FPT SOFTWARE
IoT CHALLENGE

TOPIC: SMART HOME
Group: ONE PIECE

Mentor: Dang Anh Phuc

Students:

1. Pham Trung Hieu
2. Nguyen Thi Kim Anh
3. Pham Thu Hoai
4. Nguyen Si Tu

Ha Noi, 2023

Table of Content:

1. Introduction

Providing a general overview of the accomplished content:

- General introduction to the Smart Home model
- Communication between sensors and ESP8266 and ESP32
- Pushing the collected data to Firebase
- Creating a React Native app to control devices within the Smart Home
- Facial recognition for door unlocking
- Implementing ESP NOW communication between 4 ESP devices
- Over-the-Air (OTA) firmware updates

2. Content

2.1. Overview of the Smart Home model

2.1.1. General overview of the topic

Smart Home is an intelligent automation system within a house that allows control and management of various devices and systems such as lighting, air conditioning, doors, security systems, audio, video, and other household appliances. This system is controlled through smart mobile devices or voice commands, helping users save time, increase convenience, and enhance safety in their daily lives.

2.1.2. Project objectives

- Create an illustrative model of a smart home.
- Control and display devices through a mobile app.
- Implement facial recognition for door access.
- Establish connectivity between the app, cloud, and physical devices.

2.1.3. System Construction

The system structure consists of three components: Cloud, App and Device, as depicted in the diagram below:

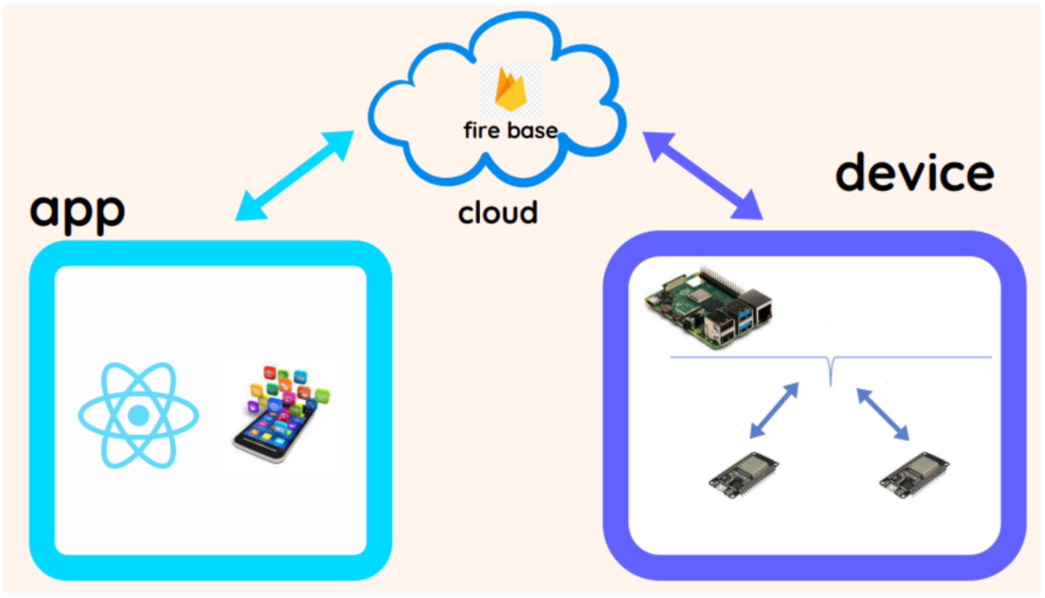


Figure 1: System structure

App: React Native is a framework that allows the development of mobile applications for iOS and Android platforms using JavaScript and React. It was created by Facebook and is currently one of the most popular mobile app development technologies in the market.

React Native enables code reusability across platforms and facilitates rapid development of mobile applications like native apps.

Firebase Cloud: Firebase is a cloud-based database service coupled with Google's powerful server infrastructure. Its main function is to simplify database operations for app developers. It provides simple application programming interfaces (APIs) for both Android and iOS platforms. Firebase Realtime Database is a cloud-based NoSQL database provided by Firebase. It is a real-time solution for web and mobile applications. The Firebase Realtime Database stores data in JSON format and allows real-time data synchronization across multiple devices. This enables applications to access data quickly and efficiently.

ESP Wi-Fi Module: The ESP Wi-Fi module is used in applications that require connectivity, data collection, and control over Wi-Fi networks, particularly in IoT-related applications.

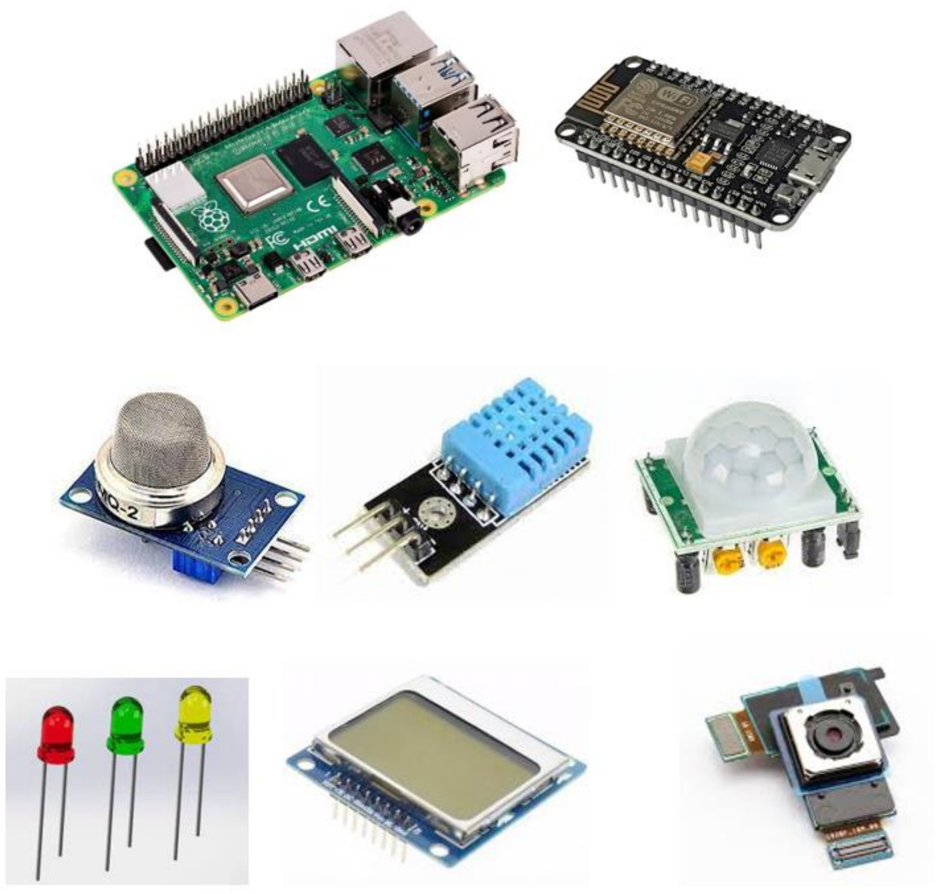


Figure 2: Some devices used in the project

2.2. Communication between sensors and ESP8266 and ESP32

2.2.1. Communication of DHT11 temperature and humidity sensor with ESP8266 and data pushing to Firebase

DHT11 is an inexpensive digital sensor used for temperature and humidity sensing. This sensor can easily communicate with various microcontrollers such as Arduino, Raspberry Pi, etc., to instantly measure humidity and temperature.

DHT11 is a relative humidity sensor. To measure the surrounding air, it utilizes a thermistor and a capacitive humidity sensor.

Circuit schematic:

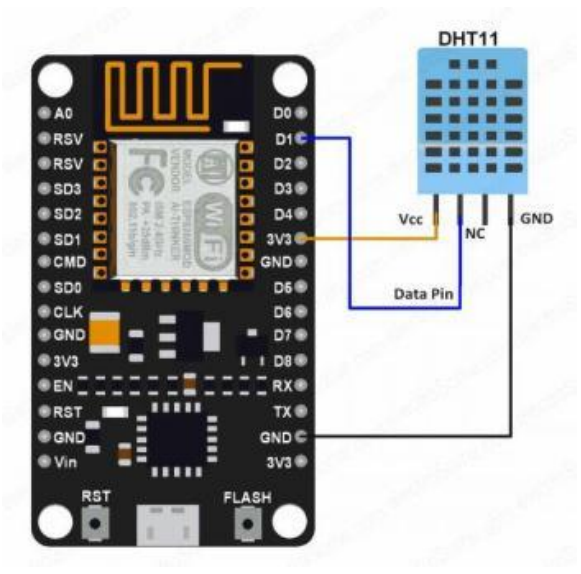


Figure 3: ESP8266 Pin Connection Diagram with DHT11

2.2.2. Communication of MQ2 gas sensor with ESP8266 and data pushing to Firebase

The MQ2 Gas Sensor module is highly useful for detecting gas leaks in both residential and industrial settings. It is suitable for detecting H2, LPG, CH4, CO, Alcohol, Smoke, or Propane. Due to its high sensitivity and fast response time, measurements can be taken promptly. The sensitivity of the sensor can be adjusted using a potentiometer. The sensor is actually enclosed within two layers of fine stainless steel mesh called the Explosion-Proof Mesh. It ensures that the heating element inside the sensor will not cause an explosion as we detect potentially combustible gases.

Circuit schematic:

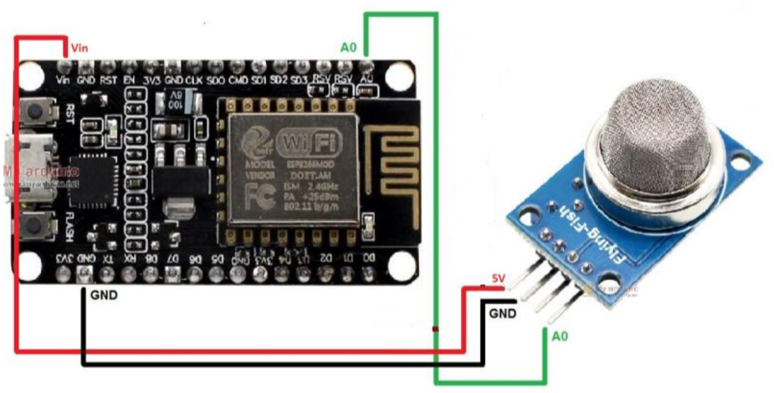


Figure 4: ESP8266 Pin Connection Diagram with MQ2

2.2.3. Communication of PIR (HC-SR501) motion detection sensor with ESP8266 and data pushing to Firebase

The HC-SR501 Passive Infrared (PIR) motion detection sensor is used to detect the motion of objects emitting infrared radiation (humans, animals, heat-emitting objects, etc.). The sensor can be adjusted for sensitivity to limit the detection range as well as the intensity of radiation from the desired object. Additionally, the sensor can adjust the trigger delay time (how long the signal remains active after activation) using the built-in potentiometer.

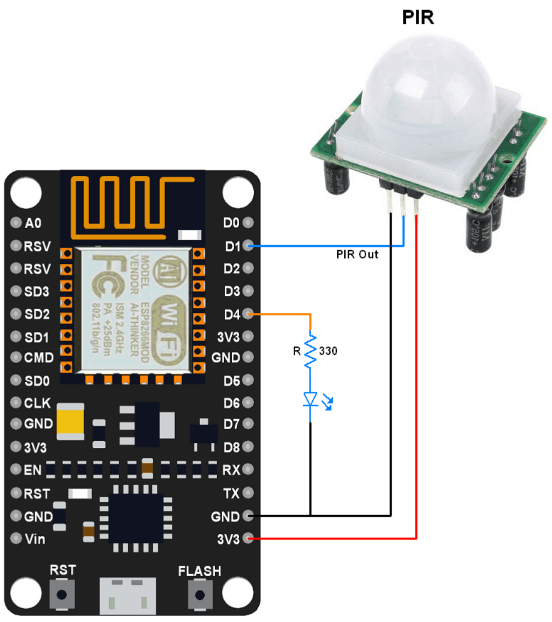


Figure 5: ESP8266 Pin Connection Diagram with SR501

2.2.4. Controlling an LED with PWM using values fetched from Firebase

Pulse Width Modulation (PWM) is a method of adjusting the output voltage or, in other words, a modulation method based on the variation of the width of a square wave, resulting in a change in the output voltage. PWM signals have the same frequency but differ in the width of the positive or negative edge.

PWM is widely used in control applications such as motor speed control, voltage regulation, etc. It is used to control the speed of a motor or, furthermore, to maintain the speed stability of a motor by adjusting the PWM duty cycle.

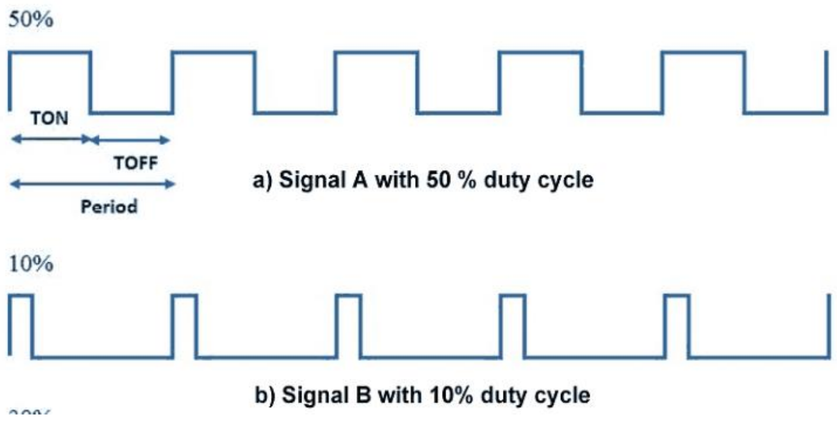

50%
TON
TOFF
Period
a) Signal A with 50 % duty cycle
10%
b) Signal B with 10% duty cycle

Figure 6: PWM Cycle

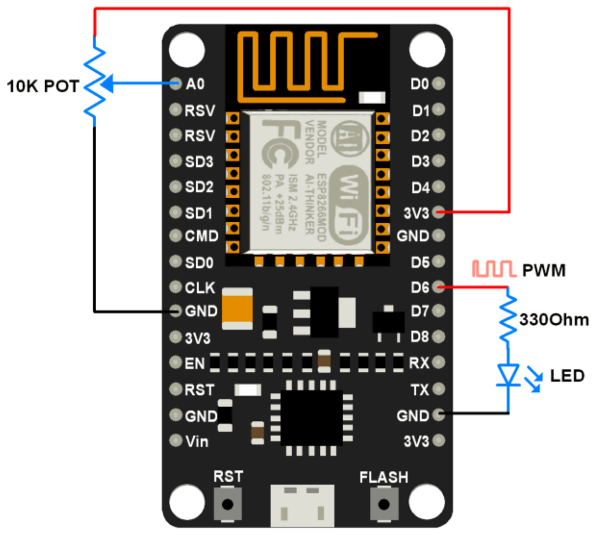
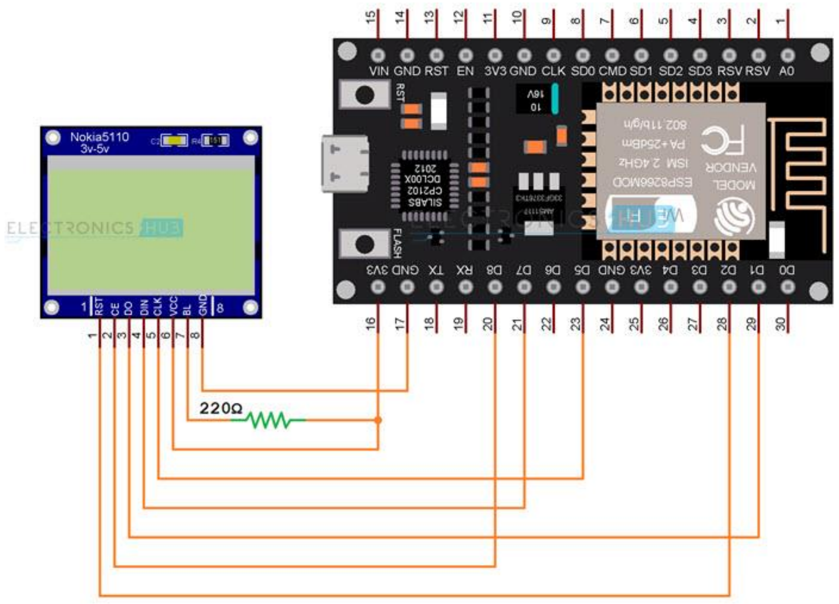


Figure 7: Pin Connection Diagram

2.2.5. Displaying text on the Nokia 5110 LCD

This graphic LCD screen is a graphical display that has been used in various applications. Initially, it was designed for the Nokia 5510 mobile phone display, which was primarily used for iconic displays. However, now we can easily utilize it for different purposes such as displaying alphanumeric characters and digits, drawing lines, shapes, and even displaying bitmap images because its monochrome 84x48 pixels translate to 84 columns and 48 rows. All the necessary functions of this display are integrated into a single small chip that operates at very low voltage. It is cost-effective, more precise, more reliable, and easier to use than traditional LCD displays.

The pin connection diagram is shown below:



VIN GND RST EN 3V3 GND CLK SD0 CMD SD1 SD2 SD3 RSV RSV A0
3V3 GND TX RX D8 D7 D6 D5 GND 3V3 D4 D3 D2 D1 D0
Nokia5110
3v-5v
RST CE DC DIN CLK VCC BL GND
220Ω
ELECTRONICS HUB

Figure 8: ESP8266 Pin Connection Diagram with Nokia 5110 LCD

2.2.6. Controlling a Servo with ESP32

A servo motor is a low-speed, high-torque motor with various shapes and sizes. Unlike DC motors and stepper motors, servo motors typically do not rotate a full 360 degrees. Instead, they are limited to a range of 180, 270, or 90 degrees.

A servo motor is a combination of a DC motor, position control system, and gears. The position of the motor shaft is adjusted by the electronic control devices inside the servo, based on the duty ratio of the PWM signal at the SIGNAL pin.

In a typical analog servo motor, a 20 ms PWM signal is used to control the motor. The 20 ms signal has a frequency of 50 Hz.

The width of the pulse varies between 1 and 2 ms to control the position of the motor shaft.

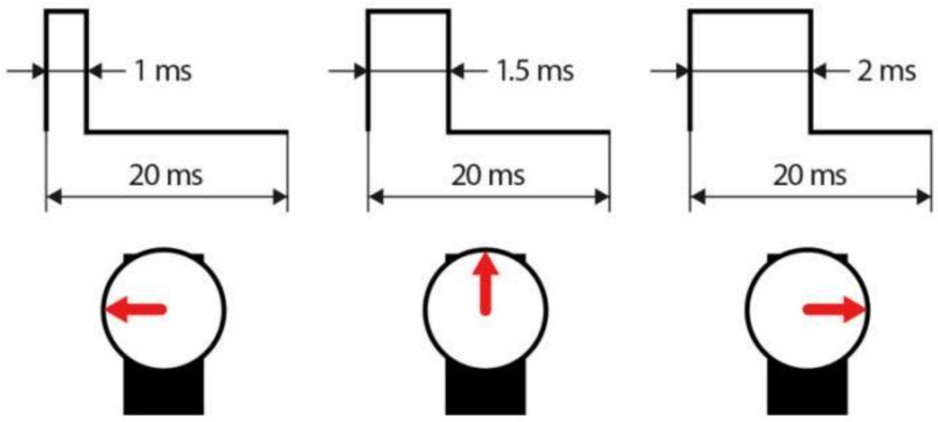


Figure 9: Pulse Width Control for Motor Position

2.3. Creating a React Native App to Control Devices in a Smart Home

2.3.1. Introduction to React Native

React Native is an open-source framework released by Facebook that allows developers to build JavaScript-based applications. It is one of the frameworks that follows a design pattern similar to React. Most of its interfaces and functionalities are composed of numerous smaller components. React Native is used to develop applications for various platforms such as Android, iOS, Web, and UWP.

Some popular applications written in React Native include Instagram, Facebook, Skype, SoundCloud Pulse, Airbnb, and more.

2.3.2. How React Native Works

React Native primarily operates by integrating two threads: the Main Thread and the JS Thread for mobile applications.

The Main Thread is responsible for updating the user interfaces (UI) and handling user interactions.

The JS Thread executes and processes JavaScript code.

These two threads are completely separate in React. To allow interaction between the threads, a Bridge is used as a connection. It enables communication between the threads and facilitates data transfer from one thread to another. The data from both threads is processed as they are exchanged.

2.3.3. Advantages and Disadvantages of React Native

- Advantages:
- Reduces the need for using multiple native languages, simplifying the platform development process.
- Shortens the development time for applications.
- Minimizes costs for businesses.
- Enables code reuse of up to 80%.
- Provides a better user experience.
- Does not require deep expertise in the underlying technology; anyone passionate about IT can learn React Native programming.

Disadvantages:

- Currently only supports the two most popular platforms, iOS and Android.
- Performance may not be on par with purely native code applications.
- Limited security measures.

React Native Component	**Android view**	**IOS view**	**Web analog**	**Description**
<View>	<ViewGroup>	<UIView>	A non-scrollling	A container for dividing the interface and containing other components that can handle touch events and provide interactive controls.
<Text>	<TextView>	<UITextView>		Displays text and text styles, and can also handle touch events. Image: Displays images in various formats
<Image>	<ImageView>	<UIImageView>	<img>	Displays images in various formats
<ScrollView>	<ScrollView>	<UIScrollView>	<div>	Similar to View but allows scrolling.
<TextInput>	<EditText>	<UITextField>	<input type="text">	Allows users to input characters

- Not entirely free; some libraries require payment for a full experience.
- Some modules have limited customization capabilities and may not perform optimally.

2.3.4. React Native Components

For each platform, such as Android or iOS, there are specific programming languages used for development. iOS cannot run Android code, and different languages are used to build applications or views for each platform. However, with React Native, after compiling, the component parts are compatible with each platform. These are referred to as native components.

Stylesheet:

In React Native, you only need to use JavaScript to define the styles for your application. All basic components will use a parameter called "style". The style parameters and their values are similar to how CSS works on the web environment, with the difference being that the parameter names follow the CamelCase convention. For example, "backgroundColor" is used instead of "background-color" in CSS.

- Height & Width: The width and height of components determine their size on the screen.
- Fixed Size: The simplest way to specify the size of components is by using the "width" and "height" parameters with fixed values in the style. All sizes used in React Native are in pixels when displayed on the screen.
- Flex Size: Use the "flex" property in the style of components to dynamically adjust the display area based on available space. Typically, "flex: 1" is used when you want a component to expand to fill the maximum space compared to other components sharing the same parent view. This is the highest ratio value that "flex" provides. For other ratios, it will be compared to the sibling components within the same parent view to determine the display ratio.
- Adjusting Layout with Flexbox: A component can define the layout for its child views using the flexbox algorithm. Flexbox is designed to provide a consistent interface across screens of different sizes. It is often used with properties like "flexDirection", "alignItems", and "justifyContent" to create the desired layout.
- Flex Direction: Add the "flexDirection" property to the style of a component to change the arrangement direction of its child components. Should the child components be arranged in a row or a column? By default, child components are arranged in a column.
- Justify Content: Adding the "justifyContent" property to the style of a component will distribute its child components relative to each other in the specified arrangement direction. Should the child components be distributed from the start to the end, be centered, from the end to the start, or evenly spaced? The options correspond to the following values: "flex-start", "center", "flex-end", "space-around", and "space-between".
- Align Items: Adding the "alignItems" property to the style of a component will change the alignment of its child objects in the remaining arrangement direction (e.g., if the initial arrangement direction is row, the remaining arrangement direction will be column, and vice versa). Should the child components be aligned from the start to the end, be centered, or automatically stretch to fill the parent object? The options correspond to the values "flex-start", "center", "flex-end", and "stretch".

Scroll View:

Scroll View is a parent interface component that can contain multiple other components within it, and it has the ability to scroll the components inside it. The scrollable components need to be uniform, and you can scroll vertically or horizontally by setting the "horizontal" attribute.

Tappable Components:

You can use "Tappable" components when you want to capture a tap action on the screen. We can define a function through the "onPress" parameter, which will be called when the touch event begins and ends inside the defined component's enclosing area.

The type of component you use will depend on the type of feedback you want to provide:

- In general, you can use "TouchableHighlight" wherever you would use a button or a link on the web. The background of the view will be darkened when the user presses the button.
- You can consider using "TouchableNativeFeedback" on Android to display a ripple effect as the default touch feedback with Material style to respond to user touch.
- "TouchableOpacity" can be used to provide feedback by reducing the opacity of the button, allowing the background to be seen through while the user is pressing.

If you need to handle an action but don't want any visible feedback to be displayed to the user, use "TouchableWithoutFeedback".

React navigation:

Stack Navigation is a way for us to navigate between screens in React Native. A stack can be a collection of one or more screens. In the case of a stack with multiple screens, it can be imagined as nested pages. For example, when entering an app that requires login, the first screen you may see is the Login Screen. However, since it's the first time logging into the app and you don't have an account yet, you will choose to register and switch to the Register Screen. You can then go back to the Login Screen after successfully registering an account.

Tab Navigation can be a Bottom Tab or Top Tab, but I will build a Bottom Tab to switch between screens. Each tab corresponds to a component that can be a Stack or Screen.

React Grid Layout:

React-Grid-Layout (RGL) is a grid layout system designed for React. As of now, this library has reached 13.3k stars on GitHub, which is quite impressive. With RGL, it has the feature of auto-packing, where widgets can be dragged and resized freely, providing flexible and responsive layouts for multiple breakpoints.

RGL is React-only and does not require jQuery.

Networking in React Native:

In this section, we will learn how to connect our application to the internet protocols. This is extremely important as most applications nowadays follow the Client-Server model.

Many mobile applications need to fetch data by accessing APIs. You can use REST APIs with the POST method, or you may simply need to retrieve static information from a server.

Using Fetch: React Native provides the Fetch API module for network connections. Fetch will be familiar if you have used XMLHttpRequest or other networking APIs before. You may need to refer to the Fetch documentation for more information.

Creating an HTTP Request: When you want to retrieve content by making a simple call to a URL, you can simply place that URL in the fetch:

Fetch has various optional parameters that allow you to customize the HTTP request. For example, you may want to add specific headers and make a POST request.

Handling the response: Networking is inherently asynchronous. The Fetch method returns a Promise, which makes it easy to write code to handle asynchronous operations.

Using other Networking libraries: The XMLHttpRequest API is integrated into React Native. This means you can use third-party libraries like frisbee or axios and work with them. Alternatively, you can also use the XMLHttpRequest API.

WebSocket support: React Native also supports WebSocket, which is a method for creating a continuous connection channel using TCP connections.

Using Firebase in React Native:

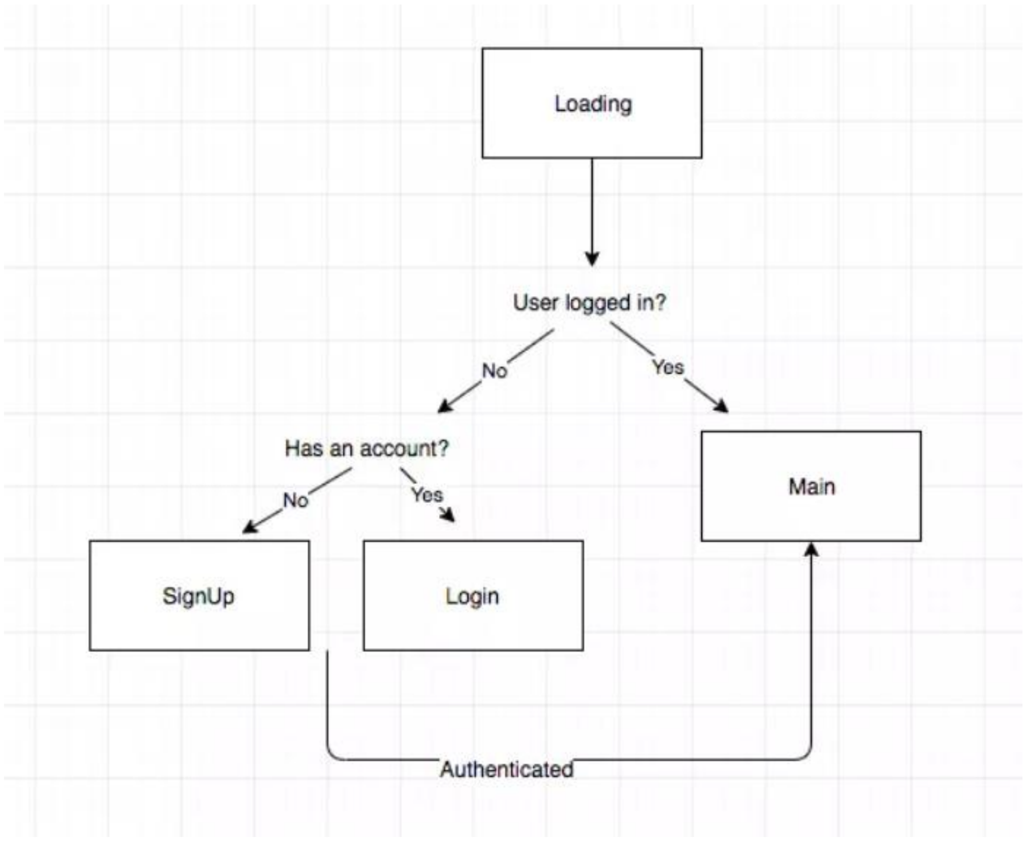


Figure 10: App integrated with Firebase for login

According to the diagram, we will have 4 screens: Loading, SignUp, Login, Main.

The Loading screen will be displayed until the user's state is determined. The SignUp screen is where users can register an account. The Login screen is where users can log in, and the Main screen is where we display the authenticated user.

We will use react-navigation to navigate between the 4 screens and implement the Authentication feature using email and password for login.

2.4. Face Recognition for Door Unlocking

Face recognition with MTCNN and FaceNet: MTCNN and FaceNet are two well-known networks for processing Face Recognition problems in general. Combining them, when given input images/videos with multiple people in real-world scenarios, can yield good results. In this case, MTCNN plays the role of Face Detection/Alignment, extracting faces from the frames as bounding boxes and adjusting/resizing them to the correct input shape for the FaceNet network. FaceNet, on the other hand, acts as a Feature Extractor + Classifier for each bounding box, generating embeddings and performing differentiation and recognition of faces.

2.4.1. Understanding MTCNN

MTCNN, which stands for Multi-Task Cascaded Convolutional Neural Networks, is a neural network that helps detect faces and facial landmarks in images. It was published in 2016 by Zhang et al. In addition to face detection, MTCNN can be used to detect other objects such as license plates and vehicles. MTCNN consists of three neural networks, also known as stages: P-Net, R-Net, and O-Net (PRO!).

The three main tasks of MTCNN are:

- Face classification: MTCNN classifies whether a region in an image contains a face or not.
- Bounding box regression: MTCNN performs regression to refine the bounding box coordinates around detected faces, making them more accurate.
- Facial landmark localization: MTCNN identifies and localizes specific facial landmarks, such as the eyes, nose, and mouth, within the detected face regions.

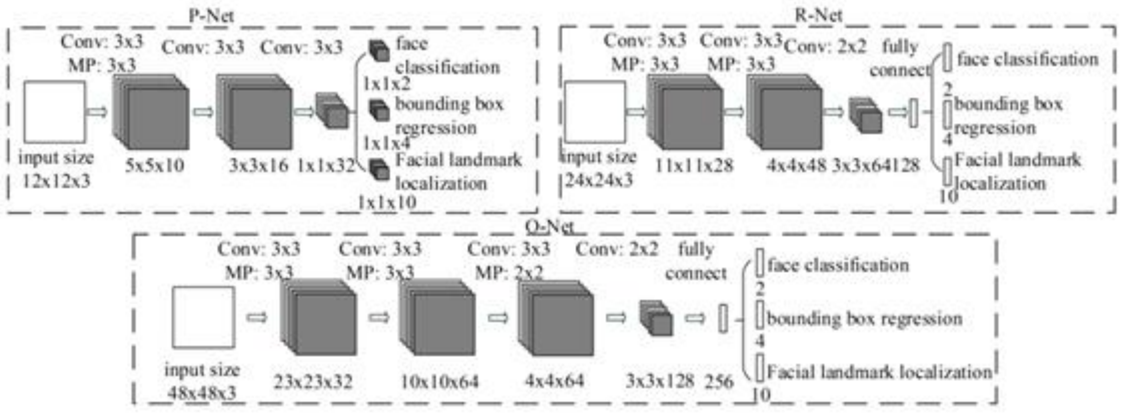


Figure 11: Architecture of an MTCNN network

Proposal Network (P-Net):

In the first step, a Fully Convolutional Network (FCN) is used. FCN differs from a CNN in that it does not use Dense layers. P-Net is used to obtain potential bounding boxes and the coordinates of those bounding boxes. Bounding box regression is a technique used to predict the position of a bounding box when we need to detect an object (in this case, a face). After obtaining the coordinates of the bounding boxes, some adjustments are made to eliminate overlapping bounding boxes. The output of this step is all the filtered bounding boxes.

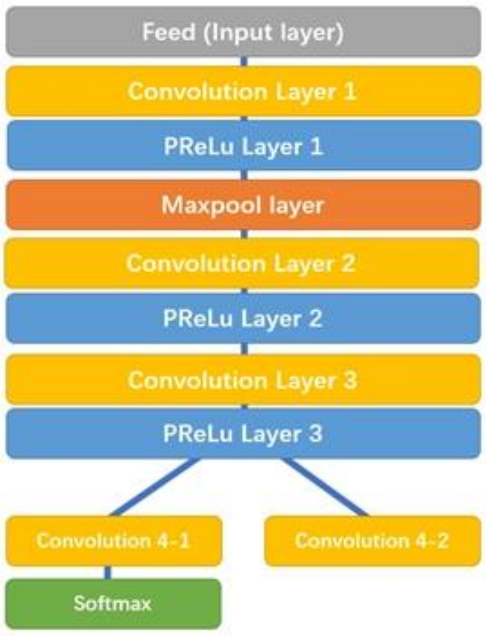


Figure 12: P-Net architecture details

In this stage, we incorporate the PRelu layer. To simplify, this layer essentially preserves non-negative values while multiplying negative values by learned coefficients.

After PRelu layer 3 splits into 2 branches, branch 4-2 is utilized to predict the coordinates of the bounding box, while branch 4-1 is employed to predict the probability of a face appearing within the bounding box.

- Refine Network (R-Net):

All bounding boxes generated by P-Net are funneled into R-Net, a CNN (Convolutional Neural Network). Unlike the preceding FCN architecture, R-Net serves a multi-faceted role, reducing the number of bounding boxes, refining their coordinates, and implementing Non-maximum Suppression (NMS) for optimal selection. R-Net also features two branches, one for predicting the presence of a face within the bounding boxes and the other for predicting the coordinates of these bounding boxes.

The output of R-Net seamlessly feeds into O-Net, where the model identifies facial landmarks, including both eyes, the nose, and two mouth locations. As you can see in the image below, near the end of O-Net, there are three branches: one predicts the probability of a face, another forecasts the bounding box coordinates, and the third pinpoints the coordinates of facial landmarks. These landmark coordinates consist of x and y values, impacting the network's layer complexity.

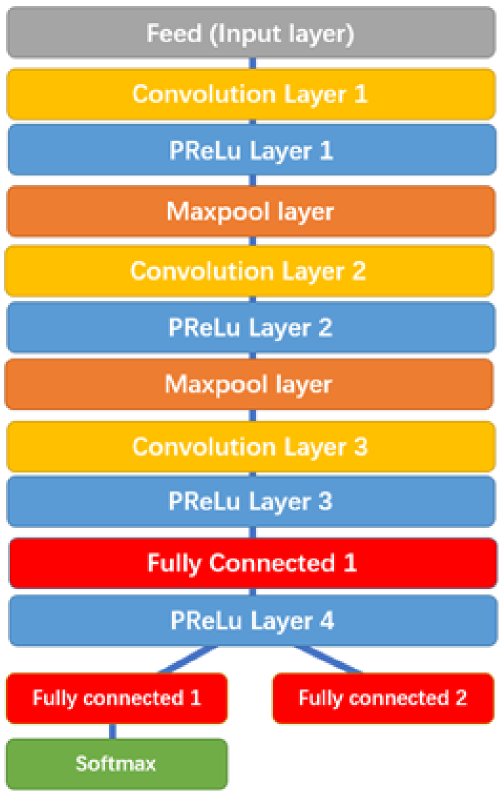


Figure 13: Details of R-Net architecture

- Output Network (O-Net):

The R-Net's output serves as the input for O-Net, where the model identifies critical facial landmarks, including two eyes, the nose, and two mouth locations. As you can see in the image above, near the end we have 3 branches: one for predicting the face's probability, another for bounding box coordinates, and a third for facial landmark coordinates. These facial landmark coordinates consist of both x and y values, affecting the layer structure accordingly.

2.4.2. FaceNet

FaceNet, created by Google researchers in 2015, is a deep neural network tailored for extracting features from facial images. It takes a person's face image as input and generates a compact 128-dimensional vector, encapsulating the face's crucial features. This vector serves as a consistent and efficient representation for comparing faces and accurately identifying individuals.

In machine learning, this vector is commonly referred to as an embedding. It encapsulates all the critical information from an image. Essentially, FaceNet condenses a person's face into a 128-dimensional vector, a process similarly applied for embedding other faces.

A potential approach for recognizing a person in an unknown image involves computing its embedding, measuring the distance to the embedding of a known person, and determining if the embedding of the face in question is sufficiently close to the embedding of person A. When the embeddings are sufficiently similar, it indicates that the image likely contains the face of person A.

- Basic concepts:

+ Embedding Vector: is a fixed-dimensional vector, typically smaller than standard feature vectors, that is learned during the training process. It encapsulates a set of features crucial for object classification within that dimension. This transformed space of embeddings is particularly valuable for identifying Nearest Neighbors within a given cluster, relying on the distance relationships between the embeddings to make these determinations.

+ Inception V1: introduced by Google in 2014, is a Convolutional Neural Network (CNN) structure characterized by Inception blocks. These blocks enable parallel learning, allowing one input to be processed by multiple Convolution layers, producing distinct results that are then concatenated into a single output. This parallel approach enhances the network's ability to capture detailed features, surpassing traditional CNNs. Additionally, Inception V1 employs 1x1 Convolution blocks to reduce network size, expediting training while maintaining efficiency.

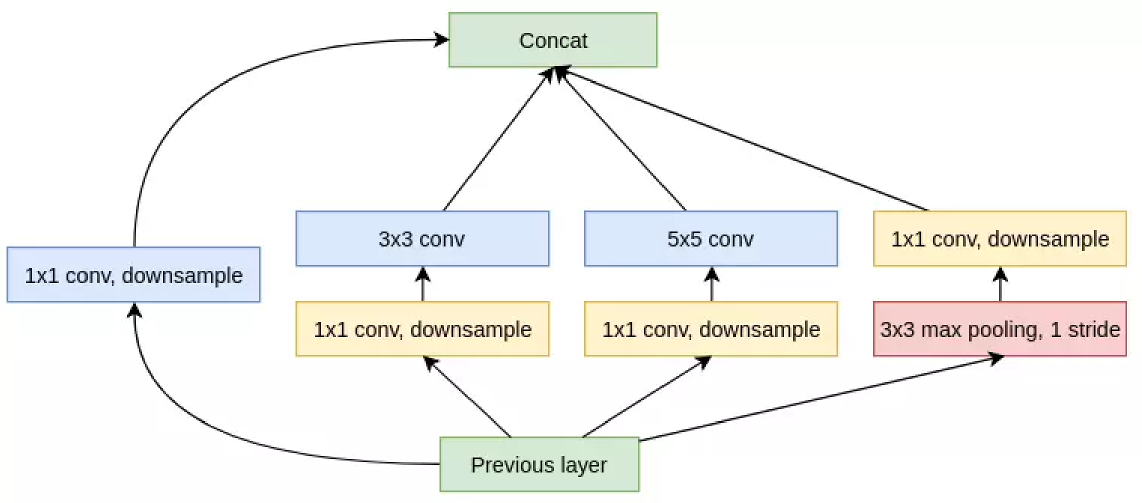


Figure 14: Inception V1 architecture

2.4.3. Implementation steps

Training process:

• Utilize a dataset comprising a diverse set of individuals, with each person associated with a specific number of photographs.

• Construct a Deep Neural Network (DNN) designed to serve as a feature extractor for the previously mentioned dataset. The desired outcome is to produce embeddings with a dimensionality of 128. In the academic paper, two noteworthy network architectures are presented: Zeiler&Fergus and InceptionV1.

• Training the DNN network so that the embedding results have good recognition ability, includes two things: using l2l2l2 normalization (Euclidean Distance) for the output embeddings and optimizing the parameters in the membrane network using Triplet Loss.

• The Triplet Loss function is intricately connected with the Triplet Selection method, a crucial component that strategically chooses embeddings to optimize the learning process most effectively.

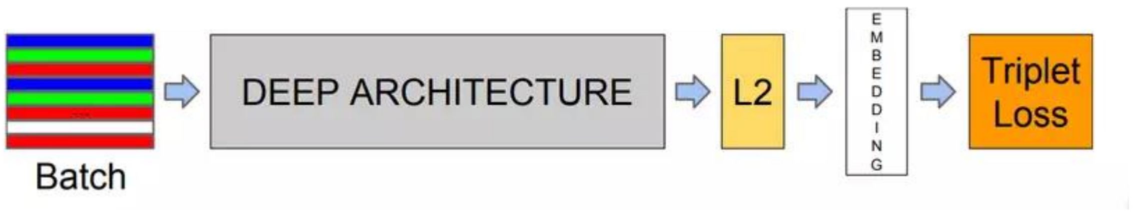


Figure 15: Training process

Inference Process:

• The process involves forwarding the face image to the Feature Extractor network, which then produces a single embedding as its output.

• Utilize the L2 (Euclidean Distance) function to compare the new embedding with the existing embeddings in the dataset. This approach mimics the k-Nearest Neighbors (k-NN) algorithm with a k value set to 1, classifying the face image based on its closest neighbor in the embedding space.

2.5. Implement ESP NOW communication between 4 esp

2.5.1. ESP NOW introduction

ESP-NOW, developed by Espressif, is a communication protocol designed for efficient short packet transmission. It enables multiple devices to communicate without relying on Wi-Fi, functioning on a low-power 2.4GHz wireless connection. Initial pairing is required for secure peer-to-peer communication. Once paired, the connection remains persistent, even if a device reboots or loses power, ensuring seamless data exchange.

ESP-NOW is a technology utilized by the ESP8266 to enable rapid data packet transmission. It finds applications in various domains, including smart lighting devices and remote sensor control. Within ESP-NOW, the IEEE802.11 standard is leveraged in conjunction with Information Element (IE) functions and CCMP encryption, ensuring robust and reliable data transfer.

One of the notable strengths of ESP-NOW is its flexibility, allowing for both one-way and two-way communication in diverse configurations and setups. This adaptability makes it a versatile choice for a wide range of IoT and communication scenarios.

Figure 16: ESP-NOW protocol

2.5.3. Advantages and disadvantages of ESP NOW

Advantage:

- Encrypted and unencrypted unicast communication;
- Mixed encrypted and unencrypted peers;
- Can transmit data up to 250 bytes;
- There is a callback function to notify the application layer about successful or failed transmission.

Disadvantages:

- Does not support broadcast
- Limit encrypted peers. Approximately 10 encrypted peers in Station mode, 6 peers in SoftAP or SoftAP + Station mode. Unencrypted peers can have more numbers but must be less than 20.
- Payload is limited to 250 bytes.

Simply put, ESP-NOW is a fast communication protocol that can be used to exchange small messages (up to 250 bytes) between ESP8266 boards.

2.5.4. One-way communication ESP NOW

In one-way communication, there can be situations such as the following:

- Transmitting data from one ESP board to another using ESP-NOW is a straightforward setup. It's highly practical for sending information between boards, such as sensor readings or GPIO control commands (e.g., turning ON or OFF).

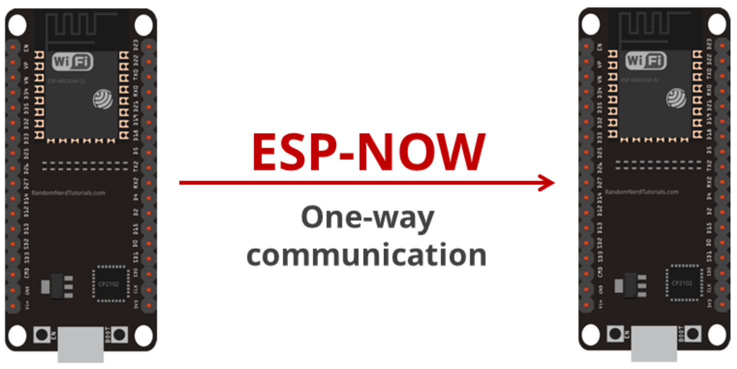


Figure 17: One-way communication from one board to another

- In this configuration, a "master" ESP board communicates with multiple "slave" Wifi32 Node boards. The master ESP board can send either identical or distinct commands to individual ESP boards. This setup is ideal for applications such as remote controls, where multiple ESP panels across a house can be controlled by a central main ESP panel, offering versatile and centralized control over various devices or systems.

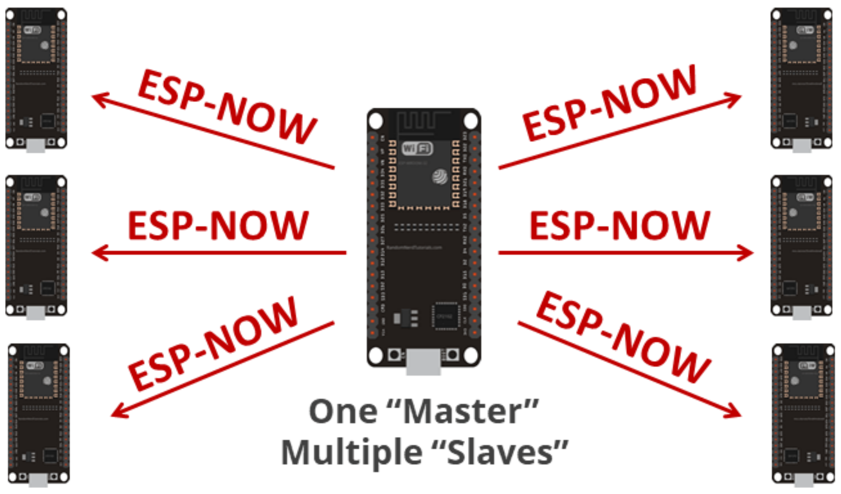


Figure 18: One-way communication from one board to other boards

- In this setup, one ESP board serves as the receiver, gathering data from multiple "master" devices. This configuration is highly suitable for scenarios where you need to aggregate data from several sensor nodes onto a single ESP32 board. It can be configured as a web server, enabling you to display data from all connected devices, making it especially useful for applications like data monitoring and visualization from various sources.

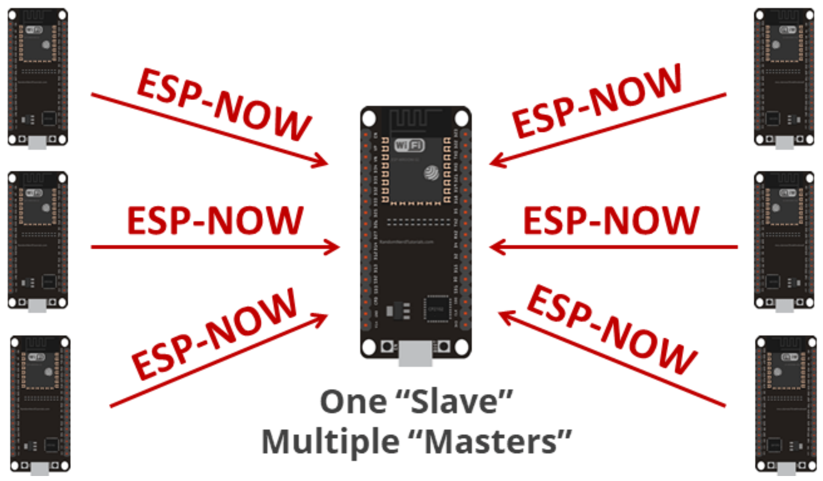


Figure 19: One-way communication from multiple boards to one board

2.5.5. Two-way communication ESP NOW

ESP-NOW enables each panel to function as both a "sender" and a "receiver" concurrently, facilitating two-way communication capabilities between the panels. This means that panels can not only transmit data but also receive data from other panels, allowing for versatile bidirectional communication.

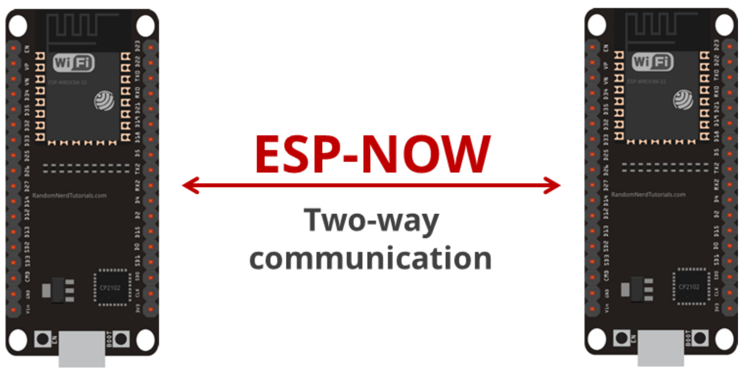


Figure 20: 2 boards communicating 2-way with each other

It's possible to expand this configuration by adding more boards, creating a network-like setup where all ESP32 boards communicate with each other. This interconnected arrangement enables extensive communication and data exchange among multiple ESP32 boards, forming a network of interconnected devices.

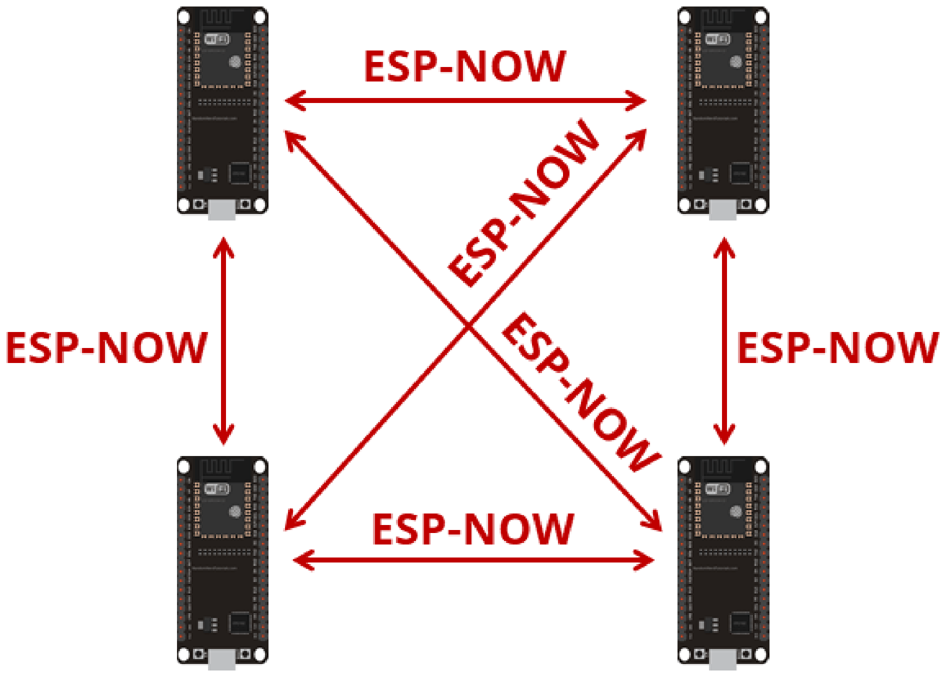


Figure 21: Two-way communication with multiple boards together

In summary, ESP-NOW is well-suited for constructing a network in which multiple ESP32 boards can seamlessly exchange data with one another.

2.5.6. Information in the ESP NOW protocol

In a local device context: PMK, Role

In a local peer context: Key, MAC Address, Role, Channel

Device	Information	Value/ length	Description	Note
Local device	PMK	16 bytes long	Primary Master Key, such as KOK in the API, is used to encrypt the peer's Key.	The system will automatically generate a default PMK without requiring manual configuration.
	Role	IDLE CONTROLLER SLAVE COMBO	This is the role of the device IDLE (no role assigned), CONTROLLER, SLAVE, COMBO (takes on 2 roles as controller & slave).	The local device role is defined through the SoftAP or Station of the ESP-NOW IDLE: transmission will not be allowed CONTROLLER (priority to Station interface) SLAVE (priority to SoftAP interface) COMBO

				(priority to both SoftAP interface and Station interface)
Peer	Key	16 bytes long	used to encrypt the payload key during communication with peers	
	Mac Address	6 bytes long	The peer's MAC address	The MAC address must be the same as the sending address. For example, if the packet is sent from the Station interface, the MAC address must be the same as the Station address
	Channel	Value from 0 - 255	Channel for local devices and peers to connect to each other.	

To set up the sending callback function and handle various scenarios, as well as configuring the receiving callback function, follow these steps:

Setting Up the Sending Callback Function:

- Define Callback Function: Create a sending callback function that will be invoked after each transmission attempt.
- Registration: Register the callback function, specifying it to be called upon transmission events.
- Handling Failures: In your callback function, implement logic to handle failures if the application layer does not receive the packet even if the callback returns success. Possible causes for this include:

- Attacks from other devices.
- Incorrect key encryption setup.
- Packet loss within the application layer.

- Handling Success: For multicast mode (including broadcast), if the callback function returns success, it signifies that the packet was successfully sent.

Setting Up the Receiving Callback Function:

- Define Callback Function: Create a receiving callback function to notify the application layer when a sent packet is received. This function should return information like the peer's MAC address and the packet's payload.
- PMK Configuration: If key encryption is used, configure the Pairwise Master Key (PMK) using the PMK configuration API (KOK). If the PMK is not explicitly configured, the default PMK will be applied.
- Protocol Selection: Choose the appropriate protocol for your device, typically using the Station interface for a CONTROLLER role, SoftAP interface for a SLAVE role, and COMBO for specific configurations.
- Key Configuration: Set up the encryption key for your device, and use the "add peer" function to manage peer devices.
- Sending Data: To send data, call the "send" function, which will return the payload of the sent packet.

By following these steps, you can effectively establish sending and receiving callback functions, configure encryption keys, and manage communication within your ESP-NOW network.

2.5.7. ESP gateway with wifi to push and receive data from firebase

If you intend to utilize Wi-Fi for web server hosting while concurrently receiving sensor data from other panels via ESP-NOW, careful planning and coordination between these functions are essential:

- The transmitting ESP32 board should operate on the identical Wi-Fi channel as the receiving board to ensure proper communication and data exchange between them.
- The Wi-Fi channel for the receiver board is automatically assigned by your Wi-Fi router, ensuring optimal channel selection for efficient communication.
- To enable seamless operation, the receiver board's Wi-Fi mode must be set to both access point and station (WIFI_AP_STA).
- You can manually configure the sender and receiver boards to operate on the same Wi-Fi channel, or alternatively, implement a small code snippet in the sender to synchronize its channel with the receiver board.

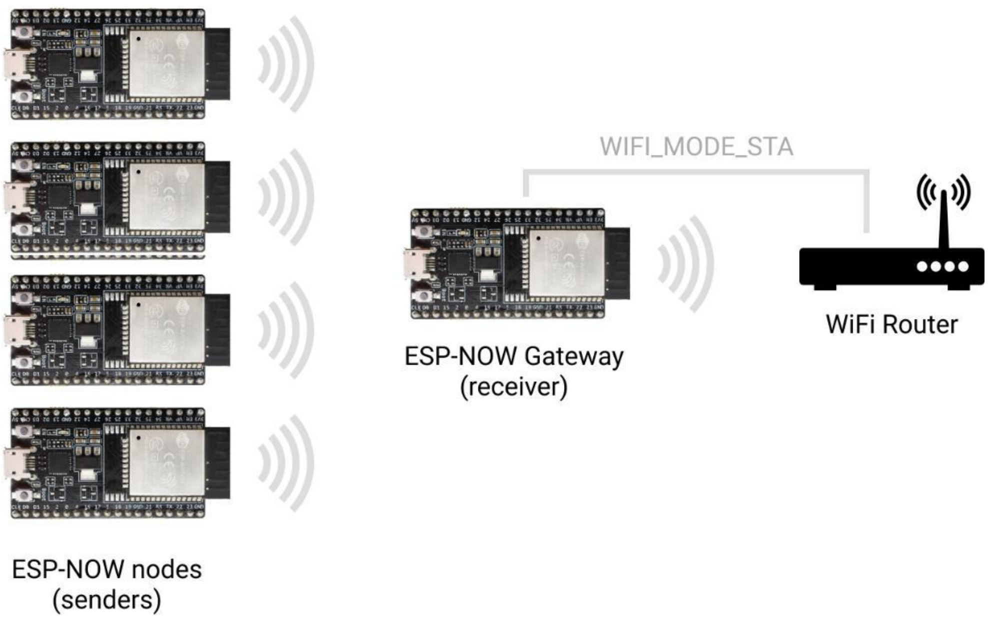


Figure 22: ESP NOW gateway

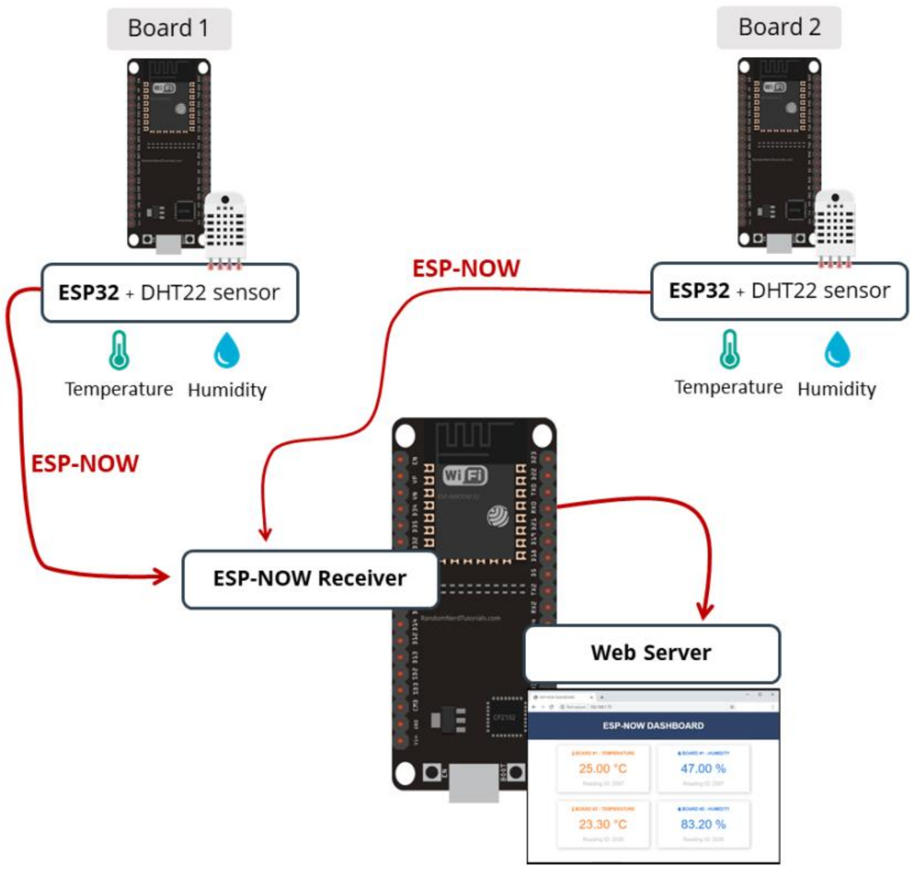


Figure 23: ESP NOW receives data and pushes it to the cloud

- Two ESP sending boards are responsible for transmitting DHT temperature and humidity data via ESP-NOW to an ESP receiving board, which aggregates multiple ESP devices configured as a single ESP-NOW network.
- The ESP receiver board receives incoming packets and subsequently presents the received sensor readings on the web server.

- The web server is updated automatically each time it receives new information using Server-Sent Events (SSE).

2.6. OTA firmware update

2.6.1. Introducing OTA

OTA (Over the Air) firmware update is the process of downloading new firmware into the ESP module instead of using the Serial port. This feature is really useful in many cases where physical connection to the ESP Module is limited.

2.6.2. How OTA web updater work:

- The initial sketch must be uploaded via the serial port, and this sketch should include the code required to establish the OTA Web Updater, allowing subsequent updates to be performed via your web browser.
- The OTA Web Updater sketch establishes a web server that enables you to upload new sketches via a web browser.
- In each sketch you upload, it's essential to implement OTA processes to facilitate future over-the-air updates and uploads.
- Uploading code without the OTA routine will result in the inability to access the web server and perform new sketch uploads over the wireless network.

2.6.3. New methods to update firmware for esp8266

There are 3 methods to load new firmware for ESP8266:

- Using Arduino IDE: this method requires using Arduino IDE software to load. It is not very flexible and should only be used when developing software.
- Using Web Browser: this method uses the ESP8266 as a server, then we access the ESP's ip and upload the new firmware and the ESP8266 will automatically load it into its flash memory.
- Using HTTP Server: this method ESP8266 automatically connects to the server, checks if there is new firmware, if so, downloads that new firmware file and updates it.

- Using Arduino IDE:
Performing OTA updates with the Arduino IDE module is typically reserved for specific scenarios, including:
+ While the firmware is active, enabling the replacement of the serial port for updates.
+ When updating modules in small quantities
+ Only if the Arduino IDE module and the computer are on the same network.
However, a challenge arises with this approach: it opens the door for potential unauthorized access, as anyone on the same network with a computer and an Arduino can potentially make alterations and upload various firmware to the ESP.
- Using Web browser:
The updates detailed in this section are executed via a web browser, offering utility in the following scenarios:
+ Once the application is installed, if direct downloads from the Arduino IDE are not convenient or face downloading issues due to various reasons.
+ After installing the application, if the user encounters difficulties in performing OTA updates through external update servers
+ To update a specific component of the device post-installation, especially when setting up an update server is impractical.

- Using HTTP Server:
With the two methods above, you can easily update the Firmware via the internal WiFi network. However, when deploying actual applications, we will need to update the Firmware remotely via the Internet, and need a Server to store the firmware and manage versions.

Most commonly:

• When the ESP8266 initializes, typically after a specified duration like one day, it establishes a connection with the server and transmits its current version information.

• When the server detects that an upgrade is necessary for the current version, it will respond by providing the new firmware.

• If the current version of the ESP8266 does not need to be updated, it will return code 304.
To accomplish this, the procedure needs to be implemented on both the ESP8266 and the server side.

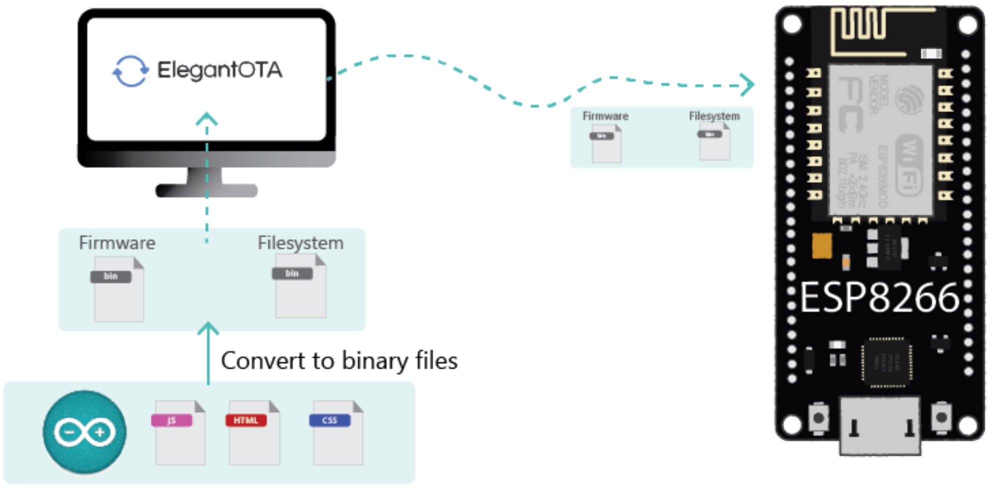


Figure 24: OTA communication

Updater class:

The Updater component is situated in the Core and serves the purpose of managing firmware writing to the flash memory, verifying its integrity, and signaling the bootloader to load the firmware during the subsequent boot.

Update process - memory view:

- The new sketch will be stored in the space between the old sketch and the spiff.
- During the next reboot, the "eboot" bootloader checks the commands.
- The new sketch will be copied.
- New sketch launch.

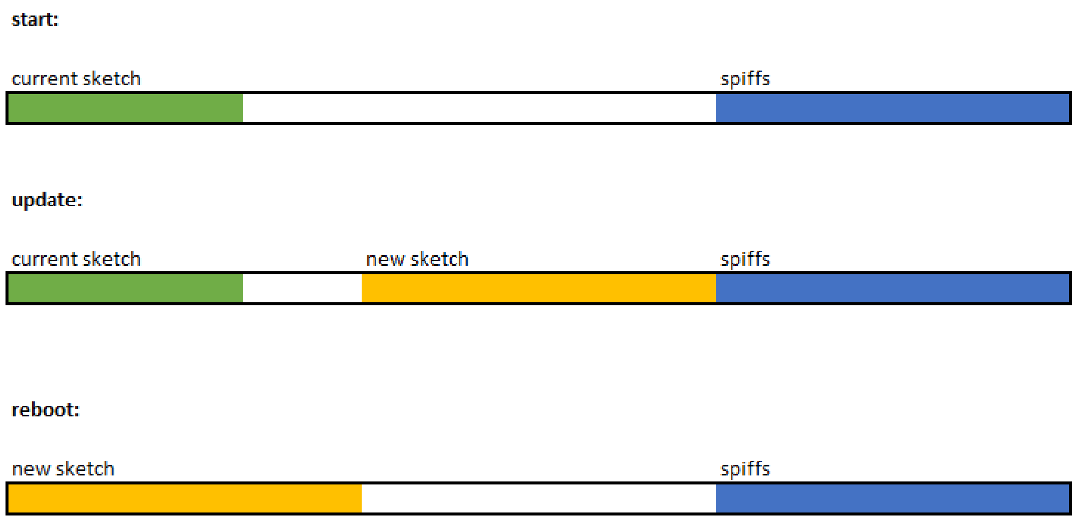


Figure 25: Update process

2.6.4. Advantages and disadvantages of OTA

Advantages:

+ Save time and effort by updating the program code of the ESP8266 directly through the Wi-Fi network, eliminating the need to disassemble the ESP8266 from the circuit. This streamlined process is more efficient compared to the traditional method of loading programs via a cable.

+ OTA offers convenience and flexibility, enabling program code updates from anywhere and at any time, provided there's a Wi-Fi network connection available. You no longer require your computer or USB cable for the update process.

+ Simplify deployment: ESP8266 OTA facilitates the straightforward deployment of applications and program code updates across a substantial number of ESP8266 devices within a designated network.

Disadvantages:

+ Memory constraint: Due to the limited memory storage capacity of the ESP8266 device, it's crucial to carefully manage OTA updates to prevent exceeding these constraints. This limitation can impact the range of applications you can effectively deploy.

+ Security element: Provides remote update capabilities for devices that may have security issues. This requires implementing tight security measures to ensure that only authorized people can use this feature.

+ Connectivity dependency: OTA updates rely on a stable and dependable Wi-Fi network connection for successful implementation. If the network connection experiences issues, the OTA process may encounter failures, potentially complicating the updating process.

3. Result

Video demo results of pushing data to cloud firebase:

https://drive.google.com/file/d/1t64b1mQk26pv34jBbzU5YL4gOxajd8jw/view?usp=sharing

React native screen:

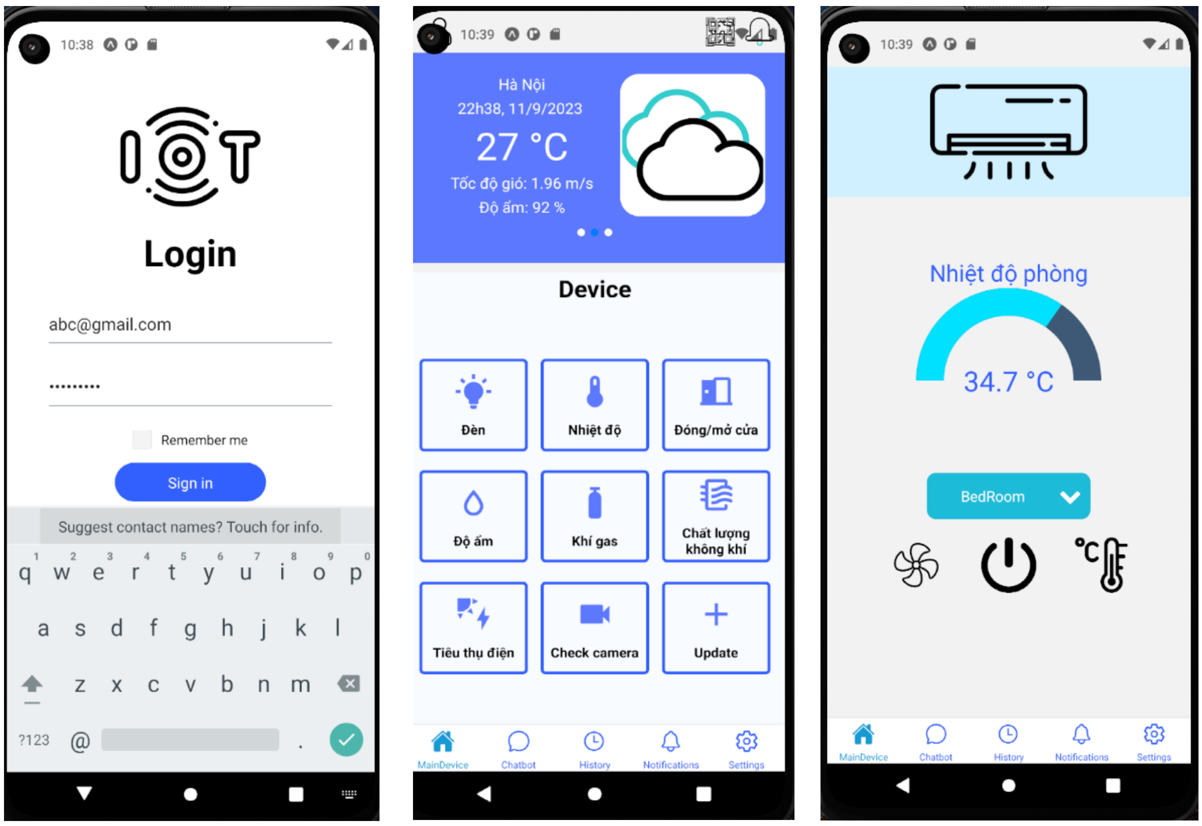
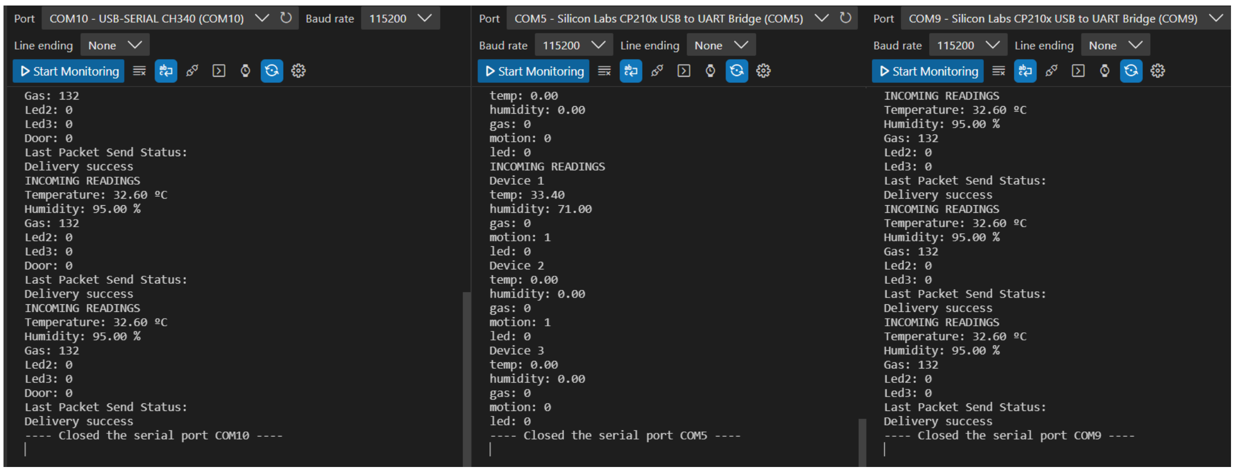


Figure 26: App UI

Esp now send data between 3 esp



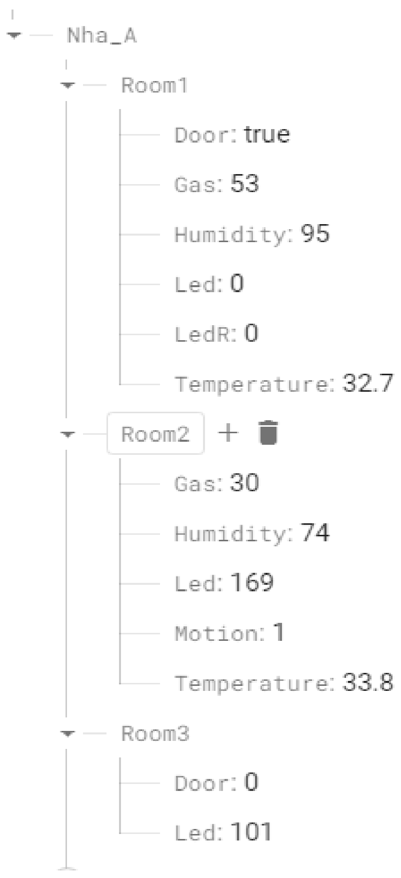
Port COM10 - USB-SERIAL CH340 (COM10)
Baud rate 115200
Line ending None
Start Monitoring
Port COM5 - Silicon Labs CP210x USB to UART Bridge (COM5)
Baud rate 115200
Line ending None
Start Monitoring
Port COM9 - Silicon Labs CP210x USB to UART Bridge (COM9)
Baud rate 115200
Line ending None
Start Monitoring

```
Gas: 132
Led2: 0
Led3: 0
Door: 0
Last Packet Send Status:
Delivery success
INCOMING READINGS
Temperature: 32.60 ºC
Humidity: 95.00 %
Gas: 132
Led2: 0
Led3: 0
Door: 0
Last Packet Send Status:
Delivery success
INCOMING READINGS
Temperature: 32.60 ºC
Humidity: 95.00 %
Gas: 132
Led2: 0
Led3: 0
Door: 0
Last Packet Send Status:
Delivery success
---- Closed the serial port COM10 ----
```

```
temp: 0.00
humidity: 0.00
gas: 0
motion: 0
led: 0
INCOMING READINGS
Device 1
temp: 33.40
humidity: 71.00
gas: 0
motion: 1
led: 0
Device 2
temp: 0.00
humidity: 0.00
gas: 0
motion: 1
led: 0
Device 3
temp: 0.00
humidity: 0.00
gas: 0
motion: 0
led: 0
---- Closed the serial port COM5 ----
```

```
INCOMING READINGS
Temperature: 32.60 ºC
Humidity: 95.00 %
Gas: 132
Led2: 0
Led3: 0
Last Packet Send Status:
Delivery success
INCOMING READINGS
Temperature: 32.60 ºC
Humidity: 95.00 %
Gas: 132
Led2: 0
Led3: 0
Last Packet Send Status:
Delivery success
INCOMING READINGS
Temperature: 32.60 ºC
Humidity: 95.00 %
Gas: 132
Led2: 0
Led3: 0
Last Packet Send Status:
Delivery success
---- Closed the serial port COM9 ----
```

Figure 27: Transmission results between 3 esp boards using esp now



```
Nha_A
  Room1
    Door: true
    Gas: 53
    Humidity: 95
    Led: 0
    LedR: 0
    Temperature: 32.7
  Room2
    Gas: 30
    Humidity: 74
    Led: 169
    Motion: 1
    Temperature: 33.8
  Room3
    Door: 0
    Led: 101
```

Figure 28: ESP now and push data to firebase

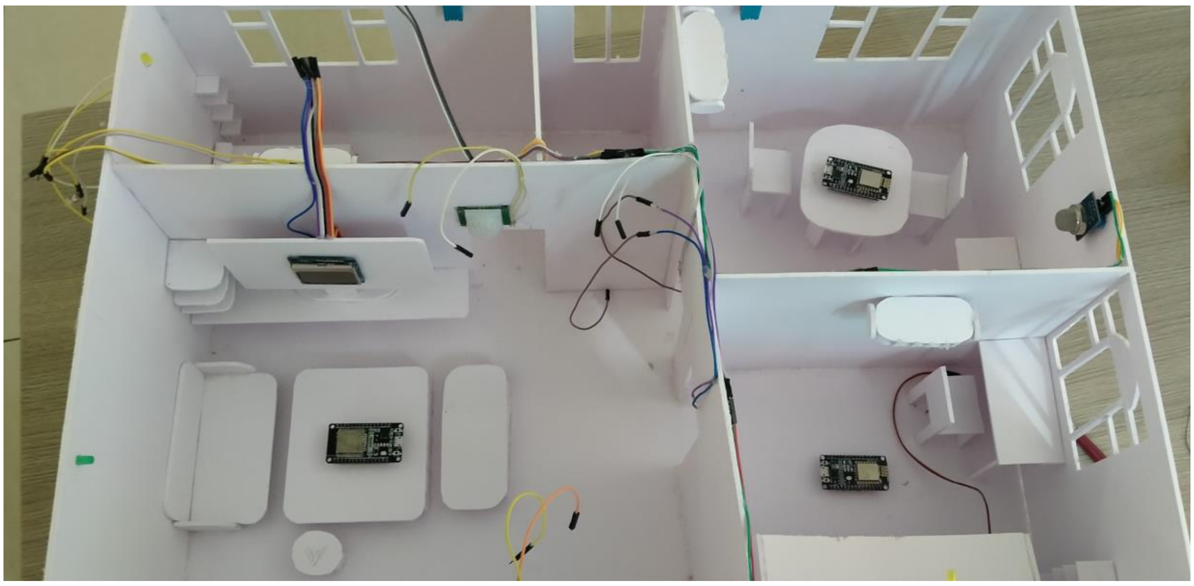


Figure 29: Demo house model

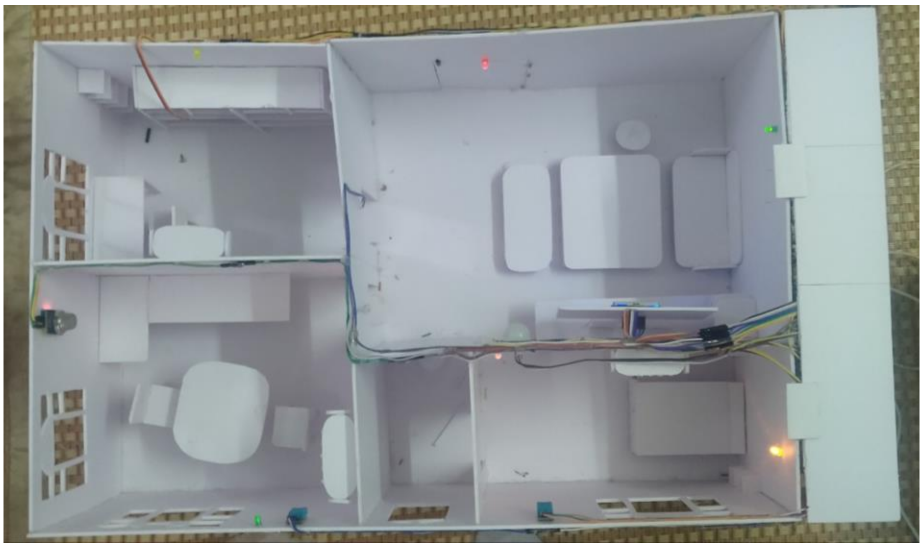


Figure 30: Demo house model

4. Conclusion

While participating in the IoT challenge, we successfully executed the Smart Home project. This encompassed acquiring expertise in connecting ESPs through the ESP-NOW protocol, establishing ESP connections to Firebase Cloud, performing read and write operations for sensor data, facilitating push and pull interactions with Firebase, controlling devices through a React Native app, and implementing facial recognition to unlock doors.

Source code: *https://github.com/hieu2k2boss/IOT_OnePiece*

5. References

[1] Firebase intergrate with ESP8266, https://www.instructables.com/id/Firebase-Integrate-With-ESP8266/

[2] Communicate DHT11 sensor with NodeMCU ESP8266, https://dientuviet.com/giao-tiep-dht11-voi-nodemcu-esp8266/

[3] React native documentation, https://reactnative.dev/docs

[4] Fetch request docs, https://developer.mozilla.org/en-US/docs/Web/API/Request

[5] ESP – NOW, https://www.espressif.com/en/products/software/esp-now/resources

[6] ESP-NOW with ESP8266: Send Data to Multiple Boards (one-to-many), https://randomnerdtutorials.com/esp-now-one-to-many-esp8266-nodemcu/

[7] ESP32: ESP-NOW Web Server Sensor Dashboard (ESP-NOW + Wi-Fi), https://randomnerdtutorials.com/esp32-esp-now-wi-fi-web-server/

[8] Getting Started with ESP-NOW (ESP8266 NodeMCU with Arduino IDE)

[9] ESP8266 NodeMCU OTA (Over-the-Air) Updates – AsyncElegantOTA using Arduino IDE, https://randomnerdtutorials.com/esp8266-nodemcu-ota-over-the-air-arduino/

[10] Update firmware, https://arduino.esp8266.vn/network/ota.html#arduino-ide0

[11] FaceNet: A Unified Embedding for Face Recognition and Clustering, https://arxiv.org/pdf/1503.03832.pdf

[12] Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, Senior Member, IEEE, and Yu Qiao, Senior Member, IEEE.

[13] Face Recognition with FaceNet and MTCNN, https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/