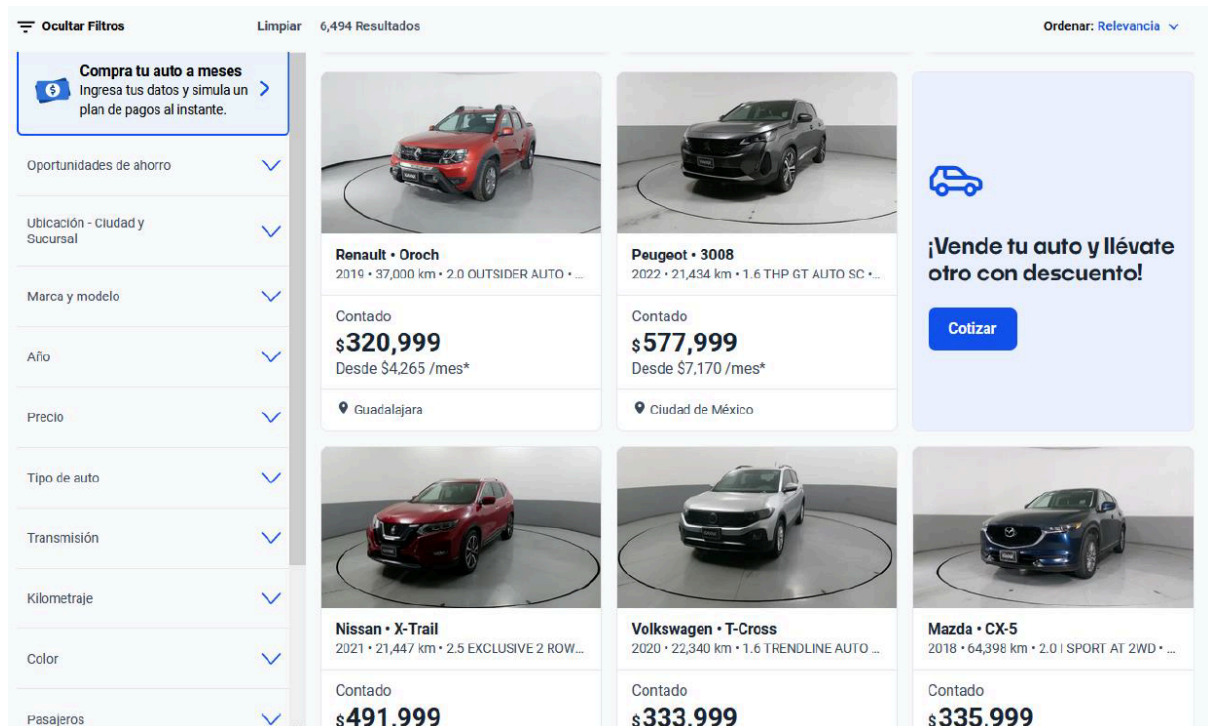


Proyecto ESPOT

Solicitante: Arrieta Mancera Luis Sebastian

Caso de estudio:

En este caso de estudio, se plantea el desarrollo de una aplicación web para mostrar el inventario de una tienda. La aplicación permitirá a los usuarios visualizar todos los productos en stock, **ordenarlos** por **precio** (de menor a mayor y de mayor a menor) y por **fecha** de creación. Además, se deben poder **filtrar** los productos por sus características. La elección del producto es libre, sin embargo, puedes tomar de referencia el filtrado y ordenación de **Kavak**:



Objetivos del caso de estudio:

- + Diseñar e implementar una interfaz intuitiva y atractiva para la aplicación.
- + Permitir el **filtrado** de productos por características.
- + Permitir **ordenar** los productos de acuerdo a su precio y fecha de creación.
- + Diseñar una base de datos persistente (relacional o no relacional) adecuada para guardar los productos y que alimente a la aplicación.

Entregables esperados:

- + Aplicación funcional que cumpla con los requisitos establecidos.
- + Documentación técnica que explique el diseño de la solución y tecnologías utilizadas.
- + Enlace del repositorio público de Github donde se aloje el código de la aplicación.

Notas adicionales:

- + El programador puede hacer uso de la tecnología que quiera para poder cumplir el objetivo.
- + El periodo de desarrollo es de una semana a partir del envío de este documento.
- + Se evaluará la limpieza del código, organización y estética.
- + Se tomarán en cuenta todos los puntos adicionales que consideres pertinentes desarrollar.

Fecha de entrega:

28 de junio a las 10:20 am, reunión vía zoom. Enlace: [<aquí va el enlace de zoom>](#)

Concepto (elección de producto): Tienda de automóviles (kavak)

Descripción General:

Este proyecto es una prueba técnica para la vacante de becario en desarrollo web. El objetivo principal del proyecto es desarrollar una base de datos que alimente la aplicación web y muestre los datos de una manera atractiva para el usuario, permitiendo el filtrado de los datos.

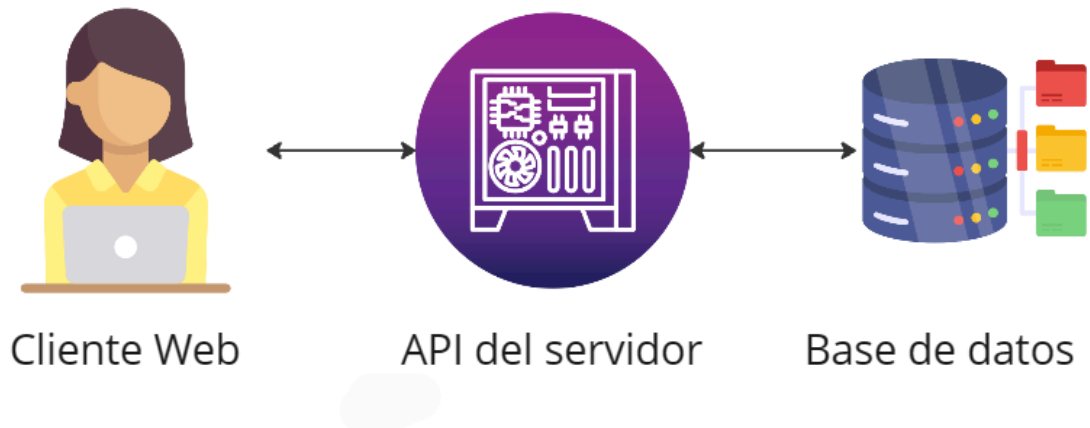
Diseño de la solución

El proyecto se basa en una arquitectura de cliente-servidor, donde el servidor maneja la lógica de negocio y la base de datos, mientras que el cliente se encarga de presentar la interfaz del usuario junto con todas las funcionalidades solicitadas por el caso de estudio.

Componentes principales:

- + **Base de Datos:** Almacena la información sobre los automóviles, esta alimenta la aplicación.
- + **Servidor API** (Backend): Proporciona endpoints para interactuar con la base de datos.
- + **Cliente Web** (Frontend): Interfaz de usuario que permite visualizar, ordenar y filtrar los automóviles por características.

Diagrama de arquitectura:



Desarrollo del proyecto

Elección del producto: Se decidió utilizar el mismo concepto de tienda de automóviles, de esta manera podemos simular un caso de la vida real como lo es Kavak.

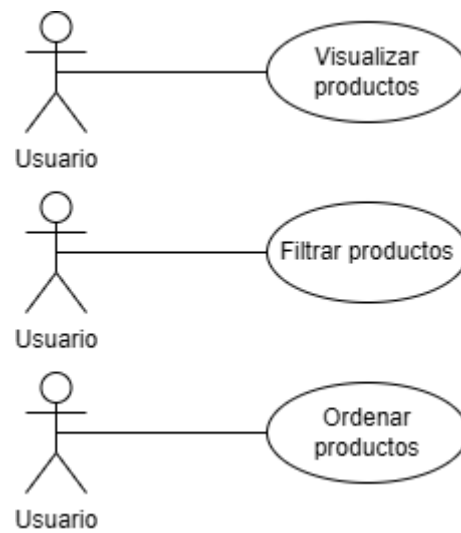
Análisis de requerimientos

Debido a que el alcance de la prueba técnica solamente consiste en desplegar productos al usuario, el modelo inicial consiste en **automóviles**. Un automóvil tiene una **marca, modelo, año, precio, transmisión, color, tipo de motor, kilometraje y stock**. Adicionalmente nos piden filtrar por **fecha de creación/entrada**, por lo que es información relevante para la empresa.

Cada automóvil requiere de **imágenes** para poder mostrarlas en la aplicación (por consideración de diseño se permitirá que los automóviles puedan o no tener imágenes).

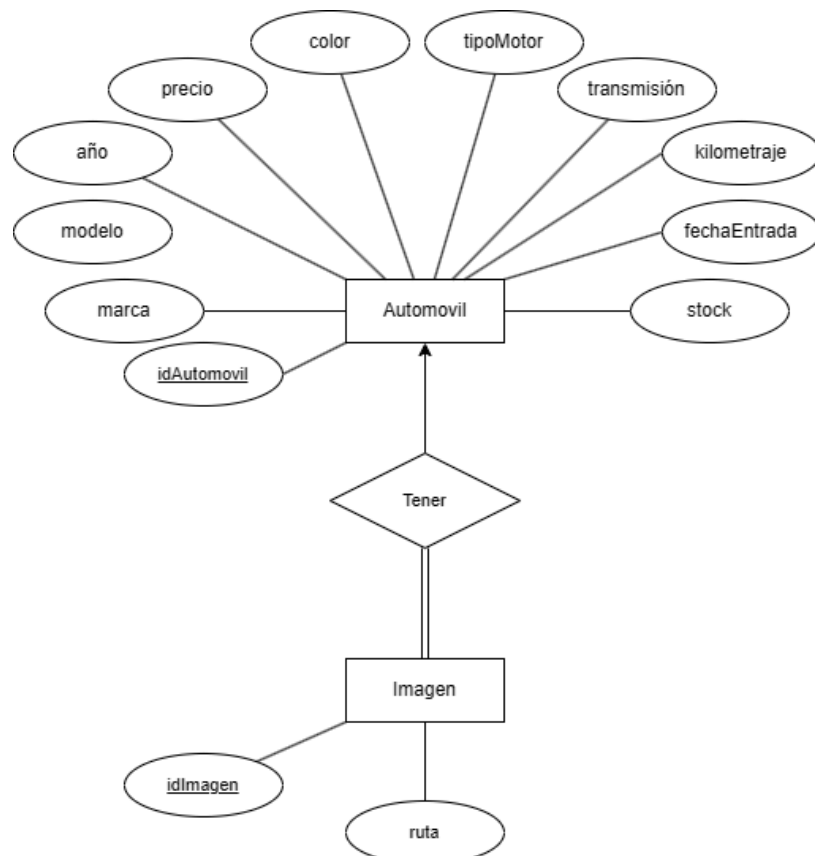
SMDB: El sistema manejador de bases de datos para este proyecto es MYSQL.

Casos de uso:



Diseño de la base de datos:

Modelo Entidad/Relación: En base al análisis de requerimientos se desarrolló el siguiente diagrama:

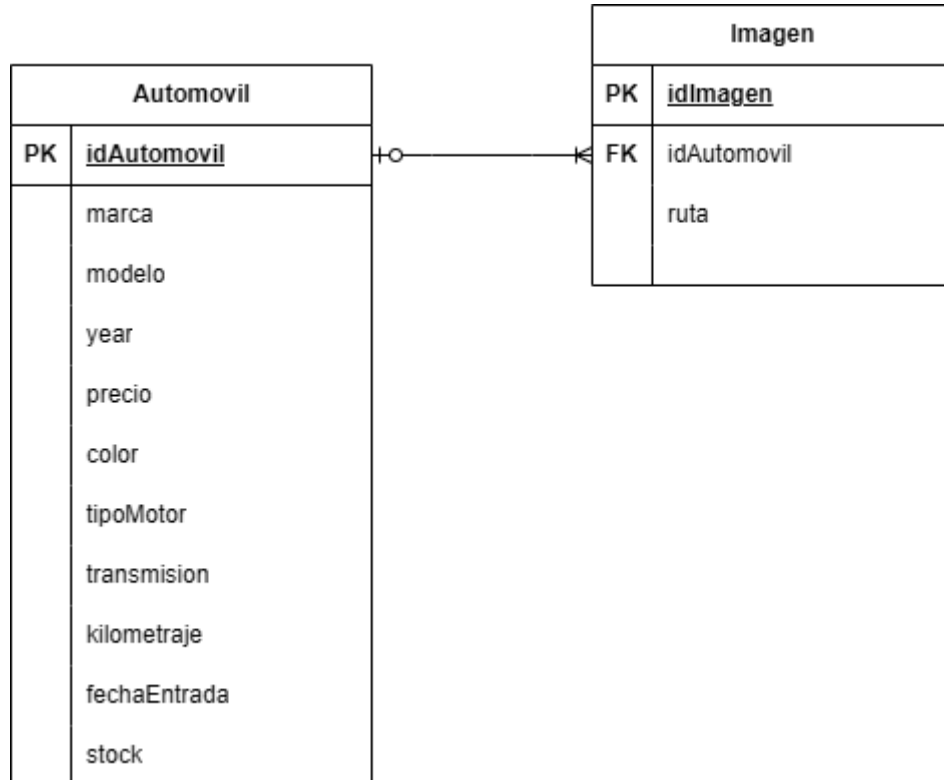


Traducción

Automovil(
 idAutomovil, PrimaryKey
 marca,
 modelo,
 year,
 precio,
 color,
 tipoMotor,
 transmision,
 kilometraje,
 fechaEntrada,
 stock
)

Imagen(
 idImagen, PrimaryKey
 ruta,
 idAutomovil ForeignKey
)

Modelo relacional



Implementación

El código de la implementación **DDL.sql** puede ser encontrado en el repositorio de github de este proyecto: <https://github.com/LuisSebs/prueba-tecnica-spot>

Tecnologías de desarrollo



Población de la base de datos:

Para la población de la base de datos se utilizó **Mockaroo**, se generarón 100 registros para la tabla **Automovil**. Por cada automóvil ingresado se ejecutaba un **disparador** que agregaba su imagen correspondiente en la tabla **Imagen**. Las imágenes no se guardaban en la base de datos, sino que se almacenaban en una carpeta llamada **imagenes_productos** en el backend, lo único que se ingresaba era la ruta a la imagen. Las imágenes fueron extraídas (la mayoría) de la página de Kavak. El archivo **DML.sql** puede encontrarse en el repositorio de github: <https://github.com/LuisSebs/prueba-tecnica-spot>

Desarrollo del backend:

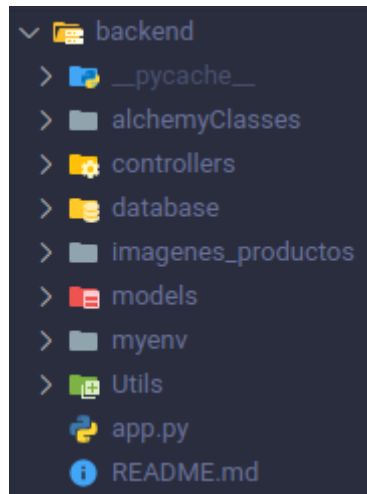
La lógica detrás del backend fue implementada en **python** junto con **flask** para manejar los endpoints de la **API**, así como también el mapeo de las estructuras de la base de datos (ORM) con **sql-alchemy**. El proyecto fue organizado en controladores, modelos y clases de sql-alchemy, además de una carpeta para las funciones auxiliares (Utils). Al ejecutarse, la aplicación corre en el **localhost** puerto **5000** en modo de desarrollo para poder imprimir en consola cuando sea necesario debuggear.

Envío de datos:

Las imágenes son enviadas codificadas en base64 para poder ser visualizadas por el usuario una vez llegan a cliente web. Cada endpoint envía la información en formato **JSON**.

Organización Backend:

La organización del proyecto se ve de la siguiente manera.




Entorno virtual:

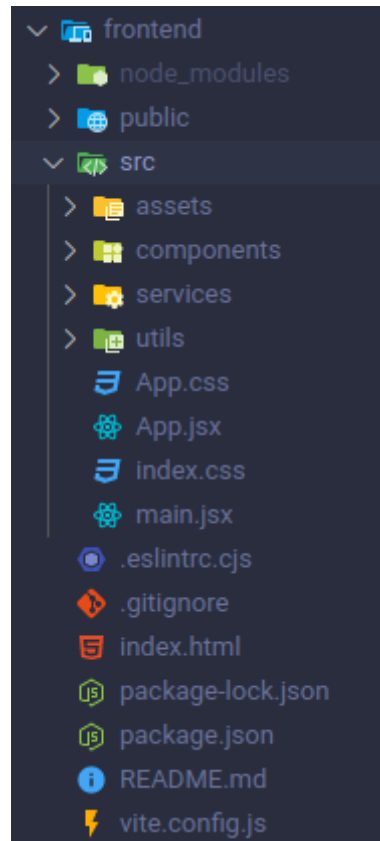
Para que el backend se ejecutara sin problemas se creó un entorno virtual para agregar todas las dependencias, algunas se enlistan a continuación:

- + Flask
- + Flask-Cors
- + Flask-SQLAlchemy
- + Pillow (para la codificación base64 de las imágenes)
- + PyMySQL
- + SQLAlchemy

Desarrollo del frontend:

La creación del proyecto fue realizado con **vite**  y **react**. Como el alcance de la prueba es una sola vista, se decidió no implementar un sistema de ruteo, sin embargo este puede integrarse en futuras versiones. Se organizaron los componentes en carpetas, intentando modularizar lo más posible. Análogo al backend se establecieron las carpetas componentes/, servicios/ y utils/.

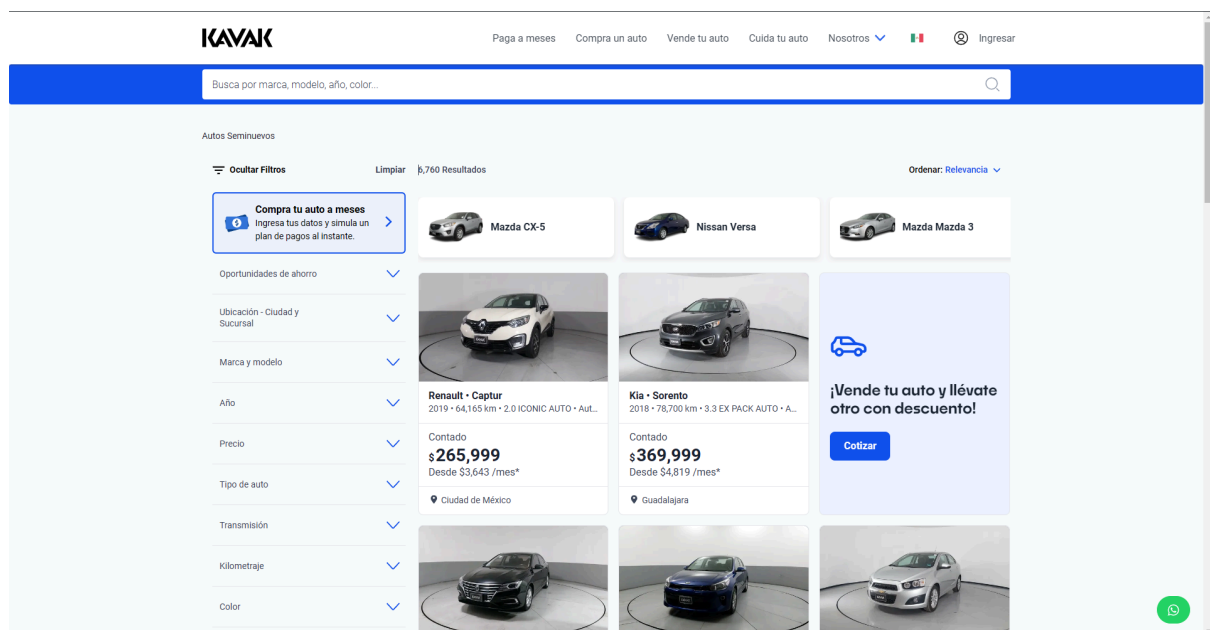
Organización Frontend:



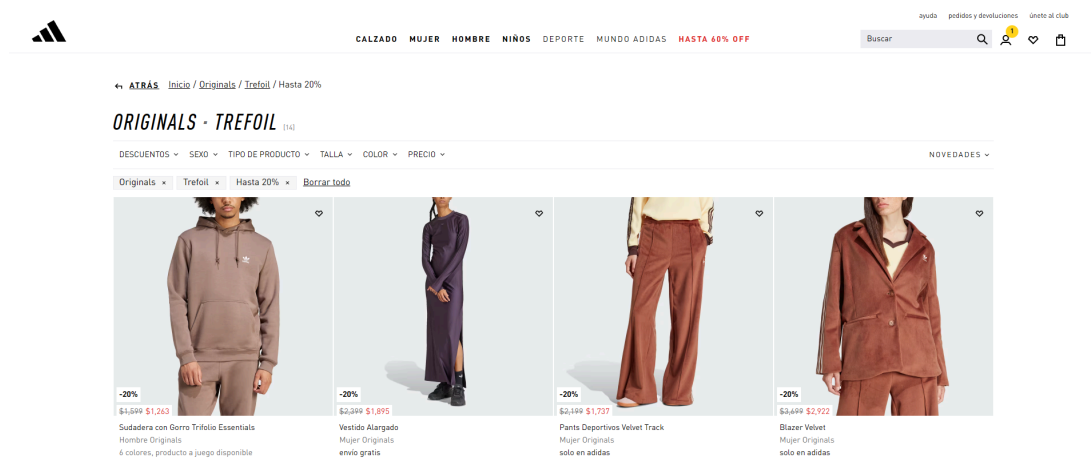
Inspiración de diseño

Como se mencionó anteriormente, las vistas se basan en la vista de compra de auto de Kavak. La interfaz del filtrado se inspiró en la página de [Adidas](#).

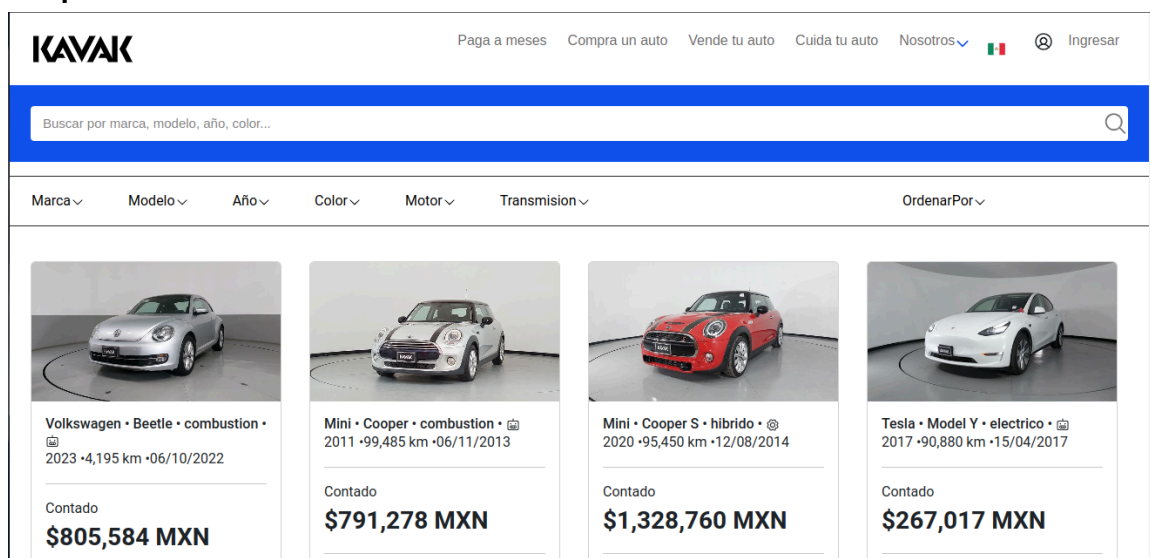
Vista Kavak



Vista Adidas



Vista prueba técnica



Recepción y manipulación de datos

La lógica detrás se encarga de acceder a la API del servidor, accediendo a cada uno de los endpoints para recuperar la información para funcionar. Se utilizó **axios** para las llamadas **ajax**. Para el diseño se utilizó **html** y **css** puro en su mayoría, mezclado con **react-bootstrap** y los iconos de **phosphor-icons**, parte de la funcionalidad de la interfaz se manejó con **javascript** puro. En esencia la aplicación se encarga en el primer renderizado de solicitar todos los automóviles a la API y estos más adelante son manipulados, sin necesidad de volver a hacer peticiones al backend. Se utilizó el algoritmo de **merge-sort** para ordenar los productos por **precio** y por fecha de creación en la base de datos. Los filtros son aplicados una vez el usuario ha seleccionado un filtro, si se aplica más de un filtro del mismo tipo entonces se van sumando productos, en caso de ser filtros diferentes se tiene una jerarquía, primero filtro por marca, modelo, año, color, motor y transmisión respectivamente. Cabe aclarar que en el código se evitó utilizar la palabra “año” para evitar problemas, esta fue remplazada por “year”.

Ejecución

Adicionalmente se agregaron archivos **README.md** para detallar la ejecución del código y un archivo `.gitignore` para descartar el seguimiento del cache de python (`__python__`). Es importante realizar los siguientes pasos en el orden establecido:

1. Activar el entorno virtual (**myenv**)
2. Correr el backend
3. Instalar las dependencias del frontend (*npm install*)
4. Correr el frontend

Futuras mejoras

- + Agregar más controladores para poder extender las posibilidades del filtrado y hacer más atractiva la interacción con el usuario. Por ejemplo, filtrar por kilometraje y establecer un rango de valores.
- + Permitir la anidación de listas desplegables. Por ejemplo, que para filtrar por modelo primero se tenga que seleccionar una marca de automóvil.

Dificultades con la implementación

Lo más complicado en términos de tiempo y esfuerzo fue definir la lógica e interacción entre los componentes del frontend con react.

Repositorio de Github:

<https://github.com/LuisSebs/prueba-tecnica-spot/tree/main>

Resumen de desarrollo:

- + Análisis de requerimientos ✓
- + Casos de uso ✓
- + Diseño de la base de datos ✓
- + Modelo entidad/relación ✓
- + Traducción ✓
- + Modelo relacional ✓
- + Implementación de la base de datos (DDL) ✓
- + Poblado de la base de datos (DML) ✓
- + Recolección de imágenes ✓
- + Creación de repositorio de Github ✓
- + Organización del proyecto ✓
- + Backend ✓
- + Frontend ✓
- + Conexión entre los componentes principales (backend, frontend) ✓
- + Pruebas ✓
- + Documentación ✓