**Resumen de Tecnologías Utilizadas en el Proyecto**

**I. Backend (API con Python y FastAPI)**

* **Python (Lenguaje de Programación):**
  + **Descripción:** Lenguaje principal para toda la lógica del backend. Elegido por su sintaxis clara, amplio ecosistema de librerías y frameworks robustos para desarrollo web.
* **FastAPI (Framework Web):**
  + **Descripción:** Framework moderno y de alto rendimiento para construir APIs con Python, basado en tipos estándar de Python. Se utilizó para crear todos los endpoints RESTful, gestionar las peticiones HTTP, y definir la estructura de la API. Sus ventajas incluyen la validación automática de datos con Pydantic, generación de documentación OpenAPI (Swagger UI y ReDoc), y soporte para programación asíncrona.
* **Uvicorn (Servidor ASGI):**
  + **Descripción:** Servidor ASGI (Asynchronous Server Gateway Interface) ligero y rápido. Se utilizó para ejecutar la aplicación FastAPI, manejando las conexiones de red y las peticiones asíncronas.
* **Firebase Admin SDK (Paquete firebase-admin):**
  + **Descripción:** Kit de desarrollo de software oficial de Google para interactuar con los servicios de Firebase desde entornos de servidor (como nuestro backend Python). Se utilizó para conectar la aplicación con la base de datos Firestore, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los datos de usuarios y coches.
* **Firestore (Base de Datos NoSQL de Firebase):**
  + **Descripción:** Base de datos NoSQL, orientada a documentos y alojada en la nube, proporcionada por Firebase. Se utilizó como el sistema de almacenamiento principal para todos los datos de la aplicación (usuarios, coches, y sus relaciones). Ofrece escalabilidad, flexibilidad en el esquema de datos y sincronización en tiempo real (aunque no la explotamos activamente en este proyecto).
* **Pydantic (Librería de Validación de Datos):**
  + **Descripción:** Librería utilizada por FastAPI (y también directamente) para la validación de datos y la gestión de configuraciones mediante anotaciones de tipo de Python. Se usó para definir los modelos de datos (models\_user.py, models\_coche.py), asegurando que los datos de entrada a la API (request bodies) y los datos de salida (responses) tengan la estructura y tipos correctos.
* **Werkzeug (Librería de Utilidades WSGI):**
  + **Descripción:** Aunque es una librería fundamental para WSGI (y Flask), en este proyecto se utilizó específicamente por sus funciones de utilidad para la seguridad, concretamente generate\_password\_hash y check\_password\_hash, para almacenar y verificar las contraseñas de los usuarios de forma segura (hashing).
* **python-jose[cryptography] (Librería JWT):**
  + **Descripción:** Librería para codificar, decodificar y verificar JSON Web Tokens (JWT). Se utilizó para implementar el sistema de autenticación basado en tokens. El backend genera un JWT cuando un usuario inicia sesión correctamente, y luego valida estos tokens en peticiones a endpoints protegidos.
* **python-dotenv (Librería para Variables de Entorno):**
  + **Descripción:** Utilizada para cargar variables de entorno desde un archivo .env (aunque decidiste no usarlo finalmente para SECRET\_KEY, la librería está instalada y es una buena práctica). Ayuda a gestionar configuraciones sensibles (como claves API, secretos) fuera del código fuente.
* **python-multipart (Librería para Parseo de Formularios):**
  + **Descripción:** Requerida por FastAPI para parsear datos de formularios (application/x-www-form-urlencoded o multipart/form-data). Se instaló para que el endpoint /token (que usa OAuth2PasswordRequestForm) funcione correctamente.
* **Entornos Virtuales (venv):**
  + **Descripción:** Herramienta estándar de Python para crear entornos aislados. Se utilizó para gestionar las dependencias del proyecto backend de forma separada, evitando conflictos entre las versiones de los paquetes de diferentes proyectos.

**II. Frontend (Aplicación Web con Python y Flask)**

* **Python (Lenguaje de Programación):**
  + **Descripción:** Lenguaje principal para la lógica del servidor del frontend y la renderización de plantillas.
* **Flask (Microframework Web):**
  + **Descripción:** Microframework ligero y flexible para construir aplicaciones web en Python. Se utilizó para gestionar las rutas del frontend, manejar las peticiones del navegador, interactuar con el backend API (enviando peticiones HTTP), gestionar sesiones de usuario (para almacenar el token JWT), y renderizar plantillas HTML dinámicas con Jinja2.
* **Jinja2 (Motor de Plantillas):**
  + **Descripción:** Motor de plantillas moderno y amigable para Python, integrado con Flask. Se utilizó para crear las páginas HTML dinámicas (.html en la carpeta templates/), permitiendo la inserción de datos de Python en el HTML, el uso de estructuras de control (bucles, condicionales), y la herencia de plantillas (con base.html).
* **Requests (Librería HTTP):**
  + **Descripción:** Librería estándar de facto en Python para realizar peticiones HTTP. Se utilizó en el frontend Flask (app.py) para comunicarse con la API del backend (FastAPI), enviando peticiones POST (para login, registro, crear coche, actualizar perfil) y DELETE (para eliminar perfil).
* **HTML (Lenguaje de Marcado):**
  + **Descripción:** Lenguaje fundamental para estructurar el contenido de las páginas web del frontend.
* **CSS (Hojas de Estilo en Cascada):**
  + **Descripción:** Utilizado para definir la presentación visual y el diseño de las páginas HTML (colores, fuentes, layout). Se implementó un archivo style.css global.
* **JavaScript (Básico, lado del cliente):**
  + **Descripción:** Se utilizó mínimamente para funcionalidades del lado del cliente, como el diálogo de confirmación (confirm()) antes de eliminar una cuenta de usuario.
* **python-dotenv (Librería para Variables de Entorno):**
  + **Descripción:** Al igual que en el backend, se utilizó para cargar configuraciones desde un archivo .env (como BACKEND\_API\_URL y FLASK\_SECRET\_KEY).
* **Entornos Virtuales (venv):**
  + **Descripción:** Se utilizó un entorno virtual separado para el frontend para gestionar sus propias dependencias (Flask, requests, etc.) de forma aislada