

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG
BỘ MÔN MÁY TÍNH – HỆ THỐNG NHÚNG**



**NGUYỄN TRIỆU THIÊN HÀO
MSSV: 1720081**

Đề tài:

**ỨNG DỤNG PRIVATE BLOCKCHAIN NETWORK
VÀO HỆ THỐNG CẬP NHẬT FIRMWARE TỪ XA
CHO THIẾT BỊ IOT**

**KHOÁ LUẬN TỐT NGHIỆP CỬ NHÂN
NGÀNH KỸ THUẬT ĐIỆN TỬ - TRUYỀN THÔNG
CHUYÊN NGÀNH MÁY TÍNH - HỆ THỐNG NHÚNG**

**NGƯỜI HƯỚNG DẪN KHOA HỌC
TS. HUỲNH HỮU THUẬN**

TP. Hồ Chí Minh, tháng 7 năm 2021

LỜI CẢM ƠN

Để hoàn thành tốt khóa luận tốt nghiệp cử nhân này, ngoài sự nỗ lực của bản thân, em còn nhận được sự quan tâm giúp đỡ của nhiều tập thể và cá nhân.

Lời đầu tiên em xin bày tỏ lòng biết ơn sâu sắc tới Giáo viên hướng dẫn Tiến sĩ thầy Huỳnh Hữu Thuận, người đã tận tâm hướng dẫn em, giúp em có định hướng chính xác về đề tài, nhắc nhở và động viên em trong suốt quá trình thực hiện khóa luận.

Tiếp theo đó, em xin gửi lời tri ân đến các thầy cô của trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh nói chung và các thầy cô khoa Điện tử - Viễn thông nói riêng. Các thầy cô không ngừng giảng dạy và truyền đạt những kiến thức quý báu cho nhiều thế hệ sinh viên và trong đây có em. Đó chính là nền tảng và cơ sở kiến thức để em thực hiện đề tài này.

Cuối cùng, em xin gửi lời cảm ơn đến những anh chị đi trước đã chia sẻ kiến thức và kinh nghiệm, cảm ơn gia đình, những người bạn trong khoa Điện tử - Viễn thông đã ủng hộ và động viên em trong suốt quá trình thực hiện khóa luận tốt nghiệp này.

Thành phố Hồ Chí Minh, tháng 07 năm 2021

Sinh viên thực hiện

Nguyễn Triệu Thiên Hào

TÓM TẮT KHÓA LUẬN

Cập nhật firmware từ xa (FOTA) là một phương pháp cập nhật khá phổ biến cho các thiết bị IoT. Phương pháp này tuy mang lại sự tiện lợi cho việc cập nhật firmware nhưng bên cạnh đó cũng mở ra nhiều vấn đề về bảo mật trong quá trình cập nhật. Quá trình có thể bị tấn công bởi các tác nhân vật lý mạng trong khi vận chuyển chương trình firmware mới đến các thiết bị.

Private Blockchain Network với hợp đồng thông minh (Smart Contract) có thể hỗ trợ bảo vệ tính toàn vẹn của chương trình firmware trong việc lưu trữ cũng như truy vấn khi cần cập nhật trên các thiết bị IoT thông qua các giao thức mạng.

Khóa luận này tập trung nghiên cứu tích hợp mạng Blockchain với hợp đồng thông minh vào việc cập nhật firmware từ xa cho các thiết bị IoT. Mục tiêu của nghiên cứu là thiết kế một hệ thống cung cấp dịch vụ cập nhật firmware từ xa cho các thiết bị IoT. Kết quả thực nghiệm cho thấy việc tích hợp trong nghiên cứu giúp cho việc cập nhật cũng như việc lưu trữ thông qua cơ sở dữ liệu của mạng blockchain được tăng tính bảo mật hơn.

MỤC LỤC

LỜI CẢM ƠN.....	II
TÓM TẮT KHÓA LUẬN.....	III
MỤC LỤC	IV
DANH MỤC TỪ VIẾT TẮT	VII
DANH MỤC HÌNH ẢNH.....	VIII
DANH MỤC BẢNG BIỂU.....	X
TỔNG QUAN ĐỒ ÁN.....	1
CHƯƠNG 1 TỔNG QUAN PRIVATE BLOCKCHAIN NETWORK.....	3
1.1 GIỚI THIỆU VỀ PRIVATE BLOCKCHAIN.....	3
1.2 KIẾN TRÚC ĐƠN GIẢN CỦA MỘT MẠNG HYPERLEDGER FABRIC.....	5
CHƯƠNG 2 TỔNG QUAN HỆ THỐNG.....	7
2.1 KIẾN TRÚC TỔNG QUAN	7
2.1.1 Hardware Layer and Protocol Layer	7
2.1.2 Proxy Layer and Blockchain Layer.....	8
2.1.3 Service Layer and Application Layer.....	8
2.2 QUÁ TRÌNH XÁC THỰC YÊU CẦU CẬP NHẬT FIRMWARE	9
CHƯƠNG 3 TRIỂN KHAI XÂY DỰNG HỆ THỐNG	11
3.1 MÔI TRƯỜNG PHÁT TRIỂN	11
3.1.1 Điều kiện tiên quyết cho thiết bị IoT.....	11
3.1.2 Điều kiện tiên quyết cho Vendor Service.....	12
3.1.3 Điều kiện tiên quyết cho Blockchain Network	13
3.2 KỊCH BẢN HỆ THỐNG.....	16
3.3 KIẾN TRÚC CƠ BẢN CỦA HỆ THỐNG INTERNET OF THINGS (IoT).....	17
3.4 KIẾN TRÚC MÔ HÌNH HỆ THỐNG	19
3.5 PHÁT TRIỂN SMART CONTRACT (HỢP ĐỒNG THÔNG MINH).....	20
3.5.1 Xác định các cấu trúc dữ liệu trong Smart Contract	21

3.5.2	Phát triển các logic giao dịch trong Smart Contract.....	22
3.5.3	Logic giao dịch push	22
3.5.4	Logic giao dịch verify	23
3.5.5	Logic giao dịch query.....	23
3.6	XÂY DỰNG CƠ SỞ HẠ TẦNG MẠNG	23
3.6.1	Xác định kiến trúc mạng.....	24
3.6.2	Giới thiệu ngôn ngữ dòng lệnh Sh (Shell Command Language)	25
3.6.3	Khởi tạo các thành phần cơ bản	26
3.6.4	Triển khai mạng.....	28
3.6.5	Cài đặt và khởi tạo Smart Contract	29
3.7	TRIỂN KHAI DỊCH VỤ WEB SERVER (VENDOR SERVICE).....	30
3.7.1	Node SDK	31
3.7.2	Giới thiệu giao thức HTTP.....	32
3.7.3	Mô hình tương tác giữa Vendor Service và Smart Contract	33
3.7.4	Các thành phần chính trong Vendor Service.....	34
3.7.5	Xử lý giao dịch trong Vendor Service.....	38
3.7.6	Các API trong Vendor Service	39
3.8	LẬP TRÌNH NHÚNG CHO THIẾT BỊ IoT.....	40
3.8.1	Giới thiệu về lập trình nhúng.....	40
3.8.2	Giới thiệu các thư viện hỗ trợ.....	40
3.8.3	Dịch vụ cập nhật OTA trên thiết bị NodeMCU ESP32	41
3.8.4	Quá trình truy vấn API và sử dụng thuật toán băm.....	41
3.9	QUÁ TRÌNH XÁC THỰC YÊU CẦU CẬP NHẬT FIRMWARE	42
CHƯƠNG 4	KẾT QUẢ THỰC HIỆN.....	48
4.1	MỤC TIÊU ĐỀ RA CHO KHÓA LUẬN VÀ KẾT QUẢ ĐẠT ĐƯỢC SAU KHI THỰC HIỆN... ..	48
4.2	KẾT QUẢ KHI ĐĂNG NHẬP VÀO HỆ THỐNG	50
4.2.1	Đăng kí tài khoản trên hệ thống	50
4.2.2	Đăng nhập tài khoản trên hệ thống.....	52
4.3	KẾT QUẢ KHI THỰC HIỆN YÊU CẦU CẬP NHẬT FIRMWARE TRÊN HỆ THỐNG.....	52
4.3.1	Đăng kí các thành phần trên hệ thống	52
4.3.2	Thực hiện yêu cầu cập nhật firmware trên hệ thống	55
CHƯƠNG 5	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	59

5.1 KẾT LUẬN.....	59
5.2 HƯỚNG PHÁT TRIỂN	59
TÀI LIỆU THAM KHẢO	61

DANH MỤC TỪ VIẾT TẮT

Kí hiệu chữ viết tắt	Chữ viết đầy đủ
API	Application Programming Interface
CA	Certificate Authority
CLI	Command Line Interface
FOTA	firmware-on-the-air
HF	Hyperledger Fabric
HTTP/HTTPS	HyperText Transfer Protocol / Secure
IBM	International Business Machines
IDE	Integrated Development Environment
IoT	Internet of Things
MAC	Media Access Control
NPM	Node Package Manager
SDK	Software Development Kit
SHA	Secure Hash Algorithms
TXID	Transaction ID

DANH MỤC HÌNH ẢNH

Hình 1.1 Kiến trúc đơn giản của một mạng Hyperledger Fabric	5
Hình 2.1 Quá trình xác thực cập nhật firmware	9
Hình 3.1 Thiết bị IoT – NodeMCU ESP32	11
Hình 3.2 Kịch bản hệ thống.....	16
Hình 3.3 Cấu trúc hệ thống IoT	18
Hình 3.4 Kiến trúc mô hình hệ thống	19
Hình 3.5 Cấu trúc của một Ledger (sổ cái)	21
Hình 3.6 Kiến trúc cơ sở hạ tầng mạng.....	24
Hình 3.7 Cấu trúc thư mục <i>crypto-config</i>	27
Hình 3.8 Kết quả trên màn hình console khi bắt đầu hoạt động mạng	28
Hình 3.9 Kết quả trên Docker Engine khi bắt đầu hoạt động mạng	28
Hình 3.10 Kết quả trên Docker Engine khi khởi tạo <i>fota</i> cho <i>peer0.org1</i>	30
Hình 3.11 Giao diện các databases trên CouchDB	30
Hình 3.12 Mô hình tương tác giữa Application với Blockchain Network.....	31
Hình 3.13 Cấu trúc cơ bản của một ứng dụng web	32
Hình 3.14 Mô hình tương tác giữa Vendor Service với FotaContract	33
Hình 3.15 Cấu trúc thư mục Wallet.....	35
Hình 3.16 Giao diện đăng nhập (Login).....	36
Hình 3.17 Giao diện đăng ký (Signin).....	37
Hình 3.18 Giao diện chính (Home)	38
Hình 3.19 Quá trình xử lý yêu cầu cập nhật firmware ở thiết bị IoT.....	43
Hình 3.20 Cập nhật thuộc tính <i>status</i> của cấu trúc dữ liệu <i>UpdateOTA</i>	43
Hình 4.1 Thông tin định danh admin trong wallet	50
Hình 4.2 Đăng ký tài khoản Manager với ID 1720081	51
Hình 4.3 thông tin định danh của Manager ID 1720081 trong wallet	51
Hình 4.4 Cơ sở dữ liệu World State trên CouchDB với đối tượng Manager 1720081	51
Hình 4.5 Thông tin chi tiết đối tượng <i>Manager</i> 1720081 trên CouchDB.....	52
Hình 4.6 Đăng nhập tài khoản Manager với ID 1720081	52
Hình 4.7 Đăng tải file binary firmware blockchain-fota0	53
Hình 4.8 Thông tin chi tiết đối tượng <i>FirmwareOTA</i> blockchain-fota0 trên CouchDB... ..	53

Hình 4.9 Đăng tải file binary firmware blockchain-fota1	53
Hình 4.10 Thông tin chi tiết đối tượng <i>FirmwareOTA</i> blockchain-fota1 trên CouchDB.	54
Hình 4.11 Hiện thị danh sách firmware.....	54
Hình 4.12 Tạo hệ thống thiết bị IoT	54
Hình 4.13 Thông tin chi tiết đối tượng <i>IoTSystem</i> trên CouchDB	55
Hình 4.14 Tạo yêu cầu cập nhật firmware từ xa cho thiết bị IoT	55
Hình 4.15 Thông tin chi tiết đối tượng <i>UpdateFOTA</i> trên CouchDB.....	56
Hình 4.16 Console khi chưa có yêu cầu cập nhật firmware	56
Hình 4.17 Console khi có yêu cầu cập nhật	56
Hình 4.18 Console khi tiến hành quá trình xác thực yêu cầu cập nhật	57
Hình 4.19 Console khi tiến hành quá trình cài đặt firmware	57
Hình 4.20 Console khi tiến hành ghi nhận kết quả yêu cầu cập nhật.....	57
Hình 4.21 Console khi khởi động lại thiết bị	58

DANH MỤC BẢNG BIỂU

Bảng 2.1 Kiến trúc phân lớp của hệ thống	7
Bảng 3.1 Các API chính trong Vendor Service.....	39
Bảng 3.2 Thông tin chung của API checkRequire	44
Bảng 3.3 Thông tin chung của API verifyRequire	45
Bảng 3.4 Thông tin chung của API recordRequire	46
Bảng 4.1 Mục tiêu đề ra và kết quả đạt được	50

TỔNG QUAN ĐỒ ÁN

1. Đặt vấn đề

Các thiết bị IoT được điều khiển và giám sát thông qua chương trình firmware được nhúng trên thiết bị. Firmware là phần mềm được viết cho các hệ thống nhúng và được lưu trữ trong các chip bộ nhớ ROM (Read Only Memory) nếu firmware đơn giản hoặc những firmware phức tạp hơn thường lưu trữ ở bộ nhớ flash để có thể cập nhật. Việc cập nhật firmware được xảy ra định kỳ vì các mục đích như tăng cường bảo mật, thêm vào các tính năng mới, sửa các lỗi tồn tại trong firmware cũ cũng như áp dụng các giao thức mới cho thiết bị. Do tính đặc trưng của các thiết bị IoT chính là giao tiếp thông qua các giao thức mạng trên mạng Internet và sự bất tiện khi cập nhật thông qua USB nên các thiết bị thường hỗ trợ cập nhật thông qua các giao thức mạng vô tuyến (FOTA updates – cập nhật firmware từ xa). Phương pháp cập nhật này mang lại các lợi ích như: không cần đến các giao tiếp vật lý khi cập nhật, tiện lợi và hiệu quả kinh tế. Nhưng sự phát triển không ngừng của hệ thống IoT đã dẫn đến số lượng của các thiết bị IoT nhanh chóng tăng lên. Khi ấy việc quản lý các thiết bị và chứng thực cho các yêu cầu cập nhật sẽ trở nên khó khăn và thiếu tính bảo mật. Tiềm năng tấn công các thiết bị mà không cần thông qua các giao tiếp vật lý là rất lớn.

2. Mục tiêu đề tài

Xây dựng một hệ thống cung cấp dịch vụ cập nhật firmware từ xa thông qua giao thức mạng cho các thiết bị IoT trên mạng Internet được ứng dụng Private Blockchain Network với hợp đồng thông minh để tăng tính bảo mật. Nhờ việc sử dụng nền tảng dữ liệu phân tán ở các nút trong mạng blockchain để đảm bảo tính toàn vẹn cũng như tính khả dụng của dữ liệu được lưu trữ (ở đây chính là các chương trình firmware) trong cơ sở dữ liệu của mạng blockchain.

Với việc hạn chế về thời gian cũng như những thiếu sót về kiến thức trong tầm hiểu biết của một sinh viên nên khóa luận hướng đến những mục tiêu sau:

- Tìm hiểu về quá trình cập nhật firmware từ xa.
- Tìm hiểu về Private Blockchain Network dựa trên nền tảng Hyperledger Fabric:

- Tìm hiểu về ngôn ngữ dòng lệnh Shell để xây dựng mạng.
- Tìm hiểu về nền tảng Docker để triển khai mạng.
- Tìm hiểu về lập trình web với ngôn ngữ Javascript (Node Js) để triển khai hệ thống cung cấp dịch vụ cập nhật firmware từ xa.
- Tìm hiểu về lập trình nhúng với ngôn ngữ C++ để viết chương trình firmware cho thiết bị IoT.

3. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu: cập nhật firmware từ xa cho thiết bị IoT ESP32 thông qua giao thức mạng WiFi để tham gia vào mạng Internet thông qua Private Blockchain Network.

Phạm vi nghiên cứu: nghiên cứu được thực hiện ở mức độ cơ bản, hệ thống đơn giản theo mô hình hệ thống IoT đơn giản gồm một web server để cung cấp dịch vụ cập nhật và client là thiết bị IoT thông qua nền tảng mạng Hyperledger Fabric.

4. Ý nghĩa thực tiễn của đề tài

Cung cấp một giải pháp cho việc quản lý và tăng tính bảo mật của các thiết bị IoT cũng như quá trình cập nhật firmware trong các dự án thành phố thông minh.

5. Nội dung của khóa luận

Nội dung trong khóa luận được chia thành chương:

- Chương 1: Tổng quan Private Blockchain Network.
- Chương 2: Tổng quan hệ thống.
- Chương 3: Triển khai xây dựng hệ thống.
- Chương 4: Kết quả thực hiện.

CHƯƠNG 1 TỔNG QUAN PRIVATE BLOCKCHAIN NETWORK

Blockchain (chuỗi khối), là một cơ sở dữ liệu phân cấp lưu trữ thông tin trong các khối thông tin được liên kết với nhau bằng mã hóa và mở rộng theo thời gian. Mỗi khối thông tin đều chứa thông tin về thời gian khởi tạo và được liên kết tới khối trước đó, kèm một mã thời gian và dữ liệu giao dịch. Blockchain được thiết kế để chống lại việc thay đổi của dữ liệu: Một khi dữ liệu đã được mạng lưới chấp nhận thì sẽ không có cách nào thay đổi được nó. Blockchain được đảm bảo nhờ cách thiết kế sử dụng hệ thống tính toán phân cấp với khả năng chịu lỗi byzantine cao. Do đó sự đồng thuận phân cấp có thể đạt được nhờ Blockchain. Vì vậy Blockchain phù hợp để ghi lại những sự kiện, hồ sơ y tế, xử lý giao dịch, công chứng, danh tính và chứng minh nguồn gốc. Việc này có tiềm năng giúp xóa bỏ các hậu quả lớn khi dữ liệu bị thay đổi trong bối cảnh thương mại toàn cầu.

Cơ bản có hai loại kiến trúc Blockchain: công khai (public) và riêng tư (private). Public Blockchain như Bitcoin và Ethereum, cho phép mọi người tham gia mạng một cách tự do được ẩn danh thông qua việc sử dụng các thuật toán đồng thuận như PoW (Proof of Work), PoS (Proof of Stake) để bảo mật tốt hơn. Private Blockchain như Hyperledger, sử dụng các CAs (Certificate Authorities – các tổ chức phát hành chứng chỉ) để xác thực những thành viên tham gia mạng thông qua cặp khóa bảo mật để xác định danh tính.

Public Blockchain là một mạng cho phép bất kì ai tham gia, tương tác với mạng và truy vấn dữ liệu mà không cần cấp quyền (permissionless). Điều này khiến mọi số liệu lưu trữ bị hiển thị một cách công khai khiến giảm đi tính riêng tư cho các Tổ chức khác nhau. Đó là lý do nghiên cứu này tập trung vào việc sử dụng Private Blockchain thay vì Public Blockchain.

1.1 Giới thiệu về Private Blockchain

Private Blockchain vẫn mang những đặc điểm chung như:

- Một sổ cái phân tán (distributed ledger) chỉ cho phép bổ sung – Blockchain có cấu trúc chuỗi khối, trong đó mỗi khối được liên kết đến khối trước đó. Nên thực chất sổ cái như tập hợp các khối trong chuỗi với mỗi khối chứa các giao dịch được ghi nhận trong sổ cái.

- Một mạng lưới ngang hàng (peer-to-peer) – mọi thành viên tham gia vào mạng giữ một bản sao của Blockchain. Những thành viên này được gọi là các nút (peer/node) và họ tương tác theo kiểu ngang hàng.
- Một cơ chế đồng thuận – các nút đạt được sự đồng thuận về tính chính xác của các giao dịch được truyền trên mạng dựa trên một cơ chế, điều này đảm bảo rằng không có dữ liệu sai trái nào được ghi vào chuỗi.

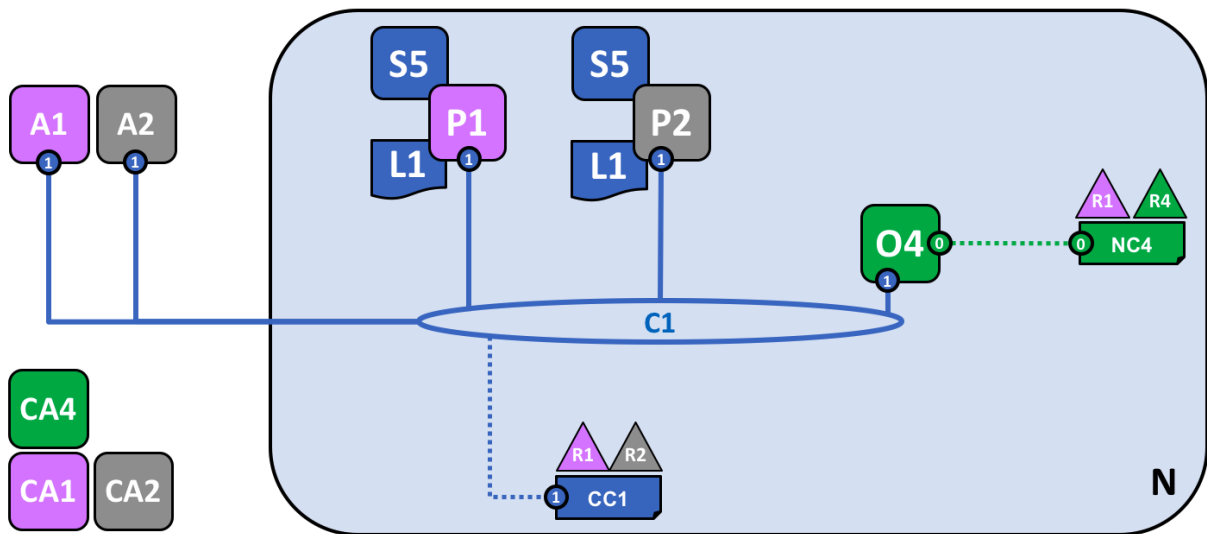
Trái ngược hoàn toàn với tính chất không cần được cấp quyền (permissionless) của các Public Blockchain, các Private Blockchain đặt ra các quy tắc về việc ai có thể tham gia và ghi dữ liệu vào chuỗi, chúng là những môi trường cần được cấp quyền (permissioned). Chúng không phải là hệ thống phi tập trung, vì có một hệ thống phân cấp rõ ràng xét về mặt kiểm soát. Tuy nhiên, chúng là các mạng phân tán, trong đó nhiều nút vẫn duy trì một bản sao của chuỗi trên máy tính của họ.

Private Blockchain phù hợp cho việc thiết lập doanh nghiệp, trong đó một tổ chức muốn thừa hưởng các thuộc tính của Blockchain mà vẫn có thể bảo vệ mạng của họ không bị những người bên ngoài truy cập.

Private Blockchain sử dụng các CAs (Certificate Authorities) là các tổ chức phát hành chứng chỉ được chỉ định để đảm nhận các chức năng nhất định cho việc phát hành và xác thực danh tính của người dùng hoặc các nút của tổ chức tương ứng. Đồng thời các nút Orderer sẽ tham gia vào quá trình đồng thuận để xác thực các giao dịch thay vì tất cả các nút của mạng đều tham gia vào quá trình này như Public Blockchain. Ngoài ra, các Smart Contract (hợp đồng thông minh) giúp hỗ trợ việc truy cập vào sổ cái của mạng, qua đó thành viên trong mạng sẽ thực hiện các giao dịch.

Hyperledger Fabric là một framework (nền tảng) để xây dựng Private Blockchain, một trong 5 framework về Blockchain nằm trong chiến lược Hyperledger Umbrella của Linux Foundation gồm: Hyperledger Indy, Hyperledger Fabric, Hyperledger Iroha, Hyperledger Sawtooth, Hyperledger Burror. Điều đặc biệt là Hyperledger Fabric được đóng góp bởi công ty IBM. Hyperledger Fabric có modularity (tính mô đun) khá cao nên nó cho phép các Doanh nghiệp dễ dàng plug and play để xây dựng một ứng dụng Private Blockchain phù hợp các yêu cầu nghiệp vụ của mình [1]. Đến đây, ta có thể hiểu rằng một mạng Hyperledger Fabric là một mạng Private Blockchain.

1.2 Kiến trúc đơn giản của một mạng Hyperledger Fabric



Hình 1.1 Kiến trúc đơn giản của một mạng Hyperledger Fabric

Các thành phần trong mạng bao gồm [2]:

- **N**: Network – Mạng.
- **NC**: Network Configuration – Cấu hình của mạng.
- **C**: Channel – Kênh, tập hợp các tổ chức có vai trò nhất định trong cùng một quy trình nào đó.
- **CC**: Channel Configuration – Cấu hình của kênh.
- **R**: Organization – Tổ chức.
- **O**: Orderer Node (Ordering Service): có chức năng như một nút quản trị cho mạng, tham gia vào quá trình đồng thuận trong mạng.
- **S**: Smart Contract (Chaincode) – Hợp đồng thông minh, được cài đặt trên kênh, định nghĩa rõ các cấu trúc dữ liệu (đối tượng), các hành động mà người dùng có thể thực hiện để tương tác trạng thái của đối tượng được lưu trong sổ cái qua đó tạo nên các giao dịch trong mạng.
- **P**: Peer node – Nút, là điểm tương tác giữa các thành viên trong tổ chức tương ứng với kênh, mọi hành động của người dùng đều phải đi qua peer. Các peer node phải được tham gia vào một kênh đã thiết lập. Mỗi peer node giữ một bản sao của sổ cái.

- **L:** Ledger – Sổ cái, lưu trữ trạng thái của các đối tượng.
- **CA:** Certificate Authority, phát hành identity (định danh) cho người dùng hoặc node của tổ chức tương ứng. Các identity sẽ được lưu trữ trong cấu trúc MSP (Membership Service Provider) của tổ chức tương ứng.
- **A:** Application, ứng dụng hay giao diện (web, mobile app) giúp người dùng tương tác với hệ thống dễ dàng hơn.

CHƯƠNG 2 TỔNG QUAN HỆ THỐNG

Giải pháp cho hệ thống cung cấp dịch vụ cập nhật firmware từ xa cho các thiết bị IoT tập trung vào một kiến trúc mang tính phân tán và phân quyền dựa trên các lớp, các module có chức năng khác nhau thông qua việc triển khai nền tảng Blockchain Network để chứng thực quá trình cập nhật firmware. Ở chương này, kiến trúc tổng quan của hệ thống sẽ được mô tả và thiết kế dựa theo tài liệu “Securing Over-The-Air IoT Firmware Updates using Blockchain” [3].

2.1 Kiến trúc tổng quan

Kiến trúc phân lớp của hệ thống bao gồm sáu lớp: lớp hardware (hardware layer), lớp giao thức (protocol layer), lớp proxy (proxy layer), lớp mạng blockchain (blockchain layer), lớp cung cấp dịch vụ (service layer), và lớp ứng dụng (application layer) mô tả theo bảng 2.1.

Layer		Description
Application Layer		Applications
Service Layer		Vendors
Blockchain Layer	Smart Contract (SubLayer)	Incident Handling
		Firmware Update Verification
	Infrastructure (SubLayer)	Consensus Protocols
		Distributed Immutable Ledgers
		Certificate Authorities (CA)
Proxy Layer		Transparent Gateway
Protocol Layer	Security (SubLayer)	TLS/DTLS
	Communication (SubLayer)	WiFi/BLE/ZigBee
Hardware Layer		IoT Devices

Bảng 2.1 Kiến trúc phân lớp của hệ thống

2.1.1 Hardware Layer and Protocol Layer

Cơ sở hệ thống phần cứng được cấu thành từ hai lớp hardware layer và protocol layer. Hardware layer chứa các thiết bị IoT khác nhau, từ nhiều hãng khác nhau. Các thiết bị IoT giao tiếp với các lớp phía trên của kiến trúc thông qua các protocol (giao thức) được định nghĩa trong protocol layer. Các giao thức mạng sử dụng để giao tiếp như WiFi, ZigBee or Bluetooth được ghi nhận tại lớp con communication (communication sublayer). Để bảo vệ

quá trình giao tiếp giữa các dịch vụ phía trên với các thiết bị IoT, lớp protocol layer cung cấp thêm lớp con security (security sublayer) với hai loại bảo mật TLS (Transport Layer Security) hoặc DTLS (Datagram Transport Layer Security).

2.1.2 Proxy Layer and Blockchain Layer

Proxy layer và blockchain layer như một lớp trung gian giữa dịch vụ phía trên với thiết bị IoT và là môi trường để trao đổi, xác nhận cũng như chứng thực thông tin.

Proxy layer như các gateway để các thiết bị IoT kết nối với Internet thông qua việc thực hiện triển khai giao thức HTTP/HTTPS để giao tiếp với các API được cung cấp từ những dịch vụ phía trên.

Blockchain layer gồm hai lớp con là infrastructure (cơ sở hạ tầng mạng) và smart contracts (hợp đồng thông minh), hỗ trợ việc cung cấp các phương thức cần cho các giao dịch giữa dịch vụ phía trên với thiết bị IoT. Lớp con infrastructure chứa các thành phần vật lý cấu thành nên kiến trúc một Private Blockchain cần thiết cho hệ thống như: các CAs (Certificate Authorities), các sổ cái phân tán bất biến (Distributed Immutable Ledgers) và các thuật toán đồng thuận (Consensus Protocols). Tính cấp quyền (permissioned) của mạng được thể hiện qua việc xác nhận danh tính của các nút trong mạng từ các CA. Các nút tham gia mạng và truy cập vào sổ cái để thực hiện các giao dịch. Thuật toán đồng thuận sẽ đảm bảo tính toàn vẹn của các giao dịch để qua đó xác thực và ghi nhận giao dịch đó vào sổ cái. Lớp con smart contracts giúp hỗ trợ thêm các chức năng cấp cao để dễ dàng giao tiếp với sổ cái. Lớp con này chịu trách nhiệm cung cấp chức năng chứng thực quá trình cập nhật firmware (Firmware Update Verification) và điều khiển những vấn đề xảy ra trong suốt quá trình cập nhật (Incident Handling).

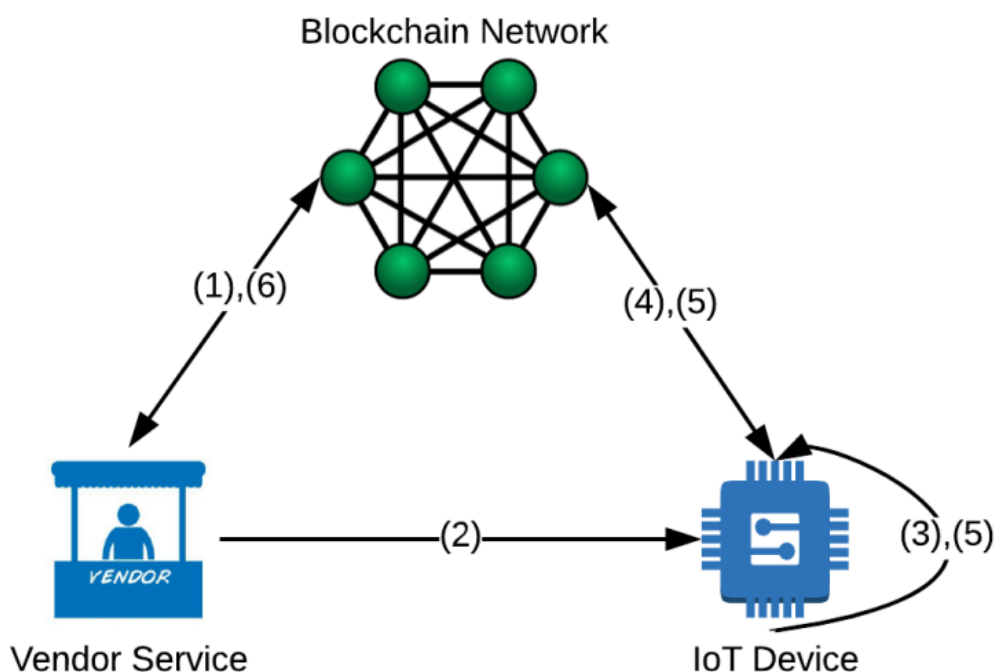
2.1.3 Service Layer and Application Layer

Service layer là các đơn vị cung cấp các dịch vụ (gọi chung là Vendor Service) như tung ra các phiên bản firmware hay yêu cầu cập nhật phiên bản firmware mới.

Application layer là các giao diện cung cấp bởi các nền tảng ứng dụng như web hay ứng dụng điện thoại di động cho phép người dùng dễ dàng thao tác các yêu cầu với Vendor Service.

2.2 Quá trình xác thực yêu cầu cập nhật firmware

Mục đích chính của hệ thống chính là xác thực việc cập nhật firmware. Ba đối tượng trực tiếp tham gia vào quá trình này là: thiết bị IoT, Vendor Service và mạng Blockchain. Các thiết bị IoT sẽ triển khai sử dụng các firmware để thu nhận dữ liệu (có thể là từ các cảm biến) hoặc thi hành các yêu cầu (như điều khiển các công tắc,...). Vendor Service cung cấp các chức năng các giao dịch cần cho việc cập nhật firmware. Và mạng Blockchain (ở đây chính là Private Blockchain Network), sẽ hỗ trợ cho việc xác thực thông qua các hợp đồng thông minh.



Hình 2.1 Quá trình xác thực cập nhật firmware

Quá trình xác thực việc cập nhật phiên bản firmware mới thông qua yêu cầu của Vendor Service bao gồm sáu bước (Hình 2.1):

- (1) Vendor Service khởi tạo một yêu cầu cập nhật firmware mới, đồng nghĩa khởi tạo một giao dịch mới chứa thông tin của thiết bị IoT đích và mã băm của phiên bản firmware mới. Kết quả của giao dịch (định danh bởi TXID) được thêm một khối mới vào sổ cái.
- (2) Vendor Service gửi file binary của phiên bản firmware mới đến thiết bị IoT đích cùng với TXID từ bước (1).

- (3) Thiết bị IoT nhận file binary của phiên bản firmware mới từ Vendor Service cùng TXID và tính mã băm từ file binary này.
- (4) Thiết bị IoT truy vấn sổ cái theo mã băm vừa tính và TXID ở bước (3) để thực hiện xác thực.
- (5) Nếu xác thực thành công, thiết bị IoT sẽ sử dụng file binary ấy để cài đặt và cập nhật tình trạng của quá trình cập nhật firmware trên sổ cái. Ngược lại, file binary sẽ được hủy bỏ và thiết bị sẽ ghi nhận vào sổ cái (thiết bị vẫn sẽ tiếp tục sử dụng phiên bản firmware cũ).
- (6) Vendor Service sẽ truy vấn sổ cái để kiểm tra tình trạng của yêu cầu cập nhật firmware đã tạo từ bước (1) và từ đó đưa ra các quyết định khác.

Thông qua các bước trên, nhận thấy rằng smart contract (hợp đồng thông minh) có thể gồm ba giao dịch cơ bản: *push()*, *verify()* và *query()*. Giao dịch *push()* được dùng bởi Vendor Service trong bước (1) để tạo yêu cầu cập nhật. Thiết bị IoT thực hiện giao dịch *verify()* để chứng thực. Và Vendor Service có thể giám sát quá trình cập nhật thông qua việc sử dụng giao dịch *query()*. Giao dịch này cung cấp các thông tin về tình trạng của yêu cầu cập nhật.

CHƯƠNG 3 TRIỂN KHAI XÂY DỰNG HỆ THỐNG

Hệ thống cung cấp dịch vụ cập nhật firmware từ xa cho các thiết bị IoT sẽ được triển khai thông qua các bước: xây dựng Private Blockchain Network dựa trên framework Hyperledger Fabric, xây dựng Vendor Service, xây dựng Web Server và thiết kế hàm chức năng hỗ trợ cập nhật nhúng xuống thiết bị IoT.

3.1 Môi trường phát triển

Dựa trên ba đối tượng chính của hệ thống (thiết bị IoT, Vendor Service và mạng Blockchain), các điều kiện tiên quyết để triển khai hệ thống được yêu cầu khác nhau.

3.1.1 Điều kiện tiên quyết cho thiết bị IoT

Trong phạm vi nghiên cứu của khóa luận này, thiết bị IoT được chọn để triển khai hệ thống là NodeMCU ESP32. Môi trường phát triển cho thiết bị IoT là Arduino IDE.

3.1.1.1 Thông tin thiết bị IoT (ESP32-NodeMCU)



Hình 3.1 Thiết bị IoT – NodeMCU ESP32

Thiết bị IoT là module NodeMCU dựa trên ESP32, có kết nối WiFi + Bluetooth, CP2102 trên bo mạch và các nút bấm. Hơn nữa, tất cả các chân I/O của module ESP-WROOM-32 đều có thể truy cập được thông qua GPIO.

Nhờ tài nguyên nguồn mở phong phú, NodeMCU ESP32 hỗ trợ phát triển theo nhiều cách khác nhau như lệnh Lua / AT commands/ mã nguồn MicroPython / Arduino / IoT, v.v. hỗ trợ nhanh chóng tạo các ứng dụng prototype IoT.

3.1.1.2 *Arduino IDE*

Arduino IDE là một phần mềm mã nguồn mở chủ yếu được sử dụng để viết và biên dịch mã vào module Arduino. Đây là một phần mềm Arduino chính thức, giúp cho việc biên dịch mã trở nên dễ dàng mà ngay cả một người bình thường không có kiến thức kỹ thuật cũng có thể làm được. Có rất nhiều các module Arduino, mỗi module chứa một bộ vi điều khiển trên board mạch được lập trình và chấp nhận thông tin dưới dạng mã. Mã chính, còn được gọi là sketch, được tạo trên nền tảng IDE sẽ tạo ra một file Hex, sau đó được chuyển và tải lên trong bộ điều khiển trên board. Môi trường IDE chủ yếu chứa hai phần cơ bản: Trình chỉnh sửa và Trình biên dịch, phần đầu sử dụng để viết mã được yêu cầu và phần sau được sử dụng để biên dịch và tải mã lên module Arduino. Môi trường này hỗ trợ cả ngôn ngữ C và C++.

3.1.2 *Điều kiện tiên quyết cho Vendor Service*

Vendor Service được thiết kế trên nền tảng ngôn ngữ JavaScript với framework NodeJS và NPM.

3.1.2.1 *NodeJS*

NodeJS là một framework được xây dựng trên V8 JavaScript Engine – trình thông dịch thực thi mã JavaScript, giúp xây dựng các ứng dụng web một cách đơn giản và dễ dàng mở rộng.

3.1.2.2 *NPM*

NPM là viết tắt của Node Package Manager là một công cụ tạo và quản lý các thư viện lập trình Javascript cho NodeJS. Trong cộng đồng Javascript, các lập trình viên chia sẻ hàng trăm nghìn các thư viện với các đoạn code đã thực hiện sẵn một chức năng nào đó. Nó giúp cho các dự án mới tránh phải viết lại các thành phần cơ bản, các thư viện lập trình hay thậm chí cả các framework.

Cài đặt: NPM có sẵn khi bạn tải NodeJS về. Để kiểm tra xem trên hệ thống của bạn đã được cài NPM chưa chúng ta sử dụng lệnh `npm -v`, nếu một phiên bản hiện ra thì hệ thống của bạn đã được cài đặt NPM. Vì NPM là một phần mềm cài đặt trên máy tính của bạn nên bạn có thể sử dụng nó để cài đặt các thư viện Javascript từ trên Internet. Để cài đặt một thư viện nào đó, chỉ cần mở cửa sổ Terminal (hoặc CMD) và thực thi lệnh giống dưới đây: `npm install package-name`

Cài đặt global và cài đặt local: Có hai cách để cài đặt một gói bằng NPM:

- Local: sẽ tạo ra thư mục `node_modules` nếu chưa có trong project hoặc nếu có rồi nó sẽ lấy code của gói cần cài đặt đưa vào đây, tức chỉ hiện diện trong thư mục của project hiện tại. Khi cần sử dụng bạn có thể sử dụng lệnh `require()`.
- Global: sẽ lưu trữ code của gói trong các file hệ thống cố định trong máy, chỉ có thể dùng các package này thông qua các hàm CLI (Command Line Interface). Không thể dùng package thông qua `require()`.

Mặc định thì các package khi cài đặt đều sẽ là cài trên project của bạn.

Package.json: Để quản lý các gói cài đặt cục bộ bằng NPM thì cách tốt nhất là thông qua file `package.json`, chính là file nằm trong thư mục gốc của project. File JSON này chứa các nội dung:

- Các gói thư viện lập trình mà project sử dụng.
- Cho phép xác định phiên bản chính xác các gói thư viện lập trình được sử dụng.
- Các gói bạn xây dựng có thể chia sẻ dễ dàng với các lập trình viên khác trên toàn cầu thông qua NPM.

Lệnh `npm init -yes` sẽ tạo ra file `package.json` mẫu.

3.1.3 Điều kiện tiên quyết cho Blockchain Network

Blockchain Network được xây dựng với framework Hyperledger Fabric (HF). Các thành phần trong kiến trúc mạng xây dựng với HF đều được chạy trên nền Docker. Hợp đồng thông minh (smart contract/ hay trong HF gọi là chaincode) được phát triển trên nền NodeJS và NPM.

3.1.3.1 Git

Git là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS), nó là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. Git cung cấp cho mỗi lập trình viên kho lưu trữ (repository) riêng chứa toàn bộ lịch sử thay đổi. Git cung cấp chương trình Git Bash, giúp hỗ trợ mô phỏng giao diện console cho Windows tương tự như terminal của Linux hay Mac. Git Bash được cài đặt sẵn khi cài

đặt Git cho Windows. Đây là môi trường chính để phát triển cũng như triển khai mạng Hyperledger Fabric.

3.1.3.2 Docker và Docker Compose

Docker là nền tảng phần mềm cho phép xây dựng, kiểm thử và triển khai ứng dụng một cách nhanh chóng. Docker đóng gói phần mềm vào các đơn vị tiêu chuẩn hóa được gọi là container có mọi thứ mà phần mềm cần để chạy, trong đó có thư viện, công cụ hệ thống, mã và thời gian chạy. Việc sử dụng Docker hỗ trợ có thể nhanh chóng triển khai và thay đổi quy mô ứng dụng vào bất kỳ môi trường nào và biết chắc rằng mã nguồn sẽ chạy được.

Docker hoạt động bằng cách cung cấp phương thức tiêu chuẩn để chạy mã nguồn. Docker là hệ điều hành dành cho container. Cũng tương tự như cách máy ảo ảo hóa (loại bỏ nhu cầu quản lý trực tiếp) phần cứng máy chủ, các container sẽ ảo hóa hệ điều hành của máy chủ. Docker được cài đặt trên từng máy chủ và cung cấp các lệnh đơn giản mà bạn có thể sử dụng để dựng, khởi động hoặc dừng container. Không giống như máy ảo (Virtual Machine) nơi mà sự ảo hóa (virtualization) xảy ra ở tầng phần cứng (hardware level), container chỉ ảo hóa ở lớp ứng dụng (app level). Nó có thể dùng 1 máy, chia sẻ kernel và giả môi trường để chạy process độc lập. Điều này làm cho container cực kì nhẹ, không chiếm nhiều tài nguyên của máy.

Docker compose là công cụ dùng để định nghĩa và chạy multi-container cho các ứng dụng Docker. Với compose bạn sử dụng file YAML để cấu hình (config) các dịch vụ (services) cho các ứng dụng. Sau đó dùng command để tạo và chạy từ những config đó.

Các thuật ngữ quan trọng trong Docker:

- *Image*: Docker image là một file bất biến - không thay đổi, chứa các source code, libraries, dependencies, tools và các files khác cần thiết cho một ứng dụng để chạy. Do tính chất read-only của chúng, những images này đôi khi được gọi là *snapshots*. Chúng đại diện cho một application (ứng dụng) và virtual environment của nó tại một thời điểm cụ thể. Tính nhất quán này là một trong những tính năng tuyệt vời của Docker. Nó cho phép kiểm tra và thử nghiệm phần mềm trong điều kiện ổn định, thống nhất. Các image chỉ là các mẫu nên không thể start hoặc run chúng. Chỉ có thể sử dụng mẫu đó làm cơ sở để xây dựng một container.

- *Container*: Docker container là một run-time environment mà ở đó người dùng có thể chạy một ứng dụng độc lập. Những container này rất gọn nhẹ và cho phép bạn chạy ứng dụng trong đó rất nhanh chóng và dễ dàng. Một tính năng quan trọng của container là tính chuẩn xác cho việc chạy các ứng dụng trong container. Không chỉ đảm bảo cho ứng dụng hoạt động như nhau trong các môi trường giống nhau, nó còn làm đơn giản việc cài đặt và chia sẻ cài đặt này cho các thành viên trong team. Vì container hoạt động độc lập, nó đảm bảo không làm ảnh hưởng xấu đến các container khác, cũng như server mà nó đang chạy trong đó.
- *Volume*: Volume trong Docker được dùng để chia sẻ dữ liệu cho container. Có thể sử dụng Volume trong Docker trong những trường hợp như chia sẻ giữa container và container hoặc giữa container và host.

Pulling các image, các tool: sau khi cài đặt Docker, cần kéo (pull) các image cần thiết từ repository của Hyperledger Fabric và gắn thẻ (tag) mới nhất (latest) cho chúng theo các lệnh sau (thực hiện trong vùng biên tập của Git Bash) và kéo các tool cần thiết cho việc xây dựng hạ tầng mạng sau này (lưu trữ trong thư mục *bin*) [4]. Có thể sử dụng công cụ cURL để thực hiện thao tác này thông qua lệnh sau trong thư mục đích:

```
curl -sSL http://bit.ly/2ysbOFE | bash -s -- 1.4.2 1.4.2
```

Lưu ý ở phạm vi khóa luận này, nền tảng Hyperledger Fabric được sử dụng là phiên bản 1.4.2.

3.1.3.3 Golang

Go hay còn gọi là Golang là ngôn ngữ lập trình mã nguồn mở, được thiết kế tại Google. Hyperledger Fabric sử dụng ngôn ngữ lập trình Go cho nhiều thành phần trong kiến trúc mạng.

3.1.3.4 CouchDB

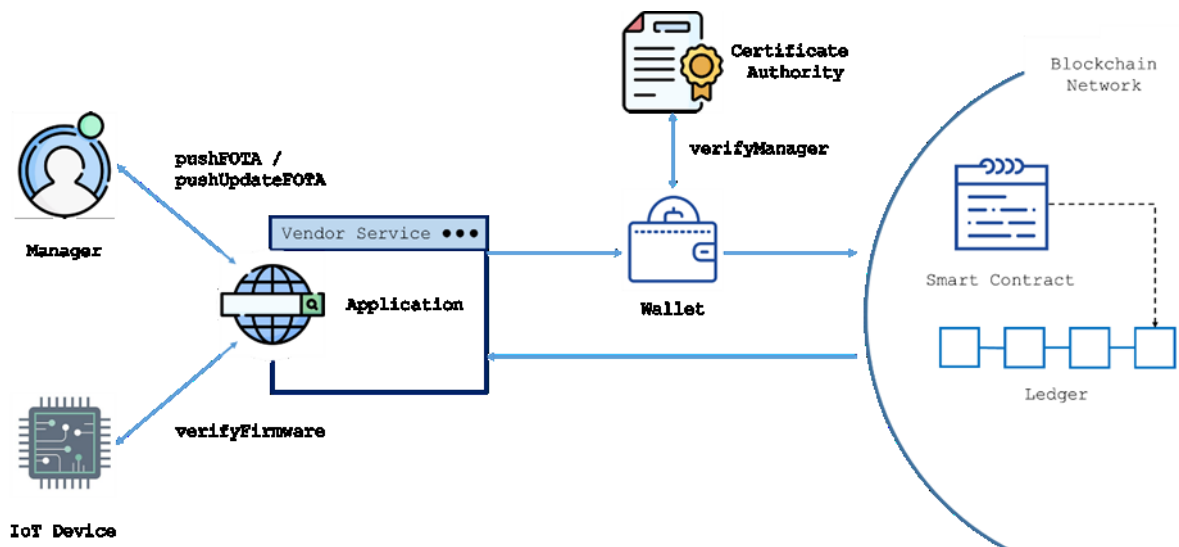
CouchDB là một sơ sở dữ liệu dạng NoSQL mã nguồn mở database lưu trữ dữ liệu dạng document/JSON. CouchDB được thiết kế nhằm tới tính dễ sử dụng và phục vụ cho môi trường web. Mô hình dữ liệu trong CouchDB:

- Database là cấu trúc dữ liệu lớn nhất của CouchDB.
- Mỗi database là 1 danh sách các document độc lập.

- Document bao gồm dữ liệu người dùng thao tác lần thông tin về phiên bản của dữ liệu để tiện việc merge dữ liệu.
- CouchDB sử dụng cơ chế phiên bản hoá dữ liệu để tránh tình trạng khoá dữ liệu khi đang ghi.

Việc lưu trữ trên CouchDB thông qua các cấu trúc dữ liệu dưới dạng các document. Đây là nền tảng cơ sở dữ liệu cho sổ cái của mạng Hyperledger Fabric. Thông qua các cấu trúc dữ liệu được định nghĩa trong Smart Contract, các thành viên trong mạng có thể khởi tạo, truy vấn và cập nhật các đối tượng trên CouchDB.

3.2 Kịch bản hệ thống



Hình 3.2 Kịch bản hệ thống

Hình 3.2 mô tả kịch bản các thành phần sẽ tương tác với nhau trong hệ thống.

Người dùng (User) được định nghĩa trong hệ thống là các Manager sẽ tương tác mạng thông qua ứng dụng khách Vendor Service để thực hiện các thao tác như:

- *Sign In*: tạo thông tin định danh (identity) được cấp từ các CA và lưu trữ vào ví điện tử (Wallet).
- *Log In*: sử dụng thông tin định danh để truy cập vào mạng Blockchain Network.
- *Push*: đăng tải thông tin về thiết bị IoT và các firmware lưu trữ trên cơ sở dữ liệu. việc khởi tạo được thực hiện thông qua các giao dịch tương tác với hợp đồng thông minh (Smart Contract) trong mạng.

- *Require Update*: gửi lên yêu cầu cập nhật firmware từ xa cho một thiết bị IoT đã được ghi nhận trên cơ sở dữ liệu.
- *Query*: truy vấn các thông tin trong mạng.

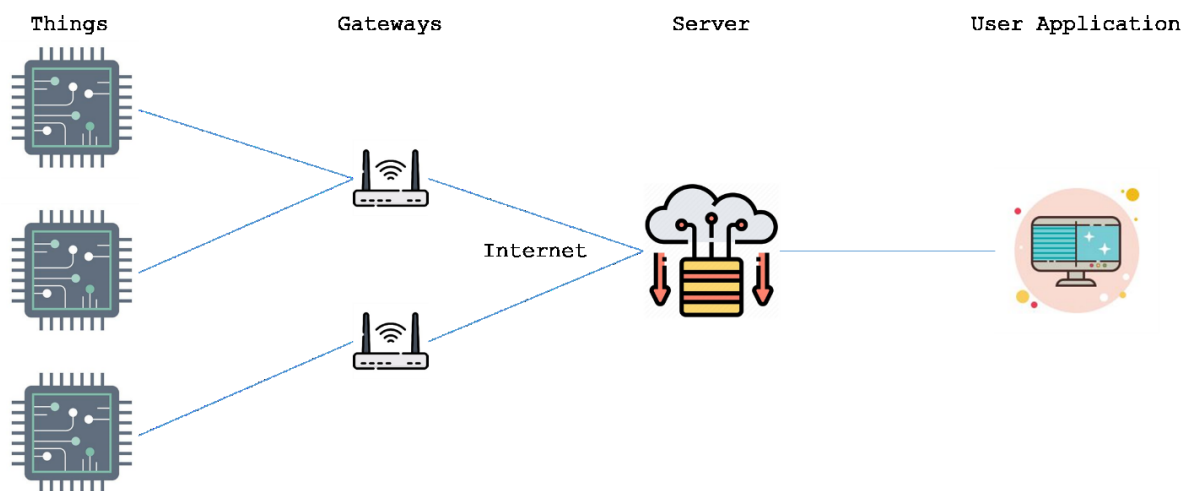
Thiết bị IoT sẽ nhận yêu cầu cập nhật từ Vendor Service và xác thực quá trình đó. Nếu xác thực thành công, thiết bị sẽ thực hiện việc cập nhật đó và ngược lại, thiết bị sẽ hủy bỏ yêu cầu và cập nhật lên trạng thái hệ thống.

3.3 Kiến trúc cơ bản của hệ thống Internet of Things (IoT)

IoT là một hệ thống các thiết bị máy tính, máy móc,... hoặc người có liên quan đến nhau, được cung cấp định dạng duy nhất và có khả năng để truyền dữ liệu qua mạng mà không cần sự tương tác giữa con người với con người hay giữa con người với máy tính. IoT đã phát triển từ sự hội tụ của công nghệ không dây, công nghệ vi cơ điện tử và Internet. Nói đơn giản hơn là một tập hợp các thiết bị có khả năng kết nối với nhau, với Internet và với thế giới bên ngoài để thực hiện một công việc nào đó.

Hay hiểu một cách đơn giản IoT là tất cả các thiết bị có thể kết nối với nhau. Việc kết nối có thể thực hiện qua Wifi, mạng viễn thông băng rộng (3G hay 4G) hiện tại là 5G, Bluetooth, ZigBee, hồng ngoại,... Các thiết bị có thể là điện thoại thông minh, máy pha cà phê, máy giặt, tai nghe và rất nhiều thiết bị khác. Một mạng lưới IoT có thể chứa đến 50 đến 100 nghìn tỷ đối tượng được kết nối và mạng lưới này có thể theo dõi sự di chuyển của từng đối tượng. Một con người sống trong thành thị có thể bị bao bọc xung quanh bởi 1000 đến 5000 đối tượng có khả năng theo dõi.

Với một hệ thống IoT chúng sẽ bao gồm 4 thành phần chính đó là thiết bị hay còn gọi là (Things), trạm kết nối hay cổng kết nối (Gateways), hạ tầng mạng hay các điện toán đám mây (Network and Cloud) và bộ phân tích và xử lý dữ liệu (Services-creation and Solution Layers).



Hình 3.3 Cấu trúc hệ thống IoT

Vật (Things): Ngày nay có hàng tỷ vật dụng đang hiện hữu trên thị trường gia dụng và công nghệ, ở trong nhà hoặc trên tay của người dùng. Chẳng hạn như xe hơi, thiết bị cảm biến, thiết bị đeo và điện thoại di động đang được kết nối trực tiếp thông qua băng tần mạng không dây và truy cập vào Internet. Giải pháp IoT giúp các thiết bị thông minh được sàng lọc, kết nối và quản lý dữ liệu một cách cục bộ, còn các thiết bị chưa thông minh thì có thể kết nối được thông qua các trạm kết nối.

Trạm kết nối (Gateways): Một rào cản chính khi triển khai IoT đó là gần 85% các vật dụng đã không được thiết kế để có thể kết nối với Internet và không thể chia sẻ dữ liệu với điện toán đám mây. Để khắc phục vấn đề này, các trạm kết nối sẽ đóng vai trò là một trung gian trực tiếp, cho phép các vật dụng có sẵn này kết nối với điện toán đám mây một cách bảo mật và dễ dàng quản lý.

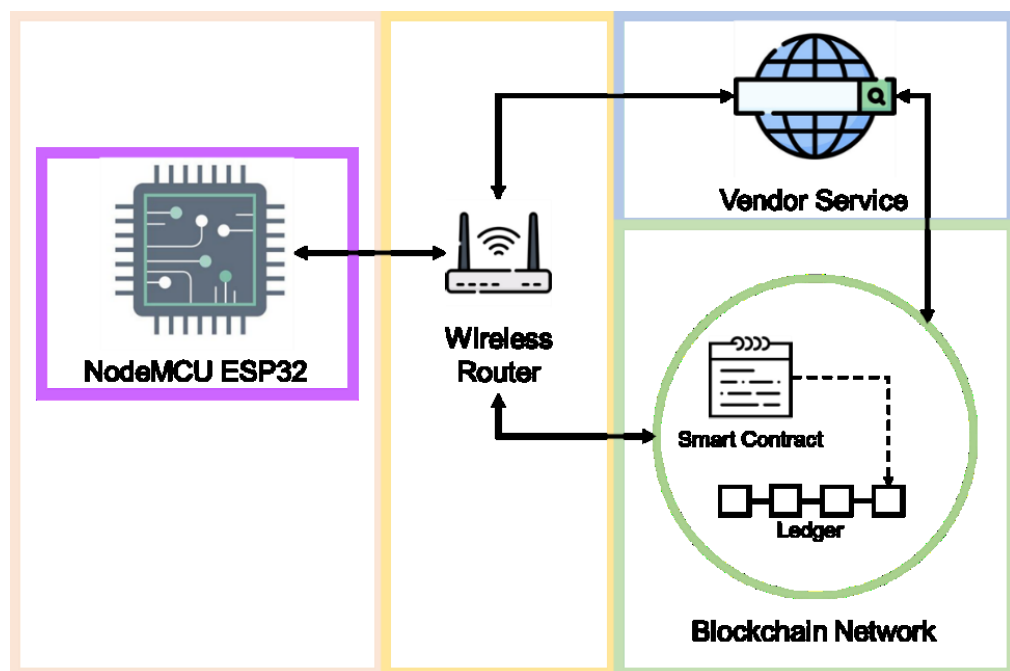
Hạ tầng mạng và điện toán đám mây (Network and Cloud):

- Cơ sở hạ tầng kết nối: Internet là một hệ thống toàn cầu của nhiều mạng IP được kết nối với nhau và liên kết với hệ thống máy tính. Cơ sở hạ tầng mạng này bao gồm thiết bị định tuyến, trạm kết nối, thiết bị tổng hợp, thiết bị lặp và nhiều thiết bị khác có thể kiểm soát lưu lượng dữ liệu lưu thông và cũng được kết nối đến mạng lưới viễn thông và cáp - được triển khai bởi các nhà cung cấp dịch vụ.
- Trung tâm dữ liệu/ hạ tầng điện toán đám mây: Các trung tâm dữ liệu và hạ tầng điện toán đám mây bao gồm một hệ thống lớn các máy chủ, hệ thống lưu trữ và mạng ảo hóa được kết nối.

Các lớp tạo và cung cấp dịch vụ (Services-Creation and Solutions Layers): cung cấp những phần mềm ứng dụng sử dụng và quản lý API (Application Programming Interface) là thông qua đó, đưa các sản phẩm và giải pháp IoT ra thị trường và tận dụng được giá trị của việc phân tích các dữ liệu từ hệ thống và tài nguyên từ hệ thống.

3.4 Kiến trúc mô hình hệ thống

Hệ thống được xây dựng gồm ba thành phần chính là mạng Blockchain Network, Vendor Service và thiết bị IoT (NodeMCU ESP32) cùng các thành phần phụ khác được thể hiện qua hình 3.4.



Hình 3.4 Kiến trúc mô hình hệ thống

Triển khai hệ thống qua bốn bước sau:

- (1) Phát triển hợp đồng thông minh.
- (2) Xây dựng cơ sở hạ tầng mạng.
- (3) Triển khai dịch vụ web server (Vendor Service).
- (4) Lập trình nhúng cho thiết bị IoT.

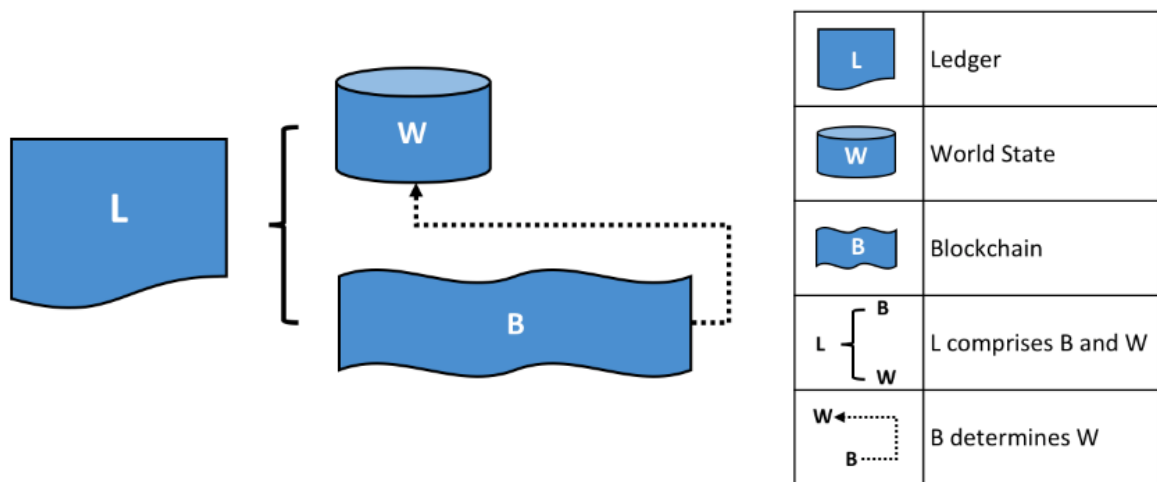
Đầu tiên cần xác định các cấu trúc dữ liệu để lưu trữ và các logic giao dịch thông qua việc phát triển hợp đồng thông minh. Từ đó hỗ trợ cung cấp các phương thức giao dịch để tương tác giữa mạng Blockchain Network với Vendor Service. Cơ sở hạ tầng mạng Blockchain

được xây dựng dựa trên nền tảng Hyperledger Fabric cung cấp các channel (kênh) để giao tiếp giữa các thành phần trong mạng. Vendor Service thực hiện việc truy cập vào các peer trong mạng để sử dụng các logic giao dịch của hợp đồng thông minh. Qua đó xử lý các yêu cầu cập nhật firmware từ phía người dùng là các Manager đến các thiết bị IoT trong hệ thống. Các thiết bị IoT được lập trình nhúng để nhận các yêu cầu cập nhật từ Vendor Service và thực hiện quá trình xác thực yêu cầu đó. Nếu quá trình xác thực thành công thì các thiết bị IoT sẽ tiến hành cập nhật firmware và sau khi cập nhật trong xong sẽ ghi nhận lại trạng thái của yêu cầu đó trên cơ sở dữ liệu, và ngược lại khi xác thực thất bại trạng thái cũng vẫn được ghi nhận lại. Các Manager có thể kiểm tra tiến trình trạng thái của yêu cầu cập nhật thông qua việc truy vấn cấu trúc dữ liệu tương ứng với yêu cầu cập nhật trên cơ sở dữ liệu hệ thống.

3.5 Phát triển Smart Contract (hợp đồng thông minh)

Smart Contract xác định các quy tắc giữa các tổ chức khác nhau trong các hành động. Các application gọi một smart contract để tạo ra các giao dịch được ghi lại kết quả trên Ledger. Một smart contract định nghĩa các logic giao dịch, sau đó smart contract được đóng gói thành chaincode nên có thể xem các thuật ngữ smart contract, chaincode là một. Trong khóa luận này, Smart Contract được phát triển có tên là *fota (fotaContract)* viết bằng ngôn ngữ JavaScript.

Ở cấp độ đơn giản nhất, một blockchain bất biến ghi lại các giao dịch cập nhật kết quả thực hiện giao dịch trong một Ledger. Smart contract có thể truy cập theo hai phần riêng biệt, một là lịch sử của tất cả các giao dịch và hai là trạng thái hiện tại của Ledger (kết quả sau khi thực hiện tất cả các giao dịch). Mỗi kênh có thể có nhiều Chaincode nhưng chỉ có một Ledger.



Hình 3.5 Cấu trúc của một Ledger (sổ cái)

Theo hình 3.5, Ledger L khi phân tích ra sẽ bao gồm hai đối tượng: Blockchain B và World State W (Trạng thái hiện tại của dữ liệu). Chúng ta có thể tùy chọn database để lưu Ledger L (ở đây dùng CouchDB). Blockchain B lưu lại toàn bộ giao dịch, giá trị của các đối tượng (được khởi tạo từ các cấu trúc dữ liệu lưu trữ trong Smart Contract) sau khi thực hiện các giao dịch ấy được lưu vào World State [5].

3.5.1 Xác định các cấu trúc dữ liệu trong Smart Contract

Trong một Smart Contract, ngoài lớp *fotaContract* chứa các logic giao dịch (hay còn gọi là các phương thức) thì cần có các cấu trúc dữ liệu dùng để định nghĩa các đối tượng trong contract.

Có bốn cấu trúc dữ liệu trong *fotaContract*:

- *Manager*: dùng để định nghĩa tài khoản người dùng khi truy cập vào application và thực hiện các giao dịch. Các *Manager* sử dụng thuộc tính *managerId* để định danh trên cơ sở dữ liệu.
- *IoTSystem*: dùng để định nghĩa hệ thống phần cứng (các thiết bị IoT). Các *IoTSystem* sử dụng địa chỉ MAC tương ứng của các thiết bị IoT được ghi nhận trong thuộc tính *macAddress* để định danh trên cơ sở dữ liệu.
- *FirmwareOTA*: dùng để định nghĩa firmware được push lên hệ thống. Các *FirmwareOTA* sử dụng phiên bản firmware được ghi nhận trong thuộc tính

firmwareVersion để định danh trên cơ sở dữ liệu. Đồng thời file binary firmware sau khi push sẽ được xử lý qua hàm băm SHA256 để lấy giá trị băm ghi nhận vào thuộc tính *firmwareHash* để đảm bảo tính toàn vẹn của dữ liệu.

- *UpdateFOTA*: dùng để định nghĩa hoạt động cập nhật firmware từ xa. Các *UpdateFOTA* sử dụng transaction ID được ghi nhận trong thuộc tính *txid* để định danh trên cơ sở dữ liệu.

3.5.2 Phát triển các logic giao dịch trong Smart Contract

Trước tiên, cần khai báo các thư viện cần sử dụng trong file *package.json*. Thông qua việc cài đặt các thư viện thông qua công cụ NPM.

```
"dependencies": {
  "fabric-contract-api": "~1.4.0",
  "js-sha256": "^0.9.0",
  "fabric-shim": "~1.4.0"
},
"devDependencies": {
  "chai": "^4.1.2",
  "eslint": "^4.19.1",
  "mocha": "^5.2.0",
  "nyc": "^12.0.2",
  "sinon": "^6.0.0",
  "sinon-chai": "^3.2.0"
},
```

Hyperledger Fabric cung cấp biến *ctx* để thực hiện việc tạo các logic giao dịch [6]. Biến *ctx.stub* cung cấp các API có thể giao tiếp với World State như *putState()*, *getState()*... [7]

Như đã thảo luận trong mục 2.2, Smart Contract cần có ba logic giao dịch chính là *push()*, *verify()* và *query()*. Trong *fotaContract*, cũng được chia thành ba loại logic giao dịch chính tương tự. Ngoài ra, *fotaContract* còn logic giao dịch *init()* dùng khi khởi tạo *fotaContract* trên peer, logic giao dịch *recordRequire()* dùng để ghi nhận lại trạng thái của yêu cầu cập nhật.

3.5.3 Logic giao dịch push

Logic giao dịch *push* dùng để khởi tạo các đối tượng mới trên World State thông qua việc sử dụng *ctx.stub.putState(key, value)*. Phương thức *putState()* sử dụng tham số *key* để định danh đối tượng đó trong World State, và tham số *value* lưu trữ dữ liệu của đối tượng đó dạng JSON. Ngoài ra loại logic giao dịch *push* sử dụng *createCompositeKey(indexName, attributes[])* cũng được cung cấp bởi biến *ctx.stub* để phục vụ cho việc truy vấn toàn bộ đối tượng cùng *indexName*.

Có bốn logic giao dịch gồm:

- *pushManager* (khởi tạo đối tượng *Manager*),
- *pushSystem* (khởi tạo đối tượng *IoTSystem*),
- *pushFOTA* (khởi tạo đối tượng *FirmwareOTA*),
- *postUpdateFOTA* (khởi tạo đối tượng *UpdateFOTA*).

3.5.4 Logic giao dịch verify

Logic giao dịch *verify* dùng để chứng thực các đối tượng trên World State.

Có hai logic giao dịch gồm:

- *verifyRequire* (chứng thực quá trình cập nhật firmware),
- *verifyManager* (chứng thực quá trình đăng nhập).

3.5.5 Logic giao dịch query

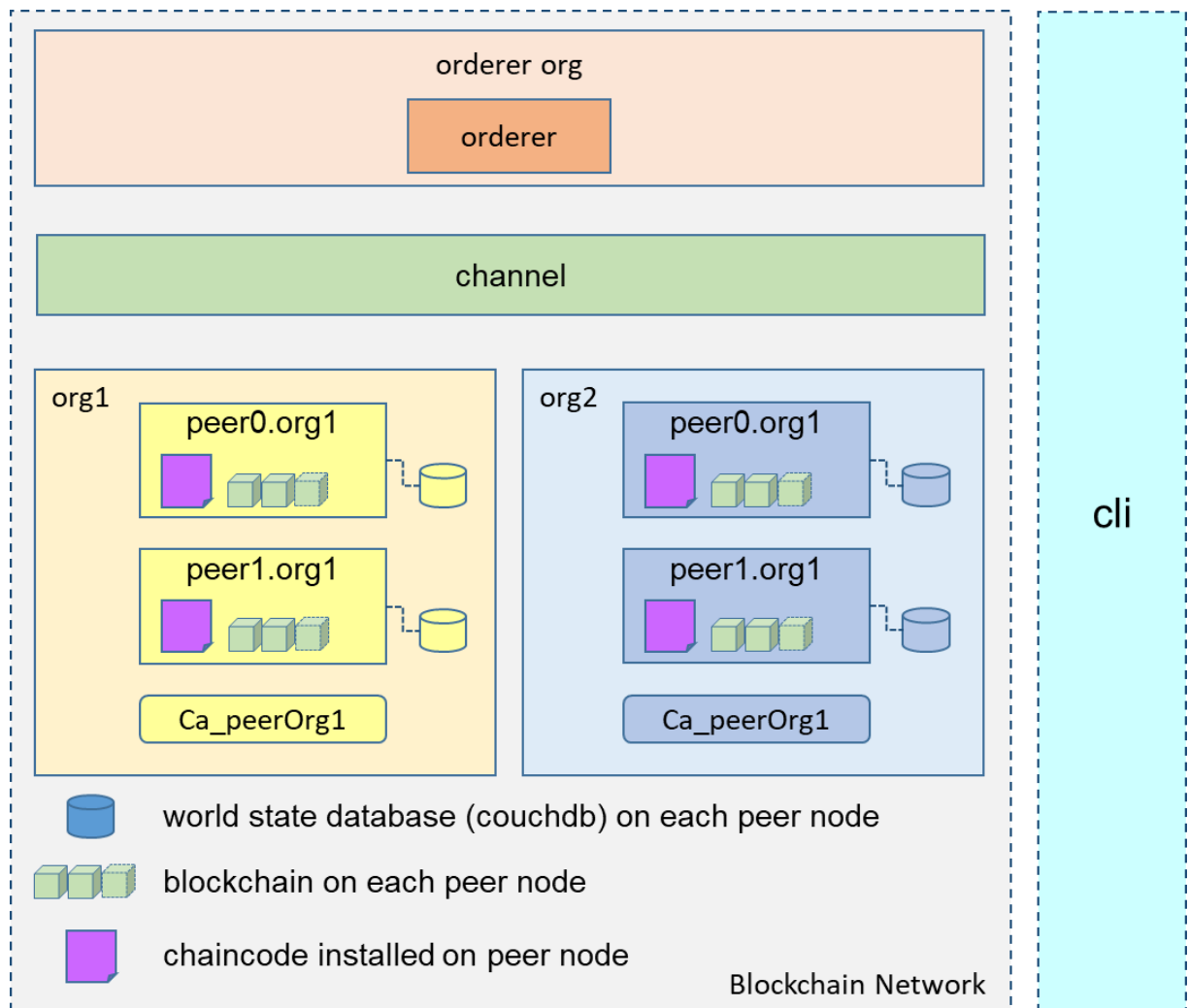
Logic giao dịch *query* dùng để truy vấn các đối tượng trên World State thông qua việc sử dụng *ctx.stub.getState(key)*. Phương thức *getState()* sử dụng tham số *key* để định danh đối tượng đó trong World State. Ngoài ra loại logic giao dịch *query* sử dụng *ctx.stub.getStateByPartialCompositeKey(indexName, key[])* để phục vụ cho việc truy vấn toàn bộ đối tượng cùng loại (*type*).

Có hai logic giao dịch gồm:

- *queryAllByPartialCompositeKey* (truy vấn các đối tượng cùng *indexKey*),
- *queryByKey* (truy vấn đối tượng bằng *key*).

3.6 Xây dựng cơ sở hạ tầng mạng

Cơ sở hạ tầng mạng sẽ được thiết kế dựa trên kiến trúc đơn giản của một mạng Hyperledger Fabric [8].



Hình 3.6 Kiến trúc cơ sở hạ tầng mạng

3.6.1 Xác định kiến trúc mạng

Các thành phần chính trong kiến trúc mạng cần thiết kể gồm:

- 1 tổ chức Orderer và nút Orderer (orderer.example.com).
- 2 Organizations (org1, org2).
- 2 CA – tổ chức phát hành chứng chỉ, tương ứng cho từng tổ chức org1 và org2.
- 2 Peers cho mỗi Org chạy một cơ sở dữ liệu CouchDB để lưu trữ thông tin sổ cái:
 - Anchor Peer (peer0.org1.example.com, peer0.org2.example.com) dùng để xác định cho một tổ chức (Organization) nào đó, mỗi tổ chức sẽ có tối thiểu một nút (peer) và nút đó là Anchor Peer. Anchor peer sẽ ghi nhận các peer

thông qua việc cập nhật để đảm bảo việc giao tiếp giữa các peer và các tổ chức xảy ra thuận lợi.

- Peer (peer1.org1.example.com, peer1.org2.example.com).
- 1 cli (fabric-tools) là giao diện dòng lệnh để có thể tương tác với mạng Fabric.
- 1 channel (mychannel) kênh chia sẻ dữ liệu giữa các thành phần trong mạng.

Các thành phần vật lý trong mạng được triển khai dưới dạng các Container và chạy trên một máy chủ (localhost).

Các thành phần này được khai báo trong các file YAML:

- Nhóm các file *docker-composer.yaml* – thiết lập cấu hình Docker Container:
 - *docker-composer-ca.yaml* – cấu hình CAs.
 - *docker-composer-cli.yaml* – cấu hình giao diện dòng lệnh cli.
 - *docker-composer-couchdb.yaml* – cấu hình cơ sở dữ liệu CouchDB.
- *crypto-config.yaml* - cấu hình orderer và peers để tạo thông tin chứng chỉ.
- *configtx.yaml* – khai báo thông tin cấu hình để xây dựng khối khởi nguyên (genesis block) và file “channel transaction”.

3.6.2 Giới thiệu ngôn ngữ dòng lệnh Sh (Shell Command Language)

Sh - ngôn ngữ dòng lệnh (Shell Command Language) là một ngôn ngữ lập trình thông dịch được mô tả theo chuẩn POSIX standard. Nó là ngôn ngữ đầu tiên được sử dụng cho các chương trình Shell và có mặt trên hầu hết các hệ thống Unix/Linux. Sh thích hợp cho việc lập trình shell vì lợi thế nhỏ gọn và tốc độ xử lý. Nhưng nó cũng có các nhược điểm như thiếu các tính năng tương tác (vd: tính năng gọi lại các lệnh trước đó - history), không có các tính năng tích hợp số học cũng như xử lý logic.

Shell là chương trình người dùng đặc biệt, cung cấp giao diện cho người dùng sử dụng các dịch vụ hệ điều hành. Shell chấp nhận các lệnh có thể đọc được từ người dùng và chuyển đổi chúng thành thứ mà kernel có thể hiểu được. Nó là một trình thông dịch ngôn ngữ lệnh

thực thi các lệnh được đọc từ các thiết bị đầu cuối vào như keyboard hoặc từ file. Shell được bắt đầu khi người dùng đăng nhập hoặc khởi động terminal.

Shell được chia làm 2 loại:

- Command Line Shell
- Graphical shell

3.6.2.1 *Command Line Shell*

Shell có thể được truy cập bởi người dùng bằng cách sử dụng command line interface. Một chương trình đặc biệt có tên Terminal trong linux/ macOS hoặc Command Prompt trong Windows OS, được cung cấp để nhập vào các lệnh có thể đọc được của người dùng như "cat", "ls" etc. và sau đó nó được thực thi. Kết quả sau đó được hiển thị trên Terminal.

Làm việc với command line shell sẽ có một chút khó khăn cho người mới bắt đầu bởi vì thật khó để nhớ hết các lệnh. Nhưng khi thành thạo thì nó rất mạnh mẽ, nó cho phép người dùng lưu trữ các lệnh trong một file và thực thi chúng cùng nhau. Với tính năng này, bất kỳ nhiệm vụ lặp đi lặp lại nào có thể xử lý tự động. Các tệp này thường được gọi là batch file trong Windows và Shell Script trong Linux / macOS.

3.6.2.2 *Graphical Line Shell*

Graphical Shells cung cấp phương tiện để thao tác với các chương trình dựa trên graphical user interface (GUI), bằng cách cho phép các hoạt động như open, close, move and resize window, cũng như chuyển trọng tâm giữa các cửa sổ.

Window OS hoặc Ubuntu OS có thể được coi là ví dụ điển hình cung cấp GUI cho người dùng để tương tác với chương trình. Người dùng không cần nhập lệnh cho mọi hành động.

3.6.3 *Khởi tạo các thành phần cơ bản*

Thao tác xây dựng cơ sở hạ tầng mạng được thực hiện tại vùng biên tập của Git Bash.

Tạo các chứng chỉ (Certificates): sử dụng công cụ *cryptogen* trong thư mục *bin* để tạo các chứng chỉ cho các organization và các user. Lấy thông tin đầu vào từ file *crypto-config.yaml* để tạo chứng chỉ. Sau đó cập nhật biến private key cho chứng chỉ của các tổ chức.

Kết quả trả về: tạo một thư mục *crypto-config* chứa các chứng chỉ cho các organization và các user. Cấu trúc thư mục gồm:

- Tổ chức Orderer (example.com) chứa peer Orderer (orderer.example.com) trong thư mục con *ordererOrganizations*,
- 2 tổ chức org1 (org1.example.com) và org2 (org2.example.com) trong thư mục con *peerOrganizations*.

```
✓ crypto-config
  ✓ ordererOrganizations\example.com
    > ca
    > msp
    > orderers\orderer.example.com
    > tlsca
    > users\Admin@example.com
  ✓ peerOrganizations
    ✓ org1.example.com
      > ca
      > msp
      > peers
      > tlsca
      > users
    ✓ org2.example.com
      > ca
      > msp
      > peers
      > tlsca
      > users
```

Hình 3.7 Cấu trúc thư mục *crypto-config*

Tạo Genesis Block: sau khi tạo chứng chỉ, phải tạo khối đầu tiên cho mạng Blockchain (khối khởi nguyên – genesis block). Sử dụng công cụ *configtxgen* trong thư mục *bin* để tạo genesis block từ file *configtx.yaml*.

Tạo Channel Configuration Transaction: sử dụng công cụ *configtxgen* trong thư mục *bin* để tạo cấu hình cho việc tạo kênh *mychannel* ghi nhận trong file *channel.tx* từ file *configtx.yaml*.

Tạo cập nhật cho Anchor Peer: sử dụng công cụ *configtxgen* trong thư mục *bin* để tạo một transaction block được ký bởi OrgMSP tương ứng với mục đích cập nhật một peer đến Anchor Peer.

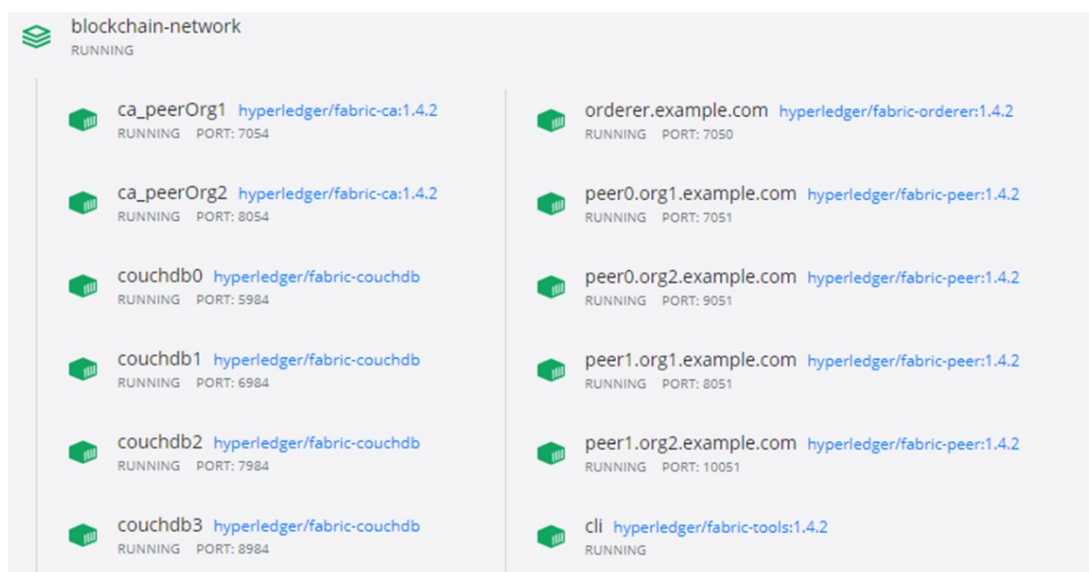
3.6.4 Triển khai mạng

Bắt đầu hoạt động mạng: sau khi đã có đủ các thành phần cơ bản thì có thể bắt đầu hoạt động các thành phần trong mạng. Chạy các file *docker-composer* để thực hiện việc tạo các container, các volume và run nó.

Kết quả trả về: các container được tạo là 1 orderer, 4 peers, 4 couchdb, 2 CA và cli đang hoạt động như hình 3.8 và 3.9.

```
Creating network "blockchain-network_blockchain_network" with the default driver
Creating volume "blockchain-network_orderer.example.com" with default driver
Creating volume "blockchain-network_peer0.org1.example.com" with default driver
Creating volume "blockchain-network_peer1.org1.example.com" with default driver
Creating volume "blockchain-network_peer0.org2.example.com" with default driver
Creating volume "blockchain-network_peer1.org2.example.com" with default driver
Creating ca_peerOrg2          ... done
Creating couchdb2             ... done
Creating orderer.example.com  ... done
Creating couchdb0             ... done
Creating ca_peerOrg1          ... done
Creating couchdb1             ... done
Creating couchdb3             ... done
Creating peer1.org1.example.com ... done
Creating peer1.org2.example.com ... done
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating cli                   ... done
```

Hình 3.8 Kết quả trên màn hình console khi bắt đầu hoạt động mạng



Hình 3.9 Kết quả trên Docker Engine khi bắt đầu hoạt động mạng

Tạo Channel và join (tham gia) các peer: sử dụng container *cli* để tạo và tham gia các peer vào channel từ file *channel.tx*. Đồng thời là cập nhật các anchor peer trong kênh *mychannel*. Lưu ý, trong *cli*, cần điều chỉnh các biến môi trường để khai báo người dùng đang xử lý trên peer nào cụ thể.

Truy cập vào container *cli* để thực hiện các cấu hình:

Thực hiện việc tạo kênh *mychannel* và join vào peer *peer0.org1*, *peer1.org1*, *peer0.org2*, *peer1.org2* và việc cập nhật các anchor peer trong kênh đó.

3.6.5 Cài đặt và khởi tạo Smart Contract

Cài đặt Smart Contract: sau khi đã phát triển smart contract *fotaContract*, nó sẽ được cài đặt vào các peer cần thiết trong mạng với tên *fota*. Thông qua smart contract, các application sau này sẽ dễ dàng truy cập vào ledger (sổ cái) tại các peer thông qua giao diện người dùng.

Khởi tạo Smart Contract: sau khi cài đặt vào các peer, để các thành phần khác được kết nối với kênh *mychannel* biết về smart contract *fota* vừa được cài đặt; ta phải khởi tạo nó trên kênh *mychannel*. Quá trình này cần cung cấp các thông tin chứng chỉ để chứng thực tại các peer đối với các tổ chức trên *mychannel* trong mạng. Thông qua đó các thành phần trên cùng *mychannel* đều biết về sự tồn tại của *fota* đã được cài đặt. Lưu ý rằng mặc dù mọi thành phần trên kênh hiện có thể truy cập vào *fota*, nhưng chúng không thể thấy logic chương trình của smart contract *fota*. Điều này vẫn được giữ riêng tư đối với những peer đã cài đặt nó. Về mặt khái niệm, điều này có nghĩa là chỉ có interface của smart contract được khởi tạo. Và, cài đặt một smart contract hiểu đơn giản là nó được physically hosted trên một peer, trong khi việc khởi tạo smart contract hiểu là nó được logically hosted trên kênh.

Chính sách chứng thực (Endorsement policy): Phần quan trọng nhất của thông tin tại thời điểm khởi tạo là một chính sách chứng thực. Nó mô tả các tổ chức nào phải phê duyệt các giao dịch trước khi chúng được các tổ chức khác chấp nhận vào bản sao của sổ cái. Định nghĩa kiểu như AND(Org1, Org2) hoặc OR(Org1, Org2).



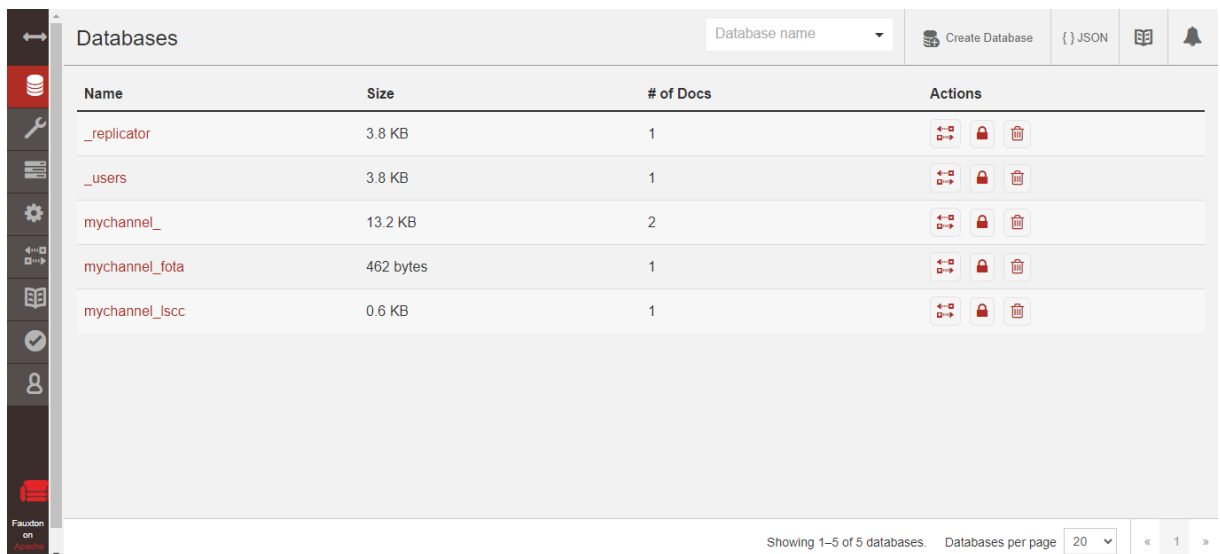
Hình 3.10 Kết quả trên Docker Engine khi khởi tạo *fota* cho *peer0.org1*

Ở hình 3.10, *dev-peer0.org1.example.com-fota-1.0.0* đối tượng *fota* đã được khởi tạo trên peer *peer0.org1*.

Ta có thể khởi tạo *fota* trên các peer đã được cài đặt *fotaContract*.

Đến đây, mạng Blockchain Network cho hệ thống đã có thể sử dụng. Thông qua giao diện dòng lệnh *cli*, ta có thể truy vấn cũng như xem các cấu trúc dữ liệu được lưu trữ trên cơ sở dữ liệu.

Truy cập vào cơ sở dữ liệu CouchDB, ta có thể thấy cơ sở dữ liệu của kênh *mychannel* (*mychannel_*) và của smart contract *fota* trên *mychannel* (*mychannel_fota*).

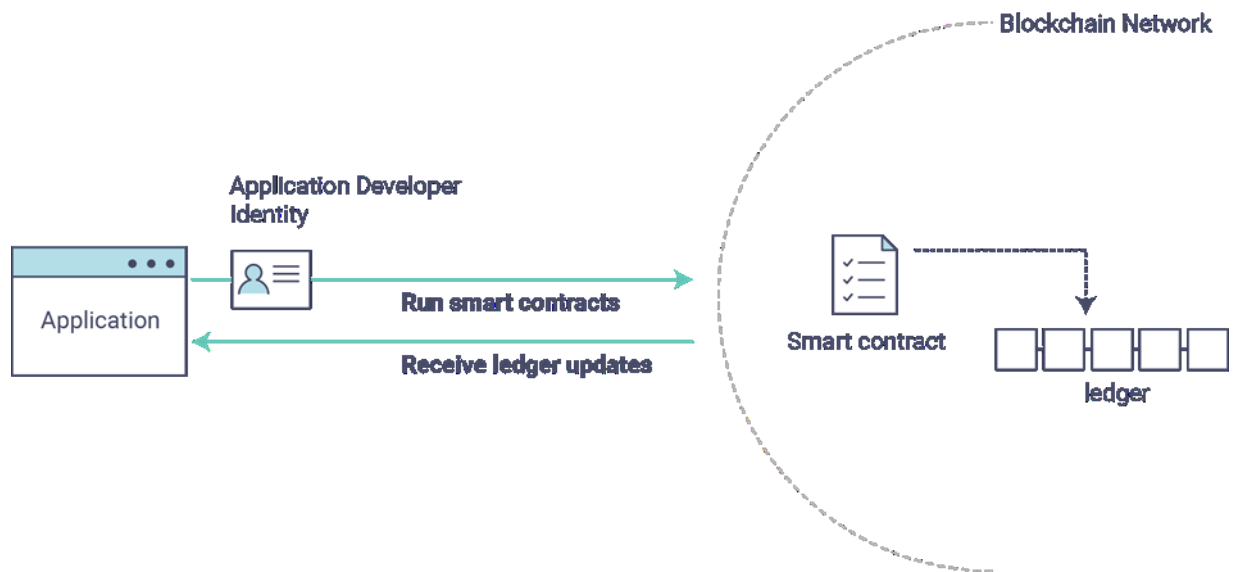


Hình 3.11 Giao diện các databases trên CouchDB

3.7 Triển khai dịch vụ web server (Vendor Service)

Trong Hyperledger Fabric, ứng dụng khách (Client Application) tương tác với mạng Hyperledger Fabric thông qua SDK (Software Development Kit). Hyperledger Fabric cung cấp SDK cho các ngôn ngữ lập trình khác nhau. Java và Node SDK được hỗ trợ chính thức. Vì Hyperledger Fabric là một nền tảng private blockchain, mọi người tham gia phải được cho phép, ủy quyền trước khi có thể tương tác với mạng. Nó được thực hiện bằng cách

cung cấp các chứng chỉ phù hợp do Tổ chức phát hành chứng chỉ (Certificate Authority hay CA).



Hình 3.12 Mô hình tương tác giữa Application với Blockchain Network

Bên trong ứng dụng khách là logic cách người dùng tương tác với mạng fabric và chaincode. Theo thiết kế Hyperledger Fabric, ứng dụng khách sẽ tương tác với Peer và Orderer, để chứng thực và tạo block. Access point (điểm truy cập) của Peer và Orderer cần được chỉ định trong ứng dụng khách. Ngoài ra, tên kênh và tên chaincode cũng được chỉ định. Một mạng fabric hỗ trợ nhiều kênh và nhiều chaincode.

Cuối cùng, khi ứng dụng khách thực hiện truy vấn hoặc gọi chaincode, hãy đảm bảo cung cấp đúng tên hàm và danh sách đối số nếu có.

3.7.1 Node SDK

Node SDK gồm 3 gói npm cho nền tảng Hyperledger Fabric, bao gồm:

- *fabric-ca-client* thành phần cho client trong Hyperledger Fabric. Component Fabric-ca cho phép các ứng dụng đăng ký Peers và người dùng ứng dụng để thiết lập danh tính đáng tin cậy trên mạng. Nếu mạng blockchain được định cấu hình các cơ quan cấp chứng chỉ tiêu chuẩn, ứng dụng không cần sử dụng gói này.
- *fabric-network* đóng gói các API để kết nối với mạng Fabric, gửi giao dịch và thực hiện các truy vấn đối với sổ cái.

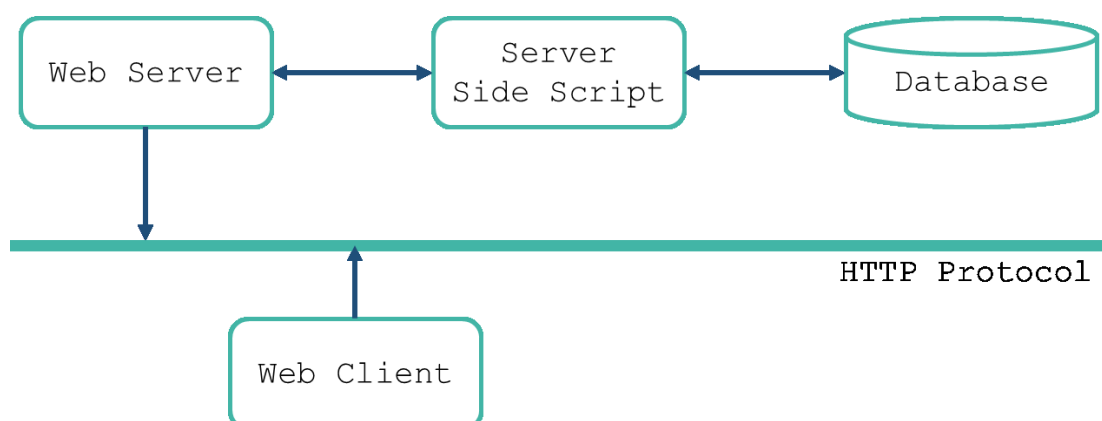
Ngoài ra còn các gói NPM phục vụ cho việc phát triển dịch vụ khác như:

- *express*: là một NodeJS framework cung cấp những tính năng cho ứng dụng web.
- *body-parser*: hỗ trợ lấy dữ liệu nhập vào từ web (như trong req.body).
- *js-sha256*: cung cấp thuật toán SHA-256.
- *multer*: khi một web client tải một tệp (upload file) lên web server, nó thường được gửi qua một biểu mẫu và được mã hóa dưới dạng dữ liệu multipart/form-data. Multer là một middleware cho NodeJS giúp dễ dàng xử lý dữ liệu multipart/form-data khi người dùng upload file.

Ứng dụng khách trong khóa luận này được biết đến thông qua tên Vendor Service.

3.7.2 Giới thiệu giao thức HTTP

HTTP là từ viết tắt của Hyper Text Transfer Protocol nghĩa là Giao thức truyền tải siêu văn bản, là một giao thức lớp ứng dụng cho các hệ thống thông tin siêu phương tiện phân tán, cộng tác. HTTP là nền tảng của truyền thông dữ liệu cho World Wide Web; một giao thức cho phép tìm nạp tài nguyên, chẳng hạn như HTML doc. Nó là nền tảng của bất kỳ sự trao đổi dữ liệu nào trên Web và cũng là giao thức giữa client (thường là các trình duyệt hay bất kỳ loại thiết bị, chương trình nào) và server (thường là các máy tính trên đám mây). Một doc hoàn chỉnh được tái tạo từ các doc con khác nhau được fetch – tìm nạp, chẳng hạn như văn bản, mô tả layout, hình ảnh, video, script,...



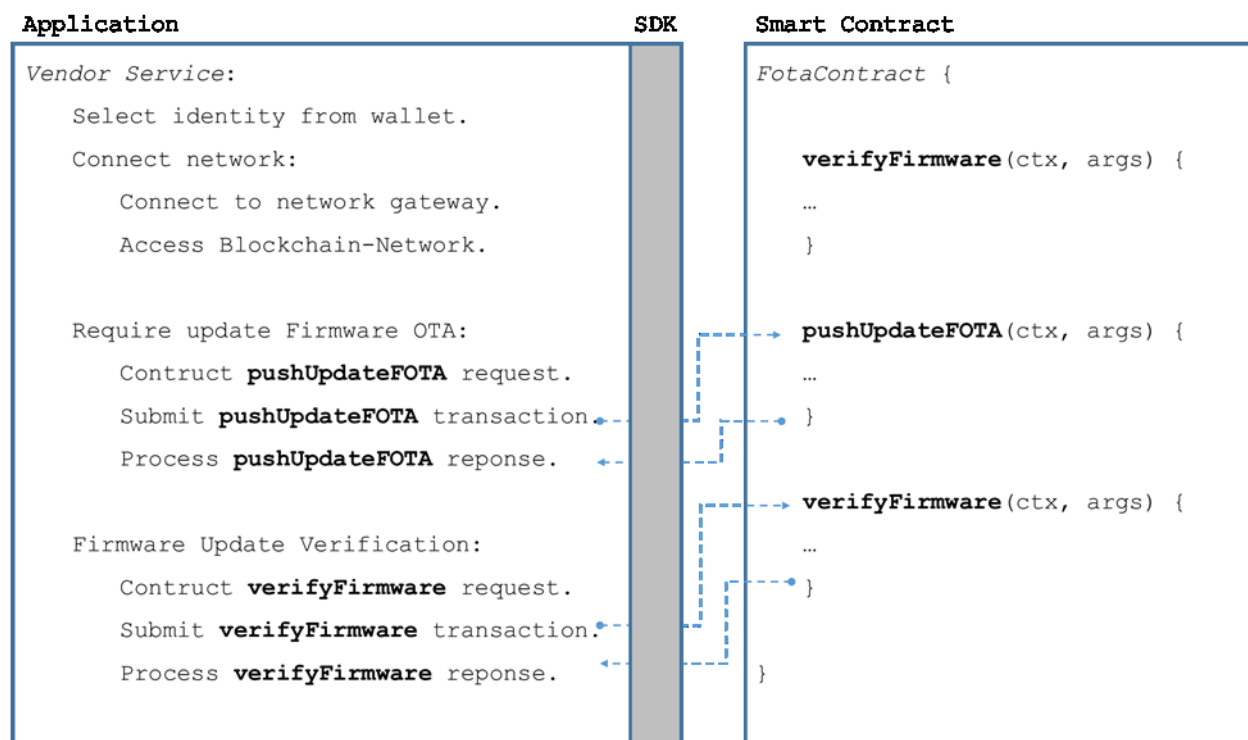
Hình 3.13 Cấu trúc cơ bản của một ứng dụng web

HTTP còn là 1 giao thức **Yêu cầu – Phản hồi** dựa trên cấu trúc Client – Server. Client và Server giao tiếp với nhau bằng cách trao đổi các message độc lập (trái ngược với 1 luồng

dữ liệu). Các message được gửi bởi client, thông thường là một trình duyệt web, được gọi là các yêu cầu và message được gửi bởi server như một sự trả lời, được gọi là phản hồi.

HTTP Request Method chỉ phương thức để được thực hiện trên nguồn được nhận diện bởi Request-URI (nội dung yêu cầu) đã cung cấp. Các phương thức phổ biến: GET, POST,...

3.7.3 Mô hình tương tác giữa Vendor Service và Smart Contract



Hình 3.14 Mô hình tương tác giữa Vendor Service với FotaContract

Vendor Service là dịch vụ cung cấp các API giúp tương tác với Blockchain Network thông qua việc thi hành các giao dịch (transaction) *push*, *verify* hoặc *query* với ledger trong smart contract bằng Node SDK. Mô hình tương tác giữa Vendor Service với Blockchain Network (thực chất là tương tác với *fotaContract*) dạng đơn giản được thể hiện qua hình 3.14.

Về tổng quan, một ứng dụng tương tác với mạng Blockchain Network cần phải theo 4 bước cơ bản được mô tả trong hình 3.13 để thực hiện các giao dịch:

- (1) Đăng nhập vào hệ thống bằng thông tin định danh (identity) từ ví (wallet) đã được đăng ký từ trước. Nếu chưa có, có thể thực hiện việc tạo thông tin định danh và thực hiện giao dịch đăng ký tài khoản.
- (2) Sau khi đăng nhập, sử dụng thông tin định danh ấy để truy cập vào Blockchain Network thông qua gateway là các peer trong mạng.

- (3) Đề nghị cập nhật firmware vào thiết bị IoT đã thiết lập trên hệ thống. Nếu chưa có, có thể thực hiện việc tạo thông tin thiết bị IoT và gửi firmware lên cơ sở dữ liệu trên hệ thống.
- (4) Thực hiện chứng thực quá trình cập nhật firmware từ xa.

3.7.4 Các thành phần chính trong Vendor Service

Các thành phần chính trong Vendor Service gồm:

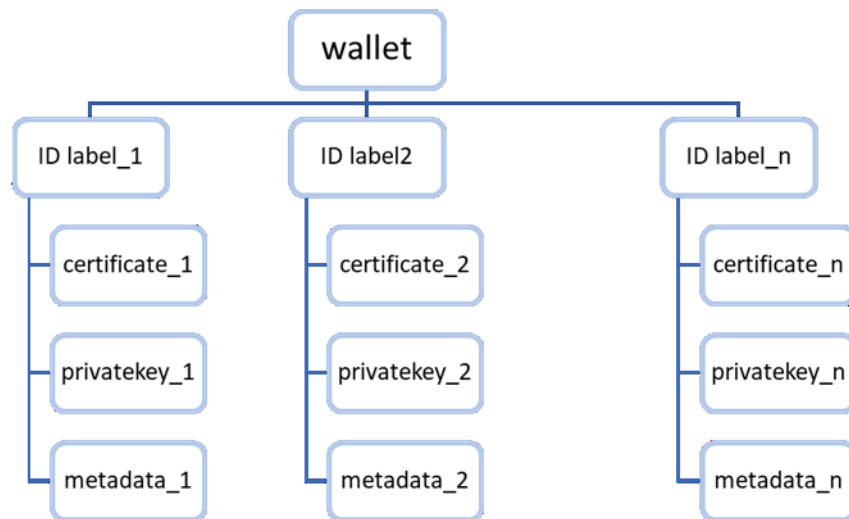
- *Wallet*: ví điện tử dùng để lưu trữ thông tin định danh của các Manager. Kiến trúc của ví theo cấu trúc thư mục với một thư mục *admin* được tạo khi khởi tạo ứng dụng khách và các thư mục lưu trữ thông tin định danh của các Manager gồm: thông tin chứng thực (certificate), khóa riêng tư (private key) và các dữ liệu của Manager (metadata) [9].
- *Network*: các Manager có thể truy cập vào các peer trong mạng thông qua thông tin định danh đã được tạo trong ví. Thông qua peer, truy cập vào smart contract được khởi tạo trên kênh để thực hiện các giao dịch [10].
- *Controllers*: các Manager giao tiếp với Vendor Service thực chất là thực hiện các API được xử lý thông qua bộ các controller có trong Vendor Service.
- *Views (Web Interface)*: giao diện web được thiết kế để các Manager dễ dàng giao tiếp.

3.7.4.1 Wallet

Với mỗi ứng dụng khách sẽ có một ví điện tử (wallet) dùng để lưu trữ các thông tin định danh. Ở Vendor Service, thông tin định danh của các Manager sẽ được lưu trữ ở hai dạng: dạng cấu trúc thư mục (chính là wallet) và dạng cơ sở dữ liệu thông qua cấu trúc *Manager* được định danh trong *fotaContract*. Khi các Manager muốn thực hiện một giao dịch nào đó cần đăng nhập vào hệ thống và thông qua xác thực chứng chỉ trong wallet cũng như thông tin đăng nhập trong cơ sở dữ liệu.

Cấu trúc thư mục wallet bao gồm các thư mục con chứa các thông tin định danh của các Manager. Mỗi thông tin định danh có một cấu trúc đặc trưng: một nhãn tên (label – trong cấu trúc *Manager* được định nghĩa là thuộc tính *managerId*) để phân biệt giữa các Manager

chứa một chứng chỉ (certificate), khóa riêng tư (private key) và các thông tin nền cho mạng (metadata).



Hình 3.15 Cấu trúc thư mục Wallet

3.7.4.2 Network

Các tương tác với mạng được thực hiện thông qua việc truy cập vào Gateway. Các ứng dụng khách kết nối đến một gateway và sau đó các tương tác mạng được quản lý bởi cấu hình gateway đã kết nối. Gateway sẽ thông qua các điểm truy cập là các đối tượng chaincode được khởi tạo trên kênh tại các peer trong mạng để thực hiện việc truy vấn các logic giao dịch được định nghĩa trong hợp đồng thông minh. Với hệ thống đang được xét đến, các Manager thông qua Vendor Service sẽ truy cập vào mạng thông qua một gateway bằng thông tin định danh khi đăng nhập. Sau đó gateway đó kết nối điểm truy cập trong mạng để thực hiện các giao dịch.

3.7.4.3 Controllers

Các controller được định nghĩa giúp hỗ trợ việc xử lý các API trong Vendor Service. Có bốn loại controller chính được định nghĩa:

- *loginController*: xử lý các API cho việc đăng nhập và đăng ký tài khoản Manager (tạo thông tin định danh và khởi tạo cấu trúc dữ liệu *Manager* trên World State CouchDB).
- *pushController*: xử lý các API cho việc khởi tạo các cấu trúc dữ liệu cần thiết cho các hoạt động trong hệ thống (khởi tạo cấu trúc dữ liệu *IoTSystem*, *FirmwareOTA*, *UpdateFOTA* trên World State CouchDB).

- *downloadController*: xử lý các API cho việc lưu trữ và tải file binary (BIN) firmware sau khi đã push firmware lên hệ thống (thông qua việc khởi tạo cấu trúc dữ liệu *FirmwareOTA*), các firmware sẽ được lưu trữ ở dạng thư mục trong máy chủ ứng dụng khách. Các thiết bị IoT khi kiểm tra có yêu cầu cập nhật sẽ truy vấn API để tải firmware này về thiết bị để tiến hành quá trình xác thực yêu cầu.
- *handleRequireController*: xử lý các API cho quá trình xác thực yêu cầu cập nhật firmware. Đây là controller quan trọng nhất vì đây là nơi tiến hành vấn đề đặt ra cho toàn bộ hệ thống. Có ba API được định nghĩa trong controller này là kiểm tra yêu cầu (*checkRequire*), xác thực yêu cầu (*verifyRequire*) và ghi nhận kết quả yêu cầu (*recordRequire*). Các API này sẽ được giới thiệu chi tiết ở những mục sau.

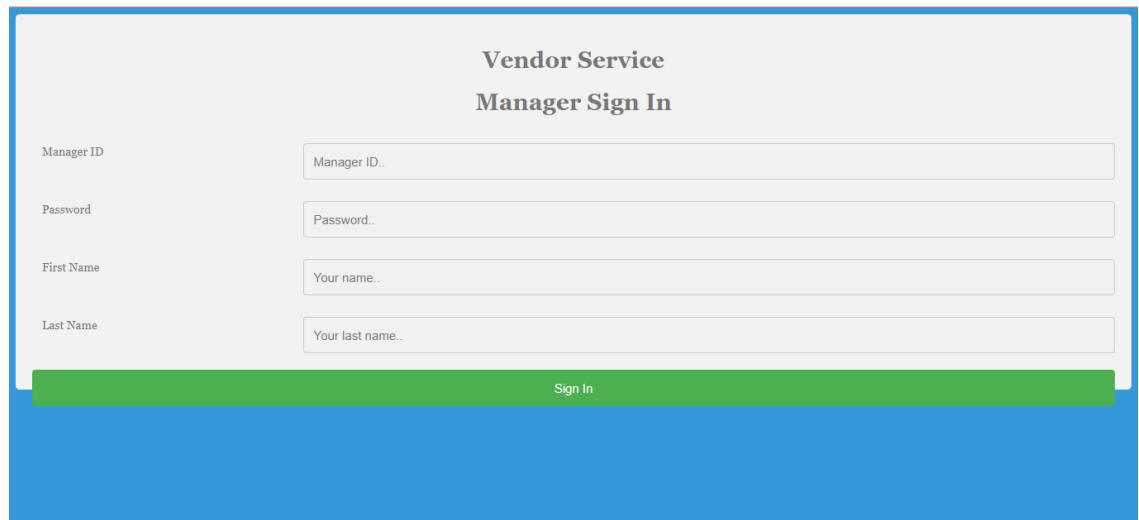
3.7.4.4 Views

Các trình duyệt web nhận tài liệu HTML từ web server để cung cấp các giao diện web cho phép Manager thực hiện đăng ký, đăng nhập và các giao dịch cần thiết (như là khởi tạo yêu cầu cập nhật firmware). Một số giao diện chính trong Vendor Service:

- Giao diện đăng nhập (Login): hỗ trợ đăng nhập tài khoản Manager thông qua việc xác thực tài khoản (được xử lý trong logic giao dịch *verifyManager*). Thông tin đăng nhập bao gồm thuộc tính *managerId* để xác thực thông tin định danh trong wallet và thuộc tính *password* để xác thực thông tin cấu trúc dữ liệu Manager được khởi tạo trên World State CouchDB. Ngoài ra, nếu chưa có tài khoản, các Manager có thể tạo thông qua việc chọn “Create your Account”.

Hình 3.16 Giao diện đăng nhập (Login)

- Giao diện đăng ký (Signin): hỗ trợ đăng ký tài khoản Manager thông qua việc khởi tạo cấu trúc dữ liệu *Manager* và đăng ký thông tin định danh trong wallet.



The screenshot displays a web form titled "Vendor Service Manager Sign In". The form is set against a light gray background with a blue border. It contains four input fields: "Manager ID" with a placeholder "Manager ID..", "Password" with a placeholder "Password..", "First Name" with a placeholder "Your name..", and "Last Name" with a placeholder "Your last name..". Below these fields is a green "Sign In" button. The entire form is positioned on a blue background.

Hình 3.17 Giao diện đăng ký (Signin)

- Giao diện chính (Home): gồm có bốn mục chính là tạo hệ thống IoT (Push IoT System), đăng tải file binary firmware (Push Firmware), hiển thị danh sách các firmware đã được đăng tải (Firmware List) và tạo yêu cầu cập nhật firmware (Push Update).
 - *Push IoT System*: giúp định nghĩa cho các thiết bị IoT trong hệ thống, được định danh thông qua địa chỉ MAC (*macAddress*) và loại (*deviceType*) của các thiết bị và ghi nhận thêm phiên bản firmware hiện tại được triển khai trên thiết bị.
 - *Push Firmware*: cung cấp việc đăng tải file binary firmware từ bộ nhớ cục bộ. Các file firmware được định danh bởi thuộc tính phiên bản firmware (*firmwareVersion*) và loại thiết bị (*deviceType*) tương thích cho firmware đó.
 - *Firmware List*: hỗ trợ hiển thị các file firmware đã được đăng tải lên server.
 - *Push Update*: cung cấp việc khởi tạo yêu cầu cập nhật firmware với phiên bản được định sẵn cho thiết bị IoT định danh bởi địa chỉ MAC.

The screenshot displays the 'Vendor Service' web interface, which is organized into four distinct sections, each with its own set of input fields and a 'Submit' button.

- Push IoT System:** This section contains three input fields labeled 'Latest Version', 'MAC Address', and 'Device Type'. Below these fields is a green 'Submit' button.
- Push Firmware:** This section includes input fields for 'Firmware Version' and 'Device Type'. Additionally, there is a 'File' section with a 'Choose File' button and the text 'No file chosen'. A green 'Submit' button is located at the bottom of this section.
- Firmware List:** This section is currently empty, featuring a 'Reset' link above the section title.
- Push Update:** This section contains two input fields, 'Firmware Version' and 'MAC Address', followed by a green 'Submit' button.

Hình 3.18 Giao diện chính (Home)

3.7.5 Xử lý giao dịch trong *Vendor Service*

Trong mô hình tương tác giữa *Vendor Service* với *FotaContract* đã được mô tả ở mục trên, để xử lý một giao dịch nào đó cần thực hiện theo ba bước:

- (1) Construct request: sau khi đã truy cập vào mạng, cần truy cập đến đối tượng *fota* (đã được khởi tạo khi thiết lập cấu hình mạng) để sử dụng các logic giao dịch của *fotaContract*. Đồng thời cần chuẩn bị các tham số cần thiết (*args*) cho logic giao dịch đó.
- (2) Submit/evaluate transaction: sử dụng các tham số đã chuẩn bị để truyền vào logic giao dịch được định trong smart contract. Tùy vào loại giao dịch, nếu giao dịch gây ra thay đổi trạng thái World State thì sử dụng *submitTransaction()*, và ngược

lại nếu loại giao dịch chỉ dùng để truy vấn thì sử dụng phương thức sau *evaluateTransaction()* [11].

- (3) Process response: sau khi xử lý xong giao dịch, cần xử lý phản thông tin hồi đáp sau giao dịch. Tùy thuộc vào từng loại giao dịch mà kết quả trả về cần xử lý khác nhau.

Lưu ý: việc giao tiếp giữa Vendor Service và *fotaContract* sẽ được thông qua bởi các tham số được lưu trữ và vận chuyển dưới dạng JSON.

3.7.6 Các API trong Vendor Service

Các API được xử lý trong các *Controller* để phục vụ cho các mục đích giao tiếp giữa người dùng (các *Manager*) hay *fotaContract* với Vendor Service.

Định tuyến	Loại yêu cầu	Nội dung yêu cầu (req)
/pushFirmware	POST	BIN file, firmware version, device type
/pushUpdate	POST	JSON
/checkRequire/:macAddress	GET	MAC Address
/files/:firmwareVersion	GET	firmware version
/verifyRequire/: dataVerify	GET	dataVerify = txid + firmware hash
/recordRequire/:dataRecord	GET	dataRecord = txid + status device
/queryRequire/:txid	GET	txid

Bảng 3.1 Các API chính trong Vendor Service

Bảng 3.1 mô tả các API chính trong Vendor Service, thông qua thành phần *Network* để giao tiếp với *fotaContract* để xử lý các giao dịch cần thiết. Các API phục vụ trong việc giao tiếp giữa Vendor Service với thiết bị IoT sẽ tham gia trực tiếp vào quá trình xác thực yêu cầu cập nhật firmware. Ngoài các API được liệt kê trên, còn có các API khác hỗ trợ cho hệ thống như đăng kí và đăng nhập cho tài khoản của Manager, ...

3.8 Lập trình nhúng cho thiết bị IoT

Như đã giới thiệu trong mục 3.1.1, thiết bị IoT được giới thiệu là NodeMCU ESP32. Thiết bị được triển khai trên môi trường Arduino IDE. Thông qua chuẩn WiFi, thiết bị sử dụng các API được cung cấp bởi Vendor Service để xử lý các yêu cầu cập nhật firmware từ *Manager*.

3.8.1 Giới thiệu về lập trình nhúng

Lập trình nhúng là một thuật ngữ lập trình dùng để chỉ đến một hệ thống có khả năng tự trị và nó thường được nhúng vào trong một môi trường hoặc một hệ thống mẹ bất kỳ nào đó. Đây là các hệ thống tích hợp cả một phần mềm và phần cứng.

Mục đích chủ yếu lập trình nhúng chính là phục vụ các bài toán chuyên dụng trong các lĩnh vực công nghiệp, tự động hóa điều khiển và truyền tin. Thông thường, hệ thống nhúng sẽ được thiết kế để thực hiện các chức năng chuyên trách hoặc riêng biệt nào đó.

Bởi vì đây là lập trình chỉ được xây dựng cho một nhiệm vụ nhất định nên người tạo lập thường sẽ tối ưu hóa được nó nhằm tối thiểu kích thước và chi phí cho lập trình. Hệ thống nhúng này thường rất đa dạng và phong phú về chủng loại.

Thành phần cơ bản của hệ thống nhúng:

- Rom: chứa chương trình, các dữ liệu được fix hoặc những constant data. Hiện nay, thì đa số các hệ thống này đều sử dụng EEPROM hoặc FLASH để thay thế cho ROM bởi vì: chúng có chức năng update chương trình mới, có khả năng ghi xóa.
- RAM: là thành phần hỗ trợ lưu chương trình thực thi và các biến tạm.
- MCU: là bộ xử lý hỗ trợ tính toán trung tâm.
- Ngoài ra, còn một vài thiết bị ngoại vi khác như: ADC, DAC, I2C, UART,...

3.8.2 Giới thiệu các thư viện hỗ trợ

Trong Arduino IDE, cần bổ sung các thư viện để hỗ trợ cho các chức năng cần trong hệ thống. Một số thư viện chính được sử dụng:

- Thư viện *HTTPClient*: cung cấp chức năng truy vấn các API được cấp bởi Vendor Service,

- Thư viện *ArduinoJson*: cung cấp chức năng serialize và deserialize các file JSON,
- Thư viện *Seeed_mbedtls*: cung cấp chức năng tính hàm băm SHA256 [12],
- Thư viện *Update*: cung cấp chức năng cập nhật firmware cho thiết bị.

3.8.3 Dịch vụ cập nhật OTA trên thiết bị NodeMCU ESP32

Cập nhật firmware OTA là tiến trình tải firmware mới vào ESP module thay vì sử dụng cổng Serial. Tính năng này thực sự rất hữu dụng trong nhiều trường hợp giới hạn về kết nối vật lý đến ESP Module. Trong tất cả các trường hợp, thì Firmware hỗ trợ OTA phải được nạp lần đầu tiên qua cổng Serial, nếu mọi thứ hoạt động trơn tru, logic ứng dụng OTA hoạt động đúng thì có thể thực hiện việc cập nhật firmware thông qua OTA.

Do việc bảo mật khi ESP32 thực thi OTA sử dụng Digest-MD5 để chứng thực việc tải firmware lên. Đơn giản là đảm bảo tính toàn vẹn của firmware bằng việc tính MD5. Sẽ không có đảm bảo an ninh đối với quá trình cập nhật OTA bị tấn công. Nó phụ thuộc vào nhà phát triển đảm bảo việc cập nhật được phép từ nguồn hợp pháp, đáng tin cậy. Khi cập nhật hoàn tất, thiết bị sẽ khởi động lại và thực thi firmware mới. Nhà phát triển phải đảm bảo ứng dụng thực trên thiết bị phải được tắt và khởi động lại một cách an toàn.

Có ba cách để cập nhật OTA cho ESP32:

- Sử dụng Arduino IDE: tận dụng trực tiếp Arduino IDE sẵn có, thay lựa chọn Port > COM bằng COM > IP.
- Sử dụng Web Browser: ESP32 tự tạo một web server để cung cấp giao diện web, qua đó có thể chọn đường dẫn file binary firmware mới.
- Sử dụng giao thức HTTP: cập nhật firmware cho ESP32 từ một web server.

Trong phạm vi khóa luận này, sử dụng HTTP Client để thực hiện quá trình cập nhật firmware từ web server Vendor Service.

3.8.4 Quá trình truy vấn API và sử dụng thuật toán băm

3.8.4.1 Quá trình truy vấn API

Thiết bị NodeMCU ESP32 hỗ trợ chuẩn WiFi, qua đó thiết bị dễ dàng truy vấn API thông qua giao thức HTTP để giao tiếp với Vendor Service. Việc truy vấn này được hỗ trợ bởi

thư viện *HTTPClient*. Hầu như nội dung phản hồi của các API thuộc dạng cấu trúc JSON. Nên cần sử dụng thư viện *ArduinoJson* cho việc phân tích nội dung phản hồi thành lại cấu trúc JSON để sử dụng thuận tiện hơn.

3.8.4.2 Sử dụng thuật toán băm

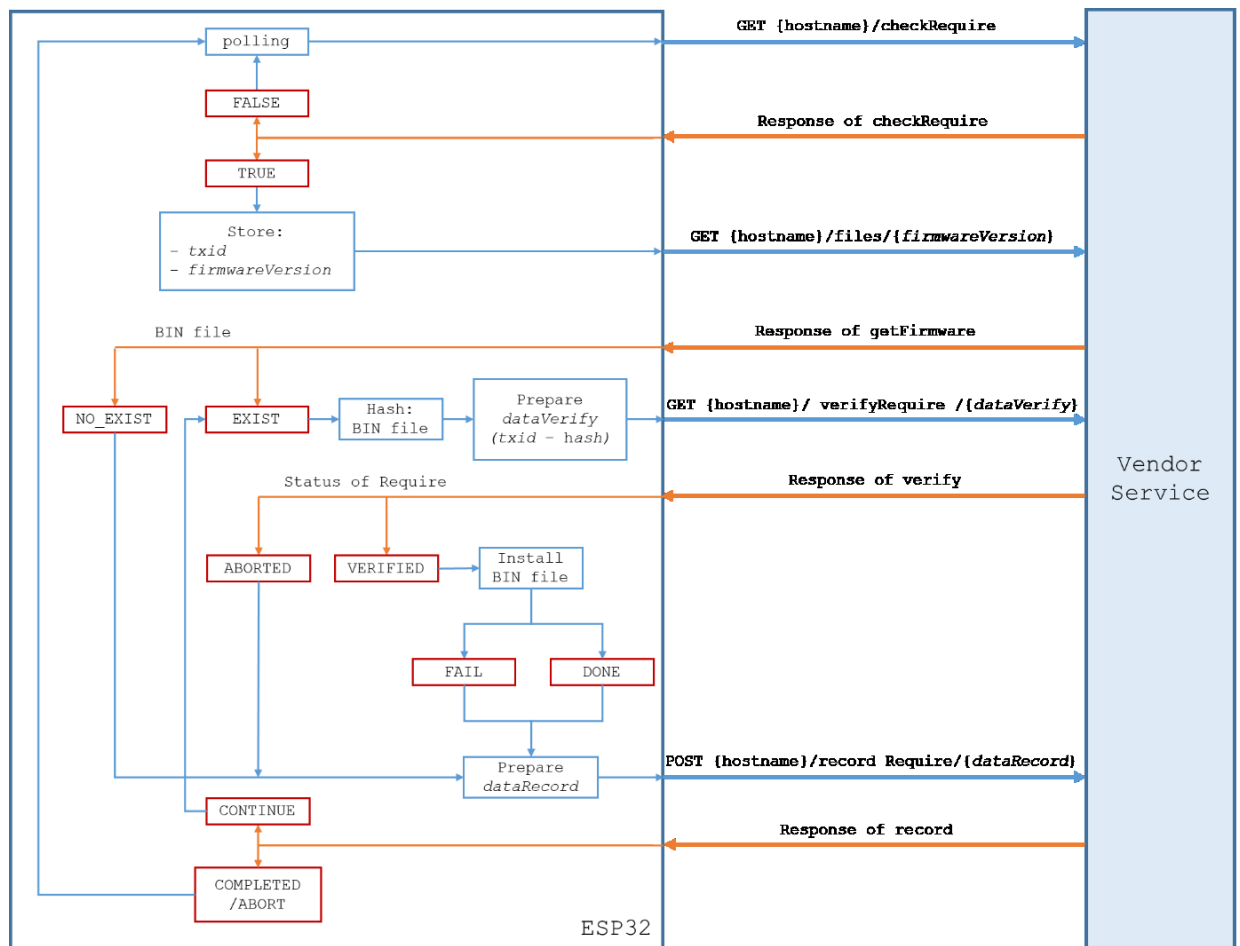
Thuật toán băm (hàm băm) là một giải thuật nhằm sinh ra các giá trị băm (hash value) tương ứng cho mỗi khối dữ liệu đầu vào. Giá trị băm có vai trò như một thông tin định danh để phân biệt các khối dữ liệu. Do hàm băm là một hàm một chiều nên không thể giải mã từ một giá trị băm ra thành khối dữ liệu ban đầu. Qua đó việc sử dụng thuật toán băm giúp đảm bảo tính toàn vẹn của khối dữ liệu đầu vào. Thuật toán băm SHA256 là một giải thuật băm nhằm sinh ra giá trị băm có độ dài 256 bit.

Khi đăng tải file binary firmware lên server, file ấy sẽ được cho qua hàm băm để ghi nhận lại giá trị băm thông qua thuộc tính *firmwareHash* trong cấu trúc dữ liệu *firmwareOTA*. Khi thiết bị truy vấn để tải file binary firmware, thiết bị hoàn toàn không thể xác thực tính toàn vẹn và tính chính xác của khối dữ liệu vừa tải xuống. Nên dựa vào thuật toán băm SHA256, thiết bị có thể xác thực chính xác với chi phí thực hiện không quá lớn. Việc sử dụng thuật toán băm SHA256 được hỗ trợ bởi thư viện *Seed_mbedtls*.

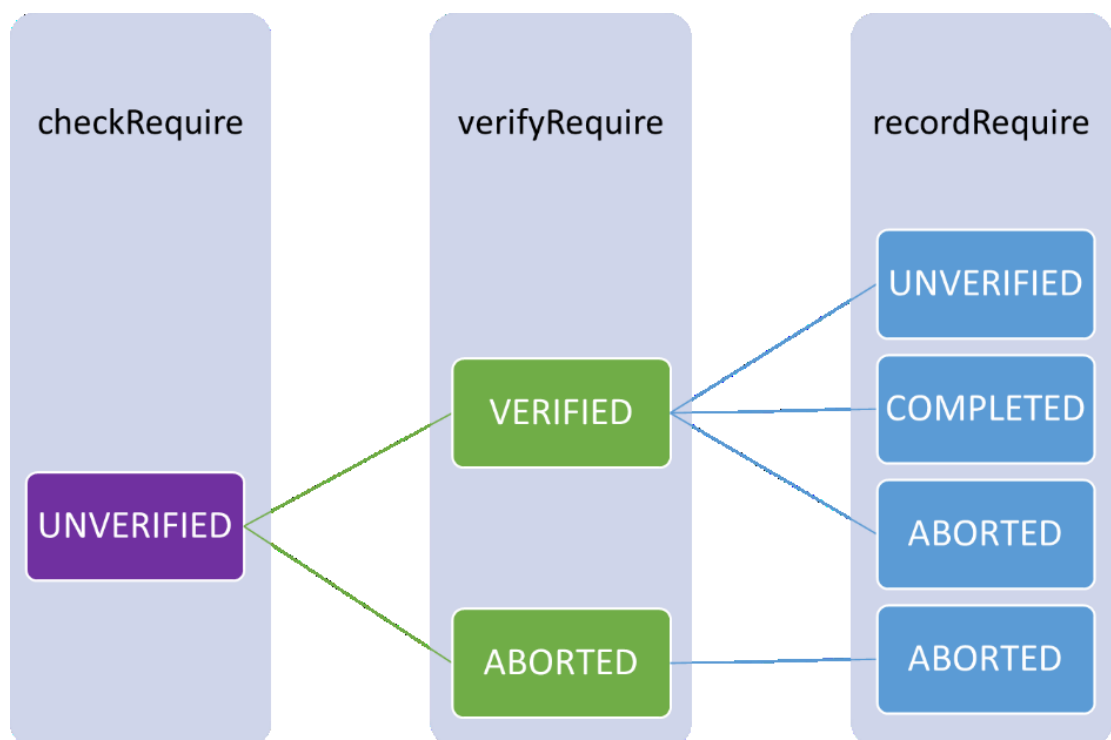
3.9 Quá trình xác thực yêu cầu cập nhật firmware

Thiết bị IoT ở trong trạng thái polling (kiểm tra định kỳ) yêu cầu cập nhật từ phía Vendor Service. Khi có yêu cầu cập nhật, thiết bị sẽ tiến hành quá trình xử lý yêu cầu cập nhật thông qua quá trình giao tiếp các API với Vendor Service. Quá trình xử lý được mô tả chi tiết trong hình 3.19.

Quá trình xác thực yêu cầu cập nhật firmware tập trung xảy ra ở việc cập nhật thuộc tính *status* của cấu trúc dữ liệu *UpdateOTA*. Trong hình 3.20 mô tả quá trình cập nhật thuộc tính *status* qua ba bước để xử lý yêu cầu cập nhật firmware: (1) kiểm tra yêu cầu (*checkRequire*), (2) xác thực yêu cầu (*verifyRequire*) và (3) ghi nhận kết quả yêu cầu (*recordRequire*). Lưu ý: các API này sẽ được truy vấn bởi các thiết bị IoT trong hệ thống.



Hình 3.19 Quá trình xử lý yêu cầu cập nhật firmware ở thiết bị IoT



Hình 3.20 Cập nhật thuộc tính *status* của cấu trúc dữ liệu *UpdateOTA*

3.9.1.1 Kiểm tra yêu cầu (*checkRequire*)

Khi có một yêu cầu cập nhật được khởi tạo (thông qua API */pushUpdate*), đồng nghĩa một đối tượng *updateFOTA* được khởi tạo trên cơ sở dữ liệu World State CouchDB và trạng thái của yêu cầu sẽ được ghi nhận trong thuộc tính *status* với giá trị “*unverified*”. Thiết bị IoT sẽ luôn trong trạng thái polling theo chu kỳ được định sẵn để kiểm tra xem có yêu cầu cập nhật firmware cho thiết bị đó (thông qua việc kiểm tra địa chỉ MAC được ghi nhận trong thuộc tính *macAddress* của các đối tượng *updateFOTA*) từ phía Vendor Service hay không thông qua API */checkRequire*. Nếu có yêu cầu, Vendor Service sẽ trả về file JSON chứa các thông tin của yêu cầu cập nhật firmware chưa được xác thực (gồm transaction ID được ghi nhận trong thuộc tính *txid* và phiên bản firmware đã được chỉ định thông qua giá trị thuộc tính *firmwareVersion*).

Định tuyến (route)	<i>/checkRequire/:macAddress</i>
Loại yêu cầu (request type)	GET
Nội dung yêu cầu (request body)	Địa chỉ MAC của thiết bị truy vấn
Nội dung phản hồi (response body)	{ <i>"response"</i> : “TRUE”, <i>"require"</i> : {chứa <i>txid</i> và <i>firmwareVersion</i> } }
	{ <i>"response"</i> : “FALSE”, <i>"require"</i> : “null”}
Logic giao dịch được sử dụng	<i>queryAllByPartialCompositeKey</i>

Bảng 3.2 Thông tin chung của API *checkRequire*

Vendor Service sẽ thực hiện giao dịch *queryAllByPartialCompositeKey* để truy vấn các đối tượng *updateFOTA* đã được lưu trữ trên World State. Sau đó duyệt các đối tượng phù hợp với yêu cầu (thuộc tính *status* phải mang giá trị “*unverified*” và thuộc tính *macAddress* phải trùng giá trị nội dung yêu cầu của API). Nếu có nhiều hơn một yêu cầu, hệ thống sẽ thực hiện tuần tự theo thời gian khởi tạo (được ghi nhận trong thuộc tính *timestampSend*). Sau đó sẽ tạo nội dung phản hồi gồm một thông tin xác nhận *response* mang giá trị “*TRUE*”

nếu có yêu cầu phù hợp và ngược lại là “*FALSE*” cùng thông tin của yêu cầu đó trong trường *require*.

3.9.1.2 Xác thực yêu cầu (*verifyRequire*)

Sau khi thiết bị đã nhận được yêu cầu, thiết bị sẽ tải file binary firmware thông qua API */files/:firmwareVersion* với nội dung yêu cầu là phiên bản firmware được ghi nhận từ trường *require* trong phản hồi của API */checkRequire*. Sau khi nhận được file firmware, thiết bị sẽ tiến hành thực hiện thuật toán băm file để lấy giá trị hash của file (*hash*) sau đó sẽ truy vấn API */verifyRequire/:dataVerify* để thực hiện việc xác thực yêu cầu với nội dung *dataVerify* gồm *txid* được ghi nhận từ trường *require* trong phản hồi của API */checkRequire* và giá trị hash của file.

Định tuyến (route)	<i>/verifyRequire/:dataVerify</i>
Loại yêu cầu (request type)	GET
Nội dung yêu cầu (request body)	<i>dataVerify</i> = <i>txid</i> + <i>hash</i>
Nội dung phản hồi (response body)	{"response": “VERIFIED”}
	{"response": lý do không thành công}
Logic giao dịch được sử dụng	<i>verifyRequire</i>

Bảng 3.3 Thông tin chung của API *verifyRequire*

Vendor Service sẽ thực hiện giao dịch *verifyRequire* để xác thực file firmware thiết bị đã tải xuống thông qua việc kiểm chứng tính toàn vẹn của giá trị *hash* nhận được từ nội dung yêu cầu với giá trị của thuộc tính *firmwareHash* được lưu trữ trong đối tượng *firmwareOTA* được lưu trữ trên World State. Ở đây việc xác thực sẽ xảy ra bên trong *fotaContract*, dựa vào *txid* đã nhận, hệ thống sẽ truy vấn đến đối tượng *updateFOTA* tương ứng và kiểm tra thuộc tính *firmwareHash* thông qua đối tượng *firmwareOTA* được định danh bằng thuộc tính *firmwareVersion*. Nếu như trùng khớp giá trị *hash* với *firmwareHash*, nội dung phản hồi sẽ trả về “VERIFIED” và ngược lại sẽ trả về lý do xác thực không thành công (không tồn tại yêu cầu đó trên World State: “NO_MATCH_TXID”, không tồn tại firmware được

ghi nhận trong yêu cầu đó: “NO_HAVE_FOTA”, không trùng giá trị hash: “NO_MATCH_HASH”).

Và lúc này thuộc tính *status* cũng sẽ được cập nhật lại giá trị cho phù hợp: “verified” nếu xác thực thành công và ngược lại “aborted” nếu không thành công đồng thời sẽ tăng số lần truy cập thất bại vào yêu cầu này ở thuộc tính *failedAttempts*.

3.9.1.3 Ghi nhận kết quả yêu cầu (*recordRequire*)

Sau khi xác thực thành công, thiết bị sẽ tiến hành cài đặt firmware ấy. và sau đó sẽ ghi nhận lại kết quả cài đặt thông qua việc truy vấn API */recordRequire/:dataRecord* với nội dung yêu cầu là *dataRecord* gồm *txid* và trạng thái của thiết bị (*statusDevice*).

Định tuyến (route)	<i>/recordRequire/:dataRecord</i>
Loại yêu cầu (request type)	GET
Nội dung yêu cầu (request body)	<i>dataRecord</i> = <i>txid</i> + “DONE”
	<i>dataRecord</i> = <i>txid</i> + “FAIL”
	<i>dataRecord</i> = <i>txid</i> + “NO_EXIST”
Nội dung phản hồi (response body)	{“response”: “COMPLETED”}
	{“response”: “CONTINUE”}
	{“response”: “ABORTED”}
Logic giao dịch được sử dụng	<i>recordRequire</i>

Bảng 3.4 Thông tin chung của API *recordRequire*

Có ba trường hợp sẽ xảy ra khi xử lý trạng thái của thiết bị *statusDevice*:

- “DONE”: thiết bị cài đặt firmware thành công:
 - Nội dung phản hồi: “COMPLETED”,
 - Thuộc tính *status* sẽ được cập nhật với giá trị “completed”.

- “FAIL”: thiết bị cài đặt firmware thất bại:
 - Nội dung phản hồi: “CONTINUE”,
 - Thuộc tính *status* sẽ được cập nhật với giá trị “*unverified*” đồng thời sẽ tăng số lần truy cập thất bại vào yêu cầu này ở thuộc tính *failedAttempts*.
- “NO_EXIST”: thiết bị không thể truy vấn được file binary firmware từ API:
 - Nội dung phản hồi: “ABORTED”,
 - Thuộc tính *status* sẽ được cập nhật với giá trị “*aborted*” đồng thời sẽ tăng số lần truy cập thất bại vào yêu cầu này ở thuộc tính *failedAttempts*.

CHƯƠNG 4 KẾT QUẢ THỰC HIỆN

4.1 Mục tiêu đề ra cho khóa luận và kết quả đạt được sau khi thực hiện

Tên đề tài: ứng dụng private blockchain network vào hệ thống cập nhật firmware từ xa cho thiết bị IoT.

Mục tiêu	Kết quả đạt được	
Hiểu về quá trình cập nhật firmware từ xa	Hiểu được về định nghĩa quá trình cập nhật firmware từ xa.	
	Hiểu được cách cập nhật firmware từ xa cho ESP32 (đặc biệt là phương pháp cập nhật thông qua giao thức HTTP).	
	Biết về cách thực hiện việc cập nhật firmware cho thiết bị.	
Hiểu về Private Blockchain Network dựa trên nền tảng Hyperledger Fabric	Biết về Blockchain Network và các thành phần chính trong mạng Blockchain (Blockchain, Ledger, Consensus,...).	
	Biết về Private Blockchain Network và đặc điểm của nó.	
	Biết về nền tảng Hyperledger Fabric.	
	Biết về các thành phần chính trong mạng.	
	Biết về Sh script để tìm hiểu cách xây dựng mạng.	
	Biết về file YAML để thiết lập cấu hình mạng.	
	Docker	Biết về nền tảng Docker và các thành phần cơ bản.
		Biết sử dụng Docker trong việc triển khai mạng.
	Smart Contract	Biết về Smart Contract trong Blockchain Network
		Biết về cách lập trình Smart Contract bằng ngôn ngữ NodeJS phục vụ cho mạng.

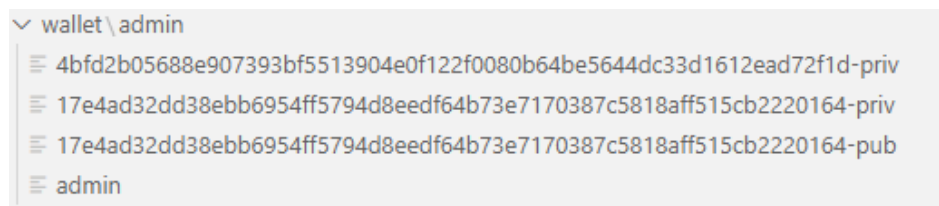
		Biết cách vận hành của logic giao dịch trong Smart Contract và kết quả của việc thực hiện.
		Biết xây dựng các cấu trúc dữ liệu hỗ trợ cho việc vận hành các logic giao dịch.
		Biết về kiến trúc Ledger trong HF.
		Biết về cơ sở dữ liệu CouchDB thông qua việc triển khai thành phần World State trong Ledger.
	Client Application	Hiểu về việc triển khai một ứng dụng khách (Client Application) cho mạng.
		Hiểu về kết nối các điểm truy cập mạng thông qua các gateway được cung cấp bởi HF.
		Hiểu về các truy vấn, khởi tạo cũng như cập nhật đối tượng lưu trữ trên World State.
Hiểu về lập trình web với ngôn ngữ Javascript (NodeJS)	Biết cơ bản các yêu cầu trong sử dụng ngôn ngữ Javascript (NodeJS).	
	Biết về NPM để sử dụng các thư viện hỗ trợ trong quá trình xây dựng web.	
	Biết cách xây dựng các API phục vụ cho các mục đích cần thiết.	
	Biết sử dụng các thẻ cơ bản HTML để xây dựng giao diện web.	
Hiểu về lập trình nhúng với ngôn ngữ C++	Biết về nền tảng và mục đích của việc lập trình nhúng.	
	Biết về lập trình nhúng với ngôn ngữ C++ để tạo các file binary firmware phù hợp cho ESP32 để phục vụ cho hệ thống.	
Hiểu về các điều kiện môi trường để phát triển và	Cơ bản nhận biết được việc xử lý các Container trong Docker để điều khiển các luồng vận hành trong mạng.	
	Sử dụng thành thạo Visual Studio Code để quản lý source code.	

triển khai hệ thống	Sử dụng thành thạo Arduino IDE để lập trình firmware cho ESP32.
Hiểu các ngôn ngữ hỗ trợ hoặc được sử dụng trong bài nghiên cứu	Hiểu về ngôn ngữ C++.
	Hiểu về ngôn ngữ Javascript (NodeJS).
	Hiểu về ngôn ngữ HTML.
	Hiểu về ngôn ngữ YAML.
	Hiểu về ngôn ngữ Sh/Bash.

Bảng 4.1 Mục tiêu đề ra và kết quả đạt được

4.2 Kết quả khi đăng nhập vào hệ thống

Sau khi triển khai dịch vụ Vendor Service, cần đăng kí tài khoản admin cho ứng dụng này trong mạng để xác thực dịch vụ này được cung cấp chứng chỉ để truy cập mạng. Thông tin định danh này được ghi nhận trong Wallet.



Hình 4.1 Thông tin định danh admin trong wallet

4.2.1 Đăng kí tài khoản trên hệ thống

Khi một Manager muốn tham gia, cần có một tài khoản để đăng nhập vào hệ thống. Ở giao diện chính đăng nhập, nếu chưa có tài khoản nào, Manager có thể đăng ký tài khoản thông qua việc chọn vào nút “Create your Account”.

Sau đó giao diện đăng ký sẽ xuất hiện. Ở giao diện đăng ký, Manager cần điền những thông tin như:

- Manager ID: thông tin định danh của Manager trên cơ sở dữ liệu,
- Password: thông tin bảo mật khi đăng nhập,
- First Name và Last Name: thông tin cá nhân của Manager.

Vendor Service
Manager Sign In

Manager ID: 1720081

Password:

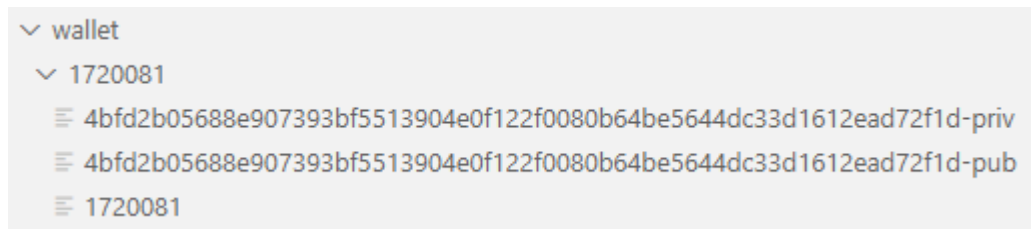
First Name: Thien Hao

Last Name: Nguyen Trieu

Sign In

Hình 4.2 Đăng ký tài khoản Manager với ID 1720081

Sau khi đăng ký, CA sẽ cấp thông tin định danh cho tài khoản vừa đăng ký và ghi nhận lại vào wallet.



Hình 4.3 thông tin định danh của Manager ID 1720081 trong wallet

Đồng thời một đối tượng thuộc cấu trúc dữ liệu *Manager* sẽ được khởi tạo và ghi nhận vào cơ sở dữ liệu World State trên CouchDB.

mychannel_fota

Document ID

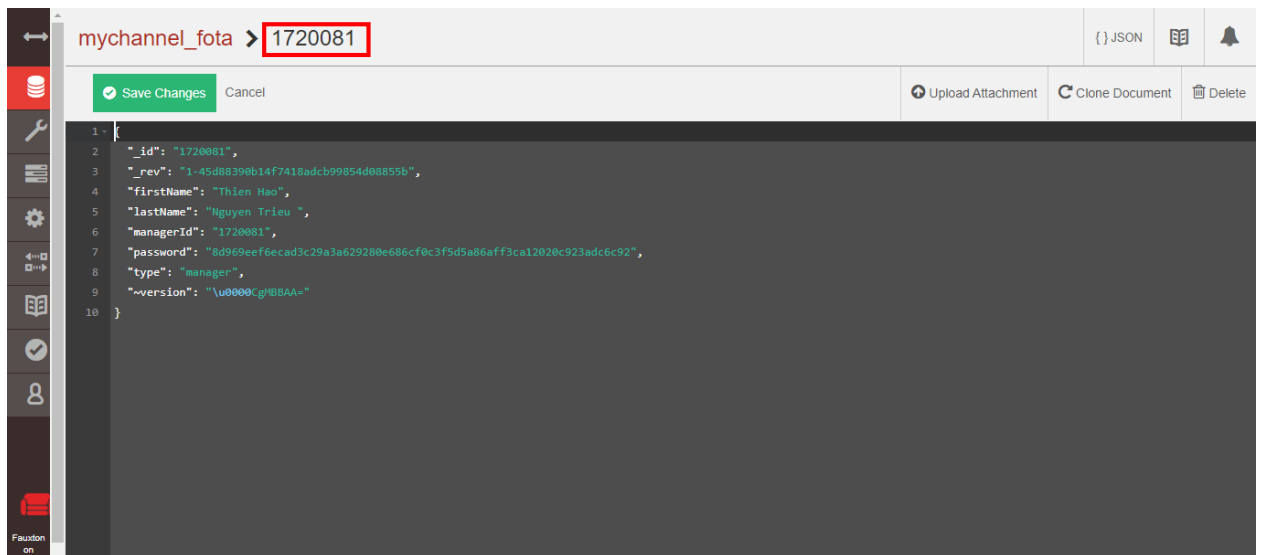
Options {} JSON

Create Document

id	key	value
manager-name	manager	1720081
1720081	1720081	{ "rev": "1-45d88390b14f7418adcb99854d08..." }
dashboard	dashboard	{ "rev": "1-663562b2d0ae1492613c0558018b..." }

Showing document 1 - 3. Documents per page: 20

Hình 4.4 Cơ sở dữ liệu World State trên CouchDB với đối tượng Manager 1720081



Hình 4.5 Thông tin chi tiết đối tượng *Manager* 1720081 trên CouchDB

4.2.2 Đăng nhập tài khoản trên hệ thống

Khi đã có tài khoản, Manager có thể dễ dàng tham gia vào hệ thống thông qua việc điền Manager ID và password đăng ký trước đó.

**Vendor Service
Manager Login**

Manager ID:

Password:

[Login](#)

[Create your Account](#)

Hình 4.6 Đăng nhập tài khoản Manager với ID 1720081

4.3 Kết quả khi thực hiện yêu cầu cập nhật firmware trên hệ thống

4.3.1 Đăng kí các thành phần trên hệ thống

Sau khi đăng nhập thành công, Manager có thể thực hiện các yêu cầu cho hệ thống. Nhưng trước hết, cần khởi tạo các thành phần cần thiết như tải file binary firmware cũng như tạo hệ thống thiết bị IoT trên hệ thống chính.

4.3.1.1 Tải file binary firmware

Ở giao diện chính của hệ thống phần “Push Firmware”, Manager cần điền những thông tin sau để đăng tải file binary firmware cũng như các thông tin về file đó:

- Firmware Version: phiên bản firmware,
- Device Type: Loại thiết bị.

Và chọn đường dẫn phù hợp trỏ đến file mong muốn. Ở đây, cần đăng tải file firmware đầu tiên là phiên bản hiện đang hoạt động trên thiết bị (blockchain-fota0), và firmware cần cập nhật (blockchain-fota1).

Push Firmware

Firmware Version:

Device Type:

File:
 fota0.bin

Hình 4.7 Đăng tải file binary firmware blockchain-fota0

Đồng thời một đối tượng thuộc cấu trúc dữ liệu *FirmwareOTA* sẽ được khởi tạo và ghi nhận vào cơ sở dữ liệu World State trên CouchDB.

mychannel_fota > blockchain-fota0

```

1 {
2   "_id": "blockchain-fota0",
3   "_rev": "1-06ae2420b227efcbd07cc3cb46865d81",
4   "deviceType": "ESP32",
5   "firmwareHash": "e3b0c44298fc149aafb4c8996fb92427ae41e4649b934ca495991b7852b855",
6   "firmwareVersion": "blockchain-fota0",
7   "managerId": "1729001",
8   "timestamp": 1628272037667,
9   "type": "fota",
10  "url": "http://localhost:5000/files/blockchain-fota0",
11  "~version": "\u0000g\u0000A="
12 }

```

Hình 4.8 Thông tin chi tiết đối tượng *FirmwareOTA* blockchain-fota0 trên CouchDB

Tương tự vậy cho blockchain-fota1.

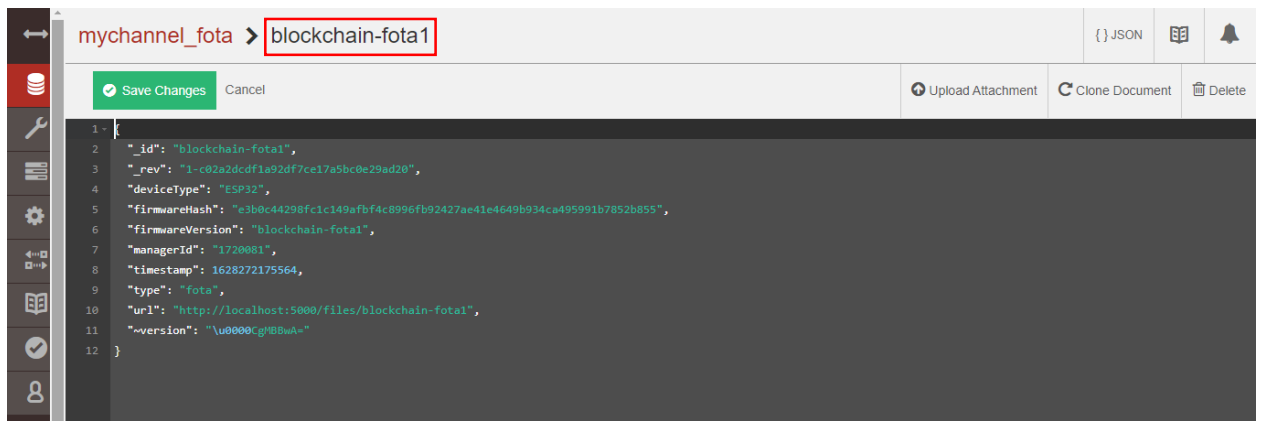
Push Firmware

Firmware Version:

Device Type:

File:
 fota1.bin

Hình 4.9 Đăng tải file binary firmware blockchain-fota1



Hình 4.10 Thông tin chi tiết đối tượng *FirmwareOTA* blockchain-fota1 trên CouchDB

Sau khi tiến hành đăng tải 2 file, ở giao diện chính mục “Firmware List” sẽ hiển thị 2 file này gồm phiên bản firmware và định dạng file (application/octet-stream).

Firmware List	
blockchain-fota0	(application/octet-stream) - file.size bytes
blockchain-fota1	(application/octet-stream) - file.size bytes

Hình 4.11 Hiện thị danh sách firmware

4.3.1.2 Tạo hệ thống thiết bị IoT

Ở giao diện chính của hệ thống phần “Push IoT System”, Manager cần điền những thông tin sau để tạo hệ thống thiết bị IoT:

- Latest Version: phiên bản firmware hiện tại đang hoạt động trên thiết bị,
- MAC Address: địa chỉ MAC tương ứng của thiết bị,
- Device Type: Loại thiết bị.

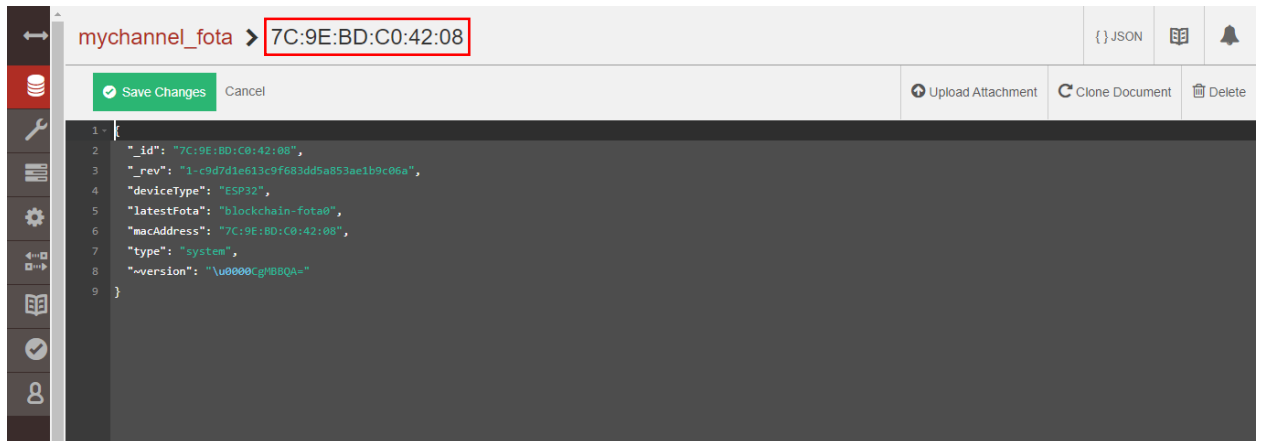
Push IoT System

Latest Version	blockchain-fota0
MAC Address	7C:9E:BD:Co:42:08
Device Type	ESP32

Submit

Hình 4.12 Tạo hệ thống thiết bị IoT

Đồng thời một đối tượng thuộc cấu trúc dữ liệu *IoTSystem* sẽ được khởi tạo và ghi nhận vào cơ sở dữ liệu World State trên CouchDB.



Hình 4.13 Thông tin chi tiết đối tượng *IoTSystem* trên CouchDB

4.3.2 Thực hiện yêu cầu cập nhật firmware trên hệ thống

Sau khi đã khởi tạo thành công các thành phần cần thiết, Manager có thể tiến hành tạo yêu cầu cập nhật firmware từ xa cho thiết bị IoT. Ở giao diện chính của hệ thống phần “Push Update”, Manager cần điền những thông tin sau để tạo yêu cầu:

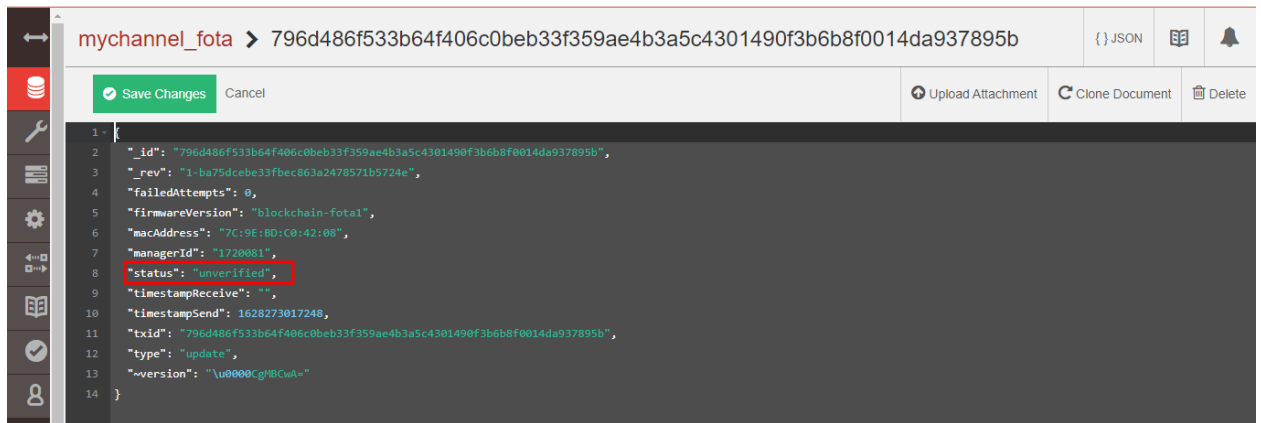
- Firmware Version: phiên bản firmware mong muốn cập nhật,
- MAC Address: địa chỉ MAC tương ứng của thiết bị đích.

Push Update

Firmware Version	<input type="text" value="blockchain-fota1"/>
MAC Address	<input type="text" value="7C:9E:BD:C0:42:08"/>
<input type="button" value="Submit"/>	

Hình 4.14 Tạo yêu cầu cập nhật firmware từ xa cho thiết bị IoT

Đồng thời một đối tượng thuộc cấu trúc dữ liệu *UpdateFOTA* sẽ được khởi tạo và ghi nhận vào cơ sở dữ liệu World State trên CouchDB. Yêu cầu này được định danh bởi giá trị Transaction ID ghi nhận trong thuộc tính *txid*. Lúc này do vừa khởi tạo nên thuộc tính *status* ở trạng thái “*unverified*”.



Hình 4.15 Thông tin chi tiết đối tượng *UpdateFOTA* trên CouchDB

Ở phía thiết bị IoT trên màn hình console, thiết bị sẽ polling kiểm tra có yêu cầu hay không. Ban đầu khi chưa có yêu cầu, nội dung phản hồi khi thiết bị thực hiện truy vấn API `/checkRequire/:macAddress` sẽ là “FALSE”.

```
*****
*                                CHECK REQUIRE                                *
*****
[HTTP] begin...
http://192.168.1.98:5000/checkRequire/7C:9E:BD:C0:42:08 [HTTP] GET...
[HTTP] GET... code: 200
{"response":"FALSE","require":"null"}
size: 37
response: FALSE
```

Hình 4.16 Console khi chưa có yêu cầu cập nhật firmware

khi có yêu cầu cập nhật, nội dung phản hồi khi thiết bị thực hiện truy vấn API `/checkRequire/:macAddress` sẽ là “TRUE”. Thiết bị tiến hành đọc trường “*require*” trong nội dung phản hồi để lưu các thông tin về yêu cầu cập nhật.

```
*****
*                                CHECK REQUIRE                                *
*****
[HTTP] begin...
http://192.168.1.98:5000/checkRequire/7C:9E:BD:C0:42:08 [HTTP] GET...
[HTTP] GET... code: 200
{"response":"TRUE","require":{"txid":"796d486f533b64f406c0beb33f359ae4b3a5c4301490f3b6b8f0014da937895b","firmwareVersion":"blockchain-fotal"}}
size: 142
response: TRUE
txid: 796d486f533b64f406c0beb33f359ae4b3a5c4301490f3b6b8f0014da937895b
firmwareVersion: blockchain-fotal
```

Hình 4.17 Console khi có yêu cầu cập nhật

Sau đó, dựa vào phiên bản firmware vừa đọc, thiết bị tiến hành tải file binary firmware về để thực hiện việc xác thực. Thiết bị sẽ sử dụng thuật toán băm SHA256 để tìm giá trị băm của khối dữ liệu vừa tải về. Sau đó thực hiện truy vấn API `/verifyRequire/:dataVerify`.

```

*****
*                               GET FIRMWARE                               *
*****
[HTTP] begin...
http://192.168.1.98:5000/files/blockchain-fotal[HTTP] GET...
[HTTP] GET... code: 200
size: 768656
*****
*                               HASH BIN FILE                               *
*****
Hash:
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
Done
hash: e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
*****
*                               VERIFY REQUIRE                               *
*****
[HTTP] begin...
http://192.168.1.98:5000/verifyRequire/796d486f533b64f406c0beb33f359ae4b3a5c4301490f3b6b8f0014da937895b,e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855[HTTP] GET...
[HTTP] GET... code: 200
{"response": "VERIFIED"}
size: 23
response: VERIFIED

```

Hình 4.18 Console khi tiến hành quá trình xác thực yêu cầu cập nhật

Như hình 4.18, có thể thấy kết quả việc xác thực là thành công với nội dung phản hồi là “VERIFIED”. Nên thiết bị sẽ tiến hành cài đặt firmware ấy lên thiết bị. Sau khi thực hiện xong việc cài đặt, thiết bị sẽ cập nhật lại thuộc tính *statusDevice* để xác nhận kết quả quá trình cập nhật.

```

*****
*                               UPDATE FIRMWARE                               *
*****
[HTTP] begin...
http://192.168.1.98:5000/files/blockchain-fotal[HTTP] GET...
[HTTP] GET... code: 200
size: 768656
*****
*                               HASH BIN FILE                               *
*****
Hash:
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
Done
statusDevice: DONE

```

Hình 4.19 Console khi tiến hành quá trình cài đặt firmware

Theo hình 4.19, có thể thấy việc cập nhật đã thành công. Thiết bị thực hiện việc truy vấn API `/recordRequire/:dataRecord` để ghi nhận lại kết quả yêu cầu cập nhật firmware trên cơ sở dữ liệu.

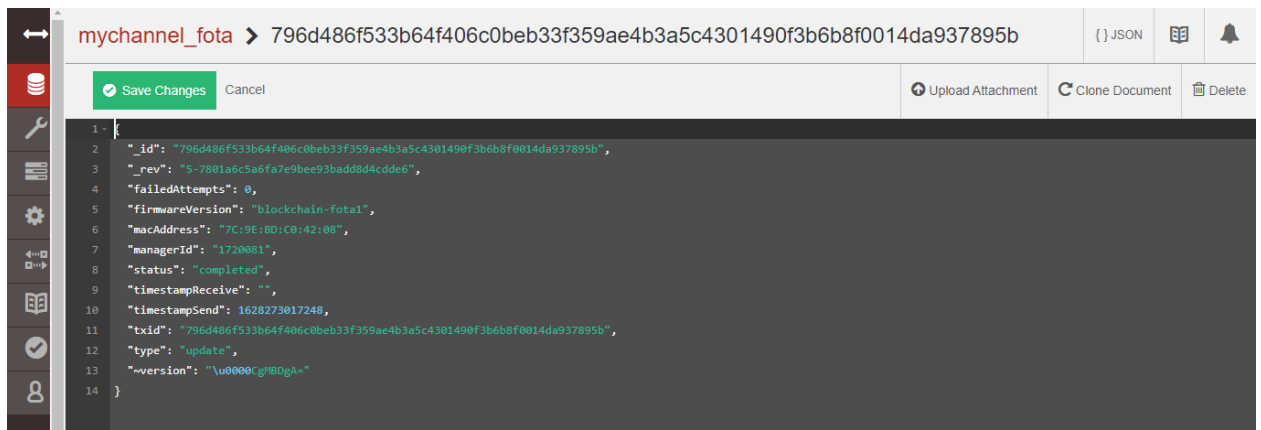
```

*****
*                               RECORD REQUIRE                               *
*****
[HTTP] begin...
http://192.168.1.98:5000/recordRequire/796d486f533b64f406c0beb33f359ae4b3a5c4301490f3b6b8f0014da937895b,DONE[HTTP] GET...
[HTTP] GET... code: 200
{"response": "COMPLETED"}
size: 24
response: COMPLETED
record: COMPLETED

```

Hình 4.20 Console khi tiến hành ghi nhận kết quả yêu cầu cập nhật

Thuộc tính *status* sẽ được cập nhật thành “*completed*” khi ghi nhận kết quả thành công (nội dung phản hồi là “COMPLETED”).



Sau khi đã hoàn tất quá trình xử lý yêu cầu cập nhật, thiết bị cần khởi động lại để hoạt động theo phiên bản firmware mới cập nhật.

```
done!!!!!!!!!!!!!!
```

```
ets Jun  8 2016 00:22:57
```

```
rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
```

```
configsip: 0, SPIWP:0xee
```

```
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
```

```
mode:DIO, clock div:1
```

```
load:0x3fff0018,len:4
```

```
load:0x3fff001c,len:1044
```

```
load:0x40078000,len:10124
```

```
load:0x40080400,len:5856
```

```
entry 0x400806a8
```

Hình 4.21 Console khi khởi động lại thiết bị

CHƯƠNG 5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Trong bối cảnh số lượng các thiết bị cũng như các hệ thống IoT tăng cao theo thời gian, việc quản lý và bảo mật các hệ thống này trở nên khó khăn hơn. Sau quá trình nghiên cứu và thực hiện việc ứng dụng công nghệ Private Blockchain Network dựa trên nền tảng Hyperledger Fabric vào quá trình cập nhật firmware từ xa cho thiết bị IoT, từ đó đưa ra kết luận rằng việc ứng dụng trên mang lại các hiệu quả như:

- Tính quản trị được tăng cao: sử dụng công nghệ sổ cái phân tán (Distributed Ledger Technology) trong HF giúp hỗ trợ việc quản lý dễ dàng hơn thông qua việc lưu trữ các thông tin hệ thống như một đối tượng trên cơ sở dữ liệu trong sổ cái.
- Tính bảo mật được tăng cao: các hệ thống IoT hoặc các thiết bị IoT đều thuộc sở hữu của các đơn vị, tổ chức khác nhau nên việc bảo mật giữa các đơn vị, tổ chức ấy rất cần thiết. HF cung cấp Certificate Authority giúp phát hành các chứng chỉ định danh hỗ trợ tính bảo mật trong quá trình sử dụng giữa các đơn vị, tổ chức.
- Tính toàn vẹn dữ liệu được tăng cao: môi trường giao tiếp chính trong hệ thống IoT là Internet, nên việc tấn công mạng là điều khó tránh khỏi. Việc đảm bảo tính toàn vẹn dữ liệu là yêu cầu hàng đầu trong việc vận chuyển gói firmware đến thiết bị IoT đích. Để đảm bảo việc ấy, thuật toán băm là một lựa chọn tốt nhưng kết quả đối chiếu cần được lưu trữ một cách bảo mật để tránh việc điều chỉnh bởi tấn công mạng. Thông qua công nghệ sổ cái phân tán được cung cấp bởi HF, các firmware được lưu trữ với độ toàn vẹn cao nhờ tính “chỉ bổ sung” của sổ cái. Mọi thông tin bổ sung đều được lưu trữ trong các khối trong chuỗi khối (blockchain).

5.2 Hướng phát triển

Do hạn chế về mặt kiến thức cũng như thời gian thực hiện còn nhiều hạn hẹp, nên việc thực hiện triển khai hệ thống có khá nhiều sai sót và chưa phù hợp với yêu cầu ban đầu đề ra. Đặc biệt ở phần xử lý yêu cầu cập nhật firmware phía thiết bị IoT, quá trình thực hiện còn phụ thuộc nhiều vào Vendor Service khiến độ bảo mật chưa cao. Do những giới hạn trong thời gian nghiên cứu, nên vấn đề truy vấn của thiết bị không trực tiếp đến cơ sở dữ liệu mà phải thông qua Vendor Service; điều này dẫn đến có thể bị tấn công mạng bất cứ

lúc nào. Cần phát triển hệ thống theo hướng liên kết thiết bị IoT trực tiếp vào mạng Blockchain. Đồng nghĩa việc các tổ chức phát hành chứng chỉ CA sẽ phát hành thông tin định danh cho cả các thiết bị IoT (hoặc các hệ thống IoT trong hệ thống mẹ nói chung) để việc truy vấn thông tin được diễn ra trực tiếp giữa thiết bị với cơ sở dữ liệu (ở đây chính là các đối tượng được khởi tạo từ cấu trúc dữ liệu trong Smart Contract trên World State của sổ cái Ledger). Qua đó cần việc xác thực chứng chỉ định danh của thiết bị đối với mạng Blockchain.

Ngoài ra, một số mục tiêu được đề ra để hoàn thiện hệ thống như:

- Tối ưu thuật toán kiểm tra xác thực yêu cầu trong Smart Contract để nâng cao hiệu suất cho hệ thống. Thông qua việc xử lý các tham số được ghi nhận như số lần xử lý yêu cầu xác thực bị thất bại, thời gian gửi và nhận trong toàn bộ quá trình xác thực (có thể khảo sát thời gian trung bình khi xử lý yêu cầu và đưa ra các ngưỡng hợp lý khi chờ xử lý),
- Tối ưu hóa các trường hợp có thể xảy ra đối với hệ thống trong môi trường thực tế, khả năng duy trì, mức chống chịu trong các điều kiện mạng khác nhau (khi thiết bị hay hệ thống IoT offline/online),
- Tối ưu hóa thời gian xử lý các giao dịch trong hệ thống,
- Tối ưu hóa giao diện người dùng.

TÀI LIỆU THAM KHẢO

- [1] IBM Corporation, The Linux Foundation, “Hyperledger Fabric Whitepaper,” 2020. [Trực tuyến]. Available: https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf. [Đã truy cập 07 /2021].
- [2] IBM Corporation, The Linux Foundation, “Blockchain network — hyperledger-fabricdocs master documentation,” 2019. [Trực tuyến]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/network/network.html#the-sample-network>. [Đã truy cập 07 /2021].
- [3] H. Xinchí, A. Sarra, G. Rose và P. Mauricio, Securing Over-The-Air IoT Firmware Updates using Blockchain, Tulsa, Oklahoma, USA: Sarra Alqahtani, 2019.
- [4] IBM Corporation, The Linux Foundation, “Install Samples, Binaries and Docker Images — hyperledger-fabricdocs master documentation,” 2019. [Trực tuyến]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/install.html#install-samples-binaries-and-docker-images>. [Đã truy cập 07 /2021].
- [5] IBM Corporation, The Linux Foundation, “Ledger — hyperledger-fabricdocs master documentation,” [Trực tuyến]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger/ledger.html#the-ledger>. [Đã truy cập 07 /2021].
- [6] IBM Corporation, The Linux Foundation, “Transaction context — hyperledger-fabricdocs master documentation,” [Trực tuyến]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/transactioncontext.html>. [Đã truy cập 07 /2021].
- [7] IBM Corporation, The Linux Foundation, “Hyperledger Fabric SDK for node.js Class: ChaincodeStub,” [Trực tuyến]. Available: <https://hyperledger.github.io/fabric-chaincode-node/release-2.2/api/fabric-shim.ChaincodeStub.html>. [Đã truy cập 07 /2021].
- [8] IBM Corporation, The Linux Foundation, “Building Your First Network — hyperledger-fabricdocs master documentation,” [Trực tuyến]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/build_network.html. [Đã truy cập 07 /2021].
- [9] IBM Corporation, The Linux Foundation, “Wallet trong Hyperledger Fabric,” Wallet — hyperledger-fabricdocs master documentation, [Trực tuyến]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/developapps/wallet.html>. [Đã truy cập 07 /2021].

- [10] IBM Corporation, The Linux Foundation, “Hyperledger Fabric SDK for node.js - Module: fabric-network,” [Trực tuyến]. Available: <https://hyperledger.github.io/fabric-sdk-node/release-1.4/module-fabric-network.html>. [Đã truy cập 07 /2021].
- [11] IBM Corporation, The Linux Foundation, “Hyperledger Fabric SDK for node.js Class: Contract,” [Trực tuyến]. Available: <https://hyperledger.github.io/fabric-sdk-node/release-1.4/module-fabric-network.Contract.html>. [Đã truy cập 07 /2021].
- [12] Seeed Studio, Seeed Technology Company,Ltd, “Seeed_Arduino_mbedtls,” [Trực tuyến]. Available: https://github.com/Seeed-Studio/Seeed_Arduino_mbedtls. [Đã truy cập 07 /2021].

