PARKGEST

IES Marqués de Comares. CFGS Diseño de Aplicaciones Multiplataforma

Curso 24-25

Tutor: Antonio Rafael Luque Castro

Autor: Isaac Delgado Román

08-06-2025

1 TABLA DE CONTENIDO

2 ANÁLISIS DE REQUISITOS DEL PROYECTO.

2.1 DESCRIPCIÓN GENERAL DE LA APLICACIÓN.

Parkgest es una aplicación para gestionar las diferentes áreas de un parking con el fin de poder mandar facturas a los clientes de forma correcta y automatizada.

Parkgest consta de dos partes, el programa de reconocimiento de matrícula, que lo único que hace es extraer la matrícula de una foto para poder insertar un registro en la base de datos cuando un coche entra o sale del parking. Y por otro lado tenemos la interfaz gráfica, en la que podemos gestionar clientes, podemos relacionar un coche con un cliente además de hacer facturas y poder introducir registros en la base de datos.

2.2 DESCRIPCIÓN DE LAS ENTIDADES PRINCIPALES.

Las entidades son las siguientes (todas estas entidades tienen un id que actúa como clave primaria, no se pondrá para no ser redundante):

Tarifa, que define la cantidad de dinero que cada cliente tendrá que pagar por hora.

Atributos:

-nombre: nombre de la tarifa.

-preciomensual: coste que tiene al mes la tarifa.

-preciohora: coste que va a tener que pagar el cliente por hora.

Cliente, que son las personas que contratan estas tarifas.

Atrtibutos:

-nombre, email, teléfono, dni: datos de identificación y fe contacto del cliente.

idTarifa: relación a la tarifa asignada.

idLocalidad: relación a la localidad asignada.

Vehículo, que es cada automóvil que aparca.

Atributos:

matrícula: la matrícula del vehículo.

Cliente_Vehiculo: tabla intermedia para ilustrar la relación N:M entre los clientes y los vehículos.

Atributos:

clienteId: cliente al que hace referencia.

vehiculoId: vehículo al que hace referencia.

Usuario: cuentas para poder identificarse y usar el sistema de interfaz gráfica.

Atributos:

nombre: nombre de usuario que tendrá la cuenta.

contraseña: contraseña de usuario que tendrá la cuenta

Registro_Parking: tabla para tener todas las entradas y salidas de vehículos.

Atributos:

idVehiculo: el vehículo que aparca.

hora_entrada, hora_salida: horas en las cuales el vehículo entra y sale.

pagado: si la estancia en el parking está pagada.

2.3 REQUISITOS EN CUANTO A TECNOLOGÍA:

2.3.1 Plataformas en las que funcionará.

Esta aplicación funcionará en cualquier plataforma que tenga el JRE (Java Runtime Enviroment) instalado sin importar el sistema operativo. Es importante que sea la versión 24, en cualquier otro caso, la aplicación puede no abrir.

2.3.2 Requisitos en cuanto al diseño de la interfaz.

Para diseñar la aplicación se ha usado un framework de Java llamado Java Swing el cual es capaz de ejecutar aplicaciones de escritorio de manera fluida y adaptándose a la pantalla.

2.3.3 Requisitos de accesibilidad.

No se ha utilizado ningún requisito concreto de accesibilidad en la interfaz puesto a la simpleza de esta y a las pocas opciones que ofrece Swing de accesibilidad.

2.3.4 Requisitos de rendimiento.

La aplicación podrá funcionar en cualquier ordenador que pueda ejecutar correctamente el Java Runtime Environment, no es una aplicación exigente.

3 HERRAMIENTAS UTILIZADAS

3.1 DESCRIPCIÓN DEL ENTORNO DE DESARROLLO LOCAL Y DE PRODUCCIÓN.

El entorno de desarrollo local ha sido un ordenador de torre que ha funcionado como servidor de bases de datos y también ha estado ejecutando la aplicación, consumiendo la propia base de datos. El entorno de producción deberá de tener la base de datos metida también en el propio servidor y no se podrá conectar a un servidor externo puesto que la configuración de base de datos está hecha para que la BBDD esté en la ip 127.0.0.1.

3.2 FRAMEWORKS Y TECNOLOGÍAS UTILIZADAS EN SERVIDOR Y CLIENTE.

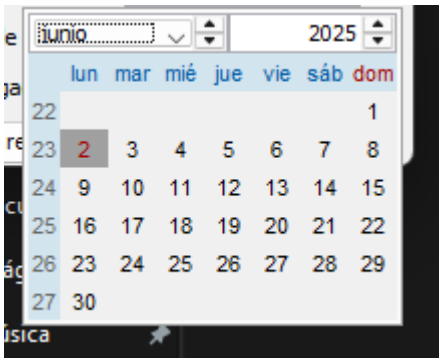
El framework que he usado en su mayoría para toda la gestión de la base de datos es JPA con EclipseLink, no he utilizado Hibernate. Quería utilizar algo que me automatizara la creación de las entidades, por lo que JDBC no es una opción y tampoco quería crear un proyecto con SpringBoot debido a la complejidad que este tendría si decido usar ese framework. JPA era el framework más rápido y útil para este caso. Luego, he usado EclipseLink debido a que desde mi punto de vista es más rápido, más intuitivo y mejor que Hibernate.

Para la interfaz visual he utilizado Swing, que es un framework de Java que te permite hacer interfaces gráficas simples que se pueden conectar con una base de datos.

He usado Maven para la importación de las librerías al proyecto con el fin de hacerlo más rápido y de que al llevarlo a otro ordenador fuese sencillo de desplegar.

He usado una librería de pdf llamada itextpdf para la creación de las facturas. Es el framework más usado con este fin, por lo que era en el que más documentación había.

He usado una librería de calendario en Swing llamada jcalendar para poder especificar mejor las fechas a la hora de introducir el registro de un parking.



Y por último he usado una librería para poder enviar peticiones HTTP a la API que me permite realizar el reconocimiento de matrículas, además de otra para leer los JSON de manera más correcta y rápida. Estas librerías son de org.apache.httpcomponents.

3.3 Herramientas para la gestión del código fuente (Git).

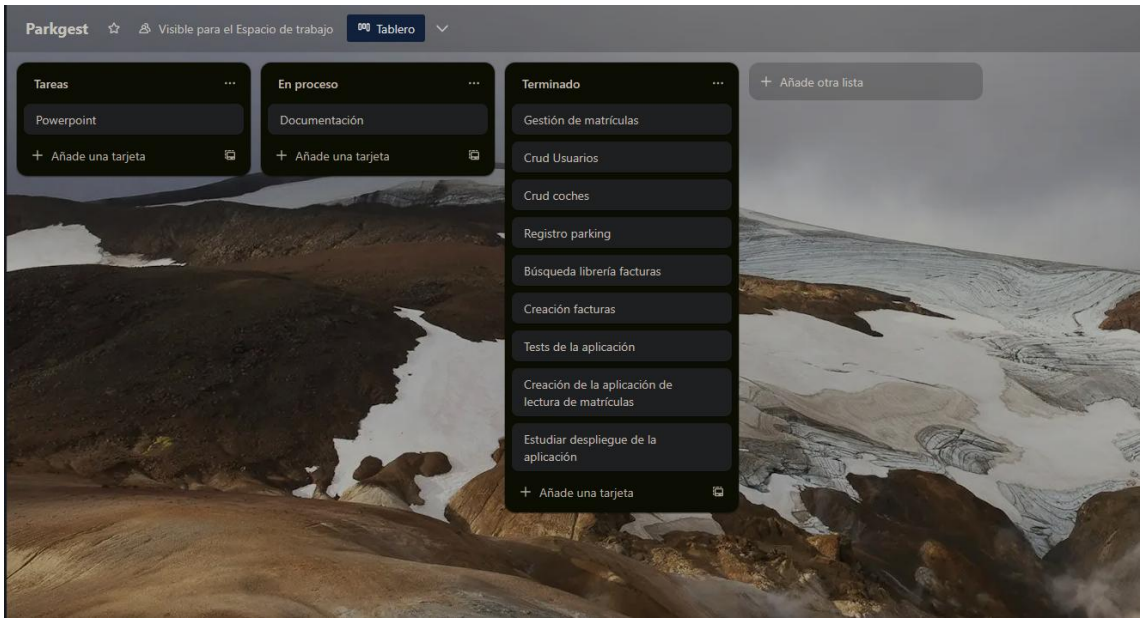
Para la gestión del código fuente he usado Git el cual lo he conectado con un repositorio remoto en Github con el fin de poder llevar el código más rápido entre los distintos equipos que he usado para programar y hacer pruebas.

3.4 Herramientas para la gestión de la documentación (Google Drive).

Para la creación de la documentación he usado tanto Google Drive para tener otro sitio donde guardar mi código además de Word para poder escribirla.

3.5 Herramientas para la gestión del proyecto (Trello, Slack...).

Para gestionar el proyecto he usado Trello, una herramienta que te permite dividir una tarea grande en pequeñas tareas y organizarlas en listas, con el fin de tener una visión más directa de por donde iba, de lo que me faltaba y de lo que había terminado.



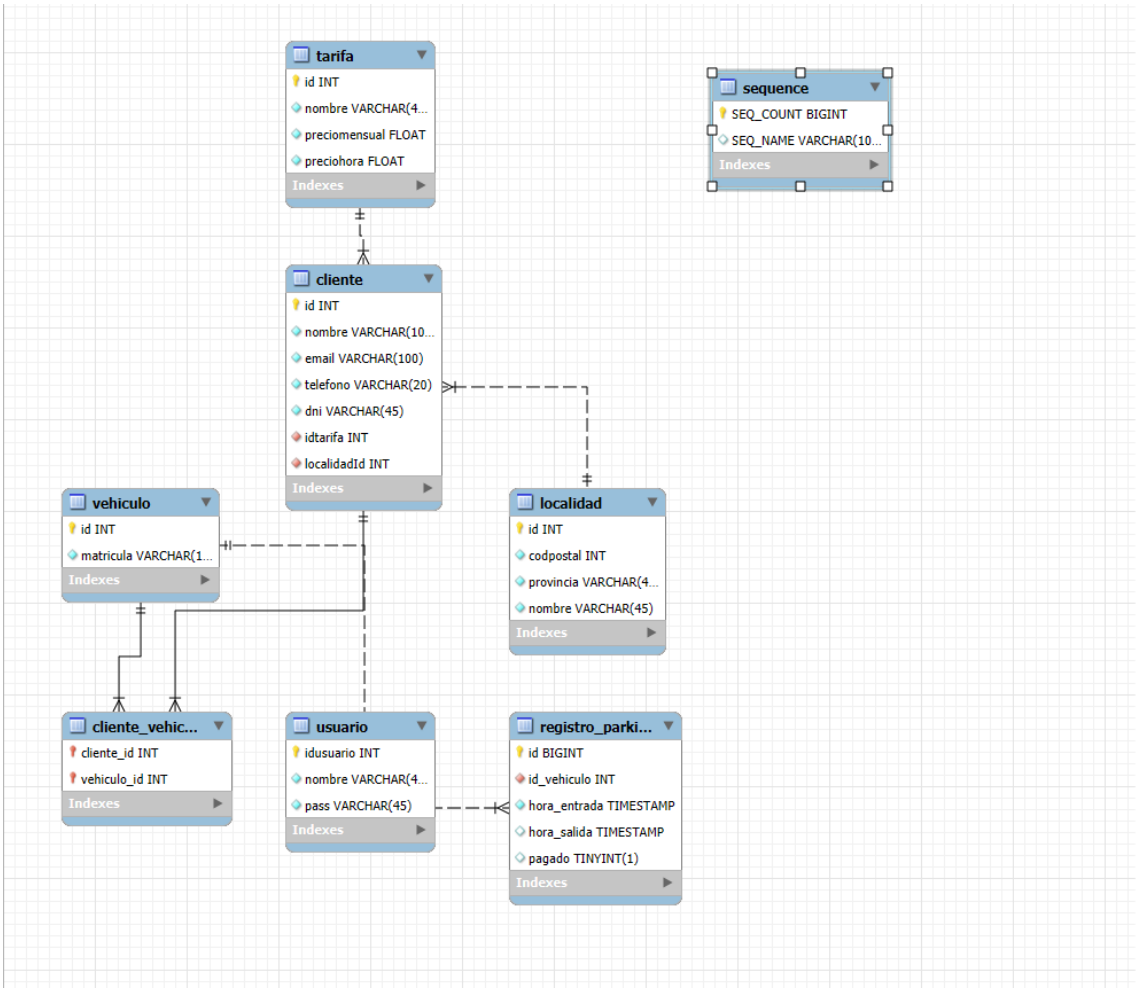
3.6 OTRAS HERRAMIENTAS UTILIZADAS (PARA DISEÑO GRÁFICO, EDICIÓN, PRUEBAS...)

No he usado ninguna herramienta más.

4 DISEÑO.

4.1 DISEÑO DE DATOS.

4.1.1 Diseño conceptual de la base de datos.



4.1.2 Diseño lógico de la base de datos.

1. Tabla: tarifa

Columna	Tipo	Restricciones	Descripción
id	INT	PRIMARY KEY	Identificador único
nombre	VARCHAR(40)	NOT NULL	Nombre de la tarifa
predomenual	FLOAT		Precio domicilio mensual
precohora	FLOAT		Precio por hora

---

2. Tabla: cliente

Columna	Tipo	Restricciones	Descripción
id	INT	PRIMARY KEY	Identificador único
nombre	VARCHAR(100)	NOT NULL	Nombre completo

Columna	Tipo	Restricciones	Descripción
email	VARCHAR(100)		Correo electrónico
telefono	VARCHAR(20)		Teléfono de contacto
doi	VARCHAR(45)		Documento de identidad
idtarifa	INT	FOREIGN KEY (tarifa.id)	Tarifa asociada
localidadId	INT	FOREIGN KEY (localidad.id)	Ubicación del cliente

---

3. Tabla: vehículo

Columna	Tipo	Restricciones	Descripción
id	INT	PRIMARY KEY	Identificador único
matrícula	VARCHAR(15)	NOT NULL UNIQUE	Matrícula del vehículo

---

4. Tabla: cliente_vehiculo (Tabla de unión)

Columna	Tipo	Restricciones	Descripción
cliente_id	INT	PRIMARY KEY, FOREIGN KEY (cliente.id)	Relación con cliente
vehículo_id	INT	PRIMARY KEY, FOREIGN KEY (vehículo.id)	Relación con vehículo

---

5. Tabla: usuario

Columna	Tipo	Restricciones	Descripción
idusuario	INT	PRIMARY KEY	Identificador único
nombre	VARCHAR(40)	NOT NULL	Nombre de usuario
pass	VARCHAR(45)	NOT NULL	Contraseña (hash)

---

6. Tabla: localidad

Columna	Tipo	Restricciones	Descripción
id	INT	PRIMARY KEY	Identificador único
codpostal	INT	NOT NULL	Código postal
provincia	VARCHAR(40)	NOT NULL	Provincia

Columna	Tipo	Restricciones	Descripción
nombre	VARCHAR(45)	NOT NULL	Nombre de la localidad

---

7. Tabla: registro_parking

Columna	Tipo	Restricciones	Descripción
id	BIGINT	PRIMARY KEY	Identificador único
id_vehiculo	INT	FOREIGN KEY (vehículo.id)	Vehículo relacionado
hora_entrada	TIMESTAMP	NOT NULL	Hora de entrada al parking
hora_salida	TIMESTAMP		Hora de salida del parking
pagado	TINYINT(1)	DEFAULT 0	Estado de pago (0/1)

Relaciones Clave

1. **Cliente → Tarifa** (1:N):
   cliente.idtarifa referencia tarifa.id
2. **Cliente → Localidad** (1:N):
   cliente.localidadId referencia localidad.id
3. **Cliente ↔ Vehículo** (M:N):
   Resuelta mediante tabla de unión cliente_vehiculo
4. **Registro → Vehículo** (1:N):
   registro_parking.id_vehiculo referencia vehículo.id

4.2 DISEÑO DE LA APLICACIÓN.

4.2.1 Diseño de backend.

El backend se ha realizado por medio de Eclipse Link, por lo tanto, he utilizado un fichero de configuración “persistence.xml”, en el que hay que especificar la IP de la BBDD, el usuario, la contraseña, las entidades que va a tener que gestionar y un nombre a la unidad de persistencia como se adjunta en la foto de abajo.

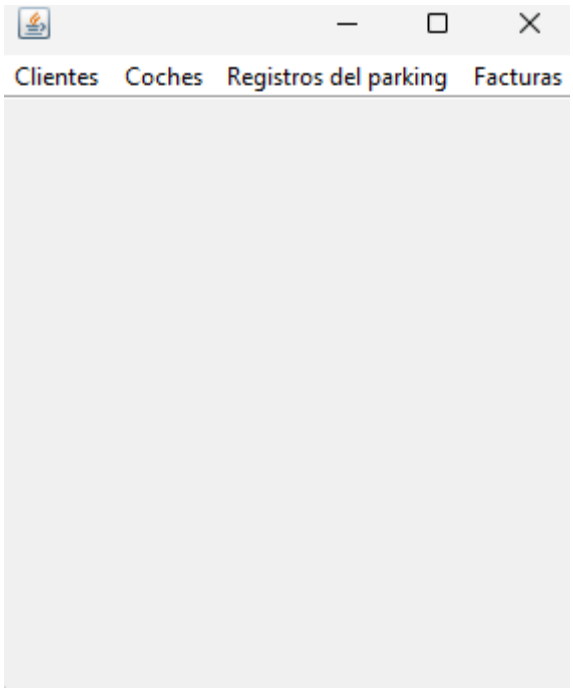
```
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd (xsi:schemaLocation with catalog)
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
      xmlns="http://xmlns.jcp.org/xml/ns/persistence"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
    <persistence-unit name="parkgest">
        <class>model.Cliente</class>
        <class>model.Localidad</class>
        <class>model.RegistroParking</class>
        <class>model.Tarifa</class>
        <class>model.Vehiculo</class>
        <class>model.Usuario</class>
        <properties>
            <property name="javax.persistence.jdbc.driver"
                value="com.mysql.cj.jdbc.Driver" />
            <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://127.0.0.1:3306/parking?serverTimezone=Europe/Madrid" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password"
                value="root" />
            <property name="eclipselink.jpa.uppercase-column-names"
                value="true" />
            <property name="eclipselink.logging.level" value="FINE" />
        </properties>
    </persistence-unit>
</persistence>
```

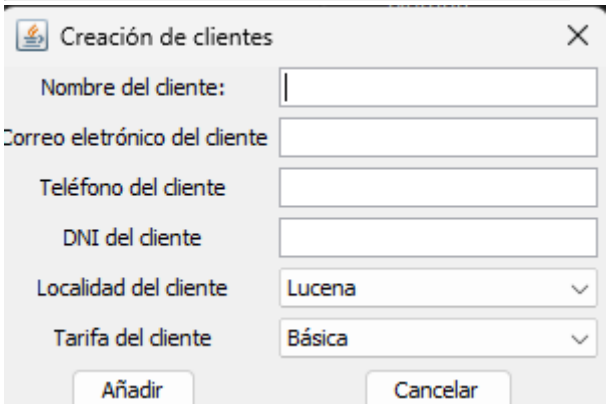
Todos los persistence.xml tienen que estar igual formados y tener un apartado de clases. Lo que puede cambiar es el apartado de propiedades, en el que yo he puesto un nivel de logging alto, para que en el desarrollo yo pueda comprender bien lo que está pasando en la aplicación y poder solucionar mejor los errores que han surgido. Esto al cliente final no le afectará.

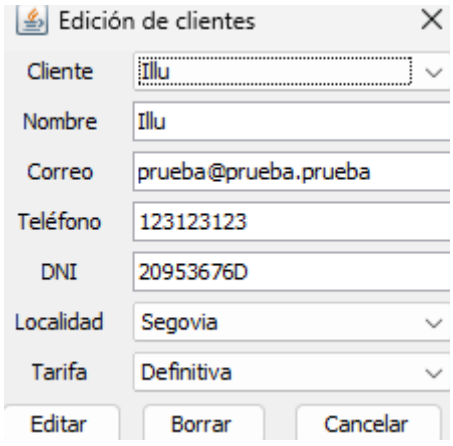
4.2.2 Flujo de la aplicación en los procesos principales.

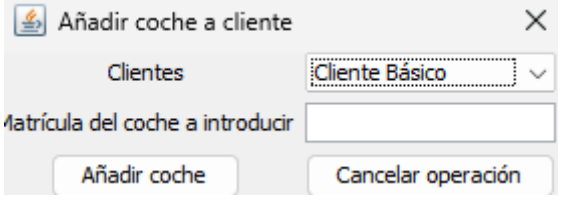
4.2.3 Diseño de la interfaz web/app.

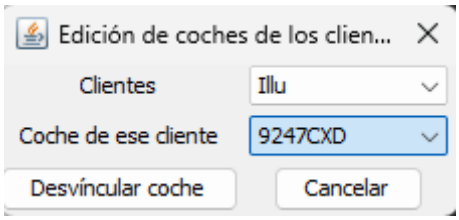
El diseño de la aplicación es muy simple, es una ventana con varios apartados en los que puedes realizar la acción que se describe en el menú.

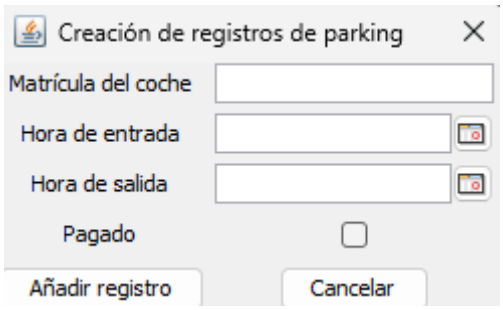

Clientes
Coches
Registros del parking
Facturas

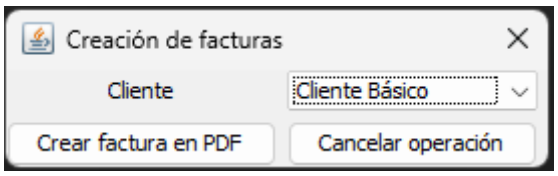

Creación de clientes
Nombre del cliente:
Correo eletrónico del cliente
Teléfono del cliente
DNI del cliente
Localidad del cliente
Lucena
Tarifa del cliente
Básica
Añadir
Cancelar


Edición de clientes
Cliente
Illu
Nombre
Illu
Correo
prueba@prueba.prueba
Teléfono
123123123
DNI
20953676D
Localidad
Segovia
Tarifa
Definitiva
Editar
Borrar
Cancelar


Añadir coche a cliente
Clientes
Cliente Básico
Matrícula del coche a introducir
Añadir coche
Cancelar operación







5 IMPLEMENTACIÓN

5.1 IMPLEMENTACIÓN DE LA BD (EXPORTAR BASE DE DATOS EN FICHERO MYSQL).

-- MySQL dump 10.13 Distrib 8.0.41, for Win64 (x86_64)

--

-- Host: localhost   Database: parking

-- --

-- Server version	8.0.41

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;

/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;

/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;

/*!50503 SET NAMES utf8 */;

/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;

/*!40103 SET TIME_ZONE='+00:00' */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;

/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;

```
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
*/;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;


--
-- Table structure for table `cliente`
--


DROP TABLE IF EXISTS `cliente`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `cliente` (
  `id` int NOT NULL,
  `nombre` varchar(100) NOT NULL,
  `email` varchar(100) NOT NULL,
  `telefono` varchar(20) NOT NULL,
  `dni` varchar(45) NOT NULL,
  `idtarifa` int NOT NULL,
  `localidadId` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `dni_UNIQUE` (`dni`),
  KEY `fkTarifa_idx` (`idtarifa`),
  KEY `fkLocalidad_idx` (`localidadId`),
  CONSTRAINT `fkLocalidad` FOREIGN KEY (`localidadId`) REFERENCES `localidad` (`id`),
  CONSTRAINT `fkTarifa` FOREIGN KEY (`idtarifa`) REFERENCES `tarifa` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;


--
-- Table structure for table `cliente_vehiculo`
--
```

```
DROP TABLE IF EXISTS `cliente_vehiculo`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `cliente_vehiculo` (
  `cliente_id` int NOT NULL,
  `vehiculo_id` int NOT NULL,
  PRIMARY KEY (`cliente_id`,`vehiculo_id`),
  KEY `vehiculo_fk` (`vehiculo_id`),
  CONSTRAINT `cliente_vehiculo_ibfk_1` FOREIGN KEY (`cliente_id`) REFERENCES `cliente` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `vehiculo_fk` FOREIGN KEY (`vehiculo_id`) REFERENCES `vehiculo` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `localidad`
--

DROP TABLE IF EXISTS `localidad`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `localidad` (
  `id` int NOT NULL,
  `codpostal` int NOT NULL,
  `provincia` varchar(45) NOT NULL,
  `nombre` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `codpostal_UNIQUE` (`codpostal`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
/*!40101 SET character_set_client = @saved_cs_client */;


--
-- Table structure for table `registro_parking`
--

DROP TABLE IF EXISTS `registro_parking`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `registro_parking` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `id_vehiculo` int NOT NULL,
  `hora_entrada` timestamp NOT NULL,
  `hora_salida` timestamp NULL DEFAULT NULL,
  `pagado` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`id`),
  UNIQUE KEY `id` (`id`),
  KEY `registro_parking_ibfk_1` (`id_vehiculo`),
  CONSTRAINT `registro_parking_ibfk_1` FOREIGN KEY (`id_vehiculo`) REFERENCES `vehiculo`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=2083 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;


--
-- Table structure for table `sequence`
--

DROP TABLE IF EXISTS `sequence`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `sequence` (
```

```
  `SEQ_COUNT` bigint NOT NULL,
  `SEQ_NAME` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`SEQ_COUNT`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `tarifa`
--

DROP TABLE IF EXISTS `tarifa`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `tarifa` (
  `id` int NOT NULL,
  `nombre` varchar(45) NOT NULL,
  `preciomensual` float NOT NULL,
  `preciohora` float NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `nombre_UNIQUE` (`nombre`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `usuario`
--

DROP TABLE IF EXISTS `usuario`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
```

```
CREATE TABLE `usuario` (
  `idusuario` int NOT NULL,
  `nombre` varchar(45) NOT NULL,
  `pass` varchar(45) NOT NULL,
  PRIMARY KEY (`idusuario`),
  UNIQUE KEY `nombre_UNIQUE` (`nombre`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `vehiculo`
--

DROP TABLE IF EXISTS `vehiculo`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `vehiculo` (
  `id` int NOT NULL,
  `matricula` varchar(10) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `matricula_UNIQUE` (`matricula`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
```

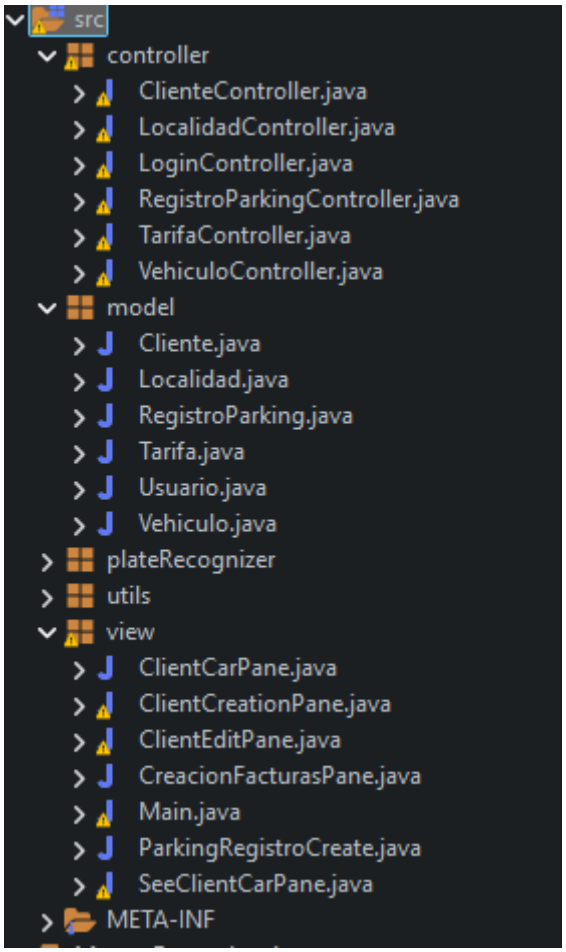
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;

/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

5.2 DESCRIPCIÓN DE LA ESTRUCTURA DE FICHEROS Y CARPETAS DEL PROYECTO.

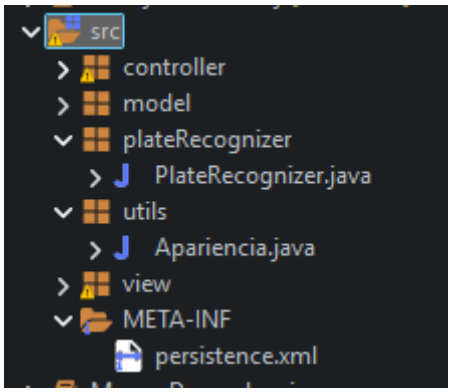
La estructura que he usado para el proyecto es un sistema MVC (Modelo Vista Controlador), por lo que tendremos tres carpetas importantes en el proyecto, una con las entidades, otra con los controladores y una última con las vistas.

Cabe destacar que un modelo es una entidad de una base de datos, una vista es una interfaz en la que el usuario puede realizar acciones y un controlador es una clase de Java que te permite operar con esos modelos para poder hacer operaciones sobre la base de datos, como obtener todos los registros de una tabla, insertar, eliminar o actualizar.



Las vistas han sido creadas con un framework de Java llamado Swing que sirve para hacer interfaces gráficas simples y poder conectarlas con una base de datos para crear una aplicación de escritorio.

Además de esas 3 carpetas importantes, hay 3 más.



La primera es META-INF, en el que está el persistence.xml que es el archivo de configuración para poder comunicarnos con la BBDD.

Luego tenemos la carpeta utils en la que solo tengo un archivo para poder cambiar la apariencia del programa y que los botones y las barras tengan una apariencia un poco más actualizada y sean las barras de cualquier programa de Windows.

Y luego en la carpeta plateRecognizer está el programa que he creado para reconocer las matrículas de las distintas fotos de los coches que pasarán por el parking. Es un programa externo a la interfaz de usuario.

5.3 DESCRIPCIÓN DE LOS FICHEROS DE CONFIGURACIÓN DE LA APLICACIÓN.

Para la aplicación solo he usado el persistence.xml. En el que se especifica la url de la base de datos, el usuario de esta, la base de datos sobre la que vamos a operar, las clases que están en la carpeta de model para que el programa sepa que esas entidades están en la base de datos y por último distintas propiedades como el nivel de logging o el nombre que tendrá la unidad de persistencia. Esta última es importante, porque si no lo ponemos bien en los controladores, habrá errores y el programa no funcionará.

```
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd (xsi:schemaLocation with catalog)
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
    <persistence-unit name="parkgest">
        <class>model.Cliente</class>
        <class>model.Localidad</class>
        <class>model.RegistroParking</class>
        <class>model.Tarifa</class>
        <class>model.Vehiculo</class>
        <class>model.Usuario</class>
        <properties>
            <property name="javax.persistence.jdbc.driver"
                value="com.mysql.cj.jdbc.Driver" />
            <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://127.0.0.1:3306/parking?serverTimezone=Europe/Madrid" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password"
                value="root" />
            <property name="eclipselink.jpa.uppercase-column-names"
                value="true" />
            <property name="eclipselink.logging.level" value="FINE" />
        </properties>
    </persistence-unit>
</persistence>
```

6 DESPLIEGUE DE LA APLICACIÓN.

6.1 INSTRUCCIONES PARA EL DESPLIEGUE DE LA APLICACIÓN.

Hay dos aplicaciones distintas, pero para desplegar cualquier de las dos tienes que tener una base de datos en el ordenador, es decir, en el localhost que se llame parking con el SQL anteriormente proporcionado, la BBDD se tiene que llamar parking y el nombre y el usuario tiene que ser root.

Simplemente tienes que crear EN EL ESCRITORIO una carpeta llamada facturas para poder crear facturas con el programa que tiene una aplicación gráfica.

Y para usar el programa que reconoce matrículas tienes que tener dos carpetas en el escritorio, una que se llama fotosEntrada y otra fotosSalida, con eso, abres los dos archivos .jar.

7 CONCLUSIONES.

7.1 DIFICULTADES ENCONTRADAS EN LA REALIZACIÓN DEL PROYECTO.

En la creación del proyecto he tenido la dificultad de crear una factura con Java, porque no sabía como crear un PDF, así que he tenido que utilizar una librería que tenía alguna que otra complicación y que me ha costado hacer que funcionase.

A la hora de reconocer las matrículas he probado distintos frameworks para hacer que la lectura fuese en local y no tener que enviarlo a una API externa que en cualquier momento puede fallar, pero por desgracia no se ha podido, porque esos frameworks que funcionan en local están desactualizados, entonces al final he tenido que usar una API externa.

Matrícula	Horas	Precio por hora	Total
1313PPPP	1	4.0€	4€
RK828AG	0,16	4.0€	0,64€
3333PPPP	3	4.0€	12€

Cuando tuve hacer la factura, para que me saliera en la tabla de esta una fila con cada matrícula he tenido que usar un Map, cosa que no sabía que era y por la que estuve atascado durante varios días, porque no sabía bien cómo hacerlo.

También EclipseLink me daba fallo a la hora de insertar y/o eliminar un registro porque no tenía creada la tabla sequence, por eso estuve parado un par de días, porque no hacía nada en la base de datos y no entendía por qué.

Por último, he tenido bastantes problemas con desplegar la propia aplicación debido a que no tenía las librerías bien implementadas y el programa no se desplegaba bien, por lo que siempre daba distintos fallos. Tras algunos días buscando y haciendo pruebas, al final lo he podido desplegar usando Maven, pero esa complicación ha sido bastante grande.

7.2 CONOCIMIENTOS ADQUIRIDOS DURANTE EL DESARROLLO DEL PROYECTO.

-Creación de PDFs con Java.

-Envío de peticiones HTTP a una API y recibo de JSON de una API de internet.

-Uso de la clase Map.

-Compilación y despliegue de un programa de Java con Maven.

-Uso de la tabla Sequence en JPA.

-Profundización en el uso de Swing.

-Gestión de fotos para enviarlas por una petición HTTP.

-Profundización en el MVC.

7.3 FUTURAS AMPLIACIONES O MEJORAS DEL PROYECTO.

-Un sistema de reconocimiento de matrículas que esté en local sin tener que recurrir a ninguna API, para tener las lecturas de matrículas gratuitas que queramos y no tener que depender del internet.

-Un sistema de historial de registro de cada coche.

-Poder otorgar determinadas plazas del párking a determinados clientes que tienen una suscripción.

-Poder enviar esas facturas PDF por correo a cada cliente.

8 BIBLIOGRAFÍA.

-Docs de Java

-Docs de Swing

-Docs de EclipseLink

-Docs de itextpdf

-Docs de ApacheHTTPComponents

-Docs de la API de PlateRecognizer

-ChatGPT

-DeepSeek

9 ANEXOS.