To obtain the degree

Master of Science

in the degree course mechatronic

at the Technical Faculty, Department mechatronic,

University of Erlangen–Nuremberg

Software Infrastructure for a Distributed EV Charging System

Master's thesis

Presented by Xinhuang Gong

Matriculation-Nr. 22595969

Erlangen, 01.06.2021

Supervisor: Prof. Dr.-Ing. Thomas Moor

Chair of Automatic Control

University of Erlangen–Nuremberg

Declaration

I confirm that I have written this thesis unaided and without using sources other than those listed and that this thesis has never been submitted to another examination authority and accepted as part of an examination achievement, neither in this form nor in a similar form. All content that was taken from a third party either verbatim or in substance has been acknowledged as such.

Erlangen, der 01.06.2021

Xinhuang Gong

Contents

List of figures

List of tables

Works Cited

[1] Moor, Thomas. "AGCCS/AGCCS-CTRL22". github.com. Web. 26 May 2021. < https://github.com/AGCCS/AGCCS-CTRL22>

[2] Thurnherr, Pascal. "Entwicklung, Aufbau und Test einer Ladeeinrichtung für Elektrofahrzeuge nach IEC 62196". github.com. 31 Aug. 2020. Web. 26 May 2021. < https://github.com/dreadnomad/FGCCS-Ctrl22>

[3] M5Stack Company. "M5StickC Documents". m5stack.com. Web. 26 May 2021. <https://docs.m5stack.com/en/core/m5stickc>

[4] Stegen Electronics. "SmartEVSE". github.com. 26 Jun. 2020. Web. 26 May 2021.< https://github.com/SmartEVSE/SmartEVSE-2>

[5] Espressif Systems. "ESP32-WROOM-32 Datasheet". espressif.com. Web. 26 May 2021. <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>

[6] Wikipedia. "SAE J1772". wikipedia.org. 5 Oct. 2019. Web. 26 May 2021. <https://en.wikipedia.org/wiki/SAE_J1772>

[7] Microchip Technology. "ATmega4808 Datasheet". Jan. 2020. Web. 26 May 2021. <http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega4808-09-DataSheet-DS40002173B.pdf>

[8] Espressif Systems. "esp-mdf". 19 May 2021. github.com. Web. 28 May 2021. < https://github.com/espressif/esp-mdf >

[9] Espressif Systems. "ESP-MDF". docs.espressif.com. Web. 28 May 2021. < https://docs.espressif.com/projects/esp-mdf/en/latest/get-started/index.html>

[10] Espressif Systems. " ESP-MESH". docs.espressif.com. Web. 28 May 2021. < https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/mesh.html>

[11] Vue. "Introduction". vuejs.org. Web. 29 May 2021. < https://v3.vuejs.org/guide/introduction.html#what-is-vue-js>

[12] Wikipedia. "Vue.js". 14 May 2021. wikipedia.org. Web. 29 May 2021. < https://en.wikipedia.org/wiki/Vue.js>

[13] Nelson, Brett. "Getting to Know Vue.js". New York: SSBM Finance Inc. 2018.

[14] Element. "Element-A Desktop UI Library". element.eleme.io. Web. 29 May 2021. < https://element.eleme.io/#/en-US>

[15] OpenJS Foundation. "About Node.js". nodejs.org. Web. 29 May 2021. < https://nodejs.org/en/about/>

[16] Wikipedia. "Node.js". 27 May 2021. wikipedia.org. Web. 29 May 2021. < https://en.wikipedia.org/wiki/Node.js>

[17] OpenJS Foundation. "Express". expressjs.com. Web. 29 May 2021. < http://expressjs.com/>

[18] Wikipedia. "Multitier architecture". 28 Apr. 2021. wikipedia.org. Web. 29 May 2021. < https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture>

[19] MoscaJS. "aedes". 14 May 2021. github.com. Web. 29 May 2021. < https://github.com/moscajs/aedes>

[20] Mapbox. "node-sqlite3". 9 Mar. 2021. github.com. Web. 29 May 2021. < https://github.com/mapbox/node-sqlite3>

[21] Tan, Guangzhi. "element-ui print table". 17 Oct. 2019. cnblogs.com. Web. 15 Mar. 2021. < https://www.cnblogs.com/woai3c/p/11692293.html>

[22] Crabbly. "Print.js". 17 Feb. 2021. github.com. Web. 15 Mar. 2021. < https://github.com/mapbox/node-sqlite3>

[23] Wikipedia. "MySQL". 1 Jun. 2021. wikipedia.org. Web. 2 Jun. 2021. < https://en.wikipedia.org/wiki/MySQL>

[24] Mysqljs. "mysql". 1 Oct. 2020. github.com. Web. 2 Jun. 2021. < https://github.com/mysqljs/mysql>

[25] Eclipse Foundation. "Eclipse Mosquitto". Mosquitto.org. Web. 2 Jun. 2021. < https://mosquitto.org/>

[26] OpenJS Foundation. " Node.js v16.3.0 documentation". nodejs.org. Web. 28 Dec. 2020. < https://nodejs.org/api/crypto.html>

[27] Wikipedia. "Advanced Encryption Standard". 5 May 2021. wikipedia.org. Web. 2 Jun. 2021. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard>

[28] Jameson Little. "crc32". 27 Apr. 2013. wikipedia.org. Web. 20 Mar. 2021. https://github.com/beatgammit/crc32

[29] Wikipedia. "Greedy algorithm". 29 May 2021. wikipedia.org. Web. 3 Jun. 2021. https://en.wikipedia.org/wiki/Greedy_algorithm>

[18] Multitier architecture

1. Introduction

This master's thesis in the degree course mechatronic is developed in the winter semester of University of Erlangen–Nuremberg at the Chair of Automatic Control, supervised by Professor Moor. It aims to advance an appropriate electric vehicle charging management system to manage and monitor the EV charging mesh established by the project AGCCS [1]. The charging mesh is composed of charging units, namely the nodes, which are designed by Pascal Thurnherr [2] and further developed by Thomas Moor.

A general overview of the project AGCCS and details of its hardware and firmware are presented in Section 2. All the relevant circuit diagrams, design documents, code, and compiled firmware have been published at the GitHub repository, https://github.com/AGCCS/AGCCS-CTRL22. In Section 2, there is also a brief introduction to the technology used in developing the EV charging management system in this paper.

To design the EV charging management system, its functional requirements and the program's structure based on these requirements are analyzed in Section 3. Subsequently, Section 4 introduces its general design, the chosen technology, and the reasons behind it. The following Section 5 deals with how the system is implemented.

In order to test the performance of the charging management system and ensure sufficient safety, different simulations and different test tools were introduced into the testing process at each phase. The whole process can be divided into three periods. In the early period, the prime functions like communication with nodes and simulated current allocation were verified with M5stickC [3]. More advanced functions such as Firmware Over-the-Air (FOTA) were tested through M5stickC and barebone boards. A final laboratory test has also been conducted to prove the actual performance of the software designed in this paper on the hardware of project AGCCS.

At the end of this thesis, concluding remarks are made in Section 7. The program designed in this paper has passed the final laboratory test, which means that it is a capable charging management system for AGCCS. However, it may not handle all the scenarios correctly independently due to the considerable complexity and unpredictability of the actual charging process. Therefore, the possible future work of this EV charging management system is also discussed in the final section.

2. Background

2.1 Project AGCCS

With the developments of charging and battery technology and the increase of public interest and awareness, the global electric vehicle market grows steadily. Accordingly, the need for electric vehicle supply equipment (EVSE) arises at the same time. Clearly, people will no longer be satisfied with only charging their EVs in charging stations and seek a more flexible and convenient way to charge. Now there are already charging approaches for individual users like wallbox, a small charging device for EV. The purpose of the project AGCCS is to extend the wallbox approach to more oversized parking garages so that there could be a suitable way to charge EVs in daily life, such as charging in a parking lot next to apartments or shopping venues. Moreover, this project attempts to provide dynamic power allocation to replace inefficient charging with fixed power allocation.

So far, the project AGCCS has been successfully realized a charging mesh, which allows the host to monitor the charging nodes and control their charging processes. However, all the operations are through the command line via a computer terminal. Therefore, a suitable management system is necessary to simplify the operation and facilitate management. Additionally, an approach to dynamically allocate power as efficiently as possible is also discussed in this paper.

2.2 Hardware

As mentioned in Section 1, the prototype of the hardware used in the project AGCCS is developed by Pascal Thurnherr [2]. Some parts of its design are based on the Open-Hardware-Project SmartEVSE [4], whose goal is to create EVSE for individual users. It is mentioned in SmartEVSE that up to 8 pieces of their equipment could be connected to supply the charging process of 8 vehicles. However, these pieces of equipment cannot communicate with each other due to the lack of communication modules. Therefore, Pascal Thurnherr integrated Espressif ESP32-WROOM-32-Modul [5] into the hardware of AGCCS as the WLAN module in the hardware to enable the WIFI function, which allows multiple devices to communicate with each other to form a mesh and make remote control possible. Users can now remotely monitor the charging process of EVs.

Furthermore, he extended the control of current to each phase of three-phase electricity. As a result, the allocation of electricity could be more flexible. One of the essential parts of his work is the current control module, which is based on the Combined Charging System (CCS) according to the SAE J1772/IEC 61851 standard [6]. It allows the microcontroller to change the value of current supplied through pulse width modulation of a 1-kHz-square-wave signal.

There are some slight defects in the circuit board design of Pascal, which Professor Moor has improved. The current test construction of the AGCCS hardware is shown in Figure 1. It consists of the mains contactors, the residual current detector, the current transformer, and the development board of AGCCS. A button and an LED are on the enclosure's housing, which is not presented in the figure below.

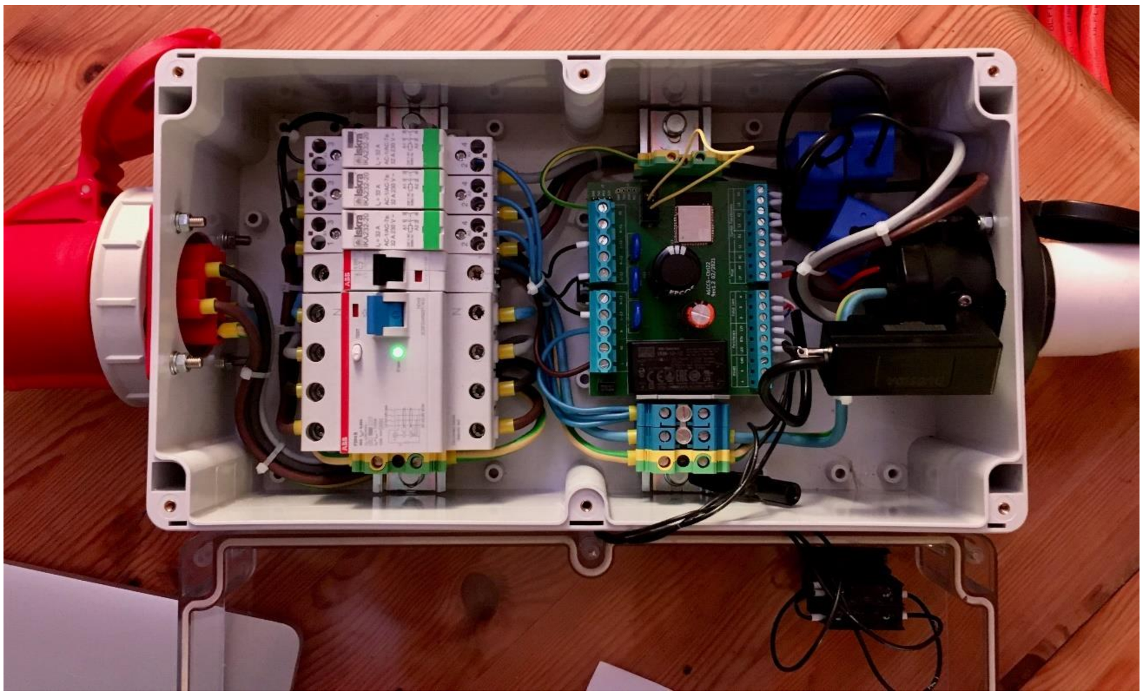


Figure 1: Test Construction of the AGCCS Charging Module [1]

2.3 Firmware

The AGCCS board applied two kinds of microcontrollers on it, Atmel ATmega4808 for the control and monitor of the charging process and Espressif ESP32-WROOM-32-Modul for the communication with the host and other nodes. Therefore, two kinds of firmware are advanced in the project AGCCS, *ctrl22c* for ATmega4808, the AVR microcontroller, and *demesh* for ESP32.

- **Ctrl22c:** The firmware for ATmega4808, which is described as AVR firmware, has two main tasks. One is to acquire essential parameters involved in the charging process. These parameters include the CCS state (CCSS), namely the state of the charging process, the state of the LED, the measured value of the current, the phases used, the maximum current set, the capacity of the cable, and some information of the node such as its mac address. The other is the communication with ESP32 via RS485 to send data to the host and receive commands through ESP32.
- **Demesh:** As mentioned above, ESP32 is the communication agent between the host and Atmega4808. In order to achieve that, Espressif Mesh Development Framework (ESP-MDF) [9] is applied in *demesh* to assist users of ESP32 to build the network ESP-WIFI-MESH [10]. This mesh allows numerous nodes to be interconnected over the coverage area of WLAN. Compared to the traditional WIFI network, in which one single access point (AP) is directly connected to all other devices, all the nodes In ESP-WIFI-Mesh could become an AP. Hence, it has a larger access range and higher capacity. Nevertheless, one AP differs from others, which is the only node directly connected to the central AP and is called the root node. The structure of the network developed by *demesh* is shown in Figure 2 below. There are two communication methods provided by *demesh*, using TCP/IP sockets or MQTT communication protocol, respectively. Both can realize the management of nodes. Meanwhile, this firmware can freely switch its corresponding circuit board when compiling to make the simulation test more convenient.

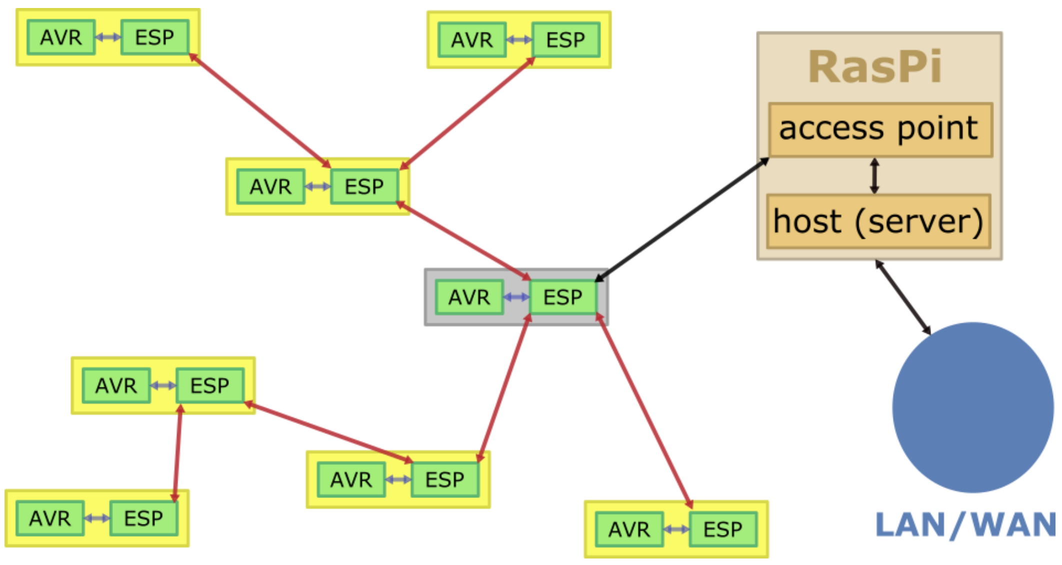


Figure 2: Network Architecture in demesh [1]

2.4 Vue.js and Element-UI

For the beginner in web application development, programming with the help of frameworks can significantly reduce the difficulty. The frameworks provide many examples to understand the source code and allow programmers to adapt additional tool libraries to simplify development flexibly. Like the other two most well-known JavaScript-based open-source front-end frameworks, React and AngularJS, Vue.js [11] is the latest model–view–viewmodel framework, which is developed by Evan You and the rest of the active core team members for building interactive Web-UI [12]. Compared with other frameworks, Vue.js is a progressive framework that focuses on the view layer, which means developers can build applications module by module as the functional requirements grow. Therefore, Vue.js could be beginner-friendly.

Element-UI is an open-source graphical toolkit for UI based on Vue.js [14]. However, the most popular JavaScript-based open-source front-end toolkit could be Bootstrap. Since Element-UI is developed for Vue.js, the corporation between them may be better. Moreover, using a graphical toolkit can avoid conflicts between graphics on the user interface so that the application appears more concise and beautiful.

2.5 Node.js and Express

Node.js is an open-source, cross-platform JavaScript runtime environment for back-end development [15][16]. Generally, a JavaScript program should be executed via a browser. However, Node.js offers developers the possibility to execute JavaScript code outside browsers like other programming languages. Furthermore, Node.js is designed based on the asynchronous event-driven method, which can greatly reduce the risk of program blocking.

When developing web applications through Node.js, Express, a web framework for Node.js [17], is also essential to avoid secondary development. It provides necessary solutions for routing, sessions, cookies, and so on to process HTTP requests.

3. Requirements Analysis

The primary purpose of this paper is to develop a proper system for the management of the charging mesh built in the project AGCCS. Consequently, the requirement profile should include the fundamental functional requirements of the back-end server and front-end user interface. Furthermore, the program should not have high-performance requirements, Since the Raspberry Pi is selected as the server and the central AP. Finally, it should respond quickly and have reasonable power allocation logic to meet the specification of the charging process in AGCCS.

The technical requirements derived from these primary conditions are discussed in the following two subsections.

3.1 Requirements for Program

1) **Simple operation and as automated as possible**
   As a management system that may be applied in general daily life, all operations should not be too complicated for ordinary people. Also, the system should have a high degree of automation to avoid taking too much time to control the charging process.
2) **Flexibility and standardization during operation**
   Due to the possible complexity when charging EVs, users should have a high degree of freedom. For example, it can be expected that users do not know the maximum allowable charging current for their EV or the cable capacity. However, they should still be able to set the charging current to any value. The system must adjust the set value so that it does not exceed the capacity. The flexibility does not mean that users could operate entirely freely. For instance, when setting the maximum current, it is forbidden to enter letters but only numbers.
3) **Fast response speed**
   The program should respond quickly to operations avoid long waiting for EVs to start charging. Additionally, it should also process the data quickly and present it to the user. Thus, the real-time status of the mesh can be monitored.

4) **Lightweight program**

   Despite the development of technology, the performance of Raspberry Pi has been reasonably high, if only the program designed in this thesis runs on it. It must be considered that there could be numerous nodes in the mesh. The communication between these nodes may occupy a considerable amount of performance. Therefore, it is reasonable to make the whole software lightweight to guarantee efficiency.

5) **Necessary security**

   For program design, security is always indispensable. In AGCCS, the safety of the EV charging process is ensured through the firmware *ctrl22c*. The program should at least be able to handle typical cyberattacks.

6) **Simplicity of maintenance and continued development**

   Although the charging management system should have the ability to fulfill the project AGCCS successfully, there can be unpredictable defects and problems. Hence, the possibility of subsequent maintenance and development should be given during program design.

7) **Open-source software**

   All source codes and all documentation of the development process should be openly accessible to allow interested private individuals and institutions to develop the system further and adapt it for their purposes. For this purpose, considering the software licenses of the libraries and source codes used here, all material required to advance the program in this thesis is published under free licenses.

3.2 Requirements for functions

1) **Restrictions on operation authority**

   Although this management system is designed for anyone, there should be a division of authority so that only authorized persons can control the mesh, and the rest without authorization can only monitor it. Therefore, the login interface and user creation function should be implemented, and the program should have the ability to distinguish users with different authorities.

2) **Data monitor**

   In order to observe the state of the mesh, a data monitor is essential. Hence, a database should be created to save the data. Besides, the monitor should be clear and easy to understand the information on it.

3) **Power allocation**

   Both firmware *ctrl22c* and *demesh* do not provide an approach to allocate the power, and the charging process is dependent on the command sent. To make it more convenient, the program developed in this paper should be able to allocate the power automatically. Furthermore, as mentioned in Section 2, a dynamic algorithm should be designed to make the allocation efficient.

4) **Basic operation on nodes**

   Instead of operation through the command line, users should be able to conduct the basic operations of nodes directly, such as pressing the operation button and blinking LED lights on the user interface

5) **Firmware Over-the-Air (FOTA)**

   As the project continues, the requirements will continue to expand. Accordingly, the firmware *ctrl22c* and *demesh* should also be adjusted. But after the AGCCS charging module is constructed, it is not easy to update the firmware via serial line. Therefore, the program should support FOTA to upload the firmware.

4. System Design

The typical logical structure of client-server web applications is called the three-tier system with the presentation tier, the application tier, and the data tier [18]. According to such a multitier architecture, a program can be divided into independent modules with different logic functions, making program development more reasonable and easier to maintain.

In web applications, programs are usually divided into two separate parts the front-end and the back-end. Generally, the front-end represents the user interface that interacts with clients, and the back-end is responsible for the function implementation. This separation prevents users from directly contacting the data, thereby effectively improving the security of the data. In this paper, the front-end is described as the console below, and the back-end is the server.

The system structure, front-end console, and back-end server design are presented in the following subsections.

4.1 Design of System Architecture

The physical system architecture of the entire AGCCS project can be divided into the user part, the server part, and the charging station part. The figure below shows how these 3 parts interact with others.

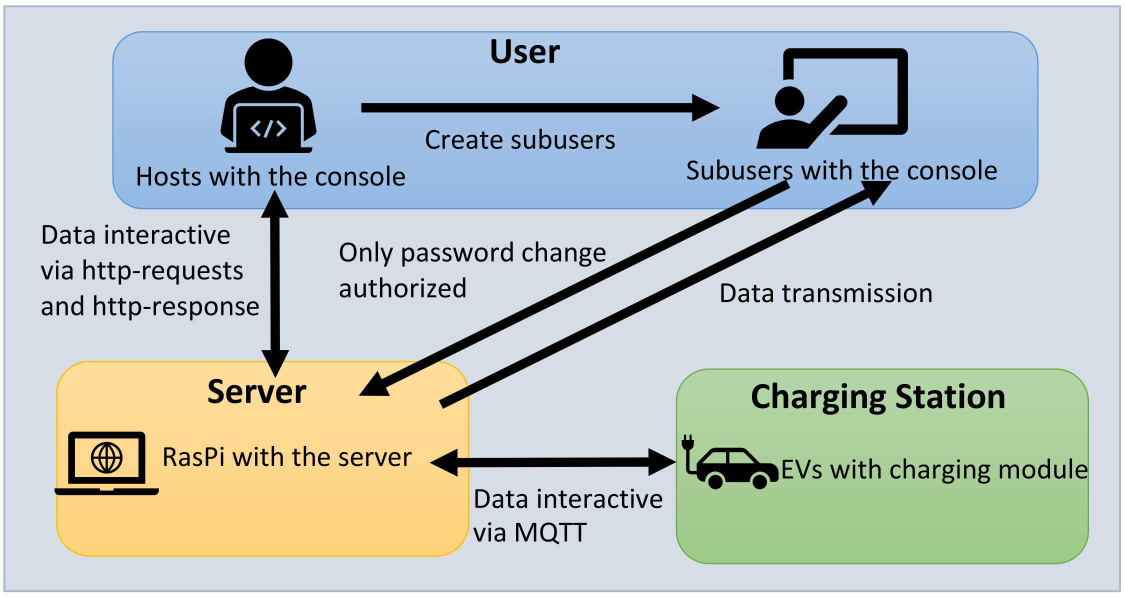

User
Create subusers
Hosts with the console
Subusers with the console
Data interactive via http-requests and http-response
Only password change authorized
Data transmission
Server
RasPi with the server
Charging Station
EVs with charging module
Data interactive via MQTT

Figure 3: Physical System Architecture

As shown in Figure 3, all the data interactions represent the functions implemented in the program, such as data monitoring, setting nodes, and creating subuser. As shown in the figure below, these functions are divided into 6 functional modules, the user module, the mesh module, the node module, the data module, the firmware module, and the communication module.

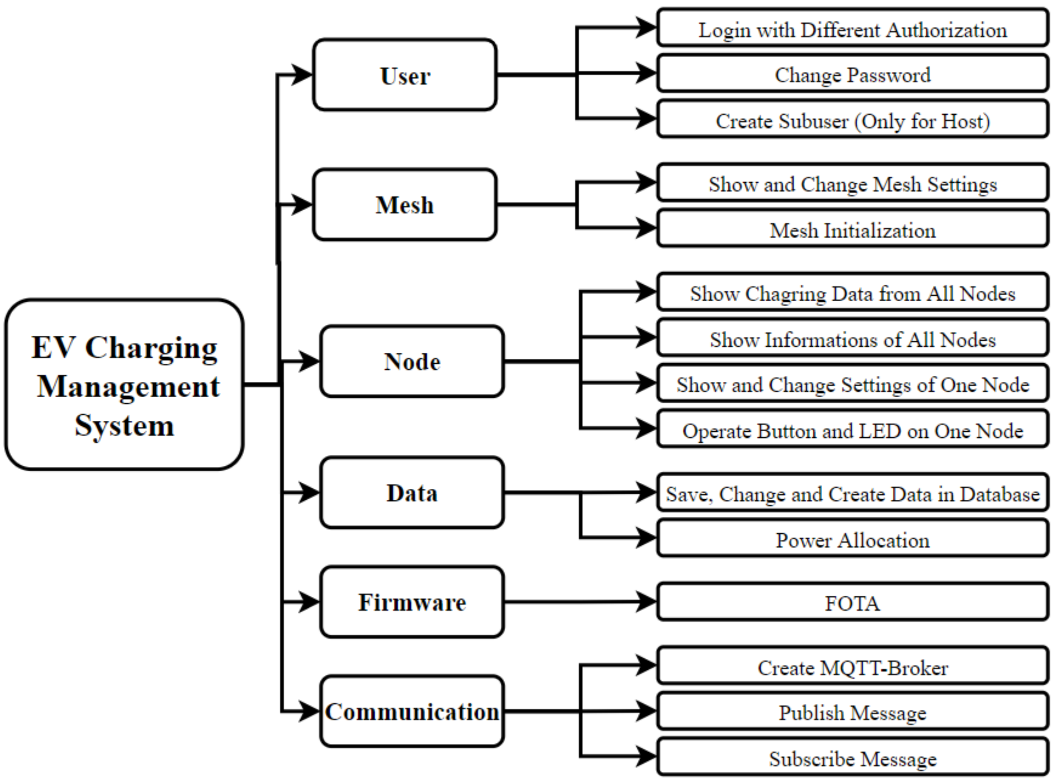

EV Charging Management System
User
Login with Different Authorization
Change Password
Create Subuser (Only for Host)
Mesh
Show and Change Mesh Settings
Mesh Initialization
Node
Show Chagring Data from All Nodes
Show Informations of All Nodes
Show and Change Settings of One Node
Operate Button and LED on One Node
Data
Save, Change and Create Data in Database
Power Allocation
Firmware
FOTA
Communication
Create MQTT-Broker
Publish Message
Subscribe Message

Figure 4: Program Function Architecture

As mentioned at the beginning of this section, web applications typically have a three-tier structure. Nevertheless, it could be subdivided into more tiers as the logic architecture of this management system with presentation tier, middle tier, application tier, data tier, and support tier.

As shown in Figure 5, the presentation tier takes the responsibility to display the user interface, whose outlook is dependent on the HTML code, the CCS code, and the static files such as pictures and icons. The middle tier forwards data and requests from the presentation tier and the application tier through Restful API and routers. All the data is transmitted in JSON format. After receiving the request, the application tier calls the code of the corresponding functional module to interact with the data tier. Finally, the application tier sends a response upstream after the data interaction is complete. The support tier may not be standard in most development cases.

At the beginning of the development, the database service is offered by MySQL, and the MQTT service requires the MQTT-broker, mosquitto. Besides, the in-memory database Redis is applied to manage the memory of Raspberry Pi. AS a result, users have to install three more software to run the EV charging management system. In order to prevent users from such a complicated initialization process, Aedes MQTT [19] and Sqlite3 database [20], which are both developed based on Node.js, are adopted in the subsequent development.

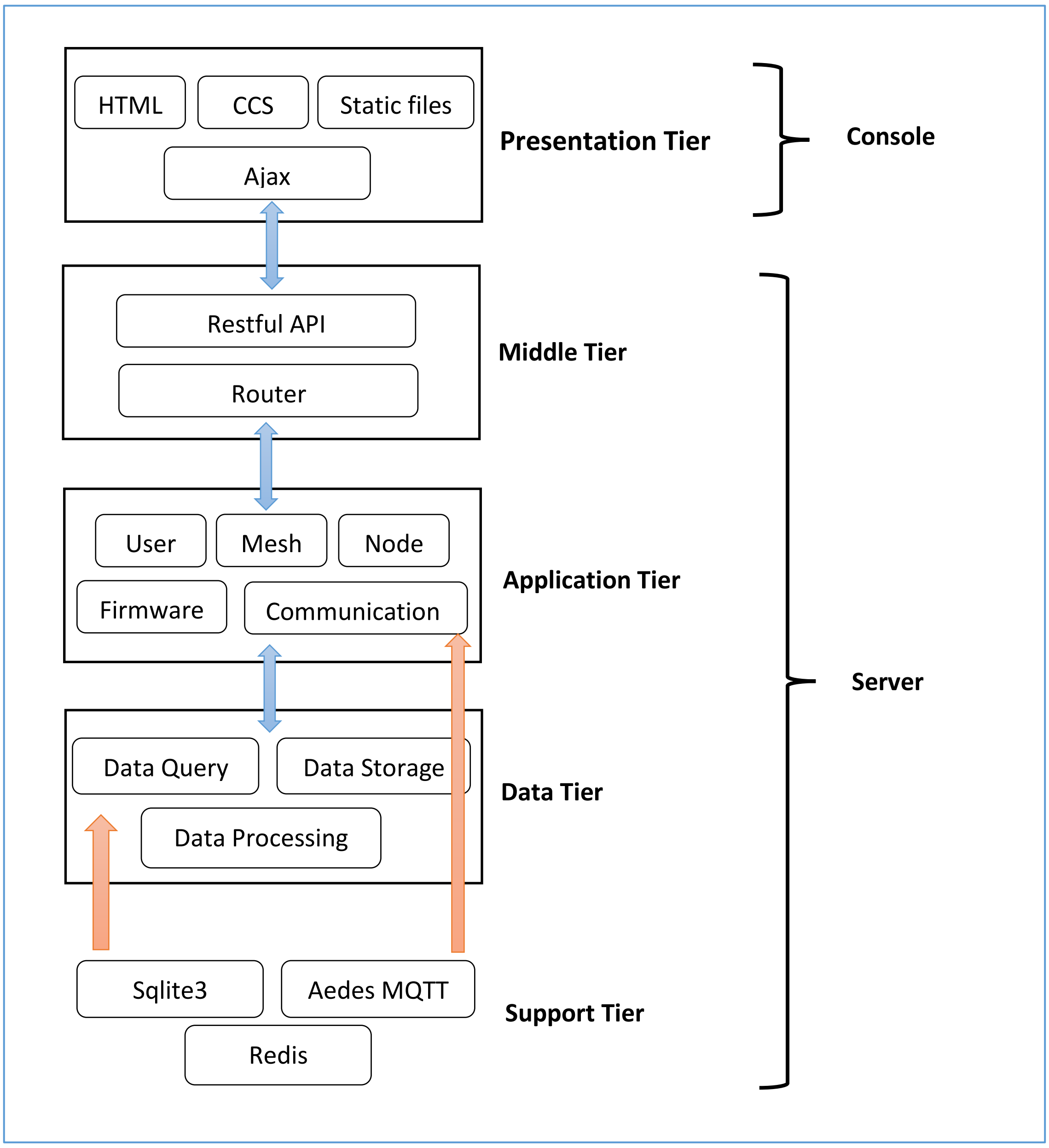


Figure 5: Program Logical Architecture

4.2 Design of Console

For the implementation of the console, the technology, web-based user interface, is chosen. In comparison with the traditional software-based user interface, the web UI has many advantages as follows:

- **No installation or updates:** Whether it is a personal computer, laptop, tablet, or smartphone, a browser is installed to browse the web. Therefore, the console platform has already been installed by the user in advance. Although webpages always need to be loaded, due to the separation of program design mentioned above, the console can be markedly lightweight, which incredibly reduces the loading time. As long as users visit the webpage, the console can be automatically updated to the latest version during loading.
- **Good compatibility:** Since all browsers use the same technologies, Html5, CCS, and JavaScript, to load web programs even on different platforms, a web user UI is compatible with all operating systems.
- **Possibility of direct access via mobile phone:** As mentioned above, a web UI possesses good compatibility, which means that users can run the console on mobile phones and computers simultaneously without redevelopment.
- **Possibility of remote access:** Because the console is a web application, users could remotely manage the charging mesh if a domain name is registered, which offers users more flexibility.

Many programming languages are used for web development, such as Java, Python, and JavaScript, which means many web development frameworks could be chosen. However, the basis of webpage compilation is still html5, ccs, and JavaScript. Therefore, a framework based on JavaScript is chosen for this thesis. There are three most popular JavaScript-based open-source front-end web frameworks, AngularJS, ReactJS, and Vue.js. In this paper, Vue.js [11] is selected to apply the front-end open-source toolkit Element [12] and speed up the development.

4.3 Design of Server

Due to the choice of Web-UI and supporting framework and toolkit when designing the console program, it is highly recommended to use Node.js to develop the back-end server. On the one hand, the increment of programming languages could be avoided to reduce the complexity of the entire program design. On the other hand, due to the use of the web framework Express without the secondary development of solutions such as routers and sessions, the development difficulty and time could be reduced.

Although the entire system is divided into two parts, for users, there is no need to install the program twice because the compiled console is integrated into the server in practical applications, which is possible since JavaScript compiles both the console and the server. Nevertheless, in the development, they are still separated to facilitate program modification.

5. Development of System

In this section, the development of the EV charging management system is introduced in detail. Mainly how the functions modules described in Section 4 are implemented is explained through tools such as core code and flowcharts. Furthermore, the special allocation algorithm is also shown in the following subsections. Equally, the encapsulation of the services of the support tier is also discussed in the final subsection 5.2.

5.1 Console Development

This subsection concentrates mainly on the development of the console, the web-UI of project AGCCS. Although the console only occupies one layer in the logical structure, as shown in Figure 5 in Section 4, it is associated with all functional modules all the functions modules except the data module. Undoubtedly, it plays the most important role in the system for users since it directly interacts with the user and feeds the user's request back to the server to complete the tasks.

All the design documents and code relevant to the console have been published at the GitHub repository of the organization AGCCS, https://github.com/AGCCS/monitor. According to the different interfaces of the console, this subsection is divided into three parts to introduce its development.

5.1.1 Login Interface

The code of the login interface is saved as *'Login.vue.'* Since the login interface has only one function of user login, its code is not complicated. Its implementation applies two components of Element-UI, the 'Form' and the 'Input.' In order to facilitate testing and simulation, the default username and password, 'admin' and '123456', are constantly provided in the input boxes.

Mobile phone users should play a more important role than other users. Therefore, two input attributes, 'clearable' and 'show-password,' with which users can clear the entered content or display the password with one click, are applied for the convenience of mobile phone users. Besides, the size of the entire box is set to width 330 and height 260 through CCS code, which is suitable for mobile phones with more than a 6.26-inch screen.

The login interface is shown in Figure 6, whose outlook is pretty plain due to the limited level of the author's UI design. A placeholder is commented in its code for the possible logo of AGCCS in the future.

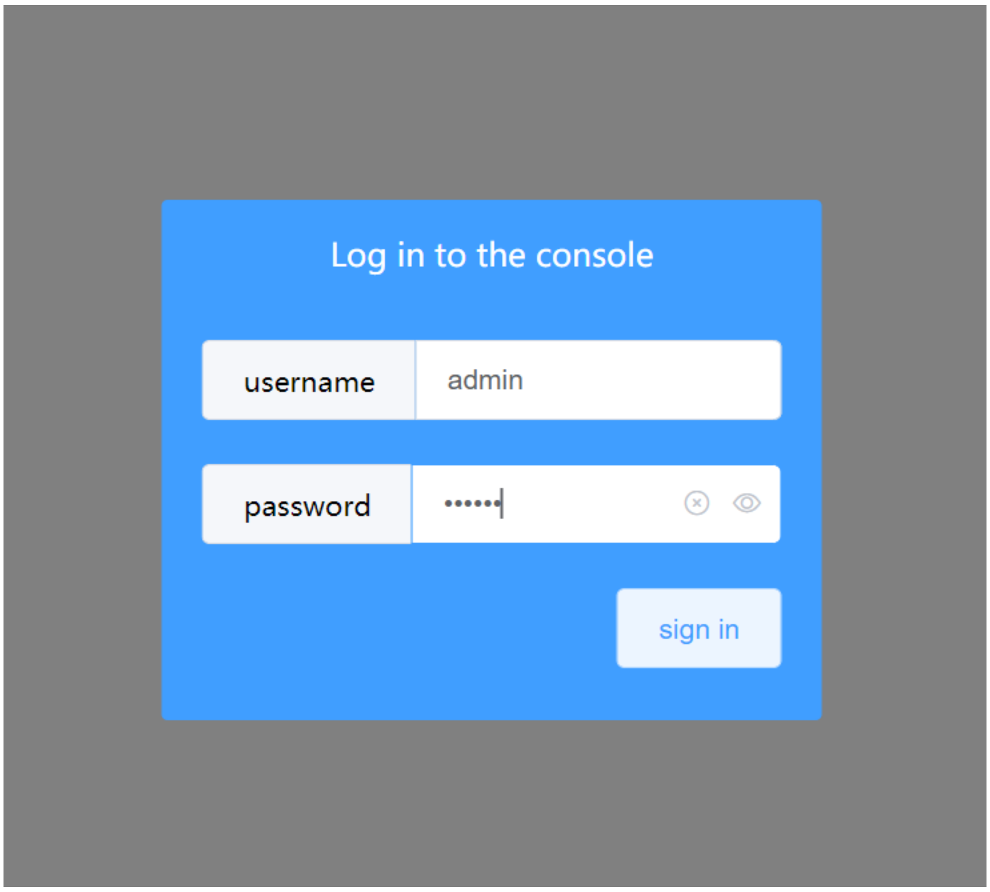


Figure 6: Outlook of the Login Interface

5.1.2 Homepage

After successfully logging in, users enter the homepage of the charging management system, which refers to the Vue file *'Home.vue.'* Meanwhile, the server sends a token to the console to give the user the corresponding permissions. Then the token is stored in the session storage of browsers to prevent illegal access, such as directly entering the URL corresponding to each console in the browser. All the accesses without a token are redirected to the login interface. Additionally, all the requests sent from the console are accompanied by the token. Thus, the server can verify the authorization of the user via token to limit their operations.

Except for the login interface, all the interfaces in the web-UI have the same layout, which applies the component 'Layout Container' of Element-UI. The layout could be divided into three parts, the header above, the side menu for navigation, the main box for displaying the contents of the interfaces. In the header, there is only one button for logging out. The main box applies the special router-view of Vue.js to direct the content to different interfaces via routers, according to the choice of navigation. In this way, the browser could display the contents of different consoles without jumping to a new interface.

There are 4 submenus in the side menu, the mesh submenu, the nodes submenu with options nodes status and nodes info, the user submenu with options change password and create subuser, and the settings submenu with options mesh setting and initialization. By selecting the first two submenus and options, the main box can be directed to the corresponding console interface. Since the latter two do not involve navigation, they are implemented as a new menu in the code. The options in the user menu refer to the remaining two functions of the functional module user, change password and create subuser. With the option mesh setting, in the settings menu, the authorized administrator could set the maximum load current of the entire system. By the way, the default maximum load current is set to 100A for test and simulation. Because the initialization clears all the data of nodes that have been stored in the database, in order to prevent misoperation, users are required to enter the username and password of the host to continue it.

Furthermore, there is an option above all the submenus, which is used to expand or collapse the navigation menu. All the selections could be made whether the menu is expanded or collapsed. The menu is designed to be collapsed by default for the experience of mobile phone users.

5.1.3 Console Interface

There are a total of three interfaces of the console, which are the Mesh monitor, the Nodes Status console, and the Nodes Info console.

The default interface of the charging management system, also shown on the homepage, is named the mesh monitor. The mesh monitor only possesses the function of displaying the information of the whole mesh, such as the total number of nodes, the number of nodes that are charging, the total used power in each phase, and the current capacity of the system.

On the Nodes Status console, users can view the working states of all nodes along with their names. The default name of all nodes is their mac address. To avoid viewing difficulties caused by too many nodes and facilitate users to find specific nodes, three methods are applied in the table of the working status. The first one is a component of Element-UI, ‘Pagination,’ with which users can divide the data into 5, 10, 20, or 40 nodes per page according to their needs. The other two methods come from the native attributes of the table component of Element-UI, ‘sortable’ and ‘filter.’ Since only the connection and the working status of nodes have a fixed value range, only these two types of data can be filtered. The working status refers to the CCSS in the firmware *ctrl22c* and *demesh* [1], which uses only numbers to describe the stage of the charging process.

As shown in Figure 7, in addition to the two abnormal states that have not been described, error and fatal error, there are 6 states: OFF, A, B, C, P, and W. These states correspond to the CCSS number presented in Table 1. If the CCSS is between 90 to 100, it means error, and if it is not in the two intervals of 0 to 60 and 90 to 100, it is an undefined state that should not occur during the charging process, namely fatal error. If the working status is still expressed as a number like it in the firmware, it can be difficult for users to understand. Therefore, the CCSS is converted to Off, Wait EV, Negotiation, Charging, Pause, Wait Power, Err, and Fatal Err, as shown in Table 1. Finally, there are 4 buttons arranged in the last column of the working states table of nodes, the setting button, the OP Button button, the Blink button, and the No Blink button. The setting button is responsible for the function ‘Show and Change the Settings of One Node’ in the functional module Node mentioned above. By clicking the setting button, users can switch the working mode to automatic or manual and check the phase and maximum current allocated to the node. The settings, phase, and maximum current can only be set unless the node is in manual working mode. The last three buttons refer to the function ‘Operate Button and LED on One Node.’

Value Range	**CCSS**	**Working Status**
0-9	OFF[x]	OFF
10-19	A[x]	Wait EV
20-29	B[x]	Negotiation
30-39	C[x]	Charging
40-49	P[x]	Pause
50-59	W[x]	Wait Power
90-99	Error[x]	Err
Others	Fatal Error	Fatal Err

Table 1: CCSS and Working Status of Nodes

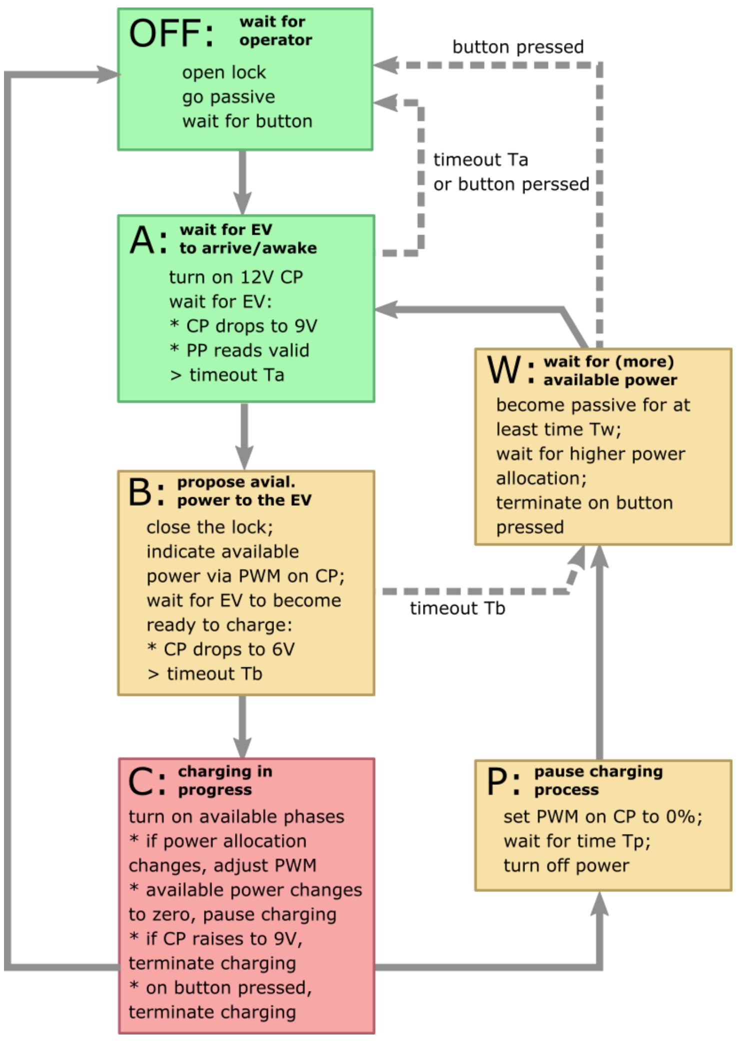
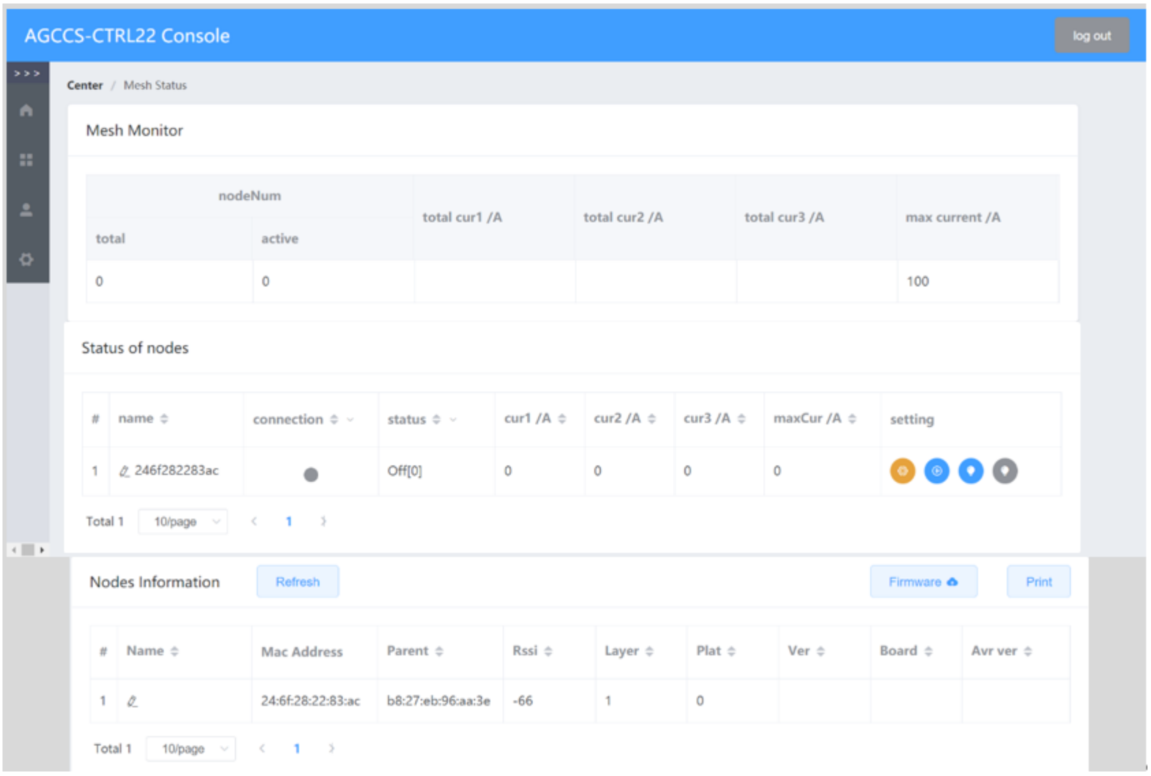

OFF: wait for operator
open lock
go passive
wait for button
button pressed
timeout Ta
or button perssed
A: wait for EV to arrive/awake
turn on 12V CP
wait for EV:
* CP drops to 9V
* PP reads valid
> timeout Ta
W: wait for (more) available power
become passive for at least time Tw;
wait for higher power allocation;
terminate on button pressed
B: propose avial. power to the EV
close the lock;
indicate available power via PWM on CP;
wait for EV to become ready to charge:
* CP drops to 6V
> timeout Tb
timeout Tb
C: charging in progress
turn on available phases
* if power allocation changes, adjust PWM
* available power changes to zero, pause charging
* if CP raises to 9V, terminate charging
* on button pressed, terminate charging
P: pause charging process
set PWM on CP to 0%;
wait for time Tp;
turn off power

Figure 7: Stages of Charging Process [1]

The last Node Info console shows the information of nodes, such as their name, mac address, parent node, received signal strength indicator (RSSI), layer in the mesh, plat in the layer, the version of ESP32 firmware, the development board, and the version of AVR firmware. All these data can be paged or sorted as the data in the Nodes Status console mentioned above. Unlike the other 2 interfaces mentioned above, the Node Info console loads data only once when entering the interface and does not automatically refresh its data. Since this information usually is static, it is unnecessary to load the data repeatedly, which is a waste of the performance. However, there is a Refresh button above the information table in case users need to load the new information of nodes.

Further, the function FOTA is implemented in this console, which is developed via the Element-UI component, ‘Upload.’ After FOTA, the information of nodes can be different. Therefore, the console should reload the data after the success of FOTA. Because there are feedback messages after the AVR firmware is successfully uploaded, the MQTT service via WebSocket is used to subscribe to it for updating the data on time. Subsequently, this console provides users with the Print button to print the data of the information of nodes instead of printing the entire interface, which is implemented through the code based on Print.js [22] developed by the programmer Guangzhi Tan [21].

All the operations except the printing of nodes information are verified for authorization by the server through the token. Only the administrator can carry out the operation. When a subuser conducts an operation, a warning message is displayed on the top side of the browser. Similarly, if someone wants to operate an unconnected node, there is also a warning message with different content. The interfaces, along with the layout of the homepage, are shown in Figure 8.



Mesh Monitor

nodeNum		total cur1 /A	total cur2 /A	total cur3 /A	max current /A
total	active				
0	0				100

Status of nodes

#	name	connection	status	cur1 /A	cur2 /A	cur3 /A	maxCur /A	setting
1	246f282283ac	●	Off[0]	0	0	0	0	

Nodes Information

#	Name	Mac Address	Parent	Rssi	Layer	Plat	Ver	Board	Avr ver
1		24:6f:28:22:83:ac	b8:27:eb:96:aa:3e	-66	1	0			

Figure 8: Outlook of the Interfaces

5.2 Server Development

This subsection is divided into 4 parts, corresponding to the 4 tiers of the server, the middle tier, the application tier, the data tier, and the support tier. Although the support tier is the lowest in the functional architecture of the program, its development is discussed at first since it is the foundation of the application tier and data tier and does not interact with other tiers. The power allocation is developed in the data tier. Thus, its algorithm and the allocation process are shown in the subsection data tier.

All the design documents and code relevant to the development of the server have been published at the GitHub repository of the organization AGCCS, https://github.com/AGCCS/UI-server. The latest version of the compiled program of the console is also integrated into it to simplify the installation.

5.2.1 Support Tier

As shown in Figure 5, the support tier enables the services of Redis, database, and MQTT.

Redis is the most common in-memory database to manage the session store. Its most significant advantage is the fast performance. With Redis, one operation takes less than one millisecond, which makes millions of operations per second still possible. However, to ensure such a fast response of database, users have to install Redis themselves, since the second advancement of software with similar performance can be difficult and may also cause the management system to no longer be lightweight. As a result, in the server, it only is used as the session storage, which is simple with the web framework Express.

The original implementation of the database in the development of the server is based on MySQL, the most popular open-source relational database management system [23], and the Node.js-based driver for MySQL [24]. But in order to minimize the complexity of installation, it is switched to Node.js-based Sqlite3 [20]. Nevertheless, the source code of the driver for MySQL is still required in the program to use its function, 'escape,' to avoid the typical web cyberattack, SQL injection, which focuses on the attack of a database based on SQL programming language. Since the server no longer communicates and cooperates with the external software MySQL, the development of the database software Sqlite3 considerably shortens the response time of the database. For a regular HTTP-GET-request on Raspberry Pi, the response time is reduced from about 20 milliseconds to about 10 milliseconds.

The source code of Sqlite3 is encapsulated into five functions to realize the database service based on Sqlite3. Two for the start and close of the database. One for the creation of new tables in the database. These three functions are designed for the creation of the database of the system. The last two functions can query or modify data. These two functions directly related to data are developed as asynchronous event-driven functions, which make it possible for the server to handle other requests while waiting for the response of the database. With the development of these five functions, the establishment of the database for the EV charging management system has been added to their source code. The data tables of the database are shown in the following tables.

	Column name	Data Type	Primary Key	Not Null	Unique
1	username	VARCHAR(25)	Y	not null	Y
2	password	VARCHAR(64)	Y	not null	N

Table 2: User Table in the Database

	Column name	Data Type	Unique	Default	Comment
1	id	INTEGER	Y	null	
2	wholeMax	SMALLINT	N	100	Maximum current of the mesh
3	manUsedCur1	FLOAT(3)	N	0	Power consumed manual nodes in phase 1
4	manUsedCur2	FLOAT(3)	N	0	Power consumed manual nodes in phase 2
5	manUsedCur3	FLOAT(3)	N	0	Power consumed manual nodes in phase 3
6	manTotalCur1	FLOAT(3)	N	0	Power allocated manual nodes in phase 1
7	manTotalCur2	FLOAT(3)	N	0	Power allocated manual nodes in phase 2
8	manTotalCur3	FLOAT(3)	N	0	Power allocated manual nodes in phase 3

Table 3: Mesh Table in the Database

	Column name	Data Type	Unique	Default	Comment
1	id	INTEGER	Y	null	
2	macADR	VARCHAR(18)	Y	100	Mac address of nodes
3	nodeName	VARCHAR(25)	Y	null	Name of nodes
4	workStatus	SMALLINT	N	null	CCSS of nodes
5	maxCur	FLOAT(3)	N	null	Maximum current of nodes
6	cmaxCur	FLOAT(3)	N	null	Maximum current of cable
7	smaxCur	FLOAT(3)	N	0	Maximum current setting
8	workmode	VARCHAR(10)	N	‘auto’	Work mode, auto or manual
9	Connect	BOOLEAN	N	null	Connection of nodes
10	chargePro	SMALLINT	N	-1	Stages of charging progress
11	Cur1	FLOAT(3)	N	null	Power consumed in phase 1
12	Cur2	FLOAT(3)	N	null	Power consumed in phase 2
13	Cur3	FLOAT(3)	N	null	Power consumed in phase 3
14	Phases	SMALLINT	N	null	Phases used by nodes

15	sPhases	SMALLINT	N	0	Phases setting
16	Parent	VARCHAR(24)	N	null	Parent nodes
17	Rssi	SMALLINT	N	null	Rssi from the parent nodes
18	Layer	SMALLINT	N	null	Layer in the mesh
19	Plat	SMALLINT	N	null	Place in the layer
20	Version	VARCHAR(8)	N	null	Version of firmware *demesh*
21	Board	VARCHAR(45)	N	null	Name of development board
22	avrVer	VARCHAR(8)	N	null	Version of firmware *ctrl22c*

Table 4: Nodestatus Table in the Database

The most important service that the server needs is the MQTT service. Although *demesh*, the firmware for ESP32, also provides communication via TCP/IP socket, MQTT could be better for an IoT project like AGCCS. The MQTT communication is based on the publish-subscribe model, which means the nodes do not always need to maintain a two-way channel with the broker, thereby reducing the pressure of the network. MQTT is also a lightweight communication protocol, which makes it possible for numerous nodes to work simultaneously. In the early stages of the server development, the MQTT-broker, Mosquitto [25], is applied for test and simulation, which reduces the difficulty considerably. For the same consideration as above, an MQTT-broker based on Aedes [19] is developed and integrated into the server. In order to facilitate the subsequent maintenance and development in the future, the origin functions for client connection, disconnection, subscription, and publication in Aedes are encapsulated to call their callback functions to check related information.

5.2.2 Middle Tier

As presented in Figure 5, the code of the middle tier is made of two parts, the restful application programming interface (API) and the routers. The restful API is the URL of the routers to which the console sends requests. It is declared in the default main program file of Express, 'app.js.' Besides the URL of the routers, a simple error handler is applied to it. The port of the API is defined in Express's default entry file 'www.' The response of the server is sent to the console through the restful API. An overview of the API interface can be found in Attachment C.

There are 4 routers in total, corresponding to the 4 functional modules involved in the console, the user router, the mesh router, the node router, and the upload router for the firmware module. In order to call the router and its function correctly, the destination URL of a request should contain the name of the router and the name of the function that the router needs to process in addition to the basic URL specified by the API. If the request is an HTTP-post request or an HTTP-put request, namely an operation on the console, it should be verified for authorization before the router starts to work. The verification is implemented through a middleware, ‘adminCheck.’ Since all routers share the same middleware, it is developed separately as one file, ‘adminCheck.js,’ so that there is no need to implement it multiple times in the routers. Except for the upload router, the other three routers all work the same way, analyzing the request and passing the data to the function in the corresponding function module developed in the application tier. The upload router first stores the files uploaded by the console in the static file storage folder of the server. Then it sends a response to the console indicating that the file was uploaded successfully. Subsequently, the console sends another request containing the necessary data of the firmware, such as the version of the firmware and the mac address of the destination node. Finally, the router starts the FOTA function.

5.2.3 Application Tier

By comparing Figure 4 and Figure 5, it is known that most of the functions of the system are implemented in the application tier. In this section, the five functional modules involved in the application layer are explained separately.

- **User Module:**
  The development of the user module is divided into two parts. One part is named ‘pwdContorl.js’ for the generating of encrypted passwords. This function is developed to generate not only the password in the user module but also the original password of the administrator during the database establishment. There is a decryption function of passwords in the code, but it is not used so far. The development of encryption and decryption has adopted the examples of Advanced Encryption Standard (AES) provided by node.js in crypto [26]. The encryption algorithm AES [27] has the advantages of high speed and low hardware requirements, which makes it very suitable for the project AGCCS. The other part is named ‘userControl.js’ to handle the data transmitted by the user router. There are 3 main functions in it, which can verify login information, modify passwords and create sub-users. While verifying login information, the server also generates a token for the authorization. All passwords for the interaction with the database are encrypted by the function, password generation, provided by ‘pwdControl,js’ before submission since the

password stored in the database is an encrypted password, which could preserve the confidentiality of users. For the same reason, the decryption of passwords is not required.

- **Mesh Module:**
  The implementation of functions of the mesh module is simple. There are 3 functions in it, which could initialize the mesh, query the data of mesh and modify the only setting of the mesh, the maximum available current value of the mesh. The initialization of mesh does not mean the initialization of the whole system but only the clearing of data of nodes.
- **Node Module:** ALL the functions in the node module are developed to handle the requests from the console. There are functions to handle the HTTP-GET-request, which show the data, information, or settings like phases of nodes, and functions to handle the HTTP-PUT-request or HTTP-POST-request, which set the name or settings of nodes. Although the control function belongs to the node module, it is developed in the communication module since it needs to publish corresponding commands through MQTT. There are numerous cases of how the node settings are changed. For instance, the settings of nodes, such as their maximum current and phases, could be changed during the charging process. As a result, the power allocation must be changed. Besides, the settings may be invalid, such as exceeding the capacity or no allocable power. These complicated situations improve the difficulty of its development. The flowcharts in Figure 9 and Figure 10 explains how it works in different cases. The setting's validity is verified by calculating whether the power allocation according to the setting could cause the exceed of the mesh's capacity. If the work status of the node, namely CCSS, is between 20 and 60, it means that there is already power allocated to it. Therefore, this part of power is required to return to the mesh for validity checking. Additionally, in order to improve the automaticity of the program, the node module should adjust the maximum giving current to its value if there is power available for allocation in the grid with the given phases (the residual current is more than 5A).
- **Firmware Module:**
  Although the firmware module is assigned to the application tier, it is advanced with the upload router. As mentioned above, its primary function is FOTA. For the firmware *demesh* for ESP32, the firmware module sends a command to call the root node to download the firmware in the static file storage folder of the server via HTTP-GET-request. In this case, the communication between the server and nodes will not only be through MQTT but also HTTP. The process of the uploading of firmware *ctrl22c* for AVR is complicated. In the first step, the firmware module sends a command to the node to inquire whether the firmware

of its AVR can be updated. After receiving a positive reply, the module divides the firmware into several 128-bit segments, uses crc32 [28] to encode these segments, and sends them to the node. Except for the first fragment, each fragment is sent after receiving the node's response that the CRC verification is successful. Finally, the module ends the MQTT client to end the function after the node tells the server that the new version of the firmware *ctrl22c* is running successfully.

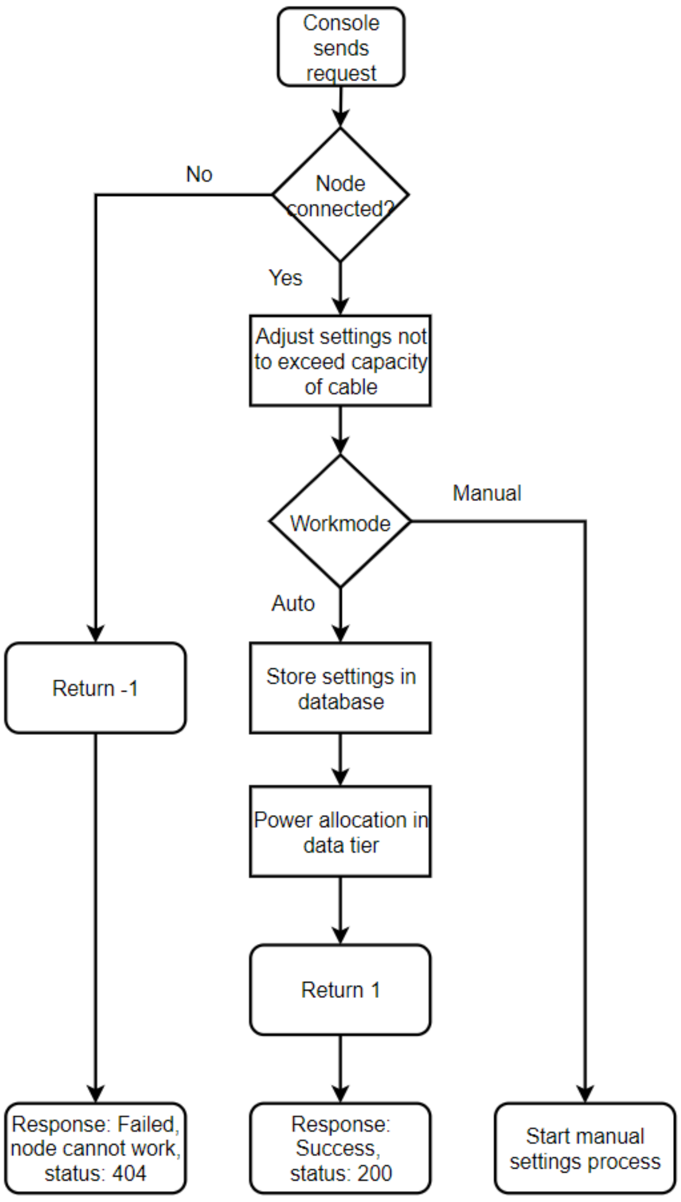

Console sends request
Node connected?
No
Yes
Adjust settings not to exceed capacity of cable
Workmode
Manual
Auto
Return -1
Store settings in database
Power allocation in data tier
Return 1
Response: Failed, node cannot work, status: 404
Response: Success, status: 200
Start manual settings process

Figure 9: Process of Changing Node Settings without Manual Working Mode

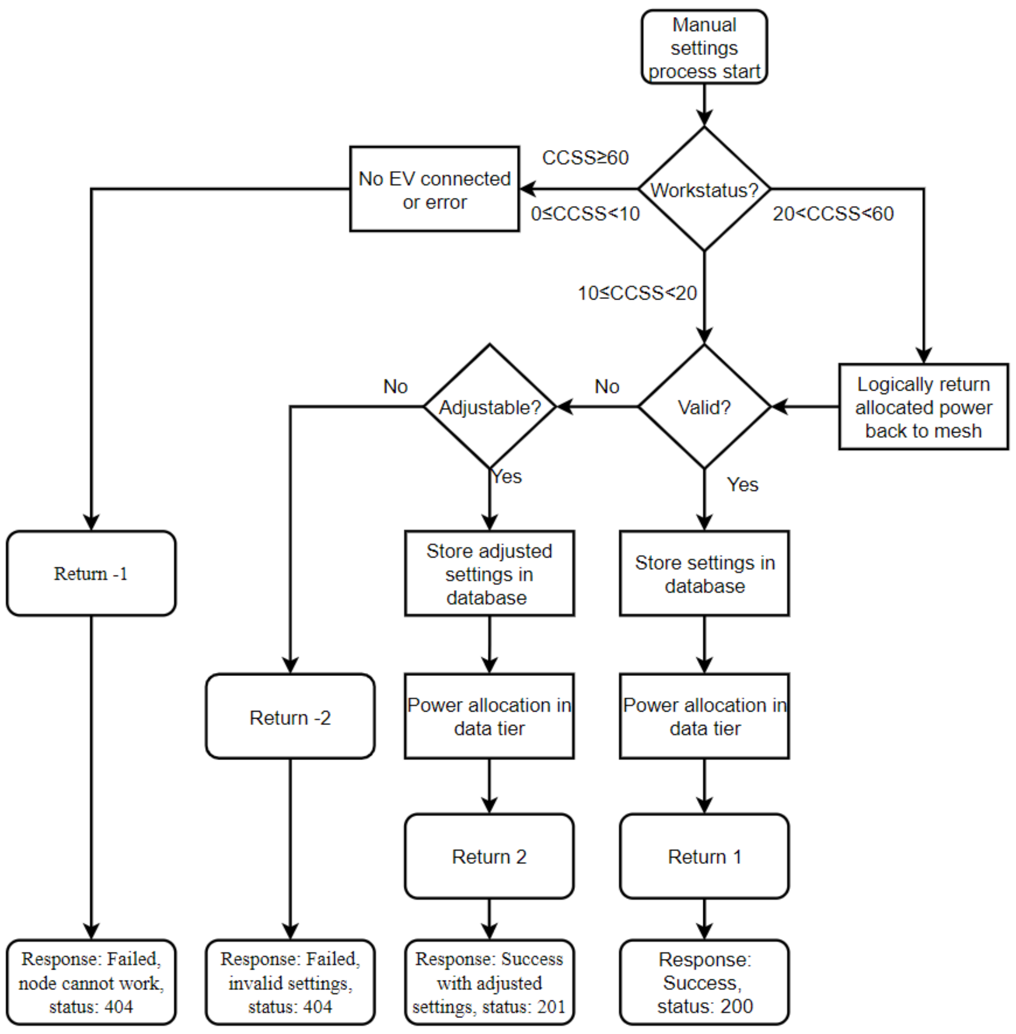

Manual settings process start
Workstatus?
CCSS≥60
0≤CCSS<10
20<CCSS<60
10≤CCSS<20
No EV connected or error
Logically return allocated power back to mesh
Valid?
No
Yes
Adjustable?
No
Yes
Return -1
Return -2
Store adjusted settings in database
Store settings in database
Power allocation in data tier
Power allocation in data tier
Return 2
Return 1
Response: Failed, node cannot work, status: 404
Response: Failed, invalid settings, status: 404
Response: Success with adjusted settings, status: 201
Response: Success, status: 200

Figure 10: Process of Changing Node Settings in Manual Working Mode

- **Communication Module:**

  The communication module involves all the functions that require interaction with nodes. On the one hand, it subscribes the message published by nodes, and then process the data and store it in the database. On the other hand, it publishes commands to the node according to the needs of other functions to operate buttons, blinking LEDs, or allocate power. Therefore, the functions of the communication module can be summarized as a subscription module and a publishing module. In the subscription module, one of the essential functions, the heartbeat read function, is implemented to receive the heartbeat data of nodes and check the connection of nodes simultaneously. Once the server receives no heartbeat data from one node for a period, which is currently set to 15 seconds by the program, this node is marked as disconnected. Equally, any time the server receives the heartbeat data of one node, then this node is marked as connected. After the data is received, the subscription module will pass it to the data and activate the corresponding function. Besides, the heartbeat read function can register the information of a new node into the database. The process of the publication module is simple. It sends commands via MQTT to the corresponding nodes once receiving requests and data from the upstream middle tier or the downstream data tier. These commands include setting the maximum current and phase of the node, as well as controlling button and LED. Finally, all MQTT clients except for the clients subscribing to heartbeat data should end their process after completing their tasks to save the system's hardware resources.

5.2.4 Data Tier

The data tier is the most complicated in the server. On the one hand, it is the agent for the interaction between the database and the application tier, which means it needs to provide services for multiple functions in the application layer. On the other hand, the power allocation is implemented in this tier for better interaction with data. Therefore, the development of the data tier is divided into three parts to describe its functions clearly.

First of all, the data tier provides the communication module and the node module with interfaces of its functions. Unlike the other functional module, the communication module is designed to have no ability to interact with the database to reduce the complexity of its code. Since these interfaces are not related to the middle layer but directly related to the node, this tier is developed separately from the application layer. Seven functions of it are required by the communication module as follows. As shown in the flowcharts in Figure 9 and Figure 10, the changes in node settings often cause changes in power allocation. Therefore, the interfaces of the power allocation function in the data tier are offered to the node module in the application tier.

- **macReg:** Register the mac address of a new node into the database and set its working mode to automatic at the same time.
- **getList:** Return the id and mac address of all nodes.
- **getInfo:** Return all the data of one node.
- **currentUpdate:** Use the data from the communication module to update the charging data of the node and judge its charging progress.
- **connectUpdate:** Update the connection of a node and reset the charging data of a disconnected node.
- **statusUpdate:** Update the working status of a node, namely its CCSS.

- **InfoUpdate:** Update the information of a node as described in Subsection 5.1.3, such as its parent node and its development board.

An essential function of the data tier is the power allocation. It is divided into two steps since there are two working modes of the node. The first step, which is much simpler than the other one, is to allocate power for nodes in manual working mode. Due to the priority of manual working mode, the program should directly allocate power according to their set maximum current and phase. During the allocation, the power consumed by nodes in manual working mode and the power allocated to them is summarized for the subsequent power allocation of nodes in auto working mode.

In the power allocation of nodes in auto working mode, a local optimal algorithm based on the greedy algorithm is designed to make the charging efficiency as high as possible. The primary idea behind this algorithm is to allocate the average power of the maximum power to each node. Since 5A is the minimum current required for charging, the algorithm ignores some EVs to be charged if the average current is less than 5A. To find the optimal local solution, 5 cases are considered. In case 1, the algorithm uses as many phases as possible to provide the node with the smaller one of the average current and its cable capacity. In cases 2, 3, and 4, the algorithm allocates the smaller one of the current of one phase or the cable capacity of the node to it. Case 5 is an extension of case 1. It is designed to prevent the available current of one of the three phases from being less than 5A after the allocation is completed, leading to a waste of power. In this case, the allocated phases are the same as in case 1, but the current becomes the smallest one of the current values in the allocated phases and the cable capacity of the node. Figure 11 shows how the power allocation works in case 1.

After the allocation solutions in all 5 cases are calculated, the algorithm eliminates solutions whose power is greater than the average power. It then selects the solution with the largest power among the remaining solutions as the optimal local solution. Finally, the server sends commands to the node to set its phases and maximum current via MQTT communication.

Since the charging current of EVs drops when they are about to be fully charged, it can cause a waste of power if the program still allocates the same maximum current to it. Thus, the power allocation should be dynamic to prevent such waste. As described above, the program judges the charging progress of a node during the update of the charging node. The dynamic solution shown is developed based on the charging progress. After a node starts charging, the system waits for its current value to reach its maximum current. Then the node is marked as charging with full power. Once the current of the marked node falls below its maximum current, the server sets a new maximum current according to the difference between its current and its maximum current. Finally, the system restarts the power allocation algorithm to use the saved power. In order to maintain the results of the above dynamic solution, the power allocation algorithm is only for the nodes with CCSS between 20 and 29, which means they are waiting for allocation in the power negotiation phase, as shown in Table 1.

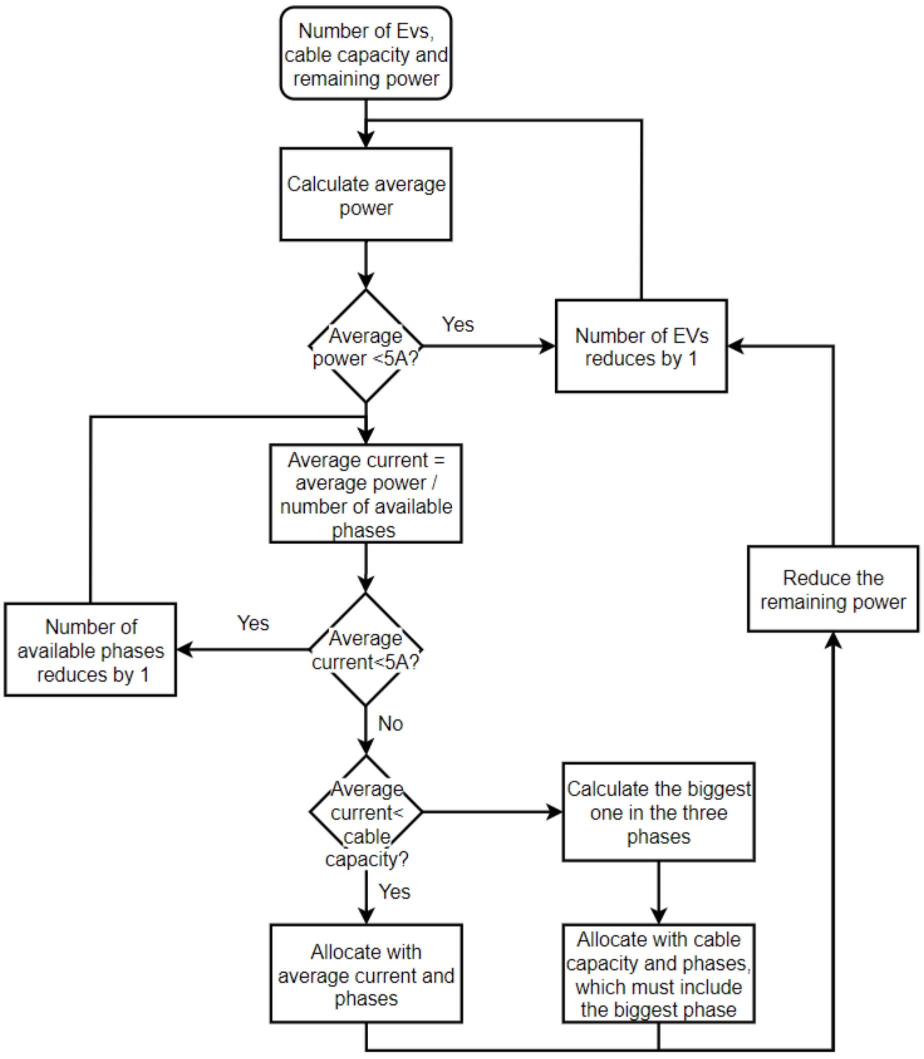

Number of Evs, cable capacity and remaining power
Calculate average power
Average power <5A?
Yes
Number of EVs reduces by 1
Average current = average power / number of available phases
Reduce the remaining power
Number of available phases reduces by 1
Yes
Average current<5A?
No
Average current< cable capacity?
Calculate the biggest one in the three phases
Yes
Allocate with average current and phases
Allocate with cable capacity and phases, which must include the biggest phase

Figure 11: Case 1 in the Power allocation

6. Test and Simulation

The testing and simulation process progresses gradually with the development of the system, and the equipment used in the simulation also changes accordingly. There are 3 kinds of development board, the M5stickC [3], the barebone board and the development board of AGCCS, and a lamp that replaces electric vehicles as the load used in the simulation. The entire process of the simulation can be divided into three stages. Therefore, this section introduces it in 3 subsections.

6.1 Simulation on M5StickCs

The first step of the test was conducted on M5StickC [3], which is a mini development board of ESP32 and integrates a 0.95-inch LCD screen, a red LED, two buttons, and other devices which are not applied in the simulation. As mentioned in Section 2, there is an operation button on the housing of the AGCCS hardware, which should be pressed to start the power allocation. In other words, the working status of the node shown in Table 1 changes from 'wait EV' to 'negotiation' after pressing the operation button. Besides, the LED could tell the user the status of the node. For example, when the node is looking for the mesh, its LED blinks. Accordingly, the M5StickC could provide almost all the functions of the AGCCS development board except for the actual charging. However, a charging process is essential since the most important function of the EV charging management system is to control the charging process of EVs. Hence, a fantasy charging process is designed to test the charging control function. After pressing the button to indicate that the electric vehicle is connected, the M5Stick sends the corresponding heartbeat data according to the set maximum current and phase. The current value in the heartbeat data begins to decay after a certain period, indicating that the EV is about to be fully charged.

The test on M5stickCs was done on a personal computer instead of a Raspberry Pi to facilitate the development. Nevertheless, the Raspberry Pi was still required as the main access point of the mesh and the MQTT-broker. This stage is not only the first stage of simulation but also the early stage of system development. Therefore the program lacks some functions such as FOTA and the provision of MQTT services. As a result, only the functions of the user module, the mesh module, the node module, and the data module were tested.

During the simulation, 10 M5stickCs worked at the same time to form a mesh. The program has demonstrated its ability to control the charging processes of at least ten nodes. All the other basic functions of the user module, the mesh module, and the node module have been verified successfully. The efficiency of power allocation was tested by randomly changing the settings of the nodes in the manual mode numerous times. At the beginning of the development, only case 1 mentioned in Subsection 5.2.4, the typical case of the power allocation, was considered. By repeating the test with different settings, it was noticed that in some extreme situations, for example, the available current of one phase is significantly less than that of other phases like 20A and 100A, the typical case could not provide the optimal power allocation for the entire mesh. Therefore, the other four cases are designed to handle the possible extreme situations.

6.2 Simulation on Barebone Boards

After the basic framework of the EV charging management system was completed, it is no longer possible to test all the system's functions using only M5sickC. Therefore, a barebone board circuit based on the J5adaptor of the project AGCCS [1] was built on a breadboard. The circuit diagram of the J5adaptor is shown in Figure 13 in Attachment B. By applying the J5adaptor circuit, the same connection between ESP32 and ATmega4808 in the circuit developed by Pascal Thurnherr can be constructed. Nonetheless, there is still no actual load that can be used for charging in the simulation. As a result, the fantasy charging process in the previous simulation stage was still used to test the program's performance.

As the development progresses, it is required to reduce the difference between the simulated charging process and the actual charging process so that the power allocation could suit the requirements of the project AGCCS. Based on the analysis of the actual charging process, the working mode of a node is switchable only when an EV is connected to the node and is automatically switched back to the auto working mode after the EV is charged. This setting is for the server to obtain the cable capacity of the node before power allocation. In the simulation, pressing the operation button once indicates that the EV is connected.

Due to the provision of the MQTT service developed based on Aedes [19], the test was done on Raspberry Pi instead of a personal computer. During the simulation, a barebone board mentioned above and 10 M5stickC were used to construct a mesh. Additionally, a computer and a mobile phone were used for logging into the console. The simulation result on barebone boards has proved that under the existing conditions, the system's functions are sufficient to fulfill the requirements of the project AGCCS. Both the firmware *demesh* and *ctrl22c* have been successfully uploaded to the ESP32 of all development boards and the ATmega4808 of the barebone board. The MQTT service has also been confirmed to be available, but its performance has not been effectively tested.

6.3 Laboratory test

The previous test and simulation have proved the performance of the EV charging management system with a fantasy charging process. However, the actual charging process of an EV could be significantly more complicated. As shown in Table 1 and Figure 7 in Subsection 5.1.3, the firmware *ctrl22c* specifies 6 kinds of CCSS for the general charging process. The P[x] and W[x] in CCSS are designed to handle situations, such as power failure or phases reset, where the charging process should be interrupted except for errors and fatal errors in CCSS. However, during the previous simulation, P[x] and W[x] were not considered since a fantasy failure is usually meaningless. As a result, a simulation with actual load is necessary to prove its performance.

To facilitate operation and improve safety, a lamp was used to replace the EV for laboratory testing. Since its power consumption is much lower than the EV, the test did not go well at the beginning. In auto working mode, the lamp could not light up because CCSS changed to W[6] instead of C[x], which means that the charging node waited for a higher power allocation, although the management system allocated 16A of power per phase to it. In CCSS W[x], its maximum feedback current became 0A, and thus the system also reset its allocated current to 0A, and then the lamp could never light up with auto working mode. Unlike EVs, the lamp only consumes a single-phase current. Therefore, the firmware on the AGCCS board continued to require the other two-phase currents that the lamp could not consume. Although the auto working mode could not make the lamp turn on, the user could still allocate a single-phase power to the lamp through the manual working mode.

The result of the first phase of the laboratory test did not meet expectations. However, it proves that the charging management system can cope with failures. In the second phase, the firmware on the AGCCS board was adjusted to support the lamp’s unique features, much lower power consumption than EVs and single-phase current only. With the adjusted firmware, all functions of the program worked as designed.

Regardless of the unsatisfactory result of the first stage because of the difference between lamps and EVs, the charging management system completed the work required by the project AGCCS as designed. It can be expected that it is capable of charging an EV. The program may have some flaws but should not cause any fatal failure.

7. Conclusion and Future Work

If the current of each phase of the three-phase electricity is equal, the power allocation algorithm developed in this paper can find the globally optimal solution for the system. Conversely, it could only find locally optimal solutions for nodes, which means that the EV charging management system developed in this paper may not achieve the highest efficiency when there are nodes in manual working mode. Some typical algorithms for optimization problems such as greedy algorithm, dynamic programming, and nonlinear programming in static optimization have already been considered. The power allocation algorithm applied in the system is based on the greedy algorithm, as mentioned in Subsection 5.2.4. Since the optimal solutions to the sub-problems, namely the power allocation strategy for each node, cannot lead to the globally optimal solution and the necessary KKT conditions in the nonlinear programming are not met, both algorithms are not applicable. Accordingly, an algorithm to calculate the globally optimal solution may be better to achieve higher efficiency.

As mentioned in the introduction of project AGCCS, it aims to provide dynamic power allocation instead of inefficient charging with fixed power allocation. The power allocation process in this paper can improve efficiency to a certain extent. However, such allocation is not enough to be called dynamic allocation. Control methods such as PID control or state-space representation are not designable since there are not enough cases to analyze the charging process of nodes. In the future, there could be more studies on the charging process with the AGCCS hardware to optimize the power allocation method.

The results of the 3 tests have proved the performance of the software developed in this paper as a charging management system for the project AGCCS. To enable anyone with a certain programming ability to develop and maintain this system, a brief introduction of the program and this paper are published at the GitHub organization, https://github.com/AGCCS.

Attachment A – M5tickC

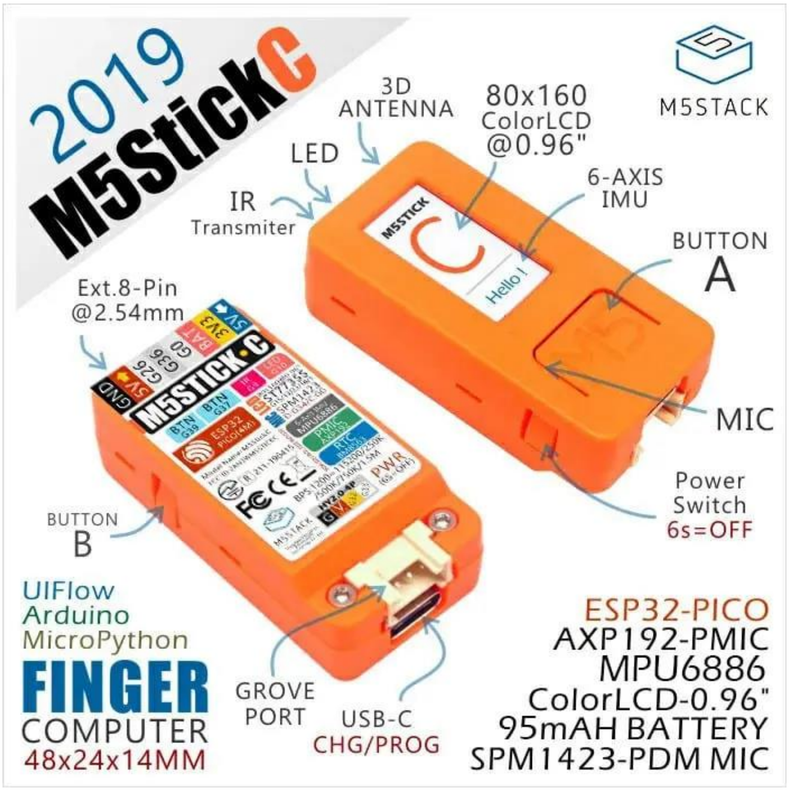


Figure 12: M5stickC [3]

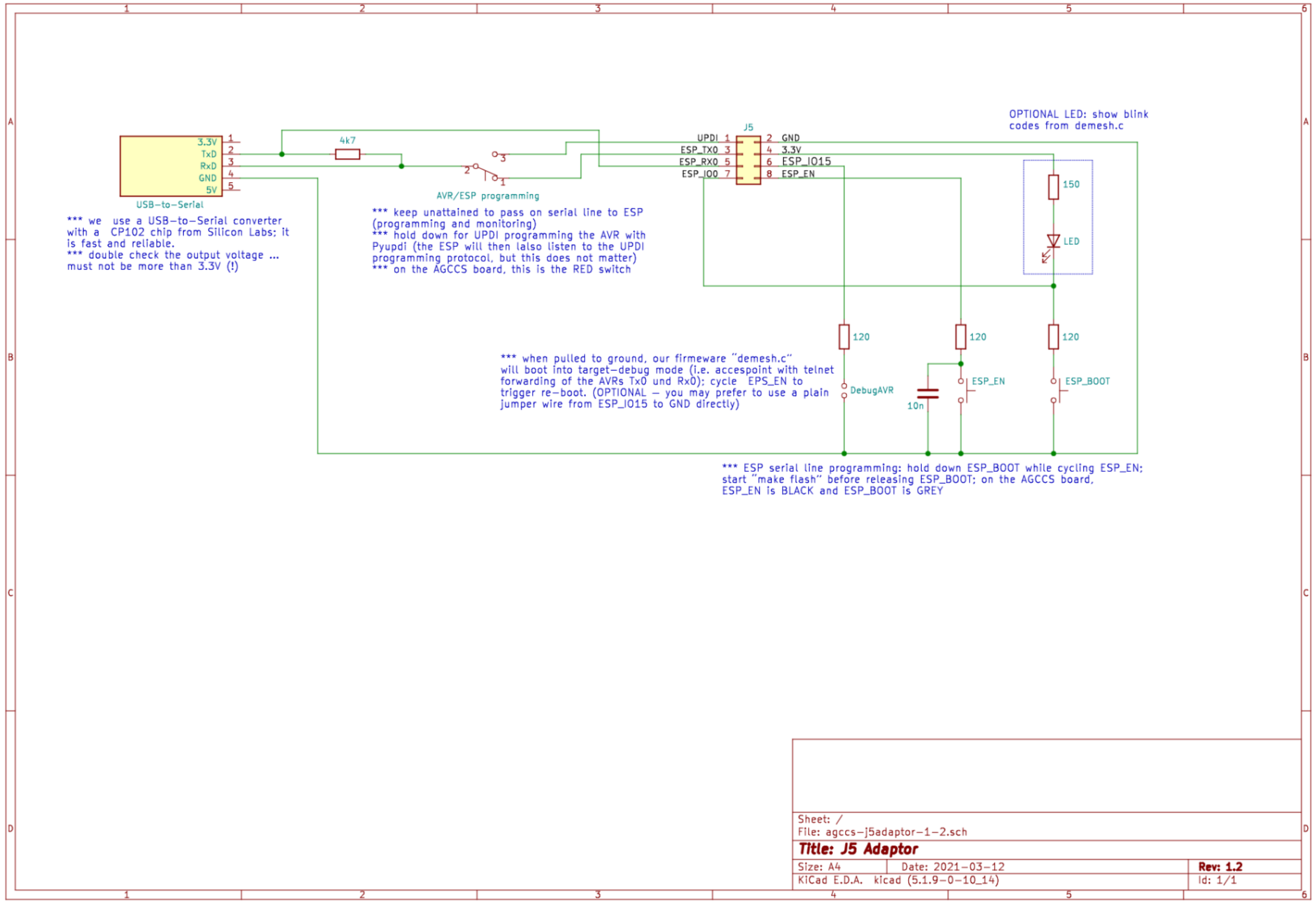
Resources	**Parameter**
ESP32	240MHz dual core, 600 DMIPS, 520KB SRAM, Wi-Fi, dual mode Bluetooth
Flash Memory	4MB
Power Input	5V @ 500mA
Port	TypeC x 1, GROVE(I2C+I/0+UART) x 1
LCD screen	0.96 inch, 80*160 Colorful TFT LCD, ST7735S
Button	Custom button x 2
LED	RED LED
MEMS	MPU6886
IR	Infrared transmission
MIC	SPM1423

RTC	BM8563
PMU	AXP192
Battery	95 mAh @ 3.7V
Antenna	2.4G 3D Antenna
PIN port	G0, G26, G36
Operating Temperature	32°F to 104°F (0°C to 40°C)
Net weight	15.1g
Gross weight	33g
Product Size	48.2*25.5*13.7mm
Package Size	55*55*20mm
Case Material	Plastic (PC)

Table 5: Specification of M5stickC [3]

Attachment B – Circuit Diagram of J5adaptor

x`



OPTIONAL LED: show blink
codes from demesh.c

*** we use a USB–to–Serial converter
with a CP102 chip from Silicon Labs; it
is fast and reliable.
*** double check the output voltage ...
must not be more than 3.3V (!)

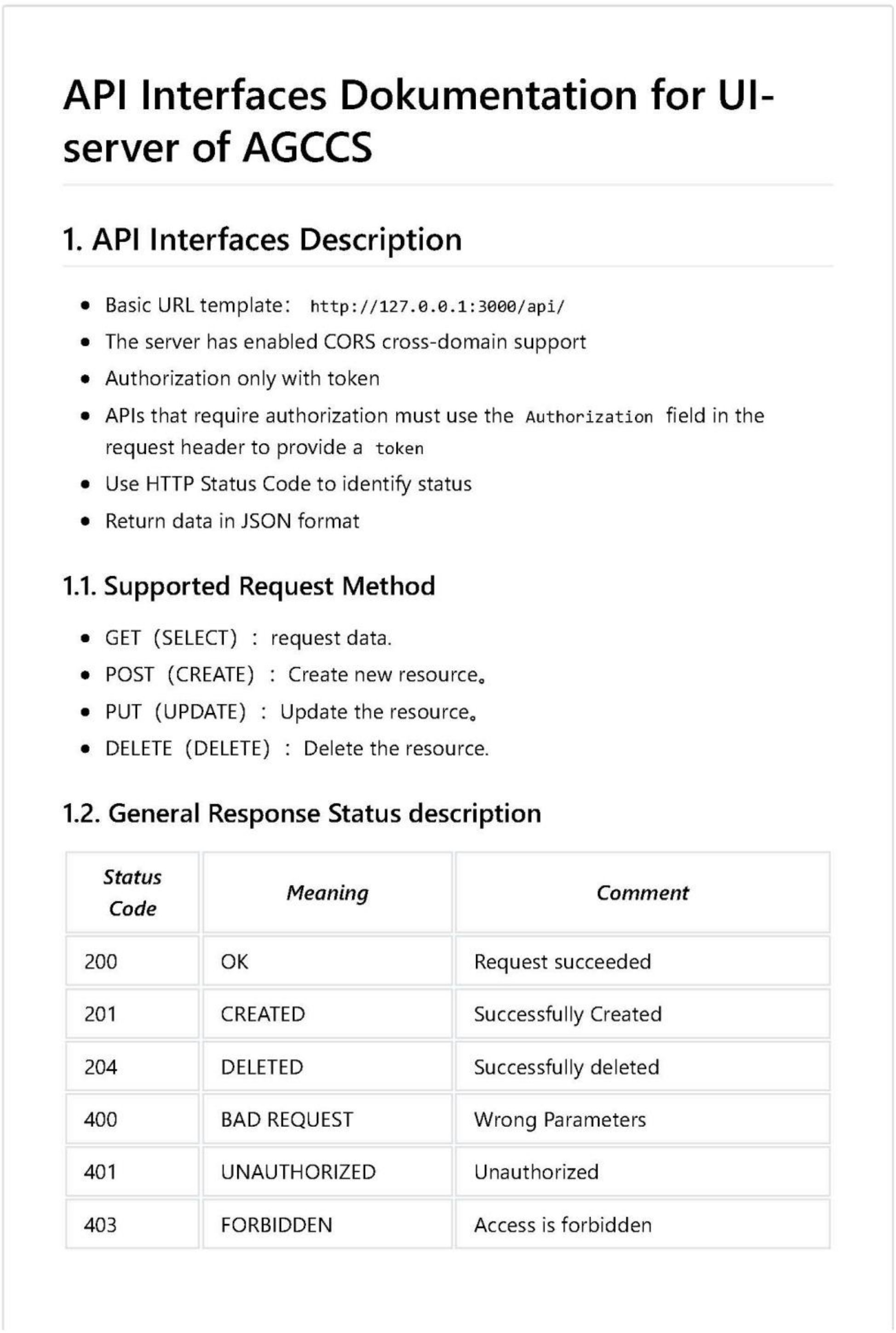
*** keep unattained to pass on serial line to ESP
(programming and monitoring)
*** hold down for UPDI programming the AVR with
Pyupdi (the ESP will then lalso listen to the UPDI
programming protocol, but this does not matter)
*** on the AGCCS board, this is the RED switch

*** when pulled to ground, our firmware "demesh.c"
will boot into target–debug mode (i.e. accespoint with telnet
forwarding of the AVRs Tx0 und Rx0); cycle EPS_EN to
trigger re–boot. (OPTIONAL – you may prefer to use a plain
jumper wire from ESP_IO15 to GND directly)

*** ESP serial line programming: hold down ESP_BOOT while cycling ESP_EN;
start "make flash" before releasing ESP_BOOT; on the AGCCS board,
ESP_EN is BLACK and ESP_BOOT is GREY

Figure 13: Circuit of J5adaptor [3]

Attachment C – API Documentation



API Interfaces Dokumentation for UI-server of AGCCS

1. API Interfaces Description

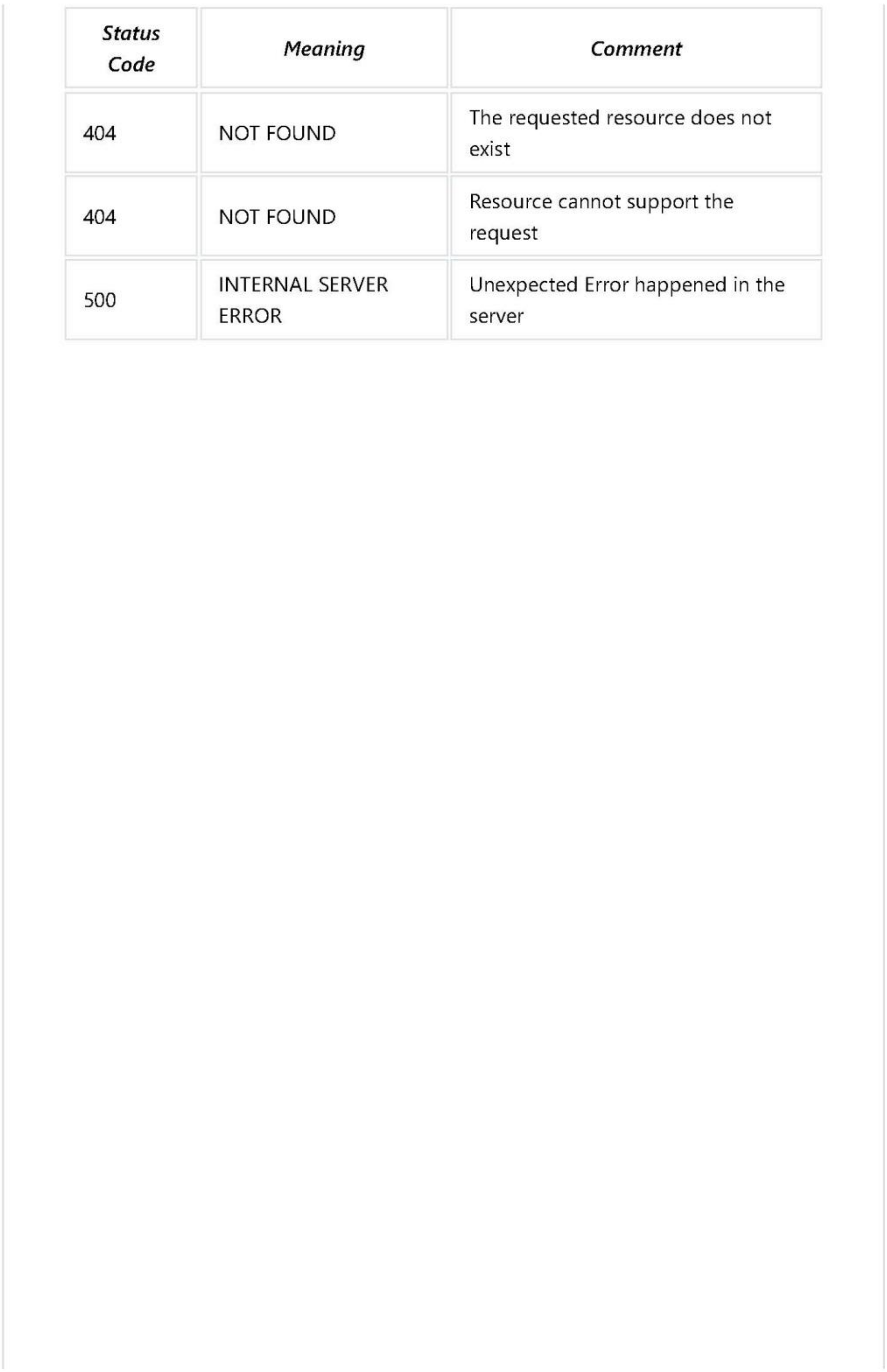
- Basic URL template: `http://127.0.0.1:3000/api/`
- The server has enabled CORS cross-domain support
- Authorization only with token
- APIs that require authorization must use the `Authorization` field in the request header to provide a `token`
- Use HTTP Status Code to identify status
- Return data in JSON format

1.1. Supported Request Method

- GET (SELECT) : request data.
- POST (CREATE) : Create new resource。
- PUT (UPDATE) : Update the resource。
- DELETE (DELETE) : Delete the resource.

1.2. General Response Status description

Status Code	*Meaning*	*Comment*
200	OK	Request succeeded
201	CREATED	Successfully Created
204	DELETED	Successfully deleted
400	BAD REQUEST	Wrong Parameters
401	UNAUTHORIZED	Unauthorized
403	FORBIDDEN	Access is forbidden



Status Code	*Meaning*	*Comment*
404	NOT FOUND	The requested resource does not exist
404	NOT FOUND	Resource cannot support the request
500	INTERNAL SERVER ERROR	Unexpected Error happened in the server

2. Users

2.1. Login Authentication

- Request url： users/login
- Method： post
- Request Parameters:

Parameter	Description	Type	Comment
username		String	Not null
password		String	Not null

- Response data:

Parameter	Description	Type	Comment
username		String	
password		String	Return as meaningless content
token		String	jsonwebtoken

2.2. Password Change

- Request url： users/password
- Method： post
- Request Parameters:

Parameter	Description	Type	Comment
username		String	Not null
password		String	Not null
newPassword		String	Not null

- Response data:

Parameter	Description	Type	Comment

2.3. Add Subuser

- Request url： users/addUser
- Method： post
- Request Parameters:

Parameter	Description	Type	Comment
subUsername	Username of subuser	String	Not null
subPassword	Password of subuser	String	Not null

- Response data:

Parameter	Description	Type	Comment

3. Mesh

3.1. Query Settings of Mesh

- Request url： mesh/setting
- Method： get
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment
id	id of mesh setting	Number	Kept for possible setting template
wholeMax	Maximum current in each phase of the entire charging station	Number	
manUsedCur1	Current consumed by nodes in manual mode in phase 1	Number	Not shown, but kept
manUsedCur2	Current consumed by nodes in manual mode in phase 2	Number	Not shown, but kept
manUsedCur3	Current consumed by nodes in manual mode in phase 3	Number	Not shown, but kept
manTotalCur1	Current allocated to nodes in manual mode in phase 1	Number	Not shown, but kept
manTotalCur2	Current allocated to nodes in manual mode in phase 2	Number	Not shown, but kept

Parameter	Description	Type	Comment
manTotalCur3	Current allocated to nodes in manual mode in phase 3	Number	Not shown, but kept

3.2. Change Settings of Mesh

- Request url： mesh/setting
- Method： put
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment

3.3. Mesh Data Initialization

- Request url： mesh/init
- Method： delete
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment

4. Nodes

4.1. Query Information List of Nodes

- Request url： nodes/list
- Method： get
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment
id	Id of nodes	Number	
nodeName	Name of nodes	String	Default is null
macADR	Mac address of nodes	String	Mac address not separated by colon
connect	Connection of nodes	Boolean	0 for no connection
Parent	Mac adress of their parent node	String	
Rssi	Received signal strength indication	Number	
Layer	Layer in the mesh	Number	
Plat	Number in the layer	Number	
Version	Version of firmware demesh	String	
Board	Development board of nodes	String	
avrVer	Version of firmware ctrl22c	String	

4.2. Query Information of one Particular Node

- Request url： nodes/list
- Method： get
- Request Parameters:

Parameter	Description	Type	Comment
id	Id of the node being queried	Id in the form of url query	

- Response data:

Parameter	Description	Type	Comment
id	Id of the node	Number	
nodeName	Name of the node	String	Default is null
macADR	Mac address of the node	String	Mac address not separated by colon
connect	Connection of the node	Boolean	0 for no connection
Parent	Mac adress of their parent node	String	
Rssi	Received signal strength indication	Number	
Layer	Layer in the mesh	Number	
Plat	Number in the layer	Number	
Version	Version of firmware demesh	String	
Board	Development board of the node	String	
avrVer	Version of firmware ctrl22c	String	

4.3. Query Working Status of Nodes

- Request url： nodes/status
- Method： get
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment
id	Id of nodes	Number	
macADR	Mac address of nodes	String	Mac address not separated by colon
connect	Connection of nodes	Boolean	Boolean, 0 for no connection
Cur1	Current value in phase 1	Number	
Cur2	Current value in phase 2	Number	
Cur3	Current value in phase 3	Number	
workStatus	CCSS of nodes	Number	
sPhases	Version of firmware demesh	Number	
smaxCur	Development board of the node	Number	
workmode	Working mode of nodes	String	Default is auto
maxCur	Maximum Current of nodes	Number	
Phases	Phases occupied by nodes	Number	

Parameter	Description	Type	Comment
cmaxCur	Cable capacity	Number	
nodeName	Name of the node	String	Default is null

4.4. Query Working Status of one Paticular Node

- Request url: nodes/status
- Method: get
- Request Parameters:

Parameter	Description	Type	Comment
id	Id of the node being queried	Number	Sent in the form of url query

- Response data:

Parameter	Description	Type	Comment
id	Id of the node	Number	
macADR	Mac address of the node	String	Mac address not separated by colon
connect	Connection of the node	Boolean	0 for no connection
Cur1	Current value in phase 1	Number	
Cur2	Current value in phase 2	Number	
Cur3	Current value in phase 3	Number	
workStatus	CCSS of the node	Number	
sPhases	Version of firmware demesh	Number	
smaxCur	Development board of the node	Number	

Parameter	Description	Type	Comment
workmode	Working mode of the node	String	Default is auto
maxCur	Maximum Current of the node	Number	
Phases	Phases occupied by the node	Number	
cmaxCur	Cable capacity	Number	
nodeName	Name of the node	String	Default is null

4.5. Operation Button of one Paticular Node

- Request url： nodes/buttonB
- Method： put
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment

4.6. Blink one Paticular Node

- Request url： nodes/Blink
- Method： put
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment

4.7. Stop Blinking

- Request url： nodes/noBlink
- Method： put
- Request Parameters:

Parameter	Description	Type	Comment

- Response data:

Parameter	Description	Type	Comment

5. Upload

5.1. File Upload

- Request url： upload/
- Method： post
- Request Parameters:

Parameter	Description	Type	Comment
File	File of the uploaded firmware	Image	If file is for ESP32, care of its filename

- Response data:

Parameter	Description	Type	Comment

5.2. ESP32 Firmware Upload

- Request url： upload/ESP32
- Method： post
- Request Parameters:

Parameter	Description	Type	Comment
Board	Name of the firmware	String	
Version	Version of the uploaded firmware	String	

- Response data:

Parameter	Description	Type	Comment

5.3. AVR Firmware Upload

- Request url：upload/AVR
- Method：post
- Request Parameters:

Parameter	Description	Type	Comment
fileName	Name of the file	String	
macADR	Mac address of the node	String	Mac address not separated by colon
macAddr	Mac address of the node	String	Mac address separated by colon

- Response data:

Parameter	Description	Type	Comment