ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

UNIVERSIDAD D CÓRDOBA

Programación web

Prácticas

Semana 10: Diseño e implementación de API (II)

Aurora Ramírez Quesada (aramirez@uco.es)

Departamento de Ciencia de la Computación e Inteligencia Artificial

Universidad de Córdoba

Índice de contenido

API REST en Spring
Introducción

- Con API REST podemos dar soporte a operaciones de actualización y borrado asociándolas de forma explícita a los métodos **HTTP**: PUT, PATCH, DELETE
- Si estas operaciones no estaban soportadas en los repositorios, necesitaremos definirlas para que puedan ser invocadas desde los controladores REST
  - Debemos decidir si un recurso es modificable de forma completa (reemplazo o no) y qué sucede con sus relaciones
  - Debemos decidir si un recurso puede modificarse parcialmente y, en tal caso, qué campos concretos se pueden actualizar (por ejemplo, no se suele permitir actualizar el ID)
  - Podemos decidir si un recurso concreto puede eliminarse o no, y si el borrado es “físico” o “lógico” (desactivación)
- Con estas nuevas operaciones, se completa el conjunto de operaciones **CRUD** a nivel de base de datos

API REST en Spring
Peticiones PUT

- El objetivo de una petición **PUT** es realizar una operación de reemplazo sobre un recurso
- Se espera que la petición del cliente contenga toda la información sobre el recurso
  - Si algún campo no está presente, la propiedad se borraría o sobrescribiría a `null`
- Según el estándar HTTP, las peticiones PUT **no** devuelven cuerpo en la respuesta al cliente

PUT

```
@PutMapping(path="/{id}", consumes="application/json")
public void putResource(@PathVariable int id, @RequestBody object){
        // Asignar el ID al nuevo recurso
        object.setID(id);

        // Procesar petición
        T updatedObject = repository.save(object);

}
```

API REST en Spring
Peticiones PUT

- **Ejemplo:** Reemplazo de un curso por otro
  1. El id del curso se toma del *path*, mientras que los datos del curso están en la petición
  2. Se comprueba si el curso existe
  3. Se reemplaza el curso por el nuevo objeto, manteniendo el ID

```java
@PutMapping(path="/{id}", consumes="application/json")
@ResponseStatus(HttpStatus.OK)
public void putCourse(@PathVariable int id, @RequestBody Course requestCourse) {
    try{
        // Get course by id
        Course currentCourse = this.courseRepository.findCourseById(id);
        if(currentCourse != null){
            requestCourse.setId(currentCourse.getId());
            boolean resultOk = courseRepository.updateCourse(requestCourse);
```

API REST en Spring
Peticiones PUT

Se recomienda utilizar clases ***Wrapper*** (`Integer` en lugar de `int`) para manejar bien los valores nulos

- Para soportar la operación a nivel de la API, es necesario:
  1. Definir nuevas operaciones en el repositorio: buscar curso por ID, actualizar curso completo, etc.
  2. Definir un DTO que permita la transferencia de la información del curso en la petición y hacia el repositorio

```java
public Course findCourseById(int courseId){
    try{
        String query = sqlQueries.getProperty(key: "select-findCourseById");
        Course course = jdbcTemplate.query(query, this::mapRowToCourse, courseId);
        return course;
    }catch(DataAccessException queryException){
        queryException.printStackTrace();
        return null;
    }
}
```

```java
public boolean updateCourse(Course course){
    try{
        String query = sqlQueries.getProperty(key: "update-fullCourseById");
        if(query != null){
            int result = jdbcTemplate.update(query, course.getName(), course.getDegree(),
                                             course.getYear(), course.getIdProfessor(), course.getId());
            if (result>0)
                return true;
            else
                return false;
        }
        else
            return false;
        } catch(DataAccessException exception){
            System.err.println(x: "Unable to update course in the database");
            exception.printStackTrace();
        return false;
    }
}
```

```java
public class Course {
    private Integer id;
    private String name;
    private String degree;
    private Integer year;
    private Integer idProfessor;

    public Course(){
        this.id = null;
        this.name = null;
        this.degree = null;
        this.year = null;
        this.idProfessor = null;
    }
```

API REST en Spring
Peticiones PATCH

- El objetivo de una petición **PATCH** es realizar actualizaciones parciales sobre un recurso
- Se espera que la petición del cliente contenga algunos de los campos del recurso
  - Se recupera el recurso original, y se modifican los campos que aparezcan en la petición (no `null`)

PATCH

```
@PatchMapping(path="/{id}", consumes="application/json")
public T patchResource(@PathVariable int id, @RequestBody object){
        // Recuperar el objeto original
        T currentObject = repository.getResourceById(id);

        // Actualizar campos requeridos
        if(object.getField()!=null)
            currentObject.setField(object.getField());
        …

        // Guardar el recurso actualizado
        repository.save(currentObject);

        // Devolver recurso actualizado
        return currentObject;
}
```

API REST en Spring
Peticiones PATCH

Se asume que no se permite actualizar a un valor nulo (p. ej. Eliminar valor de clave foránea)

- **Ejemplo:** Actualización de las propiedades de un curso
  1. El id del curso se toma del *path*, mientras que los datos del curso están en la petición
  2. Se comprueba si el curso existe
  3. Se actualizan solo los campos que se han recibido en la petición (distintos de `null`)
  4. Si la actualización en base de datos es correcta, se devuelve el objeto actualizado

- Alternativamente, el repositorio podría ofrecer operaciones para actualizar cada campo y así realizar comprobaciones más precisas, pero implica un mayor número de conexiones (lento)

```java
@PatchMapping(path="/{id}", consumes="application/json")
public Course patchCourse(@PathVariable int id, @RequestBody Course requestCourse) {
    Course response = requestCourse;
    try{
        // Get course by id
        Course currentCourse = this.courseRepository.findCourseById(id);
        if(currentCourse != null){

            // Update properties
            requestCourse.setId(currentCourse.getId());

            if(requestCourse.getName() != null){
                currentCourse.setName(requestCourse.getName());
            }

            if(requestCourse.getDegree() != null){
                currentCourse.setDegree(requestCourse.getDegree());
            }

            if(requestCourse.getYear() != null){
                currentCourse.setYear(requestCourse.getYear());
            }

            if(requestCourse.getIdProfessor() != null){
                currentCourse.setIdProfessor(requestCourse.getIdProfessor());
            }

            // Save updated resource
            boolean resultOk = courseRepository.updateCourse(currentCourse);
            if(resultOk){
                response = currentCourse;
            }
        }
    }catch(Exception e){
        return requestCourse;
    }
    return response;
}
```

API REST en Spring
Peticiones DELETE

- El objetivo de una petición **DELETE** es eliminar uno o varios recursos
- Para eliminar el recurso completo, la petición debe ir dirigida a: /resource
- Para eliminar un recurso concreto, la petición debe ir dirigida a: /resource/{id}
- El código de estado asociado a peticiones DELETE es: NO_CONTENT

DELETE

```
@DeleteMapping
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deleteResource(){
  // Borrar recurso completo
  repository.delete();
}
```

```
@DeleteMapping(path="/{id}")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deleteResourceById(@PathVariable int id){
  // Borrar recurso
  if(repository.existsResource(id)){
    repository.deleteById(id);
  }
}
```

API REST en Spring
Peticiones DELETE

- **Caso 1:** Borrado del recurso completo
  1. Se invoca a la operación correspondiente en el repositorio (nueva)
  2. Se devuelve NO_CONTENT

```java
@DeleteMapping()
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deleteAllCourses(){
    this.courseRepository.deleteAllCourses();
}
```

Ejemplo:
localhost:8080/api/courses

```java
public boolean deleteAllCourses(){
    try{
        String query = sqlQueries.getProperty(key: "delete-deleteAllCourses");
        if(query != null){
            int result = jdbcTemplate.update(query);
            if (result>0)
                return true;
            else
                return false;
        }
        else
            return false;
    }catch(DataAccessException exception){
        return false;
    }
}
```

API REST en Spring
Peticiones DELETE

- **Caso 2:** Borrado de un recurso concreto
  1. El ID del recurso a borrar forma parte del *path*
  2. Se comprueba la existencia del recurso (opcional)
  3. Se invoca a la operación de borrado parametrizada en el repositorio (nueva)
  4. Se devuelve NO_CONTENT

```java
@DeleteMapping("/{id}")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deleteCourse(@PathVariable int id){
    Course course = this.courseRepository.findCourseById(id);
    if(course != null)
        this.courseRepository.deleteCourse(id);
}
```

Ejemplo:
localhost:8080/api/courses/1

```java
public boolean deleteCourse(int courseId){
    try{
        String query = sqlQueries.getProperty(key: "delete-deleteCourseById");
        if(query != null){
            int result = jdbcTemplate.update(query, courseId);
            if (result>0)
                return true;
            else
                return false;
        }
        else
            return false;
    }catch(DataAccessException exception){
        return false;
    }
}
```

Código cliente
Peticiones PUT

- Enviar peticiones **PUT**
  - Un único método para el envío de peticiones PUT: `put("url", JavaClass, uriVariables)`
  - Si hay un error, se lanza `RestClientException` (recurso no encontrado, servidor no disponible)

```java
private static void sendPutRequests(){

    RestTemplate rest = new RestTemplate();
    String baseURL = "http://localhost:8080/api";

    // Request to put a course (without response)
    Course newCourse = new Course(name: "Introduction to Maths", degree: "Data Science", year: 3, idProfessor: 1);
    int id = 2;
    try{
        System.out.println(x: "==== REQUEST 1: PUT (no response) ====");
        rest.put(baseURL + "/courses/{id}", newCourse, id);
        System.out.println(x: "Update correct.");
    }catch(RestClientException exception){
        System.out.println(exception);
    }
}
```

PUT localhost:8080/api/courses/2

←T→	id	name	degree	year	idProfessor
☐ ✎ ✕	1	Programming	Computer Science	1	*NULL*
☐ ✎ ✕	2	Introduction to Maths	Data Science	3	1
☐ ✎ ✕	3	Software Engineering	Computer Science	2	*NULL*
☐ ✎ ✕	4	Web Programming	Computer Science	3	*NULL*

```
==== REQUEST 1: PUT (no response) ====
Update correct.
```

Código cliente
Peticiones PATCH

- Enviar peticiones **PATCH**
  - `RestTemplate` proporciona el método: `patchForObject("url", JavaClass, uriVariables)`
  - Configurar el soporte a peticiones PATCH con HttpComponentsClientHttpRequestFactory

```
private static void sendPatchRequests(){
    RestTemplate rest = new RestTemplate();
    rest.setRequestFactory(new HttpComponentsClientHttpRequestFactory());
    String baseURL = "http://localhost:8080/api";
    int id;

    // Example 1: Request to update one field
    try{
        System.out.println(x: "==== REQUEST 3: PATCH (valid) ====");
        id = 1;
        Course course1 = new Course();
        course1.setYear(year: 2);
        Course response1 = rest.patchForObject(baseURL + "/courses/{id}", course1, responseType: Course.class, id);
        System.out.println(response1.toString());
    }catch(RestClientException exception){
        System.out.println(exception);
    }
```

PATCH localhost:8080/api/courses/2

id	name	degree	year	idProfessor
1	Programming	Computer Science	2	*NULL*

```
==== REQUEST 3: PATCH (valid) ====
        ID: 1
        Name: Programming
        Degree: Computer Science
        Year: 2
        Professor ID: null
```

Código cliente
Peticiones PATCH

- Enviar peticiones **PATCH**
  - `RestTemplate` proporciona el método: `patchForObject("url", JavaClass, uriVariables)`
  - Configurar el soporte a peticiones PATCH con HttpComponentsClientHttpRequestFactory
  - En este caso, se actualizan varios campos simultáneamente

PATCH localhost:8080/api/courses/2

```
// Example 2: Request to update some fields
try{
    System.out.println(x: "==== REQUEST 4: PATCH (valid) ====");
    id = 3;
    Course course2 = new Course();
    course2.setName(name: "Introduction to Software Engineering");
    course2.setIdProfessor(idProfessor: 3);
    Course response2 = rest.patchForObject(baseURL + "/courses/{id}", course2, responseType: Course.class, id);
    System.out.println(response2.toString());
}catch(RestClientException exception){
    System.out.println(exception);
}
```

id	name	degree	year	idProfessor
1	Programming	Computer Science	2	*NULL*
2	Introduction to Maths	Data Science	3	1
3	Introduction to Software Engineering	Computer Science	2	3

```
==== REQUEST 4: PATCH (valid) ====
        ID: 3
        Name: Introduction to Software Engineering
        Degree: Computer Science
        Year: 2
        Professor ID: 3
```

Código cliente
Peticiones PATCH

- Enviar peticiones **PATCH**
  - `RestTemplate` proporciona el método: `patchForObject("url", JavaClass, uriVariables)`
  - Necesidad de configurar el uso de peticiones PATCH con HttpComponentsClientHttpRequestFactory
  - En este caso, la actualización no es posible por el valor incorrecto del id de profesor (clave foránea)

```
// Example 3: Invalid request to update the professor (id does not exist)
try{
    id = 4;
    Course course3 = new Course();
    course3.setIdProfessor(idProfessor: 6);
    System.out.println(x: "==== REQUEST 5: PATCH (invalid) ====");
    Course response3 = rest.patchForObject(baseURL + "/courses/{id}", course3, responseType: Course.class, id);
    System.out.println(response3.toString());
}catch(RestClientException exception){
    System.out.println(exception);
}
```

PATCH localhost:8080/api/courses/2

4	Web Programming	Computer Science	3	*NULL*

```
==== REQUEST 5: PATCH (invalid) ====
        ID: 4
        Name: null
        Degree: null
        Year: null
        Professor ID: 6
```

Código cliente
Peticiones DELETE

- Enviar peticiones **DELETE**
  - `RestTemplate` proporciona el método: `delete("url", uriVariables)`
  - Si ocurre algún error, lanzará la excepción `RestClientException`
  - **Ejemplo 1:** Borrado de un recurso que existe

DELETE localhost:8080/api/courses/1

```
static void sendDeleteRequests(){
    RestTemplate rest = new RestTemplate();
    String baseURL = "http://localhost:8080/api";

    // Example 1: Delete one course
    try{
        System.out.println(x: "==== REQUEST 6: DELETE ONE COURSE (valid) ====");
        rest.delete(baseURL + "/courses/{id}", ...uriVariables: 1);
    }catch(RestClientException exception){
        System.out.println(exception);
    }
```


←T→	id	name	degree	year	idProfessor
☐ ✎ ✕	2	Introduction to Maths	Data Science	3	1
☐ ✎ ✕	3	Introduction to Software Engineering	Computer Science	2	3
☐ ✎ ✕	4	Web Programming	Computer Science	3	*NULL*

Código cliente
Peticiones DELETE

- Enviar peticiones **DELETE**
  - `RestTemplate` proporciona el método: `delete("url", uriVariables)`
  - Si ocurre algún error, lanzará la excepción `RestClientException`
  - **Ejemplo 2:** Borrado de un recurso que no existe

DELETE localhost:8080/api/courses/8

```
// Example 2: Delete a course that does not exist
try{
    System.out.println(x: "==== REQUEST 7: DELETE ONE COURSE (no effect) ====");
    rest.delete(baseURL + "/courses/{id}", ...uriVariables: 8);
}catch(RestClientException exception){
    System.out.println(exception);
}
```


←T→	id	name	degree	year	idProfessor
☐ ✎ ✕	2	Introduction to Maths	Data Science	3	1
☐ ✎ ✕	3	Introduction to Software Engineering	Computer Science	2	3
☐ ✎ ✕	4	Web Programming	Computer Science	3	*NULL*

Código cliente
Peticiones DELETE

- Enviar peticiones **DELETE**
  - `RestTemplate` proporciona el método: `delete("url", uriVariables)`
  - Si ocurre algún error, lanzará la excepción `RestClientException`
  - **Ejemplo 3:** Borrado del recurso completo

DELETE localhost:8080/api/courses

```
// Example 3: Delete all courses
try{
    System.out.println(x: "==== REQUEST 8: DELETE ALL COURSES (valid) ====");
    rest.delete(baseURL + "/courses");
}catch(RestClientException exception){
    System.out.println(exception);
}
```


MySQL ha devuelto un valor vacío (i.e., cero columnas). (La consulta tardó 0.0001 seg)

consulta SQL:

```
SELECT *
FROM `Course`
LIMIT 0 , 30
```

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. Decide qué recursos deben soportar las peticiones PUT, PATCH y DELETE según el enunciado de la práctica 2
3. Continua la implementación de la API
   - Añade los métodos para soportar peticiones PUT, PATCH y DELETE en controladores (y repositorios)
   - Define los códigos HTTP apropiados según el tipo de operación
   - Implementa código cliente para probar la API
4. Consulta la bibliografía recomendada:
   - Secciones 7.1 y 7.3 del capítulo 7: "*Creating REST services*" del libro "*Spring in Action*" (6th ed.)
   - Tutorial Spring REST (avanzado): https://spring.io/guides/tutorials/rest