

Proyecto

Presentado por:

Cristian Medina - crmedinab@unal.edu.co

Cristian Montañez - cmontanez@unal.edu.co

Justin Rodriguez - jusrodriguez@unal.edu.co

Sergio Ruiz – seruizh@unal.edu.co

Profesor:

Oscar Eduardo Alvarez Rodriguez

14/11/2024



Universidad Nacional de Colombia

Facultad de Ingeniería

Ingeniería de Software

Departamento de Ingeniería de Sistemas

Plataforma de Objetos Perdidos y Encontrados

2. Levantamiento de requerimientos

Problemática

La comunidad universitaria, conformada por estudiantes, profesores y personal administrativo de la Universidad Nacional de Colombia, tiene dificultades en la gestión de objetos perdidos. Ya que los métodos actuales son dispersos y lentos, dificultando la búsqueda. Esto genera pérdida de tiempo, frustración y pérdidas económicas para los afectados.

Además, la falta de un sistema centralizado provoca desconfianza en la comunidad y que los objetos sin reclamar se acumulen en diferentes dependencias sin un mecanismo efectivo para su devolución. Esta situación impacta negativamente la experiencia universitaria, generando estrés e incomodidad.

Objetivo

El objetivo principal es desarrollar una plataforma intuitiva y eficiente que permita a los miembros de la comunidad universitaria reportar la pérdida o el hallazgo de objetos dentro del campus. La plataforma debe simplificar el proceso de registro, búsqueda y recuperación de objetos perdidos, mejorando la experiencia de la comunidad universitaria y reduciendo la cantidad de objetos que permanecen sin reclamar.

Origen de la Idea

La idea del proyecto surge a partir de la observación y experiencia directa de la problemática recurrente dentro de la Universidad Nacional de Colombia que es la pérdida y recuperación de objetos en el campus.

Miembros del equipo, que han trabajado como auxiliares estudiantiles en distintos edificios de la universidad, han visto que se encuentran objetos extraviados diariamente, los cuales en muchos casos no logran ser devueltos a sus dueños, lo que genera una acumulación constante de estos objetos en los edificios. Aunque no es una idea revolucionaria, ya que proyectos de este tipo han sido propuestos en semestres anteriores, ninguno ha sido implementado de una manera efectiva.

Elección de la idea

Inicialmente se realizó una reunión de lluvia de ideas en la que se propusieron varias ideas de proyectos, incluyendo la creación de un e-commerce de zapatos, la cual fue elegida. Sin embargo, tras analizar su propósito, se concluyó que dicha idea carecía de impacto significativo. A partir de esto, se estuvo en búsqueda de un proyecto el cual abordar, donde los miembros del equipo compartieron experiencias sobre la problemática de los objetos

perdidos en la universidad y se definió como un proyecto importante para abordar. Como resultado, se decidió desarrollar este proyecto.

Una vez definido el proyecto, se llevó a cabo la asignación de roles dentro del equipo. Donde cada integrante presentó sus habilidades, intereses y experiencia en distintas áreas del desarrollo y a través de un consenso, se estableció la distribución más adecuada de responsabilidades, asegurando que cada miembro pudiera aportar de manera óptima al proyecto.

Integrante	Rol
Daniel Montañez	Desarrollador Full Stack (Backend y Frontend)
Cristian Medina	Desarrollador Backend
Justin Rodriguez	Desarrollador Frontend y Diseñador de Interfaz
Sergio Ruiz	Desarrollador Full Stack (Backend y Frontend)

Expectativas de los Usuarios

Los usuarios esperan que el sistema permita:

- Reportar objetos perdidos o encontrados fácilmente.
- Buscar objetos por categoría, ubicación y fecha.
- Facilitar la comunicación entre el dueño y quien encontró el objeto.

Requerimientos Funcionales

Con base en las problemáticas mencionadas y las expectativas de los usuarios se han identificado las siguientes funcionalidades claves para la aplicación. Inicialmente para el correcto funcionamiento de la plataforma, es necesario que los usuarios puedan registrarse utilizando su correo. Una vez el usuario sea registrado, podrá iniciar sesión en la plataforma.

Es necesario que los usuarios puedan reportar la pérdida o el hallazgo de un objeto proporcionando información relevante como el nombre del objeto, su categoría (llaves, electrónicos, ropa, entre otros), una descripción breve, su estado (perdido o encontrado), la ubicación donde se extravió o encontró, la fecha del incidente y una imagen opcional. Además, deberán incluir una forma de contacto, como su correo institucional o número telefónico, para facilitar la comunicación con los posibles dueños.

La plataforma permitirá a los usuarios editar o eliminar sus propios reportes en caso de que necesiten actualizar la información o hayan recuperado el objeto. Asimismo, contarán con un historial de reportes, donde podrán visualizar los objetos registrados como perdidos o encontrados, junto con su estado actual.

Para agilizar la búsqueda de objetos, se implementará un sistema de filtrado que permitirá realizar consultas por palabras clave, categoría, fecha de reporte y ubicación. Esto facilitará la localización de objetos según distintos criterios y aumentará la probabilidad de recuperación.

Además, el sistema enviará notificaciones automáticas a los usuarios cuando un objeto encontrado coincida con la descripción de uno previamente reportado como perdido.

En caso de que un usuario identifique su objeto dentro de la plataforma, podrá enviar una solicitud de reclamación. El propietario del reporte recibirá una notificación con los datos del solicitante, lo que permitirá establecer contacto directo y coordinar la entrega del objeto de manera segura.

Para la gestión de usuarios, la plataforma incluirá una interfaz que permitirá visualizar la lista de personas registradas y realizar búsquedas por nombre o correo electrónico. Los administradores del sistema podrán suspender o reactivar cuentas en caso de detectar un mal uso de la plataforma, así como eliminar cuentas de manera definitiva si fuera necesario.

En cuanto a los requerimientos no funcionales, es fundamental garantizar la seguridad de los datos personales de los usuarios mediante mecanismos de encriptación. En términos de rendimiento, la aplicación debe gestionar múltiples registros y búsquedas en tiempo real sin afectar la experiencia del usuario. Además, la usabilidad es clave, por lo que la interfaz debe ser intuitiva y accesible para todos.

Expectativas del desarrollo del proyecto

Como equipo de desarrollo, esperamos obtener varios aprendizajes al llevar a cabo este proyecto, tanto a nivel académico como profesional. En primer lugar, buscamos aplicar de manera práctica los conocimientos teóricos adquiridos en la materia de Ingeniería de Software. Desarrollar una solución para una problemática real nos permite comprender mejor los retos que implica la implementación de un sistema funcional y eficiente.

Además, este proyecto representa una oportunidad para mejorar nuestras habilidades técnicas en diversas áreas del desarrollo de software. Trabajaremos en el diseño de bases de datos, programación web, gestión de proyectos y pruebas de software, esto preparándonos para desafíos en el ámbito profesional.

Por otro lado, nos motiva el impacto que esta plataforma tendrá en la comunidad universitaria. Crear una herramienta que facilite la recuperación de objetos perdidos mejorará la experiencia dentro del campus y brindará una solución a un problema recurrente. Además, promoviendo una cultura colaborativa en la universidad.

3. Análisis de requerimientos

Lista consolidada de requerimientos

Se estructuran, analizan y consolidan los requerimientos y procesos identificados en el desarrollo y síntesis de la idea del proyecto. Primero se presentan los Requerimientos Funcionales (RF) y posteriormente los Requerimientos No Funcionales (RNF).

Requerimientos Funcionales (RF): Prioridad asociada antes de hacer la clasificación MoSCoW

ID	Nombre del Requerimiento	Descripción	Prioridad
RF-1	Registro de usuarios	Permite a los usuarios registrarse con su correo institucional, asegurando la autenticidad de las cuentas mediante un correo de confirmación.	Alta
RF-2	Inicio de sesión	Permite a los usuarios autenticarse en el sistema utilizando su correo institucional y contraseña.	Alta
RF-3	Cierre de sesión	Opción para que los usuarios puedan cerrar su sesión de manera segura en cualquier momento.	Alta
RF-4	Reporte de objetos	Permite a los usuarios registrar objetos perdidos o encontrados, proporcionando detalles como nombre, categoría, descripción, ubicación y fecha.	Alta
RF-5	Editar reportes	Permite a los usuarios modificar los reportes creados por ellos mismos para actualizar información relevante.	Alta
RF-6	Eliminar reportes	Permite a los usuarios eliminar reportes creados por ellos mismos si ya no son necesarios.	Alta
RF-7	Historial de reportes	Proporciona a los usuarios un listado con todos los objetos que han reportado y su estado actual.	Alta
RF-8	Visualización de objetos	Muestra un listado con la información detallada de los objetos reportados como perdidos o encontrados.	Alta
RF-9	Búsqueda por palabras clave	Permite buscar objetos ingresando el nombre o una descripción relacionada.	Media
RF-10	Búsqueda por categoría	Filtra los objetos reportados según una categoría predefinida.	Media

RF-11	Búsqueda por fecha	Permite establecer un rango de fechas para encontrar reportes en un período específico.	Media
RF-12	Búsqueda por ubicación	Facilita la localización de objetos según la ubicación donde fueron reportados.	Media
RF-13	Solicitud de reclamación del objeto	Permite a un usuario enviar una solicitud de reclamación si identifica que un objeto encontrado le pertenece.	Media
RF-14	Validación de reclamaciones	Notifica al usuario que reportó un objeto sobre la solicitud de reclamación y le proporciona la información del solicitante para el contacto.	Media
RF-15	Notificaciones automatizadas	Envía notificaciones a los usuarios cuando se reporta un objeto que coincide con la descripción de su reporte.	Baja
RF-16	Eliminar cuentas	Permite a los administradores eliminar permanentemente cuentas de usuarios.	Baja
RF-17	Suspender o reactivar cuentas	Opción para que los administradores puedan suspender temporalmente o reactivar cuentas de usuarios.	Baja
RF-18	Búsqueda de usuarios por nombre	Permite a los administradores localizar usuarios mediante su nombre registrado.	Baja
RF-19	Búsqueda de usuarios por correo	Permite a los administradores localizar usuarios específicos mediante su correo electrónico.	Baja
RF-20	Visualización de usuarios	Muestra un listado de usuarios con información básica para fines administrativos.	Baja

Requerimientos No Funcionales (RNF): Prioridad asociada antes de hacer la clasificación MoSCoW

ID	Nombre del Requerimiento	Descripción	Prioridad
RFN-1	Seguridad	Implementar mecanismos de autenticación y cifrado para proteger los datos personales de los usuarios y garantizar la integridad del sistema.	Alta

RFN-2	Usabilidad	Diseñar una interfaz intuitiva, accesible y fácil de usar para todos los usuarios, independientemente de su nivel de experiencia con la tecnología.	Alta
RFN-3	Accesibilidad	Asegurar que la plataforma sea compatible con distintos dispositivos como computadoras, tablets y móviles, y cumpla con estándares de accesibilidad.	Alta
RFN-4	Rendimiento	Optimizar el sistema para que pueda gestionar múltiples registros y búsquedas en tiempo real sin afectar la experiencia del usuario.	Media
RFN-5	Escalabilidad	Preparar la arquitectura del sistema para permitir futuras expansiones, como incluir otros campus universitarios o integrar nuevas funcionalidades.	Baja

Priorización y estimación de tiempos

Posteriormente para la priorización se plantea clasificar los requerimientos de acuerdo al análisis MoSCoW, donde se clasifican los elementos según lo esencial que sean para la completitud del proyecto. De esta manera se tiene la siguiente convención:

- Must have (M): Requisitos esenciales para el funcionamiento del sistema.
- Should Have (S): Características importantes, pero que pueden ser pospuestas.
- Could Have (C): Deseables, pero no fundamentales para la primera versión.
- Won't Have (W): Elementos que no se incluirán en este ciclo de desarrollo.

Además se realiza la estimación de tiempos y organización de los requerimientos en un orden lógico de implementación utilizando la escala de fibonacci.

Para esta clasificación se tienen en cuenta varios factores

Recursos disponibles

El equipo está compuesto por cuatro estudiantes con diferentes roles dentro del desarrollo del proyecto, incluyendo desarrollo, arquitectura de software, diseño de experiencia de usuario y pruebas.

Los recursos disponibles presentan ciertas limitaciones que deben considerarse:

- **Disponibilidad de tiempo:** Cada miembro del equipo tiene una cantidad variable de tiempo para dedicarse al diseño, desarrollo y pruebas, ya que deben equilibrar sus responsabilidades académicas y laborales.

- **Repositorio de código:** Se utilizará GitHub como repositorio central para la gestión del código.
- **Herramientas externas:** Cada desarrollador puede elegir su entorno de desarrollo preferido, lo que permite flexibilidad en la implementación. Algunas de las opciones contempladas incluyen Visual Studio. En cuanto a tecnologías, se ha decidido utilizar React para el frontend y Express para el backend. Además, se cuenta con herramientas de apoyo como GitHub y OpenAI ChatGPT para optimizar el desarrollo.

Impacto y alcance

El objetivo principal es desarrollar una aplicación que cumpla con sus funcionalidades esenciales, que han sido priorizadas como Must (y Should) y Should have en esta fase inicial.

Basado en lo anterior se tiene que la clasificación de los requerimiento y su estimación de tiempos es la siguiente:

Must Have (M)		
ID - Nombre del Requerimiento	Esfuerzo (días)	Justificación
RF-001 - Registro de usuarios	3	Se requiere validar los datos ingresados y conectarlo con la base de datos MySQL. La autenticación se hará con JWT para garantizar sesiones seguras.
RF-002 - Inicio de sesión	3	Implementación de autenticación con JWT, gestión de sesiones y validaciones de credenciales en el backend. Es crucial para la seguridad del sistema.
RF-003 - Cierre de sesión	1	Sencillo de implementar, implica invalidar el token JWT en el frontend y limpiar las sesiones activas.
RF-004 - Reporte de objetos	2	Creación de formularios en React con validaciones de datos, envío al backend y almacenamiento en MySQL. Se prioriza una interfaz intuitiva.
RF-005 - Editar los reportes	3	Se debe permitir modificar reportes con restricciones de permisos y asegurar la actualización en tiempo real en la base de datos.
RF-006 - Eliminar los reportes	2	Implementación de eliminación lógica con confirmaciones de usuario para evitar la pérdida accidental de datos.

RF-007 - Historial de reportes	3	Implementación de consultas optimizadas para mostrar reportes anteriores con filtros básicos sin afectar el rendimiento.
RF-008 - Visualización de objetos	3	Diseño de la interfaz para mostrar los reportes almacenados con paginación y filtros básicos.
RFN-001 - Seguridad	2	Protección contra inyecciones SQL, XSS y validaciones de API para evitar vulnerabilidades.
RFN-002 - Usabilidad	5	Se enfocará en la experiencia de usuario con un diseño intuitivo y pruebas de navegación para mejorar la accesibilidad.
RFN-003 - Accesibilidad	3	Se implementarán estándares como etiquetas ARIA, contraste adecuado y compatibilidad con navegación por teclado.

Total Must Have: 30 días

Should Have (S)		
ID - Nombre del Requerimiento	Esfuerzo (días)	Justificación
RF-009 - Búsqueda por palabras clave	3	Se implementará un motor de búsqueda en MySQL con índices optimizados para mejorar la velocidad de consulta.
RF-010 - Búsqueda por categoría	2	Implementación de filtrado de reportes según categoría mediante consultas optimizadas en MySQL.
RF-011 - Búsqueda por fecha	2	Uso de rangos de fechas en MySQL con índices para optimizar las consultas.
RF-012 - Búsqueda por ubicación	1	Se integrará con ubicaciones predefinidas en la base de datos.
RF-013 - Solicitud de reclamación de objeto	3	Se creará un sistema de reclamación con validaciones y estados del objeto (en revisión, aprobado, rechazado).
RF-014 - Validación de reclamos	3	Implementación de un sistema de aprobación manual o automatizado basado en reglas predefinidas.

RFN-004 - Rendimiento	3	Se optimizarán consultas y carga de datos para garantizar tiempos de respuesta rápidos y eficiente uso de los recursos.
-----------------------	---	---

Total Should Have: 17 día

Could Have (C)		
ID - Nombre del Requerimiento	Esfuerzo (días)	Justificación
RFN-005 - Escalabilidad	3	Se diseñará una arquitectura flexible y modular para permitir el crecimiento del sistema sin afectar su rendimiento.

Total Could Have: 3 días

Won't Have (W)		
ID - Nombre del Requerimiento	Esfuerzo (días)	Justificación
RF-015 - Notificaciones automatizadas	5	Se requeriría integración con Firebase o un servicio de correos electrónicos, además de configuración de eventos automáticos.
RF-016 - Eliminar cuentas (solo administradores)	3	Implementación de eliminación permanente con validaciones y políticas de recuperación de datos.
RF-017 - Suspende o reactivar cuentas	3	Creación de estados de usuario en la base de datos con restricciones de acceso según reglas establecidas.
RF-018 - Búsqueda de usuarios por nombre	2	Optimización de consultas en MySQL con índices para mejorar la eficiencia.
RF-019 - Búsqueda de usuarios por correo electrónico	2	Implementación similar a la búsqueda por nombre, pero con validaciones adicionales.
RF-020 - Visualización de usuarios	3	Creación de un panel administrativo con información básica de los usuarios de forma segura.

RF-021 - Funcionalidades avanzadas de gestión de usuarios	8	Desarrollo de permisos personalizados y herramientas de administración avanzadas.
---	---	---

Total Won't Have: 26 días (Fuera del alcance actual)

A partir de este enfoque, se puede elaborar el siguiente cronograma preliminar para iniciar el desarrollo:

Prioridad	Requerimiento	Días
Must Have (M)	Registro de usuarios	3
	Inicio de sesión	3
	Cierre de sesión	1
	Reporte de objetos	2
	Editar los reportes	3
	Eliminar los reportes	2
	Historial de reportes	3
	Visualización de objetos	3
	Seguridad	2
	Usabilidad	5
	Accesibilidad	3
	Total (M)	30
Should Have (S)	Búsqueda por palabras clave	3
	Búsqueda por categoría	2
	Búsqueda por fecha (rango de fechas)	2
	Búsqueda por ubicación	1
	Solicitud de reclamación del objeto	3
	Validación de reclamos	3
	Rendimiento	3
	Total (S)	17
Could Have (C)	Escalabilidad	3
Total (C)		3
Total		50

4. Análisis gestión de software

- **Equipo:** 4 desarrolladores, 2 desarrolladores full stack, 1 desarrollador front y un desarrollador back
- **Duración:** 50 días
- **Tecnologías:**
 - **Frontend:** React
 - **Backend:** Node.js con Express.
 - **Base de datos:** MySQL
 - **Notificaciones:** Integración futura con Firebase o un servicio de emails.
 - **Autenticación:** JWT para sesiones seguras.

La estimación de costo de desarrollo de software se ha realizado de manera mensual, teniendo en cuenta ciertos factores.

Estimación del costo del equipo de desarrollo para los 50 días del proyecto:

Basados en las siguientes fuentes para poder dar un promedio del sueldo de los desarrolladores:

- <https://platzi.com/tutoriales/3208-programacion-basica/24348-descubre-cuanto-gana-un-programador-en-mexico-colombia-argentina-y-ee-uu-en-2023/>
- <https://evalart.com/es/blog/cuanto-ganan-los-desarrolladores-en-america-latina/>
https://www.glassdoor.com.ar/Sueldos/bogota-colombia-qa-tester-sueldo-SRCH_IL.0%2C15_IM1064_KO16%2C25.htm
- <https://co.indeed.com/career/tester/salaries>

Se obtienen los siguientes sueldos promedios, de acuerdo al equipo de desarrollo planteado anteriormente:

Rol	Número de personas	Salario mensual (COP)	Costo por 50 días de trabajo (COP)
Full Stack Developer Junior	2	2.800.000	3.724.000
Front-End Developer Junior	1	2.200.000	2.926.000
Back-End Developer Junior	1	2.400.000	3.192.000

Por lo tanto el costo total para los 50 días de desarrollo es el siguiente:

Rol	Costo total (COP)
Full Stack (2 personas)	7.448.000
Front-End (1 persona)	2.926.000
Back-End (1 persona)	3.192.000

Con estos valores, el total estimado para el equipo de desarrollo durante los 50 días de trabajo es 13.566.000 COP.

Otras herramientas

Las tecnologías seleccionadas para el proyecto son completamente gratuitas en su uso básico. React, como una biblioteca de código abierto, permite crear interfaces sin ningún costo. Node.js y Express también son herramientas de código abierto, sin costos asociados, y están distribuidas bajo la licencia MIT, lo que garantiza su uso sin restricciones. MySQL, como sistema de gestión de bases de datos de código abierto, se puede utilizar gratuitamente bajo la Licencia Pública General (GPL). Para las notificaciones, se puede integrar Firebase, que ofrece un plan gratuito con un límite de 1.000 notificaciones mensuales. Finalmente, JWT (JSON Web Tokens), utilizado para la autenticación segura, es una tecnología libre de costos, ya que no requiere servicios adicionales para su implementación, sólo bibliotecas de código abierto. Estas herramientas proporcionan una solución eficiente y sin costos en su implementación inicial, permitiendo que el proyecto se mantenga dentro del presupuesto.

Alcance

En resumen esta entrega y basado en los requerimientos esenciales tiene como objetivo desarrollar una plataforma web para la gestión de reportes de objetos, enfocándose en la seguridad, usabilidad y accesibilidad. En esta primera entrega, se implementarán funcionalidades esenciales que permitirán a los usuarios registrarse, autenticarse y gestionar reportes de objetos de manera eficiente. La plataforma contará con un historial de reportes, opciones para visualizar y modificar la información, así como medidas de seguridad para proteger los datos. Además, se garantizará una experiencia de usuario intuitiva mediante un diseño accesible y optimizado. Como parte del desarrollo futuro, se planea ampliar la funcionalidad con opciones avanzadas de búsqueda, validación de reclamos y mejoras en el rendimiento del sistema.

8. Diseño y Arquitectura

Arquitectura del sistema

Para el desarrollo de la plataforma de gestión de objetos perdidos en la Universidad Nacional de Colombia, se optó por una arquitectura **cliente-servidor**, estructurada bajo el patrón **Modelo-Vista-Controlador (MVC)**.

Se eligió esta arquitectura porque permite una separación clara de responsabilidades. Mientras que el frontend, desarrollado en **React**, se encarga de la experiencia del usuario, el backend, implementado en **Express**, gestiona la lógica de negocio y la persistencia de datos en **MySQL**. Además, facilita la comunicación entre cada componente mediante solicitudes **HTTP**, lo que garantiza una respuesta rápida y estructurada a las peticiones de los usuarios. De igual manera, permite un acceso centralizado a los datos, ya que estos residen en una única base de datos.

El uso de **MVC** se adoptó debido a sus ventajas en la organización y mantenimiento del código. A través de los modelos, se manipulan los datos utilizando consultas en **MySQL**, definiendo la estructura y las relaciones entre los objetos. La vista, implementada en **React**, proporciona una experiencia de usuario fluida y llamativa. Finalmente, los controladores gestionan las peticiones y respuestas del sistema, separando así la lógica de negocio de la interfaz gráfica para el usuario.

Esta arquitectura facilita la escalabilidad del proyecto, ya que nuevos módulos o funcionalidades pueden incorporarse sin afectar significativamente la estructura existente.

Backend

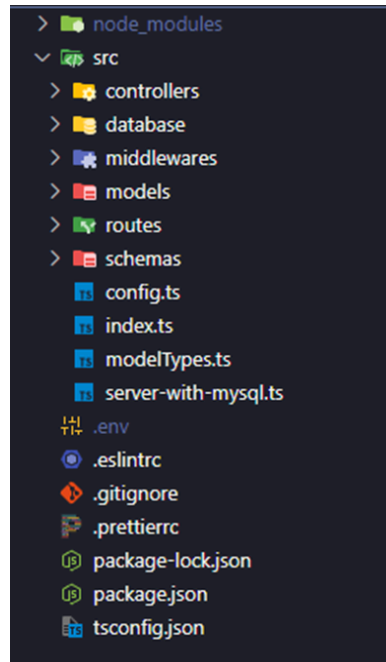
Cada decisión fue tomada pensando en que la plataforma requiere una interacción fluida entre los usuarios y el sistema para realizar reportes, búsquedas y gestionar los objetos perdidos. Para garantizar una experiencia eficiente y segura, la arquitectura elegida permite, mediante el uso de Node.js, manejar eventos asíncronos, lo que mejora en gran medida el rendimiento al procesar múltiples peticiones simultáneamente.

Además, el ecosistema de paquetes gestionados por npm proporciona una gran variedad de herramientas que facilitan el desarrollo y fortalecen la seguridad del sistema. **JWT** permite la autenticación segura mediante tokens, **CORS** gestiona el acceso entre dominios, y **Cookie-Parser** facilita el manejo de cookies. **Zod** se utiliza para la validación de datos, asegurando que la información enviada al servidor cumpla con los formatos esperados. **Bcrypt** se encarga de encriptar contraseñas, reforzando la seguridad del almacenamiento de credenciales. Finalmente, **MySQL2** optimiza la conexión con la base de datos, proporcionando un rendimiento más eficiente en comparación con otras bibliotecas similares.

Express, al ser un framework ligero y flexible, permite la construcción rápida de la API REST, integrando middlewares personalizados que optimizan el procesamiento de solicitudes. Por otro lado, el uso de TypeScript reduce la posibilidad de errores y mejora la

robustez del código, proporcionando un mejor control sobre los tipos de datos y facilitando el mantenimiento del sistema.

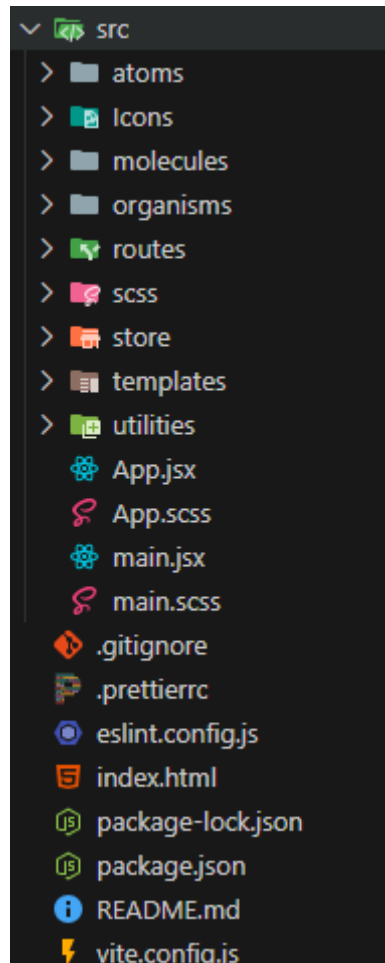
Finalmente, la estructura del backend se organizó de manera que garantizara una gestión y organización óptima, permitiendo la implementación eficiente de nuevos componentes en la aplicación sin comprometer su estabilidad o rendimiento.



Frontend

El frontend de la plataforma está siendo desarrollado en React porque es posible crear interfaces dinámicas y eficientes. Este framework nos permite construir aplicaciones modulares y de alto rendimiento, lo que nos facilita la gestión de los distintos elementos de la interfaz de usuario.

Para organizar y estructurar los componentes, seguimos la arquitectura atómica, ya que esta metodología divide la interfaz en niveles jerárquicos (átomos, moléculas, organismos, plantillas y páginas), lo que garantiza la reutilización y la coherencia en toda la aplicación. Al dividir la interfaz en pequeños componentes reutilizables, se mejora la escalabilidad y el mantenimiento, permitiendo agregar nuevas funcionalidades sin afectar la estructura existente. Además, esta organización optimiza la colaboración entre diseñadores y desarrolladores al establecer una estructura clara.



Para el diseño y la organización de los estilos, empleamos la metodología BEM (Block, Element, Modifier). Esta estrategia nos permite mantener el código SCSS modular y organizado, facilitando su lectura y evitando conflictos en los estilos. Además, BEM nos ayuda a asegurar la escalabilidad al permitir la adición de nuevas clases sin afectar otros componentes de la interfaz.

Diseño de bases de datos:

- **Explicar las decisiones tomadas en el diseño del esquema de la base de datos (por ejemplo, normalización, índices, claves primarias y foráneas).**

El diseño se basa en la normalización, dividiendo la información en tablas específicas (Users, Categories, Locations, Reports e Imágenes) para evitar redundancias y mantener la integridad de los datos. Cada entidad se maneja de forma independiente, permitiendo que los reportes se asocien correctamente a usuarios, categorías y ubicaciones, lo que facilita el mantenimiento y la escalabilidad del sistema. Se han asignado claves primarias en cada tabla que posteriormente llegarán desde el backend para garantizar la unicidad de los registros, y se han establecido claves foráneas en las tablas de Reports e Imágenes para definir relaciones claras entre las entidades. Además, el uso de restricciones como ON DELETE CASCADE asegura que, al eliminar un registro principal, se borren automáticamente los registros dependientes,

manteniendo la integridad referencial. El esquema incorpora restricciones de unicidad, lo que ayuda a prevenir duplicados. La elección cuidadosa de los tipos de datos y la configuración de valores por defecto, como el uso de `CURRENT_TIMESTAMP` y `ENUM`, refuerzan la integridad de la información y facilitan el registro y seguimiento automático de los datos.

- **Justificar si eligieron una base de datos relacional (SQL) o no relacional (NoSQL), y por qué.**

Entre los requerimientos del proyecto, lo que más destaca es la necesidad de una estructura bien definida para poder crear relaciones claras entre los elementos más importantes de la aplicación (usuarios, reportes, búsquedas), lo anterior se ajusta a un esquema relacional. La necesidad de la integridad en los datos y la capacidad de realizar consultas complejas hace que una base de datos relacional sea lo más eficiente, dado a que se puede ejecutar con más facilidad consultas que involucren múltiples criterios. Por lo tanto, escoger una base de datos relacional resulta una opción adecuada para este proyecto. La robustez, las garantías de integridad y la madurez del ecosistema de las bases de datos relacionales (como MySQL) proporcionan una solución estable y segura para gestionar los objetos perdidos y encontrados en el campus universitario.

9. Patrones de diseño

Singleton

Problema que resuelve: Asegura que una clase tenga una única instancia global (p.ej., una conexión a la base de datos), evitando crear múltiples conexiones costosas y garantizando coherencia.

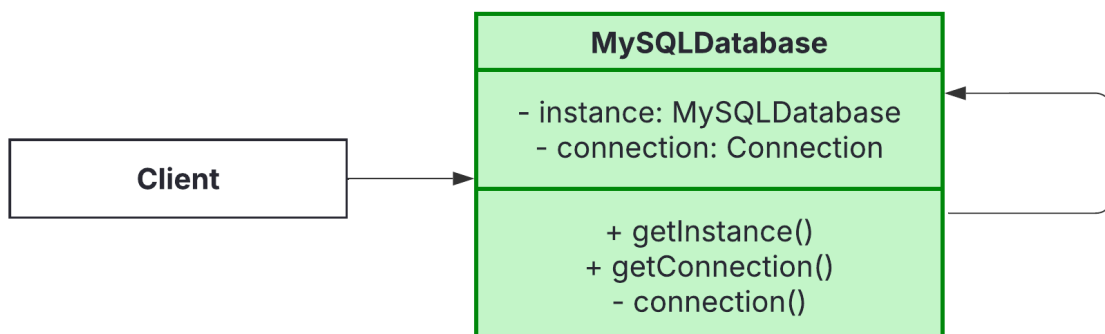
Porque fue necesario: La conexión a MySQL (MySQLDatabase) es un recurso compartido que debe gestionarse centralmente. Sin Singleton, cada modelo podría crear su propia conexión, lo que generaría sobrecarga y posibles conflictos.

Como se implementó:

```
export class MySQLDatabase {  
  private static instance: MySQLDatabase; // Instancia única  
  private constructor() {} // Constructor privado  
  public static async getInstance(): Promise<MySQLDatabase> {  
    if (!MySQLDatabase.instance) {  
      MySQLDatabase.instance = new MySQLDatabase();  
      await MySQLDatabase.instance.connect();  
    }  
    return MySQLDatabase.instance; // Retorna la misma instancia siempre  
  }  
}
```

codesnap.dev

Diagrama:



Factory Method

Problema que resuelve: Centraliza la creación de objetos complejos (como enrutadores) para evitar duplicación de código y permitir flexibilidad en la configuración.

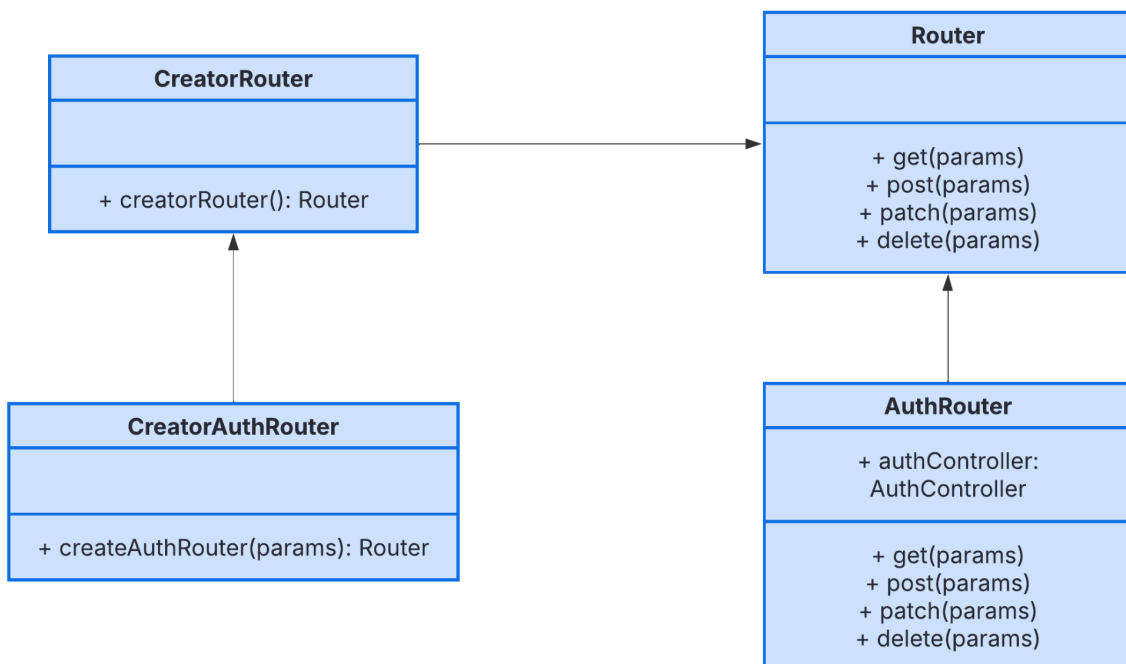
Porque fue necesario: Cada ruta (auth, user, etc.) requiere un enrutador configurado con controladores y modelos específicos. Usar funciones "factory" (como createAuthRouter) simplifica la creación de estos enrutadores y permite reutilizar lógica.

Como se implementó:

```
export const createAuthRouter = (userModel: UserModel): Router => {
  const authRouter = Router();
  const authController = new AuthController(userModel); // Inyección
  authRouter.post('/register', authController.register);
  return authRouter;
};
```

codesnap.dev

Diagrama:



Inyección de dependencias

Problema que resuelve: Evita el acoplamiento directo entre clases, permitiendo que componentes como controladores y modelos sean independientes de implementaciones concretas. Esto facilita la reutilización, el testing y la mantenibilidad.

Porque fue necesario: En el proyecto, los controladores (como AuthController) necesitan acceder a la capa de datos (modelos) para operaciones como registrar usuarios o autenticarlos. Si el controlador creara directamente una instancia de UserModel, quedaría fuertemente acoplado a él, dificultando cambios futuros (p.ej., migrar a otra base de datos) o pruebas unitarias.

Como se implementó:

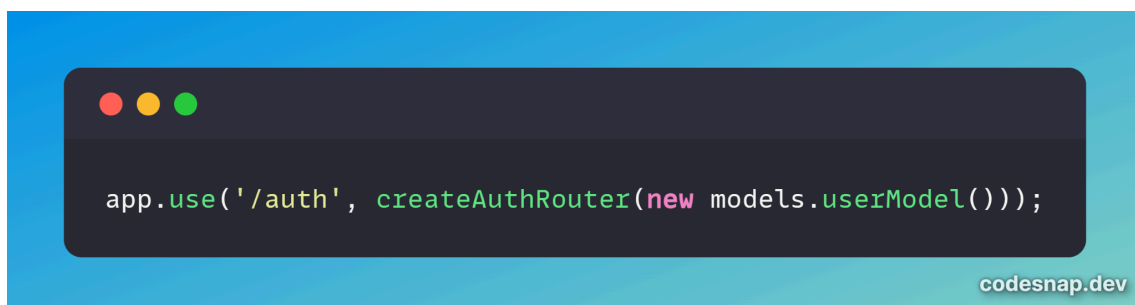
Se definen las dependencias de la aplicación en server-with-mysql.ts.



```
//server-with-mysql.ts
createApp({
  models: {
    userModel: UserModel,
    reportModel: ReportModel,
    categoryModel: CategoryModel,
    locationModel: LocationModel,
    objectModel: ObjectModel,
  },
});
```

codesnap.dev

Se ensamblan las dependencias en index.ts



```
app.use('/auth', createAuthRouter(new models.userModel()));
```

codesnap.dev

Las rutas (como authRoutes.ts) reciben el modelo como parámetro y lo inyectan en el controlador:

```
export const createAuthRouter = (userModel: UserModel): Router => {
  const authController = new AuthController(userModel); // Inyección aquí
};
```

codesnap.dev

```
export class AuthController {
  private userModel: UserModel;
  constructor(userModel: UserModel) {
    this.userModel = userModel; // Inyección a través del constructor
  }
}
```

codesnap.dev

Diagrama: (Hacer zoom)

