**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG DỮ LIỆU**

**BÙI THANH HẢI**

**KHÓA LUẬN TỐT NGHIỆP**

**PHÁT TRIỂN GIẢI PHÁP GIÁM SÁT**

**NHÀ THÔNG MINH THỜI GIAN THỰC DỰA TRÊN**

**NỀN TẢNG ĐIỆN TOÁN ĐÁM MÂY**

**DEVELOPMENT OF A SURVEILLANCE SYSTEM FOR SMART HOUSES INTEGRATED OVER CLOUD PLATFORM**

**KỸ SƯ NGÀNH TRUYỀN THÔNG VÀ MẠNG MÁY TÍNH**

**TP. HỒ CHÍ MINH, 2018**

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG DỮ LIỆU**

**BÙI THANH HẢI – 14520236**

**KHÓA LUẬN TỐT NGHIỆP**

**PHÁT TRIỂN GIẢI PHÁP GIÁM SÁT**

**NHÀ THÔNG MINH THỜI GIAN THỰC DỰA TRÊN**

**NỀN TẢNG ĐIỆN TOÁN ĐÁM MÂY**

**DEVELOPMENT OF A SURVEILLANCE SYSTEM FOR SMART HOUSES INTEGRATED OVER CLOUD PLATFORM**

**KỸ SƯ NGÀNH TRUYỀN THÔNG VÀ MẠNG MÁY TÍNH**

**GIẢNG VIÊN HƯỚNG DẪN**

**PGS.TS LÊ TRUNG QUÂN**

**TP. HỒ CHÍ MINH, 2018**

DANH SÁCH HỘI ĐỒNG BẢO VỆ KHÓA LUẬN

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số …………………… ngày ………………….. của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

* 1. …………………………………………. – Chủ tịch.
  2. …………………………………………. – Thư ký.
  3. …………………………………………. – Ủy viên.
  4. …………………………………………. – Ủy viên.

LỜI CẢM ƠN

Trong quá trình làm khóa luận em đã nhận được rất nhiều sự giúp đỡ để hoàn thành khóa luận.

Trước tiên em xin gửi lời cảm ơn đến thầy hướng dẫn PGS.TS Lê Trung Quân đã tin tưởng, chấp nhận giao đề tài, truyền đạt kiến thức, kinh nghiệm, hướng dẫn và chỉ bảo tận tình để em có thể hoàn thành khóa luận một cách tốt nhất.

Xin gửi lời cảm ơn chân thành đến quý thầy cô Khoa Mạng máy tính và Truyền thông, Trường đại học Công nghệ Thông tin đã truyền đạt nhiều kiến thức quý báu, những kiến thức này đã phần nào giúp em hoàn thành khóa luận.

Xin gửi lời cảm ơn đến anh Nhân, anh Đặng đã hỗ trợ thiết bị và tài nguyên cho em.

Sau cùng xin gửi lời cảm ơn đến gia đình, bạn bè… đã luôn động viên, giúp đỡ tôi trong quá trình làm luận văn.

Một lần nữa, xin chân thành cảm ơn!

**Bùi Thanh Hải**

MỤC LỤC

[Chương 1. GIỚI THIỆU 2](#_Toc532715314)

[1.1. Tổng quan 2](#_Toc532715315)

[1.2. Vấn đề đặt ra 2](#_Toc532715316)

[1.3. Phương pháp giải quyết 3](#_Toc532715317)

[1.4. Đối tượng và phạm vi nghiên cứu 3](#_Toc532715318)

[1.4.1. Đối tượng nghiên cứu 3](#_Toc532715319)

[1.4.2. Phạm vi nghiên cứu 4](#_Toc532715320)

[1.5. Cấu trúc khóa luận 4](#_Toc532715321)

[Chương 2. MỘT SỐ CÔNG NGHỆ LIÊN QUAN 5](#_Toc532715322)

[2.1. Công nghệ IoT 5](#_Toc532715323)

[2.2. Điện toán đám mây 6](#_Toc532715324)

[2.3. Một số thư viện, Frameworks liên quan 7](#_Toc532715325)

[2.3.1. Xây dựng Backend – Server API 7](#_Toc532715326)

[2.3.1.1. ASP.NET 7](#_Toc532715327)

[2.3.1.2. NodeJS 8](#_Toc532715328)

[2.3.2. Xây dựng Frontend – Giao diện Web 9](#_Toc532715329)

[2.3.2.1. ReactJS 9](#_Toc532715330)

[2.3.2.2. Angular 10](#_Toc532715331)

[2.4. Docker 11](#_Toc532715332)

[Chương 3. HƯỚNG TIẾP CẬN 13](#_Toc532715333)

[3.1. Mô hình tổng quan 13](#_Toc532715334)

[3.1.1. Khối xử lý trung tâm 13](#_Toc532715335)

[3.1.2. Khối thu thập dữ liệu và xử lý tín hiệu 14](#_Toc532715336)

[3.1.3. Khối giám sát và điều khiển 14](#_Toc532715337)

[3.1.4. Mô hình triển khai trên Cloud 15](#_Toc532715338)

[3.2. Một số kịch bản 15](#_Toc532715339)

[3.2.1. Quản lý xuất nhập 15](#_Toc532715340)

[3.2.2. Dùng cảm biến ánh sáng để điều khiển đèn 16](#_Toc532715341)

[3.2.3. Báo động khi phát hiện lửa hoặc nhiệt độ cao 17](#_Toc532715342)

[3.3. Hiện thực giải thuật 18](#_Toc532715343)

[3.3.1.1. Phần mềm và ngôn ngữ sử dụng 18](#_Toc532715344)

[3.3.1.2. Phần cứng sử dụng 19](#_Toc532715345)

[3.3.1.3. Các thiết bị ngoại vi 20](#_Toc532715346)

[Chương 4. XÂY DỰNG HỆ THỐNG GỬI NHẬN DỮ LIỆU THÔNG QUA API BẰNG ASP.NET CORE 22](#_Toc532715347)

[4.1. Tổng quan 22](#_Toc532715348)

[4.2. Mục tiêu 22](#_Toc532715349)

[4.3. Các bước thực hiện 22](#_Toc532715350)

[4.3.1. Bước 1: Tạo một project mới bằng Visual Studio 22](#_Toc532715351)

[4.3.2. Bước 2: Cấu trúc thư mục 24](#_Toc532715352)

[4.3.3. Bước 3: Install các package cần thiết 24](#_Toc532715353)

[4.3.4. Bước 4: Tạo Model để định dạng dữ liệu và các phương thức truyền nhận 25](#_Toc532715354)

[4.3.4.1. PinModel 25](#_Toc532715355)

[4.3.4.2. UserModel 29](#_Toc532715356)

[4.3.4.3. DHTModel 35](#_Toc532715357)

[4.3.4.4. KitKey 39](#_Toc532715358)

[4.3.4.5. UserRole 39](#_Toc532715359)

[4.3.4.6. Gender 39](#_Toc532715360)

[4.3.5. Bước 5: Tạo DBContext để tương tác với CSDL 40](#_Toc532715361)

[4.3.6. Bước 6: Chỉnh sửa file “Startup.cs” 40](#_Toc532715362)

[4.3.7. Bước 7: Tạo API 41](#_Toc532715363)

[4.3.7.1. PinController 42](#_Toc532715364)

[4.3.7.2. UserController 44](#_Toc532715365)

[4.3.7.3. DHTController 48](#_Toc532715366)

[4.3.8. Bước 8: Tạo dữ liệu mặc định 50](#_Toc532715367)

[4.3.9. Bước 9: Migration 52](#_Toc532715368)

[4.3.10. Bước 10: Chạy chương trình 53](#_Toc532715369)

[Chương 5. XÂY DỰNG HỆ THỐNG THU THẬP DỮ LIỆU 54](#_Toc532715370)

[5.1. Tổng quan 54](#_Toc532715371)

[5.2. Xây dựng hệ thống 54](#_Toc532715372)

[5.2.1. Arduino ESP8266 xử lý tín hiệu RFID 54](#_Toc532715373)

[5.2.2. Arduino ESP8266 gửi dữ liệu 55](#_Toc532715374)

[5.2.3. Arduino ESP8266 nhận dữ liệu 56](#_Toc532715375)

[Chương 6. XÂY DỰNG HỆ THỐNG GIÁM SÁT VÀ ĐIỀU KHIỂN 57](#_Toc532715376)

[6.1. Tổng quan 57](#_Toc532715377)

[6.2. Mục tiêu 57](#_Toc532715378)

[6.3. Một số chức năng 57](#_Toc532715379)

[6.3.1. Login 57](#_Toc532715380)

[6.3.2. Profile 58](#_Toc532715381)

[6.3.3. Quản lí người dùng 59](#_Toc532715382)

[6.3.4. Giám sát 59](#_Toc532715383)

[6.3.5. Điều khiển 60](#_Toc532715384)

[Chương 7. TRIỂN KHAI HỆ THỐNG LÊN OPENSTACK 61](#_Toc532715385)

[7.1. Tổng quan 61](#_Toc532715386)

[7.2. Mục tiêu 61](#_Toc532715387)

[7.3. Các bước thực hiện 61](#_Toc532715388)

[7.3.1. Tạo máy ảo trên OpenStack 61](#_Toc532715389)

[7.3.2. Truy cập vào địa chỉ máy ảo bằng Putty 62](#_Toc532715390)

[7.3.3. Tiến hành cài đặt Docker trên máy ảo 62](#_Toc532715391)

[7.3.3.1. Cài đặt môi trường 62](#_Toc532715392)

[7.3.3.2. Cài đặt Docker CE 63](#_Toc532715393)

[7.3.3.3. Cài đặt Docker Compose 64](#_Toc532715394)

[7.3.3.4. Đăng nhập vào Docker 64](#_Toc532715395)

[7.3.4. Tạo Repository trên Docker Hub 65](#_Toc532715396)

[7.3.5. Thêm Docker vào FrontEnd 65](#_Toc532715397)

[7.3.5.1. File .dockerignore 65](#_Toc532715398)

[7.3.5.2. File DockerFile 66](#_Toc532715399)

[7.3.5.3. File docker-compose.yml 66](#_Toc532715400)

[7.3.5.4. Tạo và push Docker image 66](#_Toc532715401)

[7.3.6. Thêm Docker vào ASP.NET Core 70](#_Toc532715402)

[7.3.6.1. File .dockerignore 70](#_Toc532715403)

[7.3.6.2. File Dockerfile 70](#_Toc532715404)

[7.3.6.3. File docker-compose.yml 71](#_Toc532715405)

[7.3.6.4. Tạo và push Docker image 71](#_Toc532715406)

[Chương 8. KẾT QUẢ THỰC HIỆN 74](#_Toc532715407)

[8.1. Mức độ delay 74](#_Toc532715408)

[8.2. Đánh giá kết quả 75](#_Toc532715409)

[Chương 9. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 76](#_Toc532715410)

[9.1. Kết quả đạt được 76](#_Toc532715411)

[9.2. Hạn chế 76](#_Toc532715412)

[9.3. Hướng phát triển 76](#_Toc532715413)

DANH MỤC HÌNH VẼ

[Hình 2.1: Công nghệ IoTs 5](#_Toc532658838)

[Hình 2.2: Điện toán đám mây 6](#_Toc532658839)

[Hình 2.3: Thư viện ReactJS 9](#_Toc532658840)

[Hình 2.4: Angular Framework 10](#_Toc532658841)

[Hình 2.5: Docker 11](#_Toc532658842)

[Hình 3.1: Mô hình tổng quan 13](#_Toc532658843)

[Hình 3.2: Khối xử lý trung tâm 13](#_Toc532658844)

[Hình 3.3: Khối thu thập dữ liệu và xử lý tín hiệu 14](#_Toc532658845)

[Hình 3.4: Mô hình triển khai ứng dụng lên Cloud 15](#_Toc532658846)

[Hình 3.5: Giải thuật quản lý xuất nhập 16](#_Toc532658847)

[Hình 3.6: Giải thuật dùng cảm biến ánh sáng điều khiển đèn 17](#_Toc532658848)

[Hình 3.7: Giải thuật phát hiện lửa và nhiệt độ cao 18](#_Toc532658849)

[Hình 3.8: Arduino IDE 18](#_Toc532658850)

[Hình 3.9: Arduino ESP8266 Wifi Uno 19](#_Toc532658851)

[Hình 3.10: Raspberry Pi 20](#_Toc532658852)

[Hình 4.1: Tạo một project (1) 23](#_Toc532658853)

[Hình 4.2: Tạo một project (2) 23](#_Toc532658854)

[Hình 4.3: Cấu trúc thư mục của Project 24](#_Toc532658855)

[Hình 4.4: Cài đặt các packages cần thiết 25](#_Toc532658856)

[Hình 4.5: Cấu trúc thư mục của PinModel 25](#_Toc532658857)

[Hình 4.6: Định nghĩa thuộc tính của PinModel 26](#_Toc532658858)

[Hình 4.7: Định nghĩa phương thức của PinModel 26](#_Toc532658859)

[Hình 4.8: Triển khai phương thức của PinModel (1) 27](#_Toc532658860)

[Hình 4.9: Triển khai phương thức của PinModel (2) 28](#_Toc532658861)

[Hình 4.10: Triển khai phương thức của PinModel (3) 29](#_Toc532658862)

[Hình 4.11: Cấu trúc thư mục của UserModel 30](#_Toc532658863)

[Hình 4.12: Định nghĩa thuộc tính của UserModel 30](#_Toc532658864)

[Hình 4.13: Định nghĩa phương thức của UserModel 31](#_Toc532658865)

[Hình 4.14: Triển khai phương thức của UserModel (1) 31](#_Toc532658866)

[Hình 4.15: Triển khai phương thức của UserModel (2) 32](#_Toc532658867)

[Hình 4.16: Triển khai phương thức của UserModel (3) 33](#_Toc532658868)

[Hình 4.17: Triển khai phương thức của UserModel (4) 33](#_Toc532658869)

[Hình 4.18: Triển khai phương thức của UserModel (5) 34](#_Toc532658870)

[Hình 4.19: Triển khai phương thức của UserModel (6) 34](#_Toc532658871)

[Hình 4.20: Triển khai phương thức của UserModel (7) 35](#_Toc532658872)

[Hình 4.21: Cấu trúc thư mục của DHTModel 35](#_Toc532658873)

[Hình 4.22: Định nghĩa thuộc tính của DHTModel 36](#_Toc532658874)

[Hình 4.23: Định nghĩa phương thức của DHTModel 36](#_Toc532658875)

[Hình 4.24: Triển khai phương thức của DHTModel (1) 37](#_Toc532658876)

[Hình 4.25: Triển khai phương thức của DHTModel (2) 38](#_Toc532658877)

[Hình 4.26: Triển khai phương thức của DHTModel (3) 38](#_Toc532658878)

[Hình 4.27: Nội dung file “KitKey.cs” 39](#_Toc532658879)

[Hình 4.28: Nội dung file “UserRole.cs” 39](#_Toc532658880)

[Hình 4.29: Nội dung file “Gender.cs” 39](#_Toc532658881)

[Hình 4.30: Nội dung file “ThesisAPIContext.cs” 40](#_Toc532658882)

[Hình 4.31: Thiết lập kết nối với SQLite 40](#_Toc532658883)

[Hình 4.32: Thiết lập Cors 41](#_Toc532658884)

[Hình 4.33: Tạo các Controller 41](#_Toc532658885)

[Hình 4.34: Nội dung file “PinController.cs” (1) 42](#_Toc532658886)

[Hình 4.35: Nội dung file “PinController.cs” (2) 42](#_Toc532658887)

[Hình 4.36: Nội dung file “PinController.cs” (3) 43](#_Toc532658888)

[Hình 4.37: Nội dung file “PinController.cs” (4) 43](#_Toc532658889)

[Hình 4.38: Nội dung file “UserController.cs” (1) 44](#_Toc532658890)

[Hình 4.39: Nội dung file “UserController.cs” (2) 45](#_Toc532658891)

[Hình 4.40: Nội dung file “UserController.cs” (3) 45](#_Toc532658892)

[Hình 4.41: Nội dung file “UserController.cs” (4) 46](#_Toc532658893)

[Hình 4.42: Nội dung file “UserController.cs” (5) 46](#_Toc532658894)

[Hình 4.43: Nội dung file “UserController.cs” (6) 47](#_Toc532658895)

[Hình 4.44: Nội dung file “UserController.cs” (7) 47](#_Toc532658896)

[Hình 4.45: Nội dung file “UserController.cs” (8) 48](#_Toc532658897)

[Hình 4.46: Nội dung file “DHTController.cs” (1) 48](#_Toc532658898)

[Hình 4.47: Nội dung file “DHTController.cs” (2) 49](#_Toc532658899)

[Hình 4.48: Nội dung file “DHTController.cs” (3) 49](#_Toc532658900)

[Hình 4.49: Nội dung file “DHTController.cs” (4) 50](#_Toc532658901)

[Hình 4.50: Tạo SeedData 50](#_Toc532658902)

[Hình 4.51: Nội dung file “SeedData.cs” (1) 51](#_Toc532658903)

[Hình 4.52: Nội dung file “SeedData.cs” (2) 51](#_Toc532658904)

[Hình 4.53: Khai báo SeedData 52](#_Toc532658905)

[Hình 4.54: Chạy Migration 53](#_Toc532658906)

[Hình 4.55: Kết quả chạy chương trình 53](#_Toc532658907)

[Hình 5.3: Hệ thống xử lý tín hiệu RFID 54](#_Toc532658908)

[Hình 5.4: Arduino ESP8266 gửi dữ liệu 55](#_Toc532658909)

[Hình 5.5: Arduino ESP8266 nhận dữ liệu 56](#_Toc532658910)

[Hình 6.1: Chức năng Login 57](#_Toc532658911)

[Hình 6.2: Chức năng xem và chỉnh sửa thông tin người dùng 58](#_Toc532658912)

[Hình 6.3: Chức năng quản lí người dùng 59](#_Toc532658913)

[Hình 6.4: Chức năng giám sát 59](#_Toc532658914)

[Hình 6.5: Chức năng điều khiển 60](#_Toc532658915)

[Hình 7.1: Tạo máy ảo trên OpenStack 61](#_Toc532658916)

[Hình 7.2: Truy cập máy ảo bằng Putty 62](#_Toc532658917)

[Hình 7.3: Thiết lập môi trường cho Docker 62](#_Toc532658918)

[Hình 7.4: Cài đặt Docker CE 63](#_Toc532658919)

[Hình 7.5: Kiểm tra cài đặt Docker 63](#_Toc532658920)

[Hình 7.6: Cài đặt Docker Compose 64](#_Toc532658921)

[Hình 7.7: Kiểm tra cài đặt Docker Compose 64](#_Toc532658922)

[Hình 7.8: Đăng nhập tài khoản Docker 64](#_Toc532658923)

[Hình 7.9: Tạo repository trên Docker Hub 65](#_Toc532658924)

[Hình 7.10: Nội dung file .dockerignore 65](#_Toc532658925)

[Hình 7.11: Nội dung file DockerFile 66](#_Toc532658926)

[Hình 7.12: Nội dung file docker-compose.yml 66](#_Toc532658927)

[Hình 7.13: Chạy lệnh docker-compose up 67](#_Toc532658928)

[Hình 7.14: Kết quả chạy docker-compose 67](#_Toc532658929)

[Hình 7.15: Kiểm tra kết quả trên localhost 68](#_Toc532658930)

[Hình 7.16: Lệnh push Docker image lên Docker Hub 68](#_Toc532658931)

[Hình 7.17: Nội dụng file docker-compose.yml trên máy ảo 69](#_Toc532658932)

[Hình 7.18: Kết quả chạy Docker image web trên OpenStack 69](#_Toc532658933)

[Hình 7.19: Nội dung file .dockerignore 70](#_Toc532658934)

[Hình 7.20: Nội dung file Dockerfile 70](#_Toc532658935)

[Hình 7.21: Nội dung file docker-compose.yml 71](#_Toc532658936)

[Hình 7.22: Chạy lệnh docker-compose up 71](#_Toc532658937)

[Hình 7.23: Kết quả sau khi chạy docker-compose up 71](#_Toc532658938)

[Hình 7.24: Kiểm tra kết quả trên localhost 72](#_Toc532658939)

[Hình 7.25: Lệnh push Docker image lên Docker Hub 72](#_Toc532658940)

[Hình 7.26: Nội dung file docker-compose.yml trên máy ảo 73](#_Toc532658941)

[Hình 7.27: Kết quả chạy Docker image web trên OpenStack 73](#_Toc532658942)

DANH MỤC BẢNG

[Bảng 2.1: So sánh ASP.NET Framework và ASP.NET Core 8](#_Toc532658831)

[Bảng 8.1: Bảng mô tả thời gian phản hồi trung bình của thiết bị 74](#_Toc532658832)

DANH MỤC TỪ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **Từ viết tắt** | **Tên đầy đủ** | **Nghĩa/Chức năng** |
| API | Application Programming Interface | Giao diện lập trình ứng dụng |
| CSDL | Cơ sở dữ liệu | Cơ sở dữ liệu |
| CLR | Common Language Runtime | là thành phần kết nối giữa các phần khác trong .NET Framework với hệ điều hành |
| IDE | Integrated Developement Environment | Môi trường phát triển tích hợp |
| IoT | Internet of Things | Internet vạn vật hoặc mạng lưới các thiết bị kết nối Internet |

TÓM TẮT KHÓA LUẬN

Mục tiêu của khóa luận là xây dựng được một hệ thống IoT hoàn chỉnh. Bao gồm xây dựng Backend, Frontend, kết nối database, kết nối các tiết bị IoT,… và cuối cùng là triển khai hệ thống lên Cloud mà không phụ thuộc vào các Web Server đã xây dựng sẵn như Node-RED, arest.io,… Điều này làm cho việc mở rộng, thêm tính năng, thay đổi giao diện, thêm thiết bị, triển khai ứng dụng,… trở nên dễ kiểm soát hơn.

Đồng thời mục tiêu khác của khóa luận là giúp cho những sinh viên mới tìm hiểu về công nghệ IoT có một cái nhìn tổng quan hơn, có thể hình dung được quá trình hoạt động của một mô hình IoT đơn giản từ đó xây dựng được những mô hình lớn hơn, sáng tạo hơn.

# GIỚI THIỆU

## Tổng quan

Cuộc cách mạng công nghiệp 4.0 nổi lên như một làn sóng mới trong thời đại công nghiệp hóa, hiện đại hóa đất nước. Trong cuộc cách mạng này nổi lên nhiều cụm từ như : IoT (Internet of things – Internet vạn vật), AI (Artifical Intelligence – trí tuệ nhân tạo), Cloud Computing (điện toán đám mây), AR/VR (Thực tế ảo), Big Data (khai thác dữ liệu lớn)… Internet of Things là một trong số đó.

Internet of Things là một cụm từ không còn xa lạ đối với nền công nghệ đang phát triển hiện nay trên thế giới. Nó được ứng dụng trong mọi lĩnh của đời sống và sản xuất.

Để ứng dụng xu hướng đó vào thực tế ở Việt Nam, đề tài đã nghiên cứu các cơ sở lý thuyết từ phần cứng, phần mềm, giao thức mạng liên quan. Xây dựng được mô hình nhà thông mình cho phép điều khiển và giám sát thời gian thực thông qua giao diện web với các chế độ điều khiển linh hoạt, giao diện giám sát đẹp mắt và thiết bị sử dụng phù hợp với tình hình kinh tế Việt Nam.

## Vấn đề đặt ra

Hiện nay tại Việt Nam có một số công ty, nhóm phát triển đã nghiên cứu và áp dụng ứng dụng IoT vào mô hình nhà thông minh. Tiêu biểu như dự án nhà thông mình của BKAV[[1]](#footnote-1).

Nhưng trong thực tế mô hình này chỉ áp dụng được trong một số chung cư, biệt thự lớn vì chi phí khá cao. Vì thế để áp dụng mô hình nhà thông minh rộng rãi hơn thì cần những thiết bị có giá thành thấp hơn, thuận tiện hơn và đặt biệt là người dùng có thể tự lắp đặt tại nhà.

Đồng thời tài liệu hướng dẫn về IoT chủ yếu là tài liệu nước ngoài, những tài liệu này đa số chỉ hướng dẫn một góc nhỏ nào đó về IoT làm cho việc tiếp cận và tạo ra một sản phẩm IoT hoàn chỉnh hết sức khó khăn cho người mới tìm hiểu về IoT.

## Phương pháp giải quyết

Để đưa công nghệ nhà thông minh đến gần với người dùng, đề tài dựa vào một số công nghệ đơn giản, dễ xây dựng và mở rộng theo ý tưởng của người dùng. Người dùng chỉ tốn chi phí cho các thiết bị phần cứng như Arduino, Raspberry Pi, còn phần mềm có thể tự xây dựng mô hình truyền nhận dữ liệu đơn giản thông qua API, mô hình điều khiển thông qua giao diện web và cuối cùng sử dụng mạng không dây thông qua Wifi có sẵn trong nhà.

Hiện nay công nghệ thông tin đang phát tiển rất nhanh, có rất nhiều công cụ hỗ trợ viết code miễn phí như Visual Code, Atom, Sublime Text,… Một số công cụ tìm kiếm hữu ích đặt biệt là Google[[2]](#footnote-2), Stack Overflow[[3]](#footnote-3). Vì thế người dùng tự tay xây dựng được phần mềm truyền nhận dữ liệu hay trang web điều khiển là một điều hoàn toàn có thể làm được.

## Đối tượng và phạm vi nghiên cứu

### Đối tượng nghiên cứu

Đề tài sẽ nghiên cứu một số đối tượng như:

* Framework hỗ trợ xây dựng API như ASP.NET hoặc NodeJS
* Một số thiết bị phần cứng dùng để thu thập dữ liệu từ các cảm biến hoặc điều khiển các thiết bị ngoại vi như Arduino, Raspberry Pi.
* Một số Framework hoặc thư viện hỗ trợ xây dựng giao diện web.
* Một số cảm biến hỗ trợ thu thập dữ liệu như cảm biến nhiệt độ - độ ẩm, cảm biến mưa, cảm biến ánh sáng,…
* Ứng dụng Docker để triển khai server trên nền tảng điện toán đám mây.

### Phạm vi nghiên cứu

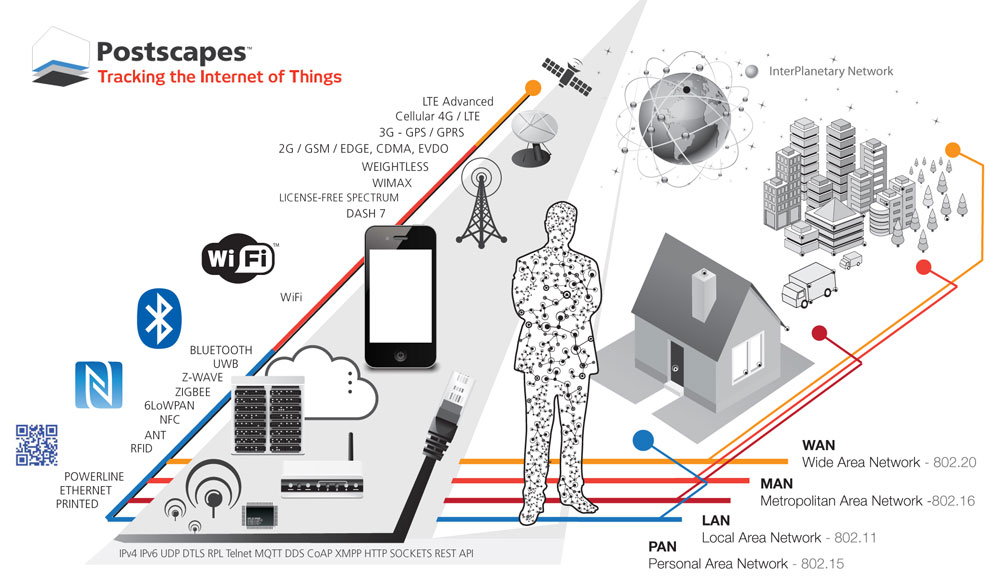
* Nghiên cứu lý thuyết
* Xây dựng mô hình thử nghiệm

## Cấu trúc khóa luận

* Tìm hiểu một số nghiên cứu và công nghệ liên quan đến đề tài.
* Đưa ra một số kịch bản, ngữ cảnh xây dựng.
* Xây dựng một số giải thuật, phương pháp để giải quyết vấn đề.
* Triển khai xây dựng mô hình.
* Phân tích và đánh giá kết quả thực hiện.

# MỘT SỐ CÔNG NGHỆ LIÊN QUAN

## Công nghệ IoT



Hình 2.1: Công nghệ IoTs

Công nghệ IoT không còn mới và nó đã đạt được nhiều thành tựu, ứng dụng trong nhiều mô hình khác nhau. Chẳng hạn như SmartHome, SmartCity, SmartGarden,…

IoT sử dụng nhiều kết nối khác nhau để kết nối các thiết bị lại với nhau như kết nối không dây (Wifi, Bluetooth) và kết nối có dây (Dây cáp).

Sử dụng linh hoạt một số giao thức mạng như Zigbee, Wifi, Bluetooth,…

Người dùng có thể điều khiển thiết bị mọi lúc, mọi nơi thông qua Internet, trên các thiết bị điều khiển như website hoặc ứng dụng di động.

## Điện toán đám mây



Hình 2.2: Điện toán đám mây

Điện toán đám mây (Cloud Computing) là mô hình điện toán sử dụng công nghệ máy tính và phát triển dựa vào mạng Internet.

Điện toán đám mây là việc ảo hóa các tài nguyên tính toán và các ứng dụng thay vì sử dụng máy chủ thật. Các tài nguyên được ảo hóa thông qua mạng Internet.

Một số tiện ích khi sử dụng điện toán đám mây:

* Khả năng mở rộng: Có thể mở rộng tài nguyên một cách dễ dàng.
* Giảm chi phí vận hành và bảo trì.
* Tài nguyên sử dụng linh hoạt.
* Tính an toàn của dữ liệu.
* Tính liên tục.

Một số bất lợi của điện toán đám mây:

* Thời gian chết: Không hệ thống nào có thể chạy 100% trong suốt vòng đời của nó. Vì thế nếu xảy ra sự cố về điện toán đám mây thì chỉ duy nhất nhà cung cấp mới giải quyết được. Người dùng và doanh nghiệp không thể can thiệp được.
* Bảo mật tài liệu: Dữ liệu được lưu trữ trên điện toán đám mây được đảm bảo an toàn, không lộ ra ngoài trừ nhà cung cấp dịch vụ.

Một số nhà cung cấp giải pháp điện toán đám mây lớn hiện nay như Dropbox, Microsoft, Google.

## Một số thư viện, Frameworks liên quan

Đề tài hướng tới xây dựng một mô hình IoT hoàn thiện từ xây dựng Server, giao diện Web, kết nối thiết bị phần cứng,…

### Xây dựng Backend – Server API

#### ASP.NET

ASP.NET là nền tảng phát triển web (Web Application Framework), cung cấp một mô hình lập trình, cơ sở hạ tầng phần mềm toàn diện và các dịch vụ cần thiết để xây dựng các ứng dụng web động mạnh mẽ cho máy tính cũng như trên các thiết bị di động.

ASP.NET là một phần của nền tảng Microsoft.NET. Ứng dụng ASP.NET được biên dịch mã, được viết bằng cách sử dụng mở rộng và tái sử dụng các thành phần hoặc đối tượng trong nền tảng NET. Các mã này được sử dụng cho toàn bộ hệ thống phân cấp của các class trong .NET.

Các ứng dụng ASP.NET có thể được viết bằng bất kỳ ngôn ngữ nào sao đây:

* C#
* Visual Basic.Net
* Jscript
* J#

ASP.NET được phát hành đầu tiên vào 12/2/2012 với tên gọi ASP.NET Framework v1.0 cùng với Visual Studio.NET 2002.

Hiện nay ASP.NET bao gồm ASP.NET Framework và ASP.NET Core.

Một số khác biệt giữa ASP.NET Framework và ASP.NET Core.

|  |  |
| --- | --- |
| **ASP.NET Framework** | **ASP.NET Core** |
| Chạy trên Window | Đa nền tảng : Windows , Mac , Linux |
| Mã nguồn đóng | Mã nguồn mở |
| Chạy trên phiên bản đầy đủ của ASP.Net Framework | Có thể chạy trên .NET Core hoặc trên phiên bản đầy đủ của .NET Framework |
| Nặng nhưng đầy đủ | Gọn nhẹ tích hợp các Framwork khác nhau lại trong một bản duy nhất là ASP.NET Core |
| Build được thực hiện trên giao diện Visual Studio | Build có thể thực hiện bằng Command line hoặc giao diện |
| Hỗ trợ đầy đủ | Không hỗ trợ Web Form |

Bảng 2.1: So sánh ASP.NET Framework và ASP.NET Core

#### NodeJS

NodeJS là một nền tảng chạy trên môi trường V8 JavaScript runtime - một trình thông dịch JavaScript cực nhanh chạy trên trình duyệt Chrome. NodeJS có khả năng mở rộng nhanh chóng, xử lý một số lượng lớn các kết nối đồng thời bằng thông lượng cao. Nếu như các ứng dụng web truyền thống, các request tạo ra một luồng xử lý yêu cầu mới và chiếm RAM của hệ thống thì tài nguyên của hệ thống sẽ được sử dụng không hiệu quả. Chính vì lẽ đó, giải pháp mà NodeJS đưa ra là sử dụng luồng đơn (Single-Threaded), kết hợp với non-blocking I/O để thực thi các request, cho phép hỗ trợ hàng chục ngàn kết nối đồng thời.

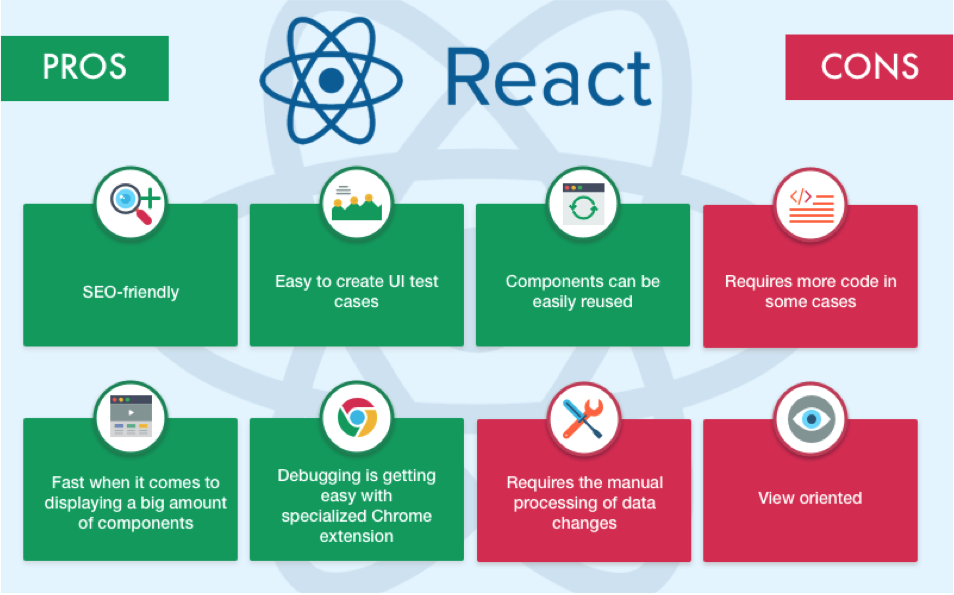
Một số đặc điểm nổi bật của NodeJS:

* Dễ dàng cài đặt
* Tốc độ thực thi nhanh
* Dễ dàng mở rộng

### Xây dựng Frontend – Giao diện Web

Hiện nay ngoài việc xây dựng Web bằng HTML và CSS thuần, còn có các Framework, thư viện hỗ trợ. Trong đó, nổi bật là ReactJS, Angular và VueJS. Trong phạm vi bài viết chỉ cân nhắc sử dụng ReactJS và Angular.

#### ReactJS



Hình 2.3: Thư viện ReactJS

ReactJS là một thư viện JavaScript dùng để xây dựng giao diện người dùng.

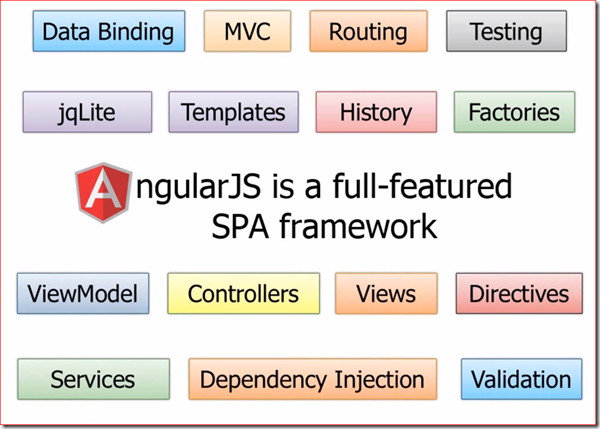
Để sử dụng được ReactJS thì người dùng cần có một chút kiến thức căn bản về Web: HTML, CSS và đặc biệt là JavaScript.

ReactJS được xây dựng dựa trên nhiều Component con và chúng độc lập với nhau. Điều này giúp cho việc tái sử dụng code một cách hiệu quả.

Các Component con nhận State từ Component cha sau đó xử lí và hiển thị dữ liệu nhận được.

React được xây dựng theo mô hình Virtual DOM Tree. Do đó, khi Tree quá lớn thì việc truyền nhận State trở nên phức tạp hơn. Để giải quyết vấn đề này có thể sử dụng Redux.

#### Angular



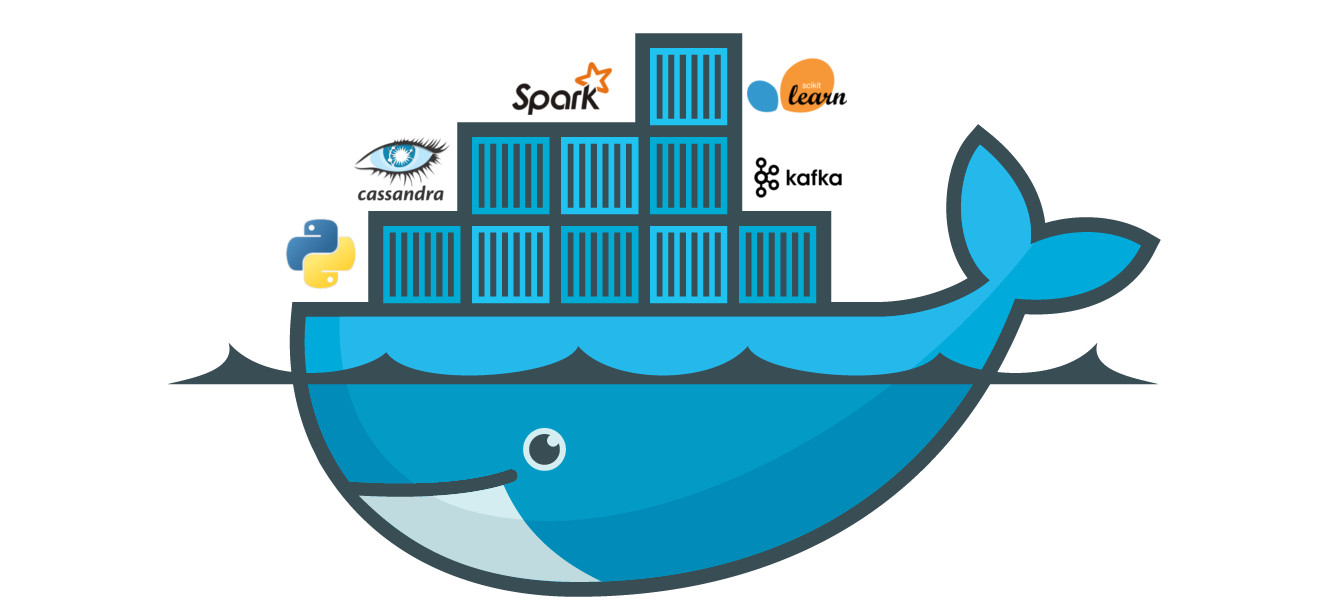
Hình 2.4: Angular Framework

Angular là một Framework cho các ứng dụng Web, được phát triển từ năm 2009 bởi Google.

Phiên bản đầu tiên có tên gọi là AngularJS và phiên bản mới nhất là Angular 6.

Angular được hỗ trợ và phát triển dựa trên JavaScript, tạo ra các ứng dụng client-side theo mô hình MVC.

## Docker



Hình 2.5: Docker

Docker là một Open platform cung cấp cho người sử dụng những công cụ và service để người sử dụng có thể đóng gói và chạy chương trình của mình trên các môi trường khác nhau một cách nhanh nhất.

Các thành phẩn cơ bản của Docker bao gồm:

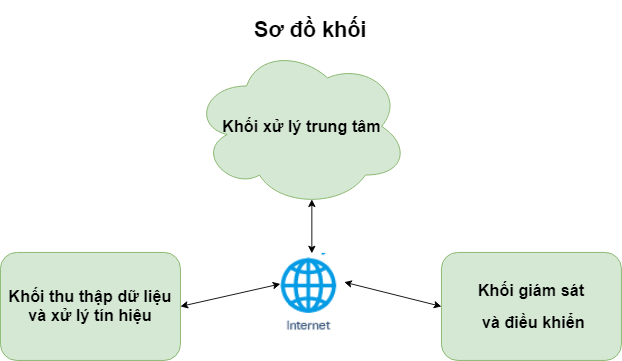
* Image: là file ảnh được đóng gói, có thể là một hệ điều hành, một nền tảng, một ngôn ngữ. Image được dùng để tạo ra Docker Container.
* Container: là một máy ảo tạo ra từ Docker image. Thực thi nội dung bên trong image đó như cài đặt môi trường, cài đặt packages, phần mềm,…
* Docker Engine: dùng để quản lí Docker images và Docker Container.
* Docker Hub: giống như Git, đây là nơi dùng để lưu và chia sẻ các images.

Một số đặc điểm nổi bật của Docker:

* Linh động: Triển khai ở bất kỳ nơi đâu do sự phụ thuộc của ứng dụng vào tầng OS cũng như cơ sở hạ tầng được loại bỏ.
* Nhanh: Do chia sẻ host OS nên container có thể được tạo gần như một cách tức thì.
* Nhẹ: Container cũng sử dụng chung các images nên cũng không tốn nhiều disks.
* Đồng nhất: Khi nhiều người cùng phát triển trong cùng một dự án sẽ không bị sự sai khác về mặt môi trường.
* Đóng gói: Có thể ẩn môi trường bao gồm cả app vào trong một gói được gọi là container. Có thể test được các container. Việc bỏ hay tạo lại container rất dễ dàng.

# HƯỚNG TIẾP CẬN

## Mô hình tổng quan



Hình 3.1: Mô hình tổng quan

### Khối xử lý trung tâm



Hình 3.2: Khối xử lý trung tâm

Khối xử lý trung tâm sẽ được xây dựng dựa trên ASP.NET Core, sử dụng Sqlite làm cơ sở dữ liệu. Cung cấp một số API để quản lý người dùng, đăng nhập, đăng kí, lưu trữ trạng thái thiết bị và dữ liệu thu thập được.

Các dữ liệu xử lý và điều khiển đều thông qua khối xử lý trung tâm.

### Khối thu thập dữ liệu và xử lý tín hiệu



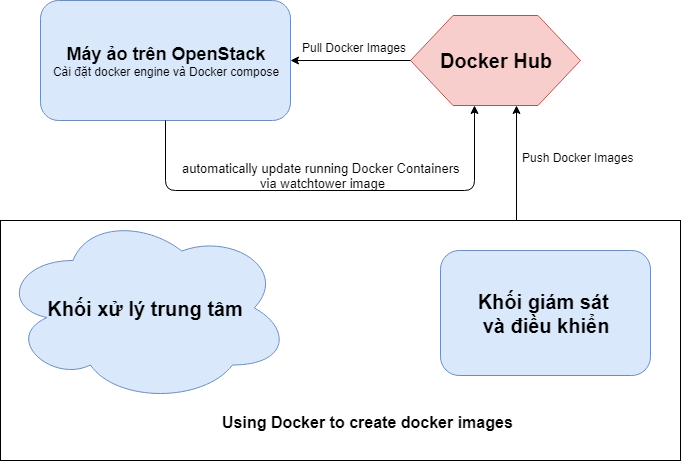
Hình 3.3: Khối thu thập dữ liệu và xử lý tín hiệu

Khối thu thập dữ liệu và xử lý tín hiệu sử dụng board mạch Arduino, Raspberry Pi cùng các cảm biến như cảm biến ánh sáng, cảm biến nhiệt độ - độ ẩm, cảm biến lửa,… để thu thập dữ liệu. Ngoài ra còn sử dụng các thiết bị như servo để đóng - mở cửa, đầu đọc RFID, đèn led để điều khiển tự động hoặc thông qua giao diện điều khiển.

### Khối giám sát và điều khiển

Khối giám sát và điều khiển được xây dựng bằng thư viện ReactJS cùng các thư viện liên quan để tạo nên một giao diện đồ họa nhằm giúp cho việc giám sát và điều khiển trở nên dễ dàng hơn.

### Mô hình triển khai trên Cloud



Hình 3.4: Mô hình triển khai ứng dụng lên Cloud

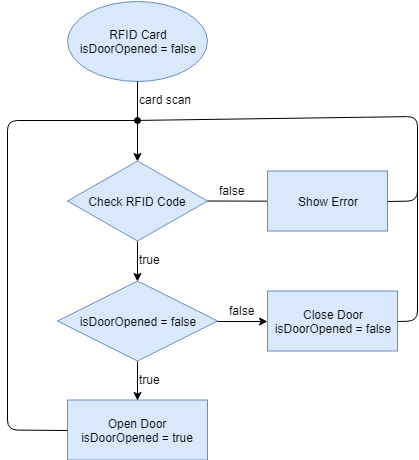
Khối xử lý trung tâm và khối giám sát và điều khiển sẽ sử dụng Docker để đóng gói thành những Docker Images, sau đó được tải lên và lưu trữ trên Docker Hub.

Máy ảo trên OpenStack sẽ được cấu hình để tự động cập nhật Docker Images mới sau một khoảng thời gian nhất định.

## Một số kịch bản

### Quản lý xuất nhập

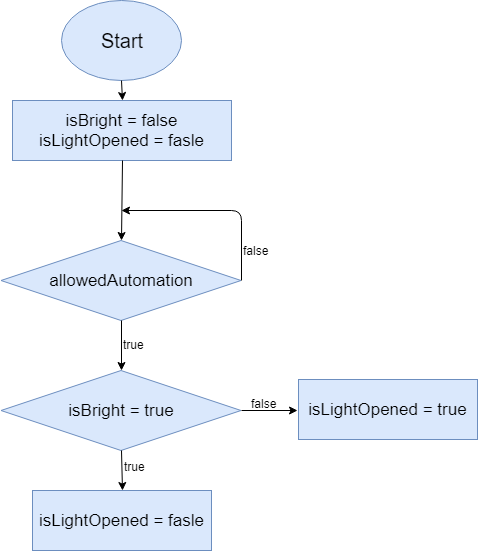
Khi người dùng quét thẻ RFID thì servo sẽ hoạt động, cửa sẽ mở. Khi người dùng quét thẻ lần nữa cửa sẽ đóng.



Hình 3.5: Giải thuật quản lý xuất nhập

### Dùng cảm biến ánh sáng để điều khiển đèn

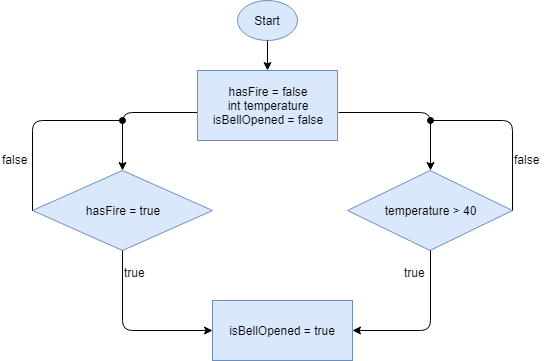
Sử dụng cảm biến ánh sáng để điều khiển đèn trong nhà. Cảm biến ánh sáng chỉ hoạt động khi mode Automation được bật.



Hình 3.6: Giải thuật dùng cảm biến ánh sáng điều khiển đèn

### Báo động khi phát hiện lửa hoặc nhiệt độ cao

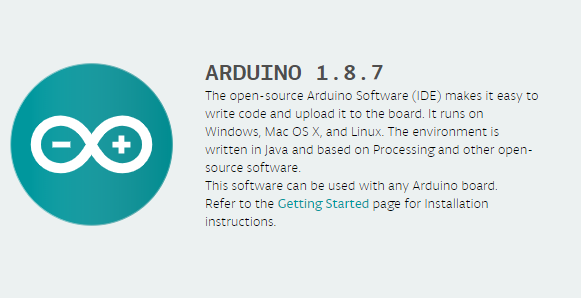
Sử dụng cảm biến phát hiện lửa và cảm biến nhiệt độ, độ ẩm. Khi có lửa hoặc nhiệt độ cao sẽ kích hoạt loa báo động.



Hình 3.7: Giải thuật phát hiện lửa và nhiệt độ cao

## Hiện thực giải thuật

#### Phần mềm và ngôn ngữ sử dụng



Hình 3.8: Arduino IDE

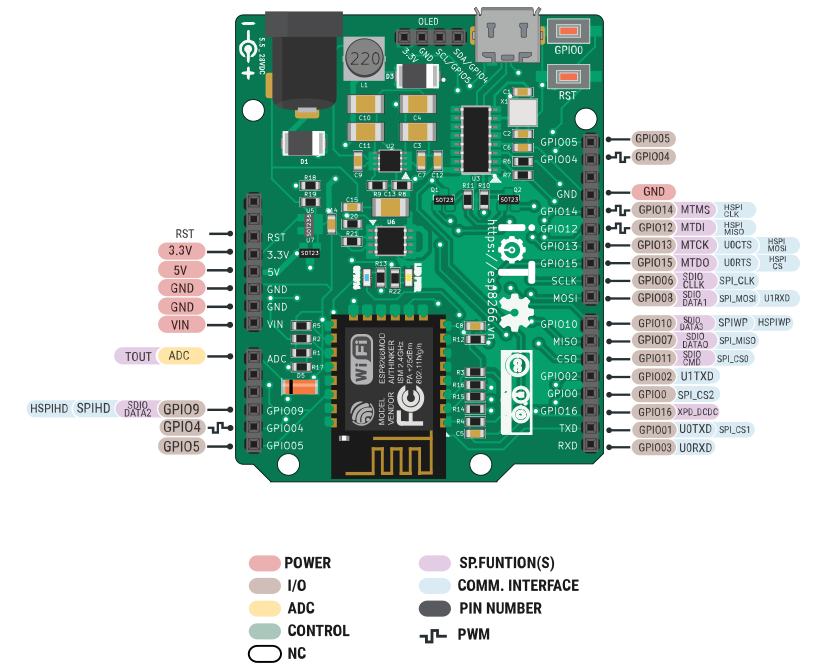
Để có thể code và nạp code vào board mạch Arduino cần sử dụng phần mềm Arduino IDE (Arduino Integrated Development Environment).

Được hỗ trợ đa nền tảng bao gồm Windows , Mac OS , Linux. Phiên bản hiện tại là Arduino IDE v1.8.5 sử dụng ngôn ngữ C để viết code, đồng thời có nhiều thư viện hỗ trợ kết nối Internet, hỗ trợ giao tiếp với các thiết bị ngoại vi, mỗi chương trình viết ra được gọi là sketch, sketch được lưu dưới định dạng .ino.

Ngoài ra còn sử dụng ngôn ngữ Python để lập trình trên Raspberry Pi, kết hợp với thư viện OpenCV để nhận diện khuôn mặt, nhận diện người hoặc phát hiện chuyển động…

Arduino IDE là một trình soạn thảo code cho các thiết bị Arduino. Bao gồm một giao diện code, quản lý kết nối, thư viện, các loại board khác nhau. Đồng thời hỗ trợ compile và nạp code một cách nhanh chóng.

#### Phần cứng sử dụng



Hình 3.9: Arduino ESP8266 Wifi Uno

Phần cứng sẽ sử dụng board mach Arduino ESP8266 Wifi Uno để giao tiếp với các thiết bị ngoại vi nhằm thu thập dữ liệu từ môi trường xung quanh hoặc điều khiển một số thiết bị ngoại vi.

ESP8266 Arduino core đi kèm với thư viện kết nối WiFi hỗ trợ TCP, UDP và các ứng dụng HTTP, mDNS, SSDP, DNS Servers. Ngoài ra còn có thể thực hiện cập nhật OTA, sử dụng Filesystem dùng bộ nhớ Flash hay thẻ SD, điều khiển servos, ngoại vi SPI, I2C.



Hình 3.10: Raspberry Pi

Raspberry Pi là một máy tính thu nhỏ có đầy đủ chức năng như một máy tính thông thường chạy hệ điều hành Linux. Sử dụng Raspberry Pi kết hợp với camera và thư viện OpenCV để nhận diện khuôn mặt, nhận diện chuyển động,…

#### Các thiết bị ngoại vi

Một số thiết bị được sử dụng như:

* Cảm biến lửa
* Cảm biến nhiệt độ - độ ẩm (DHT11)
* Cảm biến ánh sáng
* Servo
* Đèn led
* Màn hình Oled
* Loa cảnh báo
* Webcam / Pi Camera

# XÂY DỰNG HỆ THỐNG GỬI NHẬN DỮ LIỆU THÔNG QUA API BẰNG ASP.NET CORE

## Tổng quan

Trong phạm vi nghiên cứu này sẽ sử dụng ASP.NET Core 2.0 để xây dựng một số API nhằm cung cấp môi trường truyền nhận dữ liệu cho các thiết bị Arduino, Raspberry Pi và ứng dụng Web.

## Mục tiêu

Sử dụng ASP.NET Core để tạo ra một số API cơ bản nhằm hỗ trợ giám sát, điều khiển và thu thập dữ liệu.

Xây dựng được Backend có khả năng tùy chỉnh, thay đổi theo mong muốn, dễ dàng triển khai.

Xây dựng một số API cung cấp các chức năng sau:

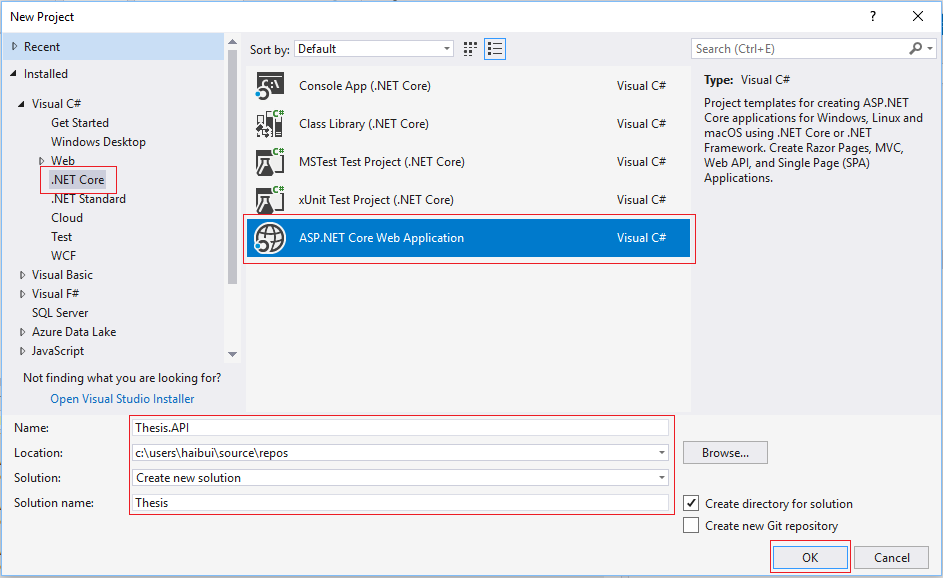
* Đăng nhập, đăng kí.
* Quản lí người dùng : thêm, xóa, cập nhật thông tin người dùng.
* Điều khiển tín hiệu: đèn, cửa ra vào
* Thu thập dữ liệu: nhiệt độ, độ ẩm, ánh sáng, chuyển động, cảnh báo lửa, trạng thái hiện tại của các đèn trong nhà…

## Các bước thực hiện

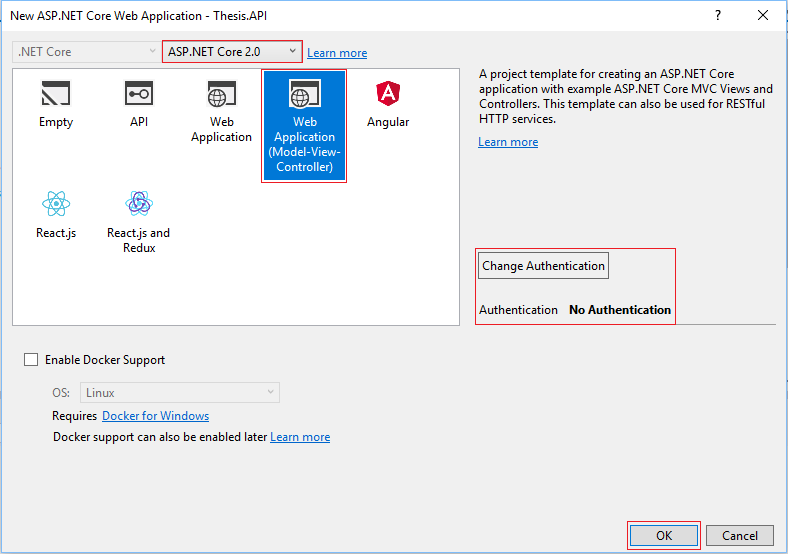
### Bước 1: Tạo một project mới bằng Visual Studio

Mở Visual Studio phiên bản nên lớn hơn 2017. Bấm tổ hợp phím “ctrl + shift + n” để tạo một project mới như hình bên dưới.

Sử dụng ASP.NET core 2.0 để xây dựng server.



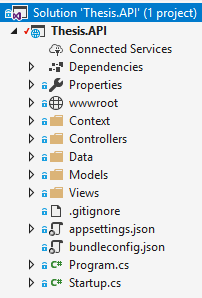
Hình 4.1: Tạo một project (1)



Hình 4.2: Tạo một project (2)

### Bước 2: Cấu trúc thư mục

Tạo cấu trúc thư mục như hình bên dưới.



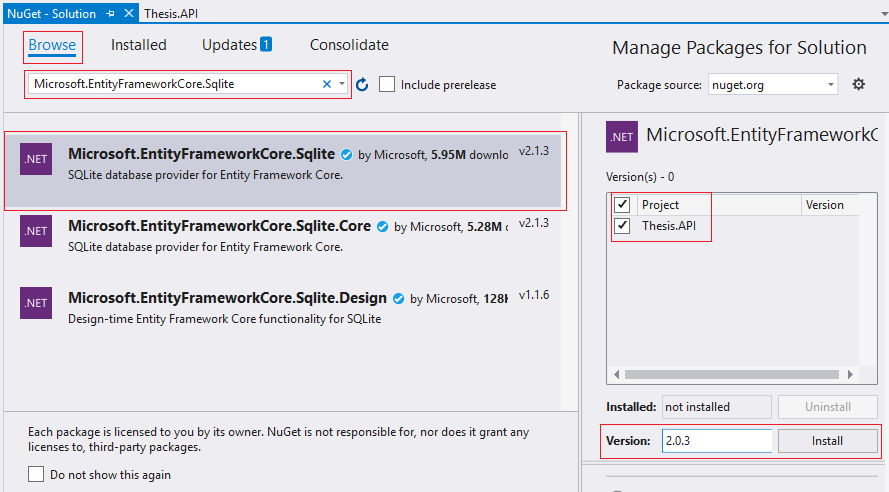
Hình 4.3: Cấu trúc thư mục của Project

### Bước 3: Install các package cần thiết

Vào Tool -> NuGet Package Manager -> Manage Nuget Packages for Solution…

Cần cài một số packages sau:

* Microsoft.EntityFrameworkCore.Tools
* Microsoft.EntityFrameworkCore.Sqlite
* Microsoft.EntityFrameworkCore.SqlServer
* System.Linq
* Microsoft.AspNet.WebApi.Cors



Hình 4.4: Cài đặt các packages cần thiết

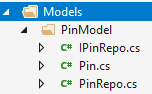
Với ASP.NET core 2.0 nên chọn package version 2.0.x và cài tương tự với các packages còn lại.

### Bước 4: Tạo Model để định dạng dữ liệu và các phương thức truyền nhận

#### PinModel

Mục đích của PinModel nhằm lưu trữ trạng thái của các cổng GPIO của Arduino. Chỉ có hai trạng thái duy nhất đó là 0 và 1.

Tạo cấu trúc thư mục cho PinModel như hình bên dưới.

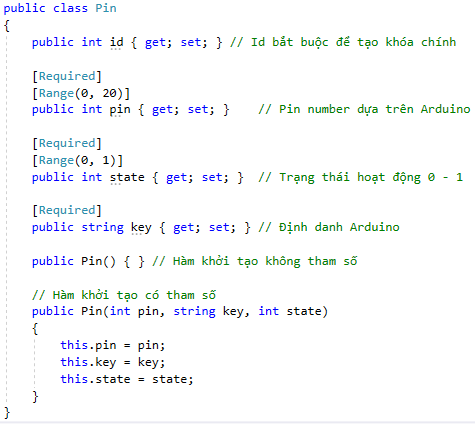


Hình 4.5: Cấu trúc thư mục của PinModel

Trong đó:

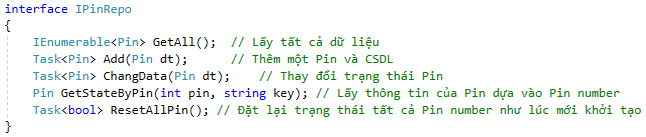
* IPinRepo.cs: Dùng để định nghĩa các phương thức sẽ được sử dụng.
* Pin.cs: Dùng để định nghĩa các thuộc tính cần thiết.
* PinRepo: Dùng để triển khai các phương thức đã được định nghĩa.

Định nghĩa các thuộc tính cần thiết như hình dưới.



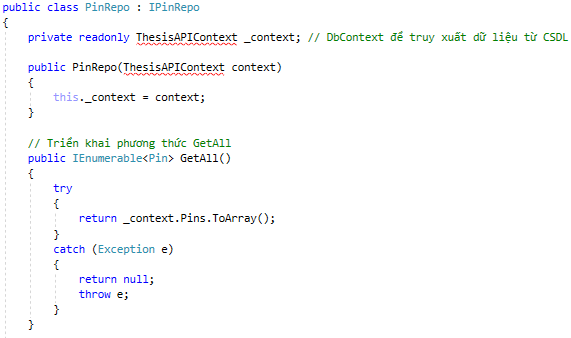
Hình 4.6: Định nghĩa thuộc tính của PinModel

Định nghĩa các phương thức dùng để truyền nhận dữ liệu.



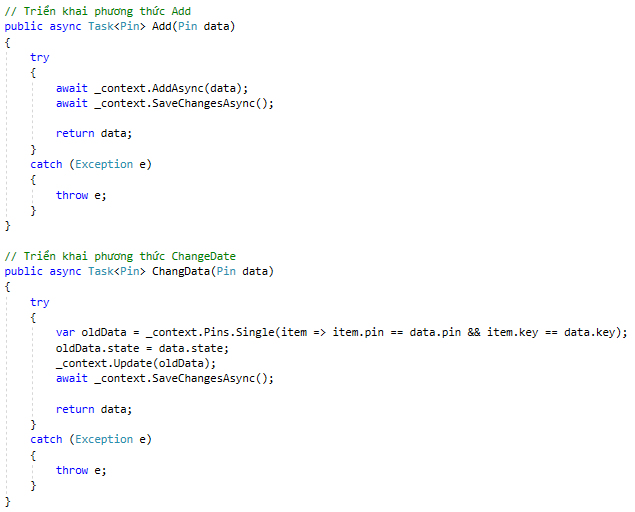
Hình 4.7: Định nghĩa phương thức của PinModel

Sau khi đã khai báo các thuộc tính và phương thức sẽ tiến hành triển khai các phương thức đã định nghĩa ở trên.

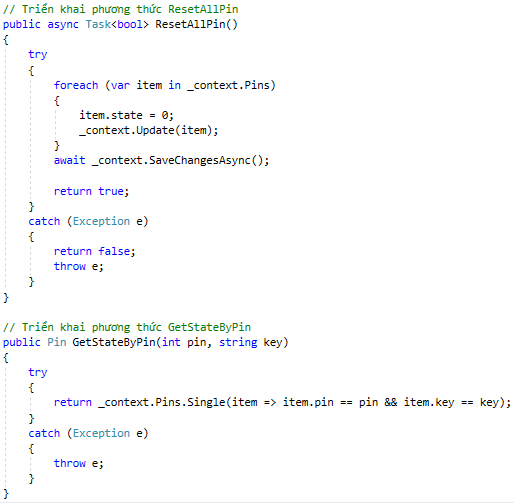


Hình 4.8: Triển khai phương thức của PinModel (1)

Ở bên trên báo lỗi không tìm thấy “ThesisAPIContext”, phần này sẽ khởi tạo bên dưới sau khi đã định nghĩa tất cả Model.



Hình 4.9: Triển khai phương thức của PinModel (2)



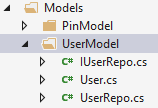
Hình 4.10: Triển khai phương thức của PinModel (3)

#### UserModel

Mục đích của UserModel nhằm lưu thông tin của người dùng bao gồm các thông tin cơ bản như tên người dùng và mã số thẻ RFID tương ứng.

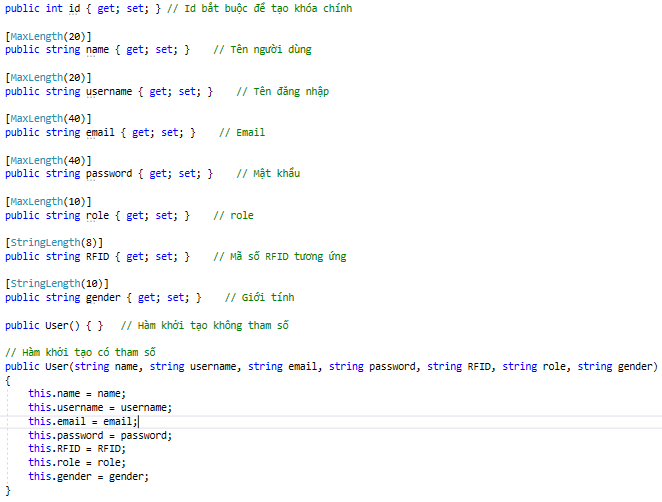
Các bước thực hiện tương tự PinModel nên sẽ không giải thích lại.

Tạo cấu trúc thư mục.



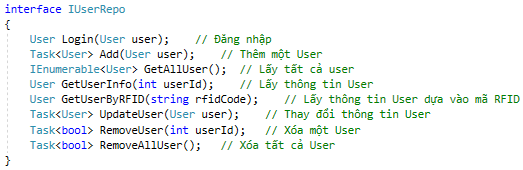
Hình 4.11: Cấu trúc thư mục của UserModel

Định nghĩa các thuộc tính.



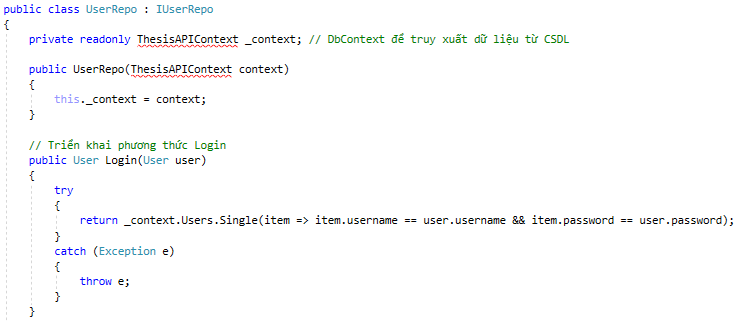
Hình 4.12: Định nghĩa thuộc tính của UserModel

Định nghĩa các phương thức.

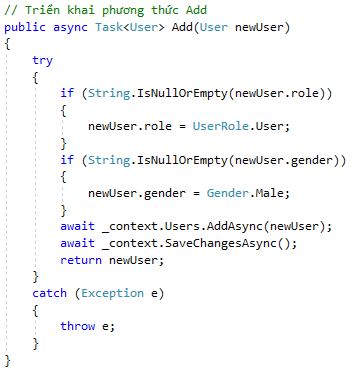


Hình 4.13: Định nghĩa phương thức của UserModel

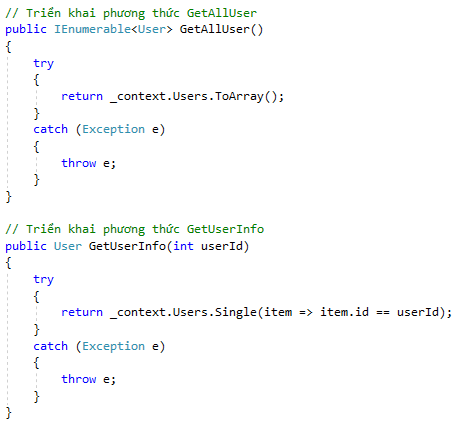
Triển khai các phương thức.



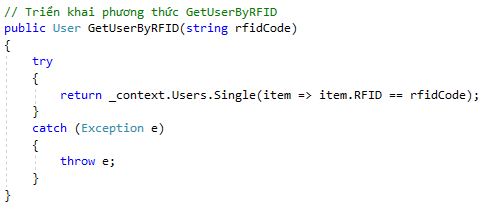
Hình 4.14: Triển khai phương thức của UserModel (1)



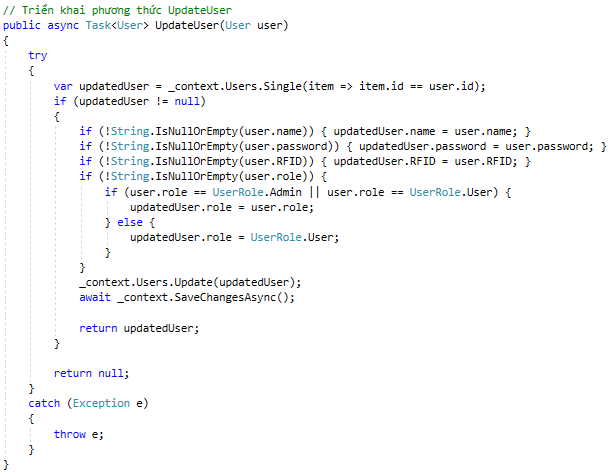
Hình 4.15: Triển khai phương thức của UserModel (2)



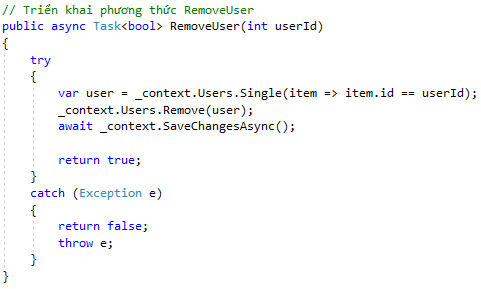
Hình 4.16: Triển khai phương thức của UserModel (3)



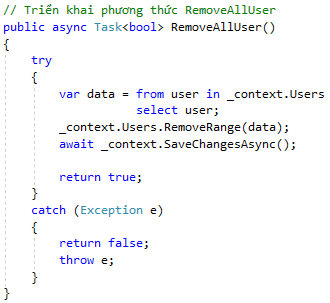
Hình 4.17: Triển khai phương thức của UserModel (4)



Hình 4.18: Triển khai phương thức của UserModel (5)



Hình 4.19: Triển khai phương thức của UserModel (6)



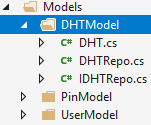
Hình 4.20: Triển khai phương thức của UserModel (7)

#### DHTModel

Mục đích của DHTModel nhằm lưu trữ thông tin về nhiệt độ và độ ẩm thu thập được từ cảm biến DHT11[[4]](#footnote-4).

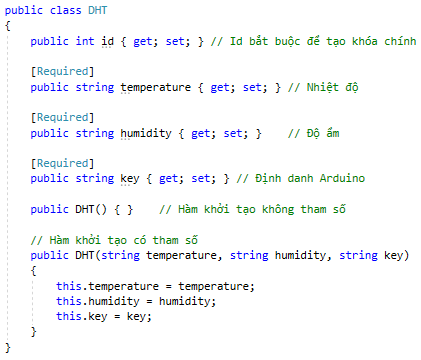
Các bước thực hiện tương tự PinModel nên sẽ không giải thích lại.

Tạo cấu trúc thư mục.



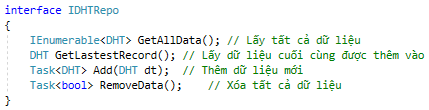
Hình 4.21: Cấu trúc thư mục của DHTModel

Định nghĩa các thuộc tính.



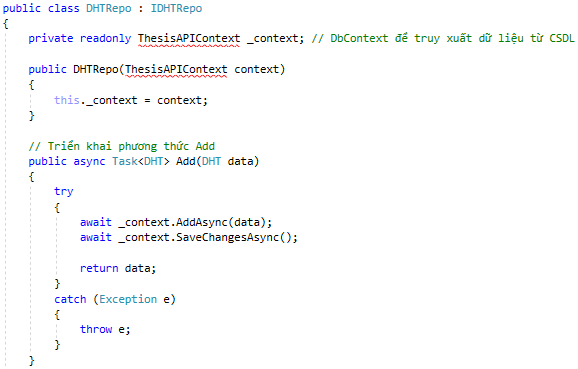
Hình 4.22: Định nghĩa thuộc tính của DHTModel

Định nghĩa các phương thức.

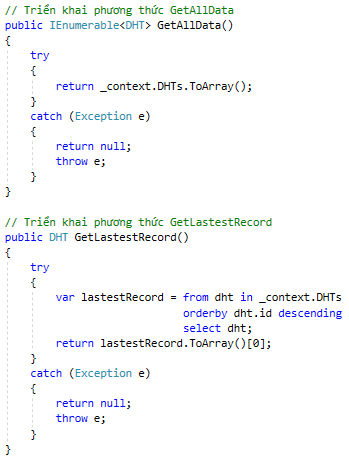


Hình 4.23: Định nghĩa phương thức của DHTModel

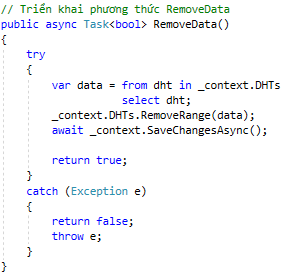
Triển khai các phương thức.



Hình 4.24: Triển khai phương thức của DHTModel (1)



Hình 4.25: Triển khai phương thức của DHTModel (2)

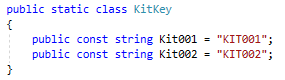


Hình 4.26: Triển khai phương thức của DHTModel (3)

#### KitKey

Định nghĩa “key” sẽ được dùng để định danh cho từng thiết bị Arduino.

Tạo một file “KitKey.cs” bên trong thư mục Models.



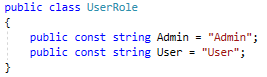
Hình 4.27: Nội dung file “KitKey.cs”

Ở đây khai báo theo số lượng thiết bị hiện tại. Nếu nhiều hơn hai thiết bị thì có thể khai báo thêm.

#### UserRole

Định nghĩa vai trò của từng User.

Tạo một file “UserRole.cs” bên trong thư mục Models.

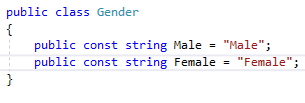


Hình 4.28: Nội dung file “UserRole.cs”

#### Gender

Định nghĩa giới tính cho User

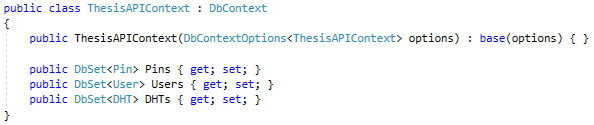
Tạo một file “Gender.cs” bên trong thư mục Models.



Hình 4.29: Nội dung file “Gender.cs”

### Bước 5: Tạo DBContext để tương tác với CSDL

Trong thư mục context, tạo một file “ThesisAPIContext.cs” có nội dung như sau.

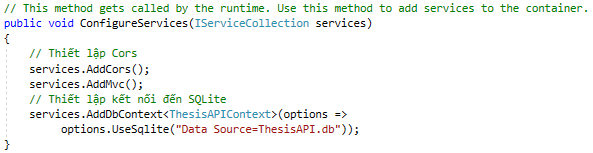


Hình 4.30: Nội dung file “ThesisAPIContext.cs”

Sau khi tạo file “ThesisAPIContext.cs”, tiến hành cập nhật cho các file “PinRepo.cs”, “UserRepo.cs”, “DHTRepo.cs”.

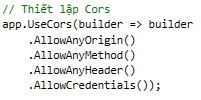
### Bước 6: Chỉnh sửa file “Startup.cs”

Trong file “Startup.cs”, thiết lập kết nối với SQLite[[5]](#footnote-5).



Hình 4.31: Thiết lập kết nối với SQLite

Thêm đoạn code bên dưới vào hàm “Configure”, trước “app.UseMvc”.



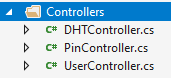
Hình 4.32: Thiết lập Cors

### Bước 7: Tạo API

Tiến hành tạo các Controller để có thể gửi nhận dữ liệu thông qua các API mà Controller cung cấp.

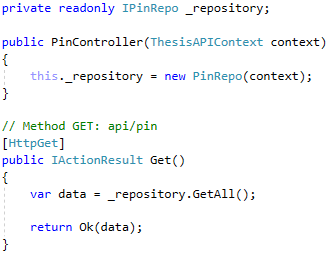
Để tạo Controller, chuột phải vào thư mục Controllers -> Add -> Controller… -> API Controller – Empty -> Add, đặt tên cho Controller và chọn Add.

Tạo Controller tương ứng với các Model như hình.

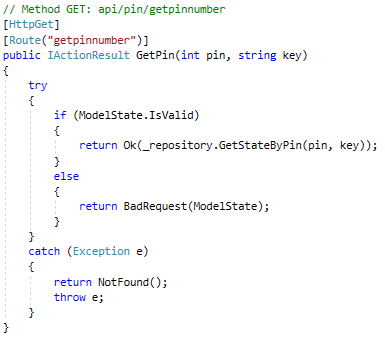


Hình 4.33: Tạo các Controller

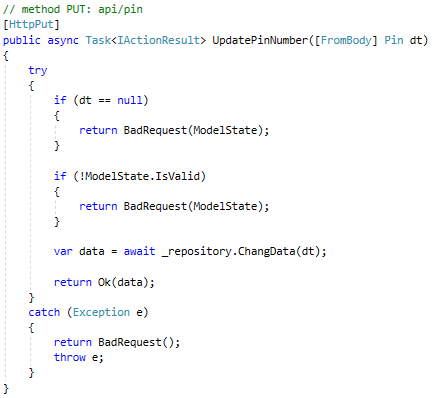
#### PinController



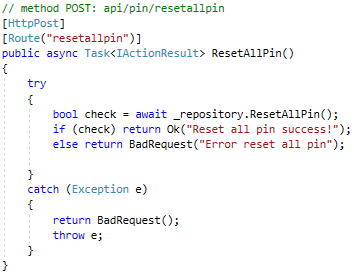
Hình 4.34: Nội dung file “PinController.cs” (1)



Hình 4.35: Nội dung file “PinController.cs” (2)



Hình 4.36: Nội dung file “PinController.cs” (3)

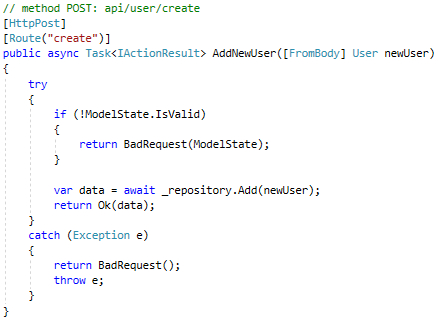


Hình 4.37: Nội dung file “PinController.cs” (4)

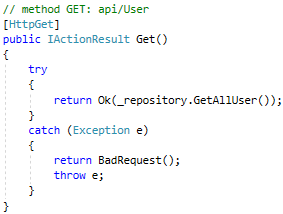
#### UserController



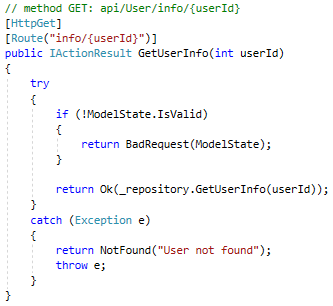
Hình 4.38: Nội dung file “UserController.cs” (1)



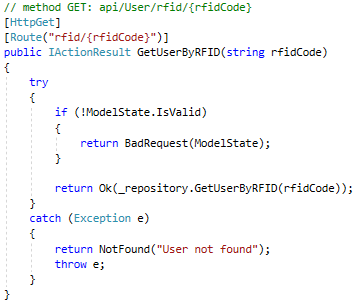
Hình 4.39: Nội dung file “UserController.cs” (2)



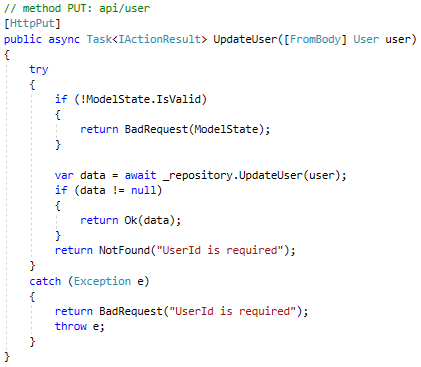
Hình 4.40: Nội dung file “UserController.cs” (3)



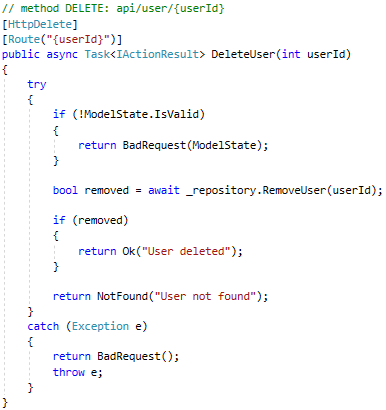
Hình 4.41: Nội dung file “UserController.cs” (4)



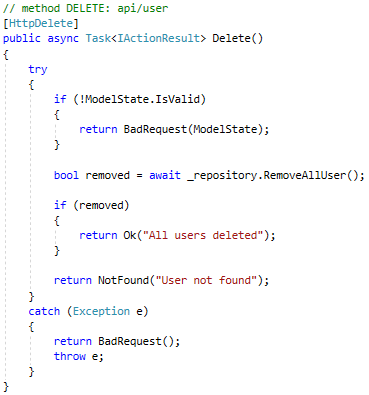
Hình 4.42: Nội dung file “UserController.cs” (5)



Hình 4.43: Nội dung file “UserController.cs” (6)

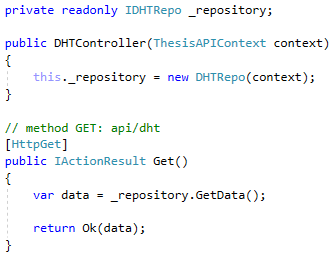


Hình 4.44: Nội dung file “UserController.cs” (7)

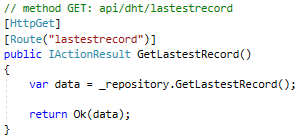


Hình 4.45: Nội dung file “UserController.cs” (8)

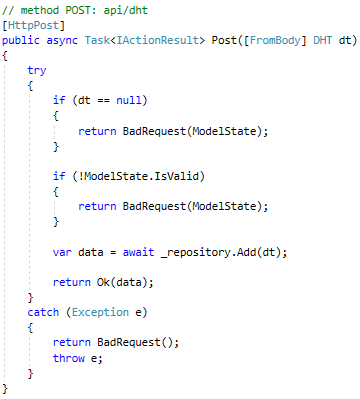
#### DHTController



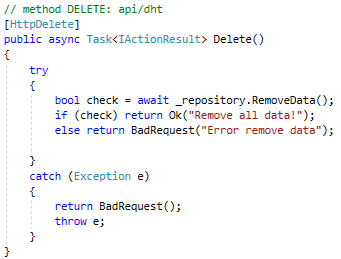
Hình 4.46: Nội dung file “DHTController.cs” (1)



Hình 4.47: Nội dung file “DHTController.cs” (2)



Hình 4.48: Nội dung file “DHTController.cs” (3)



Hình 4.49: Nội dung file “DHTController.cs” (4)

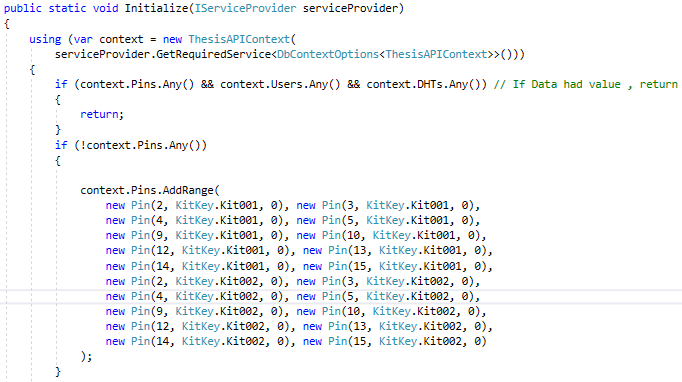
### Bước 8: Tạo dữ liệu mặc định

Trong thư mục Data, tạo một class mới tên SeedData.cs

Mục đích là tạo ra những dữ liệu mặc định trong CSDL khi chạy chương trình lần đầu tiên, hoặc sau khi xóa CSDL và tạo mới CSDL.



Hình 4.50: Tạo SeedData

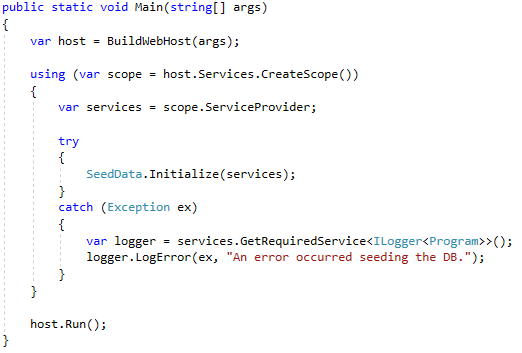


Hình 4.51: Nội dung file “SeedData.cs” (1)



Hình 4.52: Nội dung file “SeedData.cs” (2)

Khai báo dữ liệu vừa tạo vào file Program.cs -> hàm Main()



Hình 4.53: Khai báo SeedData

### Bước 9: Migration

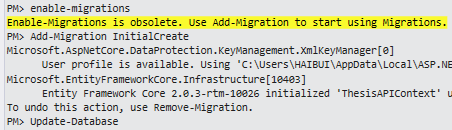
Chuột phải vào Solution -> Properties -> Common Properties -> Startup Project -> Single startup project. Chuyển “docker-compose” thành “Thesis.API”.

Chọn Tool -> Nuget Package Manager -> Package Manager Console

Chạy từng lệnh sau:

* enable-migrations
* Add-Migration InitialCreate
* Update-Database

Sau khi chạy xong ta có kết quả như sau.



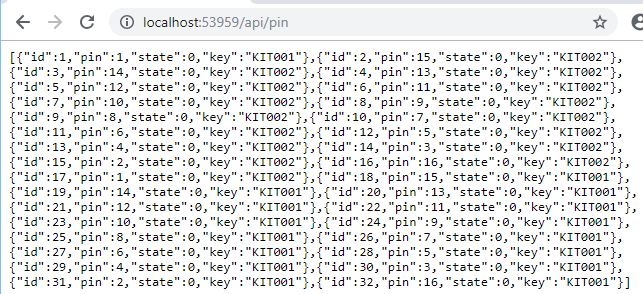
Hình 4.54: Chạy Migration

Trong project sẽ thấy một thư mục Migrations được tự động sinh ra. Thư mục này sẽ tạo bảng tự động trong CSDL khi chạy chương trình. Đồng thời xuất hiện file “ThesisAPI.db”. Đây là file SQLite, tự động sinh ra sau khi chạy dòng lệnh cuối cùng.

### Bước 10: Chạy chương trình

Có thể chạy chương trình bằng nhiều cách như : IISServer, console, dotnet run

Sau khi chạy chương trình, kiểm tra đường dẫn [http://localhost:<port>/api/pin](http://localhost:%3cport%3e/api/pin) được kết quả như sau.



Hình 4.55: Kết quả chạy chương trình

Kiểm tra tương tự với các API còn lại.

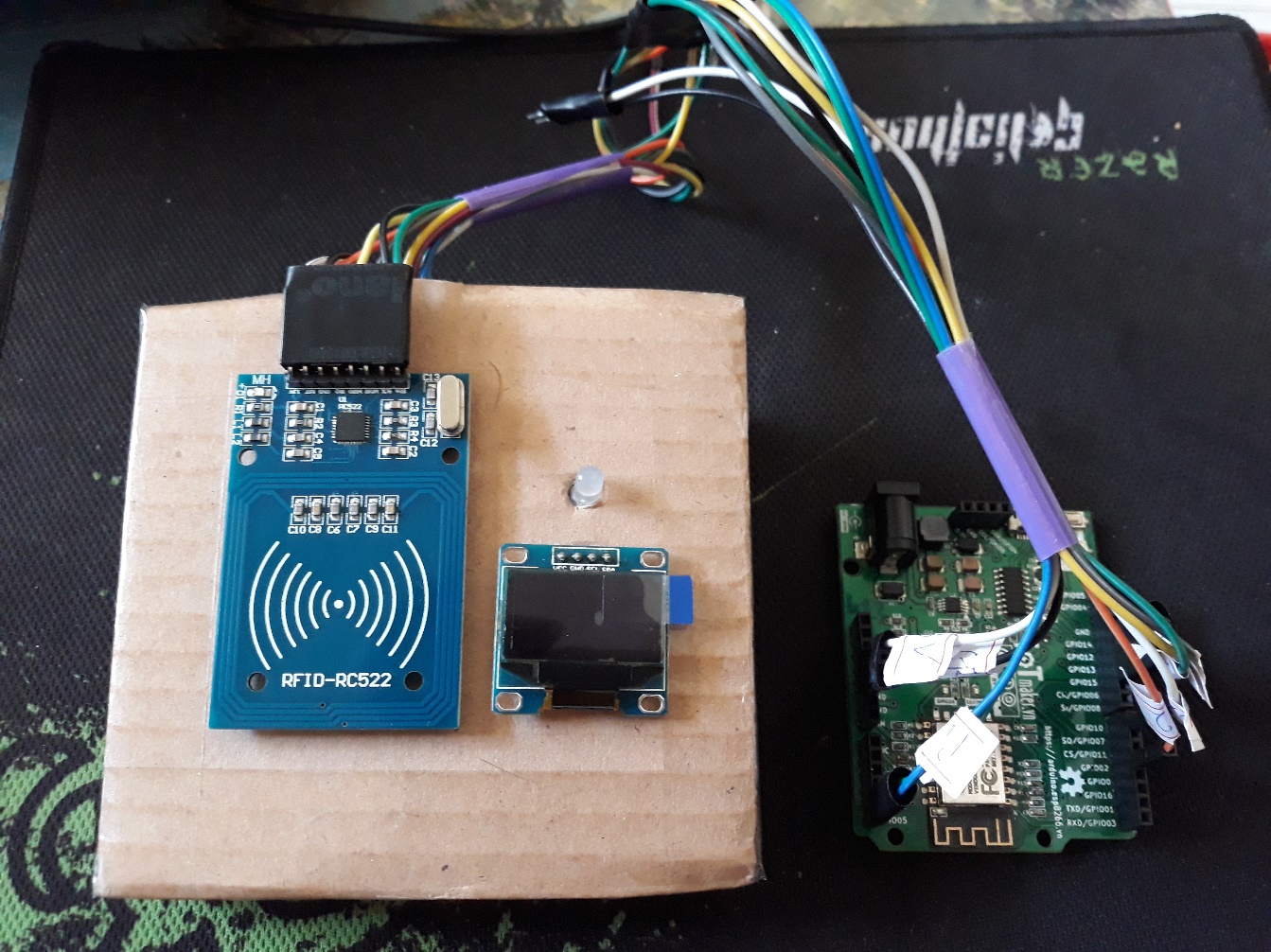
# XÂY DỰNG HỆ THỐNG THU THẬP DỮ LIỆU

## Tổng quan

Xây dựng hệ thống thu thập dữ liệu nhằm có thể giám sát được sự thay đổi của môi trường xung quanh như cảm biến cháy, nhiệt độ cao, thay đổi ánh sáng,… từ đó đưa ra những tín hiệu cảnh báo hoặc điều khiển.

## Xây dựng hệ thống

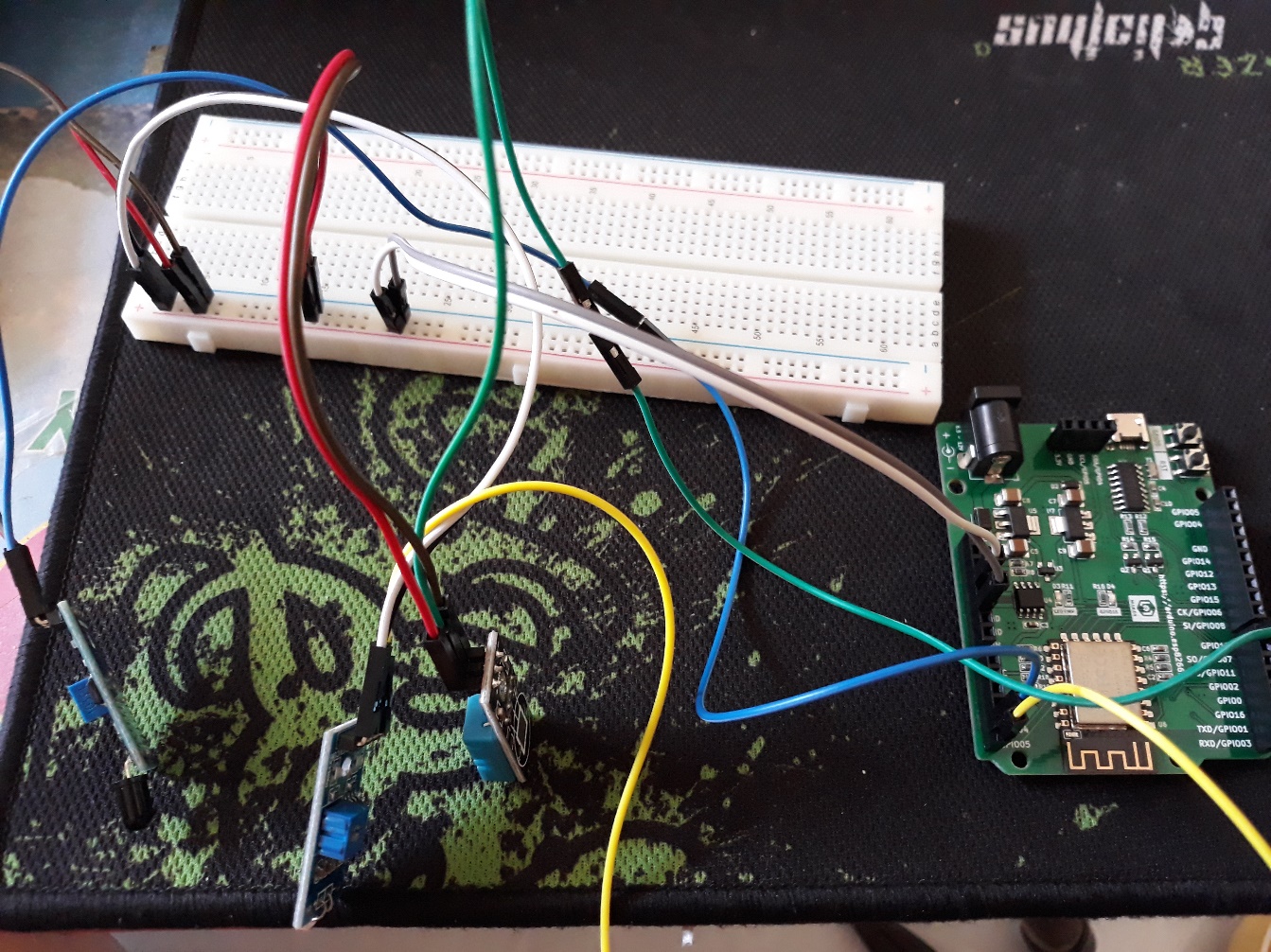
### Arduino ESP8266 xử lý tín hiệu RFID



Hình 5.3: Hệ thống xử lý tín hiệu RFID

Hệ thống này có chức năng duy nhất đó là xử lý tín hiệu nhận được từ đầu đọc RFID, sau đó xác thực thẻ RFID và gửi đi một số tín hiệu điều khiển cần thiết.

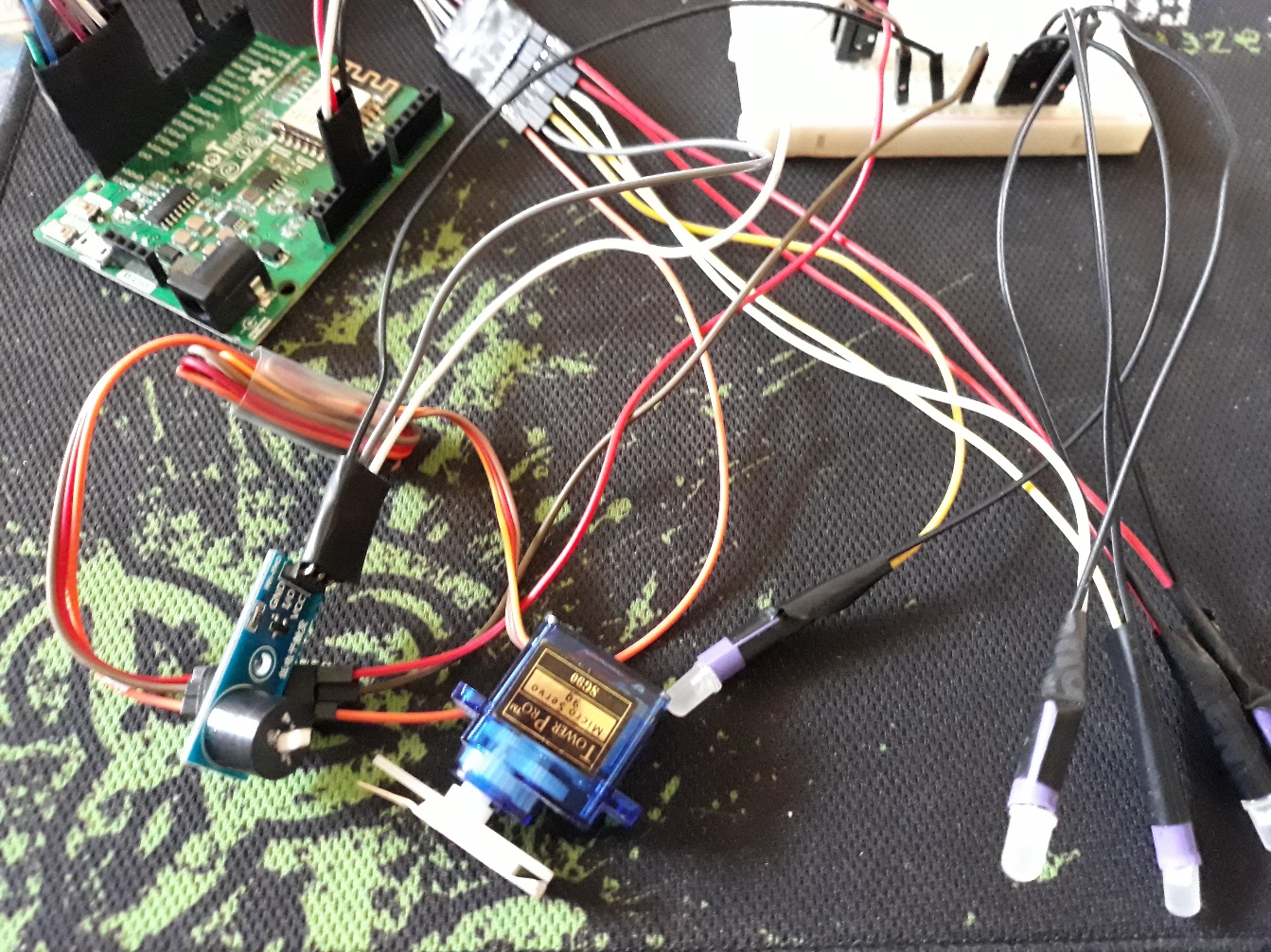
### Arduino ESP8266 gửi dữ liệu



Hình 5.4: Arduino ESP8266 gửi dữ liệu

Hệ thống này chỉ có nhiệm vụ thu thập dữ liệu thông qua các cảm biến ánh sáng, cảm biến cháy, cảm biến nhiệt độ, độ ẩm,... và gửi tín hiệu lên server thông qua các API.

### Arduino ESP8266 nhận dữ liệu



Hình 5.5: Arduino ESP8266 nhận dữ liệu

Hệ thống này có nhiệm vụ nhận dữ liệu từ server thông qua API, sau đó điều khiển các thiết bị dựa vào dữ liệu nhận được.

# XÂY DỰNG HỆ THỐNG GIÁM SÁT VÀ ĐIỀU KHIỂN

## Tổng quan

Hệ thống giám sát và điều khiển được xây dựng trên nền Web. Sử dụng thư viện ReactJS làm khuôn cho ứng dụng. Redux để hỗ trợ quản lí State của React tốt hơn. Ngoài ra còn có một số package hỗ trợ cần thiết khác.

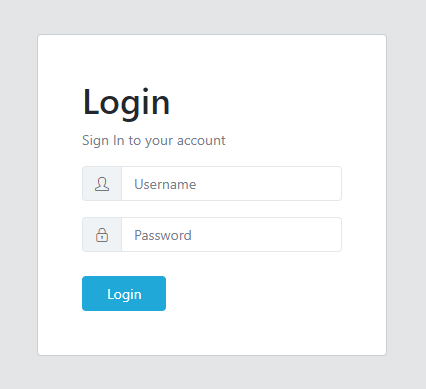
## Mục tiêu

Xây dựng một ứng dụng Web có khả năng cho phép:

* Theo dõi dữ liệu thu thập được từ các cảm biến trong nhà.
* Điều khiển được một số thiết bị trong nhà.
* Quản lí người dùng trong nhà.

## Một số chức năng

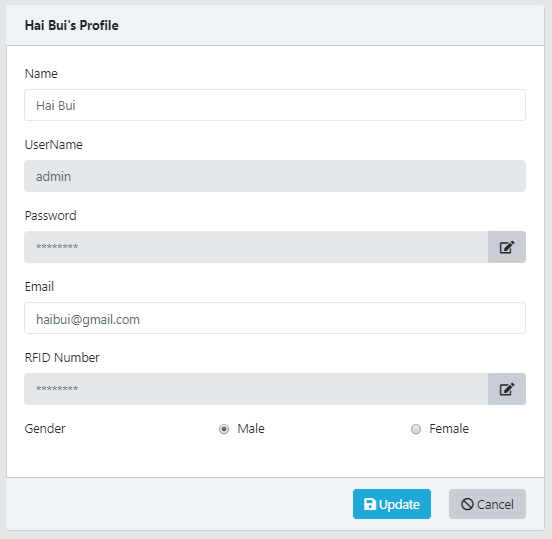
### Login



Hình 6.1: Chức năng Login

Chức năng Login dùng để hạn chế người ngoài có thể truy cập vào giao diện điều khiển và giám sát thiết bị.

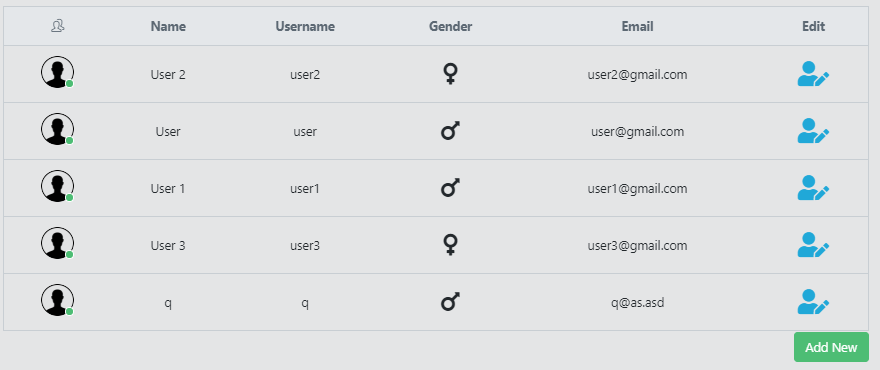
### Profile



Hình 6.2: Chức năng xem và chỉnh sửa thông tin người dùng

Chức năng này cho phép người dùng chỉnh sửa thông tin cá nhân của mình như password, mã thẻ RFID, và một số thông tin khác.

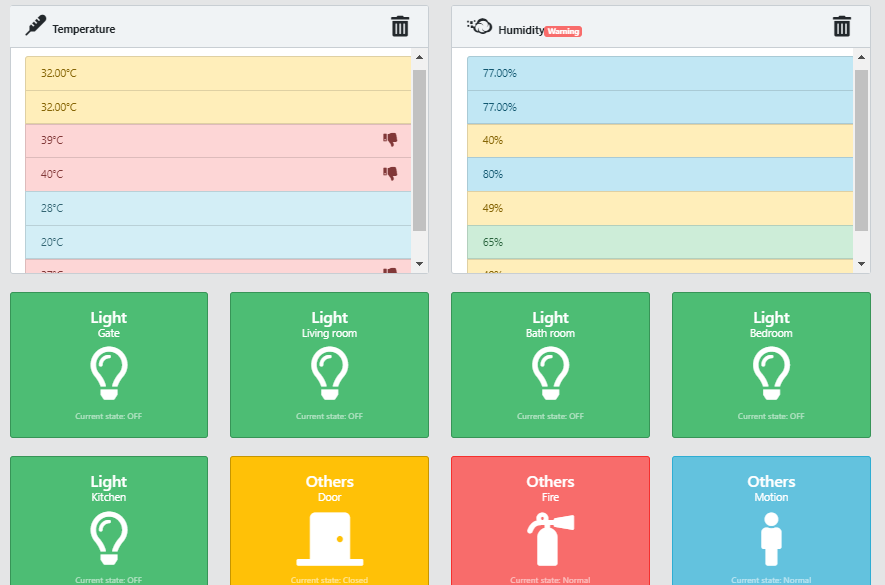
### Quản lí người dùng



Hình 6.3: Chức năng quản lí người dùng

Chức năng này chỉ dành riêng cho “Admin” – dành cho bố hoặc mẹ có thể quản lí, tránh trường hợp tạo người dùng không mong muốn.

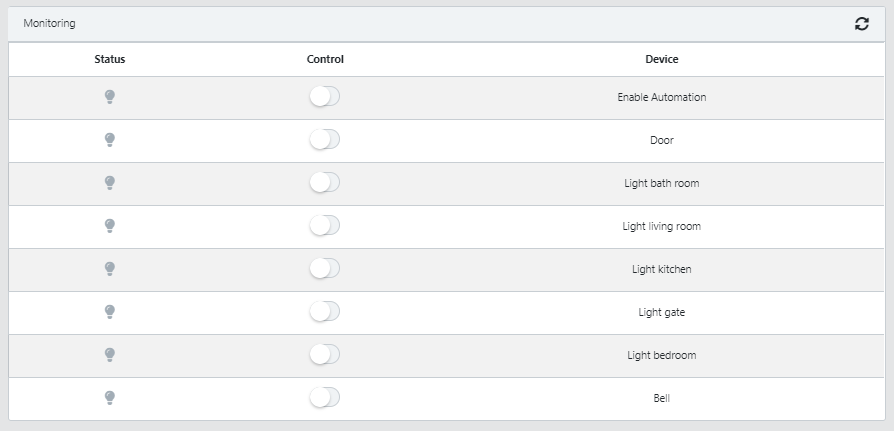
### Giám sát



Hình 6.4: Chức năng giám sát

Chức năng này cho phép người dùng theo dõi được một số thông tin về các thiết bị trong nhà như đèn nào bật, đèn nào tắt, nhiệt độ hiện tại, cửa đóng hay mở, cảnh báo lửa,…

### Điều khiển



Hình 6.5: Chức năng điều khiển

Chức năng điều khiển có nhiệm vụ điều khiển thiết bị thông qua giao diện, bao gồm điều khiển đèn và mở cửa.

Ngoài ra có thể bật tắt chế độ tự động để ngôi nhà có thể làm một số việc khi không có người trong nhà như: tắt đèn khi trời sáng, bật một số đèn khi trời tối,..

# TRIỂN KHAI HỆ THỐNG LÊN OPENSTACK

## Tổng quan

Trong nghiên cứu này sẽ sử dụng OpenStack - Mitaka của trường Đai học Công nghệ thông tin để triển khai Server.

Sử dụng Docker để triển khai Backend API và ứng dụng Web lên OpenStack một cách dễ dàng.

## Mục tiêu

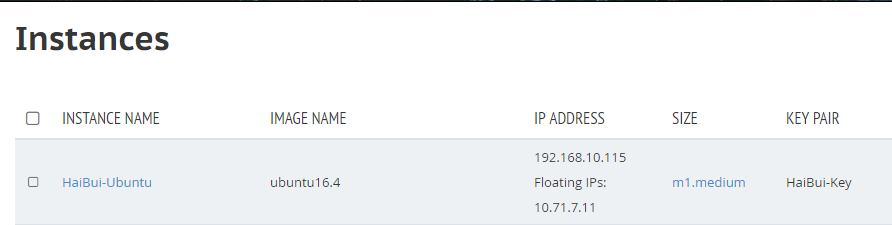
Triển khai được hệ thống lên OpenStack – Mitaka bằng Docker.

Có thể truy cập hệ thống bất cứ lúc nào trong phạm vi trường Đại học Công nghệ thông tin.

Tạo nền tảng cho việc triển khai trên các hosting khác sau này.

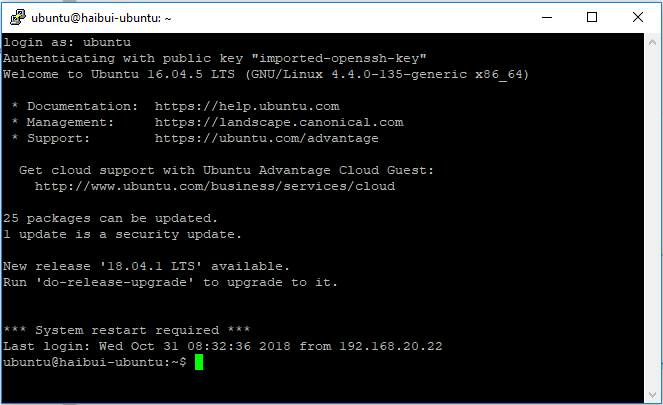
## Các bước thực hiện

### Tạo máy ảo trên OpenStack



Hình 7.1: Tạo máy ảo trên OpenStack

### Truy cập vào địa chỉ máy ảo bằng Putty

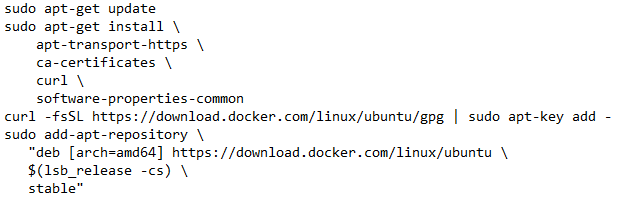


Hình 7.2: Truy cập máy ảo bằng Putty

### Tiến hành cài đặt Docker trên máy ảo

#### Cài đặt môi trường

Chạy một số lệnh sau để thiết lập các thư viện, các biến môi trường cần thiết trước khi cài đặt Docker.



Hình 7.3: Thiết lập môi trường cho Docker

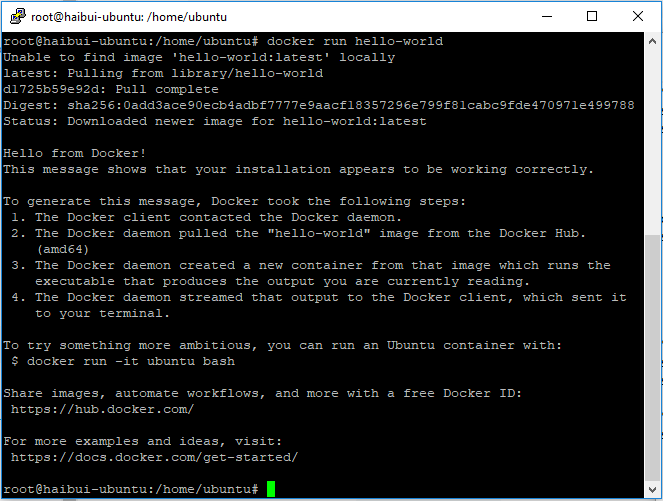
#### Cài đặt Docker CE

Sau khi đã cài đặt môi trường cho Docker, tiến hành cài đặt Docker CE như sau



Hình 7.4: Cài đặt Docker CE

Kiểm tra Docker CE đã được cài đặt



Hình 7.5: Kiểm tra cài đặt Docker

Như vậy Docker đã được cài đặt thành công trên máy ảo. Tiếp đến cần cài đặt Docker Compose cho máy ảo.

#### Cài đặt Docker Compose

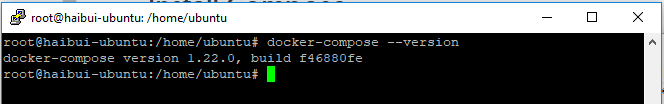
Docker Compose giúp quản lý các container của Docker một cách tốt hơn, chạy nhiều container cùng lúc chỉ với một dòng lệnh so với cách chạy container thông thường.

Cài đặt Docker Compose như sau



Hình 7.6: Cài đặt Docker Compose

Kiểm tra cài đặt của Docker Compose

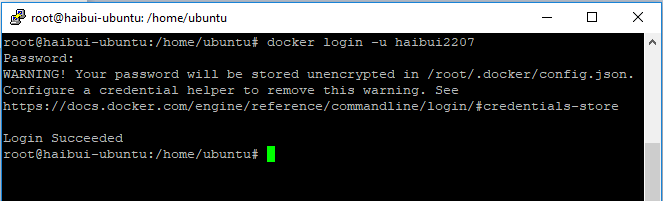


Hình 7.7: Kiểm tra cài đặt Docker Compose

Như vậy Docker Compose đã được cài đặt thành công.

#### Đăng nhập vào Docker

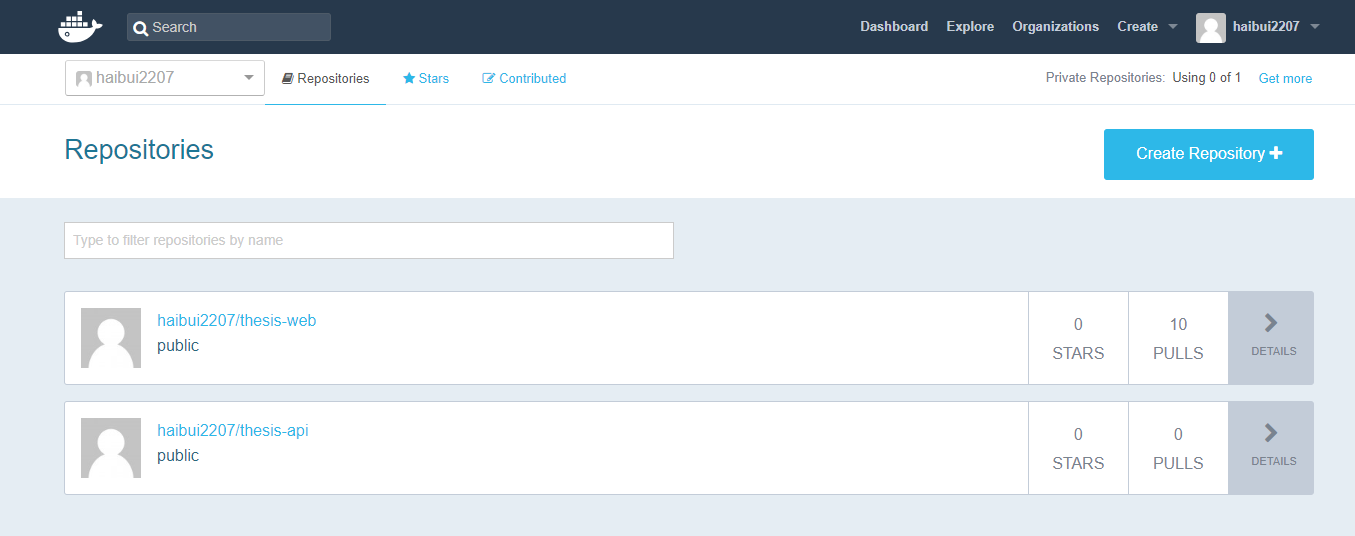
Tạo tài khoản tại địa chỉ <https://hub.docker.com/> sau đó đăng nhập như sau



Hình 7.8: Đăng nhập tài khoản Docker

### Tạo Repository trên Docker Hub

Trên Docker Hub tạo 2 repos để chứa 2 Docker images tương ứng dành cho Backend và Frontend.



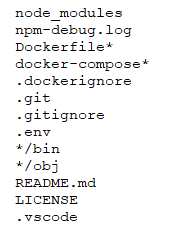
Hình 7.9: Tạo repository trên Docker Hub

### Thêm Docker vào FrontEnd

Tại thư mục root của FrontEnd, tạo mới các file sau

#### File .dockerignore

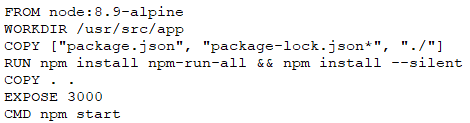
File này để loại bỏ một số file hoặc thư mục không cần thiết khi tạo Docker Image.



Hình 7.10: Nội dung file .dockerignore

#### File DockerFile

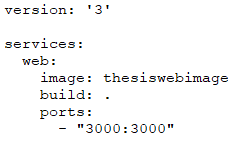
File này chứa những lệnh để tạo ra được một Docker Image



Hình 7.11: Nội dung file DockerFile

#### File docker-compose.yml

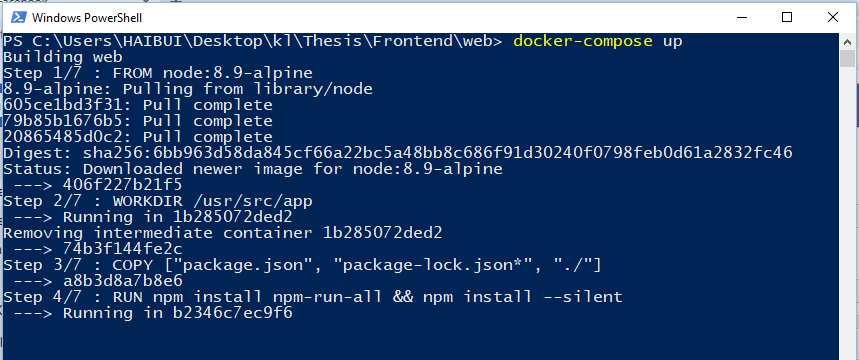
File này dùng để build Docker Image, chạy một hoặc nhiều Docker Container cùng một lúc.



Hình 7.12: Nội dung file docker-compose.yml

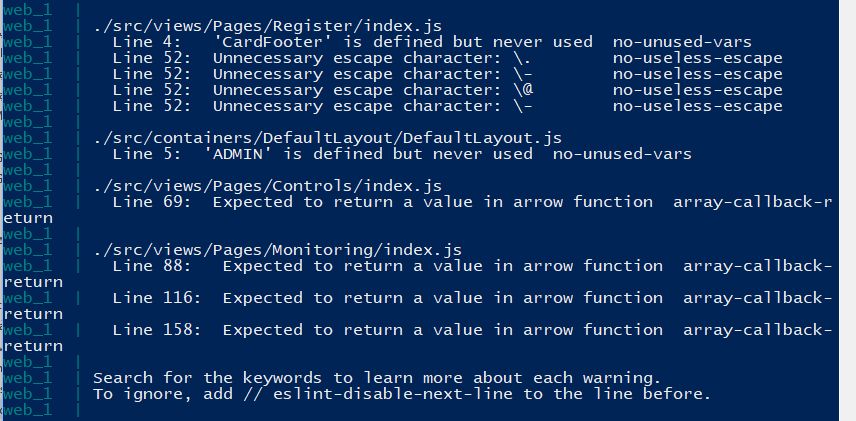
#### Tạo và push Docker image

Vào thư mục root, mở Powershell và chạy lệnh “docker-compose up”.



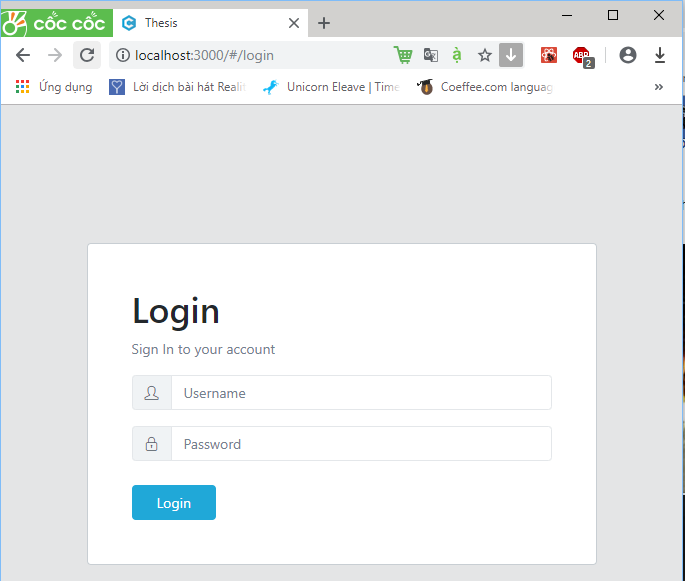
Hình 7.13: Chạy lệnh docker-compose up

Kết quả sau khi chạy xong.



Hình 7.14: Kết quả chạy docker-compose

Truy cập vào đường dẫn localhost:3000 kiểm tra kết quả.



Hình 7.15: Kiểm tra kết quả trên localhost

Như vậy Dockerfile đã chạy thành công, ta tiến hành push Docker image vừa tạo ra lên Docker Hub.

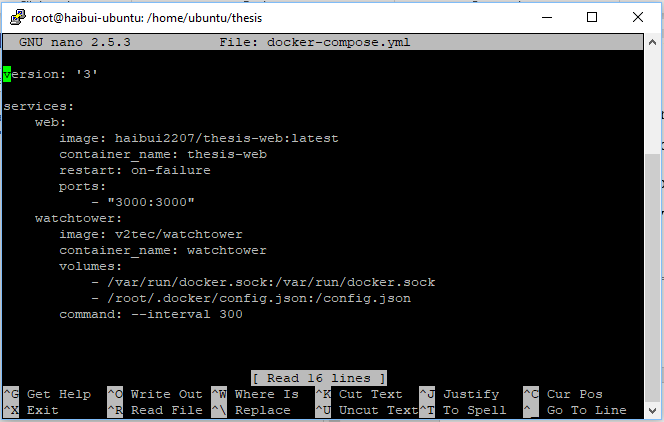
Tiến hành push Docker image lên Docker Hub như sau



Hình 7.16: Lệnh push Docker image lên Docker Hub

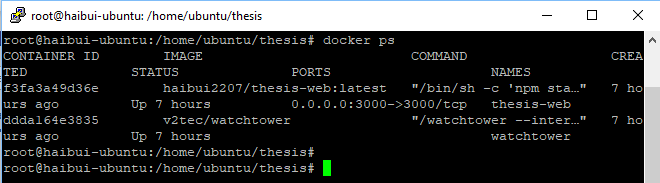
Như vậy tại repo “haibui2207/thesis-web” đã có Docker image. Bây giờ có thể sử dụng repo này để chạy tại bất cứ máy nào có cài Docker.

Truy cập vào máy ảo OpenStack để kiếm tra. Tạo một file docker-compose.yml để tiện cho việc chạy các Docker Container khác sau này.



Hình 7.17: Nội dụng file docker-compose.yml trên máy ảo

Sau đó ta chạy lệnh “docker-compose up” như trên. Kiểm tra Docker Container đang chạy bằng lệnh “docker ps”.



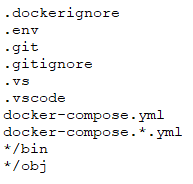
Hình 7.18: Kết quả chạy Docker image web trên OpenStack

Kiểm tra bằng cách truy cập vào địa chỉ máy ảo với port 3000 hoặc sử dụng ngrok để truy cập vào từ bên ngoài.

### Thêm Docker vào ASP.NET Core

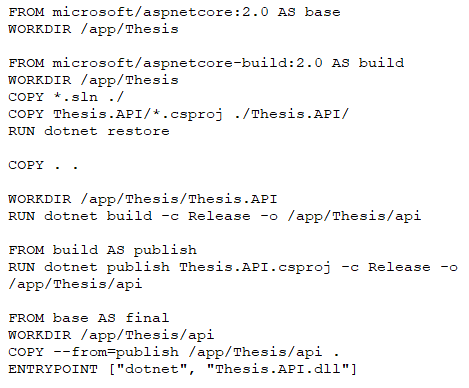
Trong thư mục Thesis tạo thêm các file sau

#### File .dockerignore



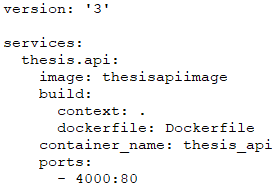
Hình 7.19: Nội dung file .dockerignore

#### File Dockerfile



Hình 7.20: Nội dung file Dockerfile

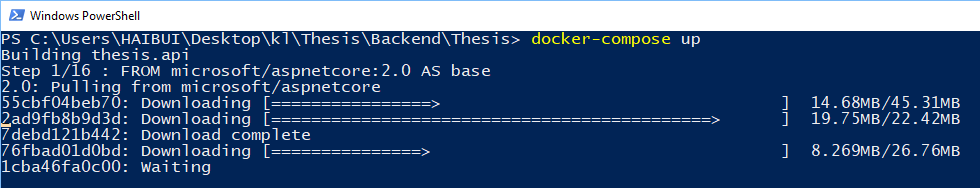
#### File docker-compose.yml



Hình 7.21: Nội dung file docker-compose.yml

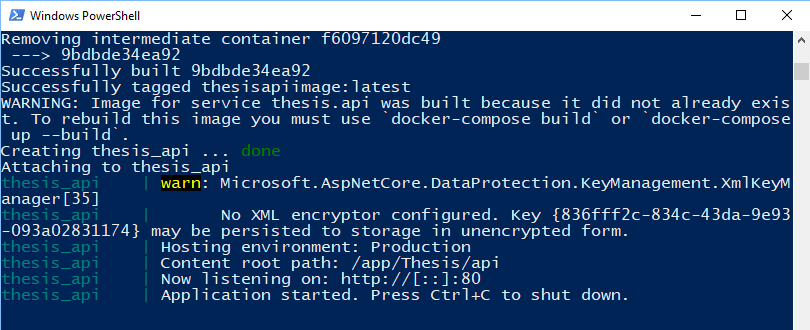
#### Tạo và push Docker image

Tại thư mục hiện tại, mở powershell và chạy lệnh “docker-compose up”.



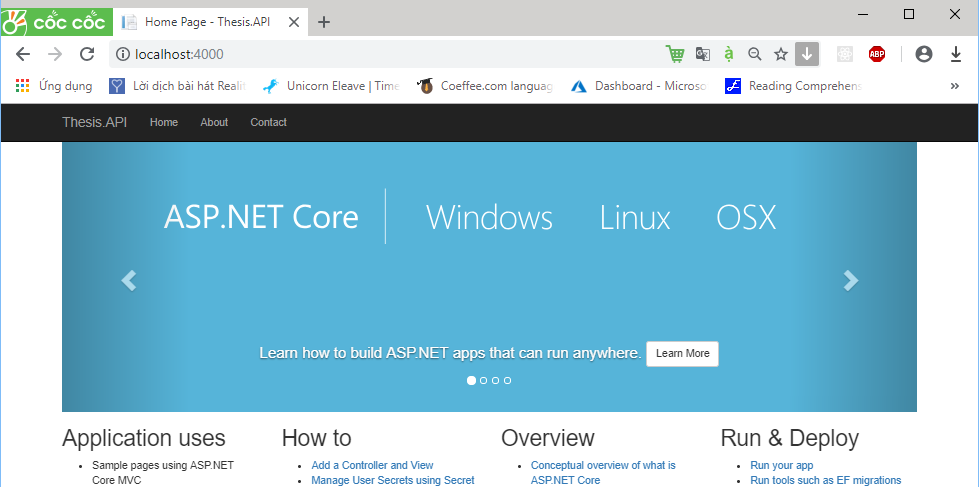
Hình 7.22: Chạy lệnh docker-compose up

Kết quả sau khi chạy xong.



Hình 7.23: Kết quả sau khi chạy docker-compose up

Truy cập vào đường dẫn localhost:4000 để kiểm tra kết quả.



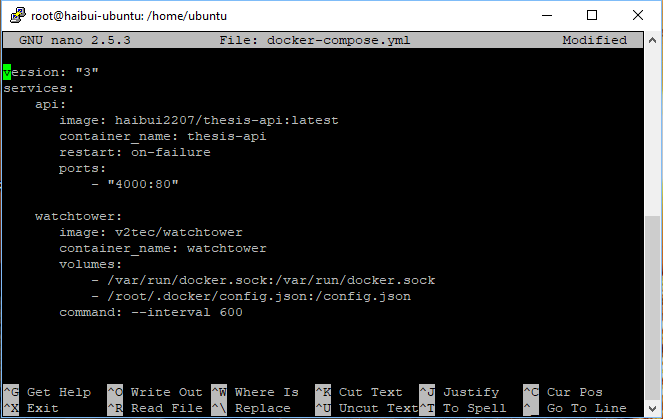
Hình 7.24: Kiểm tra kết quả trên localhost

Tiếp theo tiến hành push Docker image lên Docker Hub sử dụng các lệnh sau.



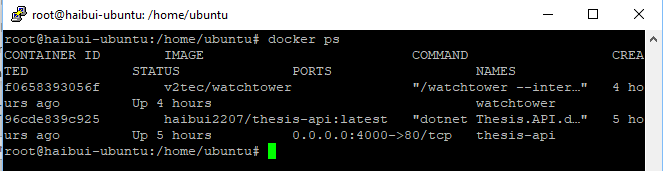
Hình 7.25: Lệnh push Docker image lên Docker Hub

Truy cập vào máy ảo OpenStack để kiếm tra. Tạo một file docker-compose.yml có nội dung như sau.



Hình 7.26: Nội dung file docker-compose.yml trên máy ảo

Sau đó ta chạy lệnh “docker-compose up” như trên. Kiểm tra Docker Container đang chạy bằng lệnh “docker ps”.



Hình 7.27: Kết quả chạy Docker image web trên OpenStack

# KẾT QUẢ THỰC HIỆN

## Mức độ delay

Trong quá trình triển khai server, sử dụng phần mềm Ngrok để tạo đường truyền ra ngoài Internet cho phép những kết nối bên ngoài Internet có thể truy cập vào localhost của máy ảo.

Nhưng Ngrok hạn chế về số lượng kết nối (tối đa 40 kết nối trong 1 giây), để khắc phục điều này trong phần code của các thiết bị có thiết lập độ trễ (delay) vì thế thời gian realtime sẽ bị trễ theo.

Dưới đây là bảng mô tả các thông số trung bình đo được sau 20 lần đo với các độ trễ khác nhau.

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Độ trễ thiết lập** | **Thời gian phản hồi** | **Ghi chú** |
| Web Frontend | không | ~ < 810ms | Tùy vào từng API có tốc độ phản hồi khác nhau |
| Arduino | 2000ms | ~ < 2800ms | Độ trễ lớn |
| 1500ms | ~ < 2200ms | Độ trễ phù hợp trong giới hạn của Ngrok |
| 1000ms | ~ < 1650ms | Vượt quá giới hạn số lượng kết nối của Ngrok |
| 500ms | ~ < 1100ms |
| không | ~ < 870ms |
| Đầu đọc RFID | không | ~ 2500ms | Thời gian chờ đọc thẻ chậm |

Bảng 8.1: Bảng mô tả thời gian phản hồi trung bình của thiết bị

## Đánh giá kết quả

Thiết lập độ trễ để áp ứng giới hạn số lượng kết nối của Ngrok làm đường truyển trở ổn định với một số lượng kết nối vừa phải trong một khoản thời gian nhất định nhưng lại làm tăng thời gian phản hồi cho các thiết bị.

Điều này hoàn toàn có thể giải quyết được bằng cách sử dụng một hosting trả phí và loại bỏ độ trễ trên các thiết bị.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết quả đạt được

Qua thời gian thực hiện, dưới sự hướng dẫn tận tình của thầy PGS.TS Lê Trung Quân, em đã cố gắng hoàn thành khóa luận trong thời gian quy định. Trong khóa luận đã hoàn thành những công việc như sau:

* Hoàn thành việc chuyển đổi giữa việc sử dụng một Web Server (Node-RED) khó tùy biến sang một môi trường tự hành, dễ dàng mở rộng, thêm tính năng, thay đổi giao diện, tùy biến, …
* Hoàn thành 3 trên 4 mục tiêu đã đề ra trong đề cương chi tiết.

## Hạn chế

Mặc dù đã rất cố gắng nhưng do kiến thức và thời gian có hạn nên em vẫn chưa hoàn thành được mục tiêu cuối cùng trong đề cương chi tiết – Xây dựng hệ thống nhận diện khuôn mặt trên Raspberry bằng thư viên OpenCV.

Ngoài ra trong quá trình làm khóa luận không tránh khỏi những thiếu sót và nhầm lẫn khác. Kính mong quý thầy cô và các bạn có thể góp ý để khóa luận này trở nên tốt hơn.

## Hướng phát triển

SmartHouses vẫn đang là một xu thế của IoT mà trong đó có rất nhiều khía cạnh để phát triển. Thông qua việc thực hiện đề tài “Phát triển giải pháp giám sát nhà thông minh thời gian thực dựa trên nền tảng điện toán đám mây” đã phần nào xây dựng được cấu trúc cơ bản của một mô hình IoT hoàn chỉnh.

Với những kết quả ở hiện tại cùng với việc xây dựng nền tảng, môi trường mới nhắm đến việc dễ dàng mở rộng, mục tiêu phát triển tiếp đến của khóa luận là tích hợp nhận diện khuôn mặt và máy học (Machine learning) vào hệ thống có sẵn, đồng thời thêm nhiều thiết bị cảm biến để giúp nhận diện sự thay đổi môi trường xung quanh một cách tốt hơn.

TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | T. Anscombe, "IoT and Privacy by design in the smart home," Tony Anscombe, 2018. |
| [2] | L. Balling, "SMART HOME," in *Computer Engineering final project 2012*, Levi Balling, 2012. |
| [3] | H. v. d. H. &. P. Brils, "Smart Homes," in *Dutch Expert Centre on Smart & Sustainable Living*, 2014. |
| [4] | S. Das, "Technology for SMART HOME," Springer India, 2013. |
| [5] | A. Gould, "Internet of Things for the Smart Home," in *Internet of Things for the Smart Home*, Vice President, Sensinode Business, 2013, pp. 1-17. |
| [6] | P. Rust, "Applications for the Internet of Things," in *Applications for the Internet of Things*, MINES Saint-Étienne, CNRS, Orange Labs, 2017, pp. 23-31. |
| [7] | R. H. (ed.), "Inside the smart home," Springer-Verlag London, UK, 2003. |
| [8] | TuanPM, "Lập trình IoT với ESP8266 Arduino," 2017. [Online]. Available: https://arduino.esp8266.vn/. |

1. BKAV: Một tập đoàn công nghệ lớn tại Việt Nam [↑](#footnote-ref-1)
2. Google: Trang tìm kiếm thông tin lớn nhất thế giới [↑](#footnote-ref-2)
3. Stack Overflow: Trang hỏi đáp lập trình lớn nhất thế giới [↑](#footnote-ref-3)
4. DHT11: <https://vngiotlab.github.io/vbluno/vi/mydoc_arduino_tut10_vi.html> [↑](#footnote-ref-4)
5. SQLite: <https://www.sqlite.org/index.html> [↑](#footnote-ref-5)