Building a RESTFull Store WebService

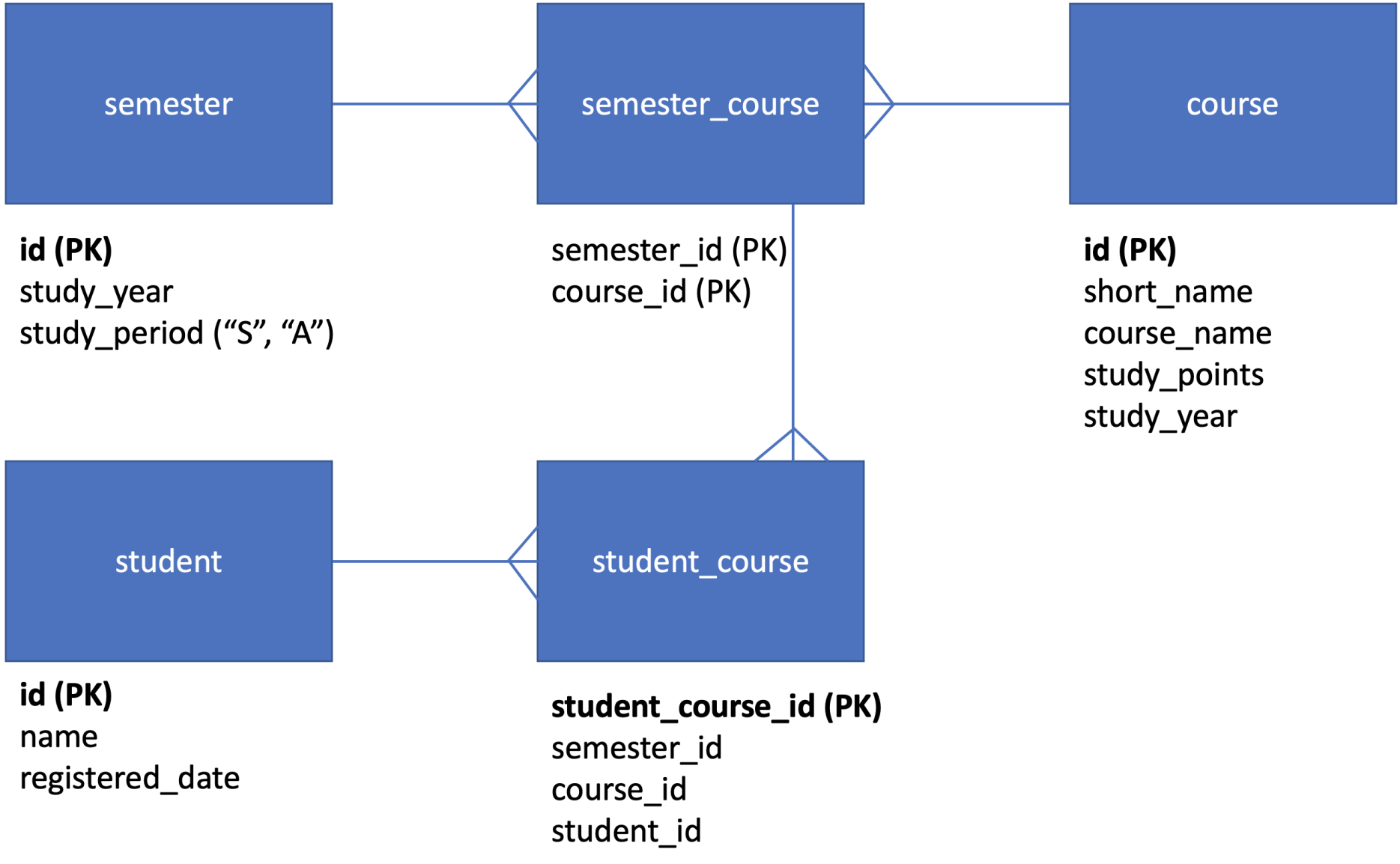
Created by Lasse Jenssen

Home

REST Store Web Service

Demo: demo-06-rest-semesters-v1.0

- Smal REST applications keeping track of "Courses" and "Semesters".
- Based on **Spring Boot** and **Spring WebMVC**.
- We'll use an H2 database
- We'll use Spring Data and JPA (JpaRepository): Hibernate (ORM API)
- *Code: demo-06-rest-semesters-v1.0.zip (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

REST API

Demo: demo-06-rest-semesters-v1.0

- /students
- /students/{id}
- /courses
- /courses/{id}
- /semesters
- /semesters/{id}

REST API

Demo: demo-06-rest-semesters-v1.0

- /students
- /students/{id}
- /courses
- /courses/{id}
- /semesters
- /semesters/{id}
- /semesters/{id}/courses
- /semesters/{id}/courses/{id}
- /semesters/{id}/courses/{id}/students

REST API

Demo: demo-06-rest-semesters-v1.0

- **GET /courses**: List all courses
- **POST /courses**: Register ny course

- GET /courses?[searchParameters][pagination]

REST API

Demo: demo-06-rest-semesters-v1.0

- **GET /courses/{id}**: Get course
- **PUT /courses/{id}**: Update a course
- **DELETE /courses/{id}**: Delete a course

REST API

Demo: demo-06-rest-semesters-v1.0

- **GET /students**: List all students
- **POST /students**: Register ny student

- GET /students?[searchParameters][pagination]

REST API

Demo: demo-06-rest-semesters-v1.0

- **GET /students/{id}**: Get student
- **PUT /students/{id}**: Update a student
- **DELETE /students/{id}**: Delete a student

- **GET /students/{id}?[filters]**: Eks: include=info | semester | history

REST API

Demo: demo-06-rest-semesters-v1.0

- **GET /semesters**: List all semesters (could also add filters)
- **POST /semesters**: Register ny semester

- **GET /semesters/{id}**: Get semester
- **PUT /semesters/{id}**: Update a semester
- **DELETE /semesters/{id}**: Delete a semester

REST API

Demo: demo-06-rest-semesters-v1.0

GET /semesters/{id}/courses

- Get all courses in a given semester

POST /semesters/{id}/courses

- Add a new course or a list of courses to a semester
- Performance implications?

DELETE /semesters/{id}/courses/{id}

- Remove a course from a semester

DELETE /semesters/{id}/courses

- Remove a list of courses from a semester

REST API

Demo: demo-06-rest-semesters-v1.0

GET /semesters/{id}/courses/{id}/students

- List all students attending a given course a given semester.

POST /semesters/{id}/courses/{id}/students

- Add a student or a list of students to a course in a given semester.

DELETE /semesters/{id}/courses/{id}/students

- Remove a student or a list of students from a course a given semester.

Sources

H2 Database

- Download and documentation:
  https://www.h2database.com/html/main.html
- Articles:
  - https://www.javatpoint.com/spring-boot-h2-database
    (Used in this presentation)

In-memory database

H2 Database

- Very fast, open source, JDBC API.
- Embedded and server modes; in-memory database (or persistant).
- Browser based Console application.
- Small footprint: around 2.5 MB jar file size.
- Uses: Early development, testing and POCs
- https://www.h2database.com/html/main.html

Spring Boot: Maven Dependencies

H2 Database

```
<dependency>
    <groupid>com.h2database</groupid>
    <artifactid>h2</artifactid>
    <scope>runtime</scope>
</dependency>
```

In-memory database

H2 Database

- Simplest form: NO other configuration needed (except from the JPA anotations).
- We'll see how we also can configure the database connection.

application.properties

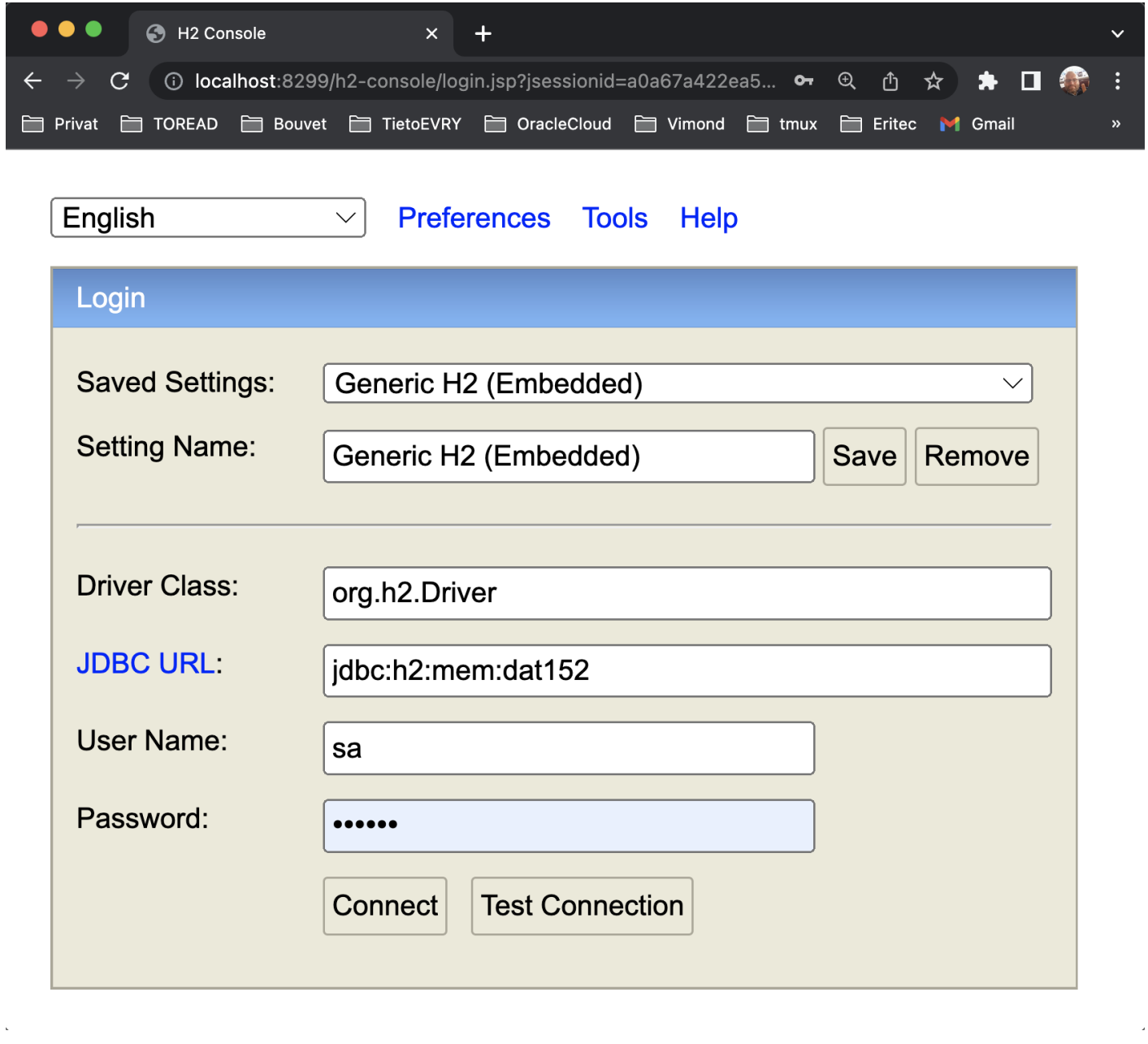
H2 Database

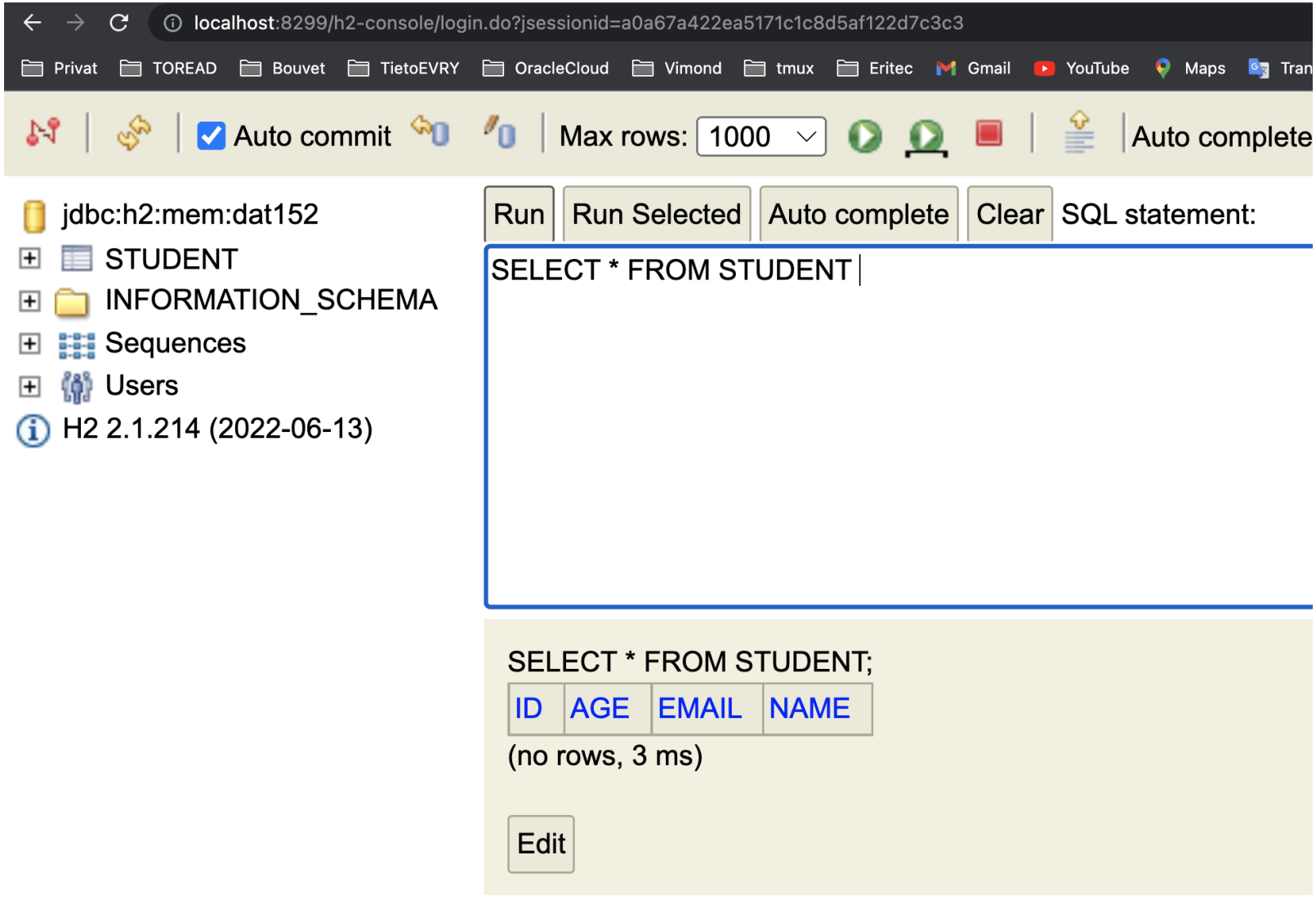
```
spring.datasource.url=jdbc:h2:mem:dat152
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=secure

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

application.properties

H2 Database

```
1 server.port=8299
2 ...
3 spring.h2.console.enabled=true
4 spring.h2.console.port=8299
```


H2 Console
localhost:8299/h2-console/login.jsp?jsessionid=a0a67a422ea5...
Privat
TOREAD
Bouvet
TietoEVRY
OracleCloud
Vimond
tmux
Eritec
Gmail
English
Preferences
Tools
Help
Login
Saved Settings:
Generic H2 (Embedded)
Setting Name:
Generic H2 (Embedded)
Save
Remove
Driver Class:
org.h2.Driver
JDBC URL:
jdbc:h2:mem:dat152
User Name:
sa
Password:
Connect
Test Connection


localhost:8299/h2-console/login.do?jsessionid=a0a67a422ea5171c1c8d5af122d7c3c3
Privat
TOREAD
Bouvet
TietoEVRY
OracleCloud
Vimond
tmux
Eritec
Gmail
YouTube
Maps
Auto commit
Max rows: 1000
Auto complete
jdbc:h2:mem:dat152
STUDENT
INFORMATION_SCHEMA
Sequences
Users
H2 2.1.214 (2022-06-13)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM STUDENT
SELECT * FROM STUDENT;
ID
AGE
EMAIL
NAME
(no rows, 3 ms)
Edit

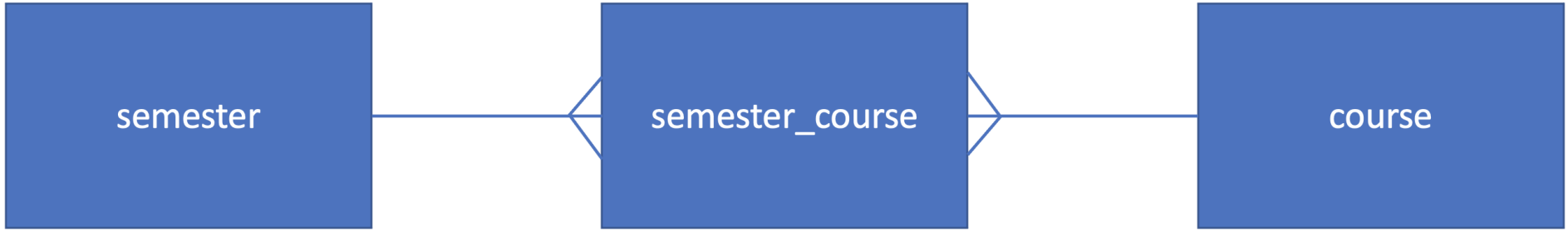
JPA Specification

Spring Data JPA

- "DAT152: not a database class".

  We'll use some magic with Spring Boot, H2, JPA and Hibernate.
- We will use (JPA configurations):
  - @Entity
  - @Id
  - @GeneratedValue(strategy=GenerationType.AUTO)
  - @Column(name="SEMESTER_ID")

  - @OneToMany(fetch = FetchType.LAZY, mappedBy = "course")
  - @ManyToOne(fetch = FetchType.LAZY)
  - @JoinColumn(name = "COURSE_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

*One of the core parts of JPA is an **entity** class*

JPA Specification

"If an entity instance is to be passed by value as a detached object (e.g., through a remote interface), the entity class must implement the Serializable interface."

- In practice, if our object is to leave the domain of the JVM, it'll require serialization.

Model classes: Semester

JPA annotations

```java
@Entity
public class Semester implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long id;

    @Column(name="STUDY_YEAR", length=4, nullable=false, unique=false)
    private String studyYear;

    @Column(name="STUDY_PERIOD", length=1, nullable=false, unique=false)
    private String studyPeriod;

    // Constructors, Getters and Setters
    ...
}
```

Model classes: Course

JPA annotations

```java
@Entity
public class Course implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long id;

    @Column(name="SHORT_NAME", length=6, nullable=false, unique=true)
    private String shortName;

    @Column(name="COURSE_NAME", length=50, nullable=false)
    private String courseName;

    @Column(name="STUDY_POINTS", nullable=false)
    private int studyPoints;

    @Column(name="STUDY_YEAR", length=1)
    private int studyYear;

    // Constructors, Getters and Setters
    ...
}
```

Model classes: Semester

JPA annotations

```java
@Entity
public class Semester implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long id;
    ...

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(
        name = "semester_course",
        joinColumns = @JoinColumn(name = "semester_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id"))
    private List< Course> courses = new ArrayList< Course>();

    public void addCourse(Course course) {
        courses.add(course);
    }

    // Constructors, Getters and Setters
    ...
}
```

Model classes: Course

JPA annotations

```
1  @Entity
2  public class Course implements Serializable {
3
4      private static final long serialVersionUID = 1L;
5
6      @Id
7      @GeneratedValue(strategy=GenerationType.AUTO)
8      private long id;
9
10     ...
11
12     @ManyToMany(fetch = FetchType.LAZY, mappedBy = "courses")
13     private List semesters = new ArrayList();
14
15     public void addSemester(Semester semester) {
16         semesters.add(semester);
17     }
18
19     // Constructors, Getters and Setters
20     ...
21 }
22
```

```java
@Component
class BootstrapCommandLineRunner implements CommandLineRunner  {
    @Autowired
    CourseRepository courseRepo;
    @Autowired
    SemesterRepository semesterRepo;

    @Override
    public void run(String... args) throws Exception {

        // Loading Courses
        Course c1 = new Course();
        c1.setCourseName("Web Programming");
        c1.setShortName("DAT108");
        c1.setStudyPoints(10);

        // Loading Semesters
        Semester s1 = new Semester();
        s1.setStudyYear("2022");
        s1.setStudyPeriod("A");

        // Linking
        s1.addCourse(c1);

        courseRepo.save(c1);
        semesterRepo.save(s1));
    }
}
```

Let's start the application

JPA, H2, Hibernate Demo

```
1 ...
2 spring.jpa.show-sql=true
3 spring.jpa.properties.hibernate.format_sql=true
4 ...
5 spring.h2.console.enabled=true   // default uri: h2-console
6 spring.h2.console.port=8299
```

More magic: SemesterRepository

Spring Data: Repository (DAO)

```java
package no.hvl.dat152.h2databaseexample.repository;

import org.springframework.data.repository.CrudRepository;
import no.hvl.dat152.h2databaseexample.model.Semester;

public interface SemesterRepository extends CrudRepository< Semester, Long>  {

}
```

More magic: CourseRepository

Spring Data: Repository (DAO)

```java
package no.hvl.dat152.h2databaseexample.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.jpa.repository.Query;

import no.hvl.dat152.h2databaseexample.model.Course;

public interface CourseRepository extends CrudRepository< Course, Long> {

    Course findByShortName(String shortName);

    @Query("SELECT c FROM Course c WHERE LOWER(c.shortName) = LOWER(:shortName
    Course retrieveByShortName(@Param("shortName") String shortName);

}
```

Let's finally write the REST controllers

@RestController

Class: SemesterController

@RestController

```java
package no.hvl.dat152.h2databaseexample.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import no.hvl.dat152.h2databaseexample.repository.SemesterRepository;

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

    final static String BASE_URL = "/semesters";

    @Autowired
    private SemesterRepository repository;

    ...
}
```

Method: getAllSemesters

@RestController

```
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     @RequestMapping(method = RequestMethod.GET)
11     public List< Semester> getAllSemesters() {
12         List< Semester> semesters = new ArrayList< >();
13         repository.findAll().forEach(semesters::add);
14
15         return semesters;
16     }
17 }
```

Method: getAllSemesters

@RestController

```
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     @GetMapping
11     public List< Semester> getAllSemesters() {
12         List< Semester> semesters = new ArrayList< >();
13         repository.findAll().forEach(semesters::add);
14
15         return semesters;
16     }
17 }
```

Method: getSemester

@RestController

```
1 @GetMapping("/{id}")
2 public Semester getSemester(@PathVariable("id") Long id) {
3
4     Optional< Semester> semester = repository.findById(id);
5
6     return semester.orElse(null);
7
8 }
```

***Demo (SoapUI)**: Call API with both existing and non-existing id (HttpStatus?).*

Method: getSemester

@RestController

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

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

    return semester
            .map( s -> ResponseEntity.ok().body(s) )                    //200 OK
            .orElseGet( () -> ResponseEntity.notFound().build() );    //404 Not

}
```

*Return **ResponseEntity** to controll HttpStatus.*

Method: createSemester

@RestController

```
1 @PostMapping
2 public Semester createSemester(@RequestBody Semester semester) {
3
4     return repository.save(semester);
5
6 }
```

@RequestBody: *JSON data as part of body in request (See SoapUI)*

Demo (SoapUI): *Call API with complete and non-complete JSON (missing NOT NULL).*

Method: createSemester

@RestController

```
@PostMapping
public ResponseEntity< Semester> createSemester(@RequestBody Semester semester
    Semester s = null;
    try {
        s = repository.save(semester);
    } catch (Exception e) {
        return ResponseEntity.badRequest().build();
    }
    return ResponseEntity.ok().body(s);
}
```

@RequestBody: *JSON data as part of body in request (See SoapUI)*

Demo (SoapUI): *Call API with complete and non-complete JSON (missing NOT NULL).*

Method: deleteSemester

@RestController

```
1 @DeleteMapping("/{id}")
2 public void deleteSemester(@PathVariable("id") Long id) {
3         repository.deleteById(id);
4 }
```

***@PathVariable**: When parameter as part of URI (uri/{param})*

***@RequestParam**: When parameter as query string (uri?param=value)*

***Demo (SoapUI)**: Test with existing and non-existing id.*

Method: deleteSemester

@RestController

```
@DeleteMapping("/{id}")
public ResponseEntity< Semester> deleteSemester(@PathVariable("id") Long id) {
    Optional< Semester> semester = repository.findById(id);

    if (semester.isPresent()) {
        repository.deleteById(id);
        return ResponseEntity.ok().build();
    }
    return ResponseEntity.notFound().build();
}
```

Why is this a bad idea?

Method: deleteSemester

@RestController

```
@DeleteMapping("/{id}")
public ResponseEntity< Semester> deleteSemester(@PathVariable("id") Long id) {
    try {
        repository.deleteById(id);
    } catch (Exception e) {
        return ResponseEntity.notFound().build();
    }
    return ResponseEntity.ok().build();
}
```

URI: /semesters/{id}/courses

Nested URI Paths

Extending SemesterRepository

Nested URI Paths

```
1 public interface SemesterRepository extends CrudRepository< Semester, Long>  {
2
3     @Query("SELECT c FROM Course c INNER JOIN c.semesters s WHERE s.id = :id")
4     List< Course> findCoursesBySemesterId(Long id);
5
6 }
```

Extending SemesterController

Nested URI Paths

```
1 @GetMapping("/{id}/courses")
2 public List< Course> getCoursesBySemesterId(@PathVariable("id") Long id) {
3
4     return repository.findCoursesBySemesterId(id);
5
6 }
```

This will give an ERROR (loops in many-to-many, generating JSON)

Refactor Course.java: Removing loop

Nested URI Paths

```
1  @Entity
2  @Data
3  public class Course implements Serializable {
4
5      private static final long serialVersionUID = 1L;
6
7      @Id
8      @GeneratedValue(strategy=GenerationType.AUTO)
9      private long id;
10     ...
11     @Column(name="STUDY_YEAR", length=1)
12     private int studyYear;
13
14     @ManyToMany(fetch = FetchType.LAZY, mappedBy = "courses")
15     @JsonIgnore
16     private List< Semester> semesters = new ArrayList< Semester>();
17
18     ...
```

@JsonIgnore: Stop loop in join

Western Norway
University of
Applied Sciences

Next

Web Services: REST Store - Part 2

Home