Team Number: 44
Dr. Berk Gulmezoglu
Benjamin Muslic
Jonathan Duron
Mohamed Elaagip
William Griner
Thecarcloudproject@gmail.com
https://sdmay25-44.sd.ece.iastate.edu/

Executive Summary

Modern vehicles generate Diagnostic Trouble Codes (DTCs) when issues arise, but interpreting these codes remains a significant challenge for most drivers. This knowledge gap often leads to unnecessary expenses and uncertainty about vehicle repairs. FixIt addresses this problem by developing an intelligent mobile application that translates complex DTCs into easily understandable information. Key design requirements include:

FixIt

DESIGN DOCUMENT

- OBD-ll reader compatibility with all compliant vehicles
- ESP32 communication via Bluetooth for sending data
- AI-based code interpretation from community forums
- Real-time diagnostics
- User friendly interface
- Cost-effective solution

This design uses multiple technologies and approaches:

- ESP32 microcontroller for OBD-ll communication
- React Native for cross platform mobile development
- Bluetooth Low Energy (BLE) for communicating to devices
- Cloud services for data processing (pending implementation)

Our teams current progress:

- Successfully established ESP32 communication with OBD-ll scanner
- Developed and tested prototype mobile application
- Implemented Bluetooth communication between ESP32 and mobile app
- Created user interface for DTC's display and interpretation

Pending features and next steps:

- Implementation of user authentication system
- Web scraping functionality for gathering repair information
- User testing and feedback collection
- Interface refinement based on user experience

- Personal vehicle profile creation

While the core functionality meets basic requirements, several key features remain in development. The project's success will be measured through user testing and feedback, focusing on ease of use and how each of the codes are interpreted. The team maintains a structured timeline to ensure completion of remaining features. This solution promises to empower drivers with the knowledge needed to make better decisions about their vehicle maintenance and repairs, potentially saving both time and money while providing peace of mind. (262 words almost there)

Learning Summary

Development Standards & Practices Used

Hardware Standards

- OBD-II Protocol Standards (SAE J1979)
- ISO 15765-4 (CAN) communication protocol
- ISO 9141-2 protocol support
- Bluetooth Low Energy (BLE) specifications

Software Development Standards

- React Native development guidelines
- TypeScript coding standards
- Git version control practices
- ESP32 programming standards
- Mobile app UI/UX design principles

Testing Equipment

- ECUsim 2000 emulator for OBD-II simulation
- Standard protocol presets for testing:
- ISO protocols (9141-2, 14230-4, 15765-4)
- Protocol verification through monitoring (SOMM)
- Fault simulation capabilities (SF)

Engineering Standards

- CAN bus communication standards
- ESP32 hardware specifications
- Bluetooth 5.0 specifications
- OBD-II pin configuration standards
- USB communication protocols

Summary of Requirements

Functional Requirements

- OBD-II reader compatibility with all compliant vehicles
- ESP32 communication via Bluetooth for data transmission
- Real-time diagnostics and monitoring
- Vehicle health monitoring capabilities

Physical Requirements

- Hardware portability (OBD-II reader and ESP32)
- Wi-Fi and cloud integration
- ECUsim 2000 for development and testing

Technical Requirements

- Support for multiple OBD protocols:
  - ISO 15765-4 (CAN)
  - ISO 9141-2
  - ISO 14230-4
  - SAE J1850 PWM/VPW

User Interface Requirements

- Clean, modern, and intuitive layout
- Easy-to-understand icons
- Clear navigation structure
- Real-time data display capabilities

User Experience Requirements

- Easy device setup and pairing
- Fast data processing and display
- Seamless cloud integration
- Secure data transmission
- Clear error code interpretation

Economic Requirements

- Total hardware cost under $100
- Cost-effective solution for DIY users

Applicable Courses from Iowa State University Curriculum

Software Development

COM S 319: Software Construction and User Interfaces

- React Native mobile application development
- User interface design for diagnostic display
- Component-based architecture implementation

COM S 309: Software Development Practices

- Version control and collaborative development
- API integration for diagnostic data

- Software testing methodologies

Hardware and Protocols

CPR E 288: Embedded Systems

- ESP32 microcontroller programming
- Protocol implementation (ISO 15765-4, ISO 9141-2)
- Hardware-software integration

CPR E 489: Computer Networking and Data Communications

- Bluetooth communication protocols
- Data transmission and error handling
- Network security implementation

CPR E 490: Senior Design

- Project management and documentation
- System integration
- Requirements analysis and validation
- Testing and verification procedures

These courses provided the foundation for implementing the OBD-II diagnostic system, from low-level protocol handling to high-level user interface development.

New Skills/Knowledge acquired that was not taught in courses

Development Technologies

- React Native mobile development
- Bluetooth Low Energy (BLE) implementation
- Cross-platform mobile application development

Automotive Protocols

- OBD-II diagnostic protocols
- ISO 15765-4 (CAN) communication
- Multiple protocol handling (ISO 9141-2, ISO 14230-4)

Testing Tools

- ECUsim 2000 emulator usage
- Protocol verification techniques
- Fault simulation procedures

Hardware Integration

- ESP32 Bluetooth configuration
- OBD-II interface programming
- Multi-protocol hardware communication

Development Practices

- Mobile app state management
- Real-time data handling
- Protocol-specific error handling
- Cross-platform deployment strategies

Table of Contents

1. Introduction

1.1. Problem Statement

Modern vehicles are equipped with onboard diagnostic systems that generate codes when something goes wrong, but for most drivers, understanding these Diagnostic Trouble Codes (DTCs) is a frustrating challenge. These codes are often cryptic, lacking sufficient explanation or context. Drivers are left scouring the internet for answers or relying on mechanics, which can result in costly repairs or potential overcharging due to a lack of understanding.

In today's fast-paced world, people don't have time to research every potential issue with their car, and many are unsure if they're receiving an honest assessment from their mechanic. This knowledge gap affects both seasoned DIY enthusiasts who want to repair their own vehicles and everyday drivers who just want to avoid unnecessary expenses.

FixIt is here to solve this problem. Our AI-driven app translates those confusing DTC codes into easy-to-understand insights by gathering information from a wide range of trusted online sources and communities. It gives users the full story of what's wrong with their vehicle, right at their fingertips, empowering them to make informed decisions, whether they're performing the repair themselves or taking the car to a professional. In doing so, FixIt not only saves users time and money but also provides peace of mind, protecting them from inflated repair costs.

1.2. Intended Users

DIY Car Enthusiasts

These are car owners who enjoy working on their vehicles, often performing maintenance and repairs on their own. They are typically knowledgeable about car systems but may not always understand the complexity behind modern diagnostic codes. They value tools that enhance their ability to maintain their cars independently.

Needs: DIY car enthusiasts need a tool that translates complex DTC codes into actionable insights, allowing them to quickly diagnose issues without sifting through multiple online forums or manuals. They want to save money and avoid relying on mechanics for every repair.

Benefits: FixIt empowers DIYers by providing clear, accurate explanations of DTC codes, drawn from AI-sourced community insights. By simplifying this process, FixIt saves them time and ensures they have all the information they need to perform repairs efficiently. This connects to the broader problem of complex diagnostic codes and eliminates the hassle of researching each issue.

Everyday Car Owners

This group includes people who rely on their vehicles for daily transportation but have little technical knowledge about cars. They typically rely on mechanics for maintenance and repairs and may be susceptible to overcharging or unnecessary services due to a lack of understanding.

Needs: Everyday car owners need a tool that demystifies car diagnostics, enabling them to understand what's going wrong with their vehicle without needing to be mechanically inclined. They want peace of mind and confidence when discussing repairs with mechanics, knowing they're not being taken advantage of.

Benefits: FixIt ensures these users get straightforward, reliable information about their car’s issues. By doing so, the app reduces the risk of being overcharged by unscrupulous mechanics and enables them to make informed decisions about repairs. This value aligns with the problem of mechanic scams and helps to protect users from inflated repair costs, giving them more control over their car maintenance.

Car Sellers

Car sellers, whether individuals or small dealerships, need to keep their vehicles in good working order to sell them for the best price. They may want to diagnose and fix issues themselves or at least understand the vehicle's condition to communicate effectively with potential buyers.

Needs: Sellers need a reliable and quick way to assess the health of a vehicle before putting it up for sale. They want to identify and fix any issues beforehand to maximize the sale price or provide potential buyers with accurate diagnostic information to build trust.

Benefits: FixIt helps car sellers by quickly identifying problems and offering solutions, ensuring their vehicle is in top condition for sale. It also helps them provide detailed diagnostics to buyers, increasing the likelihood of a successful sale by offering transparency. This addresses the broader issue of ambiguous diagnostic codes and builds trust between seller and buyer, fostering smoother transactions.

Connection to Problem Statement:

All these user groups—DIYers, everyday car owners and car sellers —benefit from FixIt because it tackles the core problem of DTC code complexity. By transforming confusing diagnostic information into clear, actionable insights, FixIt saves time, money, and effort for all users while ensuring transparency and preventing mechanic fixes quote scams. The app bridges the gap between professional-level diagnostics and everyday car maintenance, ensuring that users are equipped with the knowledge they need to make informed decisions about their vehicles.

2. REQUIREMENTS, CONSTRAINTS, AND STANDARDS

2.1. REQUIREMENTS & CONSTRAINTS

Functional Requirements:

OBD-II Reader Compatibility:
The system must be able to read DTC codes from all OBD-II compliant vehicles.
(Constraint: OBD-II compatibility required for vehicles manufactured after 1996).

ESP32 Communication:
The ESP32 must establish a stable connection with the user's phone personal hotspot via Wi-Fi to transmit data to the cloud.

AI-Based Code Interpretation:
The system must interpret DTC codes using AI to gather insights from community forums, databases, and other online resources.

Real-Time Diagnostics:
The app must provide real-time analysis of car issues, displaying results as soon as codes are read from the vehicle.

User Notifications:
The app must send notifications to users when critical car issues are detected that require immediate attention.

Vehicle Health Monitoring:
The system should continuously monitor car performance metrics and alert the user to potential preventative maintenance opportunities.

Physical & Resource Requirements:

Hardware Portability:
The OBD-II reader and any other required side hardware must be small and portable enough to be carried on the go

Wi-Fi and Cloud Integration:
The system requires a stable Wi-Fi connection via the user's phone personal hotspot to send data to the cloud for processing.
(Constraint: Requires continuous hotspot availability during operation).

ESP32 Power Supply:
The ESP32 must be powered by the car's OBD-II port or an internal battery, eliminating the need for external power sources.

Aesthetic Requirements

User-Friendly App Design:
The app's interface must be aesthetically pleasing with a clean, modern, and intuitive layout, incorporating easily recognizable icons and color schemes for different car statuses (e.g., green for good, red for urgent).

Intuitive App Design:
The app's interface must be easy for the user to understand and navigate.

User Experiential Requirements:

Easy Setup:
The Wi-Fi OBD dongle and ESP32 must be easy to set up, with clear instructions for pairing the device with the user's phone and connecting it to the cloud.

Fast Data Processing:
The system must send DTC data to the cloud and return results within a few moments

Seamless User Experience:
The user experience should be seamless, with the app automatically receiving results from the cloud once the ESP32 has sent diagnostic data.

Secure User Experience:
The user's data should be secure, which can be achieved through various cloud networking protocols.

Economic Requirements:

Cost-Effective Solution:
The Wi-Fi OBD dongle and ESP32 system must be affordable for DIY users and casual car owners. (Constraint: Total hardware cost must not exceed $100).

2.2. ENGINEERING STANDARDS

What Engineering standards are likely to apply to your project? Some standards might be built into your requirements (e.g., many projects uses 802.11 ac wifi standard) and many others might fall out of design.

Q1) Browse the videos available on the IEEE Standards University site (https://www.standardsuniversity.org/videos/(Links to an external site.)). Select and watch the following video: https://www.standardsuniversity.org/video/standards-education-young-professional/ (Links to an external site.), as well as 2 other videos of your own choosing. Note that these videos are generally only 2-3 minutes long.

Briefly describe, in your own words, the importance of engineering standards.

-

IEEE standards serve as essential building blocks for technological development, with over 1,300 standards that ensure safety, reliability, and interoperability of products and systems worldwide. These consensus-driven guidelines, developed by an amazing organization, cover crucial areas including electrical safety, energy efficiency, communication protocols, data security, and system interoperability. The standards enable technologies from different manufacturers to work together seamlessly, particularly evident in widely-used protocols like Wi-Fi (IEEE 802.11) and renewable energy integration (IEEE 1547). Through a rigorous development process involving industry professionals, academics, and government representatives, IEEE standards create a level playing field that encourages innovation while maintaining safety and reliability across industries. These standards are particularly vital in emerging fields like artificial intelligence, quantum computing, and 5G, where they continue to shape the future of technology while ensuring regulatory compliance and promoting global technological advancement.

Q2) Browse the IEEE standards website: http://standards.ieee.org/findstds/index.html (Links to an external site.). Select a sub-category as appropriate for your project (possibilities include "Computer Technology", "Software and Systems Engineering", and "Communications"). Filter by 'Active Standards'.

Select at least 3 standards that appear, based on their descriptions, to have relevance to your project. Most of these standards can be found on the IEEE Xplore digital library (http://ieeexplore.ieee.org/Xplore/home.jsp (Links to an external site.)), which you have access to while using a computer on-campus. If off-campus, you can use the ISU VPN and library portal to get access (http://instr.iastate.libguides.com/ecpe (Links to an external site.)

If your project requires *software standards*, please review this source: https://www.iso.org/standards.html

Review each of the selected standards. These can be quite lengthy documents, and you are not expected to read through them in their entirety.

Describe, entirely in your own words, what each standard is about and what it is intended to accomplish.

-

Software life cycle processes - Maintenance
IEEE/ISO/IEC 14764-2021

ISO/IEC/IEEE 14764 establishes a comprehensive framework for software maintenance throughout a system's lifecycle, recognizing maintenance as a vital component that consumes substantial resources. The standard covers both pre-delivery activities (like planning and logistics) and post-delivery activities (such as help desk support and upgrades). The standard defines four main types of maintenance:

- Corrective: Fixing identified problems
- Adaptive: Maintaining software usability as environments change
- Preventive: Addressing potential issues proactively
- Perfective: Enhancing performance and maintainability

The process framework encompasses strategy development, planning, problem analysis, modification implementation, and quality assurance. It provides guidance for both internal teams and external providers, addressing everything from minor updates to major modifications. The standard emphasizes careful planning and execution, particularly for systems that must maintain continuous operation. Key aspects include comprehensive quality assurance measures, thorough testing requirements, and robust configuration management practices. The standard mandates proper documentation of all maintenance activities, including problem reports, modification requests, and implementation details. It establishes clear procedures for change control to maintain system stability and reliability throughout the maintenance process.

Software testing -- Part 2: Test processes

IEEE/ISO/IEC 29119-2-2021

This international standard (ISO/IEC/IEEE 29119-2) establishes a comprehensive framework for software testing processes that can be applied across any organization or software development project. It provides a structured approach to testing through three main process layers: organizational test processes, test management processes, and dynamic test processes. The standard aims to help organizations standardize and improve their testing practices by defining clear processes, roles, and deliverables.

The standard is designed to be flexible and adaptable, allowing organizations to either fully implement all processes or tailor them to their specific needs. It emphasizes risk-based testing approaches, helping organizations prioritize testing efforts based on identified risks and potential impacts. The framework supports both traditional and agile development methodologies and can accommodate various types of testing, including manual, automated, functional, and non-functional testing.

A key goal of the standard is to provide those responsible for software testing with the necessary information and structure to effectively manage and perform testing activities across their organization. It accomplishes this by defining specific process requirements, outcomes, and activities while allowing for different levels of conformance. The standard also aligns with other important software development and quality standards, making it easier for organizations to integrate it into their existing processes.

The standard's practical value lies in its ability to help organizations establish consistent testing practices, improve test planning and execution, and better manage testing resources and activities. By following these standardized processes, organizations can potentially reduce testing risks, improve software quality, and create more predictable testing outcomes.

Systems and software assurance -- Part 4: Assurance in the life cycle
IEEE/ISO/IEC 15026-4-2021
This standard (ISO/IEC/IEEE 15026-4:2021) provides guidance on how to ensure and demonstrate that critical system and software properties meet their required levels of assurance throughout their lifecycle. It focuses on two main aspects: achieving specific assurance claims about a system or software, and proving that these claims have been met through proper documentation and evidence. The standard introduces two key process views - one for systems and one for software - that work alongside existing lifecycle process standards (ISO/IEC/IEEE 15288 and 12207). These views help organizations integrate assurance activities into their regular development and maintenance processes. The standard explains how to identify assurance claims (critical properties that need to be guaranteed), gather evidence to support these claims, and construct valid arguments demonstrating that the claims have been achieved.

The document is designed to be used in various contexts: for agreements between suppliers and acquirers, for regulatory compliance, or for internal development process improvement. It provides specific guidance on how to manage assurance-related activities throughout the project lifecycle, including planning, risk management, configuration control, and quality assurance.

What makes this standard particularly valuable is its focus on maintaining assurance throughout the entire system or software lifecycle - not just during initial development. It emphasizes the importance of documenting and maintaining assurance information, managing changes that might affect assurance claims, and ensuring that critical properties continue to be met even as systems evolve over time.

The standard serves as a practical guide for organizations needing to demonstrate that their systems or software meet specific critical requirements, particularly in areas where failure could have serious consequences, such as safety, security, or reliability.

Q3) After reviewing some of the technical details of the three published standards, do you believe it to have relevance to your project? Why or why not? Be specific.
We believe all of them have relevance to our project. The “Maintenance” is relevant because routinely fix identified issues within our code, and proactively plan for issues that may arise in the future. The “Test processes” standard has relevance in our project as it’s essential to our success that we validate our product’s ability to function properly. And lastly, The “Assurance in the life cycle” as we frequently document our results and how those results are achieved. This is very useful for proving that specific claims have been reached.
Q4) Review with your team the standards that each of you have selected. What other standards did some of your team members choose that are different?
TODO

Q5) What modifications do you intend to make to your project design to incorporate these standards?
To incorporate the maintenance standard, we will start to perform QA testing on our app, so that we can proactively identify and fix bugs before we move to production.
To incorporate the testing standard, we will begin to clearly define our testing goals before incorporating a new feature. This will help us have a clear vision of what results we want from the feature, and how we can test the new feature.

To incorporate the assurance in the life cycle standard, we will continue to make documentation that adequately reflects how our systems function. We will also continue to obtain records of our results to display our progress.

3 Project Plan

3.1 Project Management/Tracking Procedures

Management Style:
We are adopting a **hybrid approach** that combines agile and waterfall methodologies. The waterfall aspect ensures that we first focus on foundational tasks, such as setting up the embedded hardware and basic communication between the OBD dongle, ESP32, and cloud infrastructure. This foundational work establishes a stable base, which is essential for later development. Once the hardware is functioning, the team will switch to agile sprints, iterating on the app interface, backend development, AI integration, and cloud service improvements. This hybrid approach aligns well with our goals by creating a solid hardware base and allowing for flexible, iterative improvements on the software side.

So, to summarize:

Waterfall Phase: Focus on foundational tasks such as setting up the embedded hardware and basic communication between the OBD dongle, ESP32, and cloud infrastructure.

Agile Phase: Once hardware is functioning, switch to agile sprints for iterative development of the app interface, backend, AI integration, and cloud service improvements.

Progress Tracking:
Our team will track project progress using **Git** for code versioning and **GitHub** for team collaboration. We will also hold regular **advisor meetings** and team check-ins to review milestones and address challenges. Communication tools like **Slack** will help maintain constant team updates, while Trello (or GitHub Projects) may be used for task tracking to organize sprints and milestones.

We have two semesters to complete the FixIt project. By the end of the first semester, our goal is to have a barebones prototype that shows all essential communication between the OBD-II dongle, ESP32, cloud, and mobile app. The app skeleton will be established with basic features to demonstrate functionality. This first-semester deliverable provides a foundation for further development and refinement in the second semester.

3.2 Task Decomposition

Hardware Setup and Configuration

- Choose and configure the OBD-II dongle and ESP32 for reliable connectivity.
- Test initial data collection from OBD and data transmission to ESP32.

Basic Communication and Data Transmission

- Establish WiFi connection between ESP32 and the phone hotspot.
- Set up data transfer from ESP32 to the cloud.

Frontend App Design

- Design user interface elements for easy DTC code interpretation.
- Develop notification and alert systems for real-time diagnostic updates.

Backend and Cloud Infrastructure

- Set up server infrastructure for handling data sent by ESP32.
- Implement database and AI modules to analyze and interpret diagnostic data.

AI Integration

- Develop algorithms to interpret DTC codes and gather insights from community data.
- Integrate machine learning models for predictive maintenance insights.

Testing and Debugging

- Test hardware in a vehicle with an active check engine light to validate data accuracy.
- Conduct iterative debugging on the frontend and backend as new features are integrated.

Final Integration and Validation

- Verify that all hardware, software, and AI functionalities work as expected.
- Conduct user testing and refinement based on feedback.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

1. Hardware Communication

- Milestone: Achieve reliable OBD-II data reading and transmission
- Metric: 80% reliability in OBD-II data reading within the first 3 months
- Evaluation Criteria:

- Test with at least 1 vehicle model
- Conduct 10 consecutive read attempts
- Measure success rate of data transmission to ESP32

2. Cloud Infrastructure

a) Basic Scaling

- Milestone: Implement basic scaling for user demand
- Metric: Successfully handle 1,000 simulated users
- Evaluation Criteria:
  - Use a load testing tool to simulate user load
  - Measure response times under various load conditions
  - Ensure CPU utilization remains below 80% during peak load

b) Failover Handling

- Milestone: Implement basic failover for availability issues
- Metric: Maintain 95% uptime during simulated failures
- Evaluation Criteria:
  - Simulate failures and measure system response
  - Verify data consistency after failover
  - Ensure minimal data loss during failover process

3. Frontend App Development

- Milestone: Complete MVP app with core functionalities
- Metric: Achieve 60% user satisfaction in initial user testing within 6 months
- Evaluation Criteria:
  - Conduct usability tests with at least 5 potential users
  - Measure task completion rates for key features
  - Collect and analyze user feedback through surveys

4. AI Diagnostic Feature

- Milestone: Implement basic AI-driven DTC interpretation
- Metric: Achieve 70% accuracy in interpreting common DTCs within 6 months
- Evaluation Criteria:
  - Test against a database of at least 50 known DTCs
  - Compare AI interpretations with expert diagnoses
  - Measure precision and recall for different categories of DTCs

5. Data Aggregation

- Milestone: Implement basic data collection pipeline
- Metric: Process and store data from 100 simulated vehicles within 1 week
- Evaluation Criteria:
  - Measure data ingestion rate
  - Verify data integrity in the database
  - Assess query performance for common scenarios

3.4 PROJECT TIMELINE/SCHEDULE

1. **Lack of Active Check Engine Light in Test Vehicles**

    - Probability: 0.6
    - Impact: High
    - Mitigation Strategies:
    - Use a simulated environment with software like OBD-II Simulator to generate predictable diagnostic trouble codes (DTCs) for controlled testing.
    - Acquire a dedicated test vehicle with known issues that consistently trigger the check engine light.
    - Partner with a local mechanic or auto shop to access vehicles with active DTCs for real-world testing.

2. **Competition from Existing OBD-II Diagnostic Tools**

    - Probability: 0.7
    - Impact: Medium
    - Mitigation Strategies:
    - Differentiate FixIt by emphasizing its AI-driven insights and predictive maintenance capabilities, which provide more value than standard code readers.
    - Highlight FixIt's user-friendly interface and community features that help users understand and resolve issues without extensive technical knowledge.
    - Conduct market research to identify underserved niches or unique features that set FixIt apart from competitors.

3. **Hardware Communication and Reliability Issues**

    - Probability: 0.5
    - Impact: High
    - Mitigation Strategies:
    - Develop comprehensive testing procedures that cover various scenarios, including edge cases and failure modes.
    - Use high-quality, automotive-grade components for the OBD-II dongle and ESP32 to ensure durability and reliability.
    - Implement robust error handling and logging to quickly identify and diagnose communication issues.
    - Maintain a stock of backup hardware components to minimize downtime in case of failures during development or testing.

4. **Delays in Cloud Backend Integration**

    - Probability: 0.4
    - Impact: Medium
    - Mitigation Strategies:

- Adopt a modular architecture that decouples the frontend, backend, and hardware components, allowing development to progress independently.
- Prioritize backend development tasks and allocate additional resources if necessary to keep the project on schedule.
- Implement clear API contracts and interfaces to minimize dependencies and enable parallel development efforts.
- Regularly sync with the team to identify and address any integration challenges proactively.

3.5 Risks and Risk Management/Mitigation

Protocol Communication (P=0.7)

- Risk: OBD-II protocols may fail or provide inconsistent readings
- Mitigation:
  - Use ECUsim 2000 for development and testing
  - Implement multiple protocol support (ISO 15765-4, ISO 9141-2)
  - Fallback to basic protocol if advanced features fail

ESP32 Compatibility (P=0.6)

- Risk: ESP32 may not handle all OBD protocols efficiently
- Mitigation:
  - Implement protocol verification using SOMM command
  - Use error handling for CAN communication issues
  - Consider alternative microcontrollers if needed

Real-time Performance (P=0.6)

- Risk: Slow response times
- Mitigation:
  - Optimize communication protocols
  - Implement efficient data handling
  - Use local caching when possible

3.6 Personnel Effort Requirements

Task	Assigned Team Member(s)	Est. Hours
Create backend and cloud authentication and authorization flows for user sign in, create account, forgot	Will	15

password, change password, etc.		
Create frontend authentication and Authorization flows for user sign in, create account, forgot password, change password, etc.	Mohamed and Jonathan	15
Create a production and development environment, to enhance the developer experience by giving developers a realistic environment to test in.	Will	7
Create the ability to build and take down our prod and devl environments as needed, as we don't need either of them running 24/7	Will	4
Create various webscrapers to aggregate relevant data to pre-prompt our LLM with. These webscrapers will be run on cloud compute, so I'll need to set up the cloud environment for these as well.	Will	20
Organize database schemas and tables to store the webscraped data	Will	2
UI Drafting and Design along with general app layout along with iterative fixes and updates	Mohamed	12

Hardware setup, including OBD-II dongle and ESP32 configuration, testing, and troubleshooting.	Ben	22
App development, including implementing UI designs, integrating with backend services, and ensuring cross-platform compatibility.	Mohamed and Jonathan	40
Implement notification and alert systems for real-time diagnostic updates.	Jonathan	10
Develop algorithms to interpret DTC codes and gather insights from community data.	Will and Mohamed	30
Conduct hardware testing in a vehicle with an active check engine light to validate data accuracy.	Ben	15
Perform iterative debugging on the frontend and backend as new features are integrated.	Mohamed and Jonathan	20
Verify that all hardware, software, and AI functionalities work as expected during final integration.	Ben and Jonathan	10
Documentation of technical specifications, user guides, and project progress.	Jonathan	15

3.7 Other Resource Requirements

Technical Resources

- **OBD-II Readers**: Devices compatible with OBD-II systems for reading and transmitting Diagnostic Trouble Codes (DTCs).
- **ESP32 Modules**: Microcontroller units for establishing Wi-Fi connections and data transmission to the cloud.
- **Testing Equipment**: Tools for validating the functionality of hardware and software components.
- **Software Tools**: Access to development platforms for app programming, AI model training, and user interface design.

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or may be indirectly affected (e.g., solution is implemented in their communities)	Increasing/reducing exposure to pollutants and other harmful substances, increasing/reducing safety risks, increasing/reducing job opportunities
Global, cultural, and social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	Development or operation of the solution would violate a profession's code of ethics, implementation of the solution would require an undesired change in community practices
Environmental	What environmental impact might your project have? This can include indirect effects, such as deforestation or unsustainable practices related to materials manufacture or procurement.	Increasing/decreasing energy usage from nonrenewable sources, increasing/decreasing usage/production of non-recyclable materials
Economic	What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader	Product needs to remain affordable for target users, product creates or diminishes opportunities for economic advancement, high

	economic effects on communities, markets, nations, and other groups.	development cost creates risk for organization

4.1.2 Prior Work/Solutions

Include relevant background/literature review for the project (cite at least 3 references for literature review in IEEE Format. See link: https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf)

– If similar products exist in the market, describe what has already been done

– If you are following previous work (e.g., a previous senior design project), cite that and discuss the **advantages/shortcomings**

– Note that while you are not expected to "compete" with other existing products / research groups, you should be able to differentiate your project from what is available. Thus, provide a list of pros and cons of your target solution compared to all other related products/systems.

Detail any similar products or research done on this topic previously. Please cite your sources and include them in your references. All figures must be captioned and referenced in your text.

4.1.3 Technical Complexity

Provide evidence that your project is of sufficient technical complexity. Use the following metric or argue for one of your own. Justify your statements (e.g., list the components/subsystems and describe the applicable scientific, mathematical, or engineering principles)

1. The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles –AND–
2. The problem scope contains multiple challenging requirements that match or exceed current solutions or industry standards.

4.2 Design Exploration

4.2.1 Design Decisions

List key design decisions (at least three) that you have made or will make. These can include, but are not limited to, materials, subsystems, physical components, sensors/chips/devices, physical layout, features, etc. Describe how you made/will make these decisions and how they have affected or are likely to affect project success.

4.2.2 Ideation

For at least one design decision, describe how you ideated or identified potential options (e.g., lotus blossom technique). Describe at least five options that you considered.

4.2.3 Decision-Making and Trade-Off

Demonstrate the process you used to identify the pros and cons or trade-offs between each of your ideated options. You may wish you include a weighted decision matrix or other relevant tool. Describe the option you chose and why you chose it.

4.3 PROPOSED DESIGN

4.3.1 Overview

Provide a high-level description of your current design. This description should be understandable to non-engineers (i.e., the general public). Describe key components or sub-systems and how they contribute to the overall design. You may wish to include a basic block diagram, infographic, or other visual to help communicate the overall design.

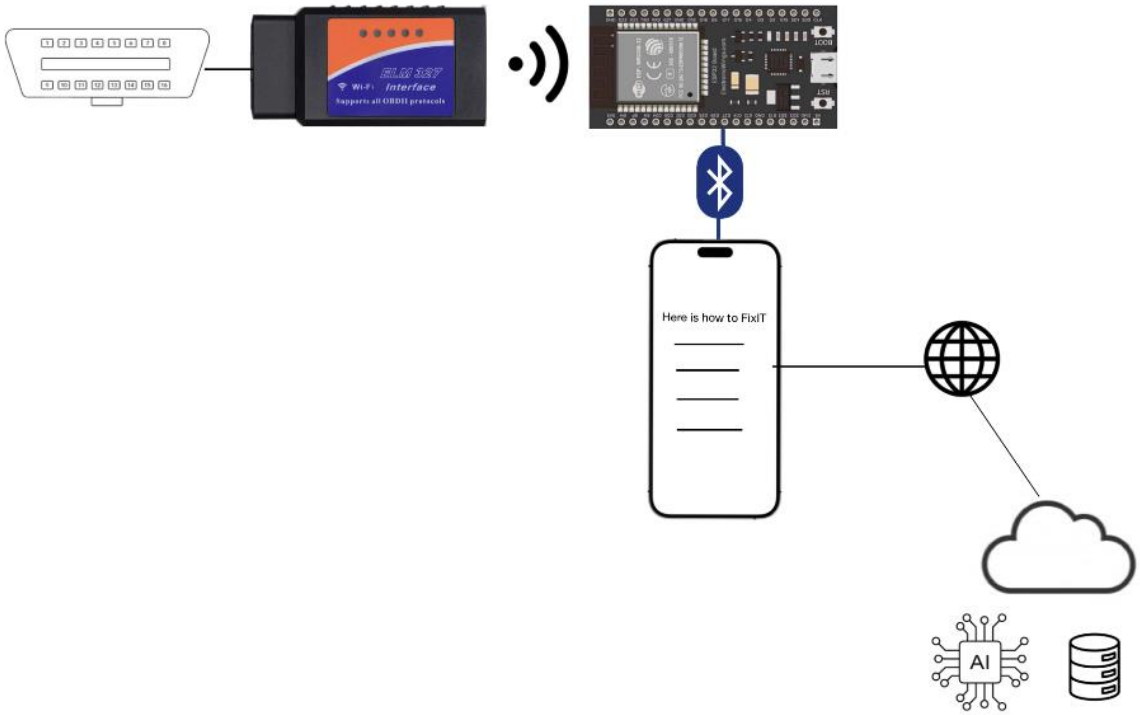
4.3.2 Detailed Design and Visual(s)

Provide a detailed, technical description of your design, aided by visualizations. This description should be understandable to peer engineers. In other words, it should be clearly written and sufficiently detail such that another senior design team can look through it and implement it.

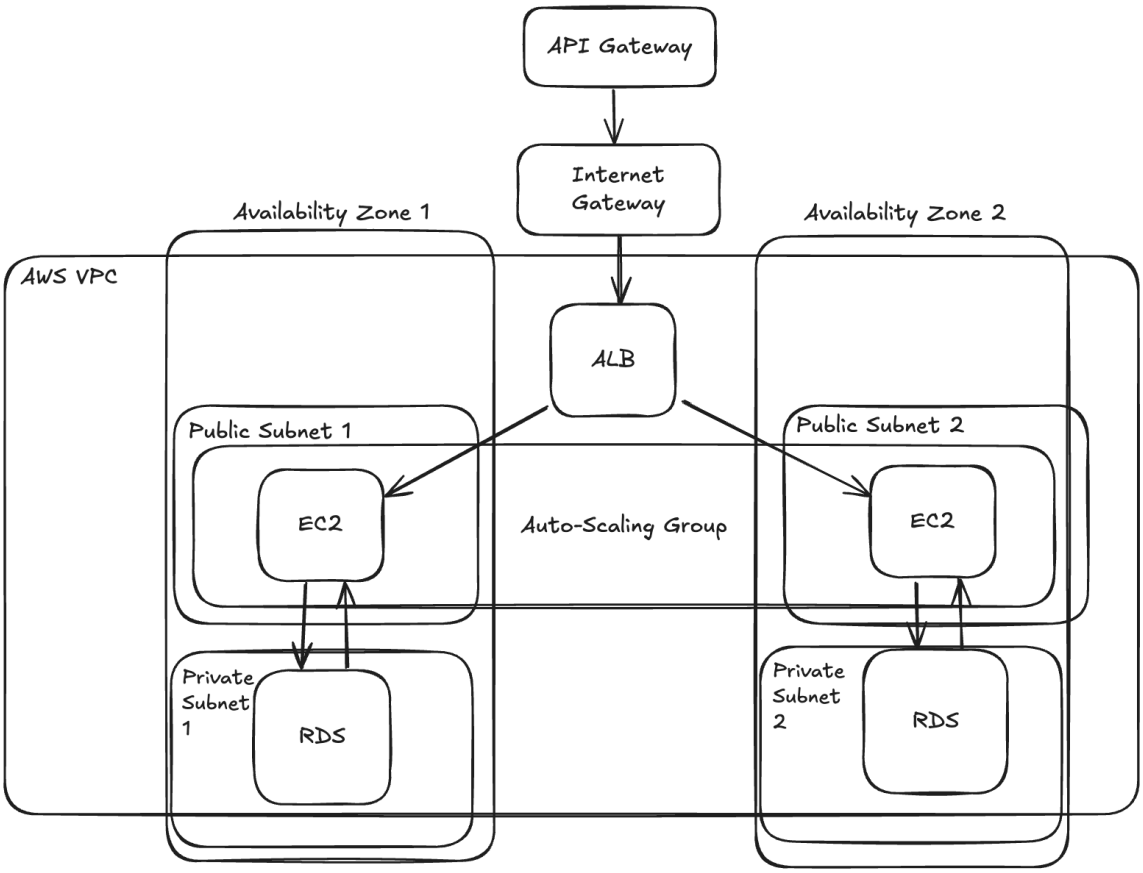
The description should include a high-level overview written for peer engineers. This should list all sub-systems or components, their role in the whole system, and how they will be integrated or interconnected. A visual should accompany this description. Typically, a detailed block diagram will suffice, but other visual forms can be acceptable.

The description should also include more specific descriptions of sub-systems and components (e.g., their internal operations). Once again, a good rule of thumb is: could another engineer with similar expertise build the component/sub-system based on your description? Use visualizations to support your descriptions. Different visual types may be relevant to different types of projects, components, or subsystems. You may include, but are not limited to: block diagrams, circuit diagrams, sketches/pictures of physical components and their operation, wireframes, etc.

Overall communication Design:



Cloud Design:



4.3.3 Functionality

Describe how your design is intended to operate in its user and/or real-world context. What would a user do? How would the device/system/etc. respond? This description can be supplemented by a visual, such as a timeline, storyboard, or sketch.

4.3.4 Areas of Concern and Development

How well does/will the current design satisfy requirements and meet user needs?
Based on your current design, what are your primary concerns for delivering a product/system that addresses requirements and meets user and client needs?
What are your immediate plans for developing the solution to address those concerns?
What questions do you have for clients, TAs, and faculty advisers?

4.4 Technology Considerations

Describe the distinct technologies you are using in your design. Highlight the strengths, weakness, and trade-offs made in technology available. Discuss possible solutions and design alternatives.

4.5 Design Analysis

Discuss what you have done so far, i.e., what have you built, implemented, or tested? Did your proposed design from 4.3 work? Why or why not? Based on what has worked or not worked (e.g., what you have or haven't been able to build, what functioned as expected or not), what plans do you have for future design and implementation work? For example, are there implications for the overall feasibility of your design or have you just experienced build issues?

5 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

The testing plan should connect the requirements and the design to the adopted test strategy and instruments. In this overarching introduction, give an overview of the testing strategy and your team's overall testing philosophy. Emphasize any unique challenges to testing for your system/design.

In the sections below, describe specific methods for testing. You may include additional types of testing, if applicable to your design. If a particular type of testing is not applicable to your project, you must justify why you are not including it.

When writing your testing planning consider a few guidelines:

- Is our testing plan unique to our project? (It should be)
- Are you testing related to all requirements? For requirements you're not testing (e.g., cost related requirements) can you justify their exclusion?
- Is your testing plan comprehensive?
- When should you be testing? (In most cases, it's early and often, not at the end of the project)

5.1 Unit Testing

Frontend Testing:

- Unit tests for individual UI components that display DTCs and their interpretations
- Testing component rendering, user interactions, and state management
- Tools: Jest and React Native Testing Library for component testing

Backend Testing:

- Unit tests for API endpoints that handle DTC data and LLM communication
- Testing request/response handling and data validation
- Tools: pytest for Python backend testing

5.2 Interface Testing

Interfaces in our Design:

Mobile App Frontend

- User Interface components for displaying DTCs and diagnostic information
- Bluetooth interface for communicating with ESP32 device
- API interface for communicating with backend services

Backend Services

- REST API endpoints handling DTC data processing
- Database interface for storing vehicle and user information
- LLM integration interface for code interpretation

Interface Testing Approach

API Integration Testing

- Testing data flow between frontend and backend services
- Validating request/response formats for DTC processing
- Testing error handling and recovery scenarios
- Tools: Bruno for API testing, Jest for frontend API integration tests

Bluetooth Communication Testing

- Testing data transmission between ESP32 and mobile app
- Validating connection stability and error recovery
- Testing boundary conditions for data packets

Database Interface Testing

- Testing data persistence and retrieval operations
- Validating data integrity across system components
- Testing concurrent access patterns
- Tools: SQLMap for database testing

5.3 Integration Testing

Critical Integration Paths

Mobile-to-ESP32 Integration

- Bluetooth communication between mobile app and ESP32 device
- Critical due to requirement for real-time diagnostics and OBD-II compatibility
- Testing using BLE simulation tools and real device testing
- Validation of data packet integrity and connection stability

ESP32-to-OBD Integration

- Communication between ESP32 and vehicle's OBD-II port
- Critical for obtaining accurate DTCs from all compliant vehicles
- Testing through hardware-in-the-loop simulation
- Verification of proper code reading across different vehicle models

Integration Points

Frontend-Backend Communication

- REST API endpoints for DTC processing and interpretation
- Validation of request/response cycles and error handling

LLM Integration

- Communication between backend and AI service for code interpretation
- Verification of interpretation accuracy and response times

Testing Approach

Point-to-Point Testing

- Each integration point tested individually for reliability
- Focus on data transformation and routing between systems
- Tools: Bruno for API testing, BLE testing frameworks

End-to-End Testing

- Test complete flow from OBD reading to user display
- Tools: Automated testing frameworks, system integration test suites

5.4 System Testing

System Level Testing Strategy

OBD-II Compliance Testing

- Testing compatibility with vehicles from 1996 onwards
- Verification of DTC code reading accuracy across different manufacturers
- Testing hardware connection stability
- Tools: OBD-II simulators

Wi-Fi Communication Testing

- ESP32 to phone hotspot connection reliability
- Data transmission success rate measurement
- Connection recovery after interruptions
- Tools: Wi-Fi analyzers, network stress testing tools

Cloud Integration Flow

- Cloud processing and AI interpretation
- Response delivery to mobile app
- Tools: AWS testing tools, cloud monitoring services such as Datadog

Hardware Validation

Power Management

- OBD-II port power supply testing
- ESP32 power consumption measurement
- Battery life testing (if applicable)
- Tools: Power consumption analyzers

Physical Requirements

- Size and portability verification
- Environmental condition testing
- Durability testing
- Tools: Environmental test chambers

Security and Data Protection

Authentication Testing

- User data security validation
- Cloud security protocol testing
- Privacy compliance verification
- Tools: Security testing tools such as mend and checkmarx

User Experience Testing

Interface Usability

- Setup process validation
- Navigation flow testing
- Visual design compliance
- Tools: Jest

Performance Metrics

- Response time measurements against requirements
- System reliability under various conditions
- Cost validation against $100 hardware constraint
- Tools: Performance monitoring suites, cost analysis tools

5.5 Regression Testing

Regression testing for our system focuses on preserving critical functionalities driven by our core requirements: OBD-II code reading, Wi-Fi communication between ESP32 and phone, real-time diagnostics delivery, and accurate AI-based code interpretation. We can have quality regression testing by integrating existing unit, integration, and end-to-end tests into a CI/CD pipeline that runs these tests automatically when new code is pushed. Test results should be logged and archived, allowing for historical comparison and trend analysis to identify gradual performance degradation.

5.6 Acceptance Testing

Functional Requirements Demonstration

Vehicle Compatibility

- Live demonstration with multiple vehicle makes and models
- Real-time DTC code reading and interpretation
- Verification of compatibility with post-1996 vehicles

Non-Functional Requirements Validation

Cost Effectiveness

- Hardware cost breakdown presentation
- Component cost optimization demonstration
- Total system cost validation against $100 constraint

Client Involvement Strategy

Phase 1:

- Client observes system setup and basic functionality
- Hands-on testing with their own vehicles
- Immediate feedback collection and implementation

Phase 2: Extended Testing

- Client receives beta version for extended testing period
- Daily usage monitoring and feedback collection
- Performance metrics gathering in real-world conditions

Phase 3: Final Validation

- Performance metrics review against acceptance criteria
- Comprehensive demonstration of all requirements
- Final adjustments based on client feedback

5.7 Security Testing (if applicable)

Authentication and Authorization

User Access Control

- Testing role-based access controls
- Use secure password policies

Data Security

- Testing encryption of vehicle diagnostic data in transit

- Testing API endpoint security and access controls

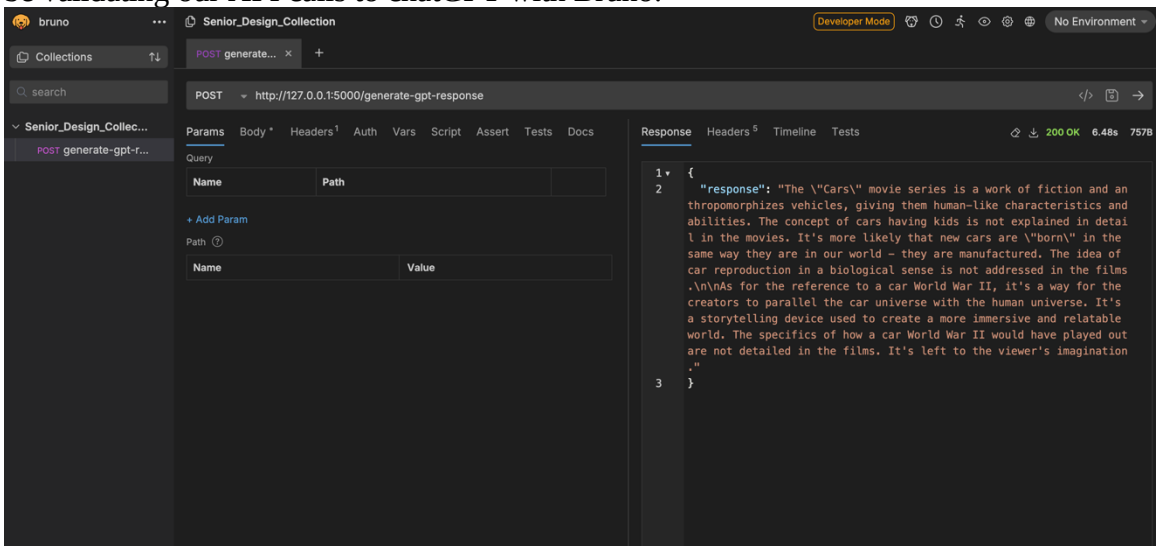
Network Security Communication Security

- Testing Bluetooth connection security between mobile app and ESP32
- Validation of secure Wi-Fi communication protocols
- Testing API endpoint SSL/TLS implementation
- Testing the API for rate limit abuse attacks
- Verification of secure cloud service communications with our virtual private cloud

5.8 RESULTS

What are the results of your testing thus far? Include any numerical, graphical, or qualitative testing results here? How do they demonstrate compliance with the requirements or addressing user needs? Use a summary narrative to discuss what you've learned and what next steps need to be taken.

Our results thus far have been basic implementations of unit and integration testing, that have allowed us to verify the basics of our software working. One such example would be validating our API calls to chatGPT with Bruno:

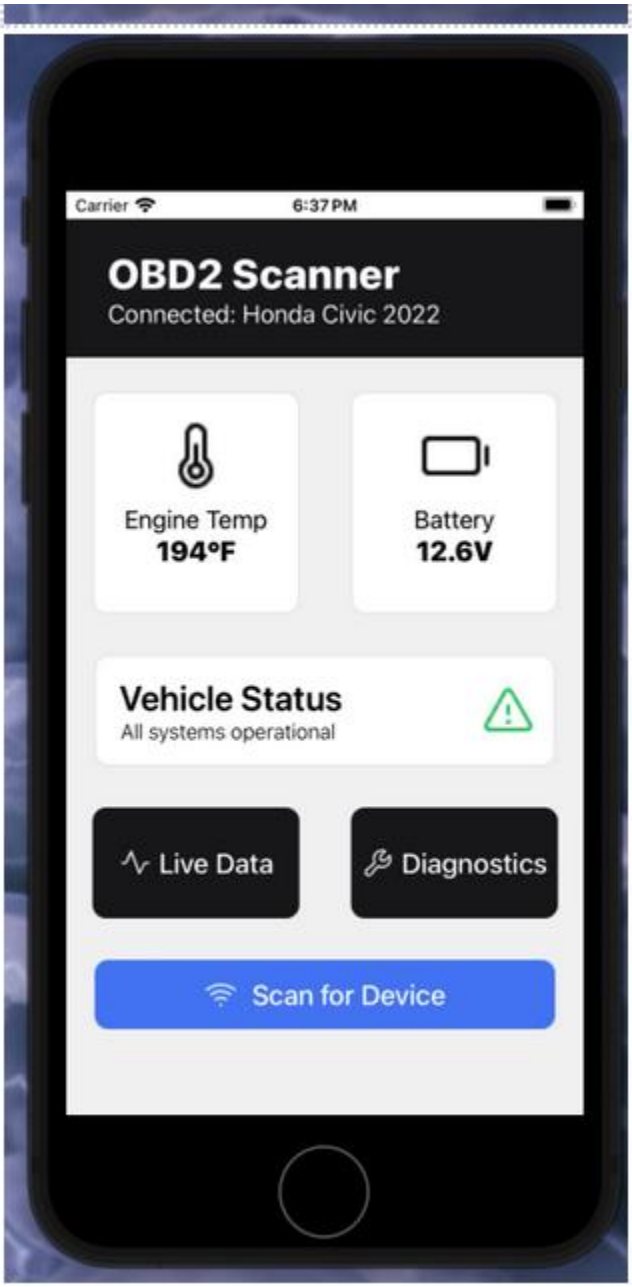


Our progress with unit and integration testing thus far has allowed us to show our compliance with our requirements by displaying that the chain of communication within our app functions properly and can serve users information. Our testing has also helped us proactively realize when a new code will break something, when integrating new features. This helps us provide for our users, as it ensures we have a working app in production.

6 Implementation

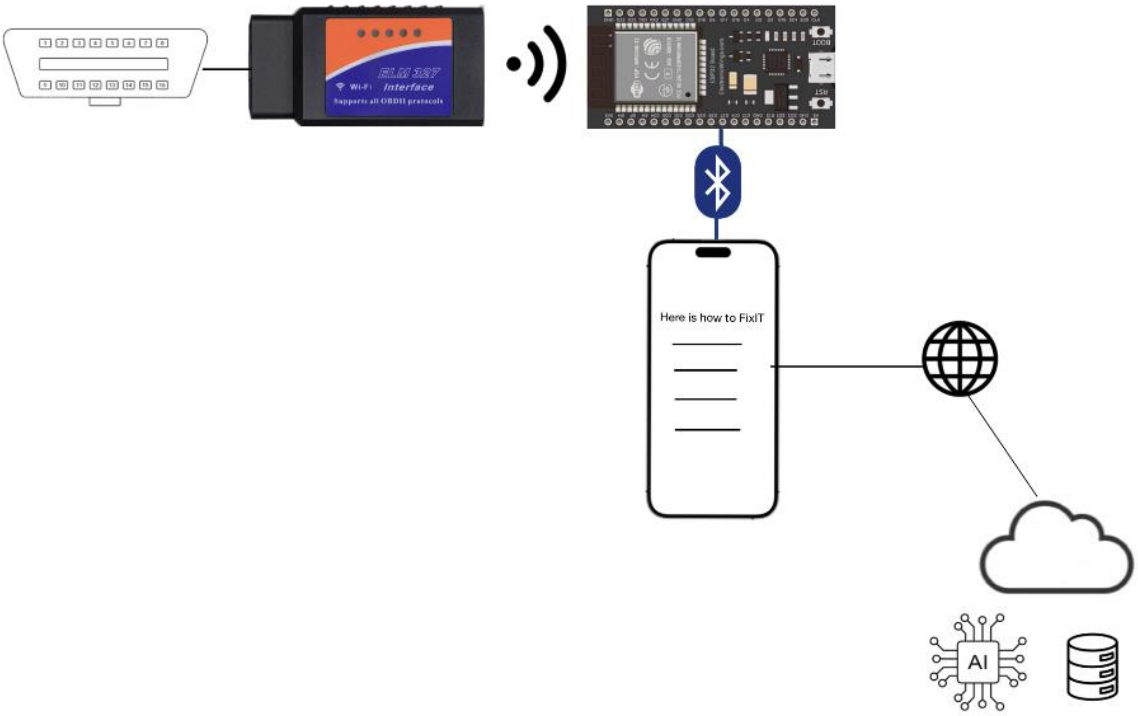
Describe any (preliminary) implementations of your design thus far. Support any general, descriptive text with relevant images. If your project has inseparable activities between design and implementation, you can list them either in the Design section or this section.

On the frontend, we've developed and iterated on a simple and effective design. Our design goals are for our UI to be easy to understand and visually appealing.



On the backend, we've implemented basic Flask endpoints that allow communicate from the mobile app, to our application server, to an LLM model, back to the application server, and back to the mobile app.

In our hardware, we've created a system that can receive DTC codes from the user's car, and send them to our mobile application.

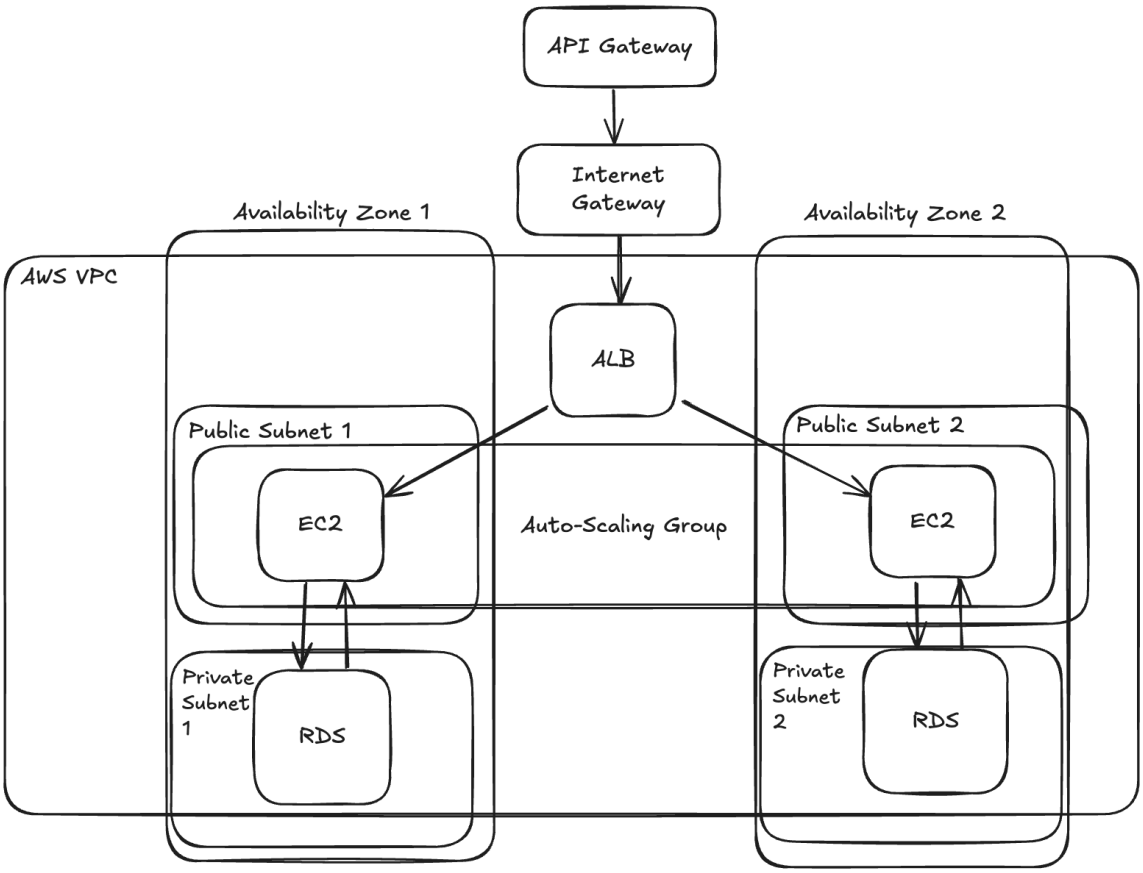


In the cloud, we've created a scalable and secure infrastructure.

Cloud

We've created a scalable, highly available, and secure infrastructure that can handle many of our compute needs. We've also focused on the developer experience by implementing a system that prevents developers from simultaneously modifying our cloud infrastructure, preventing accidental destruction of infrastructure.

Architecture diagram:



Scalability:

We use an autoscaling group to automatically increase and our EC2 instances with our traffic. As traffic increases, we create more EC2 instances. As traffic decreases, we remove our EC2 instances. Here are our conditions for scaling:

Scale-out conditions (if ANY of these metrics are true):

1. CPU Utilization > 70%
2. Memory Utilization > 80%
3. Request Count > 1000/minute/instance
4. Queue Length > 100 messages
5. Concurrent Connections > 500/instance

Scale-out protection

- Maximum 3 instances per AZ
- prevent runaway scaling and control costs

Scale-in conditions (when ALL metrics are below):

1. CPU Utilization < 30%
2. Memory Utilization < 40%
3. Request Count < 300/minute/instance
4. Queue Length < 20 messages
5. Concurrent Connections < 100/instance

Scale-in protection

- Prevent accidental termination of essential instances

-Essential instance == critical to our application's functionality. Want to protect from being terminated during scale-in events. For example the instance might be running important background jobs or handling long-running processes.

Preventing concurrent modification of cloud infrastructure:

In simple terms it's like this:

Scenario 1: Someone Else is Running Terraform

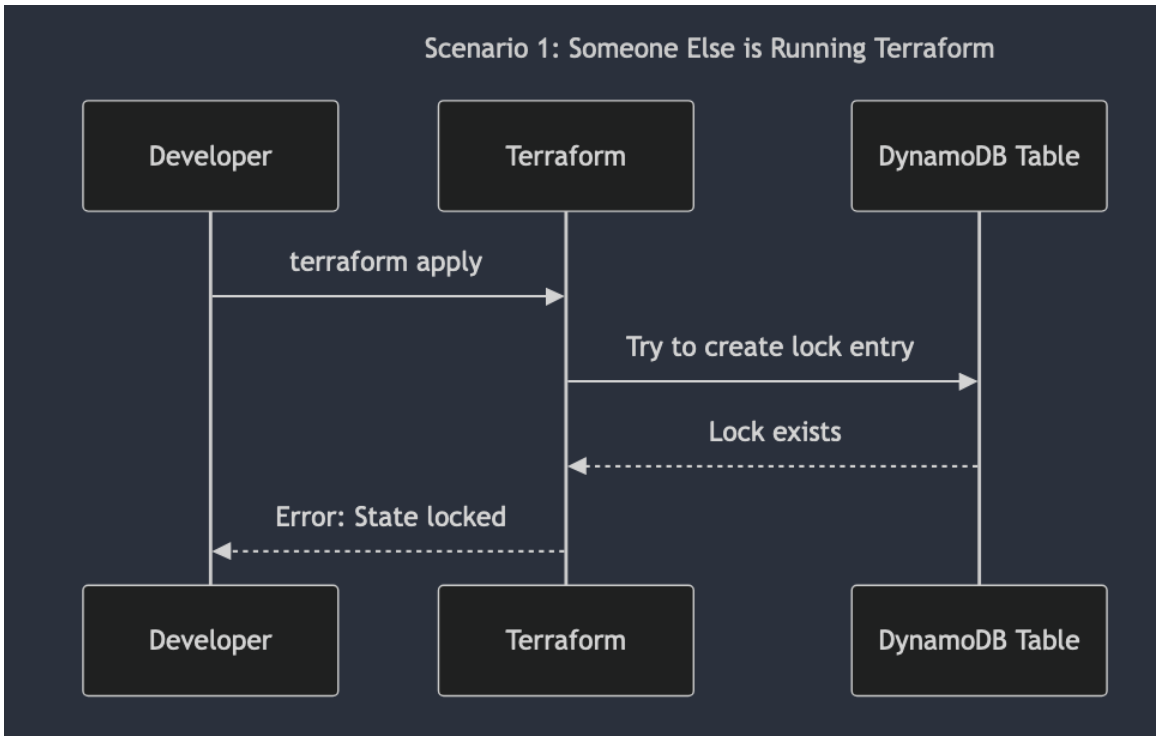
Imagine trying to use a bathroom:

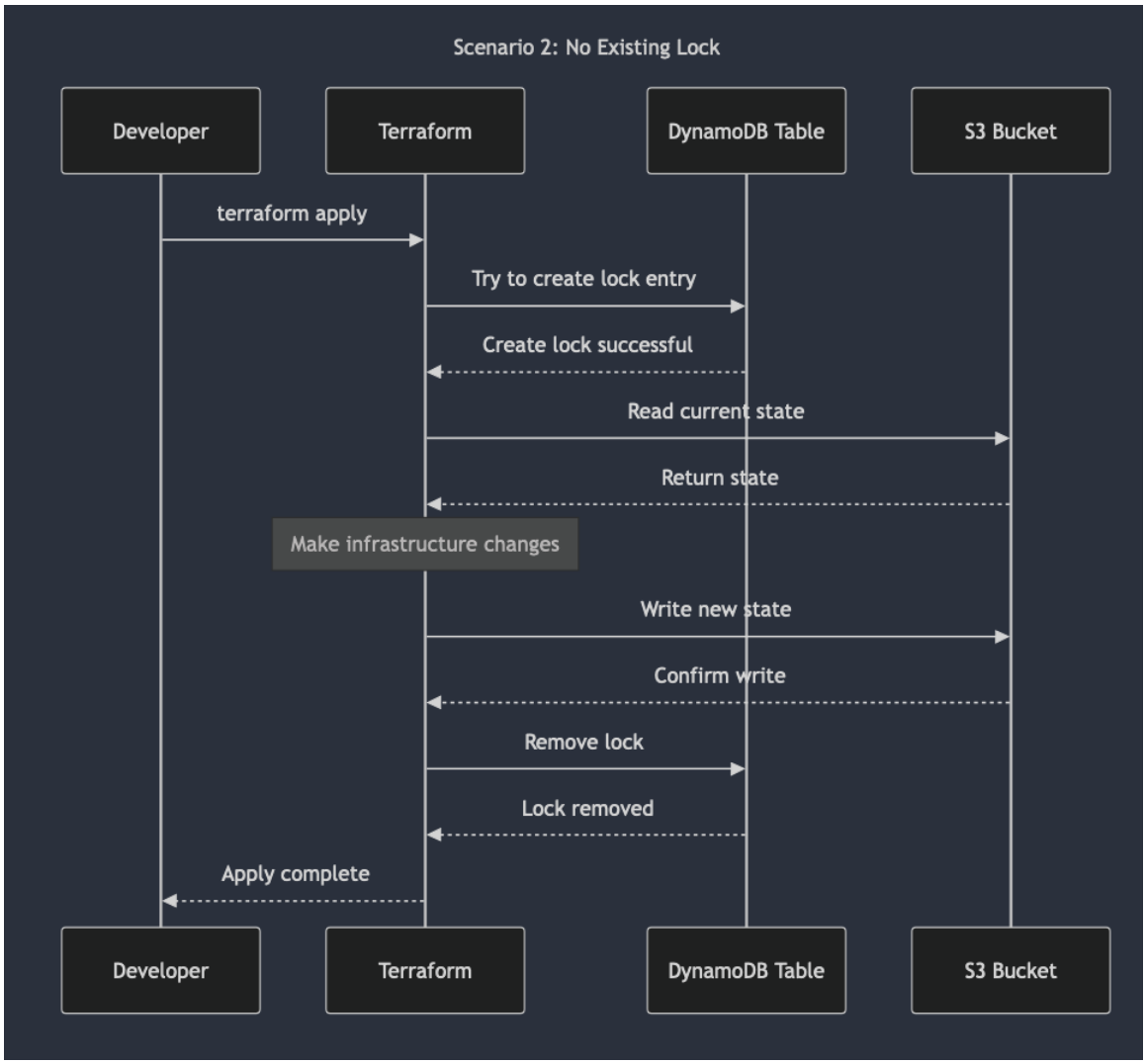
- You try to open the door (Developer runs terraform apply)
- You see it's locked (DynamoDB shows lock exists)
- You get told "Occupied!" (Terraform returns error: State locked)
- You have to wait until the person is done

Scenario 2: No Existing Lock

Like successfully using a bathroom:

- You check if it's occupied (Terraform checks DynamoDB)
- It's free, so you lock the door (Terraform creates lock)
- You look in the mirror to see what needs fixing (Terraform reads current state from S3)
- You make your changes (Terraform makes infrastructure changes)
- You check the mirror again to confirm changes (Terraform writes new state to S3)
- You unlock the door when done (Terraform removes lock)
- You leave, letting others know it's free (Terraform completes)


Scenario 1: Someone Else is Running Terraform
Developer
Terraform
DynamoDB Table
terraform apply
Try to create lock entry
Lock exists
Error: State locked
Developer
Terraform
DynamoDB Table



Scalability:

We enhance scalability with our ALB (Elastic Load Balancer), as our ALB provides
Traffic Distribution

- Evenly distributes incoming requests across EC2 instances
- Spans multiple Availability Zones for redundancy
- Automatically routes traffic to healthy instances only

Auto Scaling Integration

- Works with EC2Auto Scaling group
- Dynamically adds/removes EC2 instances based on demand
- Automatically registers new instances
- Removes failed instances from rotation

Health Management

- Continuously monitors instance health
- Routes traffic away from unhealthy instances
- Enables instance maintenance without service interruption
- Maintains service availability during scaling events

Security:

Virtual Private Cloud

- VPC creates an isolated network environment (10.1.0.0/16)
- Public subnets (10.1.1.0/24, 10.1.2.0/24):
    - Connected to Internet Gateway
    - Host public-facing resources like load balancers
    - Allow controlled inbound/outbound internet access
- Private subnets (10.1.3.0/24, 10.1.4.0/24):
    - No direct internet access
    - Host sensitive resources like databases
    - Protected from direct external access

Security Groups

Security groups allow or disallow traffic based on where the traffic is coming from and thus are essential for security.

ALB Security Layer

- Inbound: Allows all traffic (0.0.0.0/0) because mobile users connect from anywhere
- Outbound: Only allows traffic to ECS security group
- Acts as first line of defense, like a bouncer checking IDs

EC2 (compute) Security Layer

- Inbound: Only accepts traffic from ALB security group
- Outbound: Limited to:
    - HTTPS (443) for container updates
    - Specific ports for RDS access
- Protects application containers from direct external access

RDS (Database) Security Layer

- Inbound: Only allows database connections from ECS security group
- Outbound: No rules needed (private subnet blocks internet access)
- Database completely isolated from external access

API Gateway

The API Gateway essentially acts as a security checkpoint, filtering and validating all requests before they reach our VPC resources (Load Balancer → ECS → RDS), providing a crucial layer of defense for our infrastructure.

Authorization and Authentication

- Integrates with Cognito for user authentication
- Validates tokens and API keys
- Blocks unauthorized requests before VPC entry
- Enforces fine-grained access control

Traffic Protection

- Rate limiting prevents DDoS attacks
- Request validation blocks malicious inputs
- WAF integration stops common web exploits
- Throttling controls per-user usage

Security Architecture Benefits

- Centralizes security enforcement
- Handles SSL/TLS termination
- Monitors API usage patterns
- Logs security events
- Protects backend services from direct exposure

Application Load Balancer

Our ALB acts a revers proxy, which creates a secure entry point where traffic is inspected and filtered before reaching our application servers.

ALB Security Benefits as Reverse Proxy Traffic Protection
- Hides internal network structure and server IPs
- Terminates SSL/TLS connections
- Filters malicious traffic
- Blocks unauthorized access
- Prevents DDoS through rate limiting

Request Validation
- Validates HTTP headers
- Integrates with Cognito for authentication
- Performs health checks on targets
- Blocks malformed requests
- Enforces connection limits

Access Control
- Centralizes security policies
- Routes traffic based on paths/hosts/IPs
- Controls request queuing and throttling
- Manages connection pooling
- Enables circuit breaking for failure protection

The ALB acts like a security bouncer:
- Positioned in public subnet
- Evaluates all incoming requests

Forwards only legitimate traffic to ECS
Protects backend services from direct exposure
Maintains security while enabling scalability

7 Ethics and Professional Responsibility

Use this section to describe your considerations of engineering ethics and professional responsibility. Most importantly how are you defining engineering ethics and professional responsibility in the context of your project and what steps are you taken to ensure ethical and responsible conduct. Each section references one type of ethical/professional responsibility considerations. You may also use this introductory section to note any overarching ethical philosophy among your team.

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	Definition	IEEE Code Item	Team Implementation
Technical Competence	Maintaining and improving technical skills and knowledge	"maintain and improve our technical competence"	Implemented proper protocol handling and verification using ECUsim commands (SP, SOMM) for accurate diagnostics

Area of Responsibility	Definition	IEEE Code Item	Team Implementation
Technical Quality	Ensuring high standards in development and testing	"to acknowledge and correct errors promptly"	Using ECUsim testing protocols and error handling (CAN ERROR, INVALID PARAM) for reliable operation
Procedural Quality	Following established processes and standards	"to seek, accept, and offer honest criticism of technical work"	Following OBD-II protocols and standards, implementing proper command sequences
Professional Impact	Understanding how work affects others	"to avoid real or perceived conflicts of interest whenever possible"	Ensuring accurate DTC interpretation through proper protocol implementation
Societal Impact	Considering broader implications of work	"to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies"	Empowering users with accurate vehicle diagnostics

Strong Performance Area

Our team excels in Technical Quality implementation. We've established robust error handling and verification procedures using ECUsim commands like SOMM for monitoring, SF for fault testing, and proper protocol initialization. This ensures reliable diagnostic data transmission and interpretation.

Area Needing Improvement

Procedural Quality requires improvement. While we follow basic protocols, we need to implement more comprehensive documentation of command sequences and error handling procedures. Future improvements should include:

- Structured testing procedures for each protocol
- Documentation of all error scenarios

- Implementation of automated testing sequences
- Regular code reviews and protocol verification

7.2 Four Principles

Context Area	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Technical	Accurate DTC interpretation through proper protocol implementation	Error handling prevents vehicle system damage	Users can enable/disable monitoring (SOMM command)	Equal access to diagnostic information across vehicle types
Economic	Cost-effective diagnostic solution	Prevents unnecessary repair expenses	User control over diagnostic depth	Affordable access to vehicle diagnostics
Environmental	Early detection of emissions issues	Proper protocol verification reduces testing waste	User choice in maintenance timing	Equal access to emissions data
Social	Community knowledge sharing	Protection of vehicle systems through proper commands	User control over data sharing	Equal access to diagnostic tools

Important Pair: Technical-Beneficence
Our project excels in accurate DTC interpretation and protocol handling. Using standardized commands (SP, SOMM, SF) ensures reliable diagnostic information, benefiting users through precise vehicle health monitoring. Implementation of proper error handling and verification procedures maintains data integrity.

Lacking Pair: Technical-Justice
Our project currently shows limitations in technical justice due to protocol restrictions and compatibility issues. Some vehicles with specific protocols (like J1850 PWM/VPW) may have limited access to diagnostic features compared to those using ISO 15765-4 (CAN). This creates unequal access to diagnostic capabilities across different vehicle models.
To address this limitation:

1. Implement support for multiple protocols (SP command)
2. Develop protocol conversion capabilities
3. Create standardized interpretation methods across all supported protocols
4. Ensure equal diagnostic depth regardless of protocol used

7.2 VIRTUES

Team Virtues

Technical Excellence

- Importance: Ensures reliable diagnostic information
- Implementation:
    - Thorough protocol testing using ECUsim commands
    - Proper error handling (CAN ERROR, INVALID PARAM)
    - Systematic approach to protocol verification

Transparency

- Importance: Builds trust within team and with users
- Implementation:
    - Clear documentation of protocol implementations
    - Open communication about technical limitations
    - Sharing of testing results and error conditions

Individual Virtues Demonstrated

Ben Muslic (Hardware Engineer)

- Demonstrated Virtue: Precision
- Importance: Critical for accurate diagnostic readings
- Demonstration: Careful implementation of protocol commands and error handling

Jonathan Duron (Software Engineer)

- Demonstrated Virtue: Adaptability
- Importance: Essential for handling various protocols and error conditions
- Demonstration: Implementing flexible protocol switching and error recovery

Virtues to Improve

Will Griner

- Virtue to Develop: Patience
- Importance: Needed for thorough testing and debugging
- Plan: Implement more comprehensive testing procedures

Mohammed

- Virtue to Develop: Communication
- Importance: Essential for team coordination
- Plan: Better documentation of code changes and protocol implementations

8 Closing Material

8.1 CONCLUSION

Multiple setups were tried regarding the FixIt project. The end goal is to make it as easy as possible for the everyday user.

Arduino Uno with MCP2515 CAN Module

Purpose:

- Initial setup to directly read OBD-II data from the car.

Setup:

- **Hardware**:
  - Arduino Uno
  - MCP2515 CAN Bus Module
  - Male OBD connector
- **Software**:
  - Configured the MCP2515 module at 500 kbps.
  - Displayed live data like RPM and DTC codes on the Arduino serial monitor.

Outcome:

- Successfully retrieved OBD-II data but lacked advanced features like wireless connectivity.
- Too much hardware and wiring.
- Unsafe

2. WiFi OBD Dongle with ESP32

Purpose:

- To enhance the system with WiFi and cloud communication capabilities.

Setup:

- **Hardware**:
  - ESP32 microcontroller
  - WiFi OBD-II dongle
- **Communication**:
  - The ESP32 connected to the WiFi OBD-II dongle to retrieve data.

Outcome:

- Established communication between components.
- Viewed DTC codes locally

Challenges:

- Dependency on stable internet for functionality.

- User had to abandon WiFi connection to connect to ESP32
- Too much hardware
- Would have to use phone as a “router” for any web communication

3. BLE (Bluetooth Low Energy) with ESP32

Purpose:

- To simplify user connectivity and reduce WiFi dependency.

Setup:

- **Hardware**:
  - ESP32 (BLE-enabled)
  - WiFi OBD-II dongle
- **Communication**:
  - The FixIt app connects to the ESP32 via Bluetooth.
  - The ESP32 communicates with the WiFi OBD-II dongle to gather diagnostic data.

Outcome:

- Simplified user interaction using BLE.
- Reduced WiFi dependency for the user-facing app.

Challenges:

- The team is currently unable to decipher OBD traffic directly within the app, relying on the ESP32 for intermediate processing.

Summary of Current Status:

- **Current Workflow**:
  - The FixIt app connects to the ESP32 via Bluetooth, and the ESP32 communicates with a WiFi OBD reader to gather diagnostic data.
  - The ESP32 handles OBD data parsing, but the app cannot yet process raw OBD traffic directly.
- **Key Challenges**:
  - Deciphering and processing OBD-II data directly in the app to potentially eliminate intermediate hardware in the future.
  - Need batteries or power supplies to run hardware outside of laptop
  - If continuing to use hardware, a 3D printed casing is needed to combine everything into one piece
  - App has starting UI and can connect to read DTC, however no AI functionality yet exists

8.2 References

List technical references and related work / market survey references. Do professional citation style (ex. IEEE). See link: https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf

[1] Adam Varga, “How to hack your car | Part 1 - The basics of the CAN bus,” *YouTube*, Mar. 19, 2020. https://www.youtube.com/watch?v=cAAzXM5vsi0 (accessed Dec. 08, 2024).

[2] PowerBroker2, “ELMduino,” *GitHub*, Jul. 07, 2022. https://github.com/PowerBroker2/ELMduino

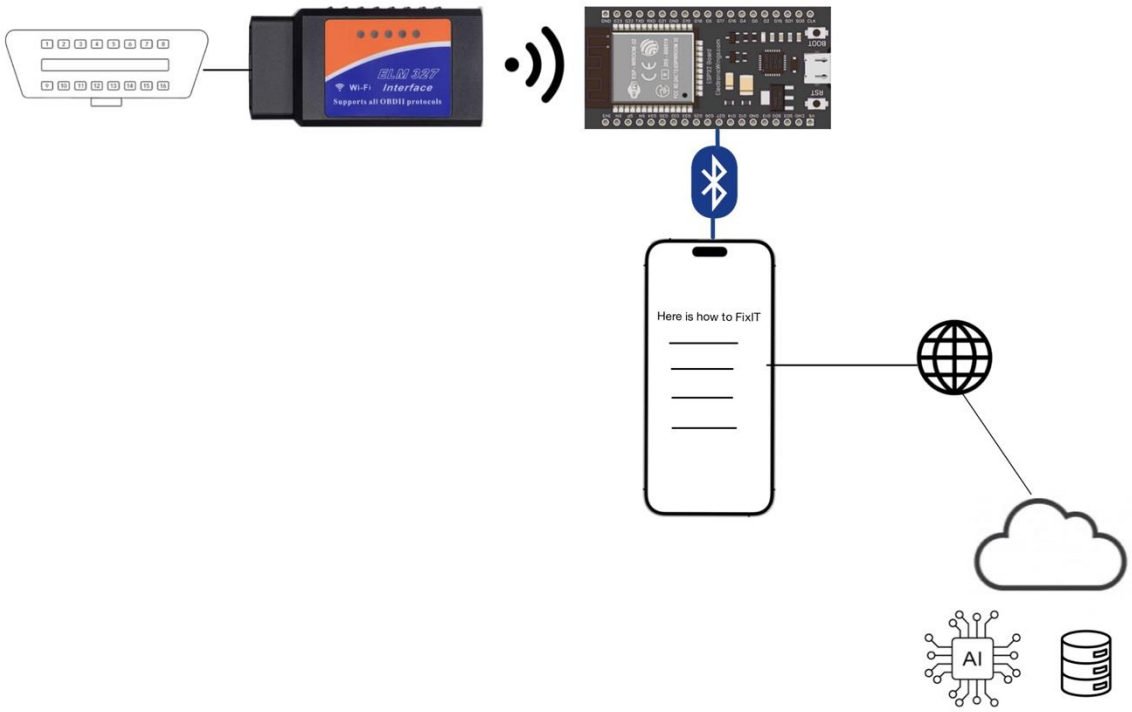
[3] dotintent, “GitHub - dotintent/react-native-ble-plx: React Native BLE library,” *GitHub*, Jul. 09, 2024. https://github.com/dotintent/react-native-ble-plx

[4] “OBD-II PIDs,” *Wikipedia*, May 19, 2021. https://en.wikipedia.org/wiki/OBD-II_PIDs

8.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc,. PCB testing issues etc., Software bugs etc.



9 Team

Complete each section as completely and concisely as possible. We strongly recommend using tables or bulleted lists when applicable.

9.1 Team Members

9.2 Required Skill Sets for Your Project

Protocol Implementation

- OBD-II communication protocols (ISO 15765-4, ISO 9141-2)
- CAN bus communication
- Error handling and verification procedures

Hardware Programming

- ESP32 configuration
- UART communication
- Memory management
- Protocol switching and timing

Testing and Verification

- Protocol verification (SOMM command)
- Fault simulation (SF command)
- Error detection and handling
- Device monitoring

Software Development

- React Native
- Bluetooth implementation
- Real-time data handling
- User interface design

Team Skill Coverage

Hardware Engineer

- Protocol implementation
- ESP32 programming
- Error handling
- Memory management

Software Engineer

- Mobile app development
- Bluetooth integration
- User interface design
- Data processing

Testing Engineer

- Protocol verification
- Fault simulation

- Performance monitoring
- Error detection

9.3 Skill Sets covered by the Team

Hardware Engineer Ben Muslic

- OBD-II protocol configuration (SP command)
- Protocol switching and timing
- Error handling (CAN ERROR, INVALID PARAM)
- Memory management

Software Engineer Jonathan Duron

- React Native mobile development
- Bluetooth communication
- User interface design
- Real-time data processing

QA Engineer Mohammed

- Protocol monitoring (SOMM command)
- Fault simulation (SF command)
- Error detection and logging
- Performance testing

Team Lead Will Griner

- ECU configuration (EA, EP commands)
- System architecture
- Protocol validation
- Documentation

9.4 Project Management Style Adopted by the team

Typically, Waterfall or Agile for project management.

9.5 Initial Project Management Roles

(Enumerate which team member plays what role)

9.6 Team Contract

Team Members:
1) _____Jonathan Duron_____ 2) _____

3) _____ 4) _____
5) _____ 6) _____
7) _____ 8) _____

Team Procedures

Day, time, and location (face-to-face or virtual) for regular team meetings:

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-
mail, phone, app, face-to-face):

3. Decision-making policy (e.g., consensus, majority vote):

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be
shared/archived):

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

3. Expected level of communication with other team members:

4. Expected level of commitment to team decisions and tasks:

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):

2. Strategies for supporting and guiding the work of all team members:

3. Strategies for recognizing the contributions of all team members:

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

2. Strategies for encouraging and support contributions and ideas from all team members:

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will
a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

2. Strategies for planning and assigning individual and team work:

3. Strategies for keeping on task:

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

2. What will your team do if the infractions continue?

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) _____ DATE
_____

2) _____ DATE _____

3) _____ DATE _____

4) _____ DATE _____

5) _____ DATE _____

6) _____ DATE _____

7) _____ DATE _____

8) _____ DATE _____