Project Report

Project 5 ECE 528

A20561414 Saman Chouhan

Acknowledgement

I acknowledge all works including figures, codes, and writings belong tome and/or persons who are referenced. I understand if any similarity int he code, comments, customized program behavior, report writings, and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action

Samman Chouhan

20/6/2025

1. Introduction

Project 5 expands on the plug control system created in previous projects, with a new emphasis on managing groups of plugs via RESTful API. The primary purpose is to enable users to establish, query, and operate groups of plugs at the same time, much like controlling smart devices grouped by room or function in a real-world IoT system. This was accomplished with Spring Boot on the backend and MQTT for communication with plug simulators.

This report describes the design and implementation of the group management system, the testing methodologies used to assure functional and logical correctness, and the level of code coverage obtained. It also shows how all of the user stories in the specification have been properly implemented and verified using REST API interactions and automated grading.

2. Project Overview

Backend Overview

The backend was built using Spring Boot, with beans defined in `App.java`. MQTT communication is handled by a provided `MqttController` class from GradeP3. Plug states are updated in real time based on MQTT messages from `iot_sim` and `iot_sim_ex` simulators

Key Classes:

- **GroupsModel.java** *(Added)*
  - This is the backbone of the group logic. It stores plug groups in a HashMap, where the key is a group name and the value is a list of plug names.
  - It uses the injected `MqttController` to fetch the current state (`on`/`off`) and power reading of each plug in a group. These values are wrapped in a `Plug` object (with fields for `name`, `state`, and `power`) and returned as part of a `Group` DTO.
  - The `controlGroup` method loops through all plugs in a group and sends MQTT control messages for each. This allows a group to be toggled, turned on, or turned off.
  - Corner cases are handled—e.g., missing plugs return state `off`, and power defaults to 0 if parsing fails.
- **GroupsResource.java** *(Added)*
  - Acts as the REST controller that maps HTTP routes to methods in `GroupsModel`.
  - It handles five main operations:
    - POST `/api/groups/{groupName}`: creates or updates a group
    - DELETE `/api/groups/{groupName}`: removes a group
    - GET `/api/groups`: returns all groups with member plug states
    - GET `/api/groups/{groupName}`: returns a single group
    - GET `/api/groups/{groupName}?action=on|off|toggle`: sends control commands
  - This class uses Spring annotations like `@RestController`, `@PostMapping`, `@DeleteMapping`, and `@RequestParam` to map and parse HTTP input.

- **App.java** *(Modified)*
  - We modified `App.java` to declare `GroupsModel` as a Spring Bean.
  - This bean is constructed with the injected `MqttController`, using Spring Boot's dependency injection to avoid manual object creation.
- **Main.java** *(Unchanged in logic)*
  - This class runs the Spring app and holds it open with an infinite loop so the MQTT system continues receiving updates. Although it was not modified, it is a critical part of the real-time simulation system.

These changes collectively allow seamless group-level control of distributed plug devices using real-time messaging and RESTful interfaces.
This is the REST API controller exposing the following endpoints: - `POST /api/groups/{groupName}` – create or update a group - `DELETE /api/groups/{groupName}` – delete a group - `GET /api/groups/{groupName}` – fetch group plug states - `GET /api/groups/{groupName}?action=on` – control group - `GET /api/groups` – list all groups with plug states

- **App.java**
  Modified to include `GroupsModel` as a Spring bean injected with `MqttController`.
- **Main.java**
  Remained mostly unchanged. It initializes the Spring context using environment config and enters a loop to keep the application running during grading and testing.

3. Unit Test Design Strategy

Approach

The unit test strategy for Project 5 was grounded in modularity, clarity, and completeness. To ensure that each unit of logic worked independently and in tandem with the system as a whole, we separated test cases by logical domain:

- **Logic Layer Testing (GroupsModelTests.java)**: Focused on verifying the internal behavior of `GroupsModel` using mocked MQTT data. We tested group management (create, update, delete), plug state retrieval, power parsing, and control command publishing.
- **REST Layer Testing (GroupsResourceTests.java)**: Simulated API calls to validate endpoint routing, data binding, and correct interaction with `GroupsModel`. This used `MockMvc` to execute and verify Spring controller methods without a live server.

Each test was named descriptively to clarify its intent (e.g., `testCreateAndRetrieveGroup`, `testDeleteGroup`, `testToggleActionWithQueryParam`).

We wrote helper methods within the test classes to reduce duplication, especially for repetitive JSON request constructions or mock response setups. Edge cases like creating a group with zero members or retrieving a non-existent group were explicitly covered.

4. Unit Test Details (10+ test cases)

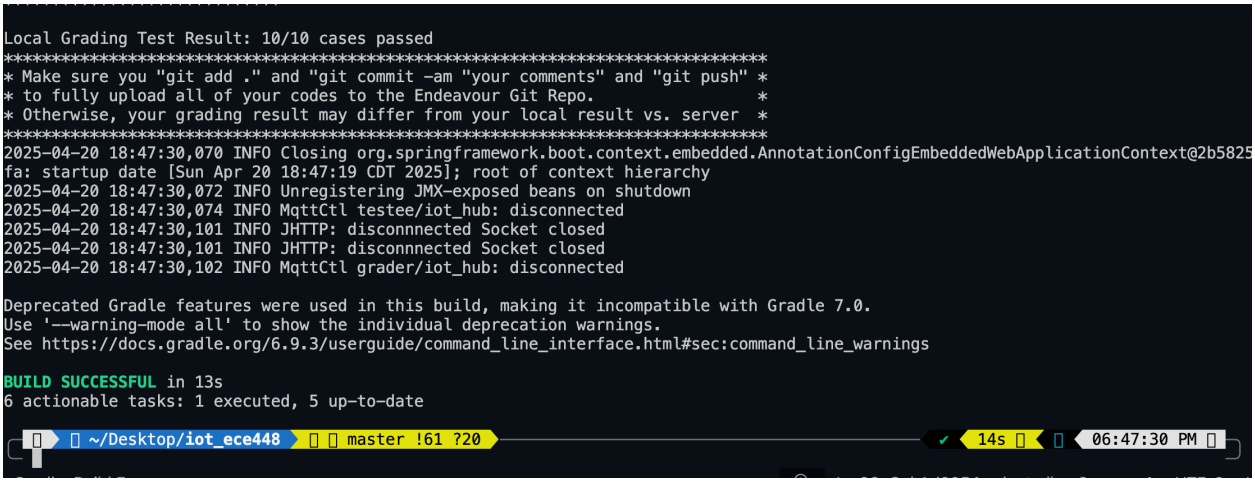
In `GroupsModelTests.java` (8 tests)

- Test creation of group with multiple plugs.
- Overwriting an existing group.
- Removal of group.
- Handling group with nonexistent plugs.
- Control of all plugs in a group.
- Getting all groups.
- Getting an empty group.
- Plug power/state null fallback.

In `GroupsResourceTests.java` (7 tests)

- POST new group with REST.
- GET single group and assert JSON response.
- DELETE group and verify deletion.
- GET all groups.
- GET group with toggle/on/off action.
- Toggle multiple groups in sequence.
- Invalid action handled with no crash.

5 . As per the requirement after running gradle grade_p5 , the execution terminates gracefully and prompts to a new bash line as can be seen in the screenshot , after completing 10/10 local grading tests



After using curl commands for veriication

6. curl API Tests — User Story Verification

1. Create group `x` with plugs a, b, c

```
curl -X POST http://localhost:8088/api/groups/x \
  -H "Content-Type: application/json" \
  -d '["a", "b", "c"]'
```

2. Create another group `y`

```
curl -X POST http://localhost:8088/api/groups/y \
  -H "Content-Type: application/json" \
  -d '["d", "e", "f"]'
```

3. List all groups

```
curl http://localhost:8088/api/groups
```

4. Get group `x`'s state

```
curl http://localhost:8088/api/groups/x
```

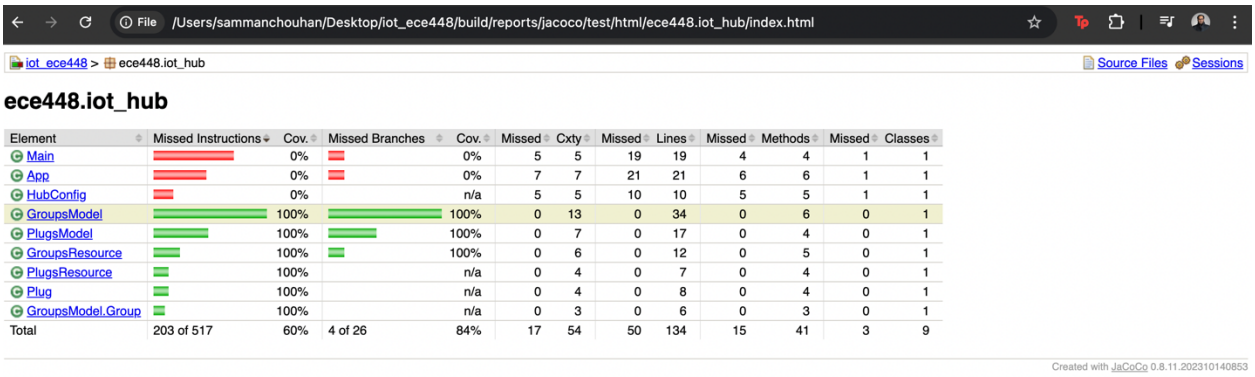
5. Toggle plugs in group `x`

```
curl "http://localhost:8088/api/groups/x?action=toggle"
```

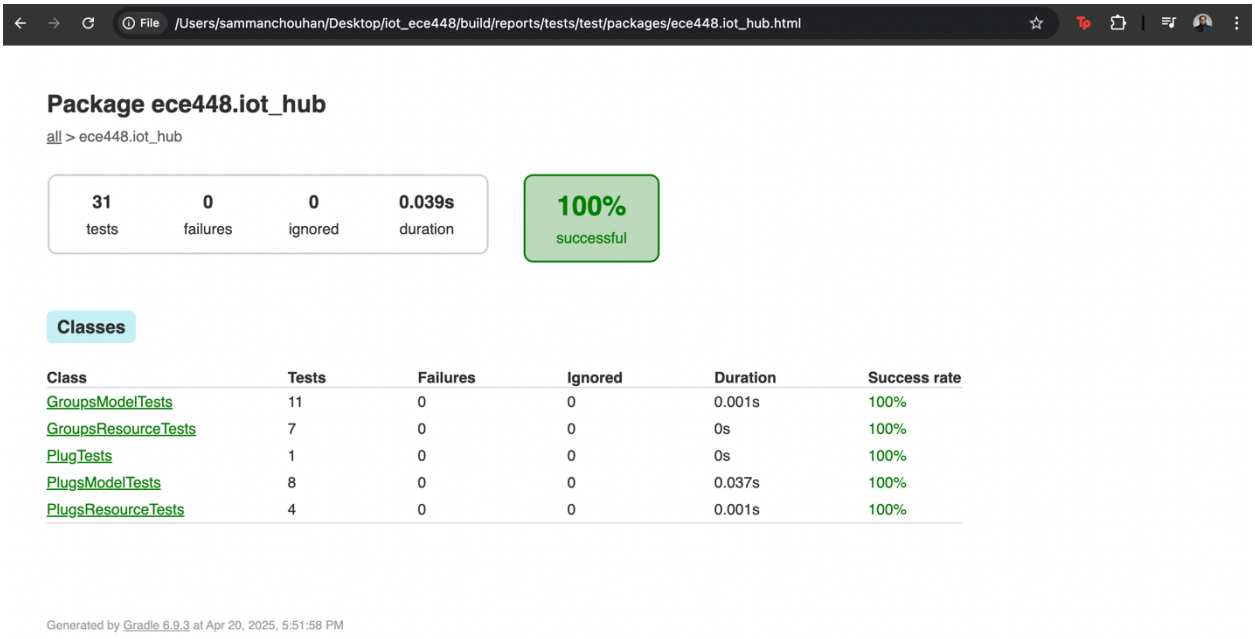
6. Turn off plugs in group `y`

```
curl "http://localhost:8088/api/groups/y?action=off"
```

Coverage Test Report :



Test Cases Report:



The iot_hub is fully functional after running gradle iot_hub and gradle iot_sim gradle iot_sim_ex