

IT4735E

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

Course Introduction

- IT4735E:IoT and Applications
- Credits: 2 (2-1-0-4)
- Evaluation:
 - Progress: 40%, Weekly exercises, Project exercises
 - Final-term: 60%,
 - Project exercises, presentation, oral exam (Q&A)
 - Final exam

Goal

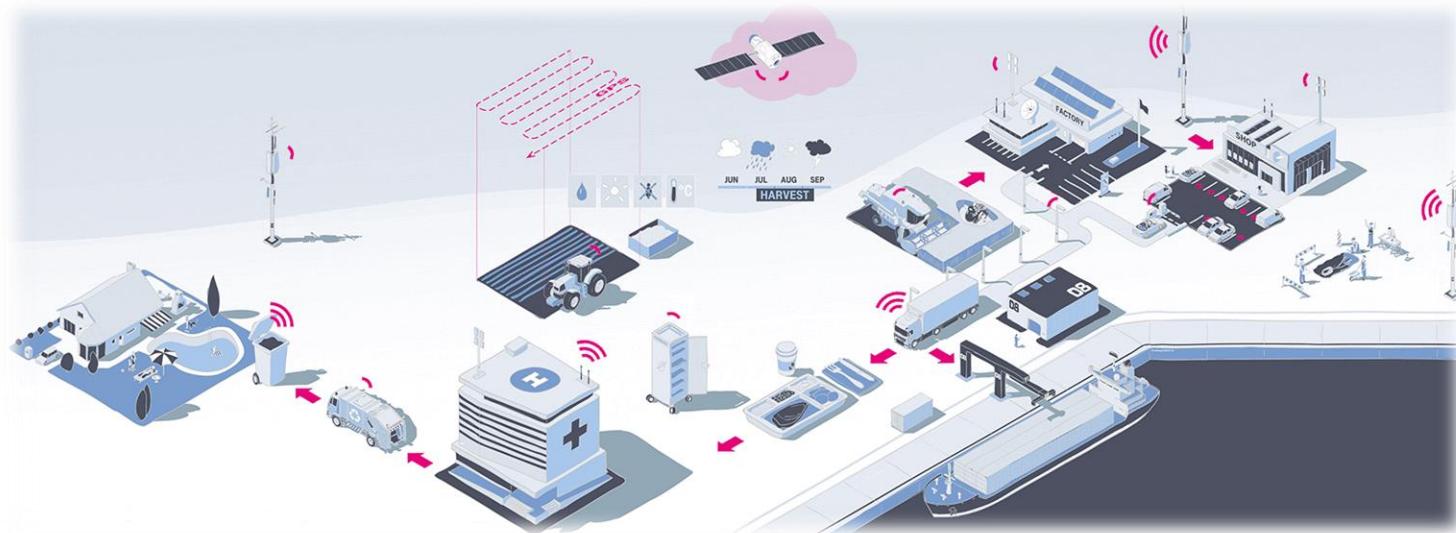
- Overview of IoT
- Architecture, layered model, components of IoT system
- IoT technologies: devices and sensors, IoT communication standards, communication protocols for IoT applications, cloud platforms
- Programming for IoT devices and systems
- Security in IoT
- Design and build an IoT application system

Content

- Chapter 1. Overview of IoT
- Chapter 2. IoT Technologies
- Chapter 3. IoT Application and Programming
- Chapter 4. IoT Safety and Security
- Chapter 5. Designing and Building IoT Systems

Chapter 1. Overview of IoT

- 1.1. Overview of IoT
- 1.2. Evolution of IoT
- 1.3. Architecture of IoT System
- 1.4. IoT Technologies
- 1.5. IoT Applications
- 1.6. Challenges of IoT



1.1. Overview of 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*

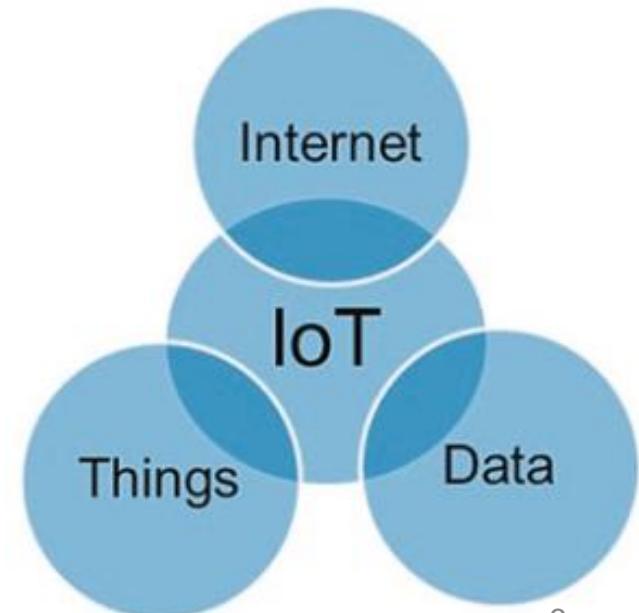
(Book: Internet of Things From Hype to Reality)

- *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”

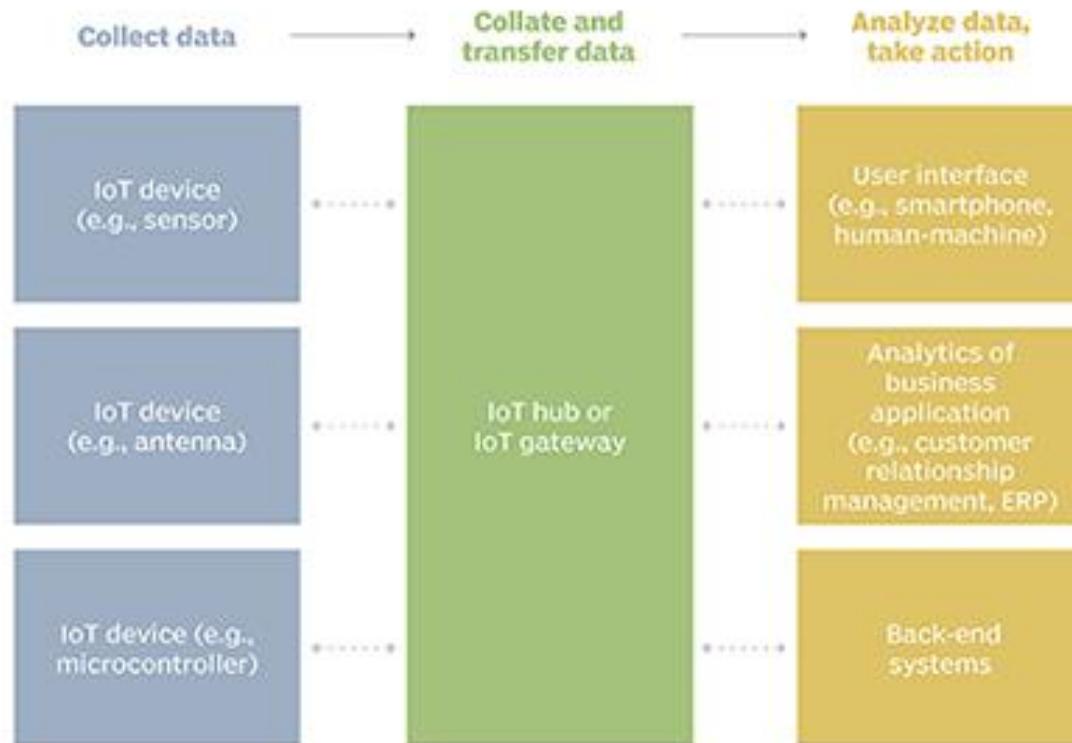
- Home appliances, building, car, people, animals, trees, plants, ...



IoE = Internet of Everything (by Cisco)

1.1. Overview of IoT (Cont.)

Example of an IoT system



1.1. Overview of IoT (cont.)

- 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.

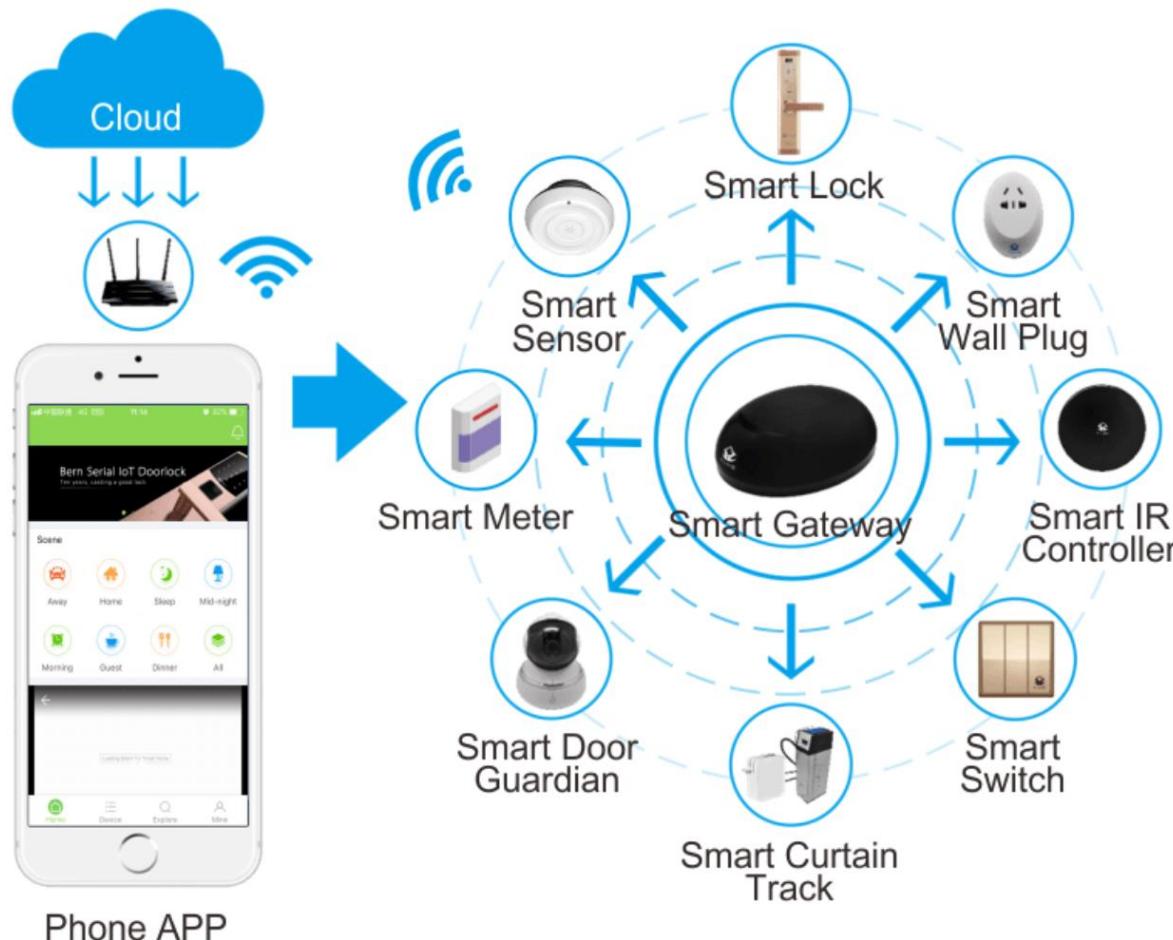
1.1. Overview of IoT (cont.)

■ 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.

1.1. Overview of IoT (cont.)

- An example of IoT smart home solution

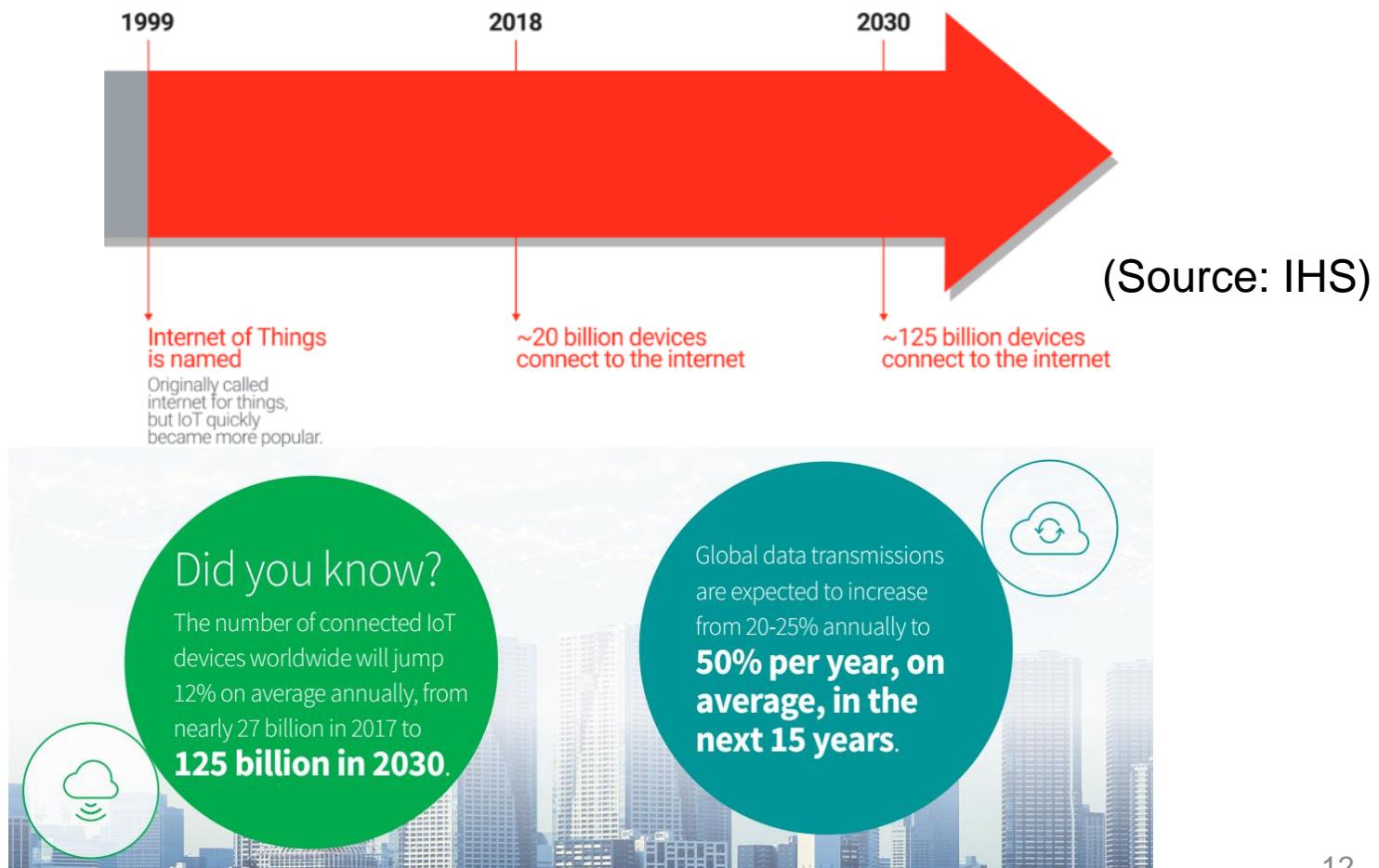


1.1. Overview of IoT (cont.)

- **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.

1.2. Evolution of IoT

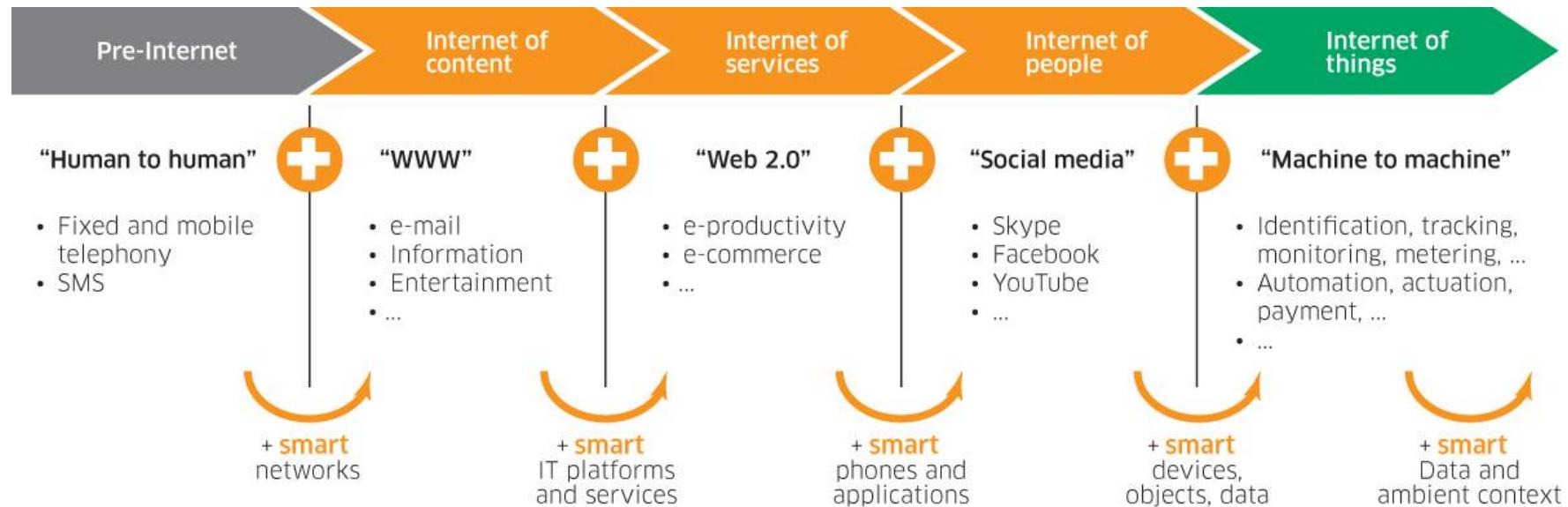
- The field of IoT has grown tremendously
- 1999: Kevin Ashton (Auto-ID Center, MIT) first mentioned the internet of things in a presentation about RFID



1.2. Evolution of IoT (cont.)

- 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

1.2. Evolution of IoT (cont.)



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/>

1.2. Evolution of IoT (cont.)

Idea: Move from Internet of People



→ Internet of Things

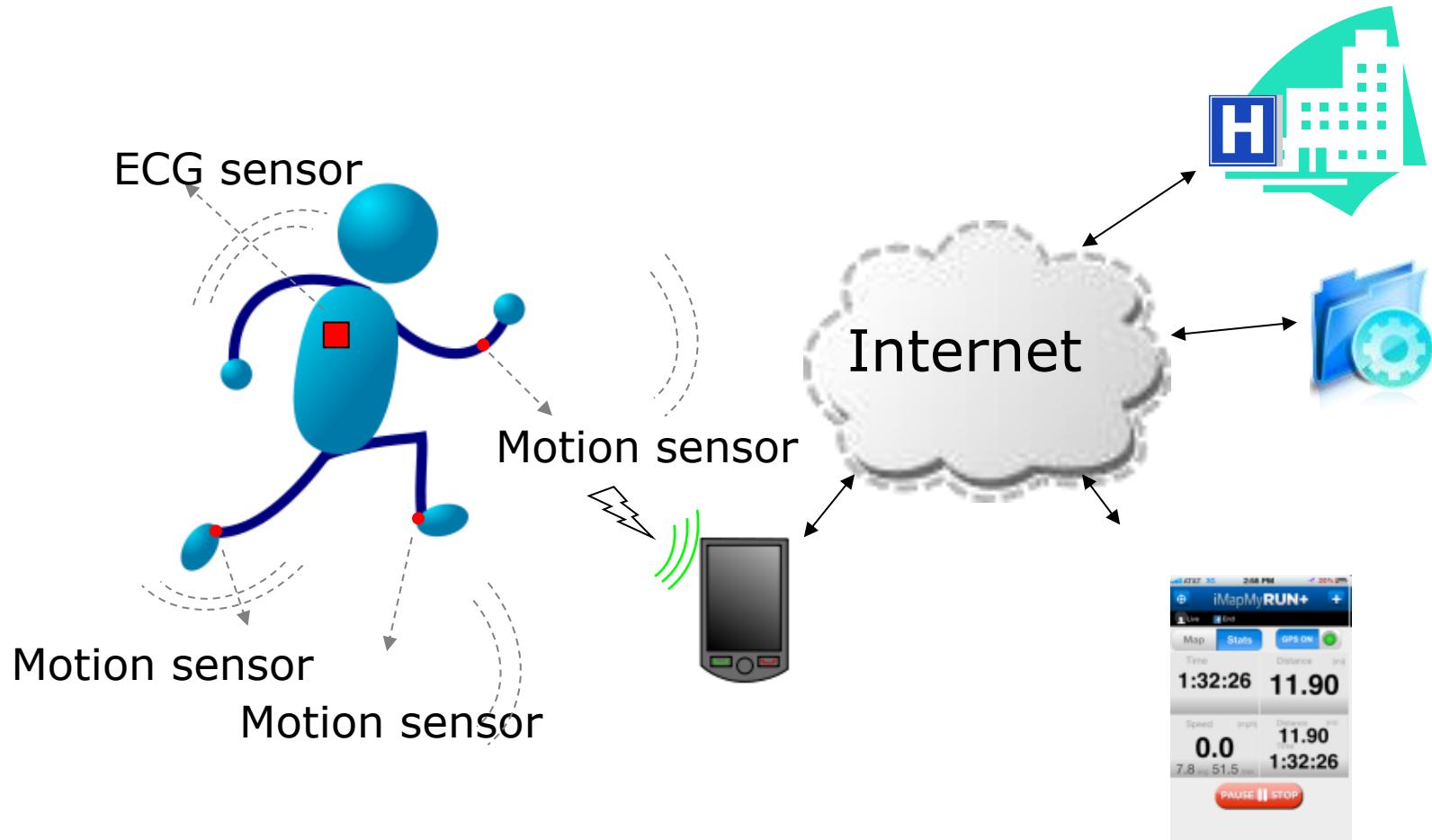


- Internet is present all over the world
- Initially, it was meant to connect human to human

- ❖ Internet of Things connects everything ("things") using existing infrastructure.

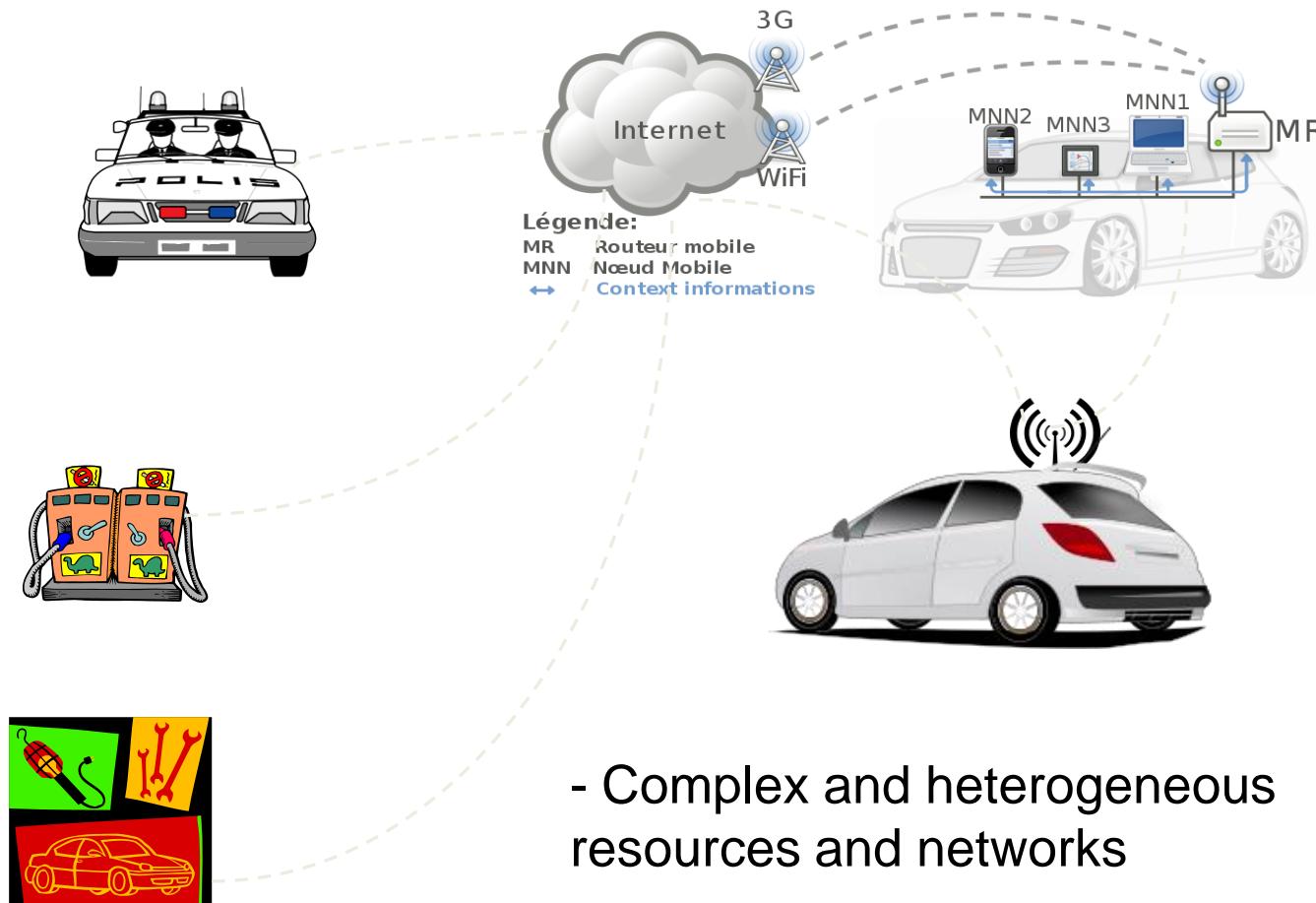
1.2. Evolution of IoT (cont.)

IoT: Human connecting with Things



1.2. Evolution of IoT (cont.)

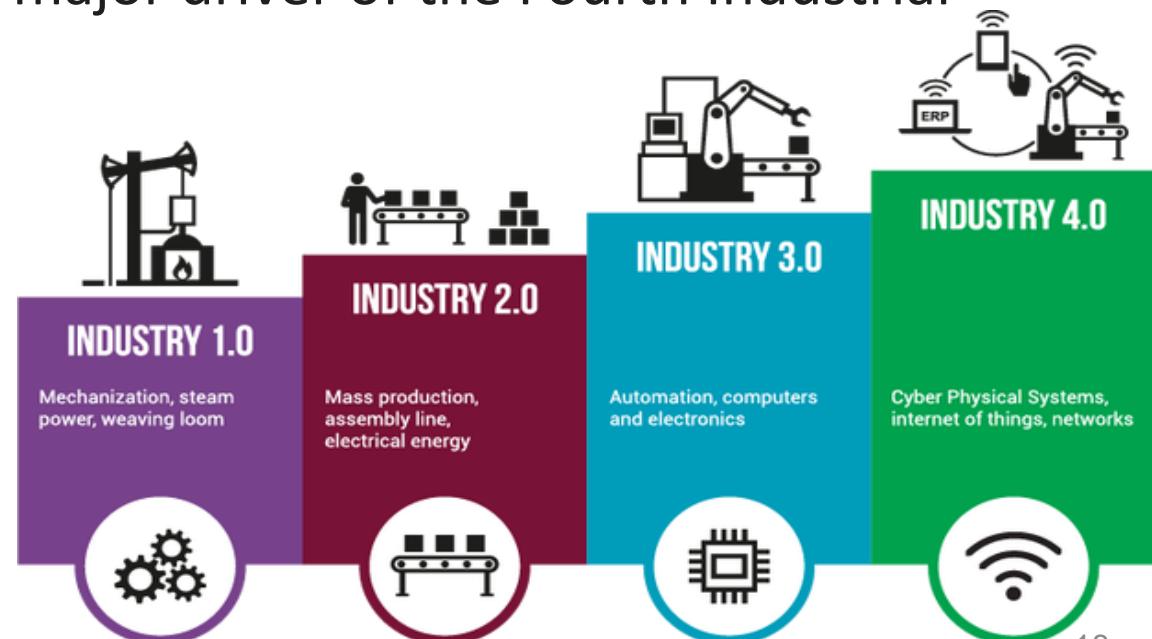
IoT: Things connecting with Things



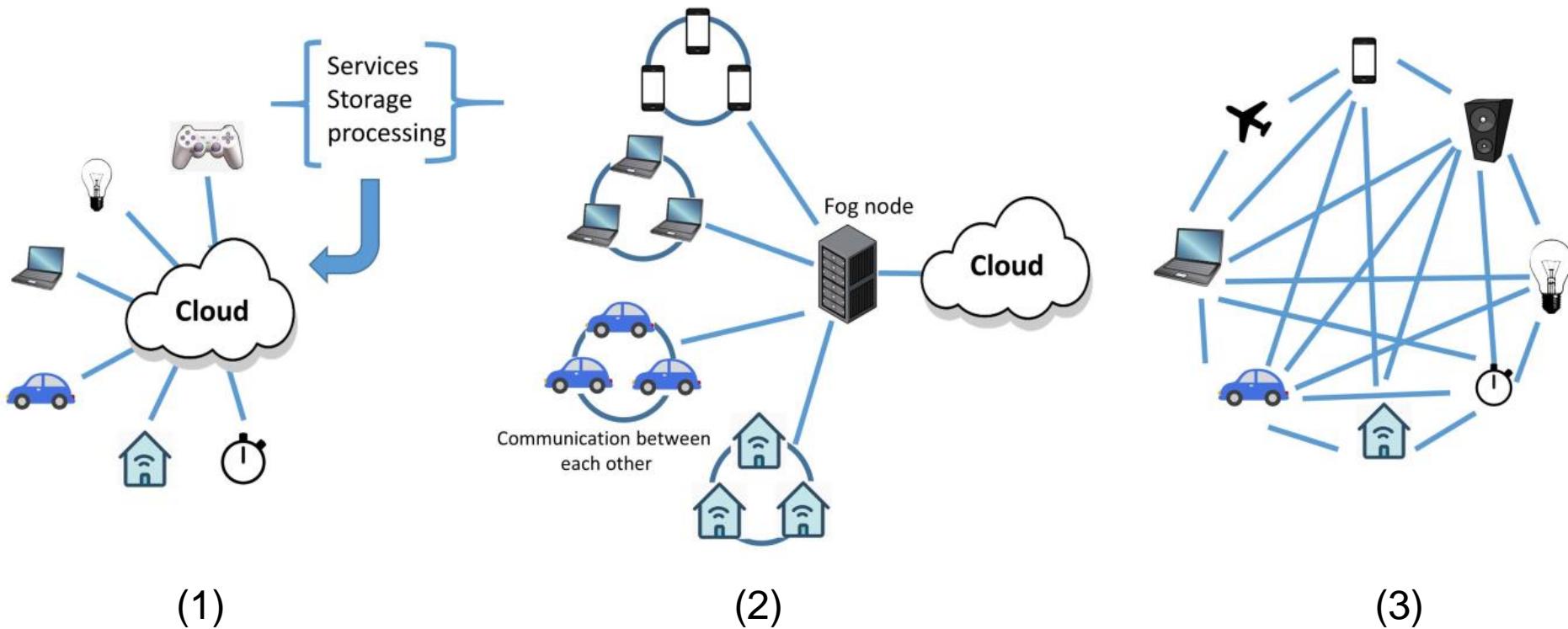
- Complex and heterogeneous resources and networks

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?

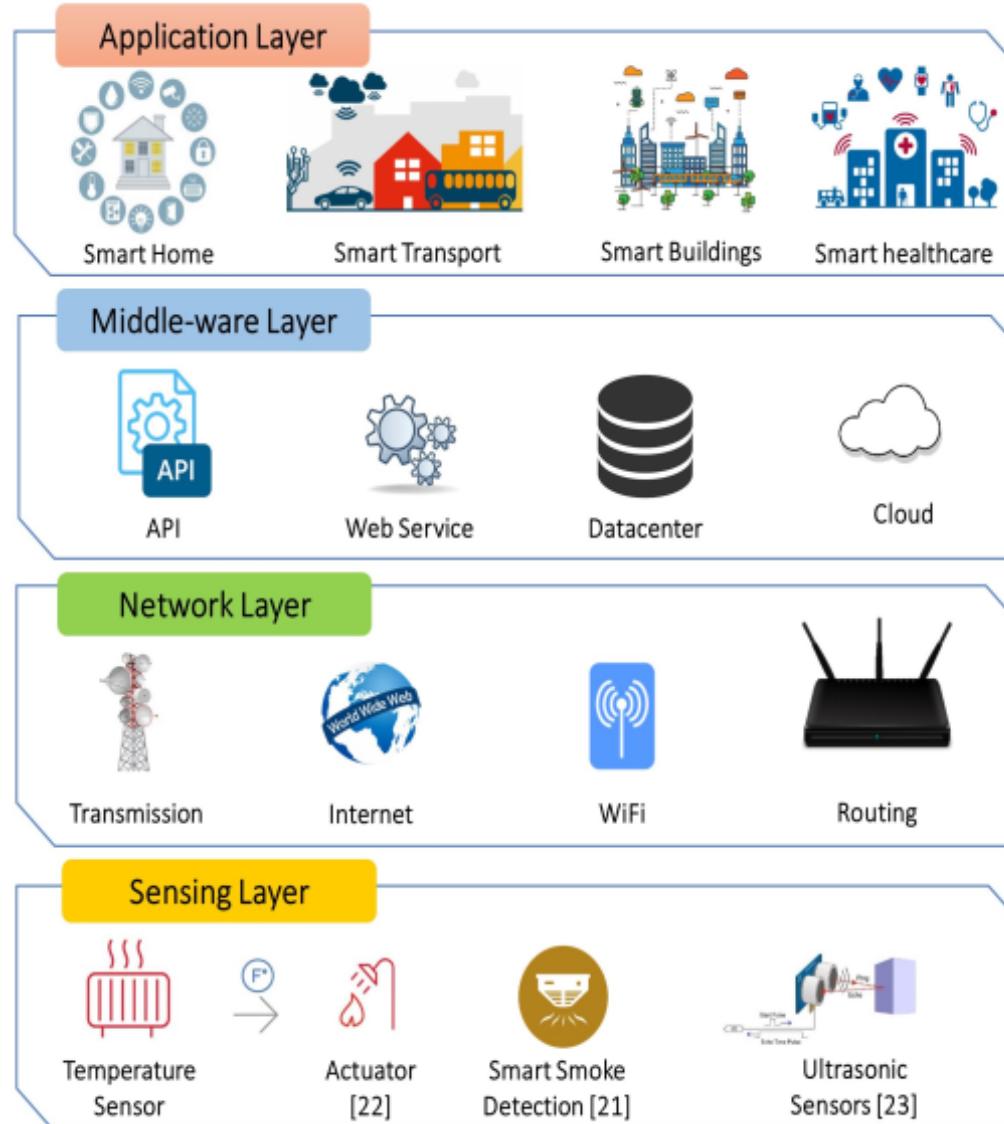


1.3. Architecture of IoT system

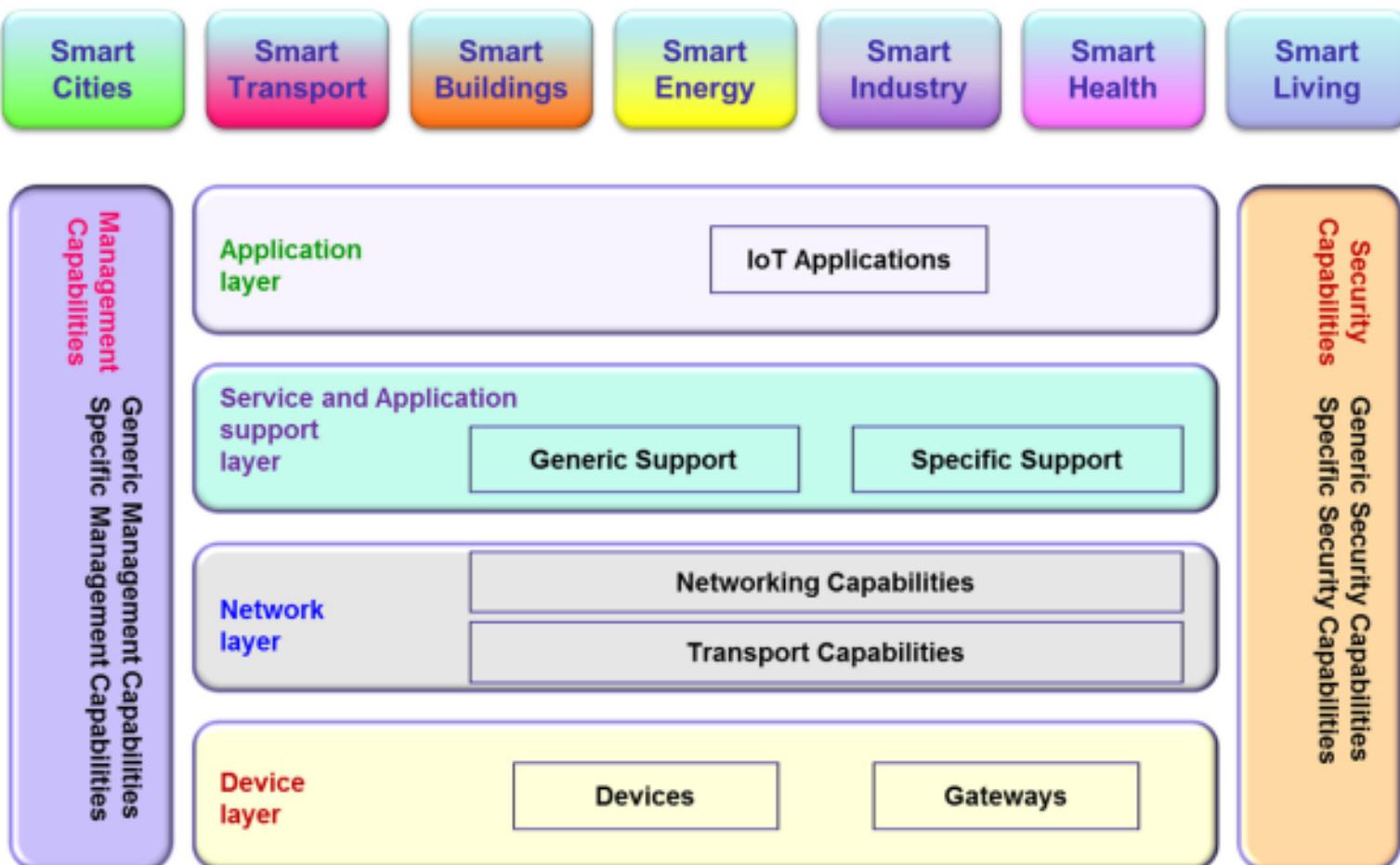


- (1) Simple architecture: Devices connect directly to a server/cloud
- (2) Hierarchical Architecture: Devices connect through an intermediate layer (Fog/Edge node, gateway)
- (3) Future Architecture: "Things" connect directly to other "Things"

1.3. Architecture of IoT system (cont.)



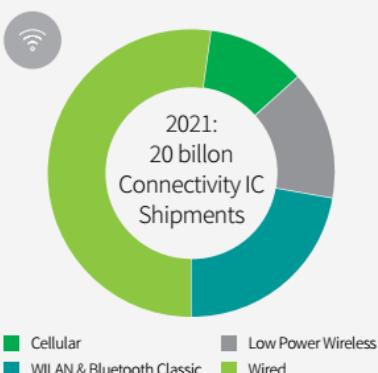
1.3. Architecture of IoT system (cont.)



IoT Layered Architecture (Source: ITU-T)

The four fundamental “pillars” of IoT

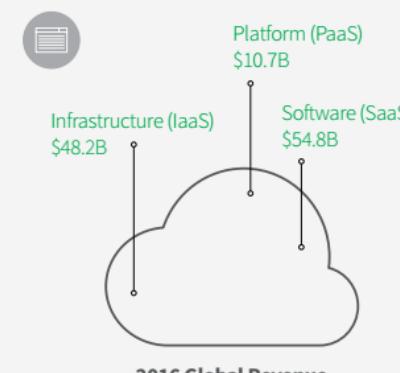
- **Connections:** Ability to connect devices and information
- **Collection:** Capability to gather data through increased connections of devices and information.
- **Computation:** capability enables the transformation of collected data into new functionalities
- **Creation:** Ability to generate new interactions, business models, and solutions



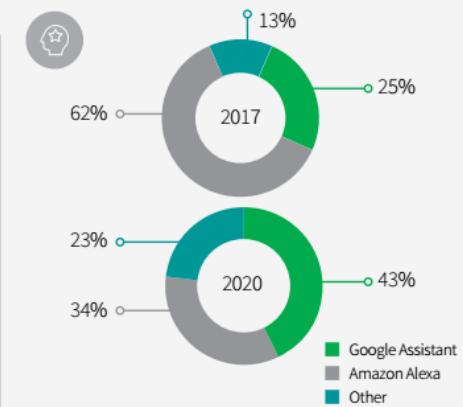
New 5G & NB-IoT capabilities shift demand for cellular activity



Emerging biometrics sensing technologies provide more convenient and stronger security



Cloud & XaaS driving new business models and cost efficiencies



Artificial Intelligence drives Google dominance in digital assistant market

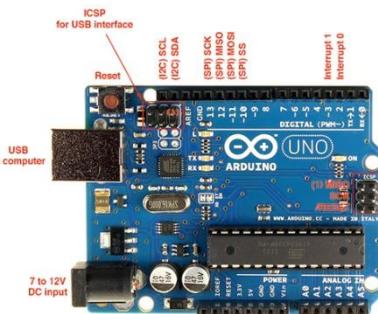
1.4. IoT technologies

- Hardware
- Communication
- Protocols
- Data Analytic
- Cloud Platforms

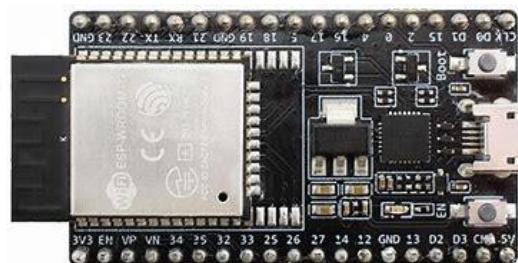
1.4.1. Hardware

Embedded Computers:

- Microcontrollers: 8-bit, 32-bit, without operating system
- Microcontrollers with simple operating system (e.g., FreeRTOS)
- High-performance 32-bit, 64-bit processors with operating systems (Raspbian, Embedded Linux, Ubuntu, Embedded Windows, ...)
- Architectures: AVR, Microchip, ARM, Intel, ...



Arduino Uno



ESP32



Raspberry Pi



Intel Galileo

1.4.2. Sensors

- Sensors can be considered the most crucial component in IoT devices
 - Analog or digital output.
- Sensor modules typically include:
 - Energy/power modules for supply and management of power
 - Sensing modules for data acquisition
 - Communication management component
 - Signal processing through communication management component (RF modules).
 - WiFi, ZigBee, Bluetooth, radio transceiver...

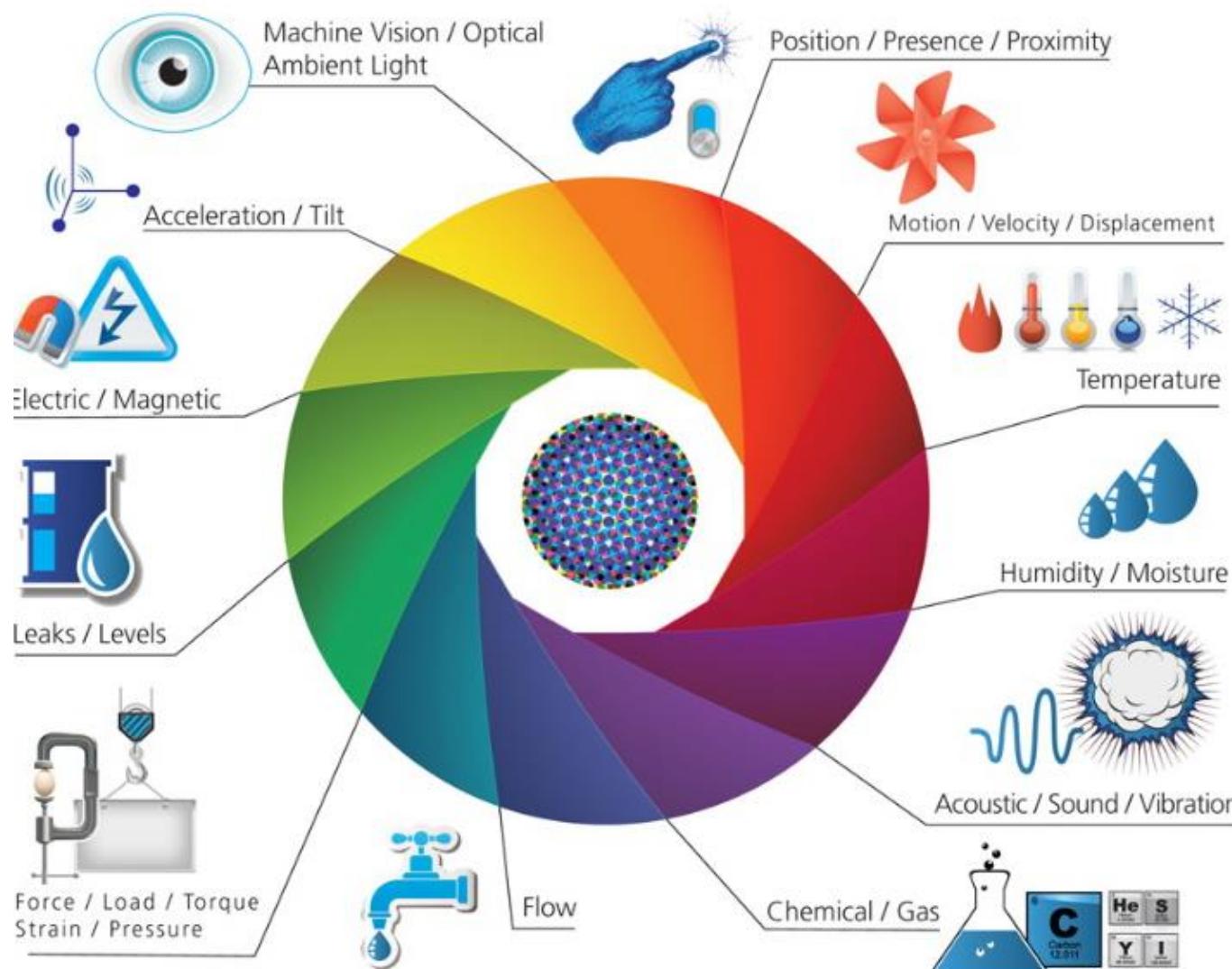


1.4.2. Sensors (cont.)

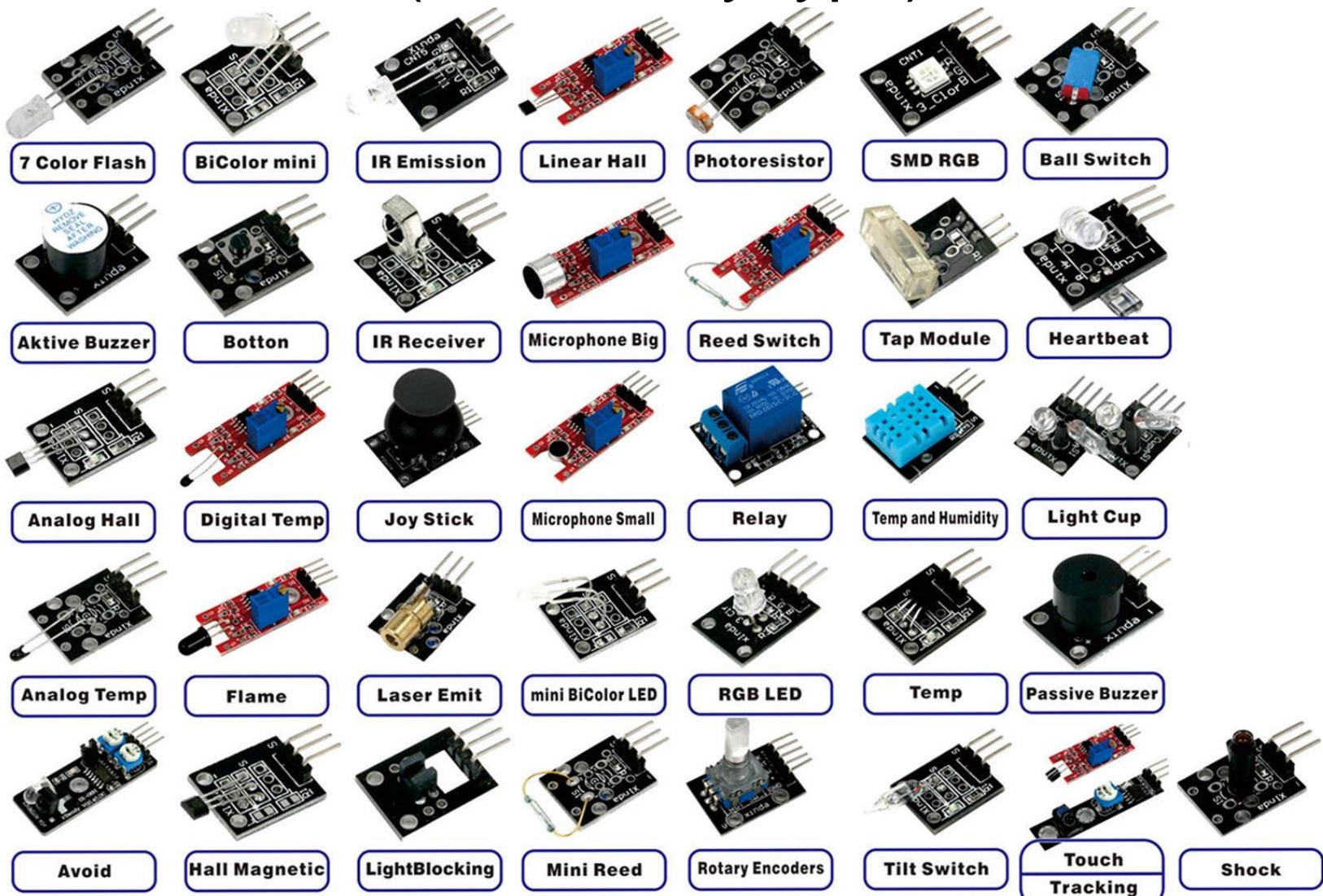
- There are many types of sensors:

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

1.4.2. Sensors (cont.)



1.4.2. Sensors (Laboratory type)



Wearables IoT

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

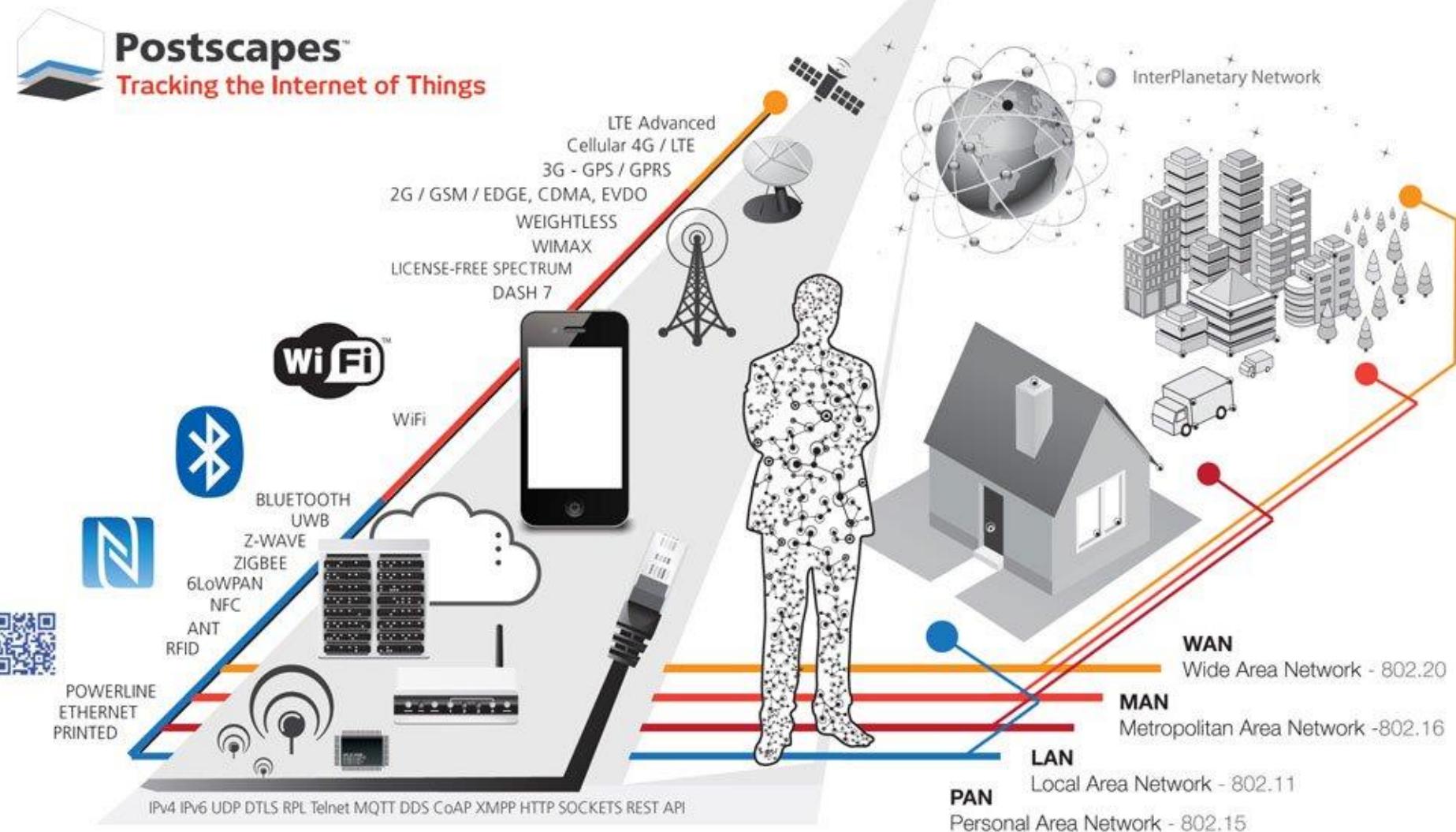


Wearables IoT



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

1.4.2. Communications



1.4.2. Communications (Cont.)

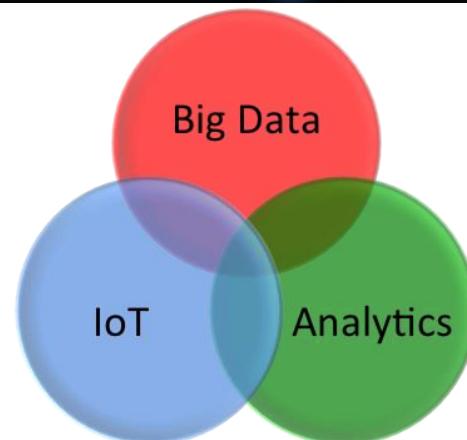
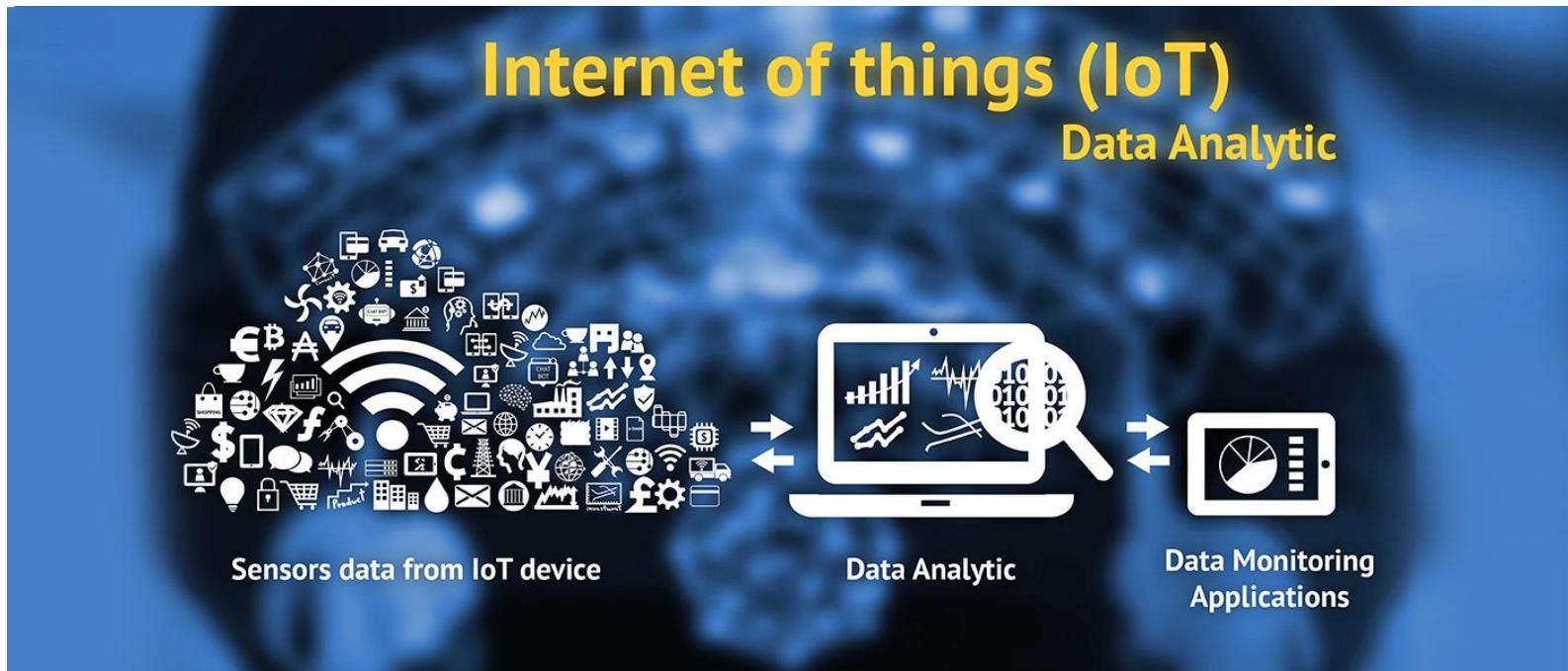
- Common communication standards for IoT:
 - NFC and RFID
 - Bluetooth
 - WiFi
 - GSM, 3G/4G/LTE, LTE-A

1.4.2. Communications (Cont.)

- Protocols for IoT applications:
 - HTTP (HyperText Transfer Protocol)
 - RESTful HTTP (Representational State Transfer)
 - 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/>

1.4.3. Data Analytic



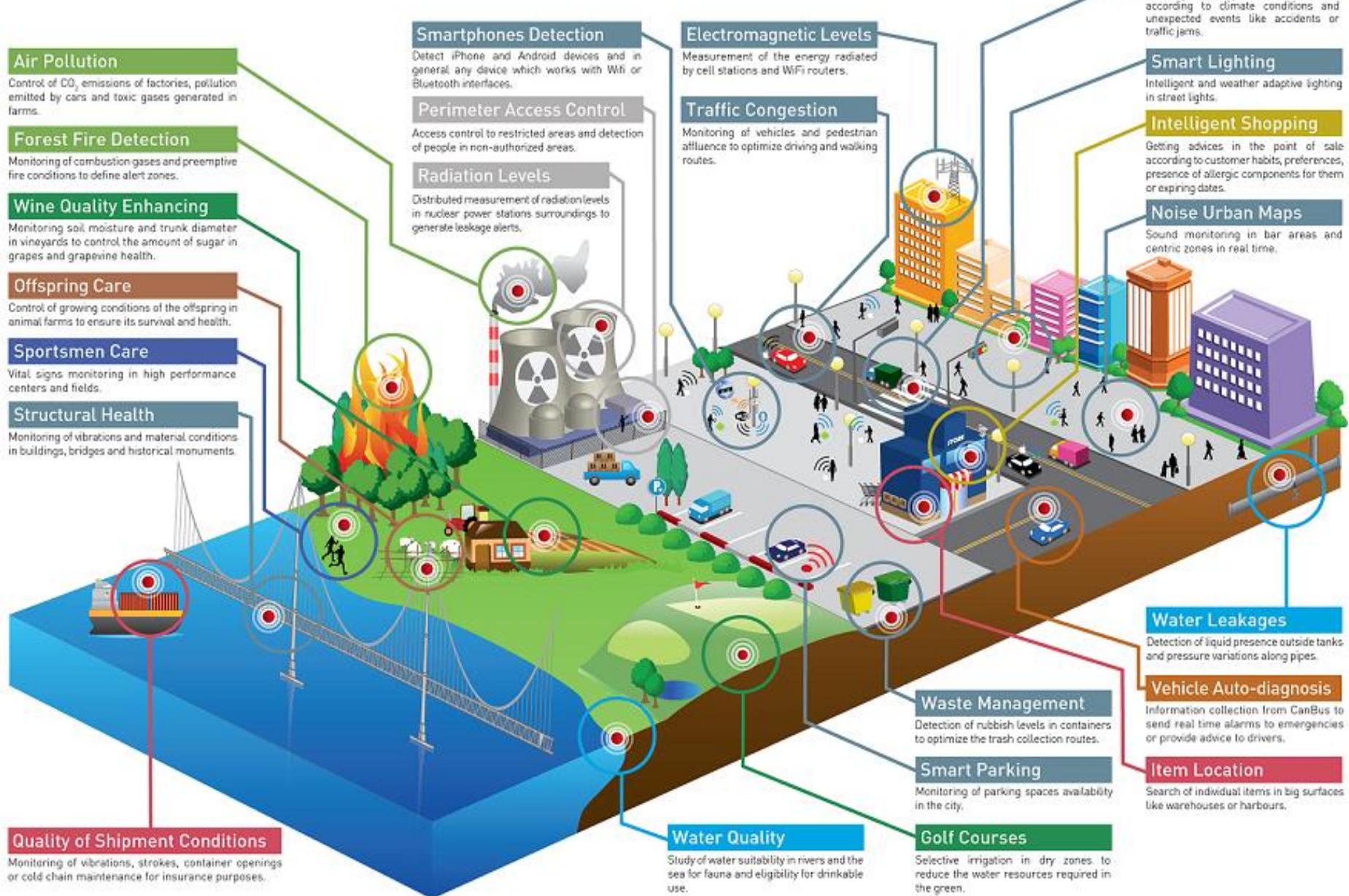
1.4.4. IoT Technologies: Cloud Platforms

- Some cloud service platforms in IoT:
 - IBM BlueMix
 - AWS IoT
 - Google Cloud IoT
 - Azure IoT



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

1.5. IoT applications



1.5. IoT applications (cont.)



CONFIDENTIAL Not For distribution All Contents © 2014 Arria Systems

Top Industrial IoT Applications

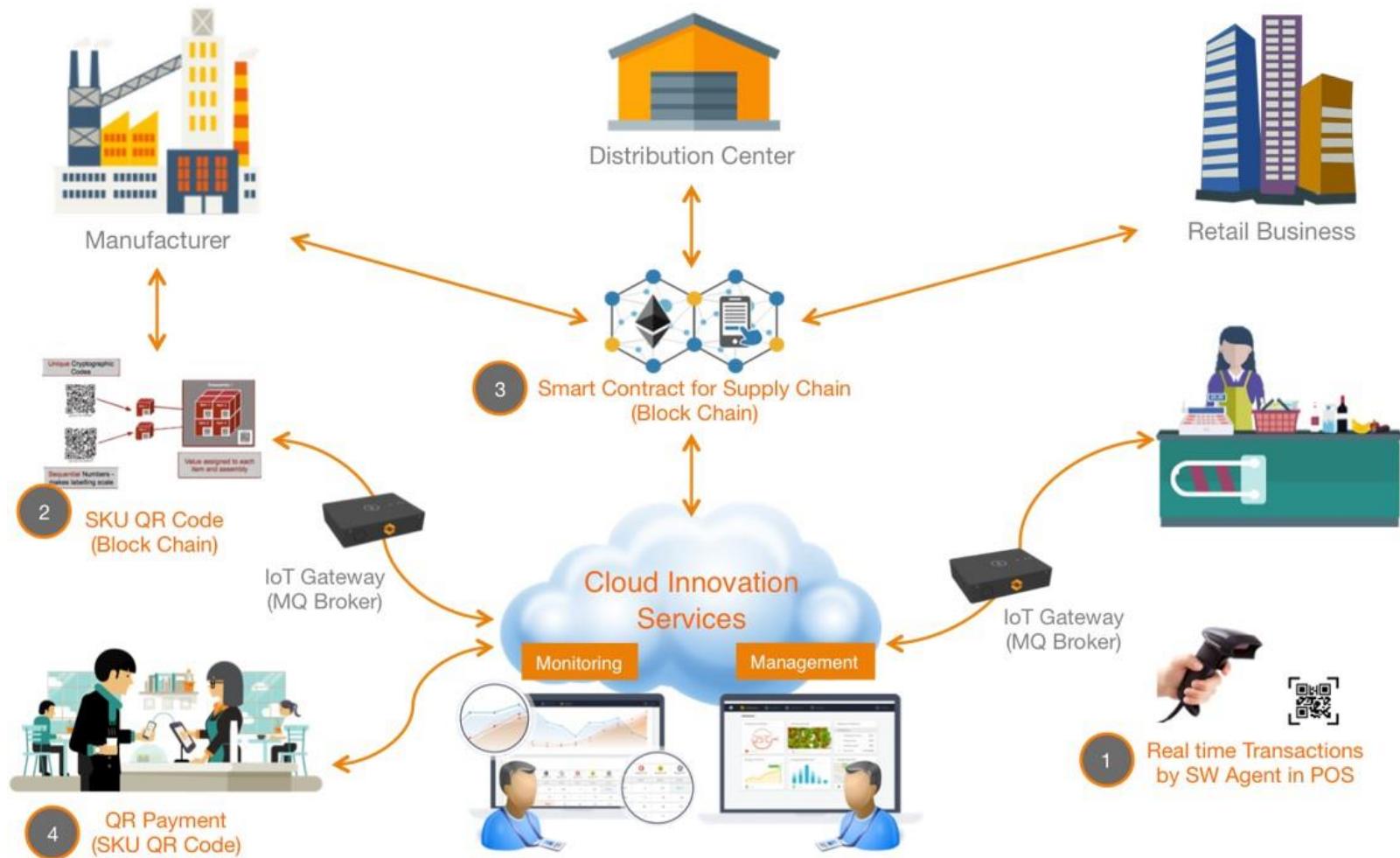
- Healthcare
- Smart Retail
- Smart Building/Smart Home
- Smart Agriculture
- Smart Utilities (Power Energy, Water)

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



INTRODUCING
amazon go

Smart Retail: Shopping



Scenario: shopping

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



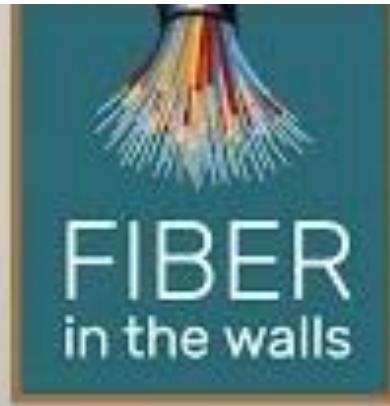
(1) When entering the doors, scanners will identify the tags on her clothing.



(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

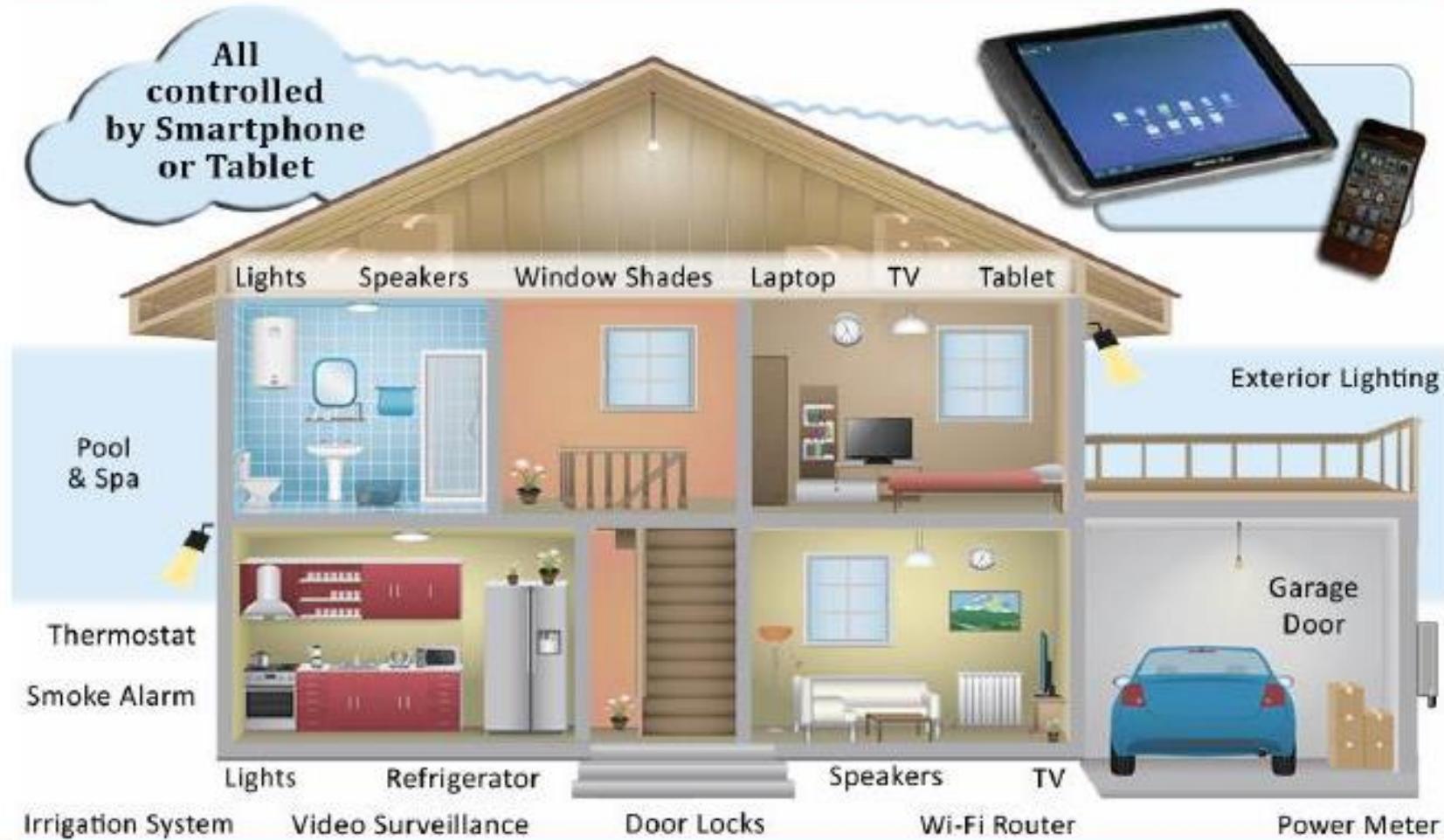


Honeywell

What is a Smart Building?

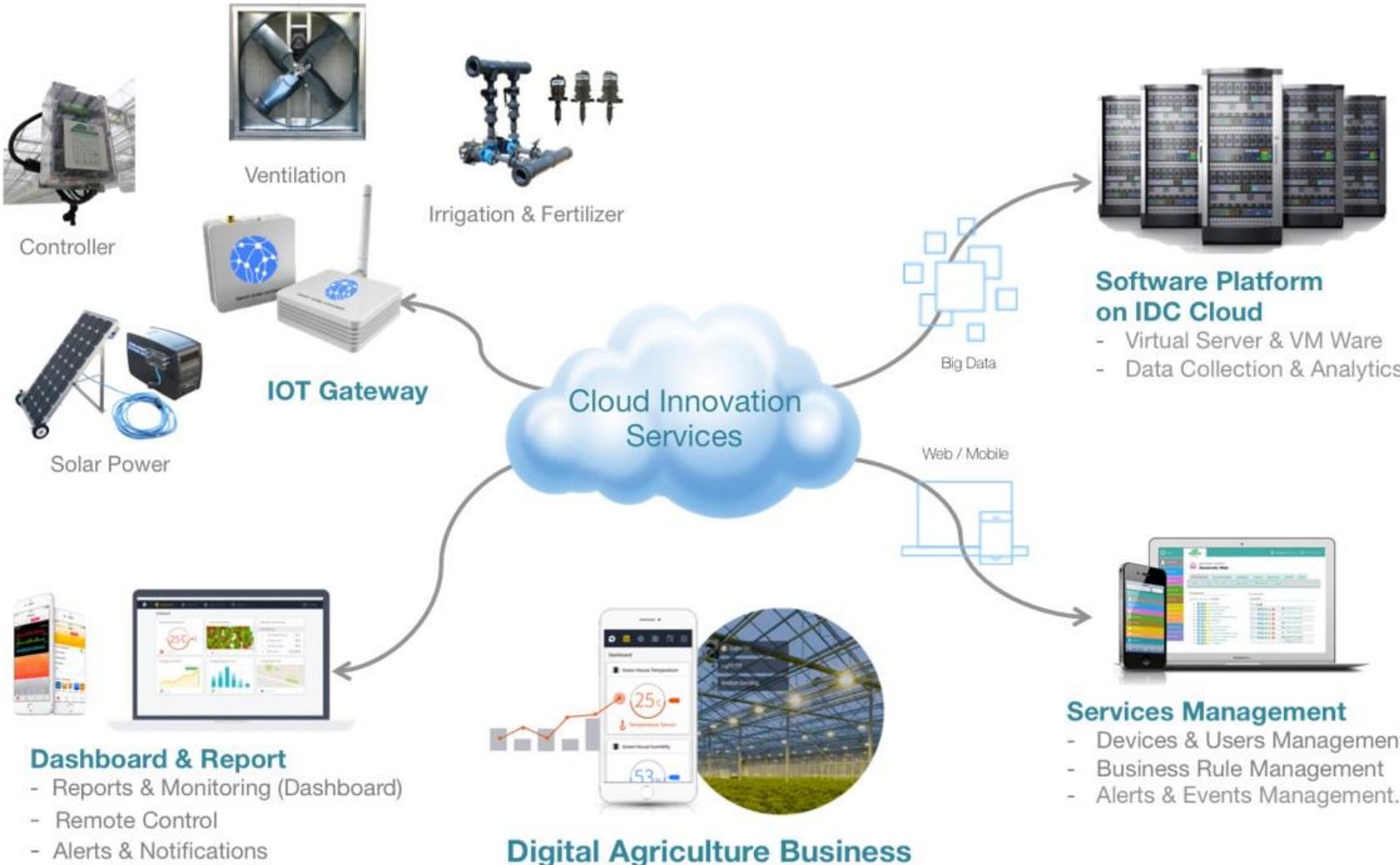
Smart Home

Home Automation



Source: Raymond James research.

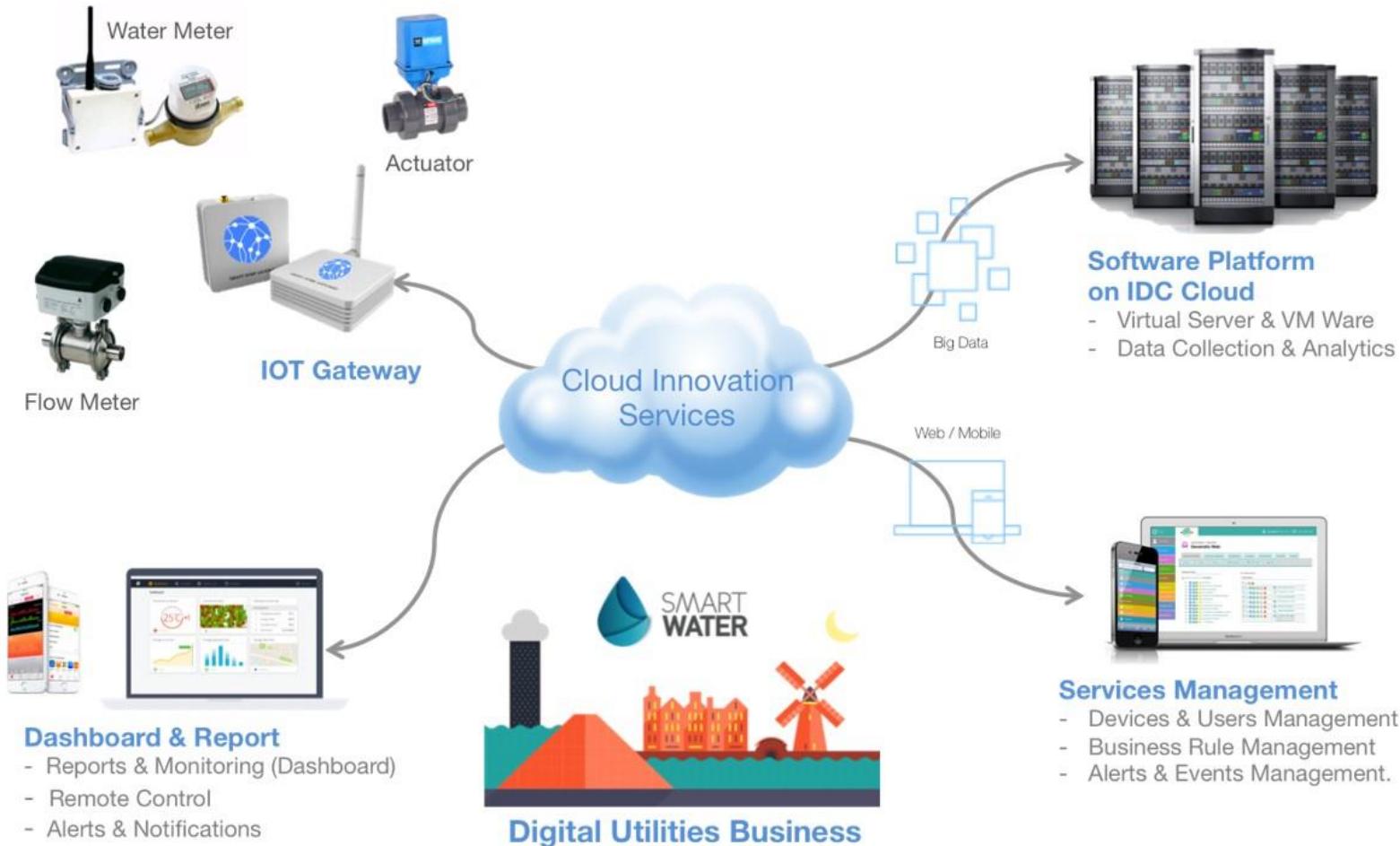
Smart Agriculture



<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

Commercial products using IoT



Commercial products using IoT

Smart Lighting

Control your bulbs one at a time or altogether. Find just the right shade of white. Pick that perfect tone to match the moment. Or recreate any color from a photo.

<http://meethue.com/>



Smart A/C

Aros learns from your budget, location, schedule, and usage to automatically maintain the perfect temperature and maximize savings for your home.

<https://www.quirky.com/shop/752-aros-smart-window-air-conditioner>

Smart Sleep System

Visualize your sleep cycles, understand what wakes you up, and compare nights. From the palm of your hand you can control your personalized wake-up, and fall-asleep programs.

<http://www.withings.com/us/withings-aura.html>



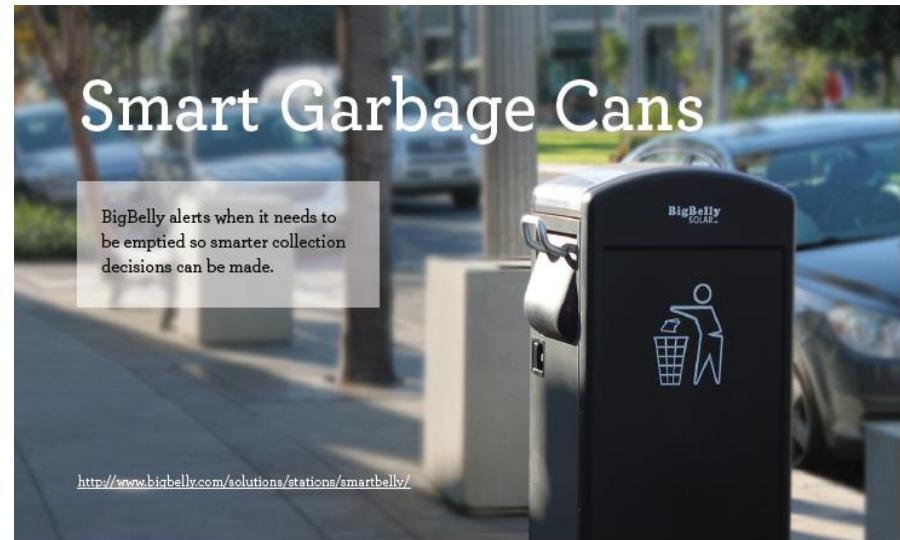
Smart Weather Station

The Netatmo Weather Station allows you to use indoor temperature, relative humidity and CO₂ readings to live in a healthier home.

<http://www.netatmo.com/en-US/product/weather-station/>



Commercial products using IoT



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?

1.6. Challenges of IoT

- Many different technologies
 - Advantages + Challenges?
- Challenges for governments in managing rapid technological advancements
 - Example: Uber, Grab
- Issue regarding privacy and security
 - Example: Facebook
- Lack of a central authority for governance and control.
 - Common issue with internet services
- Vulnerability to be attacked on the Internet:
 - Example: IP cameras

1.6. Challenges of IoT (cont.)

■ Connectivity problem:

- Currently, IoT relies on a server/client model for authentication and connecting devices within the network. This model can handle hundreds or thousands of devices. However, difficulties arise when the number of devices reaches millions or billions within the network
- Without considering appropriate network throughput design, bottlenecks may occur during data exchange at the server
- Tasks can be off-loaded to the edge devices for execution
- IoT networks will require devices capable of data analysis, machine learning, and data collection.

1.6. Challenges of IoT (cont.)

- **Legacy infrastructure** – Issue of deployment on existing infrastructure:
 - IoT devices, existing network infrastructure, and new technologies being integrated (brownfield deployment)
 - Companies face the challenge of integrating new technology devices with existing network infrastructure.

1.6. Challenges of IoT (cont.)

- **Dealing with non-standard communication protocol:**
 - The connecting networks will need to cope with an unprecedented surge in data from devices and sensors
 - Controlling, processing, and storing data will increase in line with the rise in input data load. Additionally, data, as it increases in size and frequency, must remain readily available for data analysis.

1.6. Challenges of IoT (cont.)

- **IT/OT convergence**
 - The integration/convergence of Information Technology (IT) and Operational Technology (OT) in industrial Internet of Things (IIoT) applications. An example is in a power plant
 - IT focuses on data-centric operations, while OT is centered on monitoring events. IoT blurs this distinction by simultaneously monitoring devices and generating a large volume of data
 - Industrial enterprises will need to adjust their processes to accommodate IIoT devices and data

1.6. Challenges of IoT (cont.)

- **Get actionable intelligence from data**
 - The value of data increases when smart insights are derived from the augmented data
 - IoT analysis will need to work with unstructured data, large volumes of real-time data, and even outlier data.

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.

Content

- Chapter 1. Overview of IoT
- Chapter 2. IoT Technologies
- Chapter 3. IoT Application and Programming
- Chapter 4. IoT Safety and Security
- Chapter 5. Designing and Building IoT Systems

Chapter 2. IoT Technologies

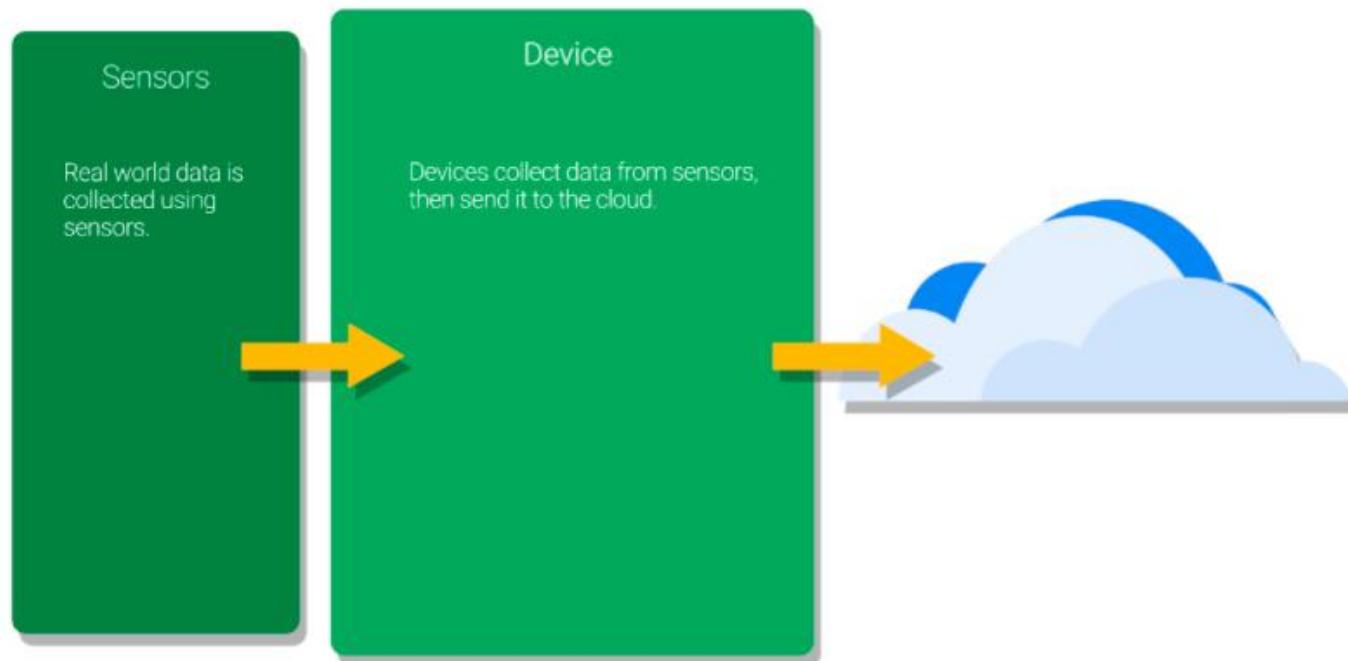
- 2.1. Sensors and devices
- 2.2. Some communication standards for IoT
- 2.3. Protocols for IoT applications
- 2.4. IoT cloud platform

2.1. Sensors and devices

- Sensor and devices:
 - Sensors and IoT devices are simply called devices with the implicit inclusion of sensors that are connected
 - In many situations, when referring to “devices” it can be understood as including both devices and sensors

Sensors and devices

- Sensors observe the changes around and send information about those changes to the device
- The devices collect data from the sensors and send it to the cloud/server



Sensors and devices

- The devices may be limited in computing resources (CPU, limited memory, ...)
- The devices can communicate privately without directly connecting to the cloud platform, for example through Bluetooth Low Energy (BLE)
- The devices can store, process and analyze data before sending it to the cloud

Types of Sensors

- According to the power supply requirement

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

- According to the type of signal

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

- According to the type of measuring device

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

Sensor selection

- Sensor selection needs to consider the importance of the factors and the priority level in the overall design
- **Durability:**
 - The durability of the sensor must be considered based on its operating environment.
 - Ensure that the device operates reliably for a significant period of time without having to incur repair or replacement costs.
 - Example: waterproof temperature sensor used for weather station

Sensor selection

- **Accuracy:**
 - The accuracy level must meet the demand (high accuracy comes with high cost)
 - Example: Measuring the temperature of a warehouse can accept an accuracy of +/- 2 degrees. Measuring the temperature of a medical equipment system requires an accuracy of +/- 0.2 degrees

Sensor selection

- **Versatility :**
 - The sensor can operate with the fluctuations of the surrounding environment
 - Example: The sensor for the weather station can work in summer and winter conditions. It is not feasible with the type of sensor that only works in indoor environments

Sensor selection

- **Power Consumption:**
 - Depending on the situation, some devices may require low power consumption, using energy-saving features
 - Example: sensors or devices using solar batteries need to design sleep mode to save energy, wake up when needed to collect and transmit data

Sensor selection

- **Consider the special environment:**
- Example: The sensor in monitoring water quality in aquaculture requires the sensor to be placed in a water environment (pH, DO, TDS, ...) different from the sensors based on sample analysis.

Sensor selection

- **Cost:**
 - IoT networks usually consist of hundreds, thousands of devices and sensors.
 - The design needs to consider the cost of installation, maintenance, replacement, etc.

IoT Devices

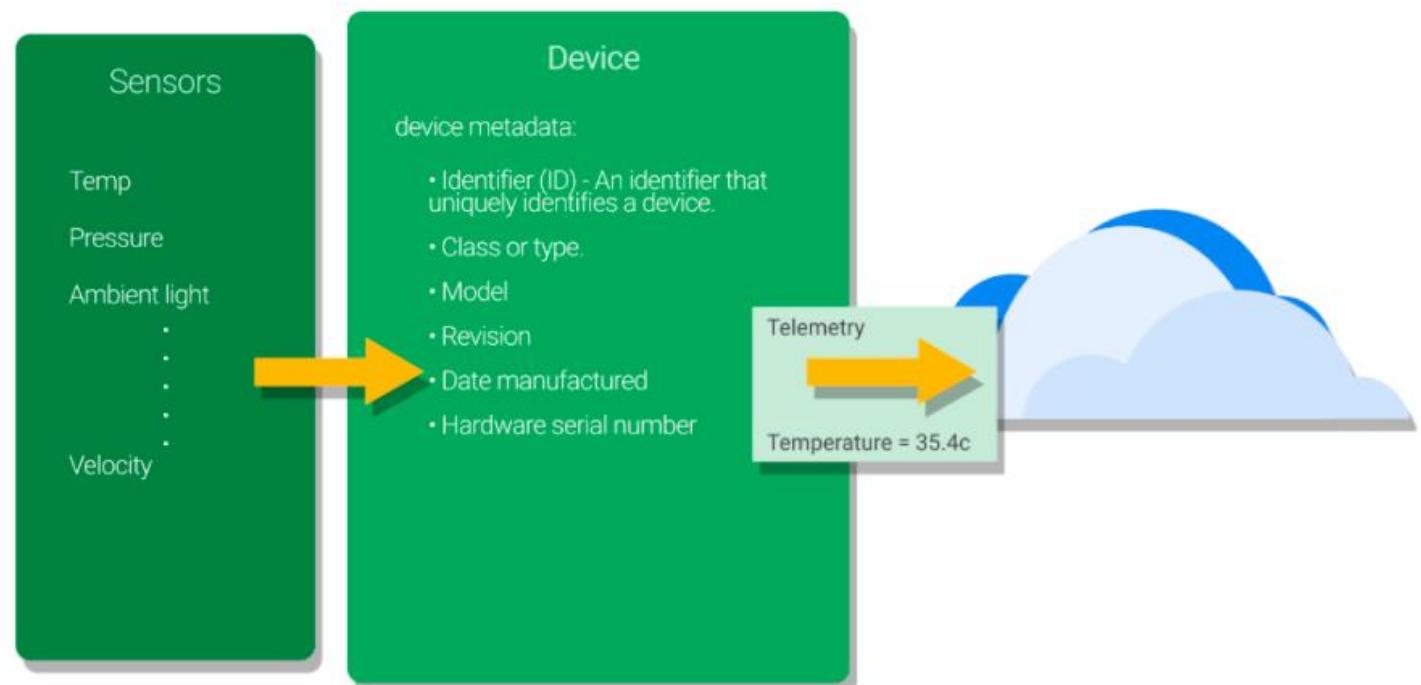
- A “Thing” in the “Internet of Things” is a processing unit that can connect to the internet to exchange data with the cloud
- The devices are often called "smart devices" or "connected devices"
- The device communicates with two types of data: telemetry (collected data) and state (device status)

Data type

- Each device can provide or use some types of data for different types of backend system
- **Device metadata:** Metadata contains descriptive information about the device (usually not changing much). For example
 - Identifier (ID) of device
 - Class or type of device
 - Model
 - Revision
 - Date manufactured
 - Hardware serial number

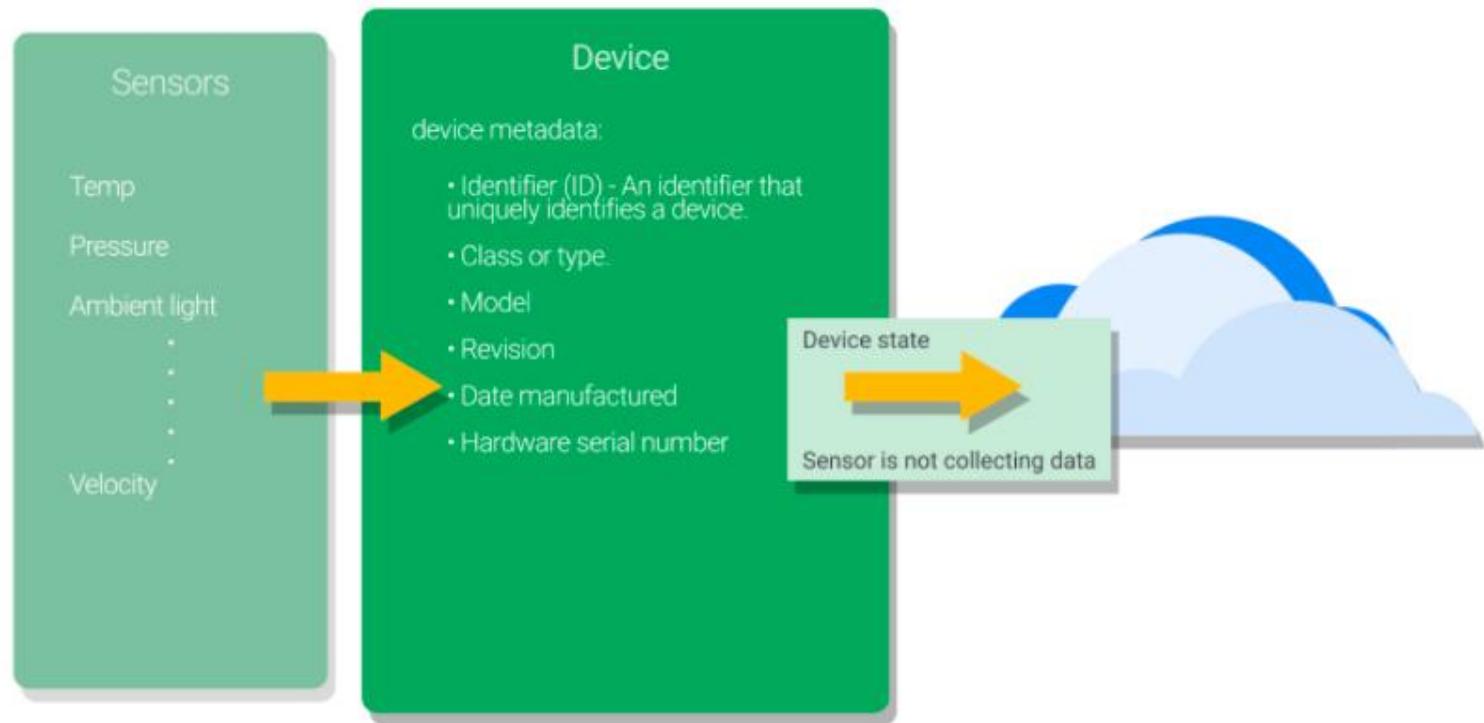
Data type

- **Telemetry:**
 - Data collected by the device through sensors is called telemetry.
 - It is read-only data (from the surrounding environment)



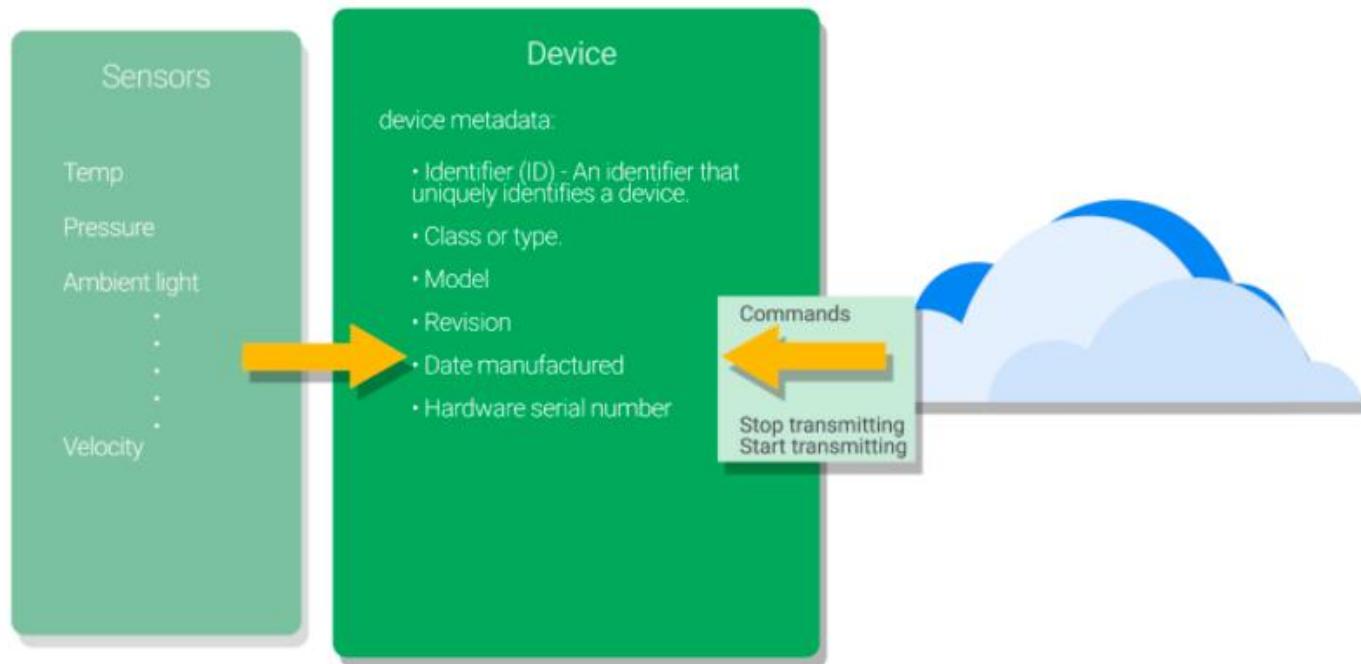
Data type

- State information:
 - Describes the current state of the device. It can be read/write, update (but less frequently)



Device Commands

- Commands: actions that are performed by the device.
- Examples:
 - Rotate 360 degrees to the right.
 - Turn on/off the light
 - Increase the data transmission frequency



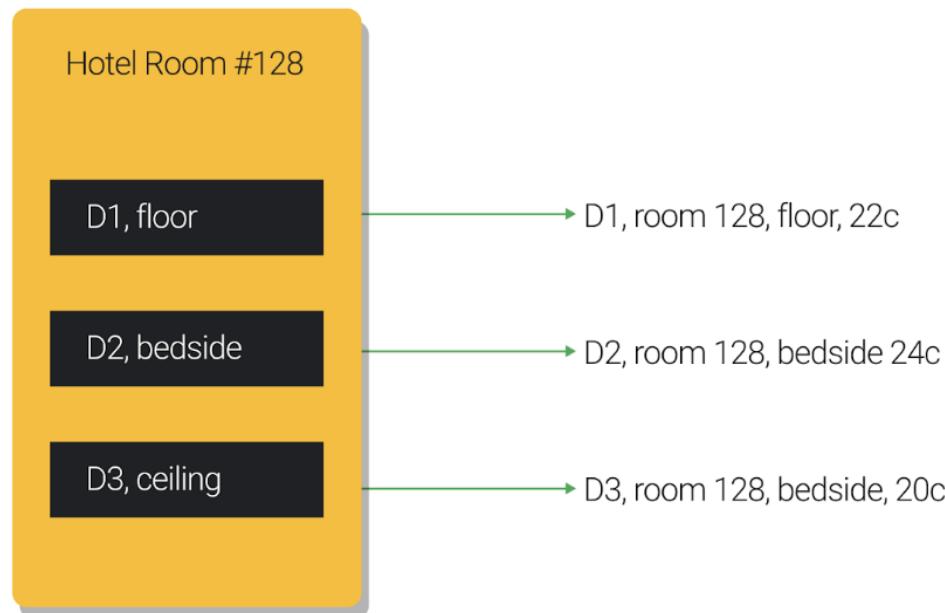
Definition of device in IoT system

- In the IoT system, the definition of a device can change depending on the project's needs.
- Each device can be considered as a separate entity, or a device can be responsible for a group of sensors.

Definition of device in IoT system - Example

- Monitoring the temperature of hotel rooms
- **Solution 1:** Each room has 3 sensors: one near the floor, one near the bed, and one near the ceiling

One hotel room, 3 sensors, 3 devices



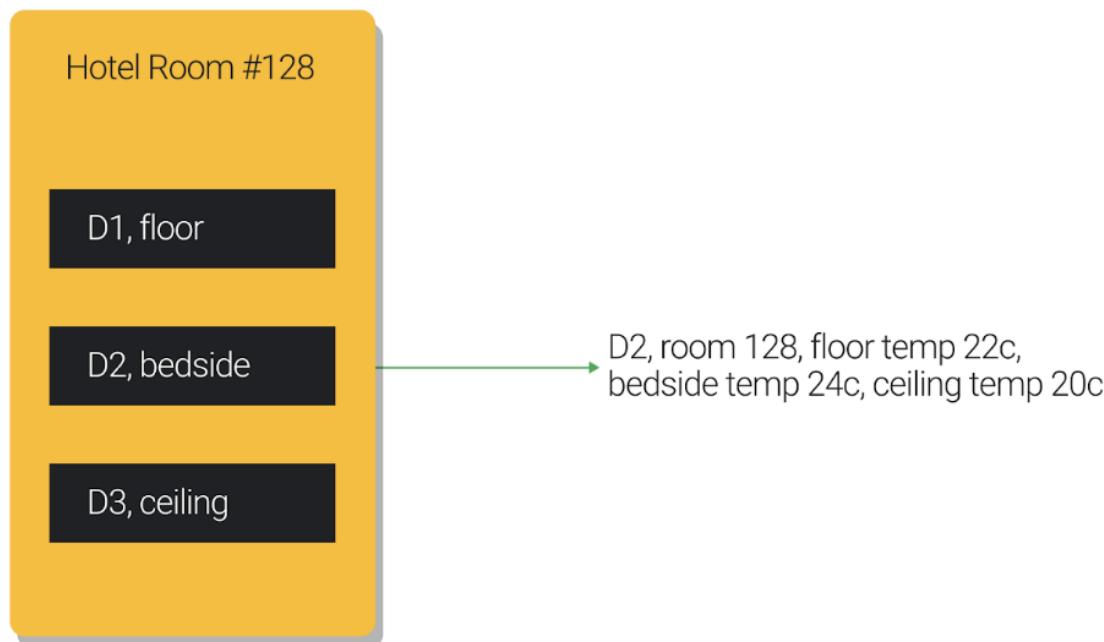
Definition of device in IoT system - Example

- **Solution 1:** Data is sent to the cloud as three different devices, each sending a temperature information to the 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 }

Definition of device in IoT system - Example

- **Solution 2:** Each room uses one device that is responsible for sending data from three sensors

One hotel room, 3 sensors, 1 device



Definition of device in IoT system - Example

- **Solution 2:** One device with three sensors transmits data to the cloud. The device can also process the data to find the average temperature..
 - {deviceID: "dh28dskja", "room": 128, "temp_floor": 22, "temp_ceiling": 24, "temp_bedside": 23, "average_temp": 23 }
- The choice of solution depends on the intention of using the information at the present time and the potential in the future

2.2. Some communication standards for IoT

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

Some communication standards for IoT

- RFID (Radio-frequency identification)



<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

Some communication standards for IoT

- NFC (Near-Field Communications)



Frequency: 13.56 MHz

Range: < 0.2 m

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

Some communication standards for IoT

- Bluetooth



Frequency: 2.4GHz

Range: 1-100m

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

Some communication standards for 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/>

Some communication standards for 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/>

Some communication standards for IoT

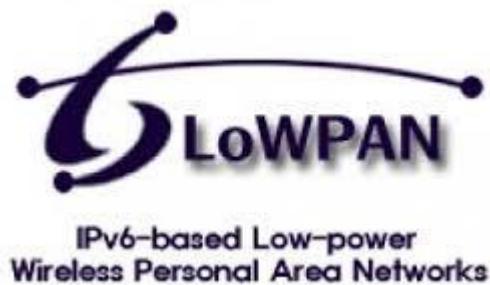
- ZigBee



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

Some communication standards for IoT

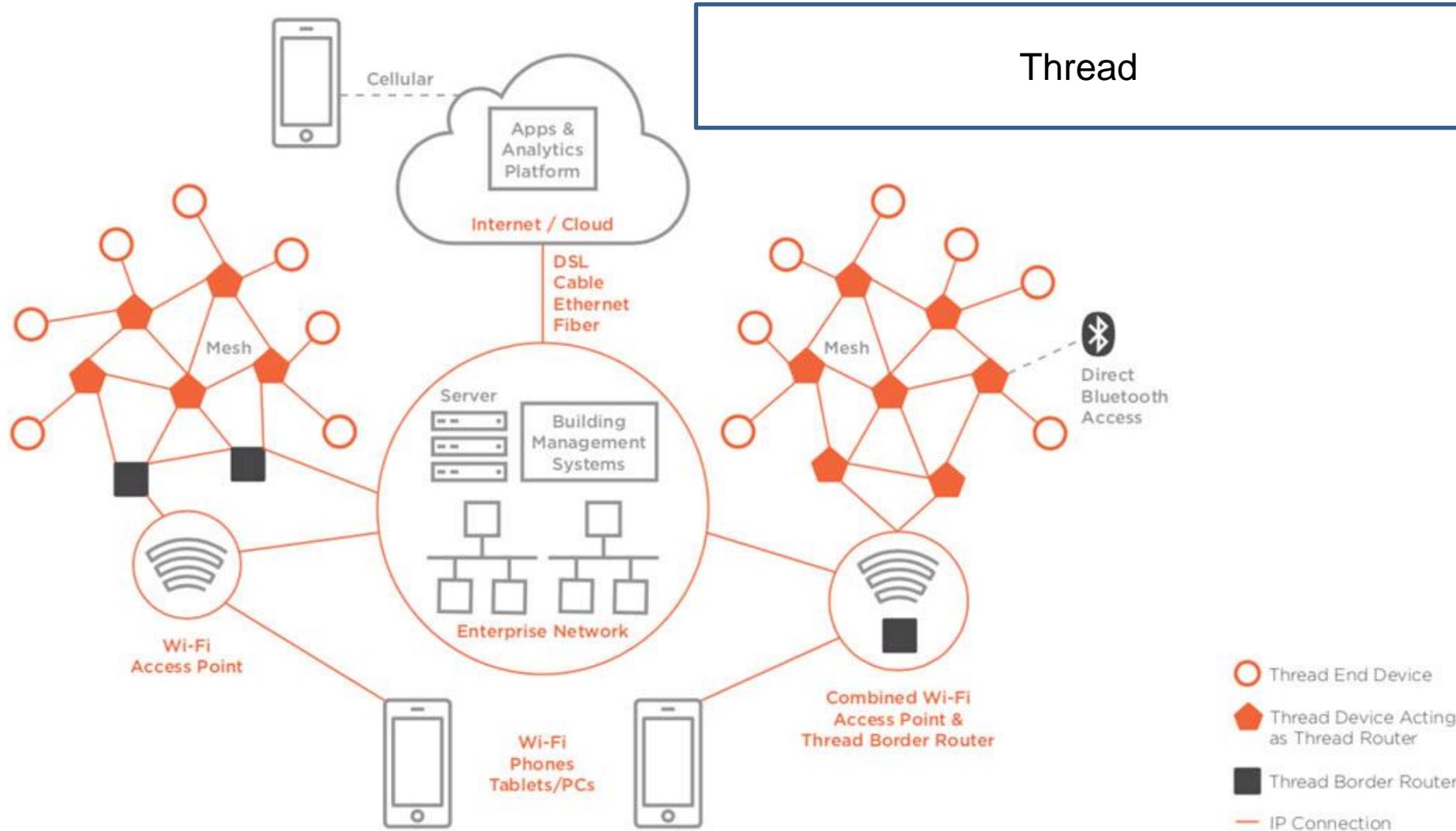
- 6LoWPAN
- IPv6 protocol over low-power wireless PANs



Some communication standards for IoT

- Thread: A new IP protocol based on IPv6 platform designed for automation in smart homes and buildings
- Introduced in mid-2014 by Theard Group, Thread protocol follows different standards, such as IEEE802.15.4, IPv6 and 6LoWPAN, and offers an IP-based solution for IoT applications
- Standard: Theard, based on IEEE802.15.4 and 6LowPAN
- Frequency 2.4GHz
- Reference: <https://smarthomekit.vn/thread-protocol/>

Some communication standards for IoT



Some communication standards for IoT

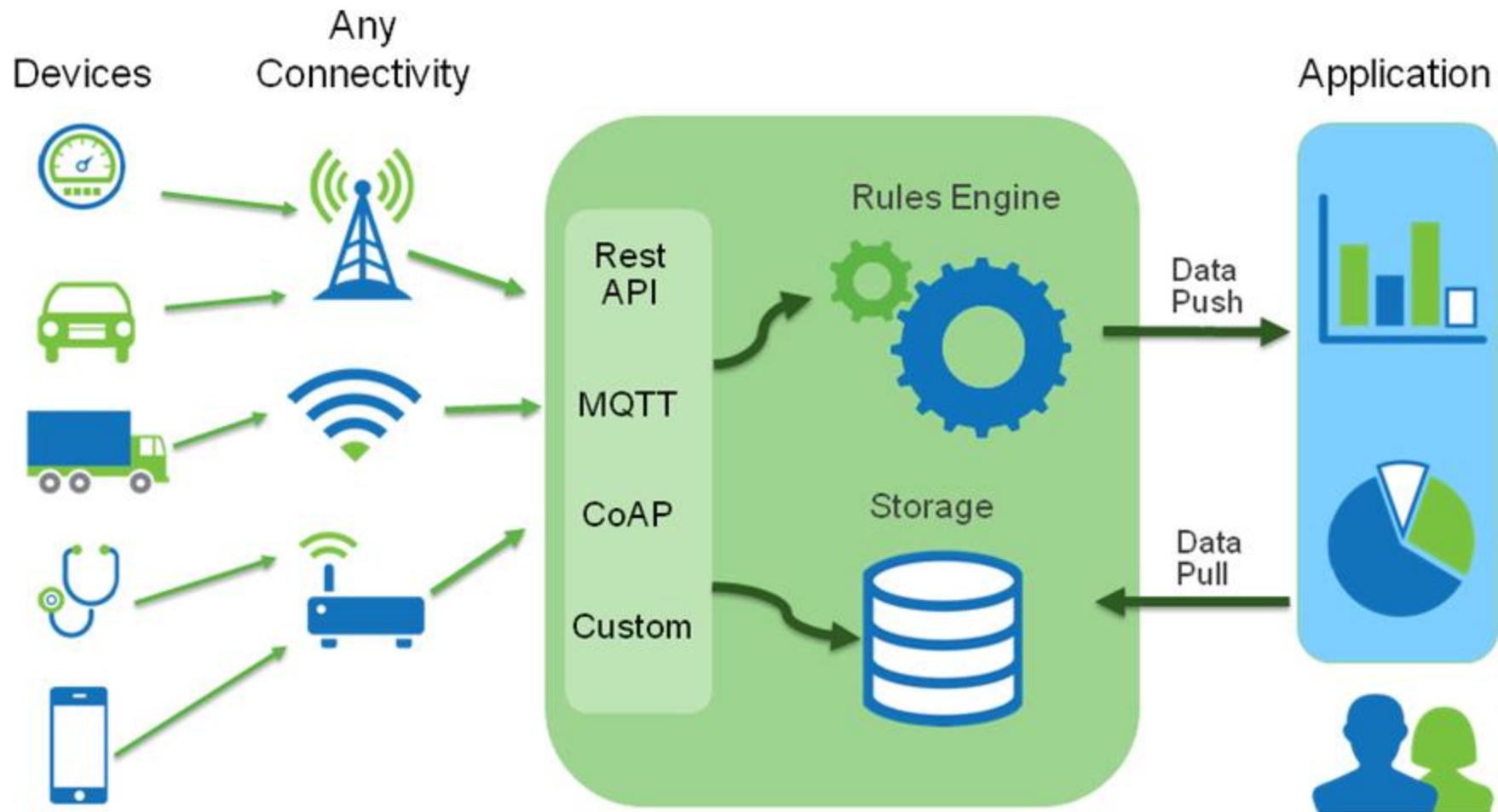
Additional:

- [3G](#)
- [4G LTE](#)
- [ANT](#)
- [Dash7](#)
- [Ethernet](#)
- [GPRS](#)
- [PLC / Powerline](#)
- [QR Codes, EPC](#)
- [WiMax](#)
- [X-10](#)
- [802.15.4](#)
- [Z-Wave](#)
- [Zigbee](#)

Backbone: IPv4, IPv6, TCP/UDP

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

2.3. Protocols for IoT applications



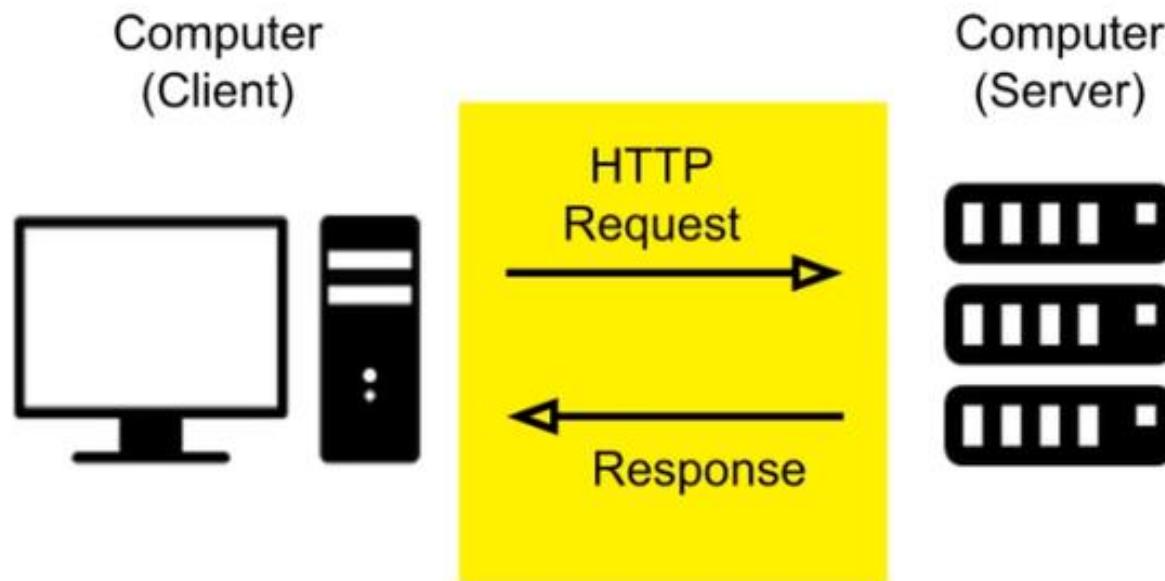
Protocols for IoT applications

- HTTP, HTTPS
- REST API HTTP (Representational State Transfer)
- MQTT (Message Queue Telemetry Transport)
- AMQP (Advanced Message Queue Protocol)

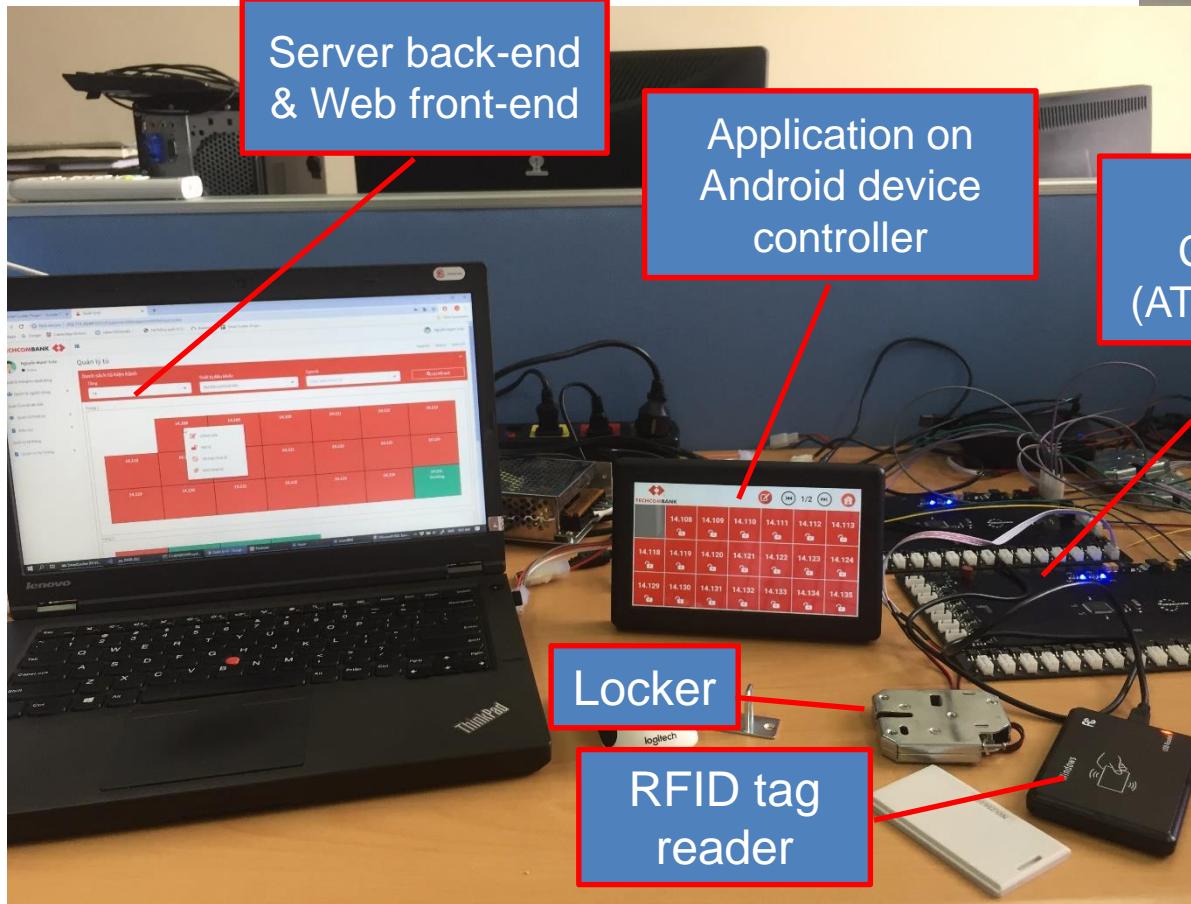
<https://www.postscapes.com/internet-of-things-technologies/>

2.3.1. HTTP

HTTP = HyperText Transfer Protocol



Example of HTTP



Smart Locker system

Example of making an http request with Java (1)

- Using the HttpURLConnection library in Java to perform an http request to an http REST api
- **Step 1:** Create an *http connection*, set the properties for the *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");
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}
```

Example of making an http request with Java (2)

- **Step 2:** Write the data to be sent to the output stream of the request. Data can be packaged using 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();
```

Example of making an http request with Java (3)

- **Step 3:** Receive http response from server. Check the return code and read data from the input stream of the 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;
}
```

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

Use Postman

The screenshot shows the Postman application interface. At the top, there's a navigation bar with icons for Home, Workspaces (dropdown), Reports, Explore, a search bar labeled "Search Postman", and various user account and settings icons.

A yellow banner at the top of the main content area says "Working locally in Scratch Pad. Switch to a Workspace".

The main workspace contains several API requests listed horizontally:

- POST https://... (red dot)
- GET https://... (red dot)
- GET https://... (red dot)
- POST https://... (red dot)
- GET https://... (red dot)
- GET https://... (red dot)
- POST https://... (red dot)

Below the requests, a specific POST request is selected for "GetUserInfo":

POST https://203.171.20.94:8080/api/AccessControl/GetUserInfo

On the right side of the selected request, there are "Save" and "Send" buttons. The "Send" button is highlighted in blue.

The left sidebar contains icons for Overview, Environments, and a list of environments. The "Overview" icon is currently selected.

The main content area has tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings, and Cookies. The "Params" tab is active, showing a table for "Query Params".

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

At the bottom, there's a "Response" section and a footer with "Find and Replace", "Console", "Runner", and "Trash" buttons.

Example of making an http request with Python

```
import requests
URL = "https://maps.googleapis.com/maps/api/geocode/json"
location = "Hanoi University of Science and Technology"
PARAMS = {'address':location}
# sending get request and saving the response as response object
r = requests.get(url = URL, params = PARAMS)
# extracting data in json format
data = r.json()

# extracting latitude, longitude and formatted address
# of the first matching location
latitude = data['results'][0]['geometry']['location']['lat']
longitude = data['results'][0]['geometry']['location']['lng']
formatted_address = data['results'][0]['formatted_address']
# printing the output
print("Latitude:%s\nLongitude:%s\nFormatted Address:%s"
    %(latitude, longitude, formatted_address))
```

Exercise 1

- HTTP request

- Create API server
 - Request Body là json object

```
{"deviceId": "8a0fc66a61a959f6", "qrCodeId": "a652d57094b7  
590b0dea115b156c07098abdea87", "qrCodeValue": "P2249824418  
2551944"}
```

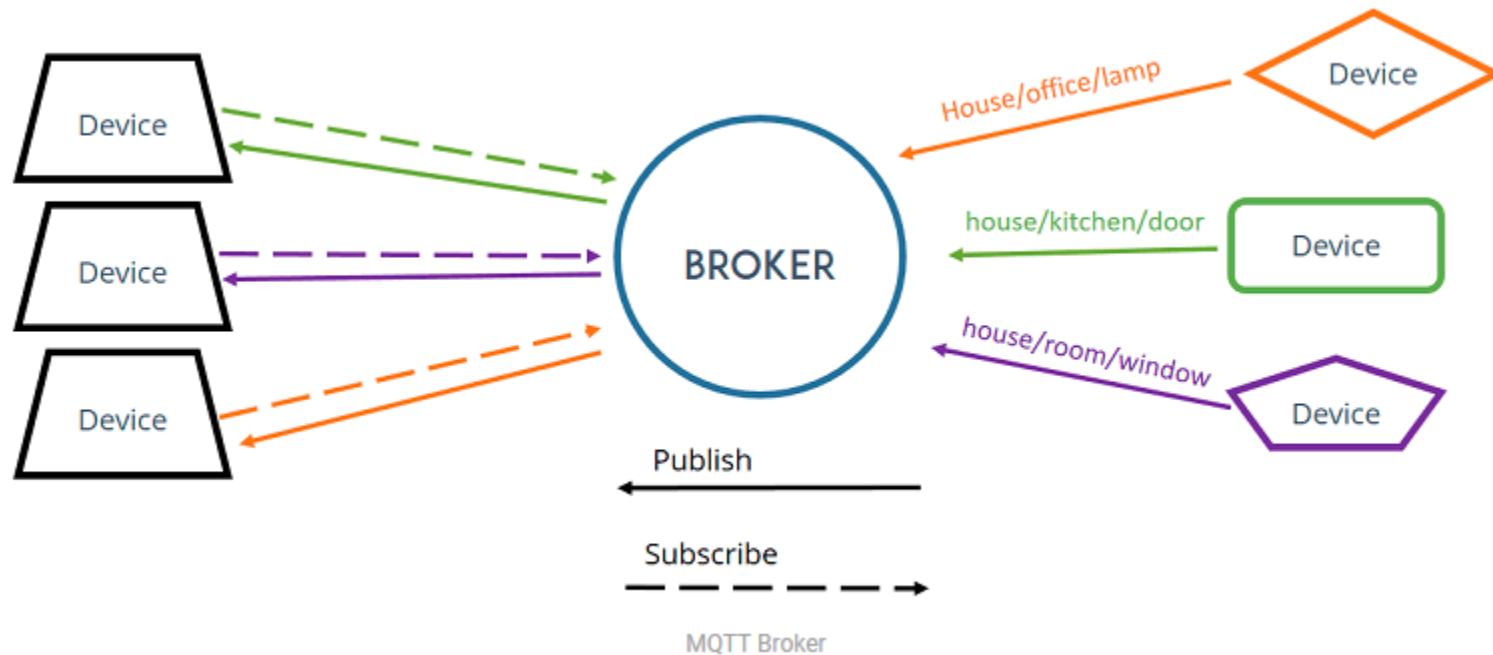
- Using Postman for testing

2.3.2. MQTT Protocol

- The Messaging and Data Exchange Protocol of the IoT
- MQTT (Message Queuing Telemetry Transport):
 - Message-passing protocol based on the publish/subscribe model
 - Uses low bandwidth, high reliability and can operate in unstable network conditions.

MQTT protocol

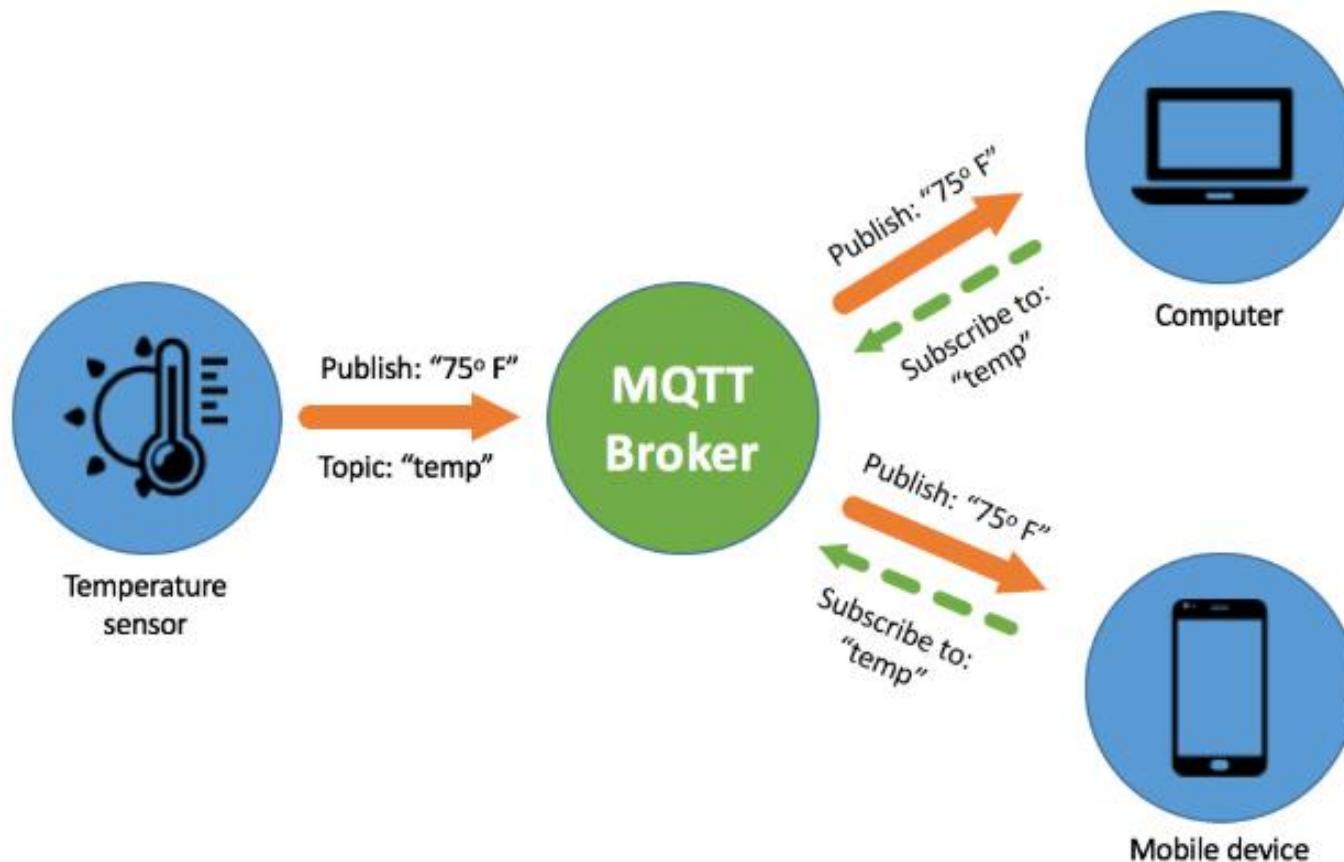
- The high-level architecture of MQTT consists of 2 main components:
 - Broker
 - Clients



MQTT Protocol

- **Publish/Subscribe:**
 - In a system using the MQTT protocol, many nodes (called mqtt clients - abbreviated as clients) connect to an MQTT server (called broker).
 - Each client will subscribe to a few channels (topics), for example “/client1/channel1”, “/client1/channel2”. This registration process is called “**subscribe**”. Each client will receive data when any other devices sends data to the registered channel.
 - When a client sends data to that channel, it is called “**publish**”.

MQTT Protocol



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

MQTT Protocol

- **QoS (Quality of Service):**
 - There are 3 QoS options when “publish” and “subscribe” :
 - **QoS0** Broker/client will send data exactly once, the sending process is confirmed by only TCP/IP protocol, (“fire and forget”)
 - **QoS1** Broker/client will send data with at least one confirmation from the other end, meaning there may be more than one confirmation of receiving data.
 - **QoS2** Broker/client ensures that when sending data, the receiver only receives exactly once, this process must go through 4 handshake steps.

MQTT Protocol

- Using the client tool: MQTTBox
 - <https://www.hivemq.com/mqtt-toolbox/>
 - Publish/Subscribe data

MQTT client tool: MQTTBox

The screenshot shows the MQTTBox client interface connected to a HiveMQ broker at `mqtt://broker.hivemq.com:1883`. The left panel is used for publishing messages, and the right panel shows the published message details.

Publish Panel (Left):

- Topic to publish:** `ktmt_collect`
- QoS:** `0 - Almost Once`
- Retain:**
- Payload Type:** `Strings / JSON / XML / Characters`
- Payload:**

```
{  
  "id":11,  
  "packet_no":123,  
  "temperature":25,  
  "humidity":60,  
  "tds":1100,  
  "pH":7.0}
```
- Buttons:** `Publish`

Published Message Panel (Right):

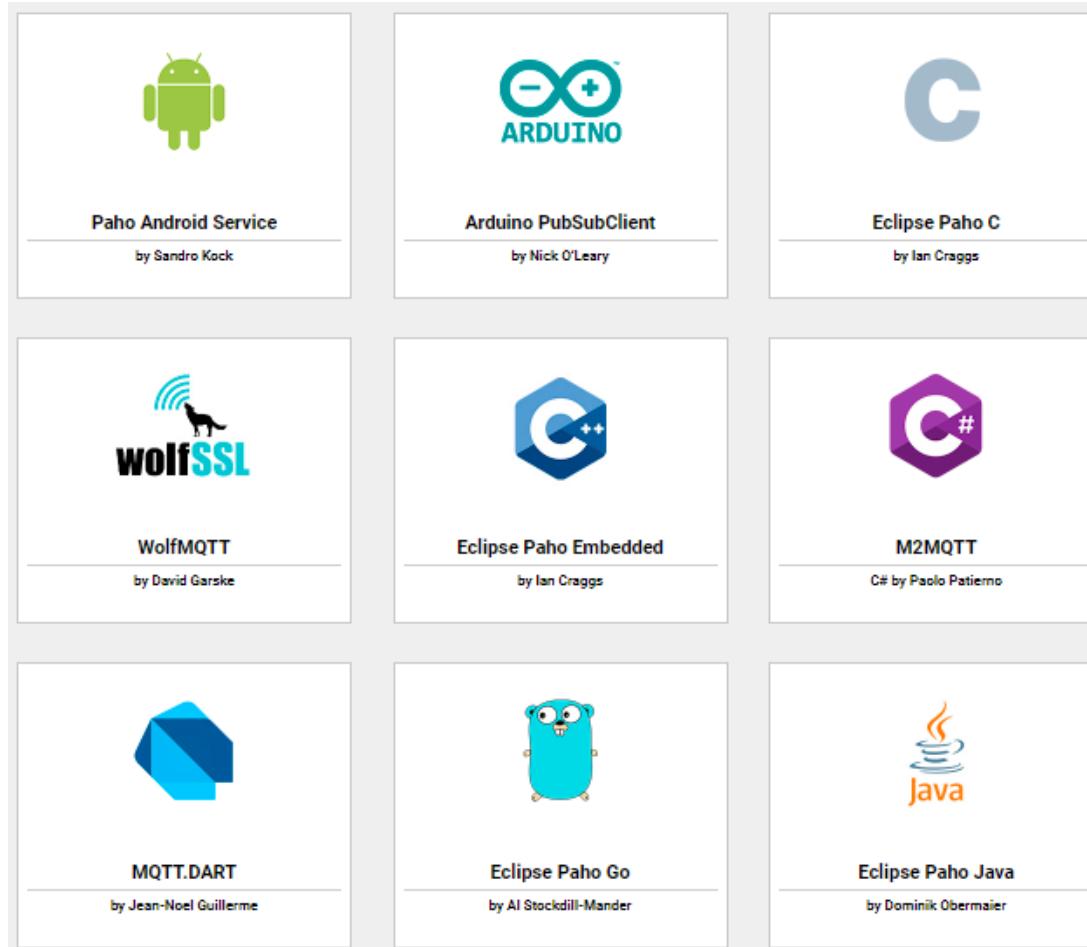
- Topic:** `ktmt_collect`
- Message Content:**

```
{ "id":11, "packet_no":123, "temperature":25,  
  "humidity":60, "tds":1100, "pH":7.0 }
```
- Message Details:**

```
qos : 0, retain : false, cmd : publish, dup : false, top  
ic : ktmt_collect, messageId : , length : 97
```

MQTT client library

- MQTT client libraries for different platforms :
 - <https://www.hivemq.com/mqtt-client-library-encyclopedia/>



Example 1

- Publish/subscribe data using Paho mqtt client library for Java
- Paho mqtt client client
 - <https://www.eclipse.org/paho/index.php?page=clients/java/index.php>
- Using the Paho Java Client:
 - Using Maven dependency
 - Or add library .jar (Build Path >> Add External Archives):
Download library file: org.eclipse.paho.client.mqttv3-1.1.1.jar

Publish message

```
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
public class App {
    public static void main(String[] args) {
        public static void main( String[] args ){
            System.out.println( "MQTT Demo!" );
            new App().doDemo();
        }
        public void doDemo() {
            String subTopic = "/ktmt/iot";
            String pubTopic = "/ktmt/iot";
            String content = "Hello World";
            int qos = 1;
            String broker = "tcp://broker.hivemq.com:1883";
            String clientId = "ID_OF_CLIENT";
            MemoryPersistence persistence = new MemoryPersistence();
            try {
                MqttClient client = new MqttClient(broker, clientId,
persistence);
```

```
// MQTT connection option
MqttConnectOptions connOpts = new MqttConnectOptions();
// retain session
connOpts.setCleanSession(true);
// set callback
client.setCallback(new OnMessageCallback());
// establish a connection
System.out.println("Connecting to broker: " + broker);
client.connect(connOpts);
System.out.println("Connected");
// Subscribe
client.subscribe(subTopic);
// Required parameters for message publishing
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(qos);
System.out.println("Publishing message: " + content);
client.publish(pubTopic, message);
System.out.println("Message published");
client.disconnect();
System.out.println("Disconnected");
client.close();
System.exit(0);
}
catch (MqttException me) {
    me.printStackTrace();
}
```

Subscribe - Callback

```
public class OnMessageCallback implements MqttCallback {  
    public void connectionLost(Throwable cause) {  
        // After the connection is lost, it usually reconnects here  
        System.out.println("disconnect, you can reconnect");  
    }  
    public void messageArrived(String topic, MqttMessage message)  
throws Exception {  
        // The messages obtained after subscribe will be executed here  
        System.out.println("Received message topic:" + topic);  
        System.out.println("Received message Qos:" + message.getQos());  
        System.out.println("Received message content:" + new  
            String(message.getPayload()));  
    }  
    public void deliveryComplete(IMqttDeliveryToken token) {  
        System.out.println("deliveryComplete-----"  
            + token.isComplete());  
    }  
}
```

Example 2:

- Receive sensor data collected from devices sent up
- Refer to the source code:
 - https://github.com/hungpn37/iot_mqttdemo

Exercise 2

- Write a mqtt client program that performs:
 - Send (publish) data to mqtt broker
 - Receive (subscribe) data from mqtt broker
 - Pack data using JSON. For example: "

```
{"id":11, "packet_no":126, "temperature":30,  
"humidity":60, "tds":1100, "pH":5.0}
```
- Use MQTTBox to illustrate sending and receiving
- Example using mqtt broker:
<tcp://broker.hivemq.com:1883>

Exercise 2

[MQTT](#)[Platform](#)[Pricing](#)[Resources](#)[Use Cases](#)[Contact](#)[Login](#)[Get HiveMQ](#)

Free Public MQTT Broker

Looking for a full-featured MQTT platform?
Try HiveMQ.

Try HiveMQ as a managed service or self-hosted.

[Start free](#)

Free Public MQTT Broker by HiveMQ

Our [Public HiveMQ MQTT broker](#) is open for anyone to use. Feel free to write an MQTT client that connects with this broker. We also keep a list of [MQTT client libraries](#) that can be used to connect to HiveMQ.

You can access the MQTT broker securely at:

Broker: broker.hivemq.com

TCP Port: 1883

Websocket Port: 8000

TLS TCP Port: 8883

TLS Websocket Port: 8884

[View public broker](#)

Free MQTT Browser Client by HiveMQ

The HiveMQ MQTT Browser Client is an MQTT WebSocket client interface. Use any modern browser on any device as a full-fledged MQTT client and take full advantage of the MQTT protocol.

- No need to download and install
- Works on all operating systems
- Full support for MQTT 3.1
- Works with any HiveMQ MQTT broker with websockets

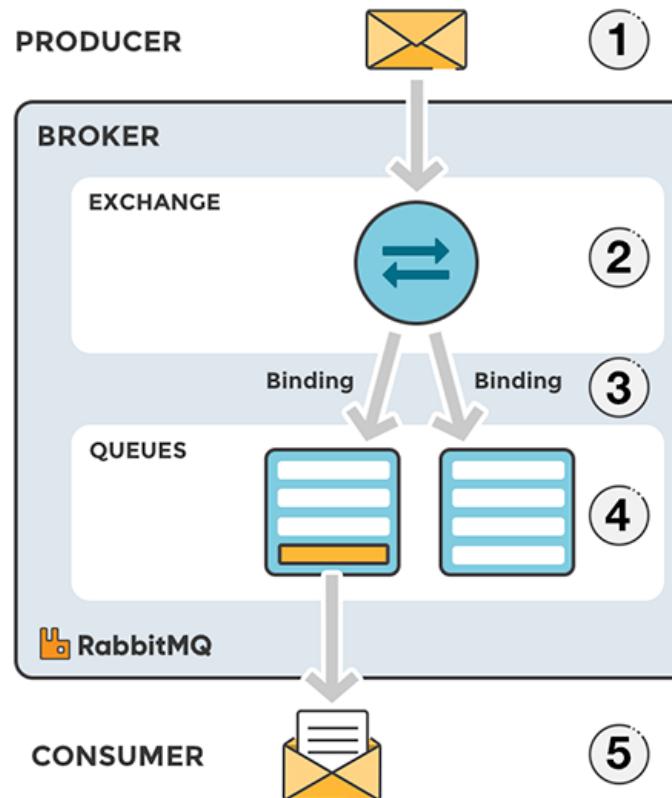
[Try MQTT browser client](#)

2.3.3. AMQP protocol

- AMQP (Advanced Message Queue Protocol): The transmission protocol uses message queues
- RabbitMQ: A broker uses the AMQP protocol
- RabbitMQ broker acts as an intermediary to store and coordinate messages between the sender (producer) and the receiver (consumer)

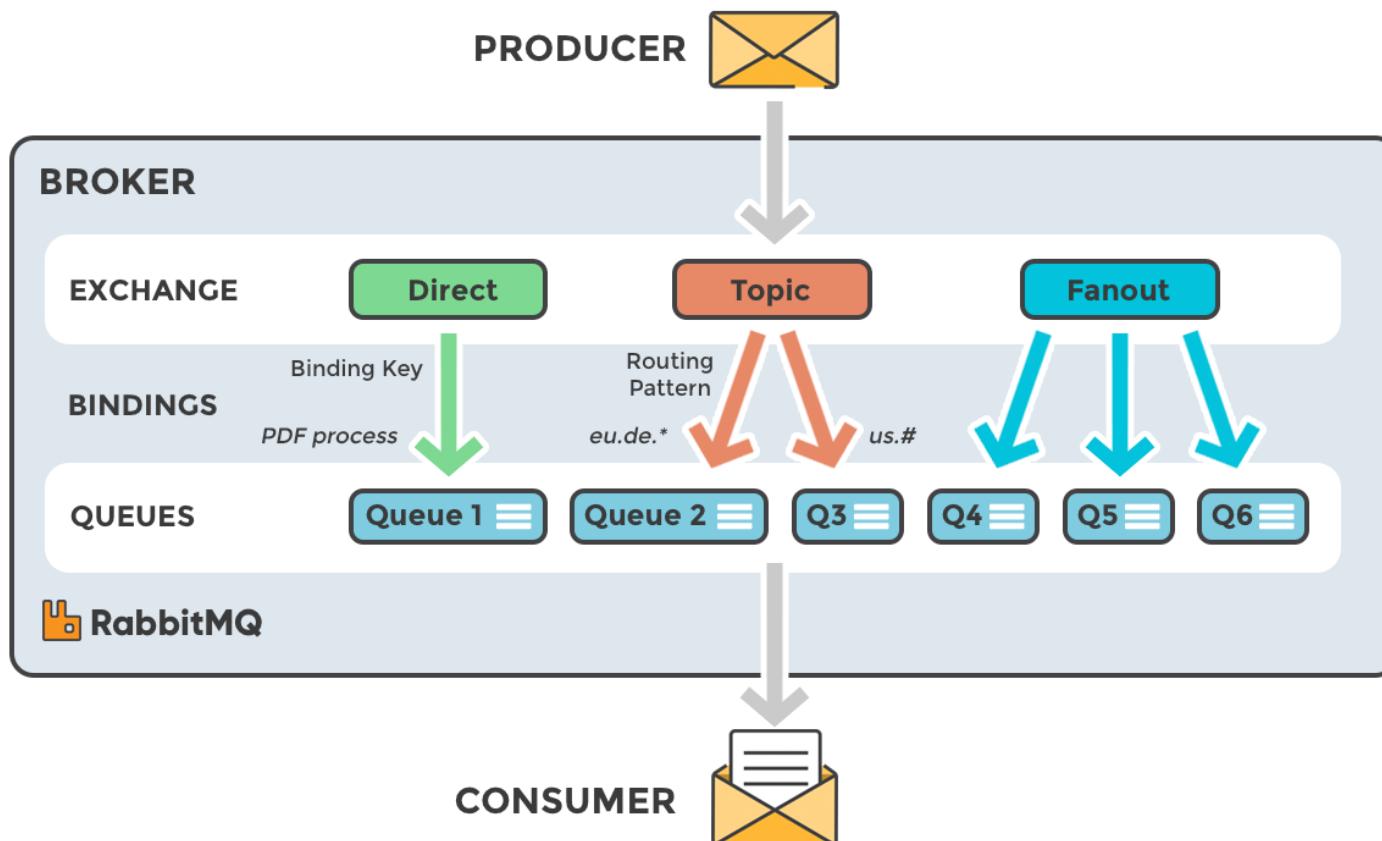
AMQP protocol

- RabbitMQ:
 - Flow of sending and receiving messages via RabbitMQ



AMQP protocol

- RabbitMQ:
 - Four Exchange types: direct, topic, fanout, headers

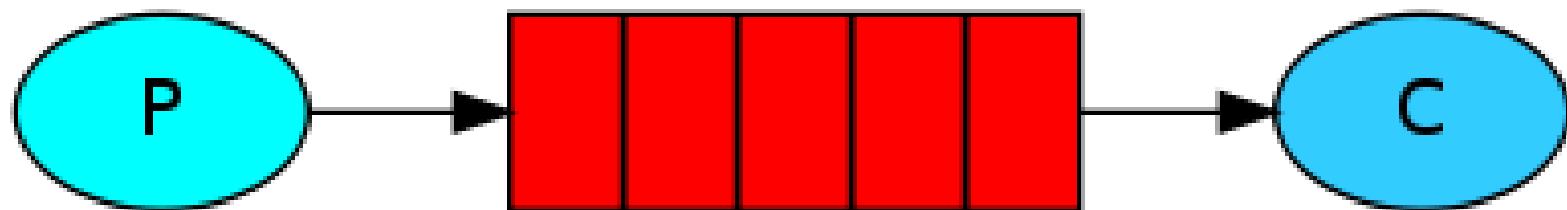


AMQP protocol

- Install RabbitMQ broker (server)
 - <https://www.rabbitmq.com/download.html>
 - <https://www.rabbitmq.com/install-windows.html>
- Administer the RabbitMQ broker using the tool:
 - <https://www.rabbitmq.com/management.html>
- RabbitMQ tutorials:
 - <https://www.rabbitmq.com/getstarted.html>

Tutorial 1

- Write a program to send and receive data via RabbitMQ
 - Install and use a RabbitMQ broker
 - Producer sends data (message)
 - Consumer receives data
 - Use default exchange (type: direct)



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

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

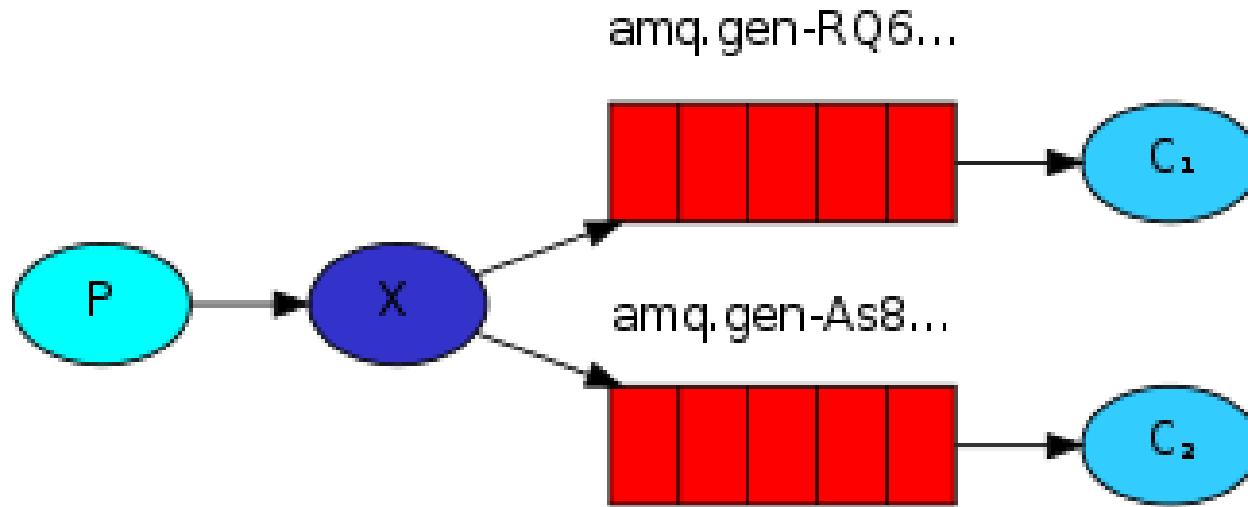
```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DeliverCallback;
public class Recv {
    private final static String QUEUE_NAME = "ktmt";
    public static void main(String[] argv) throws Exception {
        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

Tutorial 3

- Improved Tutorial 1: a Producer sends messages to all consumers
 - Use exchange type = “fanout” (broadcast)

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



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

Tutorial 4

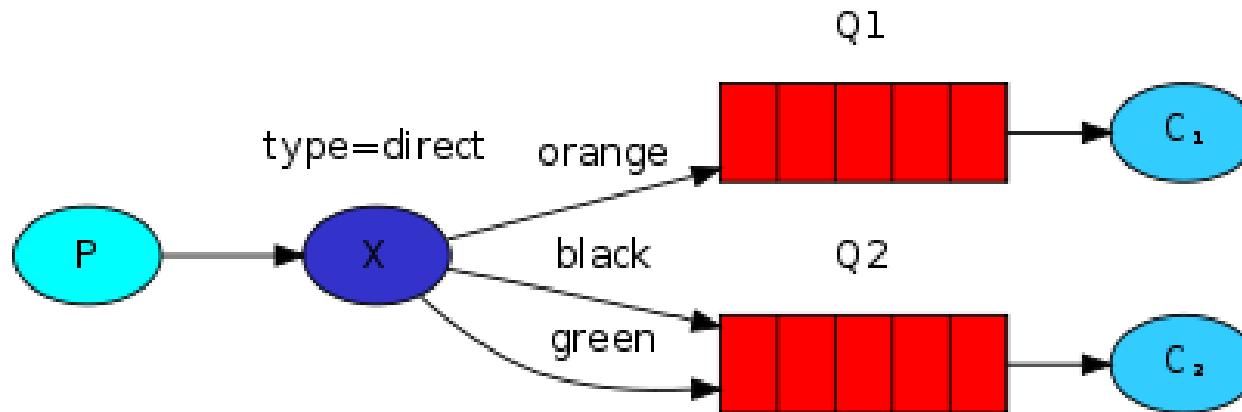
- Routing messages
- Select the message you want to receive (Receiving messages selectively)

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

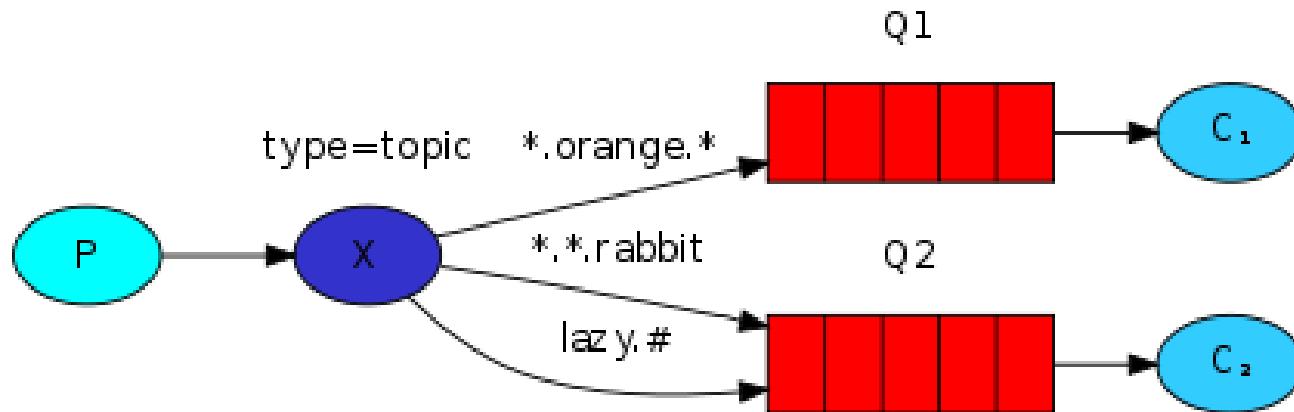
- Use exchange type = “topic”
- routing_key must be a list of words, delimited by dots.
 - *(star) can substitute for exactly one word.
 - #(hash) can substitute for zero or more words.

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>

Exercise 3

- Write a program to send/receive data via RabbitMQ broker:
 - Send (publish) data to the RabbitMQ broker
 - Receive (subscribe) data from RabbitMQ broker
 - Packaging data using JSON. For example:

```
{"id":11, "packet_no":126, "temperature":30,  
"humidity":60, "tds":1100, "pH":5.0}
```

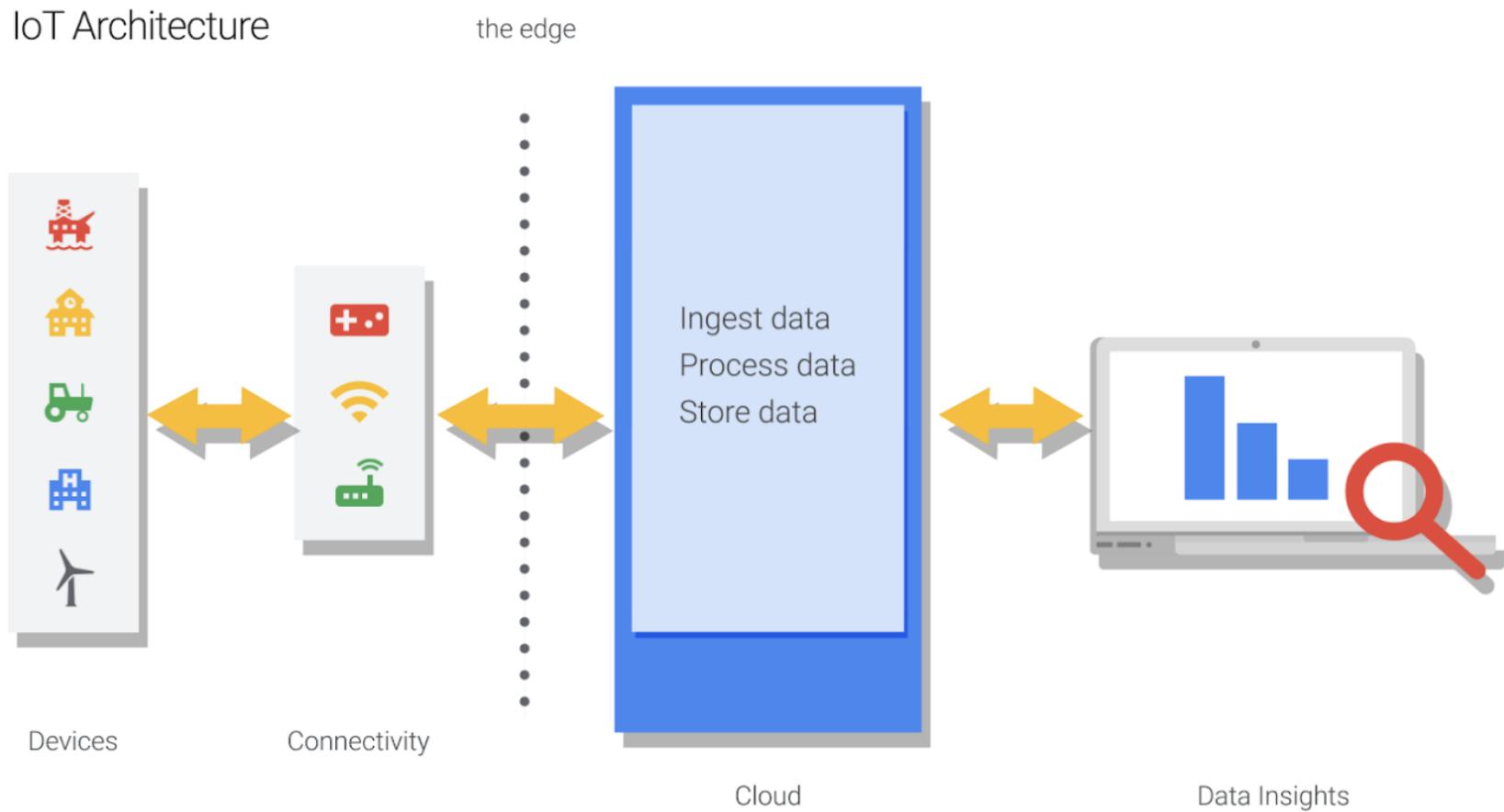
2.4. IoT Cloud Platform

- The architecture of IoT must have the ability to scale the connectivity of devices, gather data, process data, and store data.
- It should be capable of performing these tasks quickly while still conducting real-time data analysis.
- Sending large amounts of data to the cloud can slow down the processing process and may require additional bandwidth for data transmission

IoT Cloud Platform

- The popularity of distributed computing solutions, such as **fog or edge computing**, is on the rise.
- Edge computing involves performing computational tasks on nodes within the network, which are IoT devices located at the 'edge' of the network.
- This solution leads to the need for IoT devices to have the capability to locally process, clean, and analyze data. Only pre-processed data is then sent to the cloud.

IoT Cloud Platform



IoT Cloud Platforms

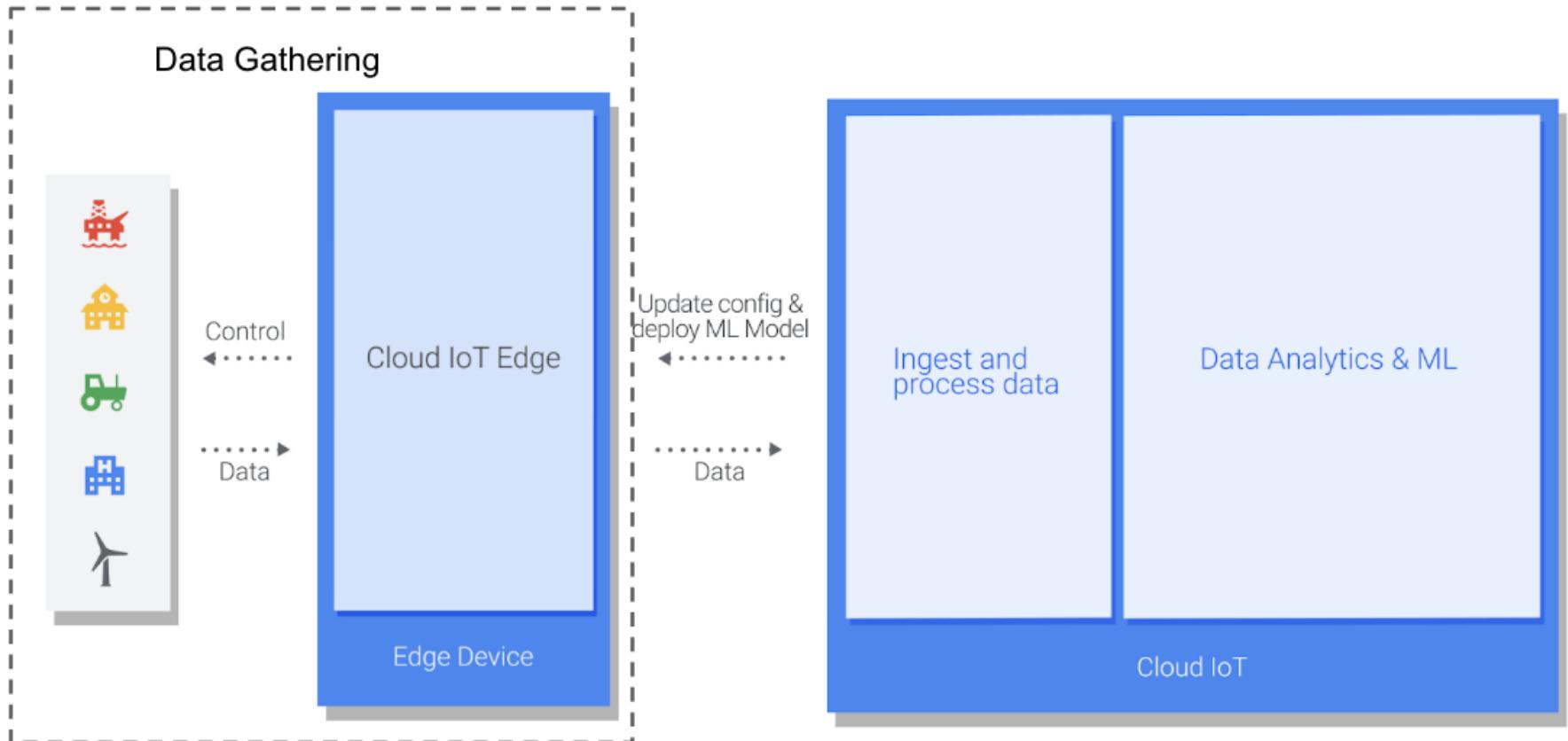
- Some IoT Cloud service platforms include:
 - IBM Watson IoT
 - AWS IoT
 - Google Cloud IoT
 - Microsoft Azure IoT



<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

- The Google Cloud IoT architecture is divided into four components:
 - data gathering
 - data ingest
 - data processing
 - data analysis

Google Cloud IoT

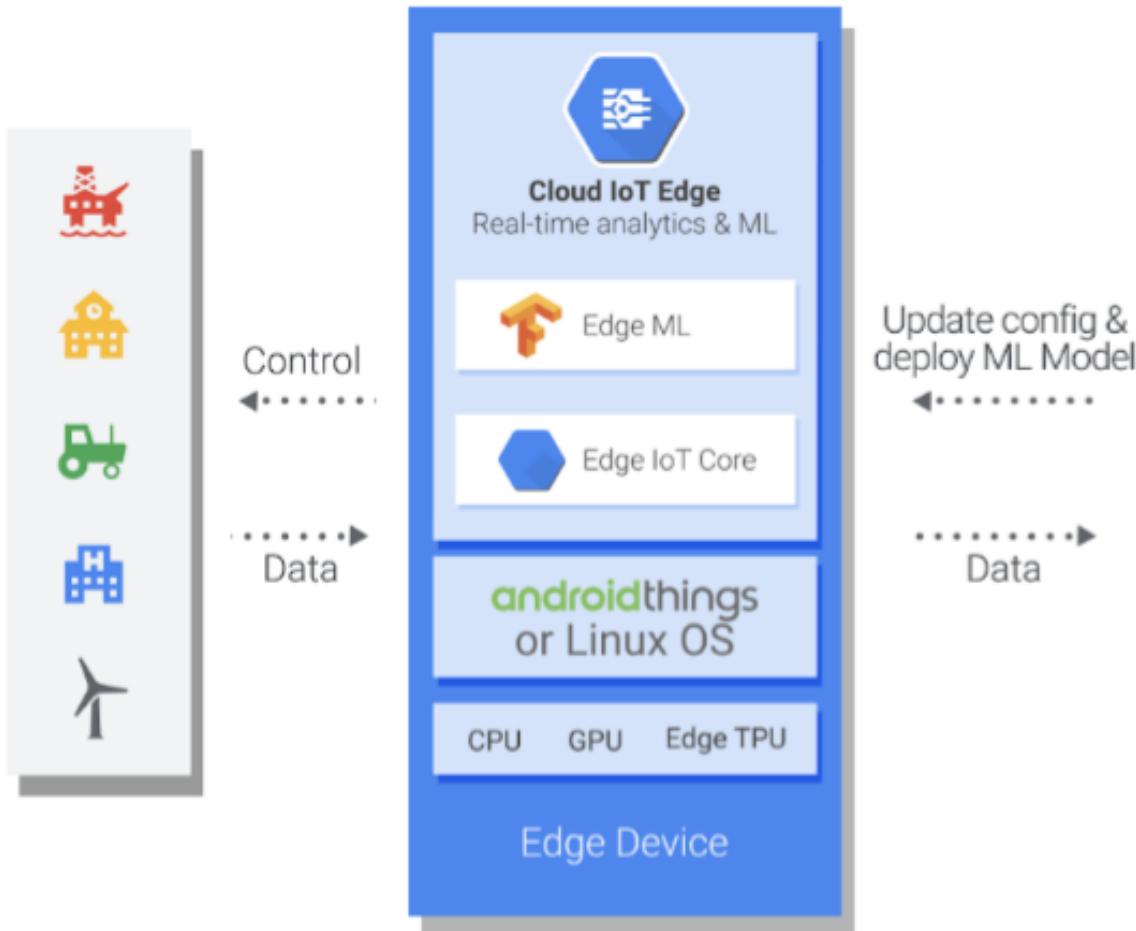
- Data Gathering (On-Device Data Collection):
 - Occurs on devices and sensors.
 - Sensors collect data from the environment and send it to the cloud, either directly or indirectly through an intermediary device.
 - Devices prepare the data for transmission to the cloud.

Depending on the type of network connection, the preparation process may include data cleaning, preprocessing, analysis, or even utilizing machine learning



Google Cloud IoT

- Cloud IoT Edge:

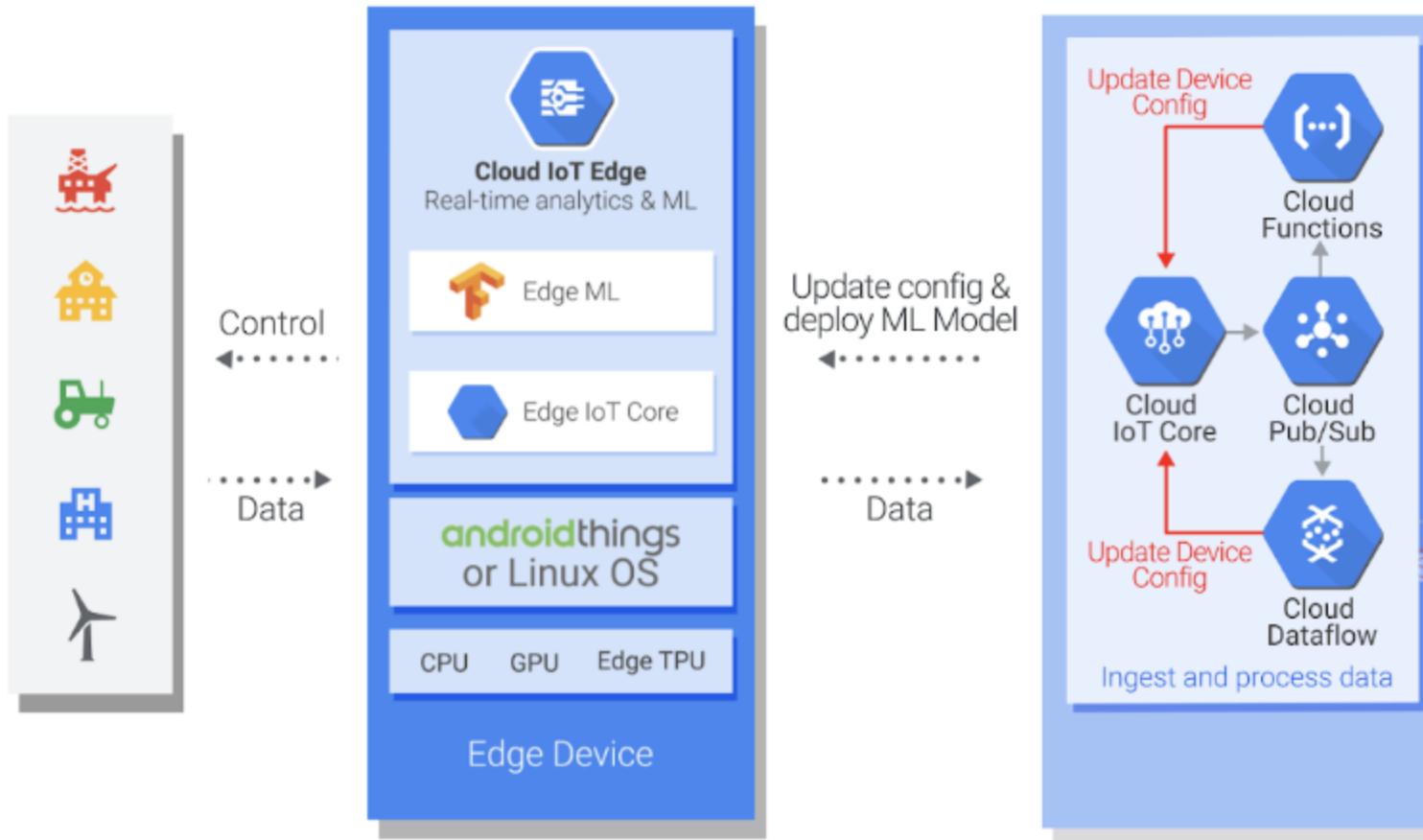


Google Cloud IoT

- Cloud IoT Edge:
 - In the Google Cloud IoT architecture, the data gathering phase can be performed on Cloud IoT Edge.
 - This component is a group of devices capable of conducting real-time analytics and machine learning. It enables the extension of data processing and ML operations to edge devices.
 - Cloud IoT Edge can utilize Android Things OS or Linux-based OS.
 - It consists of two components: Edge IoT Core and Edge ML

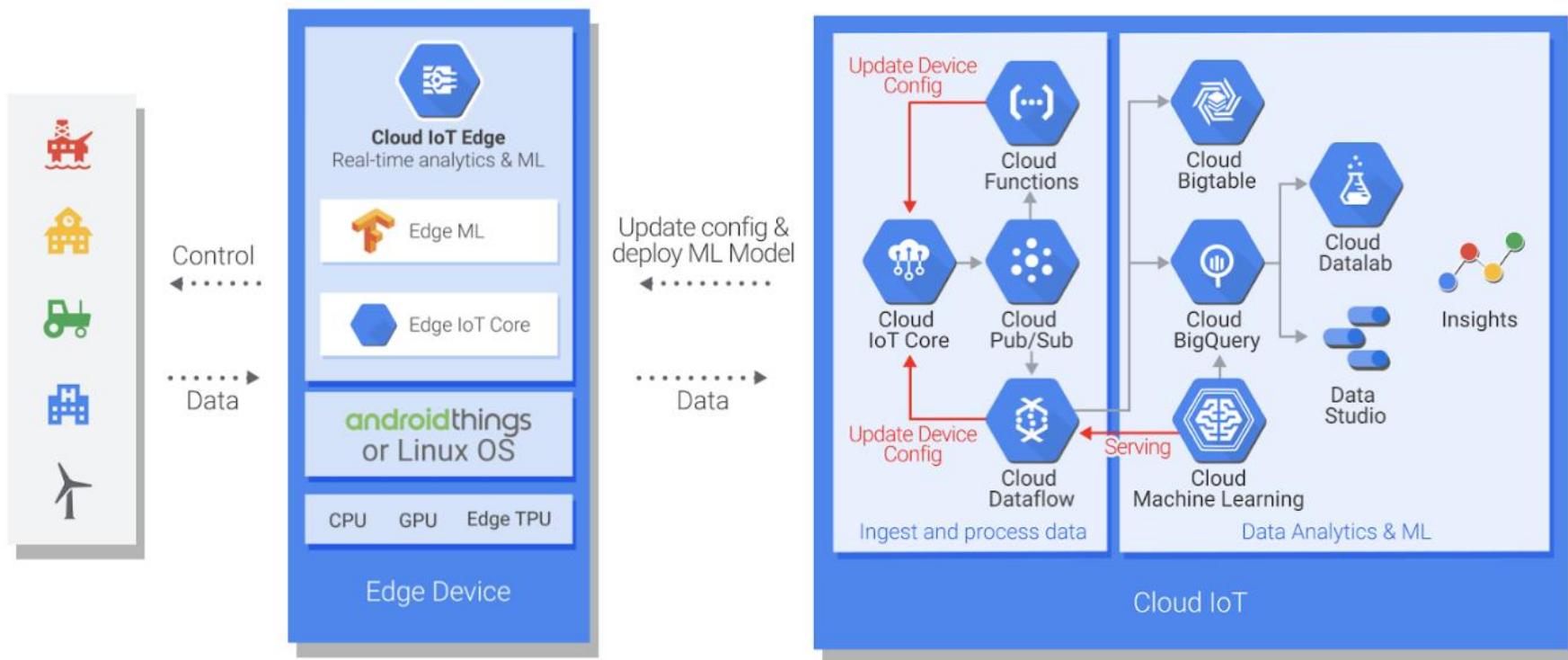
Google Cloud IoT

- Ingest and Process data:



Google Cloud IoT

- Data Analytics and ML:
 - It can be performed on the Edge or in the Cloud.
 - Google's Cloud IoT Core Data Analytics and ML are fully integrated with IoT data.



Content

- Chapter 1. Overview of IoT
- Chapter 2. IoT Technologies
- Chapter 3. IoT Application and Programming
- Chapter 4. IoT Safety and Security
- Chapter 5. Designing and Building IoT Systems

Chapter 3. IoT Application and Programming

- 3.1. Programming for IoT devices
- 3.2. Building an IoT Server
- 3.3. Exploiting IoT Cloud Platform services

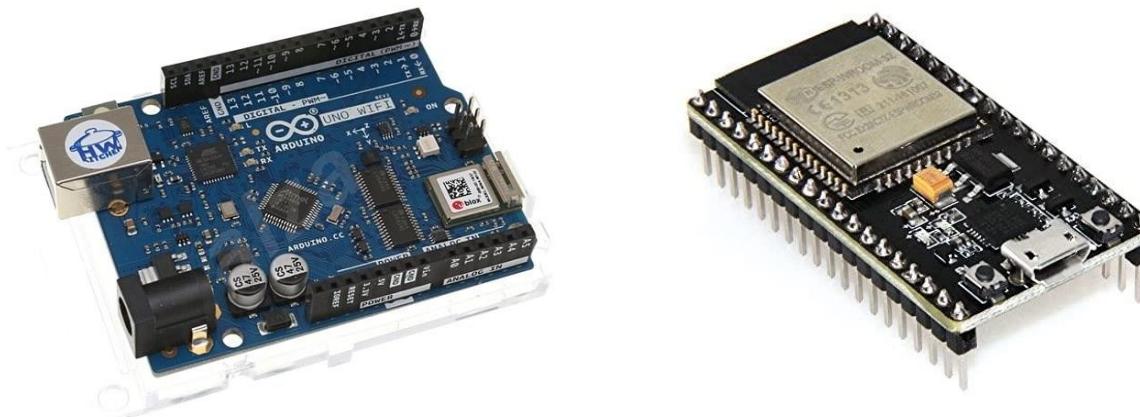
3.1. Programming for IoT devices

- Devices use microcontrollers
 - 8 bit-architecture, low performance, and do not use an operating system
 - Firmware programming, basic input-output operations using GPIO (General Purpose Input/Output), interfacing with peripherals (sensors, actuators) through common communication standards like UART, SPI, I2C
 - Examples: ATmega, PIC, AVR, Arduino



3.1. Programming for IoT devices

- Devices use 32-bit processors with moderate performance
 - Can use simple operating systems (FreeRTOS, TinyOS, uCLinux, ...)
 - Perform basic input-output operations using GPIO, interface with peripherals using common communication standards like UART, SPI, I2C.
 - capability to connect with Wi-Fi, Bluetooth, GSM module
 - Examples: ESP8266, ESP32, Arduino Uno WiFi



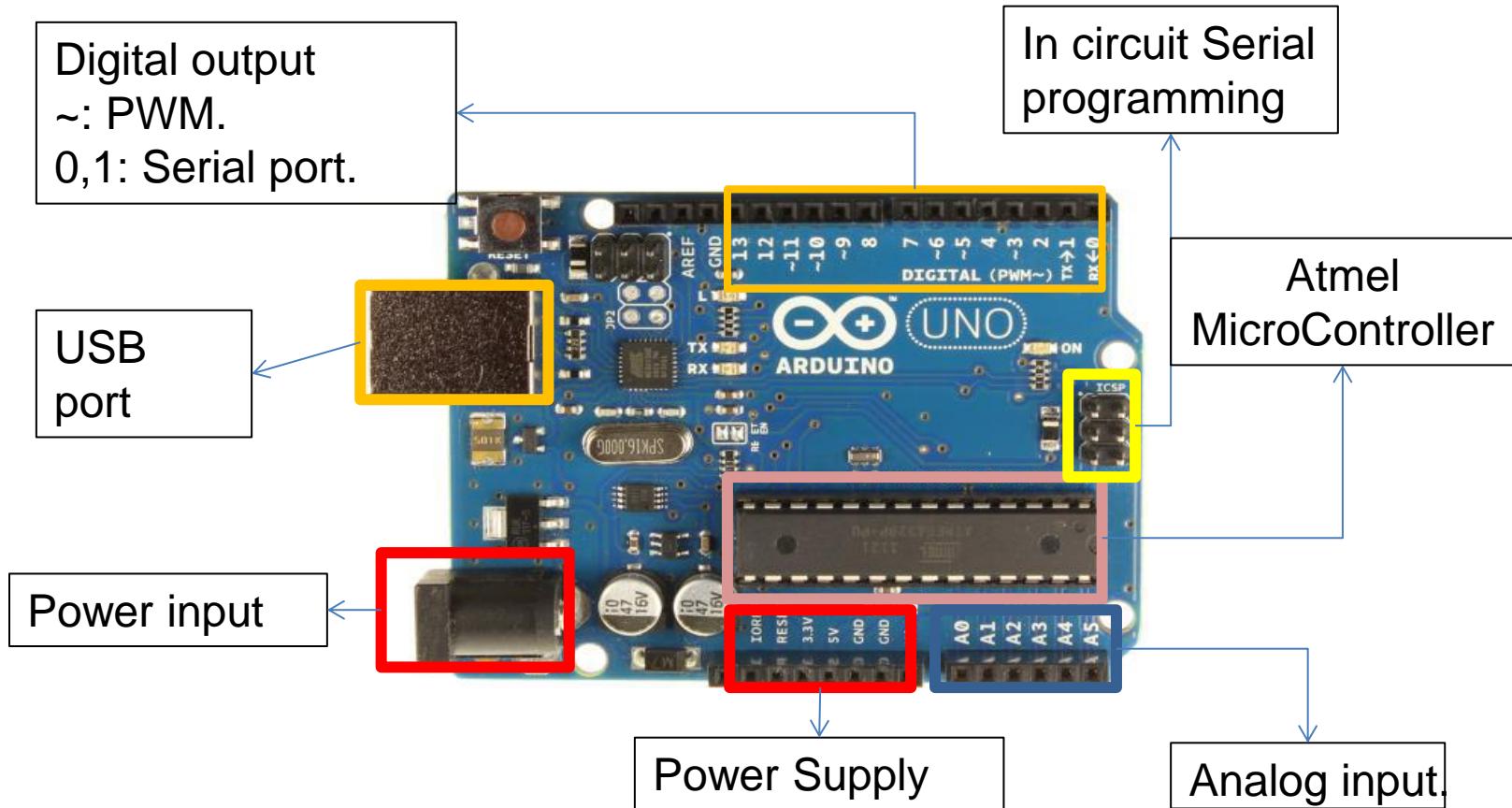
3.1. Programming for IoT devices

- Devices use 32/64-bit processors with high performance.
 - Run operating systems such as Embedded Linux, Raspbian, Windows Embedded, Android, Ubuntu
 - Have connectivity options like Wi-Fi, Ethernet.
 - Peripheral communication: through drivers on the operating system.
 - Can use libraries on the operating system like Java, Python, .Net Framework, OpenCV
 - An example is the Raspberry Pi



3.1.1. Arduino programming

- Arduino Uno: Microchip ATmega328, no operating system used



3.1.1. Arduino programming

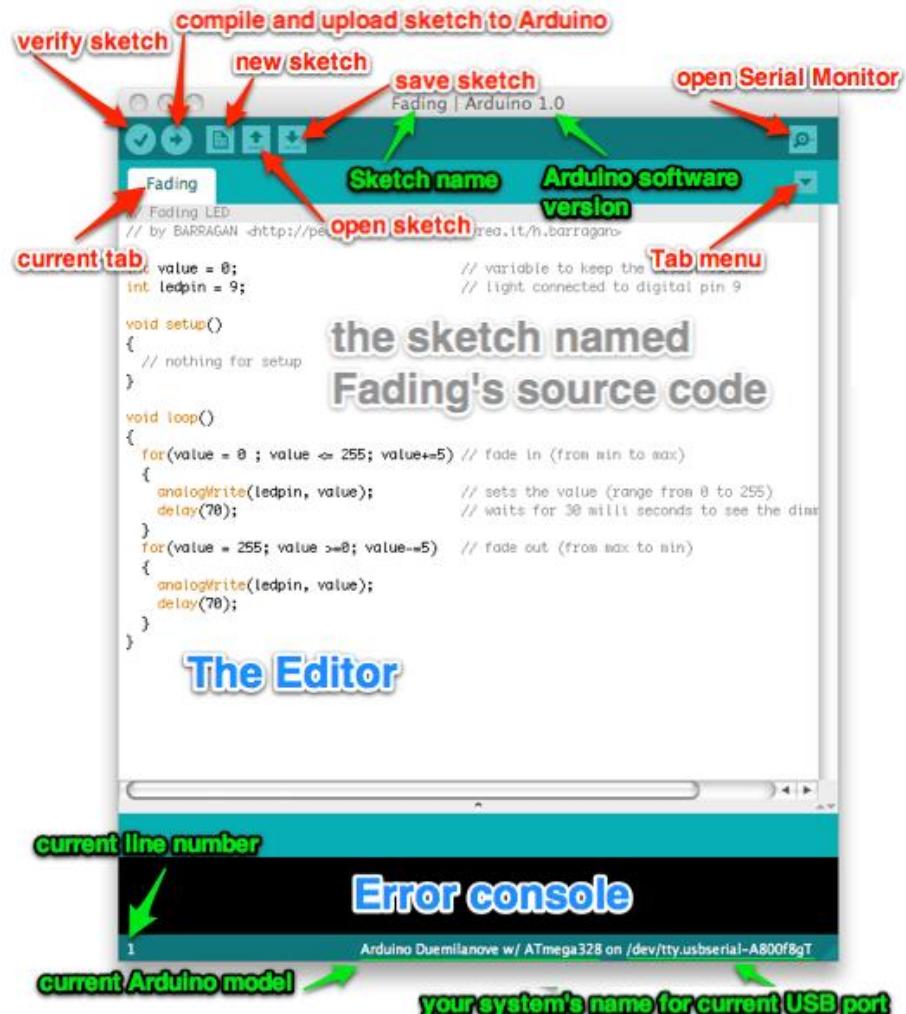
- Install Arduino IDE
 - <https://www.arduino.cc/en/guide/windows>
- Code structure

```
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.
}
```

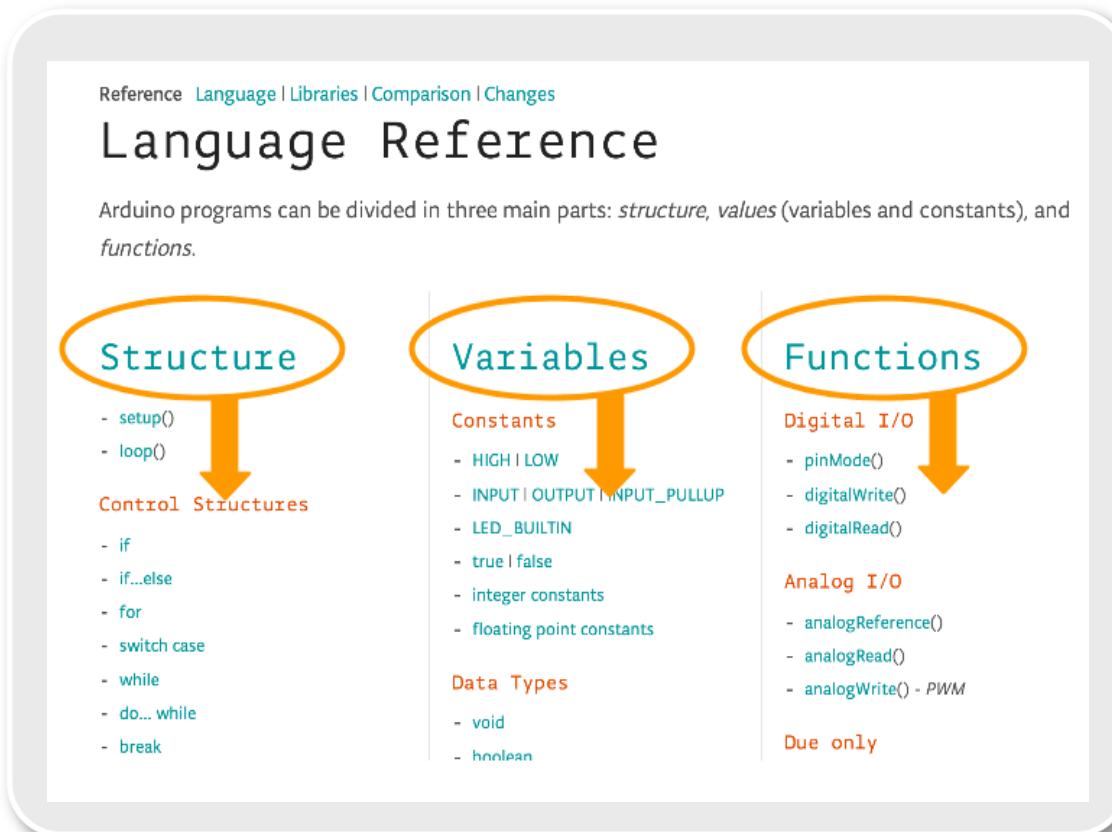
Arduino programming

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



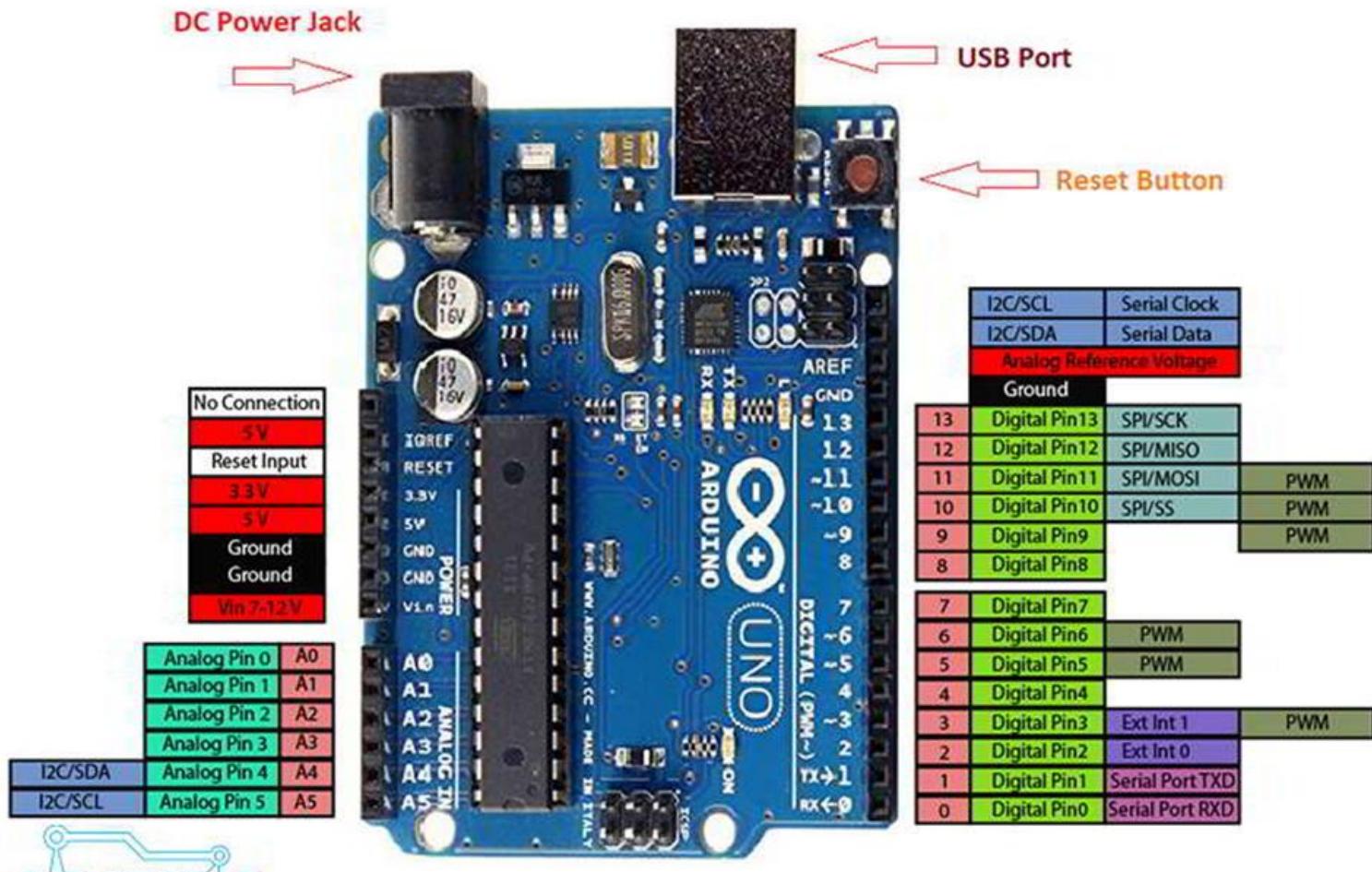
Arduino programming

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



Peripheral communication

- GPIO, UART, I2C, SPI, ADC, PWM



Example 1: GPIO Communication - Output

- LED on/off trên pin 13

```
void setup() {  
    // initialize digital pin 13 as an output.  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13, HIGH);      // turn the LED on  
    delay(1000);                // wait for a second  
    digitalWrite(13, LOW);       // turn the LED off  
    delay(1000);                // wait for a second  
}
```

Example 1: GPIO Communication - Output

- Read state of the button on pin 2, and turn on/off on 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);
}
```

Example 2: UART communication

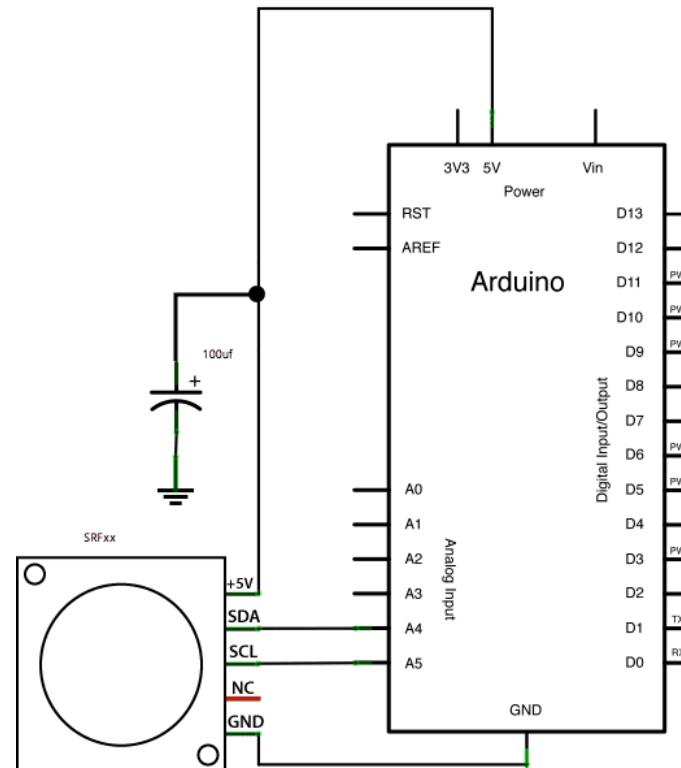
- Use Serial library of Arduino

- <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);
    }
}
```

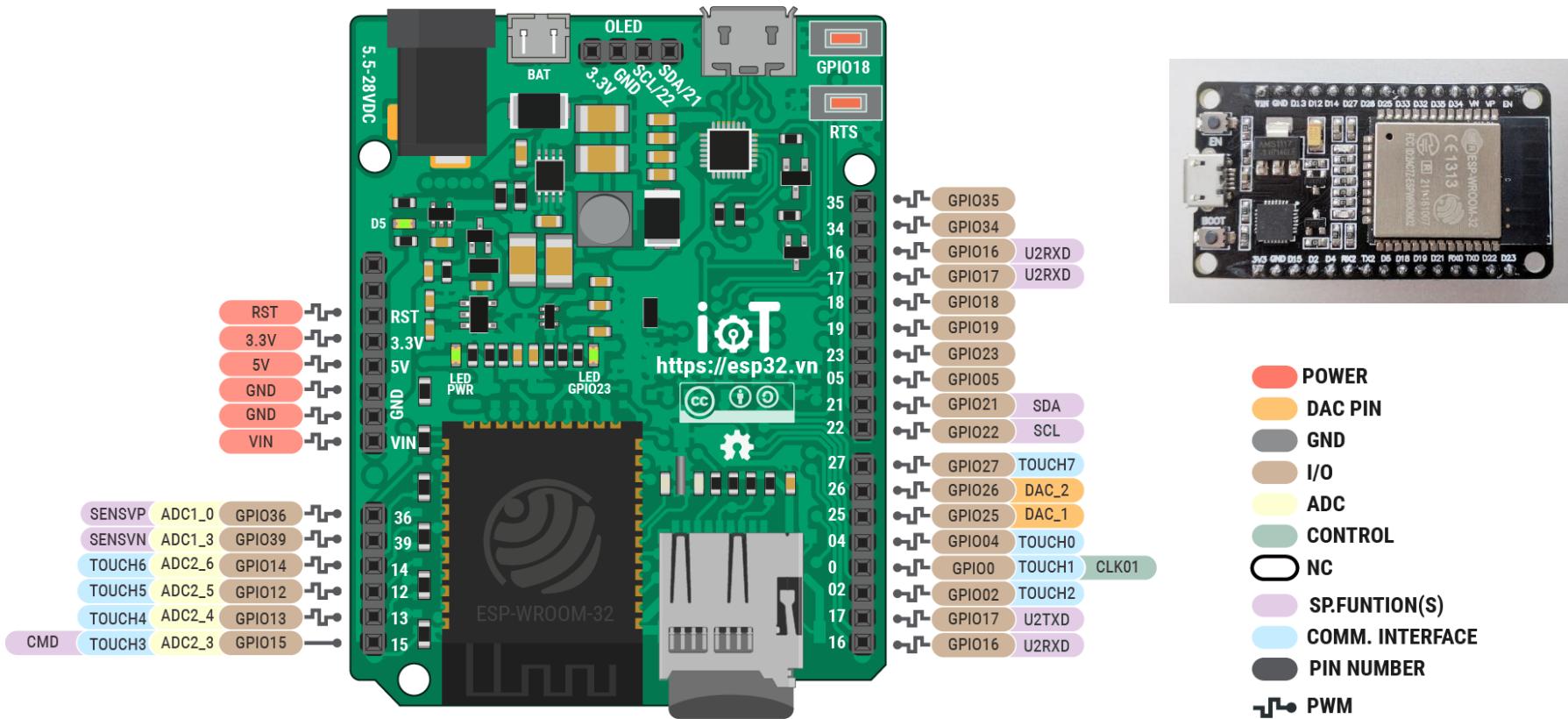
Example 3: I2C communication

- Use Wire library
 - <https://www.arduino.cc/en/Reference/Wire>
- SRFxx Sonic Range Finder interface
 - <https://www.arduino.cc/en/Tutorial/LibraryExamples/SFRRangerReader>



3.1.2. Programming ESP32/ESP8266 devices

- Programming with ESP32/ESP8266
 - <https://esp32.vn/index.html>
 - <https://randomnerdtutorials.com/projects-esp32/>

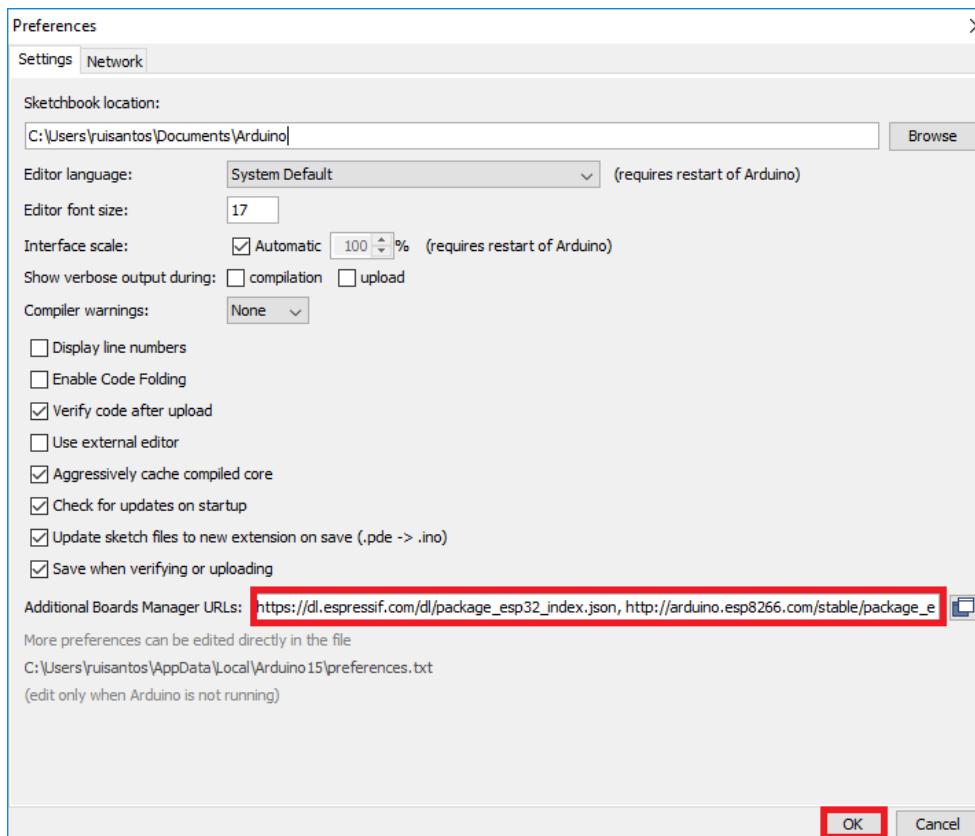


Programming ESP32/ESP8266 devices

- A. Setting up Arduino IDE for ESP32
- B. Connecting ESP32 to WiFi
- C. Communicating via HTTP with ESP32
- D. Communicating via MQTT with ESP32
- E. Programming multi-tasks with FreeRTOS

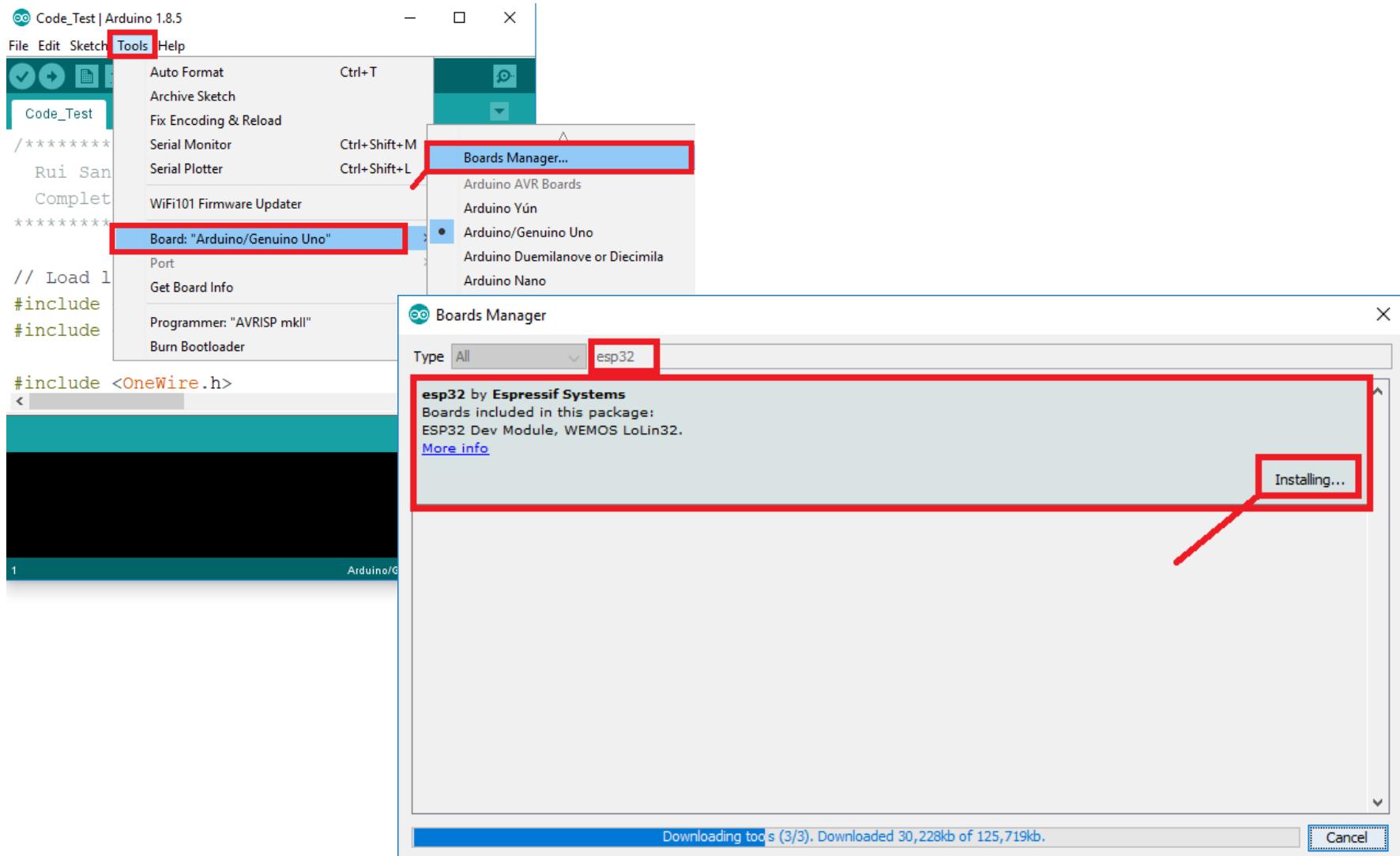
A. Setting up Arduino IDE for ESP32

- Installing the ESP32 Board Add-on on Arduino IDE:
 - 1) Open **File > Preferences**
 - 2) Enter https://dl.espressif.com/dl/package_esp32_index.json in “Additional Board Manager URLs” field



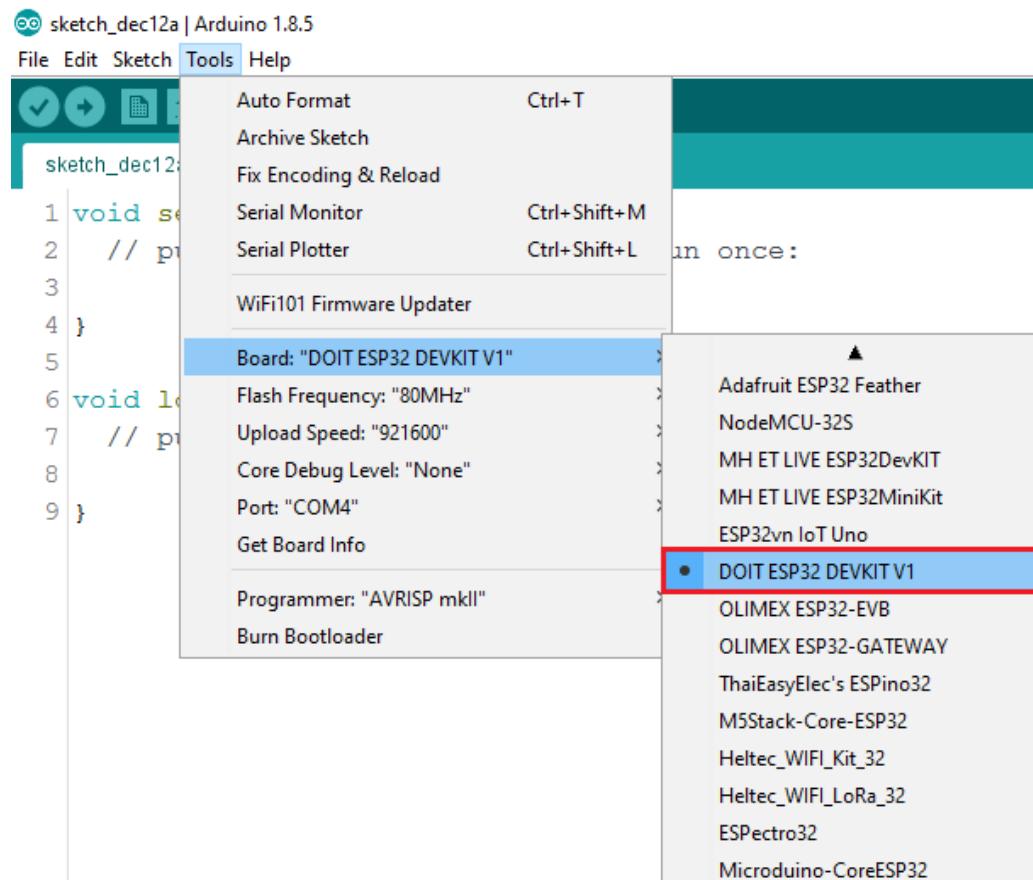
Setting up Arduino IDE for ESP32

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

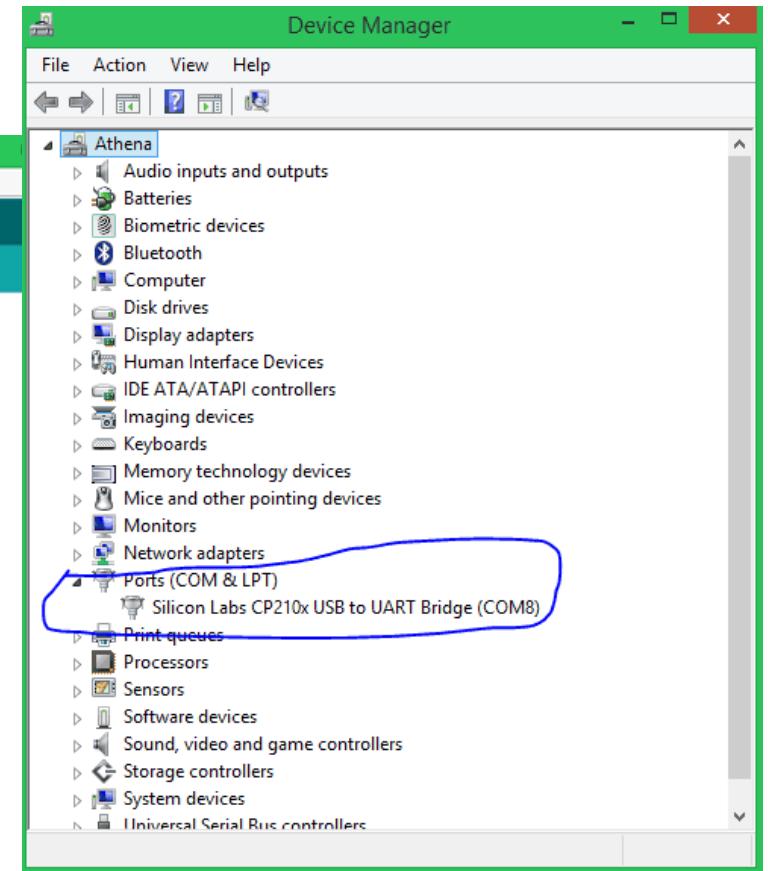
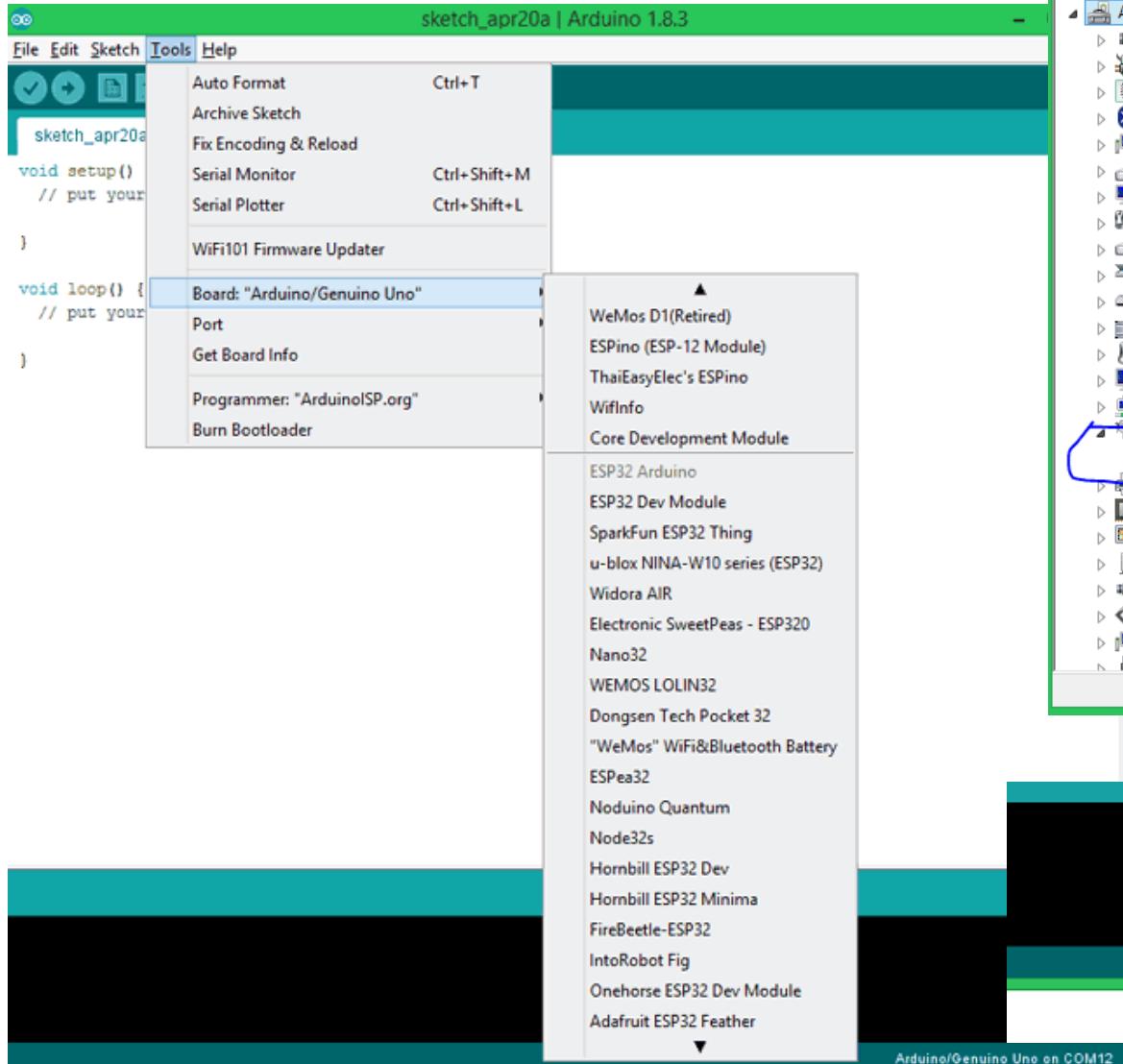


Setting up Arduino IDE for ESP32

- Connect the ESP32 board to the computer
 - Select Board in **Tools > Board** menu (example: **DOIT ESP32 DEVKIT V1**)

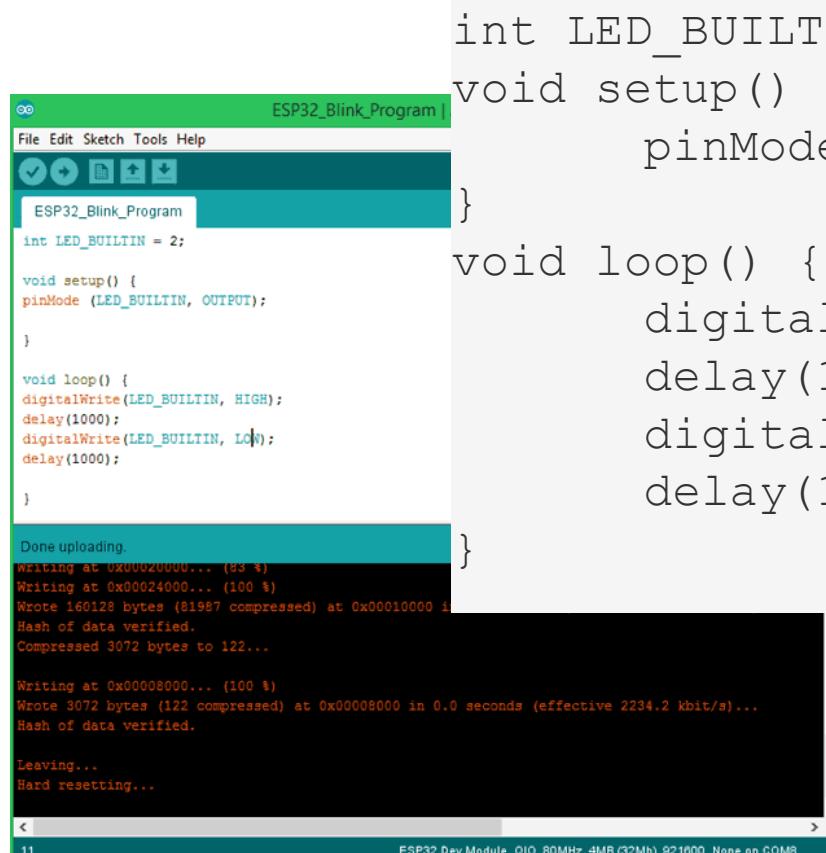


Connect to the device



Upload the program to the device

- 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 following details:

- Title Bar:** ESP32_Blink_Program
- Menu Bar:** File Edit Sketch Tools Help
- Sketch:** The code for the ESP32_Blink_Program is displayed. It includes setup() and loop() functions. The setup() function initializes pin 2 as an output. The loop() function alternates between setting the LED to HIGH and LOW every 1000 milliseconds, with a delay of 1000 ms between each state change.
- Output Window:** Shows the progress of the upload:
 - Writing at 0x00020000... (83 %)
 - Writing at 0x00024000... (100 %)
 - Wrote 160128 bytes (81987 compressed) at 0x00010000 in 0.0 seconds (effective 2234.2 kbit/s)...
 - Hash of data verified.
 - Compressed 3072 bytes to 122...
- Bottom Status Bar:** Shows the board configuration: ESP32 Dev Module, QIO, 80MHz, 4MB (32Mb), 921600, None on COM8.



B. Connecting ESP32 to WiFi

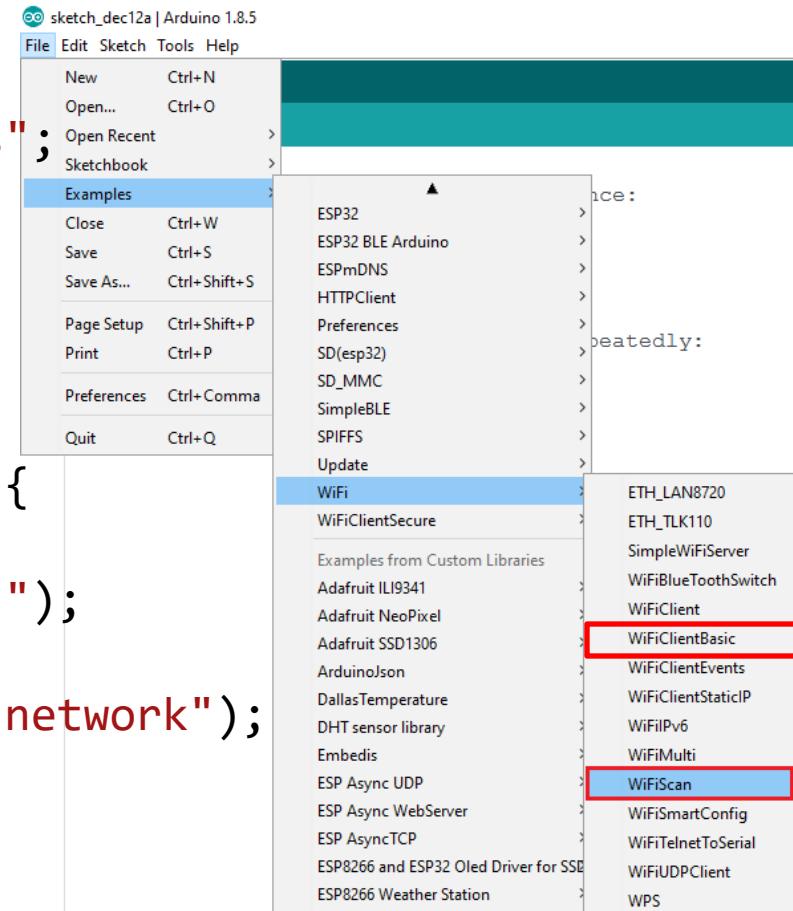
Example 1: File > Examples > WiFi (ESP32) > WiFiClientBasic

```
#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");
}

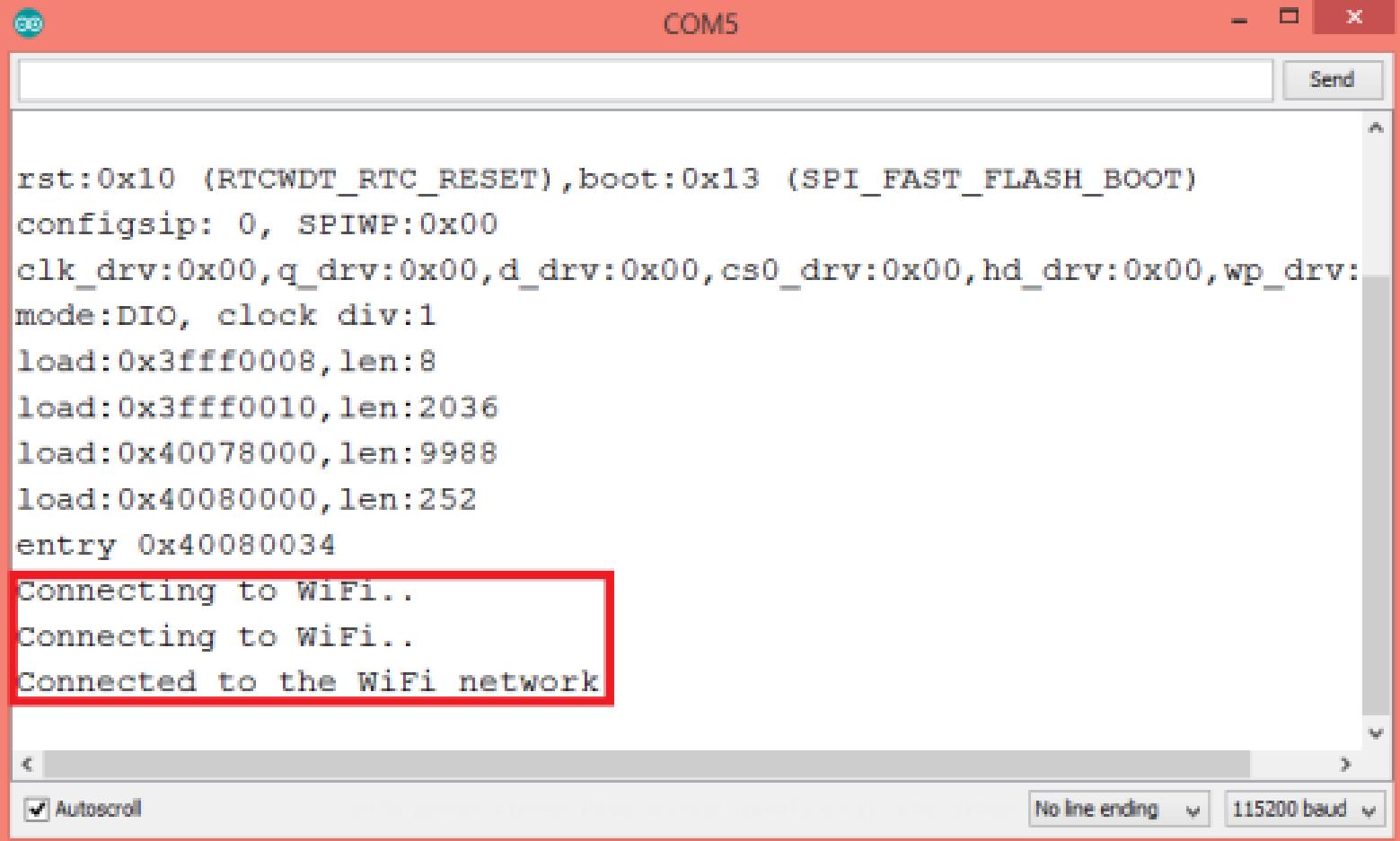
void loop() {
```



Example 2: File > Examples > WiFi (ESP32) > WiFiClient

Example of Wi-Fi connection

- Result: Write the log to the Serial port

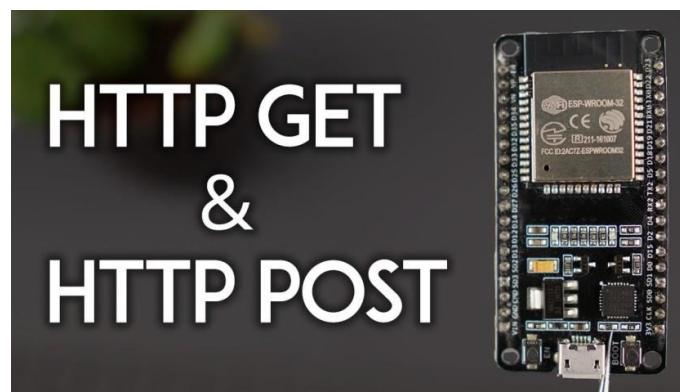


```
rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:
mode:DIO, clock div:1
load:0x3fff0008,len:8
load:0x3fff0010,len:2036
load:0x40078000,len:9988
load:0x40080000,len:252
entry 0x40080034
Connecting to WiFi..
Connecting to WiFi..
Connected to the WiFi network
```

Autoscroll No line ending 115200 baud

C. Communicating HTTP with ESP32

- HTTP request methods: GET vs. POST
- HTTP GET:
 - Used to request data/resources (get values from APIs)
 - Data is sent in the URL of the GET request
 - Example:
GET /update-sensor?temperature=value1
 - Or get JSON object:
GET /get-sensor



Communicating HTTP with ESP32

- HTTP POST:

- Send data to the server to create/update resources
- Data is sent in the body of the POST request
- Example:

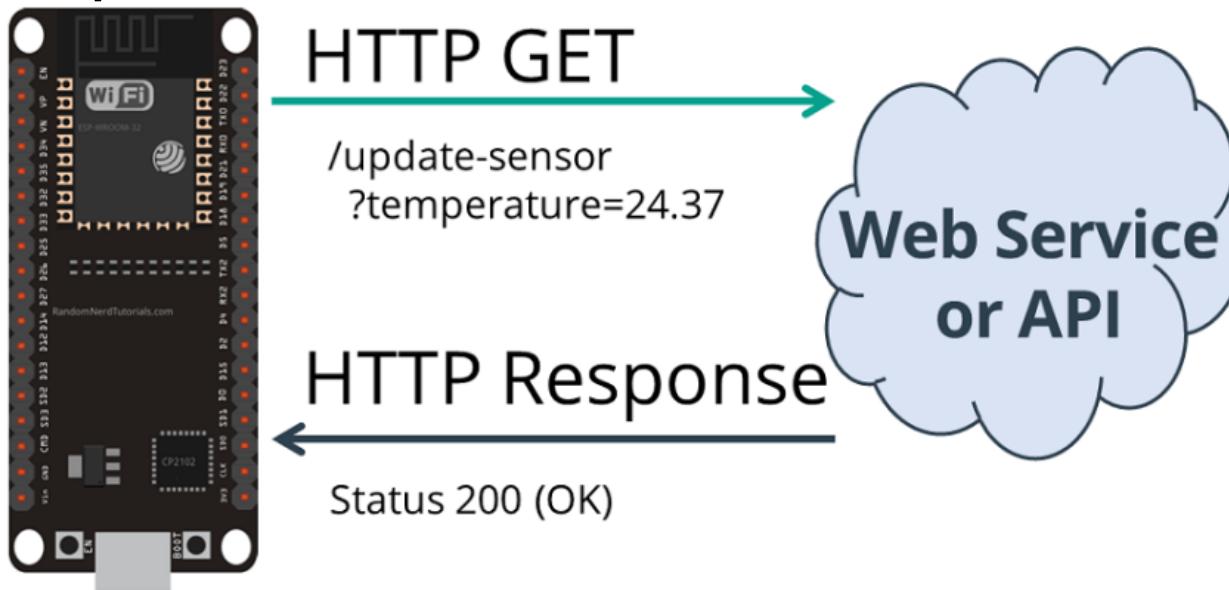
```
POST /update-sensor HTTP/1.1 Host: example.com  
api_key=api&sensor_name=name&temperature=value1&humidity=value2&pressure=value3  
Content-Type: application/x-www-form-urlencoded
```

- Or use 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

- ESP32 sends data to the server using an HTTP GET request



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

ESP32 HTTP GET – Example Code

Part 1. Connect WiFi

```
#include <WiFi.h>
#include <HTTPClient.h>
const char* ssid = "Soict";
const char* password = "Soict1234";
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);
}
```

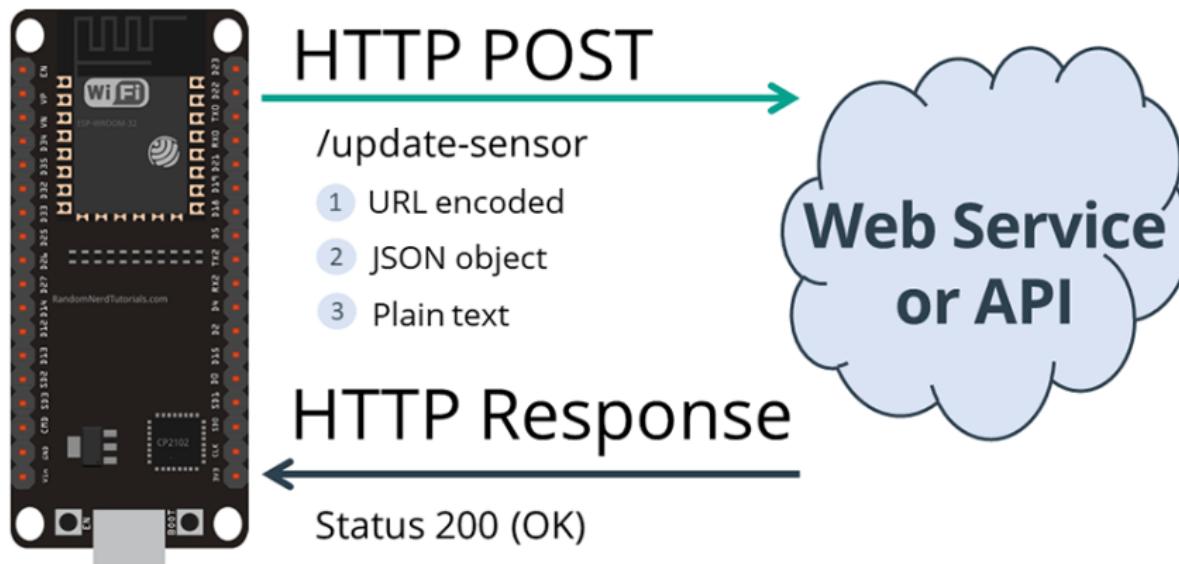
ESP32 HTTP GET – Example code

```
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 POST

- Sending data using:
 - URL encoded (packaged in the URL)
 - JSON object (packaged in the request body)
 - Plain text



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

ESP32 HTTP POST

- POST data with 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

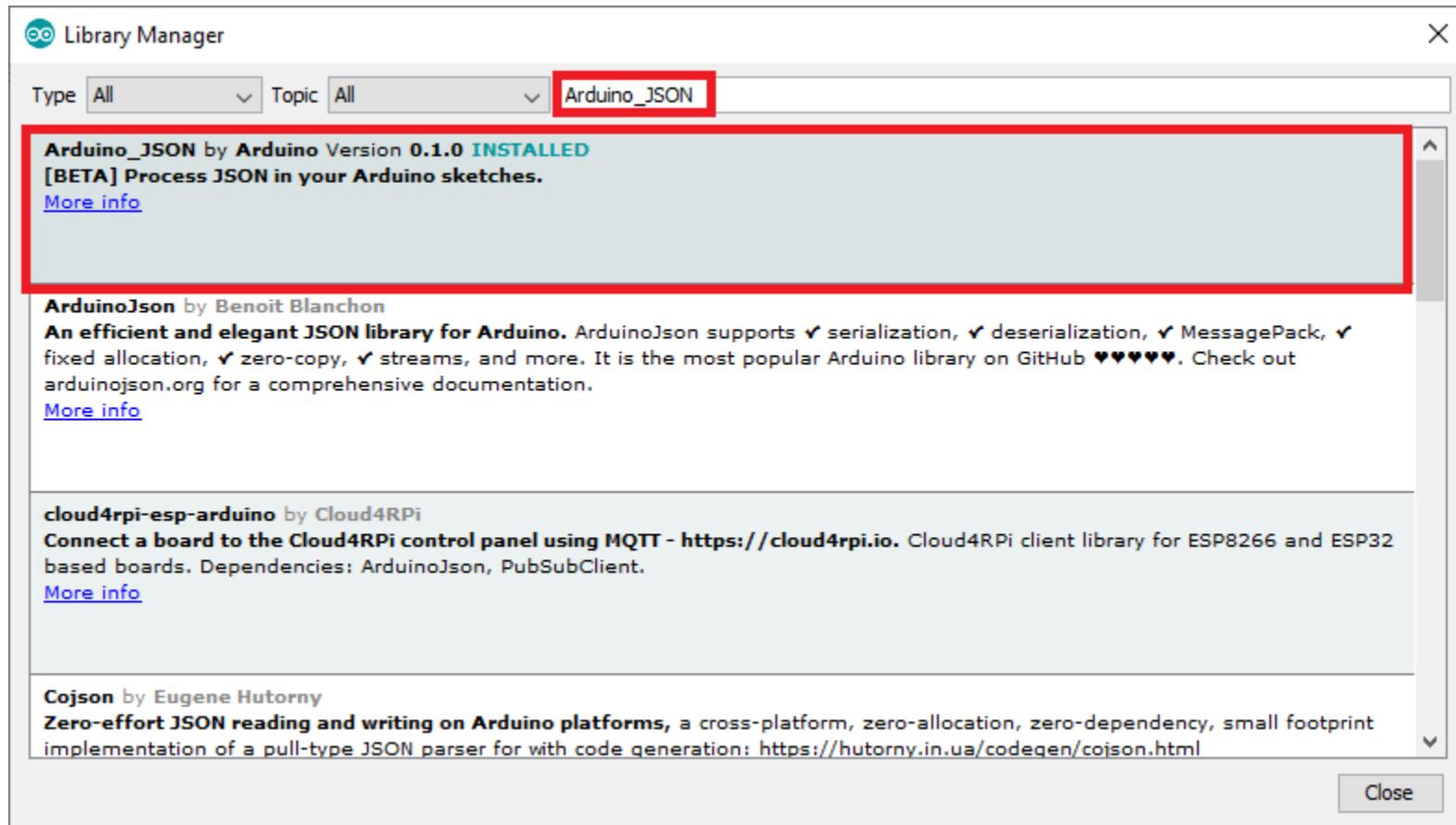
```
http.addHeader("Content-Type", "application/json");
int httpResponseCode =
http.POST("{\"api_key\": \"tPmAT5Ab3j7F9\", \"sensor\": \"BME280\", \"value1\" : \"24.25\", \"value2\" : \"49.54\", \"value3\" : \"1005.14\"}");
```

- Post Plain Text

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

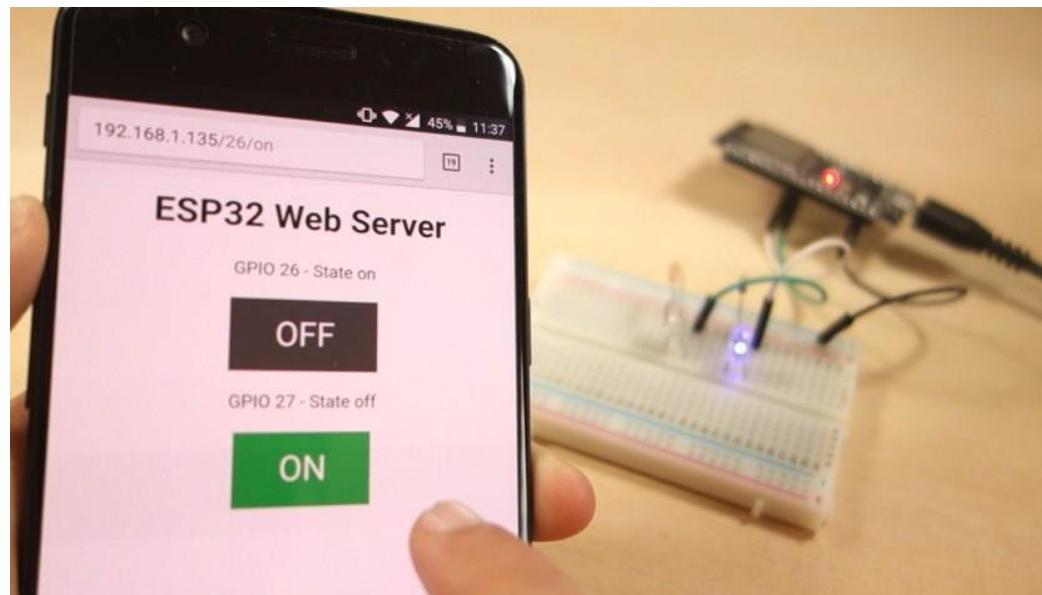
Use Arduino JSON library

- Sketch > Include Library > Manage Libraries
- Add library using Arduino Library Manager



ESP32 Web server

- Build a simple web server on ESP32: control
- Example:
 - <https://randomnerdtutorials.com/esp32-web-server-arduino-ide/>



ESP32 Web server

COM7

New Client.

GET /26/on HTTP/1.1

Host: 192.168.1.135

Connection: keep-alive

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8

Referer: http://192.168.1.135/

Accept-Encoding: gzip, deflate

Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7

GPIO 26 on

Client disconnected.

192.168.1.135/26/on

192.168.1.135/26/on

ESP32 Web Server

GPIO 26 - State on

OFF

GPIO 27 - State off

ON

Autoscroll

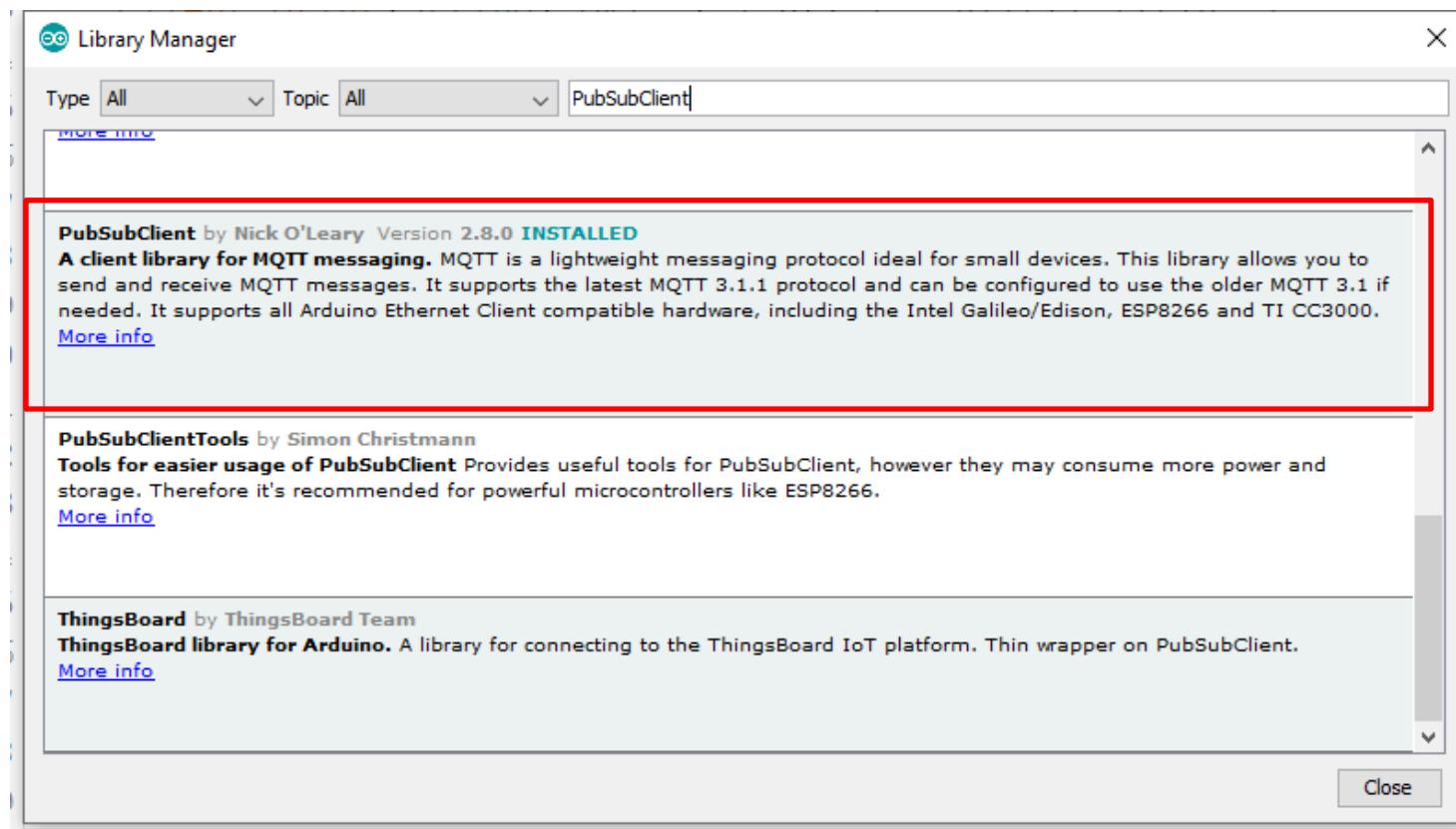
Both NL & CR

115200 baud

Clear output

D. Communicating via MQTT with ESP32

- Use the mqtt PubSubClient library for Arduino
 - <https://www.arduinolibraries.info/libraries/pub-sub-client>
 - <https://github.com/knolleary/pubsubclient>



Communicating via MQTT with ESP32/ESP8266

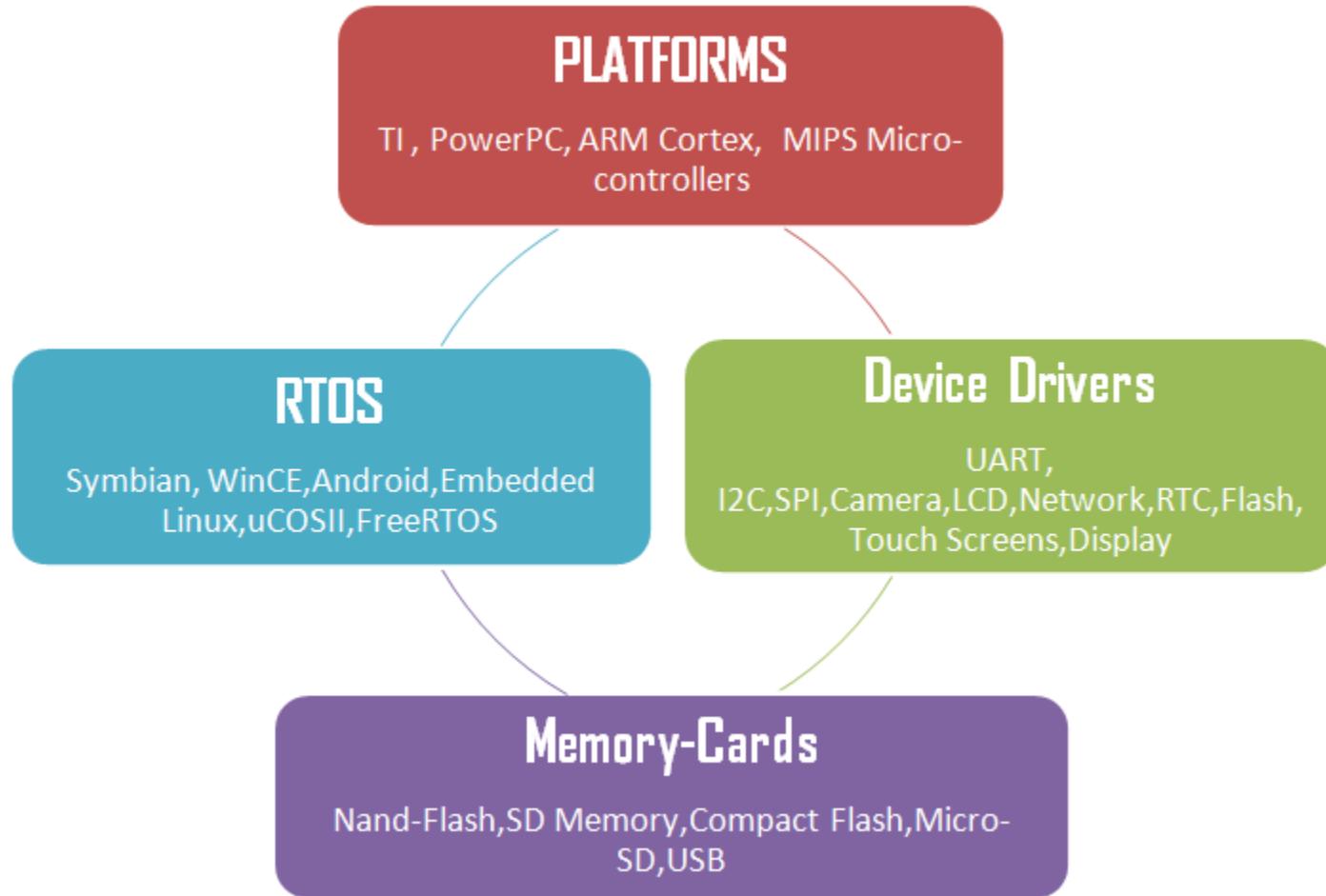
- Refer:

- 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() { ... }
```

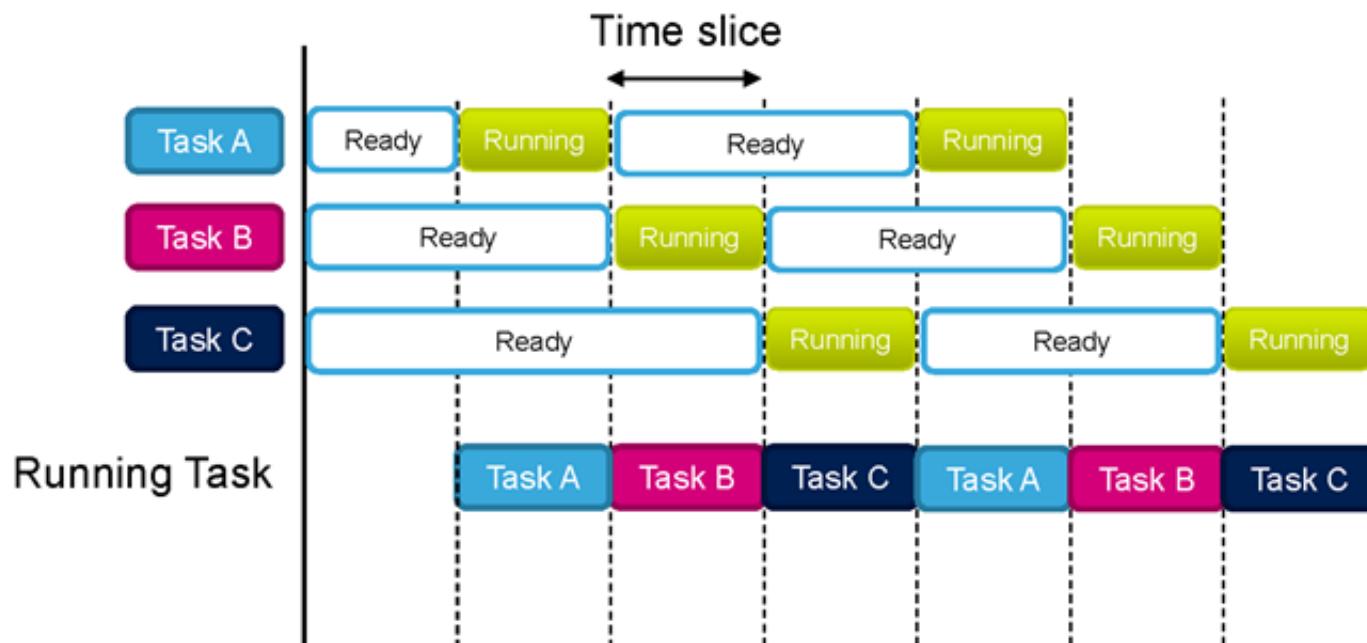
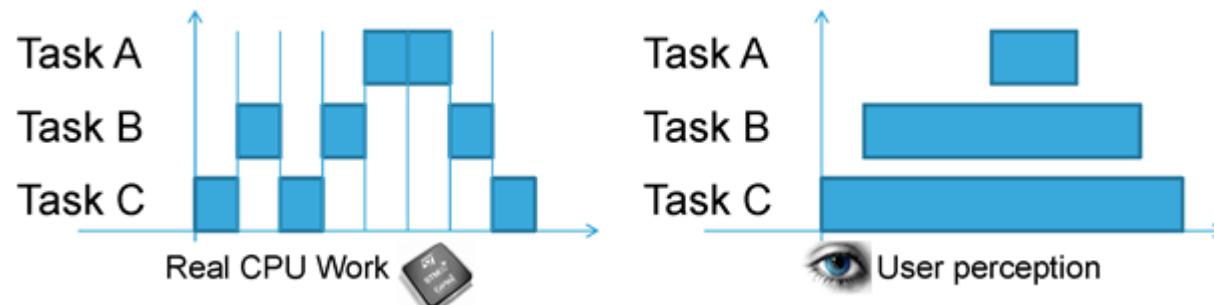
E. Programming multi-task with FreeRTOS

- RTOS = Real Time OS



E. Programming multi-task with FreeRTOS

- FreeRTOS multi-tasking



Programming multi-task with FreeRTOS

- Create a task

```
void toggleLED(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

- Create a task

```
void setup() {  
    xTaskCreate(  
        toggleLED, // Function that should be called  
        "Toggle LED", // Name of the task (for debugging)  
        1000, // Stack size (bytes)  
        NULL, // Parameter to pass  
        1, // Task priority  
        NULL// Task handle );  
}
```

<https://savjee.be/2020/01/multitasking-esp32-arduino-freertos/>

<https://savjee.be/2019/07/Home-Energy-Monitor-ESP32-CT-Sensor-Emonlib/>

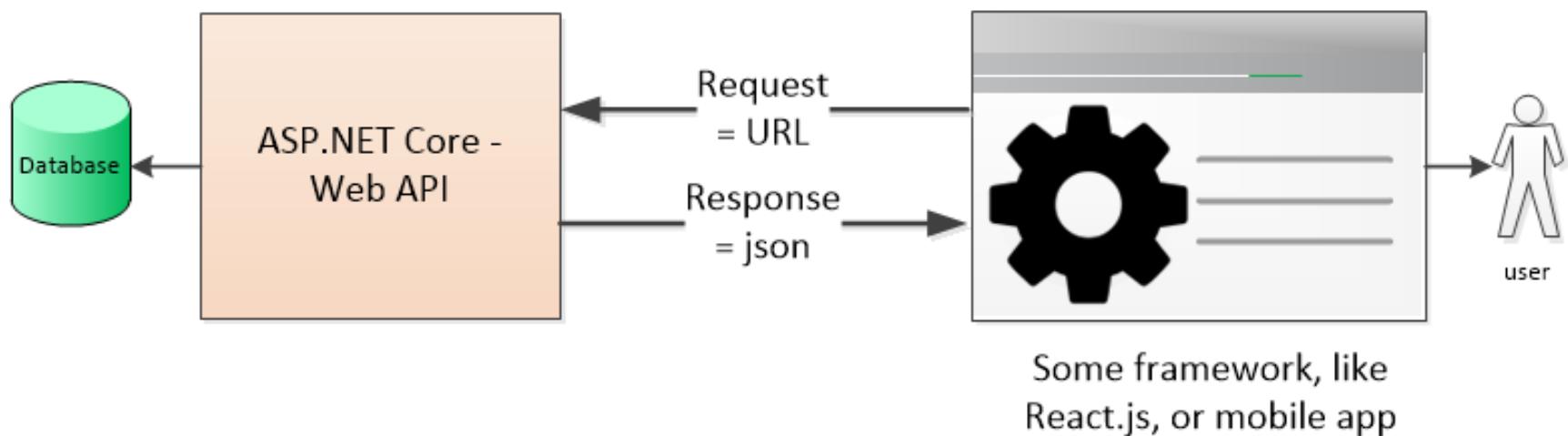
3.2. Building an IoT Server

3.2.1 .Net Core, Entity Framework

3.2.2. NodeJS, MongoDB

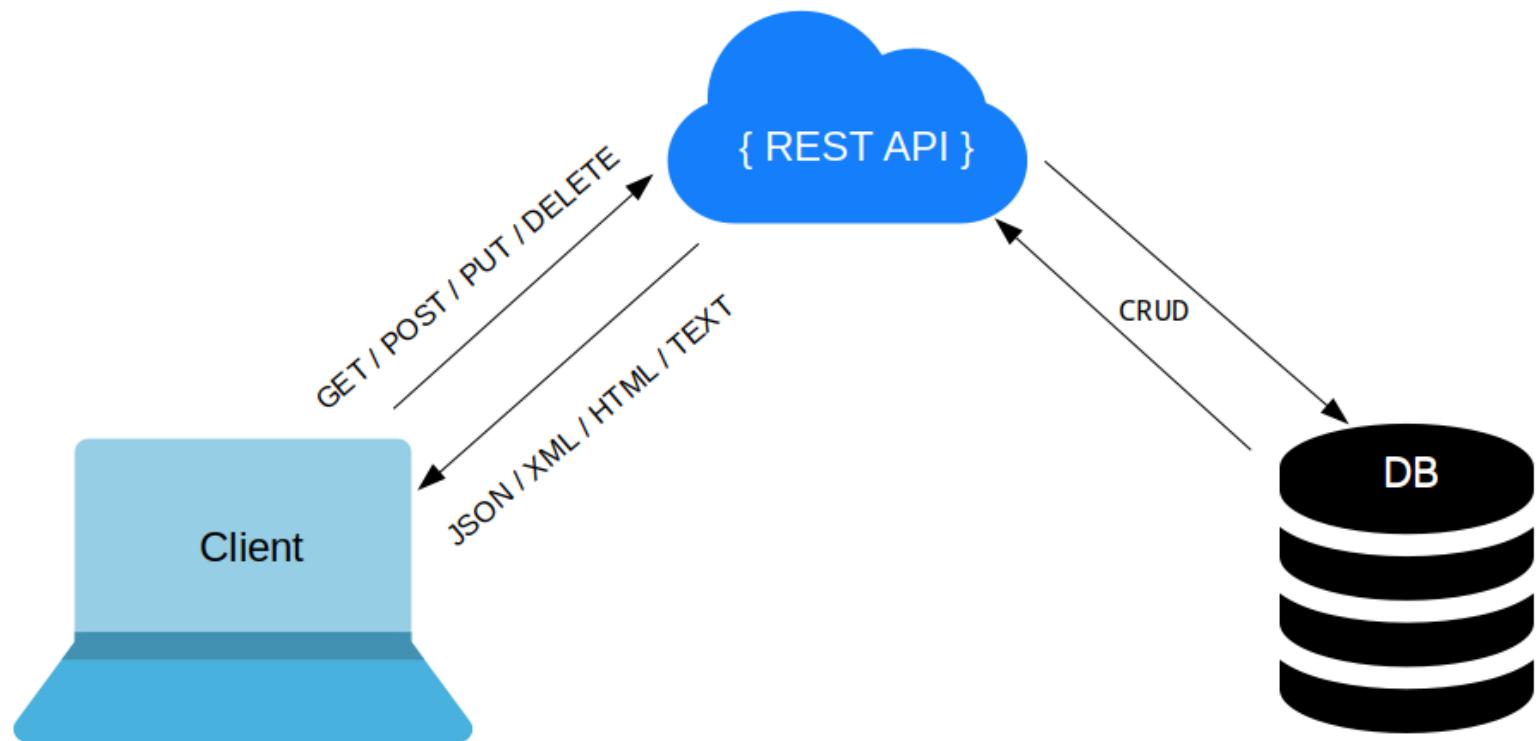
3.2.1. Building Web API with .NET Core

- Use ASP.NET Core and Entity Framework to build Web API for IoT application



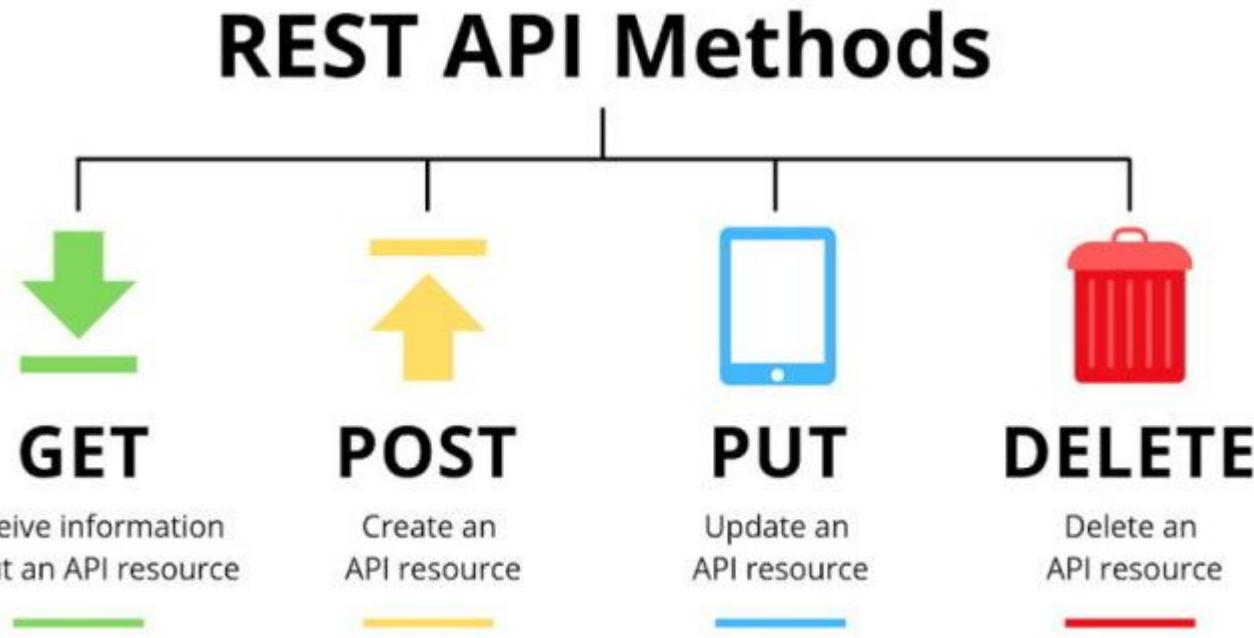
RESTful API

- REST = **R**epresentational **S**tate **T**ransfer: : Resource states are conveyed via HTTP.
- A standard used in Web API design



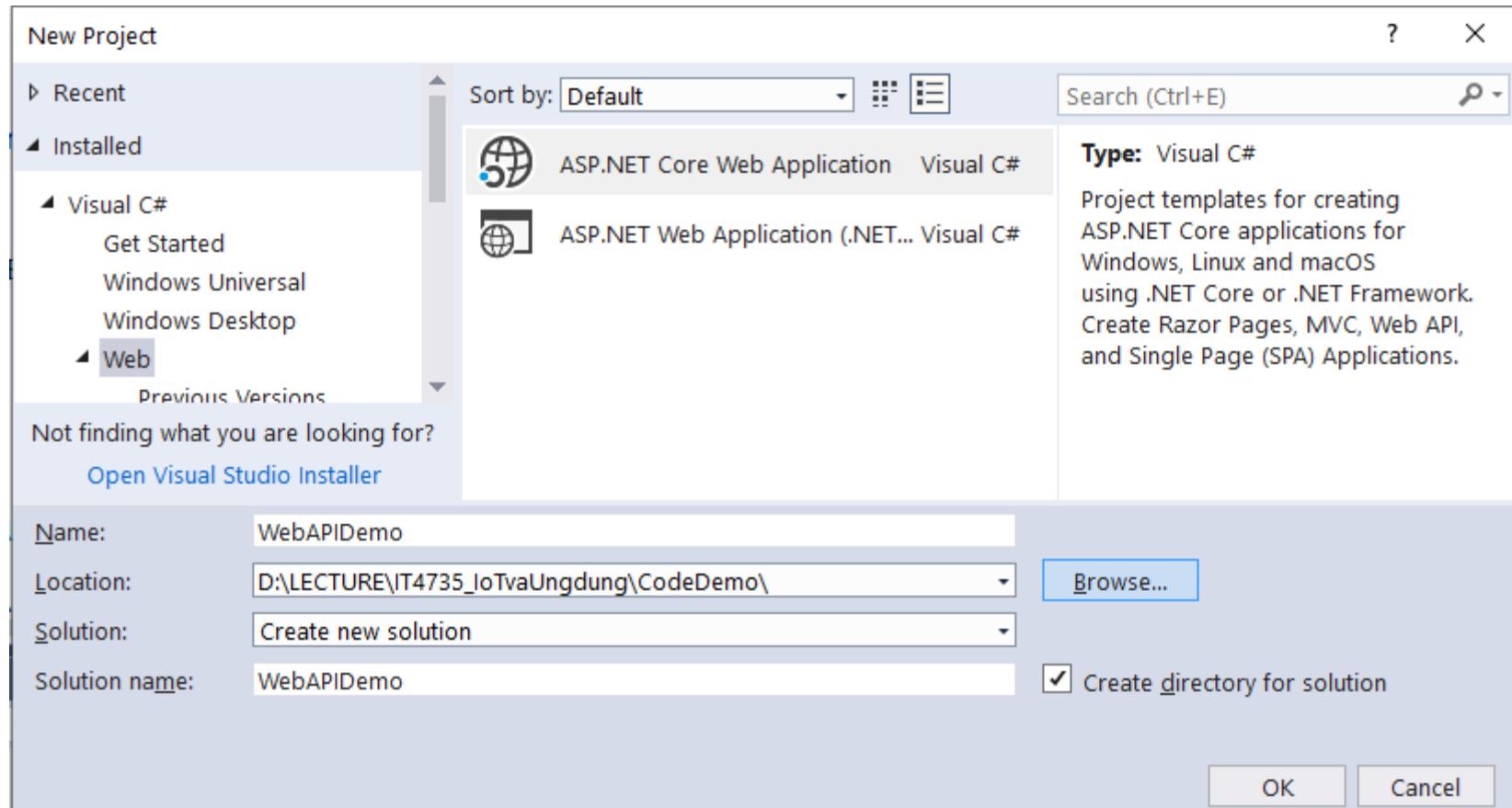
RESTful API

- Based on HTTP. Use HTTP methods:
 - GET: Retrieve (READ) specific resource information (by Id) or a list of resources.
 - POST: Create a new resource.
 - PUT: UPDATE information for a resource (by Id).
 - DELETE: DELETE a resource by ID.



Create Web API project

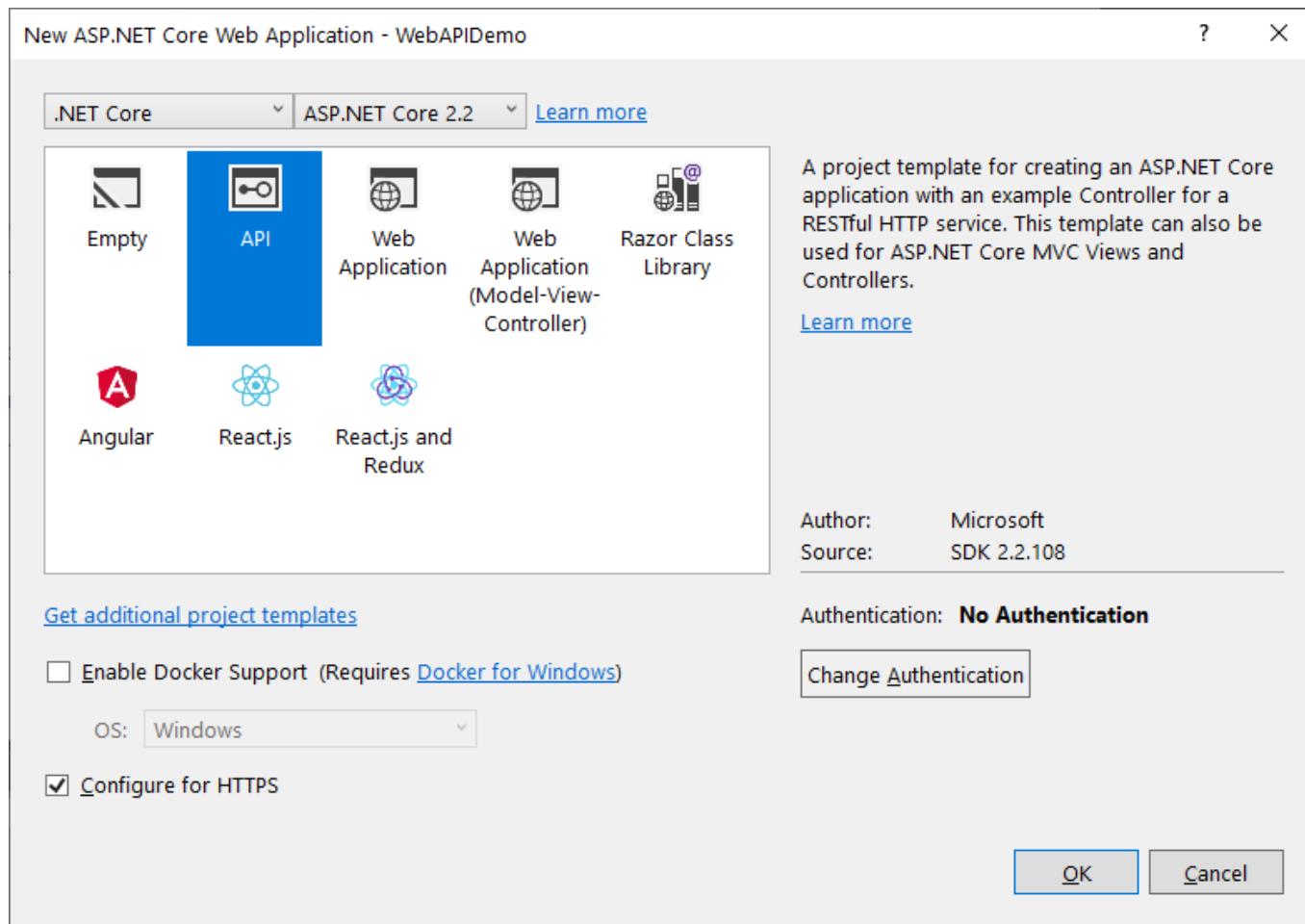
- Open File/New/Project in Visual Studio



<https://medium.com/net-core/how-to-build-a-restful-api-with-asp-net-core-fb7dd8d3e5e3>

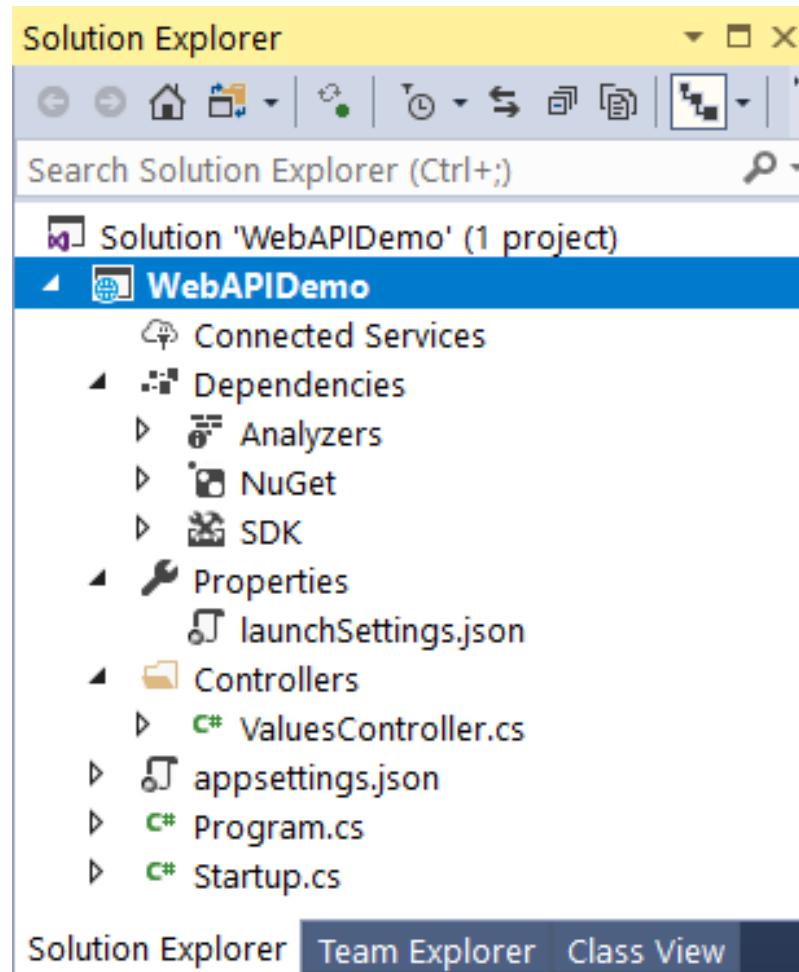
Create Web API project

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



Create Web API project

- Project WebAPIDemo



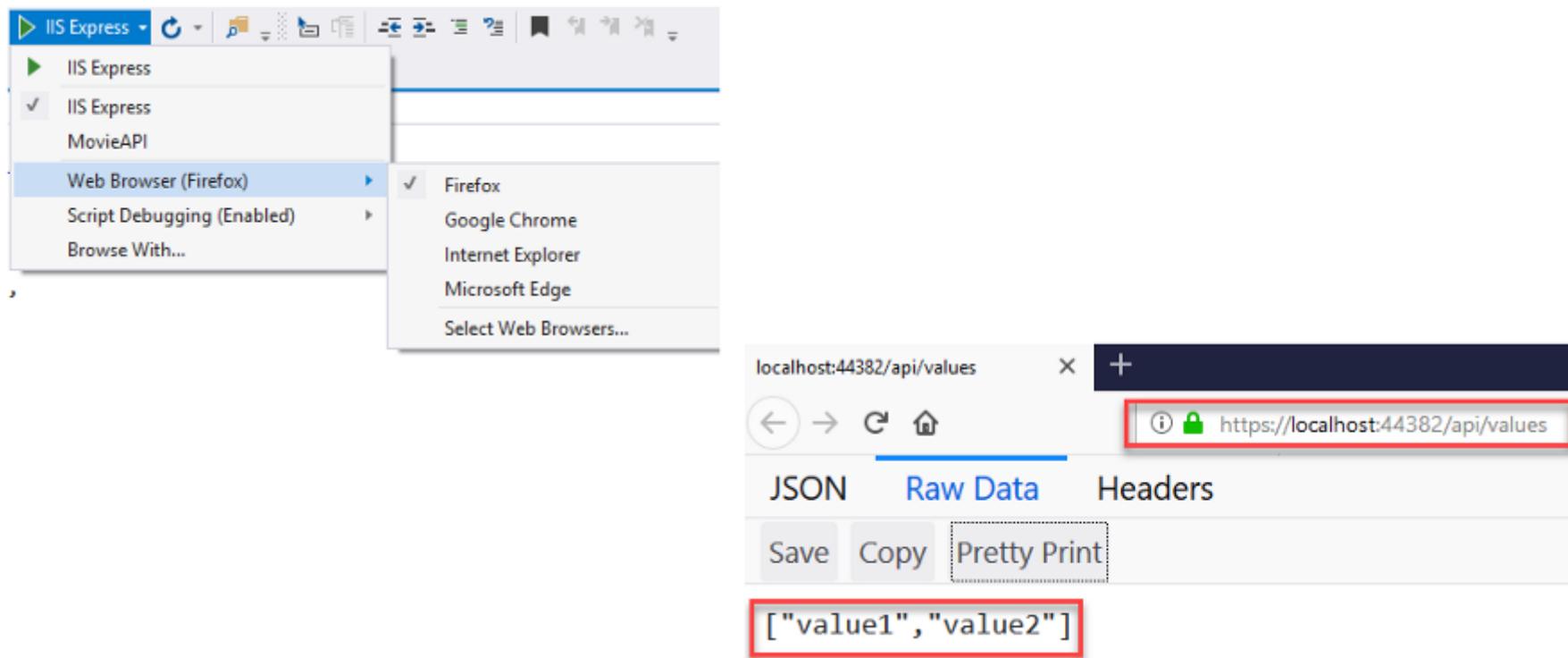
Create Web API project

- MVC (Model-View-Controller) is added in 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)
- Demo results on Web Browser



Routing and URL paths

- The default URL `https://localhost:{port}/api/values` is configured in the `launchUrl` in the file ***Properties\launchSettings.json***
- The values obtained in the browser are by default in the Get method of the ValuesController

The screenshot shows two code files in Visual Studio: `launchSettings.json` and `ValuesController.cs`.

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" };
    }
    ...
}
```

Annotations highlight specific parts of both files:

- `launchUrl: "api/values"` in `launchSettings.json` is highlighted.
- `[Route("api/[controller]")]` and `[ApiController]` in `ValuesController.cs` are highlighted.
- `// GET api/values` and `[HttpGet]` in `ValuesController.cs` are highlighted.

Build a model

- Build data model class (**Model** in MVC architecture)
- 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.

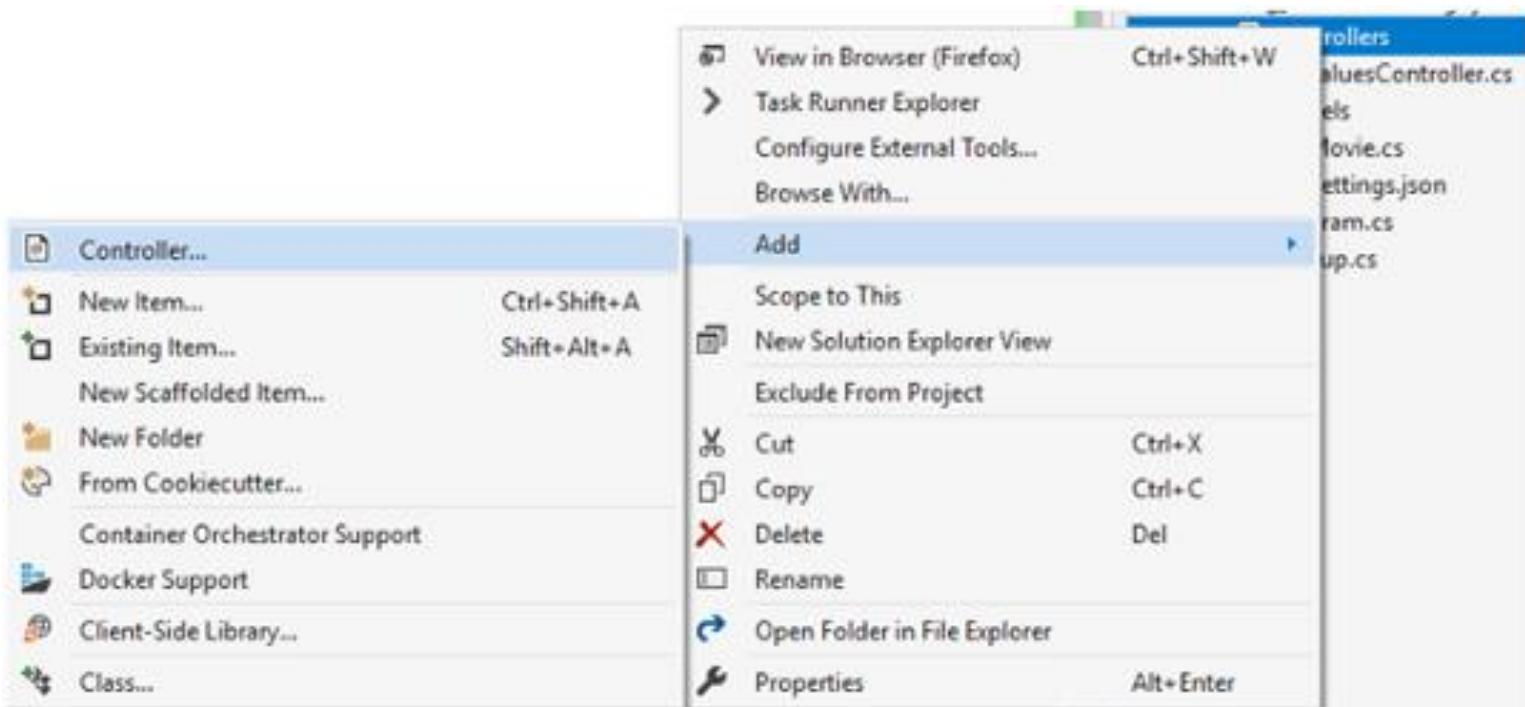
Build a model

```
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; }
    }
}
```

Create a controller

- Create a controller (Controller component in MVC architecture), associated with the SensorData model
- Use Entity Framework Core (EF Core) to work with the database



Create a controller

Add Scaffold

Installed

Common Controller

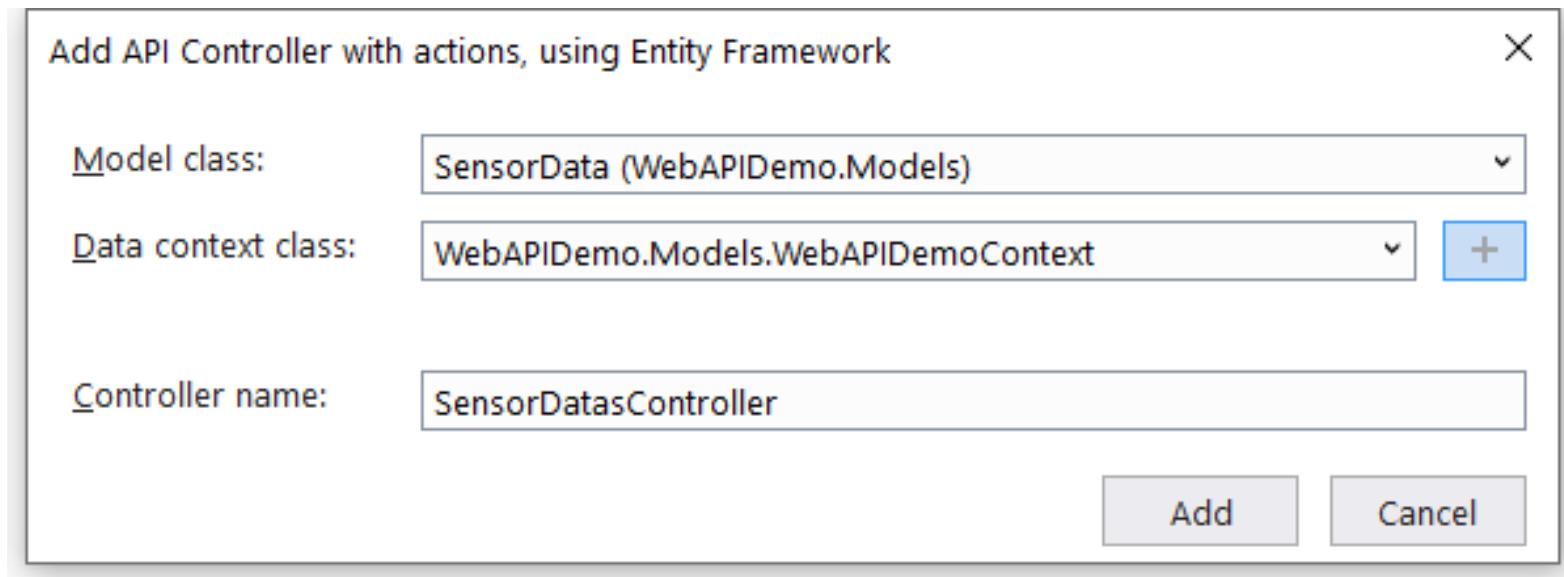
 MVC Controller - Empty	API Controller with actions, using Entity Framework by Microsoft v1.0.0.0
 MVC Controller with read/write action	An API controller with REST actions to create, read, update, delete, and list entities from an Entity Framework data context.
 MVC Controller with views, using Entity Framework	Id: ApiControllerWithContextScaffolder
 API Controller - Empty	
 API Controller with read/write actions	
 API Controller with actions, using Entity Framework	

[Click here to go online and find more scaffolding extensions.](#)

Add Cancel

Create a controller

- **scaffolding** tool in .NET Core supports automatic code generation
- The automatic creation of the database context and **CRUD** (**C**reate, **R**ead, **U**pdate, and **D**elete) action methods is known as scaffolding.



Controller Class

- Example: SensorDatasController class with methods:
 - GetSensorData (GET method)
 - PostSensorData (POST method)
 - PutSensorData (PUT method)
 - DeleteSensorData (DELETE method)
- Controller class inherits from ControllerBase

```
public class SensorDatasController : ControllerBase
```

```
namespace WebAPIDemo.Controllers
```

SensorDatasController.cs

```
{  
    [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](#).

```
using Microsoft.EntityFrameworkCore;
namespace WebAPIDemo.Models
{
    5 references
    public class WebAPIDemoContext : DbContext
    {
        0 references | 0 exceptions
        public WebAPIDemoContext (DbContextOptions<WebAPIDemoContext> options) ...
        6 references | 0 exceptions
        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-  
  }  
}
```

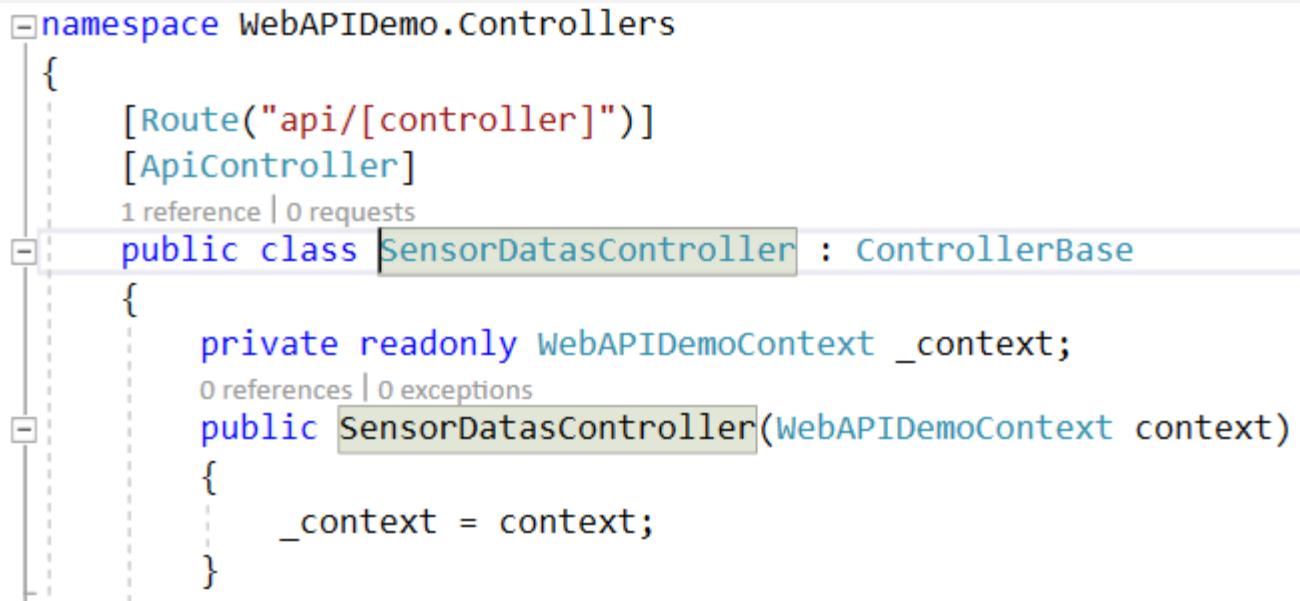
The screenshot shows the Visual Studio code editor with the JSON file open. The file contains configuration settings for logging, allowed hosts, and connection strings. The connection string for 'WebAPIDemoContext' is highlighted with a red border. The connection string value is 'Server=(localdb)\\mssqllocaldb;Database=WebAPIDemoContext-56113fd8-'.

EF Core Database Context

- Register DB Context (WebAPIDemoContext) in 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:



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

```
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;  
        }  
    }  
}
```

The `SensorDatasController` class is highlighted with a light gray background. The constructor parameter `WebAPIDemoContext context` is also highlighted with a light gray background.

Create a database using Migrations

- Use Entity Framework code first
- Migrations tool create database that corresponds to the constructed data model (code first) and updates the database schema when there are changes in the data model.
- Execute the commands of the Migrations tool:
 - Open *Tools -> NuGet Package Manager > Package Manager Console (PMC)*, execute command:
 - Add-Migration Initial (Create database)
 - Update-Database (Update database schema)

Create a database using Migrations

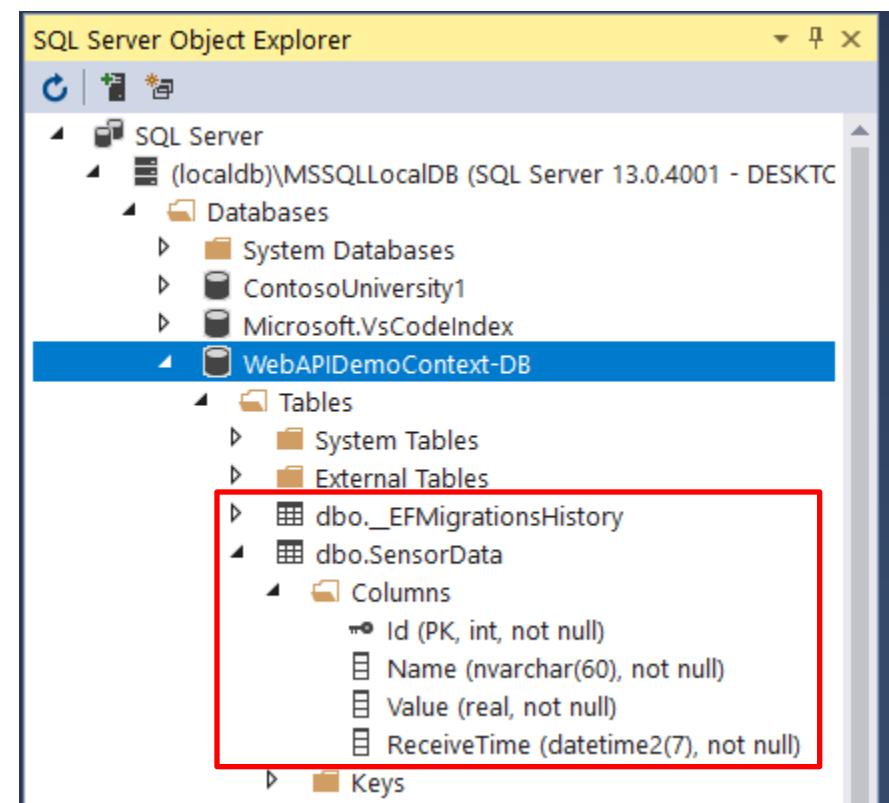
- Migration files are automatically generated in the Migration folder

```
namespace WebAPIDemo.Migrations
{
    1 reference
    public partial class Initial : Migration
    {
        0 references | 0 exceptions
        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);
                });
        }
        0 references | 0 exceptions
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "SensorData");
        }
    }
}
```

Create a database using Migrations

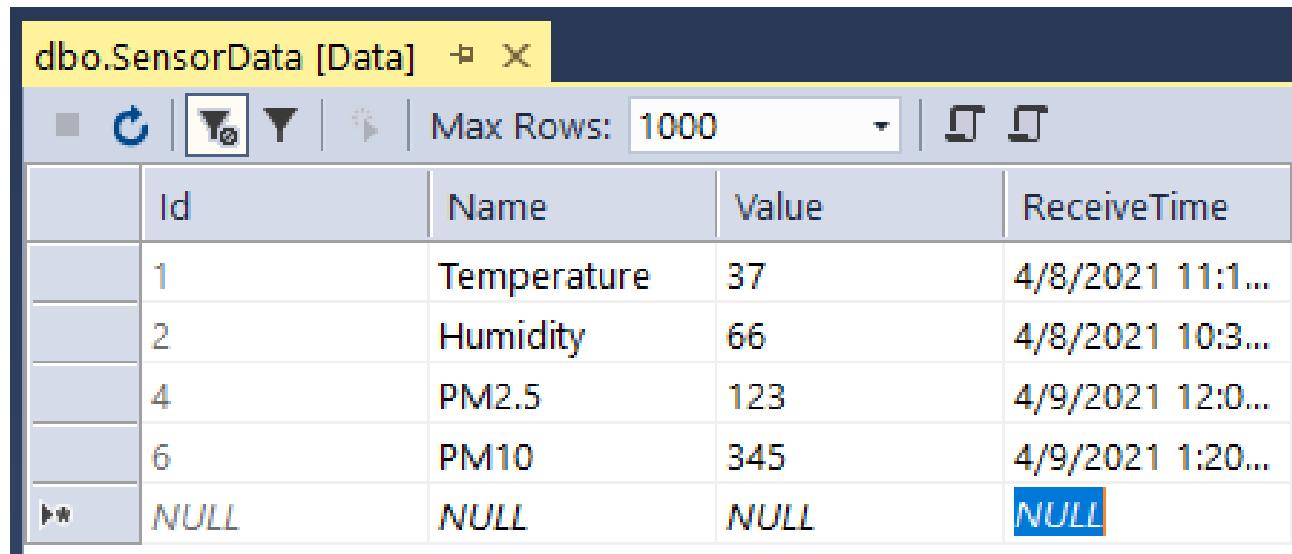
- Execute command `Update-Database` (in PMC)
- Up method in file `Migrations/{time-stamp}_Initial.cs` will be executed and will create a database
- Open SQL Server Object Explorer

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



Create a database

- Add data to the SensorData table
- Open *SQL Server Object Explorer*, right-click on the SensorData table, select *View Data* to add a record.



The screenshot shows a Microsoft SQL Server Management Studio (SSMS) window titled "dbo.SensorData [Data]". The table has five rows of data:

	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(): returns all records in the SensorData database.
- GetSensorData(int id): returns the corresponding record with the specified Id
- [HttpGet]: method handles HTTP GET requests

```
// 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}
- Call HTTP GET request from browser:
 - `https://localhost:{port}/api/SensorDatas`
 - `https://localhost:{port}/api/SensorDatas/2`
- Return type **ActionResult<T> type**
 - .NET Core automatically serializes object to JSON and write it to the body of the response packet
 - Response code: **200** (if there are no unhandled exceptions), otherwise, it returns a **5xx** error code

GET method

- Browser testing

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

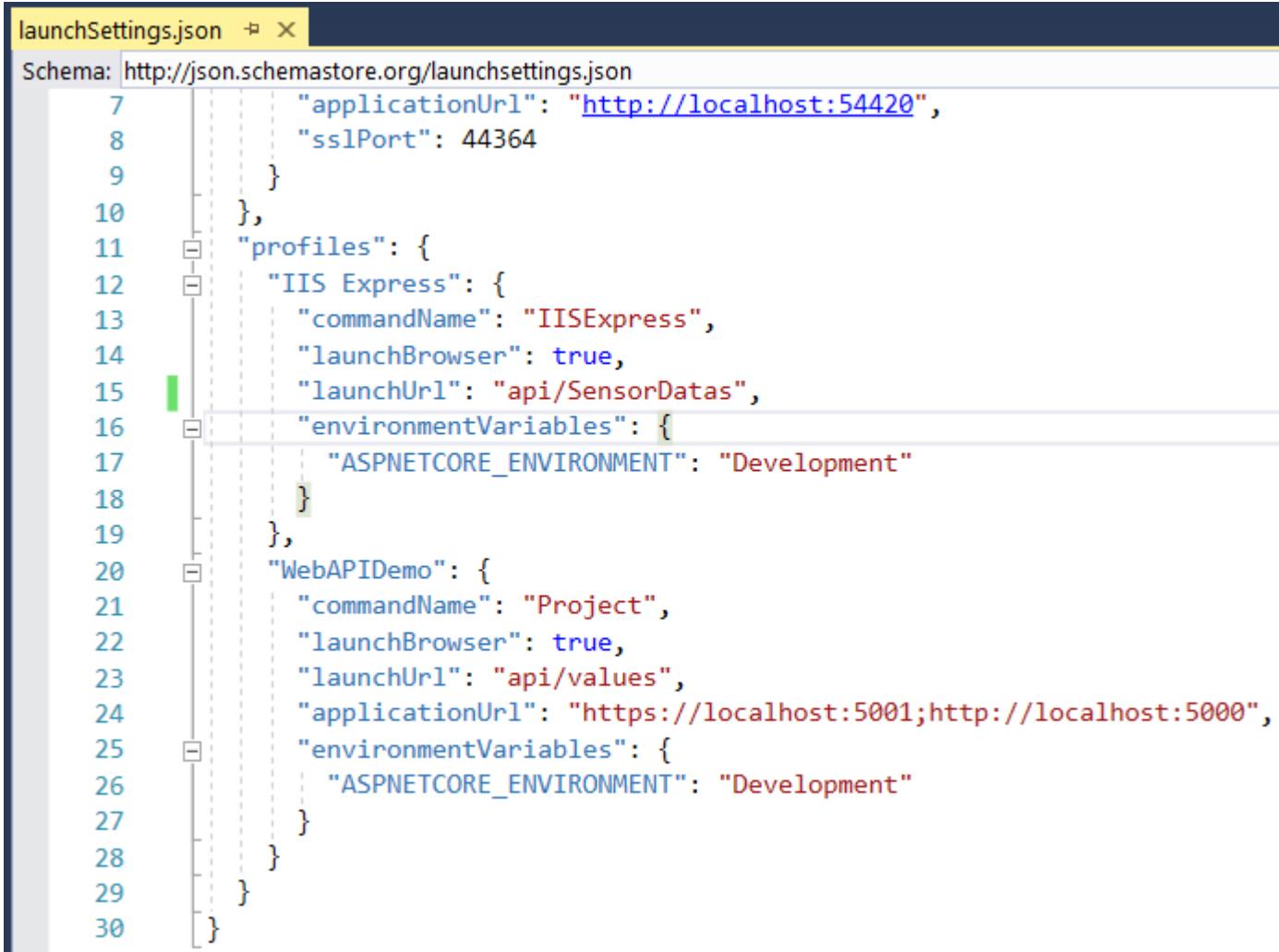
[
{
  "id": 1,
  "name": "Temperature",
  "value": 37.0,
  "receiveTime": "2021-04-08T11:16:20"
},
{
  "id": 2,
  "name": "Humidity",
  "value": 66.0,
  "receiveTime": "2021-04-08T10:30:01"
},
```

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

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

GET method

- Default URL: Configure the launchUrl in the launchsettings.json file



The screenshot shows the Visual Studio code editor with the file "launchSettings.json" open. The file contains JSON configuration for a .NET Core application. The "profiles" section includes configurations for "IIS Express" and "WebAPIDemo". The "IIS Express" profile has a "launchUrl" set to "api/SensorDatas". The "environmentVariables" for both profiles include "ASPNETCORE_ENVIRONMENT": "Development". The "WebAPIDemo" profile has an "applicationUrl" set to "https://localhost:5001;http://localhost:5000". The code is numbered from 7 to 30.

```
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

- PostSensorData() function creates a new record in the database
- Data (record) is sent in the body of the HTTP POST request
- CreatedAtAction() function:
 - Return HTTP status code 201 if successful

```
// 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);
}
```

Testing using Postman

- GET method: get all resource

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** https://localhost:44364/api/SensorDatas
- Status:** 200 OK (green)
- Time:** 188 ms
- Size:** 692 B
- Save Response:** (red)
- Params:** (dropdown)
- Query Params:** (table)

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		
- Body:** (Pretty, JSON, copy, search)

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

Testing using Postman

- GET method: get 1 resource by Id

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** https://localhost:44364/api/SensorDatas/2
- Status:** 200 OK (114 ms, 394 B)
- Body:** JSON response (Pretty printed):

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

Testing using Postman

- POST method

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST' selected as the method, the URL 'https://localhost:44364/api/SensorDatas', and buttons for 'Send' and 'Save'. Below the header, the 'Body' tab is active, showing the JSON payload sent to the server. The response tab shows a successful '201 Created' status with response details like '1339 ms' and '446 B'. The response body is also displayed in JSON format.

Body

raw JSON Beautify

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

Body

Pretty JSON

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

PUT method

- The PutSensorData() method updates a record

```
// 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

- Return code: HTTP 204 (No Content)

The screenshot shows the POSTMAN application 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: This section shows the JSON payload being sent to the server. The JSON object has the following structure:

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

Response: This section shows the response from the server. The status is '204 No Content', the time taken is '1137 ms', and the size is '252 B'. There is also a 'Save Response' button.

DELETE method

- Delete a record (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

- Delete a record (by Id)
- Rerun code HTTP OK 200 (successful)

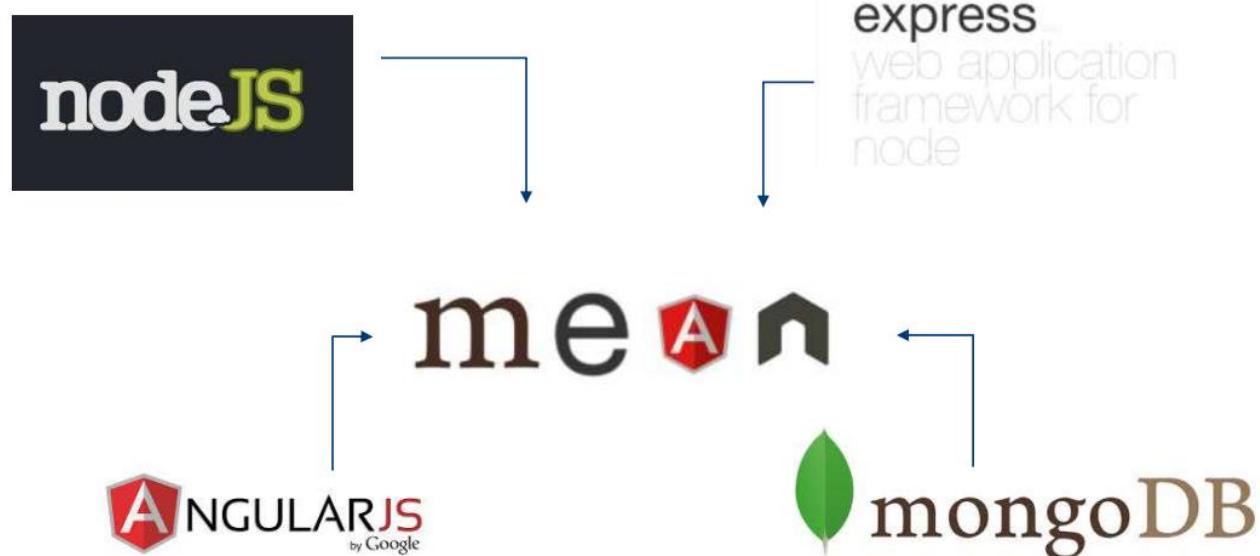
The screenshot shows a REST client interface with the following details:

- Method:** DELETE (highlighted with a red box)
- URL:** https://localhost:44364/api/SensorDatas/7 (highlighted with a red box)
- Status:** 200 OK (highlighted with a red box)
- Response Time:** 194 ms
- Response Size:** 392 B
- Save Response:** Save Response button
- Body:** JSON response (Pretty, JSON, Beautify buttons available)
- Response Content:** A JSON object representing a sensor data record:

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

3.2.2. NodeJS, MongoDB technologie

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



https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm

<https://blog.postman.com/how-to-create-a-rest-api-with-node-js-and-express/>

<https://www.youtube.com/watch?v=RfHBm5yRxuw>

3.3. Using IoT Cloud

- IoT Cloud services:
 - Collecting, Ingesting data
 - Cloud Storage, Database
 - Data Representation
 - Dashboard
- Some IoT Cloud service platforms:
 - IBM BlueMix
 - AWS IoT
 - Google Cloud IoT
 - Azure IoT



3.3.1. Using Azure IoT hub

- <https://sstechsolutioncom.wordpress.com/iot/iot-hub-esp32-dht11/>

The screenshot shows the Azure IoT Hub landing page. At the top left, there's a sidebar for "Build in the cloud free with Azure for Students" which includes "Start free", "Learn about eligibility", "Start with USD100 Azure credit", "No credit card required", and "Free services". The main content area has a dark header "Azure IoT Hub" and a sub-header "Connect, monitor, and manage billions of IoT assets." It features two buttons: "Try Azure IoT Hub free" (green) and "Create a pay-as-you-go account" (white). Below this are navigation links: Overview, Features, Security, Pricing, Get started, Resources, and FAQ. A central callout says "Build your IoT application with two-way communication". To the right, there's a detailed description of Azure IoT Hub's capabilities and four icons with descriptions at the bottom.

Azure IoT Hub

Connect, monitor, and manage billions of IoT assets.

Try Azure IoT Hub free Create a pay-as-you-go account

Overview Features Security Pricing Get started Resources FAQ

Build your IoT application with two-way communication

Enable highly secure and reliable communication between your Internet of Things (IoT) application and the devices it manages. Azure IoT Hub provides a cloud-hosted solution back end to connect virtually any device. Extend your solution from the cloud to the edge with per-device authentication, built-in device management, and scaled provisioning.

Security-enhanced communication channel for sending and receiving data from IoT devices

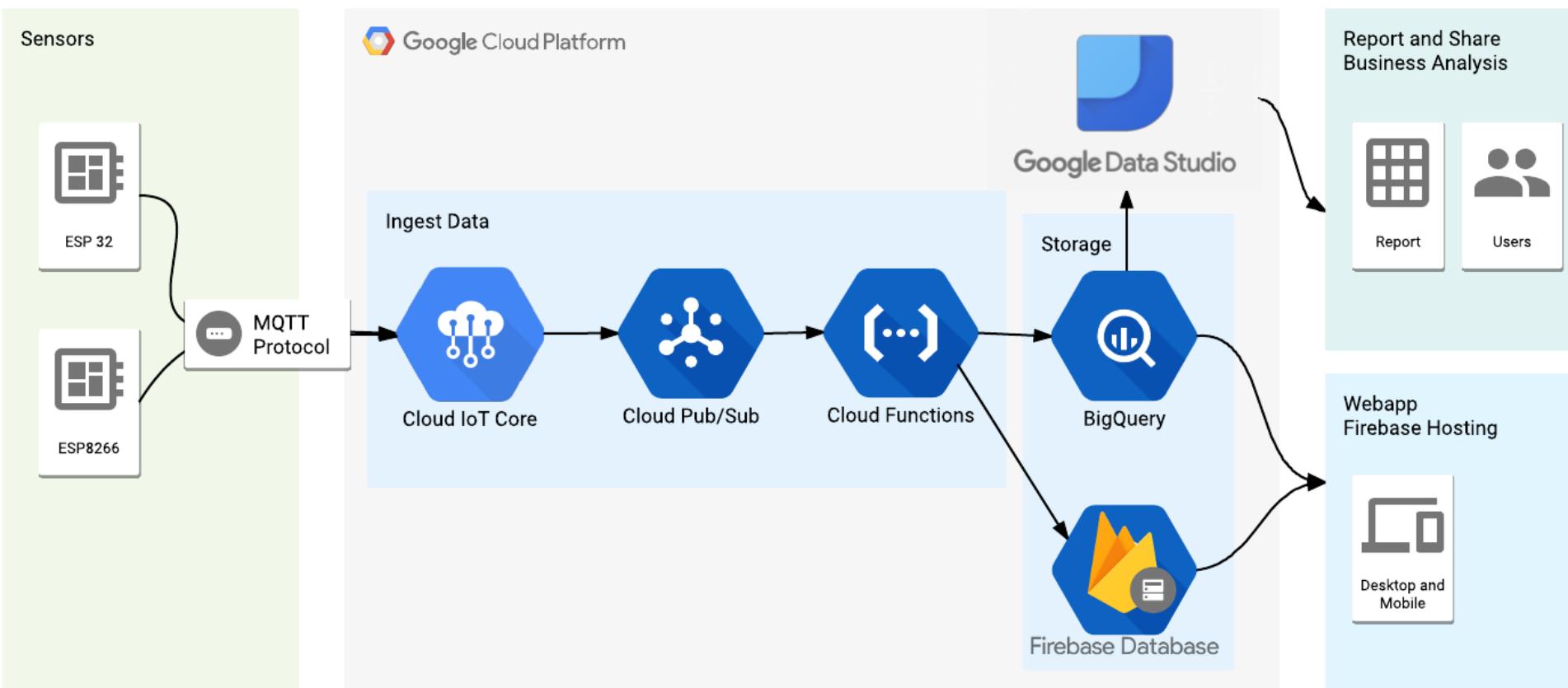
Device Update for IoT Hub enables over-the-air deployment of updates to help keep IoT devices up to date and secure

Full integration with Azure Event Grid and serverless compute, simplifying IoT application development

Compatibility with Azure IoT Edge for building hybrid IoT applications

3.3.2. Using Google Cloud IoT

- A tutorial: <https://medium.com/google-cloud/build-a-weather-station-using-google-cloud-iot-core-and-mongooseos-7a78b69822c5>



Content

- Chapter 1. Overview of IoT
- Chapter 2. IoT Technologies
- Chapter 3. IoT Application and Programming
- Chapter 4. IoT Safety and Security
- Chapter 5. Designing and Building IoT Systems

Chapter 4. IoT Safety and Security

- 4.1. Overview of IoT Security
- 4.2. Types of Attacks on IoT Infrastructure
- 4.3. The Security Vulnerabilities of IoT

4.1. Overview of IoT Security

- Issues in IoT Security:
 - Initial design for proprietary communication networks, later transitioning to IP and the Internet.
 - Firmware updates for IoT devices are challenging
 - Starting from basic security requirements, more complex security flaws have emerged.
 - Poor security in devices due to the utilization of inadequate initial designs in practical applications

Overview of IoT Security

- Classification of Risks in IoT Security:
 - **Capture:** Risks associated with 'capturing' (collecting, stealing) data from the system
 - **Disrupt:** Risks related to denial-of-service attacks, destruction, or interruption of the system
 - **Manipulate:** Risks associated with interference/change (manipulating) data and identity

Overview of IoT Security

- Security Requirements for IoT:
 - Confidentiality:
 - Data transmitted can only be read by the intended recipient.
 - Availability :
 - Communication between transmitting and receiving devices is always available
 - Integrity
 - Received data is not tampered with during transmission, ensuring the accuracy and completeness of the data.
 - Authenticity
 - The sender can always verify the sent data
 - Data can only be accessed by authorized recipients

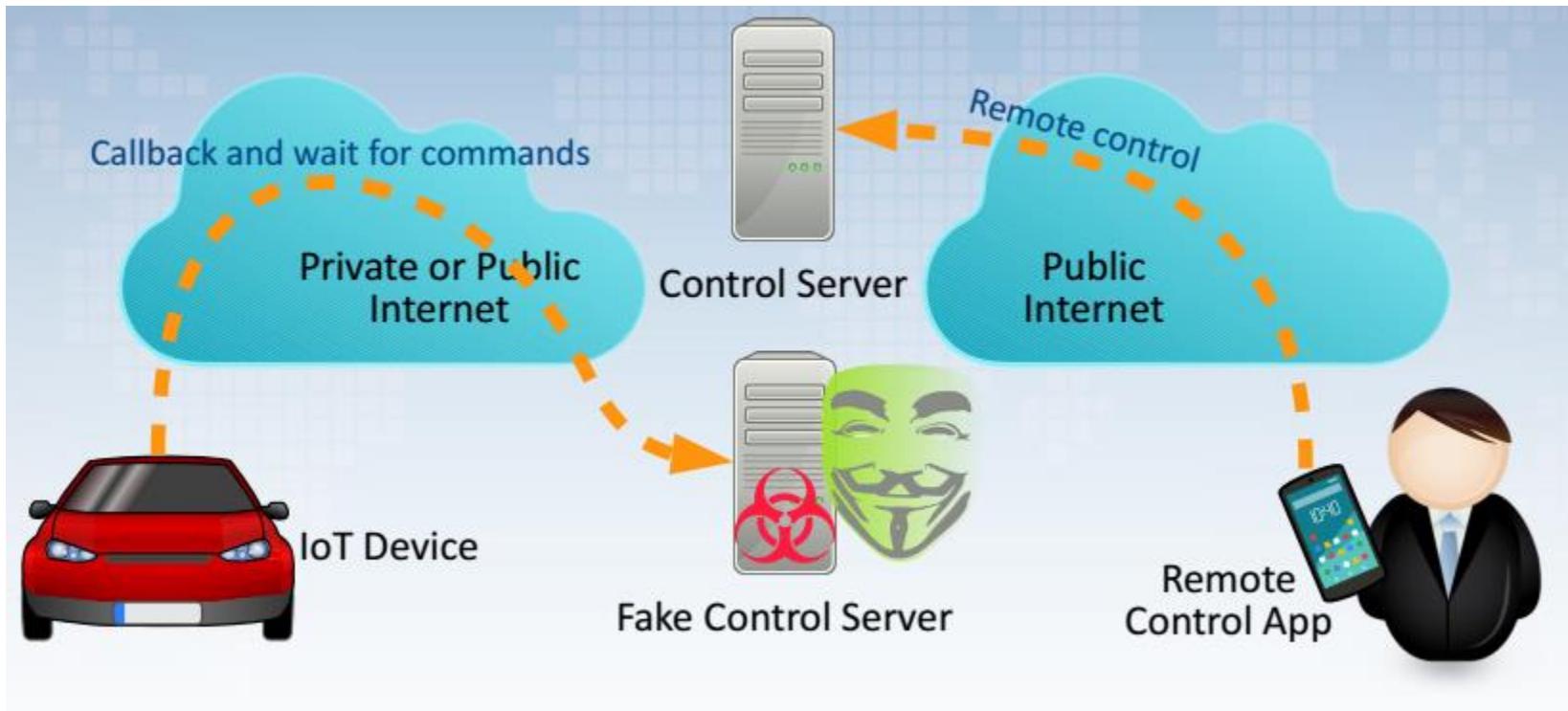
4.2. Types of Attacks on IoT Infrastructure

- Common IoT Infrastructure



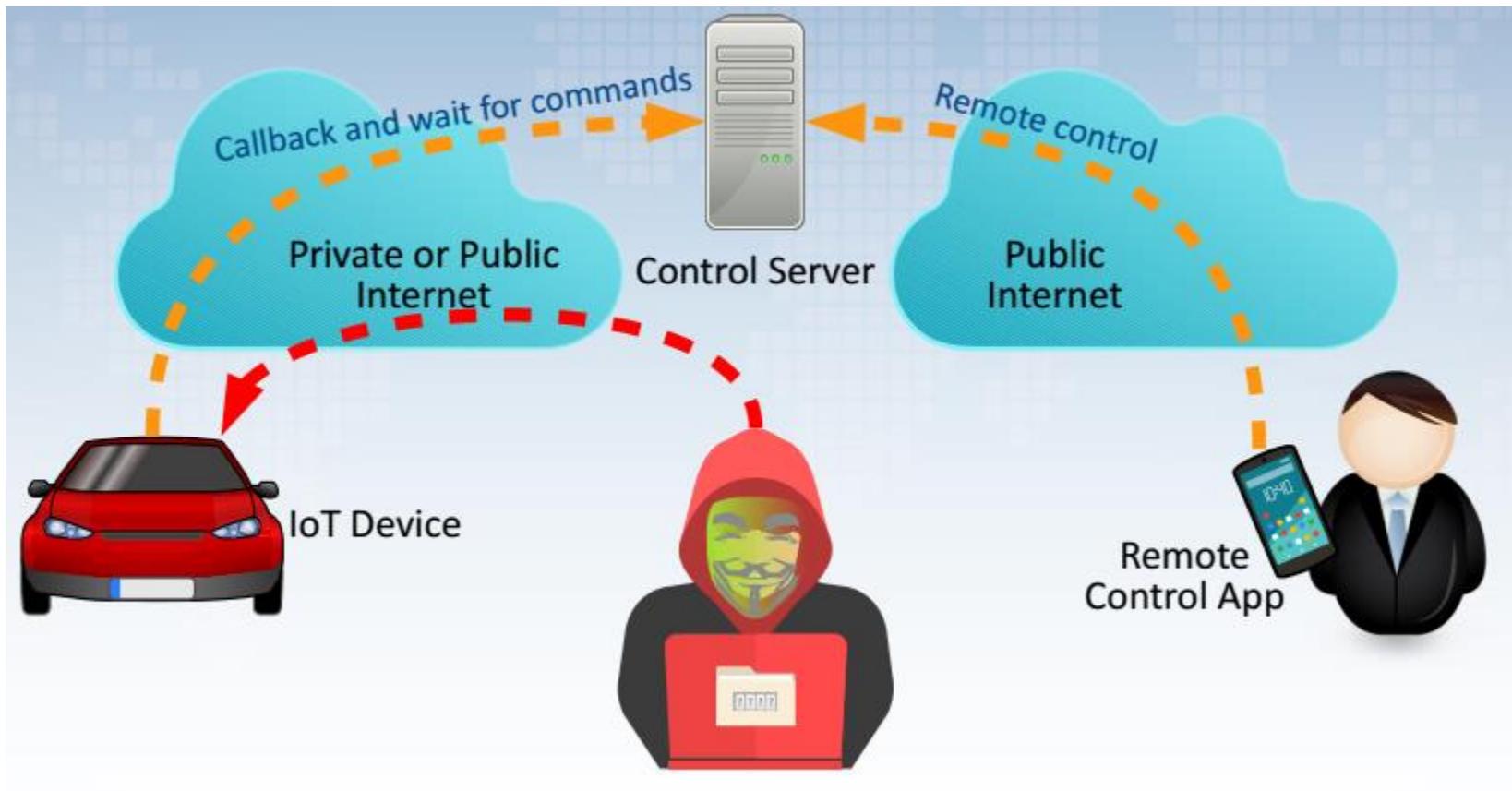
Types of Attacks on IoT Infrastructure

- Fake Control Server



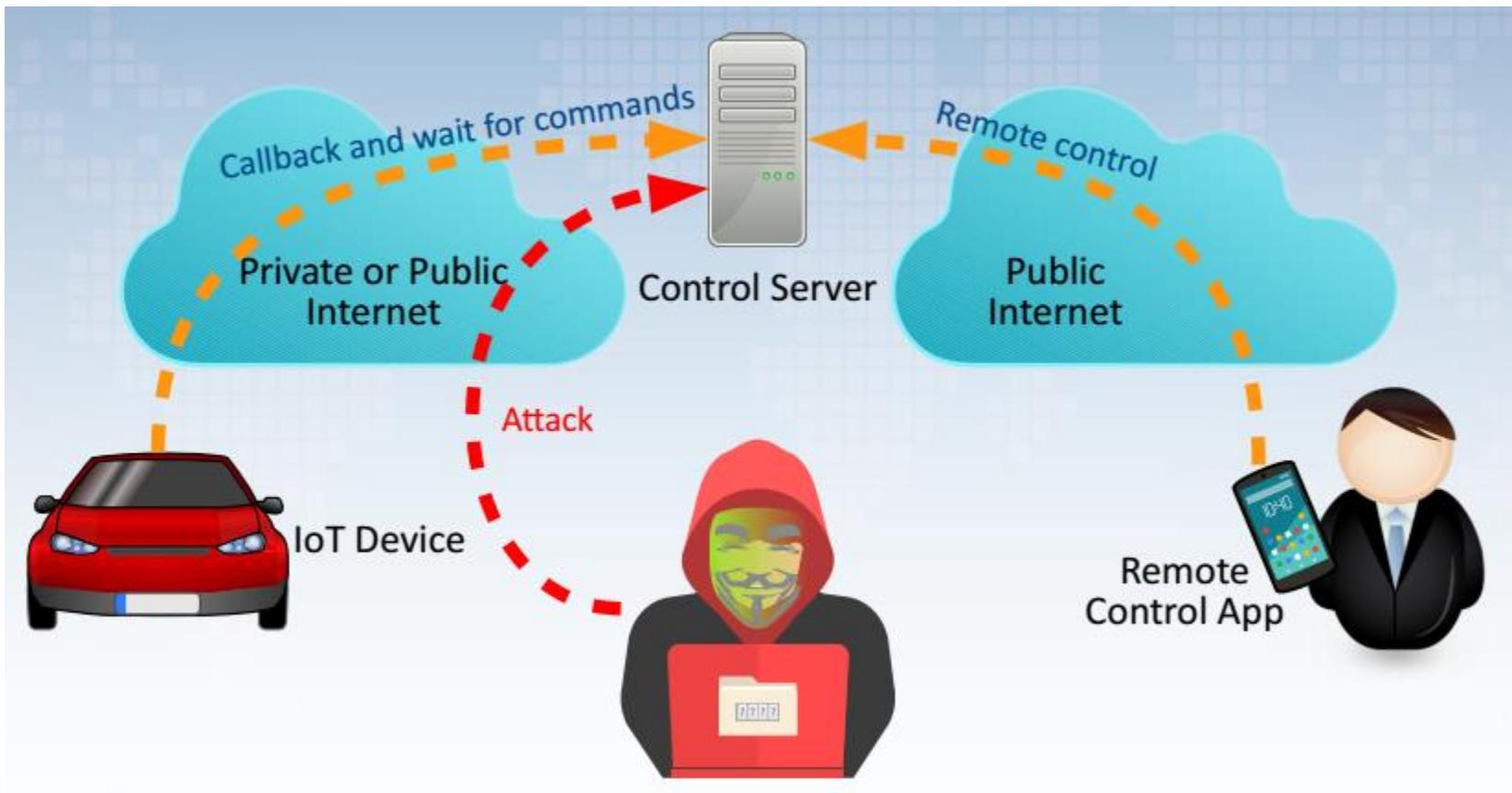
Types of Attacks on IoT Infrastructure

- Attack on device open ports



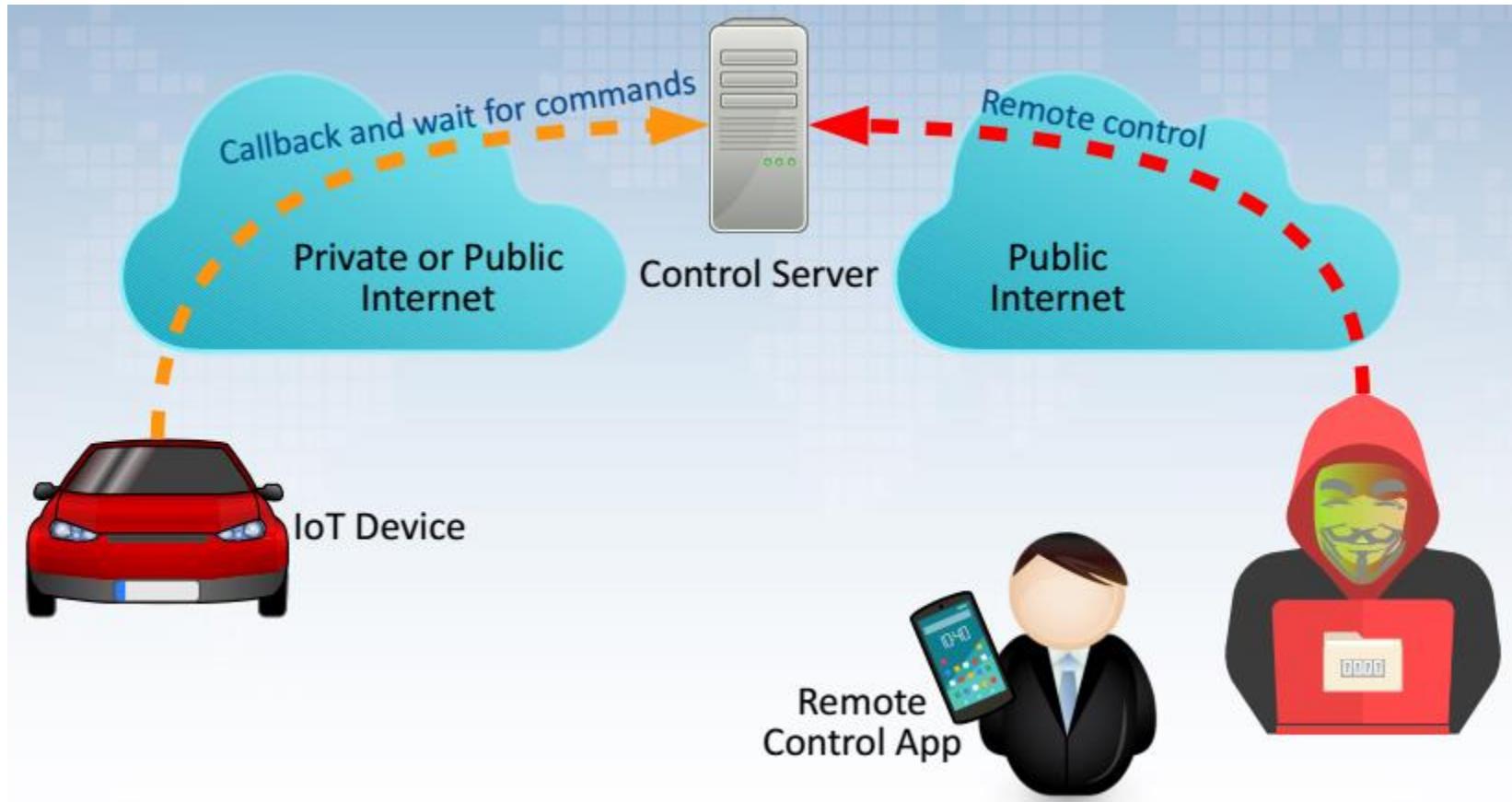
Types of Attacks on IoT Infrastructure

- Attack on server open ports



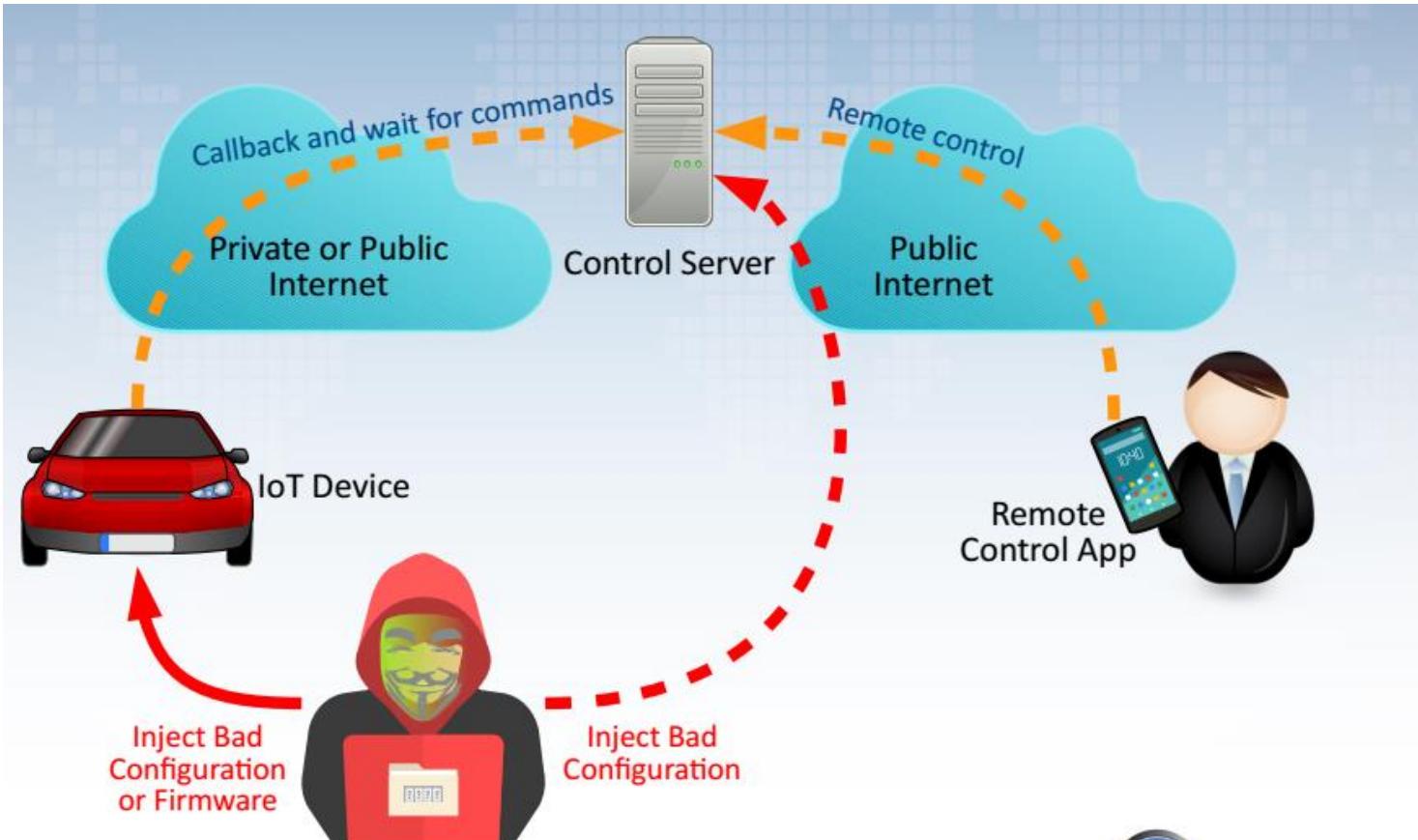
Types of Attacks on IoT Infrastructure

- Steal Credential



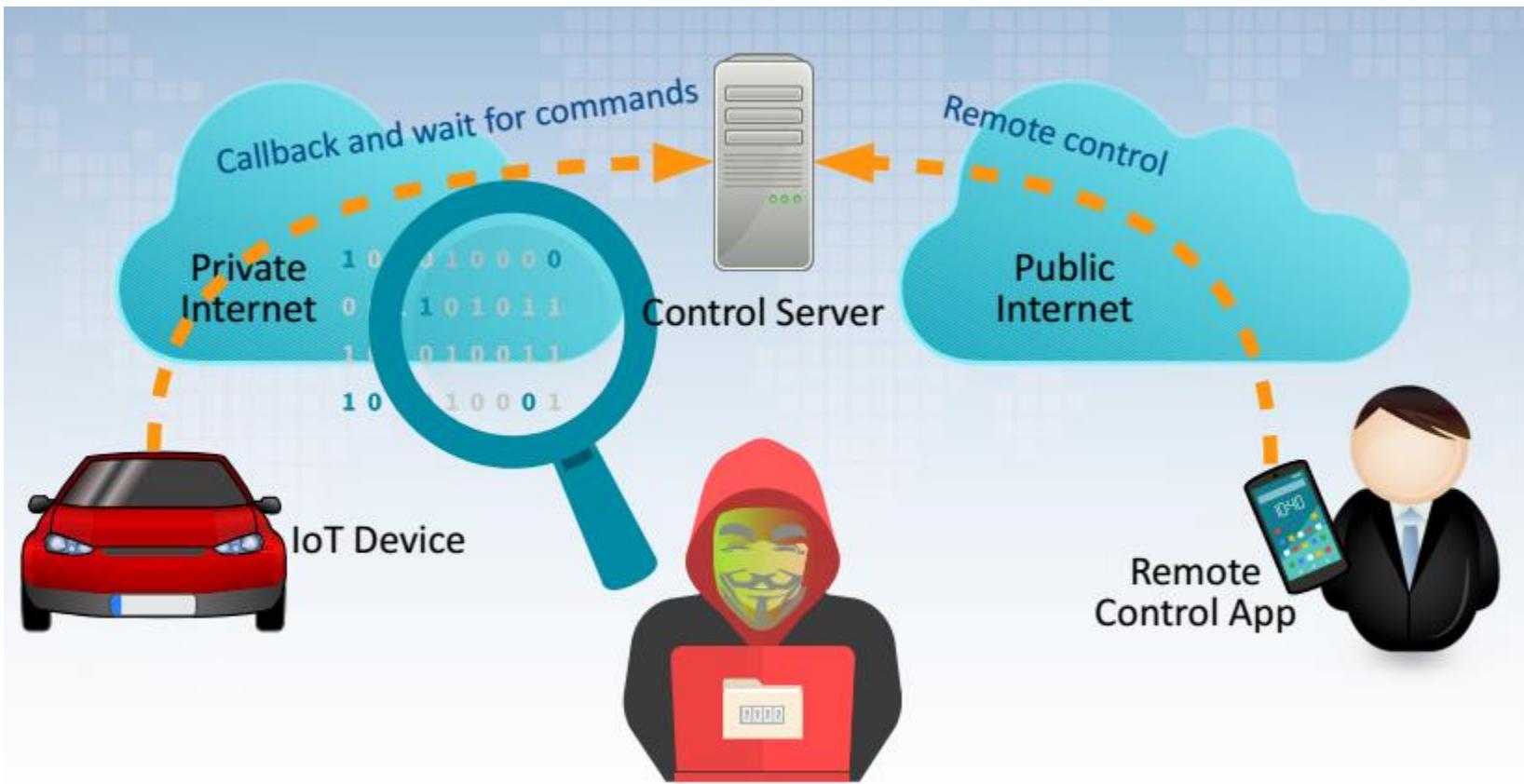
Types of Attacks on IoT Infrastructure

- Inject bad configuration or firmware



Types of Attacks on IoT Infrastructure

- Sniff data on private network



4.3. The Security Vulnerabilities of IoT

- I1. Insecure Web Interface
- I2. Insufficient Authentication
- I3. Insecure Network Services
- I4. Lack of Transport Encryption/Integrity Verification
- I5. Privacy Concerns
- I6. Insecure Cloud Interface
- I7. Insecure Mobile
- I8. Insufficient Security Configurability
- I9. Insecure Software/Firmware
- I10. Poor Physical Security

1. Insecure Web Interface



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. Insufficient Authentication/Authorization



The slide is from the OWASP Top 10 IoT 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 hand holding a smartphone connected to a network of icons representing various IoT devices like a computer monitor, a house, and a camera. A teal callout box in the center contains the title 'Insufficient Authentication/Authorization' and the subtitle 'covers all device interfaces and services'. A large number '2' is in a teal circle to the right of the callout. Below the callout, there are two columns: 'Obstacles' on the left and 'Solutions' on the right, separated by a vertical dotted line.

Insufficient Authentication/Authorization

covers all device interfaces and services

2

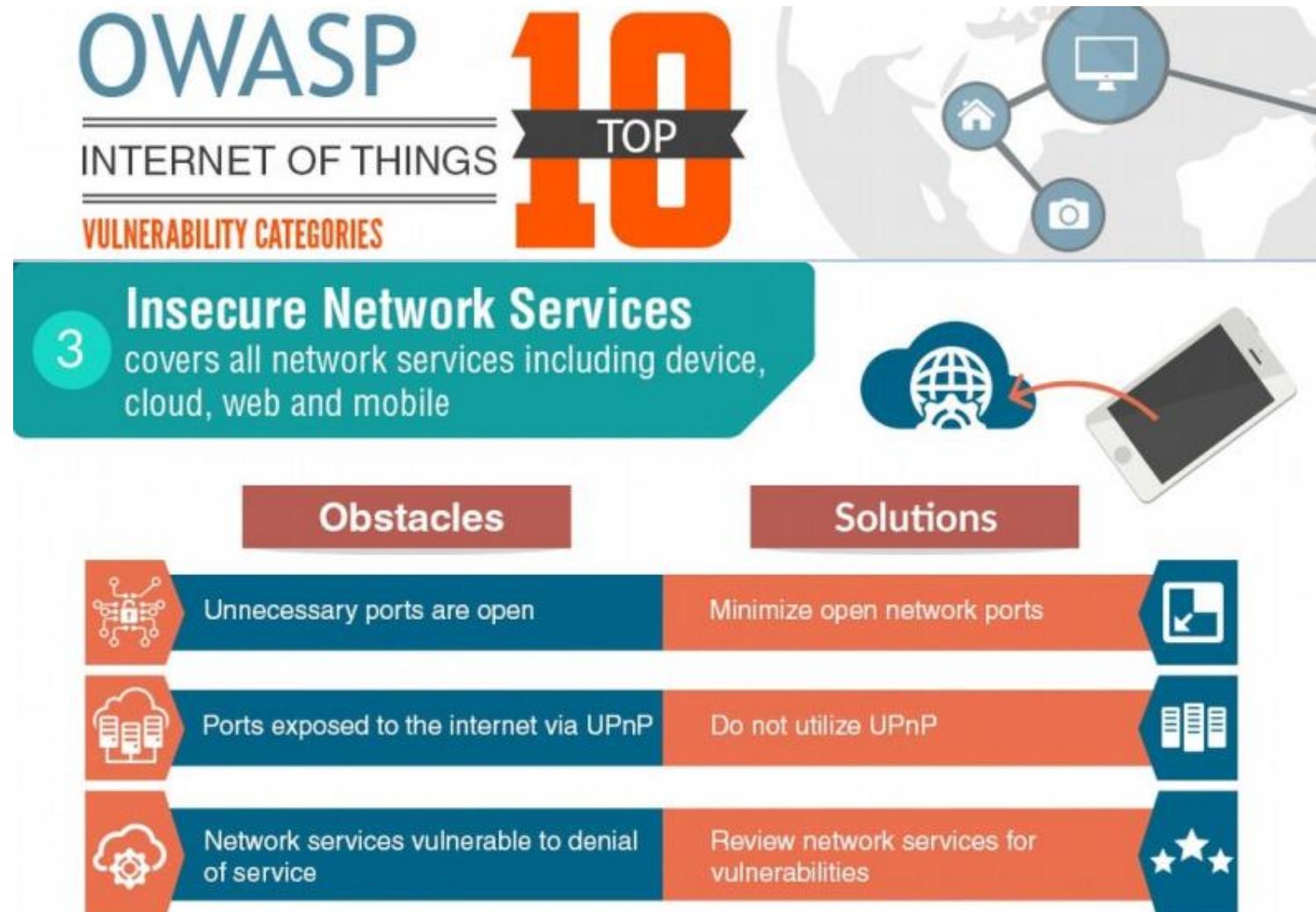
Obstacles

-  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. Insecure Network Services



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 above it, set against a background of a world map with network connections. Below this, a teal callout box contains the title "Insecure Network Services" and the number "3". The text "covers all network services including device, cloud, web and mobile" is also present. To the right of the callout is a diagram showing a smartphone connected to a cloud icon with a globe and a star.

Obstacles	Solutions
 Unnecessary ports are open	 Minimize open network ports
 Ports exposed to the internet via UPnP	 Do not utilize UPnP
 Network services vulnerable to denial of service	 Review network services for vulnerabilities

4. Lack of Transport Encryption/Integrity Verification



The slide is from the OWASP Internet of Things Top 10 Vulnerability Categories. It features a large orange '10' with 'TOP' written below it, and the text 'INTERNET OF THINGS' and 'VULNERABILITY CATEGORIES' in blue and red respectively. A world map background shows three devices connected: a computer monitor, a house icon, and a camera. On the right, a teal box contains the title 'Lack of Transport Encryption' and its description: 'covers all network services including device, cloud, web and mobile'. A green circle with the number '4' is positioned next to the title. Below the title, there are two sections: 'Obstacles' and 'Solutions'. The 'Obstacles' section lists three items: 'Sensitive information is passed in clear text', 'SSL/TLS is not available or not properly configured', and 'Proprietary encryption protocols are used'. The 'Solutions' section lists three items: 'Encrypt communication between system components', 'Maintain SSL/TLS implementations', and 'Do not use proprietary encryption solutions'. An illustration on the right shows a hand holding a megaphone over a laptop screen, with a magnifying glass examining a document.

10 TOP

INTERNET OF THINGS

VULNERABILITY CATEGORIES

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. Privacy Concerns



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 a "TOP" banner across it. Below this, a teal box contains the number "5" and the text "Privacy Concerns covers all components of IoT solution". To the right of the teal box is a graphic of a hand holding a smartphone connected to a network of icons representing various IoT components: a house, a computer monitor, and a camera.



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. Insecure Cloud Interface



The slide is part of the OWASP Top 10 IoT Vulnerability Categories series. It features a large orange '10' with 'TOP' written above it, and the text 'INTERNET OF THINGS' and 'VULNERABILITY CATEGORIES' below it. To the right is a stylized map of North America with three blue circular icons representing connected devices: a house, a computer monitor, and a camera. The main content area has a teal background. On the left, there's an illustration of a computer monitor displaying a globe icon, with a cloud icon containing a padlock icon to its right, separated by a double-headed arrow.

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

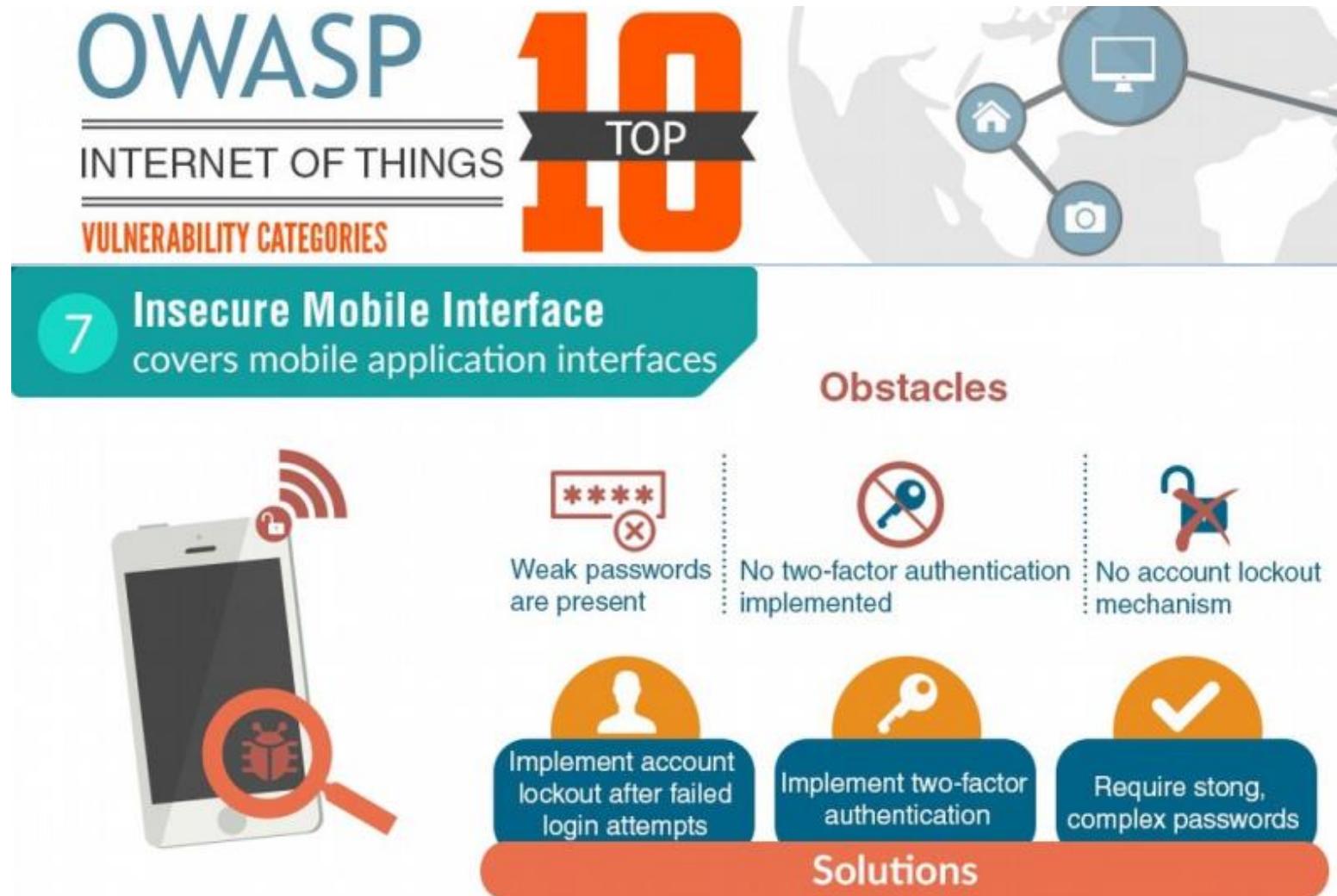
Obstacles

- 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. Insecure Mobile Interface



The slide is part of the OWASP Top 10 IoT Vulnerability Categories report. It 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 above it, set against a background of a hand holding a smartphone with a magnifying glass over it, and a network of icons (monitor, house, camera) connected by lines.

7 Insecure Mobile Interface
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



8. Insufficient Security Configurability



The slide is part of the OWASP Top 10 Internet of Things Vulnerability Categories. It features a large orange '10' with 'TOP' written below it, and the word 'Insufficient Security Configurability' followed by the number '8' in a teal circle. The background shows a map of North America with three blue circles connected by lines, each containing a white icon: a house, a computer monitor, and a camera. The slide is divided into sections: 'Obstacles' on the left and 'Solutions' on the right, separated by a central graphic of a server, laptop, and desktop computer with security icons.

OWASP
INTERNET OF THINGS
VULNERABILITY CATEGORIES

10 TOP

Insufficient Security Configurability 8
covers the IoT device

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. Insecure Software/Firmware



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 a dark grey banner across it that says "TOP". Below this, a teal box contains the number "9" and the title "Insecure Software/Firmware covers the IoT Device". To the right of the teal box is a small illustration of three interconnected icons: a house, a computer monitor, and a camera, set against a background of a world map. Further down the slide, there are two main sections: "Obstacles" and "Solutions". The "Obstacles" section has three items: "Update servers are not secured" (with a server icon), "Device updates transmitted without encryption" (with a hand touching a screen icon), and "Device updates not signed" (with a gear icon). The "Solutions" section has three items: "Sign updates" (with a shield and checkmark icon), "Verify updates before install" (with a gear and checkmark icon), and "Secure update servers" (with a shield and clock icon). Between the "Obstacles" and "Solutions" sections is a central illustration of a smartphone displaying a flame, with a virus-like character approaching it, and a shield and checkmark icon above the phone.

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

10. Poor Physical Security



The slide is part of the OWASP Internet of Things Top 10 Vulnerability Categories. It features a large orange '10' with 'TOP' written above it, and the text 'INTERNET OF THINGS VULNERABILITY CATEGORIES' below it. To the right is a stylized map of North America with three blue circular icons representing IoT devices: a house, a computer monitor, and a camera. The main title 'Poor Physical Security' is displayed in white text on a teal background banner, with the number '10' in a green circle to its right. Below the banner, the slide is divided into two sections: 'Obstacles' on the left and 'Solutions' on the right, separated by a central shield icon with a red 'X'.

Poor Physical Security

covers the IoT device

Obstacles

- Unnecessary external ports like USB ports
- Access to operating systems through removable media
- Inability to limit administrative capabilities

Solutions

- Minimize external ports like USB ports
- Properly protect operating system
- Include ability to limit administrative capabilities

Content

- Chapter 1. Overview of IoT
- Chapter 2. IoT Technologies
- Chapter 3. IoT Application and Programming
- Chapter 4. IoT Safety and Security
- Chapter 5. Designing and Building IoT Systems

Chapter 5. Designing and Building IoT Systems

- Project-Based Learning:
 - Learning Based on Solving a Specific IoT Problem/Project
- Skills:
 - Finding the problems
 - System design and Analysis
 - Implementing, Programming
 - Presentation, Report
 - Teamwork

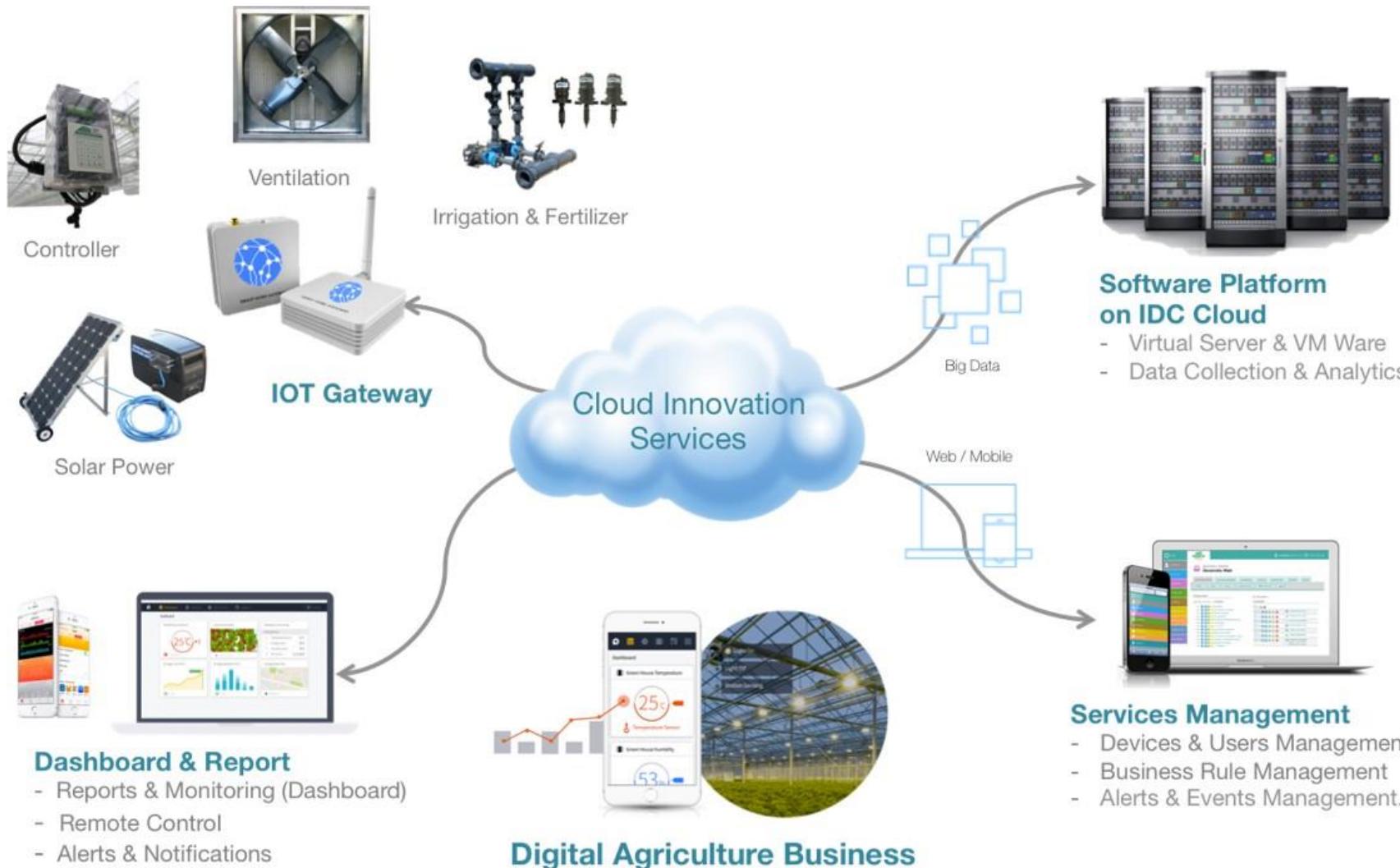
The process of conducting IoT project

- Form a team: 3-4 members
- Define the IoT application problem
- Analyze and design the IoT system:
 - Design the overall architecture of the system
 - Design the detailed components of the system
 - Select hardware devices: embedded computers, sensors
 - Choose IoT software/solution
- Build the system:
 - Program and deploy the system components
 - Apply IoT solutions, services, and technologies
- Presentation and report

Topics of IoT applications

- P1. Smart Agriculture:
 - IoT applications in Smart Agriculture

P1. Smart Agriculture



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

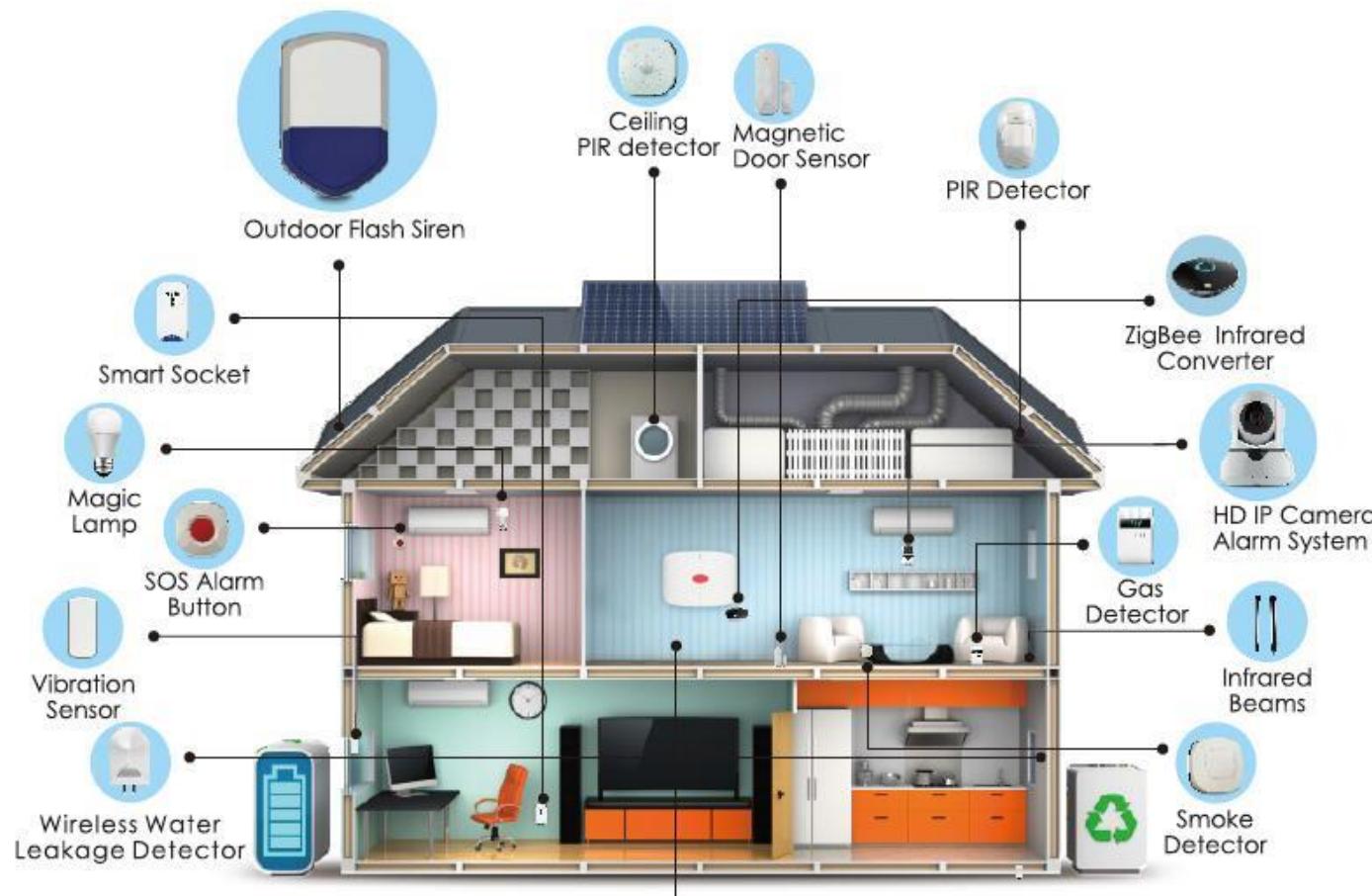
P2. Smart Home

- IoT applications in smart home:
 - Remote Device Control (Mobile App)
 - Voice-Controlled (Google assistant)
 - Remote Monitoring

P2. Smart Home

- Example:

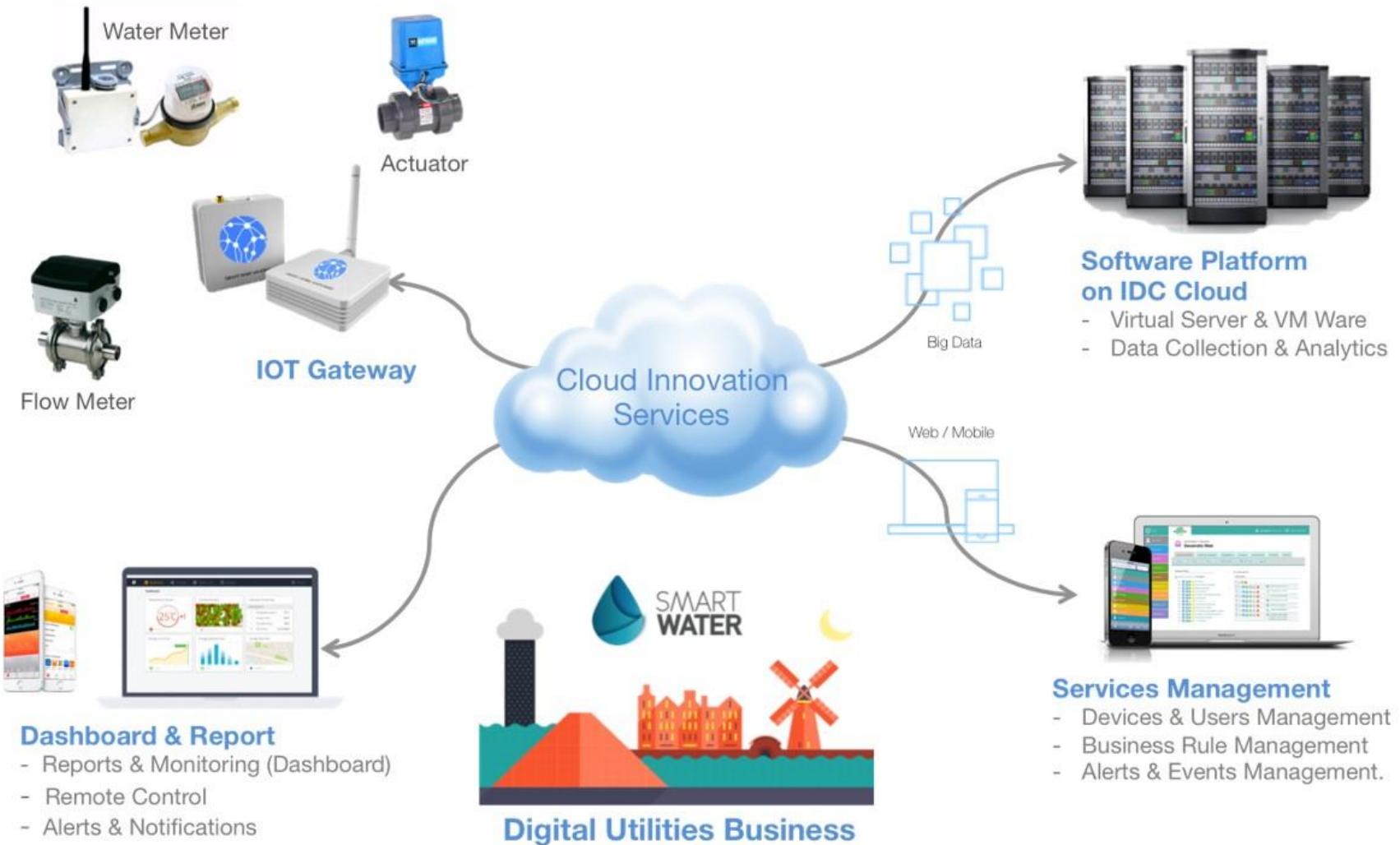
Smart Home



Topic of IoT applications

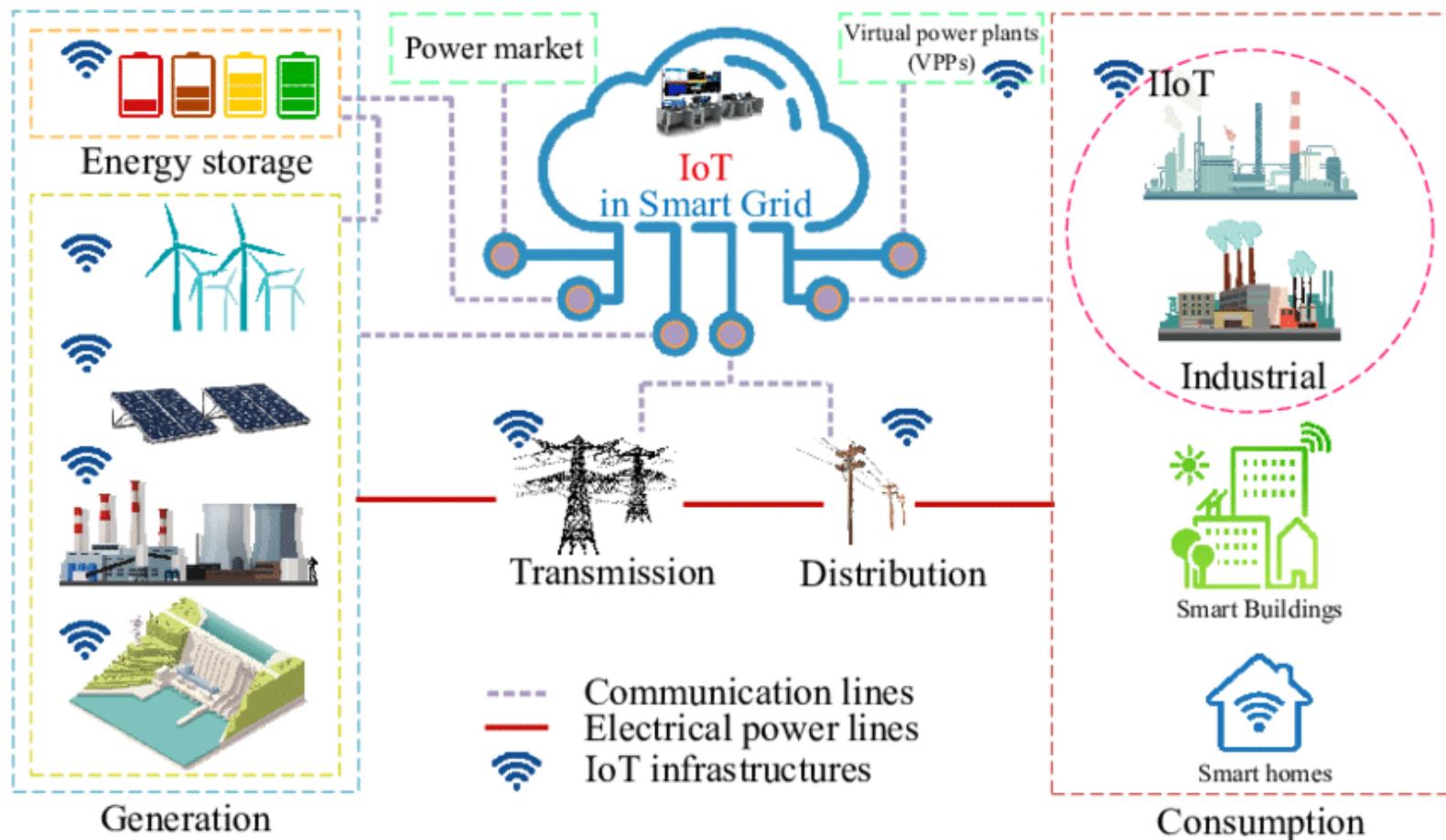
- P3. Smart Water Supply Monitoring:
 - IoT applications in smart water supply/monitoring
- P4. Smart Electrical Power Monitoring:
 - IoT Applications in Smart Power Distribution Monitoring

Smart Utilities (Electrical power, Water)



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

IoT for Smart Grids



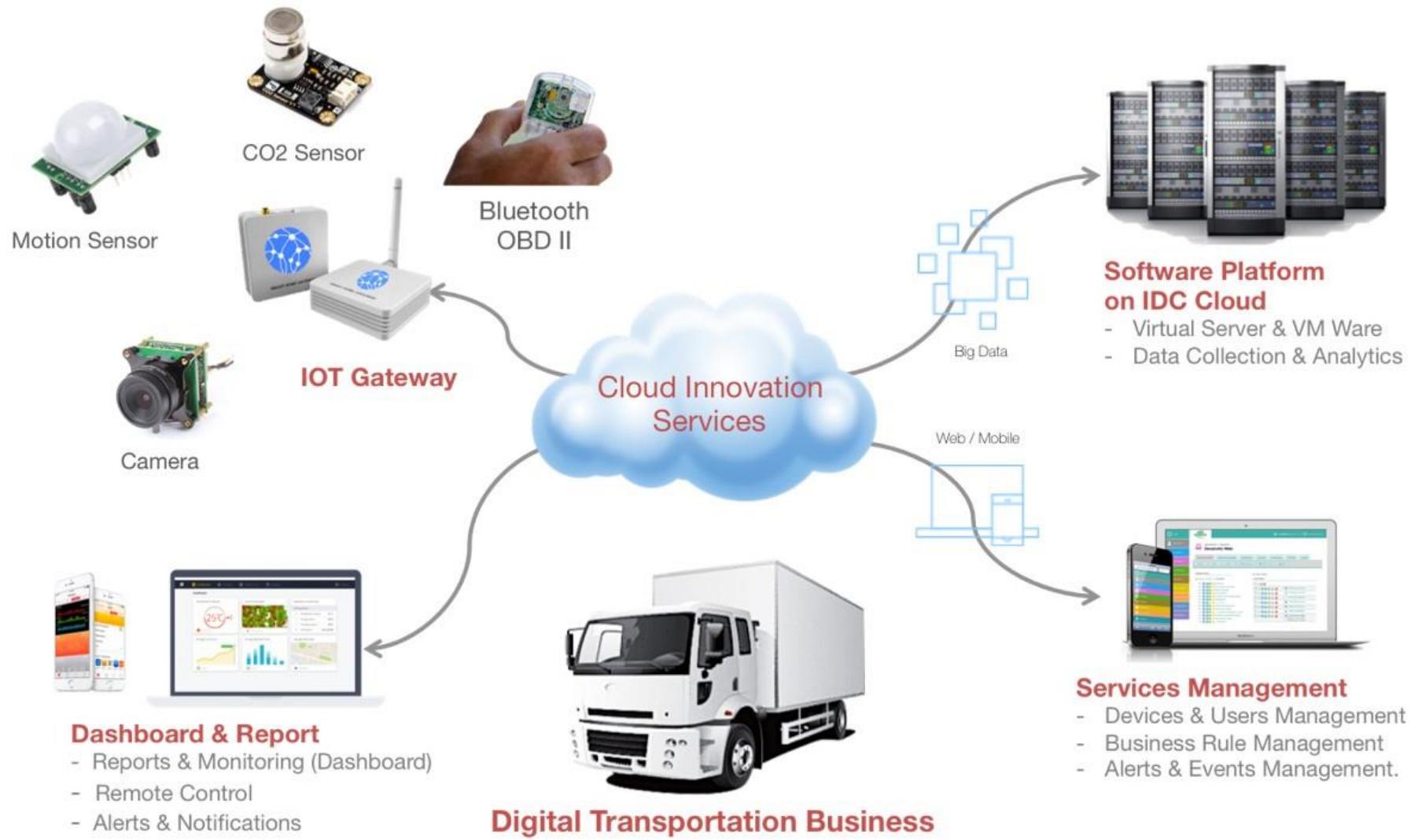
The paradigm of IoT in smart electrical grids

https://www.researchgate.net/publication/331103655_IoT_Architecture_for_Smart_Grids

Topics for IoT applications

- P5. Smart Transportation:
 - IoT Applications in Smart Transportation Monitoring

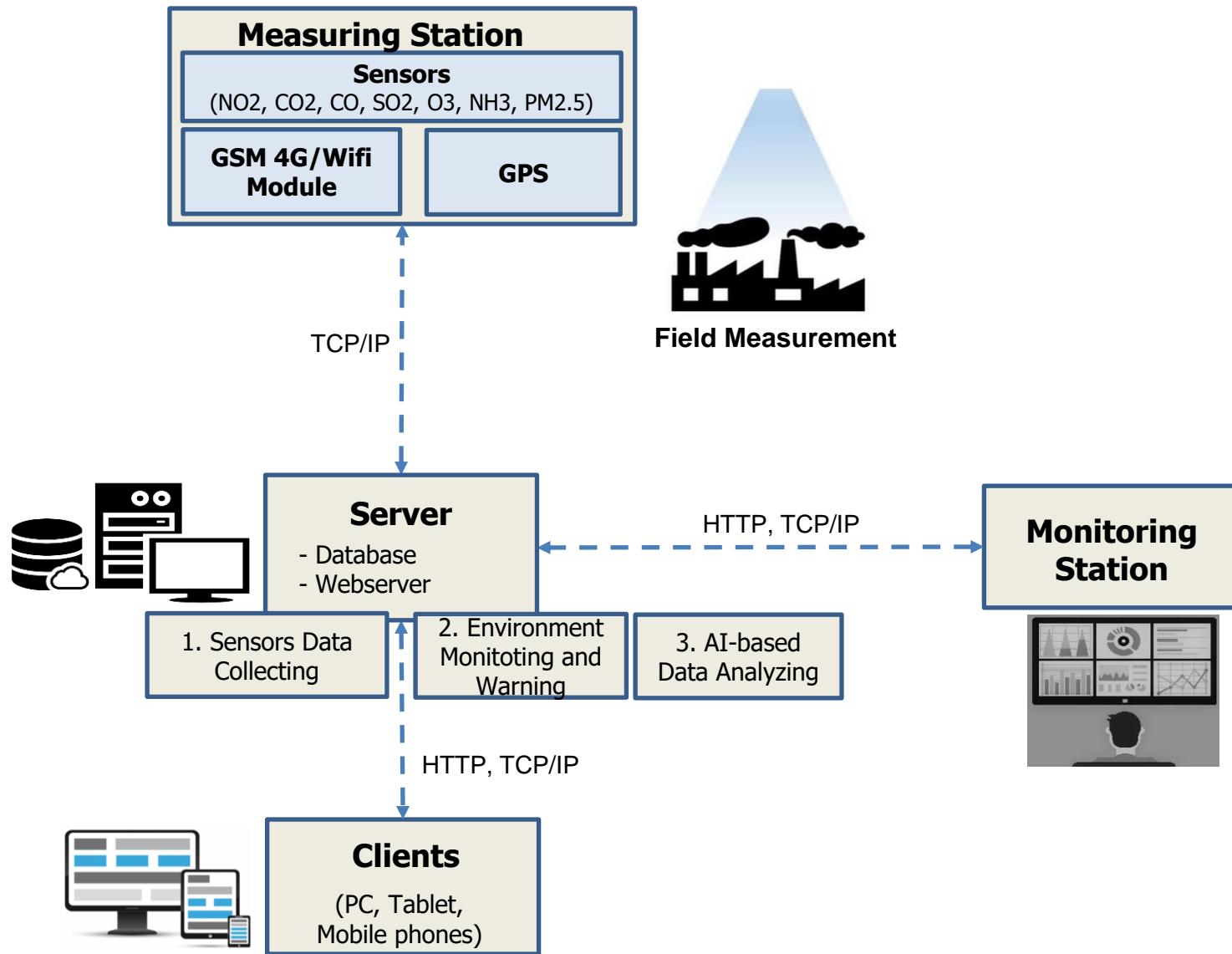
P5. Smart Transportation



Topics for IoT applications

- 6. Smart Air Quality Monitoring:
 - IoT Applications in Air Quality Monitoring

P6. Smart Air Quality Monitoring

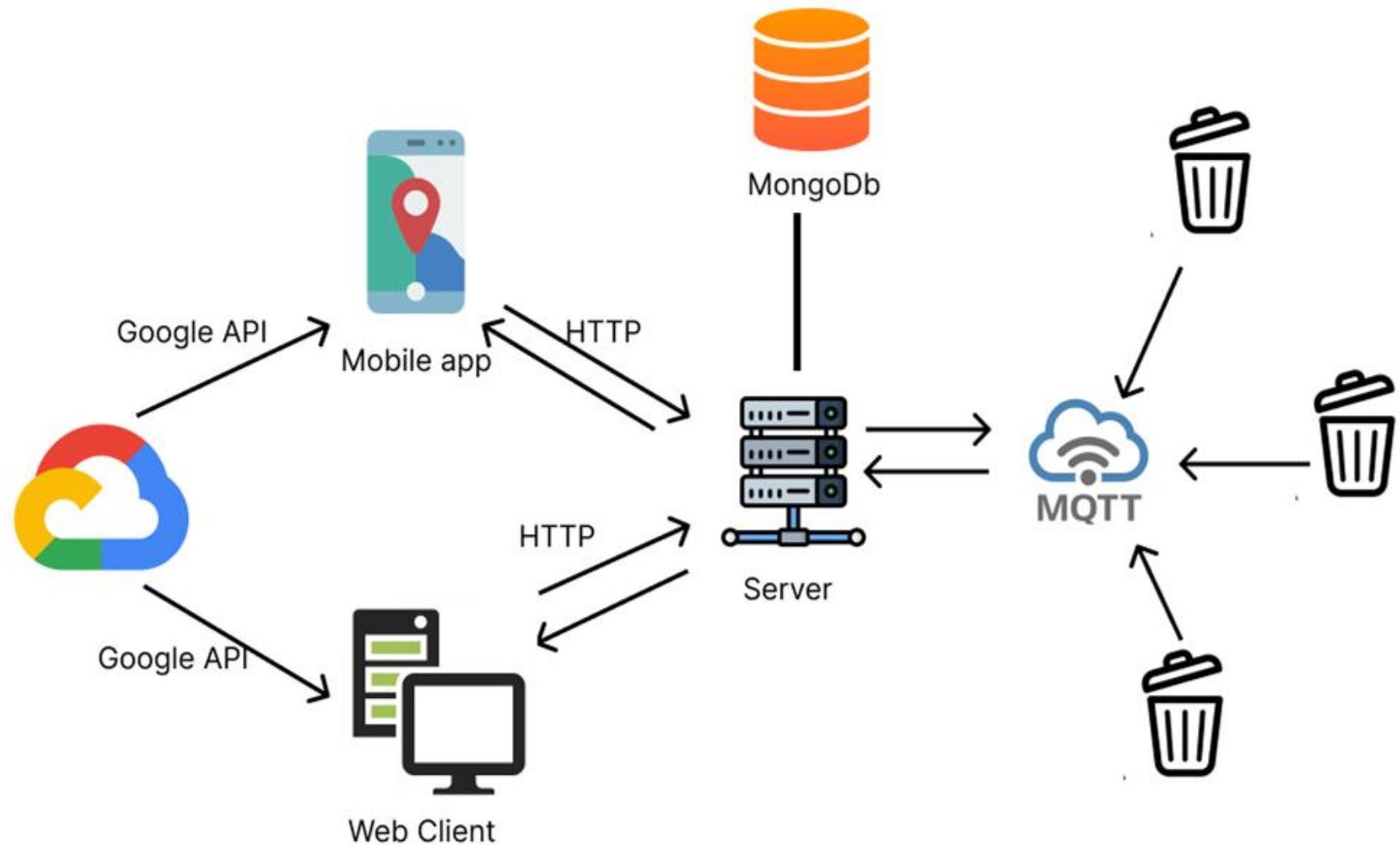


Topics for IoT applications

- 7. Smart Gabaging:
 - IoT Applications in Smart Waste Collection
 - Read: <https://www.instructables.com/id/Smart-Garbage-Monitoring-System-Using-Internet-of-/>
- 8. Smart Parking:
 - IoT Applications in Smart Parking
- 9. Smart Health Monitoring:
 - IoT Applications in Smart Health Monitoring
- 10. Smart Robotic Warehouse:
 - IoT Applications in Smart Warehousing

Project 1: Smart Gabage

- Overview

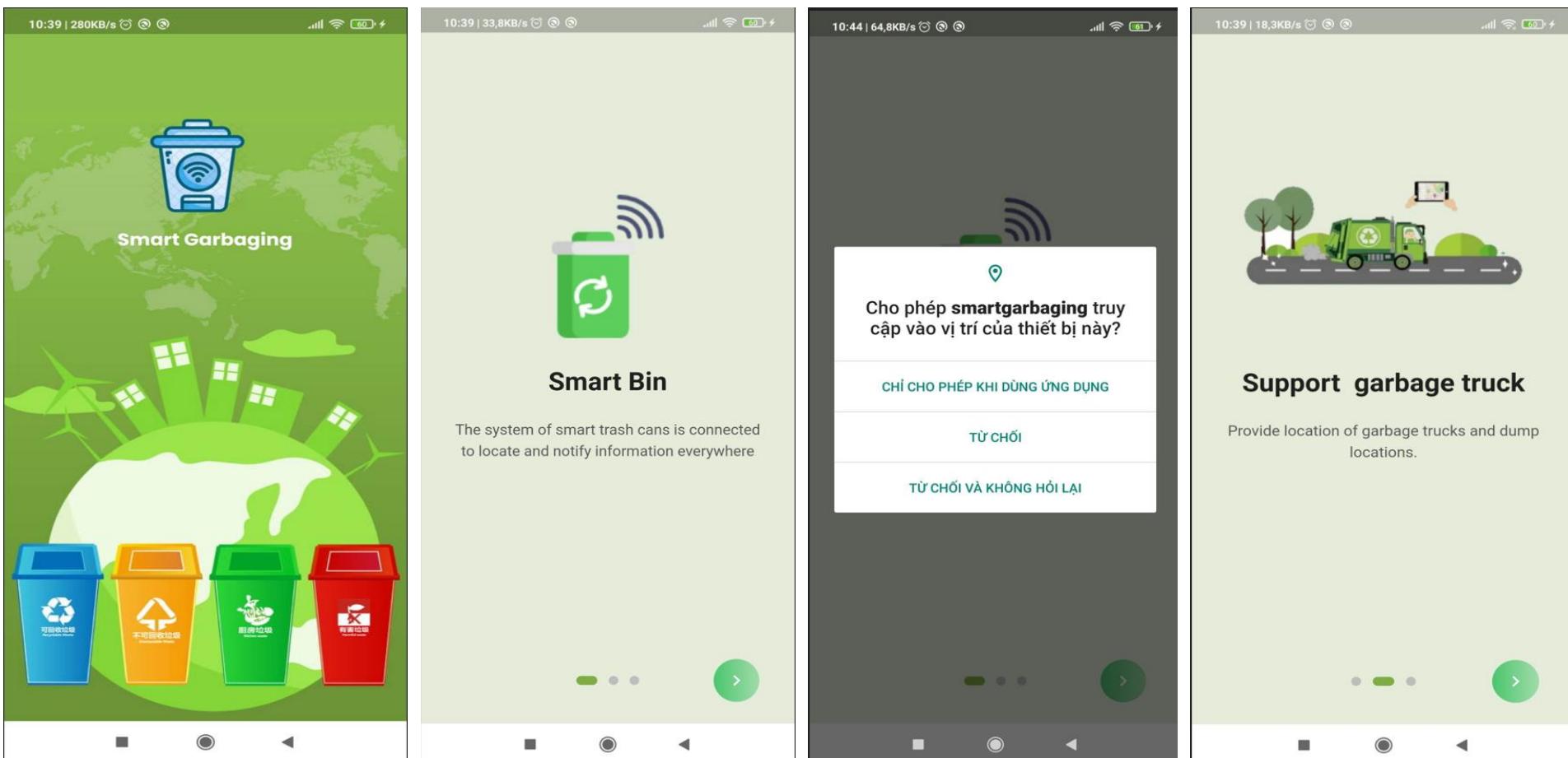


Project 1: Smart Gabage

- Web App for Waste Bin Management: Add/Edit Bins

Project 1: Smart Gabage

- Mobile app



Project 1: Smart Gabage

- Mobile app



Welcome to
Smart Garbagaging!

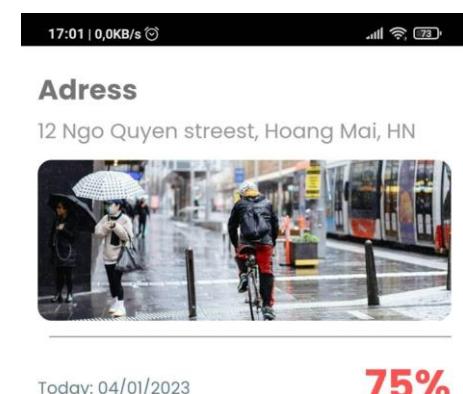
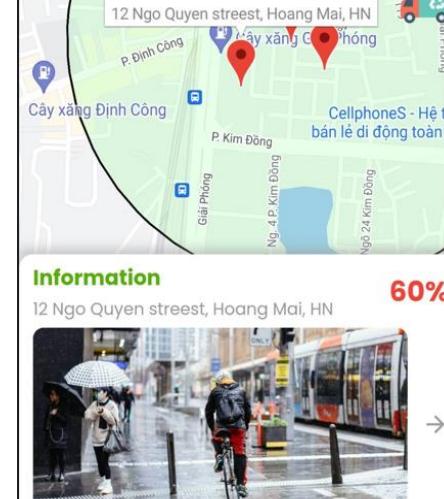
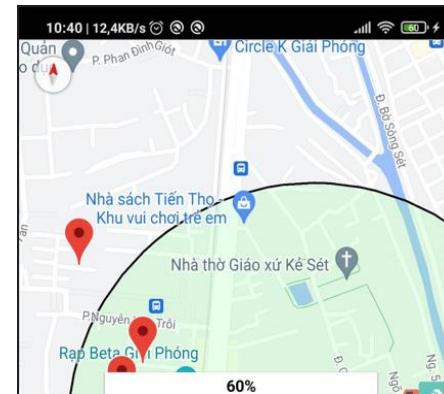
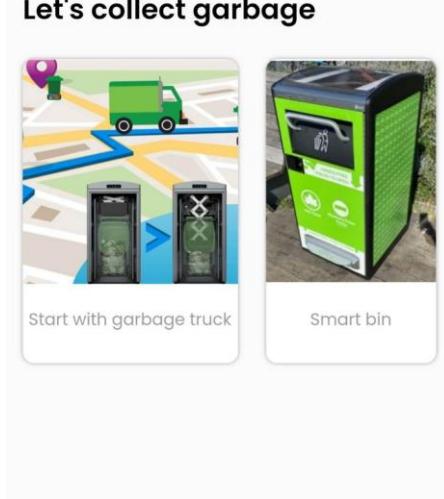
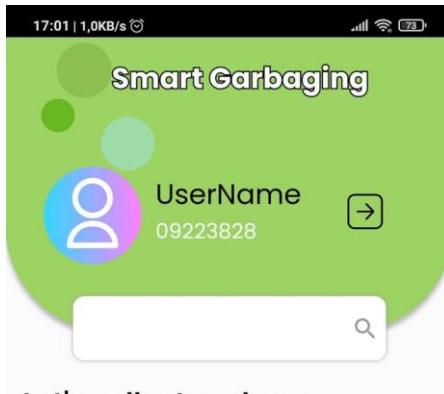
Sign in to your account

Account name

Password

Login

Don't have a account? [Sign Up](#)



Adress

12 Ngo Quyen street, Hoang Mai, HN

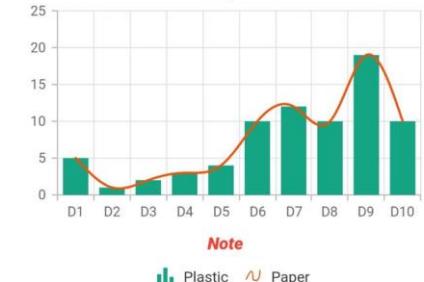


Today: 04/01/2023

75%



Days

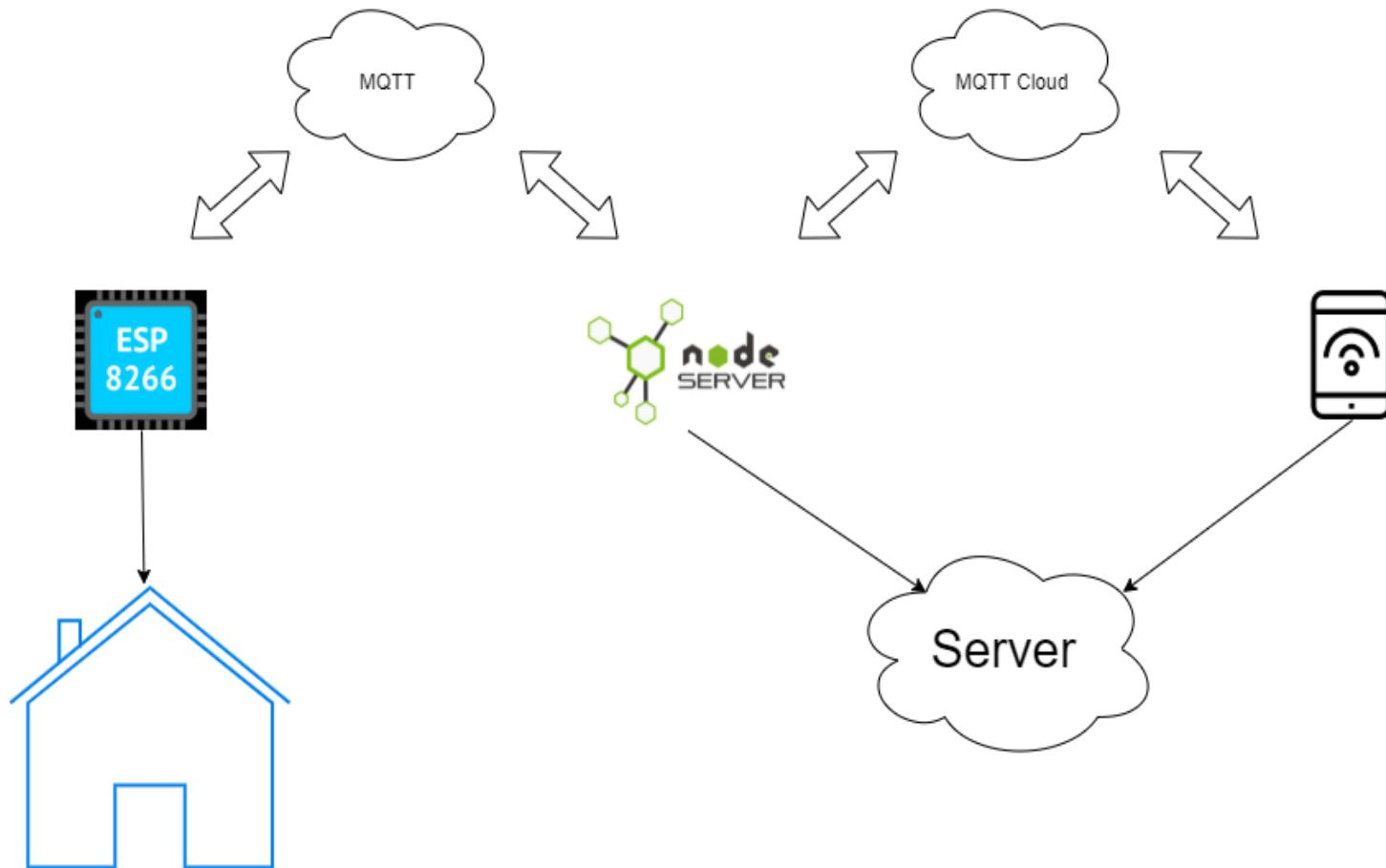


Note

Plastic Paper

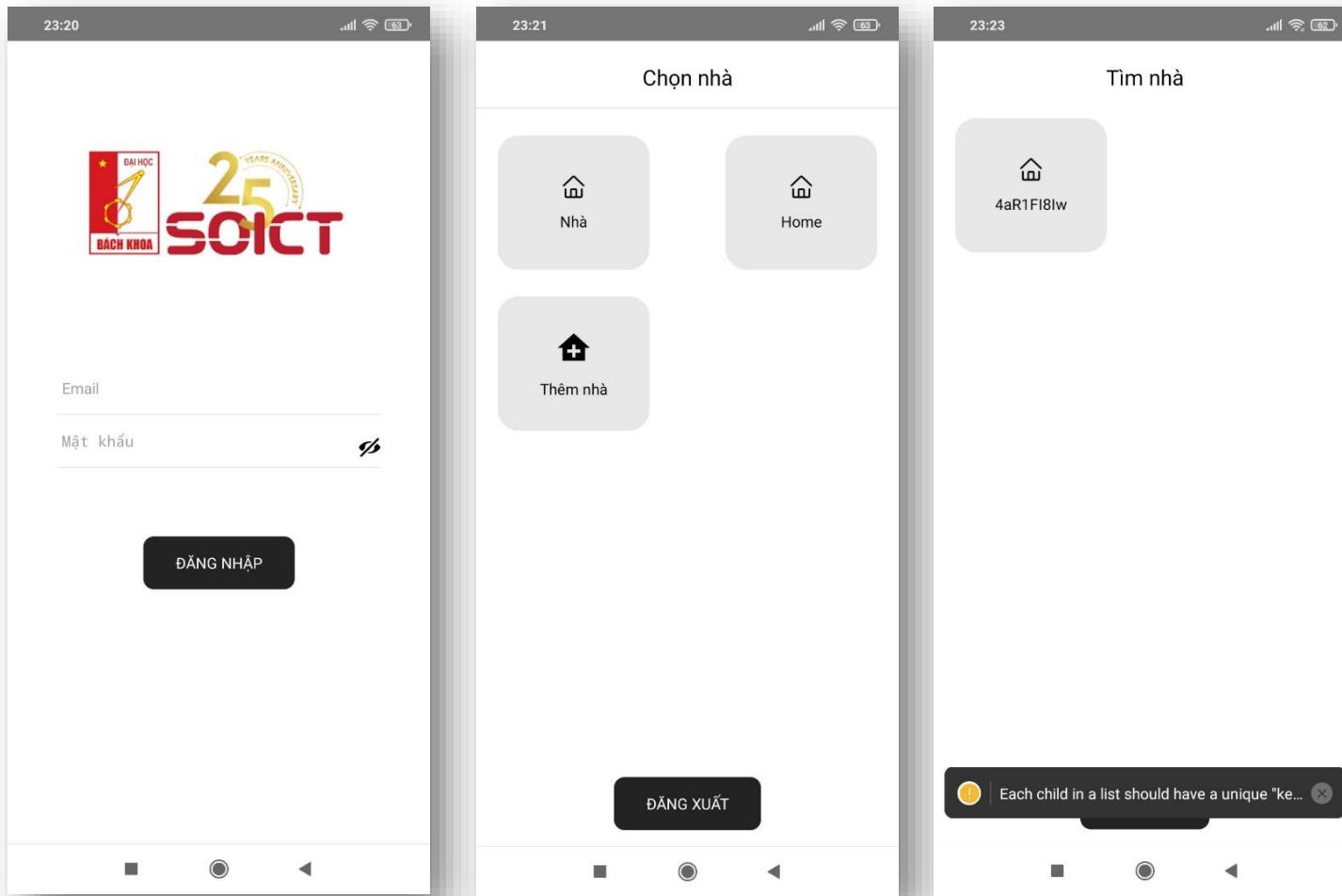
Project 2. Smart Home

- Overview:



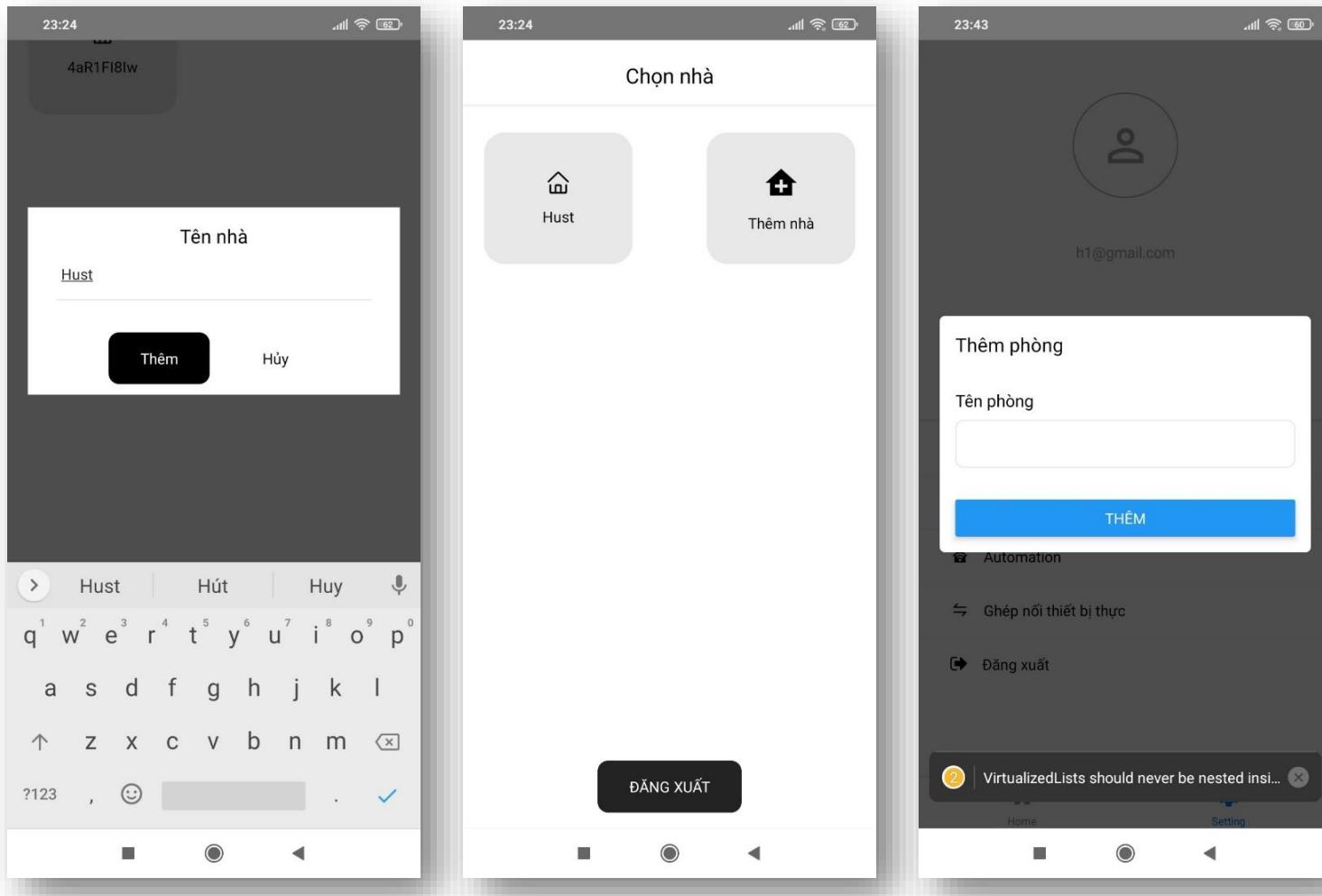
Project 2. Smart home

- Mobile app:
 - Home/Rooms Management (Add/Edit)



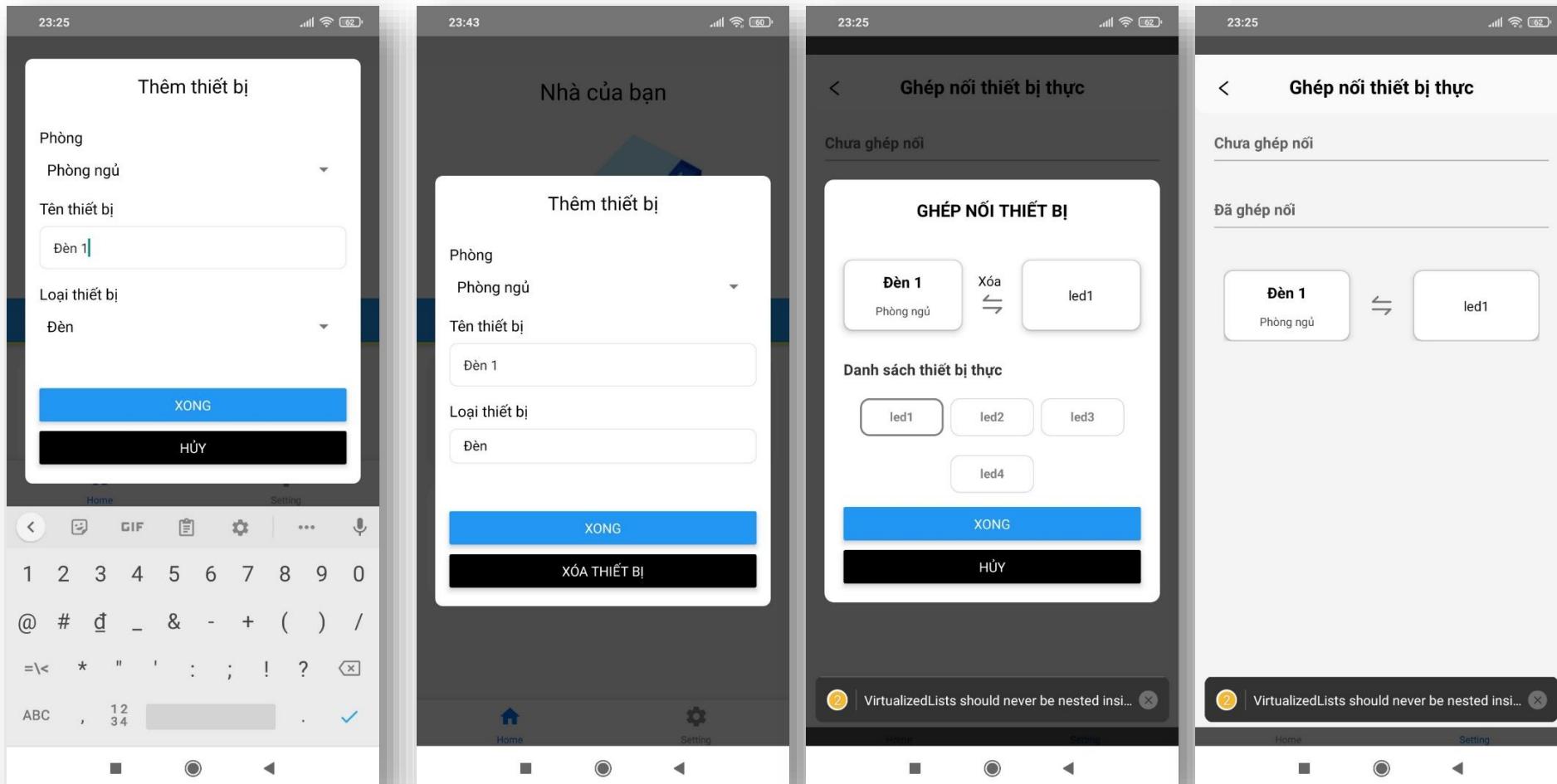
Project 2. Smart home

- Mobile app:
 - Home/Rooms Management (Add/Edit)



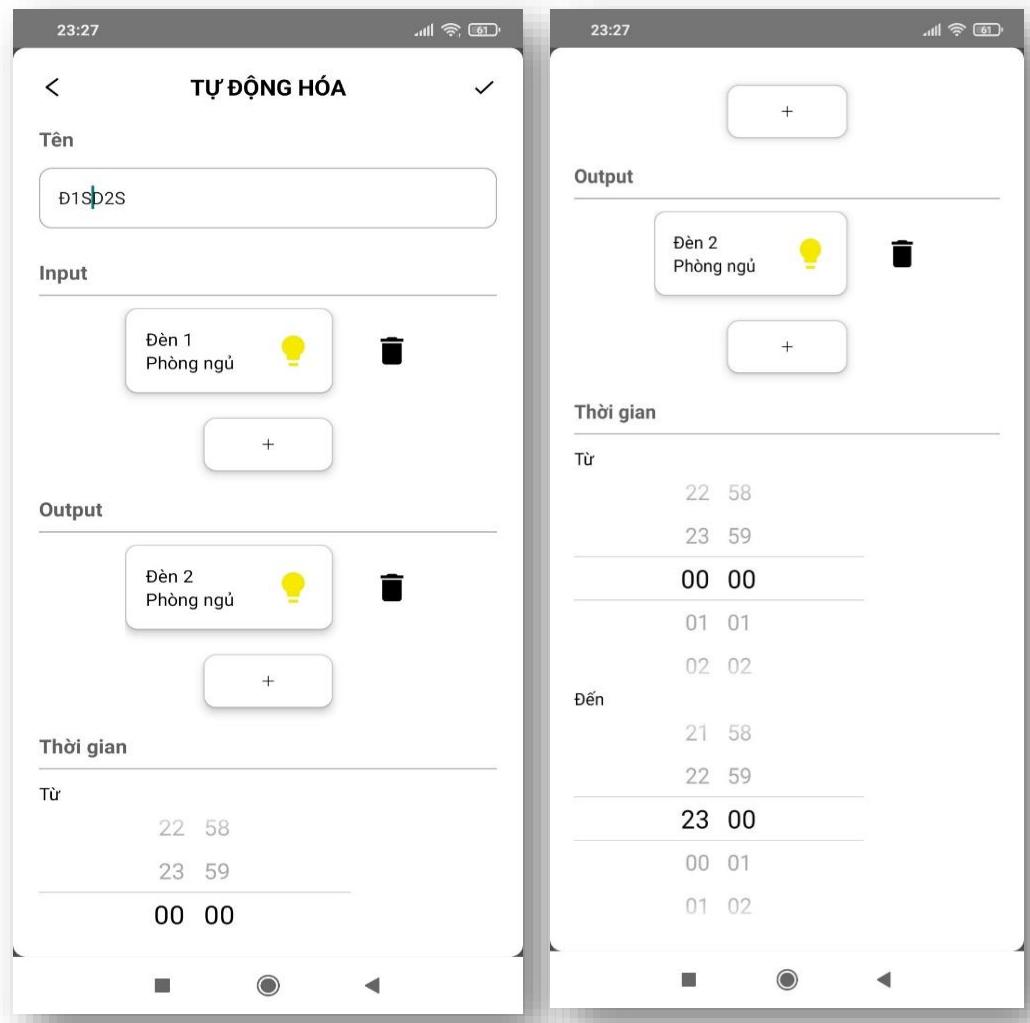
Project 2. Smart home

- Mobile app:
 - Add Device, Connect Physical Devices



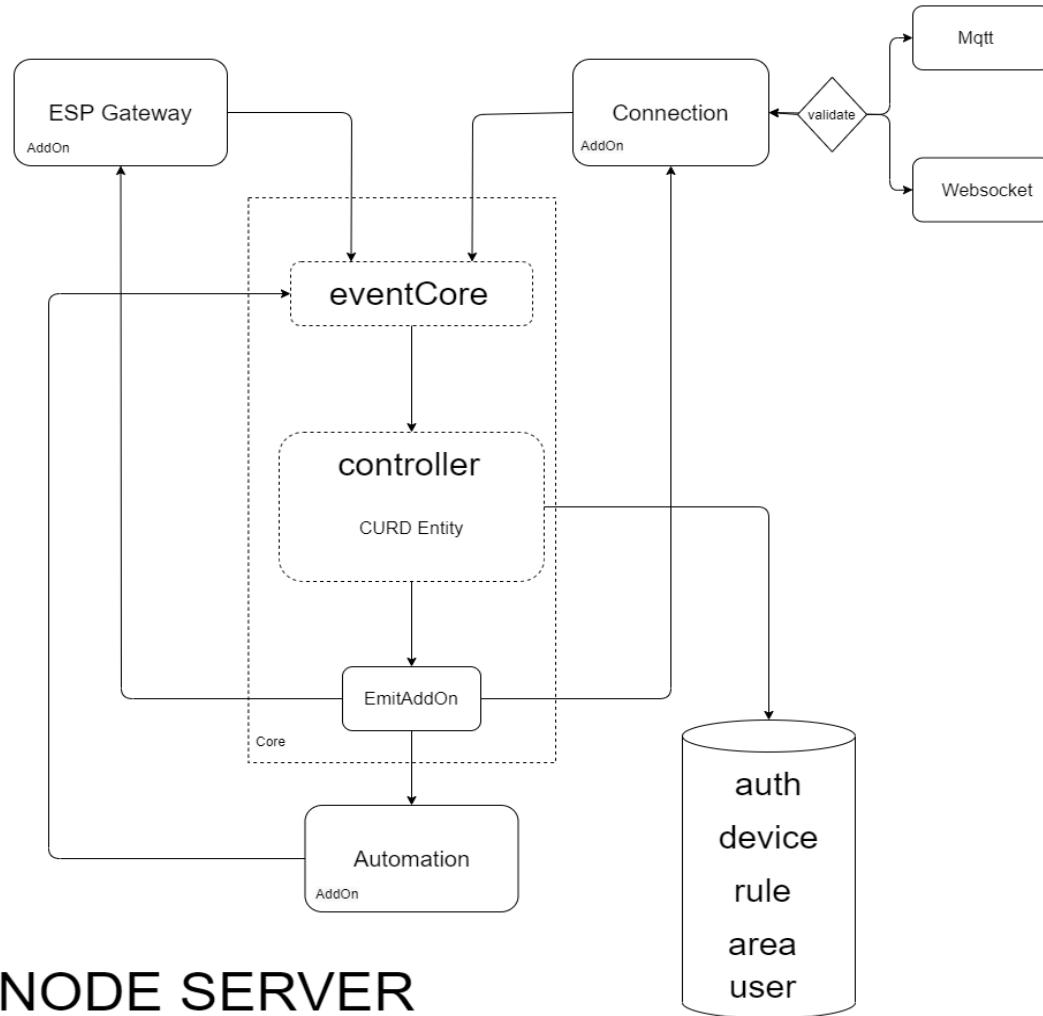
Project 2. Smart home

- Mobile app:
 - Automation



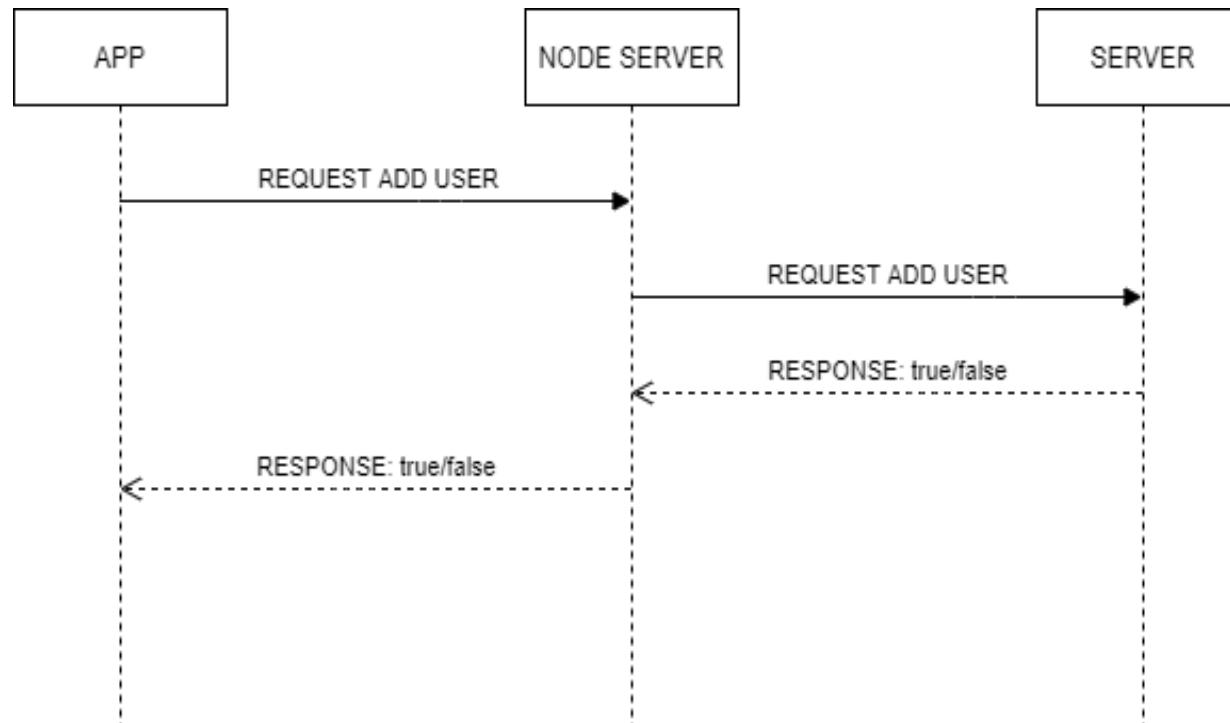
Project 2. Smart home

- Server design:



Project 2. Smart home

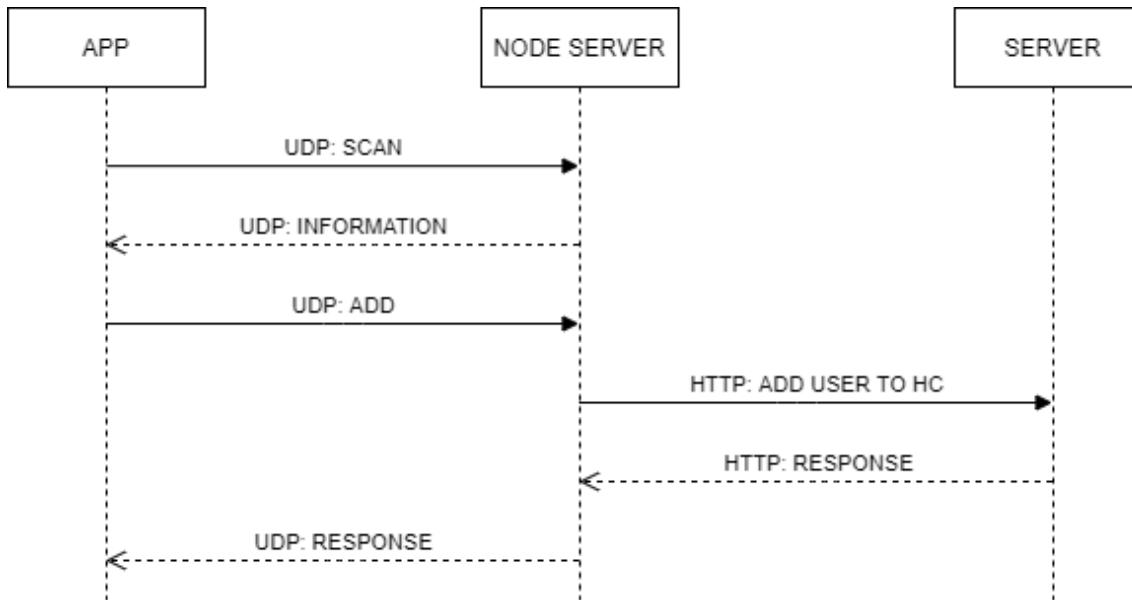
- Server design:
 - User Management Workflow for Adding/Removing Users



Add/Delete Users

Project 2. Smart home

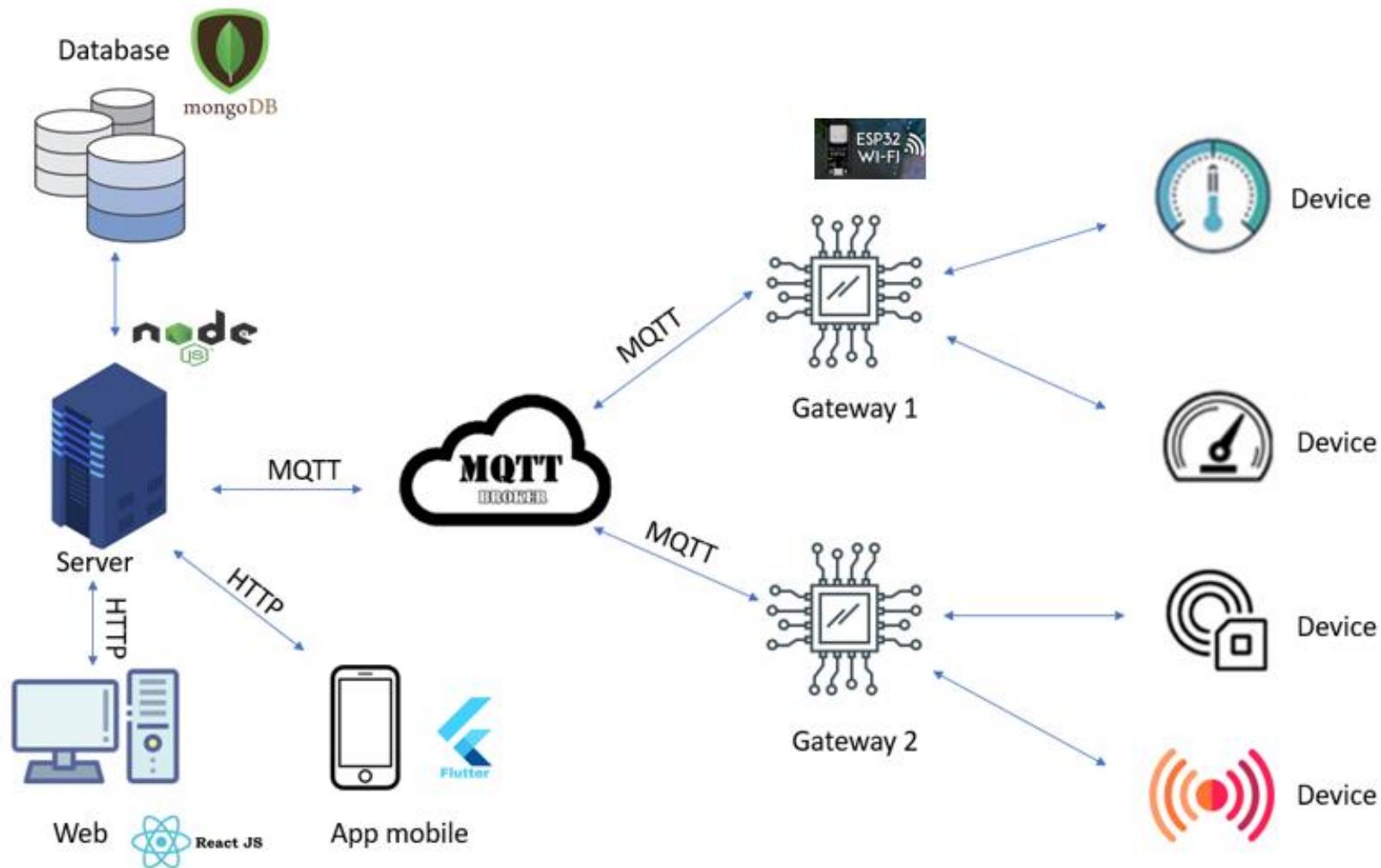
- Server design:
 - Workflow for Adding a House



Add a new house

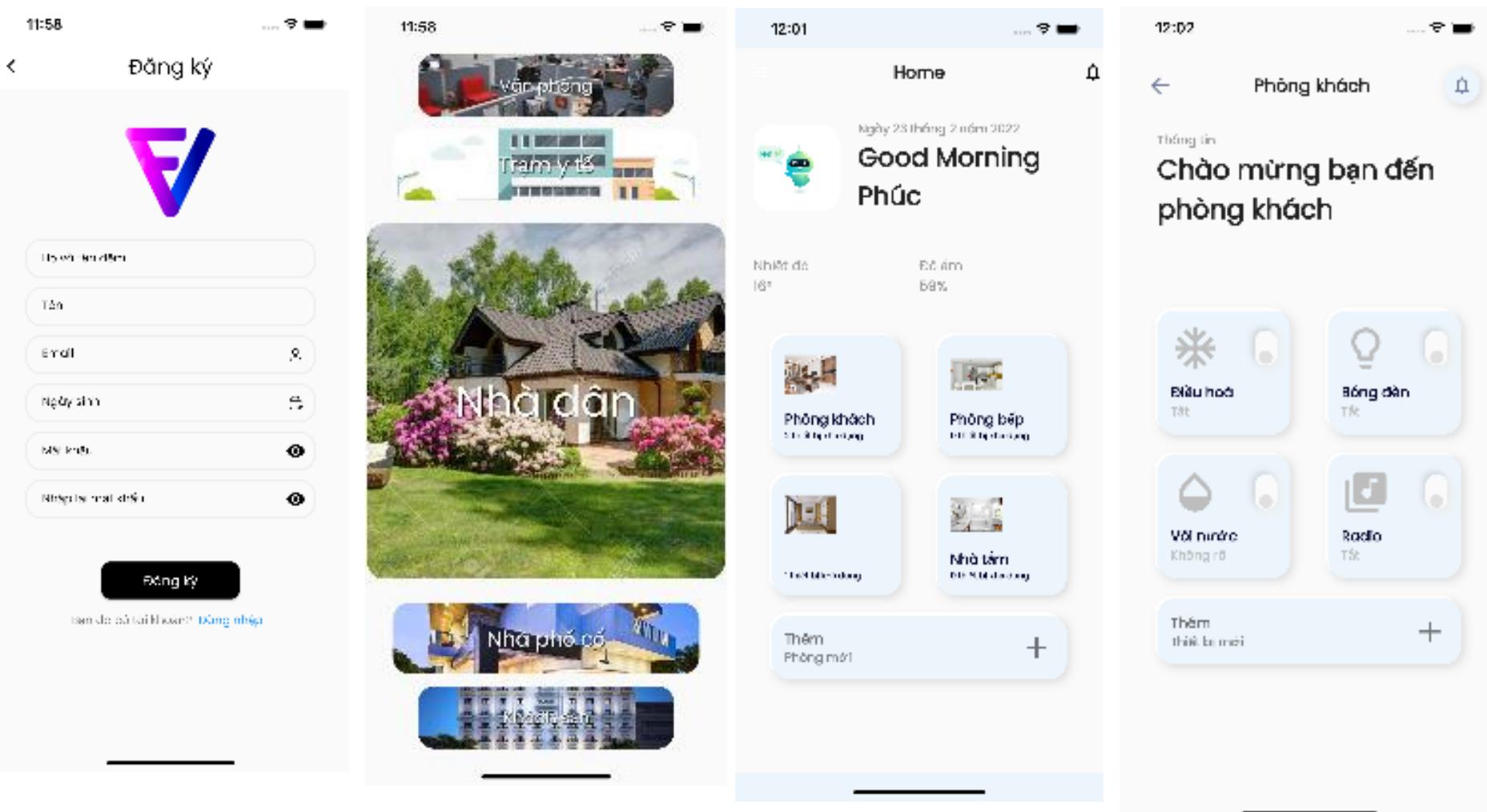
Project 3. Smart home

■ System design



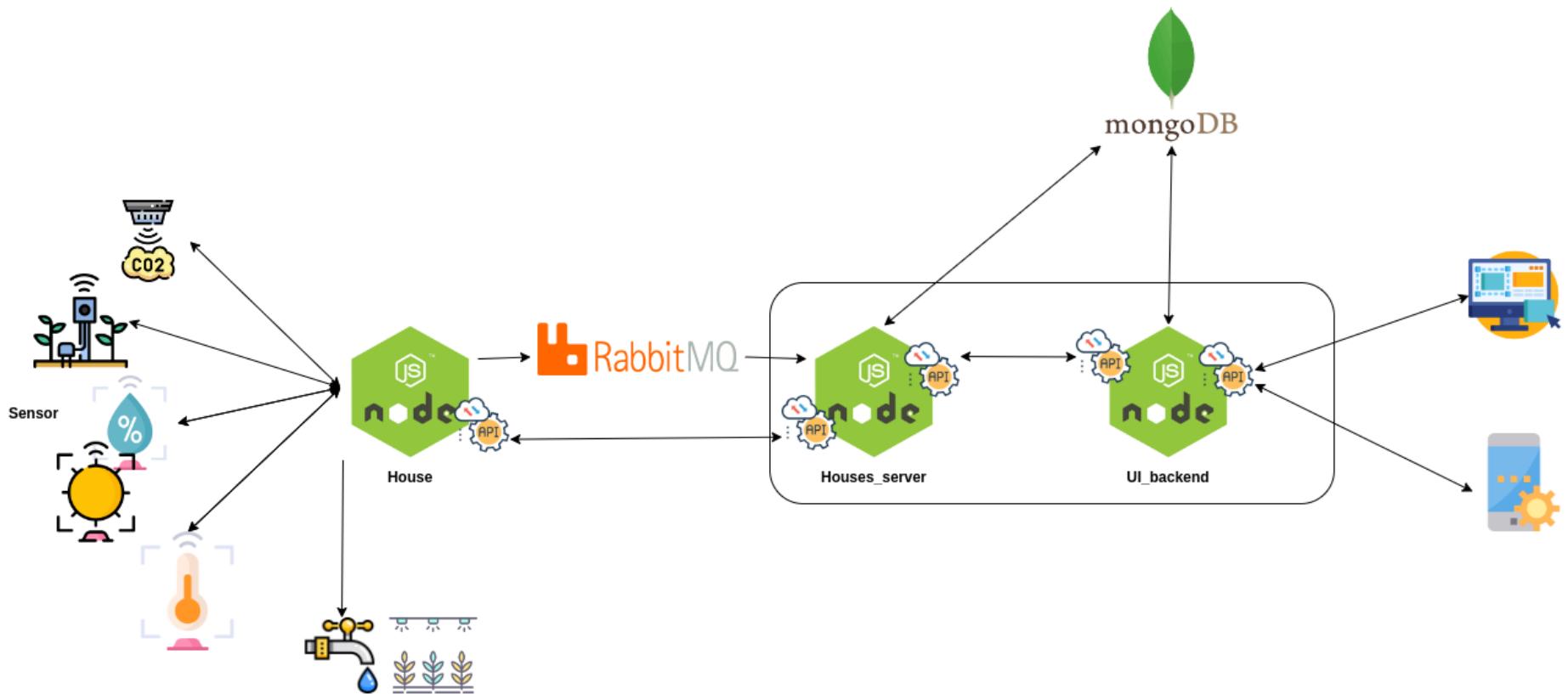
Project 3. Smart home

- Mobile app



Project 4. Smart garden

- System architecture

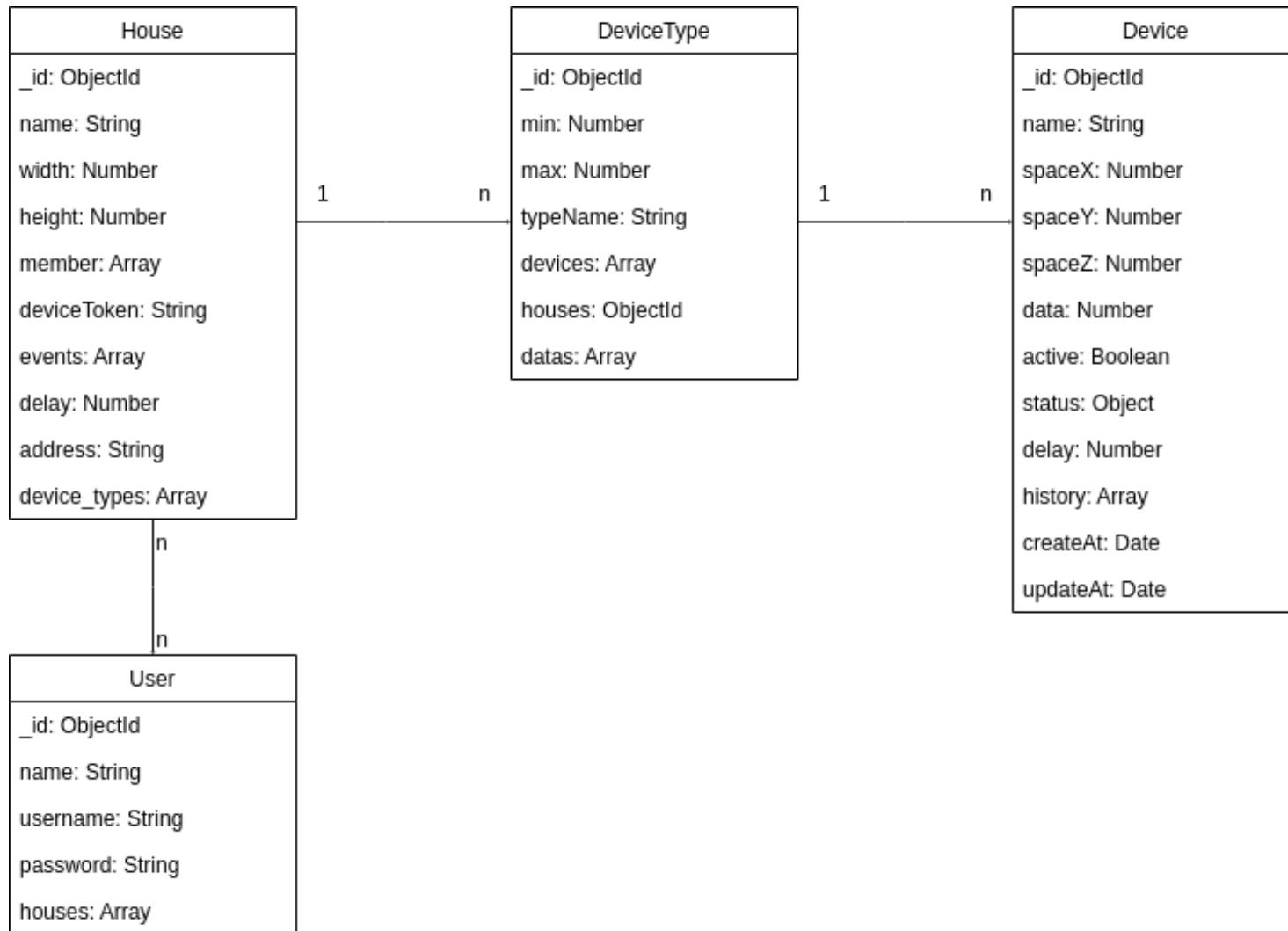


Project 4. Smart garden

- Technologies:
 - Using Synthetic Data for Sensor Devices
 - Server: NodeJS, ExpressJS.
 - Database: MongoDB.
 - Website: React JS.
 - Mobile App: React Native.

Project 4. Smart garden

■ Database design



Project 4. Smart garden

- Building APIs
 - Register: POST /v1/auth/register
 - Login: POST /v1/auth/login
 - Get list of devices for a specific house: GET /v1/device/{houseId}
 - Update device information: PUT /v1/device/{deviceID}
 - Get statistical data: GET /v1/data/{houseId}
 - Get list of warning/events for a specific house: GET /v1/house/events/{houseId}
 - Get information for a house: GET /v1/house
 - Add new house: POST /v1/house
 - Update house's information: PUT /v1/house/update-house/{houseId}
 - Add new member: PUT /v1/house/update-member/{houseId}
 - Delete a member: PUT /v1/house/delete-member/{houseId}/{userId}

Project 4. Smart garden

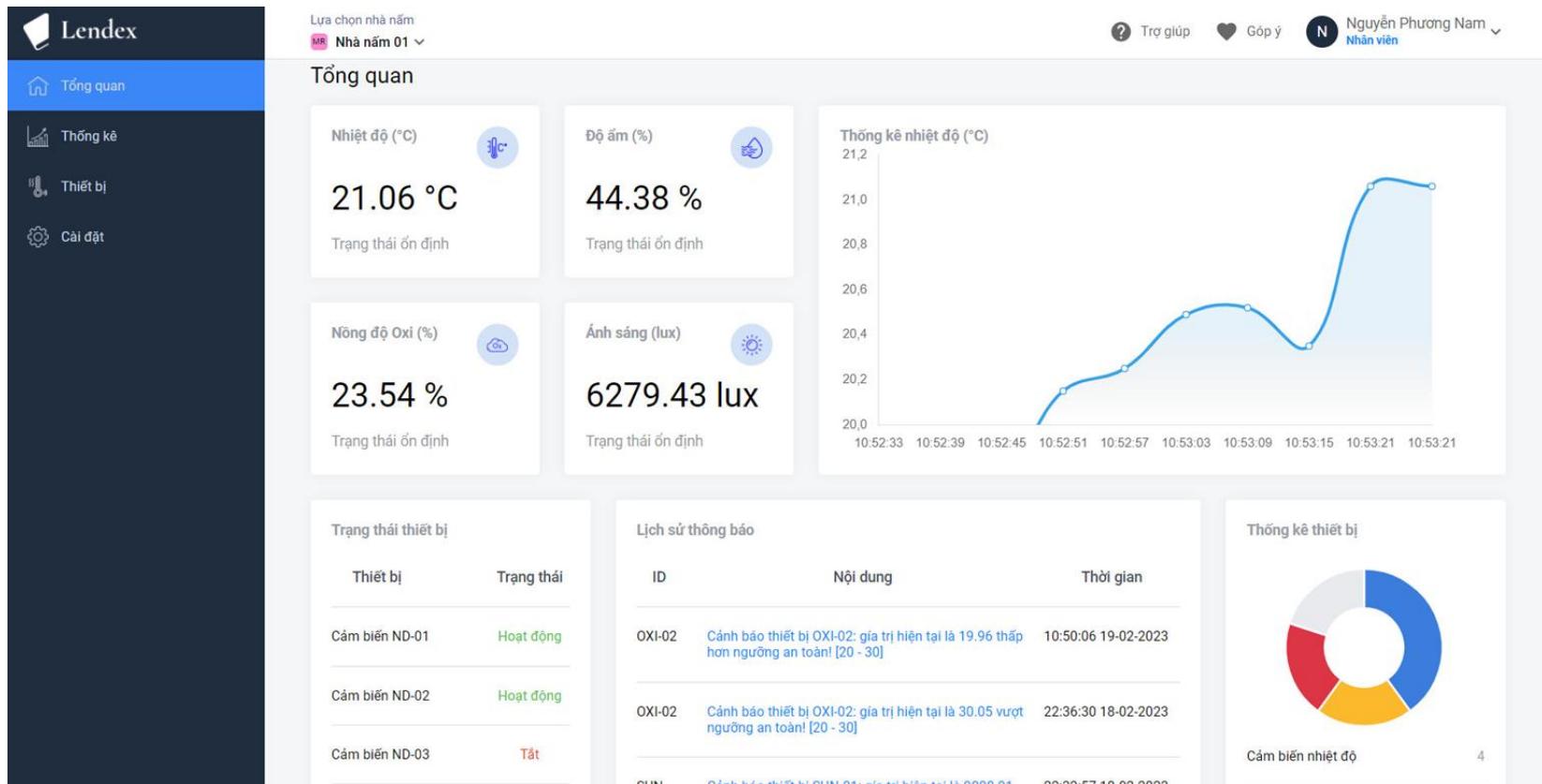
- Example API for Get list of devices for a specific house

The screenshot shows a Postman API request for a device list. The URL is `http://localhost:9000/v1/device/63e8a7ab1d624a76b5d91255`. The response status is 200 OK, with a time of 233 ms and a size of 81.27 KB. The response body is a JSON object containing a success boolean and a list of devices. Each device has properties like _id, name, active, spaceX, spaceY, spaceZ, data, created_at, __v, delay, status, updatedAt, and history. The first device is named ND-01 and the second is ND-02.

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

Project 4. Smart garden

- Admin dashboard Web app



Project 4. Smart garden

- Admin dashboard Web app:



Project 4. Smart garden

- Admin dashboard Web app:

The screenshot shows the Lendex Admin Dashboard interface. The left sidebar has a dark theme with blue highlights for 'Thiết bị' and 'Cài đặt'. The main content area is titled 'Cài đặt' (Setup). It includes sections for 'Thông tin nhà nấm' (Garden information) and 'Tạo mới nhà nấm' (Create new garden). Under 'Thông tin nhà nấm', there are fields for 'Tên nhà nấm' (Name: Nhà nấm 01), 'Chiều dài nhà nấm' (Length: 250), and 'Chiều rộng nhà nấm' (Width: 200). To the right, there is a 'Thành viên nhà nấm' (Garden members) section with a table listing three members: Nguyễn Phương Nam (Quản trị viên), Nguyễn Mạnh Duy, and Trần Đức Hải.

Thành viên	Tên đăng nhập	Chức vụ
Nguyễn Phương Nam	Nguyễn Phương Nam	Quản trị viên
Nguyễn Mạnh Duy	Nguyễn Mạnh Duy	
Trần Đức Hải	Trần Đức Hải	

Project 4. Smart garden

■ Mobile app

