TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

APLICACIÓN MÓVIL PARA PERSONAL DE SALA DE HOSTELERÍA

Autor: Manuel Jesús Morales Pérez

Grado: Ingeniería Informática

Director: Carlos Javier Ogayar Anguita
Departamento del director: Informática

Fecha: 03/09/2024

CREA

(Página intencionalmente en blanco)

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas y organizaciones que han contribuid a la realización de este proyecto.

En primer lugar, agradezco a mi tutor Carlos, el cual ha sido siempre ese solucionador de dudas que me ha ayudado cuando me he encontrado atrapado el que ha desecho todos esos nudos iniciales, me ha mostrado la línea a seguir y se ha ajustado perfectamente a lo que yo estaba buscando.

También quiero agradecer a mis compañeros de trabajo, quienes me han enseñado muchísimos conceptos que apenas conocía y me han brindado apoyo y estímulos en momentos clave del proyecto. Sus opiniones y perspectivas han enriquecido algunas funcionalidades.

A mi familia, por su aportación económica cuando lo he necesitado y por haberme cuidado cuando lo he necesitado y nunca haber dejado que me rindiese.

Finalmente, agradezco a mi novia Nuria quién me ha estado dándo el apoyo emocional y por haberme dado ese empujón que he necesitado muchos días y cuya colaboración ha sido invaluable para la realización de este proyecto.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Ingeniería Informática
Modalidad	Proyecto de ingeniería
Especialidad (solo TFG)	General
Mención (solo TFG)	General
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Manuel Jesús Morales Pérez
Fecha de asignación	21/03/2023
Descripción corta	Se trata de una aplicación móvil para hostelería, con la que los camareros podrán facilitar su labor ayudándose de la aplicación y poder atender sus mesas asociadas mediante una interfaz muy sencilla para cualquier usuario

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6.

1.1 Introducción

La industria de la hostelería ha experimentado cambios significativos en los últimos años, donde la rapidez y la eficiencia se han vuelto fundamentales. La adopción de tecnologías en este sector se ha vuelto imperativa para enfrentar las crecientes expectativas de los clientes y optimizar las operaciones internas. En este contexto, surge la necesidad de una solución innovadora que mejore la experiencia de los comensales y simplifique las tareas del personal de sala.

Este proyecto trata de una aplicación móvil con la que el personal de hostelería, más concretamente los camareros que sirven las mesas, podrá facilitar su labor apuntando las comandas en la aplicación y recibiendo notificaciones cuando el pedido esté listo, indicando la mesa donde debe ofrecer ese pedido, y también tendrán una vista rápida de las mesas para poder ubicar mejor el número de la mesa y ofrecer el pedido rápidamente, ofreciendo un mejor servicio y mejorando la experiencia de los comensales.

Al implementar esta aplicación, los camareros experimentarán una mejora significativa en la eficiencia y precisión de su trabajo. Los beneficios incluyen la reducción de los tiempos de espera, la optimización de la comunicación entre cocina y sala, y, en última instancia, una experiencia mejorada para los clientes.

1.2 Objetivos del trabajo

El objetivo principal de este trabajo es desarrollar una aplicación móvil que simplifique las tareas del personal de sala en el ámbito de la hostelería. La aplicación busca mejorar la eficiencia operativa, reducir los errores en la toma de pedidos y proporcionar una herramienta integral para la gestión de mesas.

Los objetivos específicos serían implementar una interfaz de usuario intuitiva para la toma de pedidos, garantizando una experiencia sencilla y eficiente para los camareros, desarrollar un sistema de notificaciones en tiempo real para informar al personal sobre el estado de los pedidos y facilitar una coordinación más efectiva y por último integrar una visualización rápida de las mesas asociadas a cada camarero, permitiendo a los camareros acceder rápidamente a gestionar el pedido de la misma y mejorar la rapidez en el servicio.

1.3 Antecedentes y estado del arte

En el mercado actual, existen varias aplicaciones que han sido diseñadas específicamente para la gestión de pedidos en restaurantes y bares. Aplicaciones como **Toast POS**, **Square for Restaurants**, y **Lightspeed Restaurant** son algunas de las más populares y han logrado una amplia adopción en la industria de la hostelería. Estas plataformas permiten a los camareros tomar pedidos de manera eficiente, gestionar pagos, y coordinarse con la cocina en tiempo real. Además, ofrecen funcionalidades avanzadas como el seguimiento del inventario, la creación de informes de ventas, y la integración con servicios de entrega a domicilio.

Sin embargo, aunque estas aplicaciones han sido exitosas en muchos aspectos, también presentan ciertas limitaciones que han sido señaladas por los usuarios. Por ejemplo, algunas pueden ser costosas para pequeños establecimientos, mientras que otras pueden ser demasiado complejas o carecer de personalización para adaptarse a las necesidades específicas de cada negocio. Al analizar estas aplicaciones existentes, el objetivo de este proyecto es identificar tanto sus fortalezas como sus debilidades para desarrollar una solución que no solo cumpla con los estándares actuales de la industria, sino que también sea accesible, intuitiva y adaptable a las necesidades de diferentes tipos de restaurantes.

Este análisis de las aplicaciones comerciales existentes proporciona un contexto valioso que ayudará a guiar el desarrollo de la nueva aplicación, asegurando que se aborden las necesidades reales de los usuarios y que se superen las limitaciones observadas en las soluciones actuales.

1.4 Descripción de la situación de partida

En el contexto actual, la organización cuenta con un sistema de gestión de pedidos que, si bien ha sido funcional en el pasado, presenta limitaciones que dificultan su adaptación a las necesidades tecnológicas modernas. Este proyecto se propone como una acción de cambio que permitirá llevar a la organización desde la situación actual, donde se enfrenta a diversas deficiencias en la eficiencia operativa y en la experiencia del usuario, hacia una situación final mejorada gracias a la implementación de un nuevo sistema.

La situación de partida está condicionada por varios elementos clave que influirán en el desarrollo del trabajo. Se hará un supuesto práctico de situación de un restaurante estándar, sin ningún tipo de solución tecnológica previa:

- **Recursos Humanos**: La plantilla de la organización incluye un equipo de camareros con experiencia en el uso de aplicaciones móviles, aunque su nivel de competencia tecnológica varía considerablemente. Algunos miembros del equipo han mostrado facilidad para adaptarse a nuevas tecnologías, mientras que otros, especialmente aquellos con menos familiaridad con las herramientas digitales, podrían necesitar capacitación adicional para integrarse completamente en el nuevo sistema.

- **Equipamiento Hardware**: Actualmente, la organización cuenta con una infraestructura tecnológica básica, compuesta por dispositivos móviles de distintas marcas y modelos que los camareros utilizan para tomar pedidos. Estos dispositivos varían en capacidad de procesamiento y sistema operativo, lo que genera inconsistencias en el rendimiento de las aplicaciones actuales. Además, el hardware disponible no está estandarizado, lo que podría afectar la uniformidad en la experiencia de usuario.

- **Licencias Software**: En cuanto al software, se utilizan aplicaciones móviles que operan bajo licencias comerciales, algunas de las cuales están próximas a vencer. Esto no solo representa un costo adicional, sino que también limita la capacidad de modificar y adaptar las aplicaciones a las necesidades específicas de la organización. La transición hacia una nueva aplicación ya sea para su comercialización o como software libre, requiere una evaluación cuidadosa de las licencias actuales y futuras para asegurar la sostenibilidad del proyecto.

En resumen, la situación de partida refleja un entorno con desafíos significativos en términos de recursos humanos, equipamiento y licencias, lo que justifica la necesidad de un nuevo sistema que no solo subsane las deficiencias actuales, sino que también sea capaz de escalar y adaptarse a futuras demandas tecnológicas y operativas. En los apartados siguientes se describirá en detalle el entorno actual (1.4.1) y se hará un resumen de las deficiencias y carencias identificadas (1.4.2).

1.4.1 Descripción del entorno actual

El entorno actual en el que se desarrollará e implantará el nuevo sistema es una empresa del sector de la hostelería, específicamente un restaurante que ha venido utilizando herramientas tecnológicas básicas para la gestión de pedidos y la comunicación entre los camareros y la cocina.

- **Organización y Operativa**: El restaurante cuenta con una estructura organizativa tradicional, donde el personal de sala (camareros) es responsable de tomar pedidos, atender a los clientes, y coordinar la entrega de los platos con el personal de cocina. Actualmente, cada camarero utiliza dispositivos móviles proporcionados por la empresa para gestionar los pedidos. Sin embargo, estos dispositivos, que varían en marca y modelo, presentan problemas de compatibilidad y rendimiento, lo que afecta la eficiencia operativa.
- **Sistema de Información**: El sistema de información en uso es un conjunto de aplicaciones móviles y software propietario que permiten a los camareros tomar los pedidos y enviarlos directamente a la cocina. Este sistema, aunque funcional, ha comenzado a mostrar signos de

obsolescencia, con problemas como la lentitud en la sincronización de datos, dificultades en la integración con nuevos dispositivos, y una interfaz de usuario que no está optimizada para la diversidad de usuarios que la utilizan. Además, la falta de estandarización en el hardware ha provocado inconsistencias en el rendimiento de las aplicaciones.

- **Recursos Humanos**: El personal del restaurante está compuesto por camareros con diferentes niveles de familiaridad con la tecnología. Mientras que algunos están cómodos utilizando las aplicaciones móviles actuales, otros, especialmente aquellos con menos experiencia tecnológica, enfrentan desafíos en su uso diario. Esta diversidad en las habilidades tecnológicas del equipo requiere que cualquier nuevo sistema sea intuitivo y fácil de usar para garantizar su adopción efectiva.
- **Impacto del Proyecto**: El nuevo sistema que se pretende implementar tendrá un impacto significativo en varias áreas del restaurante. Primero, mejorará la eficiencia operativa al proporcionar una herramienta más moderna, rápida y compatible con una gama más amplia de dispositivos. En segundo lugar, optimizará la experiencia del usuario tanto para los empleados como para los clientes, reduciendo errores en los pedidos y acelerando el proceso de servicio. Finalmente, permitirá una mayor flexibilidad y escalabilidad, ya que se diseñará con la capacidad de integrar futuras mejoras y adaptarse a nuevas necesidades del negocio.

1.4.2 Resumen de las deficiencias y carencias identificadas

En el desarrollo de este proyecto, se han identificado diversas deficiencias y carencias en el ámbito de la hostelería, especialmente en la gestión de pedidos y en la coordinación entre el personal de sala y la cocina. Estas deficiencias afectan directamente a la eficiencia operativa y la experiencia del cliente. El propósito principal de este trabajo es abordar estas limitaciones y mejorar significativamente la forma en que se llevan a cabo estas operaciones.

Deficiencias identificadas:

- **Procesos manuales y descoordinación:**

Actualmente, las operaciones de toma de pedidos y la comunicación entre el personal de sala y la cocina se llevan a cabo de manera manual, lo que da lugar a procesos lentos y propensos a errores. También existe la falta de un sistema centralizado para gestionar los pedidos, lo que puede generar descoordinación y demoras en el servicio.

- **Limitaciones en la experiencia del usuario:**

Las soluciones tecnológicas existentes en el sector no siempre son amigables para los usuarios, lo que puede generar resistencia, especialmente entre el personal más experimentado.

- **Inclusividad y accesibilidad:**

La brecha generacional y la falta de familiaridad con la tecnología pueden excluir a algunos miembros del personal, especialmente a aquellos que son mayores, de participar plenamente en las operaciones mejoradas.

Para superar estas deficiencias identificadas, el proyecto se enfocará en:

- **Desarrollar una aplicación intuitiva:**

Crear una aplicación móvil fácil de usar que simplifique la toma de pedidos y la gestión de tareas, incluso para aquellos con poca experiencia tecnológica.

- **Establecer un sistema centralizado:**

Implementar un sistema centralizado que mejore la coordinación entre el personal de sala y la cocina, reduciendo tiempos de espera y minimizando errores.

- **Incluir entrenamiento y soporte:**

Proporcionar capacitación y soporte para garantizar que todo el personal, independientemente de su nivel de experiencia, pueda utilizar eficazmente la nueva herramienta.

1.5 Requisitos iniciales

A continuación, se describen los requisitos principales del sistema, tanto funcionales como no funcionales. Siguiendo la norma UNE 157801, los requisitos obligatorios se expresan utilizando la palabra "debe", mientras que las sugerencias o propuestas no obligatorias se expresan mediante el término "debería".

Requisitos Funcionales:

- **Inicio de Sesión:**
    - El sistema debe permitir que los usuarios (camareros) inicien sesión utilizando un nombre de usuario y una contraseña.
    - El sistema debe validar las credenciales del usuario para garantizar que solo el personal autorizado pueda acceder.
    - El sistema debe ofrecer la opción de restablecimiento de contraseñas para aquellos usuarios que las olviden.
- **Gestión de Pedidos:**
    - El sistema debe permitir a los camareros registrar nuevos pedidos, especificando la mesa y los artículos solicitados.
    - El sistema debe permitir la actualización y modificación de pedidos ya registrados.
    - El sistema debe proporcionar la funcionalidad para eliminar pedidos en caso de error o cancelación.
- **Notificaciones en Tiempo Real:**
    - El sistema debe enviar notificaciones en tiempo real a los camareros cuando los pedidos estén listos para su entrega.
    - Las notificaciones deben incluir el número de mesa y los detalles del pedido para asegurar una rápida identificación y entrega.
- **Visualización de Mesas:**
    - El sistema debe ofrecer una visualización rápida y clara de las mesas asignadas a cada camarero.
    - El sistema debe permitir la selección de una mesa para registrar un pedido.
- **Visualización de Productos:**
    - El sistema debe permitir a los camareros visualizar un catálogo de productos disponibles.
    - Los productos deben poder ser seleccionados y añadidos a un pedido en curso.

- **Ver Cuenta:**
    - El sistema debe permitir a los camareros visualizar la cuenta total acumulada de una mesa específica.
- **Revisar Estado de Pedidos:**
    - El sistema debe permitir a los camareros revisar el estado actual de los pedidos en curso, como "en preparación" o "listo".
    - El estado de los pedidos debe actualizarse en tiempo real.
- **Actualizar Estado de Pedidos:**
    - El sistema debe permitir que los cocineros actualicen el estado de un pedido, marcándolo como “en preparación” o “listo”
    - El estado actualizado debe reflejarse en tiempo real para todos los usuarios relevantes.
- **Modificar Pedido en Tiempo Real:**
    - El sistema debe permitir a los camareros modificar un pedido existente, ya sea añadiendo o eliminando productos.
    - Las modificaciones deben guardarse en la base de datos y reflejarse en tiempo real.
- **Cerrar Sesión:**
    - El sistema debe permitir a los camareros cerrar sesión de forma segura.
    - Al cerrar sesión, el sistema debe invalidar el token de autenticación y redirigir al usuario a la pantalla de inicio de sesión.
- **Registro de Usuarios:**
    - El sistema debe permitir que nuevos camareros o cocineros se registren en la aplicación proporcionando un nombre de usuario, un correo electrónico y una contraseña.
    - Los datos del usuario deben ser almacenados de forma segura en la base de datos, y las contraseñas deben estar encriptadas.
- **Base de Datos Centralizada:**

- El sistema debe manejar todos los datos (pedidos, usuarios, etc.) a través de una base de datos centralizada para garantizar la consistencia y disponibilidad.

Requisitos no Funcionales:

- **Seguridad:**
  - El sistema debe utilizar autenticación basada en JWT (JSON Web Token) [1] para garantizar la autenticidad de los usuarios.
  - Los datos sensibles deben estar encriptados tanto en tránsito como en reposo para proteger la información del restaurante y de los clientes.
- **Rendimiento:**
  - El sistema debe ser capaz de manejar hasta 100 usuarios concurrentes sin que se perciba una degradación en el rendimiento.
  - Las notificaciones en tiempo real deben ser entregadas en menos de 2 segundos para asegurar una operación eficiente.
- **Usabilidad:**
  - La interfaz de usuario debe ser intuitiva y fácil de usar, especialmente para aquellos camareros con poca experiencia en tecnología.
- **Mantenibilidad:**
  - El código del sistema debe estar bien documentado y seguir las mejores prácticas de desarrollo, facilitando el mantenimiento y la futura ampliación del sistema.
  - La arquitectura del sistema debe permitir la fácil incorporación de nuevas funcionalidades conforme el restaurante evolucione.

1.6 Alcance

- **Desarrollo de la aplicación móvil:**
  - Entregable: Aplicación móvil funcional y amigable para el personal de sala.
  - Contenido y características: Interfaz intuitiva para la toma de pedidos, sistema de notificaciones en tiempo real, representación visual de la disposición de las mesas.
- **Sistema centralizado de gestión:**
  - Entregable: Plataforma centralizada para la coordinación entre el personal de sala y la cocina.
  - Contenido y características: Registro y seguimiento de pedidos, gestión de tiempos de espera, comunicación eficiente entre departamentos.
- **Capacitación y soporte:**
  - Entregable: Materiales de capacitación y sistema de soporte.
  - Contenido y características: Manuales de usuario y tutoriales en vídeo.
- **Implementación de medidas de accesibilidad:**
  - Entregable: Adaptaciones para garantizar la inclusividad.
  - Contenido y características: Funciones de accesibilidad en la aplicación, materiales de capacitación específicos para abordar brechas generacionales.
- **Informe de proyecto:**
  - Entregable: Documento detallado que describe el proyecto y los resultados.
  - Contenido y características: Descripción de la metodología, análisis de resultados, evaluación de impacto, conclusiones y recomendaciones.

1.7 Hipótesis y restricciones

El TFT se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

Hipótesis:

1. **Disponibilidad de Recursos Tecnológicos:**
    - Se asume que el desarrollador dispondrá de todos los recursos tecnológicos necesarios para llevar a cabo el proyecto, incluyendo un entorno de desarrollo integrado (IDE) [2], acceso a bases de datos, y herramientas de diseño de software. Se supone además que el dispositivo Android utilizado para pruebas es representativo de los que se usan en el entorno real del restaurante.
2. **Conocimientos previos del Desarrollador:**
    - Se parte de la hipótesis de que el desarrollador posee conocimientos suficientes en programación, diseño de software, y desarrollo de aplicaciones móviles en Android, lo que le permitirá completar todas las fases del proyecto sin necesidad de adquirir conocimientos adicionales significativos.
3. **Estabilidad del Entorno de Desarrollo:**
    - Se asume que las tecnologías seleccionadas y el entorno de desarrollo para Android permanecerán estables durante la duración del proyecto, sin actualizaciones o cambios significativos que obliguen a modificaciones sustanciales en el código o la arquitectura del sistema.
4. **Acceso Continuo a la Información y Recursos de Apoyo:**
    - Se asume que el desarrollador tendrá acceso continuo a la información, documentación técnica, y otros recursos necesarios, incluyendo la posibilidad de consultar con tutores y utilizar bibliotecas y repositorios en línea.

Restricciones:

1. **Limitación Temporal:**

- El proyecto está limitado a un máximo de 300 horas de trabajo, que es el tiempo total asignado para un Trabajo de Fin de Grado (TFG) de 12 créditos ECTS. Esta restricción abarca todas las etapas del ciclo de vida del proyecto, exceptuando la fase de mantenimiento.

2. **Restricciones Técnicas:**

   - La aplicación se desarrollará exclusivamente para dispositivos móviles con el sistema operativo Android. Esto implica que todas las soluciones técnicas y decisiones de diseño se orientarán a esta plataforma, descartando la compatibilidad con iOS u otros sistemas operativos en esta fase del proyecto.

3. **Alcance Funcional:**

   - El alcance del proyecto está limitado a las funcionalidades especificadas en los requisitos funcionales y no funcionales definidos previamente. No se contemplarán extensiones o funcionalidades adicionales durante el desarrollo, a menos que sean críticas para la funcionalidad principal.

4. **Restricciones de Seguridad:**

   - El sistema debe cumplir con estándares de seguridad básicos en Android, incluyendo autenticación segura de usuarios y protección de datos sensibles. Sin embargo, debido a las limitaciones de tiempo y recursos, no se implementarán mecanismos avanzados de seguridad más allá de lo especificado en los requisitos no funcionales.

5. **Pruebas y Validación:**

   - Las pruebas se realizarán exclusivamente en un entorno simulado en dispositivos Android, utilizando datos de prueba y escenarios predefinidos por el desarrollador. No se realizarán pruebas en un entorno de producción real, lo que limita la validación del rendimiento y la usabilidad bajo condiciones reales de uso.

1.8 Estudio de alternativas y viabilidad

- **Alternativa 1: Desarrollo de una aplicación móvil independiente:**
  - Ventajas: Mayor control sobre el desarrollo y diseño
  - Desventajas: Posible dificultad en la integración con sistemas existentes.
- **Alternativa 2: Adopción de una solución de terceros:**
  - Ventajas: Rápida implementación y posiblemente menor costo inicial.
  - Desventajas: Menos flexibilidad en el diseño y posibles limitaciones en la personalización.
- **Alternativa 3: Mejora de sistemas internos actuales:**
  - Ventajas: Aprovechamiento de la infraestructura existente.
  - Desventajas: Puede requerir inversiones significativas y posiblemente más complejidad en el desarrollo.
- **Alternativa 4: No implementar una solución tecnológica:**
  - Ventajas: Evitar inversiones y cambios significativos.
  - Desventajas: Mantenimiento de procesos manuales con posibles impactos en la eficiencia y experiencia del cliente.

Alternativa Seleccionada: Desarrollo de una aplicación móvil independiente. Esta decisión se basa en la necesidad de crear una solución altamente personalizada y adaptable a las necesidades específicas del personal de sala y del entorno de la hostelería. Aunque implica desafíos potenciales en la integración, creemos que los beneficios en términos de flexibilidad y control superan estas preocupaciones. La aplicación independiente permitirá una experiencia de usuario más intuitiva y una mayor eficiencia en la coordinación entre el personal de sala y la cocina.

Razones para Descartar Otras Alternativas:

- La adopción de una solución de terceros se descartó debido a la necesidad de una mayor personalización y control sobre el diseño.

- La mejora de los sistemas internos actuales se consideró, pero se descartó debido a posibles complicaciones y la necesidad de inversiones sustanciales.
- La opción de no implementar una solución tecnológica se consideró como última alternativa, pero se descartó debido a la clara necesidad de mejorar la eficiencia y la experiencia del cliente.

1.9 Descripción de la solución propuesta

La solución propuesta consiste en el desarrollo de una aplicación móvil independiente diseñada específicamente para el personal de sala en el sector de hostelería. Esta aplicación tiene como objetivo mejorar la eficiencia en la toma de pedidos, la coordinación entre el personal de sala y la cocina, y optimizar la experiencia global del cliente. A continuación, se enumeran las características clave que destacan la idoneidad de esta solución:

- **Interfaz intuitiva:** Diseño amigable y fácil de usar para garantizar una rápida adaptación por parte del personal, incluso para aquellos con menos familiaridad tecnológica.
- **Sistema de notificaciones en tiempo real:** Funcionalidad de notificaciones para informar al personal sobre el estado de los pedidos, mejorando la coordinación y reduciendo los tiempos de espera.
- **Personalización y adaptabilidad:** Capacidad para adaptarse a las necesidades específicas de cada establecimiento, permitiendo personalizaciones según el tipo de servicio y la disposición del local.
- **Sistema centralizado de gestión:** Implementación de un sistema centralizado que facilita la coordinación entre el personal de sala y la cocina, mejorando la comunicación y reduciendo posibles errores.
- **Capacitación y soporte integrados:** Materiales de capacitación incorporados, incluyendo manuales de usuario y tutoriales, para garantizar una transición suave y una adopción efectiva por parte del personal.
- **Medidas de accesibilidad:** Funciones específicas de accesibilidad para garantizar la inclusión de todos los miembros del personal, independientemente de su nivel de experiencia tecnológica.

1.10 Tecnologías utilizadas

La herramienta utilizada será IntelliJ IDEA ya que me encuentro altamente familiarizado con el IDE, cuenta con una amplia gama de características y funcionalidades, amplio soporte para lenguajes como JavaScript y frameworks como React Native e interfaz de usuario intuitiva y personalizable. Esto quiere decir que para la realización de la aplicación se utilizará React Native con TypeScript. También se utilizará Android Studio herramienta esencial para el desarrollo en React Native para Android.

Justificación de las tecnologías utilizadas:

- **IntelliJ IDEA:** ofrece una interfaz potente y eficiente con sólido soporte para JavaScript y React Native, mejorando la productividad y la calidad del desarrollo.
- **React Native con TypeScript:** permite el desarrollo ágil y eficiente de aplicaciones nativas para Android, optimizando el tiempo y recursos con un código base en JavaScript.
- **Android Studio:** como estándar para el desarrollo Android, proporciona un entorno especializado y perfecta integración con React Native, asegurando un desarrollo eficiente y aprovechando las capacidades de Android.
- **Java con Spring Boot:** este es un framework muy potente para el desarrollo del backend [3] que tendrá la aplicación para manejar todo tipo de consultas, creación de los pedidos, asignación de las mesas, gestión de las notificaciones y gestión de los usuarios ofreciendo una calidad importante en la experiencia como usuario.
- **MongoDB, MongoDB Compass y MongoDB Atlas:**
  - **MongoDB:** Para la gestión de la base de datos, se ha elegido MongoDB [11] [12], una base de datos NoSQL [4] flexible y escalable, que permite un manejo eficiente de datos no estructurados y de gran volumen. MongoDB es ideal para aplicaciones que requieren rapidez y adaptabilidad en el almacenamiento y recuperación de datos.
  - **MongoDB Compass:** MongoDB Compass es la herramienta gráfica utilizada para explorar y manipular los datos dentro de la base de

datos MongoDB. Facilita la visualización, análisis y optimización de las consultas, permitiendo un desarrollo más eficiente.

  - **MongoDB Atlas:** MongoDB Atlas se ha utilizado como la plataforma de base de datos en la nube, proporcionando un servicio gestionado que garantiza alta disponibilidad, escalabilidad automática y seguridad avanzada, sin la necesidad de gestionar la infraestructura subyacente.

- **Firebase:** Se ha integrado Firebase para la gestión de notificaciones en tiempo real. Firebase es una plataforma desarrollada por Google que ofrece servicios backend como la sincronización en tiempo real, autenticación y notificaciones push [5]. Su inclusión permite una comunicación eficiente entre el backend y los dispositivos móviles, asegurando que los camareros reciban notificaciones instantáneas cuando un pedido esté listo.

1.11 Metodología de desarrollo de software

La metodología utilizada será la metodología Ágil con Scrum, la elección de la metodología de desarrollo de software es fundamentar para el éxito del proyecto. Las razones clave por las que se ha optado por esta metodología son:

1. **Flexibilidad y adaptabilidad:** la metodología ágil se adapta bien a proyectos donde los requisitos pueden cambiar o evolucionar durante el desarrollo. Esto es especialmente importante en el desarrollo de aplicaciones, donde las necesidades del usuario pueden varias.
2. **Iteraciones rápidas:** Scrum, como marco dentro de la metodología Ágil, facilita iteraciones rápidas y entregas frecuentes. Esto permite obtener retroalimentación temprana y realizar ajustes continuos según las necesidades del usuario y del negocio.
3. **Enfoque colaborativo:** Scrum fomenta la colaboración estrecha entre los miembros del equipo de desarrollo los stakeholders [6]. La comunicación constante y la participación activa de todas las partes interesadas son esenciales para un desarrollo exitoso.
4. **Control de proyecto efectivo:** Scrum proporciona herramientas para un control efectivo del proyecto, como reuniones diarias cortas (stand-ups),

reuniones de revisión y retrospectivas. Estos elementos permiten una supervisión continua y la identificación temprana de posibles problemas.

5. **Priorización de funcionalidades:** La metodología Ágil con Scrum permite una priorización efectiva de las funcionalidades del producto, asegurando que las características más valiosas para el usuario se implementen primero.
6. **Entregas incrementales:** La entrega incremental de funcionalidades permite obtener beneficios tangibles en etapas tempranas del proyecto, lo que puede ser valioso para el cliente y facilita la validación continua del producto.

Beneficios esperados:

- Mejora de la comunicación y colaboración.
- Adaptación a cambios rápidos en los requisitos.
- Entregas frecuentes y tempranas.
- Mayor control y transparencia en la gestión del proyecto.

La elección de la metodología Ágil con Scrum se alinea con la naturaleza dinámica del desarrollo de aplicaciones, priorizando la adaptabilidad, la colaboración y la entrega incremental para garantizar el éxito del proyecto.

1.12 Análisis de riesgos

El objetivo de este apartado es identificar y evaluar los riesgos que podrían afectar al desarrollo del proyecto, tanto en su planteamiento como durante su ejecución. A continuación, se presenta una lista de riesgos clasificada con una evaluación de sus impactos y los planes de contingencia:

1. **Dependencia de librerías y componentes externos**
    - **Riesgo:** Dificultar en la configuración y uso de la librería **`jwt-decode`** de **`react-native`.**

- **Descripción:** La librería **`jwt-decode`** es utilizada para decodificar tokens JWT en la aplicación. Sin embargo, su configuración puede ser compleja y puede generar problemas de compatibilidad o errores en tiempo de ejecución.
- **Probabilidad de ocurrencia:** Media
- **Impacto:** Alto (puede impedir la autenticación de usuario y la protección de rutas, afectando al núcleo de la aplicación).
- **Plan de contingencia:**
    - **Alternativa:** Evaluar el uso de otras librerías como **`react-native-jose`** o **`jsonwebtoken`** que puedan ofrecer una funcionalidad similar con una configuración más sencilla.
    - **Mitigación:** Realizar pruebas exhaustivas en un entorno de desarrollo antes de la integración en producción y consultar la documentación y foros de desarrolladores para resolver problemas específicos.

2. **Problemas de compatibilidad y dependencias**

- **Riesgo:** Problemas de compatibilidad entre las versiones de **`react-native`** y las librerías utilizadas.
- **Descripción:** Las librerías de **`react-native`** y otros componentes pueden tener dependencias que no sean compatibles entre sí, causando errores o fallos en la aplicación.
- **Probabilidad de ocurrencia:** Alta
- **Impacto:** Medio (puede retrasar el desarrollo y requerir tiempo adicional para resolver conflictos).
- **Plan de contingencia:**
    - **Alternativa:** Mantener actualizadas todas las dependencias y utilizar herramientas como **`npm-check`** o **`yarn outdated`** para identificar y resolver conflictos de versiones.
    - **Mitigación:** Implementar pruebas de integración continua para detectar y solucionar problemas de compatibilidad.

1.13 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

Metodología de Estimación

Para realizar la estimación del tamaño y esfuerzo del proyecto, se utilizarán las siguientes métricas:

- Puntos de Función (Function Points)
- Líneas de Código (LOC - Lines of Code)
- Horas de Trabajo Estimadas

Puntos de Función (Function Points)

Los Puntos de Función (PF) son una métrica estándar utilizada para medir la funcionalidad entregada al usuario. Esta métrica se basa en la cantidad y complejidad de las entradas, salidas, consultas, archivos lógicos y archivos externos del sistema. A continuación, se presenta una estimación revisada de los puntos de función para las principales características del proyecto:

- **Gestión de usuarios:**
  - Entradas: 6
  - Salidas: 4
  - Consultas: 2
  - Archivos lógicos: 1
  - Archivos externos: 1
  - **Total PF:** 14
- **Gestión de pedidos:**
  - Entradas: 8
  - Salidas: 6
  - Consultas: 4
  - Archivos lógicos: 2

    - Archivos externos: 1
    - **Total PF:** 21
- **Gestión de mesas:**
    - Entradas: 6
    - Salidas: 4
    - Consultas: 3
    - Archivos lógicos: 2
    - Archivos externos: 1
    - **Total PF:** 16
- **Autenticación y seguridad:**
    - Entradas: 4
    - Salidas: 3
    - Consultas: 1
    - Archivos lógicos: 1
    - Archivos externos: 1
    - **Total PF:** 10
- **Notificaciones en Tiempo Real:**
    - Entradas: 2
    - Salidas: 4
    - Consultas: 2
    - Archivos lógicos: 1
    - Archivos externos: 1
    - **Total PF:** 10

Total de puntos de función del proyecto: 71

Líneas de Código (LOC – Lines of Code)

Se estima que el desarrollo del proyecto requerirá un número determinado de líneas de código. La estimación se basa en experiencias anteriores y en la complejidad

de las funcionalidades a implementar. Se presenta a continuación una estimación revisada del número de líneas de código por módulo:

- **Gestión de usuarios:** 500 LOC
- **Gestión de pedidos:** 750 LOC
- **Gestión de mesas:** 400 LOC
- **Autenticación y seguridad:** 300 LOC
- **Notificaciones en Tiempo Real:** 300 LOC
- **Total estimado de líneas de código (LOC):** 2250 LOC

Horas de Trabajo Estimadas

La estimación del esfuerzo se basa en la cantidad de horas requeridas para desarrollar cada módulo del proyecto. Se considera un ritmo de trabajo de 5 horas diarias efectivas y se presentan a continuación las estimaciones ajustadas de tiempo para cada módulo:

- **Gestión de usuarios:** 50 horas
- **Gestión de pedidos:** 90 horas
- **Gestión de mesas:** 50 horas
- **Autenticación y seguridad:** 35 horas
- **Notificaciones en Tiempo Real:** 35 horas
- **Total estimado de horas de trabajo:** 260 horas

Horas adicionales para pruebas, documentación y ajustes: 40 horas

Total ajustado de horas de trabajo: 300 horas

1.14 Planificación temporal

Metodología

Para la planificación temporal del proyecto, se ha optado por utilizar una combinación de diagramas de Gantt y principios ágiles de gestión de proyectos. El

cronograma se organiza en fases principales, cada una con hitos específicos y entregables parciales.

Cronograma

Fase 1: Inicio del proyecto (Semana 1)

- **Fecha de inicio:** 11 de mayo
- **Duración:** 1 semana
- **Actividades:**
  - Configuración del entorno de desarrollo
  - Revisión y ajuste del plan de trabajo
  - Reunión inicial con el tutor del proyecto
- **Hitos:**
  - Entorno de desarrollo configurado
  - Plan de trabajo revisado y aprobado

Fase 2: Desarrollo del módulo de gestión de usuarios (Semanas 2-3)

- **Duración:** 2 semanas
- **Actividades:**
  - Implementación de la funcionalidad de gestión de usuarios
  - Pruebas unitarias y de integración
- **Hitos:**
  - Módulo de gestión de usuarios completado
  - Pruebas iniciales realizadas

Fase 3: Desarrollo del módulo de gestión de pedidos (Semanas 4-7)

- **Duración:** 4 semanas
- **Actividades:**
  - Implementación de la funcionalidad de gestión de pedidos
  - Pruebas unitarias y de integración
- **Hitos:**

  - Módulo de gestión de pedidos completados
  - Pruebas iniciales realizadas

Fase 4: Desarrollo del módulo de gestión de mesas (Semanas 8-9)

- **Duración:** 2 semanas
- **Actividades:**
  - Implementación de la funcionalidad de gestión de mesas
  - Pruebas unitarias y de integración
- **Hitos:**
  - Módulo de gestión de mesas completado
  - Pruebas iniciales realizadas

Fase 5: Desarrollo del módulo de avisos y notificaciones (Semanas 10-11)

- **Duración:** 2 semanas
- **Actividades:**
  - Implementación de la funcionalidad de avisos y notificaciones
  - Pruebas unitarias y de integración
- **Hitos:**
  - Módulo de aviso y notificación completado
  - Pruebas iniciales realizadas

Fase 6: Pruebas finales y correcciones (Semanas 12-13)

- **Duración:** 2 semanas
- **Actividades:**
  - Pruebas de integración completa
  - Corrección de errores y ajustes finales
- **Hitos:**
  - Pruebas finales completadas
  - Sistema listo para presentación

Fase 7: Documentación y Entrega (Semana 14)

- **Duración:** 1 semana
- **Actividades:**
  - Redacción de la documentación final
  - Preparación de la presentación del proyecto
- **Hitos:**
  - Documentación final completadas
  - Proyecto presentado

Diagrama de Gantt

A continuación, se presenta un diagrama de Gantt simplificado que ilustra la planificación temporal del proyecto:

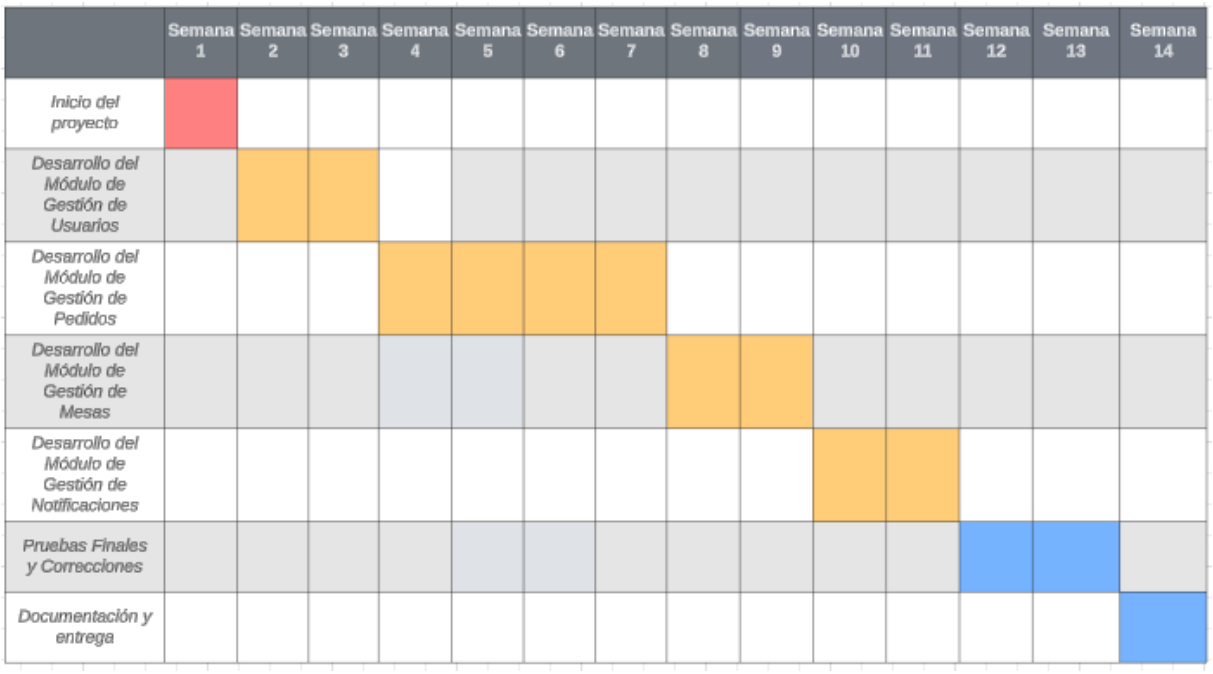

Ilustración 1-1.1 Diagrama de Gant

1.15 Presupuesto

El presupuesto se presenta de manera completa y desglosada, incluyendo los componentes de hardware, software, horas de trabajo y costes indirectos. Además, se realiza una estimación de la amortización parcial de los bienes y servicios utilizados durante el desarrollo del proyecto.

Desglose de costes

- **Componentes de Hardware**
  - Ordenador portátil Asus TUF
  - Precio de compra: 850 EUR
  - Vida útil estimada: 4 años
  - Amortización para 4 meses de uso: 850/4x12 = 70’85 EUR
- **Elementos de Software**
  - IntelliJ IDEA (versión Ultimate, gratuita)
  - Android Studio (gratuito)
  - Git (gratuito)
  - Otros componentes y librerías (gratuito)
- **Horas de trabajo**
  - Perfil: Analista-Programador
  - Convenio: Convenio Colectivo Estatal de Empresas de Consultoría, Servicios Informáticos y Estudios de Mercado
  - Salario bruto anual según convenio (2024): 24.000 EUR
  - Coste mensual: 24000/12 = 2000 EUR
  - Tiempo estimado: 4 meses
  - Coste total: 2000 x 4 = 8.000 EUR
  - Horas estimadas: 300 horas (aproximadamente 75 horas por mes durante 4 meses)
  - Tarifa estimada para trabajo de desarrollo junior: 20 EUR/hora
  - Coste total: 300 x 20 = 6000 EUR
- **Costes indirectos**
  - Utilidades (electricidad, internet, etc.)
  - Estimación mensual: 50 EUR
  - Coste total para 4 meses: 50 x 4 = 200 EUR
  - Gastos de gestión y otros costes indirectos

- 10% del coste total del proyecto (esfuerzo + amortización + utilidades): 0,10 x (8.000 + 70,85 + 200) = 827,09 EUR

Presupuesto Global Desglosado

Concepto	Coste Unitario	Cantidad	Coste Total (EUR)
Hardware			
Ordenador Portátil	70,85	1	70,85
Software			
IntelliJ IDEA	0	1	0
Android Studio	0	1	0
Git	0	1	0
Horas de trabajo			
Desarrollo y pruebas	2.000,00	4	8.000,00
Costes Indirectos			
Utilidades (electricidad, internet)	50,00	4	200,00
Gastos de gestión y otros	10% de coste total	1	827,09
Total			9097’94

Tabla 1-1 Presupuesto

Bases del presupuesto

El presupuesto se ha confeccionado sobre las siguientes bases:

- **Hardware:** Se ha considerado la amortización del ordenador portátil en un periodo de 4 meses, dada su vida útil estimada de 4 años.
- **Software:** Se han utilizado herramientas y librerías de software gratuitas.
- **Horas de trabajo:** El coste salarial se basa en el Convenio Colectivo Estatal de Empresas de Consultoría, Servicios Informáticos y Estudios de Mercado, considerando un perfil de analista-programador.
- **Costes indirectos:** Se ha calculado un 10% del coste total del proyecto para cubrir gastos indirectos, que incluyen utilidades y otros costes relacionados con la ejecución del trabajo.

1.16 Normas y referencias

A continuación, se presentan las normas, reglamentos y referencias de c ualquier tipo que han sido de aplicación en la elaboración del trabajo y/o en la puesta en práctica del mismo.

1.16.1 Disposiciones legales y normas aplicadas

En el desarrollo de este proyecto, se ha considerado el cumplimiento de las normativas básicas de protección de datos personales, en particular:

- **Reglamento General de Protección de Datos (RGPD)**: Aunque la aplicación maneja datos personales limitados (correos electrónicos), se han implementado medidas de seguridad, como la encriptación de contraseñas, para proteger la privacidad y la integridad de los datos de los usuarios, en conformidad con las directrices del RGPD.
- **Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales (LOPDGDD)**: El proyecto también se ha desarrollado siguiendo las disposiciones de la LOPDGDD, asegurando que el tratamiento de los correos electrónicos de los usuarios se realice con las garantías adecuadas de protección y confidencialidad.

1.17 Mecanismos de control de calidad

El aseguramiento de la calidad en la redacción del trabajo implica la verificación de la completitud (falta de omisiones), la integridad de la documentación, así como una redacción clara, concisa y entendible por todos los participantes e interesados en dicho trabajo.

Al estar el trabajo desarrollado por una sola persona y verificado por un/a tutor/a, no es necesario llevar un documento de control de edición y revisión de la documentación. De esta forma, se han utilizado mecanismos básicos para la verificación de la integridad y completitud, que incluyen un control de versiones a nivel de documento y un control sencillo de la trazabilidad de especificaciones y requisitos.

2 DISEÑO INICIAL

El sistema se basará en una arquitectura de cliente-servidor utilizando React Native para el desarrollo de la aplicación móvil y Spring Boot para el backend.

Componentes Principales:

- **Cliente (Aplicación Móvil):** Desarrollada en React Native, compatible con Android.
- **Servidor (Backend):** Implementado en Java utilizando Spring Boot, manejando la lógica de negocio y las solicitudes HTTP.
- **Base de datos:** MongoDB como base de datos NoSQL para almacenar información de usuarios, pedidos, mesas, etc.

Modelos de Diseño:

- **Diseño de la aplicación Móvil:** La aplicación móvil, desarrollada en React Native, estará compuesta por diferentes módulos que permitirán a los camareros gestionar los pedidos, recibir notificaciones y visualizar la disposición de las mesas. Se hará uso de componentes reutilizables para garantizar un desarrollo eficiente y un mantenimiento más sencillo.
- **Diseño del backend:** El backend, desarrollado con Spring Boot, proporcionará una API RESTful que permitirá a la aplicación móvil interactuar con el servidor. Esta API [7] manejará todas las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) necesarias para gestionar los pedidos y la información de las mesas. Spring Boot es elegido por su robustez, flexibilidad y soporte para las aplicaciones empresariales de gran escala.
- **Diseño de una vista web:** Esta vista se ha implementado con el fin de hacer más fácil el reestablecimiento de contraseña mediante correo electrónico. Es un proyecto aparte hecho en react con typescript donde se accede con una ruta y el token que llegará al correo del usuario que ha solicitado el cambio, por supuesto el usuario deberá estar registrado en la aplicación y tener un email válido para que pueda recibir el mensaje.
- **Seguridad y Autenticación:** Para asegurar la autenticidad y seguridad de los datos, se implementará un sistema de autenticación basado en JWT

(JSON Web Token). Este sistema garantizará que solo usuario autorizados pueden acceder a la aplicación y realizar operaciones.

- **Gestión de notificaciones:** El sistema de notificaciones en tiempo real se implementará utilizando servicios como FireBase Cloud Messaging (FCM), permitiendo a los camareros recibir alertas instantáneas cuando los pedidos estén listos.
- **Persistencia de Datos:** MongoDB será utilizado para la persistencia de datos debido a su flexibilidad y capacidad para manejar estructuras de datos dinámicas y no relacionales, lo que es ideal para los requisitos del proyecto.

2.1 Especificaciones del sistema

Este apartado cubre la especificación de requisitos funcionales y no funcionales del producto, así como las historias de usuario iniciales y los casos de uso. Estos elementos servirán de base para el análisis y diseño del sistema.

Requisitos funcionales	Descripción	Requisitos
Inicio de Sesión	Los camareros deben poder iniciar sesión en la aplicación utilizando un nombre de usuario y una contraseña	RF1 La aplicación debe validad las credenciales del usuario RF2 La aplicación debe permitir el restablecimiento de contraseñas olvidadas
Gestión de Pedidos	Los camareros deben poder registrar, actualizar y eliminar pedidos	RF1 La aplicación debe permitir la creación de nuevos pedidos con detalles de la mesa y los artículos RF2 La aplicación debe permitir la modificación de pedidos existentes RF3 La aplicación debe permitir la eliminación de pedidos

Notificaciones en Tiempo Real	Los camareros deben recibir notificaciones cuando los pedidos estén listos	RF1 La aplicación debe recibir notificaciones en tiempo real cuando un pedido esté listo RF2 Las notificaciones deben incluir el número de mesa y los detalles del pedido
Visualización de Mesas	Los camareros deben poder visualizar sus las mesas en el restaurante	RF1 La aplicación debe permitir la selección de una mesa desde la vista para registrar un pedido
Visualización de Productos	Los camareros deben poder visualizar un catálogo de productos disponibles para pedidos	RF1 La aplicación debe mostrar una lista de productos RF2 Los productos deben poder ser seleccionador y añadidos a un pedido
Ver cuenta	Los camareros deben poder ver la cuenta total de una mesa específica	RF1 La aplicación debe calcular y mostrar el total acumulado de los pedidos de una mesa
Revisar Estado de pedidos	Los usuarios deben poder revisar el estado de los pedidos en curso	RF1 La aplicación debe mostrar el estado actual de cada pedido, como “en preparación” o “listo” RF2 La aplicación debe actualizar automáticamente el estado del pedido en tiempo real
Actualizar estado de pedidos	Los cocineros deben poder actualizar el estado de los pedidos	RF1 La aplicación debe permitir que los cocineros cambien el estado de pedido de “pendiente” a “en preparación” y de “en preparación” a “listo” RF2 El estado del pedido debe reflejarse en tiempo real para otros usuarios

Modificar Pedido en tiempo real	Los camareros deben poder modificar un pedido ya registrado antes de que se complete	RF1 La aplicación debe permitir añadir o eliminar productos de un pedido existente RF2 Las modificaciones deben guardarse en la base de datos y reflejarse en tiempo real
Cerrar sesión	Los usuarios deben poder cerrar sesión de la aplicación de forma segura	RF1 La aplicación debe invalidar el token de autenticación al cerrar sesión RF2 Los usuarios deben ser redirigidos a la pantalla de inicio de sesión al cerrar sesión
Recuperar contraseña	Los usuarios deben poder restablecer su contraseña de forma segura si la olvidan	RF1 La aplicación debe enviar un correo al usuario si este ha solicitado restablecer contraseña RF2 La aplicación debe actualizar la base de datos con la contraseña nueva sustituyendo la antigua
Registro de usuarios	Los nuevos usuarios deben poder registrarse en la aplicación proporcionando un nombre de usuario que no exista, un correo electrónico y una contraseña	RF1 Los usuarios deben poder registrarse mediante un formulario de registro RF2 Los datos del usuario deben ser almacenados de forma segura en la base de datos RF3 Las contraseñas deben ser encriptadas antes de ser almacenadas
Base de datos centralizada	La aplicación debe manejar una base de datos centralizada para todas las operaciones	RF1 Los datos deben ser almacenados y gestionados en una base de datos centralizada en la nube RF2 La base de datos debe permitir el acceso concurrente de múltiples usuarios con consistencia

Tabla 2-1 Requisitos funcionales

Requisitos no funcionales	Descripción
Seguridad	La aplicación debe utilizar autenticación basada en JWT para asegurar la autenticidad de los usuarios Los datos sensibles deben ser encriptados tanto en tránsito como en reposo
Rendimiento	La aplicación debe ser capaz de manejar hasta 100 usuarios concurrentes sin degradación perceptible del rendimiento Las notificaciones en tiempo real deber ser entregadas en menos de 2 segundos
Usabilidad	La interfaz de usuario deber ser intuitiva y fácil de usar, incluso para camareros con poca experiencia en tecnología La aplicación del sistema debe permitir la fácil incorporación de nuevas funcionalidades
Mantenibilidad	El código debe estar bien documentado y seguir las mejores prácticas de desarrollo La arquitectura del sistema debe permitir la fácil incorporación de nuevas funcionalidades

Tabla 2-2 Requisitos no funcionales

Título	Descripción	Criterios de aceptación
Registro de Usuario	Como nuevo usuario, quiero poder registrarme en la aplicación para acceder a las funcionalidades de gestión de pedidos	Debo poder introducir mi nombre de usuario, correo electrónico y contraseña Debo recibir un mensaje de confirmación al registrarme exitosamente

Inicio de sesión	Como camarero, quiero poder iniciar sesión en la aplicación para acceder a las funcionalidades de gestión de pedidos	Debo poder introducir mi nombre de usuario y contraseña Debo recibir un mensaje de error si mis credenciales son incorrectas
Recuperar contraseña	Como usuario, quiero poder recuperar mi contraseña en caso de olvidarla para poder acceder nuevamente a la aplicación	Debo poder solicitar un enlace de recuperación de contraseña Debo poder restablecer mi contraseña utilizando el enlace recibido por correo electrónico
Cerrar sesión	Como usuario, quiero poder cerrar sesión en la aplicación de manera segura cuando termine mi turno	Debo poder cerrar sesión desde cualquier pantalla de la aplicación Debo ser redirigido a la pantalla de inicio de sesión al cerrar sesión
Elegir mesa	Como camarero, quiero poder seleccionar una mesa desde la vista de mis mesas asignadas para registrar un pedido para esa mesa	Debo poder ver las mesas que tengo asignadas Debo poder seleccionar una mesa específica para iniciar el registro de un pedido
Registro de pedido	Como camarero, quiero poder registrar un nuevo pedido para una mesa específica	Debo poder seleccionar una mesa desde la vista Debo poder agregar detalles al pedido y guardarlo

Visualización de productos	Como camarero, quiero poder visualizar los productos disponibles para seleccionar y añadir a un pedido	Debo poder ver una lista de productos con su nombre Debo poder seleccionar productos desde la lista y añadirlos al pedido
Ver cuenta	Como camarero, quiero poder ver la cuenta total de una mesa específica para presentarla al cliente	Debo poder ver el total acumulado de los pedidos realizados por una mesa Debo poder generar una cuenta para el cliente desde la aplicación
Revisar estado de pedidos	Como camarero, quiero poder revisar el estado actual de mis pedidos para mantener informados a mis clientes	Debo poder ver el estado de cada pedido como “en preparación”, “listo” o “pendiente” Debo recibir una notificación en tiempo real si alguno de los pedidos pasa a “listo”
Actualizar estado de pedidos	Como cocinero, quiero poder actualizar el estado de los pedidos para reflejar su progreso	Debo poder cambiar el estado del pedido de “pendiente” a “en preparación” y de “en preparación” a “listo” El estado del pedido debe actualizarse en tiempo real y ser visible para otros usuarios
Modificar pedido en tiempo real	Como camarero, quiero poder modificar un pedido ya registrado antes de que se complete	Debo poder eliminar o añadir productos de un pedido en curso Las modificaciones deben reflejarse en la cuenta final y en la base de datos en tiempo real
Notificaciones de pedidos	Como camarero, quiero recibir notificaciones en tiempo real cuando un pedido esté listo para	Debo recibir una notificación cuando el pedido esté listo

	poder entregarlo rápidamente	La notificación debe incluir el número de mesa y los detalles del pedido
Base de datos centralizada	Como administrador del sistema, quiero asegurarme de que todos los datos se almacenen en una base de datos centralizada para su gestión y seguridad	Debo poder acceder a todos los datos desde una base de datos centralizada La base de datos debe soportar acceso concurrente sin pérdida de datos

Tabla 2-3 Historias de usuario

Título	Actores	Descripción	Flujo principal
Registro de usuario	Camarero y cocinero	El usuario se registra en la aplicación proporcionando su nombre de usuario, correo electrónico y contraseña	El camarero ingresa su nombre de usuario, correo electrónico y contraseña La aplicación valida los datos y almacena la información en la base de datos Si el nombre de usuario ya existe, la aplicación muestra un mensaje de error
Iniciar Sesión	Camarero y cocinero	El usuario inicia sesión en la aplicación utilizando sus credenciales	El usuario ingresa su nombre de usuario y contraseña La aplicación valida las credenciales Si las credenciales son correctas, el camarero accede a la pantalla principal

Recuperar contraseña	Camarero y cocinero	El usuario solicita la recuperación de su contraseña en caso de haberla olvidado	El camarero ingresa su nombre de usuario La aplicación envía un enlace de recuperación al correo vinculado a ese nombre de usuario Si el nombre de usuario no está registrado, la aplicación muestra un mensaje de error
Cerrar sesión	Camarero y cocinero	El usuario cierra sesión en la aplicación de manera segura	El usuario selecciona la opción de cerrar sesión La aplicación invalida el token de autenticación y redirige al usuario a la pantalla de inicio de sesión
Elegir mesa	Camarero	El camarero selecciona una mesa desde el mapa interactivo para registrar un pedido	El camarero visualiza sus mesas asignadas El camarero selecciona la mesa deseada La aplicación muestra la opción de registrar un nuevo pedido para la mesa seleccionada o ver la cuenta de esa mesa
Registrar pedido	Camarero	El camarero registra un nuevo pedido para una mesa específica	El camarero selecciona una mesa desde la vista

			El camarero ingresa los detalles del pedido La aplicación guarda el pedido en la base de datos Si hay un error al guardar el pedido, la aplicación muestra un mensaje de error
Visualización de productos	Camarero	El camarero visualiza un catálogo de productos disponibles para añadir a un pedido	El camarero accede a la lista de productos El camarero selecciona los productos deseados La aplicación añade los productos seleccionados al pedido en curso
Ver cuenta	Camarero	El camarero visualiza la cuenta total de una mesa específica	El camarero selecciona la opción ver cuenta para una mesa La aplicación muestra el total acumulado de los pedidos de la mesa El camarero puede generar e imprimir la cuenta para el cliente

Revisar estado de pedidos	Camarero y cocinero	El usuario revisa el estado actual de los pedidos en curso	El usuario accede a la lista de pedidos en curso La aplicación muestra el estado de cada pedido El estado se actualiza en tiempo real
Actualizar estado de pedidos	Cocinero	El cocinero actualiza el estado de los pedidos, marcándolos como “en preparación” o “listo”	El cocinero selecciona un pedido con estado “pendiente” y lo pasa a “en preparación” o si está “en preparación” lo pasa a “listo” La aplicación actualiza el estado del pedido en la base de datos
Modificar pedido en tiempo real	Camarero	El camarero modifica un pedido en curso, añadiendo o eliminando productos	El camarero selecciona el pedido en la lista de pedidos El camarero añade o elimina productos La aplicación guarda los cambios en la base de datos
Recibir notificación	Camarero	El camarero recibe una notificación en tiempo real cuando un pedido esté listo	El sistema envía una notificación al camarero cuando el pedido esté listo El camarero recibe la notificación y

			visualiza los detalles del pedido Si la notificación no se recibe, el camarero puede consultar manualmente el estado del pedido
Base de datos centralizada	Aplicación	La aplicación gestiona y almacena todos los datos en una base de datos centralizada	La aplicación almacena todos los pedidos, usuarios y estados en la base de datos centralizada La base de datos asegura la consistencia y disponibilidad de los datos en todo momento

Tabla 2-4 Casos de uso

2.2 Análisis y diseño del sistema

En este apartado se presenta el análisis y diseño del sistema desarrollado para el proyecto. Siguiendo una metodología ágil, se ha estructurado el contenido en varias subsecciones que abarcan los aspectos fundamentales del diseño del sistema, incluyendo la base de datos, los diagramas de clases, los diagramas de caso de uso, los diagramas de secuencia y el diseño de la interfaz de usuario. Cada subsección ofrece una descripción detallada del enfoque y las decisiones de diseño tomadas, con el objetivo de proporcionar una visión clara y comprensible del sistema implementado.

2.2.1 Diseño de base de datos

El diseño de la base de datos se ha llevado a cabo utilizando MongoDB, una base de datos NoSQL que permite el almacenamiento y manejo eficiente de grandes volúmenes de datos no estructurados, MongoDB ha sido elegido por su flexibilidad, escalabilidad y su capacidad para manejar documentos JSON [8], lo que es ideal para la estructura dinámica de la aplicación desarrollada.

Esquema de la Base de Datos

La base de datos está organizada en varias colecciones, cada una representando un conjunto de documentos con características comunes. Las principales colecciones diseñadas son:

- **Usuarios**: Esta colección almacena los datos de los usuarios del sistema, en este caso los camareros y cocineros.
    - **'id'**: Identificador único del usuario.
    - **'username'**: Nombre de usuario único usado para el login
    - **'email'**: Correo electrónico del usuario (se puede repetir)
    - **'password'**: Contraseña encriptada del usuario
    - **'mesasAsignadas'**: Lista de identificadores de las mesas pertenecientes al camarero (el cocinero lo tendrá vacío)
    - **'role'**: Rol del usuario, en este caso, 'CAMARERO' o 'COCINERO'
- **Mesas**: Esta colección almacena los datos de las mesas del restaurante.
    - **'id'**: Identificador único de la mesa
    - **'numero'**: Número único de la mesa
    - **'userId'**: Id del usuario que tiene asignada la mesa
- **Pedidos**: Esta colección gestiona la información relacionada con los pedidos realizados.
    - **'id'**: Identificador único del pedido
    - **'mesa'**: Número de la mesa a la que pertenece el pedido
    - **'items'**: Lista de productos que tiene el pedido
    - **'estado'**: Estado actual del pedido (ej. Pendiente, en preparación, listo)
    - **'timestamp'**: Fecha y hora a la que se realizó el pedido
- **Notificaciones**: Esta colección se encarga de las notificaciones envidadas a los camareros

- **‘id’**: Identificador único de la notificación
- **‘titulo’**: En nuestro caso siempre será ‘Pedido Listo’
- **‘mensaje’**: Mensaje indicado el número de la mesa a la que pertenece el pedido
- **‘timestamp’**: Fecha y hora a la que se realizó la notificación, para posteriormente ordenarla en el modal de notificaciones
- **‘userId’**: Id del usuario al que pertenece la notificación
- **‘leida’**: Identificador de si la notificación ha sido leída

- **Cuentas**: Esta colección se encarga de almacenar la suma de los pedidos de cada mesa
  - **‘id’**: Identificador único de la cuenta
  - **‘timestamp’**: Fecha y hora a la que se realizó el pago
  - **‘total’**: Número en euros que indica el coste de la cuenta

2.2.2 Storyboard Inicial

Mediante Axure, un potente software para crear wireframes [9] incluso mockups [10] obtenido de la web [6], se ha realizado un pequeño flujo de cómo queremos que sea la aplicación visualmente y que posteriormente irá modificándose a lo largo de las iteraciones del proyecto. A continuación, se muestra el storyboard [11] inicial de la aplicación:

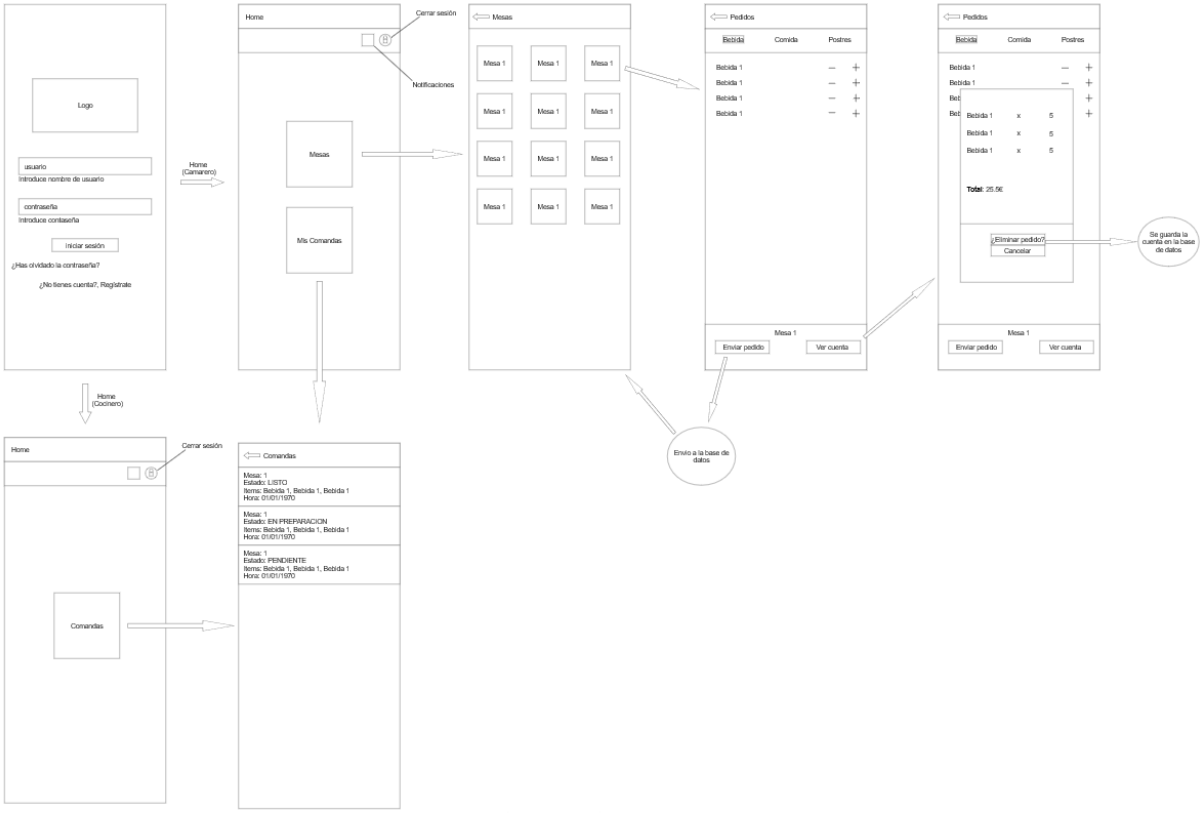

Ilustración 2-1. Stoyboard inicial

2.2.3 Diagrama de casos de uso

Para este apartado se ha creído conveniente el uso de Visual Paradigm con la licencia ofrecida por la escuela, ya que contiene una sencilla manera de hacer estos diagramas. Con estos diagramas será fácil de representar junto a tablas que acciones son necesarias para el correcto funcionamiento de la aplicación.

2.2.3.1 Caso de uso – 01: Registro

Ilustración 2-2 CU-01: Registro

Caso de uso - 01	Registro	
Actores	Camarero	
Descripción	El camarero crea una cuenta en la aplicación	
Precondición	El camarero no se ha registrado previamente	
Secuencia	**Paso**	**Acción**
	1	Selecciona la opción de registrarse en el inicio
	2	Rellena los datos
	3	El servidor valida que la información sea válida y guarda los datos en la base de datos
Postcondición	El camarero queda registrado en la base de datos y podrá iniciar sesión	

Tabla 2-5 CU-01: Registro

2.2.3.2 Caso de uso – 02: Iniciar Sesión

Ilustración 2-3 CU-02: Inicio de sesión

Caso de uso - 02	Iniciar Sesión	
Actores	Camarero o cocinero	
Descripción	El usuario accede a la aplicación con su usuario y contraseña	
Precondición	El usuario está registrado en la base de datos	
Secuencia	**Paso**	**Acción**
	1	El usuario inicia la aplicación
	2	El usuario ingresa su usuario y contraseña en la pantalla de inicio
	3	El servidor verifica que las credenciales son correctas

	4	Si son correctas, accede a la vista 'Home' y al camarero en concreto se le asignan unas mesas
Postcondición	El usuario entra en la aplicación y puede empezar a usarla	

Tabla 2-6 CU-02: Iniciar Sesión

2.2.3.3 Caso de uso – 03: Recuperar contraseña

Ilustración 2-4 CU-03: Recuperar contraseña

Caso de uso - 03	Recuperar Contraseña	
Actores	Camarero o cocinero	
Descripción	El usuario restablece su contraseña	
Precondición	El usuario no recuera su contraseña	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa el botón 'he olvidado mi contraseña'
	2	El usuario ingresa su nombre de usuario
	3	El servidor envía un correo al usuario con el enlace acompañado de su token
	4	El usuario sigue las instrucciones del correo e ingresa una nueva contraseña
Postcondición	Se actualiza la contraseña en la base de datos y el usuario puede ingresar en la aplicación	

Tabla 2-7 CU-03: Recuperar contraseña

2.2.3.4 Caso de uso – 04: Cerrar Sesión

Ilustración 2-5 CU-04: Cerrar sesión

Caso de uso – 04	Cerrar Sesión	
Actores	Camarero o cocinero	
Descripción	El usuario cierra sesión en la aplicación	
Precondición	El usuario ha iniciado sesión	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa el botón del perfil que aparece en la pantalla ‘Home’
	2	La sesión se cierra y si el usuario es camarero se desvinculan las mesas que tenía asignadas
Postcondición	Se cierra la sesión y no se puede volver a usar la aplicación hasta iniciar sesión de nuevo	

Tabla 2-8 CU-04: Cerrar sesión

2.2.3.5 Caso de uso – 05: Elegir mesa

Ilustración 2-6 CU-05: Elegir Mesa

Caso de uso – 05	Elegir mesa	
Actores	Camarero	
Descripción	El usuario selecciona la mesa que quiere atender	
Precondición	El usuario ha iniciado sesión	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa el botón ‘Mesas’ en la pantalla ‘Home’
	2	El usuario pulsa sobre una de sus mesas asignadas para atenderla
Postcondición	Aparece la pantalla para realizar el pedido para la mesa preseleccionada	

Tabla 2-9 CU-05: Elegir mesa

2.2.3.6 Caso de uso – 06: Registrar comanda

Ilustración 2-7 CU-06: Registrar comanda

Caso de uso – 06	Registrar comanda	
Actores	Camarero	
Descripción	El usuario registra un pedido	
Precondición	El usuario ha iniciado sesión y ha seleccionado una mesa	
Secuencia	**Paso**	**Acción**
	1	El usuario selecciona la mesa que quiere atender
	2	El usuario selecciona los productos que el cliente ha pedido de forma manual
	3	El servidor almacena el pedido en la base de datos asociándolo a la mesa
Postcondición	Se registra la comanda y el pedido se podrá visualizar desde 'Mis comandas'	

Tabla 2-10 CU-06: Registrar comanda

2.2.3.7 Caso de uso – 07: Visualizar productos

Ilustración 2-8 CU-07: Visualizar productos

Caso de uso – 07	Visualizar productos	
Actores	Camarero	
Descripción	El usuario podrá visualizar los productos a ofrecer viendo su nombre, precio y número de elementos que lleva	
Precondición	El usuario ha iniciado sesión y ha seleccionado una mesa	
Secuencia	**Paso**	**Acción**
	1	El usuario selecciona la mesa
	2	Los productos se muestran organizados por categoría (Bebida, Comida, Postre)
	3	El usuario puede seleccionar entre las 3 categorías dependiendo del producto que quiera ofrecer aumentando o disminuyendo la cantidad necesaria
Postcondición	El usuario podrá realizar el pedido con los productos preseleccionados	

Tabla 2-11 CU-07: Visualizar productos

2.2.3.8 Caso de uso – 08: Ver cuenta

Ilustración 2-9 CU-08: Ver cuenta

Caso de uso – 08	Ver cuenta	
Actores	Camarero	
Descripción	El usuario podrá mostrar la cuenta a los clientes	
Precondición	El usuario ha iniciado sesión y ha seleccionado una mesa	
Secuencia	**Paso**	**Acción**
	1	El usuario la mesa para la que quiere ver la cuenta
	2	El usuario pulsa el botón de ver cuenta
Postcondición	Muestra un modal en la pantalla mostrando todos los productos junto la cantidad, el precio de cada uno y la cantidad total de la cuenta	

Tabla 2-12 CU-08: Ver cuenta

2.2.3.9 Caso de uso – 09: Revisar Estado de Pedidos

Ilustración 2-10 CU-09: Revisar Estado de Pedidos

Caso de uso – 09	Revisar Estado de Pedidos	
Actores	Camarero o cocinero	
Descripción	El usuario podrá revisar en tiempo real el estado de los pedidos de cada mesa	
Precondición	El usuario ha iniciado sesión y ha pulsado el botón de ‘Comandas’ o ‘Mis comandas’ dependiendo de si es camarero o cocinero	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa el botón de ‘Comandas’ o ‘Mis comandas’ dependiendo de si es camarero o cocinero
	2	En la pantalla aparecen todos los pedidos ordenados por fecha en que se realizó de manera ascendente
Postcondición	El usuario puede visualizar el estado actual de los pedidos	

Tabla 2-13 CU-09: Revisar Estado de Pedidos

2.2.3.10 Caso de uso 10 – Actualizar estado de pedidos

Ilustración 2-11 CU-10: Actualizar estado de pedidos

Caso de uso – 10	Actualizar estado de pedidos	
Actores	Cocinero	
Descripción	El usuario podrá actualizar el estado del pedido para que el camarero esté al tanto	
Precondición	El usuario ha iniciado sesión y ha pulsado el botón de ‘Comandas’	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa el botón de ‘Comandas’
	2	El usuario pulsa sobre el pedido que quiere actualizar
	3	Aparece un dialogo preguntando si quiere actualizar el pedido a ‘EN PREPARACIÓN’ o ‘LISTO’ dependiendo de si el pedido está en ‘PENDIENTE’ o ‘EN PREPARACIÓN’
Postcondición	El estado del pedido se actualiza en la base de datos y el camarero podrá visualizar el estado de los pedidos en tiempo real. Además, si el pedido pasa a listo el camarero al que pertenezca el pedido recibirá una notificación en su teléfono	

Tabla 2-14 CU-10: Actualizar estado de pedidos

2.2.3.11 Caso de uso – 11: Modificar pedido en tiempo real

Ilustración 2-12 CU-11: Modificar pedido en tiempo real

Caso de uso – 11	Revisar Estado de Pedidos	
Actores	Camarero	
Descripción	El usuario podrá modificar los productos del pedido en tiempo real	
Precondición	El usuario ha iniciado sesión y ha realizado algún pedido para alguna mesa	
Secuencia	**Paso**	**Acción**
	1	El usuario pulsa el botón de 'Mis comandas'
	2	En la pantalla que aparecen todos los pedidos pulsa sobre el pedido que quiere actualizar
	3	La aplicación lo lleva a la vista de realizar el pedido, pero esta vez con los productos preseleccionados
	4	El usuario agrega o elimina productos
	5	Se actualiza el pedido en la base de datos para que pueda ser visualizado por los cocineros
Postcondición	En la vista de todos los pedidos aparecerá el pedido actualizado	

Tabla 2-15 CU-11: Revisar Estado de Pedidos

2.2.3.12 Caso de uso – 12: Base de datos centralizada

Ilustración 2-13 CU-12: Base de datos centralizada

Caso de uso – 12	Revisar Estado de Pedidos	
Descripción	La aplicación utiliza una base de datos para almacenar información importante y relevante	
Precondición	La aplicación está abierta y con conexión a internet	
Secuencia	**Paso**	**Acción**
	1	La aplicación accede a la base de datos de MongoDB Atlas para obtener la información de las mesas, pedidos, usuarios, notificaciones, y más
	2	La aplicación va actualizando la base de datos con nueva información cuando sea necesario
Postcondición	La base de datos contiene toda la información actualizada en tiempo real y de forma precisa	

Tabla 2-16 CU-12: Revisar Estado de Pedidos

2.2.3.13 Caso de uso – 13: Recibir notificación

Ilustración 2-14 CU-13: Recibir notificación

Caso de uso – 13	Recibir notificación	
Actores	Camarero	
Descripción	El camarero recibirá una notificación cuando el estado del pedido pase a “listo”	
Precondición	Tiene que haber un pedido registrado perteneciente a ese camarero	
Secuencia	**Paso**	**Acción**
	1	El camarero registra un pedido
	2	El cocinero entra en la lista de pedidos y pone el pedido a ‘listo’
	3	La aplicación detecta el cambio y envía una notificación al camarero al que pertenece el pedido
Postcondición	El camarero puede revisar la notificación desde la bandeja de notificaciones en la pantalla principal	

Tabla 2-17 CU-12: Revisar Estado de Pedidos

3 DESARROLLO

Dado que se ha seguido un enfoque incremental utilizando una metodología ágil, este apartado se estructura en cinco iteraciones. Cada iteración se centrará en una parte específica del desarrollo del sistema, comenzando con la configuración del entorno de desarrollo y progresando hacia funcionalidades más complejas y críticas para el funcionamiento de la aplicación.

- **Iteración 1: Configuración del Entorno de Desarrollo**
  - **Objetivo:** El objetivo de esta primera iteración fue configurar el entorno de desarrollo necesario para iniciar el proyecto. Esto incluyó la instalación y configuración de herramientas clave como Android Studio, IntelliJ IDEA, React Native y MongoDB. Para ello se obtuvo información de React Native [3]
  - **Actividades principales:**
    - Instalación de Android Studio e IntelliJ IDEA.
    - Configuración de React Native para Android.
    - Creación del proyecto inicial y configuración del archivo 'build.gradle'.
    - Configuración de MongoDB, MongoDB Atlas y MongoDB Compass para la gestión de la base de datos.
    - Pruebas iniciales para verificar la correcta integración entre las herramientas.
  - **Resultados:** Al finalizar esta iteración, el entorno de desarrollo estaba completamente configurado, y se habían realizado pruebas iniciales para garantizar la correcta interacción entre los diferentes componentes del sistema. Esto sentó las bases para el desarrollo de las funcionalidades específicas en las siguientes iteraciones.
- **Iteración 2: Implementación del Registro, Inicio de Sesión y Gestión de Usuarios**

- **Objetivo:** Implementar la funcionalidad de registro de usuarios, inicio de sesión, y la gestión de perfiles, lo que incluye el restablecimiento de contraseñas y la autenticación segura mediante JWT.
- **Actividades principales:**
  - Desarrollo del formulario de registro de usuarios.
  - Implementación de la funcionalidad de inicio de sesión con validación de credenciales.
  - Configuración de la autenticación mediante JWT para asegurar las sesiones de usuario.
  - Desarrollo de la funcionalidad de restablecimiento de contraseñas.
  - Pruebas unitarias y de integración para asegurar la robustez de estas funcionalidades.
- **Resultados:** Esta iteración culminó con la implementación completa de las funcionalidades de usuario, incluyendo registro, inicio de sesión y gestión de contraseñas. Estas funciones se integraron con la base de datos en MongoDB para un almacenamiento seguro de la información del usuario.

- **Iteración 3: Gestión de Mesas y Pedidos**
  - **Objetivo:** Desarrollar la funcionalidad para la visualización de mesas y la gestión de pedidos, permitiendo a los camareros registrar, modificar y eliminar pedidos, así como visualizar la disposición de las mesas en el restaurante.
  - **Actividades principales:**
    - Implementación del mapa interactivo para la visualización de mesas.
    - Desarrollo de la funcionalidad para registrar, modificar y eliminar pedidos.
    - Integración de la funcionalidad de mesas con la gestión de pedidos.

        - Pruebas de usabilidad para asegurar la fluidez de la experiencia del usuario.
        - Pruebas de integración para validar la comunicación entre los módulos de mesas y pedidos.
    - **Resultados:** Al finalizar esta iteración, la aplicación permitía a los usuarios visualizar las mesas y gestionar pedidos de manera eficiente, cumpliendo con los requisitos funcionales establecidos previamente.

- **Iteración 4: Notificaciones en Tiempo Real**
    - **Objetivo:** Implementar el sistema de notificaciones en tiempo real para informar a los camareros cuando un pedido está listo, mejorando la eficiencia en el servicio.
    - **Actividades principales:**
        - Desarrollo del sistema de notificaciones en tiempo real utilizando servicios de mensajería en la nube.
        - Implementación de la funcionalidad de notificación en la aplicación móvil.
        - Pruebas de latencia para asegurar que las notificaciones se entregan en menos de 2 segundos.
        - Validación de la funcionalidad mediante pruebas de estrés para verificar su comportamiento bajo carga.
    - **Resultados:** Con esta iteración, se añadió una funcionalidad crítica que mejoró la capacidad de respuesta del sistema, asegurando que los camareros reciban notificaciones rápidas y precisas cuando los pedidos están listos.

- **Iteración 5: Ajustes Finales y Mejoras**
    - **Objetivo:** Realizar ajustes y mejoras finales en la aplicación, optimizando el rendimiento y refinando la interfaz de usuario en base a las pruebas realizadas.
    - **Actividades principales:**

- Ajustes en la interfaz de usuario para mejorar la usabilidad.
- Realización de pruebas finales de aceptación para asegurar que todas las funcionalidades cumplen con los requisitos establecidos.

- **Resultados:** Al finalizar esta iteración, la aplicación estaba completamente desarrollada, optimizada y lista para su despliegue, cumpliendo con todos los requisitos funcionales y no funcionales identificados al inicio del proyecto.

3.1 Primera Iteración

En esta primera iteración, el objetivo fue establecer las bases del proyecto configurando el entorno de desarrollo y desarrollando las funcionalidades iniciales relacionadas con la autenticación y gestión de usuarios. Se sentaron los cimientos para las futuras iteraciones, asegurando que las herramientas y tecnologías estuvieran correctamente integradas.

3.1.1 Historias de usuario

Durante esta iteración, se abordaron las siguientes historias de usuario:

1. **Configuración del entorno de desarrollo:**
   a. **Historia de Usuario:** Como desarrollador, quiero configurar el entorno de desarrollo para asegurar que todas las herramientas necesarias estén listas para el desarrollo de la aplicación.
   b. **Criterio de Aceptación:**
      i. Android Studio, IntelliJ IDEA, y React Native están instalados y configurados.
      ii. MongoDB y MongoDB Atlas están configurados para la gestión de la base de datos. [10]
      iii. El proyecto está inicializado y se ejecuta sin errores.

3.1.2 Storyboards

Para ilustrar el flujo de trabajo de las principales funcionalidades implementadas en esta iteración, se han creado los siguientes storyboards:

- **Storyboard 1: Configuración del Entorno de Desarrollo**
  - **Descripción:** Este storyboard muestra los pasos seguidos para configurar el entorno de desarrollo, desde la instalación de Android Studio hasta la configuración de MongoDB y React Native.
  - **Pasos de configuración:**
    - **Creación del proyecto inicial:**
      - **Paso 1:** Instalación de Node.js [12] [4] y npm descargándolos e instalando desde el sitio web oficial.
      - **Paso 2:** Instalar React Native CLI, para ello se ejecuta el comando en la terminal '**npm install -g react-native-cli**'.
      - **Paso 3:** Se procede a instalar Gradle descargando la última versión desde el sitio web oficial.
      - **Paso 4:** Después de instalar Gradle, se debe agregar la ubicación de la carpeta 'bin' de Gradle al path del sistema. Esto se hace accediendo a las variables de entorno de nuestro sistema y agregando la ruta de la carpeta 'bin' de Gradle al PATH. Para comprobar que todo ha ido bien se ejecutará 'gradle -v' en nuestra terminal.
      - **Paso 5:** Ahora se crea el proyecto ejecutando el comando en nuestra terminal **'npx react-native init DineCrew --template react-native-template-typescript**' y ya se habrá creado el proyecto.
    - **Instalación de Android Studio:**
      - **Paso 1:** Descargar e instalar Android Studio desde el sitio oficial y seguir los pasos.

- **Paso 2:** Configurar el SDK de Android, asegurando que las versiones necesarias estén disponibles.
- **Paso 3:** Una vez instalado Android Studio se procede a configurar un emulador, para ello se crea un proyecto empty con el nombre de nuestro proyecto.
- **Paso 4:** Una vez creado el proyecto se accede a '**Device Manager**', aparecerá una ventana donde estará el botón '**Create Device**'. Una vez pulsado se indica una serie de pasos a seguir que se explican debajo mediante imágenes para que sea más fácil de entender. Con esto se quedará el emulador donde se van a ejecutar todas las pruebas configurado
- **Imágenes:**

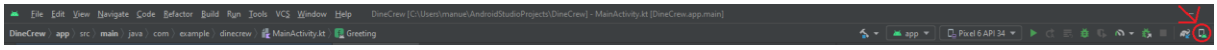

Ilustración 3-1 Ubicación de Device Manager

Pulsamos sobre '**Device Manager**' y aparecerá una pestaña indicando la ilustración de debajo

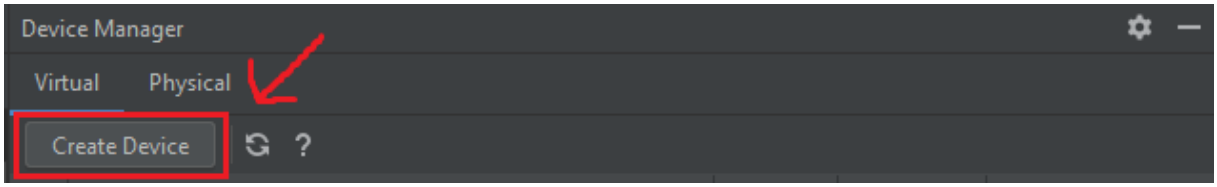

Ilustración 3-2 Creación de un dispositivo

Una vez aquí, pulsar el botón '**Create Device**' y aparecerá una ventana en nuestra pantalla como esta:

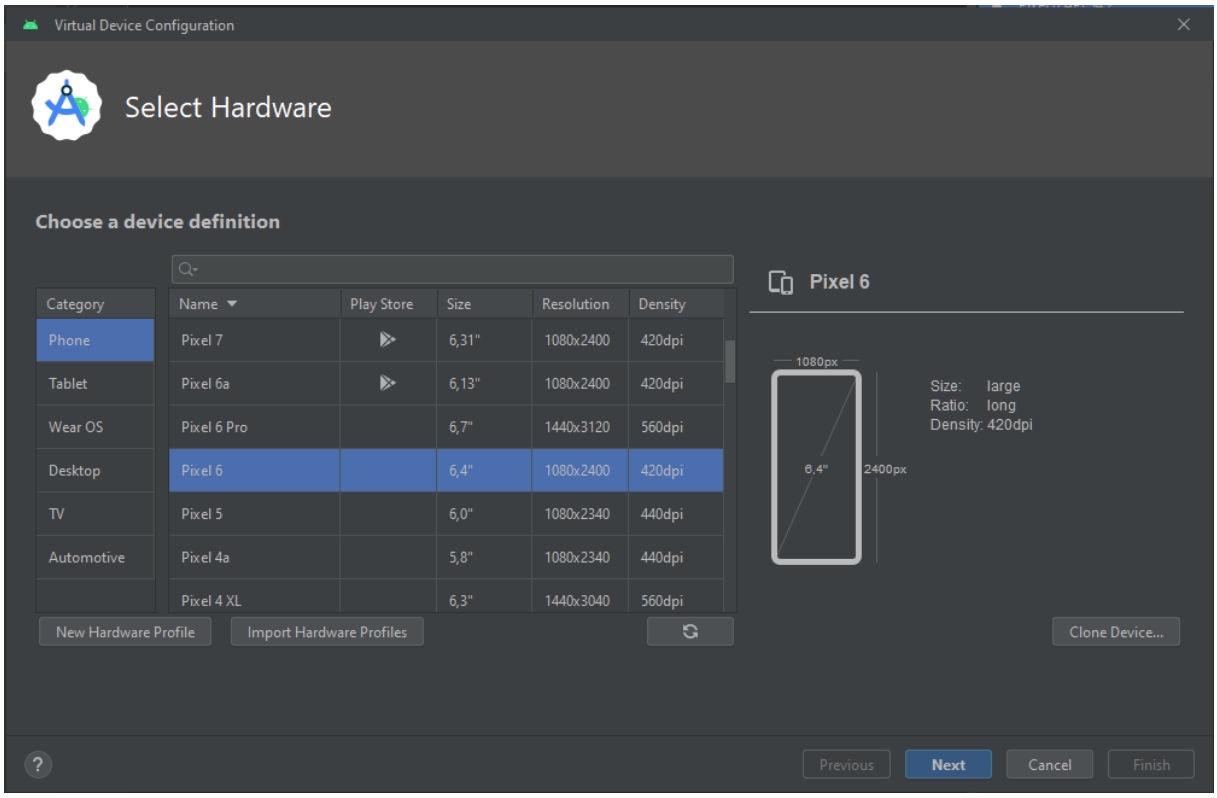

Ilustración 3-3 Pantalla inicial

Se indica el dispositivo donde se van a realizar las pruebas y se pulsa el botón '**Next**'. Aparecerá otra ventana en nuestra pantalla como esta:

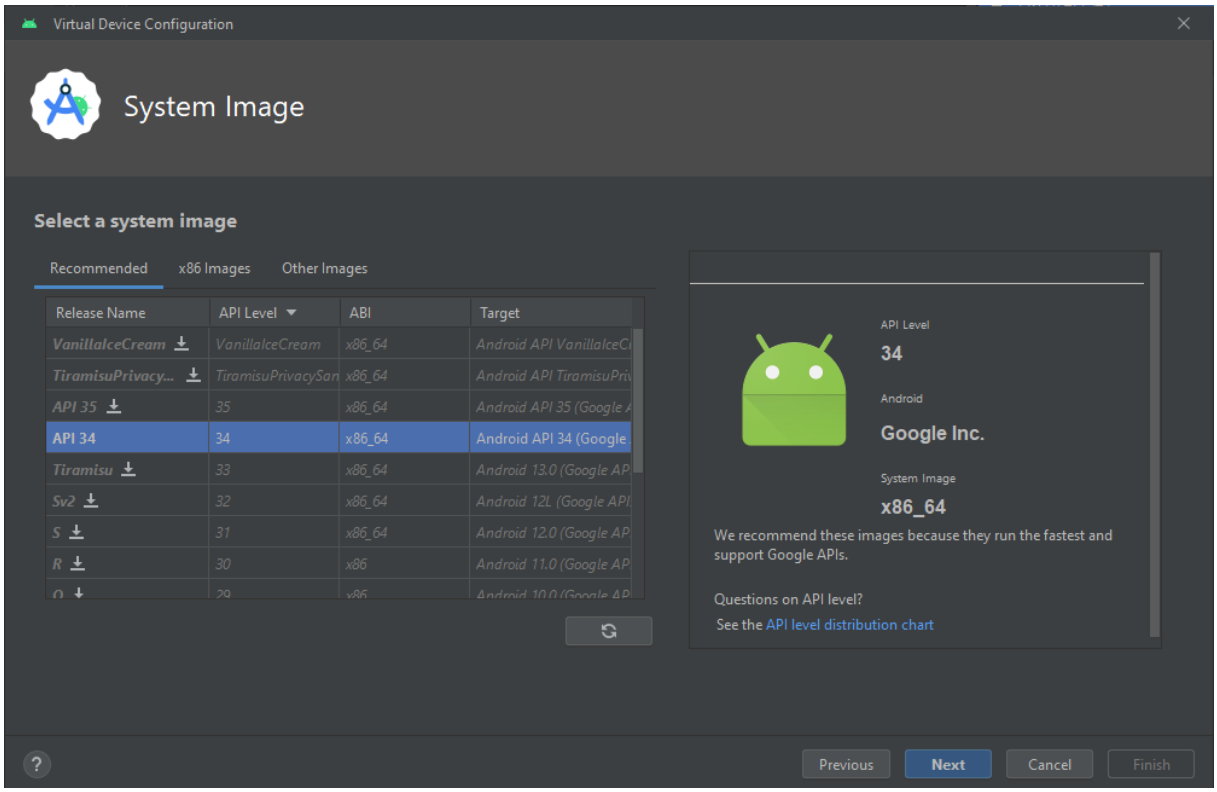

Ilustración 3-4 Versión Android

Se vuelve a pulsar '**Next**' y aparecerá de nuevo otra ventana como esta:

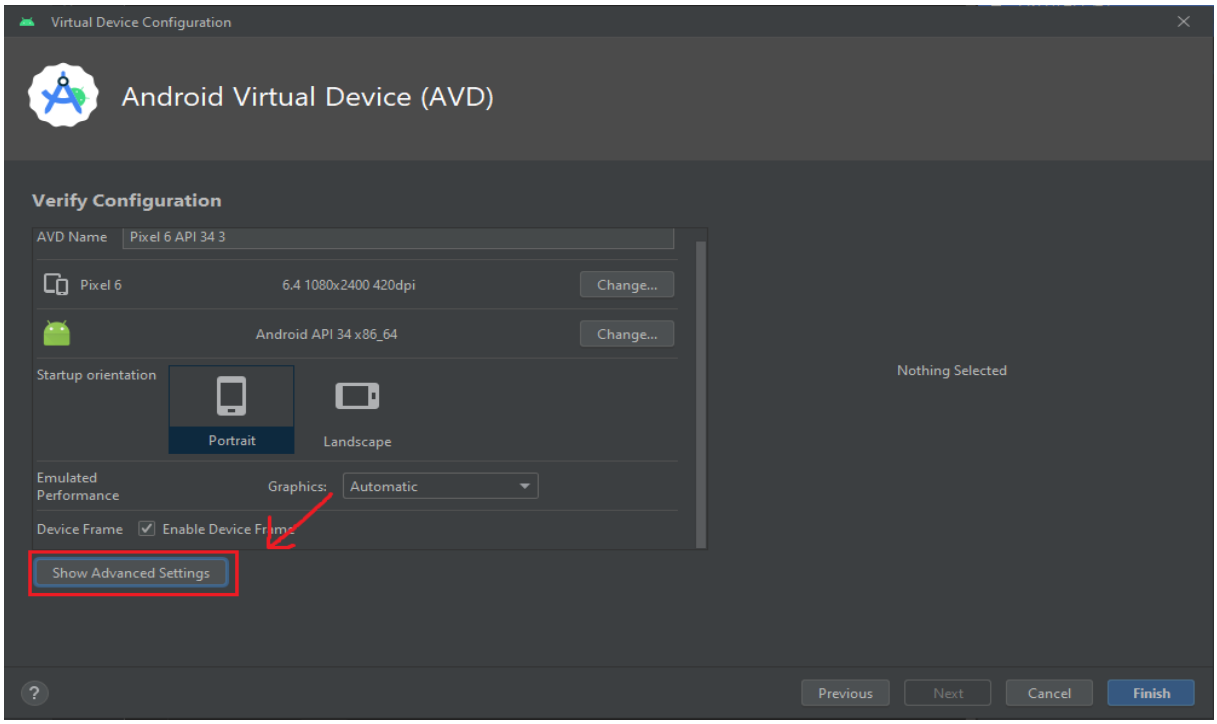

Ilustración 3-5 Configuración avanzada

Una vez aquí se pulsará sobre el botón indicado para acceder a la configuración avanzada del dispositivo para mejorarlo

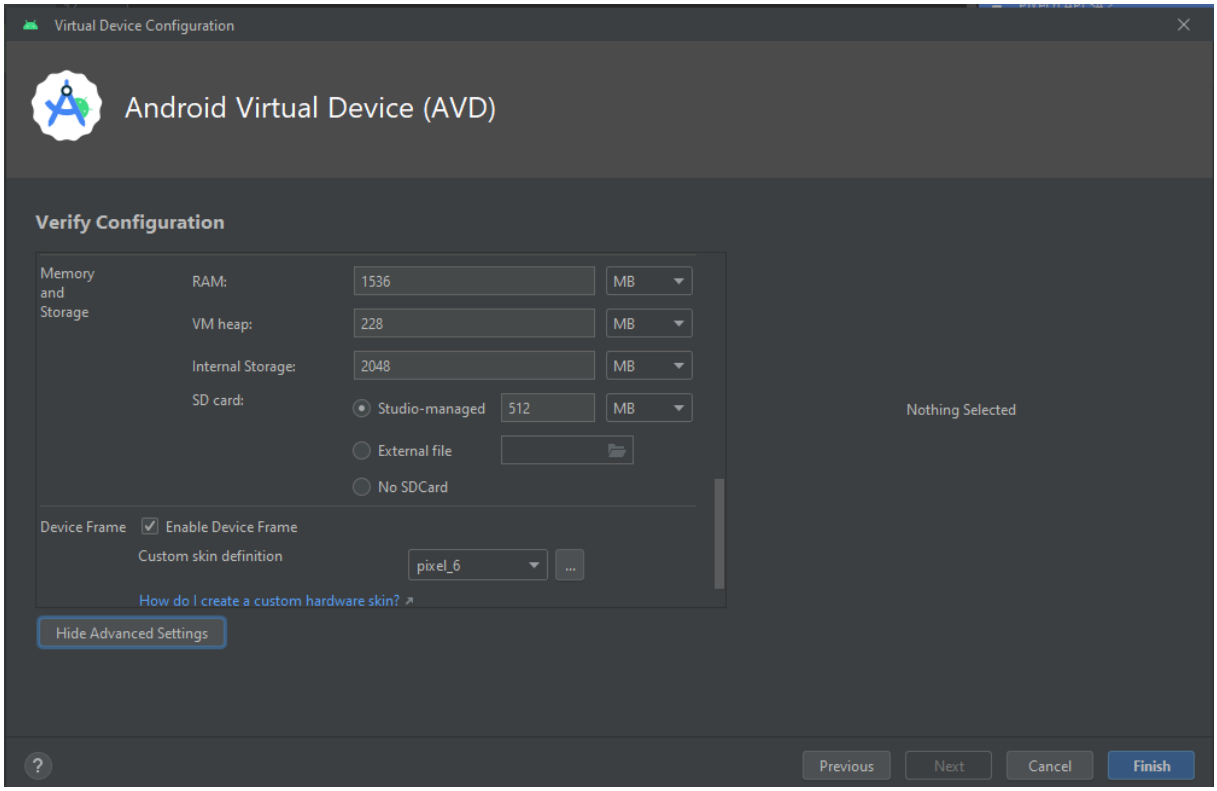

Ilustración 3-6 Configuración inicial del dispositivo

Aquí será libertad del usuario indicar la memoria RAM y el almacenamiento del dispositivo según sus expectativas, **recomendación**, no poner menos de 4GB de RAM y 2GB de almacenamiento en el disco duro. Una vez configurado el dispositivo se pulsa el botón '**Finish**' y aparecerá una pestaña como esta:

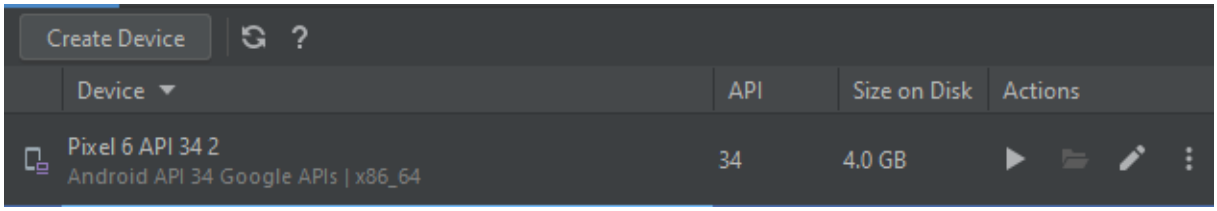

Ilustración 3-7 Representación del dispositivo configurado

- **Instalación de IntelliJ IDEA:**
  - **Paso 1:** Descargar e instalar IntelliJ IDEA

- **Paso 2:** Configurar el proyecto creado anteriormente, asegurando la integración con '**build.gradle**' para la correcta compilación.

▪ **Configuración de MongoDB:**

- **Paso 1:** Registrarse en MongoDB Atlas y crear una base de datos.
- **Paso 2:** Configurar las variables de entorno para la conexión con la base de datos.
- **Paso 3:** Verificar la conectividad desde la aplicación mediante pruebas básicas de conexión.
- **Imágenes:**

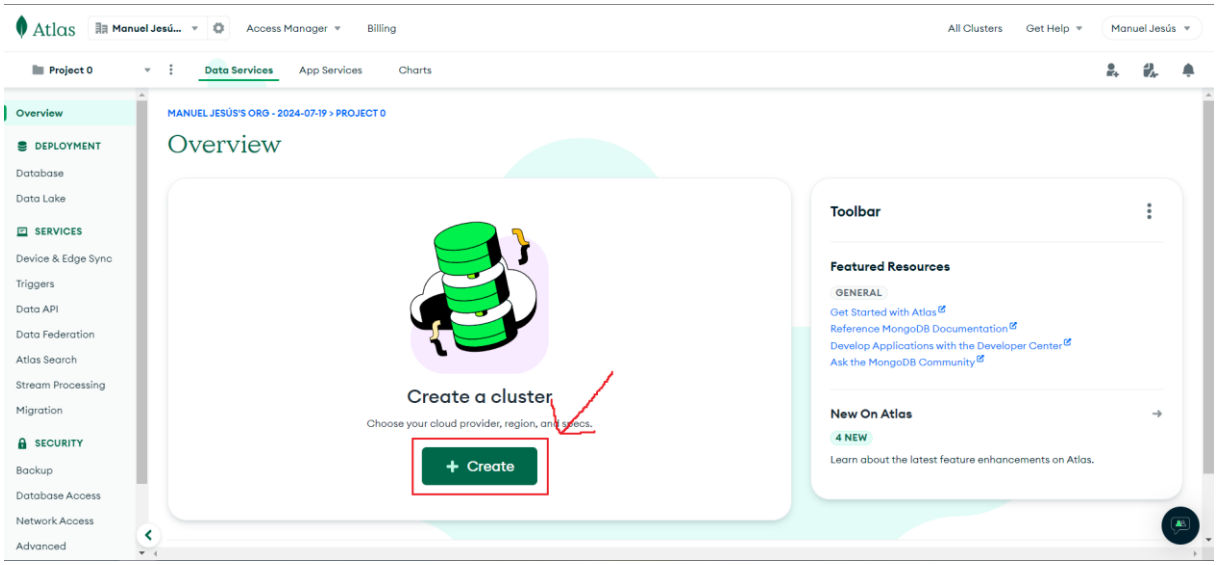

Ilustración 3-8 Inicio de MongoDB Atlas

Al pulsar el botón '**Create**', aparecerá la siguiente ventana:

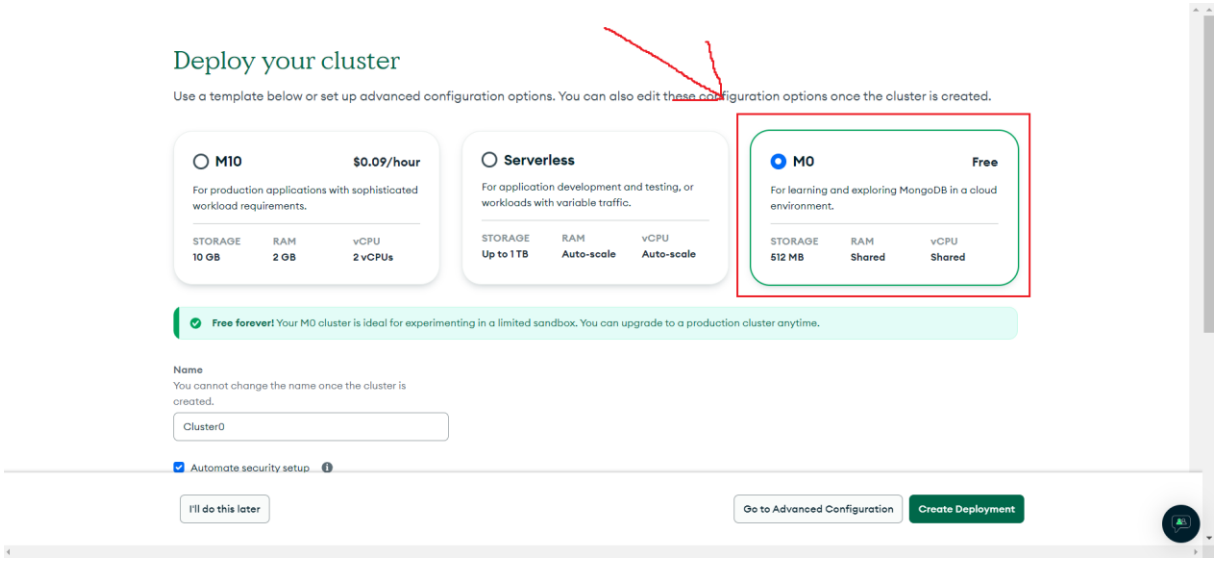

Ilustración 3-9 Elección del almacenamiento

Para el siguiente paso, elegir la opción M0 que es la opción gratuita que ofrece MongoDB Atlas al registrar un usuario por primera vez. Una vez aquí un poco más abajo se indica el nombre del cluster [13] y la región. Por defecto está seleccionada la del usuario, se recomienda no cambiar, seguidamente se pulsa el botón '**Create Deployment**'.

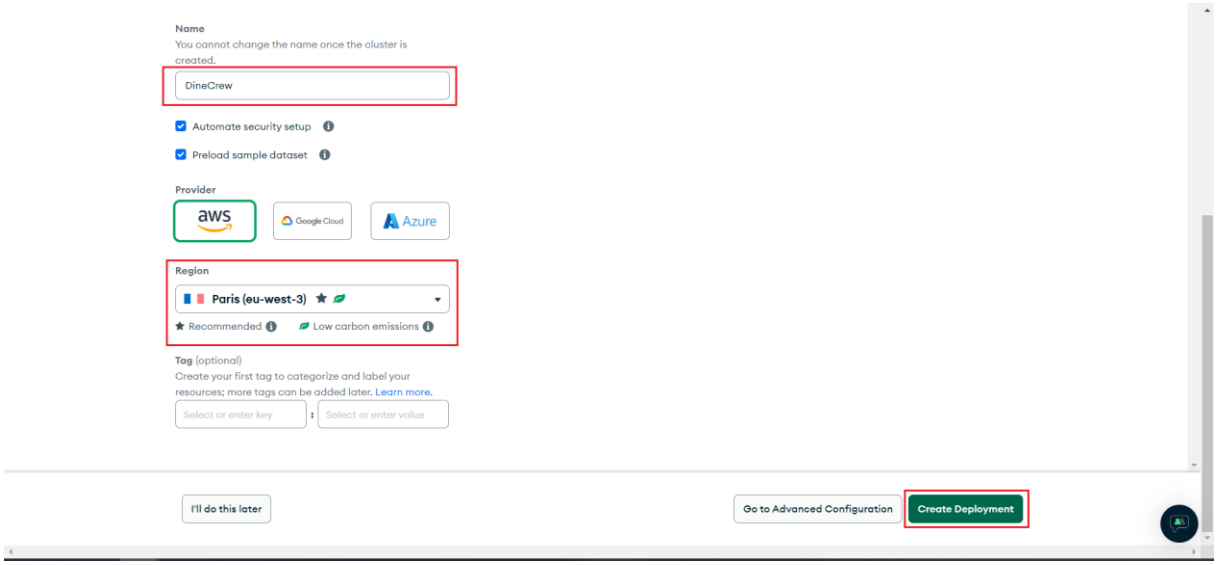

Ilustración 3-10 Configuración

El siguiente paso será copiar las credenciales de la base de datos y guardarlas, importante darle a '**Copy**' antes de pulsar '**Create Database User**'.

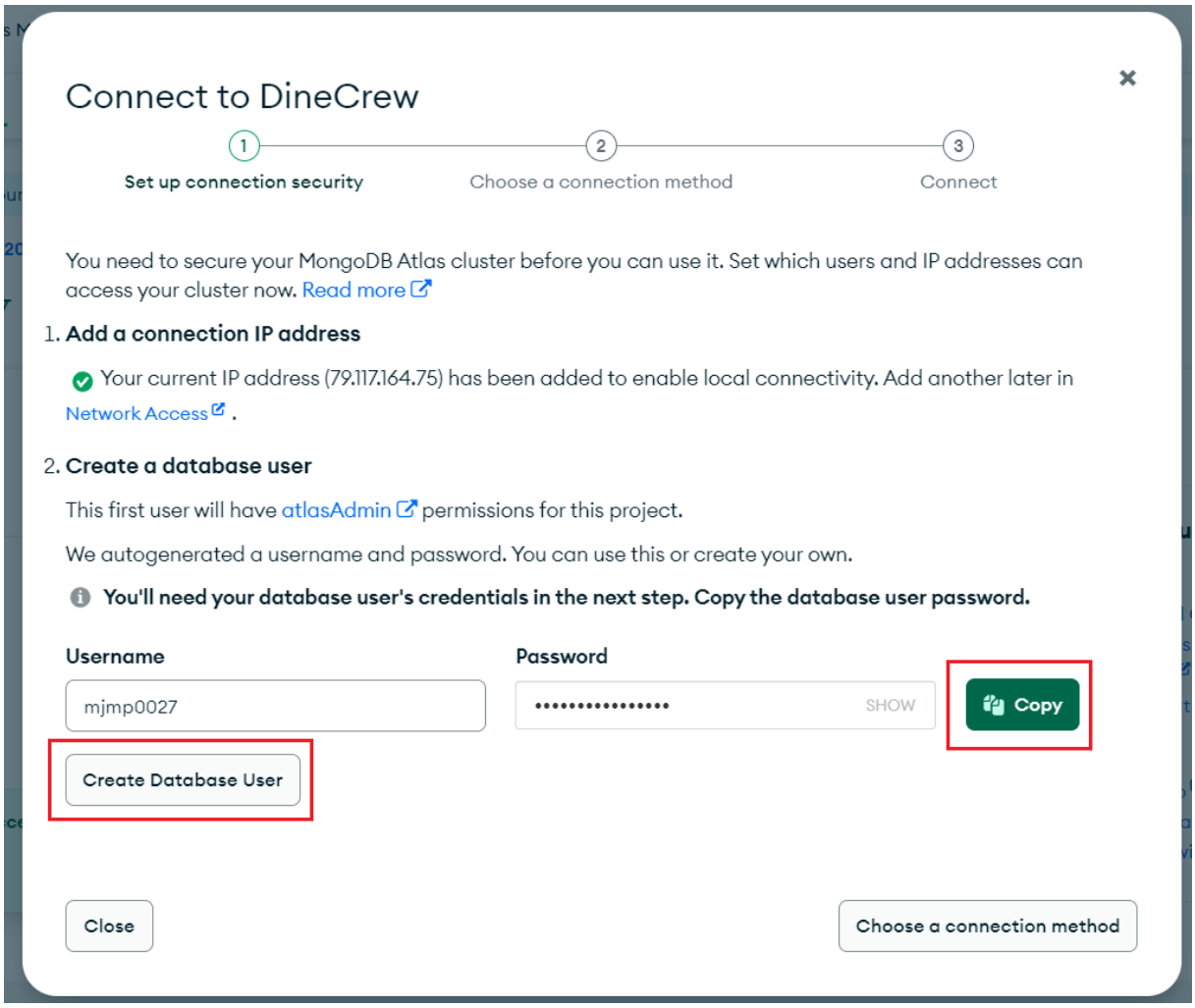

Ilustración 3-11 Credenciales

El siguiente paso será elegir '**Drivers**' para la conexión con nuestra aplicación

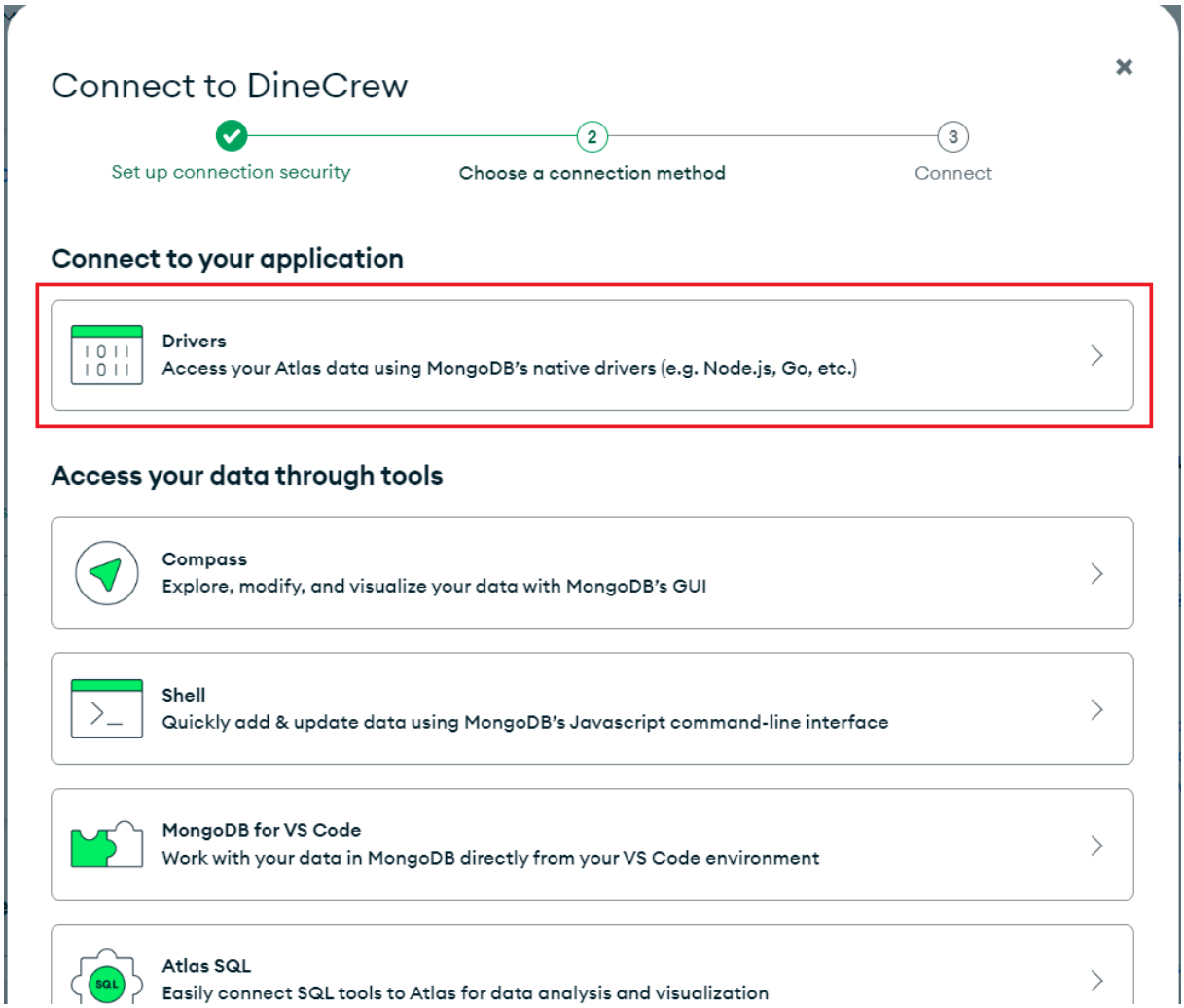

Ilustración 3-12 Drivers

Una vez aquí se copia la url para usarla posteriormente en nuestro "**archivo .properties**', como indica en la imagen remplazar '**<password>**' por la contraseña copiada anteriormente

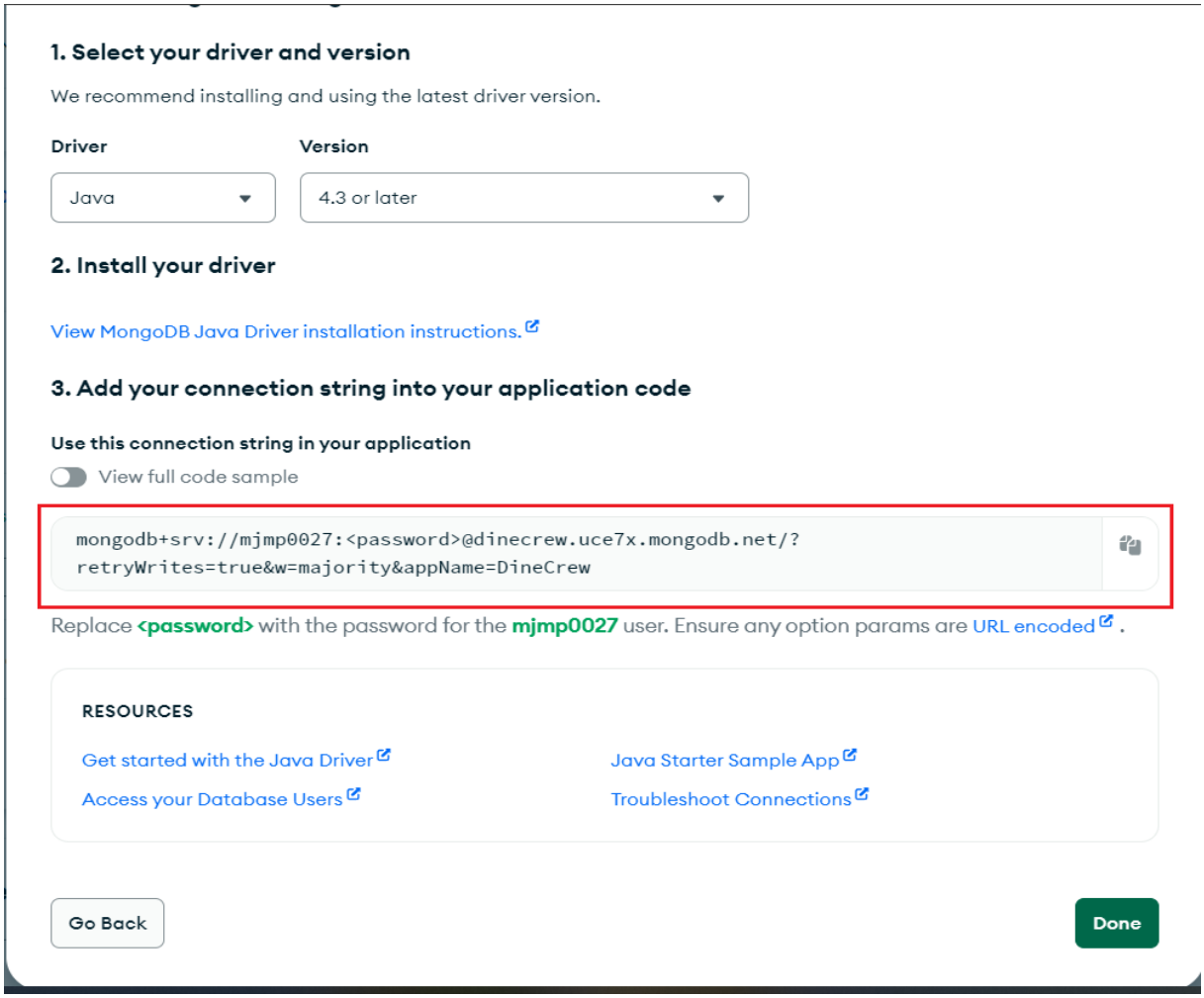

Ilustración 3-13 Url para la conexión

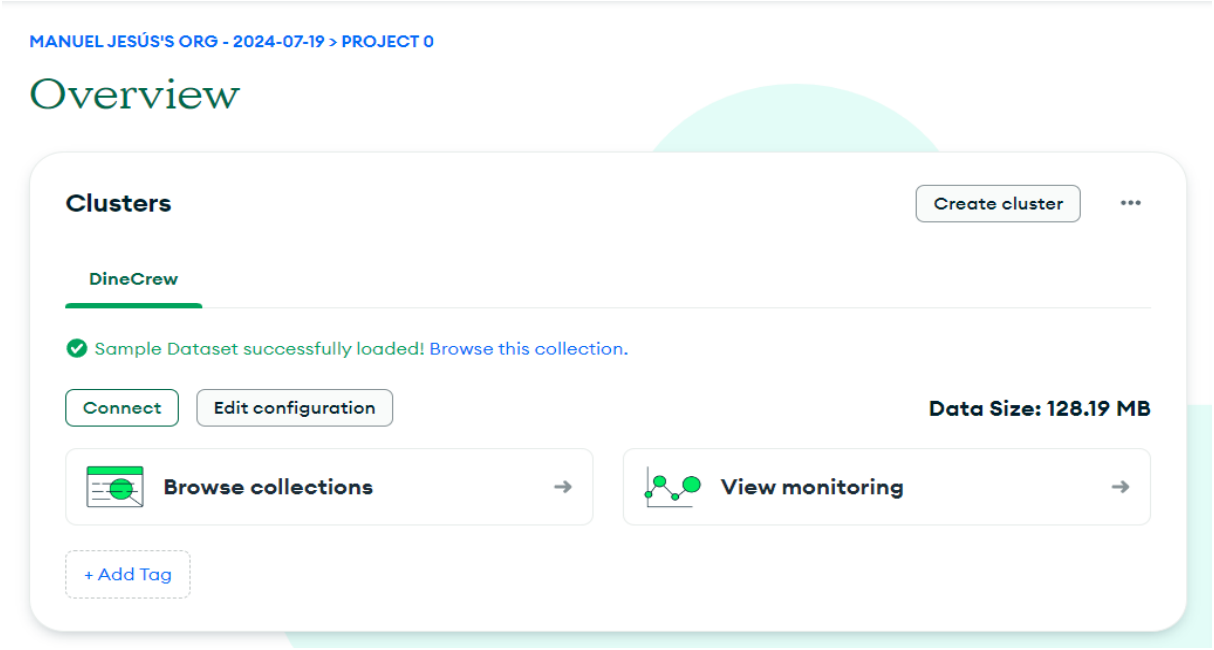

Ilustración 3-14 Vista general

Esta será la imagen final que se verá en nuestra pantalla, como se puede visualizar, indica que el tamaño de los datos es de 128.19MB, esto es ocasionado porque MongoDB Atlas proporciona una base de datos por defecto. Para eliminar estos datos que no son necesarios, pulsa el botón '**Browse collections**' y aparecerá esta ventana

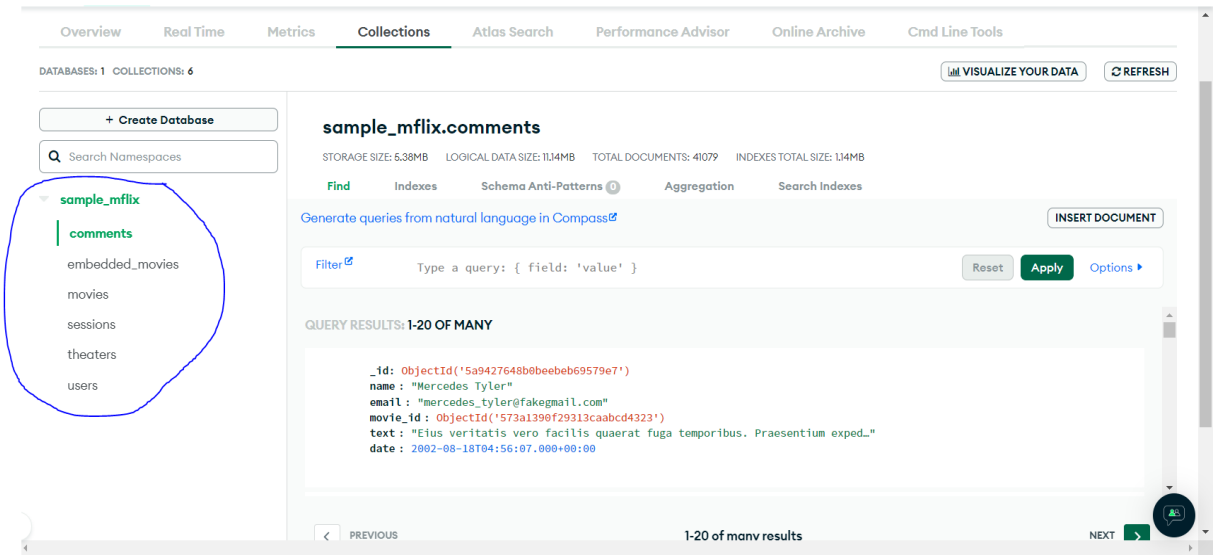

Ilustración 3-15 Selección de la base de datos

Se puede comprobar que hay una base de datos ya creada, para eliminarla hay que pulsar sobre el botón de la papelera que aparece en pequeño

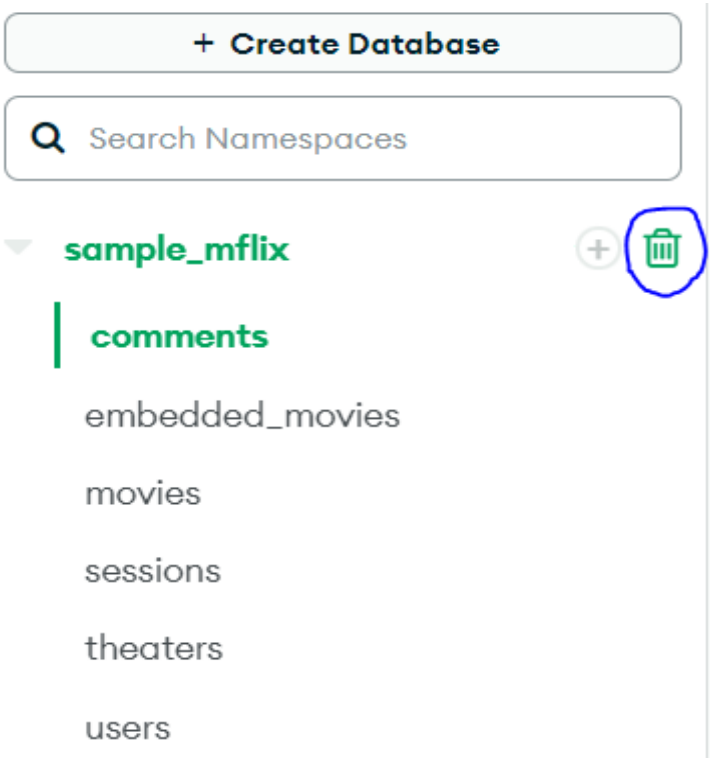

Ilustración 3-16 Eliminación de la base de datos por defecto

Una vez eliminada vamos a crear nuestra base de datos, para ellos es recomendable descargar e instalar MongoDB Compass la aplicación de escritorio para el manejo de la base de datos Mongo.

Después de realizar la instalación abrimos la aplicación y aparecerá algo similar a esto:

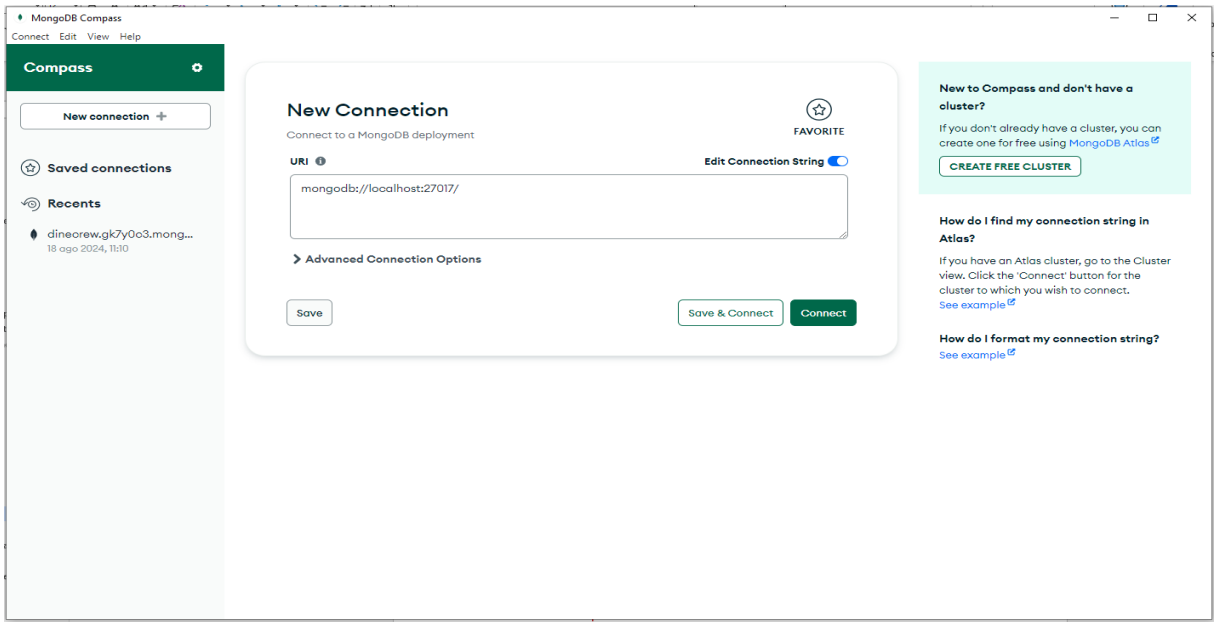

Ilustración 3-17 Inicio de MongoDB Compass

En el recuadro donde indica URI [14] será necesario pegar la url obtenida anteriormente en la ilustración 3.13, pulsaremos el botón '**Connect**' y seguidamente aparecerá una pestaña vacía. Una vez allí pulsaremos el botón '**+**' tal como aparece en la imagen

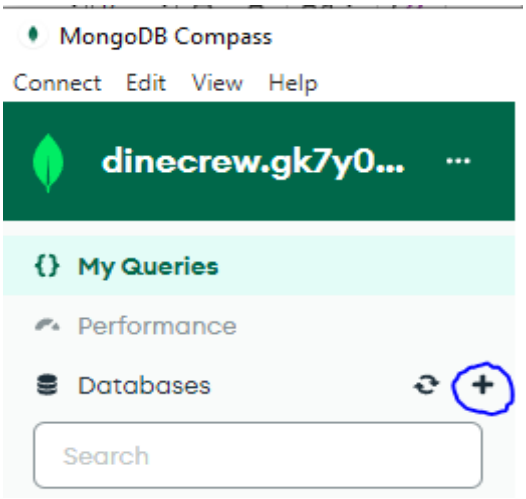

Ilustración 3-18 Agregar una base de datos

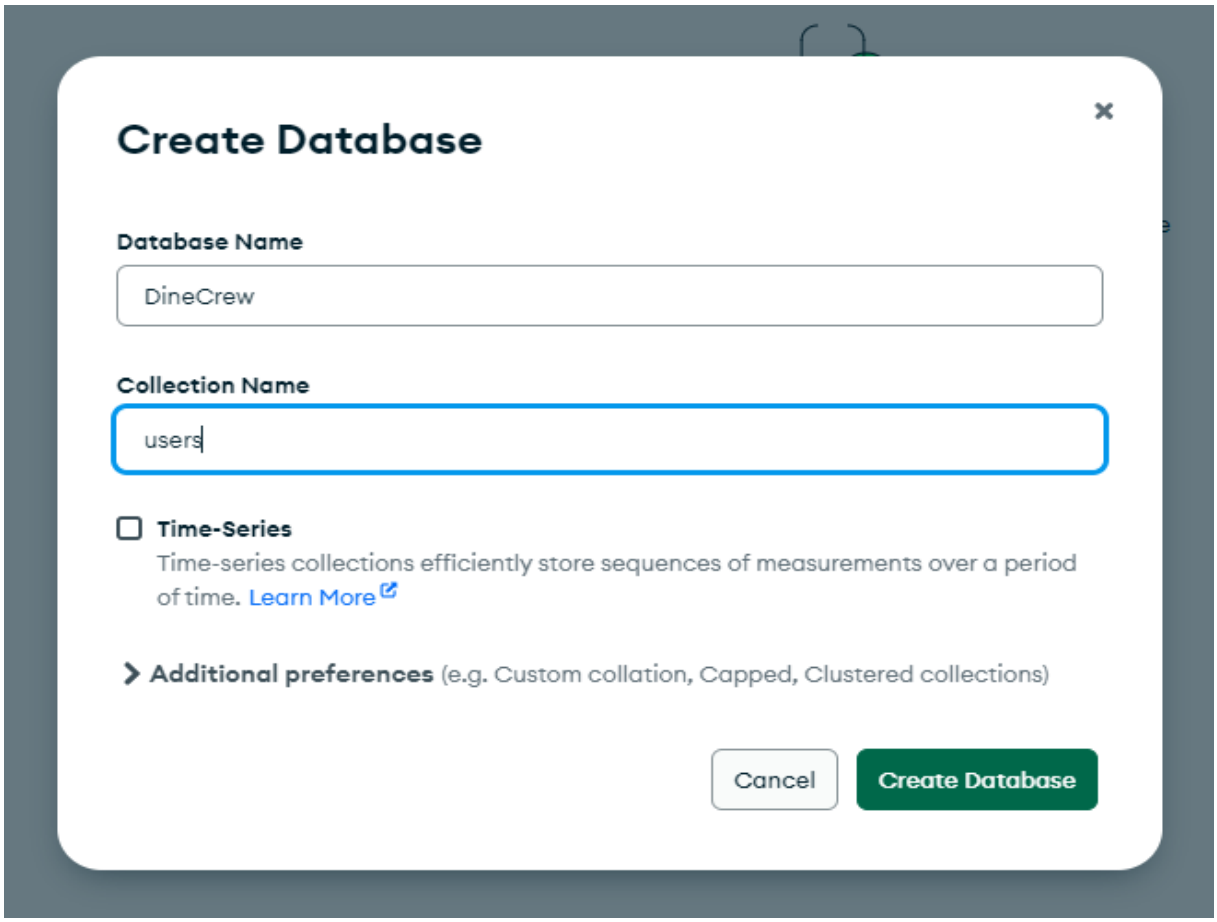

Ilustración 3-19 Credenciales

Aquí indicaremos el nombre de nuestra base de datos y el nombre de la colección que queramos crear, en mi caso '**users**', ya que será la primera colección creada. Al pulsar el botón '**Create Database**' ya se habrá terminado uno de los pasos importantes para la configuración de la base de datos.

Ahora se mostrará la configuración necesaria en nuestro proyecto Java para conseguir la conexión con la base de datos.

El primer paso será la creación de un proyecto Java con Spring Boot usando la configuración de '**Maven**' [15] eligiendo las dependencias '**Spring web**' y '**Spring Data MongoDB**' como se indica en las siguientes imágenes:

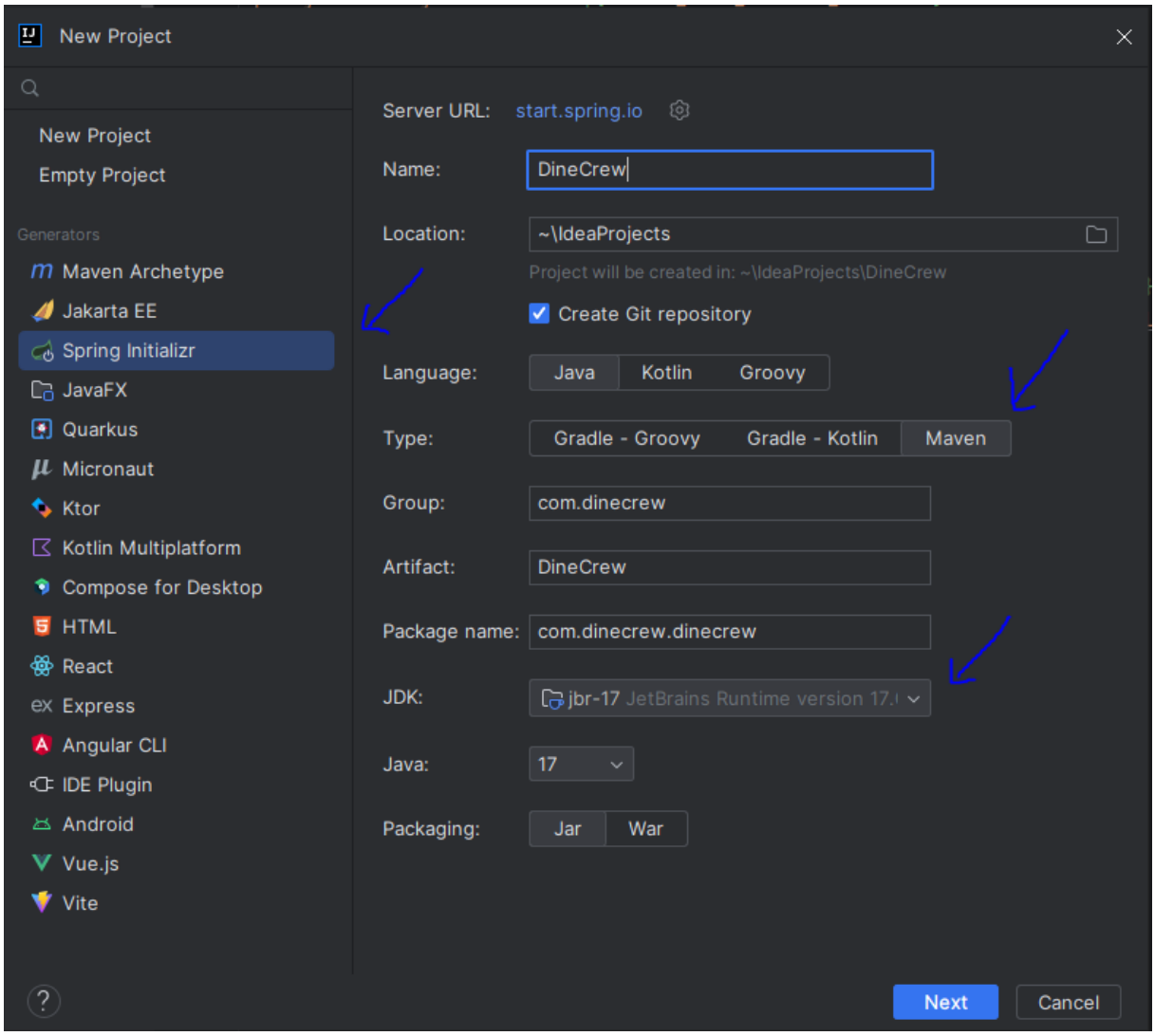

Ilustración 3-20 Inicio del backend en IntelliJ IDEA

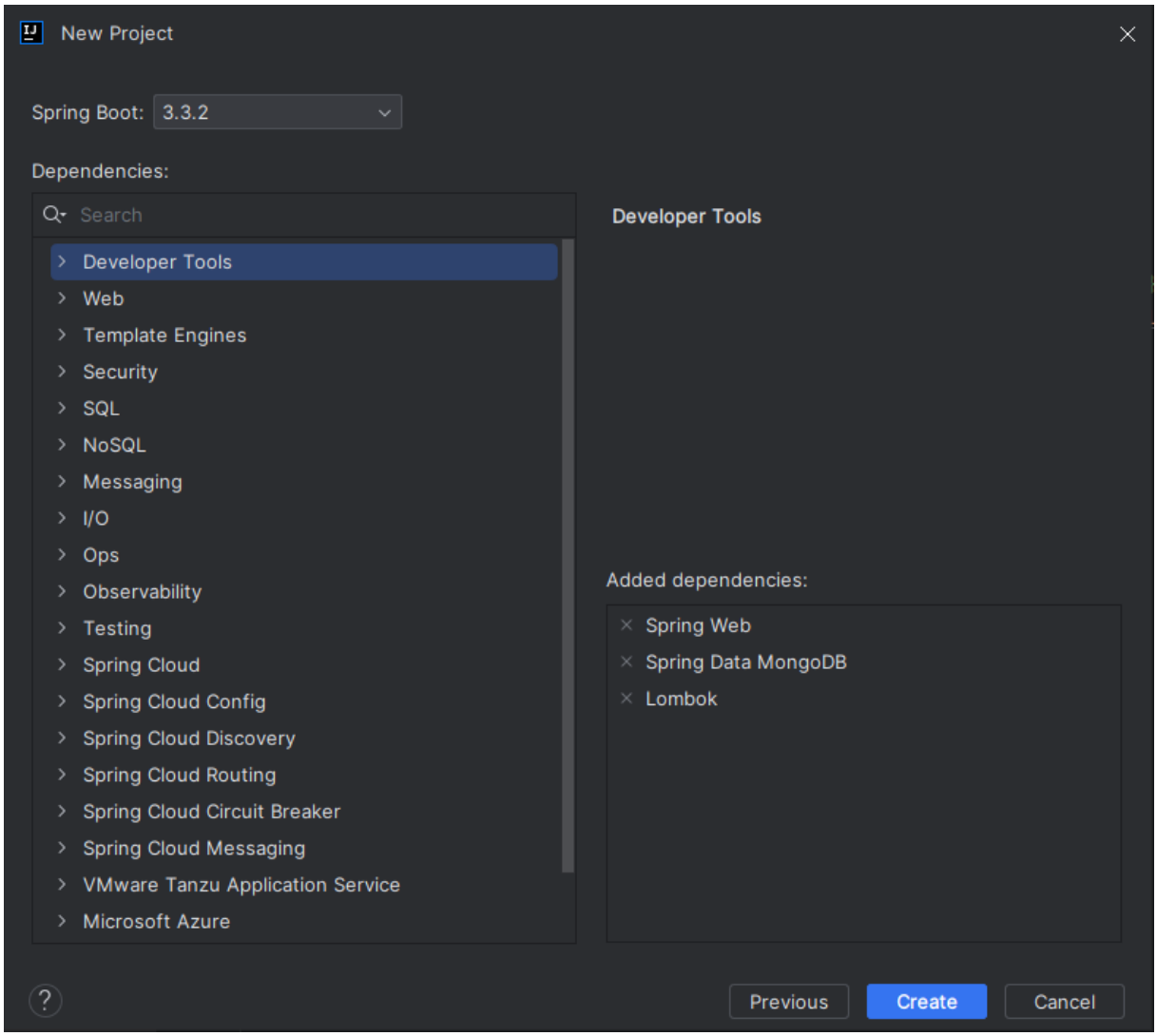

Ilustración 3-21 Elección de dependencias

Una vez creado el proyecto será necesario ir al archivo '**application.properties**' ubicado en la ruta '**src/main/resources/application.properties**'

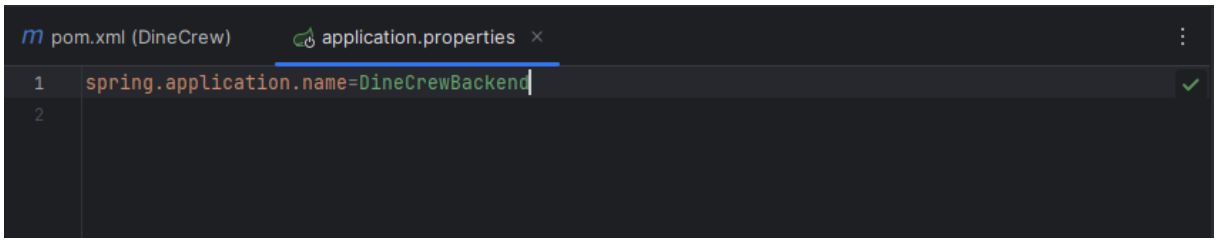

Ilustración 3-22 Inicio del proyecto

Una vez aquí procedemos a configurar la base de datos para que nuestra aplicación pueda conectar con ella como se indica en la siguiente ilustración

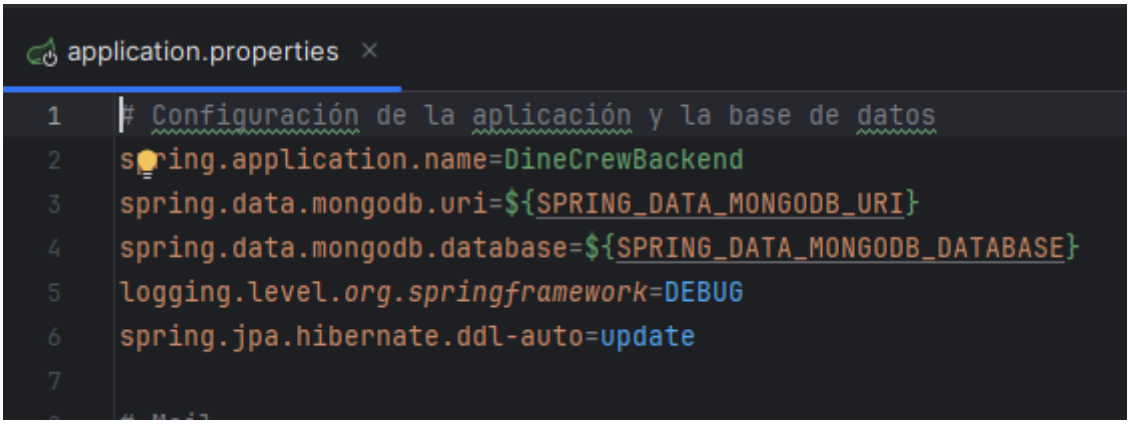

Ilustración 3-23 Configuración del archivo .properties

El siguiente paso para que la aplicación pueda conectar con la base de datos en MongoDB es abrir la clase '**main**' ubicada en '**src/main/java/com.dinecrew,dinecrewbackend/ DineCrewBackendApplication**' y anotarla con '**@EnableMongoRepositories(basePackages = {"com.dinecrew.dinecrewbackend.usuarios"}**, donde 'usuarios' será el paquete donde estará ubicado el documento '**User**' junto con su repositorio.

Tal y como se muestra en la ilustración 3.23 en '**spring.data.mongodb.uri**' hay que pegar la URI obtenida en la ilustración 3.13 y en '**spring.data.mongodb.database**' habrá que indicar el nombre de la base de datos.

3.2 Segunda Iteración

3.2.1 Historias de usuario

Durante esta iteración se abordarán las siguientes historias de usuario:

1. **Registro de Usuarios:**
    a. **Historia de usuario:** Como usuario, quiero poder registrarme en la aplicación proporcionando mi correo electrónico y una contraseña
    b. **Criterio de Aceptación:**

i. Los usuarios pueden registrarse utilizando un formulario de registro.

ii. Los datos de los usuarios se almacenan de forma segura en la base de datos

iii. Las contraseñas están encriptadas antes de ser almacenadas.

2. **Inicio de Sesión:**

   a. **Historia de usuario:** Como usuario registrado, quiero poder iniciar sesión en la aplicación utilizando mi correo electrónico y contraseña.

   b. **Criterios de Aceptación:**

      i. Los usuarios pueden iniciar sesión con sus credenciales.

      ii. La autenticación se realiza mediante JWT para garantizar sesiones seguras.

3. **Restablecimiento de Contraseña:**

   a. **Historia de Usuario:** Como usuario, quiero poder recuperar mi contraseña en caso de olvidarla.

   b. **Criterios de Aceptación:**

      i. El usuario recibe un correo electrónico con un enlace para restablecer su contraseña.

      ii. La nueva contraseña se encripta y se almacena en la base de datos.

4. **Cierre de Sesión:**

   a. **Historia de Usuario:** Como usuario autenticado, quiero poder cerrar sesión de manera segura en la aplicación.

   b. **Criterios de Aceptación:**

      i. El usuario puede cerrar sesión y se invalida el token JWT.

      ii. La sesión se termina correctamente y el usuario es redirigido a la pantalla de inicio de sesión.

3.2.2 Storyboards

Para ilustrar el flujo de trabajo de las principales funcionalidades implementadas en esta iteración, se han creado los siguientes storyboards:

- **Storyboard 1: Registro de Usuarios**
  - **Descripción:** Muestra la secuencia de pantallas y acciones necesarias para que un usuario se registre en la aplicación.
  - **Imágenes:**

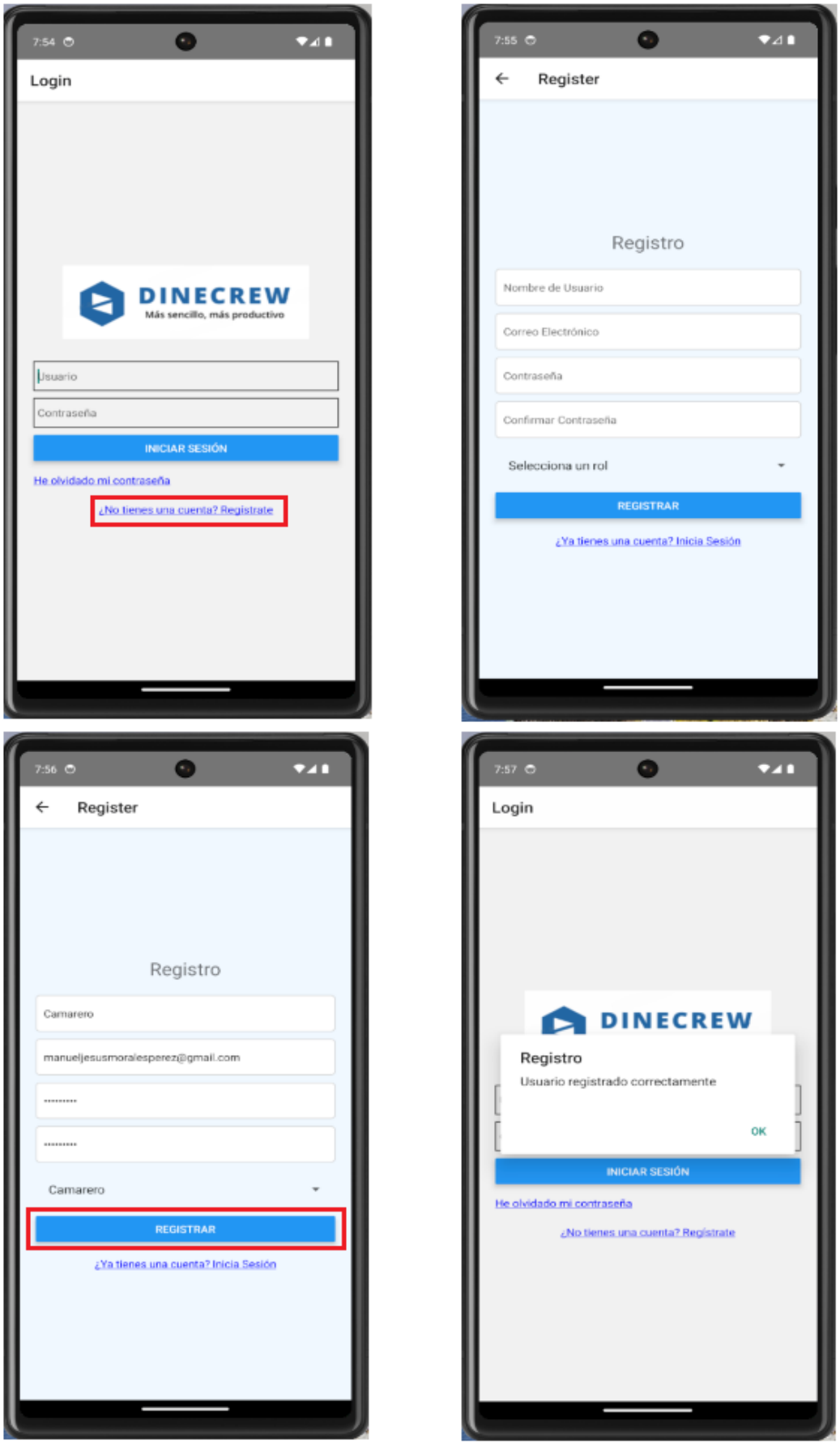

Ilustración 3-24 Flujo de Registro de usuarios

- **Storyboard 2: Inicio de Sesión**
  - **Descripción:** Detalla el proceso de inicio de sesión, desde la introducción de credenciales hasta la autenticación con JWT.
  - **Imágenes:**

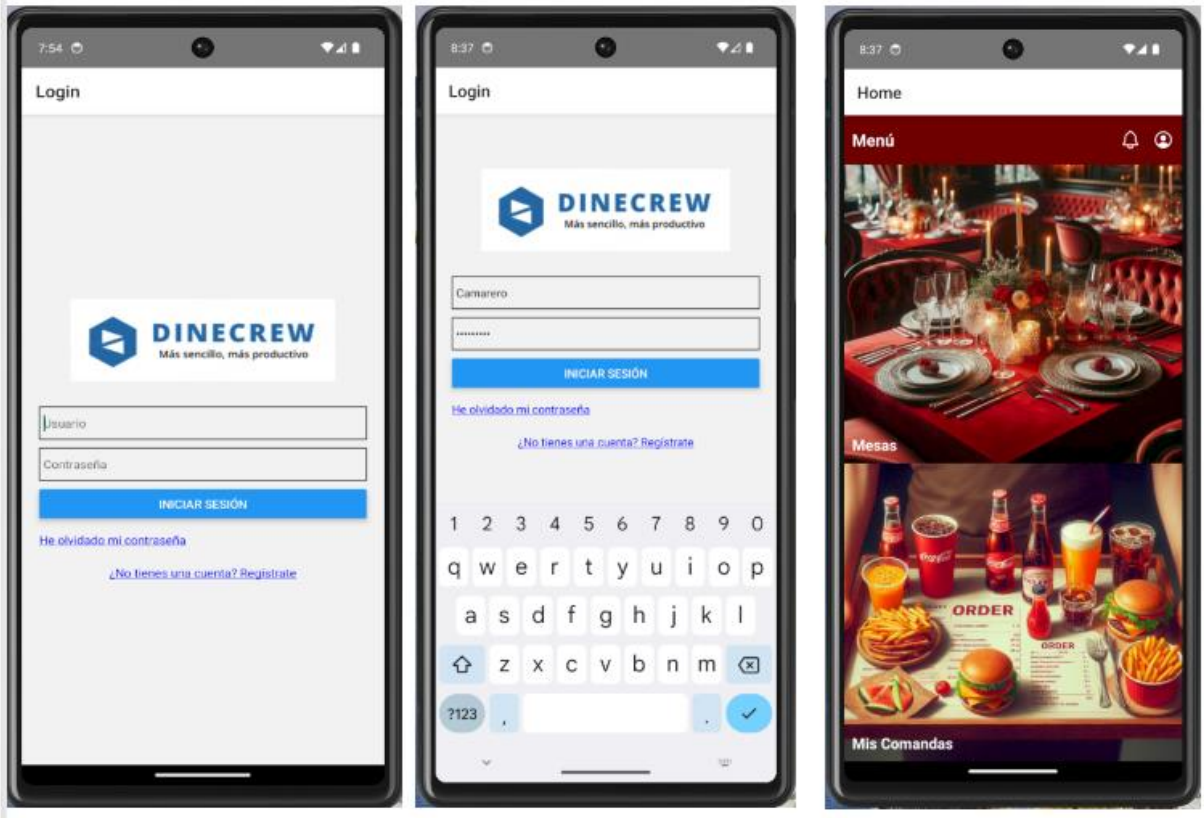

Ilustración 3-25 Flujo de Inicio de sesión

- **Storyboard 3: Restablecimiento de Contraseña**
  - **Descripción:** Muestra el flujo para el restablecimiento de contraseña, incluyendo la solicitud de un enlace de restablecimiento y el proceso de actualización de la contraseña.
  - **Imágenes:**

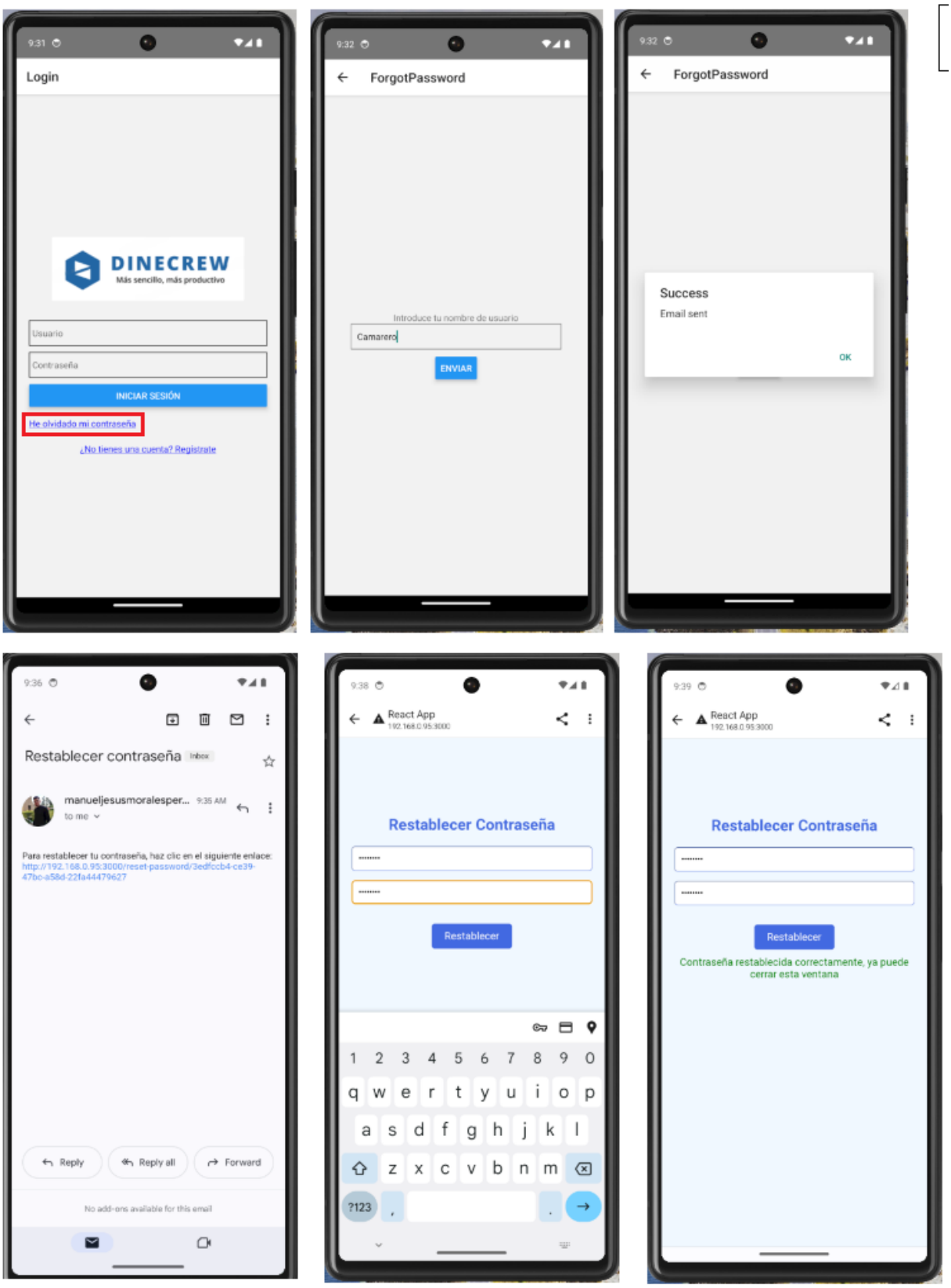

Ilustración 3-26 Flujo de restablecimiento de contraseña

- **Storyboard 4: Cierre de sesión**

- **Descripción:** Muestra el proceso para cerrar sesión en la aplicación y cómo se invalida el token JWT.
- **Imágenes:**

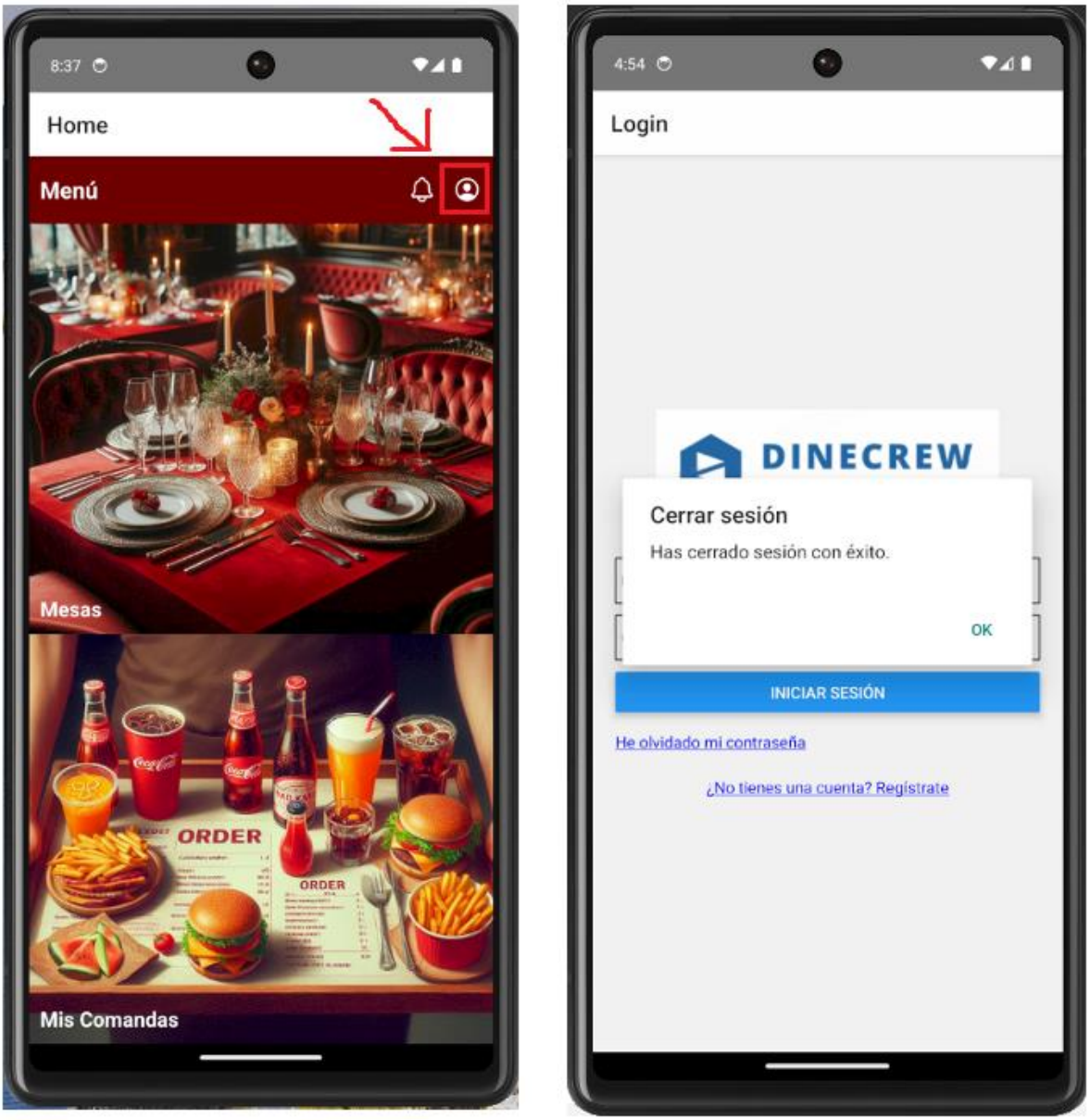

Ilustración 3-27 Flujo de cierre de sesión

3.2.3 Detalles de Implementación

En esta iteración se implementaron las siguientes funcionalidades clave:

- **Funcionalidades de Usuario:**

- **Registro de Usuarios:** Se desarrolló el backend para gestionar las solicitudes de registro, incluyendo la validación de datos y el almacenamiento seguro en MongoDB.
- **Inicio de Sesión:** Implementación de la autenticación segura utilizando JWT, con validación de credenciales.
- **Restablecimiento de Contraseña:** Desarrollo del sistema para enviar correos electrónicos de recuperación y manejo de la actualización de contraseñas encriptadas.
- **Cierre de sesión:** Implementación de la funcionalidad para que los usuarios puedan cerrar sesión, invalidando el token JWT.

- **Imágenes de Implementación:**

1. **Registro de usuarios**

```java
import com.dinecrew.dinecrewbackend.enums.Role;
import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.List;

24 usages  mjmp0027
@Document(collection = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {

    @Id
    private String id;
    private String username;
    private String email;
    private String password;
    private String resetPasswordToken;
    private List<String> mesasAsignadas; // Almacena los IDs de las mesas asignadas
    private Role role;
}
```

Ilustración 3-28 Clase Usuario [12]

```
mjmp0027
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserService userService;

    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private UserDetailsService userDetailsService;
```

Ilustración 3-29 Inyección de las clases necesarias para el controlador

Para el registro solo serán necesarias las clases **'UserService'** y '**PasswordEncoder**'.

- **UserService:** Se encarga de guardar el usuario en la base de datos y de comprobar que el nombre de usuario no existe.
- **PasswordEncoder:** Se encarga de codificar la contraseña para guardarla en la base de datos codificada.

```
mjmp0027
@PostMapping("/register")
public ResponseEntity<?> register(
        @RequestBody UserDto dto
) {
    Map<String, String> response = new HashMap<>();
    String username = dto.getUsername();
    String email = dto.getEmail();
    String password = dto.getPassword();
    Role role = dto.getRole();

    if (username == null || username.isEmpty()) {
        response.put("message", "El nombre de usuario no puede estar vacío");
        return ResponseEntity.status(400).body(response);
    }

    if (!isValidUsername(username)) {
        response.put("message", "El nombre de usuario contiene caracteres no permitidos");
        return ResponseEntity.status(400).body(response);
    }

    User userOpt = userService.findByUsername(username);

    if (userOpt != null) {
        response.put("message", "El nombre de usuario ya está en uso");
        return ResponseEntity.status(400).body(response);
    }

    if (email == null || email.isEmpty()) {
        response.put("message", "El email no puede estar vacío");
        return ResponseEntity.status(400).body(response);
    }

    if (!isValidEmail(email)) {
        response.put("message", "El email no es válido");
        return ResponseEntity.status(400).body(response);
    }
```

Ilustración 3-30 Endpoint [16] de registro de usuario

```
    if (password == null || password.isEmpty()) {
        response.put("message", "La contraseña no puede estar vacía");
        return ResponseEntity.status(400).body(response);
    }

    if (password.length() < 8) {
        response.put("message", "La contraseña debe tener al menos 8 caracteres");
        return ResponseEntity.status(400).body(response);
    }

    User user = User.builder()
            .username(username)
            .email(email)
            .password(passwordEncoder.encode(password))
            .role(role)
            .mesasAsignadas(new ArrayList<>())
            .build();

    if (userService.countUsers() == 0) {
        userService.save(user);
        response.put("message", "Primer usuario registrado");
        return ResponseEntity.ok(response);
    } else {
        user = userService.save(user);
        return ResponseEntity.ok(UserDto.fromDocument(user));
    }

}
```

Ilustración 3-31 Endpoint de registro de usuario

```
3 usages  mjmp0027
const useRegister = () : {...}  => {
  const navigation : ...  = useNavigation<NavigationProp<RootStackParamList>>();
  const [username : string , setUsername : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');
  const [email : string , setEmail : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');
  const [password : string , setPassword : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');
  const [confirmPassword : string , setConfirmPassword : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');
  const [role : string , setRole : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: 'Selecciona un rol');
  const [message : string , setMessage : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');
  const [numMesas : string , setNumMesas : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');
  const [modalVisible : boolean , setModalVisible : React.Dispatch<React.SetStateA... ] = useState<boolean>( initialState: false);

  1 usage  mjmp0027
  const handleRegister = async () : Promise<void>  => {
    const response : Response  = await fetch( input: 'http://10.0.2.2:8080/api/auth/register',  init: {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify( value: {username, email, password, role}),
    });

    if (response.ok) {
      const data = await response.json();
      const message = data.message;

      setMessage(message);
      if (message === 'Primer usuario registrado') {
        setModalVisible( value: true);
      }
      Alert.alert( title: 'Registro',  message: 'Usuario registrado correctamente');
      if (message !== 'Primer usuario registrado') {
        navigation.navigate('Login');
      }
    } else {
      const errorData = await response.json();
      setMessage( value: errorData.message || 'Error durante el registro');
    }
  };
```

Ilustración 3-32 Hook [17] de registro

2. Inicio de sesión

Para el inicio de sesión serán necesarias las clases '**AuthenticacionManager**', '**UserService**' y '**JwtUtil**'.

- **AuthenticationManager:** Se encarga de comprobar que el usuario y la contraseña son válidos pasando por una serie de filtros.
- **UserService:** Se encarga de buscar al usuario en la base de datos.
- **JwtUtil:** Se encarga de crear el token mediante el '**UserDetails**' para devolverlo en la respuesta y usarlo para el manejo de la interfaz.

```
mjmp0027
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserService userService;

    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private UserDetailsService userDetailsService;
```

Ilustración 3-33 Inyección de dependencias

```
mjmp0027 *
@PostMapping(value = "/login", consumes = "application/json")
public ResponseEntity<?> login(
        @RequestBody UserDto loginRequest
) {

    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    loginRequest.getUsername(),
                    loginRequest.getPassword()
            )
    );

    SecurityContextHolder.getContext().setAuthentication(authentication);
    UserDetails userDetails = (UserDetails) authentication.getPrincipal();
    String jwt = jwtUtil.generateToken(userDetails);

    User user = userService.findByUsername(userDetails.getUsername());
    return ResponseEntity.ok(new JwtResponse(jwt, user.getId()));
}
```

Ilustración 3-34 Endpoint de Inicio de sesión

Para realizar los siguientes pasos es necesario importar las librerías de '**JsonWebToken**' y '**SpringSecurity**' [8] [9]

```
<!-- JWT dependencies -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.2</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.2</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.2</version>
    <scope>runtime</scope>
</dependency>
```

Ilustración 3-35 Dependencias de JWT [13][14][15]

```
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>3.3.1</version>
</dependency>
```

Ilustración 3-36 Dependencia de Spring Security

```java
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;
import org.springframework.stereotype.Component;

import java.io.IOException;

 mjmp0027
@Component
public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private UserDetailsService userDetailsService;

     mjmp0027
    public JwtAuthorizationFilter(AuthenticationManager authManager) { super(authManager); }
```

Ilustración 3-37 Clase componente para manejar los filtros

```java
    1 usage  mjmp0027
    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {
        String token = request.getHeader( s: "Authorization");
        if (token != null && token.startsWith("Bearer ")) {
            token = token.substring( beginIndex: 7);
            try {
                Claims claims = Jwts.parserBuilder() JwtParserBuilder
                        .setSigningKey(JwtUtil.SECRET_KEY)
                        .build() JwtParser
                        .parseClaimsJws(token) Jws<Claims>
                        .getBody();

                String username = claims.getSubject();

                if (username != null) {
                    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
                    if (jwtUtil.validateToken(token, userDetails)) {
                        return new UsernamePasswordAuthenticationToken(userDetails, credentials: null, userDetails.getAuthorities());
                    }
                }
            } catch (Exception e) {
                return null;
            }
        }
        return null;
    }
}
```

Ilustración 3-38 Método de autenticación

```
import {useState} from 'react';
import {useAuth} from '../AuthContext';
import {useNavigation, NavigationProp} from '@react-navigation/native';
import {RootStackParamList} from '../types';

3 usages  mjmp0027
const useLogin = () : {...}  => {
  const {login} = useAuth();
  const navigation : ...  = useNavigation<NavigationProp<RootStackParamList>>();
  const [username : string , setUsername : React.Dispatch<React.SetStateA... ] = useState( initialState: '');
  const [password : string , setPassword : React.Dispatch<React.SetStateA... ] = useState( initialState: '');
  const [error : string , setError : React.Dispatch<React.SetStateA... ] = useState( initialState: '');

  1 usage  mjmp0027
  const handleLogin = async () : Promise<void>  => {
    const response : Response  = await fetch( input: 'http://10.0.2.2:8080/api/auth/login',  init: {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify( value: {username, password}),
    });
    const data = await response.json();
    if (response.ok) {
      await login(data.token, data.userId);
      navigation.navigate('Home');
    } else {
      setError( value: 'Usuario o contraseña incorrectos');
    }
  };

  return {
    username,
    setUsername,
    password,
    setPassword,
    error,
    handleLogin,
  };
};
```

Ilustración 3-39 Hook de inicio de sesión

El contexto de la aplicación se encargará de guardar el ‘token’ y el ‘userId’ devuelto por el backend al realizar el login tal y como se muestra en la ilustración 3.33. Este apartado se realizó con la ayuda de consultar la documentación de React Navigation [1] y para la siguiente funcionalidad se implementó el uso de AsycStorage [17] y también se usó la navegación [20] [21]

```
1 usage  mjmp0027
const login = async (token: string, userId: string) : Promise<void>  => {
  await AsyncStorage.setItem( key: 'token', token);
  await AsyncStorage.setItem( key: 'userId', userId);
  setUser(decodeJwtToken(token));
};
```

Ilustración 3-40 Login useContext

3. Restablecimiento de contraseña

Para restablecer la contraseña se ha habilitado el endpoint '**forgotPassword**' tal y como aparece a continuación:

```
mjmp0027
@PostMapping("/forgot-password")
public ResponseEntity<?> forgotPassword(
        @RequestBody Map<String, String> request
        )
{
    String username = request.get("username");
    userService.sendPasswordResetToken(username);
    return ResponseEntity.ok(new Response("Email enviado"));
}
```

Ilustración 3-41 Endpoint de contraseña olvidada

El método '**sendPasswordResetToken**' se encarga de configurar lo necesario para poder enviar el correo al usuario mediante el método '**sendEmail**' que se mostrará en la ilustración 3.39

```
1 usage  mjmp0027 *
public void sendPasswordResetToken(String username) {
    Optional<User> userOpt = repository.findByUsername(username);
    if (userOpt.isEmpty()) {
        throw new RuntimeException("Usuario no encontrado");
    }

    User user = userOpt.get();
    String token = UUID.randomUUID().toString();
    user.setResetPasswordToken(token);

    repository.save(user);

    String resetUrl = "http://192.168.0.95:3000/reset-password/" + token;
    String emailContent = "Para restablecer tu contraseña, haz clic en el siguiente enlace:\n" + resetUrl;

    try {
        emailService.sendEmail(user.getEmail(), subject: "Restablecer contraseña", emailContent);
    } catch (MessagingException e) {
        e.printStackTrace();
        throw new RuntimeException("Error al enviar el correo electrónico");
    }
}
```

Ilustración 3-42 Método enviar email

```
2 usages  mjmp0027
@Service
public class EmailService {

    @Autowired
    private JavaMailSender mailSender;

    1 usage  mjmp0027
    public void sendEmail(String to, String subject, String content) throws MessagingException {
        MimeMessage message = mailSender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(message, multipart: true, encoding: "UTF-8");

        helper.setFrom("manueljesusmoralesperez@gmail.com");
        helper.setTo(to);
        helper.setSubject(subject);
        helper.setText(content, html: true); // Configura el contenido como HTML

        mailSender.send(message);
    }
}
```

Ilustración 3-43 Configuración del correo

Para poder realizar envíos por correo a través del servidor será necesario agregar la dependencia '**spring-boot-starter-mail**' en la sección '**<dependencies>**'. [2] A continuación, se mostrará como queda el pom.xml:

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Ilustración 3-44 Dependencia mail

También será necesario actualizar nuestro archivo '**application.properties**' para la configuración del correo. A continuación, se muestra como queda:

```
# Configuración de la aplicación y la base de datos
spring.application.name=DineCrewBackend
spring.data.mongodb.uri=${SPRING_DATA_MONGODB_URI}
spring.data.mongodb.database=${SPRING_DATA_MONGODB_DATABASE}
logging.level.org.springframework=DEBUG
spring.jpa.hibernate.ddl-auto=update

# Mail
spring.mail.host=${SPRING_MAIL_HOST}
spring.mail.port=${SPRING_MAIL_PORT}
spring.mail.username=${SPRING_MAIL_USERNAME}
spring.mail.password=${SPRING_MAIL_PASSWORD}
spring.mail.properties.mail.smtp.auth=${SPRING_MAIL_PROPERTIES_MAIL_SMTP_AUTH}
spring.mail.properties.mail.smtp.starttls.enable=${SPRING_MAIL_PROPERTIES_MAIL_SMTP_STARTTLS_ENABLE}

# Conexiones externas
server.address=${SERVER_ADDRESS}
server.port=${SERVER_PORT}
```

Ilustración 3-45 Configuración application.properties

Para obtener la contraseña que se indica en '**spring.mail.password**' hay que realizar una serie de pasos.

1. **Activar verificación en dos pasos**
2. **Crea y gestiona contraseñas de aplicación.**

Para la recuperación de la contraseña se ha habilitado una aplicación web en React con Typescript con una única vista muy sencilla. Esta será la encargada de enviar una petición de nuevo al servidor mandando la nueva contraseña al endpoint que aparece a continuación:

```
import {useState} from 'react';
import {Alert} from 'react-native';

3 usages  mjmp0027
const useForgotPassword = () : {...}  => {
  const [username : string , setUsername : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');

  1 usage  mjmp0027
  const handleForgotPassword = async () : Promise<void>  => {
    const response : Response  = await fetch(
      input: 'http://10.0.2.2:8080/api/auth/forgot-password',
      init: {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify( value: {username}),
      },
    );

    const result = await response.json();

    if (response.ok) {
      Alert.alert( title: 'Success',  message: 'Email sent');
    } else {
      Alert.alert( title: 'Error', result.message);
    }
  };

  return {
    username,
    setUsername,
    handleForgotPassword,
  };
};

2 usages  mjmp0027
export default useForgotPassword;
```

Ilustración 3-46 Hook Contraseña olvidada

```
3 usages  mjmp0027 *
function ResetPassword() {
    const { token : string  } = useParams();
    const [newPassword : string , setNewPassword] = useState( initialState: '');
    const [confirmPassword : string , setConfirmPassword] = useState( initialState: '');
    const [message : string , setMessage] = useState( initialState: '');

    1 usage  mjmp0027 *
    const handleResetPassword = async () : Promise<void>  => {
        if (newPassword !== confirmPassword) {
            setMessage( value: "Las contraseñas no coinciden");
            return;
        }

        if (newPassword === '' || confirmPassword === '') {
            setMessage( value: "Por favor, rellena todos los campos");
            return;
        }

        if (newPassword.length < 8 || confirmPassword.length < 8) {
            setMessage( value: "La contraseña debe tener al menos 8 caracteres");
            return;
        }

        const response : Response  = await fetch( input: 'http://192.168.0.95:8080/api/auth/reset-password',  init: {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify( value: { token, newPassword }),
        });

        const data = await response.json();

        if (response.ok) {
            setMessage(data.message);
        } else {
            setMessage(data.message);
        }
```

Ilustración 3-47 Vista web restablecer contraseña

```java
mjmp0027
@PostMapping("/reset-password")
public ResponseEntity<?> resetPassword(
        @RequestBody Map<String, String> request
        )
{

    Map<String, String> response = new HashMap<>();

    String token = request.get("token");
    String newPassword = request.get("newPassword");
    if (newPassword == null || newPassword.isEmpty()) {
        return ResponseEntity.status(400).body(new Response("La contraseña no puede estar vacia"));
    }

    if (newPassword.length() < 8) {
        return ResponseEntity.status(400).body(new Response("La contraseña debe tener al menos 8 caracteres"));
    }

    try {
        userService.resetPassword(token, newPassword);
    } catch (RuntimeException e) {
        response.put("message", e.getMessage());
        return ResponseEntity.status(400).body(response);
    }
    response.put("message", "Contraseña restablecida correctamente, ya puede cerrar esta ventana");
    return ResponseEntity.ok(response);
}
```

Ilustración 3-48 Endpoint para restablecer la contraseña

```java
1 usage   mjmp0027 *
public void resetPassword(String token, String newPassword) {
    Optional<User> userOpt = repository.findByResetPasswordToken(token);
    if (userOpt.isEmpty()) {
        throw new RuntimeException("Ya ha pasado una hora desde que se solicitó el cambio de contraseña o " +
                "ya ha cambiado la contraseña, por favor solicita un nuevo enlace para restablecer tu " +
                "contraseña\n(vuelva a pulsar he olvidado mi contraseña)");
    }

    User user = userOpt.get();
    user.setPassword(passwordEncoder.encode(newPassword));
    user.setResetPasswordToken(null);

    repository.save(user);
}
```

Ilustración 3-49 Método para establecer la nueva contraseña

4. Cierre de sesión

Para el cierre de sesión se ha habilitado el endpoint '**logout**'. A continuación, se mostrará la implementación:

```java
mjmp0027 *
@PostMapping("/logout/{username}")
public ResponseEntity<?> logout(
        @PathVariable String username
) {
    Map<String, String> response = new HashMap<>();
    System.out.println("Logging out: " + username);
    response.put("message", "Sesión cerrada y mesas liberadas");
    return ResponseEntity.ok(response);
}
```

Ilustración 3-50 Endpoint de cierre de sesión

La interfaz es encargada de viajar a la pantalla de inicio y cerrar la sesión del usuario, tal y como se muestra en la ilustración 3.27.

```javascript
usage  mjmp0027
const handleLogout = async () : Promise<void>  => {
  const token : string | null  = await AsyncStorage.getItem( key: 'token');
  if (token) {
    const decodedToken = decodeJwtToken(token);
    const username = decodedToken.sub;
    const response : Response  = await fetch(
      input: `http://10.0.2.2:8080/api/auth/logout/${username}`,
      init: {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
      },
    );
    if (response.ok) {
      await logout();
      Alert.alert( title: 'Cerrar sesión',  message: 'Has cerrado sesión con éxito.');
      await messaging().unsubscribeFromTopic(username);
    } else {
      Alert.alert( title: 'Error',  message: 'Hubo un problema al cerrar sesión.');
    }
  }
};
```

Ilustración 3-51 Hook cerrar sesión

Una vez realizado el logout el contexto de la aplicación se encargará de eliminar el token y establecer el usuario a '**null**'.

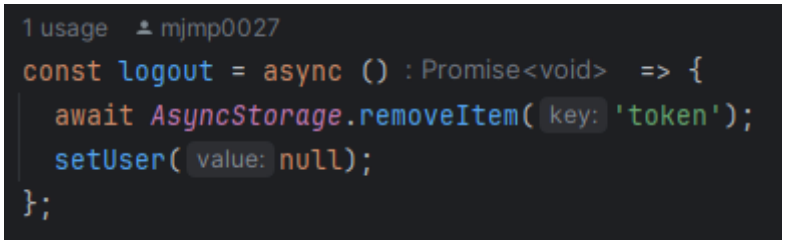

Ilustración 3-52 Logout

3.2.4 Pruebas de Verificación y Validación

En esta iteración, se realizaron pruebas para asegurar que las funcionalidades básicas estuvieran funcionando correctamente:

- **Pruebas Unitarias:** Se realizaron pruebas unitarias en los componentes de registro, inicio de sesión y restablecimiento de contraseña para asegurar que cada función individual funcionara según lo esperado.
- **Pruebas de Integración:** Estas pruebas aseguraron que los diferentes módulos interactuaran correctamente entre sí, como la conexión entre la base de datos y la lógica de negocio.
- **Pruebas de seguridad:** Se verificó que las contraseñas se encriptaran correctamente y que la autenticación con JWT funcionara de manera segura.
- **Imágenes de Pruebas:**

1. Pruebas de validación del registro

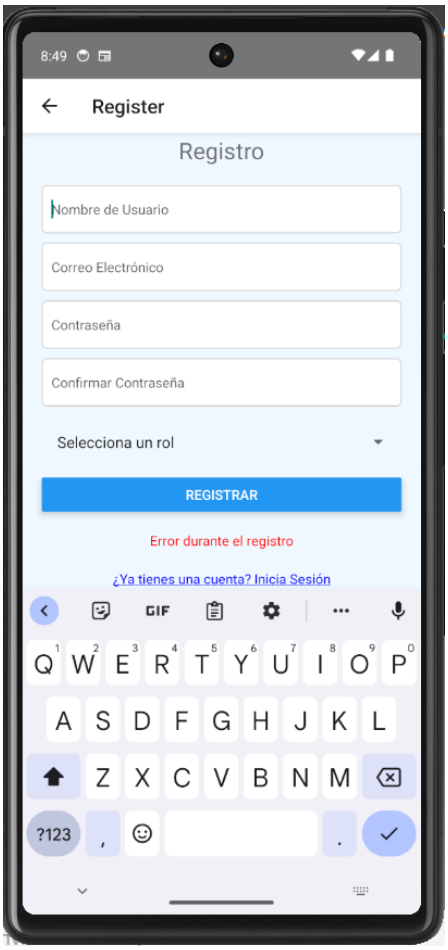

Ilustración 3-53 Prueba validación del registro

Se produce un error al deserializar [18] el objeto recibido en el servidor, ya que el rol del usuario solo puede ser ‘CAMARERO’ o ‘COCINERO’.

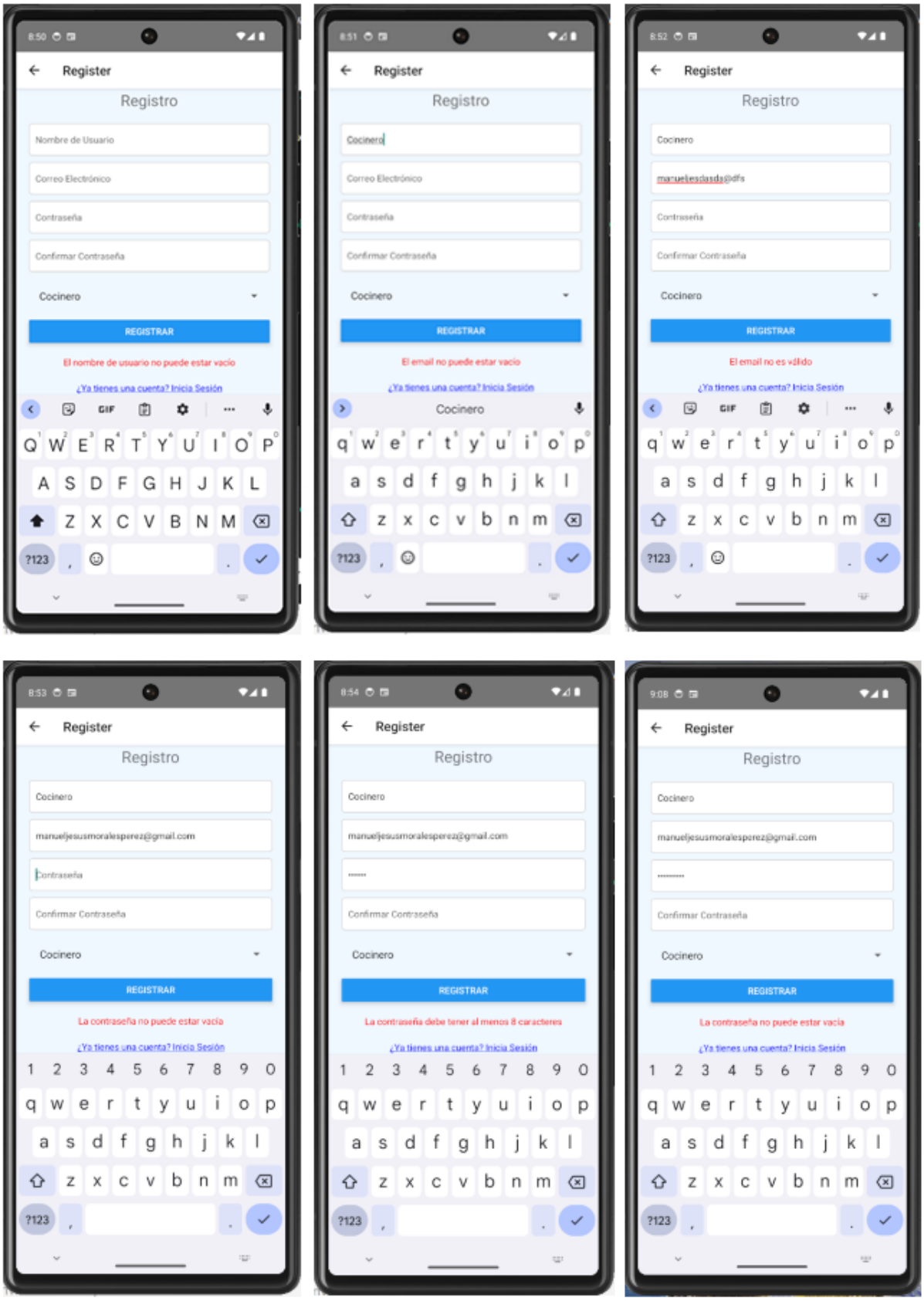

Ilustración 3-54 Pruebas registro

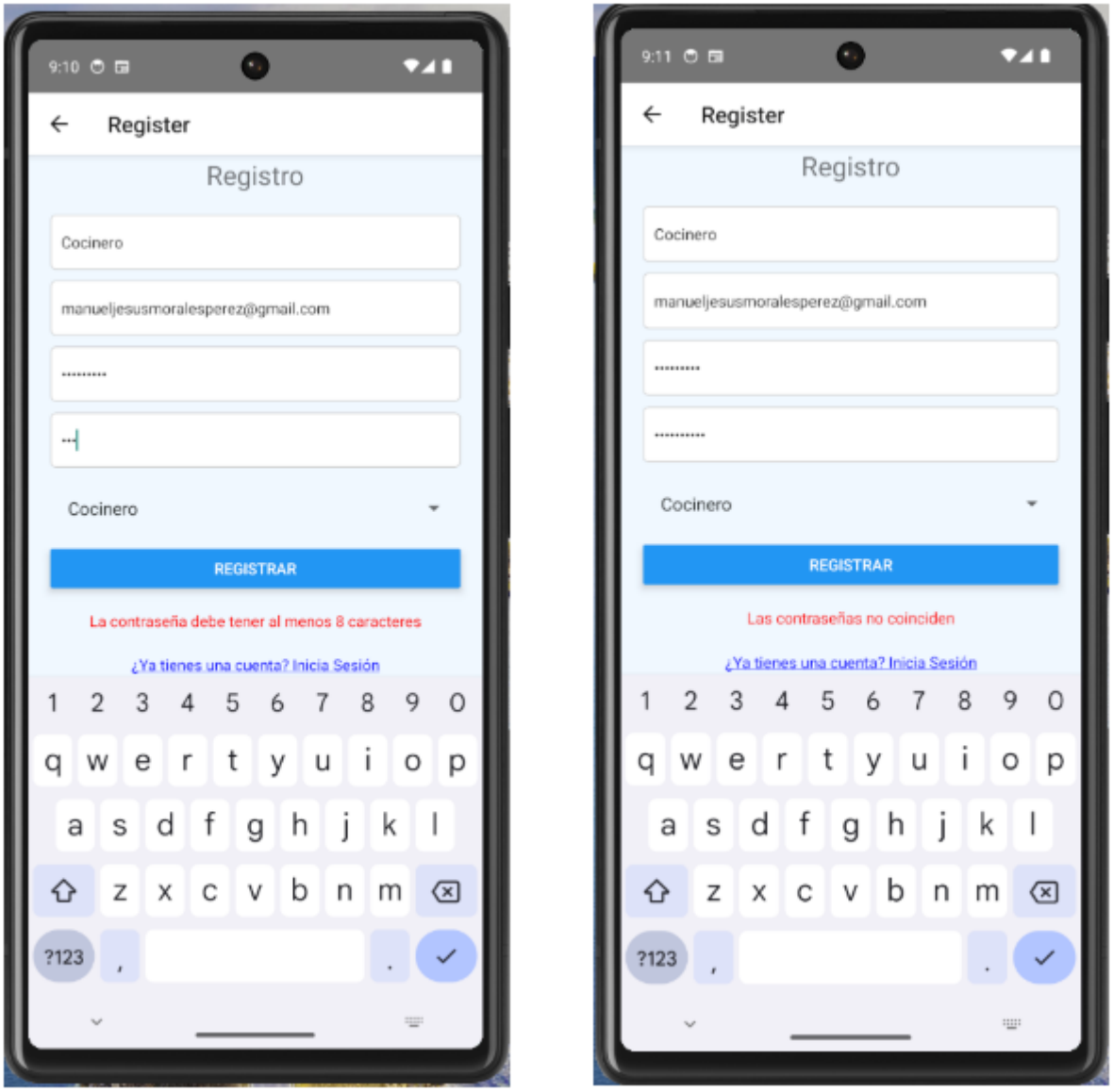

Ilustración 3-55 Pruebas registro

2. Pruebas de validación de inicio de sesión

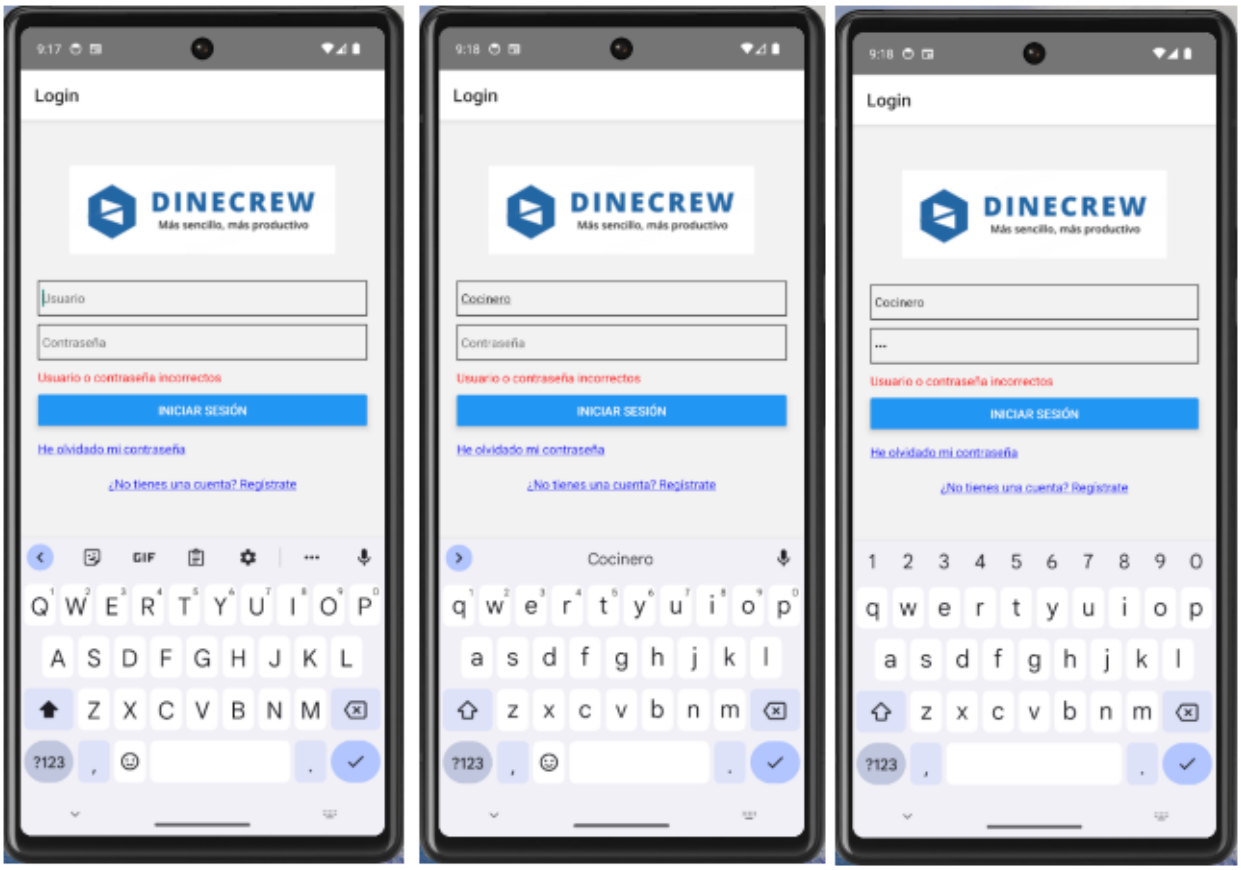

Ilustración 3-56 Pruebas inicio de sesión

3. Pruebas de restablecimiento de contraseña

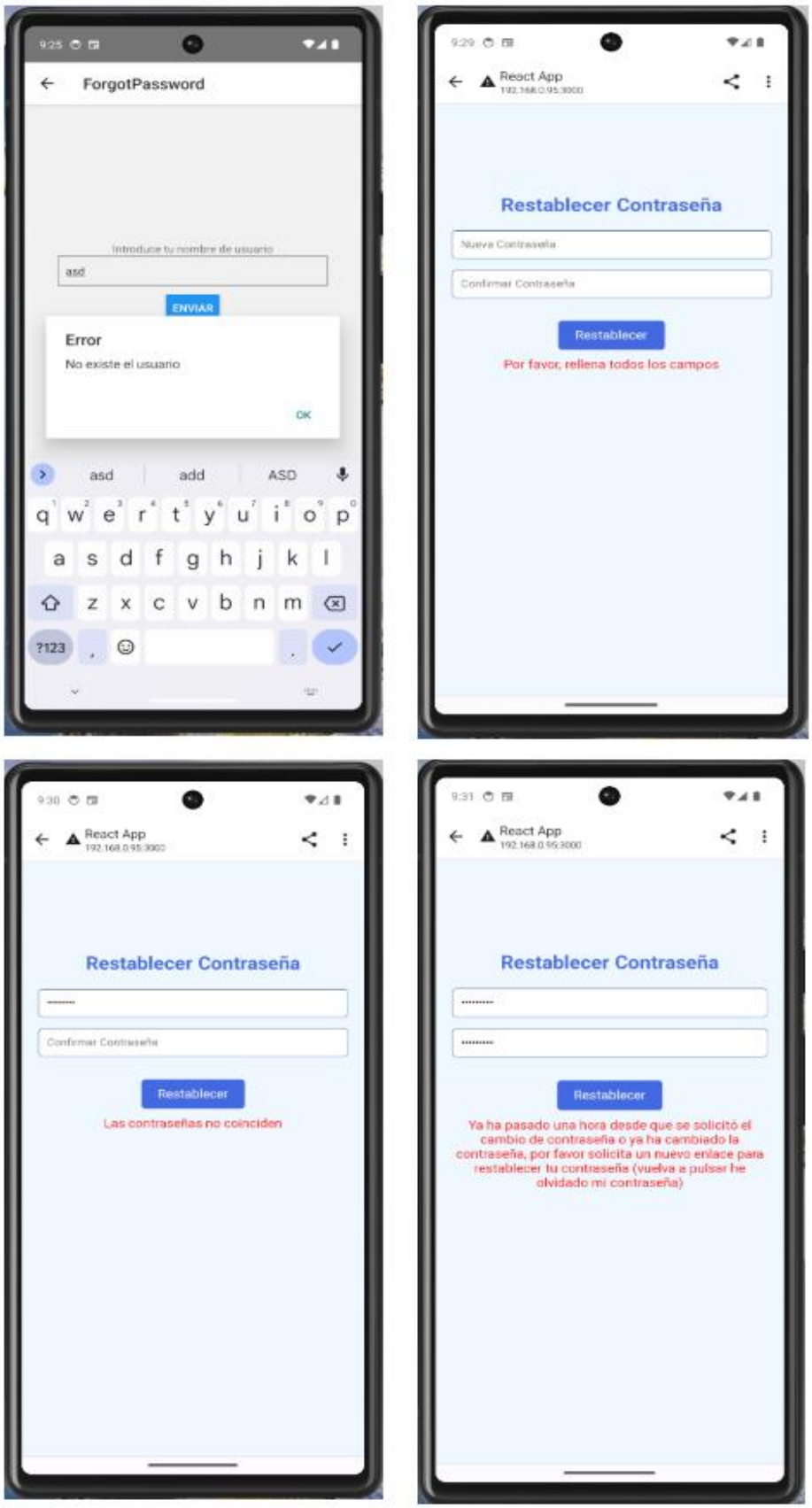

Ilustración 3-57 Pruebas restablecimiento de contraseña

4. **Pruebas de cierre de sesión**

Para este apartado no hay pruebas concluyentes ya que hay muy muy pocas posibilidades de fallo y es improbable que pase algo, aun así, está contemplado y se mostrará más adelante en la iteración 3.

3.2.5 Pruebas de Usabilidad del Sistema

Finalmente, se realizaron pruebas de usabilidad para evaluar la experiencia del usuario en las funcionalidades implementadas:

- **Pruebas de Registro y Login:** Se evaluó la facilidad de uso del formulario de registro y el proceso de inicio de sesión.
- **Pruebas de Recuperación de Contraseña:** Se evaluó la claridad del flujo de recuperación de contraseña desde la perspectiva del usuario.
- **Pruebas de Cierre de Sesión:** Se evaluó la facilidad y seguridad del proceso de logout.
- **Imágenes de Usabilidad:**

Para el siguiente gráfico nos encontramos las siguientes métricas:

- **Command:** La tasa promedio de comandos realizados por segundo durante el período de muestra seleccionado
- **Query:** La tasa promedio de consultas realizadas por segundo durante el período de muestra seleccionado
- **Update:** La tasa promedio de actualizaciones realizadas por segundo durante el período de muestra seleccionado
- **Delete:** La tasa promedio de eliminaciones realizadas por segundo durante el período de muestra seleccionado
- **GetMore:** La tasa promedio de “getMores” realizadas por segundo en cualquier cursor durante el período de muestra seleccionado. En un primario, este número puede ser alto incluso si el recuento de consultas es bajo, ya que los secundarios "getMore" del primario a menudo como parte de la replicación.

- **Insert:** La tasa promedio de inserciones realizadas por segundo durante el período de muestra seleccionado.

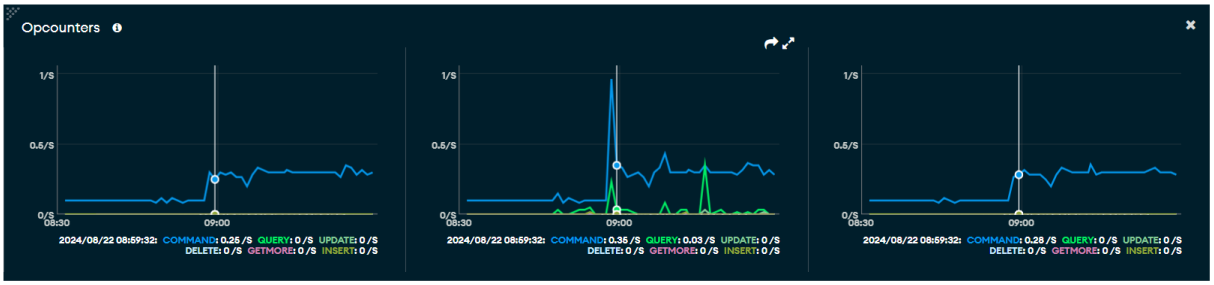

Ilustración 3-58 Gráfico Opcounters

Para el siguiente gráfico nos encontramos las siguientes métricas:

- **bytesIn:** La tasa promedio de bytes físicos (después de cualquier compresión por cable) enviados a este servidor de base de datos por segundo durante el período de muestra seleccionado.
- **bytesOut:** La tasa promedio de bytes físicos (después de cualquier compresión por cable) enviados desde este servidor de base de datos por segundo durante el período de muestra seleccionado.
- **NumRequests:** La tasa promedio de solicitudes enviadas a este servidor de base de datos por segundo durante el período de muestra seleccionado

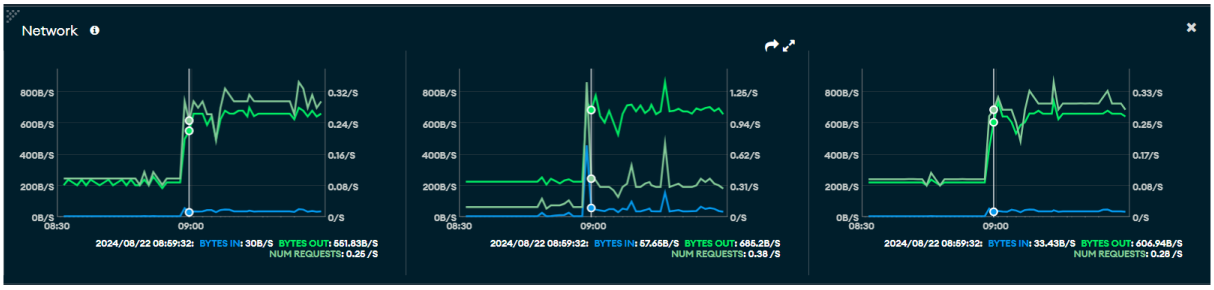

Ilustración 3-59 Gráfico Network

3.3 Tercera Iteración

En esta tercera iteración, el enfoque se centró en la gestión de pedidos y la visualización de las mesas asignadas a cada camarero, así como en la funcionalidad del rol de cocinero para gestionar el estado de los pedidos. Se desarrollaron y

perfeccionaron funcionalidades que permiten a los camareros seleccionar mesas, registrar pedidos, y visualizar productos, mientras que los cocineros pueden ver y actualizar el estado de los pedidos conforme avanza su preparación.

3.3.1 Historias de usuario

Durante esta iteración se abordaron las siguientes historias de usuario:

1. **Elegir Mesa**
    a. **Historia de usuario:** Como camarero quiero poder visualizar y seleccionar solo las mesas que tengo asignadas para gestionar los pedidos de manera eficiente.
    b. **Criterios de aceptación:**
        i. El camarero puede ver únicamente las mesas que le han sido asignadas.
        ii. Al seleccionar una mesa, el camarero puede proceder a registrar un nuevo pedido.
2. **Registrar Pedido/Comanda**
    a. **Historia de usuario:** Como camarero, quiero poder registrar un pedido para una mesa específica y agregar los detalles de los productos solicitados por los clientes.
    b. **Criterios de aceptación:**
        i. El camarero puede seleccionar una mesa y registrar un nuevo pedido.
        ii. El pedido incluye detalles como la mesa, los productos seleccionados, y las cantidades.
        iii. El pedido se guarda correctamente en la base de datos.
3. **Visualizar productos**
    a. **Historia de usuario:** Como camarero, quiero poder ver una lista de productos disponibles para poder registrar correctamente el pedido de un cliente.
    b. **Criterios de aceptación:**

i. El camarero puede acceder a una lista completa de productos desde la interfaz de registro de pedidos.

ii. Los productos se muestran con sus descripciones y precios.

4. **Modificar pedido en tiempo real**

   a. **Historia de usuario:** Como camarero, quiero poder modificar un pedido existente en tiempo real para reflejar cambios solicitados por los clientes antes de que el pedido sea procesado.

   b. **Criterios de aceptación:**

      i. El camarero puede acceder a un pedido existente y modificarlo.

      ii. Los cambios se reflejan inmediatamente en la base de datos y la interfaz de usuario.

5. **Ver cuenta:**

   a. **Historia de usuario:** Como camarero, quiero poder visualizar la cuenta total acumulada de una mesa específica para poder mostrársela al cliente.

   b. **Criterios de aceptación:**

      i. El camarero puede seleccionar una mesa y ver el total acumulado de los pedidos realizados.

      ii. La cuenta total se muestra claramente con los detalles de los productos y precios.

6. **Actualizar estado de pedidos**

   a. **Historia de usuario:** Como cocinero, quiero poder actualizar el estado de los pedidos de 'pendiente' a 'en preparación' y de 'en preparación' a 'listo' para que los camareros sepan cuándo los pedidos están listos para ser servidos.

   b. **Criterios de aceptación:**

i. El cocinero puede ver la lista de pedidos con el estado actual (pendiente, en preparación, listo).

ii. El cocinero puede cambiar el estado de un pedido de 'pendiente' a 'en preparación'.

iii. El cocinero puede actualizar el estado de un pedido de 'en preparación' a 'listo'.

3.3.2 Storyboards

Para ilustrar el flujo de trabajo de las funcionalidades implementadas en esta iteración, se han creado los siguientes storyboards:

- **Storyboard 1: Selección de Mesas (Camarero)**
  - **Descripción:** Este storyboard muestra cómo el camarero puede visualizar y seleccionar las mesas que le han sido asignadas.
  - **Imágenes:**

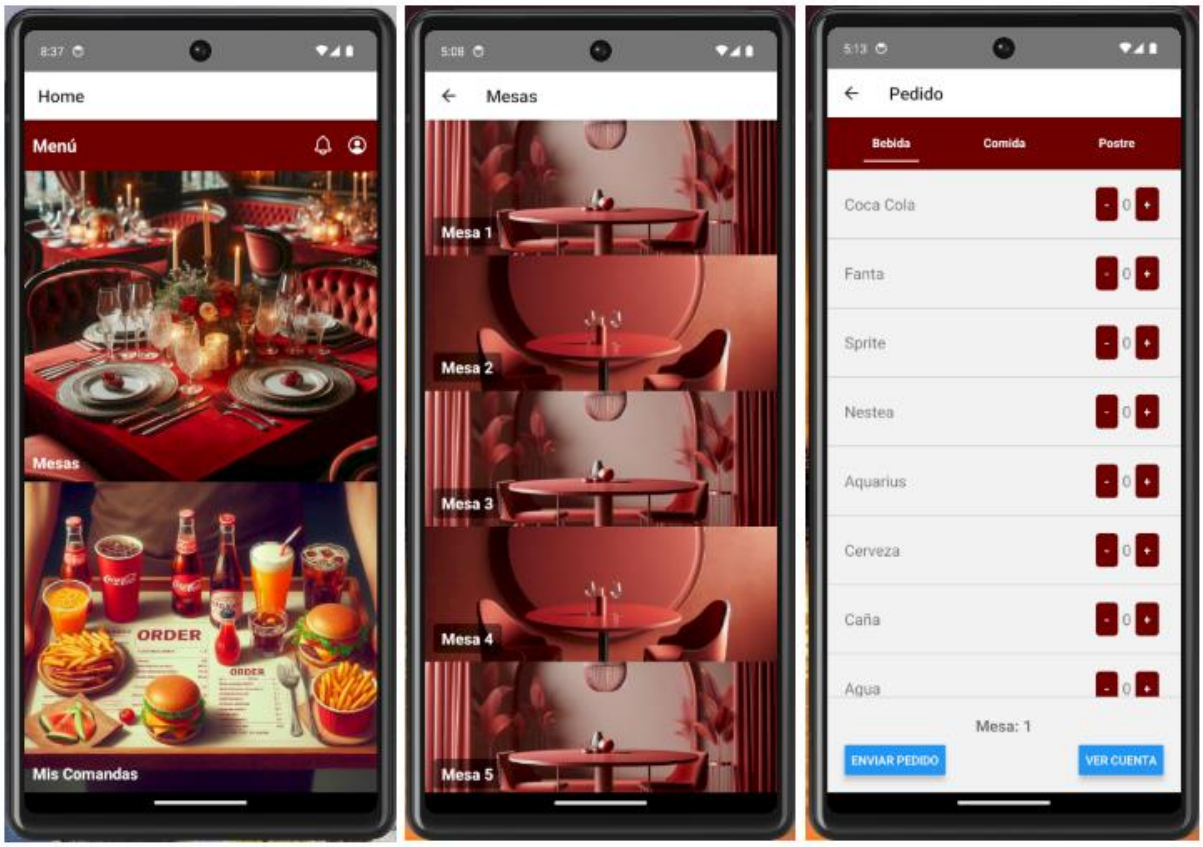

Ilustración 3-60 Flujo selección de mesa

- **Storyboard 2: Registro de Pedido (Camarero)**
    - **Descripción:** Muestra la secuencia de pantallas y acciones necesarias para que un camarero registre un pedido, incluyendo la selección de productos.
    - **Imágenes:**

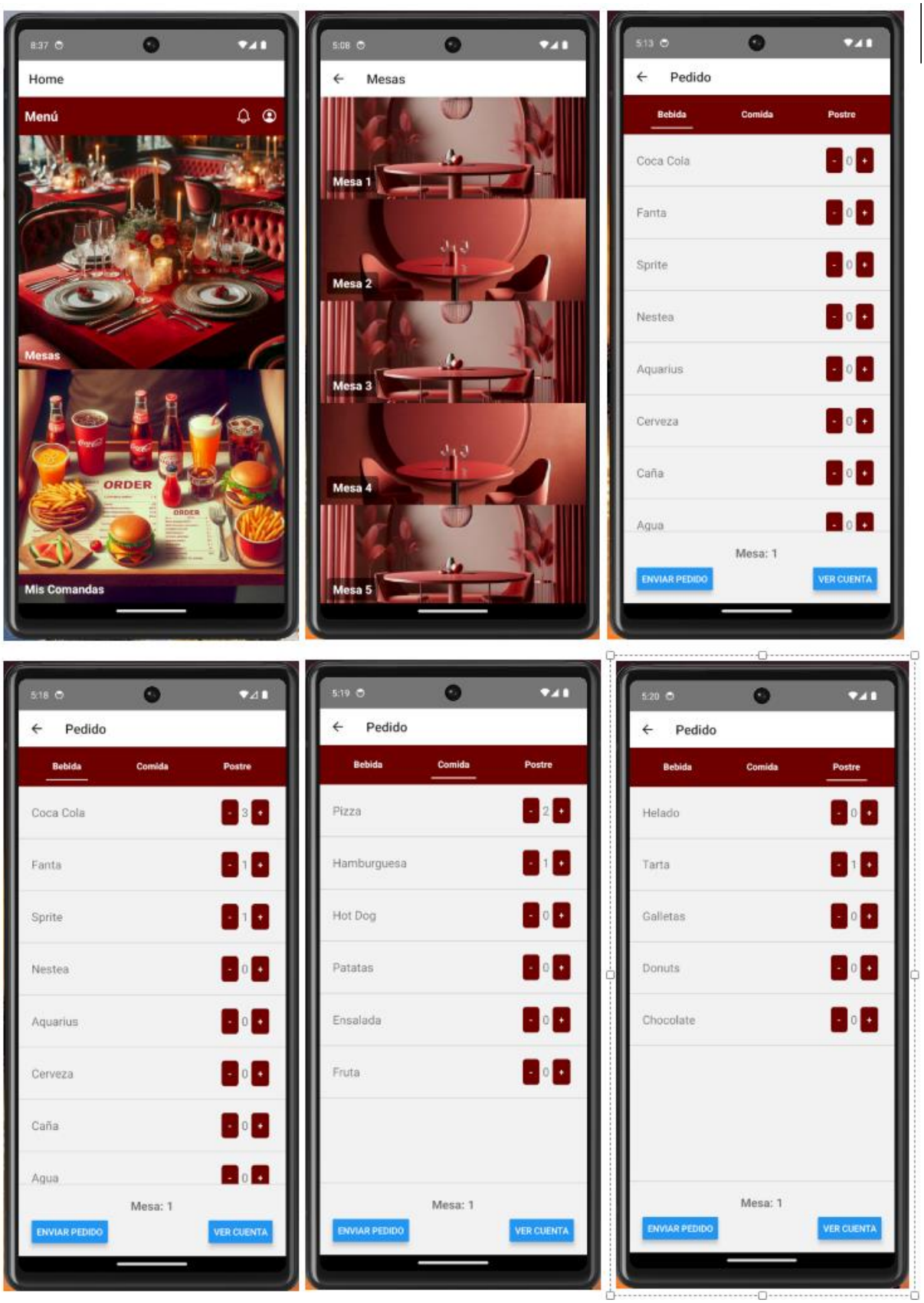

Ilustración 3-61 Flujo registrar un pedido

Ilustración 3-62 Flujo registrar un pedido

- **Storyboard 3: Modificación de Pedido (Camarero)**
  - **Descripción:** Detalla cómo un camarero puede modificar un pedido en tiempo real, desde la visualización del pedido hasta la actualización de los detalles.
  - **Imágenes:**

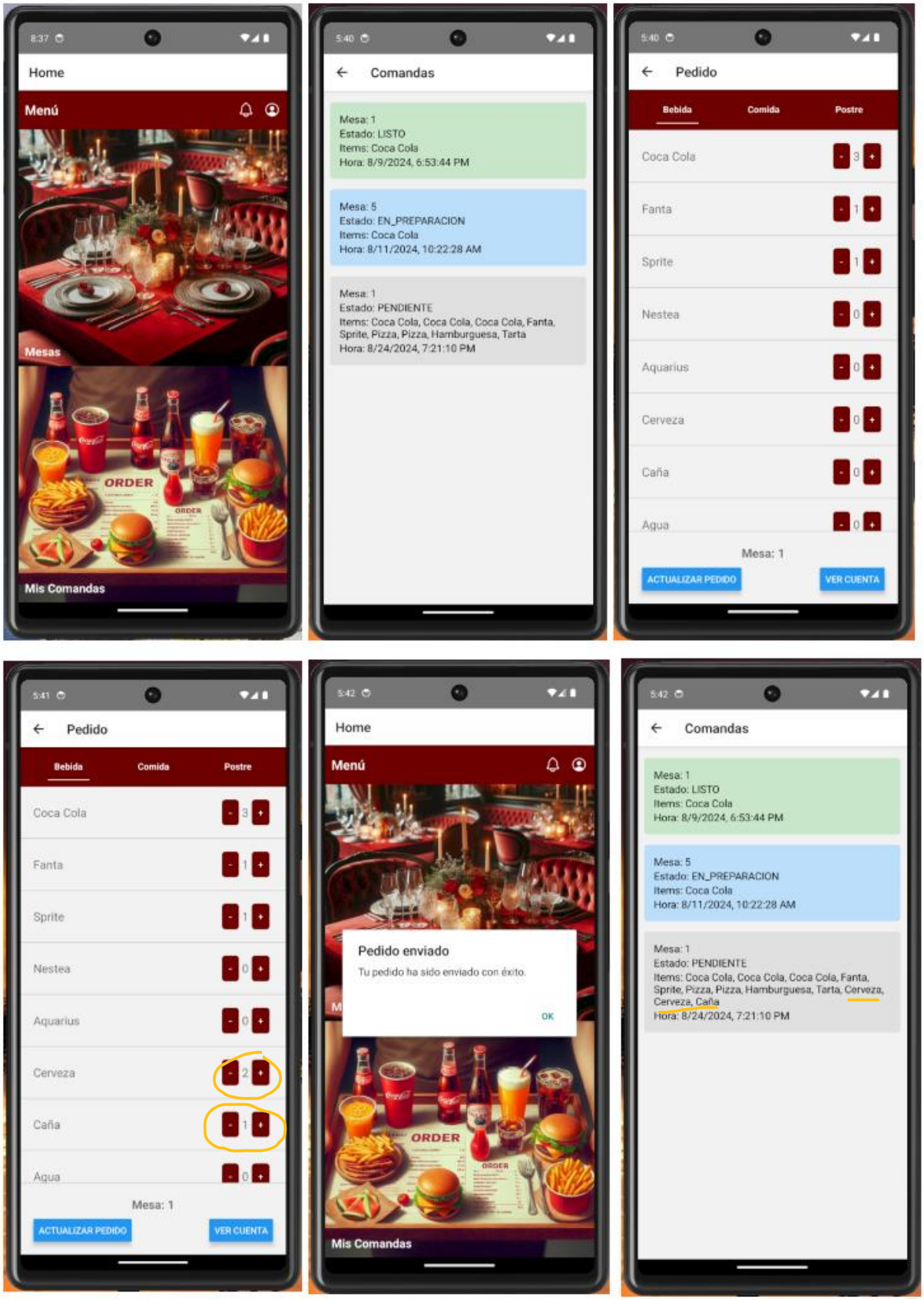

Ilustración 3-63 Flujo modificar un pedido

- **Storyboard 4: Ver cuenta (Camarero)**
  - **Descripción:** Muestra cómo el camarero puede visualizar la cuenta acumulada de una mesa específica y los detalles de los productos que conforman la cuenta.
  - **Imágenes:**

Para ver mejor esta funcionalidad se han realizado varios pedidos para la mesa 1

Ilustración 3-64 Ver cuenta

La pregunta '¿Desea eliminar estos pedidos?' es para no borrar los pedidos en caso de que la mesa todavía no haya pagado. Una vez el camarero esté seguro de que la mesa ha pagado podrá eliminar los pedidos dejando más limpia la base de datos y la vista de pedidos ya que no serán necesarios. Al

pulsar el botón 'Eliminar pedidos', la cuenta de esa mesa se guardará en la base de datos, guardando el precio y la hora a la que realizó para llevar un conteo de las ventas y a su vez se eliminarán todos los pedidos de esa mesa.

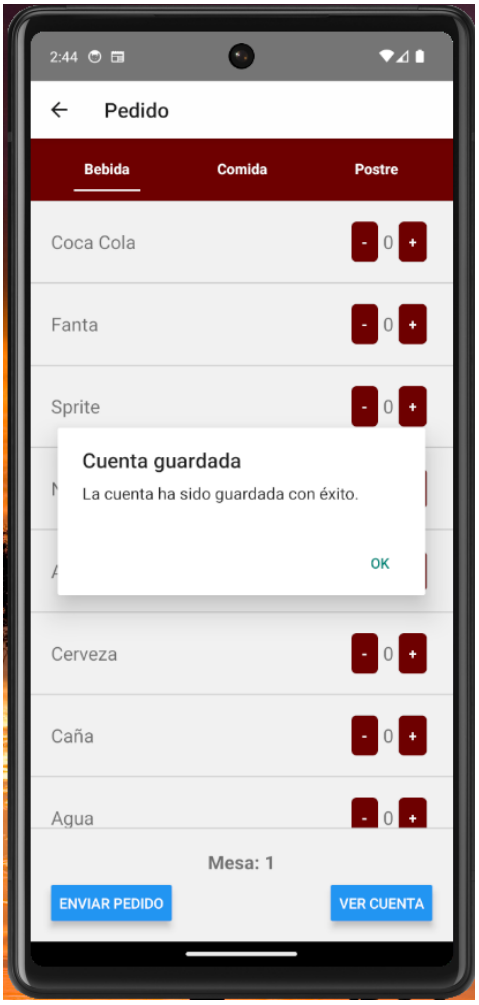

Ilustración 3-65 Servicio realizado

- **Storyboard 5: Actualización de Estado de Pedido (Cocinero)**
  - **Descripción:** Muestra cómo el cocinero interactúa con la lista de pedidos para actualizar su estado conforme avanza la preparación.
  - **Imágenes:**

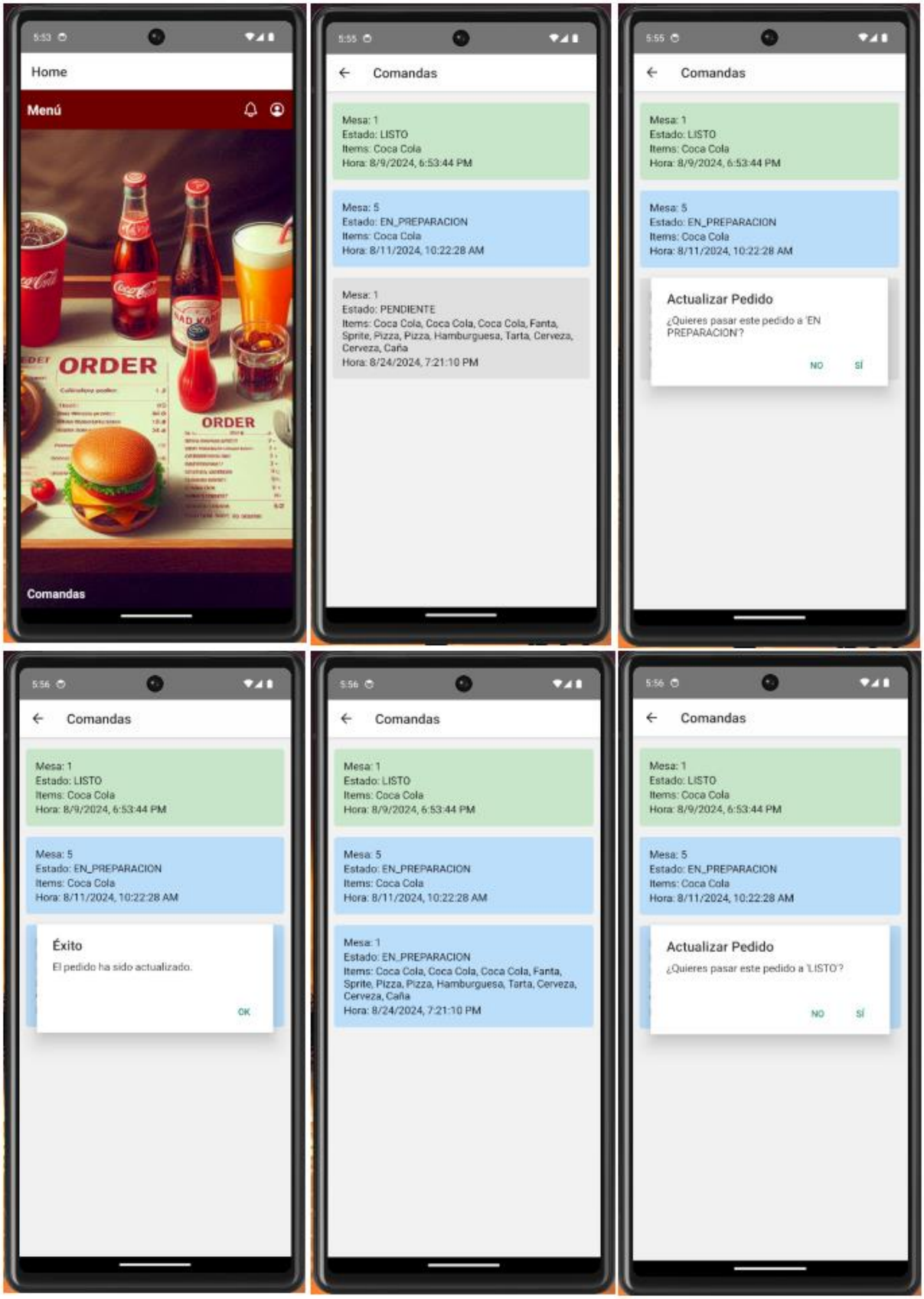

Ilustración 3-66 Flujo actualizar estado de pedido

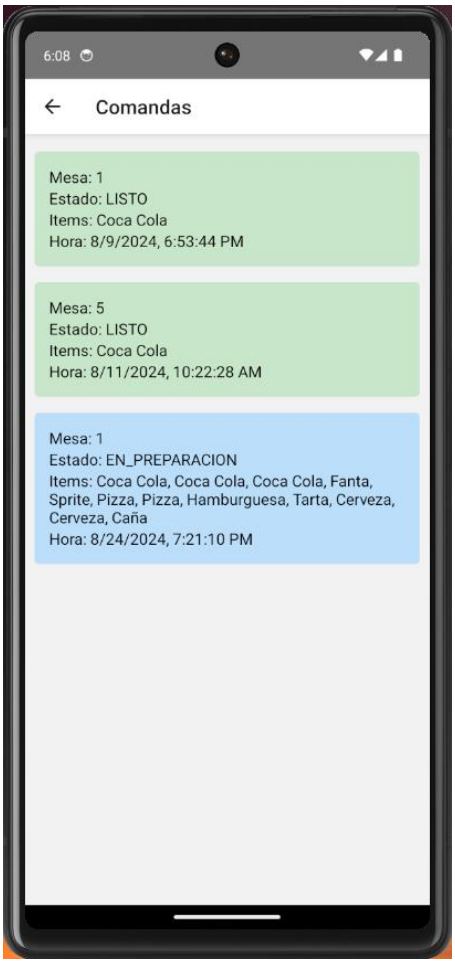

Ilustración 3-67 Flujo actualizar estado de pedido

3.3.3 Detalles de Implementación

En esta iteración se implementaron las siguientes funcionalidades clave:

- **Gestión de mesas:**
  - Visualización de las mesas asignadas a cada camarero, permitiendo una gestión más eficiente de las mismas.
  - Implementación de la lógica para asegurar que solo se muestran las mesas asignadas al camarero que ha iniciado sesión.
- **Registro y Modificación de pedidos:**
  - Desarrollo de la funcionalidad para que los camareros puedan registrar nuevos pedidos para las mesas seleccionadas, incluyendo la selección de productos desde un catálogo disponible.

  - Implementación de la funcionalidad que permite a los camareros modificar los pedidos en tiempo real, permitiendo agregar, eliminar o cambiar productos antes de que el pedido sea procesado.

- **Visualización de productos:**
  - Creación de una lista de productos que los camareros pueden consultar al registrar un pedido.
- **Gestión de Estado de pedidos:**
  - Implementación de la interfaz y lógica que permite a los cocineros visualizar todos los pedidos pendientes y actualizar su estado conforme se avanza en su preparación.
  - Configuración para que los cocineros puedan cambiar el estado de los pedidos de 'pendiente' a 'en preparación', y posteriormente a 'listo'.
- **Base de datos:**
  - Actualización de la base de datos para soportar las nuevas funcionalidades de pedidos, estados de pedidos, y la gestión de mesas.
- **Imágenes de implementación:**

1. **Gestión de mesas**

```
20 usages  mjmp0027
@Document(collection = "mesas")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Mesa {

    @Id
    private String id;
    private String numero;
    private String userId; // Id del camarero asignado (null si está libre)
}
```

Ilustración 3-68 Clase mesa

```
mjmp0027
@RestController
@RequestMapping("/api/mesas")
public class MesaRestController {

    @Autowired
    private MesaService mesaService;

    mjmp0027
    @GetMapping(value = {"/", ""})
    public ResponseEntity<?> getMesas() {
        List<Mesa> mesas = mesaService.findAll();
        return ResponseEntity.ok(MesaDto.fromDocuments(mesas));
    }
```

Ilustración 3-69 Controlador mesa

```
import {Mesa} from '../types/Mesa';

3 usages  mjmp0027
const fetchMesas = async (token: string): Promise<Mesa[]> => {
  try {
    const response : Response  = await fetch( input: 'http://10.0.2.2:8080/api/mesas', init: {
      method: 'GET',
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    if (response.ok) {
      const data = await response.json();
      return data;
    } else {
      console.log('Failed to fetch mesas');
      return [];
    }
  } catch (error) {
    console.error('Error fetching mesas:', error);
    return [];
  }
};

2 usages  mjmp0027
export default fetchMesas;
```

Ilustración 3-70 Obtener mesas

```
3 usages  mjmp0027
const useMesas = (
  token: string | null,
  decodedToken: any,
  isTokenLoaded: boolean,
) : {mesas: ..., userId: string}  => {
  const [mesas : Mesa[] , setMesas : React.Dispatch<React.SetStateA... ] = useState<Mesa[]>( initialState: []);
  const [userId : string , setUserId : React.Dispatch<React.SetStateA... ] = useState<string>( initialState: '');

  useEffect( effect: () : void  => {
    if (isTokenLoaded && token && decodedToken) {
      1 usage  mjmp0027
      const fetchUserData = async () : Promise<void>  => {
        const id : string | null  = await fetchUserId(decodedToken.sub, token);
        if (id) {
          setUserId(id);
          const mesasData : Mesa[]  = await fetchMesas(token);
          setMesas(mesasData);
        }
      };

      fetchUserData();
    }
  }, deps: [isTokenLoaded, token, decodedToken]);
  return {mesas, userId};
};

2 usages  mjmp0027
export default useMesas;
```

Ilustración 3-71 Hook obtener mesas

Se guardan las mesas junto al id del usuario que hay logueado en estos momentos para el siguiente funcionamiento:

```
type MesaListProps = {
  mesas: Mesa[];
  userId: string;
  navigation: any;
};

3 usages  mjmp0027
const MesaList: React.FC<MesaListProps> = ({mesas : Mesa[] , userId : string , navigation}) => {
  1 usage  mjmp0027
  const renderItem = ({item}: {item: Mesa}) : null | Element  => {
    const isClickable : boolean  = item.userId === userId;
    if (!isClickable) {
      return null;
    }
```

Ilustración 3-72 Componente renderizar las mesas

En el login se ha agregado lo necesario para que al iniciar sesión se asignen mesas de las que hay libres en la base datos

```
mjmp0027
@PostMapping(value = "/login", consumes = "application/json")
public ResponseEntity<?> login(
        @RequestBody UserDto loginRequest
) {

    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    loginRequest.getUsername(),
                    loginRequest.getPassword()
            )
    );

    SecurityContextHolder.getContext().setAuthentication(authentication);
    UserDetails userDetails = (UserDetails) authentication.getPrincipal();
    String jwt = jwtUtil.generateToken(userDetails);

    //Si el usuario tiene el rol cocinero no se le asignan mesas
    User user = userService.findByUsername(userDetails.getUsername());
    if (user.getRole() == Role.COCINERO) {
        return ResponseEntity.ok(new JwtResponse(jwt, user.getId()));
    }

    userService.asignarMesas(userDetails.getUsername());

    return ResponseEntity.ok(new JwtResponse(jwt, user.getId()));
}
```

Ilustración 3-73 Endpoint inicio de sesión

Por ahora la aplicación asigna 5 mesas libres a cada camarero tal y como se indica a continuación:

```
1 usage  mjmp0027
public void asignarMesas(String username) {
    User user = repository.findByUsername(username).orElseThrow(() -> new RuntimeException("Usuario no encontrado"));
    List<Mesa> mesasLibres = mesaService.findAllByUserId(null);

    if (user.getMesasAsignadas().isEmpty()) {
        if (mesasLibres.isEmpty()) {
            throw new RuntimeException("No hay mesas disponibles");
        }

        List<String> mesasAsignadasList = new ArrayList<>(); // Lista con los IDs de las mesas asignadas
        // De las mesas libres se asignarán hasta un máximo de 5 mesas por camarero
        int mesasAsignadas = 0;
        for (Mesa mesa : mesasLibres) {
            if (mesasAsignadas >= 5) {
                break;
            }
            mesa.setUserId(user.getId());
            mesa = mesaService.save(mesa);
            mesasAsignadasList.add(mesa.getId());
            mesasAsignadas++;
        }

        user.setMesasAsignadas(mesasAsignadasList);
        repository.save(user);
    }
}
```

Ilustración 3-74 Asignación de mesas

En el caso de que la aplicación llegase a mejorarse, se implementaría un algoritmo que calcule los camareros que hay en cada turno y las mesas totales en ese turno y se asignarían las mesas equitativamente a cada camarero conforme iban iniciando sesión. Esto no se ha hecho porque todavía no se ha implementado nada de horarios ni turnos de trabajo por lo que es solo un ejemplo sencillo.

También se ha agregado una línea al endpoint de cerrar sesión de liberación de mesas tal y como se muestra a continuación:

```
mjmp0027
@PostMapping("/logout/{username}")
public ResponseEntity<?> logout(
        @PathVariable String username
) {
    Map<String, String> response = new HashMap<>();
    System.out.println("Logging out: " + username);
    userService.liberarMesas(username);
    response.put("message", "Sesión cerrada y mesas liberadas");
    return ResponseEntity.ok(response);
}
```

Ilustración 3-75 Endpoint cerrar sesión

```
1 usage  mjmp0027
public void liberarMesas(String username) {
    User user = repository.findByUsername(username).orElseThrow(() -> new RuntimeException("Usuario no encontrado"));
    List<Mesa> mesasAsignadas = mesaService.findAllByUserId(user.getId());

    mesasAsignadas.forEach(mesa -> mesa.setUserId(null));
    mesaService.saveAll(mesasAsignadas);

    user.setMesasAsignadas(new ArrayList<>());
    repository.save(user);
}
```

Ilustración 3-76 Liberación de mesas

2. Registro y modificación de pedidos

```
20 usages  mjmp0027
@Document(collection = "pedidos")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Pedido {

    @Id
    private String id;
    private String mesa;
    private List<String> items;
    private State estado;

    @CreatedDate
    private LocalDateTime timestamp;
}
```

Ilustración 3-77 Clase pedido

```
mjmp0027
@PostMapping
public ResponseEntity<?> crearPedido(
        @RequestBody PedidoDtoIn dto
) {
    // Se comprueba que viene la mesa
    if (dto.getMesa() == null || dto.getMesa().isEmpty()) {
        return ResponseEntity.status(400).body("Debe indicar la mesa del pedido");
    }

    // Se comprueba que el pedido tiene al menos un producto
    if (dto.getItems().isEmpty()) {
        return ResponseEntity.status(400).body("Debe haber al menos un producto en el pedido");
    }

    // Se crea el pedido con estado PENDIENTE y se guarda en la base de datos
    Pedido pedido = Pedido.builder()
            .estado(State.PENDIENTE)
            .items(dto.getItems())
            .mesa(dto.getMesa())
            .timestamp(LocalDateTime.now())
            .build();

    pedido = pedidoService.crearPedido(pedido);

    return ResponseEntity.status(201).body(PedidoDtoOut.fromDocument(pedido));
}
```

Ilustración 3-78 Endpoint registrar pedido

```
1 usage  mjmp0027 *
const sendPedido = async () : Promise<void>  => {
  const token : string | null  = await AsyncStorage.getItem( key: 'token');
  if (token) {
    const pedido : {mesa: string, items: any[]}  = {
      mesa: mesa,
      items: Object.entries(selectedItems).flatMap( callback: ([item : string , count : number ]) =>
        Array(count).fill(item),
      ),
    };

    const response : Response  = await fetch(
      input: `http://10.0.2.2:8080/api/pedidos${editing ? `/${id}` : ''}`,
      init: {
        method: editing ? 'PUT' : 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify(pedido),
      },
    );

    if (response.ok) {
      Alert.alert( title: 'Pedido enviado',  message: 'Tu pedido ha sido enviado con éxito.');
      navigation.goBack();
      if (editing) {
        navigation.goBack();
      }
    } else {
      Alert.alert( title: 'Error',  message: 'Hubo un problema al enviar el pedido.');
    }
  }
};
```

Ilustración 3-79 Enviar pedido

```java
// Endpoint para actualizar los productos de un pedido
mjmp0027
@PutMapping("/{id}")
public ResponseEntity<?> actualizarPedido(
        @PathVariable String id,
        @RequestBody PedidoDtoIn dto
) {
    Optional<Pedido> pedidoOpt = pedidoService.obtenerPedido(id);
    if (pedidoOpt.isEmpty()) {
        return ResponseEntity.status(404).body("El pedido con el id: " + id + " no se ha encontrado");
    }

    Pedido pedido = pedidoOpt.get();

    if (dto.getMesa() == null || dto.getMesa().isEmpty()) {
        return ResponseEntity.status(400).body("Debe indicar la mesa del pedido");
    }

    if (dto.getItems().isEmpty()) {
        return ResponseEntity.status(400).body("Debe haber al menos un producto en el pedido");
    }

    pedido.setMesa(dto.getMesa());
    pedido.setItems(dto.getItems());

    pedido = pedidoService.crearPedido(pedido);
    return ResponseEntity.status(200).body(PedidoDtoOut.fromDocument(pedido));
}
```

Ilustración 3-80 Endpoint modificar pedido

3. Visualización de productos

Para esto se ha habilitado un endpoint que devuelve un archivo .csv con los productos, se ha escogido así para la facilitación del usuario a la hora de agregar productos, ya que se puede modificar el archivo .csv fácilmente. Para la implementación de esta funcionalidad se ha obtenido la información de la web [22] [23]

```java
mjmp0027
@RestController
@RequestMapping("/api/productos")
public class ProductosRestController {

    mjmp0027
    @GetMapping
    public ResponseEntity<Resource> getProductosCsv() {
        try {
            Resource resource = new ClassPathResource("static/productos.csv");
            return ResponseEntity.ok()
                    .header(HttpHeaders.CONTENT_DISPOSITION, ...headerValues: "attachment; filename=productos.csv")
                    .contentType(MediaType.parseMediaType("text/csv"))
                    .body(resource);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }
}
```

Ilustración 3-81 Endpoint lectura de productos

```tsx
useEffect( effect: () : void => {
  1 usage  mjmp0027
  const fetchProducts = async () : Promise<void> => {
    await fetchCsvData( url: 'http://10.0.2.2:8080/api/productos', setProducts);
    setIsLoading( value: false);
  };
  fetchProducts();
}, deps: []);

useEffect( effect: () : void => {
  if (items && Array.isArray(items) && items.length > 0) {
    const initialItemsCount : {[p: string]: number} = items.reduce((acc : {[p: string]: number} , item : string ) => {
      acc[item] = (acc[item] || 0) + 1;
      return acc;
    }, {} as {[key: string]: number});
    setSelectedItems(initialItemsCount);
  }
}, deps: [items]);

1 usage  mjmp0027
const incrementItem = (item: string) : void => {
  setSelectedItems( value: prevItems : {[p: string]: number} => ({
    ...prevItems,
    [item]: (prevItems[item] || 0) + 1,
  }));
};

1 usage  mjmp0027
const decrementItem = (item: string) : void => {
  setSelectedItems( value: prevItems : {[p: string]: number} => ({
    ...prevItems,
    [item]: prevItems[item] > 1 ? prevItems[item] - 1 : 0,
  }));
};
```

Ilustración 3-82 Componente para la selección de productos

4. Gestión de estado de pedidos

```java
// Endpoint que usará solo el cocinero para actualizar el estado de un pedido
mjmp0027
@PutMapping("/{id}/estado")
public ResponseEntity<?> actualizarEstado(
        @PathVariable String id,
        @RequestBody State estado
) {
    Optional<Pedido> pedidoOpt = pedidoService.obtenerPedido(id);
    if (pedidoOpt.isEmpty()) {
        return ResponseEntity.status(404).body("El pedido con el id: " + id + " no se ha encontrado");
    }

    Pedido pedido = pedidoOpt.get();
    pedido.setEstado(estado);

    pedido = pedidoService.crearPedido(pedido);

    if (estado.equals(State.LISTO)) {
        Mesa mesa = mesaService.findByNumero(pedido.getMesa());
        if (mesa.getUserId() == null) {
            return ResponseEntity.status(400).body("La mesa no tiene camarero asignado");
        }
        notificationService.save(Notificacion.builder()
                .mensaje("Pedido listo para la mesa " + pedido.getMesa())
                .titulo("Pedido Listo")
                .userId(mesa.getUserId())
                .timestamp(LocalDateTime.now())
                .leida(Estado.NOLEIDA)
                .build());
        User user = userService.findById(mesa.getUserId());
        notificationService.sendOrderReadyNotification(pedido.getMesa(), user.getUsername());
    }
    return ResponseEntity.status(200).body(PedidoDtoOut.fromDocument(pedido));
}
```

Ilustración 3-83 Endpoint actualizar estado de pedido

En la siguiente imagen vemos que si el role del usuario logueado es el cocinero se podrá actualizar el estado del pedido como se explica más arriba

```
} else if (role === 'COCINERO') {
  const nextState : "LISTO" | "EN_PREPARACION"  =
    item.estado === 'PENDIENTE' ? 'EN_PREPARACION' : 'LISTO';
  Alert.alert(
    title: 'Actualizar Pedido',
    message: `¿Quieres pasar este pedido a '${nextState.replace('_', ' ')}'?`,
    buttons: [
      {
        text: 'No',
        style: 'cancel',
      },
      {
        text: 'Sí',
        onPress: async () : Promise<void>  => {
          const success : boolean  = await updatePedidoEstado(item.id, nextState);
          if (success) {
            fetchPedidos();
            Alert.alert( title: 'Éxito',  message: 'El pedido ha sido actualizado.');
          } else {
            Alert.alert( title: 'Error',  message: 'No se pudo actualizar el pedido.');
          }
        },
      },
    ],
  );
```

Ilustración 3-84 Método actualizar estado de pedido

```
3 usages  mjmp0027
const updatePedidoEstado = async (
  pedidoId: string,
  nextState: string,
): Promise<boolean> => {
  try {
    const token : string | null  = await AsyncStorage.getItem( key: 'token');
    if (token) {
      const response : Response  = await fetch(
        input: `http://10.0.2.2:8080/api/pedidos/${pedidoId}/estado`,
        init: {
          method: 'PUT',
          headers: {
            'Content-Type': 'application/json',
            Authorization: `Bearer ${token}`,
          },
          body: JSON.stringify(nextState),
        },
      );

      return response.ok;
    }
  } catch (error) {
    console.error('Error al actualizar el estado del pedido:', error);
  }

  return false;
};

2 usages  mjmp0027
export default updatePedidoEstado;
```

Ilustración 3-85 Hook actualizar estado de pedido

5. Ver cuenta

Primero se obtienen los pedidos asociados a la mesa

```
mjmp0027
@GetMapping("/{mesa}/{estado}")
public ResponseEntity<?> obtenerPedidos(
        @PathVariable String mesa,
        @PathVariable State estado
) {
    List<Pedido> pedidos = pedidoService.obtenerPedidosPorMesaAndEstado(mesa, estado);
    return ResponseEntity.ok(PedidoDtoOut.fromDocuments(pedidos));
}
```

Ilustración 3-86 Endpoint obtener pedidos

```
1 usage  mjmp0027 *
const calculateTotalAndItems = async () : Promise<void>  => {
  const token : string | null  = await AsyncStorage.getItem( key: 'token');
  if (token) {
    const response : Response  = await fetch(
      input: `http://10.0.2.2:8080/api/pedidos/${mesa}/LISTO`,
      init: {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      },
    );
    if (response.ok) {
      const data = await response.json();
      let total : number  = 0;
      const cuentaItems: {item: string; count: number; price: number}[] = [];

      data.forEach((pedido: any) : void  => {
        pedido.items.forEach((item: string) : void  => {
          const product : Product | undefined  = products.find(p : Product  => p.name === item);
          if (product) {
            const itemIndex : number  = cuentaItems.findIndex(
              cuentaItem : {item: string, count: number, ...  => cuentaItem.item === item,
            );
            if (itemIndex > -1) {
              cuentaItems[itemIndex].count += 1;
            } else {
              cuentaItems.push({item: item, count: 1, price: product.price});
            }
            total += product.price;
          }
        });
      });
      setAccountDetails( value: {items: cuentaItems, total});
      setModalVisible( value: true);
    } else {
      Alert.alert( title: 'Error',  message: 'Hubo un problema al obtener la cuenta.');
    }
  }
};
```

Ilustración 3-87 Método para calcular la cuenta

```java
mjmp0027
@DeleteMapping("/delete/{mesa}")
public ResponseEntity<?> borrarPedidos(
        @PathVariable String mesa
) {
    pedidoService.borrarPedidosPorMesa(mesa);
    return ResponseEntity.status(200).body("Pedidos de la mesa " + mesa + " borrados");
}
```

Ilustración 3-88 Endpoint eliminar pedidos

```javascript
1 usage  mjmp0027
const handleDeleteOrders = async () : Promise<void>  => {
  const token : string | null  = await AsyncStorage.getItem( key: 'token');
  if (token) {
    await fetch( input: `http://10.0.2.2:8080/api/pedidos/delete/${mesa}`,  init: {
      method: 'DELETE',
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    setModalVisible( value: false);
    Alert.alert( title: 'Pedidos eliminados',  message: 'Los pedidos han sido eliminados.');
  }
  if (accountDetails) {
    const cuentaDto : {items: string[], total: numbe...  = {
      items: accountDetails.items.map(item : {item: string, count: number, ...  => `${item.count}x ${item.item}`),
      total: accountDetails.total,
    };

    const response : Response  = await fetch( input: 'http://10.0.2.2:8080/api/cuentas',  init: {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(cuentaDto),
    });

    if (response.ok) {
      Alert.alert( title: 'Cuenta guardada',  message: 'La cuenta ha sido guardada con éxito.');
    } else {
      Alert.alert( title: 'Error',  message: 'Hubo un problema al guardar la cuenta.');
    }
  }
};
```

Ilustración 3-89 Petición al backend

```java
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Document(collection = "cuentas")
public class Cuenta {

    @Id
    private String id;
    private List<String> items;
    @CreatedDate
    private LocalDateTime timestamp;
    private double total;
}
```

Ilustración 3-90 Clase cuenta

```java
mjmp0027
@RestController
@RequestMapping("/api/cuentas")
public class CuentaRestController {

    @Autowired
    private CuentaService cuentaService;

    mjmp0027
    @PostMapping
    public ResponseEntity<?> guardarCuenta(
            @RequestBody CuentaDto dto
    ) {
        // Se comprueba que el pedido tiene al menos un producto
        if (dto.getItems().isEmpty()) {
            return ResponseEntity.status(400).body("Debe haber al menos un producto en la cuenta");
        }

        Cuenta cuenta = Cuenta.builder()
                .total(dto.getTotal())
                .items(dto.getItems())
                .timestamp(LocalDateTime.now())
                .build();

        cuentaService.crearCuenta(cuenta);

        return ResponseEntity.ok( body: "Cuenta " + cuenta + " guardada correctamente");
    }
}
```

Ilustración 3-91 Endpoint guardar cuenta

6. Base de datos

Para actualizar la base de datos con las nuevas clases se ha realizado de la siguiente manera.

1. **Abrir MongoDB Compass**

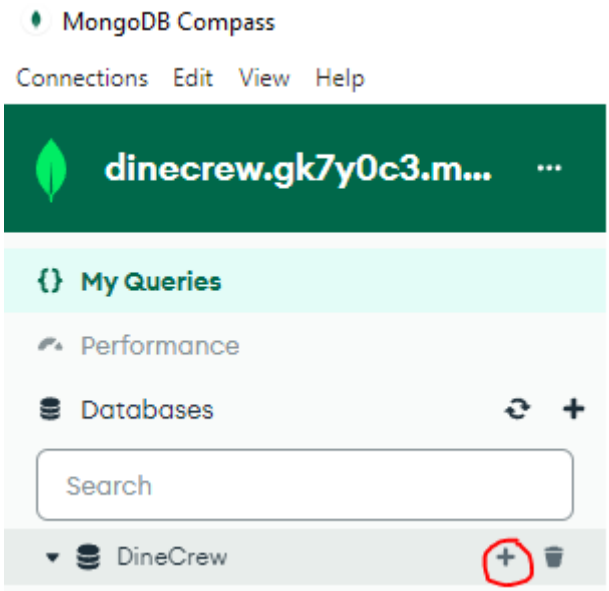

Ilustración 3-92 Creación de una colección

2. **Crear las colecciones**

Hay que asegurar que se le pone el mismo nombre que se le puso en la clase con '@Document'

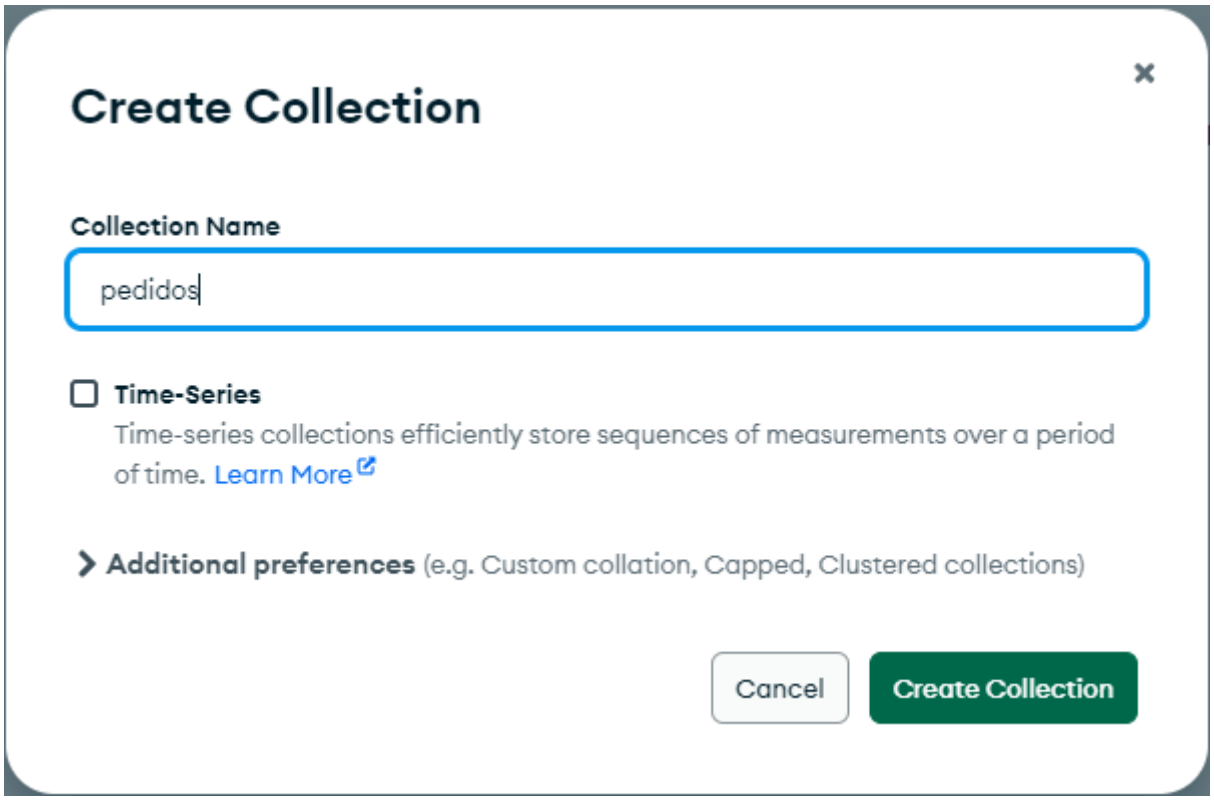

Ilustración 3-93 Creación de la colección pedidos

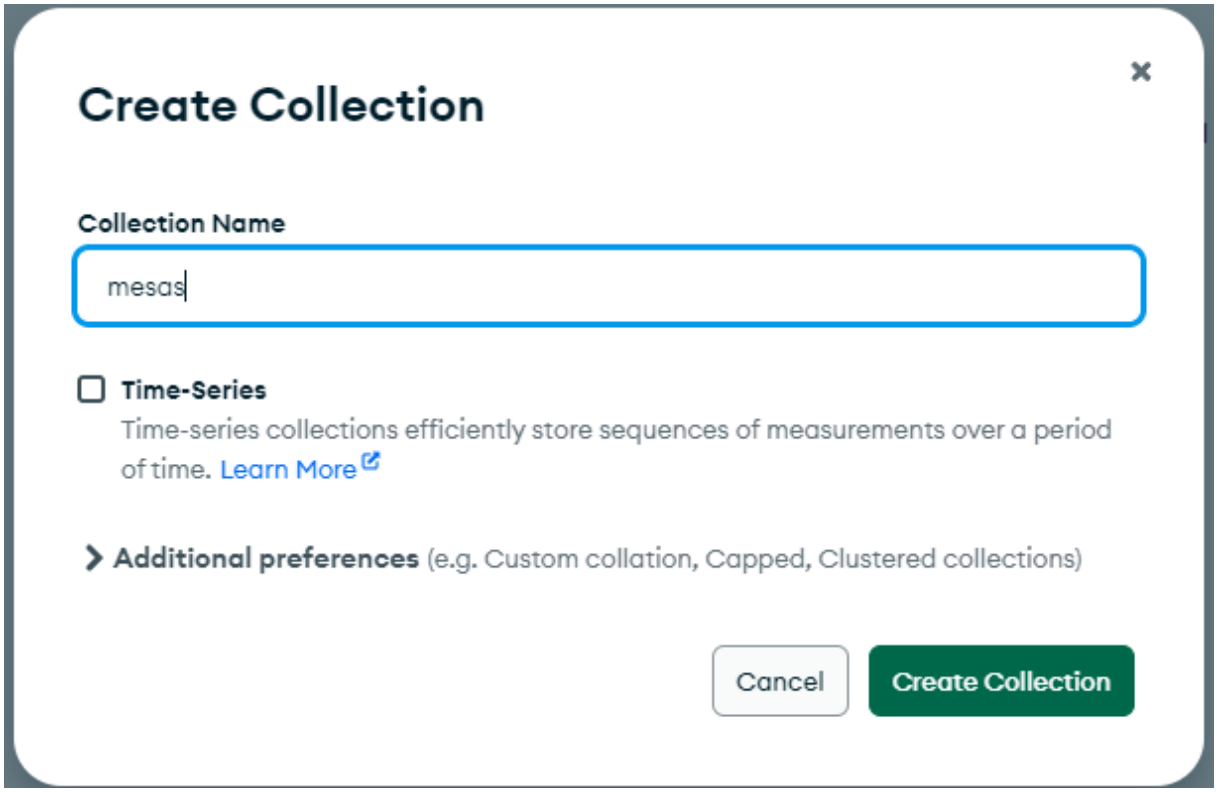

Ilustración 3-94 Creación de la colección mesas

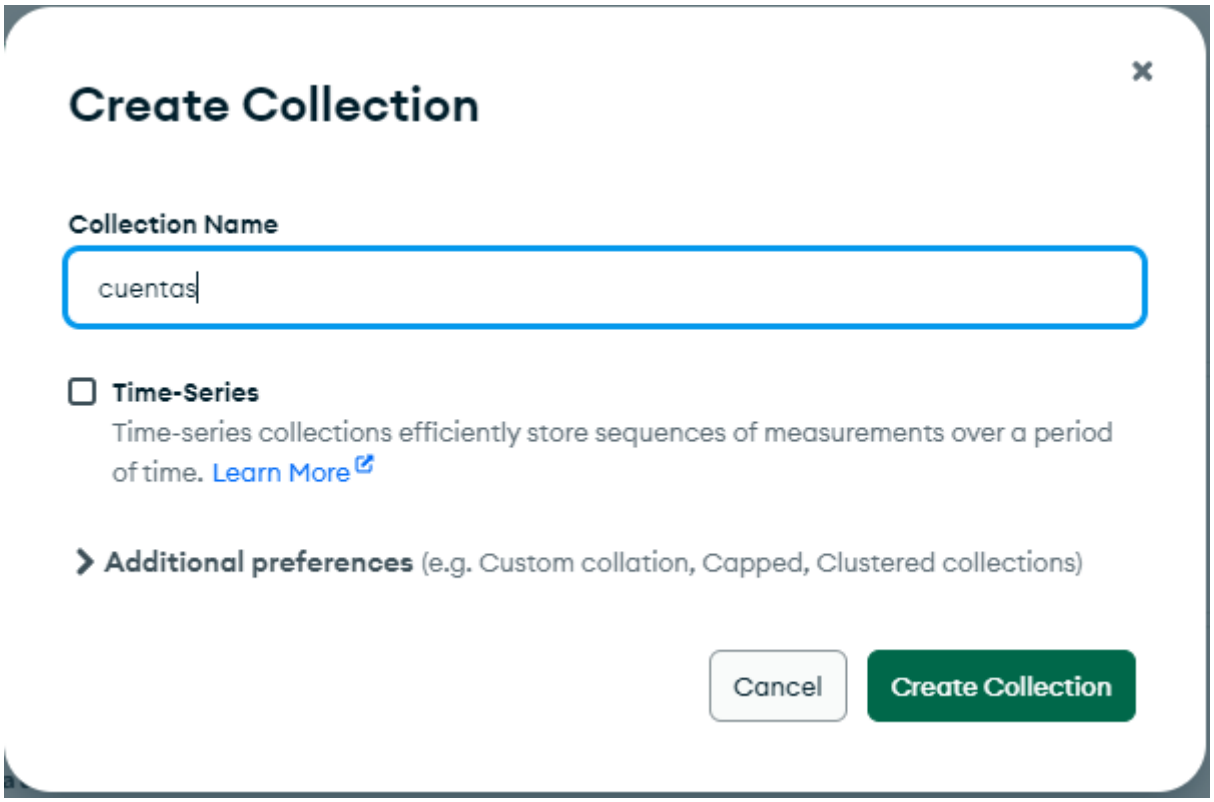

Ilustración 3-95 Creación de la colección cuentas

Tras realizar estos cambios ya podemos insertar mesas y pedidos en nuestra base de datos ya sea manualmente o a través de los repositorios

3. **Actualizar la clase principal del proyecto**

Ahora nuestra clase principal debería verse así:

```
mjmp0027 *
@SpringBootApplication
@EnableMongoRepositories(basePackages = {"com.dinecrew.dinecrewbackend.pedidos",
        "com.dinecrew.dinecrewbackend.usuarios",
        "com.dinecrew.dinecrewbackend.mesas",
        "com.dinecrew.dinecrewbackend.cuentas"})
public class DineCrewBackendApplication {

    mjmp0027
    public static void main(String[] args) { SpringApplication.run(DineCrewBackendApplication.class, args); }

}
```

Ilustración 3-96 Clase Main con los nuevos repositorios

3.3.4 Pruebas de Verificación y Validación

En esta iteración, se realizaron pruebas para asegurar que las nuevas funcionalidades estuvieran funcionando correctamente:

- **Pruebas unitarias:**
    - Se realizan pruebas unitarias en los componentes relacionados con la selección de las mesas, el registro de pedidos, la modificación de pedidos y la actualización del estado de los pedidos para asegurar que cada función individual funcionara según lo esperado.
- **Pruebas de integración:**
    - Se verificó que la integración entre la visualización de mesas, el registro de pedidos, la gestión de estados por parte del cocinero y la base de datos funcionara correctamente y sin errores.
- **Pruebas de consistencia de datos:**
    - Se realizaron pruebas para asegurar que los pedidos registrados y modificados, así como los cambios de estado, se almacenaran correctamente en la base de datos sin pérdida de información.
- **Imágenes de pruebas:**

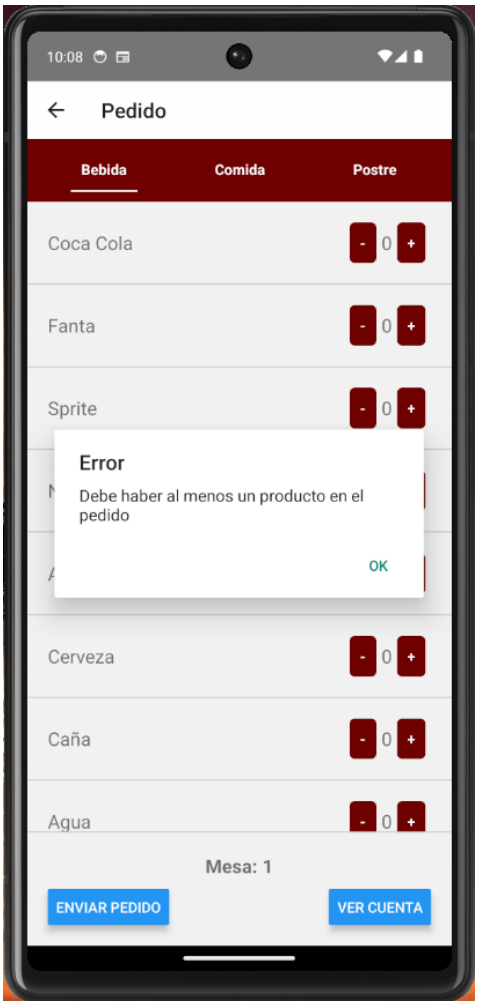

Ilustración 3-97 Prueba registro de pedido

Si al actualizar un pedido, quitamos todos los productos tampoco va a dejar

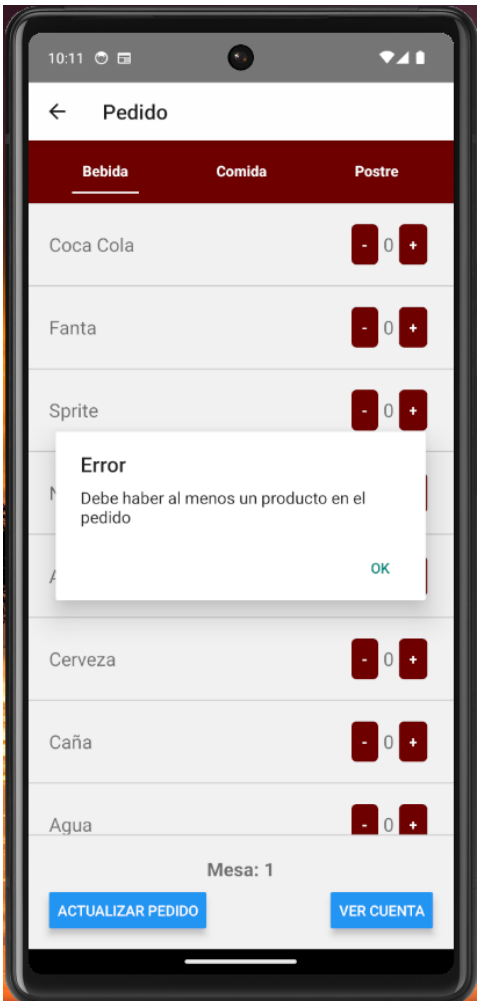

Ilustración 3-98 Prueba actualizar pedido

3.3.5 Pruebas de Usabilidad del Sistema

Finalmente, se realizaron pruebas de usabilidad para evaluar la experiencia del usuario en las funcionalidades implementadas:

- **Pruebas de selección de mesas:**
  - Se evaluó la facilidad y eficiencia con la que los camareros podían seleccionar y gestionar las mesas que les fueron asignadas.
- **Pruebas de registro de pedidos:**
  - Se evaluó la experiencia del usuario al registrar nuevos pedidos, incluyendo la selección de productos, asegurando que el proceso fuera intuitivo y rápido.
- **Pruebas de modificación de pedidos:**

- Se evaluó la facilidad con la que los camareros podían modificar pedidos en tiempo real, asegurando que los cambios pudieran realizarse sin complicaciones y que se reflejaran inmediatamente en el sistema.

- **Pruebas de actualización de estado de pedidos:**
  - Se evaluó la claridad y usabilidad de la interfaz utilizada por los cocineros para actualizar el estado de los pedidos, asegurando que el proceso fuera directo y sin ambigüedades.

- **Imágenes de usabilidad:**

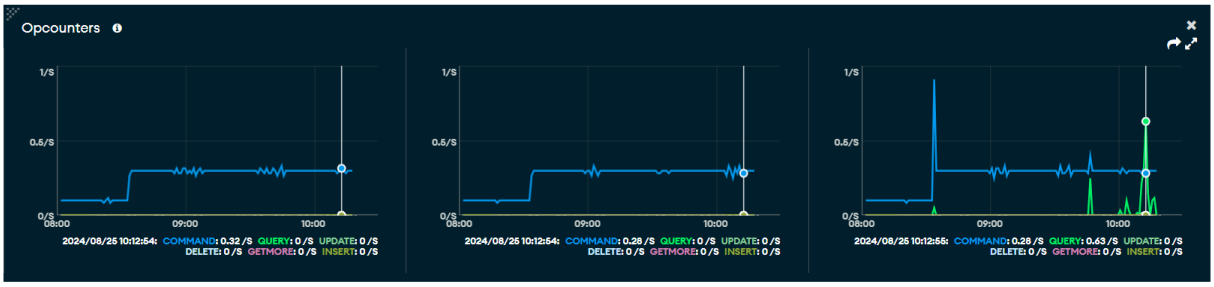

Ilustración 3-99 Opcounters

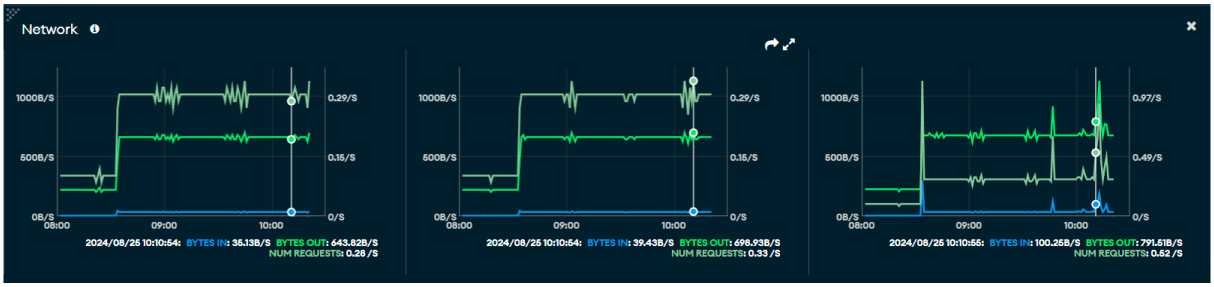

Ilustración 3-100 Network

3.4 Cuarta Iteración

El objetivo principal de esta cuarta iteración fue implementar un sistema de notificaciones en tiempo real que avise a los camareros cuando un pedido esté listo para ser servido. Para lograrlo, se utilizó Firebase Cloud Messaging [18], garantizando que el camarero responsable reciba una notificación inmediata cuando un pedido cambia su estado a "listo".

3.4.1 Historias de usuario

En esta iteración, se desarrolló la siguiente historia de usuario relacionada con las notificaciones, cubriendo el caso de uso específico:

- **Notificación de pedido listo:**
  - **Historia de usuario:** Como camarero, quiero recibir una notificación en tiempo real cuando un pedido esté listo para ser servido, para poder atender a los clientes de manera oportuna.
  - **Criterios de aceptación:**
    - El camarero recibe una notificación en su dispositivo cuando un pedido cambia su estado a "listo".
    - La notificación incluye detalles como el número de mesa y el contenido del pedido.

3.4.2 Storyboards

Se han creado los siguientes storyboards para ilustrar el flujo de trabajo de la funcionalidad de notificación implementada en esta iteración:

- **Storyboard 1: Notificación de pedido listo**
  - **Descripción:** Este storyboard muestra el flujo desde que el cocinero marca un pedido como "listo" hasta que el camarero recibe la notificación en su dispositivo.
  - **Imágenes:**

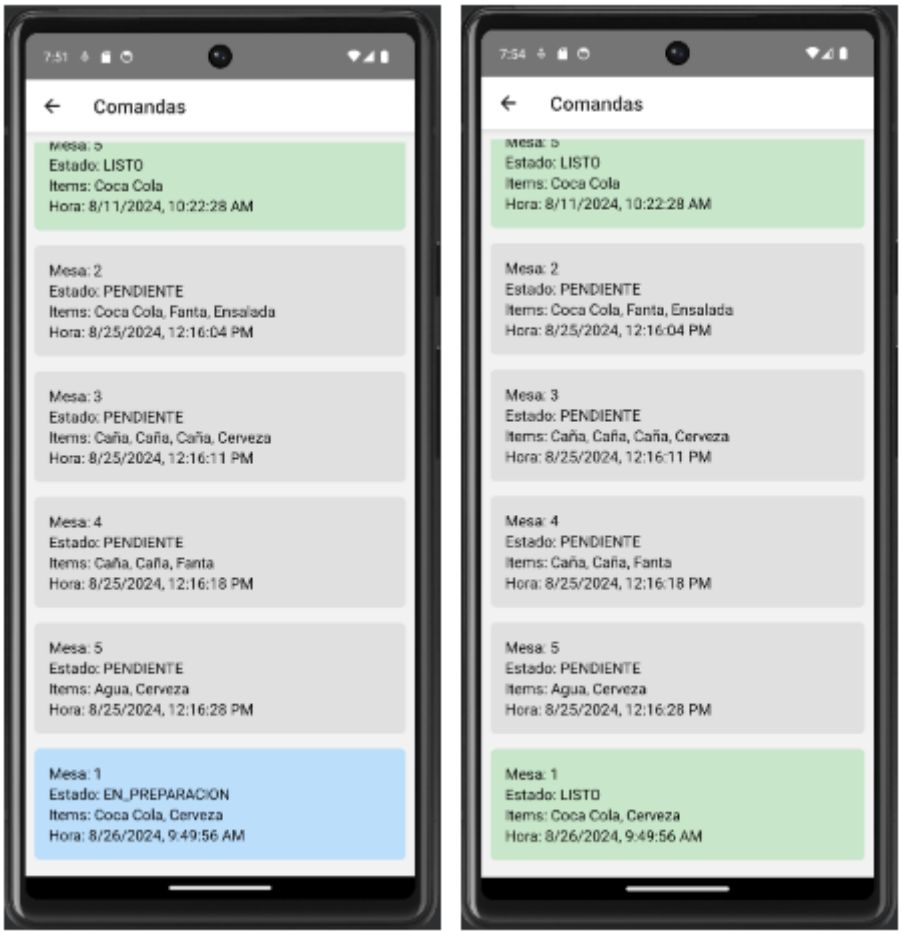

Ilustración 3-101 Lista de pedidos

Como ya se ha visto en las iteraciones anteriores, el camarero debe realizar el pedido previamente, luego se puede visualizar en la ilustración 3.93 como el cocinero pasa el pedido de la mesa 1 a 'LISTO'. Seguidamente el camarero recibe la notificación en su dispositivo y podrá consultarla en el menú clickando en la campanita

Ilustración 3-102 Notificación recibida

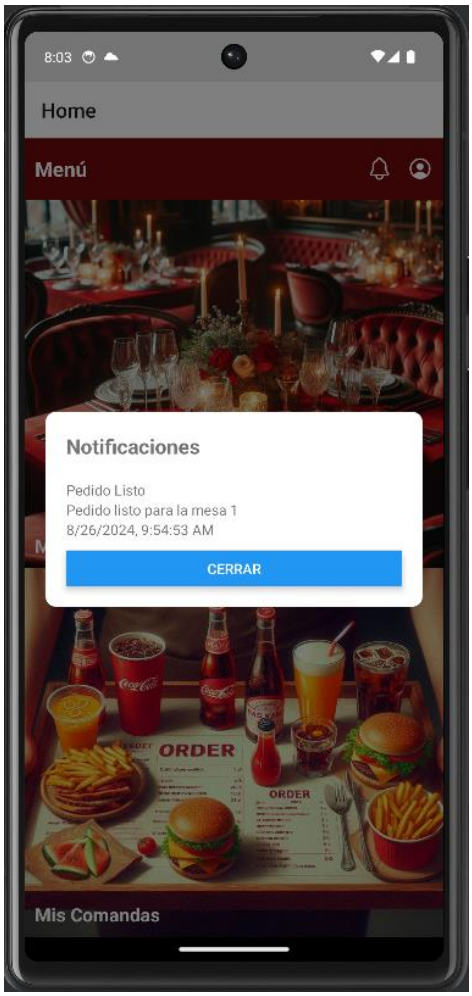

Ilustración 3-103 Notificaciones

3.4.3 Detalles de implementación

En esta iteración, se implementaron las siguientes funcionalidades clave relacionadas con el sistema de notificaciones:

- **Integración con firebase:**
  - Configuración de Firebase Cloud Messaging (FCM) [19] para enviar notificaciones push a los dispositivos móviles de los camareros.
  - Implementación de la lógica en el backend para generar y enviar notificaciones cuando un pedido cambia su estado a "listo".
- **Notificación en tiempo real:**

- Desarrollo de la funcionalidad para enviar notificaciones en tiempo real a los camareros cuando un pedido es marcado como "listo" por el cocinero.
- Configuración de los canales de notificación en la aplicación, permitiendo que las notificaciones contengan información relevante, como el número de mesa y los detalles del pedido.

- **Imágenes de implementación:**

Primero de todo hay que configurar firebase para integrarlo con nuestra aplicación, para ellos vamos a la página principal de firebase y se crea un nuevo proyecto.

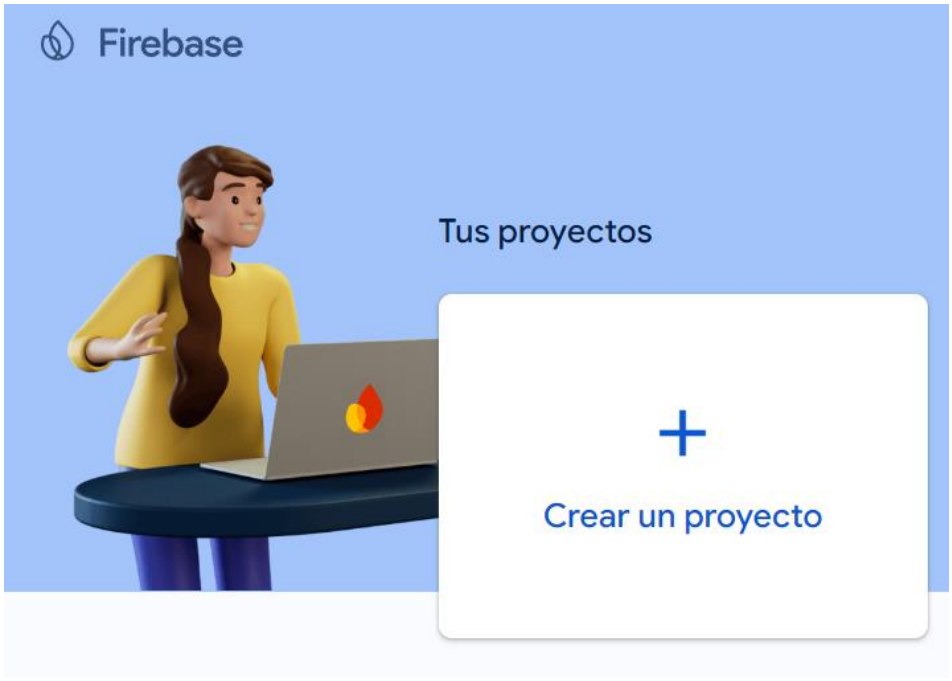

Ilustración 3-104 Inicio firebase

Ilustración 3-105 Nombre proyecto firebase

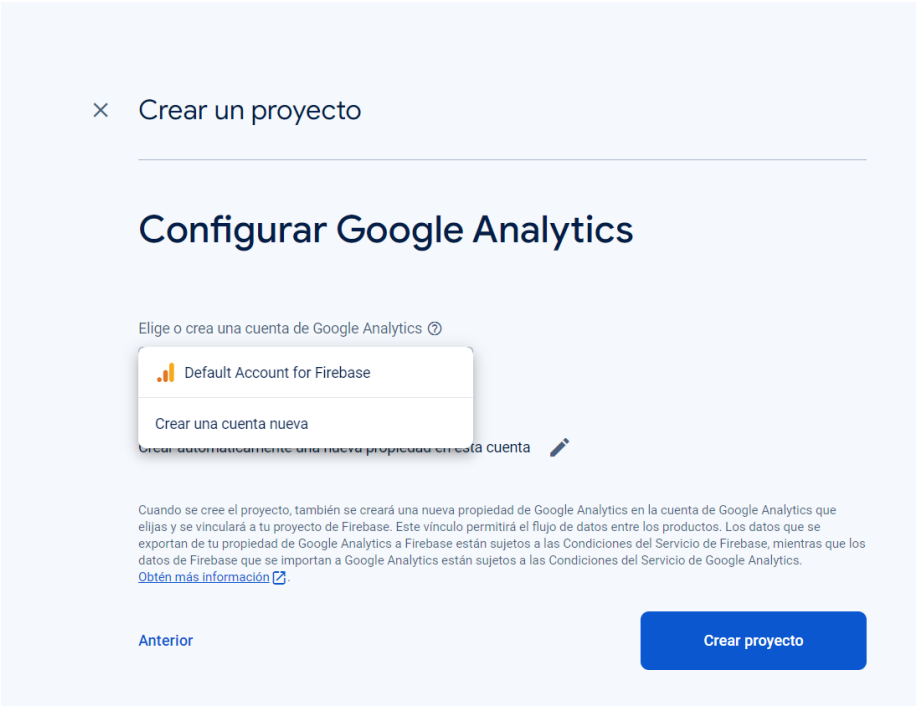

Ilustración 3-106 Selección cuenta por defecto

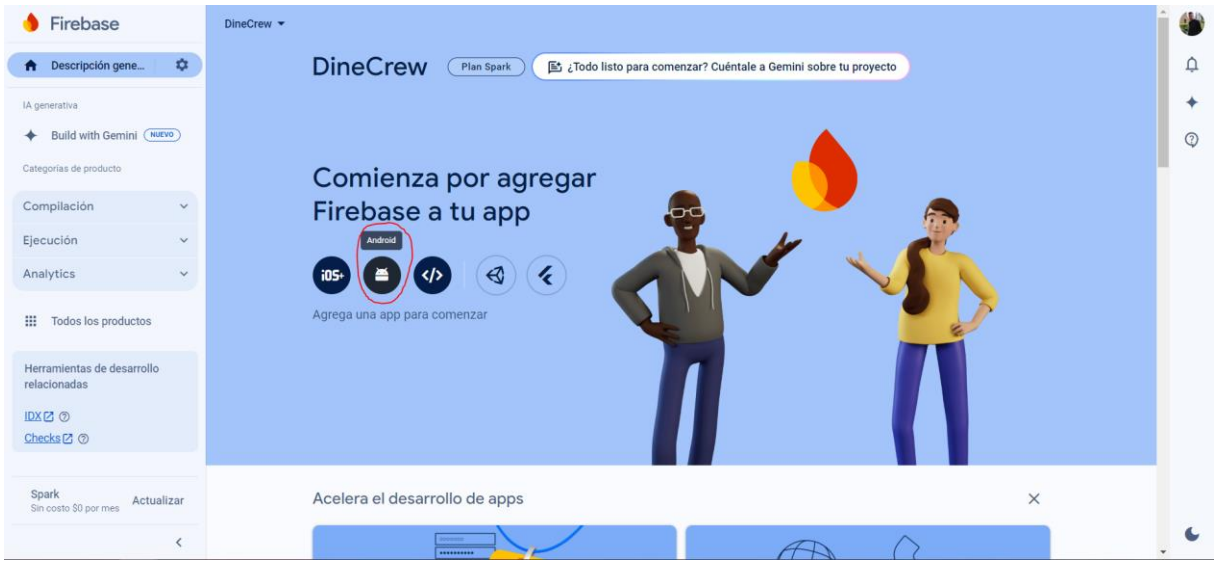

Ilustración 3-107 Selección de App

Ilustración 3-108 Registrar App

El nombre del paquete de Android lo encontraremos en la clase MainActivity.java ubicada en la carpeta Android de nuestro proyecto react native

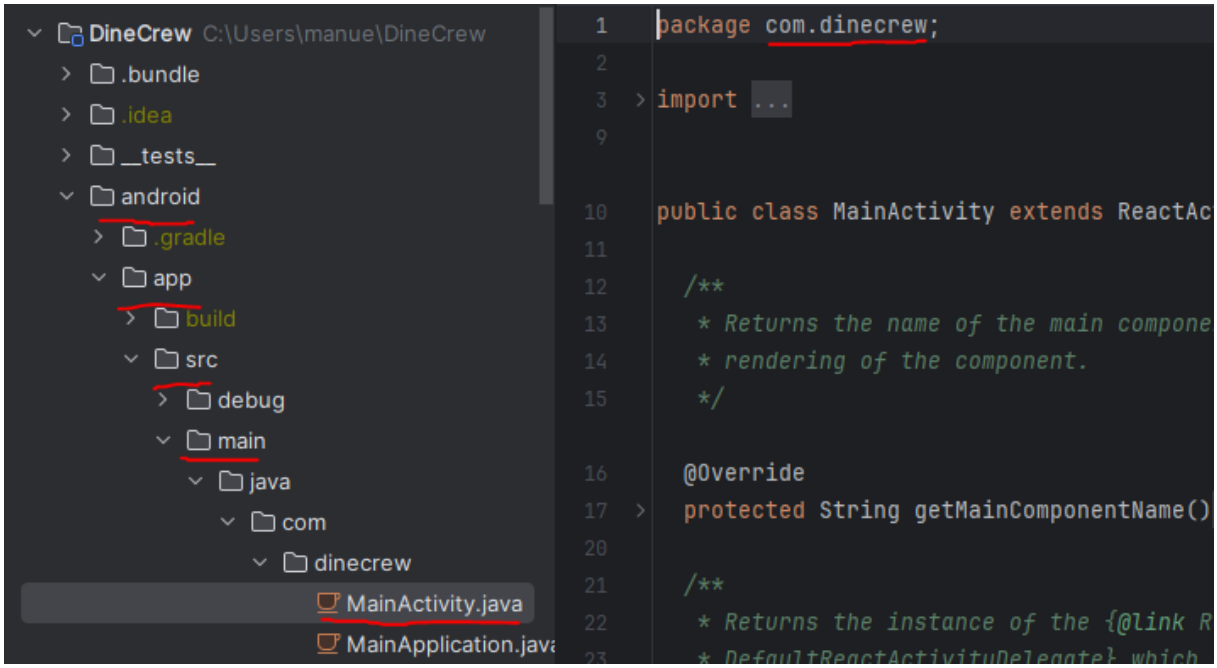

Ilustración 3-109 Nombre App

Para la obtención del certificado de firma SHA-1 es necesario seguir los siguientes pasos. Ir a la carpeta Android ejecutando ‘cd android’ en la raíz del proyecto, una vez allí se ejecutará el comando ‘./gradlew signingReport’.

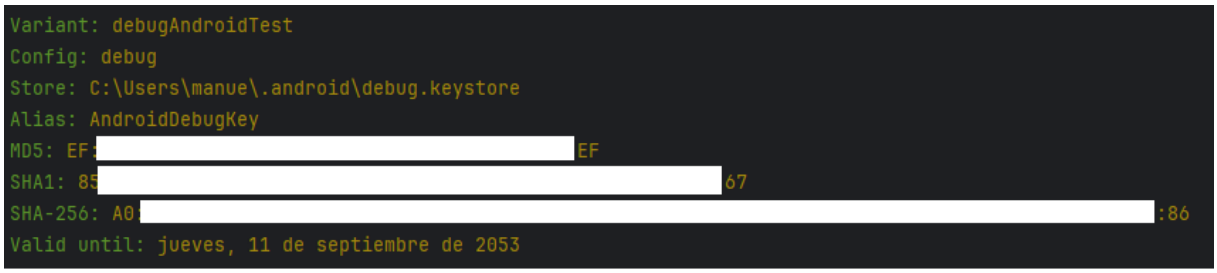

Ilustración 3-110 Clave SHA1

Una vez se registra la app será necesario descargar el archivo ‘google-services.json’ que ofrece firebase y pegarlo en la carpeta app situada en la carpeta Android quedando algo similar a esto:

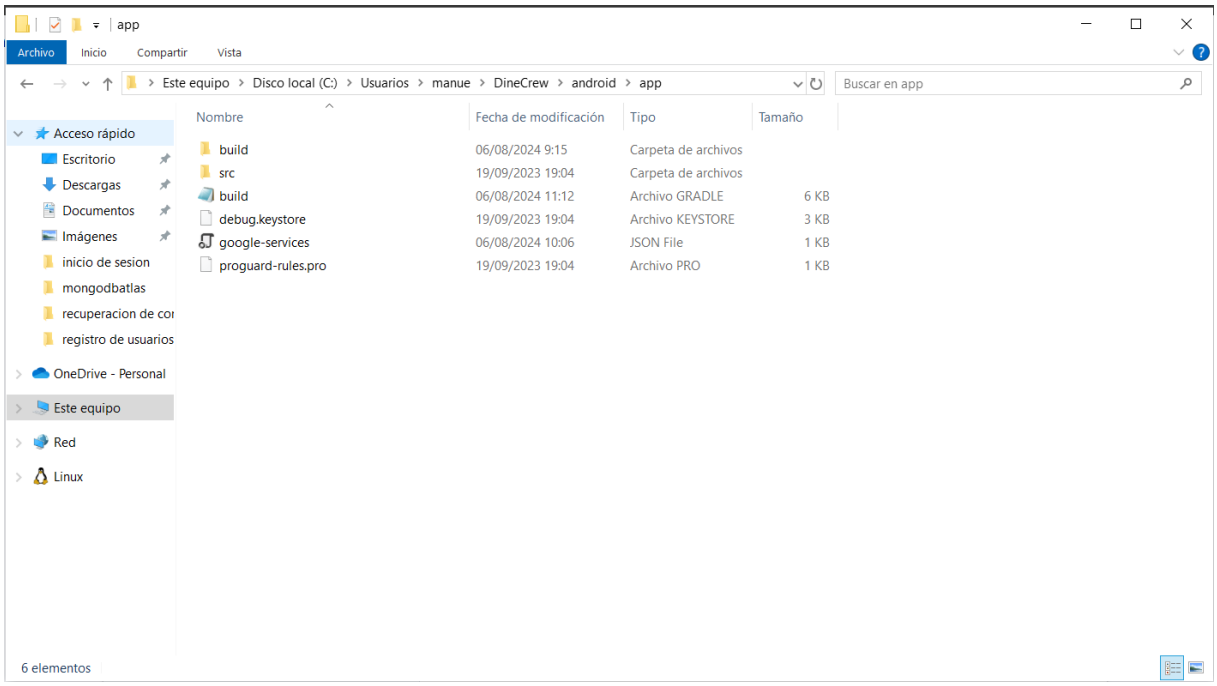

Ilustración 3-111 Vista rápida carpeta app

Terminado esto ya se podrá iniciar la aplicación.

Ahora vamos con la parte de enviar notificaciones mediante el backend, para ello será necesario agregar la dependencia '**firebase-admin**' al pom.xml.

```
<!-- Firebase -->
<dependency>
    <groupId>com.google.firebase</groupId>
    <artifactId>firebase-admin</artifactId>
    <version>8.2.0</version>
</dependency>
```

Ilustración 3-112 Dependencia firebase-admin

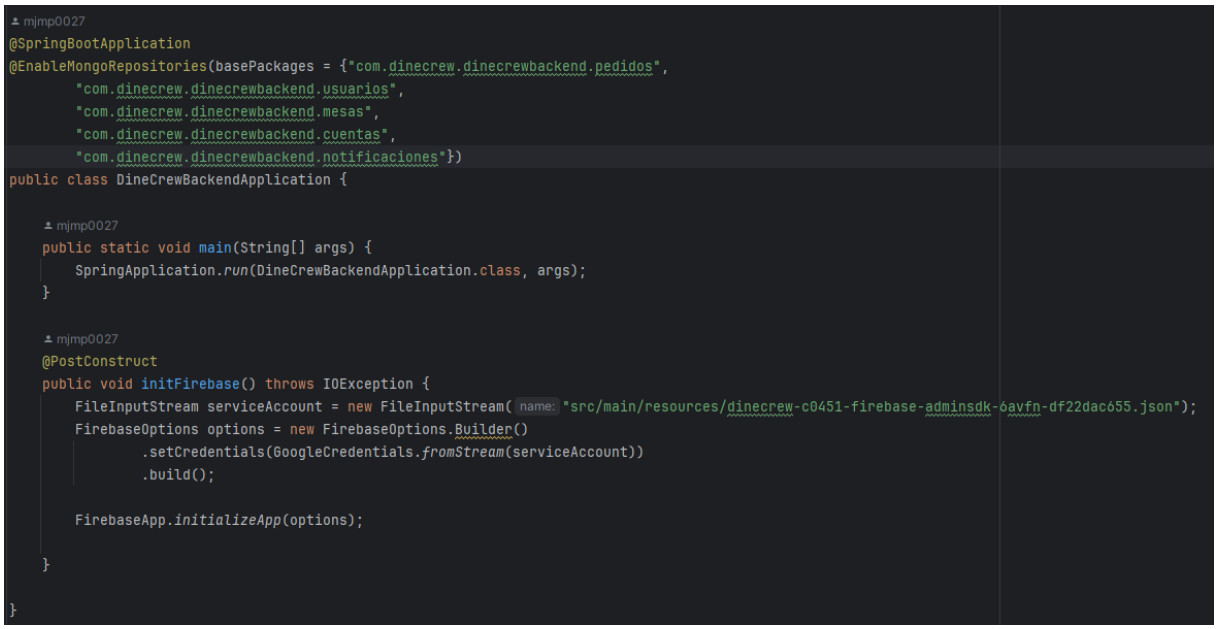

Ilustración 3-113 Clase Main

El archivo que se le indica a FileInputStream se obtendrá de la siguiente manera.

- Ir a firebase console
- Seleccionar el proyecto creado
- En el menú de la izquierda, seleccionar ‘Project Settings’ o ‘Configuración del proyecto’
- Seleccionar ‘Service Account’ o ‘Cuentas de servicio’
- Indicar el fragmento ‘Java’ y generar nueva clave privada

Ese será el archivo que se usará para iniciar firebase en el backend.

```
11 usages  mjmp0027
@Document(collection = "notificaciones")
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Notificacion {

    @Id
    private String id;
    private String titulo;
    private String mensaje;
    private LocalDateTime timestamp;
    private String userId;
    private Estado leida;
}
```

Ilustración 3-114 Clase Notificacion

Como ya se vio en la iteración 3 existe un endpoint que actualiza el estado del pedido, este a su vez llama al servicio de notificación enviando la notificación al camarero correspondiente.

```
1 usage  mjmp0027
public void sendOrderReadyNotification(String tableName, String username) {
    Notification notification = Notification.builder()
            .setTitle("Pedido Listo")
            .setBody("El pedido de la mesa " + tableName + " está listo.")
            .build();

    Message message = Message.builder()
            .setNotification(notification)
            .setTopic(username)
            .build();

    try {
        FirebaseMessaging.getInstance().send(message);
        System.out.println("Notification sent successfully");
    } catch (Exception e) {
        System.err.println("Error sending notification: " + e.getMessage());
    }
}
```

Ilustración 3-115 Método enviar notificación

```
@RestController
@RequestMapping("/api/notificaciones")
public class NotificacionRestController {

    @Autowired
    private NotificationService notificationService;

    mjmp0027
    @GetMapping("/{usedId}")
    public ResponseEntity<?> obtenerNotificaciones(
            @PathVariable String usedId
    ) {

        List<Notificacion> notificaciones = notificationService.getNotificationsByUserId(usedId);
        return ResponseEntity.ok(NotificacionDtoOut.fromDocuments(notificaciones));
    }

    mjmp0027
    @PostMapping("/{usedId}")
    public ResponseEntity<?> marcarComoLeidas(
            @PathVariable String usedId
    ) {
        notificationService.marcarComoLeidas(usedId);
        return ResponseEntity.ok().build();
    }
}
```

Ilustración 3-116 Endpoint notificaciones

Para que sea posible que el camarero reciba la notificación cuando sea necesario se ha implementado una especie de 'Publicador-suscriptor', donde el publicador es el backend que se muestra en la ilustración 3.107 y el suscriptor será el usuario al iniciar sesión. Para ello se ha cogido el nombre de usuario del usuario ya que es único por lo que cada usuario estará suscrito a un tópico diferente y luego se usará ese tópico para publicar la notificación correspondiente.

```
1 usage   mjmp0027
const subscribeToTopic = async () : Promise<void>  => {
  const token : string | null  = await AsyncStorage.getItem( key: 'token');
  if (token) {
    const decodedToken = decodeJwtToken(token);
    const username = decodedToken.sub;
    try {
      await messaging().subscribeToTopic(username);
      console.log(`Subscribed to topic "${username}"`);
    } catch (error) {
      console.error('Error subscribing to topic:', error);
    }
  }
};
```

Ilustración 3-117 Método suscribir usuario a tópico

```
useEffect( effect: () => {
  subscribeToTopic();

  const unsubscribe = messaging().onMessage( listener: async remoteMessage : FirebaseMessagingTypes.RemoteM...  => {
    console.log('A new FCM message arrived!', JSON.stringify(remoteMessage));
    Alert.alert(
      remoteMessage.notification?.title as string,
      remoteMessage.notification?.body,
    );
    console.log('Pedido Listo', remoteMessage.notification?.body);
    setNotificationCount( value: prevCount : number  => prevCount + 1);
  });

  return unsubscribe;
}, deps: []);
```

Ilustración 3-118 Método mostrar notificación en pantalla

Para la lista de notificaciones que puede ver el camarero, se ha escogido mostrar las notificaciones recibidas en las últimas 3 horas para mejora visual. Para

ello se ha implementado una Query [19] en MongoDB para obtener estas notificaciones.

```
1 usage  mjmp0027
@Repository
public interface NotificacionRepository extends MongoRepository<Notificacion, String> {
    1 usage  mjmp0027
    @Query("{userId: ?0, 'timestamp': { $gte: ?1 }, 'leida': ?2 }")
    List<Notificacion> findRecentNotificationsNoReads(String userId, LocalDateTime timestamp, Estado leida);

    1 usage  mjmp0027
    @Query("{userId: ?0, 'timestamp': { $gte: ?1 }}")
    List<Notificacion> findRecentNotificationsByUserId(String userId, LocalDateTime timestamp);
}
```

Ilustración 3-119 Query notificaciones

La primera se ha implementado para marcar como leídas aquellas que no estén leídas y la segunda se ha implementado para obtener las notificaciones recientes del usuario logueado.

```
1 usage  mjmp0027
public void marcarComoLeidas(String userId) {
    LocalDateTime threeHoursAgo = LocalDateTime.now().minusHours(3);
    System.out.println(threeHoursAgo);
    System.out.println(userId);
    List<Notificacion> notificaciones = repository.findRecentNotificationsNoReads(userId, threeHoursAgo, Estado.NOLEIDA);
    notificaciones.forEach(notificacion -> {
        notificacion.setLeida(Estado.LEIDA);
        repository.save(notificacion);
    });
}
```

Ilustración 3-120 Método marcar como leídas

No olvidar actualizar la base de datos agregando la colección con nombre ‘notificaciones’ tal y como se indica en la clase notificación. Abrir MongoDB Atlas y agregar la nueva colección

3.4.4 Pruebas de verificación y validación

En esta iteración, se realizaron pruebas para asegurar que la funcionalidad de notificaciones estuviera funcionando correctamente:

- **Pruebas unitarias:** Se realizaron pruebas unitarias para verificar que la notificación se generara y enviara correctamente cuando un pedido cambiara su estado a "listo".

- **Pruebas de integración:** Estas pruebas aseguraron que la integración con Firebase funcionara correctamente, verificando que la notificación se entregara de manera confiable a los dispositivos móviles de los camareros.
- **Pruebas de rendimiento:** Se realizaron pruebas de rendimiento para asegurar que el sistema de notificaciones funcionara eficientemente bajo diferentes cargas, manteniendo la entrega en tiempo real sin retrasos significativos.

3.4.5 Pruebas de usabilidad del sistema

Finalmente, se realizaron pruebas de usabilidad para evaluar la experiencia del usuario con la notificación implementada durante esta iteración:

- **Pruebas de notificación para camareros:**
  - Se evaluó la facilidad con la que los camareros podían recibir y gestionar la notificación de pedido listo, asegurando que la información proporcionada fuera clara y útil, pero se descubrió que las notificaciones no mostraban suficiente información.

3.5 Quinta Iteración

Para esta iteración se ha optado por una mejora visual en algunas funcionalidades de la aplicación para facilitar la labor de los usuarios, entre las mejoras se encuentra, incluir detalles del pedido en las notificaciones para mejorar la experiencia del camarero, separar en pedidos listos y no listos en la vista de los pedidos para evitar acumulación, y algunas configuraciones y cambios más.

Esta iteración será diferente a las demás ya que se solo se mostrarán los cambios realizados y las pruebas pertinentes para el correcto funcionamiento de la aplicación.

Estará dividida por secciones las cuales serán los cambios y mejoras que se van a realizar, por lo tanto, la primera sección será la mejora de las notificaciones incluyendo los detalles del pedido. A continuación, se mostrarán los detalles de implementación necesarios para llevar a cabo la actualización.

```
11 usages  mjmp0027 *
@Document(collection = "notificaciones")
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Notificacion {

    @Id
    private String id;
    private String titulo;
    private String mensaje;
    private LocalDateTime timestamp;
    private String userId;
    private Estado leida;
    private List<String> items;
}
```

Ilustración 3-121 Clase notificación

```
mjmp0027 *
@PutMapping("/{id}/estado")
public ResponseEntity<?> actualizarEstado(
        @PathVariable String id,
        @RequestBody State estado
) {
    Optional<Pedido> pedidoOpt = pedidoService.obtenerPedido(id);
    if (pedidoOpt.isEmpty()) {
        return ResponseEntity.status(404).body("El pedido con el id: " + id + " no se ha encontrado");
    }

    Pedido pedido = pedidoOpt.get();
    pedido.setEstado(estado);

    pedido = pedidoService.crearPedido(pedido);

    if (estado.equals(State.LISTO)) {
        Mesa mesa = mesaService.findByNumero(pedido.getMesa());
        if (mesa.getUserId() == null) {
            return ResponseEntity.status(400).body("La mesa no tiene camarero asignado");
        }
        notificationService.save(Notificacion.builder()
                .mensaje("Pedido listo para la mesa " + pedido.getMesa())
                .titulo("Pedido Listo")
                .userId(mesa.getUserId())
                .timestamp(LocalDateTime.now())
                .leida(Estado.NOLEIDA)
                .items(pedido.getItems())
                .build());
        User user = userService.findById(mesa.getUserId());
        notificationService.sendOrderReadyNotification(pedido.getMesa(), user.getUsername(), pedido.getItems());
    }
    return ResponseEntity.status(200).body(PedidoDtoOut.fromDocument(pedido));
}
```

Ilustración 3-122 Endpoint actualizar estado del pedido

Se han agregado los ítems que tiene el pedido a la notificación para poder enviarlos al camarero, estos se han incluido en el cuerpo de la notificación tal y como se muestra en la siguiente ilustración.

```
1 usage  mjmp0027 *
public void sendOrderReadyNotification(String tableName, String username, List<String> items) {

    String itemsString = String.join( delimiter: ", ", items);
    Notification notification = Notification.builder()
            .setTitle("Pedido Listo")
            .setBody("El pedido de la mesa " + tableName + " está listo. Productos: " + itemsString)
            .build();

    Message message = Message.builder()
            .setNotification(notification)
            .setTopic(username)
            .build();

    try {
        FirebaseMessaging.getInstance().send(message);
        System.out.println("Notification sent successfully");
    } catch (Exception e) {
        System.err.println("Error sending notification: " + e.getMessage());
    }
}
```

Ilustración 3-123 Método enviar notificación

Ilustración 3-124 Notificación

Ilustración 3-125 Lista de notificaciones

La segunda sección se trata de un cambio en la vista de pedidos separando los pedidos por estado, para lograr que el camarero ubique mejor sus pedidos y poder informar a los clientes del proceso de su pedido en caso de que estos preguntasen. Para ello los cambios realizados han sido sobre todo visuales y cambio en los estilos usando para ello la herramienta para la obtención de los códigos de colores [7].

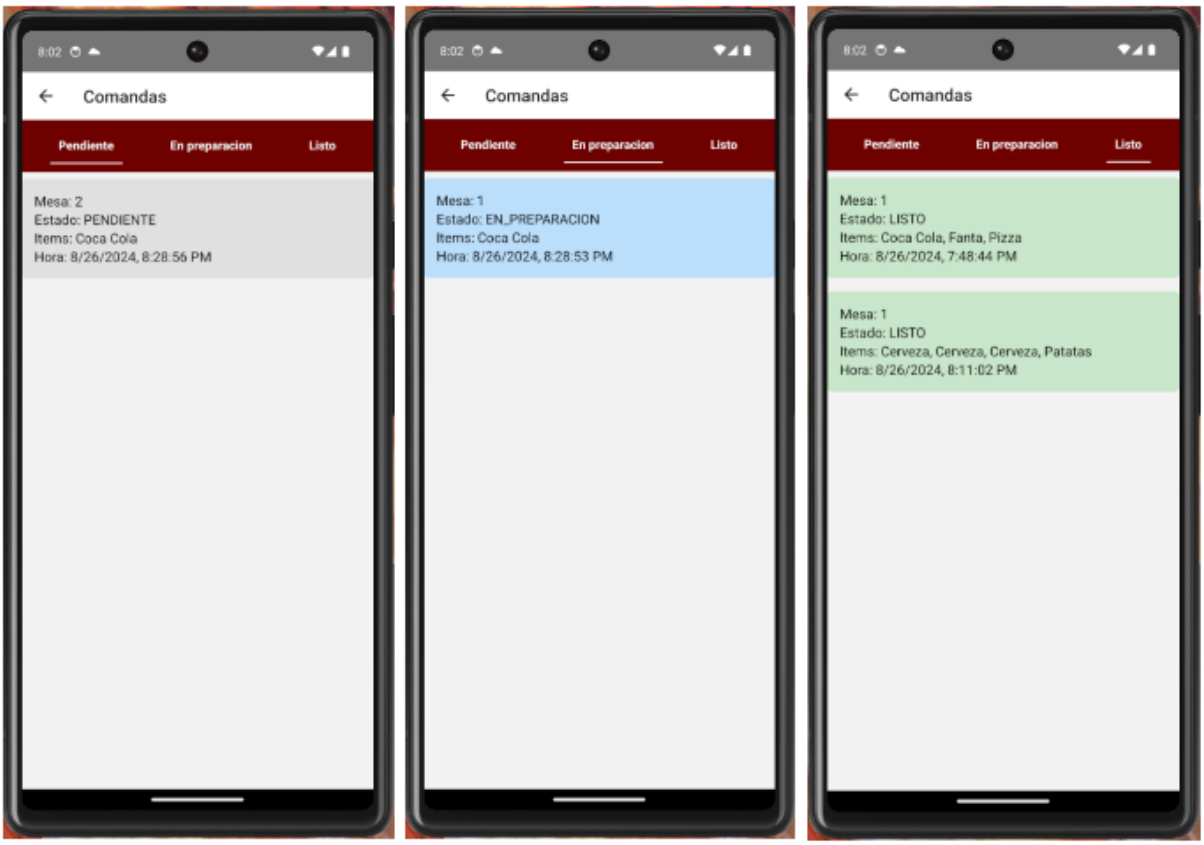

Ilustración 3-126 Lista de pedidos

Las pruebas realizadas se pueden comprobar en las ilustraciones ofrecidas anteriormente, ya que se muestra tal y como se quería. Añadir a estas, que si un pedido no tiene productos no se puede realizar por lo tanto nunca va a llegar una notificación de un pedido que no existe

3.6 Pruebas finales

Tras la conclusión del desarrollo del proyecto, se llevaron a cabo una serie de pruebas finales para asegurar la estabilidad y funcionalidad del sistema en su conjunto. Estas pruebas se realizaron para verificar que todos los módulos del sistema funcionan correctamente cuando se integran entre sí y para garantizar que la solución final cumple con las expectativas del usuario.

Las pruebas finales arrojaron resultados positivos, confirmando que el sistema es estable, funcional y se comporta según lo esperado bajo diversas condiciones de uso. A continuación, se proporciona un resumen de las pruebas realizadas:

3.6.1 Pruebas de verificación del sistema

Se revisaron las pruebas documentadas en cada iteración para confirmar que todos los requisitos del sistema fueron cumplidos.

Las pruebas de verificación incluyeron:

- Revisión de los resultados de pruebas unitarias.
- Confirmación de la cobertura de las pruebas de integración.

Los resultados de estas pruebas demostraron que cada componente individual y las interacciones entre ellos funcionan correctamente y de manera coherente con las especificaciones establecidas.

3.6.2 Pruebas de validación del sistema

Se realizaron pruebas de validación para asegurar que el sistema cumple con los requisitos del usuario final.

Esto incluyó:

- Validación de la funcionalidad según los casos de uso finales.
- Pruebas de aceptación del usuario.

Los resultados de estas pruebas fueron satisfactorios, ya que el sistema respondió adecuadamente a los escenarios previstos y los usuarios finales expresaron su conformidad con la funcionalidad ofrecida.

3.6.3 Validación de la usabilidad del sistema

Se llevaron a cabo pruebas de usabilidad para evaluar la interfaz de usuario y la experiencia general del usuario:

- Evaluaciones de usabilidad con usuarios representativos.
- Ajustes basados en los comentarios de las pruebas de usabilidad para mejorar la experiencia del usuario.

Los resultados de las pruebas de usabilidad indicaron que la interfaz es intuitiva y fácil de usar, lo que contribuye a una experiencia de usuario positiva. Los ajustes

realizados a partir del feedback mejoraron aún más la accesibilidad y la satisfacción general de los usuarios.

En conjunto, estas pruebas finales confirmaron que el sistema es funcional, cumple con los requisitos establecidos durante el proyecto y proporciona una experiencia de usuario satisfactoria. El éxito en estas pruebas indica que el sistema está listo para su despliegue en un entorno de producción.

4 CONCLUSIONES Y TRABAJOS FUTUROS

4.1 Conclusiones

Al finalizar este proyecto, puedo reflexionar sobre el proceso y los resultados obtenidos. Este trabajo ha sido un reto significativo, pero también una valiosa experiencia de aprendizaje. Desde la implementación del backend hasta la construcción de la aplicación móvil en React Native, he enfrentado y superado diversos desafíos técnicos y de diseño. El proyecto no solo me permitió aplicar conocimientos previos, sino que también me enseñó nuevas habilidades en el desarrollo móvil y la integración de sistemas, reforzando mi confianza en el manejo de tecnologías modernas.

A continuación, se presenta una lista de conclusiones clave:

1. **Cumplimiento de Requisitos:** La aplicación móvil desarrollada en React Native ha cumplido con los requisitos funcionales estalecidos. Se ha implementado una solución efectiva para la gestión de pedidos, que ofrece una experiencia fluida y moderna en dispositivos móviles.
2. **Eficiencia y Usabilidad:** La aplicación proporciona una experiencia de usuario rica en dispositivos móviles, con una interfaz intuitiva y adaptada a las interacciones táctiles. La versión web, aunque funcional, sirve principalmente como una vista adicional con capacidades limitadas en comparación con la aplicación móvil.
3. **Integración Backend-Frontend:** La integración entre el frontend móvil y el backend en Spring Boot ha sido exitosa, permitiendo una comunicación efectiva para la gestión de pedidos. Esta integración ha sido clave para garantizar que la aplicación funcione como un sistema cohesivo.
4. **Manejo de Errores y Seguridad:** Se han implementado mecanismos básicos de manejo de errores y seguridad en la aplicación móvil, garantizando la protección de los datos del usuario y una gestión eficaz de las solicitudes. Este enfoque ha sido crucial para asegurar la fiabilidad y la integridad de la aplicación en su entorno operativo.

4.2 Trabajos futuros

1. **Mejora de la Experiencia de Usuario:**
    a. **Propuesta:** Continuar optimizando la interfaz de usuario en React Native para mejorar la interacción y adaptar la experiencia a nuevas funcionalidades móviles.
    b. **Impacto Esperado:** Proporcionar una experiencia más fluida y moderna en dispositivos móviles.
2. **Optimización del Rendimiento:**
    a. **Propuesta:** Revisar y mejorar el rendimiento de la aplicación móvil, aplicando técnicas para optimizar tiempos de carga y eficiencia en el uso de recursos.
    b. **Impacto Esperado:** Aumentar la velocidad y eficiencia de la aplicación, mejorando la satisfacción del usuario.
3. **Expansión de Funcionalidades:**
    a. **Propuesta:** Explorar la integración de nuevas características en la aplicación móvil, como la autenticación biométrica y realizar pedidos por voz.
    b. **Impacto Esperado:** Ampliar la funcionalidad de la aplicación y mejorar la seguridad y la experiencia del usuario.
4. **Documentación y Mantenimiento:**
    a. **Propuesta:** Mejorar la documentación técnica del código y los procesos de desarrollo para facilitar el mantenimiento y futuras actualizaciones.
    b. **Impacto Esperado:** Facilitar el trabajo de desarrolladores futuros y asegurar una transición suave en el mantenimiento del sistema.
5. **Escalabilidad del Sistema:**
    a. **Propuesta:** Evaluar y mejorar la escalabilidad del sistema para manejar un mayor número de usuarios y solicitudes, ajustando el backend según sea necesario.

b. **Impacto Esperado:** Asegurar que la aplicación pueda crecer y adaptarse a futuras necesidades sin comprometer su rendimiento.

Resumen

El proyecto ha logrado cumplir con los objetivos establecidos, ofreciendo una solución efectiva en la aplicación móvil con React Native. Las conclusiones destacan el éxito en la implementación y la integración, mientras que los trabajos futuros proponen mejoras clave para optimizar la experiencia del usuario, el rendimiento y la escalabilidad del sistema.

5 APÉNDICES

5.1 Guía original del Trabajo Fin de Título

Este trabajo consiste en la realización de una aplicación móvil para la gestión de comandas en hostelería centrada en el personal de sala/terraza. El proceso debe ser útil y sencillo mediante una experiencia de usuario (UX) óptima, con el objeto de facilitar el trabajo a los empleados y dar un servicio rápido y fiable a los clientes. Se deja a elección del alumnado la adopción del enfoque habitual basado en menús y opciones táctiles o bien uno más avanzado basado en reconocimiento de voz. En cualquier caso, deberá plantearse un sistema que aporte claras ventajas sobre el sistema tradicional basado en notas o en el uso de la memoria.

5.1.1 Objetivos del TFG

- **Desarrollar una aplicación móvil multiplataforma** utilizando React Native con TypeScript que permita la gestión eficiente de pedidos en un entorno de restauración.
- **Implementar un backend robusto** con Spring Boot y Java, capaz de manejar la lógica de negocio, la autenticación de usuarios, y la integración con una base de datos en MongoDB Atlas.
- **Configurar e integrar un sistema de notificaciones push** a través de Firebase, permitiendo alertas en tiempo real para la actualización del estado de los pedidos.
- **Crear una interfaz web sencilla** para la gestión de la recuperación de contraseñas, desarrollada con React y TypeScript, complementando el sistema móvil.

5.2 Documentación de entrada

La información de entrada con la que contaba el alumno al inicio de este proyecto, siendo el alumno el que redacta esta misma documentación, se trata de una plantilla en Word ofrecida por el tutor y conocimientos previos basándose en experiencia yendo a bares, restaurantes o cualquier establecimiento que usara móvil

o Tablet para la realización de los pedidos. Agregar a esta que en varias ocasiones se realizaron preguntas sobre que funcionalidades agregarían a sus aplicaciones de trabajo para hacer más fácil su labor.

5.3 Instalación y configuración del sistema

5.3.1 Backend en java Spring Boot

5.3.1.1 Pre-requisitos

- Java Develpment Kit (JDK) 11 o superior.
- Maven 3.6 o superior.
- IDE como IntelliJ IDEA.

5.3.1.2 Descarga y configuración

- Clonar el repositorio: git clone https://github.com/mjmp0027/DineCrewBackend.git
- cd <nombre_del_directorio>
- Instalar dependencias: mvn install
- Configurar la base de datos:
    - Crear una cuenta en MongoDB Atlas y crear una base de datos con las colecciones ‘users’, ‘mesas’, ‘notificaciones’, ‘cuentas’, y ‘pedidos’.
    - Obtener la cadena de conexión de MongoDB Atlas desde el panel de MongoDB Atlas como se explica en la iteración 1.
    - Crear un archivo en la raíz del proyecto llamado env.env y agregar las variables necesarias para que el archivo ‘application.properties’ se quede configurado.
- Configurar Firebase:
    - Crea un proyecto en Firebase Console.

  - Seguir los pasos ofrecidos en los detalles de implementación de la iteración 4 hasta la ilustración 3.104 y seguir también los pasos que aparecen debajo.

- Ejecutar el comando ‘mvn spring-boot:run’
- El backend debería estar corriendo en ‘http://localhost:8080’.

5.3.2 Aplicación móvil en React Native con TypeScript

5.3.2.1 Pre-requisitos

- Node.js 18 o superior.
- Android Studio y SDK.
- Gradle.
- IDE como IntelliJ IDEA.

5.3.2.2 Descarga y configuración

- Clonar el repositorio: git clone https://github.com/mjmp0027/DineCrewFrontend.git
- cd <nombre_del_directorio>
- Instalar las dependencias: ‘npm install’
- Configurar Firebase:
  - Seguir los pasos mostrados en la sección 3.4.3 Detalles de implementación de la iteración 4, referentes al Frontend.
- Ejecutar la aplicación: ‘npm react-native run-android’ o para dispositivos físicos ‘npm react-native run-android –variant=release’

La aplicación móvil debería desplegarse en el emulador o dispositivo conectado.

5.3.3 Aplicación Web en React Con TypeScript

5.3.3.1 Pre-requisitos

- Node.js 18 o superior

- IDE como IntelliJ IDEA

5.3.3.2 Descarga y Configuración

- Clonar el repositorio: git clone https://github.com/mjmp0027/DineCrewWeb.git
- cd <nombre_del_directorio>
- npm start
- Comprobar la ip asociada a la aplicación web ya que será la que hay que usar para poder acceder a la vista, en mi caso se ha establecido la siguiente, esto va variando, dependiendo de la red wifi o Ethernet a la que estemos conectados, esto repercutirá a la hora de cambiar la contraseña.

Ilustración 5-1 Ip web

5.3.4 Documentación de errores

Es posible que no se pueda resetear la contraseña porque el backend está configurado para una ip específica. Como se indica más arriba la ip varía dependiendo de la red a la que estemos conectados, para ello una vez se haya arrancado la aplicación web comprobar la ip que indica 'On Your Network' y copiarla y pegarla en los lugares indicados a continuación:

Archivo: src/main/java/com/dinecrew/dinecrewbackend/config/WebConfig.java

```java
@Configuration
public class WebConfig implements WebMvcConfigurer {

    no usages  mjmp0027
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping( pathPattern: "/**")
                .allowedOrigins("http://192.168.0.127:3000")
                .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                .allowedHeaders("*")
                .allowCredentials(true);
    }
}
```

Ilustración 5-2 Archivo WebConfig.java

Sustituir la url por la obtenida

Archivo: src/ResetPassword.js

Este se encuentra en el proyecto react para la aplicación web

```javascript
const response : Response  = await fetch( input: 'http://192.168.0.127:8080/api/auth/reset-password', init: {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify( value: { token, newPassword }),
});

const data = await response.json();

if (response.ok) {
    setMessage(data.message);
} else {
    setMessage(data.message);
}
```

Ilustración 5-3 Archivo ResetPassword.js

Y, por último

Archivo:

src/main/java/com/dinecrew/dinecrewbackend/usuarios/UserService.java

```java
1 usage  mjmp0027
public void sendPasswordResetToken(String username) {
    Optional<User> userOpt = repository.findByUsername(username);
    if (userOpt.isEmpty()) {
        throw new RuntimeException("Usuario no encontrado");
    }

    User user = userOpt.get();
    String token = UUID.randomUUID().toString();
    user.setResetPasswordToken(token);

    repository.save(user);

    String resetUrl = "http://192.168.0.127:3000/reset-password/" + token;
    String emailContent = "Para restablecer tu contraseña, haz clic en el siguiente enlace:\n" + resetUrl;

    try {
        emailService.sendEmail(user.getEmail(),  subject: "Restablecer contraseña", emailContent);
    } catch (MessagingException e) {
        e.printStackTrace();
        throw new RuntimeException("Error al enviar el correo electrónico");
    }
}
```

Ilustración 5-4 Archivo UserService.java

Una vez realizados estos pasos ejecutar de nuevo el backend usando ‘mvn spring-boot:run’.

6 DEFINICIONES Y ABREVIATURAS

Json Web Token (JWT) [1]: es un estándar de autenticación y autorización que permite la transmisión segura de información entre dos partes como un objeto JSON. El token es firmado digitalmente, lo que garantiza la integridad y autenticidad de los datos, y suele utilizarse para verificar la identidad de los usuarios en aplicaciones web y móviles.

Entorno de desarrollo integrado (IDE) [2]: es una aplicación de software que proporciona herramientas completas para los desarrolladores, como un editor de código, depurador y compilador, en un solo lugar, facilitando la escritura, prueba y depuración de programas.

Backend [3]: es la parte del desarrollo de software que se encarga de gestionar la lógica del servidor, las bases de datos, la autenticación de usuarios y la comunicación con el frontend. Es responsable de procesar las solicitudes de los usuarios, realizar operaciones con los datos y enviar las respuestas al frontend. El backend es esencial para el funcionamiento de aplicaciones web y móviles, ya que maneja toda la lógica y el procesamiento de datos que ocurre detrás de escena.

NoSQL [4]: es un tipo de base de datos que no utiliza el modelo relacional tradicional basado en tablas. En lugar de eso, emplea diferentes estructuras como documentos, gráficos, pares clave-valor o columnas para almacenar y gestionar datos. NoSQL está diseñado para manejar grandes volúmenes de datos no estructurados o semi-estructurados, ofreciendo flexibilidad, escalabilidad y un alto rendimiento, lo que lo hace ideal para aplicaciones que requieren manejar datos diversos y de rápido crecimiento.

Notificación push [5]: es un mensaje que una aplicación envía a un dispositivo móvil o computadora, apareciendo en la pantalla sin que el usuario tenga que abrir la aplicación. Estas notificaciones se utilizan para alertar al usuario de eventos importantes, actualizaciones o mensajes en tiempo real, incluso cuando la aplicación no está en uso.

Stakeholder [6]: es cualquier persona, grupo u organización que tiene un interés o preocupación en un proyecto y puede influir o ser influenciado por él.

API (Interfaz de programación de aplicaciones) [7]: es un conjunto de reglas y protocolos que permite a diferentes aplicaciones o servicios comunicarse entre sí. Facilita la integración y el intercambio de datos o funcionalidades al definir cómo deben solicitarse y proporcionarse estos recursos.

JSON (JavaScript Object Notation) [8]: es un formato de intercambio de datos ligero y legible para humanos, que se utiliza para representar objetos y estructuras de datos en texto. JSON es ampliamente utilizado en aplicaciones web para transmitir datos entre un servidor y un cliente, y se basa en una sintaxis de pares clave-valor.

Wireframe [9]: es un esquema visual básico que representa la estructura y el diseño de una página o interfaz de usuario. Los wireframes muestran la disposición de los elementos, como botones, menús y campos de entrada, sin entrar en detalles de diseño gráfico, con el objetivo de planificar y definir la funcionalidad y la organización del contenido.

Mockup [10]: es una representación visual detallada de un diseño de interfaz de usuario o producto, que ilustra cómo se verá el producto final. A diferencia de los wireframes, los mockups incluyen elementos gráficos, colores y tipografía, proporcionando una vista más precisa del diseño estético y funcional.

Storyboard [11]: es una serie de ilustraciones o imágenes secuenciales que representan las principales escenas o pasos de un proceso, proyecto o historia. Se utiliza para planificar y visualizar la narrativa, el flujo de trabajo o la interacción en proyectos de diseño, cine, animación, o desarrollo de productos.

Node.js [12]: es un entorno de ejecución de JavaScript basado en el motor V8 de Google Chrome, que permite ejecutar código JavaScript en el servidor. Se utiliza

para construir aplicaciones de red rápidas y escalables, aprovechando su arquitectura de entrada/salida no bloqueante y orientada a eventos.

Cluster [13]: es un grupo de servidores o nodos interconectados que trabajan juntos para mejorar la disponibilidad, rendimiento y escalabilidad de un sistema. En un clúster, los nodos colaboran para realizar tareas compartidas y pueden actuar como una sola unidad para distribuir la carga de trabajo y garantizar la continuidad del servicio.

URI [14]: es una cadena de caracteres utilizada para identificar de manera única un recurso en la web o en una red. Un URI puede ser una URL (Uniform Resource Locator), que especifica la dirección de un recurso, o un URN (Uniform Resource Name), que proporciona un nombre único para el recurso sin especificar su ubicación.

Maven [15]: es una herramienta de gestión y automatización de proyectos para Java que se utiliza para construir, gestionar dependencias y organizar el ciclo de vida del desarrollo de software. Maven simplifica la configuración del proyecto, las compilaciones y la gestión de bibliotecas mediante un archivo de configuración XML (pom.xml).

Endpoint [16]: es una URL o dirección específica en una API (Interfaz de Programación de Aplicaciones) que permite a los clientes interactuar con el servidor. Cada endpoint corresponde a una funcionalidad o recurso particular que la API expone, como obtener, enviar, actualizar o eliminar datos.

Hook [17]: en programación, un *hook* es un mecanismo que permite a los desarrolladores insertar o modificar el comportamiento de un software sin alterar su código fuente principal. Los *hooks* son puntos de extensión que permiten ejecutar funciones personalizadas en momentos específicos del ciclo de vida de una aplicación o cuando ocurre un evento particular.

Deserializar [18]: configurar datos de un formato estructurado (como JSON o XML) de vuelta a un objeto o estructura de datos en un lenguaje de programación, para que pueda ser utilizado por el software.

Query [19]: es una consulta que permite buscar y recuperar datos de una base de datos MongoDB utilizando criterios específicos. Estas consultas se expresan en un formato JSON y pueden incluir condiciones, operadores, y proyecciones para filtrar y seleccionar los documentos que cumplen con los requisitos establecidos.

7 BIBLIOGRAFÍA

[1] React Navigation (2024). *React navigation versión 6.x.* Obtenido de https://reactnavigation.org/docs/getting-started (11-07-2024)

[2] Mail Trap. (2024). *MailTrap.* Obtenido de https://mailtrap.io/ (14-07-2024)

[3] React Native (2024). *React Native dev 0.74*. Obtenido de https://reactnative.dev/ (11-07-2024)

[4] GitHub (2024). *Github*. Obtenido de https://github.com/ (03-09-2024)

[5] Nmpjs (2014). *Node-packages for node.js*. Obtenido de https://www.npmjs.com/package/packages (18-07-2024)

[6] Axure (2024). *Axure RP 10*. Obtenido de https://www.axure.com/ (18-08-2024)

[7] Htmlcolorcodes (2024). Htmlcolorcodes. Obtenido de https://htmlcolorcodes.com/es/ (10-08-2024)

[8] Spring Security (2024). *spring-security-docs 6.3.3 API*. Obtenido de https://docs.spring.io/spring-security/site/docs/current/api/ (06-07-2024)

[9] Spring Security (2024). *Getting-spring-security*. Obtenido de https://docs.spring.io/spring-security/reference/getting-spring-security.html (06-07-2024)

[10] Spring Data (2024). *Spring data mongodb 4.3.3*. Obtenido de https://spring.io/projects/spring-data-mongodb (18-07-2024)

[11] Spring Data (2024). *Spring data mongodb 4.3.3 API*. Obtenido de https://docs.spring.io/spring-data/mongodb/docs/current/api/ (18-07-2024)

[12] Lombok (2024). *Project lombok*. Obtenido de https://mvnrepository.com/artifact/org.projectlombok/lombok (04-07-2024)

[13] JsonWebToken (2024). *Jjwt-api 0.11.2*. Obtenido de https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api/0.12.6 (14-07-2024)

[14] JsonWebToken (2024). *Jjwt-impl 0.11.2*. Obtenido de https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl/0.12.6 (14-07-2024)

[15] JsonWebToken (2024). Jjwt-jackson 0.11.2. Obtenido de https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson/0.12.6 (14-07-2024)

[16] Maven Repository (2022). *Firebase-admin 8.2.0*. Obtenido de https://mvnrepository.com/artifact/com.google.firebase/firebase-admin/8.2.0 (06-08-2024)

[17] Github (2024). *React-native-async-storage*. Obtenido de https://github.com/react-native-async-storage/async-storage/releases/tag/v1.23.1 (11-07-2024)

[18] React Native (2024). *React-native-firebase/messaging 20.3.0*. Obtenido de https://rnfirebase.io/messaging/usage (06-08-2024)

[19] Npmjs (2024). *React-native-firebase/app 20.4.0*. Obtenido de https://www.npmjs.com/package/@react-native-firebase/app (07-08-2024)

[20] Npmjs (2024). *React-navigation/native 6.1.17*. Obtenido de https://www.npmjs.com/package/@react-navigation/native (11-0-2024)

[21] Npmjs (2024). *React-navigation/native-stack 6.10.0*. Obtenido de https://www.npmjs.com/package/@react-navigation/native-stack (11-0-2024)

[22] Npmjs (2024). *React-papaparse 5.4.1*. Obtenido de https://www.npmjs.com/package/react-papaparse (31-07-2024)

[23] React Native Csv (2021). *React-native-csv 0.2.0*. Obtenido de https://react-native-csv.js.org/ (31-07-2024)