



IT4735
IoT VÀ ỨNG DỤNG
(IoT and Applications)

Phạm Ngọc Hưng, Nguyễn Đình Thuận, Đặng Tuấn Linh
Faculty of Computer Engineering
School of Information and Communication Technology (SoICT)
Hanoi University of Science and Technology
E-mail: [hungpn, thuannnd, linhdt]@soict.hust.edu.vn

Giới thiệu học phần

- IT4735 IoT và Ứng dụng (IoT and Applications)
- Credits: 2 (2-1-0-4)
- Đánh giá:
 - Quá trình: 50%, Bài tập tuần, Bài tập Project
 - Cuối kỳ: 50%,
 - Bài tập Project, báo cáo, thuyết trình, vấn đáp (50%)
 - Thi trắc nghiệm (50%)

Mục tiêu

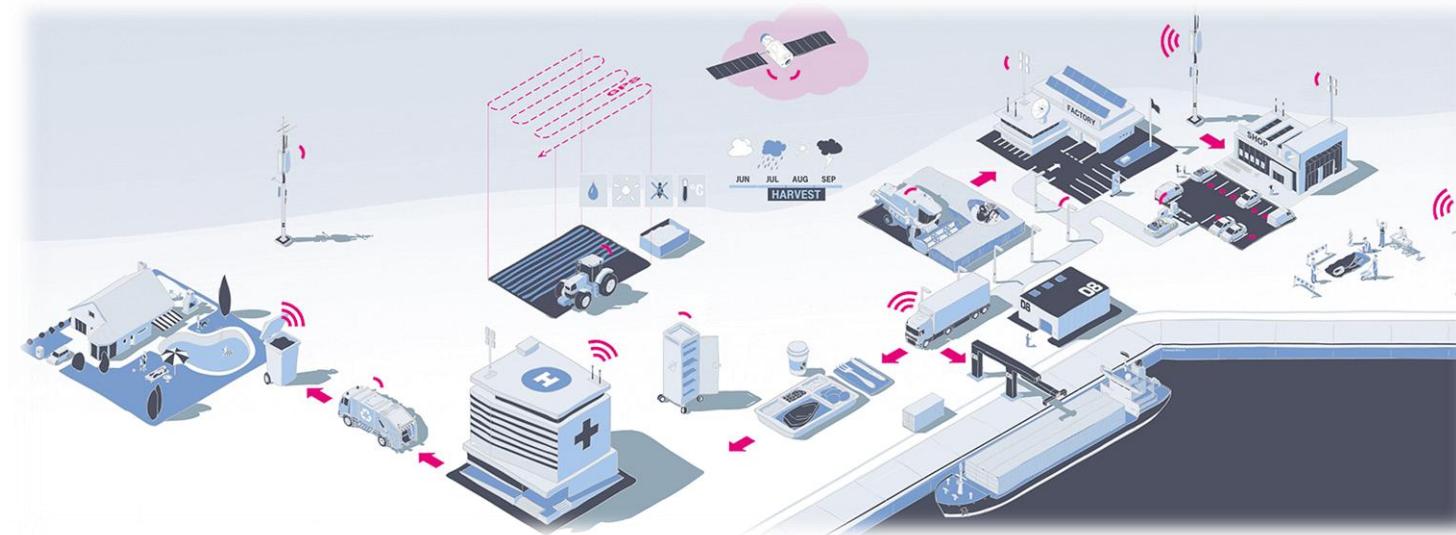
- Kiến thức tổng quan về IoT
- Kiến trúc, mô hình phân lớp, các thành phần của hệ thống IoT
- Các công nghệ IoT: thiết bị và cảm biến, một số chuẩn truyền thông IoT, các giao thức truyền thông cho ứng dụng IoT, nền tảng đám mây
- Lập trình cho thiết bị, hệ thống IoT
- Bảo mật trong IoT
- Vận dụng kỹ năng thiết kế và xây dựng một hệ thống ứng dụng IoT

Nội dung

- Chương 1. Tổng quan về IoT
- Chương 2. Các công nghệ IoT
- Chương 3. Lập trình ứng dụng IoT
- Chương 4. An toàn và Bảo mật IoT
- Chương 5. Thiết kế và xây dựng hệ thống IoT

Chương 1. Tổng quan về IoT

- 1.1. Tổng quan về IoT
- 1.2. Kiến trúc tổng quan hệ thống IoT
- 1.3. Tổng quan về các công nghệ trong IoT
- 1.4. Các ứng dụng IoT
- 1.5. Các vấn đề thách thức của IoT



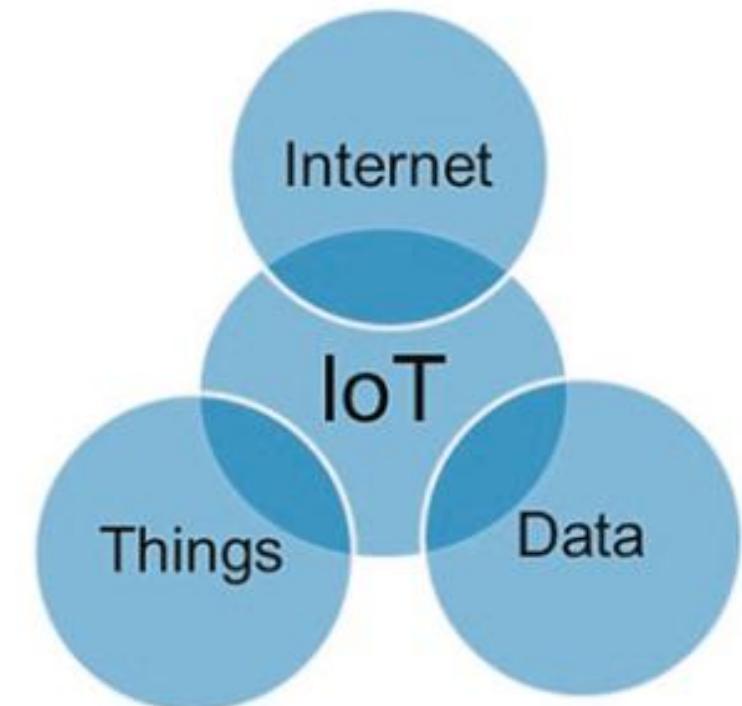
1.1. Tổng quan về IoT

■ What is the Internet of Things (IoT) ?

- *IoT is the network of things, with clear element identification, embedded with software intelligence, sensors, and ubiquitous connectivity to the Internet*
- *(Book: Internet of Things From Hype to Reality)*
- Components:
 - Sensors: to collect information.
 - Identifiers: to identify the source of data (e.g., sensors, devices).
 - Software: to analyze data.
 - Internet connectivity: to communicate and notify.

■ “Things” = “anything”, “everything”

- Home appliances, building, car, people, animals, trees, plants, ...

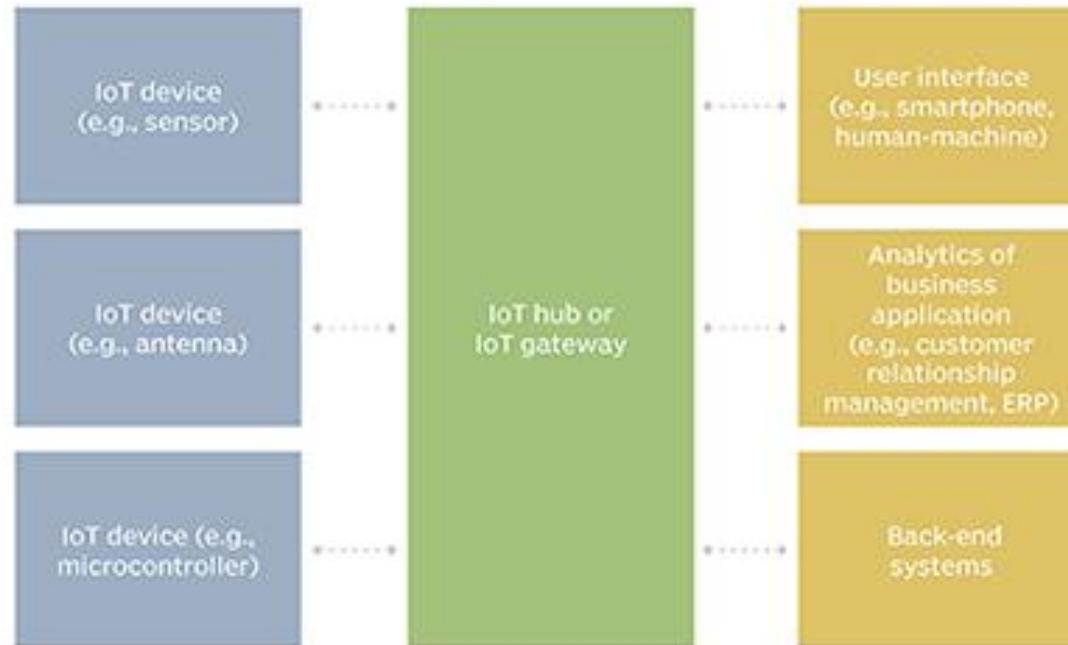


IoE = Internet of Everything (by Cisco)

Tổng quan về IoT

Example of an IoT system

Collect data → Collate and transfer data → Analyze data, take action



Tổng quan về IoT

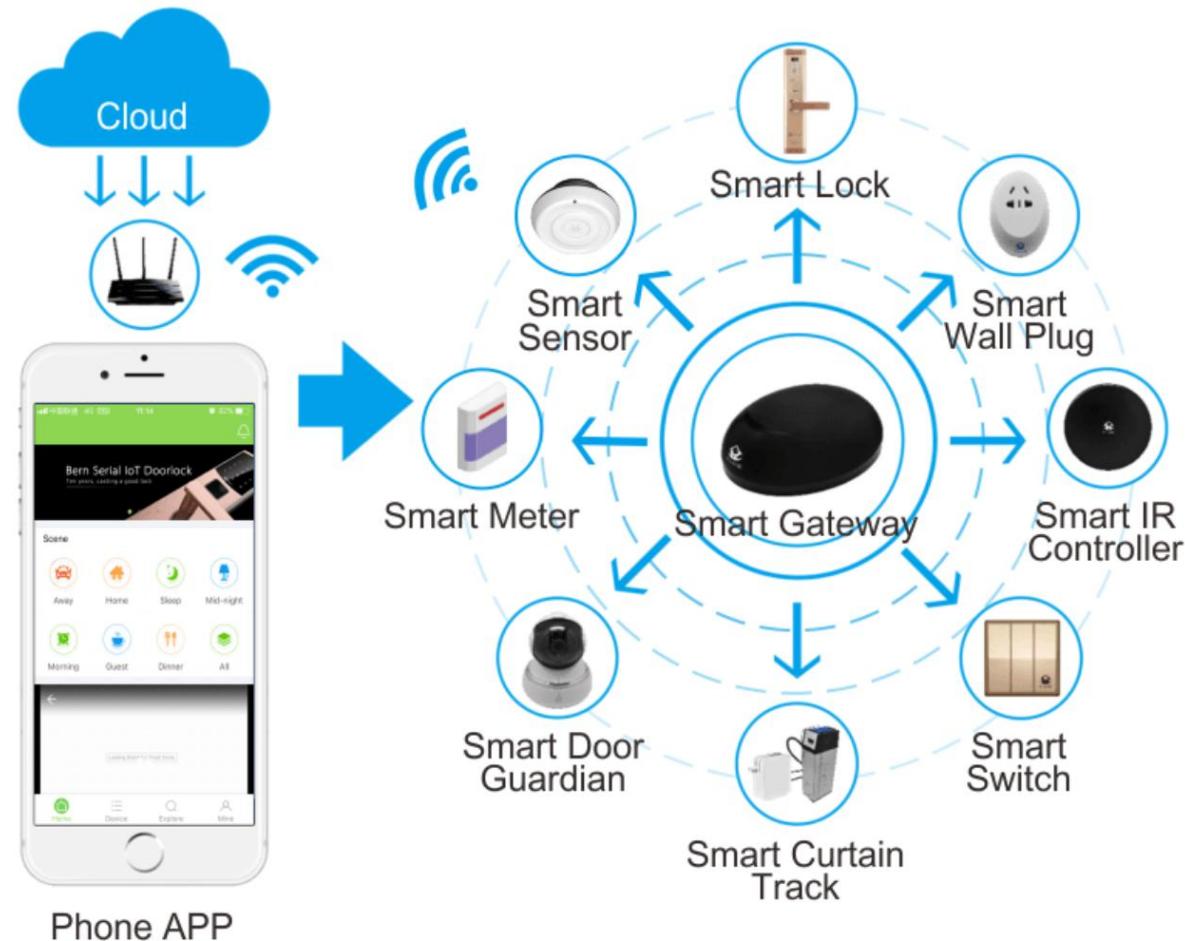
- How does IoT works ?
 - **IoT devices**: smart devices (that use embedded systems such as processors, sensors and communication hardware) to collect, send and act on data they acquire from their environments.
 - **IoT devices** share the sensor data they collect by connecting to an **IoT gateway**, which acts as a central hub where IoT devices can send data.
 - Before the data is shared, it can also be sent to an **edge device** where that data is analyzed locally. Analyzing data locally reduces the volume of data sent to the cloud, which minimizes bandwidth consumption.

Tổng quan về IoT

- How does IoT works ?
 - Sometimes, IoT devices communicate with other related devices and act on the information they get from one another.
 - The devices do most of the work without human intervention, although people can interact with the devices -- for example, to set them up, give them instructions or access the data.
 - The connectivity, networking and communication protocols used with these web-enabled devices largely depend on the specific IoT applications deployed.
 - **IoT can also use artificial intelligence** and machine learning to aid in making data collection processes easier and more dynamic.

Tổng quan về IoT

- An example of IoT smart home solution

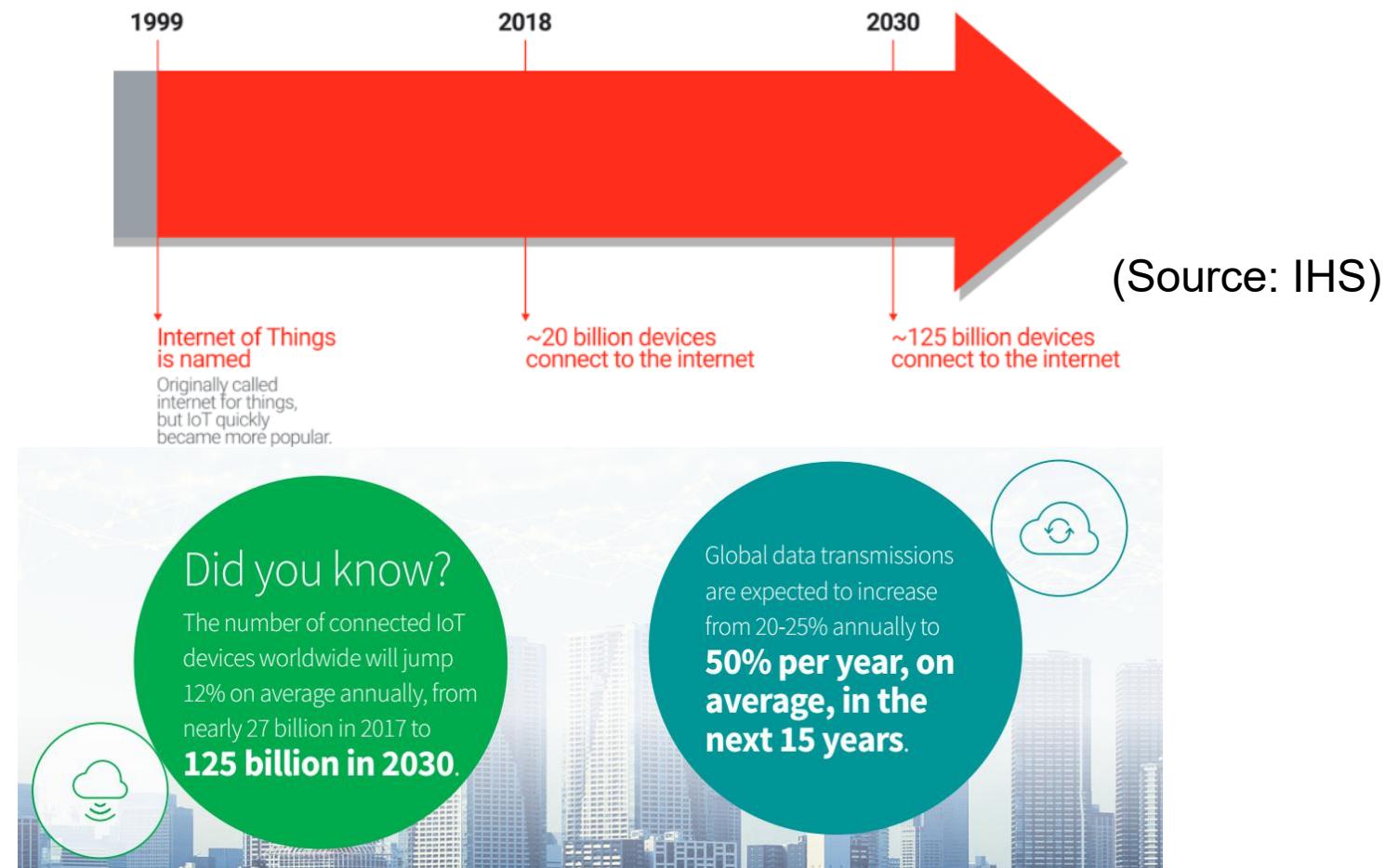


Tổng quan về IoT

- Why is IoT important ?
 - IoT helps people live and work smarter.
 - For example: IoT-embedded devices -- such as cars, smartwatches or thermostats -- improve people lives.
 - IoT is essential to business:
 - Real-time supervising of products/objects/system
 - Delivering insights into everything from the performance of machines to supply chain and logistics operations.
 - IoT enables machines to complete tedious tasks without human intervention.
 - automate processes, reduce labor costs, cut down on waste and improve service delivery.
 - less expensive to manufacture and deliver goods,
 - transparency into customer transactions.

Tiến hóa của IoT

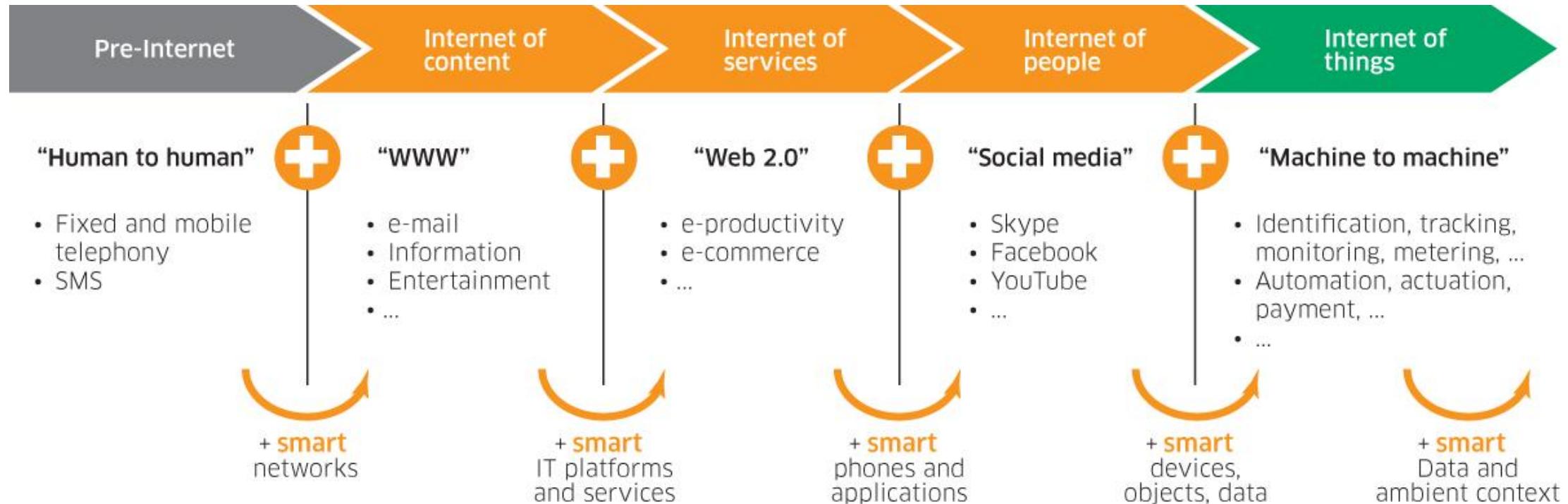
- The field of IoT has grown tremendously
- 1999: Kevin Ashton (Auto-ID Center, MIT) first mentioned the internet of things in a presentation about RFID



Tiến hóa của IoT

- IoT has evolved from the convergence of:
 - Wireless technology
 - Microelectromechanical systems
 - Microservices
 - Internet
- M2M (machine to machine) communication, evolved to next level
- 2010: broader consumer use (smart phones, smart TV)
- 2020: cellular IoT (2G/3G/4G/5G, LoRaWAN, LTE)
- 2023: billions of internet-connected devices
 - **Digital twins:** virtual representation of a real-world entity or process

Tiến hóa của IoT



1. Pre-Internet: https://en.wikipedia.org/wiki/History_of_the_telephone
2. Internet of content (WWW, 1989, Tim Berners-Lee)
https://en.wikipedia.org/wiki/History_of_the_World_Wide_Web
3. Internet of services (Web 2.0, Yahoo, Amazon, ... ~2000, dotcom companies)
4. Internet of people (smart phones, social networks, iPhone1 2007)

Internet of Things named 1999 <https://iot-analytics.com/internet-of-things-definition/>

Tiến hóa của IoT

15

Idea: Move from Internet of People → Internet of Things



- Internet xuất hiện ở khắp nơi trên thế giới
- Ban đầu là để kết nối con người – con người

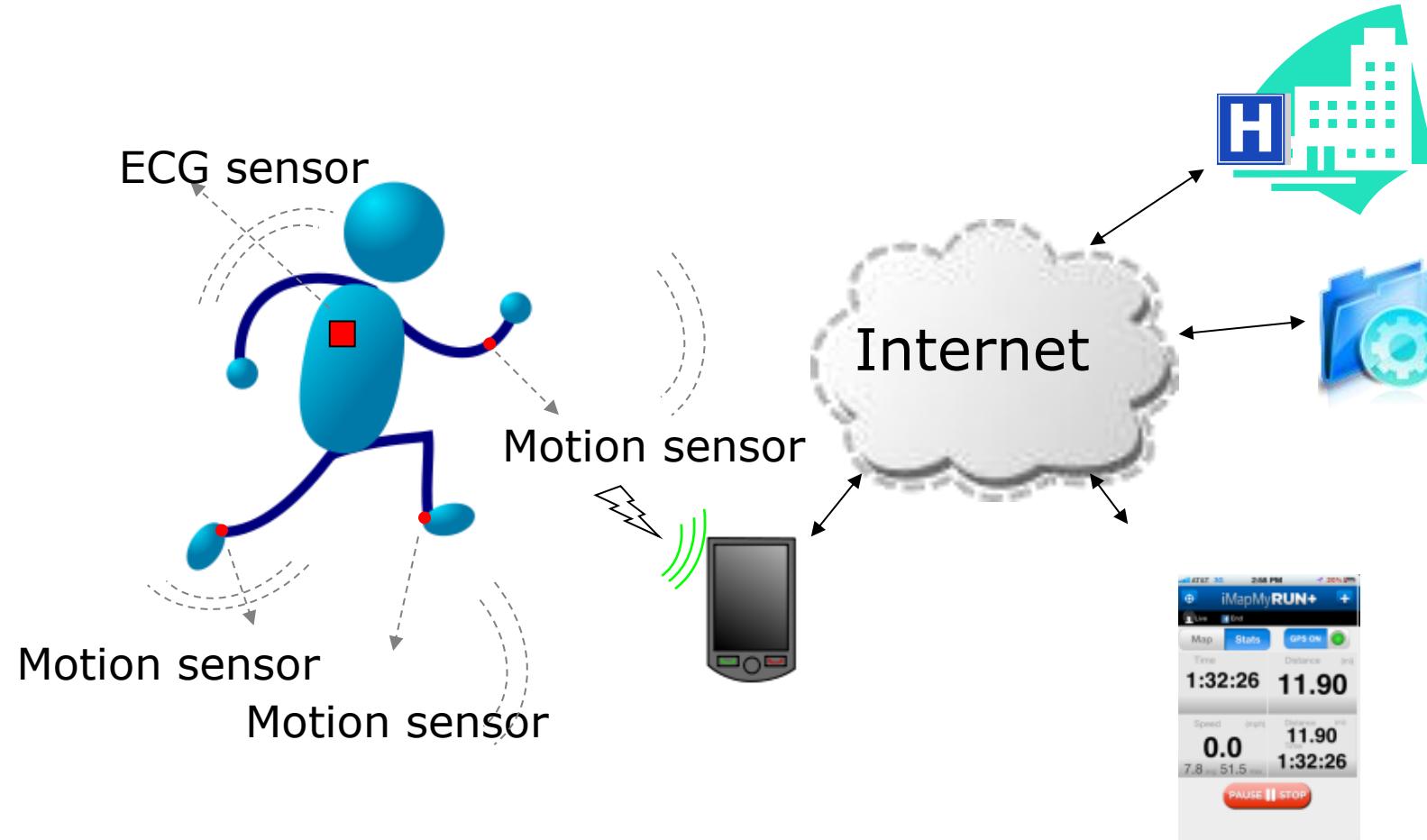
→ Internet of Things



- ❖ Internet of Things kết nối mọi thứ ("things") sử dụng các phương tiện hạ tầng đã có.

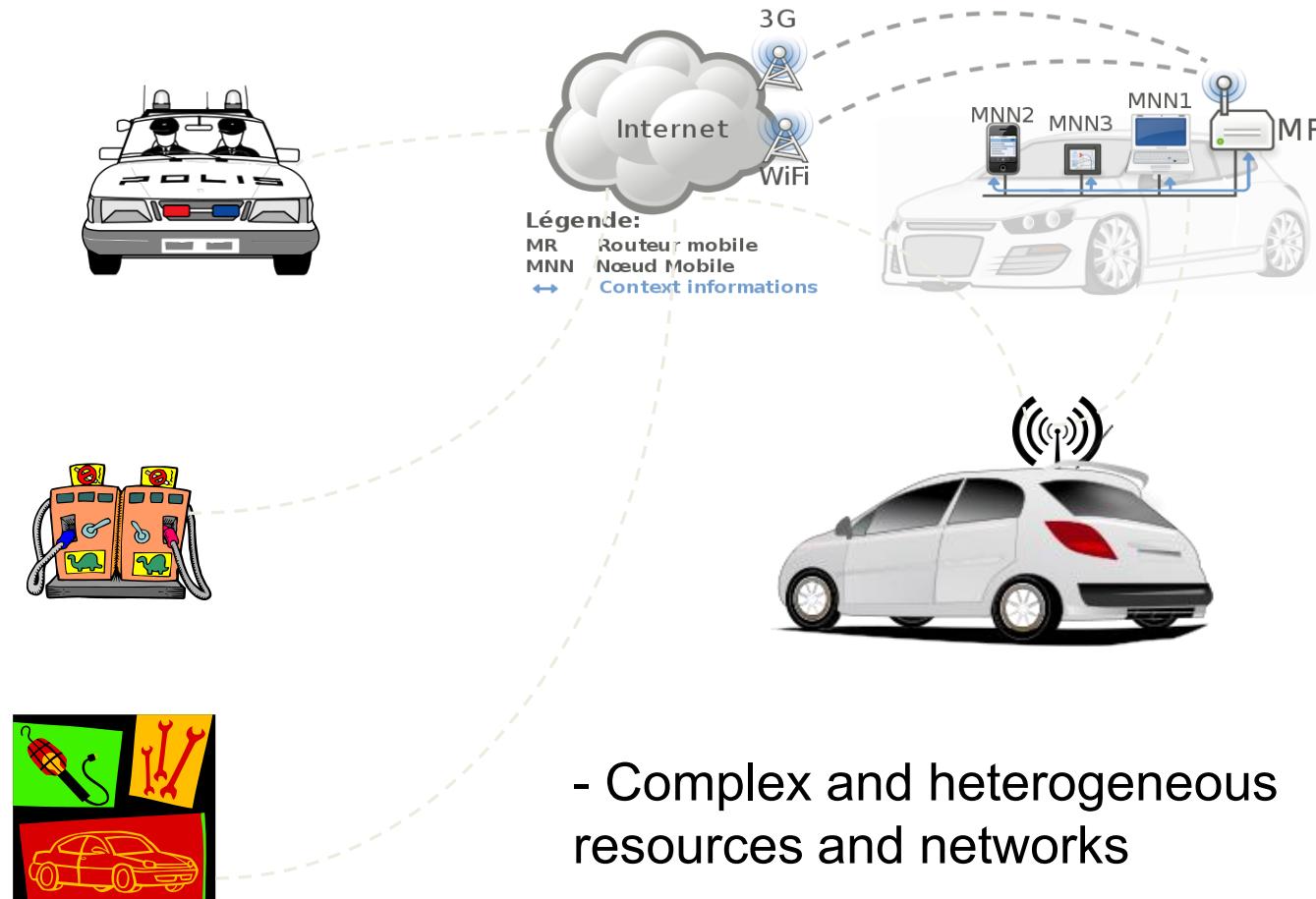
Tiến hóa của IoT

IoT: Human connecting with Things



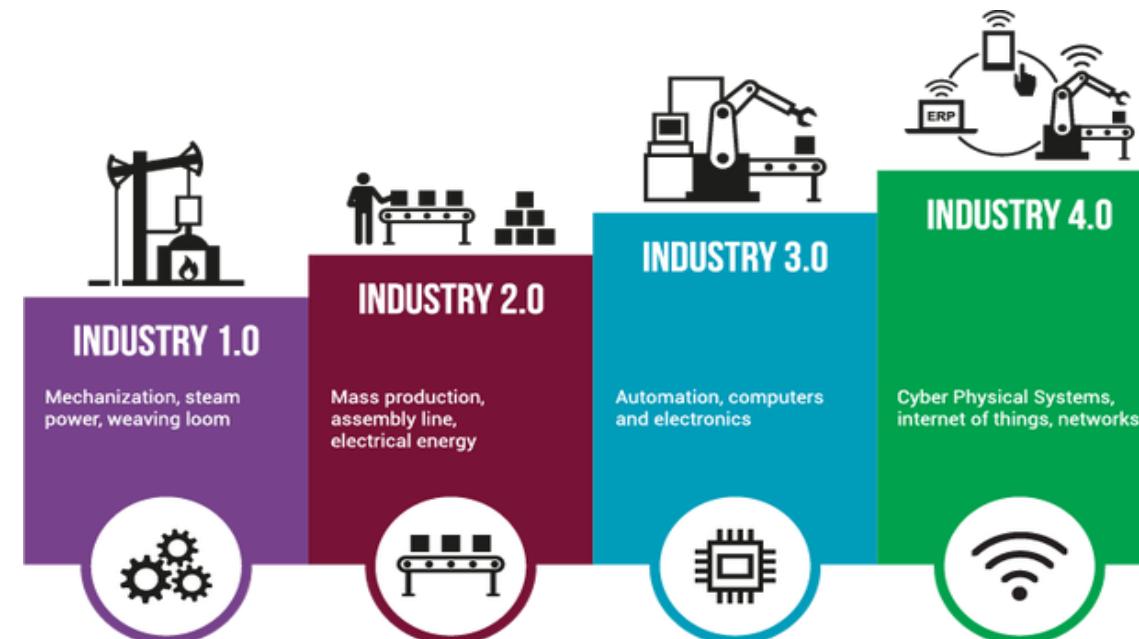
Tiến hóa của IoT

IoT: Things connecting with Things

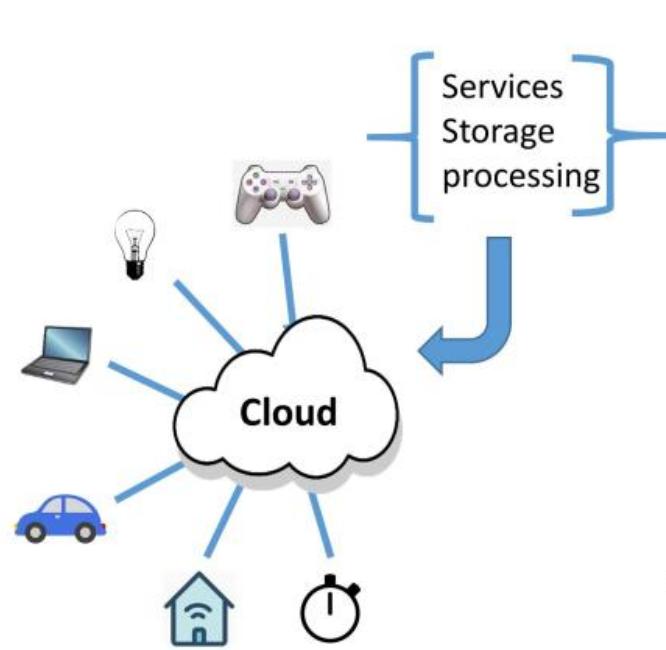


4th IR and IoT

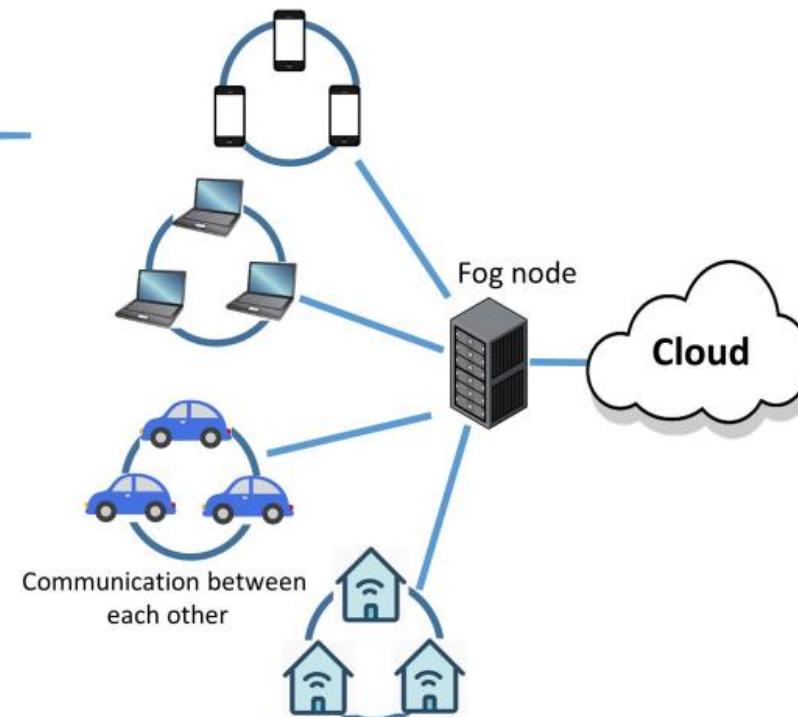
- The Fourth Industrial Revolution and IoT
 - 1st IR: transformed society with the introduction of machines and mechanized production.
 - 2nd IR: introduced electricity, which led to mass production.
 - 3rd IR: has been called the dawn of the information age.
 - 4th IR: as "the fusion of technologies that is blurring the lines between the physical, digital, and biological spheres." (by *Klaus Schwab*)
- IoT is being called a major driver of the Fourth Industrial Revolution. Why?



1.2. Kiến trúc tổng quan hệ thống IoT



(1)



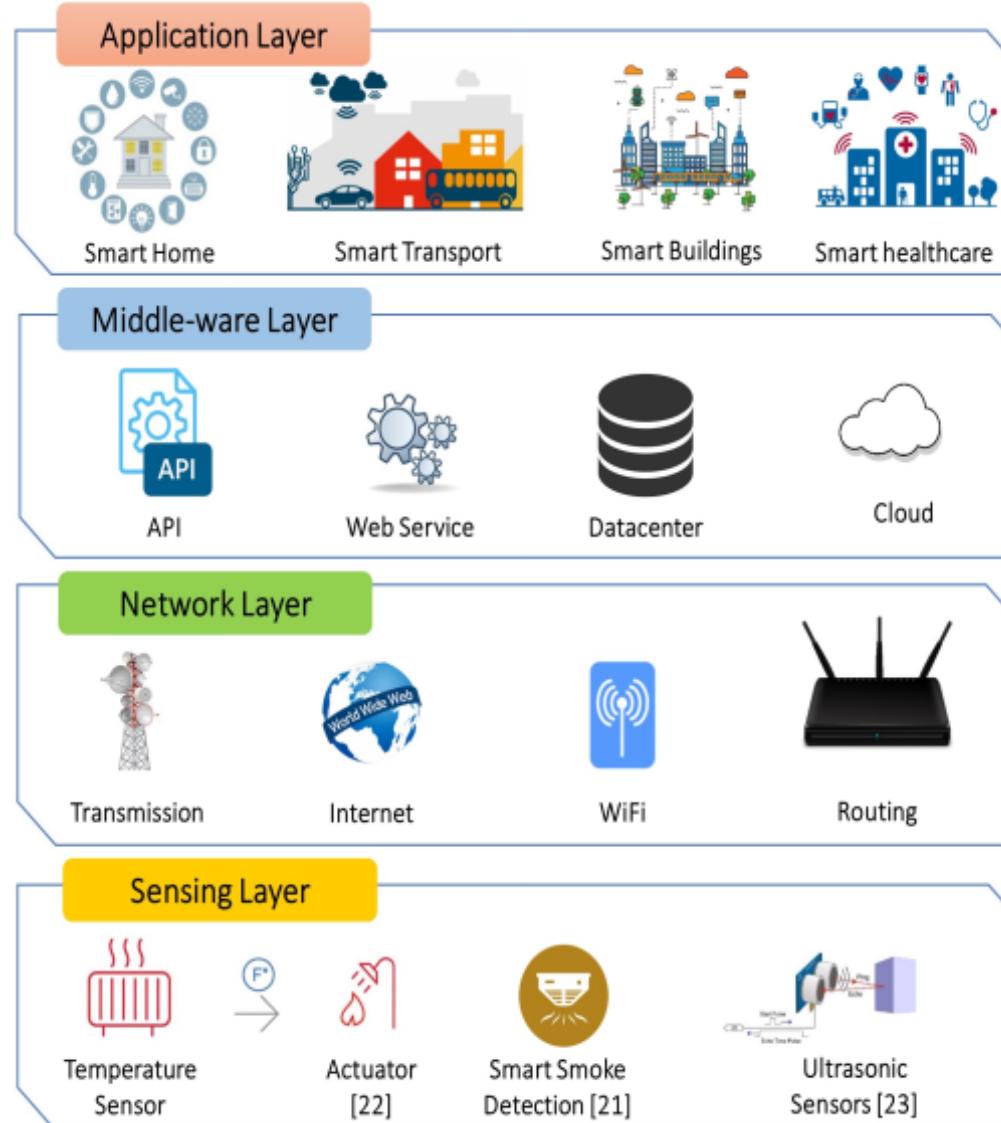
(2)



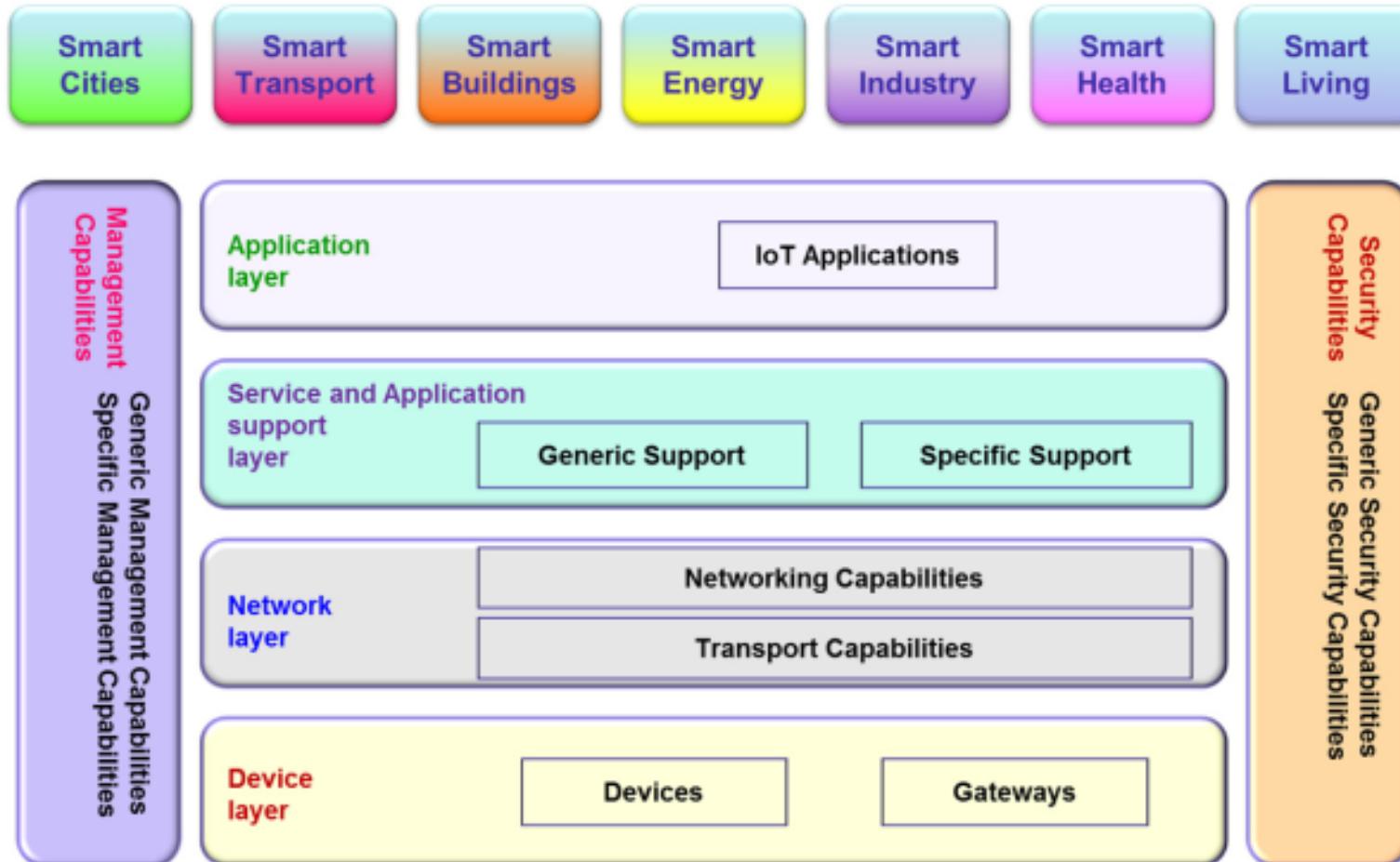
(3)

- (1) Kiến trúc đơn giản: Các thiết bị kết nối trực tiếp đến server/cloud
- (2) Kiến trúc phân cấp: Các thiết bị kết nối qua tầng trung gian (Fog/Edge node, gateway)
- (3) Kiến trúc tương lai: “Things” kết nối trực tiếp “Things”

Kiến trúc phân tầng của hệ thống IoT (1)



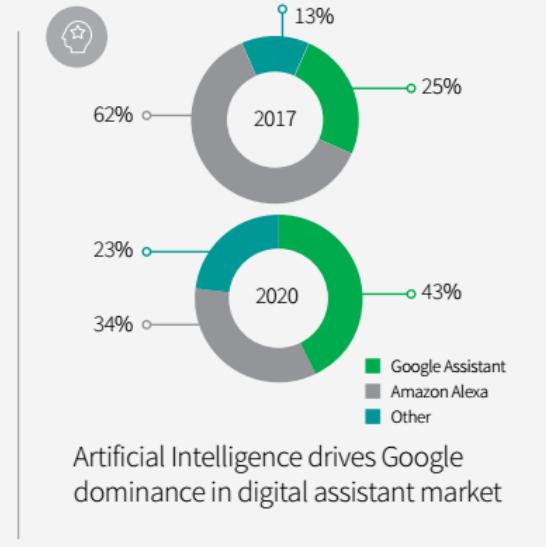
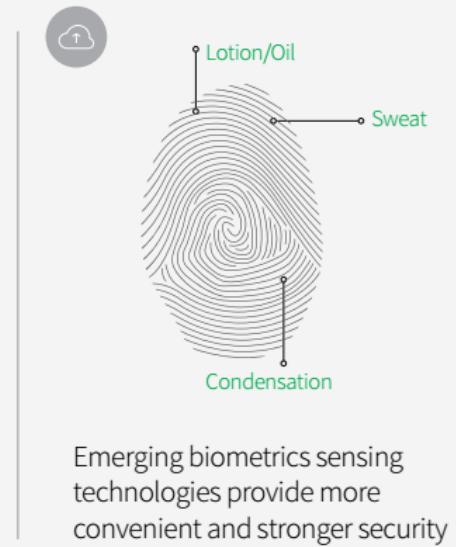
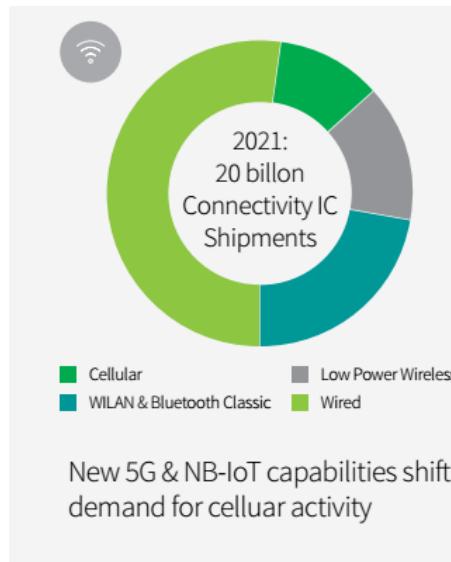
Kiến trúc phân tầng của hệ thống IoT (2)



IoT Layered Architecture (Source: ITU-T)

Bốn “trụ cột” nền tảng của IoT

- **Connections:** Khả năng kết nối (mới) của các thiết bị và thông tin
- **Collection:** Khả năng thu thập dữ liệu (lớn) từ việc gia tăng kết nối các thiết bị và thông tin
- **Computation:** Khả năng tính toán cho phép chuyển đổi từ các dữ liệu đã thu thập vào các tính năng mới
- **Creation:** Khả năng sáng tạo độc đáo của các tương tác, các mô hình kinh doanh, và các giải pháp mới



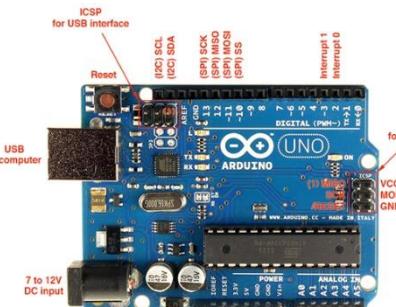
1.3. Tổng quan về các công nghệ trong IoT

- Phần cứng (Hardware)
- Truyền thông (Communication)
- Các giao thức (Protocols)
- Phân tích dữ liệu (Data Analytic)
- Nền tảng đám mây (Cloud Platforms)

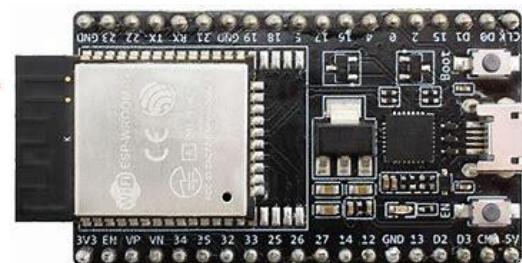
1.3.1. Phần cứng IoT (Hardware)

Các máy tính nhúng (Embedded Computers):

- Vi điều khiển: 8-bit, 32-bit, không dùng hệ điều hành
- Vi điều khiển có dùng hệ điều hành đơn giản (ví dụ: FreeRTOS)
- Bộ xử lý 32-bit, 64-bit, hiệu năng cao, có hệ điều hành (Raspbian, Embedded Linux, Ubuntu, Embedded Windows, ...)
- Các kiến trúc: AVR, Microchip, ARM, Intel, ...



Arduino Uno



ESP32



Raspberry Pi



Intel Galileo

Các cảm biến (Sensors)

- Cảm biến có thể coi là thành phần quan trọng nhất trong thiết bị IoT
 - Đầu ra là tương tự hoặc đầu ra là số
- Các module cảm biến là các thiết bị thường bao gồm:
 - Thành phần cung cấp, quản lý năng lượng (energy/power modules)
 - Thành phần cảm biến (sensing modules)
 - Thành phần quản lý giao tiếp
- Thành phần quản lý giao tiếp thông qua xử lý tín hiệu (RF modules)
 - WiFi, ZigBee, Bluetooth, radio transceiver, ...

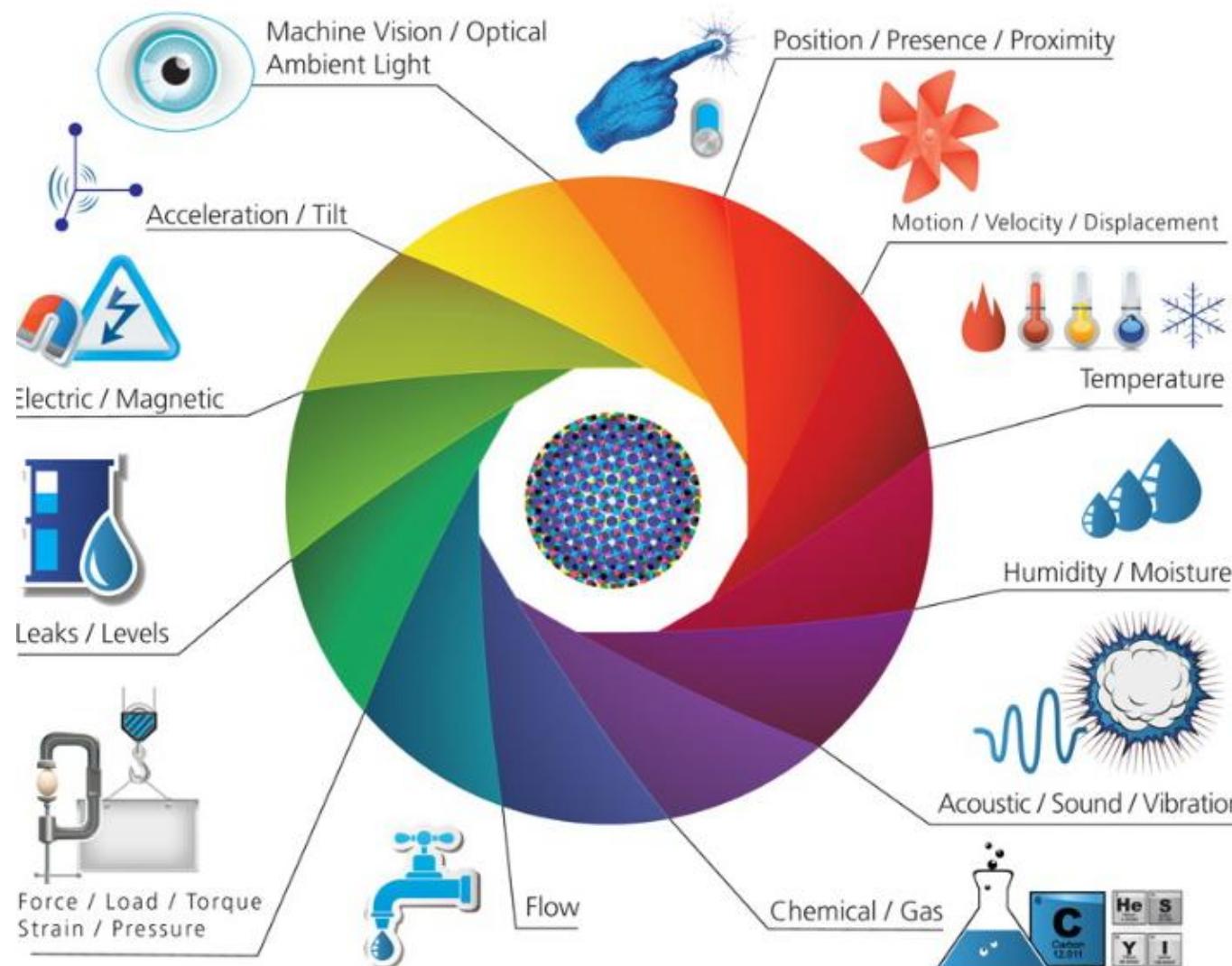


Các cảm biến (Sensors)

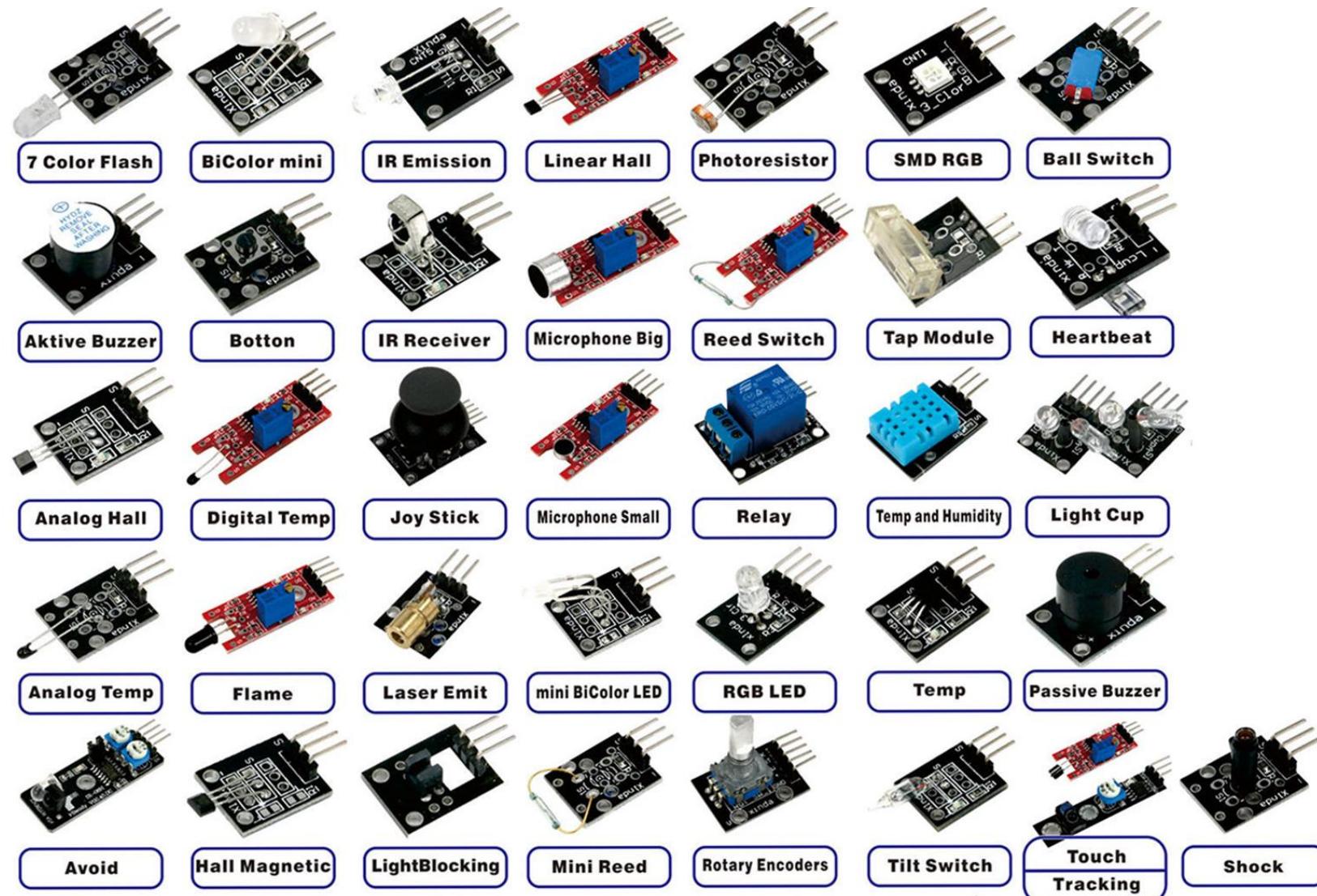
- Có nhiều loại cảm biến:

Devices	
accelerometers	temperature sensors
magnetometers	proximity sensors
gyroscopes	image sensors
acoustic sensors	light sensors
pressure sensors	gas RFID sensors
humidity sensors	micro flow sensors

Các cảm biến (Sensors)

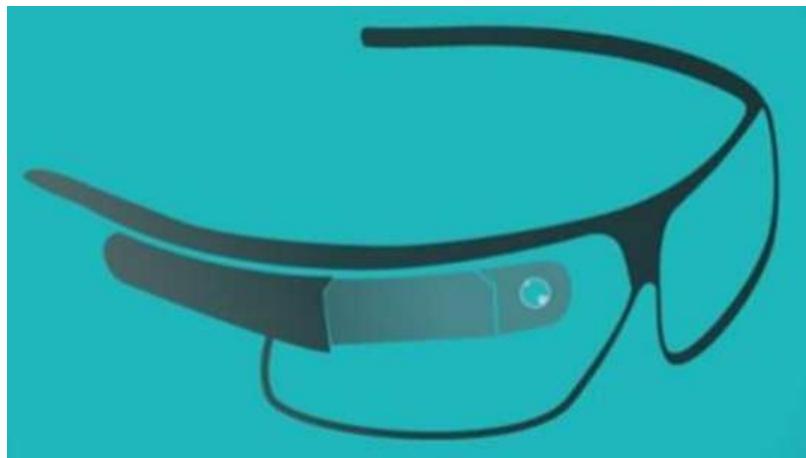


Các cảm biến (Laboratory type)



Wearables IoT

- **Head** – Helmets, glasses
- **Neck** – Jewelry, collars
- **Arm** – Watches, wristbands, rings
- **Torso** – Clothing, backpacks
- **Feet** – Socks, shoes

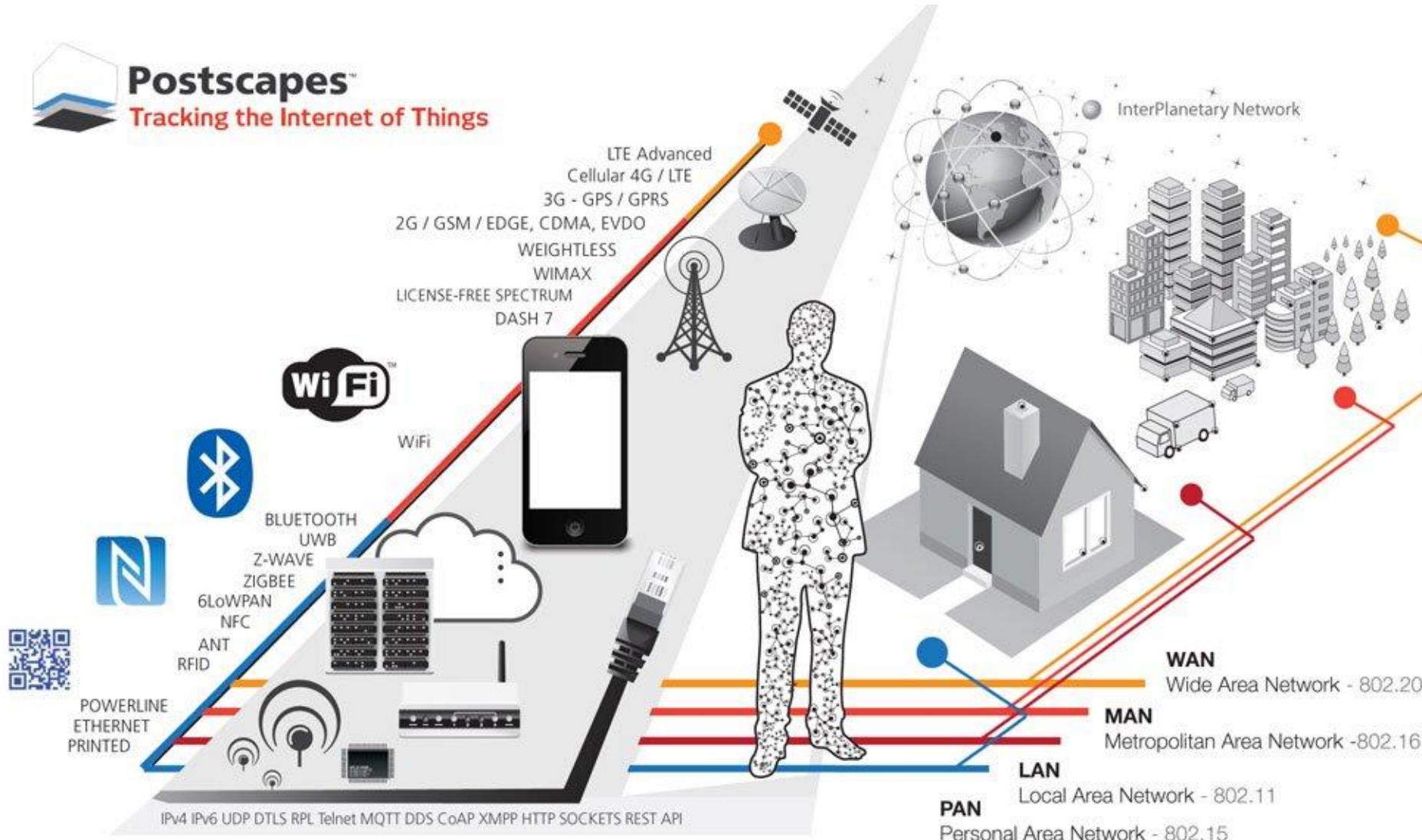


Wearables IoT



- <https://www.sportswearable.net/fitness-and-sports-wearables-world-wide-market-analysis-forecasts-and-trends-through-2019-2025/>

1.3.2. Truyền thông trong IoT (Communications)



Truyền thông trong IoT

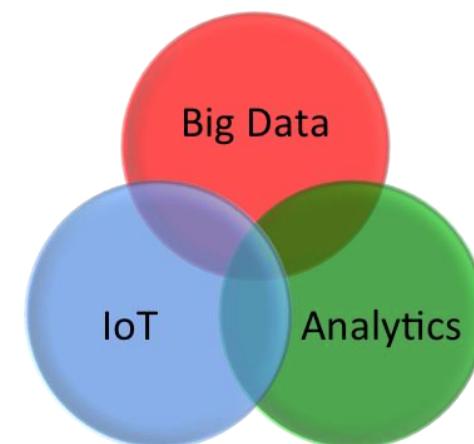
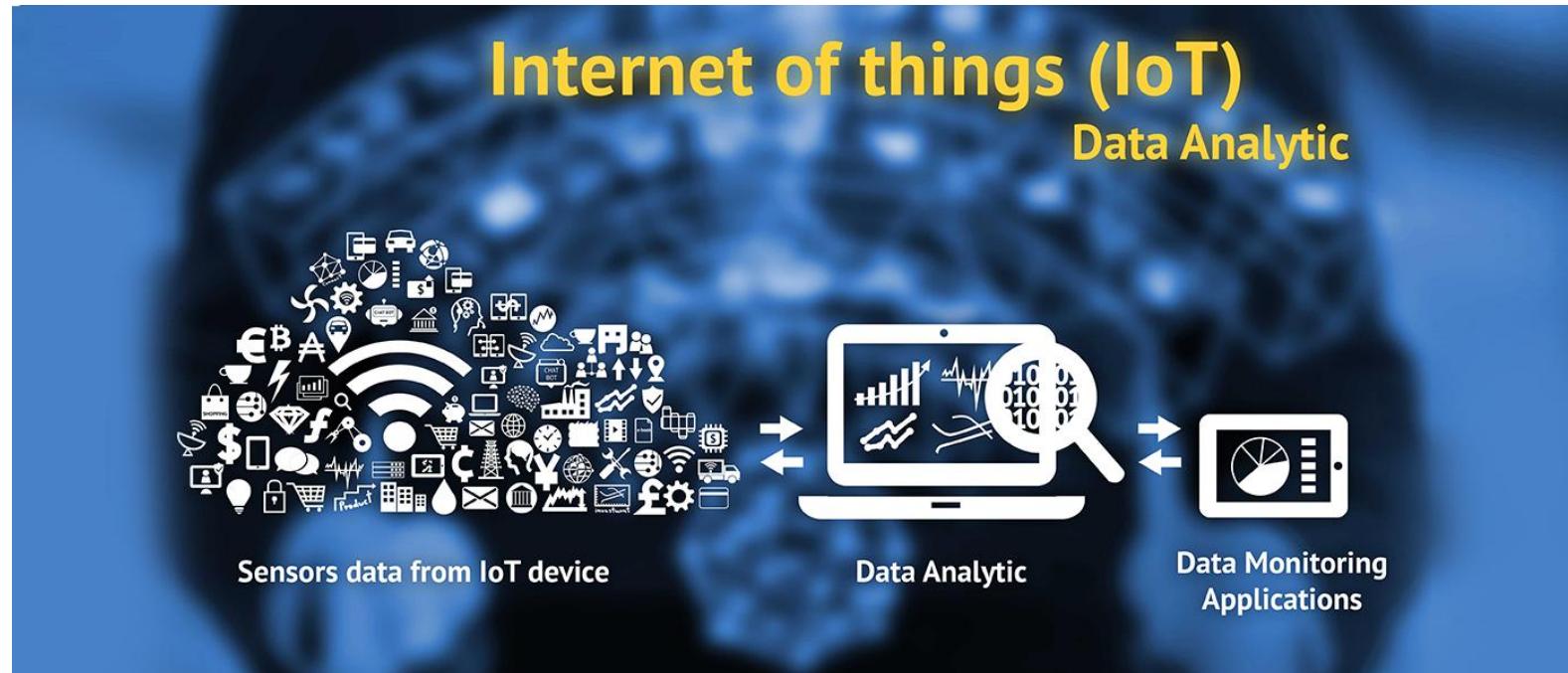
- Một số chuẩn truyền thông (communication standards) phổ biến cho IoT:
 - NFC and RFID
 - Bluetooth
 - WiFi
 - GSM, 3G/4G/LTE, LTE-A

Truyền thông trong IoT

- Một số giao thức (protocols) phổ biến cho truyền nhận dữ liệu trong IoT:
 - HTTP (HyperText Transfer Protocol)
 - MQTT (Message Queue Telemetry Transport)
 - AMQP (Advanced Message Queue Protocol)
 - CoAP (Constrained Application Protocol)
 - XMPP (Extensible Messaging and Presence Protocol)

<https://www.postscapes.com/internet-of-things-technologies/>

1.3.3. Phân tích dữ liệu trong IoT (Data Analytic)



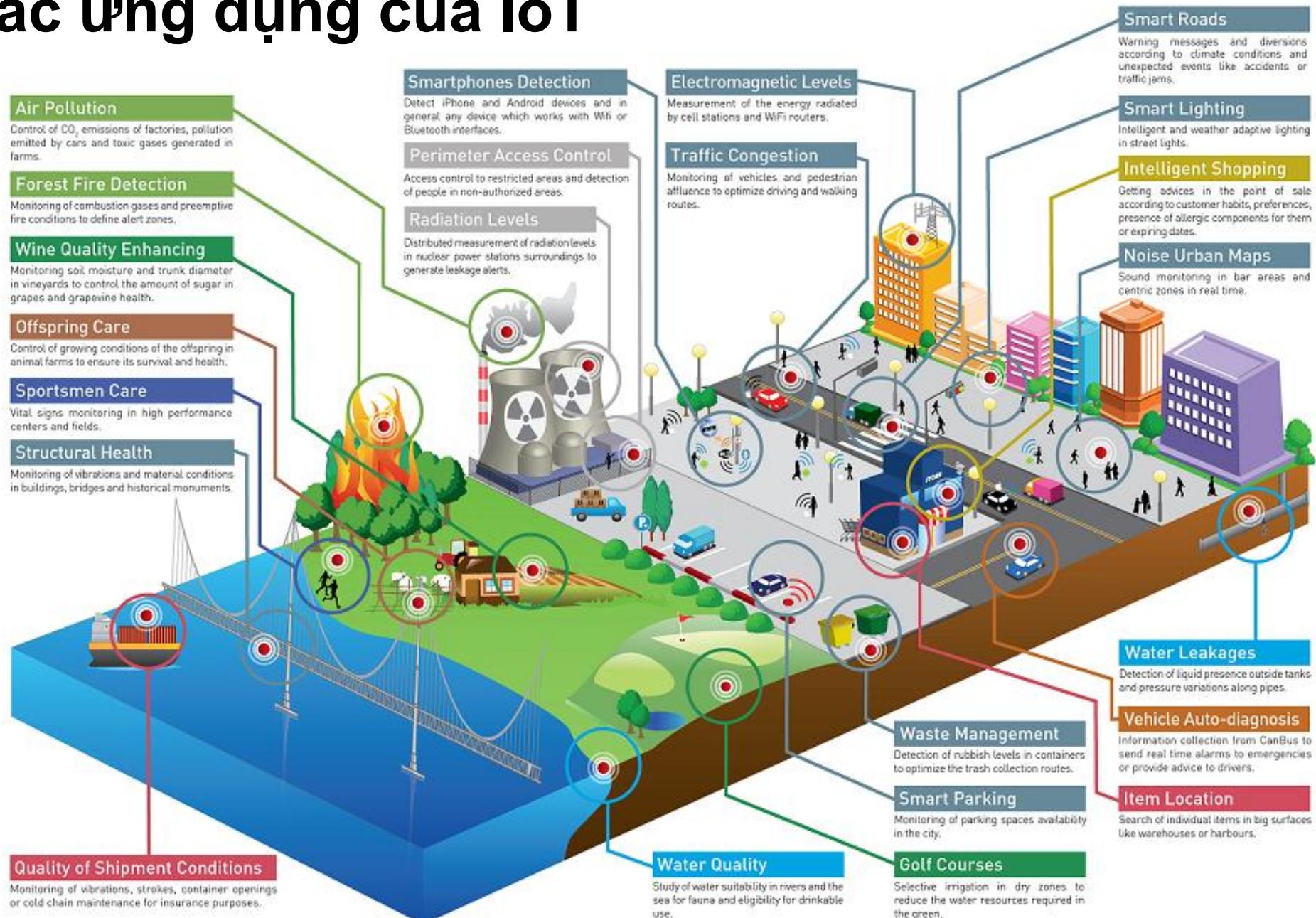
1.3.4. IoT Cloud Platforms

- Một số nền tảng IoT Cloud
 - AWS IoT
 - Google Cloud IoT
 - Azure IoT
 - IBM BlueMix



<https://www.postscapes.com/internet-of-things-technologies/>

1.4. Các ứng dụng của IoT



Các ứng dụng của IoT

- Consumer**
 - Smart home control (lighting, security, comfort)
 - Optimized energy use
 - Maintenance
- Retail**
 - Product tracking
 - Inventory control
 - Focused marketing
- Medical**
 - Wearable devices
 - Implanted devices
 - Telehealth services
- Military**
 - Resource allocation
 - Threat analysis
 - Troop monitoring



- Industrial**
 - Smart Meters
 - Wear-out sensing
 - Manufacturing control
 - Climate control
- Automotive**
 - Parking
 - Traffic flow
 - Anti-theft location
- Environmental**
 - Species tracking
 - Weather prediction
 - Resource management
- Agriculture**
 - Crop management
 - Soil analysis

Top Industrial IoT Applications

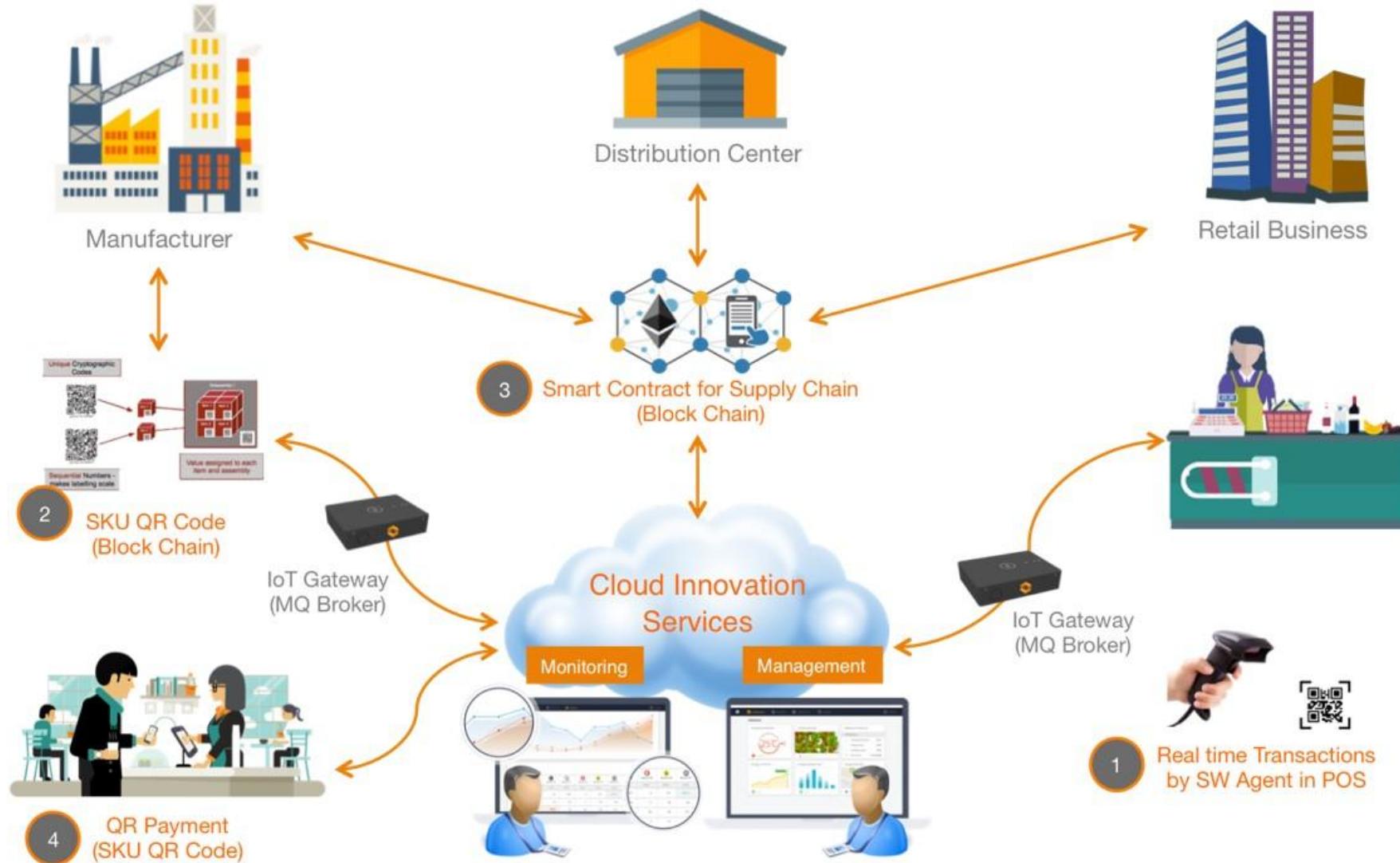
- Healthcare
- Smart Retail
- Smart Building/Smart Home
- Smart Agriculture
- Smart Utilities (Power Energy, Water)

Healthcare

- One of the fastest sectors adopting the IoT
- A lot of sensors for tracking patients



Smart Retail



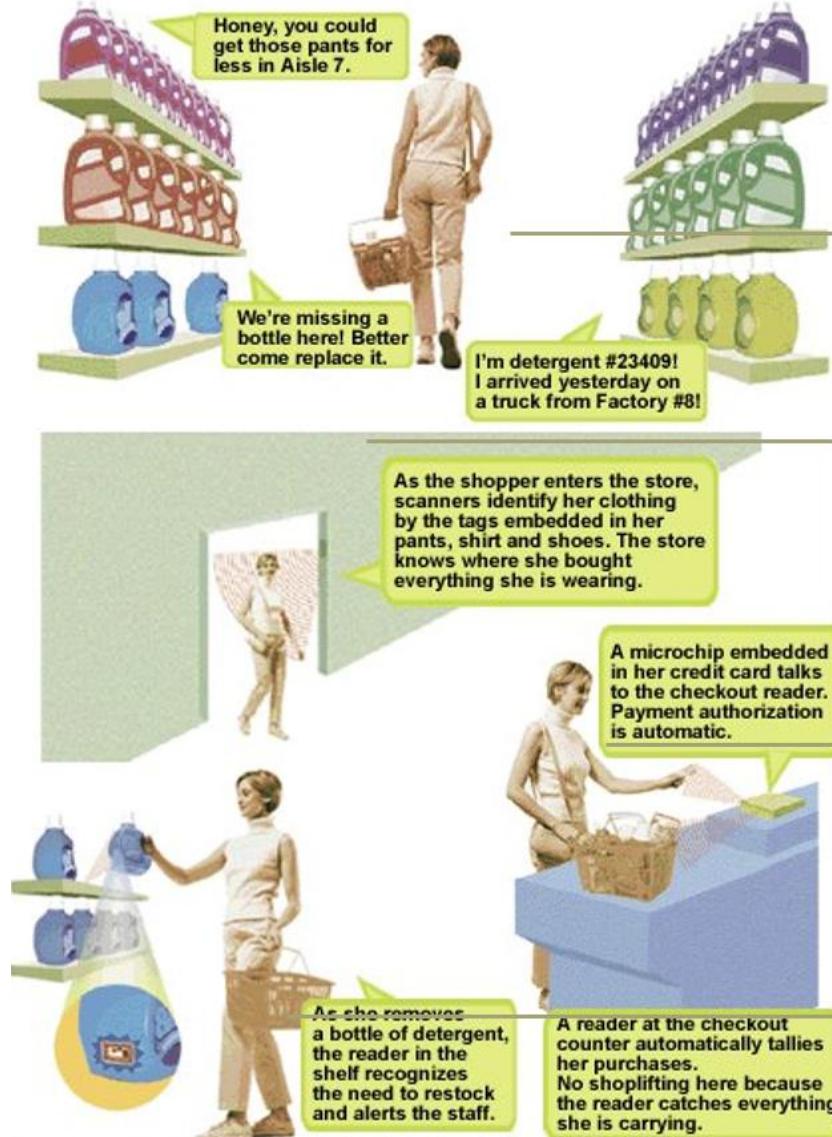
<https://www.smartofthings.co.th/2018/08/27/smart-retail-solution/>

Smart Retail

Amazon Go store



Smart Retail: Shopping



Scenario: shopping

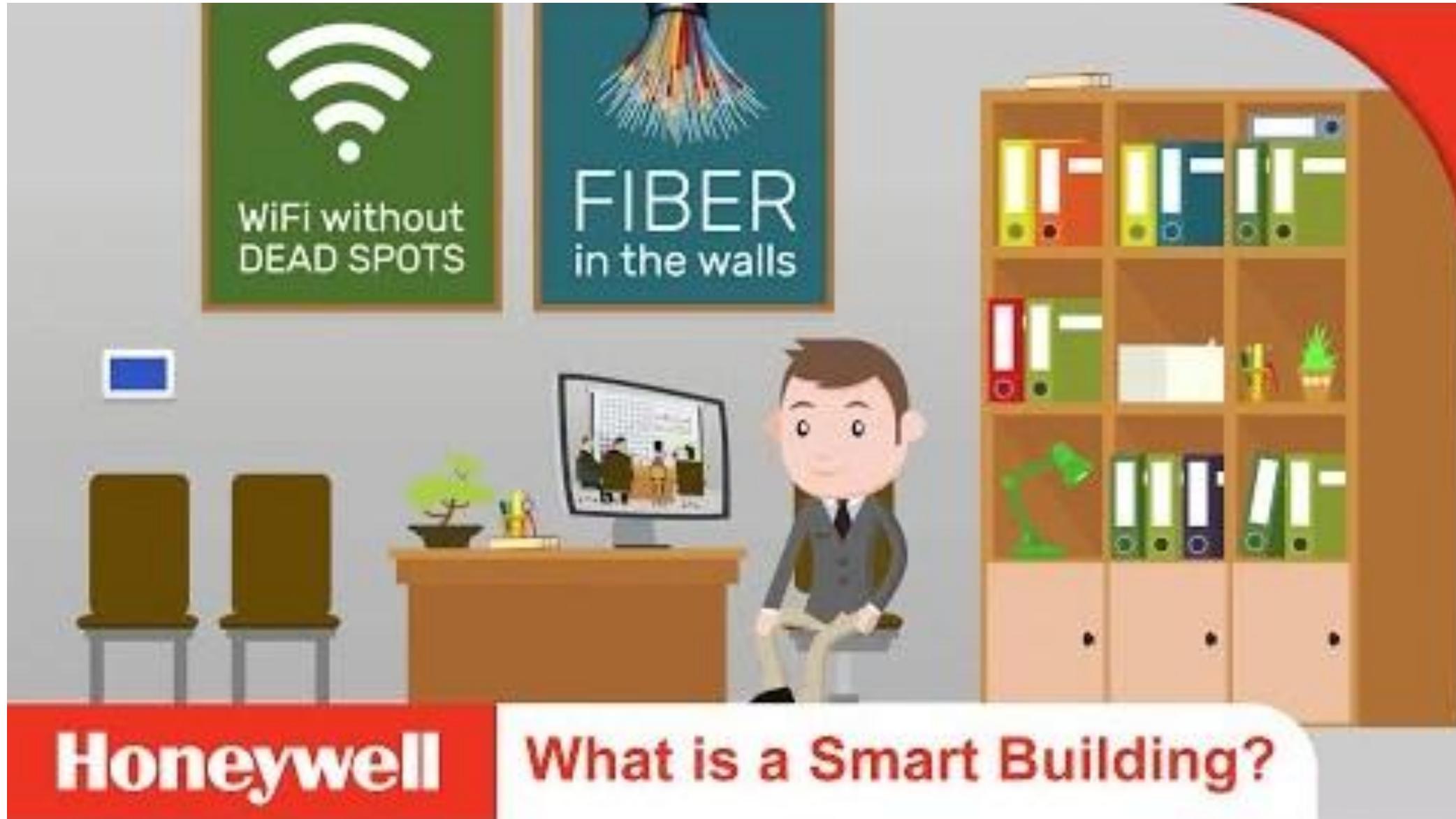
(2) When shopping in the market, the goods will introduce themselves.

(1) When entering the doors, scanners will identify the tags on her clothing.

(4) When paying for the goods, the microchip of the credit card will communicate with checkout reader.

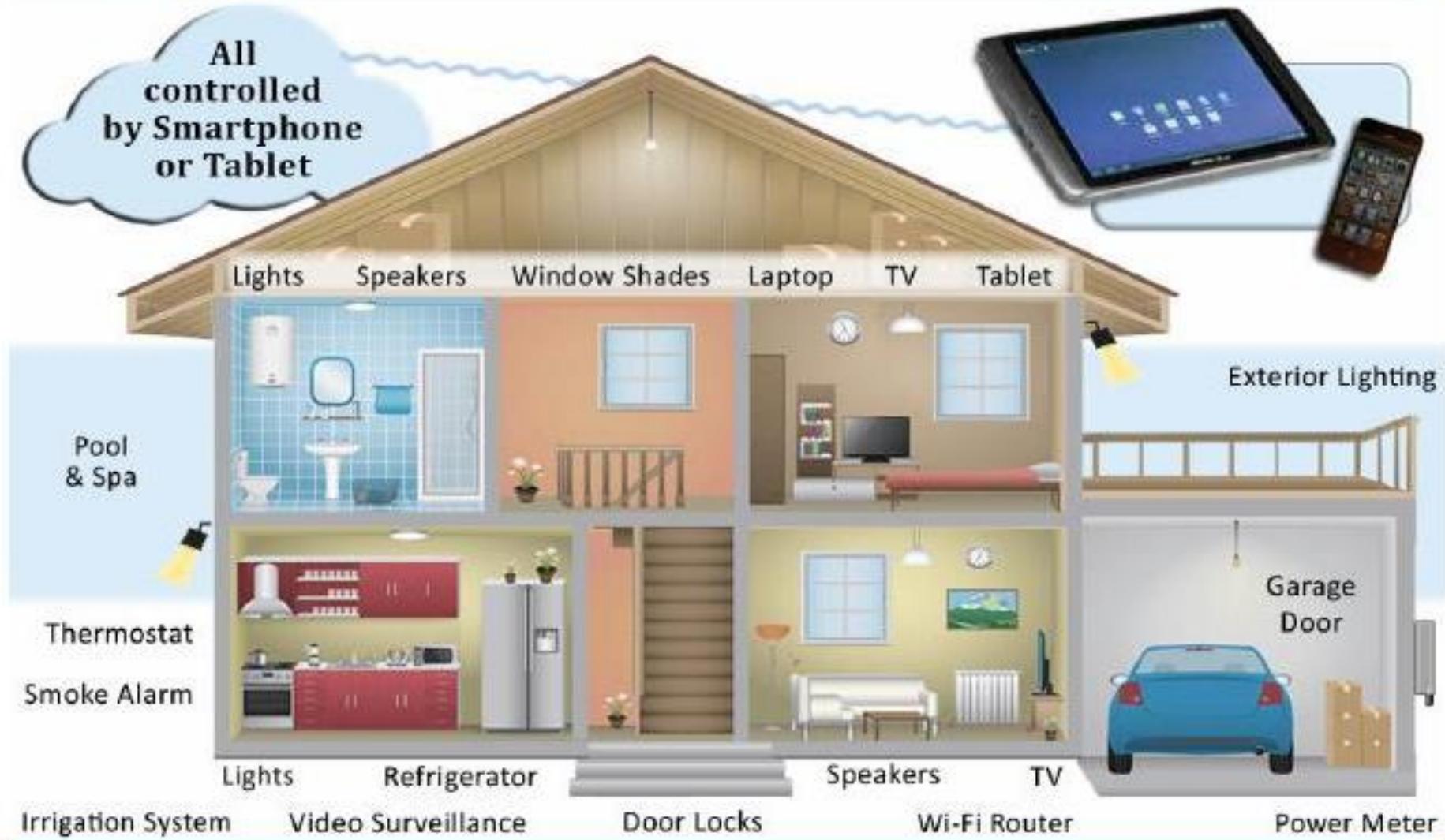
(3) When moving the goods, the reader will tell the staff to put a new one.

Smart Building



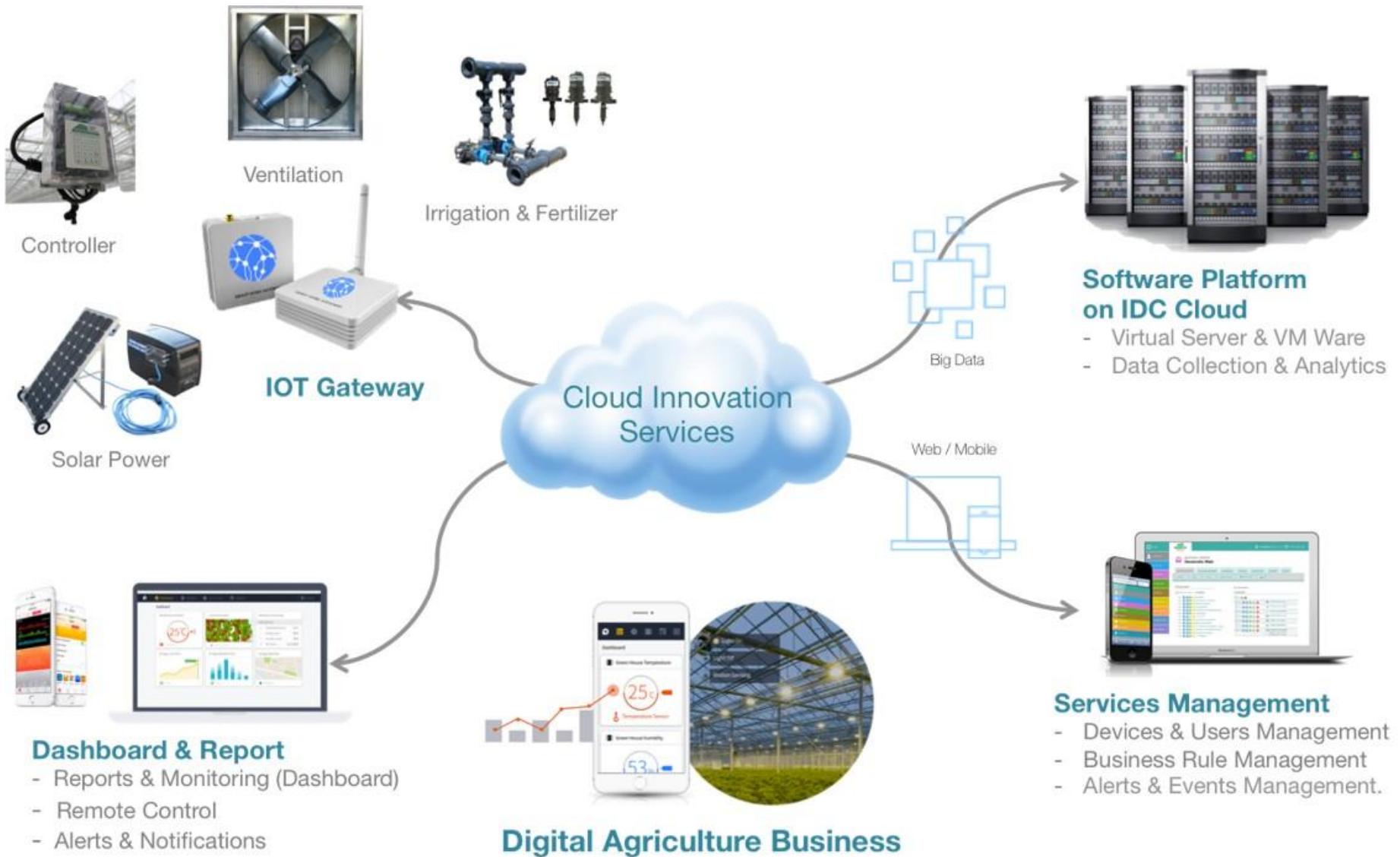
Smart Home

Home Automation



Source: Raymond James research.

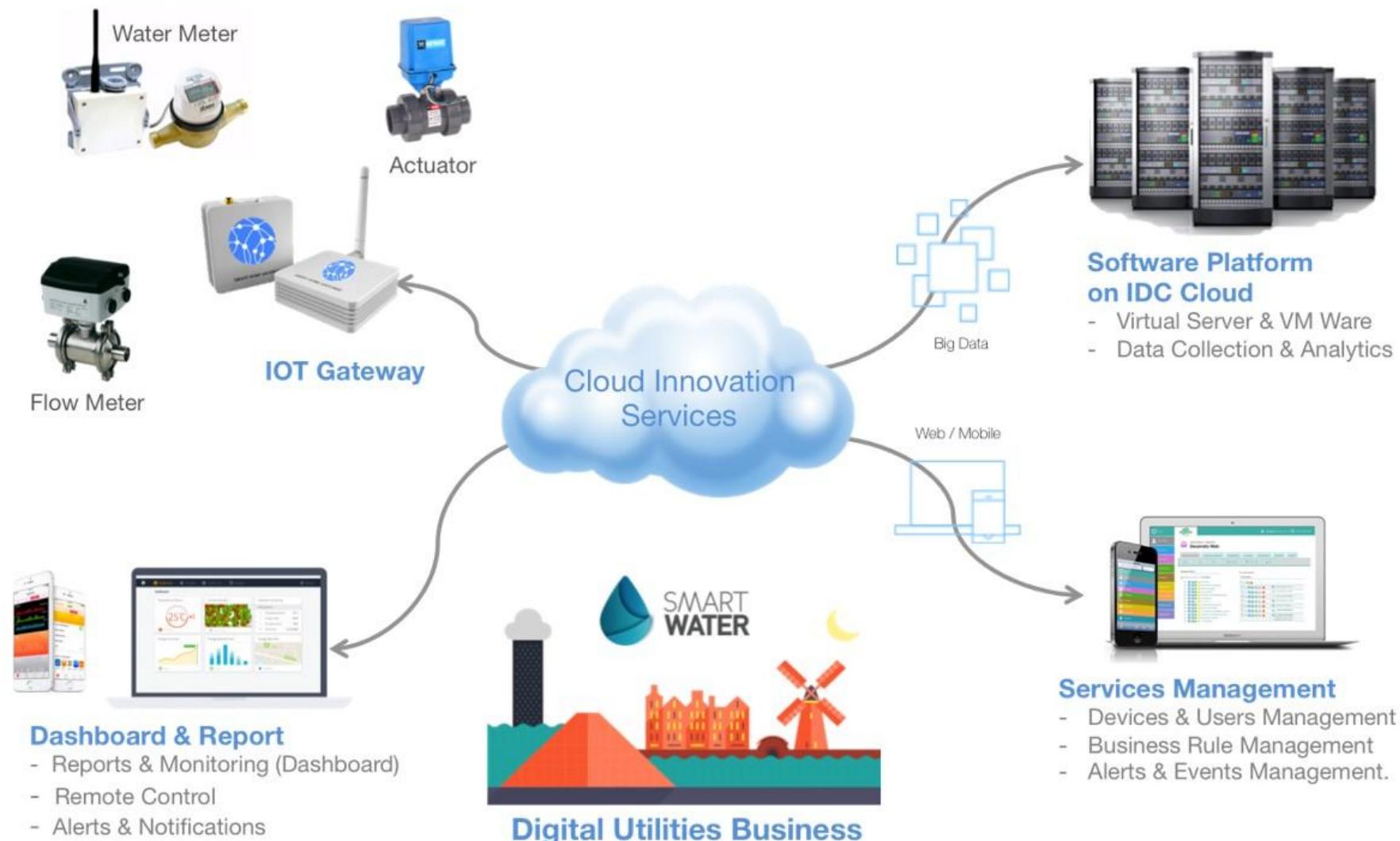
Smart Agriculture



<https://www.smartofthings.co.th/2018/08/27/smart-agriculture-solution/>

Smart Utilities

Electrical power, Water Supplying



<https://www.smartofthings.co.th/2018/08/27/smart-utilities-solution/>

Transportation



Source: Raymond James research.

IoT – Manufacturing Applications

- Intelligent Product Enhancements
- Dynamic Response to Market Demands
- Lower Costs, Optimized Resource Use, Waste Reduction
- Improved Facility Safety
- Product Safety

Một số sản phẩm thương mại ứng dụng IoT



Một số sản phẩm thương mại ứng dụng IoT



Một số sản phẩm thương mại ứng dụng IoT



Thảo luận - Discussion

- What everyday activity can be changed by IoT?
- What existing process can be changed by IoT? Will this change lead to the greater good of society? Or to the individual?

1.5. Các vấn đề thách thức của IoT

- Rất nhiều chuẩn công nghệ khác nhau:
 - Thuận lợi + Khó khăn ?
- Thách thức với các chính phủ trong vấn đề quản trị đổi mới quá nhanh của công nghệ:
 - Ví dụ Uber, Grab
- Vấn đề về quyền riêng tư (Privacy) và bảo mật (security)
 - Ví dụ: Facebook
- Thiếu vắng cơ quan quản lý, điều hành chung
 - Vấn đề chung của dịch vụ Internet
- Dễ bị tấn công trên Internet:
 - Ví dụ IP cameras

Các thách thức kỹ thuật (Challenges) ...

■ Connectivity - Vấn đề kết nối:

- Hiện tại, IoT dựa trên mô hình server/client để xác thực, kết nối các thiết bị trong mạng. Mô hình này đã có thể làm việc với hàng trăm, ngàn thiết bị. Vấn đề khó khăn khi số lượng thiết bị lên tới hàng triệu, tỷ trong mạng
- Nếu không cân nhắc đến thiết kế thông lượng mạng thích hợp, vấn đề tắc nghẽn (bottlenecks) có thể xảy ra trong quá trình trao đổi dữ liệu tại server.
- Các tác vụ có thể chuyển sang thực hiện trên các thiết bị (off-loading tasks to the edge)
- Xu hướng: Các mạng IoT sẽ cần các thiết bị có khả năng thực hiện phân tích xử lý dữ liệu trên thiết bị (edge computing)

Các thách thức kỹ thuật (Challenges) ...

- **Brownfield deployment** (legacy infrastructure) – Vấn đề triển khai trên các hạ tầng cũ đã có:
 - Các thiết bị IoT, các hạ tầng mạng đã có, các công nghệ mới được kết hợp với nhau (brownfield deployment)
 - Các công ty đổi mới với vấn đề tích hợp các thiết bị công nghệ mới với hạ tầng mạng đã tồn tại

Các thách thức kỹ thuật (Challenges) ...

- **Dealing with non-standard communication protocols:**
 - Vấn đề với các giao thức truyền thông phi chuẩn
 - Có nhiều giao thức truyền thông khác nhau được đề xuất, phát triển.

Các thách thức kỹ thuật (Challenges) ...

- **IT/OT convergence – Sự hội tụ của IT và OT**
 - Sự tích hợp/hội tụ của IT (Information Technology) và OT (Operational Technology – Công nghệ vận hành) trong các ứng dụng công nghiệp của Internet of Things (IIoT). Ví dụ về IT, OT trong nhà máy điện
 - IT: trung tâm dữ liệu (data-centric), OT: giám sát sự kiện (monitor events); IoT làm mờ sự phân biệt này thông qua việc giám sát các thiết bị đồng thời tạo ra một lượng lớn dữ liệu
 - Các doanh nghiệp vận hành công nghiệp sẽ cần điều chỉnh qui trình để thích ứng với các thiết bị IIoT và dữ liệu

Các thách thức kỹ thuật (Challenges) ...

- Get actionable intelligence from data – Khai thác khả năng thông minh từ dữ liệu:
 - Giá trị của dữ liệu gia tăng khi khả năng khai thác thông minh từ dữ liệu gia tăng
 - Phân tích IoT sẽ cần làm việc với các dữ liệu chưa được cấu trúc, lượng dữ liệu lớn theo thời gian thực và cả với các dữ liệu ngoại biên

Thảo luận - Discussion

- Industry Transformations
 - IoT will cause the most transformation in non-technology-based industries.
- Give an example of a non-technology-based industry and explain how IoT will transform it.

Nội dung

- Chương 1. Tổng quan về IoT
- Chương 2. Các công nghệ IoT
- Chương 3. Lập trình ứng dụng IoT
- Chương 4. An toàn và Bảo mật IoT
- Chương 5. Thiết kế và xây dựng hệ thống IoT

Chương 2. Các công nghệ IoT

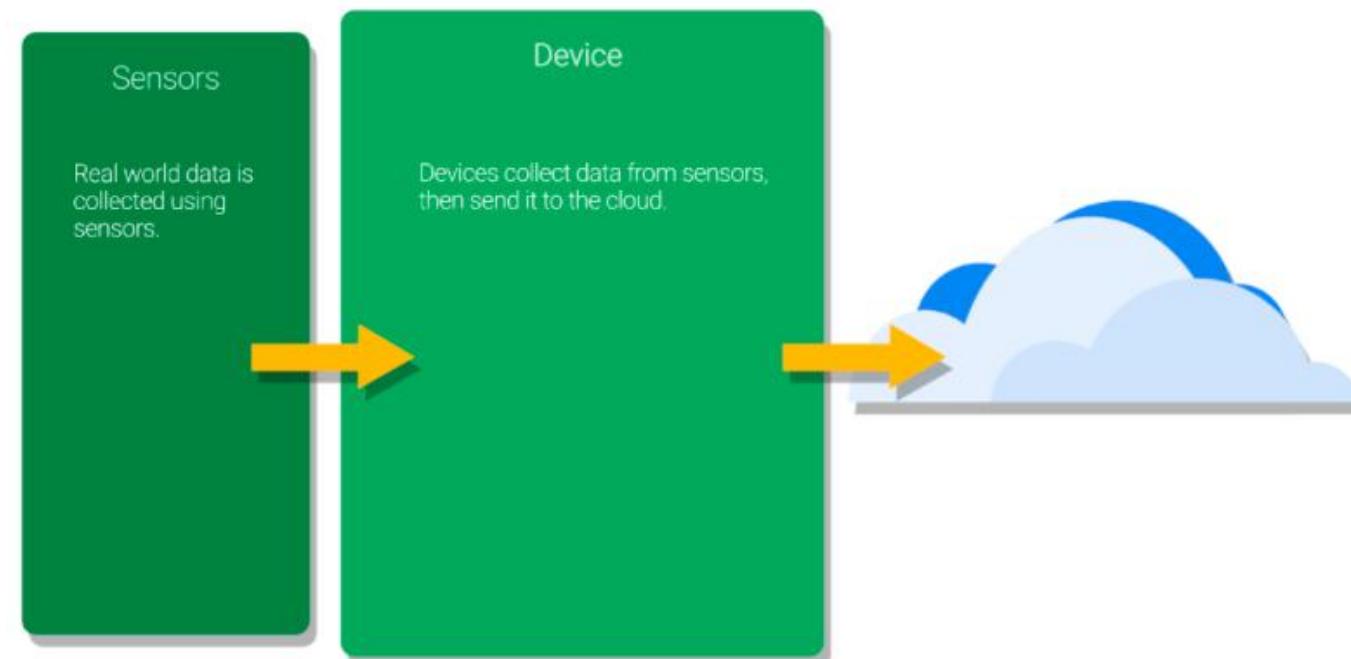
- 2.1. Cảm biến và thiết bị
- 2.2. Một số chuẩn truyền thông cho IoT
- 2.3. Các giao thức cho ứng dụng IoT
- 2.4. Nền tảng đám mây IoT

2.1. Cảm biến và thiết bị IoT

- Cảm biến và thiết bị:
 - Các cảm biến và thiết bị IoT thường được gọi đơn giản là thiết bị (devices) với ngầm định bao gồm cả các cảm biến được ghép nối
 - Trong nhiều tình huống, khi đề cập “thiết bị” có thể được hiểu là bao gồm cả thiết bị và cảm biến

Cảm biến và thiết bị IoT

- Cảm biến quan sát các thay đổi xung quanh và gửi thông tin về các thay đổi đó đến thiết bị
- Các thiết bị thu thập dữ liệu từ các cảm biến và gửi tới cloud/server



Cảm biến và thiết bị IoT

- Các thiết bị có thể hạn chế về tài nguyên tính toán (CPU, bộ nhớ hạn chế, ...)
- Các thiết bị có thể giao tiếp truyền thông riêng mà không kết nối trực tiếp với cloud platform, ví dụ thông qua Bluetooth Low Energy (BLE)
- Các thiết bị có thể lưu trữ, xử lý và phân tích dữ liệu trước khi gửi lên cloud

Phân loại cảm biến (Types of Sensors)

- Theo yêu cầu về nguồn cấp

Type	Definition	Example
passive	Does not require external power to operate. They respond to input from their environment.	A temperature sensor that changes resistance in response to temperature changes
active	Requires external power to operate.	A camera

- Theo loại tín hiệu

Type	Definition	Example
analog	Outputs an analog continuous signal	Accelerometers, temperature sensors
digital	The output is converted to discrete values (digital 1s and 0s) before transmitting to a device	Digital pressure sensor, digital temperature sensor

- Theo loại thiết bị đo

Type	Definition	Example
chemical	Responds to chemical changes in its environment	Gas sensor
mechanical	Responds to physical changes in its environment	Microswitch
electrical	Responds to electrical changes in its environment	Optical sensor

Lựa chọn cảm biến

- Lựa chọn cảm biến cần cân nhắc đến tầm quan trọng của các yếu tố và mức độ ưu tiên trong thiết kế tổng thể.
- **Durability (Tính lâu bền):**
 - Tính lâu bền phải được cân nhắc đến dựa trên môi trường hoạt động của sensor.
 - Đảm bảo thiết bị hoạt động lâu bền trong khoảng thời gian đáng kể không phải tốn chi phí sửa chữa thay thế.
 - Ví dụ: cảm biến nhiệt kháng nước dùng cho trạm thời tiết

Lựa chọn cảm biến

- **Accuracy (Độ chính xác):**
 - Cần độ chính xác đáp ứng đủ nhu cầu (độ chính xác cao đi kèm giá thành đắt)
 - Ví dụ: Đo nhiệt độ nhà kho có thể chấp nhận độ chính xác +/- 2 độ. Đo nhiệt độ hệ thống thiết bị y tế cần độ chính xác +/- 0.2 độ

Lựa chọn cảm biến

- **Versatility (Tính linh hoạt):**

- Cảm biến có thể vận hành với các biến động của môi trường xung quanh
- Ví dụ: cảm biến cho trạm thời tiết có thể hoạt động trong điều kiện mùa hè, mùa đông. Không khả thi với loại cảm biến chỉ hoạt động trong môi trường trong nhà.

Lựa chọn cảm biến

- **Power Consumption (Tiêu thụ điện năng):**
 - Tùy theo tình huống, có thể đòi hỏi các thiết bị tiêu thụ ít điện năng, sử dụng các đặc tính tiết kiệm năng lượng
 - Ví dụ: cảm biến hoặc thiết bị sử dụng pin năng lượng mặt trời cần thiết kế chế độ ngủ (sleep mode) tiết kiệm năng lượng, thức dậy (wake-up) khi cần thu thập và truyền dữ liệu

Lựa chọn cảm biến

- Cân nhắc môi trường đặc biệt:
 - Ví dụ: Cảm biến trong giám sát chất lượng nước trong nuôi thủy sản đòi hỏi sensor đặt trong môi trường nước (pH, DO, TDS, ...) khác với các cảm biến dựa trên lấy mẫu phân tích.

Lựa chọn cảm biến

- **Cost (Chi phí):**
 - Các mạng IoT thường bao gồm hàng trăm, ngàn thiết bị và cảm biến.
 - Thiết kế cần cân nhắc đến chi phí lắp đặt, bảo trì, thay thế, etc.

Devices – Thiết bị IoT

- Một “Thing” trong “Internet of Things” là một đơn vị xử lý mà có khả năng kết nối vào internet để trao đổi dữ liệu với cloud.
- Các thiết bị thường được gọi là "smart devices" or "connected devices."
- Thiết bị giao tiếp với 2 loại dữ liệu: telemetry (dữ liệu thu thập) và state (trạng thái của thiết bị).

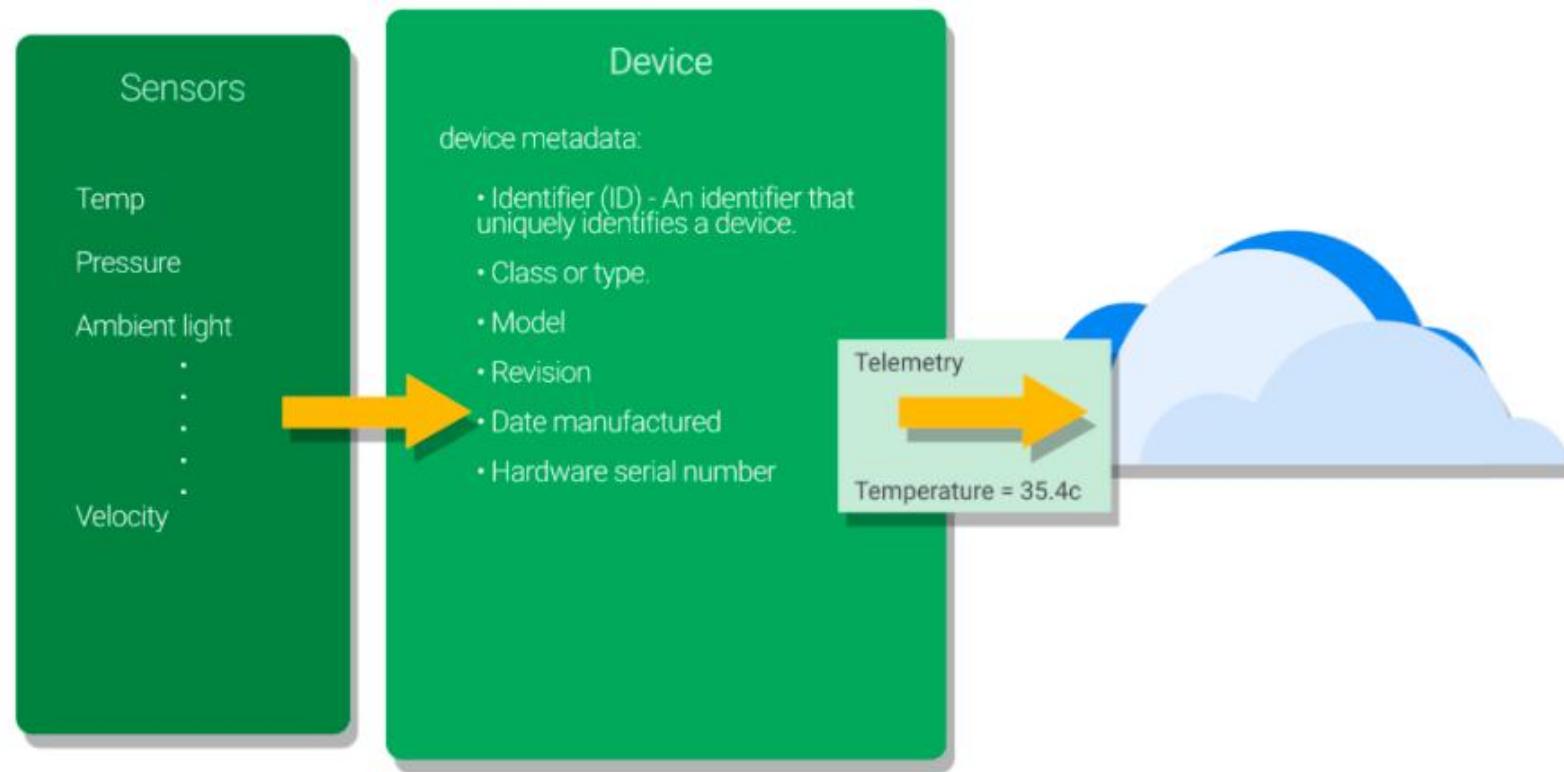
Các loại thông tin

- Mỗi thiết bị có thể cung cấp hoặc sử dụng một vài loại thông tin phục vụ cho mỗi loại hệ thống backend khác nhau.
- **Device metadata:** Metadata chứa các thông tin mô tả về thiết bị (thường ít thay đổi). Ví dụ:
 - Identifier (ID) – Định danh thiết bị
 - Class or type: Lớp/loại thiết bị
 - Model
 - Revision
 - Date manufactured
 - Hardware serial number

Các loại thông tin

■ Telemetry:

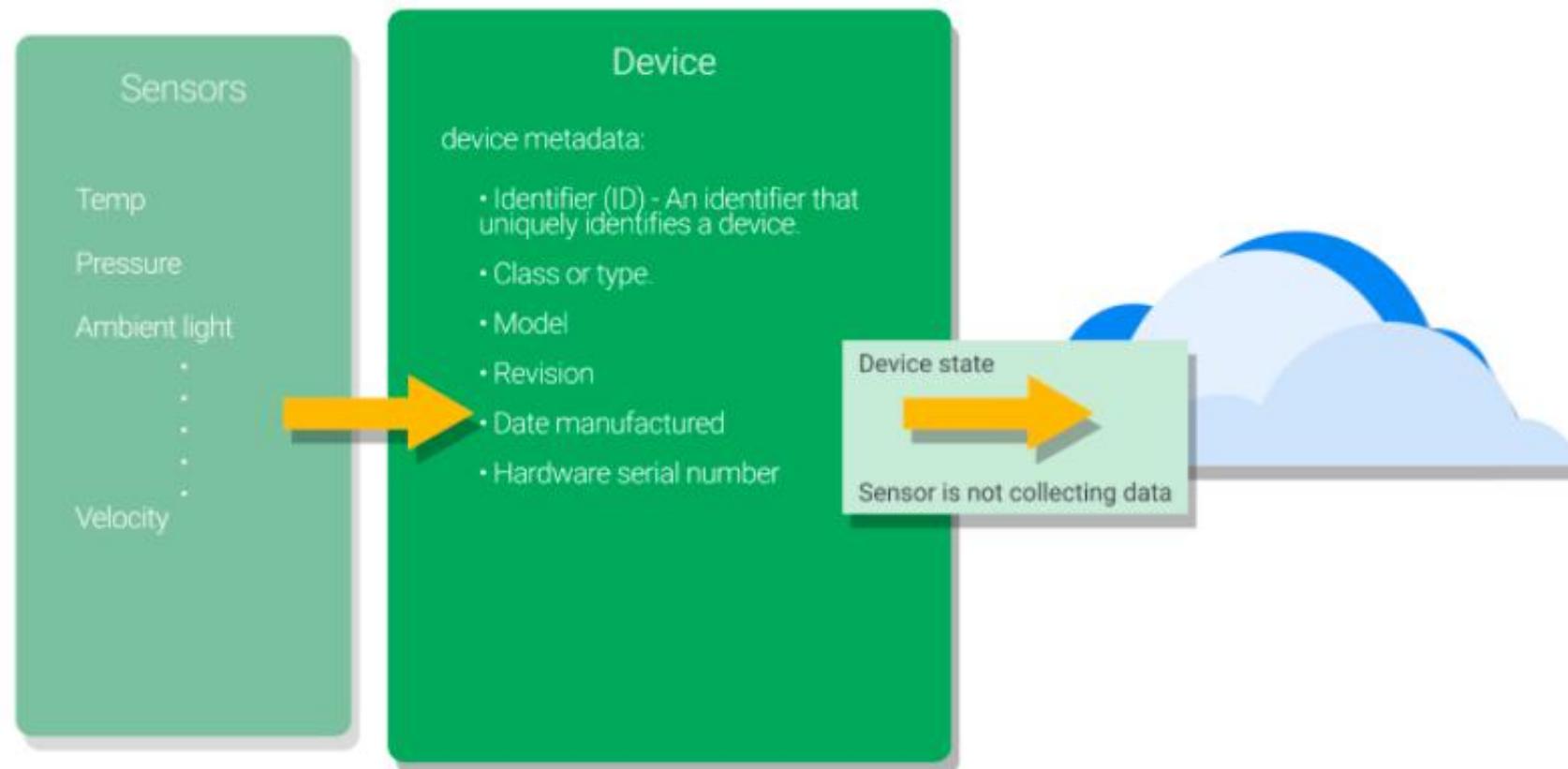
- Dữ liệu được thu thập bởi thiết bị qua các cảm biến được gọi là telemetry.
- Là dữ liệu chỉ đọc (từ môi trường xung quanh)



Các loại thông tin

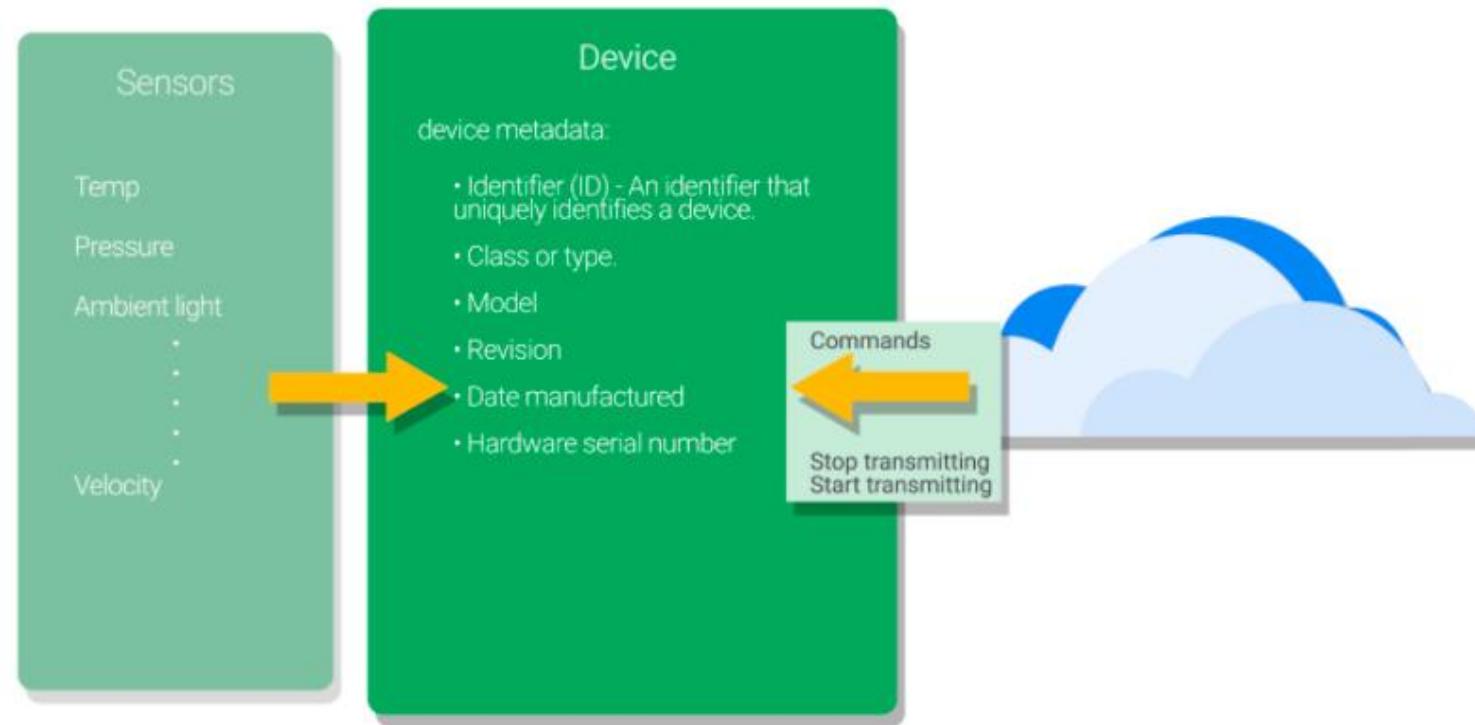
■ State information:

- Thông tin trạng thái: Mô tả trạng thái hiện thời của thiết bị. Có thể đọc/ghi, cập nhật (nhưng ít thường xuyên)



Lệnh thực thi trên thiết bị (Device Commands)

- Commands: Là các hành động được thực hiện bởi thiết bị.
- Ví dụ:
 - Quay 360 độ sang phải.
 - Tắt/bật đèn
 - Tăng tần suất truyền dữ liệu



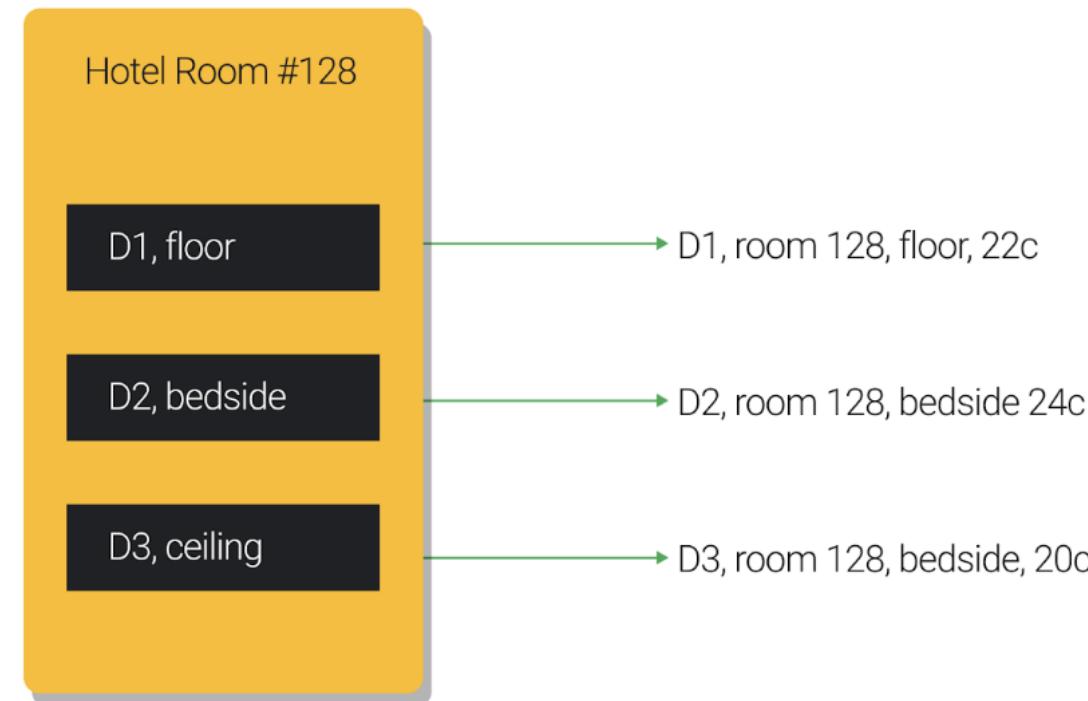
Định nghĩa thiết bị

- Trong hệ thống IoT, định nghĩa thiết bị có thể thay đổi phụ thuộc vào nhu cầu của dự án.
- Mỗi thiết bị có thể được cân nhắc như một thực thể tách biệt, hoặc một thiết bị có thể chịu trách nhiệm với một nhóm cảm biến.

Định nghĩa Thiết bị – Ví dụ

- Giám sát nhiệt độ phòng khách sạn
- **Giải pháp 1:** Mỗi phòng có 3 cảm biến: one near the floor, one near the bed, and one near the ceiling.

One hotel room, 3 sensors, 3 devices



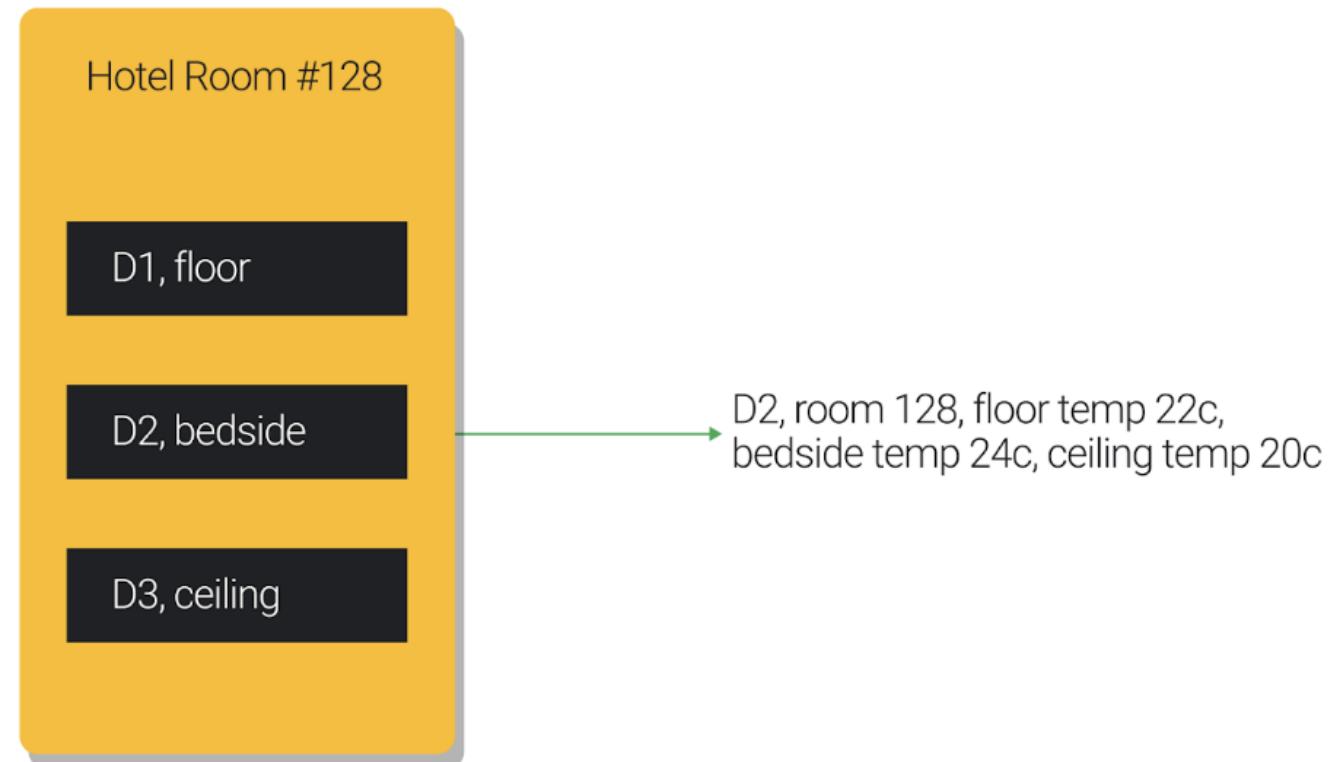
Định nghĩa Thiết bị – Ví dụ

- **Giải pháp 1:** dữ liệu được gửi đến cloud như là 3 thiết bị khác nhau, mỗi cái gửi một thông tin nhiệt độ đến cloud.
 - {deviceID: "dh28dskja", "location": "floor", "room": 128, "temp": 22 }
 - {deviceID: "8d3kiuhs8a", "location": "ceiling", "room": 128, "temp": 24 }
 - {deviceID: "kd8s8hh3o", "location": "bedside", "room": 128, "temp": 23 }

Định nghĩa Thiết bị – Ví dụ

- **Giải pháp 2:** mỗi phòng sử dụng 1 thiết bị chịu trách nhiệm gửi dữ liệu từ 3 cảm biến.

One hotel room, 3 sensors, 1 device



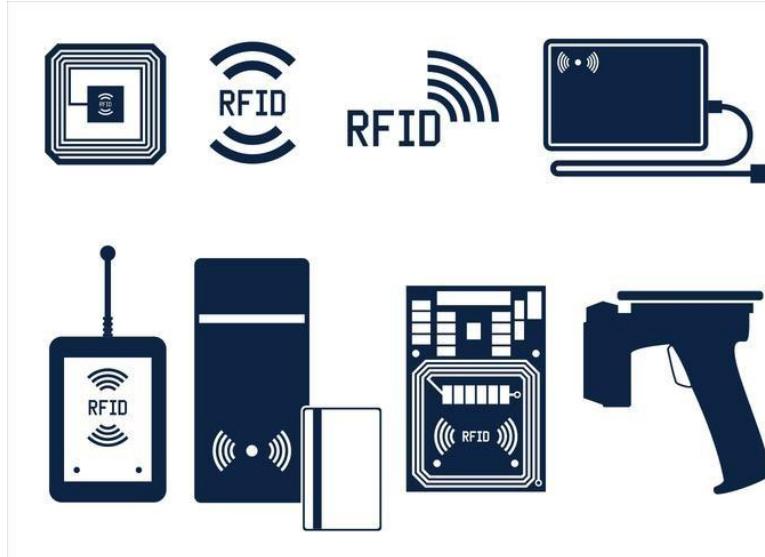
Định nghĩa Thiết bị – Ví dụ

- **Giải pháp 2:** dữ liệu được gửi đến cloud bằng 1 thiết bị với 3 cảm biến. Có thể bổ sung thêm xử lý tính nhiệt độ trung bình trên thiết bị.
 - {deviceID: "dh28dskja", "room": 128, "temp_floor": 22, "temp_ceiling": 24, "temp_bedside": 23, "average_temp": 23 }
- Việc lựa chọn giải pháp nào phụ thuộc vào ý định sử dụng thông tin ở thời điểm hiện tại và cả tiềm năng trong tương lai.

2.2. Một số chuẩn truyền thông

- Communication standards:
 - NFC and RFID
 - Bluetooth
 - WiFi
 - GSM, GPRS, 2G/3G/4G, LTE
 - ZigBee
 - 6LoWPAN
 - Thread

Một số chuẩn truyền thông trong IoT



<https://www.youtube.com/watch?v=MAA9JpGraoU>

Frequency: 120–150 kHz (LF), 13.56 MHz (HF), 433 MHz (UHF), 865-868 MHz (Europe) 902-928 MHz (North America) UHF, 2450-5800 MHz (microwave), 3.1–10 GHz (microwave)

Range: 10cm to 200m

Examples: Road tolls, Building Access, Inventory

Một số chuẩn truyền thông trong IoT

- NFC (Near-Field Communications)



Frequency: 13.56 MHz

Range: < 0.2 m

Examples: Smart Wallets/Cards, Action Tags, Access Control

Một số chuẩn truyền thông trong IoT

- Bluetooth



Frequency: 2.4GHz

Range: 1-100m

Examples: Hands-free headsets, key dongles, [fitness trackers](#)

Một số chuẩn truyền thông trong IoT

- WiFi



Frequency: 2.4 GHz, 3.6 GHz and 4.9/5.0 GHz bands.

Range: Common range is up to 100m but can be extended.

Applications: Routers, Tablets, etc

More: <https://www.postscapes.com/internet-of-things-technologies/>

Một số chuẩn truyền thông trong IoT

- GSM (Global System for Mobile communications)



Frequency: Europe: 900MHz & 1.8GHz, US: 1.9GHz & 850MHz, Full List can be found [here](#).

Data Rate: 9.6 kbps

Examples: Cell phones, M2M, smart meter, asset tracking

More: <https://www.postscapes.com/internet-of-things-technologies/>

Một số chuẩn truyền thông trong IoT

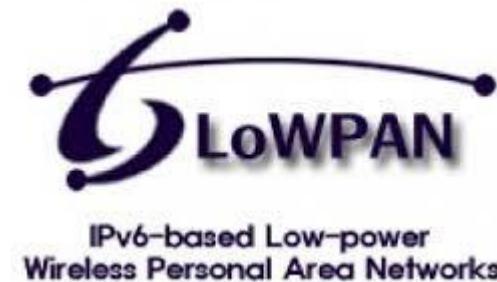
- ZigBee



- Standard: ZigBee 3.0 based on IEEE802.15.4
 - Frequency: 2.4GHz
 - Range: 10-100m
 - Data Rates: 250kbps

Một số chuẩn truyền thông trong IoT

- 6LoWPAN
- IPv6 protocol over low-power wireless PANs (sử dụng giao thức IPv6 trong các mạng PAN không dây công suất thấp)



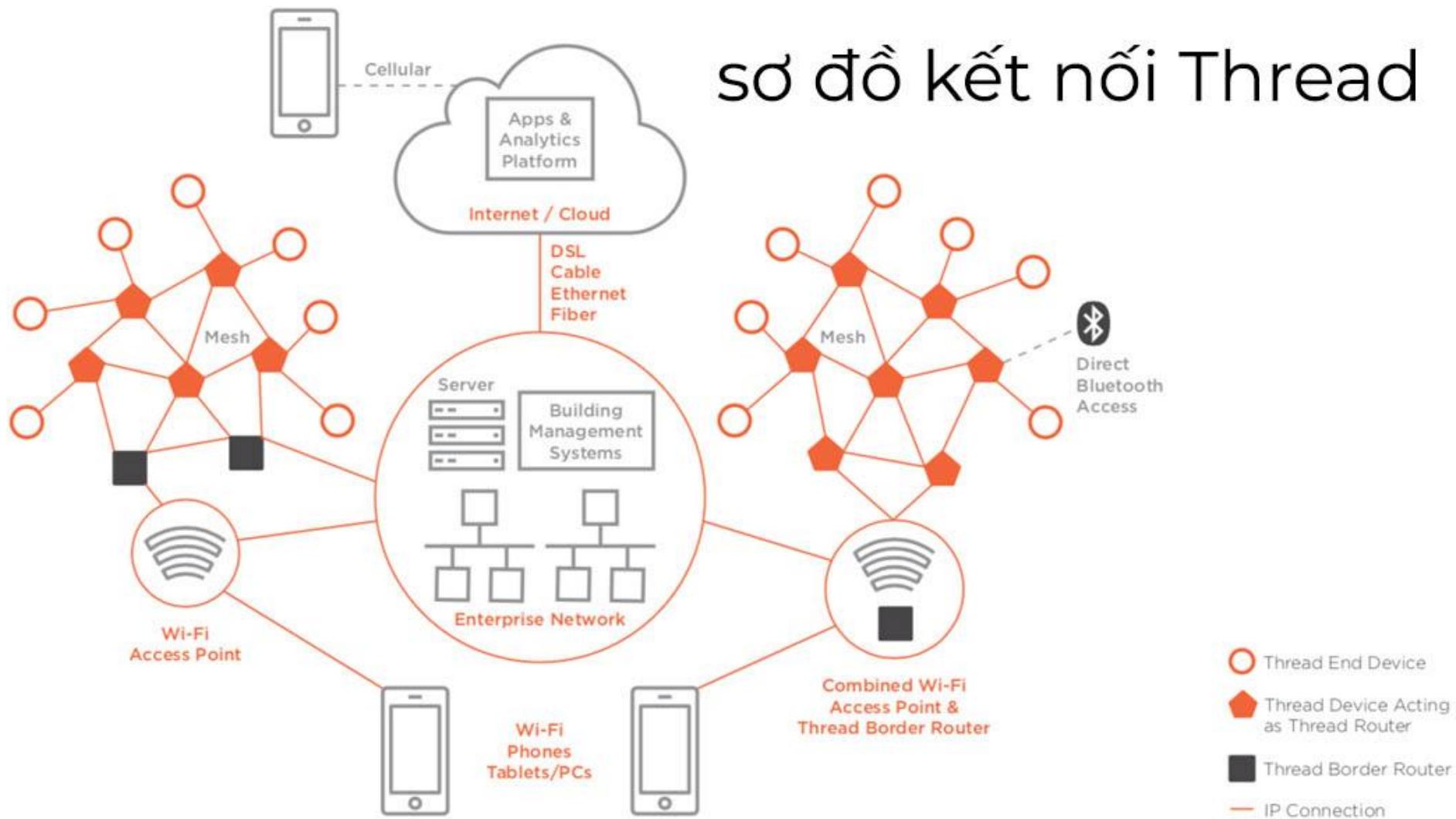
Một số chuẩn truyền thông trong IoT

- Thread: Giao thức IP mới dựa trên nền tảng IPv6 được thiết kế riêng cho mảng tự động hóa trong nhà thông minh, các tòa nhà.
- Ra mắt vào giữa năm 2014 bởi Theard Group, giao thức Thread dựa trên các tiêu chuẩn khác nhau, bao gồm IEEE802.15.4, IPv6 và 6LoWPAN, và cung cấp một giải pháp dựa trên nền tảng IP cho các ứng dụng IoT
- Tiêu chuẩn: Theard, dựa trên IEEE802.15.4 và 6LowPAN.
- Tần số: 2.4GHz (ISM).
- Tham khảo: <https://smarthomekit.vn/thread-protocol/>

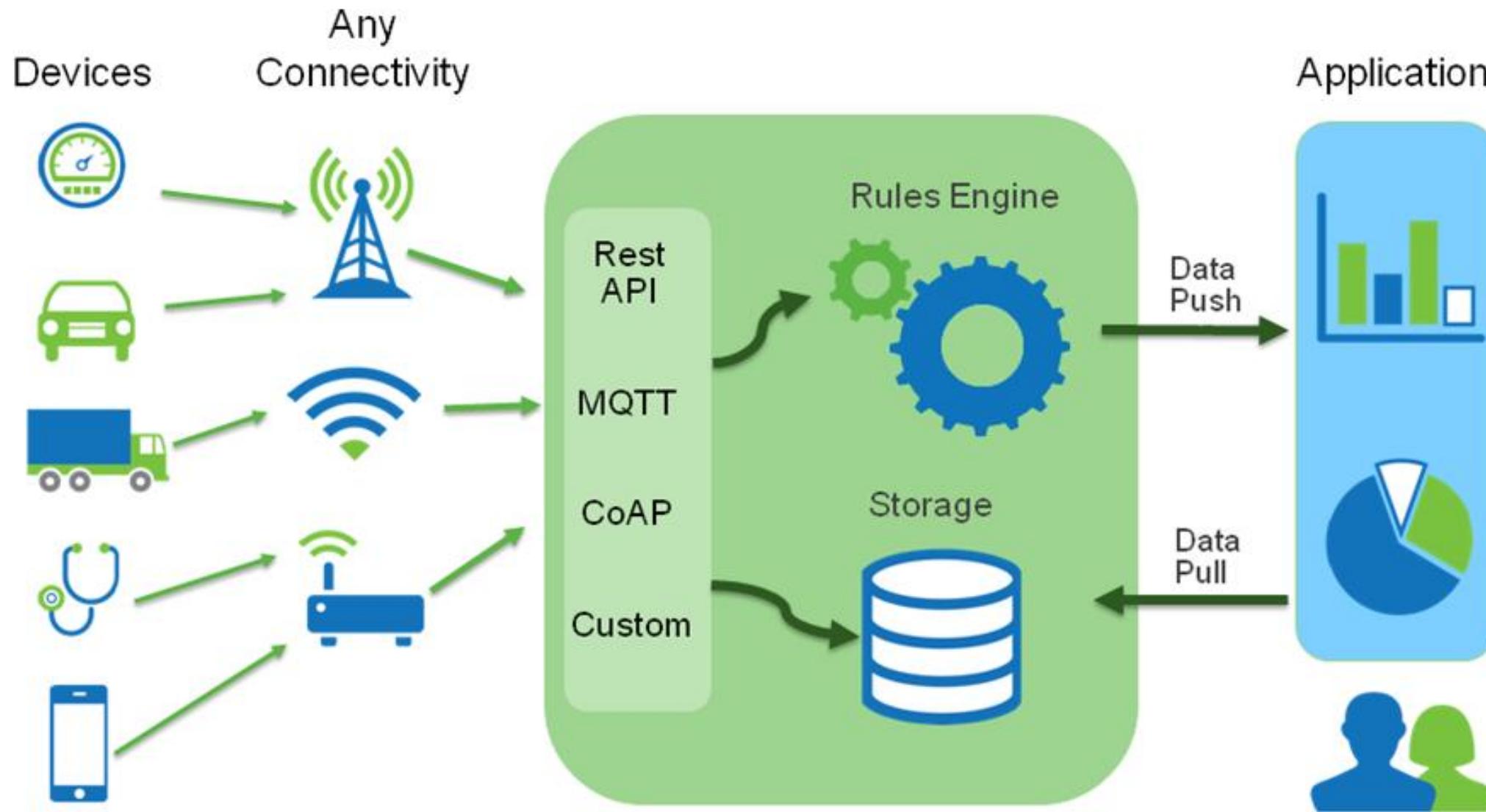
Một số chuẩn truyền thông trong IoT

■ Thread

sơ đồ kết nối Thread



2.3. Một số giao thức cho ứng dụng IoT



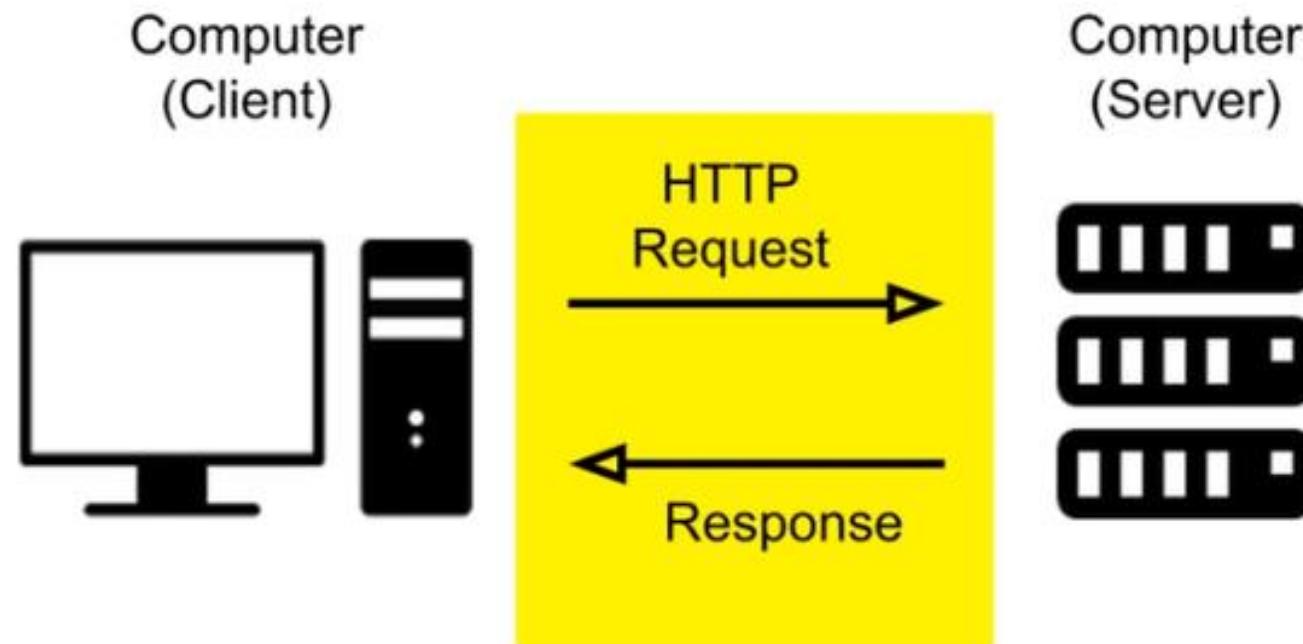
Một số giao thức cho ứng dụng IoT

- HTTP, HTTPS
- MQTT (Message Queue Telemetry Transport)
- AMQP (Advanced Message Queue Protocol)

<https://www.postscapes.com/internet-of-things-technologies/>

2.3.1. Trao đổi dữ liệu dùng HTTP

HTTP = HyperText Transfer Protocol



Ví dụ 1: Trao đổi dữ liệu sử dụng http với ThingSpeak server

- Viết 1 chương trình java gửi dữ liệu đến ThingSpeak server
- a) Dữ liệu được đóng gói trong url (url-encoded)
- GET https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5&field1=20&field2=33

Ví dụ 1.a): Gửi dữ liệu trong url-encoded

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class App {
    public static void main(String[] args) {
        try {
            // URL của API
            String url =
"https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5&field1=21&field2=79";
            // Tạo đối tượng URL
            URL obj = new URL(url);
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
            // Thiết lập phương thức GET
            con.setRequestMethod("GET");
            // Kiểm tra mã phản hồi
            int responseCode = con.getResponseCode();
            System.out.println("Response Code: " + responseCode);
        }
    }
}
```

Ví dụ 1.a): Gửi dữ liệu trong url-encoded

```
// Đọc phản hồi từ server
BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
String inputLine;
StringBuilder response = new StringBuilder();

while ((inputLine = in.readLine()) != null) {
    response.append(inputLine);
}
in.close();

// In kết quả phản hồi
System.out.println("Response: " + response.toString());

} catch (Exception e) {
    e.printStackTrace();
}
}
```

Ví dụ 1.a): Gửi dữ liệu trong url-encoded

1. Tạo đối tượng URL:

```
java
```

Sao chép

Chỉnh sửa

```
String url = "https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5&field1=20&field2  
URL obj = new URL(url);
```

Ở đây, một chuỗi URL được tạo với các tham số truy vấn cần thiết (bao gồm khóa API và các giá trị cho `field1` và `field2`). Sau đó, một đối tượng `URL` được khởi tạo từ chuỗi này.

Ví dụ 1.a): Gửi dữ liệu trong url-encoded

2. Mở kết nối HTTP:

java

Sao chép

Chỉnh sửa

```
HttpURLConnection con = (HttpURLConnection) obj.openConnection();
```

Phương thức `openConnection()` được gọi trên đối tượng `URL` để mở một kết nối tới server. Kết nối này được ép kiểu về `HttpURLConnection` để sử dụng các phương thức cụ thể cho giao thức HTTP.

Ví dụ 1.a): Gửi dữ liệu trong url-encoded

3. Thiết lập phương thức yêu cầu:

java

Sao chép

Chỉnh sửa

```
con.setRequestMethod("GET");
```

Phương thức yêu cầu được đặt là "GET" bằng cách sử dụng `setRequestMethod`. Điều này xác định loại yêu cầu HTTP sẽ được gửi đến server.

Ví dụ 1.a): Gửi dữ liệu trong url-encoded

4. Gửi yêu cầu và nhận mã phản hồi:

java

Sao chép

Chỉnh sửa

```
int responseCode = con.getResponseCode();
System.out.println("Response Code: " + responseCode);
```

Sau khi thiết lập kết nối, phương thức `getResponseCode()` được gọi để gửi yêu cầu đến server và nhận mã trạng thái HTTP từ phản hồi. Mã này cho biết kết quả của yêu cầu (ví dụ: 200 cho thành công).

Ví dụ 1.a): Gửi dữ liệu trong url-encoded

5. Đọc và hiển thị phản hồi từ server:

java

Sao chép

Chỉnh sửa

```
BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
String inputLine;
StringBuilder response = new StringBuilder();
while ((inputLine = in.readLine()) != null) {
    response.append(inputLine);
}
in.close();
System.out.println("Response: " + response.toString());
```

Ví dụ 1: Trao đổi dữ liệu sử dụng http với ThingSpeak server

- Viết 1 chương trình java gửi dữ liệu đến ThingSpeak server
- b) Gửi dữ liệu được đóng gói trong request body (json data)
- POST
https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5
- Ví dụ: request body. { "field1": 20, "field2": 33 }

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

```
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class App {
    public static void main(String[] args) {
        try {
            // URL với API key
            URL url = new
URL("https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5");

            // Mở kết nối
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            // Thiết lập phương thức POST và định dạng JSON
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type", "application/json");
            conn.setDoOutput(true);
```

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

```
// Nội dung JSON cần gửi
String jsonData = "{\"field1\": 20, \"field2\": 33}";
// Gửi dữ liệu JSON vào body request
try (OutputStream os = conn.getOutputStream()) {
    byte[] input = jsonData.getBytes("utf-8");
    os.write(input, 0, input.length);
}
// Lấy mã phản hồi
int responseCode = conn.getResponseCode();
System.out.println("Response Code: " + responseCode);

if (responseCode == HttpURLConnection.HTTP_OK) {
    System.out.println("Gửi dữ liệu thành công lên ThingSpeak.");
} else {
    System.out.println("Gửi dữ liệu thất bại.");
}
conn.disconnect();
} catch (Exception e) {
    e.printStackTrace();
}
}
```

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

1
2
3
4

Bước 1: Tạo URL kết nối đến ThingSpeak

java

 Copy

```
URL url = new URL("https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5");
```

- Khởi tạo đối tượng `URL` với endpoint của ThingSpeak.
- Gắn sẵn API Key trong URL để xác thực việc gửi dữ liệu.

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

1
2
3
4

Bước 2: Mở kết nối HTTP

java

 Copy

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

- Tạo kết nối từ URL bằng `HttpURLConnection`.
- Đây là đối tượng trung gian để gửi/nhận dữ liệu qua giao thức HTTP.

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

1
2
3
4

Bước 3: Cấu hình yêu cầu HTTP

java

 Copy

```
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "application/json");
conn.setDoOutput(true);
```

- Thiết lập phương thức là `POST` vì ta cần gửi dữ liệu lên server.
- Thêm header `"Content-Type"` để thông báo dữ liệu gửi là JSON.
- `setDoOutput(true)` cho phép gửi dữ liệu qua `OutputStream`.

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

12
34

Bước 4: Gửi dữ liệu JSON lên ThingSpeak

java

```
String jsonData = "{\"field1\": 20, \"field2\": 33}";  
  
try (OutputStream os = conn.getOutputStream()) {  
    byte[] input = jsonData.getBytes("utf-8");  
    os.write(input, 0, input.length);  
}
```

- Tạo nội dung dữ liệu ở dạng JSON (gồm 2 trường field1 và field2).
- Mở luồng `OutputStream` để ghi dữ liệu vào body của POST request.
- Gửi toàn bộ nội dung JSON lên ThingSpeak.

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

1
2
3
4

Bước 5: Đọc mã phản hồi từ server

java

```
int responseCode = conn.getResponseCode();
System.out.println("Response Code: " + responseCode);
```

- Gọi `getResponseCode()` để lấy mã phản hồi HTTP từ server.
- Ví dụ: `200 OK`, `202 Accepted`, `400 Bad Request`, v.v.

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

Bước 6: Kiểm tra kết quả và thông báo

java

Copy

Edit

```
if (responseCode == HttpURLConnection.HTTP_OK || responseCode == HttpURLConnection.HTTP_ACCEP  
    System.out.println("Gửi dữ liệu thành công lên ThingSpeak.");  
} else {  
    System.out.println("Gửi dữ liệu thất bại.");  
}
```

- Nếu mã phản hồi là 200 hoặc 202, thông báo gửi thành công.
- Nếu khác, in thông báo thất bại để dễ debug.

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

1
2
3
4

Bước 7: Đóng kết nối

java

 Copy

```
conn.disconnect();
```

- Sau khi hoàn tất, đóng kết nối HTTP để giải phóng tài nguyên.

Ví dụ 1.b): Gửi dữ liệu json đóng gói trong request body

- Bổ sung: Đọc nội dung phản hồi từ server (http response)

```
// 5. Đọc mã phản hồi
int responseCode = conn.getResponseCode();
System.out.println("Response Code: " + responseCode);

// 6. Đọc nội dung phản hồi từ server
BufferedReader in = new BufferedReader(new InputStreamReader(
    conn.getInputStream(), "utf-8"));

StringBuilder response = new StringBuilder();
String line;
while ((line = in.readLine()) != null) {
    response.append(line);
}
in.close();

// 7. In nội dung phản hồi ra màn hình ↓
System.out.println("Response Body: " + response.toString());
```

Gửi dữ liệu http bằng chương trình python

```
import requests
import json
# URL của ThingSpeak kèm theo API key
url = "https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5"
# Dữ liệu JSON cần gửi
data = {
    "field1": 20,
    "field2": 33
}
# Gửi POST request
response = requests.post(url, json=data)
# In mã phản hồi và nội dung phản hồi
print("Response Code:", response.status_code)
print("Response Body:", response.text)

# Kiểm tra kết quả
if response.status_code == 200:
    print("Gửi dữ liệu thành công.")
else:
    print("Gửi dữ liệu thất bại.")
```

Gửi dữ liệu http bằng chương trình python

🛠️ Hướng dẫn:

- Cài thư viện `requests` nếu chưa có:

```
bash
```

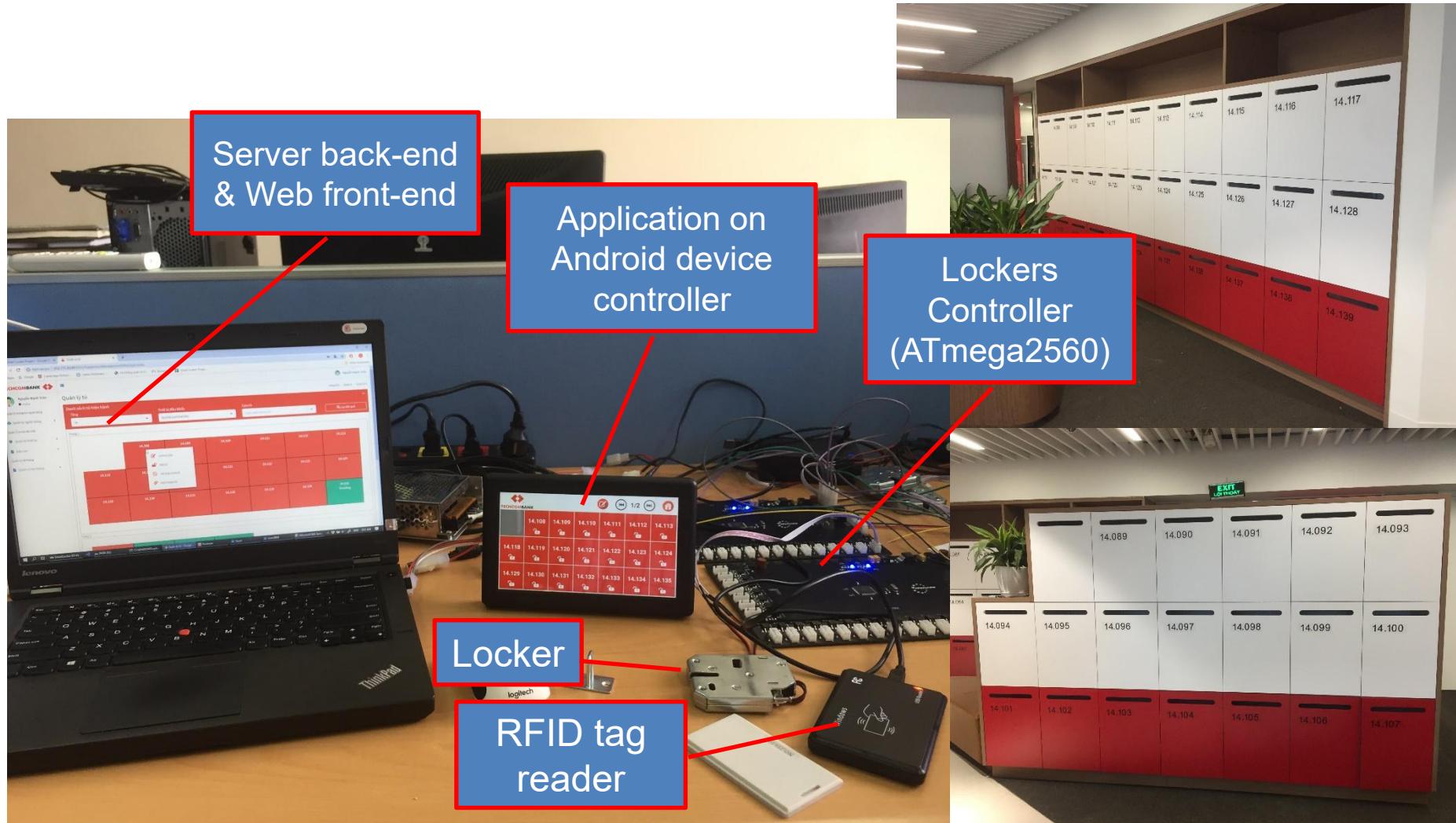
```
pip install requests
```

- Lưu file, ví dụ `send_to_thingspeak.py`, và chạy:

```
bash
```

```
python send_to_thingspeak.py
```

Ví dụ 2: Hệ thống Smart Locker giao tiếp dữ liệu sử dụng http



Hệ thống Smart Locker

Ví dụ 2: Hệ thống Smart Locker giao tiếp dữ liệu sử dụng http (1)

- Sử dụng thư viện HttpURLConnection trên Java thực hiện một http request tới http REST api
- **Bước 1:** Tạo một kết nối http connection, thiết lập các thuộc tính cho request

```
public String requestUserInfor(String devId, String qrDevId, String qrCode, int
timeout) {
    String result = "";
    try {
        String urlApi =
"https://203.171.20.94:8080/api/AccessControl/GetUserInfor";
        URL url = new URL(urlApi);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("POST");
        conn.setConnectTimeout(timeout);
        conn.setReadTimeout(timeout);
        conn.setRequestProperty("Content-Type", "application/json; charset=utf-8");
```

Ví dụ 2: Hệ thống Smart Locker giao tiếp dữ liệu sử dụng http (2)

- **Bước 2:** Ghi dữ liệu cần gửi vào luồng output stream của request. Dữ liệu có thể được đóng gói bằng JSON

```
...
OutputStream os = conn.getOutputStream();
BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os, "UTF-8"));
JSONObject jsonObj = new JSONObject();
try {
    jsonObj.accumulate("deviceId", devId);
    jsonObj.accumulate("qrCodeId", qrDevId);
    jsonObj.accumulate("qrCodeValue", qrCode);
} catch (JSONException e) {
    e.printStackTrace();
}

writer.write(jsonObj.toString());
writer.flush();
writer.close();
os.close();
conn.connect();
```

Ví dụ 2: Hệ thống Smart Locker giao tiếp dữ liệu sử dụng http (3)

- **Bước 3:** Nhận phản hồi http response từ server. Kiểm tra mã trả về và đọc dữ liệu từ luồng input stream của http response

```
...
StringBuffer sb = new StringBuffer();
if (conn.getResponseCode() == HttpURLConnection.HTTP_OK) {
    InputStream in = conn.getInputStream();
    int chr;
    while ((chr = in.read()) != -1) {
        sb.append((char) chr);
    }
    in.close();
} else {
    sb.append(conn.getResponseCode());
}
result = sb.toString();
conn.disconnect();
}
catch (IOException e) {
    e.printStackTrace();
}
return result;
}
```

Sử dụng công cụ Postman

The screenshot shows the Postman application window. At the top, there is a navigation bar with 'File', 'Edit', 'View', 'Help', 'Home', 'Workspaces', 'Reports', 'Explore', a search bar 'Search Postman', and user account options ('Sign In' and 'Create Account'). A yellow banner at the top center says 'Working locally in Scratch Pad. Switch to a Workspace'. Below the banner, the main workspace displays a list of recent requests and a single active request for 'https://203.171.20.94:8080/api/AccessControl/GetUserInfor' via POST.

Request Details:

- Method: POST
- URL: https://203.171.20.94:8080/api/AccessControl/GetUserInfor
- Buttons: Save, Edit, Send

Params Tab (Active):

- Sub-tabs: Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings, Cookies
- Section: Query Params
- Table:

KEY	VALUE	DESCRIPTION	ooo	Bulk Edit
Key	Value	Description		

Response Tab:

Find and Replace, Console, Runner, Trash

Http Response code

- 200: OK
- 201: Created
- 204: No Content
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 408: Request Timeout
- 500: Internal Server Error
- 502: Bad Gateway
- 503: Service Unavailable

Bài tập 1. Trao đổi dữ liệu qua HTTP

Yêu cầu. Viết chương trình (bằng ngôn ngữ tùy ý: C#, Java, python) thực hiện

a) Gửi dữ liệu gồm 2 trường field1, field2 lên Thinkspeak qua API theo 2 cách:

Cách 1. Các trường field1, field2 được đóng gói trong url (urlencoded)

```
GET https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5&field1=20&field2  
=33
```

Cách 2. Các trường field1, field2 được đóng gói trong body request bằng json.

```
POST https://api.thingspeak.com/update?api_key=T7H40F0X82VGW7L5
```

Ví dụ: body request.

```
{  
    "field1": 20,  
    "field2": 33  
}
```

Bài tập 1. Trao đổi dữ liệu qua HTTP

Yêu cầu. Viết chương trình (bằng ngôn ngữ tùy ý: C#, Java, python) thực hiện

b) Lấy dữ liệu về từ Thingspeak API

GET <https://api.thingspeak.com/channels/1529099/feeds.json?results=2>

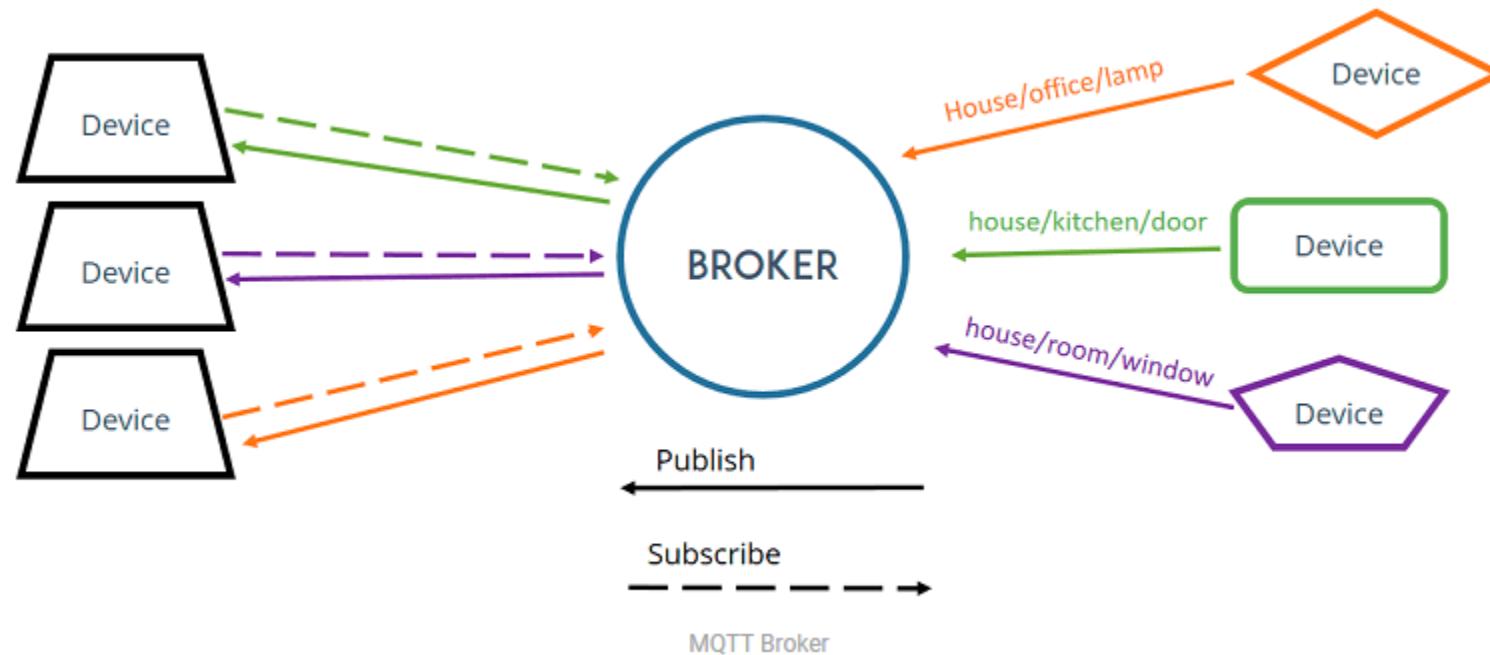
Parsing dữ liệu gửi về để lấy ra 2 trường field1 (temperature) và field2 (humidity) và hiển thị ra màn hình.

2.3.2. Trao đổi dữ liệu dùng MQTT

- The Messaging and Data Exchange Protocol of the IoT
- MQTT (Message Queuing Telemetry Transport):
 - Giao thức truyền thông điệp (message) theo mô hình publish/subscribe (xuất bản – theo dõi)
 - Sử dụng băng thông thấp, độ tin cậy cao và có khả năng hoạt động trong điều kiện đường truyền không ổn định.

Giao thức MQTT

- Kiến trúc mức cao của MQTT gồm 2 thành phần chính:
 - Broker
 - Clients

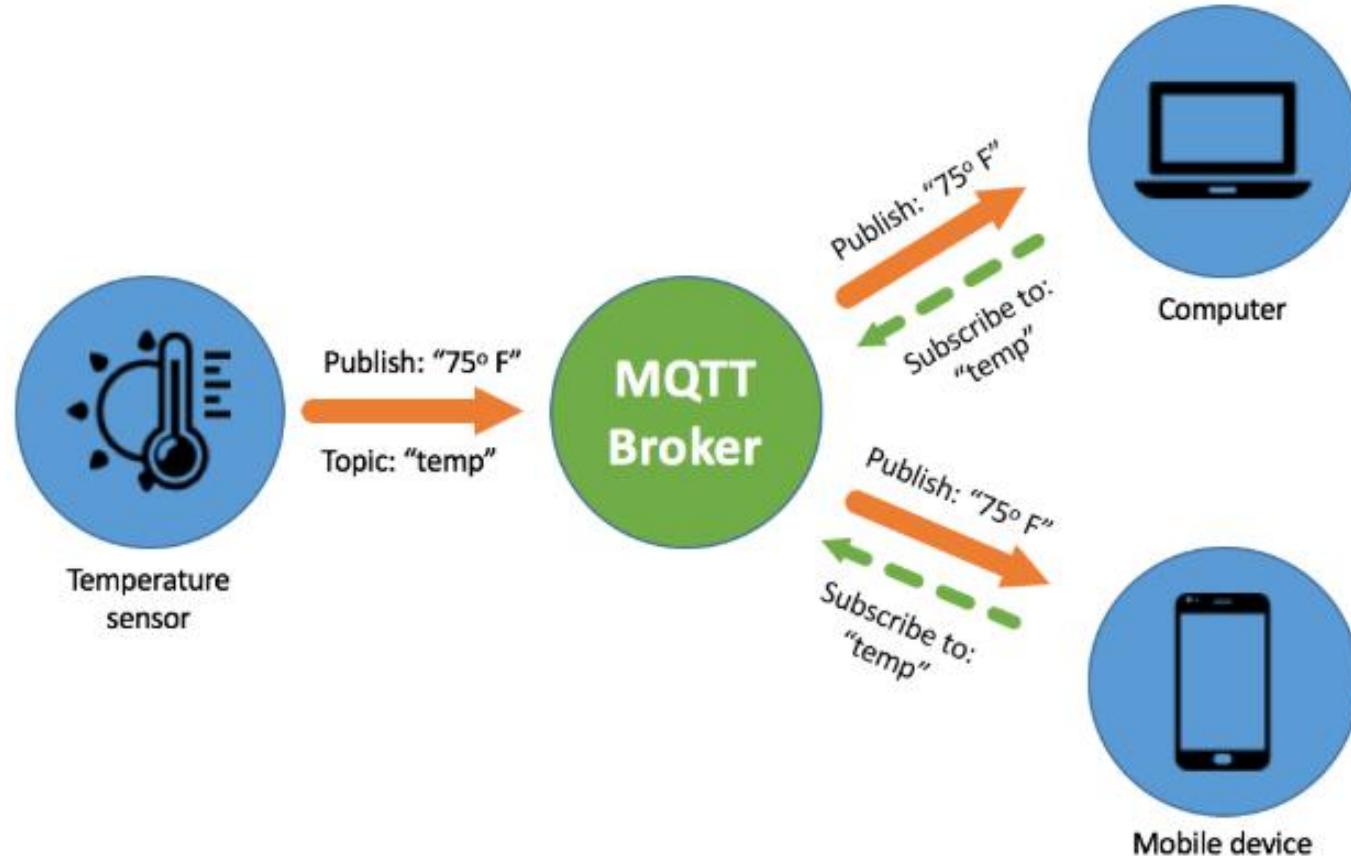


Giao thức MQTT

■ Publish/Subscribe:

- Trong một hệ thống sử dụng giao thức MQTT, nhiều node trạm (gọi là mqtt client – gọi tắt là client) kết nối tới một MQTT server (gọi là broker).
- Mỗi client sẽ đăng ký một vài kênh (topic), ví dụ như “/client1/channel1”, “/client1/channel2”. Quá trình đăng ký này gọi là “subscribe”. Mỗi client sẽ nhận được dữ liệu khi bất kỳ trạm nào khác gửi dữ liệu vào kênh đã đăng ký.
- Khi một client gửi dữ liệu tới kênh đó, gọi là “publish”.

Giao thức MQTT



<https://www.hivemq.com/mqtt-essentials/>

Giao thức MQTT

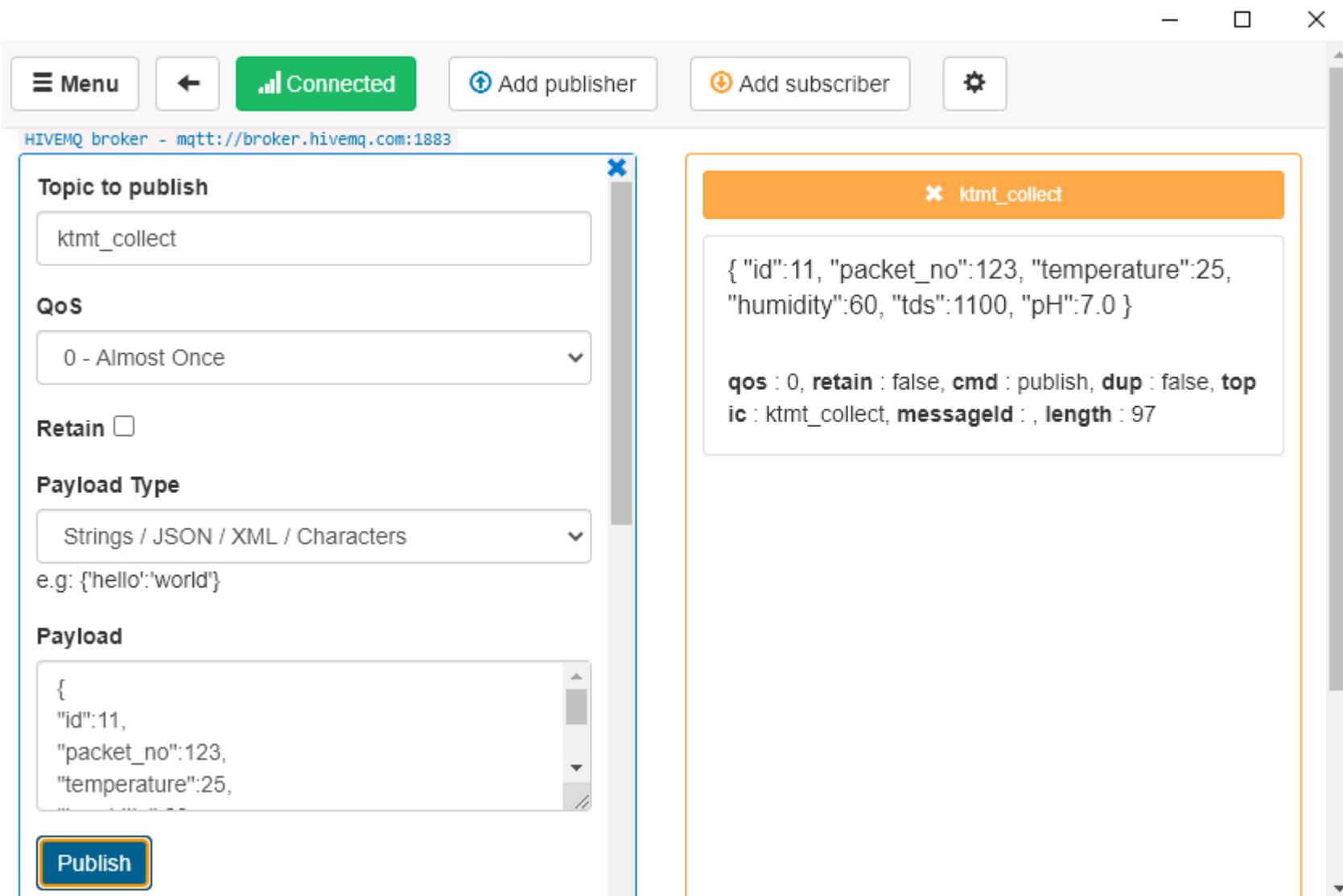
- **QoS (Quality of Service):**

- Có 3 tùy chọn QoS khi “publish” và “subscribe”:
- **QoS0** Broker/client sẽ gửi dữ liệu đúng 1 lần, quá trình gửi được xác nhận bởi chỉ giao thức TCP/IP, (“fire and forget”).
- **QoS1** Broker/client sẽ gửi dữ liệu với ít nhất 1 lần xác nhận từ đầu kia, nghĩa là có thể có nhiều hơn 1 lần xác nhận đã nhận được dữ liệu.
- **QoS2** Broker/client đảm bảo khi gửi dữ liệu thì phía nhận chỉ nhận được đúng 1 lần, quá trình này phải trải qua 4 bước bắt tay

Giao thức MQTT

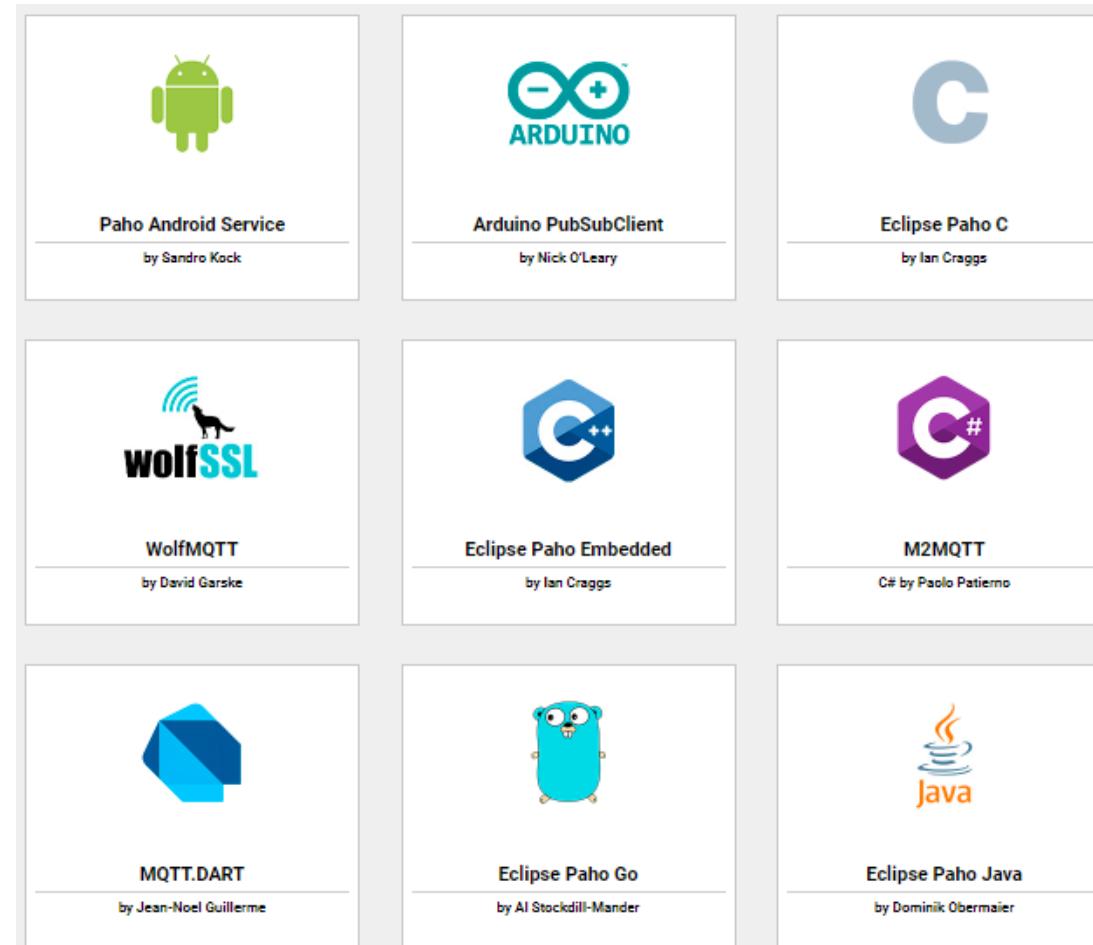
- Sử dụng công cụ client: MQTTBox
 - <https://www.hivemq.com/mqtt-toolbox/>
 - Publish/Subscribe (gửi/nhận) dữ liệu

Công cụ mqtt client: MQTTBox



Thư viện MQTT client

- Thư viện mqtt client cho các nền tảng khác nhau:
 - <https://www.hivemq.com/mqtt-client-library-encyclopedia/>



Ví dụ:

- Chương trình Java thực hiện publish/subscribe (gửi/nhận) dữ liệu sử dụng thư viện Paho MQTT Client
- Thư viện Paho Mqtt Client
 - <https://eclipse.dev/paho/downloads/>
- Using the Paho Java Client:
 - Using Maven dependency
 - Hoặc add thư viện .jar (Build Path >> Add External Archives):
Download file thư viện: org.eclipse.paho.client.mqttv3-1.1.1.jar

Chương trình MQTT Example

```
package mqttDemo;
import org.eclipse.paho.client.mqttv3.*;

public class MqttExample {
    private static final String BROKER = "tcp://broker.emqx.io:1883";
    private static final String TOPIC = "test/topic";
private static final String CLIENT_ID = "JavaClientExample";

    public static void main(String[] args) {

        try {
            MqttClient client = new MqttClient(BROKER, CLIENT_ID, null);

            // Tạo options
            MqttConnectOptions options = new MqttConnectOptions();
            options.setCleanSession(true);
```

Chương trình MQTT Example

```
client.setCallback(new MqttCallback() {
    @Override
    public void connectionLost(Throwable cause) {
        System.out.println("Kết nối bị mất: " + cause.getMessage());
    }

    @Override
    public void messageArrived(String topic, MqttMessage message) throws
Exception {
        System.out.println("Nhận tin nhắn từ topic '" + topic + "'": " + new
String(message.getPayload()));
    }

    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {
        System.out.println("Gửi thành công!");
    }
});
```

Chương trình MQTT Example

```
// Kết nối đến broker
client.connect(options);
System.out.println("Đã kết nối đến broker: " + BROKER);

// Đăng ký nhận tin từ topic
client.subscribe(TOPIC);
System.out.println("Đăng ký topic: " + TOPIC);

// Gửi dữ liệu đến topic
String content = "Xin chào từ Java MQTT!";
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(1);
client.publish(TOPIC, message);

} catch (MqttException e) {
    e.printStackTrace();
}
}
```

Chương trình MQTT Example

1. Import thư viện MQTT

java

 Copy

 Edit

```
import org.eclipse.paho.client.mqttv3.*;
```

Nhập các lớp cần thiết từ thư viện Eclipse Paho – thư viện phổ biến hỗ trợ giao tiếp MQTT trong Java.

Chương trình MQTT Example

✓ 2. Khai báo thông tin MQTT

java

 Copy

```
private static final String BROKER = "tcp://broker.hivemq.com:1883";
private static final String TOPIC = "test/topic";
private static final String CLIENT_ID = "JavaClientExample";
```

- **BROKER** : địa chỉ của MQTT Broker (ở đây dùng broker công cộng miễn phí của HiveMQ).
- **TOPIC** : tên chủ đề (channel) để gửi/nhận dữ liệu.
- **CLIENT_ID** : định danh của client kết nối (mỗi client nên có một ID duy nhất).

Chương trình MQTT Example

✓ 3. Tạo client và thiết lập kết nối

java

 Copy

```
MqttClient client = new MqttClient(BROKER, CLIENT_ID, null);
MqttConnectOptions options = new MqttConnectOptions();
options.setCleanSession(true);
```

- Tạo một client kết nối đến broker.
- `setCleanSession(true)` nghĩa là không lưu lại trạng thái cũ (subscription, tin cũ).

Chương trình MQTT Example

✓ 4. Gắn callback để xử lý dữ liệu nhận được

java

```
client.setCallback(new MqttCallback() {  
    ...  
});
```

- Dùng để xử lý các sự kiện từ MQTT:
 - `connectionLost` : khi mất kết nối với broker.
 - `messageArrived` : khi có tin nhắn mới đến từ topic đã đăng ký.
 - `deliveryComplete` : khi gửi tin thành công.

Chương trình MQTT Example

✓ 5. Kết nối đến MQTT broker

java

```
client.connect(options);
System.out.println("Đã kết nối đến broker: " + BROKER);
```

Kết nối client với MQTT server (broker).

Chương trình MQTT Example

✓ 6. Đăng ký (subscribe) topic

java

 Copy

 Edit

```
client.subscribe(TOPIC);
System.out.println("Đăng ký topic: " + TOPIC);
```

Đăng ký để nhận tin từ topic. Khi có thiết bị nào khác gửi tin đến topic này, client sẽ nhận được.

Chương trình MQTT Example

✓ 7. Gửi tin nhắn (publish)

java

```
String content = "Xin chào từ Java MQTT!";
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(1);
client.publish(TOPIC, message);
```

- Tạo nội dung cần gửi (`MqttMessage`).
- `Qos(1)` là mức đảm bảo: tin được gửi ít nhất 1 lần.
- Gửi tin đến topic đã chọn.

Chương trình MQTT Example – Hàm callback()

❖ Cú pháp tổng quát:

java

Copy

Edit

```
client.setCallback(new MqttCallback() {
    @Override
    public void connectionLost(Throwable cause) { ... }

    @Override
    public void messageArrived(String topic, MqttMessage message) throws Exception { ... }

    @Override
    public void deliveryComplete(IMqttDeliveryToken token) { ... }
});
```

MqttCallback là một interface, bạn cần override 3 phương thức chính sau:

Chương trình MQTT Example – Hàm callback()

1. connectionLost(Throwable cause)

► Khi nào được gọi?

Khi mất kết nối với MQTT broker.

► Thường dùng để làm gì?

- Ghi log lỗi.
- Tự động **reconnect** nếu muốn duy trì kết nối lâu dài.

► Ví dụ:

java

 Copy

```
@Override  
public void connectionLost(Throwable cause) {  
    System.out.println("Kết nối bị mất: " + cause.getMessage());  
}
```

Chương trình MQTT Example – Hàm callback()

2.  `messageArrived(String topic, MqttMessage message)`

➤ Khi nào được gọi?

Khi client nhận được **tin nhắn** từ một topic đã subscribe.

➤ Các tham số:

- `topic` : tên topic chứa tin nhắn.
- `message` : đối tượng chứa nội dung tin nhắn (kiểu `MqttMessage`).

➤ Thường dùng để làm gì?

- Hiển thị/ghi log nội dung tin.
- Xử lý dữ liệu nhận được (ví dụ: cập nhật GUI, lưu DB, điều khiển thiết bị...).

Chương trình MQTT Example – Hàm callback()

2. messageArrived(String topic, MqttMessage message)

► Ví dụ:

java

Copy

Edit

```
@Override  
public void messageArrived(String topic, MqttMessage message) throws Exception {  
    System.out.println("Nhận tin nhắn từ topic '" + topic + "': " + new String(message.getPayload()));  
}
```

+ Ghi chú thêm:

- `message.getPayload()` trả về mảng byte, cần chuyển thành chuỗi nếu bạn gửi dữ liệu dạng văn bản (`String`).
- Nếu bạn gửi JSON, có thể parse bằng thư viện như `Gson` hoặc `Jackson`.

Chương trình MQTT Example – Hàm callback()

3. deliveryComplete(IMqttDeliveryToken token)

► Khi nào được gọi?

Khi gửi tin nhắn thành công (chỉ áp dụng nếu bạn publish tin).

► Thường dùng để làm gì?

- Ghi log xác nhận đã gửi.
- Hiển thị trạng thái gửi trên giao diện.

► Ví dụ:

java

```
@Override  
public void deliveryComplete(IMqttDeliveryToken token) {  
    System.out.println("Gửi thành công!");  
}
```

Chương trình MQTT Example – Dữ liệu json

- Cải tiến: Gửi dữ liệu json

```
// Gửi thử một message JSON lên topic_out
JSONObject data = new JSONObject();
data.put("device", "sensor-1");
data.put("value", 42);

MqttMessage msg = new MqttMessage(data.toString().getBytes());
msg.setQos(1);
client.publish(TOPIC_OUT, msg);
System.out.println("📤 Đã gửi JSON lên topic: " + TOPIC_OUT);
```

- 🔍 Mẫu JSON gửi/nhận:

```
json

{
    "device": "sensor-1",
    "value": 42
}
```

Chương trình MQTT Example – Dữ liệu json

- Cải tiến: Nhận dữ liệu json và bóc tách dữ liệu

```
@Override  
public void messageArrived(String topic, MqttMessage message) {  
    System.out.println("✉ Nhận từ [" + topic + "]: " + message);  
  
    try {  
        // Parse nội dung JSON  
        JSONObject json = new JSONObject(new String(message.getPayload()));  
        String device = json.getString("device");  
        int value = json.getInt("value");  
  
        System.out.println("↳ Thiết bị: " + device + ", Giá trị: " + value);  
    } catch (Exception e) {  
        System.out.println("✖ Lỗi parse JSON: " + e.getMessage());  
    }  
}
```

Chương trình MQTT Example – Dữ liệu json

- Sử dụng thư viện java json

Cách 1: Tải thủ công file .jar và thêm vào Eclipse

Bước 1: Tải file .jar

Truy cập trang chính thức:  <https://mvnrepository.com/artifact/org.json/json/20210307>

→ Bấm vào Download (jar) để tải file .jar.

Bước 2: Thêm .jar vào project Eclipse

- Chuột phải vào project → Properties
- Vào Java Build Path → Tab Libraries
- Nhấn Add External JARs...
- Chọn file json-20210307.jar vừa tải → Apply & Close

Chương trình MQTT Example - Reconnect

- Tự động kết nối lại

```
MqttClient client = new MqttClient(BROKER, CLIENT_ID, null);
MqttConnectOptions options = new MqttConnectOptions();
options.setAutomaticReconnect(true);          // Tự động reconnect
options.setCleanSession(true);                // Không lưu lại phiên cũ
options.setConnectionTimeout(10);             // Timeout 10 giây

// Gắn callback để xử lý dữ liệu nhận
client.setCallback(new MqttCallback() {
    public void connectionLost(Throwable cause) {
        System.out.println("Kết nối bị mất");
    }

    public void messageArrived(String topic, MqttMessage message) {
        System.out.println("Nhận tin nhắn: " + message.getPayload());
    }
});

// Kết nối đến broker
client.connect(options);
System.out.println("Đã kết nối đến broker: " + BROKER);
```

Bài tập 2

- Viết một chương trình mqtt client thực hiện:

- Gửi (publish) dữ liệu lên mqtt broker
- Nhận (subscribe) dữ liệu từ mqtt broker
- Đóng gói dữ liệu bằng JSON. Ví dụ:

```
{  
    "DeviceName": "my-mqtt-client",  
    "temperature":30,  
    "humidity":60  
}
```

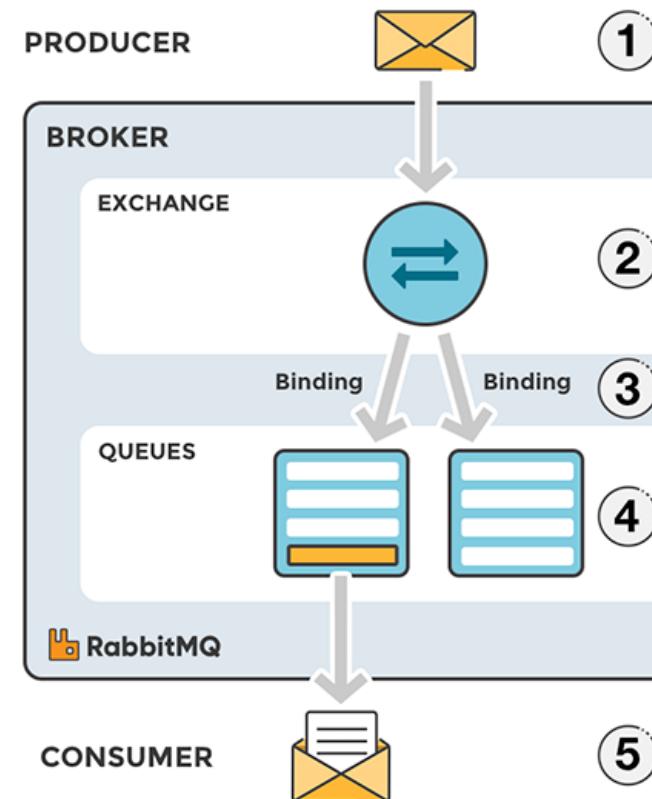
- Dùng MQTTBox để minh họa gửi nhận
- Ví dụ dùng mqtt broker: <tcp://broker.hivemq.com:1883>

2.3.3. Giao thức AMQP

- AMQP (Advanced Message Queue Protocol): Giao thức truyền nhận dùng hàng đợi thông điệp
- RabbitMQ: Một broker sử dụng giao thức AMQP
- RabbitMQ broker đóng vai trò trung gian lưu trữ cũng như điều phối thông điệp (message) giữa bên gửi (producer) và bên nhận (consumer)

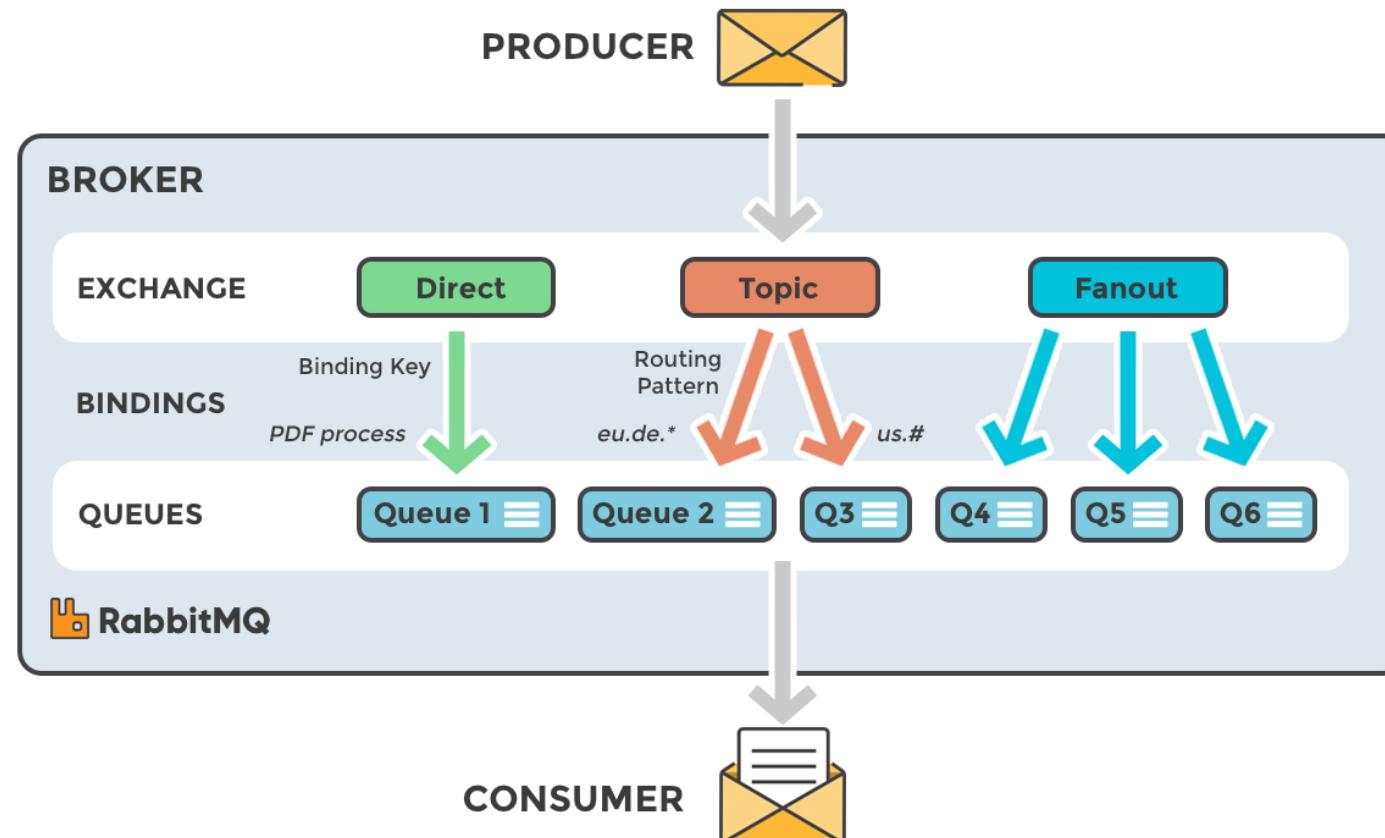
Giao thức AMQP

- RabbitMQ:
 - Luồng gửi nhận message qua RabbitMQ



Giao thức AMQP

- RabbitMQ:
 - 4 loại Exchange: direct, topic, fanout, headers

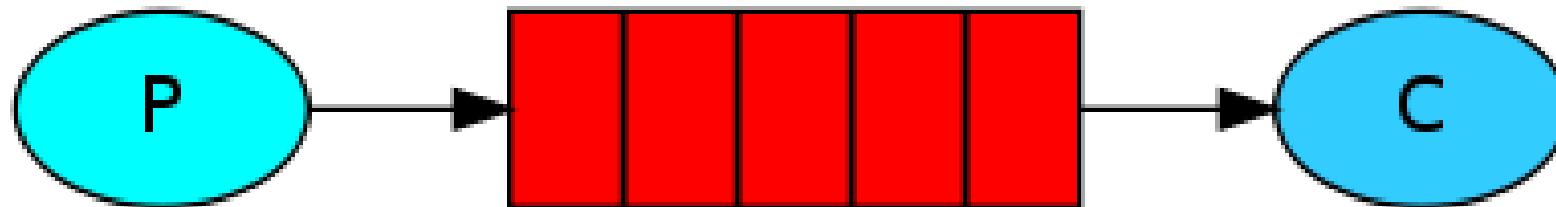


Giao thức AMQP

- Cài đặt RabbitMQ broker (server)
 - <https://www.rabbitmq.com/download.html>
 - <https://www.rabbitmq.com/install-windows.html>
- Quản trị RabbitMQ broker bằng công cụ:
 - <https://www.rabbitmq.com/management.html>
- RabbitMQ tutorials:
 - <https://www.rabbitmq.com/getstarted.html>

Tutorial 1

- Viết chương trình gửi nhận dữ liệu qua RabbitMQ
 - Cài đặt, sử dụng một RabbitMQ broker
 - Chương trình Producer gửi dữ liệu (message)
 - Chương trình Consumer nhận dữ liệu
 - Sử dụng default exchange (type: direct)



Tutorial 1

- Tham khảo:
 - Chương trình java <https://www.rabbitmq.com/tutorials/tutorial-one-java.html>
 - Chương trình python <https://www.rabbitmq.com/tutorials/tutorial-one-python>

Tutorial 1. Producer

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import java.nio.charset.StandardCharsets;

public class send {
    private final static String QUEUE_NAME = "ktmt";
    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setUsername("guest");
        factory.setPassword("guest");
        try (Connection connection = factory.newConnection();
            Channel channel = connection.createChannel()) {
            channel.queueDeclare(QUEUE_NAME, false, false, false, null);
            String message = "Hello World! Publish message to broker";
            channel.basicPublish("", QUEUE_NAME, null,
                message.getBytes(StandardCharsets.UTF_8));
            System.out.println(" [x] Sent '" + message + "'");
        }
    }
}
```

Sender.java

Tutorial 1. Consumer

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DeliverCallback;
public class Receive {
    private final static String QUEUE_NAME = "ktmt";
    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setUsername("guest");
        factory.setPassword("guest");
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        System.out.println(" [*] Waiting for messages. To exit press CTRL+C");
        DeliverCallback deliverCallback = (consumerTag, delivery) -> {
            String message = new String(delivery.getBody(), "UTF-8");
            System.out.println(" [x] Received '" + message + "'");
        };
        channel.basicConsume(QUEUE_NAME, true, deliverCallback, consumerTag-> { });
    }
}
```

Recv.java

Sử dụng CloudAMQP

- Sử dụng RabbitMQ broker trên CouldAMQP (miễn phí)
 - Truy cập trang <https://www.cloudamqp.com/>
 - Đăng ký tài khoản (Sign Up)
 - Tạo instance miễn phí (Create New Instance)
 1. Sau khi đăng nhập, vào trang quản lý và click "Create New Instance".
 2. Điền thông tin:
 - Name: tên bất kỳ (ví dụ: MyRabbitMQ)
 - Cloud provider: chọn AWS
 - Region: chọn US-East-1 hoặc khu vực gần bạn
 - Plan: chọn "Little Lemur - Free plan"
 3. Click "Create instance"

Sử dụng CloudAMQP

- Sau khi tạo xong instance thành công, lấy thông tin truy cập broker

AMQP details

User & Vhost	cohushpn
Password	m_PtS6PB02IfLhaAa7LPDr0yAHZhclF_   Rotate password
Ports	5672 (5671 for TLS)
URL	amqps://cohushpn:m_PtS6PB02IfLhaAa7LPDr0yAHZhclF_@armadillo.rmq.cloudamqp.com/cohushpn 

Sử dụng CloudAMQP

- Thiết lập thông tin kết nối broker trong chương trình (Java)

```
ConnectionFactory factory = new ConnectionFactory();
//Set CloudAMQP credentials
factory.setUsername("cohushpn");
factory.setPassword("m_PtS6PB02IfLhaAa7LPDr0yAHZhcIF_");
factory.setVirtualHost("cohushpn");
factory.setHost("armadillo.rmq.cloudamqp.com");
factory.setPort(5671); //AMQPs
factory.useSslProtocol();
```

Producer.java

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

public class Producer {
    private static final String QUEUE_NAME = "testQueue";

    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        //Thông tin từ CloudAMQP
        factory.setHost("armadillo.rmq.cloudamqp.com");
        factory.setUsername("cohushpn");
        factory.setPassword("m_PtS6PB02IfLhaAa7LPDr0yAHZhcIF_");
        factory.setVirtualHost("cohushpn");
        factory.setPort(5671); // Sử dụng AMQPS
        factory.useSslProtocol(); // SSL an toàn
        try (Connection connection = factory.newConnection()) {
            Channel channel = connection.createChannel();
            channel.queueDeclare(QUEUE_NAME, true, false, false, null);
            String message = "Xin chào từ Producer!";
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes("UTF-8"));
            System.out.println("[x] Đã gửi: '" + message + "'");
        }
    }
}
```

```
import com.rabbitmq.client.*;
public class Consumer {
    private static final String QUEUE_NAME = "testQueue";
    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        // Thông tin từ CloudAMQP
        factory.setHost("armadillo.rmq.cloudamqp.com");
        factory.setUsername("cohushpn");
        factory.setPassword("m_PtS6PB02IfLhaAa7LPDr0yAHZhcIF_");
        factory.setVirtualHost("cohushpn");
        factory.setPort(5671); // Sử dụng AMQPS
        factory.useSslProtocol(); // SSL an toàn
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);
        System.out.println(" [*] Đang chờ tin nhắn. Nhấn Ctrl+C để thoát.");
        DeliverCallback deliverCallback = (consumerTag, delivery) -> {
            String message = new String(delivery.getBody(), "UTF-8");
            System.out.println(" [x] Nhận được: '" + message + "'");
        };
        channel.basicConsume(QUEUE_NAME, true, deliverCallback, consumerTag -> {});
    }
}
```

Consumer.java

Các bước chính trong Producer

```
// 1. Tạo ConnectionFactory để cấu hình kết nối đến RabbitMQ  
ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("armadillo.rmq.cloudamqp.com");  
factory.setUsername("...");  
factory.setPassword("...");  
factory.setVirtualHost("...");  
factory.setPort(5671);  
factory.useSslProtocol(); // Kết nối an toàn bằng SSL
```

- Mục đích: cấu hình kết nối với RabbitMQ (qua internet), dùng AMQPS (port 5671).

Các bước chính trong Producer

// 2. Mở kết nối và tạo channel

```
Connection connection = factory.newConnection();
```

```
Channel channel = connection.createChannel();
```

- Mục đích: tạo kết nối TCP tới RabbitMQ và mở kênh truyền (channel) – giống như socket ảo.

Các bước chính trong Producer

```
// 3. Khai báo queue (nếu chưa có thì tạo)  
channel.queueDeclare("testQueue", true, false, false, null);
```

 Mục đích: đảm bảo queue tồn tại trước khi gửi.

- “`true` = queue bền vững (persistent).”

Các bước chính trong Producer

// 4. Gửi thông điệp

```
String message = "Xin chào từ Producer!";
channel.basicPublish("", "testQueue", null, message.getBytes("UTF-8"));
```

 Mục đích: gửi thông điệp vào queue `testQueue`.

- “`""` là exchange mặc định (direct).”
- “`basicPublish()` là hàm gửi.”

// 5. Đóng kênh và kết nối

```
channel.close();
connection.close();
```

 Mục đích: đóng tài nguyên sau khi gửi xong.

Các bước chính trong Consumer

// 1. Cấu hình kết nối giống Producer

```
ConnectionFactory factory = new ConnectionFactory();
... // (giống như bên Producer)
```

// 2. Mở kết nối và channel

```
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
```

// 3. Khai báo queue

```
channel.queueDeclare("testQueue", true, false, false, null);
```

 **Lưu ý:** Cả Producer và Consumer phải cùng dùng tên queue giống nhau để gửi-nhận.

Các bước chính trong Consumer

```
// 4. Đăng ký callback để nhận tin nhắn khi có
DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Nhận được: '" + message + "'");
};

channel.basicConsume("testQueue", true, deliverCallback, consumerTag -> {});
```

Mục đích:

- Khi có tin nhắn mới trong queue, `DeliverCallback` được gọi.
- `basicConsume()` đăng ký Consumer vào hàng đợi.

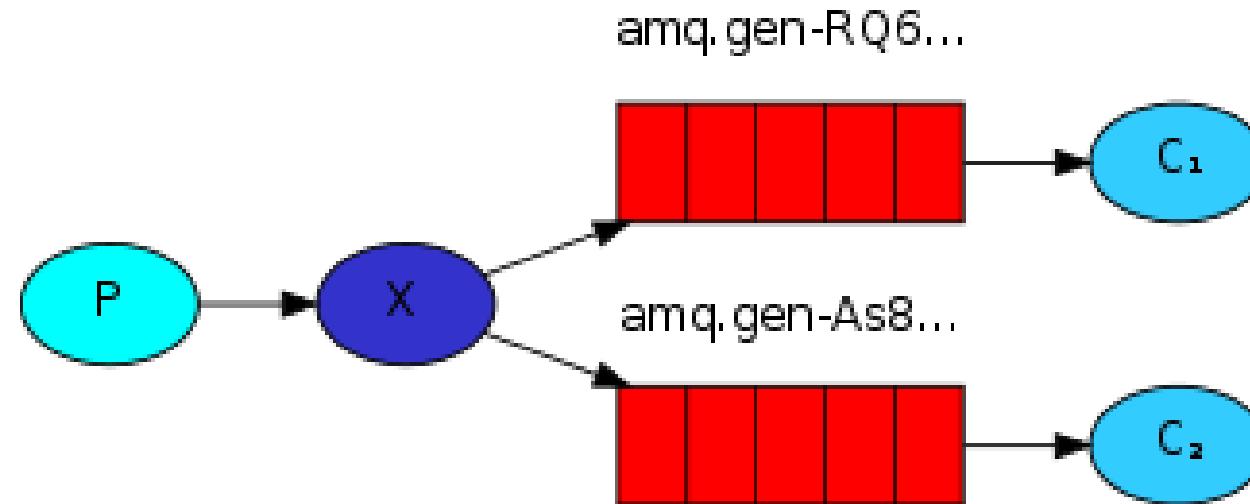
Tóm tắt quy trình Producer - Consumer

Bước	Producer	Consumer
1	Cấu hình RabbitMQ (host, user, vhost, ssl)	Cấu hình giống Producer
2	Tạo connection và channel	Tạo connection và channel
3	Khai báo queue	Khai báo queue
4	Gửi thông điệp vào queue	Đăng ký nhận và xử lý khi có tin
5	Đóng kết nối sau khi gửi xong	Giữ kết nối mở để lắng nghe

Tutorial 3

- Cải tiến Tutorial 1: một Producer gửi message đến tất cả các consumer
 - Sử dụng exchange type = “fanout” (broadcast)

```
channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
```



<https://www.rabbitmq.com/tutorials/tutorial-three-java.html>

Producer gửi broadcast đến tất cả consumer

```
try (Connection connection = factory.newConnection();
    Channel channel = connection.createChannel()) {

    //  Khai báo exchange kiểu fanout
    channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.FANOUT);

    String message = "Broadcast from Producer!";
    channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes("UTF-8"));
    System.out.println(" [x] Đã gửi broadcast: '" + message + "'");

}
```

Consumer bind với queue để nhận

```
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

// NEW Khai báo exchange kiểu fanout
channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.FANOUT);

// NEW Tạo queue tạm thời (tự động tạo tên, tự xóa khi kết thúc)
String queueName = channel.queueDeclare().getQueue();

// NEW Bind queue đó với exchange
channel.queueBind(queueName, EXCHANGE_NAME, "");

System.out.println(" [*] Đang chờ broadcast...");
```

Tutorial 4

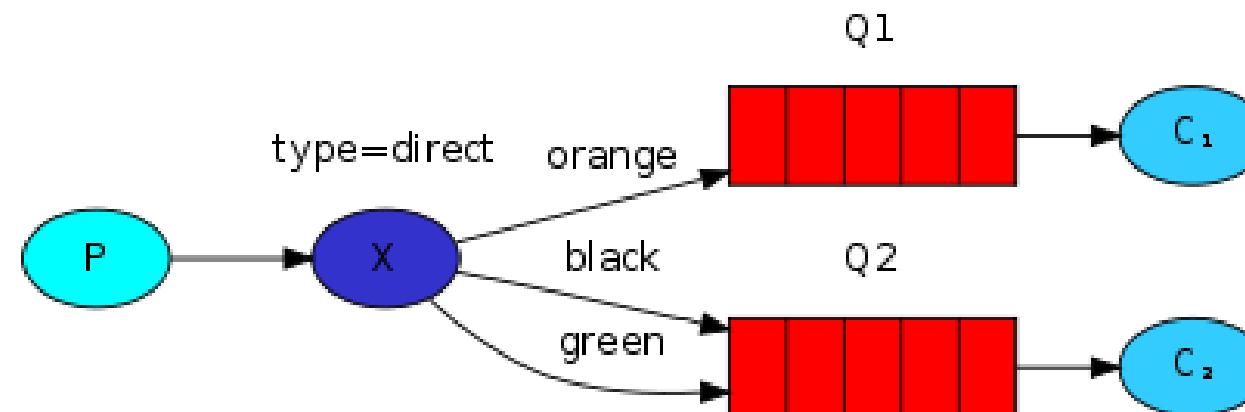
- Routing (định tuyến) thông điệp
- Chọn thông điệp muốn nhận (Receiving messages selectively)

Producer

```
channel.exchangeDeclare(EXCHANGE_NAME, "direct");
channel.basicPublish(EXCHANGE_NAME, "black", null, message.getBytes());
```

Consumer

```
channel.queueBind(queueName, EXCHANGE_NAME, "black");
```



<https://www.rabbitmq.com/tutorials/tutorial-four-java.html>

Tutorial 5

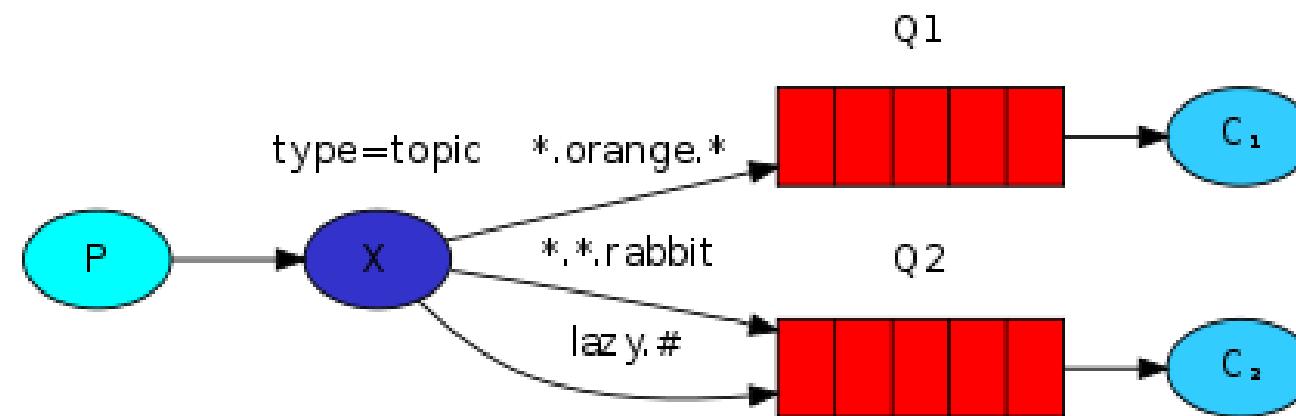
- Sử dụng exchange type = “topic”
- routing_key must be a list of words, delimited by dots.
 - *(star) can substitute for exactly one word.
 - #(hash) can substitute for zero or more words.

Producer

```
channel.basicPublish(EXCHANGE_NAME, routingKey, null, message.getBytes("UTF-8"));
```

Consumer

```
channel.queueBind(queueName, EXCHANGE_NAME, bindingKey);
```



<https://www.rabbitmq.com/tutorials/tutorial-five-java.html>

Bài tập 3

- Viết một chương trình gửi/nhận dữ liệu qua RabbitMQ broker:
 - Gửi (publish) dữ liệu lên RabbitMQ broker (cài đặt hoặc dùng CloudAMQP)
 - Nhận (subscribe) dữ liệu từ RabbitMQ broker
 - Đóng gói dữ liệu bằng JSON. Ví dụ:

```
{  
    "DeviceName": "sender-1",  
    "temperature": 30,  
    "humidity": 60  
}
```

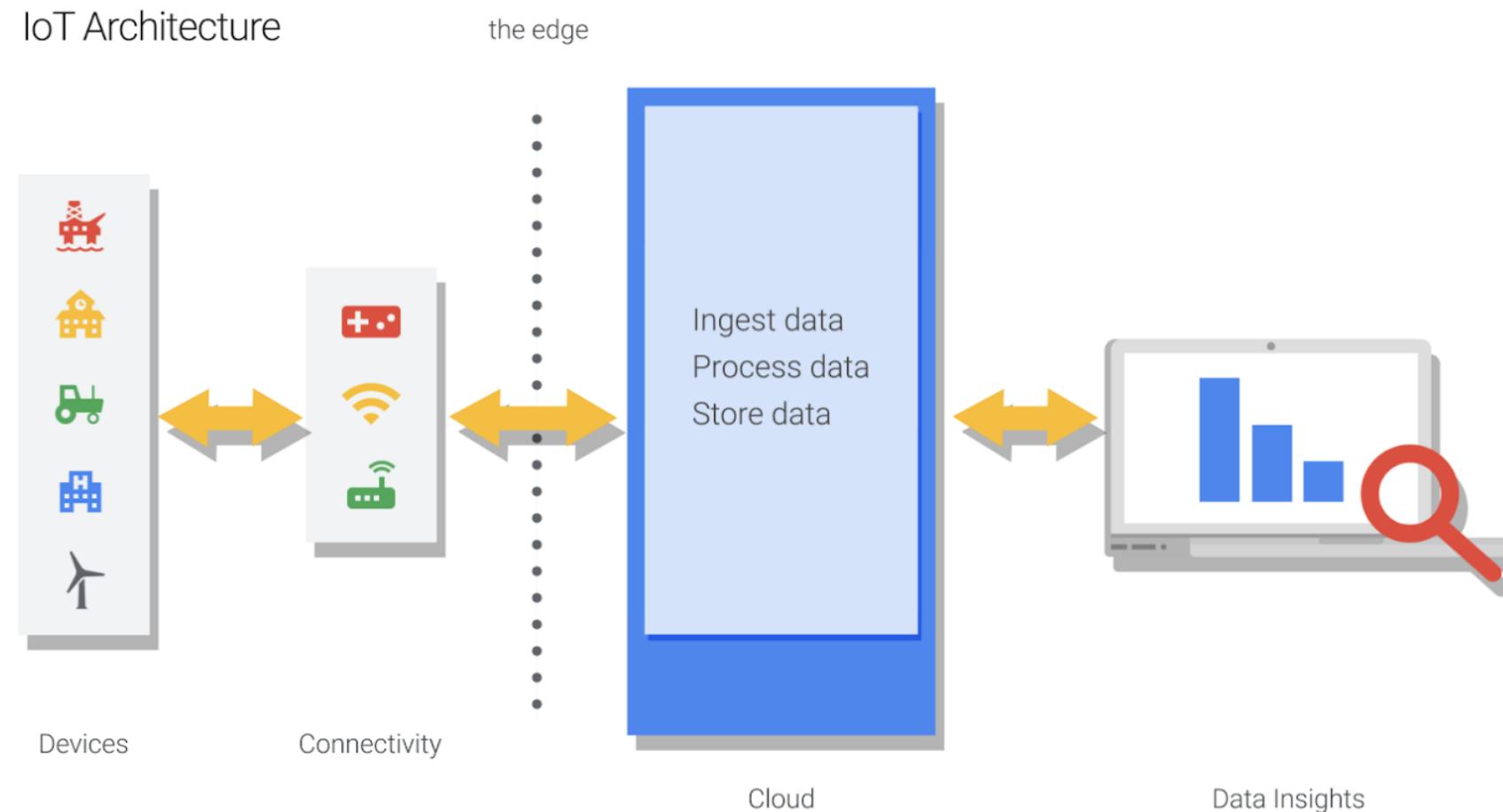
2.4. Nền tảng IoT Cloud Platform

- Các kiến trúc IoT phải có khả năng mở rộng (scaling) qui mô kết nối các thiết bị, thu thập dữ liệu (data gathering), xử lý (data processing) và lưu trữ dữ liệu (data storage)
- Khả năng thực hiện các công việc trên nhanh chóng trong khi vẫn thực hiện phân tích dữ liệu theo thời gian thực (real-time data insights)
- Gửi lượng lớn dữ liệu lên cloud làm chậm quá trình xử lý, đòi hỏi thêm băng thông cho truyền nhận dữ liệu

IoT Cloud Platform

- Giải pháp tính toán phân tán (distributed computing) ~ **fog or edge computing** trở nên phổ biến
- Edge computing (tính toán cận biên): việc xử lý tính toán thực hiện trên các nodes trong mạng là các thiết bị IoT (IoT devices) nằm ở biên “edge” của mạng
- Giải pháp dẫn đến nhu cầu cho thiết bị IoT có khả năng: xử lý, phân tích dữ liệu cục bộ (cleaning, processing, analyzing data locally). Chỉ gửi dữ liệu đã xử lý trước đến cloud

IoT Cloud Platform



IoT Cloud Platforms

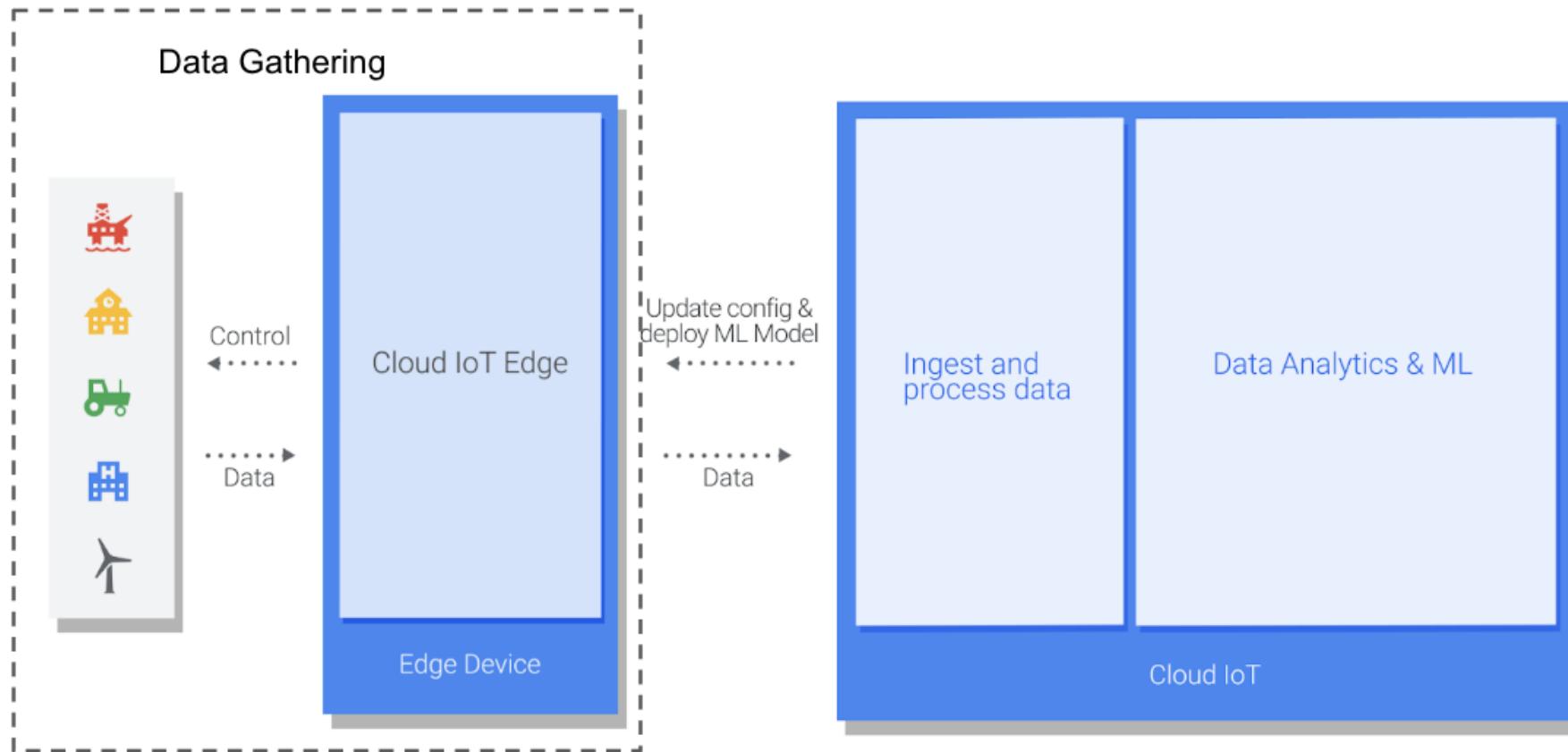
- Một số nền tảng dịch vụ IoT Cloud:
 - ThingsBoard
 - AWS IoT
 - Google Cloud IoT
 - Microsoft Azure IoT
 - IBM Watson IoT



<https://www.postscapes.com/internet-of-things-technologies/>

Google Cloud IoT Architecture

Be capable of doing data import, process, storage, and analysis for hundreds of millions of events per hour from devices all over the world



Google Cloud IoT

- Kiến trúc Google Cloud IoT chia làm 4 thành phần:
 - data gathering (thu thập dữ liệu trên thiết bị)
 - data ingest (tiếp nhận dữ liệu trên Cloud)
 - data processing (xử lý dữ liệu)
 - data analysis (phân tích dữ liệu)

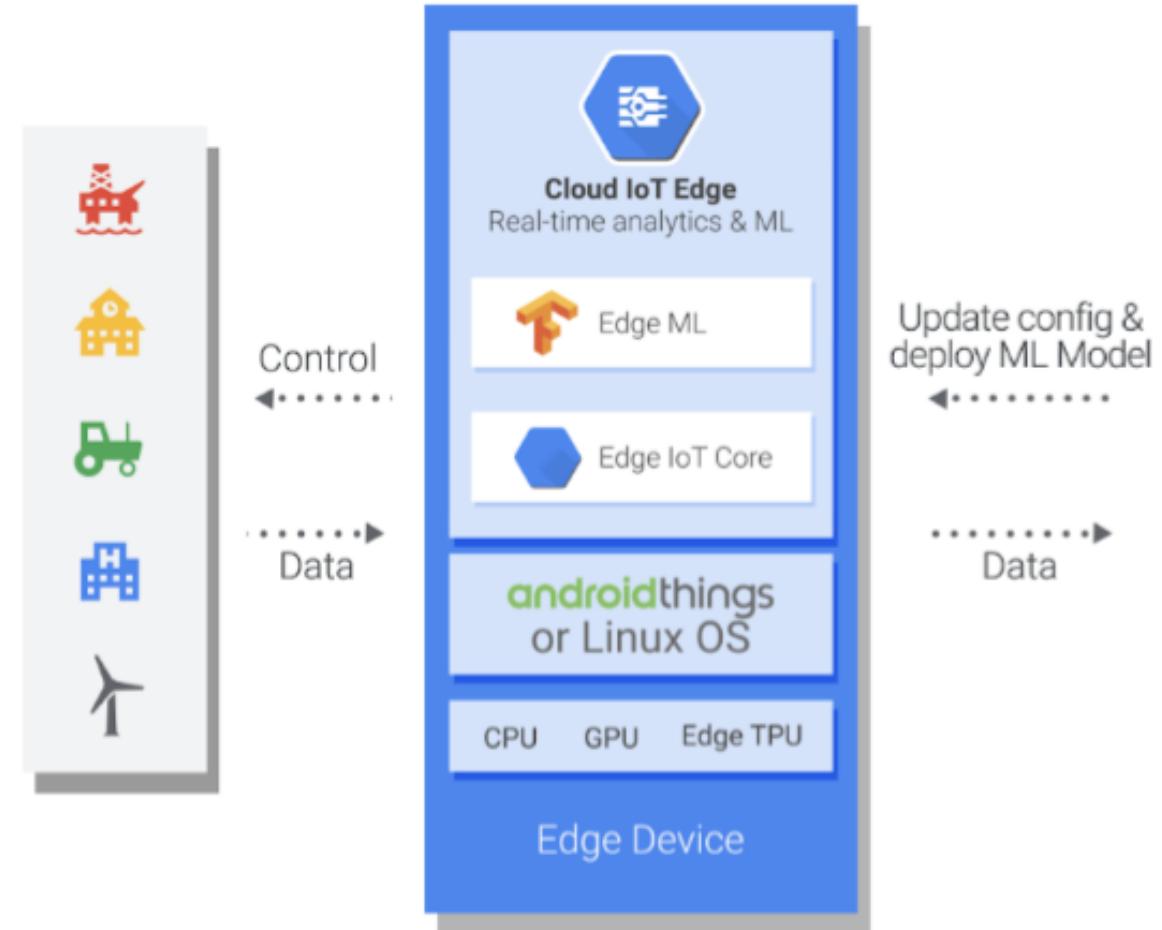
Google Cloud IoT

- Data Gathering (Thu thập dữ liệu trên thiết bị):
 - Xảy ra trên các thiết bị (devices) và cảm biến (sensors)
 - Cảm biến (sensors) thu thập (gather) dữ liệu từ môi trường, gửi lên cloud hoặc trực tiếp hoặc gián tiếp thông qua thiết bị trung gian
 - Thiết bị (devices) chuẩn bị dữ liệu truyền lên cloud. Tùy thuộc loại mạng kết nối, quá trình chuẩn bị bao gồm: làm sạch dữ liệu, tiền xử lý, phân tích hoặc có thể sử dụng cả machine learning



Google Cloud IoT

- Cloud IoT Edge:

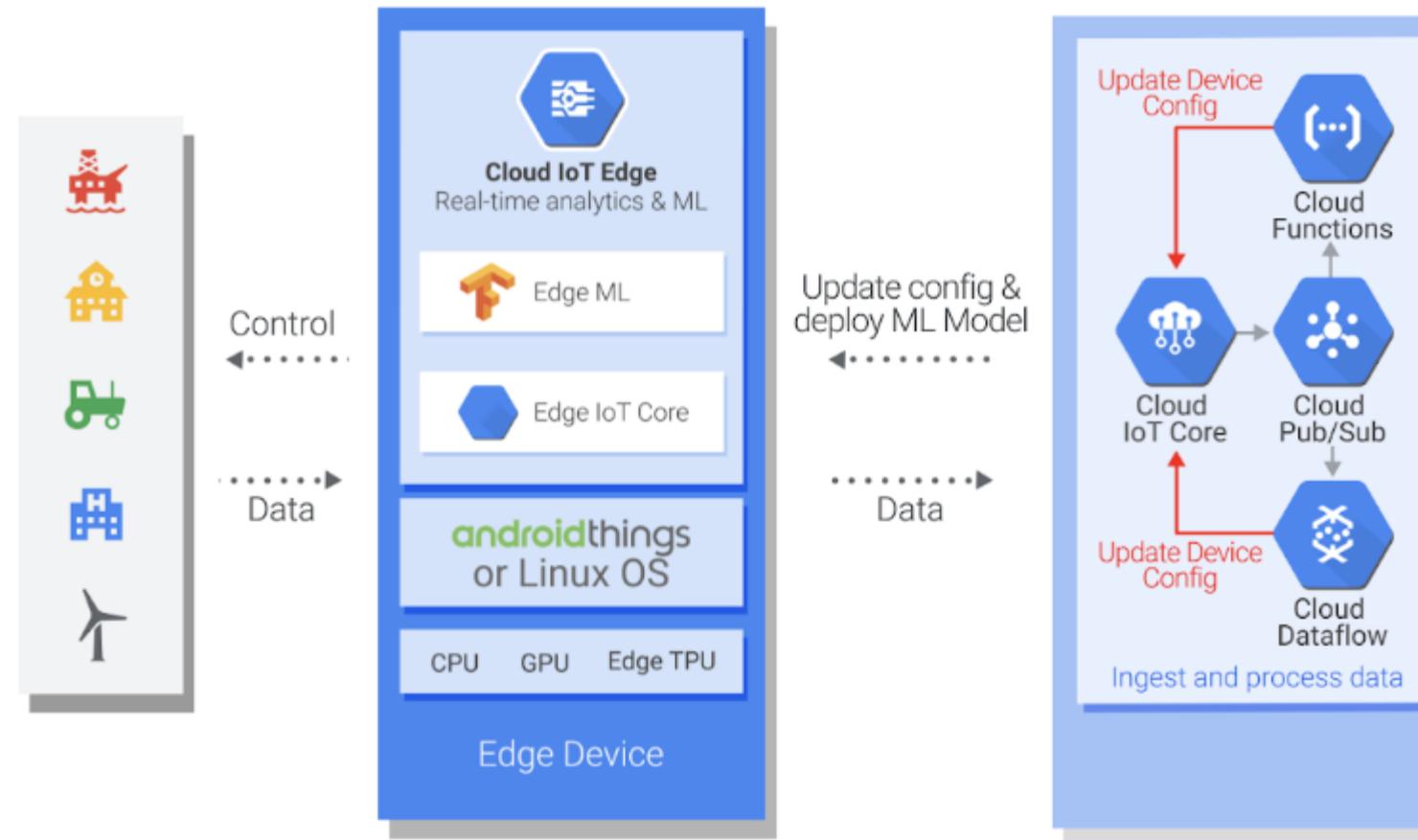


Google Cloud IoT

- Cloud IoT Edge:
 - Trong kiến trúc Goolge Cloud IoT, giai đoạn thu thập dữ liệu có thể thực hiện trên Cloud IoT Edge.
 - Thành phần này là một nhóm các thiết bị có khả năng thực hiện các phân tích thời gian thực và ML (machine learning). Cho phép mở rộng các xử lý dữ liệu và ML thực hiện trên các thiết bị (edge devices).
 - Cloud IoT Edge có thể sử dụng Android Things OS, Linux-based OS.
 - Gồm 2 thành phần: Edge IoT Core và Edge ML

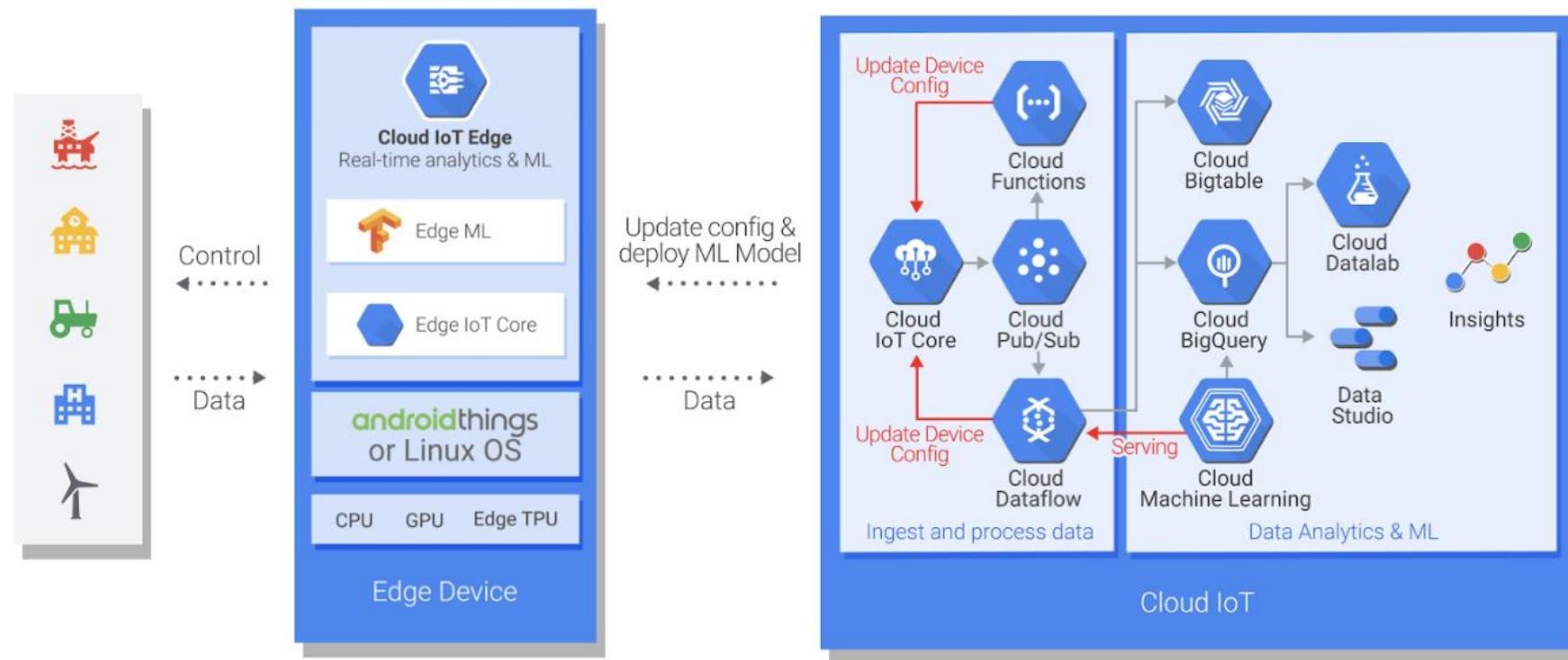
Google Cloud IoT

- Ingest and Process data (Tiếp nhận và xử lý dữ liệu):



Google Cloud IoT

- Data Analytics and ML (Phân tích dữ liệu và Học máy):
 - Có thể thực hiện trên Edge hoặc Cloud.
 - Google's Cloud IoT Core Data Analytics and ML are fully integrated with IoT data.



Nội dung

- Chương 1. Tổng quan về IoT
- Chương 2. Các công nghệ IoT
- Chương 3. Lập trình ứng dụng IoT
- Chương 4. An toàn và Bảo mật IoT
- Chương 5. Thiết kế và xây dựng hệ thống IoT

Chương 3. Lập trình ứng dụng IoT

- 3.1. Lập trình cho thiết bị IoT (ESP32)
- 3.2. Xây dựng Web API (IoT Server)
- 3.3. Khai thác dịch vụ IoT Cloud Platform

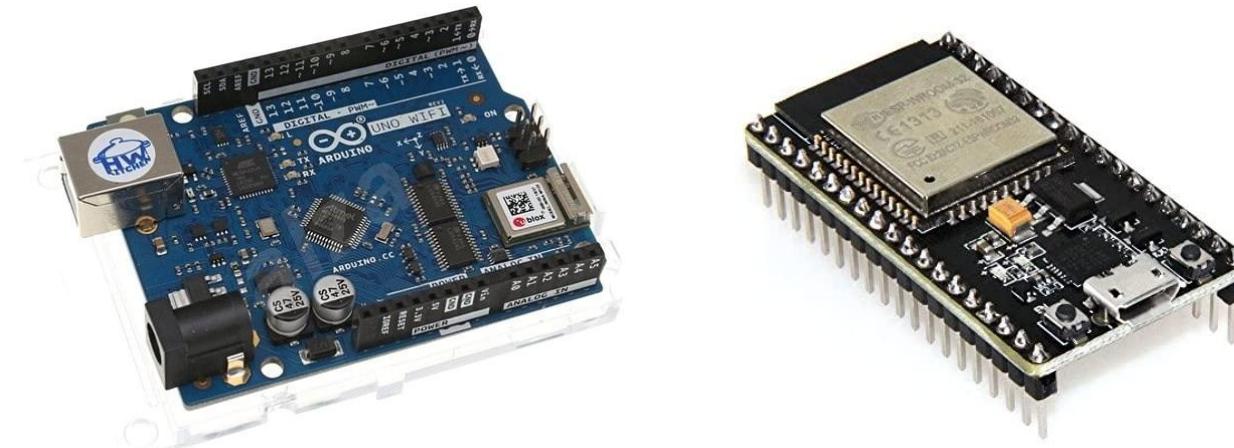
3.1. Lập trình cho thiết bị IoT

- Các thiết bị dùng vi điều khiển hiệu năng thấp
 - Kiến trúc 8 bit, hiệu năng thấp, không dùng hệ điều hành
 - Lập trình firmware, giao tiếp vào ra cơ bản GPIO, ghép nối các ngoại vi (sensor, actuator) theo các chuẩn truyền thông phổ biến UART, SPI, I2C
 - Ví dụ: ATmega, PIC, AVR, Arduino



Lập trình với thiết bị IoT

- Các thiết bị dùng Vi xử lý 32 bit, hiệu năng trung bình
 - Có thể dùng hệ điều hành đơn giản (FreeRTOS, TinyOS, uCLinux, ...)
 - Giao tiếp vào ra cơ bản GPIO, ghép nối các ngoại vi theo các chuẩn truyền thông phổ biến UART, SPI, I2C
 - Có thêm khả năng kết nối Wi-Fi, Bluetooth, GSM module,
 - Ví dụ: ESP8266, ESP32, Arduino Uno WiFi



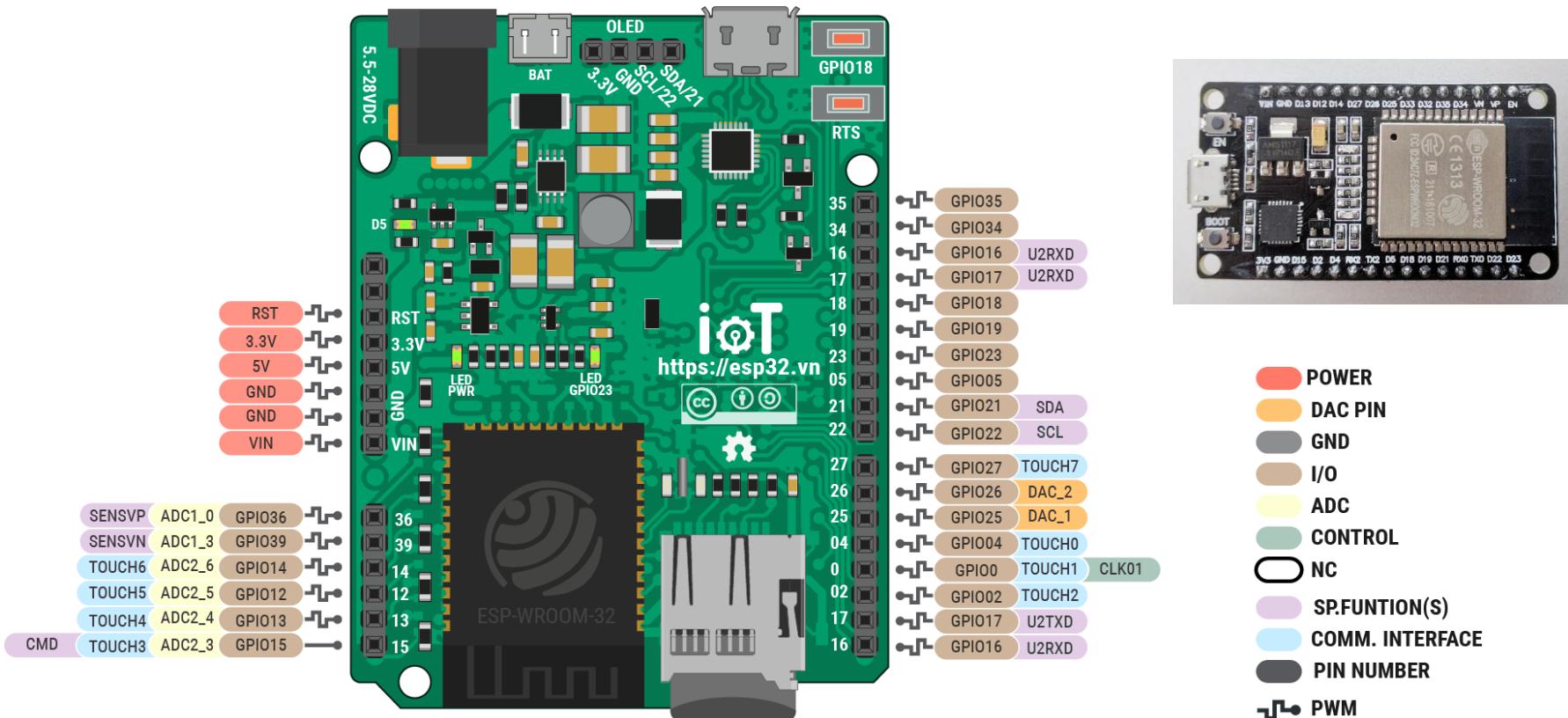
Lập trình với thiết bị IoT

- Các thiết bị dùng Vi xử lý 32/64 bit, hiệu năng cao
 - Dùng hệ điều hành (Embedded Linux, Raspbian, Windows Embedded, Android, Ubuntu, ...)
 - Kết nối Wi-Fi, Ethernet, ...
 - Các giao tiếp ngoại vi qua cơ chế driver trên hệ điều hành
 - Có thể sử dụng các thư viện trên hệ điều hành (Java, Python, .Net Framework, OpenCV, ...)
 - Ví dụ: Raspberry Pi



Lập trình cho thiết bị IoT (ESP32)

- Lập trình cho ESP32
 - <https://esp32.vn/index.html>
 - <https://randomnerdtutorials.com/projects-esp32/>

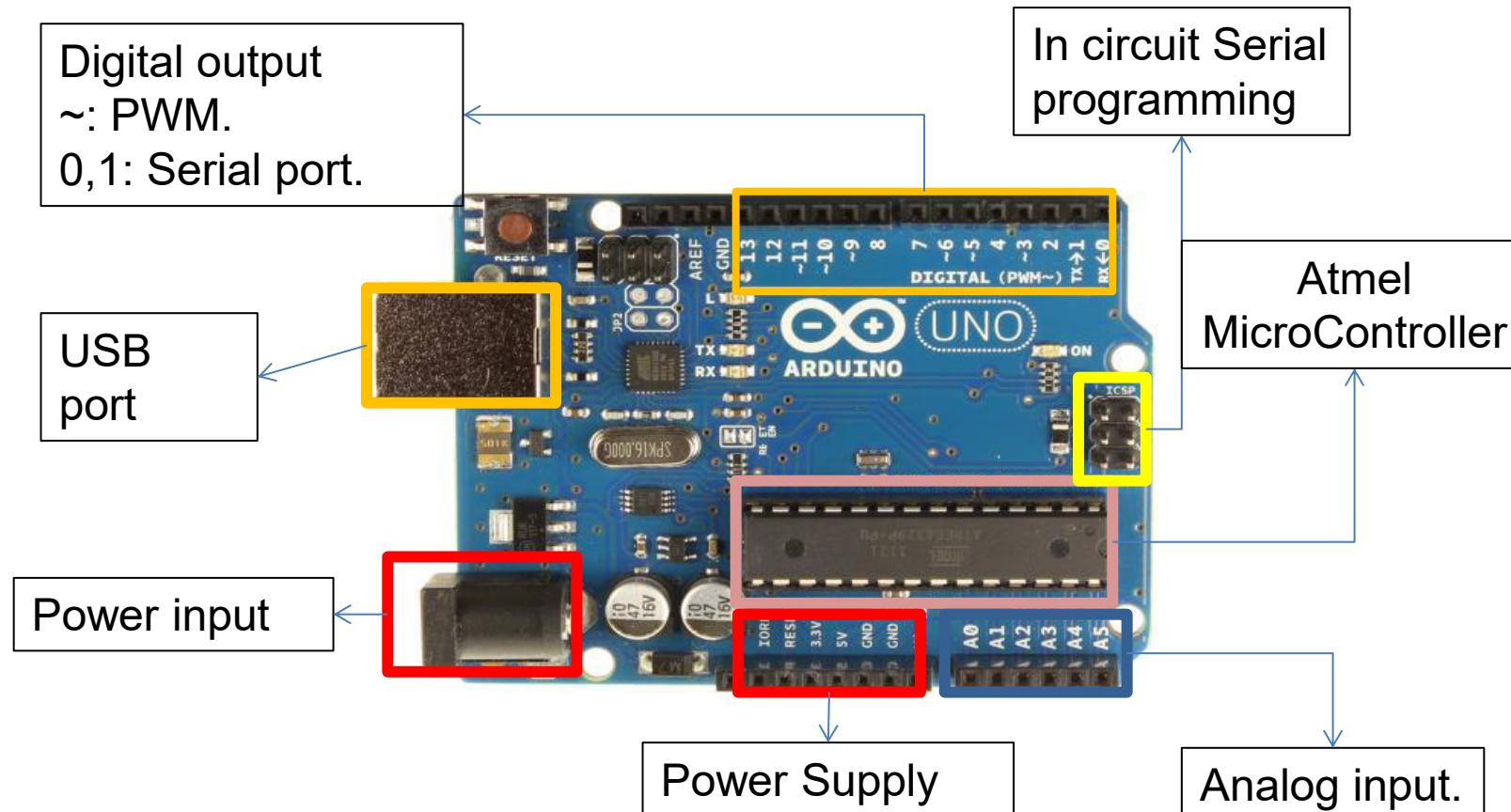


Lập trình thiết bị ESP32

- A. Môi trường lập trình Arduino IDE
- B. Kết nối WiFi cho ESP32
- C. Giao tiếp HTTP với ESP32
- D. Giao tiếp MQTT với ESP32
- E. Lập trình multi-tasks với FreeRTOS

Môi trường lập trình Arduino IDE

- Arduino IDE: Công cụ phát triển phần mềm cho vi điều khiển, máy tính nhúng
- <https://www.arduino.cc/>
- Ví dụ: Arduino Uno dùng Vi điều khiển ATmega328



Lập trình Arduino

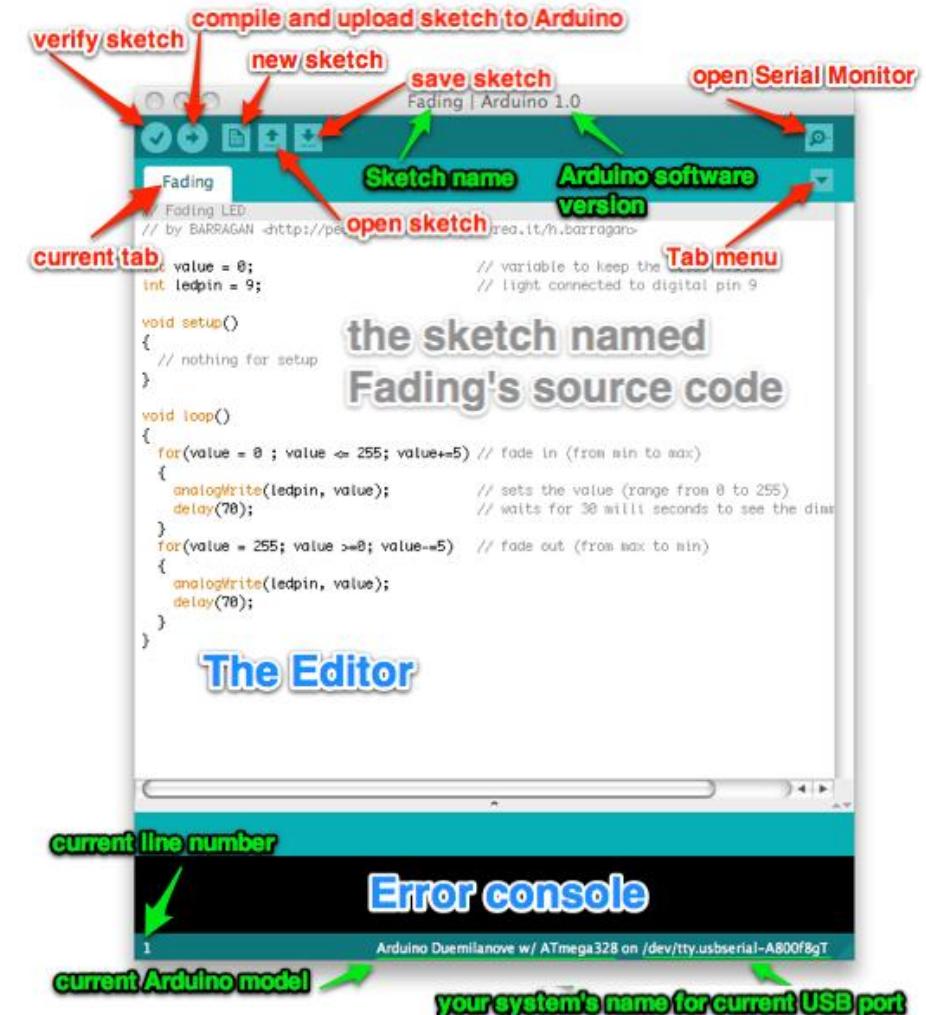
- Cài đặt Arduino IDE
 - <https://www.arduino.cc/en/guide/windows>
- Cấu trúc code

```
void setup()
{
    //Used to indicate the initial values of system on starting.
}

void loop()
{
    Contains the statements that will run whenever the system
    is powered after setup.
}
```

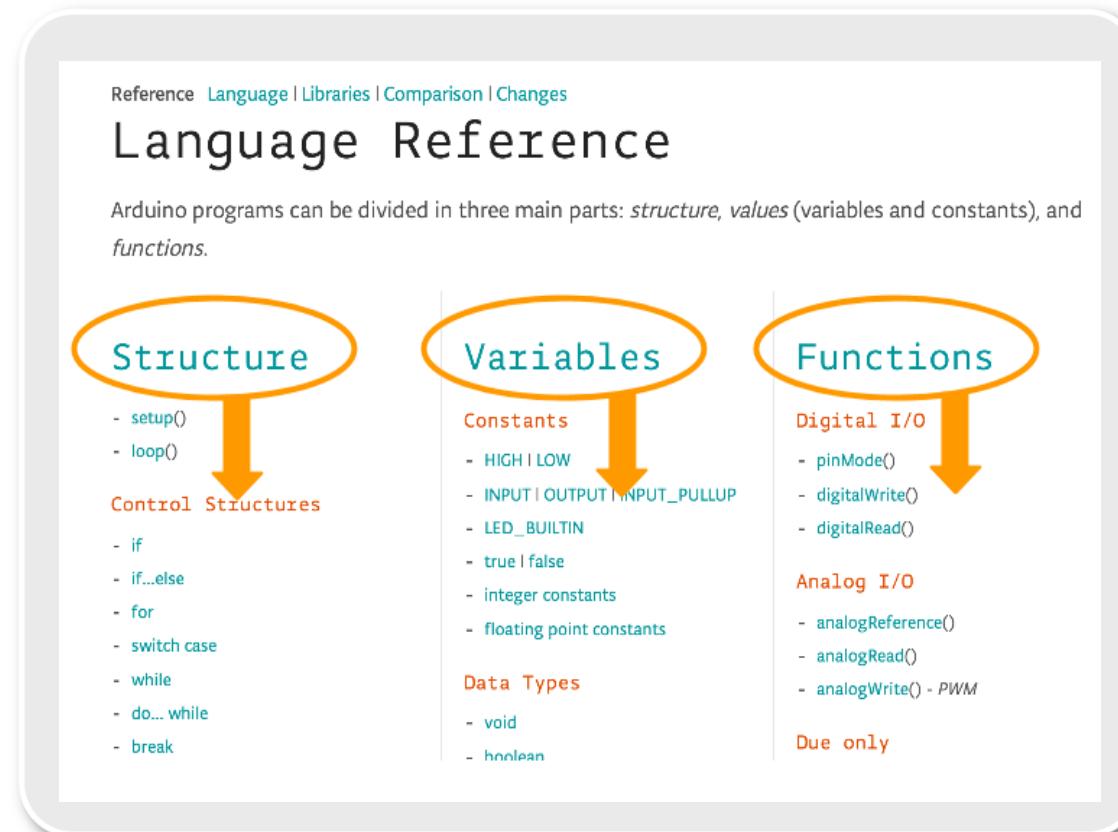
Lập trình Arduino

- Program used to code and upload it to arduino boards (using PC)
- Editor (for code edit)
- Sketch (piece of program)



Lập trình Arduino

- Tài liệu: Arduino References
- <http://arduino.cc/en/Reference/HomePage>



Ví dụ 1: Giao tiếp GPIO - Output

- LED on/off trên pin 2

```
void setup() {  
    // initialize digital pin 2 as an output.  
    pinMode(2, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(2, HIGH);      // turn the LED on  
    delay(1000);                // wait for a second  
    digitalWrite(2, LOW);       // turn the LED off  
    delay(1000);                // wait for a second  
}
```

Ví dụ 1: Giao tiếp GPIO - Input

- Đọc trạng thái nút bấm trên pin 2, turn on/off trên pin 13

```
const int buttonPin = 2; // the number of the pushbutton pin
const int ledPin = 13; // the number of the LED pin
int buttonState = 0;
void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}
void loop() {
    // read the state of the pushbutton value:
    buttonState = digitalRead(buttonPin);
    if (buttonState == HIGH) {
        digitalWrite(ledPin, HIGH);
    } else {
        digitalWrite(ledPin, LOW);
    }
    delay(100);
}
```

Ví dụ 2: Giao tiếp UART

- Sử dụng thư viện Serial của Arduino
 - <https://www.arduino.cc/reference/en/language/functions/communication/serial/>

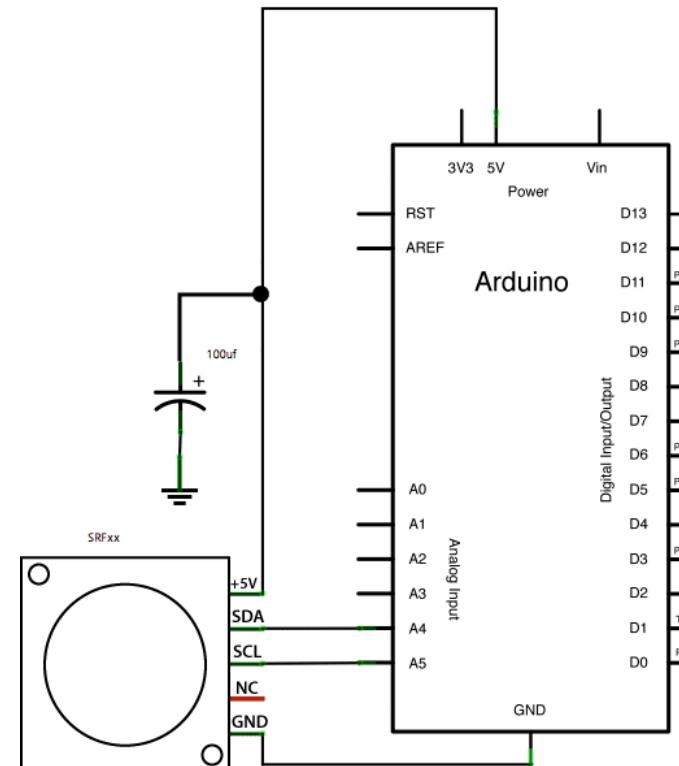
```
int incomingByte = 0;
void setup() {
    // opens serial port, sets data rate to 9600 bps
    Serial.begin(9600);
}
void loop() {
    // send data only when you receive data:
    if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = Serial.read();
        // say what you got:
        Serial.print("I received: ");
        Serial.println(incomingByte, DEC);
    }
}
```

Giao tiếp Serial

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(2, OUTPUT); //Cau hinh pin 2 la output  
    Serial.begin(9600);  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
    if(Serial.available() > 0){  
        int c = Serial.read();  
        if(c == '0'){  
            digitalWrite(2, 0);  
        }  
        else if(c == '1'){  
            digitalWrite(2, 1);  
        }  
        Serial.print("Toi nhan duoc: ");  
        Serial.println(c);  
    }  
}
```

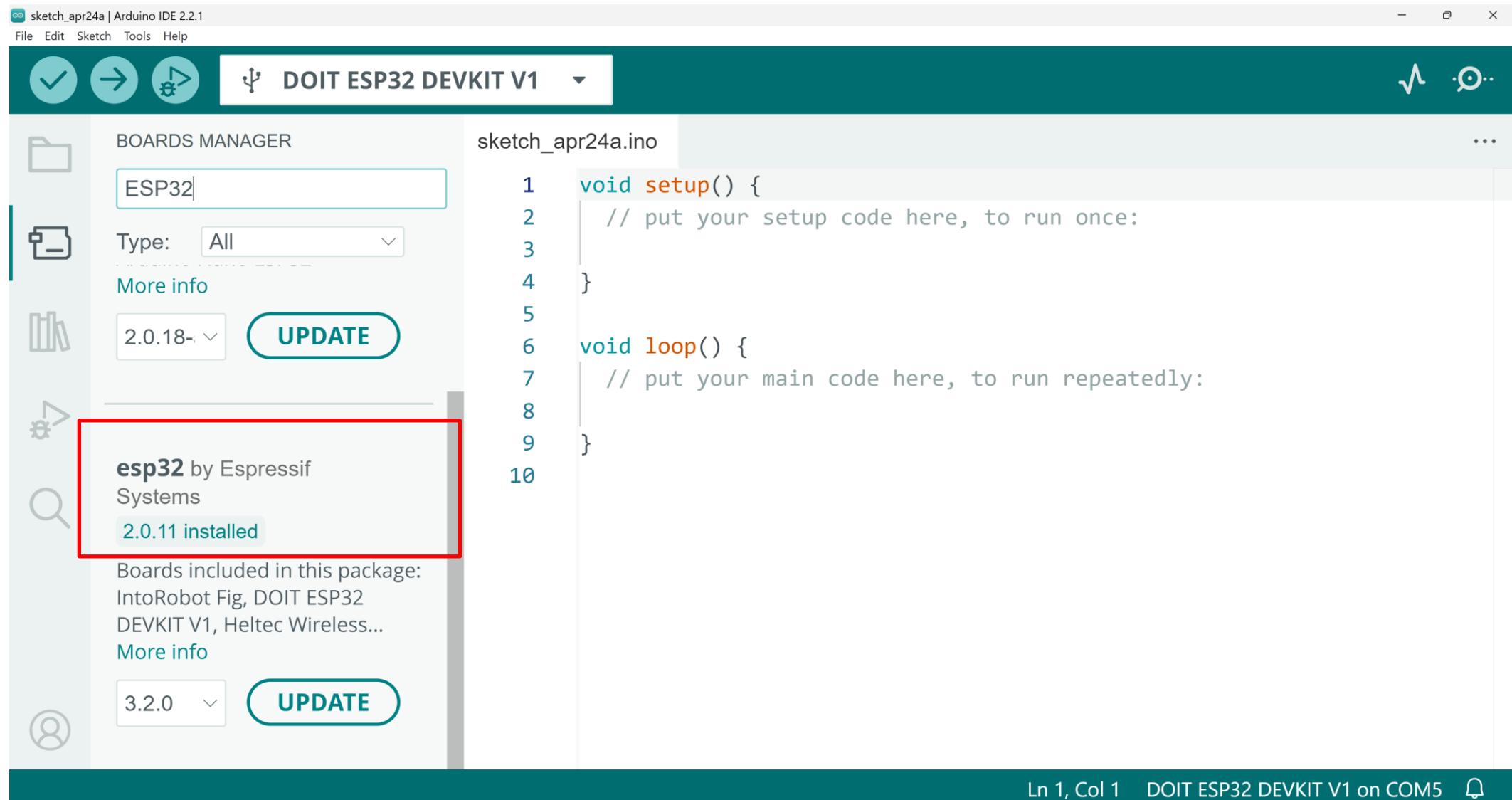
Ví dụ 3: Giao tiếp I2C

- Sử dụng thư viện Wire
 - <https://www.arduino.cc/en/Reference/Wire>
- Giao tiếp SRFxx Sonic Range Finder
 - <https://www.arduino.cc/en/Tutorial/LibraryExamples/SFRRangerReader>



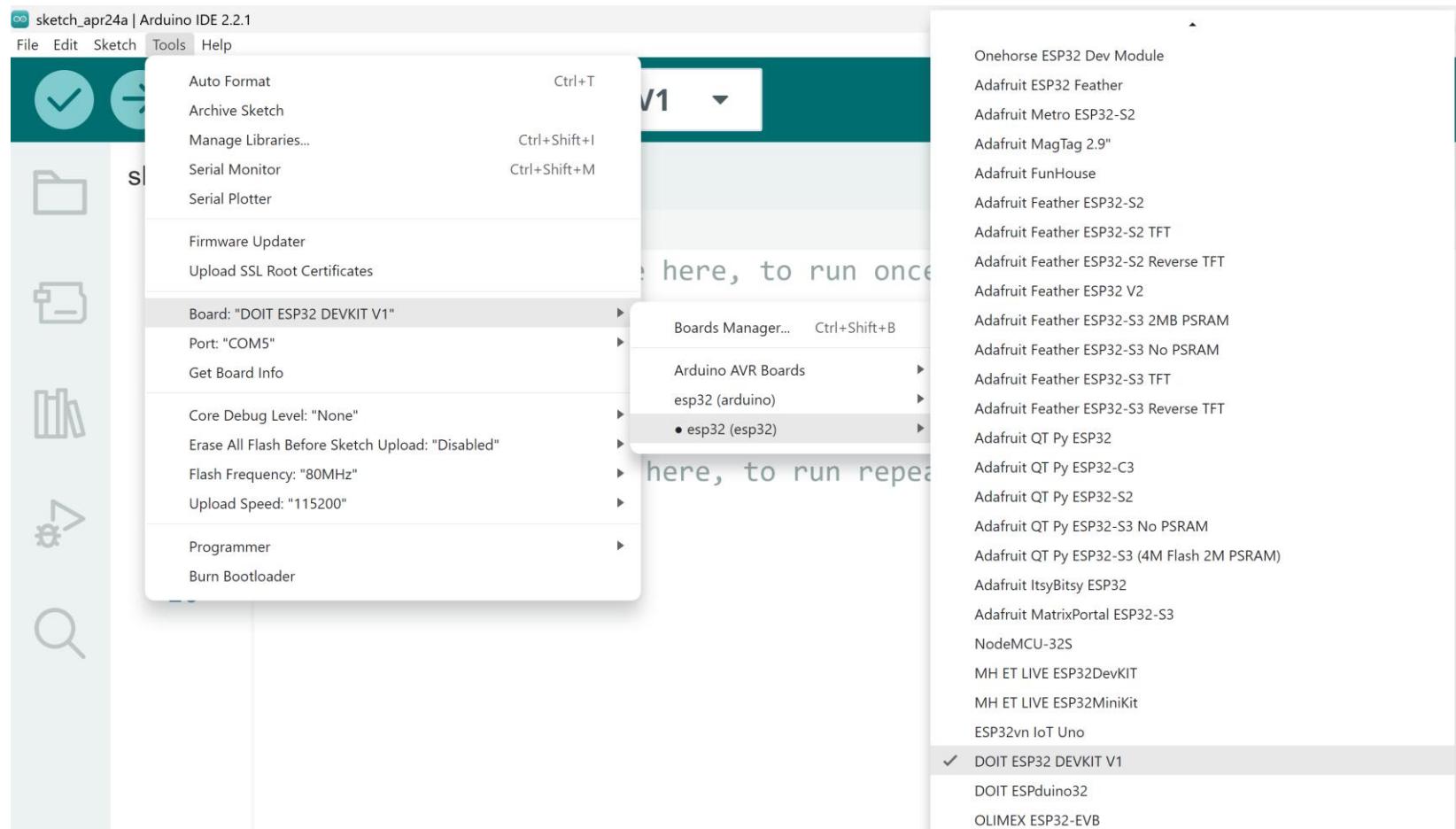
Thiết lập Arduino IDE cho ESP32

3) Chọn Tools > Board > Boards Manager...

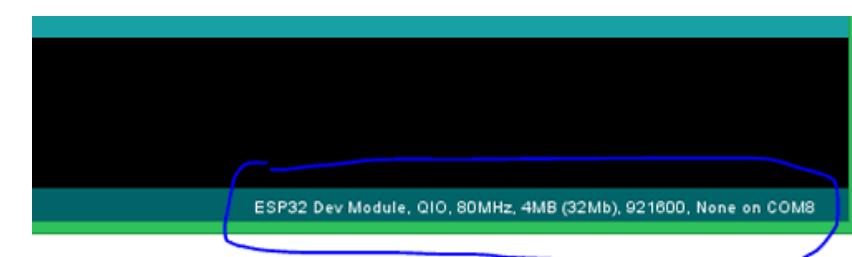
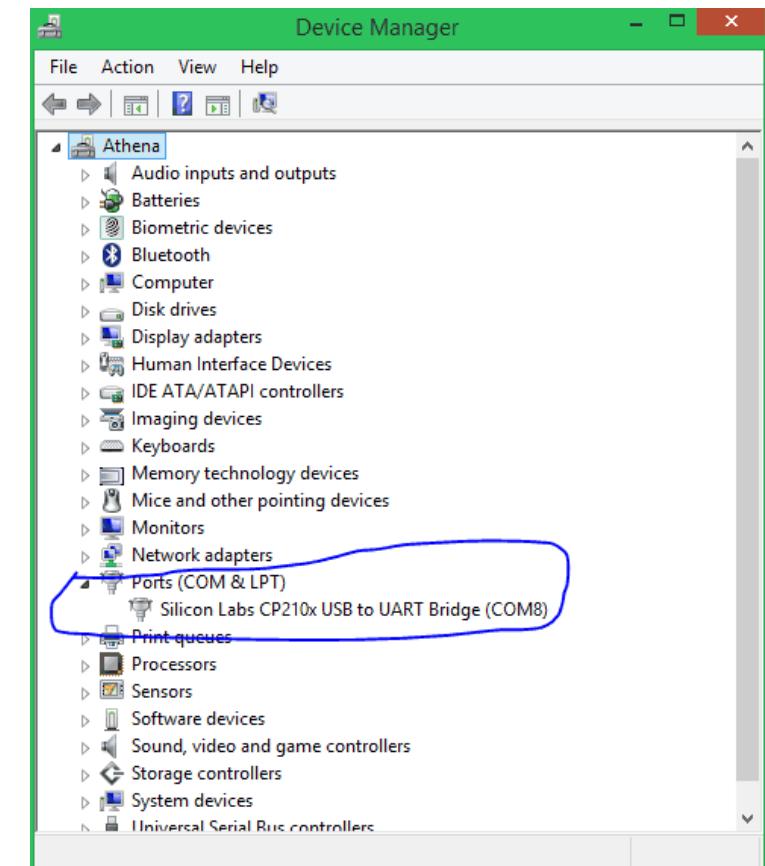
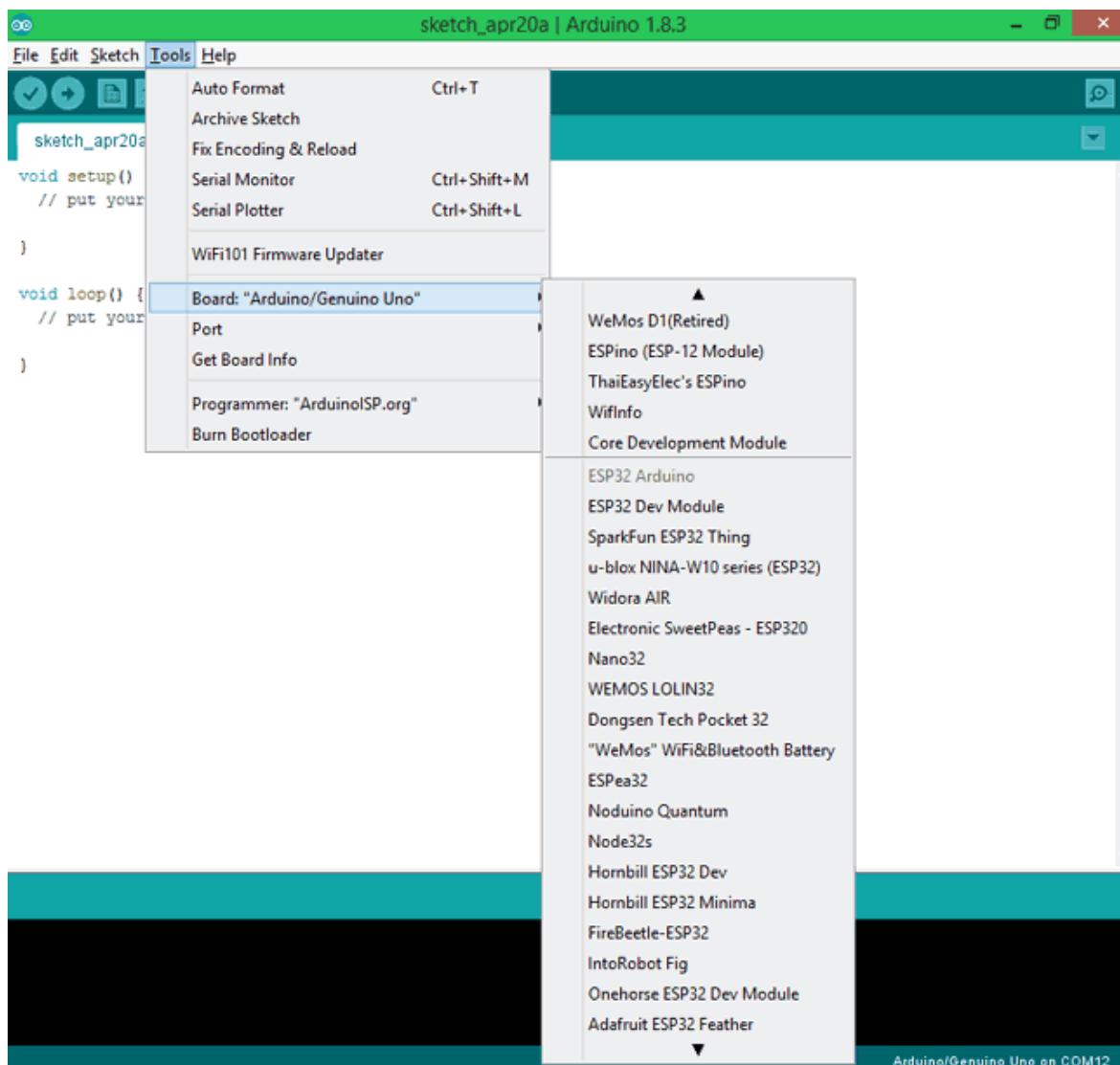


Thiết lập Arduino IDE cho ESP32

- Kết nối bo mạch ESP32 với máy tính
 - Chọn Board trong **Tools > Board** menu (ví dụ **DOIT ESP32 DEVKIT V1**)



Kết nối với thiết bị



Nạp chương trình cho thiết bị

- Upload the Blink Program. This program should blink the LED at an interval of 1 second.

The screenshot shows the Arduino IDE interface with the title bar "ESP32_Blink_Program | Arduino 1.8.3". The code editor contains the following sketch:

```
int LED_BUILTIN = 2;

void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

Below the code editor, the serial monitor window displays the upload progress:

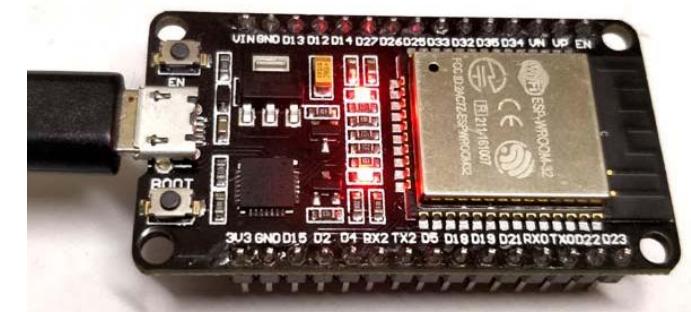
```
Done uploading.
Writing at 0x00020000... (83 %)
Writing at 0x00024000... (100 %)
Wrote 160128 bytes (81987 compressed) at 0x00010000 in 1.4 seconds (effective 914.4 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 122...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (effective 2234.2 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting...
```

At the bottom of the serial monitor, it says "ESP32 Dev Module, QIO, 80MHz, 4MB (32Mb), 921600, None on COM8".

```
int LED_BUILTIN = 2;
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```



B. Kết nối Wi-Fi cho ESP32

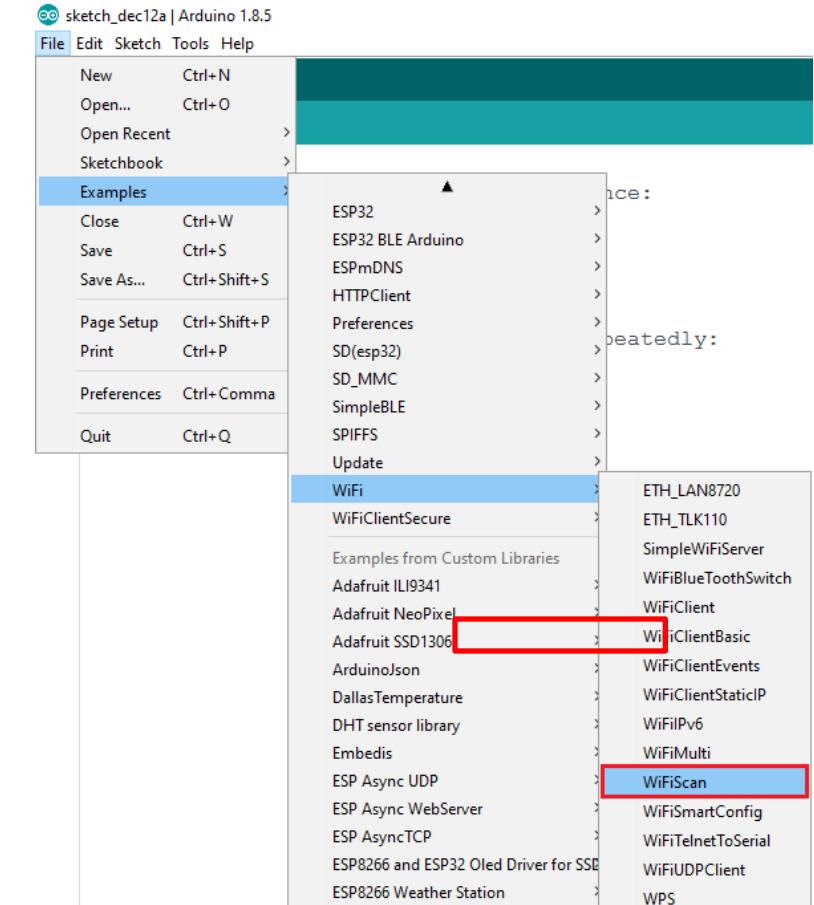
Ví dụ 1: File > Examples > WiFi (ESP32) > WiFiClientBasic

```
#include <WiFi.h>
const char* ssid = "yourNetworkName";
const char* password = "yourNetworkPass";

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println("Connecting to WiFi..");
    }
    Serial.println("Connected to the WiFi network");
}

void loop() {
```



Ví dụ 2: File > Examples > WiFi (ESP32) > WiFiClient

Ví dụ kết nối Wi-Fi

- Kết quả: Ghi log ra Serial port

```
1 #include "WiFi.h"
2 const char* ssid = "HUNG-DELLPC";
3 const char* password = "hung1234";
4 void setup() {
5     // put your setup code here, to run once:
6     Serial.begin(9600);
7     WiFi.begin(ssid, password);
8     while(WiFi.status() != WL_CONNECTED){
9         delay(500);
10        Serial.println("Dang cho ket noi WiFi ...");
11    }
12    Serial.println("WiFi connected");
13    Serial.print("Local IP: ");
14    Serial.println(WiFi.localIP());
15 }
```

Output Serial Monitor X

Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

Dang cho ket noi WiFi ...
Dang cho ket noi WiFi ...
WiFi connected
Local IP: 192.168.137.144

C. Giao tiếp HTTP với ESP32

- HTTP request methods: GET vs. POST
- HTTP GET:
 - Sử dụng để request dữ liệu/tài nguyên (get values from APIs)
 - Dữ liệu được gửi trong URL của GET request
 - Ví dụ:
GET /update-sensor?temperature=value1
 - Hoặc get JSON object:

GET /get-sensor



Giao tiếp HTTP với ESP32

■ HTTP POST:

- Gửi dữ liệu đến server để create/update tài nguyên
- Dữ liệu được gửi trong body của POST request
- Ví dụ:

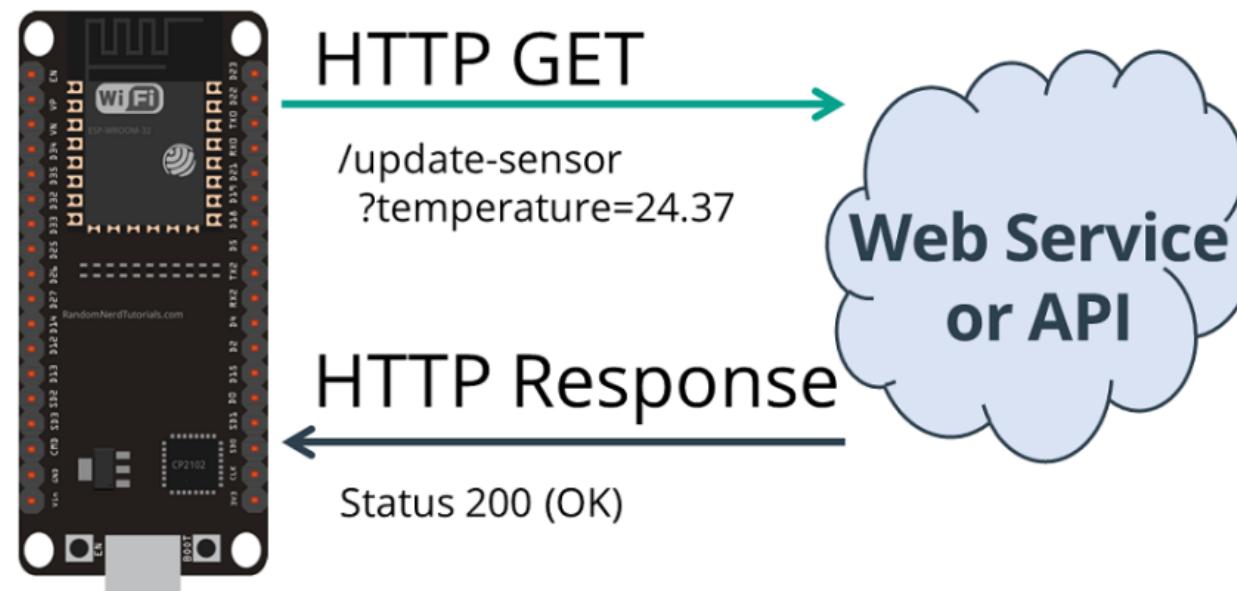
```
POST /update-sensor HTTP/1.1 Host: example.com  
api_key=api&sensor_name=name&temperature=value1&humidity=value2&pressure=value3  
Content-Type: application/x-www-form-urlencoded
```

■ Hoặc gửi JSON object:

```
POST /update-sensor HTTP/1.1 Host: example.com {api_key: "api",  
sensor_name: "name", temperature: value1, humidity: value2, pressure: value3}  
Content-Type: application/json
```

ESP32 HTTP GET

- ESP 32 gửi dữ liệu lên server dùng http get request



<https://randomnerdtutorials.com/esp32-http-get-post-arduino/>

ESP32 HTTP GET – Code minh họa

Part 1. Connect WiFi

```
#include <WiFi.h>
#include <HTTPClient.h>
const char* ssid = "HungiPhone";
const char* password = "hung1234";
String serverName = "https://postman-echo.com/get";
void setup()
{
    Serial.begin(115200);
    delay(10);
    // We start by connecting to a WiFi network
    WiFi.begin(ssid, password);
    Serial.print("Wait for WiFi... ");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.println("Connecting to WiFi ...");
        delay(1000);
    }
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    delay(500);
}
```

ESP32 HTTP GET – Code minh họa

```
void loop()
{
    if (WiFi.status() == WL_CONNECTED) {
        HttpClient http;
        String serverPath = serverName + "?temp1=24.7&temp2=30";
        // Your Domain name with URL path or IP address with path
        http.begin(serverPath.c_str()); // Send HTTP GET request
        int httpResponseCode = http.GET();
        if (httpResponseCode > 0) {
            Serial.print("HTTP Response code: "); Serial.println(httpResponseCode);
            String payload = http.getString();
            Serial.println(payload);
        }
        else {
            Serial.print("Error code: "); Serial.println(httpResponseCode);
        }
        http.end();
    }
    else {
        Serial.println("WiFi Disconnected");
    }
    delay(3000);
}
```

Part 2. Send http GET request

ESP32 HTTP GET – Mô phỏng trên Wokwi

WOKWI

SAVE ▾ SHARE ❤ ESP32-HTTP Get 🖊 Docs

sketch.ino ● diagram.json libraries.txt Library Manager ▾

```
1 #include <WiFi.h>
2 #include <HTTPClient.h>
3 String serverName = "https://postman-echo.com/get";
4 void setup() {
5     //setup for serial communication
6     Serial.begin(9600);
7     Serial.print("Connecting to WiFi");
8     //setup for WiFi connection
9     WiFi.begin("Wokwi-GUEST", "", 6);
10    while (WiFi.status() != WL_CONNECTED) {
11        delay(100);
12        Serial.print(".");
13    }
14    Serial.println("WiFi Connected!");
15 }
16 void loop() {
17    if (WiFi.status() == WL_CONNECTED) {
18        HTTPClient http;
19        String serverPath = serverName + "?temp=24.7&humid=30";
20        http.begin(serverPath.c_str()); // Send HTTP GET request
21        int httpResponseCode = http.GET();
22        if (httpResponseCode > 0) {
23            Serial.print("HTTP Response code: "); Serial.println(httpResponseCode);
24            String payload = http.getString();
25            Serial.println(payload);
26        } else {
27            Serial.print("Error code: "); Serial.println(httpResponseCode);
28        }
29        http.end();
30    }
31    else {
32        Serial.println("WiFi Disconnected");
33    }
34    delay(3000);
35 }
```

Simulation

03:05.893 46%

```
"x-forwarded-port": "443",
"host": "postman-echo.com",
"x-amzn-trace-id": "Root=1-65407dd8-3df387db2be1fa0328925412",
"user-agent": "ESP32HTTPClient",
"accept-encoding": "identity;q=1,chunked;q=0.1,*;q=0"
},
"url": "https://postman-echo.com/get?temp=24.7&humid=30"
}
HTTP Response code: 200
{
"args": {
"temp": "24.7",
```

```
#include <WiFi.h>
#include <HTTPClient.h>
String serverName = "https://postman-echo.com/get";
void setup() {
    //setup for serial communication
    Serial.begin(9600);
    Serial.print("Connecting to WiFi");
    //setup for WiFi connection
    WiFi.begin("Wokwi-GUEST", "", 6);
    while (WiFi.status() != WL_CONNECTED) {
        delay(100);
        Serial.print(".");
    }
    Serial.println("WiFi Connected!");
}
void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        String serverPath = serverName + "?temp=24.7&humid=30";
        http.begin(serverPath.c_str()); // Send HTTP GET request
        int httpResponseCode = http.GET();
        if (httpResponseCode > 0) {
            Serial.print("HTTP Response code: "); Serial.println(httpResponseCode);
            String payload = http.getString();
            Serial.println(payload);
        }
        else {
            Serial.print("Error code: "); Serial.println(httpResponseCode);
        }
        http.end();
    }
    else {
        Serial.println("WiFi Disconnected");
    }
    delay(3000);
}
```

ESP32 HTTP Get

✓ 1. Khai báo thư viện và thông tin kết nối

cpp

Copy

```
#include <WiFi.h>
#include <HTTPClient.h>
```

Dùng thư viện WiFi để kết nối mạng và HTTPClient để gửi yêu cầu HTTP.

cpp

Copy

```
const char* ssid = "HungiPhone";
const char* password = "hung1234";
String serverName = "https://postman-echo.com/get";
```

- “ssid và password : tên mạng WiFi và mật khẩu”
- “serverName : địa chỉ server để gửi dữ liệu (ở đây dùng postman-echo để test)”

ESP32 HTTP Get

✓ 2. Kết nối WiFi (trong `setup()`)

cpp

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Connecting to WiFi ...");
    delay(1000);
}
Serial.println("WiFi connected");
```

- “Lắp lại cho đến khi ESP32 kết nối được với WiFi.”
- “In địa chỉ IP sau khi kết nối thành công.”

ESP32 HTTP Get

✓ 3. Gửi dữ liệu qua HTTP GET (trong loop())

cpp

Copy

```
if (WiFi.status() == WL_CONNECTED) {  
    HttpClient http;  
    String serverPath = serverName + "?temp1=24.7&temp2=30";  
    http.begin(serverPath.c_str());
```

- “Tạo URL có chứa tham số (temp1 , temp2)”
- “Bắt đầu HTTP GET request bằng http.begin() ”

ESP32 HTTP Get

✓ 3. Gửi dữ liệu qua HTTP GET (trong loop())

cpp

Copy

```
if (WiFi.status() == WL_CONNECTED) {  
    HttpClient http;  
    String serverPath = serverName + "?temp1=24.7&temp2=30";  
    http.begin(serverPath.c_str());
```

- “Tạo URL có chứa tham số (temp1 , temp2)”
- “Bắt đầu HTTP GET request bằng http.begin() ”

ESP32 HTTP Get

cpp

Copy

```
int httpResponseCode = http.GET();
if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String payload = http.getString();
    Serial.println(payload);
}
```

- “Gửi GET request → nhận HTTP response code”
- “Nếu thành công, in ra code và nội dung phản hồi từ server (payload)”

ESP32 HTTP Get

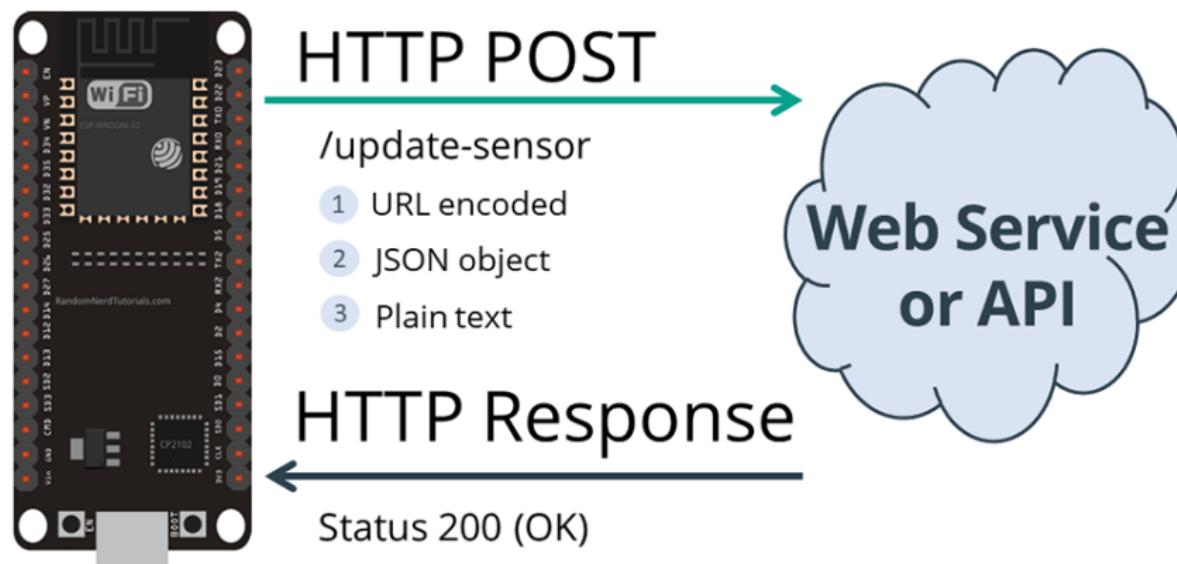
cpp

```
else {  
    Serial.print("Error code: ");  
    Serial.println(httpResponseCode);  
}  
http.end();
```

Nếu lỗi (ví dụ không gửi được), in mã lỗi.

ESP32 HTTP POST

- Gửi dữ liệu dùng:
 - URL encoded (đóng gói trong URL)
 - JSON object (đóng gói trong request body)
 - Plain text



Tham khảo code: <https://randomnerdtutorials.com/esp32-http-get-post-arduino/>

ESP32 HTTP POST

- POST dữ liệu với URL encoded

POST /update-sensor HTTP/1.1 Host: 192.168.1.106:1880

api_key=tPmAT5Ab3j7F9&sensor=BME280&value1=24.25&value2=49.54&value3=1005.14 Content-Type: application/x-www-form-urlencoded

```
// Your Domain name with URL path or IP address with path
http.begin(serverName);
// Specify content-type header
http.addHeader("Content-Type", "application/x-www-form-urlencoded");
// Data to send with HTTP POST
String httpRequestData =
"api_key=tPmAT5Ab3j7F9&sensor=BME280&value1=24.25&value2=49.54&value3=1005.14";
// Send HTTP POST request
int httpResponseCode = http.POST(httpRequestData);
```

ESP32 HTTP POST

- Post JSON object

POST /update-sensor HTTP/1.1 Host: example.com {api_key: "tPmAT5Ab3j7F9", sensor_name: "BME280", temperature: 24.25; humidity: 49.54; pressure: 1005.14} Content-Type: application/json

```
http.addHeader("Content-Type", "application/json");
int httpResponseCode =
http.POST("{\"api_key\": \"tPmAT5Ab3j7F9\", \"sensor\": \"BME280\", \"value1\": \"24.25\", \"value2\": \"49
.54\", \"value3\": \"1005.14\"}");
```

- Post Plain Text

```
http.addHeader("Content-Type", "text/plain");
int httpResponseCode = http.POST("Hello, World!");
```

Ví dụ: HTTP post url-encoded

```
#include <WiFi.h>
#include <HTTPClient.h>
String serverName = "https://postman-echo.com/post";
void setup() {
    //setup for serial communication
    //setup for WiFi connection
}
void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        float x = 30.5;
        float y = 78;
        http.begin(serverName);
        http.addHeader("Content-Type", "application/x-www-form-urlencoded");
        String httpRequestData = "field1=" + String(x) + "&field2=" + String(y);
        Serial.println(httpRequestData);
        int httpResponseCode = http.POST(httpRequestData);
        if (httpResponseCode > 0) {
            Serial.print("HTTP Response code: "); Serial.println(httpResponseCode);
            String payload = http.getString();
            Serial.println(payload);
        }
        else {
            Serial.print("Error code: "); Serial.println(httpResponseCode);
        }
        http.end();
    }
    else {
        Serial.println("WiFi Disconnected");
    }
    delay(3000);
}
```

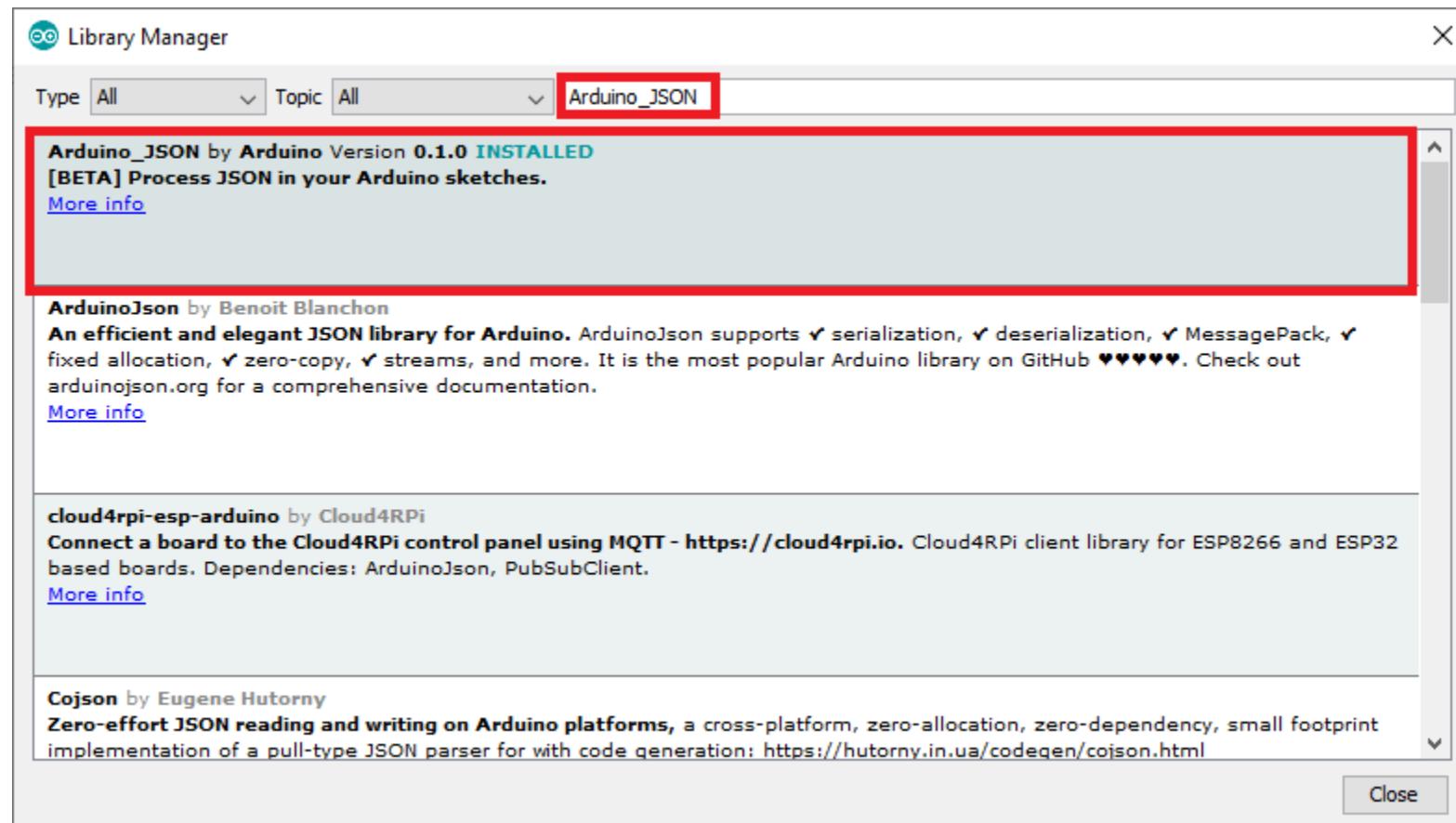
Ví dụ: HTTP post json object

```
void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        HttpClient http;
        float x = 30.5;
        float y = 78;
        int motion = 1;
        http.begin(serverName);
        http.addHeader("Content-Type", "application/json");
        String httpRequestData = "{\"value1\": 24.25,\"value2\": 49.54, \"motion\":1 }";
        //String httpRequestData = "{\"temperature\": " + String(x) + ",\"humidity\": " + String(y) +
        ",\"motion\": " + String(motion) + "}";
        Serial.println(httpRequestData);
        int httpResponseCode = http.POST(httpRequestData);

        if (httpResponseCode > 0) {
            Serial.print("HTTP Response code: "); Serial.println(httpResponseCode);
            String payload = http.getString();
            Serial.println(payload);
        }
        else { Serial.print("Error code: "); Serial.println(httpResponseCode); }
        http.end();
    }
    else { Serial.println("WiFi Disconnected");}
    delay(3000);
}
```

Sử dụng thư viện Arduino JSON

- Sketch > Include Library > Manage Libraries
- Add thư viện trên Arduino Library Manager



Sử dụng thư viện Arduino JSON

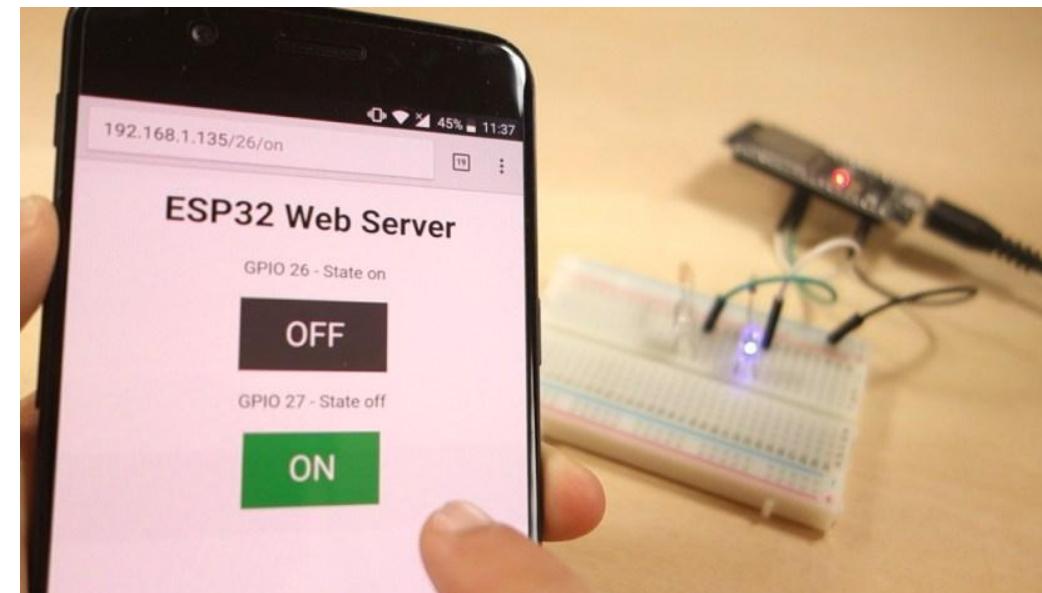
- <https://arduinojson.org/v6/example/string/>

Bài tập

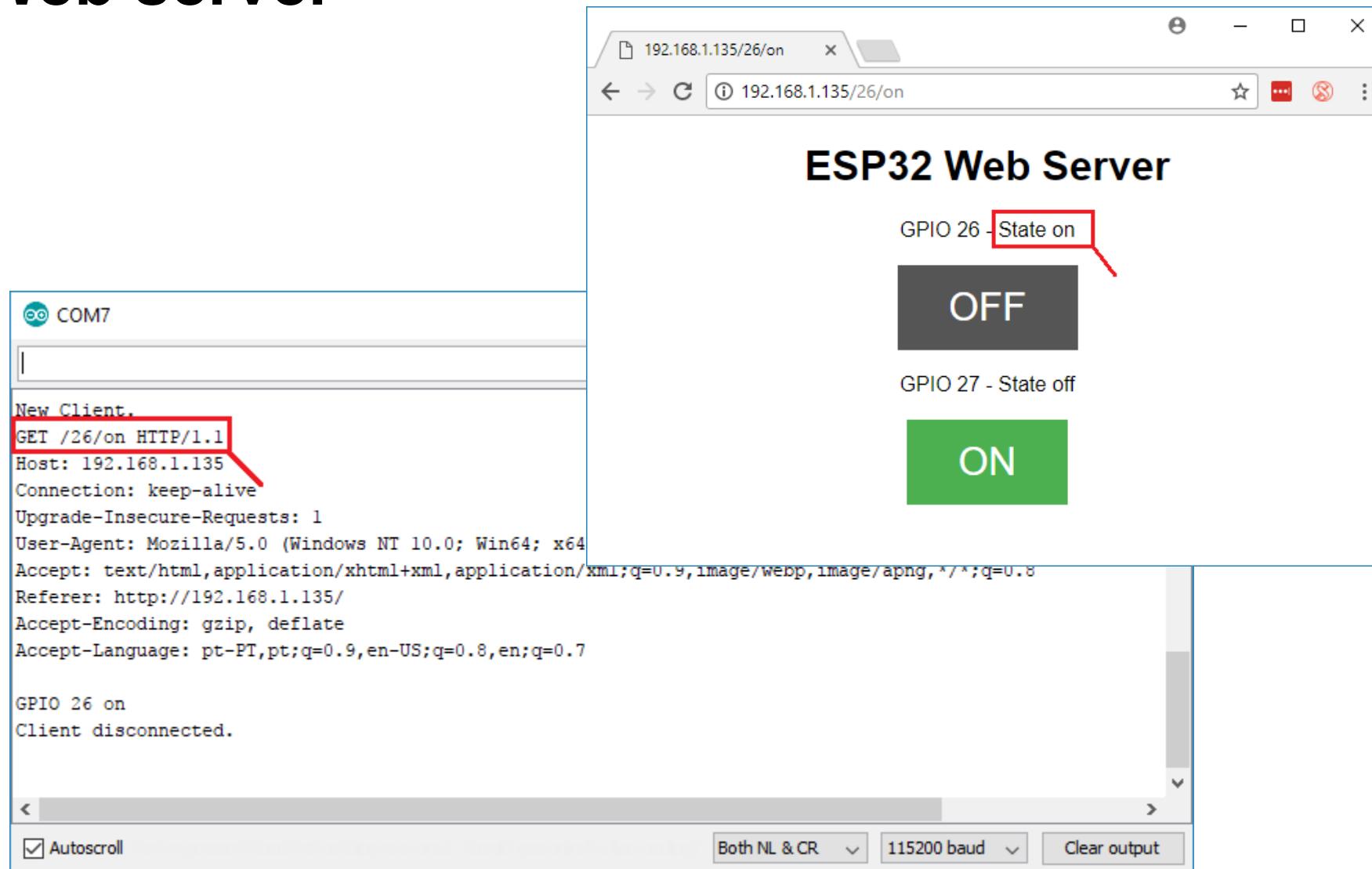
- Bài 5. Lập trình ESP32 gửi dữ liệu HTTP

ESP32 Web server

- Xây dựng một web server đơn giản trên ESP32, điều khiển GPIO
- Ví dụ:
 - <https://randomnerdtutorials.com/esp32-web-server-arduino-ide/>

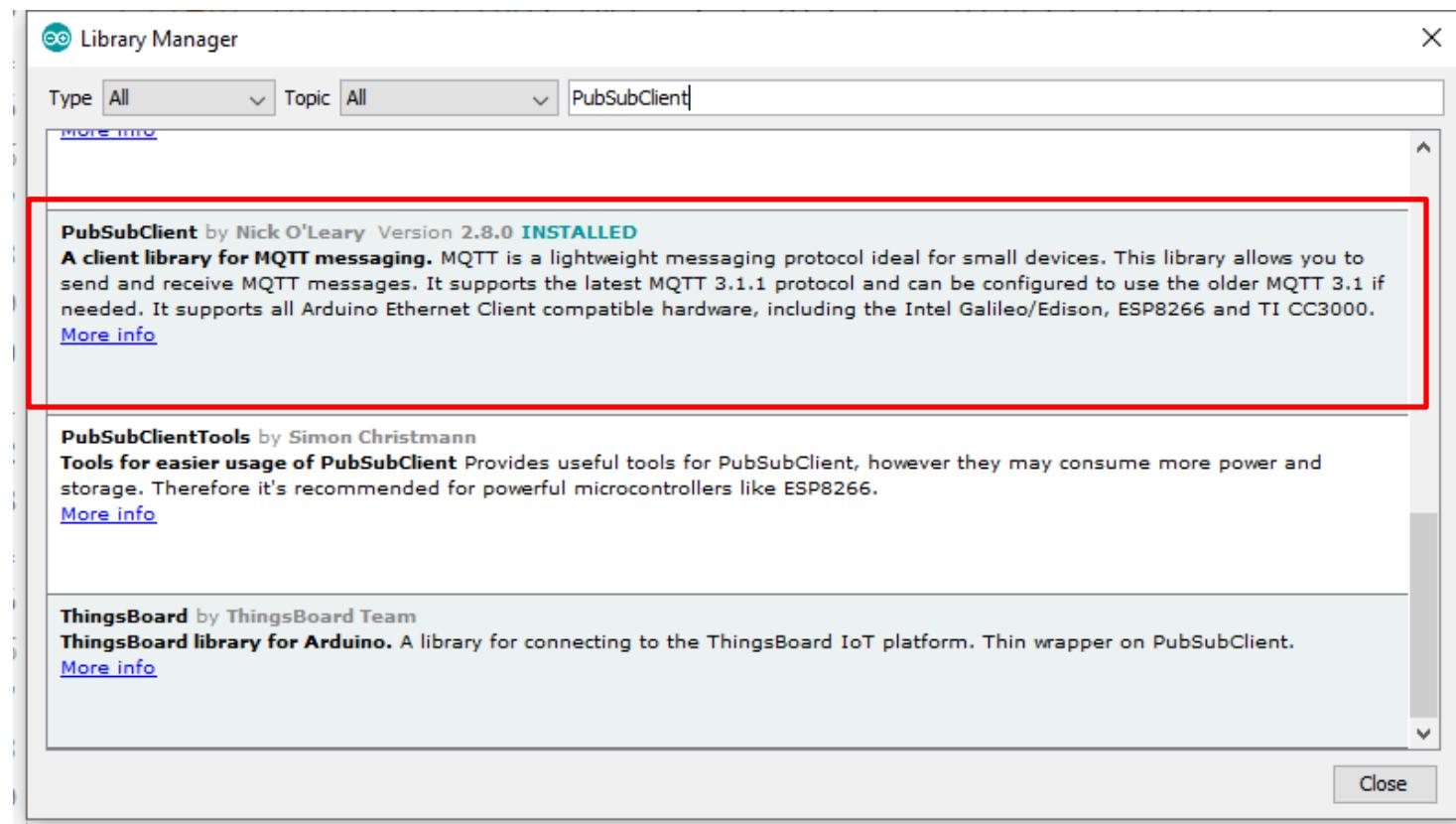


ESP32 Web server



D. Giao tiếp MQTT với ESP32

- Sử dụng thư viện mqtt PubSubClient cho Arduino
 - <https://www.arduinolibraries.info/libraries/pub-sub-client>
 - <https://github.com/knolleary/pubsubclient>



Giao tiếp MQTT với ESP32/ESP8266

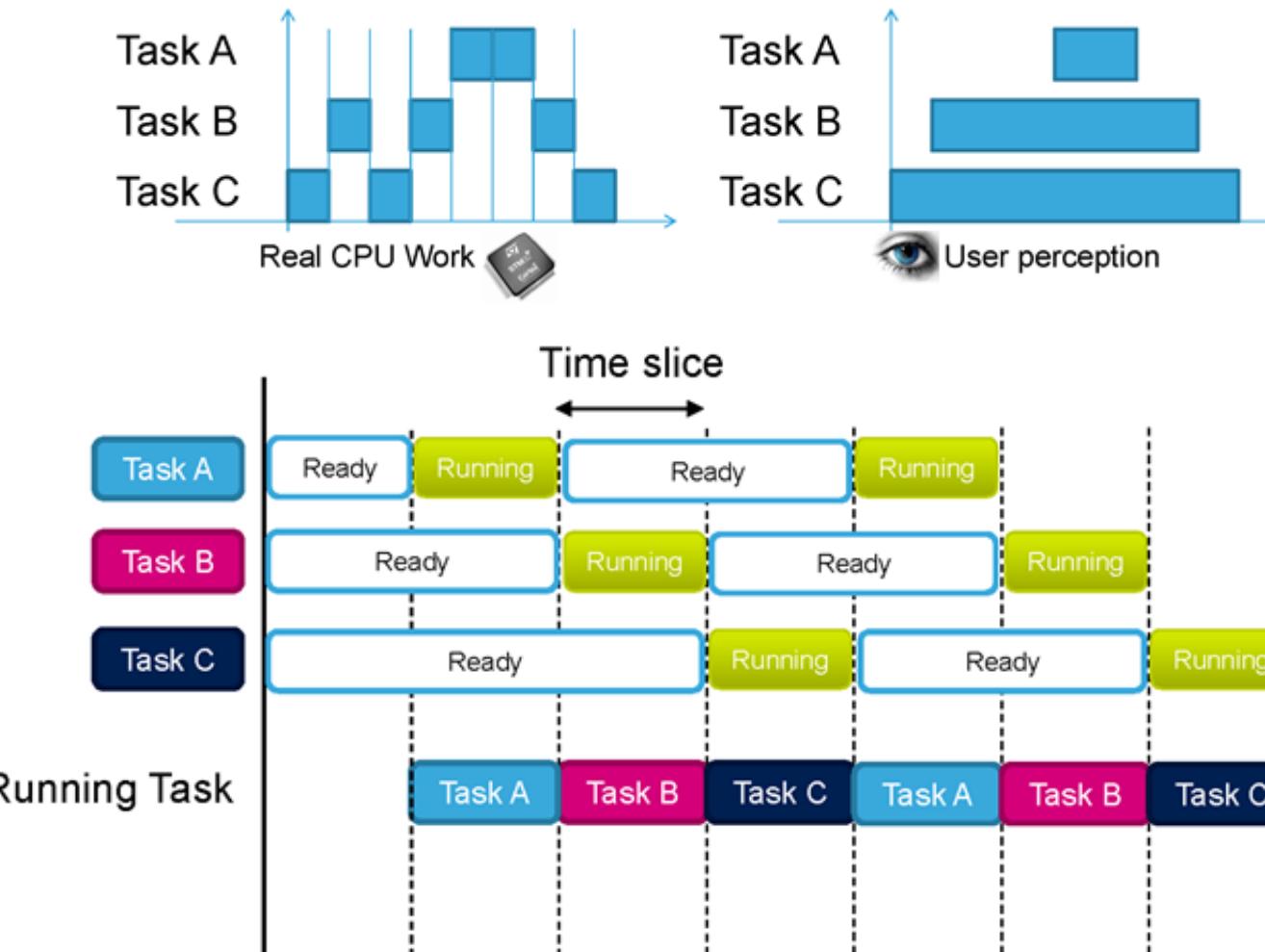
- Tham khảo mã nguồn:

- https://github.com/knolleary/pubsubclient/blob/master/examples/mqtt_esp8266/mqtt_esp8266.ino

```
#include <ESP8266WiFi.h>      //Hoặc #include <WiFi.h>
#include <PubSubClient.h>
WiFiClient espClient;
PubSubClient client(espClient);
const char* mqtt_server = "broker.mqtt-dashboard.com";
void setup_wifi() { ...}
void callback(char* topic, byte* payload, unsigned int length) { ...}
void reconnect() { ...}
void setup() {
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}
void loop() { ... }
```

E. Lập trình multi-task với FreeRTOS

- FreeRTOS multi-tasking



Lập trình multi-task với FreeRTOS

- Create a task

```
xTaskCreate(  
    BlinkLED, // Function that should be called  
    "Blink LED", // Name of the task (for debugging)  
    1024, // Stack size (bytes)  
    NULL, // Parameter to pass  
    1, // Task priority  
    NULL // Task handle );
```

Lập trình multi-task với FreeRTOS

■ Hàm xử lý của task

```
void BlinkLED(void * parameter) {  
    for(;;) { // infinite loop  
        // Turn the LED on  
        digitalWrite(led1, HIGH);  
        // Pause the task for 500ms  
        vTaskDelay(500 / portTICK_PERIOD_MS);  
        // Turn the LED off  
        digitalWrite(led1, LOW);  
        // Pause the task again for 500ms  
        vTaskDelay(500 / portTICK_PERIOD_MS);  
    }  
}
```

Programming multi-task with FreeRTOS

```
void BlinkLED( void *pvParameters );
void setup(){
    //Create the task 1 to run BlinkLED
    xTaskCreate(
        BlinkLED,      // Function that should be called
        "Blink LED",   // Name of the task (for debugging)
        1024,          // Stack size (bytes)
        NULL,          // Parameter to pass
        1,              // Task priority
        NULL           // Task handle
    );
}
void BlinkLED( void *pvParameters ){//The function for Task 1
    for(;;){ // infinite loop
        digitalWrite(led1, HIGH);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        digitalWrite(led1, LOW);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        Serial.println("LED is blinking ...");
    }
}
```

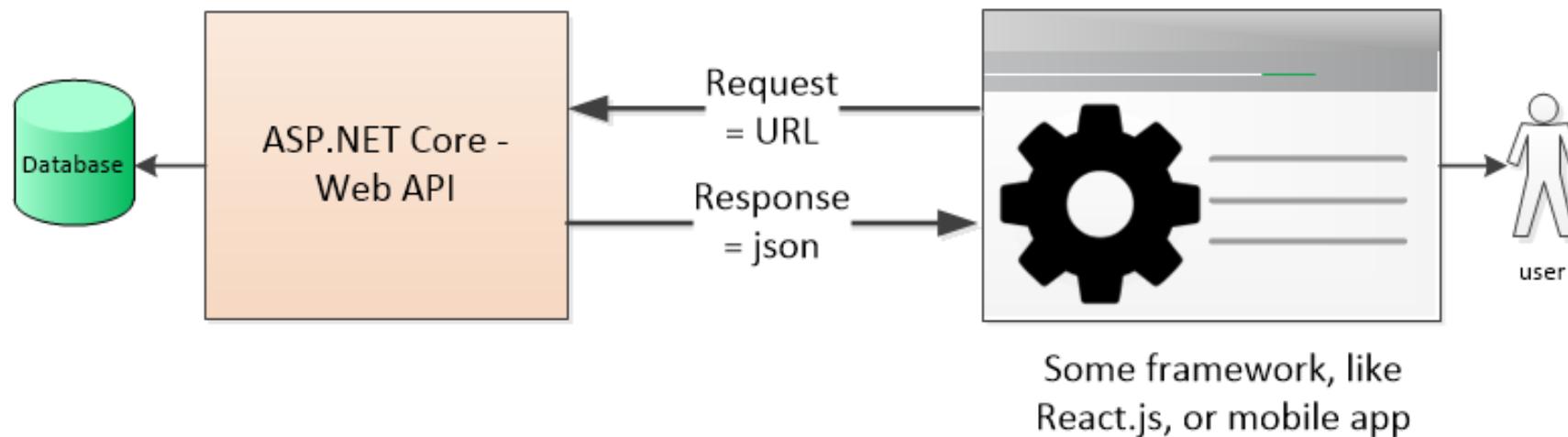
3.2. Xây dựng Web API

3.2.1. Công nghệ .Net Core, Entity Framework

3.2.2. Công nghệ NodeJS, MongoDB

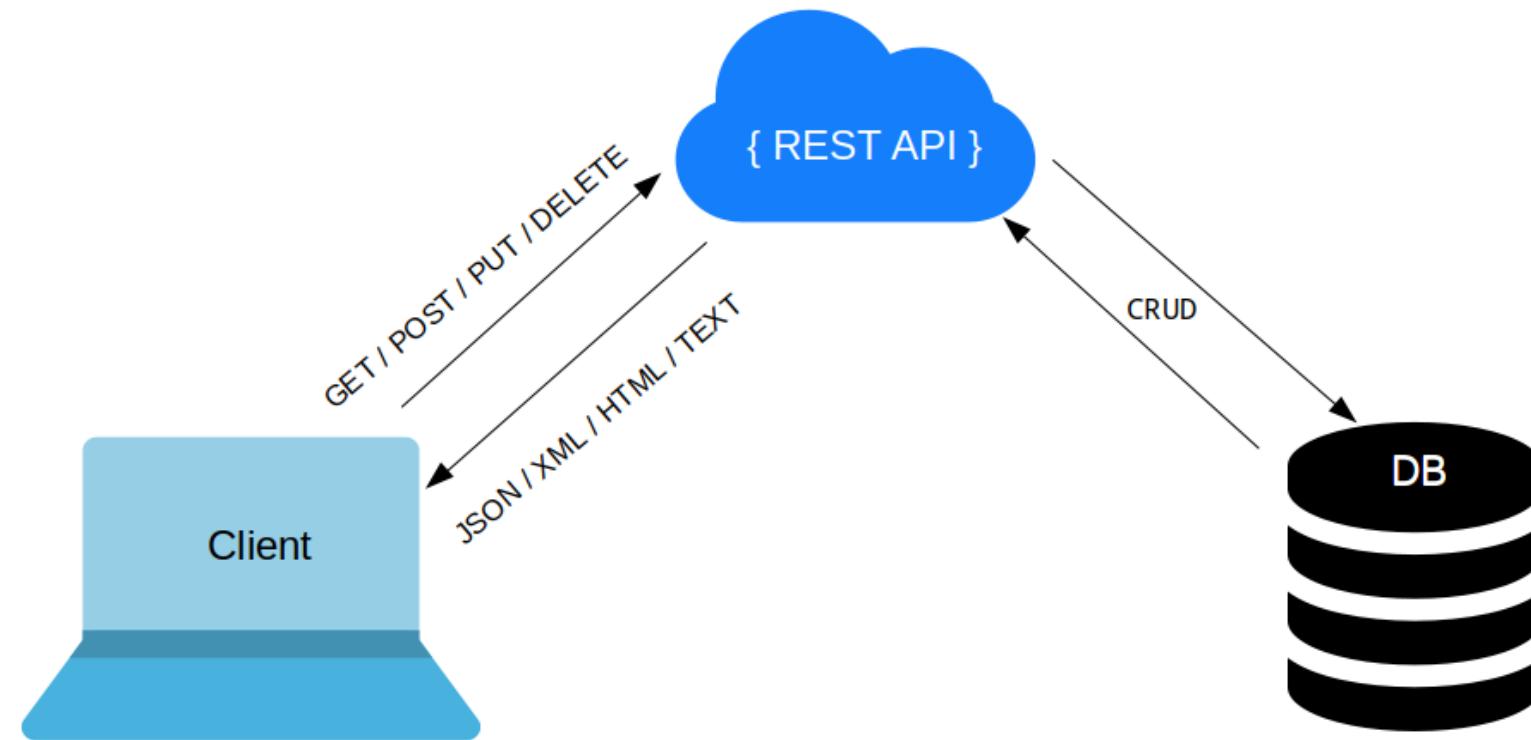
3.2.1. Xây dựng Web API với .NET Core

- Sử dụng công nghệ ASP.NET Core và Entity Framework để xây dựng Web API cho ứng dụng IoT



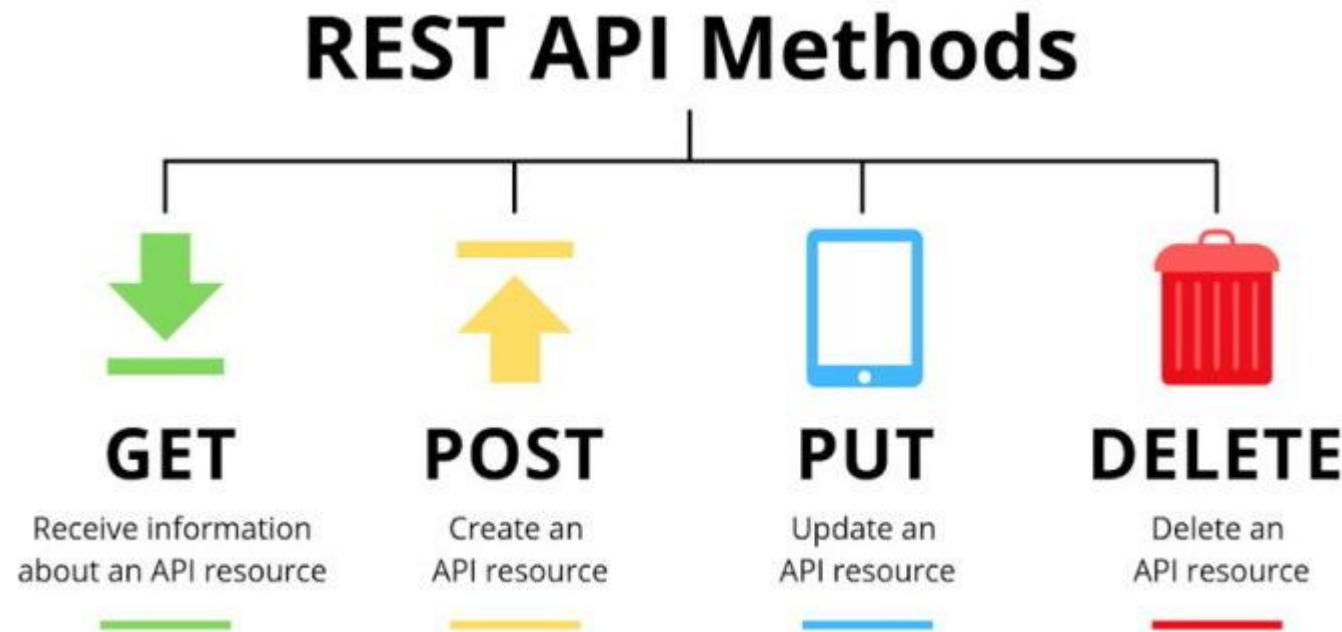
RESTful API

- REST = **R**epresentational **S**tate **T**ransfer: Các trạng thái tài nguyên được truyền tải qua HTTP
- Tiêu chuẩn dùng trong thiết kế Web API



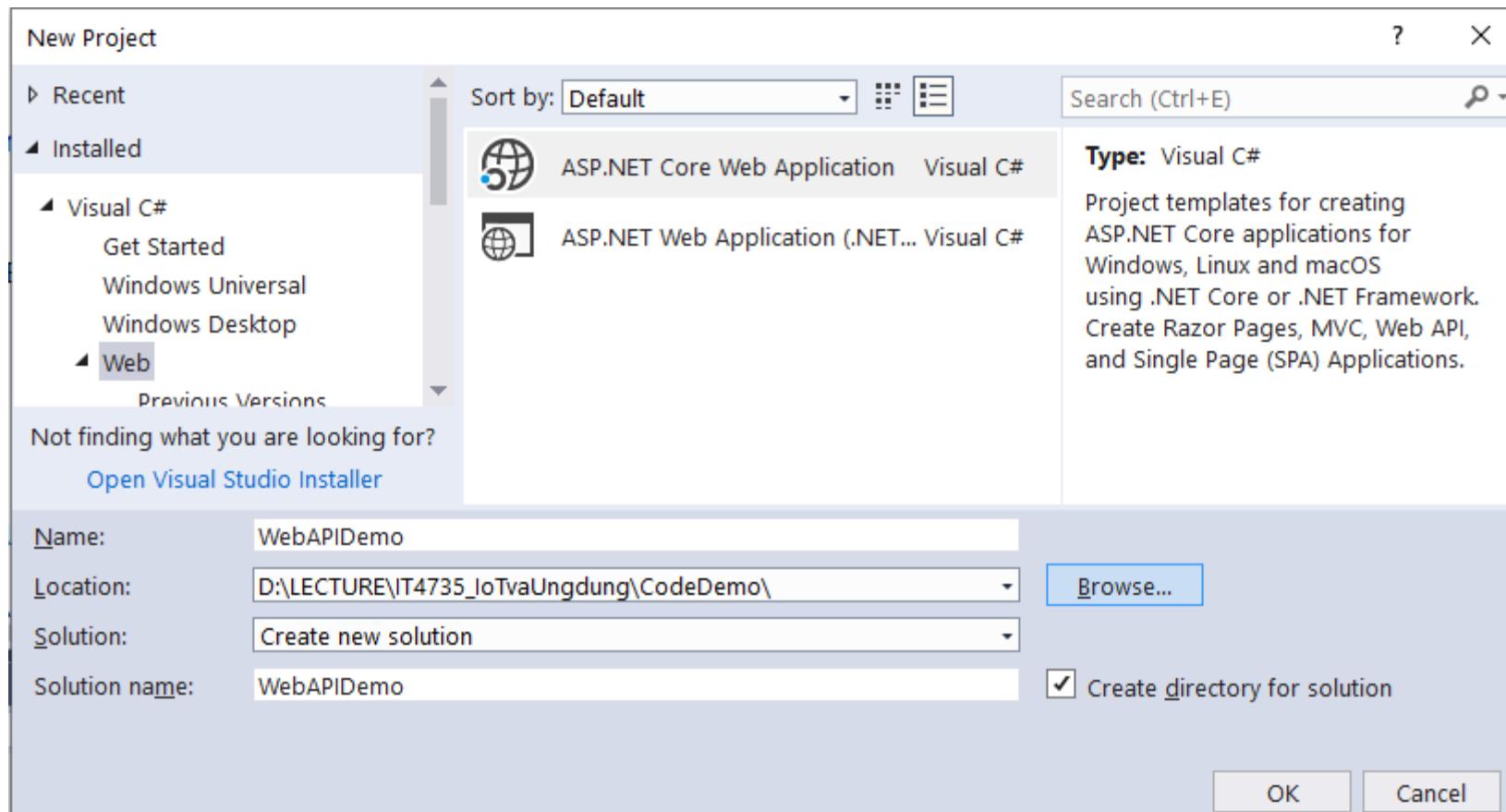
RESTful API

- Dựa trên giao thức HTTP. Sử dụng các phương thức http
 - GET: Lấy về (READ) một tài nguyên cụ thể (by Id) hoặc một danh sách tài nguyên (Resource).
 - POST: Tạo mới (CREATE) một tài nguyên.
 - PUT: Cập nhật (UPDATE) thông tin cho 1 tài nguyên (by Id).
 - DELETE: Xóa (DELETE) 1 tài nguyên (by Id).



Tạo Web API project

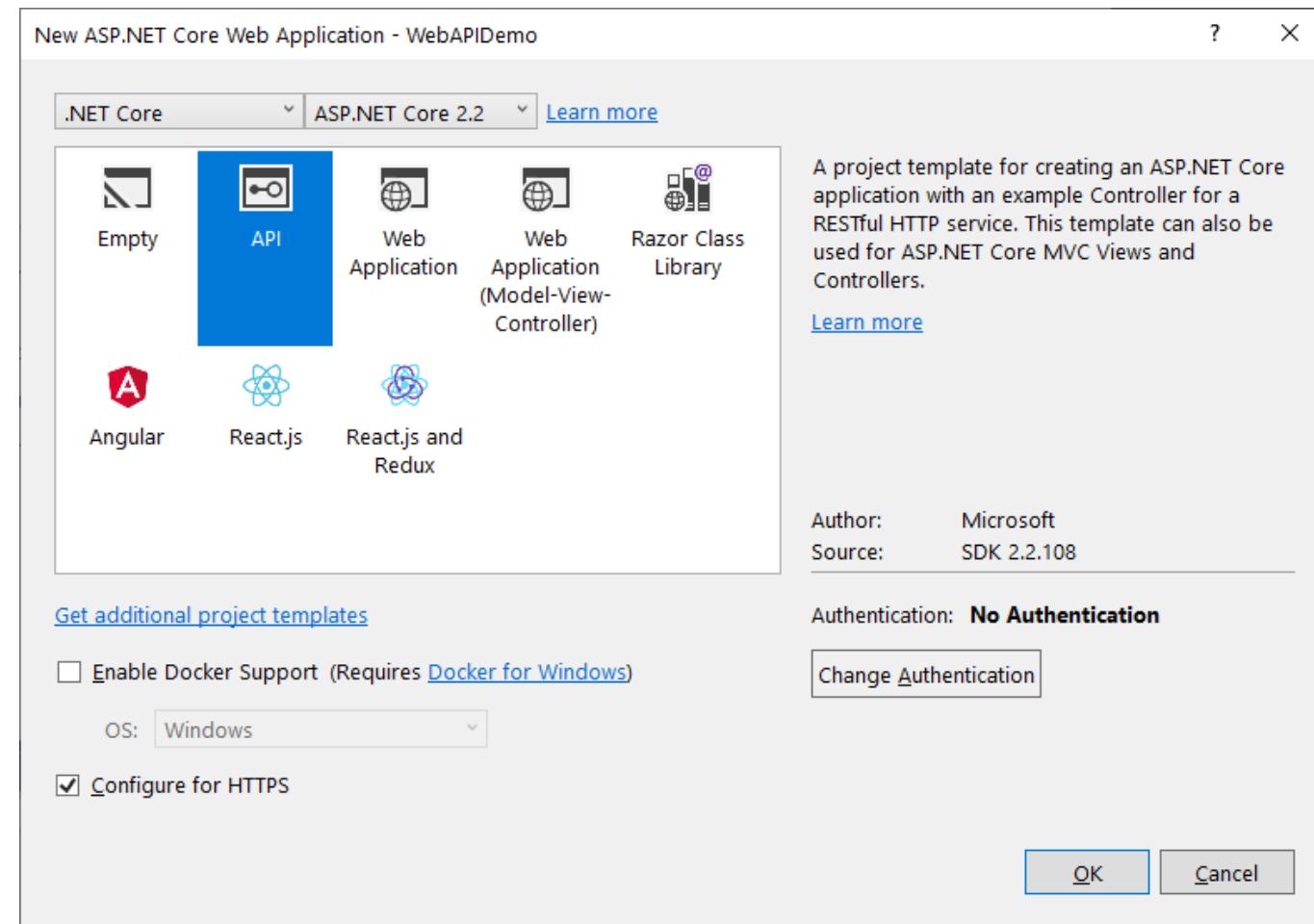
- Open File/New/Project in Visual Studio



<https://medium.com/net-core/how-to-build-a-restful-api-with-asp-net-core-fb7dd8d3e3>

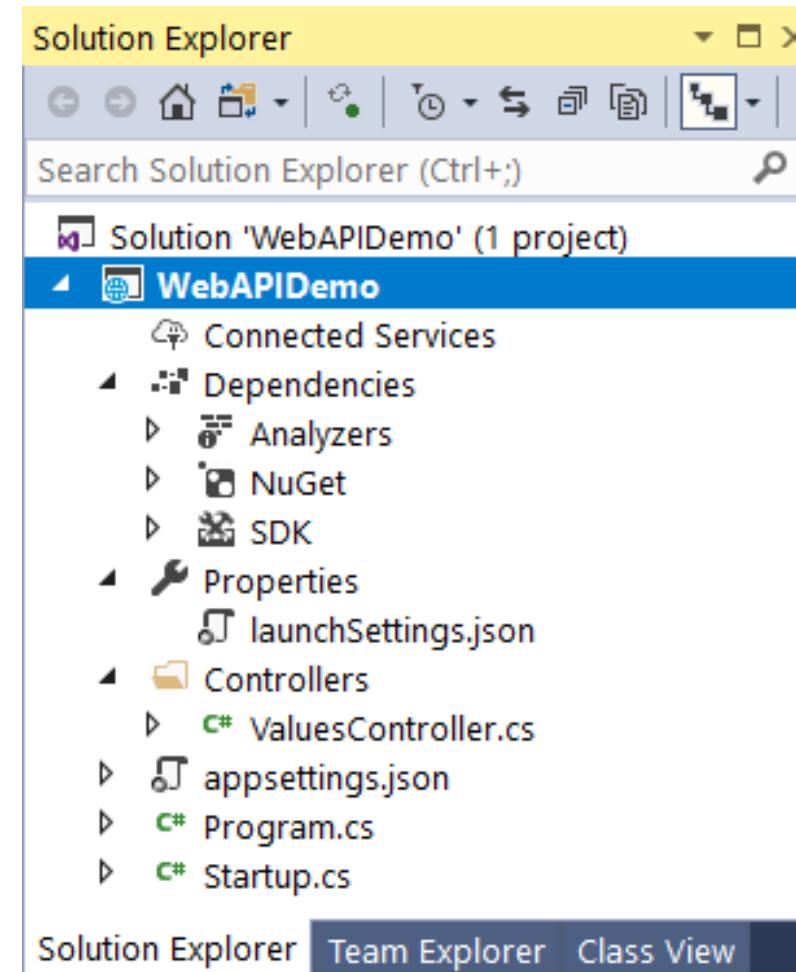
Tạo Web API project

- Chọn .NET Core, ASP.NET Core 2.2, API Project



Tạo Web API project

■ Project WebAPIDemo



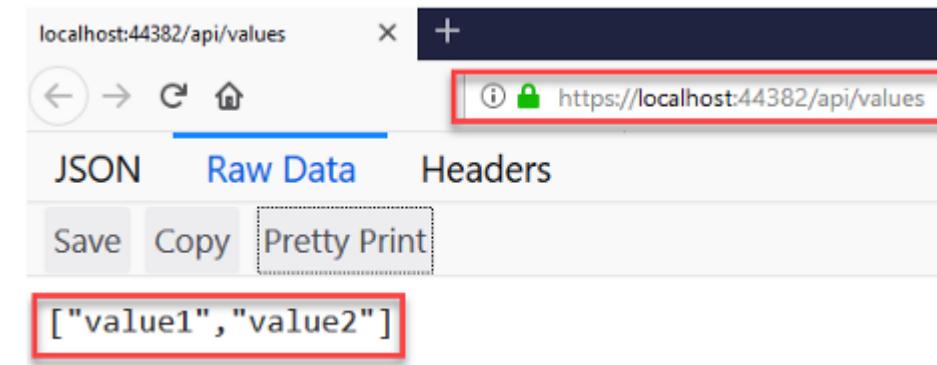
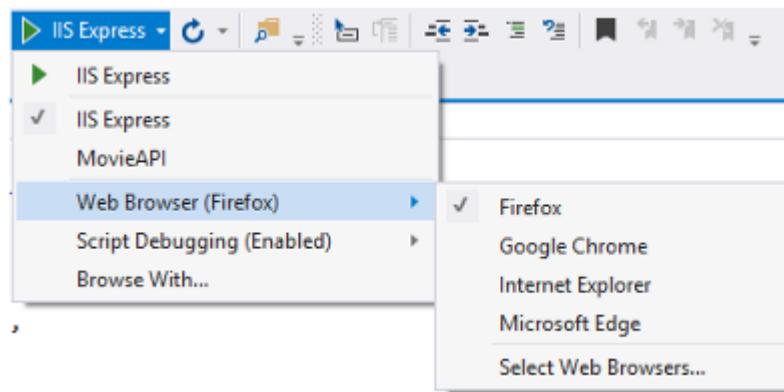
Tạo Web API project

- MVC (Model-View-Controller) được thêm trong Startup.cs

```
namespace WebAPIDemo
{
    2 references
    public class Startup
    {
        0 references | 0 exceptions
        public Startup(IConfiguration configuration)...
        1 reference | 0 exceptions
        public IConfiguration Configuration { get; }
        // This method gets called by the runtime. Use this method to add services to the container.
        0 references | 0 exceptions
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
        }
        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references | 0 exceptions
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())...
            else...
            app.UseHttpsRedirection();
            app.UseMvc();
        }
    }
}
```

Routing and URL paths

- Run app (Ctrl + F5), sử dụng IIS Express (MS webserver)
- Kết quả demo trên trình duyệt



Routing and URL paths

- URL mặc định `https://localhost:{port}/api/values` được thiết lập trong thuộc tính `launchUrl` trong file ***Properties\launchSettings.json***
- Values nhận được trên trình duyệt được mặc định trong phương thức Get của `ValuesController`

The image shows two code editors side-by-side. The left editor displays the `launchSettings.json` file, and the right editor displays the `ValuesController.cs` file.

launchSettings.json:

```
1 {  
2     "$schema": "http://json.schemastore.org/launchsettings.json",  
3     "iisSettings": {  
4         "windowsAuthentication": false,  
5         "anonymousAuthentication": true,  
6         "iisExpress": {  
7             "applicationUrl": "http://localhost:54911",  
8             "sslPort": 44382  
9         }  
10    },  
11    "profiles": {  
12        "IIS Express": {  
13            "commandName": "IISExpress",  
14            "launchBrowser": true,  
15            "launchUrl": "api/values",  
16            "environmentVariables": {  
17                "ASPNETCORE_ENVIRONMENT": "Development"  
18            }  
19        },  
20        "MovieAPI": {  
21            "commandName": "Project",  
22            "launchBrowser": true,  
23            "launchUrl": "api/values",  
24            "applicationUrl": "https://localhost:5001;http://localhost:5000",  
25            "environmentVariables": {  
26                "ASPNETCORE_ENVIRONMENT": "Development"  
27            }  
28        }  
29    }  
30}
```

ValuesController.cs:

```
[Route("api/[controller]")]  
[ApiController]  
public class ValuesController : ControllerBase  
{  
    // GET api/values  
    [HttpGet]  
    public ActionResult<IEnumerable<string>> Get()  
    {  
        return new string[] { "value1", "value2" };  
    }  
    ...  
}
```

Tạo một model

- Xây dựng data model class (**Model** trong kiến trúc MVC)
- In **Solution Explorer**, right-click the project. Select **Add > New Folder** and name the folder **Models**.
- Then right-click the **Models** folder and select **Add->Class**. Name the class **SensorData.cs** and click Add.

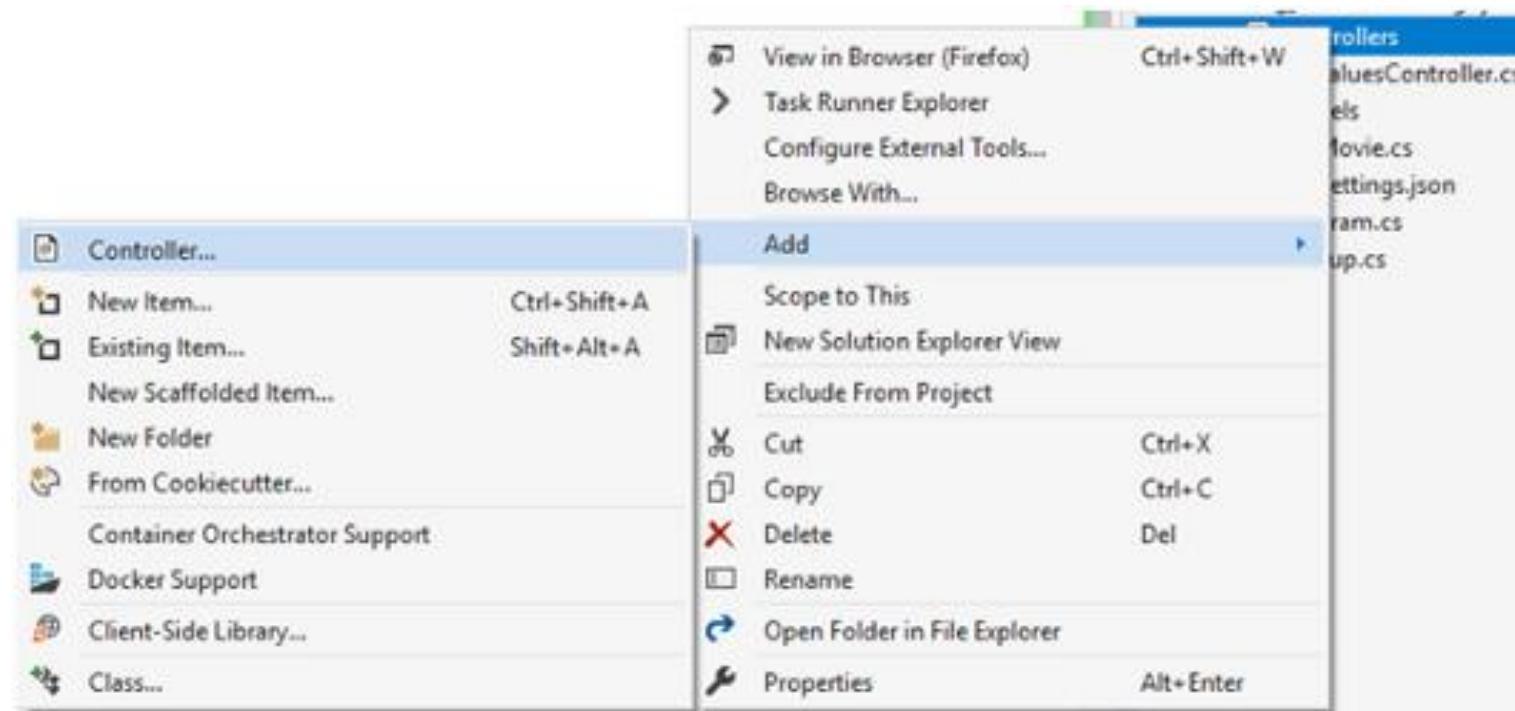
Tạo một model

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

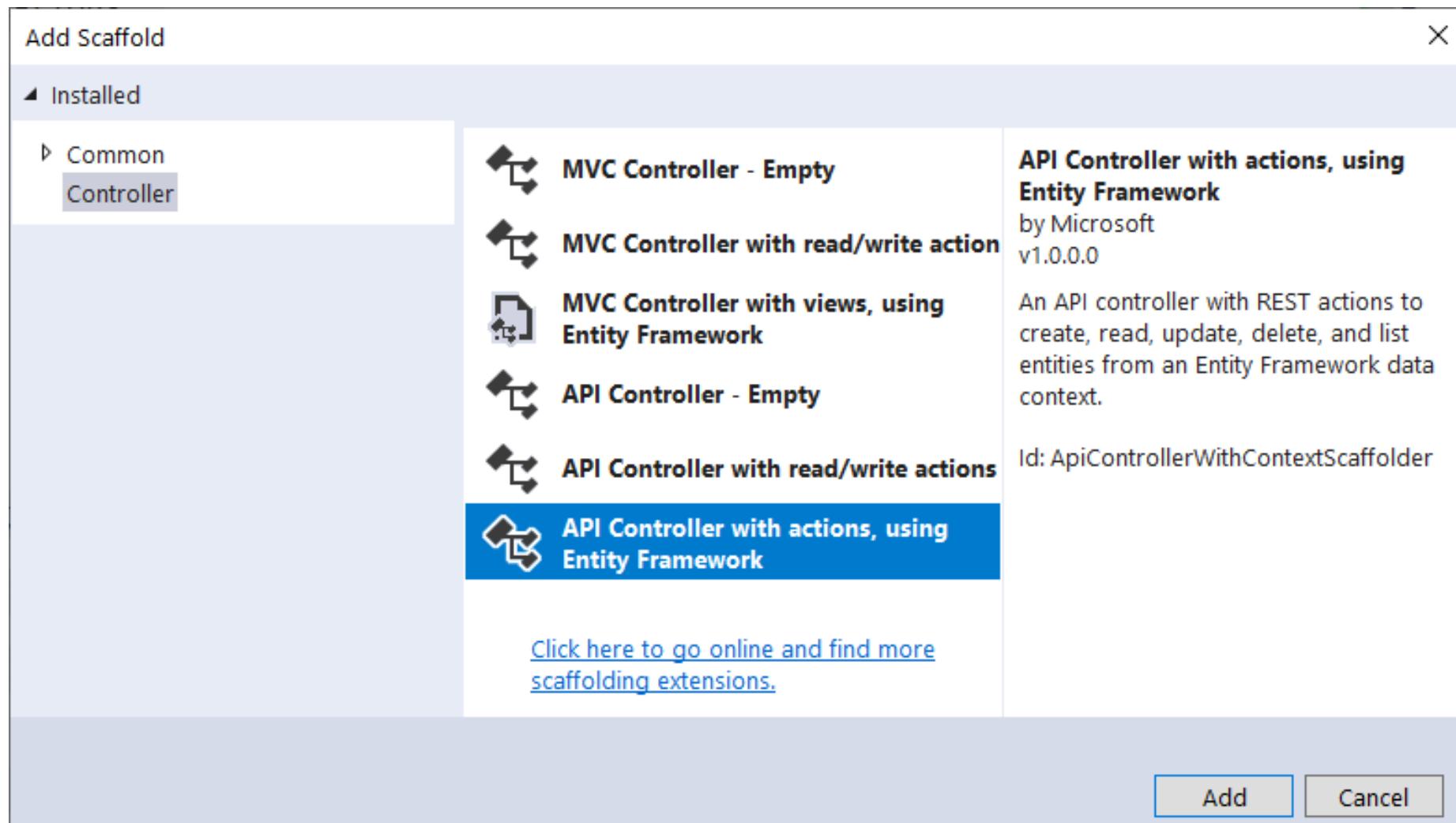
namespace WebAPIDemo.Models
{
    public class SensorData
    {
        public int Id { get; set; }
        [Required] //Data Annotation
        [StringLength(60, MinimumLength = 3)]
        public string Name { get; set; }
        public float Value { get; set; }
        [DataType(DataType.Date)]
        public DateTime ReceiveTime { get; set; }
    }
}
```

Tạo một controller

- Tạo một controller (thành phần Controller trong mô hình MVC), gắn với SensorData model
- Sử dụng Entity Framework Core (EF Core) để làm việc với database

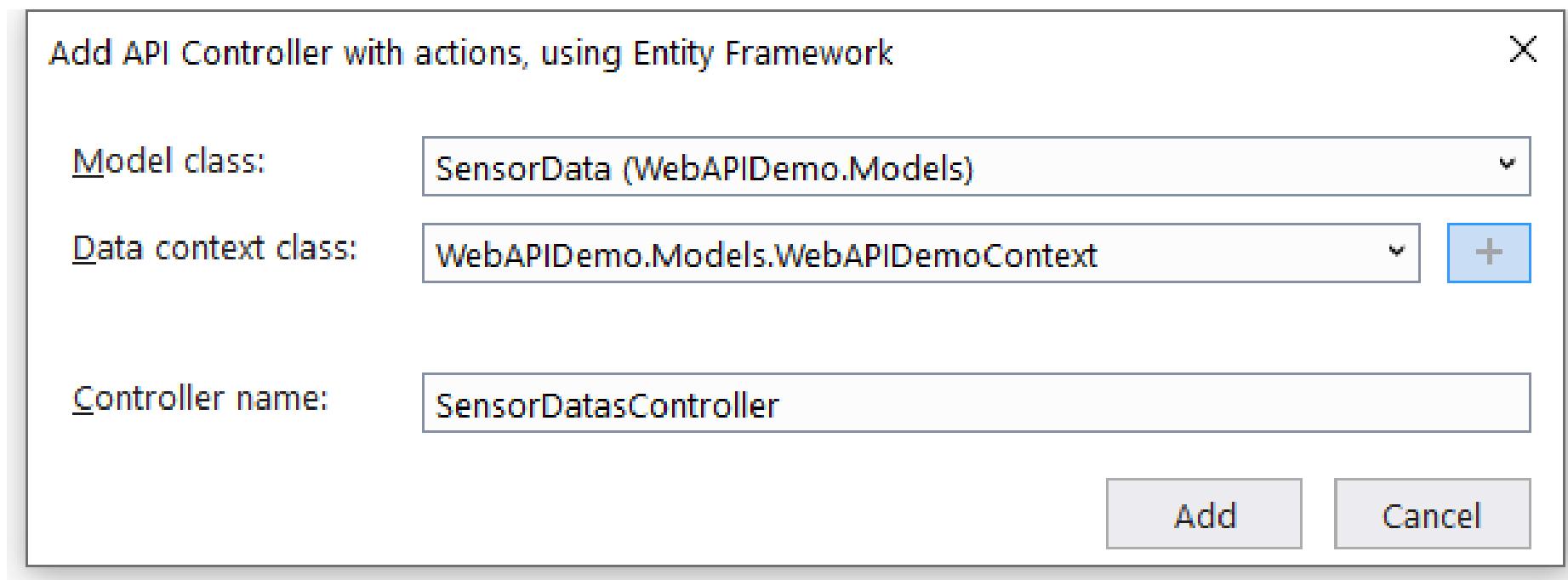


Tạo một controller



Tạo một controller

- Công cụ **scaffolding** của .NET Core hỗ trợ tự động sinh code
- The automatic creation of the database context and **CRUD (Create, Read, Update, and Delete)** action methods is known as scaffolding.



Controller Class

- Ví dụ SensorDatasController class với các methods:
 - GetSensorData (GET method)
 - PostSensorData (POST method)
 - PutSensorData (PUT method)
 - DeleteSensorData (DELETE method)
- Controller class kế thừa từ ControllerBase

```
public class SensorDatasController : ControllerBase
```

SensorDatasController.cs

```
namespace WebAPIDemo.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference | 0 requests
    public class SensorDatasController : ControllerBase
    {
        private readonly WebAPIDemoContext _context;
        0 references | 0 exceptions
        public SensorDatasController(WebAPIDemoContext context) ...
        // GET: api/SensorDatas
        [HttpGet]
        0 references | 0 requests | 0 exceptions
        public async Task<ActionResult<IEnumerable<SensorData>>> GetSensorData() ...
        // GET: api/SensorDatas/5
        [HttpGet("{id}")]
        0 references | 0 requests | 0 exceptions
        public async Task<ActionResult<SensorData>> GetSensorData(int id) ...
        // PUT: api/SensorDatas/5
        [HttpPut("{id}")]
        0 references | 0 requests | 0 exceptions
        public async Task<IActionResult> PutSensorData(int id, SensorData sensorData) ...
        // POST: api/SensorDatas
        [HttpPost]
        0 references | 0 requests | 0 exceptions
        public async Task<ActionResult<SensorData>> PostSensorData(SensorData sensorData) ...
        // DELETE: api/SensorDatas/5
        [HttpDelete("{id}")]
        0 references | 0 requests | 0 exceptions
        public async Task<ActionResult<SensorData>> DeleteSensorData(int id) ...
        1 reference | 0 exceptions
        private bool SensorDataExists(int id) ...
    }
}
```

EF Core Database Context

- *Data\WebAPIDemoContext.cs*
- *WebAPIDemoContext* kết nối với các chức năng của EF Core (Create, Read, Update, Delete, etc.) cho **SensorData** model
- *WebAPIDemoContext* kế thừa từ [Microsoft.EntityFrameworkCore.DbContext](#).

The screenshot shows a code editor window with the following code:

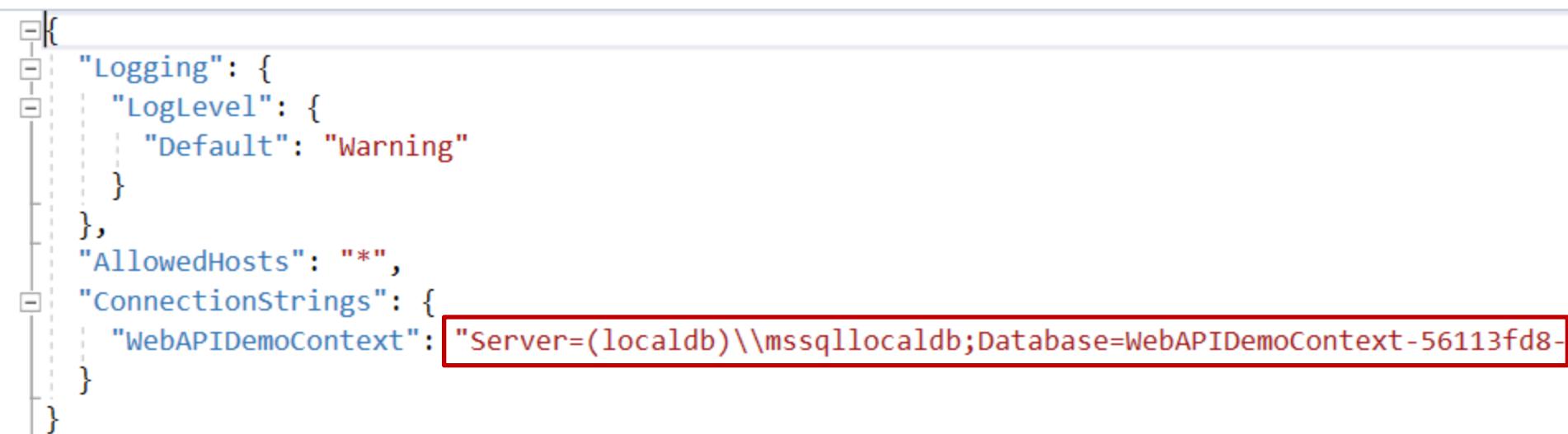
```
using Microsoft.EntityFrameworkCore;
namespace WebAPIDemo.Models
{
    public class WebAPIDemoContext : DbContext
    {
        public WebAPIDemoContext (DbContextOptions<WebAPIDemoContext> options) ...
        public DbSet<WebAPIDemo.Models.SensorData> SensorData { get; set; }
    }
}
```

The code editor highlights two parts of the code with red boxes:

- The class definition `public class WebAPIDemoContext : DbContext`
- The DbSet property `public DbSet<WebAPIDemo.Models.SensorData> SensorData { get; set; }`

EF Core Database Context

- In Entity Framework terminology, an entity set typically corresponds to a database table and an entity corresponds to a row in the table.
- ASP.NET Core configuration system reads the connection string from the *appsettings.json* file:



```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Warning"  
    }  
  },  
  "AllowedHosts": "*",  
  "ConnectionStrings": {  
    "WebAPIDemoContext": "Server=(localdb)\\mssqllocaldb;Database=WebAPIDemoContext-56113fd8-  
  }  
}
```

The screenshot shows a portion of the `appsettings.json` file. It includes settings for logging (LogLevel), allowed hosts, and connection strings. The connection string for "WebAPIDemoContext" is highlighted with a red border. The connection string value is `Server=(localdb)\\mssqllocaldb;Database=WebAPIDemoContext-56113fd8-`.

EF Core Database Context

- Đăng ký DB Context (WebAPIDemoContext) trong Startup.cs

```
// This method gets called by the runtime. Use this method to add services to the container.  
0 references | 0 exceptions  
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);  
  
    services.AddDbContext<WebAPIDemoContext>(options =>  
        options.UseSqlServer(Configuration.GetConnectionString("WebAPIDemoContext")));  
}
```

Inject the database context (WebAPIDemoContext) into the controller:

```
namespace WebAPIDemo.Controllers  
{  
    [Route("api/[controller]")]  
    [ApiController]  
    1 reference | 0 requests  
    public class SensorDatasController : ControllerBase  
    {  
        private readonly WebAPIDemoContext _context;  
        0 references | 0 exceptions  
        public SensorDatasController(WebAPIDemoContext context)  
        {  
            _context = context;  
        }  
    }  
}
```

Tạo database sử dụng Migrations

- Sử dụng Entity Framework code first
- Công cụ Migrations tạo (create) database tương ứng với data model được xây dựng (code first) và cập nhật (update) database schema khi có thay đổi của data model
- Chạy các lệnh của công cụ Migrations:
 - Mở **Tools -> NuGet Package Manager > Package Manager Console (PMC)**, thực hiện **command:**
 - **Add-Migration Initial (Tạo database)**
 - **Update-Database (Cập nhật database schema)**

Tạo database sử dụng Migrations

- Migration files được sinh tự động trong Migration folder

```
namespace WebAPIDemo.Migrations
{
    public partial class Initial : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "SensorData",
                columns: table => new
                {
                    Id = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy", SqlServerValueGenerationStrategy.IdentityColumn),
                    Name = table.Column<string>(maxLength: 60, nullable: false),
                    Value = table.Column<float>(nullable: false),
                    ReceiveTime = table.Column<DateTime>(nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_SensorData", x => x.Id);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "SensorData");
        }
    }
}
```

Tạo database sử dụng Migrations

- Chạy lệnh `Update-Database` (trong PMC)
- Phương thức `Up` trong file `Migrations/{time-stamp}_Initial.cs` sẽ được thực thi và tạo database
- Mở SQL Server Object Explorer

dbo._EFMigrationsHistory [Data]		
	MigrationId	ProductVersion
	20210408115417_Initial	2.2.6-servicing-10079
▶*	NULL	NULL

The screenshot shows the SQL Server Object Explorer window. The tree view on the left lists the database structure:

- SQL Server
- (localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - DESKTC)
 - Databases
 - System Databases
 - ContosoUniversity1
 - Microsoft.VsCodeIndex
 - WebAPIDemoContext-DB (selected)
 - Tables
 - dbo._EFMigrationsHistory
 - dbo.SensorData (selected)
 - Columns
 - Id (PK, int, not null)
 - Name (nvarchar(60), not null)
 - Value (real, not null)
 - ReceiveTime (datetime2(7), not null)
 - Keys

A red box highlights the `dbo.SensorData` table and its columns.

Tạo database

- Thêm dữ liệu vào bảng SensorData
- Mở **SQL Server Object Explorer**, chuột phải vào bảng SensorData, chọn **View Data** để thêm bản ghi

	Id	Name	Value	ReceiveTime
	1	Temperature	37	4/8/2021 11:1...
	2	Humidity	66	4/8/2021 10:3...
	4	PM2.5	123	4/9/2021 12:0...
	6	PM10	345	4/9/2021 1:20...
	NULL	NULL	NULL	NULL

GET method

- GetSensorData(): trả về tất cả bản ghi trong SensorData database
- GetSensorData(int id): trả về bản ghi Id tương ứng
- [HttpGet] phương thức xử lý khi có HTTP GET request

```
// GET: api/SensorDatas
[HttpGet]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult<IEnumerable<SensorData>>> GetSensorData()
{
    return await _context.SensorData.ToListAsync();
}

// GET: api/SensorDatas/5
[HttpGet("{id}")]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult<SensorData>> GetSensorData(int id)
{
    var sensorData = await _context.SensorData.FindAsync(id);

    if (sensorData == null)
    {
        return NotFound();
    }

    return sensorData;
}
```

GET method

- GET endpoints:
 - GET /api/SensorData
 - GET /api/SensorData/{id}
- Gọi HTTP GET request từ trình duyệt:
 - `https://localhost:{port}/api/SensorDatas`
 - `https://localhost:{port}/api/SensorDatas/2`
- Kiểu giá trị trả về **ActionResult<T> type**
 - .NET Core tự động serializes object thành JSON và ghi vào body của gói tin response
 - Response code: **200** (nếu không có unhandled exceptions), nếu có sẽ trả về mã lỗi **5xx**

GET method

- Kiểm thử trên trình duyệt

The image displays two side-by-side browser windows illustrating API responses for sensor data.

Left Browser Window: The address bar shows "localhost:44364/api/SensorDatas". The response is a JSON array containing two objects, each representing a sensor data point. The first object has an id of 1, name "Temperature", value 37.0, and receiveTime "2021-04-08T11:16:20". The second object has an id of 2, name "Humidity", value 66.0, and receiveTime "2021-04-08T10:30:01".

```
// 20210409113750
// https://localhost:44364/api/SensorDatas

[
  {
    "id": 1,
    "name": "Temperature",
    "value": 37.0,
    "receiveTime": "2021-04-08T11:16:20"
  },
  {
    "id": 2,
    "name": "Humidity",
    "value": 66.0,
    "receiveTime": "2021-04-08T10:30:01"
  }
]
```

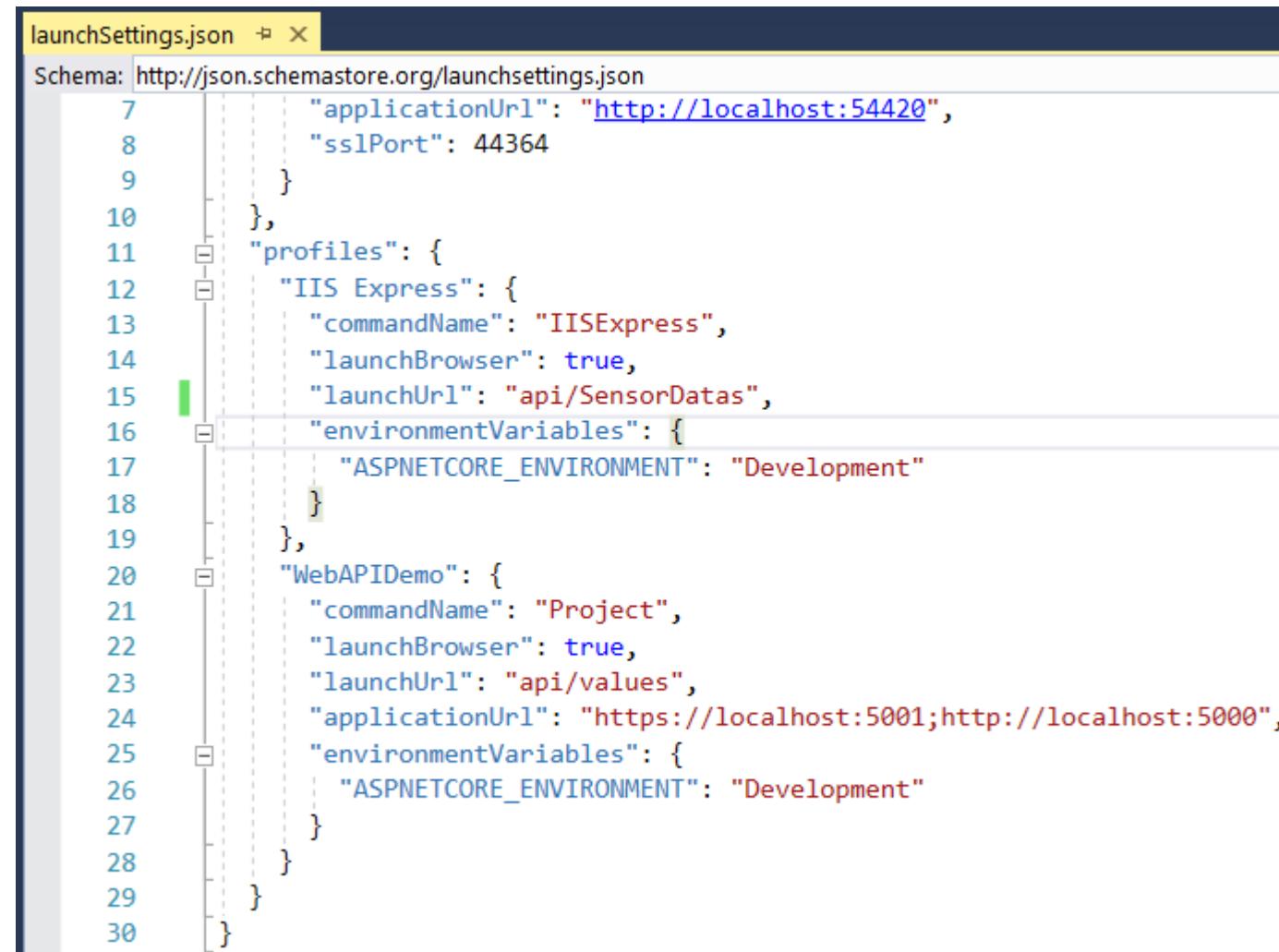
Right Browser Window: The address bar shows "localhost:44364/api/SensorDatas/2". The response is a single JSON object representing the second sensor data point. It has an id of 2, name "Humidity", value 66.0, and receiveTime "2021-04-08T10:30:01".

```
// 20210409113838
// https://localhost:44364/api/SensorDatas/2

{
  "id": 2,
  "name": "Humidity",
  "value": 66.0,
  "receiveTime": "2021-04-08T10:30:01"
}
```

GET method

- URL mặc định: Cấu hình launchUrl trong file launchsettings.json



The screenshot shows the Visual Studio code editor with the file "launchSettings.json" open. The title bar says "launchSettings.json". The schema is defined as "Schema: http://json.schemastore.org/launchsettings.json". The JSON code is as follows:

```
7     "applicationUrl": "http://localhost:54420",
8     "sslPort": 44364
9   }
10  },
11  "profiles": {
12    "IIS Express": {
13      "commandName": "IISExpress",
14      "launchBrowser": true,
15      "launchUrl": "api/SensorDatas",
16      "environmentVariables": {
17        "ASPNETCORE_ENVIRONMENT": "Development"
18      }
19    },
20    "WebAPIDemo": {
21      "commandName": "Project",
22      "launchBrowser": true,
23      "launchUrl": "api/values",
24      "applicationUrl": "https://localhost:5001;http://localhost:5000",
25      "environmentVariables": {
26        "ASPNETCORE_ENVIRONMENT": "Development"
27      }
28    }
29  }
30 }
```

POST method

- Hàm PostSensorData tạo một bản ghi mới trong database
- Dữ liệu (bản ghi) được gửi trong body của HTTP POST request
- Hàm CreatedAtAction:
 - Trả về mã HTTP 201 nếu thành công

```
// POST: api/SensorDatas
[HttpPost]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult<SensorData>> PostSensorData(SensorData sensorData)
{
    _context.SensorData.Add(sensorData);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetSensorData", new { id = sensorData.Id }, sensorData);
}
```

Kiểm thử dùng Postman

- GET method: get all resource

The screenshot shows the Postman application interface. At the top, there is a header with 'GET' selected, the URL 'https://localhost:44364/api/SensorDatas', a 'Send' button, and a 'Save' button. Below the header, the 'Params' section is expanded, showing a 'Query Params' table with columns: KEY, VALUE, DESCRIPTION, and Bulk Edit. The table has one row with 'Key' as 'Value' and 'Description' as 'Description'. To the right of the table, the 'Body' section shows a response with status '200 OK', time '188 ms', size '692 B', and a 'Save Response' button. The response body is displayed in a code editor with line numbers from 1 to 13, showing two sensor data objects in JSON format.

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

```
1  [
2   {
3     "id": 1,
4     "name": "Temperature",
5     "value": 37.0,
6     "receiveTime": "2021-04-08T11:16:20"
7   },
8   {
9     "id": 2,
10    "name": "Humidity",
11    "value": 66.0,
12    "receiveTime": "2021-04-08T10:30:01"
13  },
```

Kiểm thử dùng Postman

- GET method: get 1 resource by Id

The screenshot shows the Postman application interface. At the top, there is a header bar with a 'GET' button, a URL input field containing 'https://localhost:44364/api/SensorDatas/2', a 'Send' button, and a 'Save' button. Below the header, there are two main sections: 'Params' and 'Body'. The 'Params' section is expanded, showing a 'Query Params' table with columns: KEY, VALUE, DESCRIPTION, and Bulk Edit. The table has one row with 'Key' as 'id' and 'Value' as '2'. The 'Body' section is also expanded, showing a 'Pretty' JSON view of the response. The response status is '200 OK' with a 114 ms response time and 394 B size. The JSON data is:

```
1 {  
2     "id": 2,  
3     "name": "Humidity",  
4     "value": 66.0,  
5     "receiveTime": "2021-04-08T10:30:01"  
6 }
```

Kiểm thử dùng Postman

■ POST method

The screenshot shows the Postman interface with a successful POST request to `https://localhost:44364/api/SensorDatas`.

Request Headers:

- Method: POST
- URL: `https://localhost:44364/api/SensorDatas`
- Status: 201 Created
- Time: 1339 ms
- Size: 446 B

Request Body (raw JSON):

```
1 {  
2   "name": "Light",  
3   "value": 203,  
4   "receiveTime":  
5     "2021-04-10T12:17:20"  
6 }
```

Response Body (Pretty JSON):

```
1 {  
2   "id": 7,  
3   "name": "Light",  
4   "value": 203.0,  
5   "receiveTime": "2021-04-10T12:17:20"  
6 }
```

PUT method

- Phương thức PutSensorData() cập nhật một bản ghi

```
// PUT: api/SensorDatas/5
[HttpPut("{id}")]
0 references | 0 requests | 0 exceptions
public async Task<IActionResult> PutSensorData(int id, SensorData sensorData)
{
    if (id != sensorData.Id)
    {
        return BadRequest();
    }
    _context.Entry(sensorData).State = EntityState.Modified;
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!SensorDataExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return NoContent();
}
```

PUT method

- Mã trả về: HTTP 204 (No Content)

The screenshot shows the POSTMAN API client interface. At the top, there is a header bar with 'PUT' selected as the method, the URL 'https://localhost:44364/api/SensorDatas/7', a 'Send' button, and a 'Save' button.

The main area is divided into two sections: 'Body' and 'Response'.

Body Section: Contains tabs for 'raw', 'JSON', and 'Beautify'. The 'JSON' tab is selected, showing the following JSON payload:

```
1 {  
2   "id":7,  
3   "name": "Light",  
4   "value": 111,  
5   "receiveTime":  
6     "2021-04-10T12:17:20"  
7 }
```

Response Section: Shows the status '204 No Content' with a globe icon, response time '1137 ms', and size '252 B'. There are also 'Save Response' and download/copy/share buttons.

DELETE method

- Xóa một bản ghi (by Id)

```
// DELETE: api/SensorDatas/5
[HttpDelete("{id}")]
0 references | 0 requests | 0 exceptions
public async Task<ActionResult<SensorData>> DeleteSensorData(int id)
{
    var sensorData = await _context.SensorData.FindAsync(id);
    if (sensorData == null)
    {
        return NotFound();
    }

    _context.SensorData.Remove(sensorData);
    await _context.SaveChangesAsync();

    return sensorData;
}
```

DELETE method

- Xóa một bản ghi (by Id)
- Mã trả về HTTP OK 200 (xóa thành công)

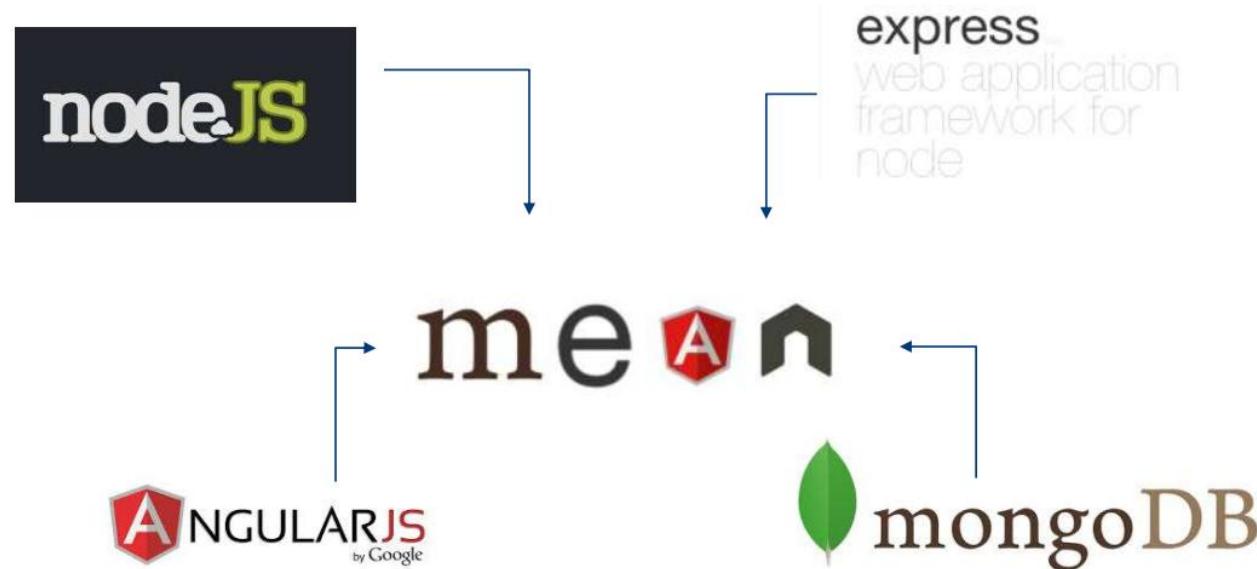
The screenshot shows a REST client interface with the following details:

- Method:** DELETE (highlighted with a red box)
- URL:** https://localhost:44364/api/SensorDatas/7 (highlighted with a red box)
- Send** button (blue)
- Save** button (grey)
- Body** tab (selected)
- raw** (disabled)
- JSON** (selected)
- Beautify** (disabled)
- Response Headers:** 200 OK, 194 ms, 392 B, Save Response
- Body Content:**

```
1 {  
2   "id": 7,  
3   "name": "Light",  
4   "value": 111.0,  
5   "receiveTime": "2021-04-10T12:17:20"  
6 }
```

3.2.2. Công nghệ NodeJS, MongoDB

- MEAN stack (mongoDB, ExpressJS, AngularJS, NodeJS)



<https://viblo.asia/p/xay-dung-restful-api-don-gian-voi-nodejs-1Je5EdewlnL>

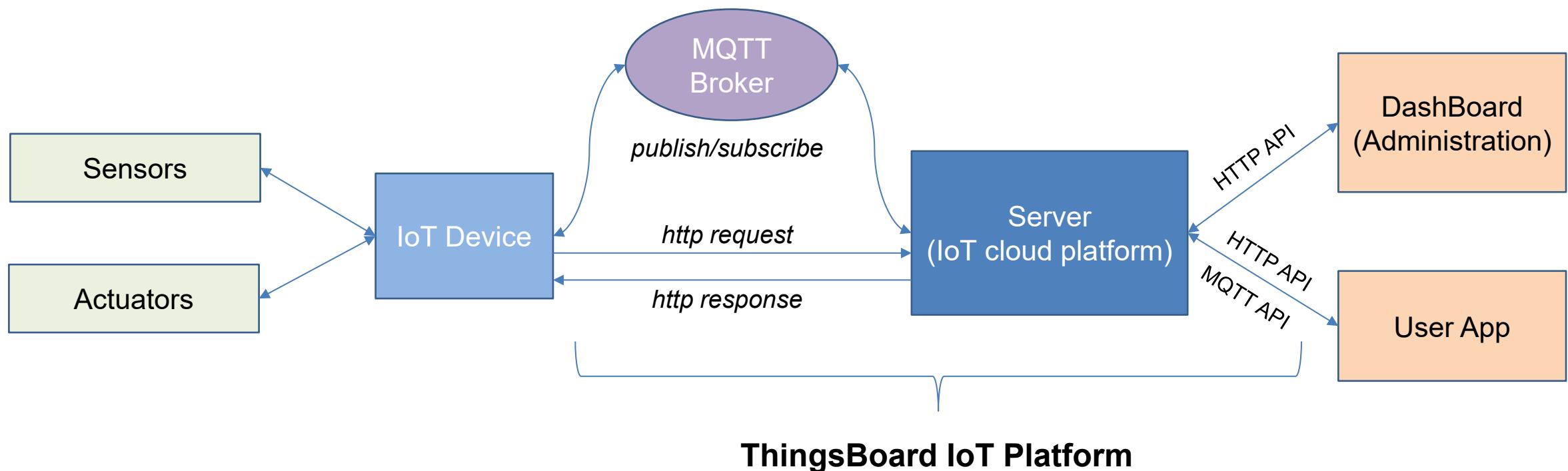
<https://viblo.asia/p/tao-rest-api-voi-spring-boot-mysql-jpa-repository-Eb85oJykl2G>

3.3. Lập trình với IoT Platform

- Các dịch vụ:
 - Tiếp nhận dữ liệu (Collecting, Ingesting data)
 - Lưu trữ dữ liệu (Cloud Storage, Database)
 - Trình bày dữ liệu (Data Representation),
 - Theo dõi giám sát, điều khiển (Dashboard)
- Thingsboard - open-source IoT platform for data collection, processing, visualization, and device management
 - <https://thingsboard.io/>

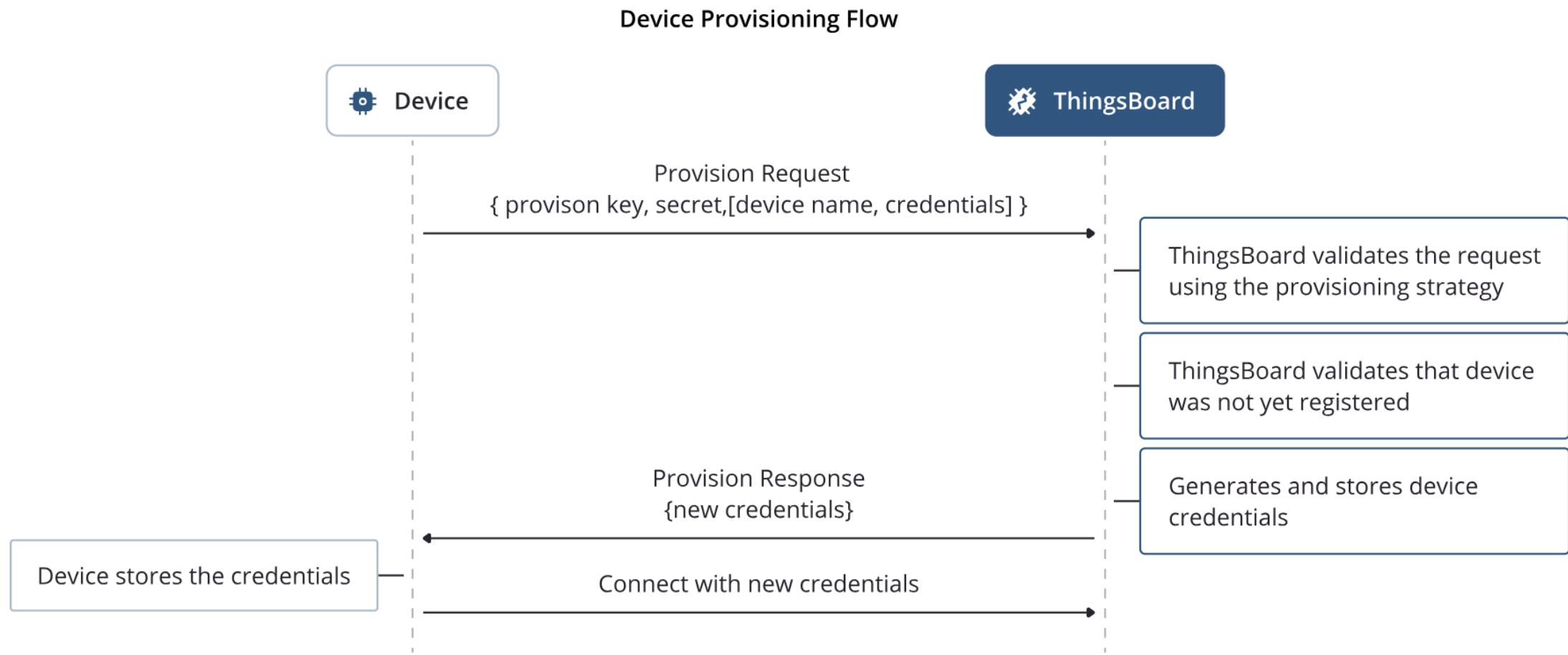
Case study: Hệ thống kiểm soát vào ra

- Kiến trúc hệ thống: Khai thác ThingsBoard IoT platform



3.3.1. Đăng ký thiết bị

- Device Provisioning: Thiết bị thực hiện đăng ký qua MQTT API



Đăng ký thiết bị

- Device Provisioning: Thiết bị thực hiện đăng ký qua MQTT API

<https://thingsboard.io/docs/user-guide/device-provisioning/>

Đăng ký thiết bị

- Step 1. Thiết bị kết nối tới ThingsBoard MQTT broker, sử dụng username = “provision”
- Step 2. Thiết bị gửi provision request lên server bằng cách publish lên topic /provision/request

```
{  
    "deviceName": "DEVICE_NAME",  
    "provisionDeviceKey": "PUT_PROVISION_KEY_HERE",  
    "provisionDeviceSecret": "PUT_PROVISION_SECRET_HERE"  
}
```

Đăng ký thiết bị

- provisionDeviceKey và provisionDeviceSecret được tạo trên ThingsBoard tương ứng với Tenant (dự án)

The screenshot shows the ThingsBoard Cloud interface for managing device profiles. The top navigation bar includes 'Profiles' > 'Device profiles' > 'BKAC device profile'. The right side of the header displays 'Current subscription: ThingsBoard Cloud Maker' and 'Status: Trial ends on the Apr 11, 2025'. The left sidebar lists various management sections: Home, Plan and billing, Alarms, Dashboards, Solution templates (marked as NEW), Entities, Devices, Assets, Entity views, Gateways, Profiles, and Device profiles (which is currently selected).

The main content area is titled 'BKAC device profile' and 'Device profile details'. It features a tab navigation with 'Details', 'Transport configuration', 'Alarm rules (0)', and 'Device provisioning' (which is underlined, indicating it is active). Under 'Device provisioning', there is a section for 'Provision strategy*' set to 'Allow to create new devices'. Below this are fields for 'Provision device key*' containing the value 'okg9ywrkcsc7cm2qy9hg' and 'Provision device secret*' containing the value 'oy7e0weytuo5cos385fd'. There are also small circular icons with checkmarks and X's in the top right corner of the main content area.

Đăng ký thiết bị

- Step 3. Thiết bị nhận phản hồi của server bằng cách subscribe topic /provision/response

```
{  
    "status": "SUCCESS",  
    "credentialsType": "ACCESS_TOKEN",  
    "credentialsValue": "sLzc0gDAZPkGMzFVTyUY"  
}
```

Kết quả: Thiết bị nhận được ACCESS_TOKEN sẽ dùng để xác thực cho các yêu cầu giao tiếp tiếp theo

Đăng ký thiết bị

- **Bài tập 1.** Viết chương trình (java, python) thực hiện đăng ký một thiết bị (IoT device) với ThingsBoard IoT platform
- <https://thingsboard.io/docs/user-guide/device-provisioning/>

3.3.2. Gửi dữ liệu từ thiết bị IoT lên server (ThingsBoard)

- **(1) Gửi dữ liệu bằng MQTT API**
 - Thiết bị thực hiện gửi dữ liệu thông qua MQTT Device API
 - Tài liệu: <https://thingsboard.io/docs/reference/mqtt-api/>
- **(2) Gửi dữ liệu bằng HTTP API**
 - Thiết bị thực hiện gửi dữ liệu thông qua HTTP Device API
 - Tài liệu: <https://thingsboard.io/docs/reference/http-api/>

Gửi dữ liệu bằng MQTT API

- Thiết bị kết nối với ThingsBoard MQTT broker
 - Sử dụng username = \$ACCESS_TOKEN
- Thiết bị gửi dữ liệu (telemetry data) lên server bằng cách publish gói dữ liệu lên topic
 - v1/devices/me/telemetry
- Dữ liệu được đóng gói bằng json data
 - Ví dụ:

```
{"ts":1451649600512, "values": {"key1":"value1", "key2":"value2"} }
```

Gửi dữ liệu bằng MQTT API

- **Bài tập 2.** Viết chương trình (Java, python) thực hiện gửi dữ liệu (json) từ thiết bị lên ThingsBoard server sử dụng MQTT

Gửi dữ liệu bằng HTTP API

- Telemetry upload API:
 - Thiết bị gửi POST request đến url:
 - `http(s)://$THINGSBOARD_HOST_NAME/api/v1/$ACCESS_TOKEN/telemetry`
 - `$ACCESS_TOKEN` của thiết bị thực hiện gửi

Gửi dữ liệu bằng HTTP API

- **Bài tập 3.** Viết chương trình (Java, python) thực hiện gửi dữ liệu từ thiết bị lên ThingsBoard server qua HTTP API

Nội dung

- Chương 1. Tổng quan về IoT
- Chương 2. Các công nghệ IoT
- Chương 3. Lập trình ứng dụng IoT
- Chương 4. An toàn và Bảo mật IoT
- Chương 5. Thiết kế và xây dựng hệ thống IoT

Chương 4. An toàn và Bảo mật IoT

- 4.1. Tổng quan về bảo mật IoT
- 4.2. Các dạng tấn công vào hạ tầng IoT
- 4.3. Các điểm yếu bảo mật trong IoT

4.1. Tổng quan về bảo mật IoT

- Các vấn đề trong bảo mật IoT:
 - Thiết kế ban đầu cho mạng truyền thông riêng sau đó được chuyển sang mạng IP và Internet
 - Cập nhật firmware cho thiết bị IoT khó khăn
 - Xuất phát từ những yêu cầu bảo mật cơ bản, sau đó xuất hiện các lỗi bảo mật kèm theo các yêu cầu bảo mật phức tạp hơn.
 - Các thiết bị bảo mật kém từ các thiết kế ban đầu đã được sử dụng trên thực tế

Tổng quan về bảo mật IoT

- Phân loại nguy cơ trong bảo mật IoT:
 - **Capture:** Các nguy cơ liên quan đến “bắt” (thu thập, ăn cắp) thông tin dữ liệu từ hệ thống
 - **Disrupt:** Các nguy cơ liên quan đến tấn công từ chối dịch vụ (denying), phá hủy (destroying), ngắt/dừng (interrupting) hệ thống
 - **Manipulate:** Các nguy cơ liên quan đến can thiệp/thay đổi (manipulating) dữ liệu, định danh.

Tổng quan về bảo mật IoT

- Các yêu cầu bảo mật IoT:
 - Confidentiality – Tính tin cẩn:
 - Dữ liệu truyền chỉ có thể được đọc bởi bên nhận
 - Availability – Tính sẵn dùng:
 - Việc truyền thông giữa các thiết bị truyền và nhận luôn luôn sẵn sàng
 - Integrity – Tính toàn vẹn
 - Dữ liệu nhận không bị nhiễu trong quá trình truyền, đảm bảo tính chính xác, vẹn toàn của dữ liệu
 - Authenticity – Tính xác thực
 - Bên gửi luôn luôn có thể xác thực dữ liệu được gửi
 - Dữ liệu chỉ có thể được truy cập bởi bên nhận được cho phép

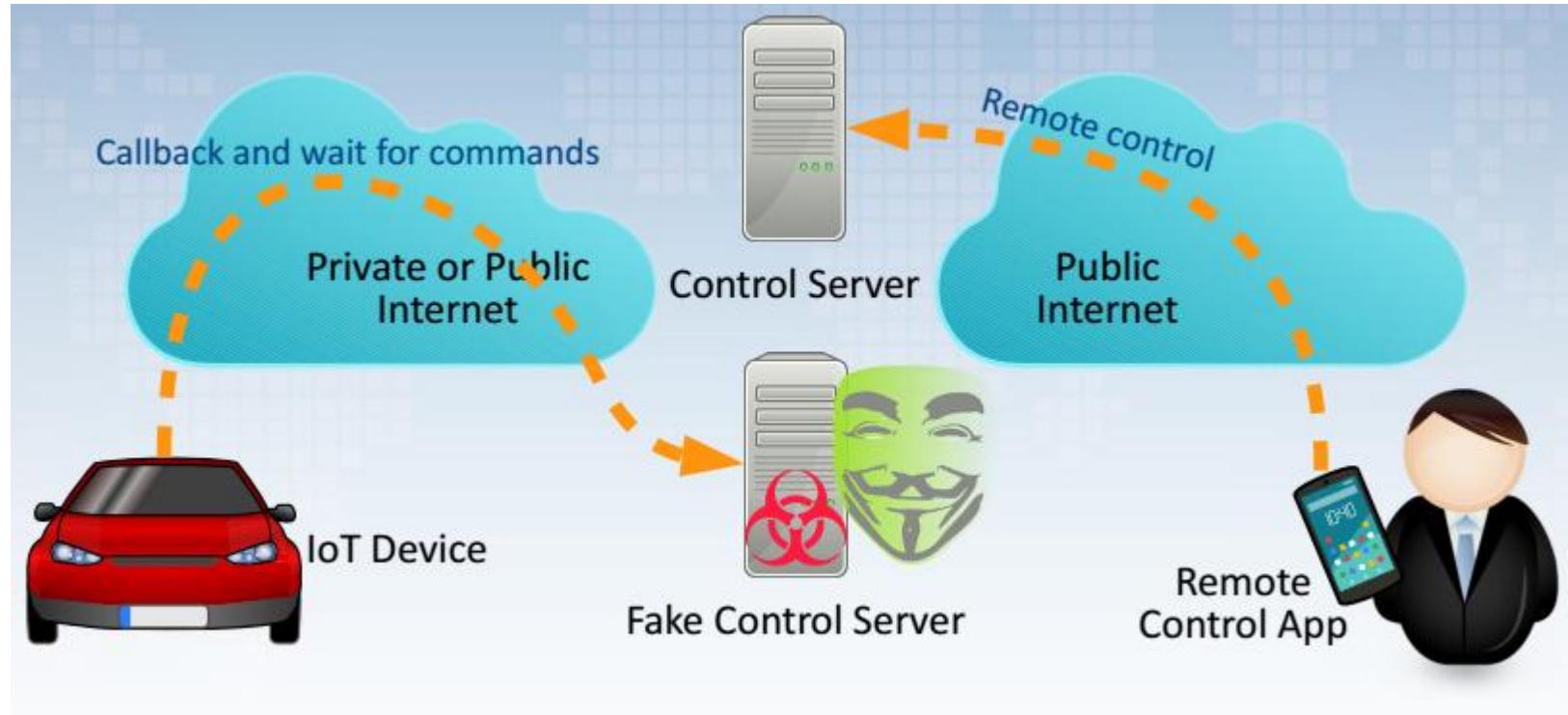
4.2. Các dạng tấn công hạ tầng IoT

- Hạ tầng IoT thông dụng



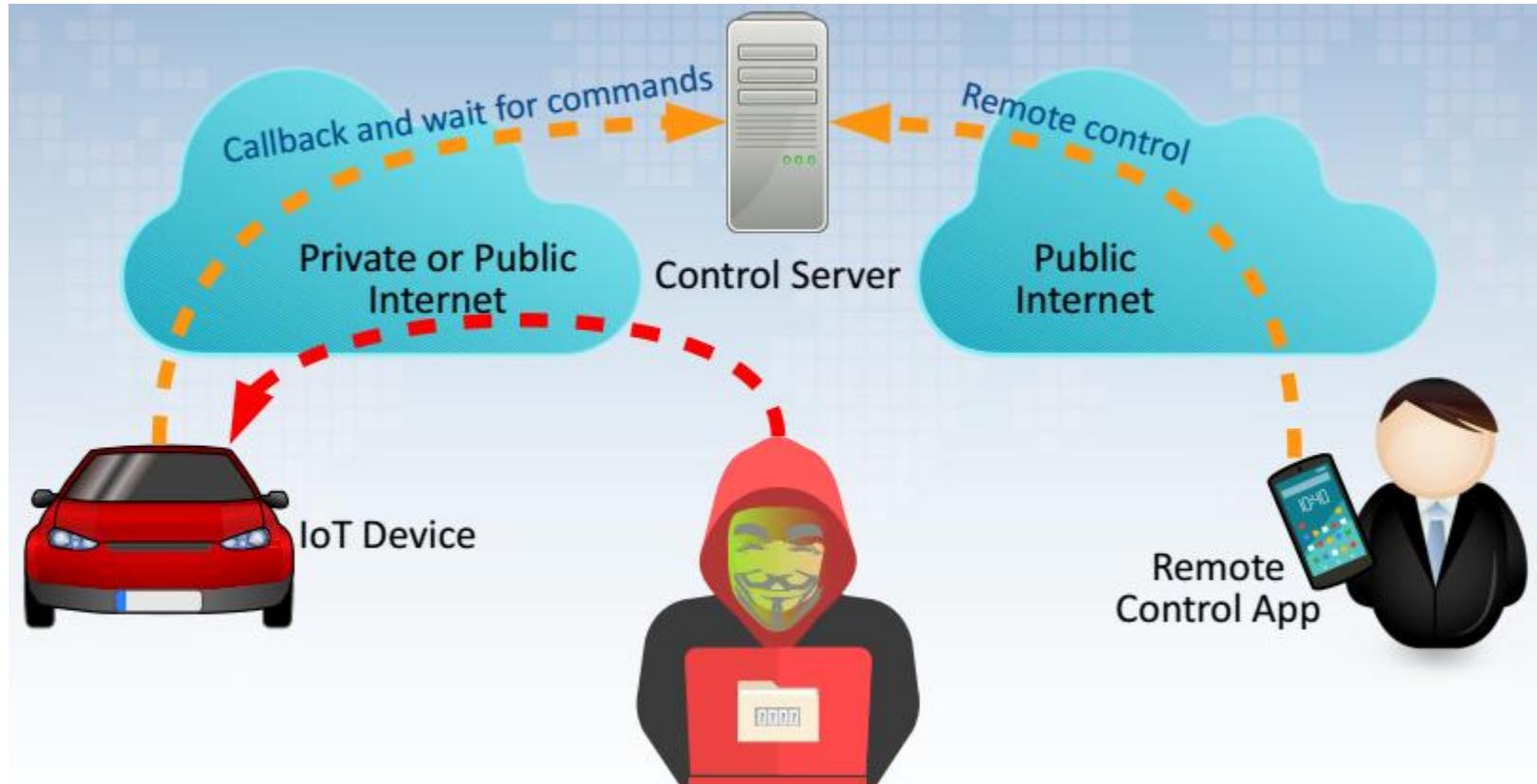
Một số dạng tấn công

- Fake Control Server



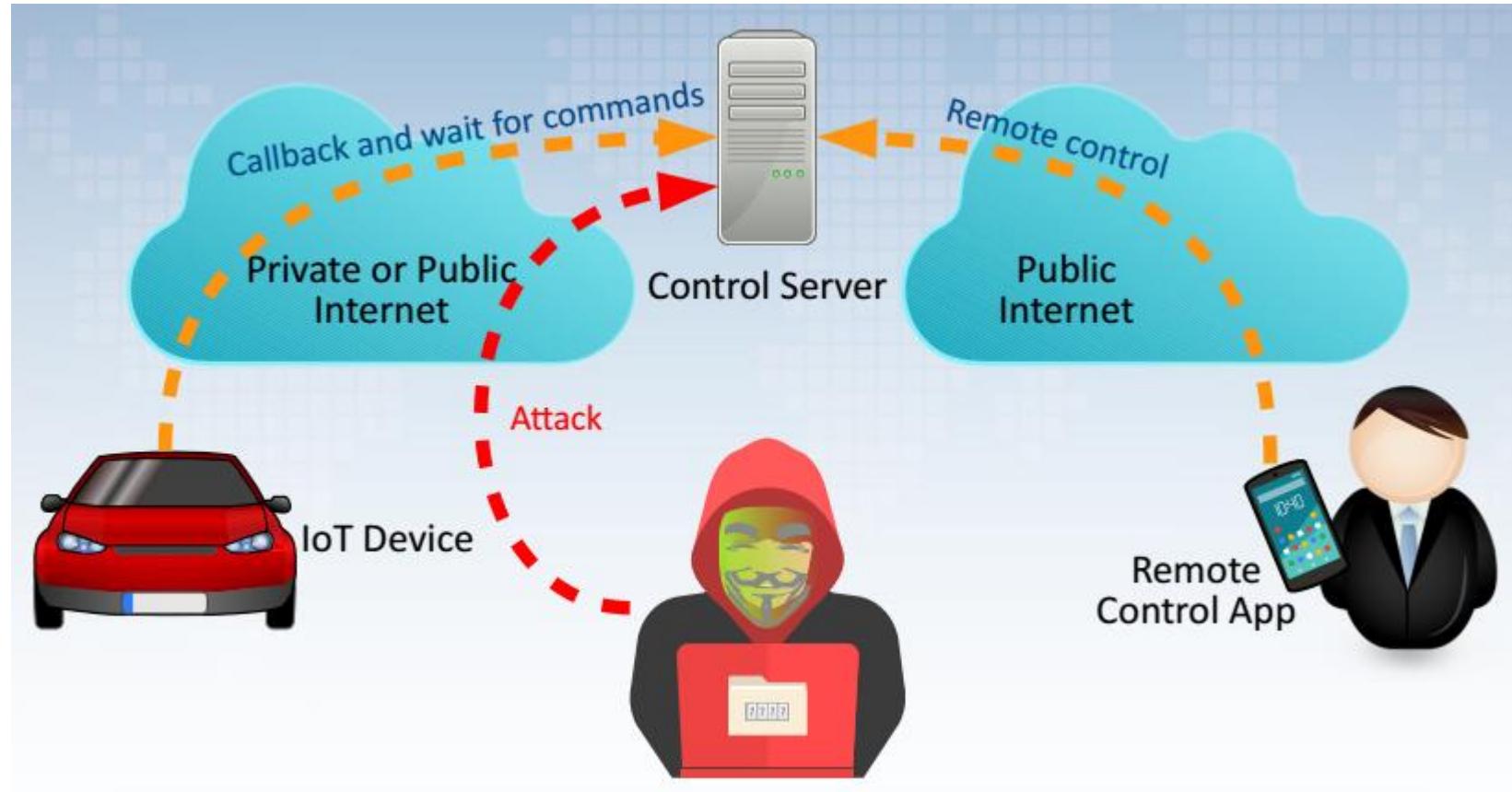
Một số dạng tấn công

- Attack on device open ports



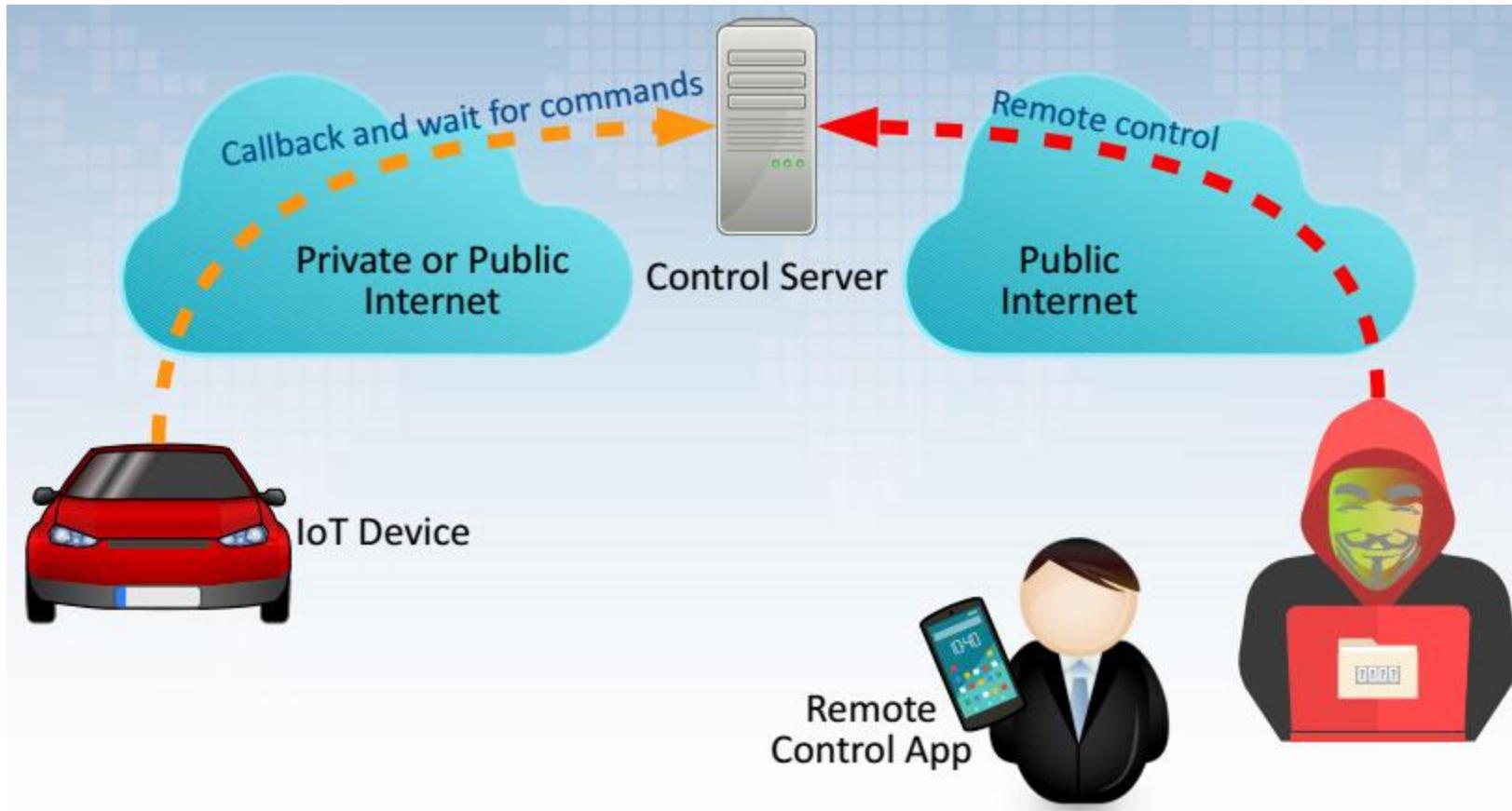
Một số dạng tấn công

- Attack on server open ports



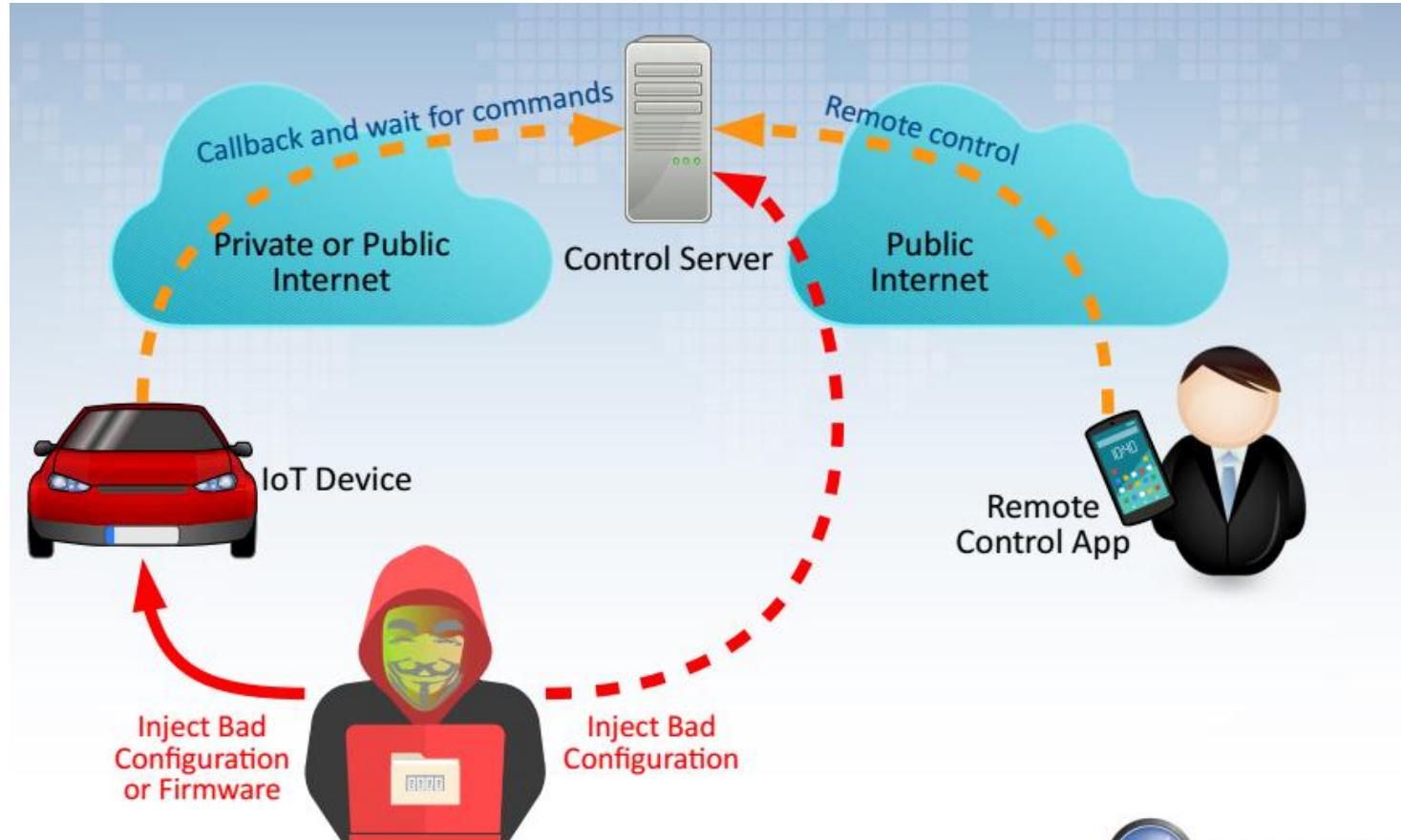
Một số dạng tấn công

- Steal Credential



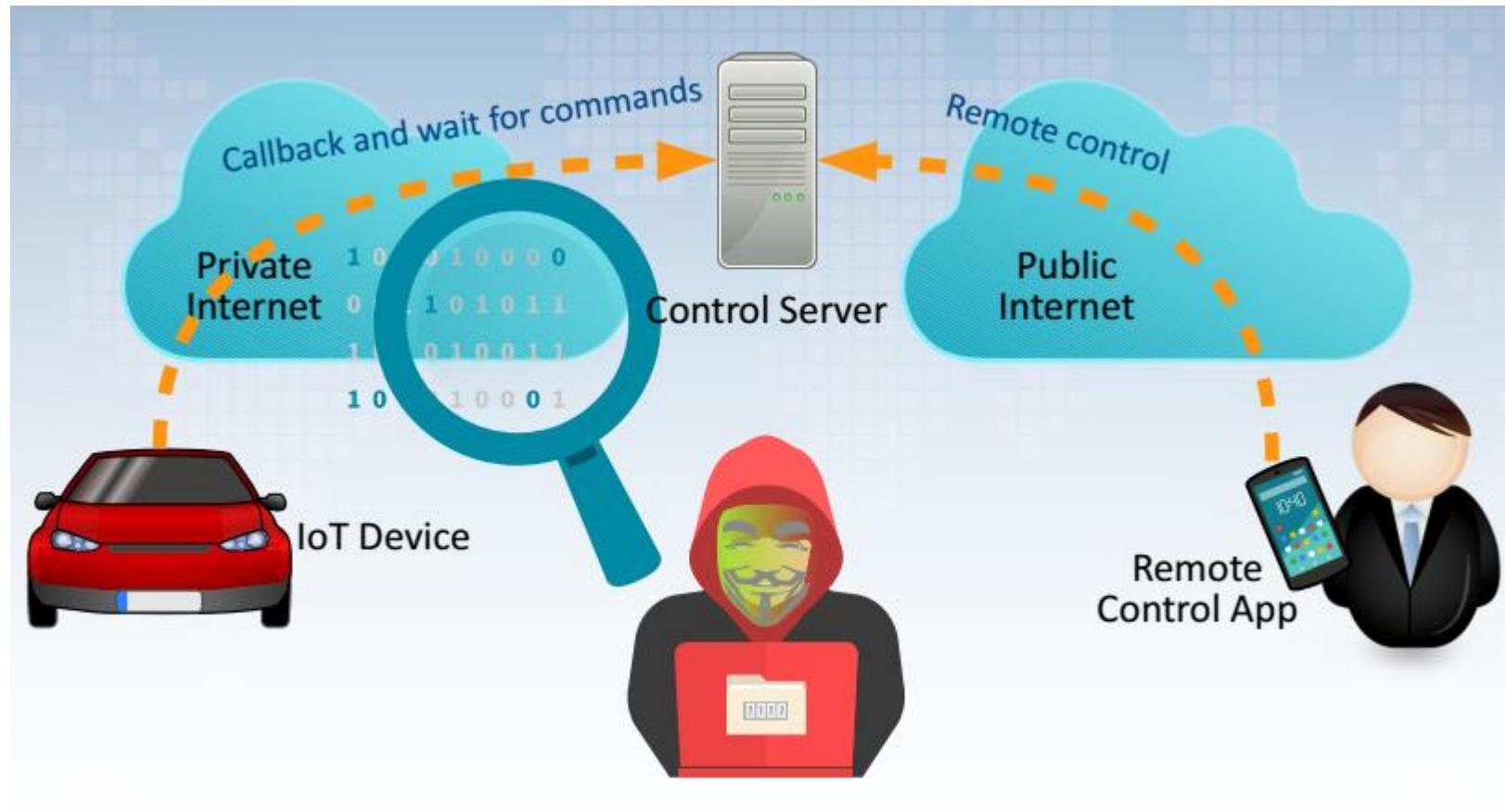
Một số dạng tấn công

- Inject bad configuration or firmware



Một số dạng tấn công

- Sniff data on private network



4.3. Các điểm yếu bảo mật IoT

- I1. Insecure Web Interface (Giao diện quản trị không an toàn)
- I2. Insufficient Authentication/Authorization (Xác thực không đủ an toàn)
- I3. Insecure Network Services (Các dịch vụ mạng thiếu bảo mật)
- I4. Lack of Transport Encryption/Integrity Verifcation (Thiếu mã hóa tầng giao vận)
- I5. Privacy Concerns (Các vấn đề liên quan quyền riêng tư)
- I6. Insecure Cloud Interface (Giao diện Cloud thiếu bảo mật)
- I7. Insecure Mobile Interface (Giao diện mobile thiếu bảo mật)
- I8. Insufficient Security Configurability (Cấu hình bảo mật không an toàn)
- I9. Insecure Software/Firmware (Phần mềm không an toàn)
- I10. Poor Physical Security (Thiếu bảo mật tầng vật lý)

1. Giao diện quản trị không an toàn

The slide is titled "OWASP INTERNET OF THINGS VULNERABILITY CATEGORIES" and features a large orange "TOP 10" banner. Below it, a teal box highlights "1 Insecure Web Interface covers IoT device administrative interfaces". To the right, there's a diagram of a network with a laptop, a cloud, and various IoT icons like a house, computer, and camera. A detailed diagram below shows a laptop connected to a cloud via dashed lines, with a lock icon indicating security issues.

1 Insecure Web Interface
covers IoT device administrative interfaces

Obstacles

	Default usernames and passwords		No account lockout		XSS, CSRF, SQLi vulnerabilities
--	---------------------------------	--	--------------------	--	---------------------------------

Solutions

	Allow default usernames and password to be changed		Enable account lockout		Conduct web application assessments
--	----------------------------------------------------	--	------------------------	--	-------------------------------------

2. Cơ chế xác thực không an toàn

The slide is from the OWASP Internet of Things Top 10 Vulnerability Categories report. It features the OWASP logo at the top left, followed by 'INTERNET OF THINGS' and 'VULNERABILITY CATEGORIES'. To the right is a large orange '10' with 'TOP' written vertically next to it. A background image shows a world map with three blue circles connected by lines, containing icons of a house, a computer monitor, and a camera. Below the main title, a teal box highlights 'Insufficient Authentication/Authorization' as 'covers all device interfaces and services' and is ranked '2'.

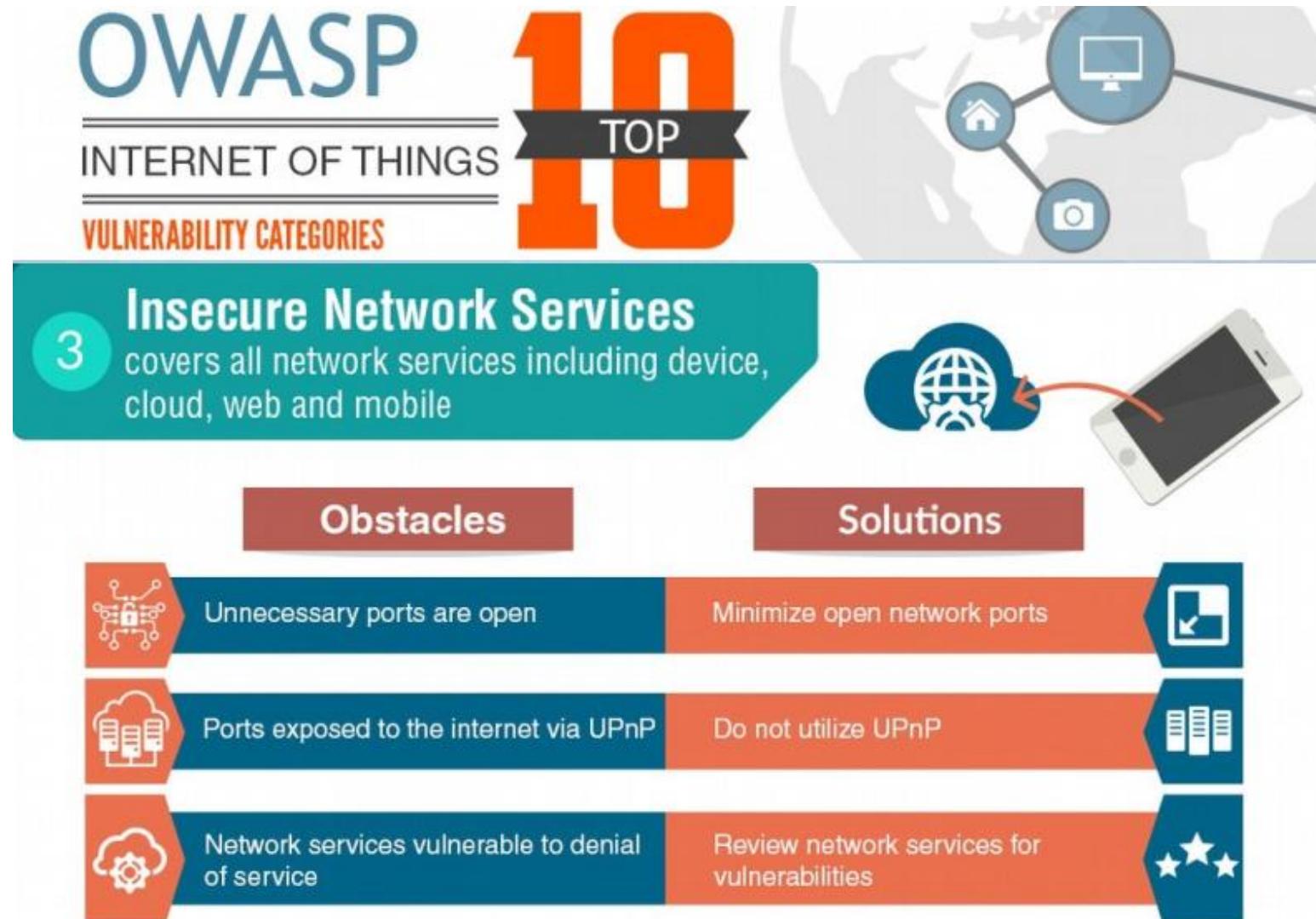
Obstacles

- Weak passwords
- Password recovery mechanisms are insecure
- No two-factor authentication available

Solutions

- Require strong, complex passwords
- Verify that password recovery mechanisms are secure
- Implement two-factor authentication where possible

3. Các dịch vụ mạng không an toàn



4. Thiếu sử dụng mã hóa tầng giao vận

The slide is titled "OWASP INTERNET OF THINGS VULNERABILITY CATEGORIES" and features a large orange "10" with "TOP" written above it. A world map background shows three devices connected: a house icon, a computer monitor icon, and a camera icon.

Obstacles

- Sensitive information is passed in clear text
- SSL/TLS is not available or not properly configured
- Proprietary encryption protocols are used

Solutions

- Encrypt communication between system components
- Maintain SSL/TLS implementations
- Do not use proprietary encryption solutions

Lack of Transport Encryption covers all network services including device, cloud, web and mobile 4

5. Các vấn đề về quyền riêng tư



The slide features the OWASP logo at the top left, followed by "INTERNET OF THINGS" and "VULNERABILITY CATEGORIES". To the right is a large orange "10" with "TOP" written vertically next to it. Below the number is a graphic of a world map with three blue circles connected by lines, containing icons of a house, a computer monitor, and a camera. A teal callout box on the left contains the number "5" and the text "Privacy Concerns covers all components of IoT solution". Below this box is a magnifying glass icon over a user profile, surrounded by various IoT icons like a key, a lock, and a shield.

Privacy Concerns
covers all components of IoT solution

Obstacles

- Too much personal information is collected
- Collected information is not properly protected
- End user is not given a choice to allow collection of certain types of data

Solutions

- Minimize data collection
- Anonymize collected data
- Give end users the ability to decide what data is collected

6. Giao diện Cloud không an toàn

The slide is part of the OWASP Top 10 IoT Vulnerabilities report. It features a world map in the background with icons representing various IoT devices (computer monitor, house, camera). The title "OWASP" is in blue, "INTERNET OF THINGS" is in grey, and "VULNERABILITY CATEGORIES" is in orange. A large orange "10" is prominently displayed with "TOP" written below it. The main content area has a teal background. On the left, there's a diagram of a computer monitor connected to a cloud icon with a lock. Below this is a section titled "Obstacles" with three arrows pointing right: "Interfaces are not reviewed for security vulnerabilities", "Weak passwords are present", and "No two-factor authentication is present". To the right is a section titled "Insecure Cloud Interface" with the number "6" in a green circle. It describes the vulnerability as covering cloud APIs or cloud-based web interfaces. Below this are three "Solutions" with icons: a cloud with a lock for "Security assessments of all cloud interfaces", a person icon for "Implement two-factor authentication", and a password field for "Require strong, complex passwords".

OWASP
INTERNET OF THINGS
VULNERABILITY CATEGORIES

10 TOP

Insecure Cloud Interface
covers cloud APIs or cloud-based web interfaces **6**

Obstacles

- Interfaces are not reviewed for security vulnerabilities
- Weak passwords are present
- No two-factor authentication is present

Solutions

- Security assessments of all cloud interfaces
- Implement two-factor authentication
- Require strong, complex passwords

7. Giao diện Mobile không an toàn

The slide is titled "OWASP INTERNET OF THINGS VULNERABILITY CATEGORIES" and features the "TOP 10" logo. A world map background shows three interconnected nodes: a house icon, a computer monitor icon, and a camera icon. Below the map, a teal callout box highlights "Insecure Mobile Interface" as category 7, which "covers mobile application interfaces".

Obstacles

- Weak passwords are present
- No two-factor authentication implemented
- No account lockout mechanism

Solutions

- Implement account lockout after failed login attempts
- Implement two-factor authentication
- Require strong, complex passwords

A large smartphone icon on the left is being examined with a magnifying glass, focusing on the screen area.

8. Thiếu cấu hình bảo mật

The slide is from the OWASP Internet of Things Top 10 Vulnerability Categories report. It features a world map background with icons of a house, computer monitor, and camera. The title 'TOP 10' is prominently displayed in orange. The category 'Insufficient Security Configurability' is highlighted in teal, listing 8 instances. Below it, 'Obstacles' and 'Solutions' are listed.

TOP 10

Insufficient Security Configurability covers the IoT device 8

Obstacles

- Password security options are not available
- Encryption options are not available
- No option to enable security logging

Solutions

- Make security logging available
- Allow the selection of encryption options
- Notify end users in regards to security alerts

9. Phần mềm không an toàn



The slide features the OWASP logo at the top left, followed by the text "INTERNET OF THINGS" and "VULNERABILITY CATEGORIES". To the right is a large orange "10" with "TOP" written vertically next to it. Below this, a teal box contains the number "9" and the category name "Insecure Software/Firmware covers the IoT Device". To the right of the teal box is a graphic showing a network of three devices (a house icon, a computer monitor icon, and a camera icon) connected to a map of the world.

9 Insecure Software/Firmware covers the IoT Device

Obstacles

-  Update servers are not secured
-  Device updates transmitted without encryption
-  Device updates not signed

Solutions

-  Sign updates
-  Verify updates before install
-  Secure update servers

A central illustration shows a computer screen with a flame icon on it, surrounded by a shield, a checkmark, and a small robot-like character.

10. Thiếu bảo mật tầng vật lý



Nội dung

- Chương 1. Tổng quan về IoT
- Chương 2. Các công nghệ IoT
- Chương 3. Lập trình ứng dụng IoT
- Chương 4. An toàn và Bảo mật IoT
- Chương 5. Thiết kế và xây dựng hệ thống IoT

Chương 5. Thiết kế và xây dựng hệ thống IoT

- Project-Based Learning:
 - Học dựa trên giải quyết một bài toán/dự án IoT cụ thể
- Kỹ năng:
 - Tìm vấn đề cần giải quyết (Finding the problems)
 - Phân tích thiết kế (System design and Analysis)
 - Xây dựng hệ thống (Implementing, Programming)
 - Trình bày (Presentation, Report)
 - Làm việc nhóm (Teamwork)

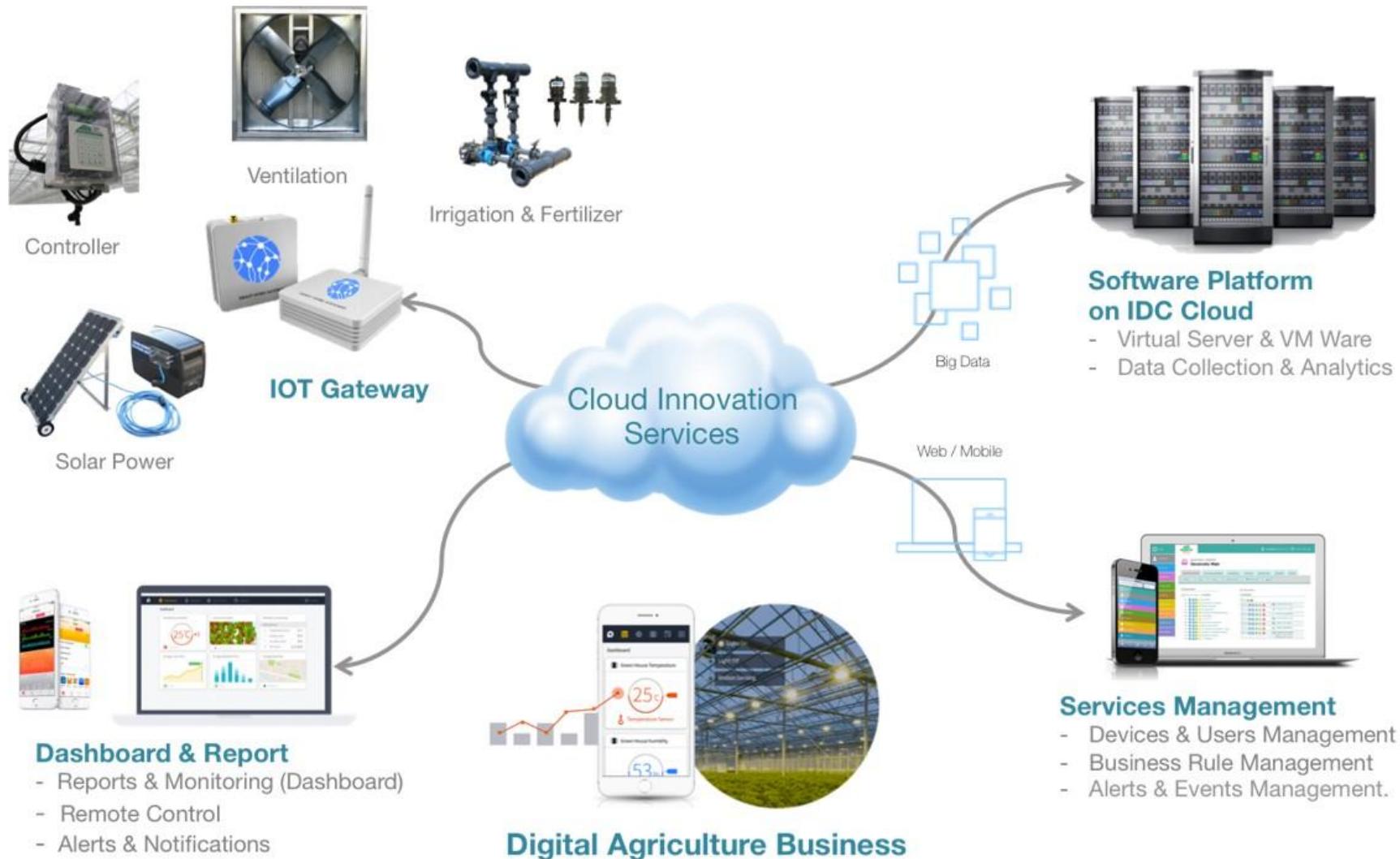
Cách thức thực hiện

- Lập nhóm dự án: 3-4 thành viên
- Xác định bài toán ứng dụng IoT
- Phân tích và thiết kế hệ thống IoT:
 - Thiết kế kiến trúc tổng quan của hệ thống
 - Thiết kế chi tiết các thành phần của hệ thống
 - Lựa chọn thiết bị phần cứng: máy tính nhúng, sensors
 - Lựa chọn giải pháp phần mềm/công nghệ IoT
- Xây dựng hệ thống:
 - Lập trình triển khai các thành phần của hệ thống
 - Vận dụng các giải pháp, dịch vụ, công nghệ IoT
- Thuyết trình và báo cáo

Các chủ đề ứng dụng IoT

- P1. Smart Agriculture:
 - Ứng dụng IoT trong nông nghiệp thông minh

P1. Smart Agriculture



<https://www.smartofthings.co.th/2018/08/27/smart-agriculture-solution/>

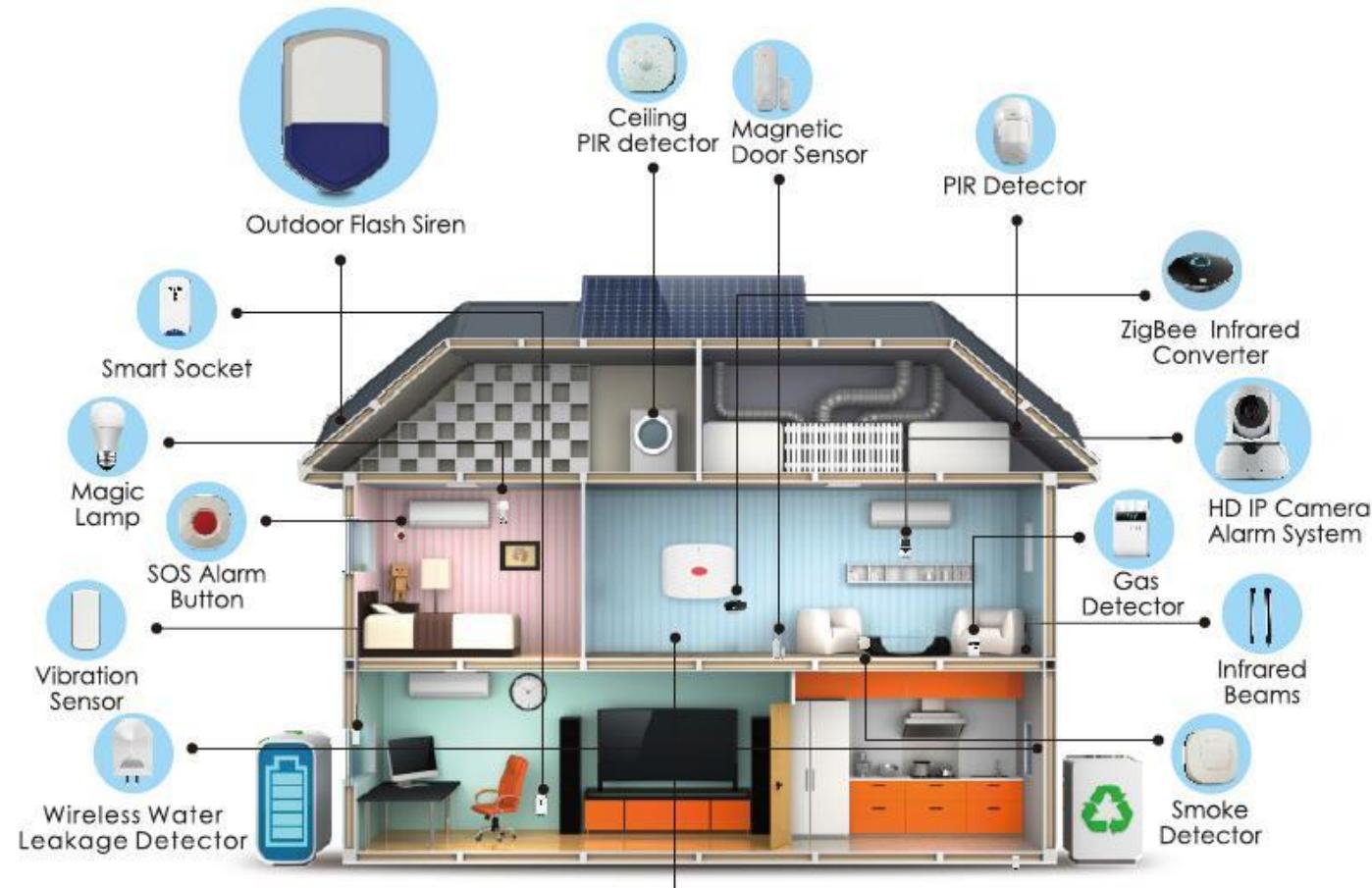
P2. Smart Home

- Ứng dụng IoT trong nhà thông minh:
 - Điều khiển các thiết bị từ xa (mobile app)
 - Điều khiển bằng giọng nói (Google assistant)
 - Giám sát từ xa

P2. Smart Home

- Ví dụ:

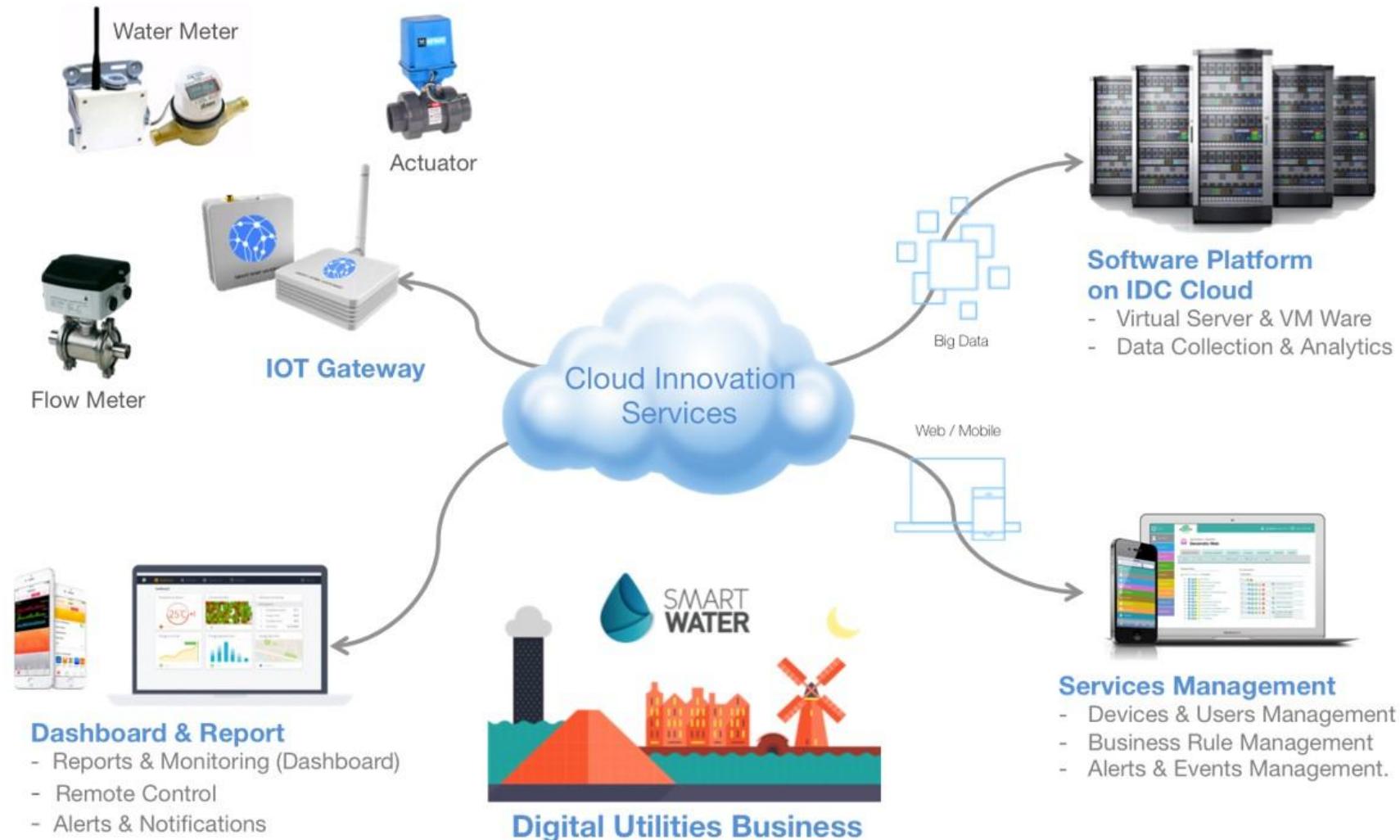
Smart Home



Các chủ đề ứng dụng IoT

- P3. Smart Water Supply Monitoring:
 - Ứng dụng IoT trong cấp nước thông minh
- P4. Smart Electrical Power Monitoring:
 - Ứng dụng IoT trong giám sát phân phối điện năng thông minh

Smart Utilities (Electrical power, Water)

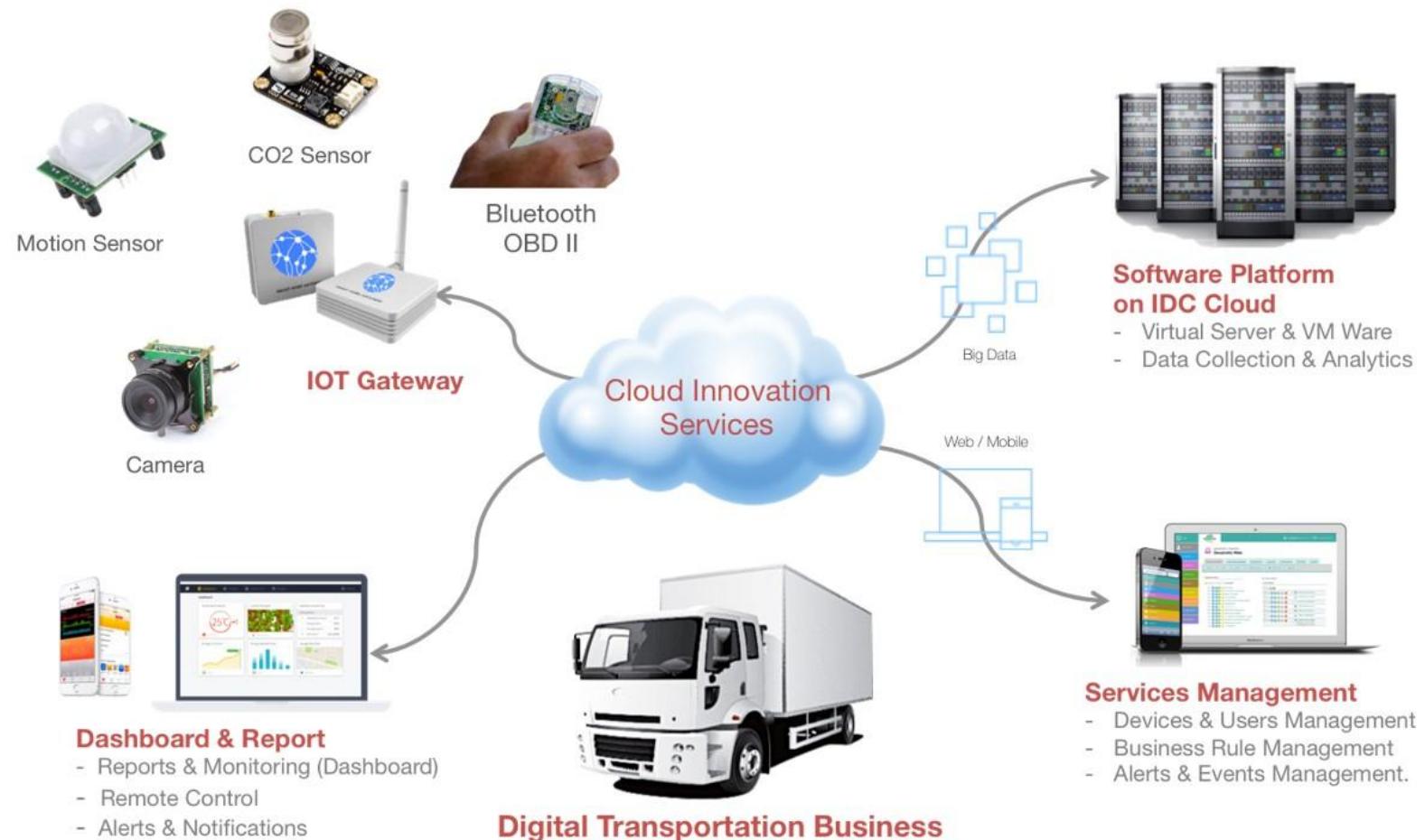


<https://www.smartofthings.co.th/2018/08/27/smart-utilities-solution/>

Các chủ đề ứng dụng IoT

■ P5. Smart Transportation:

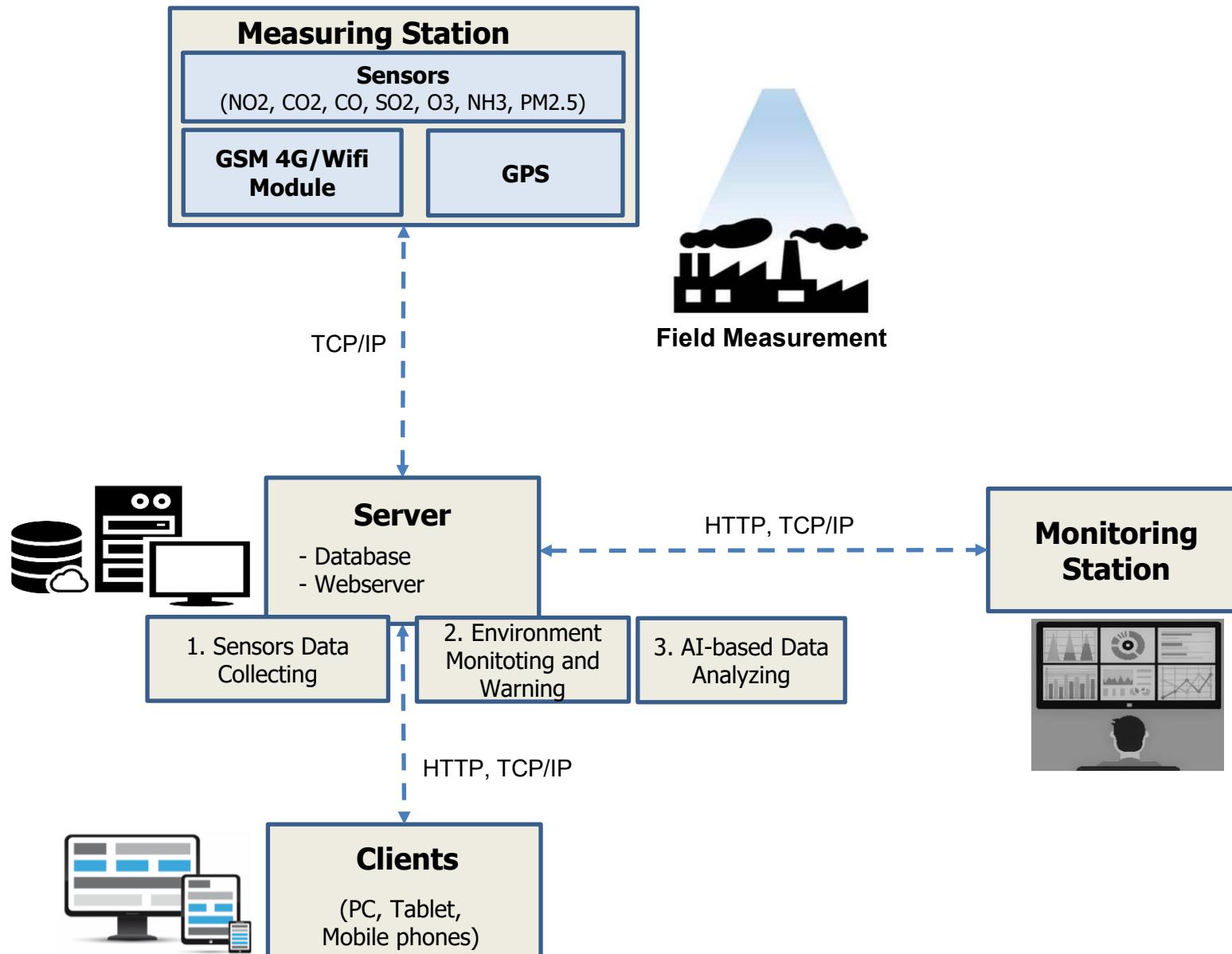
- Ứng dụng IoT trong giám sát vận tải thông minh



Các chủ đề ứng dụng IoT

- 6. Smart Air Quality Monitoring:
 - Ứng dụng IoT trong giám sát chất lượng không khí

P6. Smart Air Quality Monitoring

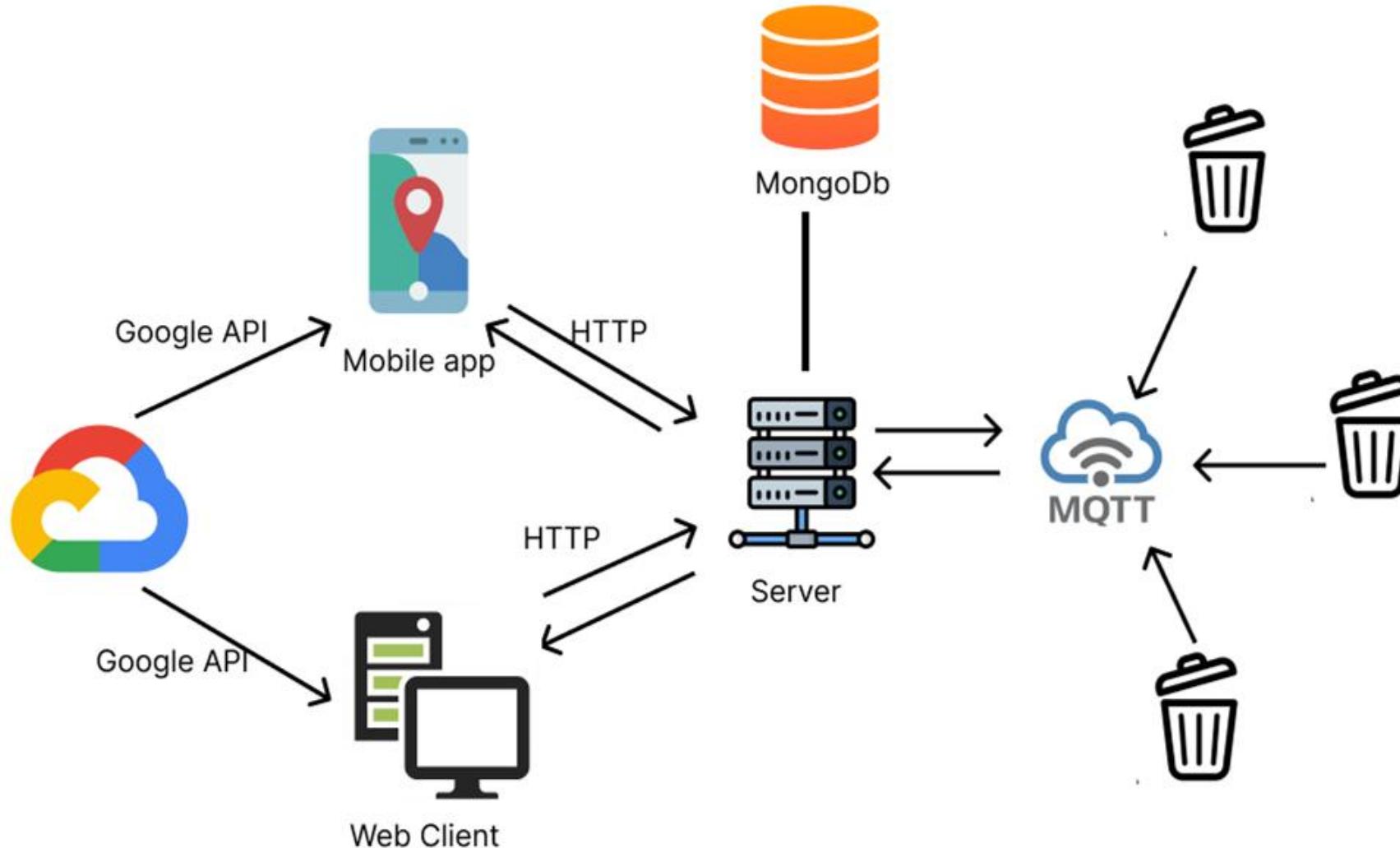


Các chủ đề ứng dụng IoT

- 7. Smart Gabaging:
 - Ứng dụng IoT trong thu gom rác thông minh
 - Tham khảo: <https://www.instructables.com/id/Smart-Garbage-Monitoring-System-Using-Internet-of-/>
- 8. Smart Parking:
 - Ứng dụng IoT trong đỗ xe thông minh
- 9. Smart Health Monitoring:
 - Ứng dụng IoT trong giám sát sức khỏe thông minh
- 10. Smart Robotic Warehouse:
 - Ứng dụng IoT trong nhà kho thông minh

Project 1: Smart Gabage

- Tổng quan hệ thống



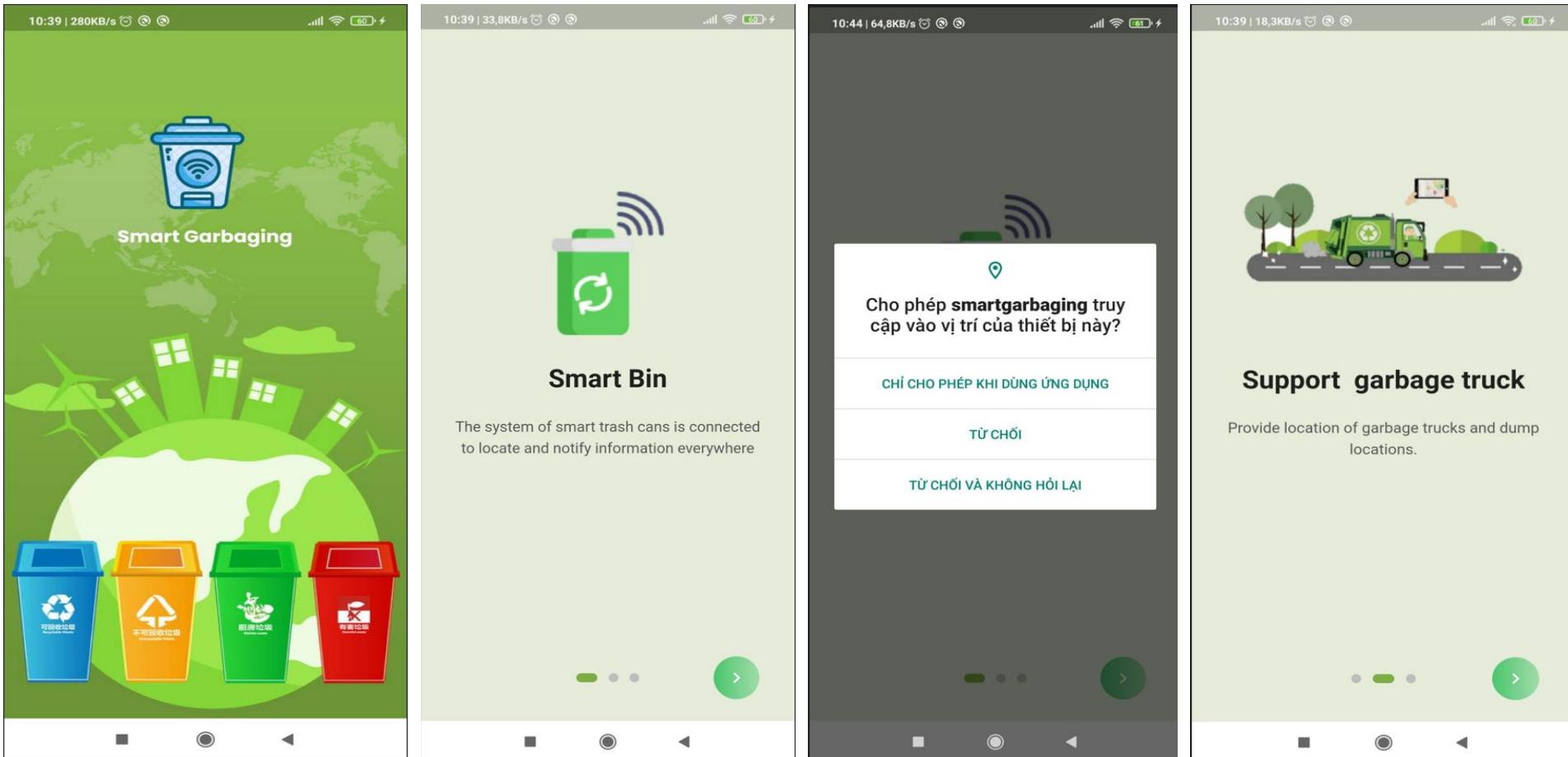
Project 1: Smart Gabage

- Web app quản lý: thêm/sửa thùng rác

The image displays a screenshot of a web application for managing trash bins. On the left, a sidebar menu lists options like 'ĐỔ RÁC' and 'THỐNG KÊ'. The main area shows a map of a city with several locations marked. A central modal window titled 'Tạo thùng rác' (Create trash bin) contains fields for 'Title' (填写) and 'Volumn' (Volume), both set to '0'. Below the form is a map showing the location of the new bin. At the bottom of the modal are 'CANCEL' and 'OK' buttons. To the right, a table lists existing trash bins with columns for 'Status' and 'Action'. A large blue button at the bottom right of the table says 'THÊM THÙNG RÁC' (Add trash bin). The top right corner of the screen shows a user profile icon with a notification count of 1.

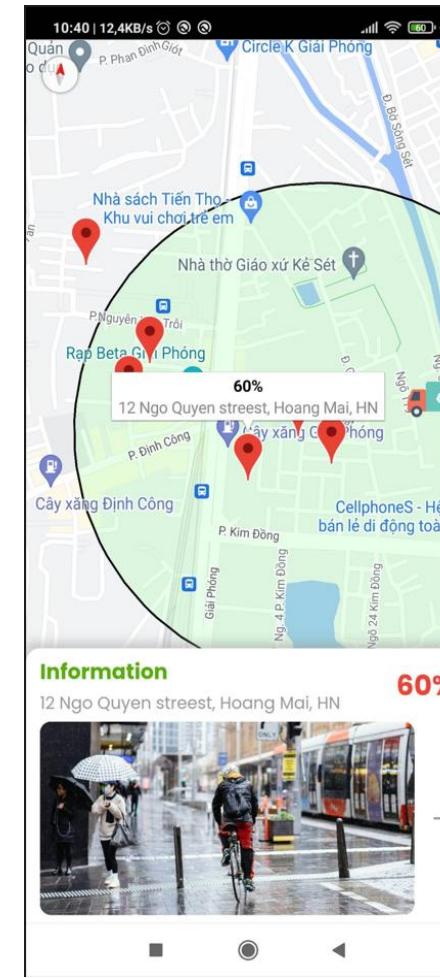
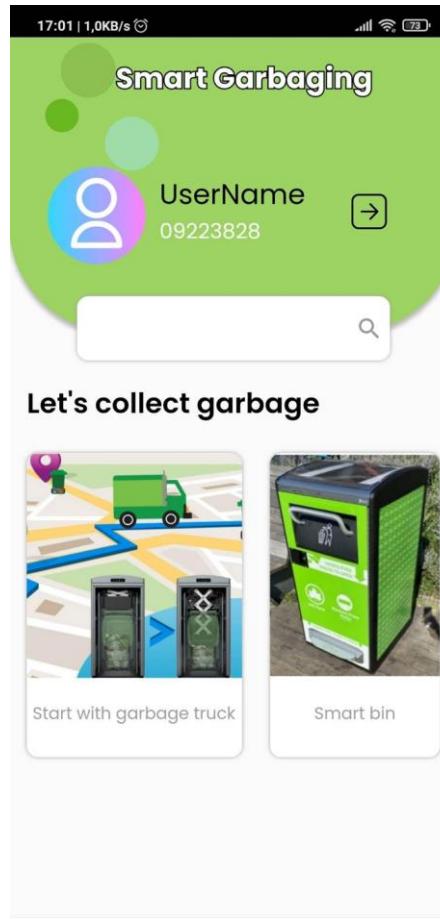
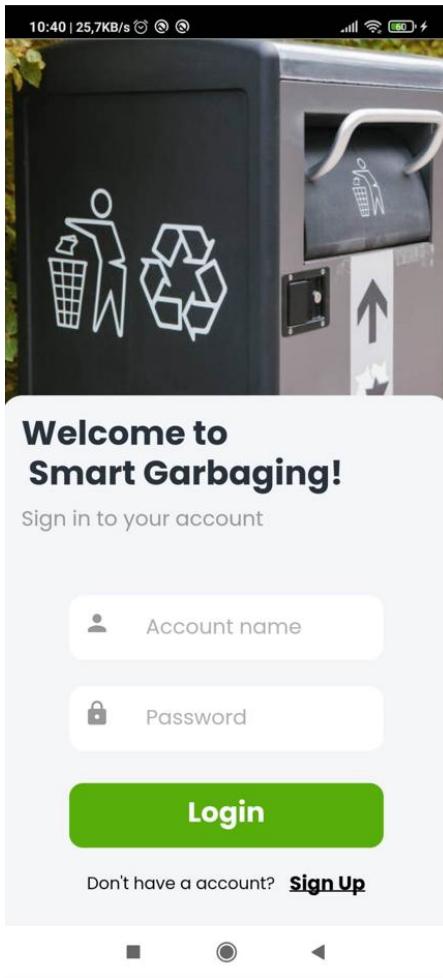
Project 1: Smart Gabage

- Mobile app



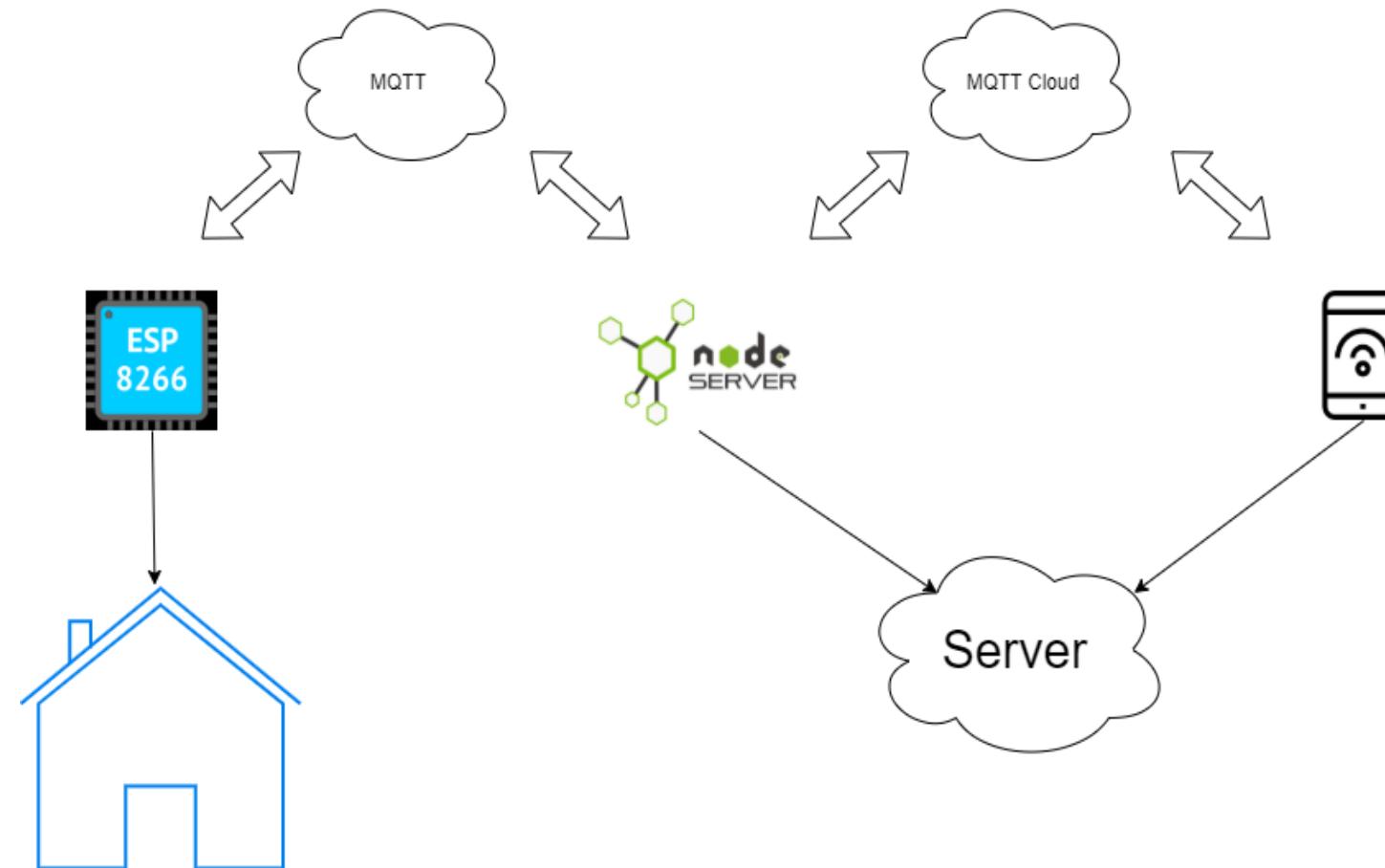
Project 1: Smart Gabage

■ Mobile app



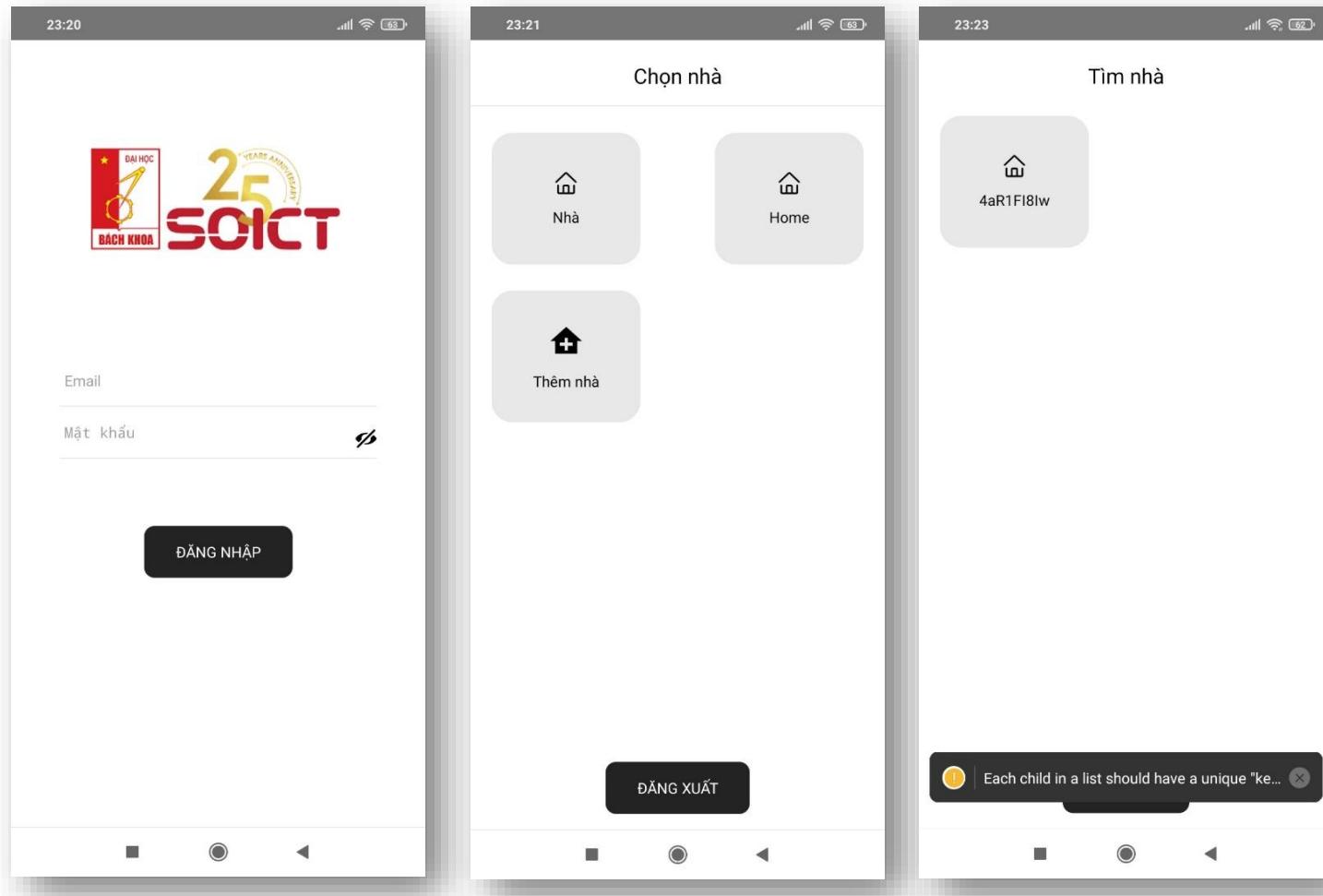
Project 2. Smart Home

- Sơ đồ thiết kế tổng quan



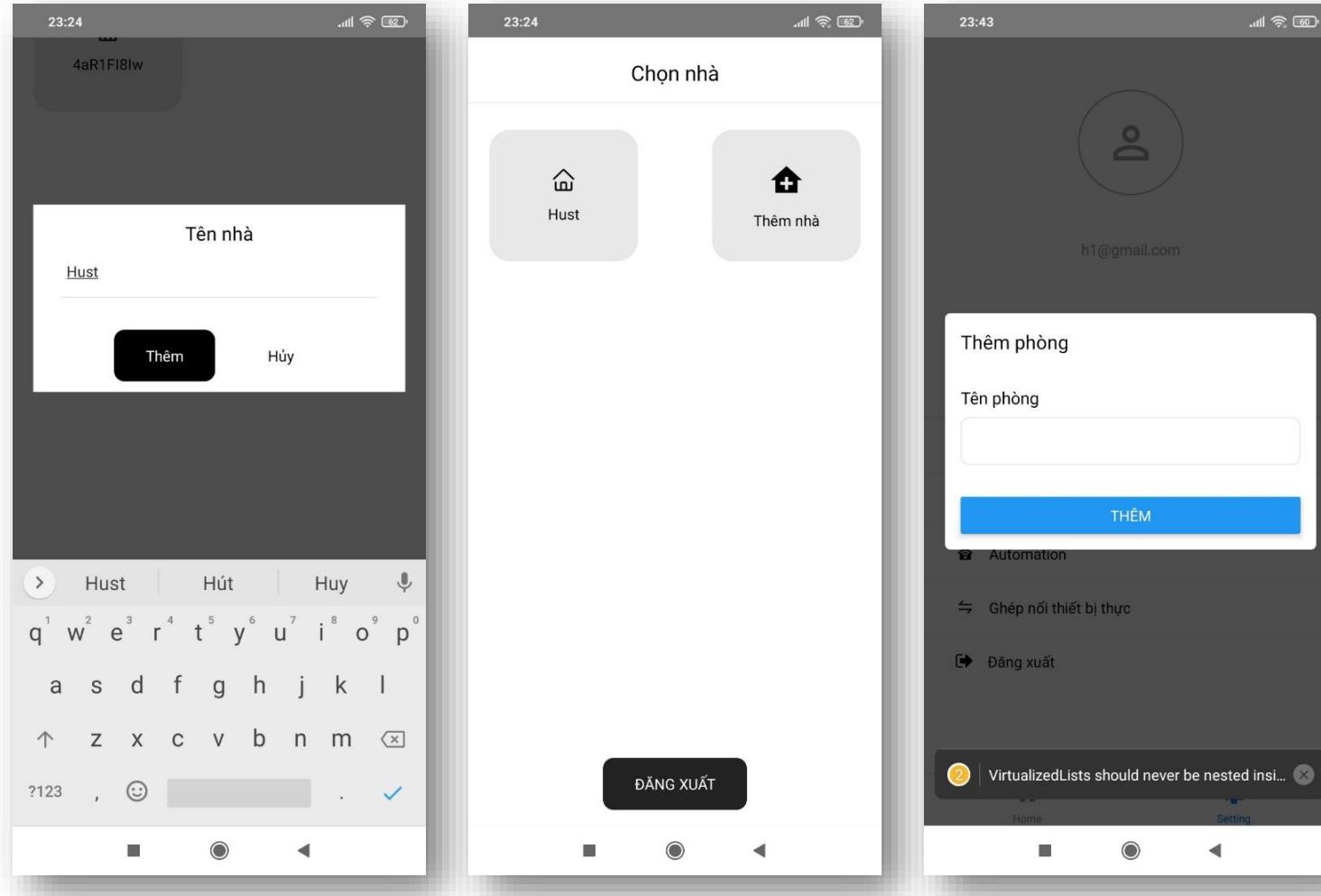
Project 2. Smart home

- Mobile app:
 - Quản lý home/rooms (add/edit)



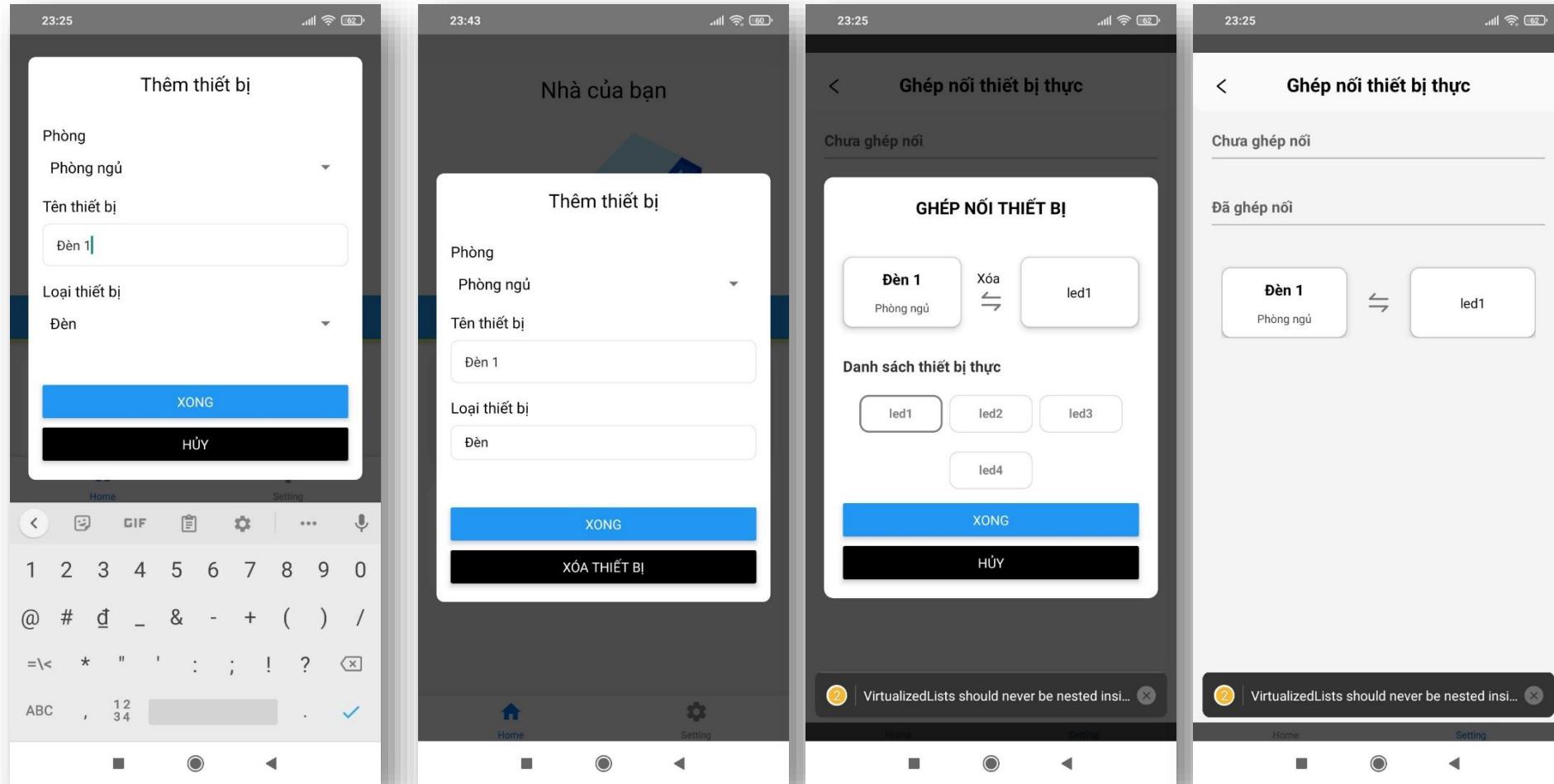
Project 2. Smart home

- Mobile app:
 - Quản lý home/rooms (add/edit)



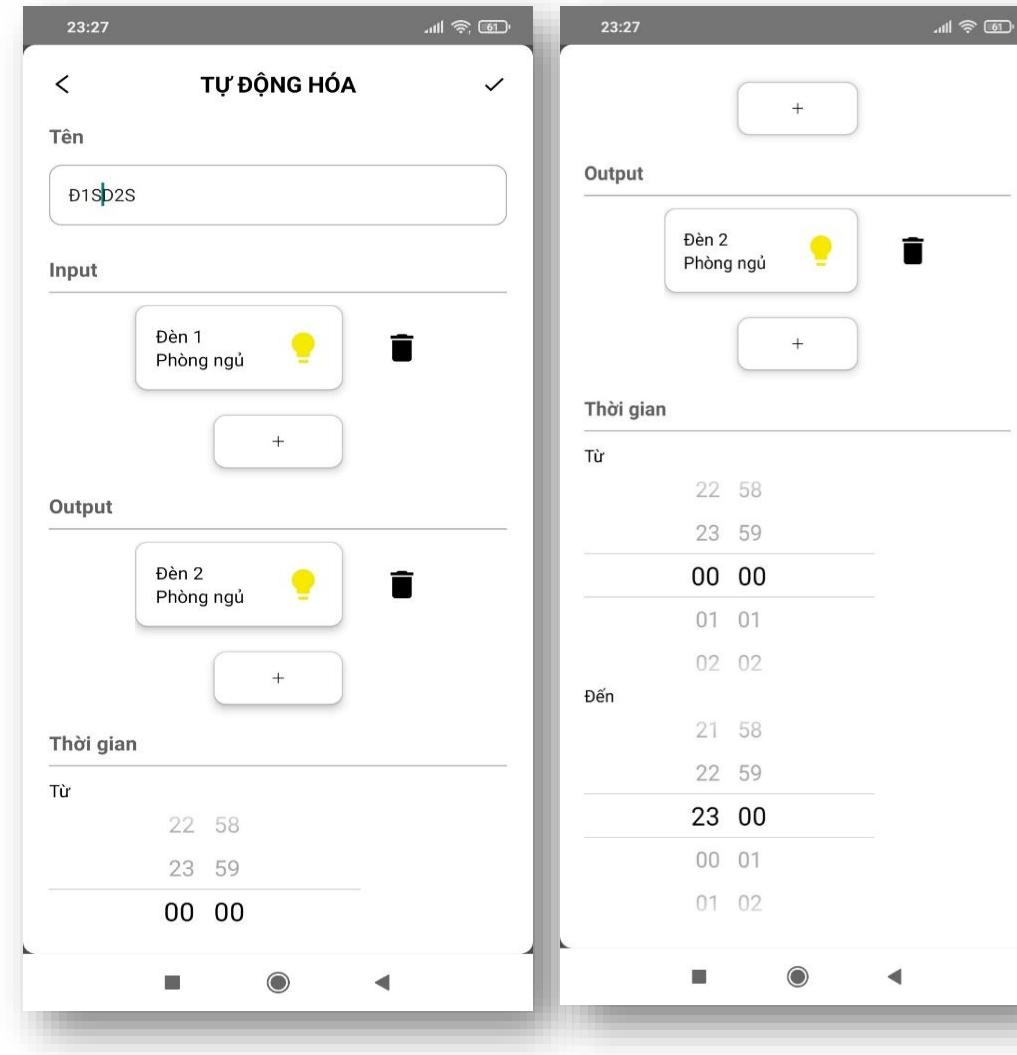
Project 2. Smart home

- Mobile app:
 - Add thiết bị, ghép nối thiết bị vật lý



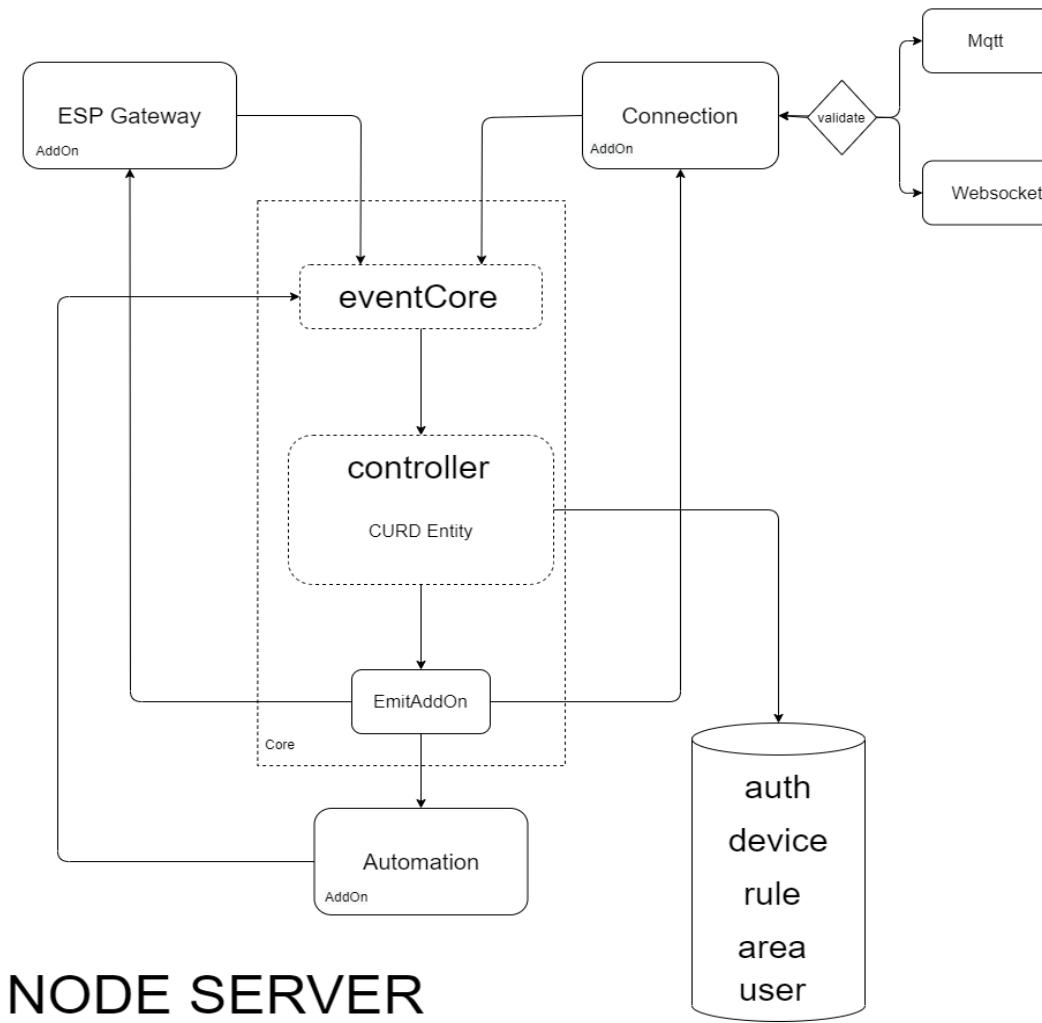
Project 2. Smart home

- Mobile app:
 - Automation



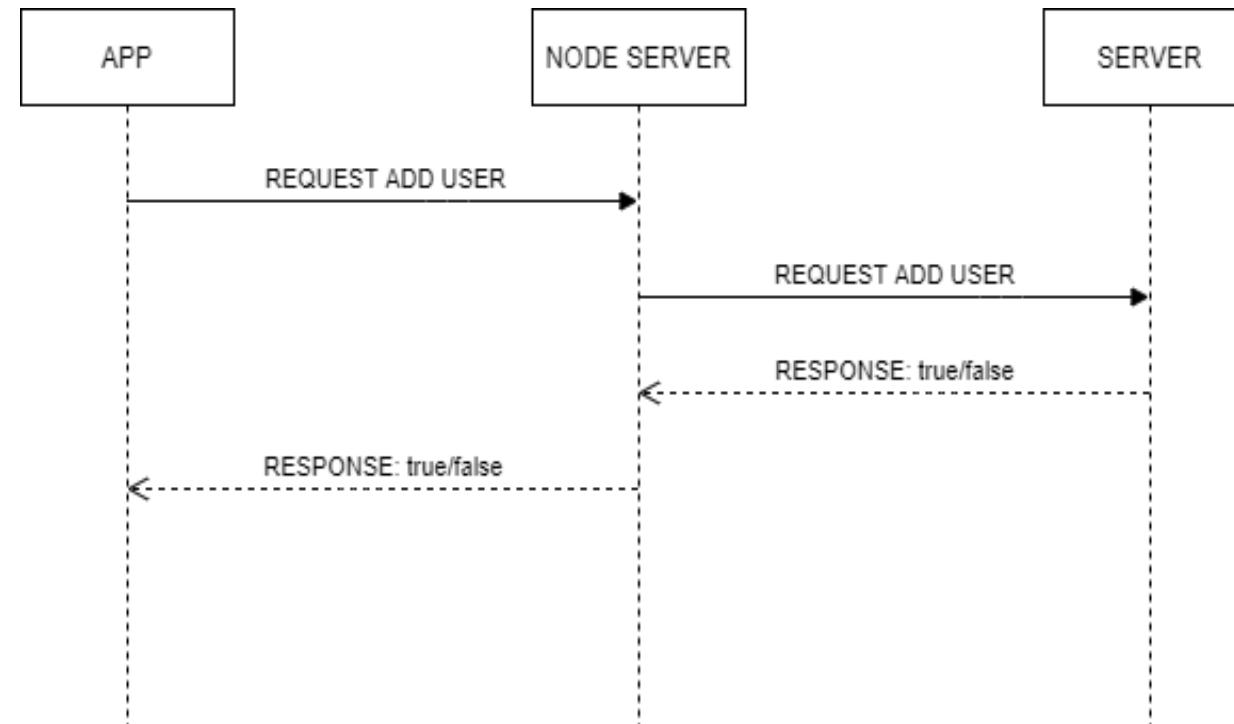
Project 2. Smart home

- Xây dựng server:



Project 2. Smart home

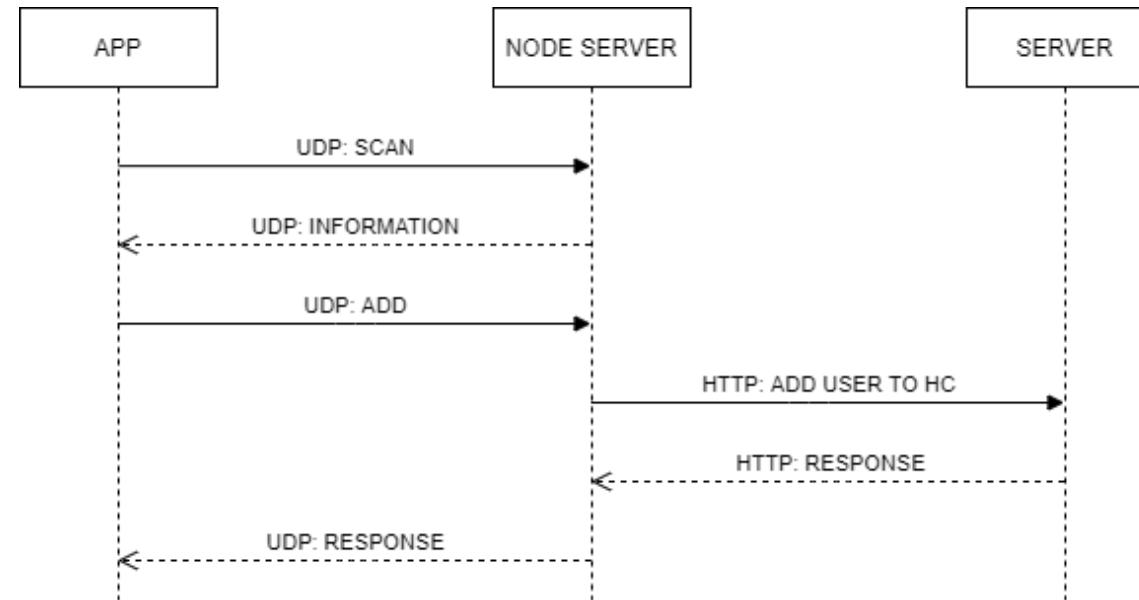
- Xây dựng server:
 - Luồng nghiệp vụ thêm/xóa người dùng



Thêm/xóa người dùng

Project 2. Smart home

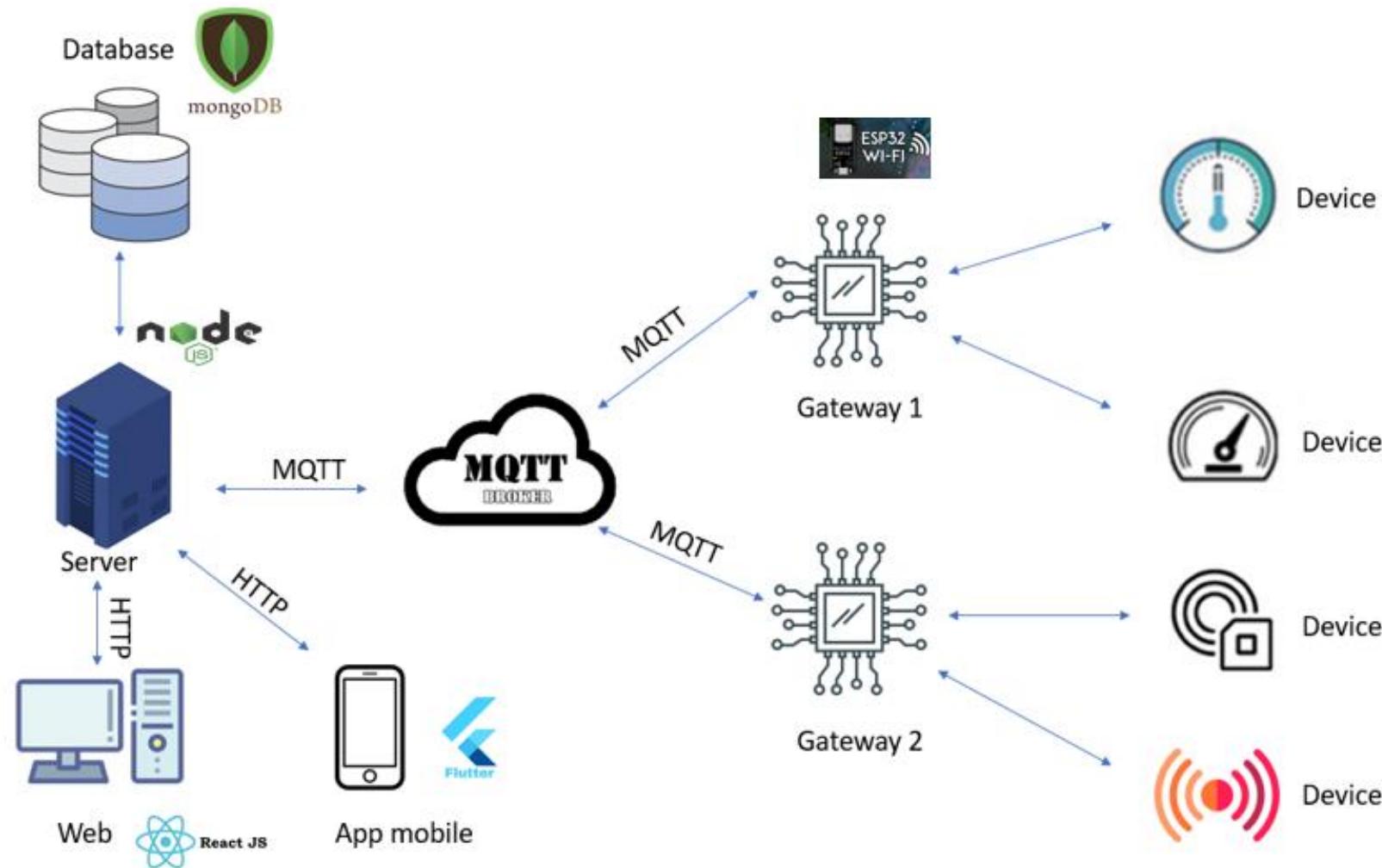
- Xây dựng server:
 - Luồng nghiệp vụ thêm nhà



Thêm một ngôi nhà mới

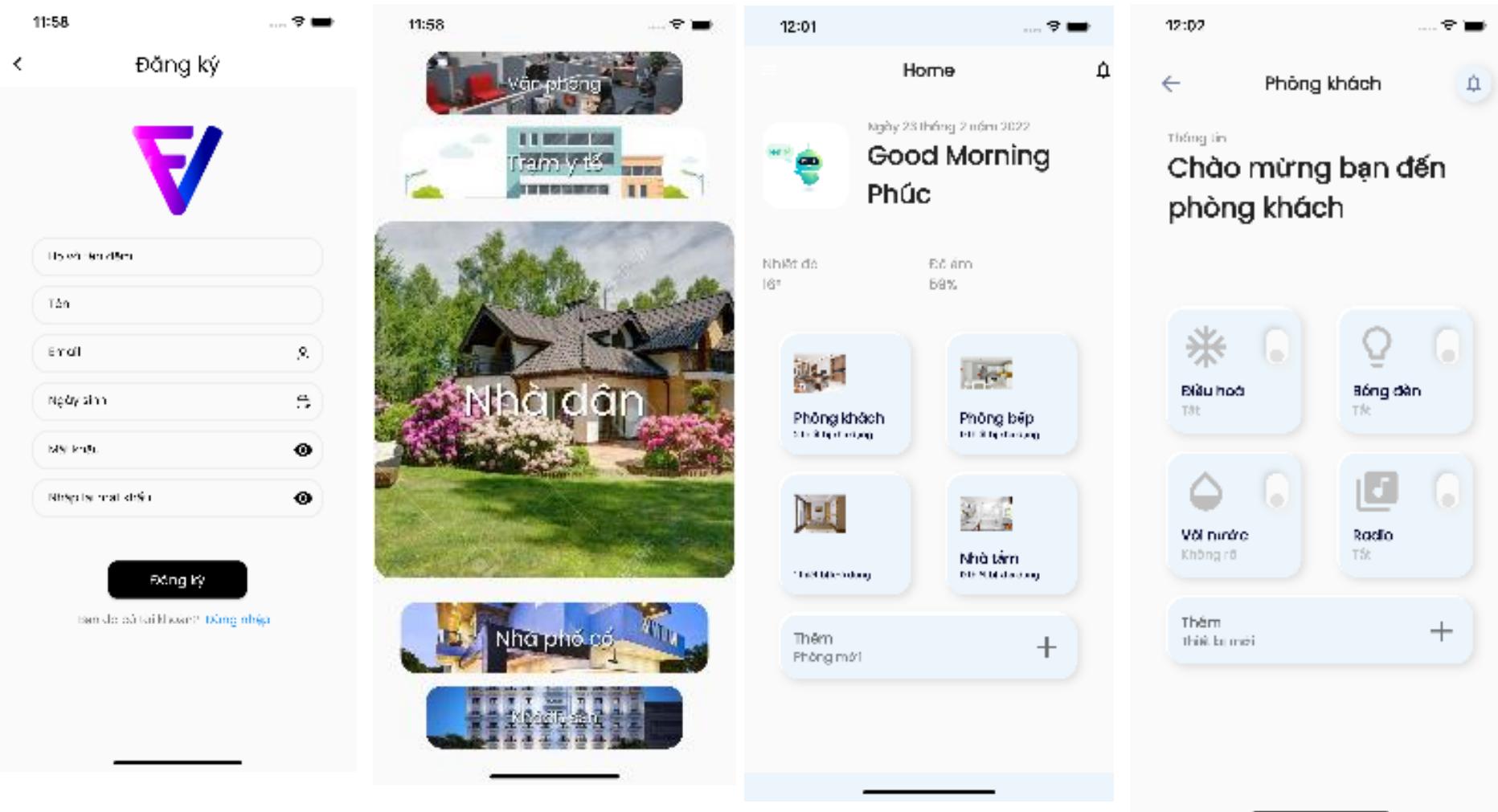
Project 3. Smart home

■ Thiết kế hệ thống



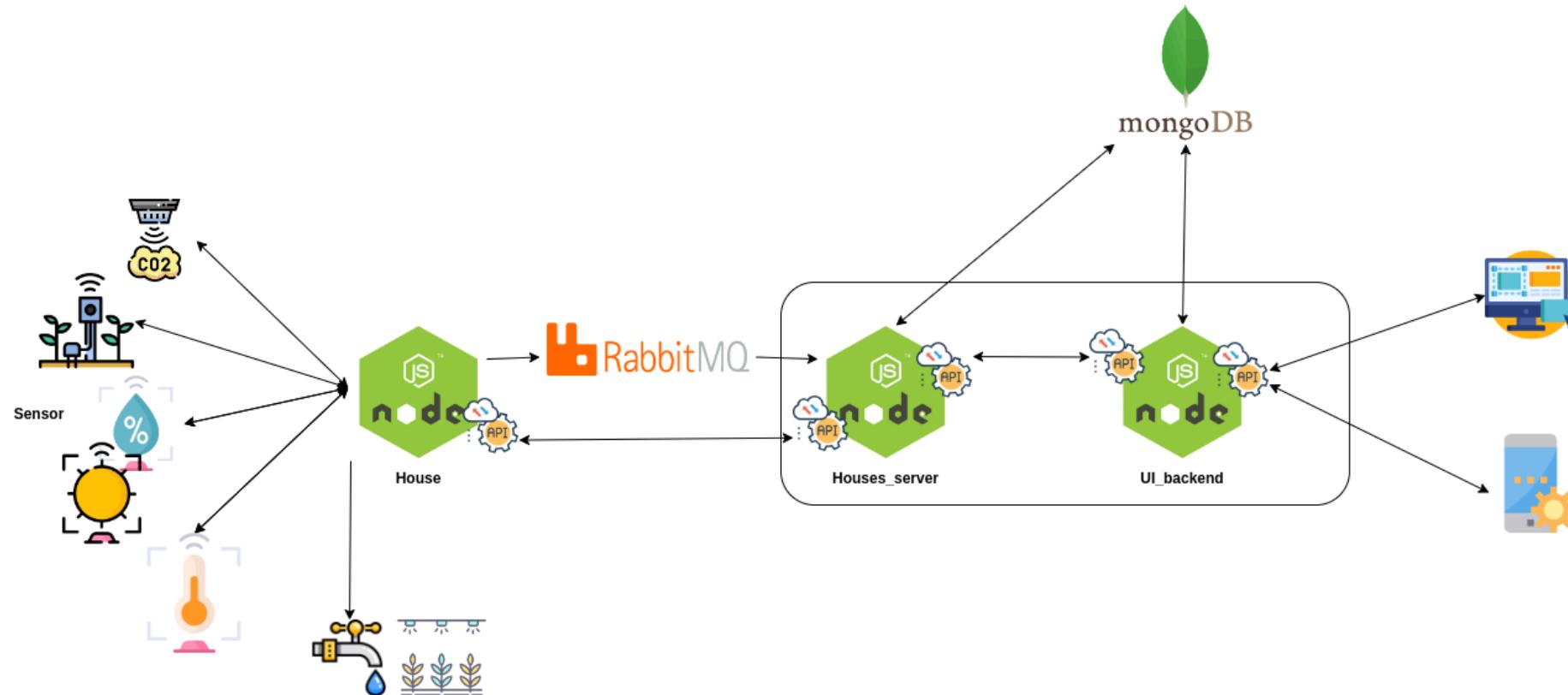
Project 3. Smart home

■ Mobile app



Project 4. Smart garden

- Kiến trúc hệ thống

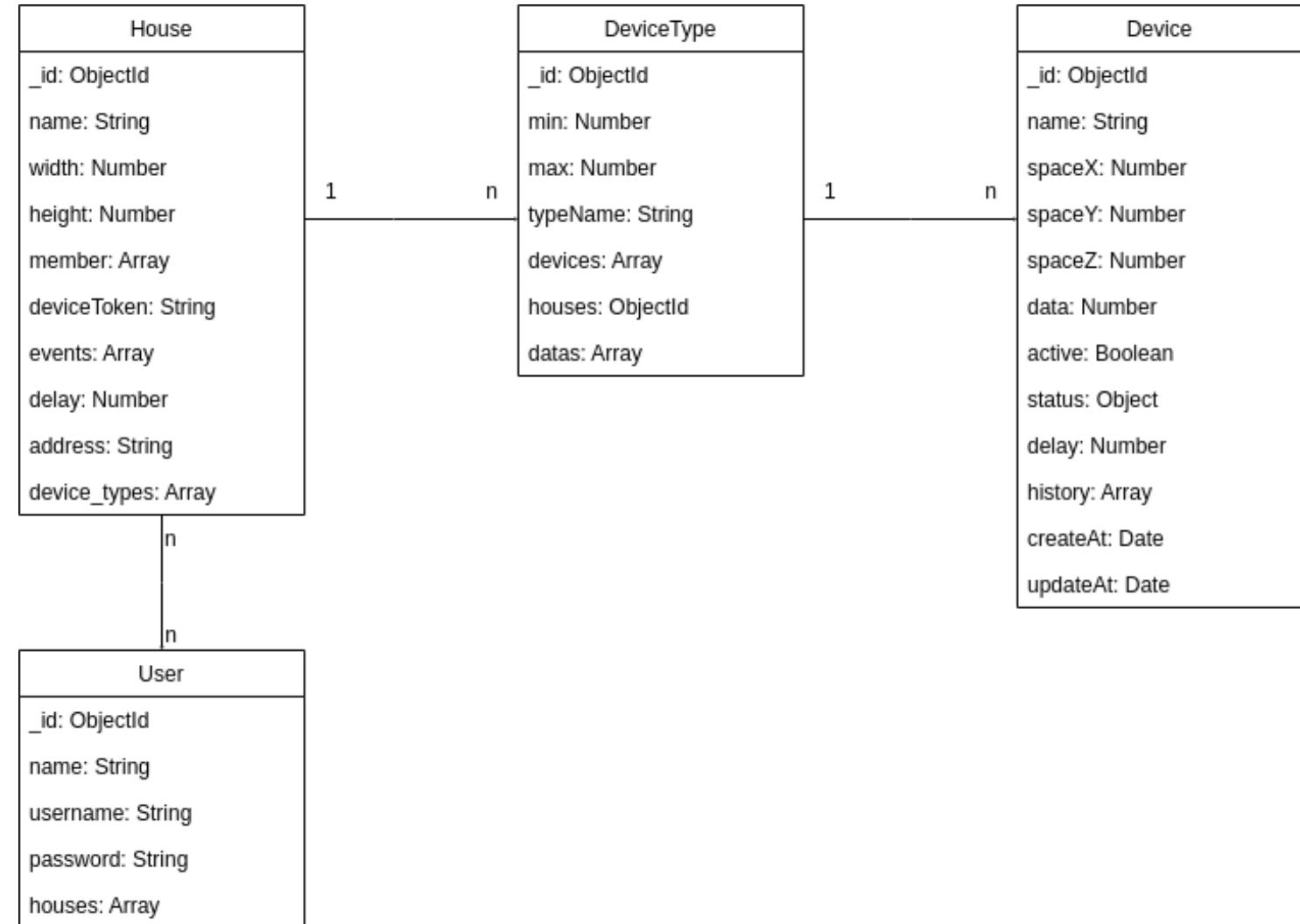


Project 4. Smart garden

- Công nghệ sử dụng:
 - Sử dụng dữ liệu ảo cho các thiết bị cảm biến.
 - Server: NodeJS, ExpressJS.
 - Database: MongoDB.
 - Website: React JS.
 - Mobile App: React Native.

Project 4. Smart garden

■ Thiết kế CSDL



Project 4. Smart garden

- Xây dựng các API
 - Đăng ký: POST /v1/auth/register
 - Đăng nhập: POST /v1/auth/login
 - Lấy danh sách các thiết bị: GET /v1/device/{houseId}
 - Sửa thông tin thiết bị: PUT /v1/device/{deviceId}
 - Lấy dữ liệu thống kê: GET /v1/data/{houseId}
 - Lấy danh sách các cảnh báo: GET /v1/house/events/{houseId}
 - Lấy thông tin nhà nấm: GET /v1/house
 - Thêm nhà nấm mới: POST /v1/house
 - Sửa thông tin nhà nấm: PUT /v1/house/update-house/{houseId}
 - Thêm thành viên: PUT /v1/house/update-member/{houseId}
 - Xóa thành viên: PUT /v1/house/delete-member/{houseId}/{userId}

Project 4. Smart garden

- Ví dụ API lấy ds thiết bị

The screenshot shows a Postman interface with the following details:

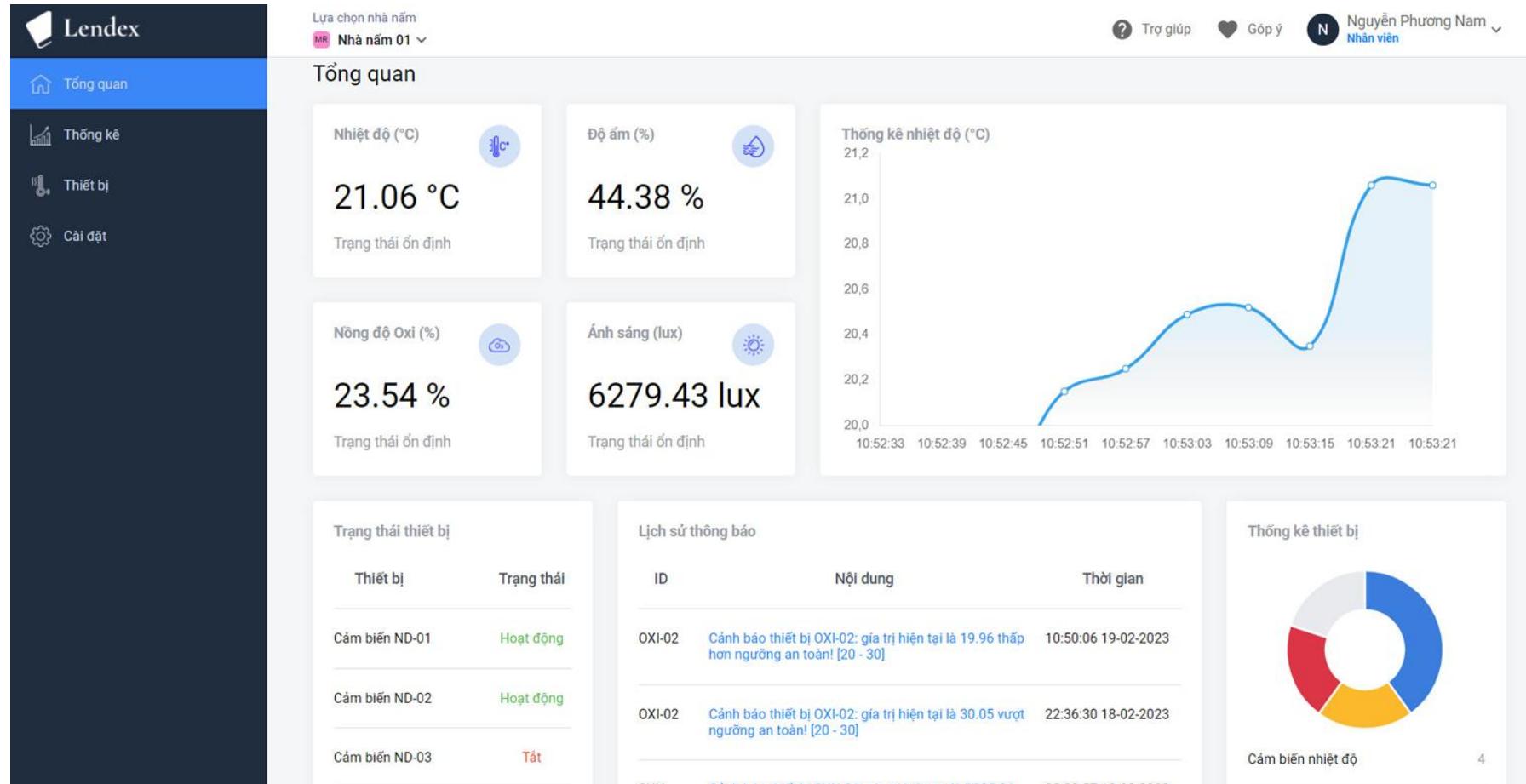
- Method:** GET
- URL:** http://localhost:9000/v1/device/63e8a7ab1d624a76b5d91255
- Headers:** (8)
- Body:** (Pretty, Raw, Preview, Visualize, JSON)
- Cookies:**
- Status:** 200 OK
- Time:** 233 ms
- Size:** 81.27 KB
- Save Response:** (dropdown menu)

The response body is a JSON object with the following structure:

```
1  "success": true,
2  "devices": [
3    {
4      "_id": "63de1380dd8742bc1ba5390a",
5      "name": "ND-01",
6      "active": true,
7      "spaceX": 100,
8      "spaceY": 0,
9      "spaceZ": 0,
10     "data": 21.8,
11     "created_at": "2023-02-04T08:12:48.776Z",
12     "__v": 0,
13     "delay": 2,
14     "status": {...},
15   },
16   {
17     "updatedAt": "2023-02-19T03:53:20.523Z",
18   },
19   {
20     "history": [...],
21   },
22   {
23     "max": 30,
24     "min": 15,
25     "type_name": "Temp"
26   },
27   {
28     "_id": "63de139ddd8742bc1ba5390f",
29     "name": "ND-02",
30     "active": true,
31   }
32 ]
```

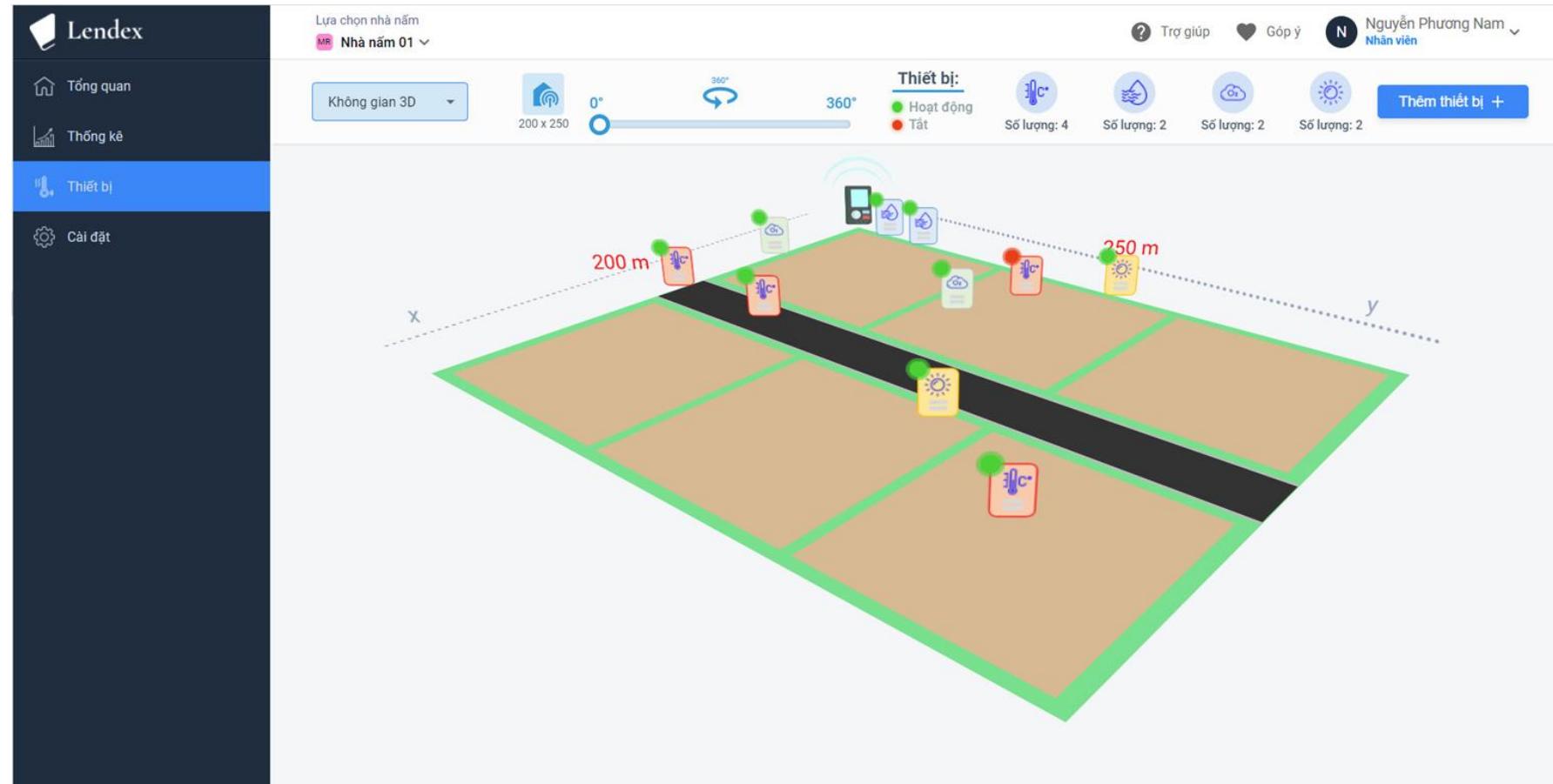
Project 4. Smart garden

■ Web app quản trị:



Project 4. Smart garden

- Web app quản trị:



Project 4. Smart garden

- Web app quản trị:

The screenshot shows the Lendex web application interface. The left sidebar has navigation links: Tổng quan, Thông kê, Thiết bị, and Cài đặt (which is highlighted). The main content area is titled 'Cài đặt' and contains two tabs: 'Thông tin nhà nấm' (selected) and 'Tạo mới nhà nấm'. Under 'Thông tin nhà nấm', there are three input fields: 'Tên nhà nấm' (Nhà nấm 01), 'Chiều dài nhà nấm' (250), and 'Chiều rộng nhà nấm' (200). A 'Chỉnh sửa' button is located at the bottom right of this section. To the right, there is a sidebar titled 'Thành viên nhà nấm' with a search bar and a list of users: Nguyễn Phương Nam (Quản trị viên), Nguyễn Mạnh Duy, and Trần Đức Hải. Each user has a trash can icon next to their name.

Project 4. Smart garden

■ Mobile app

