ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

UNIVERSIDAD D CÓRDOBA

Programación web

Prácticas

Semana 4: Acceso a base de datos con JDBC (II) y flujo MVC completo

Aurora Ramírez Quesada (aramirez@uco.es)

Departamento de Ciencia de la Computación e Inteligencia Artificial

Universidad de Córdoba

Índice de contenido

Acceso a base de datos con JDBC
Consultas avanzadas

- Algunas de las funciones de la aplicación requieren consultas más complejas:
  - Las entidades se relacionan entre sí, por clave foránea o por tablas resultantes de relaciones N:N
  - Un controlador puede necesitar acceder a varias tablas en secuencia para realizar acciones
  - Si las claves primarias son autoincrementales, los identificadores para consultas parametrizadas no son conocidos a priori
- Extensión del modelo relacional:
  - Ver script actualizado en: */resources/db/create_bd.sql*

Acceso a base de datos con JDBC
Consultas avanzadas

- **Relaciones 1:N** – Asignar profesor a curso

1. Buscar ID del curso a partir del nombre
2. Buscar ID de un profesor que pertenezca a un departamento concreto
3. Asignar el ID del profesor al curso (clave foránea)

Controlador

```
@GetMapping("/assignProfessorToCourse")
public String assignProfessorToCourse() {

    // Find course ID by name
    int idCourse = this.courseRepository.findCourseIdByName(courseName:"Maths");

    // Find professor ID by department
    int idProfessor = this.courseRepository.findProfessorIdByDeparment(departmentName:"Maths and Statistics");

    // Update foreign key
    if(idCourse!=-1 && idProfessor != -1){
        boolean success = this.courseRepository.assignProfessorToCourse(idCourse, idProfessor);
        System.out.println("[AssignProfessorCourseController] Professor (id="
            + idProfessor + ") assigned to course (id=" + idCourse + "): " + success);
    }
    else{
        System.out.println(x:"[AssignProfessorCourseController] Professor or course not found");
    }

    return new String(original:"home");
}
```

- En el ejemplo se omiten algunos aspectos (restricciones de dominio) para facilitar el desarrollo:
  - En el script de creación de la base de datos no se estableció que la clave foránea fuese "NOT_NULL", por lo que podemos crear cursos sin profesor asignado
  - Se devuelve el primer ID encontrado, tanto para el curso como para el profesor (por simplificar)
  - Podría ser necesario gestionarlas, al menos a nivel del controlador, antes de ejecutar las consultas

Acceso a base de datos con JDBC
Consultas avanzadas

- **Relaciones N:N** – Matricular estudiante(s) en curso(s)

  1. Localizar el ID del curso a partir del nombre
  2. Inscribir a cada estudiante (con su id) en el curso
  3. Al ser una relación N:N, existe una tabla específica donde se guarda la vinculación entre ambas entidades

- Al ser un ejemplo, se asumen ciertos aspectos:
  - El controlador no tiene por qué conocer los ID de los estudiantes a matricular, seguramente los recibirá de una vista
  - Hemos asumido que pueden existir cursos sin estudiantes matriculados, y también estudiantes que no están matriculados en ningún curso

Controlador

```java
@GetMapping("/enrolStudentsInCourse")
public String enrolStudentsInCourse() {

    // All students must be enrolled in course "Programming"
    int idCourse = this.courseRepository.findCourseIdByName(courseName:"Programming");
    for(int idStudent=1; idStudent<=6; idStudent++){
        this.courseRepository.enrolStudentInCourse(idStudent, idCourse);
    }

    return new String(original:"home");
}
```

Repositorio

```java
public boolean enrolStudentInCourse(int idStudent, int idCourse){
    try{
        String query = sqlQueries.getProperty(key:"insert-enrolStudentInCourse");
        if(query != null){
            int result = jdbcTemplate.update(query, idStudent, idCourse);
            if (result>0)
                return true;
            else
                return false;
        }
        else
            return false;
        } catch(DataAccessException exception){
            System.err.println(x:"Unable to enrol student in course in the database");
            exception.printStackTrace();
        return false;
    }
}
```

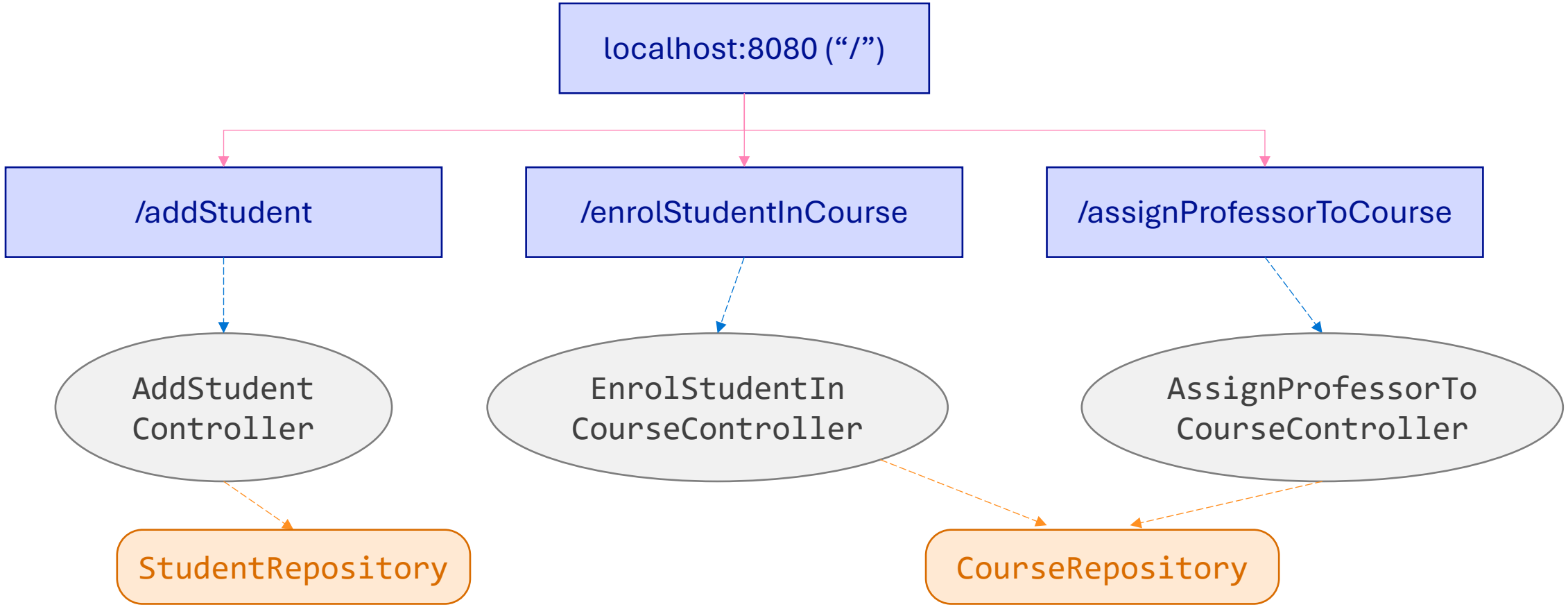
Acceso a base de datos con JDBC
Ejemplo

- Decisión de diseño: ¿qué/cuántos repositorios hacen falta?
  - Los repositorios se encargan de la gestión de objetos de dominio relacionados
  - Existirán varios repositorios, según las necesidades de la aplicación. Para el ejemplo:
    - `StudentRepository`: Busca estudiantes y los añade (semana previa)
    - `CourseRepository`: Crea cursos y le asigna estudiantes y profesores (nuevo)
  - Puede ser interesante definir una jerarquía de repositorios para reutilizar métodos (tenemos más libertad al no extender de interfaces o clases Spring)
    - `AbstractRepository`: Define las propiedades y métodos comunes a todos los repositorios (p. ej. Acceso al fichero *sql.properties*)
  - Estas decisiones tienen implicaciones en el diseño de los controladores, ya que en Spring cada controlador recibe un único repositorio como parámetro en el constructor

Acceso a base de datos con JDBC
Ejemplo

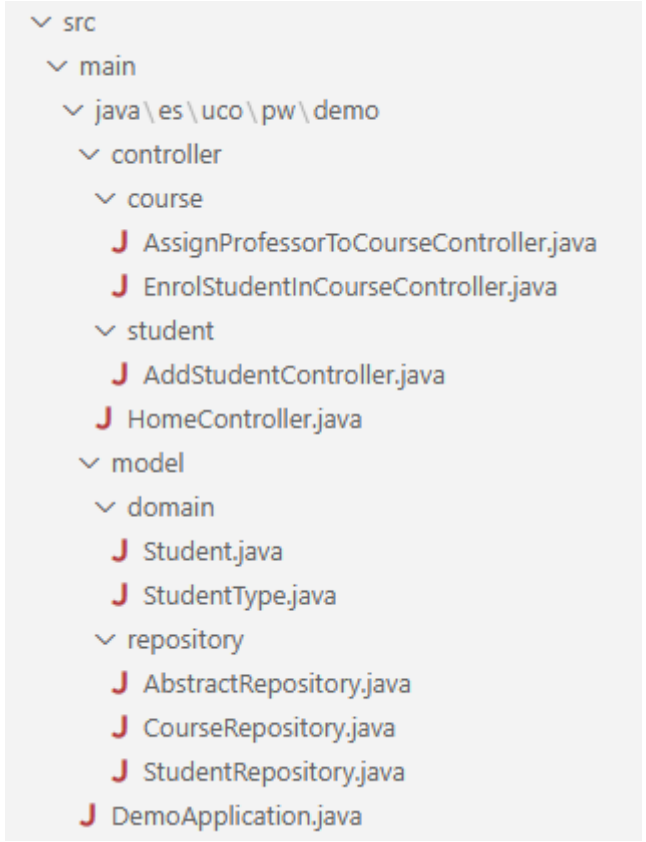
- Relación entre repositorios, controladores y vistas

Como no hemos desarrollado vistas aún, simplemente haremos un mapeo entre la URL y el controlador:
@GetMapping(URL)



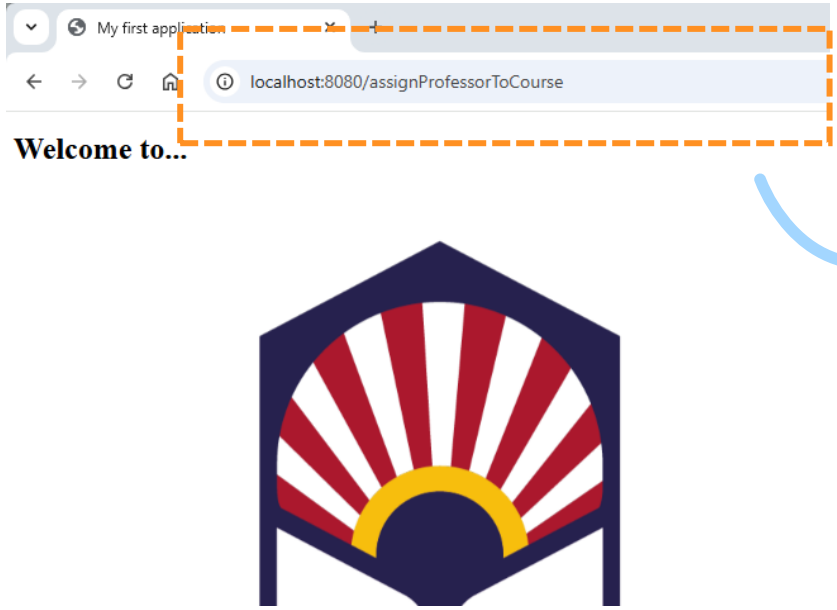
Acceso a base de datos con JDBC
Ejemplo

- A medida que aumenta el número de vistas y controladores, es necesario mantener una nomenclatura uniforme y una correcta organización del proyecto:
  - **Vistas**. Nombre significativo de la funcionalidad (*addStudentView.html*) o del estado tras ejecutar un controlador (*findStudentByIdViewSucess.html* / *findStudentByIdViewFail.html*)
  - **Controladores**. Nombre equivalente al de la vista que lo invoca: `AddStudentController.java`
  - **Modelo**. Separando clases de dominio (`.domain`) y clases repositorio (`.repository`)
  - Organizar controladores en paquetes o directorios según los conceptos de dominio: /student, /course, etc.

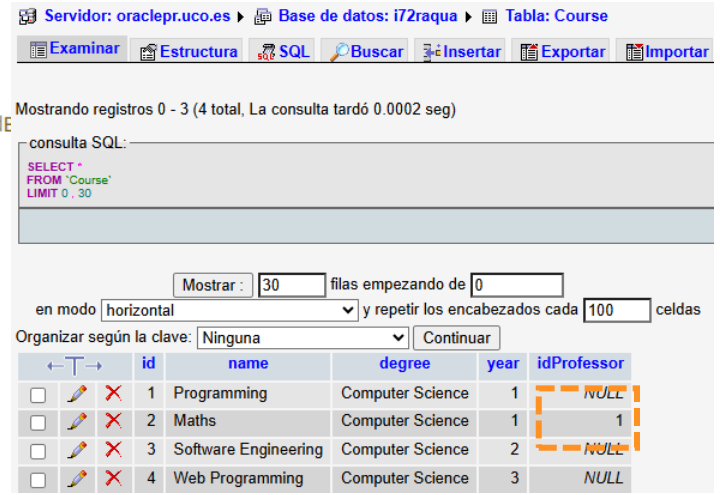


Acceso a base de datos con JDBC
Ejemplo

- Provisionalmente, cada controlador tiene asignada una URL a la que “responde” cuando se escribe esa URL en el navegador (HTTP GET)




```
public class AssignProfessorToCourseController {

    @GetMapping("/assignProfessorToCourse")
    public String assignProfessorToCourse() {

        // Find course ID by name
        int idCourse = this.courseRepository.findCourseIdE
```




```
2025-07-24T19:32:03.155+02:00  INFO 20432 --- [demo] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2025-07-24T19:32:11.147+02:00  INFO 20432 --- [demo] [nio-8080-exec-2] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2025-07-24T19:32:12.113+02:00  INFO 20432 --- [demo] [nio-8080-exec-2] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.jdbc.JDBC4Connection@7e6cf045
2025-07-24T19:32:12.117+02:00  INFO 20432 --- [demo] [nio-8080-exec-2] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
[AssignProfessorCourseController] Professor (id=1) assigned to course (id=2): true
```

Flujo MVC
Envío de datos desde el controlador a la vista

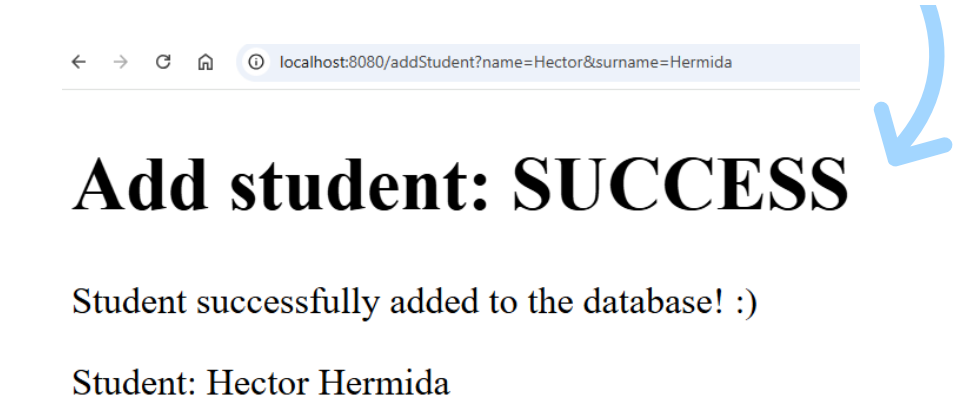
- En lugar de mostrar la salida por consola, los controladores deben dirigir el flujo de navegación a la vista correspondiente tras terminar su función:
  a) Redirigir a una vista general o de menú, sin dar información adicional
  b) Redirigir a una vista específica en base al resultado de la operación. Por ejemplo:
     i. Si el usuario se equivocó introduciendo datos, podemos volver a la vista con el formulario donde los rellenó e informar del error
     ii. Si la inserción fue correcta, podemos mostrar una vista con la información añadida y un mensaje de éxito
- En cuanto a los datos:
  - Si se han leído datos de la base de datos, tendrá que poner el objeto de domino a disposición de la vista
  - Dependiendo del alcance que queramos que tenga, será un @ModelAttribute (enlazado a la petición actual) o un @SessionAttribute (persiste en sucesivas peticiones)

Flujo MVC
Envío de datos desde el controlador a la vista

- En este ejemplo, al acceder desde el navegador a */addStudent*:

1. El controlador `AddStudentController` recoge la petición GET (**@GetMapping**)
2. Crea un nuevo estudiante a partir de los datos recibidos en la URL
3. El repositorio lo añade a la base de datos
4. Según el resultado de la inserción:
   a. Si ha sido correcta, se **redirige** a la vista *AddStudentViewSucess.html* y se muestra el nombre y apellidos
   b. Si no se ha podido insertar, se **redirige** a la vista *AddStudentViewFail.html*

```java
@GetMapping("/addStudent")
public String addStudent(@RequestParam("name") String name, @RequestParam("surname") String surname) {
    LocalDate date = LocalDate.of(year:2001, month:8, dayOfMonth:8);
    Student student = new Student(id:8, name, surname,
                                  date, StudentType.FULL_TIME);
    boolean success = studentRepository.addStudent(student);
    String nextPage;
    if(success){
        nextPage = "addStudentViewSucess.html";
    }
    else
        nextPage = "addStudentViewFail.html";
    return nextPage;
}
```



Flujo MVC

Envío de datos desde el controlador a la vista

- En este ejemplo, los datos del estudiante se extraen de la base de datos y se ponen a disposición de la vista, como objeto, a través del **modelo** con la clase `ModelAndView`
  - Recibe el nombre de la vista en el constructor: Debe estar alojada en "*/templates*"
  - El método `addObject(String name, Object o)` vincula el objeto a un nombre para que luego sea accesible desde la vista

Controlador

```java
@GetMapping("/findStudentById")
public ModelAndView findStudentById(){
    Random randomGenerator = new Random();
    int id = randomGenerator.nextInt(origin:1, bound:7);
    Student student = studentRepository.findStudentById(id);
    ModelAndView model = new ModelAndView("findStudentById.html");
    model.addObject("student", student);
    return model;
}
```

Vista (*Template*)

src > main > resources > templates > findStudentById.html > ...

```html
<!DOCTYPE html>
<html   xmlns="http://www.w3.org/1999/xhtml"
        xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>My first application</title>
        <body>
            <h1>Find Student by ID</h1>
            <p th:text="${student.id}">-1</p>
            <p th:text="${student.name}">NULL</p>
            <p th:text="${student.surname}">NULL</p>
            <p th:text="${student.birthDate}">NULL</p>
            <p th:text="${student.type}">NULL</p>
        </body>
    </head>
</html>
```

Vista (Navegador)

localhost:8080/findStudentById

Find Student by ID

5

Eric

Elliott

2002-05-25

ERASMUS

! Por simplicidad, se busca un ID válido generado de forma aleatoria

Flujo MVC
Envío de datos desde la vista al controlador

- Las peticiones HTTP GET **NO** deben usarse para realizar escrituras, pues son peticiones que no deben alterar el estado del servidor
- Las peticiones **HTTP POST** son las que habitualmente se vinculan a formularios HTML para el envío de información desde vistas a controladores
- Se utiliza el objeto `ModelAndView`, pero siguiendo un proceso distinto:
    1. El controlador crea el objeto que vayamos a rellenar con el formulario (p. ej. Clase de dominio `Profesor`)
    2. El controlador guarda el objeto de dominio en el objeto `ModelAndView`, asignándole un nombre identificativo, antes de redireccionar a la vista con el formulario
    3. El formulario invocará al controlador al pulsar el botón de envío del formulario, rellenando automáticamente el objeto con los campos del formulario
    4. El método del controlador vinculado al método HTTP POST (@PostMapping) será el encargado de recoger el objeto y procesarlo (p. ej. Enviarlo al repositorio)

Flujo MVC

Envío de datos desde la vista al controlador

Pasos 1-2: Método @GetMapping en el controlador

- La clase Java debe tener definido un constructor vacío y propiedades equivalentes a los campos del formulario
- No tiene por qué ser una clase de dominio completa, puede que necesitemos una versión reducida con solo algunos campos, aquellos que pide el formulario al usuario
- El controlador debe redireccionar a la vista que contiene el formulario (en */templates*)

Controlador

```
@GetMapping("/addProfessor")
public ModelAndView getAddProfessorView() {
    this.modelAndView.setViewName(viewName:"addProfessorView.html");
    this.modelAndView.addObject(attributeName:"newProfessor", new Professor());
    return modelAndView;
}
```

```
public class Professor {

    private int id;
    private String name;
    private String surname;
    private String department;

    public Professor(){
    }
```

Flujo MVC

Envío de datos desde la vista al controlador

Pasos 3-4: Método @PostMapping en el controlador

- El formulario hace referencia al objeto a través del nombre asignado en `ModelAndView`
- Los campos del formulario deben hacerse corresponder con las propiedades del objeto
- El controlador recibe el objeto relleno como parámetro: **@ModelAttribute**
- Si no se necesita más el objeto, es recomendable eliminarlo: `sessionStatus.setComplete();`

Vista

src > main > resources > templates > addProfessorView.html > ...

```
<!DOCTYPE html>
<html   xmlns="http://www.w3.org/1999/xhtml"
        xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>My first application</title>
        <body>
            <h1>Add a professor</h1>
            <form method="POST" th:action="@{/addProfessor}" th:object="${newProfessor}">
                <label for="id">ID:</label><br>
                <input type="text" th:field="*{id}"><br>
                <label for="name">Name:</label><br>
                <input type="text" th:field="*{name}"><br>
                <label for="surname">Surname:</label><br>
                <input type="text" th:field="*{surname}"><br>
                <label for="department">Department:</label><br>
                <input type="text" th:field="*{department}"><br>
                <input type="submit" value="Submit">
            </form>
        </body>
    </head>
</html>
```

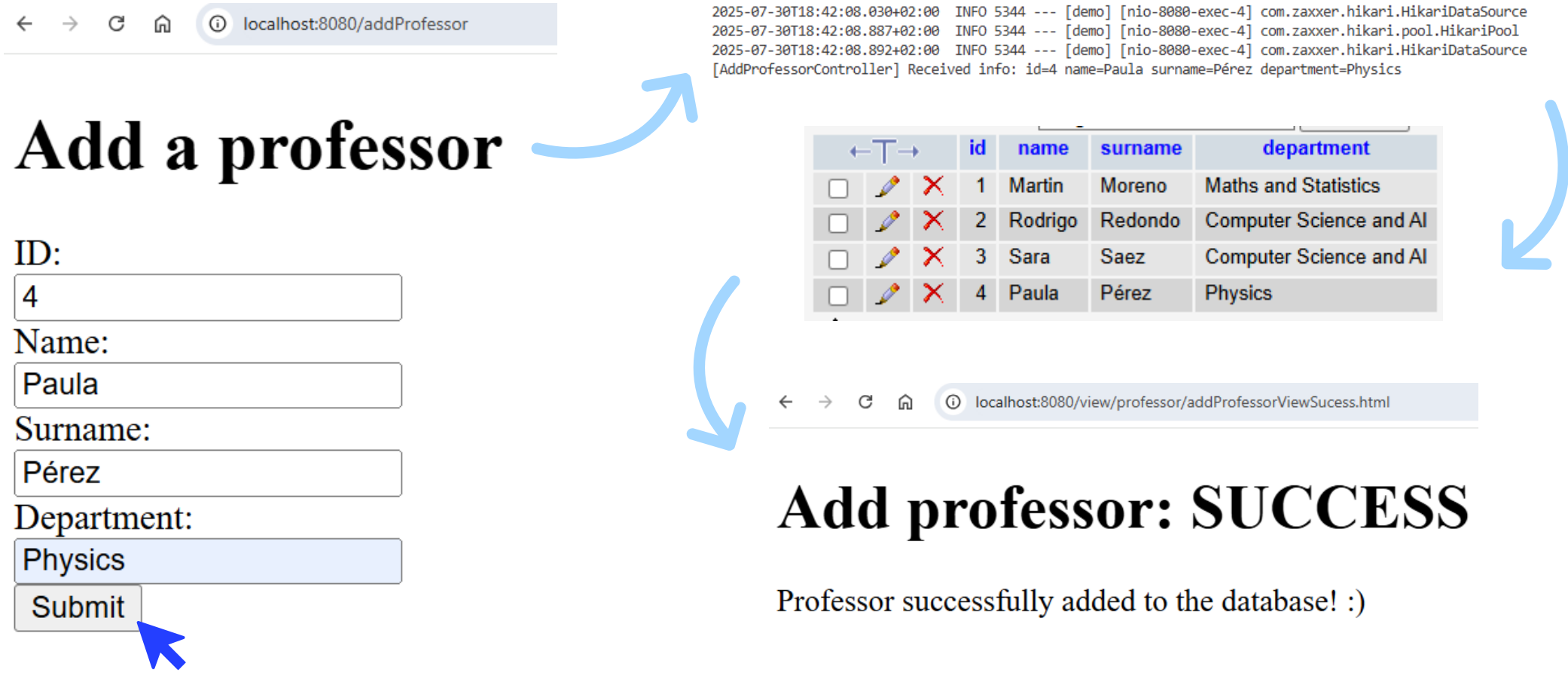
Controlador

```
@PostMapping("/addProfessor")
public String addProfessor(@ModelAttribute Professor newProfessor, SessionStatus sessionStatus) {

    System.out.println("[AddProfessorController] Received info: id=" + newProfessor.getId() +
                       " name=" + newProfessor.getName() +
                       " surname=" + newProfessor.getSurname() +
                       " department=" + newProfessor.getDepartment());
    boolean success = professorRepository.addProfessor(newProfessor);
    String nextPage;

    if(success){
        nextPage = "addProfessorViewSucess.html";
    }
    else
        nextPage = "addProfessorViewFail.html";

    sessionStatus.setComplete();
    return new String("redirect:/view/professor/" + nextPage);
}
```

Flujo MVC

Envío de datos desde la vista al controlador

- Flujo **MVC** completo



Objetivos de la semana

1. Replica el ejemplo explicado en clase, ejecutándolo sobre tu base de datos
   - Versión completa disponible en el repositorio Github: https://github.com/aramirez-uco/pw-examples
2. Incorpora nuevos controladores para procesar consultas complejas del enunciado de prácticas
   - Puede ser necesario definir nuevos objetos de dominio y repositorios
3. Comienza a definir las vistas para enviar y recibir datos
   - Estudia qué vistas serán estáticas y cuáles serán dinámicas
   - No te preocupes excesivamente por la estructura y formato (próxima semana)
4. Consulta la bibliografía recomendada:
   - Capítulo 2: "*Developing web applications*" del libro "*Spring in Action*" (6th ed.)
   - *Spring MVC and Thymeleaf: how to access data from templates:* https://www.thymeleaf.org/doc/articles/springmvcaccessdata.html