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

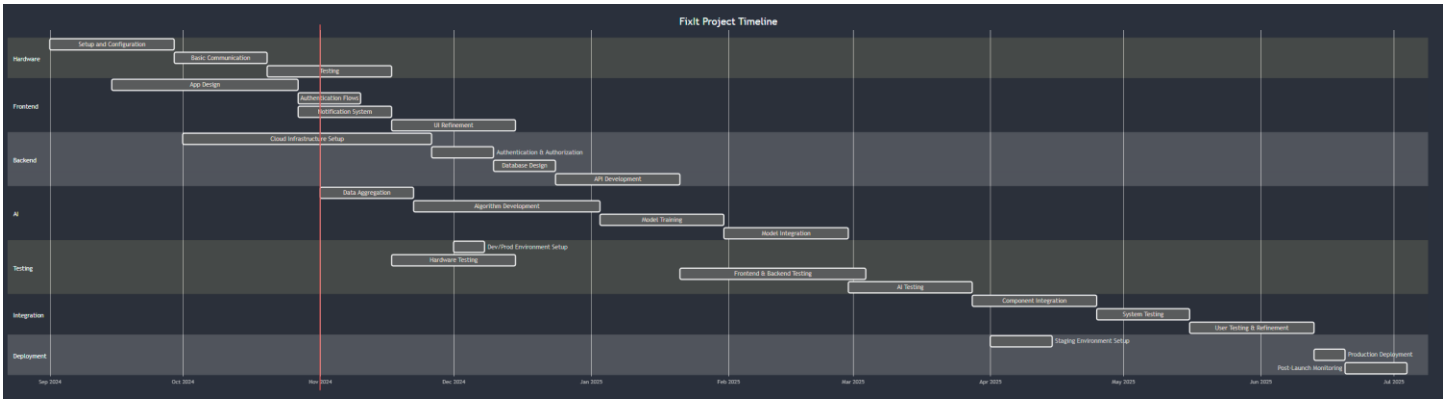
- Milestone: Implement basic AI-driven DTC interpretation

- 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



Our project timeline spans two semesters, with the goal of having a functional prototype by the end of the first semester. The timeline is divided into key tasks and subtasks:

1. **Hardware Setup and Configuration (Weeks 1-4)**
    - Configure OBD-II dongle and ESP32
    - Test initial data collection and transmission
2. **Basic Communication and Data Transmission (Weeks 3-6)**
    - Establish WiFi connection between ESP32 and phone hotspot
    - Set up data transfer from ESP32 to cloud
3. **Frontend App Design (Weeks 5-10)**
    - Design user interface for DTC code interpretation
    - Develop notification and alert systems
4. **Backend and Cloud Infrastructure (Weeks 5-12)**
    - Set up server infrastructure
    - Implement database and AI modules
5. **AI Integration (Weeks 8-14)**
    - Develop algorithms for DTC code interpretation
    - Integrate machine learning models for predictive maintenance
6. **Testing and Debugging (Weeks 10-15)**
    - Test hardware in vehicle with active check engine light
    - Conduct iterative debugging on frontend and backend
7. **Final Integration and Validation (Weeks 14-16)**
    - Verify all functionalities
    - Conduct user testing and refinement

3.5 Risks And Risk Management/Mitigation

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.

3.6 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 password, change password, etc.	Will	15
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

OBD-II Dongle and ESP32 Modules

- 2 OBD-II dongles for redundancy and parallel testing
- 3 ESP32 modules to allow for simultaneous development and testing
- Jumper wires, breadboards, and other necessary components for hardware setup

Vehicles with Active Check Engine Lights

- Access to at least 2 vehicles with active check engine lights for real-world testing
- Ideally, vehicles should have different makes, models, and years to test system versatility
- If consistent access to vehicles is not possible, consider purchasing a simulator

Cloud Service Subscription

- AWS, Google Cloud, or Microsoft Azure subscription for hosting and data processing
- Estimated cost: free-$200 per month, depending on usage and selected services
- Key services required:
    - Virtual machines for running web scrapers and AI models
    - Managed database for storing diagnostic data and user information
    - Object storage for storing raw data and backups
    - API gateway for handling requests between frontend and backend

Testing Environment and Simulated Data

- OBD-II simulator software for generating simulated vehicle data
- Diagnostic trouble code (DTC) datasets for training and testing AI models
- Test suite for automating system testing and ensuring reliability

API Calls to ChatGPT

- OpenAI API subscription for accessing ChatGPT language model
- Estimated cost: $0.002 per 1K tokens, which could amount to $50-$100 per month depending on usage
- Consider setting a monthly budget and implementing rate limiting to control costs

Development Tools and Licenses

- IDE licenses (e.g., JetBrains IDEs) for efficient development
- Git repository hosting (e.g., GitHub, GitLab) for version control and collaboration
- Project management tools (e.g., Jira, Trello) for task tracking and sprint planning

Miscellaneous Office Supplies

- Whiteboard and markers for brainstorming and system design
- Notebooks, pens, and other supplies for documentation and note-taking