Haaga-Helia

Back End Programming

REST

Haaga-Helia

Spring Boot: REST

- REST (Representational State Transfer)
- REST is an architectural style for designing distributed systems. It is not a standard but a set of constraints
- Main principles
  - Resources expose easily understood directory structure URIs
  - Representations transfer JSON or XML to represent data objects and attributes
  - Messages use HTTP methods (POST, GET, PUT, DELETE)
    - PUT to modify and POST to add new
  - Stateless interactions store no client context on the server between requests

Haaga-Helia

REST Endpoints

Http Method	Resource Endpoint	Actions
GET	/students	Get All Students
POST	/students	Add New Student
GET	/students/{id}	Get One Student
PUT	/students/{id}	Update One Student
DELETE	/students/{id}	Delete One Student
Etc.		

Haaga-Helia

Spring Boot: REST

- In Spring framework's approach to building RESTful web services, HTTP requests are handled by a **controller**
- Controllers are identified by `@RestController` annotation
- Difference between REST and MVC controller are that REST controller returns HTTP response body and MVC controller returns a view.
- REST controller will write object data to the HTTP response as JSON format
- Spring uses Jackson JSON processor to convert object data to JSON

Haaga-Helia

Spring Boot: REST

- Example
  - In the following example you have Message class with two attributes (id, text)

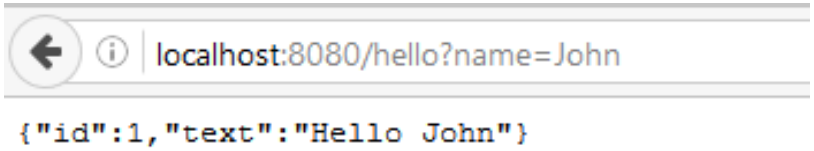
```
@RestController
public class MessageController {
   private final AtomicLong counter = new AtomicLong();

   @RequestMapping("/hello")
   public Message greeting(@RequestParam(value="name",
            defaultValue="World") String name) {
     return new Message(counter.incrementAndGet(),"Hello " + name);
   }
}
```

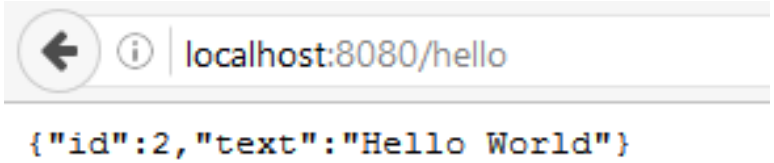
Haaga-Helia

Spring Boot: REST

- Example
  - First GET request with request parameter



  - Second GET request without request parameter



Haaga-Helia

Spring Boot: REST

- You can also create RESTful service by using `@Controller` and `@ResponseBody` annotations

```
@Controller
public class MessageController {
   private final AtomicLong counter = new AtomicLong();

   @RequestMapping("/hello")
   public @ResponseBody Message greeting(@RequestParam(value="name",
            defaultValue="World") String name) {
     return new Message(counter.incrementAndGet(),"Hello " + name));
   }
}
```

Haaga-Helia

Spring Boot: REST

- Example: Add REST service to student list application (Example project from ORM One-to-many chapter)
  - The first method returns students to Thymeleaf template. The second method returns as JSON format.

```
// Show all students in Thymeleaf template
@RequestMapping(value="/studentlist", method = RequestMethod.GET)
public String studentList(Model model) {
   model.addAttribute("students", repository.findAll());
   return "studentlist"; // studentlist.html
}

// RESTful service to get all students
@RequestMapping(value="/students", method = RequestMethod.GET)
public @ResponseBody List<Student> studentListRest() {
   return (List<Student>) repository.findAll();
}
```

Haaga-Helia

Spring Boot: REST

- You have to configure one-to-many relationship from JSON by using `@JsonIgnore or @JsonIgnoreProperties` annotation. Otherwise entity relationship will cause endless loop (First student is serialized and it contains department which is then serialized which contains students which are then serialized….)
- Next slide has the usage for this relationship

```
@ManyToOne
@JoinColumn(name = "departmentid")
private Department department;
```

Haaga-Helia

Spring Boot: REST

- **Department** class has

```
@JsonIgnoreProperties("department")
// ignoring ’department’ attribute for all students
@OneToMany(cascade = CascadeType.ALL, mappedBy =
   "department")
  private List<Student> students;
```

Spring Boot: REST

- Now /students endpoint will return students in JSON

localhost:8080/students


```
[{"id":4,"firstName":"John","lastName":"Johnson","email":"john@john.com","department":{"departmentid":1,"name":"IT"}},
{"id":5,"firstName":"Katy","lastName":"Kateson","email":"kate@kate.com","department":{"departmentid":2,"name":"Business"}}]
```


- And /studentlist endpoint will return HTML page

localhost:8080/studentlist

Students

Name	Email	Department	
John Johnson	john@john.com	IT	Delete
Katy Kateson	kate@kate.com	Business	Delete

Add Student

Haaga-Helia

Spring Boot: REST

- RESTful service to find student by id using path variable

```
// RESTful service to get on student by id
@RequestMapping(value="/student/{id}", method = RequestMethod.GET)
public @ResponseBody Student findStudentRest(@PathVariable("id") Long studentId)
    return repository.findById(studentId);
}
```

localhost:8080/student/5

{"id":5,"firstName":"Katy","lastName":"Kateson","email":"kate@kate.com","department":{"departmentid":2,"name":"Business"}}

Haaga-Helia

Spring Data REST

- Spring Data REST builds on top of Spring Data repositories and automatically exports those as REST resources
- Add dependency to get started with Spring Data REST

```
…
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
...
```

Haaga-Helia

Spring Data REST

- Spring Data REST detects public repositories to determine if a repository will be exported as REST resource
- You don't need to define any Controller
- By default Spring Data REST serves REST resources in application root path ’/’.
- Path can be changed in appliction.properties file

```
spring.data.rest.base-path=/api
```

Haaga-Helia

Spring Data REST

- If you have the following repository

```
public interface StudentRepository extends
      CrudRepository<Student, Long> { }
```

- Spring Data REST will create REST service `/students`

Example: http://localhost:8080/api/students

- The service path name is derived from the entity name (pluralized and uncapitalized)
- Spring Data REST services contains also links to resources (using HAL format)

Haaga-Helia

Spring Data REST

- All available REST resources can be found with HTTP GET request to application root URL
- Example: Student application

```
MINGW64:/c/Users/jusju

jusju@HHMX956 MINGW64 ~
$ curl http://localhost:8080/api
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   271    0   271    0     0   8468      0 --:--:-- --:--:-- --:--:--  8468{
  "_links" : {
    "departments" : {
      "href" : "http://localhost:8080/api/departments"
    },
    "students" : {
      "href" : "http://localhost:8080/api/students"
    },
    "profile" : {
      "href" : "http://localhost:8080/api/profile"
    }
  }
}

jusju@HHMX956 MINGW64 ~
$ |
```

Haaga-Helia

Spring Data REST

- Departments REST service

MINGW64:/c/Users/jusju

```
$ curl http://localhost:8080/api

jusju@HHMX956 MINGW64 ~
$ curl http://localhost:8080/api/departments
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1423    0  1423    0     0  30276      0 --:--:-- --:--:-- --:--:-- 30276{
  "_embedded" : {
    "departments" : [ {
      "name" : "IT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/departments/1"
        },
        "department" : {
          "href" : "http://localhost:8080/api/departments/1"
        },
        "students" : {
          "href" : "http://localhost:8080/api/departments/1/students"
        }
      }
    }, {
      "name" : "Business",
      "_links" : {
        "self" : {
```

curl -H "Content-Type: application/json" -X POST -d {\"username\":\"mkyong\",\"password\":\"abc\"} http://localhost:8080/api/login/

Haaga-Helia

Spring Data REST

- Services can be also used to add and delete items
- Example:
  - Following POST request will add new student to database

```
curl -H "Content-Type: application/json" -X POST -d
'{"firstName":"Jukka","lastName":"Juslin"}'
http://localhost:8080/api/students
```

- REST services can be also secured by using Spring Security (next lessons)

Haaga-Helia

Spring Data REST

- You can also use your queries from the *CrudRepository* by using Rest API
- Example

```
@RepositoryRestResource
public interface StudentRepository extends CrudRepository<Student, Long> {
  List<Student> findByEmail(@Param("email") String email);
}
```

- Add *@RepositoryRestResource* annotation to your repository class and *@Param* annotation for your parameters
- Now you can call your query by using following endpoint

```
http://localhost:8080/api/students/search/findByEmail?email=johnson@mail.com
```

Haaga-Helia

DATA-REST HAL request

To add new student

```
{
  "firstName": "Jukka",
  "lastName": "Kateson",
  "email": "kate@kate.com",
  "department": "api/departments/1"
}
```

Haaga-Helia

Example DATA REST commands with curl

```
$ curl -X PUT http://localhost:8080/api/students/6 -H
'Content-type:application/json' -d '{"firstName":
"Samwise Gamgee", "lastName": "ring bearer"}'

$ curl -X DELETE http://localhost:8080/api/students/7
```