PROYECTO DE GRADO

SISTEMA DE GESTIÓN ADMINISTRATIVA PARA MEJORAR LA EFICIENCIA EN EL REGISTRO Y PRÉSTAMO DE PARTITURAS

PARTE 2: FRONT-END

CASO: FUNDACIÓN PROMÚSICA, DIRECCIÓN DE CULTURA-ULA

Por

Br. José Jorge Briceño Araujo

Tutor: Dr. Rafael Rivas Estrada

Abril 2024

RESUMEN

Este proyecto se enfocó en desarrollar una interfaz gráfica eficiente para el sistema de gestión administrativa de la Fundación Promúsica, específicamente orientado al registro y préstamo de partituras. El análisis de las necesidades de los usuarios condujo al diseño de una interfaz intuitiva y fácil de usar, optimizando la eficiencia operativa de la organización.

La exitosa integración con una API existente garantiza la correcta manipulación de la información, fortaleciendo la capacidad del sistema para adaptarse a futuras actualizaciones y requisitos tecnológicos. La implementación y capacitación del personal aseguran una transición efectiva hacia el nuevo sistema, mejorando significativamente la eficiencia en las operaciones diarias.

Aunque el proyecto enfrentó limitaciones, como la dependencia de la información proporcionada por la Fundación Promúsica, se superaron para alcanzar los objetivos propuestos. Las recomendaciones futuras incluyen la continuación del diálogo con la Fundación para abordar posibles mejoras en el diseño de la interfaz y la exploración de oportunidades para ampliar las funcionalidades del sistema, siempre adaptándose a las necesidades cambiantes de la organización. Este proyecto sienta las bases para un continuo desarrollo y optimización del sistema, consolidando su contribución a la eficiencia de la gestión administrativa en el ámbito musical de la Fundación Promúsica.

Palabras clave: gestión administrativa, desarrollo de software, aplicación web,

gestión de préstamos, metodología de prototipos.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

CAPÍTULO 1

INTRODUCCIÓN

La Tecnología ha posibilitado que el ser humano avance y alcance sus objetivos de manera rápida, eficiente y óptima, gracias a la ayuda de sistemas automatizados. Situaciones que antes generaban inconvenientes en el control de la productividad ahora se resuelven de manera eficaz. Además, la cantidad de información generada actualmente por empresas o instituciones, que incluye datos técnicos, documentación, entre otros, ha experimentado un crecimiento considerable. Esta progresión, aunque beneficiosa, también presenta desafíos en términos de control, organización y acceso a la información.

La fundación Promúsica es un pilar fundamental en la promoción y difusión de la música en la ciudad de Mérida. Esta contiene composiciones elaboradas por distintos creadores a lo largo de la historia que se han documentado en formatos como partituras que sirven de soporte tanto a estudiantes como docentes para sus clases y prácticas. A pesar de su invaluable y arduo trabajo por mantener y compartir con los estudiantes y profesores estos documentos, el registro de préstamo de partituras se realiza de forma manual lo que representa desafíos significativos en términos de eficiencia y organización y control.

En este contexto surge este proyecto en el cual el objetivo general consiste en desarrollar una interfaz gráfica intuitiva y eficiente, que abarque las necesidades específicas de la fundación Promúsica. El análisis detallado de los requerimientos de los usuarios, la creación de una interfaz

gráfica, la integración con una API existente para garantizar la correcta manipulación de la información, y la implementación del sistema son pasos clave para lograr una solución integral y adaptada a las demandas de la fundación, que no solo simplifique, sino que potencie la eficiencia en la manipulación de partituras, permitiendo a estudiantes, músicos y profesores acceder de manera rápida y ordenada a este invaluable recurso. Para alcanzar este propósito, se han delineado objetivos específicos que abarcan desde el análisis de las necesidades de los usuarios hasta la implementación y capacitación del personal en el uso del sistema.

Este proyecto no solo busca satisfacer las necesidades actuales de gestión de préstamos de partituras, sino que también sienta las bases para futuras expansiones y mejoras, asegurando la adaptabilidad y longevidad del sistema en un entorno tecnológico.

1.1. Antecedentes de la Investigación

Los siguientes trabajos realizaron investigaciones a fines:

1.1.1. Desarrollo de un Sistema de Gestión de Biblioteca en la Institución Educativa Técnico Industrial Pedro A. Oñoro de Baranoa

Desarrollo de un sistema de gestión de biblioteca en la Institución Educativa Técnico Industrial Pedro A. Oñoro de Baranoa. Este sistema fue diseñado en la plataforma Visual Studio 2017, Asp.Net y el motor de base de datos MySQL. El propósito del sistema era permitir al encargado de la biblioteca llevar un seguimiento y orden de toda la información almacenado en el sistema, así como conocer de manera rápida y segura los datos de los libros, estudiantes, usuarios, datos de entrada y salida de los libros que se realizan diariamente, sin tener que recurrir a los

archivos manuales obsoletos que se utilizan en aquel momento. La propuesta buscaba solucionar un problema que se había presentado por mucho tiempo en la institución, ya que profesores y estudiantes no sabían realmente qué libros, textos y otros recursos estaban disponibles en la biblioteca escolar (Ruiz, 2021).

1.1.2. Sistema de Gestión y Digitalización Bibliotecaria. Caso: Carrera de Odontología U.P.E.A.

Sistema de gestión y digitalización bibliotecaria. Caso: carrera de odontología U.P.E.A. La investigación describe la propuesta de un sistema de gestión y digitalización bibliotecaria para mejorar el servicio al personal y usuarios de la biblioteca de la carrera de odontología, que tenía un control semiautomático y limitaciones en el acceso a la información y préstamo de materiales bibliográficos. El sistema se desarrolló utilizando la metodología UWE y tecnología Web, con PHP como lenguaje de programación y MariaDB como gestor de base de datos. Además, se realizó una evaluación de la calidad de producto de software con ISO/IEC 9126 y una estimación de costo de software con el modelo COCOMO II (Gutierrez, 2020).

1.1.3. Diseño e Implementación de una Aplicación para la Gestión de Partituras

Diseño e Implementación de una aplicación para la gestión de partituras. Esta investigación describe un trabajo que se enfocó en el desarrollo e implementación de una aplicación web para la gestión de partituras, con el objetivo de almacenar las partituras subidas por los usuarios y desarrollar una capa social y un motor de búsqueda. La aplicación cuenta con una arquitectura en

capas, con la capa del servidor implementada a través del framework Django y la capa del cliente implementada a través de la biblioteca React. El proyecto se desarrolló utilizando la metodología ágil Scrum (Pérez, 2022).

1.2. Planteamiento del problema

La fundación Promúsica es una organización dedicada a la promoción y difusión de la música en la ciudad; la cual está adscrita a la dirección de cultura de la Universidad de Los Andes.

La fundación posee una extensa compilación de partituras empleadas por estudiantes, músicos y profesores en sus prácticas y actuaciones. No obstante, la gestión y préstamo de estas partituras se lleva a cabo de manera manual, lo cual puede resultar incómodo tanto para los usuarios como para el personal encargado.

Este método ha ocasionado varios inconvenientes, como la pérdida de partituras, demoras en los préstamos y dificultades para mantener un registro pormenorizado del uso de las partituras.

El registro manual de las partituras implica que cada vez que un usuario desea hacer uso de una partitura, debe acudir al personal encargado para solicitarla, quien a su vez debe buscarla en los estantes y registrar la información manualmente. Esto no solo produce demoras en el procedimiento, sino que también incrementa la posibilidad de perder las partituras.

Además, la falta de un sistema de gestión administrativa eficiente ha dificultado la tarea de llevar un registro detallado del uso de las partituras. Esto implica que la fundación no cuenta con una base de datos actualizada que permita conocer la disponibilidad de las partituras, las fechas en que fueron prestadas, quiénes las utilizaron, entre otros datos relevantes.

En este sentido, se evidencia la necesidad de contar con un sistema de gestión administrativa que permita mejorar la eficiencia en el registro y préstamo de partituras en la fundación Promúsica.

Para abordar este problema, se propone desarrollar un sistema de gestión administrativa que permita optimizar el proceso de registro y préstamo de partituras, al mismo tiempo que facilite el mantenimiento de un registro detallado y actualizado sobre su utilización.

Lo antes expuesto nos lleva a formular la siguiente interrogante: ¿Cómo puede el diseño e implementación de una aplicación web para un sistema de gestión administrativa contribuir significativamente a la gestión efectiva en el registro y préstamo de partituras en la fundación Promúsica?

1.3. Justificación

El proyecto sobre la implementación de un sistema de gestión administrativa para el registro y préstamo de partituras en la Fundación Promúsica, ubicada en el Edificio San José, Av. 5 Zerpa, con Calle 24, Mérida Venezuela, responde de manera práctica a los desafíos evidentes que actualmente enfrenta la organización. La gestión manual de las partituras ha demostrado ser ineficiente, generando problemas recurrentes como la pérdida de partituras, retrasos en los préstamos y la falta de un registro detallado del uso de las mismas.

La solución práctica que ofrece este proyecto reside en la implementación de un sistema que no solo automatiza y agiliza estos procesos, sino que también aborda directamente los problemas existentes. La rápida localización de partituras mediante un sistema digitalizado reducirá drásticamente el riesgo de extravío, asegurando que cada partitura esté disponible cuando

sea necesaria. Además, al agilizar el proceso de préstamo, se eliminarán los retrasos innecesarios, mejorando la experiencia tanto para el personal como para los usuarios.

Este proyecto no solo resuelve de manera práctica los problemas operativos existentes en la Fundación Promúsica, sino que también proporciona una herramienta eficaz para mejorar la experiencia general de quienes hacen uso de las partituras en esta institución.

1.4. Objetivos

1.4.1. Objetivo General

Desarrollar la interfaz gráfica de un sistema de gestión administrativa para mejorar la eficiencia en el registro y préstamo de partituras en la fundación Promúsica, adscrita a la dirección de cultura de la Universidad de Los Andes.

1.4.2. Objetivos Específicos

Para cumplir con el objetivo general, se plantean los siguientes objetivos específicos:

- Analizar las necesidades de los usuarios de la Fundación Promúsica para diseñar una interfaz amigable y fácil de usar.
- Desarrollar una interfaz gráfica que permita el registro y préstamo de partituras de manera intuitiva y eficiente.
- Integrar la interfaz con una API existente para asegurar la correcta manipulación de la información.

- Implementar el sistema de gestión administrativa en la fundación Promúsica y capacitar al personal encargado en su uso.

1.5. Metodología

La metodología de desarrollo de prototipos es una estrategia efectiva para crear aplicaciones web de manera iterativa y centrada en el usuario. Este enfoque se caracteriza por la creación gradual de versiones funcionales de la aplicación, lo que permite obtener retroalimentación temprana y frecuente.

El proceso se desglosa en varias etapas claves:

1. **Identificación de los requisitos:** en esta etapa se identifican los requisitos funcionales y no funcionales del software, así como las limitaciones técnicas y de recursos.
2. **Diseño del prototipo:** se crea un diseño del prototipo que incluye las características y funciones más importantes que se quieren validar.
3. **Desarrollo del prototipo:** se crea el prototipo de software utilizando herramientas y tecnologías que permitan una rápida creación de interfaces de usuario y funcionalidades.
4. **Evaluación del prototipo:** se realiza una evaluación exhaustiva del prototipo para identificar posibles errores, deficiencias o mejoras necesarias.
5. **Retroalimentación del cliente:** se presenta el prototipo al cliente o usuario final para obtener retroalimentación y validación de los requisitos y funcionalidades incluidos.
6. **Ajuste del prototipo:** se realizan los ajustes necesarios en el prototipo para satisfacer las necesidades y requerimientos del cliente o usuario final.

7. **Desarrollo completo del software:** una vez validado y ajustado el prototipo, se inicia el desarrollo completo del software utilizando las especificaciones y requerimientos definidos en el prototipo.

Este modelo de prototipos permite una validación temprana del software, lo que reduce los costos y riesgos asociados a la corrección de errores y la insatisfacción del cliente. Además, permite una iteración más rápida del diseño y las funcionalidades, lo que mejora la calidad y eficiencia del proceso de desarrollo (HostingPlus, 2021).

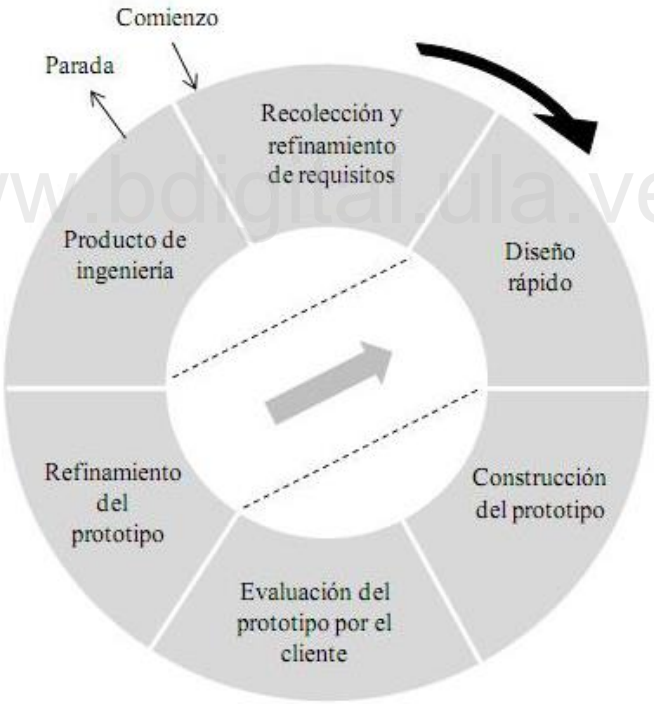

Figura 1: Modelo de desarrollo de prototipo (EcuRed, 2021)

1.6. Alcances y Limitaciones

1.6.1. Alcances:

- Se llevará a cabo un análisis detallado de las necesidades de los usuarios de la fundación Promúsica para diseñar una interfaz que sea fácil de usar y se adapte a sus requerimientos.
- Se desarrollará una interfaz gráfica que permita el registro y préstamo de partituras de manera intuitiva y eficiente, utilizando las mejores prácticas de diseño de interfaces de usuario.
- Se integrará la interfaz con una API existente para garantizar que la información se manipule de manera correcta y segura.
- Se implementará un sistema de gestión administrativa para la Fundación Promúsica, que permita el seguimiento y control de los préstamos de partituras y la gestión de las actividades relacionadas. Se capacitará al personal encargado en el uso del sistema para su correcto funcionamiento.

1.6.2. Limitaciones:

- El análisis de las necesidades de los usuarios se basará en la información proporcionada por la fundación Promúsica y puede ser limitado por la disponibilidad de dicha información.

- La interfaz gráfica se diseñará para cubrir las necesidades de los usuarios identificadas en el análisis, pero es posible que algunos requerimientos específicos no puedan ser satisfechos.
- La integración con la API existente dependerá de la disponibilidad y funcionalidad de la misma.
- La implementación del sistema de gestión administrativa se limitará a las funciones y características específicas identificadas por la fundación Promúsica.

CAPÍTULO 2

MARCO TEÓRICO

En este capítulo, se detalla el marco teórico que sustentará y enriquecerá la investigación, estableciendo las bases teóricas y definiendo los términos fundamentales que contextualizan el estudio. Al explorar este marco, se buscará proporcionar una comprensión profunda de los conceptos esenciales y las teorías relevantes que forman el cimiento de la investigación.

2.1. Bases Teóricas

2.1.1. Sistema de Información

O'Brien y Marakas (2006), en su libro sistemas de información gerencial, definen un sistema de información como un conjunto de componentes interrelacionados que recopilan, procesan, almacenan y distribuyen información con el fin de apoyar la toma de decisiones, la coordinación y el control en una organización. En esta definición, se destaca la importancia de los componentes interrelacionados y la función del sistema de información para apoyar la gestión de la organización.

La definición de sistema de información de estos autores, es relevante en este contexto, debido a que, un sistema de gestión administrativa para la gestión de partituras es, en esencia, un sistema de información que recopila, procesa, almacena y distribuye información relevante para la toma de decisiones y el control dentro de la organización.

2.1.2. Tecnología de la Información (TI)

Langer (2018), define la tecnología de la información como la combinación de hardware, software y servicios que se utilizan para gestionar y procesar la información dentro de una organización.

La definición de Langer destaca la importancia de la tecnología de la información en la administración y el procesamiento de la información dentro de una organización. La tecnología de la información es fundamental para la construcción y el funcionamiento de un sistema porque ofrece las herramientas necesarias para la gestión de datos y el intercambio de información entre los numerosos componentes del sistema.

Los sistemas pueden procesar, almacenar y transportar información de manera confiable y efectiva gracias al hardware, software y servicios de tecnología de la información, que respaldan la toma de decisiones corporativas y la formulación de estrategias. La tecnología de la información es crucial para la seguridad del sistema porque ofrece instrumentos para la protección de la información contra el acceso no autorizado y la recuperación de la información en caso de falla del sistema.

2.1.3. Sistema de Gestión Administrativa

Un sistema de gestión administrativa se refiere al conjunto de herramientas tecnológicas que utilizan las empresas para la automatización y mejora de sus procesos administrativos, con el objetivo de incrementar su eficiencia y productividad. (Overview of the Administrative Management Systems (AMS), 2016).

Debido a que permitiría la automatización y optimización de los procesos administrativos asociados a la gestión de partituras de la fundación Promúsica, la creación de un sistema para aumentar la eficiencia en el registro y préstamo de partituras es crucial porque conduciría a un mayor nivel de productividad en la gestión de estos recursos.

Herramientas como bases de datos de clientes, sistemas de monitoreo de préstamos y devoluciones, software de gestión de inventario y otros pueden ser parte de un sistema de gestión administrativa. Al automatizar estas operaciones, sería posible acelerar la gestión de recursos, disminuir la cantidad de tiempo requerido para los procesos de registro y préstamo, y eliminar por completo los errores humanos.

Un sistema de gestión administrativa también permitiría mantener un registro preciso y actualizado del inventario de partituras, lo que ayudaría en la toma de decisiones y la planificación a largo plazo. Además, podría ayudar a maximizar la capacidad de almacenamiento de las partituras, así como a mejorar la seguridad y la administración del acceso a las partituras.

2.1.3. Sistemas Integrados de Gestión Bibliotecaria (SIGB)

La gestión eficiente de recursos y servicios es fundamental para el éxito de cualquier organización, incluyendo las bibliotecas. Los sistemas de gestión de bibliotecas (SIGB) son herramientas tecnológicas que permiten gestionar de manera efectiva los recursos y servicios de una biblioteca.

Gavilán (2008), define los sistemas integrados de gestión bibliotecaria (SIGB) como herramientas tecnológicas que permiten la gestión automatizada de los recursos y servicios de una

biblioteca, incluyendo la adquisición, catalogación, circulación, préstamo y control de inventario de los materiales bibliográficos.

Este tipo de programas surgen como un intento de conseguir que las unidades de información se conviertan en centros más eficaces, con capacidad de poder gestionar de manera más eficiente sus recursos y la posibilidad de comunicación más viable con los usuarios.

Un SIGB, integra en un solo programa informático un conjunto de aplicaciones específicas que se denominan módulos, pensados para la facilitación de las tareas específicas de este, las cuales están directamente relacionadas unas con otras. Toda la información reunida, se almacena en una misma base de datos que permite el mejor intercambio de la información y el aprovechamiento de los recursos con el menor esfuerzo posible.

2.1.4. Biblioteca Musical

Smiraglia (2001) propone la siguiente definición para biblioteca musical: es una biblioteca especializada en la adquisición, procesamiento, almacenamiento y acceso de materiales relacionados con la música en cualquier formato.

La definición implica que una biblioteca musical tiene un enfoque específico en la música y sus materiales relacionados, lo que incluye partituras en diferentes formatos (por ejemplo, partituras impresas, partituras digitales, archivos de audio, etc.). Al tener una comprensión clara de las necesidades de una biblioteca musical, se puede diseñar un sistema de gestión de partituras para la fundación Promúsica que incluya las características y funcionalidades específicas necesarias para satisfacer sus necesidades.

Un sistema de gestión de partituras debe ser capaz de adquirir, procesar, almacenar y permitir el acceso a las partituras en diferentes formatos, ya sea en forma impresa o digital. Debe contar con herramientas de catalogación, metadatos y búsqueda que permitan a los usuarios buscar y encontrar las partituras de manera eficiente y precisa. También debe tener herramientas de visualización y reproducción de partituras digitales para permitir a los usuarios ver y escuchar las partituras de manera clara y precisa.

2.1.5. Plataforma Digital

Ross et al. (2019), definen la plataforma digital como un conjunto de tecnologías, estándares y acuerdos comerciales que, cuando se combinan con el contenido digital, proporcionan una experiencia integral y personalizada para los clientes, usuarios y empleados. Estos autores, argumentan que una plataforma digital exitosa debe ofrecer una solución integral a las necesidades de los usuarios, que incluye la entrega de contenido y la capacidad de interactuar y realizar transacciones en línea. También destaca la importancia de la colaboración entre las empresas y los proveedores de tecnología para construir y mantener una plataforma digital efectiva.

2.1.6. Gestión de Procesos de Negocio

La gestión de procesos de negocio (BPM, por las iniciales de la expresión en inglés Business Process Management) constituye uno de los tópicos más pronunciados cuando se abordan las Tecnologías de Información (TI) aplicadas al entorno empresarial. BPM se considera un enfoque multidisciplinario ya que presenta conectores con elementos empresariales y tecnológicos altamente relacionados entre sí. Bajo el paradigma BPM estos procesos se conciben en un ciclo donde son modelados electrónicamente y pueden ser analizados y mejorados como resultado de

varias instancias de procesos ejecutados. Este ciclo BPM se sustenta por los sistemas BPM (BPMS). Las BPMS ofrecen componentes de software integrados en un entorno único que se pueden clasificar en: herramientas de modelado, herramientas de simulación, motores de ejecución, integración de aplicaciones, portales web y monitorización (Cruz et al., 2020).

2.2. Ingeniería de Requisitos

La ingeniería de requisitos es un enfoque sistemático a través del cual el ingeniero de software recopila requisitos de diferentes fuentes y los implementa en los procesos de desarrollo de software. Las actividades de ingeniería de requisitos cubren todo el ciclo de vida del desarrollo de sistemas y software. El proceso de ingeniería de requisitos es un proceso iterativo que también indica que la gestión de requisitos se entiende como un aspecto del proceso de ingeniería de requisitos (An Effective Requirement Engineering Process Model for Software Development and Requirements Management, 2010). Es común clasificar los requisitos en funcionales y no funcionales.

2.2.1. Requisitos Funcionales

Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas (Alarcón, 2006).

Por consiguiente, estos requisitos establecen la base sobre la cual se construirá el sistema. Los requisitos funcionales especifican las funcionalidades, tareas y procesos que el software debe

cumplir, lo que significa que son fundamentales para el diseño, implementación, pruebas y validación del software.

Por tanto, los requisitos funcionales son esenciales para garantizar que el software cumpla con las necesidades y expectativas de los usuarios finales. Si los requisitos funcionales no se definen adecuadamente, es probable que el software no cumpla con los objetivos previstos, lo que puede llevar a la insatisfacción del usuario, a la necesidad de realizar cambios costosos y a la posible pérdida de oportunidades de negocio. En consecuencia, es fundamental que los requisitos funcionales sean claros, precisos y completos para que el software pueda ser desarrollado de manera efectiva.

2.2.2. Requisitos no Funcionales

Los requisitos no funcionales tienen que ver con características que de una u otra forma puedan limitar el sistema, por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, etc. Son restricciones de los servicios o funciones ofrecidos por el sistema (Alarcón, 2006).

Es por ello, que los requisitos no funcionales proporcionan limitaciones y especificaciones importantes que se deben cumplir para garantizar que el sistema cumpla con las expectativas de los usuarios. Si estos requisitos no se cumplen, el sistema puede no ser efectivo, eficiente o seguro para su uso, por lo que es fundamental tenerlos en cuenta desde el inicio del proyecto y considerarlos durante todo el ciclo de vida del software.

2.3. Arquitectura de Software

La arquitectura de software es la estructura de un producto de software. Esto incluye elementos, las propiedades visibles externamente de los elementos y las relaciones entre los elementos (Bass et al., 2012).

En base a lo anterior, Lilienthal (2019) dice que esta definición habla deliberadamente de elementos y relaciones en términos muy generales. Estos dos materiales básicos se pueden utilizar para describir una amplia variedad de vistas arquitectónicas. La vista estática (módulo) contiene los siguientes elementos: clases, paquetes, espacios de nombres, directorios y proyectos; en otras palabras, todos los contenedores que puede usar para programar código en ese lenguaje de programación en particular. En la vista de distribución, se pueden encontrar los siguientes elementos: archivos (JAR, WAR, ensamblados), computadoras, procesos, protocolos y canales de comunicación, etc. En la vista dinámica (tiempo de ejecución) estamos interesados en los objetos de tiempo de ejecución y sus interacciones.

2.3.1. Arquitectura Cliente-Servidor

La arquitectura cliente-servidor de una red informática es aquella en la que muchos clientes (procesadores remotos) solicitan y reciben servicios de un servidor centralizado (computadora host). Las computadoras cliente proporcionan una interfaz para permitir que un usuario de computadora solicite servicios del servidor y muestre los resultados que devuelve el servidor. Los servidores esperan que lleguen las solicitudes de los clientes y luego las responden (The Editors of Encyclopaedia Britannica, 2021).

2.3.2. Servidor

Un servidor es un sistema que contiene datos o proporciona recursos a los que deben acceder otros sistemas de la red. Los tipos de servidor comunes son servidores de archivos que almacenan archivos, servidores de nombres que almacenan nombres y direcciones, servidores de aplicaciones que almacenan programas y aplicaciones y servidores de impresión que planifican y dirigen los trabajos de impresión al destino (*IBM Documentation*, 2021).

2.3.3. Cliente

Un cliente es un sistema que solicita servicios o datos de un servidor. Un cliente puede solicitar código de programa actualizado o el uso de aplicaciones de un servidor de código. Para obtener un nombre o una dirección, un cliente se pone en contacto con un servidor de nombres. Un cliente también puede solicitar archivos y datos para la entrada de datos, las consultas o la actualización de registros de un servidor de archivos (*IBM Documentation*, 2021).

2.3.4. MVC (Modelo-Vista-Controlador)

Es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización. Esta "separación de preocupaciones" proporciona una mejor división del trabajo y una mejora de mantenimiento. Algunos otros patrones de diseño se basan en MVC, como MVVM (Modelo-Vista-modelo de vista), MVP (Modelo-Vista-Presentador) y MVW (Modelo-Vista-Whatever) (*MVC - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN*, 2022).

2.4. Bases de Datos

Las bases de datos son simplemente una forma estructurada y sistemática de almacenar, acceder, analizar, transformar, actualizar y mover información (a otras bases de datos) (Dowsett, 2022).

Así pues, la relevancia de las bases de datos en el desarrollo de sistemas informáticos es fundamental, ya que la mayoría de los sistemas informáticos dependen de ellas para almacenar y administrar grandes cantidades de información. Las bases de datos, permiten a los desarrolladores de software diseñar sistemas más robustos, escalables y eficientes, al mismo tiempo que proporcionan una estructura organizada para acceder y administrar la información. Además, las bases de datos son esenciales para la toma de decisiones y el análisis de datos en los sistemas informáticos, lo que los hace imprescindibles en el mundo de la tecnología de la información.

2.4.1 Bases de Datos Relacionales

Una base de datos relacional organiza los datos en filas y columnas, que en conjunto forman una tabla. Los datos normalmente se estructuran en varias tablas, que se pueden unir a través de una clave principal o una clave externa. Estos identificadores únicos demuestran las diferentes relaciones que existen entre las tablas, y estas relaciones generalmente se ilustran a través de diferentes tipos de modelos de datos. Los analistas utilizan consultas SQL para combinar diferentes puntos de datos y resumir el rendimiento empresarial, lo que permite a las organizaciones obtener información, optimizar los flujos de trabajo e identificar nuevas oportunidades (What is a relational database? | IBM, 2021).

2.5. Front-end

Según García (2021) el frontend es la parte del desarrollo web que se dedica a la parte frontal de un sitio web, en pocas palabras, del diseño de un sitio web, desde la estructura del sitio hasta los estilos como colores, fondos, tamaños hasta llegar a las animaciones y efectos. Es esa parte de la página con la que interactúan los usuarios de la misma, es todo el código que se ejecuta en el navegador de un usuario, al que se le denomina una aplicación cliente, es decir, todo lo que el visitante ve y experimenta de forma directa.

2.6. Interfaz de Usuario (UI)

En pocas palabras, una interfaz de usuario es el punto de interacción y comunicación humano-computadora en un dispositivo, página web o aplicación. Esto puede incluir pantallas de visualización, teclados, un mouse y la apariencia de un escritorio. Las interfaces de usuario permiten a los usuarios controlar eficazmente la computadora o el dispositivo con el que están interactuando. Una interfaz de usuario exitosa debe ser intuitiva, eficiente y fácil de usar (Hannah, 2023).

2.7. Experiencia de Usuario (UX)

La experiencia del usuario según Stevens (2023) se refiere a cualquier interacción que un usuario tiene con un producto o servicio. El diseño de UX considera todos y cada uno de los elementos que dan forma a esta experiencia, cómo hace sentir al usuario y qué tan fácil es para el usuario realizar las tareas deseadas. Esto podría ser cualquier cosa, desde cómo se siente un

producto físico en la mano, hasta qué tan sencillo es el proceso de pago al comprar algo en línea. El objetivo del diseño de UX es crear experiencias placenteras fáciles, eficientes, relevantes y completas para el usuario.

2.8. Diseño de Interfaz de Usuario

El diseño UI, también conocido como diseño de interfaz de usuario, se refiere al diseño estético de todos los elementos visuales de la interfaz de usuario de un producto digital; es decir, la presentación del producto y la interactividad (Hannah, 2023).

2.9. Definición de Términos Básicos

2.9.1. JavaScript

JavaScript es un lenguaje de programación que se usa con mayor frecuencia para scripts dinámicos del lado del cliente en páginas web, pero también se usa a menudo en el lado del servidor, usando un tiempo de ejecución como Node.js. JavaScript no debe confundirse con el lenguaje de programación Java. Aunque "Java" y "JavaScript" son marcas comerciales (o marcas comerciales registradas) de Oracle en los EE. UU. y otros países, los dos lenguajes de programación son significativamente diferentes en su sintaxis, semántica y casos de uso. JavaScript se usa principalmente en el navegador, lo que permite a los desarrolladores manipular el contenido de la página web a través del DOM, manipular datos con AJAX e IndexedDB, dibujar gráficos con lienzo, interactuar con el dispositivo que ejecuta el navegador a través de varias API y más. JavaScript es uno de los lenguajes más utilizados en el mundo, debido al reciente

crecimiento y mejora del rendimiento de las API disponibles en los navegadores (JavaScript - MDN Web Docs Glossary: Definitions of Web-related terms | MDN, 2023).

2.9.2. Vue

Vue es un marco de trabajo (framework) de JavaScript para construir interfaces de usuario. Se basa en HTML, CSS y JavaScript estándar y proporciona un modelo de programación declarativo y basado en componentes que lo ayuda a desarrollar interfaces de usuario de manera eficiente, ya sean simples o complejas (Introduction | Vue.js, s. f.).

2.9.3. Vuetify

Vuetify es un marco de trabajo (framework) de javascript, que contiene una colección de componentes prefabricados combinados con funciones potentes como temas dinámicos, valores predeterminados globales, diseños de aplicaciones y más. Su objetivo es proporcionar a los desarrolladores todas las herramientas necesarias para crear experiencias de usuario ricas y atractivas (Vuetify — A Vue Component Framework, s. f.).

2.9.4. PHP

PHP es un lenguaje de secuencias de comandos de propósito general de código abierto ampliamente utilizado que es especialmente adecuado para el desarrollo web y se puede incrustar en HTML (PHP: What is PHP? - Manual, s. f.).

2.9.5. Laravel

Laravel es un marco de trabajo (framework) PHP gratuito y de código abierto que proporciona un conjunto de herramientas y recursos para crear aplicaciones PHP modernas. Con un ecosistema completo que aprovecha sus funciones integradas y una variedad de paquetes y extensiones compatibles. Laravel proporciona poderosas herramientas de base de datos que incluyen un ORM (Object Relational Mapper) llamado Eloquent y mecanismos integrados para crear migraciones de bases de datos y seeders. Con la herramienta de línea de comandos Artisan, los desarrolladores pueden iniciar nuevos modelos, controladores y otros componentes de la aplicación, lo que acelera el desarrollo general de la aplicación (Heidi, 2021).

2.9.6. PostgreSQL

PostgreSQL es un sistema de base de datos altamente estable y de código abierto que brinda soporte a diferentes funciones de SQL, como claves externas, subconsultas, disparadores y diferentes tipos y funciones definidos por el usuario. Aumenta aún más el lenguaje SQL y ofrece varias características que escalan y reservan cargas de trabajo de datos meticulosamente. Se utiliza principalmente para almacenar datos para muchas aplicaciones móviles, web, geoespaciales y de análisis (Ravoof, 2023).

2.9.7. API

Según Jin et al. (2018), una API (interfaz de programación de aplicaciones) se define como "una interfaz entre dos sistemas de software que les permite comunicarse entre sí". Ellos continúan explicando que las API permiten que diferentes sistemas de software interactúen e intercambien

información entre sí, sin necesidad de que los sistemas subyacentes comprendan los detalles de implementación de los demás. Los autores enfatizan la importancia de diseñar API teniendo en cuenta las necesidades de los desarrolladores y brindan orientación práctica para crear API que sean fáciles de usar y comprender.

2.9.8. API Rest

Jin et al. (2018), definen una API RESTful (Representational State Transfer) como una API que se adhiere a un conjunto de restricciones arquitectónicas, incluido el uso de métodos HTTP (como GET, POST, PUT y DELETE) para realizar operaciones en recursos y usar URI (identificadores uniformes de recursos) para identificar recursos. Además, las API RESTful permiten a los desarrolladores crear APIs escalables, flexibles y mantenibles, y se utilizan ampliamente en el desarrollo web moderno.

2.9.9. Aplicación de una sola página (SPA)

Combinando muchas tecnologías, SPA separa una aplicación web en su frontend y backend y organiza la comunicación entre ellos. La parte del frontend crea una vista externa basada en los datos recibidos del backend y muestra el resultado en el navegador, mientras que la parte del backend maneja la lógica, maneja las solicitudes e interactúa con la base de datos. Generalmente, las partes cliente y servidor de una aplicación se comunican entre sí a través de una interfaz de programación de aplicaciones que ha sido creada expresamente para este propósito (Kornienko et al., 2021).

CAPÍTULO 3

ANÁLISIS DEL SISTEMA

En el presente capítulo, se abordará el análisis del sistema, un proceso esencial para comprender y definir sus características fundamentales. Este estudio comprenderá tanto los requisitos funcionales como los no funcionales, proporcionando una visión holística de las capacidades y restricciones del sistema en cuestión. Además, se explorarán en detalle las historias de usuario, una herramienta narrativa que captura las necesidades y expectativas de los usuarios finales, y se analizarán los diversos actores que interactúan con el sistema, identificando sus roles y responsabilidades. Este enfoque integral permitirá una comprensión profunda y sistemática del entorno en el que operará el sistema, sentando las bases para su diseño, desarrollo y posterior implementación de manera efectiva y alineada con las necesidades del usuario y los objetivos del proyecto.

3.1. Requerimientos del Sistema

El objetivo principal de la especificación de requisitos del sistema es servir como medio de comunicación entre clientes, usuarios, ingenieros de requisitos y desarrolladores. La especificación de requisitos del sistema debe incluir tanto las necesidades del cliente como las del usuario (también conocidas como necesidades comerciales, necesidades del usuario, necesidades

del cliente, etc.) y los requisitos que debe cumplir el sistema de software que se está desarrollando para satisfacer esas necesidades. A continuación se delimitan los requisitos principales del sistema.

3.1.1. Actores

Un actor es un usuario del sistema.. Un actor utiliza un caso de uso para ejecutar una porción de trabajo de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define el rol en el sistema y el alcance de su acción. Para el desarrollo de la aplicación distinguimos 2 actores: administrador y usuario prestatario.

- **Administrador**: Tiene acceso completo a todas las funciones del sistema. Está a cargo del seguimiento y gestión de las solicitudes de partituras, además de registrar y supervisar los préstamos de partituras. Se pueden registrar tanto usuarios administradores como prestatarios.

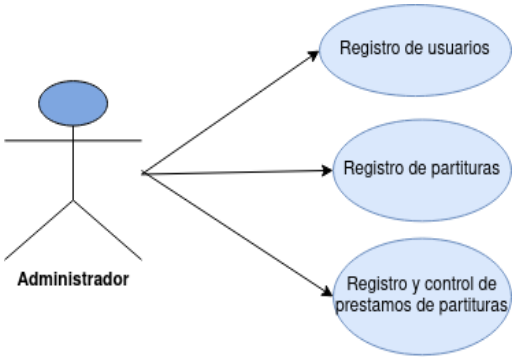

Figura 2: Usuario Administrador.

- **Usuario prestatario (Autenticado):** Tienen los permisos para solicitar préstamos del sistema de partituras.

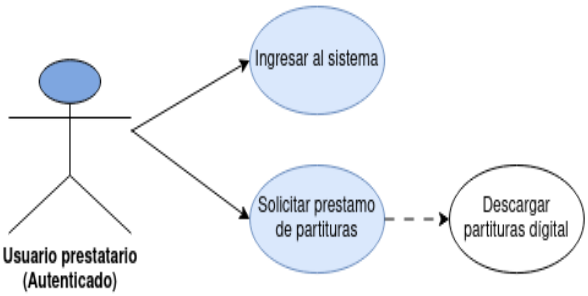

Figura 3: Usuario Prestatario (Autenticado).

- **Usuario prestatario (No autenticado):** Tiene la capacidad de visualizar el historial o listado de partituras en el sistema. Si lo requiere puede registrarse en el sistema.

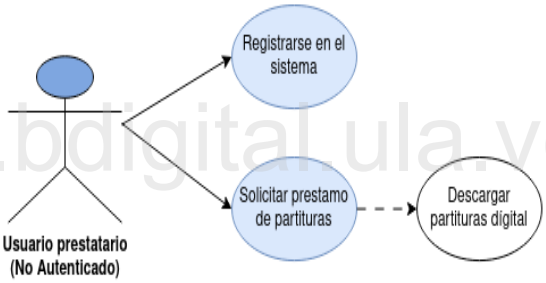

Figura 4: Usuario Prestatario (No autenticado).

3.1.2. Análisis de Requisitos

3.1.1. Identificación y Especificación de Requisitos Funcionales

El Sistema de Gestión Administrativa para Mejorar la Eficiencia en el Registro y Préstamo de Partituras tiene como objetivo principal optimizar los procesos administrativos relacionados

con el registro y préstamo de partituras en instituciones musicales. A continuación se detallan los requisitos funcionales identificados:

3.1.1.1. Gestión de Usuarios

- El sistema debe permitir la creación de cuentas de usuario con diferentes roles, como administrador, prestatario y usuario estándar.
- Los usuarios deben poder iniciar sesión utilizando credenciales únicas.
- Los administradores tienen la capacidad de gestionar y asignar roles a los usuarios.

3.1.1.2. Registro y Catalogación de Partituras

- El sistema debe posibilitar el registro de nuevas partituras, incluyendo información como título, autor, género y ubicación.
- La catalogación debe permitir la asociación de partituras a categorías específicas para facilitar búsquedas.
- El registro de una partitura debe permitir anexar una imagen de la pieza.

3.1.1.3. Préstamo y Devolución de Partituras

- Usuarios autorizados pueden solicitar el préstamo de partituras disponibles.
- El sistema debe mantener un registro del historial de préstamos, incluyendo fechas de préstamo y devolución.
- Notificaciones automáticas deben ser enviadas para recordar la devolución de partituras vencidas.

3.1.1.4. Carga y Descarga de Partituras

- El sistema debe permitir la carga y descarga de imagenes para las partituras registradas

3.2. Análisis de Requisitos no Funcionales

3.2.1. Interfaz de Usuario

- La interfaz de usuario debe ser intuitiva y fácil de usar para personas con diversos niveles de habilidad tecnológica.
- El sistema debe ser compatible con navegadores web modernos, como Chrome, Firefox o Safari.

3.2.2. Rendimiento

- El tiempo de respuesta para las operaciones críticas, como la búsqueda de partituras, no debe exceder los 2 segundos.
- La aplicación debe ser escalable para manejar un crecimiento futuro en la base de datos de partituras.

3.2.3. Seguridad

- El acceso a funciones administrativas debe estar restringido y protegido mediante roles y permisos.
- La información del usuario y las transacciones deben ser cifradas para garantizar la confidencialidad.

3.3. Historias de Usuario

Las historias de usuarios son una técnica de comunicación que combina los beneficios de la comunicación verbal y escrita cuando se utilizan en el contexto de la ingeniería de requisitos ágil. Resume una característica del software desde la perspectiva del usuario en una o dos oraciones usando el lenguaje que el usuario usaría. Lo que se destaca son las necesidades o problemas que abordará lo que se está desarrollando.. (Menzinsky et al., 2018). Las historias de usuario recopiladas fueron las siguientes:

1. **Registro de partitura**

Como administrador del sistema quiero poder registrar una nueva partitura en el sistema, para tener un registro organizado de todas las partituras disponibles en la fundación.

2. **Registro de autor**

Como administrador del sistema quiero poder registrar un nuevo autor de partitura para tener un registro organizado de los autores de partituras disponibles.

3. **Registro de géneros musicales**

Como administrador del sistema quiero poder registrar un nuevo género musical para tener un registro organizado de los géneros musicales disponibles.

4. **Registro de ubicaciones**

Como administrador del sistema quiero poder registrar una nueva ubicación para tener un registro organizado de los archivadores y/o gavetas.

5. **Registro de archivadores**

Como administrador del sistema quiero poder registrar una nuevo archivador para tener un registro organizado de los archivadores

6. **Registro de gavetas**

Como administrador del sistema quiero poder registrar una nueva gaveta para tener un registro organizado de los archivadores.

7. **Registro de usuario prestatario**

Como administrador del sistema quiero poder registrar usuarios para que puedan solicitar préstamos de partituras.

8. **Registro de usuarios administrador**

Como administrador del sistema quiero poder registrar usuarios administradores

9. **Descargar partitura digital**

Como Prestatario quiero poder descargar una partitura digital.

10. **Préstamo de partitura físico**

Como administrador del sistema quiero poder registrar un préstamo de partitura asociado a un prestatario

11. **Registrarse como prestatario**

Como prestatario quiero registrarme en el sistema para solicitar préstamos de partituras.

A continuación se presentan las historias de usuario más resaltantes del sistema.

Historia de usuario HU 001
Nombre de la historia de usuario: **Registro de partitura**
Usuario: Administrador del sistema
Descripción: Como administrador del sistema quiero poder registrar una nueva partitura en el sistema, para tener un registro organizado de todas las partituras disponibles en la fundación.
Validación: El usuario debe tener acceso al formulario de partitura. El usuario debe poder ingresar los siguientes campos: Titulo de la partitura(Campo de texto. Obligatorio) Autor de la partitura (Campo de selección. Obligatorio) Género musical al que pertenece(Campo de selección. Opcional) Ubicación física del documento(Opcional) Stock disponible (Campo de tipo numérico. Es opcional, pero se establecerá en 0 si no se proporciona)

Figura 5: Historia de usuario HU 001: Registro de Partituras

Historia de usuario HU 007
Nombre de la historia de usuario: **Registro de usuarios prestatario**
Usuario: Administrador del sistema
Descripción: Como administrador del sistema quiero poder registrar usuarios para que puedan solicitar prestamos de partituras.
Validación: El usuario debe tener acceso al formulario de registro de prestador. El usuario debe poder ingresar los siguientes campos: Nombre(Campo de texto. Obligatorio) Teléfono (Campo numérico. Obligatorio) Correo electrónico (Campo de texto. Obligatorio)

Figura 6: Historia de usuario HU 007: Registro de usuario prestatario.

Historia de usuario HU 008
Nombre de la historia de usuario: **Registro de usuario administrador**
Usuario: Administrador del sistema
Descripción: Como administrador del sistema quiero poder registrar usuarios administradores.
Validación: El usuario debe tener acceso al formulario de registro de administrador. El usuario debe poder ingresar los siguientes campos: Nombre(Campo de texto. Obligatorio) usuario (Campo de texto. Obligatorio) Contraseña (Campo de texto. Obligatorio)

Figura 7: Historia de usuario HU 008: Registro de usuario administrador.

Historia de usuario HU 009
Nombre de la historia de usuario: **Descargar partitura digital**
Usuario: Prestatario
Descripción: Como Prestatario quiero poder descargar una partitura digital.
Validación: Se debe tener un usuario registrado en el sistema. El usuario debe tener acceso al formulario de solicitud de préstamos. El usuario debe poder ingresar los siguientes campos: Partitura(Campo de selección. Obligatorio) Fecha del préstamo (Campo de tipo fecha. Obligatorio) Fecha de entrega (Campo de tipo fecha. Obligatorio) Cantidad (Campo de tipo numérico)

Figura 8: Historia de usuario HU 009: Descargar partitura digital.

Historia de usuario HU 010
Nombre de la historia de usuario: **Préstamo de partitura físico**
Usuario: Administrador del sistema
Descripción: Como administrador del sistema puedo registrar un préstamo de partitura asociado a un prestatario.
Validación: El usuario prestatario debe tener un usuario dentro del sistema. El usuario debe tener acceso al formulario de solicitud de préstamos. El usuario administrador puede aprobar préstamos de partituras. El usuario debe poder ingresar los siguientes campos: Partitura(Campo de selección. Obligatorio) Fecha del préstamo (Campo de tipo fecha. Obligatorio) Fecha de entrega (Campo de tipo fecha. Obligatorio) Cantidad (Campo de tipo numérico)

Figura 9: Historia de usuario HU 010: Préstamo de partitura físico.

CAPÍTULO 4

DISEÑO DEL SISTEMA

4.1. Arquitectura General del Sistema (Cliente - Servidor)

En el campo de la tecnología de la información, cliente-servidor representa un modelo de arquitectura de sistema que comprende dos componentes: sistemas cliente y sistemas servidores. Estos componentes se comunican entre sí a través de una red informática. Una aplicación cliente-servidor pertenece a la categoría de sistemas distribuidos y abarca tanto software de cliente como de servidor. Este tipo de aplicación ofrece un método mejorado para distribuir cargas de trabajo. El proceso del cliente establece constantemente una conexión con el servidor, mientras que el proceso del servidor permanece abierto a solicitudes de cualquier cliente. Un cliente se refiere a un dispositivo de hardware informático equipado con software que accede a los servicios puestos a disposición por un servidor. Por otro lado, un servidor es una computadora en la que se ejecuta un software dedicado que brinda servicios para cumplir con los requisitos de otras máquinas (Kumar, 2019).

La arquitectura cliente-servidor se clasifica en cuatro tipos: arquitectura de un nivel, arquitectura de dos niveles, arquitectura de tres niveles y arquitectura de N niveles (Kumar, 2019). Una aplicación de un nivel, también conocida como aplicación independiente, integra todas las capas, incluidas las de presentación, empresarial y de acceso a datos, en un único paquete de

software. Ejemplos de aplicaciones de un nivel incluyen reproductores MP3 y MS Office, que abarcan los tres niveles.

En una arquitectura de dos niveles, la aplicación se divide en aplicación cliente (nivel de cliente) y la base de datos (nivel de datos). Según Kumar (2019) el sistema cliente gestiona las capas de presentación y aplicación, mientras que el sistema servidor gestiona la capa de base de datos, por lo que se la conoce comúnmente como una aplicación cliente-servidor basada en escritorio. La comunicación se produce entre el cliente y el servidor, donde el dispositivo del cliente envía solicitudes y el servidor procesa y devuelve los datos solicitados.

La investigación de Kumar (2019) señala que la arquitectura de tres niveles, o arquitectura de aplicaciones basada en web, divide la aplicación en tres componentes: la capa de presentación (nivel de cliente), la capa de aplicación (nivel empresarial) y la capa de base de datos (nivel de datos). El sistema cliente supervisa la capa de presentación, el servidor de aplicaciones administra la capa de aplicación y el sistema servidor de bases de datos maneja la capa de base de datos.

Una capa adicional es la aplicación de N niveles, que distribuye la aplicación de manera similar a la arquitectura de tres niveles pero aumenta la cantidad de servidores de aplicaciones. Estos servidores están representados en niveles individuales para distribuir la lógica empresarial, asegurando una estructura lógica distribuida (Kumar, 2019).

Para el caso de la fundación Promúsica se eligió una arquitectura de dos niveles lo que permite gozar de las siguientes ventajas:

- **Seguridad:** el almacenamiento centralizado de datos en el servidor proporciona un control mejorado y un mayor nivel de seguridad en comparación con la dispersión de datos entre

numerosas máquinas cliente. Esto elimina la necesidad de implementar medidas de seguridad específicas en cada máquina cliente, contribuyendo a un entorno más seguro.

- **Acceso centralizado a los datos:** con la mayoría de los datos almacenados centralmente en el servidor, realizar actualizaciones de los datos se vuelve mucho más sencillo. Este enfoque simplificado facilita la gestión eficiente de los datos y garantiza un repositorio centralizado para un fácil acceso y modificación.

- **Facilidad de mantenimiento:** la arquitectura de dos niveles simplifica las actividades de mantenimiento ya que el cliente no está informado sobre las complejidades del servidor. En consecuencia, las tareas de mantenimiento del servidor, como reparaciones y actualizaciones, no afectan la funcionalidad del cliente, lo que lo convierte en un estilo arquitectónico sencillo de administrar.

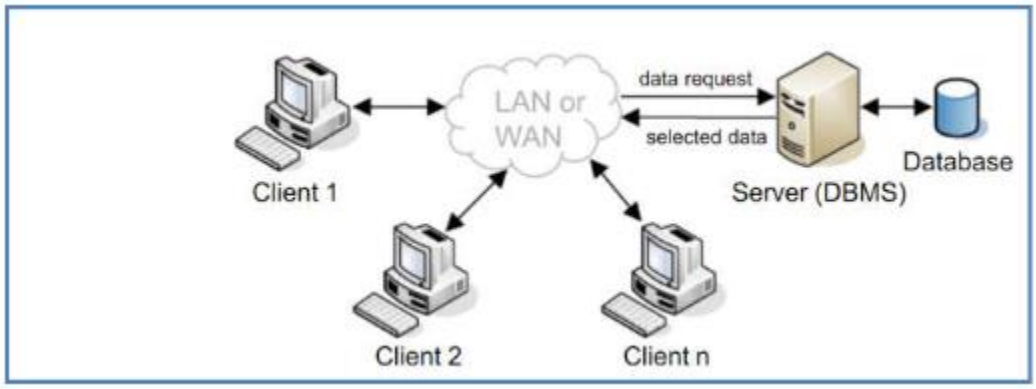

Figura 10. Arquitectura General Cliente-Servidor (Kumar, 2019)

Enfocando el diseño para el cliente a continuación se describe la arquitectura utilizada para desarrollar la aplicación web.

4.2. Aplicación de una sola página (SPA)

Una aplicación de una sola página se refiere a un sitio web que carga un solo documento y renueva la página actual al recibir nuevos datos del servidor web, en lugar de cargar las páginas individualmente desde cero. Esto implica que el contenido de la página se actualiza de manera instantánea y en tiempo real, adaptándose a las interacciones del usuario mediante transiciones ágiles y sin necesidad de recargar la página (Adobe, 2023).

Aunque el usuario interactúa con una única página web, la estructura de una Aplicación de Página Única (SPA, por sus siglas en inglés) implica la presencia de componentes de representación tanto en el lado del cliente como en el lado del servidor. Cuando un usuario accede inicialmente a una SPA, se le envía al navegador un archivo HTML que contiene los recursos necesarios. A partir de ese punto, se utiliza una Interfaz de Programación de Aplicaciones (API) para facilitar el intercambio de información del usuario y contenido actualizado entre el cliente y la SPA en tiempo real. Esta arquitectura también posibilita el desarrollo independiente del back-end y el front-end, con la capacidad de reutilizar código en ambos casos (Adobe, 2023).

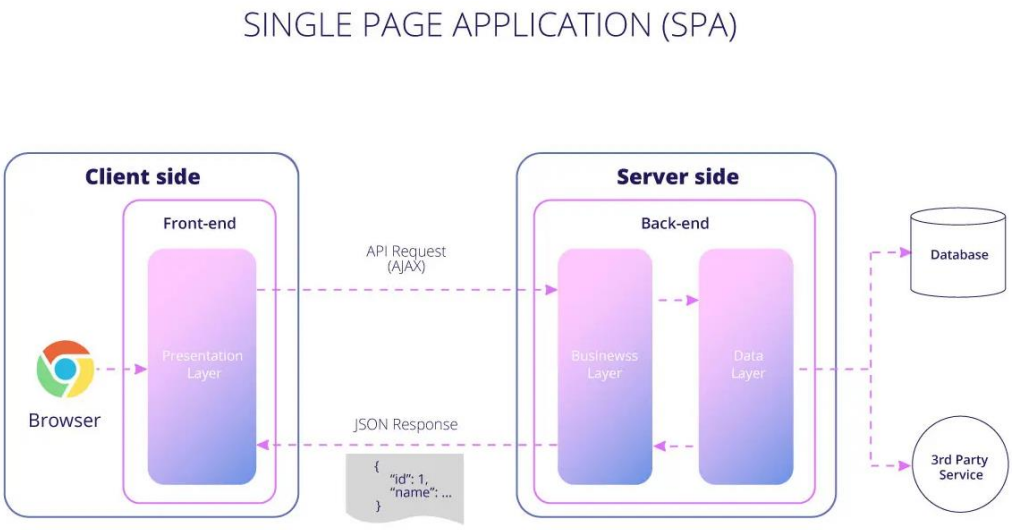

Figura 11. Arquitectura de una SPA. Fuente: https://mobidev.biz/

4.2.1. Pros de las aplicaciones de una sola página

Los beneficios clave de las aplicaciones de una sola página engloban:

- **Mejora de velocidad y capacidad de respuesta:** La eficiencia de carga se optimiza, ya que las SPAs solo cargan la información necesaria según la interacción del usuario, evitando la carga completa de una página desde cero.
- **Mayor estabilidad:** Se logra mediante el uso de funciones de almacenamiento en caché y una reducción significativa del consumo de ancho de banda, permitiendo que las páginas sean accesibles incluso en condiciones de conexión a Internet deficientes.
- **Experiencia de usuario mejorada:** La rápida carga de páginas y su apariencia interactiva fomentan una mayor participación del usuario, proporcionando una experiencia enriquecida al interactuar con el contenido.

- **Desarrollo ágil:** Las SPAs aprovechan diversas API, permitiendo a los desarrolladores trabajar de forma independiente en el front-end, back-end y conexiones, facilitando un desarrollo más rápido e iterativo.
- **Facilidad de depuración:** La modularidad de las bases de código simplifica las pruebas y correcciones, permitiendo que los desarrolladores trabajen en distintas secciones simultáneamente.
- **Compatibilidad multiplataforma:** Al utilizar una única base de código, las SPAs pueden diseñarse para ejecutarse en cualquier plataforma o navegador, ofreciendo una experiencia fluida al cambiar de dispositivo.
- **Adaptabilidad a dispositivos móviles:** La reutilización de código posibilita la creación de aplicaciones móviles y páginas responsivas que mantienen su funcionalidad tanto en navegadores como en aplicaciones de teléfonos, brindando un sólido soporte a los usuarios de dispositivos móviles.

4.3. Esquema de página (Wireframe)

Un wireframe consiste en un diagrama visual que delinea la estructura básica de un proyecto o componente tecnológico. También llamado esquema de página o plano de pantalla, este elemento visual ilustra la interconexión y disposición de los elementos dentro del proyecto, proporcionando una representación esencial de su organización (*Wireframe: qué es, cómo hacerlo y ejemplos | Miro*, s. f.).

4.3.1. Wireframe de la página principal

El siguiente wireframe representa la vista principal de la aplicación:

Figura 12. Wireframe de la página principal.

y consta de los siguientes elementos:

- **Imagen:** Fotografía del usuario o avatar.
- **Nombre:** Nombre completo de la persona actualmente autenticada.
- **Link:** Links de navegación de la aplicación.
- **Email:** Correo electrónico del usuario actualmente autenticado.
- **Área principal de visualización:** Muestra información tabulada en un tabla con buscador.

CAPÍTULO 5

DESARROLLO DE LA APLICACIÓN

Este capítulo aborda en detalle el proceso de desarrollo de la aplicación web diseñada para abordar los objetivos y requisitos establecidos en el Capítulo 3. Se describen las tecnologías utilizadas, la estructura del código, y se proporciona una visión general de las funcionalidades implementadas.

5.1. Tecnologías Utilizadas

- **Lenguaje de Programación:** El desarrollo de la aplicación web se llevó a cabo utilizando JavaScript, un lenguaje versátil y ampliamente utilizado en el ámbito del desarrollo web. JavaScript permite la creación de interfaces interactivas y dinámicas, fundamentales para el enfoque centrado en el usuario de la aplicación.
- **Framework de Front-End:** Vue.js fue seleccionado como el framework de front-end para el desarrollo de la interfaz de usuario. Vue.js es conocido por su simplicidad y flexibilidad, facilitando la construcción de componentes reutilizables y la gestión eficiente del estado de la aplicación. Su capacidad para integrarse fácilmente con otras bibliotecas y su curva de aprendizaje gradual lo convierten en una elección ideal para proyectos de diversos tamaños.

- **Editor de Texto:** Visual Studio Code (VSCode) se utilizó como el editor de texto principal durante el desarrollo. Su amplia gama de extensiones, integración con Git, y características avanzadas de depuración proporcionan un entorno de desarrollo eficiente y productivo. La interfaz intuitiva y la comunidad activa de desarrolladores respaldan su elección para maximizar la productividad.
- **Herramienta para el Control de Versiones:** Git se empleó como la herramienta principal para el control de versiones. La capacidad de Git para gestionar cambios de manera eficiente, ramificar el código, y colaborar de forma distribuida fue esencial para mantener un historial claro y asegurar la integridad del código. Git proporcionó la base para la colaboración en equipo y facilitó la gestión de versiones a lo largo del ciclo de vida del desarrollo.
- **Repositorio de Código:** El código fuente se almacenó en un repositorio Git hospedado en GitLab. GitLab ofreció una plataforma integral que no solo alojó el código, sino que también facilitó la colaboración, el seguimiento de problemas (issues), y la integración continua. La elección de GitLab se alinea con el enfoque de desarrollo colaborativo y la necesidad de una plataforma integral para la gestión del ciclo de vida del desarrollo.
- **State Management con Vuex:** Para gestionar de manera eficiente el estado de la aplicación, se implementó Vuex, la biblioteca oficial de gestión de estado para aplicaciones Vue.js. Vuex facilita la organización y el flujo de datos en la aplicación, proporcionando un almacén centralizado y patrones claros para manejar la lógica de estado compleja.
- **Enrutamiento con Vue Router:** Para la navegación dentro de la aplicación, se integró Vue Router. Esta herramienta permite la gestión de las rutas en una SPA, facilitando la

transición entre las diferentes vistas de manera suave y eficiente. Vue Router se configuró para gestionar la navegación y garantizar una experiencia de usuario coherente.

- **Manejo de Peticiones HTTP con Axios:** La biblioteca Axios se incorporó para la gestión de solicitudes HTTP dentro de la aplicación. Axios simplifica la realización de peticiones asíncronas al servidor, permitiendo la obtención y envío de datos de manera eficiente. Esta elección contribuye a una comunicación fluida entre el front-end y el back-end de la aplicación.
- **Componentes visuales con Vuetify:** Para mejorar la estética y la usabilidad de la interfaz, se incorporó Vuetify como un framework de diseño de componentes basado en Material Design. Vuetify proporciona una amplia variedad de componentes preestablecidos, como botones, barras de progreso y tarjetas, que se integran fácilmente con Vue.js, acelerando el desarrollo y asegurando una interfaz coherente y visualmente atractiva.

5.1.1 Framework

Habiendo obtenido una comprensión más profunda del funcionamiento y las ventajas de las aplicaciones de una sola página (SPA) vistas en el capítulo anterior, se puede hacer mención de los frameworks más importantes para desarrollar este tipo de aplicaciones.

- Angular
- React
- Vue.js
- Aurelia
- Backbone.js

- Ember.js
- Knockout.js
- Meteor.js
- Polymer.js

Para el desarrollo del presente proyecto se ha elegido Vue.js como framework de desarrollo. La elección de Vue.js sobre los demás se basa en una combinación de factores que hacen que Vue.js haya sido la opción preferida para este proyecto. En primer lugar, Vue.js destaca por su simplicidad y facilidad de integración, lo que acelera el proceso de desarrollo. Su curva de aprendizaje es suave, lo que facilita la adopción rápida por parte del equipo de desarrollo.

Además, la flexibilidad de Vue.js permite adaptarse fácilmente a diferentes escalas de proyectos, desde aplicaciones pequeñas hasta proyectos más grandes y complejos. Esta versatilidad es esencial para ajustarse a las necesidades específicas del presente proyecto.

Un elemento crucial en la decisión de elegir Vue.js es la experiencia previa con este framework. La familiaridad con la sintaxis y las prácticas de desarrollo de Vue.js reduce los tiempos de entrenamiento y acelera el proceso de implementación.

5.2. Estructura del Código

La estructura del código se basa en las convenciones establecidas por Vue.js y se beneficia de la configuración inicial proporcionada por vue-cli, una herramienta oficial de Vue.js para la generación rápida de proyectos.

5.2.1. Estructura de carpetas y archivos

El proyecto sigue una estructura organizada para facilitar la navegación y el mantenimiento del código. A continuación, se presenta una descripción general de la estructura principal:

- **public/:** Contiene archivos estáticos como imágenes, iconos y la página HTML principal (index.html).
- **src/:** Aquí se encuentra la mayor parte del código fuente.
    - **assets/:** Almacena archivos que se importan en los componentes, como imágenes, estilos y fuentes.
    - **components/:** Contiene los componentes Vue reutilizables utilizados en la aplicación. Cada componente tiene su propia carpeta con archivos .vue, .js, y .scss si es necesario.
    - **views/:** Contiene las vistas de nivel superior que se corresponden con las rutas de la aplicación. Cada vista puede contener componentes específicos de esa vista.
    - **router/:** Incluye la configuración del enrutador Vue Router. El archivo index.js define las rutas y cómo se vinculan a las vistas.
    - **store/:** Contiene la configuración del almacenamiento Vuex. Se divide en módulos para organizar la lógica de estado de manera modular.
    - **plugins/:** Contiene archivos de configuración de plugins. En este caso, se incluirían archivos relacionados con la configuración de Vuetify y Axios.
    - **App.vue:** El componente raíz de la aplicación que envuelve toda la interfaz.

- **main.js:** Punto de entrada de la aplicación donde se importan y configuran Vue, Vuetify, Vuex, Vue Router, y otros elementos esenciales.

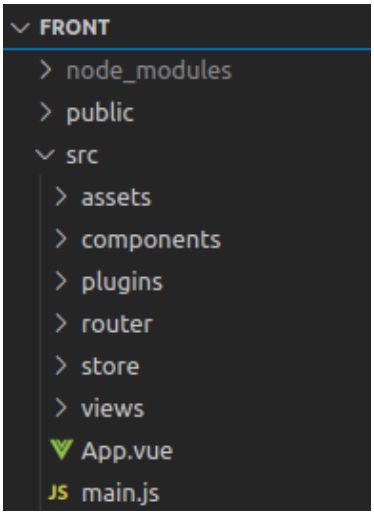

Figura 13. Estructura de carpetas y archivos.

5.3. Uso de Componentes y Vistas

Los componentes se utilizan para modularizar la interfaz de usuario, mientras que las vistas manejan la presentación general de las páginas. Vue Router se configura para asociar rutas a vistas específicas, facilitando la navegación dentro de la SPA.

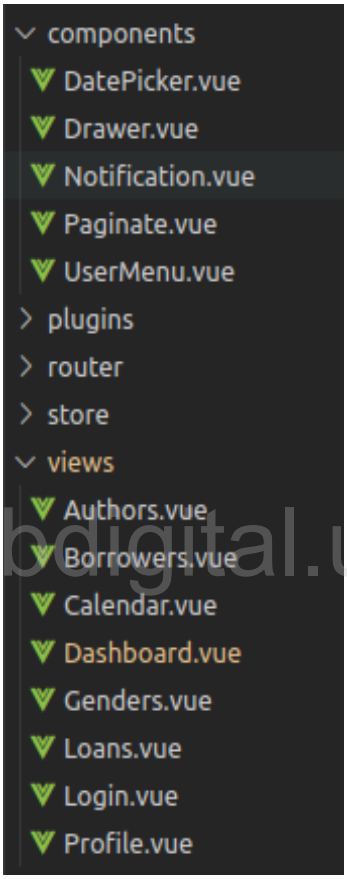

Figura 14. Componentes y vistas del sistema.

5.4. Integración con Vuex

Vuex se organiza en módulos para manejar el estado de manera modular y escalable. Axios en plugins/ se utiliza para realizar solicitudes HTTP, asegurando una comunicación eficiente con el backend.

Este código representa la configuración y la creación del almacenamiento principal Vuex de la aplicación. Vuex proporciona un patrón unidireccional para manejar datos en la aplicación de manera más estructurada y eficiente. Veamos cada parte del código:

```
import Vue from "vue";
import Vuex from "vuex";
import user from "./modules/user";
import musicSheet from "./modules/musicSheet";
import notifications from "./modules/notifications";
import authors from "./modules/authors";
```

Vue y Vuex se importan desde las bibliotecas correspondientes. Vue es necesario porque Vuex se instala como un complemento en Vue. Luego, se importan cuatro módulos específicos (user, musicSheet, notifications, authors) desde archivos separados. Cada módulo contiene su propio estado, mutaciones, acciones y getters.


```
Vue.use(Vuex);
```

Se utiliza Vue.use(Vuex) para instalar Vuex como un complemento de Vue. Esto permite que el objeto store de Vuex sea accesible desde todos los componentes de la aplicación.

```
export default new Vuex.Store({
        state: {},
        getters: {},
        mutations: {},
        actions: {},
        modules: {
          user,
          musicSheet,
          notifications,
          authors,
        },
```

```
});
```

Se exporta un nuevo objeto Vuex.Store. Este objeto define el estado global de la aplicación y cómo se puede modificar y acceder a través de acciones, mutaciones y getters.

state: Representa el estado global de la aplicación. En este caso, inicialmente está vacío ({}), pero puede contener propiedades y valores según sea necesario.

getters: Define funciones que permiten acceder y derivar valores del estado global. Estos son como métodos de sólo lectura para acceder al estado de una manera computada.

mutations: Define funciones que modifican el estado. Las mutaciones son síncronas y deben ser invocadas de manera directa.

actions: Define funciones que realizan operaciones asíncronas o manejan flujos más complejos. Las acciones pueden contener lógica asíncrona y llamar a mutaciones.

modules: Contiene módulos adicionales que dividen el almacenamiento en secciones más pequeñas y gestionables. Cada módulo (user, musicSheet, notifications, authors) se define en su propia lógica y se importa aquí.

5.4.1. Integración con Axios y Vuex

5.4.1.1. User Store

El siguiente código representa el módulo Vuex user que gestiona el estado relacionado con la autenticación de usuario. Aquí hay una explicación detallada de cada sección:

```
import router from "@/router";
import axios from "@/plugins/axios";
```

Se importa el enrutador (router) y Axios desde rutas y plugins respectivamente. Estos serán utilizados para la navegación y para realizar solicitudes HTTP. Exportando el objeto con la configuración de módulo.

```
export default {
 state: {
  user: {},
  auth: false,
 },
 getters: {
  user(state) {
   return state.user;
  },
  auth(state) {
   return state.auth;
  },
 },
 mutations: {
  SET_USER(state, user) {
   state.user = user || null;
  },
  SET_AUTH(state, auth) {
   state.auth = auth;
  },
 },
 actions: {
  async logout({ commit }) {
   try {
    let response = await axios.post("logout");
    if (response.status === 200) {
     localStorage.removeItem("user");
     await commit("SET_USER", {});
```

```
        await commit("SET_AUTH", false);
        await router.push("/");
      }
    } catch (error) {
      console.log(error);
    }
  },
  async login({ commit }, credentials) {
    try {
      await axios.get("sanctum/csrf-cookie");
      let response = await axios.post("login", credentials);
      if (response.status === 200) {
        localStorage.setItem("user", JSON.stringify(response.data.user));
        await commit("SET_USER", response.data.user);
        await commit("SET_AUTH", true);
        await router.push("dashboard");
      }
    } catch (error) {
      console.log(error);
    }
  },
  async getUser({ commit }) {
    let user = localStorage.getItem("user");
    if (user) {
      await commit("SET_USER", JSON.parse(user));
      await commit("SET_AUTH", true);
    } else {
      await commit("SET_USER", {});
      await commit("SET_AUTH", false);
    }
  },
 },
 modules: {},
};
```

- **state:** Contiene el estado del módulo. En este caso, se mantiene un objeto user que representa la información del usuario y un booleano auth que indica si el usuario está autenticado o no.
- **getters:** Definen funciones que permiten acceder al estado de manera computada. En este caso, hay getters para obtener el usuario y su estado de autenticación.
- **mutations:** Son funciones que modifican el estado de manera síncrona. Aquí, hay mutaciones para establecer el usuario y su estado de autenticación.
- **actions:** Son funciones que realizan operaciones asíncronas y pueden llamar a mutaciones. Las acciones son utilizadas para realizar operaciones como iniciar sesión, cerrar sesión y obtener información del usuario.
- **logout:** Realiza la operación de cerrar sesión. Hace una solicitud HTTP (axios.post("logout")) al servidor para invalidar la sesión, elimina la información del usuario del almacenamiento local, y realiza las mutaciones necesarias para actualizar el estado y redirige al usuario a la página principal.
- **login:** Realiza la operación de inicio de sesión. Antes de enviar las credenciales, realiza una solicitud para obtener un token CSRF, luego hace una solicitud de inicio de sesión al servidor. Si la respuesta es exitosa, guarda la información del usuario en el almacenamiento local, realiza las mutaciones necesarias y redirige al usuario al panel de control (dashboard).
- **getUser:** Obtiene la información del usuario del almacenamiento local. Si la información existe, actualiza el estado y establece la autenticación. Si no existe, se establece un usuario vacío y la autenticación en false.

5.4.1.2. MusicSheets Store

Este código representa el módulo Vuex que gestiona el estado y las operaciones relacionadas con las partituras musicales en la aplicación. A continuación hay un análisis de cada parte del código:

```
import axios from "@/plugins/axios";
```

Se importa la instancia de Axios configurada (axios) desde el archivo de plugins.

```
export default {
 namespaced: true,
 state: {
   items: [],
   currentPage: 1,
   lastPage: null,
   total: null,
   perPage: null,
 },
};
```


- **namespaced: true:** Indica que este módulo Vuex es "nombrado" o "espaciado", lo que significa que las mutaciones, acciones y getters dentro de este módulo se accederán utilizando el nombre del módulo como un espacio de nombres.

- **state:** Contiene el estado del módulo. En este caso, se mantiene un array de items que representan las partituras musicales, así como información sobre la paginación (currentPage, lastPage, total, perPage).

```
getters: {
  getTotalPages(state) {
    return state.lastPage;
  },
  getCurrentPage(state) {
    return state.currentPage;
  },
  getMusicSheets(state) {
    return state.items;
  },
  getItemsPerPage(state) {
    return state.perPage;
  },
  getItemIndex(state, item) {
    return state.items.indexOf(item);
  },
},
```


Los getters proporcionan formas de obtener información específica del estado, como el número total de páginas (getTotalPages), la página actual (getCurrentPage), las partituras musicales (getMusicSheets), la cantidad de elementos por página (getItemsPerPage), y el índice de un elemento en el array (getItemIndex).

```
mutations: {
  SET_MUSIC_SHEETS(state, data) {
    state.items = data?.music_sheets || [];
    state.currentPage = data?.meta.current_page || null;
    state.lastPage = data?.meta.last_page || null;
    state.perPage = data?.meta.per_page || null;
    state.total = data?.meta.total || null;
  },
  UPDATE_MUSIC_SHEET(state, data) {
```

```
    Object.assign(state.items[data.index], data.item);
  },
  ADD_MUSIC_SHEET(state, data) {
    state.items.push(data.item);
  },
  SET_CURRENT_PAGE(state, data) {
    state.currentPage = data;
  },
},
```

Las mutaciones son funciones que modifican el estado de manera síncrona. Aquí, hay mutaciones para:

- **SET_MUSIC_SHEETS:** Actualiza el estado con información de partituras musicales, paginación y total de elementos.
- **UPDATE_MUSIC_SHEET:** Actualiza una partitura musical existente en el array.
- **ADD_MUSIC_SHEET:** Agrega una nueva partitura musical al array.
- **SET_CURRENT_PAGE:** Establece la página actual.


```
    actions: {
      async getMusicSheets({ commit }, url) {
        try {
          let { data } = await axios.get(url);
          await commit("SET_MUSIC_SHEETS", data);
        } catch (error) {
          if (error.status === 401) {
            await commit("SET_MUSIC_SHEETS", null);
          }
        }
      },
```

```
  async saveMusicSheet({ commit }, data) {
    let response = await axios.post(data.url, data.form, {
      headers: { "Content-Type": "multipart/form-data" },
    });
    let mutation = {};
    if (data.isEdit) {
     mutation = { index: data.index, item: response.data.item };
      await commit("UPDATE_MUSIC_SHEET", mutation);
    } else {
      mutation = { item: response.data.item };
      await commit("ADD_MUSIC_SHEET", mutation);
    }
  },
  async setMusicSheets({ commit }, data) {
    await commit("SET_MUSIC_SHEETS", data);
  },
  async setCurrentPage({ commit }, data) {
    await commit("SET_CURRENT_PAGE", data);
  },
},
```


Aquí, hay acciones para:

- **getMusicSheets:** Realiza una solicitud HTTP para obtener información sobre las partituras musicales.
- **saveMusicSheet:** Guarda una nueva partitura musical o actualiza una existente haciendo una solicitud HTTP.
- **setMusicSheets:** Actualiza el estado con información de partituras musicales (utilizado probablemente cuando se necesita establecer el estado desde algún componente o contexto externo).
- **setCurrentPage:** Establece la página actual en el estado.

5.4.1.3. Notifications Store

Este módulo Vuex gestiona el estado y las operaciones relacionadas con notificaciones en la aplicación. Análisis de cada sección del código:

```
export default {
 namespaced: true,
 state: {
   notifications: [],
 },
 getters: {},
 mutations: {},
 actions: {},
};
```

- **namespaced: true:** Indica que este módulo Vuex es "nombrado" o "espaciado", lo que significa que las mutaciones, acciones y getters dentro de este módulo se accederán utilizando el nombre del módulo como un espacio de nombres.

- **state:** Contiene el estado del módulo. En este caso, se mantiene un array llamado notifications que representará las notificaciones en la aplicación.

```
mutations: {
  PUSH_NOTIFICATION(state, notification) {
    state.notifications.push({
      ...notification,
      id: Math.random().toString(36) + Date.now().toString(36).substring(2),
    });
  },
  CLEAR_NOTIFICATIONS(state) {
    state.notifications = [];
```

```
  },
},
```

- **PUSH_NOTIFICATION:** Mutación que agrega una nueva notificación al array de notifications. Cada notificación es un objeto que se compone de todas las propiedades de la notificación original (...notification) más una propiedad id que se genera de manera única utilizando Math.random() y la marca de tiempo actual.
- **CLEAR_NOTIFICATIONS:** Mutación que limpia el array de notifications, eliminando todas las notificaciones existentes.

```
    actions: {
      addNotification({ commit }, notification) {
        switch (notification?.type) {
          case "success":
            notification.bodyColor = "green darken-1";
            notification.dismissColor = "white";
            notification.textColor = "white";
            notification.icon = "mdi-bell";
            break;

          case "error":
            notification.bodyColor = "red darken-2";
            notification.dismissColor = "white";
            notification.textColor = "white";
            notification.icon = "mdi-alert-decagram";
            break;

          default:
            notification.bodyColor = "black";
            notification.dismissColor = "pink";
            notification.textColor = "white";
```

```
        notification.icon = "";
      }
      notification.show = true;

      commit("PUSH_NOTIFICATION", notification);
    },
    clearNotifications({ commit }) {
      commit("CLEAR_NOTIFICATIONS");
    },
  },
```

- **addNotification:** Acción que permite agregar una nueva notificación al estado. Antes de agregarla, ajusta el formato de la notificación según su tipo (success, error, u otro). Las propiedades como bodyColor, dismissColor, textColor, e icon se establecen según el tipo de notificación. Luego, se establece show en true para indicar que la notificación debe mostrarse y se realiza la mutación PUSH_NOTIFICATION para agregarla al array de notifications.
- **clearNotifications:** Acción que realiza la mutación CLEAR_NOTIFICATIONS para limpiar todas las notificaciones en el estado.

CAPÍTULO 6

CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

El presente trabajo de grado se enfocó en el desarrollo de la interfaz gráfica de un sistema de gestión administrativa para la Fundación Promúsica con el objetivo general de mejorar la eficiencia en el registro y préstamo de partituras, para esto se abordaron diversas metas específicas que guiaron el proceso de diseño, desarrollo e implementación de la solución propuesta.

El desarrollo y la implementación de la interfaz gráfica constituyen un avance significativo hacia la gestión efectiva en el registro y préstamo de partituras. La respuesta a la pregunta de investigación que guió este proyecto ha quedado evidenciada a lo largo de cada fase, demostrando que el diseño e implementación de una aplicación web puede tener un impacto sustancial en la eficacia de las operaciones de la fundación. La aplicación web desarrollada se erige como una herramienta fundamental que simplifica y agiliza los procesos administrativos asociados al manejo de partituras. La interfaz intuitiva y amigable no solo facilita la labor diaria del personal encargado, sino que también mejora la experiencia para los usuarios al hacer más accesible el proceso de préstamo y devolución de partituras. Esto se traduce directamente en una gestión más eficiente y transparente de los recursos musicales de la fundación.

El diseño de la aplicación se centró en la optimización de los procesos, asegurando que cada paso, desde el registro hasta el préstamo y la devolución, se realice de manera eficiente y sin

complicaciones innecesarias. La implementación de flujos de trabajo lógicos y la reducción de pasos redundantes han resultado en una significativa disminución del tiempo dedicado a tareas administrativas, permitiendo que el personal de la fundación pueda concentrarse más en sus actividades artísticas y educativas. La integración con una API existente ha permitido un acceso en línea a la información relevante, asegurando que la base de datos se mantenga actualizada y consistente. Esto no solo contribuye a la toma de decisiones, sino que también reduce la posibilidad de errores asociados a la desactualización de datos.

La aplicación web se convierte así en un canal dinámico que refleja de manera fiel el estado de las partituras disponibles y en préstamo. La implementación exitosa de la aplicación web no se limitó a la tecnología en sí, sino que también abarcó la capacitación del personal. El diseño centrado en el usuario facilitó el proceso de aprendizaje, permitiendo una transición suave y una adopción rápida por parte del equipo de la fundación. Esta capacitación personalizada aseguró que los usuarios se sintieran cómodos y competentes al utilizar la nueva herramienta, maximizando así sus beneficios.

6.2. Recomendaciones

A pesar de los éxitos alcanzados en el diseño e implementación de la aplicación web, es crucial reconocer y abordar las limitaciones inherentes al alcance y contexto del proyecto. Estas limitaciones, si bien no desmerecen los logros obtenidos, proporcionan un marco realista para entender las áreas en las que el sistema podría tener restricciones o necesidades futuras de mejora. El análisis de las necesidades de los usuarios se basó en la información proporcionada por la

Fundación Promúsica. Sin embargo, esta dependencia de la información suministrada puede resultar limitante, ya que la disponibilidad y la exhaustividad de la información pueden variar.

Las necesidades no expresadas o subestimadas podrían no haber sido completamente abordadas en el diseño de la interfaz, lo que podría generar posibles áreas de mejora en futuras iteraciones del sistema. Aunque se realizó un esfuerzo considerable para diseñar una interfaz gráfica que cubriera las necesidades identificadas en el análisis, es posible que algunas demandas específicas no hayan sido completamente satisfechas. Las preferencias individuales de los usuarios o requisitos no identificados en las etapas iniciales podrían surgir durante el uso cotidiano, destacando la importancia de mantener una comunicación constante con los usuarios y estar preparados para adaptaciones futuras. La implementación del sistema se limitó a las funciones y características específicas identificadas por la Fundación Promúsica. Esto podría resultar en una limitación en la escalabilidad del sistema para abordar nuevas necesidades que puedan surgir en el futuro. La adaptabilidad del sistema a cambios en los procesos internos de la fundación podría requerir actualizaciones y ajustes continuos.

A pesar de estas limitaciones, es crucial resaltar que el enfoque adoptado en este proyecto se centra en la flexibilidad y la capacidad de respuesta a las necesidades cambiantes. La retroalimentación continua de los usuarios, la supervisión proactiva de la funcionalidad y la disposición para realizar mejoras iterativas son elementos esenciales para mitigar estas limitaciones y garantizar la sostenibilidad y relevancia a largo plazo del sistema de gestión administrativa de la Fundación Promúsica.

Por otro lado con base en la experiencia obtenida durante el desarrollo del proyecto y las conclusiones alcanzadas, se presentan algunas recomendaciones para orientar futuras acciones y mejoras:

- Establecer mecanismos periódicos para evaluar y actualizar las necesidades de los usuarios. Esto permitirá mantener la interfaz gráfica alineada con las demandas cambiantes y garantizar su relevancia a lo largo del tiempo.
- Recopilar retroalimentación del usuario y realizar ajustes en la interfaz o en las funcionalidades según sea necesario para garantizar una experiencia óptima.
- Investigar y evaluar la viabilidad de integrar nuevas funcionalidades que puedan mejorar aún más la eficiencia operativa. Esto podría incluir herramientas de análisis de datos, informes personalizados o características específicas solicitadas por los usuarios.
- Mantener una colaboración estrecha y continua con la Fundación Promúsica. Establecer canales de comunicación efectivos para recibir retroalimentación, resolver problemas y estar al tanto de posibles cambios en las operaciones que puedan afectar el sistema.
- Establecer protocolos robustos de respaldo de datos y seguridad. Asegurar que la información crítica esté protegida contra pérdidas, accesos no autorizados y otros riesgos potenciales.
- Implementar estrategias sólidas de gestión de cambios al introducir actualizaciones o nuevas versiones del sistema. Informar y capacitar a los usuarios sobre los cambios para minimizar cualquier interrupción en sus procesos de trabajo.
- Desarrollar nuevas funcionalidades para la aplicación web, en función de las necesidades de la fundación y de los usuarios. Algunas ideas para nuevas funcionalidades incluyen un

módulo para la gestión de reservas de partituras. Un módulo para la gestión de eventos musicales. Un módulo para la gestión de instrumentos musicales.

Al seguir estas recomendaciones, se podrá fortalecer la efectividad y la longevidad del sistema de gestión administrativa desarrollado, manteniéndolo relevante y adaptado a las necesidades cambiantes de la Fundación Promúsica.

REFERENCIAS

Alarcón, V. F. (2006). Desarrollo de sistemas de información: una metodología basada en el modelado. Edicions UPC eBooks. https://dialnet.unirioja.es/servlet/libro?codigo=298995

An Effective Requirement Engineering Process Model for Software Development and Requirements Management. (2010). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/5656776/

Arias, F. (2012). El Proyecto De Investigación (6.a ed.). Caracas: EDITORIAL EPISTEME. ISBN: 980-07-8529-9.

Adobe. (19 de julio de 2023). Single-page applications (SPAs) — what they are and how they work. https://business.adobe.com/blog/basics/learn-the-benefits-of-single-page-apps-spa

Bass, L., Clements, P., y Kazman, R. (2012). Software Architecture in Practice. Addison-Wesley.

Britannica, T. Editors of Encyclopaedia (2021). client-server architecture. Encyclopedia Britannica. https://www.britannica.com/technology/client-server-architecture

Cruz, Y. E., Zamora, C., Paz, C., y Jorge, R. E. (2020). Adopción de tecnologías de gestión de procesos de negocio: una revisión sistemática. Ingeniare. Revista chilena de ingeniería, 28(1), 41-55. https://doi.org/10.4067/s0718-33052020000100041

Dowsett, C. (2022). What Is a Database? Built In. https://builtin.com/data-science/database

EcuRed. (2021). Modelo de prototipos - EcuRed. https://www.ecured.cu/Modelo_de_prototipos

Gavilán, C. M (2008). SIGB. Catálogos y gestión de Autoridades. Diseño y prestaciones de OPACs. http://eprints.rclis.org/13188/1/sigb.pdf

García, I. J. B. (2021, 30 marzo). Backend y Frontend, ¿Qué es y cómo funcionan en la programación? https://www.servnet.mx/blog/backend-y-frontend-partes-fundamentales-de-la-programacion-de-una-aplicacion-web

Gutiérrez, A. E. R. (2020). Sistema de gestión y digitalización bibliotecaria. http://repositorio.upea.bo/handle/123456789/96

Hannah, J. (2023, abril 24). What Is A User Interface & What Are The Key Elements? https://careerfoundry.com/en/blog/ui-design/what-is-a-user-interface/

Heidi, E. (2021). What is Laravel? DigitalOcean Community. https://www.digitalocean.com/community/tutorials/what-is-laravel

HostingPlus. (2021, 6 julio). Modelo de prototipos: ¿qué es y cuáles son sus etapas? | Blog | Hosting Plus Perú. Hosting Plus. https://www.hostingplus.pe/blog/modelo-de-prototipos-que-es-y-cuales-son-sus-etapas/

IBM Documentation. (2021). https://www.ibm.com/docs/es/aix/7.1?topic=systems-client-server

What is a relational database? | IBM. (2021). https://www.ibm.com/topics/relational-databases

Wireframe: qué es, cómo hacerlo y ejemplos | Miro. (s. f.). https://miro.com/. https://miro.com/es/wireframe/que-es-wireframe/

Introduction | Vue.js. (s. f.-b). https://vuejs.org/guide/introduction.html

JavaScript - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. (2023, 21 febrero). https://developer.mozilla.org/en-US/docs/Glossary/JavaScript

Jin, B., Sahni, S., y Shevat, A. (2018). Designing Web APIs: Building APIs That Developers Love. “O’Reilly Media, Inc.”

Kumar, S. (2019). A review on client-server based applications and research opportunity. International Journal of Recent Scientific Research, 10(7), 33857-3386.

Langer, A. M. (2018). Information technology and organizational learning: Managing behavioral change in the digital age. (3.a ed.). CRC Press Taylor & Francis Group. https://www.yourhomeworksolutions.com/wp-content/uploads/edd/2020/09/arthur_m._langer___information_technology_and_organizational_learning__managing_behavioral_change_in_the_digital_age_crc_press__2017___1_-1.pdf

Lilienthal, C. (2019). Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt.verlag.

MVC - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN. (2022). https://developer.mozilla.org/es/docs/Glossary/MVC

Menzinsky, A., López, G., Palacio, J., Sobrino, M., Álvarez, R., & Rivas, V. (2018). Historias de usuario. Ingeniería de requisitos ágil.

Overview of the Administrative Management Systems (AMS). (2016, 3 mayo). Financial Services. https://finance.utoronto.ca/policies/gtfm/financial-information-system-fis/overview-of-the-administrative-manaement-systems-ams/

Pérez, L. (2022). Diseño e Implementación de una aplicación para la gestión de partituras. https://ruc.udc.es/dspace/handle/2183/32088

PHP: What is PHP? - Manual. (s. f.-b). https://www.php.net/manual/en/intro-whatis.php

Ravoof, S. (2023, 17 febrero). What Is PostgreSQL? Kinsta®. https://kinsta.com/knowledgebase/what-is-postgresql/

Ross, J. W., Beath, C. M., y Mocker, M. (2019). Designed for Digital: How to Architect Your Business for Sustained Success. The MIT Press.

Ruiz, P. F. R. (2021, 31 enero). Desarrollo de un sistema de gestión de biblioteca en la Institución Educativa Técnico Industrial Pedro A. Oñoro de Baranoa. 10596/39010. https://repository.unad.edu.co/handle/10596/39010?locale-attribute=en

Smiraglia, R. P. (2001). The nature of "a work": Implications for the organization of knowledge. Lanham, Md: Scarecrow Press.

Stevens, E. (2023, 13 marzo). What Is UX Design?. https://careerfoundry.com/en/blog/ux-design/what-is-user-experience-ux-design-everything-you-need-to-know-to-get-started/

Vuetify — A Vue Component Framework. (s. f.). https://vuetifyjs.com/en/introduction/why-vuetify/