VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MULTIDISCIPLINARY PROJECT

Class: CC03 | Group: 2

Home Guardian System

Instructor: Professor Võ Thị Ngọc Châu

Students: Đoàn Trương Thanh Tịnh - 2153046
Nguyễn Chánh Tín - 2153043
Trần Minh Triết - 2153057
Lê Việt Tùng - 2153083
Nguyễn Quang Thiện - 2153994

HO CHI MINH CITY, February 2024

Contents

1 Group information

1.1 Roles

Name	ID	Role
Nguyễn Chánh Tín	2153043	Member
Trần Minh Triết	2153057	Note-taker
Lê Việt Tùng	2153083	Member
Nguyễn Quang Thiện	2153994	Member
Đoàn Trương Thanh Tịnh	2153046	**Leader**

Table 1: Team memberlist

1.2 Meeting channel

Messenger: We gathered through Facebook and choose Messenger for normal correspondence.

Google Meet: For the important decision or weekly meeting, Google Meet is a convenient tool for us to discuss in real-time.

1.3 Note & storage

OneNote: Each discussion will be summarize in OneNote before written into Weekly Report.

OneDrive: Notes, documents will be stored in our team's OneDrive

1.4 Agreement

To advoid schedule conflicts, we decided to have a regular meeting on 20h every Saturday. But it may be changed on the examination weeks, or if it overlaps with the lecturer's meeting.

1.5 Contribution

No.	Fullname	Student ID	Work	Percentage of work
1	Lê Việt Tùng	2153083	- Database design,	100%
2	Nguyễn Chánh Tín	2152108	Backend developer	100%
3	Trần Minh Triết	2153057	Frontend developer	100%
4	Đoàn Trương Thanh Tịnh	2153046	- IoT developer	100%
5	Nguyễn Quang Thiện	2153994	- IoT developer	100%

2 IoT system

2.1 Hardware device system

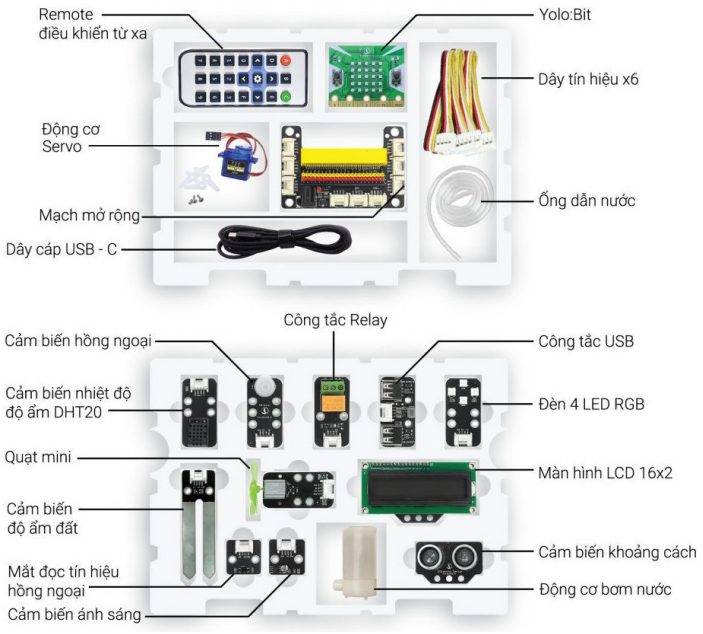


Figure 1: Devices List

To serve the Multidisciplinary Project, the image above has a few basic gadgets that must be assembled and controlled to become an IoT home. Furthermore, for enhancing the IoT Home needs more devices such as buzzers, air quality sensors, face recognition cameras, etc.

Input devices and output devices are essential parts of a home IoT (Internet of Things) system that are used to collect data and take actions depending on that data. The following are each term's basic definitions:

1. Input devices:

- Input devices are used to collect information from the environment or users and transmit data to the IoT system.
- Examples of input devices include face recognition cameras, air quality sensors, temperature sensors, humidity sensors, light sensors, etc.
- The function of input devices is to provide input information so that the IoT system can understand and respond appropriately.

2. Output devices:

- Output devices are used to display information or perform actions based on data from the IoT system.
- Examples of output devices include smart lights, smart displays, LEDs, alert devices (buzzer), controller devices (servo), etc.
- The function of output devices is to display information or generate signals or actions to notify or fulfill requests from the IoT system.

In a home IoT system, the input and output devices typically work together to create an automated, flexible, and comfortable system for users. Moreover, there are some initial ideas for the IoT Home system:

- Output: Turn ON/OFF air conditioner, Input: Clock Timer or using App.
- Output: Turn UP/DOWN air conditioner, Input: Temperature sensors or using App.
- Output: Open door (using servo), Input: Face recognition (camera) or using App.
- Output: Control fans (using servo), Input: Temperature sensors + Air quality sensors or using App.

- Output: Turn ON/OFF lights, Input: Infrared sensors or using App.
- Output: Turn on alarm, Input: Temperature sensors (have fire) + Air quality sensors for checking or using App.

2.2 UI/UX

In order to control as well as monitor the equipment status of our system, we decided to create a web application. The reason why we choose to do a web application instead of a normal application is to make it so that the user can easily access the app from anywhere, with any device that has a web browser. We believe that this is more convenient as well as more accessible for the majority of the user.

For the front-end, we will use Next.js and React.js to build a user interface that is simple, easy to use but is well equipped with functions to control the smart home system. The web application will include (but not limited to) these sections:

- A control page where the user can see the equipment that's connected to the IOT system and can perform some action on them, for example: turning on the light, adjust the fan speed, set a timer to turn the AC on/off,...
- A device manager page to check the information of connected device. The user can add new device or remove unnecessary device from the system using this page.
- A information dashboard where the user can see the information given by the sensor in their house like room temperature, humidity inside the house, lighting level in a room, lock status of a door,... If a device has battery level, this dashboard should also show it.

3 Working plan

We have planned the working schedule and task that should be completed by each week.

The primary position of each member

- IoT gateway, connection between app, Microbit, broker: Tịnh
- Microbit programming:
  - Control input: Tịnh
  - Process input data: Thiện
  - Control output: Thiện
- Web app development:

– Front-end: Trần Minh Triết
– Back-end: Nguyễn Chánh Tín
– Database: Lê Việt Tùng

Table 2: Working plan table

Time line	Tasks	Expected result
Week 4		
	- Held the first meeting and decided on the general objective and goal of the project.	- Team member has a grasp of the project and the required system. - Has an agreed meeting plan and work goal for each member.
	- Make plan and describe the system functionality.	
	- Decide on the required devices and tools to complete the project.	
	- Divide general work between members.	
Week 5		
	- Describe the domain context of YoloHome system	- Completed the description of the YoloHome system - Completed the description of the requirement, including functional and non-functional requirements - Successfully created a server with Adafruit
	- Describe the user require, system requirement	
	- Visualize and create a server with adafruit	
Week 6		
	- Draw the necessary diagram for the system • Use case diagram • Activity diagram	- Created all the necessary diagram for the system. - Integrate IoT gateway with python.
	- Integrate Iot gateway with python	

Table 2: Working plan table (Continued)

Time line	Tasks	Expected result
Week 7	**Midterm**	
	- Develop a UI/UX interface with the use of tool such as figma, Adobe XD or illustrator	- Finished develop the UI/UX interface. - Produce a database schema and the physical design. - Finish the integration for some sensors.
	- Design a database	
	- Setting up the project on github and implementing part of the front end and back end	
	- Integrate sensors together with the Microbit Iot system	
Week 8		
	- Continue in the development of the front end	- Completed a MVP of the front end. - Completed a MVP of the back end. - Completed a MVP of the Iot system.
	- Continuing in the development of the back end	
	- Continuing the development of the Iot system	
Week 9		
	- integration of the system together to have a working MVP.	- Produce a working MVP. - The service could be access via the internet via a cloud service.
	Deployment of the system to the web	
Week 10	MVP demo	
	- Prepare the project for the midterm demo.	- Demo the MVP and receive feedback from the professor.
Week 11		
	- Fix the system based on the lecturer's feedback	- The project is improved based on the professor's feed back.
	- Finalize the system	

Table 2: Working plan table (Continued)

Time line	Tasks	Expected result
Week 12		
	- Fix the project based on the professor's feed back.	- The project is improved based on the professor's feedback. - The project and report is finalized.
	- Finalize the system.	
	- Prepare for final's report	
Week 13	Final project demo	
	- Demo the final completed project.	

4 Teamwork report

4.1 First meeting

Meeting Minutes 1st

Meeting date	Feb 22nd, 2024
Meeting channel	Google meet
Time	20:00pm-21:30pm
Content	Assign work, plan and briefly describe the IoT system.

Table 3: Meeting information

4.1.1 Attendance

Name	**Student's ID**
Đoàn Trương Thanh Tịnh	2153046
Nguyễn Chánh Tín	2153043
Trần Minh Triết	2153057
Lê Việt Tùng	2153083
Nguyễn Quang Thiện	2153994

4.1.2 Meeting discussion

- Discuss about how to divide the task between each member

- Dicuss about the overall aim of the project

4.1.3 Tasks

Task	**Member**	**Due date**
Write descriptions for IoT systems, describing the specific behavior of each application device in the system.	Đoàn Trương Thanh Tịnh	February 29 th, 2024
List and break down system implementation work, assign tasks to each team member.	Nguyễn Chánh Tín	February 29 th, 2024
Summarize the meeting content, take notes and tasks that need to be done during the week	Trần Minh Triết	February 29 th, 2024
Based on the divided work, make a detailed plan each week regarding content, meetings, and time to complete each task	Lê Việt Tùng	February 29 th, 2024
Make a list of devices needed to implement the system, divide output and input devices, and calculate the number of each device	Nguyễn Quang Thiện	February 29 th, 2024

4.2 Second meeting

Meeting Minutes 2nd

Meeting date	March 2nd, 2024
Meeting channel	Google meet
Time	20:00pm-21:00pm
Content	Design, descript use cases, activity of system. Implement simple IoT gateway.

Table 4: Meeting information

4.2.1 Attendance

Name	Student's ID
Đoàn Trương Thanh Tịnh	2153046
Nguyễn Chánh Tín	2153043
Trần Minh Triết	2153057
Lê Việt Tùng	2153083
Nguyễn Quang Thiện	2153994

4.2.2 Meeting discussion

- Discuss about domain context, requirements and the usecases of the system.

- Divide 13 usecase for member to write usecase scenerios and activity diagrams according to them.

4.2.3 Tasks

Task	Member	Due date
Detect user's face, Adjust fan speed, Control lighting and Adjust air conditioner usecases and activity diagrams	Đoàn Trương Thanh Tịnh	March 8 th, 2024
Login, logout, Notify user, View overall Iot devices' usage statistics	Nguyễn Chánh Tín	March 8 th, 2024
Adjust Iot setting, Show current sensor data and Add/remove home setting usecases and activity diagrams	Trần Minh Triết	March 8 th, 2024
Domain context, user reqirements and general usecase diagram	Lê Việt Tùng	March 8 th, 2024
Adjust device timer, alarm, Open door and Validate user's face usecases and activity diagrams	Nguyễn Quang Thiện	March 8 th, 2024

4.3 Third meeting

Meeting Minutes 3^{rd}

Meeting date	March 15^{th}, 2024
Meeting channel	Google meet
Time	20:06pm-21:29pm
Content	Discuss about the techology stack and assign tasks to members.

Table 5: Meeting information

4.3.1 Attendance

Name	**Student's ID**
Đoàn Trương Thanh Tịnh	2153046
Nguyễn Chánh Tín	2153043
Trần Minh Triết	2153057
Lê Việt Tùng	2153083
Nguyễn Quang Thiện	2153994

4.3.2 Meeting discussion

- Remind to fix the domain from last report
- Discuss and choose framework for the UI and database.
- Choose date to receive and test Iot devices.
- Assign tasks to members.

4.3.3 Tasks

Task	**Member**	**Due date**
Iot devices	Đoàn Trương Thanh Tịnh	March 22 th, 2024
Design pattern and technology stacks(frameworks,...) description	Nguyễn Chánh Tín	March 22 th, 2024
UI design and implementation	Trần Minh Triết	March 22 th, 2024
Database design and implementation	Lê Việt Tùng	March 22 th, 2024
Iot devices	Nguyễn Quang Thiện	March 22 th, 2024

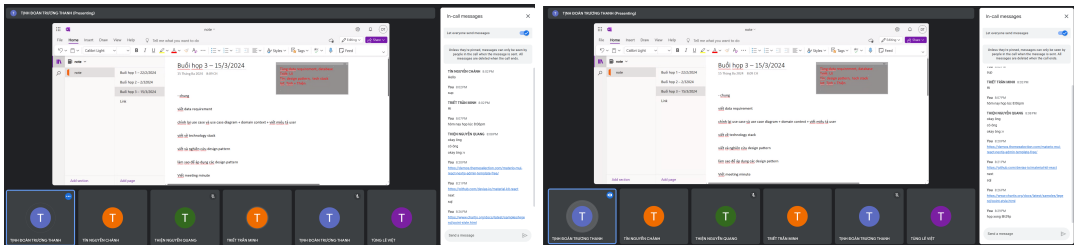


Figure 2: Meeting Time

4.4 Fourth meeting

Meeting Minutes 4^{rd}

Meeting date	March 27^{th}, 2024
Meeting channel	Google meet
Time	20:07pm-20:46pm
Content	Discuss about the implementation for the MVP and assign tasks to members.

Table 6: Meeting information

4.4.1 Attendance

Name	**Student's ID**
Đoàn Trương Thanh Tịnh	2153046
Nguyễn Chánh Tín	2153043
Trần Minh Triết	2153057
Lê Việt Tùng	2153083
Nguyễn Quang Thiện	2153994

4.4.2 Meeting discussion

- Remind to fix as the suggestion of lecturer for last reports
- Divide the MVP into parts to implement.
- Choose date to gather and test the MVP with Iot devices borrowed.
- Assign tasks to members.

4.4.3 Tasks

Task	Member	Due date
Implement Controller Page (Frontend)	Đoàn Trương Thanh Tịnh	March 22 th, 2024
Implement APIs for turning on/off light, fan, adjust fan speed, update and get temperature and humidity of sensor (Backend)	Nguyễn Chánh Tín	March 22 th, 2024
Implement Dashboard to show data(Frontend)	Trần Minh Triết	March 22 th, 2024
Refactor database design and manage the database on Azure Cloud	Lê Việt Tùng	March 22 th, 2024
Design Ohstem and AdafruitIO feeds for Iot devices	Nguyễn Quang Thiện	March 22 th, 2024

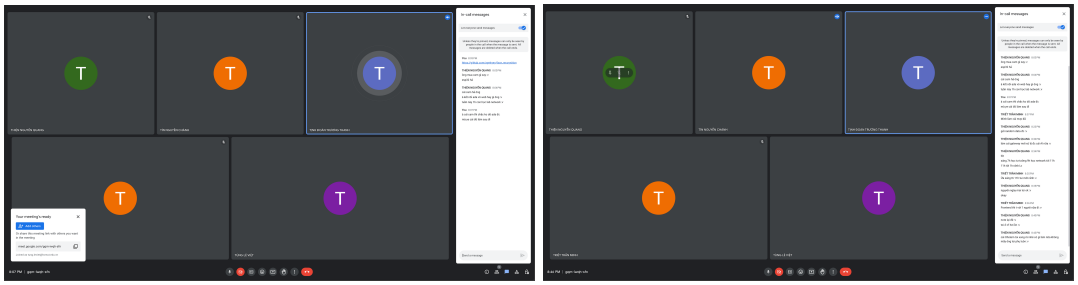


Figure 3: Meeting Time

5 Domain context

As our world continues to develop and society evolves, the issues of security and power consumption become increasingly prominent. While advancements in technology have offered unprecedented convenience and connectivity, they have also introduced new challenges. According to a 2019 report by the FBI, a violent crime occurred every 26.3 seconds, and a property crime offense was committed every 4.6 seconds [1], underscoring the pressing need for robust home security measures to ensure the safety of daily life.

However, alongside concerns regarding security, there exists a parallel apprehension regarding power consumption and sustainability. As the demand for energy escalates with technological proliferation, so does the strain on finite resources and the environment.

Our home guardian system aims to alleviate these issues by incorporating a camera system to monitor the house, promptly reporting any anomalies. Additionally, the system monitors and regulates energy consumption, automatically adjusting usage as necessary. This enhances the user's quality of life, ensuring safer living conditions while also protecting the environment.

6 Design Pattern Research

6.1 Theory

Design patterns are typical solutions to commonly occurring problems in software design. The design pattern is not a specific piece of code, also not about design, but a general concept for solving a particular problem. You can follow the pattern details and implement a solution that suits the realities of your own program. Unlike algorithms, a pattern is a more high-level description of a solution. The code of the same pattern applied to two different programs may be different.

The key features of design patterns:

- **Problem-solution approach**: Design patterns identify common design problems that programmers encounter and provide well-tested solutions that can be reused in similar situations. They essentially act as blueprints for solving specific development challenges.
- **Standardized terminology**: Each design pattern has a unique name that acts as a shorthand for both the problem it addresses and the general solution it offers. This allows developers familiar with design patterns to easily communicate and understand each other's code.
- **Flexibility**: Design patterns provide a general solution that can be adapted to specific situations. They don't dictate rigid code structures, but rather offer a framework that can be customized based on the project's needs.
- **Improved code design**: By leveraging established design patterns, developers can create more maintainable, readable, and flexible code. Patterns often promote good object-oriented practices like loose coupling and dependency injection.
- **Reusability**: A core benefit of design patterns is their reusability. Once you understand a pattern, you can apply it to various scenarios where it fits the problem. This saves development time and effort.

6.2 Types of design patterns

Design patterns differ by their complexity, level of detail and scale of applicability to the entire system being designed. Categorizing by their purpose, there are three main groups of patterns:

- **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code.
- **Structural patterns** explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient.
- **Behavioral patterns** take care of effective communication and the assignment of responsibilities between objects.

Below is 23 design patterns introduced in the book of the Gang of Four (GoF)(1994)[2] . Despite not being the first, the GoF book played a crucial role in popularizing the concept and formalizing many common design patterns in the field of software engineering. ***Note***: Numerous new design patterns have been developed after the GOF's, enriching the field of software engineering.

6.2.1 Creational patterns

- **Abstract Factory**: offers an interface for creating groups of interconnected objects without explicitly specifying their concrete classes.
- **Builder**: decouples the creation of a complex object from its representation, allowing the creation process to generate various representations.
- **Factory Method**: the subclasses determine the instantiation. It is also called as *Virtual Constructor*(like in C++).
- **Prototype**: also known as *Clone*, supports defining the types of objects, using a prototypical instance to create and generate new objects by cloning from this prototype.
- **Singleton**: is one of the simplest design patterns and has only one instance.

6.2.2 Structural patterns

- **Adapter**: enables classes with incompatible interfaces to work together by converting the interface of one class into another interface that clients expect. Another name of the pattern is *Wrapper*.
- **Bridge**: also known as *Handle/Body*, separates an abstraction from its implementation to allow them to change independently.
- **Composite**: or *Object Tree* is a structural design pattern that allows you to create tree-like structures by combining objects, and then interact with these structures as if they were singular objects.

- **Decorator**: may be called as *Wrapper*, enable the dynamic attachment of additional responsibilities to an object without relying on traditional subclassing methods but by placing these objects inside special wrapper objects that contain the behaviors.
- **Facade**: is a structural design pattern that offers a simplified interface to a complex system, such as a library, framework, or a collection of classes.
- **Flyweight**: also called *Cache*, enables efficient utilization of available memory by sharing shared parts of state among multiple objects, rather than duplicating all the data in each individual object.
- **Proxy**: is a structural design pattern that allows replacing or representing an object with a substitute. The proxy manages access to the original object, enabling to perform actions before or after the request reaches the original object.

6.2.3 Behavioral patterns

- **Chain of Responsibility**: also known as *Chain of Command*, is a behavioral pattern where requests are passed through a sequence of handlers. Each handler has the choice to handle the request or pass it to the next handler in the chain.
- **The Command**: also known as *Action* or *Transaction*, transforms a request into a self-contained object that holds all the request information. This enables passing requests as method arguments, delaying or queuing their execution, and facilitating undoable operations.
- **The Interpreter**: enables the interpretation and execution of sentences in a defined grammar or language. It provides an object-oriented approach to representing and processing language rules or expressions.
- **Iterator**: allows iterating over elements of a collection without revealing the internal structure of the collection (such as a list, stack, or tree).
- **Mediator**: also known as *Intermediary* or *Controller*, reduces dependencies between objects by limiting direct communication and enforcing collaboration through a mediator object.
- **Memento**: also known as *Snapshot*, allows you to save and restore the previous state of an object without exposing its internal implementation details.
- **Observer** also known as *Event-Subscriber* or *Listener*, allows you to establish a subscription mechanism for notifying multiple observing objects about events occurring in the object they observe.

- **State**: allows an object to change its behavior based on its internal state. This change gives the illusion that the object has changed its class.
- **Strategy**:allows encapsulating a group of algorithms into individual classes, making them interchangeable.
- **Template Method**: outlines the structure of an algorithm in a superclass. Subclasses can then customize specific steps of the algorithm without altering its overall structure.
- **Visitor**: allows you to decouple algorithms from the objects they act upon.

6.2.4 Relationship Among Design Pattern

Design patterns are commonly used solutions for addressing recurring design problems in software development. While each pattern serves a specific purpose and tackles a particular concern, there are relationships between them. Here are some typical relationships between design patterns:

- **Creational Patterns and Structural Patterns:** Creational patterns focus on object creation mechanisms, such as instantiation and composition. On the other hand, structural patterns deal with organizing and composing classes and objects to form larger structures. These two types of patterns often work together, as creational patterns create objects that are then structured using structural patterns.
- **Behavioral Patterns and Structural Patterns:** Behavioral patterns focus on object interaction and communication. They define patterns for how objects behave in different scenarios. Structural patterns, on the other hand, provide a framework for organizing objects and defining relationships, which are crucial for effectively implementing behavioral patterns.
- **Behavioral Patterns and Creational Patterns:** Behavioral patterns often rely on objects and their interactions, which are created by creational patterns. Creational patterns establish the foundation for creating the necessary objects that behavioral patterns manipulate.

In summary, these relationships highlight the interplay between different types of design patterns and how they can complement and support each other in designing software systems.

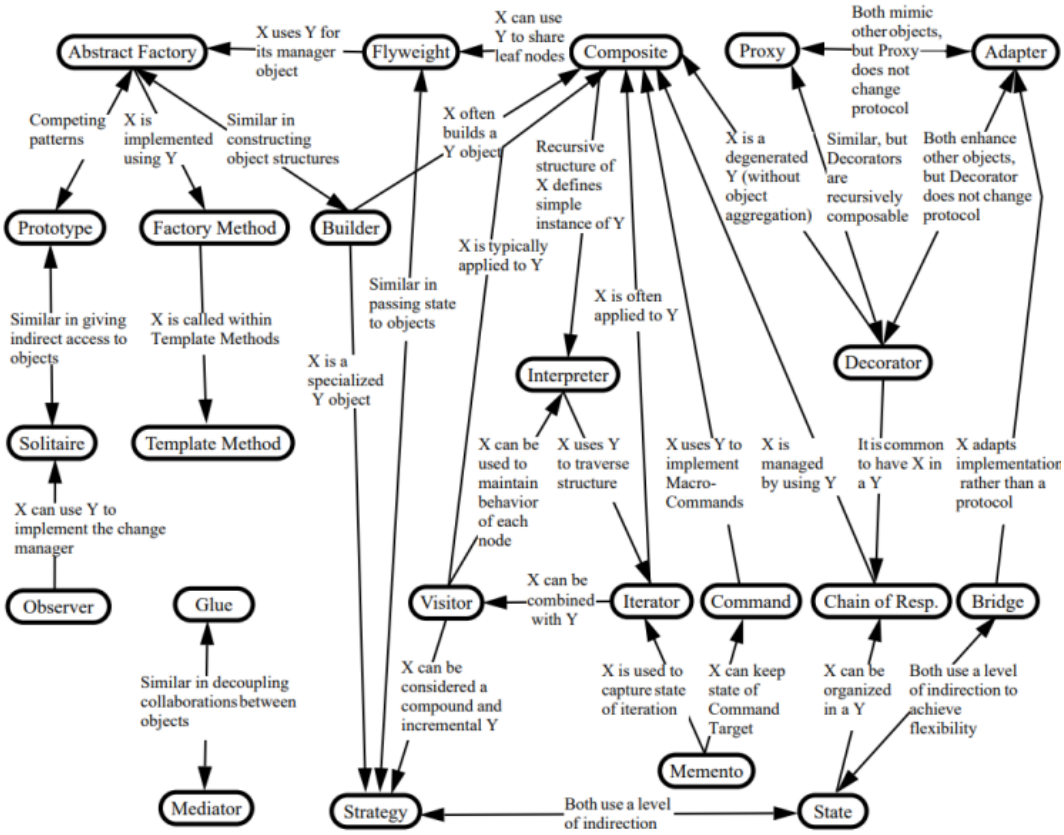


Figure 4: Relationships among design patterns

7 Applying some design patterns to the system

7.1 Singleton

As above we have mentioned, it's one of the most simple design pattern.

- It needs to restrict the number of instance of the class to be only one, providing a global access point to the instance.
- The solitary instance should support extension through subclassing, enabling clients to utilize the extended instance seamlessly without any code modifications.

7.1.1 Benefits

- It ensures a class has only one instance, providing a global access point to that instance.

7.1.2 Drawbacks

- Singletons often juggle instance creation, state management, and access control, violating the *Single Responsibility Principle*.
- The global state of the only instance makes it hard to isolate and test specific code units during unit testing.
- It requires a thread-safe environment or careful measures to prevent race conditions in multi-threaded applications.

7.1.3 Applicability

This pattern is suitable for the configuration manager of the system, a single connection to a database shared by different parts of the system in order to reduce overheads, increasing the performance of the system.[6]

```
// SqlServerDB is a struct as Singleton to hold the connection of sql server on Azure
    Cloud
type SqlServerDB struct {
  DB *gorm.DB
}

var SqlServer *SqlServerDB = nil

func ConnectSqlServerDB() *gorm.DB {
  if SqlServer == nil {
    SqlServer = &SqlServerDB{}
    dsn := "" // link to the database on Azure Cloud
    db, err := gorm.Open(sqlserver.Open(dsn), &gorm.Config{})

    if err != nil {
      panic(err)
    }

    SqlServer.DB = db
  }
  return SqlServer.DB
}
```

7.2 State

The State Design Pattern is a behavioral design pattern that allows an object to alter its behavior when its internal state changes. This pattern is particularly useful when an object's behavior depends on its state, and it can change during the object's life cycle. Some key components of the State Pattern:

- Context: The Context is the class that contains the object whose behavior changes based on its internal state. It maintains a reference to the current state object that represents the current state of the Context.
- State Interface or Base Class: The State interface or base class defines a common interface for all concrete state classes. This interface typically declares methods that represent the state-specific behavior that the Context can exhibit.
- Concrete States: Concrete state classes implement the State interface or extend the base class. Each concrete state class encapsulates the behavior associated with a specific state of the Context.

7.2.1 Benefits

- Ease of Adding New States: With the State pattern, it's easy to add more states for additional behavior. This makes the code more robust, easily maintainable, and flexible.
- Reduced Complexity: The State pattern can help reduce the complexity of conditional statements in your code. By moving state-specific behaviors into their own classes, you can avoid unnecessary if-checks and make your code more readable and easier to manage.

7.2.2 Drawbacks

- Limited Parallelism: State machines, which the State pattern is based on, are not so good at parallelism. If we need to handle multiple states simultaneously, the State pattern might not be the best choice..
- More Code to Write: Implementing the State pattern requires writing a lot of code12. Each state is represented by a separate class, and if there are many states, this can lead to a large number of classes.

7.2.3 Applicability

The State Design Pattern is particularly beneficial in front-end applications like React. React applications are inherently stateful, and managing this state is crucial for the application's performance and user experience. The State Design Pattern can help manage this state effectively, making the application more dynamic and responsive to user input.

Moreover, the pattern improves code organization by separating concerns. This separation makes the code more maintainable and scalable. In React, UI updates are typically driven by state changes, and the State Design Pattern aligns well with this model, allowing us to think about UI changes as state changes.

```
  const Dashboard = () => {
// ** States
const [temperature, setTemperature] = useState(null);
const [humidity, setHumidity] = useState(null);
const [fan_speed, setFanSpeed] = useState(null);
const [light, setLight] = useState(null);
const [openHighTempSnackbar, setOpenHighTempSnackbar] = useState(false); // >60 degrees
const [openWarningTempSnackbar, setOpenWarningTempSnackbar] = useState(false); // >40 degrees but <=60 degrees

const highTempThreshold = 45; // Threshold for high temperature in degrees Celsius
const warningTempThreshold = 35; // Threshold for warning temperature in degrees Celsius


useEffect(() => {
  const fetchData = async () => {
    try {
      const response = await axios.get(`${BackendLink}/users/getDashboardData`, {
        headers: {
          Authorization: localStorage.getItem('SavedToken'),
        },
      });


      const data = response.data;
      setTemperature(data.temperature);
      setHumidity(data.humidity);
      setFanSpeed(data.fan_speed);

      setLight(data.light_level);
      // Trigger the toast if temperature exceeds the threshold
      if (data.temperature > highTempThreshold && data.humidity < 15) {
        setOpenHighTempSnackbar(true);
      } else if (data.temperature > warningTempThreshold) {
        setOpenWarningTempSnackbar(true);
      }

    } catch (error) {
      console.error('Error fetching data:', error);
      // Handle errors gracefully
    }
  };

  fetchData(); // Fetch data immediately for testing

```

```
    const intervalId = setInterval(fetchData, 8000); // Fetch data every 8 seconds

    return () => clearInterval(intervalId); // Cleanup on component unmount
  }, []); // Empty dependency array ensures data is fetched only once on component
    mount

  // Handler to close the snackbar
  const handleCloseSnackbar = (snackbarType) => {
    if (snackbarType === 'high') {
      setOpenHighTempSnackbar(false);
    } else if (snackbarType === 'warning') {
      setOpenWarningTempSnackbar(false);
    }
  };
```

7.3 Adapter

The Adapter pattern enables classes with incompatible interfaces to work together by converting the interface of one class into another interface that clients expect. Another name of the pattern is Wrapper.

7.3.1 Benefits

- The Adapter pattern allows you to separate interface or data conversion code from the main business logic of the program.
- New adapters can be added to the program without modifying existing client code (which may break the working code), as long as they conform to the client interface.

7.3.2 Drawbacks

- The introduction of new interfaces and classes can make the code more complex, and sometimes it may be easier to modify the service class to align with the existing codebase.

7.3.3 Applicability

We apply this pattern for the external APIs outside the main server, which is used to serve the requests from the frontend, it helps the code more clean and it's easier to divide work of the services (the main server and the external APIs) for members.[4]

```
type Strategy interface {
  Execute(des any) error
}

type ExternalService interface {
```

```
  CallAPI(strategy Strategy, des any) error
}

type ExternalServiceAdapter struct {
  adaptee ExternalService
}

func NewExternalServiceAdapter(adaptee ExternalService) *ExternalServiceAdapter {
  return &ExternalServiceAdapter{
    adaptee: adaptee,
  }
}

func (a *ExternalServiceAdapter) Execute(strategy Strategy, des any) error {
  return a.adaptee.CallAPI(strategy, des)
}
type AdaFruitService struct {
  ExternalService
}

func (a *AdaFruitService) Execute(strategy Strategy, des any) error {
  return strategy.Execute(des)
}
type FaceRecognitionService struct {
  ExternalService
}

func (f *FaceRecognitionService) Execute(strategy Strategy, des any) error {
  return strategy.Execute(des)
}
```

We also use Factory Method[5] to make it simple to create an instance to use the external APIs.

7.4 Strategy

The Strategy pattern allows encapsulating a group of algorithms into individual classes, making them interchangeable.

7.4.1 Benefits

- The Strategy pattern allows for dynamic algorithm swapping within an object.
- It facilitates the separation of algorithm implementation details from the code utilizing them.
- Composition can be used instead of inheritance.

- It adheres to the Open/Closed Principle by enabling the introduction of new strategies without modifying the context.

7.4.2 Drawbacks

- If there are only a few infrequently changing algorithms, there is no need to overly complicate the program with additional classes and interfaces associated with the Strategy pattern.
- Clients must possess knowledge of the distinctions between strategies to make appropriate selections.
- Many modern programming languages offer functional type support, allowing for the implementation of various algorithm versions using anonymous functions. These functions can be utilized similarly to strategy objects, without introducing excessive classes and interfaces.

7.4.3 Applicability

We are going to combine Strategy with above Adapter, the Services are wrappers contains the smaller particular services as the strategies.The Strategies is the services for each Device for Adafruit IO Adapter and the facial images processes of Face Recognition Adapter.[7]

```
type LightOn struct {
}

type LightOff struct {
}

type LightLevel struct {
  LightLevel float64
}

type FanSpeed struct {
  FanSpeed float64
}

type FanOn struct {
}
//... Some more Strategies

func (l *LightOn) Execute(des any) error {
    // concrete implementation
}

func (l *LightOff) Execute(des any) error {
    // concrete implementation
```

```
25 }
26 //... Some more Strategies's method
```

8 Architectural Design

8.1 MVC

The MVC (Model-View-Controller) design pattern is a widely used architectural pattern in software development, particularly in building user interfaces for web and desktop applications. It divides an application into three interconnected components, each with its own distinct responsibilities [3]:

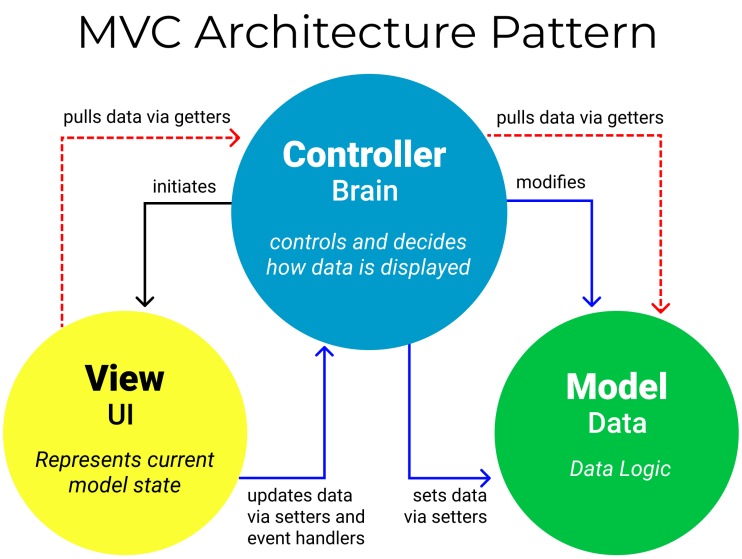


Figure 5: The MVC architecture.

- The **View** or User Interface is responsible for facilitating user interaction with the system. User input is transformed into requests and forwarded to the Controller. Additionally, the View handles the task of displaying content received from the Controller.
- The **Controller** plays a pivotal role in the system by receiving requests from the View, identifying the appropriate data, calling Model methods and returning result to View. Its key function involves categorizing requests and determining the appropriate response, making it essential for the proper functioning of the system.

- The **Model** defines the structure of data and interaction with the database (persistent layer). It serves as the primary component for requesting and receiving data from the Database (or it can be the data requested from the backend). The Model may include elements such as business logic, data abstraction, and is instrumental in designing the Database initially.

The reason we choose to use MVC Model to develop our system frontend website:

- The design is clear and straightforward to put into practice. It breaks down the system (Our frontend) into smaller, well-defined parts, each with its own functions. This division allows tasks to be assigned more clearly and independently.
- Thanks to its segmented structure, the system is easy to troubleshoot and upkeep. This design encourages the reuse of code, and the avoidance of hardcoded elements is a significant benefit.
- This architecture enjoys widespread adoption, making it among the most prevalent and utilized across various applications. Consequently, it's easy to look up tutorials and references online.

8.2 Clean Architecture

Clean architecture is a software design pattern that prioritizes the core business logic of an application. It achieves this by separating the system into layers with well-defined dependencies. The inner layers contain the business rules and entities, independent of any specific technology or framework. Outer layers handle details like databases and user interfaces, and depend on the inner layers. This structure makes the core logic testable, maintainable, and adaptable to future changes in technology [8].

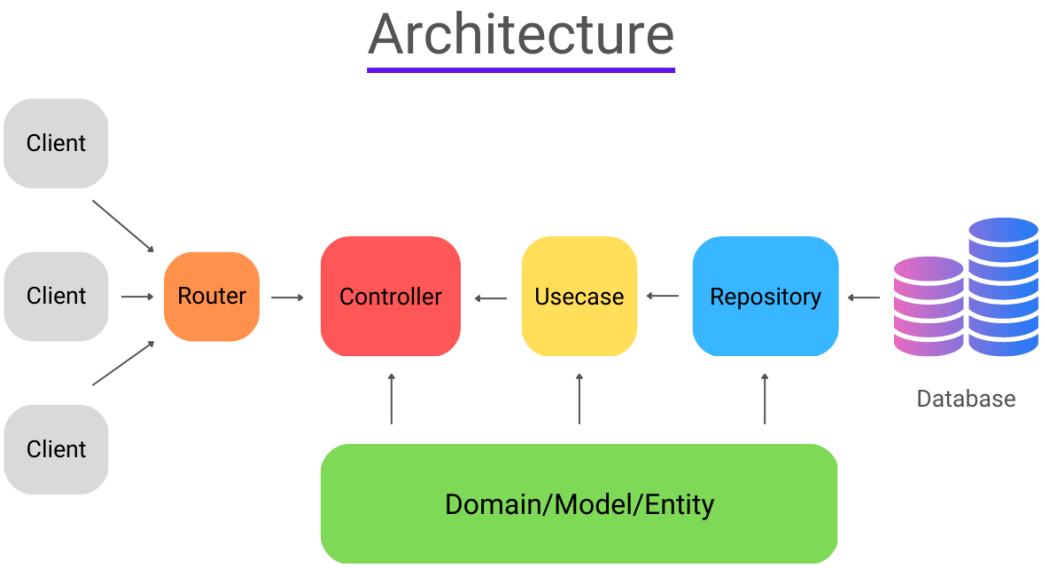


Figure 6: Clean Architecture for Backend

- **Client**: This layer represents the user interface (or frontend), such as a web browser or mobile app. It interacts with the application through the router.
- **Router**: The router receives requests from the client and directs them to the appropriate controller based on pre-defined rules.
- **Controller**: The controller handles incoming requests from the router. It determines what needs to be done and interacts with the usecase layer to carry out the tasks.
- **Usecase**: This layer contains the core business logic of the application. It defines what the application can do, independent of any specific technology or framework.
- **Repository**: The repository layer acts as an interface between the usecase layer and the database. It provides methods to retrieve, store and update data.
- **Database**: This layer stores the application's data.
- **Domain/Model/Entity**: This layer represents the data model of the application. It defines the entities and their attributes that the application works on.

We want to apply **Clean Architecture** for our Backend Implementation, which provides an array of benefits to software applications:

1. **Framework independent**: Easier to replace a package with another package if necessary, since everything is decoupled. For example, we could change the database package that we use or add another one if we need it.
2. **Highly Testable**: Easier to write tests. I have written the test for the usecase, repository, and controller layers.
3. The addition of a new feature becomes easy.
4. Easy to modify the code for any required changes.

9 Technology Stack

9.1 SQL Server

9.1.1 Introduction

Microsoft SQL Server, a prominent relational database management system (RDBMS), is recognized for its robustness, scalability, and feature richness. It caters to a wide range of applications, from large-scale enterprise deployments to mission-critical business systems.

Figure 7: SQL server

9.1.2 Pros:

- **Robustness and Scalability**: SQL Server excels at handling massive datasets and complex transactions, ensuring optimal performance for demanding applications. Its horizontal scaling capabilities enable seamless accommodation of growing data volumes.
- **Feature Rich**: SQL Server offers a comprehensive set of features, including advanced data types, robust security practices, and built-in integration with Microsoft's suite of products.
- **High Availability and Disaster Recovery**: SQL Server prioritizes data availability with features like clustering and replication, ensuring minimal downtime in case of failures.
- **Performance Optimization**: SQL Server is equipped with advanced query optimization tools and indexing mechanisms, guaranteeing efficient data retrieval and manipulation.
- **Familiarity for Windows Users**: For those accustomed to the Windows environment, SQL Server's integration and management tools provide a familiar and user-friendly experience.

9.1.3 Cons:

- **Cost**: Unlike other open source DBMS, SQL Server requires licensing fees, which can be a significant factor for cost-sensitive projects.
- **Platform Dependence**: Traditionally, SQL Server has primarily focused on the Windows Server platform, although limited support for Linux environments exists.
- **Complexity**: Due to its feature-rich nature, SQL Server administration can have a steeper learning curve compared to simpler database solutions.

SQL Server stands out as a powerful and reliable RDBMS solution for demanding applications that require robust functionalities, high scalability, and tight integration with Microsoft technologies. While licensing costs come into play, the feature set and stability it offers can be advantageous for complex enterprise systems. In spite of its limits, we chose SQL Server due to its high performance, robust functionality, and well-established industrial processes for deployment and management. It will give us a good production experience for the future work.

9.2 Next.js

9.2.1 Introduction

We've opted to utilize Next.js for our frontend development. Next.js stands out as a widely-used React framework, streamlining the creation of robust and scalable web applications. It provides a structured methodology for constructing web applications, boasting functionalities such as automatic code splitting, server-side rendering, and streamlined routing.

Figure 8: Next-js

9.2.2 Pros:

- **Efficient Routing**: NextJS simplifies routing, making it easy to create dynamic pages and navigate between them.
- **Server-Side Rendering (SSR)**: Provides SSR out of the box, enhancing performance and search engine optimization (SEO).
- **Static Site Generation (SSG)**: Enables the generation of static HTML files at build time for faster loading and reduced server load

9.2.3 Cons:

- **Dynamic Imports Complexity**: Although NextJS simplifies code splitting, dynamic imports can sometimes be tricky to manage, and developers might face challenges when dealing with asynchronous module loading.
- **Server-Side Rendering (SSR) Overhead**: While SSR improves performance, it can introduce

higher server-side loads and might not be necessary for all applications, especially if they are primarily client-rendered.

- **Integration Challenges**: Integrating certain third-party libraries or tools may require additional configuration or workarounds, especially if they are not explicitly designed for use with NextJS.

9.3 Golang

9.3.1 Introduction

Go, a simple and beginner-friendly programming language by Google, is compiled for speed and known for its focus on concurrency. This makes it powerful for building applications that handle many tasks at once, perfect for web development and large systems. Go's features like simplicity, static typing, and focus on interfaces align well with the separation of concerns principles in Clean Architecture. This makes Go a popular choice for developers who want to build applications with a clean and maintainable architecture.

Gin is a web framework written in Go (Golang). It features a martini-like API with much better performance, up to 40 times faster thanks to httprouter. If you need performance and good productivity, you will love Gin.

9.3.2 Pros:

1. Routes grouping: Organize your routes better. Authorization required vs non required, different API versions... In addition, the groups can be nested unlimitedly without degrading performance.

2. Error management: Gin provides a convenient way to collect all the errors occurred during a HTTP request. Eventually, a middleware can write them to a log file, to a database and send them through the network

3. Middleware support: An incoming HTTP request can be handled by a chain of middlewares and the final action. For example: Logger, Authorization, GZIP and finally post a message in the DB.

9.3.3 Cons:

Importing (often very large) codebases that we don't understand all, that our infrastructure is now dependant on a package that other people are maintaining. They can introduce way more complexity into our project and slow down development time.

9.4 Connect Microbit devices

In order to connect and control the microbit devices we using some libraries below in Ohstem.

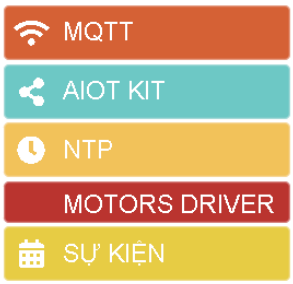


Figure 9: Using MQTT, AIOT KIT, and EVENTS

Furthermore, to update data into Adafruit server we need to create a Connect wifi function that uses the same wifi as the laptop which connects with the Mircrobit devices.

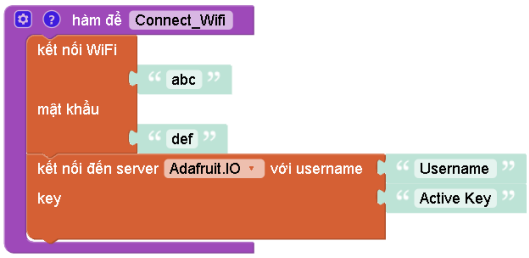


Figure 10: Connect Wifi fucntion

9.4.1 RGB LED

4 LED RGB module (Red - Green - Blue) with 3 color bases are used to distribute many colors in reality. The RGB LED device serves the aim of turning ON/OFF the lights in home by using the web.

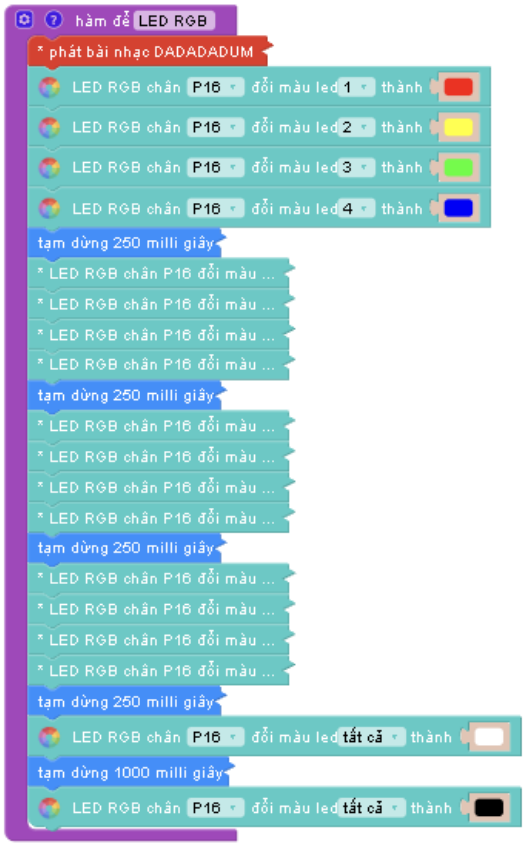


Figure 11: LED RGB function for connect successful with P16 foot

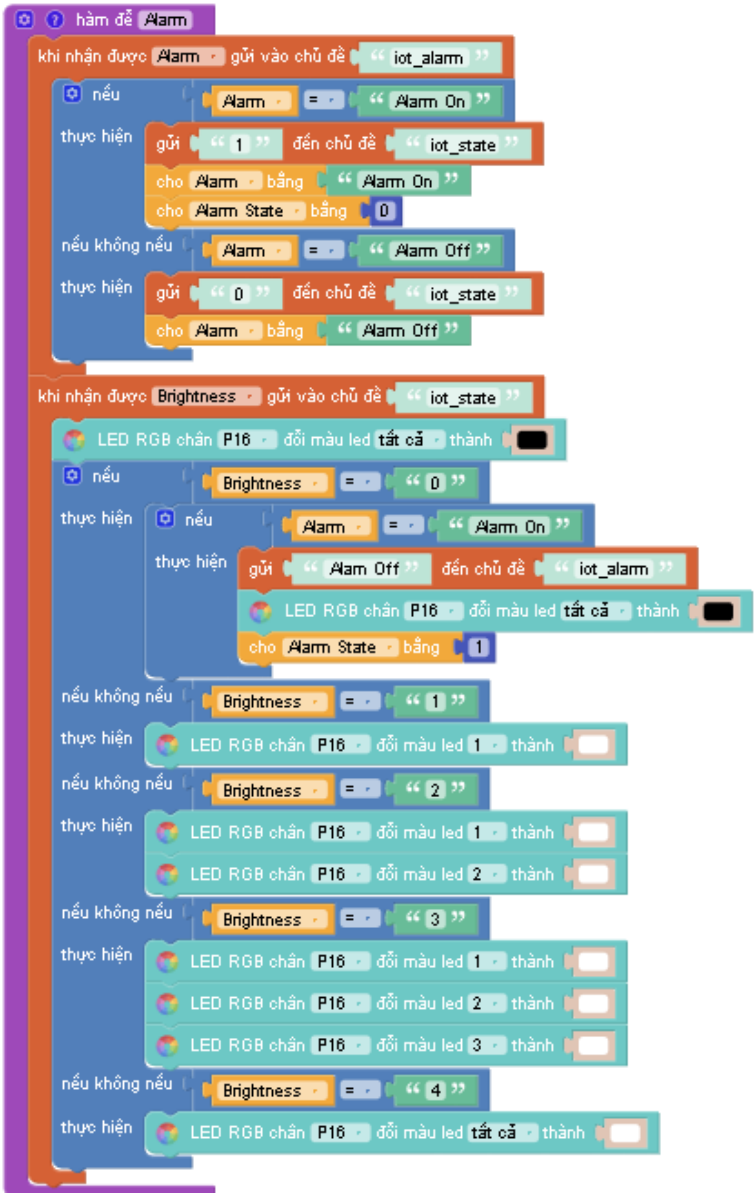


Figure 12: Control Alarm function

9.4.2 Air Temperature,Humidity sensor DHT20 and Light sensor

DHT20 is a common sensor for civil applications, with the advantage of low price and simple use. In some cases, DHT20 can also be used for some food dryers. The sensor's measuring range is quite suitable in normal environments without major fluctuations, with humidity in the range of 20 - 90 %

and temperature of 0 - 50°C. This sensor device serves the aim of turning ON/OFF the fan in home by using the web or automating. Furthermore, this project has light sensors to notice the brightness in home.

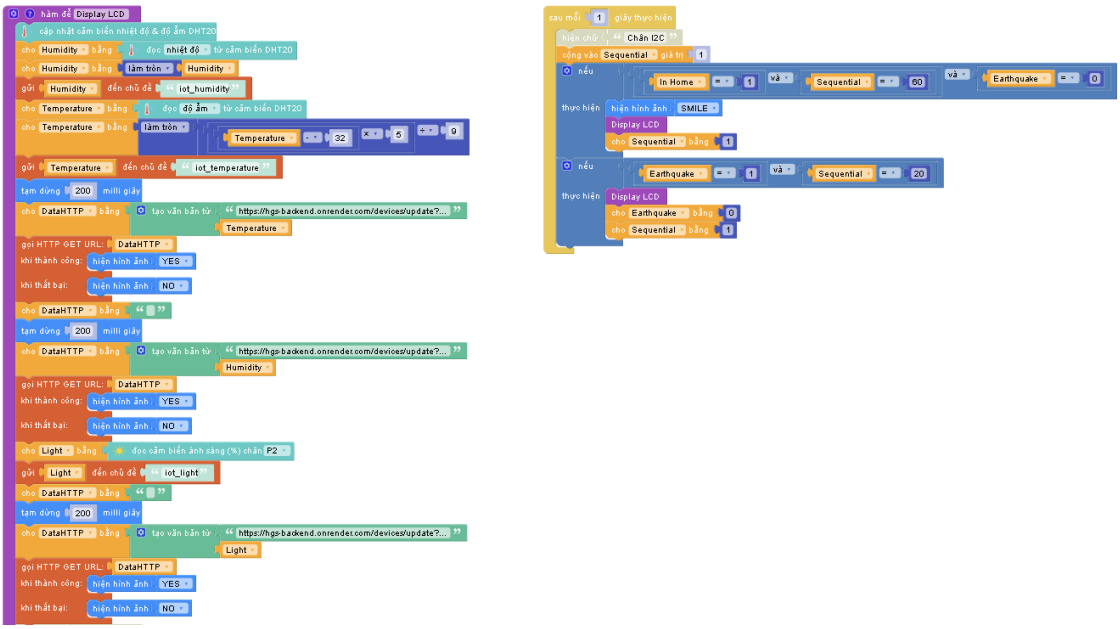


Figure 13: DHT20 Device connects with I2C foot and read data automation after each 60 seconds

9.4.3 Remote and Infrared Receiving Eye

Its working concept is comparable to that of household remote controls, such as those for TVs or air conditioners, for illustration. For a reliable and precise connection, one must aim the remote at the receiver while utilizing it. With its low reflectance, infrared wavelengths are directed waves. Thus, the likelihood of getting a signal is relatively low when it is not aimed squarely at the receiving eye. This sensor device serves the aim of OPENING the door in home by using the remote.

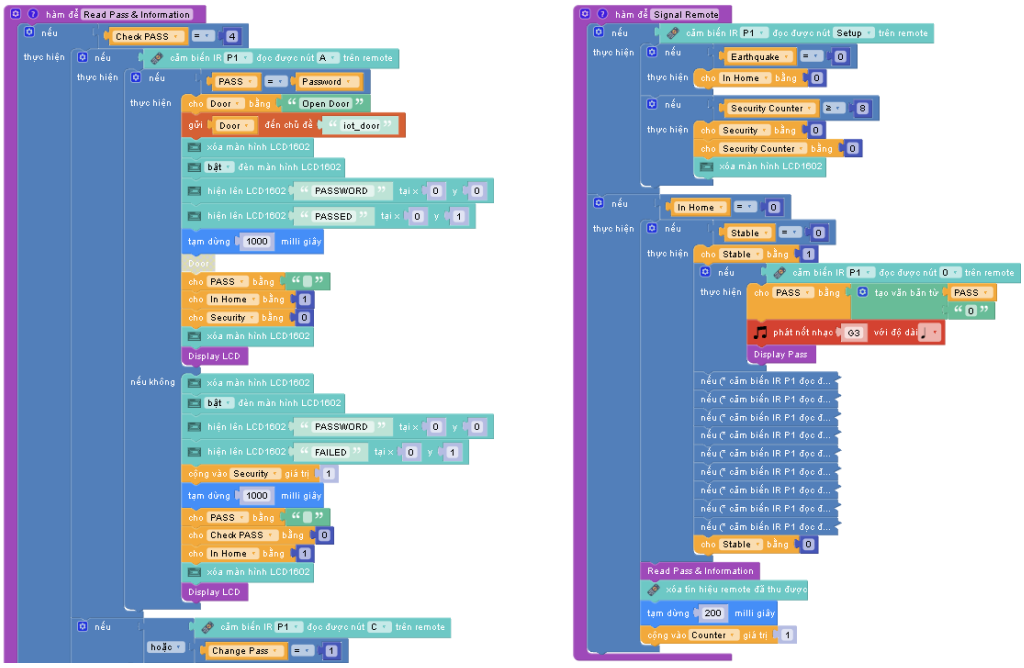


Figure 14: Remote and Infrared Receiving Eye functions_1

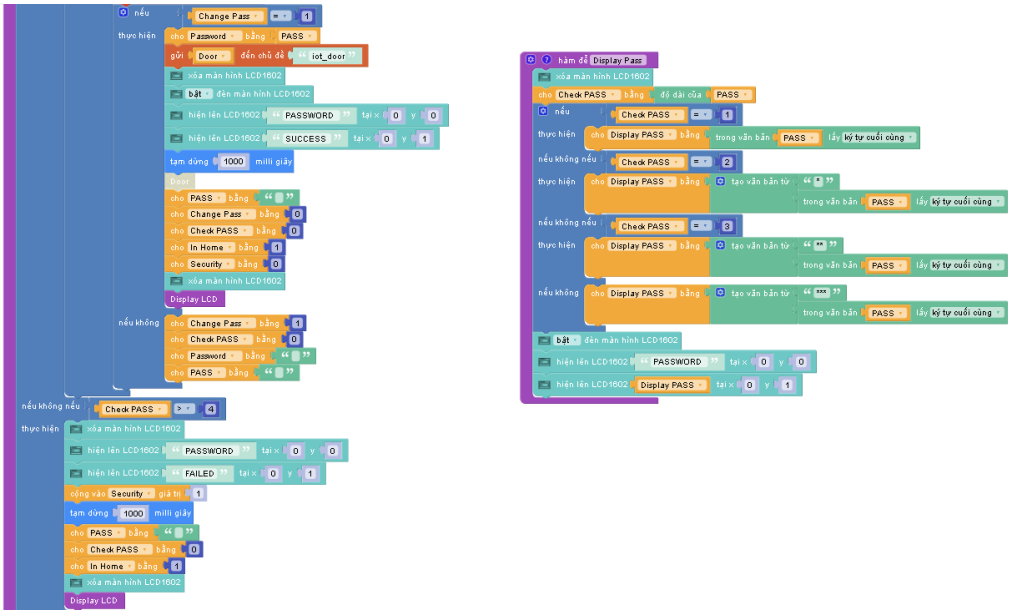


Figure 15: Remote and Infrared Receiving Eye functions_2

9.4.4 Fan control

Fans are a necessary device for smart home applications. Further integration with automatic or voice control features, we will have a compelling application led in the house. In this tutorial, we will use the Mini Fan device to illustrate controlling a fan motor. With the use of motors, we can control the fan speed depending on the purpose of the application.

This mini fan device uses a DC motor, also known as a DC (Direct Current) motor. Unlike devices that only need to be turned on and off (such as light bulbs), fans are devices that need to have their speed controlled.

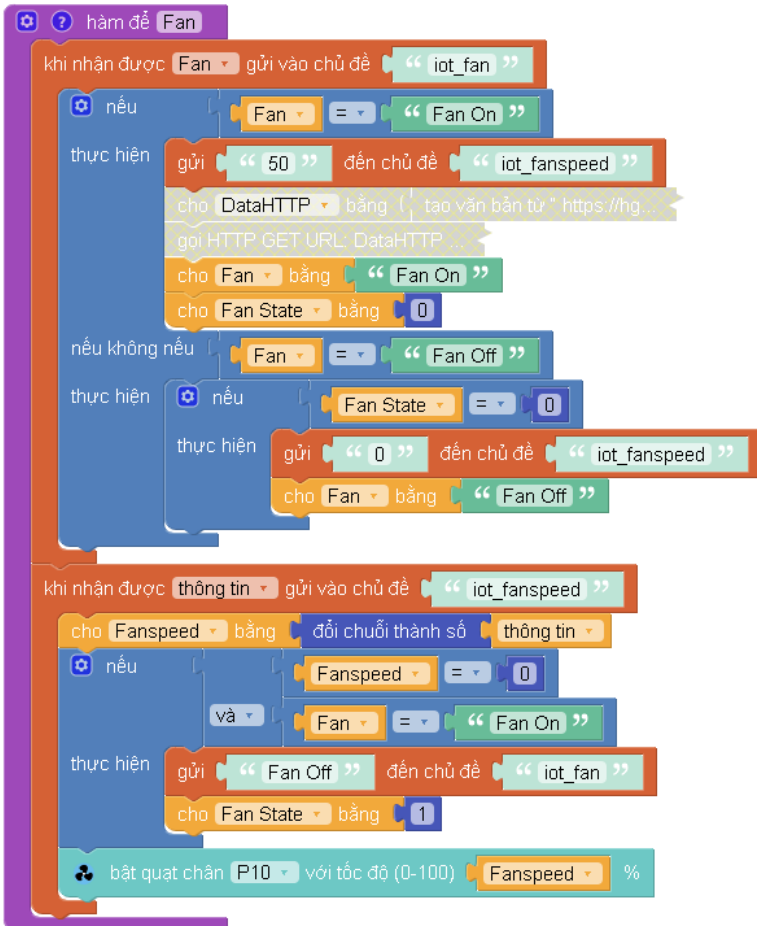


Figure 16: Control fan function

It should be noted that at small speeds, the rotational force will not be enough to make the propeller

move. This is static friction often presented in Physics. At a speed greater than 30% or even greater, the propeller begins to rotate. At that time, the force from the engine is enough to overcome its own static friction force and make the propeller rotate.

9.4.5 Door control (control through servo)

In the case of the door, my team will use servo motors to simulate the door latch. When the servo is at 90 degrees, it indicates the door is closed and below 90 degrees, the door is open.

Figure 17: Control servo function

Completely opposite to propeller motors, RC Servo has low speed but quite strong force. Inside the structure of RC Servo, it often comes with a reduction gearbox with a large transmission ratio. Therefore, just a small force on the motor side can drive a large force on the transmission mechanism.

9.4.6 Control input and output device

The yolobit circuit control code has 2 streams. One thread is continuously updated with the latest signal from adafruit to control output devices. The second is the stream that takes the signals of the sensors and sends them to the server and displays them on the dashboard for users to monitor. Below is the implementation of the above two streams. (Note the need to initialize the variables used)

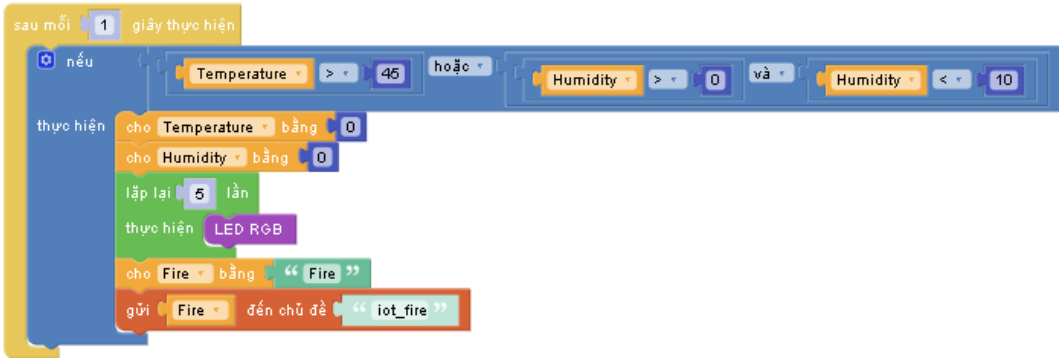


Figure 18: Warning function when having fire

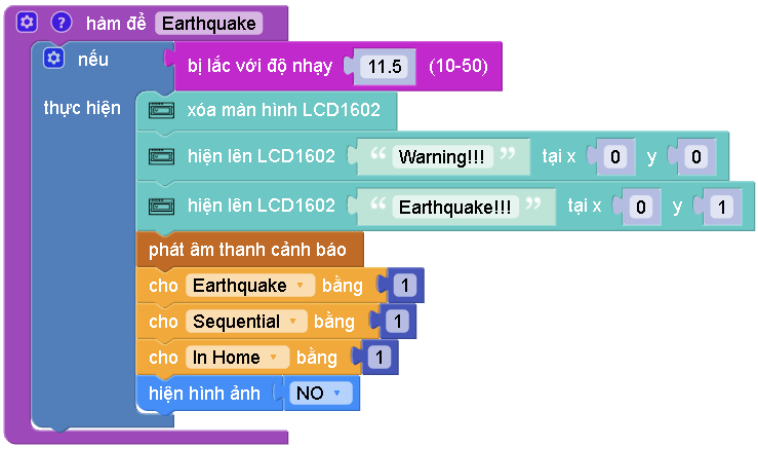


Figure 19: Warning function when having earthquake

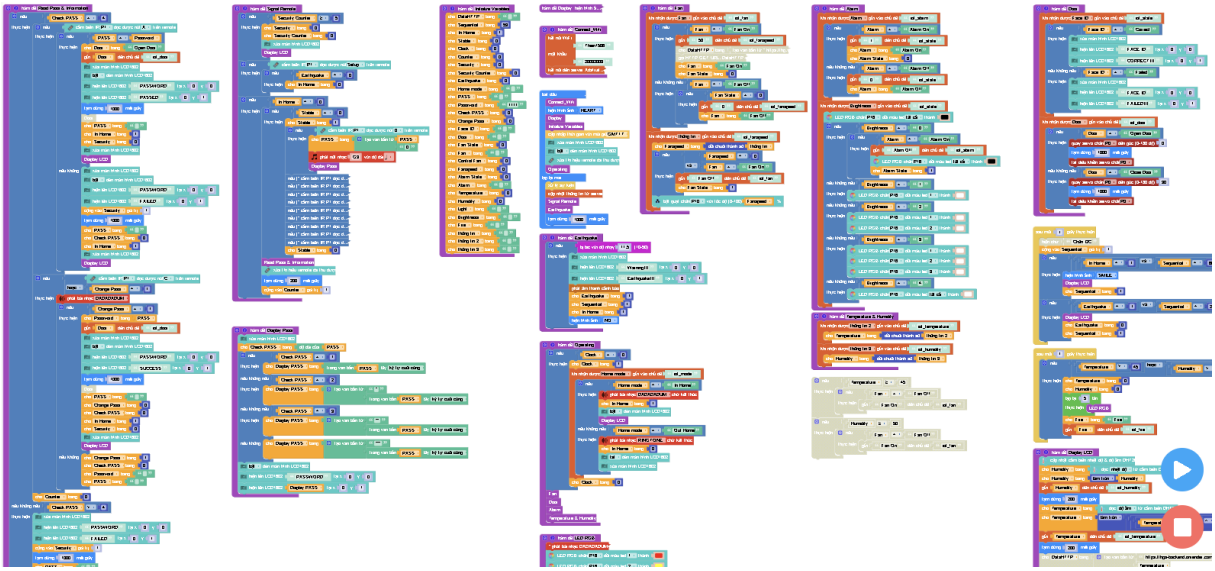


Figure 20: Control Yolo-bit and AIOT devices in Ohstem

10 Requirement enginerring

10.1 Functional requirement

- The User shall be able to login and logout of the system via an authentication provider
- The user shall be able to view a dashboard with all the statistic of their usage of the iot system.
- The user shall be able to view the data of the current sensor.
- The user shall be able to adjust the functioning of iot module such as fan, lightning, air conditioner temperture and the timing to automatically turn on the aforementioned devices.
- The user shall be able to set up a default home profile setting which will contain all the user 's default adjustment for the sensor.
- The iot system should be able to detect when a user is inside the house.
- The iot system shall be able to validate the user before allowing entry via face detection.
- The iot system shall notify user if there is five or more invalid entry in a row.

10.2 Non functional requirement

- The login and authorization process should not take more than 5 seconds under ideal condition.
- The UI should be understandable and friendly, It should follow the material UI design standard.
- The application shall be a web-based applicaiton.
- The validation of the user's face should not take more than 2 seconds under ideal conditions in full sun light.
- The time gap between when the user adjusts the iot setting and when the setting takes effect should not be longer than 2 seconds.
- The Application shall be available at all times.
- All IoT devices are replaceable and upgradable if needed.

10.3 Data requirement

1. The system must support long-term and persistent storage for IoT devices and their log.
2. The system shall have different access levels for different users.
3. The password of the user must hashed before storing it in the database.
4. Every change that the user makes on the IoT device shall be logged.
5. The sensor data shall be logged into the database every minute by taking the average number of the data over that minute.
6. Data stream for IoT devices sent to the cloud must be sent via the MQTT protocol.

11 Use-case

11.1 Use case diagram

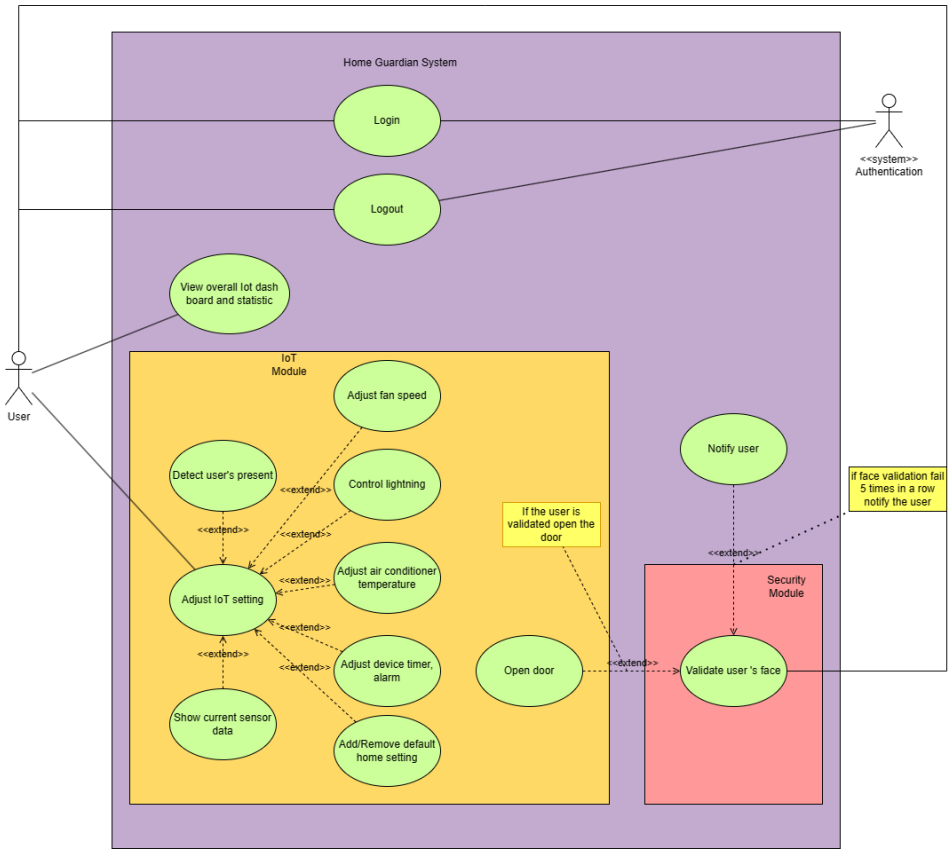


Figure 21: Overall use case diagram

11.2 Description use case

11.2.1 Usecase 1: Login/

User-case name	**Login**
Actor	Home users & Authentication system
Description	Users logging into the web application of the system
Preconditions	The server of the system is available
Postconditions	Users can view the Iot devices' usage statistics or adjust their setting
Trigger	Input username and password and click "Login" button (Login)
Normal Flow	Login: 1. User accesses to the website 2. User inputs username and password 3. User clicks "Login" button and send the request to the server 4. The system confirms the authentication in at most 5 seconds 5. The browser redirects to the dashboard
Exceptions	Login: 1. The server of the system is not available 2. Wrong username or password
Alternative Flows	None

11.2.2 Usecase: Logout

User-case name	**Logout**
Actor	Home users & Authentication system
Description	Users logging out of the web application of the system
Preconditions	The server of the system is available
Postconditions	Users return to login page
Trigger	Press "Logout" button on the navbar (Logout)
Normal Flow	Logout: 1. User clicks the "Logout" button 2. The browser redirects to Login page
Exceptions	Login: 1. The server of the system is not available 2. Wrong username or password
Alternative Flows	None

11.2.3 Usecase 2: View the Iot system's overall usage statistics

User-case name	**View overall usage statistics of the Iot devices on the dashboard**
Actor	Home users & System
Description	Users view overall statistics for usage of Iot devices on the dashboard of the web application
Preconditions	User has already logged in
Postconditions	User can view the statistics of the usage the Iot devices recently
Trigger	After users logged in or when clicking to "Dashboard" on the navbar,it redirects to the dashboard
Normal Flow	1. User logs in or clicks "Dashboard" on the navbar(logged in) 2. The browser send request to the system 3. System responds the results to the client 4. The web app renders the data into visualized graphics
Exceptions	1. The server of the system is not available
Alternative Flows	None

11.2.4 Usecase 3: Notify user

User-case name	**Notify user**
Actor	Home users & Face detector & System
Description	The iot system shall notify user if there is five or more invalid entries in a row
Preconditions	The face detector identifies 5 or more invalid entries consecutively
Postconditions	A notification will be sent to the user
Trigger	Facial recognition device detects 5 or more strange people try to enter the house
Normal Flow	1. The face detector recognizes 5 or more unregistered faces of people who try to open the door 2. The device sends request to the system's server 3. The server sends a notification to the user's web client 4. The user responds to the system to control the device to ignore or block the entries
Exceptions	1. The server of the system is not available 2. The face detectors breaks down
Alternative Flows	None

11.2.5 Usecase 4: Adjust device timer and alarm

User-case name	**Adjust device timer and alarm**
Actor	Home users
Description	Allowing users to set a time for some crucial tasks and the alarm will ring with some specific ringtones.
Preconditions	Logged into the web
Postconditions	The time is set and the button for the timer is turned ON and the alarm will ring at the time setting
Trigger	Adjust the timer and press the "Alarm" button
Normal Flow	Adjust the taskbar timer Press the "Alarm" button The alarm will ring at the preset time
Exceptions	None
Alternative Flows	None

11.2.6 Usecase 5: Adjust Door operating and Validate users' face

User-case name	**Adjust Door operating and Validate users' face**
Actor	Home users
Description	Controlling the door operation automated, interacting by using the web or face recognition
Preconditions	Logged into the web
Postconditions	The state of the door is ON or OFF on the web rely on the demand of home users
Trigger	Press the "Door" button if don't use the face recognition
Normal Flow	1: Press the "Door" button 2: Use face recognition General objective: Close/Open the door
Exceptions	None
Alternative Flows	None

11.2.7 Usecase 6: Adjust IoT system setting

User-case name	**Add/Remove home setting presets**
Actor	Home users
Description	User change the setting of devices the the IoT to their likeing
Preconditions	User has already logged in
Postconditions	The setting the user wanted to change is changed
Trigger	After users logged in and clicking the System setting option on the navbar
Normal Flow	1. User logged in 2. The user change to the Control tab 3. User select the setting they want to change 4. Browser send command to IoT system 5. Web app notify that the changes have been made
Exceptions	1. Cannot connect to the system to change the setting
Alternative Flows	None

11.2.8 Usecase 7: Show current sensor data

User-case name	**Show current sensor data**
Actor	Home users
Description	User check the value that the sensor send back
Preconditions	User has already logged in
Postconditions	The web app show the information of all sensors data
Trigger	After users logged in and clicking the Dashboard option on the navbar
Normal Flow	1. User logged in 2. The user change to the Dashboard tabs 3. Browser send request to the system 4. System accept the request and send the data back 5. Web app show the sensor data
Exceptions	1. The server of the system is unavailable
Alternative Flows	None

11.2.9 Usecase 8: Add/remove home setting preset

User-case name	**Add/Remove home setting presets**
Actor	Home users
Description	User can add or remove a preset to make the system act the same way when something trigger
Preconditions	User has already logged in
Postconditions	The preset the user want to add/remove is added/removed
Trigger	After users logged in and clicking the Control tab and select the Preset setting
Normal Flow	Add preset: 1. User change to Control tab, select Change Preset button 2. User select add preset in the Preset menu 3. User define the setting for the preset 4. User save the preset to the system 5. System prompt preset saved successfully Remove preset: 1. User change to Control tab, select Change Preset button 2. User select a preset from the preset list 3. User select delete button and confirm 4. System prompt that the preset has been deleted
Exceptions	1. Cannot connect to the system to change the setting
Alternative Flows	None

11.2.10 Usecase 9: Detect user's present

User-case name	**Detect user's present**
Actor	Home users and detection system
Description	Detect the present and absent of user to adjust fan, air-conditioner, light.
Preconditions	User has already gone in home or gone out.
Postconditions	Open and adjust the appropriate mode when the user is present. And turn it all off when the user leaves the house.
Trigger	User goes in/out home.
Normal Flow	Detect present: 1. Turn on light, air conditioner 2. If the temperature is so high, turn on fan and lower the air conditioner temperature 3. If it is so cold, increase air conditioner temperature Detect absent 1: Turn of light, fan, and air conditioner
Exceptions	User can overwrite the automation of detection by setting on app.
Alternative Flows	None

11.2.11 Usecase 10: Adjust fan speed

User-case name	**Adjust fan speed**
Actor	Home users
Description	User sets speed of fan, time, on/off
Preconditions	User has already logged in
Postconditions	Speed of fan is changed, fan on/off at the modified time
Trigger	User log in and go to setting
Normal Flow	Setting speed 1. log in 2. Input fan speed want to set 3. Confirm the speed Set time on/off 1: Log in 2: Input time to set on/off 3: Confirm the time
Exceptions	None
Alternative Flows	None

11.2.12 Usecase 11: Control lightning

User-case name	**Control lightning**
Actor	Home users
Description	User sets time, on/off light
Preconditions	User has already logged in
Postconditions	Light on/off at the modified time
Trigger	User log in and go to setting
Normal Flow	Turn on/off 1. log in 2. Click button on/off Set time on/off 1: Log in 2: Input time to set on/off 3: Confirm the time
Exceptions	None
Alternative Flows	None

11.2.13 Usecase 12: Adjust air conditioner

User-case name	**Adjust air conditioner**
Actor	Home users
Description	User sets temperature of air conditioner, time, on/off
Preconditions	User has already logged in
Postconditions	Temperature of air conditioner is changed, it on/off at the modified time
Trigger	User log in and go to setting
Normal Flow	Setting temperature 1. log in 2. Input temperature want to set 3. Confirm the speed Set time on/off 1: Log in 2: Input time to set on/off 3: Confirm the time
Exceptions	User can overwrite the automation of detection by setting on app.
Alternative Flows	None

12 Activity Diagram

12.1 Login/Logout

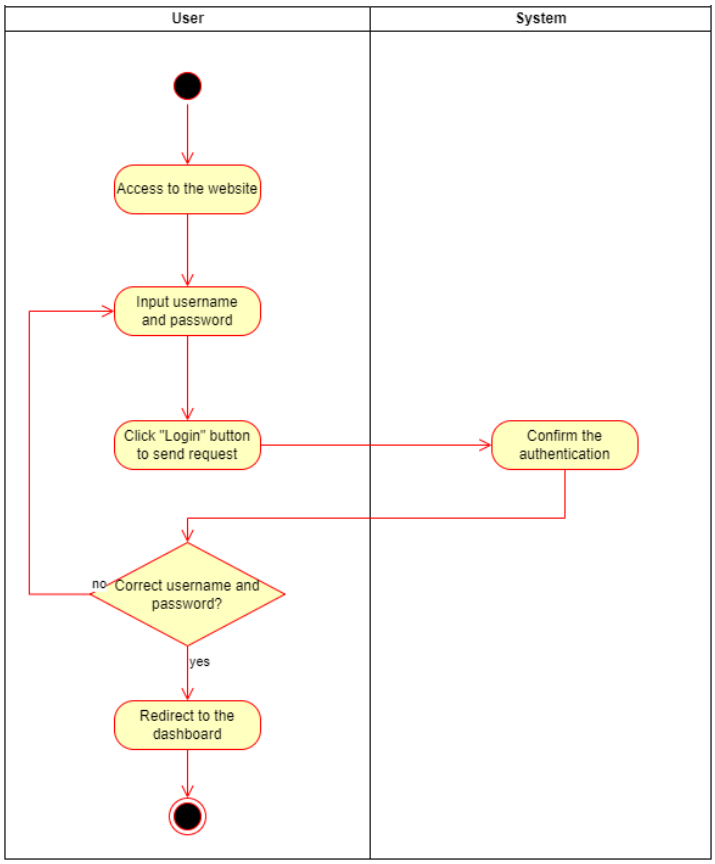


Figure 22: Activity Diagram of Login

- **Step 1:** User gets access to the Web.
- **Step 2:** User inputs username and password.
- **Step 3:** User clicks "Login" button to send authentication request to the system.
- **Step 4:** The system confirms the authentication.
- **Step 5:** It Redirects to the dashboard if username and password are both correct. Otherwise, it returns to step 2.

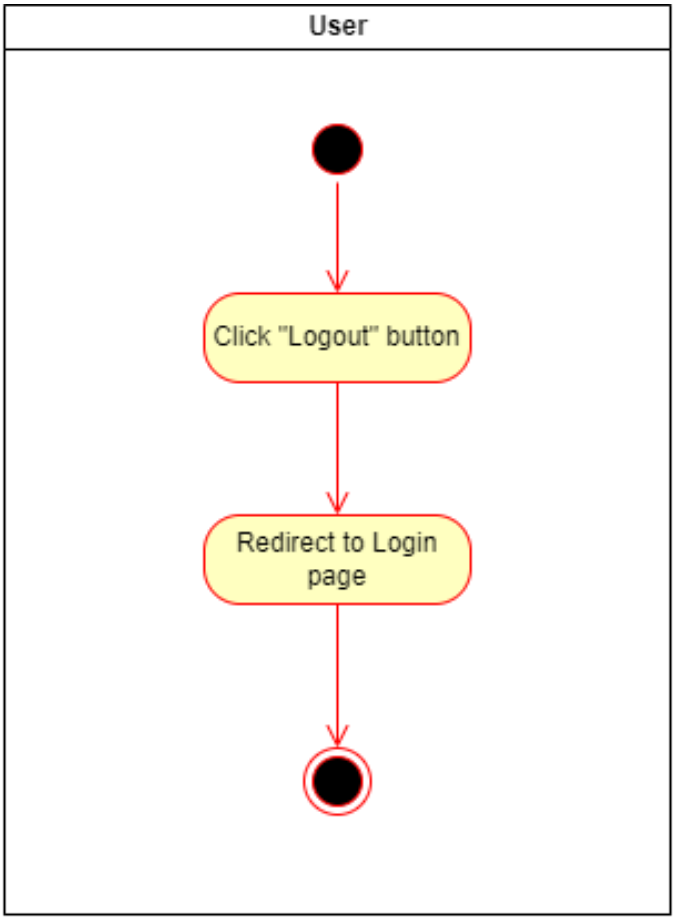


Figure 23: Activity Diagram of Logout

- **Step 1:** User clicks the "Logout" button on the navbar.
- **Step 2:** It redirects to the Login page.

12.2 View the Iot system's overall usage statistics

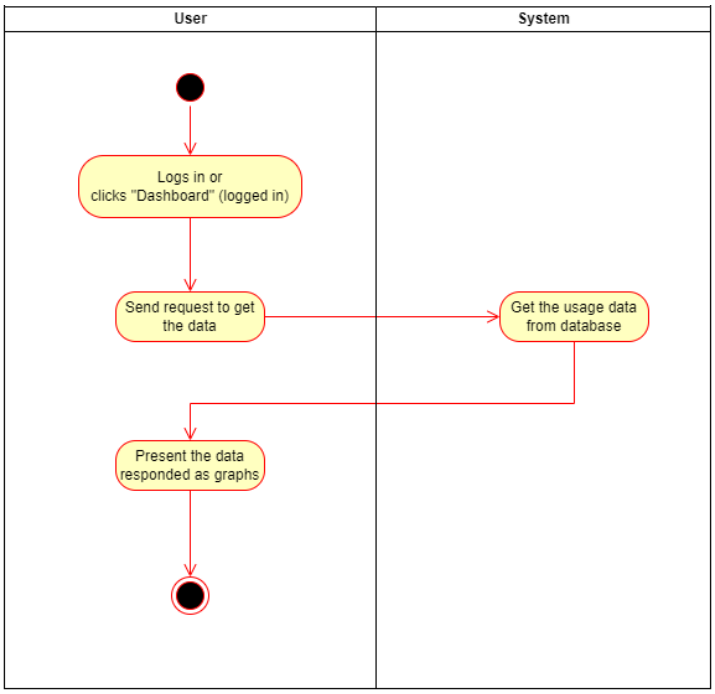


Figure 24: Activity Diagram of View overall usage statistics of Iot devices

- **Step 1:** User logins to the system or click "Dashboard" on the navigation bar when logged in.
- **Step 2:** The client sends request to the system to get data about usage of Iot devices.
- **Step 3:** The system responds the data to the client.
- **Step 4:** The web application presents the data responded into visualized graphics.

12.3 Notify user

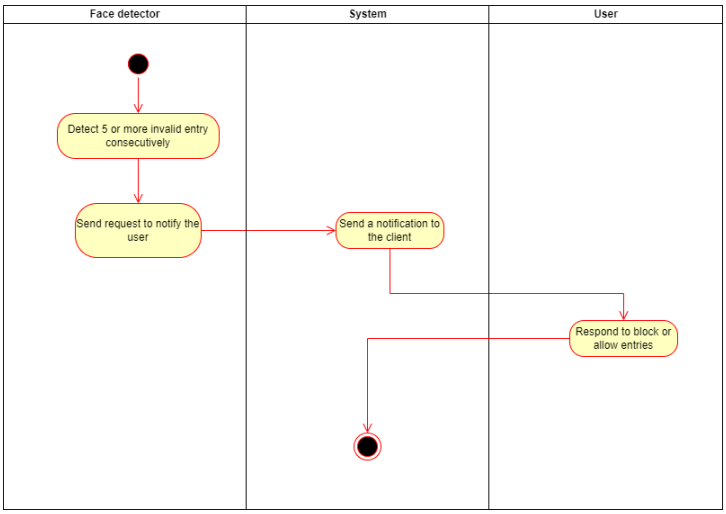


Figure 25: Activity Diagram of Notify user when too many invalid entry at the same time

- **Step 1:** Face detector identifies 5 or more people whose faces are unregistered trying to enter the house.
- **Step 2:** The device sends request to the system to notify the user.
- **Step 3:** The system send a notification to the user to ask if he/she wants to allow or block the entries.
- **Step 4:** The user responds to the Iot system to control the devices to allow or block the invalid entries.

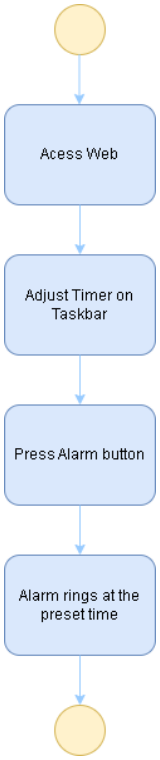


Figure 26: Activity Diagram of Timer device and Alarm

- **Step 1:** Accessing the Web.
- **Step 2:** Adjusting the taskbar timer.
- **Step 3:** Press the "Timer" button.
- **Step 4:** Alarm rings at the preset time.

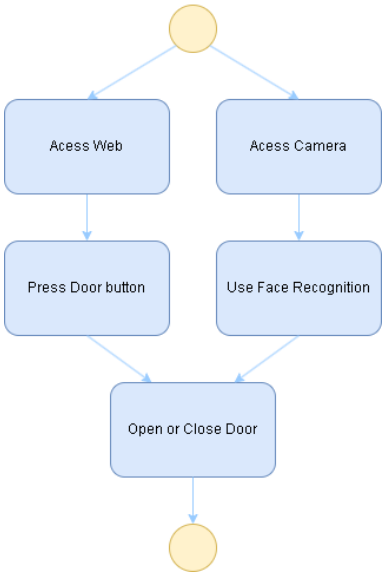


Figure 27: Activity Diagram of Door operating and Validate users' face

- **Step 1:** Accessing the Web or accessing the camera.
- **Step 2:** Press the "Door" button or use face recognition.
- **Step 3:** Open or Close the Door rely on the demand of home users.

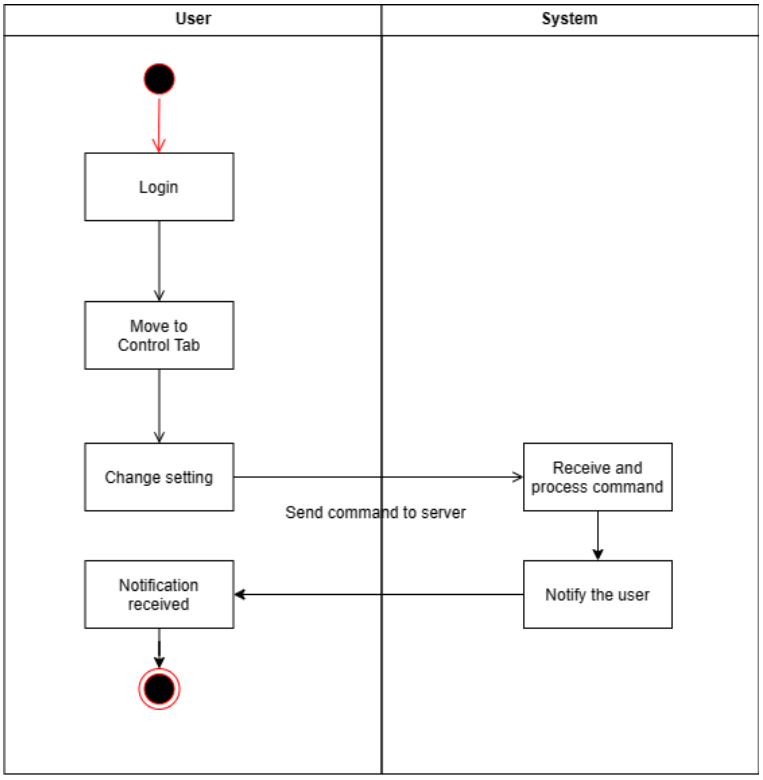


Figure 28: Activity Diagram for Adjust IoT setting

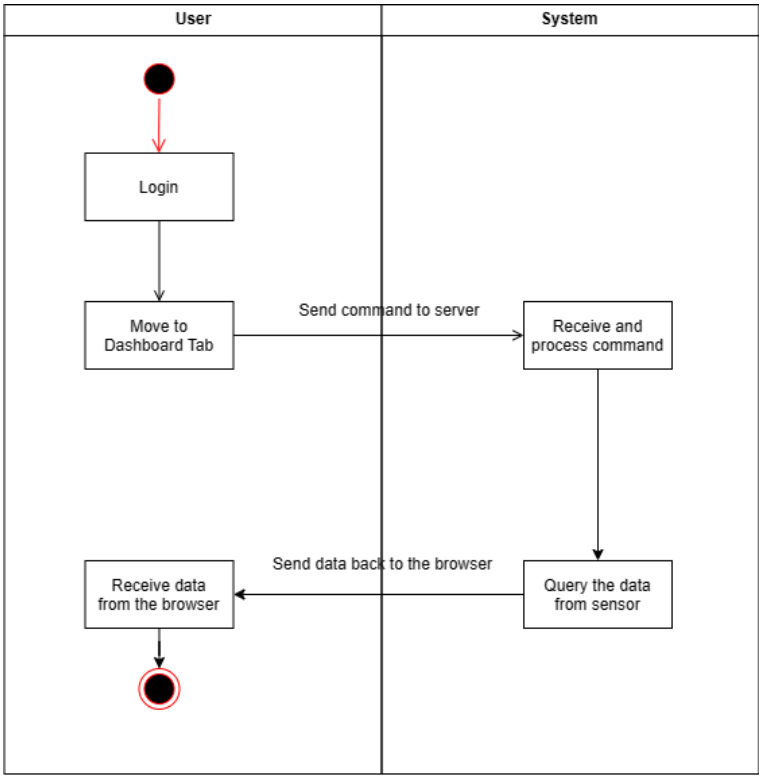


Figure 29: Activity Diagram for Show current sensor datag

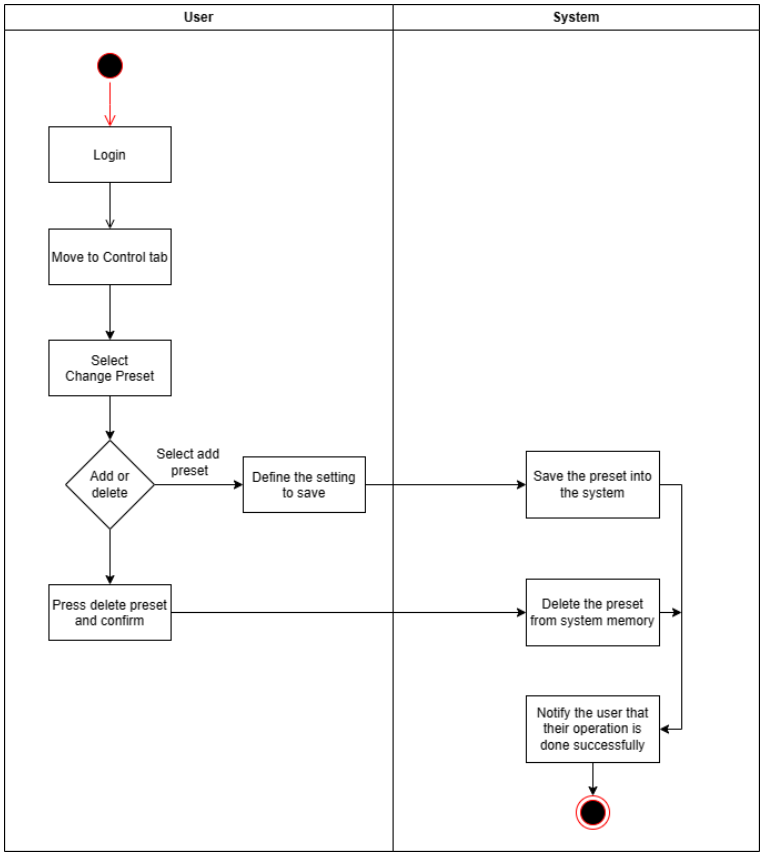


Figure 30: Activity Diagram for Add/remove home setting preset

12.4 Detect user's present

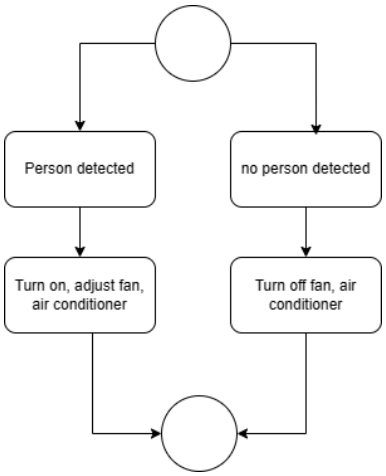


Figure 31: Activity diagram of present detection

12.5 Adjust fan and air conditioner

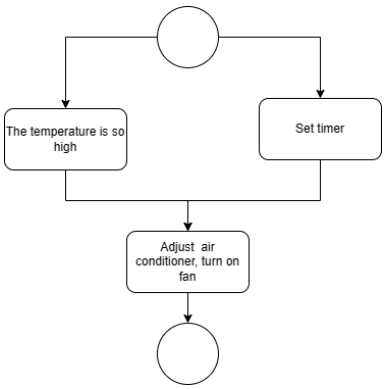


Figure 32: Activity diagram of adjusting air, fan

12.6 Control lightning

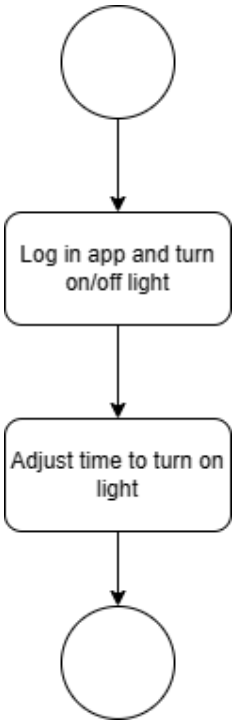


Figure 33: Activity diagram of control lighting

13 IoTs gateway

13.1 Server

Adafruit, renowned for its comprehensive Internet of Things (IoT) solutions, has emerged as a global leader in innovation, providing makers and engineers with easily accessible tools, guides, and parts. As we embark on this report, we discover the features and functions of Adafruit, specifically focusing on utilizing its versatile Feeds feature for transmitting vital data from the IoT devices of the Intelligence Home into a centralized dashboard.

For **Feeds - An Engine for Data Transmission**, Adafruit's Feeds feature serves as the engine powering the transmission of critical information. In this project, we may initialize some useful variables for receiving IoT devices' input data to form the bedrock of the data exchange mechanism between device systems on Microbit, IoT gateway, Cloud server, and Web.

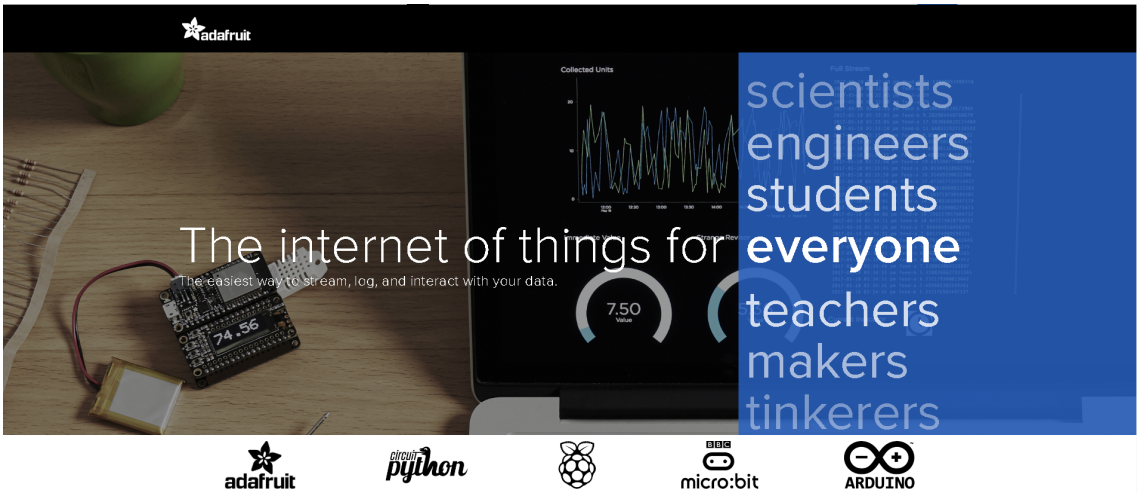


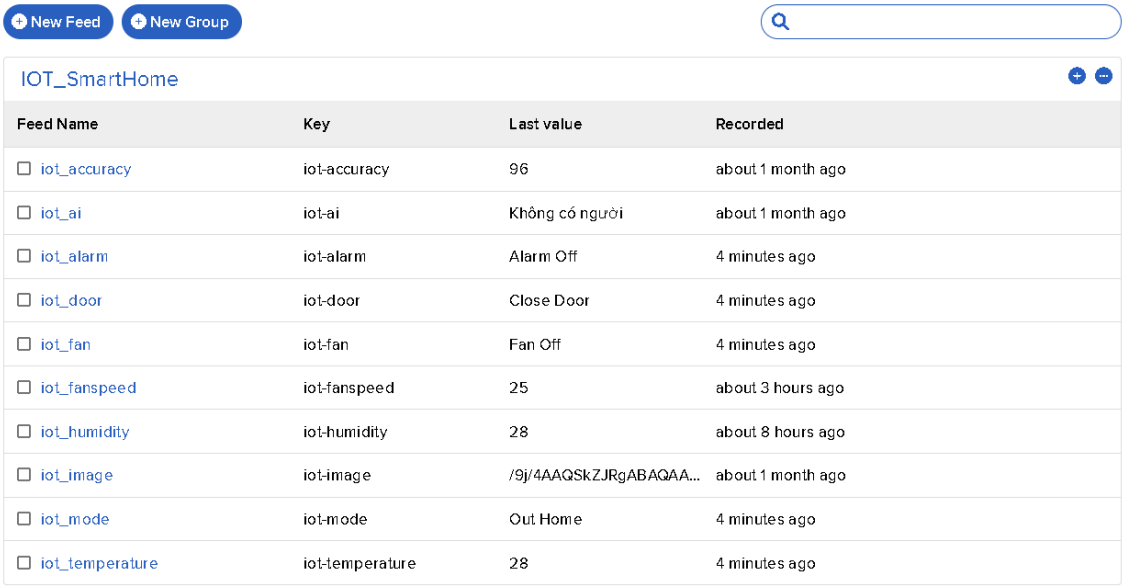
Figure 34: Adafruit

Also, we can utilize its notable and prominent feature **Dashboard Creation for Comprehensive Insights**, which is the heart of Adafruit's integration lying in the generation of a dynamic dashboard, which is designed to provide a holistic view. By leveraging the capabilities of Adafruit's Feeds, the dashboard becomes an invaluable tool for monitoring and analyzing real-time data.

On the other hand, with **Enhanced Layout Customization**, Adafruit's dashboard design is not merely functional but also highly customizable. Through the incorporation of blocks such as Live Image, Text, IoT Buttons, and Temperature/Humidity/Overall History, the layout is tailored to offer a visually intuitive representation of IoT functions in Intelligence Home. This enhances the interpretability of data for users, allowing for swift decision-making based on accurate and timely insights.

In the process of exploring Adafruit's Feeds and building a feature-rich dashboard, the paper seeks to clarify how IoT technologies and Client Dashboard/Web may work together harmoniously to create a more connected and knowledgeable future.

Feeds and Dashboard: Adafruit Server



Feed Name	Key	Last value	Recorded
iot_accuracy	iot-accuracy	96	about 1 month ago
iot_ai	iot-ai	Không có người	about 1 month ago
iot_alarm	iot-alarm	Alarm Off	4 minutes ago
iot_door	iot-door	Close Door	4 minutes ago
iot_fan	iot-fan	Fan Off	4 minutes ago
iot_fanspeed	iot-fanspeed	25	about 3 hours ago
iot_humidity	iot-humidity	28	about 8 hours ago
iot_image	iot-image	/9j/4AAQSkZJRgABAQAA...	about 1 month ago
iot_mode	iot-mode	Out Home	4 minutes ago
iot_temperature	iot-temperature	28	4 minutes ago

Figure 35: The Variables in Feeds

After that, generate the DASHBOARD, and edit Layout with some blocks such as **Live Image, Text, IoT Buttons, and Temperature/Humidity/Overall History** blocks.

Figure 36: The Dashboard with Simple Data

13.2 Client

On the client side, we will create an IoT gateway. IoT gateway is responsible for receiving serial signals from sensors and hardware and sending them to the server and vice versa. Communication with the server here will use the MQTT protocol and communication with the hardware will be uart communication.

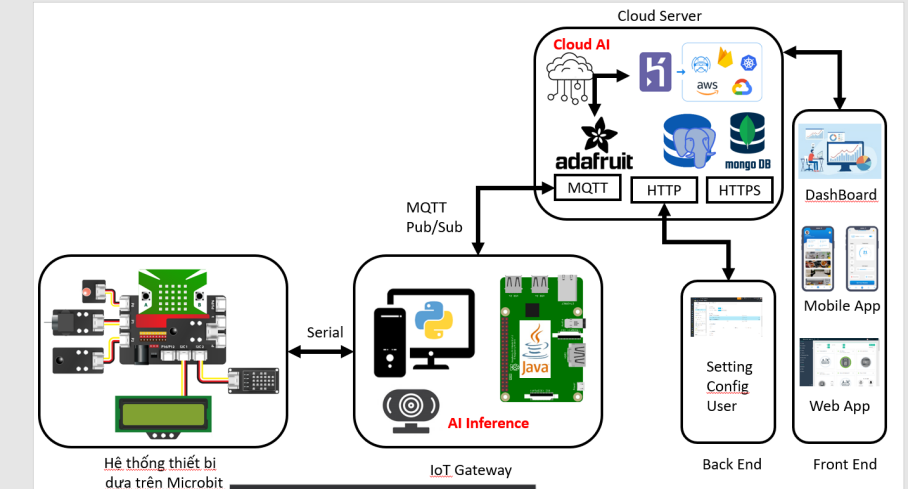


Figure 37: Two-way communication of IoT gateway

First, we will connect to the adafruit server using the MQTT library. And check the publish and subscribe methods.

```
import sys
from Adafruit_IO import MQTTClient
AIO_KEY = "aio_"
AIO_USERNAME = "Unray"
AIO_FEED_ID = ["button1", "button2"]

def connected(client):
    print("Connected to adafruit")
    for topic in AIO_FEED_ID:
        client.subscribe(topic)

def subscribe(client , userdata , mid , granted_qos):
    print("Subscribe successfully ...")

def disconnected(client):
    print("Disconnected ...")
    sys.exit (1)

def message(client , feed_id , payload):
    print("Send to adafruit " + payload)

print(1)
client = MQTTClient(AIO_USERNAME , AIO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
```

```
client.on_message = message
client.on_subscribe = subscribe
client.connect()
client.loop_background()
while True:
    pass
```

Next we will use two software to simulate the signal from the sensor. Since I haven't received the device yet, I'll use these two software to test the communication functions from the hardware to the server. First, the Hercules software will be used to create sensor signals as well as receive signals from the server. The second is com0com, which will be used to connect two com ports, the first port is opened by IoT gateway, the second port is open by Hercules, so when Hercules sends a signal, the IoT gateway will receive it.

```
import serial.tools.list_ports
import random
import time
import  sys
from Adafruit_IO import MQTTClient

def getPort():
    ports = serial.tools.list_ports.comports()
    N = len(ports)
    commPort = "None"
    for i in range(0, N):
        port = ports[i]
        strPort = str(port)
        if "USB Serial Device" in strPort:
            splitPort = strPort.split(" ")
            commPort = (splitPort[0])
    return "COM8"#commPort

if getPort != "None":
    ser = serial.Serial( port=getPort(), baudrate=115200)
    print(ser)

def processData(client, data):
    data = data.replace("!", "")
    data = data.replace("#", "")
    splitData = data.split(":")
    print(splitData)
    if splitData[1] == "T":
        client.publish("button1", splitData[2])

mess = ""
def readSerial(client):
```

```
    bytesToRead = ser.inWaiting()
    if (bytesToRead > 0):
        global mess
        mess = mess + ser.read(bytesToRead).decode("UTF-8")
        while ("#" in mess) and ("!" in mess):
            start = mess.find("!")
            end = mess.find("#")
            processData(client, mess[start:end + 1])
            if (end == len(mess)):
                mess = ""
            else:
                mess = mess[end+1:]
```

Above, my team has created a simple IoT gateway for devices to communicate with each other. This IoT will be integrated and improved when there is a web app.

14 Database design

14.1 Er diagram

We will do the conceptual design of our database using the Enhanced Entity Relation Ship diagram. It will outline the major entities required in our database, as well as there relation ship with other entities.

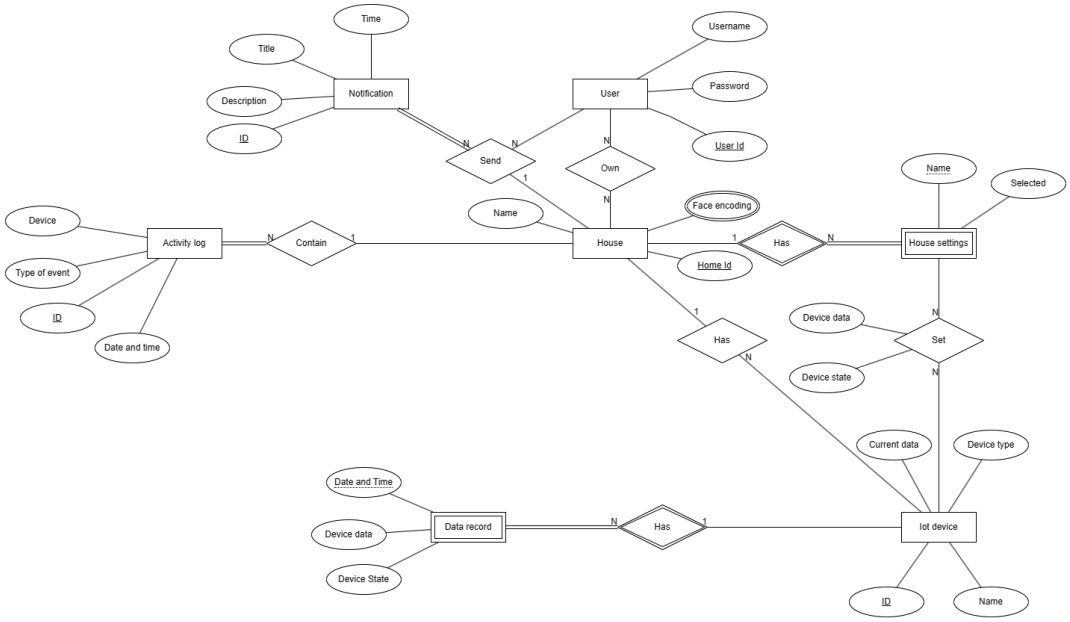


Figure 38: EER model

The schema for the diagram illustrates the following point. The **user** could own many iot **home** with each **home** could be co own by many **user**. The **user** also could receive **notification** from the system if there is something wrong in the **home**.

The iot **home** will keep the **activity log** that will detail the action of each **iot device** in the **home**. each **home** will have many default preset **home settings** which are set by user that will store the default configuration of each iot device.

The **iot device** that are available as of currently for the user to use and control include **Light**, **Temperature**, **Fan**.

14.2 Schema mapping

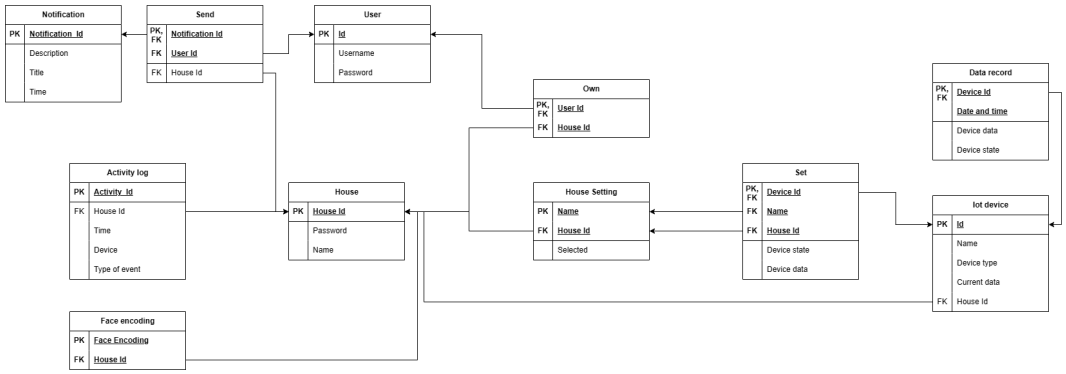


Figure 39: Schema mapping from ER diagram

14.3 Physical design

Name	**Data type**	**Constraint**	**Description**
Notification_id	INT	PK, NOT NULL	The Id of a notification
Description	NVARCHAR(255)	NOT NULL	The detail, message describing the notification
Title	NVARCHARR(255)	NOT NULL	The title of the notification
Time	SMALLDATETIME	NOT NULL	The time the notification is sent

Table 7: Notification table

Name	**Data type**	**Constraint**	**Description**
Notification_id	INT	PK, FK, NOT NULL	The Id of a notification
User_id	INT	PK, FK, NOT NULL	The User Id that the notification is sent to
House_id	INT	FK, NOT NULL	The House Id that the notification is sent from

Table 8: Send table

Name	Data type	Constraint	Description
User_id	INT	PK, NOT NULL	The Id of a user
username	NVARCHAR(50)	NOT NULL	The user's username
password	VARCHAR(128)	NOT NULL	The user's password

Table 9: Users table

Name	Data type	Constraint	Description
User_id	INT	PK, FK, NOT NULL	The Id of a user that own the house
House_id	INT	PK, FK, NOT NULL	The Id of a house that is owned by the user

Table 10: Own table

Name	Data type	Constraint	Description
House_id	INT	PK, NOT NULL	The Id of a House
Name	NVARCHAR(50)	NOT NULL	The house's name
password	VARCHAR(255)	NOT NULL	The house's password, used to gain access into the house

Table 11: House table

Name	Data type	Constraint	Description
Acitivity_id	INT	PK, NOT NULL	The Id of the Activity log
House_id	INT	FK, NOT NULL	The house's id that the log belong to
Time	SMALLDATETIME	NOT NULL	The time that the activity was recorded
Device	NVARCHAR(50)	NOT NULL	The type of the device that generated the log, must be humidity, alarm, light, temperature, fan or door
Type_of_event	NVARCHAR(50)	NOT NULL	The type of event that the device generated, must be change setting, remove device, add new device.

Table 12: Activity log table

Name	Data type	Constraint	Description
Face_encoding	VARCHAR(255)	PK, NOT NULL	The face encoding that was hashed by a face recognition system, use to recognise the house owner
House_id	INT	PK, NOT NULL	The house's id

Table 13: Face encoding table

Name	Data type	Constraint	Description
Name	NVARCHAR(50)	PK, NOT NULL	The name of the house setting
House_id	INT	PK, FK, NOT NULL	The house's id that the setting belong to
Selected	BIT	NOT NULL	The status of the setting whether it is selected by a user or not

Table 14: House setting table

Name	Data type	Constraint	Description
Device_id	INT	PK, FK, NOT NULL	The Id of the device that the setting affect
Name	NVARCHAR(50)	PK, FK, NOT NULL	The name of the setting that the device belong to
House_id	INT	PK, FK NOT NULL	The house's id that the setting belong to
Device_state	BIT	NOT NULL	The state of the device, can only be on or off (1 or 0)
Device_data	FLOAT	NOT NULL	The device's data that will be used to set for the IoT device.

Table 15: Set table

Name	Data type	Constraint	Description
Device_id	INT	PK, NOT NULL	The Id of the device
Name	NVARCHAR(50)	NOT NULL	The name of the device
Device_type	VARCHAR(50)	NOT NULL	The type of device, must be humidity, alarm, light, temperature, fan or door
Current_data	FLOAT	NOT NULL	The device's newest data
House_id	INT	FK, NOT NULL	The house'id that the IoT device belong to

Table 16: Iot device table

Name	Data type	Constraint	Description
Device_id	INT	PK, FK, NOT NULL	The Id of the device that has the record
Date_and_time	SMALLDATETIME	PK, NOT NULL	The time that the record was recorded
Device_data	FLOAT	NOT NULL	The device's recorded data
Device_state	BIT	NOT NULL	The device's state, on or off

Table 17: Data record table

15 UI Design and Implementation

15.1 UI Design

To create a visually appealing and easy to use UI, but still consist of various feature, we use Figma to desgin our application. But before we can do that effectively, we need to create a sketch based on our vision of the final website. After we have the sketch, we use Figma to create the UI:

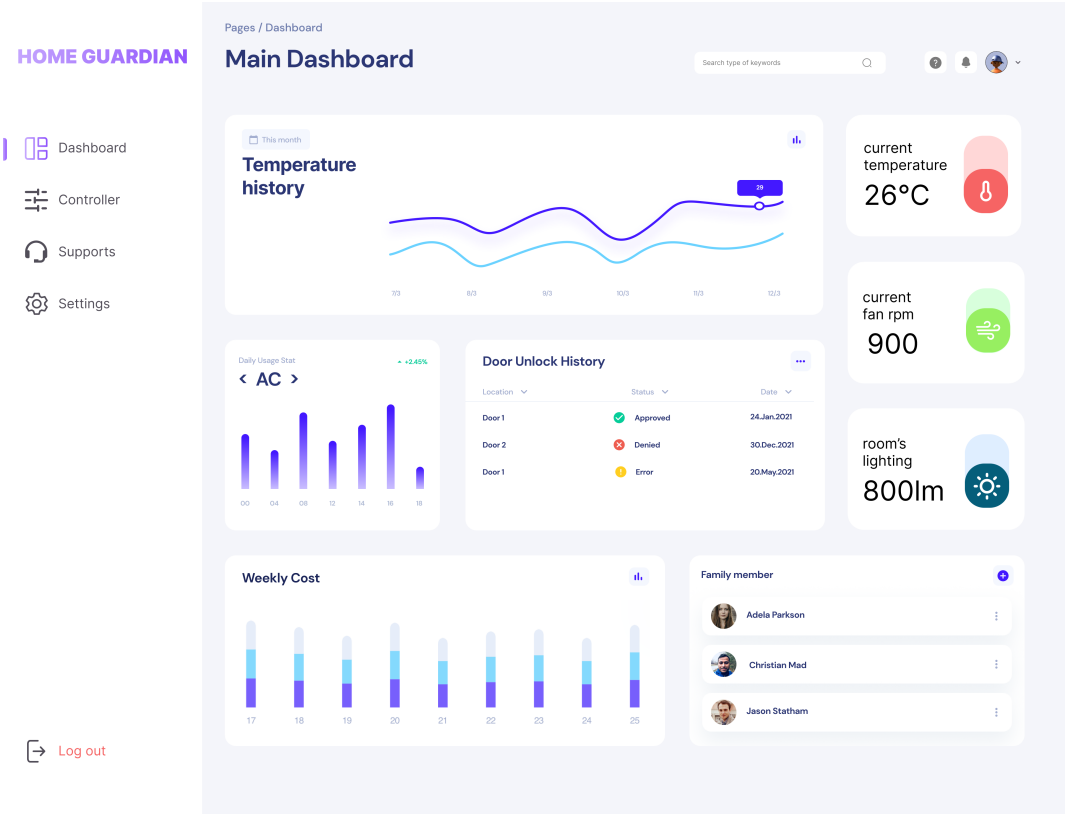


Figure 40: Dashboard

The dashboard is created with the user in mind, most of the statistics about temperature or fan can be found here as well af multiple graph to help the user better understand their smart home system

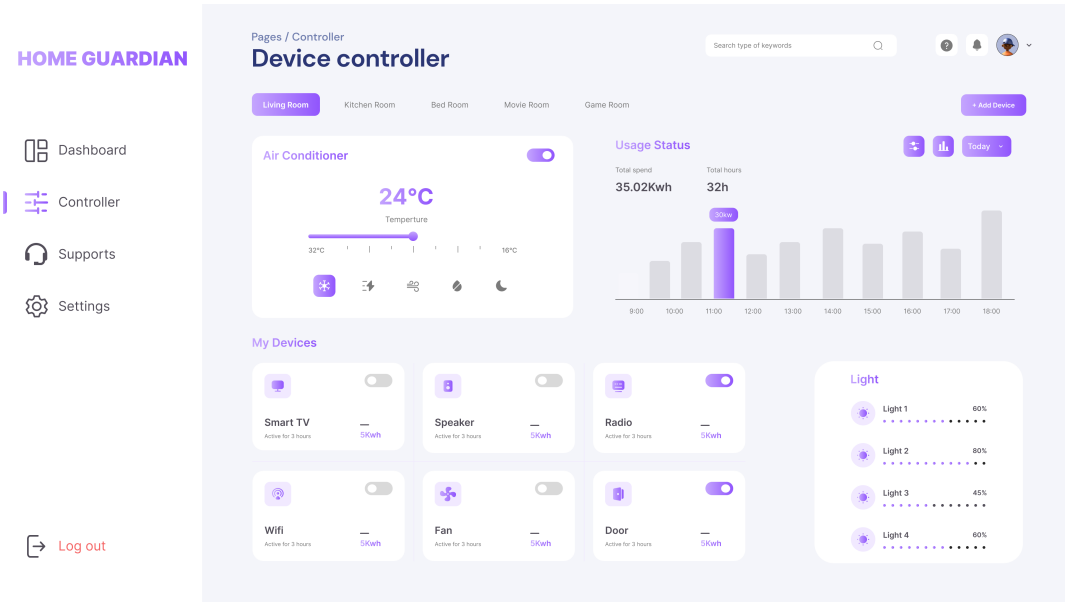


Figure 41: Device controller

In the controller tab, the user can see and control all the devices that are connected to the system, this include AC, fan, light,... All the room connected to the system can also be controlled here. The user can also add a new device to the system using a button in the UI,

The design can be checked out at this figma link: `https://www.figma.com/file/XFAMKlOscfK57isTfK0W4e/UI?type=design&node-id=0%3A1&mode=design&t=FKsY38hQc81rlnQy-1`

16 MVP demonstrations

The video will be included in the regular report submission to lecturer.

References

[1] FBI. "FBi - Crime Clock". In: (2017). URL: `https://ucr.fbi.gov/crime-in-the-u.s/2019/crime-in-the-u.s.-2019/topic-pages/crime-clock`.

[2] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st. Addison-Wesley Professional, 1994.

[3] geeksforgeeks. *MVC Design Pattern*. `https://www.geeksforgeeks.org/mvc-design-pattern/`. Accessed: 15 March, 2024. 19 Feb, 2024.

[4] Refactoring.Guru. *Adapter in Go*. https://refactoring.guru/design-patterns/adapter/go/example. Accessed: 15th April, 2024.

[5] Refactoring.Guru. *Factory Method in Go*. https://refactoring.guru/design-patterns/factory-method/go/example. Accessed: 15th April, 2024.

[6] Refactoring.Guru. *Singleton in Go*. https://refactoring.guru/design-patterns/singleton/go/example. Accessed: 12th April, 2024.

[7] Refactoring.Guru. *Strategy in Go*. https://refactoring.guru/design-patterns/strategy/go/example. Accessed: 15th April, 2024.

[8] Ryan Thelin. *Clean architecture tutorial: Design for enterprise-scale apps*. https://www.educative.io/blog/clean-architecture-tutorial. Accessed: 19 March, 2024. 18 Aug, 2024.