Building a RESTFull Store WebService - Part 2

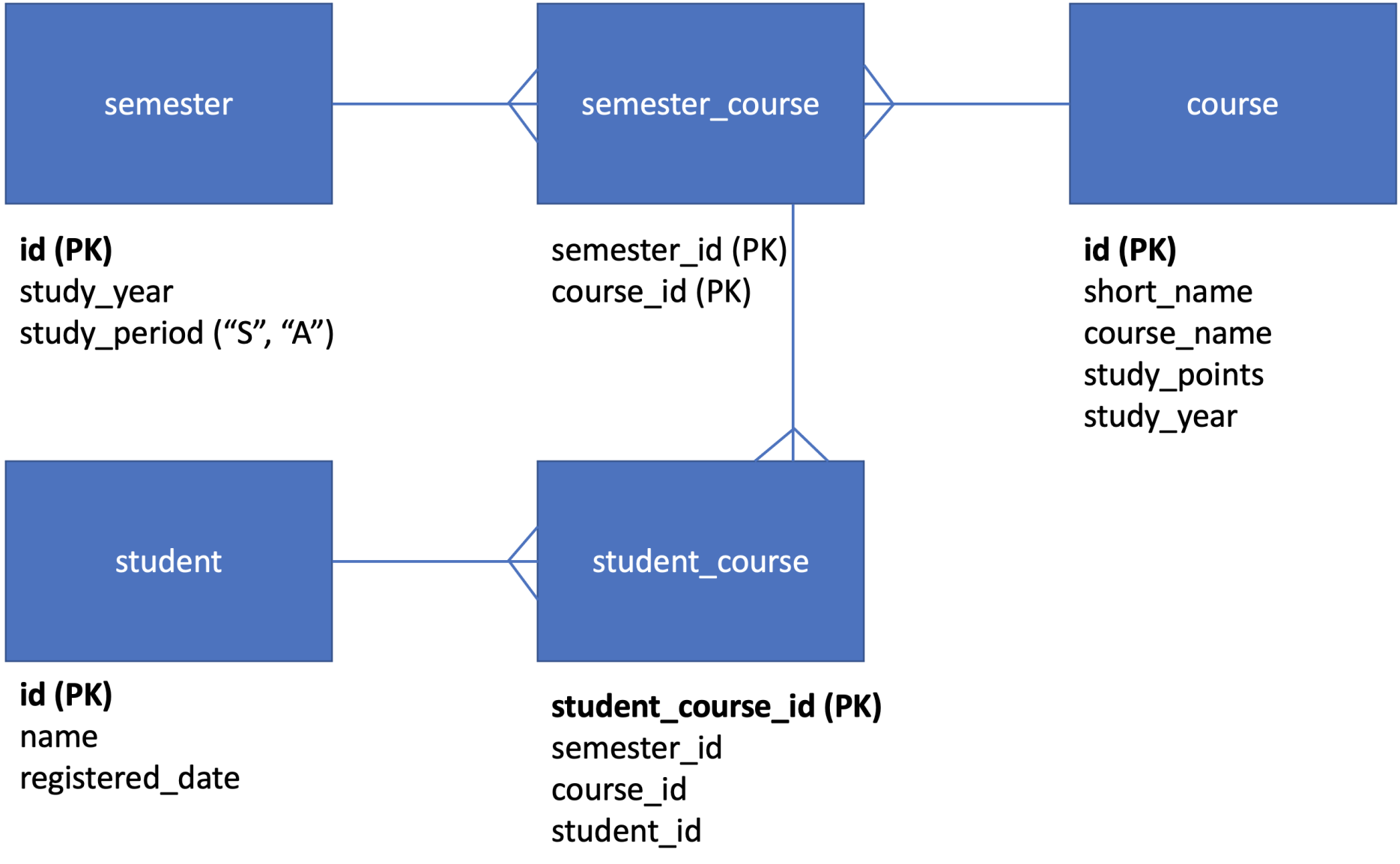
Created by Lasse Jenssen

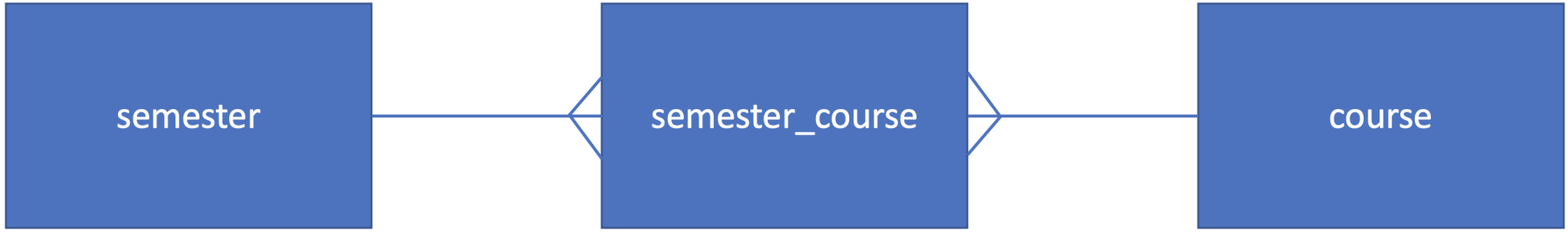
Home

REST Store Web Service

Demo: ws-03-rest-store-part2.tar.gz

- Smal REST applications keeping track of "Courses" and "Semesters".
- Based on **Spring Boot** and **Spring WebMVC**.
- We'll use JSP and JSTL (instead of Thymeleaf)
- *Code: ws-03-rest-store-part2.tar.gz (see course overview)*


semester
semester_course
course
id (PK)
study_year
study_period (“S”, “A”)
semester_id (PK)
course_id (PK)
id (PK)
short_name
course_name
study_points
study_year
student
student_course
id (PK)
name
registered_date
student_course_id (PK)
semester_id
course_id
student_id



id (PK)
study_year
study_period (“S”, “A”)

semester_id (PK)
course_id (PK)

id (PK)
short_name
course_name
study_points
study_year

Dependency: ***tomcat-embed-jasper*** *(scope: provided)*

Spring Boot, WEB and JSP

- JSP has limitations on its own,

  and even more so when combined with Spring Boot.
- That's why we used Thymeleaf earlier.
- Today I'll show how you can use JSP together with Spring Boot.

Maven Dependencies

Demo: ws-03-rest-store-part2.tar.gz

```
<dependencies>
   <dependency>
      <groupid>org.springframework.boot</groupid>
      <artifactid>spring-boot-starter-web</artifactid>
   </dependency>

   <dependency>
      <groupid>org.apache.tomcat.embed</groupid>
      <artifactid>tomcat-embed-jasper</artifactid>
      <scope>provided</scope>
   </dependency>

   <dependency>
      <groupid>javax.servlet</groupid>
      <artifactid>jstl</artifactid>
   </dependency>

   ...

</dependencies>
```

File: application.properties

Demo: ws-03-rest-store-part2.tar.gz

```
server.port=8200

spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp

...
```

Path: src/main/webapp/WEB-INF/jsp

Spring REST Client: ***RestTemplate***

Client: Consuming a REST WebService

- Alternative: **WebFlux** project comes with **WebClient**:
  - Provides a traditional synchronous API, but it also supports an efficient nonblocking and asynchronous approach.
- Deprecation Notice: RestTemplate will be deprecated in future versions.

Source: https://www.baeldung.com/rest-template

We'll look at 5 procedures

Spring REST Client: **RestTemplate**

- getForObject()
- getForEntity()
- postForObject()
- postForLocation()
- postForEntity()
- put()
- delete()
- exchange()

File: SemesterClientConfig.java (Bean: RestTemplate)

Client: Consuming a REST WebService

```java
package no.hvl.dat152.semester.client;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class SemesterClientConfig {

   @Bean
   public RestTemplate restTemplate() {
      return new RestTemplate();
   }
}
```

File: SemesterService.java

Client: Consuming a REST WebService

```
1  package no.hvl.dat152.semester.client.service;
2
3  import ...
4
5  @Service
6  @PropertySource("classpath:application.properties")
7  public class SemesterService {
8
9     @Value("${rest.api.url.semesters")
10    private String BASE_URL;
11
12    @Autowired
13    RestTemplate template;
14
15    public List< Semester> getAll(){
16       ResponseEntity< Semester[]> response =
17                          template.getForEntity(BASE_URL, Semester[].class);
18       return Arrays.asList(response.getBody());
19    }
20 }
```

File: SemesterService.java

Client: Consuming a REST WebService

```
1  package no.hvl.dat152.semester.client.service;
2
3  import ...
4
5  @Service
6  @PropertySource("classpath:application.properties")
7  public class SemesterService {
8
9     @Value("${rest.api.url.semesters")
10    private String BASE_URL;
11
12    @Autowired
13    RestTemplate template;
14
15    public List< Semester> getAll(){
16       ResponseEntity< Semester[]> response =
17                              template.getForEntity(BASE_URL, Semester[].class);
18       return Arrays.asList(response.getBody());
19    }
20 }
```

File: application.properties

Client: Consuming a REST WebService

```
1 ...
2
3 # REST APIs
4 rest.api.port=8299
5 rest.api.server=localhost
6 rest.api.url.base=http://${rest.api.server}:${rest.api.port}
7 rest.api.url.semesters=${rest.api.url.base}/semesters
8 rest.api.url.students=${rest.api.url.base}/students
```

File: SemesterService.java

Client: Consuming a REST WebService

```
1  package no.hvl.dat152.semester.client.service;
2
3  import ...
4
5  @Service
6  @PropertySource("classpath:application.properties")
7  public class SemesterService {
8
9     @Value("${rest.api.url.semesters")
10    private String BASE_URL;
11
12    @Autowired
13    RestTemplate template;
14
15    public List< Semester> getAll(){
16       ResponseEntity< Semester[]> response =
17                              template.getForEntity(BASE_URL, Semester[].class);
18       return Arrays.asList(response.getBody());
19    }
20 }
```

File: SemesterClientApplicationTests.java

JUnit5: Consuming a REST WebService

```
1  @SpringBootTest
2  class SemesterClientApplicationTests {
3
4     @Autowired
5     SemesterService service;
6
7     @Test
8     @DisplayName("Check Count Semesters")
9     public void checkCountSemesters() throws Exception {
10       int expectedValue=2;
11
12       List< Semester> list = service.getAll();
13
14       assertEquals(expectedValue, list.size());
15    }
16
17 }
```

File: SemesterClientApplicationTests.java

JUnit5: Consuming a REST WebService

```java
@SpringBootTest
class SemesterClientApplicationTests {

    @Autowired
    SemesterService service;

    @Test
    @DisplayName("Check Count Semesters")
    public void checkCountSemesters() throws Exception {
        int expectedValue=2;

        List< Semester> list = service.getAll();

        assertEquals(expectedValue, list.size());
    }

}
```

File: SemesterController.java

Spring MVC and JSTL (repetition)

```java
@Controller
@RequestMapping(SemesterController.BASE_URL)
public class SemesterController {

    final static String BASE_URL = "/semesters";

    @Autowired
    private SemesterService service;

    @GetMapping
    public String getAllSemesters(Model model) {
        model.addAttribute("items", service.getAll());
            return "semesters";
    }
}
```

File: SemesterController.java

Spring MVC and JSTL (repetition)

```java
@Controller
@RequestMapping(SemesterController.BASE_URL)
public class SemesterController {

    final static String BASE_URL = "/semesters";

    @Autowired
    private SemesterService service;

    @GetMapping
    public String getAllSemesters(Model model) {
        model.addAttribute("items", service.getAll());
        return "semesters";
    }
}
```

*File: webapp/WEB-INF/jsp/**semesters.jsp***

Spring MVC and JSTL (repetition)

We'll look at this file in Eclipse.

Extending SemesterService

File: SemesterService.java

```java
@Service
@PropertySource("classpath:application.properties")
public class SemesterService {

    @Value("${rest.api.url.semesters}")
    private String BASE_URL;

    @Autowired
    RestTemplate template;

    ...

    public Optional< Semester> getSemesterById(Long id){
        String url = BASE_URL + "/{id}";
        Map< String, Long> params = new HashMap< String,Long>();
        params.put("id", id);

        return Optional.of(template.getForObject(url, Semester.class, params));
    }

    ...
}
```

Extending SemesterService

RestTemplate.**getForObject()**

```
@Service
@PropertySource("classpath:application.properties")
public class SemesterService {

    @Value("${rest.api.url.semesters}")
    private String BASE_URL;

    @Autowired
    RestTemplate template;

    ...

    public Optional< Semester> getSemesterById(Long id){
        String url = BASE_URL + "/{id}";
        Map< String, Long> params = new HashMap< String,Long>();
        params.put("id", id);

        return Optional.of(template.getForObject(url, Semester.class, params));
    }

    ...
}
```

Adding a service class for Courses

File: CourseService.java

```
1  @Service
2  @PropertySource("classpath:application.properties")
3  public class CourseService {
4
5     @Value("${rest.api.url.courses}")
6     private String BASE_URL;
7
8     @Autowired
9     RestTemplate template;
10
11    public List< Course> getAll(){
12       ResponseEntity< Course[]> response = template.getForEntity(BASE_URL, Course[].cl
13       List< Course> courses = Arrays.asList(response.getBody());
14
15       return courses;
16    }
17 }
```

Adding a service class for Courses

RestTemplate.**getForEntity()**

```
1  @Service
2  @PropertySource("classpath:application.properties")
3  public class CourseService {
4
5     @Value("${rest.api.url.courses}")
6     private String BASE_URL;
7
8     @Autowired
9     RestTemplate template;
10
11    public List< Course> getAll(){
12       ResponseEntity< Course[]> response = template.getForEntity(BASE_URL, Course[].cl
13       List< Course> courses = Arrays.asList(response.getBody());
14
15       return courses;
16    }
17 }
```

/viewsemester?id={id}

File: SemesterController.java

```
1 @GetMapping("/viewsemester")
2 public String viewSemester(Model model, @RequestParam("id") String id) {
3    Optional< Semester> object = semesterService.getSemesterById(Long.valueOf(id));
4    Semester semester = object.get();
5    model.addAttribute("semester", semester);
6 
7    return "semester";
8 }
```

File: ***semester.jsp***

/viewsemester?id={id}

```
<h3>Semesters</h3>
<table border="1">
   <tbody><tr>
      <th>Id</th>
      <th>Year</th>
      <th>Period</th>
   </tr>
   <tr>
      <td>${semester.id}</td>
      <td>${semester.studyYear}</td>
      <td>${semester.studyPeriod.equals('A')? 'Autumn' : 'Spring'}</td>
   </tr>
</tbody></table>
<p><a href="/updatesemester?id=${semester.id}">Edit</a> |
   <a href="/deletesemester?id=${semester.id}">Delete</a> |
   <a href="/viewsemesters">Home</a>
</p>
```

REST repsonse: /semesters/{id} (id=3)

Semester includes a list of courses


```
{
   "id": 3,
   "studyYear": "2022",
   "studyPeriod": "A",
   "courses": [   {
      "id": 1,
      "shortName": "DAT108",
      "courseName": "Web Programming",
      "studyPoints": 10,
      "studyYear": 0
   }]
}
```

/viewsemester?id={id}

File: SemesterController.java

```
@GetMapping("/viewsemester")
public String viewSemester(Model model, @RequestParam("id") String id) {
   Optional< Semester> object = semesterService.getSemesterById(Long.valueOf(id));
   Semester semester = object.get();
   model.addAttribute("semester", semester);

   List< Course> courses = semester.getCourses();
   if (courses.size()>0) {
      model.addAttribute("courses", courses);
   }

   List< Course> availableCourses = courseService.getAll();
   if (courses.size()>0) {
      model.addAttribute("availableCourses", availableCourses);
   }

   return "semester";
}
```

Receiving several attributes

Refactoring **semester.jsp**

We'll look at this file in Eclipse.

*Continuing with **RestTemplate***

The postForObject() API

```
T postForObject(URI url, Object request, Class< T> responseType)
```

```
T postForObject(String url, Object request, Class< T> responseType, Map< String,?> uriVari
```

```
T postForObject(String url, Object request, Class< T> responseType, Object... uriVariables
```

Refactoring ***SemesterService.java***

The postForObject() API

```
1  @Service
2  @PropertySource("classpath:application.properties")
3  public class SemesterService {
4
5     @Value("${rest.api.url.semesters}")
6     private String BASE_URL;
7
8     @Autowired
9     RestTemplate template;
10
11    ...
12
13    public Semester save(String year, String period) throws URISyntaxException {
14       URI uri = new URI(BASE_URL);
15
16       HttpHeaders headers = new HttpHeaders();
17       headers.setContentType(MediaType.APPLICATION_JSON);
18       HttpEntity< Semester> request = new HttpEntity<>(new Semester(year, period), hea
19
20       Semester semester = template.postForObject(uri, request, Semester.class);
21
22       return semester;
23    }
24 }
```

Refactoring ***SemesterService.java***

The postForObject() API

```
1  @Service
2  @PropertySource("classpath:application.properties")
3  public class SemesterService {
4
5     @Value("${rest.api.url.semesters}")
6     private String BASE_URL;
7
8     @Autowired
9     RestTemplate template;
10
11    ...
12
13    public Semester save(String year, String period) throws URISyntaxException {
14       URI uri = new URI(BASE_URL);
15
16       HttpHeaders headers = new HttpHeaders();
17       headers.setContentType(MediaType.APPLICATION_JSON);
18       HttpEntity< Semester> request = new HttpEntity<>(new Semester(year, period), hea
19
20       Semester semester = template.postForObject(uri, request, Semester.class);
21
22       return semester;
23    }
24 }
```

Refactoring ***SemesterController.java***

/addsemester (GET and POST)>

```java
@GetMapping("/addsemester")
public String createSemester() {
   return "addsemester";
}

@PostMapping("/addsemester")
public RedirectView createSemesterSave(@RequestParam String studyYear, @RequestParam S
                                       RedirectAttributes redirectAttr) throws URISynt

   Optional< Semester> semester = Optional.of(semesterService.save(studyYear, studyPer

   if (!semester.isPresent()) {
      redirectAttr.addFlashAttribute("errmsg","A error occured when trying to save Sem
      return new RedirectView("/addsemesters");
   }
   redirectAttr.addFlashAttribute("addSemesterSuccess", true);
   redirectAttr.addFlashAttribute("newSemesterId", semester.get().getId());

   return new RedirectView("/viewsemesters");

}
```

Refactoring ***SemesterController.java***

/addsemester (GET and POST)>

```
<form action="/addsemester" method="POST">
   <table border="0"><tbody>
         <tr>
            <td>Year</td>
            <td><input type="text" name="studyYear" value="${studyYear}"></td>
         </tr>
         <tr>
            <td>Period</td>
            <td><input type="text" name="studyPeriod" value="${studyPeriod}"> </td>
         </tr>
      </tbody>
   </table>
   <p><input type="submit" name="saveitembutton" value="Save">
      <input class="button" type="button" onclick="window.location.replace('/viewsemes
</form>
```

Refactoring ***SemesterController.java***

/addsemester (GET and POST)>

```java
@GetMapping("/addsemester")
public String createSemester() {
   return "addsemester";
}

@PostMapping("/addsemester")
public RedirectView createSemesterSave(@RequestParam String studyYear,
                                       @RequestParam String studyPeriod,
                                       RedirectAttributes redirectAttr) throws URISynt

   Optional< Semester> semester = Optional.of(semesterService.save(studyYear, studyPer

   if (!semester.isPresent()) {
      redirectAttr.addFlashAttribute("errmsg","A error occured when trying to save Sem
      return new RedirectView("/addsemesters");
   }
   redirectAttr.addFlashAttribute("addSemesterSuccess", true);
   redirectAttr.addFlashAttribute("newSemesterId", semester.get().getId());

   return new RedirectView("/viewsemesters");

}
```

Checking flashAttributes

Refactoring **semesters.jsp**

```
1 <c:if test="${addSemesterSuccess}">
2    <div>Successfully added Item with ID: ${newSemesterId}</div>
3 </c:if>
```

*Continuing with **RestTemplate - The put() API***

Refactoring: SemesterService.java

```
1  public Semester update(Semester semester) throws URISyntaxException {
2     URI uri = new URI(BASE_URL + "/" + semester.getId());
3
4     HttpHeaders headers = new HttpHeaders();
5     headers.setContentType(MediaType.APPLICATION_JSON);
6     HttpEntity< Semester> request = new HttpEntity<>(semester, headers);
7
8     template.put(uri, request);
9
10    return semester;
11 }
```

***Disadvantage:** put() returns "void"*

*Continuing with **RestTemplate - The exchange() API***

Refactoring: SemesterService.java

```
1  public ResponseEntity< Semester> updateV2(Semester semester) throws URISyntaxException
2     URI uri = new URI(BASE_URL + "/" + semester.getId());
3
4     HttpEntity< Semester> request = new HttpEntity<>(semester);
5     ResponseEntity< Semester> response =
6                                 template.exchange(uri, HttpMethod.PUT, request, Semester
7
8     if (response.getStatusCode().equals(HttpStatus.CREATED)) {
9        return ResponseEntity.ok().body(semester);
10    }
11    return ResponseEntity.notFound().build();
12 }
```

*Continuing with **RestTemplate - The delete() API***

Refactoring: SemesterService.java

```
1  public void delete(Semester semester) throws URISyntaxException {
2     URI uri = new URI(BASE_URL + "/" + semester.getId());
3
4     template.delete(uri);
5  }
6
7  public void delete(Long id) throws URISyntaxException {
8     URI uri = new URI(BASE_URL + "/" + id);
9
10    template.delete(uri);
11 }
```

***Disadvantage:** delete() returns "void"*

Continuing with ***RestTemplate - The exchange() API***

Refactoring: SemesterService.java

```
1  public ResponseEntity< Semester> deleteV2(Semester semester) throws URISyntaxException
2     URI uri = new URI(BASE_URL + "/" + semester.getId());
3
4     HttpEntity< Semester> request = new HttpEntity<>(semester);
5     ResponseEntity< Semester> response =
6                               template.exchange(uri, HttpMethod.DELETE, request, Semester
7
8     if (response.getStatusCode().equals(HttpStatus.OK)) {
9        return ResponseEntity.ok().body(semester);
10    }
11    return ResponseEntity.notFound().build();
12 }
```

Disadvantage: *delete() returns "void"*

Adding a new PostMapping with location

Revisiting our RestController

```
@PostMapping
public ResponseEntity< Semester> createSemesterLocation(@RequestBody Semester semester
   URI location = null;
   Semester s = null;
   try {
      s = repository.save(semester);
      location = new URI("http://localhost:8299/semesters/"+s.getId());
   } catch (Exception e) {
      return ResponseEntity.badRequest().build();
   }
   return ResponseEntity.created(location).body(s);
}
```

Starting to become restful (but not quite there yet)

postForLocation

```
1 HttpEntity< Semester> request = new HttpEntity<>(new Semester("2024","A"));
2 URI location = restTemplate.postForLocation(fooResourceUrl, request);
```

Western Norway University of Applied Sciences

Next

Home