A new way to monitor your vehicle, OBD-2GO! (subject to change) - Team 16

1. Team info:

Owen Koster,
Peyton DuPont
Tiernan Flanagan-Caldwell
Silas Jones

The main method of communication is Discord

2. Product description

Abstract:

A notorious source of headache for drivers has always been the dreaded check engine light. When it comes on, it leaves you with the anxiety of knowing something might be terribly wrong with your vehicle – Or it could be nothing. The OBD-2GO aims to make this system more transparent so drivers don't have to worry about what might be wrong with their vehicle. The OBD-2GO will be a system embedded into your vehicle that will actively monitor and warn you of issues with your vehicle in a more detailed manner than a simple check engine light. It will also enable you to monitor and record vehicle parameters while driving, so that you can know what shows signs of failure before the problem becomes much more expensive.

Goal:
Develop software that connects to the user's vehicle to receive live updates on desired vehicle sensors and diagnostic codes.

Current practices:
Currently requires an OBD-2 scanner or mechanic diagnostic tool, which are expensive and difficult to use.

Novelty:
Our software would be more user-friendly, with the end goal of being an embedded system inside the user's vehicle.

Effects:
Will provide more accessible monitoring of vehicle health, lower barrier to entry compared to other products on market.

Technical approach:
Python-obd to communicate with the vehicle, ELM327-emulator to emulate a vehicle, HTML, CSS, and JavaScript to build the user interface.

Risks:
Learning curve of gui and OBD-2 libraries
Misjudging complexity of tasks leading to time loss.
Possible incompatibilities with certain vehicles or other technologies.

Main Features:
Ability to show diagnostic codes while driving in an intuitive manner, with more serious codes being given priority.
A driving mode that only displays data to the user and a parked mode that will allow the user to configure the software.
Ability for the user to configure the data displayed based on available sensors while in park mode.
Ability to track diagnostics over time to compare and visualize
Save driving data to log

Stretch Goals:
Maintenance interval tracker that keeps track of standard maintenance items such as oil change intervals, and user-defined intervals for different maintenance items.
Develop app for mobile devices
Depending on limitations, contribute to python-OBD library
Include suggested fixes to problems

Project Milestone 2:

Use Cases:

Peyton's Use Case: Tracking data
Actors: User

1. Triggers: Tracker being turned on
2. Preconditions: The tracker is turned on
3. Postconditions: The data from the users drive is successfully logged
4. List of steps:
    - A user opens the app and turns on the tracker
    - The user then goes for a drive and parks at their destination
    - The user then filters the display to show the data they want to see
    - The user now sees their diagnostics from the drive with some visualizations
    - The user saves the data to their log
    - The user then goes to their log
    - The data from their recent drive is there
5. Extensions/variations of the success scenario: The user forgets to turn on the tracker
6. Exceptions: The data is incorrectly stored in log and lost

Owen's Case: Error Codes:
Actor: User

1. Trigger: ECU sets error code
2. Precondition: No current error codes in ECU
3. Postcondition: Error code is displayed to user
4. List of Steps:
    - App is on and connected
    - ECU sets an error code
    - Code is transmitted through OBD2 to the app
    - App reads code and determines severity
    - code is stored in app
    - if code is severe immediately alerts user
    - else code is displayed when user is stopped
5. Extensions:
   If no codes are set the program will not alert the user
6. Exceptions:
   ECU sets a code and it is never displayed to the user even when stopped

Silas' Use Case: Driving mode:
Actor: The User or the vehicle's velocity.

1. Trigger: If the vehicle starts moving (velocity exceeds 0-5mph), or the user selects driving mode.
2. Precondition: The app is not already in driving mode.

3. Postcondition: Driving mode is activated, simplifying the GUI so that the user can see a single graph and a few stats at a glance and see severe error code popups, but can't interact with the app at all, other than to leave driving mode if their speed is low enough.
4. List of Steps:
    - The app is running and the vehicle has been started.
    - The vehicle starts moving.
    - Driving mode is activated, and it switches interfaces.
    - The vehicle stops moving.
    - The user selects to exit driving mode, or the vehicle is turned off.
    - The app exits driving mode.

5. Extensions: The user can select to enter or exit driving mode while parked, and driving mode will automatically exit when the vehicle is turned off.
6. Exceptions / potential failures: Driving mode will rely mostly on the vehicle's velocity, so it will only work correctly as long as that data is accurate (which it really should be).

Tiernan's Use Case: Data Visualization and Tracking

Actors: User

1. Trigger: User will move to separate page that displays visualization, tracking will be kept in logs from driving history
2. Preconditions: There must be previous data to display, or currently reading data from OBD-2 scanner
3. Postconditions: User will see easily readable graph and statistics
4. List of steps:
    - User selects what to monitor and log
    - Data is generated or fetched
    - Data is rendered and displayed
    - Data is saved to log for historical tracking

5. Extensions: No data is stored, nothing to read. User doesn't save output
6. Exception: Data is unreadable for some reason. Too much data crashes

Non-Functional Requirements:

1. Response time and switching between pages should not be laggy, less than 0.5 seconds

2. Serious error codes should be displayed within 30-40 seconds after check engine light activates. Non serious error codes should be displayed within 1 minute of vehicle exiting driving mode
3. Data log should be accessible to a user outside of the program

External Requirements:

- The product must be robust against errors that can reasonably be expected to occur, such as invalid user input.

- **How the product will be installable:** We can make a demo version of the desktop app that does not require the emulator to run, and we will leave the full version available but it will require complicated driver installations on Windows.

- **The software will be buildable from source by others**, and the instructions for how to install the drivers for the emulator will be in the GitHub repository.

- **The scope of the project is reasonable:** We all believe this is true.

Team Process Description:

Describe your quarter-long development process.

- **We will use python-OBD** because it is the simplest way to pull data from the OBD2 port. We will use HTML, CSS and JavaScript for the user interface because we are the most familiar with these languages and we feel will have the most flexibility in building our project. We will use GitHub Projects to help track and manage our tasks. We will use this because it will put all of the information for our project in one place, and will make it easier to manage tasks.

- **Define and justify each team member's role:**

Owen: Project Management, Python backend. Project management is important because it will enable us to keep track of tasks and progress. The python backend is important because it is what links the car and the GUI together. I feel I am suited for this role because I have training and experience in project management, I also have experience working in python and with the python-OBD library.

Peyton: CSS/GUI. The GUI is important because the user needs a clean and easy way to interact with the software. I feel I am suited for this role because I have experience working with CSS from CS 290 Web Development.

Tiernan: GUI, Data management. Application must have user friendly design to appeal to users. For data to be usable, it must be saved in a clearly defined format that can be read by the user and also visualize data inside the application. I helped contribute to these roles in my previous group project in my web development class.

Silas: Testing, debugging, GUI. Discovering and patching bugs is an important part of developing any software, and I feel confident in my ability to review code and write good test cases. As for GUI, I expect once some of our software has been implemented, I'll be able to make changes and add to it even though I don't have much experience with web development.

- **Schedule of milestones for each member for each week:**

Github project tasks completed and ready for testing within each task's allotted time frame. Will be updated over time. Early tasks include defining data schema, allowing software to interact with emulator, figuring out necessary tech stack

- **Three major risks:**
    1. Misjudging the complexity of certain tasks
    2. Not all vehicles may support the features we design around
    3. Front and back and data specifications could be mismatched, make sure data schema is clearly defined early on

- **Describe at what point in your process external feedback (i.e., feedback from outside your project team, including the project manager) will be most useful and how you will get that feedback.**

External Feedback will be most useful in testing the user experience, and ensuring it is accessible and easy to use by most users. We will get this feedback by having classmates and friends try out the software and give feedback on the software.