Group 10 - SpecCheck

Navjot Kehal
Timur Reziapov
Beck Harper
Bryan Le

Problem Statement

Many drivers lack accessible and centralized information about their vehicles. Essential details, such as engine specifications, transmission type, battery information, and maintenance-related metrics, are often difficult to locate using existing car apps. Users typically spend too much time trying to understand their car's details, and there is no convenient way to view live car data (like tire pressure, oil level, fuel, windshield fluid) while away from the vehicle . As a result, managing and understanding one's car becomes unnecessarily complicated.

This project aims to provide a unified web platform where users can easily view car information, receive live simulated vehicle data, interact with remote car controls, and use an integrated AI agent to better understand their vehicle.

Architectural Diagrams

For our project architecture, we decided to go for a simple monolithic-esque client server model. Our project specifically consists of the app server, our client pages, OpenAI API, CarQuery API and Firestore database. Below is a list of what each of these components do:

Server:

- Runs the backend and controls get and post requests made to the server
- Makes calls to the database to get and store user and car info
- Makes call to OpenAI API when user makes query to SpecCheck AI Agent
- Makes calls to CarQuery API to get car data and sends it to front end
- Sends data to front end using eJs rendering engine

Client:

- Uses eJs to render pages using data sent from backend
- Uses partials to render certain parts of a page in a modular way eg. the side menu
- Sends data to backend to be processed

OpenAI API:

- Processes queries made by users
- Uses custom built query generated by backend to imitate custom Car AI model

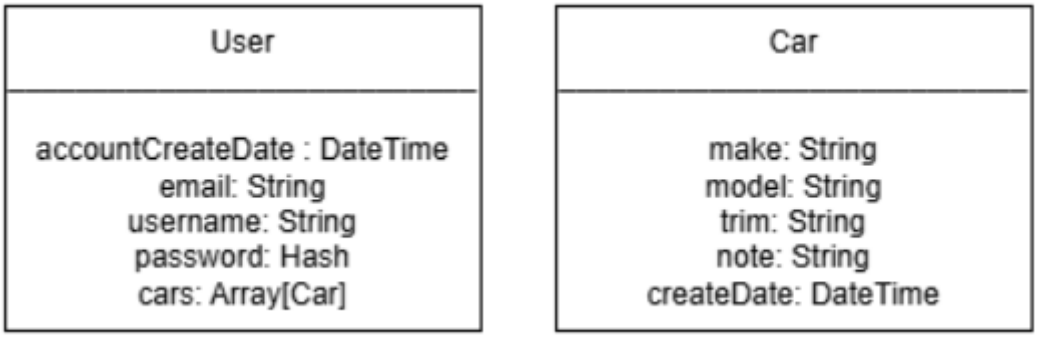
CarQuery API:

- Gives basic information about a car based on the make and model provided to it
- Used to create custom queries to give better responses when user asks a question to the SpecCheck AI Agent

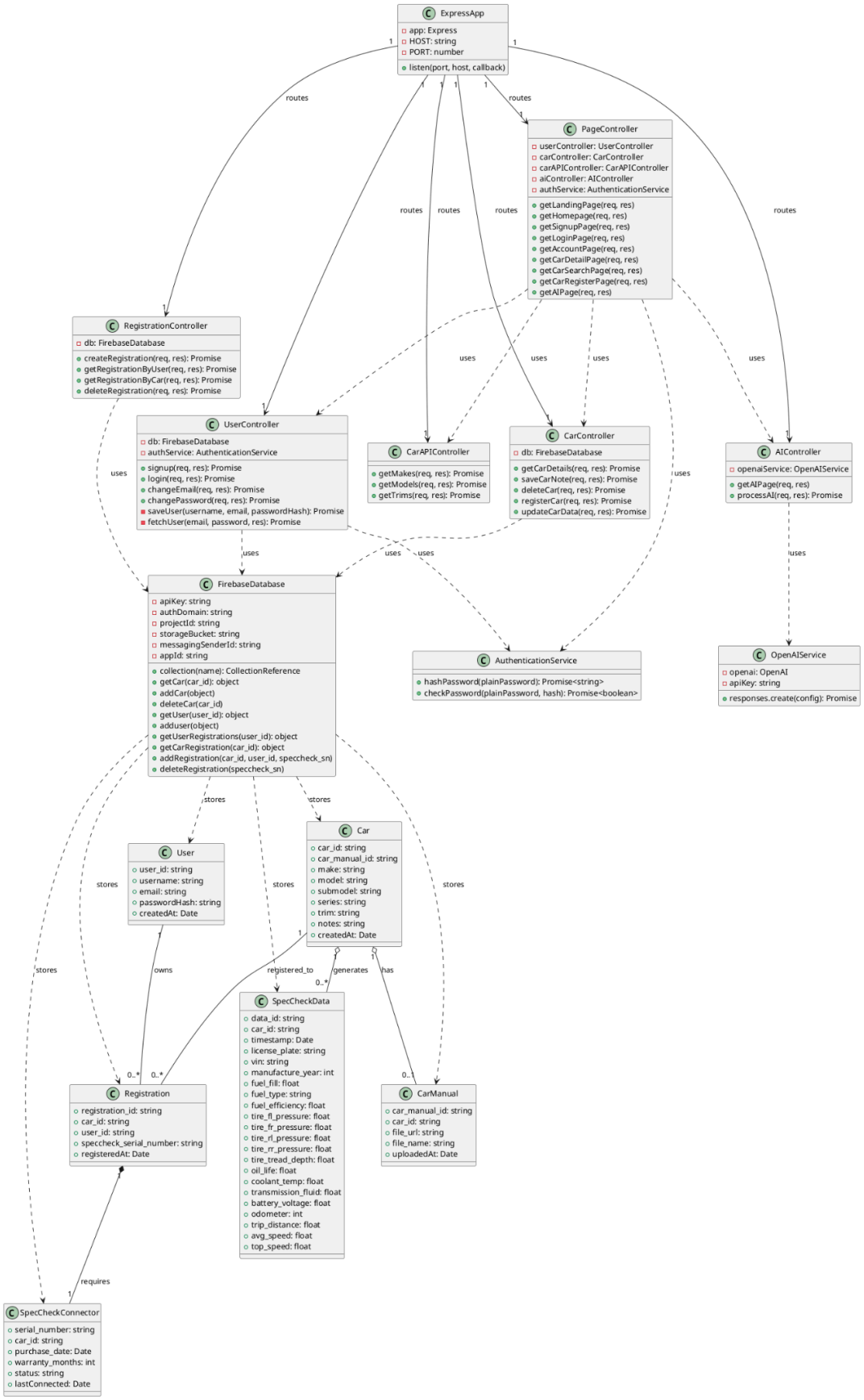
Firestore database:

- NoSQL database that uses docs to organize data
- Designed and implemented User and Car database diagrams to help organize users and their cars

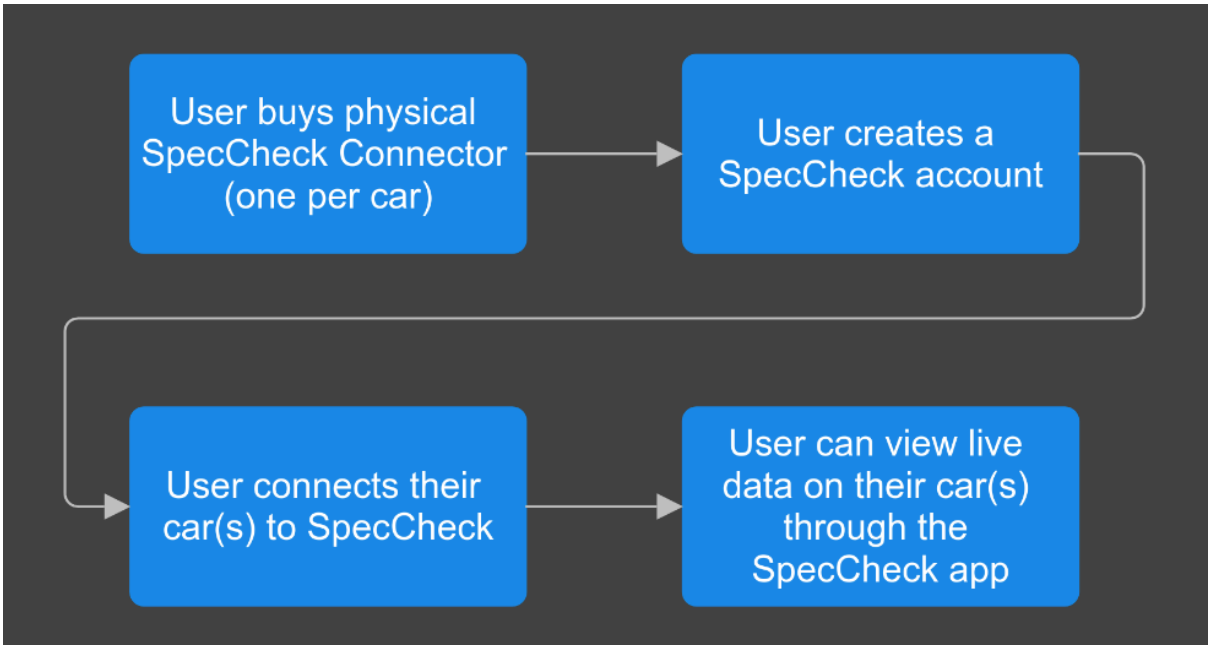
Database Diagrams

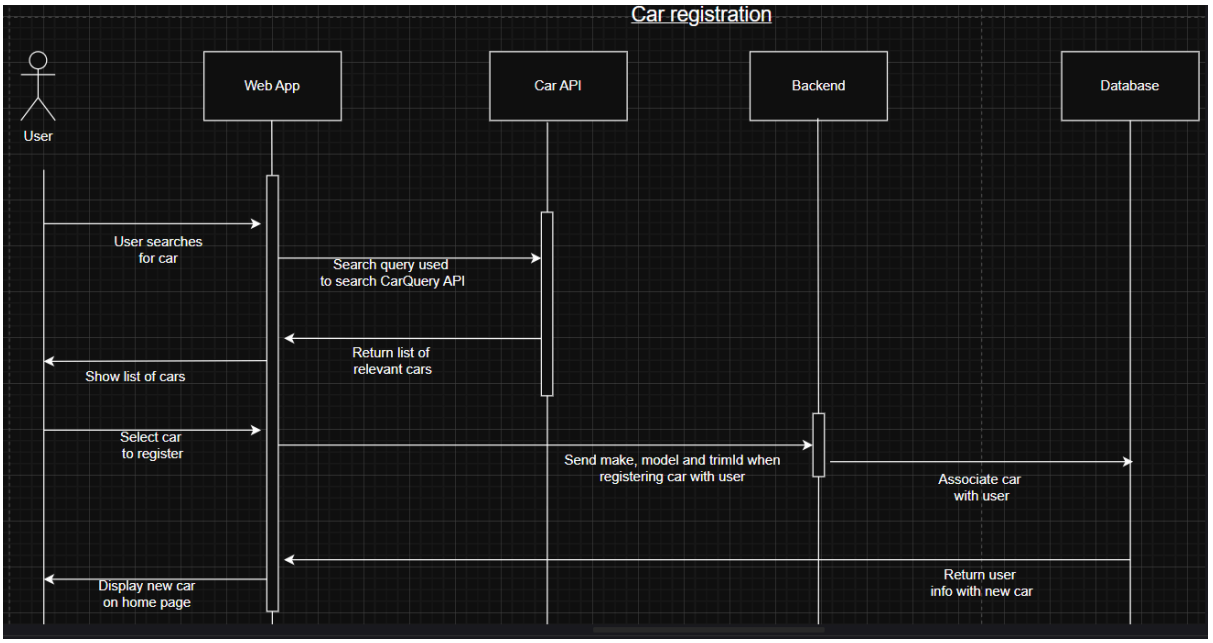


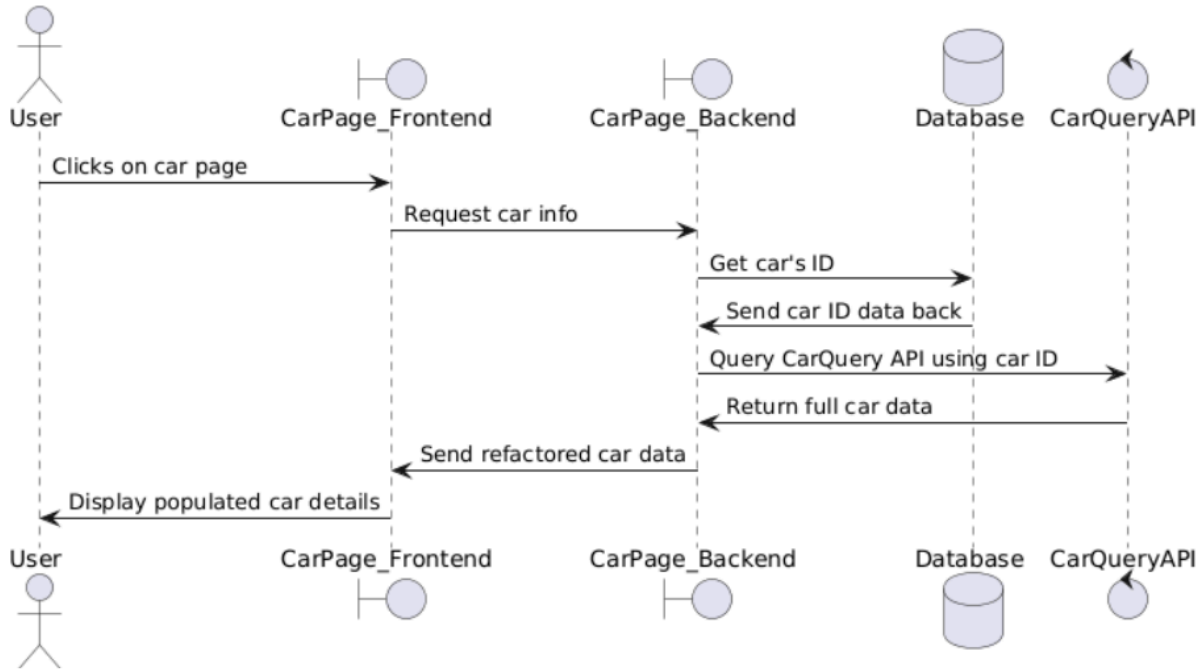
Class Diagrams

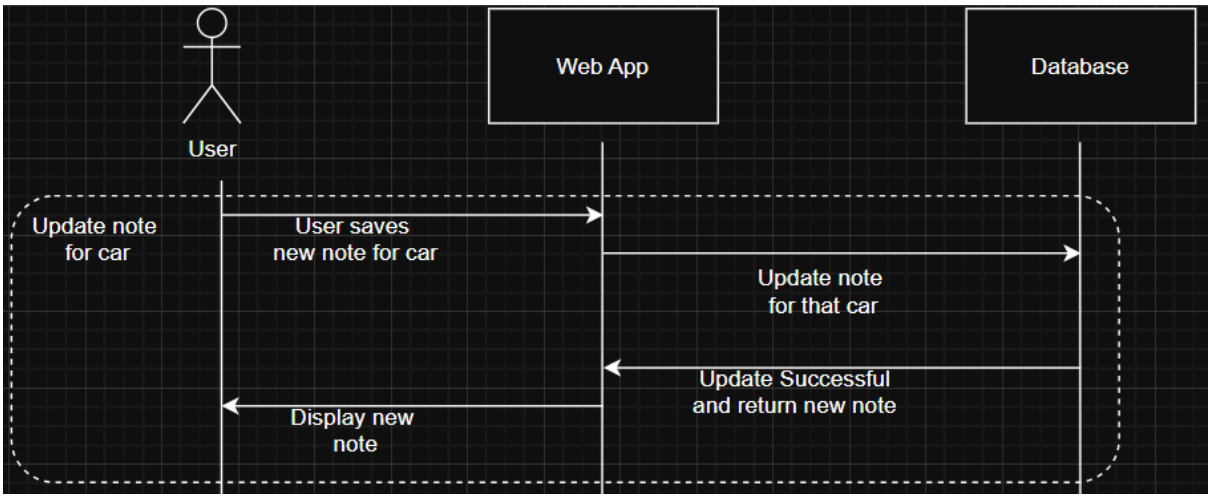


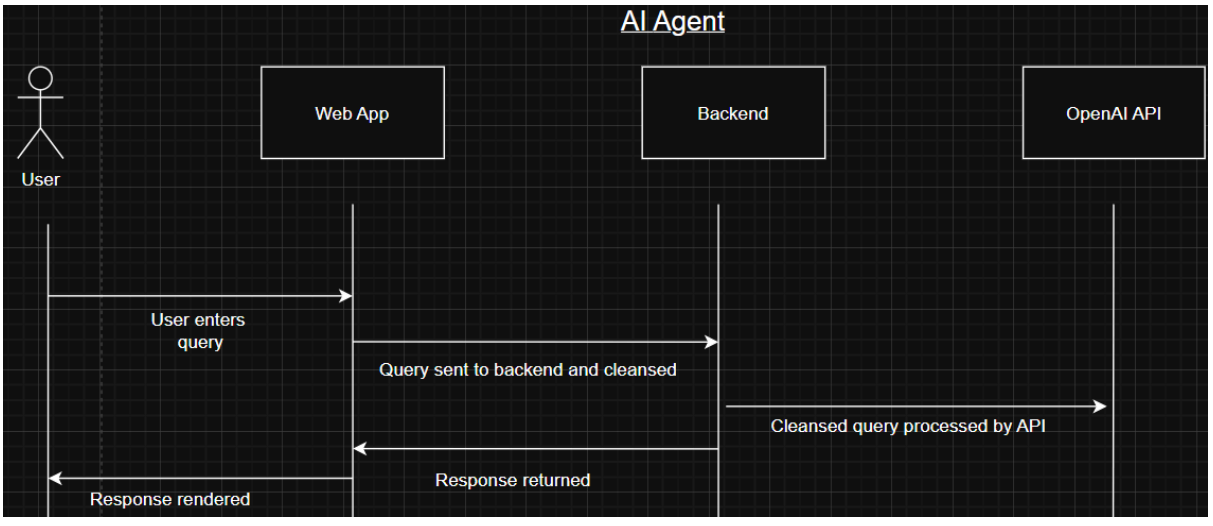
Sequence Diagrams



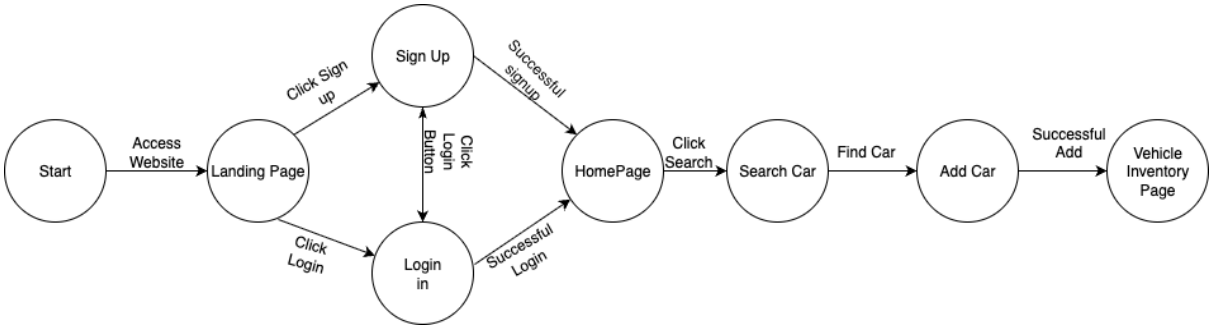


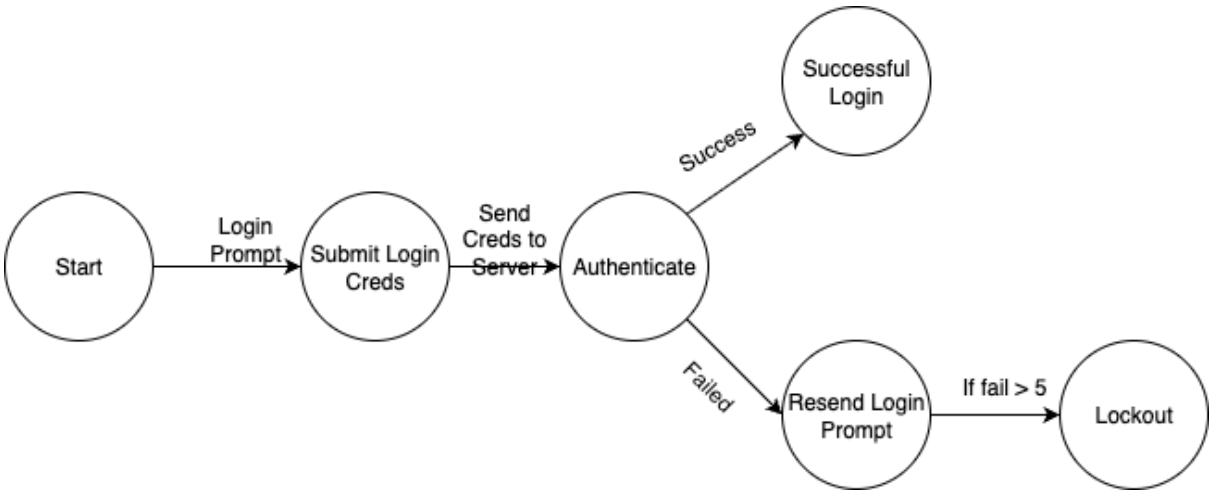

User
CarPage_Frontend
CarPage_Backend
Database
CarQueryAPI
Clicks on car page
Request car info
Get car's ID
Send car ID data back
Query CarQuery API using car ID
Return full car data
Send refactored car data
Display populated car details
User
CarPage_Frontend
CarPage_Backend
Database
CarQueryAPI


User
Web App
Database
Update note for car
User saves new note for car
Update note for that car
Update Successful and return new note
Display new note


AI Agent
User
Web App
Backend
OpenAI API
User enters query
Query sent to backend and cleansed
Cleansed query processed by API
Response returned
Response rendered

State Diagrams





Key Challenges and How They Were Resolved

1. Lack of Available and Affordable Car Manual APIs

Challenge:
We initially planned to integrate full car manuals into the app, which would be used along with an AI agent to give users accurate information about users' cars. Essentially the manual would be fed into the AI agent, along with the query, and it would tell the user only what they need to know, which would be easier and faster than reading through the entire manual to find specific info about your car. However, we found that available car manual APIs were either expensive, incomplete, or unreliable.

Resolution:
This feature was removed from scope, allowing us to focus on high-value features that were feasible with the time and resources available. To substitute this functionality, we instead used CarQuery API to feed data into the AI, which is explained below in point #4.

2. Difficulty Implementing Real-Time Car Data (tire pressure, oil, fuel, etc.)

Challenge:
Providing genuine live car diagnostics would require creating or sourcing specialized hardware (a custom OBD dongle) and building technology to read real-time vehicle data.

Resolution:
We reduced the scope and simulated live data instead. This maintained the intended user experience while avoiding the need for complex hardware development. The intention is that the creation of this specialized dongle would be its own project which would be worked on by electrical engineers.

3. Finding a Suitable Car Data Provider

Challenge:
Many automotive APIs either lacked sufficient data or required costly subscriptions.

Resolution:
We adopted the free CarQuery API, which provided a wide dataset of car models and specifications without financial barriers.

4. Creating a Custom AI Model

Challenge:
Building a specialized AI model for explaining vehicle information would have required time, training data, and infrastructure that were not feasible within the project timeline.

Resolution:
We used ChatGPT API and car info from CarQuery API to simulate the behavior of a custom AI assistant. By feeding the car info found from CarQuery API into the OpenAI API, we were able to achieve the required functionality with significantly less complexity.

Lessons Learned

1. Importance of Scope Management

Our project initially included ambitious features such as custom hardware, full manual integration, and proprietary AI models. Through evaluation and team discussions, scope was adjusted to match available time and resources. This prevented slowdowns and ensured a polished final product.

2. Communication is Critical

Weekly meetings, combined with ongoing communication outside of class, kept everyone aligned. Regular updates helped quickly identify blockers and distribute workload effectively.

3. Effective Use of Project Management Tools

Tools like Trello streamlined task tracking, prioritization, and division of duties. Creating internal “fake deadlines” ensured that milestones were met early and allowed room for revisions or setbacks.

4. The Value of Flexibility in Technical Decisions

Adapting to available APIs, replacing custom AI with ChatGPT, and simulating real-time features demonstrated that flexibility leads to sustainable solutions without sacrificing user experience.

Future Improvements

- **Integration of Real Hardware:**
  Work with electrical engineers to build a custom OBD-II dongle to provide actual live car metrics to our app rather than simulated data.

- **Car Manual Integration:**
  Incorporate a more robust or paid API to provide full car manuals within the app. To cover this cost, we can add paid subscriptions to use our app.

- **Fully Custom AI Model:**
  Train a domain-specific AI that can offer deeper, more personalized car insights.

- **Enhanced Remote Vehicle Controls:**
  Expand simulated controls (like door locks and light toggles), and eventually integrate real controls with explicit safety measures.

- **Mobile App Version:**
  Develop a mobile application for improved accessibility and real-time interaction.