**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**

**FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING**

**DEPARTMENT OF CONTROL ENGINEERING & AUTOMATION**

**PHAM MINH THUAN**

**GRADUATION THESIS**

**APPLICATION OF IOT FOR MONITORING AND MANAGING HOUSE**

**ENGINEER OF CONTROL ENGINEERING & AUTOMATION**

**HO CHI MINH CITY, 2020**

**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**

**FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING**

**DEPARTMENT OF CONTROL ENGINEERING & AUTOMATION**

**PHAM MINH THUAN - 1613435**

**GRADUATION THESIS**

**APPLICATION OF IOT FOR MONITORING AND MANAGING HOUSE**

**ENGINEER OF CONTROL ENGINEERING & AUTOMATION**

**SUPERVISOR**

**PhD NGUYEN TRONG TAI**

**HO CHI MINH CITY, 2020**

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

**FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING**

SOCIALIST REPUBLIC OF VIETNAM

Independence – Freedom – Happiness

**DEPARTMENT OF CONTROL ENGINEERING & AUTOMATION**

*HCM city, month…..day…..year….*

**EVALUATION OF SUPERVISOR**

**Name of Thesis:**

**APPLICATION OF IOT FOR MONITORING AND MANAGING HOUSE**

**Student: Supervisor:**

Pham Minh Thuan 1613435 PhD Nguyen Trong Tai

**Evaluation:**

1. Thesis book:

Pages: \_\_\_\_\_\_\_\_\_\_\_\_ Chapters: \_\_\_\_\_\_\_\_\_\_\_\_

Tables: \_\_\_\_\_\_\_\_\_\_\_\_ Picture: \_\_\_\_\_\_\_\_\_\_\_\_

References: \_\_\_\_\_\_\_\_\_\_\_\_ Product(s): \_\_\_\_\_\_\_\_\_\_\_\_

Remarks about the presentation of the thesis book:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Content of thesis:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Potential applications:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Working attitude of student:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Overall evaluation:**

**Score of student:**

Pham Minh Thuan: …………/10

**Supervisor**

(Sign and full name)

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

**FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING**

SOCIALIST REPUBLIC OF VIETNAM

Independence – Freedom – Happiness

**DEPARTMENT OF CONTROL ENGINEERING & AUTOMATION**

*HCM city, month…..day…..year….*

**EVALUATION OF JUDGE**

**Name of Thesis:**

**APPLICATION OF IOT FOR MONITORING AND MANAGING HOUSE**

**Student: Judges:**

Pham Minh Thuan 1613435

**Eluvation:**

1. Thesis book:

Pages: \_\_\_\_\_\_\_\_\_\_\_\_ Chapters: \_\_\_\_\_\_\_\_\_\_\_\_

Tables: \_\_\_\_\_\_\_\_\_\_\_\_ Picture: \_\_\_\_\_\_\_\_\_\_\_\_

References: \_\_\_\_\_\_\_\_\_\_\_\_ Product(s): \_\_\_\_\_\_\_\_\_\_\_\_

Remarks about the presentation of the thesis book:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Content of thesis:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Potential applications:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Working attitude of student:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Overall evaluation:**

**Score of student:**

Pham Minh Thuan: …………/10

**Judge**

(Sign and full name)

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

**FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING**

SOCIALIST REPUBLIC OF VIETNAM

Independence – Freedom – Happiness

**DEPARTMENT OF CONTROL ENGINEERING & AUTOMATION**

*HCM city, month…..day…..year….*

DETAILED OUTLINE

|  |  |
| --- | --- |
| **Name of Thesis: APPLICATION OF IOT FOR MONITORING AND MANAGING HOUSE** | |
| **Supervisor: PhD Nguyen Trong Tai** | |
| **Execution time:** from 29/03/2020 to 10/07/2020 | |
| **Student:**  **Pham Minh Thuan – 1613435** | |
| **Content of the subject:**  **Objective:** Try to research and build a multi-user system which can be used to monitor and manage the house, it includes : 1 IoT Core for controling and processing, 1 web app for communicating with users, 1 platform for ESP32 to communicate with IoT core and some library support that platform. Furthermore, it is scalable in many fields, such as : Agriculture, Industry, Service,…  **Object of study:** Node.js platform and NPM for building a web server, HTTP/HTTPS protocol for comunication, ESP-IDF for programming ESP32 microcontroller, database with MongoDB to store data.  **Execution method:**   * For the Back-End of web server: * Using Express.js to build a web server. * Using JSON Web Tokens for security. * Using Protobuf for making command frame. * Using JSON for making data frame. * Using Mongoose for communicating with MongoDB. * For the Front-End of server: * Using Pug for view engine. * Fixing pattern layout and adding JavaScript file for catching and processing event. * Using examples and libraries in ESP-IDF for making platform.   **Expected results:**   * Building server successfully. * Building web app successfully. * Building complete platform. * Making a house model for testing system. * Adding a web view app on mobile phone with Flutter SDK and Dart language. | |
| **Implementation plan:**   |  |  |  | | --- | --- | --- | | **N** | **Work** | **Period** | | 1 | Researching and studying JavaScript language, Node.js, HTTP/HTTPS and how to make a web server. | 29/03 – 05/04 | | 2 | Researching Google IoT Core and Blynk app | 06/04 – 08/04 | | 3 | Researching API Reference of ESP-IDF | 09/04 – 11/04 | | 4 | Researching Flutter and Dart language | 12/04 – 15/04 | | 5 | Building complete web server with Back-End part | 16/04 – 30/04 | | 6 | Building complete platform for ESP32 | 01/05 – 08/06 | | 7 | Building complete web server with Front-End part | 09/06 – 16/06 | | 8 | Putting system into operation and fix bugs, errors | 17/06 – 20/06 | | 9 | Designing adapter for ESP32-WROVER with KiCad EDA | 21/06 – 22/06 | | 10 | Making house model | 23/06 – 25/06 | | 11 | Buildind web view app | 26/06 – 27/06 | | 12 | Testing system on house model and mobile app | 28/06 – 29/06 | | 13 | Writing thesis | 30/06 – 12/07 | | 14 | Making presentation file | 13/07 – 20/07 | | 15 | The judge review the thesis | 21/07 – 24/07 | | 16 | Presenting thesis to the defense council | 29/07 | | |
| **Certified by Supervisor**  (Sign and full name) | HCM city, month….day …..year…..  **Student**  (Sign and full name) |

LIST OF THESIS DEFENSE COUNCIL

Thesis defense council, established under Decision No. ……………………. on ………………….. of principal of Ho Chi Minh city university of technology.

* 1. …………………………………………. – Chairman.
  2. …………………………………………. – Clerk.
  3. …………………………………………. – Commissioner.
  4. …………………………………………. – Commissioner.

ACKNOWLEDGEMENT

To complete my thesis, I would like to express my respect and gratitude to PhD Nguyen Trong Tai. My supervisor give me an great environment to complete my thesis. The instruction of PhD Nguyen Trong Tai helped me to go right way and have a successful solution.

# Contents

[Contents 4](#_Toc45802064)

[ABSTRACT 1](#_Toc45802065)

[Chapter 1: INTRODUCTION 2](#_Toc45802066)

[1.1. Thesis overview 2](#_Toc45802067)

[1.1.1. Google Cloud IoT Core 2](#_Toc45802068)

[1.1.2. Blynk 3](#_Toc45802069)

[1.1.3. Thesis outline 3](#_Toc45802070)

[1.2. Objective 5](#_Toc45802071)

[1.3. Thesis structure 5](#_Toc45802072)

[Chapter 2: IOT CORE 7](#_Toc45802073)

[Chương 2. 7](#_Toc45802074)

[2.1. Structure of my IoT Core 7](#_Toc45802075)

[2.1.1. MVC 7](#_Toc45802076)

[2.1.2. Inside my IoT Cor **Error! Bookmark not defined.**](#_Toc45802077)

[2.1.2.1. Files in the top of folder 9](#_Toc45802078)

[2.1.2.2. ‘.vscode’ folder 10](#_Toc45802079)

[2.1.2.3. ‘routers’ folder 10](#_Toc45802080)

[2.1.2.4. ‘middlewares’ folder 13](#_Toc45802081)

[2.1.2.5. ‘controller‘ folder 15](#_Toc45802082)

[2.1.2.6. ‘models’ folder 16](#_Toc45802083)

[2.1.2.7. ‘protobuf’ folder 19](#_Toc45802084)

[2.1.2.8. ‘views’ folder 20](#_Toc45802085)

[2.1.2.9. ‘public’ folder 24](#_Toc45802086)

[2.1.2.10. ‘documents’ folder 31](#_Toc45802087)

[2.1.2.11. ‘node\_modules’ folder 32](#_Toc45802088)

[2.2. HTTP/HTTPS Protocol 32](#_Toc45802089)

[2.2.1. HTTP GET 33](#_Toc45802090)

[2.2.2. HTTP POST 33](#_Toc45802091)

[2.3. Data frame 33](#_Toc45802092)

[2.3.1. JSON 33](#_Toc45802093)

[2.3.2. Protobuf 34](#_Toc45802094)

[2.4. Security 34](#_Toc45802095)

[Chapter 3: DATABASE 36](#_Toc45802096)

[Chương 3. 36](#_Toc45802097)

[3.1. Intruduction 36](#_Toc45802098)

[3.1.1. Database 36](#_Toc45802099)

[3.1.2. Databse management system 36](#_Toc45802100)

[3.1.3. SQL 36](#_Toc45802101)

[3.1.4. NoSQL database 37](#_Toc45802102)

[3.2. MongoDB 37](#_Toc45802103)

[3.3. Mongoose 37](#_Toc45802104)

[3.4. MongoDB Atlas 38](#_Toc45802105)

[Chapter 4: DEVICE 40](#_Toc45802106)

[Chương 4. 40](#_Toc45802107)

[4.1. Introduce about ESP32 Microcontroller 40](#_Toc45802108)

[4.1.1. Adapter for ESP32-WROVER-V4 41](#_Toc45802109)

[4.1.2. Schelmatic 42](#_Toc45802110)

[4.1.3. PCB images 45](#_Toc45802111)

[4.1.4. Production 46](#_Toc45802112)

[4.2. ESP-IDF 46](#_Toc45802113)

[4.3. Selt-made components. 46](#_Toc45802114)

[4.4. Algorithm flowchart 47](#_Toc45802115)

[4.5. Config device with mobile app. 48](#_Toc45802116)

[Chapter 5: USER 50](#_Toc45802117)

[Chương 5. 50](#_Toc45802118)

[5.1. Keyword 50](#_Toc45802119)

[5.2. Pages of web app 50](#_Toc45802120)

[5.2.1. Register Page 50](#_Toc45802121)

[5.2.2. Login Page 51](#_Toc45802122)

[5.2.3. Help Page 52](#_Toc45802123)

[5.2.4. Verify Page 52](#_Toc45802124)

[5.2.5. User Page 53](#_Toc45802125)

[5.2.6. Display Page 56](#_Toc45802126)

[5.2.7. Data Table Page 56](#_Toc45802127)

[5.2.8. Update Page 57](#_Toc45802128)

[5.2.9. Delete Page 58](#_Toc45802129)

[5.2.10. Config Page 58](#_Toc45802130)

[5.2.11. Equipment Page 59](#_Toc45802131)

[5.2.12. Equipi Page 59](#_Toc45802132)

[5.2.13. Sensor Page 62](#_Toc45802133)

[5.2.14. Sensi Page 63](#_Toc45802134)

[5.2.15. GUI Page 64](#_Toc45802135)

[5.2.16. Block Page 65](#_Toc45802136)

[5.2.17. Control Block Page 65](#_Toc45802137)

[Chapter 6: TEST MODEL 67](#_Toc45802138)

[Chương 6. 67](#_Toc45802139)

[6.1. House model 67](#_Toc45802140)

[6.2. Mobile app 67](#_Toc45802141)

[Chapter 7: SUMMERY 69](#_Toc45802142)

[Chương 7. 69](#_Toc45802143)

[7.1. Conclusion 69](#_Toc45802144)

[7.2. Future work 69](#_Toc45802145)

[BIBLIOGRAPHY 70](#_Toc45802146)

LIST OF FIGURES

[Figure 1. 1 Diagram of Google Cloud IoT Core 2](#_Toc45721239)

[Figure 1. 2 Structure of Blynk application 3](#_Toc45721240)

[Figure 1. 3 Diagram of system in thesis 4](#_Toc45721241)

[Figure 2. 1 Diagram of IoT Core in thesis 8](#_Toc45721246)

[Figure 2. 2 Directory tree in project folder 9](#_Toc45721247)

[Figure 2. 3 Single files in top of folder 9](#_Toc45721248)

[Figure 2. 4 ‘.vscose’ folder 10](#_Toc45721249)

[Figure 2. 5 ‘routers’ folder 10](#_Toc45721250)

[Figure 2. 6 ‘middlewares’ folder 14](#_Toc45721251)

[Figure 2. 7 ‘controller’ folder 15](#_Toc45721252)

[Figure 2. 8 ‘models’ folder 16](#_Toc45721253)

[Figure 2. 9 ‘protobuf’ folder 19](#_Toc45721254)

[Figure 2. 10 ‘views’ folder 20](#_Toc45721255)

[Figure 2. 11 ‘layouts’ folder 21](#_Toc45721256)

[Figure 2. 12 ‘pages’ folder 22](#_Toc45721257)

[Figure 2. 13 ‘images’ folder 25](#_Toc45721258)

[Figure 2. 14 ‘stylesheets’ folder 25](#_Toc45721259)

[Figure 2. 15 ‘javascripts’ folder 26](#_Toc45721260)

[Figure 2. 16 ‘documents’ folder 31](#_Toc45721261)

[Figure 2. 17 ‘node\_modules’ folder 32](#_Toc45721262)

[Figure 4. 1 Function block diagram of ESP32 40](#_Toc45791699)

[Figure 4. 2 ESP32-WROVER-V4 module 41](#_Toc45791700)

[Figure 4. 3 ESP32 Pico D4 kit 41](#_Toc45791701)

[Figure 4. 4 ADC Input 42](#_Toc45791702)

[Figure 4. 5 Buck Converter 5v to 3v3 with IC AMS1117 42](#_Toc45791703)

[Figure 4.6 Boot button and reset button 42](#_Toc45791704)

[Figure 4.7 DAC 43](file:///C:\Users\Win%2010\Desktop\thesis\thesis.docx#_Toc45791705)

[Figure 4.8 GPIO 43](#_Toc45791706)

[Figure 4.9 CAN 43](#_Toc45791707)

[Figure 4.10 I2C 44](#_Toc45791708)

[Figure 4.11 SPI 44](#_Toc45791709)

[Figure 4.12 output of 3.3V and 5V 44](#_Toc45791710)

[Figure 4. 13 USB flash through uart 45](#_Toc45791711)

[Figure 4. 14 PCB-top Figure 4. 15 PCB-bottom 45](#_Toc45791712)

[Figure 4. 16 Bottom of adapter Figure 4. 17 Top of adapter 46](#_Toc45791713)

[Figure 4. 18 Self-made components 46](#_Toc45791714)

[Figure 4. 19 Algorithm flowchart of devices 48](#_Toc45791715)

[Figure 4. 20 GUI of IoT Smartconfig app 49](#_Toc45791716)

[Figure 5. 1 Register card 51](#_Toc45720567)

[Figure 5. 2 Login card 51](#_Toc45720568)

[Figure 5. 3 Verify email card 52](#_Toc45720569)

[Figure 5. 4 Verify number card 52](#_Toc45720570)

[Figure 5. 5 User Page 53](#_Toc45720571)

[Figure 5. 6 Structure of Setting 54](#_Toc45720572)

[Figure 5. 7 Cards of User Page 54](#_Toc45720573)

[Figure 5. 8 Bar chart 55](#_Toc45720574)

[Figure 5. 9 Line chart 55](#_Toc45720575)

[Figure 5. 10 Data table 55](#_Toc45720576)

[Figure 5. 11 Feature of Display Page 56](#_Toc45720577)

[Figure 5. 12 Features of Data Table Page 56](#_Toc45720578)

[Figure 5. 13 Update Password card 57](#_Toc45720579)

[Figure 5. 14 Update ID card 57](#_Toc45720580)

[Figure 5. 15 Delete Account card 58](#_Toc45720581)

[Figure 5. 16 Config card 59](#_Toc45720582)

[Figure 5. 17 Equipments record 59](#_Toc45720583)

[Figure 5. 18 Turning on/off equipments card 60](#_Toc45720584)

[Figure 5. 19 Choosing type of Equipments card 60](#_Toc45720585)

[Figure 5. 20 Name of equipments card 61](#_Toc45720586)

[Figure 5. 21 Index of ports card 61](#_Toc45720587)

[Figure 5. 22 Name od ports card 62](#_Toc45720588)

[Figure 5. 23 Pins card 62](#_Toc45720589)

[Figure 5. 24 Sensors record 63](#_Toc45720590)

[Figure 5. 25 Turning on/off sensors card 63](#_Toc45720591)

[Figure 5. 26 Choosing type of sensors card 64](#_Toc45720592)

[Figure 5. 27 Name of sensors card 64](#_Toc45720593)

[Figure 5. 28 Config block card 65](#_Toc45720594)

[Figure 5. 29 Blocks record 65](#_Toc45720595)

[Figure 5. 30 Control Block Page 66](#_Toc45720596)

[Figure 6. 1 House model 67](#_Toc45720597)

[Figure 6. 2 Home Page and User Page of mobile app 68](#_Toc45720598)

LIST OF KEYWORDS

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Full** | **Meaning** |
| IoT |  |  |
| ID |  |  |
| GUI |  |  |
| API |  |  |
| MVC |  |  |
| HTTP |  |  |
| HTTPS |  |  |
| ESP-IDF |  |  |
| HTML |  |  |
| CSS |  |  |
| URL |  |  |
| JSON |  |  |
| TLS |  |  |
| SSL |  |  |
| JWT |  |  |
| DBMS |  |  |
| SQL |  |  |
| ODM |  |  |
| AWS |  |  |
| GCP |  |  |
| CPU |  |  |
| TSMC |  |  |
| ULP |  |  |
| DMIPS |  |  |
| PSRAM |  |  |
| SiP |  |  |
| PCB |  |  |
| ADC |  |  |
| DAC |  |  |
| CAN |  |  |
| I2C |  |  |
| GPIO |  |  |
| SPI |  |  |
| USB |  |  |
| NVS |  |  |
| FreeRTOS |  |  |
| Wi-Fi |  |  |

ABSTRACT

IoT is the field, which is developing very fast. We have many project, which are applications of IoT such as : Smart-home, green-agriculture, IoT in Industrial,… But a lot of projects just are built for 1 invidual, it mean that it just operates in its local area. So when we want to connect with other projects, we need to write the driver to interface, or even we need to build application again.

In the world, we have many applications , cloud support IoT, which can help us resolve this problem, such as : Blynk, Google Cloud IoT core, Azure, AWS IoT,… they are extremely great, but we need to pay money for using.

In this thesis, I endeavor to create a system, which include: 1 IoT-core (web server, database) , 1 template for microcontroller ESP32 to communicate with IoT-core, 1 web app to users can manage your house. My system support muilti-user, it mean that people can create their account and start to use it now. With HTTP protocol, not only ESP32 series but also smart-phone, PC, orther microcontroller/microprocessor which can become devices. With using free platforms, free languages, free framworks, I will give people a free application for using, developing. It allows developers to expense features, they needn’t build again and APIs from server may be useful for building other app.

# Chapter 1: INTRODUCTION

## Thesis overview

IoT or Internet of things describes a system, which include many interrelated computing devices, every “thing” has an ID and it can share data together or communicate with human. IoT is not limited by network or protocol.

Therefore, I want to build a system, which allows users to choose option in their devices and then config it on web app to use, so I decided to research some pattern systems.

### Google Cloud IoT Core

The figure below is structure of Google Cloud IoT Core, which give me an ideal:

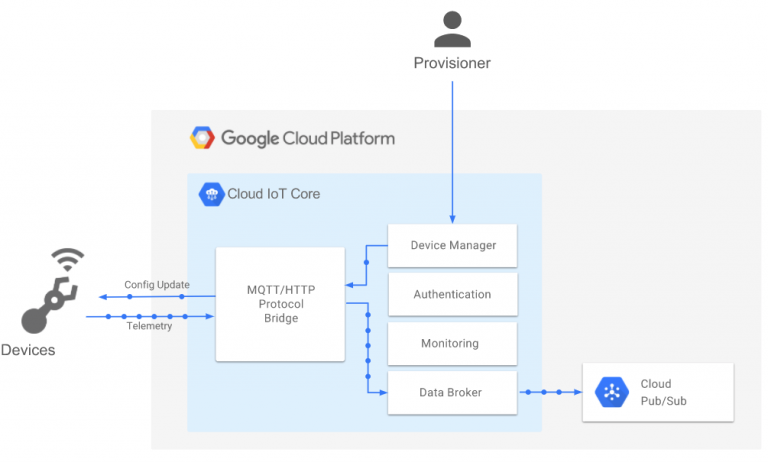


Figure 1. 1 Diagram of Google Cloud IoT Core

Cloud IoT Core is a fully managed service that allows you to easily and securely connect, manage, and ingest data from millions of globally dispersed devices. Cloud IoT Core, in combination with other services on Cloud IoT platform, provides a complete solution for collecting, processing, analyzing, and visualizing IoT data in real time to support improved operational efficiency[1].

### Blynk

Blynk was designed for the Internet of Things. It can control hardware remotely, it can display sensor data, it can store data, vizualize it and do many other cool things[2].

There are three major components in the platform:

* Blynk App: allows to you create amazing interfaces for your projects using various widgets we provide.
* Blynk Server: responsible for all the communications between the smartphone and hardware. You can use our Blynk Cloud or run your [private Blynk server](https://docs.blynk.cc/#blynk-server) locally. It’s open-source, could easily handle thousands of devices and can even be launched on a Raspberry Pi.
* Blynk Libraries : for all the popular hardware platforms - enable communication with the server and process all the incoming and outcoming commands.

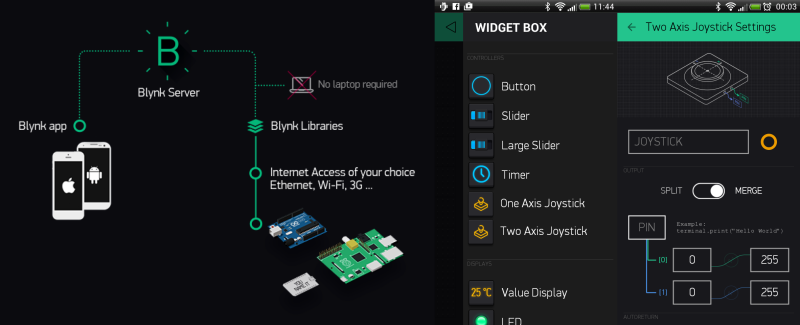


Figure 1. 2 Structure of Blynk application

### Thesis outline

I feel Cloud IoT Core of Google is so cool, in addition, I am expressed about GUI of mobile app “Blynk”. All of them make me really want to create an Application, which can support GUI like “Blynk” but it is on the web browser and my application will support multi-user, multi-device.

Finally, I decide make lightweight, powerful system, with APIs and view engine, it will look like this:

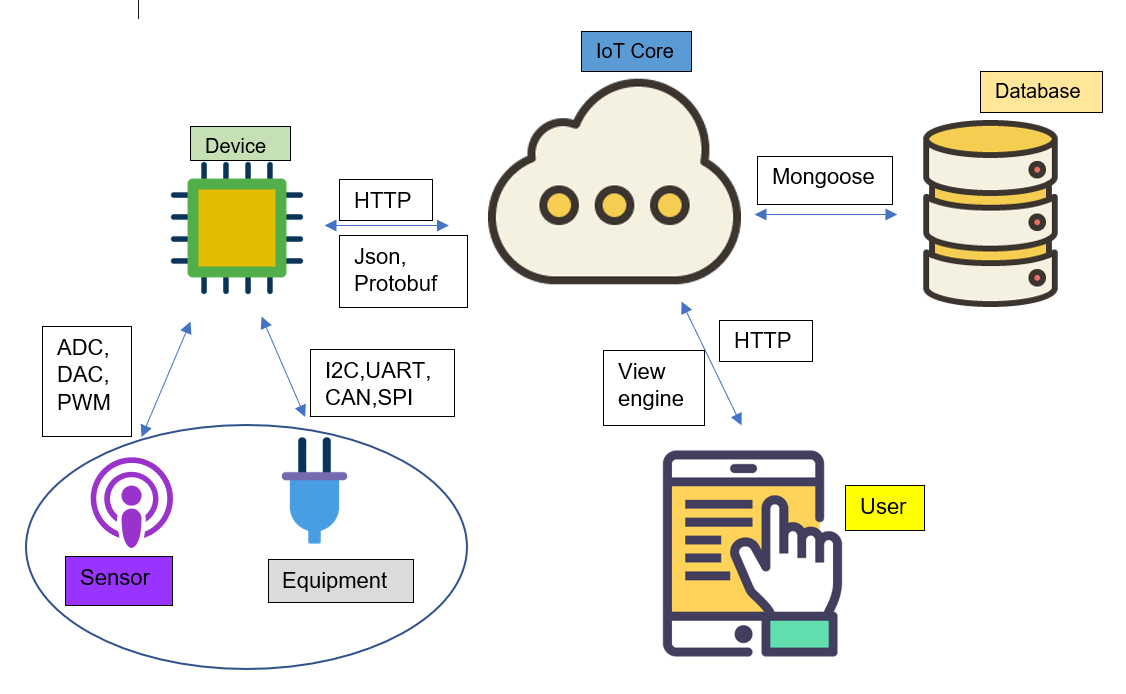


Figure 1. 3 Diagram of system in thesis

* **IoT Core**

This element is headquater of system, it looks like a “Big bridge”, help other elements to operate synchronously, smoothly. IoT core of my system is a web server, it is based on MVC.

* **User**

This element represents devices like smart-phone, PC,… or anything supported HTTP/HTTPS.

* **Device**

This element includes microprocessors or microcontrollers, which is supported HTTP/HTTPS. Developers can program them for their purposes and users just use it.

* **Database**

This element is used to store data, manage them flexibly. All data will be saved here.

* **Sensor**

This element represent sensors, devices, which collect data from environment, help the users in monitoring.

* **Equipment**

This element represent for machines, devices which the users can control and manage.

## Objective

The objective of the thesis is to build a system which can be used to monitor and manage the house, it is able to extend for other projects. In order to gain that, the thesis needs to follow tasks:

* Building a web server with MVC.
* Building a GUI for communicating easily.
* Building a template for devices to interface with web server.
* Building drivers which support for communicating between devices and sensors, equipments.

## Thesis structure

Belong to tasks of thesis and elements of system, I divide thesis into 7 chapters :

* **Chapter 1:** Introdution: I will present overview about IoT, reasons make me want to make a this system and diagram of system.
* **Chapter 2:** IoT Core: I will go to details about my IoT core, reasons I choose MVC structure, how to build it and main features.
* **Chapter 3:** Database: I will introduce about database, MongoDB generally, Mongoose.
* **Chapter 4:** Device: I will present ESP32, ESP-IDF, adapter for ESP32-WROVER designed by me and how to config a device.
* **Chapter 5:** User: I will present how to use web app.
* **Chapter 6:** Test model: I will introduce a house model I have made to test my system, with an web view app built by Flutter.
* **Chapter 7:** Summary: I will present results, limitations and development oritenations in the future of thesis.

# Chapter 2: IOT CORE



## Structure of my IoT Core

My IoT Core is a web server, it is build from: JavaScript, HTML, CSS, Pug. JavaScript is a language for client, it mean that it just runs on browser such as: Firefox, CocCoc, Chomium, Edge,…. But with the Node.js platform, which has powerful engine V8, I can build my web server on Visual Studio Code and then deploy it on Heroku. With support from NPM , an open source, I do that in the great way.

Javascript does not have top function, we will not have function like “int main()”. In theory , I can add anything in 1 file, but it will look terrible, and that is hard to fix. I need a standard struct to organize those files in my project folder.

### MVC

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects[3].

* **Model**

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data[3].

* **View**

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with[3].

* **Controller**

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data[3].

### IoT Core of thesis

I had spent a long time to understand MVC structure, and inherited their ideal. After all those, I present structure of my IoT Core as below:

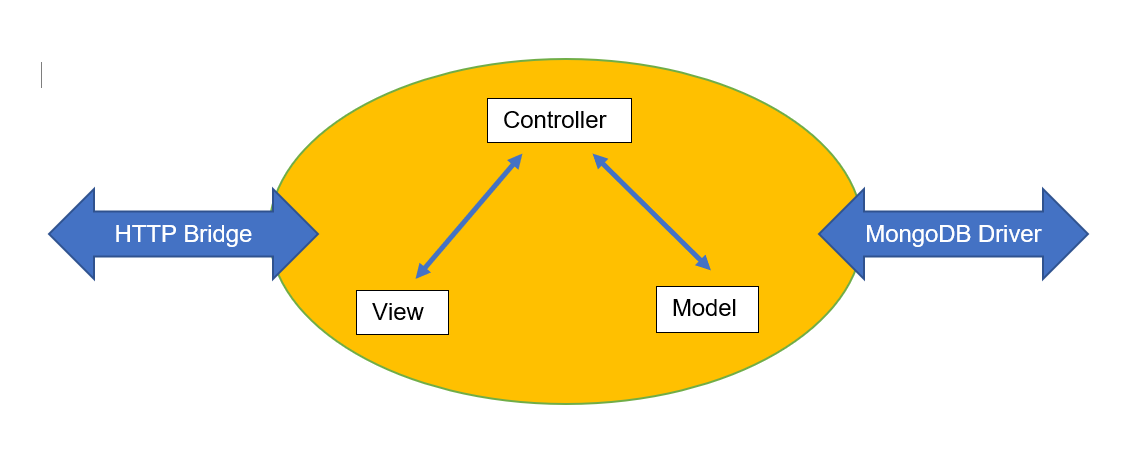


Figure 2. 1 Diagram of IoT Core in thesis

With that structure, I start to actualize it. First of all, I organize a directory tree like below:

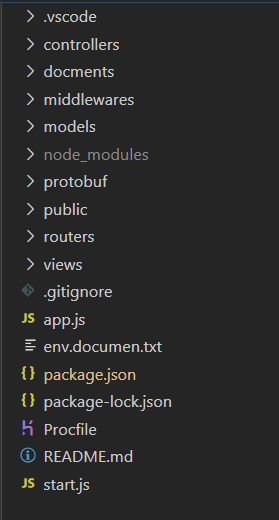


Figure 2. 2 Directory tree in project folder

I will explain these folders and files one by one to make it easy to understand.

#### Files in the top level of folder

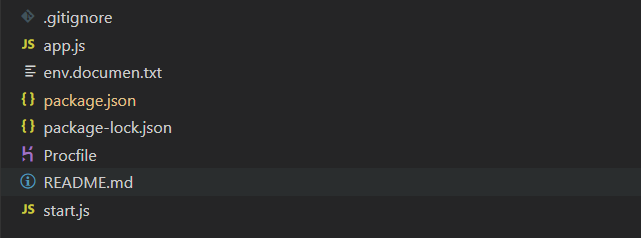


Figure 2. 3 Files in top level of folder

* + app.js: It is the most important file, it will require packages, config parameters or methods for server, setup database , declare target for URL, setup view engine and open port to listen.
  + package.json: It is is default file, which contains some parameters, which are packages used in project, name of main file, scripts to run.
  + package-lock.json: It is default file, too, which help to reduce errors when there are 2 versions of 1 package.
* .gitignore: Because I use github, I will add some files or folders which are not updated when I fix my server.
* Procfile: To declare the script to run server on Heroku resource.
* start.js: This file manages the dashboard pattern.

#### ‘.vscode’ folder



Figure 2. 4 ‘.vscose’ folder

This folder is generated when I use Visual Studio Code, it contains some default settings of developer.

#### ‘routers’ folder

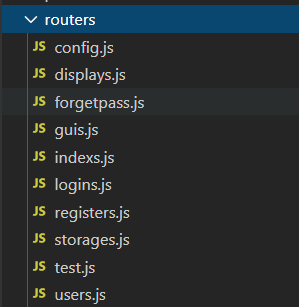
****

Figure 2. 5 ‘routers’ folder

Every path of server is a target of main URL: “<https://iot-server-365.herokuapp.com/>”. Every file represents 1 router or more.

* indexs.js: This manages router ‘/’, it is in roof of path. Because I think go to Login Page when users access web app is not interesting.
* logins.js: This file will check information of users, if they are valid, users will receive a cookie to access their resources. It manages ‘/login’ router.
* registers.js: This file will check information of registers, if they are new users, server will create a new instance and store in database. It manages ‘/register’ router.
* storages.js: this file manages ‘/storage’ router, it is a gateway to communicate with devices, where data will be post and command will be get.
* test.js: this file is ‘/test’ router , it is helpful for creator to test new features.
* users.js: this file manages 5 routers :
* ‘/user’ : this router manages main page, this page is a dashboard, it gives user many options and shows charts, data table.
* ‘/user/update’ : this router manages Update Page, users can change ID of devices, or change password.
* ‘/user/clone’ : this router manages clone page, this page allows users to download data from sensors with Excel, PDF, CSV format, or just print the table if users are connecting printer.
* ‘/user/logout’ : this router resets cookie of users and redirects them to Login Page again, users just press red button on top of User Page
* ‘/user/delete’ : this router will delete account of users, they will become register again, users should consider carefully.
* config.js : this file manages 7 routers:
* ‘/user/config’ : this router manages Config Page, which allows users to add , remove devices.
* ‘/user/config/sensors’ and ‘/user/config/equipments’: these routers manage Sensor Page and Equipment Page.
* ‘/user/config/sensors/search’,‘/user/config/equipments/search’: these routers will receive name of device you fine and go to its page.
* ‘/user/config/sensors/:id’ , ‘/user/config/equipments/:id’ : these routers manage pages which contains parameters of device.
* forgetpass.js: this file is a method to help users when they forget password and can not login, it includes 2 routers :
* ‘/forgetpass’: users will send name of their email account to this router, and it will send a authentication message to their email.
* ‘/forgetpass/verify’: this router will check special number in authentication message and check name of email account again, if the information is correct, router will send new password to their email, users can login and change other password or just use that password.
* guis.js: this file manages routers which support and manage blocks. Block is a cluster which inlucdes 1 or many equipments, one block can contain equipments from other devices. File “guis.js” includes 4 routers:
* ‘/user/gui’: this router manages Gui Page, this page allows users to add or remove their blocks, block corresponds port in device, so it has to assign with 1 device.
* ‘/user/gui/blocks’: this router manages Block Page, it show a record of blocks which you added, it support 1 search bar to fine block with its name. It receives APIs which are commands from users and save to database.
* ‘/user/gui/blocks/search’: this router will receive name of block you have required and then redirect to page of that block.
* ‘user/gui/block/:id’: this router manages pages which match its id, it will render 1 page to users can control easily.
* displays.js : this file manages routers which support chart. It includes 3 routers :
* ‘/user/display’: this router will render Display Page with 2 types of charts: line chart and bar chart.
* ‘/user/display/getdata’: this router will support data for line chart and bar chart.
* ‘/user/display/datatable’: this router will support for data table.

#### ‘middlewares’ folder

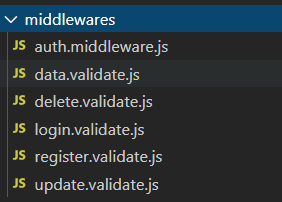
****

Figure 2. 6 ‘middlewares’ folder

Some router have to do many operations, every action in web server has to check information of users firstly, then server will decide to do it or no.

It contains 6 files for 6 validation purposes :

* auth.middleware.js: this file has one middleware function, its name is ‘requireAuth”, it will check cookies of users, the cookies is just received when users sign in, so this module looks like a filter to prevent strangers and protect data of users.
* data.validate.js: this file has 1 middleware function, its name is “checkID”, only devices which have ID and correct format are able to be accepted.
* delete.validate.js: this file has 2 middlewares:
* “checkFilled” : to ensure that users fill all necessary information.
* “checkAccount” : Deleting account means users will lost all data they have, I want to check information carefully, in case, the strangers can access to account of users, they can not delete it because stranger don’t know password.
* login.validate.js: this file is the same as ‘delete.validate.js’ file, but it render Login Page instead of Delete Page.
* register.validate.js: this file is the same as ‘login.validate.js’ file, too. But, it add a function which is used for sending message to users’s email.
* update.validate.js: this file is the same as ‘login.validate.js’ file, too.

#### ‘controller‘ folder

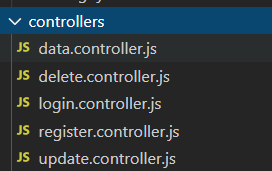
****

Figure 2. 7 ‘controller’ folder

Controller files include final actions of functions callback in processing request, dividing callback into stages will help to fix easily, increase perform, there are 5 controller files:

* data.controller.js: it just sends one response to devides which posted data successfully.
* delete.controller.js: it will render Delete Page if cookies is correct, or redirect users to Login Page if deleting is successful.
* login.controller.js: it will render Login Page if users access Login Page, if uses sign in successful, it will make a history login into database, and redirect users to User Page.
* register.controller.js: it will redirect register to Login Page to access if regis successfully.
* update.controller.js: it will update new data of users on database, if users update ID , it will send email with new ID .

#### ‘models’ folder

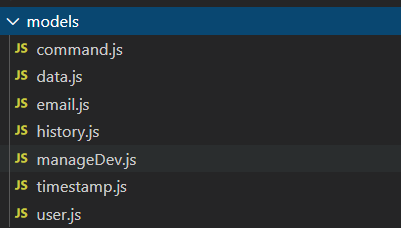


Figure 2. 8 ‘models’ folder

Structures and functions in these files just are called and used in other file.

* command.js: this is model of command to store in database, commands are used to control device :

-------------------------------------------------------------------

var mongoose = require('mongoose');

var commandSchema = new mongoose.Schema({

ID: String,

timestamp: String,

device: Number,

io: Number,

value: Number

},{ versionKey: false});

var Command = mongoose.model('Command', commandSchema, 'command');

-------------------------------------------------------------------

* data.js: this is model of data to save in database.

-------------------------------------------------------------------

var mongoose = require('mongoose');

var dataSchema = new mongoose.Schema({

ID: String, /\*\* unique Account \*/

device: Number, /\*\* unique device, odd for post/display, even for get/gui \*/

datetime : String, /\*\* for display/save realtime \*/

timestamp: Number, /\*\* for caulator realtime \*/

form: [Object] /\*\* data from device \*/

},{ versionKey: false});

var Data = mongoose.model('Data', dataSchema, 'data');

-------------------------------------------------------------------

* email.js: this is model to pack messages email. It can be reuse in other features.

-------------------------------------------------------------------

module.exports.mailServer = nodemailer.createTransport({

service: 'gmail', //https://medium.com/@.jay/sending-email-using-express-js-with-nodemailer-in-heroku-71741f29463c

host: 'smtp.gmail.com',

port: 465,

secure: true,

auth: {

user: process.env.EMAIL,

pass: process.env.PASS\_EMAIL

}

});

module.exports.form = {

from: "Server IoT core Te",

to: " ",

subject: " ",

text: " "

};

-------------------------------------------------------------------

* history.js: this model store state of users, login or logout.

-------------------------------------------------------------------

var mongoose = require('mongoose');

var historySchema = new mongoose.Schema({

timestamp: String,

email: String,

act : Number

},{ versionKey: false});

var History = mongoose.model('History', historySchema, 'history');

-------------------------------------------------------------------

* manageDev.js: this model is format to save state of equipments, data from sensors, save parameters of devices.

-------------------------------------------------------------------

var mongoose = require('mongoose');

var manageDevSchema = new mongoose.Schema({

ID: String, /\*\* unique Account \*/

timestamp: Number,

dev : Number,

mask: String,

type : Number,

child : [Object]

},{ versionKey: false});

var ManageDev= mongoose.model('ManageDev', manageDevSchema, 'manageDev');

-------------------------------------------------------------------

* timestamp.js: this is model to log timestamp with information, it is used to determine time when debug.

-------------------------------------------------------------------

console.log('Timestamp: ', Date.now());

next();

-------------------------------------------------------------------

* user.js: this model is used to save information of users, ‘status’ announces some state of users, it can be expensed and be useful in the future.

-------------------------------------------------------------------

var mongoose = require('mongoose');

var userSchema = new mongoose.Schema({

timestamp: Number,

email: String,

password: String,

status: Number

});

var User = mongoose.model('User', userSchema, 'user');

-------------------------------------------------------------------

#### ‘protobuf’ folder

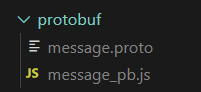
****

Figure 2. 9 ‘protobuf’ folder

* message.proto: this is frame of command.

-------------------------------------------------------------------

message Sensor {

string ID=1;

uint32 device=2;

uint32 io =3;

float value=4;

}

message Sensors {

repeated Sensor sensors=1;

}

-------------------------------------------------------------------

‘message.proto’ need to be generated before using.

* Syntax to generate proto file to JavaScript file:

-------------------------------------------------------------------

protoc --js\_out=import\_style=commonjs,binary:. <file>.proto

-------------------------------------------------------------------

After generating this file, core runtime need to be used, which should be in a file called “google-protobuf.js”, it can be installed through npm “google-protobuf”. Finally, we can use output file nomally.

* Syntax to generate proto file to file.c and file.h in C:

-------------------------------------------------------------------

protoc-c --c\_cout=. <file>.proto

*-------------------------------------------------------------------*

After generating this file, we will have 1 components with source file and header file.

#### ‘views’ folder

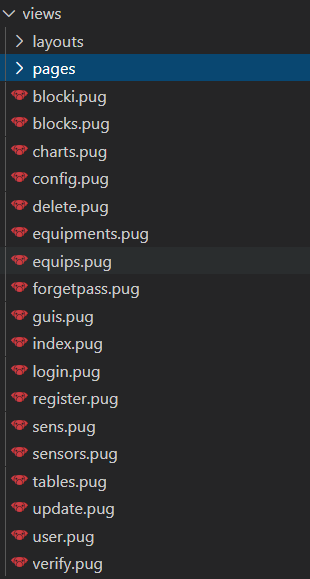


Figure 2. 10 ‘views’ folder

IoT Core use a view engine named “Pug”. Pug is an HTML preprocessor implemented in Node.js and it help to render page from Pug to HTML. It supports methods, packages to make the pages of users look better. Web server use 1 platform pattern, which design on github of David Miller[4]. It includes : layouts folder and pages folder. Pages in web app are based on layout of those resources.

* layouts:

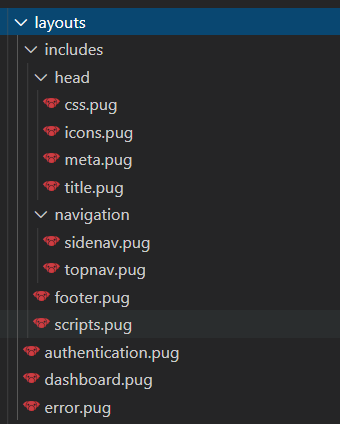


Figure 2. 11 ‘layouts’ folder

There are features standards and some forms which are unchanged, all of those will be render into layouts. It is taken and changed to be useful with thesis.

* css.pug: it includes CSS files, they are pattern which are designed to make pages look so great.
* icons.pug: this file adds icon for web app.
* meta.pug: it includes meta tags, kind of tag defines metadata about an HTML document.
* title.pug: this file helps every page to be assigned title when server render.
* sidenav.pug: it supports format on the left of User Page.
* topnav.pug: it supports format on the top of User Page.
* footer.pug: it supports format on the bottom of pages.
* scripts.pug: it includes script tag to add pattern JavaScript files.
* authentication.pug: it is a pattern layout, which is used to make Login Page or Register Page.
* dashboard.pug: it is pattern dashboard, I fixed it to render for User Page.
* error.pug: it is pattern layout for error pages, when users access wrong URL, this layout will be rendered.
* pages:

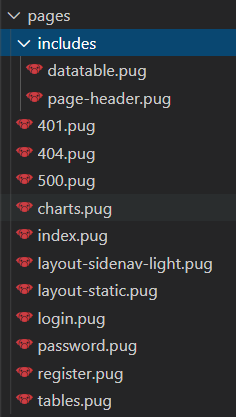


Figure 2. 12 ‘pages’ folder

These files are pattern pages, I build my pages with those format with fixing and adding JavaScript file to catch event.

* datatable.pug: this file is form of one table in HTML, data table is support by jQuery library.
* page-header.pug: this file supports format on the top of page.
* 401.pug : it is a pattern layout, rendered with 401 error, 401 error occurs when request of client has not been applied.
* 404.pug : it is a pattern layout, rendered with 404 error, 404 error occurs when pages are not found.
* 500.pug : it is a pattern layout, rendered with 500 error, 500 error happen when something has gone wrong on server.

Next, I will present my pages I made:

* index.pug : my home is rendered from this file, it is very simple, and it is made by me, not using pattern layouts.
* login.pug : this file will be rendered to Login Page, anyone wants to go to path ‘/user’ , they have to pass this page.
* register.pug : this file is rendered to Register Page, if users do not have an account, they need to go to this page.
* forgetpass.pug : I make file to render Help Page, users can go to this page to reset password.
* verify.pug : this file is rendered to Verify Page, this page help users verify email to reset.
* user.pug : this file is based on pattern dashboard, but it is added some features to be usefull in thesis.
* update.pug : this file is rendered to Update Page, users can update new password or receive new ID.
* delete.pug : this file is rendered to Delete Page, users can delete their account in this page.
* tables.pug : this file base on tables pattern, it helps to display data with tables.
* charts.pug : this file is based on chart layout patterns, I used chart.js package for it.
* config.pug : this file is based on login layout patterns, just adding more input tags and more buttons tag.
* sensors.pug : this file is rendered to Sensor Page, it is based on login layout pattern.
* sens.pug : this file is rendered to Sensi Page, it is base on login layout pattern.
* equipments.pug : this file is rendered to Equipment Page, it is base on login layout pattern..
* equips.pug : this file is rendered to Equipi Page, it is base on login layout pattern..
* guis.pug : this file is rendered to Gui Page, it is base on pattern login layout, it is quite the same of “config.pug”.
* blocks.pug : this file is rendered to Block Page, it is base on login layout pattern..
* blocki.pug : this file is rendered to Control Block Page, it is base on login layout pattern.

#### ‘public’ folder

It includes three folders: images, javascripts, stylesheets. In order to have a good page, I need CSS files , it will increase UX . These folders represent three parts of one page: content, format and event.

* ‘images’

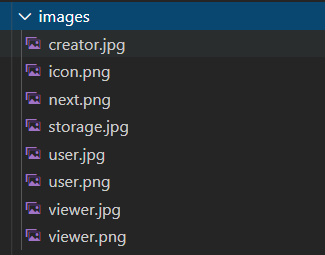


Figure 2. 13 ‘images’ folder

I put my images in here, “user.jpg” is used for the Home Page, “icon.png” is icon on all pages of my web app. Other pictrues is old, it may be reused in the future.

* ‘stylesheets’

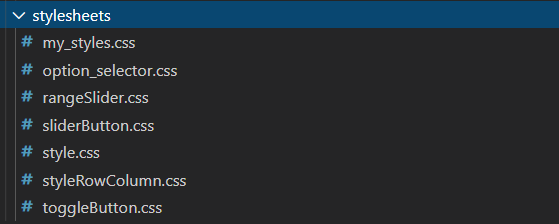


Figure 2. 14 ‘stylesheets’ folder

In the first time, I just want to make a server API , but then , I think the GUI shoud be created to users can use easily. The pages need a good format, it makes users expressed.

CSS files help HTML files look so great, nobody wants a page with only basic tag. I have collected many CSS files on the internet and put it here, but there are 3 important files :

* my\_styles.css: this file includes almost format of pattern layouts, it have many good properties.
* style.css: this file is made by me, some properties I made and some properties I have coppied from other pages like: FaceBook, Google,Lazada.
* rangeSlither.css: this file support properties to make a slider, which is used on control DAC and PWM.
* ‘javascripts’

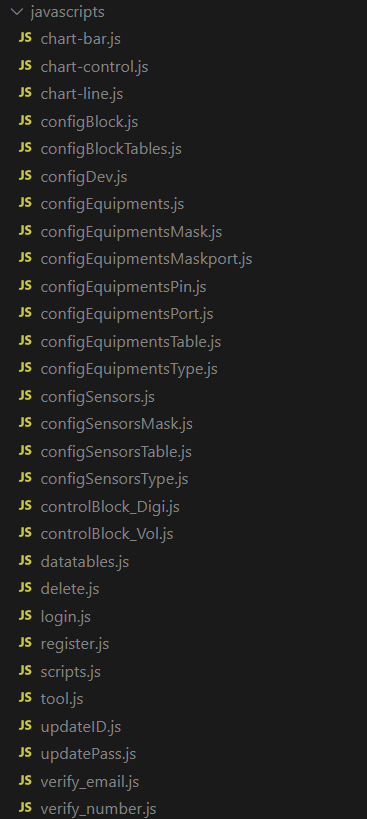


Figure 2. 15 ‘javascripts’ folder

Every page needs one or more JavaScript files to catch events and process them. This is JavaScript files for clients.

* tool.js: this file will get time of client then process and display on the top of Home Page, it looks like a clock.
* scripts.js: this is file of pattern of layout, it manages state of side navigator in dashboard
* login.js : it will wait until users press ‘login’ button, then it will read email and password of users, make a JSON string and send to IoT Core, my server will check and send a response, if information is correct, users will be directory to Users Page, or not, users will stand at Login Page with an error announcement.
* register.js: I require 3 inputs: email, password, confirm password, it will send API of that parameters to server to process, if information is correct, secer will return 1 response to redirect client to Login Page, or not, users will stand at Register Page with an error announcement.
* updateID.js: it will read information with 2 parameters: email and password, but it will send this pakage to router ‘/user/update’.
* updatePass.js: it reads 3 input parameters: email, password and newpassword. The package will sent to ‘/user/update’.
* delete.js: it will read information with 2 parameters: email, password, but it will send package to router ‘/user/delete’.
* verify\_email.js: it will send email’name of users who forget their password to router ‘/forgetpass’..
* verify\_number.js: it will send email of users and 1 number they received in message box of their email to ‘/forgetpass/verify’.
* datatables.js: it will send request to get data for table feature, pages add this file can get data continuously.
* chart-line.js: it includes function to initialize line-chart, function to get data continuously and function to clear data when full.
* chart-bar.js: it includes function to initialize bar-chart, function to get data continuously.
* chart-control.js: when users press “Choose” buton, this file will catch event “click” and set variable “start” to 1, if this variable is 1, program will start to initialize and get data.
* configDev.js: after users fill all parameters for dev and press “add”/”remove” button, it will pack data and send to server, and wait response.
* configEquipments.js: it will read state of switch button on card which allow users to disable or enable equipments, then send to server.
* configEquipmentsMark.js: it will read names of equipments which users config, then send to server.
* configEquipmentsMaskport.js: it reads name of port, which users config, one port represent one “block”, then sends to server.
* configEquipmentsPin.js: it reads index of pin on device, which users config, then sends to server.
* configEquipmentsPort.js: it reads index number of port which users config, every port has 1 unique number, then sends to server.
* configEquipmentsType.js: it will read state of switch button on card which allow users to choose type of sensor, then send to server.
* configEquipmentsTable.js: it reads name of devices, which control equipments, in the search box of equipment page, then packs and sends this information to server for checking.
* configSensors.js: it will read state of switch button on card which allow users to disable or enable sensors, then send to server.
* configSensorsMask.js: it will read names of sensors which users config, then send to server.
* configSensorsType.js: it will read state of switch button on card, which allow users to choose type of sensor, then send to server.
* configSensorsTable.js: it reads name of devices, which control sensors, in the search box of sensor page, then packs and sends this information to server for checking.
* configBlock.js: after users fill all parameters for block and press “add”/”remove” button, it will pack data and send to server, and wait response .
* configBlock\_Digi.js: it will read state of switch button on card which allow users to turn on or turn off equipments in block, then send to server.
* configBlock\_Vol.js: it will read value of slider on card which allow users to adjust value of DAC or PWM, then send to server.
* configBlockTables.js: it reads name of block in the search box of Block Page, then packs and sends this information to server to check.

Example for sending request with POST method to login router:

----------------------------------------------------------------

let response = await fetch('https://iot-server-365.herokuapp.com/login',{

method: 'post',

mode: 'cors',

headers:{

//'Accept': 'application/json, text/plain, \*/\*',

'Content-Type': 'application/json'

},

body:JSON.stringify(frame)

});

----------------------------------------------------------------

Example for sending request with GET method to getdata router:

----------------------------------------------------------------

let response = await fetch('https://iot-server-365.herokuapp.com/user/display/getdata', {

method: 'get',

mode: 'cors',

headers: {

'Accept': 'application/json, text/plain, \*/\*',

'Content-Type': 'application/json',

id:dev

}

});

----------------------------------------------------------------

#### ‘documents’ folder

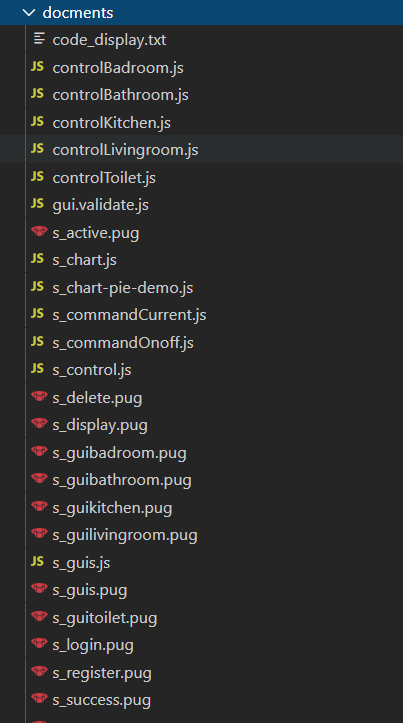
****

Figure 2. 16 ‘documents’ folder

It includes old files, when I build the system, some errors occured and some GUI were obsolete . But it is the premises for me to build a good system now. Some method I can reuse or improve in the future.

#### ‘node\_modules’ folder

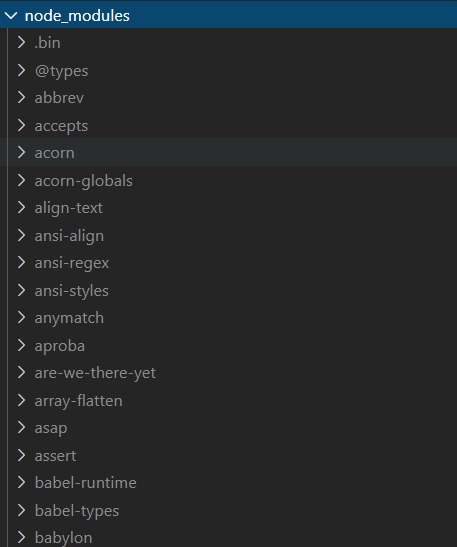


Figure 2. 17 ‘node\_modules’ folder

These are the modules of Node.js, they are based on JavaScript, TypeScript and other languages. It includes packages I added in the ‘package.json’ file, too.

## HTTP/HTTPS Protocol

The Hypertext Transfer Protocol (HTTP) is an [application protocol](https://en.wikipedia.org/wiki/Application_protocol) for distributed, collaborative, [hypermedia](https://en.wikipedia.org/wiki/Hypermedia) information systems. HTTP is the foundation of data communication for the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web) (www), where [hypertext](https://en.wikipedia.org/wiki/Hypertext) documents include [hyperlinks](https://en.wikipedia.org/wiki/Hyperlink) to other resources that the user can easily access, for example by a [mouse](https://en.wikipedia.org/wiki/Computer_mouse) click or by tapping the screen in a web browser[5].

Hypertext Transfer Protocol Secure (HTTPS) is an extension of the [Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (HTTP). It is used for [secure communication](https://en.wikipedia.org/wiki/Secure_communications) over a [computer network](https://en.wikipedia.org/wiki/Network_operating_system), and is widely used on the Internet. In HTTPS, the [communication protocol](https://en.wikipedia.org/wiki/Communication_protocol) is encrypted using [Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security) (TLS) or, formerly, Secure Sockets Layer (SSL). The protocol is therefore also referred to as HTTP over TLS, or HTTP over SSL[6].

My server build on HTTP, but when deploy to Heroku resource, it has become HTTPS. To avoid errors, it has to add a meta tag, which help to upgrade from URL with HTTP to URL with HTTPS.

-------------------------------------------------------------------------

meta(http-equiv="Content-Security-Policy" content="upgrade-insecure-requests")

-------------------------------------------------------------------------

In my thesis, I used 2 method of HTTP, that is : GET and POST.

### HTTP GET

Use GET requests to retrieve resource representation/information only – and not to modify it in any way. As GET requests do not change the state of the resource, these are said to be safe methods. Additionally, GET APIs should be idempotent, which means that making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server[7].

### HTTP POST

Use POST APIs to create new subordinate resources, for example, a file is subordinate to a directory containing it or a row is subordinate to a database table. Talking strictly in terms of REST, POST methods are used to create a new resource into the collection of resources[7].

## Data frame

### JSON

[JSON](https://en.wikipedia.org/wiki/JSON), or JavaScript Object Notation, is a minimal, readable format also used to structure data. It is also an alternative to XML and is mainly used as a way of transmitting data between a server and web application. JSON uses human-readable way as a way of transmitting data objects made up of attribute-value pairs and array data types (or another type of serializable value). JSON was derived from JavaScript[8].

Because my system uses almost JavaScript, it has function to parse JSON and ESP-IDF supports library to create JSON, all of them are great conditions to use JSON for sending data to server.

### Protobuf

[Protocol buffers](https://en.wikipedia.org/wiki/Protocol_Buffers), also known as Protobuf, is a protocol that Google developed internally to enable serialization and deserialization of structured data between different services. Google’s design goal was to create a better method than XML to make systems communicate with each other over a wire or for the storage of data. Since its development, Google has made Protobuf under an open source license. It also provides out of the box support in the most common languages, including Python, Java, Objective-C, C# and others via Google’s new proto3 language version[8].

A JSON string is easy to generate, but it is so hard to parse when using C language, because this language do not  have built-in support for OOP concepts. Protobuf help ESP32 to parse APIs exactly and easily, and send to other devices easily.

## Security

Every users sign in to my system, server will generate a token and assign it on Cookies of client.

JSON Web Token (JWT) is an open standard ([RFC 7519](https://tools.ietf.org/html/rfc7519)) 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[9].

This access token help client to get or post data to server, because it contain 2 keys, 1 key of client, 1 key of server. After server use it’s key to decode access token, it will have key of client, then it will use this key to fine and check information of user.

# Chapter 3: DATABASE



## Intruduction

### Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a [database management system (DBMS)](https://www.oracle.com/database/what-is-database.html#WhatIsDBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database[10].

My system is service muilti-user, it will have problem if I use file system to save data of users. Other way, server shouldn’t becom a storage, I need 1 thing which allows me to save data with a easy structure , but it can be access easily.

### Databse management system

A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data[11].

### SQL

Structured Query language (SQL) pronounced as "S-Q-L" or sometimes as "See-Quel" is the standard language for dealing with Relational Databases. A relational database defines relationships in the form of tables[12].

### NoSQL database

NoSQL databases (aka "not only SQL") are non tabular, and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads[13].

## MongoDB

MongoDB is a [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) [document-oriented database](https://en.wikipedia.org/wiki/Document-oriented_database) program. Classified as a [NoSQL](https://en.wikipedia.org/wiki/NoSQL) database program, MongoDB uses [JSON](https://en.wikipedia.org/wiki/JSON)-like documents with optional [schemas](https://en.wikipedia.org/wiki/Database_schema). MongoDB is developed by [MongoDB Inc.](https://en.wikipedia.org/wiki/MongoDB_Inc.) and licensed under the Server Side Public License[14].

MongoDB is supported on many platforms , it includes nodejs, it is easy to use for applications like my server. MongoDB support a global cloud database service.

## Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB[15].

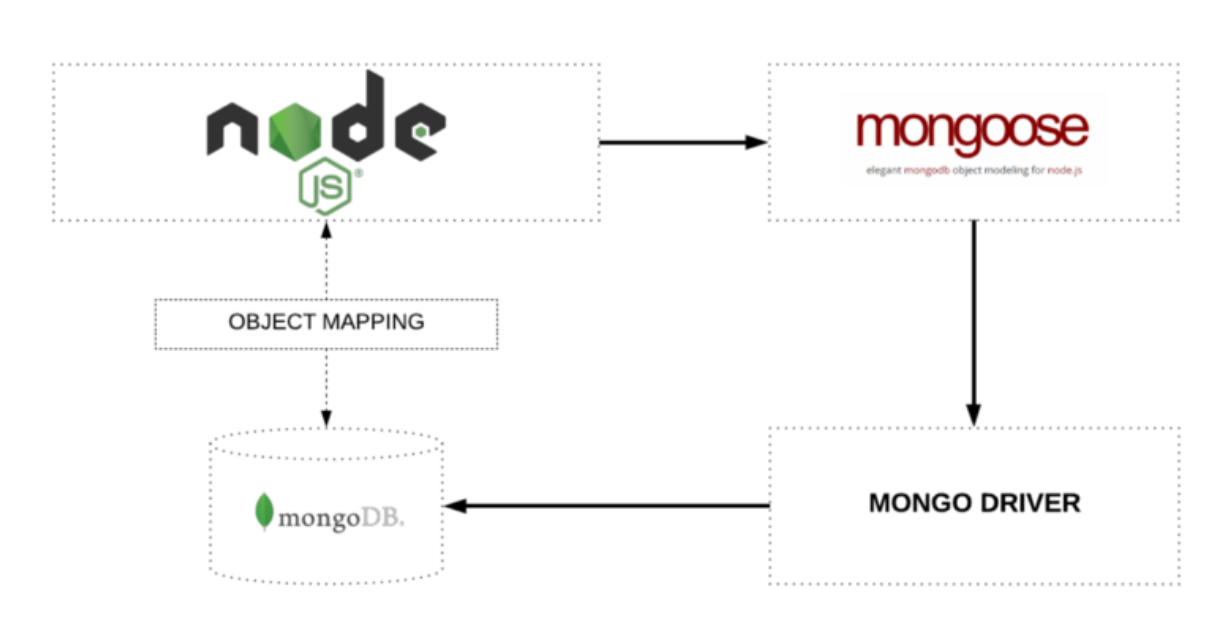


Figure 3. 1 Operation of Mongoose

With Mongoose, I can make a model for document easily, it supports many functions with filters when fine, update, delete, create documents.

## MongoDB Atlas

MongoDB Atlas is the global cloud database service for modern applications. Deploy fully managed MongoDB across AWS, Azure, or GCP. Best-in-class automation and proven practices guarantee availability, scalability, and compliance with the most demanding data security and privacy standards[16].

A cloud database will allow my server access from other cloud and hand to the database. This is my batabase stored on MongoDB Atlas:

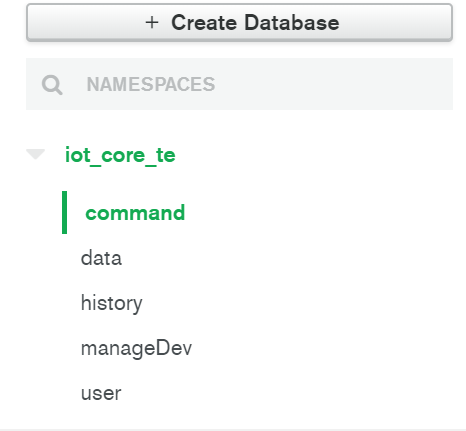


Figure 3. 2 Database iot\_core\_te

# Chapter 4: DEVICE



## Introduce about ESP32 Microcontroller

ESP32 is a single chip 2.4 GHz Wi-Fi and Bluetooth combo chip designed with TSMC ultra low power 40 nm technology. It is designed and optimized for the best power performance, RF performance, robustness, versatility, features and reliability, for a wide variety of applications, and different power profiles[17].

CPU of ESP32 is Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, it operates at 160 or 240 MHz and performing at up to 600 [DMIPS](https://en.wikipedia.org/wiki/Dhrystone), and supports Ultra low power (ULP) co-processor. In addition, its memory is 512 KiB SRAM.

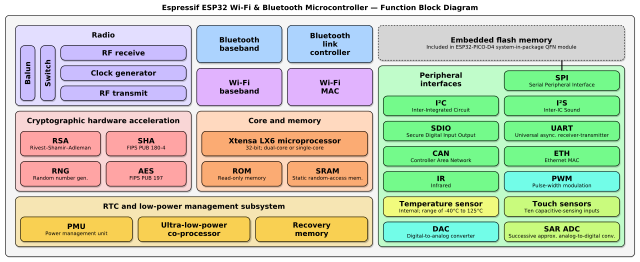


Figure 4. 1 Function block diagram of ESP32

In my thesis, I choose ESP32-WROVER module and ESP32 Pico D4 Kit for building test model:

* ESP32-WROVER-V4 module : PSRAM make it more powerful.

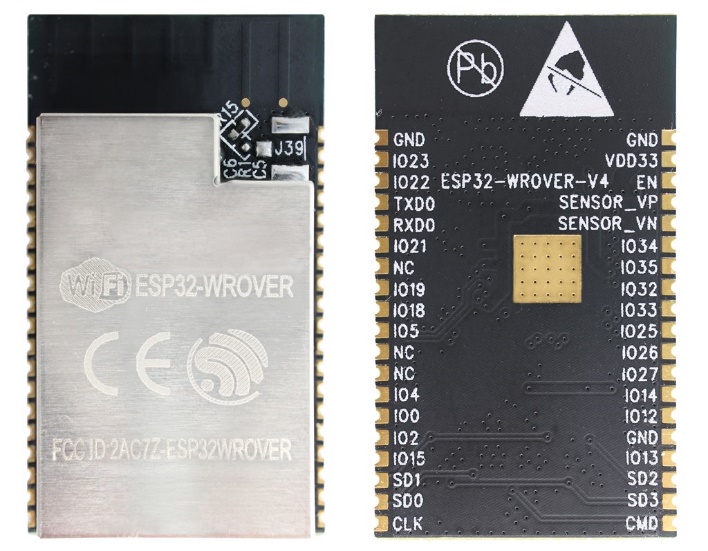


Figure 4. 2 ESP32-WROVER-V4 module

* ESP32 Pico D4 kit : it uses ESP-PICO-D4 which is a System-in Package(SiP) module.

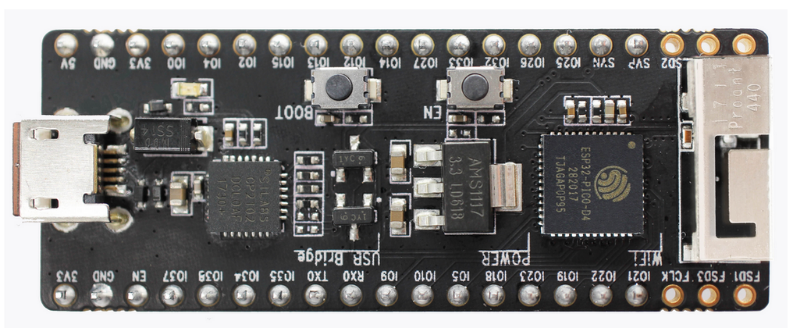


Figure 4. 3 ESP32 Pico D4 kit

### Adapter for ESP32-WROVER-V4

ESP32-WROVER-V4 module needs an adapter to be programed and connected to sensors or equipments. So, I has designed the adapter with support from KiCad sortware.

KiCad is an open-source software tool for the creation of electronic schematic diagrams and PCB artwork. It support macOS, Microsoft Windows, Ubuntu,…We Just download and use easily.

### Schelmatic

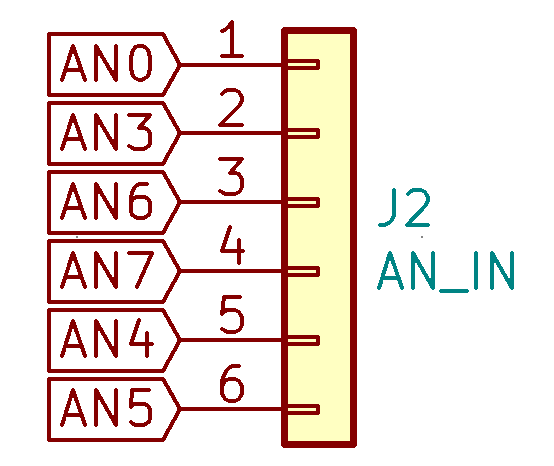


Figure 4. 4 ADC Input

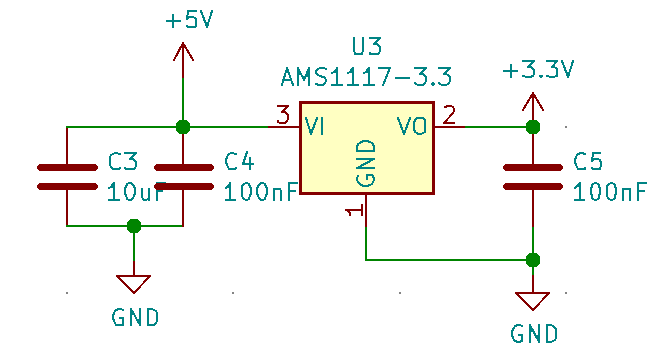


Figure 4. 5 Buck Converter 5v to 3v3 with IC AMS1117

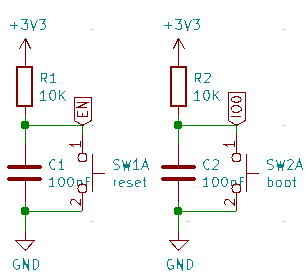


Figure 4.6 Boot button and reset button

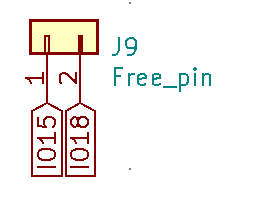
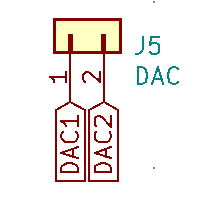


Figure 4.7 DAC

Figure 4.8 GPIO

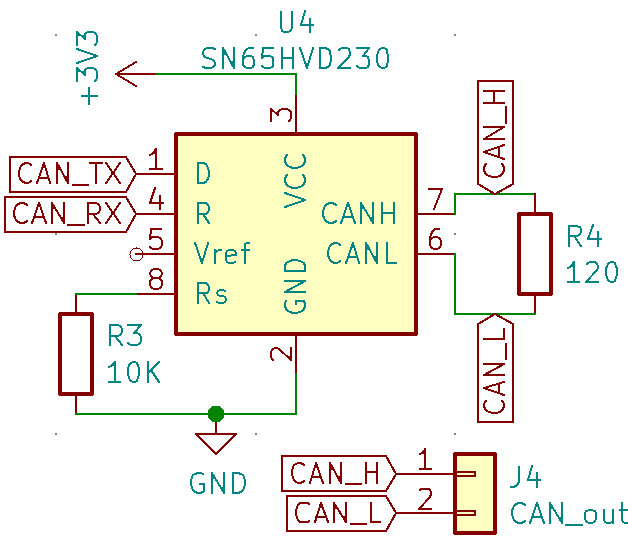


Figure 4.9 CAN

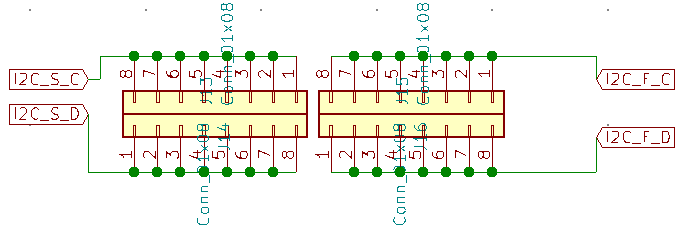


Figure 4.10 I2C

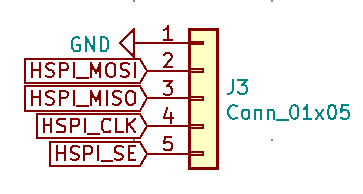


Figure 4.11 SPI

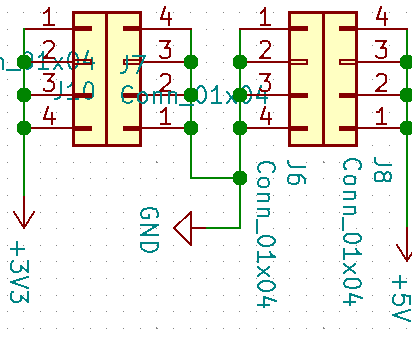


Figure 4.12 output of 3.3V and 5V

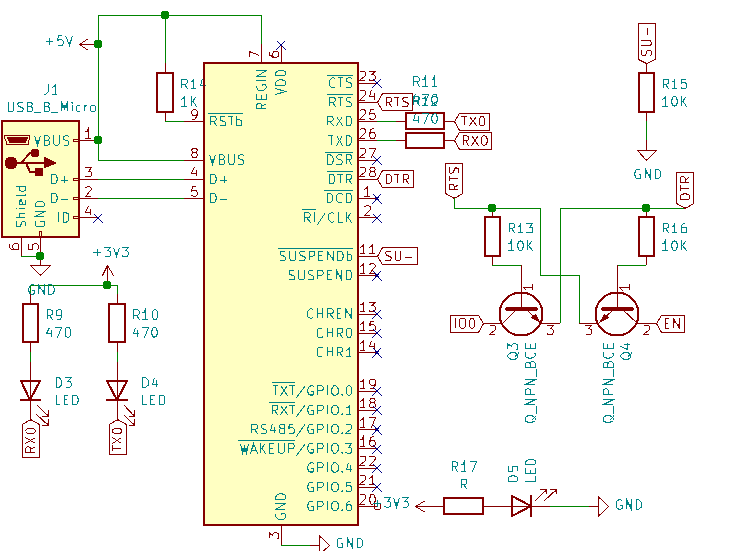


Figure 4. 13 USB flash through uart

### PCB images

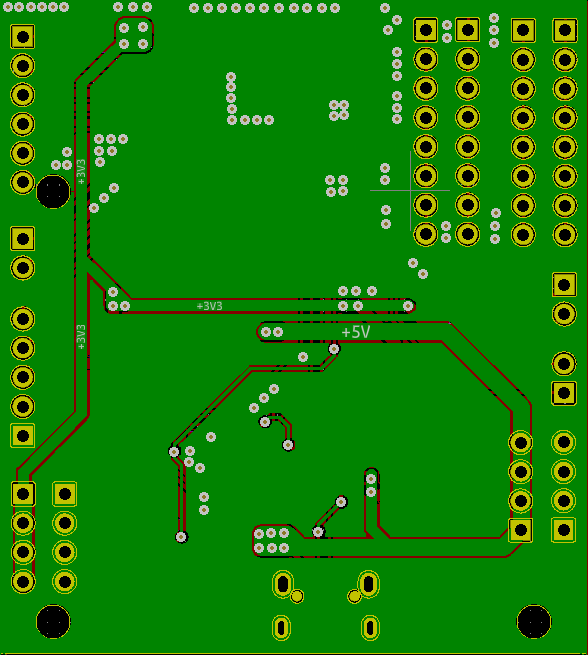
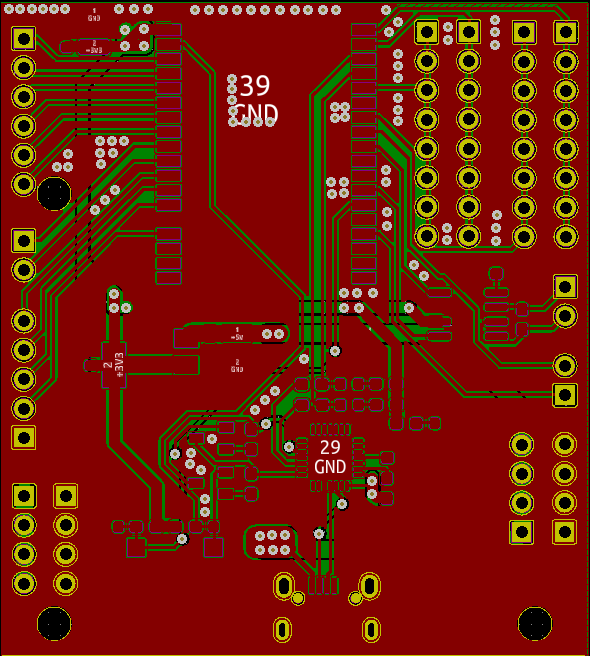


Figure 4. 14 PCB-top Figure 4. 15 PCB-bottom

### Production

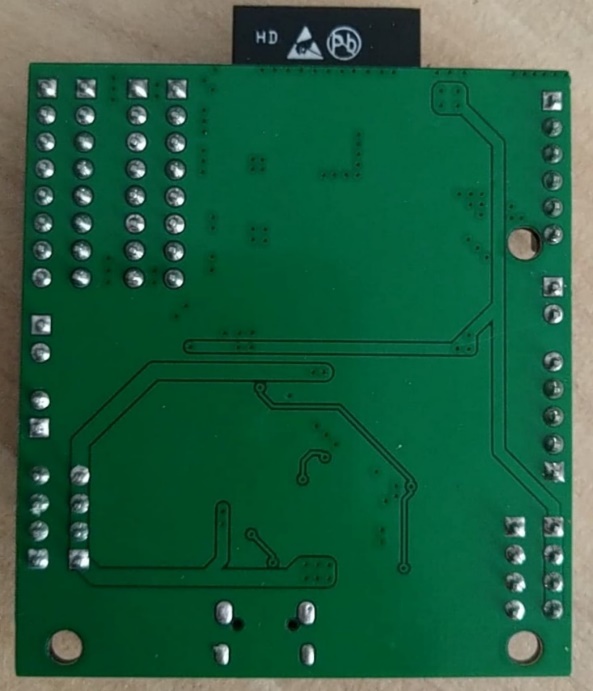


Figure 4. 16 Bottom of adapter Figure 4. 17 Top of adapter

## ESP-IDF

It means Espressif IoT Development Framwork. This framework support many APIs for peripherals, protocols,… ESP-IDF is flexible, easy for using and developing. In additionally, ESP-IDF is integrated FreeRTOS, a real-time operating system.

## Selt-made components.

I has make some components, which is helpful for programming template on devices.

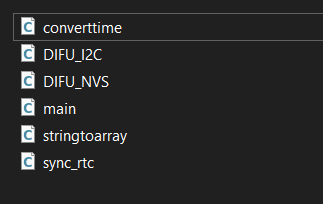


Figure 4. 18 Self-made components

* main.h: It includes some necessary libraries for template, I make a struct which contains parameters of devices.

----------------------------------------------------------------

typedef struct

{

char ID[13];

uint32\_t device;

uint8\_t user\_wifi[33];

uint8\_t pass\_wifi[65];

uint16\_t reg\_digi;

uint16\_t reg\_dac;

uint16\_t reg\_pwm;

}esp\_parameter\_t;

----------------------------------------------------------------

* DIFU\_I2C: It support functions to initialize parameters for I2C and use I2C with master mode.
* DIFU\_NVS: It support functions to make a repository on NVS and get or set data into it.
* sync\_rtc: It support functions to synchronize time on ESP with real time on Internet or DS3231 module, convert time between timestamp and string for displaying.
* stringtoarray.h: It support convert a string to array and return amount of element of this array, length of each element is 1 byte.
* converttime: It includes 1 function, which can be convert parameters of 1 time struct to timestamp.

## Algorithm flowchart

With FreeRTOS , I can distribute features follow tasks , ESP32 will proccess every task in their period. The task which communicate with server will have the highest priority to ensure that data are perfect.

I present the control diagram as below :

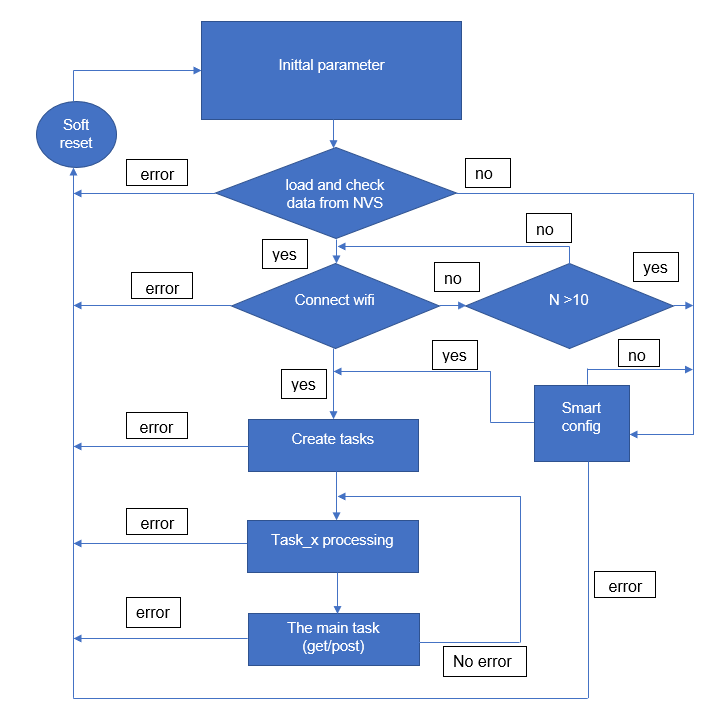
****

Figure 4. 19 Algorithm flowchart of devices

If any error occur, devices will be restarted, because errors can do harm devices, restarting will initialize all parameters.

## Config device with mobile app.

There are many mobile apps which support smart-config, users can use whatever mobile app they want.

In my thesis, I choose IoT Smartconfig, author of this app is “Tuan PM”. IoT Smartconfig is a free app on Google Play, users can download easily.



Figure 4. 20 GUI of IoT Smartconfig app

Syntax to config for ESP32:

-------------------------------------------------------------------------

<SSID>

<ID>.<Device index>.<Passwork>

-------------------------------------------------------------------------

* SSID : Name of Wi-Fi.
* ID : This is ID in Gmail of user after registering successfully.
* Device index : user can choose any number, let remember it to config on Config Page.
* Password : password of Wi-Fi.

Mobile phone of users has to use Wi-Wi which users want devices connect to and turn on location on mobile phone of users.

# Chapter 5: USER



## Keyword

* **Device:** It is ESP32, every device has 1 unique index ( 0 < index < 4294967295).
* **Sensor:** Which collects data of environment (other microprocessing, sensor,...).
* **Equipment:** Which you can control through sending command (didital , non-digital).
* **ID:** Identify of 1 user.
* **Type of device :** I divide devices into 2 types ( it can be expensed in the future) : 0 ( to get command from server and do it ) and 1 ( to send data to server ). It is GET method and POST method in HTTP. Devices which is type 0 manage Equipments, device which is type 1 manage Sensors ( read data).

## Pages of web app

### Register Page

If users want to use my application, they have to register an account through their Gmail. If they fail, they will receive a notification in this page, or if they are successful, their browser will be redirected to Login Page. And then, users will receive an ID in their email which is used to config devices.

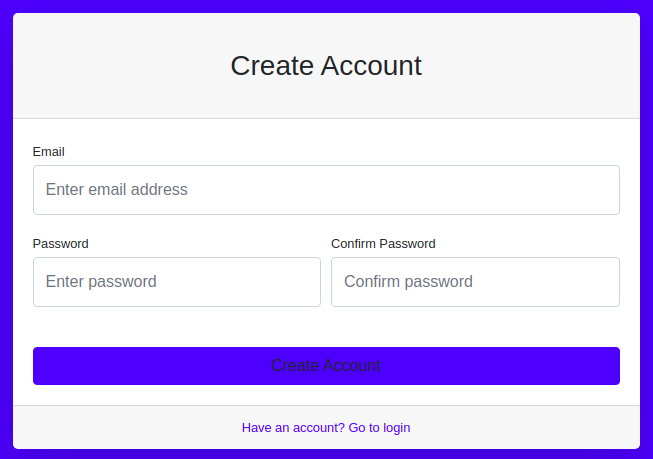


Figure 5. 1 Register card

### Login Page

Only users who have permission to sign in, anyone tries accessing to path ‘/user’ or its sub-path will be redirected to Login Page. When users sign in successfully, their browser will be redirected to User Page. In addition, if users forget their password, they can click “Forgot password?” to reset password.

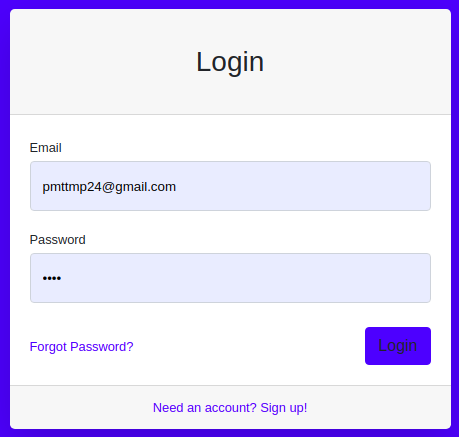


Figure 5. 2 Login card

### Help Page

In this page, users need to fill their email and then they will receive 1 number. This number is used for verifying their email.

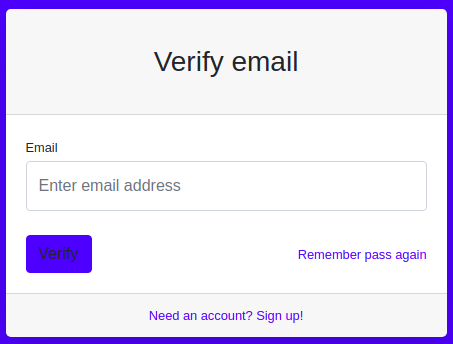


Figure 5. 3 Verify email card

### Verify Page

It is very easy, you just fill email again and the number you received in email. If everything is correct, users will receive second message in email, it contains a new password. Users should update password after signing in with new password.

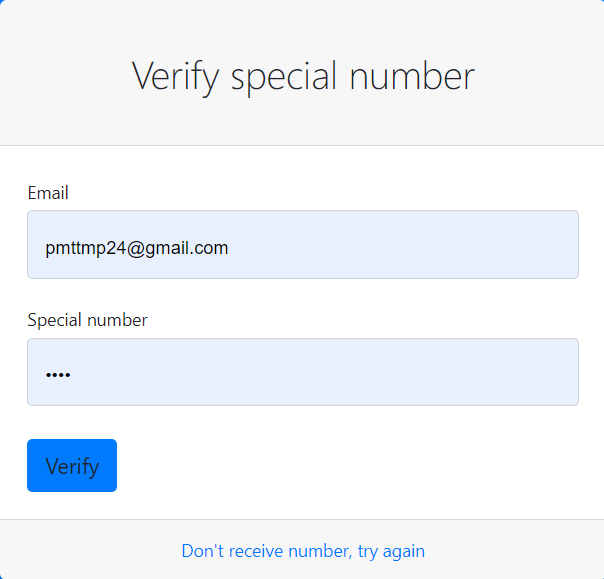


Figure 5. 4 Verify number card

### User Page

The dashboard will display data of the first device,which is type 1, users can see charts, data table

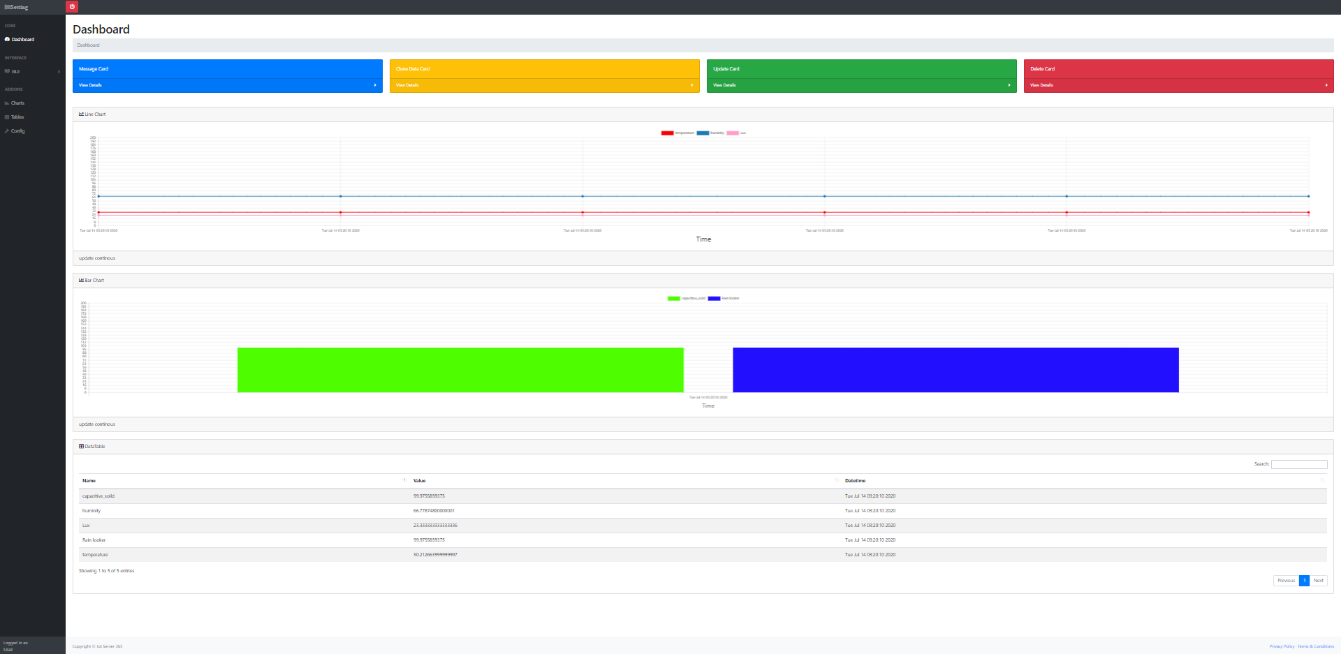


Figure 5. 5 User Page

With pattern dashboard, I can fix and add my options. When users press the letter “Setting” in the upper left corner of page, a list will appear. I will summarize in one flowchart as below:



Figure 5. 6 Structure of Setting

I supoort 4 card, with 3 cards are used, 1 card may be used in the future, users can fine it on “Setting 🡪 INTERFACE 🡪GUI 🡪Features”.

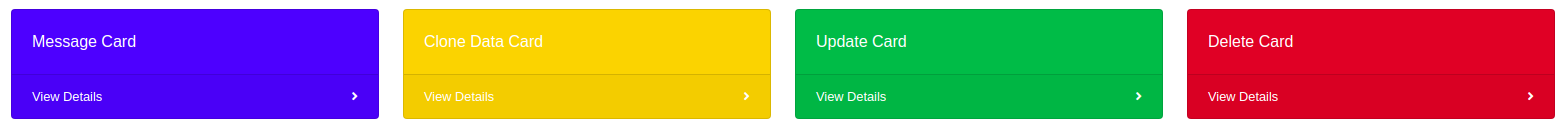


Figure 5. 7 Cards of User Page

I just support 6 data points on every chart, to see more, users need to go to Display Page.

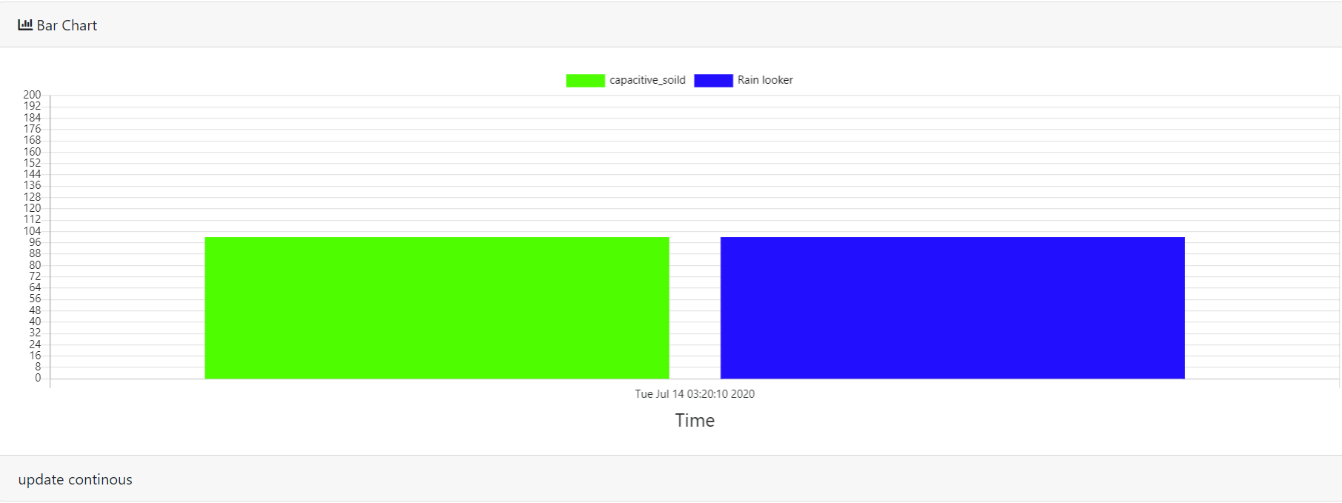


Figure 5. 8 Bar chart



Figure 5. 9 Line chart

Data table just show the lastest data of first device. To have more features, users have to go to Data Table Page.

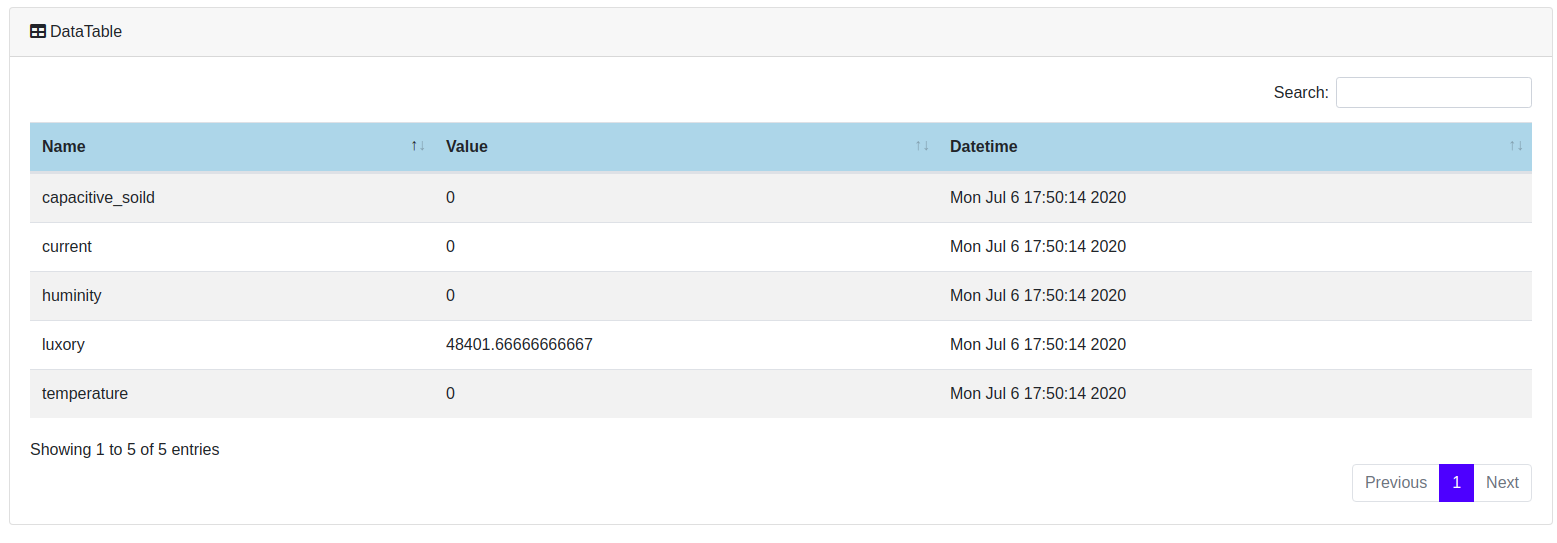


Figure 5. 10 Data table

### Display Page

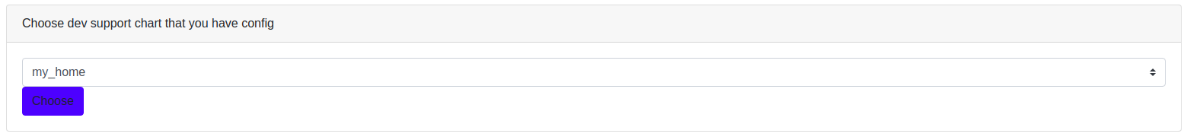


Figure 5. 11 Feature of Display Page

This page includes charts which look like charts in User Page, but it allows users to choose devices, and after pressing “Choose” button, charts will be update continuously.

### Data Table Page

In this page, there are 2 options:

* Filling amount of data and press green button, users will get the lastest 1500 data.
* Selecting period time, its resolution is 1 minute, then press blue buttons, users will get all data which server can send.

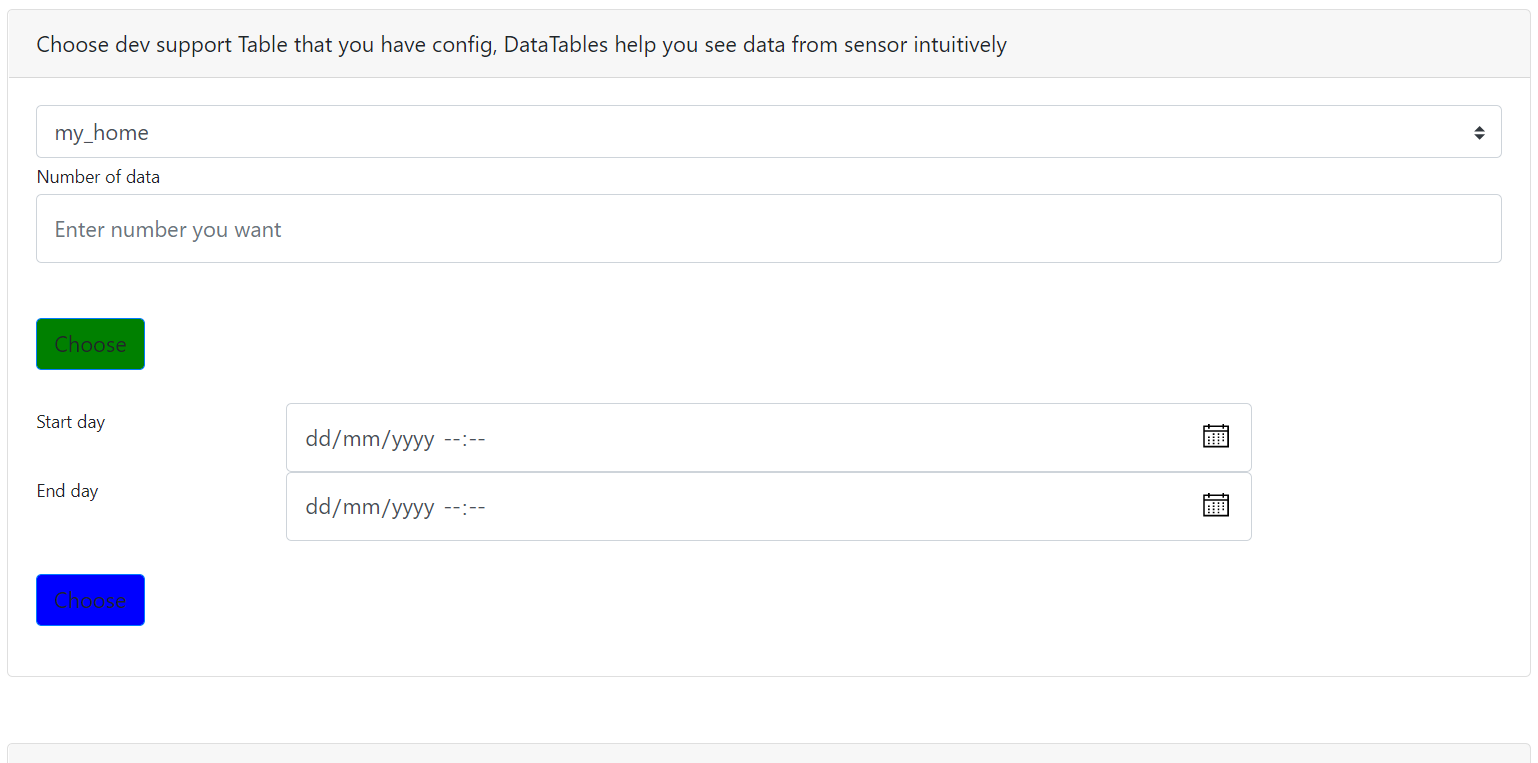


Figure 5. 12 Features of Data Table Page

### Update Page

In this page, users can update password and ID. With 2 cards for 2 purposes, these parameters are independent.

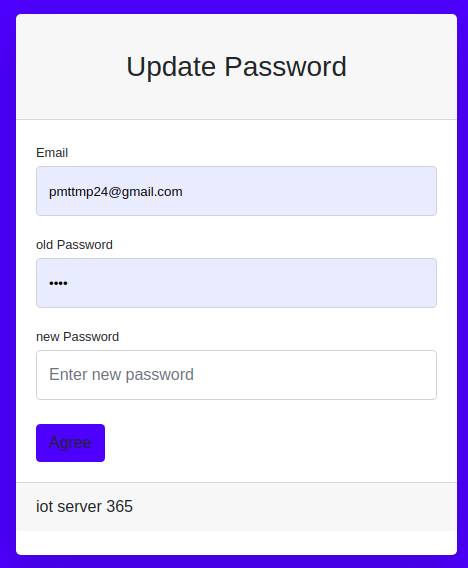


Figure 5. 13 Update Password card



Figure 5. 14 Update ID card

### Delete Page

In this page, users can delete their accounts, they should consider carefully because it is a one-way street.

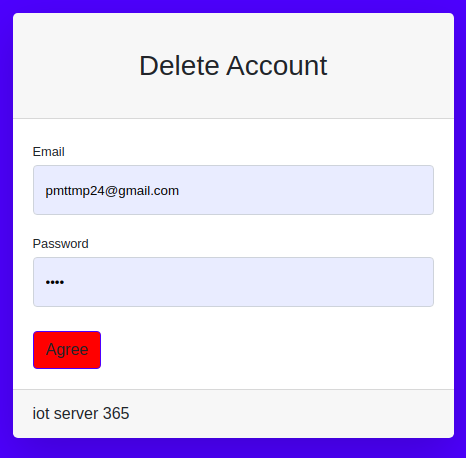


Figure 5. 15 Delete Account card

### Config Page

This is an important page, because it will allow users to add or remove devices which they want to use. In other way, users may increase or decrease number of equipments or sensors of devices, name of devices.

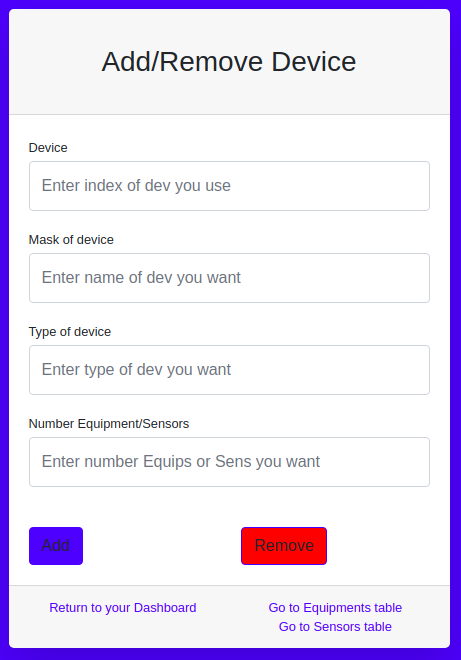


Figure 5. 16 Config card

### Equipment Page

There are 2 features: 1 record of devices with index of devices in every element, 1 search bar for finding devices with name.

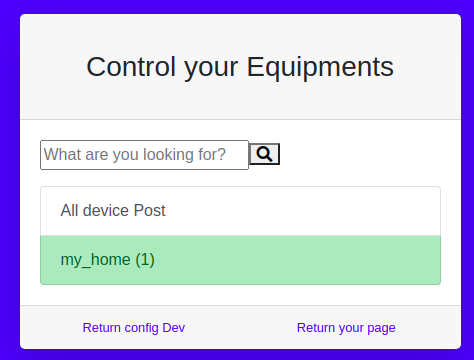


Figure 5. 17 Equipments record

### Equipi Page

Every device, which is type 1, has 1 Equipi Page, it allow users to config parameters. I make 6 cards with 6 parameters.

* This card allows users disable or enable equipments. After 1 equipment has been disabled, users can not change state of equipment.

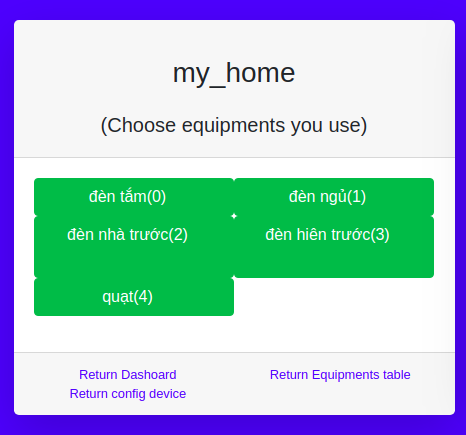


Figure 5. 18 Turning on/off equipments card

* There are 2 types of equipments, digital and non-digital. With digital type, it control on/off, with non-digital type, users just control PWM and DAC with value from 0 to 255.

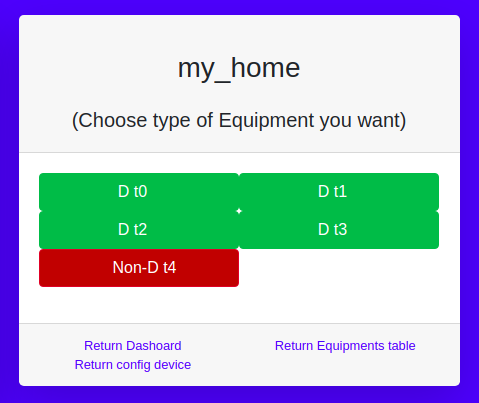


Figure 5. 19 Choosing type of Equipments card

* I allow users to change name of equipments, it is useful because users need name to distinguish equipments from each other.

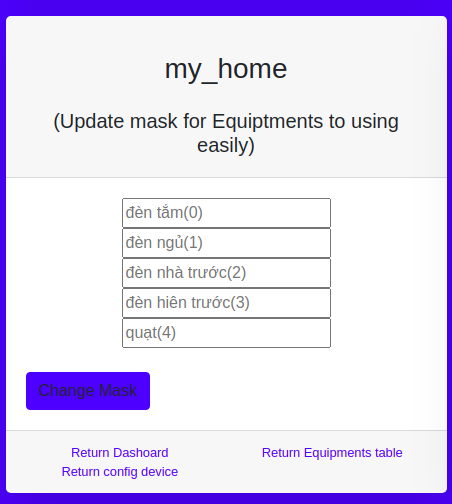


Figure 5. 20 Name of equipments card

* Port is used to gather pins into block. Every port should have a name, but users needn’t change here, it can change through creating block.

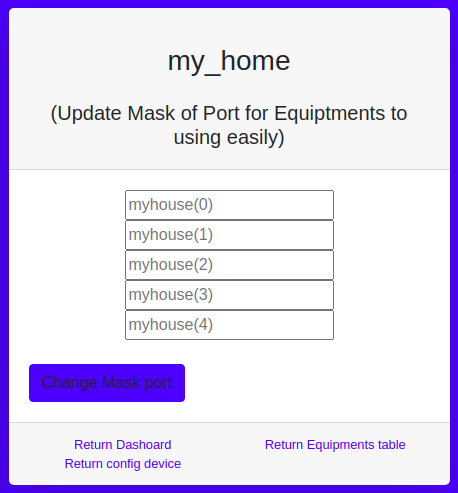


Figure 5. 21 Index of ports card

* Ports will have unique index, it help server process easily.



Figure 5. 22 Name od ports card

* Pin is index of pin on device, pins have to be in 1 device.

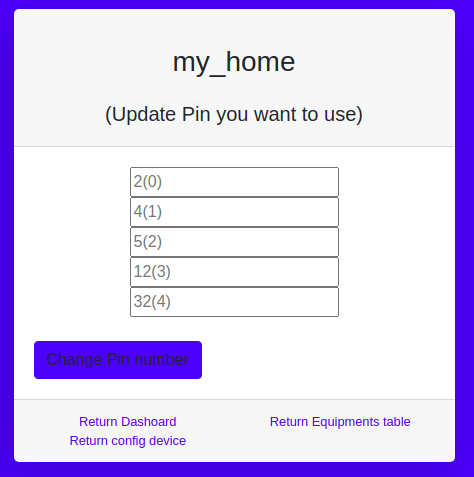


Figure 5. 23 Pins card

### Sensor Page

It is like Equipment Page with 1 search bar and 1 sensors record.

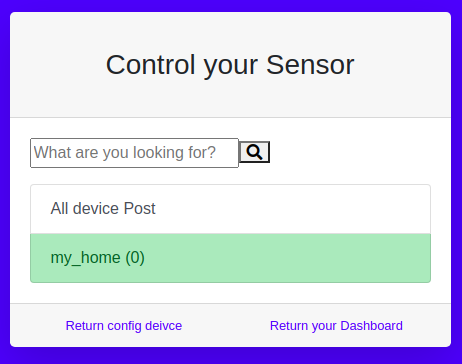


Figure 5. 24 Sensors record

### Sensi Page

Every device, which is type 1, has 1 Sensi Page, it allow users to config parameters. I make 3 cards with 3 parameters.

* This card allows users disable or enable sensors. After 1 sensor has been disabled, users can not see it in list of sensors.

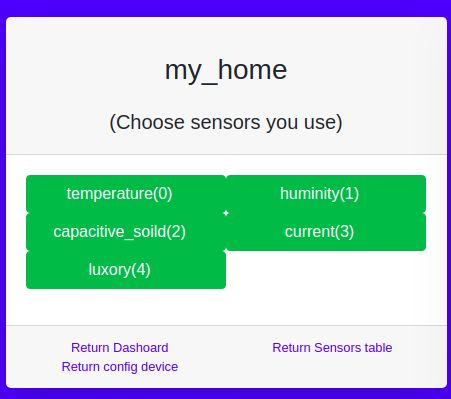


Figure 5. 25 Turning on/off sensors card

* I make 2 types of charts : line-chart and bar-chart.

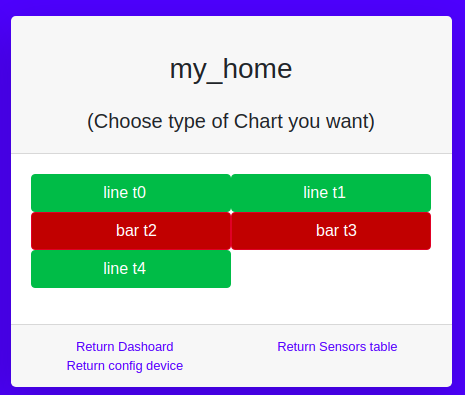


Figure 5. 26 Type of sensors card

* I allow users to change name of sensors, it will help users fine sensors they need easily and turn it on.



Figure 5. 27 Name of sensors card

### GUI Page

The index of block and block’s mask will be index of port and port’s mask of device. Block is assigned to 1 device everytime config.

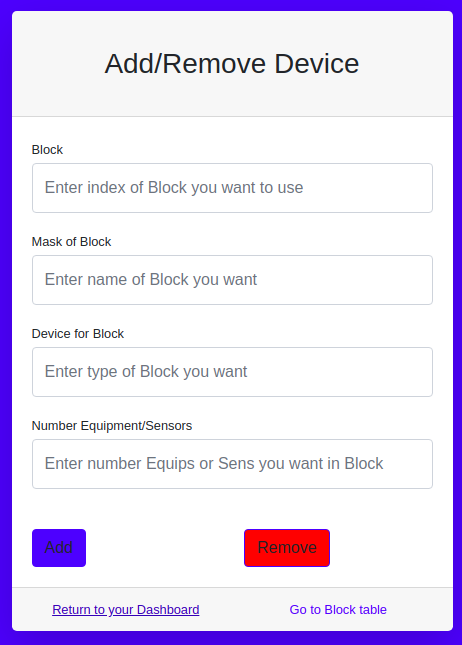


Figure 5. 28 Config block card

### Block Page

It is like Equipment Page with 1 search bar to fine block and 1 blocks record.

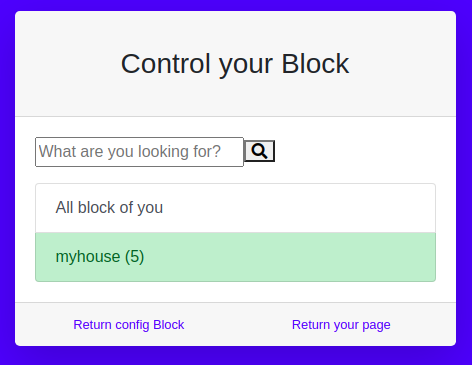


Figure 5. 29 Blocks record

### Control Block Page

I made 2 cards, one for digital type and one for non-digital type. Digital card includes switch buttons, non-digital card includes slider bars.

Equipments, which are disabled, will have apple icon or if they are enabled, they will have androi icon.

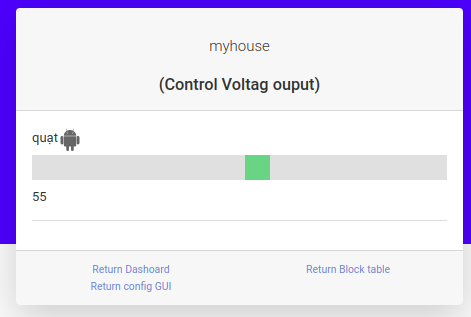
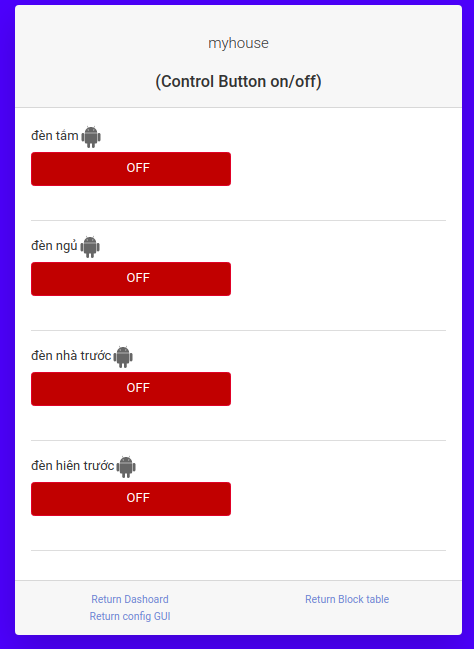


Figure 5. 30 Control Block Page

# Chapter 6: TEST MODEL



## House model

My model includes: 1 relay with 4 channels for 4 lights, a L298N module for controlling 1 DC fan, 1 DS3231 for RTC (using SNTP for time synchronization is default), 2 devices : ESP32 Pico D4 Kit for GET method and ESP32-WROVER board for POST method.



Figure 6. 1 House model

## Mobile app

I have tried building a mobile app for my system, but it is just a web plugin, like browser on mobile phones.

This app use Flutter, Flutter is Google’s UI toolkit for building beautiful, natively compiled applications for [mobile](https://flutter.dev/docs), [web](https://flutter.dev/web), and [desktop](https://flutter.dev/desktop) from a single codebase[18], it use Dart language. But with package “flutter\_webview\_plugin”, it does not support for downloading data, users only download data table with web app version now.

Dart is made by Google, it is a programming language optimized for building user interfaces with features such as the [spread operator](https://dart.dev/guides/language/language-tour#spread-operator) for expanding collections, and [collection if](https://dart.dev/guides/language/language-tour#collection-operators) for customizing UI for each platform[19].

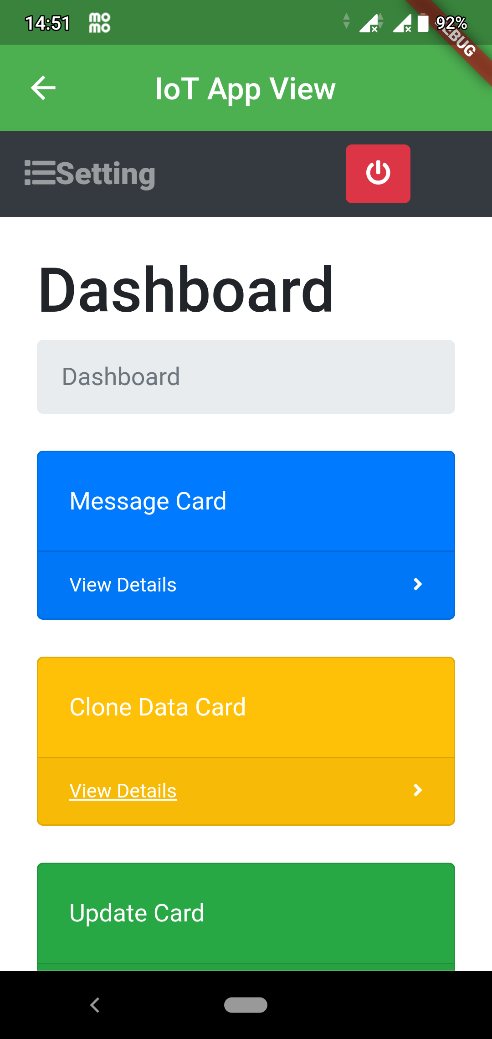
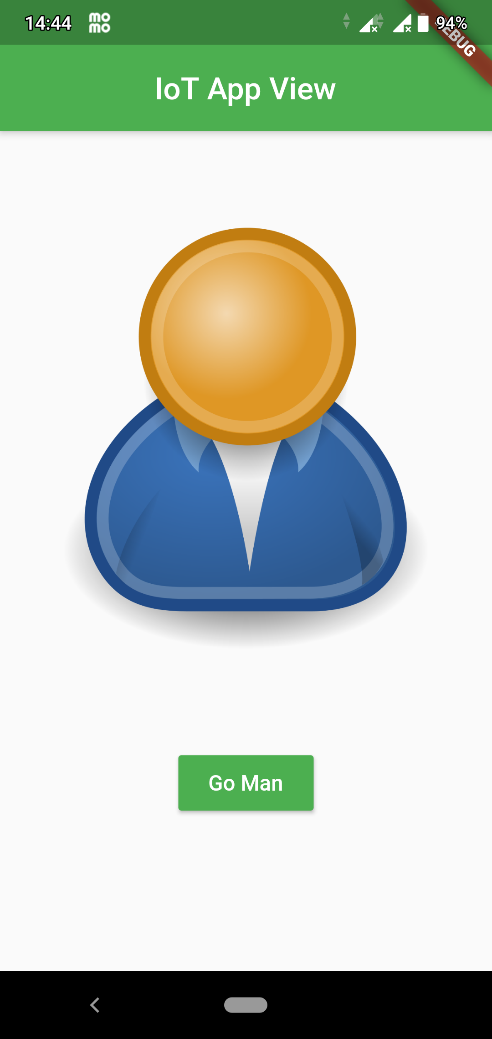


Figure 6. 2 Home Page and User Page of mobile app

This prove that developers can make application on mobile, PC with using APIs from my server.

# Chapter 7: SUMMERY



## Conclusion

* The results have been achieved:
  + 1 web complete web server.
  + 1 stable template for ESP32 with some self-made libraries.
  + 1 mobile web view app on mobile.
  + 1 stable web app.
* Limitations:
  + Latency on line is high(the lowest line latency is always greater 3 seconds).
  + Type of command is not diverse.
  + Do not have alarm feature.

## Future work

For further development, in the future, I want to expense more options such as:

* Imrpoving GUI, build a complete mobile app.
* Adding more types of commands to devices.
* Making a complete APIs, which can be port to other embedded devices.
* More options for database management system.
* Adding alarm feature.

BIBLIOGRAPHY

1. Online. Retrieved from <https://cloud.google.com/iot-core>
2. Online. Retrieved from <https://docs.blynk.cc/>
3. Online.Retrieved from <https://www.tutorialspoint.com/mvc_framework/mvc_framework_quick_guide.htm>
4. Online. Retrived from <https://github.com/StartBootstrap/sb-admin-angular>
5. Online. Retrived from <https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol>
6. Online. Retrived from <https://en.wikipedia.org/wiki/HTTPS>
7. Online. Retrived from <https://restfulapi.net/http-methods/>
8. Online. Retrived from <https://www.bizety.com/2018/11/12/protocol-buffers-vs-json/>
9. Online. Retrived from <https://jwt.io/introduction/>
10. Online. Retrived from <https://www.oracle.com/database/what-is-database.html>
11. Online. Retrived from <https://www.techopedia.com/definition/24361/database-management-systems-dbms>
12. Online. Retrived from <https://www.guru99.com/sql-vs-nosql.html>
13. Online. Retrived from <https://www.mongodb.com/nosql-explained>
14. Online. Retrived from <https://en.wikipedia.org/wiki/MongoDB>
15. Online. Retrived from <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>
16. Online. Retrived from <https://www.mongodb.com/cloud/atlas>
17. Online. Retrived from <https://datasheet.octopart.com/ESP32-D0WDQ6-Espressif-Systems-datasheet-138896080.pdf>
18. Online. Retrived fromv <https://flutter.dev/>
19. Online. Retrived from <https://dart.dev/>
20. Nodejs 12.x.x document. Retrieved from [https://nodejs.org/dist/latest- v12.x/docs/api/](https://nodejs.org/dist/latest-%20%20%20%20%20%20v12.x/docs/api/).
21. NPM 6.x.x document. Retrieved from <https://docs.npmjs.com/>.
22. StartBootstrap-sb-admin. Retrieved from github [https://github.com/StartBootstrap/ startbootstrap-sb-admin](https://github.com/StartBootstrap/%20%20startbootstrap-sb-admin).
23. ESP-IDF 4.1 document with API Reference. Retrieved from [https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api- reference/index.html](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-%20%20%20%20%20%20reference/index.html).
24. ESP-IDF 4.1. Retrieved from github <https://github.com/espressif/esp-idf> .
25. Mongoose 5.xx document. Retrieved from <https://mongoosejs.com/docs/documents.html>.
26. Mongodb 4.x . Retrieved from <https://docs.mongodb.com/manual/core/document/>