**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**

**FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING**

**DEPARTMENT OF CONTROL ENGINEERING & AUTOMATION**

**GRADUATION THESIS**

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

**HO CHI MINH CITY, JULY 2020**

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

**FACULTY OF ELECTRICAL & ELECTRONICS ENGINEERING**

**DEPARTMENT OF CONTROL ENGINEERING & AUTOMATION**

**GRADUATION THESIS**

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

**SUPERVISOR : PhD NGUYEN TRONG TAI**

**STUDENT : PHAM MINH THUAN – 1613435**

**HO CHI MINH CITY, JULY 2020**

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

Electrical & Electronic Engineering

SOCIALIST REPUBLIC OF VIETNAM

Independence – Freedom – Happiness

Control Engineering & Automation

**GRADUATION EVALUATION OF SUPERVISOR**

**Name of Thesis:**

**IOT APPLICATIONS IN THE CONSTRUCTION OF EQUIPMENTS MONITORING AND MANAGEMENT SYSTEM IN HOME**

**Student: Supervisor:**

Pham Minh Thuan 1613435 PhD Nguyen Trong Tai

**Eluvation:**

1. Thesis book:

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

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

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

Remarks about the presentation of the thesis book:

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

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

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

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

1. Thesis evaluation:

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

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

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

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

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

Electrical & Electronic Engineering

SOCIALIST REPUBLIC OF VIETNAM

Independence – Freedom – Happiness

Control Engineering & Automation

**GRADUATION EVALUATION OF SUPERVISOR**

**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. Thesis evaluation:

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

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

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

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

1. Potential applications:

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

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

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

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

1. Working attitude of student:

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

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

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

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

**Overall evaluation:**

**Score of student:**

Pham Minh Thuan: …………/10

**Judge**

(Sign and full name)

**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**

[TÓM TẮT LUẬN VĂN 2](#_Toc502147723)

[CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI 3](#_Toc502147724)

[1.1 Tính cấp thiết của Luận văn 3](#_Toc502147725)

[1.2 Mục tiêu của Luận văn 4](#_Toc502147726)

[1.3 Đối tượng và phạm vi nghiên cứu 5](#_Toc502147727)

[1.4 Nhiệm vụ và nội dung của Luận văn 5](#_Toc502147728)

[CHƯƠNG 2: GIỚI THIỆU TỔNG QUAN VỀ HEXAPOD 7](#_Toc502147729)

[2.1. Tổng quan về thiết kế và điều khiển tay máy song song Hexapod theo mô hình Stewart – Gough Platform 7](#_Toc502147730)

[2.1.1 Giới thiệu sơ lược về tay máy song song 7](#_Toc502147731)

[2.1.2 Tình hình nghiên cứu về tay máy song song kiểu Stewart – Platform 8](#_Toc502147732)

[2.2 Phân loại Hexapod 14](#_Toc502147733)

[2.3 Mô hình hóa Hexapod 18](#_Toc502147734)

[2.3.1 Lựa chọn mô hình thực hiện 18](#_Toc502147735)

[2.3.2 Phương trình động học mô tả hệ thống 20](#_Toc502147736)

[2.4 Khảo sát khả năng hoạt động của mô hình đã xây dựng 27](#_Toc502147737)

[2.4.1 Yêu cầu của mô hình 27](#_Toc502147738)

[2.4.2 Thông số đề xuất 28](#_Toc502147739)

[2.4.3 Khảo sát khả năng hoạt động của mô hình bằng Matlab 28](#_Toc502147740)

[CHƯƠNG 3: THIẾT KẾ, THI CÔNG MÔ HÌNH HEXAPOD 47](#_Toc502147741)

[3.1 Thiết kế mô hình bằng SolidWorks 47](#_Toc502147742)

[3.2 Bản vẽ thiết kế 47](#_Toc502147743)

[CHƯƠNG 4: BỘ ĐIỀU KHIỂN, PHẦN MỀM VÀ GIẢI THUẬT ĐIỀU KHIỂN MÔ HÌNH 61](#_Toc502147744)

[4.1 Bộ điều khiển Hexapod và các thiết bị chấp hành 61](#_Toc502147745)

[4.1.1 Chip điều khiển STM32F407VG 61](#_Toc502147746)

[4.1.2 Chip điều khiển Max II EMP570GT100C4 61](#_Toc502147747)

[4.1.3 Driver PID điều khiển động cơ Servo 62](#_Toc502147748)

[4.1.4 Thiết kế mạch điều khiển 63](#_Toc502147749)

[4.1.5 Động cơ điều khiển 69](#_Toc502147750)

[4.2 Xây dựng phần mềm điều khiển 70](#_Toc502147751)

[4.2.1 Xây dựng phần mềm điều khiển trên máy tính bằng phần mềm Visual Studio V15 70](#_Toc502147752)

[4.2.2 Xây dựng phần mềm điều khiển bằng màn hình Weintek 6071iP 73](#_Toc502147753)

[4.3 Sơ đồ hệ thống kết nối và giải thuật điều khiển 76](#_Toc502147754)

[4.3.1 Sơ đồ hệ thống kết nối 76](#_Toc502147755)

[4.3.2 Giải thuật điều khiển 78](#_Toc502147756)

[CHƯƠNG 5: THỬ NGHIỆM MÔ HÌNH 80](#_Toc502147757)

[5.1 Công cụ sử dụng để thử nghiệm mô hình 80](#_Toc502147758)

[5.2 Một số kết quả kiểm tra mô hình theo các trục 81](#_Toc502147759)

[5.2.1 Kiểm tra độ chính xác của mô hình theo trục z 81](#_Toc502147760)

[5.2.2 Kiểm tra độ chính xác của mô hình theo trục x, y 83](#_Toc502147761)

[5.2.3 Chạy mô hình theo các quỹ đạo định sẵn 85](#_Toc502147762)

[CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 86](#_Toc502147763)

[6.1 Kết luận 86](#_Toc502147764)

[6.2 Hướng phát triển của đề tài 86](#_Toc502147765)

[TÀI LIỆU THAM KHẢO 88](#_Toc502147766)

**List of Tables**

[Bảng 2.1 So sánh các dòng Hexapod có khả năng đáp ứng vị trí độ chính xác cao 14](#_Toc502147808)

[Bảng 2.2 So sánh các dòng Hexapod có khả năng đáp ứng chuyển động 15](#_Toc502147809)

[Bảng 2.3 So sánh các dòng Hexapod theo cơ cấu và cách thức truyền động 16](#_Toc502147810)

[Bảng 2.4 So sánh ưu nhược điểm của các mô hình 17](#_Toc502147811)

[Bảng 2.5 Chi tiết các bộ phận chính của mô hình 19](#_Toc502147812)

[Bảng 2.6 Thông số mô hình Hexarotor 28](#_Toc502147813)

[Bảng 2.7 Một số thông số chính của mô hình Hexapod 28](#_Toc502147814)

[Bảng 2.8 Khảo sát ảnh hưởng từng thông số đến vùng hoạt động của mô hình 33](#_Toc502147815)

[Bảng 2.9 Khảo sát mô hình cho 4 thông số cùng chạy 37](#_Toc502147816)

[Bảng 2.10 Khảo sát vị trí của tấm chuyển động với một trục x, y hoặc z cố định 39](#_Toc502147817)

[Bảng 2.11 Khảo sát vị trí của tấm chuyển động khi cho một góc Euler cố định 43](#_Toc502147818)

[Bảng 5.1 Một số thông số mô hình 80](#_Toc502147819)

[Bảng 5.2 Bảng thông số cảm biến đo khoảng cách HG-C1400 81](#_Toc502147820)

[Bảng 5.3 Kết quả thử kiểm nghiệm độ chính xác mô hình theo trục z 82](#_Toc502147821)

**List of Figures**

[Hình 1.1 Tàu sân bay Washington, ảnh chụp tháng 6, 2011 3](#_Toc502147851)

[Hình 2.1 Mô hình tay máy song song kiểu Stewart – Gough Platform 7](#_Toc502147852)

[Hình 2.2 Tay máy song song của hãng Dunlop Rubber 9](#_Toc502147853)

[Hình 2.3 Thiết bị mô phỏng bay của tiến sĩ Stewart 10](#_Toc502147854)

[Hình 2.4 Delta Robot 10](#_Toc502147855)

[Hình 2.5 Thiết bị gia công Hexapod – dự án của NIST 11](#_Toc502147856)

[Hình 2.6 Mô hình Hexapod PR6-02 trên thực tế 12](#_Toc502147857)

[Hình 2.7 Ứng dụng trong gia công cơ khí 13](#_Toc502147858)

[Hình 2.8 Ứng dụng trong Y học 13](#_Toc502147859)

[Hình 2.9 Ứng dụng trong mô phỏng 13](#_Toc502147860)

[Hình 2.10 Ứng dụng trong thiên văn 13](#_Toc502147861)

[Hình 2.11 Ứng dụng trong trắc địa 13](#_Toc502147862)

[Hình 2.12 Mô hình mẫu thiết kế Hexapod 19](#_Toc502147863)

[Hình 2.13 Hệ tọa độ quy chiếu 21](#_Toc502147864)

[Hình 2.14 Các hệ quy chiếu trong mô hình tay máy Hexapod 22](#_Toc502147865)

[Hình 2.15 Minh họa tọa độ của chân và Servo 23](#_Toc502147866)

[Hình 2.16 Hình chiếu trên mặt phẳng song song với Base 24](#_Toc502147867)

[Hình 2.17 Phần mềm khảo sát được xây dựng bằng Matlab – guide. 29](#_Toc502147868)

[Hình 2.18 Vẽ mô hình trong không gian khi tấm payload ở vị trí “Home” 30](#_Toc502147869)

[Hình 2.19 Tỷ lệ đạt được vùng hoạt động khi Platform Joint Angle thay đổi 33](#_Toc502147870)

[Hình 2.20 Tỷ lệ đạt được vùng hoạt động khi Base Joint Angle thay đổi 34](#_Toc502147871)

[Hình 2.21 Tỷ lệ đạt được vùng hoạt động khi Platform Radius thay đổi 34](#_Toc502147872)

[Hình 2.22 Tỷ lệ đạt được vùng hoạt động khi Base Radius thay đổi 35](#_Toc502147873)

[Hình 2.23 Tỷ lệ đạt được vùng hoạt động khi Servo Arm Length thay đổi 35](#_Toc502147874)

[Hình 2.24 Tỷ lệ đạt được vùng hoạt động khi Connecting Arm Length thay đổi 36](#_Toc502147875)

[Hình 2.25 Tỷ lệ đạt được vùng hoạt động khi Default Z Height thay đổi 36](#_Toc502147876)

[Hình 2.26 Tỷ lệ đạt được vùng hoạt động khi cho 7 thông số mô hình thay đổi 38](#_Toc502147877)

[Hình 3.1 Mặt trước của mô hình 48](#_Toc502147878)

[Hình 3.2 Mô hình nhìn từ trên xuống 48](#_Toc502147879)

[Hình 3.3 Bản vẽ chốt bắt ổ bi 49](#_Toc502147880)

[Hình 3.4 Bản vẽ trục gài ổ bi giữa cánh tay chống và cánh tay quay 50](#_Toc502147881)

[Hình 3.5 Bản vẽ trục bắt ổ bi giữa tay chống và bảng bắt khớp 51](#_Toc502147882)

[Hình 3.6 Bản vẽ bảng bắt động cơ 52](#_Toc502147883)

[Hình 3.7 Bản vẽ tấm đế dưới (Base) 53](#_Toc502147884)

[Hình 3.8 Bản vẽ thanh chống chân kết nối với tấm chuyển động (payload) 54](#_Toc502147885)

[Hình 3.9 Bản vẽ tấm chuyển động (payload) 55](#_Toc502147886)

[Hình 3.10 Bản vẽ trục của cánh tay chống 56](#_Toc502147887)

[Hình 3.11 Bản vẽ cánh tay quay 57](#_Toc502147888)

[Hình 3.12 Bản vẽ lắp của mô hình Hexapod hoàn chỉnh 58](#_Toc502147889)

[Hình 3.13 Ảnh chụp mặt trước của mô hình 59](#_Toc502147890)

[Hình 3.14 Ảnh chụp mặt dưới mô hình từ dưới lên 60](#_Toc502147891)

[Hình 4.1 Chip STM32F407VG của ST 61](#_Toc502147892)

[Hình 4.2 Chip Max II EMP570GT100C4 Altera 62](#_Toc502147893)

[Hình 4.3 Mạch điều khiển động cơ DC Servo PID 200W MSD-E21V1 62](#_Toc502147894)

[Hình 4.4 Khối nguồn 64](#_Toc502147895)

[Hình 4.5 Khối mạch STM và CPLD 64](#_Toc502147896)

[Hình 4.6 Khối ENCODER 65](#_Toc502147897)

[Hình 4.7 Khối ADC 65](#_Toc502147898)

[Hình 4.8 Một số khối giao tiếp 66](#_Toc502147899)

[Hình 4.9 khối PWM 66](#_Toc502147900)

[Hình 4.10 Khối Input 67](#_Toc502147901)

[Hình 4.11 Khối Output 67](#_Toc502147902)

[Hình 4.12 Ảnh 3D mặt trước của Main Board 68](#_Toc502147903)

[Hình 4.13 Ảnh 3D mặt sau của Main Board 68](#_Toc502147904)

[Hình 4.14 Ảnh Main board thực tế 69](#_Toc502147905)

[Hình 4.15 Động cơ DC Servo Harmonic được sử dụng 70](#_Toc502147906)

[Hình 4.16 Phần giao diện thực hiện chức năng chạy mô hình. 72](#_Toc502147907)

[Hình 4.17 Phần kết nối cổng COM giữa máy tính với mô hình 72](#_Toc502147908)

[Hình 4.18 Vùng cài đặt thông số mô hình 73](#_Toc502147909)

[Hình 4.19 Vùng điều khiển mô hình 73](#_Toc502147910)

[Hình 4.20 Màn hình chính điều khiển chương trình. 74](#_Toc502147911)

[Hình 4.21 Phần cài đặt thông số các động cơ cho mô hình 74](#_Toc502147912)

[Hình 4.22 Phần điều khiển trực tiếp mô hình 75](#_Toc502147913)

[Hình 4.23 Phần cài đặt các thông số mô hình 75](#_Toc502147914)

[Hình 4.24 Mô hình kết nối hệ thống 77](#_Toc502147915)

[Hình 4.25 Giải thuật điều khiển mô hình 78](#_Toc502147916)

[Hình 4.26 Giải thuật nội suy đường thẳng theo phương pháp xung chuẩn 79](#_Toc502147917)

[Hình 5.1 Cảm biến HG-C1400 80](#_Toc502147918)

[Hình 5.2 Bố trí thử nghiệm độ chính xác theo trục z của mô hình 81](#_Toc502147919)

**List of Keywords**

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Full** | **Meaning** |
| ADC | Analog to Digital Converter | Bộ chuyển đổi tín hiệu tương tự sang tín hiệu số |
| ASME | American Society of Mechanical Engineers | Hội kỹ sư cơ khí Mỹ |
| ARM | Advanced RISC Machine | Cấu trúc vi xử lý kiểu RISC |
| CAN | Controller Area Network | Chuẩn giao tiếp giữa vi điều khiển và ngoại vi không cần host computer |
| CNC | Computer Numerical Control | Hệ thống máy gia công cơ khí điều khiển bằng máy tính |
| CPLD | Complex Programmable Logic Device | Vi mạch khả lập trình được xây dựng trên nền EEPROM |
| CPU | Central Processing Unit | Bộ xử lý trung tâm |
| DAC | Digital to Analog Converter | Bộ chuyển đổi tín hiệu số sang tín hiệu tương tự |
| DC | Direct Current | Dòng điện một chiều |
| DMA | Direct Momory Access | Phương pháp cho phép thiết bị vào / ra gửi và nhận dữ liệu trực tiếp không cần thông qua CPU |
| FSMC | Flexible Static Memory Controller | Bộ điều khiển giao tiếp bộ nhớ ngoại vi |
| GUI | Graphical User Interface | Giao diện đồ họa người dùng |
| HMI | Human – Machine Interface | Giao diện tương tác người – máy |
| I2C | Inter – Intergrated Circuit | Chuẩn giao tiếp nối tiếp 2 dây giữa vi điều khiển và ngoại vi |
| IC | Integrated circuit | Vi mạch tích hợp |
| IDE | Intergrated Development Environment | Môi trường phát triển / lập trình tích hợp |
| IEEE | Institute of Electrical and  Electronics Engineers | Viện Kỹ sư Điện và Điện tử |
| IFAC | International Federation of  Automatic Control | Liên đoàn quốc tế về điều khiển tự động |
| IFToMM | International Federation for the Promotion of Mechanism and Machine Science | Liên đoàn quốc tế về Phát triển khoa học máy và Cơ cấu |
| IMU | Inertial Measurement Unit | Cảm biến góc quay |
| Matlab | Matrix Laboratory | Phần mềm cung cấp môi trường tính toán số và lập trình |
| MCU | Micro Controller Unit | Vi điều khiển |
| NIST | National Institute of Standards and Technology | Viện tiêu chuẩn và công nghệ quốc gia (Mỹ) |
| PC | Personal Computer | Máy tính cá nhân |
| PID | Proportional Integral Derivative | Bộ điều khiển vi tích phân tỷ lệ – Bộ điều khiển PID |
| PWM | Pulse Width Modulation | Điều chế độ rộng xung |
| SPI | Serial Peripheral Interface | Giao tiếp ngoại vi nối tiếp (SPI) |
| SRAM | Static Random Access Memory | Bộ nhớ truy cập ngẫu nhiên tĩnh |
| UART | Universal Asynchronous Receiver – Transmitter | Chuẩn truyền thông nối tiếp bất đồng bộ |
| USB | Universal Serial Bus | Chuẩn kết nối nối tiếp có dây trong máy tính |
| USART | Universal Synchronous Asynchronous Receiver – Transmitter | Chuẩn truyền thông nối tiếp bất đồng bộ / đồng bộ |

**ACKNOWLEDGEMENT**

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

In the world, we have many application , 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 to use it.

In this thesis, I endeavor to create a system, which include : 1 IoT-core (websever, database) , 1 template for micro-controller esp32 to communicate with IoT-core, 1 webapp to users can use to manage your house. My system support muilti-user, it mean many people can create your account and start to use it now. With HTTP protocol, not only esp32 serials but also a smart-phone, PC, laptop, stm32-serials, orther microcontroller, microprocessing... which can become a device and allow user to manage them. With using free platform, free language, free framwork, I will give for people a free application to use, develop. It allow developer can expanse feature, do not build again, create mobile-app with using API from Server, build many platfrom for other microprocessing/microcontroller, with users, it is easy to use or fix.

# Chapter 1:

# INTRODUCTION

* 1. Thesis overview

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

Applications of IoT appeared in home, Industrial Automation, Agriculture and so on. The common project we can mention : Smart-Home, Home Automation, Green Agriculture, Health Monitoring system, Smart Parking System.

Because I do not want the low-level belong to hight-level so much, my application have to allow users to choose option in a low-level and then config it on a hight-level to use, or just config whatever you want, then you can program device with function you have choosen. So, I decided to research some pattern system to learn how to do.

The figure below is structure of Google Cloud IoT Core(1),which give me a ideal :

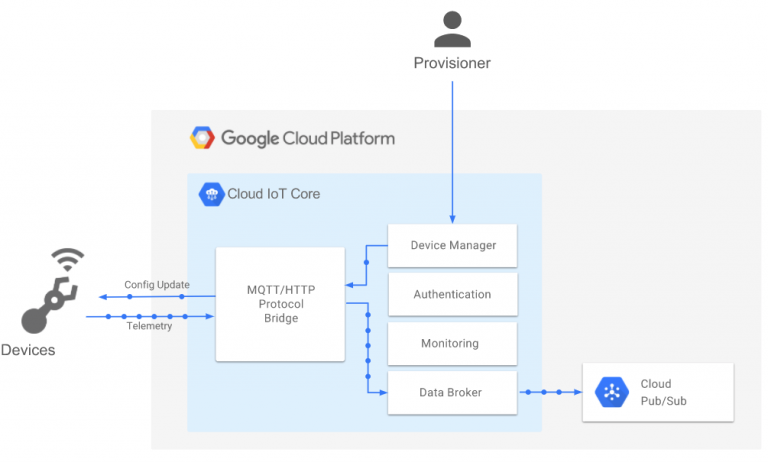


Figure 1.1 Diagram of Google Cloud IoT Core

That figure describe how to Google Cloud IoT Core operation, it will be a processing center and controller center, it also look like an environment where devices, provisoner, clouds become a block, and they can communicate with one another.

Because I feel this architecture of Google is so cool, in addition, I am expressed about mobile-app “Blynk”(2). All of them make me really want to create an Application, which can support GUI like “Blynk” but it is on the web browser ( not only on the mobile phone but also PC(3), tablet,…) , and my application have a enduring struct to support muilti-user, muilti-device.

Finally, I decide make a lightweight, powerful system, it have a API(4), have view engine, flow this diagram below:

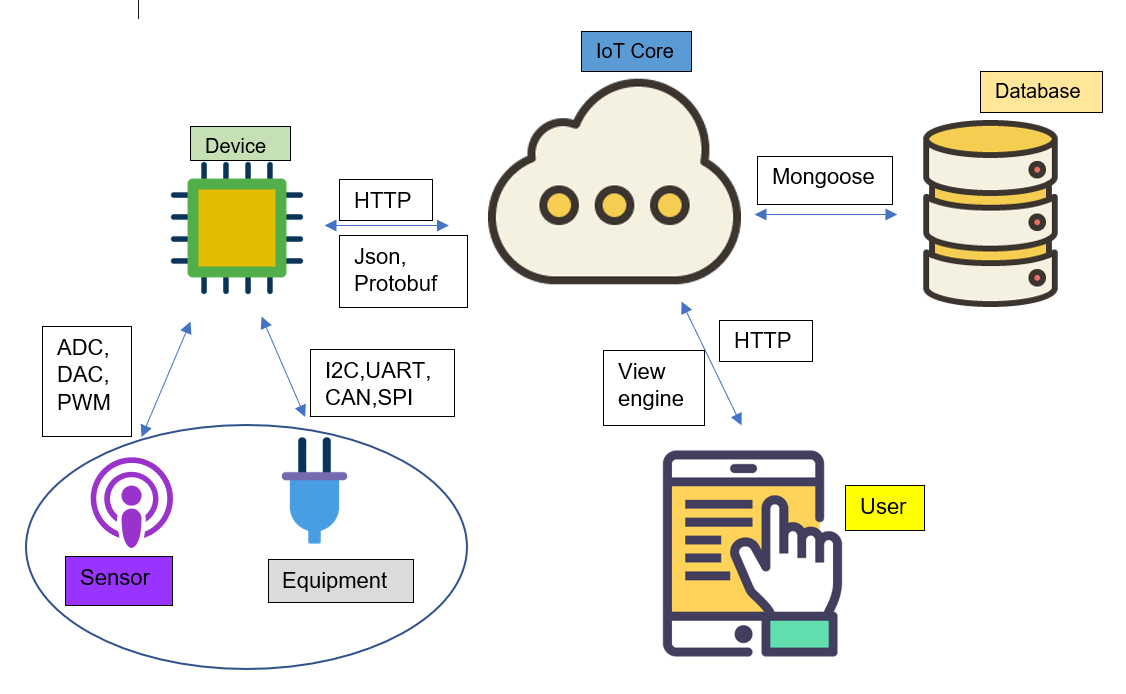


Figure 1.2 Diagram of system in thesis

**IoT Core**

This element is headquater of system, it like a “Big bridge” help other element can operate synchronously, smoothly. IoT core of my system is a websever, it is based on MVC(5) architectual.

**User**

This element represent for device like smartphone, PC,… or anything that is supported REST(6) and HTTP/HTTPS(7).

**Device**

This element is microprocessors, microcontrollers, which is supported HTTP, TCP/IP. Developer can program them for your purposes and users just use it with pattern platform.

**Database**

This element is used to store data, manage them flexibl. All data of user , data of command will be save here.

**Sensor**

This element represent for sensor , 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.

1.2 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 many other IoT(8) applicaion. In addition, hardware of device and system is not belong each other, it mean developers can create PCB and then program firmware with peripheral they want, after that , they just config on server and use it.

In order to gain this, the thesis needs to follow some tasks-core:

* Building a websever with MVC.
* Building a GUI(9) to users can communicate easily.
* Building template(10) for devices to interface with websever.
* Creating a database with simple form and lightweight storage.
* Building driver which support for communicating between device and sensors,equipments.

1.3 Thesis structure

I divide thesis into 6 chapter , belong to task and element of system:

**Chapter 1 :** Introdutrion : I will present overview about IoT , some Application about IoT , reason I want to create this system, mention issues when I begin and prerequisite tasks I need to gain in this thesis.

**Chapter 2:** IoT core : I will go to details about it, the reason I choose websever, how to build it with MVC architectual, how to communicate with device, how to build webapp with view engine.

**Chapter 3:** Database : I will introduce about MongoDB(11) , reason I choose it and which driver I choose to communicate with MongoDB.

**Chapter 4:** Device : I will present how I use esp-idf to build platform for esp32 microcontroller, how to using Json and Protobuf to communicate with IoT Core.

**Chapter 5:** User : I will introduce abount REST(12) and RESTful API(12) , how users can communicate with IoT core through GUI of webapp and manage your device through IoT core.

**Chapter 6:** Summary : I will introduce about results, limitations and development oritenation in the future of thesis.

# Chapter 2:

# IOT CORE

2.1. Structure of IoT core.

The my IoT Core is a websever, it is build from: JavaScript, HTML, CSS, PUG, NPM . JavaScript is a language for client, it mean it just run on browser like Firefox, CocCoc, Chomium, Edge,...But with the platform Nodejs, which have powerful engine V8, I can build my websever on it. With support from NPM , an open source, I can build sever in the great way.

Javascript do not have a “top function”, we will not have function like “int main()” in file named “main.js”. In theory , I can add anything in 1 file, but it will look terrible, and will be hard to fix bug. Those files need to be sorted into folder tree. But how? and what is standard ?

2.1.1.MVC architectual pattern.

“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.”( <https://www.tutorialspoint.com/mvc_framework/index.htm>)

**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”.

**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”.

**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”.

2.1.2 Inside my IoT Core.

I had spent a long time to understand MVC structure, and inherited their ideal. After all that, I represent a structure of my IoT Core :

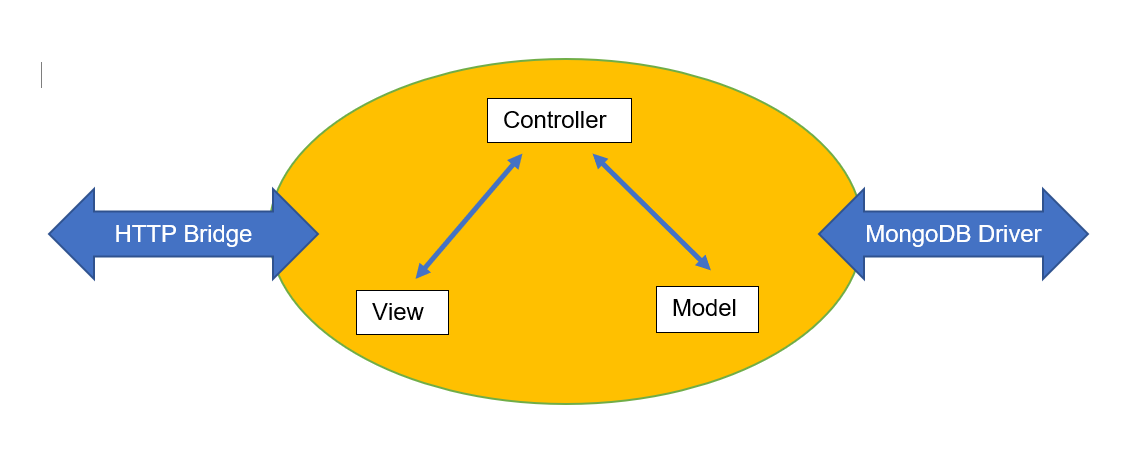


Figure 2.1 Diagram of IoT Core in thesis

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

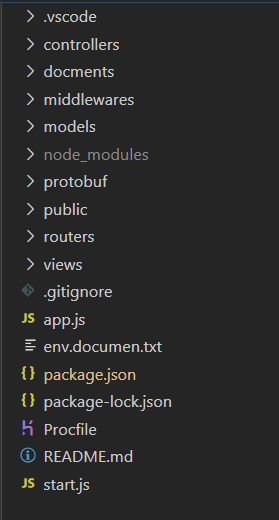


Figure 2.2 directory tree in the sever

I will explain folders and file one by one to make it easy to understand compatible things with MVC

* **Files in the top folder:**

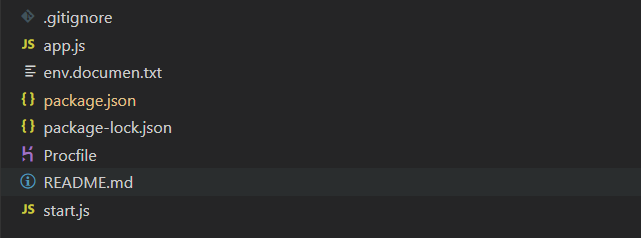
****

Figure 2.3

* app.js : it is the most important file, it will require package, config parameter or method of sever, setup database , declare target for URL , setup view engine and start sever.
* Package.json : it is is default file, which contain some setting on sever , like package used in the sever, name of main file, scripts to run.
* Package-lock.json : it is default file , which help reduce error when we add 2 version of 1 pakage.
* .gitignore : because I use github, I will add some file or folder which do not update when I fix my sever.
* Procfile : to declare the script to run sever, my sever is located in Heroku, which give me a resource , a domain name to use, so I need this file to Heroku know how to run.
* start.js : this is a file to manager the platform dasdboard, I use platform patern for view engine to make a GUI.
* **.vscode**



Figure 2.4

this folder is generated when I use Visual Studio Code, it contain some settings of developers, but with my sever, it just be a empty file.

* **routers**

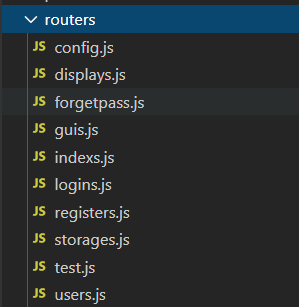
****

Figure 2.5

The sever have a URL to browser can request to it , every path is a target of main URL “<https://iot-server-365.herokuapp.com/>” . Every file routers manage 1 or some path.

* indexs.js : this manages router ‘/’, it is in roof of path. Because I think go to login page when user access webapp is not interesting.
* logins.js : this file will check information of users, if it is valid, users will receive a cookie to access your resourch, until they log out. It manages router ‘/login’.
* registers.js : this file will check information of registers, if they are new users, sever will create a new instance and store in database. It manages router ‘/register’.
* storages.js : this file manages router ‘/storage’, it is a gate to communicate with devices, where data will be post and command will be get.
* test.js : this file is test router , it helpful for creator to test a new add on feature.
* users.js : this file manages 5 routers :
* ‘/user’ : this router manages main page of user, this page is a dashboard, it give user many option and show some chart.
* ‘/user/update’ : this router manages update page, user can change your ID device , or change your password
* ‘/user/clone’ : this router manages clone page, this page is allow users clone your data they receive from devices with Excel, PDF, CSV format , or just print if users are connecting printer.
* ‘/user/logout’ : this router reset cookie of users and redirect them to login again, users just press ‘logout’ button.
* ‘/user/delete’ : this router will delete account of user, they will become register again, user should consider carefully.
* config.js : this file manages 7 routers:
* ‘/user/config’ : this router manages config page, this page allow users to add , remove device esp32 .
* ‘/user/config/sensors’ and ‘/user/config/equipments’: these routers manage sensor page and equipment page, these pages allow users to see your device’s record.
* ‘/user/config/sensors/search’ and ‘/user/config/equipments/search’: these routers will receive name of device you fine and go to its page, I think it is useful when you have many devices.
* ‘/user/config/sensors/:id’ and ‘/user/config/equipments/:id’ : these routers manage a page which contains parameter of device esp32.
* forgetpass.js: this file is a method to help users when they forget your password and can not login, it include 2 routers :
* ‘/forgetpass’: users will send to this router their email, and it will send a authentication message to their email.
* ‘/forgetpass/verify’: this router will check your number of authentication message and check users’s email again, if their 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 is manages gui page, this page allow users add or remove their block, block corresponds port in device, and it have to assign with 1 device.
* ‘/user/gui/blocks’: this router manages block page, it show a record of block which you added, it support search bar to fine block with its name. It receives API which is command from users and save to database.
* ‘/user/gui/blocks/search’: this router will receive name of block you need and then redirect to page of this block, it help full when you have many blocks.
* ‘user/gui/block/:id’: this router manages 1 router which match its id, it will render 1 page to users can control easily.
* displays.js : this file manages routers which support chart and show data from sensors. It includes 3 routers :
* ‘/user/display’: this router will render display page with 2 kind of chart : line chart and bar chart.
* ‘/user/display/getdata’: this router will help display page init data and send lastest data to display page to perform. It just support line chart and bar chart.
* ‘/user/display/datatable’: this router will support for table, it will send data when it receives correct request
* **middlewares**

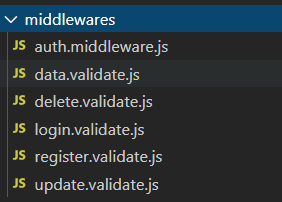
**s**

Figure 2.6

Some router have to do many operation, every action in my sever have to check information of data first, then sever will decide to do it or no.

It contains 6 file for 6 validation purpose :

* auth.middleware.js: this file has 1 middleware, its name is ‘requireAuth”, it will check cookies of users, the cookies is just received when users login, so this module like a filter to prevent stranger and protect data of users.
* data.validate.js: this file has 1 middleware , its name is “checkID” , I want that only devices which have IDs of users and correct form will be accepted and stored
* delete.validate.js: this file has 2 middleware:
* “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 2 middleware likes delete.validate.js file, but it render login page instead of delete page.
* register.validate.js: this file likes login.validate.js file, too. But, it add a function which is used to send email t verify ID for users.
* update.validate.js: this file like login.validate.js file, too. But it will render update page.
* **controller**

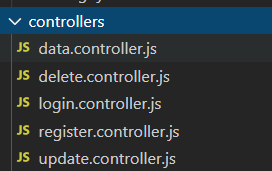
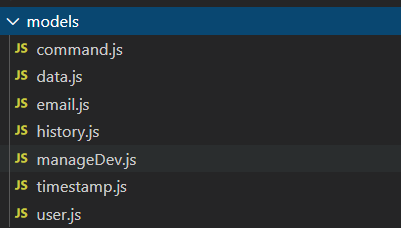
****

Figure 2.7

Controllers inlucde final actions of functions callback in processing request, dividing callback into stages will help to fix easily, increase perform, I create 5 file controller:

* data.controller.js: it just sends one response to devides which post 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 we access login router, if we login 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



To be store in the database, data need to arrange with type JSON(11), and some functions which is reused many time. Form and function in these file just are called and use in other file, do not change their struct.

* 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: data model is form to save from device to 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 send authentication email, verify email. It can be reuse in other feature .

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

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 form 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 have bug, or time data processing of IoT core, time response device.

---------------------------------------------------------------------------------------------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 expanse and useful in the future.

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

var mongoose = require('mongoose');

var userSchema = new mongoose.Schema({

timestamp: Number,

email: String,

password: String,

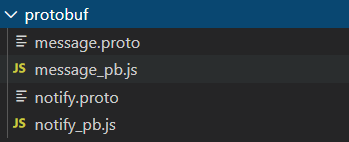
status: Number

});

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

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

* **protobuf**

****

Protobuf is a language-neutral of google , it a good way to make a API like JSON and XML. It will serialize a form data into an array, size of one element in this array is 1 byte.

I use JSON to make a form data which sent from device, but I use protobuf to make a form command which sent from IoT Core. There 2 reasons : C language does not have object concept, so analyzing a JSON string is so hard; the secondly, protobuf is so easy to deploy. I made 2 protobuf form :

* message.proto: This is form to send command to devide, because I control DAC, PWM, Digital and they are assigned to pin on device, I can send value of pin to control it.

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

message Sensor {

string ID=1;

uint32 device=2;

uint32 io =3;

float value=4;

}

message Sensors {

repeated Sensor sensors=1;

}

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

* notify.proto: This form is used to send a message to device. Firstly, I create it to send a message to device and it will display on a screen or may be used to send message between users like Messanger. After thinking, I decided to do not do that, and it may be useful in future.

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

message Notify {

string ID=1;

uint32 device=2;

string header=3;

string content=4;

string comment=5;

}

message Notifies {

repeated Notify notifies=1;

}

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

* Syntax to generate proto file to JavaScript file:

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

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

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

After generating this file, we need core runtime, 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:

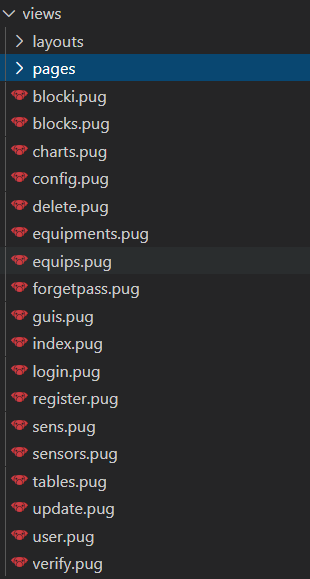
---------------------------------------------------------------------------------------------

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

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

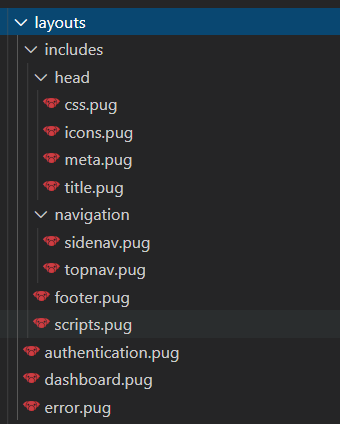
After generating this file, we will have 1 components with source file and header file.

* **views**

****

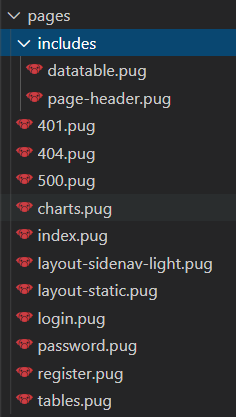
IoT core use a view engine named “pug”. Pug is an HTML preprocessor implemented in Node.js and it help us render page from Pug to HTML. It supports methods, packages to make the pages of users look better. I use 1 Platform pattern, which design on github of David Miller. It is include : layouts and pages folder. My pages I create is based on pattern and layout of that resources.

* layouts:



There are features standards and some forms which are unchange, all of that will be render into layouts. I take it and change some feature which are usefull with my thesis.

* css.pug: it includes sources file CSS, they are pattern which is design to make pages look so great.
* icons.pug: this file add icon of webapp.
* meta.pug: it includes meta tags, kind of tag defines metadata about an HTML document. Metadata is data (information) about data
* title.pug: this file help every page is assign title when sever render.
* sidenav.pug: it support format on the left of dashboard form.
* topnav.pug: it support format on the top of dashboard form.
* footer.pug: it support format on the bottom of pages.
* scripts.pug: it include script tag to add pattern JavaScript files.
* authentication.pug: it is a pattern layout, which is used to make a login page or register page.
* dashboard.pug: it is pattern dashboard, I fix it to render for user page.
* error.pug: it is pattern layout for error page, when users access wrong URL, this layout will be rendered.
* pages:



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

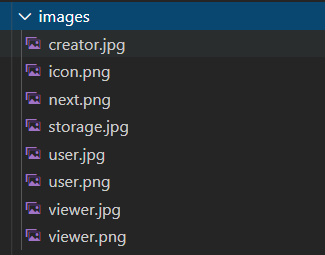
* datatable.pug: this file form of one table in HTML, data table is support bt jQuery library.
* page-header.pug: this file support format on the top of page.
* 401.pug : it is a pattern layout, rendered with 401 error, 401 error occur when request of client has not been applied.
* 404.pug : it is a pattern layout, rendered with 404 error, 404 error occur 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 sever.

Next, I will present my pages I made:

* index.pug : my home is rendered from this file, it is very simple, and It make my 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 an account, they need to go to this page.
* forgetpass.pug : I make file to render forgetpass 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 you email to reset.
* user.pug : this file base on pattern dashboard layout, but I add some feature to make it useful on my 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 your account in this page.
* tables.pug : this file base on pattern tables layout, it help to display data with tables
* charts.pug : this file base on pattern charts layout, I using chart.js package for it.
* config.pug : this file base on pattern login layout, just adding more input tags and more buttons tag.
* sensors.pug : this file is rendered to sensor page, it is base on pattern login layout.
* sens.pug : this file is rendered to sensi page, it is base on pattern login layout.
* equipments.pug : this file is rendered to equipment page, it is base on pattern login layout.
* equips.pug : this file is rendered to equipi page, it is base on pattern login layout.
* guis.pug : this file is rendered to equipi 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 pattern login layout.
* blocki.pug : this file is rendered to page, based on index of block , it is base on pattern login layout.
* **public**

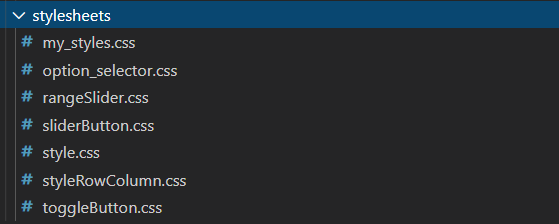
It includes three folder : images, javascript, stylesheets. In order to have a good view, I need CSS files , it will increase UX . These folder represent three parts of one page : content, format and event.

* images



I put my images in here, “user.jpg” is used in the first page when user access sever, “icon.png” is icon on all pages of my webapp. Other pictrue is old, it may be used in the future.

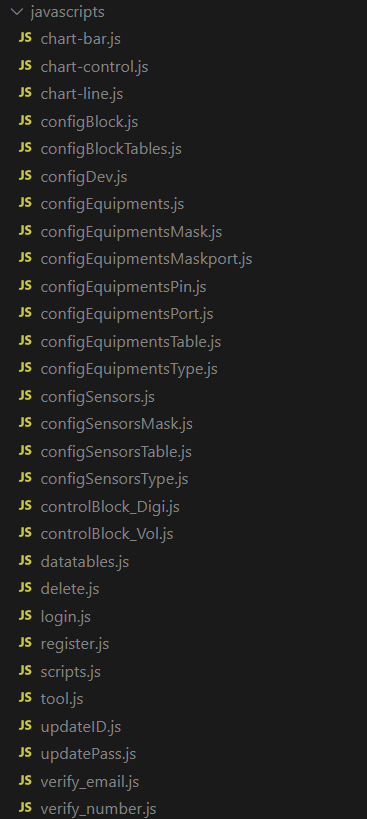
* stylesheets



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

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

* my\_styles.css: this file includes format of pattern layouts I use, 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 format properties to make a slider, which I use on control DAC and PWM of device.
* javascripts



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 and process and display on the top of home page, like a clock.
* scripts.js: this is file of pattern 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 sever 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 input : email, password, confirm password, it will send API of that parameters to sever 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 like login, 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 like login, but it will send package to router ‘/user/delete’, it support
* verify\_email.js: it will send email of users who forget their password to router ‘/forgetpass’, this file support forgetpass page.
* verify\_number.js: it will send email of users and number they receive in email to ‘/forgetpass/verify’, this file support verify page.
* datatables.js: it will send request to get data for tables feature, which pages add this file can get data continuously.
* chart-line.js: it includes function to init line-chart, function to get data continuously and function to clear data when full.
* chart-bar.js: it includes function to init bar-chart, function to get data continuously.
* chart-control.js: when users press “Choose”, 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 sever, and wait response .
* configEquipments.js: it wil read state of switch button on card which allow users to disable or enable equipments, then send to sever.
* configEquipmentsMark.js: it will read names of equipments which you config, then send to sever.
* configEquipmentsMaskport.js: it read name of port which you config, one port represent one “block”, then send to sever.
* configEquipmentsPin.js: it read index number of pin on device which you config, then send to sever.
* configEquipmentsPort.js: it read index number of port which you config, every port has 1 unique number, then send to sever.
* configEquipmentsType.js: it wil read state of switch button on card which allow users to choose type of sensor, then send to sever.
* configEquipmentsTable.js: it read name of device which control equipments, in the search box of equipment page, then pack and send this information to sever to check.
* configSensors.js: it wil read state of switch button on card which allow users to disable or enable sensors, then send to sever.
* configSensorsMask.js: it will read names of sensors which you config, then send to sever.
* configSensorsType.js: it wil read state of switch button on card which allow users to choose type of sensor, then send to sever.
* configSensorsTable.js: it read name of device which control sensors, in the search box of sensor page, then pack and send this information to sever to check.
* configBlock.js: after users fill all parameters for block and press “add”/”remove” button, it will pack data and send to sever, and wait response .
* configBlock\_Digi.js: it wil read state of switch button on card which allow users to turn on or turn off equipments, then send to sever.
* configBlock\_Vol.js: it wil read value of slider on card which allow users to adjust value of DAC or PWM, then send to sever.
* configBlockTables.js: it read name of block in the search box of block page, then pack and send this information to sever to check.

Fuction pattern to send request to sever, 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)

});

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

Fuction pattern to send request to sever, 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, \*/\*',

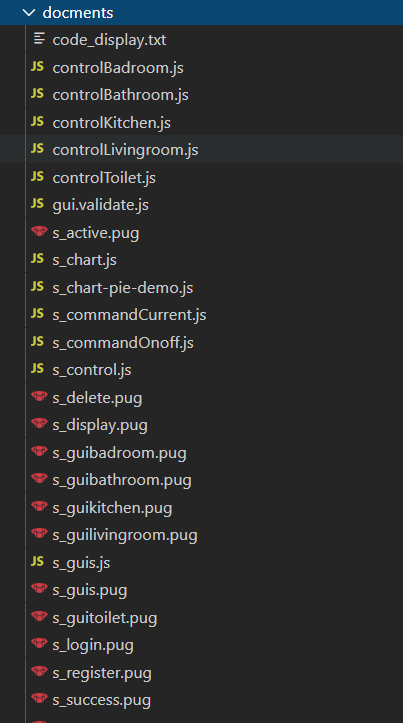
'Content-Type': 'application/json',

id:dev

}

});

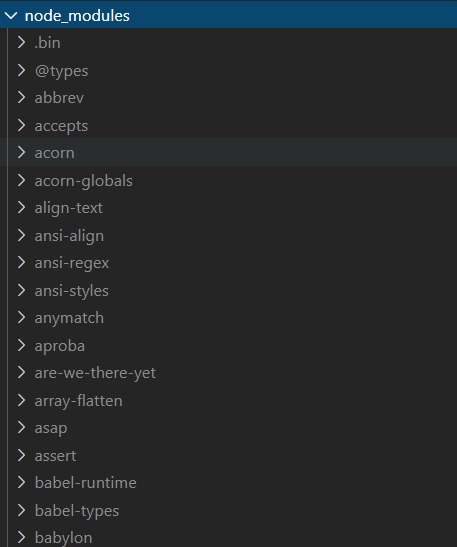
* **documents**

****

It includes old files, when I build the system, some errors occured and some GUI were obsolete . But it is the premise for me to build a good system now.

Some method I can reuse or improve in the future.

* **node\_modules:**



These are the modules of nodejs, based on JavaScript, TypeScript and other languages. It includes package I added in the package.json file, too.

In conclusion, with inheritance ideal from MVC structure, and refer some direction of developer on github, I have complete my IoT core successfully.

# Chapter 3:

# MongoDB

**3.1 Intruduction**

**3.1.1 Database**

Concept of database (Oracle wesite [https://www.oracle.com/database/what-is-database.https://www.oracle.com/database/what-is-database.html#:~:text=Products%20A%E2%80%93Z-,Database,electronically%20in%20a%20computer%20system.&text=The%20data%20can%20then%20be,for%20writing%20and%20querying%20data..](https://www.oracle.com/database/what-is-database.html#:~:text=Products%20A%E2%80%93Z-,Database,electronically%20in%20a%20computer%20system.&text=The%20data%20can%20then%20be,for%20writing%20and%20querying%20data.)

)

“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”.

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

**3.1.2 Databse management system**

<https://www.techopedia.com/definition/24361/database-management-systems-dbms>

“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”.

<https://www.guru99.com/sql-vs-nosql.html>

**3.1.3. 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.”

**3.1.4. NoSQL database**

<https://www.mongodb.com/nosql-explained>

“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.”

**3.2 MongoDB**

<https://en.wikipedia.org/wiki/MongoDB>

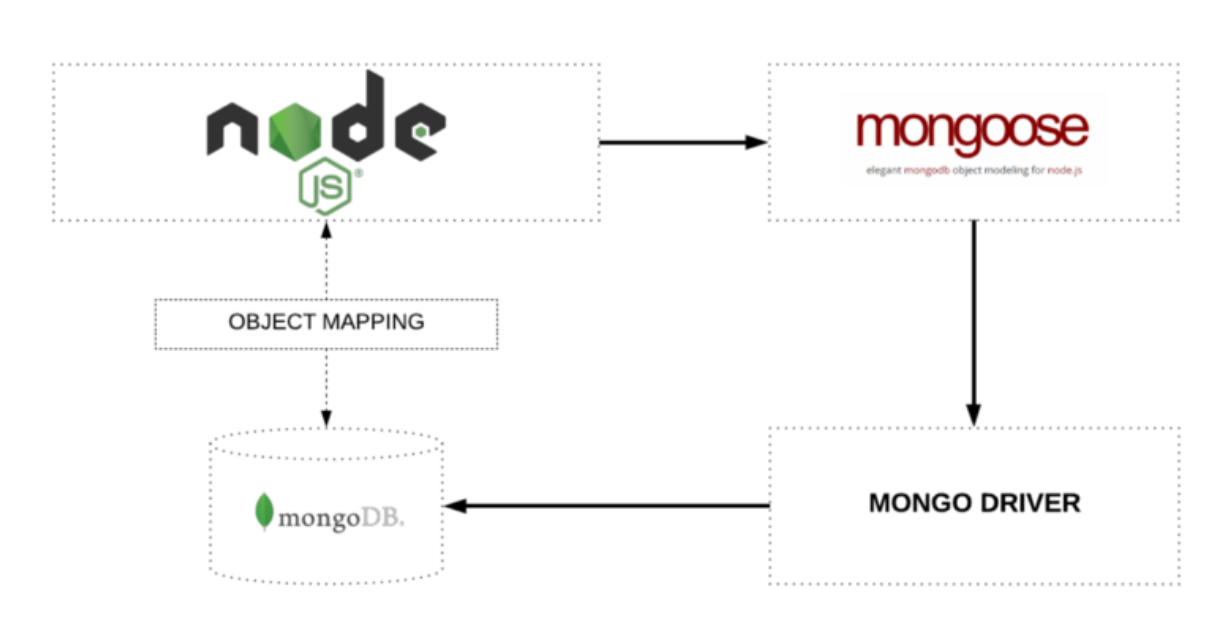
“MongoDB is a [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) [document-oriented database](https://en.wikipedia.org/wiki/Document-oriented_database) program. Classified as a [NoSQL](https://en.wikipedia.org/wiki/NoSQL) database program, MongoDB uses [JSON](https://en.wikipedia.org/wiki/JSON)-like 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”.

MongoDB is supported on many platforms , it includes nodejs, it is easy to use for application like my sever. MongoDB support a global cloud database service, it will storage data free : Up to 5GB storage and Shared RAM.

**3.3 Mongoose**

<https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>

“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”.



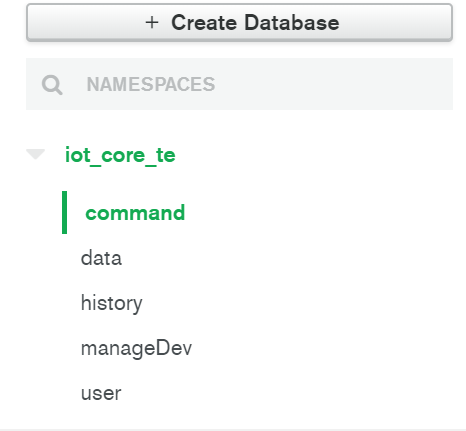
With mongoose, I can make a model for document easily, it supports many function with filters when fine, update, delete, create documents.

**3.4 MongoDB Atlas**

<https://www.mongodb.com/cloud/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”.

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



**Chapter 4:**

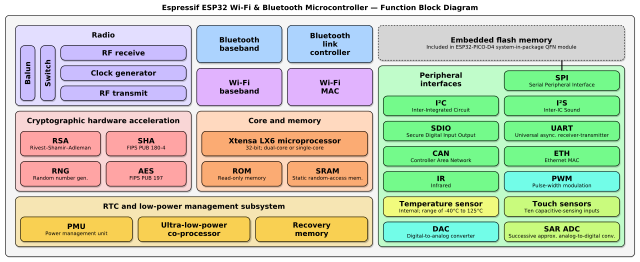
**DEVICE**

**4.1.Introduce about ESP32 Microcontroller**

<https://datasheet.octopart.com/ESP32-D0WDQ6-Espressif-Systems-datasheet-138896080.pdf>

“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”.

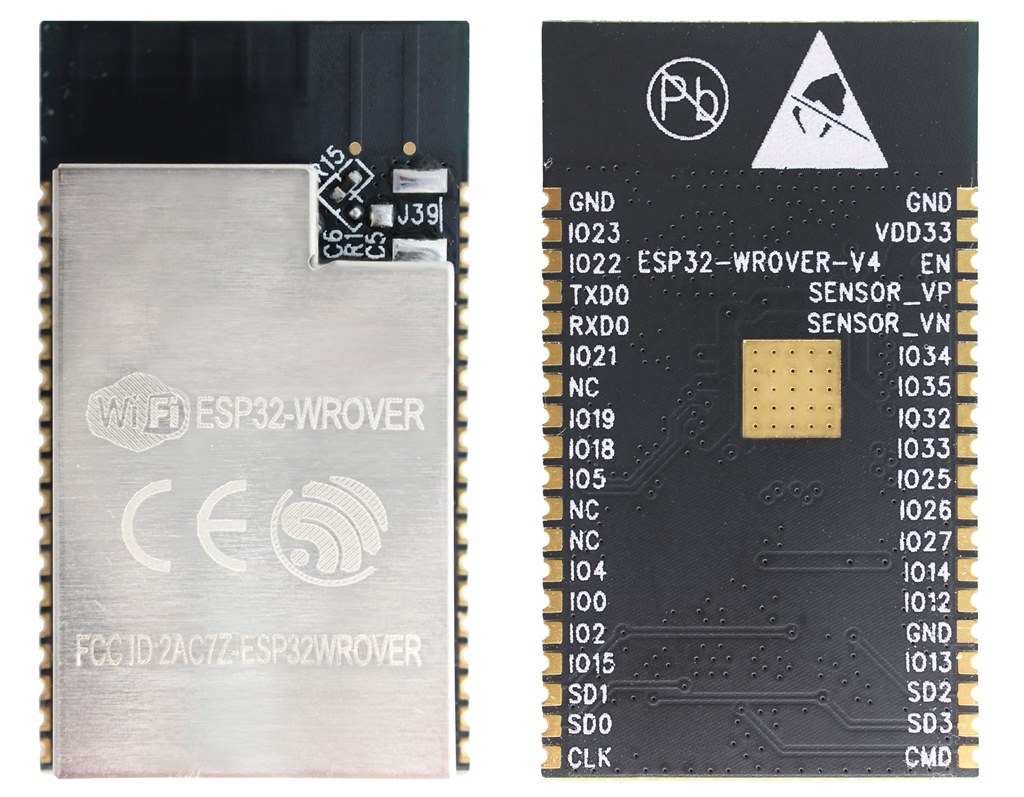
CPU of ESP32 is Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, it operates at 160 or 240 MHz and performing at up to 600 [DMIPS](https://en.wikipedia.org/wiki/Dhrystone), and supports Ultra low power (ULP) co-processor. In addition, it have memory is 512 KiB SRAM.



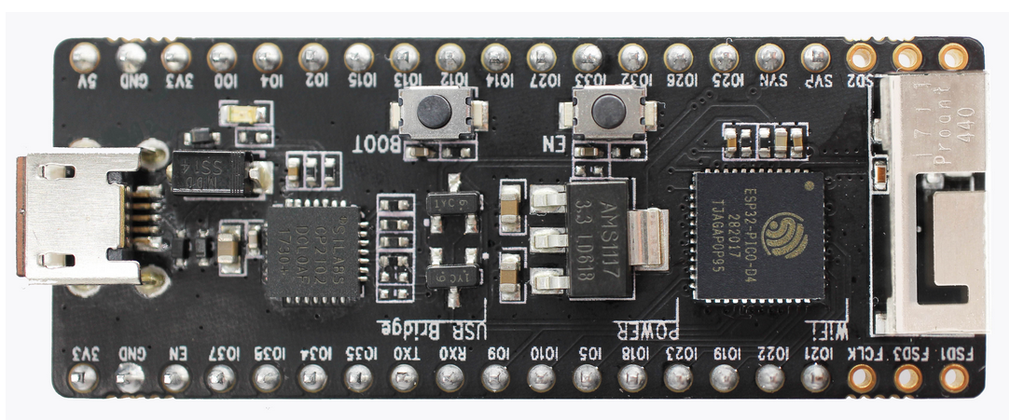
In my thesis, I choose esp32-wrover module and esp32 pico D4 kit for my thesis:

* Esp32 wrover module : it supports PSRAM make it more powerful.

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/modules-and-boards.html>



* Esp32 pico D4 kit : it use ESP-PICO-D4 is a System-in Package(SiP) module base on ESP32



**Proe**

**4.2. Adapter for ESP32-WROVER-V4.**

Because ESP32-WROVER-V4 module needs a adapter to be programed and connect to sensors or equipments. I has make a adapter for it with support from KiCad sortware.

4.1.2.Schelmatic

4.1.2. PCB and 3D images

4.1.3. Production.

**4.2.Introduce ESP-IDF**

It mean Espressif IoT Development Framwork. This framework support many API for peripherals, WI-Fi, Protocols,…

When I use ESP-IDF and Arduino platform, I have some compare : ESP-IDF is more flexible, easier for using and develop library, when esp32 use Wi-Fi, with arduino make devide hot and crash, othewise, ESP-IDF make esp32 operation normally.

In additionally, ESP-IDF integrate FreeRTOS, a real-time operating system, and we will have power from 2 core

**4.3.Feature of device**

The most important feature is HTTP protocol, because sever will response API through HTTP. In the first time, I think esp32 should be a sever, and then I see the power of nodejs , cloud resource, I decided esp32 must be client and it will use API of HTTP Client of ESP-IDF

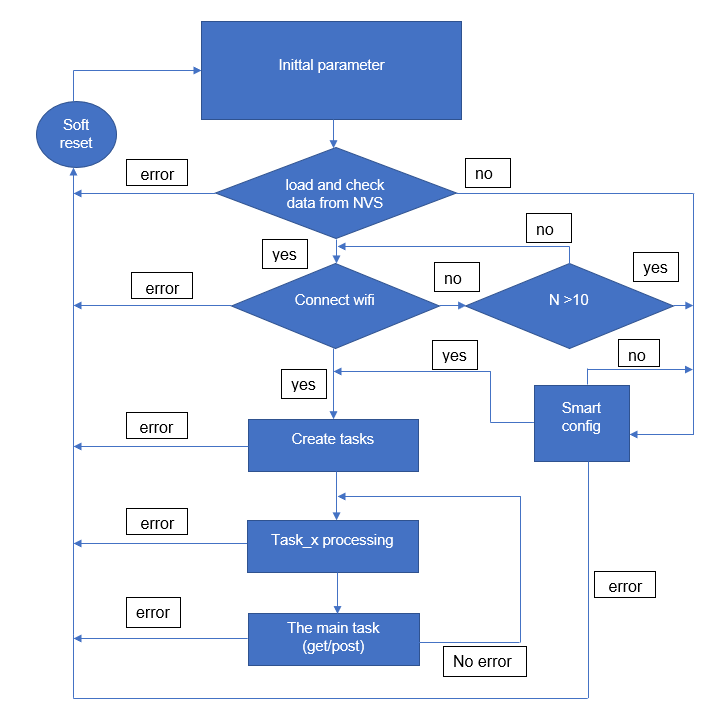
Next is Wi-Fi, I do not want to pay for Ethernet wire , and Wi-Fi a wireless standard very common.

The Thirdly, with the “euipment” I use feature DAC,PWM ,Digital output to control it, with “sensor” I use ADC, I2C, Digital input to get data. Because my IoT core is not involved with device, developer can use other standard communicate like UART,SPI,… to get data or control somethings

**4.4.Algorithm flowchart**

With FreeRTOS , I can distribute feature follow task , esp32 will proccess every task in their period. The task which communicate with sever will have the highest property to ensure it will be process in time.

I give the **Algorithm flowchart below :**

****

If any error happen, I will restart the device, because error can be a crash a missing a data, it may be do harm device, restart wil ensure esp23 initial all parameter.

**Chapter 5:**

**USER**

**5.1 Key-word in my system**

**Key Word I use :**

**Device :** index of esp32 which you choose ( 0 < device < 2^32-1)

**sensor :** which give data of enviroment ( orther microprocessing, sensor DHT21,...)

**euipment :** which you can control througth send command to server ( didital , non-digital)

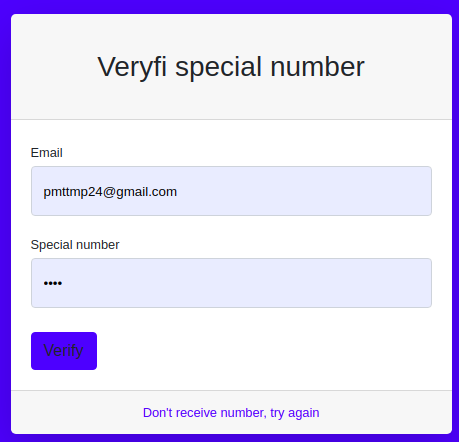
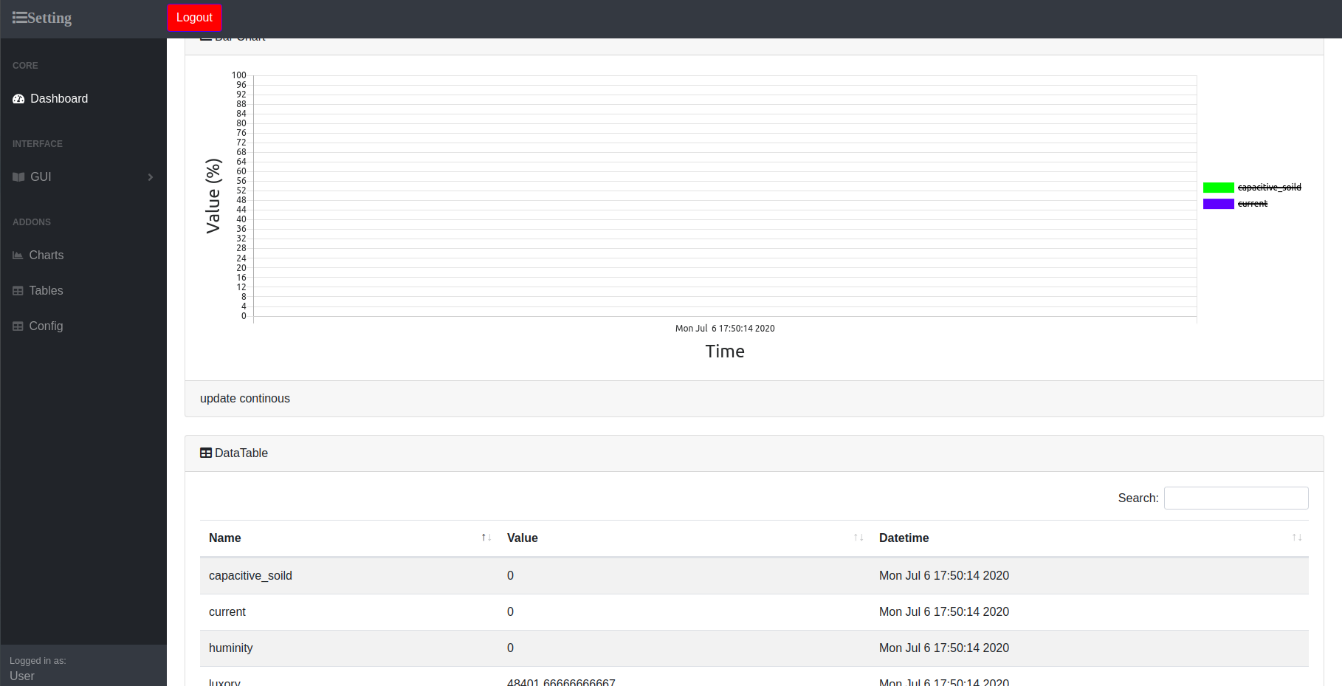
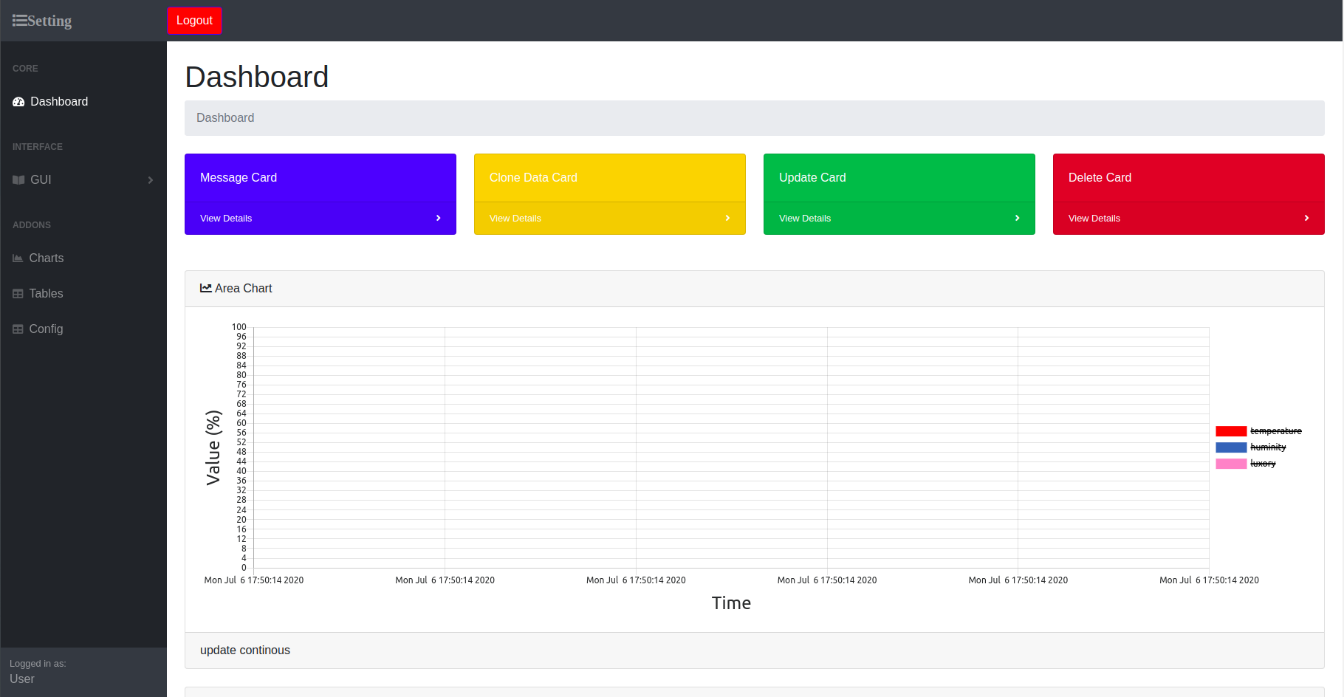
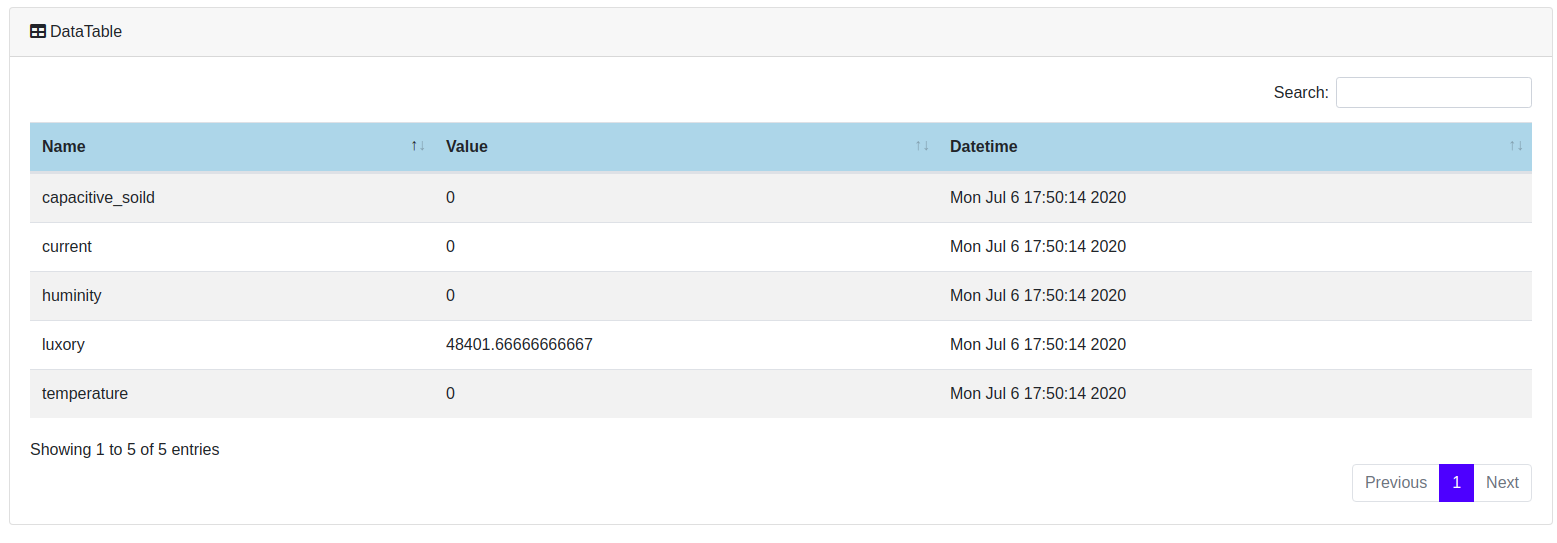
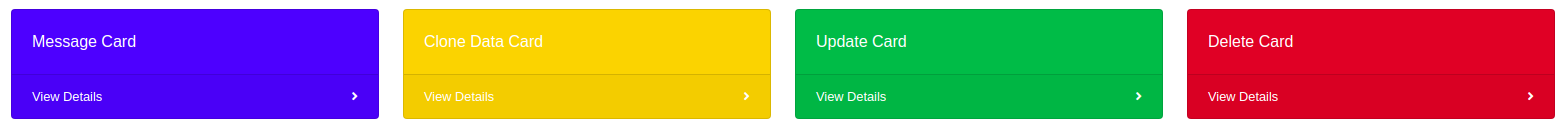
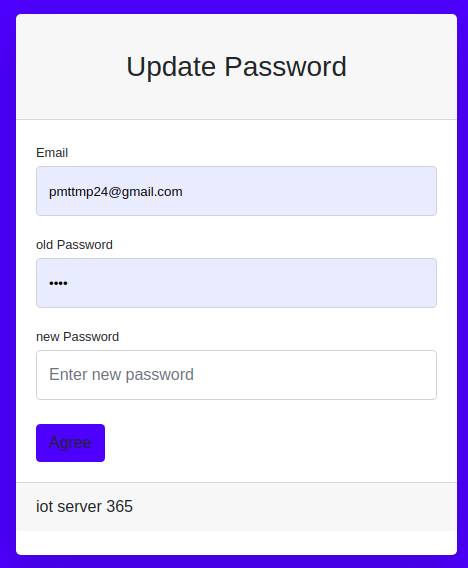
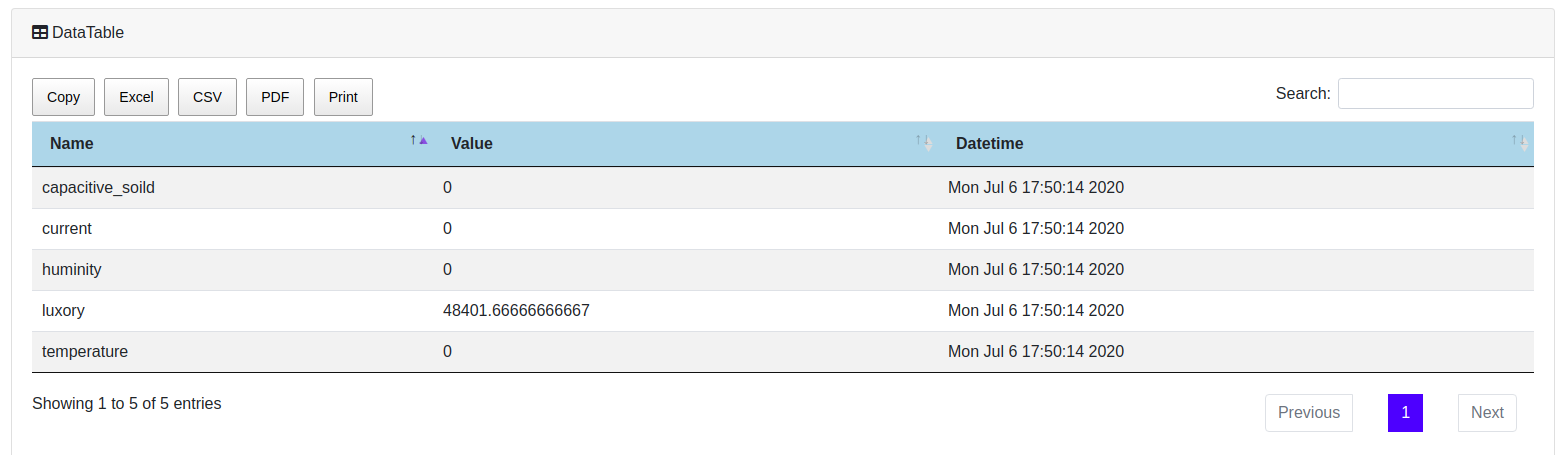
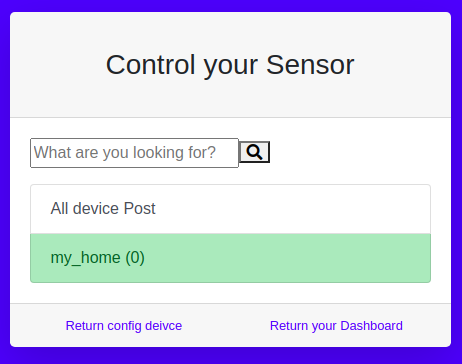
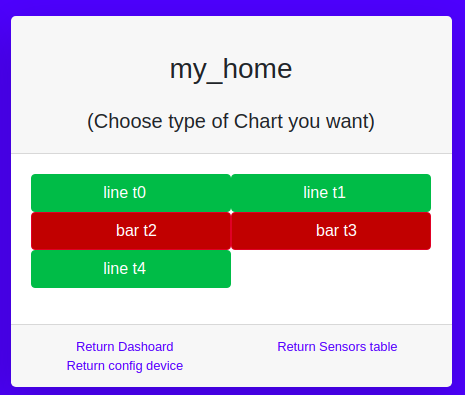
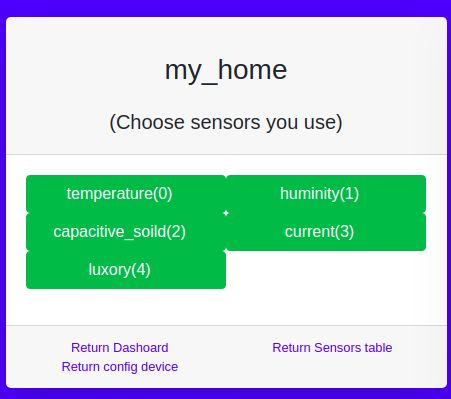
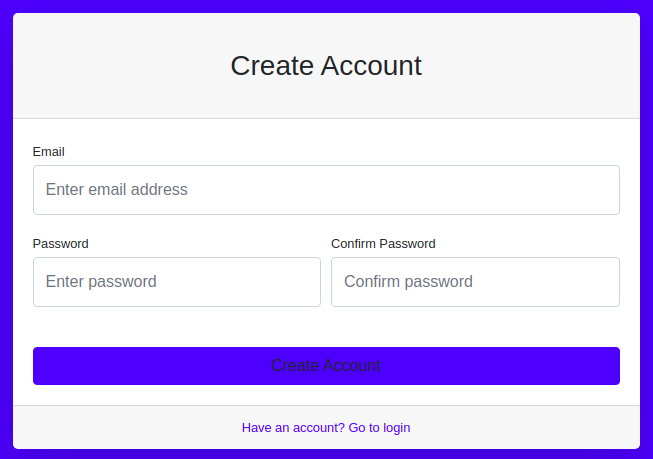
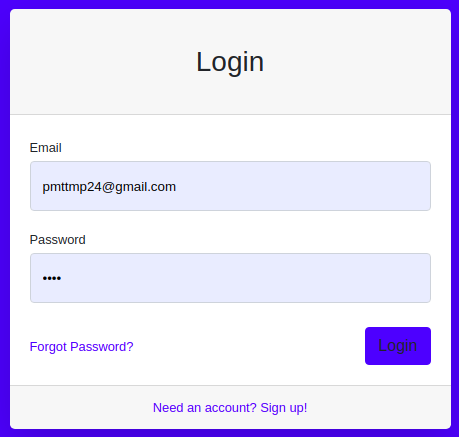
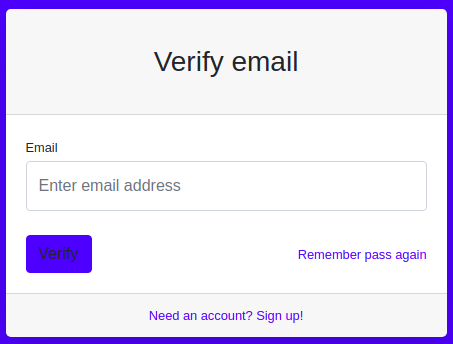
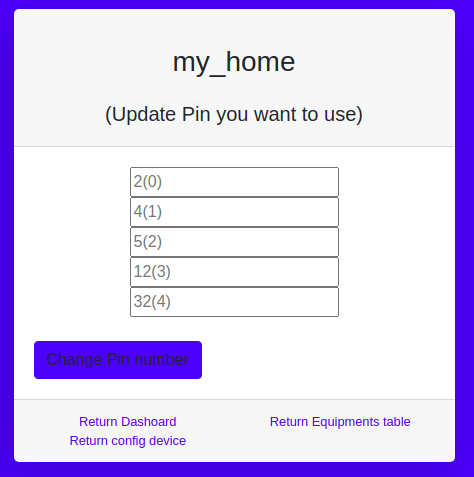
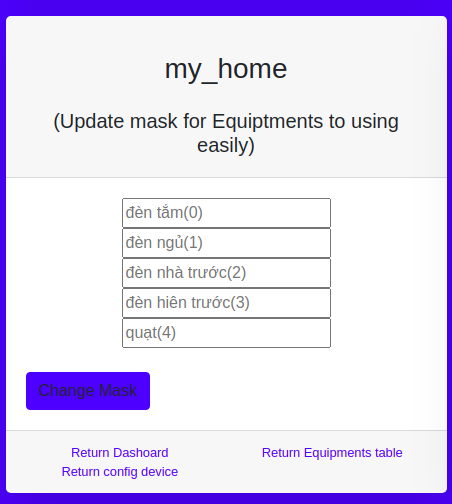
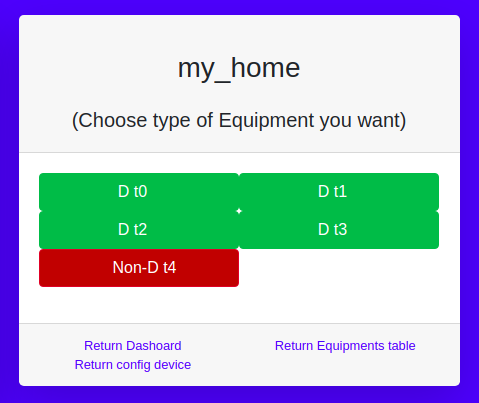
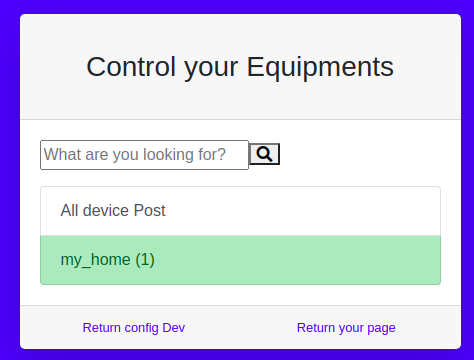
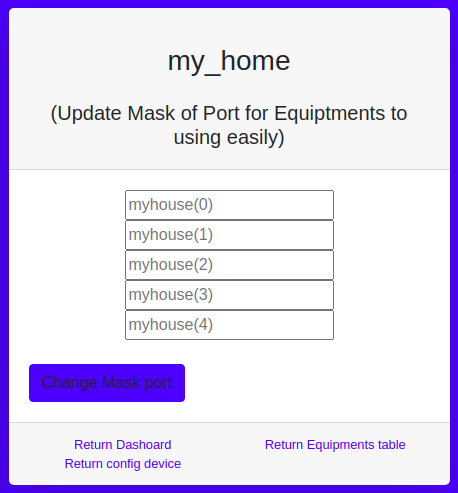
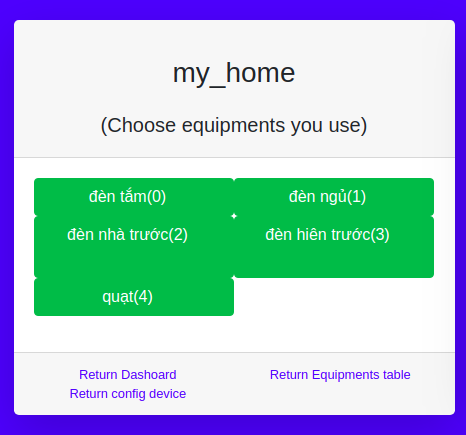
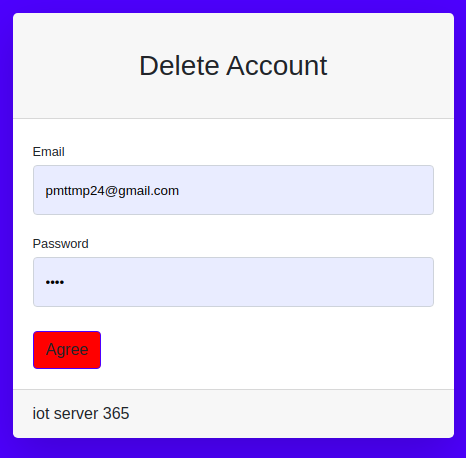
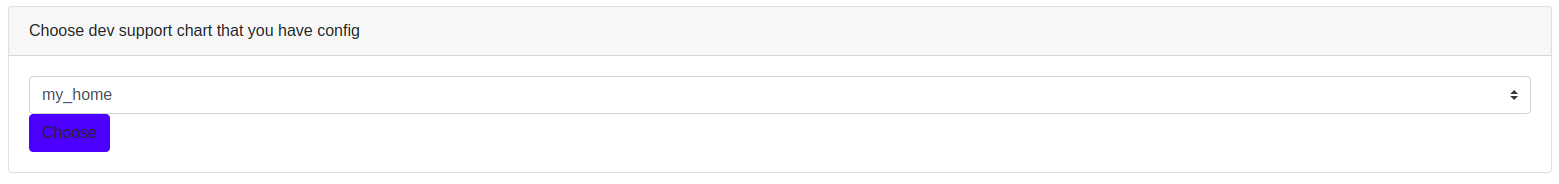
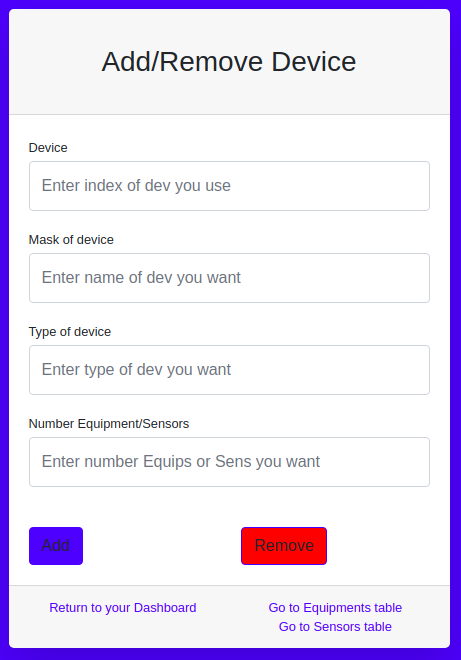
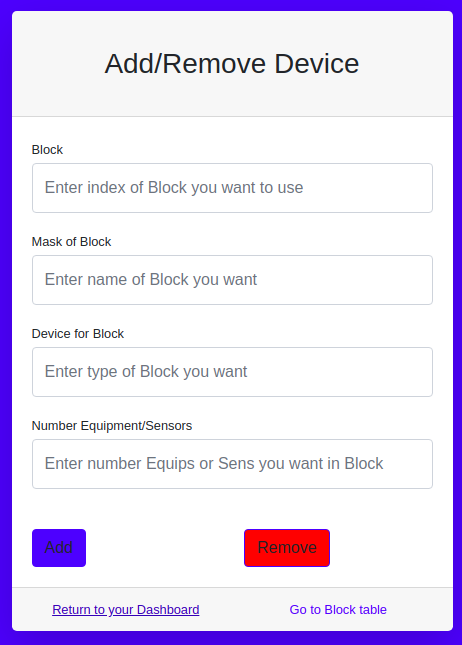
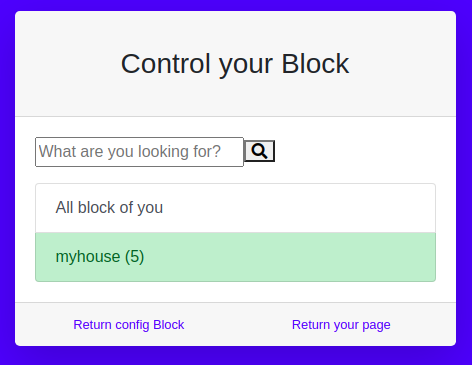
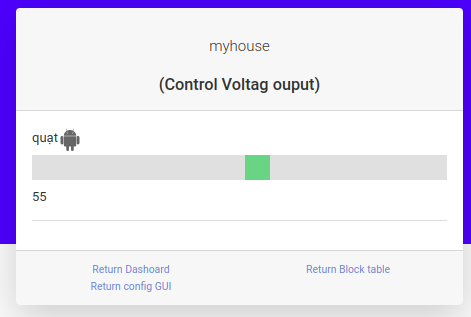
**ID :** Identify of unique User , User can have many device but only 1 ID, remember

**Type Of devide :** I divide into 2 type ( can be expanse in the future ) , basic is : 0 ( to get command and do it ) and 1 ( to send data to server ). It is GET and POST in HTTP

**Device type 0 manage Equipment ( receive command),device type 1 manage Sensor ( read data).**

**ConFig** : it help you announce your device and add it to you profile

**5.1 Pages sever supoort**

****

**Chapter 6:**

**SUMMARY**

Bibliography

1. Nodejs 12.x.x document. Retrieved from [https://nodejs.org/dist/latest- v12.x/docs/api/](https://nodejs.org/dist/latest-v12.x/docs/api/).
2. NPM 6.x.x document. Retrieved from <https://docs.npmjs.com/>.
3. StartBootstrap-sb-admin. Retrieved from github [https://github.com/StartBootstrap/ startbootstrap-sb-admin](https://github.com/StartBootstrap/%20%20startbootstrap-sb-admin).
4. 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).
5. ESP-IDF 4.1. Retrieved from github <https://github.com/espressif/esp-idf> .
6. Mongoose 5.xx document. Retrieved from <https://mongoosejs.com/docs/documents.html>.
7. Mongodb 4.x . Retrieved from <https://docs.mongodb.com/manual/core/document/>.