ECE 448/528
Application Software Design

Lecture 16. RESTful Web Services – Part I
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

RESTful Web Service Interface Design

RESTful Interface Design

- The first step to building any RESTful web services.
  - Similar to designing a method header in Java.
  - Name? Parameters? Returned value?
  - Don’t worry about how to implement it.
- Three things to decide.
  - The path: should be meaningful.
  - Parameters: either via query or request body.
  - Returned data: as compositions of maps and arrays.
- Services and applications using a RESTful web service only need to know its RESTful interface.

RESTful Interface Example

- To manage groups of strings
  - Each group should have its name.
- Path for a group: `/api/groups/{group}`
  - A prefix like `/api` ensures that RESTful API and static web pages won't interfere with each other.
  - `{group}` is to be replaced with the name of the group this path refers to.
  - `groups` provides a hint that this path should be handled together among a set of similar paths.
- GET for read: return a JSON object
  - e.g., `{"name":"A", "members":["1","2","3"]}`
- POST for create: members as a JSON array in request body.
- DELETE for delete
- No PUT for simplicity
  - Update a group by creating a new one with the same name.

RESTful Interface Example (Cont.)

- Path with query: `/api/groups/A?action=on`
  - An alternative way to provide simple parameters instead of using request body.
  - Use together with the GET method.
  - Use this with the GET method to control plugs whose names are members of the group.
  - Conceptually, switching on/off plugs may update the states of group members, so we should use POST or PUT methods. But we choose to use GET, as it is easier for browsers and command line tools to send GET requests.
- Collection: `/api/groups`
  - GET: return a JSON array of groups with each group encoded individually as explained in the previous slide.
  - No POST/PUT/DELETE: better to work with individual groups.
- Design multiple sets of such RESTful interfaces to support complex functionalities.

Testing RESTful Interfaces

- By writing code, e.g., in `GradeP4.java` and `GradeP5.java`.
- Use command line tools like `wget` and `curl`.
- Interactively via browsers.
  - Run the code in `src/main/ece448/lec16` under branch `lec16-rest` and test it before learning the details.

RESTful Web Service Implementation

- A web server: handle HTTP requests and responses.
  - Though it is possible to use JHTTP again, it is better to use an established web server to support more complex web applications.
  - We will use the **Apache Tomcat Web Server** bundled with **Spring Boot**.
- Data models: allows to manipulate data.
  - As Java classes, each is responsible for a piece of data.
  - May need to interact with other services and databases.
- RESTful resources: map between HTTP requests/responses and methods of data models.
  - As Java classes, each is responsible for a set of closely related RESTful interfaces.
  - A RESTful resource may utilize multiple data models.

Data Model

Class Design for Data Model

```
@Component
public class GroupsModel {
  private HashMap<String, HashSet<String>> groups =new HashMap<>();
```

- Use group management as an example.
  - Self-contained and does not require interactions with other services or databases.
- groups key=group name; value=list of members(plugs)
- Choose a proper data structure.
  - Composition of maps, arrays, and user-defined classes.
- Use `@Component` since data models are almost always singletons.
- Provide getters and setters for CRUD operations.

Concurrency

```
@Component
public class GroupsModel {
  private HashMap<String, HashSet<String>> groups = new HashMap<>();

  synchronized public List<String> getGroups() {
    return new ArrayList<>(groups.keySet());
  }

  synchronized public List<String> getGroupMembers(String group) {
    HashSet<String> members = groups.get(group);
    return (members == null)? new ArrayList<>(): new ArrayList<>(members);
  }
```

- Multiple RESTful requests may be handled by multiple threads.
  - They may access the data model at the same time.
- Use the intrinsic lock to protect CRUD operations.
- Getters need to make copies.
  - If your getters return internal data structures, then accesses to them cannot be protected by the lock.

Setters

```
@Component
public class GroupsModel {
  private HashMap<String, HashSet<String>> groups = new HashMap<>();
  ...
  synchronized public void setGroupMembers(String group, List<String> members)
    groups.put(group, new HashSet<>(members));
  }

  synchronized public void removeGroup(String group) {
    groups.remove(group);
  }
}
```

- Protected by the intrinsic lock as well

RESTful Resource

Class Design for RESTful Resource

```
@RestController
public class GroupsResource {
  private final GroupsModel groups;
  public GroupsResource(GroupsModel groups) {
    this.groups = groups;
  }
```

- Each class will include methods to handle a set of closely related RESTful interfaces.
  - Almost always a singleton.
- `@RestController`: a special `@Component`.
  - Maps requests to methods via paths
  - Decodes requests into parameters to methods.
  - Encodes returned values from methods into responses.
- Where do the parameter `groups` in the constructor come from?
  - Spring Boot uses dependency injection to locate `groups` automatically.
  - You may use multiple data models if necessary.

GET and Path

```
@RestController
public class GroupsResource {
  ...
  @GetMapping("/api/groups/{group}")
  public Object getGroup(@PathVariable("group") String group) {
    Object ret = makeGroup(group);
    logger.info("Group {}: {}", group, ret);
    return ret;
  }
```

- `@GetMapping`: handle GET requests on the path.
- The name of the group, specified by `{group}` in the path, is retrieved via `@PathVariable` as a parameter to the method.
  - Additional sub-strings from the path can be specified via additional `{}`'s and be retrieved via additional `@PathVariable`.

Returned Value and JSON

```
@RestController
public class GroupsResource {
  ...
  @GetMapping("/api/groups/{group}")
  public Object getGroup(@PathVariable("group") String group) {
    Object ret = makeGroup(group);
    logger.info("Group {}: {}", group, ret);
    return ret;
  }
  protected Object makeGroup(String group) {
    HashMap<String, Object> ret = new HashMap<>();
    ret.put("name", group);
    ret.put("members", groups.getGroupMembers(group));
    return ret;
  }
```

- Spring Boot uses the same Jackson library as we discussed in Lecture 12 to map between Java objects and JSON representations.
- The returned `HashMap` will be encoded into a JSON string in the response body automatically.

Summary

- RESTful web services.
- Steps to design and implement RESTful web services.
  - Design the RESTful interface.
  - Design/implement classes for data models.
  - Design/implement classes for RESTful resources.
- RESTful web services can be tested by command line tools and browsers.