Technical Report - **Product specification**

My Sweet Home

Course:	IES - Introdução à Engenharia de Software
Date:	Aveiro, 24th October 2023
Students:	108969 : Rodrigo Silva Aguiar 108840 : José Gameiro 107348 : Pedro Ramos 107603 : Daniel Madureira
Project abstract:	My Sweet Home is a system where a user can associate multiple output devices installed in his home to certain actions and rules that act upon the information from input devices.

Table of contents:

1 Introduction

This project takes place as the final assignment for the Introduction to Software Engineering (IES) course. As we embark on the journey of software development, it's essential to understand the broader context and objectives of this particular project.

In a world where technology continues to reshape the way we live, 'My Sweet Home' emerges as a way to simplify the setup and connection between the ever growing quantity of intelligent devices that live in our homes. Our vision is to create a seamless and intuitive system that empowers users to effortlessly connect and orchestrate the devices in their homes, transforming ordinary living spaces into intelligent, responsive environments. 'My Sweet Home' is a project that promises to redefine the way we interact with our living spaces by enabling users to associate input devices, such as thermostats, daylight sensors, biometric sensors, with customized output actions.

Imagine a home where your preferences are not just known but actively catered to, where your actions harmonize with the desires of others in your household. 'My Sweet Home' will bring this vision to life by tailoring actions to each specific user and ensuring that they do not conflict with the preferences of fellow residents. For instance, if one user wants to enjoy music at a specific time, 'My Sweet Home' will ensure that the music does not disrupt another's peace and quiet.

This is not just about automation; it's about personalization. From simple tasks like adjusting the temperature or routing your phone's music to the house's speakers upon your arrival, to more complex scenarios limited only by your imagination.

With a user-friendly front-end interface, you can effortlessly define and refine these action-reaction pairs, making your home smarter, more comfortable, and uniquely yours.

Welcome to 'My Sweet Home,' where technology enhances your life and transforms your home into a sanctuary of customized comfort and convenience.

2 Product concept

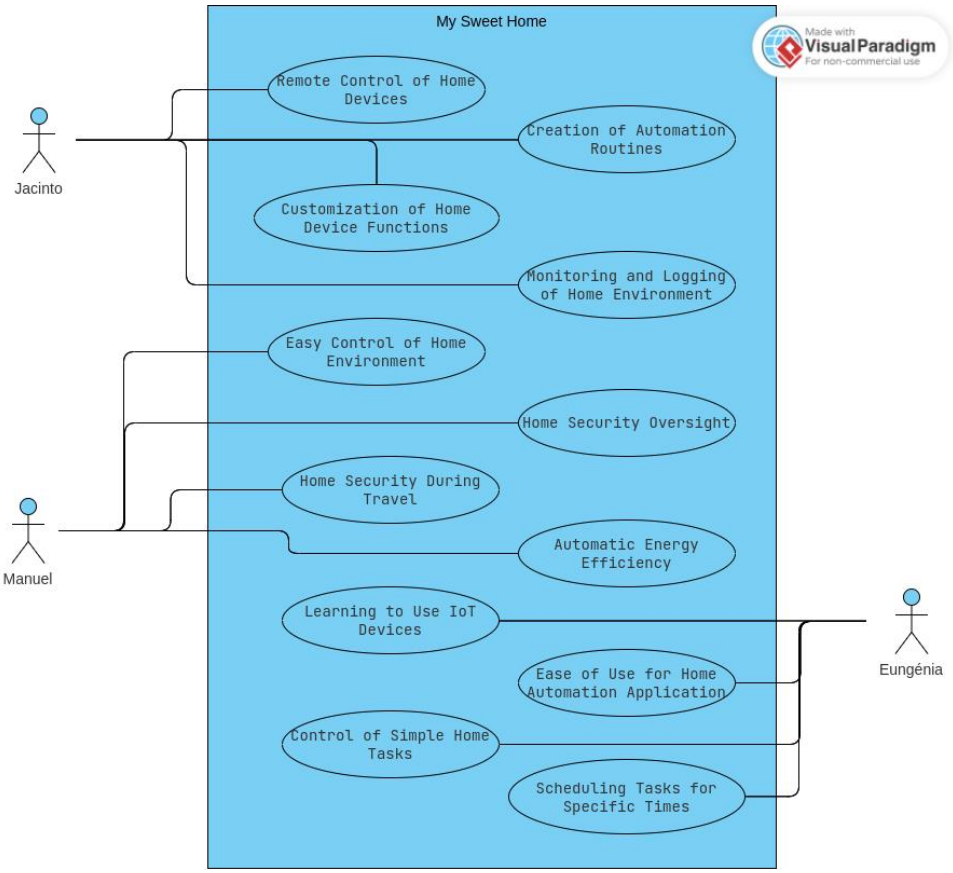
Vision statement

- Our aim with My Sweet Home is to develop a system where a user can associate input devices in his house to certain actions.
- This project involves several receivers (inputs, such as thermostats, daylight-sensors, biometric-sensors, etc...) to be associated with output actions defined by the user and executed by a set of output devices.
- The actions would be tailor made for each specific user and would need to verify if, for example, a user's action would interfere with another user's preference, like if a user likes to turn on music at 11:00h but another user, which we know is inside the same house, doesn't want to be bothered by music until later.
- One example of a simple task would be to close the windows if the temperature got too cold for the specifications of the users inside the house, or to pass the music streaming of a user's phone to the house's speakers after he arrives home.
- In the front-end we would be able to set these action-reaction pairs.

Many other products like these exist, but most are both closed source and proprietary, meaning that even if you already have such smart devices, they many times lack the capacity to communication between them or the brains necessary to associate the wanted action-reaction pairs for proper home automation.

We aim to make our implementation both open source and easy to work with, so new devices can be implemented and even adapted to work with our solution with easy with some internal know how.

Use case Diagram



Personas and Scenarios

A Persona is a character utilized to tell stories that represent future possible use cases for the system. A Persona is an instanciated actor with a given set of characteristics made to to humanize and help define the usability context of the system and motivations.

A Scenario is a narrative describing interactions between actors and a system. It can be use to describe a specific use case's flow of events, to outline the existing system's usage, or to illustrate a required functionality in a new system. It serves to better understand the use case and give a better view of the problem.

Persona #1 - Jacinto

Name: Jacinto

Age: 35

Occupation: IT Manager

Background: Jacinto lives alone in a smart home equipped with various IoT devices. As a busy professional, he values efficiency and convenience. He often travels for work and likes to keep an eye on his property.

Goals: Jacinto wants to remotely control his home devices and automate routines to simplify his life. He also wants to ensure his home is secure while he's away.

Pain Points: Jacinto may struggle with the setup and maintenance of the system due to his busy schedule. He may also have concerns about data privacy and security.

Persona #2 – Manuel

Name: Manuel

Age: 67

Occupation: Retired Teacher

Background: Manuel lives with his spouse in a small house. Due to their ages, they are looking for ways to make their home more accessible and easier to manage.

Goals: Manuel wants to use home automation to help manage tasks around the home, like controlling lights, adjusting the thermostat, or locking doors. He also wants to use automation to help with energy efficiency.

Pain Points: Manuel may struggle with the technical aspects of setting up and operating the system. He might also have concerns about the cost of implementing and maintaining a home automation system.

Persona #3 - Eugénia

Name: Eugénia

Age: 55

Occupation: Nurse

Background: Eugénia lives alone and has recently moved into a home with pre-installed IoT devices. She is not very familiar with technology and has minimal experience with smartphones and computers.

Goals: Eugénia wants to take advantage of the installed home automation system to make her life easier, but she's unsure where to start. She would like to control her lighting, heating, and security systems remotely.

Pain Points: Eugénia is not confident about using technology and worries about accidentally triggering the wrong functions. She also fears that she might not be able to troubleshoot issues when they arise. She needs a system that is intuitive and easy to use, with clear instructions and support available.

Scenario #1

Scenario: Smart Home Automation Setup

Jacinto, the IT Manager, is on a business trip. He decides it's the perfect time to set up his smart home automation system. He remotely connects to his home network using the webapp and with a few clicks, he configures his IoT devices to create an automated routine. He sets up the lights to turn on and off at specific times, adjusts the thermostat, and activates the security system for his peace of mind.

Scenario #2

Scenario: User-Friendly Assistance

Manuel and his wife usually spend a lot of time inside their home doing tasks and relaxing. Due to their age, Manuel would like to have as much devices set up for security as possible, for example, Manuel wants to set up their oven to make sure neither he or his wife left it turned on for too long, or have the house blinds close after dark so they don’t need to leave the house at night.

Scenario #3

Scenario: Intuitive Home Automation

Eugénia learned that her house has a smart oven that knows when it is fully preheaten and can send information to our app. She also knows that the app can turn on and off various devices, like lamps and window blinds arround her home, at the times she desires. She wants to learn how to set up so she watched a youtube tutorial and decided to try the app.

Eugénia follows the instructions and quickly learns to control her lighting and simple heating devices remotely. The system's intuitive design and user-friendly interface make her feel more confident and secure.

Product requirements (User stories)

Based on the user stories provided, here are some epics to better represent our goal in this project:

Epic 1: Remote Control and Monitoring

User Story #1: As Jacinto, I want to be able to remotely control my home devices, such as lights, thermostat, and security system, so that I can ensure my home is secure and comfortable even when I'm away on business trips.

User Story #5: As Manuel, I want to easily control the lights and thermostat in my home, so that I can manage my home environment and ensure my safety and comfort without relying on manual adjustments.

User Story #7: As Manuel, I want to make sure my home is secure while I travel to my vacation house, so that I can ensure my house is safe while I am away.

Epic 2: Automation and Routine Creation

User Story #2: As Jacinto, I want to create automation routines for my home devices, such as turning off lights and adjusting the thermostat at specific times, so that I can save energy and simplify my daily routines.

User Story #3: As Jacinto I want to create be able to map certain functions like turning on music and pausing it depending on the current time and who else is in the house.

User Story #8: As Manuel, I want to automatically turn off the lights and central heating when necessary, so that I can increase my energy efficiency and waste less money on power.

User Story #12: As Eugénia, I want to be able to set up certain tasks to specific times of the day, so that I can, for example, wake up to music two hours before the next appointment in my schedule but never after 10 am.

Epic 3: Home Security

User Story #6: As Manuel, I want to have a security system overwatching the house while I do tasks, so that I can ensure that I don't forget any important security tasks like closing the fridge or turning off the stove.

User Story #10: As Eugénia, I want the application to be easy to use so that I can be sure I made the correct decisions in setting up my house's devices and functions.

Epic 4: IoT Device Usage and Education

User Story #9: As Eugénia, I want to easily learn how to use my house's installed IoT devices so that I can take advantage of their benefits.

User Story #11: As Eugénia, I want to control simple devices and functions so that I can crack down on as many small home tasks as I can.

Epic 5: Environmental Control and Monitoring

User Story #4: As Jacinto, I want to be able to monitor and log different receivers like thermometers around the house so that I can check how different parameters like temperature varied around the day.

3 Architecture notebook

Key requirements and constrains

As a realtime data visualization system, our main architectural focus is to build a system which can provide a flow from data generation to user presentation with as little latency as possible.

Another goal is to provide easy access and documentation so inexperienced users can easilly utilize the website and more informatically knowledgeable users can even implement their own sensors.

- Provide a functional and easy-to-understand interface with appropriate documentation so that users can associate their own sensors and be able to visualize the data on the website.

- The flow from data generation to its presentation in the frontend should have low latency, not exceeding a maximum of a couple seconds (example: 5 seconds).

- Create a clean and minimalistic frontend experience (website) so that the product is readable and understandable by a large range of people.

- The Processing Layer must be able to handle receiving and sending data to/from multiple sensors without showing abnormal function or inconsistent behaviours.

- Persistence Layer must be robust so that its possible to retrieve/store data from multiple users and sensors at the same time.

- Fast and simple message broker to handle messaging between the Data Generation Layer and the Back-End Processing Layer, with support for multiple programming languages to cover the widest possible range of sensors and allows users with simple programming expertise to even make their own sensors.

Architetural view

For ease of development and coding simplicity, we decided to break down the final implementation into five broad parts:

- Data sources: These are the small IoT devices that calculate/generate the real world data that we want to analyse in the rest of the application.

  Examples: thermometer, rain sensor, weather data and cameras.

- Output devices: Like the data sources, these are IoT devices that can make execute the corresponding output actions that we tell them to.

  Examples: lights, speakers, locks, windows and outlets.

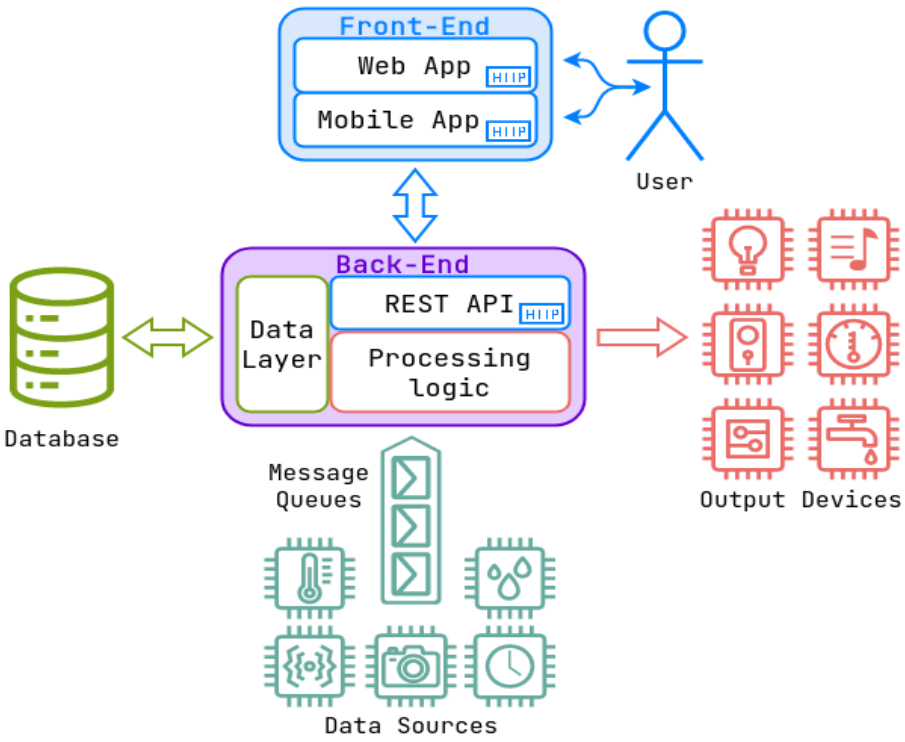
- Back-End: The main brains behind the application, here the values from the data sources aresrocessed and the corresponding output is then communicated to the output devices. The back-end receives the values through the message queues, processes them, saves the important data in the database and sends the corresponding action to the output, if needed.

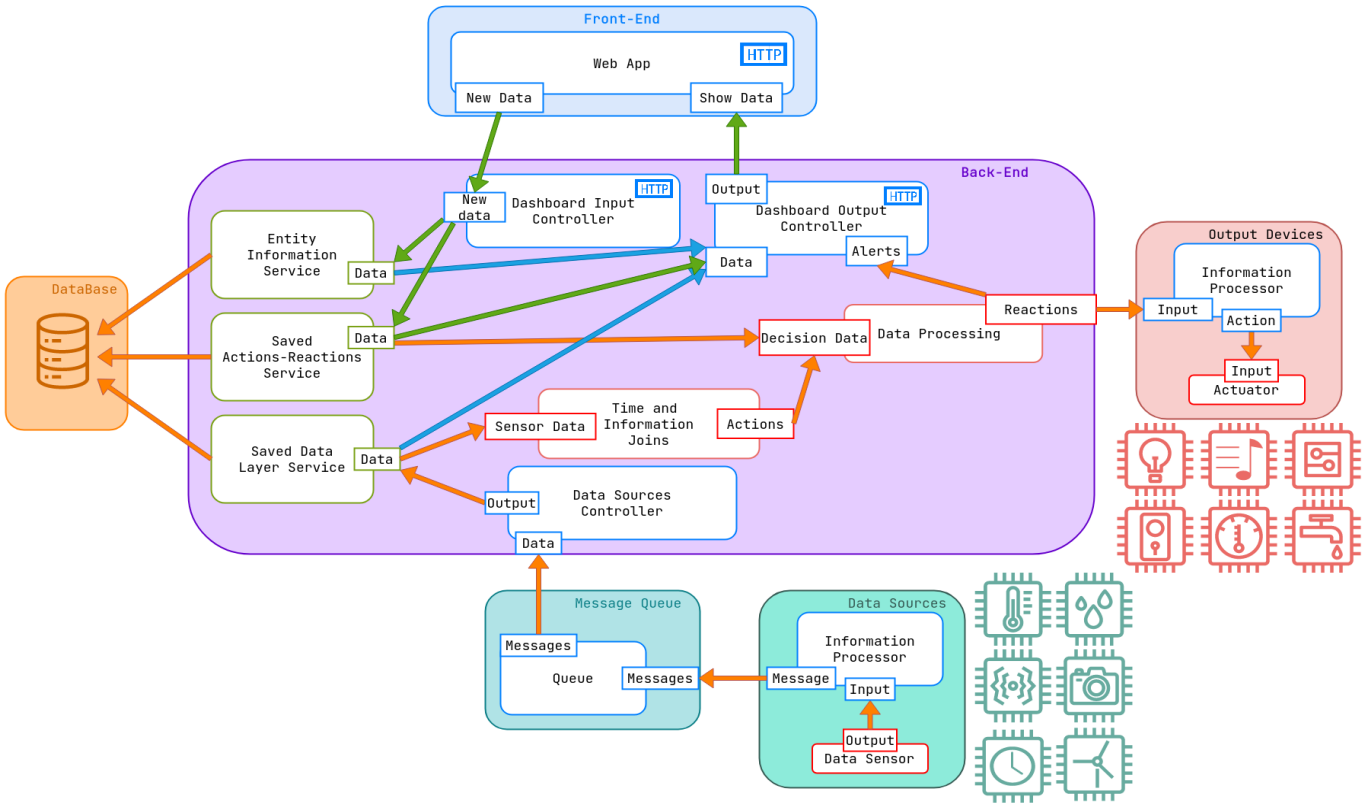
  The back-end must also give data and system management support to the Front-End in the form of a REST API so that the User can then interact with the system in a more humane way.

- Front-End: The Front-End is the main application UI that the users can interact with. It must be completely separate from the back-end and communicates with it through basic HTTP methods.

  The Front-End must be simple to use and, in this case, will be accessed completely inside a regular web browser.

- Database: This is the simple database that we will use to keep track of user information, input and output data and user made relations between it. The database must be durable and persistent to keep the app's consistency.


Front-End
Web App
HTTP
Mobile App
HTTP
User
Back-End
Data Layer
REST API
HTTP
Processing logic
Database
Message Queues
Output Devices
Data Sources


Front-End
Web App
HTTP
New Data
Show Data
Back-End
New data
Dashboard Input Controller
HTTP
Output
Dashboard Output Controller
HTTP
Alerts
Entity Information Service
Data
Saved Actions-Reactions Service
Data
Saved Data Layer Service
Data
DataBase
Data
Sensor Data
Time and Information Joins
Actions
Decision Data
Data Processing
Reactions
Output
Data Sources Controller
Data
Output Devices
Information Processor
Input
Action
Input
Actuator
Message Queue
Messages
Queue
Messages
Data Sources
Information Processor
Message
Input
Output
Data Sensor

Module interactions

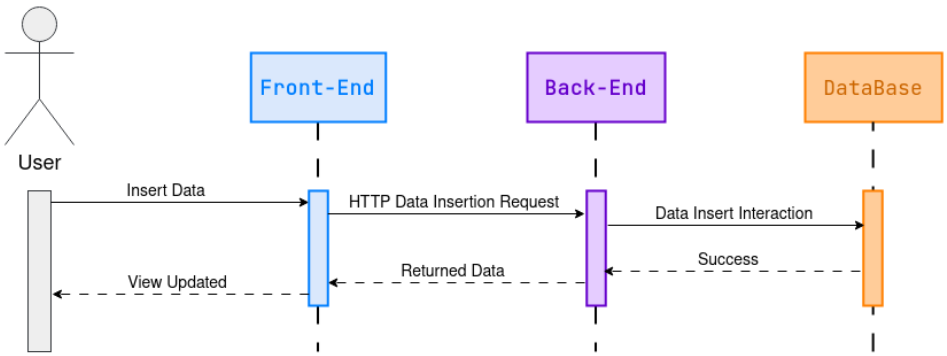
- **User input and control:**

The user's only source of communication with the application should be through the dedicated Front-End system of the application.

The front-end should then contact the back-end through a series of HTTP Requests directed to the back-end's REST API endpoints.

The back-end should then have the responsiblility of parsing and processing the given data as well as updating the database records to the newer state.

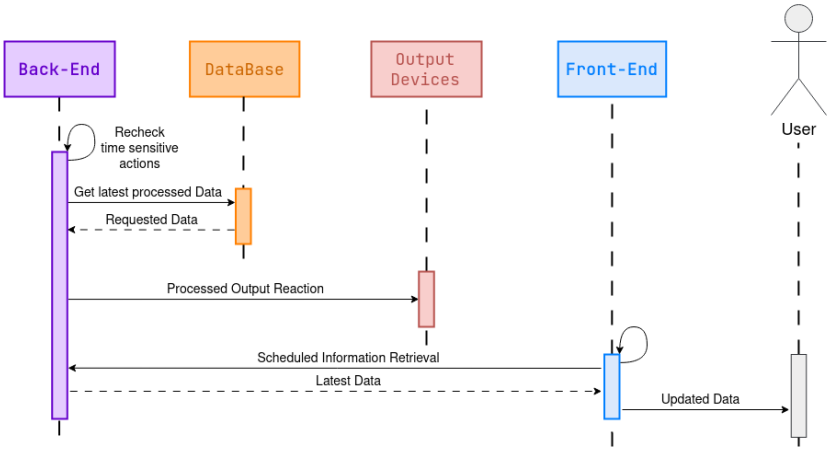
The required information is then passed back up the chain so the user can view the new state represented in the front-end.



- **Time sensitive output reaction:**

At every time interval (ex: each minute), the back-end should query the current database to check if the current state of the list of sensor data allows for a user defined to be executed. For example, if a user defines that he wants music to turn on whenever he is inside the house after 18h, the back-end should check if these requirements are met every minute or whenever any of the related pieces of information (users in the house, time of day) changes.

If the check returns a sucessful state, the corresponding output is sent to the defined output devices and the next time the front-end queries for new information the updated state should be returned so the user can get the most accurate representation of the whole service's current state.



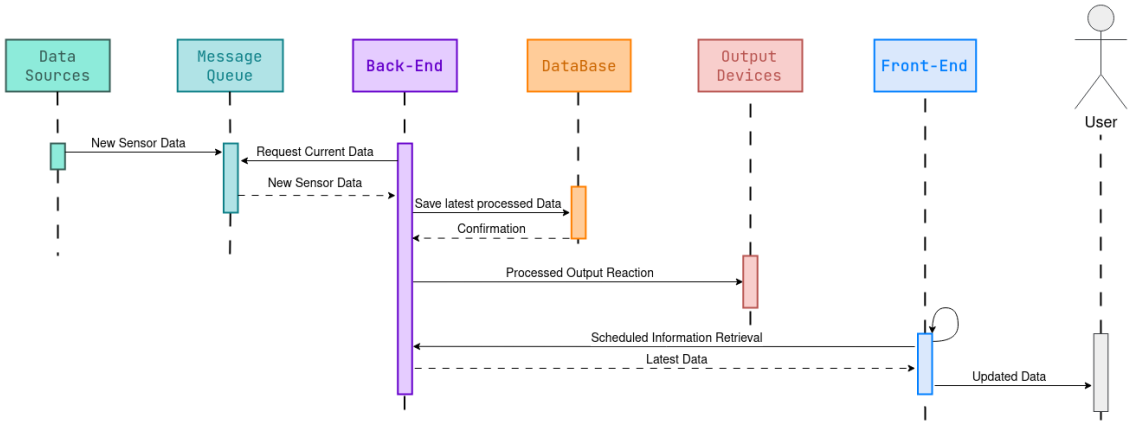
- **New sensor data input reaction:**

After generating a new piece of raw data, the connected sensors register that information inside a Message in the Message Queue.

The back-end gets notified of the new message and starts processing the associated information.

This information is then updated inside the database and an action check is made to see if any action is programmed for the any output device after the data change.

The front-end's next valid data request must be responded with the new system state, so the user can have the most updated data representation possible.



Intended technologies

For this project, we decided to use technologies that we already have a grasp of, while making sure that they are adequate for the requirements of the product.

Back-End Components:

Spring Boot – Robust and easy framework to build Restfull APIs, avoiding boilerplate code and promoting a modular and scalable architecture.

RabbitMQ Message Broker - Efficient and reliable messaging middleware that facilitates communication between microservices in a distributed system. RabbitMQ's support for multiple messaging patterns and programming languages makes it suitable for our wanted ease-of-use and language agnostic device integration.

MongoDB Database - A NoSQL database that provides flexibility in handling unstructured data and is particularly well-suited for applications with rapidly evolving schemas. MongoDB's document-oriented model allows for seamless storage and retrieval of complex data structures, as well as keeping records of data with volatile structures, allowing dynamic value structures without being forced to use a pre-defined table structure for the rest of the project's lifespan or spending time on making new value structures cross-compatible with older values.

Frontend Components:

React - A powerful JavaScript library for building user interfaces. React is widely used for developing interactive and dynamic frontend applications. Its component-based architecture and virtual DOM enable efficient updates and seamless user experiences without constant full page updates.

Vite - A fast and modern build tool that enhances the development experience for frontend applications. Vite focuses on speed during development by leveraging native ES module imports and providing a highly performant development server. It supports features like hot module replacement (HMR) for quick feedback and optimized build times for deployment.

NGINX - A fast, scalable and hassle-free HTTP server that we used as a proxy between client requests to the final production server and the Vite implementation. NGINX allows us to quickly set up the final implementation without messing with complicated distribution protocols.

Data Sources:

Java – For easier testing and integration with the rest of our code, we used a set of Java devices to act as virtual data-sources that simulate sensors. These were implemented as "default" sensors inside the maven built back-end, but are totally independent from the spring implementation.

Bash, Python and Crystal – For demonstration purposes, we built some devices using these three languages to prove that, as long as the provided language is RabbitMQ compatible, input sensors of all makes and models can be integrated alongside our application, without the need to hardcode/rewrite the back-end data and device processing.

Production implementation

As a final demonstration, we implemented the final product inside a provided server that simulates an actual production environment.

For this purpose, we joined all of our work inside a set of dockers managed by a docker-compose.

We utilized four main dockers in our final implementation:

Front-End – This docker contains all the necessary front-end processes to allow users connecting to our machine to access and manipulate the system given the provided website.

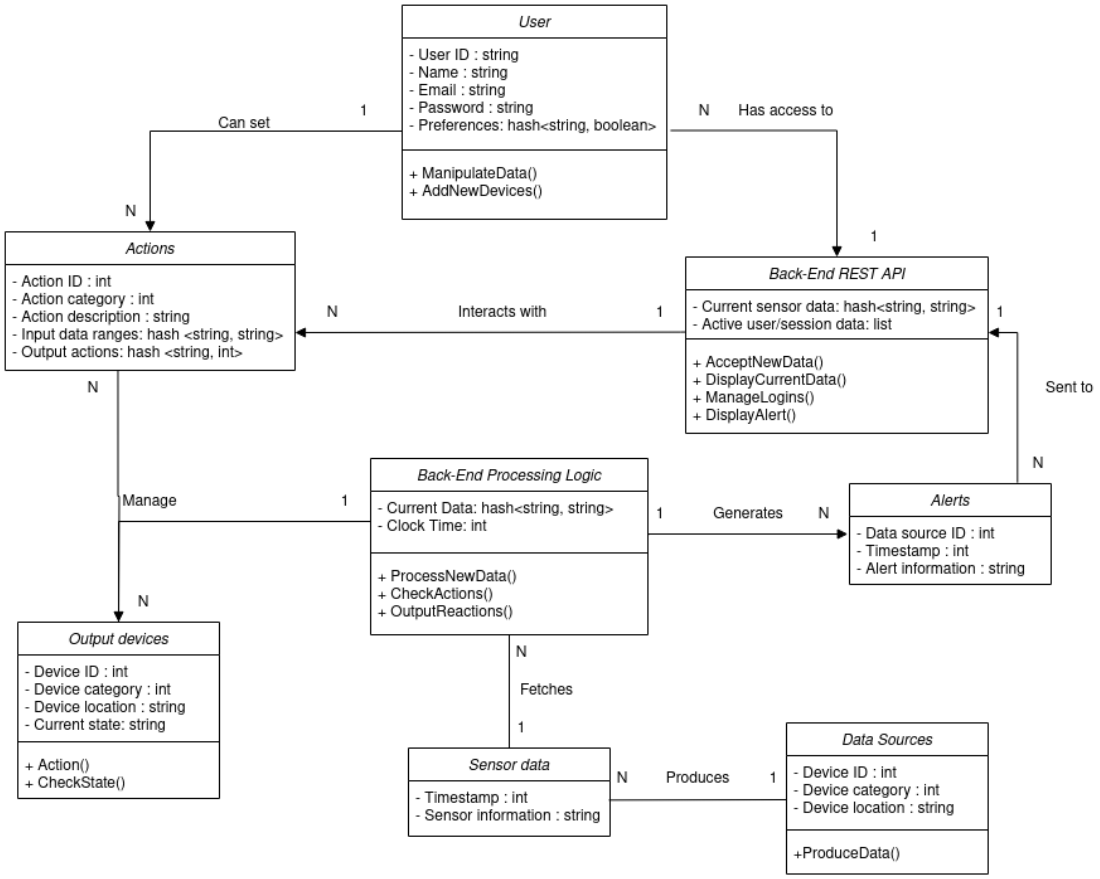
Back-End – The back-end docker is responsible for the main service implementation, creating the corresponding spring-boot application and some example devices.

Mongo – As the name sugests, this container is responsible for creating and providing the main mongoDB database for our project to interact with.

RabbitMQ – Finally, the RabbitMQ container provides the intermediant message queue that devices use to communicate with the back-end. Even the default sensors integrated inside the back-end container first pass their messages through the RabbitMQ implementation.

These communicate through two distinct networks, one between the front and back ends of the application and another for bridging the database, devices, rabbitMQ and spring-boot input data processes.

4 Information perspetive



Entities

User:

Represents a typical user of the product. Has its own personal information along with a unique user ID to identify the user in the system and a set of defined preferences.

He has the ability to manipulate data and add new data sources or output devices by interacting with the REST API in the frontend.

Actions:

Specific user defined actions that the system can perform.

They have a ID and a Category so they can be easily identifiable, a description of the action, the input data ranges that control in what situations this action will be triggeredand the specific output action(s) that should happen when this action gets executed.

Back-end Rest API:

The middlepoint between the user interaction with the frontend and the business logic of system. Responsible for updating the frontend with the latest information and also handling user requests to the processing layer, such as logging in, defining a new action or adding a new data source.

All new user data must also be parsed here.

Back-end Processing Logic:

Central processing point of the system.

Is responsible for fetching the latest all the new values from the data sources and verifying if any action should be triggered, aswell as generating alerts in case of an action trigger or abnormal values.

This entity also controls the interaction with output devices and the database, keeping control of user-defined actions and guaranteeing that the database is always updated with the most recent data values.

Alerts:

Alerts may happen in the case that data provided by the data sources falls within unusual values (warning or error alert) or an action on an output device was performed (informational alert). The range for unusual values may be system-defined (default) but can also be defined by the user (overwritten).

Output Devices:

Devices which upon Actions will act on. This may be for example a lamp or a television, that when a action is triggered may change state, for example turning on/off.

Sensor Data:

Information provided by the various data sources. Contains the timestamp to when it was created and also the respective information, being indexed by the respective data source ID.

Data Sources:

Responsible for generating the data to feed the system (would tipically be sensors). Identified by their unique ID, category and location (x, y and floor number). Have the ability to produce the Sensor Data which will be forwarded to the Back-end Processing Layer trough a messaging server.

5 References and resources

https://spring.io/projects/spring-boot
https://www.mongodb.com/compatibility/spring-boot
https://www.rabbitmq.com/tutorials/tutorial-one-java.html

https://react.dev
https://vitejs.dev
https://daisyui.com
https://nginx.org/en/