UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Elaboración de un lector de libros electrónicos colaborativo para su utilización en el ámbito académico

Trabajo de graduación en modalidad de Megaproyecto presentado por Cristina María Bautista Silva, Andrés Estuardo Quan Littow, Pablo Antonio Ruiz Campos para optar al grado académico de Licenciados en Ciencia de la Computación y Tecnologías de la Información

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Elaboración de un lector de libros electrónicos colaborativo para su utilización en el ámbito académico

Trabajo de graduación en modalidad de Megaproyecto presentado por Cristina María Bautista Silva, Andrés Estuardo Quan Littow, Pablo Antonio Ruiz Campos para optar al grado académico de Licenciados en Ciencia de la Computación y Tecnologías de la Información

Guatemala,

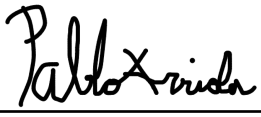
2022

Vo.Bo.:

(f) 

Ing. Pablo Ignacio Arriola Díaz

Tribunal Examinador:

(f) 

Ing. Pablo Ignacio Arriola Díaz

(f) 

Ing. Eddy Omar Castro Jauregui

(f) 

MSc. Douglas Leonel Barrios Gonzalez

Fecha de aprobación: Guatemala, 29 de diciembre de 2022.

Prefacio

A lo largo de este trabajo se utilizó *frontend* y *backend* para describir elementos clave del lector. Se utilizó *frontend* para describir la parte de un programa, software o aplicación que es vista y utilizada por un usuario, que en este caso es un estudiante. Por otro lado, se utilizó *backend* para describir la parte de un programa, software o aplicación que es utilizada indirectamente por el usuario a través del *frontend* (como por ejemplo, la base de datos).

Se prefirió el uso de la palabra *frontend* (palabra técnica) sobre *front end* (sustantivo) o *front-end* (adjetivo) para hacer referencia directamente a un *front end* o *front-end* en el campo computacional. Si bien *frontend* no se encuentra definida en el diccionario: *Oxford Dictionary of English* de Oxford Dictionaries (2010), esta palabra es comúnmente utilizada en el campo computacional y ha existido interés sobre la forma correcta de utilizarla (Lin, 2011). Se utilizó *frontend* en este trabajo para poder distinguirla claramente como una palabra que hace referencia a la computación. Lo mismo aplica para la palabra *backend*, que se prefirió sobre la palabra *back end* o *back-end* para hacer referencia a un término estrictamente computacional.

Se agradece grandemente a todos aquellos que directa o indirectamente hicieron de este trabajo una realidad.

Índice

Lista de figuras

Lista de cuadros

Resumen

El presente trabajo busca diseñar una interfaz de usuario o *frontend*, un *backend* y un módulo de ciencia de datos para un lector de publicaciones electrónicas colaborativo en el ámbito académico, mediante la metodología de *design thinking*.

En diversas instituciones educativas, los estudiantes utilizan sesiones grupales para poder realizar sus trabajos, lecturas o bien, estudiar. Bajo esta premisa, se ideó una herramienta que permitiría sesiones de estudio enfocadas por medio de un lector de publicaciones electrónicas colaborativo. Tomando como base la población del Campus Central de la Universidad del Valle de Guatemala, se puso a prueba la idea para diseñar este lector. Mediante encuestas, entrevistas y observación se logró definir las características que buscan los estudiantes en un lector de este tipo. Como resultado, se obtuvo una interfaz, un *backend* y un módulo de ciencia de datos para un lector de publicaciones electrónicas colaborativo para ser utilizado por estudiantes, de acuerdo con las necesidades identificadas en la población de estudio. El prototipo fue elaborado por medio de *Angular*, *NestJS*, *Tableau* y los principios SOLID para poder generar una base de código segura, escalable y mantenible que pueda ser tomada como base para futuros proyectos.

Se recomienda realizar una integración de este lector con la biblioteca de la UVG y el programa central de estudiantes y docentes: Canvas, para poder tener una forma accesible de estudio y que la literatura de la biblioteca pueda ser tomada en cuenta para planificar cursos.

También se observó el potencial que este lector puede tener en otras instituciones educativas y en editoriales de libros, por lo que se sugiere también explorar estos campos.

Abstract

This work seeks to develop a user interface, also known as a *frontend*, a *backend*, and a data science module for a collaborative e-reader focused on the academic sector using the design thinking methodology.

In several educational institutions, students use group sessions to accomplish tasks such as: completing coursework, reading, or studying. Under this premise, a tool that would allow group study sessions to be more focused through collaborative reading was devised. The population of students at Universidad del Valle de Guatemala's central campus was used as a reference to test the idea and design the collaborative e-book reader. Through forms, interviews, and observation it was possible to define the characteristics that students look for in such a reader. As a result, an interface was developed for a collaborative e-book reader to be used by scholars, along with a backend, and a data science module, according to the needs that were identified in the studied population. The prototype was assembled using *Angular*, *Nest.js*, *Tableau* and SOLID in order to generate a safe, secure, scalable, and maintainable code base that can be used for future projects.

It is left as a recommendation to integrate this reader with UVG's library and Canvas: the central program used by students and teachers in order to have an accessible way to study and so that the literature present in the institutional library can be taken into consideration when planning courses.

The potential for this reader in other educational institutions and book editorials was also noticed so a deeper exploration of these fields is advised.

CAPÍTULO 1

Introducción

Uno de los aspectos más importantes para el avance de una comunidad es la capacidad de aprendizaje de esta. No obstante, existen limitantes que complican el acceso al conocimiento y la fomentación de discusiones alrededor de él, especialmente si no se tiene acceso a las herramientas y tecnología adecuada (UNICEF, 2020).

Por otro lado, el trabajo grupal ha sido una herramienta comúnmente utilizada en los salones de clase. Campos como la ciencia, tecnología, ingeniería y matemática cada vez más incorporan trabajo grupal formal e informal en sus cursos (Wilson et al., 2018). Se ha demostrado que el trabajo grupal que promueve la colaboración entre estudiantes para obtener una meta de aprendizaje compartida, como un grupo de estudio, aumenta la persistencia, logros académicos y aptitudes científicas (Johnson et al., 2013; Tanner et al., 2003). Además, la colaboración, discusión y estudio grupal, ha mostrado ser una manera de facilitar el aprendizaje, especialmente aquel de carácter académico (Hammar, 2014).

Este proyecto busca atender esa necesidad: facilitar el estudio grupal y proveer una herramienta para mejorar los logros académicos de una población. Se integra la lectura de publicaciones académicas con el aspecto social, didáctico y personal de una comunidad. Para hacer esto, se realizó la interfaz o *frontend* de un lector de publicaciones electrónicas colaborativo por medio de *Angular*, un *backend* por medio de *Nest.js* y un módulo de ciencia de datos con *Tableau*; todo esto utilizando los principios SOLID de programación y la metodología de *design thinking*.

Por medio de *design thinking*, tomando en cuenta las necesidades presentes en la comunidad y formas de estudio, y pruebas alineadas a los conceptos de interacción humano-computador, se logró desarrollar la interfaz o *frontend*, así como un *backend* y un módulo de ciencia de datos para el lector. Se utilizó también SOLID para generar una base de código con buenas prácticas de programación.

El lector elaborado permite la lectura de textos, así como la realización de comentarios y resaltado de pasajes por parte de los usuarios con acceso a la publicación en el *frontend*. Además, el *backend* cuenta con la capacidad de manejar funcionalidades importantes para

un lector de esta índole, así como alimentar el módulo de ciencia de datos. Este último módulo -de ciencia de datos-, permite visualizar métricas importantes para un lector de este tipo.

Asimismo, se permite la modularidad de las bases de datos por medio de la normalización de las mismas. Se cuenta además con el guardado de la actividad de los estudiantes para asistir al módulo externo de ciencia de datos: el cual despliega métricas de interés de utilidad para un administrador de lecturas, como por ejemplo: un catedrático.

Además, se desarrollaron los perfiles de usuario de estudiante y catedrático que podrían llegar a interactuar con el lector y que podrían ser de utilidad para futuros trabajos. Esta vista a futuro se alinea con la arquitectura del proyecto, la cual respeta la filosofía *MVC*, presente en *Angular* y *Nest.js*. El uso de los principios SOLID, así como el uso de las tecnologías mencionadas anteriormente permitió desarrollar una base de código segura, escalable y mantenible que puede ser utilizada en futuros proyectos y extender el alcance de este trabajo.

CAPÍTULO 2

Justificación

Actualmente existen varios lectores electrónicos y dispositivos a través de los que se puede tener acceso a publicaciones, como lo son computadoras, tabletas y celulares. Sin embargo, al momento de que un grupo quiera estudiar un texto, usualmente se utilizan aplicaciones externas para tomar notas, hacer preguntas o estudiar en conjunto un pasaje, lo cual entorpece el proceso de aprendizaje y lo hace invisible hacia los catedráticos.

El poder integrar el proceso de estudio de los alumnos en un lector colaborativo e interactivo permite tanto al estudiante como al catedrático visualizar más a profundidad áreas de repaso y temas complicados o de interés. Por ejemplo, si un tema cuenta con muchos comentarios o si este es resaltado por muchos estudiantes, es muy posible que este sea relevante para la clase. Por el lado del estudiante, el contar con comentarios de sus compañeros dentro de grupos de estudio, le permitirá visualizar un mismo tema desde diferentes perspectivas y enriquecerá su experiencia al tener notas de varias personas.

Por lo general, los libros utilizados en las aulas de clase no tienen un aspecto social que permita comentar y resaltar textos para que toda la clase pueda visualizar los comentarios y preguntas de los demás mientras se estudia. Tampoco existe una herramienta utilizada de forma estandarizada en universidades y escuelas que permita un estudio sencillo de textos académicos con un aspecto social. Por medio de este proyecto, se espera dar el primer paso para permitir que el estudio colaborativo sea extendido a los libros también. Se espera que la herramienta permita visibilizar de mejor manera el proceso de estudio y que esta pueda mostrar puntos de interés sean de utilidad tanto para estudiantes y docentes en el futuro.

Se ha demostrado que estudiar en grupos ayuda a los estudiantes a aprender de manera más efectiva (Schoenherr, 2020). Además, se ha evidenciado la eficacia del trabajo colaborativo para promover una actitud positiva hacia algún tema de estudio (Springer et al., 1999). No obstante, también se ha identificado la importancia de que los estudiantes cuenten con una guía en el uso adecuado de grupos de estudio (Rybczynski y Schussler, 2011). Debido a que este proyecto busca fomentar el estudio grupal guiado dentro y fuera de los salones de clase mediante el lector colaborativo, se espera que se pueda fomentar un ambiente sano y efectivo de aprendizaje. Finalmente, el contar con una interfaz pulida, así como un *backend*

robusto permitirá a los estudiantes interactuar de forma fluida con el lector. Junto con el módulo de ciencia de datos se podría potenciar la lectura de un texto y fomentar discusiones académicas para enriquecer en gran manera el aprendizaje de un estudiante.

CAPÍTULO 3

Objetivos

3.1. Objetivo general

Elaborar un *frontend*, *backend* y un módulo de ciencia de datos que permita generar la base para un lector de publicaciones electrónicas colaborativo para su uso en el ámbito académico.

3.2. Objetivos específicos

- Diseñar y elaborar la interfaz de usuario para un lector de publicaciones electrónicas colaborativo para estudiantes a través de la metodología de *design thinking*.
- Identificar a los distintos usuarios y sus necesidades para un lector de publicaciones electrónicas colaborativo enfocado en el ámbito académico.
- Implementar una interfaz capaz de desplegar comentarios y resaltado de textos en publicaciones electrónicas que puedan ser vistos por los usuarios con acceso a la publicación.
- Realizar una base de código de *frontend* que pueda ser mantenida y continuada en futuros proyectos.
- Implementar un lector electrónico inspirado en elementos del mundo físico, utilizando la interacción humano-computador.
- Crear una herramienta lógica en un servidor que permita el manejo de recursos de *EPUB* de manera efectiva, teniendo en consideración todos aquellos elementos que impidan el manejo de los mismos del lado del cliente, facilitando así el procesamiento del mismo.

- Crear métodos administrativos que permitan cambios a documentos de manera dinámica, así pudiendo agregar, remover y modificar elementos como, pero no limitados a, exámenes o notas.
- Crear métodos tipo *REST* que consuman tanto *SQL* como *NoSQL* que permitan el uso del lector como bajos recursos, teniendo en cuenta que debe de tener un bajo uso de recursos de red también.
- Crear distinciones entre tipos de usuarios existentes de tal manera que sea jerárquicamente significativo y aprobado por distintos usuarios del público objetivo, tal como universidades.
- Mantener alta escalabilidad en la aplicación desarrollada, tratando de mantener un sistema que pueda llevar cambios constantes, dependiendo de lo que se pida.
- Establecer una conexión a la base de datos para el análisis de los datos.
- Analizar los tiempos de atención en las lecturas.
- Analizar las encuestas que se realizarán al final de las lecturas.
- Crear un *wordcloud* de las palabras claves de los comentarios.
- Crear un *dashboard* que muestre los hallazgos encontrados de los datos recolectados.

CAPÍTULO 4

Alcance

Este proyecto busca generar una base de un lector colaborativo de publicaciones electrónicas para su uso en el ámbito académico. En los párrafos a continuación se describen los módulos desarrollados para esta proyecto. El proyecto consta de tres módulos principales en total, siendo estos el módulo de *Design Thinking y Frontend*, el módulo de *Backend y Bases de datos* y el módulo de *Data Science*.

El módulo de *design thinking y frontend*, elaborado por Pablo Antonio Ruiz Campos, se enfoca en realizar la interfaz que será utilizada por estudiantes en el lector colaborativo. Para ello, se realizan diversas pruebas a usuarios y se profundiza en la interacción humano-computador, tomando como influencia elementos del mundo físico y adaptándolos para reflejar una interfaz de usuario amigable y comprensible. Además, se encarga de la implementación de un *frontend* capaz de mostrar las pantallas diseñadas, el cual se puede llegar a conectar con el *backend* elaborado en el módulo de *backend* y bases de datos del proyecto. Todo esto se realiza bajo la metodología de *design thinking*.

Por otro lado, el módulo de *backend y bases de datos*, elaborado por Andrés Estuardo Quan Littow, se enfoca en realizar un sistema modular de *backend* para afrontar conflictos de seguridad, de almacenaje, y de optimización constante para un sistema de lector colaborativo de estudiantes. Para ello, se hacen estudios de población con el módulo de *frontend* del presente proyecto, tratando de escoger y alcanzar un sistema que logre satisfacer las necesidades de dicha población. Se busca hacer que el sistema sea altamente escalable, tenga bajos costos, y que satisfaga las necesidades del público objetivo tanto como se pueda.

El módulo de *ciencia de datos* (*data science*), elaborado por Cristina María Bautista Silva, se enfoca en la creación de un modelo de análisis de emociones positivas, neutras o negativas. Este es aplicado a los comentarios que se añadan a la base de datos, siendo estos válidos en el idioma español. Asimismo, implementa un *dashboard* con información obtenida de uniones de datos de distintas tablas y encuestas, tratando de responder así preguntas varias de los profesores.

CAPÍTULO 5

Marco teórico

5.1. *Design thinking*

5.1.1. Definición

Design thinking es una metodología de diseño centrada en el humano y se practica en una variedad de campos como: literatura, arte, música, ciencia, ingeniería, negocios, entre otros. Es un proceso iterativo, no lineal que busca comprender a los usuarios, retar suposiciones previas, redefinir problemas y crear soluciones creativas e innovadoras, las cuales se pueden prototipar y probar (Friis y Yu, 2022).

Friis y Yu (2022) argumentan que, en esencia, *design thinking* se basa en un interés profundo en comprender a las personas para las cuales se diseña un producto o servicio y ayuda a observar y desarrollar empatía con el usuario objetivo. La empatía es algo clave para el diseño debido a que se refiere a entender los problemas y necesidades del usuario para diseñar las mejores soluciones posibles. *Design thinking* permite además cuestionar el problema, las suposiciones existentes y las implicaciones de realizar algo, lo que permite generar soluciones creativas. También es útil al momento de abordar problemas los cuales no tienen una definición clara o que son desconocidos. Por último, involucra la experimentación continua a través de dibujos, prototipos, pruebas e introducción de nuevas ideas.

5.1.2. Breve historia

Design thinking, conocido en el español como pensamiento de diseño, es una metodología de diseño que surgió en 1959 cuando John E. Arnold lo introdujo en el área de ingeniería. Posteriormente, Robert McKim lo utilizó en su libro *Experiences in Visual Thinking*, utilizado después en la Universidad de Stanford en Estados Unidos. Sin embargo, el concepto tomó impulso en 1991 cuando David M. Kelley fundó la empresa IDEO para trabajar en este concepto (ITMadrid, 2020).

5.1.3. Fases

Al ser un proceso no lineal *design thinking* permite realizar fases en paralelo, repetir fases o regresar a una fase anterior en cualquier punto del proceso. Esto lo hace una metodología muy ágil que permite la elaboración de productos pulidos e innovadores (Friis y Yu, 2022). Estas fases se pueden aplicar de forma indefinida y no necesariamente tiene un fin, debido a que se puede seguir retando aún un producto que se considere terminado para mejorarlo o incluso cambiarlo por una solución distinta.

Empatizar

Este es un paso crítico par iniciar procesos de *design thinking* debido a que se busca entender al usuario objetivo. Se busca también comprender a profundidad el problema que se está resolviendo para poder satisfacer las necesidades existentes (Babich, 2021). Una de las técnicas más comunes utilizadas en esta fase, según Babich (2021) son las entrevistas, la investigación contextual y el mapa de empatía.

Una técnica utilizada para empatizar se basa en **entrevistas**, debido a que se valora mucho que los usuarios cuenten historias y su experiencia con el problema que se busca resolver o alguno similar. Esto se hace con la finalidad de entender las necesidades del usuario objetivo para llegar a comprender cómo podría solucionarse el problema o incluso, retar suposiciones iniciales que se tenían sobre él. Existen varios tipos de entrevistas: estructuradas, semi-estructuradas y no estructuradas.

Las **entrevistas semi-estructuradas** presentan una estructura base, donde ciertas preguntas son definidas de antemano por el entrevistador. Sin embargo, este tipo de entrevista no se limita únicamente a las preguntas preparadas, ya que, de encontrar un elemento de interés durante la entrevista, el entrevistador puede indagar más. Son útiles para recopilar datos cualitativos y permiten explorar los pensamientos, las emociones y las creencias de los entrevistados (DeJonckheere y Vaughn, 2019).

Existen otras maneras de realizar entrevistas. Entre ellas se encuentran las **entrevistas empáticas**. Estas entrevistas son usualmente una conversación uno-a-uno que utilizan preguntas abiertas para descubrir necesidades. Este tipo de entrevistas ayudan al entrevistador a inquirir de forma más profunda en las historias que una entrevista tradicional (Nelsestuen y Smith, 2020).

También existen técnicas para empatizar como la **investigación contextual**. Esta técnica lleva al investigador involucrarse el ambiente físico en el cual frecuenta el usuario objetivo para observar cómo interactúan con productos existentes y por ende obtener un conocimiento más profundo sobre los problemas que estos tienen al utilizarlos (Babich, 2021). También permite observar cómo se desarrollan en su campo para poder aplicar elementos similares a la solución final de una forma creativa.

Finalmente, en la fase de empatizar se pueden utilizar **mapas de empatía**. Un mapa de empatía es una visualización que busca resumir lo que un equipo o persona conoce sobre el usuario objetivo. Este mapa describe lo que el usuario dice, piensa, hace y cómo se siente, por lo que realizarlo permite entender de mejor manera al público objetivo (Babich, 2021).

Un ejemplo de esta visualización puede ser observada en la Figura 1.

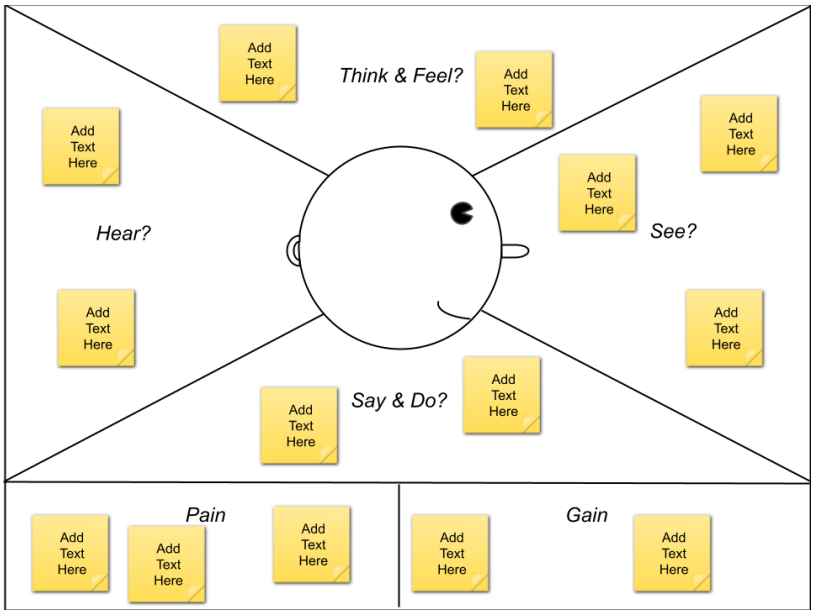


Figura 1: Mapa de empatía, tomado de las plantillas de Google

Definir

En este paso se analizan los datos que se obtuvieron de la fase de empatizar, descrita en la sección 5.1.3. Esta información se utiliza para definir el problema y sirve como apoyo en la fase de ideación, descrita en la sección 5.1.3 de este capítulo. Aquí se puede identificar qué funciones necesita el usuario final para resolver sus problemas.

Al estar utilizando una técnica centrada en el humano, el problema se debe definir de una forma que empiece con: «El usuario...». Por ejemplo, en el caso de este proyecto, un lector colaborativo para estudiantes, el problema se define de la forma: «Los estudiantes...». Esta fase puede utilizar distintas técnicas. A continuación se describen algunas.

Una de estas técnicas es el **análisis del viaje de usuario**, también conocido como *customer journey analysis* en inglés. Es una descripción visual del proceso que un usuario ejecuta a lo largo de una solución. Se consideran los pasos que este realiza, así como sus sentimientos, puntos de dolor y momentos de alegría (Babich, 2021).

También existe la técnica de **«cómo podríamos nosotros»**. Para realizarla, se deben plantear preguntas de la forma: «¿Cómo podríamos nosotros...». Esta pregunta se plantea seguida de un punto de dolor del usuario, lo cual puede permitir que un equipo piense más a profundidad sobre los problemas de este (Babich, 2021). No se limita la posibilidad técnica para implementar las soluciones durante esta fase.

Idear

En esta fase se generan ideas: es decir, se pasa de entender los problemas a explorar potenciales soluciones para estos. Las ideas que se identifican pueden ser prototipadas y probadas por usuarios que representen al público objetivo.

Esta fase puede utilizar distintas técnicas, tales como: **la peor idea posible** y **dibujos**. La peor idea posible es un método de ideación donde se busca la peor solución que se pueda encontrar para estimular el pensamiento creativo; los dibujos, por otro lado, son una forma eficiente de visualizar ideas. Estos no deben ser necesariamente de alta calidad, pero sí deben representar algo y poder expresar una idea.

Prototipar

La fase de prototipado involucra convertir las ideas generadas en la etapa de idear a elementos que pueden ser probados con usuarios reales. No deben ser completamente funcionales, pero sí deben representar la idea generada. Se recomienda iniciar con prototipos básicos e irlos refinando al contar con más retroalimentación por parte de los usuarios (Babich, 2021).

Los prototipos pueden existir como **prototipos en papel**, los cuales son generalmente hechos de forma rápida, permitiéndose validar una hipótesis con mínimo esfuerzo.

También se pueden elaborar **prototipos digitales**. Estos permiten visualizar de forma realista cómo se vería el producto final. En lugar de utilizar prototipos dibujados a papel, un prototipo digital puede utilizar herramientas digitales que representen el diseño real de los componentes a pesar de no contar con una funcionalidad completa.

Finalmente, los **prototipos programados** son utilizados para representar el funcionamiento final en una aplicación elaborada. Estos requieren de más tiempo para su elaboración y son más difíciles de cambiar en caso se descubra un punto de cambio mayor, por lo que usualmente son elaborados después de un cierto número de iteraciones con prototipos de menor calidad.

Probar

En esta fase se prueban los prototipos elaborados para identificar qué partes son efectivas y qué partes no. Para ello, se le pide al usuario o a los usuarios realizar tareas con el prototipo. Estas pruebas permiten identificar problemas con el diseño, cómo se siente el usuario y la forma en la que completan tareas (Babich, 2021).

Estas pruebas pueden ser realizadas como **pruebas de usabilidad moderada**, las cuales se realizan a personas que pertenecen al público objetivo. Se pueden realizar preguntas sobre las formas en que un usuario realiza algo dentro de la aplicación para obtener un conocimiento más profundo.

También se puede realizar **grupos de enfoque** o ***focus groups***, como se conocen en

inglés. Estos son grupos de entre 6 a 9 personas que discuten y analizan una solución. Este método es útil cuando se quiere explorar un tema en específico (Babich, 2021).

Otra forma de realizar pruebas a prototipos son las **pruebas A/B**. Estas involucran un proceso de prueba donde dos o más versiones de un elemento (como por ejemplo, el diseño de alguna pantalla) son mostradas a diferentes segmentos de los visitantes de una página para determinar cuál versión es superior (Gallo, 2017).

5.2. Aplicación

Una aplicación, es una herramienta de *software* (un conjunto de instrucciones, programas o datos) para realizar operaciones o funciones específicas. Usualmente son concebidas para facilitar ciertas tareas y facilitar el flujo de trabajo de personas. Por lo general presentan una interfaz de usuario para que este interactúe con ellas (Roman, 2020).

5.2.1. Diferencia entre programa y aplicación

El término programa se utiliza frecuentemente como sinónimo de aplicación, sin embargo, ciertos autores diferencian los términos: indicando que programa se refiere estrictamente a un conjunto de instrucciones que puede ser reconocido por una computadora para generar un resultado, mientras que una aplicación es un programa o una colección de programas que puede ser utilizada por un usuario final a través de una interfaz (Roman, 2020).

5.2.2. Aplicaciones *web*

Las aplicaciones web son tipos de aplicaciones que pueden ser accedidas a través de un navegador *web*, como por ejemplo: *Google Chrome, Firefox, Edge, Safari* (Roman, 2020). Este tipo de aplicaciones, a diferencia de un sitio *web* tradicional, permite que los usuarios interactúen con el contenido presente en el sitio, como por ejemplo: manipular formularios, comentarios, entre otros. Un ejemplo de esto es *Gmail*, que permite a los usuarios redactar correos electrónicos. Dada la naturaleza de las aplicaciones *web*, por lo general se requiere de autenticación para poder manipular los contenidos (por ejemplo, en *Gmail* no se pueden enviar correos sin haber iniciado sesión).

No se debe confundir las aplicaciones *web* con las aplicaciones móviles, que son aquellas que son diseñadas para ser descargadas en dispositivos móviles como celulares y dispositivos inteligentes (Pham, 2022).

5.3. Interfaz de usuario

Una interfaz de usuario (comúnmente encontrada como *UI* por sus siglas en inglés), es el punto en donde el usuario final interactúa con una aplicación (The Interaction Design Foundation, 2017b). Es decir, una interfaz de usuario es el punto en donde una persona se

relaciona con un dispositivo como por ejemplo, computadoras, teléfonos, tabletas electrónicas, etc. Su definición puede extenderse a la interacción entre un usuario y un producto, no necesariamente *software*, sin embargo, en este trabajo se utilizará la definición de interfaz de usuario como el punto donde una persona (el usuario) interactúa con una aplicación.

5.3.1. Elementos de una interfaz de usuario

Una interfaz de usuario se compone de 4 elementos principales: controles de entrada, elementos de navegación, elementos informativos y contenedores (UX Design Institute, 2022). La definición de estos elementos, según *UX Design Institute*, se muestra a continuación:

Controles de entrada

Son todos aquellos que permiten al usuario ingresar información. Para poder llevar esto a cabo, usualmente se tienden a utilizar botones, entradas de texto, listas u otros elementos donde el usuario pueda ingresar datos. Usualmente manejan **eventos** por medio de un lenguaje de programación para poder realizar algo con los datos ingresados.

Elementos de navegación

Estos permiten al usuario navegar la interfaz para completar una tarea deseada. Se incluyen elementos como campos de búsqueda y menús de tipo hamburguesa. La navegación puede llevar a un usuario a distintos sitios, o simplemente a otras partes de la misma página.

Elementos informativos

Dan información al usuario, como mensajes, barras de progreso y notificaciones. En ciertas ocasiones, las notificaciones pueden ser activas o pasivas, y se basan en el *UX* (o experiencia de usuario) para poder transmitir información de manera más inteligente y amigable.

Contenedores

Se utilizan para agrupar contenido en secciones significativas. Los contenedores no solo permiten que el usuario pueda visualizar la información de una manera más fácil, sino que también permiten que el programador trabaje en dichas secciones de manera más lógica y sencilla, separando la información como desee.

5.3.2. Diseño de una interfaz de usuario

El diseño de una buena interfaz de usuario es clave para que un usuario pueda comprender cómo utilizar un programa o aplicación. Un buen diseño de la interfaz de usuario, según The

Interaction Design Foundation (2017b) debe seguir los siguientes lineamientos:

1. Los botones deben comportarse de forma predecible
2. Los íconos deben estar claramente identificados y deben ser fáciles de descubrir.
3. Las interfaces deben ser simples.
4. Se debe tener un enfoque en la jerarquía y legibilidad de elementos.
5. Se debe minimizar el número de acciones para realizar tareas.
6. Los controles deben encontrarse próximos a los elementos que controlan.
7. Se debe proporcionar retroalimentación al usuario.
8. Se deben utilizar patrones de diseño apropiados donde apliquen.
9. Se debe mantener una consistencia de marca.
10. Se debe proporcionar pasos siguientes que los usuarios puedan deducir de forma natural.

El diseño de una interfaz de usuario considera cómo funcionan todos los elementos que la componen en conjunto. Este diseño es elaborado para hacerlo visualmente atractivo y fácil de navegar (UX Design Institute, 2022). El diseño, según esta misma fuente, cubre los aspectos de interactividad, diseño visual y arquitectura de información. También se pueden utilizar las heurísticas de Nielsen, 2020, que son reglas amplias para el diseño de interfaces:

1. Visibilidad del estado del sistema
2. Coincidencia entre el sistema y el mundo real
3. Control y libertad del usuario
4. Consistencia y estándares
5. Prevención de errores
6. Reconocimiento en lugar de memorización
7. Flexibilidad y eficiencia de uso
8. Diseño minimalista y estético
9. Ayuda en la recuperación de errores
10. Ayuda y documentación

Interactividad

La interactividad de una interfaz de usuario incluye cómo se comportan los elementos y las funciones de los mismos.

Diseño visual

Esto es cómo se mira la interfaz. Se consideran elementos como tipografía, imágenes, gráficas, logos, íconos y el espaciado entre elementos.

Arquitectura de información

Esto es cómo el contenido de una interfaz es organizado, es decir, cómo se distribuyen los distintos elementos de la interfaz y el por qué detrás de ello. A diferencia del diseño visual, que se enfoca en una distribución estética, la arquitectura de información se enfoca en la organización de elementos por su funcionalidad y el etiquetado de estas funcionalidades.

5.4. Interacción humano-computador

La interacción humano-computador, también conocida como *HCI* por sus siglas en inglés es un campo multidisciplinario de estudio que se enfoca en el diseño de tecnologías computacionales y la interacción entre humanos y computadoras. Si bien, inicialmente fue un campo de estudio enfocado únicamente en computadoras de escritorio tradicionales, *HCI* se ha expandido para cubrir casi todas las formas de diseño de tecnologías de la información (The Interaction Design Foundation, 2017a).

HCI es un campo multidisciplinario el cual incluye: ciencias de la computación, ciencias del comportamiento, ciencia cognitiva, ergonomía, fisiología y principios de diseño. La interacción humano-computador se enfoca en diseñar, implementar y evaluar interfaces interactivas que mejoran la experiencia de usuario en dispositivos computacionales, lo cual incluye el diseño de interfaces, diseño centrado en el usuario y diseño de experiencias de usuario (Kanade, 2022).

5.4.1. Componentes principales

Según Vijay Kanade (2022), la interacción humano-computador tiene 4 componentes principales: el usuario, la tarea, la interfaz y el contexto.

El usuario

El usuario es un individuo o grupo de individuos que participan en una tarea común. *HCI* estudia sus necesidades, metas y patrones de interacción para proveer una experiencia libre de problemas al momento de interactuar con un sistema de computación.

La tarea

Un usuario opera un sistema computacional con un objetivo en mente. Este objetivo, o tarea, debe de ser estudiado para proporcionar una mejor experiencia al usuario, tomando en cuenta los siguientes factores:

- La complejidad de la tarea.
- Las habilidades y el conocimiento necesario para interactuar con el objeto digital.
- El tiempo que lleva a cabo completar la tarea.

La interfaz

La interfaz es un punto clave de *HCI* que puede mejorar la experiencia de usuario por medio de su interacción con ella.

El contexto

Si bien *HCI* se enfoca en proveer una manera de comunicar personas con computadoras, esta disciplina también se encarga de evaluar el contexto en el cual se accede al sistema. Por ejemplo, el comportamiento visual de una aplicación bajo diferentes condiciones de luz ambiental. El contexto es ajeno al programa, pero puede tener una fuerte incidencia en cómo este debe ser diseñado para ser amigable.

5.4.2. Áreas de interés

La usabilidad y la experiencia de usuario son clave para el desarrollo utilizando *HCI*. A continuación se definen estas áreas según lo expresado por Vijay Kanade (2022).

Usabilidad

Es la clave para que diferentes usuarios puedan aprender y utilizar sistemas de computación con relativa facilidad. Un buen diseño cuenta con las siguientes características:

- Una forma sencilla de utilizarlo.
- Cuida a los usuarios de situaciones peligrosas o no deseadas.
- Es eficiente.
- Es efectivo en cumplir con sus metas.
- Es de utilidad para completar las tareas del usuario.
- Los usuarios disfrutan de utilizar el sistema.

Experiencia de usuario

La experiencia de usuario o *UX* por sus siglas en inglés, se enfoca en cómo los usuarios se sienten al interactuar con un sistema computacional. Se clasifican los patrones de interacción con una interfaz de dos maneras: características deseables y características no deseables (Kanade, 2022).

5.4.3. Acercamientos de diseño

Eberts (1994) describe en su libro *User Interface Design* 4 acercamientos para diseñar interfaces eficientes, intuitivas y amigables con el usuario: el **acercamiento antropomórfico**, el **acercamiento cognitivo**, el **acercamiento empírico** y el **acercamiento predictivo**.

Acercamiento antropomórfico

Este involucra diseñar una interfaz de usuario que posea características similares a las de un humano. Esto puede representarse en elementos como avatares o la forma de escribir mensajes del sistema, en una forma similar a la que lo haría un humano.

Enfoque cognitivo

Este considera las habilidades del cerebro humano y la percepción sensorial para desarrollar una interfaz que ayude al usuario final. Dentro de este hay 3 herramientas principales:

- Diseño a través de metáforas
- Modelos de atención y carga de trabajo
- Modelo de procesamiento de información humana (*HIP*)

Enfoque empírico

Este se utiliza principalmente para examinar y comparar la usabilidad de múltiples diseños conceptuales. Se puede realizar previo a la producción y entre sus herramientas se encuentran:

- Medidas de desempeño en tareas (como por ejemplo, evaluar si un usuario completa o no una tarea).
- Pruebas A/B

Enfoque predictivo

Este se basa en examinar componentes individuales de la experiencia de usuario en términos del tiempo que toma a un usuario completar una tarea de la forma más eficiente posible. Esto se realiza para estimar lo que podría ocurrir en el caso de que un usuario realice o no una acción.

5.4.4. Beneficios en accesibilidad

HCI permite diseñar sistemas que son accesibles, eficientes y seguros para todos. Esto implica que se diseña un sistema para ser utilizado por cualquier persona, incluyendo aquellas personas con alguna discapacidad.

Pautas de Accesibilidad para el Contenido *Web* (*WCAG*)

Las Pautas de Accesibilidad para el Contenido *Web* o *WCAG* por sus siglas en inglés, son estándares que establecen un estándar compartido para hacer el contenido *web* más accesible para personas con discapacidad (Initiative, 2018). Existen 3 niveles de criterios de éxito en *WCAG*: A, AA y AAA, siendo este último el mejor nivel. La versión más reciente, *WCAG* 2.1 fue publicada el 5 de junio de 2018 y se espera que la versión 2.2 sea finalizada en 2023. Estas pautas, según Initiative (2018) son principalmente dirigidas a:

1. Desarrolladores de contenido *web*
2. Desarrolladores de programas de autor
3. Desarrolladores de herramientas para la evaluación de la accesibilidad *web*
4. Otros que necesiten un estándar para la accesibilidad *web*, incluyendo la accesibilidad móvil

American with Disabilities Act (*ADA*)

ADA es una ley en los Estados Unidos que prohíbe la discriminación en contra de individuos con discapacidad. Esta ley se ha extendido a las páginas web. Las reglas de cumplimiento de accesibilidad de *ADA* apuntan a *WCAG*. Se recomienda cumplir con al menos los estándares AA de *WCAG* 2.1 (Essential Accessibility, 2022).

5.5. Arquitectura de un proyecto de software

La arquitectura de un proyecto de *software* es similar a pensar en la arquitectura tradicional. Es diferente elaborar un programa que funcione a elaborar un programa correctamente. La arquitectura de un proyecto de *software* se enfoca en elaborar un programa de la forma

correcta (Martin, 2017). A grandes rasgos, se podría pensar en la arquitectura de un proyecto de *software* de la misma manera en que se piensa en la arquitectura de una casa o un edificio. La arquitectura de *software* es entonces el conjunto de las estructuras fundamentales que permiten desarrollar un programa (o varios) para permitir que ciertas disciplinas se desarrollen dentro del mismo. Una frase famosa, por Robert C. Martin para describir esto, es:

> No toma mucho conocimiento y habilidad para hacer que un programa funcione correctamente. Estudiantes de secundaria lo hacen todo el tiempo. Hombres y mujeres jóvenes en universidades empiezan negocios multi-millonarios basados en solo algunas lineas de código en *PHP* o *Ruby*. Grandes cantidades de programadores *junior* en granjas de cubículos alrededor del mundo pasan por documentos de requerimientos enormes en sistemas de seguimiento de errores vastos, haciendo a sus programas "funcionar" por medio de pura fuerza bruta de voluntad. El código que estos producen, puede que no sea hermoso, pero funciona. Funciona porque, el hacer que algo funcione -una- vez no es difícil. El hacerlo correctamente es algo completamente distinto. El crear software correctamente es *difícil*.
>
> - Robert C. Martin

La buena arquitectura de un proyecto no solo es el hacer que algo funcione, sino que algo funcione y sea fácil de escalar, trabajar, entender y cambiar. Una buena arquitectura, para un proyecto, es una de las razones principales por las que un proyecto tendría altos o bajos costos de cambio.

Existen diferentes modelos de arquitectura y entre ellos se encuentran: por capas, *model-view-controller (MVC)*, *model-view-viewmodel (MVVM)*, *model-view-presenter (MVP)* y micro servicios.

5.5.1. *Frontend* y *backend*

Una de las decisiones de arquitectura fundamentales es el saber qué tipo de interacciones tendrán los distintos componentes dentro de una aplicación. Esta decisión debe enfocarse en saber si un proyecto utilizará una arquitectura definida por un *Frontend* y un *Backend*.

Frontend

El *frontend* (también conocido como desarrollo del lado del cliente o usuario) de una aplicación incluye todo lo que los usuarios finales pueden ver y operar en un programa, como por ejemplo: botones, entradas de texto, títulos, colores, etc. El desarrollo del *frontend* involucra el desarrollo de la interfaz de usuario, propiamente dicho, la forma en que los usuarios interactúan con esta interfaz y usualmente también incluye los aspectos estéticos de diseño dentro de la aplicación (Berkeley Extension, 2020).

Backend

El *backend* se enfoca en el desarrollo de elementos de un sitio o aplicación que el usuario no puede ver. También se le conoce como: «desarrollo por el lado del servidor». Esto permite realizar una aplicación interactiva. Para ello, el desarrollo backend se concentra en manejar bases de datos, desarrollar lógica, programación de APIs, arquitectura y servidores. Si bien los usuarios no interactúan directamente con el *backend*, estos interactúan de forma indirecta con él a través del *frontend*.

A diferencia de un *frontend*, el *backend* podría ser pensado como la funcionalidad que el *frontend* necesita para funcionar con cosas detrás de lo que el usuario puede ver. Tienden a ser modulares y flexibles, gracias a que necesitan saciar muchas necesidades. Los *backend* se concentran en manejar cosas como bases de datos, desarrollar lógica (a partir de las necesidades de negocio), el definir las arquitecturas de los servidores y de los proyectos de las *APIs*, etc. Estos son los que interactúan con las partes más secretas de una aplicación, por lo que deben ser seguros (Coursera, 2022).

El *backend* es usualmente utilizado para abstraer al cliente de lo que pasa en una base de datos. De igual manera, es utilizado para poder hacer operaciones más grandes que no serían ideales desde el lado del cliente. Gracias a que los *backend* son populares en la industria, existen una gran cantidad de *frameworks* que ayudan en su desarrollo.

Diferencias entre *frontend* y *backend*

La diferencia principal es que el *frontend* se encarga de desarrollar todo aquello que se corre del lado del cliente (usuario), mientras que el *backend* se encarga de todo lo que se corre del lado del servidor. Los lenguajes de programación utilizados usualmente difieren, debido a que el *frontend* debe programar también elementos de la interfaz de usuario.

Separación de responsabilidades

La separación de responsabilidades tiene que ver mucho con la programación escalable. Esta se basa en la división de un programa en muchas secciones con una responsabilidad distinta y única. La responsabilidad puede llegar a ser cualquier cosa, como *Manejar información general de un usuario en la base de datos*. Este tipo de programación permite la creación de sistemas por capas, tales como son enseñados en las arquitecturas de *software*. Las distintas secciones de un programa terminan siendo menos complejas, permitiendo que funciones de mayor grado solo las utilicen y sean más fáciles de entender. Asimismo, se evita la duplicidad de código. A grandes rasgos, se podría pensar que es una estrategia de tipo **dividir y conquistar** que lleva al diseño modular de una aplicación (Laplante y Kassab, 2022).

Principios *SOLID*

Los principios *SOLID* son cinco principios definidos por *Robert C. Martin* para ayudar a mejorar el diseño de programas orientados a objetos. *SOLID* es un acrónimo mnemónico con prácticas para poder realizar código orientado a objetos que sea más entendible, flexible y mantenible. Las referencias aquí escritas pueden ser encontradas en el libro de *Clean Architecture* por Robert C. Martin (2017).

Los cinco principios son:

1. ***Single Responsability Principle***, el cual describe que una clase solo debe de tener una sola razón para cambiar (es decir, una clase solo debería de tener una responsabilidad).
2. ***Open-Closed Principle***, el cual describe que los objetos deberían de permitir su extensión, pero no deberían de permitir su modificación. Esto básicamente argumenta que el uso de las clases abstractas debería de ser común en todo lo que es herencia. Esto significa que las entidades de *software* deben poder extenderse pero no modificarse.
3. ***Liskov Substitution Principle***, el cual describe que, dada una clase A que hereda de B, en cualquier caso donde B sea requerido, A puede ser usado.
4. ***Interface Segregation Principle***, el cual define que un cliente no debería de implementar interfaces que no utilizan, mucho menos los métodos que no utilizan. Esto aumenta la modularidad de un sistema por medio de interfaces diferentes entre sí.
5. ***Dependency Inversion Principle***, el cual define que las entidades deberían siempre de depender de abstracciones, no implementaciones de algo. Un módulo de alto nivel no debe de depender nunca de la implementación de un módulo de bajo nivel, sino de sus interfaces.

(Martin, 2017)

5.5.2. Modelos de arquitectura

Para que un *software* sea bueno, este debe de seguir ciertas reglas y guías, aumentando así su propia escalabilidad. La arquitectura de un *software* es, por definición, la organización de un *software* o un sistema. Estas estructuras fundamentales definen las relaciones entre los distintos elementos que forman parte del sistema. Esto se encuentra descrito en la sección 5.5. Sin embargo, existen varios tipos de modelos que buscan satisfacer las distintas necesidades a partir de distintos tipos de filosofías.

Por capas

Este tipo de programación es un término que envuelve a otros términos. La programación por capas es un tipo de programación que permite la separación de clases en conjuntos similares para poder manejarlas por medio de responsabilidades únicas (Buschmann et al.,

1996). Esta arquitectura se toma como base para explicar las que se presentan en las próximas secciones.

Model-View-Controller (*MVC*)

MVC es descrito como un patrón de diseño de software utilizado para la implementación de interfaces de usuario, de información, y de control de lógica. Enfatiza la separación entre la implementación de la lógica de negocio y lo que se muestra. Tiene mucho que ver con la separación de responsabilidades, descrito en la sección 5.5.1, aumentando así la escalabilidad (MDN, 2022).

Las arquitecturas *MVC* son modulares y escalables, lo cual permite contar con grandes partes de código que no es mostrado a un usuario final. Gracias a su modularidad, tienen costos de cambio relativamente bajos, lo cual hace a las arquitecturas *MVC* populares.

Model-View-ViewModel (*MVVM*)

MVVM es una versión de arquitectura de software parecida a *MVC*. Este fue desarrollado en *Microsoft* para poder manejar interfaces manejadas por eventos en el *frontend*. A grandes rasgos, este tipo de arquitectura implementa un Modelo, una Vista, un Modelo-Vista, y un *Binder*. Su función es remover todo tipo de código de la interfaz gráfica de la capa de Vista (Microsoft, 2005).

Model-View-Presenter (*MVP*)

MVP es una variación de *MVC*. Se utiliza comúnmente para elaborar interfaces de usuario y la diferencia principal con *MVC* es que no utiliza controladores, sino presentadores, los cuales actúan como un tipo de negociador intermedio que trabaja toda la lógica de presentación (Potel, 1995).

Micro servicios

Los micro servicios son un tipo de arquitectura que sigue la filosofía de composición de un programa. Esta trata de hacer pensar que un programa no es nada más que una composición de muchos servicios de pequeña escala (*micro servicios*) que pueden ser extraídos y que están débilmente acoplados para que puedan ser cambiados (Richardson, 2021). Entre sus mayores ventajas, se encuentra que son:

- Fáciles de mantener y de probar
- Débilmente acoplados
- Desplegables de forma independiente
- Organizados alrededor de las necesidades del negocio

- Mantenidos por un equipo pequeño

Los *micro servicios* han permitido a los contenedores, como *docker*, ganar popularidad ya que son fragmentos de un programa más grande que pueden correr solos. De igual manera, evitan que un programa sea monolítico y difícil de cambiar.

5.6. *Framework*

En la computación, un *Framework* (también conicidos como *Software Frameworks*) son marcos de gestión de *software* que sirven como abstracciones de funciones básicas y genéricas de un programa. Es decir, simplifican ciertos componentes para facilitar el proceso de desarrollo. Son plataformas que proveen bases para implementar distintos tipos de aplicaciones, que por lo general, resulta un desarrollo más rápido y conciso, al enfocar los esfuerzos del programador en lugar de tener que manejar una base compleja de elementos no abstraída (Riehe y Gross, 1998). Como es de costumbre, esto significa que muchas de las funciones están abstraídas, a grandes rasgos, de los programadores desarrollando el programa.

Muchos de los problemas modernos con el diseño y la integración de *Frameworks* tiene que ver con el diseño de programas a base de *clases*. Las clases serán mencionadas más adelante en la sección 5.8, Paradigmas de programación, en la página 27. Aunque el uso de clases es bastante bueno para permitir que el humano entienda lo que se está diseñando (gracias a que tendemos a reconocer distintas partes de la realidad como objetos singulares discretos), es difícil hacer que los objetos interactúen entre sí de manera efectiva (Riehe y Gross, 1998).

5.6.1. *Angular*

Angular es un marco de trabajo desarrollado por *Google* basado en componentes para desarrollar el *frontend* de una aplicación. Este se encuentra construido sobre *TypeScript* (Google, 2022). *Angular* incluye:

- Un marco de trabajo basado en componentes para construir aplicaciones *web* escalables.
- Una colección de librerías para manejar rutas, campos, comunicación cliente-servidor, entre otros.
- Herramientas de desarrollador para desarrollar, construir, probar y actualizar código.

Angular permite elaborar una estructura modular de proyectos (Maaoui, 2021). Según Maaoui, 2021 las ventajas principales de *Angular* son:

- Tiene una buena reusabilidad de código.

- La estructura por componentes permite una alta reusabilidad y se facilita el uso de componentes en la interfaz de usuario.
- Las pruebas unitarias se simplifican.
- La legibilidad del código es buena.
- Es de fácil mantenimiento.

Modelo *MVC* de *Angular*

Angular utiliza un patrón de arquitectura *MVC* (Figura 2) sencillo de implementar de forma adecuada (Praphamontripong, 2022), lo que permite al desarrollador mayor libertad de programar: en lugar de tener que dividir la aplicación en diferentes componentes de *MVC* y construir el código para unir estas partes, *Angular* únicamente requiere que el desarrollador divida la aplicación. Esto permite un desarrollo más enfocado y elimina la necesidad de generar código adicional. Las directivas (especificaciones sobre cómo un compilador u otro traductor debe procesar las entradas) pueden ser manejadas por otro equipo. Todo esto, en conjunto, permite la creación de aplicaciones más rápidas, con requerimientos de memoria en disco menores. Algunos consideran que la implementación de arquitectura de *Angular* combina *MVC* con *MVVM*, lo cual hace más sencillo abstraer la interfaz gráfica (Maaoui, 2021).

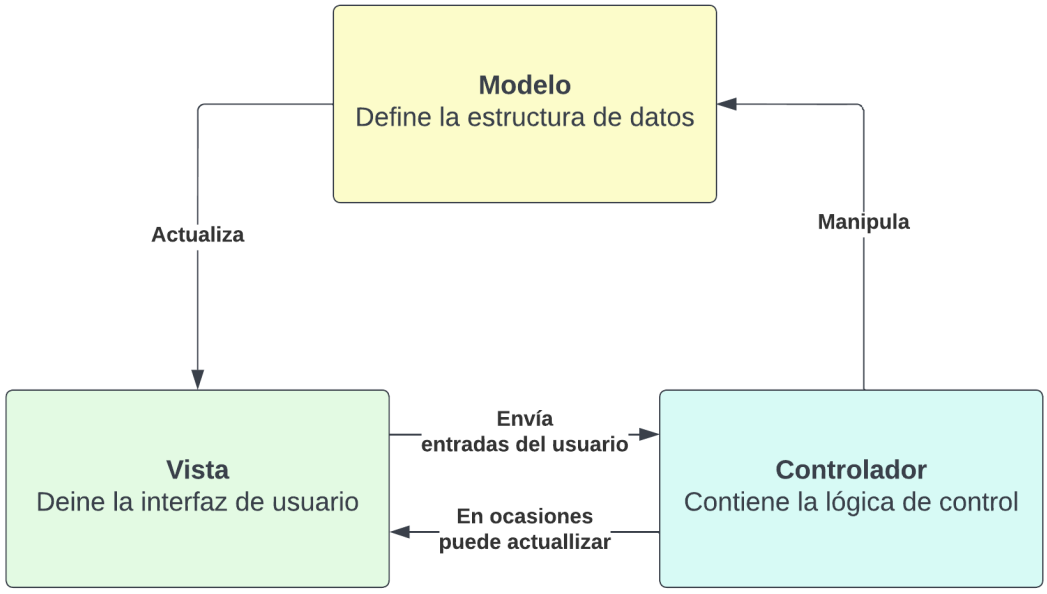


Figura 2: Arquitectura *MVC*

Otras ventajas de *Angular* son: permite enlazamiento de dos direcciones de datos para reflejar cambios en la vista y en la aplicación; se facilita la manipulación del objeto (*HTML*) y el rendimiento en el servidor es bueno (Gazta, 2021). No obstante, también existen desventajas, tales como: un menor rendimiento en dispositivos antiguos; una mayor curva de

aprendizaje y capacidades limitadas de optimización en buscadores (*SEO*), sin embargo, existen librerías que pueden ayudar con este último (Kyslova, 2021).

TypeScript

Angular utiliza *TypeScript*, que es un lenguaje fuertemente tipado que se construyó sobre *JavaScript*. Esto permite una mayor seguridad al soportar tipos (tanto primitivos como interfaces). Permite interceptar y eliminar errores de forma temprana en el desarrollo o mientras se realizan tareas de mantenimiento.

Estructura modular

Angular organiza el código en *buckets* (conocidos como módulos dentro de *Angular*). Esta estructura modular hace que sea más sencillo organizar la estructura del proyecto y permite la reutilización de código. En casos como este proyecto, la modularidad de código es ideal para poder adaptarlo a distintas necesidades o incluso, para la elaboración de futuros lectores que solo requieran parte de la funcionalidad desarrollada.

Soporte a largo plazo

Angular también fue seleccionado debido a que es soportado a largo plazo por *Google* y tiene amplia documentación. El contar con amplia documentación podría permitir a nuevos desarrolladores comprender la lógica del *frontend*. Esto, combinado con el soporte a largo plazo por parte de *Google* fomentan un ambiente ideal para un proyecto que pueda ser desarrollado y mejorado a lo largo del tiempo por medio de varias iteraciones.

Prettier

Prettier es una herramienta de formato de código con soporte para *TypeScript* y que puede ser utilizado en conjunto con *Angular*. Esta herramienta se asegura de que el código desarrollado cumpla con un estilo consistente (Prettier, s.f.).

ESLint

ESLint es una herramienta que busca y encuentra problemas en código de *JavaScript* (ESLint, s.f.). Debido a que *TypeScript* es construido a partir de *JavaScript*, *ESLint* puede ser utilizado con *TypeScript* también. *ESLint* presenta configuraciones para alinearse mejor con el estilo y estándares de la comunidad de *Angular*. La versión actual de *Angular* utiliza *ESLint*, como su herramienta principal de análisis de código o *linting* (Spickerman, 2022).

Socket.IO

Socket.IO es una plataforma escalable que permite abrir *sockets* (un punto para comunicación en dos vías) para obtener datos en tiempo real. Esta plataforma utiliza *WebSocket* para reducir la latencia entre cliente y servidor. También es escalable a múltiples servidores (Socket.IO, s.f.). Este puede ser integrado con *Angular*.

5.6.2. *Nest.js*

Nest.js es un marco de trabajo para el *backend*. Este se puede utilizar con *Express* y es el marco de trabajo implementado en el módulo de *backend* de este proyecto.

5.7. Pilas de desarrollo

Una pila de desarrollo o simplemente un *stack* (o *stack* de *software*) es un conjunto de tecnologías que se utilizan para desarrollar una aplicación. En el caso de *Angular*, la pila más común utiliza: *MongoDB* (base de datos), *Express* (*backend*), *Angular* (*frontend*) y *Node.js* (*runtime*). Esta pila se le conoce como *MEAN*, por las siglas de sus componentes. También existe la pila *MERN*, que utiliza *React* en lugar de *Angular* (Gadhavi, 2022). La combinación de tecnologías usualmente permiten que el desarrollo del software se vuelva rápido y con soporte nativo.

5.8. Paradigmas de programación

5.8.1. Paradigmas

Un paradigma, por su definición general, es un ejemplo que sirve como un patrón o un modelo. Sin embargo, en la programación, un paradigma puede servir como una escuela de pensamiento para el diseño de programas de una computadora (Nørmark, 2013). A grandes rasgos, existen cuatro tipos de programación populares, habiendo estando evolucionando constantemente con la computación de por sí:

- Programación imperativa
- Programación funcional
- Programación lógica
- Programación orientada a objetos

En este documento, por alcance, solo se estarán explicando tres: la programación imperativa, la programación funcional y la programación orientada a objetos.

Programación imperativa

La programación imperativa es el paradigma de programación más viejo que se basa en dar instrucciones con secuencia clara a una computadora (IONOS, 2020). Los lenguajes imperativos son lenguajes muy específicos con controles de flujo dentro de ellos para poder ejecutar programas con muchos posibles resultados. Un ejemplo de este tipo de lenguaje es ***C++***.

Programación funcional

La programación funcional es un tipo de programación nacida de disciplinas matemáticas (en específico, de la teoría de funciones) que describe el funcionamiento de un programa por medio de la composición de funciones. Por esto, lenguajes de programación que utilizan esta filosofía tienden a ser mucho más simples de manejar (Nørmark, 2013). Un ejemplo de este tipo de lenguaje es ***C***. Este lenguaje tiene un punto de entrada en la función *main*, haciendo que esta sea compuesta de todas las demás funciones necesarias para el programa.

La entrada a un programa de un lenguaje que siga este paradigma puede verse así:

```
// main.c

int main(int argc, char* argv[]) {
    return 0;
}
```

Como se puede apreciar, la programación funcional puede empezar directamente por una función de por sí, como pasa con el lenguaje de *C*.

Programación orientada a objetos

La Programación Orientada a Objetos es un paradigma de la programación que permite ver a diferentes componentes dentro de un programa como objetos discretos que manejan su propio estado. Es una manera de programar que permite al programador visualizar todo un programa como un conjunto de operaciones o interacciones entre objetos de varios tipos. El Dr. Alan Kay, un computólogo con antecedentes en biología y matemáticas, lo describió como vez primera como un conjunto de objetos que asemejaban a células biológicas, o a computadoras singulares en una red que solo se podían comunicar por medio de mensajes (Kay, 2003). La programación orientada a objetos es mucho más fácil de pensar cuando se tratan de conocer solo los objetos de por sí, gracias a que la vida real funciona de esta misma manera según el humano: los objetos que existen son discretos (para este ejemplo) y funcionan por medio de interacciones entre sí. Sin embargo, el mundo real, si es descrito como solo eventos, puede ser descrito mejor como Programación Funcional, descrita en la sección 5.8.1.

La programación orientada a objetos es un paradigma de programación muy popular con muchos lenguajes modernos. El más conocido, no tan moderno, siendo *Java*, que funciona

utilizando una clase *Main* como punto de entrada. Un ejemplo de un programa que sigue este paradigma:

```
// Main.java
public class Main {
    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}
```

Es importante notar que cada paradigma tiene sus propias ventajas y desventajas. Mientras que algunos considerarían a la programación funcional como más rápida, su complejidad puede aumentar a tal punto que ya no es sostenible. Asimismo, mientras que la programación orientada a objetos puede ser más fácil de entender para el humano, la verdad es que puede lograr ser costosa en algunos casos. A final de cuentas, muchos programadores han tomado esto ya como un tema de preferencias.

5.9. *GitHub*

GitHub es un servicio de control de versiones que utiliza *Git*. Permite trabajar con otras personas en proyectos (usualmente de *software*) desde cualquier parte. También, *GitHub* permite la integración de varios elementos como *bots* de seguridad, generación de *builds* de forma automatizada, creación de ramas y protección de ramas. Es muy popular entre empresas de desarrollo bien establecidas.

Github también cuenta con *GitHub Issues*, lo cual permite una funcionalidad de *backlog* básica que se integra de forma nativa con los repositorios, lo cual lo hace ideal para manejar un equipo reducido.

5.10. *ePUB*

Un *EPUB*, también conocido como *ePub* o *epub* es un tipo de formato popular que sirve para el almacenaje de publicaciones electrónicas. Son formatos flexibles. Muchas personas pueden confundirlos con los *PDF*, gracias a que ambos despliegan contenido electrónico que puede ser leído. Estos se diferencian en que, mientras un *PDF* está hecho para ser estático e igual en todos los lugares, un *EPUB* puede cambiar todas las características de su texto (incluidos el tamaño, el tipo, el color, etc). Existen herramientas que permiten la traducción de *PDF* a *EPUB*, lo cual hace que la lectura de estos sea más sencilla. Debido a la compresión presente en los archivos *EPUB*, estos cuentan con un tamaño reducido a comparación de otros formatos como *PDF* (Wisley, 2021).

5.11. Daltonismo

Esta es una condición visual cuando una persona no es capaz de ver los colores en una forma normal. También se le conoce como ceguera al color (Turbert, 2022). Existen diversos tipos de daltonismo. Los principales son:

- Protanopia: no se percibe el color rojo.
- Deuteranopia: no se percibe el color verde.
- Tritanopia: no se percibe el color azul.
- Acromatopsia: no se perciben colores (solo blanco y negro).
- Monocromía del cono azul: se tiene una discriminación baja del color.

También existen anomalías de visión que presentan en menor grado los síntomas de daltonismo.

5.12. REST

Una aplicación *REST* es un tipo de aplicación diseñada alrededor del estilo de arquitectura que permite la interacción entre servicios a partir de una interfaz bien conocida y fácil de entender («What is a REST API?», s.f.). A grandes rasgos *REST*, significando *Representational State Transfer*, es una de las formas más populares de transmitir información entre servicios web. Este tipo de *API* tiende a poder ser representada por medio de *JSON*, *HTML*, *XLT*, *Python*, *PHP* o texto plano.

Una respuesta de *REST* es de tipo *HTTP* y puede contener varias partes: Encabezados y parámetros, *Uniform Resource Identifier*, *Caching*, *Cookies* y más. Para que una *API* pueda ser considerada como una *RESTful API*, esta tiene que tener las siguientes características:

- Una arquitectura de cliente-servidor que esté formada por clientes, servidores y recursos, con peticiones manejadas por medio de *HTTP*.
- Comunicación cliente-servidor que no tenga estado. Esto significa que la información del cliente nunca es guardada entre peticiones, por lo que cada una de estas es separada y no están conectadas.
- La información puede tener la capacidad de ser guardada en *caché*, facilitando la interacción entre clientes y servidores.
- Una interfaz uniforme entre componentes para que la información pueda ser transmitida de manera estándar. Por esto, se requiere que:
    - Los recursos pedidos son identificables y separados de las representaciones enviadas al cliente.

    - Los recursos pueden ser manipulados por un cliente mediante la representación que este recibe porque la representación contiene la suficiente información para hacerlo.
    - Los mensajes retornados al cliente son auto-descriptivos y tienen la suficiente información para describir cómo es que el cliente la debe de procesar.
    - El uso de *hypertext* o *hypermedia* está disponible, significando que el cliente puede usar *hyperlinks* para encontrar todas las demás acciones disponibles después de haber accedido al recurso.
- Un sistema por capas que organiza cada tipo de servidor (como por ejemplo, aquellos que son responsables por la seguridad, el manejo de balanceos de carga, etc) que tomen un rol para la toma de la información accedida, organizándolo todo en jerarquías, invisibles para el cliente.
- La opción de mandar código a demanda. Esta característica es **opcional**, pero popular.

(«What is a REST API?», s.f.) *REST* compite principalmente contra el protocolo de *SOAP*. Sin embargo, este es tomado como más fácil de utilizar y de implementar, incluso si un poco más inseguro. Asimismo, *REST* es mucho más liviano que *SOAP*.

5.12.1. Stateless

El no existir con estado, para una *API Web*, es el existir sin ningún tipo de estado guardado entre peticiones en un servicio web. Es decir, cada petición sucede en total aislamiento, sin guardar lo que pasó en otras peticiones. Más que todo, esta filosofía es argumentada como fácil de escalar y con mucha menos complejidad que otros tipos de arquitectura. Asimismo, gracias a que los datos no tienen que ser guardados entre peticiones, existe mucho mejor rendimiento. Sin embargo, esto también significa que el tamaño de las peticiones es mucho mayor que en otras arquitecturas (Singh y Tomar, 2016).

5.13. Bases de datos

Una base de datos es una colección de información estructurada guardada (típicamente) en un sistema electrónico en un sistema de cómputo. Estos son manejados por medio de un sistema de manejo de Bases de Datos *DBMS* para poder proporcionar acceso a la información necesitada por las aplicaciones asociadas a estas. Las bases de datos existen en dos formas muy populares: las *SQL* y las *NoSQL* (Oracle, s.f.). A diferencia de archivos convencionales de texto, las bases de datos tienen el control de:

- Cómo la información es guardada y manipulada.
- Quién tiene acceso a la información guardada.
- Cuánta información puede ser guardada.

(Oracle, s.f.)

5.13.1. SQL

Las bases de datos *SQL* (*Structured Query Language*) son un tipo de base de datos relacionales que permiten el control sobre fragmentos de información relacionados entre sí. *SQL* hace referencia al lenguaje utilizado para poder llevar a cabo las distintas operaciones dentro de una base de datos. Este tipo de base de datos puede ser encontrada en la forma de:

- Bases de Datos Relacionales
- Bases de Datos Orientadas a Objetos
- Bases de Datos Distribuidas
- *Data Warehouses*

Este tipo de bases de datos tiene la ventaja que los datos pueden tener relaciones entre sí desde el diseño, junto con un tipo de dato predefinido. Esto no solo ayuda a que se puedan guardar muchos tipos de datos, sino también a la normalización.

Las bases de datos *SQL* modernas tienen soporte para formatos *JSON*, por lo que es posible simular, de cierta manera, a una base de datos *NoSQL* (sección 5.13.2) dentro de sí, pero esto quita el control a la base de datos y se lo da al programador.

Seguridad y *SQL injection*

A grandes rasgos, las bases de datos no contienen muchos sistemas de seguridad implementados dentro de ellas, por lo que recae en el programador y al ambiente a protegerlas. Gracias a que las bases de datos se basan en un lenguaje de *queries* (*SQL*), es posible hacer ataques por medio de una inyección de datos. Un ataque muy popular es el ataque de *SQL Injection*, el cual consiste en la inserción de un *query* de *SQL* por medio de una entrada de información desde el cliente a la aplicación. Si uno de estos ataques es exitoso, virtualmente toda la base de datos es considerada comprometida, ya que todos los poderes encima de esta están ahora disponibles (OWASP, s.f.).

Ciertos patrones de diseño pueden mitigar este tipo de ataque, tal como el patrón de diseño de un repositorio. En vez del diseño a partir de *queries*, se manejan clases y modelos dentro de la base de datos, facilitando enormemente todo lo que tiene que ver con seguridad (Matsutomo, 2021).

5.13.2. NoSQL

Por el otro lado, mientras que las bases de datos *SQL* es un tipo de base de datos estructurada, *NoSQL* es completamente lo contrario. Este tipo de base de datos, que significa *Not Only SQL*, es un tipo de base de datos no relacional popular con los sistemas web. Su nacimiento provino del hecho que el guardar cosas (es decir, el costo por *MB*) ya no era tan alto. Por esto, los programadores eran las partes más costosas de un desarrollo. El desarrollar

ID	first_name	last_name	cell	city
1	Leslie	Yepp	8125552344	Pawnee

Cuadro 1: Tabla *SQL* de *User*. Ejemplo del sitio web de *MongoDB*.

bases de datos complicadas, tales como las estructuradas, era mucho más costoso (tanto en su diseño como en su consumo), por lo que se crearon las bases de datos *NoSQL* que no tenían que tener ningún tipo de relación entre **documentos** (MongoDB, s.f.). Gracias a su diseño, las *NoSQL* tienen las siguientes ventajas:

- Esquemas extremadamente flexibles
- Fáciles de escalar horizontalmente
- *Queries* extremadamente rápidos
- Facilidad de uso para los programadores

Claro, al igual que las bases de datos *SQL*, existen varios tipos de bases de datos *NoSQL*:

- Bases de Datos de Documentos
- Bases de Datos por Llaves
- Bases de Datos de columnas anchas
- Bases de Datos de Grafos

La mayor diferencia entre los dos tipos de bases de datos tienen que ver con cómo está guardada la información (MongoDB, s.f.).

Sin embargo, es importante mencionar que, aunque los datos no requieren tener relación entre sí, **las bases de datos *NoSQL* tienen soporte para relacionar datos, solo de manera distinta.**

Uso de documentos

Una base de datos *NoSQL*, como *MongoDB*, usualmente utiliza **documentos** en vez de tablas. Un documento es un fragmento de memoria en formato *JSON*. El sitio web de *MongoDB* da el siguiente ejemplo:

> Digamos que se quiere tener la información de un usuario y sus *hobbies*. Para esto, debemos de guardar, primero, los datos de un usuario más básicos. Demostrado en el Cuadro 1.
>
> A esta tabla se le tiene que hacer otra tabla para relacionarla con la tabla inicial, permitiendo relacionar los *hobbies* de *Leslie Yepp* con su perfil. Esto se puede apreciar en el Cuadro 2. Para poder obtener toda la información de un usuario

ID	user_id	hobby
10	1	*scrapbooking*
11	1	*eating waffles*
12	1	*working*

Cuadro 2: Tabla *SQL* de *Hobbies*. Ejemplo del sitio web de *MongoDB* para denotar los *hobbies* de alguien en la tabla de *User*.

y sus pasatiempos, la información de la tabla de los usuarios debe de ser unida con la información de la tabla de sus pasatiempos. Esto lo trabaja *MongoDB* de una manera distinta:

```
// Ejemplo tomado de MongoDB
{
    "_id": 1,
    "first_name": "Leslie",
    "last_name": "Yepp",
    "cell": "8125552344",
    "city": "Pawnee",
    "hobbies": ["scrapbooking", "eating waffles", "working"]
}
```

Con *MongoDB*, las relaciones de algún objeto y sus propiedades existen en un solo lugar, y pueden ser accedidas desde la base de datos. No se requieren hacer uniones, permitiendo respuestas más rápidas.

Casos de uso

NoSQL se encuentra en casi cualquier industria. Sus casos de uso se basan en las siguientes necesidades:

- Desarrollo ágil y rápido
- Información estructurada o semi-estructurada
- Altos volúmenes de información
- Requerimientos de escalabilidad
- Uso de micro servicios y *streaming*

5.14. Estructuras y formatos prominentes

5.14.1. JSON

JSON es un tipo de formato para el intercambio de información (json, s.f.). Su uso fue descrito por primera vez en el estándar de *Javascript*, obvio por sus siglas *JavaScript Object*

Notation, en 1999. Un *JSON* es conformado de dos partes principales: un nombre y un valor (que puede ser otro *JSON* de por sí). Cada nombre es separado por dos puntos de su valor. Cada par está separado por comas de otros pares.

La mayor ventaja del *JSON* es que tiene un manejo muy sencillo. Es fácil de entender y es infinitamente extensible. Por esto, la comunicación que este proporciona es bastante completa y sencilla (json, s.f.).

5.15. Seguridad de información

La seguridad de la información, apodada como *InfoSec*, se refiere al conjunto de procesos y herramientas diseñadas y desplegadas para proteger información sensible de clientes. Esto hace referencia a la protección de la modificación, disrupción, destrucción e inspección no autorizada de la información de un cliente («What Is Information Security?», 2022).

Este tipo de seguridad tiene varios tipos:

- **Seguridad de aplicaciones**, que hace referencia a la seguridad y protección de las vulnerabilidades de un software en el ámbito del internet y aplicaciones móviles. También hace referencia a la protección de autenticación y autorización de los usuarios, integridad del código y sus configuraciones, y de políticas y procesos.
- **Seguridad de nube**, que hace referencia a la construcción de aplicaciones seguras, junto con su almacenamiento. Hace referencia también a cómo estas están protegidas del consumo de aplicaciones de terceros, ya que el correr en la nube es correr en un espacio compartido.
- **Criptografía**, el cual hace referencia a cómo la información es cifrada cuando esta está en tránsito o almacenada, teniendo en cuenta su confidencialidad y su integridad.
- **Seguridad de infraestructura**, el cual hace referencia a las protecciones físicas internas y externas de redes, laboratorios, centros de datos, servidores, computadores y dispositivos móviles.
- **Respuestas a incidentes**, el cual estudia y describe cómo se debería de actuar en respuesta a posibles incidentes.
- **Manejo de vulnerabilidades**, el cual describe cómo se deberían de manejar los puntos débiles en la infraestructura de un proyecto, teniendo en cuenta su ámbito y cómo es este accedido.

5.15.1. Autenticación

La autenticación es un proceso que verifica que alguien o algo es quien dice ser. En los sistemas modernos, usualmente existen maneras de usar una forma de autenticación para asegurar el acceso a alguna aplicación y su información (OneIdentity, s.f.).

Un ejemplo de un fallo de autenticación es cuando una entidad trata de hacer una petición a un servicio, sin haber hecho *login* con anterioridad. El servicio no sabe quién es, y lo rechaza. El manejo de sesiones con entidades de *JWT* es popular con la autenticación.

5.15.2. Autorización

La autorización es el proceso seguro que determina el nivel de acceso de un usuario o servicio. En la tecnología, se utiliza la autorización para dar usuarios o servicios los permisos para acceder información o llevar a cabo ciertas acciones (OneIdentity, s.f.). En el ejemplo anterior, si un usuario trata de acceder a todos los recursos de una aplicación, pero no tiene el rol permitido (incluso si sí está dentro de la aplicación), entonces a este se le niega el acceso y se le envía una respuesta de error.

La autorización es común de ver con lo que son las jerarquías de usuarios por medio de roles. Una jerarquía define permisos y qué es lo que puede hacer un usuario.

JWT

Los *JWT*, iniciales para *Json Web Token*, son parte del *Open Standard RFC 7519* que define una manera compacta y auto-contenida de transmitir información entre dos clientes como un objeto de tipo *JSON*. Aunque estos fueron mencionados en la sección 5.15.1, estos tienen que ver mucho más con la autorización. Lo que tienen de especial estos tokens es que pueden ser firmados de manera digital utilizando un «secreto». Esta «firma» es un conjunto (*set*) de caracteres generados por una función especial. Si la información dentro del *JWT* se pone dentro de esta misma función, y el *output* de la función no es la misma que la firma, entonces se puede decir que el *JWT* es inválido. Por ende, se pueden guardar cosas como los usuarios dentro de un *JWT* (sin tratar de revelar información sensible).

5.16. Generación de información

La generación de información es el proceso de generación de información falsa, pero creíble, para poder ser utilizada en entornos en donde no existe la información suficiente para llegar a conclusiones. La información generada debe de tener las siguientes características:

- **Debe de ser creíble**, ya que la información debe de ser un reemplazo directo para la información real
- **Debe de ser indistinguible de la información real**, ya que la información generada debe de seguir los mismos patrones que la información real.
- **Debe de poder ser generada en masa**, ya que esta debe de ser lo suficientemente genérica para ser generada en grandes cantidades al mismo tiempo, pero específica como para ser reconocible en su propio entorno.
- **Debe de ser significativa**, ya que la generación de data “basura” no tiene ningún sentido

- **Debe de poder ser reemplazada por la real**, como si esta hubiera sido la real desde un principio.

La generación de información es utilizada para poder hacer pruebas de concepto en varias aplicaciones en desarrollo. La generación de por sí debe de ser cuidada, ya que se pueden llegar a hacer patrones no creíbles o engañosos.

5.17. Servicios de nube

Los servicios de la nube son descritos como la infraestructura, las plataformas o el software que se mantiene en los servidores o servicios de un tercero, haciendo que estén disponibles para otros usuarios en el internet (Redhat, 2022). Estos permiten que el flujo de la información sea mucho más sencilla para clientes de varios entornos, tales como aquellos que existen solo con un *frontend* (Redhat, 2022).

5.17.1. *AWS*

AWS es una plataforma de la nube de *Amazon* que permite la creación y utilización de distintos servicios, ofreciendo más de 200 servicios a nivel global. Se toma como uno de los líderes en estos temas, ya que tienen una gran capacidad para ejecutar todo aquello que necesiten, y han completado una gran parte de su visión (Amazon, s.f.-c).

S3

Amazon permite el guardar objetos en la nube por medio de varios servicios. El servicio más popular para guardar objetos es *S3*. Este servicio tiene las ventajas de ser altamente escalable, extremadamente eficaz y escalable. Además, es fácil de utilizar, permitiendo que la lógica recaiga sobre el programador (Amazon, s.f.-b).

EC2

Amazon también ofrece lo que son las *EC2*, que son instancias de *Elastic Computing*. Estas proveen capacidad de almacenamiento y ejecución de programas. A grandes rasgos, las instancias existen como imágenes de sistemas operativos remotos que pueden ser utilizados en conjunción con sistemas de direccionamiento del internet para funcionar como *APIs*. Estas son tomadas como muy seguras, gracias a los sistemas de cifrado que *Amazon* posee. Estas pueden ser configuradas para existir solamente en ciertas subredes, o como se desee (Amazon, s.f.-a).

5.18. Análisis de emociones

5.18.1. ¿Qué es el análisis de emociones?

El análisis de emociones, se produce a través del uso de un conjunto de disciplinas como procesamiento de lenguaje natural *NLP*, lingüística computacional, además de análisis de texto, este conjunto lo que permite es identificar y extraer la información necesaria. El objetivo del análisis de emociones es la predicción de la clasificación, de un texto o múltiples textos, del emoción, esto se logra al predecir la polaridad de los texto, como si estos son positivos, negativos o neutros (BIG DATA International Campus, 2020).

De acuerdo con el Dr. Álvarez (2021), existe una relación entre las emociones y el aprendizaje. Las emociones pueden afectar a la memoria y la atención de una persona. Uno de los argumentos que compartió fue: «la neurociencia nos dice que hay que trabajar las emociones desde el aula, porque las emociones comparten redes neuronales con el aprendizaje. Hay un binomio entre cognición y emoción que los maestros no podemos olvidar».

5.19. Fases de modelo de análisis de emociones

5.19.1. Lenguaje de programación

El lenguaje de programación de *Python* es uno de los más utilizados entre los desarrolladores que tienen un enfoque dirigido a análisis de datos o para la aplicación de técnicas estadísticas. Existe una tendencia hacia el lenguaje de *Python* en los desarrolladores que desean hacer su trayectoria enfocada a la ciencia de datos. Otra ventaja que tiene el lenguaje de *Python* es la integración, en un grado menos complicado, a aplicaciones web, así como la posibilidad de conexión con bases de datos. Finalmente, los modelos elaborados con este lenguaje son más fáciles de implementar, por lo que la relación costo-tiempo, se ve beneficiada al usar *Python* (Rozo et al., 2018).

5.19.2. Librerías para análisis

Para carga y manejo de datos se puede utilizar la librería ***Pandas***. Esta librería se especializa en la estructuración, análisis de estructuración y manejo de datos. Como complemento a *Pandas* se puede utilizar ***Numpy***. *Numpy* es una librería se especializa en el uso de arreglos (*arrays*), que son utilizados en gran parte con la librería de *Pandas*. Para la visualización gráfica del análisis obtenido, se puede hacer uso de la librería ***Matplotlib***. Esta librería se especializa en la creación de visualizaciones, que pueden ser estáticas, dinámicas y/o interactivas para los usuarios.

Por otro lado En el área de limpieza de datos de palabras, una de las librerías más conocidas es la librería de ***regex***. Esta permite crear expresiones regulares para crear secuencias de caracteres -patrones-, para que estos, al darle un conjunto de texto, se valide que pertenezca o no pertenezca a la expresión indicada. La librería más importante para el uso de conjuntos

de datos de palabras es la librería de ***NLTK***, la cual permite programar para procesamiento del lenguaje. Sus iniciales significan *Natural Language Toolkit* en inglés. Finalmente, para obtener los modelos de clasificación, se puede utilizar la librería ***scikit-learn***, con está se puede realizar la separación de los conjuntos de entrenamiento y pruebas (*testing*).

5.19.3. Dataset

Un *dataset* o un conjunto de datos se puede utilizar para alimentar distintos modelos de análisis de datos y entrenarlos. Al ser este proyecto enfocado en el idioma español, es necesario la utilización de una fuente de datos que tenga palabras en idioma español clasificadas. Al seleccionar una fuente de datos se puede importar a un libro de trabajo, como por ejemplo: (***jupyter notebook***) para luego pasar al segmento de preprocesamiento.

5.19.4. Técnica *Mechanical Turk*

Mechanical Turk es un servicio que creó *Amazon*. Este servicio permite a los usuarios tener a su alcance mano de obra distribuida. Esta mano de obra se puede utilizar para: llenar encuestas, clasificación de datos, entre otras cosas. La técnica de *Mechanical Turk* consiste en utilizar a un usuario o múltiples usuarios para realizar la tarea necesaria a las exigencias de un proyecto. El beneficio de esta técnica, es que, al ser datos validados por humanos, se puede identificar lo que significan los datos. Un ejemplo de esto es que un ser humano puede interpretar lo que un *emoji* o emoticón quiere expresar bajo un contexto específico (Amazon, s.f.-c).

5.19.5. Preprocesamiento

Este eslabón es uno de los pasos en los que se invierte más tiempo dentro de la ciencia de datos, debido a que un correcto preprocesamiento permite generar un conjunto de datos más estructurado, organizado y limpio. El contar con datos bien definidos permite entrenar los modelos correctamente y obtener mejores métricas.

Limpieza de datos

Cuando se trabaja con datos, se puede llegar a la conclusión que los mismos no se encuentran correctamente estructurados o limpios. Esto se evidencia especialmente cuando se trabaja con datos generados por usuarios. Comúnmente se pueden encontrar errores humanos. Algunos ejemplos de estos errores son: escribir dos letras con mayúscula en una misma palabra, escribir una tilde en una posición errónea, entre otros. La limpieza de datos puede incluir pasos como cambios a todos los datos de mayúscula a minúsculas o viceversa. Sin embargo, esta limpieza puede incluir también la eliminación de *tags*, *urls* y puntuaciones. Cada una de estas opciones se aplican dependiendo de la finalidad de cada proyecto, pero cumplen el mismo objetivo de dejar los datos lo mayormente limpios y estructurados posibles.

Tokenización

La *tokenización* es uno de los primeros pasos que se debe realizar para el procesamiento de lenguaje natural. La *tokenización* toma un texto y lo divide en porciones. Esta división puede ser por palabras (*Tokenización* por palabra) u oraciones (*Tokenización* por oración) (NLTK, s.f.). La *tokenización* es un paso requerido, debido a que distintas palabras dentro de una oración pueden contener diferentes significados. Utilizando la *tokenización*, se pueden utilizar técnicas de estadística para obtener mayor información (Gunjal, 2020).

Stopwords

Se conoce las *stopwords* como **palabras vacías**, ya que están carecen de significado importante; pueden ser artículos, conjunciones, preposiciones, adverbios. Estas palabras carecen de un significado de valor por sí solas. Para evitar que estás palabras generen error o ruido en los modelos, se deben eliminar de la fuente de datos. La librería de *NLTK*, provee una lista de 325 *stopwords* (Ortega, 2021).

Duplicados

Uno de los problemas consistentes al manejar los datos, son las entradas duplicadas. Estas pueden afectar directamente en la fase del modelado y también pueden generar otros problemas como: la obtención de una predicción incorrecta; retorno de datos obsoletos o bien, pueden disminuir las métricas obtenidas del modelo (King, 2022).

5.19.6. Análisis exploratorio

Stemmer

El trabajo de aplicar *Stemmer* a los datos es el de obtener las palabras raíces: por lo que a una palabra compuesta por prefijos o sufijos regresara a su palabra raíz. Lo que esto realiza es obtener todas aquellas palabras en su forma básica, por lo que hace que un conjunto de datos retorne todas aquellas palabras raíces. Esto hace que, en vez de tener tres palabras como: «jugar», «jugó» y «jugando», se obtenga solo una: «jugar» (Jabeen, 2018).

Histogramas

Los histogramas se utilizan cuando se necesita una visualización de distribución de valores, sobre la cantidad de instancias de cada valor. Esto permite tener un mejor panorama de si será necesario, en un futuro paso, balancear los datos (IBM, 2021).

Wordcloud

Es un recurso ilustrativo para mostrar un conjunto de palabras, donde el tamaño y proporción, dentro del *Wordcloud*, depende de la frecuencia con la que se repite esa palabra. Este recurso puede indicar si palabras se repiten múltiples veces dentro de un conjunto de datos. La ventaja de utilizar esta visualización, es la de reconocer si es necesario aplicar otro paso de limpieza de duplicados (Roncal, 2021).

5.19.7. Modelación

División entre datos de entrenamiento y *testing*

La separación de los datos provee un recurso para entrenar un modelo y después de ser entrenado se pueden realizar pruebas con datos que se saben que si son cual es la respuesta correcta. Al hacer la separación de los datos, se debe tomar en cuenta las porciones que se le otorgaran a cada parte, ya que de esto dependerá si habrá sobreajuste o desajuste (Microsoft Learn, 2022).

Balanceo de datos

Cuando se tienen limpio el conjunto de datos, se debe revisar si los valores están balanceados. Un ejemplo es que se tienen dos clases a y b, se dice que están balanceados cuando el contador de estas clases sea el mismo número y se dice que los datos están desbalanceados cuando no se cumple con la condición anterior. El desbalanceo de datos significa que una clase tiene un contador mayor a las otras, lo que indica que acumula un mayor número de clasificaciones que las demás (Vladislavleva et al., 2010). Debido a esto, una predicción del modelo será más propensa a ser correcta de una clase sobre las otras, bajando el rendimiento del modelo. En muchos casos la clase de menor contador es la clase objetivo, por lo que el problema de desbalanceo de datos se debe mitigar (García, 2021).

Oversampling: El *Oversampling* es una técnica para el balanceo de datos, el camino a seguir con está técnica es tomar el contador de la clase mayoritaria y se determina como el número de instancias que deberán tener todas las clases. Las clases minoritarias aumentan para que tengan la misma cantidad de muestras que la clase mayoritaria. Este aumento se da a través de la creación de réplicas de cada clase minoritaria. En otras palabras, se balancean los datos creando instancias para que todas tengan el máximo número de instancias (García, 2021).

Dentro de la técnica existen 3 métodos

1. *Random Oversampling*: Método no heurístico que replica las instancias ya existentes de la clase hasta llegar al número de instancias de la clase mayorista (García, 2021).
2. *SMOTE Synthetic Minority Over-Sampling Technique*: Método heurístico que replica instancias sintéticas de la clase minorista, tomando como apoyo el método de los k

vecinos más cercanos (García, 2021).

3. *ADASYN Adaptive Synthetic Sampling*: Siendo una aplicación del método *SMOTE*, *ADASYN* aplica la misma metodología que *SMOTE*, con un paso extra de adición de distorsión a los datos, con el argumento que los hace más realistas (García, 2021).

Undersampling: El *Undersampling* es una técnica, que busca balancear a todas la clases, tomando el contador de muestras de la clase con menos muestras y hacer que las otras clases posean la misma cantidad de instancias a esa. Por lo que, se eliminan instancias de las clases con más muestras, dejando todos los datos con una cantidad menor pero igual a la cantidad general. Una de las desventajas de este método, es la pérdida de información original, que pudiese sido utilizada para el entrenamiento o *testing* del modelo.

Dentro de la técnica existen 2 métodos

1. *Random Undersampling (RUS)*: Método no heurístico que elimina las instancias, de manera aleatoria, ya existentes en la clase hasta llegar al número de instancias de la clase minorista (García, 2021).

2. *Condensed Nearest Neighbors*: Método heurístico que parte del método de k vecinos más cercanos. Debido a esto, toma los casos o instancias más similares a través de clasificación de las instancias. (García, 2021).

Distribución entre características

Este segmento puede ser utilizado para identificar que los datos estén balanceados, incluso al estar separados en conjuntos de entreno y *testing*. Esto permite visualizar que tanto en conjunto de entreno como en el conjunto de pruebas las clases tengan porcentajes similares.

CountVectorizer

Esta herramienta proviene de la librería de *scikit-learn* de *Python*. Esta permite transformar un texto en forma de un vector y , con un conjunto de texto, crea una matriz con cada palabra única. Al tener esta estructura de datos se tiene cierta ventaja en la estructuración de los datos (GeeksforGeeks, 2022).

Bag of words

Es un método que permite extraer características relevante de textos. El método permite crear un vocabulario de palabras únicas que se pueden encontrar en los conjuntos de datos. Su objetivo es representar texto con contadores, sin tomar en consideración el orden en el que estén las palabras de los textos (Dubey, 2018).

Sobreajuste (*overfitting*)

Se da cuando un modelo intenta ajustarse cuando ha sido entrenado con casos particulares, por lo que este se comporta de forma errónea al recibir datos ligeramente distintos a los datos de entrenamiento. Esto afecta en las pruebas, debido a que el modelo no tiene la capacidad para reconocer nuevos casos. Se puede reconocer cuando un modelo tiene *overfitting*, cuando al ser puesto a prueba con un nuevo caso, este no logre identificarlo (Aprende Machine Learning, 2017).

Desajuste (*underfitting*)

Se da cuando un modelo intenta ajustarse cuando ha sido entrenado con solo una clase. Esto afecta en las pruebas, debido a que el modelo no tiene la capacidad para reconocer algo diferente de esa clase. Se puede reconocer cuando un modelo tiene *underfitting*, cuando al ser puesto a prueba con una nueva clase, este no logre identificarlo (Aprende Machine Learning, 2017).

5.19.8. Modelos de clasificación

Los modelos de clasificación son los que tienen el objetivo de asignar los objetos de entrada en categorías. La extracción de datos por clasificación es una de las técnicas más comúnmente aplicadas (Microsoft Learn, s.f.).

BernoulliNB

Naive Bayes es uno de los algoritmos de clasificación más simples. *BernoulliNB* es normalmente utilizado en conjuntos de datos de entrenamiento en clasificación de texto, especialmente en clases binarias (Khushijain, 2022).

LogisticRegression

LogisticRegression es uno de los algoritmos mayormente utilizados para modelos de clasificación; normalmente es utilizado para clasificación binaria. Este algoritmo tiene la versatilidad de volverse un algoritmo *multiclase*. En el caso de *muticlase* se deben tener diferentes parámetros.

MultinomialNB

Naive Bayes es uno de los algoritmos de clasificación más simples. *MultinomialNB* es normalmente utilizado cuando las clases representan la cantidad de veces que se encuentran en el conjunto de datos, utilizado con vectores y la frecuencia de uso de los textos determinará a que grupo clasifica la entrada (Khushijain, 2022).

RandomForestClassifier

El *RandomForestClassifier* puede ser tanto un modelo de clasificación como un modelo de regresión. Este se comprende de arboles, que por lo general son arboles de decisión. Usualmente, entre más árboles tenga mejor será el resultado (DataCamp, 2018).

SVM

Las *Support Vector Machines* (*SVM*), son muy similares a la regresión logística (*LogisticRegression*), pero no necesita de tantos recursos computacionales como la regresión logística. Pueden ser utilizadas en clasificación *multiclase*. Utilizan el concepto de hiperplanos y vectores de apoyo para la clasificación (Gandhi, 2018).

5.19.9. Métricas de modelos

Reporte de clasificación

***Accuracy*:** Es la métrica que indica el porcentaje total de elementos clasificados correctamente. Es la métrica que nos dice con mayor exactitud cómo fue el rendimiento del modelo (Educative, s.f.).

***Precision*:** Es la métrica que toma el número de elementos identificados correctamente como positivos del total de elementos clasificados correctamente (Educative, s.f.).

***Recall*:** Es la métrica que proporciona el número de elementos que se predijeron correctamente de una clase (Toward AI, 2022).

***F1-score*:** Es la métrica que proporciona el resultado de la media entre el campo de *precision* y el campo de *recall* (Toward AI, 2022).

Matriz de confusión

True positive La cantidad de clasificaciones con correcta predicción positiva en la clasificación de una entrada.

True negative La cantidad de clasificaciones con correcta predicción negativa en la clasificación de una entrada

False positive La cantidad de clasificaciones con incorrecta predicción positiva en la clasificación de una entrada.

False negative La cantidad de clasificaciones con incorrecta predicción negativa en la clasificación de una entrada.

5.20. Visualización

5.20.1. *Tableau Prep*

Es una herramienta de análisis, limpieza y estructuración de datos que permite realizar estas acciones de forma efectiva. Una ventaja del uso de *Tableau Prep* es que se puede utilizar datos crudos, además que es muy fácil de integrar a la herramienta de visualización de *Tableau Desktop* (trustRadius, 2019).

La sincronización de datos se puede programar si se tiene un flujo de datos en *Tableau Cloud*. Si no se tiene la disponibilidad de *Tableau Cloud*, se puede utilizar *Tableau Prep* sin ningún problema: solo se deben correr los flujos de datos desde local para que sean actualizados.

Tableau Prep permite tener un archivo con múltiples flujos. Otra cualidad que tiene permite la rápida unión por columnas y unión por filas de dos flujos diferentes.

Flujo de datos

En el punto anterior, se hablaron de flujos de datos. Un **flujo de datos** es un equivalente a un *DataFrame* de *Python*. Es un conjunto de datos que se encuentra preprocesado y analizado, listo para ser utilizado en alguna herramienta de visualización (trustRadius, 2019).

Hyper archivos

Hyper es la tecnología desarrollada, de *Tableau*, de motor de datos de memoria. Esta se encuentra optimizada para el consumo rápido de datos y procesamiento de consultas de analítica en conjuntos grandes de datos (Tableau, s.f.).

5.20.2. *Tableau Desktop*

Es un software de visualización que permite el acceso a datos, para análisis de los mismos. Con estas facilidades se pueden encontrar aprendizajes (*insights*) de valor, que normalmente están escondidos en los datos. Esta herramienta tiene la capacidad de obtener *insights* valiosos en menor tiempo. Al mostrar las visualizaciones, *Tableau Desktop* permite un mejor despliegue de escenarios para una mejor toma de decisiones. Además, se pueden compartir los documentos con los *dashboards* y colaborar en los proyecto con *Tableau Server*. *Tableau* ofrece una licencia para estudiantes y cursos para aprender a usar la herramienta de manera gratuita con *Tableau for Education*.

CAPÍTULO 6

Metodología y diseño de prototipo

6.1. Contexto de la Universidad del Valle de Guatemala

La Universidad del Valle de Guatemala (UVG) es una universidad privada ubicada en la República de Guatemala. Esta universidad cuenta con 7 facultades: ingeniería, educación, ciencias y humanidades, ciencias sociales, *business and management school*, colegio universitario y *design innovation & arts school*. Dentro de estas facultades, UVG tiene disponibles más de 45 licenciaturas y 20 maestrías. Esta universidad cuenta con 3 campus: central, en la Ciudad de Guatemala; altiplano, en Sololá y sur, en Escuintla (Universidad del Valle de Guatemala, 2022).

La descripción de la universidad anteriormente mencionada es de importancia, gracias a que da a conocer la fragmentación de la educación que existe dentro de una solo entidad educativa, incluso si estas tres partes componen una misma institución en un entorno universitario. Por esto, la elección de un paradigma de diseño, como fue descrito en la página 27, en la sección 5.8.1, fue clave para la elaboración de este proyecto.

Para realizar el proyecto de por sí, se escogió un *Framework* para el *backend* con enfoque en lo **escalable** y en lo **seguro**. Gracias a que el *framework* escogido en el *frontend* es *Angular*, el *backend* estuvo muy bien adaptado para el uso de *Nest.js*.

Para poder entender los procesos del sistema, se trataron de descubrir las distintas necesidades del público objetivo con el módulo de *frontend*, donde se siguió el proceso de *design thinking* para entender las necesidades de los estudiantes y de los docentes, tomando al campus central como población de estudio. Mediante la fase inicial, se trató de encontrar todo lo que tenían en común los varios miembros del público objetivo. Sin embargo, se tomó también nota de aquellas cosas que no compartían para diseñar este proyecto.

6.2. Herramientas utilizadas para la elaboración del frontend

6.2.1. *Google Forms*

Google Forms fue utilizado para la elaboración de encuestas exploratorias debido a la posibilidad de validar correos institucionales de la Universidad del Valle de Guatemala. Esto permitió verificar que las encuestas fueran respondidas una sola vez y que solamente pudieran ser accedidas por miembros de la comunidad UVG. También permitió confirmar que la encuesta elaborada para estudiantes y la encuesta elaborada para catedráticos fueran respondidas solamente por estos. Cualquier encuesta respondida de forma errónea (por ejemplo, un estudiante respondiendo la encuesta de catedráticos) sería descartada, sin embargo esto no sucedió.

La utilización de *Google Forms* también permitió mantener las políticas de seguridad de la universidad al momento de difundirlas por correo. Esto es debido a que la universidad utiliza *Gmail*, el correo de *Google*. El haber utilizado un enlace de otro proveedor de encuestas podría haber generado desconfianza.

6.2.2. *Figma*

Figma es una herramienta para el diseño de interfaces y cuenta con aplicaciones de escritorio para *Windows* y *macOS* y fue utilizado para realizar prototipos digitales de las distintas pantallas del lector. Fue seleccionada debido a que integra todas las herramientas necesarias para el diseño de una interfaz, incluyendo generación de código *css* base, maquetado y diseños de pantallas. Además, esta herramienta cuenta con colaboración en tiempo real, lo que lo hizo ideal para un proyecto de este tipo, debido a que este módulo se diseñó en conjunto con el *backend* para poder ser integrado. Si bien el *backend* no maneja diseños, el poder ver el proceso de prototipado permitió un mejor diseño de las posibles consultas a realizar.

El módulo de ciencia de datos («Elaboración de un recolector de datos de archivos EPUB para la creación de analizador de información del lector de libros electrónicos colaborativos para su utilización en el ámbito académico») también se benefició con esta herramienta, debido a que al ser de carácter colaborativo permitió ver cómo sería la forma de capturar los datos desde el inicio (*frontend*), etapa media (*backend*) y procesamiento final (ya dentro del módulo de ciencia de datos. Esto resultó en un mejor flujo de comunicación entre los distintos módulos del proyecto.

6.2.3. *Coblis - Color Blindness Simulator*

El simulador de daltonismo, *Coblis*, elaborado por Wickline (2001) permitió visualizar diferentes paletas de colores para resaltados y fondos de la forma en la que lo vería una persona con daltonismo. Este simulador se utilizó para probar los colores de resaltado teóricos y posteriormente, validar las paletas programadas en la interfaz. El funcionamiento de este simulador puede ser visto en la Figura 3.

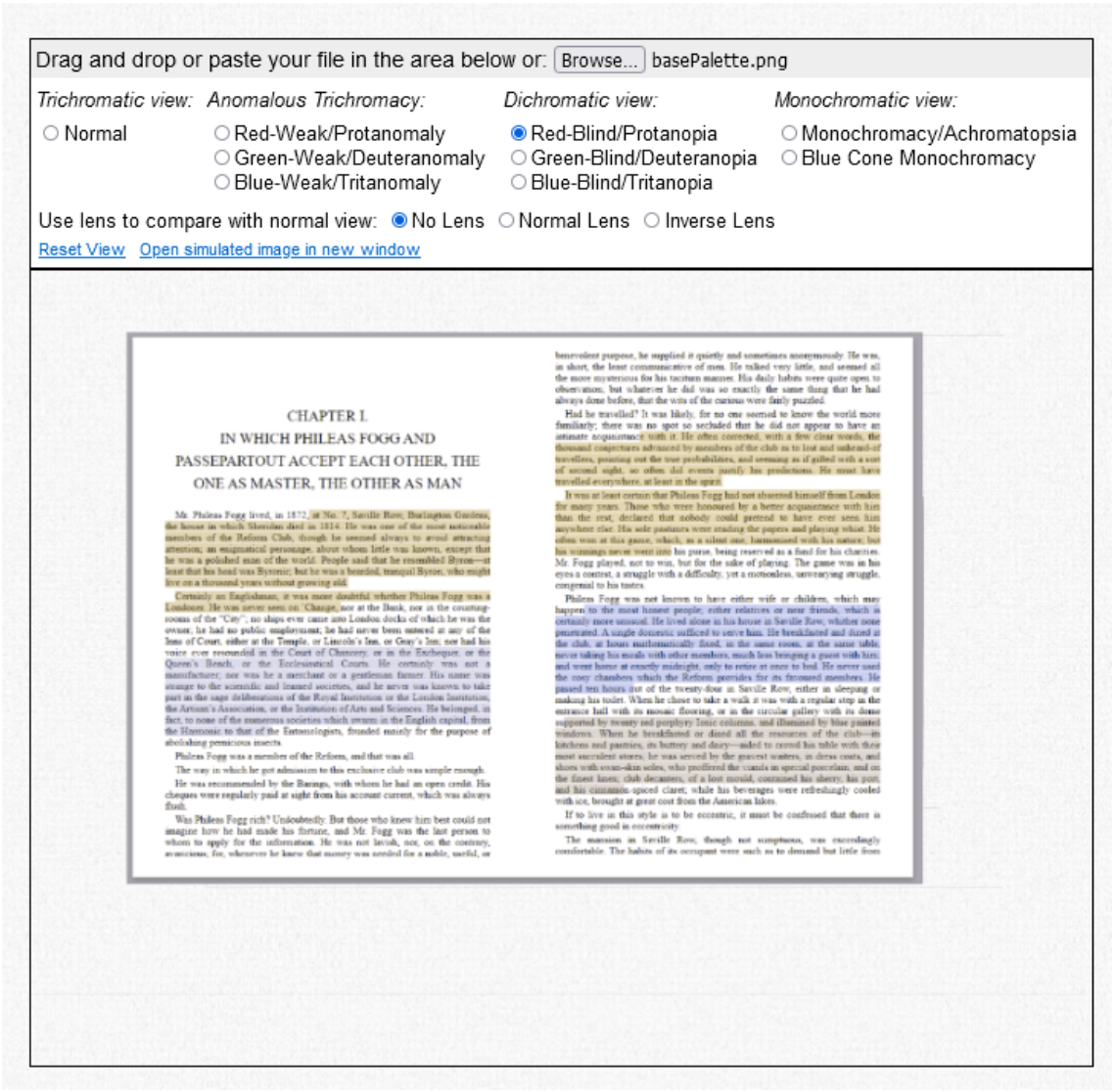


Figura 3: Vista de filtros por medio de Coblis

6.2.4. *WebAIM Contrast Checker*

WebAIM se utilizó para medir el contraste entre diferentes colores dentro de la aplicación. Su uso más intensivo fue para medir los contrastes teóricos entre los colores de resaltado y textos del lector para poder generar una paleta óptima que se adaptara a diferentes usuarios y siguiendo los requerimientos de *WCAG Web Content Accessibility Guidelines* (Pautas de Accesibilidad para el Contenido Web en español).

Se seleccionó esta herramienta debido a que está realizada por un instituto dedicado a la accesibilidad de páginas web por todo tipo de usuarios y cuenta con una forma gráfica de ver los contrastes así como el cumplimiento de los diferentes niveles de *WCAG* (WebAIM, s.f.). El funcionamiento de esta herramienta puede ser visto en la Figura 4. Para validar que sus mediciones fuera correctas, sus resultados fueron comparados con la herramienta *Color Check for ADA Image Compliance* de la sección 6.2.5.

6.2.5. *Color Check for ADA Image Compliance*

Esta herramienta fue utilizada en conjunto con *WebAIM Contrast Checker* (sección 6.2.4) para validar que los colores seleccionados cumplieran con las pautas de accesibilidad web. *Color Check for ADA Image Compliance* permite analizar los colores directamente desde imágenes, lo que facilitó la medición de contraste para elementos donde solamente se tenía una imagen, debido a que en lugar de tener que encontrar individualmente los códigos hexadecimales de los colores que se tenían que medir, se podía subir la imagen a la herramienta y analizarla directamente. *Color Check for ADA Image Compliance*, elaborado por Off-Site Services, Inc. (s.f.) fue utilizado para medir las imágenes generadas con filtros a partir de *Coblis* (sección 6.2.3) y su correcto funcionamiento fue validado al comparar los resultados obtenidos por esta herramienta con aquellos obtenidos por *WebAIM Contrast Checker*.

6.2.6. *GitHub*

GitHub fue seleccionado como herramienta para manejo de versiones y colaboraciones. Esto es debido a que los miembros del equipo ya contaban con amplia experiencia en esta herramienta, además, se ha observado que la mayoría de personas en ciencias de la computación dentro de la Universidad del Valle de Guatemala utilizan *GitHub*, lo que lo ideal para que en el futuro otros desarrolladores puedan ingresar al proyecto y conocer el funcionamiento del control de versiones existente.

Para manejar el código, se elaboró una organización dentro de *GitHub* con 3 repositorios: 1 para cada módulo. En el caso de este módulo, se aprovechó *GitHub* para tener una rama principal y otras ramas de desarrollo. Se utilizó una rama *main* con el código principal y una rama *dev* de desarrollo con corta vida para probar funcionalidades. Si bien, no se lanzaron *builds* a clientes finales de la aplicación, se siguió una metodología similar a *trunk based development*, para probar la aplicación en *Docker* (sección 6.2.8). Es decir, la aplicación final fue lanzada a *Docker* desde la rama principal.

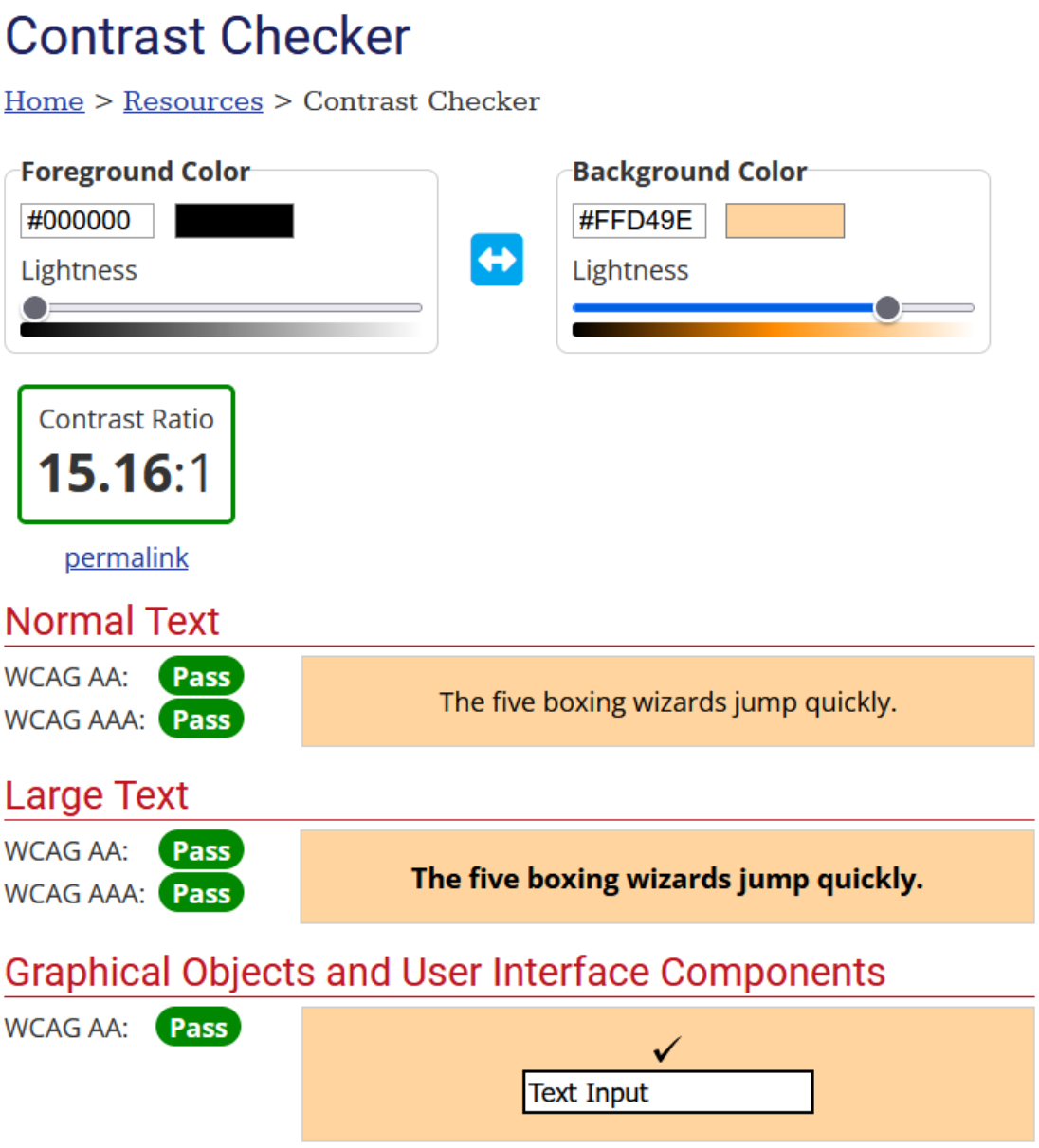


Figura 4: Vista de pruebas de contraste en WebAIM Contrast Checker

6.2.7. *GitHub Issues*

GitHub Issues fue utilizado para generar elementos de trabajo para los distintos repositorios dentro del proyecto. En el caso de este módulo, fue utilizado más durante una fase de prototipado más avanzada donde ya se programaron las pantallas. Para un equipo más grande se recomienda considerar otras herramientas como *Jira* y utilizar *GitHub Issues* como una herramienta de comunicación con una comunidad.

6.2.8. Docker

Docker fue utilizado para probar el *frontend* en ambientes de producción. Esta herramienta permite separar la aplicación de la infraestructura local (en este caso, la computadora del autor de este módulo) para entregar *software* de forma rápida. Este es de código abierto y los contenedores generados por *Docker* pueden correr en cualquier servidor de *Linux*, el cual podría ser utilizado en un futuro para publicar el lector.

6.2.9. *Angular*

Angular es la base del *frontend* de este módulo. Este marco de trabajo o *framework* fue utilizado para programar e implementar las distintas funcionalidades de la interfaz del lector colaborativo. Esto incluye la elaboración de los distintos componentes del lector; la estilización de los componentes elaborados; la implementación de los servicios y lógica detrás de estos componentes y los servicios para conectarse con el *backend*.

Uso de *TypeScript*

El uso de *TypeScript* en *Angular* fue un gran punto a favor para la selección de este marco de trabajo debido a que se busca que el proyecto pueda ser mantenido en el futuro, tanto por personas dentro del megaproyecto, como nuevos desarrolladores que deseen contribuir al lector o bien quieran desarrollar proyectos de graduación en torno a él. De esta forma, se podrán hacer megaproyectos tecnológicos con un alcance aún mayor.

Tipo de arquitectura

El estar en constante comunicación con el módulo de «*Backend* y bases de datos» permitió la definición de herramientas compatibles entre sí, con un patrón de arquitectura similar. Tanto *Nest.js* como *Angular* utilizan un patrón de arquitectura de tipo *MVC*, *Model View Controller* (Modelo Vista Controlador). También, debido a que *Nest.js* utiliza *Express* por defecto, se definió una pila de desarrollo similar a *MEAN*, modificada para utilizar una base de datos relacional.

Selección por tipo de aplicación

Angular es una herramienta adecuada para manejar proyectos complejos debido a su arquitectura definida. Si bien, no permite la flexibilidad que otros marcos de trabajo pueden proporcionar, el contar con una arquitectura más definida permite construir aplicaciones de gran escala de una forma más sencilla (Pfalzgraf, 2022) por esta razón se decidió utilizar *Angular* para este proyecto. Esto es debido a que, de haber proyectos futuros que extiendan los resultados de este, la escala de la aplicación podría aumentar considerablemente y contar con una arquitectura definida ayudaría a mantener una misma línea de desarrollo.

Uso de *SOLID*

Para desarrollar el *frontend* se utilizaron los principios *SOLID* para poder generar código de buena calidad para futuros proyectos.

Uso de *Prettier* y *ESLint*

Se utilizó *Prettier* (herramienta de formato de código) y *ESLint* (herramienta de análisis de código) en conjunto con *Angular* para poder estilizar código de una forma consistente. Además, se definieron reglas de accesibilidad en *ESLint* para poder programar de una forma que incluya elementos necesarios para personas con diferentes capacidades. Las reglas definidas incluyen:

- **@angular-eslint/template/accessibility-valid-aria:** (*Accessible Rich Internet Applications*) incluye reglas y atributos para hacer el contenido web más accesible para personas con discapacidades.
- **@angular-eslint/template/accessibility-alt-text:** este provee una descripción de elementos visuales para personas con alguna discapacidad visual.
- **@angular-eslint/template/accessibility-elements-content:** Se utiliza para validar que los encabezados, *anchors* y botones tengan contenido accesible.
- **@angular-eslint/template/accessibility-label-for:** Se asegura de que las etiquetas se encuentren asociadas con formularios.
- **@angular-eslint/template/no-positive-tabindex:** Se asegura que el índice de distintos elementos de la interfaz puedan ser accedidos por medio del teclado en una forma lógica por medio de la tecla tabulador.
- **@angular-eslint/template/click-events-have-key-events:** Ayuda a que los elementos con *click* utilicen también eventos de teclado.
- **@angular-eslint/template/mouse-events-have-key-events:** Se asegura de que cualquier evento con eventos de *mouse* también cuente con eventos de borrosidad (*blur*) y que también cuente con eventos de enfoque (*focus*).

6.2.10. *Epubjs*

Epubjs es una librería de *JavaScript* que permite desplegar libros digitales en el formato *epub*. Esta herramienta se seleccionó debido a que contaba con la mayor documentación y soporte. También es de código abierto y cuenta con una licencia permisiva, ideal para este tipo de proyectos en caso se quiera utilizar para fines comerciales o masificar.

Se probó utilizar componentes ya generados con *Epubjs* para desplegar el lector, sin embargo estos contaban con vulnerabilidades de seguridad, por lo que la implementación de *Epubjs* en el lector fue realizada directamente en el proyecto.

6.3. *Design thinking*

Se aplicó el proceso de *design thinking* para empatizar con los potenciales usuarios del lector y generar ideas creativas para comprender de mejor manera las necesidades de los usuarios y la forma en la cual diseñar un lector. Este proceso fue aplicado en conjunto con los principios de la interacción humano-computador (*HCI*) para elaborar una interfaz natural e intuitiva para los estudiantes. En este trabajo se presentan las fases de *design thinking* en el orden de: empatizar, definir, idear, prototipar y probar. No obstante, cabe mencionar que las fases de idear, prototipar y probar no fueron realizadas en orden necesariamente. Esto es debido a que inicialmente se idearon las pantallas, luego se prototiparon y finalmente, se pusieron a prueba. Sin embargo, al momento de encontrar oportunidades de mejora en las pruebas, se volvió a seguir el proceso de idear, prototipar y probar. El desarrollo del proceso de *design thinking* se realizó en el segundo semestre del año 2022, durante los meses de julio, agosto, septiembre, octubre y noviembre.

6.3.1. Empatizar

Esta fase fue realizada principalmente a inicios del proyecto (meses de julio y agosto del proyecto) sin embargo, se siguió empatizando con el usuario final a lo largo de todo el proyecto a través de pruebas de prototipos, investigación y entrevistas informales.

Encuestas iniciales

Para realizar este proyecto se elaboraron encuestas para ser respondidas por distintos estudiantes pertenecientes a la Universidad del Valle de Guatemala (UVG). Para ello se realizaron dos encuestas: una enfocada a los estudiantes de esta universidad y otra a los docentes. Si bien, el alcance de este proyecto se determinó en realizar un lector para los estudiantes, también se estudió a los catedráticos en caso también quieran utilizar este lector o para utilizarse como base para un proyecto en el futuro que aumente el alcance de este. En el caso de la encuesta a estudiantes, se indagó sobre sus técnicas de estudio, tipos de libros que utilizan (digitales o físicos) y lo que buscan en lectores de libros y aplicaciones académicas.

En el caso de la encuesta a docentes, se buscó información sobre sus procesos de enseñanza y la retroalimentación que necesitan al momento de dar una clase. También se indagó sobre el tipo de literatura que utilizan en sus clases y lo que buscan en aplicaciones de lectura de libros y académicas.

Los estudiantes y docentes encuestados fueron todos pertenecientes al campus central. Las encuestas fueron difundidas por el correo institucional y *WhatsApp*. Se siguieron aceptando respuestas a lo largo de todo el proceso y se obtuvo un total de 33 respuestas de estudiantes y 9 respuestas de catedráticos.

Entrevistas

Se entrevistó tanto a estudiantes como docentes del campus central de UVG con preguntas abiertas para captar de forma más personalizada distintas necesidades y poder empatizar mejor con la situación de la universidad y el usuario objetivo. En total se entrevistó a 10 estudiantes y 5 catedráticos. Las entrevistas a usuarios fueron elaboradas por el autor de este módulo en formato uno-a-uno, mientras que las entrevistas a catedráticos fueron realizadas con 2 miembros de este megaproyecto a la vez: 4 de 5 entrevistas fueron realizadas por la autora del módulo de «Elaboración de un recolector de datos de archivos EPUB para la creación de analizador de información del lector de libros electrónicos colaborativos para su utilización en el ámbito académico», Cristina Bautista, y el autor de este módulo, Pablo Ruiz. Uno de los catedráticos fue entrevistado por el autor del módulo de «*Backend* y bases de datos», Andrés Quan y el autor de este módulo.

Debido a que las entrevistas a estudiantes fueron en formato uno-a-uno y también debido a que el alcance de este módulo es elaborar un lector para estudiantes, se decidió realizar el doble de entrevistas a estudiantes que a catedráticos.

En cuanto a las entrevistas a catedráticos, se decidió entrevistar a 5 personas debido a que, al ser una entrevista con 2 miembros del equipo, se consideró que sería más sencillo extraer conocimientos latentes o *insights*. La mayoría de estas entrevistas realizaron en conjunto con Cristina Bautista para poder indagar más sobre las estadísticas que serían útiles para catedráticos que podrían ser útiles en su módulo. Los resultados de estas entrevistas fueron compartidos con los miembros del megaproyecto de forma anónima.

Diseño de las entrevistas empáticas

Las entrevistas fueron diseñadas de forma semi-estructurada y según las especificaciones de *Empathy Interviews* por Kari Nelsestuen y Julie Smith (2020). Para ello se tomó en cuenta los siguientes aspectos:

- Consentimiento informado
    - Se le informó al entrevistado que la entrevista estaba siendo conducida para entender mejor el ambiente de UVG para este megaproyecto.
    - Se explicó que los datos serían utilizados de forma anónima y que su nombre no aparecería en el estudio.

    - Se expuso que el responder era totalmente voluntario y que en cualquier momento podría decidir no responder a alguna pregunta.

- Se inició con una pregunta positiva

    - Para los estudiantes se utilizó: "¿Qué te motiva a estudiar?"
    - Para los catedráticos se utilizó: "¿Qué lo motiva a ser catedrático?"

- Se evitó utilizar acrónimos y palabras complejas.

- Se buscó limitar el sesgo al utilizar preguntas sin sugestión de una respuesta correcta.

- Se preguntó por historias principalmente.

- Se indagó sobre lectores fuera de la universidad y experiencias no limitadas al campo de estudio.

- En caso de encontrar un punto donde se consideraba necesario profundizar, se utilizaron preguntas como: "¿por qué?.º "¿cómo te hizo sentir?".

La estructura de la entrevista para estudiantes puede ser observada en el anexo 11.1 y la de catedráticos puede ser observada en el anexo 11.2.

Técnicas de investigación contextual

Además de las entrevistas y encuestas, se observó el ambiente de la Universidad del Valle de Guatemala para poder evidenciar las formas de estudio y los dispositivos que utilizaban los estudiantes al momento de estudiar. Al ser los autores de este megaproyecto miembros activos de esta universidad durante la realización de este proyecto, se pudo obtener una idea general de lo que viven los estudiantes en el día a día. Sin embargo, para evitar una visión sesgada de la comunidad, el autor de este módulo buscó también realizar una investigación contextual para observar de forma general los patrones de estudio de otros estudiantes.

Esta observación se realizó principalmente en 3 tipos de ambientes: plazas, salones de clase y la biblioteca. Mediante las técnicas de observación se pudo identificar la estructura de los grupos de estudio, y los dispositivos utilizados para estudiar, realizar trabajos y leer dentro de la universidad.

También se visitó la librería Sophos y KitaPenas de la Ciudad de Guatemala para observar la estructura de las mismas. En estas librerías se identificó el proceso de selección de libros y compra de los mismos, así como la presentación de diferentes textos.

Las técnicas de observación fueron realizadas a lo largo de todo el proyecto, sin embargo, se concentró en los meses de julio y agosto de 2022.

6.3.2. Definir

Definición de usuarios

Una vez recopilados los datos iniciales, utilizando como base la fase de empatizar se definieron los tipos de usuarios que interactuarían con el lector colaborativo bajo la pregunta: «¿Cómo podríamos nosotros realizar un lector colaborativo para el ámbito académico que se adapte a las necesidades de los estudiantes?». Los resultados de esta fase se pueden evidenciar en el capítulo de resultados: 7 7.1.

Se definió a los usuarios principales: estudiantes y catedráticos. A pesar de que este módulo solamente buscó desarrollar la interfaz para estudiantes, también se generó un perfil de catedrático por si en el futuro se desee continuar sobre este trabajo. Además, entender de mejor manera el perfil de catedráticos ayudó a complementar la información proporcionada por los estudiantes y fue útil para desarrollar el *backend* y elaborar estadísticas para ciencia de datos.

También se estableció de manera general aspectos clave para introducir una aplicación en la universidad, lo cual se definió a partir de los resultados de un estudio realizado en 2018 y principios de 2019, donde se investigó sobre la introducción de *Canvas* en la universidad y los pasos a realizar al momento de introducir una nueva aplicación en UVG.

Definición de requerimientos del lector

Los requerimientos iniciales de la interfaz eran los siguientes:

1. La interfaz del lector debe poder desplegar publicaciones de tipo *epub*.
2. El lector debe contar con la funcionalidad de añadir comentarios y resaltar textos.
3. La plataforma de desarrollo del lector debe permitir una conexión con un *backend*.

Estos fueron establecidos a gran escala previo a la fase de empatizar, para posteriormente ser refinados al momento de contar con más información por medio de las técnicas de la fase inicial del proceso de *design thinking*.

Después de la fase inicial de *design thinking* (empatizar), se definió más a detalle todo aquello que era necesario para la interfaz:

1. El lector debe ser desarrollado para computadoras, debido a que la mayoría de estudiantes utiliza estos dispositivos para acceder a libros electrónicos al momento de estudiar.
2. El despliegue de documentos debe tener suficiente contraste para ser de fácil uso para personas con diferentes capacidades de visión.
3. Los resaltados y comentarios no deben tener significado únicamente por su color, para ser amigable con personas con daltonismo.

4. No debe de tomar más de 2 pasos entrar a un libro a partir del momento en que se inicia sesión.
5. Se prioriza la facilidad de uso de las funcionalidades necesarias sobre funcionalidades adicionales que no estén pulidas o que compliquen los flujos de usuario.
6. Se debe poder buscar libros de forma sencilla a través de la interfaz.
7. Se debe agregar medidas desde el momento de desarrollo que faciliten un lector inclusivo para personas con diferentes capacidades.
8. La forma de desarrollar el lector debe permitir que futuros desarrolladores puedan conectarlo con *APIs* (para poder integrarse con *Canvas*, por ejemplo).
9. El código desarrollado debe contar con herramientas que mantengan su legibilidad y estilización, en el caso de que se requiera que un equipo más grande trabaje sobre él en el desarrollo de futuros proyectos.

Posterior a estos requerimientos se fueron agregando elementos mediante se descubrían nuevas necesidades por medio de la fases de prototipar y probar.

6.3.3. Idear

SCAMPER

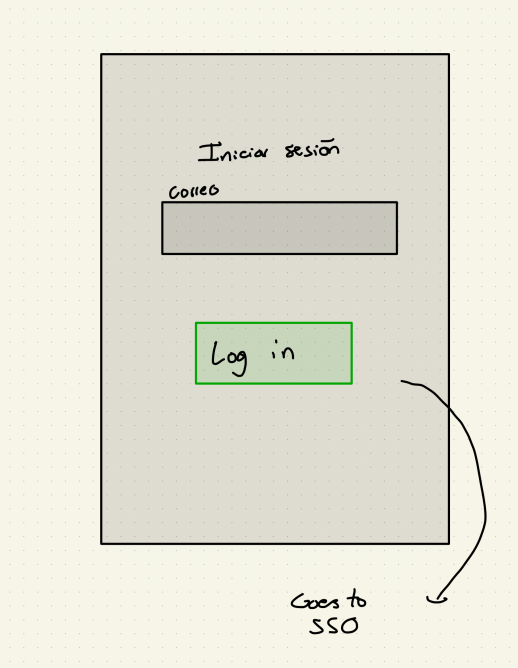
Se utilizó la técnica de *SCAMPER* (sustituir, combinar, adaptar, modificar, poner en otro uso, eliminar y reorganizar) para idear el lector inicial. Para ello se utilizó *combinar* y *adaptar*. El lector inicial se ideó como una combinación entre un lector tradicional como *Kindle* o *iBooks* y *Google Drive*. Esto permitiría la combinación entre la lectura de textos y la generación de comentarios y resaltados sobre un mismo texto. También se buscó adaptar la lectura tradicional a un diseño virtual, tomando como influencia lo observado en librerías, el manejo de libros físicos y las entrevistas realizadas.

Ideación por medio de dibujos

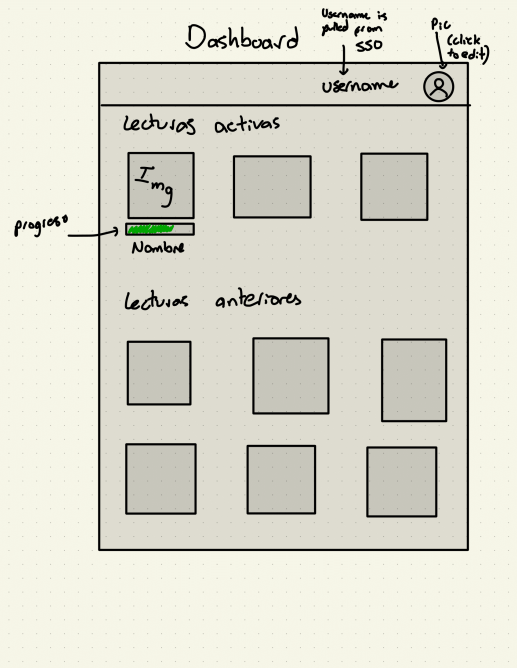
Al tener el lector inicial definido, se hizo dibujos de las pantallas base para observar posibles elementos necesarios en el *backend* para el funcionamiento del lector. Estos dibujos se tomaron como base para prototipar en *Figma*, sin embargo, los prototipos iniciales se realizaron de forma básica, para que los usuarios pudieran expresar de una manera más abierta sus necesidades al momento de realizar la fase de pruebas de estos prototipos básicos. Los resultados de la fase de ideación por medio de dibujos se puede observar en la Figura 5.

Ideación posterior

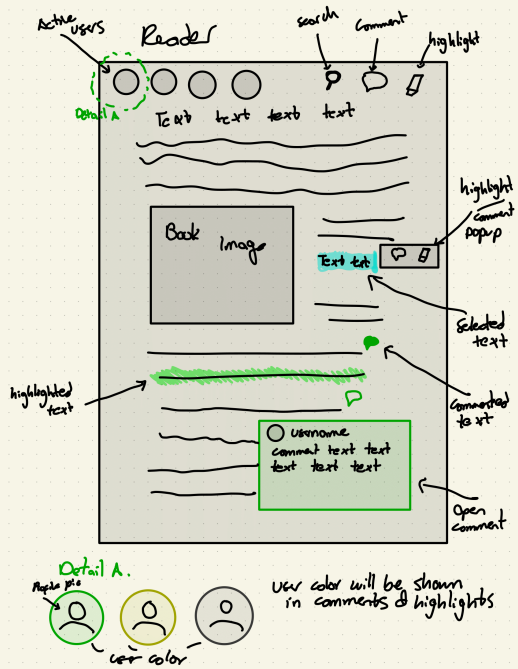
Una vez completa la ideación inicial por medio de *SCAMPER* y de dibujos, se prototipó por medio de *Figma*. Los cambios realizados a estos prototipos fueron ideados por medio de los aprendizajes de entrevistas y pruebas de usuario.



(a) Inicio de sesión.



(b) Biblioteca (*storefront*).



(c) Lector.

Figura 5: Dibujos de la fase de ideación

6.3.4. Prototipar

La elaboración de prototipos fue realizada por medio de *Figma* para diseños de pantallas y por medio de Angular para la validación final de funcionalidades y flujos del programa. Los prototipos iniciales fueron muy generales y con un despliegue de funcionalidades básico, para que por medio de las entrevistas de prueba, los usuarios tuvieran mayor libertad de expresar sus necesidades.

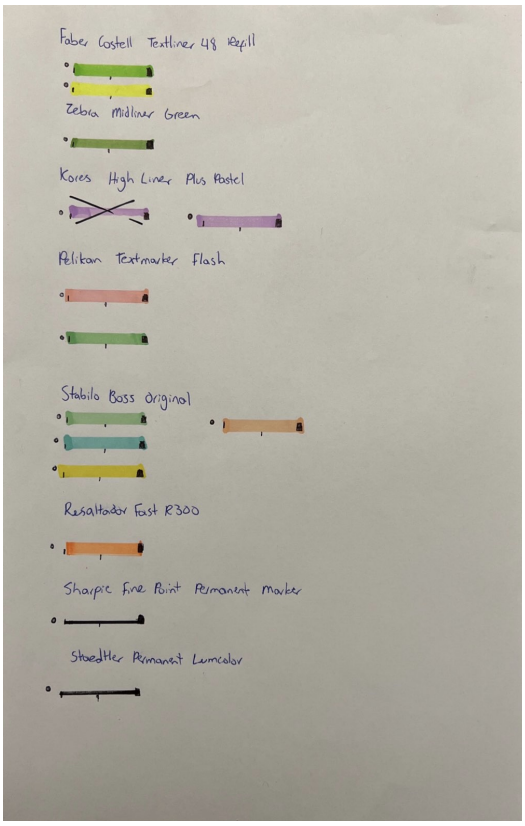
También se prototiparon distintas paletas de colores para ser utilizadas en el lector. En este caso, la paleta base de resaltado se limitó a 5 colores debido a que se tomó el grupo ideal de estudio como 4 miembros (Corrégé y Michinov, 2021). No obstante, se dio la flexibilidad de contar con 5 miembros que también es efectivo (David Eccles School of Business, 2015). Se definió esta paleta para idealmente realizar grupos de 4 estudiantes. Sin embargo, si se desea un grupo más grande también es factible bajo una paleta probada.

Prototipos de resaltado

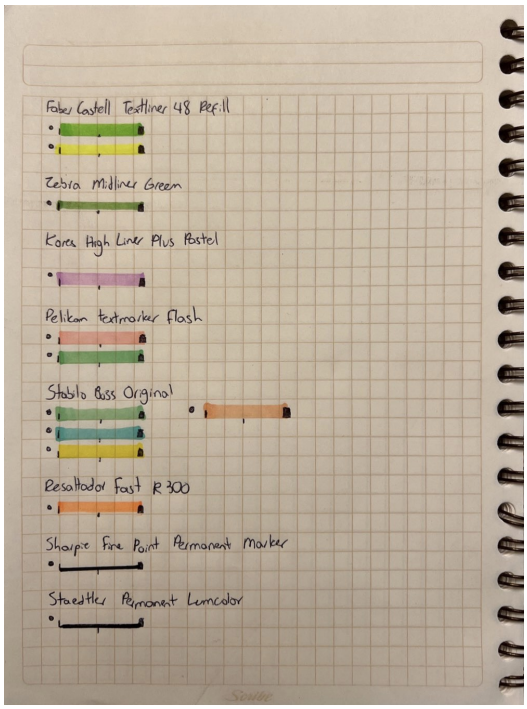
Para los prototipos de la fase de resaltado se exploró el comportamiento de resaltadores en papel real. Para ello se utilizaron 11 tipos de resaltadores. Estos resaltadores se encontraban distribuidos entre las marcas: *Faber Castell, Zebra, Kores, Pelikan, Stabilo Boss* y *Fast*. Además, se utilizaron 2 marcadores permanentes como control. Estos eran de las marcas *Sharpie* y *Staedtler* respectivamente.

Los resaltadores se probaron en 3 tipos de hoja diferentes: 1 hoja blanca tamaño oficio marca *International Paper Chamex*; 1 hoja de cuaderno *Scribe Ecológico* y 1 hoja de cuaderno *DaVinci Sketchbook*. Para ello se separó la hoja en 11 espacios, agrupados por marca. Cada espacio contaba con una medida de 3 centímetros la cual fue posteriormente resaltada con el resaltador indicado. A continuación, se midió el punto medio de cada resaltado (1.5 cm) y se escaneó la hoja en un ambiente de luz amarilla utilizando la aplicación *CamScanner*. Los resultados de este proceso pueden ser observados en la Figura 6.

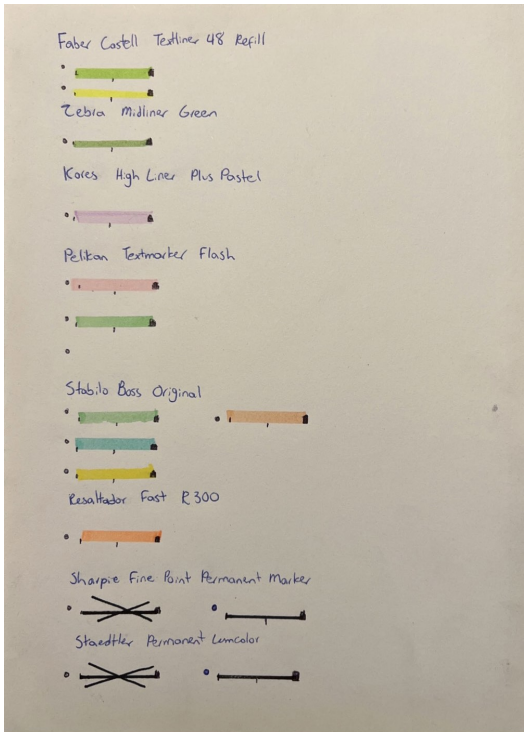
Posteriormente, estas mismas hojas fueron sometidas a una prueba estandarizada. Para esta prueba se utilizó un cuarto totalmente oscuro (para asegurarse de que no hubiera luz, se hizo la prueba de noche y en un sótano; además, se taparon las posibles entradas de luz al cuarto). Luego, se fijó un *iPad Pro 2020* a una altura de 30 cm sobre cada papel y se encendió su linterna a su máxima intensidad. Posteriormente, con un *iPhone 13 Pro* se tomó una fotografía con el lente 1x a cada hoja a una altura de 30 cm. Las imágenes resultantes pueden ser observadas en la Figura 7.



(a) International Paper Chamex.

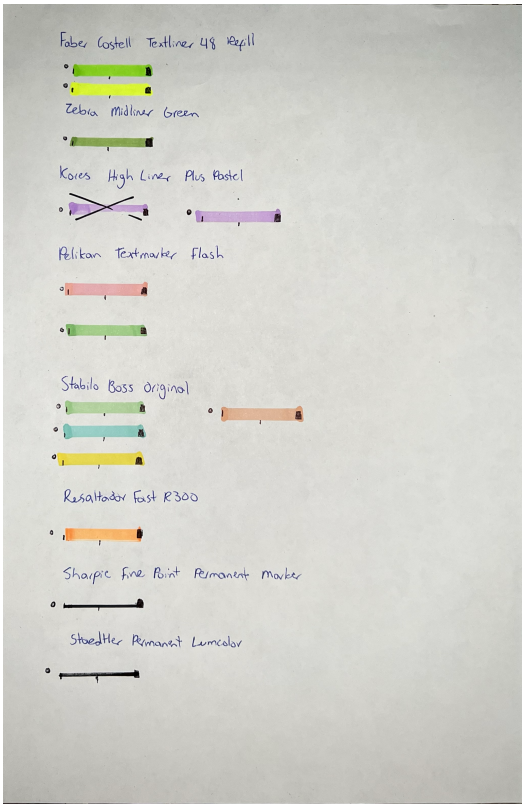


(b) Scribe Ecológico.

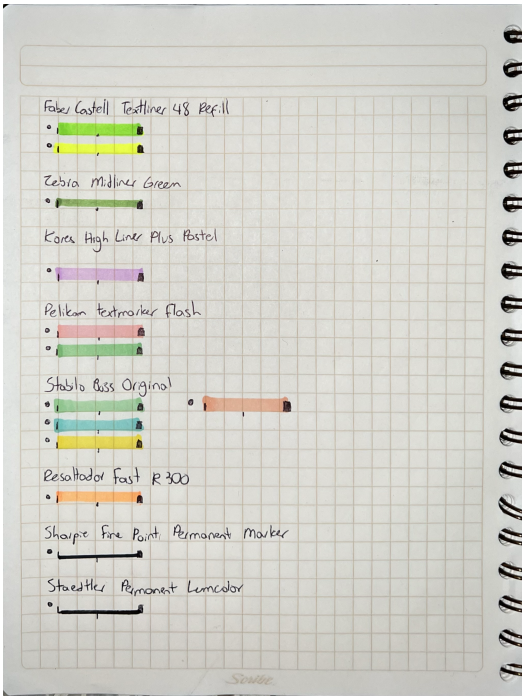


(c) DaVinci Sketchbook.

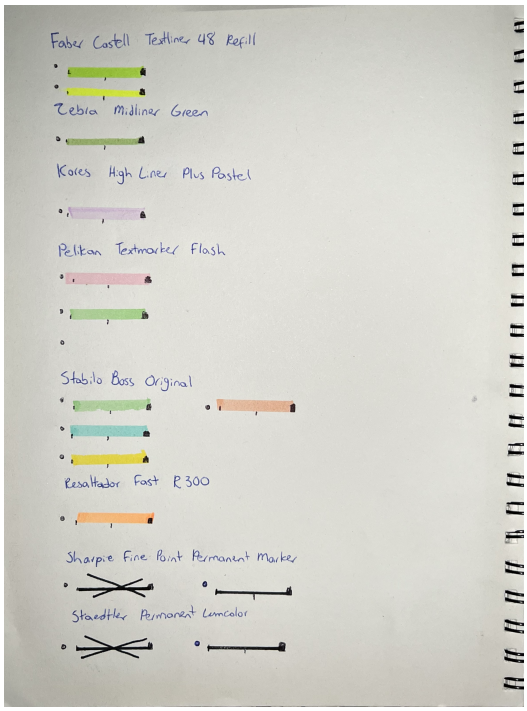
Figura 6: Pruebas de resaltadores bajo luz amarilla



(a) International Paper Chamex.



(b) Scribe Ecológico.



(c) DaVinci Sketchbook.

Figura 7: Resaltadores bajo prueba estandarizada

Una vez tomadas las imágenes, se midió los contrastes entre las hojas (fondo) y los resaltadores (pintados sobre la hoja) en los distintos tipos de condiciones. Estos resultados fueron tomados como base para la paleta de resaltado.

Prototipos iniciales de pantallas

Los prototipos iniciales de pantallas fueron elaborados en *Figma*, para poder mostrar la ubicación de elementos y diseños preliminares de componentes. Se realizaron 4 prototipos en esta herramienta. Conforme se fue obteniendo retroalimentación, estos prototipos se fueron refinando en esta herramienta.

El prototipo inicial consistió de 4 pantallas base: inicio de sesión, menú principal (biblioteca o *storefront*), búsqueda y lector. Estas pantallas tuvieron 4 iteraciones de prototipado. Estos prototipos pueden ser vistos en las figuras 8, 9, 10 y 11.

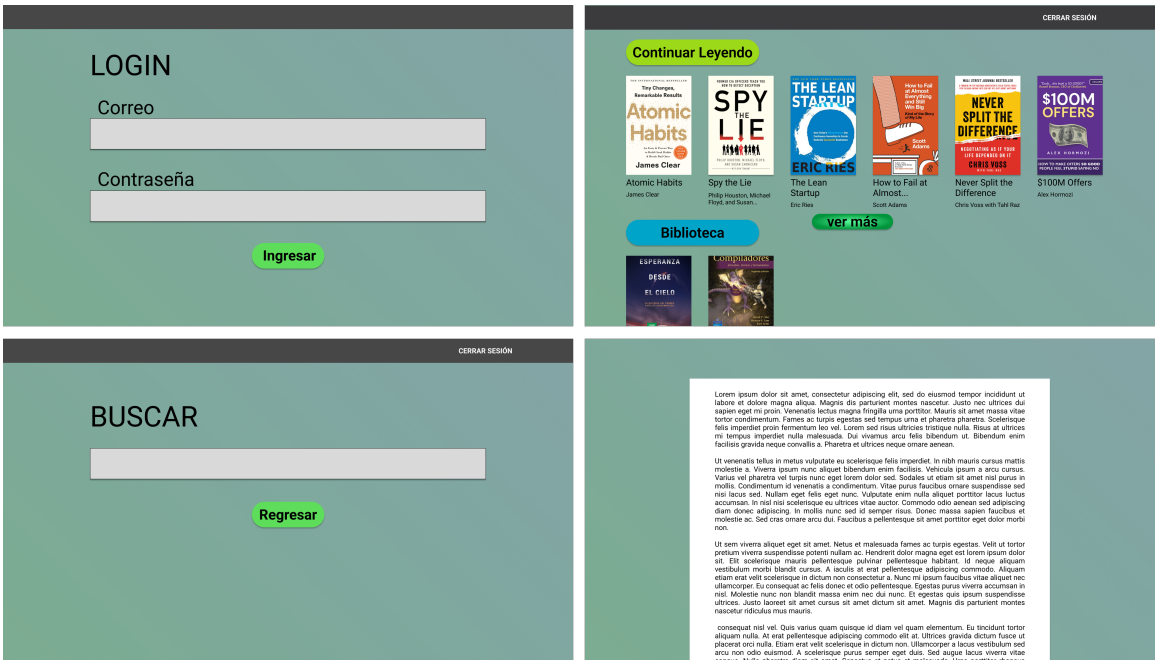


Figura 8: Prototipo #1

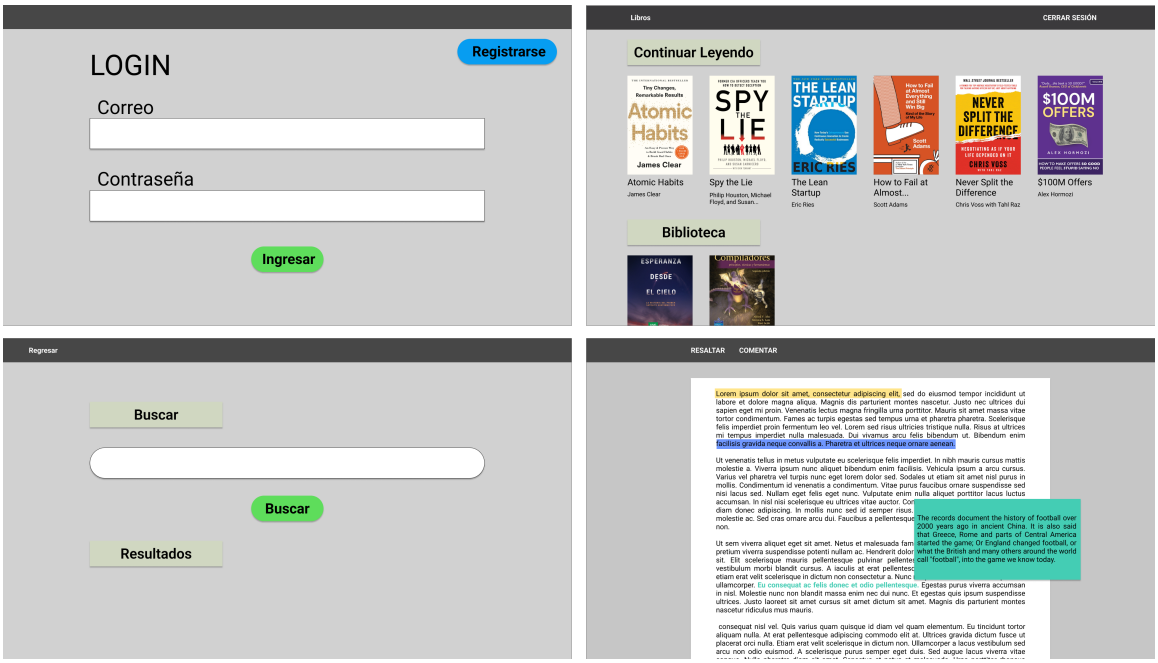


Figura 9: Prototipo #2

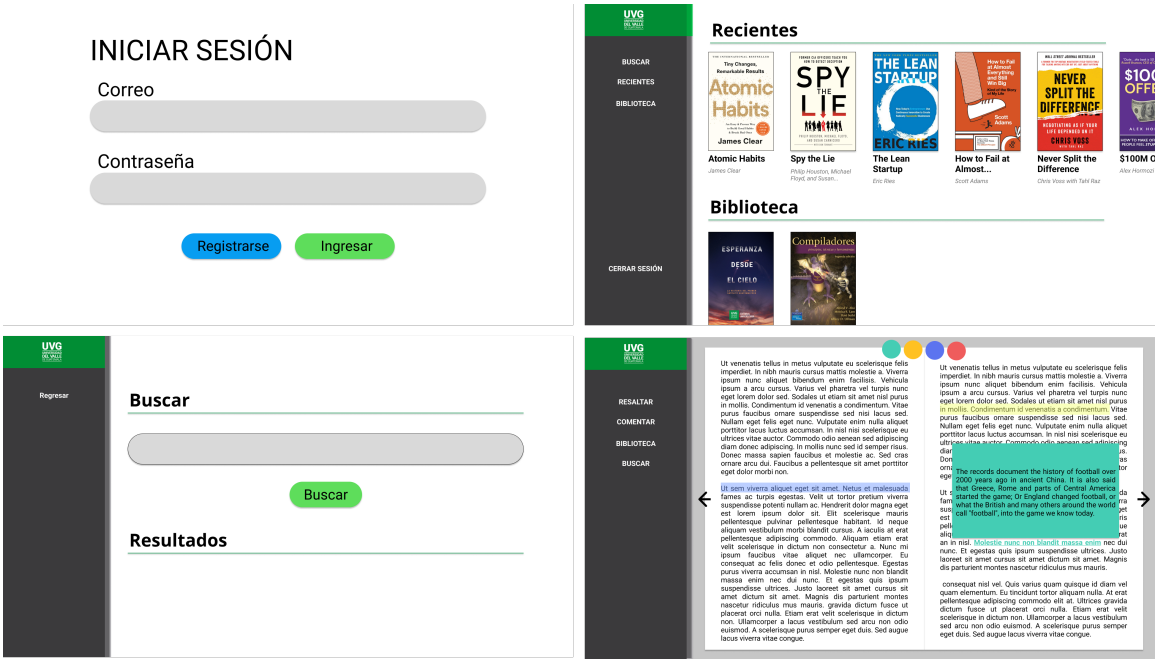


Figura 10: Prototipo #3

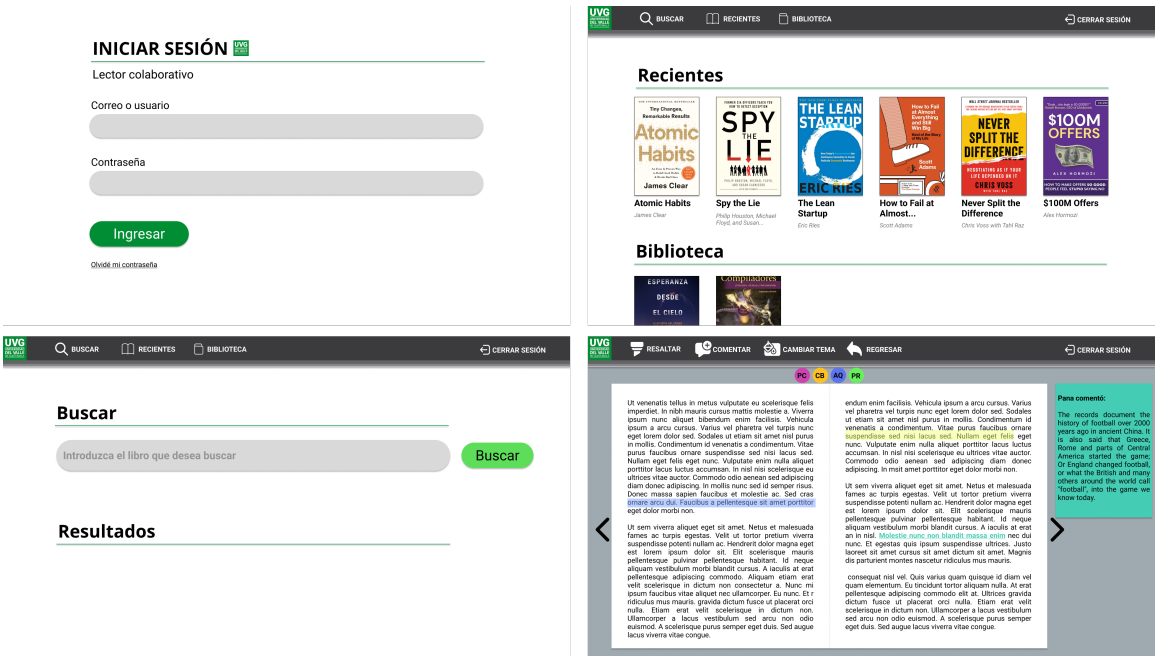


Figura 11: Prototipo #4

Prototipos programados de pantallas

Los prototipos programados se realizaron una vez hubo varias iteraciones de los prototipos iniciales. Estos fueron programados en *Angular*.

6.3.5. Probar

Entrevistas con prototipos en *Figma*

Una vez elaborados los prototipos en *Figma*, estos se pusieron a prueba mediante entrevistas. Estas fueron de carácter semi-estructurado, donde a cada entrevistado se le preguntó en cada pantalla (inicio de sesión, biblioteca, búsqueda y lector) su opinión sobre el diseño y las funcionalidades presentes. En caso de haber una funcionalidad importante se le preguntó al usuario acerca de cómo la realizaría. Las preguntas detalladas pueden observarse en el anexo 11.3

En total se hizo 20 entrevistas para las 4 iteraciones de prototipos, es decir, se hizo un total de 5 entrevistas por prototipo. Se estructuró de esta manera debido a que no existe un consenso establecido sobre cuántas entrevistas se deben hacer para llegar a un punto de saturación en un estudio cualitativo, es decir, a un punto donde se hayan descubierto los patrones principales de necesidades del usuario (Rosala, 2021). No obstante, los profesionales de experiencia de usuario, según Rosala (2021) recomiendan hacer entrevistas con 5 personas del grupo objetivo. Se buscó que el total de entrevistas para estos prototipos fuera de 20 para poder acercarse a un punto de saturación del 90 al 95 % tomando como base un estudio realizado sobre entrevistas en marketing (Griffin y Hauser, 1993).

El criterio de selección de estudiantes para estas entrevistas fue que las personas entrevistadas no hubieran participado en la fase de empatía (a excepción de las encuestas preliminares). Además, se contempló únicamente a aquellas personas pertenecientes a la comunidad UVG que fueran estudiantes activos del campus central. También se limitó la participación de estudiantes en Ingeniería en Ciencia de la Computación y Tecnologías de la Información a 1 por cada grupo de 5 estudiantes, para contar con una mayor variedad de opiniones.

Pruebas de elección

Se diseñaron estas pruebas como una versión simplificada de las pruebas A/B para comprobar que los prototipos en *Figma* tuvieran un avance de acuerdo a las necesidades de los usuarios. Para ello, se hizo 3 grupos de prototipos: el prototipo 1 y 2; el 2 y 3, y el 3 y 4 y se entrevistó a estudiantes para ver cuál era su preferido. La entrevista proseguía de la siguiente manera: se le mostraba 1 pantalla al estudiante, donde de un lado se encontraba una versión del prototipo y del otro lado se mostraba otra versión de la misma pantalla. Luego, el entrevistado debía votar por su pantalla favorita. Esto se hizo para las pantallas de: inicio de sesión, menú principal (biblioteca o *storefront*), búsqueda y lector. También se le preguntó al entrevistado acerca de sus preferencias en las pantallas, sin embargo, esto no se

hizo con preguntas semi-estructuradas debido a que este proceso se realizó en las entrevistas de prototipos, no en las pruebas de elección.

El criterio de selección para estas pruebas fue únicamente que el entrevistado no hubiera participado en ninguna entrevista anterior.

Grupo de pruebas

Además de las entrevistas se realizó un grupo de pruebas. Este se seleccionó de la siguiente manera: 2 estudiantes de ingeniería mecatrónica, 1 estudiante de ingeniería electrónica, 1 estudiante de diseño de producto (graduado), 1 estudiante de ingeniería civil arquitectónica, 1 estudiante de psicología. Todos estos estudiantes fueron seleccionados por ser estudiantes activos o graduados de la Universidad del Valle de Guatemala y se buscó diversidad de carreras en el grupo.

Este grupo de pruebas fue realizado de forma remota por medio de conversaciones de *WhatsApp* para validar de forma rápida diseños de pantallas y detalles de las mismas. Este se utilizó para hacer votaciones sobre diseños, específicamente:

- La forma de visualizar libros (carrusel o crecimiento hacia abajo).
- El color de fondo del lector.
- El menú de navegación (debido a que se obtuvo respuestas mixtas en las entrevistas sobre este).

Para realizar la votación, a cada persona del grupo se le enviaron imágenes de las opciones para decidir por su favorita. También, en caso de necesitar profundizar más sobre algún diseño (como en el caso del menú de navegación), se le preguntó a cada persona sus preferencias. Si se debía validar algún diseño más a profundidad, se les preguntó de forma más abierta. Cabe mencionar que las preguntas y las votaciones fueron realizadas por conversaciones privadas, para evitar que las demás personas del grupo pudieran ver los votos y las opiniones de los demás miembros. De hecho, las personas dentro del grupo de pruebas no conocían la identidad de las demás personas del grupo ni tuvieron interacción entre sí.

Pruebas de usabilidad

Una vez terminadas las pruebas en *Figma*, se procedió a hacer pruebas de usabilidad con el prototipo programado. Para ello, se utilizó una computadora tipo *laptop Acer Predator Helios 300* de 15.6 pulgadas y se le pidió a los usuarios realizar distintas acciones en el lector. Se probaron las siguientes acciones:

1. Iniciar sesión
2. Ir a la biblioteca

3. Ir a recientes
4. Buscar un libro
5. Entrar a un libro
6. Resaltar un texto
7. Comentar un pasaje
8. Mostrar comentarios
9. Esconder comentarios
10. Buscar un pasaje en un libro
11. Cambiar de página
12. Cambiar el estilo del libro
13. Regresar al menú desde un libro
14. Cerrar sesión

Se midió el tiempo para realizar estas acciones. Una vez realizadas, si el estudiante así deseaba, se le dejó utilizar la aplicación libremente. Al finalizar el proceso se le preguntó acerca de su experiencia y si modificaría, agregaría o eliminaría algo de la aplicación.

Este proceso se realizó con 5 estudiantes del campus central de la Universidad del Valle de Guatemala y 1 persona no activa como estudiante superior a los 50 años de edad, para simular un caso atípico de uso. El único requisito para ser entrevistado fue no haber participado en ninguna de las fases anteriores. Esto se hizo así para poder capturar una mayor variedad de opiniones.

6.3.6. Total de personas involucradas en el proceso de *design thinking*

En total se involucró a 62 personas en el diseño del lector. Este cálculo no incluye a las personas que participaron en las encuestas debido a que no se excluyeron a estas de las entrevistas posteriores. Debido a esto, se estima que el número final de personas es mayor, mas no se puede garantizar. Las personas involucradas se encuentran distribuidas de la siguiente manera:

- 10 estudiantes entrevistados inicialmente
- 5 catedráticos entrevistados inicialmente
- 20 estudiantes entrevistados para prototipos en Figma
- 15 estudiantes involucrados en pruebas de elección
- 6 personas involucradas en pruebas de usabilidad
- 6 personas involucradas en el grupo de pruebas

Característica	*Laravel*	*Express.js*	*Nest.js*	*Django*
Lenguaje	PHP	Javascript	Typescript	Python
Arquitectura	MVC	Ninguna	MVC	MVT
Seguridad	Muy seguro	Depende	Muy seguro	Seguro
Escalabilidad	Escalable	Escalable	Escalable	Muy escalable
Costo de cambio	Alto	Medio	Bajo	Medio
Comunidad	Grande	Muy grande	Mediana	Grande
Velocidad	Lento	Muy rápido	Rápido	Medio

Cuadro 3: Comparativa de *frameworks* populares. Este cuadro corresponde al estudio preliminar realizado para entender bien qué *framework* se adaptaba mejor a las necesidades de los usuarios y del programa, así como cuál se adaptaba mejor al equipo que lo estaba trabajando. Se trataron de tomar en cuenta tantos lenguajes y *frameworks* como fuesen posibles para permitir una decisión informada. *Frameworks* basados en *Javascript* fueron los más populares, mientras que los que utilizaban *Rust* eran los más escasos (incluso si su comunidad es muy fuerte). Los cuatro *frameworks* más populares o fuertes son mostrados, de siete finalistas en total. **Importante:** *Nest.js* es un *wrapper* de *Express.js*, pero se compararon por las distintas filosofías que estos poseen.

6.4. *Framework* de *backend*

Los programas usualmente son lo suficientemente modulares como para hacer que varias entidades trabajen en ellos al mismo tiempo, comunicándose por medio de interfases. Sin embargo, gracias a que este trabajo tomaba en cuenta las necesidades de los estudiantes, se tuvo que tomar en cuenta cuáles eran las necesidades de cada uno. Muchos de los estudiantes en la universidad objetiva tienen acceso a dispositivos móviles, cosa que no puede ser común en otros entornos universitarios. En específico, la Universidad del Valle de Guatemala tiene una población socio-económica de más alto nivel que otras universidades.

Gracias al alcance del proyecto, y gracias a que muchos de los documentos se terminarían utilizando en los contextos comunes de la universidad, se terminó decidiendo por una arquitectura modular basada en el *frontend* y el *backend* de un programa más grande. Como fue mencionado con anterioridad, se terminó utilizando *Nest.js* para el desarrollo de la funcionalidad de *backend*, gracias a su gran modularidad y bajos costos de cambio. Asimismo, se tuvo en mente que la universidad escogida no es la regla, sino la excepción a la misma. Todos los procesos están hechos para que sea sencillo el cambiarlos sin mayor problema.

El *backend* es uno de los temas más populares que existen para su discusión, gracias a que muchos *frameworks* terminan teniendo un sin fin de filosofías sobre cómo debería de estar todo organizado. Asimismo, existen una gran cantidad de lenguajes en los que un *framework* puede estar basado, todos teniendo ciertas ventajas y desventajas. Para poder escoger un lenguaje y un *framework*, se hizo una comparativa de varios lenguajes y arquitecturas, tratando de encontrar cuál era el mejor para el proyecto. Los lenguajes variaban desde aquellos bien conocidos por las tecnologías web, como lo es *Javascript*; hasta aquellos que se consideran más como lenguajes para programación de sistemas, como *Rust* con *WASM*. Esta comparativa, demasiado extensa para el presente documento, puede ser encontrada en el Cuadro 3.

Es imposible decir que un *framework* es perfecto. Los casos de uso que cada uno de estos poseen son variados, y pueden depender incluso del programador que los maneja. Exacta-

mente todos los *frameworks* mencionados tienen ventajas y desventajas bastante grandes, pero se trataron de estudiar las características más importantes para una población de estudiantes en un estrato social medio, o medio-alto para poder asegurar que las características escogidas eran las más fortuitas.

6.4.1. Lenguaje

El lenguaje es un tema importante por varios aspectos. Primero que todo, un lenguaje puede definir enormemente la arquitectura de una aplicación, creando o eliminando problemas para los programadores. Gracias a que los lenguajes son los temas más básicos para un proyecto, esta fue la primera característica pensada, pero no necesariamente la más importante. Los programadores que programarían todo también tenían preferencias, por lo que la exploración de cada filosofía era de extrema importancia.

PHP

Laravel es un *framework* de gran popularidad basado en *PHP*. Este lenguaje, desarrollado en 1994, es un lenguaje diseñado para que se pueda correr en cualquier navegador. Esto lo hizo muy popular al principio entre los programadores del proyecto, pero lentamente fue desechado por características importantes que no satisfacía. *PHP* es un lenguaje extremadamente flexible, independiente de qué plataforma se utilice para correrlo, con una curva de aprendizaje bastante modesta. Su vida media es bastante larga, dándole soporte por mucho más tiempo por versión que otros lenguajes. Esto permite que su utilización se base mucho en módulos y librerías, ya que estas pueden adaptarse a la versión actual. Sin embargo, *PHP* **no es considerado muy seguro** (Coelho, 2016). La mayoría de sus problemas pueden verse afectados por una gran cantidad de *bugs* existentes dentro de su código. Por esto, aunque *PHP* era una opción popular, la seguridad del mismo era preocupante.

Javascript

Javascript es el lenguaje utilizado en *Express.js*. Este es un lenguaje de extrema popularidad con especialización en el desarrollo de páginas web y la lógica que permite que estas evolucionen. Al momento de escribir este documento, *Javascript* es utilizado en el 98 % de los sitios web en todo el internet cuando se habla del lado del cliente. Gracias a que no es un lenguaje **tipado**, su aprendizaje es rápido y fácil. Por desgracia, gracias a que los navegadores actuales aceptan *Javascript* en virtualmente cualquier lado, estos fueron vulnerables a una gran cantidad de problemas de seguridad, tal como los famosos *Cross-Site Vulnerability Attacks*. Es difícil negar el poder que tiene *Javascript* en todo lo que es tecnología web (Eich, 2008). Gracias a que los navegadores van cambiando constantemente, los distintos problemas de seguridad han ido desapareciendo, siendo reemplazados por maneras más creativas de *hacking*. Por su naturaleza sencilla, gran manejo de información web, y naturaleza concurrente, *Javascript* fue uno de los lenguajes más populares entre los programadores de tanto *frontend* y *backend*.

Typescript

El *bias* implícito de los programadores no se comenzó a mostrar hasta que se comenzó a hablar de *Typescript*. Este lenguaje es posiblemente una de las opciones más populares para el desarrollo de aplicaciones cuando se esté tratando de desarrollar algo a base de *Javascript*. *Typescript* utilizado en *Nest.js* no es nada más que un *wrapper* moderno de *Javascript* que permite el añadir tipos a un programa durante la fase de desarrollo, evitando así problemas humanos durante la fase de compilación. Los tipos de *Typescript* son extremadamente estrictos, por lo que es muy difícil hacer que el programador cometa errores constantemente. Asimismo, *Typescript* permite el uso de lo que son tipos personales, así como interfaces para la comunicación entre componentes que siguen metodologías *SOLID*, como fueron descritas en la sección 5.5.1. *Typescript* permite un desarrollo mucho más rápido, con muchos menos errores por parte del programador. Gracias a que es un lenguaje seguro y tipado, es fácil ver dónde hay errores de lógica. Este fue el lenguaje más popular entre los programadores.

Python

Python es probablemente uno de los lenguajes más conocidos y populares entre programadores. Es muy difícil no admitir qué tan popular es para hacer exactamente todo (que puede ser considerado una de sus debilidades más grandes). Este lenguaje, al igual que *Javascript*, es interpretado en vez de compilado, por lo que la velocidad de programación es muy alta. Es extremadamente fácil de aprender por su sintaxis fácil y concisa, haciéndolo un lenguaje popular entre programadores de cualquier ámbito. El *framework* que lo representaba es *Django*, que busca ser un *framework* de fácil aprendizaje y alta escalabilidad. Es importante notar que *Python* no solo no tiene tipos, sino que también puede ser muy lento, a comparación de otros lenguajes. Sin embargo, gracias a las optimizaciones de *Django*, este termina siendo considerablemente más rápido que otros lenguajes en esta lista. Por desgracia, este lenguaje puede ser difícil de mantener, gracias a que se basa completamente en el *whitespace* para sus ámbitos, lo que no fue una elección popular ante el programador de *backend*.

6.5. Arquitectura

La arquitectura de un proyecto es el tema más importante cuando se debe de hablar de escalabilidad del mismo. Mientras que ciertos lenguajes poseen suficiente flexibilidad para poder subsistir con cualquier arquitectura, la verdad es que cada *framework* tiene, no solo su propia arquitectura, sino también su propia filosofía sobre cómo cada cosa debe de ser añadida a esta. Como se puede apreciar en el Cuadro 3, más de la mitad de los *frameworks* tienen ya una arquitectura definida sobre cómo deberían estos de ser utilizados. Asimismo, se puede observar que las arquitecturas más populares son *MVC*, seguido por *MVT* y *Express.js*, que no tiene ninguna arquitectura en específico.

La arquitectura *MVC*, descrita por vez primera en la sección 5.5.2, es una de las arquitecturas más populares en el desarrollo de aplicaciones y proyectos masivos por su fácil escalabilidad y bajo costo de cambio. Asimismo, vale aclarar que la arquitectura *MVT* puede

ser vista como una variación de la arquitectura *MVC*. Este modelo trabaja por medio de capas lógicas que permiten un gran nivel de escalabilidad y separación de responsabilidades. De entre todas las arquitecturas estudiadas, *MVC* posiblemente sea la más escalable de todas.

A final de cuentas, los *frameworks* más populares eran aquellos que terminaban siendo los más escalables. Por consecuencia, la mayoría tenían, de por sí, una arquitectura similar a los demás. Por esto, la arquitectura terminó siendo un punto nulo entre los *frameworks* más populares, ya que todos se parecían entre sí. Este fue el punto que separó mucho a los *frameworks* mejor calificados con los peor calificados.

6.5.1. Seguridad

Difícilmente existe un tema a discutir más popular que el de la seguridad en los sistemas de computación. Los sistemas que no tengan la suficiente seguridad son mal vistos y no logran llegar lejos en la industria. Asimismo, los *frameworks* que no tengan la suficiente seguridad, fueron descartados sin pensarlo más. Todos, excepto uno: *PHP*. Este lenguaje es un caso interesante, gracias a que el lenguaje es considerado inseguro, pero *Laravel* es considerado un *framework* bastante seguro. *Laravel* implementa librerías de seguridad populares, tal como los **pasaportes** y los ***JWT*** de manera nativa. Solamente esta característica hizo que muchos de los problemas anteriormente descubiertos se ignoraran, gracias a que tener seguridad de la información implementada es un *plus* muy difícil de ignorar.

Nest.js implementa ambas librerías mencionadas anteriormente, con la ventaja que son extremadamente modulares. La modularidad de ambas librerías permite que puedan ser fácilmente cambiadas, dada la necesidad de hacerlo. Muchos de los *frameworks* explorados no tenían seguridad implementada de manera nativa. El más popular siendo *Express.js*. Este último no implementa nada gracias a que se enfoca más que todo en ser considerado *lightweight* y le da todas las responsabilidades al programador. Por el otro lado, *Django* es bastante seguro a nivel de ataques, pero pueden haber problemas por la falta de tipos en *Python*, llevando a *bugs* no descubiertos hasta mucho tiempo después.

Gracias a que *Nest.js* implementa tantas librerías de seguridad, está escrito en *Javascript* y ya era conocido previamente, este fue escogido también como ganador de esta característica. Su estructura modular ayuda también a encontrar problemas en la programación del mismo, ayudado por *Typescript* y sus interfaces.

6.5.2. Escalabilidad

No existe ningún aspecto más importante para temas de este proyecto que la escalabilidad de una aplicación. Para poder entender por qué esto es tan importante, se utilizó el libro de *Robert C. Martin*: *Clean Architecture*. En la Figura 12 (página 73) se puede observar el crecimiento constante de ingenieros para dar mantenimiento a una aplicación a lo largo de su vida útil. Este crecimiento no es ninguna sorpresa, gracias a que muchos programas terminan siendo de gran tamaño y de gran dificultad para mantener. Sin embargo, el problema puede comenzarse a ver en la Figura 13 (página 74). Dado el mismo periodo de tiempo, la cantidad

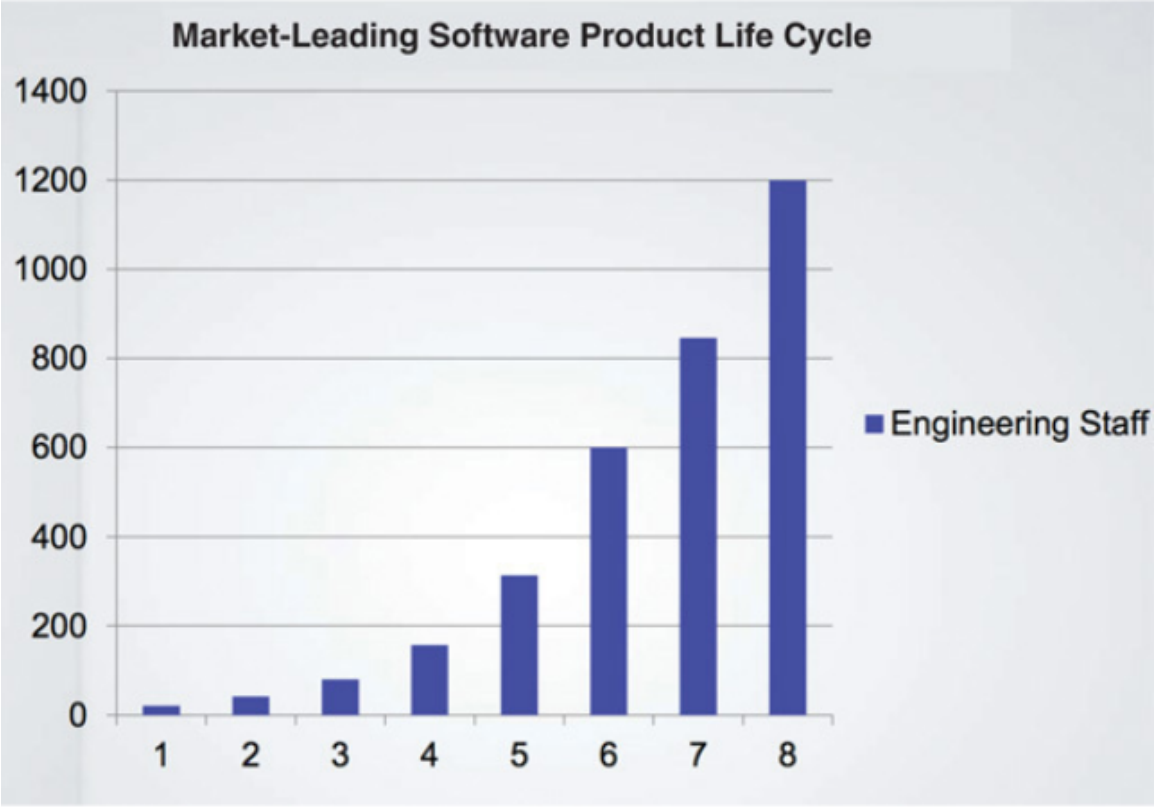


Figura 12: Crecimiento gradual de personal en una compañía de software grande

de empleados creció considerablemente, alcanzando un 1200 % en ocho ciclos. Sin embargo, este crecimiento no puede ser apreciado en la segunda gráfica, en donde la productividad parece haberse apenas duplicado en el mismo periodo de tiempo. Esto es un problema de **arquitectura** y de **escalabilidad**. Por esto, la escalabilidad fue una de las características más importantes cuando se estaba tratando de escoger un *framework*.

Django es el rey de los *frameworks* escalables, teniendo a *Nest.js* y a *Laravel* bastante cerca. *Express.js* es un caso interesante, ya que solo es escalable en aquellos casos en los que el programador de por sí sabe lo que está haciendo. Cualquiera de los tres *frameworks* más populares puede ser utilizado en este sentido. Todos ellos utilizan arquitecturas similares, por lo que la escalabilidad es bastante alta en todos. Por las razones anteriormente descritas en temas de **seguridad** y **lenguaje**, el *bias* implícito hizo que *Nest.js* ganara este tema también.

6.5.3. Costo de cambio

No todo tiene que ver solamente con qué tan fácil de implementar es algo para el programador. Muchos de los temas explorados se tuvieron que hacer desde el punto de vista de una empresa, o una entidad que observa gastos y busca retornos. El costo de cambio, por ende, es uno de los temas que se exploraron a gran detalle. Este depende de la escalabilidad y el soporte que disfruta un *framework* y su respectivo lenguaje. Esto último se debe a que

Figura 13: Evolución gradual de productividad de una compañía de software grande

distintos programadores prefieren distintos tipos de lenguajes, por lo que los costos pueden ser vistos como una variable dependiente de la popularidad de ambos.

De entre todos los *frameworks*, aquí es donde falla principalmente *Laravel*. Este *framework* goza de una comunidad activa, pero *PHP* es visto como un lenguaje viejo y con características faltantes. Por el otro lado, la popularidad de otros lenguajes, tales como *Python* y *Javascript*, ha hecho que sus respectivos *frameworks* tengan comunidades más grandes y activas. No es difícil encontrar desarrolladores que trabajen con estos lenguajes, por lo que tampoco es difícil encontrar documentación sobre cualquier problema que se tenga mientras se está trabajando en uno de estos lenguajes. Asimismo, gracias a que *Typescript* es un *wrapper* de *Javascript*, este sufre la popularidad de este de una manera interesante, ya que la documentación de *Javascript* puede ser utilizada para resolver problemas con *Typescript*.

6.5.4. Comunidad

Como fue ya anteriormente mencionado en la sección 6.5.3, un *framework* puede sufrir de altos costos de cambio si este no tiene una comunidad activa que permita encontrar problemas en común, y cómo resolverlos. Esto mismo puede pasar si una comunidad es grande, pero muy desorganizada y, por ende, no pueda trabajar en conjunto. La comunidad más grande es la de *Express.js* gracias a la fácil utilización de este. Muchos programadores no permiten que otros *frameworks* les digan qué hacer o cómo programar, por lo que prefieren irse por aquello que no tenga opiniones de por sí. Sin embargo, debemos de recordar que *Nest.js* es un *wrapper* de *Express.js*, por lo que, aunque este goza de una comunidad mediana, también goza de la comunidad de *Express.js* para resolver dudas. Asimismo, la documentación de *Nest.js* fue tomada como una de las documentaciones mejor hechas de

entre todos los *frameworks*. Las comunidades de *Javascript* y de *Typescript* son de tamaños bastante respetables, teniendo estas las mismas ventajas que *Nest.js* y *Express.js*.

El tamaño de los demás *frameworks* es respetable, pero existen algunos problemas dentro de ellos. Primero que todo, se debe de mencionar que la comunidad de *Django* es de un tamaño bastante notable, pero esta termina siendo fragmentada por discusiones generales de *PIP* o paquetes individuales. Esto puede ser ventajoso en el sentido de una arquitectura basada en la composición, pero depende mucho del proyecto desarrollado. Por el otro lado, *Laravel* sufre mucho por su comunidad. En específico, es tan grande, que resulta siendo un poco desorganizada, por lo que muchos de los patrones discutidos pueden llegar incluso a ser contradictorios entre sí.

6.5.5. Velocidad

Por último, la velocidad del *framework* y del *lenguaje* son temas muy recurrentes en los contextos de costo y de tecnologías web. Esto se debe a que un *framework* no solo debe de ser rápido en sus operaciones, sino también debe de saber servir a varias solicitudes al mismo tiempo. El lenguaje que mejor maneja esto es posiblemente *Javascript* por su naturaleza concurrente. *Nest.js* tampoco se queda atrás, gracias a que su arquitectura modular permite que muchas cosas sean trabajadas por medio de servicios y caché.

La mayoría de los *frameworks* discutidos son rápidos, con la excepción de *Laravel*. *Express.js* tiene gran velocidad por su falta de librerías instaladas y naturaleza *lightweight*. *Nest.js* es rápido gracias a optimizaciones a su código, una arquitectura favorable y un *pipeline* extremadamente optimizado y escalable. *Django* es de velocidad media gracias a que *Python* es un lenguaje relativamente lento a comparación de los demás lenguajes, pero también goza de una gran cantidad de optimizaciones, aumentando enormemente su velocidad.

6.6. Utilización de Nest.js

No es difícil ver por qué es que se terminó escogiendo a *Nest.js* como *framework*, dadas todas las características ventajosas que este proporciona tanto al programador como a cualquier entidad que tenga que pagar los costos. La combinación entre una fuerte seguridad, alta escalabilidad, gran comunidad y fácil aprendizaje hizo que este se volviera el *framework* pro defecto del proyecto. Asimismo, el módulo de *frontend* terminó escogiendo el *framework* de *Angular*, cuya arquitectura y forma de programación se asemeja mucho a *Nest.js*. Las filosofías similares permiten la fácil comunicación entre los dos *frameworks*, así como el dejar que los programadores de *frontend* puedan revisar el código de *backend* para entender mejor la implementación de los distintos métodos. El uso de *Nest.js* terminó creando distintos **submódulos** que permitieron dividir el trabajo en varias partes distintas. En conjunto con *Nest.js*, se terminó creando un micro-servicio singular que interactúa con la aplicación como un cliente, basado en *Socket.IO*.

Nombre	Responsabilidad principal
Auth	**Manejo de usuarios fuera de la aplicación.** Permite la creación de sesiones.
Document	**Manejo de documentos y la lógica que estos necesitan.** Maneja instancias y almacenamiento de estos
User	**Manejo jerárquico de usuarios en el programa.** Busca mantener los niveles necesarios de Autenticación y Autorización
Group	**Manejo de grupos.** Maneja usuarios e instancias de documentos para que los documentos sean "únicos" por grupo

6.6.1. Módulos de Nest.js

Nest.js tiene una ventaja que no puede ser ignorada: la creación de módulos por cada responsabilidad que el programa maneja. Varios problemas seguían apareciendo durante la fase de diseño del *backend*, teniendo todos que ver con las responsabilidades que cada uno de los controladores manejaban. Por esto, se terminó creando una gran cantidad de módulos para facilitar el manejo de la información.

A final de cuentas, se puede decir que un módulo de *Nest.js* es una abstracción de una acción a tomar. Todos los módulos de *Nest.js* fueron diseñados con esto en mente, permitiendo hacer que la lógica de cada uno terminara en varios lugares del programa. El trabajar con módulos no solo permite abstraer las responsabilidades, sino que también facilita la programación de todo lo demás. Son mucho más fáciles de pensar e implementar, y ayudan a decrementar los tamaños de los distintos archivos del programa.

Los módulos también permiten la creación de servicios que decrementan la cantidad del código duplicado. Todo esto no sería tan sencillo de implementar si no fuese por el uso de las interfaces proporcionadas por *Typescript*. Es importante saber también que los servicios fueron diseñados para mantener solo lógica que tienen que ver con ellos mismos, respetando la división de responsabilidades tanto como se pueda.

A continuación se detallan los módulos **más importantes para aplicación**, pero no todos.

6.6.2. Módulo de *Auth*

El Módulo de *Auth* fue diseñado para ser el módulo que permite la entrada a la aplicación. Por su naturaleza, es muy simple y fácil de entender. A grandes rasgos, fue inicialmente diseñado para manejar cualquier tipo de petición que no tuviera ningún tipo de autenticación. Su uso es extremadamente extensible, gracias a que se utilizan repositorios por medio de *TypeOR* (descrito en la sección 6.8.2). Sus dos funciones más importantes son las del *login* y la del registro. Cada una de estas funciones utiliza *DTOs* para poder protegerse a sí mismas, como lo es expuesto en la sección 6.9. El uso de los mismos no solo termina abstrayendo la seguridad que estos proporcionan, sino también hacen que los cambios a los cuerpos de las peticiones puedan ser revisados en tiempo de corrida y que todos sean hechos

en un solo lugar, en vez de varios. El módulo está protegido de ataques de mala fe por estos. Asimismo, este módulo revisa que todos los datos sean correctos para evitar que las partes más vulnerables del programa sean expuestas a problemas de seguridad.

6.6.3. Módulo de *Document*

El módulo de *document* es un módulo que permite el manejo efectivo de documentos dentro de la base de datos y los sistemas de archivos de *AWS* (u otro servicio). En específico, este módulo permite la creación, el almacenamiento, el envío y el acceso autorizado de los distintos archivos dentro del sistema. De por sí, el módulo permite el funcionamiento del módulo de *Machine Learning* del proyecto gracias a que este también permite acceso a estadísticas de sesiones de los documentos. Todas las solicitudes dentro del módulo son *REST*, por lo que pueden ser descritas como que no tienen un estado predefinido. Sin embargo, este módulo maneja ciertas sesiones de *Sockets* para el manejo de las instancias de los documentos. El módulo de documentos también cuenta con varios servicios que le permiten al *frontend* encontrar los documentos de un estudiante, así como guardar los comentarios que estos hacen en instancias de documentos (que pueden ser compartidas o singulares). Es importante notar que este módulo, por motivos de seguridad, solo permite que se hagan peticiones de información descritas en el código, evitando así ataques de *SQL Injection*.

6.6.4. Módulo de *User*

El módulo de *User* es uno de los módulos más utilizados en el programa. Este permite el manejo jerárquico de los distintos tipos de usuarios, permitiendo saber cómo estos tienen permitido interactuar con el portal. Otro módulo dentro del programa, llamado el módulo de *Roles* interactúa mucho con este módulo, tratando de permitir que los usuarios tengan roles específicos, en vez de combinaciones de permisos. Sin embargo, los usuarios pueden tener permisos por separado, permitiendo así que cada uno de estos tengan roles más dinámicos. Sin embargo, por temas de espacio, es recomendado que los roles sean la regla, en vez de la excepción. A grandes rasgos, este módulo también permite obtener la información principal de un usuario, sea este un estudiante, un maestro, un *superuser*, u otro. Todos los procesos que necesiten de un usuario se van a este módulo y a su servicio por temas de granularidad de responsabilidades.

El módulo de *User* está optimizado para hacer consultas rápidas, teniendo la libertad de añadir cosas con naturaleza *NoSQL* a las tablas que los usuarios manejan. Asimismo, es posiblemente la tabla más utilizada entre todas, por lo que tiene que estar optimizada para una gran cantidad de consultas.

6.6.5. Módulo de *Group*

El módulo de *Group* es un módulo relativamente simple. Básicamente, este módulo busca implementar funciones o servicios que sirvan para asistir a los módulos de *User* y de *Document* y a las relaciones creadas entre ellos. Esto fue hecho así por el diseño escogido que sigue las filosofías de *SOLID*. Los módulos de *User* y *Documents* estaban haciendo

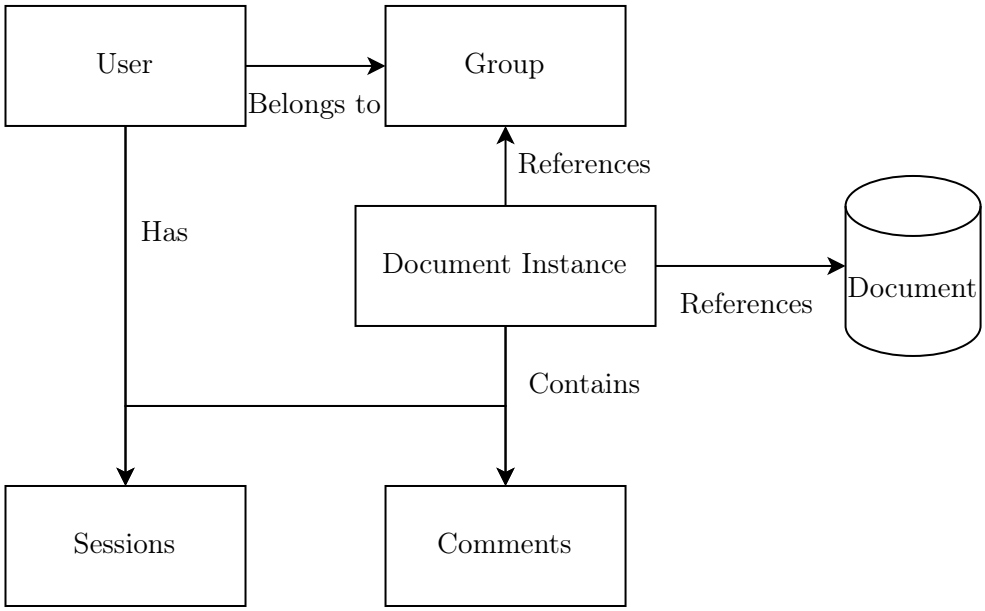


Figura 14: Instancias de Documentos. Las instancias de documentos son fragmentos de información que relaciona grupos de usuarios con documentos, permitiendo la creación de comentarios y sesiones para cada usuario. La creación de este tipo de tabla, gracias a la utilización de un *UUID* en su identificación, permite la creación de grupos en *Socket.IO*.

demasiadas cosas entre sí; tenían demasiadas responsabilidades y fallaban en modularidad por esto. El uso de este módulo también permite manejar instancias de documentos para su uso con usuarios.

6.6.6. Instancias de documentos

Los documentos son archivos guardados en un sistema de archivos externo, tal como el *S3* de *Amazon Web Services*. Los documentos no pueden ser modificados por un grupo de estudiantes, o todos terminarían viendo las notas de todos los demás estudiantes, lo que puede llegar a ser caótico y problemático. En vez, se manejan las «Instancias de Documentos». Estas no son instancias en el sentido de la palabra que describe a un objeto en memoria. En vez, las instancias de documentos son relaciones entre documentos, grupos, usuarios, comentarios y sesiones. Estos permiten crear documentos «imaginarios» que pueden ser vistos solamente en grupos de estudiantes, en vez de en toda una institución. Esta relación puede ser apreciada de mejor manera en la Figura 14

Modularidad

Las instancias de documentos permiten el manejo modular de la información. Su lógica permite a un usuario solo ver a aquellos comentarios y sesiones hechos por otros usuarios. Su uso está controlado por dos partes: un manejo de *Socket.IO* y un manejo de *REST*. Estos dos servicios solo pueden crear o quitar entradas en una base de datos, los cuales están perfectamente abstraídos por el *backend* del *frontend*. Sin embargo, se debe de entender

que el manejo de *Socket.IO* puede resultar caro para la aplicación, por lo que un sistema aparte, que puede existir fuera de la aplicación, fue creado. Este sistema tiene su propio tipo de autenticación y autorización, existiendo solamente como un micro-servicio. Como fue discutido en la sección 5.5.2, de micro-servicios, estos son reemplazables y fáciles de ajustar y cambiar gracias a su uso de interfaces para comunicarse. Es importante mencionar que *Socket.IO* es un micro-servicio mientras que *REST* es el módulo entero.

6.7. Jerarquía de usuarios

6.7.1. Necesidad de una jerarquía

La aplicación no sería funcional sin una jerarquía de usuarios. Un usuario, en la aplicación, es una entidad que puede consumir información, manejarla, restringirla o cambiarla. Estas acciones son definidas mediante los roles que tienen los usuarios en el portal. No solo tienen los usuarios roles en todo el portal, sino también tienen roles que tienen que ver directamente con los documentos que estos consumen. Por esto, una jerarquía de usuarios es necesaria: los documentos deben de tener siempre un dueño, incluso si estos no tienen exactamente a otros que los consumen.

Los documentos no solo son más seguros cuando se manejan con un dueño siempre, sino que la creación de una jerarquía también facilitó mucho la programación de por sí. Una jerarquía por roles tiene la misma lógica en todas partes. Es decir, el dueño de un documento tiene los mismos poderes administrativos en todos sus propios documentos. Una jerarquía permite, igualmente, un manejo más sencillo de autenticación y autorización. Un documento solo puede ser revisado por aquellos que estén autenticados y autorizados previamente. Asimismo, los roles pueden ser modificados entre sí, de «arriba para abajo». Es decir, un rol que tenga más permisos usuales puede crear y dar permisos a otros usuarios (o grupos) para poder manejarlo todo. Este tipo de relación puede ser apreciada de mejor manera en la Figura 15.

Como se puede observar, los roles fueron diseñados para que el rol más alto de entre todos (en este caso, el rol de *owner*) tenga poderes por encima de los demás roles. Gracias a que los roles terminan siendo combinaciones de permisos, los permisos pueden llegar a darse de manera discreta. Los roles tienen permisos discretos entre sí, como puede ser apreciado en la Figura 16. Se puede observar que un rol, por definición, tiene un conjunto de permisos que permiten que ciertos usuarios tomen acciones encima de un documento. Esto mismo puede ser aplicado al portal.

Originalmente, se había pensado en un sistema de permisos basado en operaciones *AND* entre números que permitiera guardar los permisos **por usuario**. Este sistema estaría guardando los datos como se aprecia en la Figura 17. Como se puede observar, este sistema utiliza un *byte* de memoria, pero no lo utiliza todo. El *owner* de un documento tiene todos los permisos, pero solo existen cinco permisos iniciales. Esto significa que el sistema estaría utilizando un total de cerca del 60 % de los datos disponibles. un **47.5 % de los datos no estarían siendo utilizados bajo este sistema**. Incluso si se puede representar un permiso por cada bit (como es observado en la Figura 18), existe gasto de espacio. Asimismo, el agregar permisos a este sistema es extremadamente difícil. Si se quiere agregar un nuevo

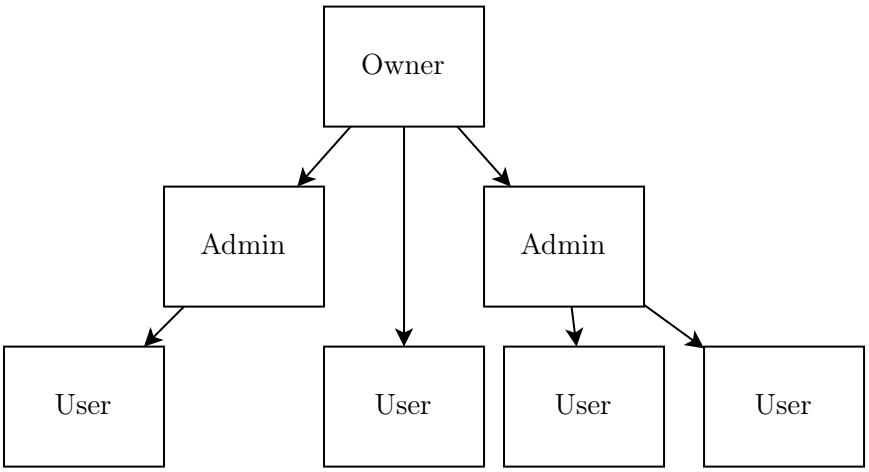


Figura 15: Jerarquía de Usuarios. Gráfica de ejemplo que muestra las relaciones entre roles extensibles y modulares entre varios tipos de usuarios. Este tipo de relaciones es aplicable en dos contextos: en el manejo de documentos y en el manejo del *backend* para el portal.

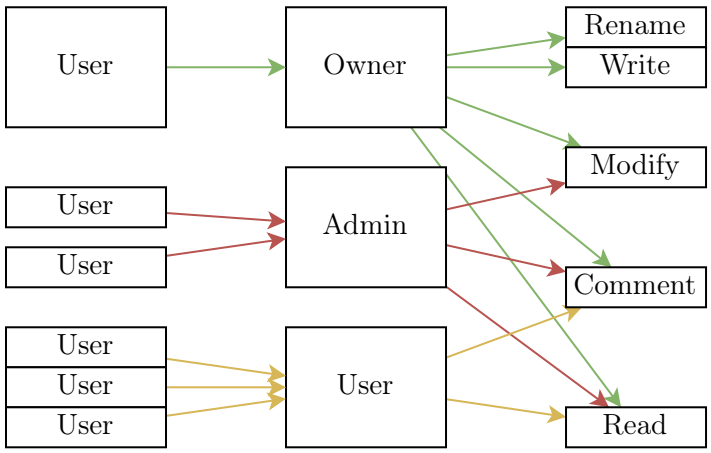


Figura 16: División de Permisos. Los permisos, por defecto, están definidos por rol. Esto significa que un rol juega uno de dos puestos dentro del portal: **ser un padre en una jerarquía** y **ser un conjunto de permisos por defecto para un usuario.** Asimismo, un rol puede dar un conjunto de permisos, pero más permisos pueden ser dados de manera discreta.

Owner	Admin	User
0001 1111	0001 1111	0001 1111

Figura 17: División de Permisos en el Diseño Viejo. Antes, los permisos eran guardados utilizando un número de variable tamaño. En este caso, se guardan todos los permisos dentro de un *byte*. Sin embargo, este sistema terminó siendo poco escalable y bastante confuso en su manejo.

Owner

0	Empty
0	Empty
0	Empty
1	Rename
1	Write
1	Modify
1	Comment
1	Read

Figura 18: Ejemplo de División de Permisos en el Diseño Viejo. El *owner* de un documento tiene todos los permisos descritos por un número 1 en una operación *AND*. Sin embargo, existen ceros al principio del número que solo están gastanto espacio. No solo esto, sino que el tratar de escalar esto para más permisos requiere un cambio gigante al sistema y a los tipos de datos que este maneja.

permiso (para un total de nueve permisos), esto significaría que se debe de agregar un *byte* más al tipo de dato. Esto significaría que el espacio vacío sería ahora de **siete bits sin usar cuando se agrega un bit más.**

En vez de complicarlo todo, se decidió utilizar un sistema normalizado que permitiera la creación de tantos permisos como se quisiesen. Este sistema se basa en la utilización de permisos en una tabla aparte, infinitamente extensible. Cada permiso es referido por un rol del portal o por un usuario mediante una tabla aparte. Como estos permisos son creados por medio de relaciones, como es visto en la Figura 16, estas son infinitamente extensibles. Las relaciones también pueden llevar a la creación de casos especiales, como lo es en el caso de la Figura 19. Un administrador puede llegar a tener el permiso de crear otros administradores, pero no puede modificar el rol del *owner*.

6.7.2. Costos de cambio de una jerarquía

Como fue descrito ya, los costos de cambio de una jerarquía son extremadamente bajos. Para poder hacer un cambio, cualquier cambio, solo se debe de modificar la tabla de los roles. Es decir, solo se debe de insertar un nuevo rol y hacer que este tenga como padre al rol deseado. Este efecto puede ser apreciado en la Figura 20. El crear la gráfica de ejemplo toma más tiempo en promedio que el agregar un nuevo rol al sistema. Asimismo, nuevos permisos funcionan de la misma manera.

Algo importante de saber es que el usuario por defecto, o *default* en la base de datos del proyecto, no tiene ningún permiso en la base de datos, y maneja solo los permisos *default* en ella.

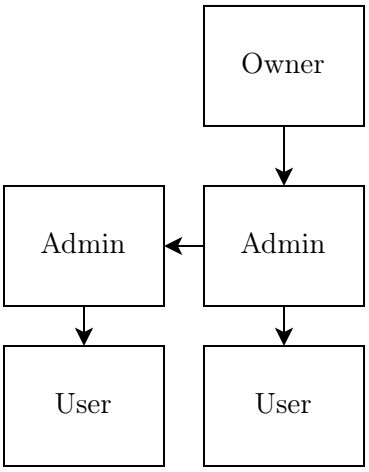


Figura 19: Jerarquía de Usuarios de Mismo Nivel. La creación de usuarios de administrador (en el caso de este sistema) puede estar permitida por permisos singulares de un administrador. Sin embargo, este administrador puede ser único para una jerarquía, forzando a que los demás no compartan ese mismo permiso.

id	Name	ParentId
1	Owner	null
2	Admin	1
3	User	2
4	Mod	2

Figura 20: Adición de un Rol a la Tabla. La creación de un rol en la tabla es extremadamente sencillo. Cada rol puede ser modificado por los roles que están encima de él. En el caso del nuevo rol de *mod*, aunque este tenga el mismo padre que *user*, este no puede ser creado o modificado por este. Ya que *owner* es el padre de *admin*, y *admin* es el padre de *mod*, *owner* **puede** crear y modificar a los roles de *mod*.

Asimismo, no se requieren cambios en el código con la adición de nuevos roles. En vez, estos roles pueden ser añadidos solamente en la base de datos, gracias a que estos no son nada más que conjuntos de permisos y una jerarquía. Sin embargo, el añadir nuevos permisos puede llevar a cambiar cosas en el código, gracias a que estos sí necesitan de cierta lógica dentro del código.

6.8. Base de datos

La base de datos fue diseñada para ser una combinación entre las tecnologías de *SQL* y *NoSQL*. Es decir, la base de datos es **semi estructurada**. La implementación total está hecha en *MariaDB* para facilitar la programación. Pero ¿cómo es creada la parte *NoSQL* de la base de datos? Básicamente, cada módulo que necesita interactuar con un *frontend* de una manera u otra tiene una columna especial que permite que estos guarden cualquier tipo de información no sensible dentro de ella, utilizando el formato *JSON* para poder guardar lo que se requiera dentro de las tablas. Este formato es trabajado sin ningún tipo de *checks*. El manejo de los distintos tipos de datos requeridos, junto con la posibilidad de guardar

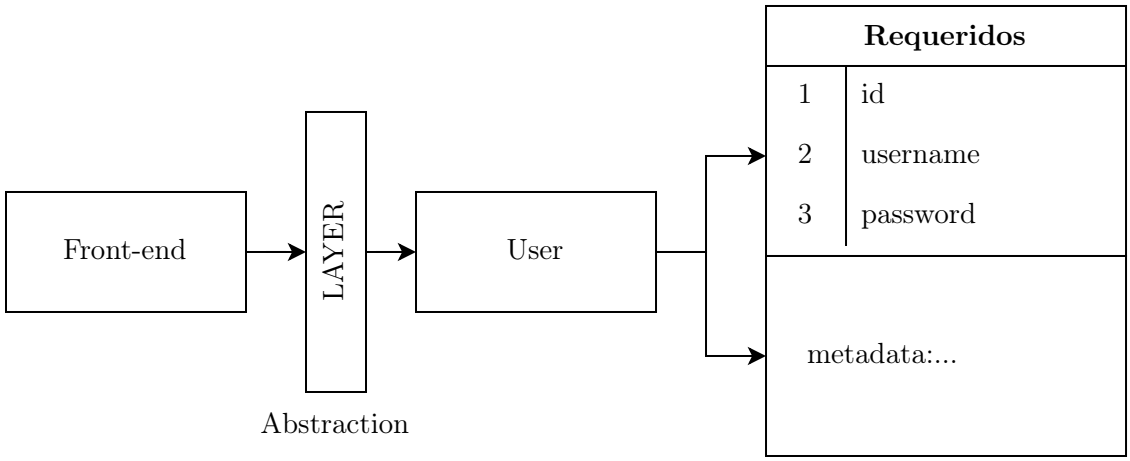


Figura 21: Fragmentación de una Tabla en *SQL* y *NoSQL*. Una tabla que interactúa con el *frontend* tiene dos partes discretas: una parte que maneja los datos esenciales (y, usualmente, los más buscados en una tabla) y una parte que permite el uso de objetos de cualquier forma. La búsqueda dentro de *NoSQL* solo es posible por medio de *Regex*, y puede llegar a ser costosa. El *backend* se encarga de esconder todo lo que pasa con la base de datos. Para el *frontend*, solo son objetos.

datos sin ninguna forma en particular hace que los documentos sean fáciles de manejar. Sin embargo, la búsqueda de datos dentro de estos documentos debe de ser hecha mediante *Regex*, por lo que se recomienda siempre guardar solo las cosas necesarias dentro de este. La relación de los datos puede ser encontrada dentro de la Figura 21.

Es importante aclarar que, aunque se logra emular con éxito partes de una base de datos *NoSQL*, esta **no es una base de datos *NoSQL***. Se tomaron los conceptos que se necesitaban de este tipo de base de datos y se aplicaron a una base de datos relacional. Esto se debe a que el módulo de ciencia de datos necesitaba este tipo de infraestructura para su trabajo. Sin embargo, gracias a los objetivos, el emular la base de datos *NoSQL* era necesario, por lo que se llegó a lo mejor de ambos mundos. La información dentro de estas columnas **no tiene relación con otras tuplas en cualquier tabla**, sirviendo solamente como una manera de manejar datos sin una estructura rígida, como pasaría en una base de datos *NoSQL*.

6.8.1. Uso de *TypeORM*

El uso de tablas en la base de datos probó ser bastante complicado en un inicio. En las fases iniciales del proyecto, se comenzó utilizando *Sequelize* para poder trabajar todo lo que tuviera que ver con almacenamiento estático y *queries*. Sin embargo, esto rápidamente se volvió tedioso, gracias a que todo debía de ser clasificado de manera discreta, por cada propiedad. Aunque existen modelos dentro de *Sequelize*, auto generados con *Sequelize-Auto*, estos rápidamente probaron no ser tan flexibles. Esto se debía a que el poder lo tenía la base de datos, no el programa de por sí. Por ende, la base de datos era la que ponía las reglas, no el código. Por esto, se terminó utilizando *TypeORM* con *Nest.js*. Más adelante se terminó aprendiendo que *TypeORM* tiene soporte nativo por medio de *Nest.js*.

TypeORM es una librería de Node que permite el uso de modelos para las bases de datos.

ORM proviene de *Object-Relational Mapping* que *virtualiza* una base de datos entera, como si se estuviese trabajando con un sistema orientado a objetos, como es descrito en la sección 5.8.1. Su manejo de información es por entidades y tiene protecciones añadidas en contra de una gran cantidad de ataques de mala fe a una base de datos. Una de sus mejores características es su protección nativa en contra de *SQL Injection*, como es descrito en la sección 5.13.1. La utilización de *TypeORM* permitió la abstracción de muchos temas de seguridad. El manejo de las tablas como clases, en vez de estructuras de datos aledañas al programa, facilita mucho cómo es que funciona su lógica dentro del programa. Es importante notar que *TypeORM* tiene muchos patrones de diseño dentro de sí ya, pero se basa mucho en el uso de los repositorios, visto en la sección 6.8.2. Asimismo, está ya optimizado para la creación y uso de las acciones en *bulk*, significando que se pueden hacer varias inserciones al mismo tiempo a una base de datos.

El uso de *TypeORM* permite abstraer, no solo la seguridad de la información y de las cosas que se buscan, sino también permite la creación de tablas de relaciones ***ManyToMany***, ***OneToMany***, ***ManyToOne*** y ***OneToOne***, aumentando la escalabilidad del programa, ya que se crean un sinfín de posibles combinaciones que, a final de cuentas, terminan siendo nada más que propiedades entre clases. Estas combinaciones también permiten el manejo sencillo de los *joins* entre tablas (sea cual sea el tipo de *join*).

6.8.2. Uso de repositorios

Uno de los mayores problemas del sistema anterior tenía que ver mucho con el uso de los modelos directamente, no de un patrón de diseño de repositorios. Los repositorios permiten la abstracción de muchos métodos y funciones de una base de datos, permitiéndonos enfocar solo en la modificación, inserción y supresión de datos.

TypeORM permite el uso de un patrón de diseño de Repositorio, el cual da las reglas exactas de cómo interactuar con una base de datos. Sin embargo, es importante notar que el repositorio es una extensión y no reemplaza a *TypeORM* o a sus entidades. El uso de repositorios dentro del proyecto permitió el crear ciertos cambios para mejorar la lectura del proyecto, aumentando así su propia documentación en código, y creando formas de escalar el proyecto sin problemas. El uso de los repositorios también permiten evitar mucha de la duplicidad de código, creando servicios que son manejados directamente por *Nest.js*. Es importante notar que cada uno de estos servicios puede ser manejado como un *singleton* o como un proceso propio. Por cómo está diseñado el programa (es decir, para permitir la escalabilidad sin problemas), no se utilizan los *singletons* en este tipo de servicios en donde sea posible.

6.9. Uso de *DTOs*

Las aplicaciones de *Nest.js* tienden a tener una capa de revisión de solicitudes a partir de los parámetros que se reciben. En *Nest.js*, estas capas son basadas en *DTOs*, o *Data Transfer Objects*. Estos permiten ver si los parámetros de una solicitud son correctos y completos o no, y si siguen estos ciertas reglas puestas dentro de la clase. El uso de *DTOs* en *Nest.js*

permite que no todas las solicitudes entren al *core* del programa.

La utilización de los *DTOs* en el programa facilita la lectura y la escalabilidad del mismo. Los *DTOs* permiten, a grandes rasgos, no permitir el acceso no autorizado de aquellas peticiones que tengan valores erróneos o extraños. Están diseñados para que nada del código esté expuesto en la respuesta (como es recomendado por motivos de seguridad) y son extremadamente rápidos. Gracias a que no necesitan estar dentro de un controlador, estos están abstraídos completamente de este, estando incluso guardados en otros lugares del programa. Cada *DTO* permite la creación de un camino específico en el programa, y pueden extenderse entre sí, siendo nada más que clases.

Como fue mencionado anteriormente, los *DTO* corren con el programa, siendo parte *Javascript*, en vez de *Typescript* (como es el caso con las interfaces). Por ende, las protecciones son vistas en tiempos de *runtime*, no solo de compilación. Para poder hacer que el código fuese conciso y limpio, se tomó la ruta de los decoradores de *Javascript*. Los decoradores, vistos en los ejemplos más adelante, son una característica de *Javascript* que permiten hacer un *wrapper* alrededor de un fragmento de código, utilizando una función. Es importante notar que estos solo pueden existir alrededor de clases, funciones y propiedades, no dentro del código de por sí. Por ende, es muy común verlos como descriptores.

Un ejemplo de un *DTO*, el cual es utilizado para hacer *login* a la aplicación:

```
// login-credentials.dto.ts
import {
    @IsString,
    @MinLength,
    @MaxLength,
} from 'class-validator';

export class LoginCredentialsDto {
    @IsString()
    @MinLength(1)
    @MaxLength(20)
    username: string;

    @IsString()
    @MinLength(8)
    password: string;
}
```

Como fue mencionado anteriormente, gracias a que la utilización de los *DTOs* se basa en la implementación de clases de *Javascript*, estos pueden terminar extendiéndose entre sí, haciendo que la implementación sea mucho más sencilla y que, a final de cuentas, exista cierta dependencia entre conceptos que lo requieren:

```
// registration-params.dto.ts
import { LoginCredentialsDto } from './login-credentials.dto.ts';
import {
    @IsNumber,
    @IsOptional,
```

```
    @IsDateString,
} from 'class-validator';


export class RegistrationParamsDto extends LoginCredentialsDto {
    @IsBoolean()
    @IsOptional()
    sex: boolean = true;

    @IsDateString()
    dateOfBirth: string;

    @IsNumber()
    portalRoleId: number;

    @IsNumber()
    institutionId: number;
}
```

Es importante tomar nota de cuándo es que se quiere relacionar dos conceptos. En este caso, gracias a entrevistas de usuario y requerimientos del *frontend*, se terminaron relacionando los conceptos de *login* y *register*, visto por la frase *extends* en la clase de *RegistrationParamsDto*. Esta es una decisión peligrosa que no debe de ser tomada muchas veces en el programa, ya que termina uniendo dos conceptos que pueden tener nada en común. Incluso aquí, el haber unido estas dos clases fuerza a que los parámetros de *username* y *password* estén siempre presentes. La fase de registro, gracias a que solo se está buscando trabajar con una universidad por el momento (por alcance), está diseñada para aceptar ambos parámetros. El módulo de registro es totalmente opcional. El uso del módulo de *login* es modular, para que cualquier cosa pueda ser utilizada en vez del *login*.

Gracias a que los *DTOs* manejan su propia lógica, se podría pensar que la aplicación está separada por capas, como lo muestra la Figura 22. Como se mencionó anteriormente, sin la utilización de un *DTO*, cada fragmento de código en un controlador tendría que revisar que todo esté en orden, aumentando la posible duplicidad de código.

6.10. Objetivo de uso de *NoSQL* sin uso de *MongoDB*

Para ampliar lo previamente discutido, conforme a la Figura 21, se debe de aclarar el uso de *NOSQL* sin *MongoDB*. Uno de los objetivos del proyecto tenía que ver con el uso de estructuras de datos que asimilaran o fueran parte de *NoSQL*. Gracias a cómo trabaja *SQL*, fueron posibles ciertos tipos de columnas en la base de datos. En específico, para lograr que sí existieran ciertas columnas, con cierta *Metadata*, se utilizó el tipo de dato *JSON* (más información en la sección 5.14.1). El utilizar *JSON* permite el guardar cuanta información se necesite de acuerdo al *frontend*. A grandes rasgos, se hizo que la base de datos fuera completamente híbrida, permitiéndose la creación de relaciones entre varias tuplas sin tener que seguir una forma 100 % rígida. Gracias a este diseño, cosas que pertenecen más al

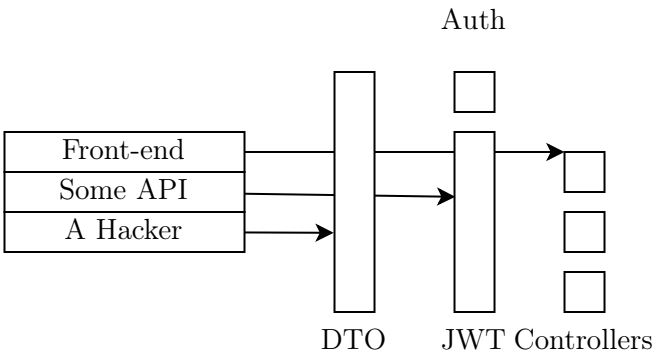


Figura 22: Protección de una Capa de *DTO* para el Resto del Programa. Una capa de *DTOs* permite que el resto del programa no se ejecute si no es necesario. Como está antes que todo el programa, esta capa permite que el código no pueda ser expuesto de manera remota. Alguien haciendo peticiones de mala fe fallaría el *DTO* sin saber por qué. Una *API* no autorizada fallaría el JWT. Finalmente, el *frontend*, dado que sí esté autenticado, llegaría hasta los controladores.

frontend, como lo son las personalizaciones de un usuario, pueden ser guardadas en la base de datos sin problema. Obviamente, con esto hay que tener mucho cuidado. el guardado de información dentro de estos objetos puede terminar siendo redundante con lo que ya existe dentro de la base de datos, por lo que se ha manejado cuidadosamente todo aquello que se guarda con el módulo de *frontend*. Asimismo, se han llegado a acuerdos sobre qué se puede mantener en estos objetos (como colores de usuario, por ejemplo) y qué no (como lo que son contraseñas o información sensible).

No todas las tablas tienen la columna de tipo *JSON*. Esta columna está reservada solamente para las tablas que interactúan directamente con el *frontend*, no solo con el *backend*. El tener este tipo de columna en cualquier otro tipo de tabla **no sería nada más que un desperdicio de espacio**, por lo que se decidió no utilizarla. Todas las tablas que existen solamente para su interacción con el *backend* son, por ende, perfectamente estructuradas, sin utilizar *NoSQL* en ellas.

6.11. Manejo de respuestas en el ámbito global

Uno de los problemas más tempranos cuando se estaba diseñando la *API*, era el de la comunicación entre esta y cualquier otra *API* que la consumiera. Gracias a que estos temas son muy bien conocidos en la industria, se pudo tomar inspiración de otros *frameworks*, así como patrones de diseño que estos siguen. Las salidas y las entradas del programa debían de ser normalizadas de tal manera que fueran seguras, escalables, y siguieran la filosofía de *Nest.js*. Por esto, se terminaron utilizando los *interceptors* y los *JWT*, que actuaban en las peticiones y en las respuestas. Mientras que los *JWT* fueron utilizados, más que todo, para el manejo de sesiones, los *interceptors* tomaron acción directa en todo lo que es la entrada y la salida del programa.

6.11.1. Uso de *interceptors* como puntos de control de entrada y salida

Los *interceptors* permitieron crear las entradas y salidas del programa de manera uniforme. Esto fue inspirado completamente en patrones de diseño seguidos en empresas grandes que utilizan *ASP.NET*. Las respuestas tienen una forma normalizada basada en:

- Qué tipo de operación se tuvo.
- Si fue el resultado de la operación **esperado o no**.
- Si la operación tuvo un resultado o un error manejado por el programa.

El uso de un *interceptor* ayuda a que el programa no sufra de redundancias constantes. La implementación de un *pipeline* para que este funcionara permite que las respuestas sean tomadas justo al final de una función, cuando esta retorna un valor (si es que lo hace). El uso de estos fragmentos de código permite que las respuestas no dependan de clases aledañas que se tengan que instanciar después de cada respuesta. Asimismo, evita el uso de funciones de alto nivel que funcionen como *wrappers*. Es importante notar que los **decoradores** no son posibles en este contexto de la misma manera que lo son con las propiedades y las funciones, ya que se vuelven intrusos en cada una de las funciones y, a final de cuentas, hacen que el código sea un poco más difícil de mantener. Un cambio en cualquier parte del programa aumentaría el costo de cambio considerablemente. Finalmente, es importante mencionar que un *interceptor* actúa sobre **todas las respuestas del programa**, por lo que no se puede tener demasiada información específica dentro de él. Asimismo, este no debe de agregar información redundante a una respuesta, como lo es el código de *HTTP* que se generó en la respuesta. Esto puede apreciarse a detalle en la Figura 23.

Gracias a que el proyecto se basa mucho en la comunicación entre controladores, más que entre el *frontend* y el *backend*, la utilización de *interceptors* se dejó extremadamente modular como para que se puedan añadir y quitar sin problemas, tal sea la necesidad. Esta modularidad, a final de cuentas, se hizo a partir de la necesidad de aumentar la escalabilidad del programa, la disminución de costos de cambio y el seguir las filosofías de *Nest.js* según su propia documentación.

6.11.2. Uso de *JWT* y manejo de sesiones

Los *JWT*, discutidos en la sección 5.15.2, son utilizados en la mayoría de la aplicación. Son la forma más básica para dar autenticación a un usuario. Es importante saber que los *JWT* contienen el usuario de aquella persona que esté haciendo la petición, y están firmados para saber que no han habido cambios en la información del *JWT*. Este token contiene poca información para hacer que las peticiones no crezcan mucho tampoco. Esto sigue la filosofía de solo comunicar lo necesario, así no hay desperdicio cuando se trata de procesar algo.

El proceso del *JWT* es relativamente simple, ya que está abstraído a tal punto que todos sus procesos tienen que ver directamente con librerías de terceros, dos de las cuales ya están implementadas dentro de *Nest.js*. La librería de *passport* permite el crear estrategias sobre cómo una petición debe de existir para poder utilizarse. *Nest.js* lo implementa para poder

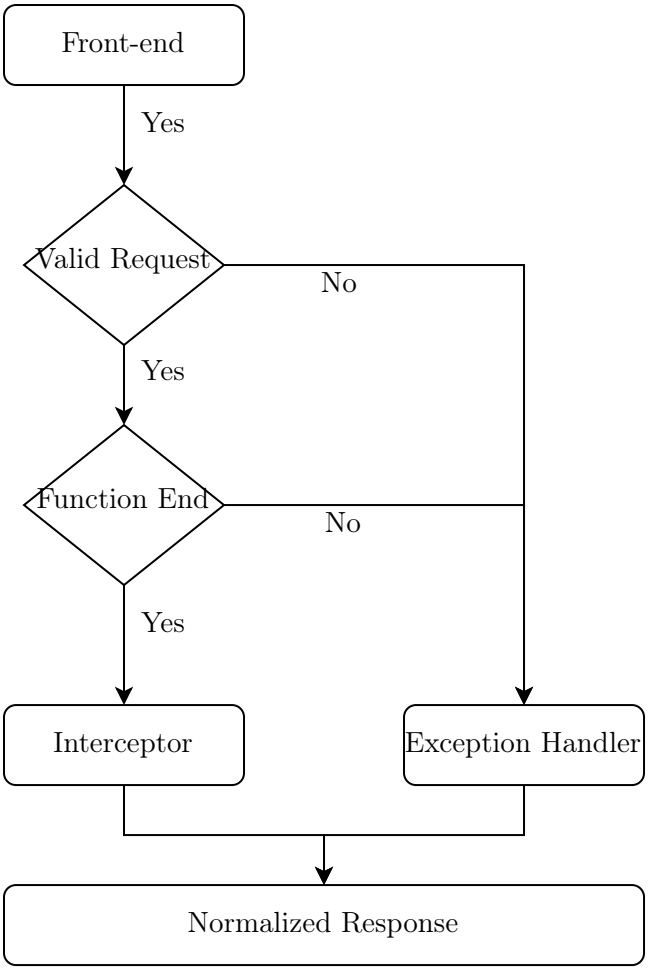


Figura 23: Flujo de Respuestas a Nivel Macro del Programa. El programa de por sí necesita disminuir el error humano y normalizar todas las respuestas que este da. Por esto, se tomo en cuenta que algunas respuestas pueden tener un código de *HTTP* de 200, pero pueden haber errores dentro del proceso. Por esto, todos los controladores hacen que sus respuestas pasen siempre por un *interceptor* o un *exception handler*, hechos para el proyecto, pero implementados en el *pipeline* por *Nest.js*.

hacer que la implementación sea tan sencilla como sea posible. Asimismo, *Nest.js* también implementa la librería necesaria para los *JWT*. Cómo funciona puede también apreciarse en la Figura 22.

6.12. Manejo de excepciones

Las excepciones en un programa normal pueden ser manejadas o simplemente hacen que un programa pare. Para motivos de un programa web, el permitir que una excepción pare el programa es un error abismal. Por esto, *Nest.js* permite el manejo de excepciones con un *exception handler*. En el programa creado, se tuvo que hacer uno de estos personalizado para el ámbito global. Claro, este tipo de respuestas permiten la normalización de la misma manera que los *interceptors* discutidos anteriormente. Sin embargo, estas tienen un pipeline más complicado de seguir, pero todavía bastante simple.

En temas de seguridad, las excepciones no dan demasiada información sobre el error ocurrido. Asimismo, no tienen permitido el enseñar información de qué está pasando en temas de bajo nivel, como sucede con conflictos de modelos de usuario. Por esto, está hecho para solamente dar errores a partir de los códigos *HTTP* y un error genérico. Gracias a cómo trabaja *Nest.js*, la aplicación nunca terminará abruptamente por culpa de una petición malintencionada.

6.12.1. Excepciones por autenticación

Las excepciones existen de varios tipos. El primer tipo importante es el tipo por autenticación. El tipo por autenticación ocurre cuando un usuario no posee un *JWT*. Este tipo de autenticación puede ser apreciada en la Figura 22, sufrida por la *API* no autenticada. Esta sí tiene los datos necesarios para pasar el *DTO*, pero no tiene un *token* para poder pasar por los *interceptors* (también llamados guardias) que vienen antes del controlador. Por esto, la respuesta del servidor es una respuesta 403 (*Forbidden*).

Hay algunas peticiones que sí tienen un *JWT* que fue generado por el *backend* y, por lo tanto, válido. Sin embargo, no todos los tokens generados son generados por mucho tiempo. Los *JWT* tienen una vida media que no permite que sean usados hasta el infinito. En realidad, este tipo de error también se puede dar cuando un *JWT* tiene una vida expirada. Este tipo de error sucede dentro de los controladores, ya que son estos los que revisan que el token de por sí sea válido todavía. La misma librería que los genera puede revisar si el *hash* es correcto o no. Un ejemplo muy simple de este proceso puede ser visto en la Figura 24.

6.12.2. Excepciones por autorización

Las excepciones por autorización terminan teniendo el mismo proceso que la Figura 24, pero ignorando completamente el *JWT*. En vez, si un usuario no tiene los permisos necesarios, estos son tratados con la misma respuesta que en la sección 6.12.1. Imaginemos la misma jerarquía encontrada en la Figura 16. En esta jerarquía, si un usuario normal tratara de

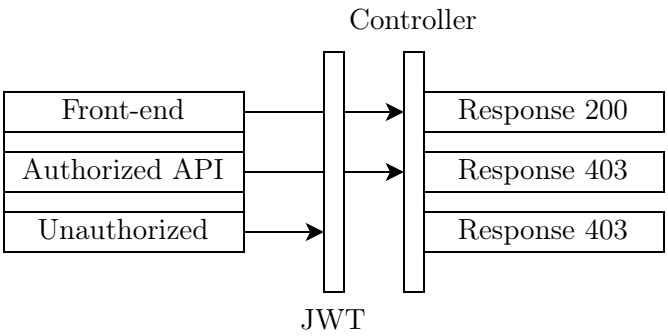


Figura 24: Manejo de Excepciones por JWT. De tres entidades, dos pasan a los controladores, pero solo uno recibe la respuesta deseada. Si un *JWT* ya no es válido, este se trata como si no existiese y la petición estuviera vacía. En este caso, el *Authorized API* **no tiene un token que siga estando activo.**

hacer una modificación al documento, no le sería posible. La respuesta del servidor sería de 403, junto con las protecciones que existen en el *frontend*. Esta doble capa de seguridad hace que los datos mantengan su propia integridad.

Sin embargo, es importante pensar en cómo los roles están hechos: Un rol puede heredar permisos de otros roles en su creación. Por esto, se dice que los roles están hechos por composición, y dependen mucho de la jerarquía de usuarios impuesta. Gracias a esta interacción de entidades, esta parte termina siendo extremadamente modular, haciendo que la pirámide de permisos se pueda cambiar sin ningún problema.

6.13. Generación de datos

Uno de los temas más recurrentes en la actualidad es la generación y estudio de los datos en un sistema. En específico, la generación de datos, en este proyecto, hace referencia a la generación de una cantidad n de datos que puedan ser utilizados para la creación de modelos para el módulo de ciencia de datos. Estos modelos, ya que están basados en la vida real, tienen que tener información creíble y que se asimile a la real de la mejor manera posible. Por esto, se terminó utilizando la librería *Faker.js* para la generación de la información.

Faker.js permite la generación constante de datos en masa. Sin embargo, gracias a que el módulo de *Machine Learning* no interactúa con todas las tablas, solo era necesario generar información para las tablas de estudio (tal como la tabla de sesiones y comentarios) y todas las tablas en las que estas dependen (como la tabla de usuarios, de documentos, etc.) La generación de información tiene como objetivo principal el poder generar instancias de documentos, mencionadas en la Figura 14.

6.14. Manejo de configuraciones

El manejo de las configuraciones debe de ser manejado al principio de la aplicación por métodos asíncronos. Estos utilizan módulos de *Nest.js* para poder manejar cualquier tipo de

configuración, ya sea por parte de las variables de entorno, o directamente de un directorio dentro del *root* del proyecto. Se escogió este método porque era ya nativo de *Nest.js*, pero también porque acepta las variables de entorno sin problema. Asimismo, existe la posibilidad de utilizar archivos de *.env* para declarar variables que solo están disponibles durante tiempo de desarrollo.

Una de las ventajas más grandes de este tipo de módulo es que permite que las cosas terminen siendo trabajadas como si fueran componentes más dentro de la aplicación. En específico, las configuraciones se pueden trabajar como servicios, permitiendo que puedan ser inyectados en cualquier parte donde sea conveniente, decrementando el tiempo de programación.

6.15. Combinación de *sockets* y REST

Gracias a las entrevistas de usuario, se terminó trabajando en lo que son *Sockets* y una arquitectura *REST*. Los *sockets* son un servicio aparte que existen dentro del mismo proyecto. Gracias a que técnicamente están separados, este puede ser extraído y convertido en un microservicio. Los *sockets* no existen si no existe una instancia de documento a cual relacionarlo. Se trabaja así gracias a que una instancia de *Node.js* puede manejar 9000 conexiones al mismo tiempo, promedio. El programa puede manejar más, pero el haberlo hecho un servicio aparte hace que el servicio principal no termine sufriendo por culpa de este. Asimismo, se pueden crear varias instancias de este mismo servicio para manejar varios *sockets* y aumentar la escalabilidad horizontal. Más información sobre cómo los sockets están diseñados pueden ser encontrados en la Figura 25.

6.15.1. *Sockets*

Los *sockets* en el proyecto actúan como formas de hacer que las cosas se vean en tiempo real. Cada acción es sometida a un socket, como lo es, por ejemplo, con el dejar un comentario. Los *sockets* tienen una vida media. Si un usuario se sale del mismo, este *socket* termina desapareciendo hasta que un nuevo usuario se conecte. Todos los *sockets* son manejados con *UUIDs* para poder evitar colisiones. Este es el mismo *UUID* que el de la instancia del documento. La necesidad de los *sockets* nació gracias a que los usuarios pereferían las cosas inmediatas, tal como pasan en otros tipos de servicios, como en *Google Docs* o en *Microsoft Word Online*.

6.15.2. REST

REST, por definición en el proyecto, se utiliza para todo lo que no tiene que ver con las instancias de documentos. Son las bases para el *frontend* cuando este quiere manejar información de usuarios, crear y borrar documentos, cambiar roles y hacer cambios a ámbitos de los distintos usuarios. Es por medio de la *API* de *REST* que se terminan teniendo los *Sockets* ya que, sin el *REST*, la autenticación simplemente no es posible. Los *sockets* también consumen a esta *API* de manera interna, ya que estos solo necesitan hacer que las cosas

sean inmediatas.

6.16. Diseño a partir de conveniencia

El proyecto sigue la filosofía de la mínima acción para la programación de características del proyecto. Gracias a que el proyecto está hecho para servir un tipo de necesidad, el programar de más solo haría que las cosas se entorpezcan. Asimismo, esto permite que el camino del proyecto no esté todavía definido, dándole mucha más flexibilidad a este.

El proyecto, gracias a la filosofía anteriormente mencionada, así como la modularidad del mismo, está diseñado para que tantas cosas sean tan escalables como sean posibles. Las cosas son intercambiables, en vez de trabajar solo como un sistema puramente monolítico. La escalabilidad de por sí también ayuda a que los costos de cambio sean extremadamente bajos, gracias a que el proyecto no es difícil de cambiar y, dadas las responsabilidades que existen en cada componente, no sea difícil de extender. Sin embargo, cierto cuidado es necesario aquí, ya que el extenderlo requiere que las mismas filosofías se sigan.

Gracias al alcance del proyecto, no se manejan contenedores de programas, pero todo está programado para que dicho cambio sea sencillo. Existen archivos en el proyecto para poder servirse con *Docker*.

6.17. Modelo de análisis de emociones

El modelo de análisis de sentimientos se creó con el lenguaje de programación de *Python*, debido a que es uno de los lenguajes de programación más utilizados en el ámbito de la ciencia de datos. Desde un contexto empresarial, el costo de creación en el lenguaje de *Python*, es mucho menor al que sería necesario para realizarlo con sus competidores, *R* y *Julia*.

La fuente de datos de los comentarios se obtuvo de la competencia de *Spanish Arilines Tweets Sentiment Analysis* por medio de la plataforma *Kaggle*. Este conjunto de datos se encuentra dentro del alcance del proyecto, debido a que poseen las cualidades de múltiples comentarios en el idioma español. Estos se encuentran clasificados, además de que se encuentra en formato *csv*, lo cual hace la carga del conjunto de datos más sencilla.

6.17.1. Fases del modelo de análisis de emociones

Se cargó el conjunto de datos al tipo de estructura de datos *DataFrame*. Se mantuvo 4 columnas de 10. Las dos columnas más importantes son: *airline sentiment* y *text*.

Una vez se encontró la carga correcta, se continuó con el paso de procesamiento.

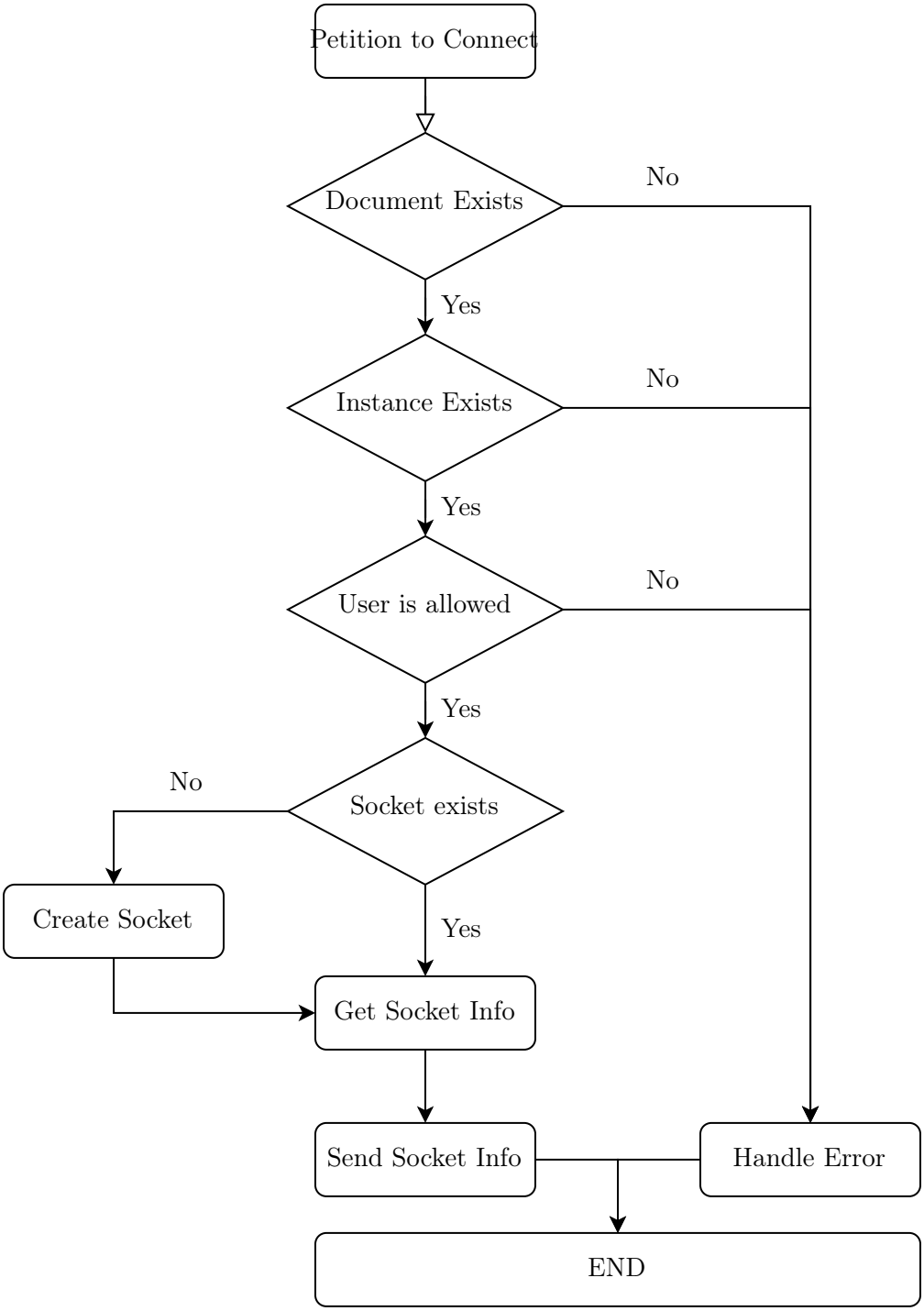


Figura 25: Relación entre *Sockets* y *REST*. Aunque los sockets son partes singulares del proyecto, se decidió diseñarlos a partir de la asunción que estos no pueden existir sin *REST*. A final de cuentas, solo permiten hacer que las funciones de *REST* sean inmediatas para todos los usuarios.

6.17.2. Preprocesamiento

El primer paso fue cambiar las letras para que todas estuvieran en un mismo formato. Para ello se tuvo que decidir si dejar las palabras todas mayúsculas o minúsculas, en este caso se decidió en dejarlas todas en minúsculas.

Un segundo paso dentro de la limpieza de datos fue la eliminación de *URLs* que se podían encontrar en los textos. Fue importante eliminar todo segmento y no solo segmentos como «http», por lo que se tuvo que declarar que si un segmento empezaba con eso, debía eliminarse todo. Otro tema fueron los *tags* a personas, eso también se tuvo que eliminar para evitar nombres juntos en un formato extraño que cambiara el rendimiento del modelo.

A continuación, como tercer y cuarto paso, se realizaron los cambios en las palabras que tuviesen algún tipo de tilde o caracteres especiales como la «ñ». Hacer esos cambios permitió un entrenamiento del modelo con menos problemas.

El quinto paso fue hacer la *tokenización* de las palabras dentro de la columna *text*. Esto permitió evaluar cada segmento del texto y también facilitó la elaboración del sexto paso: eliminar palabras vacías de la columna de *text*. El séptimo paso fue un cambio de uso de cadenas de texto a números para la clasificación.

6.18. Análisis exploratorio

Dentro del segmento de análisis, es importante verificar si el conjunto de datos se encuentra balanceado para tener en consideración que más adelante se debe crear un paso de balanceo de datos. En este caso se verificó para poder visualizar si se encontraba balanceado.

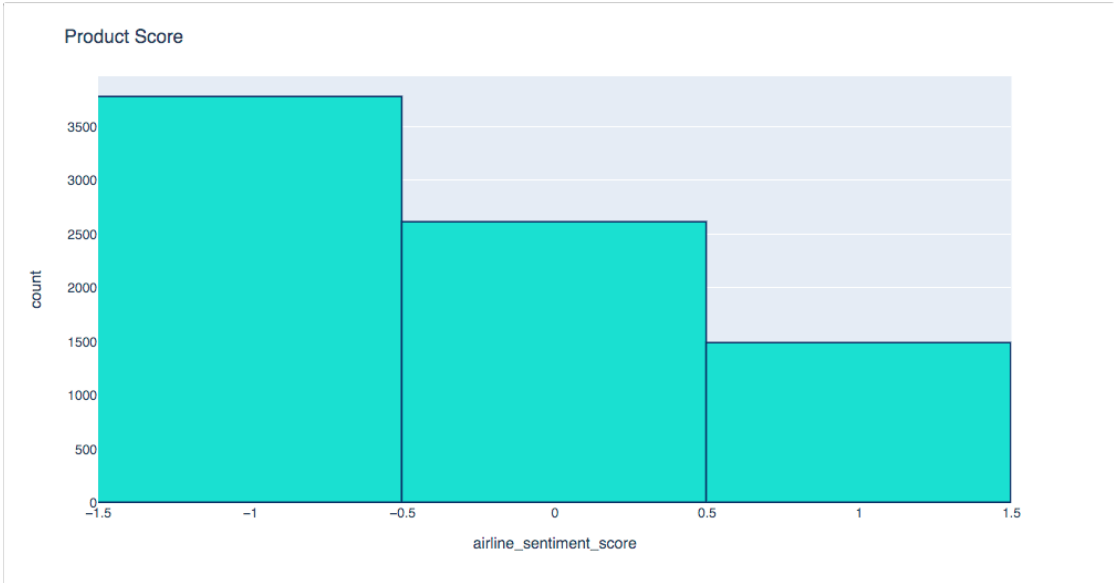


Figura 26: Gráfica de desbalanceo de datos entre clases

De acuerdo a la Figura 26, se puede evidenciar que fue necesario un paso de balanceo de

datos.

El octavo paso fue la creación de un *text stemmer*, el procedimiento fue crear una columna extra con los textos aplicados al *stemmer*. Como parte del análisis exploratorio, se pudo observar que el uso de ese *stemmer* en el idioma español no era tan robusto como en el idioma inglés. Se hizo la comparación al dejar solo el texto y se evidenció que palabras como «pobre» se transformaban en «pobr» y palabras como «siempre» se transformaban en «siempr». Por esta razón, se tomó la decisión de dejarlo sin el paso de *stemmer*.

El siguiente paso fue dejar solo la columna *text* y la columna *airline sentiment score*.

Se decidió crear un modelo que evaluara cada palabra en el *jupyter notebook* original. Para la primera prueba se utilizó la función *explode*, que toma cada uno de los elemento de una lista y los separa dejándoles la clasificación de la cadena entera a estas palabras.

Esta primera prueba demostró más adelante, que existen muchas palabras repetidas dentro del conjunto de datos. Al usar la función de *explode*, se puede crear una palabra con diferentes clasificaciones. En este tipo de situaciones un modelo no se entrenaría de la forma correcta.

En el *jupyter notebook* de la segunda prueba y la prueba final, se implementó el método de *Mechanical Turk*: un concepto de la empresa de *Amazon* que establece que una persona o un grupo de individuos, pueden ser clasificados.

La implementación del método *Mechanical Turks* se comprendió en los siguientes pasos:

1. Crear un *csv* con todas las palabras repetidas, se crea una copia, para trabajar sobre ella y no interrumpir el documento original.

2. Dentro de la copia se ordenan las palabras por orden alfabético.

3. Se eliminan duplicados.

4. Se separa por grupos para dar a diferentes compañeros.

5. Se explica como se deben clasificar las palabras, por los tres tipos de clases que hay.

6. Se crea énfasis en el hecho que deben tomar cada palabra como individual, sin contexto, si no es clasificada como una palabra negativa o positiva, se debe dar la clasificación de neutra.

7. Una vez se obtienen todas las verificaciones, se crea un tercer documento, donde se comparan las palabras originales, donde se mandan a llamar la clasificación de las palabras del anterior documento. Se guarda y este nuevo conjunto de datos se carga al proyecto.

Se pudo visualizar cómo la clasificación de las palabras tuvo un mayor sentido: previamente la palabra «gusten» era una palabra neutra, pero con el método de *mechanical turk* obtuvo la clasificación de una palabra positiva.

	text	airline_sentiment_score
0	gusten	0
1	cancun	0
2	viaja	0
3	disfruta	0
4	manera	0

Figura 27: Salida sin método *mechanical turk*

	text	airline_sentiment_new_score
0	gusten	1
1	cancun	0
2	viaja	0
3	disfruta	1
4	manera	0

Figura 28: Salida con método *mechanical turk*

Separación de comentarios

En las tres pruebas se pudo visualizar una mejora entre los *wordclouds*. En el *jupyter notebook* del primer intento se pudo visualizar en grande la palabra «sí» en el *wordcloud* de emoción negativo. Este comportamiento cambia en tanto en el *jupyter notebook* del segundo intento y el último intento. La diferencia entre el segundo intento y el último, es el que no hubo tanta repetición de palabras y se pudo observar más palabras con ponderaciones similares.

Figura 29: *Wordcloud* del primer intento

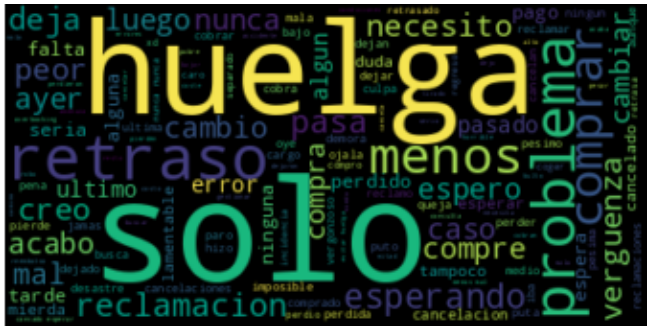


Figura 30: *Wordcloud* del segundo intento

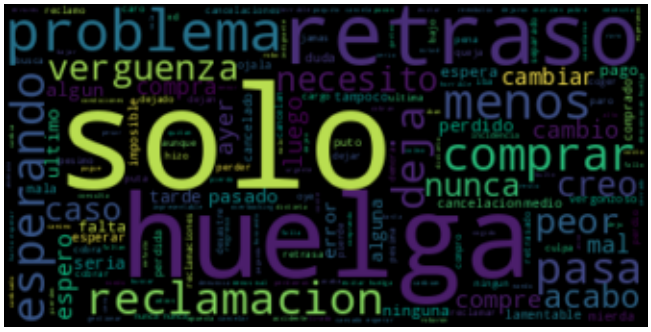


Figura 31: *Wordcloud* del tercer intento

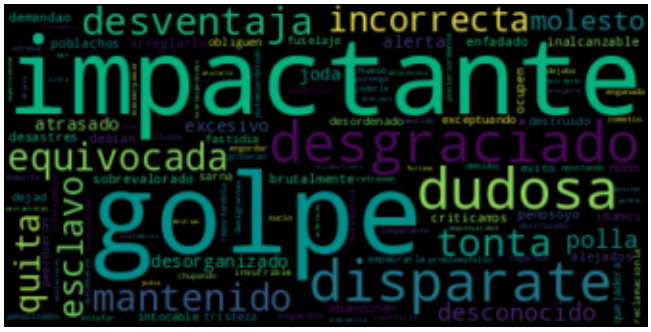


Figura 32: *Wordcloud* del tercer intento sin duplicados

Uno de los detalles a considerar fue el factor de los duplicados dentro de un conjunto de

datos, debido a que estos podrían haber creado *overfitting* o *underfitting* en el modelo. Para mitigar esto, se elaboró un paso de eliminación de duplicados antes de pasar a la sección de modelado.

6.19. Modelado

Se creó la separación entre las características, las palabras, y sus etiquetas. En las clasificaciones, se corroboró si existían datos duplicados. Se pudo ver un caso positivo de duplicados en el *jupyter notebook* del segundo intento, mientras que el primero y el segundo no lo tuvieron.

Dentro la sección de modelado se definió el segmento de balanceo de datos y se verificó si era necesario el balanceo de datos de nuevo o si se mantenía la respuesta.

La técnica de balanceo que se utilizó fue la de *Oversampling*, debido a que eran muy pocas las palabras con clasificación positiva y negativa. Se utilizó la técnica de *Random Oversampler* debido a que con está técnica es posible el uso de cadenas de texto, por otro lado, si se hubiera querido usar la técnica *SMOTE* o *ADASYN* era necesario que los datos fueran de tipo *float*. Por lo que, se optó por el *Random Oversampler*.

La separación entre el conjunto de entrenamiento y el conjunto de *testing* se mantuvo a 70 % de entrenamiento y 30 % de *testing*. Esto se realizó en los 3 *jupyter notebooks*. Otro aspecto que se consideró en fue el tener la seguridad que dentro de los conjuntos de entrenamiento y *testing* se debían estar lo más balanceados posibles los datos. Por esta razón se realizó la prueba de una manera gráfica.

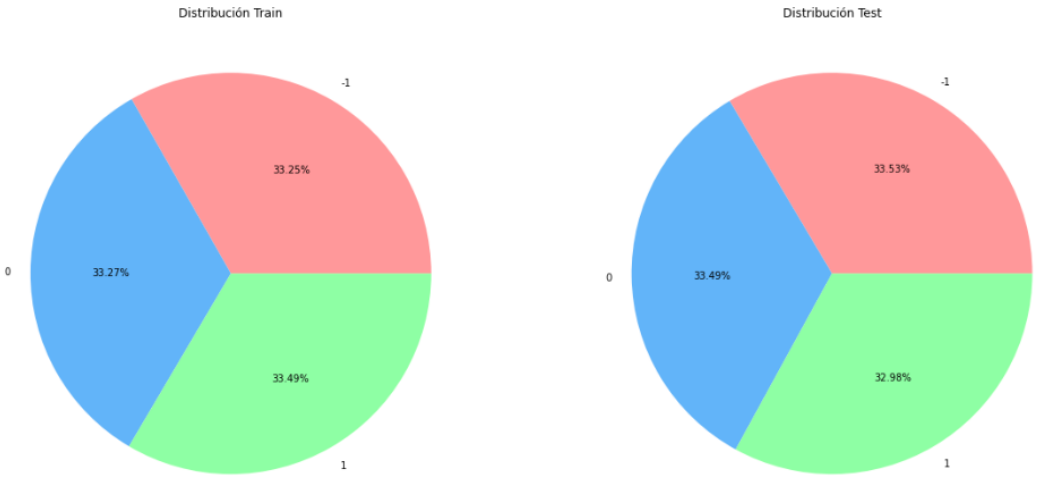


Figura 33: Gráficas de distribución de clases en conjuntos de datos de entrenamiento y *testing*

Lo siguiente que se creó fue un *CountVectorizer*, como se explicó en el marco teórico, esta función permite convertir los textos a forma de vector, lo que los hace más estructurados para el entrenamiento. Después de tener los datos en vectores, se creó un *Bag of Words*, que guarda las palabras únicas del conjunto de datos.

Lo siguiente que se hizo fue la creación de los modelos: se tomaron los modelos de clasificación que comúnmente se utilizan, para ser modelos de análisis de emociones. Se crearon 7 modelos: 4 de estos modelos son similares, ya que se les cambió una característica para ser el mismo tipo de modelo de clasificación. Se adjuntaron en un diccionario, para que el manejo de estos fuera lo más ordenado posible.

Se creó una función de nombre *evaluation*, con el propósito de simular un reporte de clasificación (*classification report*). El objetivo de esto es poder comparar cada modelo con las dimensionalidades uno directamente después del otro, en vez de crear *Classification Reports* individuales y tener que ir comparando uno con otro. Esto brindó un diccionario, que luego se convirtió en una estructura de *DataFrame*. Con esto se logró visualizar los resultados de cada modelo.

Función que devuelve un classification report de un modelo en forma de un diccionario

Sus parametros son:

```
Modelo -> Modelo que se va a evaluar
Nombre -> Nombre del modelo
X_train -> Variables de entrada del conjunto de datos de entrenamiento
X_test -> Variables de entrada del conjunto de datos de test
y_train -> Variable de salida del conjunto de datos de entrenamiento
y_test -> Variable de salida del conjunto de datos de test
```

Dentro de un diccionario tiene como keys:

```
Nombre
Accuracy
Precision
Recall
F1-score
```

Figura 34: Parámetros de la función de evaluación

En el primer intento se encontró el fallo dentro del conjunto de datos de entrenamiento y *testing*. Esto pudo darse debido a palabras que son similares, pero con una clasificación diferente. Este fenómeno hace que los modelos reaccionen con poco rendimiento a la hora de ser evaluados con el conjunto de datos de *testing*.

En el segundo intento se encontró un caso de *overfitting* en los modelos. Esto se dio debido a que este fue el *jupyter notebook* donde se encontraban los duplicados, más los duplicados en instancias por el *Oversampling*. Se observó este sobreajuste debido a que las métricas fueron muy altas en los distintos modelos, lo cual no es común en modelos de clasificación binaria para este tipo de aplicaciones.

En el último intento, se logró ver que los modelos sí reaccionaron de la forma en la que debían reaccionar. Los modelos de clasificación binaria reaccionaron como debía ser, y los modelos *multiclase* tuvieron un buen rendimiento. Los modelos de *LogisticRegression* y *SVM Kernel linear* tuvieron un cierto grado de *overfitting* por los duplicados de las instancias en el *Oversampling*. Por ende, se destacaron entre los otros modelos y mostraron un mejor rendimiento al ser modelos *multiclase*.

nombre_modelo	accuracy	precision	recall	f1-score
BernoulliNB	0.483764	0.797748	0.483764	0.408374
MultinomialNB	0.467829	0.463959	0.467829	0.378592
LogisticRegression	0.483764	0.797748	0.483764	0.408374
Random Forest d_20	0.410704	0.787423	0.410704	0.310561
Random Forest d_50	0.415815	0.788092	0.415815	0.318374
SVM Kernel linear	0.483764	0.797748	0.483764	0.408374
SVM Kernel rbf	0.483764	0.797748	0.483764	0.408374

Figura 35: Tabla de *clasification report* del primer intento

nombre_modelo	accuracy	precision	recall	f1-score
BernoulliNB	0.906449	0.927094	0.906449	0.908736
MultinomialNB	0.906449	0.927094	0.906449	0.908736
LogisticRegression	0.906449	0.927094	0.906449	0.908736
Random Forest d_20	0.628142	0.825056	0.628142	0.623612
Random Forest d_50	0.640781	0.827905	0.640781	0.638137
SVM Kernel linear	0.906449	0.927094	0.906449	0.908736
SVM Kernel rbf	0.906449	0.927094	0.906449	0.908736

Figura 36: Tabla de *clasification report* del segundo intento

nombre_modelo	accuracy	precision	recall	f1-score
BernoulliNB	0.663672	0.498587	0.663672	0.553011
LogisticRegression	0.997958	0.997970	0.997958	0.997959
MultinomialNB	0.663672	0.498587	0.663672	0.553011
Random Forest d_20	0.579130	0.813503	0.579130	0.562254
Random Forest d_50	0.580355	0.813744	0.580355	0.564133
SVM Kernel linear	0.997958	0.997970	0.997958	0.997959
SVM Kernel rbf	0.663672	0.498587	0.663672	0.553011

Figura 37: Tabla de *clasification report* del tercer intento

Se crearon tablas comparativas sobre el rendimiento de cada una de las columnas del *Classification Report*, para proveer una mejor visibilidad de los resultados de los modelos por cada métrica.

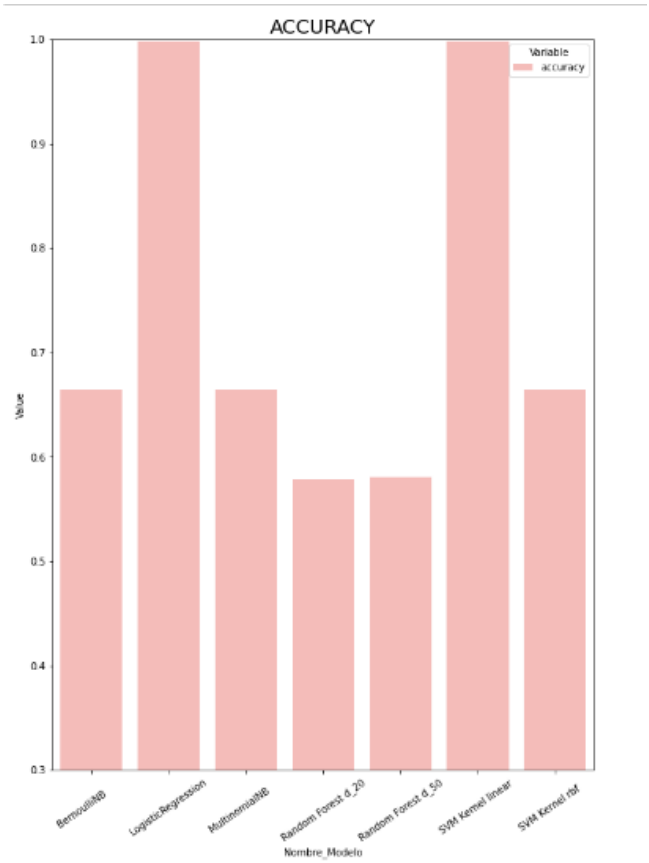


Figura 38: Gráfica del rendimiento de los modelos sobre la métrica del *accuracy*

Por último se realizó una función de matriz de confusión para ser utilizada para ser llamada y visualizar los resultados de las pruebas con el conjunto de *testing*. Se visualizaron los datos en una gráfica tipo *heatmap*.

Funcion para crear el heatmap de las matrices de confusión

Sus parámetros:

```
cm -> Confusion Matrix
classes -> polaridad -1, 0, 1
title -> Titulo de la Gráfica
cmap -> Paleta de colores de la gráfica
```

Figura 39: Parámetros de la función de plot confusion matrix

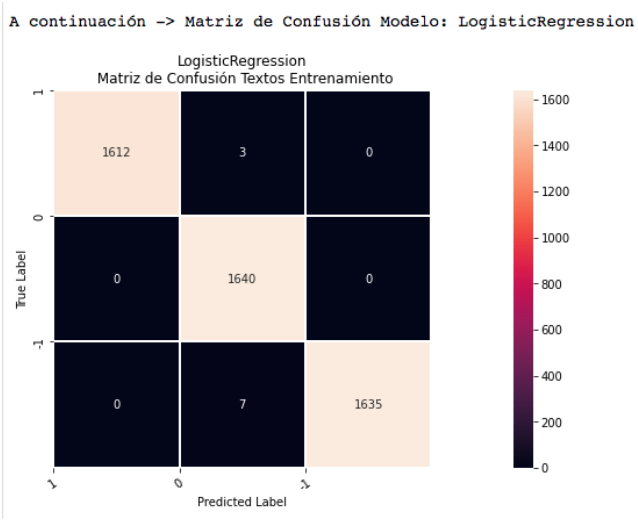


Figura 40: Gráfica del rendimiento de los modelos sobre la métrica del *accuracy*

De acuerdo a los resultados de los modelos, se escogió utilizar el modelo de *Support Vector Machines*, de acuerdo con Keeling et al., 2019, el modelo de *Support Vector Machines*, tiene una leve ventaja contra el modelo de *LogisticRegression*, en el ámbito de clasificación de textos *multiclase*. Debido a esto, se dicidió usar *SVM* como modelo de análisis de emociones.

Como se entrenó al modelo para palabras individuales se tuvo que tomar un arreglo y se iteró sobre cada elemento para aplicar la predicción del modelo. Posteriormente, se tomó el primer elemento para agregarlo al contador.

```
string_prueba = 'Aprendi mucho de esta lectura'
string_prueba = string_prueba.lower()
string_prueba = string_prueba.split(" ")
string_prueba
for i in string_prueba:
    print(i)
string_prueba = np.array(string_prueba)
print(type(string_prueba))
print(string_prueba)
```

```
aprendi
mucho
de
esta
lectura
<class 'numpy.ndarray'>
['aprendi' 'mucho' 'de' 'esta' 'lectura']
```

```
contador_ponderación = 0
for i in string_prueba:
    cantidad = svm_kernel_lin.predict(i)
    contador_ponderación += cantidad[0]
print(contador_ponderación)
```

```
[1]
[1]
[0]
[0]
[0]
2
```

Figura 41: Aplicación del modelo de *SVM*, sobre un comentario

6.20. Visualización

Se realizó una correcta conexión a la base de datos elaborada en el módulo de *backend*. Con esta conexión creada, se pudo tomar las tablas de la base de datos como un conjunto de datos, o crear consultas *SQL* personalizadas.

Se creó un flujo de datos en *Tableau Prep*; su origen fue de dos consultas a la base de datos. La primera fue sobre el la tabla de *session*, ya que contenía toda la información sobre uso del lector por parte de los usuarios estudiantes, la segunda consulta fue sobre el conjunto de tablas que contenían información de las encuestas que se realizaban a los usuarios. Ya que ambos campos poseían la misma columna de usuario, se realizó una unión por columnas *join*. Este *join* se realizó después de limpiar los datos de ambas consultas.

Una vez que se creó el flujo de datos deseado, se importó el archivo *.hyper* de la salida a *Tableau*, para realizar las tablas cruzadas, gráficas necesarias para la creación del *dashboard*.

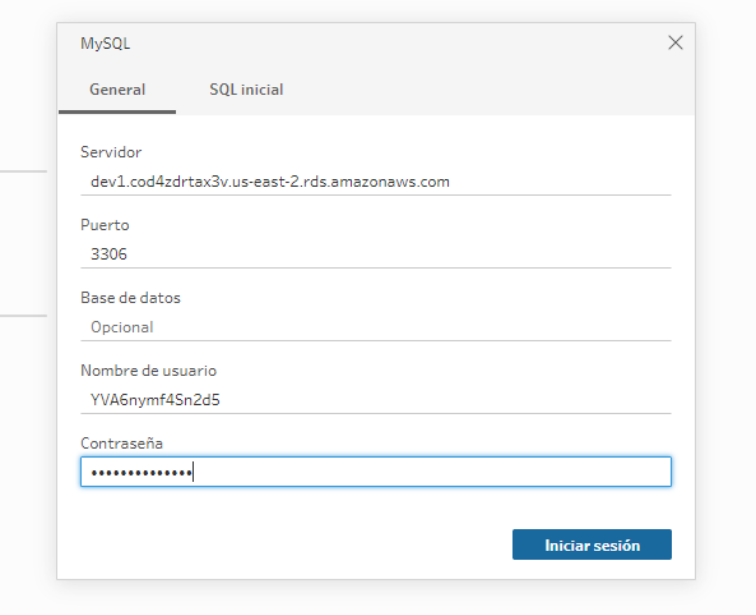


Figura 42: Imagen de la conexión a la base de datos desde *Tableau Prep*

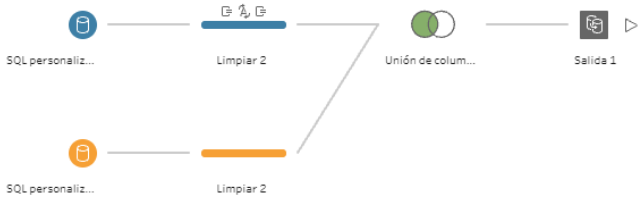


Figura 43: Imagen del flujo de datos creado en *Tableau Prep*

Las primeras visualizaciones creadas permitieron visualizar diversas métricas provenientes del conjunto de datos y de la encuesta. Además, se pudo mostrar la puntuación de los usuarios estudiantes en las primeras dos preguntas de comprensión de lectura, para posteriormente revisarlas.

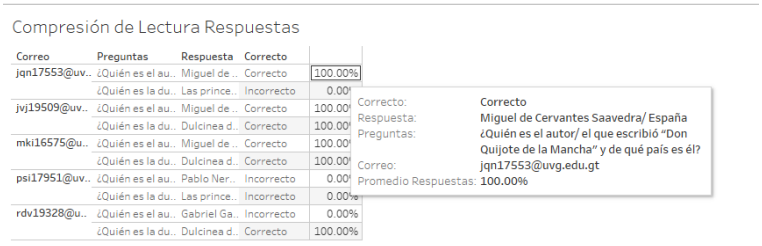


Figura 44: Imagen de la tabla de texto de compresión de lectura

Otra de las tablas importantes es la visualización del *engagement* de los usuarios es-

tudiantes con el lector digital. Muestra la hora de inicio y fin, la fecha de inicio y fin, la cantidad de tiempo que se pasó leyendo por sesión en minutos, y la cantidad de palabras que puede leer por minuto.

Tiempo en lector

			Fecha Inicio / Fecha Fin			
			Tiempo invertido		Palabra por Minuto	
			30	31	30	31
Correo	Hora Inicial	Hora Final	31	31	31	31
jvj19509@uvg.edu.gt	00:13:34	01:19:34		66		597.5
	23:51:16	02:15:16	144		565.4	
psi17951@uvg.edu.gt	16:12:18	18:39:18		147		334.5
rdv19328@uvg.edu.gt	04:49:44	06:13:44		84		323.9

Figura 45: Imagen de la tabla de texto de retroalimentacion de la lectura

Dentro de las encuestas a los usuarios catedráticos, se encontró una característica que se deseará tener en la aplicación: es la de ver la participación de los alumnos. Los catedráticos explicaron que es un tema complicado de manejar dentro del salón de clases y mucho más complicado en actividades como proyectos que se realicen fuera del salón de clase. Es un tema que les interesaría verlo plasmado.

Participación

Correo	
jqn17553@uvg.edu.gt	0
jvj19509@uvg.edu.gt	1
mki16575@uvg.edu.gt	0
psi17951@uvg.edu.gt	1
rdv19328@uvg.edu.gt	1

Figura 46: Imagen de la tabla de participación de acuerdo a los datos recolectados de la base de datos

El último paso fue agregar todas las tablas y gráficas, dentro de un «nuevo *dashboard*», lo primero que se pudo observar en este *dashboard* fueron los correos de los usuarios estudiantes y el filtro de participación.

Se contó con todos los datos sobre una misma fuente de datos, por lo que la interactividad del tablero se pudo mantener y, por ende, se obtuvo dos maneras de filtrar. La primera fue

a través del segmento de filtración por usuarios que participaron o que no participaron. El segundo se pudo obtener presionando algún valor del *dashboard*, este automáticamente permitiría mostrar todos los datos que tengan coincidencia con el seleccionado.

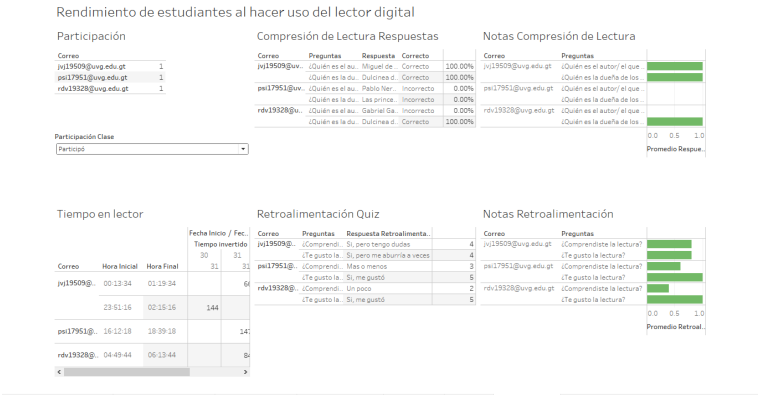


Figura 47: Imagen del tablero siendo afectado por la selección de los usuarios que participaron

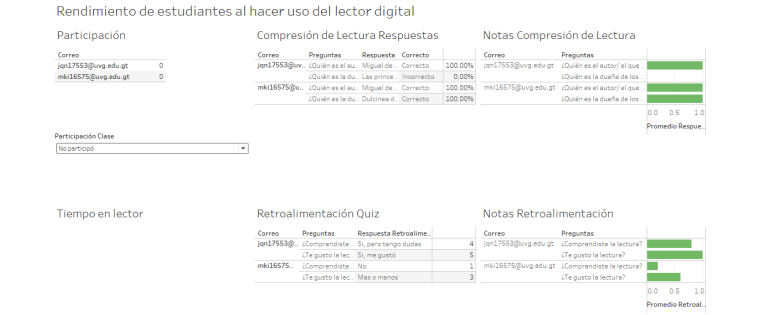


Figura 48: Imagen del tablero siendo afectado por la selección de los usuarios que no participaron

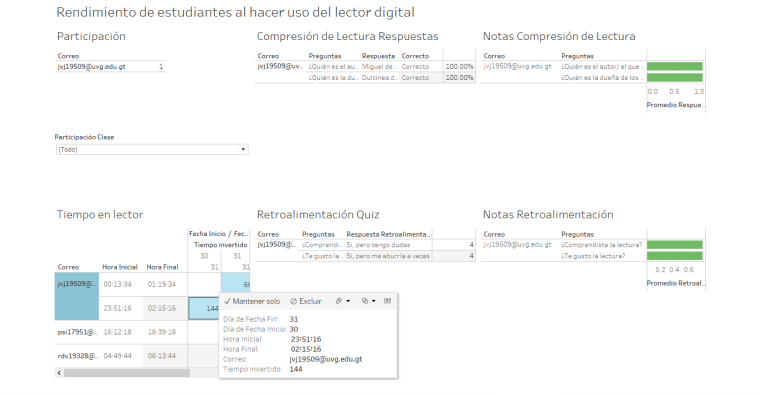


Figura 49: Imagen de la tabla de participación de acuerdo con los datos recolectados de la base de datos

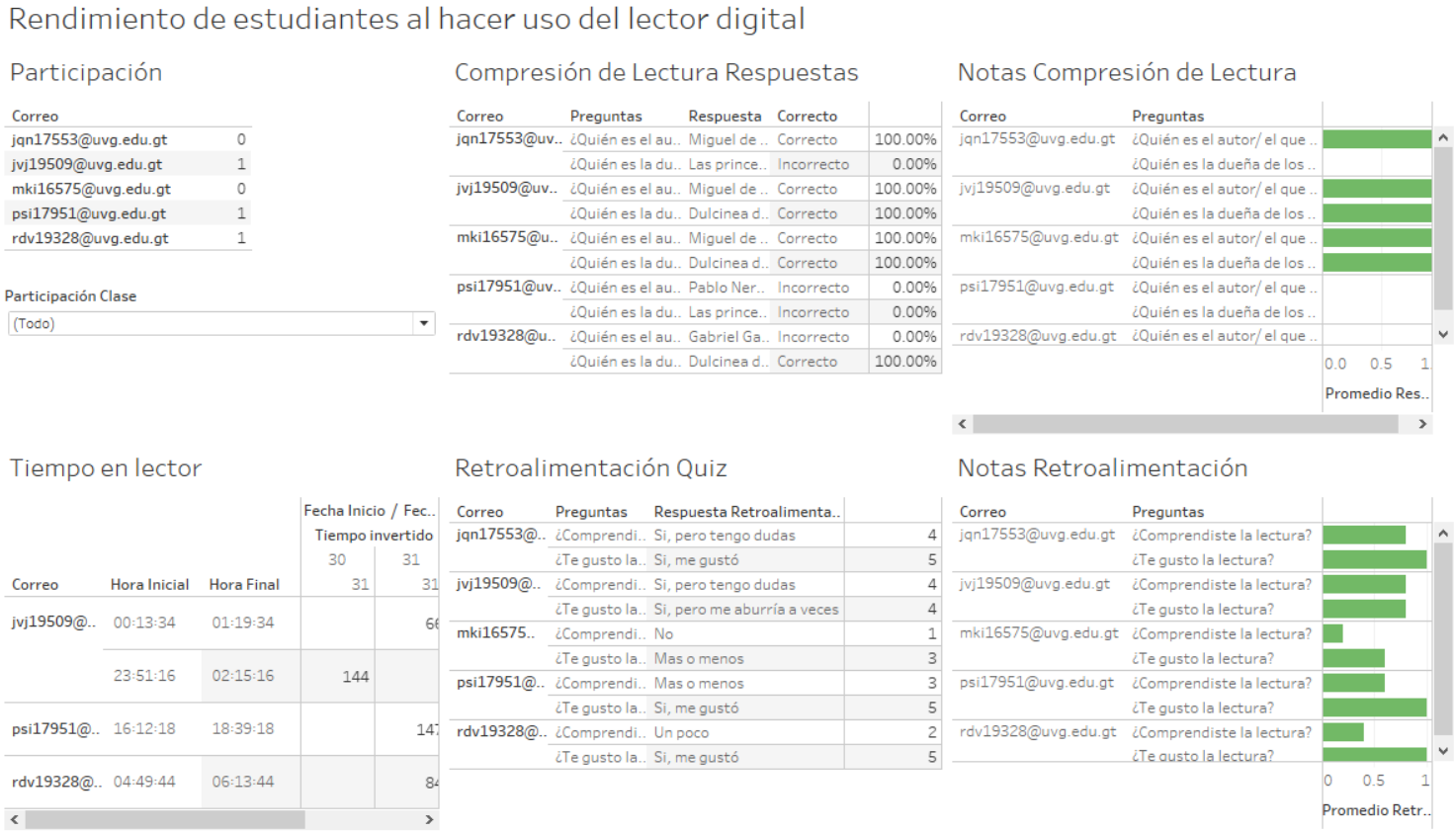


Figura 50: Dashboard de datos recolectados de la base de datos

CAPÍTULO 7

Resultados

7.1. Definición de usuarios

A través de la definición de usuarios se puede genera un perfil sobre quién utilizará la aplicación y qué es lo que buscan. A pesar de que el alcance del módulo de *frontend* se limita a desarrollar la interfaz para estudiantes, también se incluye el perfil de usuario de un catedrático por si se desea en un futuro aumentar el alcance de este proyecto.

7.1.1. Mapa de empatía

A continuación se muestran los mapas de empatía elaborados para catedráticos y estudiantes. Se utilizó una plantilla presente en *Google Drawings*.

Estudiantes

El mapa de empatía de un estudiante puede ser observado en la Figura 51. Este fue realizado con base en las 10 entrevistas iniciales realizadas a estudiantes, así como con las encuestas respondidas. Fue refinado con las entrevistas de prototipos y también se complementó con las técnicas de investigación contextual.

De este mapa de empatía se destacan los siguientes aspectos:

- Los estudiantes se distraen mucho, lo que los hace propensos a desorganizarse y pensar que no tienen tiempo.

- Usualmente ve el material de sus cursos en una pantalla (específicamente una computadora tipo *laptop*).

- Utiliza su computadora portátil para estudiar porque es más cómodo.

- Por lo general, lee de forma digital el material de sus cursos debido a que los libros físicos son caros y puede conseguir documentos *PDF* del material.

- Considera que estudiar es una forma de mejorar su desempeño y al mismo tiempo, cree que contar con una carrera universitaria es un punto clave para sus metas a largo plazo.

- Le estresa no entender todo en clase y en ocasiones, por la desorganización (observado por medio de la investigación contextual) pierde las grabaciones de sus clases y sus libros electrónicos, lo cual aumenta su estrés y frustración.

- Le molesta cuando no todos ponen de su parte en trabajos grupales y se preocupa por sus notas.

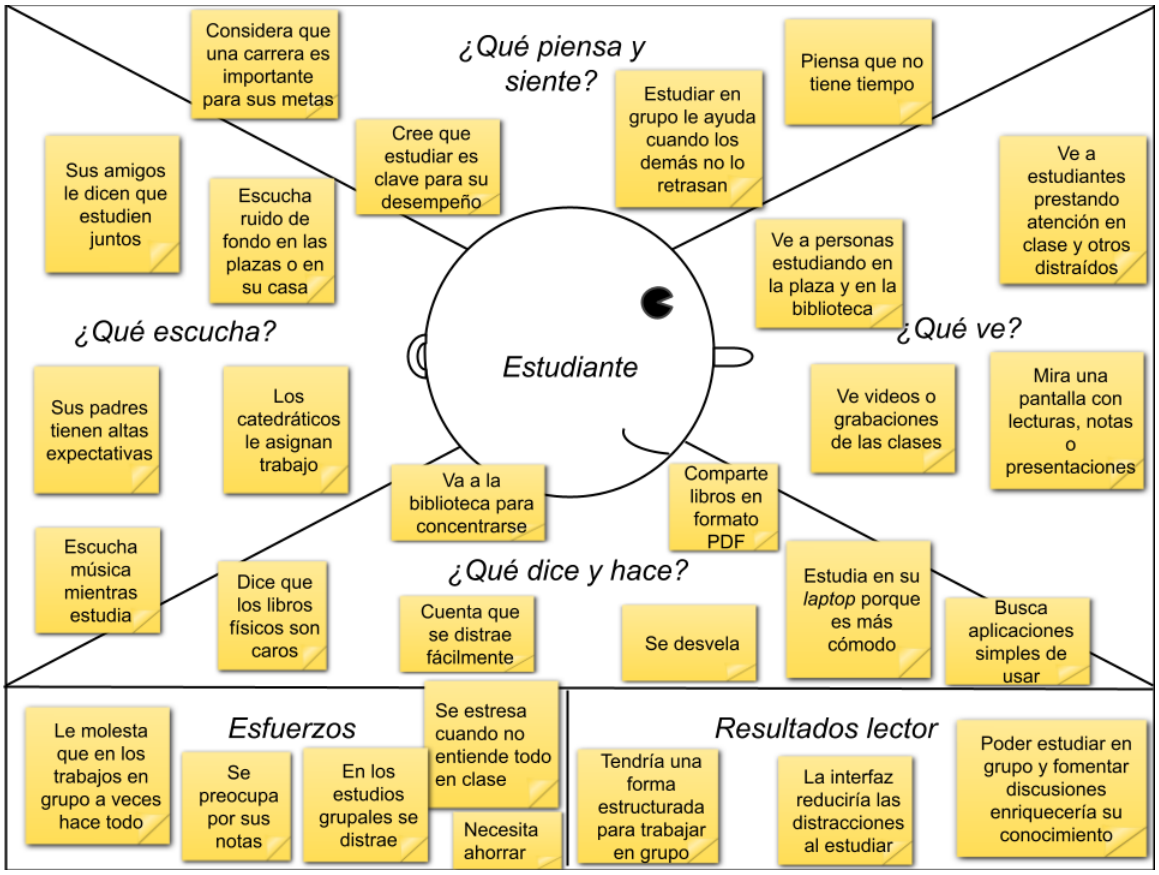


Figura 51: Mapa de empatía de estudiante

Catedráticos

El mapa de empatía de un catedrático puede ser observado en la Figura 52. Este fue realizado con base en las 5 entrevistas iniciales realizadas a catedráticos, así como las encuestas respondidas. También se complementó con las técnicas de investigación contextual.

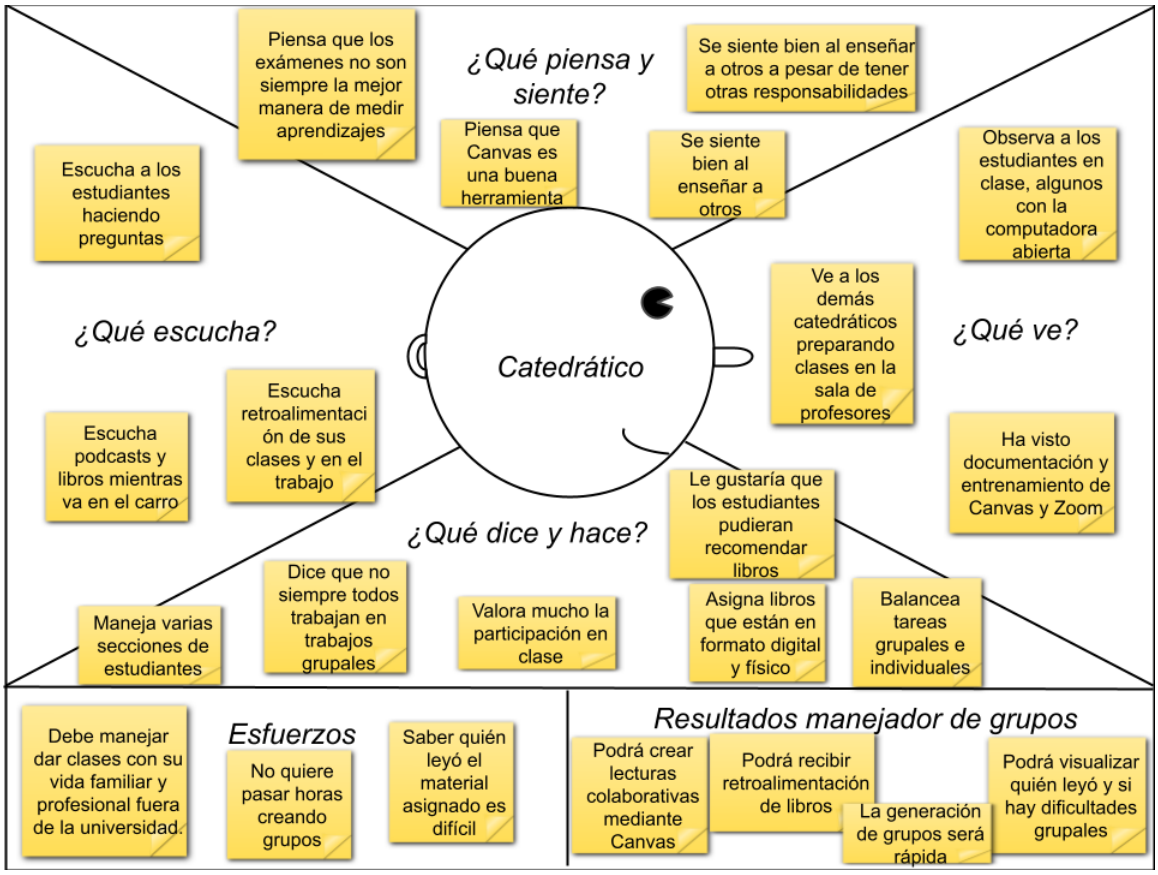


Figura 52: Mapa de empatía de catedrático

De este mapa de empatía se destacan los siguientes aspectos:

- Los catedráticos se encuentran bajo mucha presión y no cuentan con mucho tiempo al tener que balancear 3 áreas principales: su familia, su trabajo fuera de la universidad y su trabajo como catedrático.
- Tiene conocimiento sobre que a veces no todos trabajan en grupos por lo que le gustaría poder ver una estadística sobre esto.
- Le gustaría poder ver quién lee y quién no lee el contenido de clase, así como recibir retroalimentación sobre este contenido.
- Para familiarizarse con herramientas ha utilizado tutoriales de *Zoom* y *Canvas*, proporcionados por la Dirección General de Innovación y Tecnologías para el Aprendizaje (DITA) de la UVG.
- En sus cursos utiliza libros que están en formato digital y físico.

- Balancea tareas individuales y grupales.
- Da clases porque es algo que le gusta y su meta es inspirar a otros, no necesariamente como un trabajo a tiempo completo, aunque hay catedráticos que solamente se dedican a dar clases.

7.1.2. Personaje de usuario

Utilizando los mapas de empatía y la fase de empatizar de *design thinking*, se generaron los personajes de usuario para estudiantes y catedráticos. Para la fotografía de la persona en el personaje de usuario se utilizó una imagen generada por medio de la aplicación *This Person Does Not Exist* (Hellsten y Karras, s.f.).

Personaje de usuario de estudiante

Este puede ser observado en la Figura 53. De este perfil se puede definir a un estudiante típico como un adulto joven que es muy curioso. Utiliza principalmente su computadora para estudiar, pero se distrae mucho en clase y en sesiones grupales de estudio. Se distrae mucho por su teléfono y busca que las aplicaciones educativas con las que interactúa sean simples de utilizar. Prefiere algo que tenga menos funciones, pero que sea sencillo de usar y que se encuentre pulido. Le duelen los ojos después de leer mucho tiempo en la computadora y cuando lee por placer prefiere los libros físicos debido a que son tangibles.

Personaje de usuario de catedrático

El personaje de usuario de catedrático puede ser observado en la Figura 54. De este perfil se puede definir a un catedrático típico como un adulto que busca dejar un legado positivo a los demás. Considera que el trabajo grupal es útil, pero no siempre sabe quiénes trabajaron. Actualmente no tiene forma de saber qué estudiantes han leído el material del curso ni una forma de recibir retroalimentación sobre la literatura utilizada. Busca una aplicación simple de usar que no le quite mucho tiempo.

7.1.3. Elementos clave para introducir una aplicación en la universidad

Si bien el alcance de este proyecto no incluye una implementación del lector en la Universidad del Valle de Guatemala, se dejan a continuación datos clave que podrían ser útiles si en un futuro se desea realizarlo como parte de un proyecto de graduación. No se detallan los pasos específicos para introducir una nueva aplicación a la universidad para evitar filtrar datos confidenciales.

Se recomienda tomar en cuenta los siguientes aspectos:

- *Canvas* es actualmente una plataforma clave para la universidad y se utiliza como algo central en su funcionamiento. La implementación de este lector se podría realizar por

Jan Estudiante

Aspectos demográficos

- Tiene 20 años
- Vive en la Ciudad de Guatemala
- No tiene ingresos propios, pero sus papás la ayudan
- Es de clase media alta

Retos

- Se distrae mucho en clase
- Cuando estudia en grupo se pone a hablar con los demás
- Cuando no entiende se estresa y deja todo
- Su celular la distrae mucho
- Se frustra cuando no le sale algo

Identificadores

- Utiliza la red social Instagram, WhatsApp y TikTok
- Pasa mucho tiempo en Netflix y Disney+
- Le gusta ir con sus amigos al cine o de fiesta
- Utiliza su *laptop* para estudiar

¿Qué podemos hacer?

- Hacer un lector simple de usar para reducir su frustración.
- Realizar una interfaz pulida que no la distraiga al momento de estudiar.
- Generar un programa que le permita tener una guía para estudiar en grupo.

Historia

- Jan es estudiante de licenciatura dentro de la Universidad del Valle de Guatemala.
- No tiene hijos y vive con sus padres.
- Se graduó de un colegio privado.
- Le gusta salir con sus amigos, ver series y nadar.
- Es muy curiosa.

Metas

- Desea graduarse de la universidad
- Quiere fundar su propia empresa con los conocimientos de su carrera
- Se propuso dormir lo suficiente al organizar su horario
- Quiere ser una nadadora profesional

Cosas que ha dicho

- "Soy muy curiosa"
- "Prefiero tener menos funcionalidades en un programa, pero que sea fácil de usar"
- "Me duelen los ojos después de leer mucho en la compu"
- "Para la universidad leo casi solo en digital, pero cuando leo por placer uso libros físicos"
- "Los libros físicos son caros"
- "Uso la compu para estudiar porque el teléfono me distrae y es muy pequeño"
- "Busco que las aplicaciones que uso para estudiar sean simples"
- "Quiero hacer algo grande con mi vida. Cuando me gradúe quiero usar esos conocimientos para empezar mi propia empresa"
- "Estudiar me permite enriquecer mi conocimiento y ser una mejor persona al superar mis propias expectativas"

Razones por las que no utilizaría el lector

- Es muy difícil de usar
- No tiene los libros que busca
- No se puede utilizar en la computadora
- No se puede resaltar ni comentar
- El texto del libro no se puede seleccionar
- No sabe dónde están las funcionalidades
- La interfaz la confunde
- No es amigable para su daltonismo
- La letra no se distingue del fondo

Figura 53: Personaje de usuario de estudiante

Ben Catedrático

Historia

- Ben estudió en la Universidad del Valle de Guatemala y sacó maestrías en el extranjero
- Tiene una esposa y 1 hijo
- Le gusta escuchar podcasts
- Va al gimnasio
- Trabaja además de dar clases

Aspectos demográficos

- Tiene 45 años
- Es de clase media alta
- Vive en la Ciudad de Guatemala
- Trabaja como catedrático y como gerente en una empresa privada

Identificadores

- Utiliza la red social WhatsApp, Facebook e Instagram
- Escucha podcasts mientras va en el carro
- Le gusta salir con sus amigos o familia a restaurantes
- Utiliza su *laptop* para calificar en Canvas

Retos

- Tiene que balancear su vida familiar, profesional y académica
- En los trabajos grupales no sabe quién trabajó
- No tiene una forma de saber qué tan buenos son los libros que usa

¿Qué podemos hacer?

- Optimizar la generación de grupos colaborativos para que sea algo rápido de hacer
- Proporcionar estadísticas sobre los estudiantes que leyeron y cómo leyeron
- Establecer una forma de calificar libros y que los estudiantes recomienden lecturas

Metas

- Desea retirarse con buenos ahorros
- Quiere invertir lo suficiente en bienes raíces para contar con un ingreso pasivo significativo
- Se propuso ir al gimnasio 3 veces por semana
- Quiere pasar más tiempo con su familia
- Desea transmitir su conocimiento y dejar una huella de bien en los estudiantes

Cosas que ha dicho

- "Doy clases porque me gusta. Llevo 14 años haciéndolo y es algo que gracias a Dios se me ha dado bien"
- "La forma de crear grupos en Canvas es muy buena"
- "La biblioteca quitó algunas librerías porque nadie las usaba"
- "Tengo que balancear mi vida familiar, dar clases y trabajar… hago un esfuerzo extra por dar clases porque me gusta"
- "Quiero poder compartir un poco de todo lo que pude aprender en el extranjero"
- "Creo que dar clases es una forma de cambiar a Guatemala para el bien. Hay un gran porcentaje de jóvenes en Guatemala y con la educación se puede hacer la diferencia"

Razones por las que no utilizaría el manejador del lector

- Es muy difícil de usar
- No tiene los libros que necesita para su clase
- Crear grupos es muy tardado
- Crear grupos es muy complicado
- Las estadísticas que le da el programa no le sirven de nada
- No se integra con Canvas
- No hay capacitación para utilizar la herramienta
- Dificulta el proceso de dar clases

Figura 54: Personaje de usuario de catedrático

medio del *REST API* proporcionado por *Canvas* y con un enlace a la biblioteca de la UVG.

- El prestigio es algo muy importante para la universidad, por lo tanto, si se deseara agregar más funcionalidades, estas deben estar pulidas.
- En caso de manejar datos confidenciales, toda la comunicación debe estar cifrada.
- Para implementar *Canvas* la universidad se tardó entre 6 y 8 meses; al ser el lector una aplicación de menor escala, se estima que el proceso de implementación sería menor.
- La universidad utiliza tanto estadísticas micro como macro para evaluar el rendimiento de aplicaciones.
- Es clave contar con buen soporte y documentación para capacitar a los usuarios al momento de implementar una nueva aplicación.
- Se busca que haya un soporte continuo de la aplicación.

7.2. Paleta de colores

La paleta de colores fue evolucionando a lo largo de los prototipos. Inicialmente se probó con paletas distintas a la de la Universidad del Valle de Guatemala para dar más libertad a los entrevistados de sugerir colores. Sin embargo, conforme fueron avanzando los prototipos se evidenció que los colores de la universidad gustaron a los entrevistados. Además, al ser un lector enfocado a la UVG, se decidió utilizar una paleta de colores alineada con la universidad.

7.2.1. Colores generales

Los colores generales son: verde, gris y blanco. Estos cuentan con el siguiente código:

- Verde UVG: #008F2E
- Verde botón: #1ED072
- Gris: #3B3A3C
- Blanco: #F8F8F8

El blanco no es blanco puro (el blanco puro tiene el código #FFFFFF) para reducir la fatiga visual de ver texto negro sobre un fondo blanco. Los colores pueden ser vistos en la Figura 55. Para los botones no se utilizó el verde UVG debido a que al poner texto blanco o negro no se obtiene un contraste mayor a 7, que es requerimiento para cumplir con las normas de *ADA* y *WCAG* AAA.

Figura 55: Colores base de la aplicación

7.2.2. Comentarios y resaltadores

Para definir los colores de comentarios y resaltadores se utilizó como base los resultados de las pruebas de resaltadores en hojas reales.

Cuadro 4: Contraste entre resaltadores y fondo de página en prueba estandarizada

Marca	Color	International Paper Chamex	DaVinci Sketchbook	Scribe Ecológico	Promedio
Faber Castell Textliner 40 Refill	Verde	1.06	1.13	1.26	1.15
	Amarillo	1.38	1.19	1.18	1.25
Zebra Midliner	Verde	1.98	1.74	2.23	1.98
Kores High Liner Plus Pastel	Morado	1.57	1.15	1.44	1.39
Pelikan Textmarker Flash	Rosado	1.56	1.33	1.37	1.42
	Verde	1.45	1.38	1.53	1.45
Stabilo Boss Original	Verde	1.26	1.27	1.35	1.29
	Azul	1.37	1.39	1.50	1.42
	Amarillo	1.19	1.12	1.20	1.17
	Anaranjado	1.43	1.29	1.44	1.39
Fast R300	Anaranjado	1.31	1.23	1.25	1.26
Sharpie Fine Point Permanent Marker	Negro	14.57	11.98	12.24	12.93
Staedtler Permanent Lumcolor	Negro	13.87	13.33	13.03	13.41

Cuadro 5: Contraste entre resaltadores y fondo de página en prueba de luz amarilla

Marca	Color	International Paper Chamex	DaVinci Sketchbook	Scribe Ecológico	Promedio
Faber Castell Textliner 40 Refill	Verde	1.77	1.49	1.80	1.69
	Amarillo	1.04	1.01	1.02	1.02
Zebra Midliner	Verde	2.13	2.01	2.33	2.16
Kores High Liner Plus Pastel	Morado	1.70	1.26	1.58	1.51
Pelikan Textmarker Flash	Rosado	1.45	1.29	1.45	1.40
	Verde	1.62	1.50	1.92	1.68
Stabilo Boss Original	Verde	1.46	1.44	1.79	1.56
	Azul	1.58	1.74	1.84	1.72
	Amarillo	1.20	1.13	1.17	1.17
	Anaranjado	1.29	1.32	1.36	1.32
Fast R300	Anaranjado	1.34	1.22	1.36	1.31
Sharpie Fine Point Permanent Marker	Negro	8.31	7.89	8.12	8.11
Staedtler Permanent Lumcolor	Negro	7.42	7.34	8.18	7.65

Se puede observar en el Cuadro 4 que el promedio de contraste más bajo (1.15) se encuentra en el resaltador *Faber Castell Textliner 40 Refill* en la prueba estandarizada, mientras que en el Cuadro 5 se puede observar que el resaltador *Faber Castell Textliner 40 Refill* en amarillo contó con el contraste más bajo (1.02). El promedio global de contraste se muestra a continuación:

- Prueba estandarizada: 1.38
- Prueba de luz amarilla: 1.50

Tomado estos contrastes como base, se definió que los contrastes de la paleta de resaltadores con respecto al fondo de los temas del lector debían superar como mínimo 1.15 en al menos la mitad de fondos. Además, no se tomaría como válido ningún color con un contraste inferior a 1.02 (que es el peor contraste promedio de la prueba en luz amarilla). Para una paleta de alto contraste, se definió que esta debía contar con un contraste promedio superior a 1.50 (el promedio de contraste más alto entre la prueba estandarizada y la prueba de luz amarilla).

Comentarios

Al analizar los resultados de resaltadores en papel, se diseñó una paleta de colores sólida, que fue utilizada para todos los elementos de color sólido: como comentarios y personas activas en el documento. Los resultados de esta paleta pueden ser observados en la Figura 56. Los contrastes de estos colores con diferentes fondos pueden ser observados en el anexo 11.4. Estos colores son:

- Verde: #A9F071
- Amarillo: #CBC300
- Celeste: #76D1EE
- Rosado: #CBC300
- Verde oscuro: #76B496

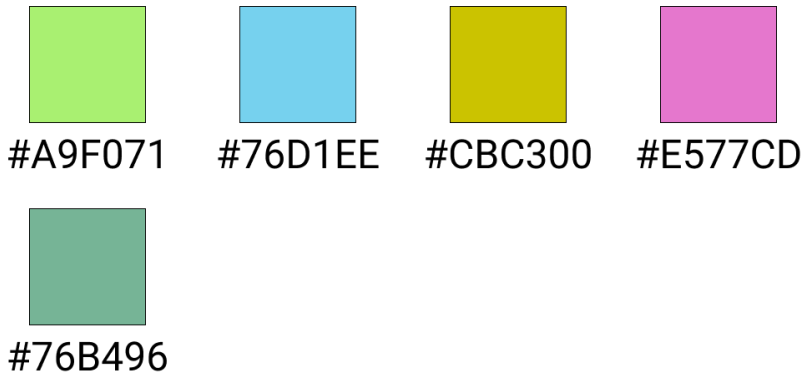


Figura 56: Paleta de colores de comentarios definida

Resaltadores

Se intentó utilizar la misma paleta definida para comentarios al momento de implementar el resaltado programado, sin embargo, debido a la transparencia de los resaltadores no era

muy claro, por lo que se ajustó la luminosidad de los colores para hacerlos más oscuros. Al hacer esto, los colores de resaltado fueron más claros. Los resultados de la paleta oscurecida pueden ser observados en la Figura 57. Esta fue definida de la siguiente forma:

1. Verde: #478F0F
2. Amarillo: #8A8400
3. Celeste: #8A8400
4. Rosado: #D831B4
5. Verde oscuro: #4E8D6F

Cabe mencionar que los números de cada color corresponden a los números de resaltador en el Cuadro 6.

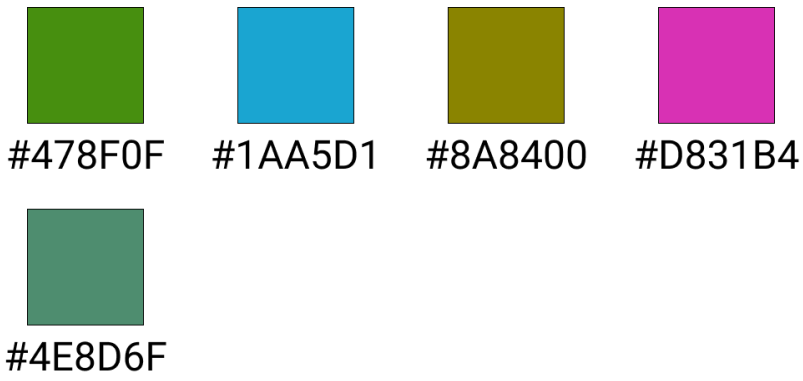


Figura 57: Paleta de colores de resaltado definidas

No obstante, para identificar elementos de color sólido en la aplicación se utilizó la paleta inicial, debido a que al no tener transparencia, esta paleta funciona bien según los estándares de *ADA* y *WCAG*, con respecto a texto de color negro. Los resultados de la paleta sólida y la paleta utilizada con transparencia pueden ser observados en la Figura 58 y la comparación entre el uso de ambas paletas en un libro puede ser observado en la Figura 60.

(a) Paleta inicial. (b) Paleta final.

Figura 58: Paletas de resaltadores

Cuadro 6: Contraste entre la paleta de resaltadores y fondos de libros

Color de fondo	**Resaltador**	**Condición visual**					
		Normal	**Protanopia**	**Deuteranopia**	**Tritanopia**	**Monocromía del cono azul**	**Achromatopsia**
Claro	1	1.41	1.42	1.41	1.42	1.45	1.47
	2	1.31	1.32	1.31	1.32	1.35	1.37
	3	1.38	1.39	1.38	1.39	1.40	1.41
	4	1.50	1.49	1.49	1.50	1.47	1.44
	5	1.37	1.38	1.37	1.37	1.39	1.41
Libro	1	1.43	1.43	1.43	1.42	1.46	1.35
	2	1.34	1.34	1.35	1.33	1.36	1.26
	3	1.40	1.40	1.41	1.40	1.41	1.30
	4	1.52	1.50	1.52	1.51	1.48	1.32
	5	1.39	1.39	1.41	1.39	1.40	1.30
Gris	1	1.15	1.14	1.15	1.15	1.10	1.08
	2	1.29	1.28	1.28	1.30	1.22	1.20
	3	1.18	1.17	1.19	1.18	1.15	1.14
	4	1.07	1.07	1.08	1.07	1.08	1.11
	5	1.29	1.18	1.19	1.19	1.17	1.14
Oscuro	1	1.42	1.41	1.41	1.42	1.35	1.32
	2	1.60	1.60	1.60	1.60	1.50	1.47
	3	1.46	1.46	1.45	1.46	1.41	1.40
	4	1.34	1.36	1.34	1.34	1.32	1.36
	5	1.46	1.45	1.45	1.46	1.43	1.40
Claro (resaltado gris)	1	2.09	2.10	2.09	2.10	2.09	2.09
	2	2.09	2.10	2.09	2.10	2.09	2.09
	3	2.09	2.10	2.09	2.10	2.09	2.09
	4	2.09	2.10	2.09	2.10	2.09	2.09
	5	2.09	2.10	2.09	2.10	2.09	2.09
Libro (resaltado gris)	1	2.10	2.10	2.11	2.10	2.09	2.08
	2	2.10	2.10	2.11	2.10	2.09	2.08
	3	2.10	2.10	2.11	2.10	2.09	2.08
	4	2.10	2.10	2.11	2.10	2.09	2.08
	5	2.10	2.10	2.11	2.10	2.09	2.08

En el Cuadro 6 se puede observar que se cumplió la condición de elaborar paletas de resaltadores con un contraste superior a 1.15 en al menos el 50 % de los fondos utilizados en los libros (los colores de fondo de los libros se pueden observar en la Figura 61). De hecho, el único fondo donde no se cumple la condición es en el fondo gris. Sin embargo, este sigue superando un contraste de 1.02 en todos los colores de resaltado. También se puede observar

en el Cuadro 6 que las paletas de alto contraste: claro (resaltado gris) y libro (resaltado gris) cuentan con un promedio de contraste superior a 1.50. De hecho, estas paletas cuentan con un promedio de contraste superior a 2 entre el color de resaltado y el fondo del libro bajo las condiciones de visión analizadas. Para el color de resaltado de estas paletas se utilizó negro (#000000), debido a que la transparencia del resaltado lo hace ver gris y permite el mayor contraste posible con una hoja clara mientras aún se mantiene un contraste superior a 7 con las letras negras.

Se midió el color de resaltador por cada fondo y tomando en cuenta diferentes condiciones de visión. Al contar con transparencia, se tuvo que medir individualmente cada color por fondo, debido a que no se podía tomar el código hexadecimal de la paleta como base. En la Figura 59 se pueden observar las diferentes tonalidades que adquieren los resaltadores bajo diferentes fondos y condiciones de visión. En el anexo 11.5 se puede observar las diferentes condiciones de visión en los diferentes fondos de libros.

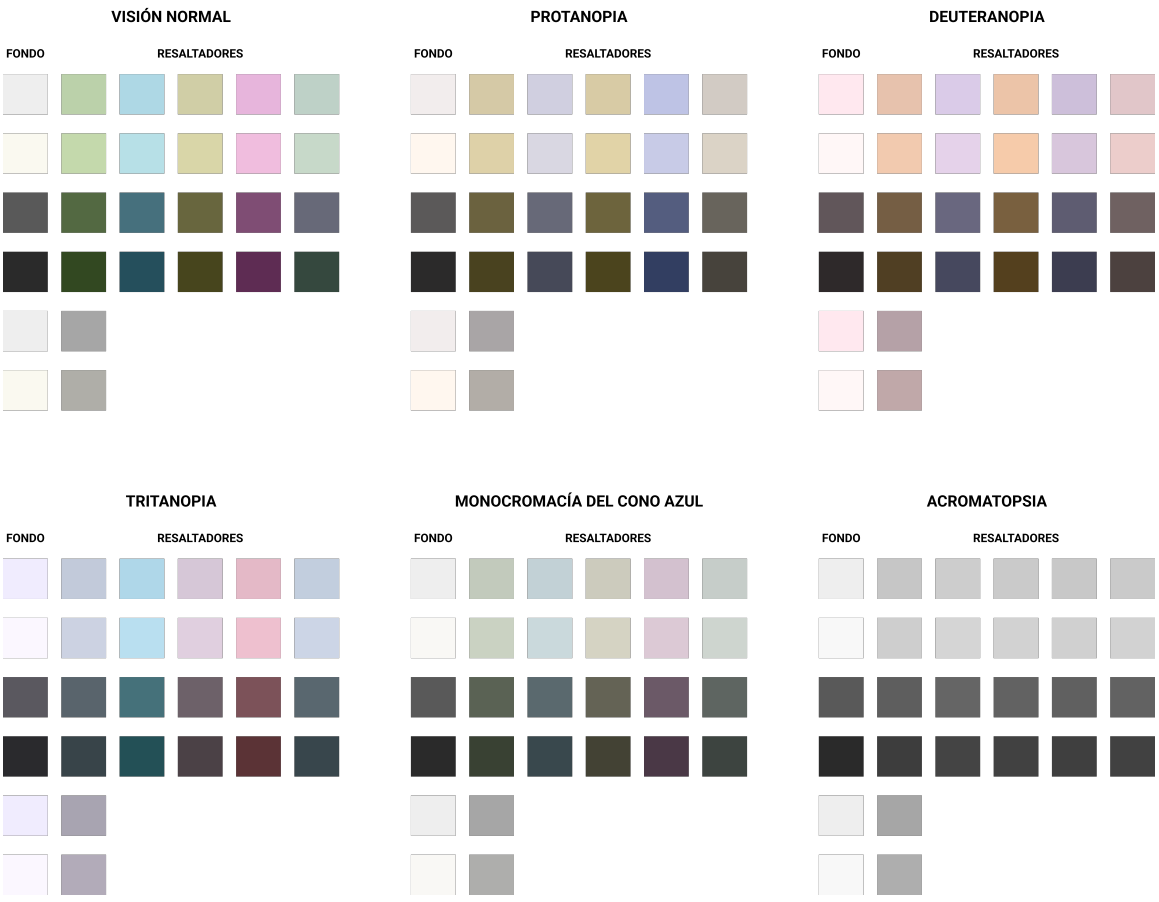
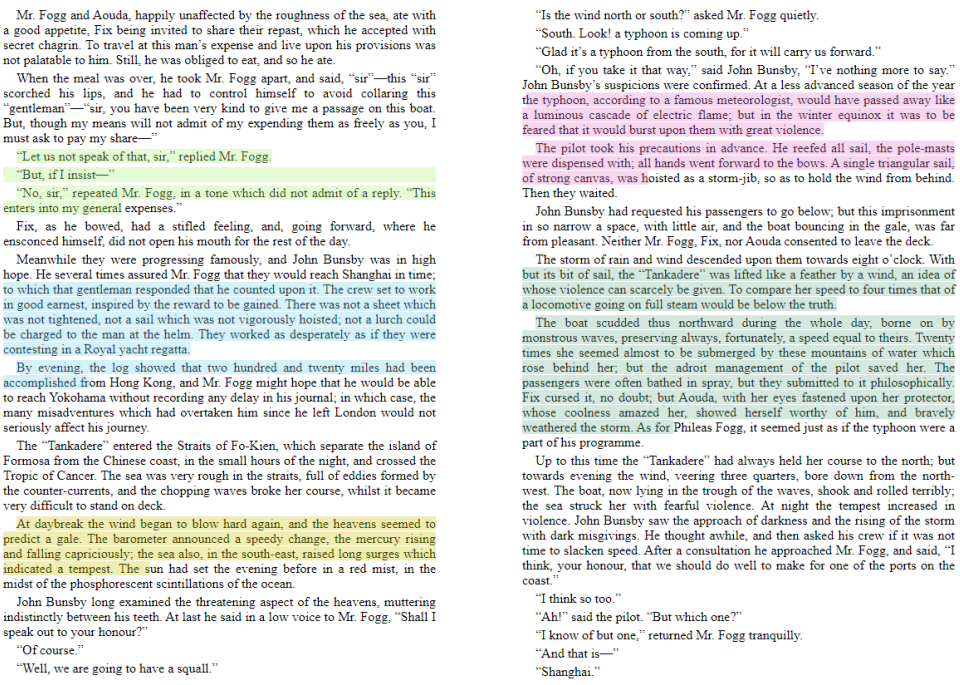


Figura 59: Colores de resaltado bajo diferentes condiciones de visión



Mr. Fogg and Aouda, happily unaffected by the roughness of the sea, ate with a good appetite, Fix being invited to share their repast, which he accepted with secret chagrin. To travel at this man's expense and live upon his provisions was not palatable to him. Still, he was obliged to eat, and so he ate.

When the meal was over, he took Mr. Fogg apart, and said, "sir"—this "sir" scorched his lips, and he had to control himself to avoid collaring this "gentleman"—"sir, you have been very kind to give me a passage on this boat. But, though my means will not admit of my expending them as freely as you, I must ask to pay my share—"

"Let us not speak of that, sir," replied Mr. Fogg.

"But, if I insist—"

"No, sir," repeated Mr. Fogg, in a tone which did not admit of a reply. "This enters into my general expenses."

Fix, as he bowed, had a stifled feeling, and, going forward, where he ensconced himself, did not open his mouth for the rest of the day.

Meanwhile they were progressing famously, and John Bunsby was in high hope. He several times assured Mr. Fogg that they would reach Shanghai in time; to which that gentleman responded that he counted upon it. The crew set to work in good earnest, inspired by the reward to be gained. There was not a sheet which was not tightened, not a sail which was not vigorously hoisted; not a lurch could be charged to the man at the helm. They worked as desperately as if they were contesting in a Royal yacht regatta.

By evening, the log showed that two hundred and twenty miles had been accomplished from Hong Kong, and Mr. Fogg might hope that he would be able to reach Yokohama without recording any delay in his journal; in which case, the many misadventures which had overtaken him since he left London would not seriously affect his journey.

The "Tankadere" entered the Straits of Fo-Kien, which separate the island of Formosa from the Chinese coast, in the small hours of the night, and crossed the Tropic of Cancer. The sea was very rough in the straits, full of eddies formed by the counter-currents, and the chopping waves broke her course, whilst it became very difficult to stand on deck.

At daybreak the wind began to blow hard again, and the heavens seemed to predict a gale. The barometer announced a speedy change, the mercury rising and falling capriciously; the sea also, in the south-east, raised long surges which indicated a tempest. The sun had set the evening before in a red mist, in the midst of the phosphorescent scintillations of the ocean.

John Bunsby long examined the threatening aspect of the heavens, muttering indistinctly between his teeth. At last he said in a low voice to Mr. Fogg, "Shall I speak out to your honour?"

"Of course."

"Well, we are going to have a squall."

"Is the wind north or south?" asked Mr. Fogg quietly.

"South. Look! a typhoon is coming up."

"Glad it's a typhoon from the south, for it will carry us forward."

"Oh, if you take it that way," said John Bunsby, "I've nothing more to say." John Bunsby's suspicions were confirmed. At a less advanced season of the year the typhoon, according to a famous meteorologist, would have passed away like a luminous cascade of electric flame; but in the winter equinox it was to be feared that it would burst upon them with great violence.

The pilot took his precautions in advance. He reefed all sail, the pole-masts were dispensed with; all hands went forward to the bows. A single triangular sail, of strong canvas, was hoisted as a storm-jib, so as to hold the wind from behind. Then they waited.

John Bunsby had requested his passengers to go below; but this imprisonment in so narrow a space, with little air, and the boat bouncing in the gale, was far from pleasant. Neither Mr. Fogg, Fix, nor Aouda consented to leave the deck.

The storm of rain and wind descended upon them towards eight o'clock. With but its bit of sail, the "Tankadere" was lifted like a feather by a wind, an idea of whose violence can scarcely be given. To compare her speed to four times that of a locomotive going on full steam would be below the truth.

The boat scudded thus northward during the whole day, borne on by monstrous waves, preserving always, fortunately, a speed equal to theirs. Twenty times she seemed almost to be submerged by these mountains of water which rose behind her; but the adroit management of the pilot saved her. The passengers were often bathed in spray, but they submitted to it philosophically. Fix cursed it, no doubt; but Aouda, with her eyes fastened upon her protector, whose coolness amazed her, showed herself worthy of him, and bravely weathered the storm. As for Phileas Fogg, it seemed just as if the typhoon were a part of his programme.

Up to this time the "Tankadere" had always held her course to the north; but towards evening the wind, veering three quarters, bore down from the north-west. The boat, now lying in the trough of the waves, shook and rolled terribly; the sea struck her with fearful violence. At night the tempest increased in violence. John Bunsby saw the approach of darkness and the rising of the storm with dark misgivings. He thought awhile, and then asked his crew if it was not time to slacken speed. After a consultation he approached Mr. Fogg, and said, "I think, your honour, that we should do well to make for one of the ports on the coast."

"I think so too."

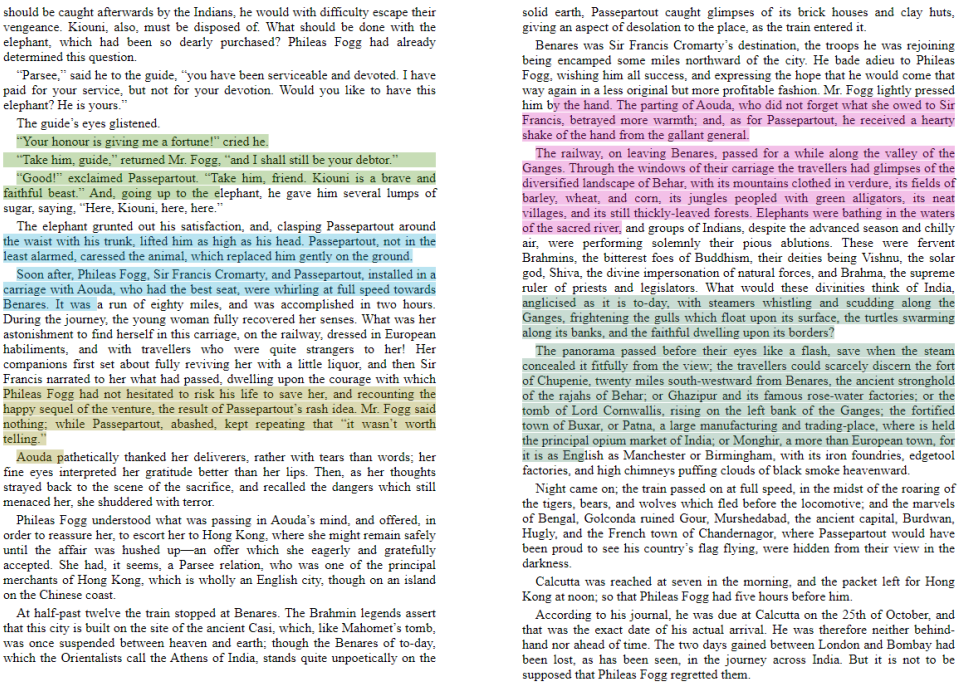
"Ah!" said the pilot. "But which one?"

"I know of but one," returned Mr. Fogg tranquilly.

"And that is—"

"Shanghai."

(a) Paleta sólida.



should be caught afterwards by the Indians, he would with difficulty escape their vengeance. Kiouni, also, must be disposed of. What should be done with the elephant, which had been so dearly purchased? Phileas Fogg had already determined this question.

"Parsee," said he to the guide, "you have been serviceable and devoted. I have paid for your service, but not for your devotion. Would you like to have this elephant? He is yours."

The guide's eyes glistened.

"Your honour is giving me a fortune!" cried he.

"Take him, guide," returned Mr. Fogg, "and I shall still be your debtor."

"Good!" exclaimed Passepartout. "Take him, friend. Kiouni is a brave and faithful beast." And, going up to the elephant, he gave him several lumps of sugar, saying, "Here, Kiouni, here, here."

The elephant grunted out his satisfaction, and, clasping Passepartout around the waist with his trunk, lifted him as high as his head. Passepartout, not in the least alarmed, caressed the animal, which replaced him gently on the ground.

Soon after, Phileas Fogg, Sir Francis Cromarty, and Passepartout, installed in a carriage with Aouda, who had the best seat, were whirling at full speed towards Benares. It was a run of eighty miles, and was accomplished in two hours. During the journey, the young woman fully recovered her senses. What was her astonishment to find herself in this carriage, on the railway, dressed in European habiliments, and with travellers who were quite strangers to her! Her companions first set about fully reviving her with a little liquor, and then Sir Francis narrated to her what had passed, dwelling upon the courage with which Phileas Fogg had not hesitated to risk his life to save her, and recounting the happy sequel of the venture, the result of Passepartout's rash idea. Mr. Fogg said nothing; while Passepartout, abashed, kept repeating that "it wasn't worth telling."

Aouda pathetically thanked her deliverers, rather with tears than words; her fine eyes interpreted her gratitude better than her lips. Then, as her thoughts strayed back to the scene of the sacrifice, and recalled the dangers which still menaced her, she shuddered with terror.

Phileas Fogg understood what was passing in Aouda's mind, and offered, in order to reassure her, to escort her to Hong Kong, where she might remain safely until the affair was hushed up—an offer which she eagerly and gratefully accepted. She had, it seems, a Parsee relation, who was one of the principal merchants of Hong Kong, which is wholly an English city, though on an island on the Chinese coast.

At half-past twelve the train stopped at Benares. The Brahmin legends assert that this city is built on the site of the ancient Casi, which, like Mahomet's tomb, was once suspended between heaven and earth; though the Benares of to-day, which the Orientalists call the Athens of India, stands quite unpoetically on the solid earth, Passepartout caught glimpses of its brick houses and clay huts, giving an aspect of desolation to the place, as the train entered it.

Benares was Sir Francis Cromarty's destination, the troops he was rejoining being encamped some miles northward of the city. He bade adieu to Phileas Fogg, wishing him all success, and expressing the hope that he would come that way again in a less original but more profitable fashion. Mr. Fogg lightly pressed him by the hand. The parting of Aouda, who did not forget what she owed to Sir Francis, betrayed more warmth; and, as for Passepartout, he received a hearty shake of the hand from the gallant general.

The railway, on leaving Benares, passed for a while along the valley of the Ganges. Through the windows of their carriage the travellers had glimpses of the diversified landscape of Behar, with its mountains clothed in verdure, its fields of barley, wheat, and corn, its jungles peopled with green alligators, its neat villages, and its still thickly-leaved forests. Elephants were bathing in the waters of the sacred river, and groups of Indians, despite the advanced season and chilly air, were performing solemnly their pious ablutions. These were fervent Brahmins, the bitterest foes of Buddhism, their deities being Vishnu, the solar god, Shiva, the divine impersonation of natural forces, and Brahma, the supreme ruler of priests and legislators. What would these divinities think of India, anglicised as it is to-day, with steamers whistling and scudding along the Ganges, frightening the gulls which float upon its surface, the turtles swarming along its banks, and the faithful dwelling upon its borders?

The panorama passed before their eyes like a flash, save when the steam concealed it fitfully from the view; the travellers could scarcely discern the fort of Chupenie, twenty miles south-westward from Benares, the ancient stronghold of the rajahs of Behar; or Ghazipur and its famous rose-water factories; or the tomb of Lord Cornwallis, rising on the left bank of the Ganges; the fortified town of Buxar, or Patna, a large manufacturing and trading-place, where is held the principal opium market of India; or Monghir, a more than European town, for it is as English as Manchester or Birmingham, with its iron foundries, edgetool factories, and high chimneys puffing clouds of black smoke heavenward.

Night came on; the train passed on at full speed, in the midst of the roaring of the tigers, bears, and wolves which fled before the locomotive; and the marvels of Bengal, Golconda ruined Gour, Murshedabad, the ancient capital, Burdwan, Hugly, and the French town of Chandernagor, where Passepartout would have been proud to see his country's flag flying, were hidden from their view in the darkness.

Calcutta was reached at seven in the morning, and the packet left for Hong Kong at noon; so that Phileas Fogg had five hours before him.

According to his journal, he was due at Calcutta on the 25th of October, and that was the exact date of his actual arrival. He was therefore neither behind-hand nor ahead of time. The two days gained between London and Bombay had been lost, as has been seen, in the journey across India. But it is not to be supposed that Phileas Fogg regretted them.

(b) Paleta resaltadores.

Figura 60: Paletas de colores en libro

7.2.3. Estilos de libros

Los estilos de libros se definieron para facilitar su legibilidad entre las letras y el fondo y reducir la fatiga visual al leer. Todos los fondos cuentan con un contraste de al menos 7 entre la letra del texto y el fondo del libro para cumplir con los estándares de *ADA* y *WCAG* AAA. También, por medio de la retroalimentación de usuarios se evitó utilizar negro y blanco para el fondo, para reducir la fatiga de leer en una computadora. Al momento de realizar pruebas de usabilidad, los entrevistados mencionaron que los colores libro y gris fueron los más cómodos para leer. Los colores utilizados para los libros se pueden observar en la Figura 61 y los colores utilizados para el color de letra se pueden observar en la Figura 62

Se utilizaron los siguientes colores para las hojas:

- Claro: #EEEEEE
- Libro: #FAF9F0
- Gris: #595959
- Oscuro: #2A2A2A

Para el color de letra se utilizó la siguiente paleta:

- Temas claros (claro y libro): #000000
- Tema gris: #FFFFFF
- Tema oscuro: #F0F0F0

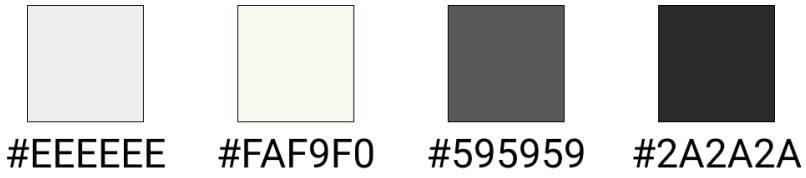


Figura 61: Colores de hojas de libros

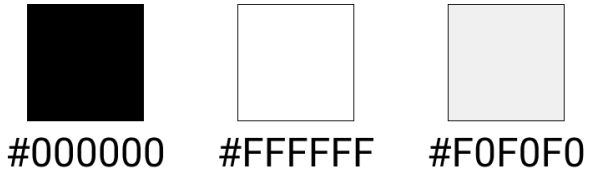


Figura 62: Colores de letra en libros

7.3. Resultados de pruebas de prototipos

A continuación se muestran los resultados de pruebas de elección de prototipos, que fueron una simplificación de las pruebas A/B.

7.3.1. Pruebas de elección de prototipos

Los prototipos comparados se pueden observar en la Figura 63, si se desea ver con más detalle un prototipo individual, estos se encuentran detallados en las figuras: 8 (prototipo 1); 9 (prototipo 2); 10 (prototipo 3) y 11 (prototipo 4) dentro de la metodología.

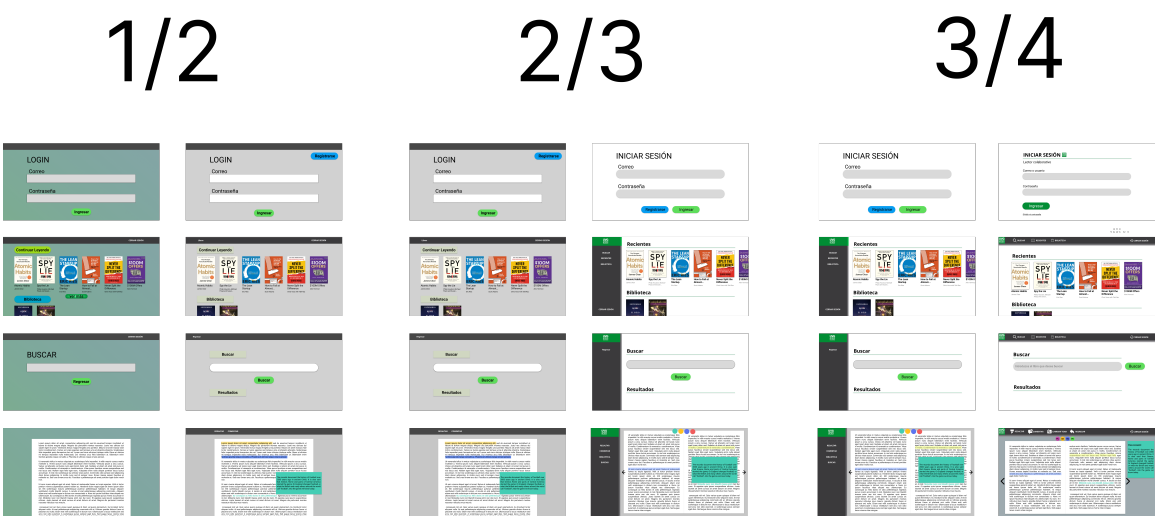


Figura 63: Comparación de prototipos

En el Cuadro 7 se puede observar que el prototipo 2 fue superior al prototipo 1 en todas las pantallas menos en la búsqueda. Al indagar sobre el por qué esto sucedió, los entrevistados respondieron que el título de «buscar» parecía un botón, lo que hacía confusa la pantalla porque parecía que se podían presionar 2 elementos.

Cuadro 7: Cantidad de votos por prototipo 1 y 2

Pantalla	Prototipo 1	Prototipo 2
Inicio de sesión	0	5
Menú	0	5
Búsqueda	3	2
Lector	0	5

Posteriormente, al comparar el prototipo 2 y 3 (Cuadro 8), se pudo observar que el prototipo 3 fue superior en todas las pantallas. Sin embargo, al preguntarle a los entrevistados que favorecieron el lector en el prototipo 2 el por qué detrás de su elección, se obtuvo una respuesta interesante: en ocasiones prefieren leer documentos en una sola hoja debido a que leerlo por medio de 2 hojas puede resultar más cansado si es muy pequeño el lector. Por medio de esta retroalimentación, se implementó un lector de una sola hoja en el prototipo final programado cuando el este tiene un ancho menor al ideal para desplegar 2 hojas.

Cuadro 8: Cantidad de votos por prototipo 2 y 3

Pantalla	Prototipo 2	Prototipo 3
Inicio de sesión	0	5
Menú	1	4
Búsqueda	0	5
Lector	2	3

Finalmente, al comparar los prototipos 3 y 4 (Cuadro 9) se pudo observar que el prototipo 4 fue superior al 3 en todas las pantallas menos en el menú. Esto es debido a que le agradó a los usuarios la idea de un menú lateral en el prototipo 3. Debido a esto, en el prototipo final programado, se implementó un menú lateral con íconos, para evitar un diseño cargado del lado izquierdo.

Cuadro 9: Cantidad de votos por prototipo 3 y 4

Pantalla	Prototipo 3	Prototipo 4
Inicio de sesión	1	4
Menú	3	2
Búsqueda	1	4
Lector	1	4

7.3.2. Tiempos para completar acciones en la aplicación programada

En el Cuadro 10 se pueden observar los tiempos que les tomó a los usuarios de prueba del prototipo programado realizar ciertas acciones. Todas las acciones fueron completadas exitosamente por todos los sujetos de prueba. En este cuadro se pueden observar elementos de interés: primero, el inicio de sesión cuenta con tiempo variados debido a que no se utilizaron datos reales para validarlo. Si bien el diseño del inicio de sesión está pensado en que sea integrado con el inicio de sesión único de la universidad, este no se implementó a nivel de lógica de esta manera para no manejar el registro activo de la universidad y evitar riesgos de seguridad a los datos personales de los estudiantes. En este cuadro también se puede observar que el promedio de tiempo para realizar la mayoría de acciones es inferior a 4 segundos, lo cual se considera una métrica exitosa debido a que los usuarios entrevistados no habían utilizado el prototipo previamente.

Sin embargo, también se puede observar en el Cuadro 10 que el resaltar un texto tomo más tiempo que las demás acciones. Esto fue debido a que se observó que los estudiantes intentaban primero seleccionar el resaltador y luego seleccionar el texto, en lugar de seleccionar el texto y luego resaltar. Sin embargo, se espera que este tiempo anormal sea cuestión de una sola vez, debido a que al momento de comentar un texto (un proceso similar a resaltar), las personas entrevistadas redujeron sus tiempos. A pesar de esto, se implementó resaltado en ambas direcciones en el prototipo final. También es de interés mencionar que la persona externa tomó considerablemente más tiempo en el resaltado de un texto a los estudiantes debido a que esta no se encontraba familiarizada con el *trackpad* de la computadora portátil donde se hicieron las pruebas. Al haber completado la prueba, se le preguntó cuál fue su mayor dificultad al momento de resaltar un texto y se descubrió que esta no fue debido al programa sino debido al dispositivo de prueba. No obstante, sería interesante explorar alternativas de resaltado que no requieran seleccionar texto.

Cuadro 10: Tiempos para completar acciones en la aplicación programada en segundos

Acción	Estudiante 1	Estudiante 2	Estudiante 3	Estudiante 4	Estudiante 5	Persona externa	Promedio
Iniciar sesión	22.3	5	4.2	20.1	13	14.5	13.2
Ir a la biblioteca	4.3	2.5	2.3	2.2	0.8	1.3	2.2
Ir a recientes	2.5	1.2	2.2	0.8	1.0	1.3	1.5
Buscar un libro	1.7	1.2	1.7	1.3	0.4	2.2	1.4
Abrir un libro	2.3	1.1	2.5	1.0	1.5	3.1	1.9
Resaltar un texto	12.2	5.1	18.2	9.4	8.1	29.6	13.8
Comentar un texto	4.8	3.8	2.3	1.0	1.2	3.4	2.8
Mostrar comentarios	3.8	2.7	1.4	4.1	1.6	2.3	2.7
Esconder comentarios	3.9	2.1	1.2	1.5	1.1	2.3	2.0
Buscar un texto en un libro	1.2	1.5	1.2	0.6	1.2	1.4	1.2
Cambiar de página	3.5	1.2	1.0	1.2	2.3	11.0	3.4
Cambiar el estilo	3.1	3.5	3.2	1.5	1.7	3.4	2.7
Regresar al menú	2.1	1.9	2.1	3.1	1.1	5.2	2.6
Cerrar sesión	3.2	2.8	1.6	1.4	1.6	3.8	2.4

7.4. Pantallas finales

Para las pantallas principales: inicio de sesión, menú principal (biblioteca o *storefront*), búsqueda y lector, se elaboraron 4 prototipos por medio de *Figma*. Estos fueron puestos a prueba y mejorados. Utilizando el conocimiento adquirido por medio de estos prototipos se realizaron los prototipos programados. Estos fueron una vez más sometidos a pruebas hasta llegar a los resultados mostrados en esta sección. Las pantallas fueron programadas por medio de *Angular* y los principios de *SOLID*.

7.4.1. Inicio de sesión

El resultado de esta pantalla puede ser observado en la Figura 64

Consideraciones al ser un inicio de sesión para UVG

Cabe mencionar que el diseño de inicio de sesión fue elaborado según los requerimientos de UVG, sin embargo, la lógica de este inicio de sesión no está programada bajo un inicio de sesión único utilizado por UVG debido a que no se manejaron datos reales de la universidad. No obstante, el diseño final de la pantalla representa cómo se vería un inicio de sesión único en la interfaz, lo cual se describe a continuación.

En UVG se maneja un sistema de inicio de sesión único, por lo que no se incluyó un botón de registro. Esto es debido a que los estudiantes tienen una cuenta institucional que les permite iniciar sesión en varios servicios de la universidad, como *Canvas*. Sin embargo, sí se incluyó un enlace de contraseña olvidada que lleva a la herramienta proporcionada por *Microsoft* para manejar cuentas institucionales. En caso de hacer una integración con *Canvas* por medio de su *REST API*, se considera que esta pantalla no será necesaria debido a que el inicio de sesión será manejado directamente por *Canvas*.

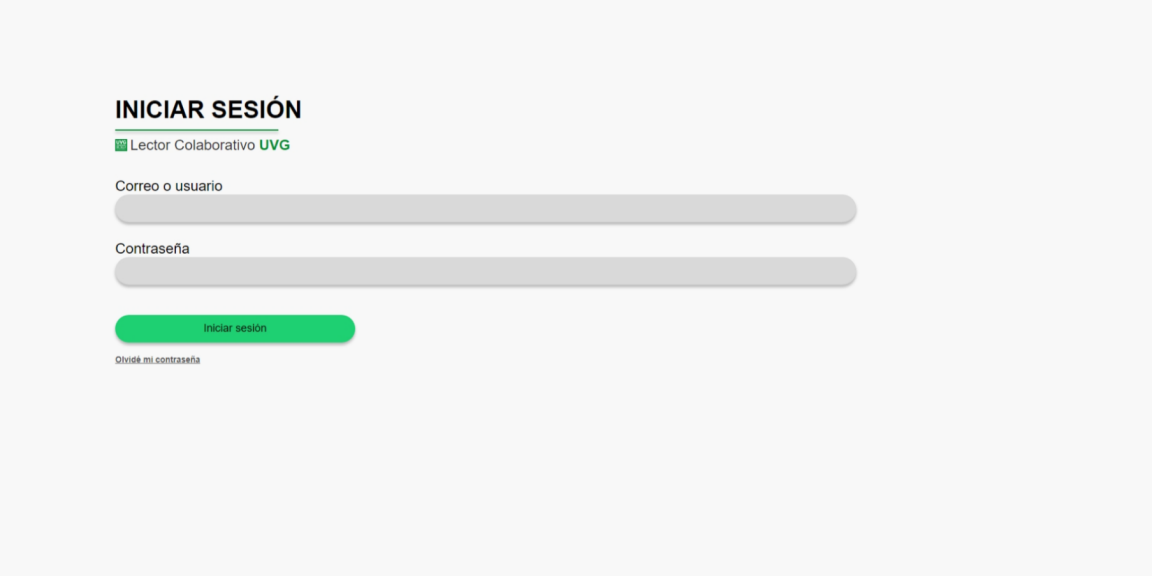


Figura 64: Pantalla final de inicio de sesión

Descripción del diseño

La composición de este inicio de sesión es vertical para dar un sentido de alerta (Bradley, 2014). Esto se hizo con la finalidad de que el usuario pueda estar al pendiente de lo que introduce en los campos y poder minimizar los errores. El botón de iniciar sesión destaca al contar con un color distinto al resto del inicio de sesión. Esto es debido a que la composición se realizó con colores neutros y el botón presenta un verde más encendido, lo cual le da peso visual y facilita que el usuario lo encuentre (Babich, 2021). También se le da más peso visual al título debido a que presenta un subrayado y tiene un tamaño mayor de fuente en negrita. Finalmente, los elementos se alinean ligeramente arriba del punto medio de la pantalla, lo cual hace la composición menos pesada (no parece que se está «hundiendo») y coloca los campos de introducción de datos en uno de los puntos visuales más atractivos para el usuario, ligeramente arriba del centro geométrico de la composición (Bradley, 2014).

7.4.2. Menú principal

El resultado de esta pantalla puede ser observado en la Figura 65.

Influencia de la investigación contextual

Para el diseño del menú se utilizó un diseño basado en metáforas hacia el mundo real, utilizando el conocimiento adquirido por medio de la investigación contextual al visitar librerías y asociándolo con la interacción humano-computador. Para ello, se tomó como base la vista de libros desplegados en una estantería, como se observa en la Figura 66. Los libros, al llenar una fila, crecen de manera vertical, lo que es una metáfora hacia una estantería, como se observa en la Figura 67. Esta fue la opción preferida por el grupo de pruebas. La

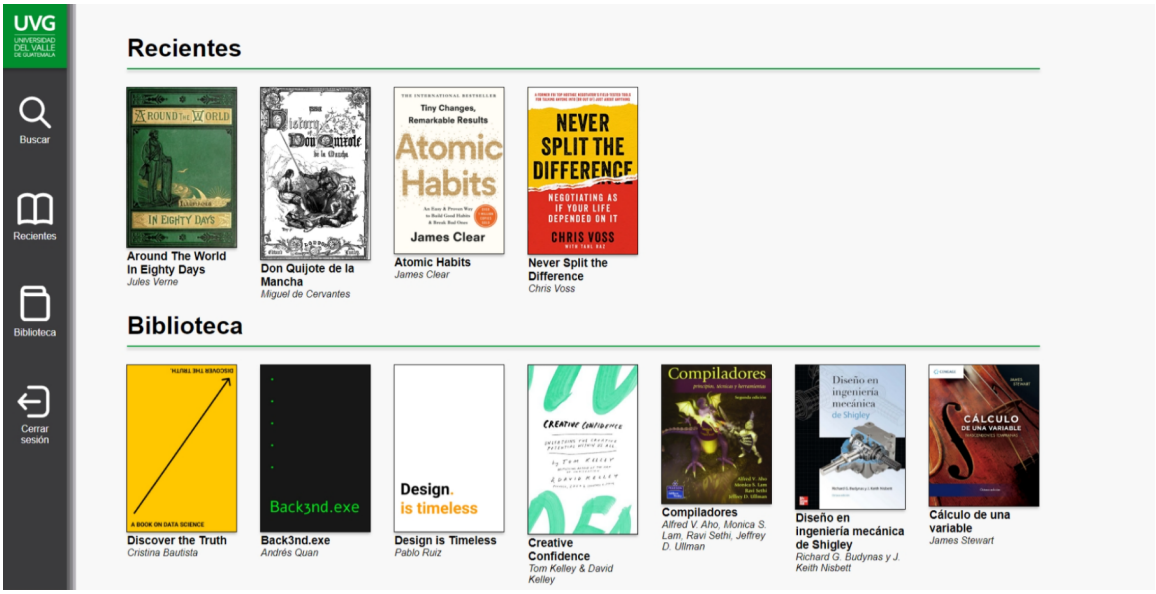


Figura 65: Pantalla de menú

pantalla también se dividió en 2: recientes y biblioteca, como se observa en la Figura 65, siguiendo los modelos de atención y trabajo de la interacción humano-computador, para evitar confusión al usuario final. Esto permite que la percepción de carga percibida sea menor al interactuar con la interfaz al contar con bloques de atención (Foraker Labs, 2015).

(a) Inspiración. (b) Metáfora implementada.

Figura 66: Metáfora de libros

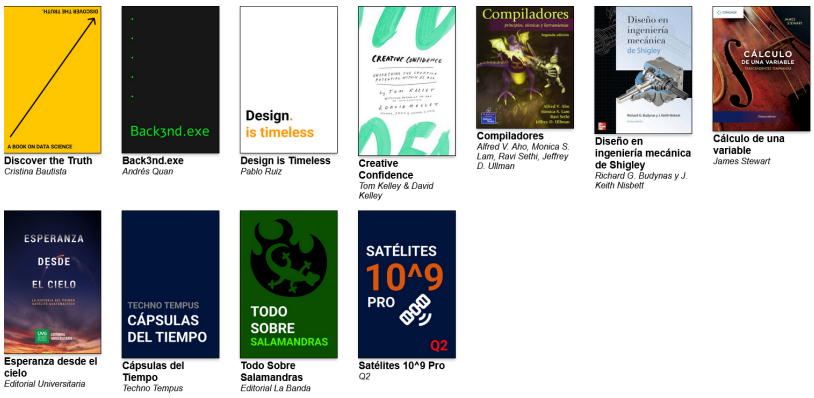


Figura 67: Crecimiento vertical de libros

Diseño de la navegación

La navegación del menú se organizó en términos de utilidad para los usuarios según las necesidades descubiertas a través de las entrevistas iniciales y pruebas de prototipos. Se descubrió que la búsqueda de libros era algo clave para los usuarios, por lo que se colocó esto en la parte superior de la navegación. Posteriormente se colocó la sección de recientes, biblioteca y cerrar sesión. Cerrar sesión se colocó de último para facilitar la concentración de los estudiantes y que no sea la primera opción que vean al momento de navegar. En esta navegación se utilizaron íconos con metáforas que se encuentran vacíos por dentro para distinguirlos de los íconos utilizados en el lector, que tienen un relleno sólido.

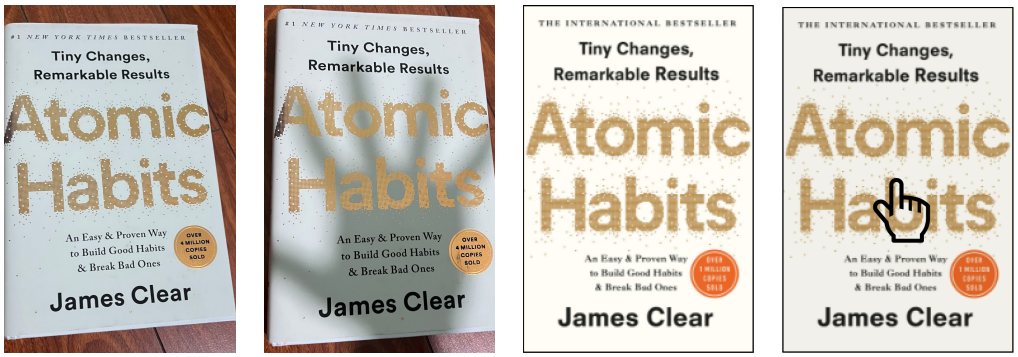
Diseño de la selección de libros

La selección de libros se basa en una metáfora también, como se observa en la Figura 68. Esta metáfora utiliza la sombra para hacer referencia a la selección de un libro en la vida real. Se usó una sombra sutil para evitar sobrecargar al usuario o distraerlo del objetivo principal (seleccionar un libro).

7.4.3. Búsqueda

La pantalla de búsqueda fue diseñada para también contar con bloques claros y reducir la carga visual percibida por el usuario. Al utilizar una arquitectura modular, se aprovechó un diseño similar para la búsqueda de libros (Figura 69) y búsqueda de texto dentro de un libro (Figura 70).

Se utilizó la lupa en el botón de búsqueda según retroalimentación de pruebas de usuario. Además, se decidió utilizar un diseño horizontal para la entrada de texto y el botón, en caso de que un estudiante se encuentre estresado, debido a que este diseño promueve la armonía (Bradley, 2014). Finalmente, se utilizó el principio de localidad para agrupar los resultados



(a) Libro real. (b) Libro real siendo elegido. (c) Libro no seleccionado. (d) Libro con el cursor encima.

Figura 68: Metáfora de selección de libros

en la sección de resultados, lo que hace más intuitiva la interfaz (Kennedy, 2019).

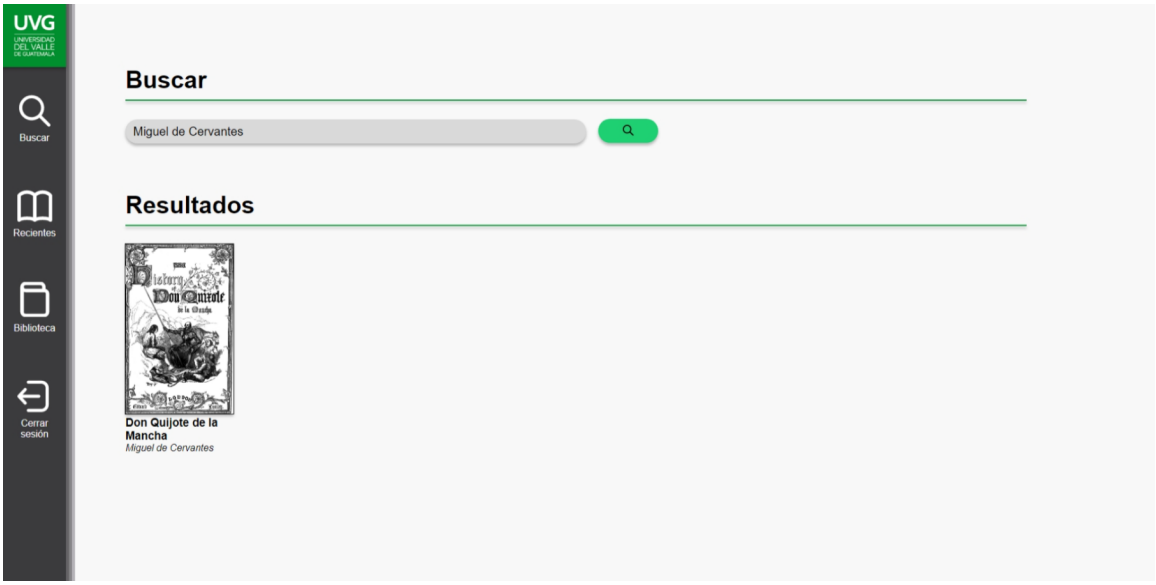


Figura 69: Pantalla de búsqueda de libros

7.4.4. Lector

El lector permite resaltar, comentar, desplegar comentarios, esconder comentarios, buscar pasajes, cambiar el estilo del libro y observar los usuarios activos en un libro. El resultado final puede ser observado en la Figura 71.

Este se diseñó con base en las entrevistas de usuario, la investigación contextual, prácticas de interacción humano-computador y principios de diseño de interfaces. Se utilizó como base el diagrama de la Figura 72, el cual fue elaborado a través de técnicas de observación en la biblioteca y plazas de la universidad, para definir los usuarios activos en la posición superior de la pantalla. Esto es debido a que cuando un estudiante se encuentra estudiando, aunque

Figura 70: Pantalla de búsqueda de pasajes en libros

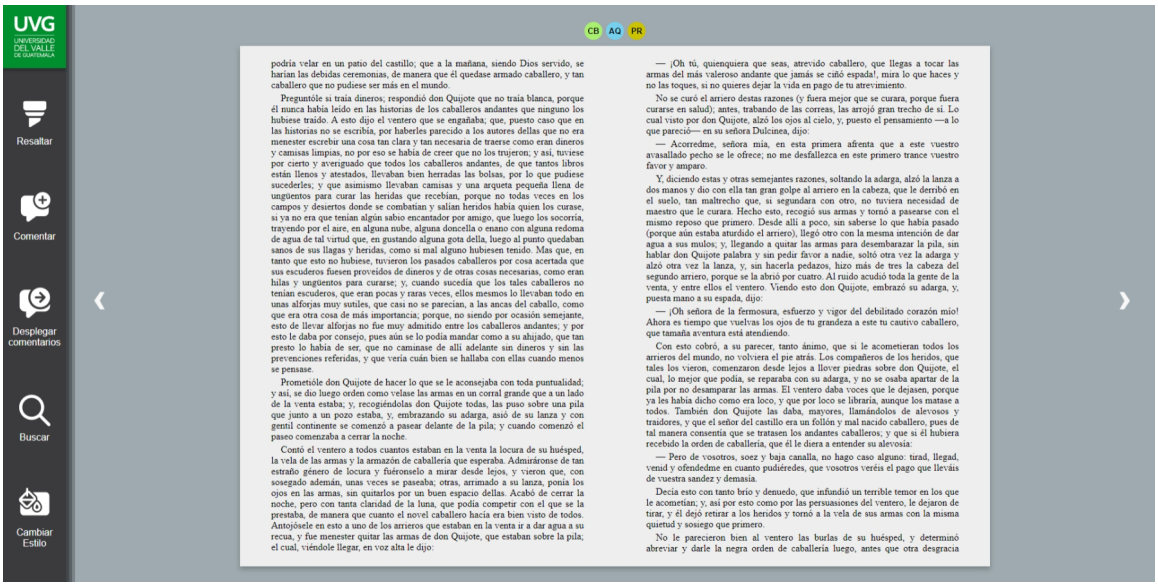


Figura 71: Lector final

este cuente con personas a sus costados, las personas que este más fácil verá son las que se encuentran frente a su rostro. Sin embargo, al estar concentrado en la lectura o algo más no prestará mucha atención a estas personas, por lo que los íconos de las personas son relativamente pequeños. No obstante, al momento de levantar su rostro para ver a una de estas personas (análogo a mover el ratón de la computadora sobre un ícono), se despliega el nombre de la persona, haciendo alusión a que el usuario está concentrado en conocer la identidad de la otra persona. Este proceso puede ser observado en la Figura 73.

También se utilizó como base el diagrama mostrado en la Figura 74. Este muestra la organización del espacio de trabajo de un estudiante y fue construido a partir de las técnicas

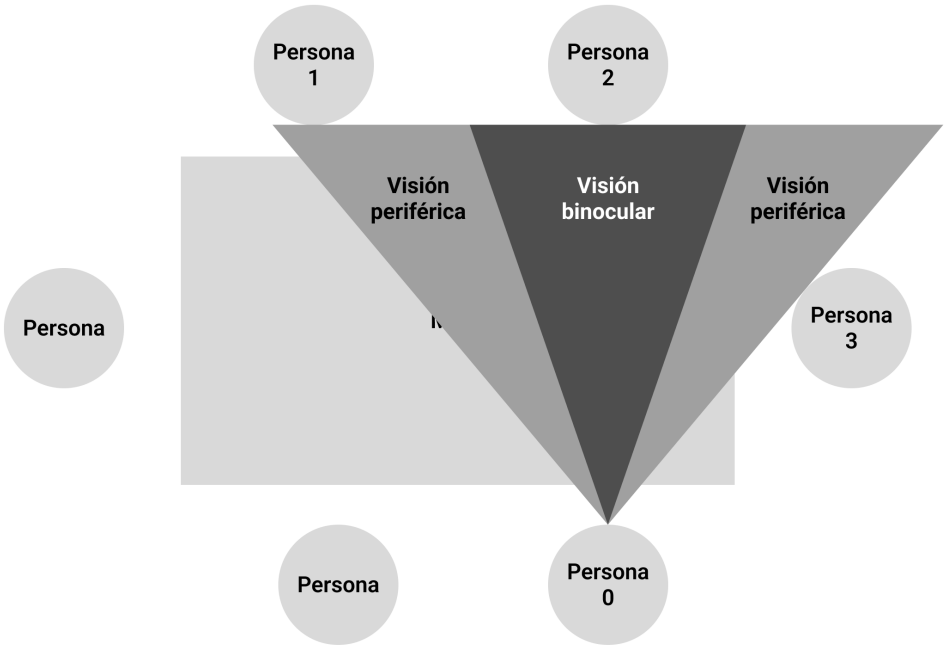


Figura 72: Diagrama de estudio grupal observado

(a) Personas desenfocadas.

(b) Personas enfocadas.

Figura 73: Sección de personas activas en el lector

de investigación contextual. Se puede observar que en el centro de atención del estudiante se encuentran los libros, la computadora o la tableta que utiliza para estudiar. También se observó una combinación de estos elementos en ese espacio, sin embargo, a través de la observación se logró concluir que los elementos de mayor importancia para el estudiante se colocaban en este espacio (la posición central del espacio de trabajo). Por lo tanto, el libro se colocó en la misma posición dentro del diseño del lector. Posteriormente, a los costados se observó que los estudiantes colocaban herramientas. Estas podían ser herramientas como: lápices, marcadores, resaltadores o incluso, un celular. Debido a esto y de acuerdo con los resultados obtenidos a través de las pruebas de usuario y entrevistas, se colocó la barra de herramientas a un costado del lector.

Para los comentarios, se hizo un menú al lado derecho. Esto hace referencia a las notas que se toman en libros físicos a un costado de la página. También se pueden esconder para reducir distracciones, lo cual va en concordancia con los resultados de las entrevistas y pruebas de usuario. Una maqueta general de comentarios puede ser observada en la Figura 75, mientras que una visualización de comentarios y resaltados puede ser observado en la Figura 76. También se puede observar un menú de comentarios vacío en la Figura 77.

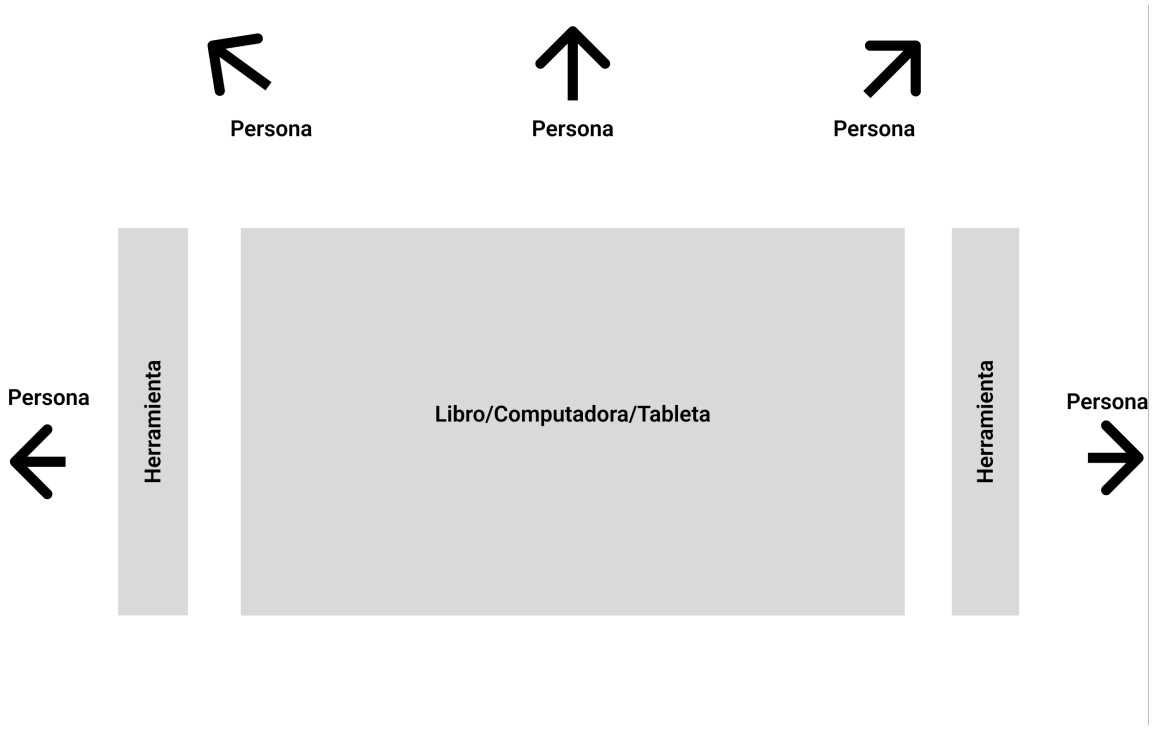


Figura 74: Organización del escritorio de un estudiante

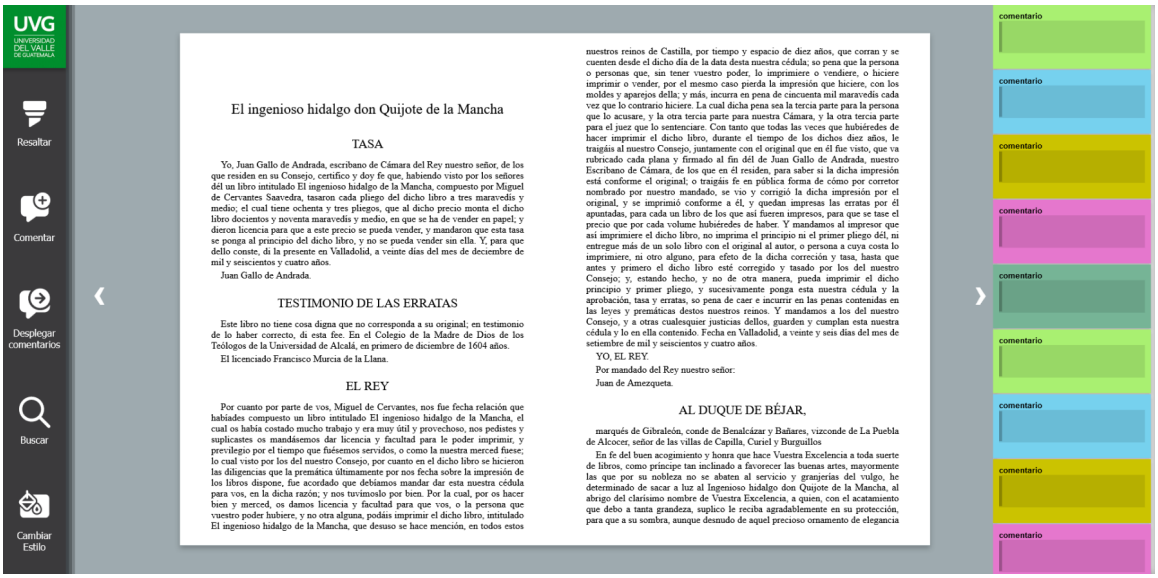


Figura 75: Maqueta general de comentarios

Para publicar un comentario se utilizó la metáfora de pegar una nota en un libro. Esta puede ser observada en la Figura 78. Para ello, se hizo que el fondo del libro se desenfocara al momento de comentar, lo que simula la visión humana al momento de enfocar un objeto más cercano. Sin embargo, como se observa en la Figura 79, no se desenfocan los controles primarios en caso se desee regresar al menú o realizar otra acción, lo cual le da mayor flexibilidad al usuario y libertad, un principio de diseño de interfaces. No obstante,

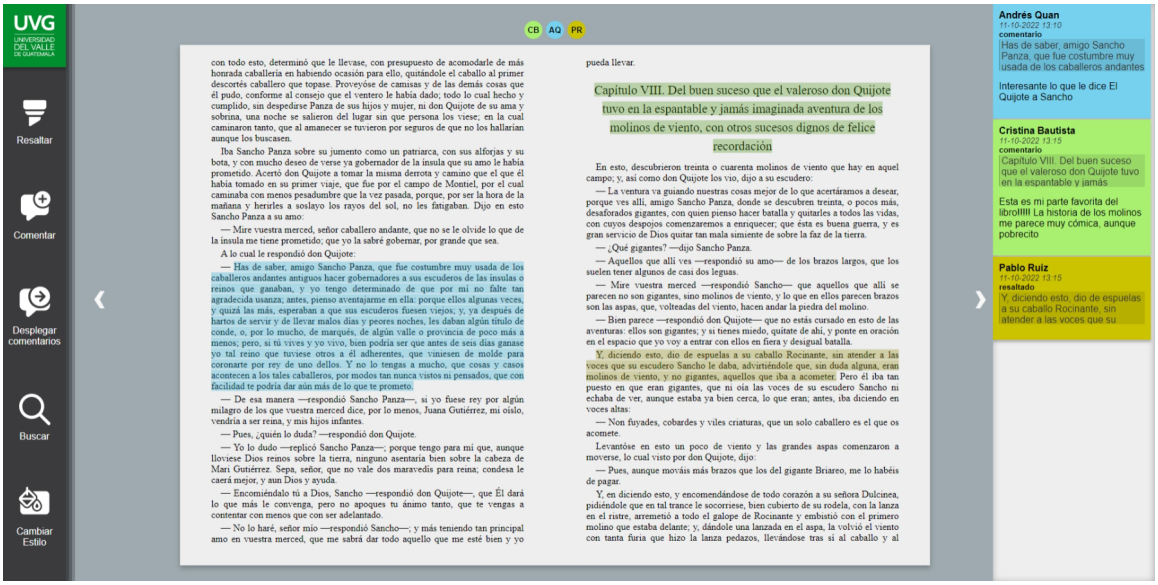


Figura 76: Comentarios y resaltado en un libro

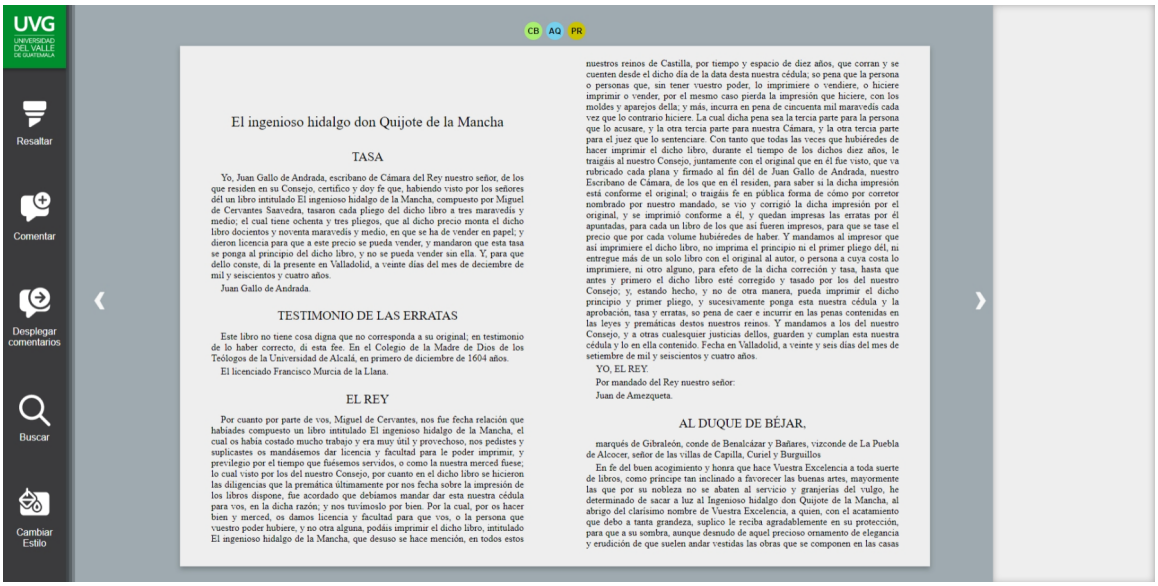
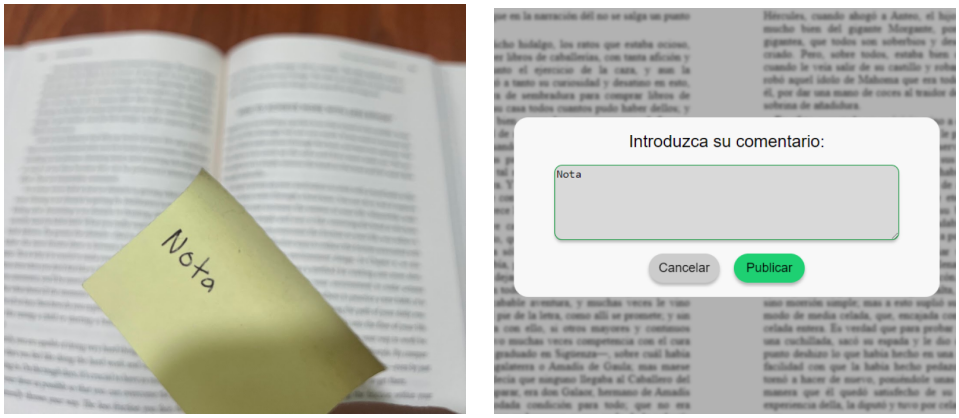


Figura 77: Pantalla sin comentarios

el comentario se colocó en el centro de la pantalla para permitir al usuario concentrarse en la tarea a realizar: publicar un comentario.

Para los elementos del menú de opciones se utilizó la metáfora de levantar una herramienta, como se observa en la Figura 80. Este efecto se logra mediante una sombra proyectada por los elementos de la barra al momento de pasar el cursor sobre ellos. El diseño de los íconos tiene también un relleno sólido para hacer una distinción con los íconos del menú principal.

También, se elaboraron varios temas para el lector: claro, libro, gris, oscuro y de alta accesibilidad (o de concentración). Estos se muestran en las figuras: 81 y 82.



(a) Vista de una nota real. (b) Comentario programado.

Figura 78: Metáfora de comentarios

Figura 79: Comentario en libro con controles disponibles

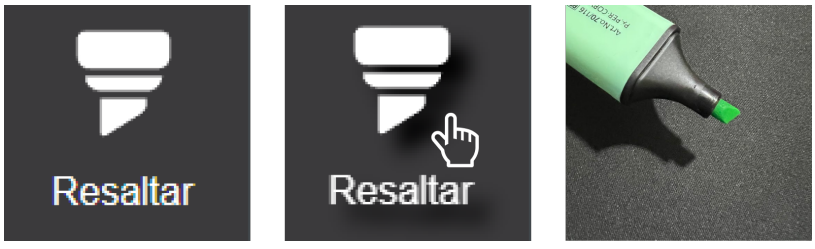
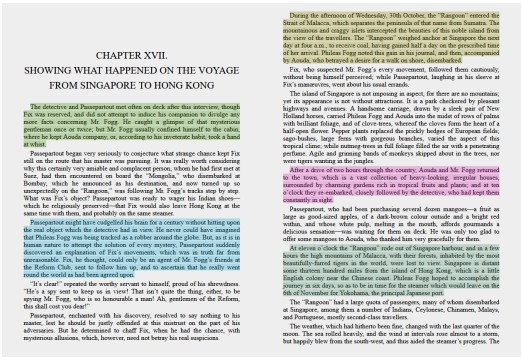
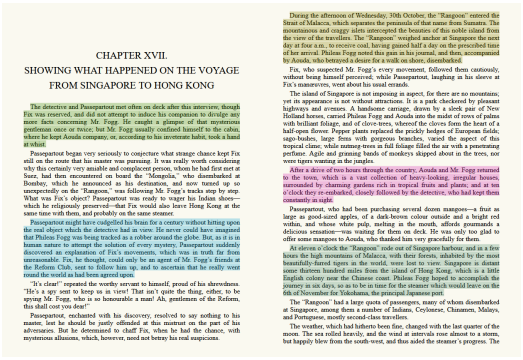


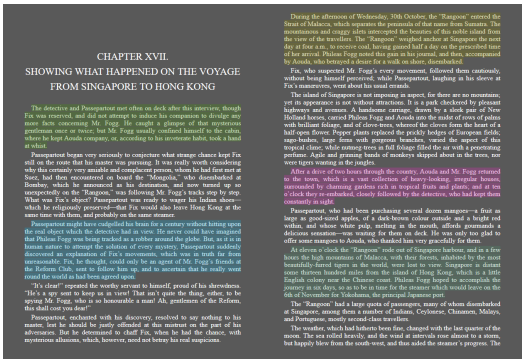
Figura 80: Metáfora de levantar una herramienta



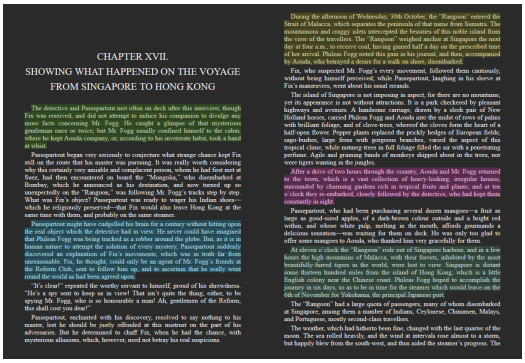
(a) Claro.



(b) Libro.

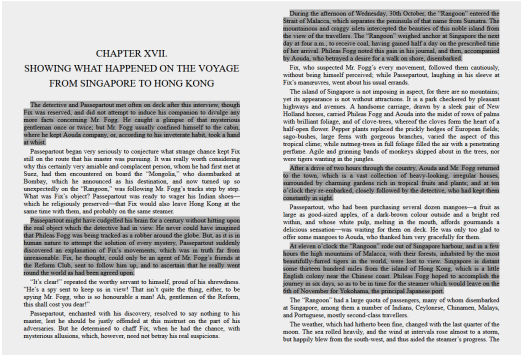


(c) Gris.

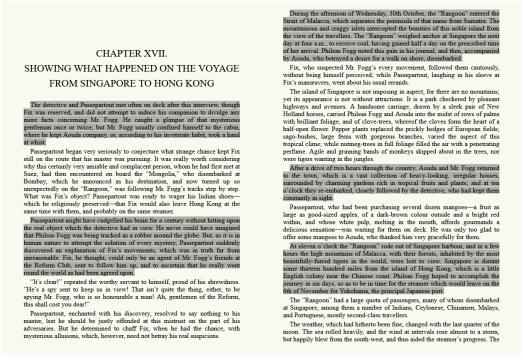


(d) Oscuro.

Figura 81: Temas de libros



(a) Claro (resaltado gris).



(b) Libro (resaltado gris).

Figura 82: Temas de alta accesibilidad en libros

7.4.5. Exámenes

Se elaboró también un prototipo de exámenes de lectura. Estos podrían ser colocados por un catedrático en posiciones estratégicas del libro para evaluar el progreso de los estudiantes y su comprensión. Se elaboraron utilizando el principio de agrupación, para reducir la carga percibida por un estudiante al momento de presentarse una comprobación. Para ello, cada pregunta se encuentra dentro de su propio contenedor con su título respectivo. El resultado de este prototipo puede ser observado en la Figura 83.

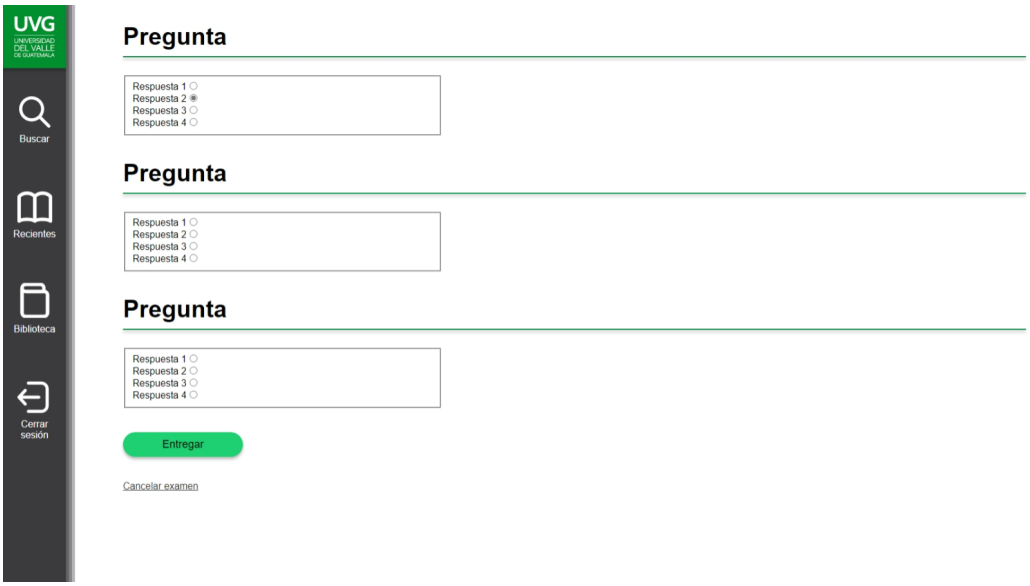


Figura 83: Pantalla de exámenes de lectura

7.5. Prototipos para desarrollo futuro

7.5.1. Propósito

Estos prototipos fueron elaborados para extender el alcance del trabajo presente. Sin embargo, no se encuentran dentro de la definición inicial de alcance, por lo que podrían ser utilizados en futuros proyectos como referencia.

7.5.2. Generación de grupos

Esta pantalla se deja como referencia para un futuro proyecto que pueda extender el alcance de este. La pantalla de generación de grupos podría ser utilizada por catedráticos para generar instancias de libros grupales. Para esta pantalla se realizaron 2 iteraciones: la primera con un prototipo en *Figma* y la segunda la pantalla programada. Para llegar a la pantalla programada se realizó una prueba de prototipo con un catedrático de la Universidad del Valle de Guatemala.

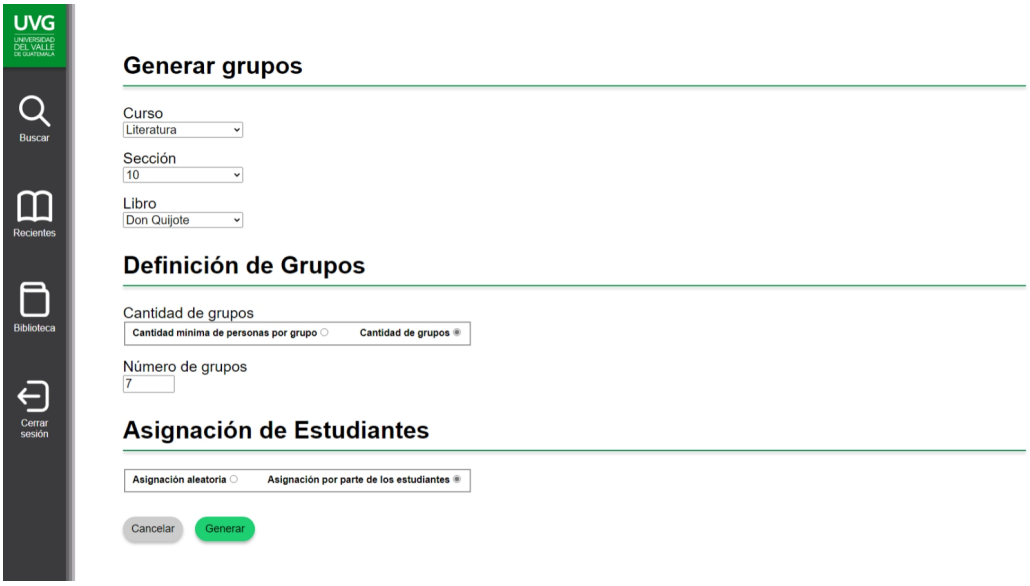


Figura 84: Pantalla de generación de grupos

7.6. Definición de arquitectura

La arquitectura utilizada terminó siendo la que *Nest.js* maneja de manera nativa: *MVC*. Con esto, se terminaron creando varios módulos (controladores) que permiten el fácil acceso y control de todas las peticiones a la aplicación. Aunque no tengan el mismo nombre, la arquitectura puede ser apreciada en la Figura 85.

Esta arquitectura terminó siendo utilizada por su gran escalabilidad y velocidad de programación. Además, gracias a que el *frontend* está programado en *Angular*, las filosofías se complementan y son fáciles de entender, dada la necesidad de hacer cualquier tipo de cambio.

Algo muy importante de entender es que se terminaron haciendo varios cambios en cómo estaba programado el proyecto desde un inicio. La arquitectura final separa a los servicios del controlador casi completamente, permitiendo que toda la lógica estén dentro de solamente el servicio. Esto es por razones de *MVC*. El controlador solo debería de estar manejando las rutas, mientras que el servicio maneja directamente el *business logic* de una petición. Asimismo, el controlador, por ende, no tiene contacto con los modelos, sino que solamente tiene contacto con las guardias y los *DTOs*.

7.7. Definición de población objetivo

Una de las cosas poco usuales para un desarrollador de *backend* es hacer entrevistas a un público objetivo como si fuese de *frontend*. Sin embargo, gracias a la naturaleza del proyecto, se necesitó utilizar la información del *frontend* para poder diseñar las distintas partes del proyecto. En específico, las partes más afectadas fueron las partes que tenían que ver con la seguridad y la velocidad. No se conectaron las partes a un token externo, ya que están fuera

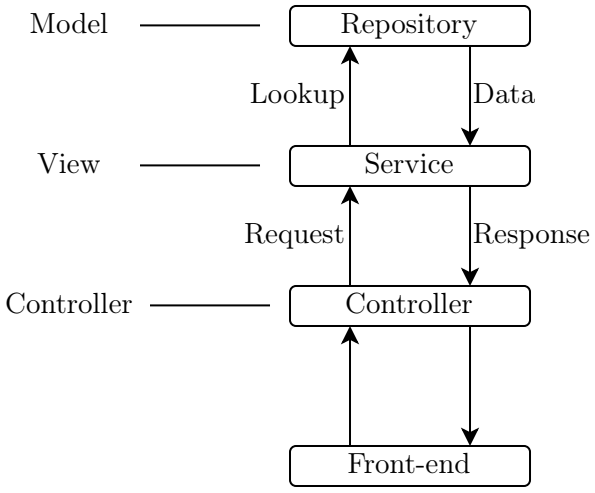


Figura 85: Equivalencia de *MVC* en el Proyecto. El proyecto tiene distintos nombres para sus componentes, porque así es como se llaman en sus propios contextos y librerías. Sin embargo, el *MVC* puede ser visto en cómo está diseñado el proyecto.

Tipo de Tiempo	**User**	**Document**	**Data-Generation**
Linear	01:29.823	02:42.594	07:39.253
Concurrente	00:34.293	00:58.293	01:55.676

Cuadro 11: Comparación de Tiempos con Programación Linear y Concurrente en los Controladores. Los controladores y los servicios más modulares de lo que se había planeado terminaron con los resultados más bajos.

del alcance. Sin embargo, los servicios son mucho más modulares que inicialmente descrito gracias a que estos debían de ser rápidos.

Una gran cantidad de optimizaciones se terminaron haciendo, y se utilizó mucho la programación concurrente para poder servir muchas más peticiones de las que se planeaban originalmente. Es importante recordar que *Javascript* es un lenguaje concurrente, no paralelo. Por esto, la concurrencia debía de ser tratada con mucho más respeto que el paralelismo. La única instancia en la que esto es diferente, es con el uso de los *Sockets*. Los *sockets* existen como un micro servicio por aparte que permiten su utilización en cualquier otro lado.

7.8. Separación de procesos

Existen una gran cantidad de procesos. Tantos, que no sería efectivo describirlos todos. Sin embargo, existen algunos servicios importantes:

- Autenticación
- Autorización
- Configuración
- Creación y manejo de instituciones

- Creación y manejo de documentos
- Creación y manejo de usuarios
- Manejo de jerarquía de usuarios
- Manejo de instancias
- Manejo de personalización
- Manejo de queries extranormales
- Generación de información dummy
- Generación de tablas y sus relaciones por modelos

Cada uno de estos servicios nació del querer hacer que las cosas fueran modulares. Por supuesto, también terminaron siendo inyectables por *Nest.js*, aumentando la velocidad de programación.

7.8.1. Autenticación y autorización

El servicio de autenticación se encarga de revisar que las peticiones estén correctamente autenticadas con un *JWT*. Este es uno de los únicos tres servicios que no son 100 % programados para el proyecto, ya que se basa directamente en la seguridad que proporciona *Nest.js*. Sin embargo, también es importante saber que este proceso está definido a nivel de *root*. Es decir, este servicio está disponible en cualquier lado de la aplicación.

Por el otro lado, el servicio de autorización es virtualmente usado en todos los demás servicios alrededor de la aplicación. Este tiene como principal labor el reforzar la jerarquía de usuarios que se hablo en la metodología. Su labor se basa en hacer que la integridad de la información sea tan sensata como sea posible. Su conexión con todos los módulos del proyecto hace que esté puesto en el módulo de la aplicación, en vez de en cualquier singular. El árbol de dependencias lo mantiene muy arriba.

7.8.2. Configuración

La configuración es el tercer y último servicio definido por *Nest.js*. Es utilizado para poder mantener en un mismo lugar todas las variables de configuración, sean estas de entorno o no. La mayoría de su uso está basado solamente al principio de la aplicación, ya que es este servicio el que permite saber cuáles son las credenciales de varios otros componentes, como la base de datos. Gracias a cómo funciona *Nest.js* y la inyección de dependencias, este servicio tiende a ser implementado mediante procesos asíncronos. Este es el único servicio que se inicializa antes que todos los demás por defecto, como puede ser observado en la Figura 86

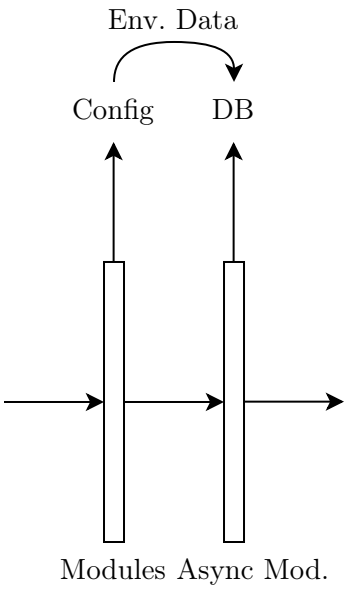


Figura 86: Módulo de Configuración de *Nest.js*. La configuración del proyecto requiere tener ciertas variables de entorno listas en el código. Por eso, las configuraciones de *Nest.js* son realizadas de forma asíncrona.

7.8.3. Creación y entidades

Dentro del proyecto existen una gran cantidad de maneras de manejar instancias y entidades. Aunque la mayoría son basadas en normalización de la base de datos, hay algunos que son bastante importantes por cómo es que se manejan los permisos de la aplicación. Entre estos existen las instituciones, los documentos y los usuarios. A grandes rasgos, estos son tres conjuntos de cómo se manejan las cosas. Se pueden pensar como en localidades de la información que permiten a los módulos de autenticación y autorización trabajar.

Instituciones

Las instituciones hacen referencia a universidades o instituciones educativas que existen. Estas pueden tener un tipo y, lo más importante, una localidad. La localidad de las instituciones permite hacer que estas existan como únicas en un municipio, en vez de un departamento o un país, logrando así hacer que los nombres se repitan.

Una institución tiene la siguiente relación con su territorio:

Institution ← County ← State ← Country

En este caso, la institución sabe del *County* o municipio, pero este es independiente de la institución. Este diseño permite normalizar la base de datos sin mayor problema y, a final de cuentas, fuerza a que un territorio específico tenga solo una institución con ese nombre. **Una institución es única cuando su tipo y su nombre no se repiten en un mismo territorio**.

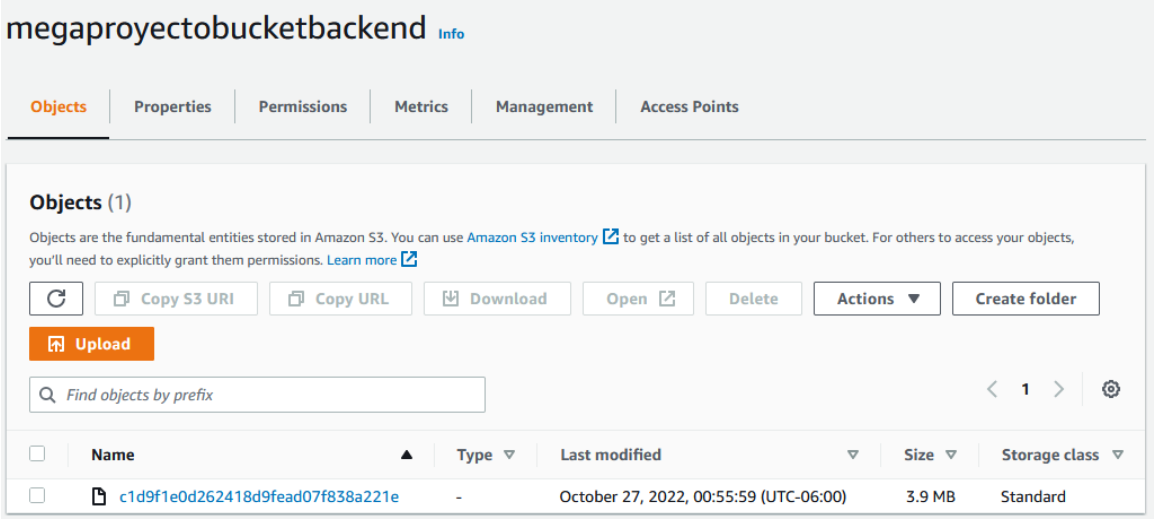


Figura 87: Los archivos guardados dentro del *bucket* de *AWS* son guardados utilizando el *UUID* del documento en la base de datos, ahorrando espacio.

Documentos

Los documentos hacen referencia a archivos en algún sistema de almacenamiento. Originalmente se tenía planeado el hacer que los archivos estuviesen guardados dentro de carpetas que solo fuesen accesibles por medio de identificadores de usuario (por nombre). Sin embargo, esto resultó siendo demasiado complicado para el proyecto. Al final de cuentas, este problema fue resuelto al usar el id del documento en la tabla como su nombre en el sistema de archivos. El efecto de esto puede ser encontrado en la Figura 87

Por supuesto, rápido se encontró que los usuarios preferían utilizar *PDFs* en vez de *EPUBs*. Esto fue resuelto rápidamente con la utilización de librerías que permiten la conversión entre *PDFs* y *EPUBs*. Sin embargo, esto no fue implementado en el *frontend* para evitar la piratería. Esto fue **inspirado en el proceso de conversión de *Amazon* con su *Kindle*.**

Usuarios

Los usuarios son una parte fundamental del sistema. Gracias a que estos terminaron siendo leídos muchas veces, algunas de sus características terminaron siendo extraídas a tablas adyacentes, con la posibilidad de solicitar dichas características. A final de cuentas, esto es porque el *query* estaba terminando demasiado grande. Por seguridad, estos cambios se deben de hacer dentro del código para reflejar características en una base de datos. Las características más importantes que pueden ser obtenidas por medio de *URLEncodedParams* son:

- ***customized***: Devuelve las características personales por usuario, tales como un color seleccionado.
- ***groups***: Devuelve los grupos a los que pertenece un usuario

- ***comments***: Devuelve los comentarios de un usuario. Se puede combinar con ***inDocument*** para poder seleccionar un documento en específico.
- ***inDocument***: Devuelve los datos en un documento. Si no hay ningún otro filtro que lo afecte, devuelve exactamente toda la información de un usuario de un documento.
- ***metadata***: Devuelve toda la información que sea tomada como «*metadata*», tal como la institución, la localidad, la creación, etc. entre otros.
- ***documents***: Devuelve todos los documentos que estén relacionados con el usuario de por sí. Devuelve, no importando el rol.

7.8.4. Jerarquía de usuarios

La jerarquía de usuarios terminó siendo exactamente como fue descrito en la sección de metodología. No en los roles, sino que en el tema del diseño de las tablas. Se pueden hacer referencia a las Figura 16 para entender cómo es que funciona la jerarquía de usuarios; y a la Figura 20 para saber cómo es que funciona el añadir un nuevo usuario al sistema. Sin embargo, sí se hizo un cambio muy leve, pero en el ámbito de los usuarios.

Los usuarios ahora tienen dos tipos de contextos en los que pueden tener permisos o roles. El primero es el contexto de **documento**, el cual maneja qué es lo que pueden hacer los usuarios en un documento; y el contexto de **institución**, que describe qué cambios pueden hacer los usuarios al portal de una institución de por sí.

Por documento

La jerarquía de un documento tiene solo algunos permisos:

- Lectura
- Escritura
- Comentarios
- Cambios de información básica
- Cambios de información de bajo nivel
- Cambios de documento
- Borrar o crear cosas de documento
- Gestión de procesos internos del documento

Todos estos se explican a sí mismos. El último punto hace referencia a que se pueden poner preguntas en el documento de por sí. Asimismo, existen los siguientes roles:

- *Owner*

- *Admin*
- *Mod*
- *Student*

El rol de *Owner* tiene todos los permisos, mientras que el de *student* solo tiene de lectura y comentarios.

Por institución

La jerarquía de una institución es bastante similar a la de los documentos. En específico, la jerarquía de institución tiene todos los roles para hacer cambios a un portal de la misma manera que pasa con un documento. Sin embargo, existen roles más roles:

- *Superuser*
- *Institution Owner*
- *Institution Manager*
- *Institution Admin*
- *Institution Dean*
- *Institution Teacher*
- *Institution Student*

La relación entre estos roles es la misma que de los roles de los documentos.

7.8.5. Instancias

Las instancias de documentos fueron implementadas exitosamente en el lado de *REST*, ya que no tienen ningún equivalente en los *Sockets*. Se hizo que los *Sockets*, al momento de tener algún tipo de evento que tenga que ver con abrir o cerrar el programa, o hacer o borrar un comentario, simplemente consumirán al servicio *REST* en la misma instancia de *EC2* de *Amazon*.

Gracias a que estas son instancias imaginarias (es decir, solo permiten las relaciones entre documentos, grupos, usuarios, comentarios y sesiones), no existe mucha configuración previa. Sin embargo, estas tienen dos funciones muy importantes: el avisar al *REST API* que han habido cambios, y el avisar a todos los demás *sockets* que estos deben de hacer un *pull* de los cambios. Estas funciones pueden ser apreciadas en el Cuadro 12.

Fuente	Evento	Función
userSocket	newComment	Avisa que un nuevo comentario se ha hecho.
allSockets	newComment	Avisa a los demás sockets que deben de hacer un get de los comentarios existentes.

Cuadro 12: Eventos de *Socket.IO* más Básicos. El *Socket* solo existe en el programa para avisar que hubo un cambio en alguna parte. No se encarga de administrar información por temas de seguridad y de superficie.

userId	username	password	institutionId	Metadata
BIGINT	VARCHAR(191)	BINARY(60)	INTEGER	JSON

Cuadro 13: Tabla de *User* Mostrando capacidad *SQL* y *NoSQL*. Gracias a que se necesitaban tener grados de personalización más altos, se terminaron creando columnas de tipo *JSON* en las tablas que interactuaban directamente con el *frontend*. El tipo de dato de *password* es *Binary(60)* gracias a que este debe de guardar los *hash* generados por *bcrypt*, no las contraseñas de por sí. **Esto es por temas de seguridad.**

7.8.6. Personalización

Como fue planeado en la parte de la metodología, solo las tablas que son utilizadas con el *frontend* tienen columnas de tipo *JSON* para poder simular lo que sería un *NoSQL*. Esto permite que cosas de un *frontend* sean altamente personalizables sin tener que cambiar la estructura de la base de datos, teniendo solo las cosas requeridas como estáticas. Un ejemplo de cómo es que se mira una de las tablas (en específico, la tabla de *user*) puede ser visto en el Cuadro 13

Los valores están optimizados para lectura, más que para escritura, gracias a que muchas de las tablas solo requieren que se lean. El escribir a estas solo se da mucho en las tablas de comentarios y sesiones.

7.8.7. Generación de información

Se define como generación de información a la generación de datos que eran necesarios para el módulo de ciencia de datos. Las tablas generadas fueron:

- *Country*
- *State*
- *County*
- *Institution*
- *Role*
- *User*

- *Document*
- *Group*
- *UserGroup*
- *Session*
- *Comment*

Todas estas se generan por las tablas de *session* y de *comment* son las tablas que se utilizan para poder hacer estudios de parte del módulo de *Machine Learning*.

7.8.8. Generación de tablas

No solo las tablas vistas en el módulo de Generación de información existen. Entre las que se terminaron utilizando, están esas, y muchas que se utilizan para mantener un alto grado de normalización en la tabla, ahorrando espacio. Sin embargo, es importante notar que estas no están diseñadas para hacer cambios constantemente, ya que la mayoría de estas están hechas para escritura. Sin embargo, también hay que tomar en cuenta que el *backend* está diseñado para ser más para consultas que algo que requiere muchos cambios repentinos. Muchos de los procesos son realizados en el *frontend*.

Las tablas no solo existen separadas, sino que también existen como vistas. Las vistas permiten que se puedan hacer *pseudo queries* sin la necesidad de ir a tocar la base de datos todo el tiempo. Esto permite optimizar enormemente el *backend*, ya que una de las cosas que más tiempo toma es ir a tomar datos del almacenamiento estático.

Relaciones por modelo

Las tablas generadas son hechas por medio de *TypeORM* en vez de *SQL* directamente. Esto se debe a que es mucho más sencillo el escribir clases (y cómo estas interactúan entre sí), en vez de escribir *joins* en *SQL*. Además, es mucho más escalable.

Las relaciones dentro de este tipo de programación son auto-generadas, con la excepción de las relaciones *ManyToMany*. Estas son generadas por medio de decoradores. El siguiente fragmento de código es un ejemplo de dos tablas relacionadas:

```
@Entity()
@Unique(['username', 'institution'])
export class User {
    @PrimaryGeneratedColumn({ type: 'bigint', unsigned: true })
    id: number;

    @Column({ type: 'varchar', length: 191, nullable: false })
    username: string;

    @Column({ type: 'binary', length: 60, nullable: false })
```

```
    password: string;

    @OneToMany(() => Document, document => document.user)
    documents: Document[];

    @ManyToOne(() => Institution, Institution => Institution.users)
    institution: Institution;

    @ManyToOne(() => Role, role => role.users)
    role: Role;

    @OneToMany(() => UserGroup, UserGroup => UserGroup.associatedUser)
    userGroups: UserGroup[];

    @OneToMany(type => Comment, comment => comment.commentCreator)
    leftComments: Comment[];

    @OneToMany(type => Session, Session => Session.user)
    sessions: Session[];
}
```

Este fragmento de código tiene más columnas que las que fueron enseñadas en el Cuadro 13. Esto se debe a que más de la mitad de las propiedades son solo de relaciones de esta tabla con las demás tablas. Por consiguiente, en el caso de la tabla de *Role*:

```
@Entity()
export class Role extends IdAndName({ primaryColumnType: 'integer' }) {
@OneToMany(() => User, user => user.role)
users: User[];
}
```

La relación entre las dos tablas es entendida por *Nest.js* y guardada para su uso posterior. Estas pueden ser hechas como *lazy loading*. Las columnas de *ID* y *name* no aparecen porque el *extends* devuelve una clase personalizada que tiene estas dos ya. Este método fue escogido por qué tan comunes son estos patrones.

7.9. *Logging*

El *logging* está implementado utilizando *Winston*, una librería de *Node.js* que permite hacer distintos tipos de transportes por tipo de *logger*. Este módulo, por el momento, está puesto de manera tan reemplazable como sea posible, ya que no implementa muchas cosas. Entre los tipos de mensajes que trata de implementar están:

- *Log*
- *Warn*

- *Error*

Pero solamente estos son implementados. No existe configuración especial más que la forma del mensaje, que incluye los tiempos, el servicio, el módulo, y el error generado.

7.10. Separación de módulos

Los módulos fueron separados a partir de los principios *SOLID* para mejor trabajar con la modularidad de servicios. Muchos ya fueron mencionados en la fase de metodología, teniendo varios muy importantes. Los módulos están diseñados para que la comunicación que estos disfrutan sea por medio de servicios que pueden ser inyectados a cada uno. No se utilizan *singletons* para evitar el atraso de respuestas. Asimismo, cada uno está diseñado para que pueda ser utilizado en cualquier controlador.

7.10.1. *Auth*

El módulo de *Auth* se encarga de admitir todas las peticiones para hacer *login* o registrarse en el sistema. Gracias a decisiones por parte del *frontend*, ya no existe un módulo de registro válido. En vez, se utilizarán usuarios proporcionados por las distintas universidades.

El módulo de *Auth* básicamente solo busca generar *JWTs* y guardar datos de sesiones de los usuarios que acceden al sistema. No busca nada más como parte de las metodologías *SOLID*.

7.10.2. *Document*

El módulo de *document* se encarga solamente de gestionar la información de los distintos documentos dentro del sistema. Asimismo, se encarga de asegurarse que todos los documentos tengan el tipo correcto de dato, este siendo de tipo *EPUB*. Su funcionamiento hace que todos los demás módulos puedan asumir que lo que está dentro de esta tabla sea siempre de un formato y un tamaño máximo (que no ha sido definido). Existen varios *checks* dentro de este módulo, los cuales incluyen temas de *autorización* en vez de autenticación.

7.10.3. *DocumentInstance*

El módulo de *DocumentInstance* interactúa con el módulo de *Socket*, ya que es este el que hace que se abran los distintos grupos dentro de un *socket*. Sin este módulo, todo tendría que ser hecho por el módulo de *Socket*, rompiendo las reglas definidas por *SOLID*. Sin embargo, este módulo también maneja sesiones y comentarios como parte de sus relaciones.

7.10.4. *Socket*

El módulo de *socket* es el módulo más sencillo. Consume a *DocumentInstance* para agregar o quitar comentarios, o crear nuevas sesiones en la base de datos. Avisa a este cuando se deba de cerrar una sesión de *socket* (para reducir costos) y permite que el servicio de *sockets* externo se enfoque en el servicio de *REST*. Se puede pensar como una capa de abstracción.

7.10.5. *Group*

El módulo de *Group* es generalmente modificado por roles distintos a *student*. Este módulo permite la fácil adquisición de datos de un grupo, la creación y supresión de los mismos, y la creación de relaciones con los mismos. Es importante también mencionar que estos grupos son relativamente versátiles y solo son representaciones en memoria dinámica de relaciones en la base de datos. Este módulo es utilizado por el *DocumentInstance* para poder crear los *sockets*.

7.10.6. Controladores

Por definición, los controladores cambiaron drásticamente desde la fase de diseño. Antes, los controladores estaban diseñados para manejar ciertas partes de la lógica también, pero se terminaron cambiando completamente para seguir las reglas de *MVC*. Los controladores del programa **solo manejan solicitudes**. Su rol también incluye el comunicarse directamente con *interceptors* y son la entrada a las varias rutas. Son gracias a estos que las políticas de los *DTOs* pueden ser reforzadas.

7.10.7. Servicios

Los servicios, como fue mencionado anteriormente en la Figura 85, tienen que ver con las *views* en una arquitectura *MVC*. Por esto, los servicios de la aplicación están diseñados para implementar toda la lógica del negocio dentro de la aplicación. Gracias a cómo funcionan los servicios dentro de *Nest.js*, todos los servicios son inyectables y pueden ser utilizados en cualquier lugar que los requiera. Solo necesitan ser requeridos dentro del módulo y todos los que estén dentro pueden utilizarlos. Hay servicios globales, como sucede con *JWT* y *Passport*; y servicios locales, como sucede con los servicios para la creación de repositorios.

Los servicios son los únicos que interactúan con los repositorios y, por ende, con la base de datos. Esto disminuye el área de ataque y aumenta la seguridad. Por cómo funcionan, los servicios no tienen autenticación, sino solo autorización. La autenticación es manejada a nivel de controlador.

7.10.8. Modelos y repositorios

Los modelos y repositorios son partes importantes de la aplicación, pero son mayormente manejados por *TypeORM*. Los repositorios no son modificados o extendidos, a menos que hayan patrones muy prominentes. Un ejemplo de esto es en el repositorio de *user*, en donde existen los métodos de ***lookupByUsername*** y ***lookupById***. Claro, más lógica puede ser manejada aquí, pero la modularidad es mucho mejor vista también aquí.

Más que nada, los repositorios fueron diseñados bajo la filosofía de «has menos, logra más». Por ende, los repositorios terminan, frecuentemente, siendo abstracciones encima de las funciones de ***getRepository(«TYPE»)***.

7.11. Respuestas

Las respuestas se dejaron de la misma manera que se habían hablado en la metodología. El ámbito de estas se sigue basando en todo lo que es global, porque es necesario tener respuestas normalizadas para todo. Todas las respuestas deben de salir por uno de dos: los *interceptors* o los *Exception Handlers*. Solo existen uno de cada uno, ya que no se requiere mucho de cada uno. Asimismo, siempre retornan respuestas como la siguiente:

```
// POST Login / Register
{
    "success": true,
    "statusCode": 200,
    "payload": {
        "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
            eyJ1c2VybmFtZSI6InF1YTE3NjUyQHV2Zy5lZHUuZ3QiLCJpY
            XQiOjE2NjcyNDc4NDQsImV4cCI6MTY2NzI1MTQ0NH0.6liWoh
            pJEWAJg—MMUbpEqPHmB4g1AqE75wQpPGotrg"
    }
}
```

Las respuestas tratan de guardar el objeto de *payload* en vez de hacer que el valor del token sea el del token para ayudar con la escalabilidad y disminuir el costo de cambio. Las siguientes propiedades se encuentran en todas las respuestas:

- *success*
- *statusCode*

Estas existen solo para seguir normativas impuestas en otros *frameworks*, como lo es *ASP.NET*. Esto permite que las respuestas sean más fáciles de leer, y no añaden mucha más peso a la respuesta.

7.11.1. *Interceptors*

Los *interceptors* terminaron siendo de dos tipos: uno que va antes, y uno que va después. La lógica es tan fácil como su explicación. Los guardias se aseguran que todas las peticiones sean válidas. Estos incluyen a los *DTOs* y a la librería de *class-validation* de *Node.js*. Sin la validación, todo el programa entraría en pánico, gracias a que se tendría que crear otro servicio universal para poder manejar las distintas peticiones. El programa utiliza solo un tipo de guardia por clase, siendo este el de los *JWT*. Las clases tienen el decorador de *@UseGuard* para protegerse. Un ejemplo de una clase que utiliza una guardia dentro del proyecto:

```
import { AuthGuard } from '@nestjs/passport';
import {   UseGuards } from '@nestjs/common';
@UseGuards(AuthGuard('jwt'))
export class DocumentController {
        // ...
}
```

Por el otro lado, también existen los interceptores de salida, los cuales sí han sido mencionados muchas más veces. Estos solo funcionan para normalizar las respuestas por medio de un *map*. Dentro de la clase del *Interceptor*, existe la función *intercept*, cuya lógica es:

```
return next.handle().pipe(map((value: any) => {
    return {
        success: true,
        statusCode: 200,
        payload: value
    };
}));
```

Como es de esperarse, estos dos tipos de *handlers* solo toman en cuenta aquellas respuestas que no sean errores explícitamente. Por ende, el código de respuesta termina siendo un *HTTP200*.

7.11.2. *Exception handlers*

Los *exception handlers* dentro del proyecto buscan normalizar el manejo de los errores de una manera sensata y con tanta información como sea necesaria, pero tan poca como sea posible. Si se trata de registrar un usuario dos veces, por ejemplo, la respuesta es:

```
{
    "success": false,
    "statusCode": 409,
    "error": "Conflict"
}
```

Este tipo de respuesta nos dice que el servidor sigue vivo, pero que hubo un error. El tipo de error puede ser encontrado en el cuerpo o en la respuesta. En este caso, es 409 porque el

usuario ya existía anteriormente. El error da a entender esto, ya que es un conflicto, pero no informa de más, por cuestiones de seguridad. La programación de los *Exception Handlers* es casi igual a la del *Interceptor* de la sección 7.11.1. Sin embargo, este es el complemento del interceptor. Este termina tomando solamente aquellas respuestas que hayan producido un error. Por esto, estos deben de saber manejar el error (si es necesario) y dependen del módulo de por sí. El error es usualmente un *string*, pero puede ser también cualquier otro tipo de *payload*. Está diseñado para ser extensible e intercambiable.

7.12. Manejo de *sockets*

Los *sockets* son lo más sorpresivo de este proyecto. No son difíciles de entender, y mucho menos de llegar a implementar. Se manejan por medio de otro servicio que puede ser replicado (y debe de ser registrado). Este micro servicio es extremadamente escalable, ya que no está directamente conectado con el servicio principal de *REST*. En vez, está implementado como si fuese otra *API* completamente distinta, cuyo único propósito es avisarle a otros usuarios que existen cambios para poder hacer *pull* a las cosas. Este servicio no está del todo optimizado, ya que puede llegar a utilizar solo los comentarios más nuevos, pero se decidió que esta optimización podía hacerse más adelante en la vida del proyecto.

Los *sockets* son extremadamente fáciles de cambiar y de consumir, gracias a que son fragmentos de programa extremadamente pequeños y ágiles. Estos pueden ser puestos con *Docker* sin problema y, gracias a que se registran solos, estos pueden utilizar la *API* cuando esta se necesita de ampliar horizontalmente.

Los *sockets* implementados son muy permisivos con su forma, ya que están implementados con *Socket.IO* y *Express.js*. Su bajo costo de implementación se debe también a su escalabilidad horizontal. Cualquier cambio en el código de un *socket* puede hacer cambios a todos los demás *sockets* sin problemas.

7.13. *Stemmer*

En el marco teórico se describió este paso en Preprocesamiento. Este es un paso importante, para tener una mayor cantidad de palabras únicas desde una palabra raíz. No obstante, una vez aplicado a un conjunto de datos en español, se mostró que afectaba a muchas palabras de una manera no positiva, en algunos casos sí mostraba las palabras raíces, pero en otros eliminaba letras importantes de la palabra. En otros casos no cambio palabras que sí se debían convertir en palabras raíces. Debido a esto, *Stemmer* no se agregó al conjunto de datos de este proyecto.

7.14. Clasificación de textos

Uno de los primeros factores para un buen entrenamiento de un modelo es que la data esté lo más limpia y clasificada correctamente. En el primer intento se descubrió que separar

text_stemmer	text_string
gusten cancun viaja disfruta manera igual	gusten cancun viaja disfruta manera igual
sabiai trata bien santiago chilet cambia asien...	sabiais trata bien santiago chilete cambia asi...
nunca nunca nunca pidai cafe ryanair bueno ven...	nunca nunca nunca pidais cafe ryanair bueno ve...
exito	exito

Figura 88: Imagen de la clasificación del primer intento

cada una de las palabras de los textos y conservar la clasificación de estos, puede no ser el mejor camino. Esto se pudo evidenciar al comparar alguna de las palabras entre el primer y segundo o tercer intento. Se pudo evidenciar un resultado positivo para la técnica de *Mechanical Turks*.

La razón de ser una buena opción la técnica de *Mechanical Turks*, es que hay un humano detrás que entiende lo que significan las palabras sin tomar en cuenta la semántica de una oración y solo tomando en cuenta el sentimiento que promueve una palabra. Esto se pudo comprobar al utilizar esta técnica al analizar palabras.

En las siguientes imágenes (iniciando desde la Figura 89) se puede visualizar cómo se mantuvo la salida del *explode*. Se pudo evidenciar también que este no era un camino viable para entrenar la data con un conjunto de datos en ese estado, donde una persona podría distinguir y comentar que la palabra «gusten» o «disfrutas» que se tomaron como neutras, cuando en realidad son palabras positivas.

	text	airline_sentiment_score
0	gusten	0
1	cancun	0
2	viaja	0
3	disfruta	0
4	manera	0

Figura 89: Imagen de la clasificación del primer intento

7.14.1. *Wordclouds*

Los *Wordclouds*, de los sentimientos, se vieron grandemente afectados entre el primer intento y el segundo o tercer intento por la técnica *Mechanical Turk*.

	text	airline_sentiment_new_score
0	gusten	1
1	cancun	0
2	viaja	0
3	disfruta	1
4	manera	0

Figura 90: Imagen de la clasificación del primer intento

Figura 91: *Wordcloud* del primer intento

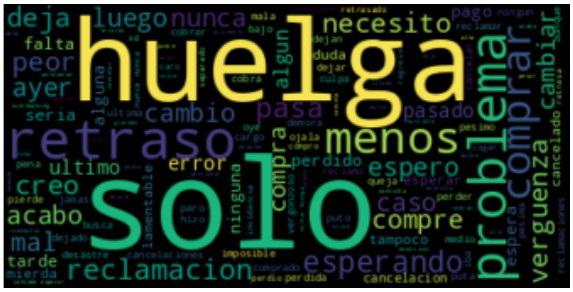


Figura 92: *Wordcloud* del segundo intento

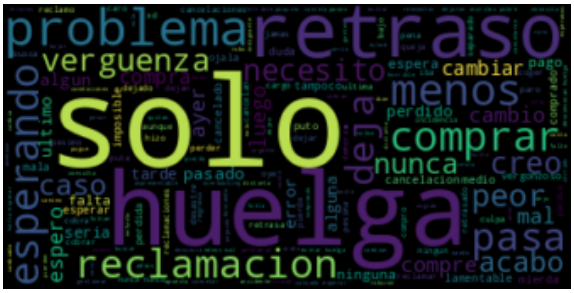


Figura 93: *Wordcloud* del tercer intento

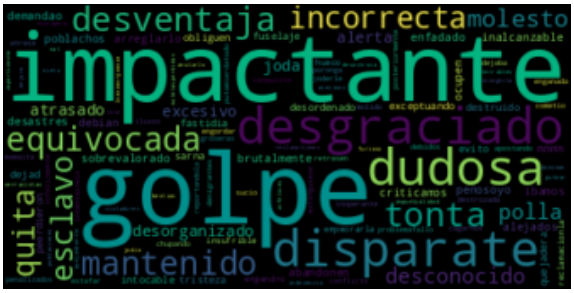


Figura 94: *Wordcloud* del tercer intento sin duplicados

7.15. Balanceo de datos

Fue importante crear la misma cantidad de instancias entre las clases y escoger el mejor método para el balanceo de los datos. En el primer intento se puede observar cómo la clase mayorista es de palabras negativas. Pero, una vez clasificadas las palabras de diferente manera, se puede ver como la clase mayorista debería ser la clase de palabras neutras.

```
-1    3695
 0    2214
 1    1044
Name: airline_sentiment_score, dtype: int64
```

Figura 95: Distribución de las instancias de cada clase en el primer intento

```
 0    31735
 1    11796
-1     6229
Name: airline_sentiment_new_score, dtype: int64
```

Figura 96: Distribución de las instancias de cada clase en el segundo intento

```
 0    5441
-1     784
 1     728
Name: airline_sentiment_new_score, dtype: int64
```

Figura 97: Distribución de las instancias de cada clase en el tercer intento

7.16. Modelo de clasificación

Se crearon 7 modelos de clasificación, 2 de los 7 modelos provienen de la misma función pero con diferentes parámetros

- *BernoulliNB*: Se creó un modelo de *BernoulliNB* sin personalizarlo. Se sabe que este modelo es para clasificación de clases binarias, funciona como modelo control de las métricas
- *MultinomialNB*: Se creó un modelo de *MultinomialNB* sin personalizarlo. Se sabe que este modelo es para clasificación de clases binarias, funciona como modelo control de las métricas
- *LogisticRegression*: Se creó un modelo de *LogisticRegression* personalizado con el parámetro *solver* igual a *lbfgs* para el manejo y pérdida *multiclase*, con el parámetro *multiclass* igual a *multinomial* (indica que es para un modelo *multiclase*)

- *RandomForestClassifier*: Se crearon dos modelos *RandomForestClassifier*, la diferencia entre ellos es el parámetro *max depth*, el primero de 20 y el segundo de 50

- *Support Vector Machines*: Son los últimos dos modelos, el primero con el parámetro del kernel igual a linea (ideal para clasificación de textos), mientras que el segundo el parámetro del kernel es igual a *rbf* (ideal para conjunto de datos no lineales).

7.17. *Classification reports*

El resultado del primer intento, apoyó la hipótesis de que algo estaba mal en el conjunto de datos. Esto ocasionó que se examinara realmente como se estaban clasificando las palabras. El primer intento dio la pauta para agregar una serie de pasos adicionales, añadiéndole a la técnica de *Mechanical Turks*, la creación de diferentes archivos. Tampoco se dejó en un solo archivo, ya que cada que se quisiera correr todo flujo, se borraría instantáneamente.

nombre_modelo	accuracy	precision	recall	f1-score
BernoulliNB	0.483764	0.797748	0.483764	0.408374
MultinomialNB	0.467829	0.463959	0.467829	0.378592
LogisticRegression	0.483764	0.797748	0.483764	0.408374
Random Forest d_20	0.410704	0.787423	0.410704	0.310561
Random Forest d_50	0.415815	0.788092	0.415815	0.318374
SVM Kernel linear	0.483764	0.797748	0.483764	0.408374
SVM Kernel rbf	0.483764	0.797748	0.483764	0.408374

Figura 98: Distribución de las instancias de cada clase en el primer intento

En el resultado del segundo intento, se puede explorar y demostrar el hecho, que es importante aplicar el paso de limpieza de duplicados, para que no suceda demasiado *overfitting*. Esto es debido a que se pudo observar con los modelos binarios que tuvieron un rendimiento muy bueno, cuando la realidad no debía ser esa.

En el resultado del tercer intento, se pudo explorar que los resultados demostraron un mejor comportamiento en cuanto a los modelos de *BernoulliNB* y *MultinomialNB*, debido a que llegaron arriba de un 0.66. Dado que ambos son modelos para clasificaciones binario, esto indica que 2 de 3 clases fueron identificadas. Se pudo observar un poco de *overfitting*, lo cual se pudo dar en el momento que se utilizó la técnica de balanceo de *Random Oversampling*, porque balancea insertando instancias que ya existían en el conjunto de datos.

nombre_modelo	accuracy	precision	recall	f1-score
BernoulliNB	0.906449	0.927094	0.906449	0.908736
MultinomialNB	0.906449	0.927094	0.906449	0.908736
LogisticRegression	0.906449	0.927094	0.906449	0.908736
Random Forest d_20	0.628142	0.825056	0.628142	0.623612
Random Forest d_50	0.640781	0.827905	0.640781	0.638137
SVM Kernel linear	0.906449	0.927094	0.906449	0.908736
SVM Kernel rbf	0.906449	0.927094	0.906449	0.908736

Figura 99: Distribución de las instancias de cada clase en el segundo intento

nombre_modelo	accuracy	precision	recall	f1-score
BernoulliNB	0.663672	0.498587	0.663672	0.553011
LogisticRegression	0.997958	0.997970	0.997958	0.997959
MultinomialNB	0.663672	0.498587	0.663672	0.553011
Random Forest d_20	0.579130	0.813503	0.579130	0.562254
Random Forest d_50	0.580355	0.813744	0.580355	0.564133
SVM Kernel linear	0.997958	0.997970	0.997958	0.997959
SVM Kernel rbf	0.663672	0.498587	0.663672	0.553011

Figura 100: Distribución de las instancias de cada clase en el tercer intento

7.18. Gráficas de métricas de *classification report*

En el primer intento se puede observar que la única gráfica que mostró datos fue la de precisión, esto se dio por ser la métrica que identificaba correctamente a las predicciones positivas con la clasificación positiva.

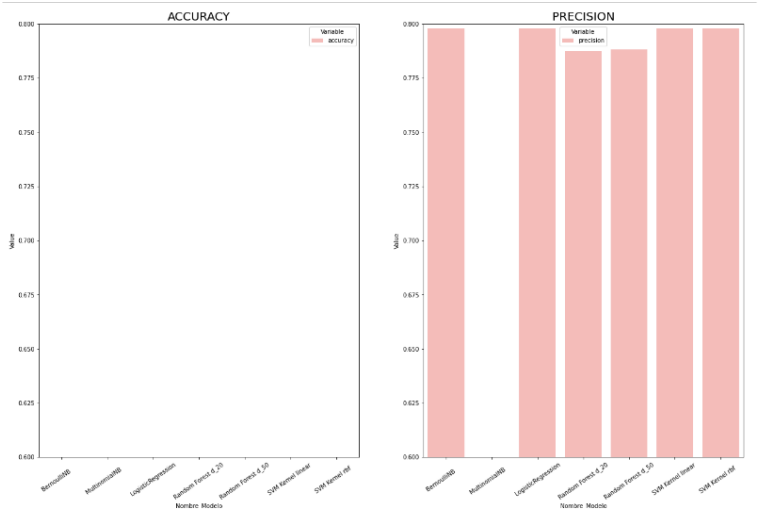


Figura 101: Gráficas de métricas de accuracy y precision del primer intento

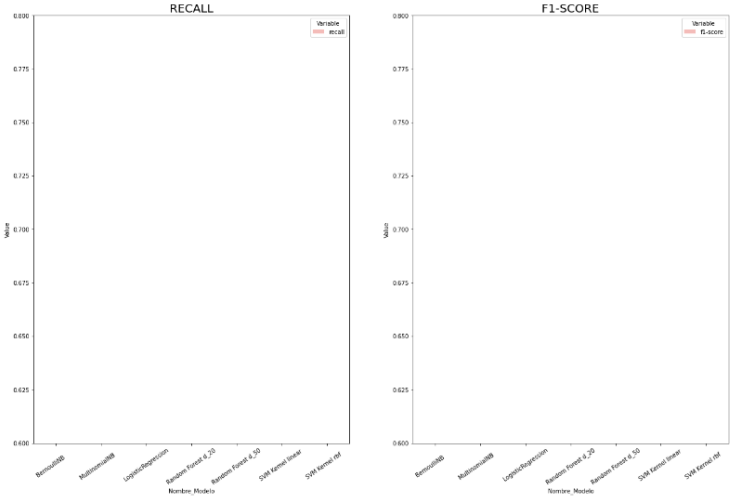


Figura 102: Gráficas de métricas de recall y f1-score del primer intento

En el segundo intento se puede ver como las gráficas esta vez sí muestran datos, se identificó que 5 de los 7 modelos entrenados comparten el mismo porcentaje en todas las métricas, las excepciones fueron los modelos de *Random Forest*.

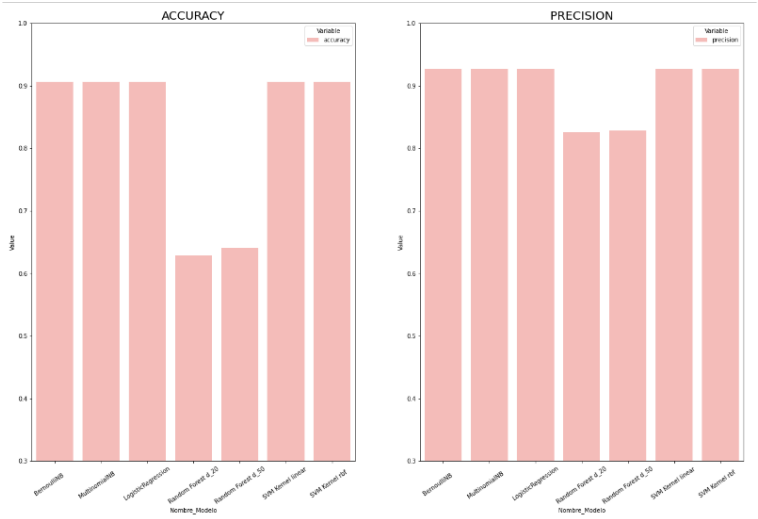


Figura 103: Gráficas de métricas de accuracy y precision del segundo intento

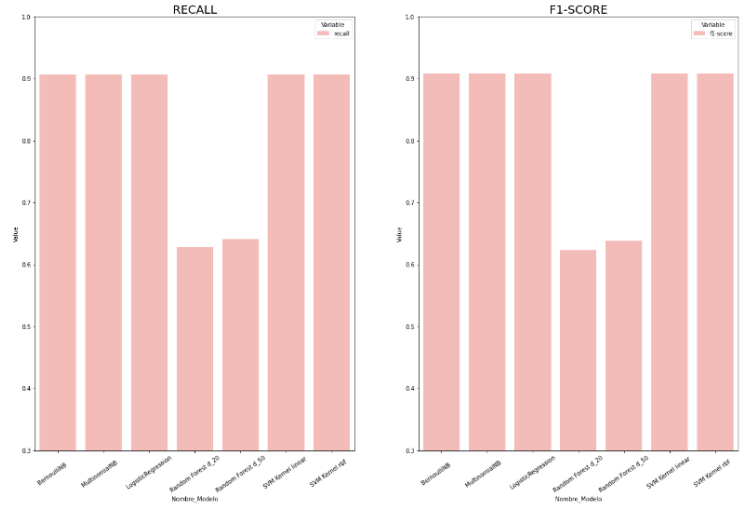


Figura 104: Gráficas de métricas de recall y f1-score del segundo intento

En el último intento se puede ver como los modelos que de *LogisticRegression* y *SVM Kernel linear*, son los modelos que tuvieron: mejor rendimiento, mejor porcentaje de clasificaciones correctas, mejor porcentaje de números de elementos que son de la clase positiva y se clasificaron correctamente.

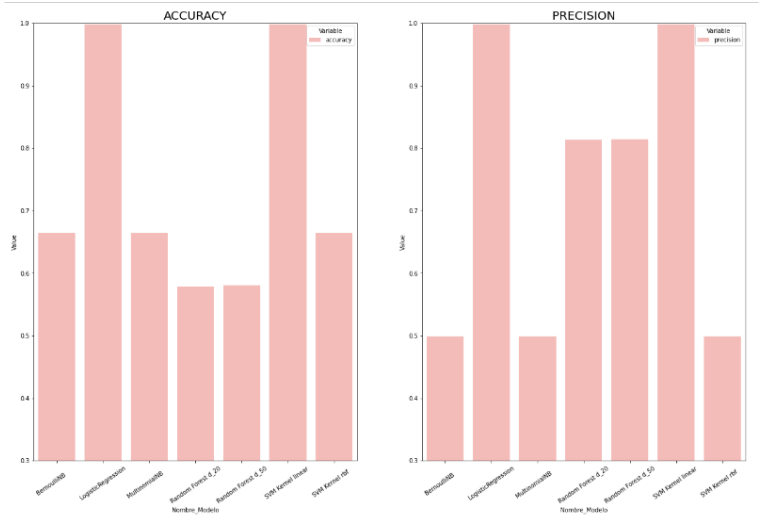


Figura 105: Gráficas de métricas de accuracy y precision del último intento

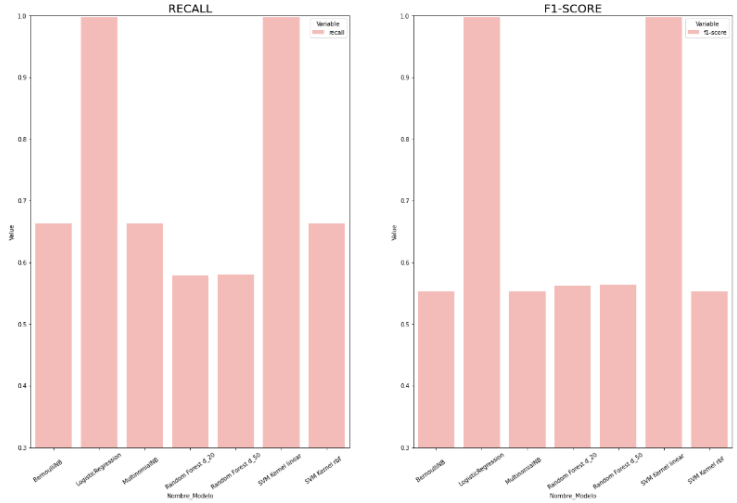


Figura 106: Gráficas de métricas de recall y f1-score del último intento

7.19. *Heatmap* de matrices de confusión

El modelo de *LogisticRegression* fue evolucionando entre los intentos, siempre logró identificar correctamente las tres clases. A lo largo de los distintos intentos fue mejorando la clasificación y la disminución de los falsos positivos y falsos negativos.

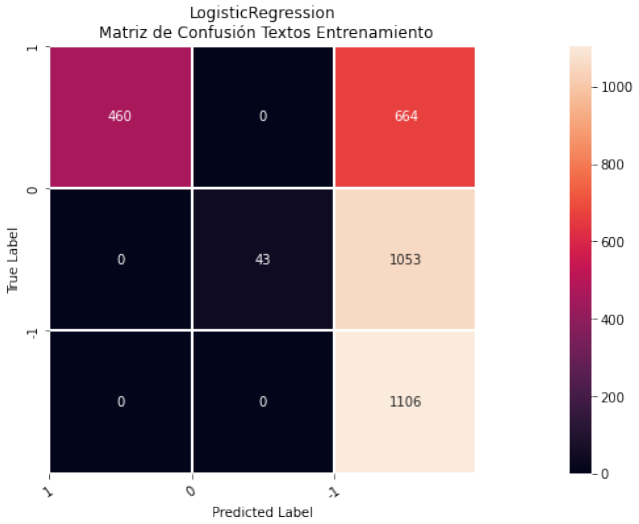


Figura 107: *Heatmap* de la matriz de confusión de modelo de *LogisticRegression* del primer intento

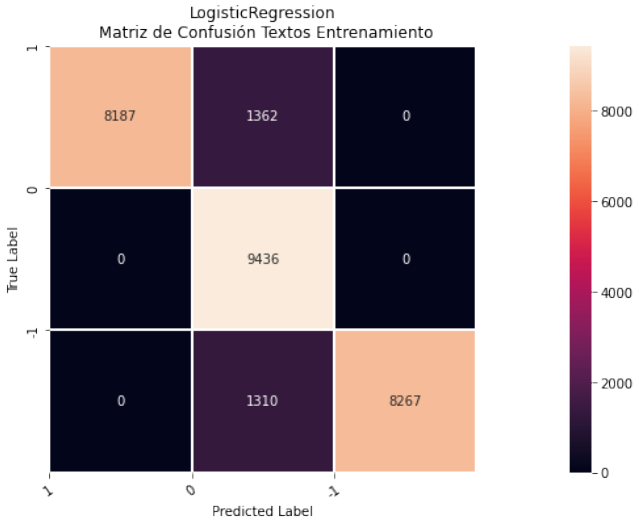


Figura 108: *Heatmap* de la matriz de confusión de modelo de *LogisticRegression* del segundo intento

El modelo de *SVM Kernel linear* fue evolucionando entre los intentos, como el modelo de *LogisticRegression*. Este siempre logró identificar correctamente las tres clases. La cantidad de clasificaciones correctas para la clase neutral fue la más baja en el primer intento, pero aumentó en cantidades para el segundo y el último intento.

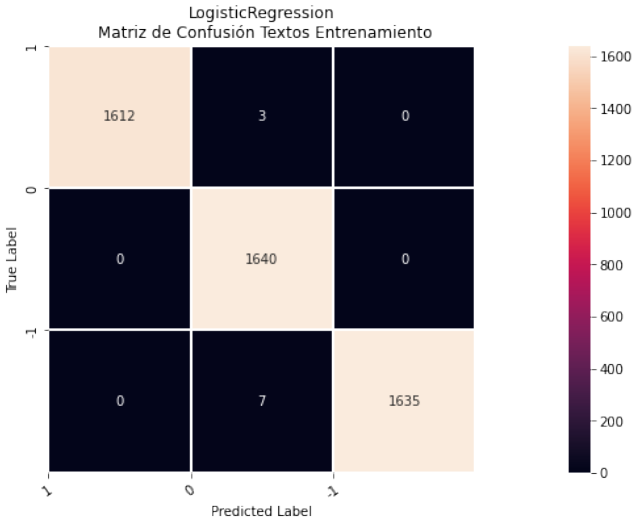


Figura 109: *Heatmap* de la matriz de confusión de modelo de *LogisticRegression* del tercer intento

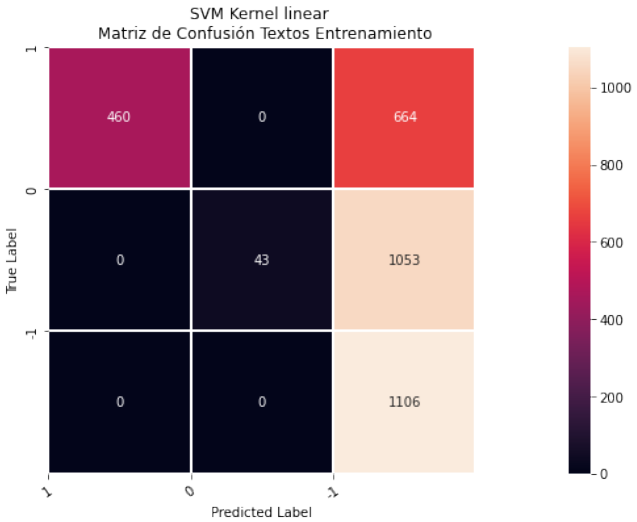


Figura 110: *Heatmap* de la matriz de confusión de modelo de *SVM Kernel linear* del primer intento

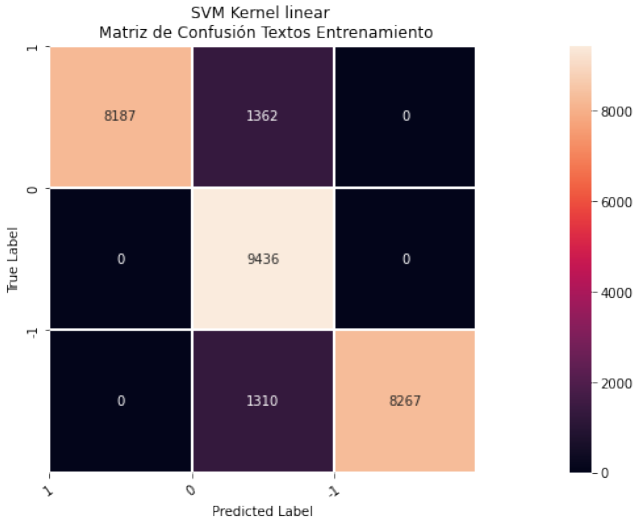


Figura 111: *Heatmap* de la matriz de confusión de modelo de *SVM Kernel linear* del primer intento

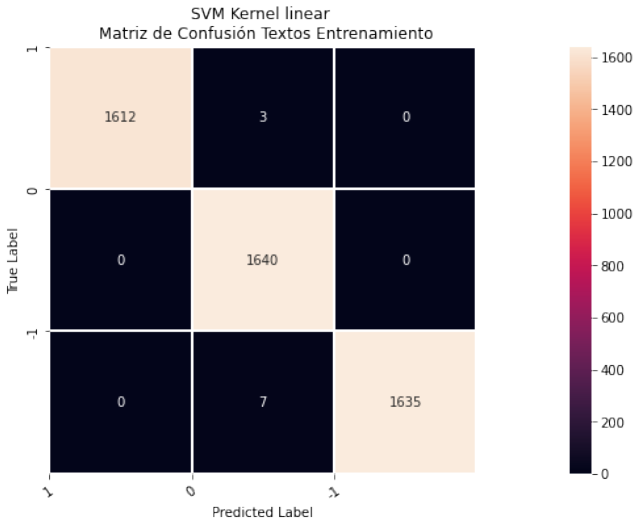


Figura 112: *Heatmap* de la matriz de confusión de modelo de *SVM Kernel linear* del primer intento

7.20. Visualización

A continuación, se muestra un de flujo compuesto por dos consultas personalizadas a la base de datos, unidos en *Tableau Prep*.

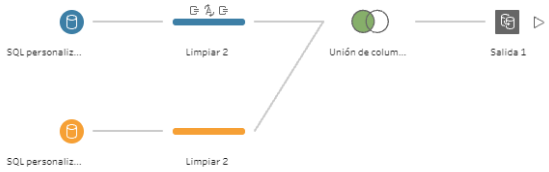


Figura 113: Flujo de datos a importar en *Tableau*

Se crearon tablas y gráficas para mostrar de manera visual el rendimiento de los usuarios estudiantes. De manera, que el conjunto de estas, se vuelva un *dashboard*, y que este tenga la característica de ser interactivo.

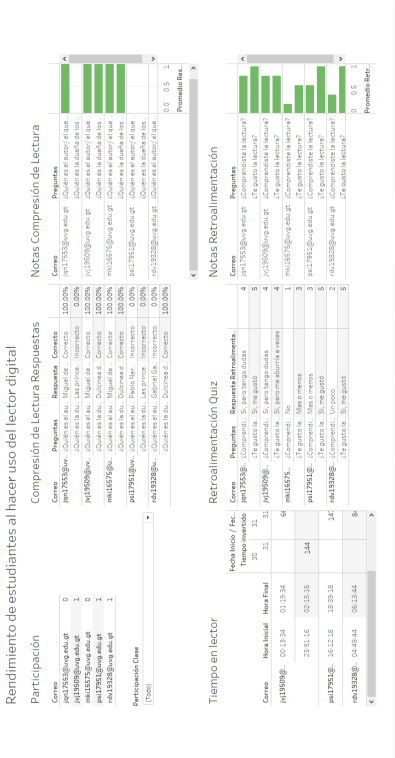


Figura 114: Dashboard de datos recolectados de la base de datos

Un *insight* que se puede ver con las visualizaciones, es que dos alumnos no participaron en una lectura, pero sí lograron contestar preguntas sobre esta.

Otro *insight* que se puede identificar, es que uno de los dos usuarios, contestó correctamente las preguntas de compresión de lectura, pero en la respuesta de retroalimentación podemos ver que respondió que no comprendió la lectura y que no le gustó la misma.

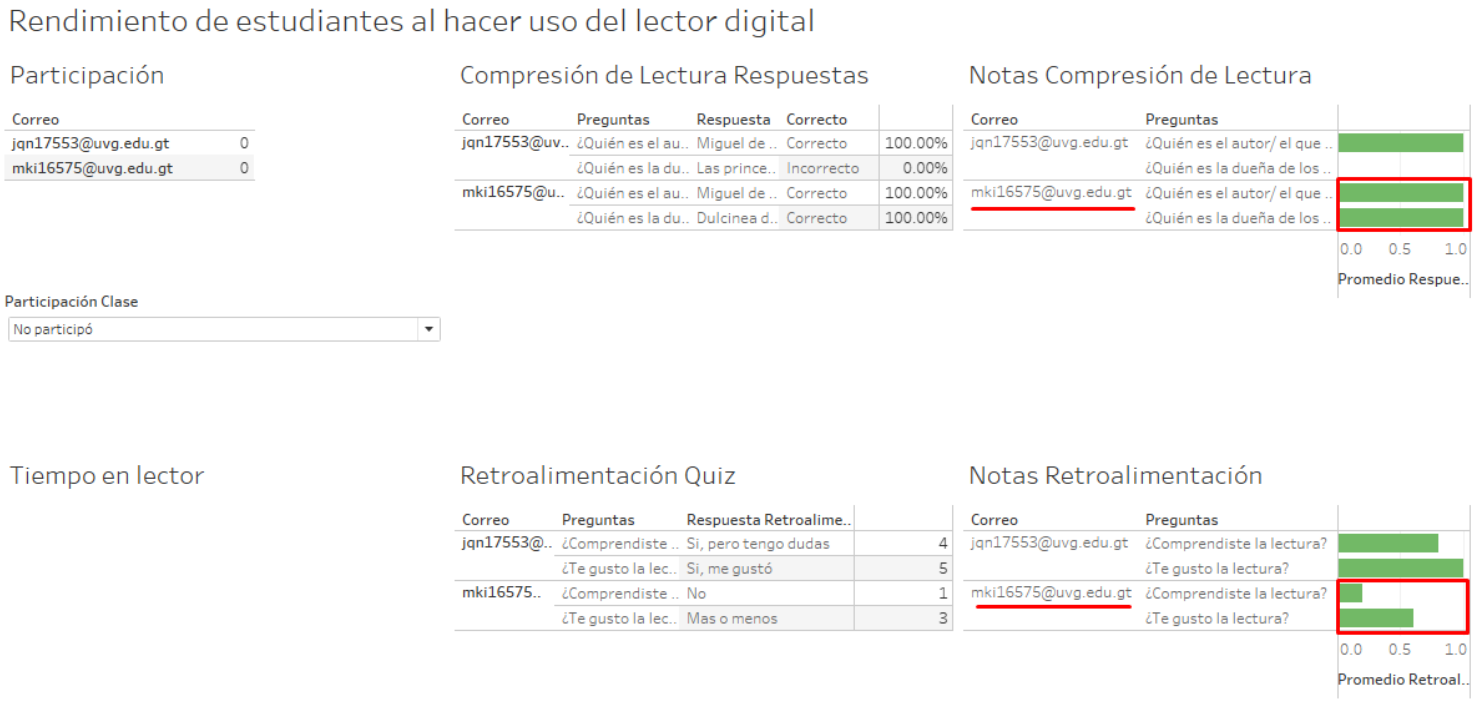


Figura 115: Hallazgos que encontrados al interactuar con el *dashboard*

CAPÍTULO 8

Conclusiones

- Se logró elaborar un *frontend*, un *backend* y un módulo de ciencia de datos que funcionan como base para un lector de publicaciones electrónicas colaborativo para su uso en el ámbito académico.
- Se logró elaborar la interfaz de un lector colaborativo para su uso en el ámbito académico a través de la metodología de *design thinking*. Esta es extensible y se encuentra alineada con las necesidades de usuario encontradas.
- Los perfiles de usuario generados permiten identificar las necesidades de los estudiantes para un lector de publicaciones colaborativo en el ámbito académico.
- El prototipo elaborado es funcional. Este permite resaltar, comentar y leer publicaciones electrónicas, además permite desplegar instancias grupales de libros para ser utilizadas por estudiantes con acceso a ellas. Este permite un uso satisfactorio y puede ser utilizado como base para futuros proyectos.
- La base de código de *frontend* desarrollada permite que futuros desarrolladores la mantengan. Esta puede ser utilizada en futuros proyectos para aumentar la funcionalidad del prototipo actual.
- Se logró implementar metáforas del mundo físico en el diseño del lector. Utilizando la interacción humano-computador se pulieron los diseños para ser intuitivos para los estudiantes.
- Se crearon métodos administrativos que permiten que distintos documentos se puedan cambiar de manera dinámica, permitiendo los cambios para agregar, remover y modificar elementos, tales como exámenes y notas en los archivos.
- Se crearon métodos de tipo *REST* que permiten un estilo de lector con procesamiento de datos mucho más eficiente, según las arquitecturas escogidas. Esto es dicho bajo la impresión que se tiene un bajo impacto en el tráfico de red, así como el procesador del dispositivo. Asimismo, las operaciones dadas abstraen las operaciones en el *backend*, permitiéndose consumir datos de tipos *string*, *bool*, *int* y *json* de la base de datos.

- Se crearon distinciones entre tipos de usuarios existentes con su debida jerarquía extensible, aumentando la escalabilidad del programa. Los usuarios son aprobados por el *frontend* y el público objetivo.
- Se mantiene un nivel de escalabilidad en la aplicación acorde a las filosofías y objetivos seleccionados, permitiendo el desarrollo de cambios constantes según sea la necesidad del negocio.
- Se logró establecer una conexión exitosa a la base de datos lo que permitió albergar la información que se recopila de los *EPUBs*, permitiendo la generación de consultas a la misma, desde la aplicación de *Tableau Prep* para su limpieza y estructuración.
- El análisis del tiempo de atención de las lecturas se logró identificar y mostrar dentro de un tablero interactivo, para los tiempos se tomó en cuenta la cantidad de minutos que se mantiene la sesión total, la hora inicial, la hora final y la cantidad de palabras que lee por minuto.
- Se identificaron dentro de las respuestas de la encuesta, los resultados de cada usuario y se demostró los resultados de cada uno, con ponderaciones sobre los segmentos de la encuesta, para visualizarlos dentro de un *dashboard* en *Tableau*.
- Se crearon tres distintos *wordclounds* de las palabras clave de los comentarios, cada uno consistiendo para cada clase de sentimientos (positivo, neutro, negativo) en la etapa de preprocesamiento de los datos.
- Se completó un *dashboard* montado en la herramienta de *Tableau*, con los datos limpios y estructurados, tomando los datos de los tiempos de lectura y respuestas de la encuesta del grupo a visualizar, para que los usuarios sean libres de interactuar con el *dashboard* y puedan visualizar el rendimiento del grupo.

CAPÍTULO 9

Recomendaciones

- Se recomienda realizar un módulo de integración con *Canvas* y la biblioteca de la Universidad del Valle en caso se quiera integrar en esta institución. Esto permitiría que se utilicen de mejor manera los recursos disponibles y que los catedráticos puedan utilizar literatura presente en la biblioteca para dar sus cursos
- Es clave para la continuación sostenible a largo plazo del proyecto continuar con los principios *SOLID* y el uso de herramientas para limpieza y estilización de código.
- Se recomienda hacer un menú flotante en el lector para permitir una mayor personalización del lector y la ubicación de las herramientas. Esto permitiría replicar el comportamiento de un escritorio donde pueden moverse elementos de acuerdo con las necesidades de estudio.
- Se recomienda realizar un módulo de lectura de textos para poder incorporar de mejor manera a usuarios con ceguera y personas que prefieren los audiolibros. Además, se recomienda seguir trabajando en distintos elementos de accesibilidad en el lector para atender aún más necesidades.
- Es factible utilizar este lector en otros campos como editoriales de libros u otras instituciones educativas. Para hacerlo, se recomienda explorar más a profundidad estos ámbitos.
- Se recomienda la utilización de un sistema de autenticación externo que permita el abstraer la seguridad del sistema a una entidad grande. El uso del sistema actual añade complejidad al sistema, como fue mencionado.
- Se recomienda la creación de **más** micro servicios y la utilización de *software* de contenedores, tales como *Docker*, para el fácil manejo de estos.
- Se recomienda establecer un canal de comunicación centralizado entre el *frontend* y el *backend* para poder tener un proceso de comunicación más eficiente que reduzca potenciales malentendidos en el futuro, especialmente si se realizará un proyecto a ma

- Se recomienda la utilización de tan pocas librerías externas como sea posible gracias a la generación de basura y desperdicio, combinado con la poca seguridad.
- Se recomienda la creación de *Data Flows*, *UMLs*, etc. para el diseño del programa en general.
- Se recomienda el uso de *MongoDB* en conjunto con *MariaDB* para el manejo de la información en vez de una emulación de partes del mismo, ya que la información en masa pudo haber sido más sencilla de manejar de esta manera.
- Se recomienda utilizar los comentarios que se obtengan de los *EPUBs* cuando estos lleguen a ser una cantidad lo suficiente extensa para entrenamiento y *testing* de los modelos.
- Cuando el proyecto tenga una cantidad de tiempo en uso, se puede contratar *Tableau Server*, para que se puedan acceder a los reportes desde cualquier parte. La tarea de actualización de los datos se puede programar para evitar una que una persona lo haga todos los días.

CAPÍTULO 10

Bibliografía

Álvarez. (2021, 16 de agosto). *La importancia de las emociones en el salón de clases: un recurso esencial en la mejora del aprendizaje*. https://mexico.unir.net/educacion/noticias/importancia-emociones-en-salon-de-clases-recurso-esencial-mejora-aprendizaje/

Amazon. (s.f.-a). *What is Amazon EC2?* Consultado el 2 de noviembre de 2022, desde https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Amazon. (s.f.-b). *What is Amazon S3?* Consultado el 2 de noviembre de 2022, desde https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

Amazon. (s.f.-c). *What is AWS*. Consultado el 2 de noviembre de 2022, desde https://aws.amazon.com/what-is-aws/

Aprende Machine Learning. (2017). *Qué es overfitting y underfitting y cómo solucionarlo*. https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/

Babich. (2021). Design Thinking Process and Its Phases | Adobe XD. https://xd.adobe.com/ideas/principles/design-systems/design-thinking-process/

Berkeley Extension. (2020). Front-End vs. Back-End vs. Full-Stack Developers. https://bootcamp.berkeley.edu/blog/front-end-vs-back-end-vs-full-stack-developers/

BIG DATA International Campus. (2020). *Análisis de sentimiento*. https://www.campusbigdata.com/big-data-blog/item/158-analisis-de-sentimiento

Bradley. (2014). Design Principles: Visual Weight And Direction. https://www.smashingmagazine.com/2014/12/design-principles-visual-weight-direction/

Buschmann, F. ., Schmidt, D. . C. ., Regine Meunier, Rohnert, H. ., Stal, M. ., Sommerlad, P. ., Meunier, Kircher & Jain. (1996). *Pattern-Oriented Software Architecture, A System of Patterns*. Wiley.

Coelho. (2016). *PHP-related vulnerabilities on the National Vulnerability Database*. Consultado el 23 de octubre de 2022, desde http://www.coelho.net/php_cve.html

Corrégé, J B., & Michinov, N. (2021). Group Size and Peer Learning: Peer Discussions in Different Group Size Influence Learning in a Biology Exercise Performed on a Tablet With Stylus. *Frontiers in Education*, *6*. https://doi.org/10.3389/feduc.2021.733663

Coursera. (2022, 30 de septiembre). *What Does a Back-End Developer Do?* Consultado el 31 de octubre de 2022, desde https://www.coursera.org/articles/back-end-developer

DataCamp. (2018). *Understanding Random Forests Classifiers in Python Tutorial*. https://www.datacamp.com/tutorial/random-forests-classifier-python

David Eccles School of Business. (2015). 5 Tips for an Effective Study Group. https://eccles.utah.edu/news/5-tips-for-an-effective-study-group/

DeJonckheere, M., & Vaughn, L M. (2019). Semistructured interviewing in primary care research: a balance of relationship and rigour. *Family Medicine and Community Health*, *7*(2), e000057. https://doi.org/10.1136/fmch-2018-000057

Dubey. (2018). *An introduction to Bag of Words and how to code it in Python for NLP*. https://www.freecodecamp.org/news/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04/

Eberts, R. . E. . (1994). *User Interface Design (Prentice-hall International Series in Industrial & Systems Engineering)*. Pearson College Div.

Educative. (s.f.). *What is the F1-score?* https://www.educative.io/answers/what-is-the-f1-score

Eich. (2008, 3 de abril). *Popularity – Brendan Eich*. Consultado el 23 de octubre de 2022, desde https://brendaneich.com/2008/04/popularity/

ESLint. (s.f.). Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter. https://eslint.org/

Essential Accessibility. (2022). ADA Website Accessibility Requirements and Guidelines. https://www.essentialaccessibility.com/blog/ada-guidelines

Foraker Labs. (2015). Usability First - Methods - HCI Design Approaches. https://www.usabilityfirst.com/usability-methods/hci-design-approaches/index.html

Friis & Yu. (2022). What is Design Thinking and Why Is It So Popular? https://www.interaction-design.org/literature/article/what-is-design-thinking-and-why-is-it-so-popular

Gadhavi. (2022). Full-Stack vs MEAN Stack vs MERN Stack: The Right Technology Stack for You in 2022. https://radixweb.com/blog/full-stack-vs-mean-stack-vs-mern-stack-development

Gallo. (2017). A Refresher on A/B Testing. https://hbr.org/2017/06/a-refresher-on-ab-testing

Gandhi. (2018). *Support Vector Machine — Introduction to Machine Learning Algorithms*. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

García. (2021). *Comparativa de técnicas de balanceo de datos. Aplicación a un caso real para la predicción de fuga de clientes*. Universidad de Ovideo.

Gazta. (2021). Good And Bad Of Angular Development. https://www.odinschool.com/blog/programming/good-and-bad-of-angular-development

GeeksforGeeks. (2022). *Using CountVectorizer to Extracting Features from Text*. https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/

Google. (2022). Angular. https://angular.io/guide/what-is-angular

Griffin, A., & Hauser, J R. (1993). The Voice of the Customer. *Marketing Science*, *12*(1), 1-27. https://doi.org/10.1287/mksc.12.1.1

Gunjal. (2020). *Tokenization in NLP*. https://www.kaggle.com/code/satishgunjal/tokenization-in-nlp

Hammar. (2014). Group work as an incentive for learning – students' experiences of group work. *Frontiers in Psychology*, *5*. https://doi.org/10.3389/fpsyg.2014.00558

Hellsten & Karras. (s.f.). This Person Does Not Exist. https://thispersondoesnotexist.com/

IBM. (2021). *Uso de histogramas*. https://www.ibm.com/docs/es/spss-modeler/saas?topic=node-using-histograms

Initiative. (2018). Introducción a las Pautas de Accesibilidad para el Contenido Web (WCAG). https://www.w3.org/WAI/standards-guidelines/wcag/es

IONOS. (2020, 11 de febrero). *Imperative programming: Overview of the oldest programming paradigm*. Consultado el 23 de octubre de 2022, desde https://www.ionos.com/digitalguide/websites/web-development/imperative-programming/

ITMadrid. (2020). Qué es y para qué sirve Design Thinking | ITMadrid. https://www.itmadrid.com/que-es-y-para-que-sirve-design-thinking/

Jabeen. (2018). *Stemming and Lemmatization in Python*. https://www.datacamp.com/tutorial/stemming-lemmatization-python

Johnson, Johnson & Smith. (2013). Cooperative Learning: Improving University Instruction by Basing Practice on Validated Theory. *Journal on excellence in college teaching*, *25*, 85-118. http://celt.miamioh.edu/ject/fetch.php?id=594

json. (s.f.). *JSON*. Consultado el 1 de noviembre de 2022, desde https://www.json.org/json-en.html

Kanade. (2022). What Is HCI (Human-Computer Interaction)? Meaning, Importance, Examples, and Goals. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-hci/

Kay. (2003, 23 de julio). *Dr. Alan Kay on the Meaning of ."Object-Oriented Programming"*. Consultado el 25 de octubre de 2022, desde https://www.purl.org/stefan_ram/pub/doc_kay_oop_en

Keeling, R., Chhatwal, R., Huber-Fliflet, N., Zhang, J., Wei, F., Zhao, H., Shi, Y., & Qin, H. (2019). Empirical Comparisons of CNN with Other Learning Algorithms for Text Classification in Legal Document Review. *2019 IEEE International Conference on Big Data (Big Data)*, 2038-2042. https://doi.org/10.1109/BigData47090.2019.9006248

Kennedy. (2019). 4 Rules for Intuitive UX. https://www.learnui.design/blog/4-rules-intuitive-ux.html

Khushijain. (2022). *Naive Bayes Algorithm: Python Implementation From Scratch*. https://medium.com/analytics-vidhya/naive-bayes-algorithm-implementation-from-scratch-f9a2a12789b5

King. (2022). *Data problem has an evil twin that is much worse*. https://www.openprisetech.com/blog/duplicate-data-problem-evil-twin-much-worse-duplicate-fields/

Kyslova. (2021). SEO tips and tricks for Angular web apps. https://proxify.io/articles/is-angular-good-for-seo

Laplante, P. . A. ., & Kassab, M. . (2022, 3 de noviembre). *What Every Engineer Should Know about Software Engineering* (2.ª ed.). CRC Press.

Lin. (2011). Is it "front-end", "frontend", or "front end". https://english.stackexchange.com/questions/34447/is-it-front-end-frontend-or-front-end

Maaoui. (2021). *why Angular is your best choice for your next projects ?* https://medium.com/@maaouikimo/why-angular-is-your-best-choice-for-you-next-projects-9d754fb18f91

Martin. (2017, 10 de septiembre). *Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series)* (1.ª ed.). Pearson.

Matsutomo. (2021, 28 de julio). *TypeORM - Prevent SQL Injection with Node.js, React and TypeScript*. Consultado el 1 de noviembre de 2022, desde https://dev.to/yoshi_yoshi/typeorm-prevent-sql-injection-with-node-js-react-typescript-in-2021-1go4

MDN. (2022, 21 de septiembre). *MVC*. Consultado el 31 de octubre de 2022, desde https://developer.mozilla.org/en-US/docs/Glossary/MVC

Microsoft. (2005, 8 de octubre). *Introduction to Model/View/ViewModel pattern for building WPF apps*. Consultado el 31 de octubre de 2022, desde https://learn.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps

Microsoft Learn. (s.f.). *Creación y descripción de modelos de clasificación en el aprendizaje automático - Training*. https://learn.microsoft.com/es-es/training/modules/understand-classification-machine-learning/

Microsoft Learn. (2022). *Training and Testing Data Sets*. https://learn.microsoft.com/en-us/analysis-services/data-mining/training-and-testing-data-sets?view=asallproducts-allversions

MongoDB. (s.f.). *What Is NoSQL? NoSQL Databases Explained*. Consultado el 1 de noviembre de 2022, desde https://www.mongodb.com/nosql-explained

Nelsestuen & Smith. (2020). Empathy interviews. *The Learning Professional*, *41*(5), 59-62. https://learningforward.org/wp-content/uploads/2020/10/tool-empathy-interviews.pdf

Nielsen. (2020). 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics/

NLTK. (s.f.). *NLTK :: nltk.tokenize package*. https://www.nltk.org/api/nltk.tokenize.html

Nørmark. (2013). *Programming Paradigms*. Consultado el 23 de octubre de 2022, desde https://homes.cs.aau.dk/%5C%7Enormark/prog3-03/html/notes/paradigms_themes-paradigms.html

Off-Site Services, Inc. (s.f.). Color Check for ADA image compliance. https://www.oss-usa.com/color-check-ada-image-compliance

OneIdentity. (s.f.). *Authentication vs. Authorization: What's the Difference?* Consultado el 1 de noviembre de 2022, desde https://www.onelogin.com/learn/authentication-vs-authorization

Oracle. (s.f.). *What is a Database?* Consultado el 1 de noviembre de 2022, desde https://www.oracle.com/database/what-is-database/

Ortega. (2021). *Desarrollo de motores de búsqueda utilizando herramientas open source* (1.ª ed.). Marcombo.

OWASP. (s.f.). *SQL Injection | OWASP Foundation*. Consultado el 1 de noviembre de 2022, desde https://owasp.org/www-community/attacks/SQL_Injection

Oxford Dictionaries. (2010). *Oxford Dictionary of English*. Oxford University Press.

Pfalzgraf. (2022). Angular vs. Vue (again...) https://dev.to/josunlp/angular-vs-vue-again-315a

Pham. (2022). Mobile application: Definition, Technology types and examples 2022. https://magenest.com/en/mobile-application/

Potel. (1995). MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. *Taligent*. http://wildcrest.com/Potel/Portfolio/mvp.pdf

Praphamontripong. (2022). CS4640 Angular overview. https://www.cs.virginia.edu/%5C%7Eup3f/cs4640/supplement/overview-angular.html

Prettier. (s.f.). What is Prettier? · Prettier. https://prettier.io/docs/en/

Redhat. (2022, 14 de marzo). *What are cloud services?* Consultado el 1 de noviembre de 2022, desde https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services

Richardson. (2021). *What are microservices?* Consultado el 31 de octubre de 2022, desde https://microservices.io/

Riehe & Gross. (1998). *Role Model Based Framework Design and Integration.* Consultado el 23 de octubre de 2022, desde https://riehle.org/computer-science/research/1998/oopsla-1998.html

Roman. (2020). Application. https://www.twilio.com/docs/glossary/what-is-an-application

Roncal. (2021). *Wordcloud hecho en Python - Samuel David Roncal Vidal.* https://medium.com/@davrv93/wordcloud-hecho-en-python-5375f52b60ca

Rosala. (2021). How Many Participants for a UX Interview? https://www.nngroup.com/articles/interview-sample-size/

Rozo, Cusba, Medina & León (Eds.). (2018). Herramientas de analítica para la explotación de datos. https://herramientas.datos.gov.co/sites/default/files/2020-11/Inventario%20herramientas%20anal%C3%ADtica_0.pdf

Rybczynski, S M., & Schussler, E E. (2011). Student Use of Out-of-Class Study Groups in an Introductory Undergraduate Biology Course. *CBE—Life Sciences Education*, *10*(1), 74-82. https://doi.org/10.1187/cbe-10-04-0060

Schoenherr, N. (2020). Discovering why study groups are more effective - The Source - Washington University in St. Louis. https://source.wustl.edu/2006/07/discovering-why-study-groups-are-more-effective/

Singh, A. P., & Tomar, P. . (2016). Web Service Component Reusability Evaluation: A Fuzzy Multi-Criteria Approach. *International Journal of Information Technology and Computer Science*, *8*(1), 40-47. https://doi.org/10.5815/ijitcs.2016.01.05

Socket.IO. (s.f.). Socket.IO. https://socket.io/

Spickerman. (2022). Angular Upgrades: Painless Migration from TSLint to ESLint. https://www.bitovi.com/blog/angular-upgrades-painless-migration-from-tslint-to-eslint

Springer, L., Stanne, M E., & Donovan, S. S. (1999). Effects of Small-Group Learning on Undergraduates in Science, Mathematics, Engineering, and Technology: A Meta-Analysis. *Review of Educational Research*, *69*(1), 21-51. https://doi.org/10.3102/00346543069001021

Tableau. (s.f.). *Hyper Support Resources.* https://www.tableau.com/support/hyper-resources

Tanner, K., Chatman, L S., & Allen, D. (2003). Approaches to Cell Biology Teaching: Cooperative Learning in the Science Classroom—Beyond Students Working in Groups. *Cell Biology Education*, *2*(1), 1-5. https://doi.org/10.1187/cbe.03-03-0010

The Interaction Design Foundation. (2017a). What is Human-Computer Interaction (HCI)? https://www.interaction-design.org/literature/topics/human-computer-interaction

The Interaction Design Foundation. (2017b). What is User Interface (UI) Design? https://www.interaction-design.org/literature/topics/ui-design

Toward AI. (2022). *Multi-class Model Evaluation with Confusion Matrix and Classification Report.* https://towardsai.net/p/l/multi-class-model-evaluation-with-confusion-matrix-and-classification-report

trustRadius. (2019). *Tableau Prep OVERVIEW.* https://www.trustradius.com/products/tableau-prep/reviews

Turbert. (2022). What Is Color Blindness? https://www.aao.org/eye-health/diseases/what-is-color-blindness

UNICEF. (2020). *La falta de igualdad en el acceso a la educación a distancia en el contexto de la COVID-19 podría agravar la crisis mundial del aprendizaje, según UNICEF.* https://www.unicef.org/guatemala/comunicados-prensa/la-falta-de-igualdad-en-el-acceso-la-educaci%5C%C3%5C%B3n-distancia-en-el-contexto-de-la

Universidad del Valle de Guatemala. (2022). Nosotros - UVG Nosotros. https://www.uvg.edu.gt/nosotros/inicio/

UX Design Institute. (2022). What is UI design? A complete introductory guide. https://www.uxdesigninstitute.com/blog/what-is-ui-design/

Vladislavleva, E., Smits, G., & den Hertog, D. (2010). On the Importance of Data Balancing for Symbolic Regression. *IEEE Transactions on Evolutionary Computation*, *14*(2), 252-277. https://doi.org/10.1109/TEVC.2009.2029697

WebAIM. (s.f.). WebAIM: Contrast Checker. https://webaim.org/resources/contrastchecker/

What is a REST API? (s.f.). Consultado el 1 de noviembre de 2022, desde https://www.redhat.com/en/topics/api/what-is-a-rest-api

What Is Information Security? (2022, 16 de agosto). Consultado el 1 de noviembre de 2022, desde https://www.cisco.com/c/en/us/products/security/what-is-information-security-infosec.html

Wickline. (2001). Coblis — Color Blindness Simulator – Colblindor. https://www.color-blindness.com/coblis-color-blindness-simulator/

Wilson, K J., Brickman, P., & Brame, C J. (2018). Group Work. *CBE—Life Sciences Education*, *17*(1), fe1. https://doi.org/10.1187/cbe.17-12-0258

Wisley. (2021). ePub VS PDF, Which One is Better and Why? https://www.cisdem.com/resource/epub-vs-pdf.html

CAPÍTULO 11

Anexos

11.1. Estructura de entrevista a estudiantes

Número de entrevistado:

Carrera y año:

¿Qué te motiva a estudiar?

// Iniciar con lo positivo

¿Qué herramientas utilizas para estudiar? // Identificar si utiliza computadora, teléfono, tableta, libros físicos y qué aplicaciones utiliza. También profundizar en historias que ha tenido con estas herramientas.

¿Cómo estudias usualmente?

// Ver historias que ha tenido de estudio, cómo se prepara, si ha tenido experiencias de estudio fuera de la universidad.

¿Cuál es tu experiencia al estudiar en grupo?

// Que cuente historias sobre estudios grupales o trabajos que ha realizado. Identificar lo que le funciona y lo que no.

¿Cómo crees que las aplicaciones o tecnología podrían impactar tu experiencia de estudio grupal e individual?

// Obtener retroalimentación con su experiencia con tecnología y estudio.

¿Qué buscas en aplicaciones o programas académicos?

// Experiencias que ha tenido con herramientas y aplicaciones al momento de estudiar o desenvolverse en su vida estudiantil.

¿Qué buscas en aplicaciones o programas colaborativos?

// Por ejemplo: historias de trabajo grupal, estudio grupal. Cosas que hayan salido bien y cosas que no.

¿Qué te gustaría tener en una aplicación o programa para leer libros?

// Preguntar sobre funcionalidades que le gustaría ver, elementos que utiliza frecuentemente o si no las utiliza. Si es lector principalmente de libros físicos, indagar sobre cómo traducir esta experiencia a los libros digitales.

¿Cuáles son tus experiencias con aplicaciones o programas de lectura de libros electrónicos?

// Buscar elementos que le han funcionado y los elementos que no. Fomentar historias sobre sus hábitos de lectura dentro y fuera de la universidad.

¿Qué te parecen las aplicaciones o programas donde varias personas pueden editar un solo documento?

// Historias que han tenido con aplicaciones colaborativas, como Google Docs.

11.2. Estructura de entrevista a catedráticos

Número de entrevistado:

Facultad(es) donde imparte clases:

¿Qué le motiva a ser catedrático?

// Iniciar con lo positivo

¿Qué piensa sobre la metodología actual de dar clases?

// Historias sobre elementos que podrían mejorar en la forma actual de dar clases y lo que le ha gustado.

¿Qué tipo de literatura utiliza en sus cursos?

// Profundizar sobre el tipo de libros que utiliza (físicos o digitales) historias de cómo los consiguen los estudiantes.

¿Cuál es su experiencia con tareas o proyectos grupales?

// Historias sobre su metodología de enseñanza y si realiza este tipo de actividades.

¿Qué busca en aplicaciones o programas académicos?

// Experiencias que ha tenido con herramientas y aplicaciones al momento de estudiar o desenvolverse en su vida estudiantil.

¿Qué busca en aplicaciones o programas colaborativos?

// Por ejemplo: historias de trabajo grupal, estudio grupal. Cosas que hayan salido bien y cosas que no.

¿Qué le gustaría tener en una aplicación o programa para leer libros?

// Preguntar sobre funcionalidades que le gustaría ver, elementos que utiliza frecuentemente o si no las utiliza. Si es lector principalmente de libros físicos, indagar sobre cómo traducir esta experiencia a los libros digitales.

¿Qué le parecen las aplicaciones o programas donde varias personas pueden editar un solo documento?

// Historias que han tenido con aplicaciones colaborativas, como Google Docs y si le es útil como catedrático.

¿Qué impacto cree que las herramientas de estudio colaborativo podrían tener en su clase?

// Historias de cómo se han utilizado y cómo podría cambiar en el futuro.

¿Qué estadísticas le gustaría ver de su clase que actualmente no posee?

// En conjunto con *data science* para obtener mejores *insights*.

11.3. Preguntas por prototipo

11.3.1. Prototipo 1

Inicio de sesión

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece el diseño de esta pantalla?
- ¿Cambiarías algo de esta pantalla?
- ¿Qué elementos agregarías, quitarías o modificarías de esta pantalla?

Menú (biblioteca o storefront)

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece el diseño de esta pantalla?
- ¿Cambiarías algo de esta pantalla?
- ¿Qué elementos agregarías, quitarías o modificarías de esta pantalla?

Búsqueda

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece el diseño de esta pantalla?
- ¿Cambiarías algo de esta pantalla?
- ¿Qué elementos agregarías, quitarías o modificarías de esta pantalla?

Lector

- ¿Qué funcionalidades te gustaría ver en este lector?
- ¿Cómo deberías poder interactuar con el lector?
- ¿Cómo lees usualmente?
- ¿Qué elementos agregarías, quitarías o modificarías de esta pantalla?

11.3.2. Prototipo 2

Inicio de sesión

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece la distribución de los elementos?
- ¿Hay algo que se debería agregar, quitar o modificar de esta pantalla?
- Observaciones adicionales

Menú (biblioteca o storefront)

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece la distribución de los elementos?
- ¿Hay algo que se debería agregar, quitar o modificar de esta pantalla?
- Observaciones adicionales

Búsqueda

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece la distribución de los elementos?
- ¿Hay algo que se debería agregar, quitar o modificar de esta pantalla?
- Observaciones adicionales

Lector

- ¿Qué funcionalidades te gusta utilizar al momento de leer una publicación electrónica? (previo a esto se indagó sobre su preferencia de lectura).
- ¿Cómo se debería resaltar y comentar?
- ¿Qué te parece la forma de ver los comentarios y resaltado?
- Observaciones adicionales

11.3.3. Prototipo 3

Inicio de sesión

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece la organización de los elementos?
- ¿Cómo te sentirías utilizando este inicio de sesión?
- Observaciones adicionales

Menú (biblioteca o storefront)

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Qué te parece el menú de la izquierda?
- ¿Cambiarías algo de esta pantalla?
- Observaciones adicionales

Búsqueda

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Te gustaría ver algo más en esta pantalla?
- Observaciones adicionales

Lector

- ¿Cómo realizarías las distintas funcionalidades mostradas en el lector?
- ¿Qué te parece la organización del lector?
- ¿Cómo te hace sentir este diseño?
- ¿Hay algo que te gustaría ver en el lector adicional a lo plasmado en este diseño?
- Observaciones adicionales

11.3.4. Prototipo 4

Inicio de sesión

- ¿Qué es lo primero que miras en esta pantalla?
- ¿Cómo te hace sentir esta pantalla?
- ¿Hay algo que te molesta en esta pantalla?
- Observaciones adicionales

Menú (biblioteca o storefront)

- ¿Qué es lo primero que miras de esta pantalla?
- ¿Qué te parece el menú?
- ¿Cómo te hace sentir esta pantalla?
- Observaciones adicionales

Búsqueda

- ¿Qué es lo primero que miras de esta pantalla?
- ¿Hay algo que no entiendes sobre esta pantalla?
- Observaciones adicionales

Lector

- ¿Qué es lo primero que miras de esta pantalla?
- ¿Qué te parece la distribución del lector?
- ¿Hay algo que te gustaría ver en el lector adicional a lo plasmado en este diseño?
- Observaciones adicionales

11.4. Contraste entre la paleta sólida y distintos fondos

Cuadro 14: Contraste entre la paleta sólida y fondos de color

Color de fondo	Condición visual					
	Normal	**Protanopia**	**Deuteranopia**	**Tritanopia**	**Monocromía del cono azul**	**Acromatopsia**
	1.36	1.37	1.38	1.37	1.52	1.60
	1.73	1.74	1.74	1.72	1.89	1.96
Blanco	1.86	1.85	1.85	1.86	2.10	2.19
	2.67	2.64	2.66	2.67	2.67	2.55
	2.40	2.42	2.42	2.39	2.59	2.67
	1.29	1.29	1.31	1.29	1.44	1.51
	1.63	1.65	1.65	1.63	1.78	1.84
Blanco amarillento	1.75	1.74	1.75	1.75	1.98	2.06
	2.53	2.49	2.52	2.52	2.52	2.40
	2.27	2.28	2.29	2.26	2.44	2.52
	5.13	5.05	5.07	5.11	4.57	4.36
	4.04	3.97	4.02	4.06	3.70	3.57
Gris	3.78	3.75	3.79	3.76	3.33	3.19
	2.61	2.62	2.63	2.62	2.61	2.74
	2.91	2.87	2.90	2.92	2.70	2.61
	15.38	15.25	15.16	15.32	13.72	13.07
	12.12	12.00	12.03	12.16	11.10	10.70
Negro	10.42	11.34	11.33	11.28	9.98	9.57
	7.85	7.92	7.86	7.85	7.84	8.22
	8.73	8.67	8.67	8.75	8.09	7.83

11.5. Resaltado sobre libros en diferentes condiciones de visión

En la Figura 116 se pueden observar los distintos temas del lector bajo diferentes condiciones de visión. De izquierda a derecha se presentan: visión normal, protanopia, deuteranopia, tritanopia, monocromacía del cono azul y acromatopsia.

Los temas de los libros, de arriba hacia abajo son: claro, libro, gris, oscuro, claro (resaltado gris) y libro (resaltado gris).

Figura 116: Paleta de colores de resaltado en temas de libros con diferentes condiciones de visión

11.6. Repositorios

Se adjuntan los enlaces a los distintos repositorios pertenecientes a este proyecto.

11.6.1. Frontend

Repositorio - https://github.com/CriPQ/front-end-web

11.6.2. Backend

Repositorio - https://github.com/CriPQ/Back-end

11.6.3. Ciencia de datos

Repositorio del proyecto

Repositorio - https://github.com/CriPQ/data-science-visualization

Dataset de comentarios

Repositorio - https://www.kaggle.com/competitions/spanish-arilines-tweets-sentiment-analysis/submissions