INGENIERÍA DEL SOFTWARE

Curso 2025 / 2026

Memoria de Prácticas

Grupo Prácticas M122, Lunes A 10:00-12:30

Autora: Paula Melero Laviña

NIP: 901449

Autora: Ainhoa Carolina

Romero Bustamante

NIP: 901808

Resumen

Este documento reúne las fases de **Análisis**, **Diseño** y **Modelado** de una **aplicación móvil** que se encargará de gestionar las reservas en una tienda de alquiler de quads. Su objetivo principal es centralizar y optimizar la administración de las operaciones del propietario.

En la **primera fase del proyecto (Práctica 1)**, nos enfocamos en capturar los requisitos y definir el modelo de análisis. Se establecieron los casos de uso y se creó un primer Diagrama de Clases a Nivel de Análisis, junto con los Diagramas de Secuencia correspondientes, lo que sentó las bases del sistema y ayudó a entender sus responsabilidades funcionales. La **Práctica 2 complementó este trabajo al diseñar la capa de vista (UI)**, transformando los requisitos funcionales en un storyboard o prototipo de interfaz que asegura una experiencia de usuario adecuada.

En **esta tercera fase (Práctica 3)**, nos centramos en el Diseño del Sistema y el Diseño de Objetos, convirtiendo el modelo conceptual de análisis en un modelo de diseño técnico y constructivo. El diseño se basa en una arquitectura de software de tres capas, inspirada en el patrón Model-View-ViewModel (MVVM) y respaldada por la persistencia de datos local a través de Room (DAO/Database).

Los principales artefactos generados en esta fase son:

Modelo lógico de la base de datos relacional, que define las tablas, claves primarias y foráneas, tipos de datos y restricciones que garantizan la integridad. Parte del modelo conceptual, con el diseño del modelo Entidad-Relación, y lo normaliza para evitar redundancias y anomalías de actualización. Incluye reglas (NOT NULL, CHECK, UNIQUE) y decisiones de clave artificial/natural según el caso. Este modelo nos ha servido como base para la definición de las sentencias SQL de creación y destrucción de la base de datos en SQLite.

Diagrama de Clases a Nivel de Diseño, que modela la estructura estática del sistema con un enfoque orientado al código, estableciendo las clases ViewModel, Repository e interfaces DAO para implementar la arquitectura MVVM.

Diagrama de Paquetes, que define la estructura modular y la organización lógica del sistema, asegurando un bajo acoplamiento entre las capas de Presentación (ui), Lógica (repository) y Persistencia (data).

Diagrama de Componentes, que describe la arquitectura lógica, separando UI, servicios de dominio y acceso a datos mediante interfaces. Permite ver quién ofrece y quién requiere servicios, reduciendo dependencias. Este diagrama facilita el cambio de implementación sin afectar al resto de capas.

Diagrama de Despliegue, que representa la arquitectura física: nodos de ejecución y artefactos instalables/ejecutables. Indica dónde vive cada componente lógico y las dependencias entre artefactos en tiempo de ejecución. En nuestro caso, el APK contiene la app y la base de datos se materializa como fichero SQLite en el dispositivo.

Diagramas de Secuencia a Nivel de Diseño, que ilustran el modelado dinámico en detalle, mostrando las trazas de mensajes entre las instancias de las clases diseñadas. Estos diagramas

validan los flujos principales (Creación, Eliminación y Consulta) y demuestran cómo se logra la reactividad y la cohesión del sistema.

Durante la **práctica 4** implementamos la aplicación en **Android Studio**, materializando la arquitectura y validando el modelo de datos. En la **práctica 5** continuamos esa implementación añadiendo el envío de mensajes al cliente, aplicando el patrón **Bridge** para soportar WhatsApp y SMS. Esto lo reflejamos en los nuevos diagramas de clases, paquetes, despliegue y secuencia actualizados.

Finalmente, en la **práctica 6** se aborda la fase de **Pruebas del Sistema**, cuyo objetivo es validar el correcto funcionamiento de la aplicación y evaluar su robustez frente a distintos escenarios de uso. En esta fase se diseñan y ejecutan **pruebas unitarias de caja negra** sobre los métodos del repositorio, aplicando técnicas como las **particiones de equivalencia**, así como **pruebas de sistema de volumen** mediante el análisis de valores límite y una **prueba de sobrecarga** para determinar los umbrales de funcionamiento del sistema. Los resultados obtenidos permiten analizar el comportamiento de la aplicación, detectar posibles errores o limitaciones y realizar las correcciones necesarias, cerrando así el ciclo de desarrollo y asegurando la calidad del producto final.

Índice

Introducción y objetivos

Este informe presenta las etapas de Análisis, Diseño, Modelado e Implementación de una aplicación móvil que se encargará de gestionar las reservas en una tienda de alquiler de quads. Después de definir los requisitos y crear el modelo de análisis en la Práctica 1, diseñar la interfaz de usuario (Storyboard) en la Práctica 2, enfocarnos en el Diseño del Sistema y el Diseño de Objetos en la práctica 3, implementar en la práctica 4 y 5, pasamos finalmente a la última fase .

El objetivo principal de esta fase final es validar y verificar el correcto funcionamiento del sistema desarrollado, asegurando que la aplicación cumple los requisitos funcionales y no funcionales definidos en las fases anteriores. Para ello, se diseñan y ejecutan pruebas unitarias de caja negra, pruebas de sistema de volumen y una prueba de sobrecarga, aplicando técnicas como las particiones de equivalencia y el análisis de valores límite. El análisis de los resultados obtenidos permite detectar errores, evaluar la robustez del sistema y realizar las correcciones necesarias, consolidando así la arquitectura MVVM y la implementación basada en Room desarrolladas previamente.

Requisitos

Catálogo de requisitos

En este apartado llevamos a cabo el estudio de los requisitos de la aplicación, distinguiéndose entre aquellos que son funcionales o no funcionales.

Los requisitos funcionales serían los siguientes:

Código	Descripción
RF1	El sistema debe permitir crear quads.
RF2	El sistema debe permitir consultar un listado de quads pudiendo ser ordenados por matrícula, tipo o precio.
RF3	El sistema debe permitir modificar los quads.
RF4	El sistema debe permitir eliminar los quads.
RF5	El sistema debe permitir la creación de una reserva.
RF6	El sistema debe verificar que la fecha de devolución de una reserva sea igual o posterior a la del día de recogida.
RF7	El sistema debe verificar que no se produzcan solapes en las reservas de los quads.
RF8	El sistema debe garantizar que la cantidad de cascos entregados no exceda el número solicitado, sin excepción según el tipo de quad.
RF9	El sistema debe permitir consultar un listado de reservas pudiendo ser ordenadas por nombre de cliente, número móvil, fecha de recogida o fecha de devolución.
RF10	El sistema debe permitir la modificación de las reservas.
RF11	El sistema debe permitir la eliminación de las reservas.
RF12	El sistema debe permitir el cálculo automático del precio total en la creación de una reserva.

RF13	El sistema debe permitir el cálculo automático del precio total en la modificación de la reserva en caso de realizar una selección distinta de quads.
RF14	El sistema debe verificar que el precio total se mantenga aunque se modifique posteriormente el precio del alquiler diario de un quad.
RF15	El sistema debe generar un mensaje de la información de la reserva.

A continuación, describimos los requisitos no funcionales:

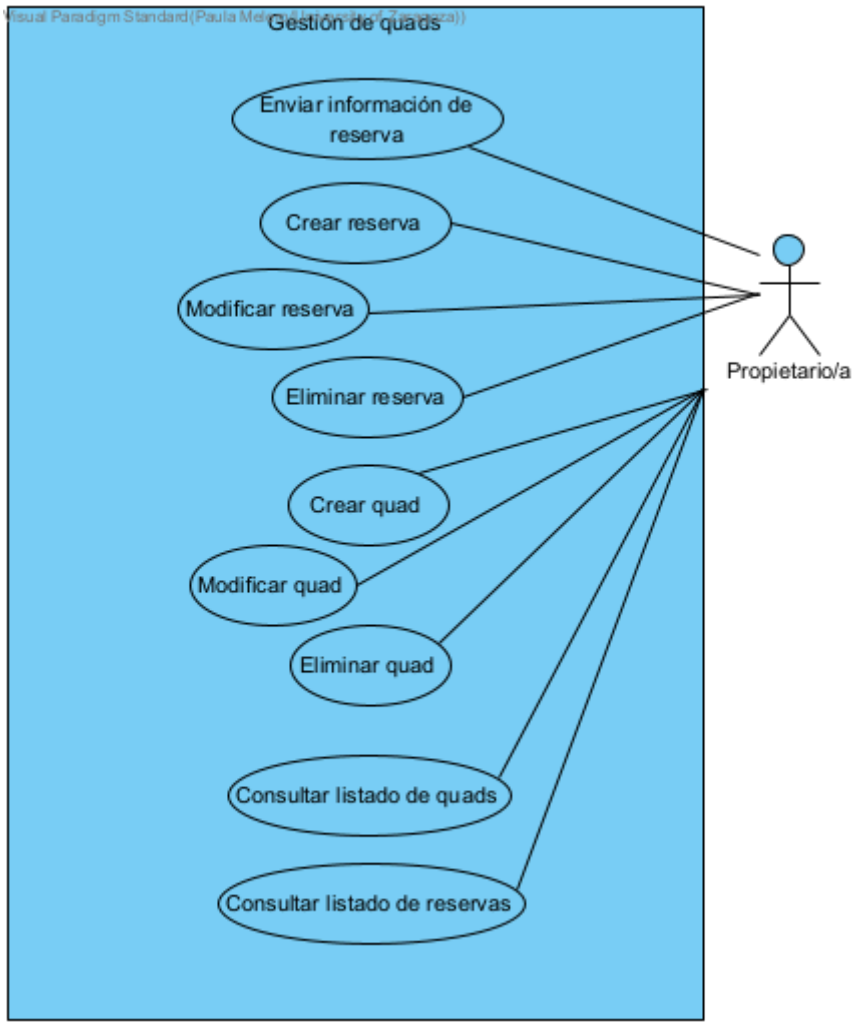
Código	**Descripción**
RNF1	El sistema debe funcionar en dispositivos móviles.
RNF2	El sistema debe contar con un método de envío de mensajes a los clientes.
RNF3	El sistema debe soportar un máximo de 100 quads.
RNF4	El sistema debe soportar un máximo de 20000 reservas almacenadas.
RNF5	El sistema debe cumplir la ley de protección de datos y privacidad.

En este apartado también especificamos las restricciones que debe cumplir el sistema:

Restricciones
El sistema debe soportar el sistema operativo Android.
El registro de un quad debe guardar la matrícula, el tipo de quad, el precio en euros del alquiler por día, y una breve descripción en texto libre acerca de cualquier característica relevante como información sobre la marca, modelo o color.
El formato de la matrícula consiste en un número de cuatro cifras seguido de tres letras.
El tipo de quad debe ser o monoplaza o biplaza.

El formato de la reserva debe consistir en el nombre de cliente, el número móvil del cliente, la fecha de recogida y devolución, y una selección de quads junto al número de cascos necesarios para su conducción.
El número de cascos necesarios para la conducción de un quad puede tomar el valor 0, si ya se dispone de un casco; 1, como máximo en el caso de un quad monoplaza; y 2, como valor máximo en el caso de un quad biplaza.

Diagrama de casos y descripción de flujos de eventos



Aquí se recogen los casos de uso que resumen qué puede hacer el propietario dentro del sistema y cómo interactúa con él.

Con este diagrama podemos obtener una visión clara de las funcionalidades disponibles y de los diferentes flujos de eventos que puede soportar el sistema, que son los siguientes:

Crear reserva

Caso de uso: Crear reserva

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el usuario solicita crear una reserva.
- El usuario introduce el nombre y número móvil del cliente.
- El usuario selecciona la fecha de recogida y devolución.
- El usuario selecciona los quads a reservar junto al número de cascos necesarios para su conducción.
- El sistema guarda la reserva con los datos proporcionados por el cliente.

Modificar reserva

Caso de uso: Modificar reserva

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el propietario/a selecciona una reserva existente y solicita su modificación.
- El propietario modificar uno o varios de los siguientes elementos:
    - Nombre del cliente.
    - Número de móvil del cliente.
    - Fecha de recogida.
    - Fecha de devolución.
    - Número de cascos solicitados (sin superar el máximo permitido por quad).
    - Quads asociados a la reserva (añadir o eliminar quads).
- El sistema valida los datos modificados.
- El sistema guarda la reserva actualizada.

Eliminar reserva

Caso de uso: Eliminar reserva

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el usuario selecciona una reserva existente y solicita su borrado.
- El sistema borra la reserva.

Enviar información de reserva

Caso de uso: Enviar información de reserva

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el usuario selecciona una reserva existente y solicita enviar al móvil del cliente la información de la misma.
- El sistema envía la información de la reserva al cliente.

Consultar listado de reservas

Casos de uso: Consultar listado de reservas

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el usuario propietario solicita listar las reservas registradas en el sistema.
- El sistema genera un listado con las reservas registradas, ordenadas, por defecto, por fecha de recogida.

Flujo de eventos alternativo

- El caso de uso comienza cuando el usuario selecciona un filtro de ordenación por fecha de recogida.
- El sistema genera un listado con las reservas registradas ordenadas por fecha de recogida.

Flujo de eventos alternativo

- El caso de uso comienza cuando el usuario selecciona un filtro de ordenación por fecha de devolución.
- El sistema genera un listado con las reservas registradas ordenadas por fecha de devolución.

Flujo de eventos alternativo

- El caso de uso comienza cuando el usuario selecciona un filtro de ordenación por nombre de cliente.

- El sistema genera un listado con las reservas registradas ordenadas por nombre de cliente alfabéticamente.

Flujo de eventos alternativo

- El caso de uso comienza cuando el usuario selecciona un filtro de ordenación por número de móvil.
- El sistema genera un listado con las reservas registradas ordenadas por número de móvil.

Crear quad

Caso de uso: Crear quad

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el usuario propietario solicita crear un quad.
- El usuario da la matrícula y el tipo de quad a alquilar.
- El usuario propietario da el precio en euros del alquiler por día y una breve descripción en texto libre acerca de cualquier característica relevante.
- El sistema guarda el quad con los datos proporcionados por el usuario.

Modificar quad

Caso de uso: Modificar quad

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el propietario/a selecciona un quad existente y solicita su modificación.
- El propietario/a puede modificar uno o varios de los siguientes datos:
    - Tipo de quad (por ejemplo, monoplaza o biplaza).
    - Precio de alquiler.
    - Descripción del quad.
- El sistema valida los datos introducidos.
- El sistema guarda el quad con las modificaciones realizadas**.**

Eliminar quad

Caso de uso: Eliminar quad

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el usuario propietario selecciona un quad existente y solicita su borrado.
- El sistema borra el quad.

Consultar listado de quads

Caso de uso: Consultar listado de quads

Actor: Propietario/a

Flujo de eventos principal

- El caso de uso se inicia cuando el usuario propietario solicita listar los quads registrados en el sistema.
- El sistema genera un listado con los quads registrados, ordenados, por defecto, por matrícula de manera alfanumérica.

Flujo de eventos alternativo

- El caso de uso comienza cuando el usuario selecciona un filtro de ordenación por matrícula.
- El sistema genera un listado con los quads registrados ordenados por matrícula de manera alfanumérica.

Flujo de eventos alternativo

- El caso de uso comienza cuando el usuario selecciona un filtro de ordenación por tipo.
- El sistema genera un listado con los quads registrados ordenados por tipo.

Flujo de eventos alternativo

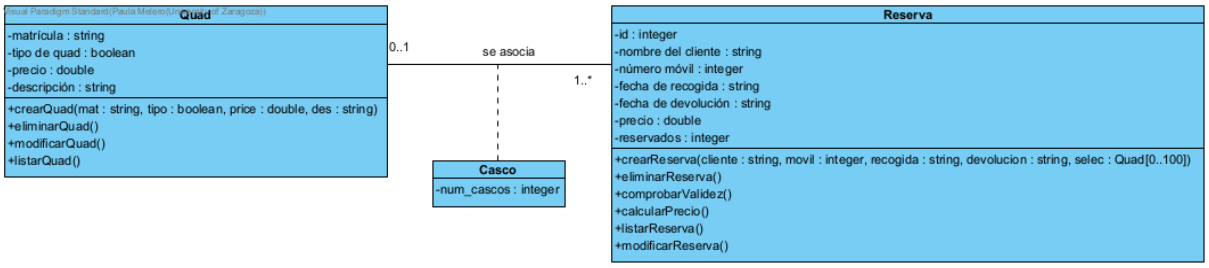
- El caso de uso comienza cuando el usuario selecciona un filtro de ordenación por precio.
- El sistema genera un listado con los quads registrados ordenados por precio ascendente.

Análisis

Hemos realizado un estudio del sistema y lo hemos plasmado mediante modelado estático y dinámico, utilizando diagramas y descripciones que representan el comportamiento esperado. Estos modelos permiten visualizar las interacciones entre el propietario y los distintos elementos que conforman el sistema.

Modelado estático

A continuación se presenta el diagrama de clases que refleja la estructura estática del sistema —clases, atributos, métodos y relaciones— y ayuda a entender cómo se organizan los datos internamente.



El sistema de alquiler de quads está representado mediante tres clases principales: *Quad*, *Casco* y *Reserva*.

La clase *Quad* modela los vehículos disponibles para el alquiler, incluyendo atributos como matrícula, tipo, precio y descripción. Además, dispone de métodos que permiten gestionar su ciclo de vida, como la creación, modificación, eliminación y listado de quads.

La clase *Casco* se asocia con los quads para indicar el número de cascos asignados a un vehículo dentro de una reserva, asegurándose así de que no se excedan los límites permitidos.

Por otro lado, la clase *Reserva* gestiona toda la información relacionada con las reservas realizadas por los clientes, tales como el nombre del cliente, el número de móvil, las fechas de recogida y devolución, y los quads seleccionados. También incluye métodos para crear, modificar y eliminar reservas, comprobar su validez y calcular el precio total.

En conjunto, estas clases permiten representar de forma estructurada y coherente el funcionamiento del sistema de alquiler, garantizando una correcta organización de la información y facilitando la interacción entre los diferentes elementos del modelo.

Modelado dinámico

En esta sección incluimos los diagramas de secuencia que detallan, paso a paso, cómo se desarrollan las interacciones entre los objetos del sistema, especificando los métodos necesarios para llevar a cabo cada una de las funcionalidades del sistema. Cada uno de ellos representa un proceso concreto (crear quad, realizar una reserva, modificar datos, etc.).

Diagrama de secuencia - Crear quad

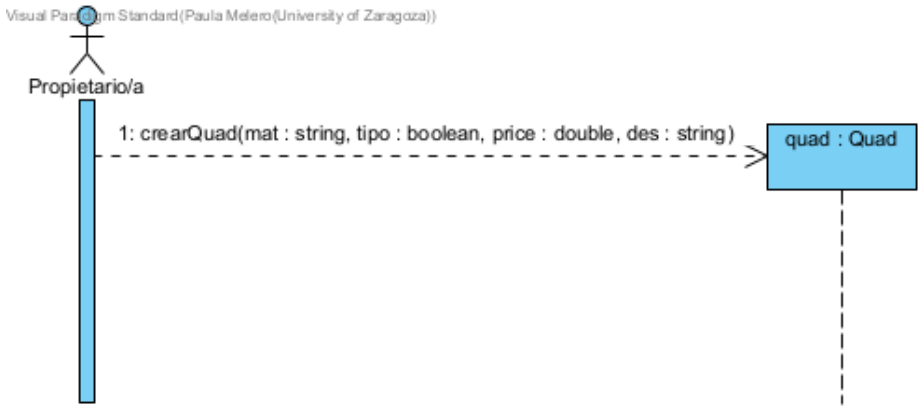


Diagrama de secuencia - Eliminar quad

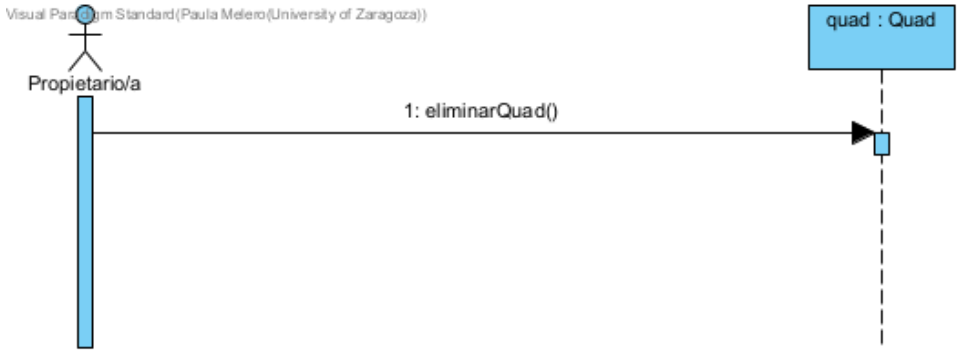


Diagrama de secuencia - Modificar quad

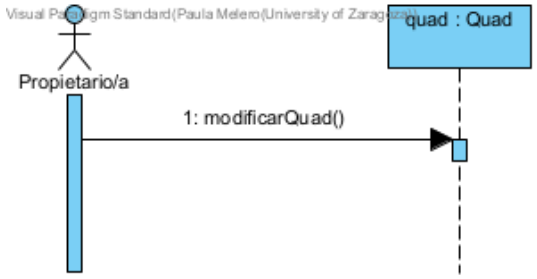


Diagrama de secuencia - Consultar listado de quads

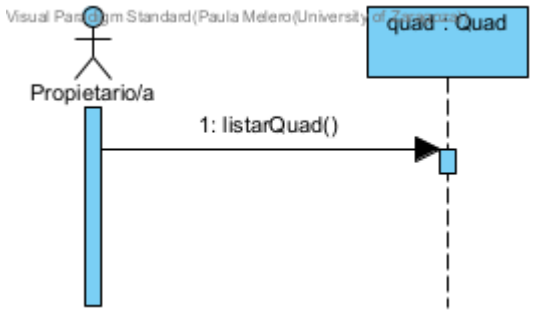


Diagrama de secuencia - Crear reserva

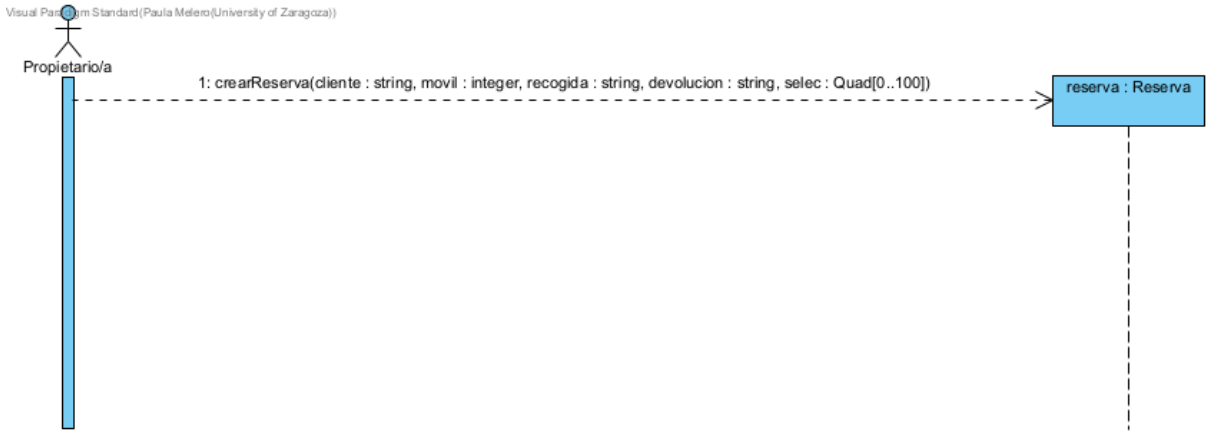


Diagrama de secuencia - Eliminar reserva

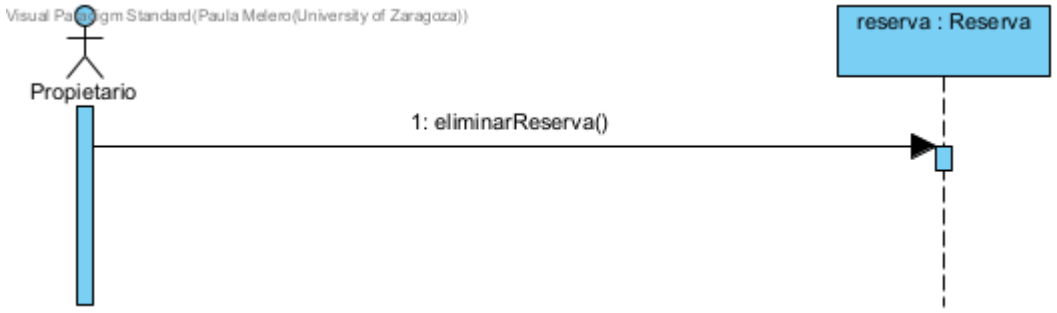


Diagrama de secuencia - Modificar reserva

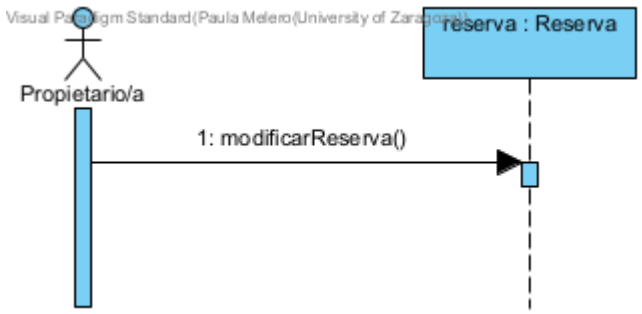


Diagrama de secuencia - Consultar listado de reservas

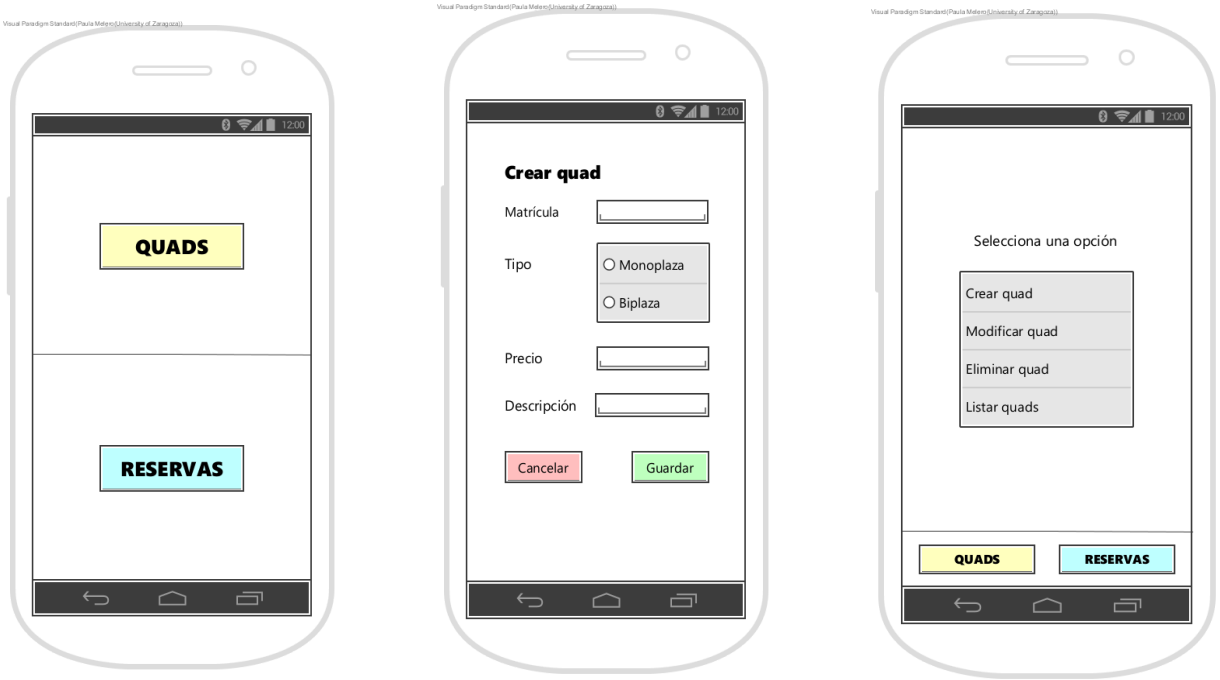
Descripción de la interfaz de usuario: Prototipado de pantallas

La interfaz desarrollada se basa en un enfoque minimalista y simple, centrado en la claridad visual y la facilidad de uso. Desde el principio se ha buscado evitar la sobrecarga de elementos gráficos, reduciendo al mínimo los distractores y priorizando la funcionalidad. La paleta cromática es deliberadamente limitada, utilizando solo unos pocos colores que permiten establecer asociaciones visuales claras: el azul se utiliza para todo lo relacionado con las reservas, mientras que el amarillo identifica las secciones vinculadas con los quads. De esta forma, el usuario puede orientarse de manera intuitiva mediante el color, sin necesidad de leer en detalle cada opción.

El diseño mantiene una consistencia interna muy marcada, lo que refuerza la sensación de coherencia y continuidad entre pantallas. Todos los botones conservan el mismo tamaño, forma, tipografía y posición relativa, garantizando una experiencia de uso uniforme. Esta coherencia visual y estructural facilita el aprendizaje y reduce el esfuerzo cognitivo del usuario, ya que los patrones de interacción se repiten de manera predecible en toda la aplicación.

Además, se ha incorporado un menú inferior permanente, que permite acceder en todo momento a las secciones principales de la aplicación. Gracias a esta barra fija, el usuario puede cambiar fácilmente entre las áreas de reservas y quads, manteniendo una navegación ágil y accesible desde cualquier punto del recorrido.

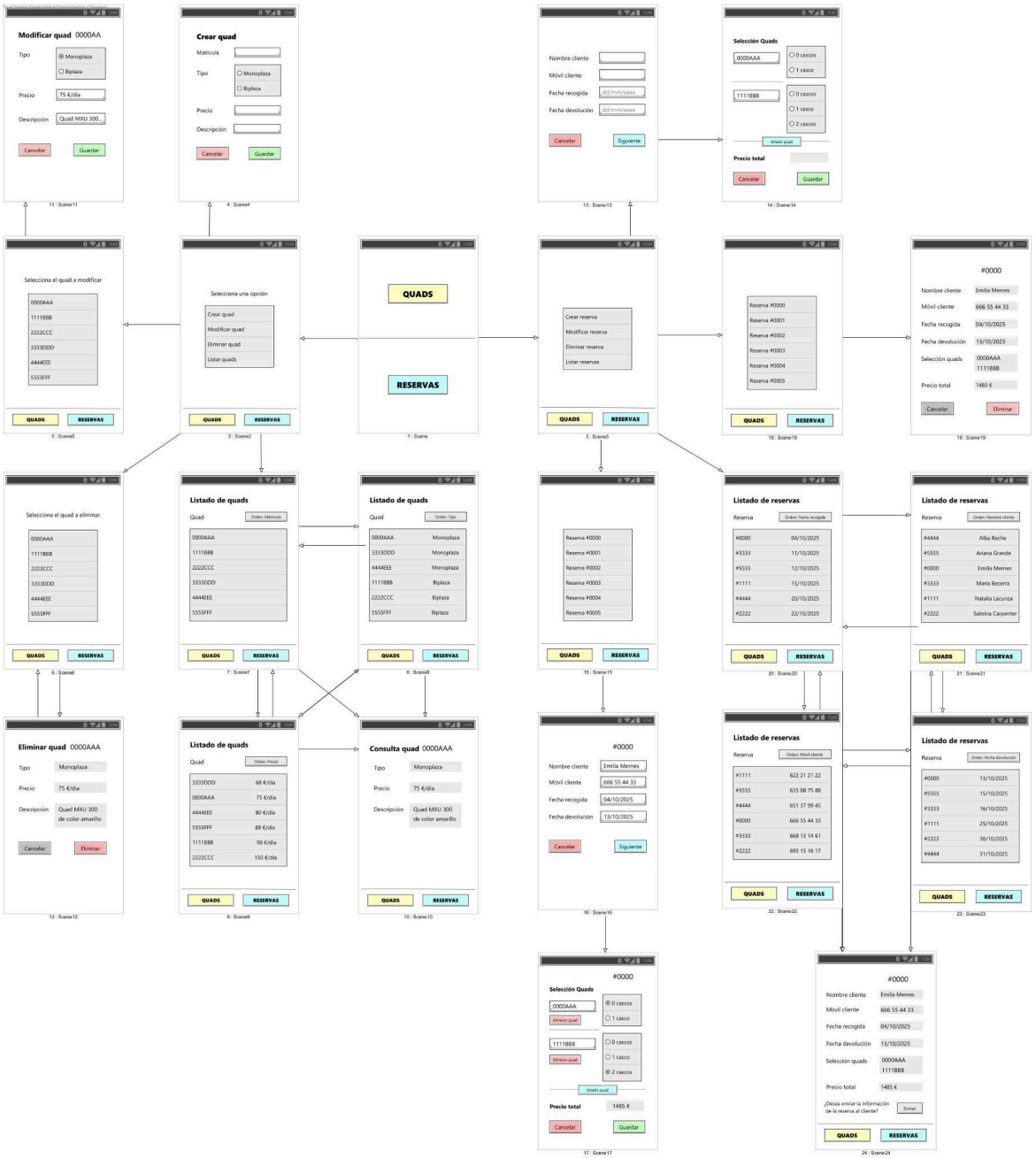
En conjunto, el prototipo presenta una interfaz limpia, equilibrada y coherente, pensada para que la interacción resulte natural, eficiente y visualmente agradable.



Descripción de la interfaz de usuario: Mapa de navegación

En este mapa de navegación se han representado las relaciones principales entre las pantallas de la aplicación. Es importante destacar que no se han incluido flechas de todas las pantallas hacia el menú principal de “Quads” y “Reservas”, ya que agregarlas haría que el diagrama se volviera demasiado confuso y difícil de interpretar.

Sin embargo, todas las pantallas siguen siendo accesibles a través del menú inferior, lo que garantiza que el usuario pueda navegar fácilmente hacia las secciones de Quads y Reservas sin perder claridad en el diagrama. De esta manera, se prioriza una visualización ordenada y comprensible del flujo de navegación principal.



Diseño del Sistema

Diagrama de paquetes

El Diagrama de Paquetes muestra la organización lógica del sistema y su estructura modular, reflejando la arquitectura en capas definida en el apartado anterior.

El sistema, como ya se ha mencionado, sigue el patrón arquitectónico MVVM (Model-View-ViewModel) característico de Android, y se encapsula dentro del paquete principal ***es.unizar.ingenieria_del_software.quad_reserva***, que se divide en los siguientes módulos:

- **ui**: contiene las clases de la interfaz de usuario (por ejemplo, ControlView), responsables de la presentación visual.
- **repository:** define la capa de acceso a datos, actuando como intermediaria entre la interfaz y las fuentes de información.
- **data**: agrupa los elementos de persistencia:
    - **model**: clases de dominio (*Quad_Diseño*, *Reserva_Diseño*).
    - **dao**: interfaces DAO (Data Access Object) encargadas de las operaciones sobre la base de datos local.
    - **database**: clase que inicializa y configura la base de datos.

Además de la estructura interna, el sistema interactúa con **otros paquetes externos:**

- **eina (es.unizar.eina)**: Representa un paquete externo al módulo principal de la aplicación.
    - **send**: Un submódulo encargado de gestionar el envío de mensajes con servicios externos. (Patrón Bridge)
- **android y androidx:** Paquetes de librerías esenciales del sistema operativo Android.
    - **android**: Se refiere a las clases y APIs fundamentales del SDK de Android.
    - **androidx**: Se refiere a las bibliotecas de soporte y componentes modernos de Jetpack.

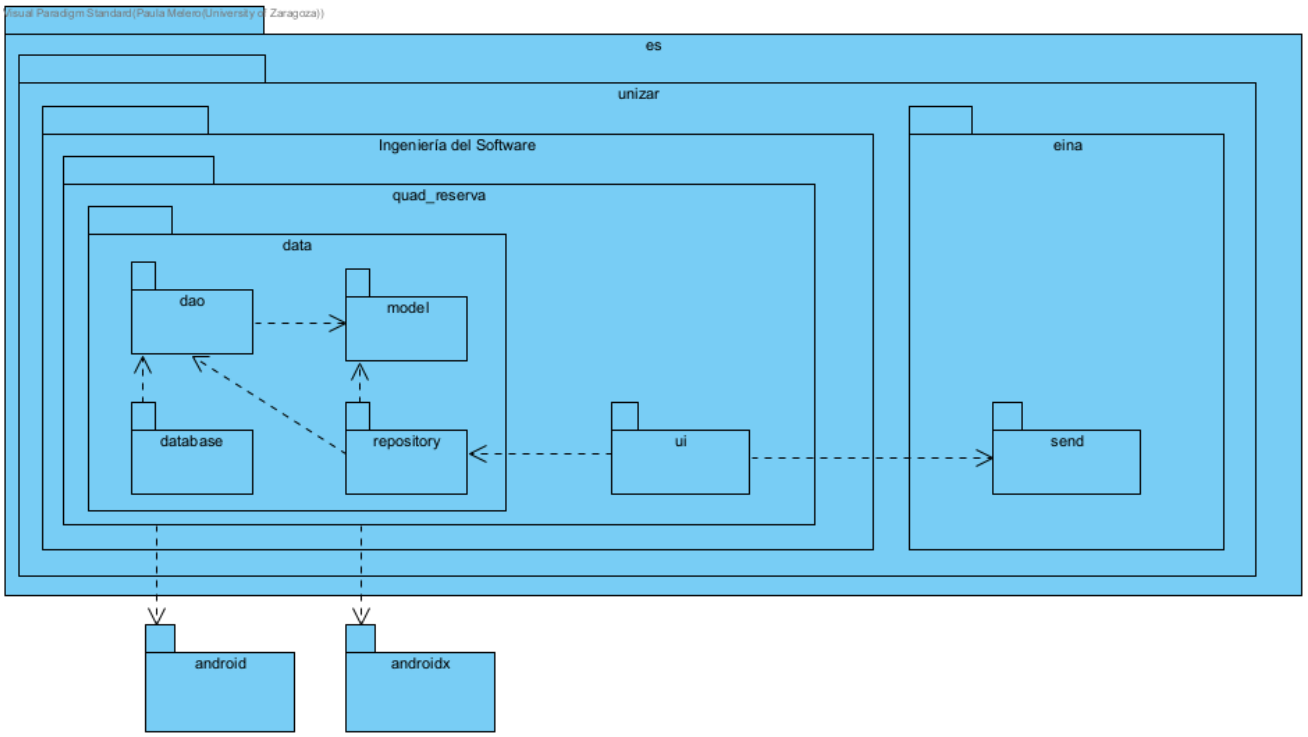


Diagrama de componentes

En nuestro **diagrama de componentes**, se muestra como el sistema se organiza por **capas con interfaces explícitas**: la *AplicaciónPropietario* consume los servicios de *GestorQuads* y *GestorReservas* mediante *IGestorQuads* e *IGestorReservas*. Ambos gestores dependen de un *Repository* que centraliza la persistencia y, a su vez, accede a la base de datos local (SQLite) a través de la interfaz *SQLiteAPI*. Esta separación reduce el acoplamiento entre UI, lógica de dominio y datos.

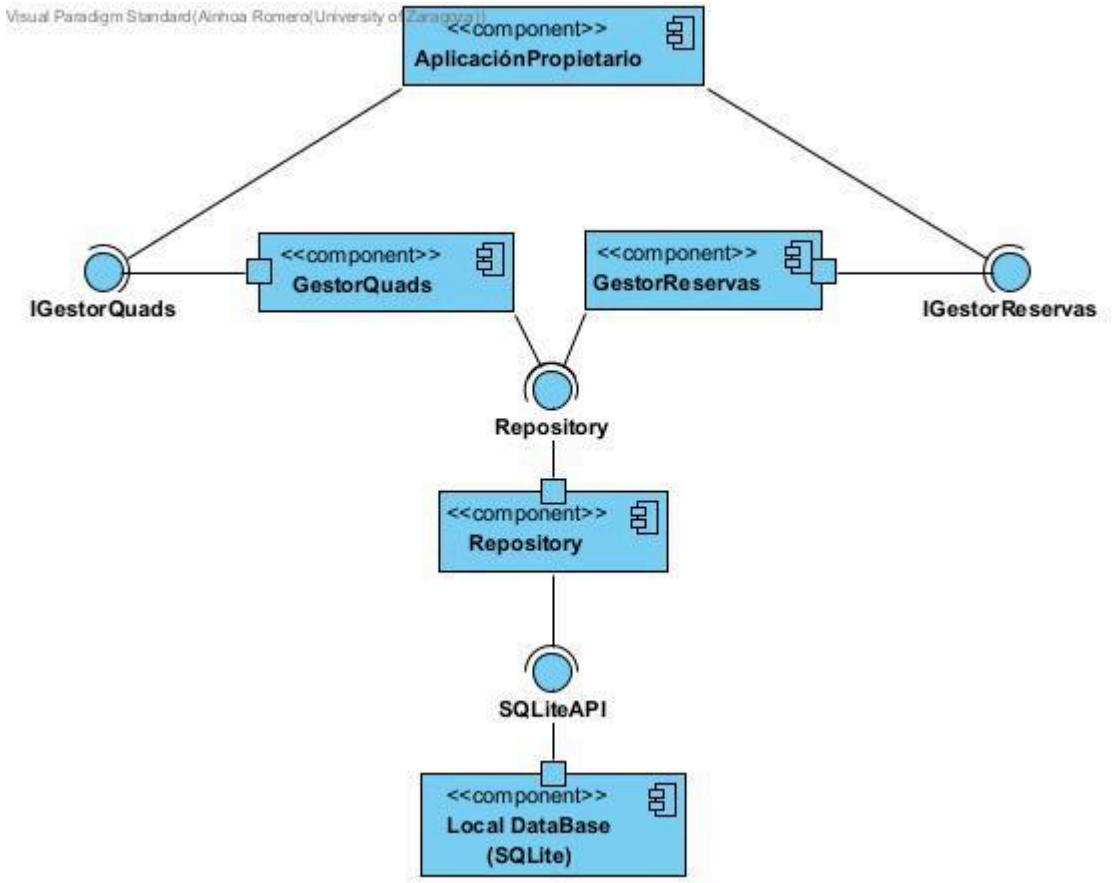
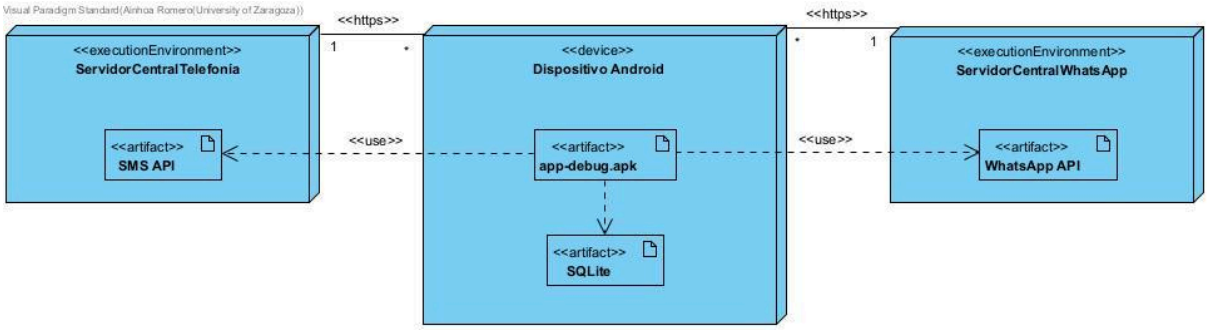


Diagrama de despliegue

En el **diagrama de despliegue** mostramos cómo nuestro sistema se ejecuta en el ***Dispositivo Android***, donde se despliegan dos artefactos: el ***app-debug.apk***, que contiene toda la aplicación (UI, gestores y repositorio), y ***SQLite***, que representa la base de datos local materializada como fichero *.db* en el propio dispositivo. Además, se reflejan dos nodos externos: ***ServidorCentralWhatsApp*** (artefacto ***WhatsApp API***) y ***ServidorCentralTelefonía*** (artefacto ***SMS API***), con caminos de comunicación HTTPS desde el dispositivo. Las dependencias *<<uses>>* desde el APK hacia cada servicio indican que, en tiempo de ejecución, la app contacta con WhatsApp o con SMS según el implementor seleccionado.



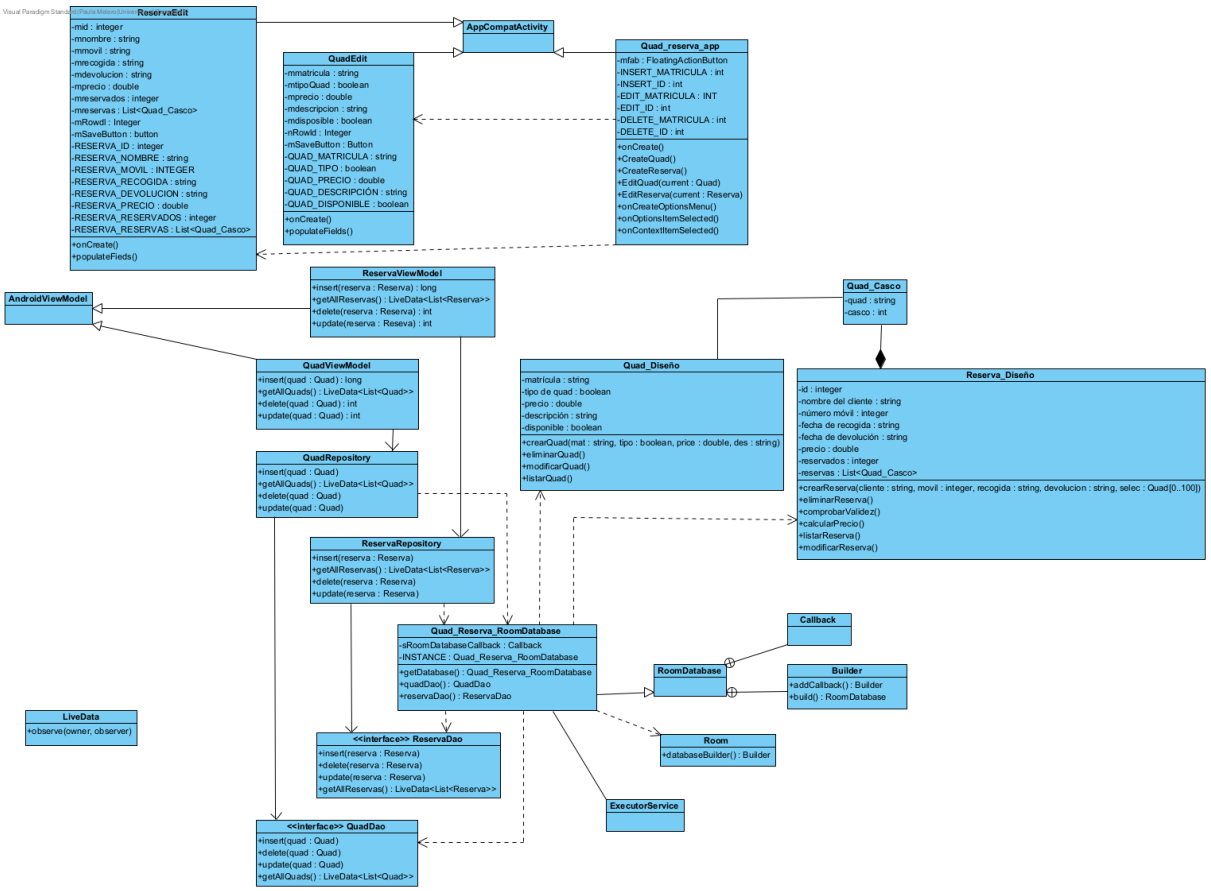
Diseño de objetos

Diagrama de clases a nivel de diseño

En este apartado vamos a describir el modelo estructural estático del sistema de Gestión de Quads. El **Diagrama de Clases Orientado a Diseño** desarrolla los conceptos definidos durante el análisis y los traslada a un nivel de detalle más cercano a la implementación. Este diseño se basa en una arquitectura en capas (MVVM / Repository / Room), lo que permite delimitar responsabilidades y definir con mayor claridad las interfaces y relaciones entre los componentes del sistema.

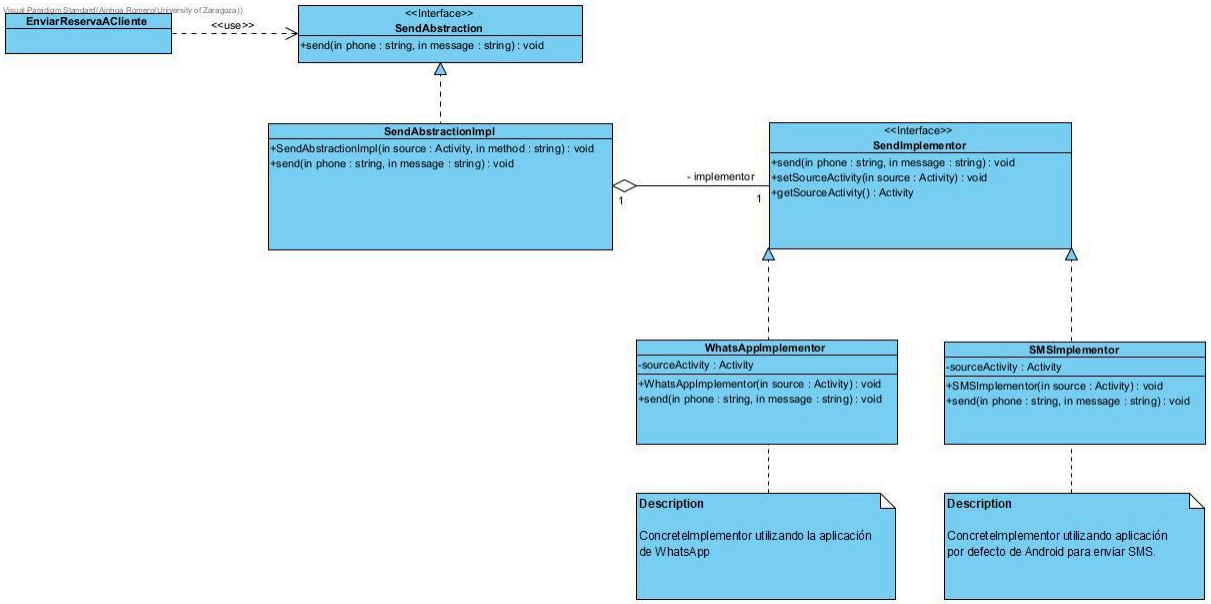
Las clases se organizan en tres capas principales:

- **Entidades de Dominio**: representan los objetos del negocio junto con sus atributos y comportamiento.
- **Capa de Presentación y Lógica**: gestiona la interfaz de usuario y la lógica de presentación, garantizando un flujo de datos coherente y reactivo.
- **Capa de Persistencia**: implementa el patrón Repository, desacoplando la lógica de acceso a datos y utilizando la base de datos local mediante Room.



Integración del patrón Bridge en el diagrama de clases

Este diagrama de clases muestra el método para enviar la reserva al cliente usando el patrón Bridge. La parte de abstracción la define la interfaz *SendAbstraction*, que declara la operación de envío, y su implementación de alto nivel *SendAbstractionImpl*, que mantiene una asociación con un *SendImplementor*. La jerarquía de implementación se modela con l ainterfaz *SendImplementor* y sus concretas *WhatsAppImplementor* y *SMSImplementor*, responsables de realizar el envío a través de la app correspondiente. El cliente (*EnviarReservaCliente*) depende únicamente de la abstracción, de modo que la elección del medio (WhatsApp o SMS) lo resolvemos por delegación en tiempo de ejecución.



Diagramas de secuencia a nivel de diseño

Para completar la visión estática del sistema, este apartado incorpora su vista dinámica mediante los **Diagramas de Secuencia a Nivel de Diseño**.

Estos diagramas muestran la interacción entre los objetos durante la ejecución de los principales flujos de los casos de uso, basándose en la estructura definida por el modelo de clases.

Diagrama de secuencia de crear quad

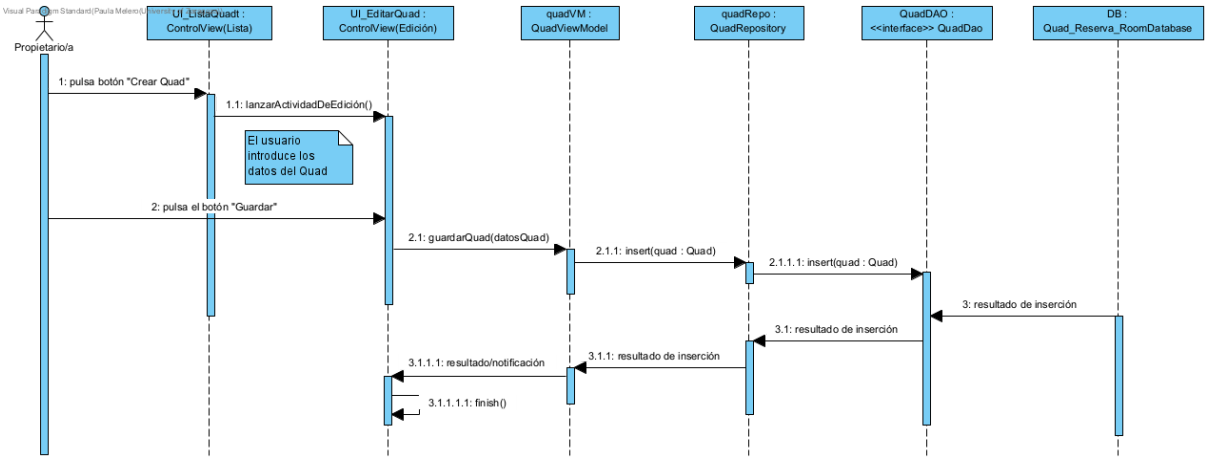


Diagrama de secuencia de eliminar quad

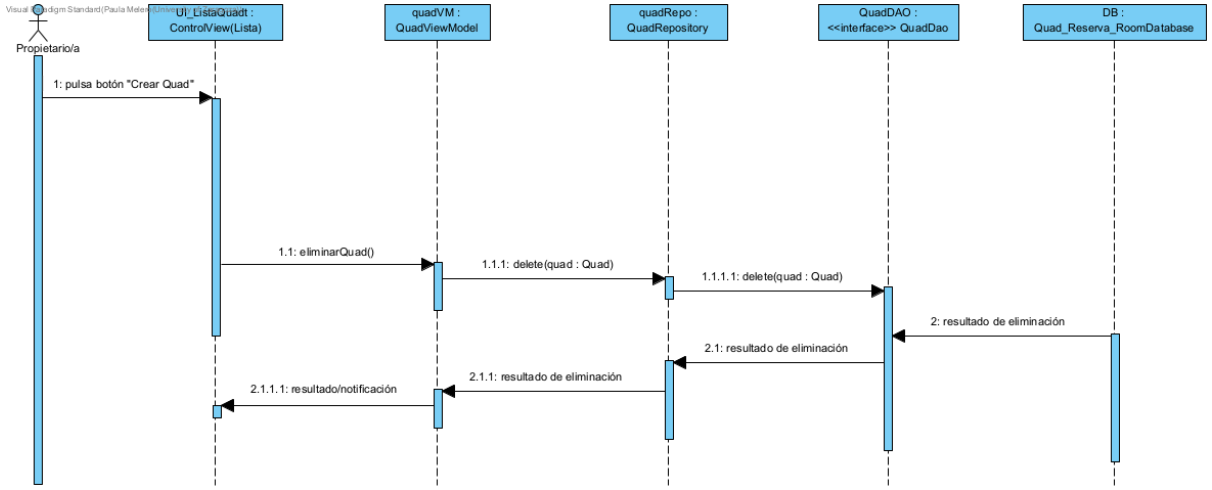


Diagrama de secuencia de listar quads

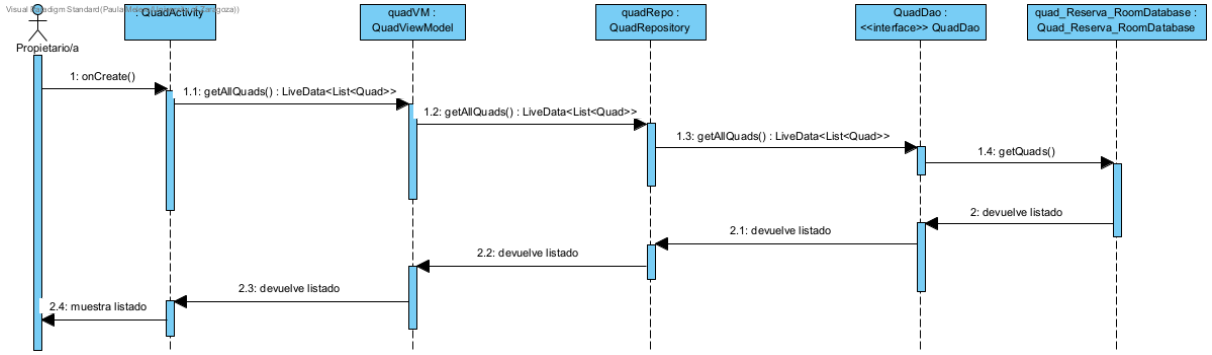


Diagrama de secuencia de crear reserva

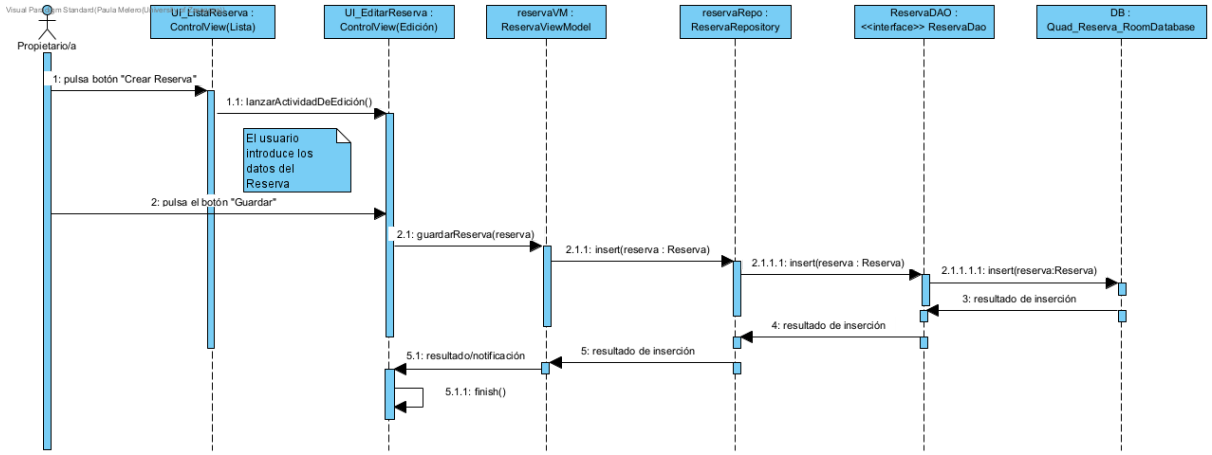


Diagrama de secuencia de eliminar reserva

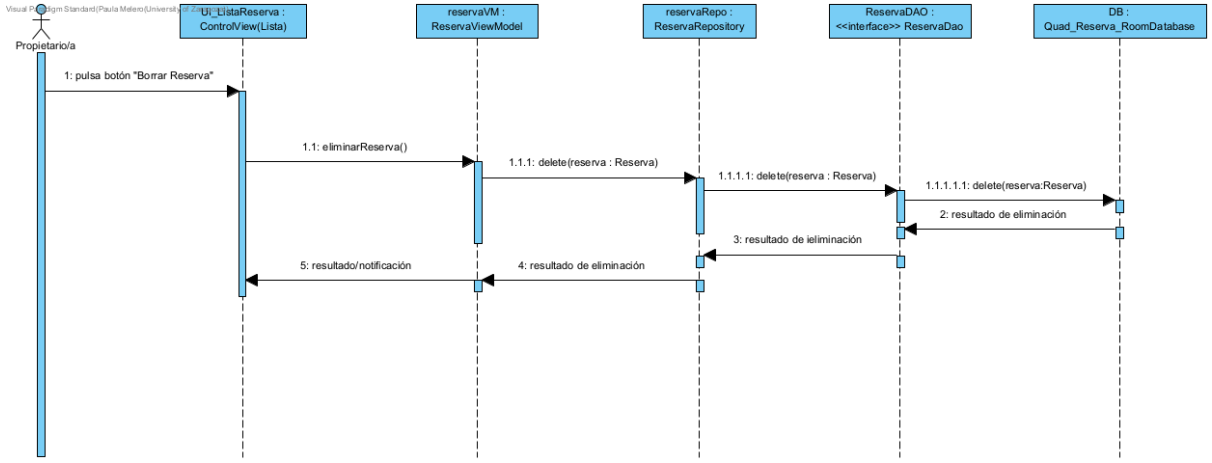


Diagrama de secuencia de listar reservas

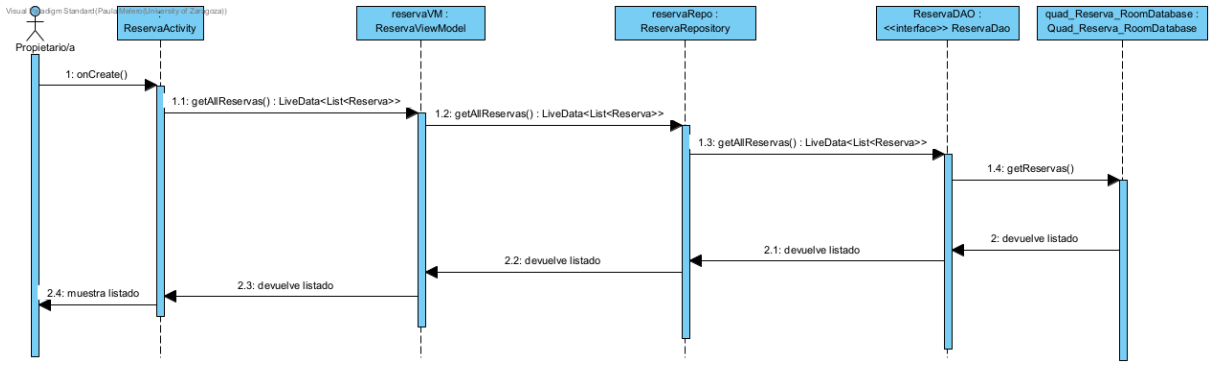
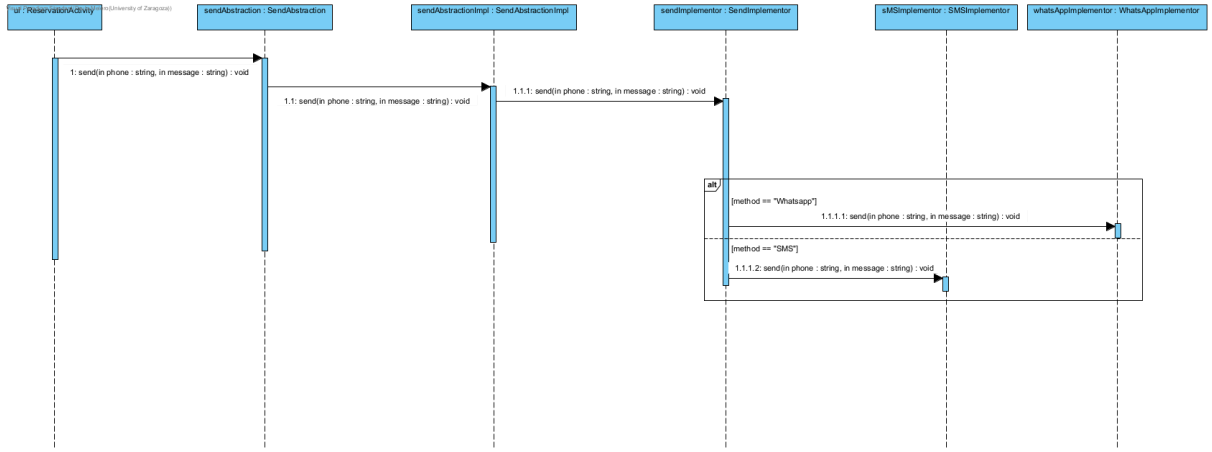
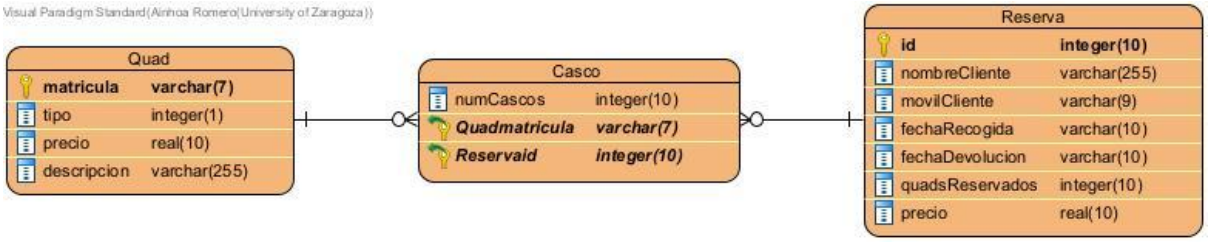


Diagrama de secuencia de enviar mensaje



Modelo lógico de la base de datos relacional

En este apartado se describe el modelo lógico de la base de datos relacional que materializa el E/R mostrado.



A partir de las entidades *Quad* y *Reserva* y de la relación N:M entre ambas (con atributo propio *numCascos*), se han obtenido tres tablas: *Quad*, *Reserva* y la tabla asociativa *Casco*. La cardinalidad N:M se resuelve con *Casco*, cuya clave primaria compuesta *(quad, reserva)* impide duplicidades para un mismo quad en una misma reserva y, a la vez, arrastra las claves foráneas hacia *Quad(matrícula)* y *Reserva(id)* para garantizar la integridad referencial.

Al inicio, se activa *PRAGMA foreign_keys = ON* para que SQLite haga cumplir esta integridad referencial en *INSERT*/*UPDATE*/*DELETE*. Así, el script de borrado elimina en orden para respetar las dependencias cuando las Foreign Keys están activas.

Una de las decisiones de diseño tomadas consiste en los identificadores. Para la tabla *Reserva* adoptamos una clave artificial *id* autoincremental como clave primaria, de modo que la identidad quede separada del contenido y los valores de ID no se reutilicen tras eliminaciones.

En las sentencias SQL, se incluyen *CHECK* para validar valores y *NOT NULL* para no permitir campos vacíos en los atributos no opcionales. Así, comprobamos estas restricciones en la propia base de datos, disminuyendo los fallos en la capa de aplicación y manteniendo la consistencia.

El script correspondiente a la creación de tablas sería el siguiente:

```
PRAGMA foreign_keys = ON;

CREATE TABLE Quad (
        matricula TEXT PRIMARY KEY,
        tipo INTEGER NOT NULL CHECK (tipo IN (0,1)),
        precio REAL NOT NULL CHECK (precio >= 0),
        descripcion TEXT NOT NULL
);

CREATE TABLE Reserva (
        id INTEGER PRIMARY KEY AUTOINCREMENT, -- clave artificial
        nombreCliente TEXT NOT NULL,
        movilCliente TEXT NOT NULL,
        fechaRecogida TEXT NOT NULL,
        fechaDevolucion TEXT NOT NULL,
        quadsReservados INTEGER NOT NULL CHECK (quadsReservados >= 0),
        precio REAL NOT NULL CHECK (precio >= 0)
);

CREATE TABLE Casco (
        numCascos INTEGER NOT NULL CHECK (numCascos >= 0),
```

```
    quad TEXT,
    reserva INTEGER,
    CONSTRAINT PK_casco PRIMARY KEY (quad, reserva),
    CONSTRAINT FK_quad FOREIGN KEY (quad) REFERENCES Quad(matricula),
    CONSTRAINT FK_reserva FOREIGN KEY (reserva) REFERENCES Reserva(id)
);
```

Y, a continuación, se muestra el script de borrado de tablas:

```
DROP TABLE Casco;
DROP TABLE Reserva;
DROP TABLE Quad;
```

Fase de implementación

Durante el desarrollo de la Práctica 4 se ha llevado a cabo la implementación completa de la aplicación Android a partir del diseño definido en la Práctica 3. Se ha seguido la estructura y las recomendaciones proporcionadas en el enunciado, tomando como referencia el proyecto Notepad y adaptando su código al dominio de gestión de quads y reservas.

El sistema se encuentra totalmente operativo y todas las funcionalidades descritas en el catálogo de requisitos funcionales han sido implementadas y verificadas. La gestión de quads, reservas y la relación muchos-a-muchos entre ambas entidades se ha resuelto correctamente.

Además, durante el proceso de implementación se han introducido una serie de mejoras respecto al diseño inicial, especialmente orientadas a la visualización y disposición de los contenidos. Estas modificaciones han permitido obtener una interfaz más clara, coherente y atractiva para el usuario, facilitando la navegación y la comprensión de la información presentada.

Asimismo, la documentación generada mediante Javadoc se encuentra actualizada y completa, reflejando fielmente el estado final del sistema.

Pruebas unitarias de caja negra

En este apartado se describen las pruebas unitarias de caja negra realizadas sobre los métodos del repositorio de la aplicación de gestión de reservas de quedas. El objetivo principal de estas pruebas es verificar el correcto funcionamiento de las operaciones de inserción, actualización y eliminación de quads y reserva, comprobando que los métodos devuelven los resultados esperados tanto para entradas válidas como para entradas no válidas.

El diseño de los casos de pruebas se ha llevado a cabo utilizando la técnica de clase de equivalencia, identificando clases de equivalencia válidas y no válidas a partir de las restricciones definidas en el catálogo de requisitos.

Las pruebas se han ejecutado de forma automática mediante código específico integrado en la aplicación, registrando los resultados obtenidos a través de la utilización del LogCat, lo que ha permitido analizar el comportamiento del sistema y detectar posibles errores en la validación de datos.

Pruebas unitarias de caja negra para métodos de la clase QuadRepository

En este subapartado se presentan específicamente las pruebas unitarias de caja negra correspondientes a los métodos de la clase QuadRepository. Partiendo del planteamiento general descrito anteriormente, aquí se evalúan de manera concreta las operaciones de creación, modificación y eliminación de quads.

Las tablas que se muestran a continuación recogen las clases de equivalencia consideradas y los casos de prueba diseñados para cada método del repositorio, permitiendo comprobar que el comportamiento del sistema es el esperado en los distintos escenarios analizados.

Tabla 1.1 : Clases de equivalencia para el método insert de la clases de QuadRepository

Parámetro	Clases válidas	Clases no válidas
matrícula	1) el formato de la matrícula consiste en 4 cifras seguidas de 3 letras 2) la matricula no existe en el sistema	3) el formato de la matrícula no es correcto 4) la matricula existe en el sistema
tipo	5) tipo = Monoplaza 6) tipo = Biplaza	7) tipo = null
descripción	8) descripcion != null	9) descripcion = null
precio	10) precio > 0	11) precio < 0

Tabla 1.2 : Casos de prueba para el método insert de la clase QuadRepository

Caso	matricula	tipo	descripcion	precio	Resultado esperado	Clases cubiertas
Casos de prueba para las clases de equivalencia válidas						
1	1234ABC	Monoplaza	“Rojo”	65.00	> 0	1, 2, 5, 8, 10
2	1235ABC	Biplaza	“Rojo”	65.00	> 0	1, 2, 6, 8, 10
Casos de prueba para las clases de equivalencia no válidas						
3	A24B	Monoplaza	“Rojo”	65.00	-1	3
4	3333ABC	Monoplaza	“Rojo”	65.00	-1	4
5	1236ABC	null	“Rojo”	65.00	-1	7
6	1237ABC	Biplaza	null	65.00	-1	9
7	1238ABC	Biplaza	“Rojo”	-5.00	-1	11

Tabla 2.1 : Clases de equivalencia para el método update de la clases de QuadRepository

Parámetro	Clases válidas	Clases no válidas
matrícula	1) la matricula debe existir en el sistema	2) la matricula no existe en el sistema
tipo	3) tipo = Monoplaza 4) tipo = Biplaza	5) tipo = null
descripción	6) descripcion != null	7) descripcion = null
precio	8) precio > 0	9) precio < 0

Tabla 2.2 : Casos de prueba para el método update de la clase QuadRepository

Caso	matricula	tipo	descripcion	precio	Resultado esperado	Clases cubiertas
Casos de prueba para las clases de equivalencia válidas						
1	1234ABC	true	“Rojo”	65.00	> 0	1, 3, 6, 8
2	1234ABC	false	“Rojo”	65.00	> 0	1, 4, 6, 8
Casos de prueba para las clases de equivalencia no válidas						
3	3333ABC	true	“No existe”	65.00	-1	2
4	4444ABC	null	“Rojo”	65.00	-1	5
5	1237ABC	false	null	65.00	-1	9
6	1238ABC	false	“Rojo”	-5.00	-1	11

Tabla 3.1 : Clases de equivalencia para el método deleteByMatricula de la clases de QuadRepository

Parámetro	Clases válidas	Clases no válidas
matrícula	1) la matricula debe existir en el sistema	2) la matricula no existe en el sistema

Tabla 3.2 : Casos de prueba para el método deleteByMatricula de la clase QuadRepository

Caso	matricula	Clases cubiertas
Casos de prueba para las clases de equivalencia válidas		
1	1234ABC	1
Casos de prueba para las clases de equivalencia no válidas		
2	3333ABC	2

Pruebas unitarias de caja negra para métodos de la clase ReservaRepository

En este apartado se presentan las pruebas unitarias de caja negra realizadas sobre los métodos de la clase ReservaRepository. Su finalidad es comprobar que las operaciones sobre reservas producen los resultados esperados en distintas situaciones. Se muestran las clases de equivalencia consideradas y los casos de prueba diseñados para cada método, permitiendo verificar que el repositorio responde correctamente ante diferentes escenarios de uso.

Tabla 4.1 : Clases de equivalencia para el método insert de la clases de ReservaRepository

Parámetro	Clases válidas	Clases no válidas
nombreCliente	1) nombreCliente != null	2) nombreCliente = null
movilCliente	3) movilCliente != null 4) movilCliente se compone unicamente de carácteres numéricos	5) movilCliente = null 6) movilCleinte contiene cualquier tipo de carácter
fechaRecogida	7) cumple el formato dd/mm/aaaa 8) es una fecha válida 9) es igual o anterior a la fechaDevolucion	10) no cumple el formato dd/mm/aaaa 11) no es una fecha válida 12) es posterior a la fechaDevolucion
fechaDevolucion	13) cumple el formato dd/mm/aaaa 14) es una fecha válida 15) es igual o posterior a la fechaRecogida	16) no cumple el formato dd/mm/aaaa 17) no es una fecha válida 18) es anterior a la fechaRecogida

Tabla 4.2 : Casos de prueba para el método insert de la clase ReservaRepository

Caso	nombreCliente	movilCliente	fechaRecogida	fechaDevolucion	Resultado esperado	Clases cubiertas
Casos de prueba para las clases de equivalencia válidas						
1	CL_001	612458920	01/01/2026	03/01/2026	> 0	1, 3, 4, 7, 8, 9, 13, 14, 15
Casos de prueba para las clases de equivalencia no válidas						
3	“”	234678598	10/01/2026	13/01/2026	-1	2
4	CL_002	“”	10/01/2026	13/01/2026	-1	5
5	CL_003	a5463b	10/01/2026	13/01/2026	-1	6
6	CL_004	234678598	2026/01/01	13/01/2026	-1	10
7	CL_005	234678598	10/00/2026	13/01/2026	-1	11
8	CL_006	234678598	10/01/2026	13/01/2026	-1	12
9	CL_007	234678598	10/00/2026	2026/01/01	-1	16
10	CL_008	234678598	10/00/2026	13/00/2026	-1	17
11	CL_009	234678598	13/01/2026	10/01/2026	-1	18

Tabla 5.1 : Clases de equivalencia para el método delete de la clases de ReservaRepository

Parámetro	Clases válidas	Clases no válidas
id	1) id existe en la base de datos	2) id no existe en la base de datos

Tabla 5.2 : Casos de prueba para el método delete de la clase ReservaRepository

Caso	id	Clases cubiertas
Casos de prueba para las clases de equivalencia válidas		
1	1	1
Casos de prueba para las clases de equivalencia no válidas		
2	1	2

(El caso 2 se ejecuta posteriormente al caso 1 de forma que la reserva ya no existe en la base de datos)

No se han incluido tablas específicas de clases de equivalencia y de casos de pruebas para el método update de ReservaRepository, ya que las clases de equivalencia consideradas son exactamente las mismas que en el método insert.

Esto se debe a que ambos métodos comparten las mismas restricciones sobre los atributos de la reserva (nombre del cliente, número de teléfono, fechas y formato de los datos), ya que en ambos casos el sistema debe garantizar que una reserva almacenada en la base de datos siempre sea válida. El método update no introduce nuevas condiciones ni relaja las existentes, sino que aplica las mismas reglas de validación que en la inserción inicial.

Por tanto, las clases de equivalencia válidas y no válidas asociadas al método insert cubren completamente los posibles escenarios del método update, lo que hace redundante la construcción de una nueva tabla específica.

La inserción de los quads vinculados con sus respectivos cascos no se ha tenido en cuenta en los casos de prueba ya que no es posible reproducir ningún caso de error. La app gestiona que los quads estén disponibles y que el número de cascos por quad sea correcto, es decir, que dicho número se encuentre en el intervalo adecuado (Monoplaza -> [0 - 1] y Biplaza -> [0 - 2])

Pruebas del sistema

Prueba de volumen

En este apartado se presentan las pruebas de volumen realizadas sobre los repositorios ReservaRepository y QuadRepository. Los casos de prueba se han diseñado utilizando la técnica de análisis de valores límite, considerando los requisitos del sistema que establecen que no se espera más de 100 quads en la aplicación ni más de 20.000 reservas almacenadas. Se evalúa el comportamiento del sistema tanto para valores dentro del rango permitido como para valores que superan dichos límites, con el objetivo de comprobar que las operaciones se ejecutan correctamente y que el sistema mantiene un rendimiento estable.

Tabla 6: Valores límite para una prueba de volumen en la aplicación de gestión de reservas de quads.

Criterio	Rango de valores	Valores límite
Número de reservas	0 - 20.000	-1, 0, 20.000, 20.001
Número de quads	0 - 100	-1, 0, 100, 101

Los resultados obtenidos de estas pruebas se recogen en las tablas 6.1 y 6.2, donde se detallan los valores utilizados y el comportamiento observado para cada caso. Esto permite verificar de manera objetiva cómo responde el sistema ante cargas elevadas y aproximarse a los límites de operación de los repositorios.

Tabla 6.1: Caso de prueba para una prueba de volumen para la gestión de reservas.

Caso	Número de reservas	Resultado esperado
1	-1	Caso de pruebas imposible ya que no se puede provocar un número de reservas negativo.
2	0	La aplicación funciona correctamente.
3	20.000	La aplicación funciona correctamente.
4	20.001	La aplicación indica que no es posible la última inserción solicitada.

Tabla 6.2: Caso de prueba para una prueba de volumen para la gestión de quads.

Caso	Número de quads	Resultado esperado
1	-1	Caso de pruebas imposible ya que no se puede provocar un número de quads negativo.
2	0	La aplicación funciona correctamente.
3	100	La aplicación funciona correctamente.
4	101	La aplicación funciona correctamente.

Pruebas de sobrecarga

El objetivo de esta última prueba es determinar la longitud máxima de texto que el sistema es capaz de soportar para el atributo de descripción de un quad. Para ello, se han empleado una serie de valores de prueba con longitudes crecientes, con el fin de estimar el umbral a partir del cual la inserción de un queda deja de realizarse correctamente o el sistema comienza a mostrar problemas de funcionamiento.

Cabe destacar que, aunque la clase String en Java soporta teóricamente longitudes de hasta 2^31 -1 caracteres, en un entorno real como aplicación Android este no suele alcanzarse. Por tanto, en caso de producirse un fallo durante la ejecución de la prueba, este no estaría necesariamente relacionado con el tipo de dato utilizado, sino con otras limitaciones del sistema, como la gestión de memoria, el funcionamiento de la base de datos y el rendimiento general de la aplicación. Los resultados se recogen a continuación.

Tabla 7: Caso de pruebas para una prueba de sobrecarga sobre el atributo “descripcion” de la clase Quad

Caso	length(descripción)	Resultado obtenido
1	10	La aplicación funciona correctamente.
2	50	La aplicación funciona correctamente.
3	100	La aplicación funciona correctamente.
4	500	La aplicación funciona correctamente.
5	1000	La aplicación funciona correctamente.
6	500	La aplicación funciona correctamente.

Bibliografía

Miro. (s.f.). *Diagrama de objetos UML: perspectivas y técnicas*. Recuperado de https://miro.com/es/diagrama/que-es-diagrama-objetos-uml/ (26/09/2025)

Visual Paradigm. (s.f.). *Todo lo que necesitas saber sobre los diagramas de secuencia*. Recuperado de https://blog.visual-paradigm.com/es/everything-you-need-to-know-about-sequence-diagrams/ (24/09/2025)