ECE 448/528
Application Software Design

Lecture 17. RESTful Web Services – Part II
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

RESTful Resource (Cont.)

Query and Parameters

```
...
@GetMapping("/api/groups/{group}")
public Object getGroup(
  @PathVariable("group") String group,
  @RequestParam(value ="action", required =false) String action) {
  if (action ==null) {
    Object ret =makeGroup(group);
    logger.info("Group {}: {}", group, ret);
    return ret;
  }
  ...
```

- Update the method `getGroup` to handle the query.
- Spring Boot will parse the query, which can be retrieved via `@RequestParam` as a parameter to the method.
- You may specify whether a query key is required or not.
  - If it is required but not presented, Spring Boot will simply notify the client of the error without calling the method.
  - If it is not required and not presented, the method will receive a `null` reference.

POST and Request Body

```
@RestController
public class GroupsResource {
  ...
  @PostMapping("/api/groups/{group}")
  public void createGroup(
    @PathVariable("group") String group,
    @RequestBody List<String> members) {
    groups.setGroupMembers(group, members);
    logger.info("Group {}: created {}", group, members);
  }
```

- `@PostMapping`: handle POST requests on the path.
- While it is convenient to pass simple parameters via query, more complex parameters can be passed in the request body as JSON.
  - Used with the POST method.
  - Retrieved via `@RequestBody`.

DELETE

```
@RestController
public class GroupsResource {
  ...
  @DeleteMapping("/api/groups/{group}")
  public void removeGroup(
    @PathVariable("group") String group) {
    groups.removeGroup(group);
    logger.info("Group {}: removed", group);
  }
```

- `@DeleteMapping`: handle DELETE requests on the path.

Advanced Data Model Design

DELETE

- A data model may interact with external services.
  - In order to update the internal state of the data model and to cause other systems to change.
- For example, our IoT hub server backend needs a data model for the plugs.
  - Unlike the IoT simulator, we don't know the plugs or control them directly.
  - Plug states, including their names, come from the MQTT \broker.
  - Plugs are switched on/off and toggled by publishing messages to the MQTT broker.
- Other important data model features.
  - Persistence: preserve data if the services need to restart or crash.
  - Authorization: decide who can do what on the data model and enforce such rules.

MQTT Component

- Need a component to handle MQTT messages.
  - To support a data model for the plugs.
- Two choices
  - Make use of `GradeP3.MqttController`, where a single component works as the data model for the plugs while encapsulating MQTT details.
  - Or, create your data model class that depends on a `MqttClient` component.
- It is a good idea to study `GradeP3.MqttController` first for both choices.

MqttClient as a Component

```
@SpringBootApplication
public class App {
  @Bean(destroyMethod = "disconnect")
  public MqttClient mqttClient(Environment env) throws Exception {
    String broker = env.getProperty("mqtt.broker");
    String clientId = env.getProperty("mqtt.clientId");
    MqttClient mqtt = new MqttClient(broker, clientId, new MemoryPersistence())
    mqtt.connect();
    logger.info("MqttClient {} connected: {}", clientId, broker);
    return mqtt;
  }
  private static final Logger logger = LoggerFactory.getLogger(App.class);
}
```

- `@SpringBootApplication` on the `App` class allows to define factory methods within to create components.
- You'll need to specify `destroyMethod` as `"disconnect"` for Spring to destroy the component properly.
- By default, Spring will try to call the method `close` to destroy a component if it is presented.
- But this is not for MqttClient – check Lecture 13 Slide 4.

Multiple Data Models within a RESTful Resource

- In Project 5, the responses of various RESTful requests related to groups are required to include plugs states.
- This implies that the `GroupsResource` you are going to implement for Project 5 also needs to access the data model for plugs.
  - In addition to `GroupsModel`.
- Use dependency injection to locate the data model of plugs automatically.
- Make sure that concurrency is handled properly via intrinsic locks on both models.

Additional Hints for Project 4 and 5

Working with IoT Hub

- Your code for Projects 4 and 5 should be in the `iot_hub` package.
- The IoT hub is decoupled from the IoT simulator.
  - They only communicate via the MQTT broker.
  - It is simply wrong to access the classes/objects in the `iot_sim` package.

Minor Spring Boot Issue

```
@RestController
public class PlugsResource {
  ...
  // Allow . in the path variable 'plug'
  @GetMapping("/api/plugs/{plug:.+}")
  public Object getPlug(
    @PathVariable("plug") String plug,
  ...
}
```

- Spring Boot has an undesired feature that interferes with path variables containing a dot.
  - e.g., for a plug name like b.100
- To resolve such issue, e.g., for the path variable `plug`, use `{plug:.+}` to allow explicitly.
- This won't affect our grading but may cause issues later in Project 6 when you need to work with the Web frontend.

Unit Tests

- Unit tests are supposed to be simple.
  - Don’t have to be as comprehensive as our grading tests.
- Just need to ensure coverage.
- It is perfectly OK to start with a few unit tests and add more later when you are passing more grading tests.