Technical Report - **Product specification**

# HomeMaid

Course: IES - Introdução à Engenharia de Software

Date: Aveiro, 7 de outubro de 2024

Students: 109061 - Ângela Ribeiro
114220 - Rita Silva
114129 - Regina Tavares
113889 - Hugo Castro

Project abstract: HomeMaid is a system for monitoring and managing devices and conditions in smart houses. Using consumption sensors, the platform collects real-time data and identifies anomalous conditions. The system allows users to remotely monitor and control the environment through a web portal.

Table of contents:

# 1 Introduction

HomeMaid is a cutting-edge web application designed to provide homeowners with an intuitive, seamless way to monitor and control their homes. As modern technology continues to advance, the demand for convenience and automation in daily life has grown significantly. People today expect their homes to integrate with their lifestyles, offering solutions that save time and simplify routine tasks. HomeMaid rises to meet this expectation by delivering an easy-to-use, highly interactive platform that empowers users to manage their household environments with minimal effort.

Our app transforms the concept of smart homes into a reality, giving users the ability to control various aspects of their living spaces—from lighting and temperature to appliances—all from a single interface. With just a few clicks, HomeMaid turns the complexities of home management into simple, streamlined actions.

The vision behind HomeMaid is to alleviate the burden of repetitive household chores and enhance everyday comfort through automation. We aim to redefine how people interact with their homes, offering features that not only make life easier but also give users more time to focus on what truly matters. Whether it's scheduling the perfect ambiance for a relaxing evening or ensuring that everything is deactivated while you’re away, HomeMaid simplifies these tasks, delivering a smarter, more efficient living experience.

By harnessing the power of modern technology, HomeMaid opens the door to a future where your home is more than just a place to live—it's a dynamic, responsive space that works with you, for you.

# 2 Product concept

## Vision statement

HomeMaid is a web-based application designed to give homeowners an intuitive platform to monitor, control, and automate various aspects of their household environment. At its core, the system will be used to enhance the convenience of home management, allowing users to control devices like lights, temperature settings and even household appliances. Through the HomeMaid interface, users will interact with smart home devices without needing to understand the technical underpinnings, making the system user-friendly and accessible to all.

The high-level business problem that HomeMaid addresses is the increasing demand for automated home management solutions that simplify everyday tasks. In today’s fast-paced world, people are seeking ways to streamline mundane chores, reduce the amount of manual intervention required in household management, and improve the overall quality of life by optimizing how they interact with their living spaces. HomeMaid

solves this problem by offering a comprehensive home automation platform that makes controlling a smart home easier, more efficient, and accessible to a broader audience.

Initially, our vision for HomeMaid included the integration of a voice-control feature, allowing users to manage their homes via voice commands. However, due to technical challenges and the desire to focus on delivering a stable and intuitive visual interface first, we opted to delay this feature for future iterations of the product. Instead, we prioritized a responsive web interface that can be accessed from both desktop and mobile devices, ensuring that users can control their home systems remotely with ease. By shifting focus to perfecting the UI/UX experience and integration with existing smart home devices, we have laid a stronger foundation for potential future enhancements, such as voice control.

HomeMaid is similar to other smart home management systems like Google Home or Amazon Alexa in its goal of providing a centralized platform for controlling a variety of home devices. However, HomeMaid differentiates itself by offering a more customizable, user-centric interface that emphasizes simplicity without sacrificing control. Unlike some well-known platforms that require users to invest heavily in proprietary devices, HomeMaid aims to be compatible with a broader range of existing smart home devices, making it a more versatile option for users who may have a variety of smart home products from different manufacturers.

Additionally, while some platforms focus heavily on voice control or ecosystem lock-ins, HomeMaid emphasizes device independence, offering users the freedom to integrate and control devices across various brands and technologies with ease.

## Personas and Scenarios

Persona 1:

Name: Maria
Age: 38
Location: Espinho
Job Title: Marketing Manager
Income: Upper Middle Class

As a single mother of two children, Maria juggles the demands of her job while caring for her family. Safety and peace of mind remain her top priorities, particularly when her children are home alone after school. In addition to ensuring their safety, Maria wants to stay informed about activities in the house, including when home automation systems are triggered, like the air conditioner or television being turned on.

Goal**:**Maria wants to be aware of when automations in her house are set off, such as someone turning on the air conditioner, TV, or other devices. She wants timely notifications about these activities to ensure she knows what is happening at home and to verify that her children are responsibly using devices.

Pain Points**:**Maria often feels disconnected and anxious because she cannot track which automations are being used or triggered in her house. Her frustration stems from not having a centralized way to monitor these events easily, as managing multiple devices and systems simultaneously becomes overwhelming. She desires a streamlined solution to keep her updated without adding to her already full schedule.

Persona 2:

Name: Catarina
Age: 32
Location: Penafiel
Job Title: CEO
Income Level: Upper Class

With a busy schedule filled with meetings, work trips, and social commitments, Catarina values efficiency and looks for solutions that save her time and effort. For Catarina, automating all kinds of repetitive tasks is crucial to simplifying her daily life.

Goal: Automate repetitive and tedious tasks, such as adjusting the temperature, turning off lights, and playing music at specific times, to simplify her daily life.

Frustrations: She experiences significant frustration when she loses time on tasks that could easily be automated. It is common for her to forget to adjust the home settings before leaving for work, leaving her dissatisfied with the manual processes that consume her precious time.

Persona 3:

Name: João
Age: 35
Location: São Miguel
Job Title: IT Professional
Income Level: Upper Class

Proud of maintaining order in every aspect of his life, João values full control over his environment and dislikes unpredictability. João is detail-oriented and loves having complete visibility of everything in his life while also being eco thoughtful.

Goal: Monitor his house at any given moment and be able to correct something that is wrong (e.g., an appliance left on). He also wants to be able to improve his house to be as eco-friendly as possible.

Frustrations: His frustrations arise from a lack of real-time updates or delays in receiving notifications. This uncertainty and the occurrence of unexpected issues when he is away from home heighten his anxiety and dissatisfaction, as he wishes to monitor his house at any moment and act swiftly if something goes wrong.

Scenario 1:
Maria recently changed jobs and now works an hour away from home, compared to her previous job that was just 10 minutes away. Due to this change, Maria won't be home when her children arrive from school. Maria needs to stay informed about what is happening at home, especially when her children are using devices. She wants to receive notifications whenever home automations, such as the air conditioner, TV, or other devices, are triggered. This helps her ensure that her children are safe and responsibly using the devices until she gets home.

Scenario 2:
Catarina has been very occupied lately and has been sleeping less, so in the morning she wants her coffee to be made automatically at 5:30am. She also wants her walk-in closet to be warmed up by the morning (5:00am) so her clothes aren't cold when she needs to dress fast. At night she wants the app to set her water running so her bath is ready when she gets there (9:00pm).

Scenario 3:
João's vacuum cleaner has been malfunctioning for some days, the device turns on itself and this has been worrying him because it's wasting energy. Due to this, he wants to receive an alert as soon as this happens so he can turn it off.

## Product requirements (User stories)

Epic 1: Monitoring
User Story 1: As João, I want to check the status of all devices and ambience in my home instantly, ensuring that nothing is left on unintentionally and the atmosphere is perfect. (High)

Epic 2: User Automation
User Story 2: As Catarina, I want to set up an automation to my coffee machine a certain hour to ensure that I save time. (Medium)
User Story 3: As João, I want to be able to delete automations whenever I see fit, so I can have better control over my home and adapt to changes in my routines or preferences. (Medium)

Epic 4: Notifications
User Story 4: As João, I want to be able to select which notifications I receive about my home, so I can focus on the information that matters most to me. (Medium)

User Story 5: As Maria, I want to mark notifications as read after reviewing them, so I can keep track of what I've already seen. (High)

User Story 6: As Catarina, I want to delete notifications I no longer need, so I can maintain a clean and organized notification history. (Low)

User Story 7: As João, I want to receive notifications when devices in my home start or stop, so I can stay informed about their activity and usage. (High)

Epic 4: User Account Management

User Story 8: As Maria, I want to log in to my account quickly and securely and be able to add and remove devices to my account, so that I can customize my smart home system as my family's needs change. (High)

Epic 5: Device Control

User Story 9: As Catarina, I want to control all the devices in my home from one app, so I don't waste time switching between different apps. (High)

Epic 6: Integration with external Resources

User Story 10: As Maria, I want to integrate my smart home system with Alexa, so that I can control my home through voice commands when I'm busy with my children. (Low)

Epic 7: Data Backlog

User Story 11: As João, I want to see a detailed breakdown of my home's usage trends, so I can identify any spikes or areas where energy is being wasted. (Medium)

Epic 8: Account Security

As Catarina, I want my account to be secured, so I can ensure that only I have access to control my home automations. (Low)

# 3 Architecture notebook

## Key requirements and constraints

The HomeMaid system is designed to provide users with an efficient and secure platform

for managing smart home devices. To ensure the system meets user expectations and functions effectively in real-world scenarios, several technical and functional requirements have been identified:

- The HomeMaid system must support a wide range of devices.
- Since HomeMaid handles sensitive information (such as the status of doors and security cameras), it is crucial to ensure that all communications between users and devices are secure.
- Users should be able to remotely access the HomeMaid system via internet-connected devices, such as smartphones or computers.
- The system must be designed to handle multiple users and devices simultaneously, ensuring it can scale as more devices are added
- HomeMaid should allow users to create custom automation rules

## Architetural view

The HomeMaid system will utilize **virtual sensors** simulated by agents in **Python** to represent devices like temperature and humidity sensors. These sensors generate data in real time and send it to **Kafka**, a message broker that routes the data through queues. The data is then stored in a **InfluxDB** database through a middleware built with **Spring Boot**, which also handles requests for historical and aggregated data via a **REST API**.

In addition to sensors, **output devices** like smart lights and thermostats allow users to control home automation in real time. These devices execute commands received from users via the **Spring Boot** middleware. For instance, users can turn on lights or turn down the air conditioner temperature, and the output devices will respond to these commands. Real-time updates are sent back to the front end via **WebSockets**, ensuring users receive immediate feedback when device states change. The system also includes a background **thread** to manage scheduled automations, ensuring devices execute predefined actions at specified times.

Security is managed through **Spring Boot** using **JWT authentication**, with tokens stored in the browser's local storage. Access to sensitive endpoints is restricted to admin users, and passwords are securely encrypted using **Bcrypt**.

The front end, developed with **ReactJS**, **Vite**, **TailwindCSS**, and **DaisyUI**, focuses on a sleek and user-friendly interface. Real-time updates are sent back to the front end via **WebSockets**, ensuring users receive immediate feedback when device states change. All traffic between the **Web App** and **Spring Boot** passes through an **Nginx proxy**.

In summary, HomeMaid integrates **virtual sensors for monitoring**, **output devices for control**, a message broker, secure authentication, and a responsive front end to provide a scalable smart home management solution..

## Module interactions

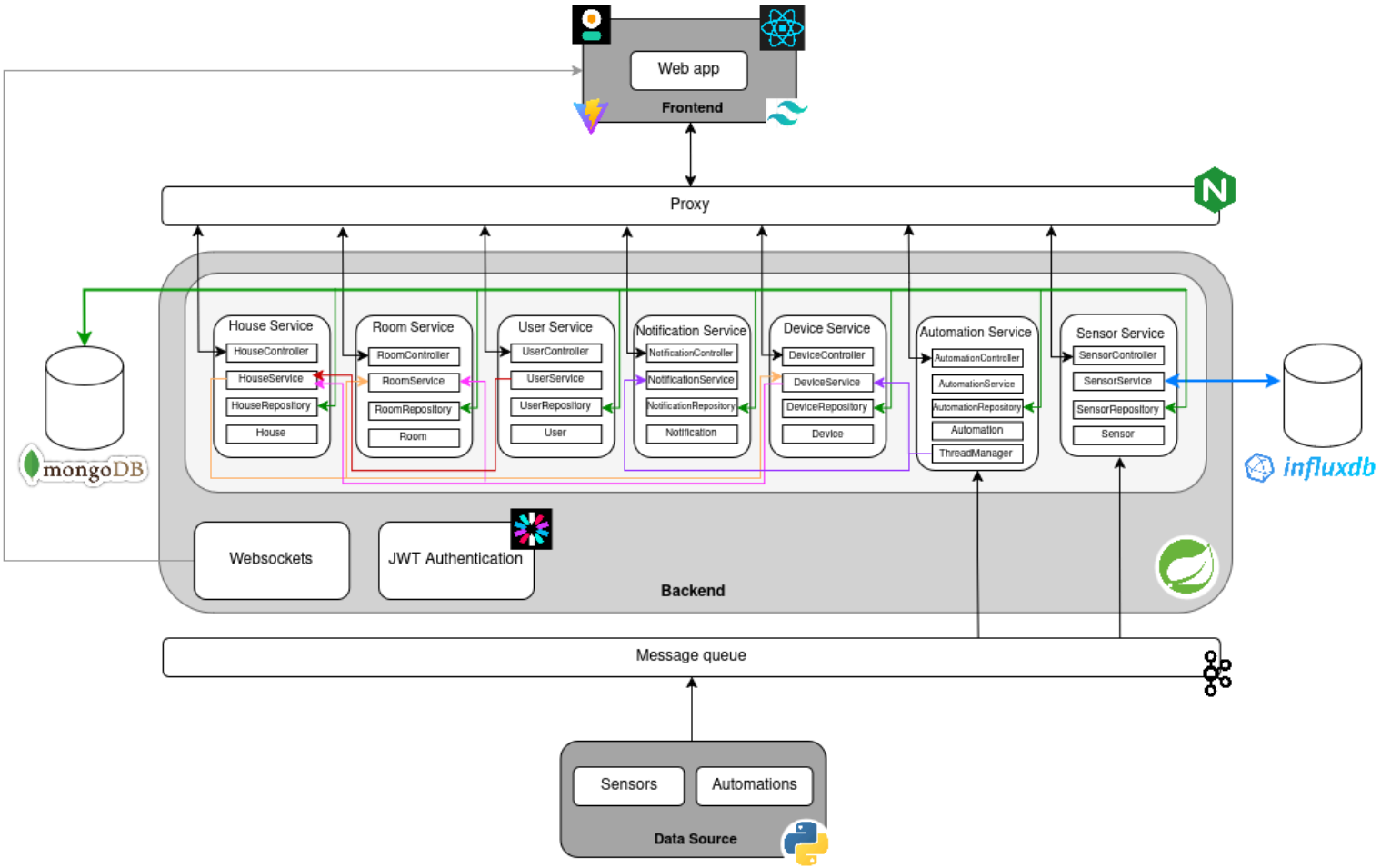
The HomeMaid system consists of interconnected modules that communicate seamlessly to provide a robust and scalable smart home management solution. The key interactions between these modules are as follows:

1. **Virtual Sensors and Kafka**:
   Virtual sensors, simulated by Python agents, generate real-time data such as temperature and humidity. This data is sent to Kafka, a message broker responsible for routing the information through queues. Kafka ensures reliable and efficient delivery of the data to downstream systems.
2. **Spring Boot Middleware and databases:**
   The Spring Boot middleware serves as the central orchestrator of the system. It subscribes to Kafka topics to consume sensor data, processes it, and stores it in a MongoDB database for device and user-related data. For time-series sensor data, it utilizes InfluxDB, which is optimized for handling large volumes of time-stamped data. The middleware exposes a REST API for clients to access aggregated sensor statistics, such as average temperature and humidity, calculated over specific timeframes (daily, weekly, or monthly).
3. **Output Devices and Spring Boot Middleware:**
   Output devices such as smart lights and thermostats receive control commands from users via the Spring Boot middleware. These commands are processed and relayed to the respective devices, enabling real-time control. The middleware ensures that device states are updated promptly and correctly.
4. **Scheduled Automations and WebSockets:**
   A background thread within the Spring Boot middleware manages scheduled automations. It periodically checks for predefined automation rules and executes the corresponding actions by interacting with the device modules. When an automation is executed, the system updates the state in real time using WebSockets, ensuring immediate synchronization across all connected clients. Notifications are also generated to inform users about automation activations.
5. **Security with JWT Authentication**:
   All user authentication is handled through the Spring Boot middleware using JWT (JSON Web Token) authentication. Tokens are generated upon successful login and securely stored in the browser's local storage. JWT tokens are verified for access to sensitive endpoints. Passwords are encrypted using Bcrypt for enhanced security.
6. **Frontend:**
   The front end communicates with the Spring Boot middleware through REST API endpoints for data retrieval and WebSockets for real-time updates. The sleek and user-friendly interface allows users to monitor sensor data, control devices, and receive notifications. All communication between the frontend and backend passes

through an Nginx proxy, ensuring load balancing and improved security.

7. **Nginx Proxy:**

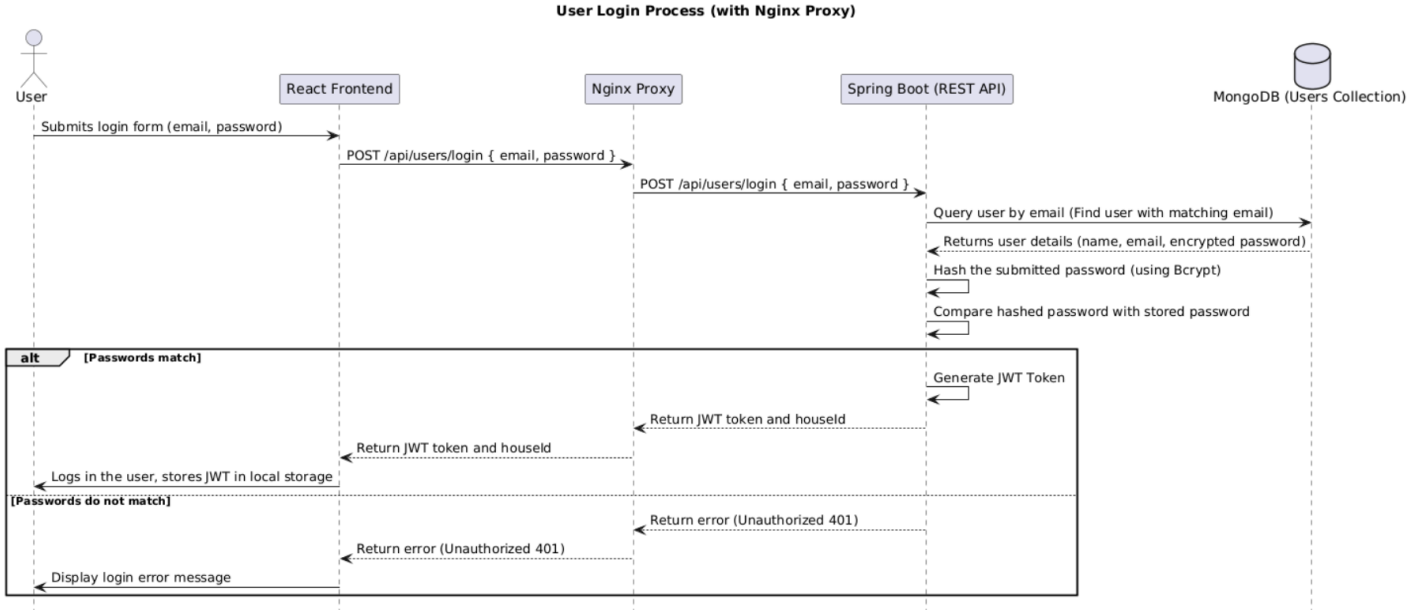
   The Nginx proxy acts as a gateway between the Web App (frontend) and the Spring Boot middleware. It routes client requests to appropriate services, handles load balancing, and enhances security by managing traffic flow.



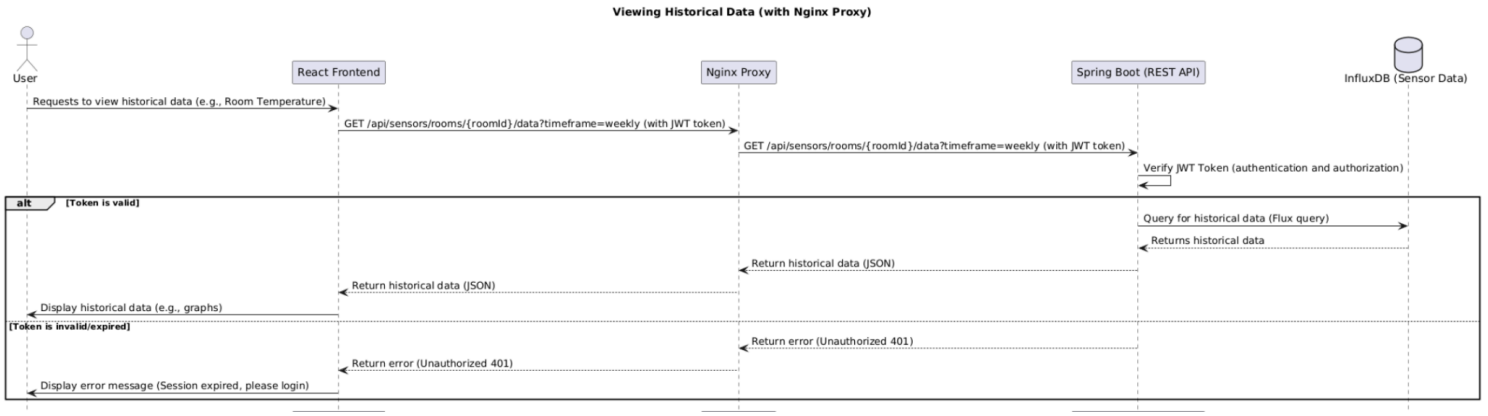

Examples of interactions between the modules:

- **User Login & JWT Authentication**
  1. User submits login credentials via the Frontend.
  2. The Frontend sends a POST request to Nginx at /api/users/login.
  3. Nginx forwards the request to the Spring Boot /login endpoint.
  4. Spring Boot verifies the user's credentials by querying MongoDB for user data.
  5. Spring Boot hashes the submitted password and compares it with the stored password.
  6. If the credentials are valid, Spring Boot generates a JWT token.
  7. Spring Boot returns the JWT token and houseId to Nginx.

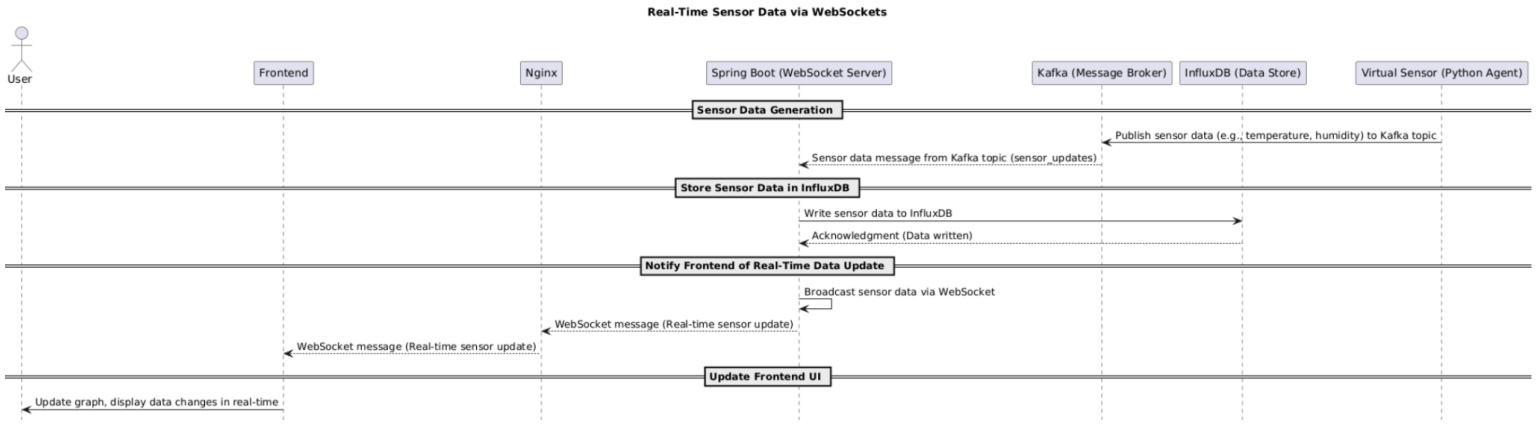
8. Nginx forwards the JWT token and houseId to the Frontend.
9. The Frontend stores the JWT token in local storage for use in future authenticated requests.



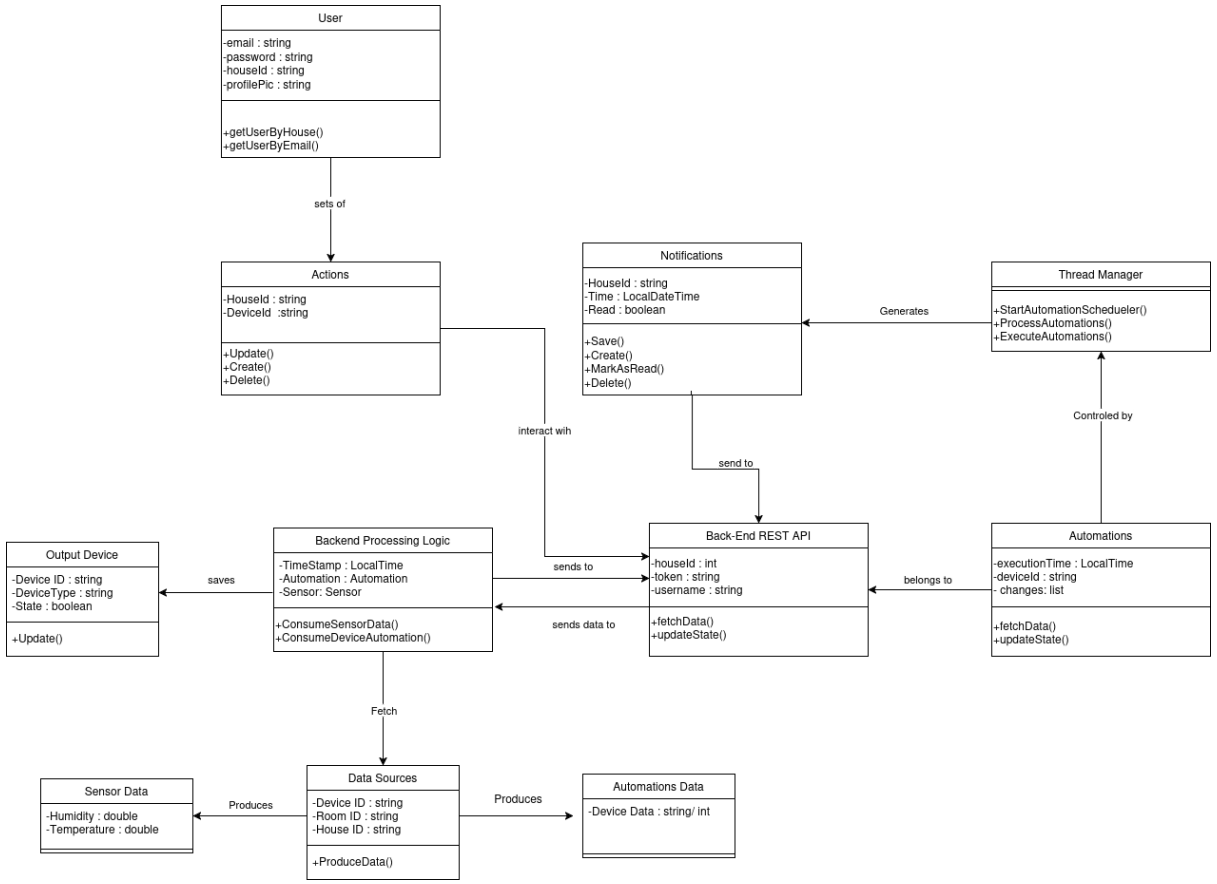

- **Viewing Sensor Data (Historical Data via REST API)**
    1. User requests aggregated sensor data (e.g., "weekly average temperature") via the Frontend.
    2. The Frontend sends a GET request to Nginx at /api/sensors/rooms/{roomId}/average-temperature?timeframe=weekly, attaching the JWT token in the request headers.
    3. Nginx forwards the request to the Spring Boot endpoint for retrieving aggregated data.
    4. Spring Boot validates the JWT token to ensure the user is authenticated.
    5. If the token is valid, Spring Boot queries InfluxDB for the specified sensor's historical data using a Flux Query.
    6. InfluxDB returns the relevant data (e.g., average temperature for the last 7 days) to Spring Boot.
    7. Spring Boot processes and formats the data into a usable JSON response.
    8. Spring Boot sends the response to Nginx.
    9. Nginx forwards the JSON response to the Frontend.
    10. The Frontend displays the aggregated data (e.g., as a graph) to the User.




- **Real-Time Sensor Data Flow**
    1. A virtual sensor (via a Python agent) generates new sensor data (temperature, humidity, etc.).
    2. Kafka acts as a message broker that forwards the sensor data to Spring Boot.
    3. Spring Boot subscribes to the Kafka topic and receives the sensor data
    4. Spring Boot writes the sensor data to InfluxDB for future querying.
    5. Spring Boot broadcasts the new sensor data to all WebSocket clients (i.e., the user on the Frontend).
    6. The Frontend receives the WebSocket message and updates the UI in real-time (e.g., graph, card, or notification).



# 4 Information perspetive




1. User

- Represents the system's end-user.
- The user is responsible for defining **Actions** and interacting with the **Back-End REST API**.
- **Relationship**:
    - The user **sets up** and configures **Actions**.
    - Interacts with the **Back-End REST API** to manage the system.

2. Actions

- Actions are user-defined tasks that the system can perform.
- They contain specific configurations, including input triggers and corresponding outputs.
- **Relationship**:
    - Actions **interact with** the **Back-End REST API**, which processes them.
    - Actions are tied to **Output Devices** through triggers and results.

3. Back-End REST API

- Serves as the intermediary layer between the user and the system's business logic.
- It processes user input, updates the frontend, and forwards data to the **Backend Processing Logic**.
- **Relationship**:
    - Receives input data and requests from the **User** and **Actions**.
    - Sends processed data to the **Backend Processing Logic**.
    - Forwards control to **Automations** for triggered tasks.

4. Backend Processing Logic

- The core processing engine of the system.
- Responsible for analyzing input data, validating conditions, and executing tasks.
- **Relationship**:
    - Fetches data from **Data Sources**.
    - Saves changes and outputs to **Output Devices**.
    - Sends processed information to the **Back-End REST API**.
    - Generates **Alerts** when abnormal data or trigger conditions occur.

5. Notifications

- Represent alerts or messages generated as a result of backend processing.
- Notifications may include warnings, errors, or informational messages.
- **Relationship**:
    - **Generated by** the **Backend Processing Logic** when unusual conditions are detected or an action is performed.

6. Thread Manager

- Manages the execution of automated tasks, ensuring timely and efficient handling of processes.
- **Relationship**:
    - **Controls** the execution of **Automations**.

7. Automations

- Automations represent predefined logic or routines that are executed based on triggers.
- **Relationship**:
    - **Belong to** the **Backend Processing Logic**, which decides when to trigger automations.
    - Are **controlled by** the **Thread Manager** for execution.
    - Automations are tied to **Automations Data**, which contains specific configurations.

8. Output Devices

- Represent devices (e.g., lamps, televisions) that change their state as a result of **Actions** being executed.
- **Relationship**:
    - **Saves** states and outputs defined by the **Backend Processing Logic**.

9. Sensor Data

- Represents the information generated by **Data Sources** (e.g., sensors).
- Each data entry includes timestamps and values.
- **Relationship**:
    - **Produced by Data Sources**.
    - Is forwarded to the **Backend Processing Logic** for analysis.

10. Data Sources

- Represent origins of input data, such as sensors or IoT devices.
- They generate **Sensor Data** that feeds into the backend for processing.
- **Relationship**:
    - **Produce Sensor Data**.
    - The data is **fetched by** the **Backend Processing Logic**.

# 5 References and resources

https://www.influxdata.com/blog/visualizing-time-series-data-chartjs-influxdb/https://www.influxdata.com/blog/visualizing-time-series-data-chartjs-influxdb/

https://www.baeldung.com/java-influxdb

https://youtu.be/NoYParYMWcs?si=Qp5DKRMnzt4QA6ue

https://www.influxdata.com/blog/getting-started-java-influxdb/

https://github.com/InfluxCommunity/SpringBoot-InfluxDB-Demo/tree/main

https://www.w3schools.com/java/java_threads.asp

https://jwt.io/introduction

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications