

Documentación del Proyecto TeslaFXStore

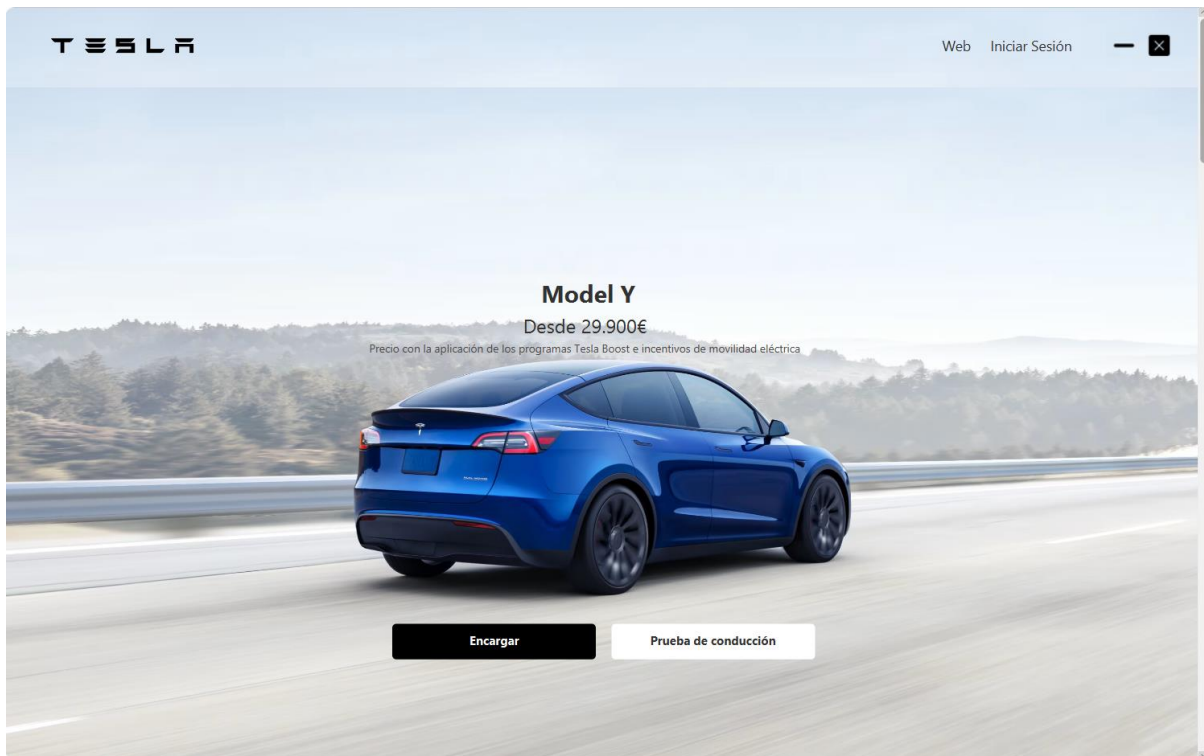
Índice

- 1. Introducción: pág.2**
 - 1.1 Propósito del proyecto:
 - 1.2 Contexto académico
- 2. Tecnologías Utilizadas: pág.3**
 - 2.1 Lenguaje de programación
 - 2.2 Framework
 - 2.3 Estilos y animaciones
 - 2.4 Base de datos
 - 2.5 IDE
- 3. Estructura del Proyecto: pág.5**
 - 3.1 Modelo-Vista-Controlador (MVC)
 - 3.2 Paquetes principales
 - 3.3 Fuentes adicionales
 - 3.4 Visualización estructura
- 4. Descripción de la Aplicación: pág.7**
 - 4.1 Modo Cliente
 - 4.2 Modo Administrador
- 5. Ventanas de la Aplicación: pág.8**
 - 5.1 Componentes comunes
 - 5.2 Ventana Principal
 - 5.3 Ventana Crear Cuenta
 - 5.4 Ventana de Login
 - 5.5 Ventana Mis Datos
 - 5.6 Ventana de Compra
 - 5.7 Ventana de Administración
- 6. Colecciones de la Base de Datos: pág.17**
 - 6.1 customer
 - 6.2 customer_administrator
 - 6.3 vehicle
 - 6.4 vehicle_stock
- 7. Validaciones Implementadas: pág.18**
 - 7.1 Validaciones en cliente
 - 7.2 Validaciones en administrador
- 8. Dificultades y Lecciones Aprendidas: pág.19**
- 9. Opciones de Herramientas e IDE: pág.20**
 - 9.1 Evaluación de NetBeans y Eclipse
 - 9.2 Ventajas de NetBeans para JavaFX
- 10. Aprendizaje de Nuevas Tecnologías: pág.21**
 - 10.1 JavaFX
 - 10.2 SceneBuilder
 - 10.3 CSS para JavaFX
 - 10.4 FXML
- 11. Código adicional: pág.22**
 - 11.1 Manejo de celdas personalizadas en tablas con javaFX
 - 11.2 Carrusel automático de imágenes con JavaFX
 - 11.3 Mostrar Tooltip
 - 11.4 If Ternario para validar modelo
- 12. Recursos utilizados: pág.25**
 - 12.1 Curso JavaFX en Youtube
 - 12.2 Documentación oficial JavaFX
 - 12.3 ChatGPT
- 13. Conclusión: pág.27**

1. Introducción

1.1 Propósito del proyecto

Este documento describe el proyecto "TeslaFXStore", desarrollado como parte del Trabajo Final de Grado (TFG) del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Multiplataforma (DAM). El propósito principal de este proyecto es crear una aplicación de escritorio para la gestión y simulación de compras en una tienda Tesla, implementando funcionalidades tanto para clientes como para administradores.



1.2 Contexto académico

El proyecto tiene como objetivo principal demostrar el dominio de tecnologías modernas, diseño orientado al cliente y administrador, y la integración de bases de datos no relacionales.

2. Tecnologías Utilizadas

2.1 Lenguaje de Programación: Java

Java es un lenguaje versátil y ampliamente utilizado en el desarrollo de aplicaciones de escritorio. Su orientación a objetos permite un diseño modular y escalable, facilitando la implementación del patrón Modelo-Vista-Controlador (MVC). Además, cuenta con una extensa comunidad de desarrolladores y una amplia variedad de bibliotecas que simplifican tareas complejas, como la conexión a bases de datos y el manejo de interfaces gráficas. Su independencia de plataforma, gracias a la máquina virtual de Java (JVM), garantiza que la aplicación sea compatible con diferentes sistemas operativos.



2.2 Framework para Interfaces Gráficas: JavaFX

JavaFX es una evolución moderna para el desarrollo de interfaces gráficas en Java, ofreciendo capacidades avanzadas como animaciones, controles estilizados, y una estructura flexible basada en el uso de archivos **.fxml**. A diferencia de Swing, JavaFX permite un diseño más intuitivo y profesional, con soporte nativo para CSS y una integración simplificada con SceneBuilder, que facilita la creación visual de interfaces. Su capacidad para manejar gráficos complejos y su orientación hacia experiencias de usuario modernas lo convierten en la elección ideal para este tipo de proyecto.



2.3 Estilos y Animaciones: CSS

La integración de CSS en JavaFX permite separar claramente la lógica de la aplicación y el diseño visual, siguiendo las mejores prácticas en desarrollo de software. Aunque CSS para JavaFX tiene diferencias con el CSS utilizado en desarrollo web (por ejemplo, prefijos como **-fx-**), sigue siendo una herramienta poderosa para personalizar estilos, añadir transiciones y mejorar la experiencia visual del usuario. Esto resulta esencial para proyectos como **TeslaFXStore**, donde una interfaz atractiva y profesional es clave para simular una experiencia de tienda de alta gama.



2.4 Base de Datos: MongoDB

MongoDB, una base de datos no relacional, es ideal para aplicaciones como **TeslaFXStore** que requieren una estructura flexible de datos. Su modelo basado en documentos permite almacenar información compleja, como detalles de vehículos y personalización de clientes, sin la rigidez de las bases de datos relacionales. Además, MongoDB es altamente escalable, lo que asegura que la aplicación pueda manejar un creciente volumen de datos en el futuro.



2.5 IDE Utilizado: NetBeans

NetBeans ofrece un entorno de desarrollo completo para proyectos JavaFX, facilita la creación y gestión de proyectos JavaFX, ofreciendo plantillas predefinidas y soporte para configuraciones avanzadas. Incluye soporte para archivos **.fxml**, lo que facilita la personalización de interfaces gráficas. La elección de NetBeans permitió agilizar el desarrollo, integrar dependencias de forma sencilla y aprovechar un entorno diseñado específicamente para proyectos JavaFX.



3. Estructura del Proyecto

La estructura del proyecto sigue el patrón Modelo-Vista-Controlador (MVC), que separa claramente la lógica de la aplicación, la interfaz de usuario y el acceso a datos. A continuación, se describen las carpetas principales:

3.1 Source Packages

- **<default package>**: Contiene la clase `module-info`, que define el módulo principal de la aplicación **TeslaFXStore**. En este archivo se especifican las dependencias externas del proyecto (como JavaFX y MongoDB) y se definen los paquetes que se exportan o se abren para su uso por otros módulos o frameworks.
- **app**: Incluye la clase principal de la aplicación, responsable de inicializar y ejecutar el programa.
- **controller**: Este paquete contiene los controladores encargados de gestionar la lógica de negocio asociada a las ventanas. Facilitan la comunicación entre la vista (archivos FXML) y el modelo.
 - **controller.administrator**: Subpaquete que organiza los controladores específicos para las funcionalidades del administrador, como la gestión de clientes, stock y vehículos.
- **dao**: Representa la capa de acceso a datos. Aquí se gestionan las operaciones relacionadas con MongoDB, como la creación, lectura, actualización y eliminación de datos.
- **model / model.vehicle**: Contiene las clases que definen las entidades principales del negocio. Por ejemplo:
 - **Customer**: Representa a los clientes.
 - **Vehicle**: Define las propiedades y comportamientos relacionados con los vehículos.
- **util**: Este paquete agrupa clases auxiliares que optimizan el desarrollo. Por ejemplo:
 - Métodos para leer entradas de teclado con validación de tipo.
 - Métodos auxiliares para reducción de código.
- **vehiclePrices**: Incluye clases para asignar precios y tarifas a los vehículos. Facilita la gestión de costes dentro de la aplicación.

3.2 Other Sources

- **view**: Contiene los archivos **.fxml**, que definen las interfaces gráficas del usuario. Estos archivos separan la estructura visual de la lógica del programa, siguiendo las mejores prácticas de desarrollo.
- **icons**: Este directorio almacena los íconos utilizados en la aplicación, como botones o elementos decorativos.

- **images:** Contiene las imágenes que enriquecen la experiencia visual del usuario, como fondos o gráficos relevantes.
- **styles:** Agrupa los archivos CSS que proporcionan estilos personalizados a los componentes de la aplicación. Permiten crear una experiencia de usuario coherente y estética.

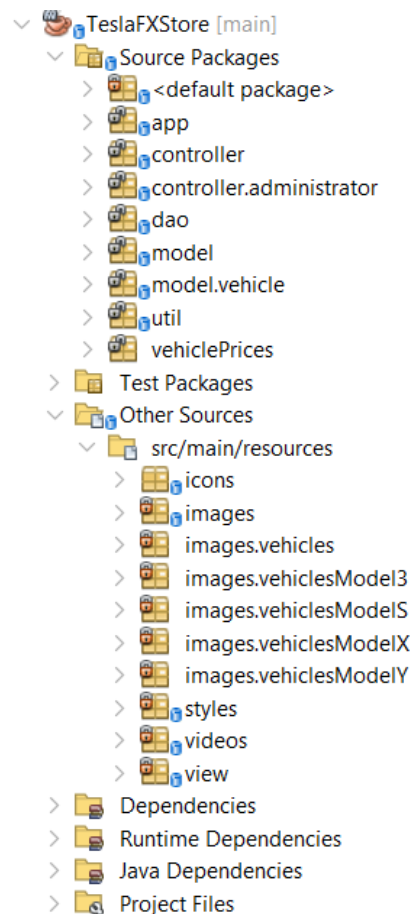
3.3 Visualización estructura

1. Source Packages:

- **<default package>:** Contiene La clase module-info define el módulo de la aplicación TeslaFXStore.
- **app:** Contiene la clase principal para ejecutar la aplicación.
- **controller:** Incluye los controladores para manejar la lógica de las ventanas.
- **controller.administrator:** Controladores específicos para las funcionalidades del administrador.
- **dao:** Capa de acceso a datos, utilizada para interactuar con MongoDB.
- **model / model.vehicle:** Define las clases que representan las entidades del negocio, como Customer y Vehicle.
- **util:** Contiene clases auxiliares, para leer de teclado con comprobación de tipo de datos.
- **vehiclePrices:** Contiene clase para asignar precios a los vehículos y tarifas.

2. Other Sources:

- **view:** Archivos .fxml para las interfaces gráficas.
- **icons:** Contiene los iconos utilizados en la aplicación.
- **images:** Contiene las imágenes utilizadas en la aplicación.
- **styles:** Contiene los archivos css para los estilos de los componentes de la aplicación.

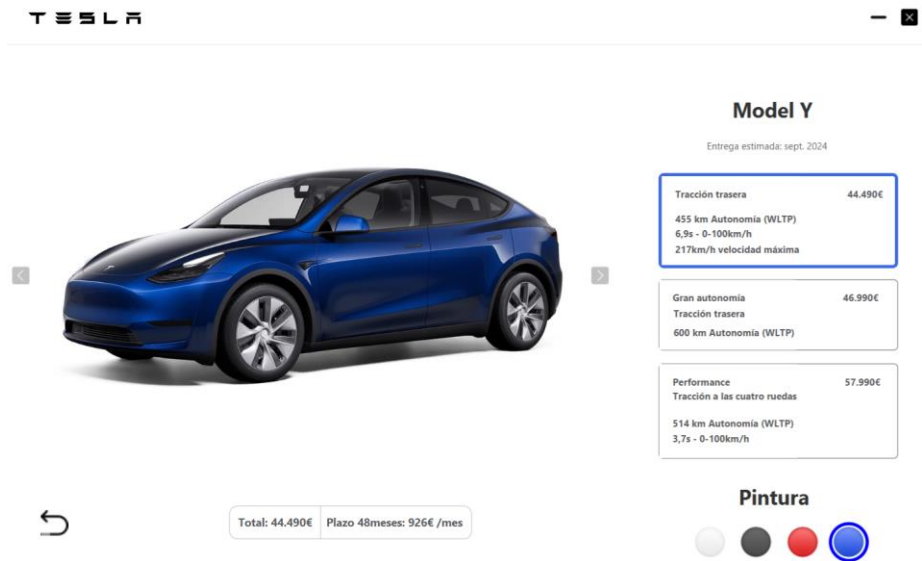


4. Descripción de la Aplicación

4.1 Modo cliente

El modo cliente permite al usuario:

- Explorar vehículos Tesla en una ventana principal con scroll.
- Iniciar sesión, crear una cuenta para realizar compras o eliminar usuario.
- Simular la compra de vehículos seleccionando opciones y tarifas personalizadas.



4.2 Modo administrador

El modo administrador ofrece herramientas para gestionar clientes, vehículos y el stock disponible. Solo se puede acceder mediante cuentas de administrador.



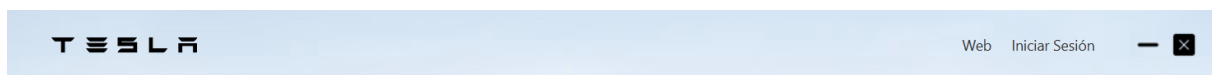
5. Ventanas de la Aplicación

Esta sección describe en detalle las ventanas de la aplicación, incluyendo sus componentes comunes, funcionalidades y especificaciones únicas.

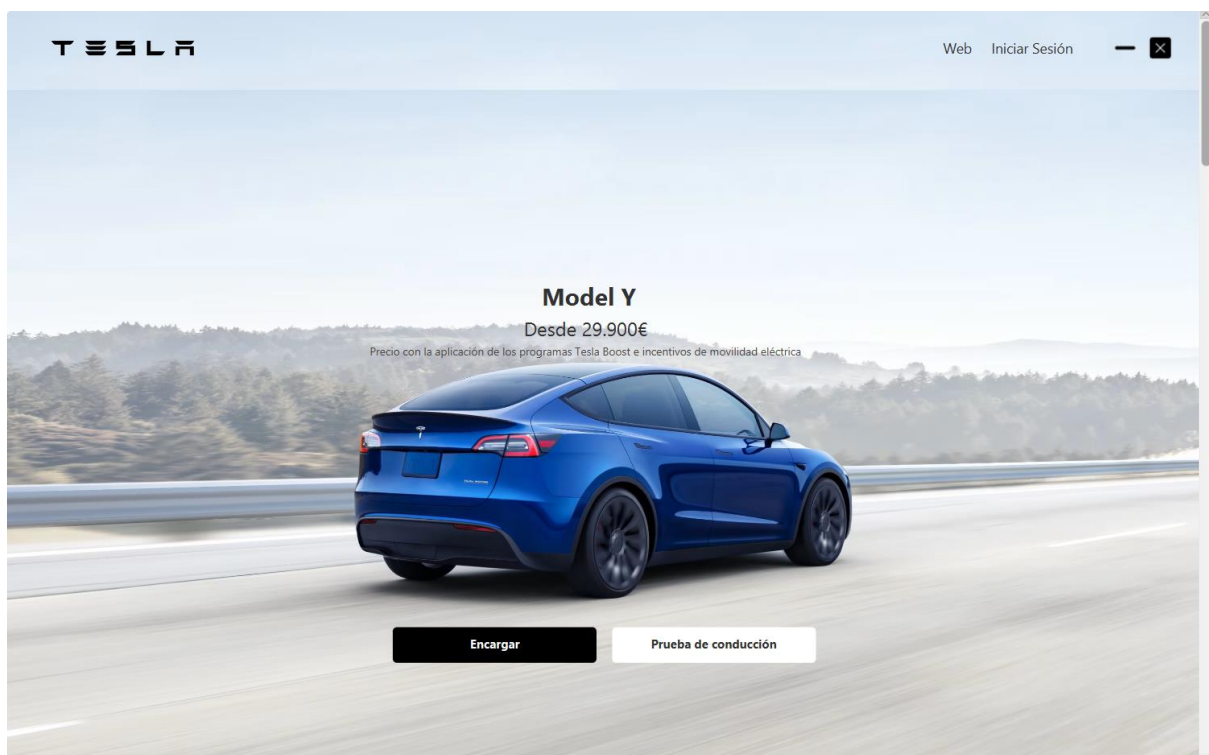
5.1 Componentes comunes

Todas las ventanas comparten los siguientes componentes:

- **Cabecera con el logo de Tesla:** Al hacer clic en el logo, se redirige a la Ventana Principal.
- **Botón de minimizar:** Permite minimizar la ventana (Icono).
- **Botón de cerrar:** Cierra la aplicación (Icono).



5.2 Ventana principal



Propósito: Permite al usuario cerrar sesión, encargar vehículo y acceder a la página web oficial, está dividida en las siguientes secciones:

- **Cabecera:** Contiene una barra de menú con las opciones:
 - **Web:** Redirige a la página oficial de Tesla.

- **Iniciar sesión:** Permite iniciar sesión o, si ya está iniciada, muestra el nombre del usuario/cliente. Este menú contiene dos **subelementos**:
 - **Mis datos:** Redirige a la Ventana Mis Datos.
 - **Cerrar sesión:** Cierra la sesión activa.
- **Cuerpo principal:**
 - Contiene un componente tipo **scroll** el cual muestra los vehículos disponibles: Model X, Model Y, Model S, y Model 3.
 - Cada vehículo tiene dos **botones**:
 - **Encargar:** Redirige a la Ventana de Compra del modelo seleccionado.
 - **Prueba de conducción:** Abre el enlace a la página oficial de Tesla para agendar una prueba de manejo.

5.3 Ventana Crear Cuenta

Propósito: Registrar a nuevos clientes en la base de datos.

- **Validación en tiempo de ejecución:**
 - **Rellena los campos:** Cuando uno o todos los campos están vacíos
 - **Edad con formato incorrecto:** Edad válida tipo numérico (1-150)
 - **Correo con formato incorrecto:** Correo con formato correcto “solo permite cuentas @gmail.com, @hotmail.com, @outlook.com

Crear Cuenta

Nombre

País

Edad

Correo electrónico

Contraseña

Rellena todos los campos

Crear cuenta

Crear Cuenta

Nombre

País

Edad

Correo electrónico

Contraseña

Formato de edad incorrecto

Crear cuenta

Crear Cuenta

Nombre

País

Edad

Correo electrónico

Contraseña

Formato de correo incorrecto

Crear cuenta

- **Botones:**

- **Crear cuenta:** Crea la cuenta y redirige a la ventana principal con sesión activa.

5.4 Ventana de Login

TESLA

Iniciar Sesión

Correo electrónico

Contraseña

Siguiente

[¿Tiene problemas para iniciar sesión?](#)

☐

Crear Cuenta

Propósito: Permitir a los usuarios acceder a su cuenta o crear una nueva.

- **Botones:**

- **Crear cuenta:** Redirige a la ventana de registro.

- **Mostrar/Ocultar contraseña:** Muestra y oculta el texto del campo contraseña (Icono).
- **Volver atrás:** Regresa a la ventana principal (Icono).
- **Validación en tiempo de ejecución:**
 - **Rellena los campos:** Cuando uno o todos los campos están vacíos
 - **Correo no registrado:** Si el correo no está asociado a ninguna cuenta
 - **Contraseña incorrecta:** La contraseña no es correcta.

Iniciar Sesión

Correo electrónico

Contraseña

Rellena los campos

Siguiente

Iniciar Sesión

Correo electrónico

Contraseña

Correo no registrado

Siguiente

Iniciar Sesión

Correo electrónico

Contraseña

Contraseña incorrecta

Siguiente

5.5 Ventana Mis Datos

TESLA

Eliminar Cuenta



Mis Datos

Nombre

País

Edad

Correo electrónico

Contraseña

Actualizar

Al continuar, entiendo y acepto el [Aviso de privacidad](#) y los [Términos de uso](#) de Tesla para crear una cuenta Tesla



Propósito: Esta ventana permite al usuario mostrar/modificar datos y eliminar cuenta del usuario/cliente.

- **Mostrar sus datos:** Los campos no son editables, no se puede visualizar la contraseña del usuario por seguridad.

- **Modificar sus datos:** Los campos son editables después de confirmar la contraseña "El correo no es editable".

Mis Datos

Nombre

Pepita

País

liechtenstein

Edad

37

Correo electrónico

pepita37@gmail.com

Contraseña

Actualizar

Actualizar datos

Nombre

Pepita

País

liechtenstein

Edad

37

Correo electrónico

pepita37@gmail.com

Contraseña

.....



Guardar

- **Eliminar la cuenta:** Requiere la confirmación de la contraseña para proceder con la eliminación.
- **Validación en tiempo de ejecución:** La contraseña no es correcta.

Confirmación de usuario

Nombre

Pepita

País

liechtenstein

Edad

37

Correo electrónico

pepita37@gmail.com

Contraseña

Siguiente

Introduzca su contraseña para confirmar que esta cuenta le pertenece.

Confirmación de usuario

Nombre

Pepita

País

liechtenstein

Edad

37

Correo electrónico


pepita37@gmail.com


Contraseña


Contraseña incorrecta

Siguiente

5.6 Ventana de Compra











Model S

Entrega estimada: mar. – abr. 2025

| | |
|--|---------|
| Motor Dual Tracción a las cuatro ruedas | 92.990€ |
| 634 km Autonomía (WLTP) 3,02s - 0-100km/h 250km/h velocidad máxima | |

| | |
|---|----------|
| Plaid | 107.990€ |
| 600 km Autonomía (WLTP) 2,1s - 0-100km/h | |


Pintura



Piloto automático mejorado

3800 €

- Navegar en piloto automático



Total: 92.990€

Plazo 48meses: 1.937€ /mes

Propósito: Comprar vehículos de diferentes modelos.

Imágenes y precio final:

- Muestra imágenes del vehículo seleccionado.
- Las imágenes se actualizan automáticamente según el color seleccionado.
- Se actualiza dinámicamente el precio total según las opciones seleccionadas



Scroll con características del vehículo:

- **Componentes** comunes para los cuatro modelos:
 - **Pintura:** Selección del color.
 - **Piloto automático mejorado o capacidad de conducción autónoma total:** Solo se puede seleccionar una de estas opciones o ninguna.
 - **Wall connector:** Puede o no ser seleccionado.
- **Tarifas** específicas según el modelo:
 - **Modelos S y X:** Tarifas disponibles:
 - Dual Motor
 - Plaid
 - **Modelos 3 y Y:** Tarifas disponibles:
 - Tracción trasera
 - Gran autonomía
 - Performance
- **Botón Comprar:** Finaliza la compra y muestra un mensaje de confirmación.

Carga

☐ Wall Connector

535€

Encargue su model S

Entrega estimada: mar. – abr. 2025

Comprar

Model S

Entrega estimada: mar. – abr. 2025

Motor Dual 92.990€
Tracción a las cuatro ruedas
634 km Autonomía (WLTP)
3,02s - 0-100km/h
250km/h velocidad máxima

Plaid 107.990€
600 km Autonomía (WLTP)
2,1s - 0-100km/h

Pintura



Piloto automático mejorado

3800 €

- Navegar en piloto automático
- Cambio de carril automático
- Autopark

En el futuro:

- Summon
- Smart Summon

Añadir

Capacidad de conducción autónoma total

7500 €

- Todas las funcionalidades del Piloto automático básico y Piloto automático mejorado
- Control de semáforos y señales de stop

En el futuro:

- Autogiro en vías urbanas

Añadir

5.6 Ventana de Administración

The screenshot shows a web application interface for Tesla administration. It features a header with the Tesla logo and the user name 'Jhovanny'. The main area is divided into three vertical panels:

- Clientes:** Contains buttons for 'Añadir', 'Modificar', 'Buscar (Email)', 'Buscar (ID)', 'Buscar (País)', 'Mostrar Lista', 'Total Clientes', 'Buscar (Edad)', 'Eliminar', and 'Atrás'. Below these is a text input field labeled 'Introduce el Email'.
- Stock de vehiculos:** Contains buttons for 'Total Stock', 'Modificar', 'Mostrar Lista', and 'Atrás'. Below these is a text input field labeled 'Introduce el Modelo(ModelY/Model3/ModelS/ModelX)'.
- Vehiculos:** Contains buttons for 'Añadir', 'Modificar', 'Buscar (ID)', 'Buscar (Model)', 'Buscar (Cliente)', 'Total vehiculos', 'Total (Model)', 'Mostrar Lista', 'Eliminar', and 'Atrás'. Below these is a text input field labeled 'Introduce el Modelo(ModelY/Model3/ModelS/ModelX)'.

A back arrow icon is located in the bottom left corner of the Clientes panel.

La ventana de administración contiene botones específicos para gestionar cada colección, con las siguientes descripciones:

Clientes

Añadir: Agrega un nuevo cliente.

Modificar: Permite editar los datos de un cliente.

Buscar (Email): Busca un cliente por su email.

Buscar (ID): Busca un cliente por su ID.

Buscar (País): Muestra una tabla con todos los clientes de un país específico.

Mostrar Lista: Muestra una tabla con la lista de todos los clientes.

Total Clientes: Muestra la cantidad total de clientes.

Buscar (Edad): Muestra clientes dentro de un rango de edad.

Clientes

This form shows the 'Buscar (Email)' functionality in detail. It includes the same set of buttons as the main interface. The 'Introduce el Email' field contains the text 'pepita37@gmail.com'. Below this is a large blue button labeled 'Buscar(Email)'. The results are displayed in a list of input fields:

- Id: 672fb3999c3c5f2a80985634
- Nombre: Pepita
- País: liechtenstein
- Edad: 37
- Email: pepita37@gmail.com
- Contraseña: 123456

Stock de Vehículos:

Total Stock: Muestra la suma total del stock de todos los modelos.

Modificar: Permite editar la cantidad de stock de cada modelo.

Mostrar Lista: Muestra un área de texto con el stock de cada modelo.

Atrás: Reinicia los componentes.

Vehículos:

Añadir: Agrega un vehículo.

Modificar: Edita la información de un vehículo.

Buscar (ID): Busca un vehículo por su ID.

Buscar (Modelo): Muestra los vehículos de un modelo específico.

Buscar (Cliente): Muestra los vehículos comprados por un cliente específico.

Total Vehículos: Muestra la cantidad total de vehículos comprados.

Total (Modelo): Muestra la cantidad de vehículos comprados por modelo.

Mostrar Lista: Muestra una tabla con todos los vehículos comprados.

Eliminar: Elimina un vehículo.

Atrás: Reinicia los componentes

Stock de vehiculos

Total Stock

ModificarMostrar Lista

Atrás

Introduce el Modelo(ModelY/Model3/ModelS/ModelX)

- ModelY = 39
- ModelX = 36
- Model3 = 26
- ModelS = 52

Vehiculos

AñadirModificar

Buscar (ID)Buscar (Model)Buscar (Cliente)

Total vehiculosTotal (Model)Mostrar Lista

EliminarAtrás

Introduce el id del vehiculo

6781a94257d164482d67f574

Buscar

Introduce los datos

Id: 6781a94257d164482d67f574

Modelo: Model 3

Email: pepita37@gmail.com

Pintura: Black

Precio: 65.525€

☐ Piloto Automático Mejorado

☐ Tracción Trasera

☒ Capacidad total conducción autonoma

☐ Gran Autonomía

☒ Wall Connector

☒ Performance

6. Colecciones de la base de datos

6.1 Customer

Campos: _id, age, country, customerName, email, password.

6.2 customer_administrator

Campos: _id, email, name, password

6.3 vehicle

- **Campos comunes:**
 - _id: ObjectId
 - email: String
 - paint: String
 - wallConnector: Boolean
 - enhancedAutopilot: Boolean
 - fullSelfDrivingCapability: Boolean
 - model: String
- **Campos específicos según el modelo (X, S):**
 - dualMotor: Boolean
 - plaid: Boolean
- **Campos específicos según el modelo (3, Y):**
 - rearWheelDrive: Boolean
 - highAutonomy: Boolean
 - performance: Boolean

6.4 vehicle_stock

- Contiene un documento con el stock de los modelos disponibles.

```
_id: ObjectId('66dc496292c1444e84d4c5b6')
▼ models : Array (4)
  ▼ 0: Object
    model : "ModelY"
    stock : 39
  ▼ 1: Object
    model : "ModelX"
    stock : 36
  ▼ 2: Object
    model : "Model3"
    stock : 26
  ▼ 3: Object
    model : "ModelS"
    stock : 52
```

7. Validaciones Implementadas

7.1 En cliente

- Impide crear cuentas con correos duplicados.
- Verifica formato de campos como: edad y correo.

Crear Cuenta

| | | |
|--|--|--|
| Nombre Pepita | Iniciar Sesión | Iniciar Sesión |
| País España | Correo electrónico elonmusk@gmail.com | Correo electrónico jhovannyat@gmail.com |
| Edad 30 | Contraseña | Contraseña |
| Correo electrónico pepita37@gmail.com | Contraseña | Contraseña |
| Contraseña | Contraseña | Contraseña |
| Crear cuenta | Siguiente | Siguiente |

7.2 vehicle_stock

No permite añadir vehículos a correos inexistentes.

Introduce los datos

Model X

noexiste@gmail.com

Wall Connector

Motor Dual

Plaid

Actualizar precio

Desactivar campos

Añadir

8. Dificultades y Lecciones aprendidas

Durante el desarrollo de TeslaFXStore, me encontré con múltiples desafíos técnicos que representaron una curva de aprendizaje significativa y enriquecedora:

Uso de módulos en Java y configuración de module-info.java

Cuando inicié el proyecto, desconocía el propósito de los módulos en Java y cómo configurarlos. Inicialmente intenté evitar el uso de `module-info.java`, lo que resultó en numerosos errores de compilación al integrar dependencias externas como JavaFX y MongoDB. Fue necesario dedicar varias horas a investigar y aprender sobre la modularidad en Java, entendiendo cómo declarar dependencias y exportar paquetes en este archivo.

Transición de Swing a JavaFX

Aunque tenía experiencia previa con Java y Swing, decidí optar por JavaFX para incorporar una tecnología más moderna. Esta decisión implicó aprender desde cero el diseño de interfaces gráficas con JavaFX, sus capacidades, y sus limitaciones en comparación con Swing.

Selección del IDE para el desarrollo

Elegir entre Eclipse y NetBeans también fue un proceso deliberado. Tras analizar sus capacidades, me decidí por NetBeans debido a su integración nativa con JavaFX, lo que facilitó el desarrollo del proyecto.

Integración de MongoDB con JavaFX

La integración de MongoDB con JavaFX fue uno de los aspectos más desafiantes del proyecto. Encontré poca documentación actualizada sobre cómo conectar estas tecnologías. Los errores recurrentes, como incompatibilidades entre versiones de dependencias, demandaron investigación y pruebas para resolverlos.

Manejo de dependencias y errores de compilación

La combinación de JavaFX, MongoDB y el sistema modular de Java presentó desafíos constantes. Aprender a manejar dependencias en un entorno modular y configurar correctamente `module-info.java` fue clave para solucionar los problemas de compilación y permitir la funcionalidad deseada.

9. Opciones de Herramientas e IDE

9.1 Evaluación de NetBeans y Eclipse

Se evaluó **NetBeans** y **Eclipse** como posibles IDE para el desarrollo del proyecto. **NetBeans** destaca por su integración nativa con **JavaFX**, permitiendo un trabajo más fluido con archivos `.fxml` y herramientas como **Scene Builder**. También ofrece una gestión más sencilla de proyectos y dependencias, ideal para desarrollos medianos como **TeslaFXStore**. Por otro lado, aunque **Eclipse** es más versátil y extensible, requiere configuraciones adicionales para JavaFX, lo que hizo a **NetBeans** la elección más eficiente.

9.2 Ventajas de NetBeans para JavaFX

NetBeans ofrece múltiples ventajas para trabajar con JavaFX, como:

- Integración directa con **Scene Builder** para diseñar interfaces gráficas.
- Gestión simplificada de módulos y dependencias con `module-info.java`.
- Plantillas preconfiguradas para proyectos JavaFX, ahorrando tiempo en la configuración inicial.
- Soporte nativo para CSS con herramientas específicas para personalizar componentes.

Estas características facilitaron un flujo de desarrollo eficiente y enfocado, optimizando el trabajo en **TeslaFXStore**.

10. Aprendizaje de Nuevas Tecnologías

10.1 JavaFX

JavaFX se utilizó como framework principal para el desarrollo de interfaces gráficas. Su capacidad para crear aplicaciones modernas, con soporte para gráficos avanzados y un enfoque modular, lo convierte en una excelente opción para proyectos robustos como **TeslaFXStore**.

10.2 Scene Builder para diseño de interfaces

Scene Builder permitió diseñar las ventanas de forma visual mediante archivos `.fxml`. Esta herramienta reduce significativamente el tiempo necesario para construir interfaces, ya que permite arrastrar y soltar elementos mientras genera automáticamente el código asociado.

10.3 CSS para personalización de JavaFX

El CSS en JavaFX se utilizó para mejorar la estética de la aplicación. Aunque similar al CSS web, su sintaxis requiere prefijos como `-fx-` y presenta limitaciones específicas para componentes de JavaFX.

10.4 FXML para diseño de interfaces gráficas

FXML es un lenguaje basado en XML para definir la estructura de las interfaces en JavaFX. Aprender FXML supuso entender cómo vincular los elementos visuales al código del controlador, facilitando la separación de la lógica y el diseño de la aplicación. Esto ayudó a mantener una arquitectura clara y modular en el proyecto.

11. Código adicional

11.1 Manejo de celdas personalizadas en tablas con JavaFX

```
this.tableColumn_dualMotor.setCellValueFactory(cellData -> {
    TeslaVehicle vehicle = (TeslaVehicle) cellData.getValue();
    if (vehicle instanceof ModelX) {
        ModelX mx = (ModelX) vehicle;
        return new SimpleBooleanProperty((mx).isDualMotor()).asObject();
    } else if (vehicle instanceof ModelS) {
        return new SimpleBooleanProperty(((ModelS) vehicle).isDualMotor()).asObject();
    } else {
        return null;
    }
});
setEmptyCellHighlight(tableColumn_dualMotor);
private <T> void setEmptyCellHighlight(TableColumn<TeslaVehicle, T> column) {
    column.setCellFactory(col -> new TableCell<TeslaVehicle, T>() {
        @Override
        protected void updateItem(T item, boolean empty) {
            super.updateItem(item, empty);

            if (empty || item == null) {
                setText(null);
                setStyle("-fx-background-color: #7994ff;"); // Estilo para celdas vacías
            } else {
                setText(item.toString());
                // Cambiar el color del texto basado en el contenido
                if ("true".equalsIgnoreCase(item.toString())) {
                    setStyle("-fx-text-fill: blue;"); // Texto azul para "true"
                } else if ("false".equalsIgnoreCase(item.toString())) {
                    setStyle("-fx-text-fill: red;"); // Texto rojo para "false"
                } else {
                    setStyle(""); // Restablecer estilo para otros valores
                }
            }
        }
    });
}
```

En este fragmento de código se muestra cómo agregar una columna personalizada a una tabla en JavaFX utilizando una función lambda para asignar el valor de la celda. Se valida si un vehículo es de tipo `ModelX` o `ModelS` y se muestra un valor booleano para la propiedad `dualMotor`. Además, se incluyen personalizaciones visuales como resaltar celdas vacías y cambiar el color del texto según el valor de la propiedad.

11.2 Carrusel automático de imágenes con JavaFX

```
private void startImageChange(ImageView imageViewVehicle) {  
    // Crear un Timeline que cambia la imagen cada 3 segundos  
    Timeline timeline = new Timeline(  
        new KeyFrame(Duration.seconds(4), e -> {  
            // Cambiar a la siguiente imagen cíclicamente  
            currentIndex = (currentIndex + 1) % images.size();  
            // Cambiar la imagen del ImageView  
            imageViewVehicle.setImage(images.get(currentIndex));  
        })  
    );  
    // Hacer que el Timeline se repita indefinidamente  
    timeline.setCycleCount(Timeline.INDEFINITE);  
    timeline.play(); // Iniciar el Timeline  
}
```

Esta función inicia un cambio de imagen cíclico en un componente `ImageView` cada 4 segundos, utilizando la clase `Timeline` de JavaFX. El índice de la lista de imágenes `images` se actualiza cada vez que se ejecuta el `Timeline`, y la imagen del `ImageView` se cambia en consecuencia. El proceso se repite indefinidamente gracias a la configuración del ciclo del `Timeline`.

11.3 Mostrar tooltip

```
private void mostrarToolTip() {  
    // Crear un nuevo hilo para realizar la espera sin bloquear el hilo principal  
    new Thread(() -> {  
        try {  
            // Mostrar el Label en el hilo principal  
            Platform.runLater(() -> label_tooltip.setVisible(true));  
  
            // Esperar 4 segundos  
            Thread.sleep(4000);  
  
            // Ocultar el Label en el hilo principal  
            Platform.runLater(() -> label_tooltip.setVisible(false));  
        } catch (InterruptedException ex) {  
            ex.printStackTrace();  
        }  
    }).start(); // Iniciar el hilo  
}
```

Este método, llamado `mostrarToolTip`, se utiliza para mostrar temporalmente un `Label` que contiene un mensaje de información. Al invocar el método, el `Label` se hace visible durante 4 segundos y luego se oculta automáticamente. Esto es útil para mostrar

información contextual en la interfaz en el momento preciso en que se necesite, como por ejemplo al realizar una acción específica o al dar una pista al usuario.

El método utiliza un nuevo hilo para manejar la espera, asegurando que el hilo principal de la interfaz gráfica no se bloquee. Además, las actualizaciones de visibilidad del `Label` se realizan de manera segura utilizando `Platform.runLater`, lo que garantiza que los cambios en la UI se ejecuten en el hilo principal de JavaFX.

11.4 If Ternario para validar modelo

```
private String buscarModelo(String model) {  
    model = model.trim();  
  
    String finalModel = (model.equalsIgnoreCase("modelx") || model.equalsIgnoreCase("model x"))  
        ? "Model X"  
        : (model.equalsIgnoreCase("model3") || model.equalsIgnoreCase("model 3"))  
        ? "Model 3"  
        : (model.equalsIgnoreCase("modely") || model.equalsIgnoreCase("model y"))  
        ? "Model Y"  
        : (model.equalsIgnoreCase("models") || model.equalsIgnoreCase("model s"))  
        ? "Model S"  
        : "";  
  
    return finalModel;  
}
```

El método `buscarModelo` toma como entrada una cadena de texto que representa un modelo de Tesla y devuelve una versión formateada y validada del nombre del modelo. Este método es útil para normalizar y corregir entradas de texto no estandarizadas que pueden provenir de usuarios o fuentes externas.

12. Recursos utilizados

Esta sección detalla las fuentes utilizadas para adquirir conocimientos y resolver problemas relacionados con el desarrollo en JavaFX durante el proyecto.

12.1 Curso JavaFX en YouTube por "DiscoDurodeRoer"

Este recurso consiste en una serie de videos tutoriales publicados en el canal de YouTube "DiscoDurodeRoer". El curso, titulado **JavaFX**, me proporcionó una base sólida para entender y trabajar con esta tecnología. A continuación, destaco los videos específicos que utilicé:

- **"Tutorial - Cómo preparar el entorno para programar JavaFX en Windows":**
- Aprendí a configurar el entorno de desarrollo para JavaFX en Windows, incluyendo la instalación del JDK, configuración de librerías y la creación de un proyecto básico. Este paso fue esencial para iniciar con JavaFX en NetBeans.
- **"Tutorial - Cómo usar Scene Builder JavaFX":**

Este video me enseñó cómo usar Scene Builder, una herramienta visual para diseñar interfaces gráficas en JavaFX. Gracias a este tutorial, aprendí a crear ventanas de manera eficiente, utilizando un enfoque visual en lugar de escribir todo el código manualmente.

- **"Ejercicios JavaFX #3 - Tabla de personas":**

Este ejercicio me ayudó a comprender cómo trabajar con tablas en JavaFX. Aprendí a mostrar datos dinámicos de colecciones dentro de una `TableView`, una funcionalidad clave en mi proyecto para visualizar los datos de mis colecciones de manera estructurada.

En general, este recurso fue fundamental para adquirir los conocimientos necesarios para el desarrollo inicial de mi proyecto.

12.2 Documentación oficial de JavaFX

La página oficial de JavaFX, accesible en <https://openjfx.io/openjfx-docs/>, fue otro recurso clave. En particular, utilicé esta documentación para:

- Integrar un proyecto modular de JavaFX en NetBeans:

Seguí las instrucciones detalladas en la guía oficial para configurar correctamente un proyecto modular, garantizando la compatibilidad entre las diferentes versiones de Java y JavaFX.

Este recurso me proporcionó una referencia confiable y actualizada para abordar los aspectos técnicos avanzados de JavaFX.

12.3 ChatGPT

Durante el desarrollo del proyecto, recurrí a ChatGPT para:

- Solucionar errores de código:

Utilicé este recurso para identificar y corregir problemas técnicos específicos, como configuraciones incorrectas o errores en el uso de componentes de JavaFX.

- Comprender el funcionamiento de JavaFX:

Gracias a ChatGPT, pude profundizar en conceptos y funcionalidades clave de JavaFX, como el manejo de eventos, la personalización de celdas en tablas, y el uso de `Timeline` para animaciones.

Este recurso fue especialmente valioso por la rapidez en la resolución de dudas y la flexibilidad para explorar distintas alternativas de implementación.

Resumen

Cada uno de estos recursos desempeñó un papel importante en el éxito de mi aprendizaje y desarrollo con JavaFX, complementándose entre sí para abarcar tanto los aspectos básicos como los avanzados de esta tecnología.

13. Conclusión

Este proyecto ha sido desarrollado como parte de mi formación en el ciclo superior de Desarrollo de Aplicaciones Multiplataforma (DAM) y tiene como objetivo principal demostrar las habilidades y conocimientos adquiridos durante mis estudios. A través de la implementación de la aplicación TeslaFXStore, he puesto en práctica conceptos clave como el diseño de interfaces gráficas, la programación orientada a objetos, la integración de bases de datos no relacionales y la organización de un proyecto siguiendo el patrón Modelo-Vista-Controlador (MVC).

Este trabajo no solo evidencia mi capacidad técnica, sino también mi compromiso con la resolución de problemas y el diseño de soluciones completas.