Home&Co

Ynov - Development Project

Antoine de Barbarin
Nicolas Moyon

April 5th, 2025

Contents

1 Presentation

This project is made by Computer Science students for an assignment involving a client application, in this case a website, and a real time interaction with the physical world via ESP32 boards.

Home&Co *(Co for Connect, Control, Company)* is a project using **ESP32** microcontrollers and a **Raspberry Pi 3B+** to monitor, control and automate home appliances like the lights, heaters, front door, water valves for watering plants, etc.

Its goals are to provide **security**, **control**, **comfort**, **peace of mind** and **energy saving** to your home.

1.1 Security

Our front door device will control the access policy of the front door of your home:

- **RFID** sensor needing a **badge** to open the door
- **doorbell button** ringing the bell and **notifying you on your phone**
- **locking mechanism** controlled **remotely** (no need to give keys to anybody)
- **presence detector** to **monitor** the activity in front of the door
- **house locking function** to **automatically turn off** all the lights and use all presence detectors as **intrusion detectors**

1.2 Control

You can use the website to **remotely control at any time** the *lights*, the *door*, the *heaters* and the *watering of plants*.

You have **full control of your house** from any device.

1.3 Comfort

You can **automate the heaters** to activate at any time of the day, to find a warm house when coming back from a hike, vacation or any activity.

1.4 Peace of mind

You will not have to go through your checklist three times or ask your neighbor to water your plants before going out!

You can **monitor your house from any device** using the website in **real time**: from the *presence detectors* to the *lights*, *heaters*, *humidity of the plants* and *temperature of any room*.

1.5 Energy saving

No need to keep heating your home when you're not there!

You will be able to control and monitor the **heaters** remotely and according to your needs and presence at home.

You will also be able to **monitor the energy of specific high consumption appliances** with our **smart power outlet** and control them remotely, and even give them schedules to be working or not.

2 System Design

2.1 Needs Analysis

- **Goal**: fully functional home with remote control (from the web browser)
- **Security Constraint**: no direct access to ESP32 microcontrollers with their sensors and actuators, to database, MQTT or backend application.
- **Components**:
  - Raspberry Pi 3 or more for Wi-Fi access point, MQTT, backend app and webserver.
  - ESP32 for sensors & actuators

2.2 Technical choices

- **Golang** for the backend app and webserver with GORM:
  - Golang is an excellent choice for building efficient, concurrent applications due to its simplicity and strong type system.
  - The Go standard library is comprehensive, making it easy to handle tasks like networking, file I/O, and concurrency.
  - GORM, a popular ORM for Golang, simplifies database interactions by providing a clean and intuitive API.
- **PostgreSQL** for the database:
  - PostgreSQL is an advanced, open source relational database management system known for its robustness, scalability, and ACID compliance.
  - It offers features like transactions, indexing, and support for complex data types, making

it ideal for handling diverse and large-scale applications.

- **Mosquitto** for the MQTT broker:
  - Mosquitto is a lightweight, open source MQTT broker that's highly reliable and easy to configure.
  - Its simplicity and performance make it an excellent choice for real-time communication between devices and services, especially in scenarios where low latency and high availability are crucial.
- **RaspAP** for the Wi-Fi Access Point in the Raspberry Pi:
  - RaspAP provides a user-friendly interface for setting up and managing a Wi-Fi access point on a Raspberry Pi.
  - It simplifies the process of configuring network settings, security protocols, and other related tasks, making it accessible even to users with limited technical expertise.
- **Websocket** between JavaScript clients and Golang server:
  - Websockets provide full-duplex communication channels over a single TCP connection, enabling real-time data exchange between the frontend and backend.
  - This is particularly useful for applications requiring immediate updates and interactions, such as chat applications or live dashboards.
- **PlatformIO** and **C/C++** for the ESP32:
  - PlatformIO offers a unified development environment for embedded systems, making it easy to manage multiple projects and boards.
  - It supports a wide range of platforms and tools, including C/C++, which is ideal for writing efficient and low-level code for microcontrollers like the ESP32.

2.3 Basic features

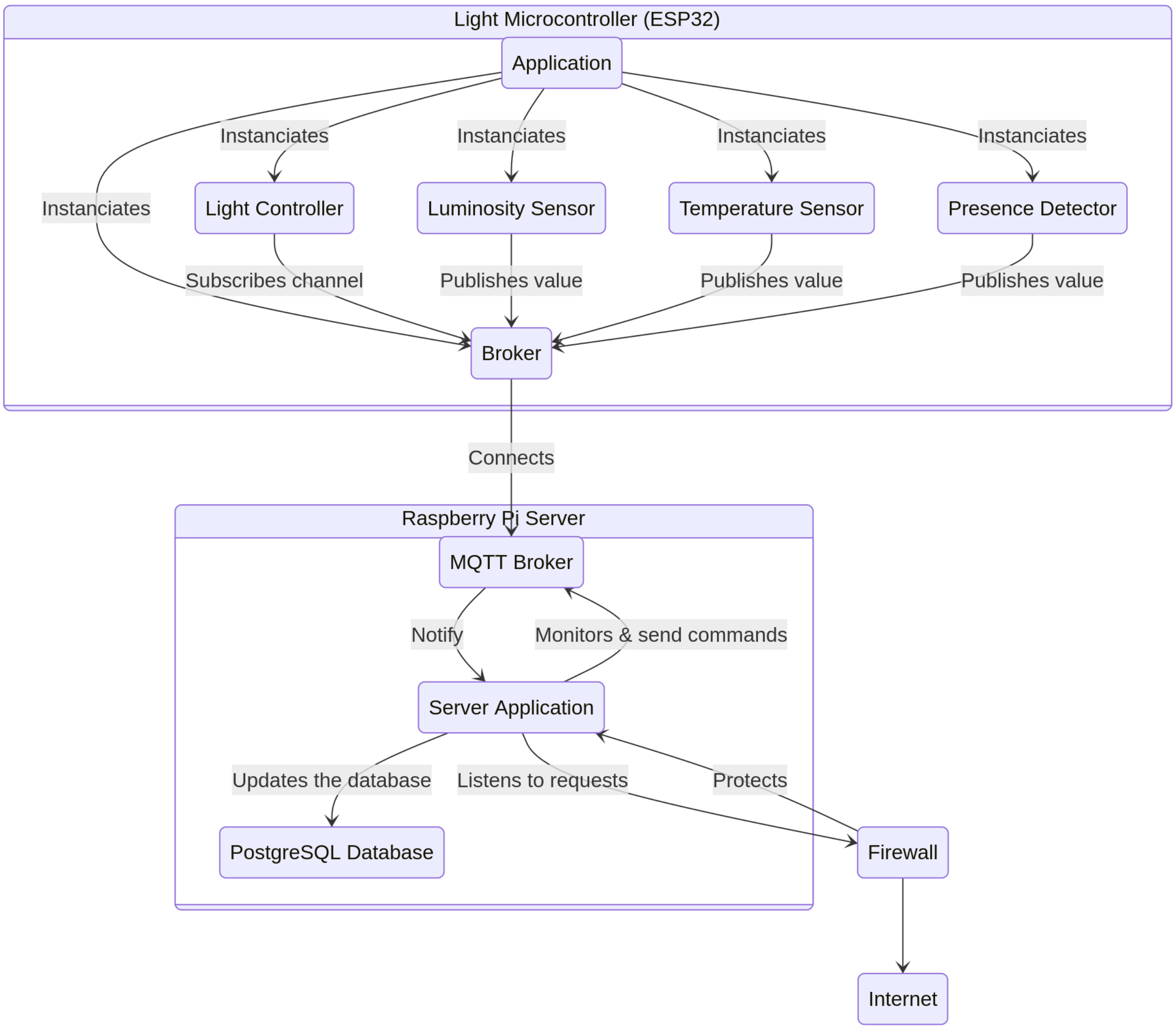
- Monitoring sensors with the web interface
- Controlling actuators from the web interface
- Data collection in the database
- Light and temperature management
- Front door monitoring with `RFID` badge

2.4 Additional features (future release)

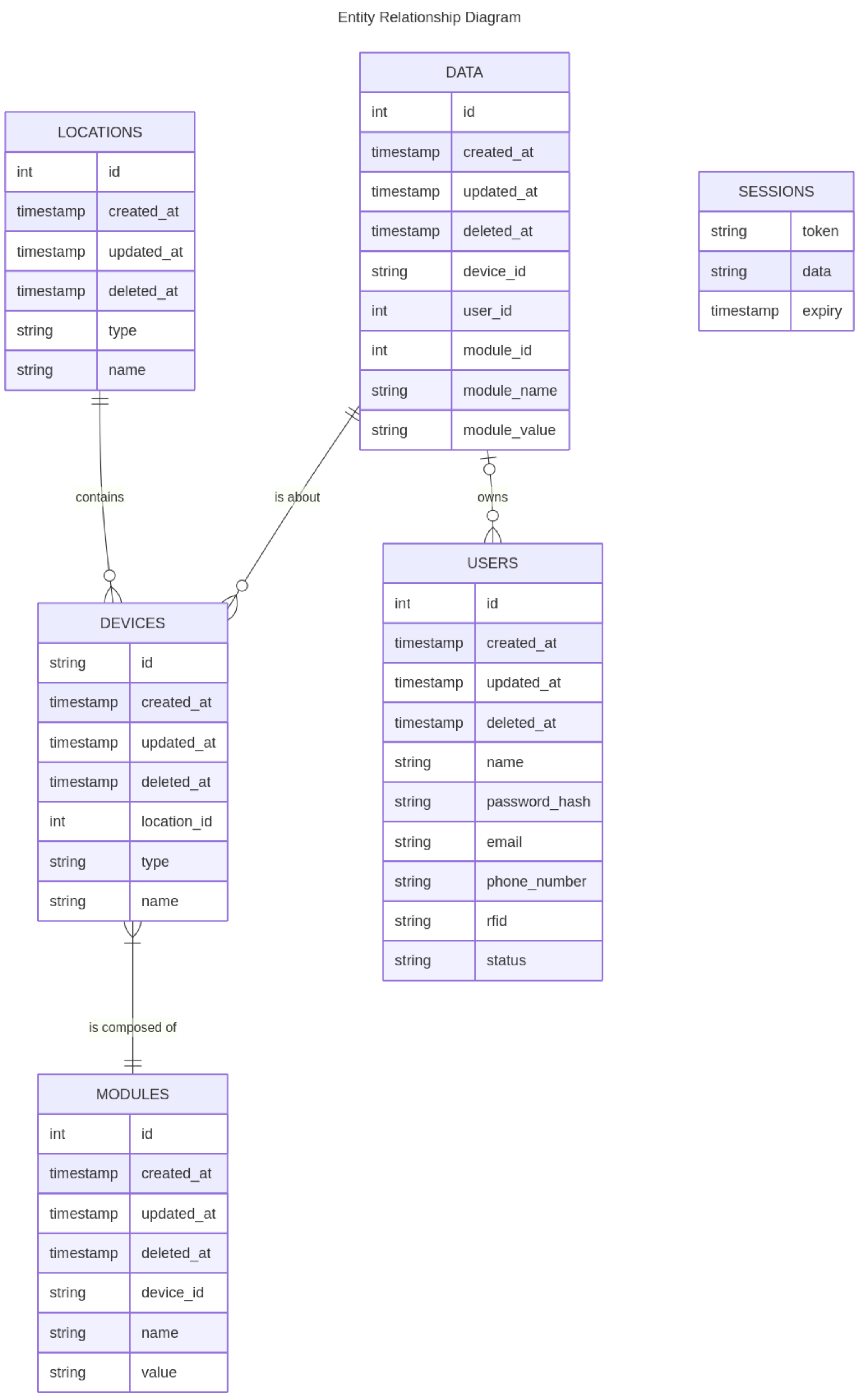
- Energy consumption management
- Shutter and window management
- Fire and smoke detection
- Alarm mode
- Ventilation management
- Scheduler for actuators (heaters, light, shutters, ventilation)
- Statistics and prevision dashboards

2.5 Components Architecture

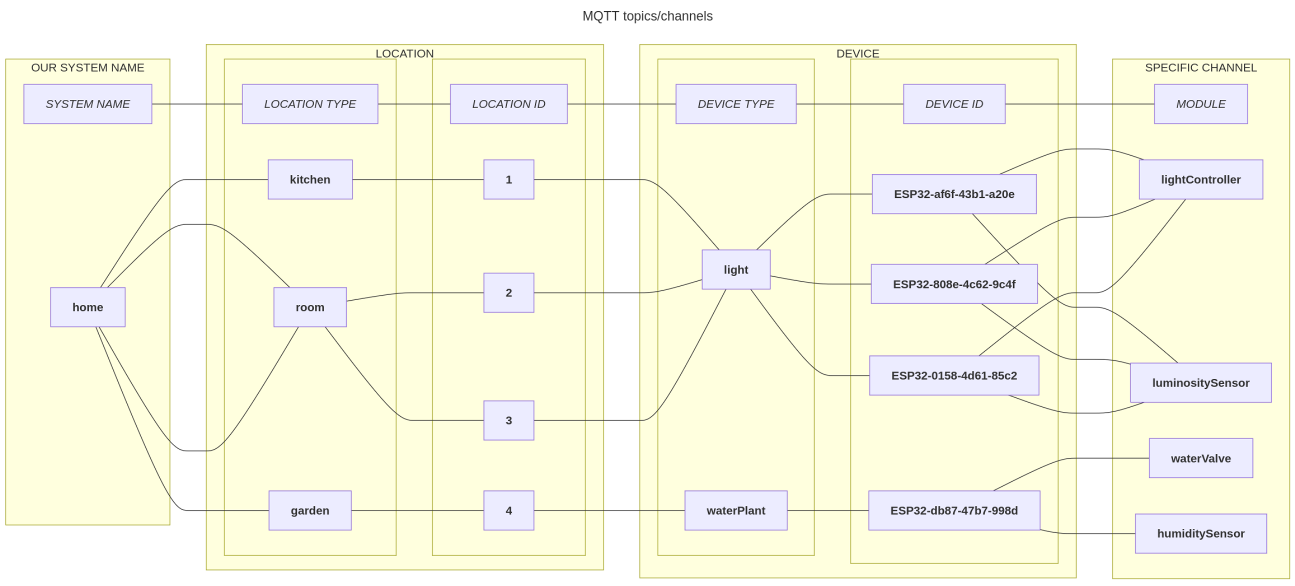
2.5.1 Overall Infrastructure



2.5.2 Database



2.5.3 MQTT



2.5.4 Webserver File Architecture

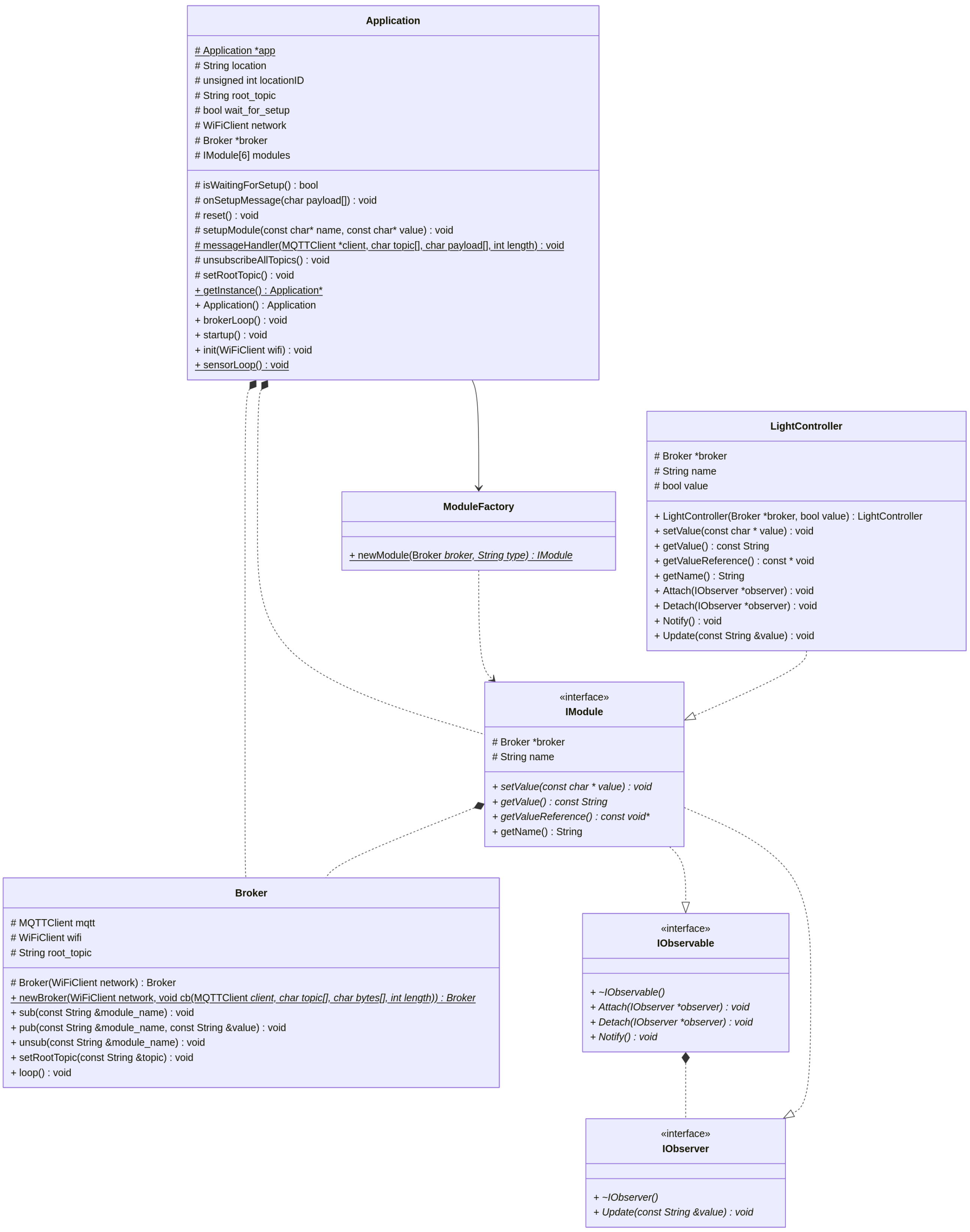
```
|-- cmd
|   |-- web
|       |-- context.go
|       |-- handlers.go
|       |-- helpers.go
|       |-- main.go
|       |-- middleware.go
|       |-- models.go
|       |-- routes.go
|       |-- server.go
|       |-- templates.go
|-- go.mod
|-- go.sum
|-- internal
|   |-- data
|   |   |-- broker.go
|   |   |-- consumption-sensor.go
|   |   |-- data.go
|   |   |-- device.go
|   |   |-- imodule.go
|   |   |-- light-controller.go
|   |   |-- light-sensor.go
|   |   |-- location.go
|   |   |-- luminosity-sensor.go
|   |   |-- models.go
|   |   |-- module.go
|   |   |-- presence-detector.go
|   |   |-- reset.go
|   |   |-- startup.go
|   |   |-- subscription.go
|   |   |-- temperature-sensor.go
|   |   |-- value-conversion.go
|   |-- mailer
|   |   |-- mailer.go
|   |   |-- templates
|   |       |-- alert-notification.tmpl
|   |-- validator
|       |-- validator.go
|-- README.md
|-- ui
|   |-- assets
|   |   |-- css
|   |   |   |-- base.scss
|   |   |   |-- style.scss
|   |   |-- font
|   |   |   |-- Dosis-VariableFont_wght.ttf
|   |   |-- img
|   |       |-- logo
|   |           |-- logo.png
|   |-- efs.go
|   |-- templates
|       |-- base.tmpl
|       |-- pages
|       |   |-- error.tmpl
|       |   |-- home.tmpl
|       |-- partials
|-- vendor
```

2.5.5 ESP32 C++ File Architecture

```
|-- include
|   |-- Application.h
|   |-- Broker.h
|   |-- ConsumptionSensor.h
|   |-- environment.h
|   |-- IModule.h
|   |-- IObservable.h
|   |-- IObserver.h
|   |-- LightController.h
|   |-- LightSensor.h
|   |-- LuminositySensor.h
|   |-- ModuleFactory.h
|   |-- MyAny.h
|   |-- PresenceDetector.h
|   |-- README
|   |-- TemperatureSensor.h
|   |-- utils.h
|-- lib
|   |-- README
|   |-- xht11
|       |-- xht11.cpp
|       |-- xht11.h
|-- LICENSE
|-- platformio.ini
|-- README.md
|-- src
   |-- Application.cpp
   |-- Broker.cpp
   |-- ConsumptionSensor.cpp
   |-- HomeIoT.ino
   |-- LightController.cpp
   |-- LightSensor.cpp
   |-- LuminositySensor.cpp
   |-- ModuleFactory.cpp
   |-- PresenceDetector.cpp
   |-- TemperatureSensor.cpp
   |-- utils.cpp
```

2.5.6 ESP32 Class Diagram

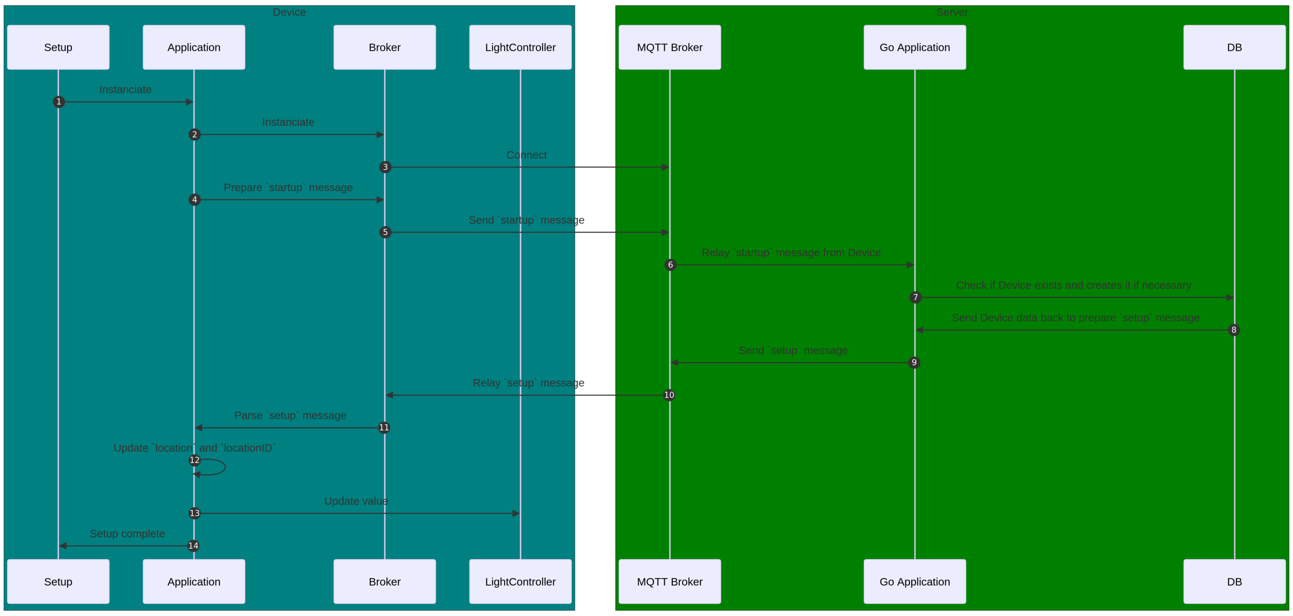
Device class diagram


Application
# Application *app
# String location
# unsigned int locationID
# String root_topic
# bool wait_for_setup
# WiFiClient network
# Broker *broker
# IModule[6] modules
# isWaitingForSetup() : bool
# onSetupMessage(char payload[]) : void
# reset() : void
# setupModule(const char* name, const char* value) : void
# messageHandler(MQTTClient *client, char topic[], char payload[], int length) : void
# unsubscribeAllTopics() : void
# setRootTopic() : void
+ getInstance() : Application*
+ Application() : Application
+ brokerLoop() : void
+ startup() : void
+ init(WiFiClient wifi) : void
+ sensorLoop() : void
LightController
# Broker *broker
# String name
# bool value
+ LightController(Broker *broker, bool value) : LightController
+ setValue(const char * value) : void
+ getValue() : const String
+ getValueReference() : const * void
+ getName() : String
+ Attach(IObserver *observer) : void
+ Detach(IObserver *observer) : void
+ Notify() : void
+ Update(const String &value) : void
ModuleFactory
+ newModule(Broker *broker*, String type) : IModule
«interface»
IModule
# Broker *broker
# String name
+ setValue(const char * value) : void
+ getValue() : const String
+ getValueReference() : const void*
+ getName() : String
Broker
# MQTTClient mqtt
# WiFiClient wifi
# String root_topic
# Broker(WiFiClient network) : Broker
+ newBroker(WiFiClient network, void cb(MQTTClient *client*, char topic[], char bytes[], int length)) : Broker
+ sub(const String &module_name) : void
+ pub(const String &module_name, const String &value) : void
+ unsub(const String &module_name) : void
+ setRootTopic(const String &topic) : void
+ loop() : void
«interface»
IObservable
+ ~IObservable()
+ Attach(IObserver *observer) : void
+ Detach(IObserver *observer) : void
+ Notify() : void
«interface»
IObserver
+ ~IObserver()
+ Update(const String &value) : void

2.5.7 Setup/Startup

```json
{
    "id": "ESP32-af6f-43b1-a20e",
    "type": "light",
    "location_id": 3,
    "location_type": "room",
    "location_name": "room 3",
    "modules": [
        {
            "name": "lightController",
            "value": "False"
        },
        {
            "name": "lightSensor",
            "value": "True"
        },
        {
            "name": "luminositySensor",
            "value": "150.0"
        },
        {
            "name": "presenceDetector",
            "value": "True"
        },
        {
            "name": "temperatureSensor",
            "value": "22.5"
        },
        {
            "name": "consumptionSensor",
            "value": "32.45"
        }
    ]
}
```

2.5.8 Setup Sequence Diagram



3 Project Management

3.1 Role Management

- **Overall structure**: Antoine & Nicolas
- **Documentation**: Antoine & Nicolas
- **MQTT topics**: Antoine
- **Raspberry Pi setup**: Antoine
- **ESP32 structure & code**: Antoine & Nicolas
- **Database**: Antoine & Nicolas
- **Golang webserver & Application**: Antoine & Nicolas
- **Frontend**: Antoine
- **Websockets**: Nicolas

3.2 Workflow

We organized our project and advancement using an agile workflow with sprints.

3.2.1 Sprints

☒ **Project Structure**: 07/03 - 14/03

☒ **MQTT topics**: 07/03 - 14/03

☒ **Database**: 10/03 - 14/03

☒ **ESP32 Structure**: 14/03 - 21/03

- ☒ **Raspberry Pi Setup**: 14/03 - 24/03
- ☒ **ESP32 Code**: 14/03 - 24/03
- ☐ **Golang Webserver & Application**: 10/03 - ongoing
- ☐ **Websockets**: 07/04 - ongoing
- ☐ **Frontend**

3.3 Issues

- **System design**: implementing logic and standards with specific constraints.
  We resolved this issue making schemas and lists of constraints on paper to have a better overview and be able to think more properly.
- **Environment differences** between Linux & Windows (terminal commands, environment variables).
  This issue was resolved creating a *powershell* script that loads in the current terminal session all *environment variables* for Windows.