Proyecto Final

Presentado por Grupo 6:

Ana María González Hernández - *anagonzalezhe@unal.edu.co*
Daniel Felipe Soracipa - *dsoracipa@unal.edu.co*
Juan José Medina Guerrero - *jmedinagu@unal.edu.co*
Samuel Josué Vargas Castro - *samvargasca@unal.edu.co*

Profesor:
Oscar Eduardo Alvarez Rodriguez
oalvarezr@unal.edu.co

Febrero 9 de 2025

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
2025

Levantamiento de requerimientos

Los requerimientos de un sistema son fundamentales para establecer las bases del desarrollo de una aplicación que cumpla con las necesidades de sus usuarios. Este documento detalla los requerimientos funcionales y no funcionales de la aplicación web destinada a conectar a los dueños de chazas en la Universidad Nacional con los estudiantes.

Los requerimientos funcionales se enfocan en las características específicas y las acciones que el sistema debe realizar, como la gestión de productos y la búsqueda por parte de los clientes. Por otro lado, los requerimientos no funcionales establecen los estándares de calidad, seguridad, rendimiento y usabilidad que garantizarán una experiencia óptima para todos los usuarios. Estos elementos en conjunto aseguran el desarrollo de una solución tecnológica eficiente y accesible.

Contexto del negocio

La aplicación móvil tiene como objetivo principal mejorar la visibilidad y accesibilidad de las chazas (pequeños negocios dentro del campus) de la Universidad Nacional. Esto permitirá que los dueños de estos puestos aumenten su clientela al exponer sus productos y su ubicación de manera digital, incentivando una competencia saludable y facilitando la toma de decisiones de los estudiantes.

Por otro lado, los estudiantes obtendrán una plataforma donde pueden explorar opciones, comparar precios, productos, y ubicaciones, optimizando su experiencia dentro del campus. Esta solución fomenta la digitalización de pequeños negocios y crea un ecosistema dinámico que beneficia tanto a los vendedores como a los consumidores.

Surgimiento de la idea

La idea surge de la identificación de problemáticas y necesidades comunes entre los miembros del equipo, quienes comparten la experiencia de la vida universitaria. En particular, se busca crear un sistema de e-commerce que conecte de manera eficiente las "chazas" (establecimientos de comida informales dentro de la universidad) con los usuarios.

Esta iniciativa nace de la necesidad de facilitar el acceso a la oferta gastronómica de las chazas, superando las limitaciones de tiempo y distancia que a menudo enfrentan los estudiantes. Se espera que esta solución tecnológica mejore la experiencia de compra, tanto para los usuarios como para los propietarios de las chazas, optimizando el proceso de pedido y entrega.

Elección de la idea

La elección de la idea para el proyecto fue un proceso colaborativo en el que cada miembro del equipo aportó sus propuestas y se realizó una votación para seleccionar la opción más viable y atractiva.

Propuestas iniciales

- Aplicación de e-commerce para la universidad: Esta idea central se enfocaba en conectar a los usuarios de la universidad con los diversos establecimientos de comida y otros servicios disponibles en el campus.
- Aplicación de objetos perdidos: Se propuso una plataforma para facilitar la búsqueda y devolución de objetos perdidos en la universidad.
- Aplicación de rutinas de ejercicio: Esta idea buscaba ofrecer a los estudiantes y otros miembros de la comunidad universitaria rutinas de ejercicio personalizadas, adaptadas a las instalaciones y equipos disponibles en la universidad.

Criterios de selección

Para elegir la idea ganadora, el equipo consideró los siguientes criterios:

- Interés y motivación: Se valoró el entusiasmo de los miembros del equipo hacia cada propuesta, buscando un proyecto que motivara a todos a dar lo mejor de sí.
- Viabilidad y recursos: Se evaluó la disponibilidad de tiempo, conocimientos técnicos y recursos necesarios para llevar a cabo cada proyecto en el plazo establecido.

Resultado de la votación

Tras un análisis de las propuestas y una votación democrática, el equipo decidió enfocarse en la aplicación de e-commerce para conectar a los usuarios con las "chazas" de la universidad. Esta idea fue seleccionada por su potencial para resolver una necesidad real de la comunidad universitaria, su viabilidad técnica y el entusiasmo que generó en el equipo.

La decisión de elegir esta idea refleja el compromiso del equipo con la creación de un proyecto que no solo sea innovador, sino que también tenga un impacto positivo en la vida de los estudiantes y otros miembros de la universidad.

Problema principal

El problema principal radica en que los estudiantes de la Universidad Nacional suelen adquirir productos en las chazas del campus, pero enfrentan dificultades para identificar cuáles ofrecen los mejores precios o productos que se ajusten a sus necesidades, especialmente cuando desconocen la existencia de algunas chazas con ofertas más convenientes o específicas. Al mismo tiempo, muchas chazas tienen poca visibilidad entre los estudiantes, lo que limita su capacidad para alcanzar una mayor clientela, generando dificultades en sus ventas y, en algunos casos, reduciendo su sostenibilidad económica. Esta falta de conexión entre estudiantes y chazas representa una oportunidad para implementar una solución que beneficie a ambas partes.

¿Qué expectativas tienen los usuarios potenciales del sistema?

Expectativa de los estudiantes:

- Ahorro de tiempo y energía: Olvidarse de las largas caminatas y filas, y encontrar la comida perfecta en cuestión de segundos, optimizando así su valioso tiempo y energía.
- Poder de decisión al comprar: Acceder a la información de precios permite que los estudiantes puedan elegir la opción que mejor se adapte a sus necesidades.
- Información al alcance de la mano: Acceder a menús detallados, precios actualizados, descripciones de productos e incluso información nutricional, todo en un solo lugar y al alcance de un clic.

Expectativas de los dueños de chazas:

- Visibilidad amplificada: Aumentar el alcance de sus negocios y llegar a un público más amplio de estudiantes, dando a conocer sus productos y ofertas de manera efectiva.
- Marketing estratégico: Promocionar sus productos y ofertas de manera segmentada, dirigiéndose a grupos específicos de estudiantes y aumentando el impacto de sus campañas publicitarias.

¿Qué beneficios esperan ustedes al desarrollar este proyecto?

Al desarrollar este proyecto, esperamos generar un impacto positivo en la comunidad universitaria al facilitar la conexión entre estudiantes y "chazas". Buscamos mejorar la experiencia de compra y promover el desarrollo de los negocios locales, sin descartar la posibilidad de obtener beneficios económicos a largo plazo que permitan garantizar la sostenibilidad de la aplicación.

La aplicación debe contar con un **sistema de registro de chazas**, permitiendo que los dueños de las chazas puedan agregarlas a su cuenta para poderlas gestionar desde su perfil de manera segura. Cada vendedor podrá añadir información sobre su chaza, incluyendo una descripción, horarios de atención y métodos de pago aceptados. Además, podrán **agregar productos** a su catálogo especificando el nombre, precio y una descripción opcional para que los clientes puedan comparar entre distintas opciones. Para mejorar la visibilidad de los productos, se permitirá la **adición de fotografías** y la inclusión de un **código de barras**, el cual podrá ingresarse manualmente o escanearse con la cámara del dispositivo.

La plataforma también debe contar con una **gestión de ubicación** para que los vendedores puedan marcar en un mapa interactivo la ubicación exacta de su chaza dentro del campus, facilitando la localización por parte de los clientes. Además, podrán **agregar fotografías de la chaza**, lo que permitirá que los estudiantes identifiquen con mayor facilidad cada punto de venta.

Por otro lado, los clientes podrán registrarse en la aplicación a través de un **sistema de registro de usuario** y su respectivo **inicio de sesión**, el cual almacenará sus datos de manera segura. A través de su cuenta, los clientes podrán acceder a diversas funciones, como la

búsqueda de productos por nombre o por **código de barras**, que les permitirá encontrar rápidamente los productos disponibles en las chazas. Los resultados se organizarán mostrando las opciones más económicas en primer lugar, brindando así una herramienta eficiente para comparar precios dentro del campus.

Además de ello, la aplicación debe contar con un **sistema de filtrado de chazas**, permitiendo a los clientes refinar su búsqueda según diferentes criterios. Los usuarios podrán filtrar las chazas por horario de atención, asegurando que solo se muestren aquellas abiertas en el momento de la consulta. También podrán seleccionar chazas según la categoría de productos que ofrecen, facilitando la búsqueda de alimentos, bebidas, papelería u otros artículos específicos. Además, se incluirá un filtro por medios de pago aceptados, permitiendo a los clientes encontrar rápidamente aquellas chazas que reciban efectivo, transferencias o pagos digitales, mejorando la conveniencia al momento de realizar compras.

Para mejorar la experiencia de los usuarios, la aplicación incluirá una gestión de favoritos, donde los clientes podrán **guardar chazas de su preferencia** o crear una **lista de productos favoritos**, facilitando el acceso a los artículos que compran con mayor frecuencia. Asimismo, la plataforma contará con un **sistema de reseñas**, permitiendo que los estudiantes dejen comentarios y califiquen las chazas según su experiencia, lo que servirá como referencia para otros clientes.

Con el fin de mejorar la visibilidad de las chazas y dinamizar las ventas, los vendedores podrán **publicar promociones temporales**, notificando a los clientes sobre descuentos o nuevas ofertas en su negocio. Además, los clientes recibirán **notificaciones sobre eventos en sus chazas preferidas**, alertándolos cuando haya cambios en precios, nuevos productos o promociones activas.

La aplicación también debe incluir un **mapa interactivo**, donde los clientes puedan explorar todas las chazas del campus, visualizar sus ubicaciones exactas y acceder a la información de cada una con un solo clic. Para potenciar el alcance de los vendedores, la plataforma permitirá la **integración con redes sociales**, facilitando el compartir perfiles de chazas y promociones en plataformas externas.

En cuanto a la personalización de la experiencia, se considerará la **implementación de soporte multilingüe** en caso de identificarse una demanda dentro de la comunidad universitaria.

En cuanto a los requerimientos no funcionales, la plataforma debe garantizar un **entorno seguro** para los datos de los usuarios, utilizando cifrado para la autenticación y los medios de pago. En términos de **rendimiento**, las búsquedas de productos y la carga de perfiles de chazas deben completarse en menos de dos segundos, asegurando una navegación fluida. Además, la **usabilidad** será un aspecto clave en el diseño, proporcionando una interfaz intuitiva y accesible para facilitar el uso de la aplicación tanto para vendedores como para clientes.

Análisis de requerimientos

Para el análisis de requerimiento se plantea clasificar inicialmente los requerimientos siguiendo el método **M**o**SC**o**W**, 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.

Por ello, los requerimientos quedaron asignados de la siguiente manera:

Requerimientos funcionales

- Registro de chazas (M)
- Adición de productos (M)
- Adición de fotografías del producto (S)
- Adición de códigos de barras del producto (C)
- Adición de ubicación de la chaza (S)
- Adición de fotografías de la chaza (C)
- Registro de cliente (M)
- Inicio de sesión (M)
- Búsqueda de productos por nombre (M)
- Búsqueda de productos por nombre por código de barras ©
- Filtro de chazas (C)
- Gestión de chazas favoritas (S)
- Gestión de lista de productos favoritos (S)
- Creación de sistema de reseñas para calificar chazas (C)
- Publicación de promociones temporales (W)
- Notificaciones a los clientes por eventos en chazas preferidas (W)
- Mapa interactivo de las chazas (W)
- Integración con redes sociales (W)

- Soporte multilingüe (W)

Requerimientos no funcionales

- Seguridad (M)
- Rendimiento (S)
- Usabilidad (S)

Una vez realizada esta clasificación, solamente se continuará el proceso con las historias de usuario correspondientes a las prioridades Must, Should y Could, siguiendo las indicaciones proporcionadas. Las historias clasificadas como Won't se descartan en esta etapa, ya que no representan una prioridad actual y su desarrollo implicaría un desgaste innecesario de planeación y recursos. Además, este enfoque permite adaptarnos de manera más eficiente a los cambios de prioridades en ciclos futuros, cumpliendo con la metodología iterativa y evolutiva de gestión de requerimientos.

Posteriormente, se desarrolla el proceso de estimación de tiempos basada en la sucesión de Fibonacci, donde cada requerimiento requerirá un número de días perteneciente a esta secuencia. Además, se organiza en un orden lógico de secuencia, con posibilidad de ser modificado en el futuro.

Estimación de requerimientos			
Prioridad	**Requerimiento**	**Justificación**	**Estimación**
Must	Registro de vendedores	Requiere la configuración del sistema de autenticación y la creación de una base de datos segura para almacenar los perfiles de los usuarios.	2
	Registro de clientes	Requiere diseñar las relaciones en la base de datos para almacenar la información de las chazas y definir los accesos de los usuarios.	2
	Inicio de sesión	Implica desarrollar la pantalla de inicio de sesión y configurar la base de datos para verificar la identidad del usuario.	1

Estimación de requerimientos			
Prioridad	**Requerimiento**	**Justificación**	**Estimación**
	Adición de productos	Requiere diseñar la interfaz y conectarla con la base de datos para registrar los productos de forma eficiente.	2
	Búsqueda de productos por nombre	Requiere implementar un sistema de búsqueda optimizado.	2
	Seguridad	Incluye la implementación de cifrado, manejo seguro de contraseñas, y pruebas exhaustivas para proteger los datos de los usuarios.	3
	Subtotal Acumulado		**12**
Should	Usabilidad	Requiere diseñar y probar una interfaz fácil de usar.	3
	Adición de fotografías del producto	Implica diseñar una funcionalidad para cargar, almacenar y mostrar imágenes en el perfil de los productos.	3
	Adición de ubicación de la chaza	Integra la aplicación con servicios de mapas y la representación de la ubicación dentro del campus.	3
	Gestión de chazas favoritas	Relaciona los perfiles de los clientes y las chazas en la base de datos.	1
	Gestión de Lista de Productos Favoritos	Similar al caso anterior, pero centrado en los productos.	1
	Rendimiento	Se necesita optimizar las consultas y la arquitectura del sistema para asegurar tiempos de respuesta adecuados.	3
	Subtotal Acumulado		**26**
Could	Adición de códigos de barras del producto	Integra en el sistema el registro y consulta de productos a través de códigos de barras.	2
	Búsqueda de productos por código de barras	Se requiere implementar un sistema de escaneo de código de barras del producto y realizar la conexión a la base de datos	2

Estimación de requerimientos			
Prioridad	**Requerimiento**	**Justificación**	**Estimación**
		de productos	
	Adición de fotografías de la chaza	Requiere una funcionalidad similar a la de las fotografías de productos, pero adaptada al perfil de las chazas.	3
	Creación de sistema de reseñas para calificar chazas	Diseñar la funcionalidad para capturar, almacenar y mostrar reseñas, además de permitir la moderación.	3
	Filtro de chazas por horario de atención, categoría de productos o medios de pago aceptados	Implementa filtros dinámicos en la aplicación con su correcta implementación con la base de datos	3
Total Estimación			**39**

De esta manera, los requerimientos que tienen que estar (M), se desarrollarán en 12 días. Al incluir los requerimientos que deberían estar (S), se llega a un total de 26 días y si se incluye el requerimiento que podría estar (C), se llega a un total estimado de duración del desarrollo del proyecto de 39 días.

Sumado a este análisis se puede plantear si algunos de los requerimientos se encuentran en conflicto, sin embargo, en el alcance actual no se observa ningún tipo de conflicto.

Análisis de gestión de software

En la gestión de proyectos de software, es fundamental analizar la relación entre tiempo, costo y alcance para garantizar una planificación efectiva. Estos tres factores están interconectados, y cualquier modificación en uno de ellos impacta directamente en los demás. Este análisis permitirá estimar el tiempo requerido para el desarrollo de cada módulo, calcular los costos asociados considerando sueldos, licencias, infraestructura y herramientas externas, y definir con claridad los límites del sistema dentro del MVP. Además, se investigará el valor real del trabajo en el mercado para establecer expectativas alineadas con la industria y evitar subvaloraciones en la estimación de costos.

Alcance

El alcance del proyecto se enfocará en incluir únicamente los requerimientos Must (imprescindibles), aquellos que son esenciales para el funcionamiento básico y operativo de la aplicación. Esto asegura que las funcionalidades clave hagan parte del MVP, como el registro y autenticación de vendedores y clientes, la gestión de productos, y el sistema de búsqueda sean implementadas de manera efectiva. Al concentrarse en estos elementos, se logrará una aplicación funcional desde su lanzamiento, atendiendo las necesidades principales de los usuarios.

Al limitarse a los requerimientos Must, el proyecto podrá ser completado dentro de los plazos establecidos, garantizando una experiencia de usuario básica pero efectiva. Las funcionalidades adicionales, incluidas dentro de las categorías Should, Could y Won’t, no se incluirán en esta fase inicial, pero podrán ser consideradas para futuras iteraciones, permitiendo una evolución gradual del sistema sin comprometer la entrega o el rendimiento en el corto plazo.

Tiempo

El tiempo estimado para el desarrollo del proyecto se ha determinado a partir del análisis de requerimientos utilizando la metodología MoSCoW. Según esta evaluación, el desarrollo de los requerimientos clasificados como "Must have" requiere un total de 12 días, por lo que el tiempo

Esta planificación considera las etapas necesarias para cada funcionalidad, incluyendo diseño, desarrollo y pruebas, asegurando que el producto final sea funcional y estable antes de su lanzamiento.

Costo

Para el desarrollo del proyecto, se contará con un equipo conformado por un desarrollador Flutter, encargado de la implementación del frontend móvil, y dos desarrolladores backend, uno senior y otro junior, responsables de la construcción y mantenimiento de la API REST en NestJS, así como de la gestión de la base de datos. Además, se incluirá un diseñador UX/UI, quien se enfocará en la creación de interfaces intuitivas y una experiencia de usuario optimizada. La coordinación y planificación del proyecto estarán a cargo de un Project Manager, asegurando una adecuada gestión de tiempos, recursos y comunicación entre los miembros del equipo.

Para estimar el presupuesto del equipo de desarrollo, se consideran los siguientes roles y sus remuneraciones promedio en el mercado laboral a lo largo de un mes, teniendo en cuenta que el tiempo de desarrollo del proyecto son 12 días laborales:

- Desarrollador Flutter
- Desarrollador Backend NestJS Senior
- Desarrollador Backend NestJS Junior
- Diseñador UX/UI
- Project Manager

Por otra parte, para el despliegue en tiendas móviles, se incurrirá en un costo único de $25 USD para la creación de la cuenta de desarrollador en Google Play Store y $99 USD anuales para la cuenta en App Store. En cuanto a la infraestructura en la nube, se utilizará Render para el despliegue del backend y la base de datos PostgreSQL. Render ofrece un plan gratuito para pruebas, pero se estima que el costo mensual de un plan de pago para producción (con 1 GB de RAM para el backend y 10 GB de almacenamiento en la base de datos) será de $17 USD/mes. Adicionalmente, se utilizará la API de Google Maps, la cual tiene un crédito gratuito mensual de $200 USD. Este crédito es suficiente para cubrir la mayoría de las operaciones, como

geolocalización y visualización de mapas, siempre que el tráfico se mantenga dentro de los límites de uso gratuito.

Los anteriores datos de costos fueron obtenidos de los siguientes sitios web:

- Google Play Store
- App Store -Infobae
- Render
- PostgreSQL- Render
- API de Google Maps

El desarrollo de la aplicación se basará en tecnologías open-source: Flutter para el frontend y NestJS para el backend, lo que no implica costos adicionales por las herramientas de desarrollo.

En total, los costos del proyecto quedan detallados de la siguiente manera:

Categoría	Descripción	Estimación	Costo promedio (USD)	Costo promedio (COP)
Trabajadores	Desarrollador Flutter	2.800.000 COP a 4.800.000 COP mensuales.	926,83	3.800.000
	Desarrollador Backend NestJS Senior	6.000.000 COP a 10.000.000 COP mensuales.	1951,22	8.000.000
	Desarrollador Backend NestJS Junior	2.500.000 COP a 4.500.000 COP mensuales.	853,66	3.500.000
	Diseñador UX/UI	2.500.000 COP a 4.500.000 COP mensuales.	853,66	3.500.000
	Project Manager	6.000.000 COP a 10.000.000 COP mensuales.	1951,22	8.000.000
Licencias	Cuenta de desarrollador Google Play	25 USD pago único	25,00	102.500
	Cuenta de desarrollador AppStore (anual)	99 USD	99,00	405.900

Servicios en la nube	Render (mensual)	17 USD	17,00	69.700
Herramientas externas	API Google Maps	0 USD	0,00	0
TOTAL			6677,59	27.378.100

Las estimaciones de los salarios fueron mayormente obtenidas al realizar investigación en plataformas como GlassDoor, Indeed Colombia, Computrabajo y LinkedIn, aunque en estos últimos muchas veces en las ofertas de trabajo no se muestra el salario o rango de salario esperado.

- GlassDoor
- Computrabajo
- Linkedin

Diseño y Arquitectura

El diseño y la arquitectura de un sistema son fundamentales para garantizar su eficiencia, escalabilidad y mantenimiento a lo largo del tiempo. Una planificación adecuada permite estructurar el flujo de datos, optimizar el rendimiento y facilitar futuras modificaciones sin afectar la estabilidad del proyecto.

En este apartado se presentan dos secciones clave: Arquitectura del Sistema, donde se describe la organización de los componentes y su interacción, y Diseño de Base de Datos, donde se detalla la estructura de almacenamiento, las relaciones entre entidades y la justificación del modelo elegido. Ambos aspectos son esenciales para asegurar la solidez y el correcto funcionamiento de la aplicación.

Arquitectura del Sistema

El proyecto sigue una arquitectura cliente-servidor, donde el frontend en Flutter se comunica con el backend en NestJS a través de una API REST. El backend, a su vez, gestiona las solicitudes de datos, interactuando con la base de datos PostgreSQL para el almacenamiento de información y con la API de Google Maps para obtener datos de ubicación y mapas.

Esta arquitectura fue elegida por su simplicidad y eficiencia, permitiendo una clara separación de responsabilidades entre la interfaz de usuario y la lógica de negocio. Al utilizar Flutter, se garantiza una experiencia multiplataforma eficiente para dispositivos móviles, mientras que NestJS, basado en Node.js, proporciona un backend escalable y modular. PostgreSQL se seleccionó por su capacidad para manejar consultas complejas y su estabilidad en aplicaciones de producción.

La estructura cliente-servidor permite futuras mejoras, como la implementación de autenticación, optimización del rendimiento mediante almacenamiento en caché o la posible incorporación de nuevos servicios sin afectar la base del sistema. Esta arquitectura es adecuada para los requisitos del proyecto y se adapta bien a los recursos disponibles.

Base de datos

El diseño de la base de datos se centra en los casos de uso iniciales de la aplicación, priorizando la gestión de chazas, productos y usuarios. Se implementará un CRUD (Crear, Leer, Actualizar, Eliminar) para la gestión de chazas, permitiendo a los usuarios crear sus propias chazas. Cada chaza podrá tener múltiples productos y categorías, facilitando futuras implementaciones.

Estructura de la Base de Datos

La base de datos se compone de las siguientes tablas, cada una con un propósito específico:

- **Usuarios**: Almacena información de los usuarios, utilizando UUID como clave primaria por seguridad.
- **Chazas**: Almacena información de las chazas, incluyendo su ubicación, descripción y relación con el usuario que la creó.
- **Productos**: Almacena información de los productos, incluyendo su categoría y código de barras.
- **Categorias**: Almacena información de las categorías de productos.
- **Photo**: Almacena las URLs de las fotos de las chazas y productos, las cuales se guardarán en un bucket (Firebase Storage o AWS S3).
- **Chaza_Producto**: Tabla de relación muchos a muchos entre Chazas y Productos, incluyendo el precio y la cantidad.
- **Chazas_favoritas**: Almacena las chazas favoritas de cada usuario.
- **Productos_favoritos**: Almacena los productos favoritos de cada usuario.
- **Calificacion**: Almacena las calificaciones y comentarios de los usuarios sobre las chazas.
- **Medios_pago**: Almacena los medios de pago aceptados por cada chaza.
- **Horario_chaza**: Almacena los horarios de atención de cada chaza.

Relaciones entre Tablas

Las tablas se relacionan de la siguiente manera:

- **Uno a Muchos**: Un usuario puede tener muchas chazas, una chaza puede tener muchos productos, una categoría puede tener muchos productos, etc.
- **Muchos a Muchos**: Una chaza puede tener muchos productos y un producto puede pertenecer a muchas chazas (a través de la tabla Chaza_Producto).

Claves Primarias

Se utilizan IDs enteros (INT) como claves primarias para todas las tablas, excepto la tabla Usuarios que utiliza UUID por seguridad. Se eligió INT en lugar de DOUBLE para las claves primarias debido al crecimiento esperado de usuarios y para optimizar el uso de memoria.

Almacenamiento de Fotos

Las URLs de las fotos se gestionarán en la tabla Photo de la base de datos, mientras que los archivos de imagen se almacenarán en un bucket. Para este propósito, hemos seleccionado Supabase Storage, una solución que facilita la creación y el almacenamiento de manera eficiente.

Supabase Storage ofrece una serie de ventajas que la hacen ideal para nuestro proyecto:

- **Desarrollo ágil:** Supabase simplifica el proceso de desarrollo, lo que nos permite implementar la funcionalidad de almacenamiento de imágenes de forma rápida y sencilla.
- **Integración perfecta:** Supabase se integra de forma nativa con otras herramientas y servicios, lo que facilita la creación de una solución completa y robusta.

Si bien Supabase Storage es nuestra opción preferida debido a su facilidad de uso y sus características, reconocemos que existen otras alternativas viables, como Firebase Storage y AWS S3. La elección final de la plataforma fue influenciada en cuál nos permite un desarrollo de manera más rápida y escalable, también influye que en la misma plataforma (supabase) ya se encuentra nuestra base de datos de producción, lo cual permite una integración más rápida.

Índices y Triggers

Dado el diseño y tamaño actual de la base de datos, la creación de índices resulta innecesaria. El impacto en la velocidad de las consultas sería mínimo o nulo, ya que muchas de ellas se realizan a través de claves primarias, que ya actúan como índices.

La implementación de triggers tampoco se considera necesaria en este momento. Los casos de uso y el diseño de la aplicación garantizan que la información se suministra a la base de datos de forma granular, lo que hace que los triggers sean redundantes.

Normalización

El diseño de la base de datos sigue un enfoque de normalización, lo que significa que los datos están organizados de manera eficiente para evitar redundancias y garantizar la integridad. La normalización se aplica hasta la tercera forma normal (3NF), lo que implica que cada tabla tiene una única responsabilidad y que los datos no se duplican innecesariamente. Por ejemplo, la tabla Chaza almacena información específica sobre las chazas, como su nombre, descripción y ubicación, mientras que la tabla Productos maneja los productos asociados a cada chaza. La relación entre chazas y productos se gestiona mediante una clave foránea (Chaza_id) en la tabla Productos, lo que evita la duplicación de datos y facilita la gestión de la información.

La normalización también ayuda a prevenir anomalías en la inserción, actualización y eliminación de datos. Por ejemplo, si un producto cambia de categoría, solo se necesita actualizar un registro en la tabla Productos, en lugar de modificar múltiples registros en diferentes tablas. Esto no solo mejora la eficiencia, sino que también reduce el riesgo de errores.

Elección de una base de datos relacional (SQL):

La elección de una base de datos relacional (SQL) para este proyecto se basa en varios factores clave que se alinean con los requisitos y la naturaleza de la aplicación. En primer lugar, la aplicación maneja datos altamente estructurados, como usuarios, chazas, productos, calificaciones y medios de pago, que tienen relaciones bien definidas entre sí. Por ejemplo, un usuario puede crear múltiples chazas, una chaza puede tener varios productos, y un producto puede pertenecer a una categoría específica. Estas relaciones son manejadas de manera eficiente

por una base de datos relacional, que permite realizar consultas complejas mediante operaciones JOIN y mantener la integridad referencial a través de claves primarias y foráneas.

Además, la aplicación requiere transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) para garantizar que las operaciones, como la creación de una chaza o la asignación de un producto, se realicen de manera segura y consistente. SQL es ideal para este tipo de operaciones, ya que asegura que los datos no se corrompan incluso en escenarios de alta concurrencia. Por ejemplo, si dos usuarios intentan calificar la misma chaza al mismo tiempo, SQL garantiza que ambas calificaciones se registren correctamente sin conflictos.

Otro aspecto importante es la necesidad de realizar consultas complejas, como obtener todas las chazas favoritas de un usuario, los productos de una chaza específica o las calificaciones promedio de una chaza. SQL es especialmente adecuado para este tipo de consultas gracias a su soporte para operaciones JOIN y agregaciones. En contraste, una base de datos NoSQL, aunque más flexible para datos no estructurados, no maneja tan bien las relaciones complejas ni las consultas JOIN, lo que la hace menos adecuada para este proyecto.

Patrones de diseño

En el desarrollo de esta aplicación, se han implementado patrones de diseño que permiten mejorar la organización del código, facilitar su mantenimiento y promover la reutilización de componentes. Estos patrones ayudan a desacoplar las distintas capas del sistema, garantizando una arquitectura flexible y escalable. En esta sección, se presentan algunos patrones de diseño utilizados, explicando el problema que resuelven, su importancia dentro del proyecto y la manera en que fueron aplicados, junto con diagramas UML que ilustran su implementación.

Frontend con Flutter: Interfaz dinámica y eficiente

El frontend de la aplicación ha sido desarrollado con Flutter, un framework que permite la creación de interfaces de usuario atractivas y altamente optimizadas para múltiples plataformas. Gracias a su arquitectura basada en widgets y su eficiente sistema de renderizado, se logra una experiencia fluida y responsiva para el usuario. En esta sección se describe la estructura del frontend, los patrones de diseño utilizados, y cómo se implementaron las principales funcionalidades para garantizar una interfaz intuitiva y eficiente.

Patrón Repository

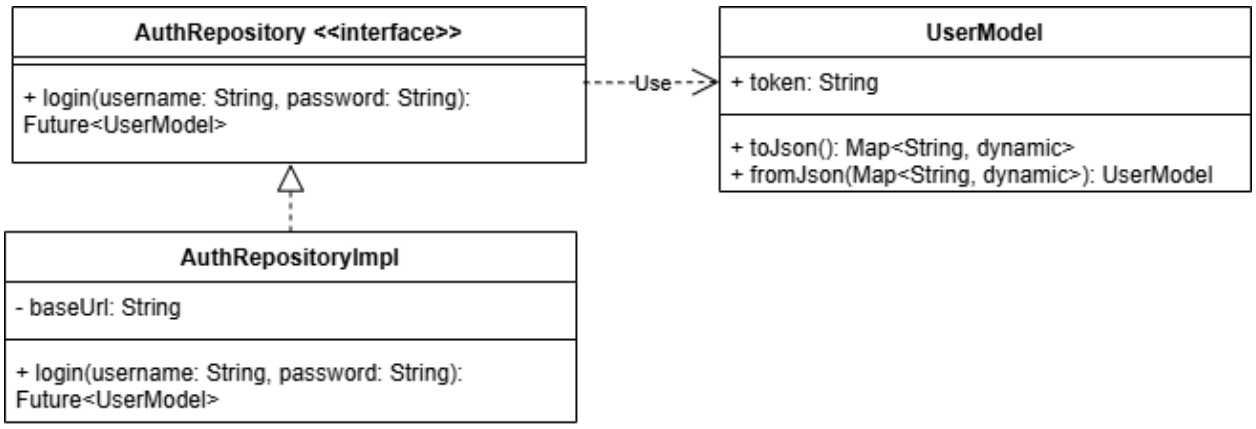
El patrón Repository proporciona una capa de abstracción entre la lógica de negocio y la fuente de datos, evitando que otras partes del sistema dependan directamente de una API, base de datos o cualquier otra fuente de almacenamiento. Esto facilita el mantenimiento y la prueba del código, además de permitir cambiar la fuente de datos sin afectar al resto del sistema.

En el proyecto, el repositorio de autenticación (AuthRepository) era necesario para desacoplar la lógica de negocio de la autenticación de la implementación específica de la API. Esto permite cambiar la API en el futuro sin afectar otras partes del código y facilita la simulación de datos en pruebas unitarias.

Para su implementación se siguió:

- Se creó una interfaz AuthRepository, que define el contrato para la autenticación.
- Se implementó AuthRepositoryImpl, que se encarga de realizar la petición HTTP a la API.
- Otras partes del sistema dependen solo de AuthRepository sin preocuparse por cómo se obtiene la autenticación.

De esta manera, la relación entre las clases es la siguiente:



Patrón Singleton

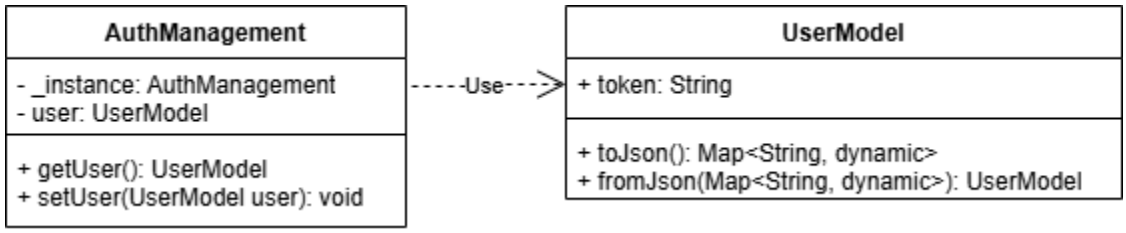
El Singleton asegura que una clase tenga una única instancia en toda la aplicación y proporciona un punto de acceso global a ella. Esto es útil cuando hay datos que deben mantenerse consistentes en todo el sistema, como la sesión del usuario autenticado.

Se necesitaba una única instancia del usuario autenticado (UserModel) disponible en toda la aplicación sin necesidad de pasarlo explícitamente entre clases. Esto evita la repetición de datos y mejora la eficiencia.

Para su implementación se realizó:

- Se creó la clase AuthManager, que almacena una única instancia de UserModel.
- Se utilizó un constructor privado con una fábrica estática para garantizar una única instancia.

De esta manera, la relación entre las clases es la siguiente:



Backend con NestJS: Arquitectura robusta y escalable

El backend de nuestra aplicación está construido con NestJS, un framework de Node.js que nos permite crear aplicaciones escalables y mantenibles gracias a su arquitectura modular y al uso de patrones de diseño. A continuación, describimos algunos de los patrones clave que hemos implementado:

Patrones creacionales

- **Singleton:** NestJS implementa el patrón Singleton de forma predeterminada para la gestión de instancias de clases. Esto garantiza que cada clase tenga una única instancia, que se inyecta mediante el sistema de inyección de dependencias del framework. Esto optimiza el uso de recursos y facilita la gestión del estado de la aplicación.

- **Builder:** Utilizamos el patrón Builder para la construcción de información proveniente de la base de datos. Este patrón nos permite crear instancias de manera más legible y flexible, facilitando la manipulación y transformación de los datos antes de ser utilizados por la aplicación.

Patrones estructurales

- **Decorator:** El patrón Decorator es ampliamente utilizado en NestJS para añadir funcionalidades a clases, métodos o propiedades. En nuestro backend, los decorators se utilizan para definir controladores e inyectar dependencias, como la instancia de la base de datos. NestJS ya proporciona soporte integrado para este patrón, lo que facilita su implementación.
- **Adapter:** Dada la arquitectura hexagonal de nuestra aplicación, el patrón Adapter es una herramienta fundamental. Nos permite transformar la información entre las capas internas y externas, asegurando una comunicación fluida y desacoplada entre los diferentes componentes del sistema.