# **Robot Management System**

Mester Darius 30431

### **Project Specification**

## **Introduction**

This project is a **Robot Management System** designed to help users manage their robots and the tasks of their robots. The system allows users to **create, update, delete, and assign tasks** to robots; the admin can still do any CRUD operation on anything.

## **Main Features**

### **User Management**

* **Admins** can create, update, and delete users.
* **Regular users** can create robots, but they cannot modify other users' robots.

### **Robot Management**

* Users can **add, edit, and remove** robots.
* Each robot is assigned to a specific user (the creator).

### **Task Management**

* Admins can assign tasks to robots.
* Users can track robot activity and progress.

### **Security & Access Control**

* **JWT authentication** ensures only authorized users can access certain features.
* Different permissions for **admins** and **regular users**.

### **AI Code Generation for Microcontrollers**

* Users can generate microcontroller code using AI (ChatGPT).
* The system automatically uploads the generated code with wire transmission.

## 

## **Technology Stack**

|  |  |
| --- | --- |
| **Component** | **Technology Used** |
| **Backend** | Java (Spring Boot) |
| **Frontend** | Swing (initially), migrating to a web frontend |
| **Database** | PostgreSQL |
| **Security** | Spring Security + JWT |
| **Testing** | JUnit |
| **Automation** | Selenium for web automation |

# **Functional Requirements**

## **1. User Management**

**Admin Features:**

* Create, read, update, and delete users.
* View a list of all users.

**Regular User Features:**

* Can only create robots.
* Cannot modify or delete other users.

## **2. Robot Management**

**Admin Features:**

* Create, edit, delete, and assign robots to users.
* View all robots in the system.

**Regular User Features:**

* Create robots (ownership is automatically assigned).
* View their own robots.
* Cannot modify or delete robots they don’t own.

## **3. Task Management**

**Admin Features:**

* Create, assign, and delete tasks for robots.
* View task status updates.

**Regular User Features:**

* View assigned tasks for their robots.
* Track task progress.

## **4. Authentication & Authorization**

* Users must **log in** using **JWT authentication**.
* **Role-based access control** (Admin vs. Regular User).
* Passwords are **hashed and stored securely**.

## **5. AI-Generated Microcontroller Code**

**User Features:**

* Users provide verbal instructions to **generate robot code** using AI (ChatGPT).
* The system **uploads the generated code throught micrUsb connection.**

## **6. Web Automation for Robot Interaction**

**Selenium Automation:**

* The system automates web interactions **(e.g., uploading generated code to microcontrollers)**.

## **7. File Management**

**Robot Firmware Handling:**

* Users can **upload robot firmware files**.
* The system finds and **copies the latest firmware file** from the **Downloads** folder to the correct robot storage location.

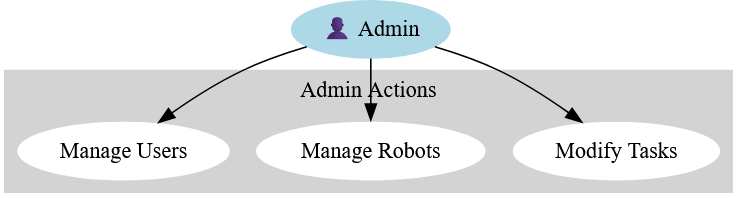
## **8. Database & Persistence**

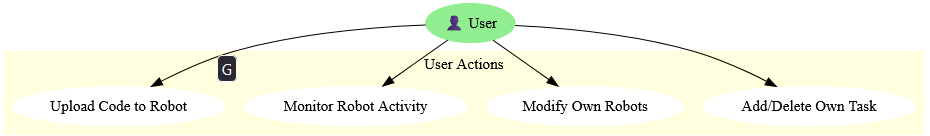
* All users, robots, and tasks are **stored in PostgreSQL**.
* **JPA (Hibernate)** handles database interactions.

## **9. Testing & Reliability**

**Automated Testing:**

* **JUnit tests** for user, robot, and task management.
* **Mockito for service testing**.





**Non-Functional Requirements**

## **1. Security**

**JWT Authentication** for secure user sessions.  
 **Role-based access** to restrict user actions.  
 **Data encryption** for storing passwords (BCrypt hashing).

## **2 . Usability & User Experience**

UI is **simple and intuitive**.  
 Users are able to perform actions with **minimal explanations**.  
 Feedback is be provided for user and admin actions (ex: success/error messages).

## **3. Integration & Compatibility**

The system integrates with:

* **AI API (OPENAI)** for code generation.
* **Microcontroller ( Micro:bit)** for code deployment.
* **Web automation tools (Selenium)** for robot interactions.  
   The system works on **Windows (tested)** without issues**, and to be tested on macOS and Linux.**

### **Design Constraints**

Here are some rules and limits we need to follow while building the system:

1. **Programming Languages & Frameworks**
   1. The **backend** must be built using **Java Spring Boot**.
   2. The **frontend** will start with **Swing**, but later we’ll migrate to a **web-based UI**.
   3. We must use **Spring Data JPA** for database interactions.
2. **Database Requirements**
   1. The system must use **PostgreSQL** as the database.
   2. We are required to use **JPA (Java Persistence API)** for managing data.
3. **Security Requirements**
   1. Authentication must be handled using **JWT (JSON Web Tokens)**.
   2. Passwords must be **hashed and stored securely**.
4. **Development Tools & Environment**
   1. We must use **Maven** for project dependencies and build management.
   2. The code should be tested using **JUnit**.
   3. **Git** should be used for version control.
5. **System Architecture**
   1. The backend must follow an **MVC-like structure** to separate logic properly:

**-Model**: fetch\_data

**-View**: main, draw

**-Controller**: Logic that connects everything

* 1. The system should be **modular**, making it easy to expand later.

1. **Robot Communication**
   1. The system must generate **microcontroller code** based on user commands.
   2. The code should be **uploaded with wired connection.**