Smart OBD Diagnostic System (SODS)

Team Name: Ctrl Alt Elite
Group Members: Cole Arduser, Luke Farmer, Sam Loecke, Sam Nicklaus
Date: 11/25/2023

Executive Summary

In the current landscape of automotive technology, car owners and enthusiasts face a unique challenge: accessing real-time vehicle diagnostics and conducting detailed post-trip analyses. Traditional tools offer limited functionality, typically by using the On-Board Diagnostics (OBD) port of the car to grab current data such as speed, rpm, and fuel economy, or to diagnose engine codes. The current systems sold today are specialized towards car enthusiasts and are usually not user friendly. This makes a space in today's market, one that our project, the Smart OBD Diagnostic System (SODS), aims to fill.

SODS is a project designed to provide both real-time data visualization in the vehicle and extensive post-trip data analysis through an easy-to-use web application. This functionality addresses the growing need for comprehensive, user-friendly tools that allow car owners to make informed decisions about their vehicles' performance and maintenance.

Our approach is two-fold. Inside the car, SODS features a physical dashboard that displays essential vehicle data, such as engine temperature, oil pressure, and fuel efficiency. This display will enhance the driving experience by providing crucial information to the user. In the event of any diagnostic alerts, the system will evaluate and communicate the severity of these issues, allowing drivers to respond appropriately.

Outside the car, SODS extends its utility through a web application designed for detailed post-trip analysis. This platform offers users a deeper dive into their vehicle's performance and health, providing advanced analytics, historical data review, and trend analysis. Whether a user seeks a simple overview of their car's health or detailed insights into specific metrics, SODS will cater to a wide range of needs and expertise levels.

In developing SODS, we will ensure that it is scalable, secure, and user-friendly. The system will be built to support future enhancements, such as integrating additional sensor readings and expanding analytical capabilities.

In summary, the Smart OBD Diagnostic System aims to provide car owners and enthusiasts alike the data and information they need to safely, securely, and efficiently operate and maintain their vehicles. This project will transform the way car owners understand their vehicles by providing both real-time data access and comprehensive post-drive analytics in a user-friendly format. SODS is built to allow users to be more informed, connected with their driving experience.

1) Problem Statement

In today's era of ever-advancing automotive technology, there is an increasing need for a comprehensive system that not only facilitates real-time access to vehicle diagnostics but also offers post-trip analytics. Current solutions utilizing the OBD port in vehicles typically fall short in providing an all-in-one solution for both live data access and post-process analysis. This project seeks to fill this gap by introducing a system that seamlessly integrates live vehicle data visualization in the car with an easy-to-use web application for detailed analytics and further processing after the trip. The system will include a physical dashboard in the vehicle that will display real-time data such as speed, rpm, and other user customizable details, focusing on presenting this information in a way that prioritizes driving safety and minimizes distractions. Additionally, the device will assess the severity of any diagnostic codes that may pop up during the drive, guiding drivers to make informed decisions about their vehicle's operation and maintenance in real-time.

Beyond the physical dashboard in the vehicle, this system extends its utility through a comprehensive web application designed for post-trip analysis. The web component will enable users to dive deeper into their vehicle's performance and health data, offering advanced analytics, historical data review, and trend analysis. This approach ensures that users not only receive vital information in real-time but also have the opportunity to engage in extensive analysis and informed decision-making regarding their vehicle's long-term maintenance and performance. The integration of immediate data access with extensive analytics aligns with the growing demand for clean, user-friendly vehicle diagnostic and analysis tools. This project represents a changing trend in giving car owners an informed approach to vehicle care and maintenance. Many car owners today want to be more informed before having to trust their mechanic blindly. Our system will do just that by informing the owner of their vehicle's issues.

With access to both real time and post-trip vehicle data, car owners will enjoy many benefits. These include more informed use of the vehicle, knowledge of engine troubles before going to the mechanic, and a safety alert feature for serious engine error codes when driving to save the car owner from driving when it is unsafe.

2) Design Objectives

1) Design and implement a collection device to collect the relevant data from the vehicle's OBD 2 port. This device must relay data to a database through a web server.

2) Design and implement a database to store data collected from the vehicle during operation. This data will be used for historical data analysis in the web application.

3) Design and create a dashboard to show live engine data and engine codes in the vehicle while operating.

4) Design and create a web application that is used for viewing historical data collected in the car. The web application will be used to run advanced analytics and spot trends.

5) Design and implement a web server to facilitate communication between the collection system, the database, and the web application.

3) Major System Requirements and Constraints

a) Major System Requirements

The major system requirements will be organized into level one (L1) and level two requirements (L2). The level one requirements will be a wide-ranging scope and be overarching system requirements. The level two requirements will be narrowly scoped requirements that should be easily testable individually.

L1.1: The system must collect data from the car's OBD II port

- L2.1.1: The system shall use a device capable of reading OBD II interface data to extract the available data from the OBD II port.
- L2.1.2: The data collection device shall have a Bluetooth interface for communicating data to a Bluetooth-enabled device.
- L2.1.3: The data collection device shall collect all available data from the OBD II port.
- L2.1.4: The data collection device shall automatically start collecting data when the car ignition is turned on and automatically stop when it is turned off.

L1.2: The system shall save collected data in a local database

- L2.2.1: The Bluetooth-enabled device receiving the data shall have a local storage database.
- L2.2.2: The local database shall store at least 48 driving hours of data at a maximum sampling rate before requiring synchronization with the web server.
- L2.2.3: The database shall implement efficient data storage formats.

L1.3: The user shall have a physical touchscreen interface for selecting and viewing live data in the car

- L2.3.1: The touchscreen interface shall display at least 720p resolution.
- L2.3.2: The interface shall have a menu for selecting from all available data points.
- L2.3.3: The interface shall display up to ten live data points simultaneously.
- L2.3.4: The touchscreen interface shall be no less than 7 inches in screen size.
- L2.3.5: The physical device shall have a controlled shutdown process when power is lost.

L1.4: The physical in-vehicle system shall display engine codes and alert the user of their severity

- L2.4.1: The system shall display a message for detected engine trouble codes.
- L2.4.2: The system shall display a severity assessment based on the error code.
- L2.4.3: Severity assessments shall be rated on a scale of 1 - 5.
- L2.4.4: The system shall recommend whether to stop immediately, stop when possible, continue with care, or continue as normal.

L1.5: The in-car system shall upload the collected data to a web server

- L2.5.1: The system shall upload data to a specified web server.
- L2.5.2: The system shall only attempt to upload data with an established internet connection.
- L2.5.3: The system shall only upload data collected since the last successful upload.

L1.6: The web server shall collect and store received information

- L2.6.1: The web server shall be always on and hosted online.
- L2.6.2: The web server shall receive data from the authorized in-vehicle device.
- L2.6.3: The web server shall only accept data from authorized devices.
- L2.6.4: The web server shall utilize an efficient database.

L1.7: The system shall include an efficient database for storing collected data

L1.8: The system shall include a web application for viewing, comparing, and analyzing collected data

- L2.8.1: The web application shall communicate with the web server for data retrieval.
- L2.8.2: The web application shall include options for viewing all available data points.
- L2.8.3: The application shall allow users to select a time period for graphing specific data points.
- L2.8.4: The web application shall have a homepage displaying common data points on page load.
- L2.8.5: The application shall allow users to compare and graph selected data points over a specified time range.

b) Major System Constraints

SC1: Size and Weight: The combined system shall fit within a maximum volume of 25 cm x 20 cm x 10 cm (L x W x H).
SC2: Power Consumption and Supply: The system shall operate on a standard 12V DC car power supply. The total power consumption of the system should not exceed 15 Watts under normal operating conditions.
SC3: User Interface Response Time: The system shall respond to all user interface actions with a latency not exceeding 250 milliseconds.
SC4: Data Transmission and Connectivity: The Bluetooth connection between the OBD II adapter and the Raspberry Pi shall maintain stable connectivity within a range of 5 meters, with a data transmission latency not exceeding 500 milliseconds under normal conditions.
SC5: Operating Temperature: The system shall operate effectively in a temperature range from -10°C to 50°C (14°F to 122°F).
SC6: Durability and Vibration Resistance: The system shall withstand vibrations typical of a moving vehicle environment without malfunctioning or losing structural integrity

c) Standards

- Vue.js v3.3.9: Adherence to ES6 for JavaScript and compliance with modern web development standards including HTML5 and CSS3.
- Python v3.12.0 & Flask v3.0.x & Kivy 2.2.1: Using the latest stable release for security and performance.

- PostgreSQL v16.1: Compliance with data protection laws (like GDPR) for handling and storing user car data.
- SQLite v3.44.2
- Google OAuth 2.0
- RESTful API Design
- Implementing robust encryption for the event of transferring data
- OBD-II Protocols: Compliance with OBD-II standards for vehicle diagnostics and data retrieval.
- Bluetooth: Adherence to Bluetooth Core Specification for wireless data transmission.
- Wi-Fi: Compliance with IEEE 802.11
- Compliance with safety standards for electronic devices to prevent hazards such as overheating or electrical malfunctions
- Compliance with the NHTSA standards for Federal Motor Vehicle Safety Standards

d) Societal Factors

Public Health, Safety, and Welfare

Positive Impact: Enhances vehicle safety by providing real-time diagnostics and alerts. This can lead to early detection of potential vehicle malfunctions, reducing the risk of accidents due to vehicle failure.

Negative Impact: Any in-car system carries the risk of distracting the driver. Even though our project is designed to minimize distractions, the presence of a touchscreen and real-time data can potentially divert attention from driving.

Social and Cultural Factors

Positive Impact: By allowing access to advanced vehicle diagnostics, our project can help bridge the knowledge gap for car owners who are not automotive experts. This can lead to a more informed public, capable of making better decisions regarding vehicle maintenance.

Negative Impacts: There are concerns regarding privacy, as the system collects and transmits vehicle data. Ensuring robust data security is crucial to prevent misuse of personal information.

Knowledge Sensitivity: The system should be designed to be user-friendly for a diverse user base, considering different levels of technical expertise.

Environmental Factors

Positive Impact: Efficient vehicle maintenance and diagnostics can lead to better fuel efficiency and reduced emissions, contributing positively to the environment.

Negative Impact: Consideration must be given to the environmental footprint of system components such as the recyclability of the Raspberry Pi, touchscreen, and other components we use.

Economic Factors
Affordability: Keeping the system affordable is important for production in the future. We Utilized existing hardware and open-source software to attempt to keep costs down.

Economic Viability: The project must balance cost with functionality to ensure it is economically viable for the majority of users who own a vehicle.

Global Factors
Scalability and Accessibility: The system should be adaptable to different vehicle types, considering variations in vehicle technology and diagnostic standards across the United States.

Broad Impact: If widely adopted, the system could contribute to global road safety and environmental sustainability by enabling better vehicle maintenance and performance across different countries and cultures.

4) Design Concept

a) Research and Investigation

In the initial phase of our project, we conducted an in-depth technical feasibility study to ensure the viability of the Smart OBD Diagnostic System (SODS). This involved researching existing solutions, assessing hardware and software compatibility, and identifying unique contributions our team could make to the project.

1. Previous Related Projects

- We started by examining similar projects that integrate Bluetooth technology for communication between an OBD sensor and a Raspberry Pi with a display setup. Numerous open-source projects provided insights into the practical implementation of such systems. One notable reference was a Raspberry Pi-based OBD car project, which utilized Bluetooth for data transmission. This project served as a proof-of-concept, demonstrating the technical feasibility of our proposed system.
- Raspberry Pi OBD Car Project

2. Library and Software Compatibility

- Our research confirmed the compatibility of necessary libraries and software with the Raspberry Pi platform. We identified python-OBD, a comprehensive library for handling OBD data, which is compatible with Raspberry Pi and suitable for our project. For the web server and application, we drew upon our prior experience in creating a website interfaced with Raspberry Pi data, affirming the feasibility of integrating web connectivity and data management.

- OBD Python Library, Kivy Code Link

3. Hardware Feasibility

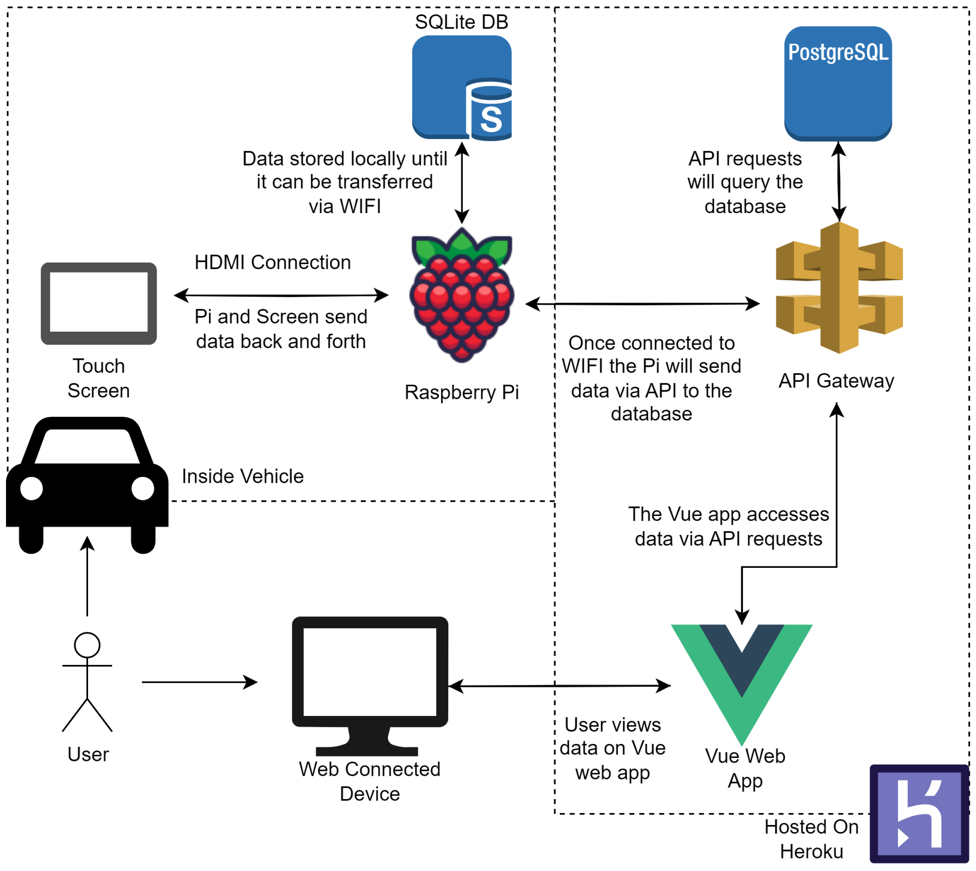
- We evaluated the hardware components required for our system, including the Raspberry Pi, Bluetooth OBD II adapter, and touchscreen display. Our selection criteria were based on processing capabilities, compatibility, and ease of integration. The uninterrupted power supply (UPS) system's feasibility was verified to ensure stable operation during power fluctuations in automotive environments.
- **Raspberry Pi Power, Touch Screen Link, Bluetooth OBD**

4. Differentiation from Existing Projects

- While there are existing projects with similar components, SODS distinguishes itself in its comprehensive integration of real-time vehicle data visualization, advanced post-trip analysis via a web application, and user-friendly interface focused on minimizing driver distraction. Our project leverages and extends open-source technologies, but the unique value lies in our system's holistic approach, combining in-vehicle real-time data display with extensive data analytics capabilities on a web platform.

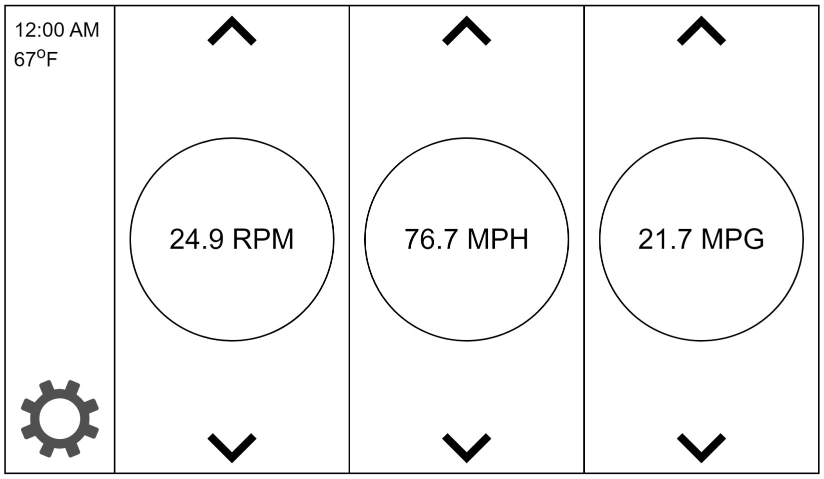
b) Selection of Design Concept

Block Diagram of Entire System:

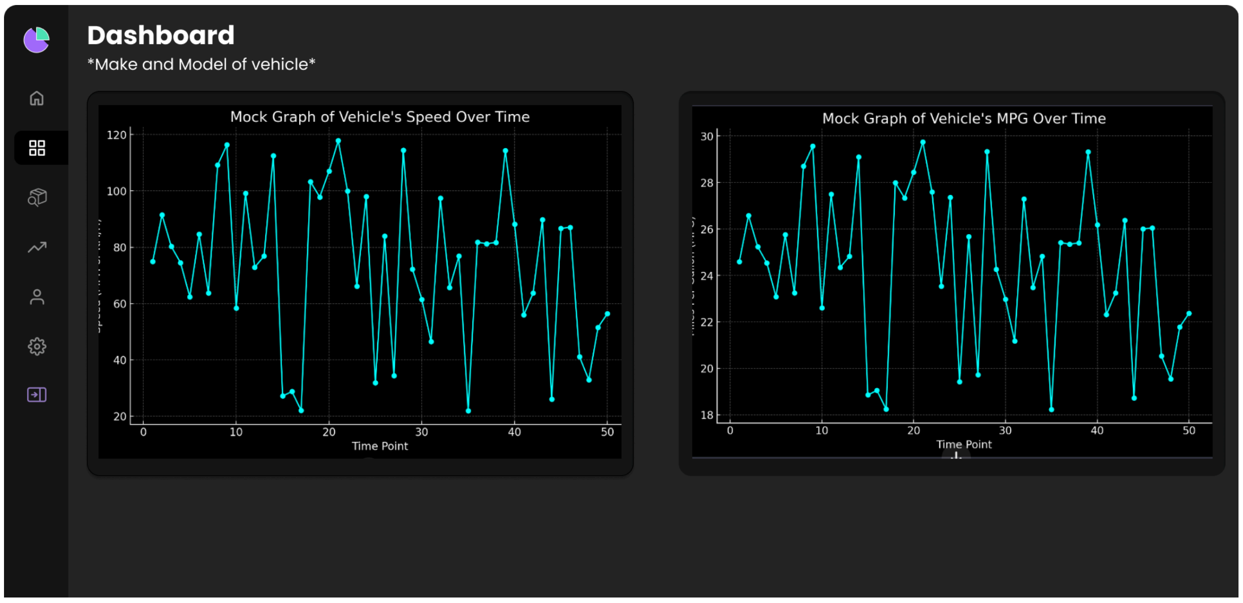

SQLite DB
PostgreSQL
Data stored locally until it can be transferred via WIFI
API requests will query the database
HDMI Connection
Pi and Screen send data back and forth
Touch Screen
Raspberry Pi
Once connected to WIFI the Pi will send data via API to the database
API Gateway
Inside Vehicle
The Vue app accesses data via API requests
User
Web Connected Device
User views data on Vue web app
Vue Web App
Hosted On Heroku

WireFrames:

TouchScreen



View Web App


Dashboard
Make and Model of vehicle
Mock Graph of Vehicle's Speed Over Time
Time Point
Mock Graph of Vehicle's MPG Over Time
Time Point

Overview:

As shown in our block diagram for our entire system, our design employs many different technologies and frameworks. The system starts with the bluetooth enabled OBD II reader. This reader communicates the OBD II data with the raspberry pi which serves as our local host for running the in-vehicle application on the touch screen. The touch screen displays data read from the OBD II port. The user will be able to select which data points to display. The screen will only display live data. When connected to wifi, the raspberry pi will communicate the saved data to the web server over Python Flask API. The web server and database will be hosted on

Heroku. The web server will communicate with the database and a frontend Vue application. The web server will be the only point of contact with the database. The Vue application is where the analytics and viewing of collected data will be allowed. The Vue application will also be hosted on Heroku and will communicate with the web server via Python Flask API

5) Deliverables and Milestones

a) Project Deliverables

1. Hardware Components

1.1 OBD II Bluetooth Adapter: A device capable of reading data from the OBD II port and transmitting it via Bluetooth.
1.2 Raspberry Pi: The central processing unit of the system, equipped with necessary software and interfaces for data collection and transmission.
1.3 Touch Screen Display: A user interface device for displaying real-time data visualization in the vehicle
1.4 Uninterrupted Power Supply: Ensures a controlled shutdown of the Raspberry Pi when the car ignition is turned off.

2. Software Components

2.1 In-Car Application Software: Developed using the Kivy Python UI framework, this software runs on the Raspberry Pi and provides real-time data visualization on the touch screen.
2.2 Web Server Software: A Flask Python-based server responsible for managing data transmission between the Raspberry Pi and the web application.
2.3 Web Application: Developed using Vue.js, this application allows users to analyze, compare, and visualize historical vehicle data. It includes engine code explanations, repair suggestions, and integrates with the Python Flask API.

3. Database System

3.1 Local Database Storage on Raspberry Pi: For storing data collected from the OBD II port before synchronization with the web server.
3.2 Cloud-based Database: Hosted on Heroku, this database stores all the data collected and transmitted from the vehicle.

4. Documentation and Manuals

4.1 User/Operator Manuals: Comprehensive guides for end-users on how to operate the in-car system and the web application.
4.2 Design Documentation: Detailed technical documentation including system architecture, component specifications, and software design.

5. Additional Deliverables

5.1 Full Product (Near-Final): A refined version of the total system including all of parts 1, 2, and 3 that incorporates feedback from the initial prototype testing

6. Project Management and Reporting Tools

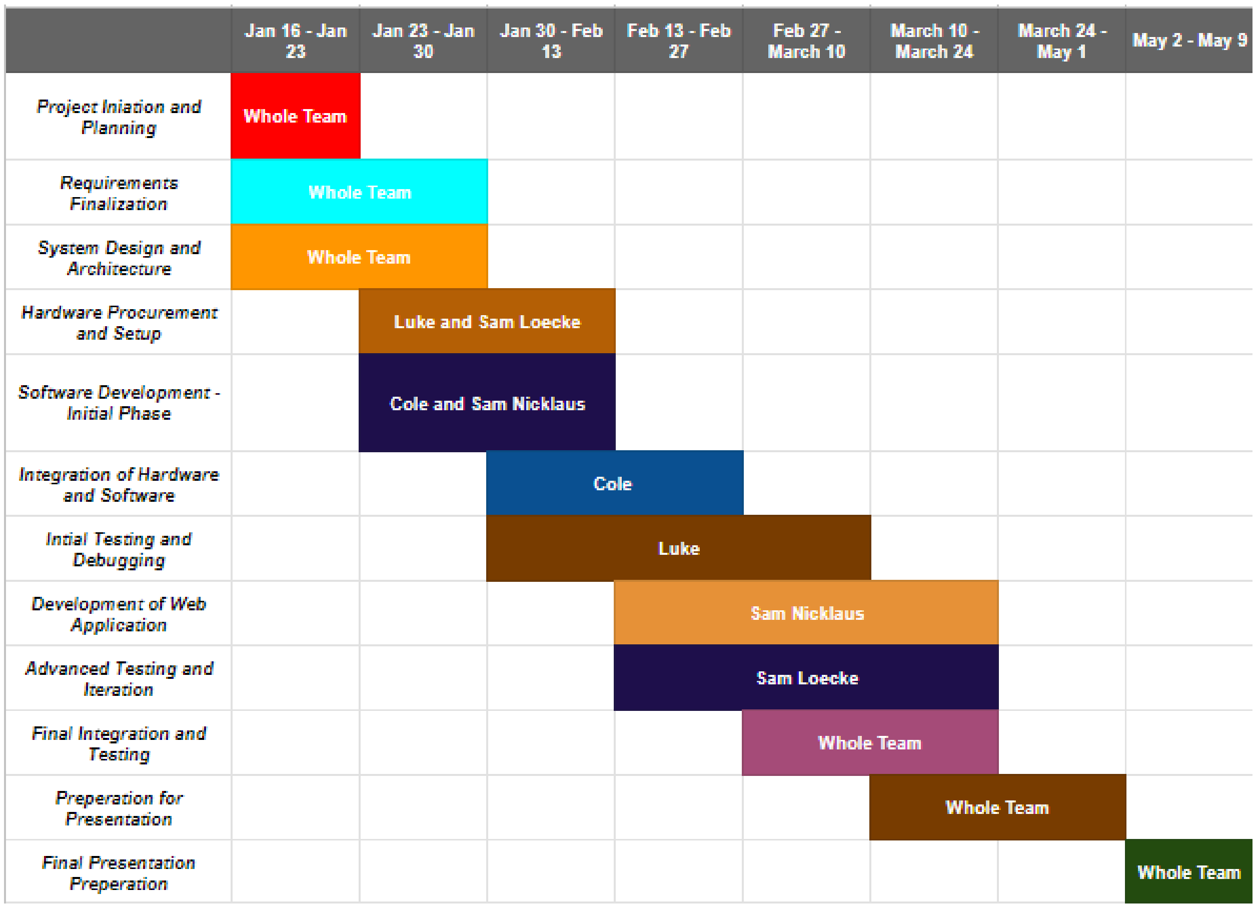
6.1 Gantt Chart and Timeline: Document outlining the project schedule, milestones, and deadlines.

6.2 Risk Management Plan: Detailed strategy for identifying, mitigating, and managing potential project risks.

7. Software Code and Repository

7.1 Source Code: All developed software code for the in-car system and web application.
7.2 Version Control Repository: Access to the Git repository hosting the project's source code for ongoing development and version tracking.

b) Schedule and Timeline



	Jan 16 - Jan 23	Jan 23 - Jan 30	Jan 30 - Feb 13	Feb 13 - Feb 27	Feb 27 - March 10	March 10 - March 24	March 24 - May 1	May 2 - May 9
Project Iniation and Planning	Whole Team							
Requirements Finalization	Whole Team	Whole Team						
System Design and Architecture	Whole Team	Whole Team						
Hardware Procurement and Setup		Luke and Sam Loecke	Luke and Sam Loecke					
Software Development - Initial Phase		Cole and Sam Nicklaus	Cole and Sam Nicklaus					
Integration of Hardware and Software			Cole	Cole				
Intial Testing and Debugging			Luke	Luke	Luke			
Development of Web Application				Sam Nicklaus	Sam Nicklaus	Sam Nicklaus		
Advanced Testing and Iteration				Sam Loecke	Sam Loecke	Sam Loecke		
Final Integration and Testing					Whole Team	Whole Team		
Preperation for Presentation						Whole Team	Whole Team	
Final Presentation Preperation								Whole Team

1. Tasks and Subtasks

a. Project Initiation and Planning (Whole Team)

i. Develop project charter (Cole)
ii. Define project scope (Luke)
iii. Identify key stakeholders and requirements (Sam Loecke)
iv. Establish project goals and objectives (Sam Nicklaus)

b. Requirements Finalization (Whole Team)

i. Finalize system requirements (Cole)
ii. Validate requirements with stakeholders (Luke)
iii. Document and sign-off requirements (Sam Loecke)
iv. Create requirements traceability matrix (Sam Nicklaus)

c. System Design and Architecture (Whole Team)

    i. Draft initial design sketches (Cole)
    ii. Outline system architecture (Luke)
    iii. Review and select technology stack (Sam Loecke)
    iv. Prepare design documents (Sam Nicklaus)

d. **Hardware Procurement and Setup (Luke and Sam Loecke)**
    i. Research and select OBD II Bluetooth adapter (Luke)
    ii. Source Raspberry Pi and peripherals (Sam Loecke)
    iii. Order and verify delivery of hardware components (Luke)
    iv. Set up initial hardware testbed (Sam Loecke)

e. **Software Development - Initial Phase (Cole and Sam Nicklaus)**
    i. Set up development environment (Cole)
    ii. Begin coding of basic functions (Sam Nicklaus)
    iii. Conduct peer reviews of initial code (Cole)
    iv. Test initial software modules (Sam Nicklaus)

f. **Integration of Hardware and Software (Cole)**
    i. Develop integration test plans (Cole)
    ii. Execute initial integration of hardware and software (Cole)
    iii. Debug integration issues (Cole)
    iv. Validate integrated system against requirements (Cole)

g. **Initial Testing and Debugging (Luke)**
    i. Design comprehensive test cases (Luke)
    ii. Execute system testing (Luke)
    iii. Record and analyze test results (Luke)
    iv. Debug identified issues and retest (Luke)

h. **Development of Web Application (Sam Nicklaus)**
    i. Design web app user interface (Sam Nicklaus)
    ii. Implement front-end components (Sam Nicklaus)
    iii. Develop back-end services (Sam Nicklaus)
    iv. Integrate web app with data API (Sam Nicklaus)

i. **Advanced Testing and Iteration (Sam Loecke)**
    i. Perform advanced system testing (Sam Loecke)
    ii. Gather user feedback on system usability (Sam Loecke)
    iii. Iterate on software based on feedback (Sam Loecke)
    iv. Refine system for improved performance (Sam Loecke)

j. **Final Integration and Testing (Whole Team)**
    i. Conduct final integration checks (Cole)
    ii. Perform end-to-end system tests (Luke)
    iii. Finalize documentation and user manuals (Sam Loecke)
    iv. Prepare system for deployment (Sam Nicklaus)

k. **Preparation for Presentation (Whole Team)**
    i. Develop presentation outline (Cole)
    ii. Create slides and visual aids (Luke)
    iii. Rehearse presentation delivery (Sam Loecke)
    iv. Gather system demonstration materials (Sam Nicklaus)

I. **Final Presentation Preparation (Whole Team)**
    i. Finalize presentation content (Cole)
    ii. Conduct final rehearsal (Luke)
    iii. Ensure all materials are ready (Sam Loecke)
    iv. Review feedback mechanisms for audience (Sam Nicklaus)

6) Risk Management

a) Primary Risks to Project Success

1. Technical Challenges with Components

Risk: Issues with the Raspberry Pi, Bluetooth OBD II adapter, or touchscreen functionality could hinder the project. This includes hardware malfunctions, compatibility issues, or limitations in processing capabilities.
Impact: These challenges could lead to delays in development, increased costs, or failure to meet project objectives.

2. Software Development Risks

Risk: Developing the software stack, including the integration of Python OBD library, Vue.js for the web app, and Flask API, might face complexities. Bugs, integration issues, and challenges in implementing features are potential risks.
Impact: Could lead to project delays, suboptimal performance of the system, and potential security vulnerabilities.

3. Connectivity and Data Transmission Issues

Risk: Ensuring reliable data transmission between the car's OBD port and the Raspberry Pi, and from the Pi to the web server could be challenging, especially if Bluetooth connectivity is interrupted due to the environment.
Impact: Could affect the real-time data feed, impacting the system's reliability and user experience.

4. Data Security and Privacy Concerns

Risk: Handling user data, especially vehicle diagnostics, is sensitive. There is a risk of data breaches or inadequate data protection measures.
Impact: Privacy concerns or data breaches can lead to legal issues and loss of user trust.

5. Power Supply and UPS Reliability

Risk: The uninterrupted power supply system must be robust enough to handle abrupt power changes (like when a car starts or stops). Failure in this system can lead to data loss or hardware damage.
Impact: Could result in system instability, data corruption, and potential hardware failure.

6. Team Dynamics and Human Resources

Risk: Loss of a team member, interpersonal conflicts, or mismanagement can significantly affect the project. This could include issues with task delegation, communication, or alignment of goals.
Impact: Poor team dynamics can lead to inefficiencies, reduced morale, and project delays or failure.

b) Risk Management Plan

1. Technical Challenges with Components

Mitigation Strategy: Establish testing protocols for all hardware components. This includes stress testing and compatibility checks.
Contingency Plan: In case of hardware failure or incompatibility, switch to alternative components if available. Design the system for modularity to facilitate easy swapping of parts.

2. Software Development Risks

Mitigation Strategy: Utilize an agile development approach with continuous integration and deployment. Regular code reviews, unit testing, and integration testing should be crucial parts of our workflow.
Contingency Plan: If certain software components prove problematic, we should be prepared to explore alternative libraries or frameworks. Maintain flexibility in design to accommodate such changes without major overhauls.

3. Connectivity and Data Transmission Issues

Mitigation Strategy: Implement the local storage of information to ensure minimal data loss when we are unable to reliably transmit data.
Contingency Plan: Develop a local database on the Raspberry Pi to store data temporarily when connectivity is lost.

4. Data Security and Privacy Concerns

Mitigation Strategy: Implement encryption and security protocols for data transmission and storage.

5. Power Supply and UPS Reliability

Mitigation Strategy: Choose high-quality UPS systems and conduct testing under various scenarios, including simulated power outages and fluctuations.
Contingency Plan: Have a backup power solution ready, and design the system to save critical data and enter a safe state before shutting down in case of power failure.

6. Team Dynamics and Human Resources

Mitigation Strategy: Establish clear roles and responsibilities within the team, along with regular check-ins and updates to ensure alignment and address any issues early.
Contingency Plan: In case of losing a team member or encountering significant interpersonal conflicts, have a plan for redistributing workload and responsibilities.

c) Contingency Plan

In the event of a campus emergency that limits access to labs, equipment, and in-person interactions (such as a COVID-19 pandemic scenario), we have taken the following into consideration:
Tools and Platforms: Utilize online collaboration tools such as Zoom for meetings, and Microsoft Teams for daily communication
Regular Meetings: Schedule regular virtual meetings for the entire team to discuss progress, challenges, and next steps. This ensures everyone is on the same page.
Documentation and Sharing: Use cloud storage solutions like Google Drive or Dropbox for sharing documents and project files.
Version Control: Utilize Git and a platform like GitHub for software development, allowing for collaborative coding and version tracking.
Remote Access to Equipment: Setup remote access to the raspberry pi allowing any individual to work on the hardware.

7) Budget

Our budget will be itemized by different sections with links to the equipment or software we plan to buy.

- Total Cost: $96.47
  - OBD Bluetooth Display: $7.99
  - Raspberry Pi: Already Have
  - Touch Screen: $58.49
  - Uninterrupted Power Supply: $29.99
  - Heroku Server Host: $300 in student credit

8) Team Considerations

a) Knowledge and Skills

The successful completion of the Smart OBD Diagnostic System (SODS) project relies on a diverse set of knowledge and skills, including areas such as software development, data analysis, and more. Our team collectively possesses a good mix of these competencies, derived from academic coursework and hands-on co-op/internship opportunities.

- **Software Development**

  Proficiency in Python is essential for leveraging the python-OBD library and developing backend server logic. Our experience was gained through advanced programming courses and individual coding projects.

  Expertise in Vue.js and modern web development (HTML5, CSS3, JavaScript ES6) is important for the creation of our user-friendly web app. Webdev experience was gained through on the job experience.
- **Electronics and Hardware Integration**

  Experience with Raspberry Pi and embedded systems is fundamental for integrating the OBD II Bluetooth adapter, touchscreen, and uninterrupted power supply. Our group has acquired these skills in past coursework and projects. Our group in IoT created a smart chicken coop which used some of the same systems.

  Understanding of automotive electronics, specifically OBD II systems, essential for real-time data collection and interpretation. Our group members have experience in the automotive industry dealing with the OBD codes.
- **Data Analysis**

  Skills in data processing and analytics, vital for interpreting vehicle data and providing insightful information on the web app. Our group members have taken database and data science courses.
- **Project Management**

  Leadership and organizational abilities, crucial for managing the project's timeline, deliverables, and team coordination. Strengthened through participation in group projects and leadership roles in student organizations.

b) Team Organization and Function

- Samuel Nicklaus will focus on software development and web app creation
- Luke Farmer and Sam Loecke will focus on Raspberry Pi configuration and hardware integration
- Cole Arduser will specialize in data analysis and server-side processing

Our team's academic pursuits and practical experiences have equipped us with a comprehensive skill set that aligns perfectly with the requirements of the SODS project. We are confident in successfully designing, developing, and implementing this innovative automotive diagnostic system.

c) Self-Sponsored Project

With our team doing a self-sponsored project, we will need to have more accountability as a team than would normally be provided by a sponsor. The extra accountability will be most obvious in a few areas: specifying system requirements, approving design decisions, and acceptance of deliverables. We have listed out these factors and how we plan to be accountable in these core areas.

- **Specifying System Requirements**

  We've laid out the key system requirements in this proposal, and we will consistently measure our progress against the requirements we've created in this document.

- **Approving Design Decisions**

  The subject matter expert in our group is Sam Loecke. He has lots of exposure and experience working with cars and OBD codes. With this in mind, we will have Sam have a final sign off on all of the design decisions. We won't change any design decisions unless we have Sam's approval

- **Acceptance of Deliverables**

  Similar to approving design decisions, Sam Loecke will have the final approval on all of the deliverables because he is the biggest stakeholder in the project.