Firmware over the air

By

Ahmed Sayed, Salma Ahmed, Salma Tarek,

Sondos Aly, Mario Medhat, Michael Adel

A Thesis Submitted to the

Faculty of Engineering at Cairo University in Partial Fulfilment of the Requirements for the Degree of

Bachelor of Science

In

Electronics and Communications Engineering

Under the Supervision of

Prof. Hassan Mostafa

Prof. Samah Tantawy

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT 2021

Acknowledgment

Above all, praise to **ALLAH.** By whose grace this work has been completed and never leaving us during this stage.

We would like to express our sincere gratitude to **Dr. Hassan Mostafa,** Prof. of Electronics, Faculty of Engineering, Cairo University, for his academic supervision and encouragement through the year.

Our deep thanks to **Dr. Samah Tantawy**, Assistant Prof. of Mathematics, Engineering Mathematics and Physics, Faculty of Engineering, Cairo University, for supervising the project and the technical and emotional support during the year.

Special Thanks to **Eng. Abdelrahman Sobeih,** Research Assistant at Onelab, for his guidance, supervision and encouragement through the year.

We would like to thank **Eng. Bishoy Wasfey,** Expert at Valeo Egypt, for his mentorship, guidance and support in the project.

We would also like to deeply thank **Eng. Ahmed Assaf**, General Manager at IMT School, for suggesting the idea of the project, mentorship, guidance and support during the year.

We specifically thank **Academy of Scientific Research and Technology (ASRT)** for all the financial support of the project.

Table of Contents

List of figures

List of Tables

Abstract

Nowadays, any automotive vehicle has a very large number of Electronic Control Units (ECUs), it may have around 100 ECUs and approximately 100 million lines for software coding. These numbers are increasing rapidly since connected cars have been introduced.

It’s suggested that by 2020, there will be 300 ECUs in a single car, to manage most of the functions and features within a car. After the vehicle's release, OEMs usually need to fix bugs, add new features, or update the ECUs’ software. This requires the customer to visit the maintenance center periodically. Recalling the vehicles to do that will not be considered as a good option for OEMs as it negatively affects their reputation, user experience, unnecessary cost and time.

We aim to update the software of a car by flashing the code on the ECUs over the air without using any physical connection, FOTA enables cars to be upgraded remotely, without having the user to worry about a software-related recall or update.

Our solution fulfill security, proper timing and also solving real life problems which impact cost reduction for OEMs and user satisfaction and make our life easier.

Chapter 1: FOTA Introduction

1.1. History of FOTA

Firmware Over-The-Air (FOTA) is a Mobile Software Management (MSM) technology in which a mobile device's operating firmware is upgraded and updated wirelessly by its manufacturer. FOTA-capable phones directly download the updates from the service provider. Depending on link speed and file size the process typically takes three to 10 minutes.

FOTA took 4 management stages process:

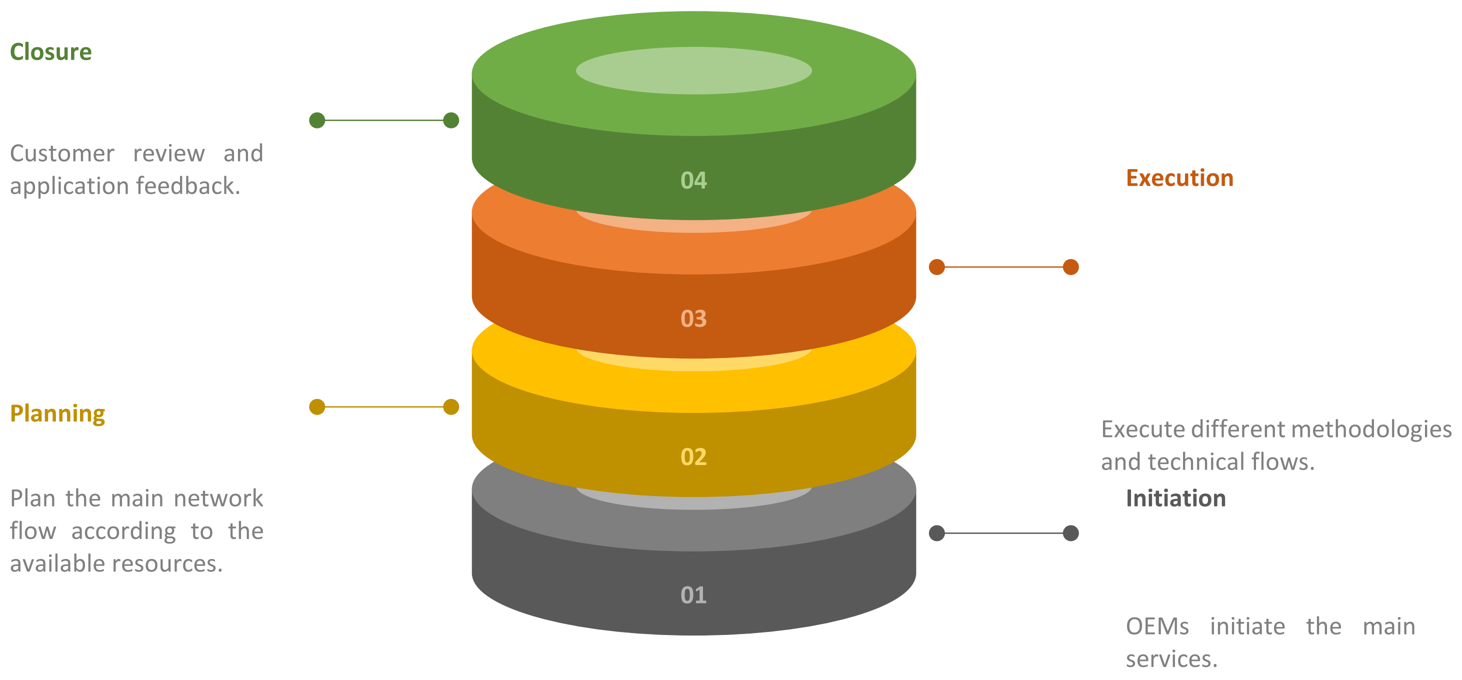

Closure
Customer review and application feedback.
04
Execution
03
Planning
02
Execute different methodologies and technical flows.
Initiation
Plan the main network flow according to the available resources.
01
OEMs initiate the main services.

Figure 1: FOTA management process.

Through these stages FOTA shall facilitate:

- Enables fabricators to patch glitches in new systems.
- Allows OEMs to be able to send and install new software updates and features remotely after customers have bought the cars.

FOTA represent a tremendous opportunity for automotive manufacturers, with a clear route to cost savings, particularly during vehicle warranty periods; revenue gains from the sales of new features and services, and a more engaged relationship with clients.

So, the management process took its time to bring out the whole fully experience to the customer and the OEMs, the imperative to move towards FOTA stems from the autonomous, connected, and electric vehicles trend in the automotive sector. As these vehicles are using increasingly complex hardware and operating systems, vendors would need a mechanism to repair, manage and enhance any aspect of vehicle efficiency.

In order to achieve that it was about the first steps toward any idea a good strategy to overcome the complexity of such application by dividing the process within different structures.

Next, we will summarize the historical keys of development.

1.2. Problem Definition

Today, an automotive vehicle has on average 100 **Electronic Control Units** (ECU) and about 100 million lines of software code. At an average cost of $10 per line, these electronic systems aren't cheap and recalls lead to multi-billion-dollar losses and also leave a dent on manufacturers' reputations. This situation will be worse as this number is growing rapidly such that experts suggest that there will be 300 ECUs in a car in 2020 so that, OEMs must reduce the impact of product recalls by managing the software efficiently over the lifecycle of the vehicle.
So, there are several constrains to update the cars' software:

1. **Time:** upgrading the firmware could take many days.
2. **Complexity:** upgrading the firmware must be completely handled at the application layer.
3. **Cost:** to upgrade the firmware of the car, people must go to maintenance centers.

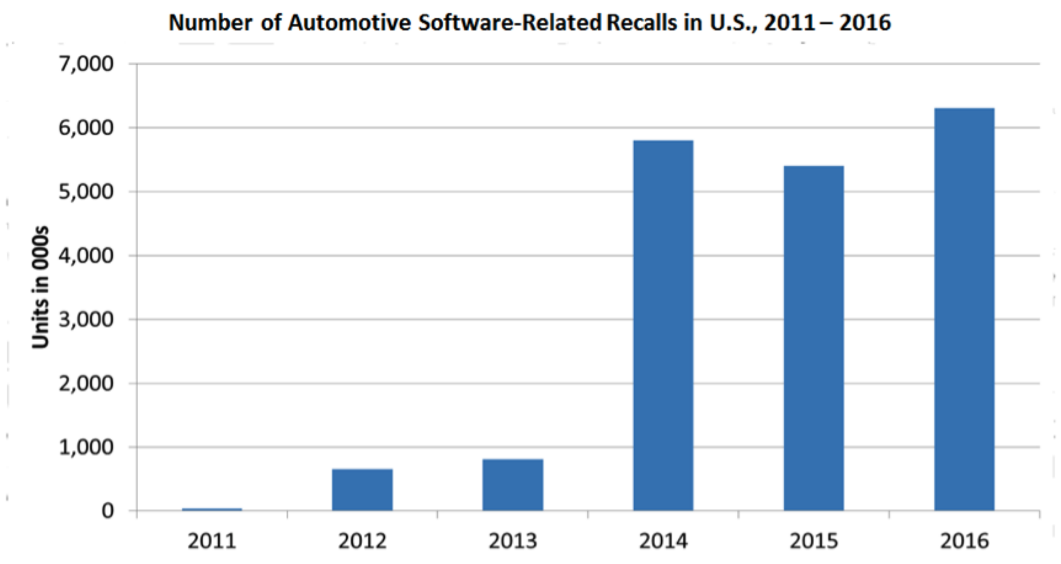


Figure 2: Automotive software recalls

1.3. Key developments timeline

In the automobile industry, in September 2012, Tesla delivered the first FOTA software update for their 'Model S' vehicles. The business published an update and use either car's integrated 3 G network link or a Wi-Fi signal from the customer's home network.

Other automotive OEMs such as General Motors, BMW, Volvo, Detroit Diesel Organization and Mercedes-Benz have begun to deliver updates to OTA from 2017-2018. Ford also plans to deliver updates to OTA by 2020. Agricultural machinery manufacturers including John Deere, AGCO Company, and CNH Industrial have begun providing simple OTA updates for their farm machinery along with automotive OEMs.

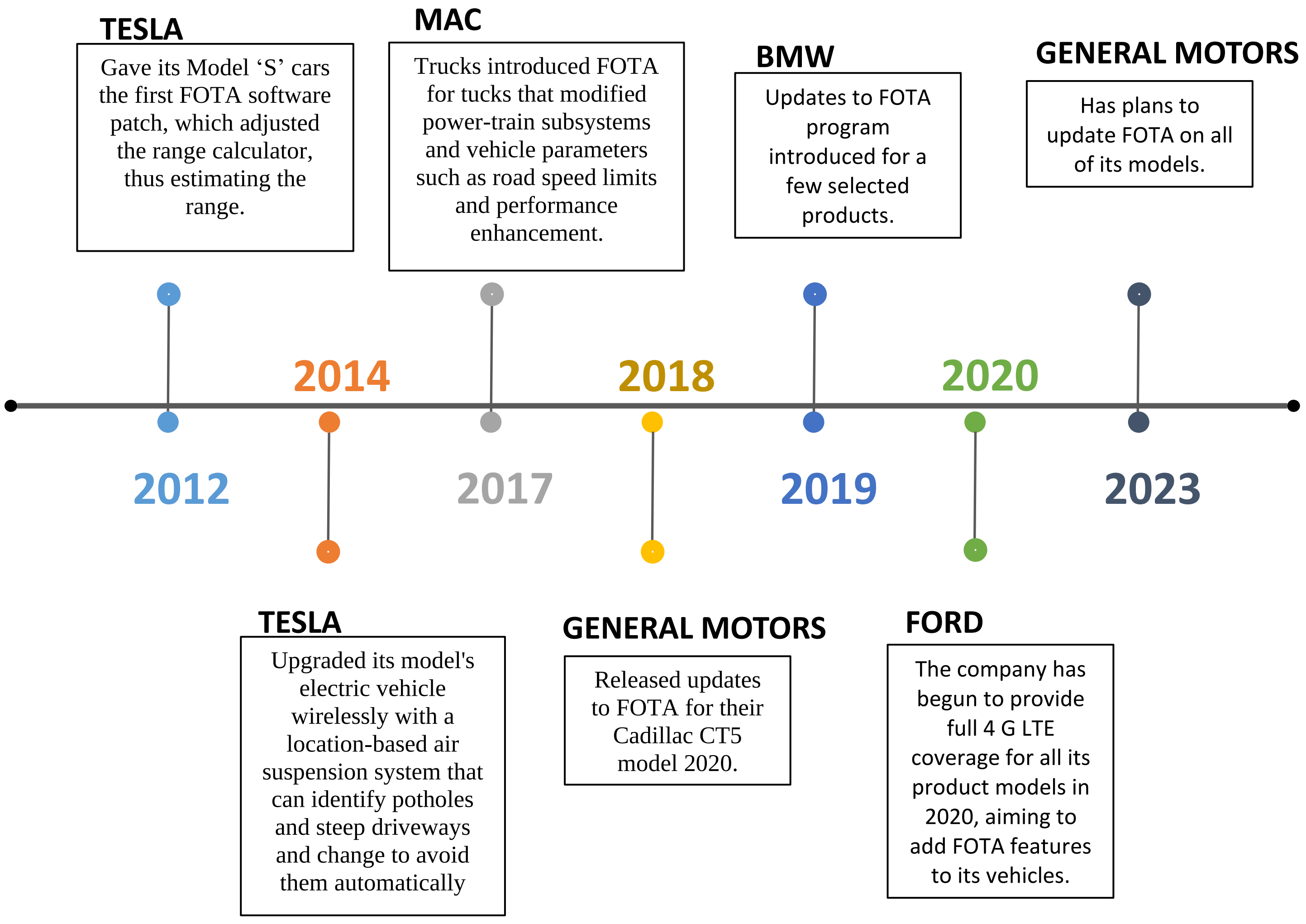

TESLA
Gave its Model 'S' cars the first FOTA software patch, which adjusted the range calculator, thus estimating the range.
MAC
Trucks introduced FOTA for tucks that modified power-train subsystems and vehicle parameters such as road speed limits and performance enhancement.
BMW
Updates to FOTA program introduced for a few selected products.
GENERAL MOTORS
Has plans to update FOTA on all of its models.
2014
2018
2020
2012
2017
2019
2023
TESLA
Upgraded its model's electric vehicle wirelessly with a location-based air suspension system that can identify potholes and steep driveways and change to avoid them automatically
GENERAL MOTORS
Released updates to FOTA for their Cadillac CT5 model 2020.
FORD
The company has begun to provide full 4 G LTE coverage for all its product models in 2020, aiming to add FOTA features to its vehicles.

Figure 3: Key developments timeline.

In the automotive industry, FOTA updates go hand in hand with self-driving and connected vehicles. IOT updates will be an important part of the near future, according to industry analysts, by 2030, 98 percent of cars sold worldwide are projected to be connected vehicles. The value produced by next-generation cars (in terms of the cost of the vehicle) will be totally different than the current ones; the software will be a key selling point in next-generation cars. To sell their vehicles in the near future, automotive OEMs will deliver robust software along with hardware systems. [1]

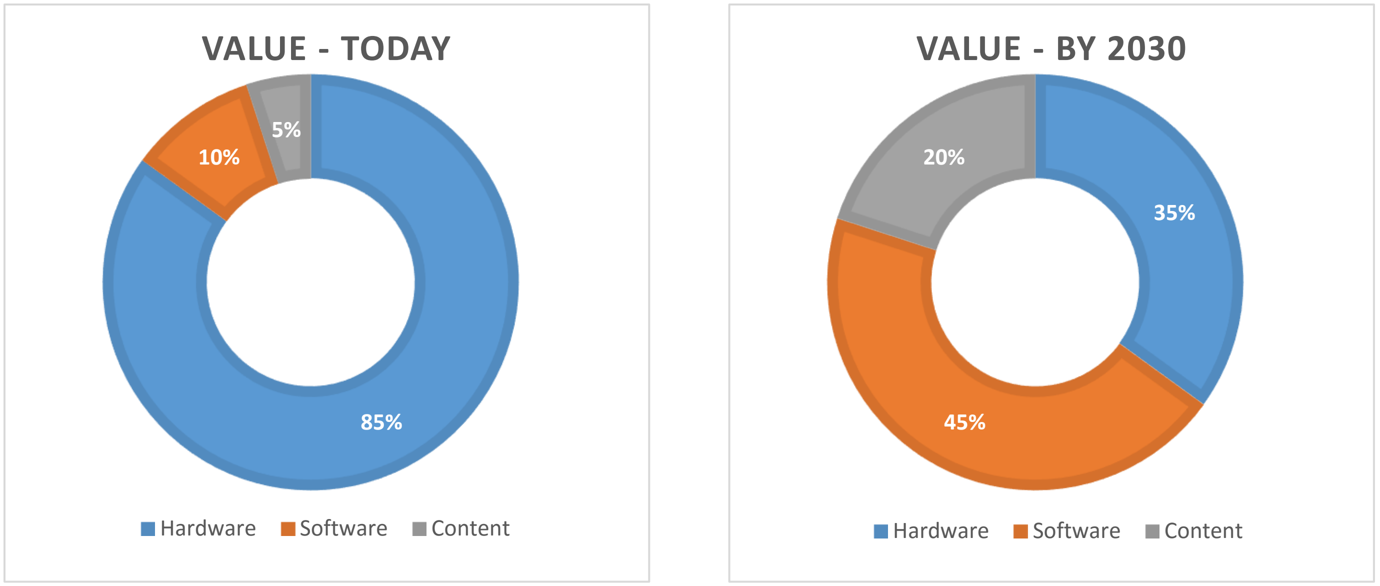


Figure 4: Automotive value

Apparently, industry forecasts say that FOTA 's future in automotive is looking promising. This is mainly relevant to the automotive industry which is shifting its gear towards software-driven autonomous systems development.

The Research and Economies reported that the Automotive Over the Air (FOTA) updates market is expected to expand at a CAGR of 58.15 per cent over the period 2018-2022.And many of the largest automobile players are already on the path of making FOTA in automotive systems seamless and stable upgrade, a fact.

1.4. Key players and market

The over-the-air update is a means of delivering new applications, firmware, and other cloud-based upgrading facilities. The FOTA updates were only available for smartphones until recent time; however, with the significant increase in the number of connected cars around the globe and vehicles becoming more software-dependent, the need for FOTA updates has increased significantly. Software upgrades have long been achieved by local software upgrade procedures, whereby the car is sent to a vendor / mechanic who upgrades the software afterwards.

Tesla Motors has started to deliver FOTA capability for its electric cars, which has shaken the automotive industry. Other global automakers, such as BMW, Mercedes-Benz and General Motors, then started investing in providing FOTA capabilities for their connected vehicles. FOTA updates are supposed to help OEMs save a large sum they spend as a result of car recalls. It also gives car owners the ease of updating all their software without having to visit the location of the dealer.

Robert Bosch GmbH, NXP Semiconductors N.V, Verizon Communications, Inc., Continental AG, Infineon Technologies AG, Qualcomm Incorporated, Intel Corporation, HARMAN International, Airbiquity Inc, Aptiv, HERE Technologies, BlackBerry QNX Software Systems Limited, Garmin Ltd. and Intellias Ltd are some of the major players on the market.

The global FOTA automotive updates market is characterized by the presence of numerous national, regional, and local suppliers. The market is highly competitive, competing with all players to achieve full market share.

According to the report of Market Research Future (MRFR), as of 2018, HARMAN International, Aptiv, QNX Computer Systems Limited, Intel Corporation and Airbiquity Inc. are some of the major players on the global FOTA automotive update market. By growth, mergers & acquisitions, and by providing a broad product range, these businesses continue to maintain their strong global grip.

Another main announcement in the market that Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, previously confirmed that its 5G test networks would be extended to include new end-to - end over-the-air (FOTA) systems for both (mmWave) and sub-6 GHz bands.

While Verizon Introduces Hum, an all-in-one connected car solution that provides users with vehicle position, diagnostics, on-road assistance, travel history and more. It's the first 4G LTE connected car solution to have Google Assistant built-in. [2]

Figure 5: HUM - device

Chapter 2: Project Overview

2.1. Introduction

In this chapter, a quick overview of the project features and flow will be explained in detail, in addition to the block diagram of the system. The solution consists of two main features: Software Updates and Live Diagnostics. The flow of each of them will be discussed in detail.

2.2. Block Diagram

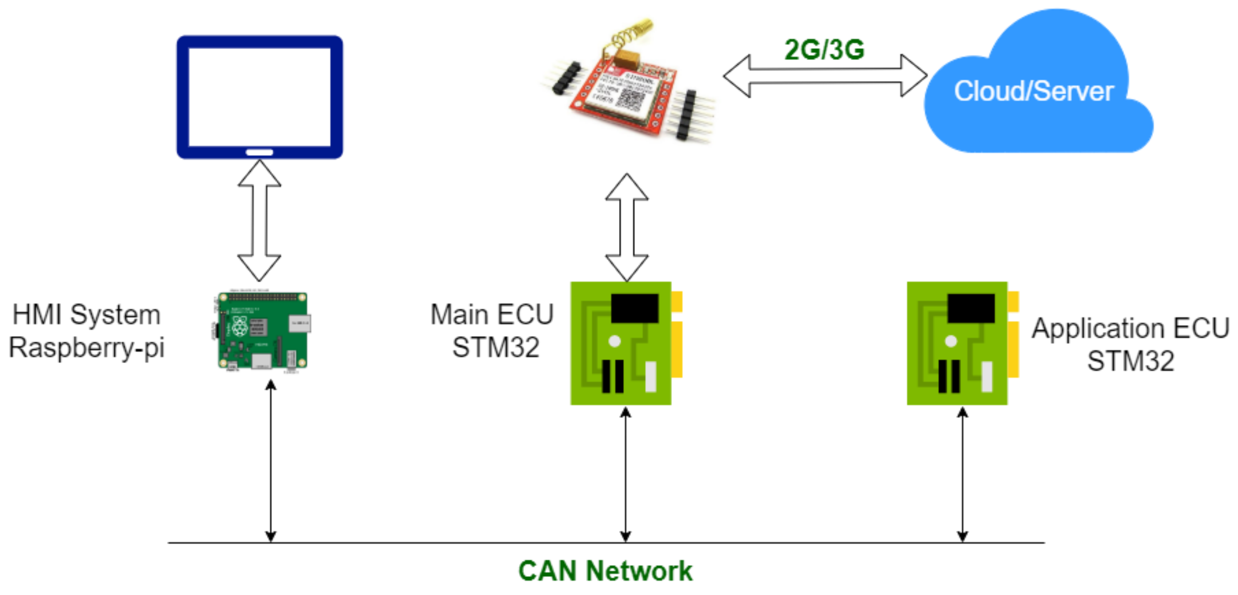


Figure 6: System Block Diagram

As shown in figure 6, the system consists of 3 main controllers:

1- **Main ECU:** It is connected to a SIM800L GSM Module and acts as the gateway to the server.
2- **Application ECU:** It acts as a faulty node whose software needs to updates or have a feature added to it.
3- **HMI System:** It consists of Raspberry-pi Model 3B and an HDMI Touch Screen. It acts as an interface to the user.

The three ECUs of the system are connected through a CAN Network.

2.3. Software Updates

The software updates project flow is as shown in Figure 7.

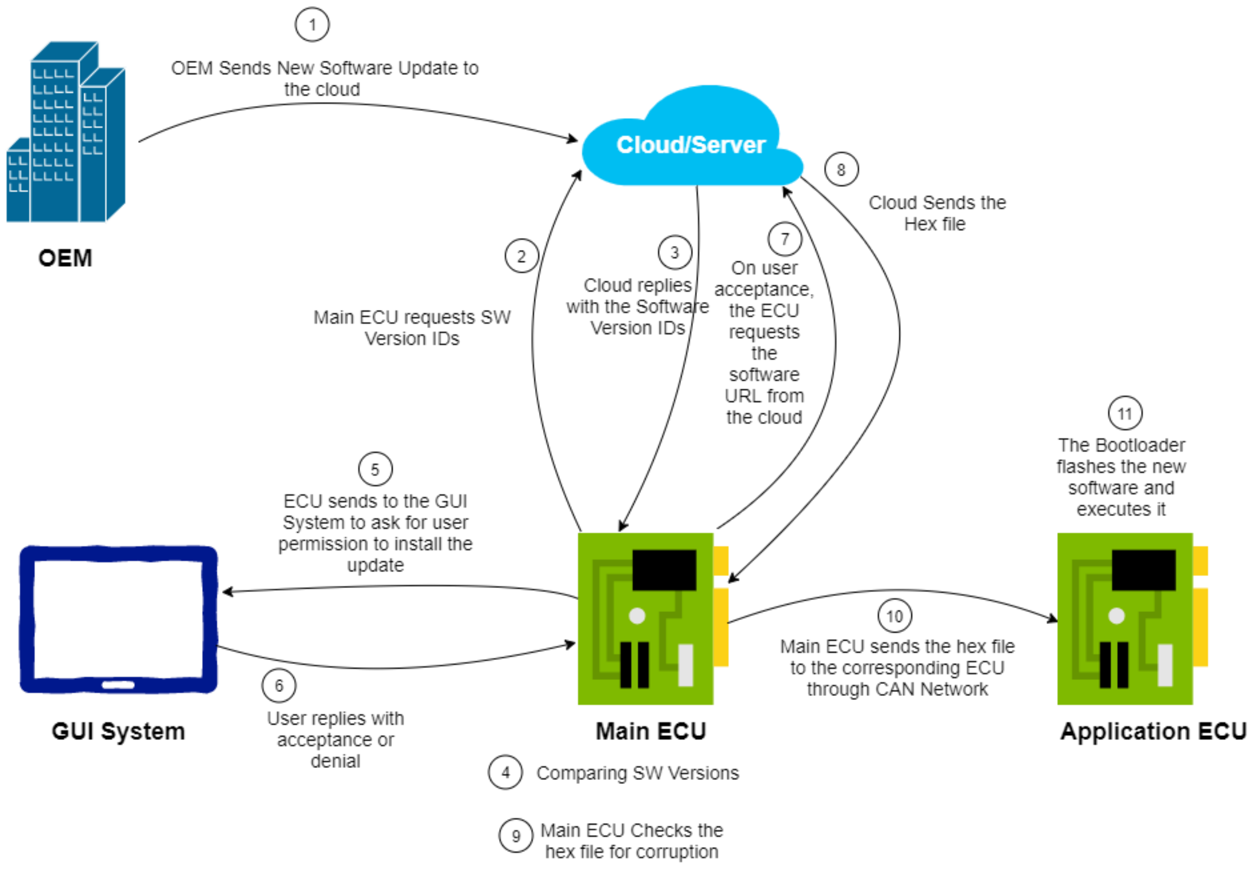

1
OEM Sends New Software Update to the cloud
Cloud/Server
OEM
2
Main ECU requests SW Version IDs
3
Cloud replies with the Software Version IDs
7
On user acceptance, the ECU requests the software URL from the cloud
8
Cloud Sends the Hex file
11
The Bootloader flashes the new software and executes it
5
ECU sends to the GUI System to ask for user permission to install the update
10
Main ECU sends the hex file to the corresponding ECU through CAN Network
GUI System
6
User replies with acceptance or denial
Main ECU
Application ECU
4
Comparing SW Versions
9
Main ECU Checks the hex file for corruption

Figure 7: Software Updates Project Flow

1- The OEM uploads a new software to the server.
2- On the next ignition cycle in the vehicle, the main ECU requests the software versions IDs from the server, it also requests the software version IDs from the application ECU.
3- The server replies with them in a sematic version format.
4- The main ECU compares the versions.
5- If an update is found it sends a CAN Message to the GUI requesting for the user acceptance for the update. In case no update is found, nothing happens.
6- Once the user accepts or rejects the update, the GUI sends a CAN Message in both cases.

7- In case of user acceptance, the main ECU requests the new hex file from the server and sends an update request message through CAN to the application ECU to perform a soft reset and start executing the bootloader.
8- The server sends the hex file to the Main ECU.
9- The Main ECU validates the file for errors.
10- The Main ECU sends the hex file through CAN Network to the application ECU, it also sends the update progress to the GUI to display it.
11- The bootloader receives the file and checks for any error, flashes it in the flash memory then jumps to the new application.

2.4. Live Diagnostics

Since a connectivity is already established in the system, a live diagnostics feature is implemented to inform the OEM in case of failure or error in the system. The flow is as shown in figure 8.

1- The user asks to send the diagnostics through the GUI.
2- The GUI sends a CAN Message to the main and application ECUs requesting for a diagnostics session.
3- The application ECU sends its DTCs through the CAN Bus.
4- The Main ECU checks if it is currently with the GSM performing an update, if not, it sends the DTCs to the server.
5- The server receives the DTCs and displays them on the server terminal.

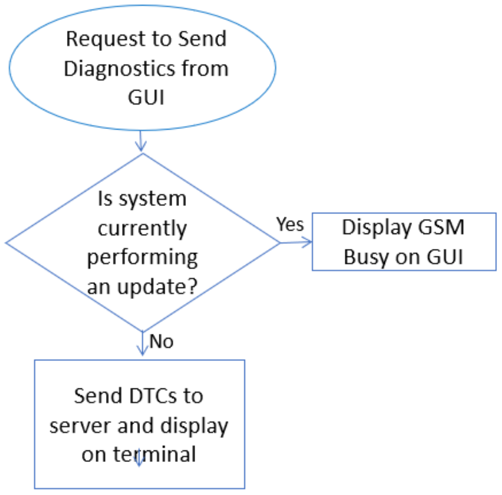

Request to Send Diagnostics from GUI
Is system currently performing an update?
Yes
Display GSM Busy on GUI
No
Send DTCs to server and display on terminal

Figure 8: Live Diagnostics Flow

Chapter 3: Cloud

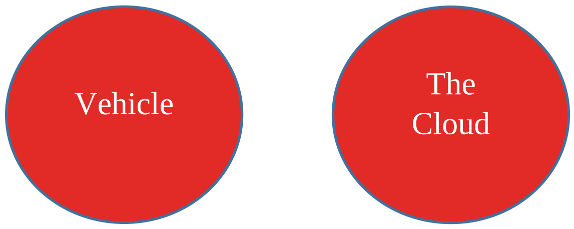
3.1. Introduction

“IoT is a tool or a method. The essential quality of an IoT system lies in how much you can innovate by effectively using this tool.”

Internet of Things (IoT) is a wide set of technologies and use cases that no clear definition would wrap it. In our case, we may define it as the use of network-connected vehicles, to communicate with the OEM’s servers.

Because of outstanding opportunities IoT promises, OEMs seek for the inclusion of it into their business models. However, when it comes to reality, this brilliant idea appears too complicated to be implemented. Given the number of vehicles, conditions needed to make it work, and the security of such model. In other words, the problem of establishing a reliable architecture of Internet of Things inevitably enters the stage.

3.1.1. Top-level components



In order to establish the connection between the cloud and the vehicle, some metadata must be planned first. Each vehicle can provide or consume various types of information. Each form of information might best be handled by a different backend system, and each system should be specialized around the data rate, volume, and preferred API.

For the vehicle, it will have some factory settings in its ROM. Such settings can be: vehicle identification code, component identification code, API end-points, secret keys used to exchange secure data with the server, protocols used, etc.

These settings should remain constant for as long as we could, and therefore a very strict considerations must be taken into place.

For the cloud, a good cloud service provider (AWS, Google cloud services, Azure) should be selected, with almost 100% up-time, high security measures, and high-end hardware to handle the requests as efficiently as possible.

3.2. Overview

3.2.1. Technology Stack

First component of the cloud is to choose the stack. We decided to work with MEAN stack. MEAN stands for MongoDB, Express, Angular, NodeJS. MongoDB is a cross-platform document-oriented database program. Express is a minimal and flexible NodeJS web application framework that provides a robust set of features for web and mobile applications. It has been called the de facto standard server framework for NodeJS. Angular is open-source web application framework led by the Angular Team at Google that we used to build the UI for the OEM. NodeJS is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside a web browser built on Chrome's V8 JavaScript engine. So, basically MongoDB is our database, Express is the server, Angular is the UI framework, and NodeJS is the programming language.

3.2.2. Why MEAN?

3.2.2.1. JavaScript Everywhere

One of the prominent reasons to choose MEAN stack over other technologies is that it makes use of the same language for frontend and backend development. Here all the technologies are written in JavaScript, which makes it possible to have neat codes during development. The server-side coding is taken care by Node.js, which brings JavaScript to the backend while the client-side coding is taken care by AngularJS, which makes it possible for the developers to reuse the codes from backend to frontend.

3.2.2.2. Market value

For the companies, they don’t have to go behind hiring specialists to take care of the different needs of the project. Simply hire MEAN stack developer and they are done. MEAN stack has just come up with a new profile which is nothing but full-stack JavaScript developer. Here companies can have a team of JavaScript developers so as to have better team management and collaboration for the project. And that will add to our skill-set for the current software market.

3.2.2.3. NodeJS

You will be able to have easier deployment as you will have web server included in the app. In order to process HTTP requests, NodeJS operates on a single thread. To handle multiple incoming requests efficiently, it makes use of non-blocking I/O calls. NodeJS is very much fast and scalable compared with other web servers like Apache and it supports a large number of concurrent connections. Without having the client to request data, NodeJS sends it by enabling web sockets.

3.2.2.4. AngularJS

It is used to build single-page applications. You can easily add AJAX-driven rich components and interactive functions on the client-side using Angular.js. Since you have a server-side solution provided by NodeJS, both server and client-side have JavaScript implementation. This way, MEAN applications becomes quite effective.

3.2.2.5. JSON

For data-interchange across different layers MEAN stack makes use of JSON format. So during the client-side and server-side interaction, the data can be converted without using any libraries. Moreover, with JSON, you can easily work with external APIs.

3.2.2.6. MongoDB

When you have large volumes of data on huge tables like the vehicles with different software for each car component, the OEM teams’ info and developers, API statistics, etc. then for managing them, you need the right database system like MongoDB. Adding a field in MongoDB is easier compared to any other database as there is no need to update a complete table. Moreover, it comes as a document model database, which makes it easier to be used across different applications. Just like the objects in the object-oriented programming language, MongoDB works.

3.2.2.7. Free and open source

All the technologies which you use MEAN stack is open source in nature and free to use. So, if any developments or updates take place in this area, then you can get the benefit of the same as it has the support of a large community.

3.2.3. Cloud service

For OEMs this would be a very crucial choice as they will need 100% up-time and flexible architecture with the cost in mind.

We started by choosing Heroku, it has easy deployment, it’s 100% free and had MEAN stack deploy option. We saw a lot of drawbacks that we had to look for another service, like we can’t SSH into the server, you can't create multiple HTTP network services without creating separate apps, it had hard caching that it was very annoying to test with the vehicle.

Second choice was Google cloud services. College students has a limited free option. It’s very flexible to choose the machine with specific requirements (CPU, RAM, Storage, location, etc.). And it didn’t have the drawbacks of Heroku.

3.3. Architecture

Some components need to be designed before starting to implement the project. And figuring out those components needs some planning itself.

We saw some here:

1. The OEM’s UI and API.

2. The vehicle’s API.

3. File system structure

4. Database model

3.3.1. Server UI Screens

Planning all the screens that should be shown on the UI. Each point will be displayed with a screenshot for the final result.

1. **Landing page:**

   Homepage for all the users and visitors where they see a short overview of what the platform has to offer.

2. **Teams page:**

   Overview of all the teams and descriptions of their role, as well as option to add a new team and delete a current one.

3. **Team page:**

   For each specific team to see each team member, their contribution, and the ability to add or delete a team member, along with firmwares linked to this team.

4. **Console:**

   We wanted a way to see the vehicles requests in real-time as the server receives them. This page will have a console-like screen of events and ASI calls in real-time.

5. **Statistics:**

To see the actual benefit of FOTA, and the efficiency of software releases of teams; A statistics screen was made to see how many requests are made, their type, and failure rate.

6. **Upload page:**

It has all the info needed for firmware upload and options to add new firmware from scratch or completely delete a current one.

3.3.2. File system structure

A very common problem with MEAN stack projects is the poor file system design at the start of the project. Usually, developers don't anticipate the size or the features they will later use.

We used the three-folder approach for all the API.

1. routes: the API endpoints that we have (e.g. /firmware/get/:firmwareID)
2. Models: The MongoDB database model that's used for specific route
3. Controllers: Where the actual logic and methods of the endpoints residue.

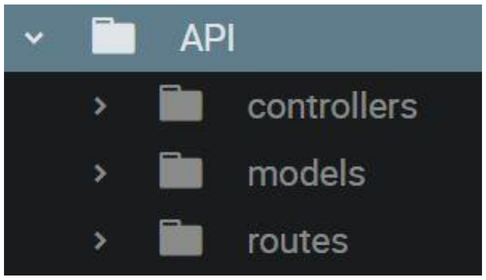


Figure 9: API file structure.

Finding a suitable folder structure for Angular projects is something that's always hard to get from the start - especially when the application grows in size – you end up continuously adding new features with no real structure in place. This makes it hard to locate files and making additional changes to the folder structure becomes time-consuming.

The CoreModule takes on the role of the root AppModule , but is not the module which gets bootstrapped by Angular at run-time. The CoreModule should contain singleton services (which is usually the case), universal components and other features where there's only once instance per application. To prevent re-importing the core module elsewhere, you should also add a guard for it in the core module' constructor.

The authentication folder simply handles the authentication-cycle of the user (from login to logout).

The footer- and header folders contains the global component-files, statically used across the entire application. These files will appear on every page in the application.

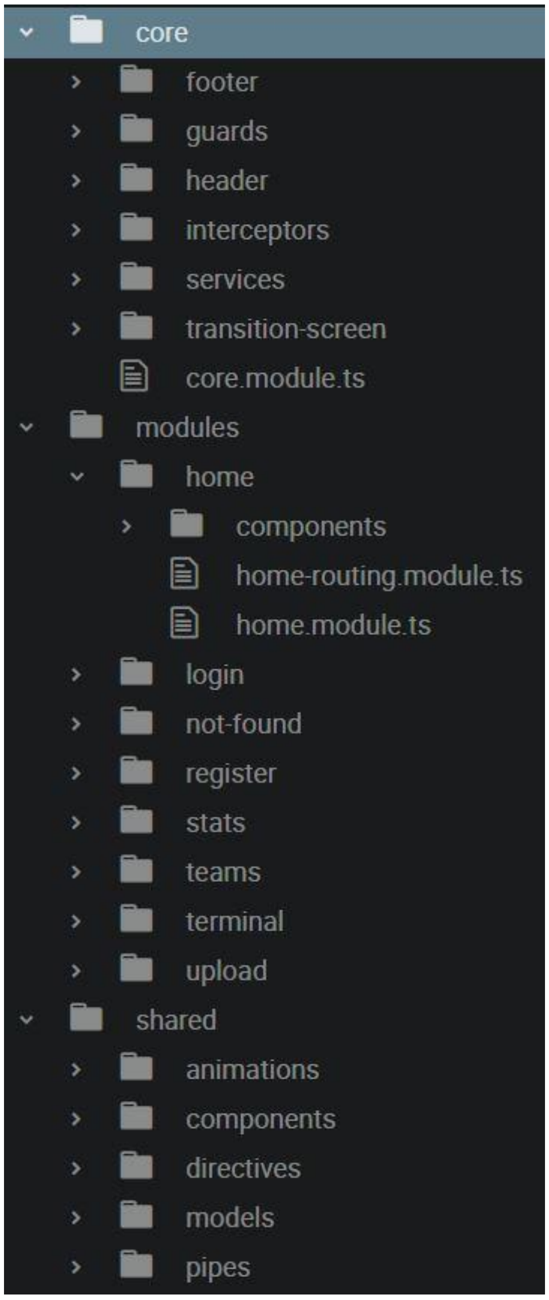
The **http** folder handles stuff like http calls from our application. I've also added a api.service.ts file to keep all http calls running through our application in one single place. The folder does otherwise contain folders for interacting with the different API-routes.

The **guards** folder contains all of the guards I use to protect different routes in my applications.

All additional singleton services are placed in the **services** folder.

The **SharedModule** is where any shared components, pipes/filters and services should go. The SharedModule can be imported in any other module when those items will be re-used. The shared module shouldn't have any dependency to the rest of the application and should therefore not rely on any other module.

The components folder contains all the "shared" components. This are components like **loaders** and **buttons**, which multiple components would benefit from.



```
core
    footer
    guards
    header
    interceptors
    services
    transition-screen
    core.module.ts
modules
    home
        components
        home-routing.module.ts
        home.module.ts
    login
    not-found
    register
    stats
    teams
    terminal
    upload
shared
    animations
    components
    directives
    models
    pipes
```

Figure 10: Angular file structure

The **config** folder contains app settings and other predefined values.

The global styles for the project are placed in a **scss** folder under assets.

3.3.3. Database model

We have five main DB models that we have to take in mind:

1. **Firmware:**
    a. Firmware name
    b. Version: We used semantic versioning to keep track of the current firmware version on both the cloud and the vehicle.
    Given a version number MAJOR.MINOR.PATCH, increment the:
    MAJOR version when you make incompatible API changes, MINOR version when you add functionality in a backwards compatible manner, and PATCH version when you make backwards compatible bug fixes.
    c. Location: storage location on DB disk. On larger scale applications this can be the reference URL for the file on another sever.
    d. Comment: comments for the developer or team leaders to identify this firmware or comment on the current version.
    e. Statistics reference ID: ID of the relevant statistics document.
2. **Statistics:**
    a. Download request: This includes an integer counter for:
        i. Success: Number of successful firmware downloads that the vehicle reported back.
        ii. Failure: Number of failed attempts to get the file. This may be due to firmware file corruption, or error reading the file,
        iii. Unknown: If we can’t find the firmware in the DB itself or if the DB didn’t respond. This may happen due to low server host resources or incorrectly deleting a firmware that is still used by a vehicle.
    b. Version request: The number of times vehicle requested this firmware.

3. **Team:**
    a. Team name: Name to identify the team.
    b. Description: What is this team responsible of.
    c. Members: array of user IDs that belong to this team.
    d. Logo: location of the team logo
    e. Secret: A secret key used for the whole team OTP synchronization from their mobiles.
    f. Join data: When was this team created.
    g. Firmware: Array of firmware IDs that this team manage.
4. **User:**
    a. Email: Login e-mail of the user.
    b. Name: the first and last name of the user.
    c. Uploads: number of times this user uploaded a firmware.
    d. Join Date: The date this user joined the platform.
    e. Password: we store a bcrypt-ed hashed password. This way if the DB is compromised (hopefully not) the attacker won't be able to retrieve the original results.
    f. Image: The user's image to display on the platform.
    g. Role: The user role that will adjust their privileges. Allowed options: admin, leader, or developer.
    h. Team: The team ID that this user belongs to.
    i. User key: The hashed key that will allow the user to register to the platform at the first place from the invitation e-mail.
5. **Vehicle:**
    a. Vehicle name: the login name for authentication of the vehicle
    b. Password: A bcrypt-ed hashed password for the vehicle authentication with the API
    c. Hash type: all the firmware hex files that are sent are hashed using a hashing algorithm. This algorithm can be changed with each firmware download request or for each vehicle. Say one line of production vehicles are using MD5 while

others are using SHA256; this model allows the flexibility to do so. This adds to the security of the system and it makes the firmware tailor-made for each vehicle.

d. Public key: used in combination with the hashing to provide a more secure way for firmware downloading.

3.4. User roles

To apply the least-privilege principle, we did three user roles. Each has a specific set of actions that they can execute.

1. **Developer:**
    a. Read own team, and update own team firmware file
    b. Read/update own profile
2. **Team Leader:**
    a. Same as developer
    b. Create/Delete own team firmware
3. **Developer:**
    a. Same as team leader
    b. Create any user and team
    c. Read any team
    d. Delete any profile or team.
4. **Visitor:**
    a. Visit the homepage and see the API

3.5. API Endpoints

Simply put, an endpoint is one end of a communication channel. The vehicle sends a request to some URL with specific parameters or shape to get specific into from the server.

GET /API (visitor)

- Swagger UI to get all the available end-points, their description, and test the request/response shape.

Vehicle End-points

POST /firmware/v/{firmwareID} (visitor)

- Called from the vehicle to the API with the vehicle's credentials to get a specific firmware semantic version (in the form of "vXX.XX.XX")

POST /firmware/get/{firmwareID} (visitor)

- Returns the firmware's hash followed by hex data. Each divided by n-chunks
- Called from the vehicle to the API with the vehicle's credentials to get the firmware's hashed hex data followed by the hex data itself. Each divided by n-chunks.

POST /vehicle/diagnostics/{uniqueID} (visitor)

- Sends diagnostics to the terminal

Firmware End-points

DELETE /firmware/delete/{firmwareID} (Admin)

- Deletes the firmware and remove its file

POST /firmware/update (Leader/Developer)

- Creates or updates a new firmware

GET /statistics/{firmwareID (Admin/Leader)

- Returns the number of failed and succeeded requests made by vehicles as well as the number of version requests.
-

Team End-points

GET /team/{teamID} (Admin, Leader, and Developer)

- Return a specific team info

DELETE /team/{teamID} (Admin)

- Deletes a team and remove its logo file.

GET /teams (Admin)

- View all teams

GET /team/firmwares (Leader, and developer)

- Gets a list of logged in user team’s firmwares

POST /team (Admin)

- Add a new team

User End-points

POST /user/login (visitor)

- Login the user

POST /user/add (Admin)

- Adds a new user

DELETE /user/{userID} (Admin)

- Deletes a user and removes their profile image

POST /user/register (Admin)

- Registers the user

3.6. Screens

3.6.1. Homepage

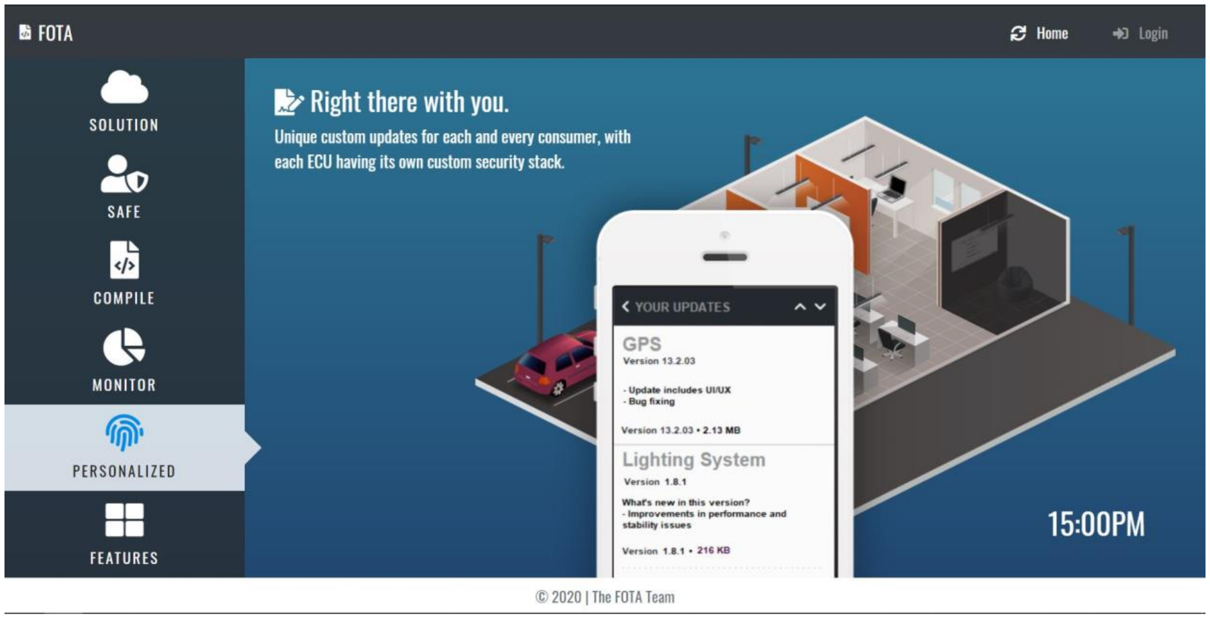


Figure 11: Home screen

Visitors will land on this welcome screen to see an overview of what the solution offer as well as go to the login page to get into the account.

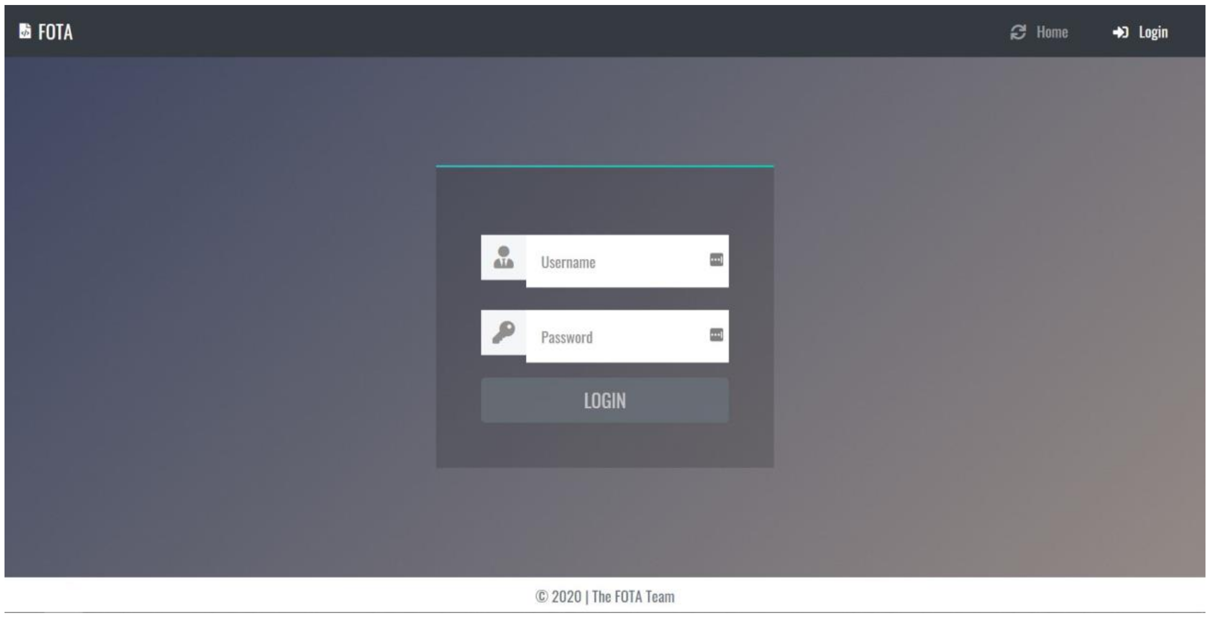


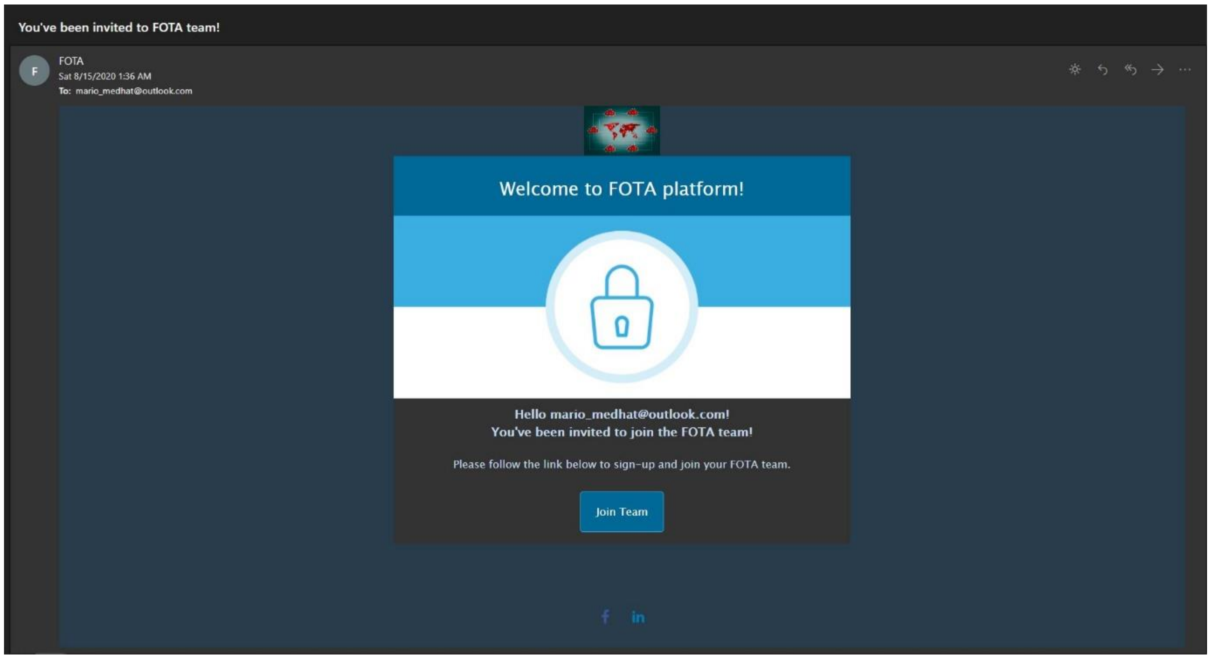
Figure 12: Login screen

There is not registration page as this feature is only limited to admin accounts which are added from the server shell access (SSH) by the devops team.

3.6.2. Registration

Upon adding a new user by the admin, an e-mail is sent to the user to activate and setup their account.

This is done by a signed JWT token which we will discuss later in further detail.


You've been invited to FOTA team!

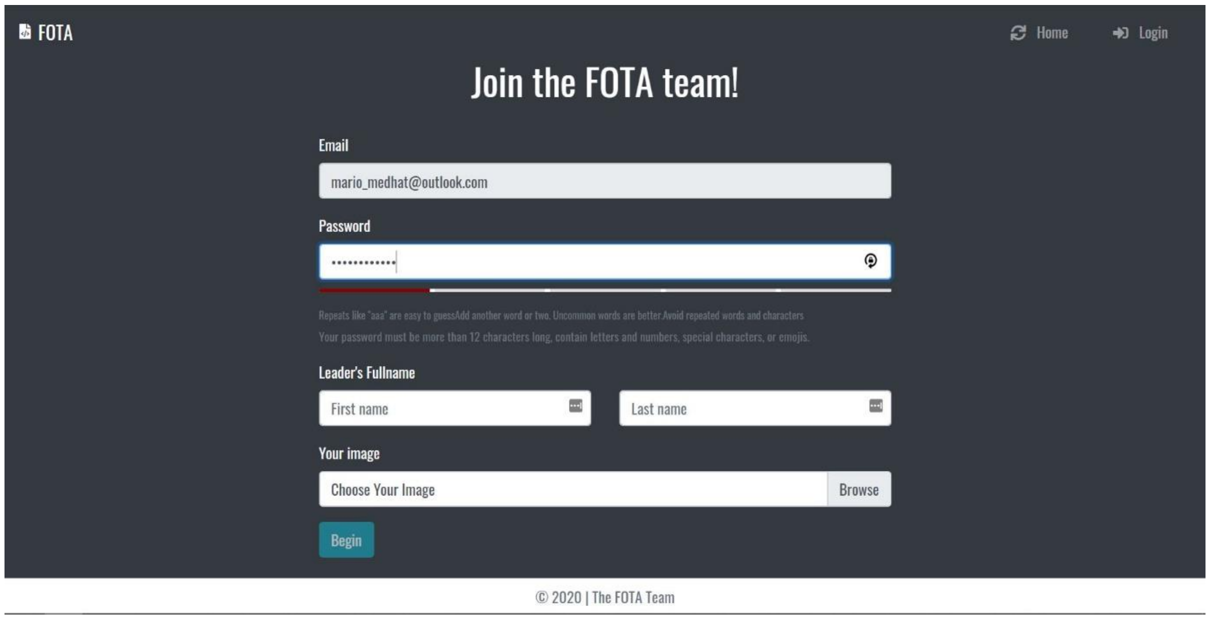
FOTA
Sat 8/15/2020 1:36 AM
To: mario_medhat@outlook.com

Welcome to FOTA platform!

Hello mario_medhat@outlook.com!
You've been invited to join the FOTA team!

Please follow the link below to sign-up and join your FOTA team.

Figure 13: Invitation e-mail



FOTA

Home Login

Join the FOTA team!

Email

mario_medhat@outlook.com

Password

••••••••••••

Repeats like "aaa" are easy to guessAdd another word or two. Uncommon words are better.Avoid repeated words and characters

Your password must be more than 12 characters long, contain letters and numbers, special characters, or emojis.

Leader's Fullname

First name

Last name

Your image

Choose Your Image

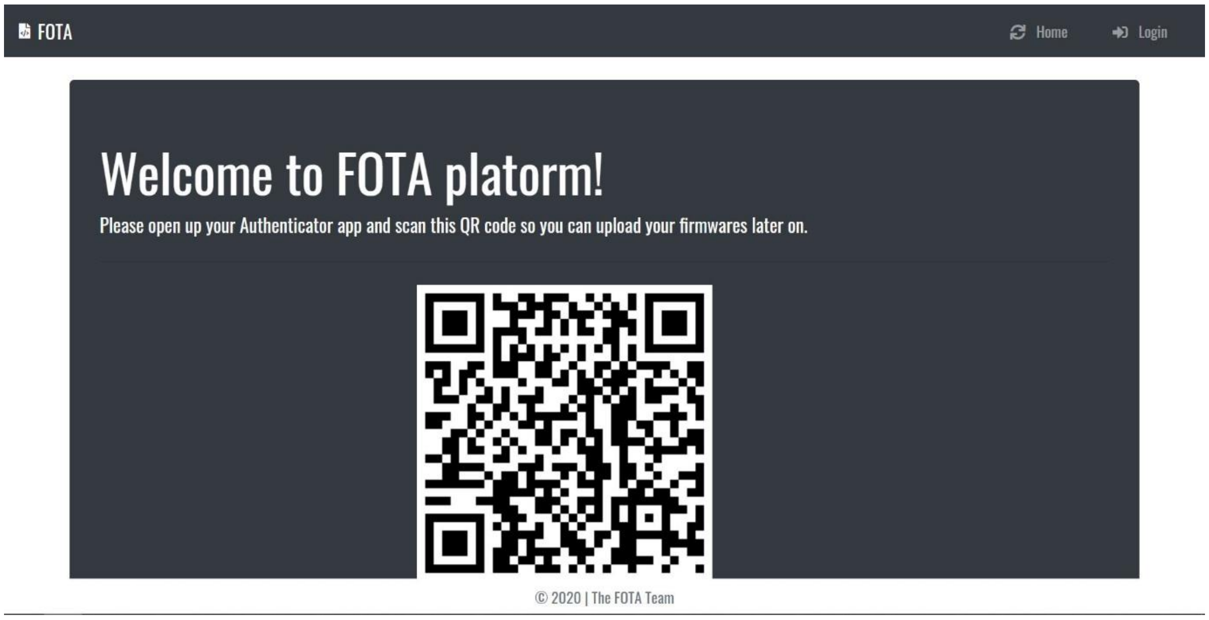
Browse

Begin

© 2020 | The FOTA Team

Figure 14: Register screen

After clicking on the on-screen link, the user is welcomed to their new team with some info to fill. Note that the password must be complete in order to encourage the team members to make stronger passwords.



FOTA

Home Login

Welcome to FOTA platorm!

Please open up your Authenticator app and scan this QR code so you can upload your firmwares later on.

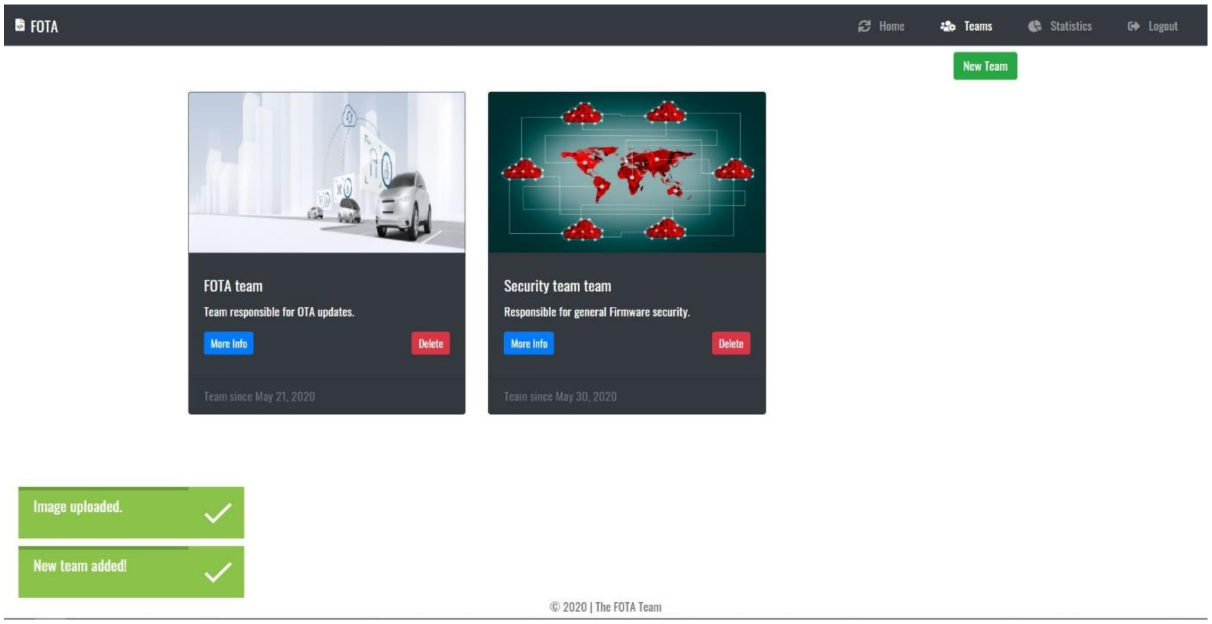
© 2020 | The FOTA Team

Figure 15: Authentication QR code

After filling the team member's info, the developer must then scan a QR code to be able later on to upload firmwares and engage on the platform.

3.6.3. Teams

Admin has access to overview of all teams. As well as the ability to add or delete teams.



FOTA team
Team responsible for OTA updates.
Team since May 21, 2020

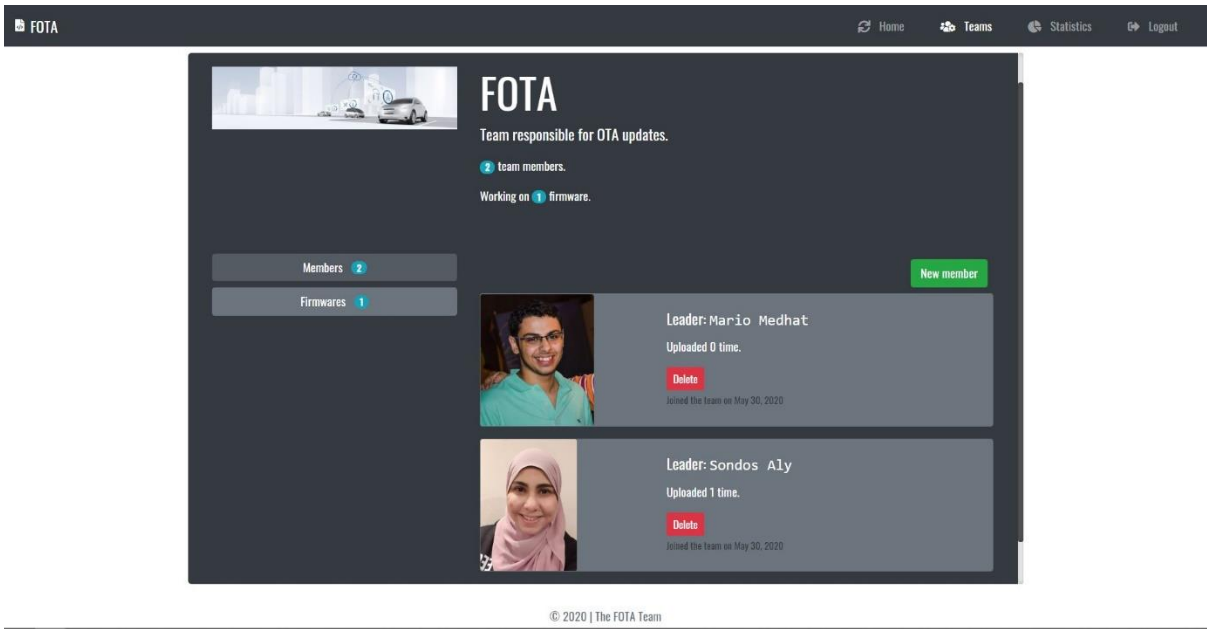
Security team team
Responsible for general Firmware security.
Team since May 30, 2020

Image uploaded.
New team added!

© 2020 | The FOTA Team

Figure 16: All teams Screen

3.7. Team info



FOTA

Team responsible for OTA updates.

2 team members.

Working on 1 firmware.

Leader: Mario Medhat

Uploaded 0 time.

Joined the team on May 30, 2020

Leader: Sondos Aly

Uploaded 1 time.

Joined the team on May 30, 2020

© 2020 | The FOTA Team

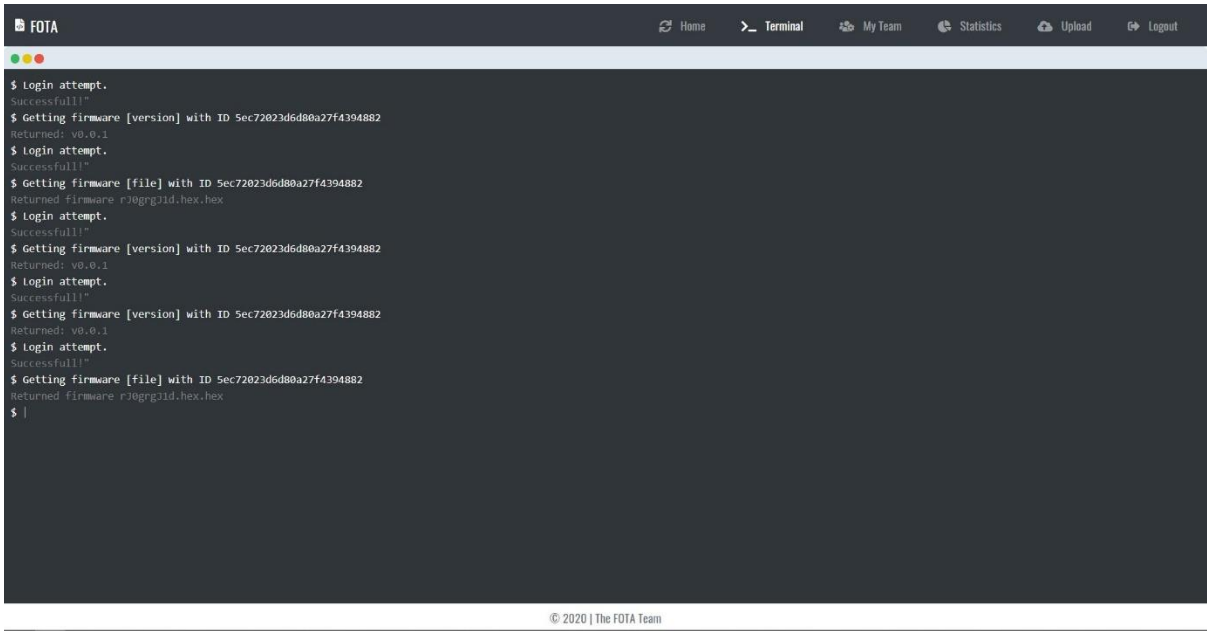
Figure 17: Team info

For all team members and admins, they can see a full-detailed screen for a specific team.

It includes the team members, their roles, contribution, and the assigned firmwares to that team.

The admin has the ability to add or remove new team members. If a team member is already on the platform but in another team, they will be invited to that new team.

3.7.1. Terminal



```
$ Login attempt.
Successfull!"
$ Getting firmware [version] with ID 5ec72023d6d80a27f4394882
Returned: v0.0.1
$ Login attempt.
Successfull!"
$ Getting firmware [file] with ID 5ec72023d6d80a27f4394882
Returned firmware rJ0grgJ1d.hex.hex
$ Login attempt.
Successfull!"
$ Getting firmware [version] with ID 5ec72023d6d80a27f4394882
Returned: v0.0.1
$ Login attempt.
Successfull!"
$ Getting firmware [version] with ID 5ec72023d6d80a27f4394882
Returned: v0.0.1
$ Login attempt.
Successfull!"
$ Getting firmware [file] with ID 5ec72023d6d80a27f4394882
Returned firmware rJ0grgJ1d.hex.hex
$ |
```

Figure 18: Terminal

To see a real-time feed for the vehicle requests, a terminal-like screen was made to display the requests flow as they come in.

This terminal screen will be used as well in software diagnostics where a vehicle can send to the terminal using unique ID and the developer can see software info sent by the vehicle and diagnose it.

3.7.2. Statistics

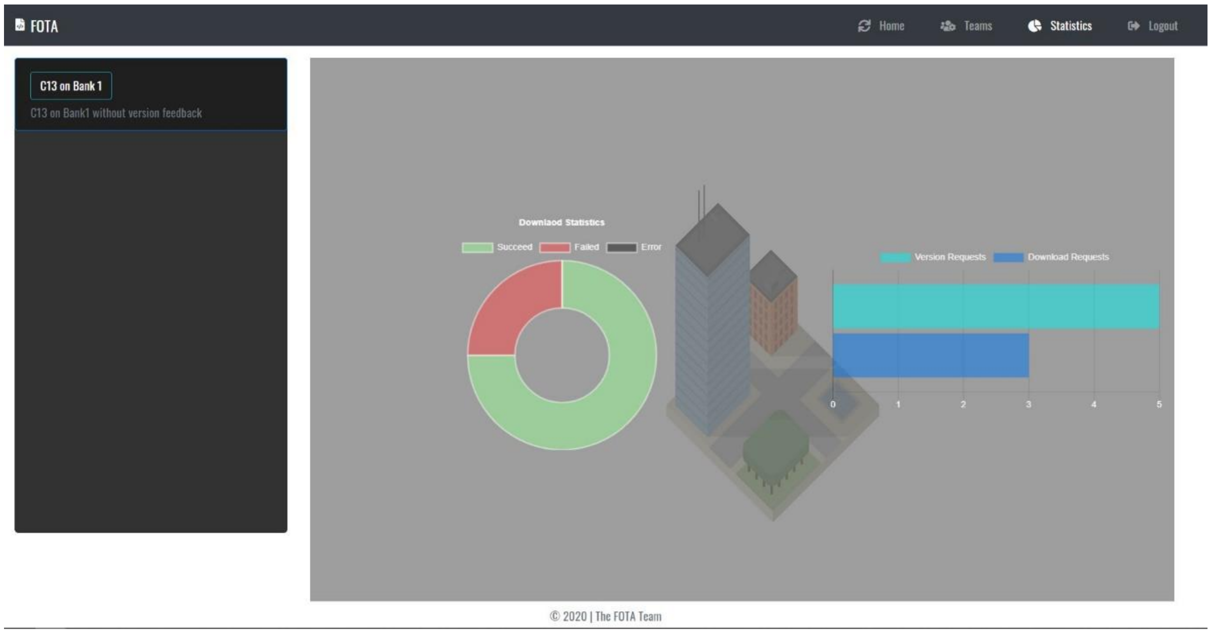


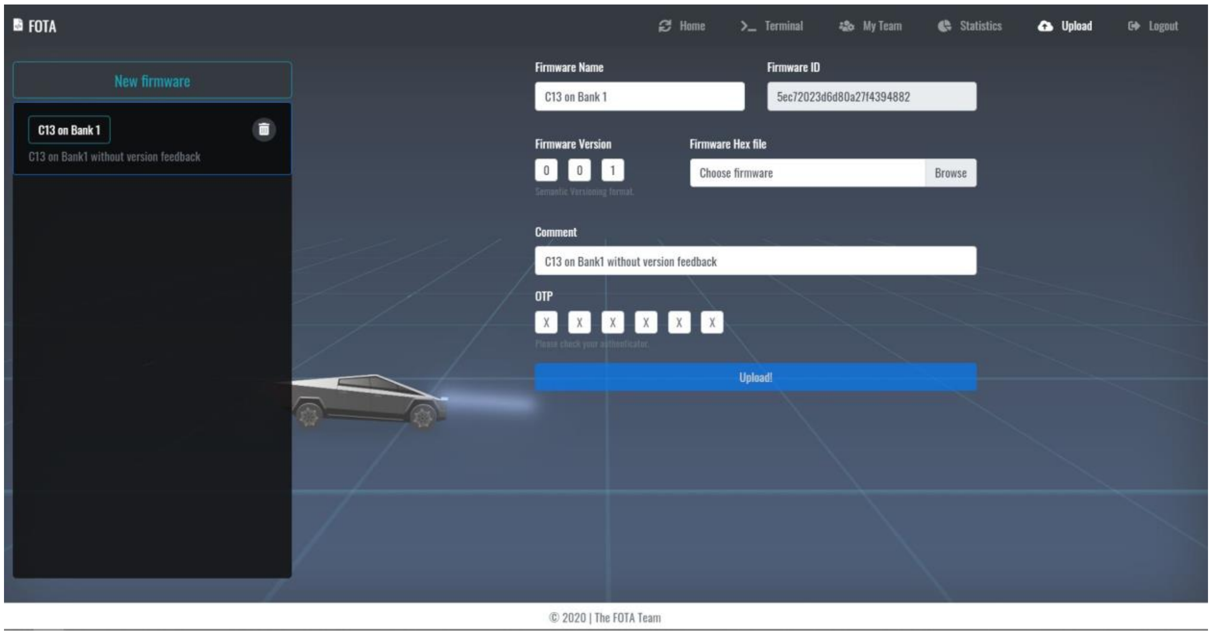
Figure 19: Statistics Screen

For the admin and team leaders, they need to see the impact and benefit of their software releases. This screen demonstrates the download success rate and version request vs. download request.

For the business team, as big as the version request gets larger, that's how much recalls the company saved.

These statistics are saved for all software versions for each firmware of the team.

3.7.3. Upload Screen



C13 on Bank 1
C13 on Bank1 without version feedback

Firmware Name: C13 on Bank 1
Firmware ID: 5ec72023d6d80a27f4394882
Firmware Version: 0 0 1
Semantic Versioning format.
Firmware Hex file: Choose firmware
Comment: C13 on Bank1 without version feedback
OTP: X X X X X X
Please check your authenticator.

Figure 20: Upload screen

This page is accessible by the team leader and developer to actually upload the hex file that will be downloaded by the vehicles.

Each team will have access to firmwares to work on. Each firmware is identified by a unique hashed MD5 key. This key is statically stored on the vehicle's flash memory so that it requests it from the server. This unique hash key is automatically calculated by MongoDB.

Also, a human-readable name and description is supplied by the developer to offer more info to the user's HMI system inside the vehicle as well as to other developers working on the same firmware.

Each firmware has a unique version that identifies to the vehicle what kind of update happened.

We use Semantic versioning here which is most commonly used in software components. Given a version number MAJOR.MINOR.PATCH, we increment the MAJOR version when you make incompatible API changes, MINOR version when you add functionality in a backwards compatible manner, and PATCH version when you make backwards compatible bug fixes.

Last thing is a 6-digit OTP that the developer must supply from their mobile phone. This ensures the developer is the only one capable of applying this update.

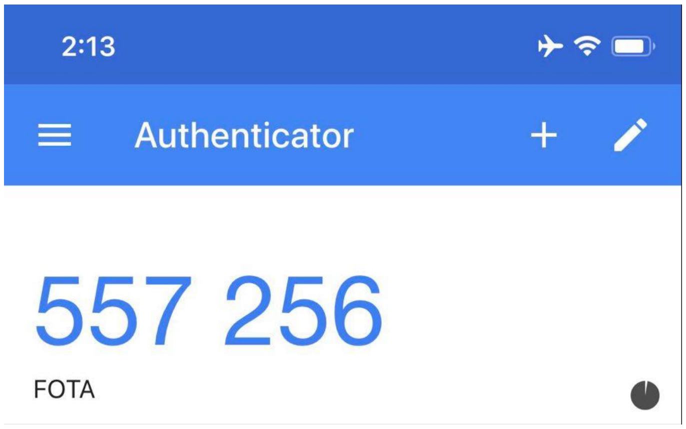


Figure 21: Google Authenticator OTP

After the update happens, the new hex file will be stored in a quarantined folder so it’s not accessible by the external API yet. The first bytes of the file are then checked to make sure it’s indeed a hex file and has its signature. If all checks pass, the file is then moved to the working directory of the database where vehicles can read it, and the new version is applied to the DB so that vehicles can identify that they have a new firmware ready.

3.7.4. Security

As HTTP is stateless protocol, we used JWT tokens as they are a good way of securely transmitting information between parties because they can be signed, which means you can be sure that the senders are who they say they are. Additionally, the structure of a JWT allows you to verify that the content hasn't been tampered with. (See Appendix A)

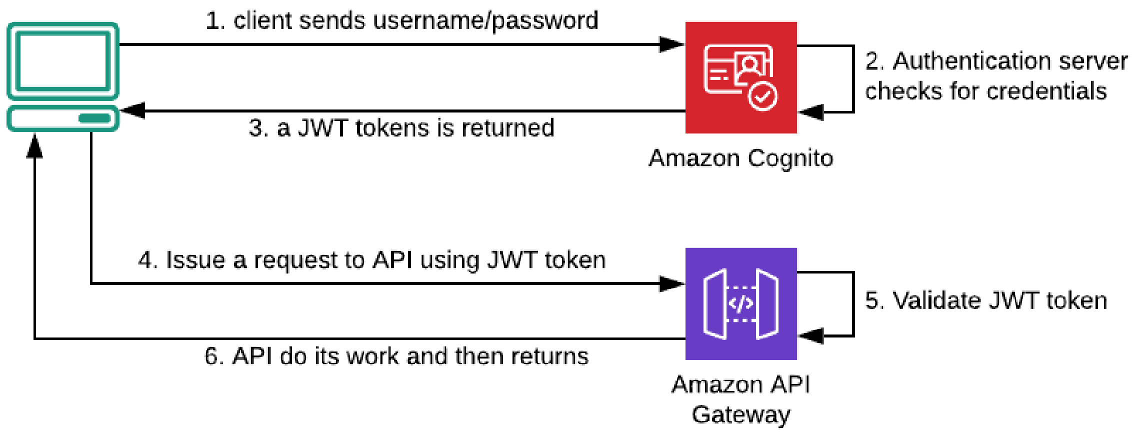

1. client sends username/password
2. Authentication server checks for credentials
3. a JWT tokens is returned
Amazon Cognito
4. Issue a request to API using JWT token
5. Validate JWT token
6. API do its work and then returns
Amazon API Gateway

Figure 22: JWT token

Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema. The content of the header should look like the following:

Authorization: Bearer <token>

3.7.5. Why did we use JWT?

As JSON is less verbose than XML, when it is encoded its size is also smaller, making JWT more compact than SAML. This makes JWT a good choice to be passed in HTML and HTTP environments.

Security-wise, JWT can only be symmetrically signed by a shared secret using the HMAC algorithm. However, JWT and SAML tokens can use a public/private key pair in the form of a X.509 certificate for signing. Signing XML with XML Digital Signature without introducing obscure security holes is very difficult when compared to the simplicity of signing JSON.

JSON parsers are common in most programming languages because they map directly to objects. Conversely, XML doesn't have a natural document-to-object mapping. This makes it easier to work with JWT than SAML assertions.

Regarding usage, JWT is used at Internet scale. This highlights the ease of client-side processing of the JSON Web token on multiple platforms, especially mobile.

3.7.6. Refresh token

They carry the information necessary to get a new access token. In other words, whenever an access token is required to access a specific resource, a client may use a refresh token to get a new access token issued by the authentication server. Common use cases include getting new access tokens after old ones have expired, or getting access to a new resource for the first time. Refresh tokens can also expire but are rather long-lived. Refresh tokens are usually subject to strict storage requirements to ensure they are not leaked. They can also be blacklisted by the authorization server.

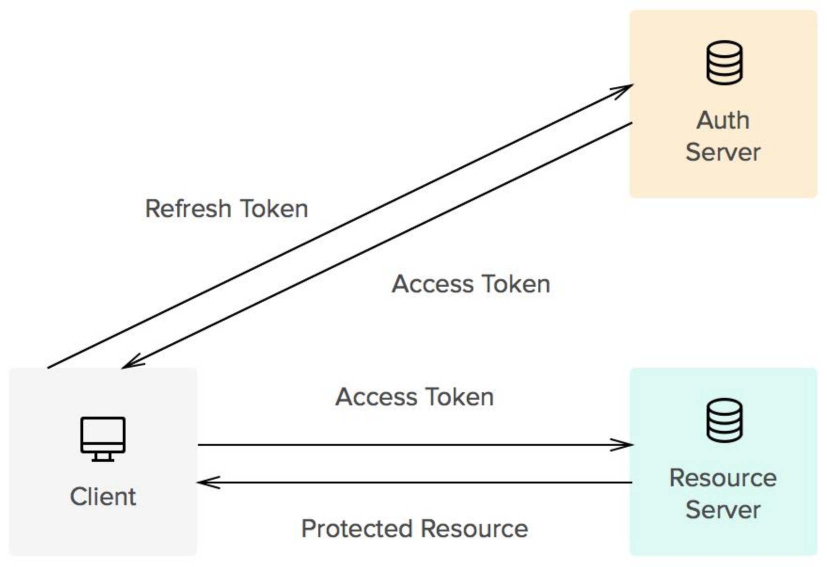


Figure 23: Refresh token flow

3.7.7. XSS and Injection attacks

We made sure all fields are sanitized and only some set of characters are allowed to disallow Cross-site scripting attacks and database injection attacks where the user injects some code to gain access to sensitive info on the database of gain information about the server or DB architecture.

3.7.8. Passwords

We used a hash function for passwords so that in the worst-case scenario, if the DB is compromised, the user's passwords won't be leaked.

All passwords are hashed using `*bcrypt*` before storing in the DB. It's based on the famous Blowfish cipher beside incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power.

3.7.9. Headers and SSL

Connection to the server from vehicle and from the UI must be established on a secure channel (SSL) to make sure no eavesdroppers can get any info if the channel is leaked.

Also, CSP headers are presented as an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks like packet sniffing attacks.

3.7.10. Social Engineering

After all, the worst kind of attack that can happen on a any platform is social engineering. We tried to limit this by applying the principle of least privilege (Admin, Leader, and Developer roles)

Chapter 4: Communication Protocols

4.1. UART

UART stands for Universal Asynchronous Receiver Transmitter, it’s a dedicated hardware associated with serial communication. The hardware for the UART could be a dedicated IC or it could be an integrated circuit on the microcontroller. Unlike SPI and I2C, that are just communication protocols.

UART is one of the most commonly used Serial Communication techniques as it’s very simple. Today, UART is being used in many applications like GSM, Modems, GPS Receivers, Wireless Communication Systems, Bluetooth Modules, and GPRS, RFID based applications etc.

Most of the microcontrollers have dedicated UART hardware built in to their architecture. The main reason for integrating the UART hardware in to microcontrollers’ architecture is that it is a serial communication and requires only two wires for communication, one for transmission and the other for reception.

4.1.1. Advantages of UART

1. Serial protocol
   Serial protocols are much better than parallel protocols as parallel protocols have a lot of problems such as data skew, high cost, high complexity, and interference between wires due to magnetic field.
2. This protocol requires only two wires for full duplex data transmission (apart from the power lines).

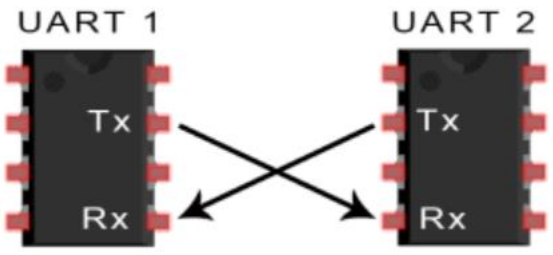


Figure 24: Connection of UART

3. No need for clock or any other timing signal.
4. Parity bit ensures basic error checking is integrated in to the data packet frame.4

4.1.2. UART Frame

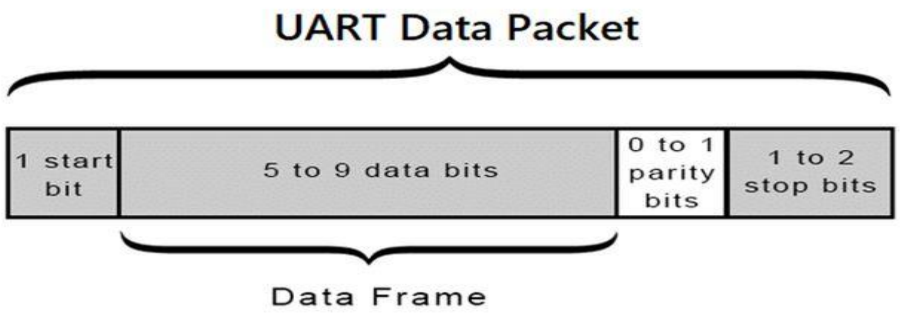


Figure 25: UART Frame

1) Start Bits

The IDLE state of the UART transmission line is high voltage when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for only one clock cycle. When the receiver detects the high to low voltage transition, it begins to read the bits in the data frame at the frequency of the baud rate.

2) Data Bits

The data bits contain the actual data that is being transferred. Its length can vary from 5 bits up to 8 bits long if a parity bit is used. If there is no parity bit, the data length can be 9 bits long. In most cases, the data is sent with the least significant bit first.

3) Parity Bits

Parity describes the evenness or oddness of a number. The parity bit is a way for the receiver of the UART to detect if any data has been changed during data transmission. Data could be changed due to mismatched baud rates, electromagnetic radiation, or long-distance data transfers. After the receiver reads the data frame, it counts the number of ones in it and checks if there is an even or odd number of ones. If the parity bit is a 0 this indicates even parity, which means that the total number of ones in the data should be an even number. If the parity bit is 1 which indicates odd parity means that the total number of ones in the data should be an odd number. If the parity bit matches the data, the UART knows that the transmission is free of errors and has been done successfully. But if the parity bit is the frame is 0, and the total number of ones in the data is odd; or the parity bit is a 1, and the total number of ones is even, the UART knows that an error occurred during transmission and bits in the data frame have changed.

4) Stop Bits

To signal the end of the data packet, the transmitter signals the data transmission line from a low voltage to a high voltage to make it in the IDLE state again for at least two-bit durations.

4.2. CAN Bus

4.2.1. What is CAN?

CAN stands for Controller Area Network

The CAN Bus is a serial two-wire half-duplex communication that allows multiple nodes in a system to communicate efficiently with each other. Each node is capable of sending and receiving messages, not all nodes can be communicating at once. All nodes receive all broadcast data from the bus then each node decide whether or not that data is relevant according to the filter masks of each node as shown in the following figure.

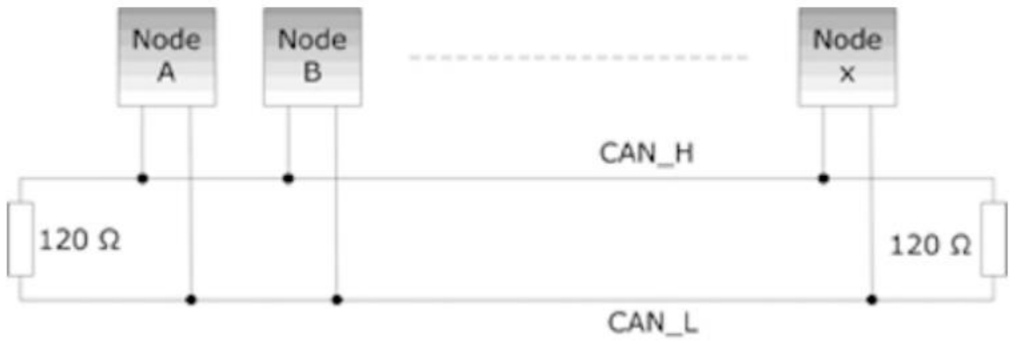


Figure 26: CAN Network

Any node wants to send a message it puts it on the bus with specific ID , then each node from the other nodes will check weather this ID is included in its filters or no , if yes it reads the message from the bus ,otherwise it ignores it.

CAN protocol is already widely used in vehicle and vessel internal components. In recent years, it has seen adoption in data communications and control in industry field. [3]

4.2.2. Why did we choose CAN?

- **Speed variety**
  - Low Speed:

    CAN network offers low signal transfer rates that ranges between 40 kbps to 125 kbps.

  - High Speed:

    CAN offers high signal transfer rates between 40 kbps and 1 Mbps. High-speed CAN networks are terminated at both ends of the bus line with a 120-ohm resistor between CAN high and CAN low lines.

    The lower signaling rates allow communication to continue on the bus, even when a wiring failure takes place, while high-speed doesn’t.

- **Low cost**

When the CAN protocol was created, its aim and primary goal was to make faster communication between modules and electronic devices in vehicles while decreasing the amount of wiring (and the amount of copper) necessary.

- **Built-in Error Detection**

Error handling is built in the CAN protocol, each node can check for its errors during transmission and maintain its own error counter. When errors are detected nodes transmit a special Error Flag message and will destroy the offending bus traffic to prevent the error from spreading through the whole network. Even the node which is generating the fault will detect its own error in transmission, and raise its error counter, this eventually led the device to "bus off" and cease participating in the CAN network traffic. In this way, CAN nodes can both detect errors and prevent faulty devices from creating any useless bus traffic.

- **Flexibility**

To understand the flexibility of the CAN bus protocol in communications, we need to know the difference between message-based protocols and address-based and. In address-based communication protocol, nodes communicate directly with each other by configuring themselves onto the same address.

The CAN bus protocol is a message-based communication protocol. In this type of protocol, nodes on the bus have no address to identify them. As a result, nodes can easily be added or removed without performing any software or hardware updates on the system.

This feature makes it easy to integrate new electronic devices into the CAN bus network without significant programming overhead and supports a modular system that is easily modified to suit your specs or requirements. [4]

4.2.3. CAN specifications

- Wired

- Serial
- Asynchronous
- Multi-Master No Slave (MMNS)
- Half duplex

4.2.4. CAN Transmission Handling

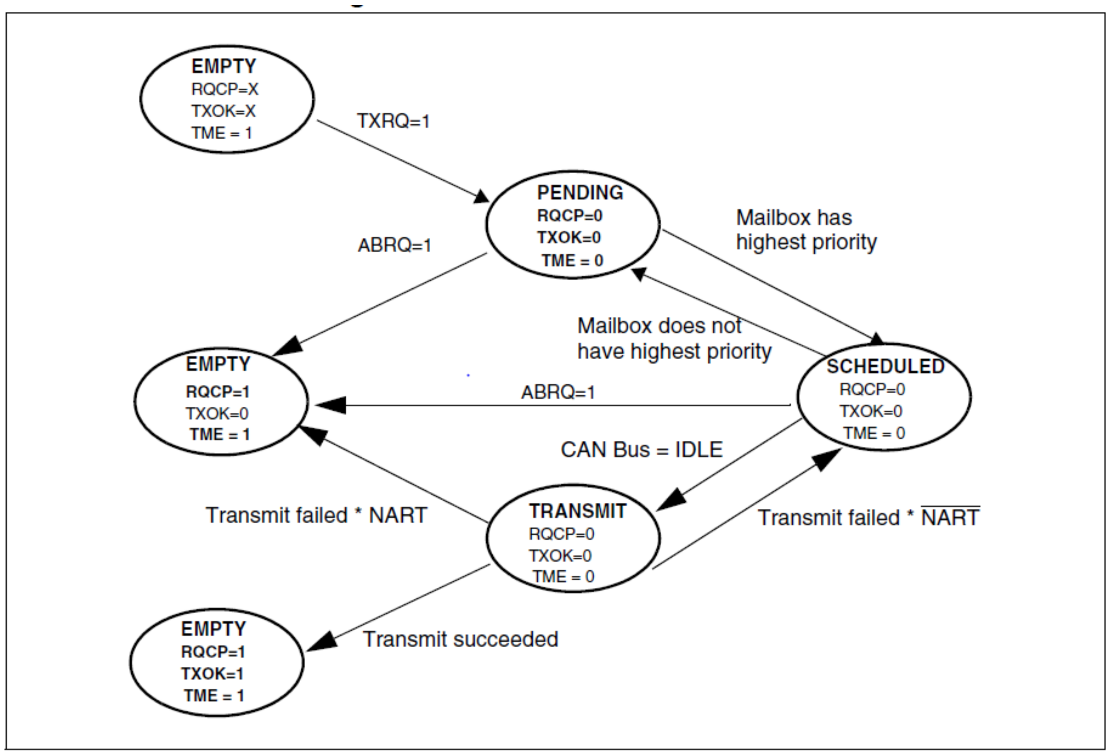


Figure 27: Transmit Mailbox States

1. In order to transmit a message, the application must select one empty transmit mailbox, setup the identifier, the data length code (DLC) and the data before requesting the transmission by setting the corresponding TXRQ bit (Transmit mailbox request) in the CAN_TIxR register.

2. When the mailbox left empty state and after the TXRQ bit has been set, the mailbox enters **pending** state and waits to become the highest priority mailbox **(in our case the highest priority is defined by transmit request order)** As soon as the mailbox has the highest priority it will be **scheduled** for transmission.

3. The transmission of the message will start (enter **transmit** state) when the CAN bus becomes idle, then once the mailbox message has been successfully transmitted by setting the RQCP(**Request Complete**) and TXOK(**Transmission Ok**) bits in the CAN_TSR register the mailbox becomes empty and we can use it again.

4.2.5. CAN Reception Handling

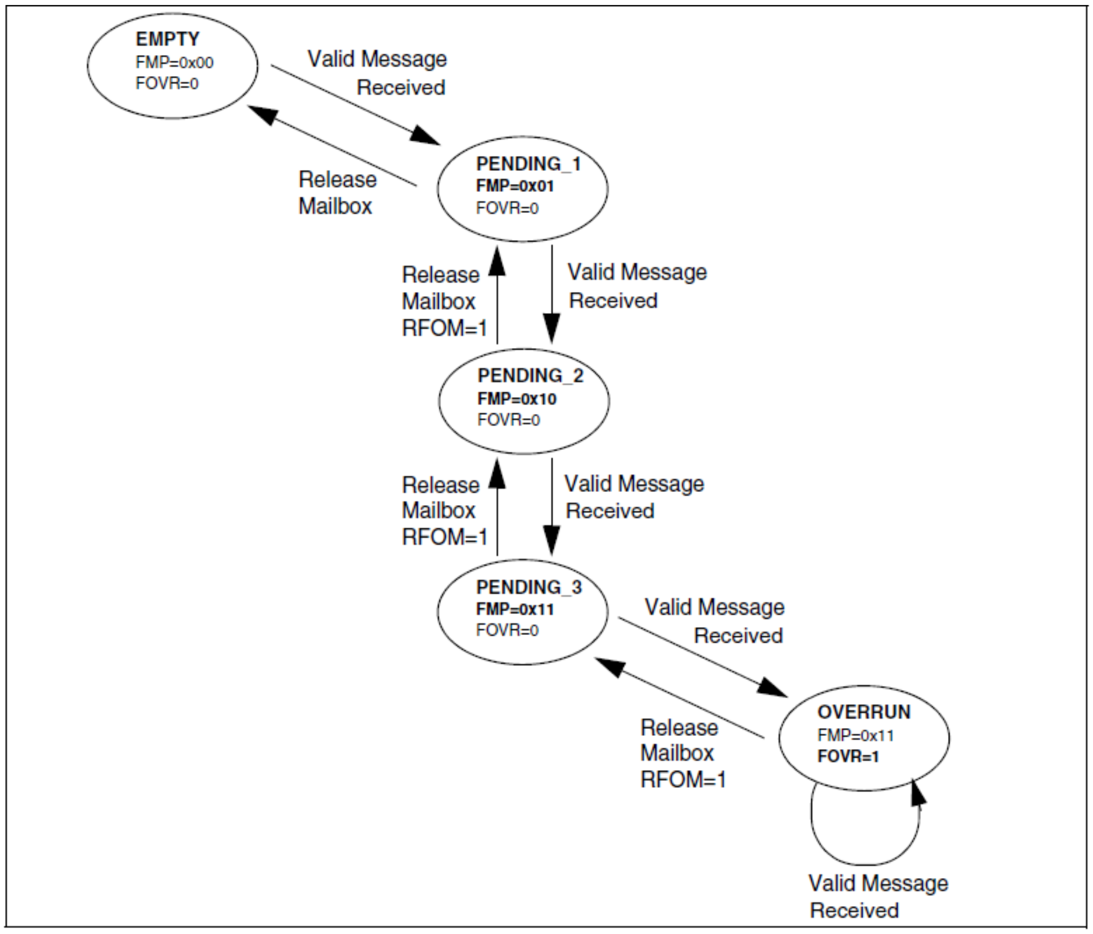


Figure 28: Receive FIFO States

We can receive up to three messages in the FIFO mailbox then if we start from the empty state, the first valid message received is stored in the FIFO which becomes pending_1 and if we receive the next valid message, it will be stored in the FIFO and enters pending_2 until we received three messages then we must release the FIFO mailbox in order to receive another messages. [5]

4.2.6. CAN Filters

We have 14 configurable and scalable filter banks (13-0) to the application in order to receive only the messages the software needs and each filter bank consists of two 32-bit registers.

- **Mask Mode:**

Filter masks are used to know which bits in the message identifier (ID) should be compared with the filter bits, as following:

The mask bit may be set to one or zero, if it is set to one the corresponding identifier bit will be compared with the filter bit, if they are matched the messages will be accepted otherwise it will be rejected. While, if a mask bit is set to zero, the corresponding identifier bit will automatically be accepted, regardless of the value of the filter bit.

Example 1:

We wish to accept only frames with ID of 00001657 (hexadecimal values)

Set filter to 00001657

Set mask to 1FFFFFFF

When a message is received its ID should be compared with the filter and all bits must match; if only one bit does not match the ID 00001567, this frame will be rejected.

Example 2:

We wish to accept the frames that have IDs from 00001650 to 0000165F

Set filter to 00001650

Set mask to 1FFFFFF0

When a message is received its ID should be compared with the filter and all bits except bits 0,1,2, and 3 must match; if any other bit does not match, this frame will be rejected.

- **Identifier list mode:**

We have a list of specific IDs, if the transmitted message carries ID that matches with anyone in the list, it will be accepted otherwise the frame is rejected. [6]

4.2.7. CAN frame

Frame types:

- **Error Frame – Reports error condition**

Any node in the CAN network sender or receiver, may signal an error condition at any time during a data or remote frame transmission.

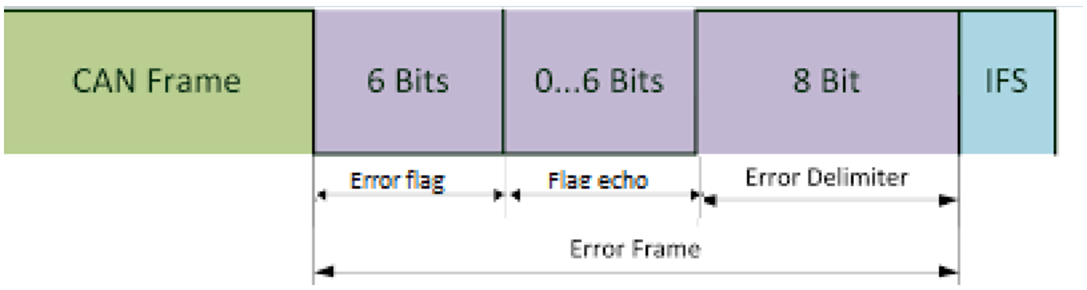


Figure 29:Error Frame

- **Overload Frame – Reports node overload**

A node sends overload frame to request a delay between two remote or data frames, so the overload frame can only occur between data or remote frame transmissions.

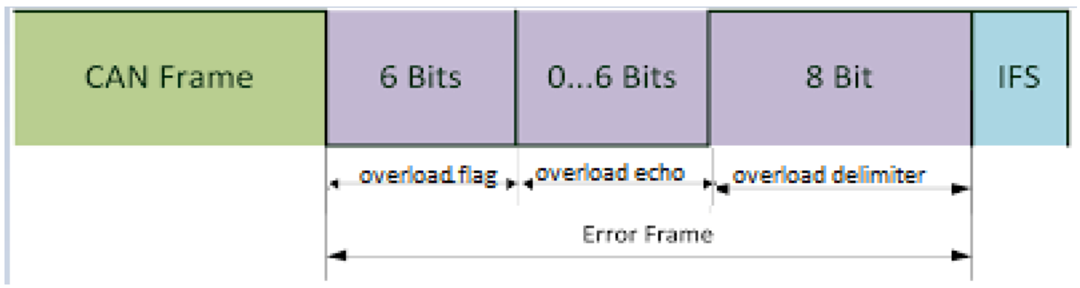


Figure 30: Error Frame

- **Data Frame – Sends data**

Data transfer from node to the CAN bus and the nodes that are interested in the message can read it (This is done according to the filters of the node).

- **Remote Frame – Requests data**

Any node may request data from other nodes. The remote frame represents the request so its consequently followed by a data frame containing the requested data.

Both data and remote frame have the same frame but in remote frame data = 0

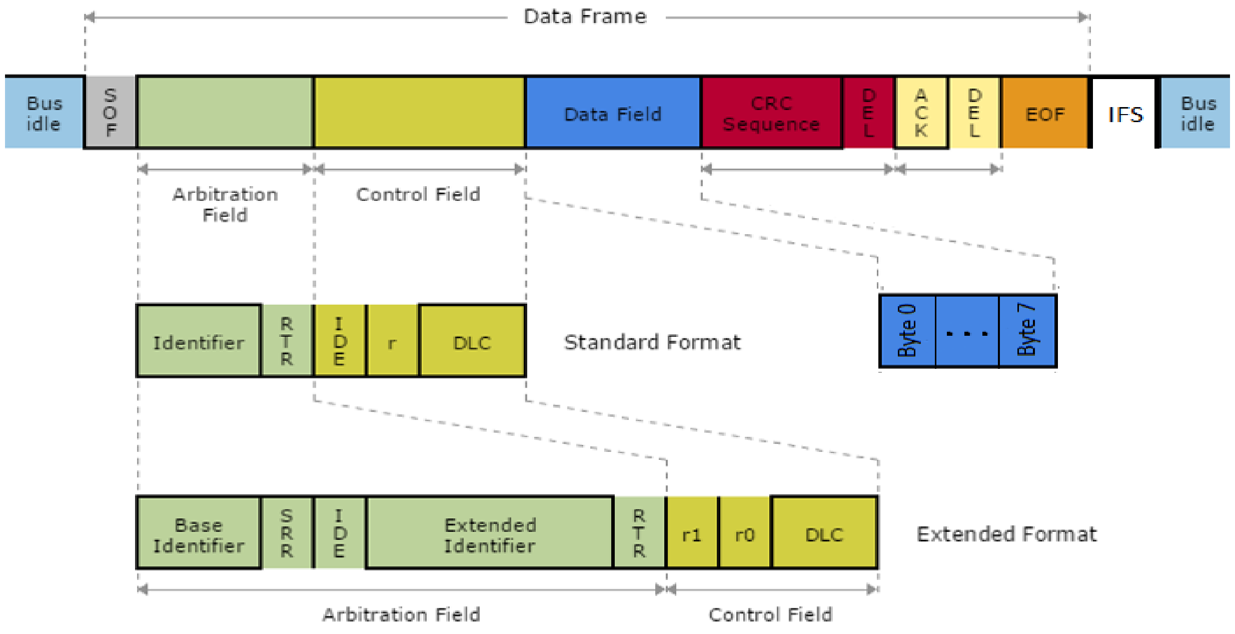


Figure 31: Data and Remote frames

- **SOF** (Start of Frame) (1 bit) – Marks the beginning of data and remote Frames
- **Arbitration Field** – Includes the message ID (11bits in standard ID and 29 bits in extended ID) and RTR (Remote Transmission Request) bit, which distinguishes data and remote frames.
- **Control Field** – DLC to determine data size (4 bits) and IDE (1 bit) to determine message ID length.

- **Data Field** – The actual data (which is zero in remote frame and contains the actual data frame)
- **CRC Field** (16 bits) – Checksum, CRC stands for Cyclic Redundancy Check, it contains Checksum and delimiter.
- **ACK Field** (2 bits) – Acknowledge and delimiter.
- **EOF** (End of Frame) (7 bits)– Marks the end of data and remote frames
- **IFS Field** (3 bits) **–** Inter-frame space

ACK delimiter: The acknowledgement is sent from the receiving node to the transmitting node and it need some time so ACK delimiter is used.

CRC delimiter: The ECU needs some time to calculate the CRC so a delimiter bit is introduced to make some delay for the ECU. [7]

4.2.8. CAN in our project

The specifications that are suitable with our needs are:

- Speed : 500 Kbps.
- Priority : By transmit request order.
- Using 3 mailboxes each can have 8 bytes of data

We used CAN remote frames to control our system , here is the usage of remote frames in our system :

- Ask and get response from the user weather he wants to accept the update or not.
- Give ACK that the data is transmitted from main to app ECUs.
- Get the version on the app ECU
- Wait for any update request from main to app ECU

While data frames are used to transmit hex file from main ECU to app between ECUs and also transmit update progress from main ECU to GUI.

4.2.9. Problems that faced us with CAN

1. CAN Network should be supplied with 5v.
2. The connections using wires is not stable, try to make a PCB that contains the CAN network.

4.2.10. Procedures to follow to check if your network is working or not

1. First put a list of the ID's that each node can accept in the filter registrs
2. Try with only two nodes and only one mailbox and one FIFO.
3. Work with loopback mode on the first node (this mode transmits only) and with silent mode on the second one (this mode recives only).
4. If step 2 worked try to change the first node to silent mode and the other to loopback mode
5. If step 3 worked well , that's great , now try normal mode.
6. If step 5 worked now you should only scale the driver to use the 3 mailboxes and 2 FIFOs .

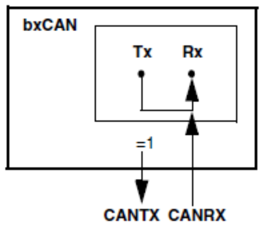


Figure 33: Node in silent mode

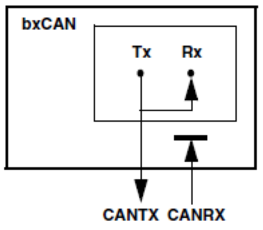


Figure 32: Node in loopback mode

This chapter was about our communication protocols that we used in our project and how each of them could work. The next chapter will talk about the Main ECU in our project which can connect between our server and all ECUs in our project.

Chapter 5: Main ECU

5.1. Introduction

Main ECU plays a critical role in Firmware update cycle as it acts as the wireless communication gate way for the vehicle. After vehicle ignition main ECU starts to fetch all vehicle ECUs software versions, then starts to request latest ECUs software version existing on the OEM server, by comparing vehicle ECUs software versions and the latest ECUs software versions from OEM server response it decides which ECUs need an update, then it ask the user through HMI system whether he accept to make this update or not, After that it starts to download the needed software versions sequentially, using several cryptographic algorithms Main ECU authenticate the file sender and check integrity of the file to guarantee that it received the file from the intended sender and the received file wasn't modified in the middle channel, after it check the received file validity it starts to send the file through CAN bus to the intended ECU. Main ECU hardware include GSM module and STM32 microcontroller based on ARM cortex M3 Architecture.

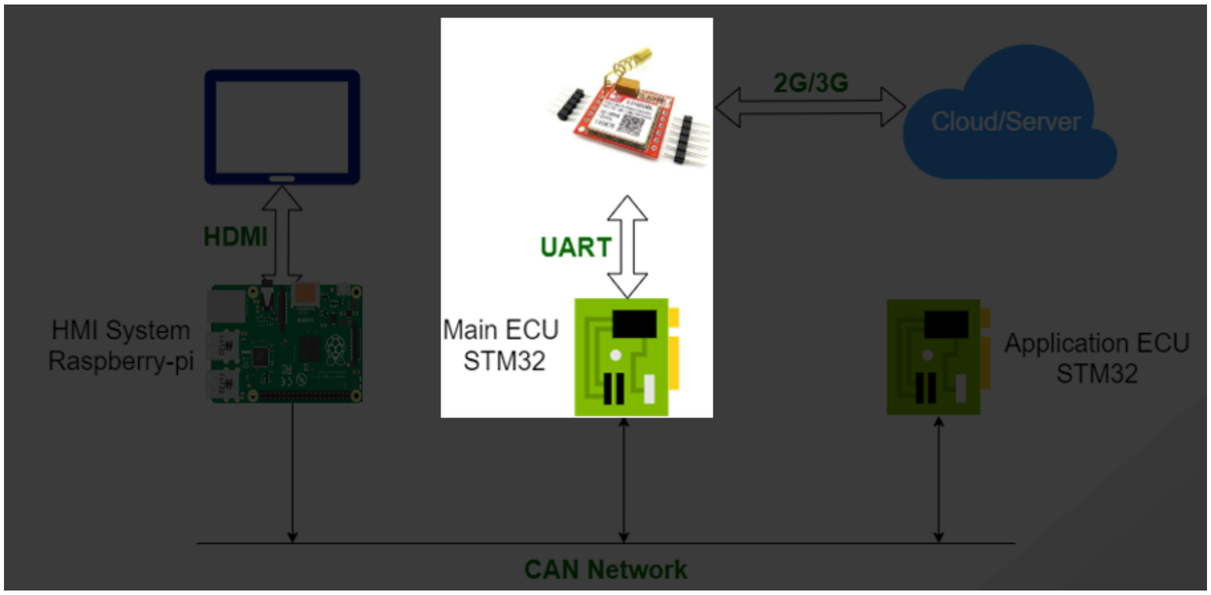


Figure 34: Main ECU interfacing with system

5.2. Literature review

Pervious FOTA systems depend on different implementations for Main ECU (Telematic ECU), Communication system with internet is the most critical aspect in this cycle which distinguish different FOTA systems from others, every system has its own advantages and disadvantages, trade-off between different systems is based on security, reliability, cost, user friendly and download speed.

I. Wi-Fi based implementation, Wi-Fi module connected to the microcontroller to access the internet through local network connection, this implementation didn't achieve the appropriate user experience, as internet local network isn’t available everywhere and even where it available you shouldn’t move with your vehicle but wait until the update download is finished.

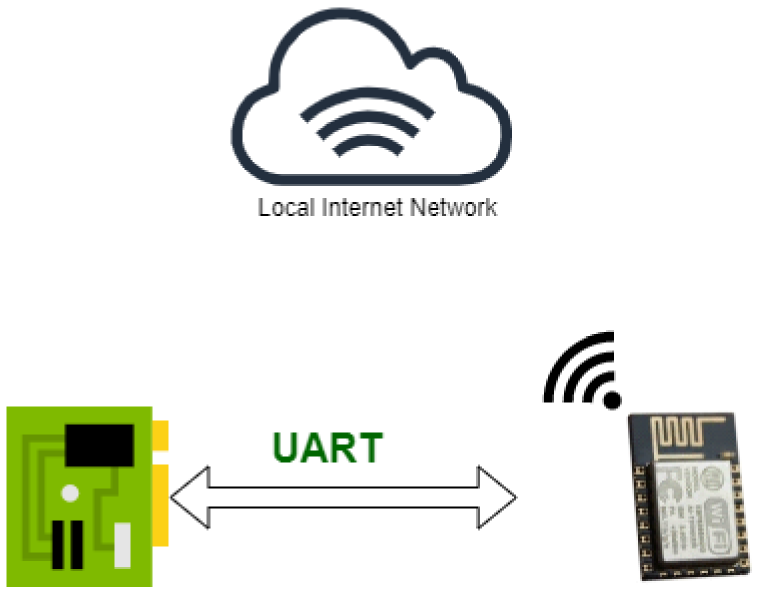


Figure 35: WI-FI based implementation scheme

II. Bluetooth and Mobile phone based implementation, mobile is connected with internet through cellular network and connected to ECU through Bluetooth connection, dissipation of mobile battery to enable data connection with internet and to enable Bluetooth connection with ECU is considered as system drawback, file download speed will be negatively affected as the file is downloaded to the mobile phone then mobile will send it to ECU through Bluetooth so the data will suffer double latency.

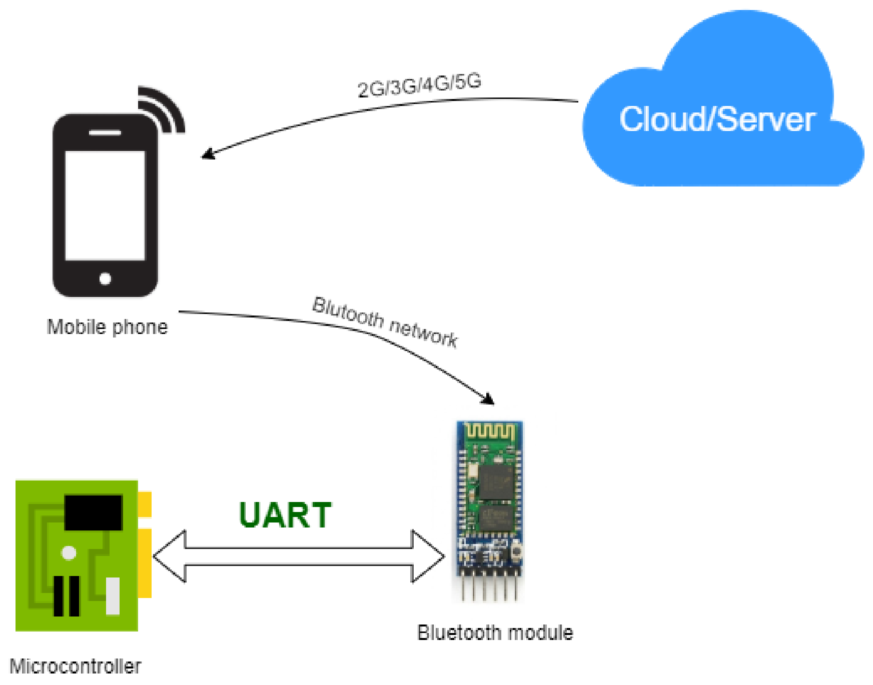


Figure 36: Bluetooth and mobile based implementation scheme

III. Cellular network-based implementation, vehicle connected with the server directly through cellular network data communication over data communication built-in module, this implementation satisfies user expectations, all system building blocks is independent on any external devices, seamless internet connection is achieved in through this implementation.

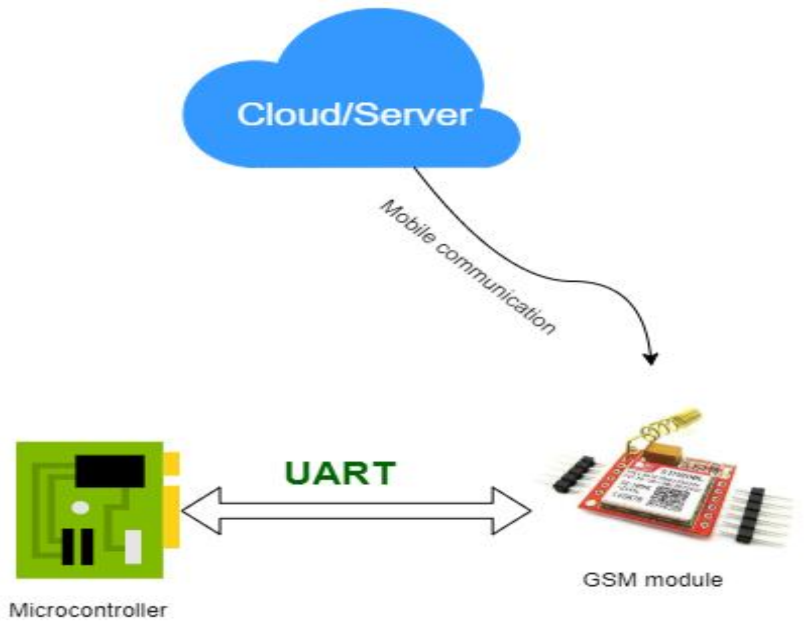


Figure 37: Cellular network-based implementation scheme

In this implementation cellular communication standards and protocols plays an important role, first generation of mobile networks was only reliant on analog radio system, so user can only perform voice calls over network, user can’t send text or data to another user, 1G was introduced in Japan in 1979. 1G working mechanisms depend on cell towers within the coverage area, this network suffers from signal unreliability and security drawbacks.

2G replaced 1G with digital signals instead of analog one, 2G introduced to the world a new communication dimension, 2G phone calls are encrypted and mobile data transfer is provided by 2G network, 2G also provide short message service (SMS) and multimedia message service (MMS), 2G was launched in Finland in 1991.

The Global system for mobile communications (GSM) is a standard developed by the European Telecommunication standard institute (ETSI), 2G digital cellular networks protocols launched over GSM, GSM stands as 2G standard protocol for many years and by 2010 it became a global standard for mobile communications with 90% of market share over a lot of countries. GSM introduce a digital, circuit-switched network optimized for full duplex mobile telephony. By the time data communication is added to GSM standard, data communication started with circuit-switched transport, then by packet data transport over General Packet Radio Service (GPRS) and

Enhanced Data Rates for GSM Evolution (EDGE), GSM operating frequencies laying in 900 MHz or 1800 MHz bands, GSM is considered to be secure wireless system, it has authentication method by using a pre-shared key and challenge-response. General Packet Radio Service (GPRS) is a packet oriented mobile data standard on 2G and 3G cellular communication network's Global System for mobile communications (GSM), GPRS also introduced by European Telecommunications Standards Institute (ETSI). GPRS is best-effort service, so it has variable throughput and latency depending on the number of users sharing the service simultaneous. GPRS provide data rates of 56 – 114 kbit/sec for 2G systems, GSM modules and GPRS modules can be integrated within any electrical or electronic device, it considered as an embedded hardware. Frequency division duplex (FDD) and time division multiple access are multiple access methods used in GSM with GPRS. [8][9]

3G technology support a higher date rates than the previous technologies with at least 144 kbits/s and up to several Mbits/s, then 3.5G introduced to achieve better performance for mobile telephony and data technologies and then 3.75G introduced till 4G is reached.

4G technology introduced to us mobile web access, IP telephony, online gaming services, high-definition mobile TV, video conferencing and 3D television, International Telecommunications Union specified 4Gpeak speed at 100 Mbit per second for high mobility applications and 1 gigabit per second for low mobility communication. [10]

5.3. Our Solution

After reviewing related FOTA work done previously we chose to build our FOTA over cellular network to achieve reliability, good user experience and security, our communication system is built over GSM standards by using SIM800L module.

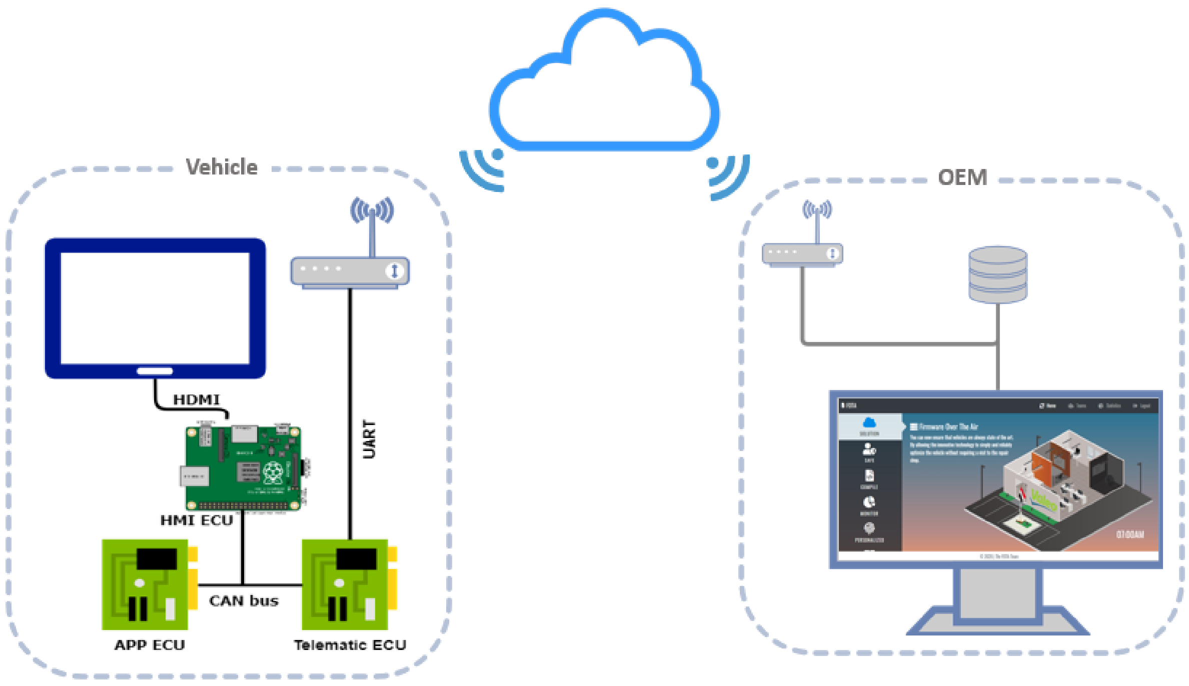


Figure 38:FOTA system

5.3.1. GSM services

GSM standard support several types of services, first service is teleservices at which a bearer service is used by a Teleservice to transport data such as voice calls, emergency calls, video calls and short text messages. Second service is bearer services at which data services are used through GSM module, sending and receiving is the essential building block which lead to widespread of mobile internet access and mobile data transfer. Third service is supplementary services include caller identification, call forwarding, call waiting, multi-party conversations, and barring of outgoing (international) calls. [11]

5.3.2. Accessing GSM Network

Three things are needed to access a GSM network:

- Billing policy with the mobile phone operator.
- GSM module compliant and operates at same frequency as the operator.
- A subscriber identity module (SIM card) [12]

5.3.3. Internet access using GSM

The following figure illustrates conventional GSM network accessing, Mobile node with pc card emulates a modem connected to the handset in order to use that modem-based software, The GSM data provider has a modem pool located in the Mobile switching center, There are three different networks between mobile client to a fixed server which are GSM, PSTN and Internet. [8]

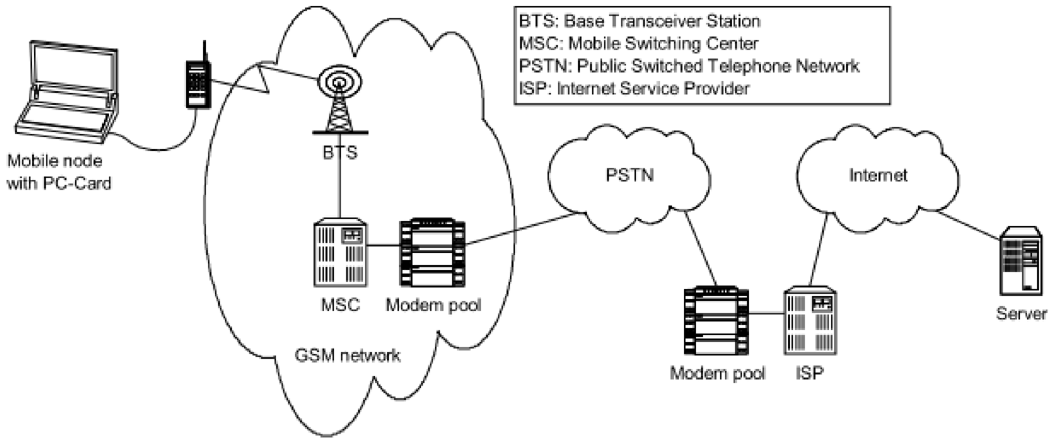

BTS: Base Transceiver Station
MSC: Mobile Switching Center
PSTN: Public Switched Telephone Network
ISP: Internet Service Provider
Mobile node with PC-Card
BTS
MSC
Modem pool
GSM network
PSTN
Internet
Modem pool
ISP
Server

Figure 39: Conventional access to Internet through GSM networks [13]

5.3.4. GSM module

There’s a lot of GSM modules in the market with different antennas and different development boards, SIM800L GSM module is used in our FOTA system. SIM800L is a quad-band GSM/GPRS module, GSM850MHz, EGSM900MHz, DCS1800MHz and PCS1900MHz. SIM800L features GPRS multi-slot class 12/ class 10 and supports the GPRS coding schemes CS-

1, CS-2, CS-3 and CS-4. SIM800L can meet almost all the space requirements in user applications, such as smart phone, PDA and other mobile devices. [14]

5.3.5. SIM800L interfacing

STM32 communicate with SIM800L over UART communication protocol through transmitter and receiver pins in STM32 and SIM800L.

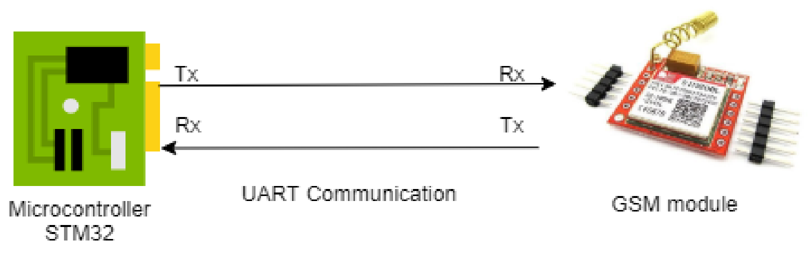


Figure 40: SIM800L interfacing with STM32

5.3.6. AT commands

AT commands are series of short text string which combined to produce useful commands to make a certain operation on the target, these commands are classified according to their functionality. STM32 control SIM800L through sending AT commands and read SIM800L responses through these commands.

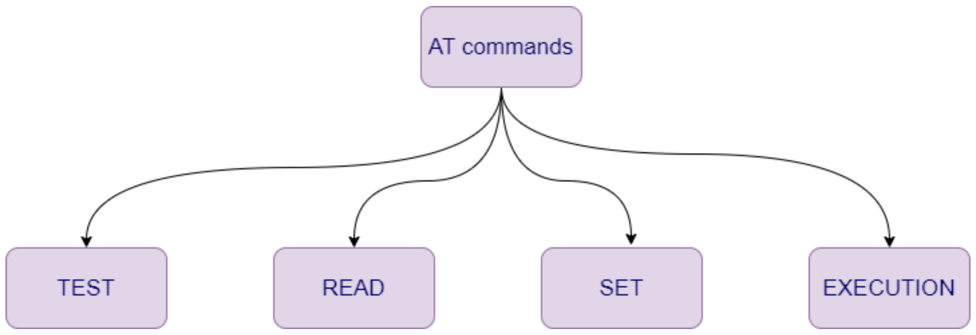


Figure 41:Classification of AT commands

After system starts, STM32 tests the connection validity with SIM800L with a test command and the response of the command proves if the SIM800L running correctly or there's an error, then

STM32 starts to set SIM800L configurations with a bunch of supported Set commands, then STM32 send some execution commands to start connection with server depending on the setting configurations, then it reads server response with read commands. The following example illustrate HTTP POST request needed commands.

Table 1: HTTP POST request commands

Command	**Expected response**	**function**
at+sapbr=3,1,"contype","gprs"	'OK' or 'ERROR'	Bearer settings for applications based on IP
at+sapbr=1,1	'OK' or 'ERROR'	Open bearer
AT+HTTPINIT	'OK' or If error is related to ME functionality: '+CME ERROR: <err>'	Initialize HTTP service
AT+HTTPSSL=1	'OK' or 'ERROR'	Enable HTTPS
AT+HTTPPARA="CID",1	'OK' or 'ERROR'	Set HTTP Parameters Value
AT+HTTPPARA="REDIR",1	'OK' or 'ERROR'	Set HTTP Parameters Value
AT+HTTPPARA="content","application/json"	'OK' or 'ERROR'	Set HTTP Parameters Value
AT+HTTPPARA="URL","34.65.42.223/firmware/v/5e6e8d5a027c8961656183ca"	'OK' or 'ERROR'	Setting request URL
AT+HTTPDATA=100,10000	'Download …. OK' or 'ERROR'	In case of post request, this command is used to insert request data, first

		parameter is the number of the data symbols and the second parameter is the command timeout. The user shoud wait after reading 'DOWNLOAD' from module response then send request data then wait to receive 'OK'
AT+HTTPACTION=1	'OK' or 'ERROR'	Used to perform the request, its reposnse carry Error status either if the request succeed or failed
AT+HTTPREAD	'OK' or 'ERROR'	Used to read the response payload

5.3.7. Power system

One of the most challenging parts in main (telematic) ECU is the power circuit, as SIM800L has a lot of power constrains to operate well, SIM800L is powered up with voltage range between 3.8v and 4.2v, the typical voltage is 4.1v, and current could reach 2A at some special situations, power system stability is considered as critical part of SIM800L, instability in power system cause the SIM800L to reset and sending unpredictable messages which in order will disturb the system. LM2596S DC-DC Adjustable Step-Down Module is used in our system to produce 4.1v to feed SIM800L.

Figure 42:Adjustable power supply

5.3.8. Software Architecture

Main ECU software is following layered architecture pattern, it consists of 4 main layers which are microcontroller abstraction layer (MCAL), hardware abstraction layer (HAL), application layer (APP) and operating system layer.

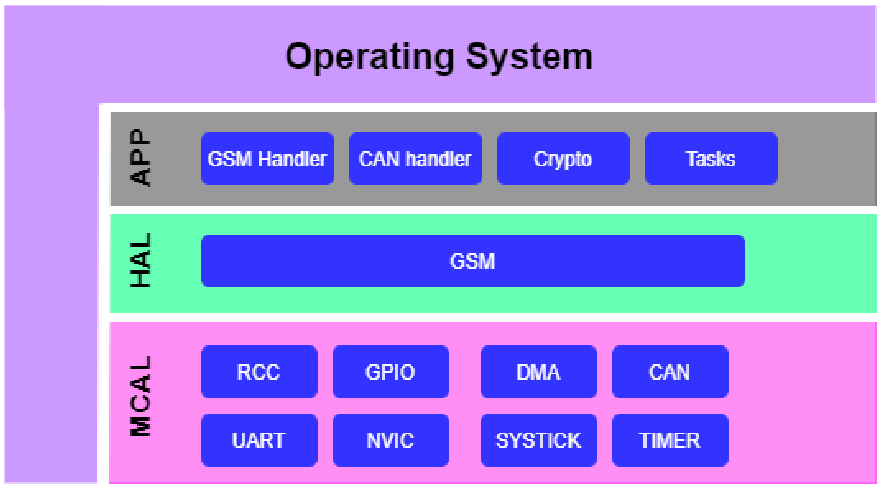


Figure 43: Main ECU software architecture

5.4. Microcontroller abstraction layer (MCAL)

MCAL layer is responsible to provide APIs for all microcontroller hardware peripherals.

- RCC
  This peripheral function is to set microcontroller clock and reset configurations.
- GPIO
  This peripheral allows us t interact with microcontroller output and input pins through memory mapped registers.
- UART
  The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator.
- NVIC
  This peripheral allows us to program, control and configure all interrupts options.
- DMA
  Direct memory access (DMA) is used in order to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions. This keeps CPU resources free for other operations.
- CAN
  The Controller Area Network (CAN bus) is the nervous system, enabling communication with high speed. 'nodes' or 'electronic control units' (ECUs) are like parts of the body, interconnected via the CAN bus. Information sensed by one part can be shared with another.

- SYSTICK
  The System Tick Time (SysTick) generates interrupt requests on a regular basis. This allows an OS to carry out context switching to support multiple tasking. For applications that do not require an OS, the SysTick can be used for time keeping, time measurement, or as an interrupt source for tasks that need to be executed regularly.

- Timer

  It provides normal timer functionality.
- SCB (System Control Block The System control block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. It is used to set the interrupt priority of the Systick Interrupt.

5.5. Hardware abstraction layer (HAL)

- GSM API

GSM software component is considered one of the system HAL layer components, GSM software component responsible for initializing SIM800L by setting all the needed configurations, performing web requests and handle their responses.

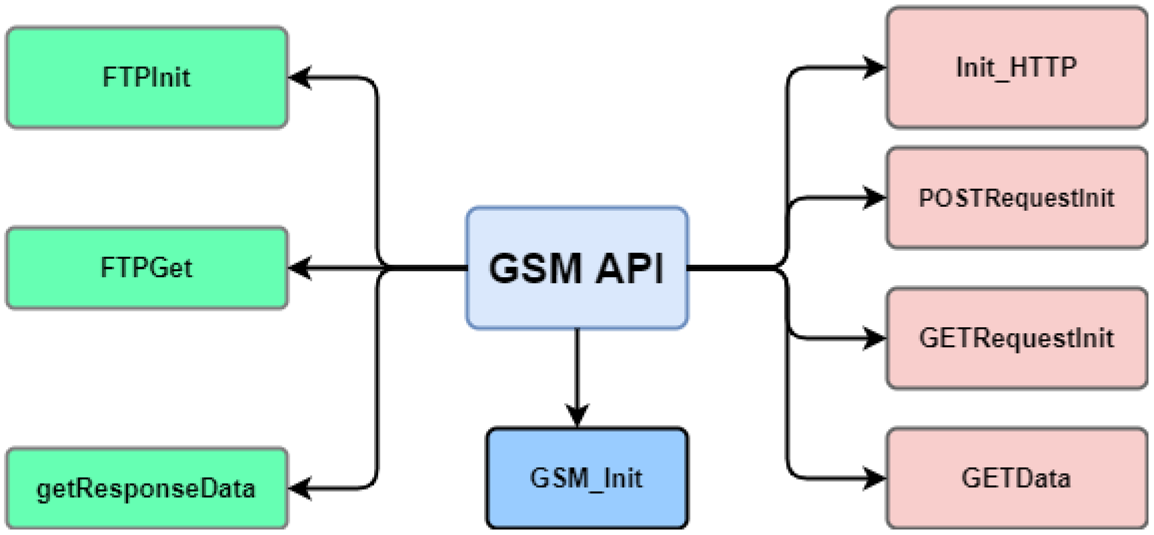


Figure 44: GSM API

1. GSM_init function: responsible to test the connection between STM32 and SIM800L, if the connection is established then proceed to next operations and if there’s a problem in connection with SIM800L it will return error status, then setting the communication network parameters to enable internet connection.
2. Init_HTTP function: it used to open HTTP connection and configures it.

3. POSTRequestInit function: it is used to send a post request; it’s required to supply this function with the URL of the request and post request data.
4. GETRequestInit: it used to perform a get request, it’s required to provide the URL to this function.
5. GETData: it’s responsible to extract the useful data out of the HTTP response frame.
6. FTPInit: Send FTP request and dynamically receives the response in a buffer.
7. FTPGet: Execute FTP get requests.
8. getResponseData: extract the file data from the FTP response frame.

5.6. Application layer

- **GSM handler**

  GSM Handler handles the commands sent to the GSM in order to receive the hex fileor send the diagnostics frame.

  1. GSMHANDLER_vidTask: Periodic Task developed as a state machine of the GSM as shown in Figure 24 starting from state DisbaleEcho.
  2. GSMHANDLER_vidStartDiag: Starts the diagnostics session if the GSM isn’t busy with an update, it sends a CAN Message stating the GSM is Busy to the GUI otherwise.
  3. vidSendCommand: A private function called by the GSMHANDLER_vidTask, it has two states: IDLE meaning a message is going to be sent through DMA and it proceeds to the next state, WaitingForMessage which waits for an OK or an ERROR response from the GSM.
  4. vidSendHTTPData: Send the AT Command for HTTPDATA. It has the same two states of the vidSendCommand but it waits for response DOWNLOAD or ERROR instead of OK or ERROR.
  5. vidSendVehicleName: Sends the vehicle name and password. It also adds the DTC array to the parameters in case of Diagnostics Session. It has the same two states if the vidSendCommand function and waits for response OK or ERROR.

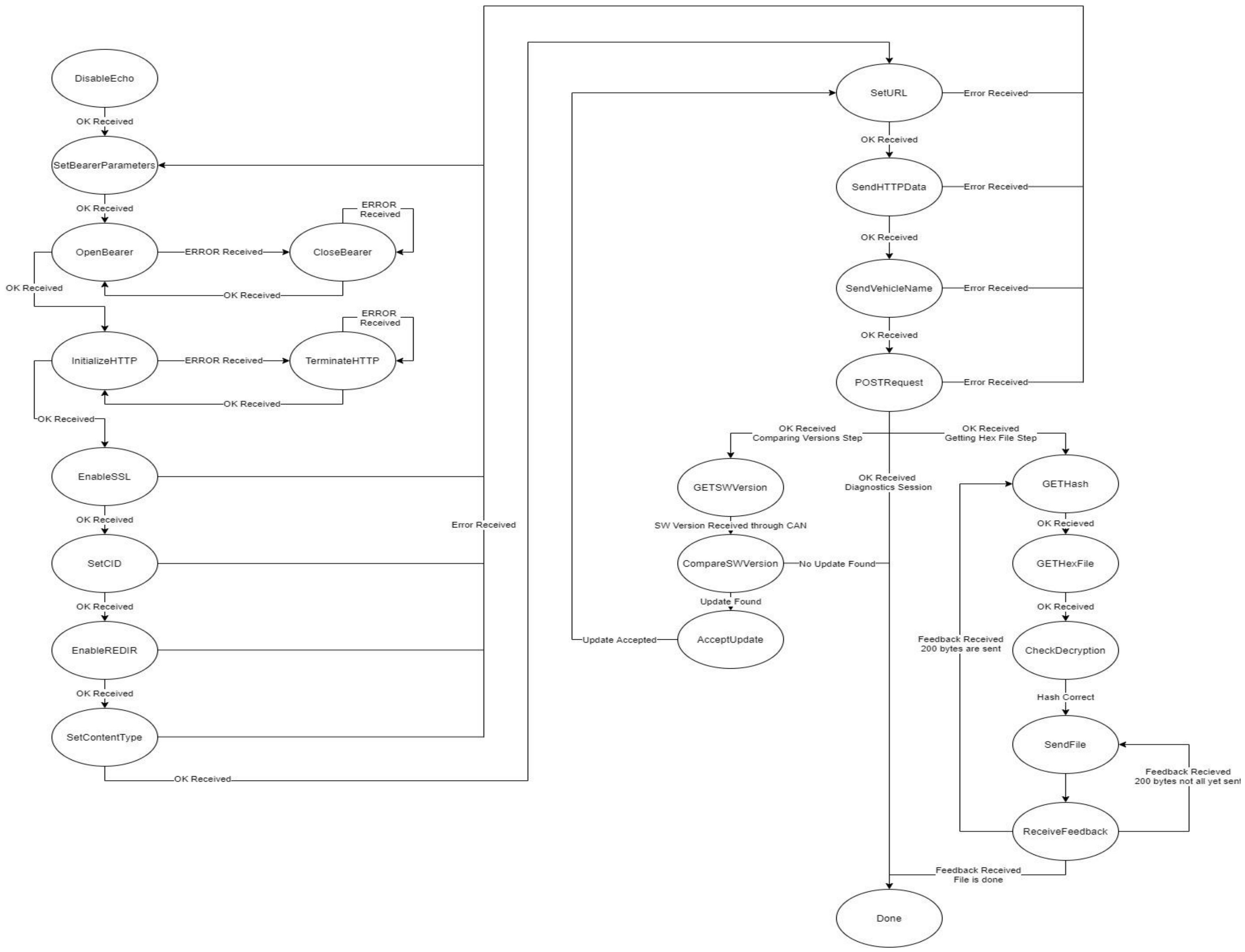


Figure 45: GSM Handler Task State Machine

6. u32SendPostRequest: Sends post request command AT+HTTPACTION=1. It has the same two states of vidSendCommand Function and waits for the command response.
7. u32SendPostRequest: Sends post request command AT+HTTPACTION=1. It has the same two states of vidSendCommand Function and waits for the command response.

8. u16GETData: Send AT+HTTPREAD command and extracts the data from the file read by the GSM and returns the data length.
9. vidCheckUpdate: Compares the Software versions to check if an update is available.
10. vidSendHexFile: Sends the hex file through CAN.
11. vidDMAIRQ: Interrupt Handler of UART Sending DMA Channel. It sets allowing any of the Sending Commands Functions to check for the received response.
12. u8CheckBufferTermination: Searches for "\r\n" in the UART Response.

11. enuFindString: Searches for a specific string in a character array.
12. vidClearBuffer: Sets the elements of an array to zeros.

- **CAN handler**

  It handles the received and sent CAN Messages.
  1. CANHANDLER_vidSend: Sends a CAN Message.
  2. CANHANDLER_vidSendTask: A periodic task to send CAN Messages that couldn't be sent using the Send function because the mailboxes are full.
  3. CANHANDLER_vidReceive: Periodic Function Handling the received CAN Messages.

- **Crypto**

  Crpto software component is responsible to perform different cryptographic algorithms such as HMAC, HASH, RSA and AES.
- **Tasks**

  Tasks software components is responsible to handle all application layer functions such as extract the useful information from server response, make comparisons and take decisions.

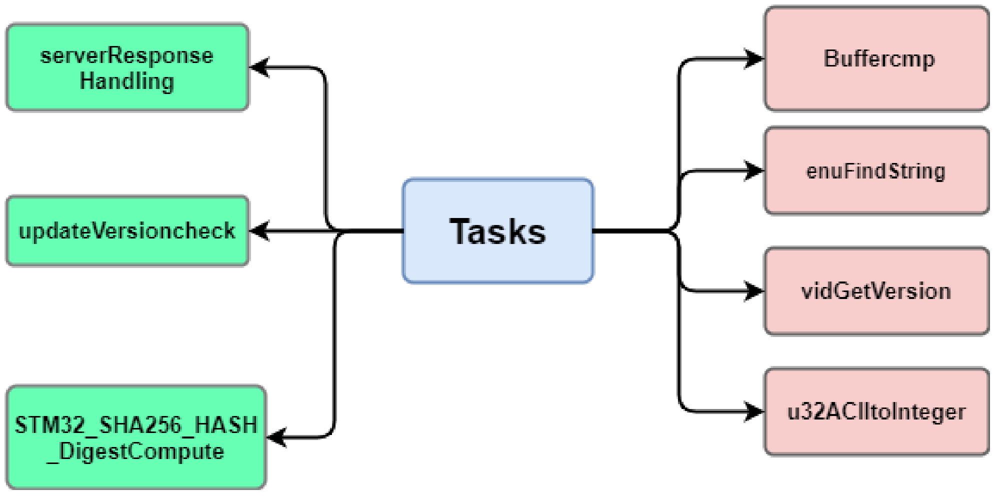


Figure 46: TASKS software component

1. serverResponseHandling is responsible to check update request status in Server response.
2. UpdateVesioncheck is responsible to compare a certain file version on the server and the existing version on a certain ECU.
3. STM32_SHA256_HASH_DigestCompute, It compute HASH digest code according to SHA256(Security stack).
4. Buffercmp used to compare expected output which is extracted from server response with the generated Hash digest code.
5. enuFindString used to find the expected string in the buffer.
6. vidGetVersion used to convert version ID from string form to major, minor and patch values.
7. u32ACIItoInteger used to convert array of ASCII number to its integer value.

5.7. Operating system

The operating system is made of a co-operative scheduler with a static priority for the tasks and a tick time of 10 ms. The system has 3 tasks arranged according to priority: GSMHANDLER_vidTask, CANHANDLER_vidReceive and CANHANDLER_vidSendTask, each of them periodic with a period of 10 ms.

The scheduler is based on a Systick interrupt, but since the CAN Driver is based on interrupts, and the priority of Systick Interrupt is initially higher than the CAN Rx and TX Interrupts priorities, a change of interrupt priorities is done to allow nesting from the scheduler to the CAN interrupts. The DMA Interrupt is also used in the system, but since its function isn’t time critical, its interrupt priority is left to its default value.

Using the NVIC Driver, the priority groups of both the CAN Interrupts are set to Group 0 and subgroup 3 to prevent them from nesting each other, and using the SCB Driver, the Systick Interrupt is set to Group 1 and subgroup 0, allowing nesting for the CAN Interrupts which have a higher priority group.

Chapter 6: Application ECU

6.1. Introduction

The Application ECU is the ECU that is running an application and going to be updated. The flash memory is typically divided into three sections: Branching Code, Bootloader Section and Application Section. The branching code can be an individual section or a part of the bootloader section, which is the case in our project. We will consider each of these sections in detail.

6.2. Bootloader

6.2.1. Introduction

Bootloader is a software code that runs on a microcontroller that needs to be programmed. The purpose of a bootloader is to flash or burn a new software on a microcontroller. As an example, when programming any controller, it should be connected to a PC where the PC flashes the hex file to the Flash Memory of the controller. A bootloader typically does the same job of the PC, it receives the hex file through any communication protocol as UART, SPI, CAN, etc and stores it in the Flash memory, usually using the Flash Controller Peripheral of the microcontroller.

6.2.2. Flash Controller Overview

To understand the importance of Flash Controller, we need to understand the theory of operation of flash memory.

Flash memory, also known as flash storage, is a type of nonvolatile memory that erases data in units called blocks and rewrites data at the byte level. Flash memory architecture includes a memory array stacked with a large number of flash cells. A basic flash memory cell consists of a storage transistor with a control gate and a floating gate, which is insulated from the rest of the transistor by a thin dielectric material or oxide layer. The floating gate stores the electrical charge and

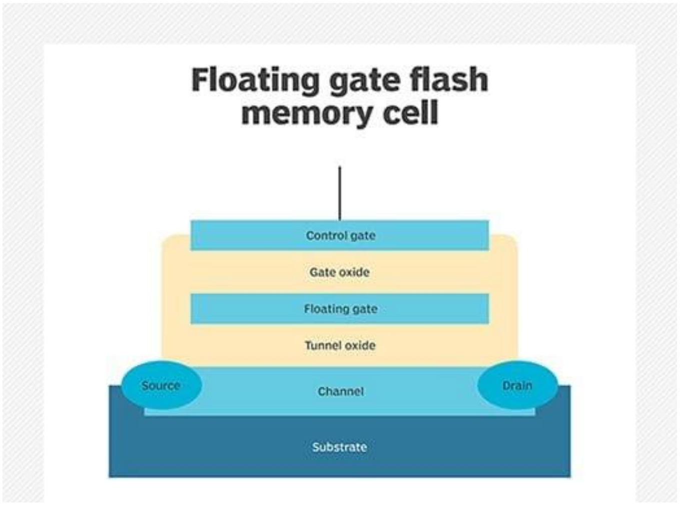


Figure 47: Floating gate flash Memory Cell

controls the flow of the electrical current. Electrons are added to or removed from the floating gate to change the storage transistor's threshold voltage. Changing the voltage affects whether a cell is programmed as a zero or a one. [15]

The process of moving electrons from the control gate and into the floating gate is called Fowler–Nordheim tunneling, and it fundamentally changes the characteristics of the cell by increasing the MOSFET's threshold voltage. This, in turn, changes the drain-source current that flows through the transistor for a given gate voltage, which is ultimately used to encode a binary value. The Fowler-Nordheim tunneling effect is reversible, so electrons can be added to or removed from the floating gate, processes traditionally known as writing and erasing. [16]

Thus, the process of programming (writing or erasing) flash memory needs high voltages. The hardware flashers used to flash a chip usually have the ability to produce such high voltages, but in order to flash the microcontroller through software without the use of an external circuit, an internal circuit providing this high voltage should be present internally in the microcontroller, this circuit is controlled using the Flash Controller.

6.2.3. Flash Memory Module in STM32F103C8T6

The microcontroller used in the project is a medium-density device with a 128KB flash memory with every 1KB as a page i.e. flash memory has 128 pages of memory starting from memory address 0x0800000 to memory address 0x0801FFFF.

The Information block in the flash memory starts from memory address 0x1FFFF000. It has 2KB for System Memory, which contains the standard bootloader from STM32, followed by 2 bytes for Option Bytes. Option Bytes are 2 non-volatile data bytes in the flash memory reserved for user use. They aren’t erased after reset which can be handy in storing information without the need of an external EEPROM.

Flash controller registers are present in the memory starting from address 0x40022000 to address 0x40022023.

The memory organization is shown in Table 2.

Table 2: Flash module organization

Block	Name	Base Addresses	Size (bytes)
Main Memory	Page 0	0x0800 0000 - 0x0800 03FF	1 KBytes
	Page 1	0x0800 0400 - 0x0800 07FF	1 KBytes
	Page 2	0x0800 0800 - 0x0800 0BFF	1 KBytes
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 KBytes
	Page 4	0x0800 1000 - 0x0800 13FF	1 KBytes
	. . .		
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 KBytes
Information Block	System Memory	0x1FFF F000 - 0x1FFF F7FF	2 KBytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash Memory Interface Registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4

	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

The standard STM32F103C8T6 Bootloader supports several communication protocols, however it doesn’t support CAN Bootloader, so a custom bootloader in the program memory is developed to flash the program in the memory. [17]

6.2.4. Intel Hex Format

Now that we understand the use of the Flash Memory Controller, the next step is to study the hex file format to be able to flash the program in the flash memory.

Intel hexadecimal object file format, Intel hex format or Intellec Hex is a file format that conveys binary information in ASCII text form. It is commonly used for programming microcontrollers, EPROMs and other types of programmable logic devices. In a typical application, a compiler or assembler converts a program's source code (such as in C or assembly language) to machine code and outputs it into a HEX file. Common file extensions used for the resulting files are. HEX or .H86. The HEX file is then read by a programmer to write the machine code into a PROM or is transferred to the target system for loading and execution.

Intel HEX consists of lines of ASCII text that are separated by line feed or carriage return characters or both. Each text line contains hexadecimal characters that encode multiple binary numbers. The binary numbers may represent data, memory addresses, or other values, depending on their position in the line and the type and length of the line. Each text line is called a record.

The Record structure is 6 parts as follows:

1- Start code, one character, an ASCII colon ':'.
2- Byte count, two hex digits (one hex digit pair), indicating the number of bytes (hex digit pairs) in the data field. The maximum byte count is 255 (0xFF). 16 (0x10) and 32 (0x20)

are commonly used byte counts. 16 is the most used in the hex files used in our project, however other byte numbers may also be found.

3- Address, four hex digits, representing the 16-bit beginning memory address offset of the data. The physical address of the data is computed by adding this offset to a previously established base address, thus allowing memory addressing beyond the 64 kilobyte limit of 16-bit addresses. The base address, which defaults to zero, can be changed by various types of records. Base addresses and address offsets are always expressed as big endian values.

4- Record type, two hex digits, 00 to 05, defining the meaning of the data field. The record types are explained in details after Record Structure.

5- Data, a sequence of n bytes of data, represented by 2n hex digits. Some records omit this field (n equals zero). The meaning and interpretation of data bytes depends on the application.

6- Checksum, two hex digits, a computed value that can be used to verify the record has no errors.

As mentioned, Record type may have a value from **00** to **05** as follows:

1- **00:** Data. Contains data and a 16-bit starting address for the data. The byte count specifies number of data bytes in the record. Example:
:10280000000500020212A0008272A00082B2A00084F
This line contains hex data: 0x02000500, 0x08002A21, 0x08002A27, 0x08002A2B respectively. Their memory locations are determined according to the Extended Linear Address record before them.

2- **01:** End of File. It must occur exactly once per file in the last line of the file. The data field is empty (thus byte count is 00) and the address field is typically 0000. Example:
:00000001FF

3- **02:** Extended Segment Address. The data field contains a 16-bit segment base address (thus byte count is always 02) compatible with 80x86 real mode addressing. The address field (typically 0000) is ignored. The segment address from the most recent 02 record is multiplied by 16 and added to each subsequent data record address to form the physical starting address for the data. This allows addressing up to one megabyte of address space. This type of record is not used in the hex files used in the project.

4- **03:** Start Segment Address. For 80x86 processors, specifies the initial content of the CS (code segment): IP (instruction pointer) registers (i.e., the starting execution address). The address field is 0000, the byte count is always 04, the first two data bytes are the CS value, the latter two are the IP value. Since the startup code manages the memory sections and code segments and the bootloader manages the execution start address of the application, this record is also not present in the hex files used in the project.

5- **04:** Extended Linear Address. Allows for 32 bit addressing (up to 4GiB). The record's address field is ignored (typically 0000) and its byte count is always 02. The two data bytes (big endian) specify the upper 16 bits of the 32 bit absolute address for all subsequent type 00 records; these upper address bits apply until the next 04 record. The absolute address for a type 00 record is formed by combining the upper 16 address bits of the most recent 04 record with the low 16 address bits of the 00 record. If a type 00 record is not preceded by any type 04 records then its upper 16 address bits default to 0000. This record is usually the first record in the hex file; however, it can be repeated anywhere in the hexfile if the base address if the data is changed. Example:

:020000040800F2

In this record, the address is 0x0800, which is concatenated to the address in the data record to get the final address. For example, in the previous data record example: :10280000000500020212A0008272A00082B2A00084F

The data will be stored starting from address 0x08002800.

6- **05:** Start Linear Address. The address field is 0000 (not used) and the byte count is always 04. The four data bytes represent a 32-bit address value (big-endian). [18] It represents the start address of the program. Example:

:040000050800290DB9

The start address of this program is 0x0800290D.

6.2.5. Our Solution

6.2.5.1. Flash Memory Sections.

The main flash memory starting from address 0x0800000 to 0x0801FFFF should typically contain three sections: Branching Section, Bootloader Section and Application Section. The branching section is used to jump to either the application or the bootloader and it can be a part of the bootloader or has an individual section in the Flash Memory. In this project, the branching code is a part of the bootloader section, thus, the main flash memory has two sections: Bootloader section and Application Section.

Instead of having only one application section, the application section in the project contains two banks; the reason for this is that as the flash memory must be erased before writing any data to it, if only one application bank is present then if any error or interruption of communication occurs during the flashing of the file, the ECU would stop its current functionality completely. For this reason, the application section is divided into two banks with one active bank which becomes a back-up in case of error, and another bank for programming the new software.

Therefore, the 128KB flash memory is divided as follows: 10KB at the start of the main memory is reserved for the bootloader starting from address 0x0800000 to address 0x080027FF, followed by 59KB for the first Application Bank starting from address 0x08002800 to address 0x08013FFF then 59KB for the second Application Bank starting from address 0x08014000 to address 0x0801FFFF.

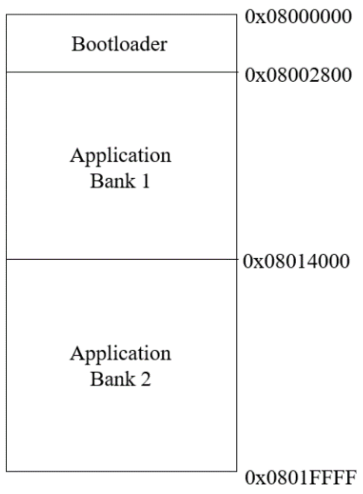


Figure 48: Flash Memory Sections

6.2.5.2. Flash Option Bytes

As mentioned before, the option bytes are 2 non-volatile data bytes in the flash memory for user use. Since Option Bytes are a part of the flash memory, they cannot be programmed or erased directly, but only through the flash controller module. The erase sequence is as follows:

1- Check that no Flash memory operation is ongoing by reading the BSY bit in the FLASH_SR register.
2- Unlock the OPTWRE bit in the FLASH_CR register.
3- Set the OPTER bit in the FLASH_CR register.
4- Set the STRT bit in the FLASH_CR register.
5- Wait for BSY to reset.
6- Read the erased option bytes and verify.

And the programming procedure is as follows:

1- Check that no Flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register.
2- Unlock the OPTWRE bit in the FLASH_CR register.
3- Set the OPTPG bit in the FLASH_CR register
4- Write the data (half-word) to the desired address
5- Wait for the BSY bit to be reset.
6- Read the programmed value and verify.

The Option Bytes should also be erased before any write operation. The flash controller programs the option bytes through a register and the new programmed value is only programmed after the next system reset.

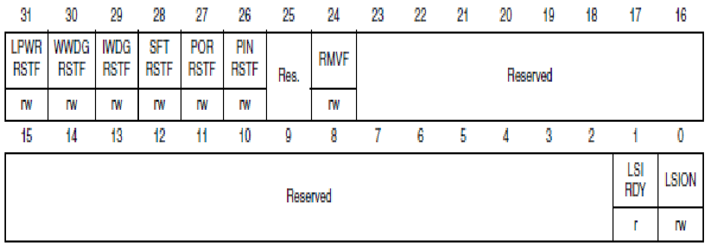
The Option Bytes Data0 Byte is used in our project to determine which application bank is currently on use, and thus determine which one is the one used for flashing the new software. Data0 can have one of three values:

1- **0xFF:** This means that the option bytes are not programmed and that there is no application in the application banks.
2- **0x00:** This means that Application Bank 0 is in use.
3- **0x01**: This means that Application Bank 1 is in use.

6.2.5.3. Branching Code

As mentioned previously, the branching code is the software that decides whether to jump to the application or the bootloader. In the project it is considered a part of the bootloader section in the memory.

The branching code in the project is mainly based on soft reset. As in the application, a soft reset is performed when an update request is received through CAN, so the branching decision is taken based on the soft reset. A soft reset is detected by checking the twenty eighth bit in the RCC (Reset and Clock Control) Control/Status Register (RCC_CSR).



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPWR RSTF	WWDG RSTF	IWDG RSTF	SFT RSTF	POR RSTF	PIN RSTF	Res.	RMVF	Reserved							
rw	rw	rw	rw	rw	rw		rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														LSI RDY	LSION
														r	rw

Figure 49: RCC_CSR Register Bits

The branching code sequence is as shown in Figure 50:

1- Get the option bytes and determine which bank is currently in use. If they aren't programmed, the new software is programmed in the first bank.
2- Get the soft reset flag from the RCC_CSR Register.
3- If a soft reset occurred, continue to the bootloader.
4- If no soft reset occurred, check if there is a program in the currently used bank by reading the first 4 bytes of its start address.
5- If a program is present, get the program reset vector address, whose address is stored in the second 4 bytes of the flash memory according to the linker script, and jump to that address by a pointer to function. If no program is present, the code continues to the bootloader.

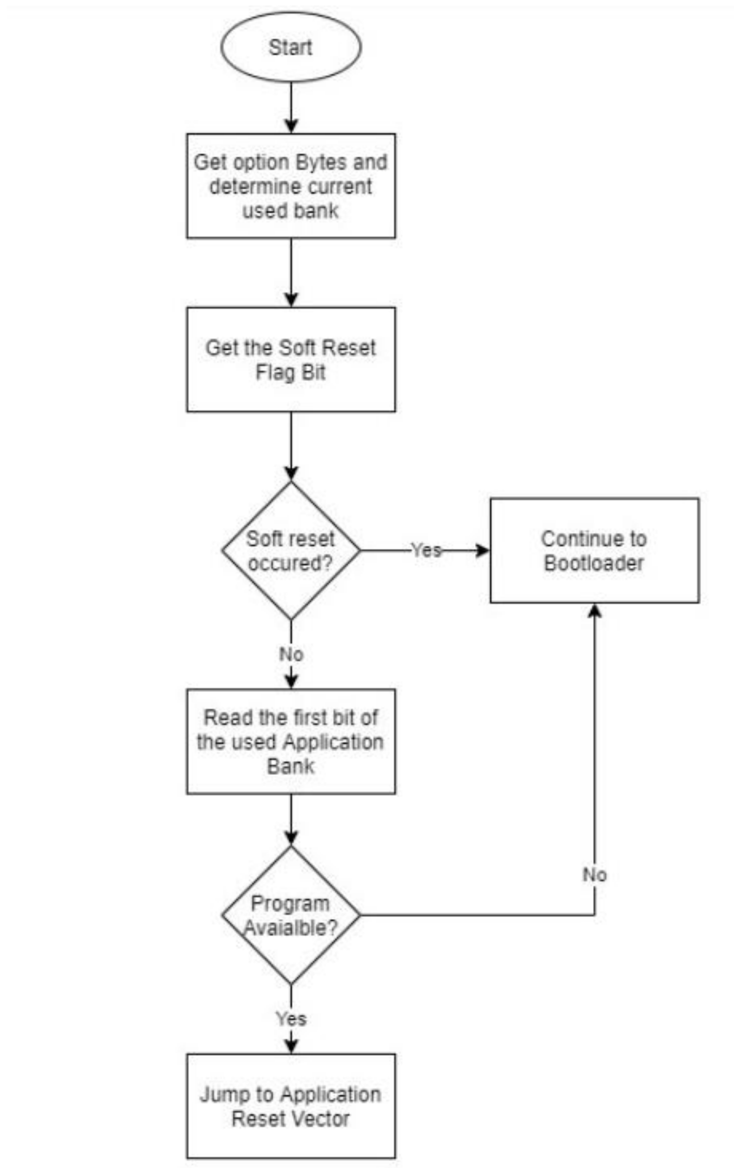

Start
Get option Bytes and determine current used bank
Get the Soft Reset Flag Bit
Soft reset occured?
Yes
Continue to Bootloader
No
Read the first bit of the used Application Bank
Program Avaialble?
No
Yes
Jump to Application Reset Vector

Figure 50: Branching Sequence Flowchart

6.2.5.4. Bootloader Code

The bootloader sequence is always performed after the branching code, so the currently used bank and the programmed bank are known to the software. The code has one main module: HexDataProcessor which processes the hex record, extracts the data and flashes it in the flash memory, in addition to a super loop which handles the received CAN Messages. We will first consider the super loop sequence then discuss the HexDataProcessor Module.

6.2.5.4.1. Bootloader Main Sequence

The bootloader sequence is as shown in Figure 51:

1- Erase the 59 pages of the programming bank as no new data can be programmed unless the banks are erased.

2- Wait for a new CAN message.

3- If a can message is received. It can be one of the four messages in the CAN filter:

   1. **HexField:** This message contains a part of the hex record. Since the CAN messages can carry a maximum of 8 bytes per message, the hex record is received as fragments of 8 bytes then re-combined at the bootloader end in a hex line array. The main ECU is ensured to not send a part of the new line in the CAN Message but terminate the CAN message with the remained number of bytes of the frame.
   2. **SoftwareVersionRequest:** This message is a remote frame asking for the current SW Version on the ECU. Since this message is originally meant to be received by the application code, receiving it in the bootloader sequence means that there is probably no application in the application banks, so a message with version number 0.0.0 is sent with the same CAN Message ID as a data frame, so that when compared to any version from the server, an update is requested.
   3. **UsedBankRequest:** This message is a remote request requesting for the currently used bank for the application. The used bank is sent with the same CAN Message ID as a data frame.
   4. **Request DTCs:** This message is a remote frame asking for the DTCs array. The DTCs array is sent in a CAN Message with the same ID as a data frame.

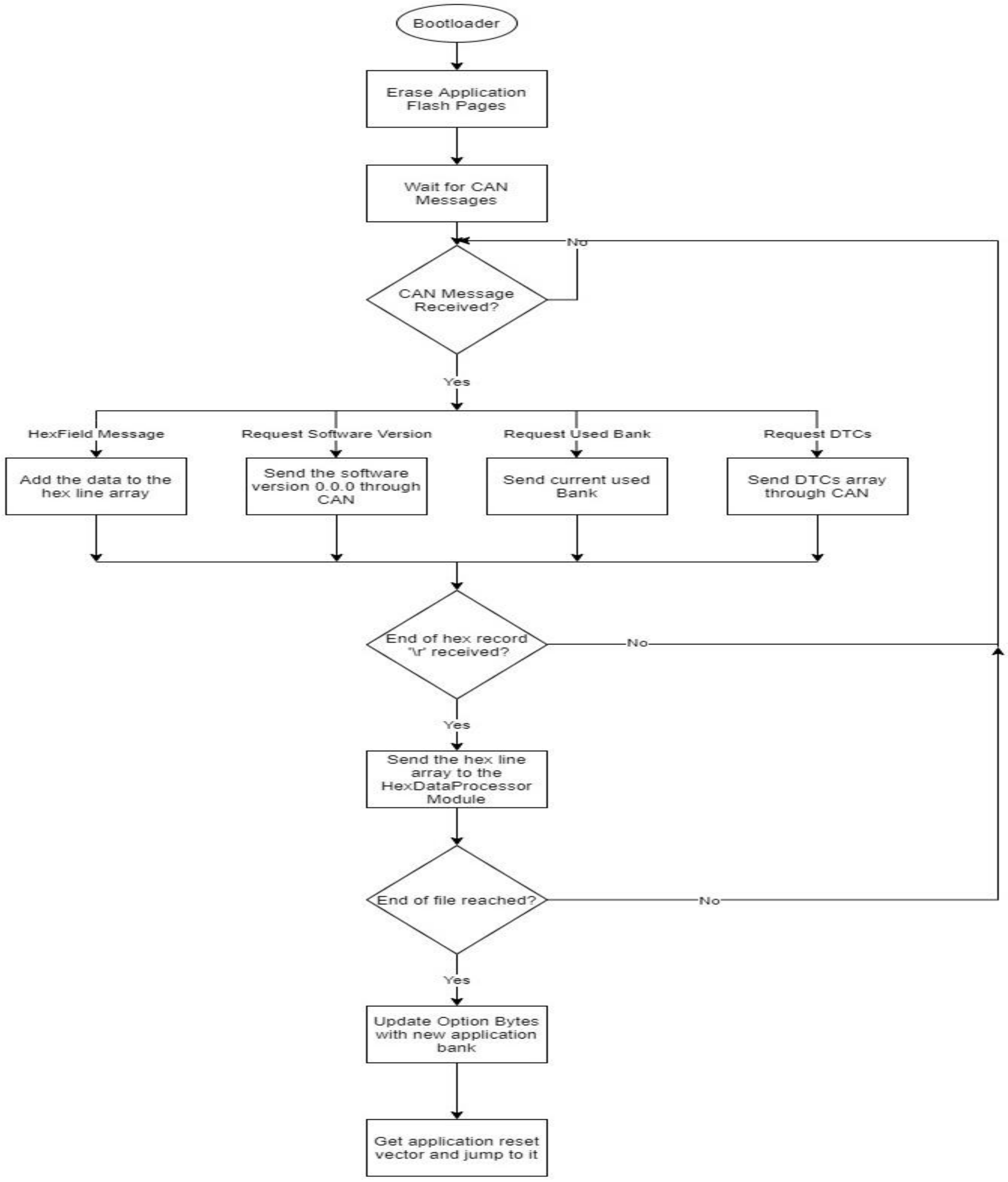

Bootloader
Erase Application Flash Pages
Wait for CAN Messages
CAN Message Received?
No
Yes
HexField Message
Request Software Version
Request Used Bank
Request DTCs
Add the data to the hex line array
Send the software version 0.0.0 through CAN
Send current used Bank
Send DTCs array through CAN
End of hex record '\r' received?
No
Yes
Send the hex line array to the HexDataProcessor Module
End of file reached?
No
Yes
Update Option Bytes with new application bank
Get application reset vector and jump to it

Figure 51: Bootloader Code Flowchart

4- Whether the received CAN Message was a Hex Field message or not, check if the last bit added in the Hex line array is the termination bit '\r', if so, continue to the next step, or else wait for a new CAN Message.

5- Send the Hex Array line to the HexDataProcessor to process the data and flash it in the flash memory and receive the feedback from the module.

6- Check the feedback, if the end of the file is received, continue to the next step, or else wait for a new CAN Message.

7- The whole file is now received, the reset vector address is retrieved using the same procedure of the branching code and the bootloader jumps to it through a pointer to function.

6.2.5.4.2. HexDataProcessor Module

The Hex Data Processor Module is responsible for taking the hex record and extracting the data and addresses and flashing the data in the flash memory. The module is divided into 4 states/functions in order to decrease the code complexity:

1- **Check Validation:**
In this step, the software checks if the hex record is a valid hex record by checking the start code ':' and checking that the number of the characters in the record is an even number and not exceeding the maximum record number.

2- **Convert to Hex:**
Since the hex file is received in ASCII format, this step converts the ASCII format into hexadecinal bytes format.

3- **Parse Line:**
This step parses the hex data into corresponding fields of the record structure into a structure and determine the record type. It also stores the data in a 32-bit array in the structure since the flash memory architecture and flash memory module deals with 32-bit data type. Since the data in the hex line is in big endian format, but the data format in the system is little endian, it rearranges the data as desired. The step of rearranging the

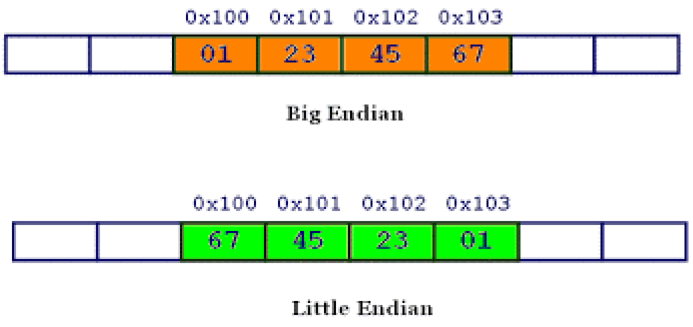


Figure 52: Big endian and Little endian

data is only done in case of a data record as the other records don't need this step. It also adds 0xFF if the data length doesn't complete a 32-bit variable, so that the next record can program the rest of the data. This step also checks the CRC Checksum and returns a NOK in case of error.

4- **Write Data to flash:**
In this step, an action is done according to the record type: in case odd data, 32-bit chunks of data are sent one by one to the flash controller to program them according to the last extended linear address and the address in the data record. In case of extended linear address, the address is stored in a static variable to be used when storing the data.

6.3. Application

As a proof of concept, a simple application is developed using some sensors. The application consists of the following: a parking assistant, a radiator cooler which should work in case of high temperature and a CAN Message handler. The application is developed in form of a super loop. Each if these modules will be discussed in details.

6.3.1. Parking Assistant

The parking assistant consists of two main components: an IR sensor and a piezoelectric buzzer.

Figure 53: IR Sensor

The IR sensor is a digital sensor which sends an IR wave in front of it and senses the medium, if the wave was reflected this means a barrier is in front it, so it returns a high signal on its out pin.

The application procedure is as follows: If a high signal is received, a buzzer is turned on to alert the driver that there is a barrier during parking, otherwise the buzzer is off.

As a proof of concept, the first application has this feature is turned off, after the update, the parking assistant works fine.

6.3.2. Radiator Cooler

The cooler consists of two components: an LM35 temperature sensor and a motor with a 2N2222A transistor as a driver for it.

Figure 54: LM35 Temperature Sensor

The temperature sensor ratings are 1mV/1°C. The controller reads the temperature sensor voltage feedback through a 12-bit ADC Peripheral and converts it into temperature through the following equation:

$$Temperature = \frac{ADC\ Reading}{2^{12}} * 3.3 * 100$$

If the temperature is found to be above a certain limit, a DC Motor should be turned to act as a fan to the radiator. If the temperature exceeds another higher limit, a DTC of value 0x50 which corresponds to High Temperature DTC is added to the DTCs array to be sent to the server in case of diagnostics request.

As a proof of concept, the threshold in the application to turn on the DC Motor is higher than the DTC Threshold and thus a DTC is recorded and the DC Motor isn’t turned. After the software update, the temperature threshold is adjusted and the fan works fine.

6.3.3. CAN Messages Handling

The Application typically receives 4 CAN Messages according to its CAN Filters:

1- **Update Software Request:** This message is sent in case a new update is found on the server and the user accepts the update. When this message is received, the RCC Reset flags are reset and a software reset is performed in order to jump to the bootloader code.

2- **SoftwareVersionRequest:** This message is a remote frame asking for the current SW Version on the ECU. A can message with the same CAN Message ID is sent as a data frame containing the current software version.

3- **UsedBankRequest:** This message is a remote request requesting for the currently used bank for the application. The used bank is read from the option bytes and sent with the same CAN Message ID as a data frame.

4- **Request DTCs:** This message is a remote frame asking for the DTCs array. The DTCs array is sent in a CAN Message with the same ID as a data frame.

Chapter 7: GUI (Graphical User Interface)

7.1. What is GUI?

When operating machines needed an encyclopedic knowledge of code and the inner workings of a program. They had to become more user-friendly for the computers to take off.

The graphical user interface (GUI) has heralded a new age for user interaction with computers and devices. This breakthrough gave people a different way to communicate with systems so they didn't need to know any coding concepts. This destroyed the entry into a previously steep learning curve.

A GUI (the graphical user interface) is a computer software framework with interactive visual elements. An Interface shows objects conveying knowledge, and represents actions that the user may take. When the user communicates with them, the objects change color, size or visibility.

GUI objects include keys, icons, and cursors. Occasionally, these visual elements are enhanced with sounds, or visual effects such as transparency and falling shadows.

The big advantage of a GUI is that systems that use one are accessible to people of all knowledge levels, from a beginner to an advanced developer or other tech-savvy individual. We make it easy for anyone to open menus, transfer files, launch programs or check the internet without having to tell the machine to perform a task through the command line.

Instant feedback is also provided by GUIs. For example, if you click an icon it will open, and that can be seen in real time. Using a command line gui, once you hit return, you won't know if it's a legitimate entry; if it isn't correct, nothing will happen.

7.2. History of GUI

Alan Kay, Douglas Engelbart and other researchers created the first graphical user interface at Xerox PARC in 1981, recognizing that providing a graphical representation of an operating system would make it more available to the masses.

The first commercial use of a GUI occurred in 1983 on the Apple Lisa computer. Until that, machines like MS-DOS and Linux used command-line UIs, so their use was restricted to professional business users rather than customers.

A year later the Apple Macintosh with a GUI became the most common commercial device. Microsoft followed suit with Windows 1.0 in 1985, while Windows 2.0 was a marked change when it was released in 1997. This wasn't until 1995 that Microsoft caught up with Apple's market success in the field of GUI systems and the introduced Windows 95.

7.3. GUI Layers

From the user side he deals with the graphical interface which appears in front of him on a hardware display where icons, gestures and tools which he can choose from all appears on the display server as windows manager in (windows interfaces) which is linked to the operating system (kernel) such Linux then the hardware which is the whole container which the user use it for controlling.

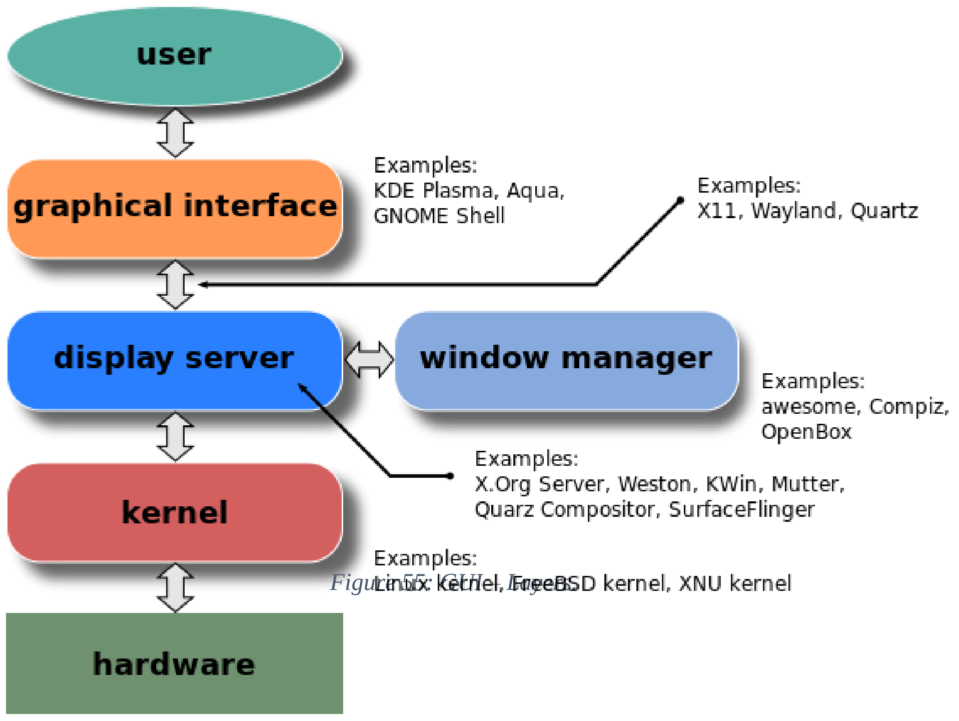


Figure 55: GUI – Layers.

Figure 56: GUI – Layers.

7.4. GUI in cars

Figure 57: GUI in car

Tier 1 and OEM manufacturers understand the need to offer groundbreaking digital instrument clusters, driver information screens, and in-vehicle infotainment systems to keep up with rapidly changing market demands. Storyboard is the only platform that brings together designers and developers, treating them as equals without sacrificing resources or performance to design rich, intuitive in-vehicle experiences.

Automotive human machine interface (HMI) solutions enable drivers to interact with touchpads, multi-touch dashboards, built-in screens, control panels, pushbuttons, and traditional keypads. Since a car is an entire ecosystem of interconnected components, high-quality HMI software is crucial to the automotive industry.

The automotive HMI program renders cars of the next generation versatile and customized using the new technologies. Modern consumers are demanding a seamless experience and HMI solutions can meet the demand for smooth car interactions. Human machine interfaces, enabled by smart systems and embedded sensors, ensure vehicles react to driver's intent and preferences.

The User Experience is all about on-screen GUI (Graphic User Interfaces) in cars. In the latest car models the on-screen user experience could be divided into these categories:

- Entertainment and communication: Music, television, email, internet, third-party applications UI-related. Such features are often accessed through the monitor in the center console...
- The tachometer dashboard UI, alarm, driving assist and other driving-related functions.
- Anything else, such as climate control, general settings and other vehicle settings.

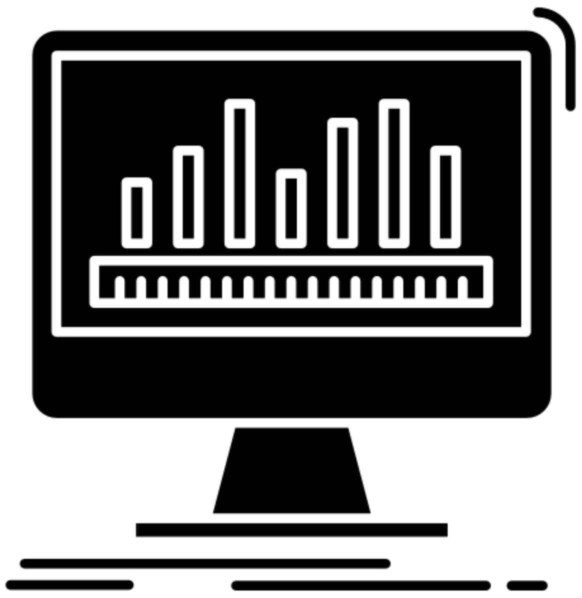


Figure 58: GUI- Applications

Different programming languages are used in developing desktop application and graphical user interfaces for multi-tasking applications, c ++ and Python are mainly used in those kinds, due to the easy interaction with those languages when dealing with heavy duty applications which needs you as developer to be easily able to deal with different kind of complexities.

So with this kind of differences companies have challenged each other around the efficiency of a GUI and the comfort of the user who is handling it.

Apple and Google are both allies and competitors of car companies in this race, and automakers are being increasingly forced to avoid providing their services because customers already have a long history with the ecosystems of those firms. Car companies have recognized that they can deliver excellent digital user experiences by investing in digital design, and maximize the added benefit of deep integration with key driving experience elements.

7.5. GUI in our project

7.5.1. Flow of the GUI in our system

GUI in our project represent to the user all the alerts and the updates needed for the car at which the OEMs send it to the user to be confirmed by the user, all the diagnostics gathered by the ecus in the car is sent by the user to the server after few interactions with the user, let’s take a look on the architecture of the GUI in our system.

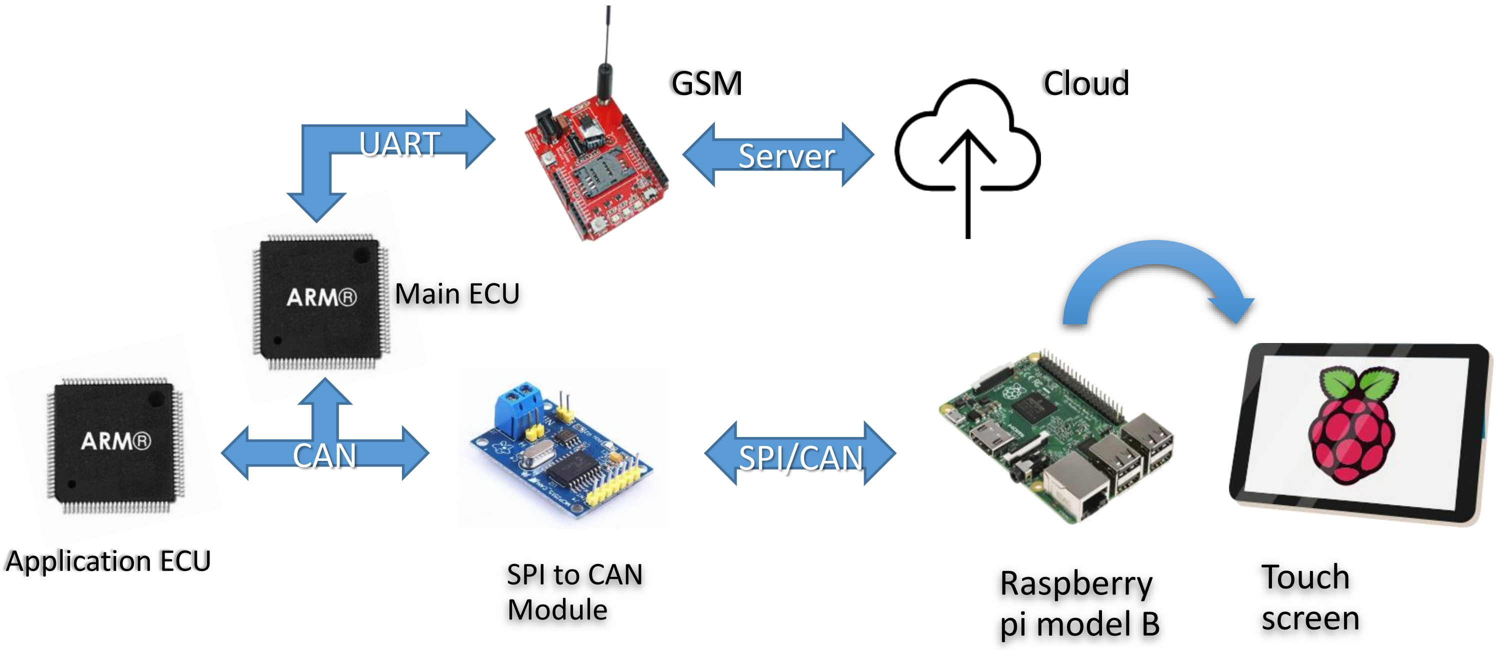


Figure 59: Flow of the GUI in the system.

The main 2 scenarios implemented:

<u>**Case-1:**</u> **FOTA update** where the server sends an update to the user through the **Main ECU** to appear on the touch screen waiting for the confirmation from the user by yes or no. The

communication channel between the **ECU**s in the car gone through CAN where a **SPI** to **can** module is used to convert the signals from the **ECU**s to the control unit of the **Touch screen** which is the **Raspberry pi model B** (which is using **SPI** to communicate).

Case-2: **Diagnostics** are sent from the **ECU**s to appear on the **GUI** system to give the user chance to send it to the server where the OEMs and the manufactures can study those warning, threats or failures and come up with an update for the problem it's a monitoring system.

Now let's talk about the process of implementation, tools, technologies and codes in a specific way.

7.5.2. Why is python used?

The implementation of the GUI is done using Python 3.4.3 on Pycharm 2019 edition IDE. Python used in the design due o it simplicity where you can handle multi modules on the same time using 3 main modules which make it more comfortable to the developer to use it in GUI applications. Most of the python modules used in GUI are built using c/c ++ but using it in python make it easier to handle.

7.5.3. Tkinter vs.PyQt5

I chose to usePYQt5 over Tkinter in my program because of many reasons which I'm going to state:

1- **Stability**: PyQt is used in many large-scale applications and has stood the test of time.
2- **PyQt has a straightforward API** with its classes corresponding to Qt C++'s, and as such, the API documentation for C++ works for Python — the namespaces, properties, methods are all the same. If you have experience working with Qt and/or C++, you will find PyQt easy to work with.
3- **Signal/slot mechanism allows for code flexibility**: GUI programming with Qt is designed around the notion of signals and slots for object communication. When an event occurs (e.g. a button is pressed), a signal is emitted, and slots are callable functions which handle the event (e.g. when a button is clicked, display a pop-up). This provides versatility when dealing with Interface events and results in a cleaner codebase.

4- **Many learning resources available**: PyQt is one of the most popular UI frameworks for Python. It has an active community with many third-party code examples and tutorials available. [19]

Now, let’s discuss the pyqt5 side used in the project.

The 3 main modules which I have used it in the implementation are:

- **QtCore** Module contains the **core non-GUI classes**, including the **event loop** and **Qt's signal and slot mechanism**. It also includes platform independent abstractions for Unicode, threads, mapped files, shared memory, regular expressions, and user and application settings.
  - The **QEventLoop** class provides a means of entering and leaving an event loop.
  - **Signals and slots** are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another.

    For example, if a user clicks a Close button, we probably want the window's close() function to be called.

    Other toolkits achieve this kind of communication using callbacks. A callback is a pointer to a function, so if you want a processing function to notify you about some event you pass a pointer to another function (the callback) to the processing function. The processing function then calls the callback when appropriate. While successful frameworks using this method do exist, callbacks can be unintuitive and may suffer from problems in ensuring the type-correctness of callback arguments.

Signals are emitted by objects when they change their state in a way that may be interesting to other objects. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation, and ensures that the object can be used as a software component.

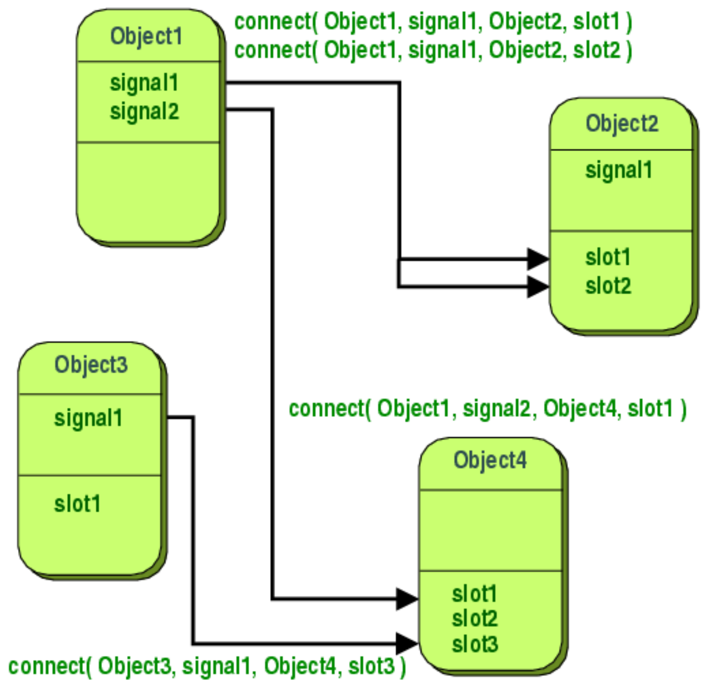

Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

Figure 60: Signals and slots.

Slots can be used for receiving signals, but they are also normal member functions. Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it. This ensures that truly independent components can be created with Qt.

You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need. It is even possible to connect a signal directly to another signal. (This will emit the second signal immediately whenever the first is emitted.)

Together, signals and slots make up a powerful component programming mechanism.

- **QtGui** Module contains the **majority of the GUI classes**. These include a number of **table**, **tree** and **list classes** based on the model–view–controller design pattern. Also provided is a **sophisticated 2D canvas widget** capable of storing thousands of items including ordinary widgets.[20]
- **QtWidgets** Module provides a set of UI elements to create classic desktop-style user interfaces, as for example:

- **Widgets (QWidget)** which are the primary elements for creating user interfaces in Qt.

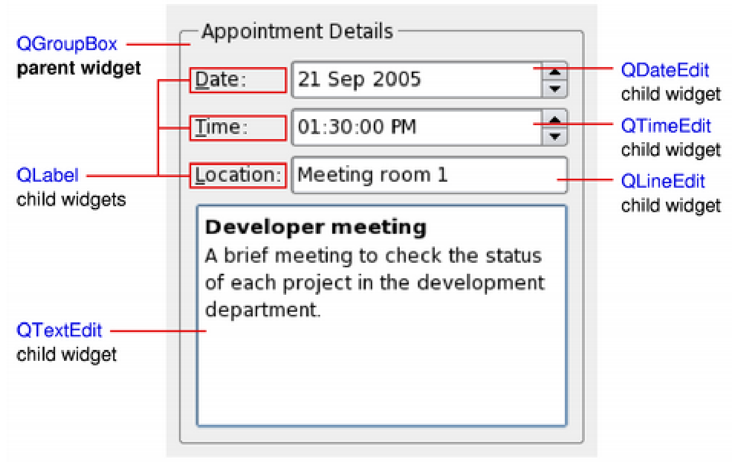


Figure 61: QWidget Interface.

- **Layouts (QLayout)** are an elegant and flexible way to automatically arrange child widgets within their container.

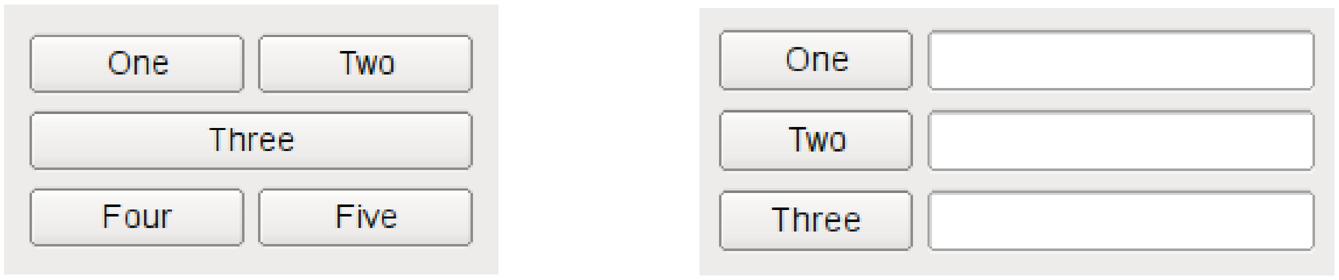


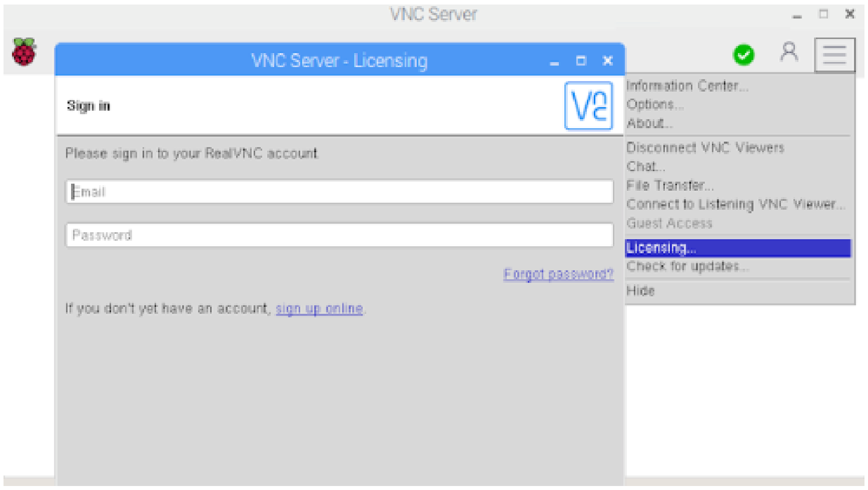
Figure 62: QWidget Layout.

7.5.4. Controller used: Raspberry pi 3 model B

Figure 63: Raspberry pi 3 model B.

The Raspberry Pi hardware has evolved through several versions that feature variations in the type of the central processing unit, amount of memory capacity, networking support, and peripheral-device support, which help in developing different application programming interfaces (APIs), using python and Linux OS with its raspbian system makes it easy to install different libraries and work through different structures.

VNC (Virtual Network Computing) is used in order to remotely control the desktop interface of one computer (running VNC Server) from another computer or mobile device (running VNC Viewer), VNC Viewer transmits the keyboard and either mouse or touch events to VNC Server, and receives updates to the screen in return. We can see the raspbian system on our windows screen on the laptop in order to virtually control it.



Please sign in to your RealVNC account

If you don't yet have an account, sign up online

Figure 65: VNC Server.

7.6. SPI TO CAN Module

We build our GUI system using touch screen connected to raspberry pi so that we can tell the user that there is a new update and ask him to accept the update requests or refuse them and also show the progress of the updating process if the user accepts the update. Because we want all ECUs in our project to be connected to the same CAN network, we use SPI TO CAN module because Raspberry Pi doesn't have a built-in CAN Bus but its GPIO includes SPI Bus, that is supported by large number of CAN controllers. So, we will use a bridge between Raspberry Pi and CAN Bus which is SPI Bus.

SPI TO CAN Module consists of CAN controller (**MCP2515**) because CAN Bus is a multi-master protocol, each node needs a controller to manage its data. And CAN transceiver (**MCP2551**) because CAN controller needs a send/receive chip to adapt signals to CAN Bus levels. Controller and Transceiver are connected by two wires TxD and RxD.

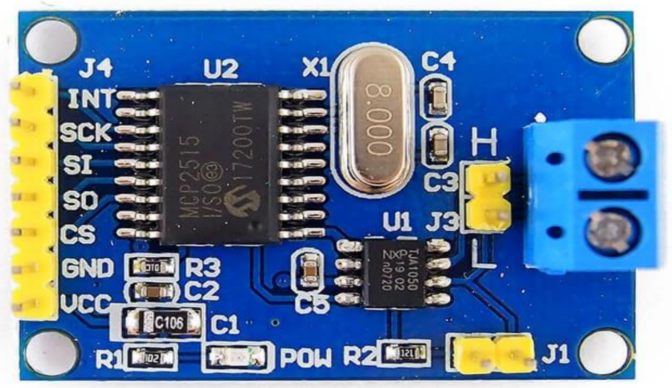


Figure 66: SPI TO CAN Module.

So,we connect Raspberry pi to SPI bus in the SPI TO CAN Module which has four connections as follow:

- MOSI (Master Out Slave In).
- MISO (Master In Slave Out).
- SCLK (Serial Clock).

Adding to that two other pinouts:

- CS or SS (Chip Select, Slave Select) to enable and disable the chip.
- INT (Interruption) to interrupt the chip when needed

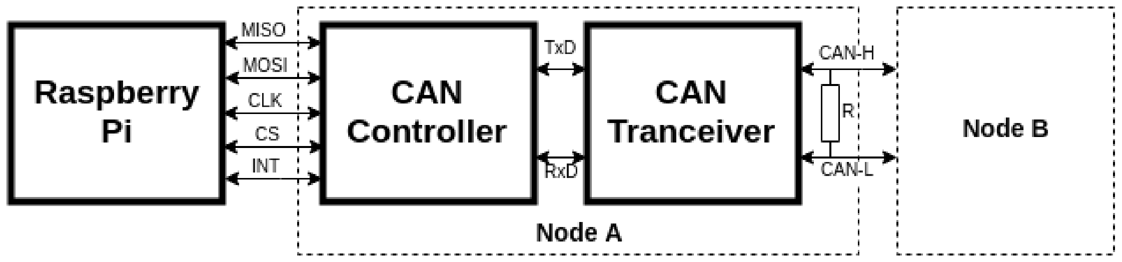


Figure 67: The connection of Raspberry pi to SPI TO CAN Module

7.7. Design Aspects

Our GUI system is based on the **flat design** which appears in the professional user interfaces on the market.

Flat design is a design style of the user interface using plain, two-dimensional elements and bright colors. It is often contrasted with the skeuomorphic style by copying real-life properties which gives the impression of three dimensions. Upon the introduction of Windows 8, Apple's iOS 7 and Google's Material Design, all of which make use of flat design, its prominence became prominent.

Flat design was originally designed for responsive design, in which the content of a website seamlessly varies depending on the screen size of the user. Use simple shapes and minimal textures, flat design ensures responsive designs work well and load quickly (especially important because mobile devices have slower internet speeds). Flat design provides users with a simpler and more efficient user experience by reducing the amount of visual noise (in the form of textures and shadows).

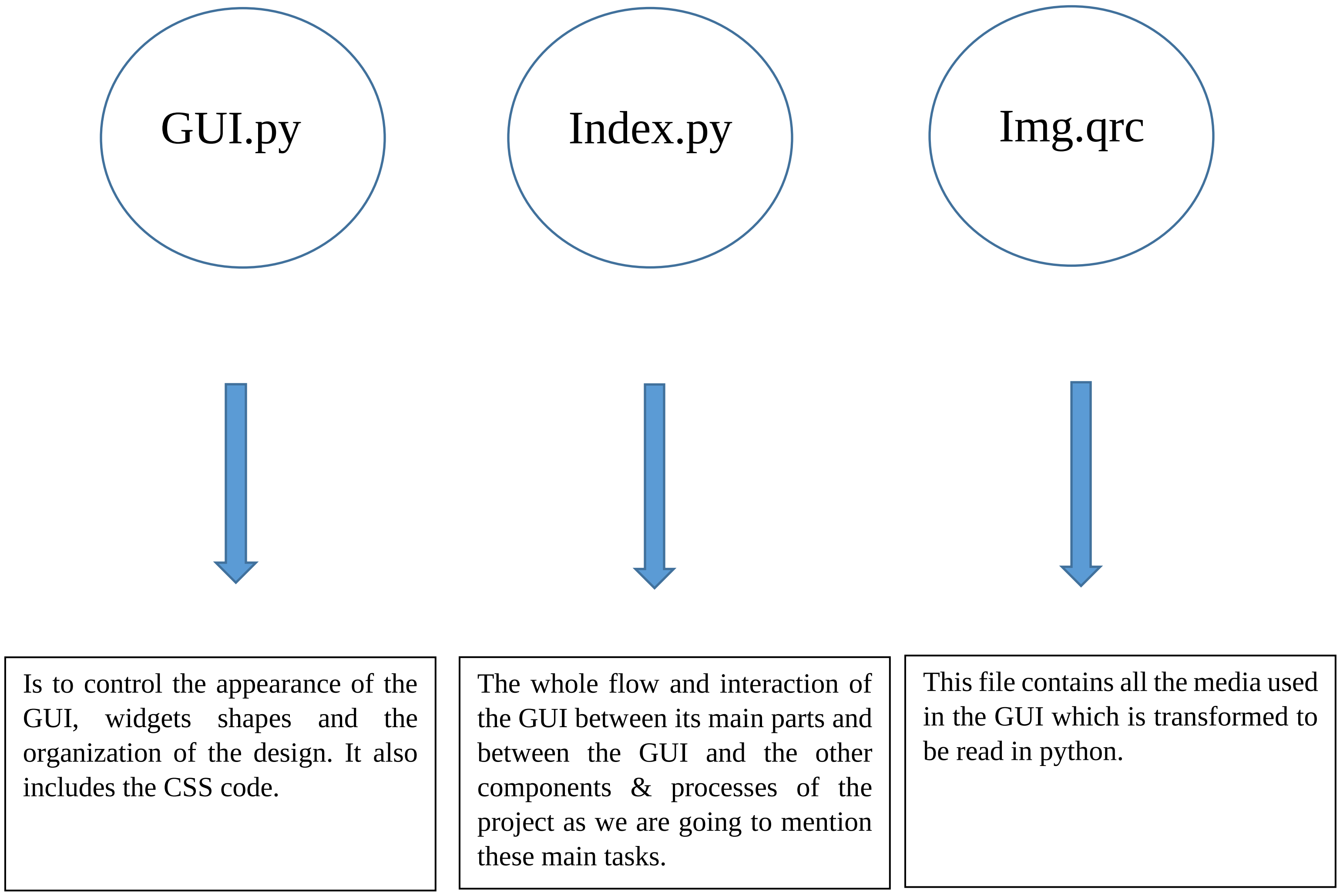
So, we took advantage of that in our system using **CSS** (Cascading Style Sheets) language we wrote 1300 lines styling the whole system colors, fonts, sizes, contrasts, etc.

CSS (Cascading Style Sheets) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts.[3] This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

The css code of our system will be attached in the appendix.

7.7.1. The GUI system consists of 3 files which includes 3 codes



Is to control the appearance of the GUI, widgets shapes and the organization of the design. It also includes the CSS code.

The whole flow and interaction of the GUI between its main parts and between the GUI and the other components & processes of the project as we are going to mention these main tasks.

This file contains all the media used in the GUI which is transformed to be read in python.

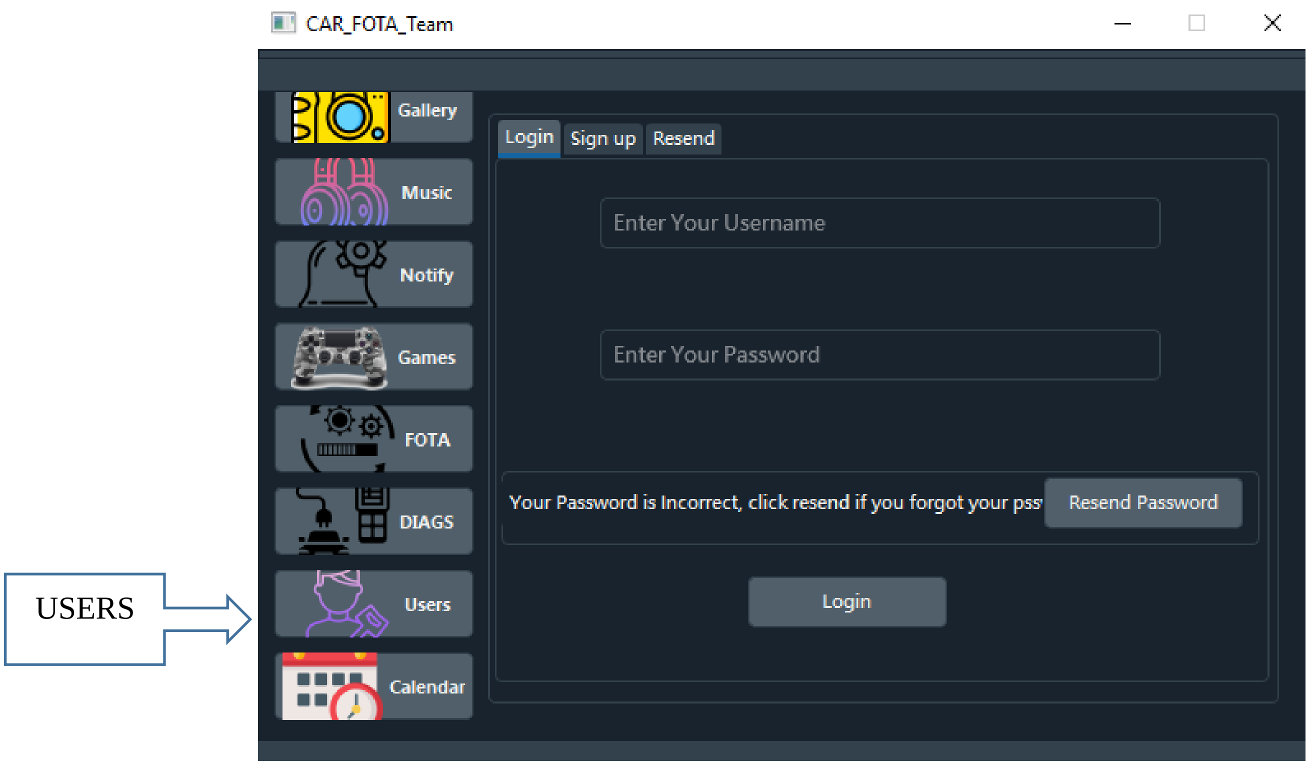
Index.py:

1- It includes the signaling and slots concept.
2- A virtual lining between tabs and widgets.

3- Receiving different kinds of text to appear on the GUI.
4- Sending signals and controllable values.
5- Security is handled by a process of user login and sign up.

7.8. GUI features

7.8.1. User Sign up & login



Your Password is Incorrect, click resend if you forgot your pss

Figure 68: USER Login.

Here as we can see, this is the user tab where the user can login with his username and password fields, there is also an option if the user forgot his password, he can click the resend password button in order to receive his password.

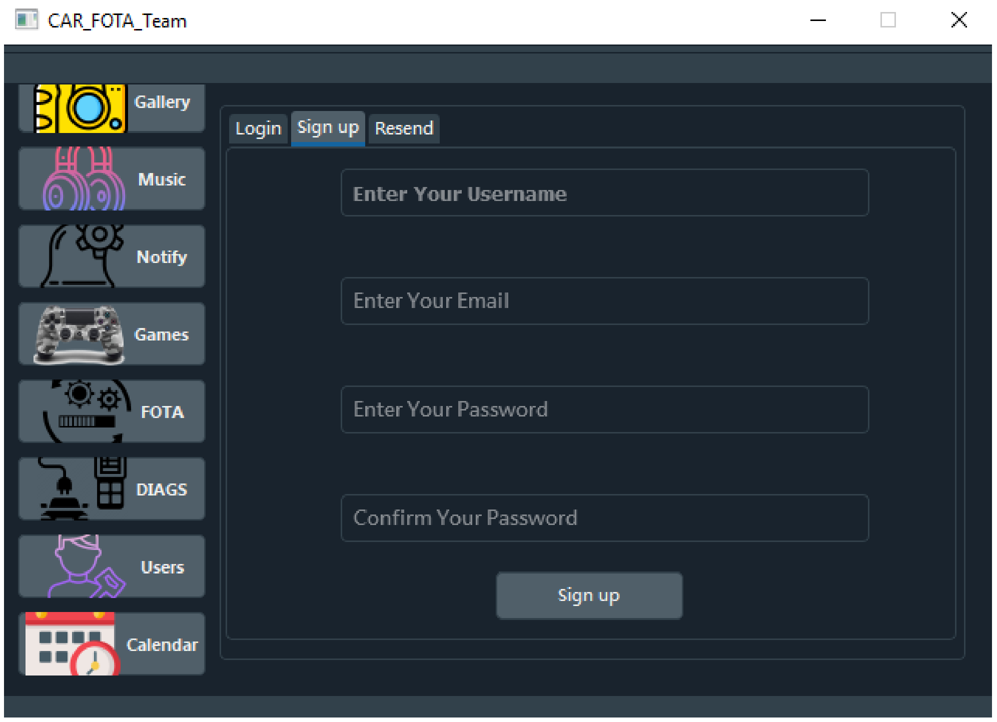


Figure 69: USER Signup.

Here as we can see, sign up tab where 4 fields are existing to serve a new user.

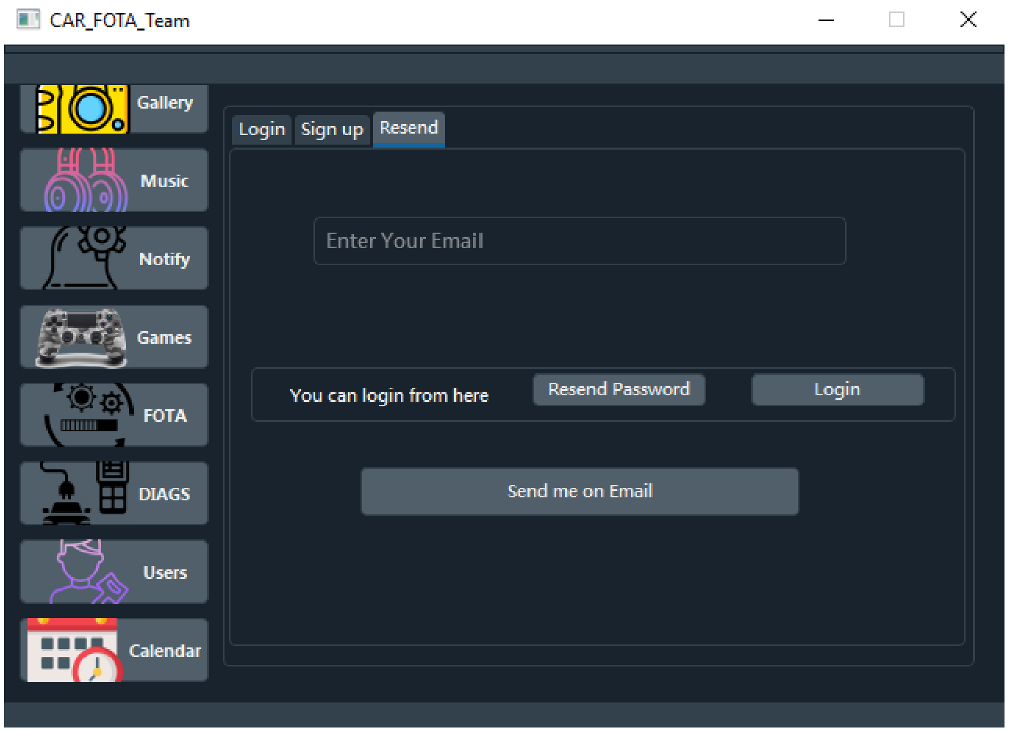


Figure 71: Resend tab.

Here as we can see, Resend tab where the user can enter the email he signed up with in order to receive his old password.

7.8.2. The main 2 features

7.8.2.1. FOTA

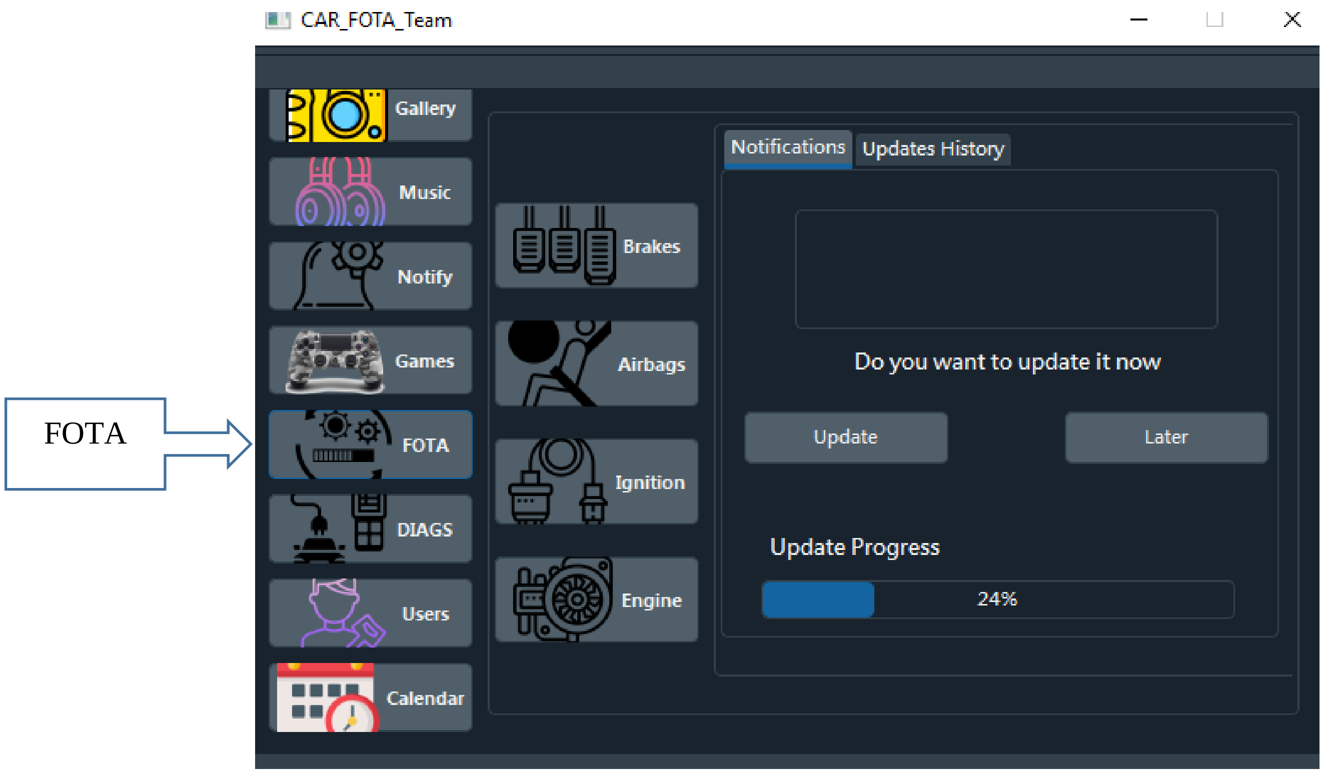
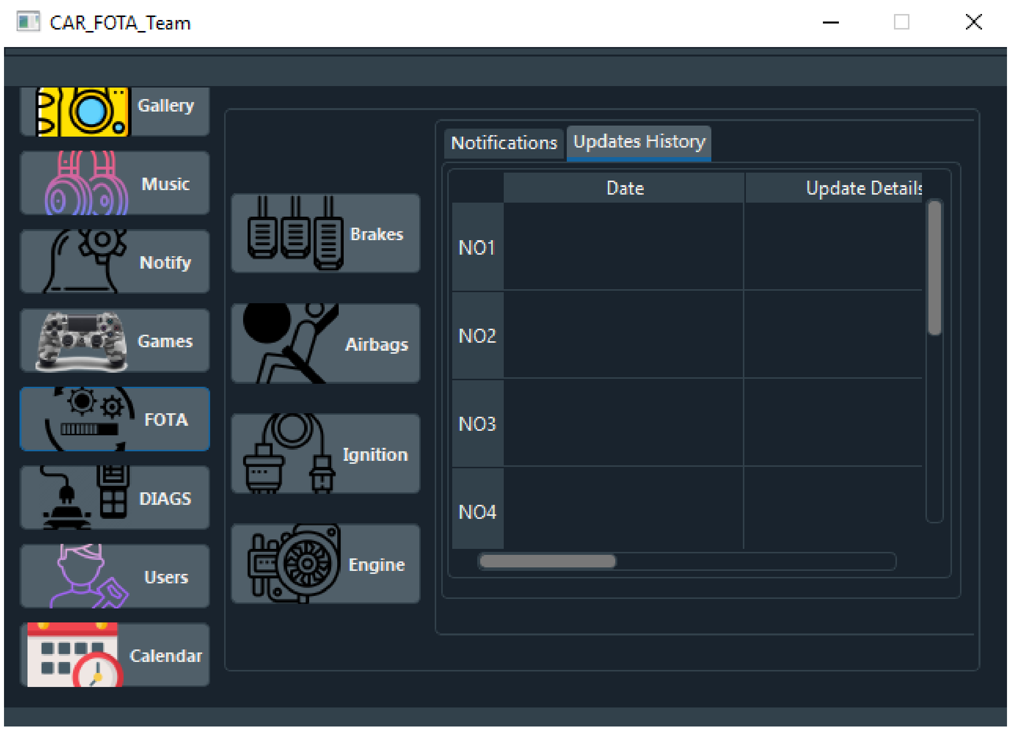


Figure 72: FOTA (Notifications).

Here as we can see, our main role the FOTA main tab which contains our Notifications tab & and Updates history tab,

The notification tab contains the different ecus updates which the user can check it any time there is a text box which will

allow the user to receive its update message and asking the user whether yes or no for the update with 2 push buttons



	Date	Update Details
NO1		
NO2		
NO3		
NO4		

Figure 73: FOTA (Updates History).

Here as we can see, the updates history tab which handle the history of the updates and its details in order for the user

to see these details. (Its GUI handling is done but it needs a further work with database which is gathering the info we

will talk about this point as it is considered a future work.)

7.8.2.2. Diagnostics

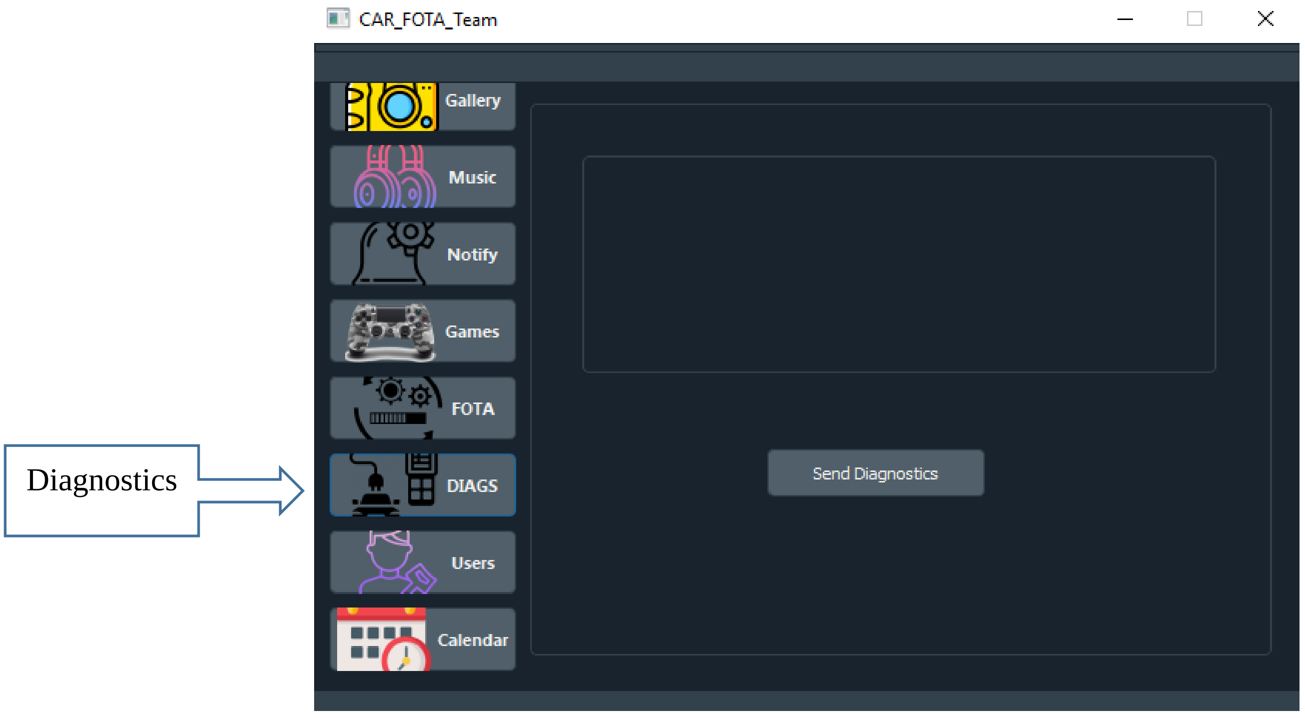


Figure 74: Diagnostics.

Here as we can see, the Diagnostics tab where if there is a diagnostic sent by any of the ecus to the user he can see it and

send it to the server in order to be seen and studied by the OEMs.

7.9. GUI advantages

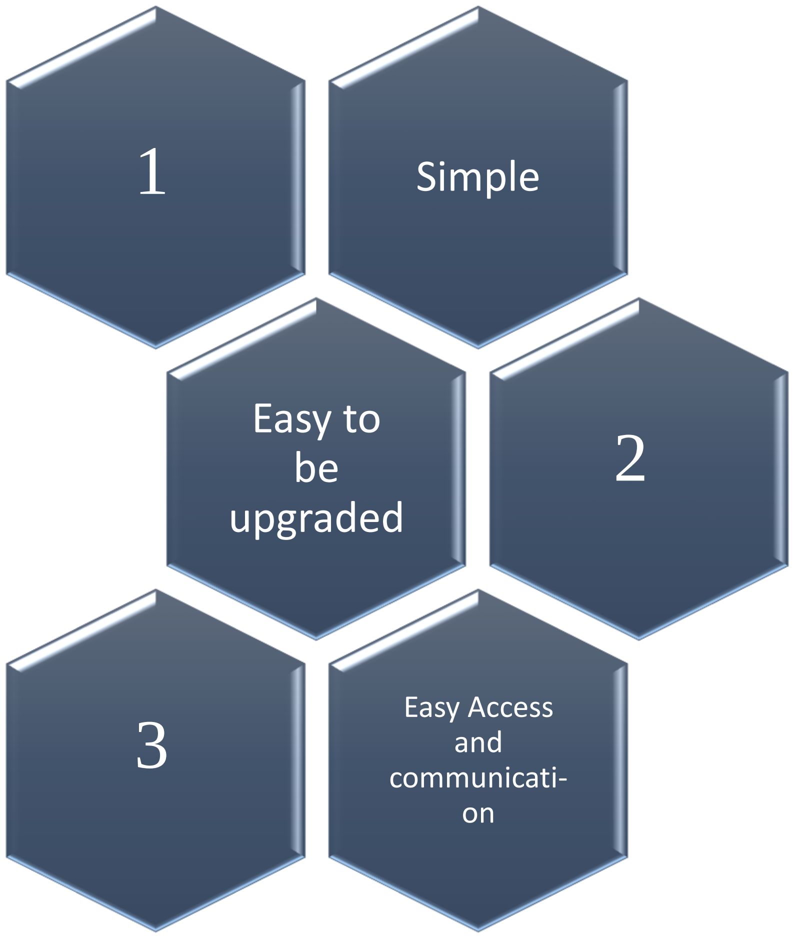


Figure 75: GUI Advantages.

7.10. GUI future work

For the Users login, sign up and updates history in the FOTA tab these features are handled from the gui side as receiving from the main code but in real life it works with databases it depends on the database side to receive from the database the user information and check it upon the saved data also the same thing for the (updates history) tab the server must save all the updates info in a database which the gui can access it and gather the info to view it. All this can be handle using

MySQL database or SQLite database using python extensions and connections but as it is something out of our scope so we have postponed it to a later time.

Chapter 8: Security

8.1. Cryptography

Cryptography is the science of encrypting and decrypting messages and text, or the study of hidden writing and it is a method of protecting communications and information through the use of codes, so that we can prevent public from reading private messages, only those for whom the information is intended can read and process it. The cryptography word consists of two prefixes, crypto stands for hidden and graphy stands for writing.

8.1.1. Cryptography techniques

Cryptography is related to the disciplines of cryptanalysis and cryptology and. It includes

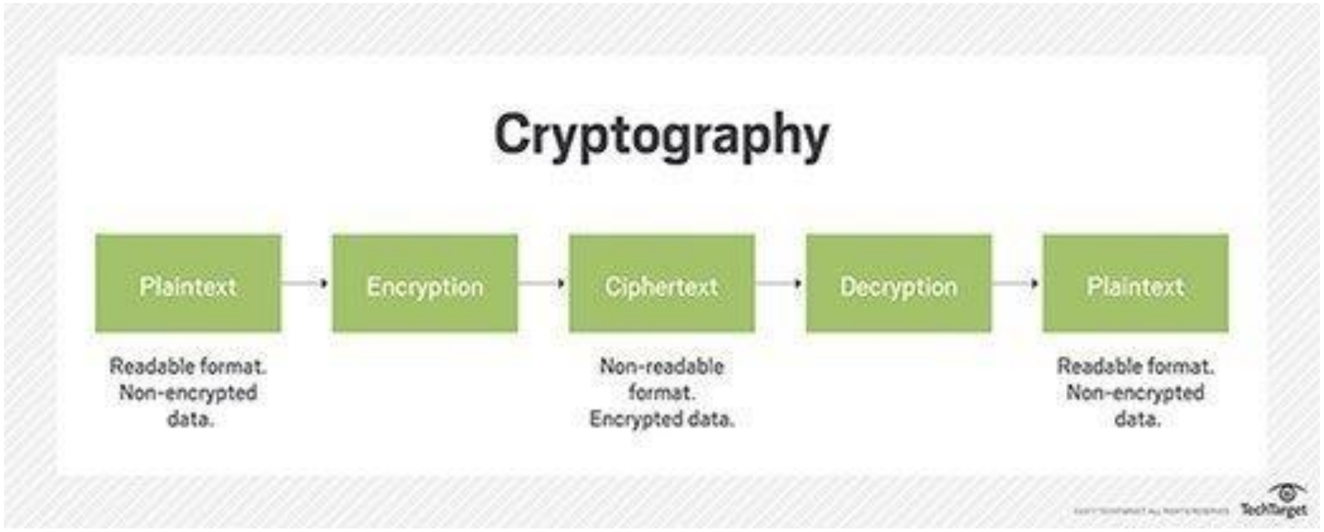

Cryptography
Plaintext
Encryption
Ciphertext
Decryption
Plaintext
Readable format. Non-encrypted data.
Non-readable format. Encrypted data.
Readable format. Non-encrypted data.
TechTarget

Figure 76:Cryptography system

techniques to hide information in storage or transit such as merging words with images, microdots and other ways. But, Today in computer-centric world, cryptography is about scrambling plaintext (Nonencrypted data) into ciphertext(Encrypted data) and this process is called **encryption**, then taking the ciphertext and backs it to plaintext and this process is called **decryption**. Cryptographers are the people who practice this field.

Modern cryptography should achieve the following four objectives:

1. **Confidentiality**: the information cannot be understood by anyone for whom it was unintended

2. **Integrity:** the information cannot be altered in storage or transit between sender and intended receiver without the alteration being detected

3. **Non-repudiation**: the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information

4. **Authentication**: the sender and receiver can confirm each other's identity and the origin/destination of the information.

8.1.2. Cryptographic algorithm

Cryptosystems use ciphers or cryptographic algorithms, to encrypt and decrypt messages so that it can secure communications among devices such as smartphones, computer systems and applications. A cipher suite uses one algorithm for encryption, another algorithm for key exchange and another algorithm for message authentication. This process involves digital signing and verification for message authentication, and key exchange, public and private key generation for data encryption/decryption.

8.1.3. Types of cryptography

- **Single-key or symmetric-key encryption** (AES, DES): uses single key for encryption and decryption.

- **Public-key or asymmetric-key encryption**(RSA): uses key for encryption and another key for decryption.

- **Hash functions** (SHA-1, SHA-2, SHA-3): uses mathematical transformation to encrypt data.

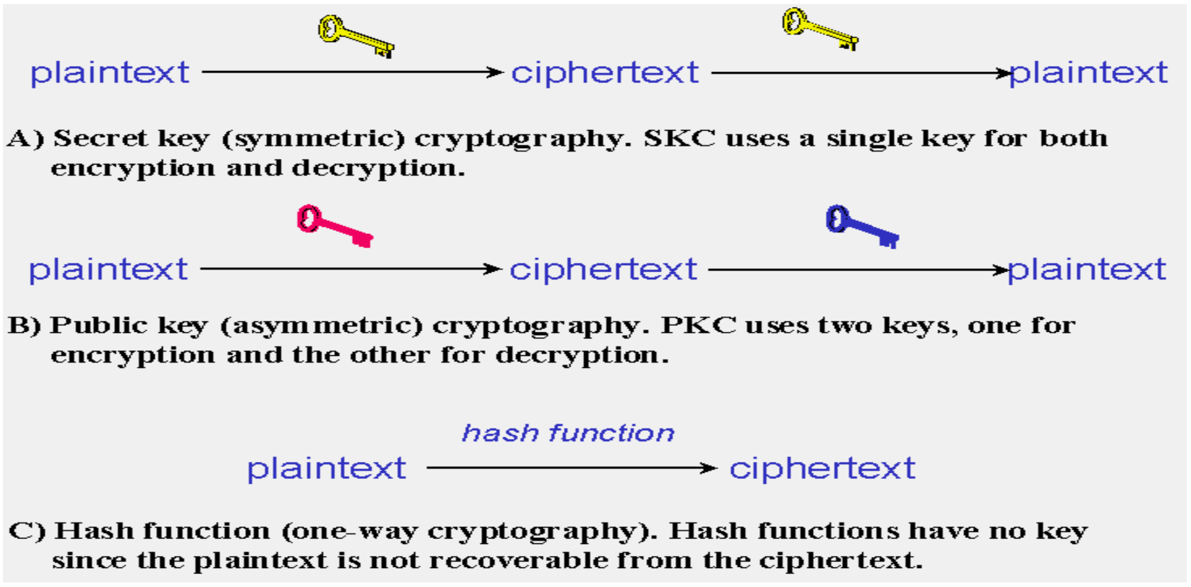

plaintext
ciphertext
plaintext
A) Secret key (symmetric) cryptography. SKC uses a single key for both encryption and decryption.
plaintext
ciphertext
plaintext
B) Public key (asymmetric) cryptography. PKC uses two keys, one for encryption and the other for decryption.
hash function
plaintext
ciphertext
C) Hash function (one-way cryptography). Hash functions have no key since the plaintext is not recoverable from the ciphertext.

Figure 77:Three types of cryptography

8.2. HMAC

8.2.1. HMAC history

The definition and analysis of the HMAC construction was first published in 1996 in a paper by Mihir Bellare, Hugo Krawczyk, and Ran Canetti.They also wrote RFC 2104 in 1997. The 1996 paper defined a nested variant called NMAC. FIPS PUB 198 standardizes and generalizes the use of all HMACs. HMAC is used within the SSH, IPsec, JSON Web Tokens, and TLS protocols.

8.2.2. What is HMAC?

HAMC stands for Hash Message Authentication Code, it is a technique to verify the integrity of a message transmitted between two parties which agree on a shared secret key.

Any cryptographic hash function, such as SHA-3 or SHA-2, may be used to calculate HMAC function; the resulting MAC algorithm is termed HMAC-X, where X is the hash function used (e.g. HMAC-SHA256 or HMAC-SHA3-256). The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size and quality of the key, and the size of its hash output.

8.2.3. HMAC Algorithm

HMAC combines the original message and the secret key to compute a message digest function. The transmitter of the message calculates the HMAC of the message and the key then transmits the HMAC with the original message. The receiver recalculates the HMAC using the message and the secret key, then compares the received HMAC with the calculated HMAC to see if they match or not . If the two HMACs are identical, then the receiver knows that the original message has not been changed because the message digest hasn't changed, and that it is authentic because the transmitter knew the shared key, which is presumed to be secret.

Any change to the data or the hash value results in a mismatch, because knowing the secret key is required to change the message and reproduce the correct hash value. Therefore, if the original and computed hash values match, the message is will be authenticated.

8.2.4. How HMAC function works?

HMAC uses two passes of hash computation. First the secret key is used to derive two keys – inner and outer. The first pass of the algorithm produces an internal hash which is the output of the HASH function when it takes the inner key and the original message as its two 2 inputs. The second pass produces the final HMAC code derived from the same HASH function when it takes the inner hash result and the outer key as inputs. Thus, this algorithm provides better immunity against attackers.

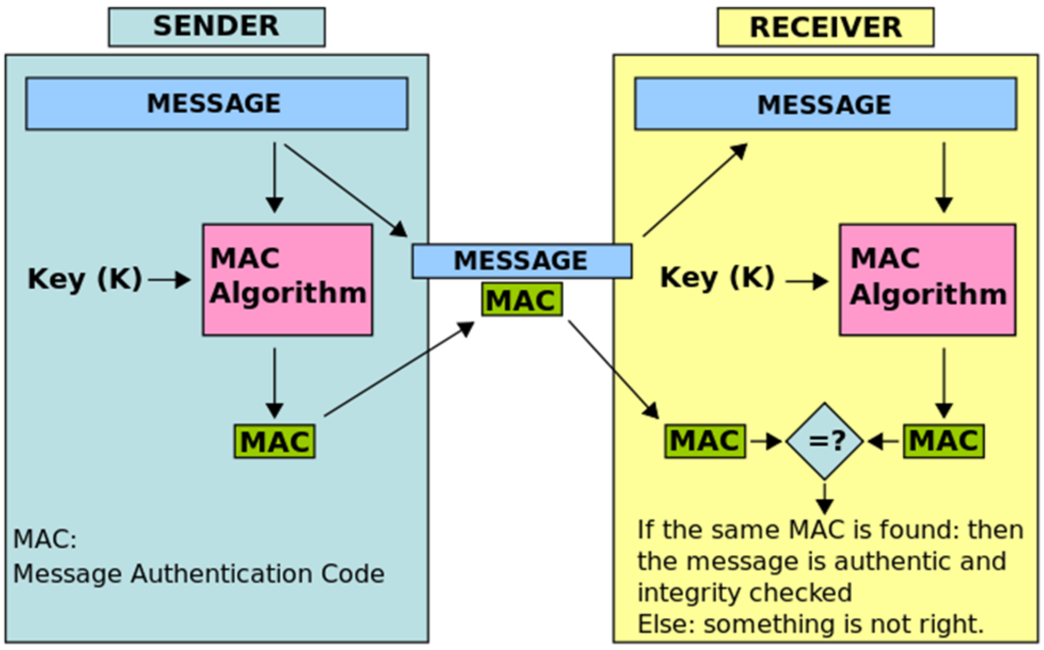

SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

Figure 78: HMAC Process

8.2.5. Properties of HMACSHA256

- The key can be any length. However, its recommended size is 64 bytes (512 bits).
- The output hash is of length 256 bits.
- SHA-256 operates on 512-bit blocks.
- HMACs in general are fast because they use hash functions rather than public key mathematics.

8.2.6. HMAC in our FOTA

The client and server agree on a common hash function, which is SHA256 that has been chosen because its characteristics are suitable with our needs.

Before the server sends out the file, it first obtains a hash of that file using the HMAC-SHA256 hash function. It then sends this hash along with the file itself. Upon receiving the two items (i.e. the file and the hash), the client obtains the HMAC-SHA256 hash of the downloaded file and then compares it with the downloaded hash. If the two hashes are matched, then that would mean the file was not tampered along the way. Otherwise it means that the file have been attacked and the data is changed.

Chapter 9: Future Work

9.1. Seamless FOTA

The main idea of the seamless FOTA is, to have two blocks of FLASH memory for code execution in each microcontroller of the single ECUs. The first block (Block A) is used to execute the actual code and the second block (Block B) is a spare FLASH block. So, the new software can be programmed into block B in the background while driving the car. Then, the code execution will be swapped from block A to block B and this will happen after all ECUs finish the pre-storing process. The SWAP will be completed with a final restart of the ECUs.

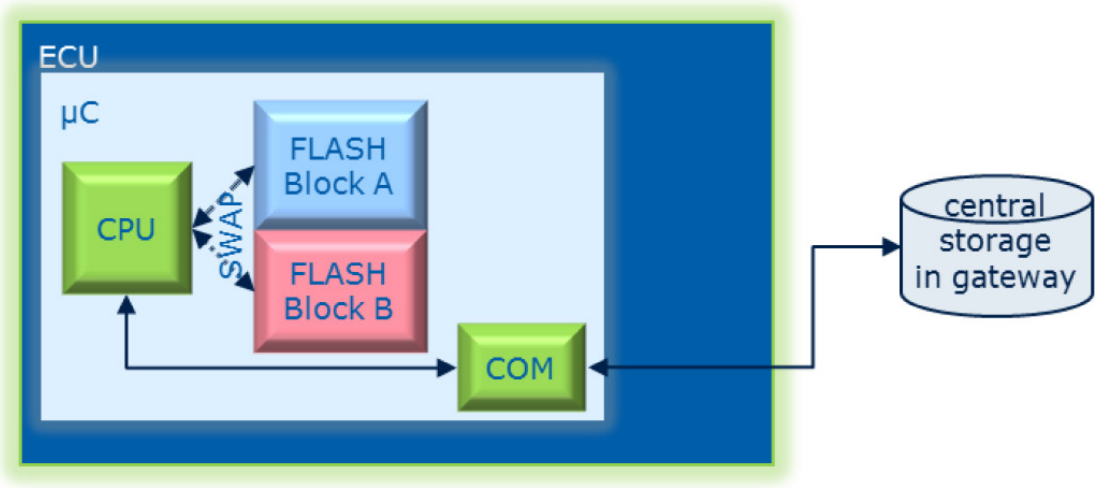


Figure 79: A/B swap process

9.1.1. Advantages and disadvantages of seamless FOTA

The advantages of seamless FOTA are:

1. Almost there is zero downtime of the vehicle.
2. A restart takes no more than a few milliseconds and then can be added to the power-up or -down cycle as mentioned before.
3. There is a fallback solution available within block A if something went wrong in the vehicles network.

4. There is always the option to switch back to block A of all ECUs of the actual service update within a few milliseconds, If the new software image of any ECU turns out to be corrupted.

The disadvantages of seamless FOTA are:

1. The size of embedded program FLASH is doubling and also today, the mechanisms of swapping process are not available for most microcontrollers.
2. The demand for swapping mechanism of a seamless memory is a challenge for any microcontroller supplier. The potential impact on the entire system architecture should be carefully considered in order to come to a robust implementation of the swap mechanism apart of the necessary availability of components with the required memory sizes.
3. When the FLASH of a microcontroller is doubled from 4MB to 8MB, this obviously will increase the cost.
4. The A/B Swap challenge approach is even graver for components at the upper memory limit of the actual technology note.

9.2. Delta file

9.2.1. What is delta file

Delta file is a very important feature that would add a very great value to the project, as it has a great effect in using large files, it saves time and storage .Delta file is mainly used to reduce the size of the sent file, it is a way of storing or transmitting data in the form of differences (deltas) between sequential data instead of complete files; this is known as data differencing. Delta encoding is also sometimes called **delta compression**.

9.2.2. Advantages of delta compression

- Reduce the potential for security flaws to be exploited before the affected software can be updated.
- Allow the Software to be updated more quickly.

- Reduce error due to wireless network.
- Reduce Transmission Time.

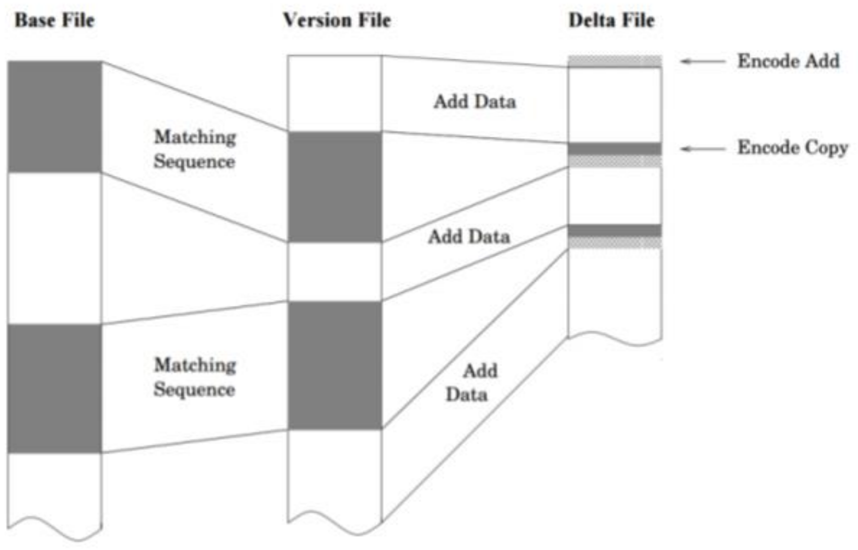


Figure 80: Delta File Algorithm

9.2.3. Generating the delta (Patch) file

- **Patch File Consists of 3 files:**

1. Control File containing ADD and INSERT Instructions.
    - Each ADD instruction specifies an offset in the old file and a length; after that the correct number of bytes are read from the old file and added to the same number of bytes from the difference file.
    - INSERT instruction specify only a length; the specified number of bytes is read from the extra file.
2. Difference File.
3. Extra file.

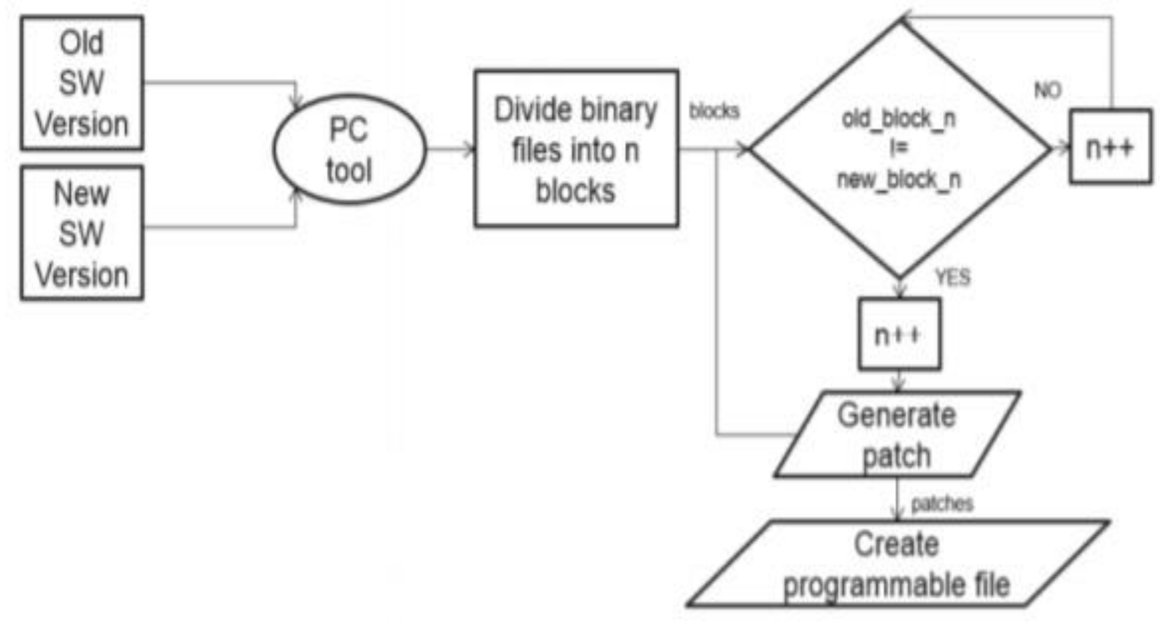


Figure 81: generating delta file

In FOTA delta file should be generated at the server using specific technique and only this file will be sent to the main ECU then after the ECU downloads the delta file it will mix both the old file and the delta file with each other and by using a specific algorithm, a new file will be generated a new file which should be burn on the microcontroller by the bootloader.

APPENDIX A

JSON Web Token (JWT) (jwt, 2020)

What is JSON Web Token?

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

What is the JSON Web Token structure?

In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

- Header
- Payload
- Signature

Therefore, a JWT typically looks like the following: xxxxx.yyyyy.zzzzz

Let's break down the different parts.

Header

The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

For example:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Then, this JSON is Base64Url encoded to form the first part of the JWT.

Payload

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.

Registered claims: These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: iss (issuer), exp (expiration time), sub (subject), aud (audience), and others.

Notice that the claim names are only three characters long as JWT is meant to be compact.

Public claims: These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.

Private claims: These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.

An example payload could be:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

The payload is then Base64Url encoded to form the second part of the JSON Web Token.

Signature

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

For example, if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:

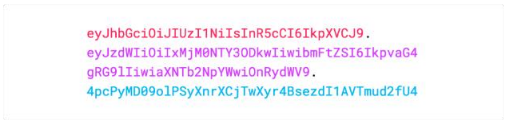
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

Putting all together

The output is three Base64-URL strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML.

The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Figure 82: JWT example

References

[1] https://www.futurebridge.com/blog/over-the-air-software-updates-reaping-benefits-for-the-automotive-industry/

[2]https://www.marketresearchfuture.com/reports/automotive-over-the-air-updates-market-7606

[3]https://www.icpdas-usa.com/cancheck.html

[4]https://www.totalphase.com/blog/2019/08/5-advantages-of-can-bus-protocol/

[5] RM0008 Reference Manual

[6]http://www.cse.dmu.ac.uk/~eg/tele/CanbusIDandMask.html

[7]https://copperhilltech.com/blog/controller-area-network-can-bus-message-frame-architecture/

[8] https://en.wikipedia.org/wiki/GSM

[9] https://en.wikipedia.org/wiki/General_Packet_Radio_Service

[10] https://en.wikipedia.org/wiki/4G

[11] https://www.tutorialspoint.com/gsm/gsm_user_services.htm

[12]
https://en.wikipedia.org/wiki/GSM_services#:~:text=In%20order%20to%20gain%20access,has%20been%20consumed%20(postpaid)

[13] https://www.researchgate.net/figure/Conventional-access-to-Internet-through-GSM-networks_fig1_2274364

[14] SIM800L Datasheet

[15] https://searchstorage.techtarget.com/definition/flash-memory

[16] https://www.hyperstone.com/en/Solid-State-bit-density-and-the-Flash-Memory-Controller-1235,12728.html

[17] PM0075 Programming manual, STM32F10xxx Flash memory microcontrollers

[18] https://en.wikipedia.org/wiki/Intel_HEX

[19]https://www.slant.co/versus/16724/22768/~tkinter_vs_pyqt

[20]https://doc.qt.io/qt-5/classes.html