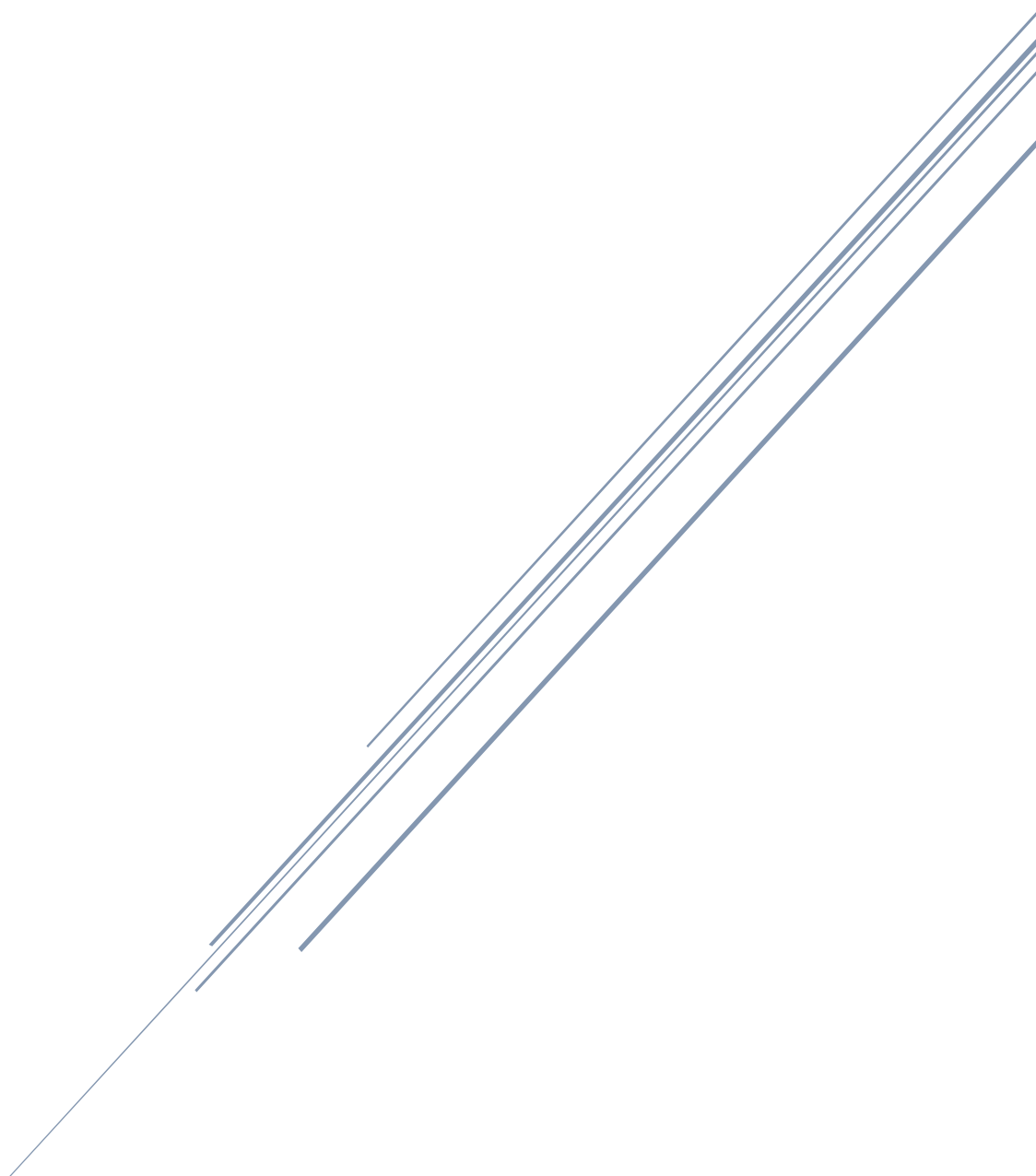


# Memoria aparcamiento



Sergio Simón Fernández

1ºDAM

## Índice

Introducción.....	2
Objetivos.....	2
Tecnologías y Metodologías.....	2
Recursos.....	3
Presupuesto.....	5
Porque del diseño.....	6
Vista principal.....	6
Vista Agregar.....	7
Vista gestionar.....	8
Aparcar Vista.....	9
Diagramas.....	10
Diagrama de clases.....	10
Diagramas de casos de uso.....	11
Diagramas de secuencia.....	12
Requisitos funcionales, no funcionales y de información.....	13
Requisitos funcionales.....	13
Requisitos no funcionales.....	13
Requisitos de información.....	14
Informes de casos de uso.....	15
Enunciado.....	21

# Introducción

Para este proyecto he planteado llevar a acabo un sistema para llevar un registro de los vehículos aparcados en un parquin.

Este supuesto parquin tiene 10 plazas y dos de ellas con capacidad para vehículos eléctricos si estas plazas se ocupan por vehículos eléctricos no se permitirá entrar más a no ser que sean híbridos.

# Objetivos

Queremos poder almacenar la información de los vehículos aparcados y tener una base de datos con los vehículos y sus propietarios dados de alta en el sistema

Poder gestionar los vehículos que entran y la hora de entrada de estos

# Tecnologías y Metodologías

Se ha utilizado como lenguaje de programación Kotlin, para la interfaz gráfica se ha utilizado JavaFX como base de datos se ha usado SQLite y como inyector de dependencias Koin.

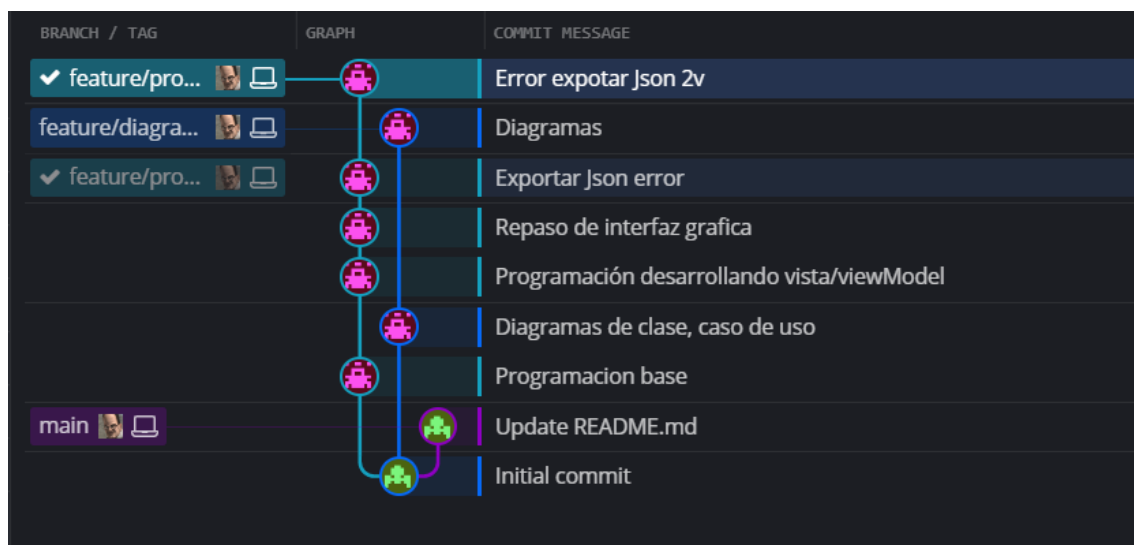
Se han utilizado utilizado arquitecturas que respetaran los principios SOLID

El proyecto se ha almacenado en un repositorio remoto en Github para la organización del proyecto se ha utilizado gitflow

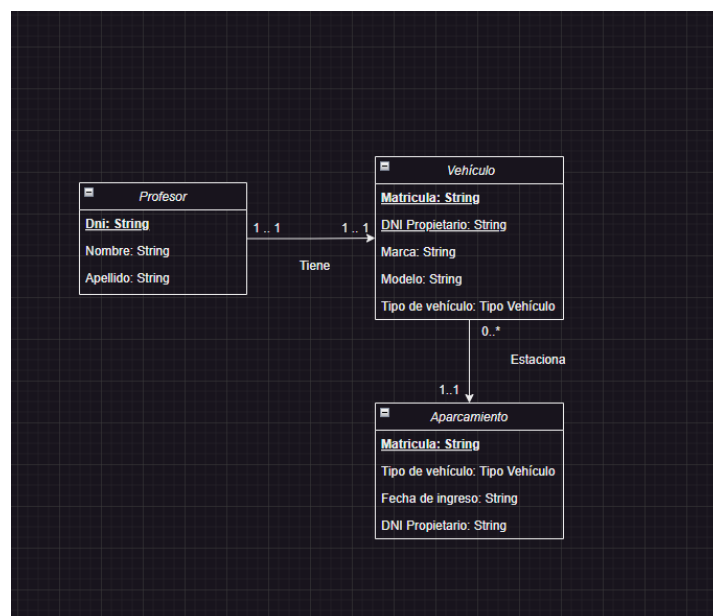
Se ha elegido utilizar las anteriores tecnologías y metodologías debido a que son las que se han visto en clase

# Recursos

Para manejar de una manera visual el repositorio se ha utilizado git kraken



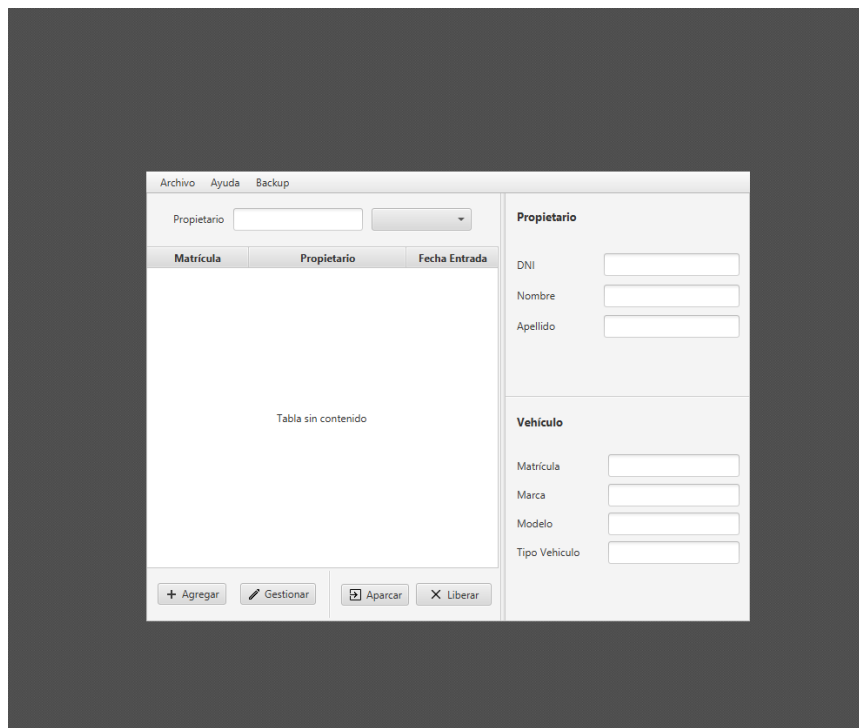
Para poder dibujar los diagramas se ha utilizado Draw.io



Como programa para poder codificar el código se ha usado IntelliJ



Para diseñar la interfaz de manera mas cómoda se ha utilizado SceneBuilder



# Presupuesto

Para calcular el presupuesto supondremos que como un desarrollador Freelance que cobraremos unos 30€ la hora.

Y según una extensión para poder tener un registro de horas al proyecto de programación se le han dedicado un total de 19 horas mas unas 6 aprox que no pudo registrar al solo contar las horas programadas . En total 25

Proyecto-Aparcamiento 15 hrs 26 mins	Proyecto-Aparcamiento-Definitivo 3 hrs 5 mins
---	--

Los proyectos están separados por dos nombres porque no me funcionaba una cosa y la estaba comprobando en otra

Por lo tanto  $25 * 30 = 750€$  le costaría la aplicación

# Porque del diseño

## Vista principal

Archivo Ayuda Backup

Propietario

Matrícula	Propietario	Fecha Entrada
Tabla sin contenido		

**Propietario**

DNI

Nombre

Apellido

**Vehículo**

Matrícula

Marca

Modelo

Tipo Vehículo

En la vista principal podremos ver un cuadro para buscar el nombre del propietario del vehículo aparcado en la tabla.

Un ComboBox para filtrar los tipo de vehículos que aparecerán en la tabla

Los formularios de la derecha se rellenaran al seleccionar un vehículo aparcado

Al pulsar el botón Agregar veremos la vista agregar

Al pulsar el botón Gestionar veremos la vista gestionar

Al pulsar el botón Gestionar veremos la vista aparcar

Para liberar una plaza se debe seleccionar un vehículo de la tabla y pulsar liberar

## Vista Agregar

Archivo Ayuda

**Propietario**

DNI

Nombre

Apellido

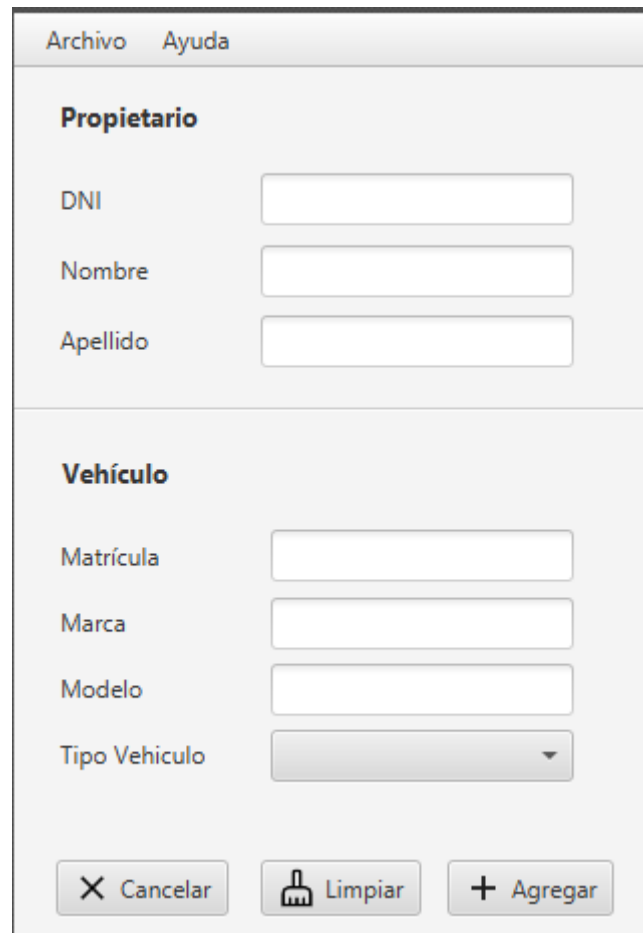
**Vehículo**

Matrícula

Marca

Modelo

Tipo Vehículo

✕ Cancelar  Limpiar + Agregar

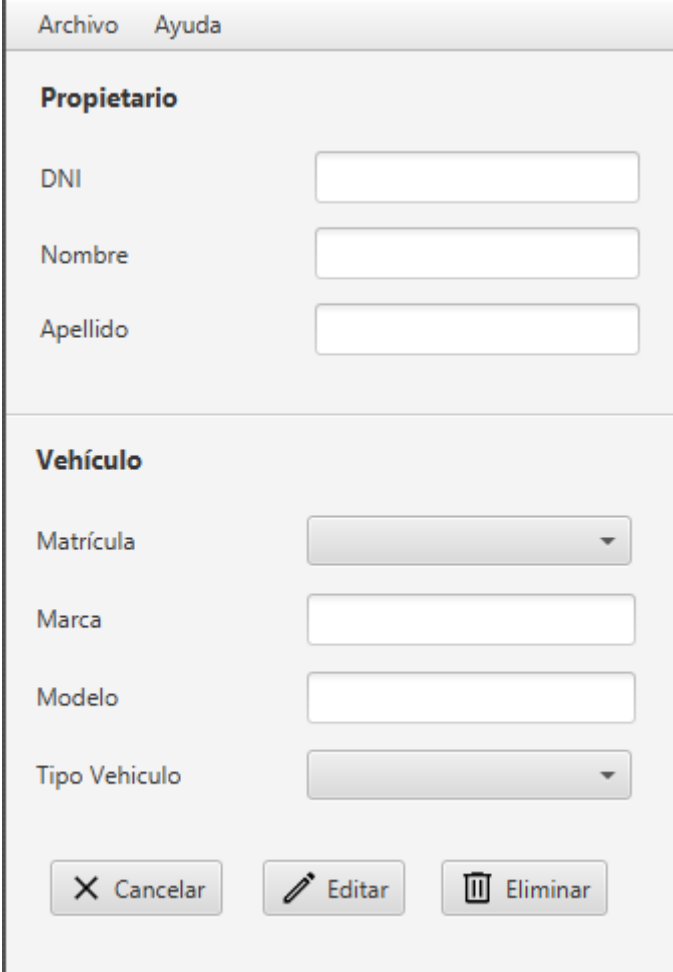
Aquí el usuario podrá introducir los datos del propietario y el vehículo

Si elige la opción de limpiar se eliminara los datos del formulario

Si elige la opción de Agregar y el DNI y la matricula no están previamente registrados se permitirá agregar los datos a la base de datos



## Vista gestionar



Archivo Ayuda

**Propietario**

DNI

Nombre

Apellido

**Vehículo**

Matrícula

Marca

Modelo

Tipo Vehículo

Aquí solo se podrá realizar una acción cuando se haya elegido una matricula registrada

Para poder eliminar un propietario y su vehículo seleccionaremos una matricula y el botón eliminar. Al seleccionar la matricula se nos rellenara el formulario a modo de información de los datos asociados

Para editar los datos de un propietario y su vehículo seleccionaremos una matricula y el botón eliminar. Se considera que no debería poderse editar la matricula de un vehículo ya que esta es inmutable si se quisiera modificar debería crear uno nuevo y eliminar el actual

## Aparcar Vista

Archivo Ayuda

**Propietario**

DNI

Nombre

Apellido

**Vehículo**

Matrícula

Marca

Modelo

Tipo Vehículo

Aquí el usuario solo puede escoger una matricula se cargaran los datos que no son evitables y podrá elegir si aparcarlo o no. Al aparcarlo se mostrara en la tabla de vehículos aparcados

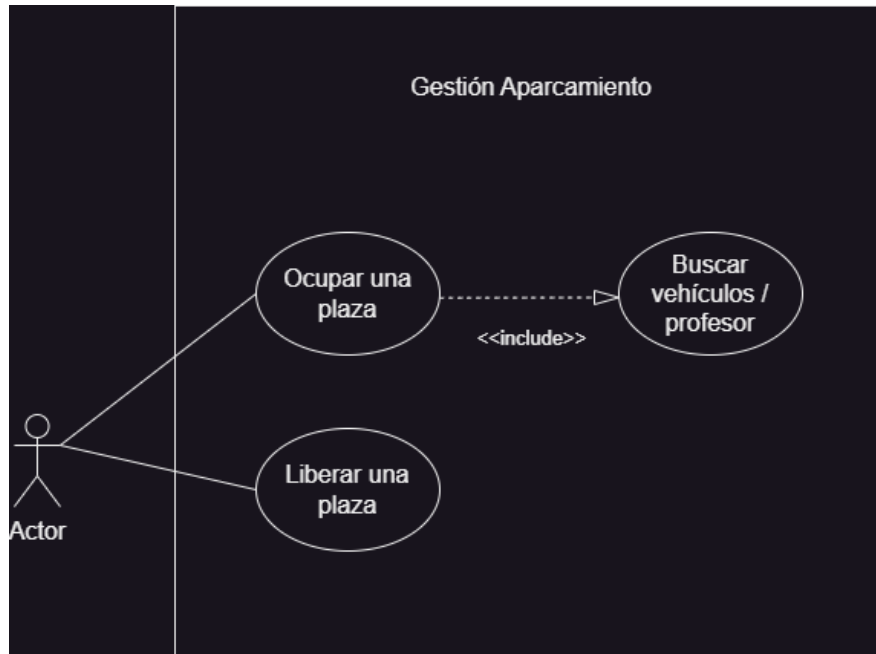
# Diagramas

## Diagrama de clases

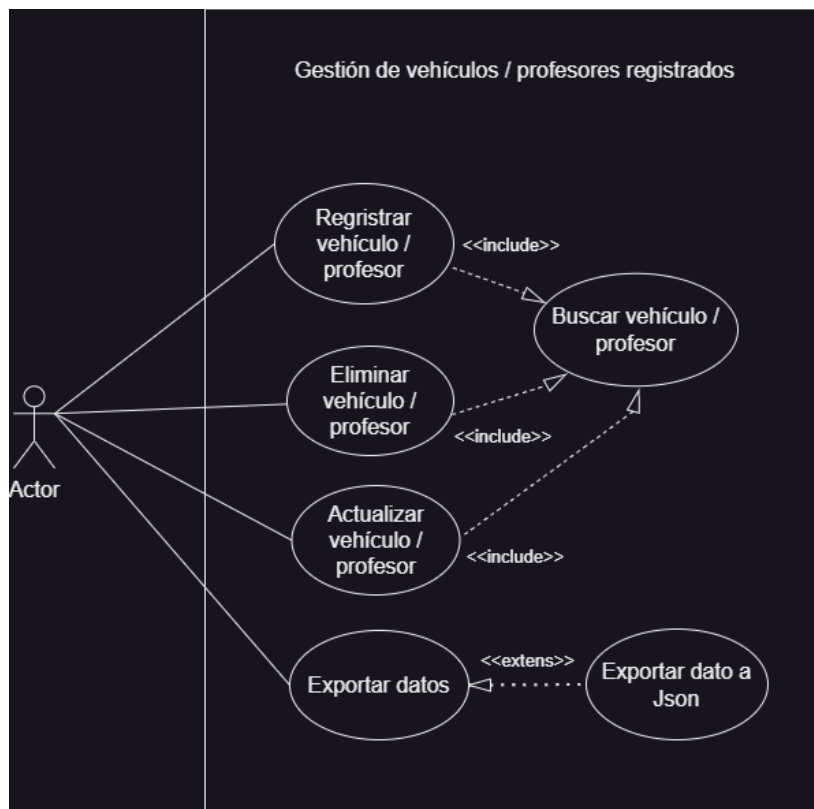


## Diagramas de casos de uso

### Gestionar Aparcamiento

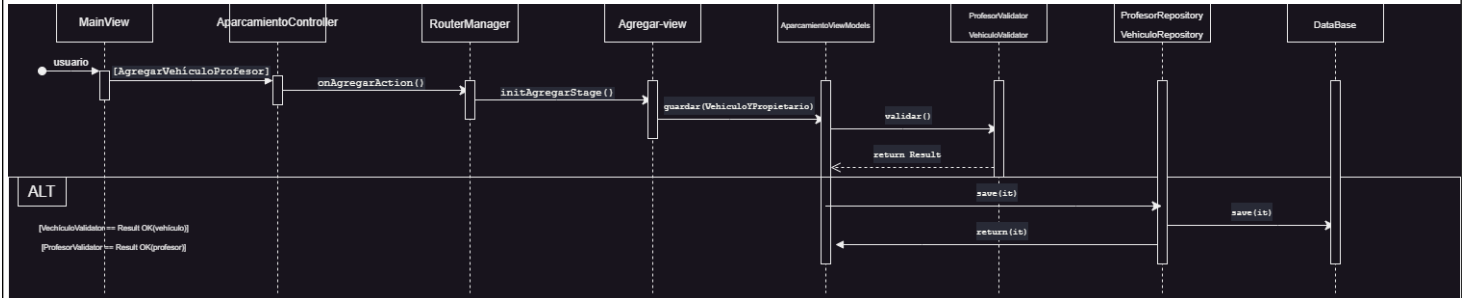


### Gestionar Vehículos y Profesores

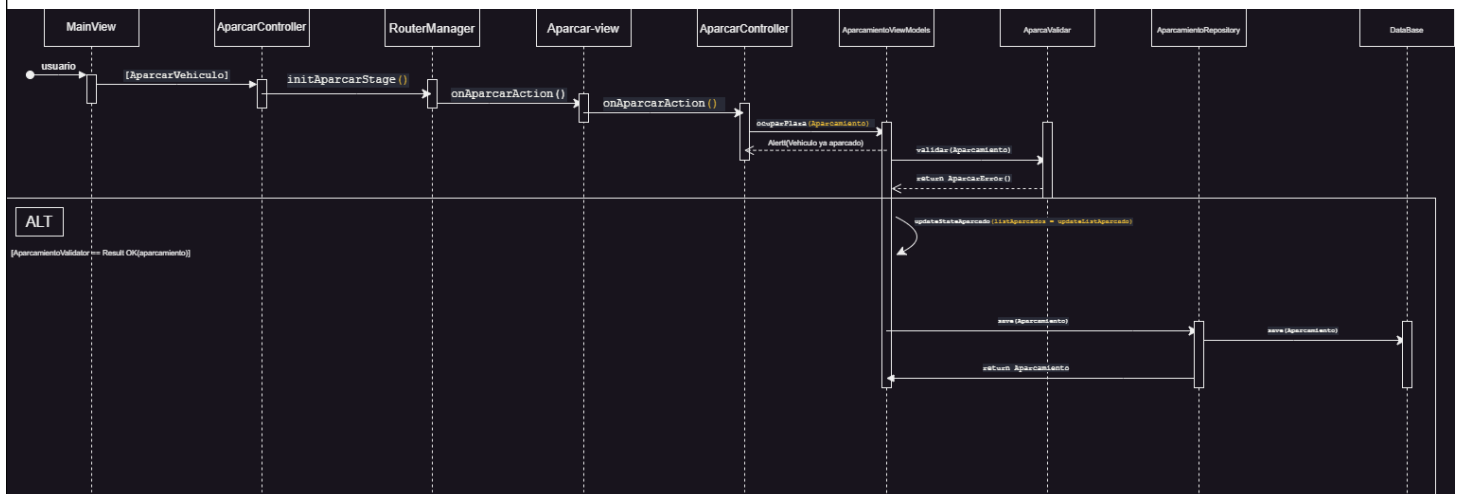


# Diagramas de secuencia

## Diagrama de secuencia Aparcar



## Diagrama de secuencia Guardar Vehículo



# Requisitos funcionales, no funcionales y de información

## Requisitos funcionales

RF-01 El usuario podrá registrar un vehículo proporcionando los datos del vehículo y su propietario

RF-02 El usuario podrá editar los datos sobre un vehículo y su propietario al seleccionar la matrícula

RF-03 El usuario podrá eliminar los datos sobre un vehículo y su propietario al seleccionar la matrícula

RF-04 El usuario podrá aparcar un vehículo siempre y cuando haya sido registrado anteriormente

RF-05 Si un vehículo está aparcado el usuario podrá consultar los datos del vehículo y su propietario al seleccionarlo

RF-06 El usuario podrá liberar una plaza eliminando un vehículo aparcado en el registro

## Requisitos no funcionales

RNF-01: La base de datos tiene que ser en SQLite

RNF-02: Realizar una aplicación de escritorio

RNF-03: Utilizar Koin para inyección de dependencias

RNF-04: Uso de Model-View-ViewModel para el código

RNF-05: Utilizar Kotlin como lenguaje

RNF-06: Uso de Kotlin logger para tener un log de la aplicación

RNF-07: Uso de gson para crear DTO

RNF-08: Uso de JavaFX para la creación de la interfaz

RNF-09: Testeo de ViewModel, repositorios, controladores y la base de datos

RNF-10: Uso de programación reactiva (observers, listeners)

RNF-11: No se pueden aparcar más de 10 vehículos a la vez

RNF-12: Exportar vehículos y sus propietarios registrados en la BBDD en formato JSON

RNF-13: Usar Railway Oriented Programming en el proyecto

RNF-14: Respetar siempre que sea posible los principios SOLID

# Requisitos de información

## **RI-01 Profesor**

- DNI [String Único]
- Nombre [String]
- Apellido [String]

## **RI-02 Vehículo**

- Matricula [String Único]
- DNI\_Propietario [String]
- Marca [String]
- Modelo [String]
- Tipo de Vehículo [Tipo Vehículo]

## **RI-03 Aparcamiento**

- Matricula [String Único]
- Tipo de Vehículo [Tipo Vehículo]
- DNI\_Propietario [String]
- Fecha de ingreso [String]

# Informes de casos de uso

## RF-01 Registrar un vehículo y su propietario

1. **Resumen:** El caso de uso de registrar vehículo y su propietario al actor usuario registrar en el sistema un vehículo con su propietario en el sistema para luego poder aparcarse el vehículo.
2. **Código:** RF-01
3. **Nombre:** Registrar un vehículo y su propietario
4. **Actor:** Usuario
5. **Precondiciones:**
  - El dni del propietario no está registrado en el sistema
  - La matrícula del vehículo no está registrada en el sistema
6. **Postcondiciones:**
  - Se guarda en la base de datos un vehículo
  - Se guarda en la base de datos un profesor
7. **Secuencia de pasos:**
  - Flujo Principal:
    1. El usuario selecciona la opción de “Agregar”
    2. El sistema muestra un formulario de vehículos y de profesores
    3. El usuario rellena los datos del formulario
    4. El usuario confirma la acción
    5. El sistema valida los datos y comprueba que no se haya guardado un propietario con el mismo dni ni un vehículo con la misma matrícula
    6. El sistema guarda los datos en la base de datos
  - Flujo Alternativo
    1. 1. En cualquier paso del flujo principal, el usuario puede cancelar la acción, regresando al inicio.
    2. 5.1 En caso de que ya exista un propietario con el mismo dni o un vehículo con la misma matrícula o los datos no se hallan podido validar se cancelará la acción de guardar los datos.



## RF-02 Editar un vehículo y su propietario

1. **Resumen:** El caso de uso de editar vehículo y su propietario permite al actor usuario registrar en el sistema un vehículo con su propietario en el sistema para luego poder aparcar el vehículo.
2. **Código:** RF-02
3. **Nombre:** Editar un vehículo y su propietario
4. **Actor:** Usuario
5. **Precondiciones:**
  - Debe de haber al menos un vehículo con su propietario asociado registrado en el sistema
6. **Postcondiciones:**
  - Se edita en la base de datos el vehículo con la matricula seleccionada
  - Se edita en la base de datos el profesor que posee el vehículo
7. **Secuencia de pasos:**
  - Flujo Principal:
    1. El usuario selecciona la opción de “Gestionar”
    2. El sistema muestra un formulario de vehículos y de profesores
    3. El usuario selecciona la matricula del vehículo que quiere editar
    4. El sistema carga los datos asociados al vehículo y su propietario
    5. El usuario modifica los datos que sean necesarios
    6. El usuario selecciona la opción “Editar”
    7. El sistema valida los datos y comprueba que el dni del propietario no pertenece a otro ya registrado
    8. El sistema actualiza los datos en la base de datos del vehículo y su propietario
  - Flujo Alternativo
    1. 1. En cualquier paso del flujo principal, el usuario puede cancelar la acción, regresando al inicio.
    2. 7.1 En caso de que ya exista un propietario con el mismo dni o los datos no se hallan podido validar se cancelara la acción de guardar los datos.

## RF-03 Eliminar un vehículo y su propietario

1. **Resumen:** El caso de uso de eliminar un vehículo y su propietario permite al actor usuario eliminar del sistema un vehículo con su propietario en el sistema para luego poder aparcar el vehículo.
2. **Código:** RF-03
3. **Nombre:** Eliminar vehículo y propietario
4. **Actor:** Usuario
5. **Precondiciones:**
  - Debe de haber al menos un vehículo con su propietario asociado registrado en el sistema
6. **Postcondiciones:**
  - Se elimina en la base de datos un vehículo
  - Se elimina en la base de datos un profesor
7. **Secuencia de pasos:**
  - Flujo Principal:
    1. El usuario selecciona la opción de “Gestionar”
    2. El sistema muestra un formulario de vehículos y de profesores
    3. El usuario selecciona la matricula del vehículo que quiere eliminar
    4. El sistema carga los datos asociados del vehículo y su propietario
    5. El usuario selecciona la opción “Eliminar”
    6. El sistema elimina el vehículo con la matricula selecciona y su propietario
  - Flujo Alternativo
    1. 1. En cualquier paso del flujo principal, el usuario puede cancelar la acción, regresando al formulario.

## RF-04 Aparcar un vehículo

1. **Resumen:** El caso de uso de aparcar un vehículo permite al actor usuario registrar en el sistema un vehículo con su propietario en el sistema para luego poder tener un registro de estos.
2. **Código:** RF-04
3. **Nombre:** Aparcar un vehículo
4. **Actor:** Usuario
5. **Precondiciones:**
  - Debe de haber al menos un vehículo con su propietario asociado registrado en el sistema
6. **Postcondiciones:**
  - Se guarda en la base de datos un vehículo aparcado (aparcado)
7. **Secuencia de pasos:**
  - Flujo Principal:
    1. El usuario selecciona la opción de “Aparcar”
    2. El sistema muestra un formulario de vehículo y de profesor
    3. El usuario elige una matricula de los vehículos registrados
    4. El sistema rellana los datos del formulario
    5. El usuario secciona la opción de aparcar
    6. El sistema comprueba que no se haya aparcado el mismo vehículo anteriormente
    7. El sistema guarda los datos en la base de datos y actualiza la lista de vehículos aparcados
  - Flujo Alternativo
    1. 1. En cualquier paso del flujo principal, el usuario puede cancelar la acción, regresando al inicio.
    2. 6.1 En caso de que ya exista un vehículo aparcado el sistema se cancela la acción

## RF-05 Seleccionar vehículo aparcado

1. **Resumen:** El caso de uso de seleccionar vehículo aparcado permite al actor usuario consultar los datos relacionados del vehículo y su propietario.
2. **Código:** RF-05
3. **Nombre:** Seleccionar vehículo aparcado
4. **Actor:** Usuario
5. **Precondiciones:**
  - Debe de haber al menos un vehículo aparcado
6. **Postcondiciones:**
  - Se mostrara la información en el formulario del vehículo asociado y su propietario en el formulario
7. **Secuencia de pasos:**
  - Flujo Principal:
    1. El usuario selecciona un vehículo aparcado
    2. El sistema rellena los datos del formulario asociado al vehículo seleccionado
  - Flujo Alternativo
    1. 1. En cualquier paso del flujo principal, el usuario puede cancelar la acción, regresando al inicio.

## RF-06 Liberar una plaza

1. **Resumen:** El caso de uso de liberar una plaza permite al actor usuario eliminar en el sistema un vehículo aparcado para que otro pueda ocupar su lugar.
2. **Código:** RF-06
3. **Nombre:** Liberar una plaza
4. **Actor:** Usuario
5. **Precondiciones:**
  - Debe de haber al menos un vehículo aparcado
  - Se debe seleccionar un vehículo
6. **Postcondiciones:**
  - Se elimina el vehículo seleccionado de la lista de vehículos seleccionados
7. **Secuencia de pasos:**
  - Flujo Principal:
    1. El usuario selecciona un vehículo
    2. El usuario selecciona la opción “Liberar”
    3. El sistema elimina de la lista de vehículos aparcados y la base de datos el vehículo aparcado
  - Flujo Alternativo
    1. Si el usuario no selecciona un vehículo de la lista de vehículos aparcados no se realizara ninguna acción

# Enunciado

Se nos ha solicitado desarrollar una aplicación de escritorio para un instituto. Para la interfaz gráfica, se utilizará JavaFX, y el lenguaje de programación elegido será Kotlin. Para el almacenamiento de información, se utilizará SQLite.

En este instituto, contamos con un estacionamiento que tiene capacidad para 10 vehículos. Los vehículos eléctricos solo podrán estacionar en las plazas designadas para vehículos eléctricos. Además, los vehículos híbridos también podrán utilizar estas plazas, dándoles prioridad. Esto significa que si llegan dos vehículos híbridos, no habrá espacio para más vehículos eléctricos, pero sí para los híbridos. Cabe mencionar que no se permitirá la entrada de cualquier vehículo al centro; estos deben estar registrados en el sistema, incluyendo las especificaciones básicas del vehículo, como la matrícula, la marca, el modelo y el tipo de motor. Asimismo, se deben registrar los datos básicos del propietario, como el DNI, el nombre y el apellido.

Se nos solicita desarrollar una aplicación que permita mostrar los vehículos estacionados en una tabla, donde se muestre la matrícula, el nombre del propietario y la fecha de entrada. Además, se deberá implementar la funcionalidad de registrar, editar y eliminar vehículos junto con sus propietarios, así como la posibilidad de estacionar un vehículo en una plaza y liberarla. Al seleccionar un vehículo en la tabla, se cargarán los datos asociados del vehículo y su propietario en un formulario para visualizar la información.

Para prevenir la pérdida de información en la aplicación, se podrá exportar los datos de los vehículos registrados y sus propietarios a un archivo JSON.

# Implementación

## Koin

Se ha optado por usar la librería Koin para la inyección de dependencias y así facilitar la codificación del código

```
val AppDIModule = module { this: Module

    singleOf(:: AppConfig)

    singleOf(:: DataBaseService)

    singleOf(:: AparcameintoRepositoryImpl) { this: BeanDefinition<AparcameintoRepositoryImpl>
    |   bind<AparcamientoRepository>()
    | }

    singleOf(:: VehiculosRepositoryImpl) { this: BeanDefinition<VehiculosRepositoryImpl>
    |   bind<VehiculosRepository>()
    | }

    singleOf(:: ProfesorRepositoryImpl) { this: BeanDefinition<ProfesorRepositoryImpl>
    |   bind<ProfesorRepository>()
    | }

    singleOf(:: StorageAparcameintoImpl) { this: BeanDefinition<StorageAparcameintoImpl>
    |   bind<StorageAparcamiento>()
    | }

    singleOf(:: AparcamientoViewModels)

}
```

Como se puede se ha inyectado:

- AppConfig para obtener de un fichero de configuración la url de la base de datos
- Los repositorio donde se realizamos las consultas
- ViewModel el cual hace de controlador entre las vistas y los demás módulos del programa

## Gson

Para exportar la información a Json se ha utilizado la librería Gson al ser las fácil de configurar para JavaFX

```

override fun save(file: File, vehiculos: List<Vehiculo>, profesor: List<Profesor>): Result<Long, VehiculoError> {
    return try {
        val gson = GsonBuilder()
            .setPrettyPrinting()
            .registerTypeAdapter(ProfesorDto::class.java, ProfesorDtoAdapter())
            .registerTypeAdapter(VehiculoDto::class.java, VehiculoDtoAdapter())
            .registerTypeAdapter(ExportBBDDJson::class.java, ExportBBDDJsonAdapter())
            .create()

        val exportBBDDJson = ExportBBDDJson(profesor.toListDto(), vehiculos.toListDto(), )
        val type: Type = object : TypeToken<ExportBBDDJson>() {}.type

        val jsonString = gson.toJson(exportBBDDJson, type)
        file.writeText(jsonString)
        Ok(vehiculos.size.toLong())
    } catch (e: Exception) {
        Err(VehiculoError.SaveJson("Error al escribir el JSON: ${e.message}"))
    }
}

```

Se han tenido que usar .registerTypeAdapters para poder exportar las listas debido a que Gson no las podía ver al notificar que eran privadas.

## SQLite

Para la base de datos se ha optado por SQLite al ser más fácil de transportar los datos

```

override fun findAll(): List<Profesor> {
    logger.debug { "Buscando profesores" }
    val sql = "SELECT * FROM profesores"
    var profesores = mutableListOf<Profesor>()
    databaseService.db.use { it: Connection!
        it.prepareStatement(sql).use { stm →
            val rs = stm.executeQuery()
            while (rs.next()) {
                profesores.add(
                    Profesor(
                        dni = rs.getString( columnLabel: "dni"),
                        nombre = rs.getString( columnLabel: "nombre"),
                        apellido = rs.getString( columnLabel: "apellido")
                    )
                )
            }
        }
    }
    return profesores
}

```



## Railway Oriented Programming (Result)

Para poder validar de manera más efectiva se ha optado por usar Railway Oriented Programming para validar los datos del sistema.

```
fun Vehiculo.validar(): Result<Vehiculo, VehiculoError> {  
  
    val matriculaRegex = "[0-9]{1,4}(?!.*(LL|CH))[BCDFGHJKLMNPRSTVWXYZ]{3}".toRegex()  
  
    if (!matriculaRegex.matches(this.matricula)) {  
        return Err(VehiculoError.ValidateProblem("La matricula esta mal formada"))  
    }  
  
    if (this.marca.isEmpty()) {  
        return Err(VehiculoError.ParametroNoIntroducido("La marca no ha sido introducida"))  
    }  
  
    if (this.modelo.isEmpty()) {  
        return Err(VehiculoError.ParametroNoIntroducido("El modelo no ha sido introducido"))  
    }  
  
    if (this.tipoVehiculo == Vehiculo.TipoVehiculo.NONE) {  
        return Err(VehiculoError.ParametroInvalido("El tipo vehiculo no exite o no es valido"))  
    }  
  
    return Ok( value: this )  
}
```