Western Norway
University of
Applied Sciences

Building a RESTFull Store WebService - Part 3 (Spring HATEOAS and Spring Security)

Created by Lasse Jenssen

Home

Agenda

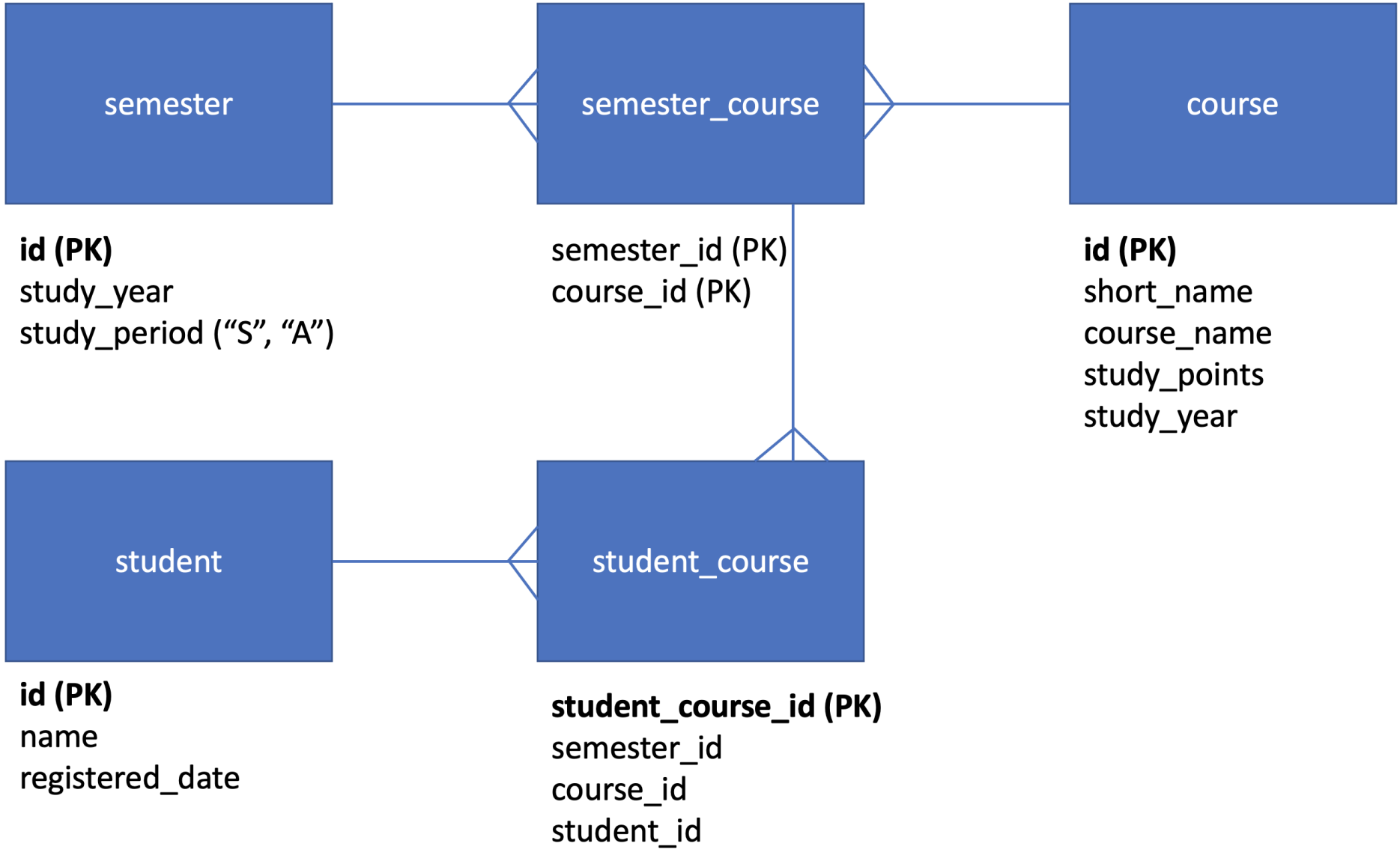
Last Lecture: Web Frameworks

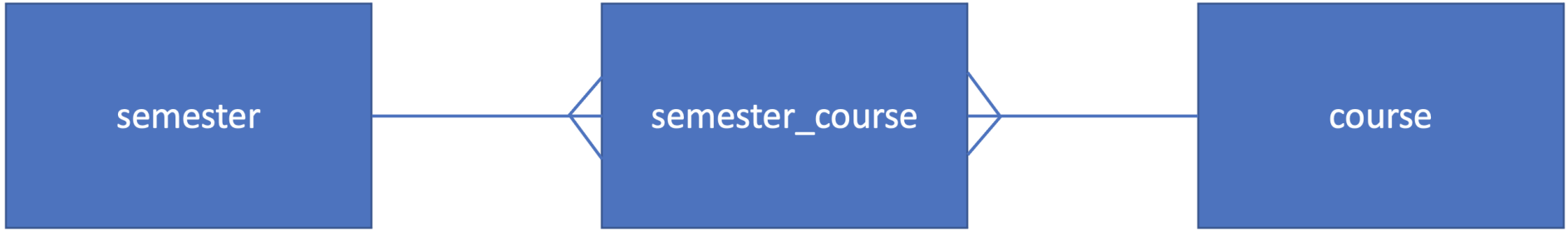
- Part 1: (Spring) **HATEOAS** -
  Hypermedia As The Engine Of Application State. (REST Web Service)
- Part 2: (Spring) **Security** (Web Application)

REST Store Web Service

Demo: ws-03-rest-store-part3.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-part3.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

Sources

What is REST and when is my service truly RESTful?

Link: https://allaroundjava.com/what-is-rest-when-is-service-restful/

Spring HATEOAS Documention

Link: https://spring.io/projects/spring-hateoas

Six architectual constraints for a truly (REPITITION)

RESTFul WebService (API)

First architectual constraints

Client - Server Architecture

Enforces **Separation of concerns**.

Separating the user interface concerns from the data storage concerns.

Second architectual constraints

Statelessness

No session information is retained by the server.

The client application must entirely keep the session state.

Third architectual constraints

Cacheability

A response should implicitly or explicitly label itself as cacheable or non-cacheable.

Fourth architectual constraints

Layered System

A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way (for instance proxy or load balancer).

Fifth architectual constraints

Code on demand (optional)

Servers can temporarily extend or customize the functionality of a client by transferring executable code: for example, compiled components such as Java applets, or client-side scripts such as JavaScript.

Sixth architectual constraints

Uniform Interface

1. Identification of Resources (**URI**)
2. Manipulation of resources through **representations** (Usually JSON objects).
3. **Self-descriptive** messages. For instance:
    - Request type and protocol: `GET / HTTP/1.1`
    - Protocol and response status: `HTTP/1.1 200 OK`
    - Media type: `Content-Type: text/html`
4. **Hypermedia** as the engine of application state (HATEOAS).

HATEOAS

Hypermedia as the engine of application state

When is a service RESTFul enough?

Richardson Maturity model

Level 0 - Single URI over a single HTTP method

Richardson Maturity model

- All the calls to our service are done through a single HTTP method (typically POST).
- All the calls are done through the same URI.

- Resembles SOAP style remote procedure calls the most, although it's neither SOAP nor REST.

```
POST http://localhost:8080/semesters
```

Level 1 - Multiple URIs over a single HTTP method

Richardson Maturity model

- This level introduces unique ids through URIs

```
POST http://localhost:8080/semesters/1
```

- All the operation on a semester would be operated on this unique URI on a single HTTP method

Level 2 - Multiple URIs over multiple HTTP methods

Richardson Maturity model

- This level makes use of the Uniform interface REST principle.
- Ensures all the operations we can perform on a resource are tied to proper interface methods
- Most popular level (and the level we have been on so far)

```
1 GET http://localhost:8080/semesters
2 GET http://localhost:8080/semesters/1
3 POST http://localhost:8080/semesters
4 PUT http://localhost:8080/semesters/1
5 DELETE http://localhost:8080/semesters/1
```

Level 3 - Hypermedia Links

Richardson Maturity model

- Adds **Hypermedia Links** to Level 2 services so that they are **self-descriptive**.
- HATEOAS -> Hypermedia As The Engine Of Application State

```
{
   "id" : "1",
   "studyYear" : "2022",
   "studyPeriod" : "A",
   "_links" : {
      "self" : {
         "href" : "http://localhost:8080/semesters/1"
      },
      "semesters" : {
         "href" : "http://localhost:8080/semesters"
      }
   }
}
```

Could also include TEMPLATES

Richardson Maturity model

```
{
    "id" : "1",
    "studyYear" : "2022",
    "studyPeriod" : "A",
    ...,
    "_templates" : {
      "default" : {
        "method" : "put",
        "properties" : [ {
          "name" : "studyYear",
          "required" : true
        }, {
          "name" : "studyPeriod",
          "required" : true
        } ]
      }
    }
}
```

Hypermedia As The Engine of Application State

Spring HATEOAS

- Provides some APIs to ease creating REST representations that follow the HATEOAS principle
  (when working with Spring and especially Spring MVC).
- **Source**: An Introduction to Spring HATEOAS, Baeldung:
  https://www.baeldung.com/spring-hateoas-tutorial

Maven Dependencies: with Spring Boot

Preparing for **Spring HATEOAS**

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

Maven Dependencies: with Spring Framework

Preparing for **Spring HATEOAS**

```
<dependency>
   <groupid>org.springframework.hateoas</groupid>
   <artifactid>spring-hateoas</artifactid>
   <version>1.4.1</version>
</dependency>
<dependency>
   <groupid>org.springframework.plugin</groupid>
   <artifactid>spring-plugin-core</artifactid>
   <version>1.2.0.RELEASE</version>
</dependency>
```

A class that transform a Semester

SemesterModelAssembler

- **Input**: Semester
- **Output**: EntityModel< Semester>

We'll be using Spring HATEOAS to do the magic.

A class that transform a Semester

SemesterModelAssembler

Input (as an object, but shown as JSON below):

```
{
    "id" : "1",
    "studyYear" : "2022",
    "studyPeriod" : "A",
}
```

A class that transform a Semester

SemesterModelAssembler

Output (as an object, but shown as JSON below):

```
{
    "id" : "1",
    "studyYear" : "2022",
    "studyPeriod" : "A",
    "_links" : {
       "self" : {
          "href" : "http://localhost:8299/semesters/1"
       },
       "semesters" : {
          "href" : "http://localhost:8299/semesters"
       }
    }
}
```

RepresentationModelAssembler *interface*

SemesterModelAssembler

```
1  import org.springframework.hateoas.server.RepresentationModelAssembler;
2
3  @Component
4  public class SemesterModelAssembler
5                   implements RepresentationModelAssembler< Semester, EntityModel< Semester>> {
6
7     @Override
8     public EntityModel< Semester> toModel(Semester semester) {
9
10       return  EntityModel.of(semester,
11          linkTo(SemesterController.class,
12                   (SemesterController.class).getMethod("getSemester", Long.class )
13                ).withSelfRel(),
14          linkTo(SemesterController.class,
15                   (SemesterController.class).getMethod("getAllSemesters")
16                ).withRel("semesters")
17       );
18    }
19 }
```

EntityModel.of() *method*

SemesterModelAssembler

```
1  import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
2  import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
3
4  @Component
5  public class SemesterModelAssembler
6                  implements RepresentationModelAssembler< Semester, EntityModel< Semester>> {
7
8     @Override
9     public EntityModel< Semester> toModel(Semester semester) {
10
11       return  EntityModel.of(semester,
12          linkTo(SemesterController.class,
13                   (SemesterController.class).getMethod("getSemester", Long.class )
14                ).withSelfRel(),
15          linkTo(SemesterController.class,
16                   (SemesterController.class).getMethod("getAllSemesters")
17                ).withRel("semesters")
18       );
19    }
20 }
```

WebMvcLinkBuilder.linkTo() *method*

SemesterModelAssembler

```
1  import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
2  import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
3
4  @Component
5  public class SemesterModelAssembler
6                  implements RepresentationModelAssembler< Semester, EntityModel< Semester>> {
7
8     @Override
9     public EntityModel< Semester> toModel(Semester semester) {
10
11       return  EntityModel.of(semester,
12          linkTo(SemesterController.class,
13                   (SemesterController.class).getMethod("getSemester", Long.class )
14                ).withSelfRel(),
15          linkTo(SemesterController.class,
16                   (SemesterController.class).getMethod("getAllSemesters")
17                ).withRel("semesters")
18       );
19    }
20 }
```

Testing Assembler

SemesterModelAssembler

```
@Autowired
TestRestTemplate template;

@Autowired
SemesterModelAssembler assembler;

final String baseUri = "/semesters";

@Test
public void testSemesterModelAssembler() throws Exception {
   Semester semester = template.getForObject(baseUri + "/3", Semester.class);

   EntityModel< Semester> model = assembler.toModel(semester);

   assertTrue(model.hasLinks());
}
```

WebMvcLinkBuilder.methodOn() *method*

SemesterModelAssembler

```
1  import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
2  import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
3
4  @Component
5  public class SemesterModelAssembler
6                   implements RepresentationModelAssembler< Semester, EntityModel< Semester>> {
7
8     @Override
9     public EntityModel< Semester> toModel(Semester semester) {
10
11       return EntityModel.of(semester,
12             linkTo(
13                methodOn(SemesterController.class).getSemester(semester.getId())
14             ).withSelfRel(),
15             linkTo(
16                methodOn(SemesterController.class).getAllSemesters()
17             ).withRel("semesters")
18          );
19    }
20 }
```

Back to the controller class (Method without HATEOAS)

File: **SemesterController.java**

```
@GetMapping("/{id}")
public ResponseEntity< Semester> getSemester(@PathVariable("id") Long id) {

	Optional< Semester> semester = repository.findById(id);

	return semester
		.map( s -> ResponseEntity.ok().body(s) )
		.orElseGet( () -> ResponseEntity.notFound().build() );

}
```

Using the SemesterModelAssembler

File: **SemesterController.java**

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

    final static String BASE_URL = "/semesters";

    @Autowired
    private SemesterRepository repository;

    @Autowired
    private SemesterModelAssembler assembler;

    ...
}
```

Using the SemesterModelAssembler

File: **SemesterController.java**

```
1  @RestController
2  @RequestMapping(SemesterController.BASE_URL)
3  public class SemesterController {
4
5     final static String BASE_URL = "/semesters";
6
7     @Autowired
8     private SemesterRepository repository;
9
10    @Autowired
11    private SemesterModelAssembler assembler;
12
13    ...
14 }
```

Using the SemesterModelAssembler

File: **SemesterController.java**

```
1 @GetMapping("/{id}")
2 public ResponseEntity< EntityModel< Semester>> getSemester(@PathVariable Long id){
3
4    Optional< Semester> semester = repository.findById(id);
5
6    return semester
7       .map( s -> ResponseEntity.ok().body(assembler.toModel(s)) )
8       .orElseGet( () -> ResponseEntity.notFound().build() );
9 }
```

Using the SemesterModelAssembler

File: **SemesterController.java**

```
@GetMapping("/{id}")
public ResponseEntity< EntityModel< Semester>> getSemester(@PathVariable Long id){

   Optional< Semester> semester = repository.findById(id);

   return semester
      .map( s -> ResponseEntity.ok().body(assembler.toModel(s)) )
      .orElseGet( () -> ResponseEntity.notFound().build() );
}
```

- Demo: SoapUI

Old: getAllSemesters

File: **SemesterController.java**

```
1 @GetMapping
2 public ResponseEntity< List< Semester>> getAllSemesters() {
3    List< Semester> semesters = new ArrayList<>();
4    repository.findAll().forEach(semesters::add);
5
6    return ResponseEntity.ok().body(semesters);
7 }
```

New: getAllSemesters

File: **SemesterController.java**

```
@GetMapping
public ResponseEntity< CollectionModel< EntityModel< Semester>>> getAllSemesters() {

   List< Semester> semesters = new ArrayList<>();
   repository.findAll().forEach(semesters::add);

   List< EntityModel< Semester>> entitySemesters = semesters
               .stream()
               .map( s -> assembler.toModel(s))
               .collect(Collectors.toList());

   return ResponseEntity.ok().body(CollectionModel.of(entitySemesters,
         linkTo(methodOn(SemesterController.class).getAllSemesters()).withSelfRel()));
}
```

New: getAllSemesters

File: **SemesterController.java**

```
@GetMapping
public ResponseEntity< CollectionModel< EntityModel< Semester>>> getAllSemesters() {

   List< Semester> semesters = new ArrayList<>();
   repository.findAll().forEach(semesters::add);

   List< EntityModel< Semester>> entitySemesters = semesters
               .stream()
               .map( s -> assembler.toModel(s))
               .collect(Collectors.toList());

   return ResponseEntity.ok().body(CollectionModel.of(entitySemesters,
         linkTo(methodOn(SemesterController.class).getAllSemesters()).withSelfRel()));
}
```

New: getAllSemesters

File: **SemesterController.java**

```
@GetMapping
public ResponseEntity< CollectionModel< EntityModel< Semester>>> getAllSemesters() {

    List< Semester> semesters = new ArrayList<>();
    repository.findAll().forEach(semesters::add);

    List< EntityModel< Semester>> entitySemesters = semesters
                .stream()
                .map( s -> assembler.toModel(s))
                .collect(Collectors.toList());

    return ResponseEntity.ok().body(CollectionModel.of(entitySemesters,
          linkTo(methodOn(SemesterController.class).getAllSemesters()).withSelfRel()));
}
```

New: getAllSemesters

File: **SemesterController.java**

```
@GetMapping
public ResponseEntity< CollectionModel< EntityModel< Semester>>> getAllSemesters() {

   List< Semester> semesters = new ArrayList<>();
   repository.findAll().forEach(semesters::add);

   List< EntityModel< Semester>> entitySemesters = semesters
               .stream()
               .map( s -> assembler.toModel(s))
               .collect(Collectors.toList());

   return ResponseEntity.ok().body(CollectionModel.of(entitySemesters,
         linkTo(methodOn(SemesterController.class).getAllSemesters()).withSelfRel()));
}
```

New: getAllSemesters

File: **SemesterController.java**

```
@GetMapping
public ResponseEntity< CollectionModel< EntityModel< Semester>>> getAllSemesters() {

   List< Semester> semesters = new ArrayList<>();
   repository.findAll().forEach(semesters::add);

   List< EntityModel< Semester>> entitySemesters = semesters
               .stream()
               .map( s -> assembler.toModel(s))
               .collect(Collectors.toList());

   return ResponseEntity.ok().body(
         CollectionModel.of(
            entitySemesters,
            linkTo(methodOn(SemesterController.class).getAllSemesters()).withSelfRel()));
}
```

Securing our web application

Demo: ws-03-rest-store-part3-client.tar.gz

- Back to the Client Shopping List Web Application (from last lecture).
- Based on **Spring Boot** and **Spring WebMVC**.
- We'll use JSP and JSTL (instead of Thymeleaf)
- Know we are adding **Spring Security**
- *Code: ws-03-rest-store-part3-client.tar.gz (see course overview)*
  (Note! MUST be ran againt REST Web Service from part 1: ws-03-rest-store-part1.tar.gz)

Securing our web application

Framework **Spring Security**

- **Authentication** & **Authorization** functionalities.
- **Login & Logout** functionalities
- Servlet API integration & Optional integration with Spring Web MVC
- Allow/Block access to URLs for logged in users based on Role
- Protection against attacks like session fixation, click-jacking, cross site request forgery (CSRF), etc.
- Supports various 3rd party integrations to enhance security features.
- *Code: ws-03-rest-store-part3-client.tar.gz (see course overview)*

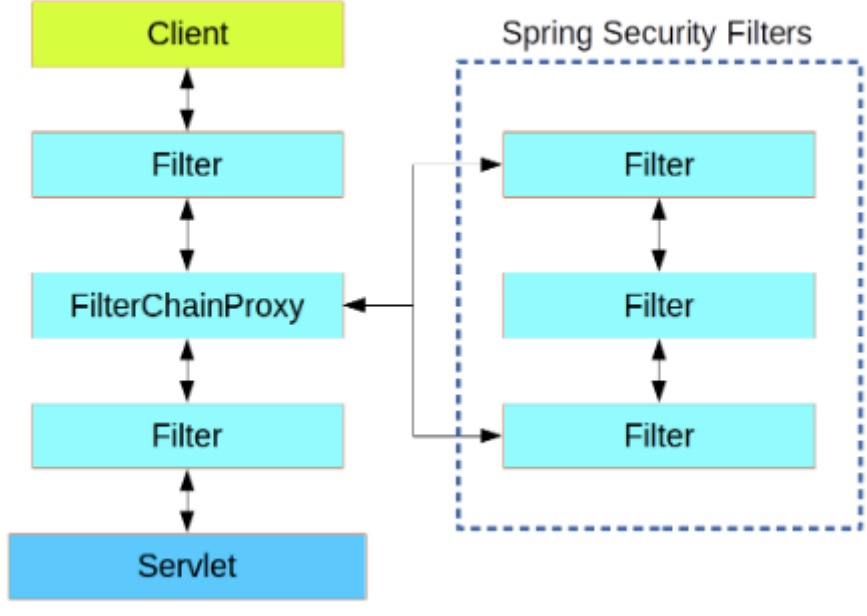
Maven dependencies with Spring Boot

Framework **Spring Security**

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

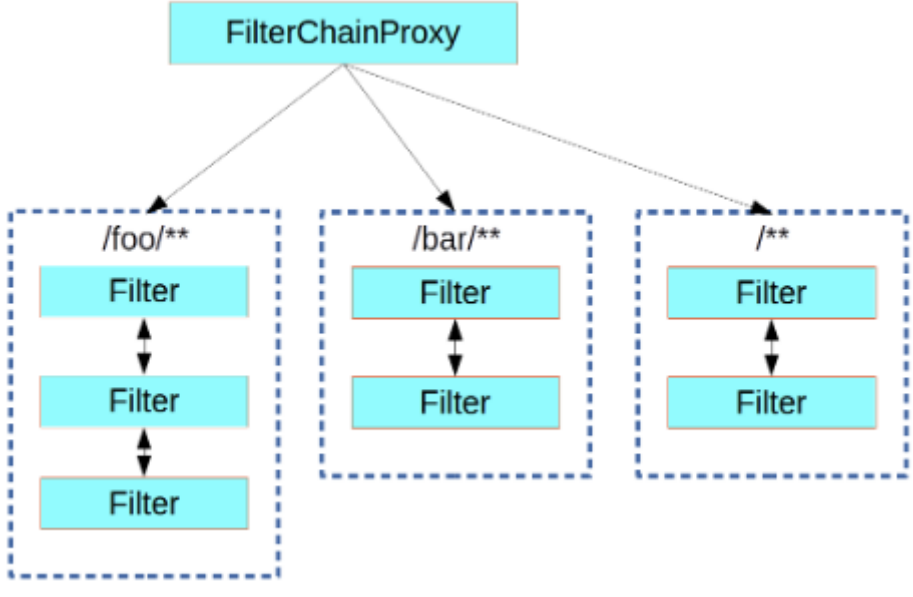
Design Pattern: Filter

Framework **Spring Security**



Design Pattern: Filter

Framework **Spring Security**



Demo: Without Any Configuration

Framework **Spring Security**

Implementing Security

Framework **Spring Security**

```
1  @Configuration
2  @EnableWebSecurity
3  public class SemesterSecurityConfig {
4
5     @Bean
6     public PasswordEncoder passwordEncoder() {
7           return new BCryptPasswordEncoder();
8     }
9
10    @Bean
11    public InMemoryUserDetailsManager userDetailsService() {
12       ...
13
14       return new InMemoryUserDetailsManager(...);
15    }
16
17 }
```

Implementing Security

Framework **Spring Security**

```
1  @Bean
2  public InMemoryUserDetailsManager userDetailsService() {
3     UserDetails user = User.withUsername("user")
4               .password(passwordEncoder().encode("pass"))
5               .roles("USER")
6               .build();
7        UserDetails admin = User.withUsername("admin")
8               .password(passwordEncoder().encode("secure"))
9               .roles("ADMIN")
10              .build();
11
12       return new InMemoryUserDetailsManager(user, admin);
13 }
```

*Demo: With **Some** Configuration*

Framework **Spring Security**

Implementing Security

Framework **Spring Security**

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
   http.authorizeRequests()
      .antMatchers("/admin/**")
      .hasRole("ADMIN")
      .antMatchers("/login*")
      .permitAll()
      .anyRequest()
      .authenticated()
      .and()
      .formLogin()
      .defaultSuccessUrl("/viewsemesters", true)
      .and()
      .logout();

   return http.build();
}
```

*Demo: With **More** Configuration*

Framework **Spring Security**

Western Norway
University of
Applied Sciences

Congratulation: Web Frameworks is completed

Well ... only the mandatory exercise and exam left ;-)

Home