Cairo University
Faculty of Computers and Artificial Intelligence
Department of Information Technology

Firmware Over-The-Air (FOTA)

Supervised by:
PhD. Haitham Safwat Hamza
Vice-Dean for Postgrad and Research - Cairo University

Prepared by:

1. Kareem Abd El-Moneam Fawzy Yassin 20200391
2. Mahmoud Alaa Eldeen Fathy Sadek 20200502
3. Mohammed Gabr Ahmed Khamis 20200765
4. Salma Sherif Ibrahim Moustafa 20200227
5. Ahmed Adel Emad Eldeen Farag 20200025
6. Zyad Mahmoud Abbady Amin 20200206

Graduation Project
Final Documentation
2023-2024

Table of Contents

Table of Figures

1. Introduction

1.1. Evolution from Mechanical Automobiles to Smart Cars

The history of automobiles is a fascinating journey that spans more than a century, marked by groundbreaking innovations and transformative advancements. From the early days of mechanical ingenuity to the present era of smart, connected vehicles, the automotive industry has undergone a remarkable evolution.

1.1.1. The Mechanical Marvels

The journey begins in the late 19th century when automotive pioneers such as Karl Benz and Henry Ford laid the foundation for the mechanical marvels, we now take for granted. The advent of the internal combustion engine and mass production techniques ushered in an era of mobility, transforming the world's transportation landscape.

For much of the 20th century, automotive innovation focused on refining mechanical components. Engineers constantly sought ways to enhance engine efficiency, optimize suspension systems, and improve safety features. This mechanical evolution culminated in sleek and powerful vehicles that became synonymous with freedom and progress.

1.1.2. The Rise of Electronics

As the automotive industry entered the late 20th century, electronic components began to play a pivotal role. The introduction of Electronic Control Units (ECUs) marked a shift towards incorporating digital technologies for better control over various aspects of vehicle performance.

The integration of electronics extended beyond engine control to include features like anti-lock braking systems (ABS), airbags, and electronic stability control. These electronic enhancements significantly improved safety, efficiency, and overall driving experience.

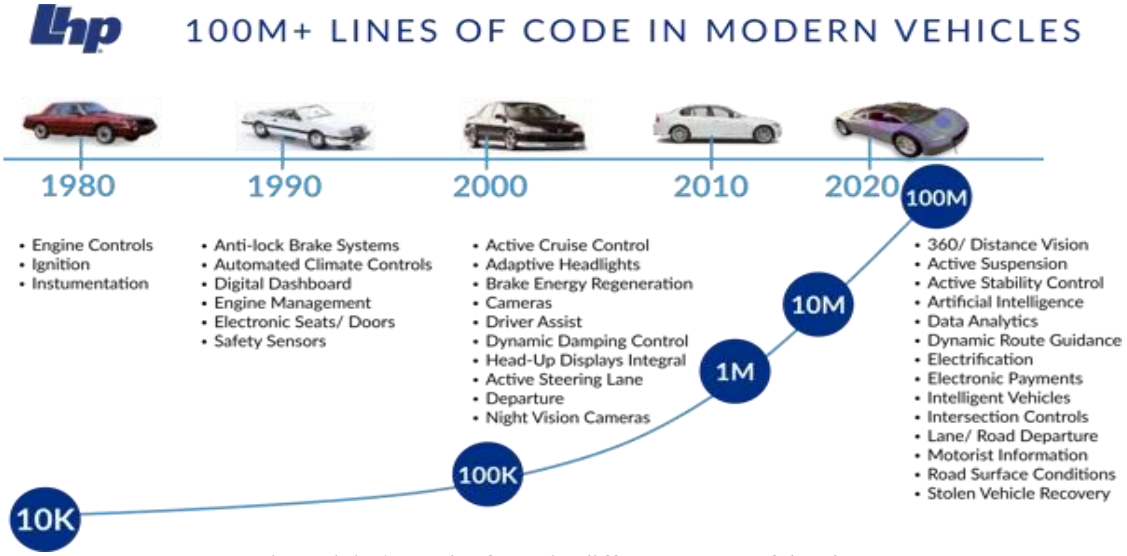


Figure1.1. A graph of cars in different stages of development

1.1.3. The Rise of Embedded Systems

In recent decades, the automotive industry has witnessed a paradigm shift towards smart vehicles equipped with advanced embedded systems and software. This transition from purely mechanical to software-driven functionality has redefined what it means to drive and own a car.

1.2. Problem Statement: The Challenge of Software Evolution

The evolution of automobiles from mechanical marvels to smart, software-driven cars has brought about unprecedented advancements, revolutionizing the way we perceive and interact with vehicles. While this transition has introduced numerous benefits in terms of safety, efficiency, and functionality, it has also given rise to a significant challenge: the seamless and secure evolution of software within these sophisticated automotive systems.

1.2.1. Complexity of Embedded Systems

With the rise of embedded systems and Electronic Control Units (ECUs), vehicles have transformed into complex interconnected systems. The challenge lies in managing and updating the intricate web of software that controls everything from engine performance to advanced driver assistance systems (ADAS) and connectivity features.

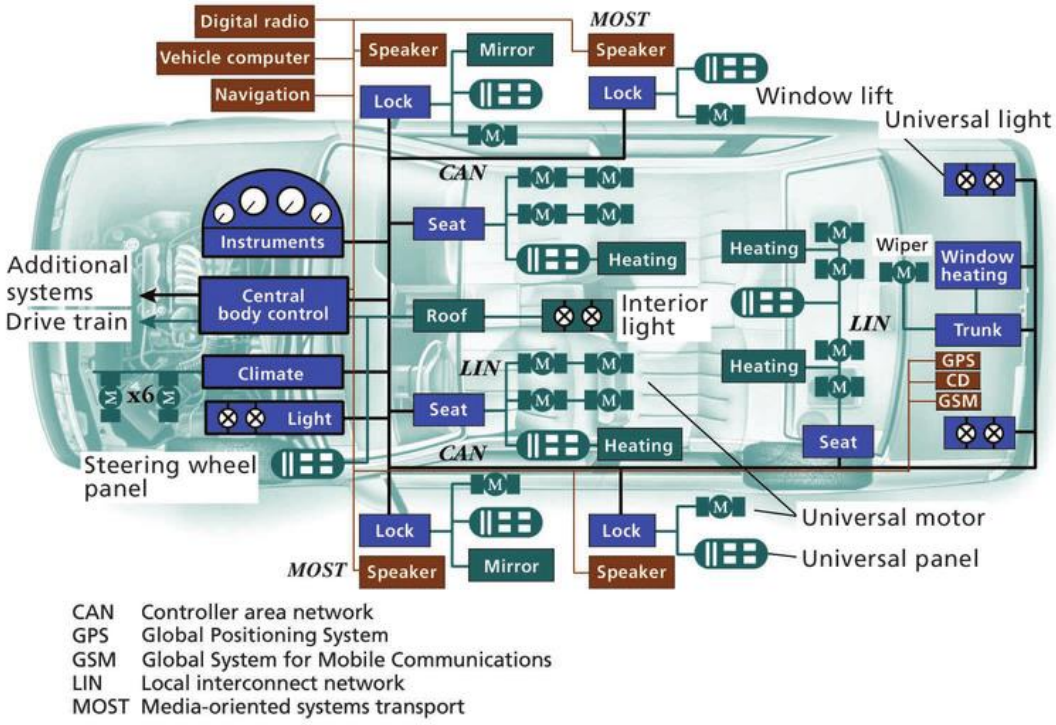


Figure1.2. A diagram of an electronic vehicle system architecture

1.2.2. Over-the-Air Software Updates

The advent of connected and autonomous vehicles has necessitated advancements in automotive technologies, particularly in the fields of software updates and cybersecurity. Firmware Over-The-Air (FOTA) systems have become a critical component for modern vehicles, allowing manufacturers to update the vehicle's software remotely. Concurrently, automotive cybersecurity has become a paramount concern, given the increasing vulnerabilities that come with connectivity. This section examines the intersection of FOTA systems and automotive cybersecurity, highlighting key developments, challenges, and future directions.

1.2.2.1. Need for Real-time Adaptability

Smart cars now rely on over-the-air (OTA) software updates to introduce enhancements, bug fixes, and even entirely new functionalities. The need for real-time adaptability has become paramount, requiring a robust infrastructure to facilitate secure and efficient transmission of software updates.

1.2.2.2. Security Concerns

As vehicles become more connected, the vulnerability to cyber threats increases. Ensuring the security of over-the-air updates is a critical concern, requiring sophisticated encryption methods, secure authentication, and continuous monitoring to prevent unauthorized access and tampering.

1.2.2.3. Compatibility Challenges

Smart cars often consist of a multitude of ECUs and diverse software modules. Ensuring compatibility across various hardware configurations and software versions poses a formidable challenge. Failing to address this can lead to issues such as system instability, malfunctions, and compromised safety features.

1.3. User Experience and Trust

1.3.1. User Perception and Acceptance

Introducing regular software updates requires not only technical finesse but also effective communication with users. Building trust in the reliability and safety of these updates is crucial for user acceptance and satisfaction.

1.3.2. Downtime and Reliability

Balancing the need for updates with minimizing downtime and ensuring the reliability of the update process is a delicate task. Users expect seamless integration of new features without compromising the functionality or safety of their vehicles.

2. Literature Review

2.1. FOTA Systems in the Automotive Industry

Definition and Functionality FOTA refers to the technology that allows manufacturers to wirelessly update the firmware of electronic control units (ECUs) in vehicles. This capability enables the delivery of new features, bug fixes, and performance improvements without requiring physical access to the vehicle.

Benefits of FOTA

- **Convenience and Efficiency:** FOTA eliminates the need for vehicle recalls for software updates, saving time and resources for both manufacturers and customers.
- **Enhanced Performance and Features:** Continuous updates can enhance vehicle performance, introduce new features, and ensure compliance with regulatory standards.
- **Cost Savings:** Reduces the logistical costs associated with traditional update methods and minimizes downtime for vehicles.

Implementation Challenges

- **Complexity of Vehicle Systems:** Modern vehicles have multiple ECUs with interdependent software, making the update process complex.
- **Compatibility Issues:** Ensuring compatibility of new firmware with existing hardware and software configurations is crucial.
- **Testing and Validation:** Thorough testing is required to ensure that updates do not introduce new issues or vulnerabilities.

2.2. Automotive Cybersecurity

Importance of Cybersecurity With increased connectivity, vehicles are more vulnerable to cyberattacks. Ensuring the security of automotive systems is critical to protect against unauthorized access, data breaches, and potential physical harm.

Common Threats and Vulnerabilities

- **Remote Exploits:** Attackers can exploit vulnerabilities in wireless communication protocols.
- **Malware:** Malicious software can be introduced through various entry points, including FOTA updates.
- **Data Theft and Privacy Concerns:** Unauthorized access to sensitive data can lead to privacy violations and financial losses.

Cybersecurity Measures

- **Encryption:** Secure communication channels using robust encryption methods.

- **Authentication:** Strong authentication mechanisms to ensure that only authorized updates are applied.
- **Intrusion Detection Systems (IDS):** Monitoring systems to detect and respond to suspicious activities.

2.3. Intersection of FOTA and Automotive Cybersecurity

Securing FOTA Updates

- **Cryptographic Security:** Ensuring that firmware updates are signed and encrypted to prevent tampering and unauthorized access.
- **Secure Boot:** Ensuring that the vehicle's ECUs only run authenticated and verified firmware.
- **Over-The-Air Protocols:** Developing secure OTA protocols to protect the integrity and authenticity of updates.

Challenges in Securing FOTA Systems

- **Real-time Verification:** Ensuring that updates can be verified and applied in real-time without compromising vehicle safety.
- **Resource Constraints:** ECUs have limited computational resources, making it challenging to implement complex security measures.
- **Update Rollback:** Ensuring that vehicles can safely revert to previous firmware versions if an update fails or introduces issues.

Case Studies and Examples

- **Tesla:** Tesla's implementation of FOTA has been widely recognized for its effectiveness in delivering new features and improving vehicle performance. However, it also highlights the importance of robust cybersecurity measures to prevent unauthorized access.
- **Jeep Cherokee Hack:** The infamous Jeep Cherokee hack demonstrated vulnerabilities in connected vehicles and underscored the need for secure FOTA implementations.

3. Discussion

Enhanced Security Frameworks

Developing comprehensive security frameworks that integrate FOTA systems with advanced cybersecurity measures is essential. This includes adopting multi-layered security approaches and continuous monitoring.

Collaboration and Standardization

Industry-wide collaboration and the development of standardized protocols can help in addressing the challenges associated with FOTA and cybersecurity. Organizations such as the Auto-ISAC (Automotive Information Sharing and Analysis Center) play a crucial role in this regard.

Advanced Technologies

Emerging technologies such as blockchain can provide additional security layers for FOTA updates, ensuring transparency and immutability. Artificial intelligence and machine learning can also be leveraged to enhance intrusion detection and response systems.

4. Conclusion

The integration of FOTA systems with robust automotive cybersecurity measures is critical for the safe and efficient operation of modern vehicles. While significant progress has been made, ongoing research and development are necessary to address the evolving threats and challenges. Ensuring the security of FOTA updates not only protects against cyber threats but also enhances the overall reliability and performance of automotive systems.

5. STM32F401RCT6: Two-View Electronic Control Unit

5.1. Microcontroller Overview

The **STM32F401RCT6** is utilized as a Two-View Electronic Control Unit (ECU), functioning both as the Main ECU and the Application ECU. This dual role allows it to serve as the central processing unit, managing the overall system and controlling various peripherals and applications.

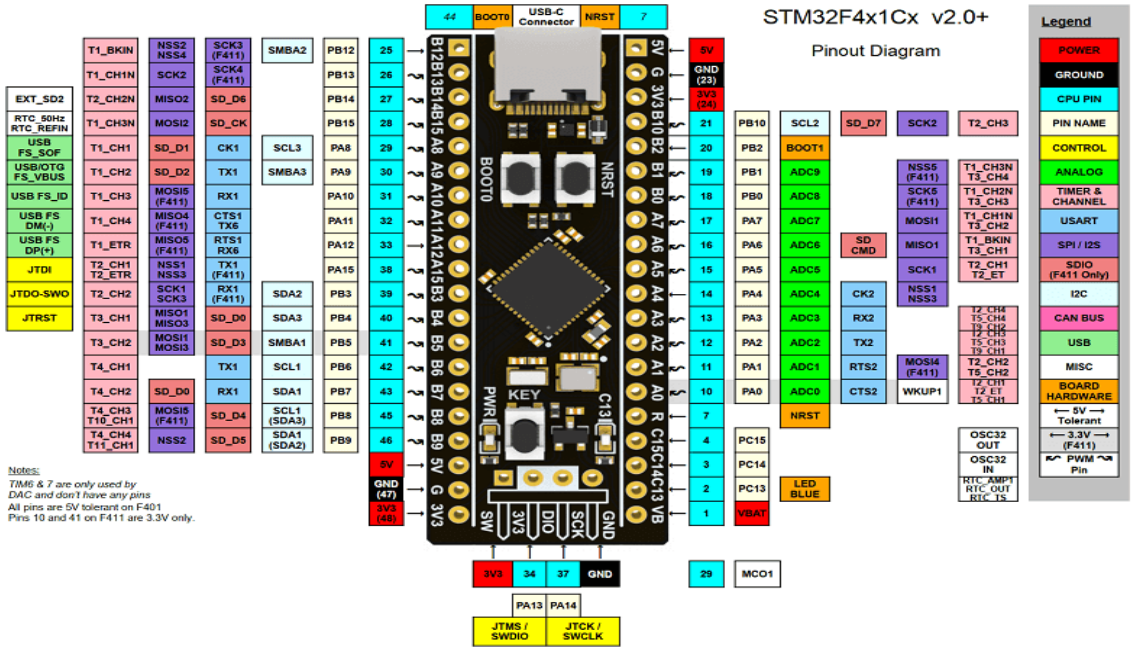


Figure5.1. STM32F401RCT6 Pinout Diagram

The STM32F401RCT6, commonly referred to as the "Black Pill," is a microcontroller from the STM32F4 family developed by STMicroelectronics. This microcontroller is based on the ARM Cortex-M4 core, which is known for its high performance and efficiency, making it suitable for a wide range of applications, including embedded systems, consumer electronics, and automotive control units.

Key Features

1. **Core and Performance**:
    - **ARM Cortex-M4 CPU**: The STM32F401RCT6 features a 32-bit ARM Cortex-M4 CPU with a Floating-Point Unit (FPU), operating at a maximum frequency of 84 MHz. This provides a good balance between performance and power consumption.
    - **DSP Instructions**: The core includes Digital Signal Processing (DSP) instructions, enhancing its capabilities for signal processing tasks.
2. **Memory**:
    - **Flash Memory**: It includes 256 KB of Flash memory, which is used for storing the firmware and applications.
    - **SRAM**: The microcontroller is equipped with 64 KB of SRAM, providing ample space for variable storage and program execution.
3. **Peripherals and Interfaces**:
    - **GPIO**: The STM32F401RCT6 offers up to 50 General Purpose Input/Output (GPIO) pins, which are highly configurable and can be used for various digital and analog functions.
    - **Communication Interfaces**: It supports multiple communication protocols including:
        - **I2C**: Two I2C interfaces for serial communication.
        - **SPI**: Three SPI interfaces for high-speed serial communication.
        - **USART/UART**: Four USART/UART interfaces for asynchronous serial communication.
        - **USB**: A full-speed USB 2.0 interface for connectivity.
    - **Timers**: It includes multiple timers for managing tasks such as event counting, waveform generation, and timing control.
    - **ADC**: A 12-bit Analog-to-Digital Converter (ADC) for analog signal conversion.
4. **Power Management**:
    - The STM32F401RCT6 operates at a voltage range of 1.8V to 3.6V, with various power-saving modes to optimize energy consumption.
5. **Development and Debugging**:
    - **Development Tools**: It is supported by a comprehensive suite of development tools including STM32CubeMX for configuration, and various Integrated Development Environments (IDEs) like Keil, IAR, and STM32CubeIDE.
    - **Debugging Support**: It features debugging capabilities with JTAG and Serial Wire Debug (SWD) interfaces, facilitating effective debugging and firmware development.

Applications

The STM32F401RCT6 is versatile and widely used in numerous applications due to its robust feature set:

- **Embedded Systems**: Ideal for small to medium-sized embedded systems requiring efficient processing.
- **Consumer Electronics**: Used in devices like smart home appliances, wearables, and personal gadgets.
- **Automotive Systems**: Employed in automotive control systems, including ECUs, due to its reliability and performance.
- **Industrial Control**: Utilized in industrial automation and control systems for its robustness and extensive interfacing capabilities.

5.2. Linker Script File Configuration

The linker script file is essential in configuring the STM32F401RCT6 (Black Pill) to operate as a dual-function Electronic Control Unit (ECU). This script is responsible for ensuring the microcontroller's correct behavior in its two distinct roles:

1. **Main ECU**: During the flashing of code and the OTA updates via the bootloader, the linker script ensures the microcontroller functions as the Main ECU. It handles the initial loading process, allocates memory appropriately, and establishes the environment necessary for the bootloader's operation.
2. **Application ECU**: When integrating peripherals with the hardware components to manage the application (***i.e. Obstacles Avoidance Vehicle***), the linker script configures the microcontroller to act as the Application ECU. This involves mapping peripheral interfaces, configuring interrupt vectors, and defining memory regions to facilitate the application's smooth operation.

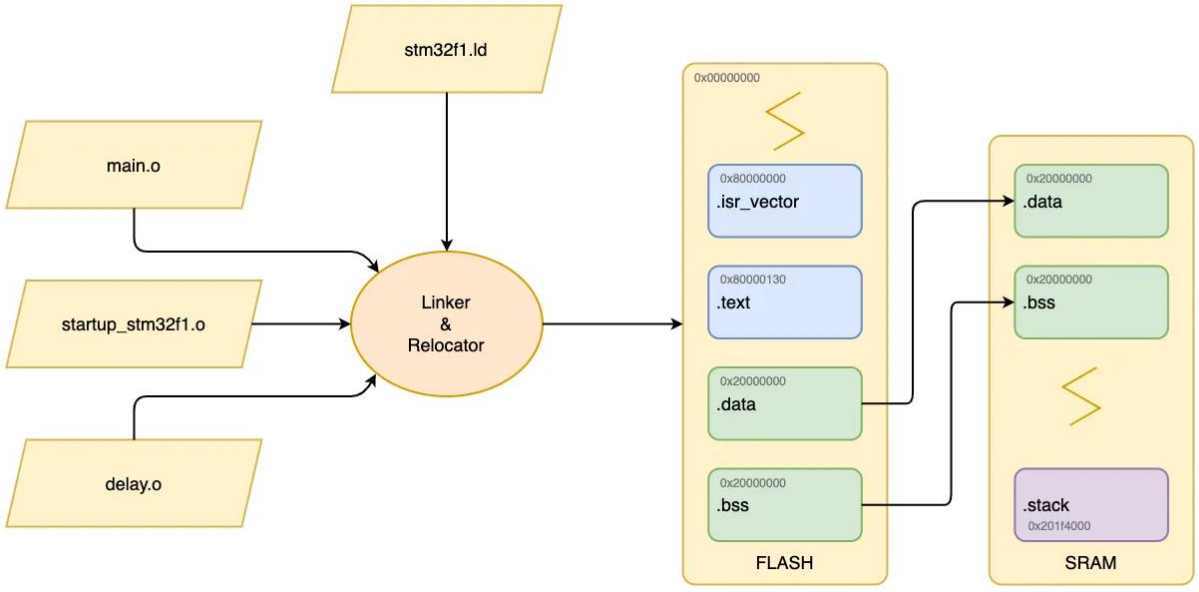


Figure 5.2. Linking & Relocation flow

By defining memory regions, entry points, and peripheral configurations, the linker script ensures that the STM32F401RCT6 can seamlessly transition between its roles as Main ECU and Application ECU. This dual functionality supports both the bootloading process and the operational requirements of the obstacle avoidance vehicle, thereby enabling efficient and effective system management.

Memory Architecture

Our Memory is split into four partitions:

- **Bootloader:** It is a small application that is used to flash application in flash memory.
- **Application 1:** It is the version one of application, these space from the flash, the bootloader loads the application hex file in it.
- **Application 2:** It is the second version of the application 1, it run if the data corruption when the bootloader flash update or another file in the space of applicatin1.
- **Request flag:** It is the indication that indicates there is an update or no update.

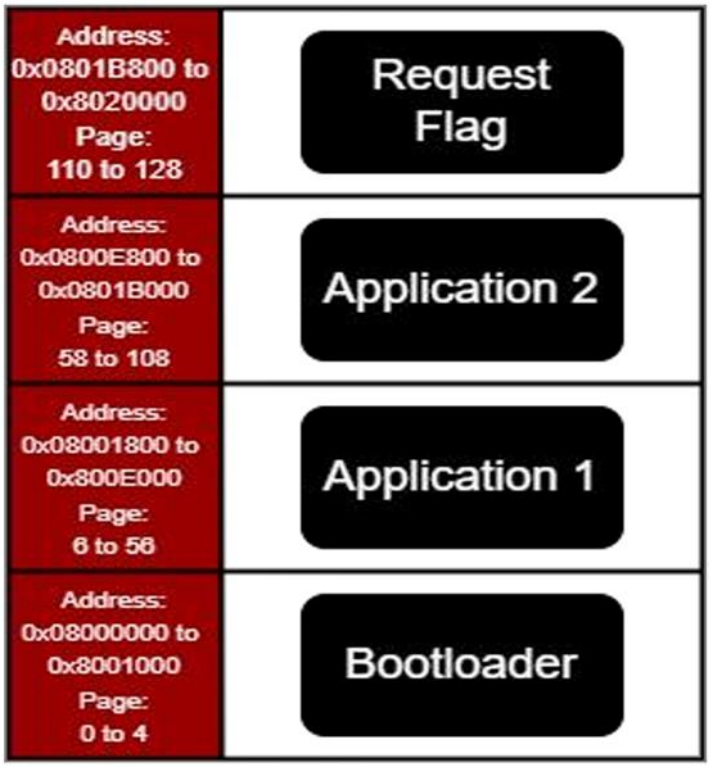


Figure5.3. Memory Architecture Partitions

5.3. Pinout Configuration

This section details the pinout configuration for the STM32F401RCT6 (Black Pill) microcontroller used in the project, specifying the assignment of various peripherals and their corresponding ports and pins.

Port	Pin	Peripheral	Pin Function
Port A	2	NodeMCU ESP32	TX2 (UART2)
Port A	3	NodeMCU ESP32	RX2 (UART2)
Port A	6	PWM Source	TIM3_CH1
Port A	10	Buzzer	Control
Port A	11	Bluetooth HC-05	TX6 (UART6)
Port A	12	Bluetooth HC-05	RX6 (UART6)
Port B	6	Ultrasonic Sensor	Echo (TIM4_CH1)
Port B	7	Ultrasonic Sensor	Trigger
Port B	12	DC Motors 1,2	Control
Port B	13	DC Motors 1,2	Control
Port B	14	DC Motors 3,4	Control
Port B	15	DC Motors 3,4	Control

Abbreviations

- UART2: Universal Asynchronous Receiver/Transmitter 2
- UART6: Universal Asynchronous Receiver/Transmitter 6
- TIM3_CH1: Timer 3 Channel 1
- TIM4_CH1: Timer 4 Channel 1
- E: Enable Pin
- State: Status Pin
- TRIG: Trigger Pin
- ECHO: Echo Pin
- Control: Control Pins

Table 2.1. Pinout Configuration

The above configuration provides a comprehensive overview of the pin assignments for the STM32F401RCT6 (Black Pill) microcontroller. Each peripheral is connected to specific ports and pins, ensuring proper functionality and integration within the system. This detailed mapping is crucial for the development and troubleshooting of the application, particularly in controlling various components such as the Bluetooth module, NodeMCU, DC motors, ultrasonic sensor, PWM, and buzzer.

6. System Architecture

6.1. Layered Architecture

The system architecture of the project is structured around a layered framework, emphasizing modularity and cohesive integration of components. This architectural design encompasses two distinct perspectives of the ECU: the Main ECU and the Application ECU.

6.1.1. First View: Main Electronic Control Unit (ECU)

Using STM32F401RCT6 (Black Pill) as The Main ECU serving as the central processing unit responsible for managing the overall system and controlling various peripherals. Under the Main ECU, the Microcontroller Abstraction Layer (MCAL) provides an abstraction of low-level hardware details.

6.1.1.1. Microcontroller Abstraction Layer (MCAL)

The MCAL within the Main ECU is enhanced with a bootloader, which is a specialized program responsible for managing the firmware update process. The bootloader is designed to handle and flash the updates to the target ECUs. It provides a secure mechanism for receiving, verifying, and installing new firmware versions on the system. The bootloader is crucial for maintaining system integrity and ensuring a successful update process.

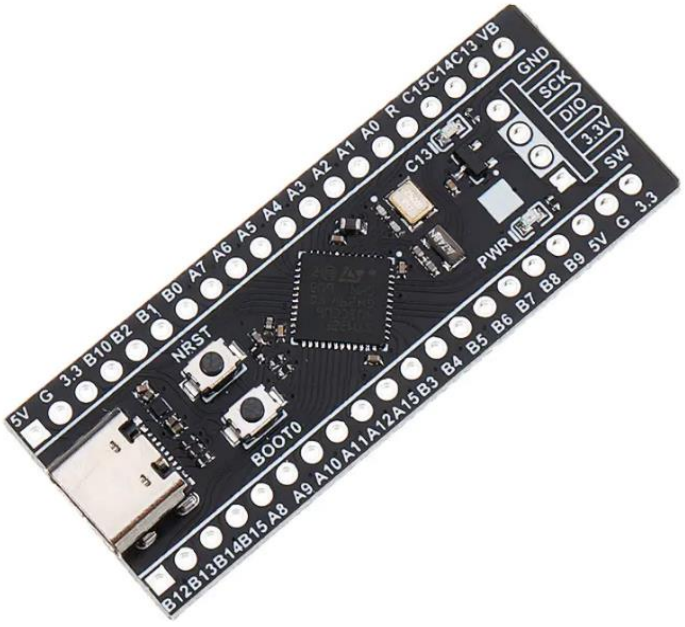


Figure 6.1. STM32F401RCT6 (Black Pill) development board

6.1.1.2. Hardware Abstraction Layer (HAL)

1. **Bootloader**
   A bootloader is a small program or piece of code that resides in the non-volatile memory of a microcontroller. Its primary function is to initialize the system and load the main application program (firmware) into the system's RAM or flash memory for execution.

2. **Gateway**
   Using **NodeMCU ESP32** WiFi development kit as a gateway between the server and the Main ECU offers a robust solution for continuously receiving updates, services, and notifications in a connected vehicle system.
   The ESP32 is a highly versatile and widely used WiFi-enabled microcontroller chip. It is renowned for its low cost, compact size, and powerful capabilities, making it a popular choice for a variety of IoT (Internet of Things) and embedded projects.

Why NodeMCU ESP32?

The NodeMCU ESP32 serves as a crucial intermediary between the server and the Main ECU of the vehicle, benefits from its WiFi capabilities to establish connectivity with the internet and facilitate communication with the server.

Figure 6.2. NodeMCU ESP32 development board

This connection enables real-time reception of updates, services, and notifications from the server, ensuring continuous communication between the vehicle system and external services. When updates or other services are published by the OEM, the server transmits them to the NodeMCU, which then processes the received data and forwards relevant information to the Main ECU using UART communication. The NodeMCU easily communicates with the Main ECU, aligning with the vehicle's electronics architecture.

Upon receipt of updates, the Main ECU executes the necessary bootloader to install and apply them on, ensuring that the vehicle's software remains up-to-date and functional. Additionally, the NodeMCU serves as a relay for notifications and alerts from the server to the vehicle's dashboard or user interface, providing drivers with crucial information about events or changes in the system's status.

This integration of the NodeMCU ESP32 as a gateway enhances the vehicle system's connectivity, enabling efficient data exchange, update delivery, and communication between the Main ECU and external services.

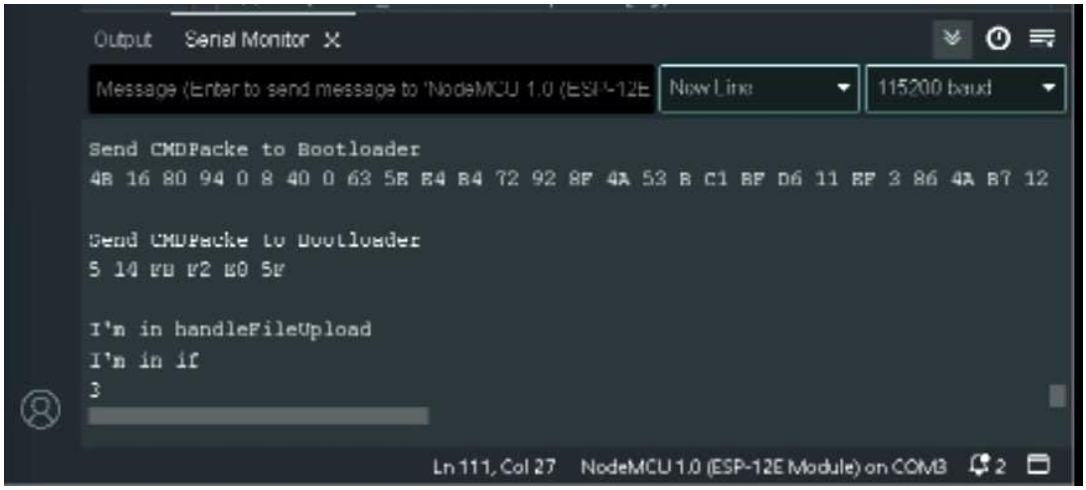


Figure 6.3. Gateway Test

6.1.2. Second View: Application Electronic Control Unit (ECU)

The Application ECU is structured into three key components: the Microcontroller Abstraction Layer (MCAL), the Hardware Abstraction Layer (HAL), and the Application Layer. This perspective focuses on executing the application logic and interfacing with higher-level functionalities within the system architecture.

6.1.2.1. Microcontroller Abstraction Layer (MCAL)

The MCAL within the Application ECU focuses on abstracting low-level details specific to the STM32F401RCT6 microcontroller. It plays a pivotal role in facilitating seamless communication with hardware peripherals and incorporates various drivers for essential functionalities such as:

- **NVIC (Nested Vectored Interrupt Controller)**
  NVIC drivers offer both Low-Level (LL) and Hardware Abstraction Layer (HAL) options. NVIC LL drivers provide direct access to NVIC registers, while HAL NVIC drivers offer a higher-level abstraction for simplified interrupt handling.

Why NVIC?
Using NVIC for handling interrupts is an important step for working with updates and for “snoozing updates” feature, as it prioritizes interrupts, allowing critical tasks related to the firmware update to take precedence.

- **SysTick Timer**
  The SysTick Timer driver is primarily considered a Low-Level (LL) driver. From initiating communication with the server to managing the intervals between critical tasks, one of its crucial roles lies in managing timeouts during updates. It enables the timely triggering of actions or error-handling mechanisms when necessary, ensuring that the update process stays on course.

Why SysTick?
Using SysTick timer ensures a seamless flow of operations in managing timeouts during updates and to create a timer used by the system to work with multiple functions.

- **DIO (Digital Input/Output - GPIO)**
  DIO drivers encompass both LL and HAL options. LL DIO drivers provide direct access to GPIO registers, allowing detailed configuration. HAL DIO drivers offer a higher-level abstraction for simplified GPIO configuration and usage.

Why DIO?
Using DIO handles input/output operations to identify each signal as 0 or 1 in reading inputs and to write 0 or 1 signals as an output in its directory in the memory.

- **RCC (Reset and Clock Control)**
  The RCC driver includes both LL and HAL options. LL RCC drivers provide direct access to RCC registers, allowing fine-grained control over reset and clock configurations.
  HAL RCC drivers offer a more abstract interface for simplified RCC configurations.

Why RCC?
RCC allows control over reset and clock configurations to have a synchronous system and to ensure all processes execute based on a common clock signal and that different components will work in harmony and will be aligned in time.

- **DMA (Direct Memory Access)**
  Providing both LL and HAL options. LL DMA drivers offer direct register-level access, while HAL DMA drivers offer a more abstract and user-friendly interface

for configuring and using DMA.It facilitates efficient data transfers between peripherals and memory without involving the CPU.
It configures and manages the data transfers, specifying source and destination addresses, transfer size, and other parameters.

Why DMA?
DMA ensures not involving the CPU at each step in the memory handling operations, offers time to the Main ECU to perform critical tasks and to work with the processed data.

- **UART (Universal Asynchronous Receiver Transmitter)**
  UART also includes both LL and HAL options. LL UART driver provides direct access to UART registers, allowing detailed configuration.
  HAL UART driver offers a higher-level abstraction for simplified UART configuration and usage.

  It facilitates serial communication between microcontrollers and external devices and supports asynchronous communication modes, enabling the transmission and reception of data. We will talk more about the UART, its frame, and how it works under the ***UART Serial Communication Protocol section at page 55.***

6.1.2.2. Hardware Abstraction Layer (HAL)

1. DC Motors

- **Function:** Primarily used to drive the vehicle with a four-wheel drive system, enabling dedicated movements and avoiding obstacles.
- **Power Requirements:** Typically operate at 3V to 12V, depending on the motor specifications. Ensure the motors receive adequate voltage and current for optimal performance.
- **Connection:**

  - **Power Supply:** Connect the positive and negative terminals of each motor to the corresponding outputs of the H-Bridge motor driver.
  - **Control:**
    - The direction and speed of the motors are controlled via the H-Bridge motor driver.
    - Use the microcontroller to send signals to the H-Bridge to manage motor operations.

  - **Operation:**

    - The motors receive commands from the microcontroller through the H-Bridge, which dictates their speed and direction.

- By varying the PWM signals, you can control the speed of each motor.
- Direction control is achieved by setting the H-Bridge input pins to the appropriate high/low values.

Figure 6.4. DC Motor

2. H-Bridge L298N Motor Drivers

- **Function:** The H-Bridge L298N motor drivers are responsible for controlling the speed and direction of the DC motors using Pulse Width Modulation (PWM).
- **Power Requirements:**
  - **Motor Voltage:** Typically operates at 5V to 35V, depending on the motor specifications.
  - **Logic Voltage:** Requires 5V for the control logic.

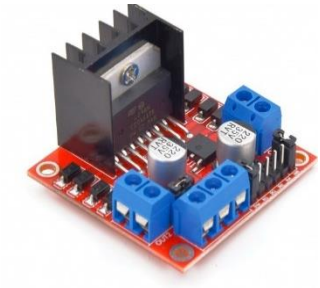


Figure 6.5. Dual H-Bridge L298N Motor Drivers

- **Connection:**
  - **Power Supply:** Connect the motor power supply (e.g., 11.1V from the battery pack) to the Vcc input of the H-Bridge.
  - **Motor Connections:** Connecting the DC motors to the output terminals of the H-Bridge.
  - **Logic Power:** Provides 5V to the logic power input from a voltage regulator.

- **Control Pins:**
    - Connect the IN1, IN2, IN3, and IN4 pins to GPIO pins on the microcontroller to control motor direction.
    - Connect the ENA and ENB pins to PWM-capable GPIO pins on the microcontroller to control motor speed.
- **Ground:** All grounds (motor power supply, logic power supply, and microcontroller) are connected to a common ground.
- **Operation:**

By adjusting the PWM signal sent to the ENA and ENB pins, the speed of the motors can be controlled. The direction of the motors is controlled by setting the IN1, IN2, IN3, and IN4 pins high or low.

3. Ultrasonic Sensor

- **Function:** The ultrasonic sensor is strategically placed to effectively detect obstacles, optimizing the performance of the obstacle detection system. Its wide coverage allows it to accurately measure distances, enhancing the vehicle's ability to navigate safely.

- **Power Requirements:** Typically operates at 5V.

Figure 6.6. Ultrasonic Sensor

- **Connection:**

    - **Power Supply:** Connect the Vcc pin to a 5V regulator output.
    - **Ground:** Connect the GND pin to the common ground.
    - **Control Pins:**
        - Connect the Trigger (TRIG) pin to a GPIO pin on the microcontroller.
        - Connect the Echo (ECHO) pin to another GPIO pin on the microcontroller.

    - **Operation:**
        - The microcontroller sends a pulse to the TRIG pin to initiate a distance measurement.
        - The sensor emits an ultrasonic pulse and waits for it to bounce back from an obstacle.

- The ECHO pin outputs a signal whose duration corresponds to the time taken for the pulse to return, which the microcontroller uses to calculate the distance.

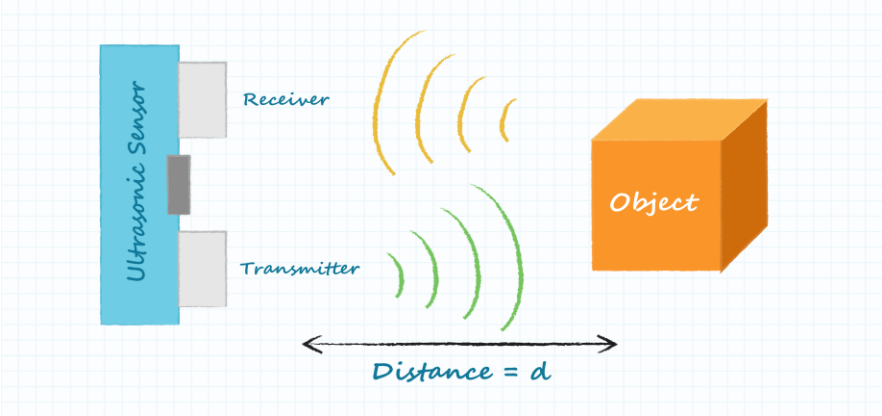


Figure 6.7. Ultrasonic Sensor Operation

- **Algorithm Simplification:**
  - Using a single ultrasonic sensor simplifies the scanning algorithm, making the system more efficient and reliable.
  - The straightforward setup reduces complexity and enhances the overall efficiency of the obstacle detection system.

- **Performance Optimization:**
  - The sensor's wide coverage ensures that obstacles are detected accurately and promptly.
  - This optimization helps the vehicle navigate safely by providing precise distance measurements, allowing for timely and appropriate responses to detected obstacles.

4. Buzzer

- **Function**: Acts as an auditory alert when the ultrasonic sensor detects an obstacle in its path.

Figure 6.8. Buzzer

- **Power Requirements:** Typically operates at 3.3V to 5V, depending on the model.
- **Ground:** Connect the negative terminal to the common ground.
- **Operation:** When the ultrasonic sensor detects an obstacle within a certain distance, the microcontroller will trigger the buzzer. The microcontroller can send a high signal to the GPIO pin connected to the buzzer, activating it and producing a sound.

5. HC-05 Bluetooth Module

The HC-05 is a widely used Bluetooth module that facilitates wireless communication between electronic devices. It's commonly utilized in hobbyist projects, robotics, IoT (Internet of Things) devices, and various other applications.

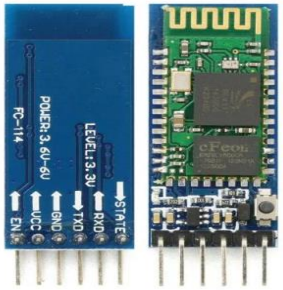


Figure 6.9. HC-05 Bluetooth Module

1- Why HC-05 Bluetooth Module?

The HC-05 Bluetooth Module was chosen to facilitate communication between the mobile application and the Main ECU. It provides an efficient means for the user interface and the system to communicate seamlessly. The module enables multiple microcontrollers to connect with each other effectively, thanks to its robust Bluetooth capabilities. Additionally, the HC-05 includes UART functionality, allowing for the transmission of simple serial commands, further enhancing its versatility and ease of use in our system.

2- How it works?

- **Bluetooth Communication**: The HC-05 module utilizes Bluetooth technology to establish a wireless communication link between two devices. It operates in the 2.4GHz ISM (Industrial, Scientific, and Medical) frequency band.

- **Serial Communication Interface**: One of the notable features of the HC-05 module is its UART (Universal Asynchronous Receiver-Transmitter) serial interface. This allows microcontrollers or other devices with UART capabilities to communicate with the module using simple serial commands. It typically operates at a baud rate of 9600 bps by default, but this can be configured as needed.

- **Modes of Operation**: The HC-05 module can operate in two main modes: AT command mode and communication mode.

    1. **AT Command Mode**: In this mode, the module accepts AT commands sent via the serial interface. These commands allow users to configure various parameters such as the device name, Bluetooth pairing settings, baud rate, and operating mode.

    2. **Communication Mode**: Once configured, the module can switch to communication mode, where it establishes a Bluetooth connection with another device. In this mode, data can be transmitted wirelessly between the HC-05 module and the paired device.

- **Master/Slave Configuration**: The HC-05 module can be configured to operate as either a master or a slave device in a Bluetooth network. As a master, it can initiate connections with other Bluetooth devices. As a slave, it can only accept connections initiated by other master devices.
- **Pairing and Security**: The HC-05 module supports various Bluetooth pairing modes, including pairing with a fixed PIN code or in a secure mode that requires authentication. This helps ensure secure communication between paired devices.
- **Power Supply**: The HC-05 module typically operates at 3.3V, although some versions may support a wider voltage range. It requires a stable power supply to function properly.

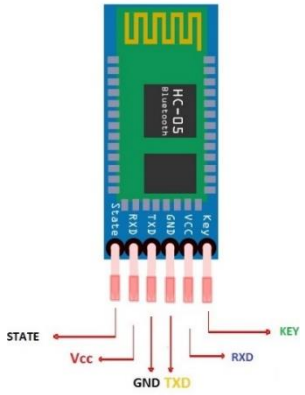


Figure 6.10. HC-05 Bluetooth Module pinout diagram

6.1.2.3. Application Layer

- **Obstacles Avoidance Vehicle: Mini-ADAS System**

The Application Layer of the obstacle avoidance vehicle is responsible for managing the car's behavior, focusing on obstacle detection and avoidance. It interfaces with DC motors, and ultrasonic sensors to navigate the car safely and a buzzer to alert obstacle detection. This layer has a modular architecture, separating obstacle detection, navigation logic, and motor control for better maintainability and flexibility.

It plays a crucial role in orchestrating the interaction between hardware components and implementing intelligent algorithms for safe navigation. By integrating DC motors, ultrasonic sensor, and buzzer it ensures effective obstacle detection and avoidance, allowing the car to navigate autonomously.
To ensure safety, reliability, and ease of maintenance, we will ensure that these functions operate flawlessly using our well-organized and regularly updated Firmware Over-The-Air (FOTA) system.

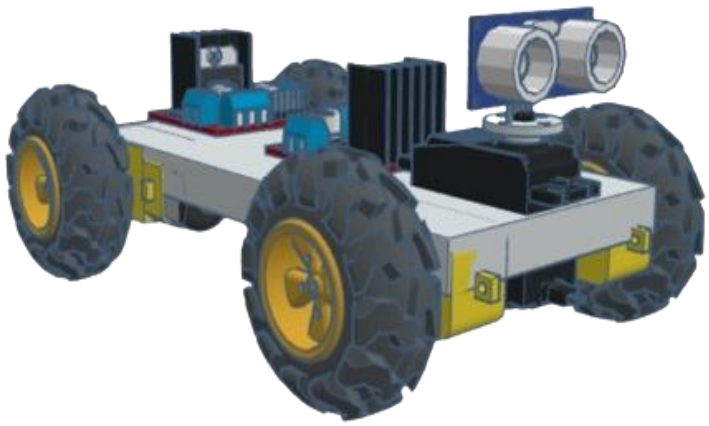


Figure 6.11. Obstacles Avoidance Vehicle 3D Model

- **Application Functionality**

The Ultrasonic sensor scans for obstacles ahead of the vehicle. When an obstacle is detected, the ultrasonic sensor emits sound waves and calculates the distance by measuring their reflection time. The Application ECU processes this data and, based on obstacle detection, commands the dual H-bridge L298N motor drivers to adjust the DC motors. This mechanism allows the vehicle to effectively avoid collisions.

This entire process operates in a feedback loop, continuously detecting obstacles, measuring distances, and adjusting motor controls to dynamically navigate the vehicle around obstacles while maintaining its course towards the destination.

6.2. User-friendly Dashboard

6.2.1. Mobile Application: Overview

A Mobile Application must be used to enable vehicle owners to interact with the system's internal programming and execute commands such as accepting firmware updates, snoozing unnecessary updates, or requesting specific updates or guidance to resolve issues. This mobile application will be designed to run on the dashboard within the vehicle, simplifying the user's interaction with the system and providing a responsive dashboard for the vehicle.

Figure 6.12. Automotive User Interface

6.2.2. Mobile Application: Functionalities

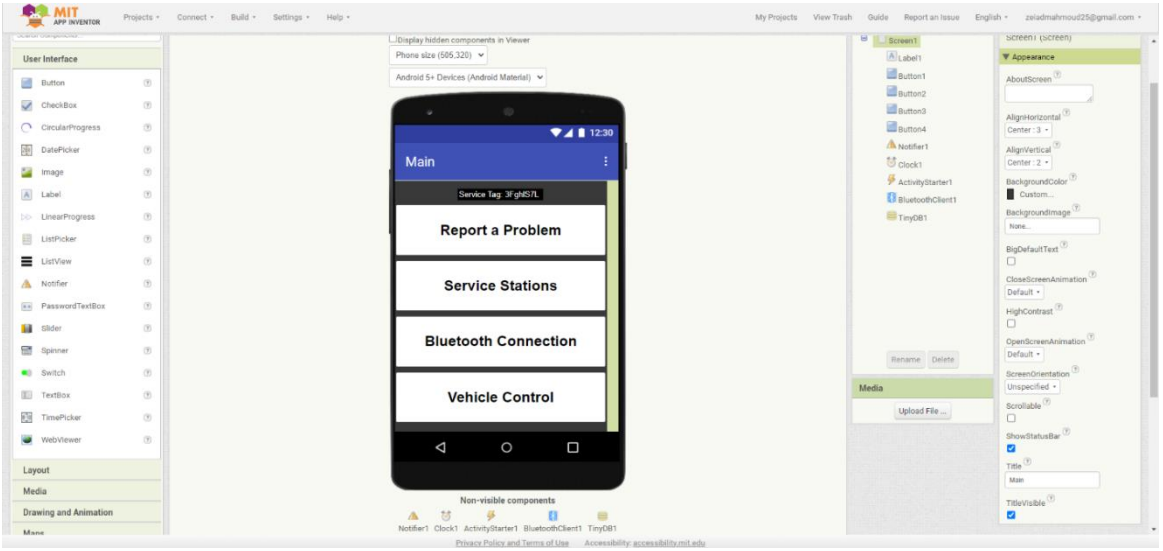


Figure 6.13. Mobile Application Main Page

1- Report a Problem

The "Report a Problem" function is a critical component located on the main page of the application. Its primary function is to allow users to report any issues they encounter with their system directly to the server. When a user clicks this button, they are presented with a text box to enter the details of their problem. Upon clicking the "Send" button, the entered message is transmitted via Bluetooth to the connected device, ensuring that the issue is promptly communicated for resolution.

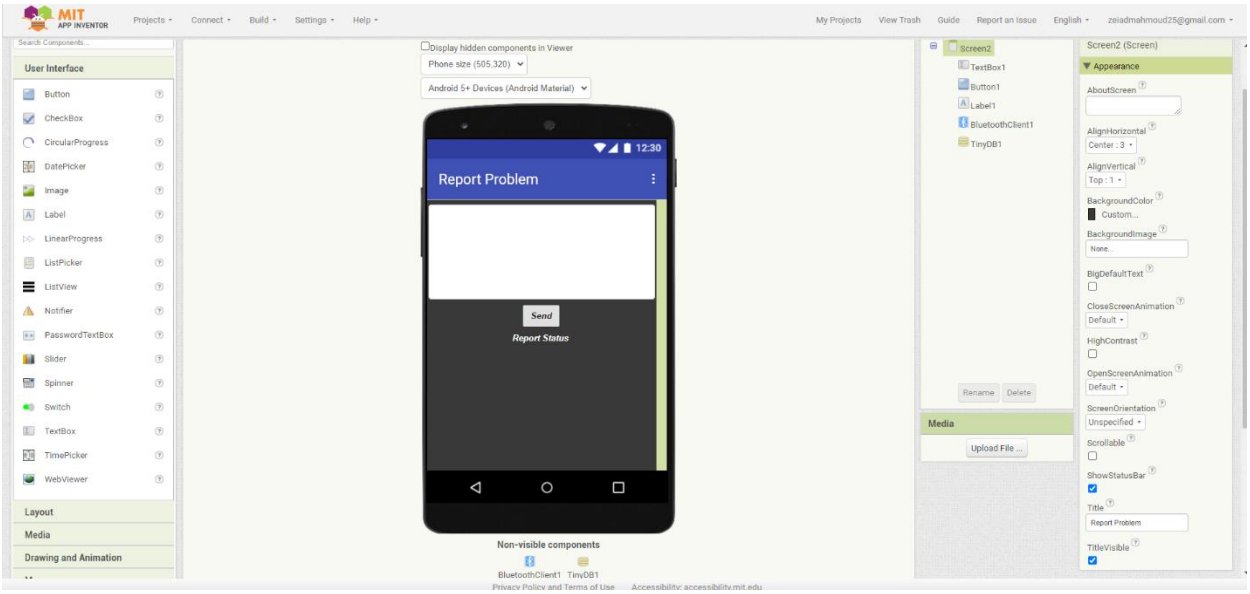


Figure 6.14. Report a Problem

Page 2 is designed to facilitate this reporting process efficiently. It includes the "Report Problem" button, a text input field for users to describe their issue, and a "Send" button to submit the report. The design is user-friendly, ensuring that users can easily navigate and report problems without any hassle. Additionally, the screen includes status indicators to inform the user about the connection status with the Bluetooth device, enhancing the overall user experience by providing immediate feedback on their actions.

2- Service Stations:

The "Service Stations" function on the app provides users with easy access to nearby service stations. Originally, this functionality depended on a GPS hardware module to determine the user's location. However, this has been replaced with a software-based solution, which leverages the device's built-in location services through Google Maps APIs. When a user clicks the "Service Stations" button, the app uses the device's native location capabilities to obtain the current coordinates.

These coordinates are then used to query an online service or database that returns a list of nearby service stations. The results are displayed on the screen, often with additional details such as distance, address, and available services.

By shifting from a GPS hardware module to a software solution, the application benefits from greater compatibility across different devices, reduced hardware costs, and simplified maintenance. The software-based approach also allows for more frequent updates and improvements, ensuring users have access to the most accurate and up-to-date location information.

3- Bluetooth Connection:

The "Bluetooth Connection" function is essential for establishing a connection between the app and external devices via the HC-05 Bluetooth module. When a user clicks this button, the app scans for available Bluetooth devices and presents a list for the user to select from. Upon selection, the app attempts to connect to the chosen device using the HC-05 module. Page 3 is designed to streamline this connection process. It includes the "Bluetooth Connection" button, a list picker for displaying available devices, and a status label that informs the user about the connection status. The screen might also feature additional options to disconnect or refresh the list of devices.

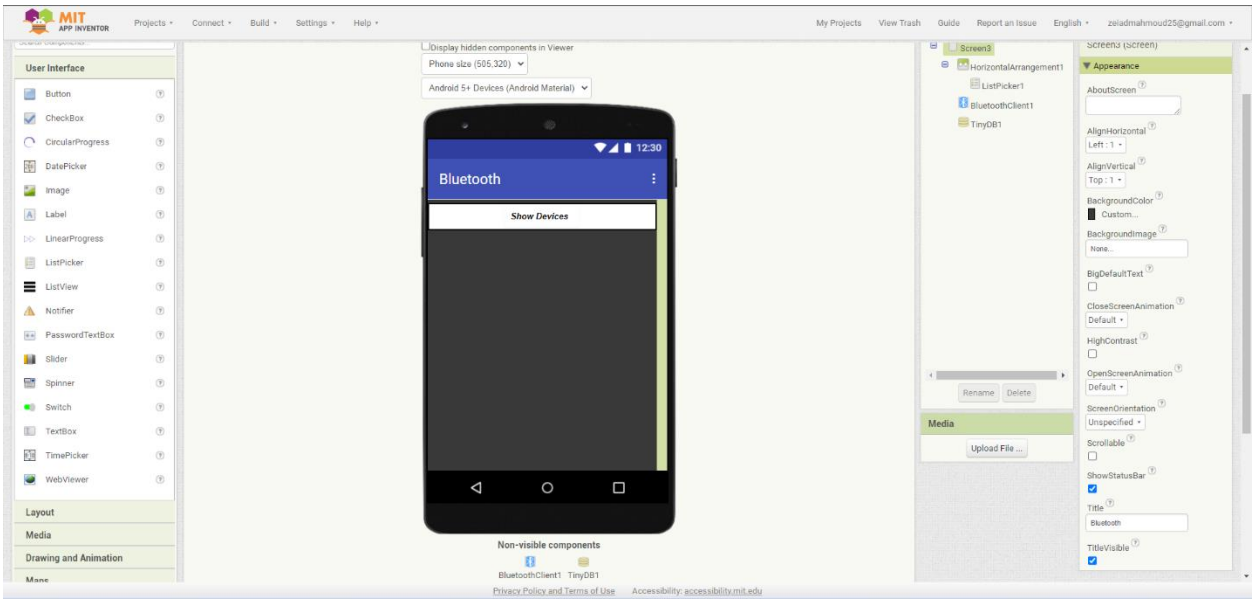


Figure 6.15. Bluetooth Connection

By using the HC-05 Bluetooth module, the app can send and receive data wirelessly, facilitating functionalities such as reporting problems, receiving updates, and more. The integration of the HC-05 module ensures robust and reliable communication between the app and the connected devices, enhancing the overall user experience by providing seamless connectivity and immediate feedback on their actions. In our case we are using it to connect with our application (i.e. obstacle avoidance vehicle) to perform the functionalities associated with the vehicle.

4- Vehicle Control:

The "Vehicle Control" function is specifically designed to enable controlling the movement of the vehicle in four directions: forward, backward, left, and right. When we interact with these buttons, they are provided with an interface to navigate the vehicle remotely. Page 4 is equipped with directional control buttons that correspond to the four movement directions. Each button is clearly labeled and positioned to allow intuitive control. The user can press these buttons to send commands to the vehicle, instructing it to move in the desired direction.

This page includes feedback mechanisms to inform the user about the vehicle's current status and movement. This feedback ensures that users are aware of the vehicle's responses to their commands, enhancing safety and control. The design of the page is optimized for ease of use, with a layout that minimizes the risk of errors and ensures that vehicle control is straightforward and responsive. In real life we don’t need this functionality as it’s used only to test the vehicle movements.

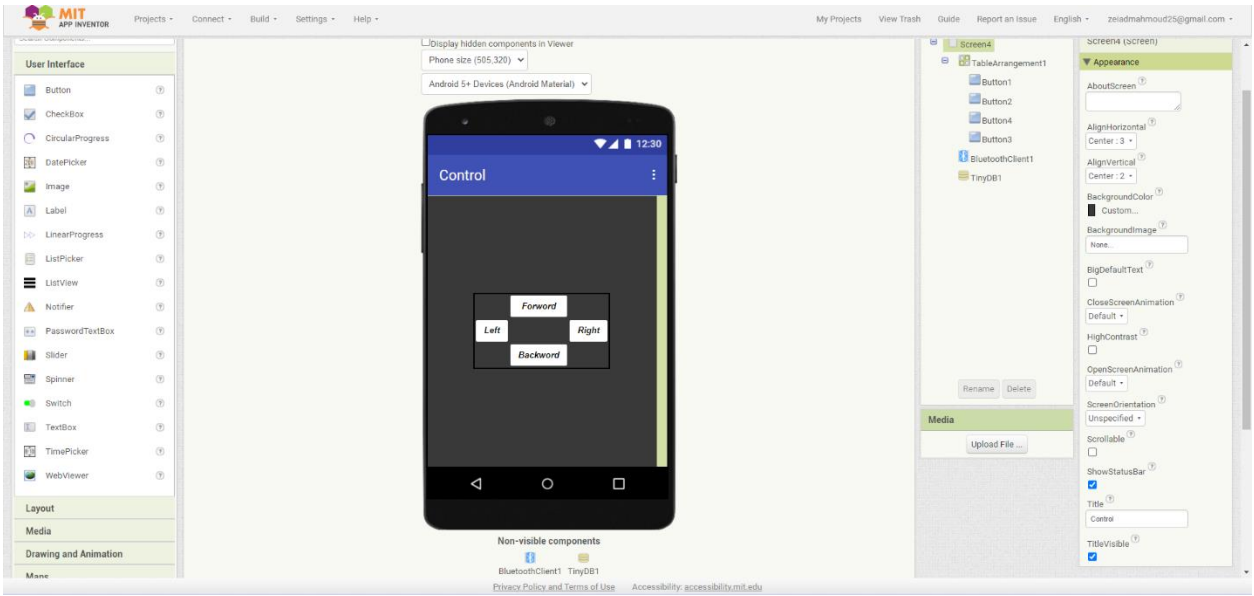


Figure 6.16. Vehicle Controls

6.2.3. Notifications and Alerts

Notifications serve as fundamental functions within the user interface, conveying essential information that users need to know about the system. Examples include notifications confirming the availability of a system firmware update, alerts warning of specific malfunctions in the vehicle, and reminders in case the user chooses to postpone an update to a later time. Additionally, notifications will prompt users to update the system once the snoozing time has elapsed, along with other important alerts aimed at helping users maintain the integrity of the vehicle's system. This enhances the user experience and interactions, serving our FOTA System scenarios and applications.

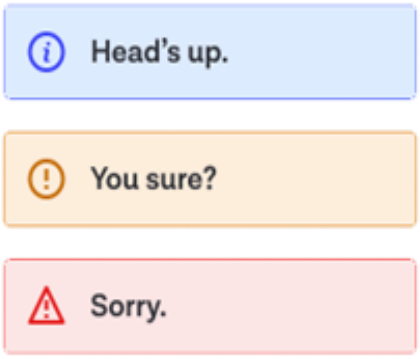


Figure 6.17. interactive notifications and alerts

6.2.3.1. Types of Messages and Alerts

1. **Additional Feature Updates:**

- **Trigger**: Sent by the server when a new optional feature is available for the user to download.
- **Notification Content**: "You have an additional feature update. Do you want to download it?"
- **User Actions**:
    - **Accept**: Initiates the download of the new feature.
    - **Snooze**: Delays the notification for a specified period.
- **Server Interaction**: The server sends a message with a specific code (e.g., "0") to indicate the presence of an additional feature update. The application receives this code and triggers the appropriate notification.

2. **Critical Updates**:

- **Trigger**: Sent by the server when a critical update is mandatory.
- **Notification Content:** "A critical update is downloading now."

- **User Actions:** No user action is required; the update process starts automatically.
- **Server Interaction:** The server sends a message with a different code (e.g., "1") to indicate the critical update. The application receives this code and immediately starts the download process, bypassing user consent.

6.2.3.2. Handling Notifications and Alerts

1. **Receiving Messages**:

   - **Bluetooth Client**: This event triggers when a new message is received from the server via Bluetooth. The message content is analyzed to determine the type of notification to display.
   - **Message Parsing**: The application parses the received data to identify the message code (e.g., "0" for additional features, "1" for critical updates).

2. **Displaying Notifications**:

   - **Notifier Component:** Used to show various types of notifications and dialogs to the user.
   - **ShowChooseDialog:** For additional feature updates, this dialog presents the user with "Accept" and "Snooze" options.
   - **ShowAlert:** For critical updates and problem reporting confirmations, this alert shows the relevant message without user interaction options.

3. **User Interaction**:

   - **Notifier.AfterChoosing**: This event handles the user's choice from the ShowChooseDialog.
     - Accept: Sends a confirmation message to the server and starts the update process.
     - Snooze: Enables a timer (e.g., Clock1) to re-display the notification after the specified delay.

4. **Timer Management**:

   - **Clock1.Timer**: Re-triggers the additional feature update notification if the user chooses to snooze it. This ensures the user is periodically reminded until they accept the update.

Example Workflow:

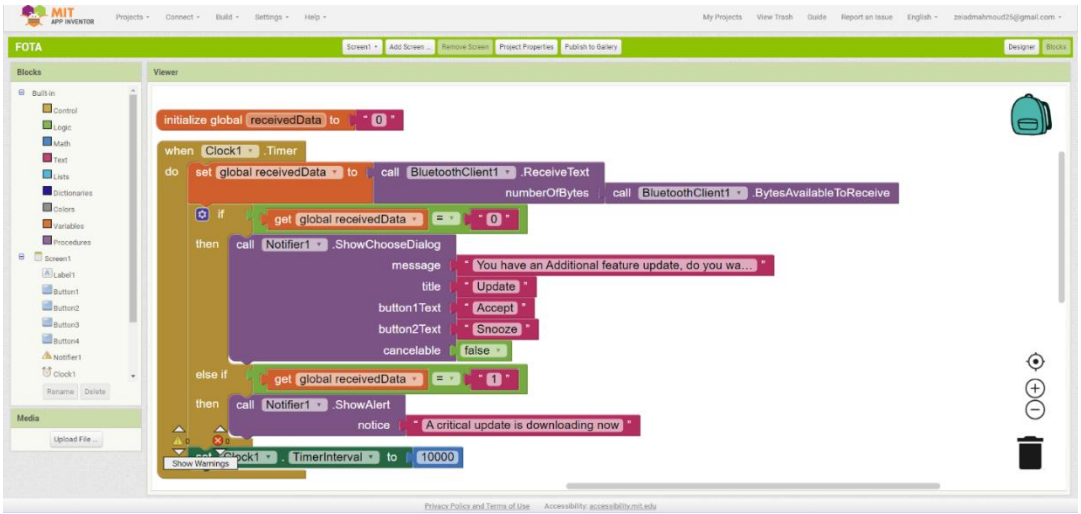
1- **Server sends "0"**: The application receives the code "0" via Bluetooth, triggering a ShowChooseDialog for an additional feature update.

2- **User clicks "Snooze"**: The application enables Clock1 to re-display the notification after 10 seconds.

3- **Clock1.Timer fires**: The ShowChooseDialog is displayed again until the user clicks "Accept".

4- **User clicks "Accept"**: The application sends a confirmation to the server and shows an alert, "The update is downloading".

5- **Server sends "1"**: The application immediately displays an alert, "A critical update is downloading now", and starts the download process.



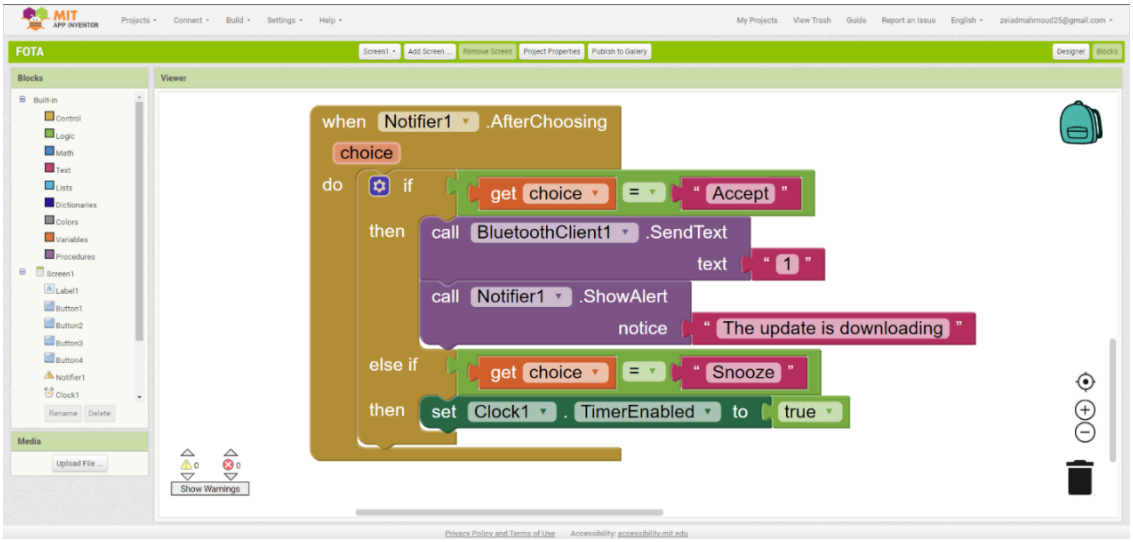


Figure 6.18. Handling Notifications and Alerts

7. FOTA Server

The FOTA server constitutes the core of the vehicle software management ecosystem, serving as a pivotal platform for orchestrating the release and updates of software across a diverse range of vehicles. Its primary objective is to ensure that vehicles are equipped with the latest features and improvements, thereby enhancing performance, functionality, and user experience. Moreover, the FOTA server offers flexibility through customization to align with OEM policies, thereby meeting specific requirements and regulatory standards unique to each car manufacturer.

7.2. Server Functionalities

7.2.1. Server Pages

The FOTA Server encompasses several essential pages that are instrumental in the management and operation of vehicle software:

1- Login Page:

- Members access the FOTA server securely using their credentials.
- Initial login involves generating a QR code via Google Authenticator for added security.
- Subsequent logins require entering credentials followed by an OTP sent to the member's registered Google Authenticator, ensuring robust authentication.

Figure 7.1. Server Login Page

2- Uploads Page:

The Upload Page on our FOTA server is a pivotal hub where members, including Original Equipment Manufacturers and developers, securely upload updates and upgrades, ensuring continuous enhancement of vehicle software. This platform fosters collaboration among stakeholders, enabling them to contribute to ongoing improvements and integrate the latest technological advancements into our software ecosystem.

- **Secure Uploads**: Members can securely upload updates and upgrades, leveraging advanced encryption technologies such as Symantec AES, which ensures robust data confidentiality and integrity throughout transmission and storage. This rigorous security framework protects sensitive information from unauthorized access, bolstering trust and reliability among users.

- **Streamlined Process**: Our platform streamlines the upload process, facilitating efficient management of software enhancements. This includes version control mechanisms that maintain consistency across diverse vehicle models and software configurations, ensuring seamless integration and optimal performance of uploaded updates.

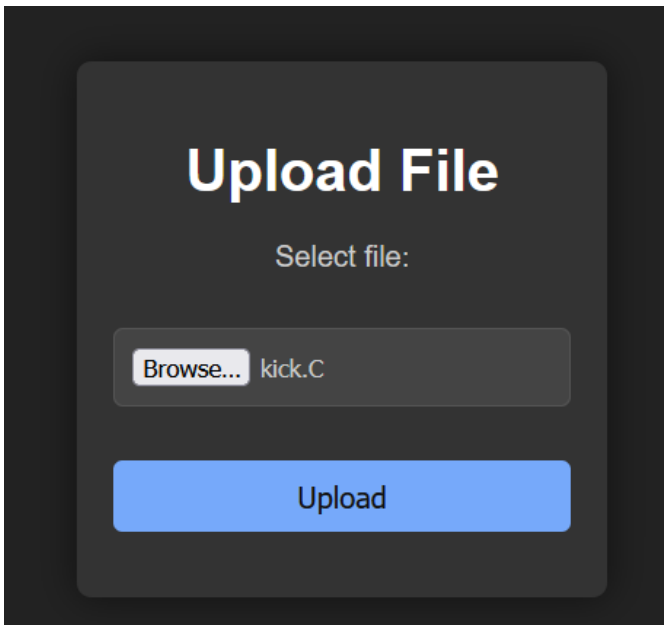


Figure 7.2. Server Uploads Page

3- Dashboard:

The Dashboard serves as a personalized control center for each member, ensuring they have exclusive access to manage and monitor their software contributions with precision and efficiency.

- **Personalized Control Center**: Each member enjoys a dedicated Dashboard tailored to their specific needs and responsibilities within the ecosystem. This personalized approach ensures that members can navigate and oversee their uploaded files and software activities seamlessly.
- **Detailed File Management**: The Dashboard provides comprehensive visibility into uploaded files, presenting detailed information such as file names, upload dates, sizes, and download options. This detailed overview empowers members to maintain accurate software versioning and effectively coordinate development efforts.

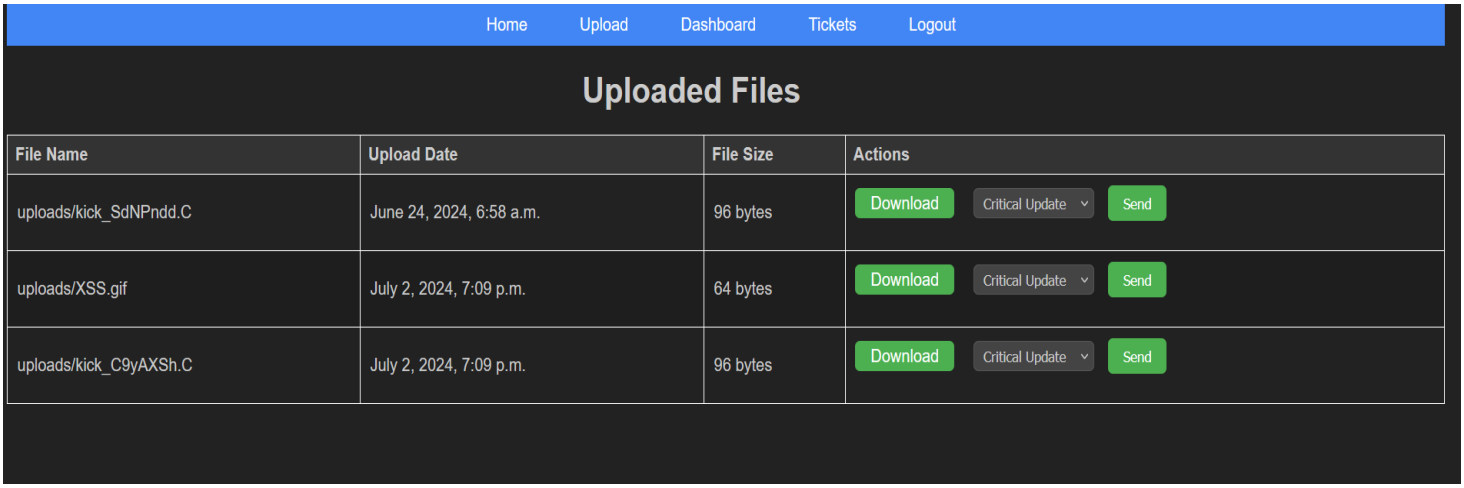


Figure 7.3. Server Dashboard

- **Actions:** Members benefit from actions available through the Dashboard. They can initiate critical updates ("push") directly to vehicles, ensuring timely deployment of essential patches and enhancements. Additionally, members can request feature updates ("pull") that require administrative approval, promoting collaboration and strategic alignment.

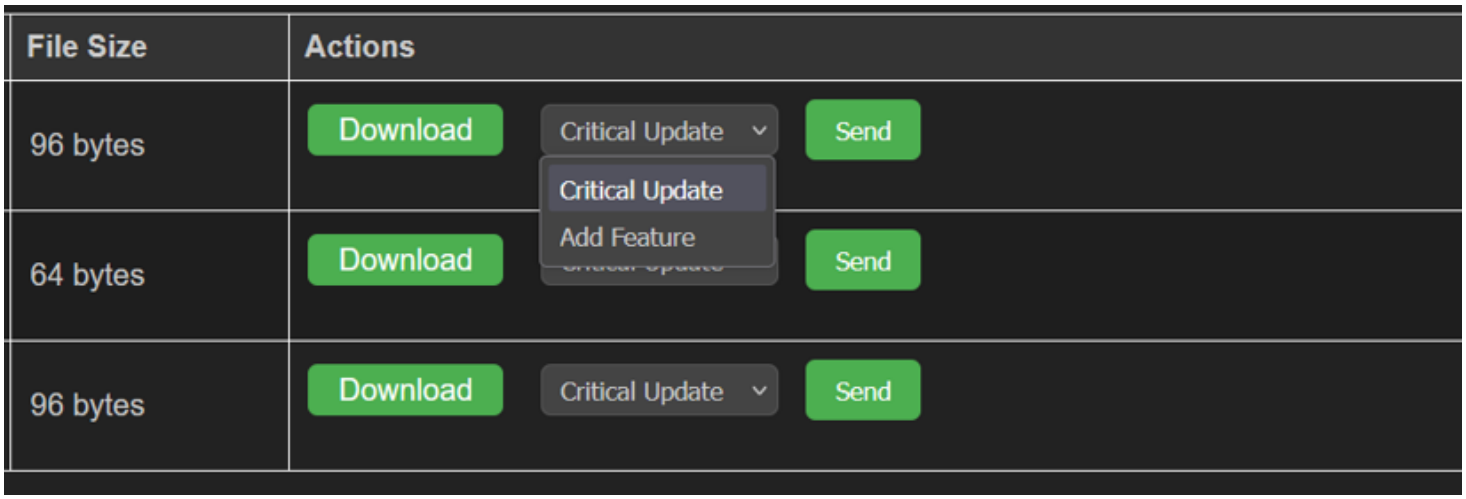


Figure 7.4. Server Actions

- **Download Capability**: Members have the ability to download files directly from their Dashboard, facilitating easy access to uploaded software updates and resources. This feature enhances workflow efficiency and supports seamless integration of new software versions into vehicle systems.
- **Enhanced User Experience**: Designed for usability and accessibility, each member's Dashboard enhances user experience by prioritizing intuitive navigation and real-time updates. It facilitates proactive management of software lifecycles, keeping members informed and engaged in optimizing software performance.

4- Tickets Page:

- **Customer Complaint Submission:** Users submit complaints, service requests, and enhancement suggestions related to vehicle software through this dedicated platform.
- **Ticket Details:** Each ticket includes comprehensive information such as the customer's name, contact details, submission date, and specific details regarding the issue or request.
- **Continuous Improvement:** Utilizes data from tickets to identify trends, address recurring issues, and implement improvements in vehicle software. This proactive approach enhances software reliability and customer satisfaction over time.

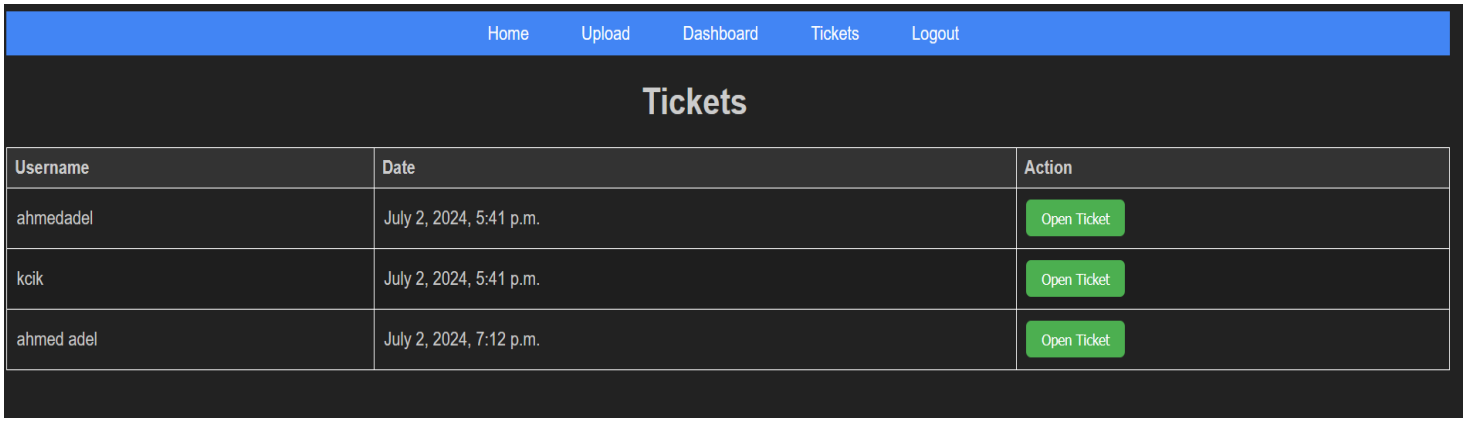


Figure 7.5. Tickets Page

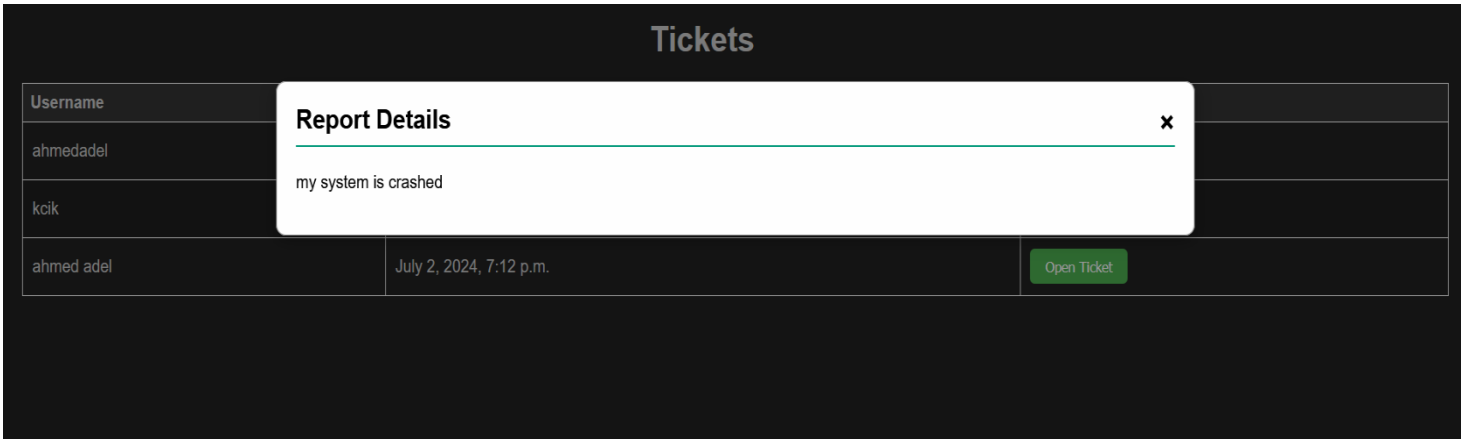


Figure 7.6. Report Details

5- Admin Panel

- **Admin Login**: Administrators access the Admin Panel securely by entering their credentials followed by a one-time password (OTP) generated via two-factor authentication (2FA). This robust authentication process ensures enhanced security and prevents unauthorized access to sensitive administrative functionalities.
    - **Two-Factor Authentication (2FA)**: Implemented using industry-standard protocols such as Google Authenticator or similar tools, 2FA adds an additional layer of security by requiring administrators to verify their identity with a time-sensitive OTP sent to their registered device.

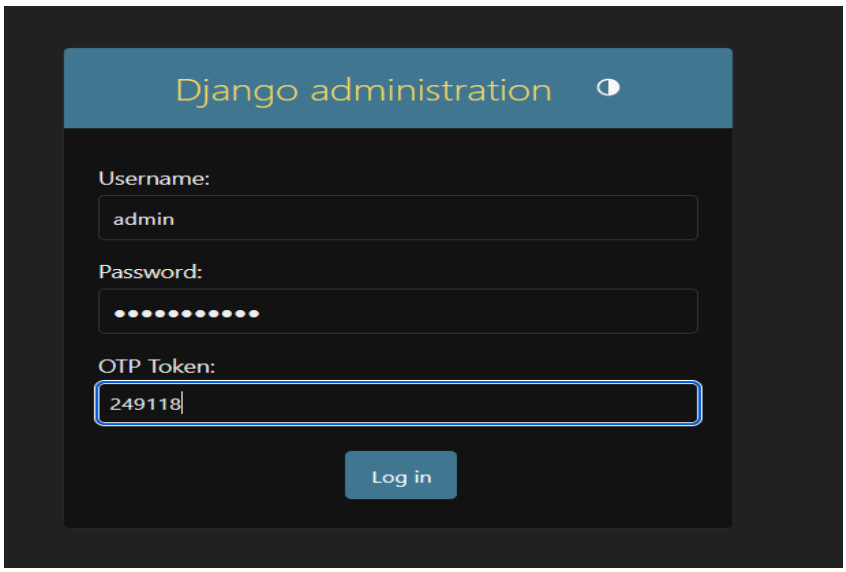


Figure 7.7. Admin Login

6- TOTP Device Management page

The TOTP Device Management page in our Admin Panel allows administrators to securely manage their Time-based One-Time Password (TOTP) devices, ensuring robust two-factor authentication (2FA) for accessing sensitive administrative functions.

1- Add TOTP Device:

- Administrators can add new TOTP devices to their accounts securely.
- Generates QR codes for easy setup of TOTP on compatible authenticator apps.

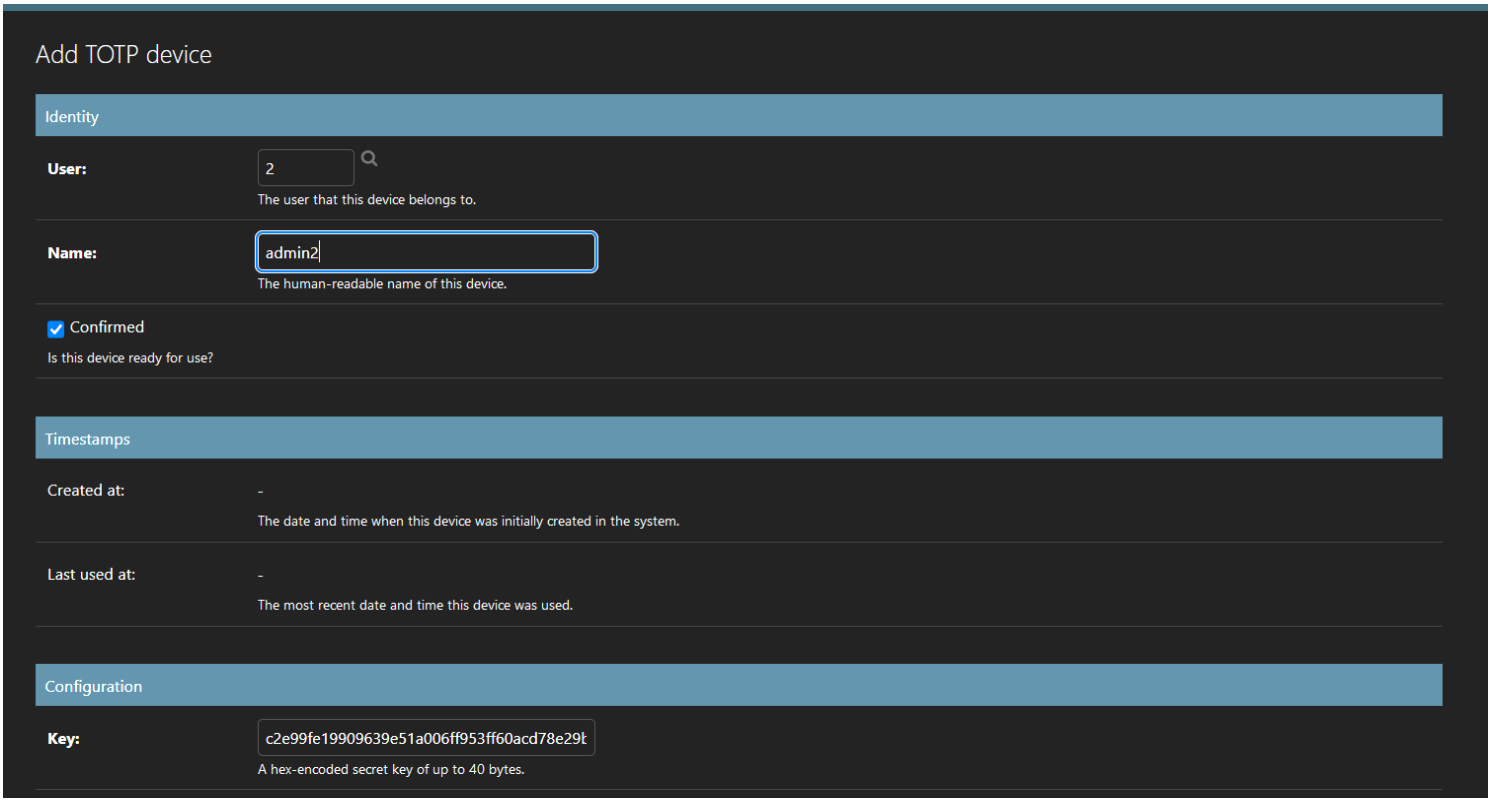


Figure 7.8. Add TOTP Device

2. Delete TOTP Device:

- Provides the option to remove outdated or compromised TOTP devices from administrator accounts.
- Ensures security integrity by preventing unauthorized access through old devices.

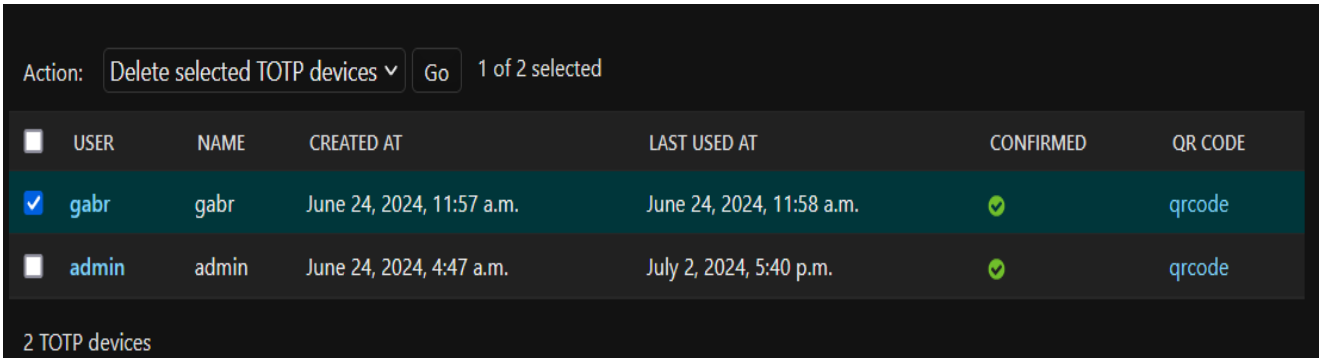


Figure 7.9. Delete TOTP Device

3. **Show TOTP**:

   - Allows administrators to reset TOTP settings if they forget their TOTP device.
   - Generates a new QR code for re-establishing 2FA access, ensuring continuous security.

Figure 7.10. Show TOTP QR Code

- **Security Measures**:
  - Encrypts TOTP data to safeguard sensitive information stored within the Admin Panel.
  - Implements stringent authentication protocols to verify administrator identities during TOTP management.
- **User Interface**:
  - Provides an intuitive interface for administrators to manage their TOTP devices effectively.
  - Enhances user experience with clear instructions for adding, deleting, and resetting TOTP settings.

7- Accounts Management Page

- **User and Admin Overview**: Provides a comprehensive overview of all users and administrators within the system, detailing their roles, permissions, and current active status. This visibility ensures efficient management and oversight of user access across the FOTA server.

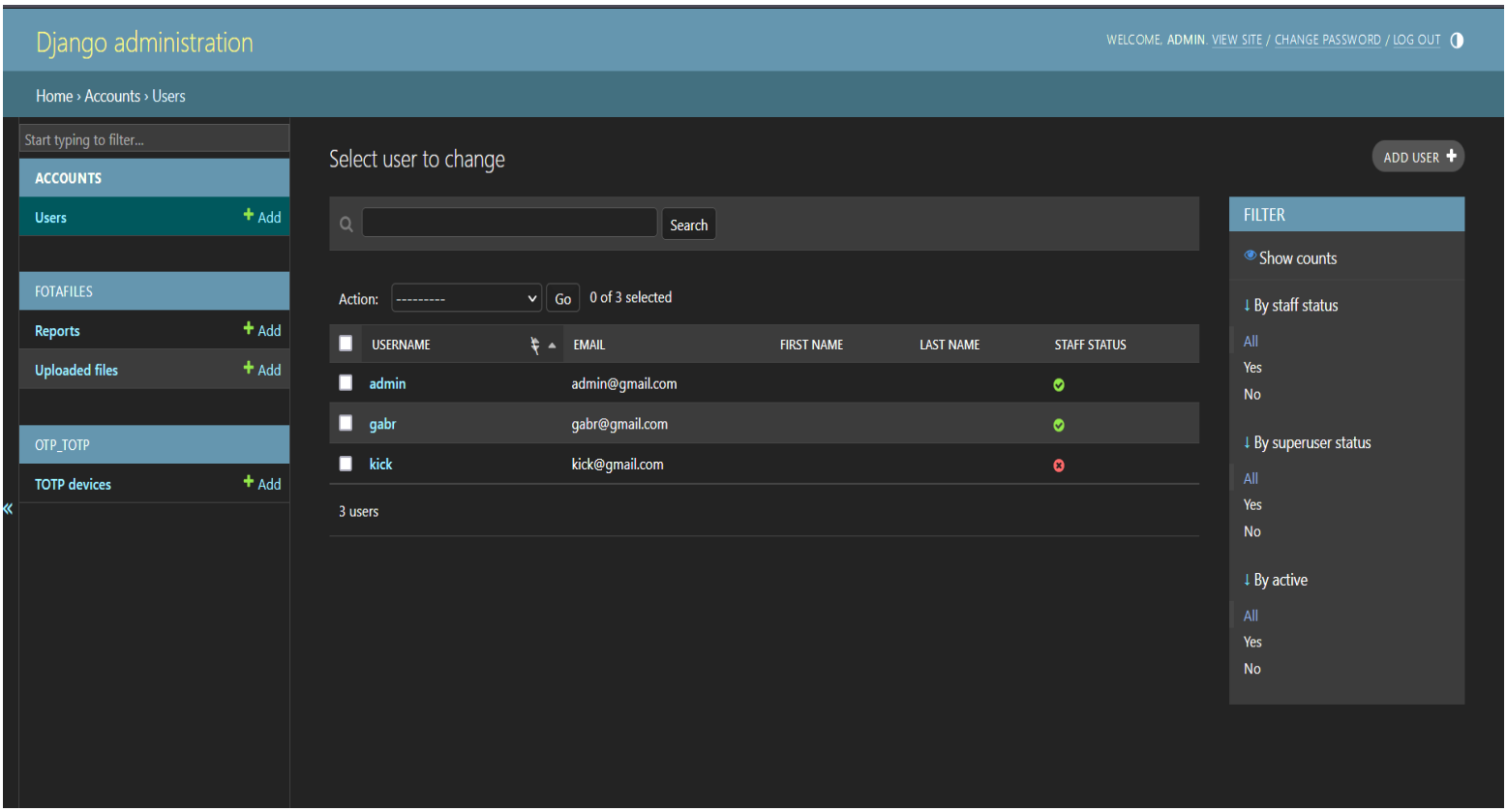


Figure 7.11. Accounts Management Page

- **Account Deletion**: Empowers administrators with the capability to delete user accounts as needed. This functionality helps maintain system integrity by promptly removing inactive or unauthorized accounts, thereby safeguarding against potential security risks and ensuring compliance with organizational policies.

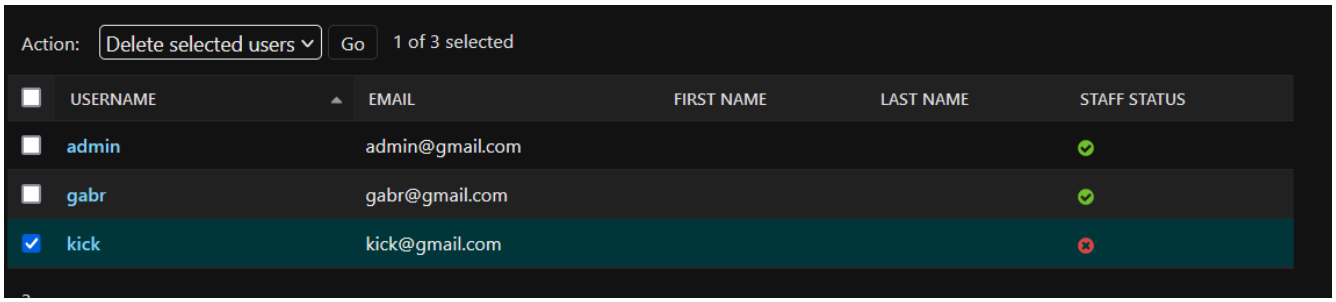


Figure 7.12. Account Deletion

- **QR Code Resend**: Facilitates the resend of QR codes to members during their initial login process. This feature enhances user experience by enabling seamless authentication via Google Authenticator, ensuring secure access to the FOTA server while minimizing login-related issues and operational disruptions.

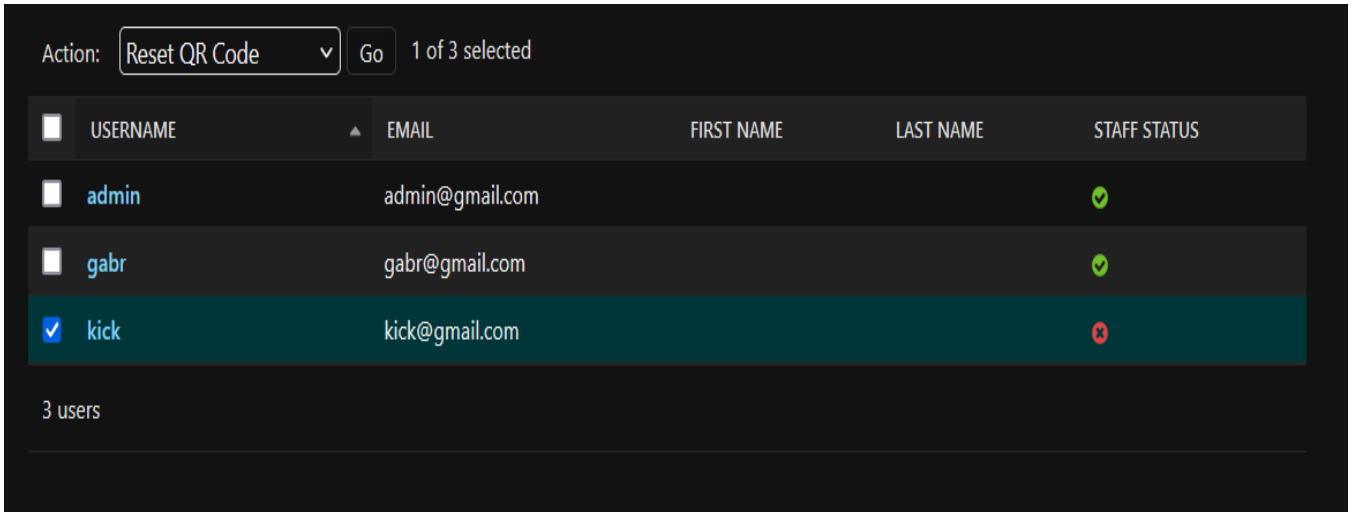


Figure 7.13. QR Code Resend

8- Report Management Page

- **Report Overview**: Displays all reports generated by users, providing essential details such as report names, submission dates, and associated users. This centralized view enables administrators to efficiently track and manage the flow of reports within the FOTA server.
- **Delete Reports**: Offers administrators the capability to delete reports as necessary. This functionality ensures data integrity and system cleanliness by removing outdated or irrelevant reports, thereby optimizing storage space and maintaining an organized reporting environment.

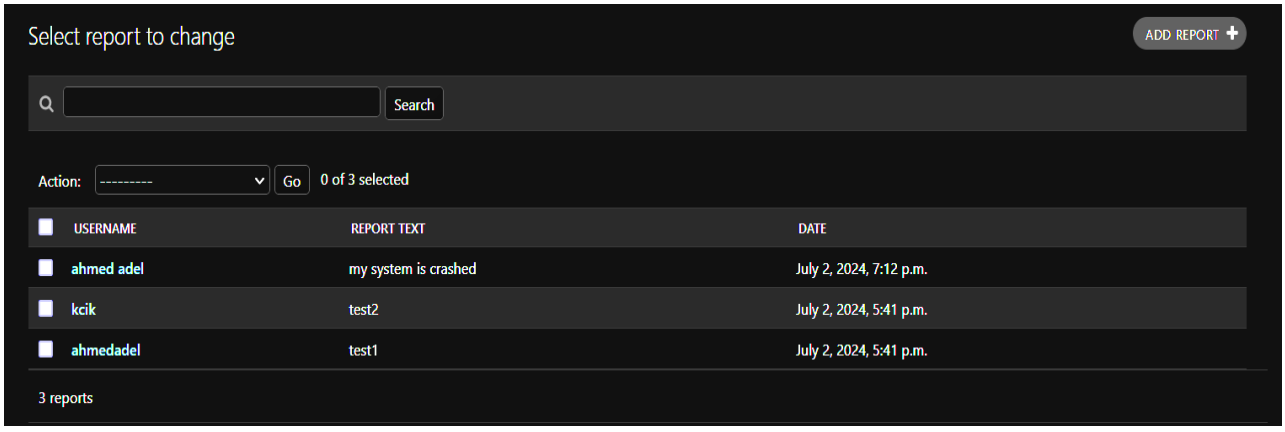


Figure 7.14. Report Management Page

This feature-rich report management page enhances administrative capabilities within the FOTA server, promoting streamlined operations and effective oversight of user-generated reports

9- Uploaded Files Management

Within the Admin Panel, the Uploaded Files Management section serves as a critical component for overseeing and maintaining the integrity of software updates contributed by members across the ecosystem. This section provides comprehensive tools and functionalities designed to enhance system governance, security, and operational efficiency.

1. **Comprehensive File Listing**:
    - Presents a detailed catalog of all uploaded files, meticulously organized by member names, upload dates, and specific file attributes such as size and type. This structured approach enables administrators to quickly locate, review, and manage each file within the repository.

2. **Secure and Efficient File Deletion**:
    - Empowers administrators with the capability to securely delete files that are obsolete, redundant, or potentially malicious. This proactive management approach mitigates risks associated with outdated software versions and enhances system hygiene.

3. **Member Attribution and Accountability**:
    - Associates each uploaded file with the corresponding member's identity, providing clear visibility into the origin of software contributions. This attribution enhances accountability within the ecosystem, enabling swift identification and response to potential issues or discrepancies.

    - Facilitates effective communication and resolution strategies, ensuring that administrators can collaborate with members to address any concerns related to specific file uploads.

4. **Configuration Item Management**:
    - Adheres to best practices outlined in ITIL framework for Configuration Management. Each uploaded file is treated as a configuration item, meticulously tracked throughout its lifecycle to support system stability and compliance.

    - Implements version control mechanisms and documentation practices to maintain a comprehensive record of changes, updates, and dependencies associated with each file, fostering a structured approach to system configuration.

5. **Operational Insights and Continuous Improvement**:
    - Provides actionable insights into file utilization patterns, performance metrics, and system usage trends. These insights enable administrators to make informed decisions regarding resource allocation, system optimizations, and future software development initiatives.

    - Supports continuous improvement initiatives by analyzing upload behaviors and identifying opportunities to enhance operational efficiency, user satisfaction, and overall system performance.

6. **File Addition Functionality**:
    - Facilitates seamless integration of new software updates and upgrades through an intuitive file addition function within the Admin Panel. This functionality enables administrators to upload and distribute files across the ecosystem, ensuring timely deployment and availability of the latest enhancements.

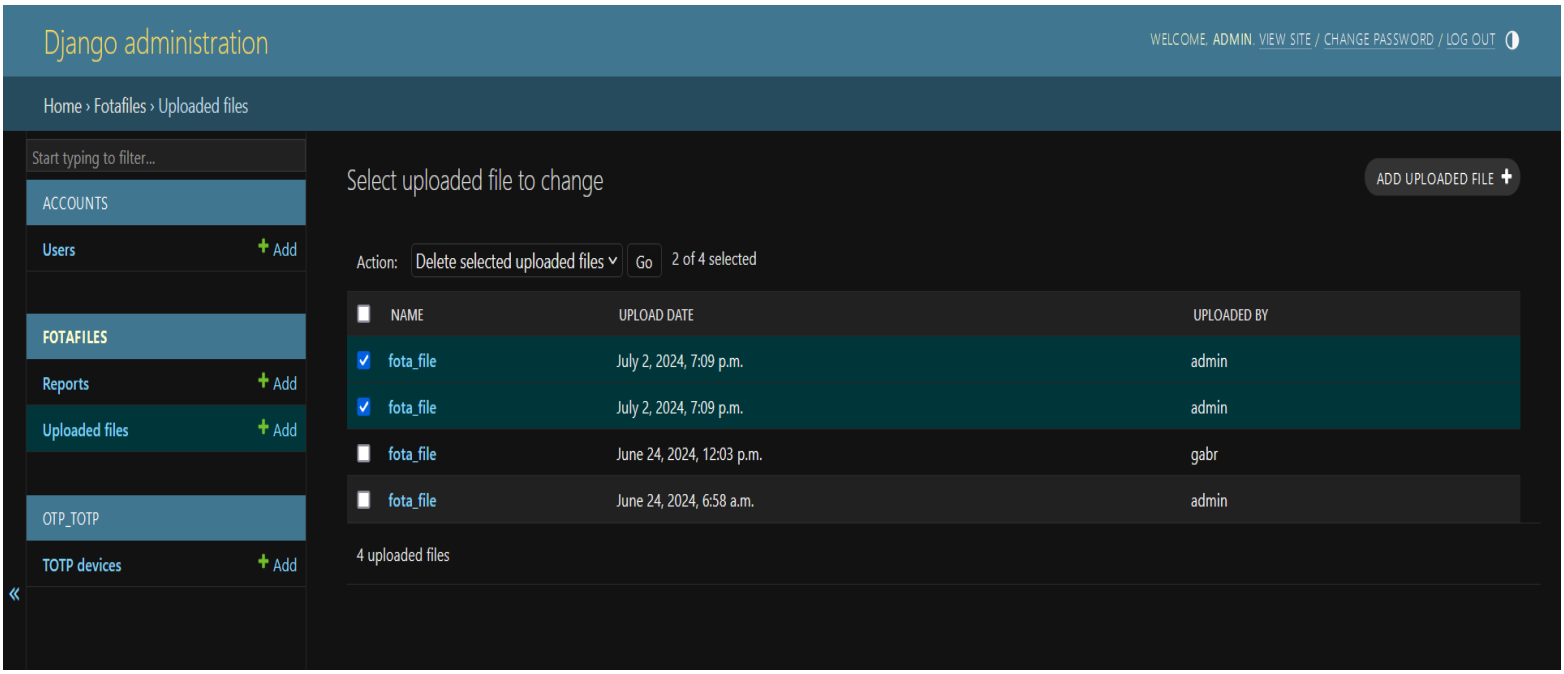


Figure 7.15. Uploaded File Management

The Uploaded Files Management section within the Admin Panel epitomizes a robust framework engineered to optimize software governance, foster collaborative innovation, and uphold stringent security standards. By leveraging advanced functionalities and adhering to industry best practices, this section fortifies the FOTA server's role as a dependable platform for managing and distributing vehicle software updates effectively.

7.2.2. Implementation of Access Control:

1- **Role-Based Access:** Access to specific Dashboard features and functionalities is controlled through role-based access control . Members' access privileges are determined by their roles within the ecosystem, ensuring that only authorized actions and data are accessible. For example, developers may have permissions limited to uploading and testing software updates, while administrators have broader access for overseeing and approving critical updates.

2- **Customized Permissions:** Each member's Dashboard is tailored to reflect their roles and responsibilities within the ecosystem. Customized permissions extend to activities such as uploading, managing, and distributing software updates. Administrators have the capability to configure permissions based on specific roles, granting or restricting access to critical update functionalities accordingly. This approach enhances operational efficiency by aligning access levels with member responsibilities.

3- **Configuration Item (CI) Management:** Utilizing CI principles from the Intel framework, the FOTA server maintains rigorous management of configuration items. CI management ensures accurate tracking, versioning, and deployment of software updates, enhancing traceability and consistency throughout the software development lifecycle. By integrating CI practices, the FOTA server optimizes deployment processes and maintains high standards of quality and compliance with industry regulations.

7.2.3. Server Privileges

1- **Member Privileges:**

- **File Management:**
  - Members can securely upload and download files to and from the server, facilitating the continuous enhancement of vehicle software.
  - They have the ability to manage uploaded files, including viewing file details such as names, dates, and sizes, to ensure effective software contributions.

- **Ticket Management:**
  - Members are responsible for managing tickets submitted by users ("cars") regarding software issues or requests for enhancements.
  - They prioritize, track progress, and communicate resolutions to ensure timely and effective customer support.

2- Admin Privileges:

- **User Management:**
  - Admins have the authority to add new users and grant access to the FOTA server, ensuring a secure and regulated ecosystem. They can delete users as necessary to maintain system integrity and security.

- **QR Code Management:**
  - Admins are empowered to resend QR codes to members for authentication purposes during login processes, ensuring seamless access to the FOTA server. They have access to view QR codes specifically designed for administrative purposes, which differ significantly from those issued to members.

- **Ticket Administration:**
  - Admins oversee all tickets submitted by users ("cars"), including the ability to view, add, and delete tickets as part of their comprehensive management responsibilities. They ensure efficient ticket resolution and prioritize user concerns to optimize service delivery and user satisfaction.

- **Uploaded Files Management:**
  - Admins have visibility into all uploaded files, including details such as file names, upload dates, and contributors' identities. They can delete or add files as necessary, facilitating effective file management and ensuring compliance with organizational policies and regulatory requirements.

8. FOTA Overflow

8.1. Scenario Case (1)

The original equipment manufacturer (OEM) has introduced a new firmware feature, providing vehicle owners with the option to either accept or decline the update through the user interface, particularly when the update is **not critical**.
In the event of refusal, the update will be temporarily deferred or "snoozed." Conversely, upon acceptance, the firmware will be transmitted through the Node MCU. Subsequently, the bootloader will initiate the flashing process, transferring the update file to the targeted Electronic Control Unit (ECU). This mechanism ensures a formalized and user-centric approach to firmware updates within the automotive system.

Upon the vehicle owner's decision to snooze the firmware update, a snooze request is transmitted to the Main ECU through Node ECU as a Gateway. Subsequently, the Main ECU activates the Timer Peripheral, initiating a countdown/up for a predefined duration. Upon the expiration of this period, the Timer Peripheral generates an interrupt request, prompting the Main ECU via the Nested Vectored Interrupt Controller (NVIC).
Following this event, the Main ECU issues a notification, prominently displayed on the vehicle dashboard.
The notification offers the vehicle owner the opportunity to reevaluate their decision, presenting the option to either accept the update or opt for a further snooze period. This formalized process ensures a systematic and user-friendly approach to firmware update management within the vehicle system.

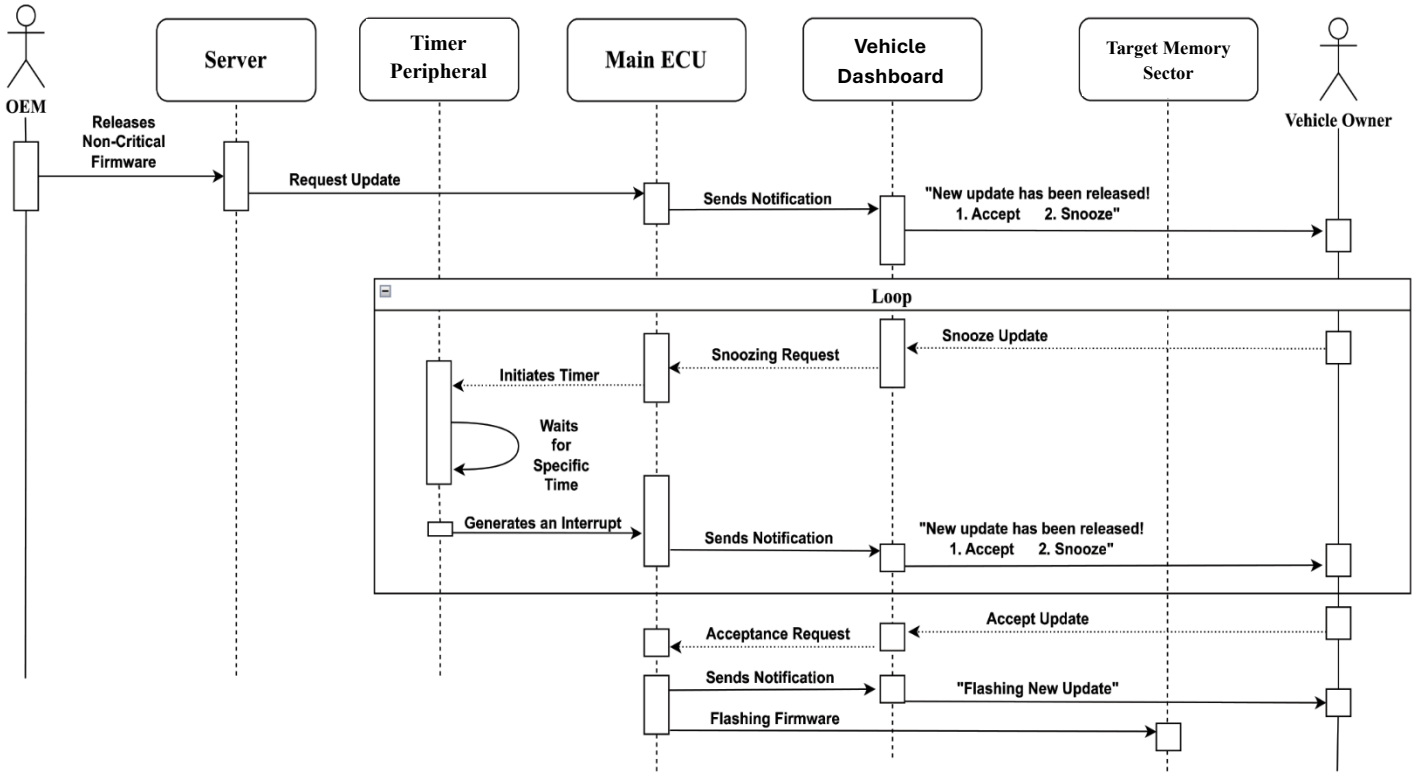


Figure8.1. The first scenario described by a sequence diagram

8.2. Scenario Case (2)

The OEM has introduced a critical firmware update that starts automatically without asking the vehicle owner to either accept or snooze the update. A notification will simply appear on the vehicle dashboard to let the owner know that the update is starting. The Main ECU's bootloader will then handle the process of flashing the update on the Target ECU.
This way, important updates can happen smoothly and quickly in the vehicle's system.

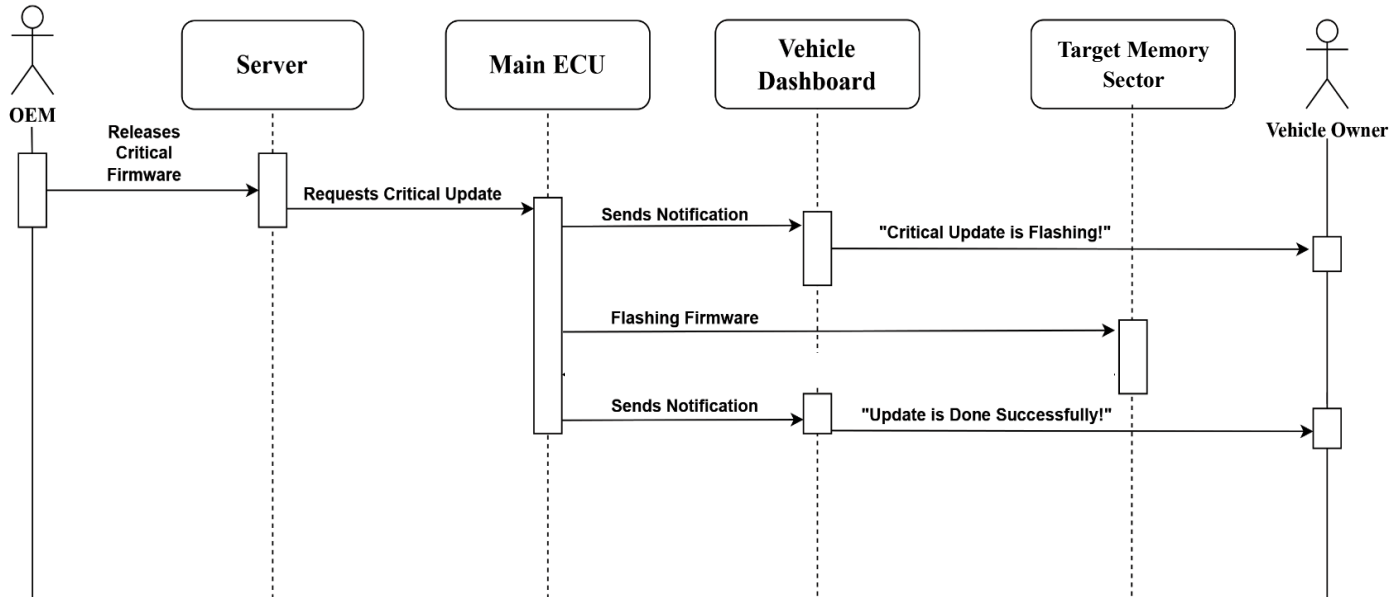


Figure8.2. The second scenario described by a sequence diagram

8.3. Scenario Case (3)

A user reports a problem through the vehicle dashboard, the OEM examines the report. Depending on the nature of the issue, the OEM provides the user with either a relevant update or guidance instructions. If an update is required, the firmware is transmitted, and the Bootloader subsequently flashes the update onto the target ECU. If the problem persists, the Main ECU uses Google Maps API to detect the user's location and directly shows information about the nearest service stations.
This comprehensive approach ensures efficient problem resolution within the vehicle system, incorporating both software updates and on-the-ground support.

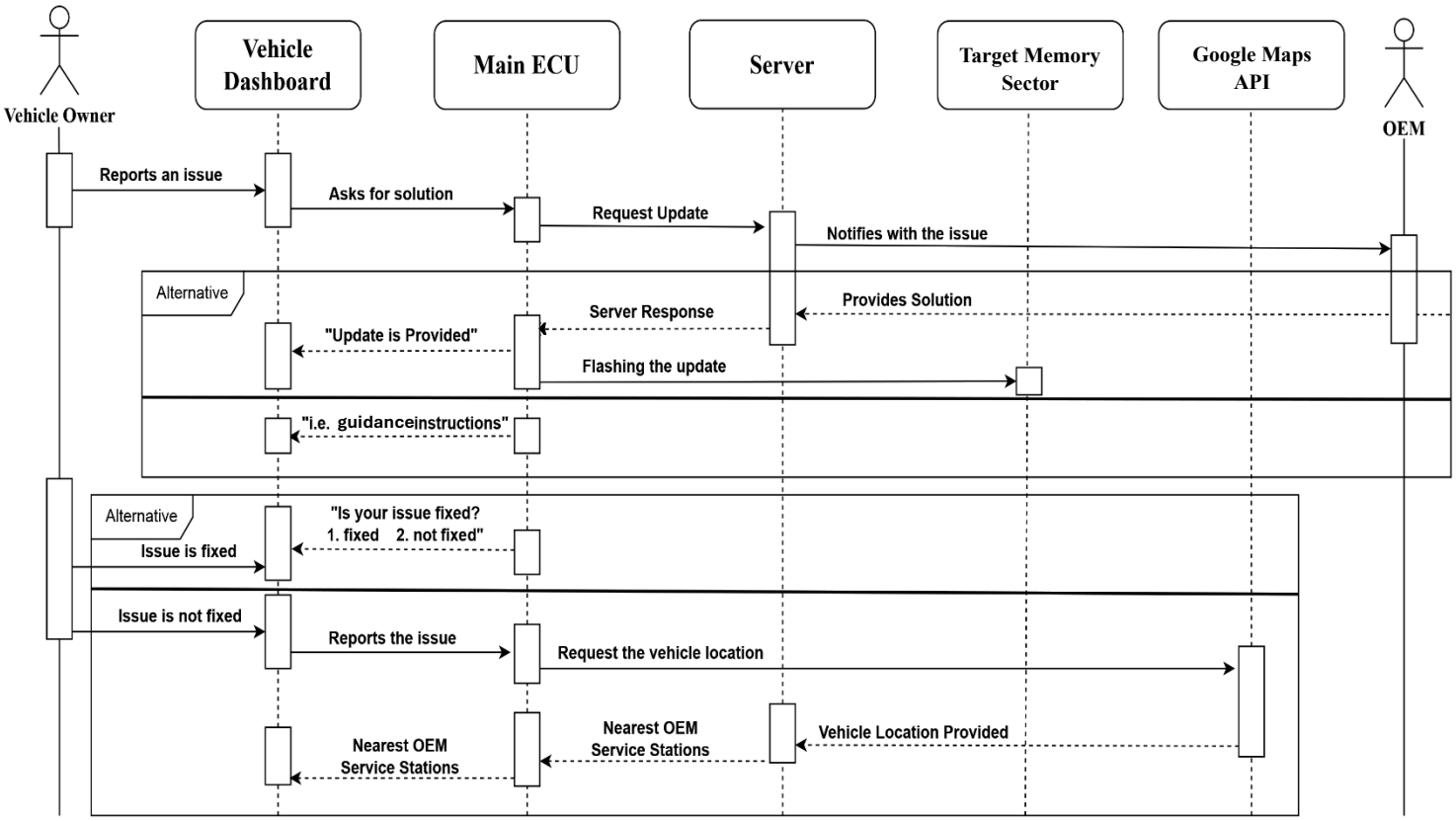


Figure8.3. The third scenario described by a sequence diagram

9. Automotive Cybersecurity Techniques

9.1. Bootloader

A bootloader is a small program or piece of code that resides in the non-volatile memory of a microcontroller. Its primary function is to initialize the system and load the main application program (firmware) into the system's RAM or flash memory for execution.

Bootloader Characteristics

- **Initialization:** The bootloader is the first code that runs when the embedded system is powered on or reset. Its main task is to initialize the hardware components, set up the necessary peripherals, and prepare the system for the main application.
- **Loading Firmware:** The bootloader's most critical function is to load the main application firmware into memory.
- **Communication Interface**: Bootloaders often include communication interfaces that allow for firmware updates.
- **Security:** Many embedded systems bootloaders incorporate security features to ensure that only authorized firmware can be loaded onto the device.
- **Flexibility and Customization:** Bootloaders can be customized to meet the specific requirements of the embedded system.

9.2. Secure Boot

Secure Boot ensures that the boot technology and operating system software are the legitimate manufacturer version and have not been altered or tampered with by any malicious actor or process.

9.2.1. How it works?

This process is crucial for protecting a system against unauthorized access and ensuring that only trusted software is executed.

Figure 9.1. shows an explanation of each step:

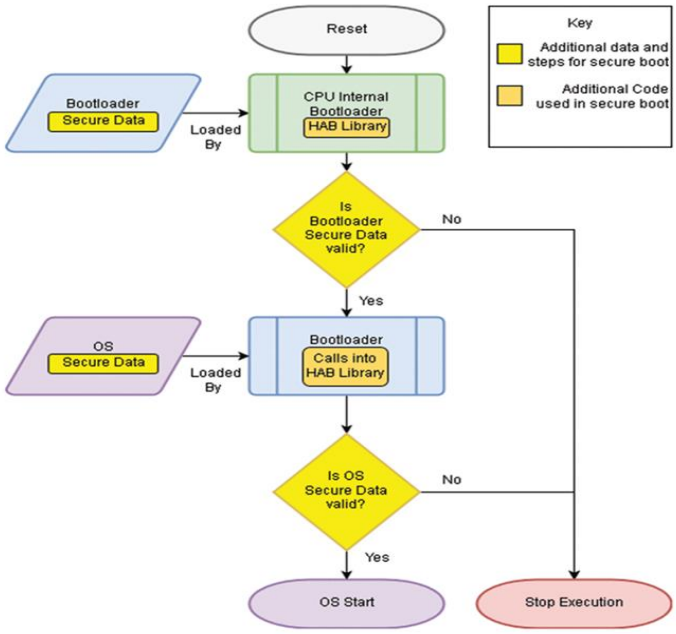


Figure 9.1. Secure Boot flowchart

- **Reset**: The system starts or resets, initiating the boot process.

- **CPU Internal Bootloader (HAB Library):** The Hardware Abstraction Boot (HAB) library is part of the CPU's internal bootloader, which is a trusted piece of code stored in read-only memory (ROM). This is the first code that runs when the system starts and is responsible for initializing the hardware and loading the external bootloader.

- **Bootloader Secure Data Validation:** The HAB library checks the bootloader's secure data, which typically includes cryptographic signatures or hashes. This secure data is used to verify the integrity and authenticity of the bootloader, ensuring it hasn't been tampered with.

    1. If the bootloader's secure data is not valid, the process stops, preventing the system from booting an untrusted bootloader.

    2. If the secure data is valid, the bootloader is loaded, and execution continues.

- **Bootloader Loads OS Secure Data**: Once the bootloader is validated and loaded, it proceeds to load the Operating System (OS) along with its secure data. The bootloader may also call into the HAB library to perform additional security checks.

- **OS Secure Data Validation**: The HAB library or similar trusted mechanism checks the OS's secure data to validate its integrity and authenticity.

    1. If the OS secure data is not valid, the system halts, stopping the execution to prevent an untrusted OS from loading.

    2. If the OS secure data is valid, the OS starts, and the system continues to boot into the operational environment.

    This secure boot process ensures that each stage of the system startup is validated, creating a chain of trust from the hardware up to the operating system. Any failure in this chain will prevent the system from booting, which is a security feature designed to protect against rootkits, bootkits, and other low-level threats.

9.3. Cryptographic Security

AES (Advanced Encryption Standard) Algorithm

AES is a symmetric block cipher Algorithm.

- It employs key sizes of 128, 192, and 256 bits for encryption, enhancing its robustness against hacking attempts.
- AES is widely used as a security protocol in various applications, including wireless communication.
- The algorithm operates on fixed-length blocks of data and uses a symmetric key for both encryption and decryption processes.
- Brute force attacks on AES-128 require approximately 2^128 attempts to break, highlighting its strong resistance to decryption through brute force methods.
- The algorithm's resistance to various cryptographic attacks, including differential and linear cryptanalysis, adds to its credibility and widespread adoption.

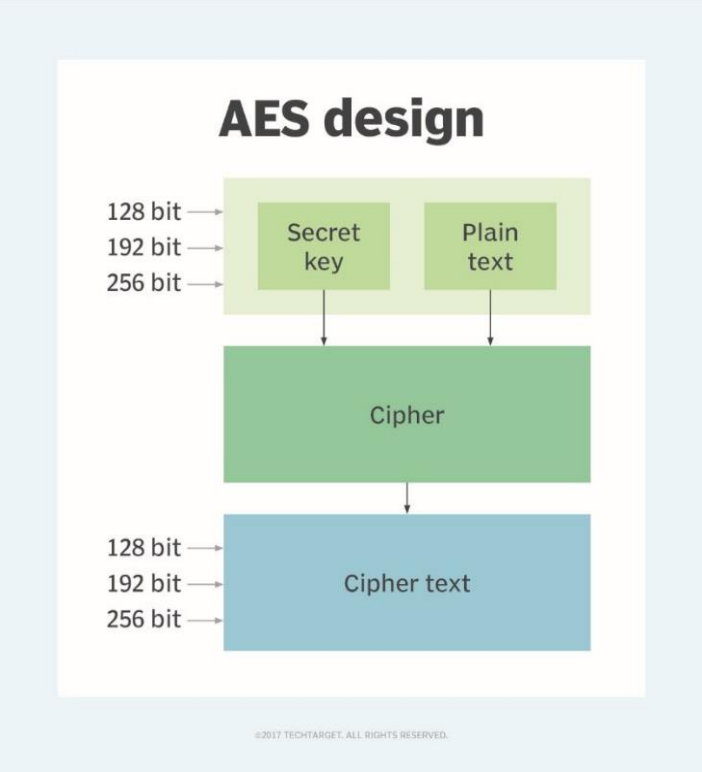


Figure 9.2. AES Design

9.4. Securing FOTA Server

Objective: Enhance server security for the Firmware Over-The-Air (FOTA) server by implementing robust authentication measures and conducting penetration testing to identify and mitigate common vulnerabilities, specifically targeting the OWASP Top 10.

1. Two-Factor Authentication (2FA):

- **Definition:** 2FA serves as an additional layer of security beyond the traditional username and password login process. It requires users to provide two different authentication factors to verify their identity.

- **Implementation:** Enforced on the FOTA server to significantly reduce the risk of unauthorized access.

- **Options for Second Factor:**
  - **One-Time Password (OTP) via Email:** Users receive a unique OTP via email for verification.
  - **One-Time Password (OTP) via SMS:** Users receive a unique OTP via SMS to their phone.
  - **One-Time Password (OTP) via Mobile Application:** Users receive a unique OTP via Mobile Application (user Interface of the vehicle).

- **Benefits of 2FA:**
  - **Enhanced Security:** 2FA adds an extra layer of security, making it harder for unauthorized users to gain access.
  - **Reduced Risk of Unauthorized Access:** Even if passwords are compromised, attackers would still need the second factor to log in.

2. Server Operation Cycle:

- **Admin Responsibilities:**
  - **Purpose:** Administrators manage member registrations, monitor updates, and review server logs.
  - **Authentication:** Administrators also log in with 2FA for security purposes.

- **Member Interaction:**
  - **Purpose:** Members upload and download firmware updates/upgrades.
  - **Authentication:** Members must log in with 2FA before accessing server functionalities.

3. Penetration Testing for OWASP Top 10 vulnerabilities:

Objective: Our goal is to ensure the security of our system by performing a penetration test. This test helps us identify any weak spots or vulnerabilities that hackers could exploit to gain unauthorized access. Once we uncover these weaknesses, we'll promptly address them to strengthen the security of our system.

Example for Vulnerabilities:

1. **SQL Injection (SQLi):** SQL injection occurs when an attacker injects malicious SQL queries into input fields or URLs, allowing them to manipulate the database and potentially access or modify sensitive data.
2. **Cross-Site Scripting (XSS):** Cross-site scripting vulnerabilities enable attackers to inject malicious scripts into web pages viewed by other users, leading to session hijacking, data theft, or unauthorized actions.

Benefits of Penetration Testing for OWASP Top 10:

- **Proactive Security**: Identifies and addresses common vulnerabilities before they can be exploited by attackers.
- **Cost Savings**: Identifies vulnerabilities early, reducing the potential impact and cost of addressing security incidents.

9.5. Future Cybersecurity Plans for Vehicle Safety

1- Autonomous Detection of Malicious Updates

One of the primary objectives of our project is to enable vehicles to autonomously detect malicious updates using artificial intelligence (AI). This innovation will significantly enhance vehicle safety by allowing them to analyze the behavior of incoming updates. The AI system will be trained to recognize patterns and anomalies that indicate potential threats. By integrating this capability, vehicles will be able to defend themselves against cyber threats independently, even in scenarios where manufacturers might be compromised. This self-sufficient defense mechanism reduces reliance on external servers and adds an extra layer of security.

2- Event Log Triggers for SOC Analysts

Another project currently under study aims to develop advanced trigger event logs for Security Operations Center (SOC) analysts. These logs will be meticulously designed to capture and record any suspicious activities within the vehicle's network. The detailed and structured nature of these logs will enable SOC analysts to conduct thorough investigations and respond more effectively to potential cyber attacks. By providing SOC analysts with comprehensive event logs, we can make it increasingly difficult for hackers to breach vehicle security systems. This proactive approach not

only aids in detecting and mitigating threats but also serves as a valuable resource for forensic analysis and future threat prevention.

Together, these two projects—autonomous detection of malicious updates and advanced trigger event logs—represent a significant advancement in vehicle cybersecurity. By empowering vehicles with AI-driven self-defense mechanisms and equipping SOC analysts with detailed event logs, we aim to create a robust and resilient security framework. These initiatives will ensure that our vehicles remain safe and secure in an increasingly complex and challenging cyber landscape.

10. UART Serial Communication Protocol

- **UART (Universal Asynchronous Receiver Transmitter)**

UART is a communication protocol used for transmitting and receiving serial data between electronic devices. It facilitates asynchronous serial communication, meaning data is sent without the need for a shared clock signal between the sender and receiver.

10.1. How it works?

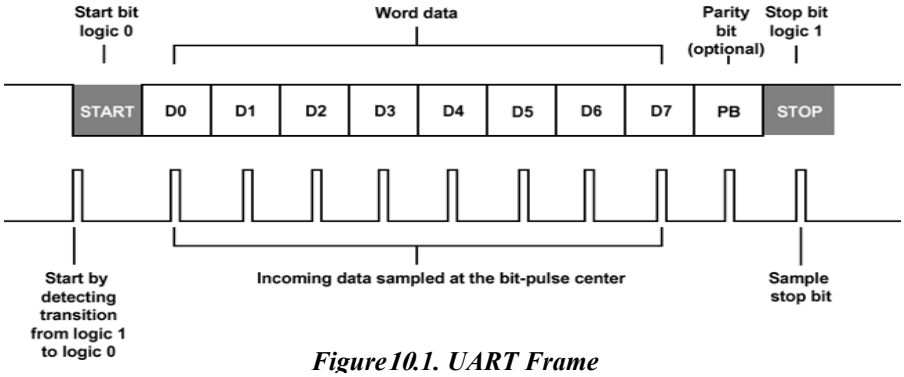


Figure 10.1. UART Frame

- **Data Transmission:** In UART communication, data is transmitted as a series of bits, typically 8 bits per byte. Each byte is preceded by a start bit and followed by one or more stop bits, providing synchronization between the sender and receiver.

- **Asynchronous Operation:** Unlike synchronous communication protocols such as SPI and I2C, UART operates asynchronously, meaning the timing of data transmission is not synchronized with a clock signal. Instead, both the sender and receiver agree on a predefined baud rate to establish communication.

- **Simplex Communication:** UART supports simplex communication, where data flows in only one direction at a time. However, it can be configured for full-duplex communication by using two UART channels, one for transmitting and one for receiving data.

- **Error Detection:** UART lacks built-in error-checking mechanisms. However, parity bits can be optionally added to each byte for basic error detection also we can use checksums, although they are less commonly used in modern implementations due to their limited effectiveness.

10.2. UART Frame Types

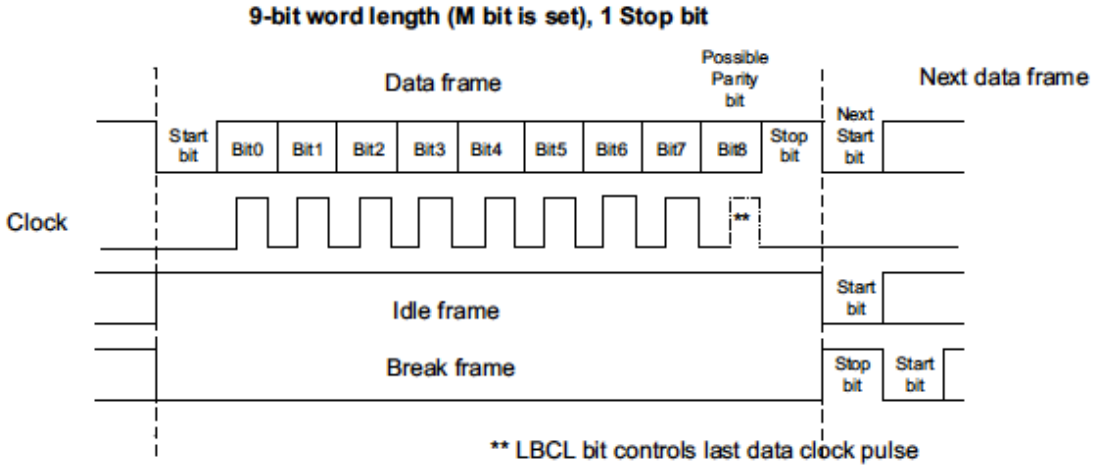


Figure10.2. UART Frame Types

- **Data frame:**
  Data frames are fundamental units of data transmission in various communication protocols, such as Ethernet or CAN bus. They carry payload information from the sender to the receiver.
  It contains the actual data to be transmitted, including headers, payload, and error-checking information.
- **Idle frame:**
  An idle frame, also known as an idle bus message, is a pseudo-random data frame transmitted when there are no active transmissions on the bus. It consists of pseudo-random data patterns generated by the transmitting device when there is no other data to transmit.

- **Break frame:**
  Break frames are special control frames used in certain communication protocols, such as UART (Universal Asynchronous Receiver-Transmitter). They signal the start or end of a transmission and can be used for synchronization. It consists of a predefined sequence of bits with specific timing characteristics that indicate the start or end of a transmission.

10.3. Why UART?

Using UART facilitates asynchronous communication between the Main ECU and the server, enabling the transmission and reception of data between them. The Main ECU serves as the central component of the system, directly interfacing with the server to check for updates and download them. This communication protocol ensures efficient and reliable data exchange, allowing the Main ECU to easily interact with the server to retrieve necessary information and perform required tasks, such as updating firmware or fetching data.

11. Testing and Validation

11.1. V-Model: Overview

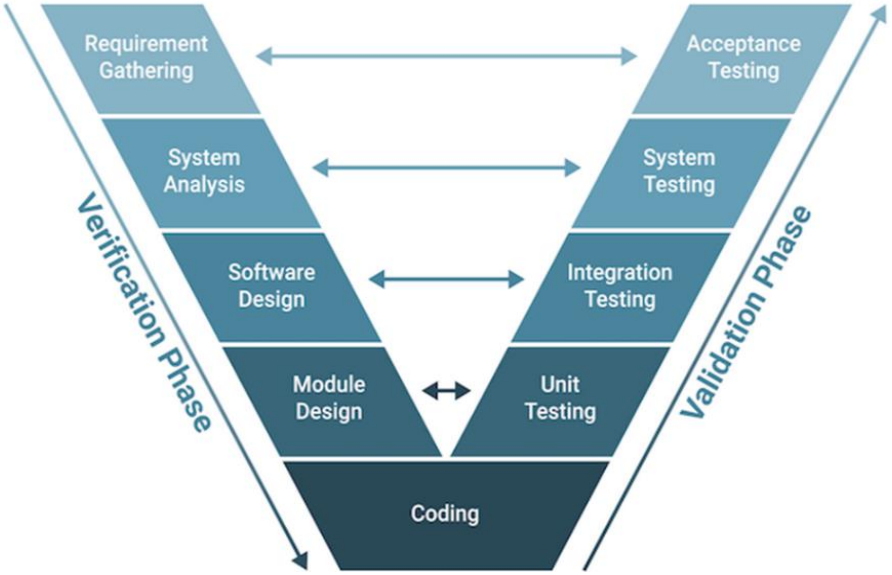


Figure 11.1. V-Model Testing

The V-Model is a well-established software and hardware development process model that emphasizes the importance of verification and validation at each development stage. It is particularly beneficial for embedded systems projects, ensuring that each phase of development is directly associated with its corresponding testing phase. This paper outlines the structure of the V-Model, its stages, and its application in embedded systems development.

In embedded systems development, ensuring the reliability and functionality of the product is crucial. The V-Model offers a structured approach that integrates development and testing activities, providing a clear path from requirements to final product validation.

11.2. Why V-Model?

It is easy to manage due to the strength of the model, as each phase of V-Model has specific deliverables and a review process. Due to proactive defect tracking – that is defects are found at an early stage. It provides a clear link between the requirements and the final product, making it easier to trace and manage changes to the software.

Structure of the V-Model

The V-Model is shaped like a "V," representing the sequential phases of development and corresponding testing activities. It consists of two main parts: the development (left side) and the testing/validation (right side).

11.3. Verification Phases

1. **Requirement Analysis:**
    - **Objective:** Gathering and analyzing system requirements.
    - **Through stage:** Identifying all necessary hardware components, software libraries, tools, and data required to implement the project.
    - **Outcome:** A comprehensive System Requirement Specification (SRS), detailing performance, security, availability, and interfacing requirements.
    - **Verification:** Conducting thorough requirements review to ensure all goals are met and obtain approval.
2. **System Analysis:**
    - **Objective:** Defining the system architecture and breaking it down into subsystems and modules.
    - **Through stage:** Decomposing the project into smaller subsystems, assign roles, and create detailed project scenarios.
    - **Outcome:** Detailed definitions of subsystems and their respective roles.
    - **Verification:** Performing an architecture review and validating scenarios.
3. **Software Design:**
    - **Objective:** Detail the software design of each module.
    - **Through stage:** Develop the server and mobile applications in parallel.
    - **Outcome:** Functional server and mobile applications.
    - **Verification:** Conduct a software review to ensure all features are implemented correctly.
4. **Module Design:**

- **Objective:** Detail the design of Hardware components.
- **Through stage:** Gathering and preparing all necessary hardware components for implementation.
- **Outcome:** Obstacles Avoidance Vehicle Model.
- **Verification:** Review the detailed hardware design.

5. **Implementation:**

- **Objective:** Developing the actual code of hardware components.
- **Through stage:** Developing and implementing the code for each hardware component and creating a bootloader for software updates.
- **Outcome:** Complete source code for the project.
- **Verification:** Performing code reviews and unit testing.

11.4. Testing and Validation Phases

Each development phase has a corresponding testing phase to ensure thorough verification and validation.

1. **Unit Testing:**
   - **Objective:** Verifying the functionality of individual components.
   - **Corresponding Phase:** Module Design.
   - **Method:** Test individual code sections independently.
2. **Integration Testing:**
   - **Objective:** Verifying the interaction between integrated units.
   - **Corresponding Phase:** Software Design.
   - **Method:** Test interactions with both the UI and the server.
3. **System Testing:**
   - **Objective:** Verifying the complete and integrated system.
   - **Corresponding Phase:** System Analysis.
   - **Method:** Test the entire system to ensure it meets all requirements.
4. **Acceptance Testing:**
   - **Objective:** Validating the system against our requirements.
   - **Corresponding Phase:** Requirement Analysis.
   - **Method:** Test the system in the real-world environment to ensure it meets user expectations.

11.5. Benefits of the V-Model

- **Early Detection of Defects:** Each development phase is verified through corresponding tests, enabling early identification of issues.
- **Structured and Systematic Approach:** Provides a clear and organized framework for development and testing.
- **Improved Documentation:** Ensures comprehensive documentation at each stage, aiding in project management and future maintenance.

In embedded systems development, the V-Model ensures that both software and hardware components are developed and tested systematically. For instance:

- **During System Design**, the overall architecture, including microcontrollers, sensors, and software libraries, is defined.
- **Software Design** involves detailed design of subsystems like communication interfaces, and power management.
- **Module Design** focuses on assembling hardware components.
- **Implementation** covers implementing the source code.

Each stage is followed by rigorous testing, from unit testing individual modules to system testing the complete embedded system, ensuring the final product is robust and reliable.

12. References

1- Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., ... & Koscher, K. (2011). Comprehensive Experimental Analyses of Automotive Attack Surfaces. USENIX Security Symposium.
Link: https://www.usenix.org/legacy/event/sec11/tech/full_papers/Checkoway.pdf

2- Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., ... & Savage, S. (2010). Experimental Security Analysis of a Modern Automobile. IEEE Symposium on Security and Privacy.
Link: https://ieeexplore.ieee.org/document/5504804

3- Petit, J., & Shladover, S. E. (2015). Potential Cyberattacks on Automated Vehicles. IEEE Transactions on Intelligent Transportation Systems, 16(2), 546-556.
Link: https://ieeexplore.ieee.org/document/6971141

4- SAE International. (2021). J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems. SAE International.
Link: https://www.sae.org/standards/content/j3061_201601/

5- Tesla. (2021). Tesla Software Updates. [Online] Available at: https://www.tesla.com/support/software-updates

6- Park, J., Lee, J., Jeong, H., & Lee, J. (2018). Efficient Firmware Update Scheme for Smart Car using Over-the-Air (OTA). Sensors, 18(7), 2275.
Link: https://www.mdpi.com/1424-8220/18/7/2275

7- Rieke, R., Seifert, J. P., & Heinz, A. (2016). Security and Safety Requirements Engineering for Automotive Software Development. In Proceedings of the 5th ACM SIGSAC Conference on Computer and Communications Security.
Link: https://dl.acm.org/doi/10.1145/2976749.2978364

8- Kumar, K., & Lee, H. J. (2014). Security Issues in Healthcare Applications Using Wireless Medical Sensor Networks: A Survey. Sensors, 12(1), 55-91.
Link: https://www.mdpi.com/1424-8220/12/1/55

9- Pease, J., Howitt, I., & Fischmeister, S. (2017). Secure Boot for Embedded Automotive Platforms. In Proceedings of the ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS).
Link: https://ieeexplore.ieee.org/document/7936537

10- DiMase, D., Collier, Z. A., Heffner, K., & Linkov, I. (2015). Systems Engineering Framework for Cyber Physical Security and Resilience. Environment Systems and Decisions, 35(2), 291-300.
Link: https://link.springer.com/article/10.1007/s10669-015-9554-6

11- Wang, H., Huang, X., & Xie, L. (2015). Design and Implementation of UART Communication Based on FPGA. International Journal of Hybrid Information Technology, 8(5), 243-252.
Link: http://article.nadiapub.com/IJHIT/vol8_no5/19.pdf

12- Abeyratne, T., Shen, C. C., & Daga, A. (2018). Implementation and Performance Evaluation of a Hardware-based UART Protocol Analyzer. In Proceedings of the 2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC). Link: https://ieeexplore.ieee.org/document/8409711

13- Forsberg, K., & Mooz, H. (1991). The Relationship of System Engineering to the Project Cycle. In Proceedings of the First Annual Symposium of National Council on System Engineering. Link: https://www.sebokwiki.org/wiki/Vee_Diagram