“MYRESTAURANT”

PROYECTO DE DESARROLLO DE UNA APLICACIÓN MÓVIL

CICLO FORMATIVO DE GRADO SUPERIOR DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA

MANUEL DAVID CASTILLO PÉREZ

ALFONSO GARCÍA

Curso 2024-2025

ÍNDICE

1. Resumen

Este proyecto ha sido realizado con el fin de crear una aplicación móvil, con capacidad para migrar a web con facilidad, que permita gestionar las comandas de un negocio de hostelería. Un sector que se ha vuelto muy exigente con los años. La aplicación desarrolla una serie de funcionalidades clave, como la gestión de mesas, comandas, facturas y productos. Funcionalidades creadas para optimizar el tiempo de servicio, reducir errores, mejorar la organización, etc. El objetivo de este proyecto es desarrollar una aplicación con una base de datos, un servidor y una interfaz que mejore la eficiencia de un negocio del sector de la hostelería a través de sus funcionalidades. En esta aplicación se usará React Native y TypeScript para el desarrollo de la interfaz en todo tipo de dispositivos móviles, sin importar el sistema operativo, Spring Boot y Java para el servidor, ofreciendo la posibilidad de crear un *backend* seguro y escalable, y MySQL para la base de datos. Para el desarrollo se ha utilizado la metodología Scrum, donde se realizaron sprints en los que se iban implementando diversas mejorar a lo largo del desarrollo. Se necesitó de ciertos dispositivos, como un ordenador para el desarrollo y de un móvil para probar la aplicación en un dispositivo real, además de ciertos entonos de desarrollo, emuladores y otros software para manejo de bases de datos o solicitudes HTTP. Se ha llevado a cabo un registro de las tareas realizadas además de una planificación temporal para organizar la temporalización de este proyecto. Utilizamos Git con la plataforma GitHub para el control de versiones. Para el desarrollo la selección de ciertas tecnologías y librerías ha sido de vital importancia debido a sus características y sus aportaciones al proyecto. Finalmente concluimos que se ha logrado desarrollar una aplicación que cumple con los objetivos propuestos, aunque admite mucho margen de mejora debido a la magnitud del sector y la diversidad de elementos que este maneja.

Palabras clave: proyecto, React Native, TypeScript, Spring Boot, Java, MySQL.

2. Abstract

This project was carried out with the goal of creating a mobile application that can easily migrate to the web, allowing the management of orders in a hospitality business. A sector that has become very demanding over the years. The application develops a series of key features, such as table management, orders, invoices, and products. Features designed to optimize service time, reduce errors, improve organization, and more. The objective of this project is to develop an application with a database, a server, and an interface that improves the efficiency of a hospitality business through its functionalities. In this application, React Native and TypeScript will be used for developing the interface across all types of mobile devices, regardless of the operating system, Spring Boot and Java for the server, offering the possibility of creating a secure and scalable backend, and MySQL for the database. Scrum methodology was used for development, where sprints were carried out to implement various improvements throughout the development process. Certain devices were needed, such as a computer for development and a mobile phone to test the application on a real device, as well as specific development environments, emulators, and other software for handling databases and HTTP requests. A record of tasks carried out was kept, along with a timeline to organize the schedule for this project. We used Git with the GitHub platform for version control. For development, the selection of certain technologies and libraries has been of vital importance due to their characteristics and contributions to the project. Finally, we conclude that an application has been successfully developed that meets the proposed objectives, although there is still much room for improvement due to the size of the sector and the variety of elements it handles.

Keywords: project, React Native, TypeScript, Spring Boot, Java, MySQL.

3. Introducción

El sector de la hostelería enfrenta un entorno altamente competitivo, donde la capacidad para ofrecer un servicio rápido, eficiente y de calidad puede ser el diferenciador entre el éxito y el fracaso de un negocio. La demanda de una gran organización y eficacia se ha incrementado, impulsada tanto por las expectativas de los clientes como por la necesidad de adaptarse a un mundo cada vez más digital. En este contexto, las herramientas tecnológicas juegan un papel esencial, facilitando la gestión de operaciones complejas y contribuyendo a la mejora de la experiencia del cliente.

Trabajar con un software de gestión especializado se ha convertido en una prioridad para los negocios de hostelería, ya que permite agilizar procesos de gran importancia como el registro de pedidos, la comunicación entre el personal de sala y cocina, y la gestión de los pagos y reservas. La digitalización de estos procesos no solo contribuye a reducir los errores, sino que también permite un control y seguimiento más exhaustivo de las operaciones del día a día, aportando eficiencia y transparencia.

La literatura técnica y algunas de las opiniones sobre aplicaciones que se dedican a este sector para la gestión de comandas y pedidos en hostelería sugieren que estas deben cumplir con ciertos requisitos fundamentales para ser efectivas. Entre estos, destacan los siguientes puntos:

- **Optimización del tiempo de servicio**: Pérez (2024) señala que la agilidad en la comunicación y procesamiento de pedidos ayuda a reducir los tiempos de espera y mejorar la satisfacción del cliente.
- **Reducción de errores**: También Pérez (2024) indica que un software que minimice la probabilidad de error en la toma de pedidos y cobros es esencial para evitar malentendidos y mantener una buena reputación.
- **Mejor organización y flujo de trabajo**: Por otro lado, el blog de Supercash (2022) nos dice que facilitar la comunicación entre sala y cocina optimiza el tiempo y reduce desplazamientos innecesarios del personal.

- **Facilidad de uso**: Soft Restaurant (2020) nos habla de que la interfaz debe ser intuitiva, permitiendo que los empleados, incluso aquellos con poca experiencia tecnológica, puedan adaptarse rápidamente.
- **Compatibilidad y flexibilidad**: La posibilidad de operar en múltiples plataformas (iOS, Android, web) garantiza que el sistema se adapte a las necesidades de cualquier negocio es una característica que nos indica FU.DO. (2024) en su blog.
- **Mejora en la experiencia del cliente**: El blog de Last.app (s.f.) nos hace saber que los clientes perciben un servicio más fluido, eficiente y personalizado.
- **Integración con otras funciones del negocio**: Speed Tecnología (2022) señala que algunas aplicaciones permiten incorporar funciones adicionales como la gestión de inventario o el control de reservas, ayudando a una administración más completa.
- **Escalabilidad**: Por último, de nuevo Soft Restaurant (2020) nos habla de que las soluciones deben poder adaptarse tanto a pequeños locales como a negocios más grandes, creciendo junto con las necesidades del negocio.

Este proyecto busca contribuir a la modernización del sector de la hostelería mediante una aplicación de gestión de comandas que cumpla con estos requisitos, apoyando así a los negocios en su transición hacia un entorno digital más eficiente y competitivo.

Existen otras aplicaciones en el mercado que cumplen esta función como Mesereando TPV, Kyte, Camarero TPV o el punto de venta para restaurantes de Odoo, sin embargo, en el desarrollo de myRestaurante se busca dar un resultado a largo plazo más completo que permita organizar por completo este tipo de negocios.

4. Descripción y Objetivos

Descripción de la Aplicación

myRestaurant es una aplicación diseñada para optimizar y hacer más eficientes las operaciones que se producen en un negocio de hostelería. Cuenta con una interfaz intuitiva que permite a los empleados gestionar con facilidad y eficiencia el flujo de trabajo. Las pantallas principales son:

1. **Pantalla para Crear un usuario (Register)**: Permite crear una cuenta en la aplicación.
2. **Pantalla de Inicio de Sesión (Login)**: Da acceso a la pantalla principal y al perfil del usuario.
3. **Pantalla Principal (Home)**: En esta sección se desarrollan todas las funcionalidades clave, como la gestión de mesas, la creación de comandas, la generación de facturas, la visualización de la carta de productos y actualización del perfil. Veremos estos apartados con más profundidad en Contenidos.

La aplicación permitirá registrar y organizar las mesas del negocio. Además de crear y gestionar la carta del negocio, registrando todo tipo de productos en categorías específicas. Podrás generar comandas asociadas al usuario y en las que relacionamos la cantidad de un producto, con una mesa y una factura. Finalmente podrás ver una factura asociada a una mesa con un resumen de todas las comandas y un precio final.

¿Hacia que usuarios está destinada?

El público objetivo de *myRestaurant* son principalmente los camareros, quienes podrán usar su propio teléfono o uno facilitado por el negocio para registrar y manejar los pedidos, así como acceder a la información relevante del servicio.

Objetivo General

El objetivo general del trabajo es desarrollar una aplicación de gestión de comandas en un negocio de hostelería, donde se creará una base de datos relacional que permita almacenar y gestionar la información del negocio, donde se desarrollará un servidor que manejará y organizará las peticiones a la base de datos y donde se diseñará y desarrollará una interfaz intuitiva y amigable que permita los empleados trabajar de manera rápida y eficiente.

Objetivos Específicos

1. **Gestión de Usuarios**: Crear, editar y eliminar usuarios en la aplicación, permitiendo un control de acceso adecuado y seguro para cada perfil de usuario.
2. **Gestión de Mesas**: Crear, editar, eliminar y visualizar las mesas del negocio, facilitando la asignación de pedidos.
3. **Gestión de Productos y Carta**: Crear y gestionar la carta del negocio, registrando cada producto en su categoría específica, con detalles como nombre, descripción, ingredientes, alérgenos y precio.
4. **Gestión de Comandas**: Registrar los pedidos de cada mesa, asociando los productos seleccionados y generando un historial de comandas que facilite la organización y permita reducir errores en el servicio.
5. **Generación de Facturas**: Crear un sistema automatizado de facturación que permita emitir facturas detalladas basadas en los pedidos realizados, la factura incluirá el total, fecha, mesa y estado de pago.

5. Contenidos

Vistas de la aplicación

A la hora de desarrollar la aplicación se ha buscado ofrecer una interfaz sencilla, intuitiva y amigable, utilizando colores relacionados con la alimentación, además de aprovechar la posición de la mano para moverse de manera ágil dentro de esta.

Consta de tres pantallas principales, pantalla de **Login**, de **Register** (ver Anexo 2) y de Home, esta última donde residen las funcionalidades principales de la aplicación. Dentro de la Home tenemos 4 subapartados por los que podemos navegar a través de un tab que se sitúa en la parte inferior de la pantalla. En este podemos diferenciar:

- **Pantalla de mesas** (ver Anexo 3): aquí veremos un listado de estas, además de un botón para crear nuevas. En cada una de las mesas encontramos un botón con forma de lápiz donde abriremos un menú de edición para actualizarlas, pudiendo cambiar el nombre y el estado, o eliminarlas. También podemos observar un icono con una hoja y un lápiz, en el podemos escribir nuevas comandas para esa mesa.
- **Pantalla de comandas** (ver Anexo 4): veremos un listado de las comandas con el nombre del producto, la cantidad, la mesas, la fecha y su estado. En la parte inferior de la pantalla observamos un botón para crear nuevas comandas. En cada una de estas comandas tenemos un botón de edición con forma de lápiz y un botón con una hoja y el símbolo del dólar, que nos llevará a la factura asociada con esa comanda.
- **Pantalla de facturas** (ver Anexo 5): podemos observar un listado de facturas que nos ofrecen datos como la fecha, la mesa, un resumen de las comandas y el total de la factura, así como si está o no pagada. Además, podemos editar algunos datos de la comanda utilizando el icono del lápiz y marcar estas como pagadas con el botón inferior de “Pagar”.
- **Pantalla de productos y categorías:** aquí podemos diferenciar dos niveles de profundidad, el primero de ellos en el que muestra las categorías y el segundo en el que vemos los productos de una sola categoría.
    - **Pantalla de categorías** (ver Anexo 6)**:** muestra un listado con rectángulos de colores (elegidos por el usuario), pulsando sobre ellos

accedemos a esa categoría en específico. Al final de esta lista tenemos un botón que nos abre un modal para crear nuevas categorías.

  - **Pantalla de productos** (ver Anexo 7)**:** observamos un listado de los productos de una categoría, en cada uno de ellos tendremos el nombre, la descripción, ingredientes, alérgenos si los tiene, el precio y dos botones, uno de ellos con el icono de un lápiz para editar el producto y otro con una hoja y un lápiz para crear una comanda con este producto. Por otro lado, en la parte inferior tenemos un botón que nos abrirá la vista para crear un nuevo producto de esa categoría.

- **Pantalla de perfil** (ver Anexo 8): si clicamos en el icono superior derecho en la cabecera podremos acceder a la pantalla de perfil, en esta podemos acceder a diferentes sub-pantallas en las que cambiar el nombre, la contraseña o cerrar sesión.

En cuanto a la tecnología utilizada para dar estilos a la aplicación se ha utilizado Tailwind CSS, lo cual se ha hecho a través de su documentación oficial obtenida de Tailwind CSS (s.f.), las razones por las que se ha utilizado esta tecnología son las siguientes:

- **Productividad mejorada**, permitiendo escribir estilos directamente en el HTML. Abba (2022).
- **Flexibilidad y personalización**, permite diseñar interfaces personalizadas. Raiola Networks (2024).
- **Mantenimiento simplificado**, evitando la creación de nombres de clases personalizadas. Guadalupe (2023).
- **Optimización del tamaño del archivo**, genera el CSS necesario para el proyecto. Sanchez (2024).
- **Diseño responsivo simplificado**, facilita la creación de diseños adaptables al tamaño de la pantalla. Hernandez (2024).

Flujo de navegación

En *myRestaurant* se diferencia al iniciar la aplicación la pantalla de Inicio de sesión y la de Registro. Se puede navegar fácilmente entre estas dos, pero para acceder a la

Principal, donde se producen las funcionalidades de la aplicación, tendremos que acceder con nuestro usuario y contraseña.

Una vez en esta pantalla Principal, entre los diferentes tipos de navegación que propone Ferraris (2017) como el Hamburguer Menu, el Tab Bar o la Navegación Basada en Gestos se ha optado por el Tab Bar para navegar entre las pestañas principales de nuestra pantalla principal, debido a su fácil implementación, a que la cantidad de pestañas que vamos a ofrecer son cuatro, que permite un acceso rápido y fácil y además su visibilidad es constante.

De esta manera en la pantalla principal podemos navegar entre las 4 secciones del Tab Bar (Mesas, Comandas, Facturas y Productos) y con la pantalla de Perfil. Dentro de cada una de estas sub-pantallas podemos acceder a funcionalidades específicas de cada una de ellas, como podría ser crear o editar mesas en su pantalla correspondiente. Todo el flujo de navegación completo se puede ver en la imagen del Anexo 9.

Conjunto de datos utilizados

En esta aplicación vamos a utilizar 6 tipos diferentes de datos (Usuarios, mesas, categorías de producto, productos, comandas y facturas), que vamos a guardar en una base de datos relacional MySQL en forma de tabla.

Estos datos serán obtenidos a través de formularios que el usuario rellenará en toda la aplicación, pudiendo en el futuro ver, editar y eliminar estos datos. A continuación, vamos a explicar detalladamente la estructura de cada uno de los datos utilizados:

- **Usuarios** (ver Anexo 10): *user_id*, un identificador único de tipo int (entero) que actúa como clave primaria y se incrementa automáticamente; *name*, almacena el nombre del usuario de tipo varchar(100) (cadena de texto); *email*, correo electrónico del usuario, también de tipo varchar(100), con la característica de unique para evitar duplicados; y *password*, contraseña del usuario, también de tipo varchar(100).

- **Mesas** (ver Anexo 11): *table_id*, identificador único de las mesas, de tipo int, será la clave primaria de esta tabla y aumentará incrementalmente; *table_name*, nombre de la mesa, de tipo varchar (100); *available*, disponibilidad de la mesa, de tipo tinyint(1); *user_id*, será la calve foránea que relaciones la tabla de *tables* con la de *users*.
- **Categorías de producto** (ver Anexo 12)**:** *category_id*, identificador de las categorías, de tipo int, será la clave primaria y aumentará incrementalmente; *category_name*, nombre de la categoría de tipo varchar(100); *user_id*, clave foránea que relaciona el usuario con las categorías; *color*, dato de tipo enum(enumeración) que nos permite establecer el color de la categoría entre varios valores predefinidos, que posteriormente se mostrará en el menú de categorías.
- **Producto** (ver Anexo 13)**:** *producto_id*, identificador de los productos, de tipo int, auto-incremental y clave primara de esta tabla; *producto_name*, nombre del producto, del tipo varchar(100); *description*, descripción del producto, de tipo varchar(255); *ingredients*, ingredientes del producto, de tipo varchar(255); *allergens*, alérgenos del producto, de tipo varchar(100); *price*, precio del producto, del tipo decimal(10, 2) (decimal con profundidad de dos dígitos); *user_id*, clave foránea que relaciona el usuario con los productos; *category_id*, clave foránea que relaciona la categoría con los productos.
- **Comandas** (ver Anexo 14)**:** *order_id*, identificador de las comandas, de tipo int, auto-incremental y clave primaria de esta tabla; tenemos varias claves foráneas (*table_id, producto_id, invoice_id* y *user_id*) que relacionan comandas con diferentes tablas, todas de tipo int; *quantity*, indica la cantidad de ese producto, de tipo int; *order_date*, establece la fecha en la que se ha generado la comanda, de tipo timestamp; *status*, muestra el estado de la comanda, de tipo *enum*, permite seleccionar entre 4 valores predeterminados.
- **Facturas** (ver Anexo 15)**:** *invoice_id*, identificador de las facturas, de tipo int, auto-incremental y clave primaria de esta tabla; *total*, cantidad de dinero total de la factura, de tipo int; *invoice_date*, fecha en la que se genera la facura; *paid*, valor de tipo tinyint(1) utilizado para identificar si la factura está o no pagada; *table_id* y *user_id*, claves foránes que conectan facturas con sus respectivas tablas.

6. Metodología y planificación

Enfoque del proyecto y metodología de trabajo.

En mi proyecto he optado el método ágil conocido como Scrum, en el cual he divido los bloques de trabajo en sprints donde buscaba desarrollar la aplicación de manera iterativa e incremental. Es decir, buscaba ofrecer versiones funcionales de la aplicación con pequeñas mejoras e implementaciones que aporten valor a cada bloque de trabajo realizado.

Este enfoque es interesante debido a sus características:

- **Adaptabilidad y flexibilidad:** permite adaptarse rápidamente a los cambios, incorporar nuevas ideas o implementar ajustes durante el desarrollo. (Bailon, 2024)
- **Entregas incrementales y valor continuo:** mediante los sprints permite entregas regulares e incrementales. (Martins, 2024)
- **Mejora la calidad del producto:** promueve la colaboración entre el equipo, lo que contribuye al conocimiento profundo de los requisitos y a la identificación temprana de posibles problemas. (Chuuck, 2016)
- **Transparencia y visibilidad:** a través de reuniones y entregas regulares. (Ledesma, s.f.)
- **Mejora continua y responsabilidad:** al incentivar la reflexión y la adaptación, permite identificar áreas a mejorar y procesos que pueden ser optimizados. (Lopez, 2023)

Requisitos necesarios

Para la realización de este proyecto he necesitado varios elementos, los cuales son:

- **Dispositivos**: por supuesto es necesario disponer de un ordenador, además es recomendable también el uso de dispositivos móviles, si es posible Android y iOS para poder hacer pruebas en ambientes reales. En mi caso dispongo de un dispositivo Android, aunque la mayor parte del tiempo he utilizado emuladores tanto de Android como de iOS.
- **Entornos de Desarrollo Integrado:** hemos necesitado de cuatro IDEs, cada uno de ellos utilizado por sus características.

  - **Visual Studio Code:** utilizado para desarrollar el front-end debido a su facilidad de uso con React-Native y Type-Script. (*Visual Studio Code - Code Editing. Redefined*, 2021)
  - **Inteligent Idea:** utilizado para desarrollar el back-end de la aplicación debido a gran experiencia de desarrollo con Java. (*IntelliJ IDEA – The Leading Java And Kotlin IDE*, 2021)
  - **Android Studio:** utilizado para emular el dispositivo android en mi ordenador. (*Cómo Descargar Android Studio y App Tools - Android Developers*, s. f.)
  - **XCode:** utilizado para emular el dispositivo iOS en mi ordenador. (Apple Inc., s. f.)
- **Otros softwares relacionados:** también hemos utilizado softwares para agilizar y hacer más eficiente el proceso de desarrollo como son:
  - **MySQL Workbench:** una herramienta visual integrada que facilita el diseño, desarrollo y administración de la base de datos MySQL. (*MySQL :: MySQL Workbench*, s. f.)
  - **Insomnia:** aplicación de desarrollo de APIs, utilizada para pruebas mediante solicitudes HTTP, como Post y Get. Facilita la configuración de encabezados, parámetros y cuerpo de la solicitud. (*The Collaborative API Development Platform*, s. f.-b)

Tareas realizadas y descripción

A continuación, vamos a desglosar las tareas realizadas a lo largo del proyecto:

- Inicio del proyecto: en esta fase se buscan establecer las bases y objetivos de la aplicación.
  - Búsqueda de ideas y objetivos: investigación de cual podría ser una buena idea que desarrollar, estableciendo objetivos acordes a esta idea.
  - Selección de los lenguajes y tecnologías: investigación de que lenguajes y tecnologías de programación son los adecuados en función de los objetivos.
  - Instalación de los softwares necesarios.
  - Creación de las carpetas básicas del proyecto.
  - Creación del repositorio: implementación de un control de versiones.

- Diseño de la aplicación e implantación de las tecnologías a utilizar:
    - Diseño de los modelos de datos
    - Creación de las tablas en la base de datos
    - Creación de la estructura básica de carpetas en la api(servidor) y de los modelos.
    - Diseño visual de la aplicación y del flujo de navegación de esta.
    - Creación de la estructura básica de la app(front) y primeros pasos en el flujo de navegación
- Desarrollo de la aplicación:
    - Creación de las lógicas y visuales para el registro y autentificación de usuarios.
    - Instalación de Tailwind CSS, mejora de los estilos.
    - Controlar las excepciones y errores mediante la creación de una clase (CustomException).
    - Mejorar el control de errores en la app.
    - Implementar Expo Router para conseguir navegaciones más complejas. Uso de diferentes pantallas(Login, Register y Home) y de Tab Bar (Tables, Orders, Invoices y Products) .
    - Implementación de iconos y mejoras visuales.
    - Implementación de variables de entorno.
    - Creación de la vista de profile y desarrollo de cierre de sesión.
    - Creación de la lógica y de las pantallas de cambiar nombre y cambiar contraseña de usuario.
    - Crear la primera versión de los validadores
    - Crear la primera versión de los validadores.
    - Implementación del encryptedStorage para guardar el identificador del usuario.
    - Implementar JasonWebToken en la api y actualizar peticiones utilizando el token.
    - Crear la lógica en la api y crear las pantallas en la app relacionada con el manejo de mesas.
    - Crear la lógica en la api y crear las pantallas en la app relacionada con el manejo de categorías.

    - Refactorizar encryptedStorage
    - Implementar Toast en la aplicación.
    - Crear la lógica en la api y crear las pantallas en la app relacionada con el manejo de productos.
    - Crear la lógica en la api y crear las pantallas en la app relacionada con el manejo de comandas.
    - Crear la lógica en la api y crear las pantallas en la app relacionada con el manejo de facturas.
    - Mejorar los validadores
    - Mover el acceso del perfil del TabBar a la esquina superior derecha.
    - Crear la lógica y los cambios en las vistas para manejar los colores de las categorías.
- Fase final
    - Repasar la aplicación en general
    - Hacer el merge de la rama de develop a main.
    - Elaborar el documento README del proyecto.

Planificación temporal

En cuanto a la planificación temporal del proyecto se ha dividido en 4 fases, la primera de ellas, Inicio del proyecto, Diseño de la aplicación e implementación de tecnologías, Desarrollo de la aplicación y Fase final.

En la fase de Inicio del proyecto se realizan una serie de tareas para ir creando la estructura del proyecto, se instalan los softwares necesarios y se prepara el entorno para empezar a trabajar. Esta etapa empezó el 10 de octubre y terminó el 21 del mismo mes.

Sobre la fase de Diseño de la aplicación e implementación de tecnologías, se trata de una etapa en la que se establecen las bases de la base de datos y las bases del proyecto en cuanto al código del *frontend* y del *backend.* Además de los diseños del flujo de datos y de la aplicación. Esta etapa empezó el 22 de octubre y terminó el 6 de noviembre.

La fase principal es la de Desarrollo de la aplicación, en esta fase se van a crear todas las funcionalidades que dan lugar a la aplicación final, como son peticiones a la base de datos, las lógicas que manejan el servidor, las peticiones de la app al servidor, los elementos visuales de la app, etc. Esta etapa empezó el 7 de noviembre y terminó el 17 de enero.

Por último, tenemos la Fase final, en la que vamos a revisar el proyecto, unirlo a la rama principal y crear el documento Readme. Empezó el 18 de enero y terminó el 20 del mismo mes.

Para ver información más detallada sobre la planificación temporal puedes ver el Anexo 16.

Sistema de control de versiones

En cuanto al control de versiones he optado por Git junto con la plataforma GitHub. Esto aporta seguridad y control a la hora de programar, permite controlar el historial de cambios en el código, guardar el código en la nube, además de trabajar en varias ramas. En este caso he utilizado dos ramas, la rama develop donde he ido subiendo todas las actualizaciones del código y la rama main donde he subido la primera versión del proyecto.

Puedes acceder al repositorio completo de este proyecto con este enlace: https://github.com/manuelda27999/myRestaurant

7. Desarrollo

Tecnologías seleccionadas

Vamos a desarrollar las tecnologías utilizadas, diferenciando *frontend* y *backend*:

- **Frontend**:
  - Esta parte se va a desarrollar utilizando el framework de React-Native, debido a que permite realizar desarrollo multiplataforma, es decir, escribir una única base de código que se traduce en código nativo para varias plataformas (Android, iOS y web) (SLU, 2023) y debido a que su rendimiento es cercano a nativo, ofreciendo una experiencia de usuario similar a las aplicaciones desarrolladas de forma nativa (Jose, 2024).
  - En cuanto al leguaje utilizado junto a React-Native utilizamos TypeScript, este lenguaje que extiende de JavaScript, debido a su tipado estático, su mejora de la legibilidad del código, a el mantenimiento simplificado y a la compatibilidad con estándares JavaScript (Surra, 2024).
- **Backend**:
  - Para la parte del servidor se utilizará el lenguaje de programación Java debido a que la Máquina Virtual de Java (JVM) lo hace altamente portable a otras plataformas (Jesús, 2024), a que está orientado a objetos, por lo que es sencillo desarrollar estructuras modulares y escalables (*AIX 7.3*, s. f.-b), también debido a su seguridad, al incorporar gestión automática de memoria y verificado de tipos en tiempo de ejecución (Jesús, 2024), además de su rendimiento por las mejoras en la JVM y técnicas como la compilación *Just-In-Time* (Jesús, 2024) y amplio ecosistema y soporte comunitario.
  - Junto a Java vamos a utilizar el framework de Spring Boot debido a su configuración automática, es decir, reduce la necesidad de configuraciones manuales extensas, a que facilita la creación de aplicaciones autónomas que se ejecutan por sí mismas, tiene una amplia comunidad y ecosistema, se puede integrar Spring Security, debido a sus características permite un desarrollo rápido y una mayor productividad. (IBM, s.f.)

Dependencias y librerías

En cuanto a las dependencias y librerías de igual manera vamos a distinguir *frontend* y *backend* para su explicación:

- **Frontend**: a continuación, se expone un listado de las librerías y paquetes más significativos utilizados en esta parte de la aplicación, acompañados de una breve explicación.
    - Expo/vector-icons: proporciona iconos vectoriales personalizables.
    - React-native-async-storage: permite el almacenamiento de datos de manera persistente en el dispositivo.
    - React-native-picker: ofrece una componente de selector para elegir entre opciones.
    - Classnames: facilita la integración de código JavaScript dentro de las clases de CSS.
    - Lodash: ofrece utilidades de programación funcional y modular, en mi caso lo he utilizado para hacer copias profundas de objetos.
    - Nativewind: integra Tailwind CSS en React Native.
    - React-native-root-toast: muestra notificaciones de tipo toast en la aplicación.
    - React-native-safe-area-context**:** gestiona áreas seguras en dispositivos con pantallas con muescas o bordes curvos.
    - React-native-screens**:** optimiza la navegación y el rendimiento de las pantallas.
    - React-native-web**:** permite que los componentes React Native se ejecuten en la web.
    - Tailwindcss: framework de utilidades CSS para estilos rápidos y responsivos.
    - Babel/core**:** es el compilador de JavaScript.
    - Types/react**:** proporciona los tipos para React en TypeScript.
    - Expo**:** framework y plataforma para construir aplicaciones React Native.
    - Expo-router**:** gestiona la navegación y las rutas en aplicaciones Expo.
- **Backend**: en esta parte vamos a desarrollar las librerías y paquetes utilizados en la parte del servidor, junto a una breve explicación:

- Spring-boot-starter-data-jpa: proporciona soporte para la persistencia de datos con JPA (Java Persitance API), facilita la integración de JPA en aplicaciones Spring Boot, es utilizado para realizar operaciones CRUD.
- Spring-boot-starter-web: incluido debido a que permite realizar servicios RESTful, gestionando solicitudes HTTP y facilitando la creación de la API.
- Mysql-connector-java: es el conector JDBC oficial de MySQL para Java, esencial para establecer comunicaciones entre este tipo de base de datos y Java.
- Lombok: es una biblioteca que permite reducir la verbosidad del código al generar métodos comunes.
- Java-dotenv: permite cargar variables de entorno desde un archivo .env
- JJWT: permite trabajar con JWT (Json Web Token) en Java, permite crear, firmar, verificar y analizar este tipo de datos en aplicaciones Java.

Requisitos del sistema

- Sistema operativo: se recomienda el uso de Windows 10 o superior, macOS 12 o superior o Ubuntu 16 o Fedora 24 en el caso de Linux. Cabe destacar que no será posible el uso de XCode sin el sistema macOS.
- Procesador: se recomienda un Intel Core i7 de 10ª o 11ª generación, un AMD Ryzen 7 de series 3000 o 5000, o el Apple M2.
- Memoria RAM: se recomiendan al menos 8GB de RAM.
- Almacenamiento: se recomiendan al menos unos 25GB de almacenamiento libre, aunque podrían ser más.

Diario de desarrollo

A lo largo del desarrollo de esta aplicación han surgido algunos problemas y errores que se han ido solucionando paulatinamente. A continuación, me dispongo a explicar algunos de ellos que se deben de destacar:

- Creación del modelo de datos: a la hora de crear los modelos de los datos con los que vamos a trabajar han surgido algunos conflictos sobre como relacionar las comandas con el resto de los datos, debido a que este es una combinación de los demás, necesitaba pertenecer a un usuario, tener asociado un producto,

una mesa y una factura, además de una fechan cantidad y un estado que posteriormente se añadiría. Entonces ha sido un reto buscar que este y el resto de los datos utilizados coexistan y se relacionen sin afectar a la integridad de la base de datos.

- Selección de la herramienta de estilos: al comenzar con los diseños más primarios de la aplicación se encontrón que la manera de dar estilos a los componentes no era para nada ágil, para poner solución a este problema se implementa Tailwind CSS con el objetivo de agilizar y facilitar el uso de estilos responsivos en el fututo.
- Control de errores personalizado: conforme se fueron desarrollando las primeras funcionalidades se observó cómo no era posible comunicar de manera personalizada al usuario los errores o avisos que se producían durante la ejecución de la aplicación. Por esta razón se buscón desarrollar una clase de excepciones personalizadas que habilita al código la capacidad de ofrecer excepciones con un Http.status y un mensaje adaptado a la situación.
- Cambio de react-navigation a expo-router: cuando se intentaron crear flujos de navegación más complejos con implementaciones como TabBar nos vimos ante el problema de que con react-navigation no era suficiente. Tras investigar descubrimos expo-router como una alternativa totalmente adaptada a expo donde es muy sencillas la implementación de diferentes tipos de rutas utilizando un sistema basado en archivos, además de ofrecer varias herramientas para crear vistas muy útiles.
- Uso de variables de entorno: se observó que algunos de los datos críticos de la aplicación como el puerto al que realiza las peticiones la app hacia la api, o el puerto, usuario y contraseña de la base de datos estaban visibles, por lo que para solventar esto se hizo uso de las variables de entono tanto el parte del *frontend* como del *backend*.
- Uso de tokens: en este proyecto se estaba utilizando el identificador del usuario como medio de seguridad para ver quien realiza las peticiones al servidor, esto es seguro debido a que sería más interesante utilizar un token generado en este caso con Json Web Token. Al hacer esta implementación ahora las peticiones al servidor son hechas con un token que identifica al usuario, ofreciendo mayor seguridad al ser un dato difícil de replicar.

- Almacenar token y guardar sesión: para poder acceder al token desde cualquier parte de la aplicación y además guardar la sesión del usuario que ya ha iniciado sesión aun cuando cierra la aplicación se ha utilizado react-native-async-storage, el cual nos permite almacenar y recuperar información sin importar si se ha cerrado la aplicación.
- Uso de Toast: a la hora de ofrecer más información al usuario de los procesos que han ido correctamente se implementa react-native-root-toast, una librería que nos permite lanzar toast de forma sencillas con mensajes personalizados, facilitando de esta manera que el usuario reciba pequeños mensajes como “Comanda editada con éxito” cuando se edita una comanda, por ejemplo.
- Asegurar integridad de la base de datos: este ha sido uno de los mayores retos durante el desarrollo de la aplicación. Esto se debe a que varios de los datos están conectados, como por ejemplo las comandas con los usuarios, mesas, productos, facturas, etc. Por esta razón fue de vital importancia asegurarnos de que la modificación de un dato no comprometía la integridad de la base de datos. Para ellos se realizan numerosos controles donde nos aseguramos de que los datos están actualizados entre ellos o que no se eliminan datos que dependen de otros. Por ejemplo, a la hora de generar una factura esta se ve afectada por las comandas que tiene asignadas, la lógica del servidor se encarga de actualizar el total de una factura cuando una de sus comandas es editada, evitando fallos como que el valor de las comandas no se relaciona con el total de la factura.
- Mejorar comandas y categorías: a estos dos datos se le implementaron dos mejoras para conseguir una experiencia de usuario más completa. A las comandas se le asignaron 4 estados (Pendiente, cocinando, lista y entregada) que ofrecen información muy valiosa sobre el estado de preparación. En cuanto a las categorías se añadió un listado de colores entre los que se puede elegir para que diferenciar entre ellas sea más sencillo y rápido.
- Cambiar la posición del botón de perfil: por último surgía el problema de que en nuestro TabBar teníamos 5 elementos, entre ellos el perfil, el problema radica en que el perfil es una pantalla poco utilizada y que no necesita tener un acceso tan rápido como podrían ser comandas o facturas, por esta razón para liberar el TabBar de 5 a 4 elementos y darles de tal forma más protagonismo a estos

se decidió mover el botón que nos dirige a perfil a la esquina superior derecha de la cabecera.

Estructura de carpetas

La estructura de carpetas de este proyecto tiene dos claras vertientes, la parte del *frontend* y la parte del *backend*, por lo que vamos a explicar ambas por separado:

- **Frontend**: esta área del proyecto contiene varios archivos de configuración como el package.json, el tailwind.config.js o el metro.config.js. También contiene el env, el cual gaurda las variables de entorno. Además de estos contiene el index.ts, archivo principal, el cuál es el punto de carga de toda la aplicación. Por otro lado, tenemos una serie de carpetas de las cuales vamos a desarrollar las más importantes:
    - app: se trata de la carpeta principal del frontend, en ella se ubican todas las rutas de la app. Podemos diferenciar una primera estructura formada por un layout, que controla el flujo de navegación entre el archivo index (pantalla de login), el archivo register, la carpeta home y carpeta profile, en la cual tenemos dentro la pantalla principal de este componente y sus diferentes modales. Si seguimos ahondando en la carpeta home observamos un layout que controla el flujo de navegación dentro de esta y las carpetas de tables, orders, invoices y products, las cuales contienen tanto sus vistas principales como sus correspondientes modales.
    - logic: en esta carperta se guardan todas las lógicas que realizan las peticiones al servidor, organizadas en diferentes carpetas en función del dato con en el que trabajan, de esta manera encontramos dentro de logic las carpetas categories, invoices, orders, products, tables y users.
    - utilities: en esta carpeta he reunido algunas las lógicas de utilidad general en la app, como son customAlerts, encryptedStorage, toastClass y validators.
- **Backend**: en esta sección del proyecto podemos observar varias carpetas y archivos, entre estos últimos cabe destacar el pom.xml, archivo de configuración de Maven, el .env, donde se guardan las variables de entorno y el .gitignore que define los archivos y carpetas que va a ignorar git. Por otro

lado en cuanto a las carpetas podemos destacar .mvn, que permite ejecutar Maven sin tenerlo instaldo globalmente y src, donde tras ahondar se encuentra la lógica principal de esta sección y la cuál vamos a desglosar a continuación:

- Clase ApiAplication: es el punto de entrada principal de la aplicación, desde ella se ejecutan el resto de los elementos.
- controllers: en ella encontramos los controladores REST.
- entitys: modelos originales que representan las entidades de la base de datos.
- customEntitys: en ella observamos modelos personalizados que tienen más datos que las entidades de la base de datos.
- error: en esta carpeta están las clases relacionadas con excepciones personalizadas.
- interfaces: podemos observar las interfaces para los servicios.
- services: contiene la lógica de negocio o servicios de la aplicación.
- utilities: contiene la clase JWTUtils la cuál será útil en los controllers de la aplicación.

Base de datos

La base de datos utilizada ha sido MySQL, debido a que tiene ciertas características como son que es gratuita, tiene un alto rendimiento, es fácil de usar, ofrece una gran flexibilidad y escalabilidad y posibilidad de implementar ciertas herramientas de seguridad como autentificación o cifrado de datos (Jesús, 2024).

En esta base de datos se han utilizado comandos DDL (Data Definition Languaje) para crear la estructura de la base de datos con CREATE, ALTER y DROP. En el Anexo 17 se pueden observan los diferentes comandos DDL que utilicé para crear las tablas.

Ver el Anexo 18 para obtener más información sobre cómo se relacionan las diferentes tablas de la base de datos.

8. Conclusiones y proyección a futuro

Conclusión

El proyecto de myRestaurant ha concluido ofreciendo una primera versión que cumple las necesidades básicas de una aplicación para manejo de comandas. La selección de tecnologías y las estrategias seguidas durante su desarrollo han dado lugar al resultado esperado, ofreciendo una aplicación que agiliza, facilita y aporta seguridad a los negocios de este sector. Cabe destacar que durante el desarrollo de esta se han detectado carencias y se ha observado que las aplicaciones para el manejo de este tipo de negocios pueden llegar a contener gran cantidad de funcionalidades, por lo que es un sector en el que el desarrollo de software tiene mucho trabajo que hacer.

Puntos que mejorar

Sin embargo, el desarrollo de esta aplicación tiene algunos aspectos que se pueden mejorar como son:

- Diseño responsivo: la implementación de este puede ofrecer una mejor experiencia al usuario sin importar en qué tipo de dispositivo la está utilizando.
- Filtros para las facturas y comandas: realizar filtros en función de fechas o estados en facturas y comandas es otra característica que mejoraría mucho la experiencia de usuario.
- Implementación de imágenes: este punto ayuda con la experiencia de usuario y la personalización de productos.
- Uso de spring security: esta implementación daría un nivel más de seguridad a la aplicación mediante autentificaciones y autorizaciones.
- Cifrado de contraseñas: sería un punto de gran importancia que datos críticos no estén accesibles en la base de datos a simple vista.
- Mejorar el custom alert de app: realizar un modal para el control de errores más personalizado ofrecería una mejor experiencia e usuario.
- Realizar tests: otro punto fundamental, testear el código mediante herramientas específicas como JUnit.
- Control de inventario: sería otro punto a revisar muy interesante para llevar la aplicación al siguiente nivel.

9. Bibliografía

1. Abba, I. (2022, 17 de marzo). *¿Cómo usar Tailwind CSS para desarrollar rápidamente sitios web elegantes?* Kinsta. https://kinsta.com/es/blog/tailwind-css/
2. AIX 7.3. (s. f.-b). https://www.ibm.com/docs/es/aix/7.3?topic=monitoring-advantages-java
3. Apple Inc. (s. f.). XCode - Apple Developer. Apple Developer. https://developer.apple.com/xcode/
4. Bailon, G. (2024, septiembre 30). *¿Por qué utilizar Scrum en la gestión de proyectos?* TAKTIC. https://taktic.es/blog/por-que-utilizar-scrum-en-la-gestion-de-proyectos/
5. Chuuck. (2016, 6 mayo). *Ventajas de usar Scrum en tu proyecto*. Platzi. https://platzi.com/blog/ventajas-scrum/
6. Expo. (s.f.). *Expo Router*. https://docs.expo.dev/versions/latest/sdk/router/
7. Ferraris, J. C. (2017, 16 de febrero). *Patrones básicos de Navegación en Apps Móviles.* Medium. https://medium.com/@juancaferraris/patrones-b%C3%A1sicos-de-navegaci%C3%B3n-en-apps-m%C3%B3viles-5b0b160ed1bb
8. FU.DO. (s.f.). *Guía definitiva: ¿Qué sistema POS para restaurantes me conviene?* Requisitos y sugerencias para elegir software gastronómico. https://blog.fu.do/guia-definitiva-que-sistema-pos-para-restaurantes-me-conviene-requisitos-y-sugerencias-para-elegir-software-gastronomico
9. Guadalupe. (2023, 1 de marzo). *¿Qué es Tailwind CSS? Descubre 3 principales beneficios de usarlo.* https://codersfree.com/posts/que-es-tailwind-css-beneficios-usarlo
10. Hernandez, I. (2024, 29 de mayo). *Tailwind Vs. Bootstrap: ¿Qué Framework CSS Necesitas?* DreamHost. https://www.dreamhost.com/blog/es/tailwind-vs-bootstrap-que-framework-css-necesitas/
11. IBM. (s.f.). *¿Qué es Java Spring Boot?* https://www.ibm.com/mx-es/topics/java-spring-boot
12. IntelliJ IDEA – the Leading Java and Kotlin IDE. (2021, 1 junio). JetBrains. https://www.jetbrains.com/idea/

13. Jesús. (2024, 21 de febrero). *Ventajas y desventajas de Java lenguaje de programación.* Tutoriales Dongee. https://www.dongee.com/tutoriales/ventajas-y-desventajas-de-java-lenguaje-de-programacion/
14. Jesús. (2024, 31 de marzo). *MySQL: Características, Ventajas y Desventajas.* https://www.dongee.com/tutoriales/mysql-caracteristicas-ventajas-y-desventajas/
15. José. (2024, 10 mayo). *¿Cuál es el rendimiento de Flutter frente al desarrollo nativo y a React Native?* | Juice Studio. Juice Studio. https://juice-studio.com/cual-es-el-rendimiento-de-flutter-frente-al-desarrollo-nativo-y-a-react-native/
16. José. (2024, 31 de marzo). *MySQL: Características, Ventajas y Desventajas*. https://www.dongee.com/tutoriales/mysql-caracteristicas-ventajas-y-desventajas/
17. Ledesma, E. (s. f.). *Por qué Scrum? Capacitación, Consultoría Dirección Proyectos* TenStep. https://www.tenstep.ec/portal/servicios-de-formacion/por-que-scrum
18. López, M. (2023, 24 febrero). *Metodología SCRUM: principales ventajas* - IMMUNE. IMMUNE Technology Institute. https://immune.institute/blog/metodologia-scrum-caracteristicas/
19. Last.app. (s.f.). *¿Qué debería tener una app de gestión de comandas?* https://www.last.app/recursos/blog/que-deberia-tener-una-app-de-gestion-de-comandas
20. Martins, J. (2024, 15 febrero). *Scrum: conceptos clave y cómo se aplica en la gestión de proyectos Asana.* Asana. https://asana.com/es/resources/what-is-scrum
21. MySQL :: MySQL Workbench. (s. f.). https://www.mysql.com/products/workbench/
22. Pérez, M. (26 de julio de 2024). *Sistema de gestión para comandas: ¡Optimiza tu restaurante!* Yimi Blog. https://blog.yimiglobal.com/sistema-de-gestion-para-comandas-optimiza-tu-restaurante/
23. Raiola Networks. (2024, 31 de enero). *Descubre Tailwind CSS: Ventajas, instalación y conceptos clave*. https://raiolanetworks.com/blog/tailwind-css/

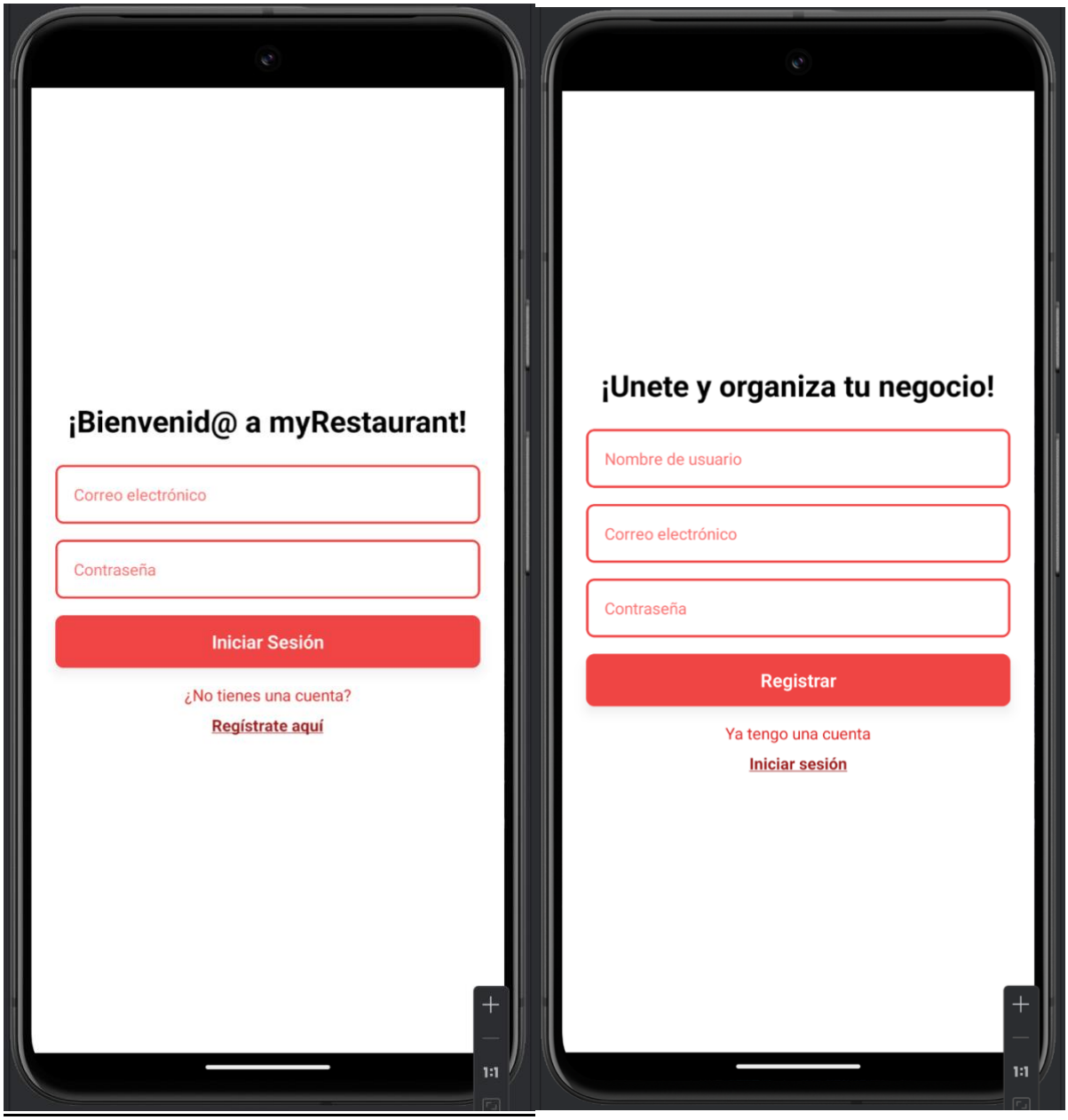
24. Sánchez E. (2024, 18 de abril). *Tailwind CSS: Razones por las que es el framework CSS del futuro.* Medium. https://kikesan.medium.com/tailwind-css-razones-por-las-que-es-el-framework-css-del-futuro-136199ba5a20
25. SLU, A. (2023, 12 septiembre). 👉 *Qué es React Native y su aplicación en el desarrollo de aplicaciones.* ReÁnima Soluciones Digitales. https://www.reanimasoluciones.com/actualidad/322-que-es-react-native-y-su-aplicacion-en-el-desarrollo-de-aplicaciones
26. Soft Restaurant. (s.f.). *Cualidades que debe tener un buen sistema para restaurantes.* https://softrestaurant.com/blog-restaurantero/cualidades-que-debe-tener-un-buen-sistema-para-restaurantes
27. Speed Tecnología. (2022, 31 de enero). *6 Requisitos indispensables de un sistema para restaurantes.* https://www.speedtecnologia.com.mx/2022/01/31/6-requisitos-indispensables-de-un-sistema-para-restaurantes/
28. Supercash. (2022, abril 26). *Cómo organizar las comandas de un restaurante*. Supercash. https://www.supercash.es/blog-hosteleria/comandas-hosteleria-como-organizar/
29. Surra, B. (2024, 12 enero). *Lenguaje de programación TypeScript: definición, ventajas y desventajas.* MyTaskPanel Consulting. https://www.mytaskpanel.com/lenguaje-de-programacion-typescript/
30. Tailwind CSS. (s.f.). Tailwind CSS: Rapidly build modern websites without ever leaving your HTML. https://tailwindcss.com/
31. The collaborative API development platform. (s. f.-b). Insomnia. https://insomnia.rest/
32. Visual Studio Code - Code editing. Redefined. (2021, 3 noviembre). https://code.visualstudio.com/
33. Xcode. (s. f.). Apple Developer. https://developer.apple.com/xcode/

10. Anexos

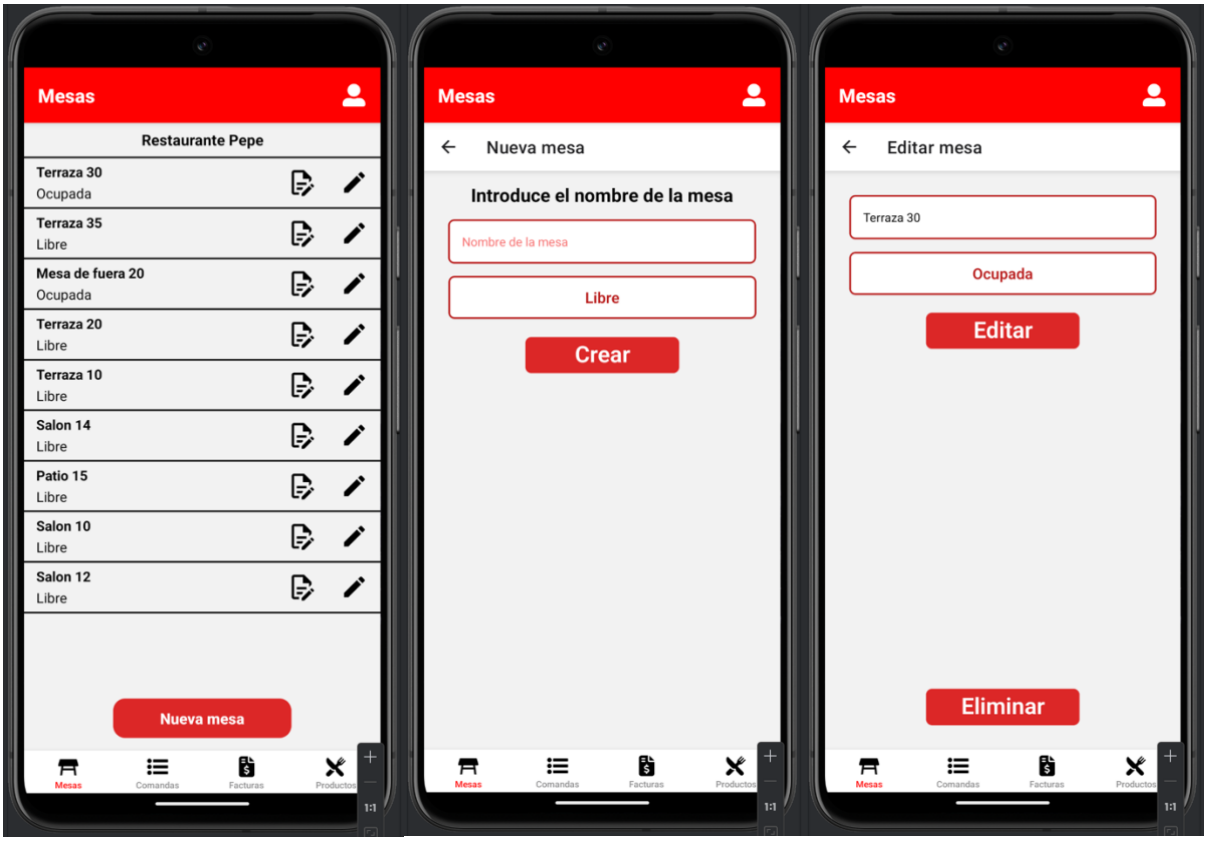
Anexo 1: Repositorio del proyecto en Github

Puedes acceder al repositorio de Github para ver el código completo a través de este enlace: https://github.com/manuelda27999/myRestaurant

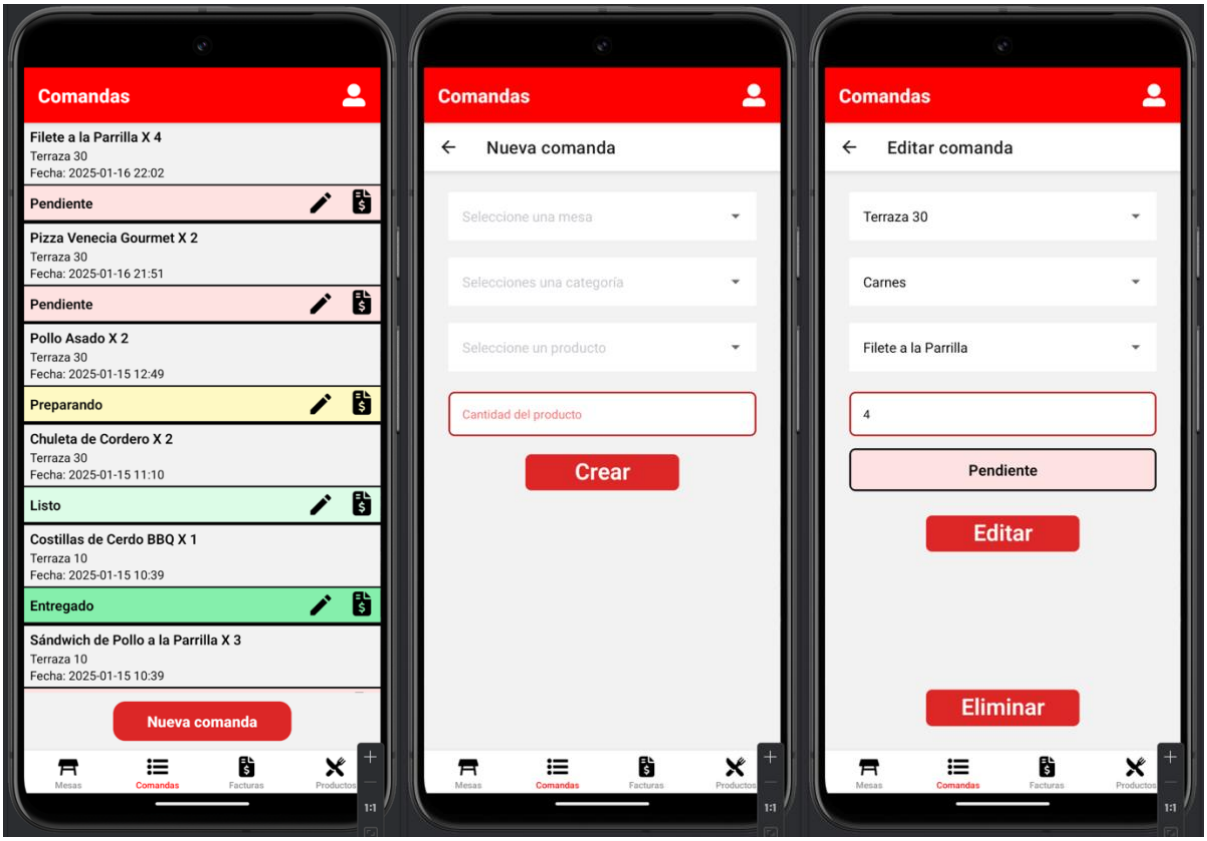
Anexo 2: Imágenes de la pantalla de Login y Register



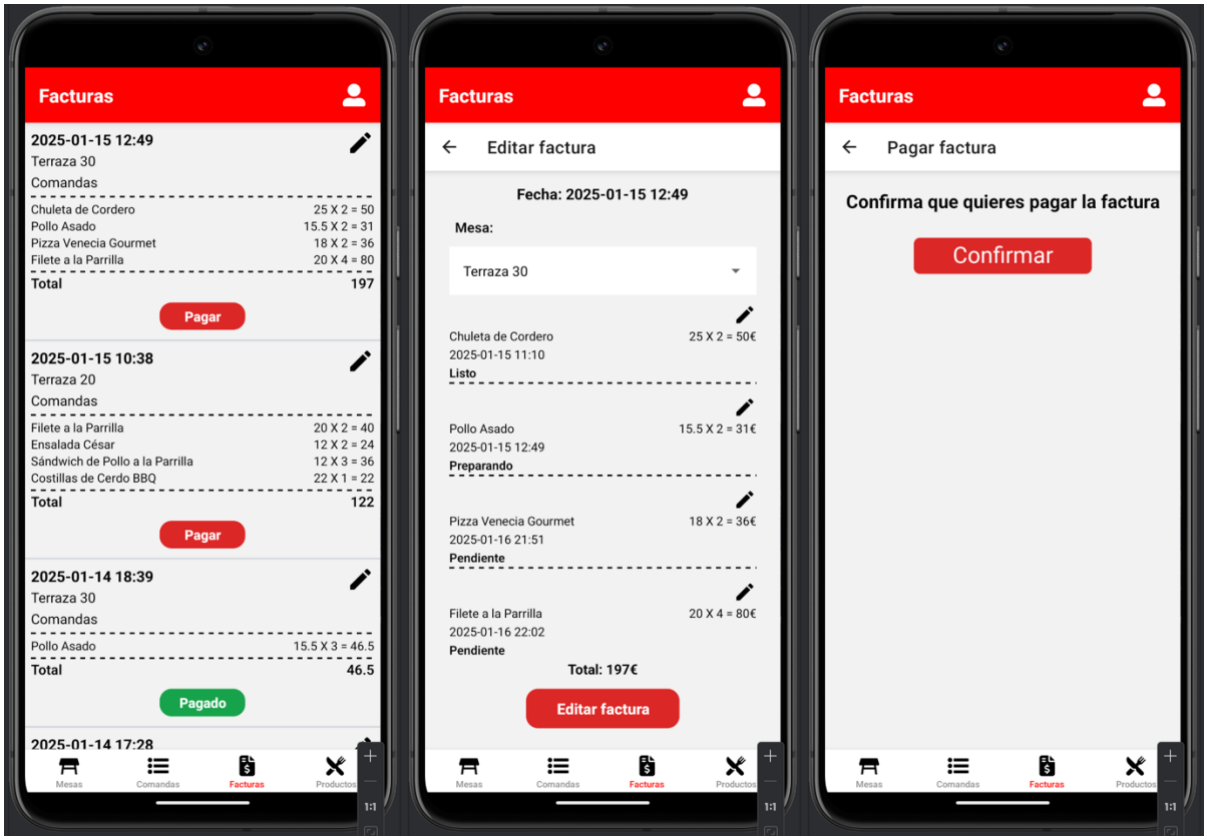
Anexo 3: Imágenes de la pantalla de Home, Mesas (Listado, Crear y Editar mesa)



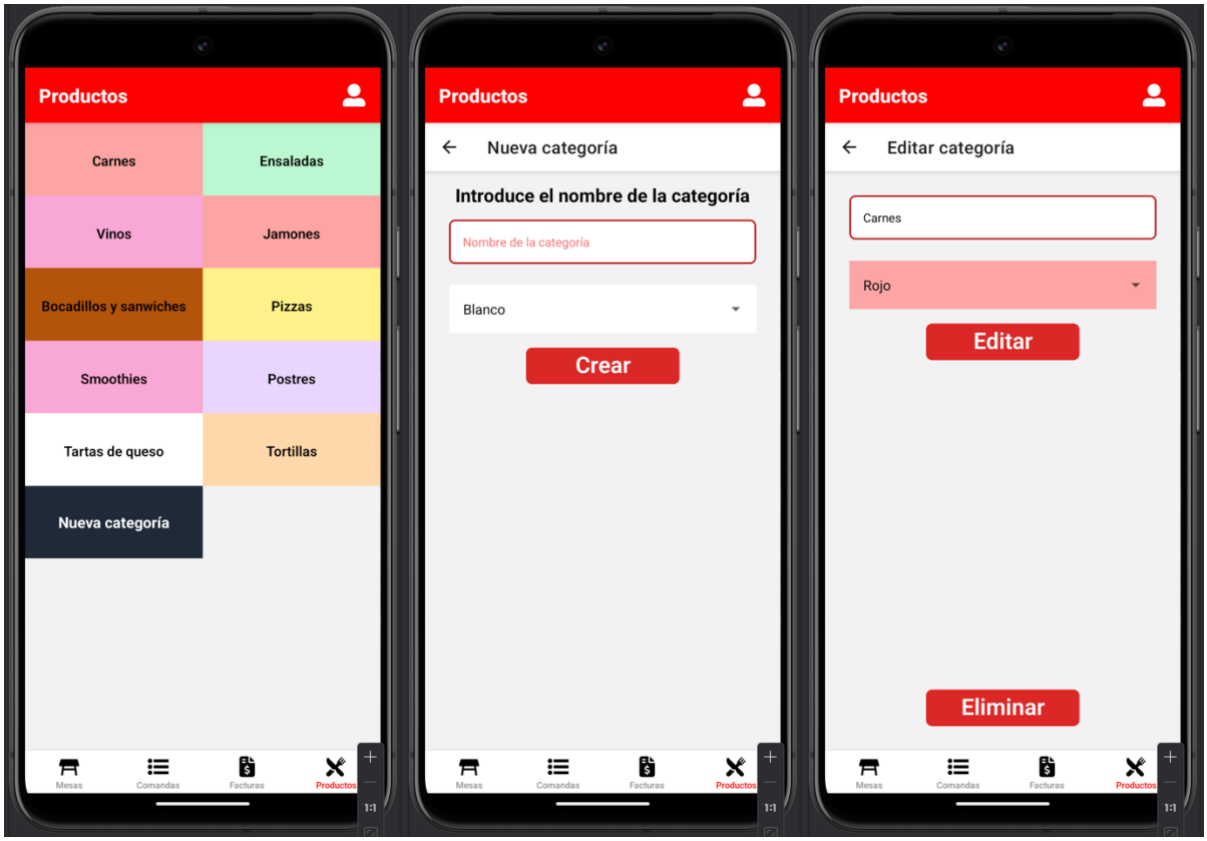
Anexo 4: Imágenes de Home, Comandas (Listado, Crear y Editar comanda)



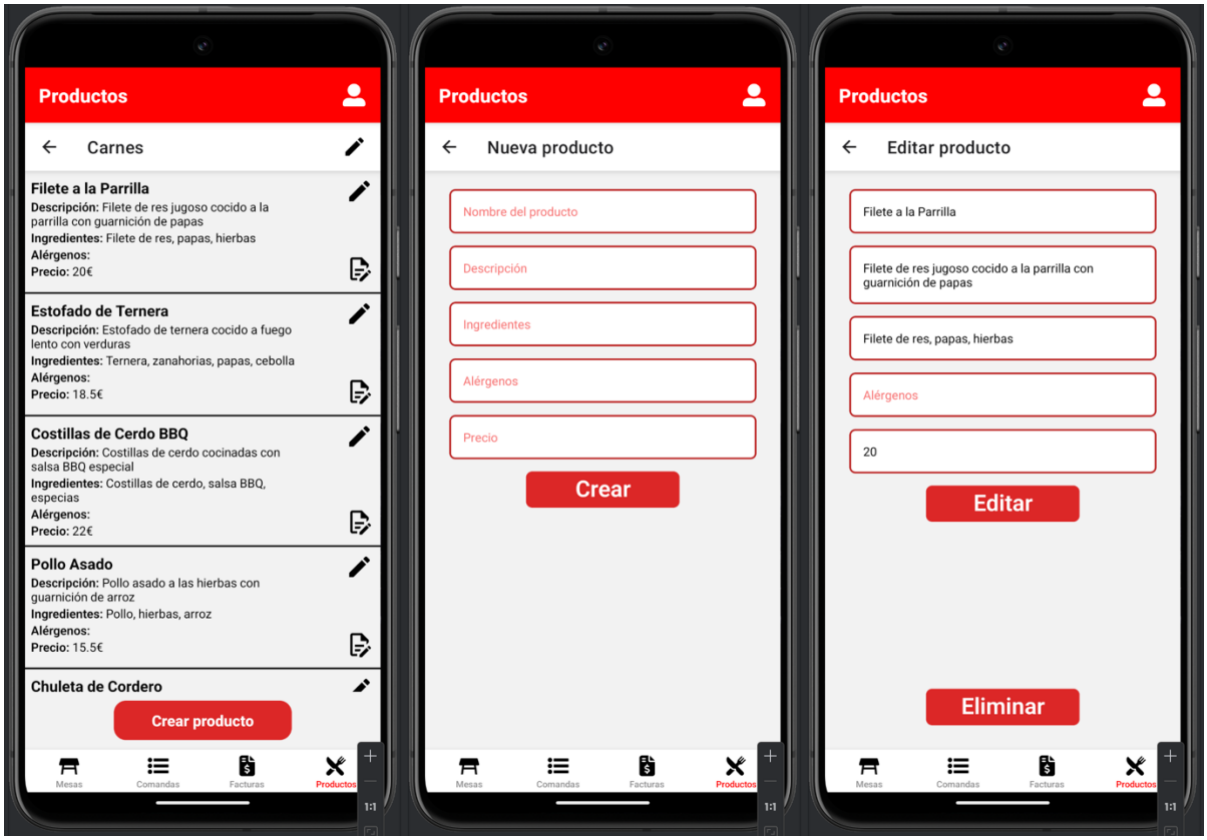
Anexo 5: Imágenes de Home, Facturas (Listado, Editar y Pagar facturas)



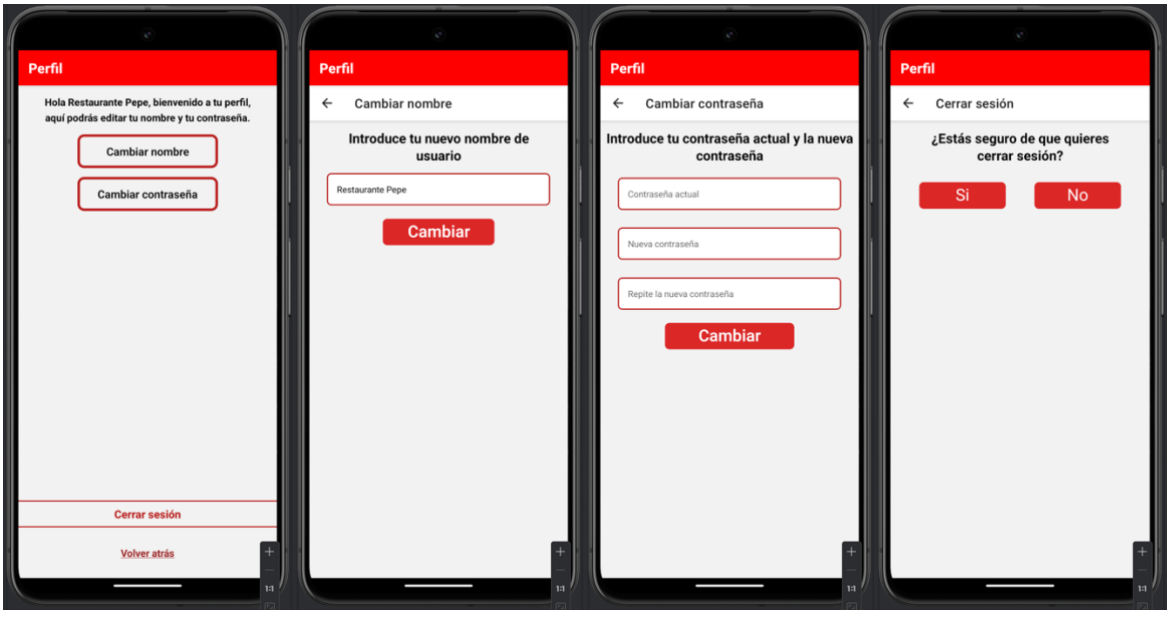
Anexo 6: Imágenes de Home, Productos (Listar, Crear y Editar categorías)



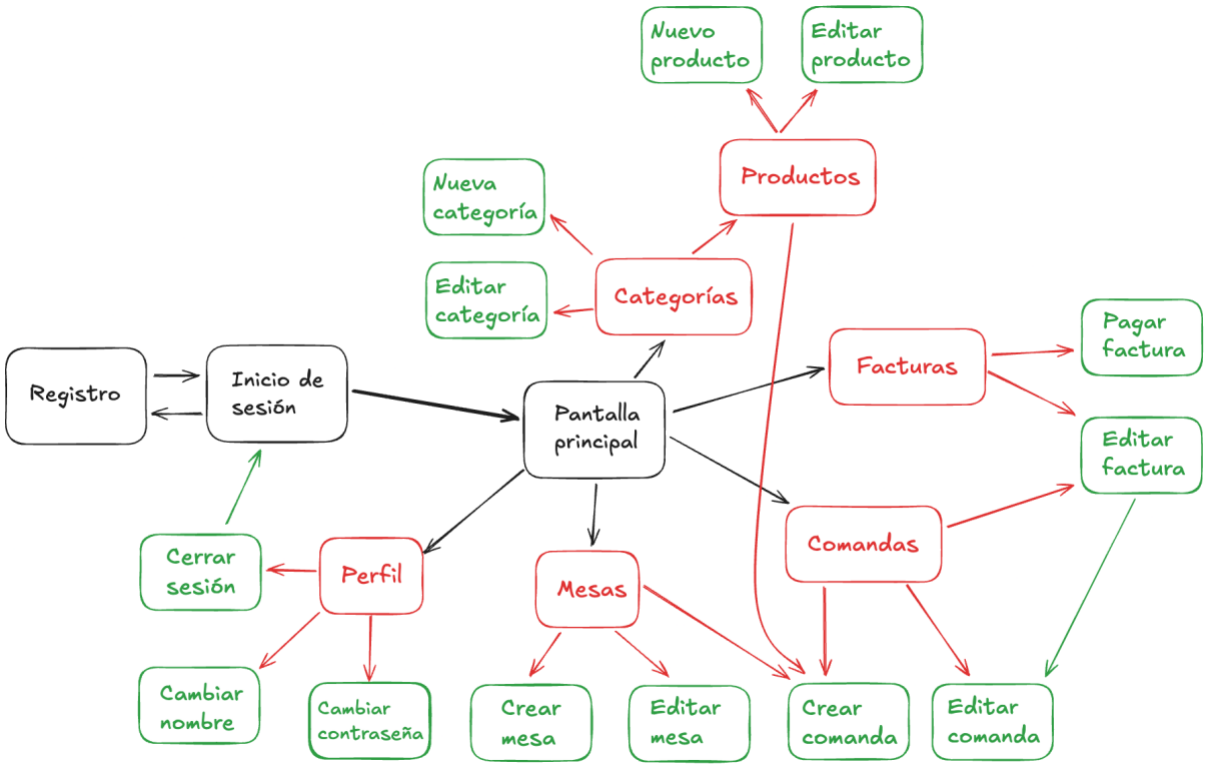
Anexo 7: Imágenes de Home, Productos (Listar, Crear y Editar productos)



Anexo 8: Imágenes de Home, Perfil (Principal, Cambiar nombre, Cambiar contraseña y Cerrar sesión)



Anexo 9: Flujo de navegación



Anexo 10: Tabla de datos de Usuarios

Column	Type	Nullable	Indexes	Extra
user_id	int	NO	PRIMARY	auto_increment
name	varchar(100)	NO		
email	varchar(100)	NO		unique
password	varchar(100)	NO		

Anexo 11: Tabla de datos de Tablas

Column	Type	Nullable	Indexes	Extra
table_id	int	NO	PRIMARY	auto_increment
table_name	varchar(100)	NO		
available	tinyint(1)	NO		
user_id	int	NO	fk_user_id	

Anexo 12: Tabla de datos de Categoría de producto

Column	Type	Nullable	Indexes	Extra
category_id	int	NO	PRIMARY	auto_increment
category_name	varchar(100)	NO		
user_id	int	NO	fk_user_id	
color	enum('RED','GREEN','BLUE','YELLOW','PINK','ORANGE','GRAY','PURPLE','BROWN','WHITE')	YES		

Anexo 13: Tabla de datos de Productos

Column	Type	Nullable	Indexes	Extra
product_id	int	NO	PRIMARY	auto_increment
product_name	varchar(100)	NO		
description	varchar(255)	YES		
ingredients	varchar(255)	YES		
allergens	varchar(100)	YES		
price	decimal(10,2)	NO		
user_id	int	NO	fk_user_id	
category_id	int	NO	fk_category_id	

Anexo 14: Tabla de datos de Comandas

Column	Type	Nullable	Indexes	Extra
order_id	int	NO	PRIMARY	auto_increment
table_id	int	NO	fk_table_id	
product_id	int	NO	fk_product_id	
quantity	int	NO		
order_date	timestamp	NO		DEFAULT_GENERATED
invoice_id	int	NO	fk_invoice_id	
user_id	int	NO	fk_user_id	
status	enum('PENDING','PREPARING','READY','DELIVERED')	NO		

Anexo 15: Tabla de datos de Facturas

Column	Type	Nullable	Indexes	Extra
invoice_id	int	NO	PRIMARY	auto_increment
total	int	NO		
invoice_date	timestamp	NO		DEFAULT_GENERATED
paid	tinyint(1)	NO		
table_id	int	NO	fk_table_id	
user_id	int	NO	fk_user_id	

Anexo 16: Tabla de planificación temporal

Nombre	Duración	Fecha de inicio	Fecha de fin	Horas
Inicio del proyecto				
Búsqueda de ideas y objetivos	2 días	10/10/2024	11/10/2024	8
Selección de los lenguajes y tecnologías	2 días	15/10/2024	16/10/2024	8
Instalación de los softwares necesarios	2 días	18/10/2024	19/10/2024	8
Creación de las carpetas básicas del proyecto	1 día	20/10/2024	20/10/2024	3
Creación del repositorio	1 día	21/10/2024	21/10/2024	4
Diseño de la aplicación e implantación de tecnologías				
Diseño de los modelos de datos	2 días	22/11/2024	23/11/2024	8
Creación de las tablas en la base de datos	2 días	25/11/2024	26/11/2024	8
Creación de la estructura básica de carpetas en la API y de los modelos	2 días	27/11/2024	28/11/2024	8
Diseño visual de la aplicación y del flujo de navegación	1 días	29/11/2024	29/11/2024	4
Creación de la estructura básica de la app y primeros pasos en el flujo de navegación	3 días	4/11/2024	6/11/2024	12
Desarrollo de la aplicación				
Creación de las lógicas y visuales para el registro y autenticación de usuarios	2 días	7/11/2024	8/11/2024	12
Instalación de Tailwind CSS y mejora de los estilos	1 día	9/11/2024	9/11/2024	4
Control de excepciones y errores mediante la creación de una clase (CustomException)	1 día	10/11/2024	10/11/2024	6
Mejora del control de errores en la app	1 día	11/11/2024	11/11/2024	4
Implementación de Expo Router para navegaciones más complejas	4 días	16/11/2024	19/11/2024	20

Implementación de iconos y mejoras visuales	1 día	20/11/2024	20/11/2024	4
Implementación de variables de entorno	1 día	23/11/2024	23/11/2024	4
Creación de la vista de perfil y desarrollo de cierre de sesión	4 días	29/11/2024	02/12/2024	20
Creación de la lógica y de las pantallas para cambiar nombre y contraseña de usuario	2 días	06/12/2024	07/12/2024	12
Creación de la primera versión de los validadores	1 día	08/12/2024	08/12/2024	4
Implementación de EncryptedStorage para guardar el identificador del usuario	1 día	09/12/2024	09/12/2024	6
Implementación de JSON Web Token en la API y actualización de peticiones utilizando el token	1 día	10/12/2024	10/12/2024	8
Creación de la lógica en la API y pantallas en la app para manejo de mesas	3 días	11/12/2024	13/12/2024	15
Creación de la lógica en la API y pantallas en la app para manejo de categorías	8 días	14/12/2024	21/12/2024	24
Refactorización de EncryptedStorage	1 día	22/12/2024	22/12/2024	4
Implementación de Toast en la aplicación	1 día	23/12/2024	23/12/2024	4
Creación de la lógica en la API y pantallas en la app para manejo de productos	4 días	23/12/2024	26/12/2024	20
Creación de la lógica en la API y pantallas en la app para manejo de comandas	4 días	27/12/2024	30/12/2024	20
Creación de la lógica en la API y pantallas en la app para manejo de facturas	6 días	10/01/2024	15/01/2024	36
Mejora de los validadores	1 día	16/01/2024	16/01/2024	3
Movimiento del acceso al perfil del TabBar a la esquina superior derecha	1 día	16/01/2024	16/01/2024	3

Creación de la lógica y cambios en las vistas para manejar los colores de las categorías	1 día	17/01/2024	17/01/2024	6
Fase final				
Revisión general de la aplicación	1 día	18/01/2025	18/01/2025	6
Merge de la rama develop a main	1 día	19/01/2025	19/01/2025	2
Elaborar el documento README del proyecto.	1 día	20/01/2025	20/01/2025	6
Total 294 horas				

Anexo 17: Comandos DDL utilizados para crear las tablas

```
CREATE TABLE `users` (
  `user_id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(100) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `email` (`email`)
)

CREATE TABLE `tables` (
  `table_id` int NOT NULL AUTO_INCREMENT,
  `table_name` varchar(100) DEFAULT NULL,
  `available` tinyint(1) DEFAULT '1',
  `user_id` int DEFAULT NULL,
  PRIMARY KEY (`table_id`),
  UNIQUE KEY `table_name` (`table_name`),
  KEY `user_id` (`user_id`),
  CONSTRAINT `tables_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`)
)
```

```
 CREATE TABLE `products` (
  `product_id` int NOT NULL AUTO_INCREMENT,
  `product_name` varchar(100) NOT NULL,
  `description` varchar(255) DEFAULT NULL,
  `ingredients` varchar(255) DEFAULT NULL,
  `allergens` varchar(100) DEFAULT NULL,
  `price` decimal(10,2) NOT NULL,
  `user_id` int DEFAULT NULL,
  `category_id` int NOT NULL,
  PRIMARY KEY (`product_id`),
  KEY `user_id` (`user_id`),
  KEY `category_id` (`category_id`),
  CONSTRAINT `products_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`),
  CONSTRAINT `products_ibfk_2` FOREIGN KEY (`category_id`) REFERENCES `product_categories` (`category_id`)
)

CREATE TABLE `product_categories` (
  `category_id` int NOT NULL AUTO_INCREMENT,
  `category_name` varchar(100) NOT NULL,
  `user_id` int DEFAULT NULL,
  `color` enum('RED','GREEN','BLUE','YELLOW','PINK','ORANGE', 'GRAY','PURPLE', 'BROWN', 'WHITE' ) DEFAULT NULL,
  PRIMARY KEY (`category_id`),
  KEY `fk_user_id` (`user_id`),
  CONSTRAINT `fk_user_id` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`)
)
```

```
CREATE TABLE `orders` (
  `order_id` int NOT NULL AUTO_INCREMENT,
  `table_id` int DEFAULT NULL,
  `product_id` int DEFAULT NULL,
  `quantity` int NOT NULL,
  `order_date` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `invoice_id` int DEFAULT NULL,
  `user_id` int DEFAULT NULL,
  `status` enum('PENDING','PREPARING','READY','DELIVERED') NOT NULL DEFAULT 'PENDING',
  PRIMARY KEY (`order_id`),
  KEY `table_id` (`table_id`),
  KEY `product_id` (`product_id`),
  KEY `invoice_id` (`invoice_id`),
  KEY `fk_user_id_orders` (`user_id`),
  CONSTRAINT `fk_user_id_orders` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`),
  CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`table_id`) REFERENCES `tables` (`table_id`),
  CONSTRAINT `orders_ibfk_2` FOREIGN KEY (`product_id`) REFERENCES `products` (`product_id`),
  CONSTRAINT `orders_ibfk_3` FOREIGN KEY (`invoice_id`) REFERENCES `invoices` (`invoice_id`)
)
```

```
CREATE TABLE `invoices` (
  `invoice_id` int NOT NULL AUTO_INCREMENT,
  `total` decimal(10,2) NOT NULL,
  `invoice_date` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `paid` tinyint(1) DEFAULT '0',
  `table_id` int DEFAULT NULL,
  `user_id` int DEFAULT NULL,
  PRIMARY KEY (`invoice_id`),
  KEY `table_id` (`table_id`),
  KEY `user_id` (`user_id`),
  CONSTRAINT `invoices_ibfk_1` FOREIGN KEY (`table_id`) REFERENCES `tables` (`table_id`),
  CONSTRAINT `invoices_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`)
)
```

Anexo 18: Diagramas de entidad relación