Gardian - Automated Garden Watering System

Bi Bianca Ciobanu

TABLE OF CONTENTS

- User Story #8 – Notifications and Alerts

Interface description

  - 1. Login Page
  - 2. Registration Page
  - 3. Home Page
  - 4. Plant Recognition Page
  - 5. Plant Profiles Management Page
  - 6. Settings Page
  - 7.Notifications and Alerts Panel

Protocol, Application Data Structure and Message Flow

- Protocol
- Application Data Structure and Message Flow:

Scalability Considerations

  - Machine Specifications
  - Maximum Load

Introduction

App description

The Automated Garden Watering application is a smart tool designed to help users effectively care for their plants by integrating Arduino technology and user-friendly interfaces. It offers features such as user authentication for secure plant data storage, real-time soil moisture monitoring, comprehensive environmental tracking, automated and manual water pump control, and personalized plant care based on plant recognition. Users can create detailed profiles for each plant, set watering schedules, and receive notifications for critical conditions, ensuring optimal plant health and maintenance with minimal effort.

Used Technologies

Creating an automated garden watering system involves using a mix of software and hardware technologies to build a reliable and efficient solution. This project uses Spring

Boot for the backend and ReactJS for the frontend development and the ESP32 microcontroller for IoT functionalities. By integrating these technologies, we can achieve real-time monitoring and control to ensure plants get the right amount of water. Here's an overview of the technologies used in this project.

Software

Frontend:

- **ReactJS**: Main framework for building the user interface.
- **Axios**: For making HTTP requests to the backend.
- **Material-UI**: Component libraries for UI design.
- **Socket.IO**: For real-time communication with the backend.

Backend:

- **Spring Boot**: Main framework for developing the backend.
- **Spring Web**: For building RESTful web services.
- **Spring Data JPA**: For data persistence and database interaction.
- **Spring Security**: For securing the application.
- **WebSockets**: For real-time communication between the server and client.
- **Maven/Gradle**: Build automation tools.

Database:

- **MySQL**

Integration and Communication:

- **REST APIs**: For standard HTTP communication between the frontend and backend.
- **WebSockets**: For real-time, bidirectional communication between the frontend, backend, and ESP32.
- **Postman**: For API testing.
- **Arduino**

Hardware

ESP32

App Functionalities

User Authentication

Allows user to keep all information regarding its plants stored in one place.

Soil Moisture Monitoring

Involves using soil moisture sensors to measure the moisture levels in the soil. The ESP32 sends this data to the backend server periodically, allowing for real-time monitoring.

Environmental Monitoring

Extends the first functionality by measuring temperature, humidity, and light levels, sending this data to the backend for comprehensive environmental tracking.

Water Pump Control

Based on the moisture levels or direct commands from the server, the ESP32 can activate or deactivate the water pump. This control also includes a manual override function, enabling users to turn the pump on or off via commands from the server.

Plant recognition based on plant name

The application includes a plant recognition algorithm where users can input the name of a plant and retrieve personalized watering information from a database. This database stores information about various plants, including optimal watering requirements, frequency, and other care instructions.

Profiles for different plants

Users can create and store profiles for different plants in their garden, including details such as plant name, watering frequency, preferred sunlight, and soil type. Each plant profile integrates sensor data to monitor and display real-time information about the plant's status, such as soil moisture levels and temperature.

Scheduling functionalities

Allow automated watering at specific times, giving users a scheduling interface to set their watering preferences.

Notifications and Alerts

Inform users of critical conditions, such as extremely low or high soil moisture levels, or other sensory dysfunctions and connectivity issues within the ESP32 device.

User Stories

User Story #1 – User Authentication

As a user,

I want to be able to authenticate into the application,

So that I can securely access and manage all information regarding my plants in one place.

Acceptance Criteria:

1. **AC#1 – Registration Page**
    - **Given that** I am a new user,
    - **When** I navigate to the registration page,
    - **Then** I should be able to create a new account by entering a username, email, password, and password confirmation.
2. **AC#2 – Login Page**
    - **Given** that I am a registered user,
    - **When** I navigate to the login page,
    - **Then** I should be able to log in by entering my username and password.

User Story #2 – Soil Moisture Monitoring

As a user,

I want to monitor soil moisture levels in real-time,

So that I can ensure my plants are being watered adequately.

Acceptance Criteria:

1. **AC#1 – Real-Time Moisture Data**
    - **Given that** I have soil moisture sensors set up,
    - **When** the sensors measure soil moisture,
    - **Then** the data should be sent to the backend server and displayed on my plant profile in real-time.

User Story #3 – Environmental Monitoring

As a user,

I want to monitor temperature, humidity, and light levels,

So that I can track the environmental conditions affecting my plants.

Acceptance Criteria:

1. **AC#1 – Environmental Data Collection**
    - **Given that** I have environmental sensors set up,
    - **When** the sensors measure temperature, humidity, and light levels,
    - **Then** the data should be sent to the backend server and displayed on my dashboard.
2. **AC#2 – Comprehensive Dashboard View**
    - **Given that** I have environmental data,
    - **When** I view my dashboard,
    - **Then** I should see a comprehensive display of soil moisture, temperature, humidity, and light levels.

User Story #4 – Water Pump Control

As a user,

I want to control the water pump based on soil moisture levels or direct commands,

So that I can automate or manually manage the watering of my plants.

Acceptance Criteria:

1. **AC#1 – Automated Pump Control**
    - **Given that** the soil moisture levels are below a certain threshold,
    - **When** the system detects low moisture,
    - **Then** the ESP32 should activate the water pump automatically.
2. **AC#2 – Manual Pump Control**
    - **Given that** I am on the dashboard,
    - **When** I click the manual override button,
    - **Then** I should be able to turn the water pump on or off directly from the server.

User Story #5 – Plant Recognition

As a user,

I want to recognize plants by their name and get personalized watering information,

So that I can ensure each plant receives appropriate care.

Acceptance Criteria:

1. **AC#1 – Plant Recognition Input**
    - **Given that** I am on the plant recognition page,
    - **When** I input a plant name,

- **Then** the system should retrieve and display personalized watering information from the database.

User Story #6 – Plant Profiles

As a user,

I want to create and store profiles for different plants,

So that I can monitor their status and manage their watering needs more effectively.

Acceptance Criteria:

1. **AC#1 – Create Plant Profile**
   - **Given** that I am on the plant profiles page,
   - **When** I add a new plant profile,
   - **Then** I should be able to enter details such as plant name, watering frequency, preferred sunlight, and soil type.
2. **AC#2 – View Plant Profiles**
   - **Given** that I have created plant profiles,
   - **When** I view the plant profiles page,
   - **Then** I should see a list of all my plants with their profiles and current status based on sensor data.

User Story #7 – Scheduling Watering

As a user,

I want to set automated watering schedules,

So that my plants are watered at specific times without manual intervention.

Acceptance Criteria:

1. **AC#1 – Set Watering Schedule**
   - **Given that** I am on the scheduling page,
   - **When** I set a new watering schedule,
   - **Then** I should be able to specify the days and times for the watering to occur.
2. **AC#2 – Automated Watering**
   - **Given that** I have set a watering schedule,
   - **When** the specified time is reached,
   - **Then** the system should automatically activate the water pump according to the schedule.

User Story #8 – Notifications and Alerts

As a user,

I want to receive notifications and alerts for critical conditions,

So that I can take timely actions to address any issues with my plants.

Acceptance Criteria:

1. **AC#1 – Soil Moisture Alerts**
    - **Given that** the soil moisture levels are critically low or high,
    - **When** such a condition is detected,
    - **Then** the system should send me a notification or alert.
2. **AC#2 – Sensor Dysfunction Alerts**
    - **Given that** there is a dysfunction or connectivity issue with the ESP32 device,
    - **When** such an issue is detected,
    - **Then** the system should send me an alert to inform me of the problem.

Interface description

1. Login Page

- **Role:** Allows users to authenticate into the application to access and manage plant information.
- **Elements:**
    - Username input field.
    - Password input field.
    - Login button.
    - Link to the registration page for new users.
    - Forgot password link.

2. Registration Page

- **Role:** Enables new users to create an account in the application.
- **Elements:**
    - Username input field.
    - Email address input field.
    - Password input field.
    - Confirm password input field.
    - Register button.

    - Link to the login page for existing users.

3. Home Page

- **Role:** Centralizes all plant profiles and other functionalities of the application.
- **Elements:**
    - Panels for viewing and managing each plant profiles.
    - Panel for setting automated watering schedules.
    - Notifications and alerts for critical conditions or sensor malfunctions.

4. Plant Recognition Page

- **Role:** Allows users to input a plant's name to receive personalized watering information from an integrated database.
- **Elements:**
    - Input field for entering the plant's name.
    - Search button to fetch plant information.
    - Search results and displayed information about the selected plant.

5. Plant Profiles Management Page

- **Role:** Provides functionality for users to create, view, and manage profiles for different plants in their garden.
- **Elements:**
    - List of plant profiles with details such as name, watering frequency, sunlight preferences, and soil type.
    - Panel for monitoring soil moisture levels.
    - Panel for monitoring environmental conditions (temperature, humidity, light).
    - Manual control button for the water pump.
    - Option to add new plant profiles.
    - Edit and delete functionalities for existing plant profiles.

6. Settings Page

- **Role:** Allows users to customize application settings and preferences.
- **Elements:**
    - General settings for language preferences or units (metric/imperial).
    - Account management options (change password, update profile).

7.Notifications and Alerts Panel

- **Role:** Displays real-time notifications and alerts about critical conditions or system updates.
- **Elements:**
  - Options to configure notification preferences.
  - List of recent notifications.
  - Details of each notification with timestamp and type (e.g., sensor alert, watering schedule update).

Protocol, Application Data Structure and Message Flow

Protocol

The application uses HTTP as the standard protocol for communication between the frontend and backend components. The backend communicates with the ESP32 devices using WebSockets for real-time sensor data exchange.

Port Matrix:

- **Frontend to Backend:** HTTP (Port 80/443 for HTTPS)
- **Backend to ESP32 Devices:** WebSockets (Port 80/443 for WebSockets over HTTP/HTTPS)

Application Data Structure and Message Flow:

1. User Authentication

   Log in: User enters their email and password to log in.

   i. **Request**: `POST /login`

   ii. **Body**:

   ```
   { "username": "user@example.com",
   "password": "password123" }
   ```

   i. **Response**: Returns authentication token and user information

2. Plant Profile Management

   Viewing plant profiles: User navigates to their plant profiles page.

   i. **Request**: `GET /plants`

   ii. **Response**: Returns a list of plant profiles associated with the user.

   Adding a new plant profile: User adds a new plant to their garden.

   i. **Request**: `POST /plants/add`

   ii. **Body**:

```
{ "plantName": "Tomato",
 "wateringFrequency": "Daily",
 "preferredSunlight": "Full Sun",
 "soilType": "Loamy" }
```

iii. **Response**: Returns the newly created plant profile.

3. Soil Moisture and Environmental Monitoring

Retrieving sensor data: Backend requests the latest sensor data from an ESP32 device.

i. **Request**: `GET /sensors/data`

ii. **Response**: Returns current sensor data (soil moisture, temperature, humidity, light levels) from the ESP32.

ESP32 sending sensor data: ESP32 sends updated sensor data to the backend.

i. **WebSocket Message**: `ws://backend/sensor/update`

ii. **Message Body**:

```
{ "sensorId": "12345",
"soilMoisture": 45,
"temperature": 22,
"humidity": 60,
"lightLevel": 700 }
```

4. Water Pump Control

Automated watering based on moisture level: ESP32 activates water pump based on soil moisture data.

i. **Request**: `POST /pump/control`

ii. **Body**:

```
{ "sensorId": "12345",
 "action": "activate" }
```

iii. **Response**: Confirms activation or deactivation of the water pump.

Manual override: User manually activates or deactivates the water pump from the frontend.

i. **Request**: `POST /pump/manual`

ii. **Body**: `{ "action": "deactivate" }`

iii. **Response**: Confirms the manual action taken on the water pump.

5. Plant Recognition and Personalized Watering Information

Plant recognition by name: User inputs the name of a plant to get personalized watering information.

i. **Request**: `POST /plants/recognize`

ii. **Body**: `{ "plantName": "Tomato" }`

iii. **Response**: Returns personalized watering instructions and care tips for the plant from the database.

6. Scheduling Functionalities

   Setting up a watering schedule: User sets a specific time for automated watering.

   i. **Request**: `POST /schedule/set`

   ii. **Body**: `{ "plantId": "12345", "time": "07:00", "frequency": "Daily" }`

   iii. **Response**: Confirms the scheduling setup for the plant.

7. Notifications and Alerts

   Receiving alerts: Backend sends alerts to the user for critical conditions.

   i. **WebSocket Message**: `ws://frontend/alerts`

   ii. **Message Body**: `{ "alertType": "LowSoilMoisture", "message": "Soil moisture is below the threshold for Tomato plant." }`

Scalability Considerations

Machine Specifications

CPU: 2-4 processors
RAM: 8-16 GB RAM
Storage: 100GB SSD
Network: High-speed internet connection, 500 Mbps

Maximum Load

Requests Per Second (RPS): Between a minimum of 100 and a maximum of 300

- This range ensures that the server can handle frequent requests from multiple users, including sensor updates and user interactions.

Simultaneous Users: 2000 simultaneous users
Number of Simultaneous Sensor Data Updates: Minimum 500, maximum 1500
Number of Active WebSocket Connections: 1000

- WebSockets are used for real-time communication between the ESP32 devices and the backend server, allowing for instant updates and commands. The system can support up to 1000 active connections at any given time.