Ingeniería del Software

CURSO 2025-2026

Memoria

Sistema de Gestión de Reservas de Quads

T234-quads

Año 3 - Cuatrimestre 1
2025/2026

Índice

1 Introducción y objetivos

El principal objetivo es realizar el análisis inicial de un sistema de gestión de reservas de quads, que sirva como base para fases posteriores de diseño, implementación y pruebas.

En esta práctica se aborda el proceso de especificación de requisitos, tanto funcionales como no funcionales, así como la definición de las restricciones del sistema. A partir de ellos, se han elaborado los casos de uso y sus flujos de eventos, representados mediante diagramas UML (casos de uso, clases y secuencia) que describen la estructura y el comportamiento dinámico del sistema.

Los objetivos principales de este trabajo son:

- Identificar y priorizar los requisitos del sistema (con MoSCoW).
- Establecer las restricciones que limitan el funcionamiento del sistema.
- Modelar los casos de uso y detallar sus flujos principales y alternativos.
- Representar gráficamente la estructura estática del sistema mediante un diagrama de clases.
- Describir el comportamiento dinámico del sistema mediante diagramas de secuencia.
- Sentar las bases para la posterior fase de diseño e implementación de la aplicación de gestión de reservas de quads.

2 Requisitos

2.1 Requisitos Funcionales

Nº Requisito	**Requisito funcional**	**MoSCoW**
R.F. 1	El Sistema debe permitir la adición de un nuevo quad, almacenando su matrícula, tipo de quad, precio en euros del alquiler por día y descripción.	Must
R.F. 2	El Sistema debe permitir la consulta de un listado de quads previamente creados, pudiéndose ordenar por matrícula, tipo o precio.	Must
R.F. 3	El Sistema debe permitir la modificación de los quads previamente creados, así como la actualización del precio de las reservas a las que esté asociado dicho quad.	Must
R.F. 4	El Sistema debe permitir la eliminación de los quads, así como la actualización del precio de las reservas a las que esté asociado dicho quad previamente creados.	Must

Nº Requisito	**Requisito funcional**	**MoSCoW**
R.F. 5	El Sistema debe permitir la creación de una reserva, indicando un nombre de cliente, el número móvil del cliente, la fecha de recogida y devolución, y una selección de quads, indicando también si se va a necesitar casco para su conducción.	Must
R.F. 6	El Sistema debería permitir la consulta de un listado de las reservas previamente creadas, pudiéndose ordenar por nombre de cliente, número de móvil, fecha de recogida, o fecha de devolución.	Must
R.F. 7	El Sistema debe permitir la modificación de las reservas previamente creadas.	Must
R.F. 8	El Sistema debe permitir la eliminación de las reservas previamente creadas.	Must
R.F. 9	El Sistema debe permitir calcular automáticamente el precio total de una reserva.	Must
R.F. 10	El Sistema debe permitir el envío al móvil del cliente de la información de la reserva (incluido precio).	Must

2.2 Requisitos No Funcionales

Nº Requisito	**Requisito no funcional**	**MoSCoW**
R.N.F. 1	El Sistema no permitirá la creación de un nuevo quad si hay 100 quads existentes.	
R.N.F. 2	El Sistema no permitirá la creación de una nueva reserva si hay 20000 reservas existentes.	

3 Restricciones

La aplicación debe estar disponible para los dispositivos con un sistema operativo Android.

4 Análisis

4.1 Diagrama de casos de uso

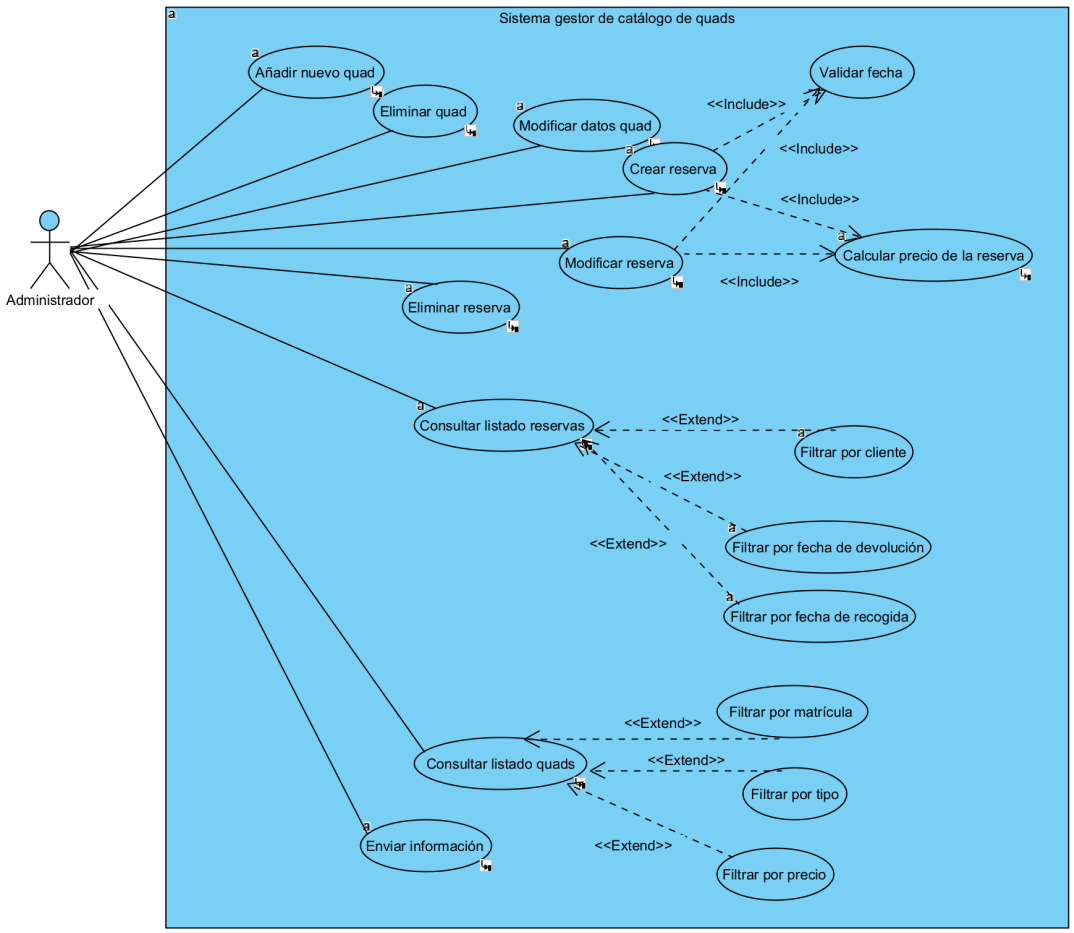

Figura 1: Diagrama de casos de uso completo

4.1.1. Añadir nuevo quad

Flujo principal:

1. El caso de uso comienza cuando el administrador selecciona la opción “Añadir quad” desde la pantalla principal.
2. El sistema muestra un formulario solicitando los datos del quad: matrícula, tipo, precio por día y descripción.
3. El administrador rellena los campos y pulsa el botón “CONFIRMAR”.
4. El sistema valida los datos, guarda el quad en la base de datos y muestra un mensaje de confirmación.
5. El caso de uso termina volviendo a la pantalla principal.

Flujo alternativo:

- En el paso 3, el administrador pulsa "Aceptar" sin rellenar todos los campos requeridos o con datos inválidos, como una matrícula duplicada.
- El sistema muestra un mensaje de error indicando los campos a corregir.
- El caso de uso continúa en el paso 2 del flujo principal.

4.1.2. Consultar listado de quads

Flujo principal:

1. El administrador selecciona la opción "Consultar listado de quads".
2. El sistema muestra un listado de los quads disponibles.
3. *extend* Ordenar listado de quads por matrícula, tipo o precio.
4. El caso de uso termina cuando el administrador regresa a la pantalla principal.

Flujo alternativo:

- En el paso 2, si no existen quads registrados, el sistema muestra un mensaje de advertencia "No hay quads registrados".
- El caso de uso finaliza al cerrar el aviso.

4.1.3. Modificar datos quad

Flujo principal:

1. El caso de uso parte del flujo de "Consultar listado de quads".
2. El administrador selecciona un quad del listado y pulsa "Modificar".
3. El sistema muestra un formulario con los datos actuales del quad.
4. El administrador modifica los campos deseados y pulsa "Aplicar cambios".
5. El sistema actualiza los datos y muestra un mensaje de confirmación.
6. El listado se actualiza con la información modificada.

Flujo alternativo:

- En el paso 4, el administrador deja algún campo requerido vacío o introduce un valor inválido.
- El sistema muestra los campos en rojo y un mensaje indicando el error.
- El caso de uso continúa en el paso 3.

4.1.4. Eliminar quad

Flujo principal:

1. El caso de uso parte de "Consultar listado de quads".
2. El administrador selecciona un quad y pulsa la opción "Eliminar".
3. El sistema muestra un mensaje de confirmación.
4. El administrador pulsa "Aceptar" y el quad se elimina del sistema.
5. El caso de uso termina actualizando el listado.

Flujo alternativo:

- En el paso 4, el administrador pulsa "Cancelar".
- El quad no se elimina y el caso de uso termina sin cambios.

4.1.5. Crear reserva

Flujo principal:

1. El administrador selecciona la opción "Crear reserva".
2. El sistema muestra un formulario solicitando nombre del cliente, número de móvil, fechas de recogida y devolución, selección de quad y cascos necesarios.
3. El administrador rellena los datos y pulsa "Aceptar".
4. *include* Validar fechas de la reserva.
5. *include* Calcular precio total automáticamente.
6. El sistema guarda la reserva y muestra un mensaje de confirmación.

Flujo alternativo:

- En el paso 3, algún campo requerido no está relleno y el sistema muestra un mensaje de error.
- En el paso 4, las fechas no son válidas, por ejemplo devolución antes que recogida, y el sistema muestra error y devuelve al paso 2.

4.1.6. Consultar listado de reservas

Flujo principal:

1. El administrador selecciona la opción "Consultar reservas".
2. El sistema muestra un listado de las reservas existentes.
3. *extend* Ordenar listado de reservas por cliente, móvil, fecha recogida o devolución.
4. El caso de uso termina cuando el administrador vuelve a la pantalla principal.

Flujo alternativo:

- En el paso 2, si no existen reservas registradas, el sistema muestra un mensaje de advertencia "No hay reservas registradas".
- El caso de uso finaliza al cerrar el aviso.

4.1.7. Modificar reserva

Flujo principal:

1. El caso de uso parte de "Consultar listado de reservas".
2. El administrador selecciona una reserva y pulsa "Modificar".
3. El sistema muestra un formulario con los datos actuales de la reserva.
4. El administrador edita los campos y pulsa "Aplicar cambios".
5. *include* Validar fechas de la reserva.
6. *include* Recalcular precio total.
7. El sistema actualiza los datos y muestra un mensaje de confirmación.

Flujo alternativo:

- En el paso 4, algún campo requerido queda vacío o inválido y el sistema marca los errores y vuelve al paso 3.
- En el paso 5, las fechas no son válidas y el sistema muestra error y se vuelve al paso 3.

4.1.8. Eliminar reserva

Flujo principal:

1. El caso de uso parte de "Consultar listado de reservas".
2. El administrador selecciona una reserva y pulsa "Eliminar".
3. El sistema solicita confirmación.
4. El administrador pulsa "Aceptar" y la reserva se elimina.
5. El listado se actualiza y el caso de uso termina.

Flujo alternativo:

- En el paso 4, el administrador pulsa "Cancelar".
- La reserva no se elimina y el caso de uso termina sin cambios.

4.1.9. Calcular precio de la reserva

Flujo principal:

1. El caso de uso comienza cuando el sistema recibe los datos de una reserva (fechas y quad seleccionado).
2. El sistema calcula la duración de la reserva en días.
3. El sistema multiplica el número de días por el precio por día del quad seleccionado.
4. Si se han solicitado cascos, añade el suplemento correspondiente.
5. El sistema guarda el precio total en la reserva y lo muestra en pantalla.
6. El caso de uso termina devolviendo el control al caso de uso padre.

Flujo alternativo:

- En el paso 2, si las fechas no son válidas (ej. devolución anterior a recogida), el sistema no puede calcular el precio y muestra un mensaje de error.
- El caso de uso retorna al paso 2 del flujo principal de "Crear reserva" o "Modificar reserva".

Nota: Si el administrador modifica el precio diario de un quad, las reservas ya existentes no se ven afectadas, conservando el precio total calculado en el momento de su creación. Solo las nuevas reservas usarán el precio actualizado.

4.1.10. Enviar información de la reserva al cliente

Flujo principal:

1. El caso de uso comienza cuando al menos una reserva ha sido validada y guardada en el sistema.

   Nota: Una reserva se considera válida cuando:

   - Las fechas de recogida y devolución son coherentes (la devolución es posterior a la recogida).
   - El quad seleccionado está disponible en ese rango de fechas.
   - Se ha proporcionado un nombre y número de teléfono válidos del cliente.
   - Se ha calculado correctamente el precio total asociado a la reserva.

2. El administrador selecciona una reserva y le da al botón de "Enviar Información".
3. El sistema prepara un mensaje con los datos de la reserva: nombre del cliente, fechas, quad seleccionado, cascos y precio total.
4. El sistema envía el mensaje al número de móvil proporcionado por el cliente.
5. El caso de uso termina devolviendo el control al caso de uso padre.

Flujo alternativo:

- En el paso 3, si no se puede enviar el mensaje (ej. número inválido), el sistema muestra un aviso al administrador.
- El caso de uso termina sin enviar la notificación, pero la reserva queda registrada.

4.2 Diagrama de clases

El diagrama de clases presentado a continuación representa el modelo estático del sistema de gestión de alquiler de quads a nivel de análisis. Este modelo captura las entidades fundamentales del dominio del problema, sus atributos, operaciones y relaciones, sin considerar aún aspectos de implementación o interfaz de usuario.

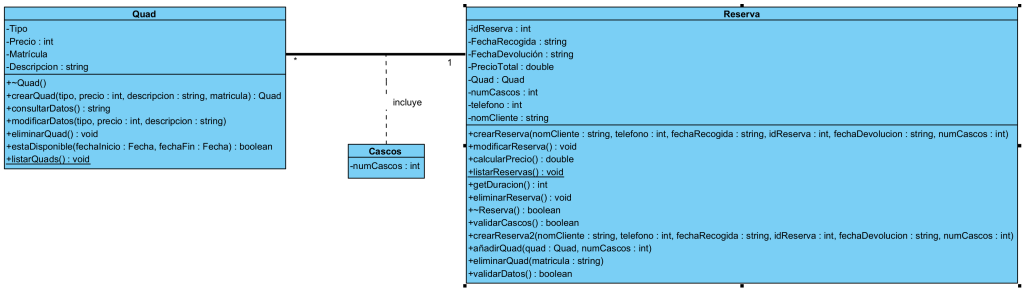

Figura 2: Diagrama de clases - Análisis

4.2.1. Clase Quad

Representa los vehículos disponibles para alquiler en el sistema.
Atributos:

- **Tipo:** Clasificación del quad (monoplaza o biplaza)
- **Precio (double):** Coste de alquiler por día en euros
- **Matrícula (string):** Identificador único del vehículo (formato: 4 dígitos + 3 letras)
- **Descripción (string):** Información adicional sobre marca, modelo, color, etc.

Operaciones:

- `Quad(matricula: String, tipo: String, precioDia: double, descripcion: String)`: Constructor para crear nuevos quads.
- `~Quad()`: Destructor de la clase.
- `consultarDatos(): string` Devuelve la información completa del quad.
- `modificarDatos(tipo, precio: int, descripción: string): void` Permite actualizar los datos del quad.
- `eliminarQuad(): void` Elimina el quad del sistema.
- `estaDisponible(fechaInicio, fechaFin): boolean` Verifica si el quad está disponible en un rango de fechas.
- `listarQuads(): void` Operación de clase para obtener todos los quads del sistema.

4.2.2. Clase Reserva

Representa las reservas realizadas por los clientes.
Atributos:

- **idReserva (int):** Identificador único de la reserva
- **nomCliente (string):** Nombre del cliente
- **telefono (int):** Número de teléfono del cliente
- **FechaRecogida (string):** Fecha de inicio del alquiler
- **FechaDevolución (string):** Fecha de fin del alquiler
- **PrecioTotal (double):** Coste total de la reserva

Operaciones:

- `crearReserva(idReserva: int, nomCliente: String, telefono: int, fechaRecogida: String, fechaDevolucion: String, quad: Quad, numCascos: int)`: Constructor para crear nuevas reservas.
- `~Reserva()`: Destructor de reserva.
- `getDuration(): int` Calcula la duración de la reserva en días.
- `calcularPrecio(): double` Determina el precio total basado en la duración y el quad.
- `validarFechas(): boolean` Verifica que las fechas sean coherentes.
- `validarCascos(): boolean` Comprueba que el número de cascos sea adecuado para el tipo de quad.
- `listarReservas(): void` Operación de clase para obtener todas las reservas.
- `enviarInformacion(): void` Envía al cliente los datos de la reserva (nombre, fechas, quad, cascos y precio total) al número de teléfono registrado.

4.2.3. Entidad Cascos

Aunque en el diagrama inicial aparece como clase separada, se considera que `numCascos` debería ser un atributo de la relación `Quad_Reserva`, ya que los cascos no tienen existencia independiente en el sistema.

4.2.4. Relaciones entre clases

Reserva & Quad: Relación de asociación donde cada reserva hace referencia a los quad alquilados.
Multiplicidad: Una reserva está asociada a uno o varios quads y un quad puede tener de cero a múltiples reservas (N:M).

4.3 Diagramas de secuencia

4.3.1. Añadir nuevo quad

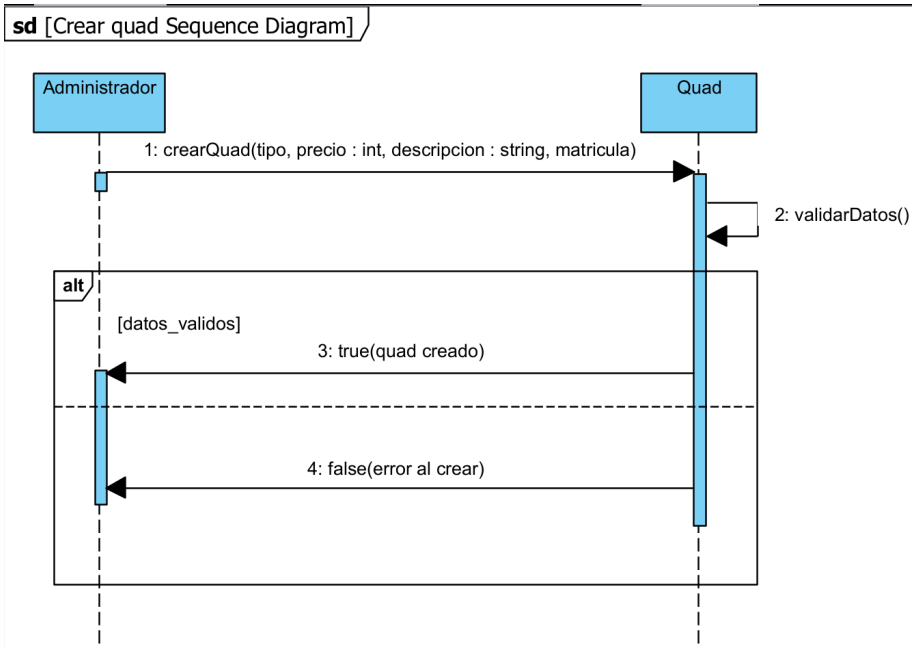

Figura 3: Diagrama de secuencia - Añadir nuevo quad

4.3.2. Consultar listado de quads

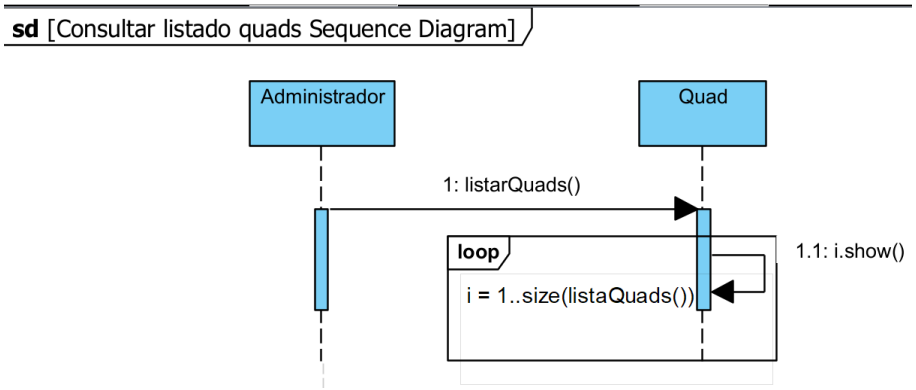

Figura 4: Diagrama de secuencia - Consultar listado de quads

4.3.3. Modificar quad

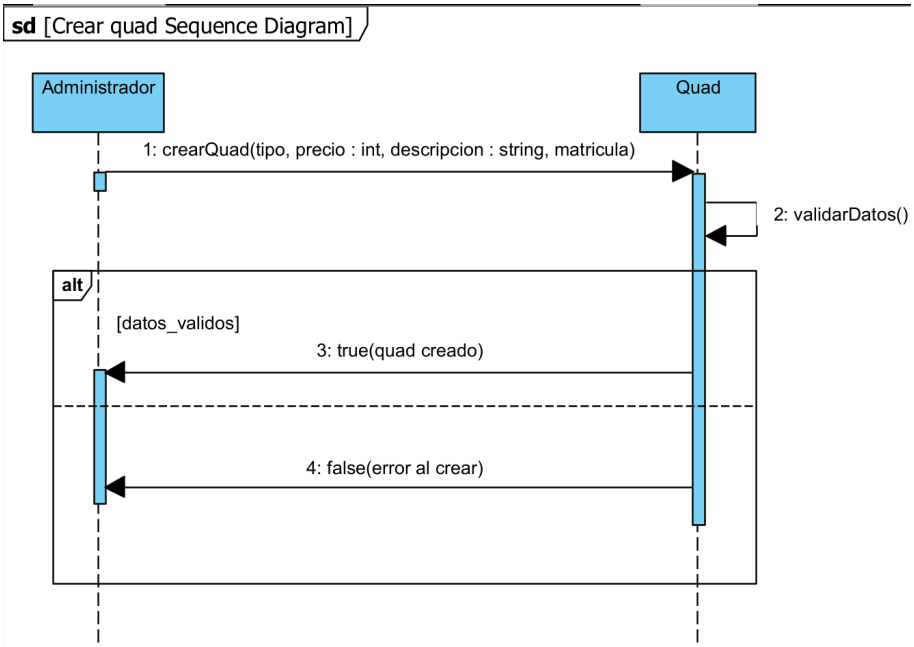

Figura 5: Diagrama de secuencia - Modificar quad

4.3.4. Eliminar quad

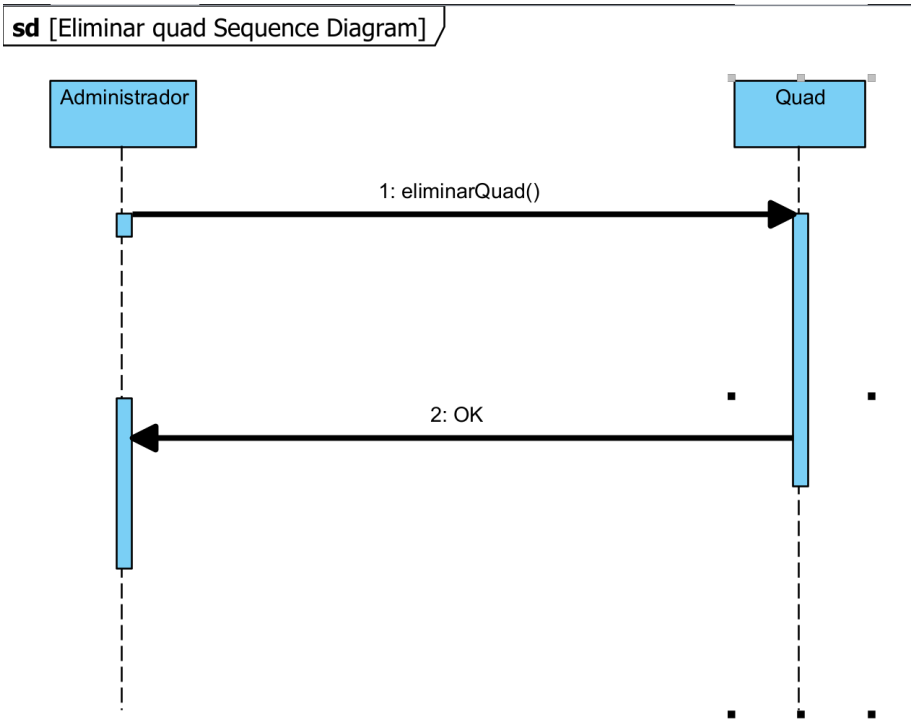

Figura 6: Diagrama de secuencia - Eliminar quad

4.3.5. Crear reserva

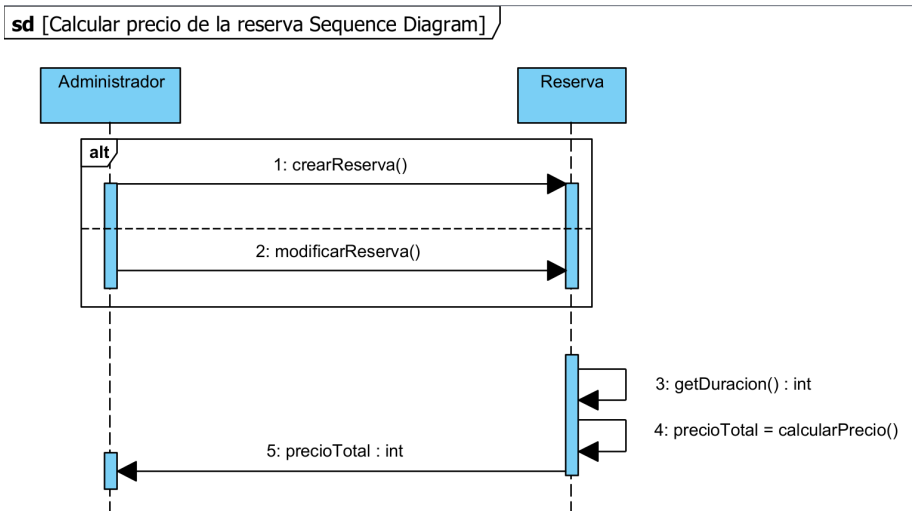

Figura 7: Diagrama de secuencia - Crear reserva

4.3.6. Consultar listado de reservas

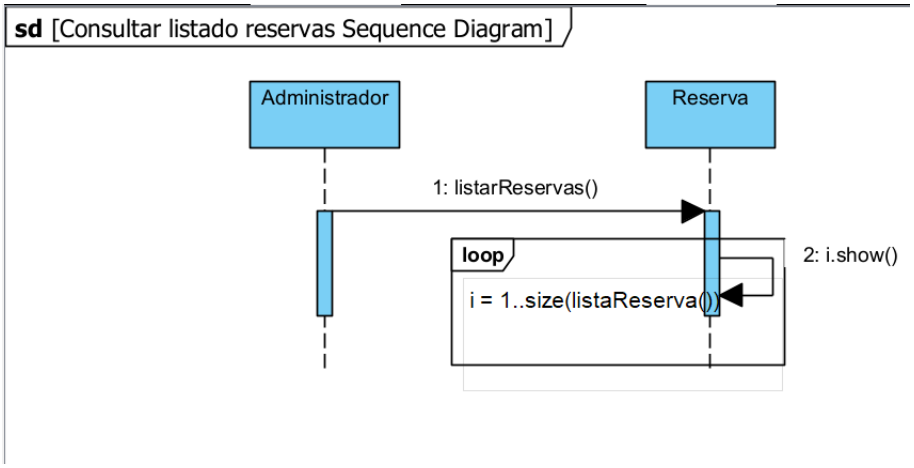

Figura 8: Diagrama de secuencia - Consultar listado de reservas

4.3.7. Modificar reserva

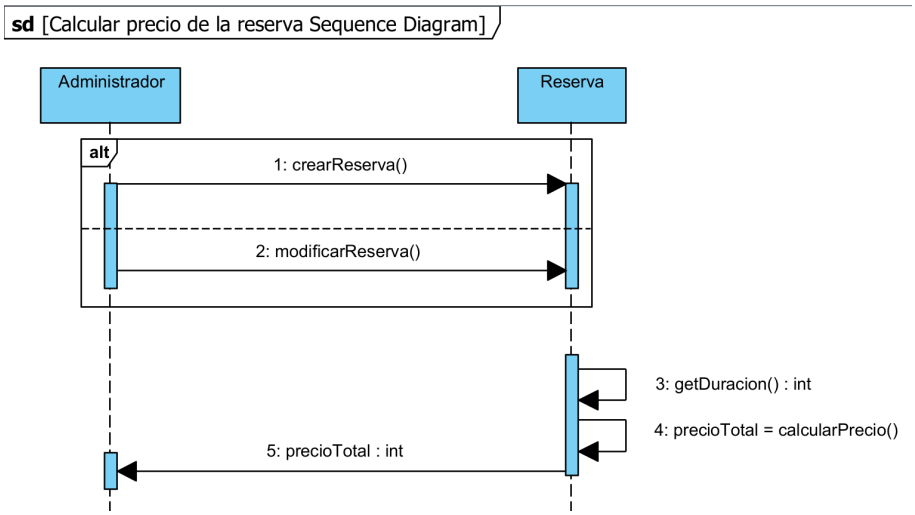

Figura 9: Diagrama de secuencia - Modificar reserva

4.3.8. Eliminar reserva

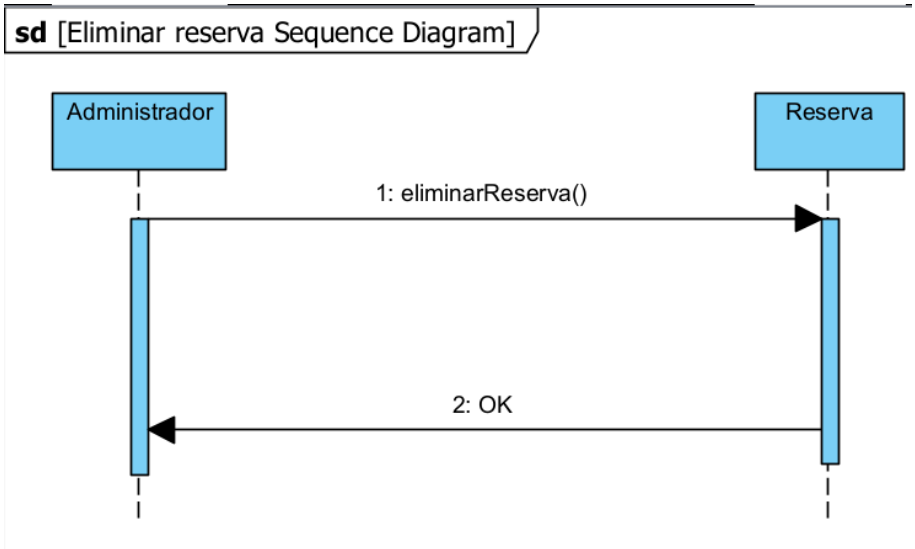

Figura 10: Diagrama de secuencia - Eliminar reserva

4.4 Prototipos de pantalla

El prototipado de la interfaz para el usuario se ha hecho a partir de Visual Paradigm, usando la herramienta Android Phone Wireframe. El diseño de la interfaz sigue estrictamente las especificaciones definidas en los requisitos funcionales definidos anteriormente, pudiéndose realizar todos los casos de uso también definidos.

Las pantallas se muestran a continuación:

Figura 11: Pantalla principal con opciones Quads y Reservas

Figura 12: Pantalla de listado de quads

Figura 13: Pantalla de añadir/modificar quad

Figura 14: Pantalla de detalles de quad

Figura 15: Pantalla de listado de reservas

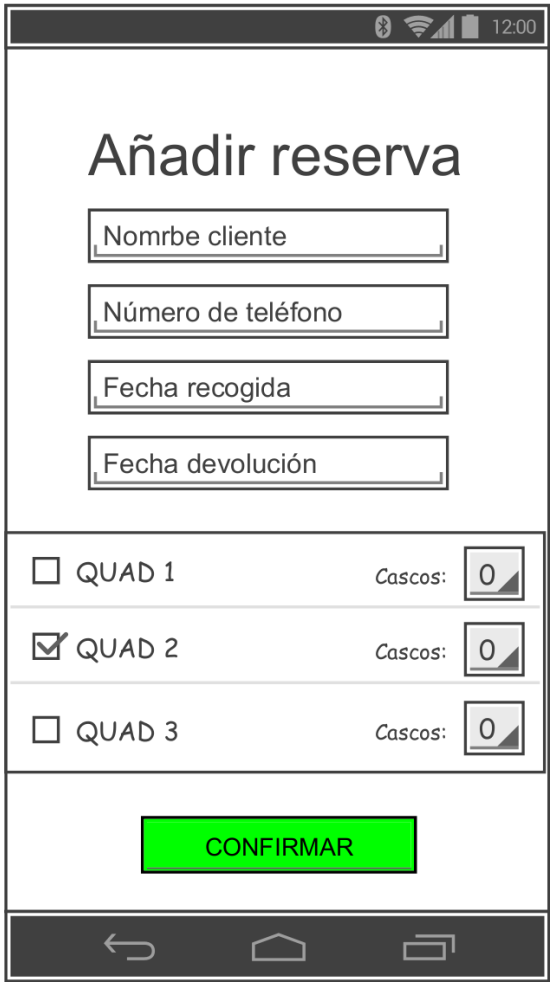

Figura 16: Pantalla de crear/modificar reserva

Figura 17: Pantalla de detalles de reserva

La interfaz del sistema se ha diseñado para ser sencilla, intuitiva y funcional. Desde la pantalla de inicio, el administrador puede acceder a los apartados “Quads” y “Reservas”. En el apartado Quads, el sistema permite añadir un nuevo quad o consultar el listado de quads registrados. En dicho listado, los quads pueden filtrarse y ordenarse por matrícula, tipo o precio. Al seleccionar un quad concreto, se muestra una pantalla con sus datos, desde la cual es posible modificar su información o eliminarlo.

El apartado Reservas funciona de forma análoga: se pueden crear nuevas reservas, consultar el listado existente, modificar o eliminar una reserva. Además, desde la pantalla de detalle de cada reserva, el sistema ofrece la opción de enviar la información de la reserva al cliente.

No se incluyen pantallas de retroalimentación así como manejo de errores o feedback para éxito por simplicidad en la fase de desarrollo de diseño. Estas se incluirán en la fase del desarrollo del prototipo funcional.

4.5 Mapa de navegación

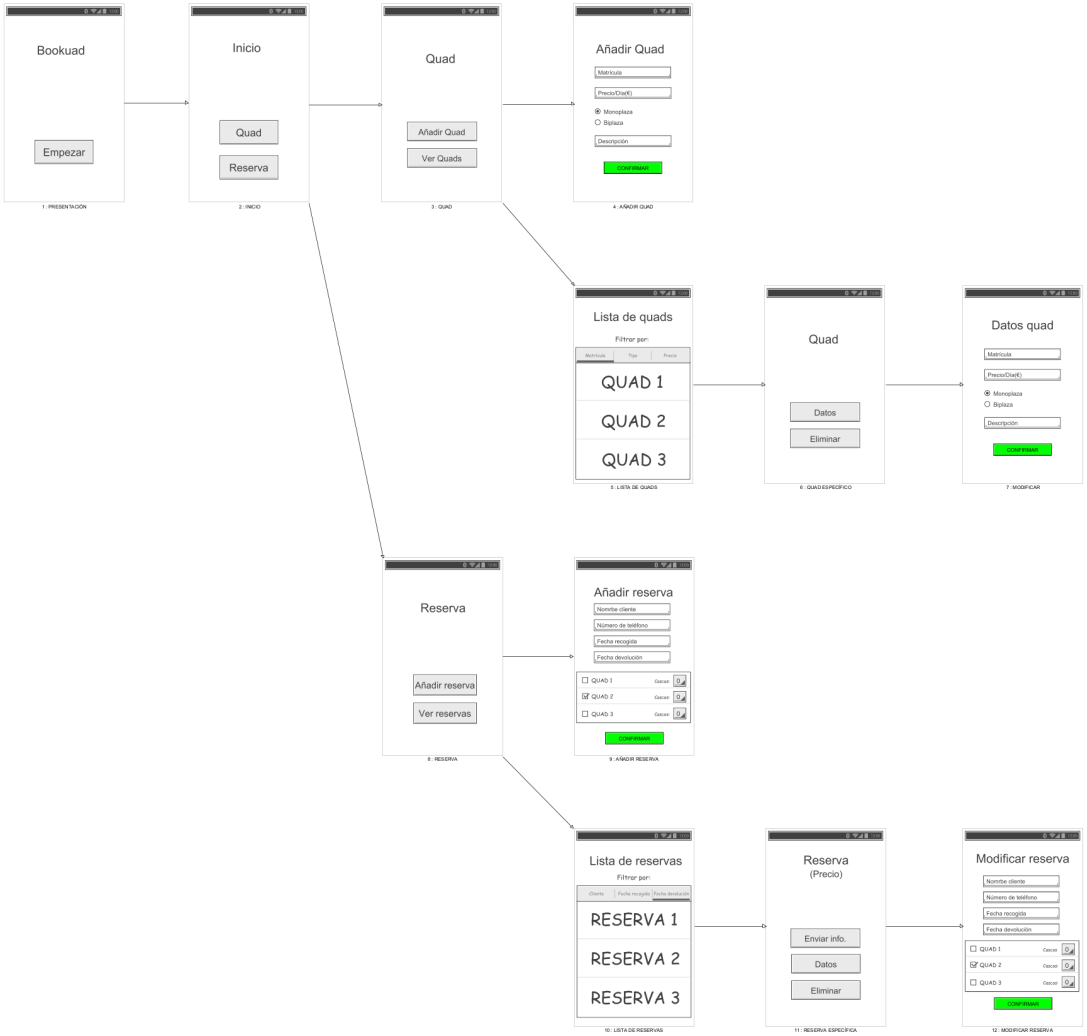

Figura 18: Mapa de navegación completo mostrando flujos entre pantallas

El mapa de navegación muestra la estructura jerárquica de las pantallas de la aplicación y los flujos de navegación entre ellas. Desde la pantalla principal se puede acceder a las secciones de Quads y Reservas, y desde cada una de estas se pueden realizar las operaciones CRUD correspondientes.

4.6 Modelo lógico de la base de datos relacional

El desarrollo de este modelo se ha llevado a cabo mediante la herramienta Entity Relationship Diagram del paquete Database Modeling. El diagrama consta de 2 entidades principales, Quad y Reserva, unidas mediante una relación N:M de la cual sale la entidad `Quad_Reserva`, dentro de la cual se ha añadido el atributo `numCascos`, que representa el número de cascos asignados a uno de los quads en una reserva determinada.

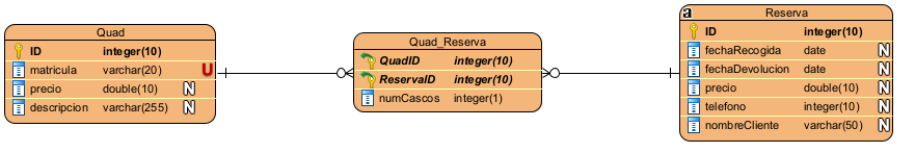

Figura 19: Diagrama entidad-relación de la base de datos

El diagrama se muestra a continuación:

```
-- ------------------------------------------------------
-- Tabla: Quad
-- Basada en la configuracion que acabamos de hacer.
-- ------------------------------------------------------
CREATE TABLE Quad (
    id_quad INTEGER PRIMARY KEY AUTOINCREMENT,
    matricula VARCHAR(20) NOT NULL UNIQUE,
    tipo VARCHAR(20) NOT NULL,
    precio_dia REAL NOT NULL,
    descripcion TEXT
);

-- ------------------------------------------------------
-- Tabla: Reserva
-- ------------------------------------------------------
CREATE TABLE Reserva (
    id_reserva INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre_cliente VARCHAR(255) NOT NULL,
    telefono_cliente VARCHAR(20),
    fecha_recogida DATETIME NOT NULL,
    fecha_devolucion DATETIME NOT NULL,
    precio_total REAL NOT NULL
);

-- ------------------------------------------------------
-- Tabla: Quad_Reserva (Tabla intermedia para N:M)
-- Resuelve la relacion Muchos a Muchos entre Reserva y Quad.
-- ------------------------------------------------------
CREATE TABLE Quad_Reserva (
    id_reserva_fk INTEGER NOT NULL,
    id_quad_fk INTEGER NOT NULL,
    numCascos INTEGER NOT NULL,
    PRIMARY KEY (id_reserva_fk, id_quad_fk),
```

```
34      FOREIGN KEY (id_reserva_fk) REFERENCES Reserva (id_reserva),
35      FOREIGN KEY (id_quad_fk) REFERENCES Quad (id_quad)
36  );
```

4.7 Diagrama de paquetes

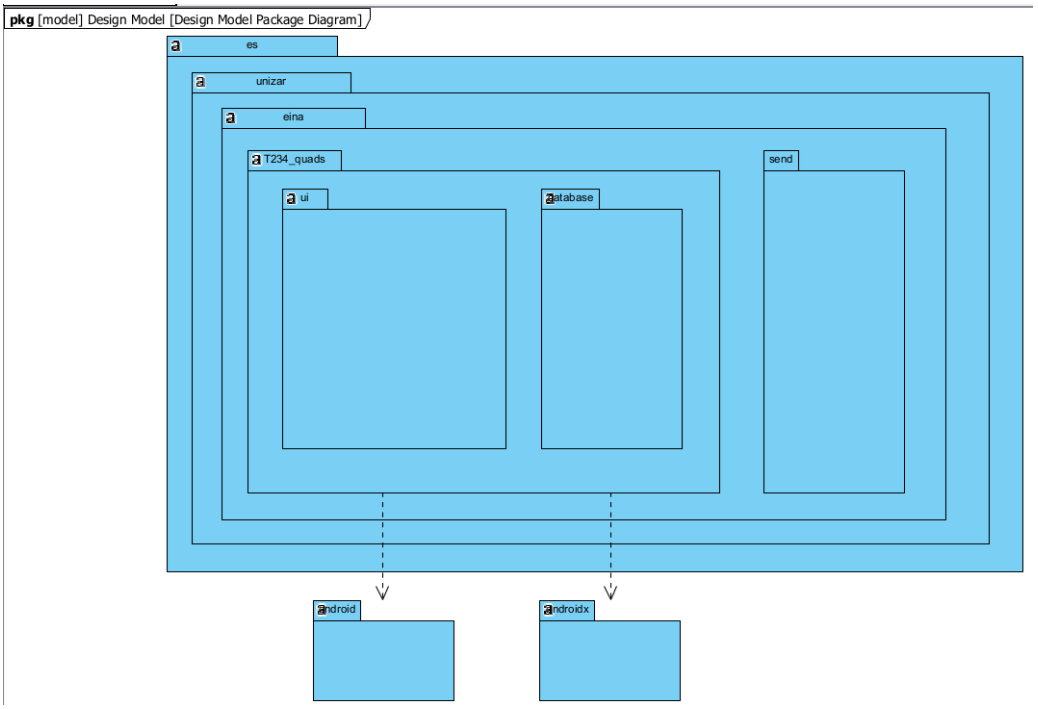

Figura 20: Diagrama de paquetes del sistema

El paquete `ui` es el encargado de todos los aspectos relacionados con la interfaz del usuario. Este se descompone a su vez en los sub-paquetes `uiReserva` y `uiQuad`, que agrupan la lógica de presentación (como ViewModels, Activities, etc.) para la gestión específica de las reservas y el manejo de los quads, respectivamente.

Por otro lado, el paquete `database` es el encargado de todos los aspectos relacionados con la base de datos del sistema (DAO, Repositorios, AppQuadsRoomDatabase). Como indican las flechas de dependencia, tanto `uiReserva` como `uiQuad` necesitan acceder al paquete `database` para consultar y persistir la información.

Finalmente, el paquete `send` es el encargado de todo lo relacionado con la gestión de una comunicación cómoda con el usuario (como el envío de notificaciones o mensajes SMS de confirmación).

4.8 Diagrama de componentes

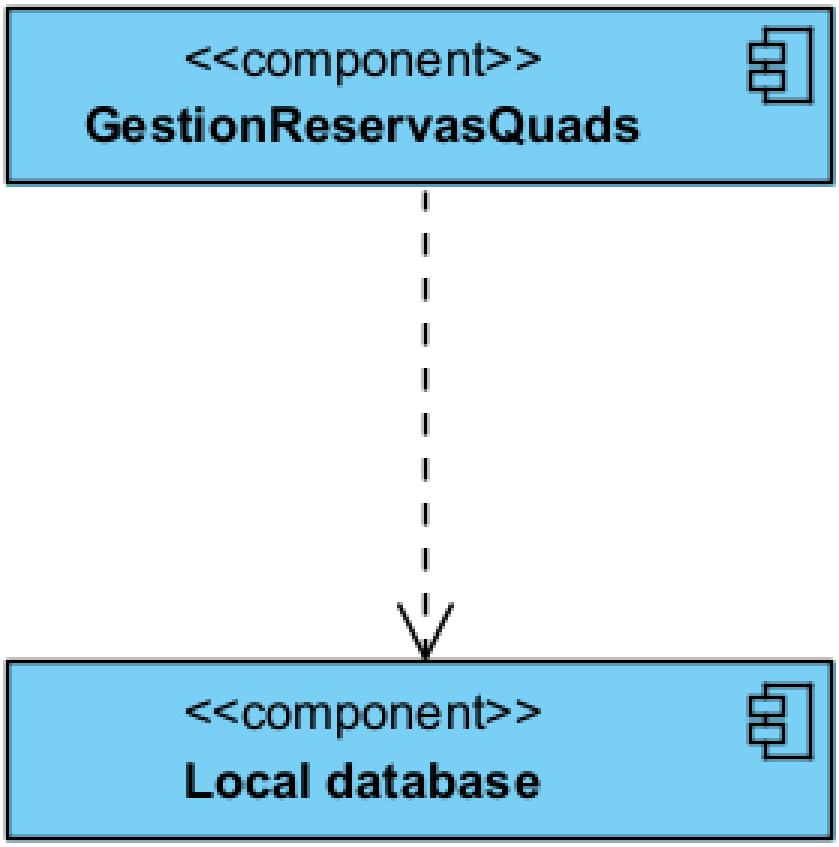

Figura 21: Diagrama de componentes

El componente “GestionReservasQuad” representa la aplicación de gestión de reservas de quads y el componente “Local Database” representa la base de datos que almacena los quads y las reservas de la aplicación.

4.9 Diagrama de despliegue

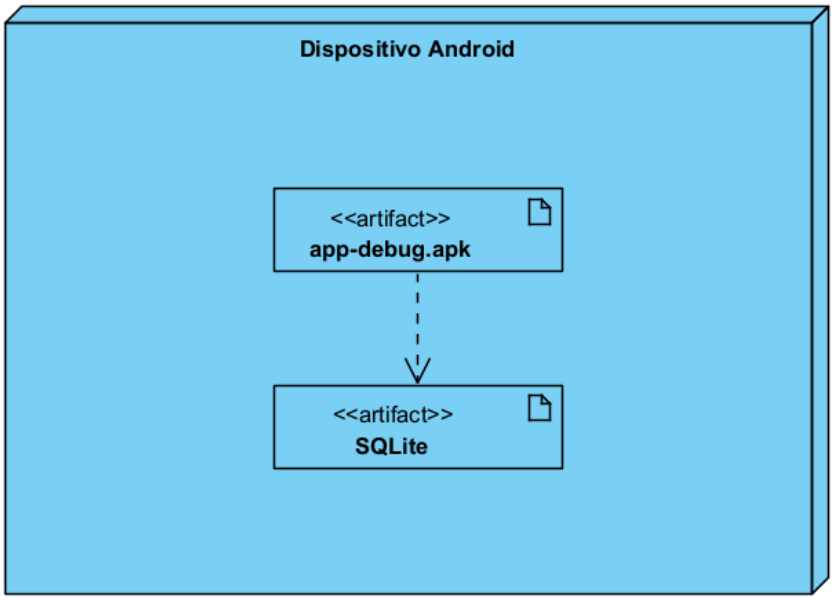

Figura 22: Diagrama de despliegue

Hay 2 artefactos: `app-debug.apk`, que es el paquete instalable de la aplicación generado al compilar el proyecto, y `SQLite`, que representa la base de datos local que utiliza la aplicación para almacenar y gestionar los datos.

5 Diseño de objetos

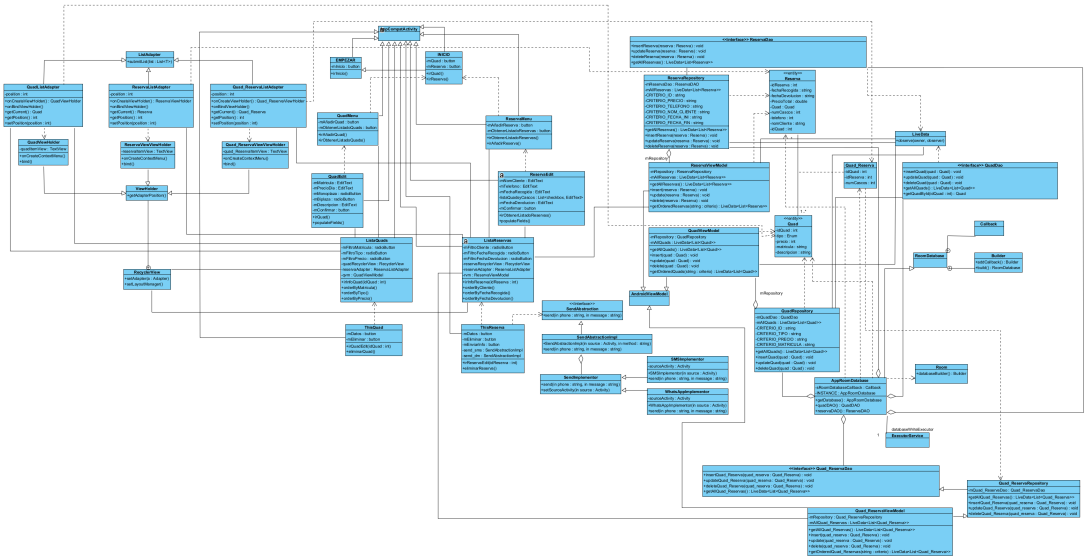

Figura 23: Diagrama de diseño de objetos

El diagrama de diseño de objetos muestra la estructura detallada de las clases del sistema con sus atributos, métodos y relaciones, incluyendo los detalles de implementación específicos de la plataforma Android.

6 Diagramas de secuencia de diseño

6.1 Crear reserva

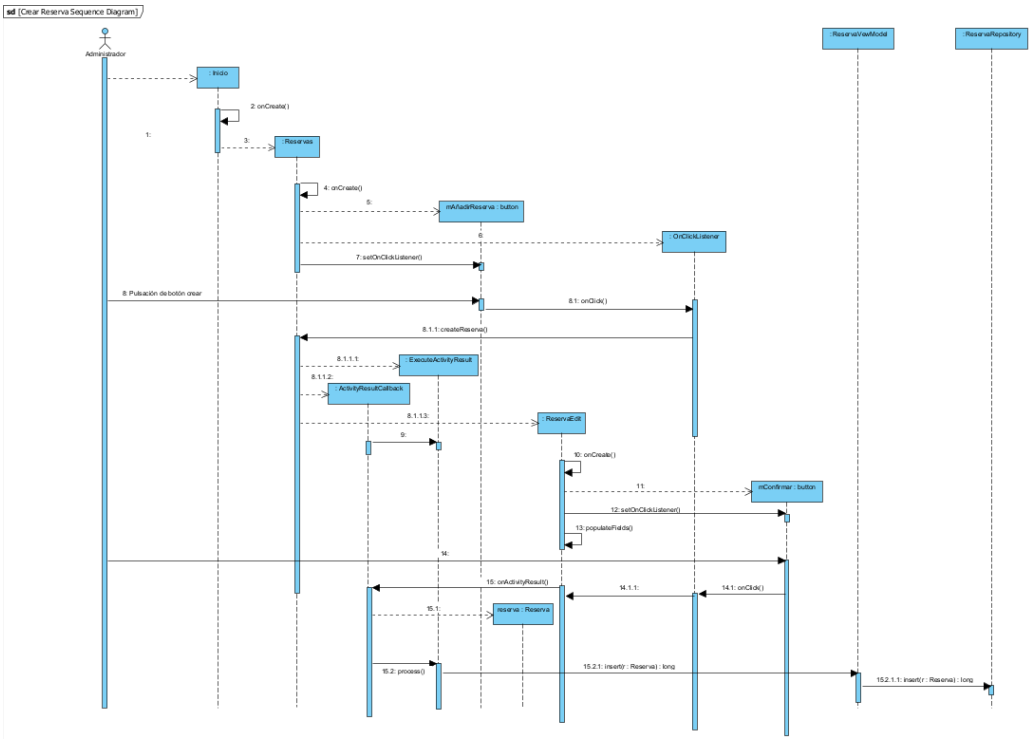

Figura 24: Diagrama de secuencia de diseño - Crear reserva

Este diagrama muestra la interacción detallada entre los componentes del sistema durante la creación de una reserva, incluyendo la interfaz de usuario, el ViewModel, el Repository, el DAO y la base de datos.

6.2 Obtener listado de reservas

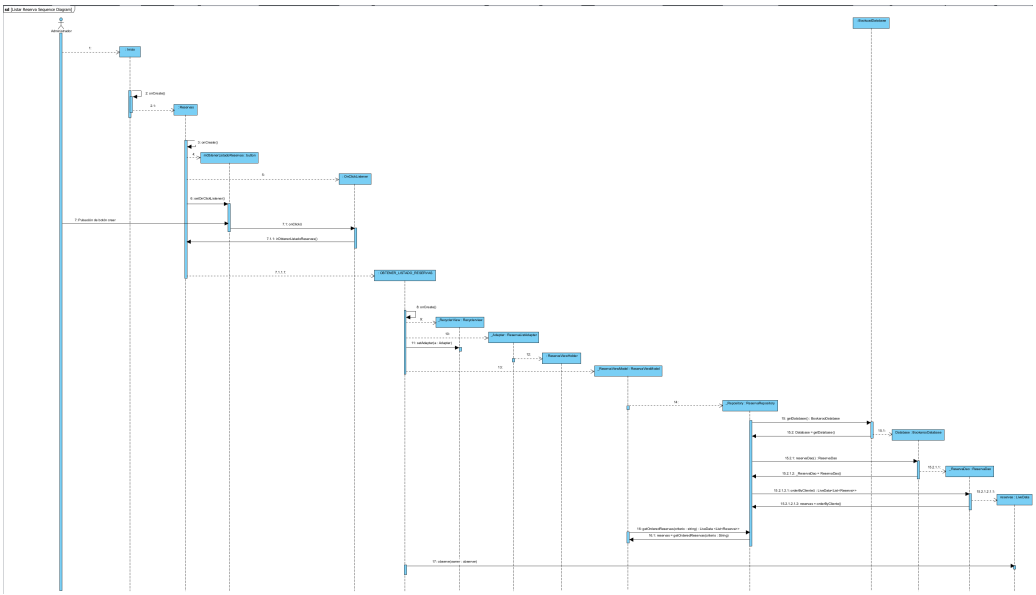

Figura 25: Diagrama de secuencia de diseño - Obtener listado de reservas

Este diagrama ilustra el flujo de obtención y visualización del listado de reservas, desde la petición del usuario hasta la presentación de los datos en la interfaz.

6.3 Eliminar Reserva

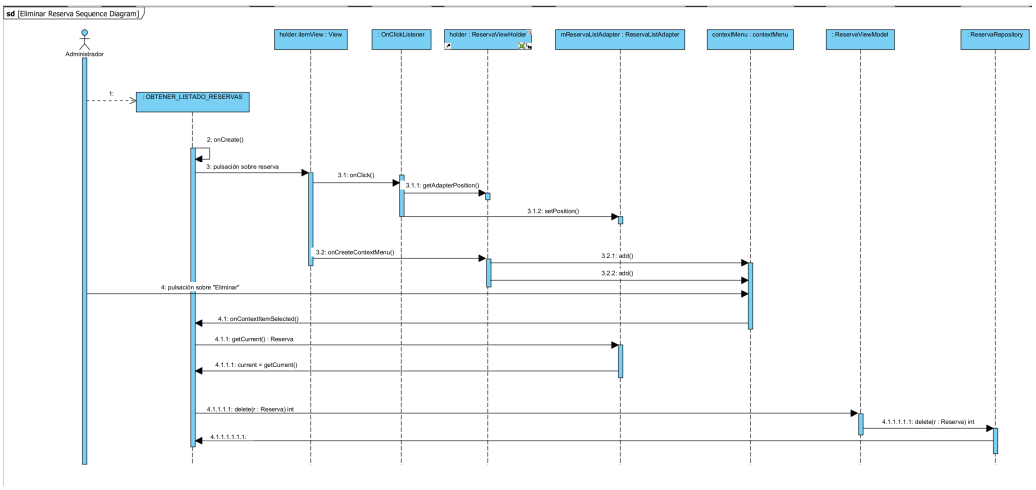

Figura 26: Diagrama de secuencia de diseño - Eliminar reserva

Este diagrama detalla el proceso de eliminación de una reserva, incluyendo la confirmación del usuario y la actualización de la interfaz tras la operación.

7 Implementación en Android Studio

Esta sección describe el proceso de implementación del sistema de gestión de reservas de quads en Android Studio, siguiendo los diseños y especificaciones definidos en las secciones anteriores.

7.1 Tecnologías y herramientas utilizadas

7.1.1. Entorno de desarrollo

- **Android Studio:** IDE oficial para desarrollo Android (versión Hedgehog — 2023.1.1 o superior)
- **Lenguaje:** Java
- **SDK mínimo:** Android API 24 (Android 7.0 Nougat)
- **SDK objetivo:** Android API 34 (Android 14)
- **Sistema de construcción:** Gradle 8.0

7.1.2. Bibliotecas y frameworks

- **Room:** Librería de persistencia para gestión de base de datos SQLite
- **LiveData:** Componente de arquitectura para datos observables
- **ViewModel:** Componente de arquitectura para gestión del estado de UI
- **RecyclerView:** Para mostrar listas eficientemente
- **Material Design Components:** Para componentes de interfaz modernos
- **SMS Manager:** Para envío de mensajes SMS a los clientes

7.2 Arquitectura de la aplicación

La aplicación sigue el patrón de arquitectura **MVVM (Model-View-ViewModel)** recomendado por Google para aplicaciones Android, que proporciona una separación clara de responsabilidades y facilita el testing y mantenimiento del código.

7.2.1. Capas de la arquitectura

1. **Capa de presentación (UI):**
   - Activities y Fragments
   - Adapters para RecyclerView
   - Layouts XML con Material Design
2. **Capa de ViewModel:**

- QuadViewModel
- ReservaViewModel
- Gestión del estado de la UI
- Comunicación con la capa de datos

3. **Capa de datos:**
   - Repository pattern para acceso a datos
   - QuadRepository
   - ReservaRepository
4. **Capa de persistencia:**
   - Room Database
   - DAOs (Data Access Objects)
   - Entidades (Entities)

7.3 Estructura del proyecto

La estructura de paquetes del proyecto sigue las mejores prácticas de organización:

```
app/src/main/java/com/example/quadsreservas/
  database/
    AppQuadsRoomDatabase.java
    entities/
      Quad.java
      Reserva.java
      QuadReservaCrossRef.java
    dao/
      QuadDao.java
      ReservaDao.java
    repository/
      QuadRepository.java
      ReservaRepository.java
  ui/
    quad/
      QuadActivity.java
      QuadListActivity.java
      QuadViewModel.java
      QuadAdapter.java
    reserva/
      ReservaActivity.java
      ReservaListActivity.java
      ReservaViewModel.java
      ReservaAdapter.java
    MainActivity.java
  utils/
    DateUtils.java
```

```
    PriceCalculator.java
    ValidationUtils.java
  send/
    SMSSender.java
```

7.4 Implementación de la base de datos

7.4.1. Entidades Room

Se han implementado las entidades correspondientes a las tablas de la base de datos:
Quad.java:

- Anotada con `@Entity`
- Campos: id_quad, matricula, tipo, precio_dia, descripcion
- Índice único en matrícula

Reserva.java:

- Anotada con `@Entity`
- Campos: id_reserva, nombre_cliente, telefono_cliente, fecha_recogida, fecha_devolucion, precio_total

QuadReservaCrossRef.java:

- Entidad de relación para la asociación N:M
- Campos: id_reserva_fk, id_quad_fk, numCascos
- Claves foráneas definidas

7.4.2. Data Access Objects (DAOs)

Se han implementado los DAOs con las operaciones necesarias:
QuadDao:

- `@Insert`: Insertar nuevo quad
- `@Update`: Actualizar quad existente
- `@Delete`: Eliminar quad
- `@Query`: Consultas personalizadas (listar todos, buscar por ID, ordenar por diferentes campos)

ReservaDao:

- `@Insert`: Insertar nueva reserva
- `@Update`: Actualizar reserva existente
- `@Delete`: Eliminar reserva
- `@Query`: Consultas personalizadas (listar todas, buscar por ID, ordenar por diferentes campos, obtener reservas con quads asociados)

7.5 Implementación de la interfaz de usuario

7.5.1. MainActivity

Pantalla principal que proporciona acceso a las dos secciones principales:

- Botón para gestión de Quads
- Botón para gestión de Reservas
- Implementa Material Design con CardViews

7.5.2. Gestión de Quads

QuadListActivity:

- RecyclerView para mostrar listado de quads
- FloatingActionButton para añadir nuevo quad
- Menú de opciones para ordenar por matrícula, tipo o precio
- Click en item para ver detalles

QuadActivity:

- Formulario para añadir/editar quad
- Validación de campos (matrícula, tipo, precio, descripción)
- Guardado en base de datos mediante ViewModel
- Opciones de modificar y eliminar quad

7.5.3. Gestión de Reservas

ReservaListActivity:

- RecyclerView para mostrar listado de reservas
- FloatingActionButton para crear nueva reserva
- Menú de opciones para ordenar por cliente, móvil, fecha de recogida o devolución
- Click en item para ver detalles

ReservaActivity:

- Formulario completo para crear/editar reserva
- DatePickers para seleccionar fechas
- Spinner para seleccionar quad
- Campo numérico para número de cascos

- Cálculo automático del precio total
- Validación de fechas y campos obligatorios
- Botón para enviar información por SMS
- Opciones de modificar y eliminar reserva

7.6 Funcionalidades implementadas

7.6.1. Validaciones

- **Validación de matrícula:** Formato 4 dígitos + 3 letras, sin duplicados
- **Validación de fechas:** Fecha de devolución posterior a fecha de recogida
- **Validación de campos obligatorios:** Todos los campos requeridos deben estar completos
- **Validación de límites:** Máximo 100 quads y 20000 reservas (requisitos no funcionales)
- **Validación de cascos:** Según tipo de quad (1 para monoplaza, 1-2 para biplaza)

7.6.2. Cálculo de precio

La clase `PriceCalculator` implementa la lógica de cálculo:

- Calcula días entre fecha de recogida y devolución
- Multiplica días por precio diario del quad
- Añade suplemento por cascos si corresponde
- Devuelve precio total

7.6.3. Envío de SMS

La clase `SMSSender` utiliza la API de Android para enviar SMS:

- Solicita permisos de envío de SMS
- Construye mensaje con información de la reserva
- Envía SMS al número del cliente
- Maneja errores de envío

7.7 Persistencia de datos

7.7.1. Room Database

La clase `AppQuadsRoomDatabase` extiende RoomDatabase:

- Singleton pattern para única instancia
- Define las entidades de la base de datos
- Proporciona acceso a los DAOs
- Configuración de versión de base de datos
- Estrategia de migración si es necesaria

7.7.2. Repositories

Los repositorios abstraen el acceso a datos:
QuadRepository:

- Métodos para insertar, actualizar, eliminar quads
- Métodos para consultar quads con LiveData
- Operaciones en background thread

ReservaRepository:

- Métodos para insertar, actualizar, eliminar reservas
- Métodos para consultar reservas con LiveData
- Gestión de relaciones con quads
- Operaciones en background thread

7.8 Gestión del ciclo de vida

7.8.1. ViewModels

Los ViewModels sobreviven a cambios de configuración:

- Mantienen el estado de la UI
- Exponen LiveData observables
- Comunican con los Repositories
- No mantienen referencias a Activities

7.8.2. LiveData

Uso de LiveData para actualizaciones reactivas:

- Observación del ciclo de vida de componentes
- Actualización automática de UI cuando cambian los datos
- Evita memory leaks
- Datos siempre actualizados en rotaciones de pantalla

8 Bibliografía

- Visual Paradigm Forums. Foro oficial de Visual Paradigm.
  Disponible en: https://forums.visual-paradigm.com/
- Stack Overflow. Sección “visual-paradigm” en Stack Overflow.
  Disponible en: https://stackoverflow.com/questions/tagged/visual-paradigm
- Android Developers. Documentación oficial de Android.
  Disponible en: https://developer.android.com/docs
- Android Room Persistence Library. Guía de Room.
  Disponible en: https://developer.android.com/training/data-storage/room
- Android Architecture Components. Guía de arquitectura.
  Disponible en: https://developer.android.com/topic/architecture
- Material Design for Android. Guía de Material Design.
  Disponible en: https://material.io/develop/android
- Gradle Build Tool. Documentación de Gradle.
  Disponible en: https://gradle.org/
- Stack Overflow. Comunidad de desarrolladores Android.
  Disponible en: https://stackoverflow.com/questions/tagged/android