xideral®

WEEK 5

FINAL PROJECT

LUIS MIGUEL SÁNCHEZ FLORES

Java Backend Academy – Monterrey, N.L.

September 17th, 2024

Task Application

Fitted for the tools / concepts learned throughout the academy

CRUD operations for maintaining a list of relevant tasks

Subprojects showcasing a MVP of the task application

Spring Web

Spring Data JPA

MVC Application

Thymeleaf

JUnit + Mockito

Model Component

Task Entity for JPA persistence

Spring Data JPA annotations
(@Entity, @Table, @Id)

Leverage **Lombok** library
to reduce boilerplate code

```java
import jakarta.persistence.*;

@Entity // Specify that the class is an entity
@Table(name = "task") // Sets the database table to which a

// LOMBOK ANNOTATIONS
@Data // Generate getters, setters, toString and hashCode m
@AllArgsConstructor // Sets up a constructor that requires
@NoArgsConstructor // Sets up an empty constructor
public class Task {

    // Primary key attribute
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int ID;

    private String title;

    // Specify how the enum value should be persisted (in t
    @Enumerated(EnumType.ORDINAL)
    private Status status;

    public enum Status {
        TODO, DONE
    }
}
```

Controller Component

Central hub that interacts with Model and View

```
19 @Controller
20 @RequestMapping("/list") // Base URL
21 public class TaskController {
22
23     // Inject the Service layer to interact with the
24     @Autowired
25     private TaskService taskService;
26
27
```

```
// Retrieve the list of existing tasks beforehand:
@ModelAttribute
public void addTasksToModel(Model model) {

    List<Task> allTasks = taskService.findAllTasks();

    // Add the existing task to the model:
    model.addAttribute("curList", allTasks);

    // Add a new Task object to the model, as part of
    // the simple creation form included in the page:
    model.addAttribute("task", new Task());
}
```

```
// return the view of list:
@GetMapping
public String getList() {
    return "list";
}

//Get the view for edit a specific task, based on it's id:
@GetMapping("/task/{id}")
public String editTask(@PathVariable("id") int id, Model model) {

    // Get the existing task by its id:
    Task taskToEdit = taskService.getTaskById(id);


    // Add the task to the model:
    model.addAttribute("editTask", taskToEdit);

    return "edit-task";
}
```

View Component

Uses **Thymeleaf** to render dynamic webpages

Offers special attributes and expressions to access the Model's attributes

Enable Thymeleaf:

```
1 <!DOCTYPE html>                                    <!-- Enable Thymeleaf's attributes and tags -->
2 <html xmlns="http:/ /www.w3.org/1999/xhtml"          xmlns:th="http:/ /www.thymeleaf.org">
```

delete-task.html

```
<h2>Delete Task ?</h2>
<form role="form" th:method="delete" th:action="@{/list/task/delete/{id}(id=${delTask.ID})}" th:object="${delTask}"
    <input type="hidden" name="_method" value="delete">
    <h3 th:text="'Are you sure you want to delete the task with ID # ' + ${delTask.ID}">Delete task</h3><br>
    <b>Title: </b><br>
    <p th:text="${delTask.title}"></p><br>
    <b>Status: </b><br>
    <p th:text="${#strings.toUpperCase(delTask.status)}"></p><br>

    <button type=submit>Delete Task</button>
</form>
```

edit-task.html

```
4
5       <!-- For each of the tasks included in the current list ....-->
6       <div th:each="curTask : ${curList}">
7           <!-- Display the task's title and status : -->
8           <b th:text="${curTask.title}">Title</b>
9           <p th:text="${curTask.status}">Status:</p>
0
1           <!-- Create hyperlinks that directs the user to either the editing or delete
2           <a th:href="@{/list/task/{id}(id=${curTask.ID})}">Edit Task</a>
3           <a th:href="@{/list/task/delete/{id}(id=${curTask.ID})}">Delete Task</a>
4       </div>
5
6
7
8       <h4> Add a new Task : </h4>
9
0       <!-- Simple form section that creates a new task for the list: -->
1       <form method="POST" th:object="${task}" th:action="@{/list}">
2           <input th:field="*{title}" type="text" placeholder="Enter task title"/>
3
4           <input type="submit" value="Add Task">
5       </form>
6
```

Interaction with Database

Added Repository and Service layer for task persistence in the Database

```java
import java.util.List;

public interface TaskRepository extends JpaRepository<Task, Integer> {

    // Get list of tasks order by the status:
    @Query("SELECT t FROM Task t ORDER BY t.status ASC")
    List<Task> findAllOrderByStatus();

}
```

```java
import java.util.List;

@Service
public class TaskServiceImpl implements TaskService{

    @Autowired
    private TaskRepository taskRepository;

    @Override
    public List<Task> findAllTasks() {
        List<Task> allTasks = taskRepository.findAllOrderByStatus();
        return allTasks;
    }

    @Override
    @Transactional
    public Task addTask(Task task) {
        Task newTask = taskRepository.save(task);
        return newTask;
    }
```

More on that later....

Adding Security

Added **Spring Security service**

Restricts the Task List to authorized users with the USER role

Login and Denied access page

```
// Set up the data source where the list of users are stored in:
@Bean
public UserDetailsManager userDetailsManager(DataSource dataSource) {
    // Work with a JDBC-based user details manager:
    return new JdbcUserDetailsManager(dataSource);
}
```

Adding Security

Adding restriction to HTTP requests:

```
// Security-related settings to configure to relevant project resources:
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
            // Set up authorization for HTTP requests
            .authorizeHttpRequests(config ->

                    config
                    .requestMatchers("/list", "/list/**").hasRole("USER") // The list
                    .requestMatchers("/").permitAll() // Allow anyone to access the in
                    .anyRequest().authenticated() // Any other requests should be authe
                    ).
```

Setting login page:

```
// Add the login page and enable its access to everyone:
formLogin(login ->

        login
        .permitAll()).

// Add the denied page
exceptionHandling(config ->

        config
        .accessDeniedPage("/restricted-access")

        ).

// Finally, add the logout feature to the application:
logout(logout ->

        logout
        .permitAll()
)

.build();
```

Unit testing

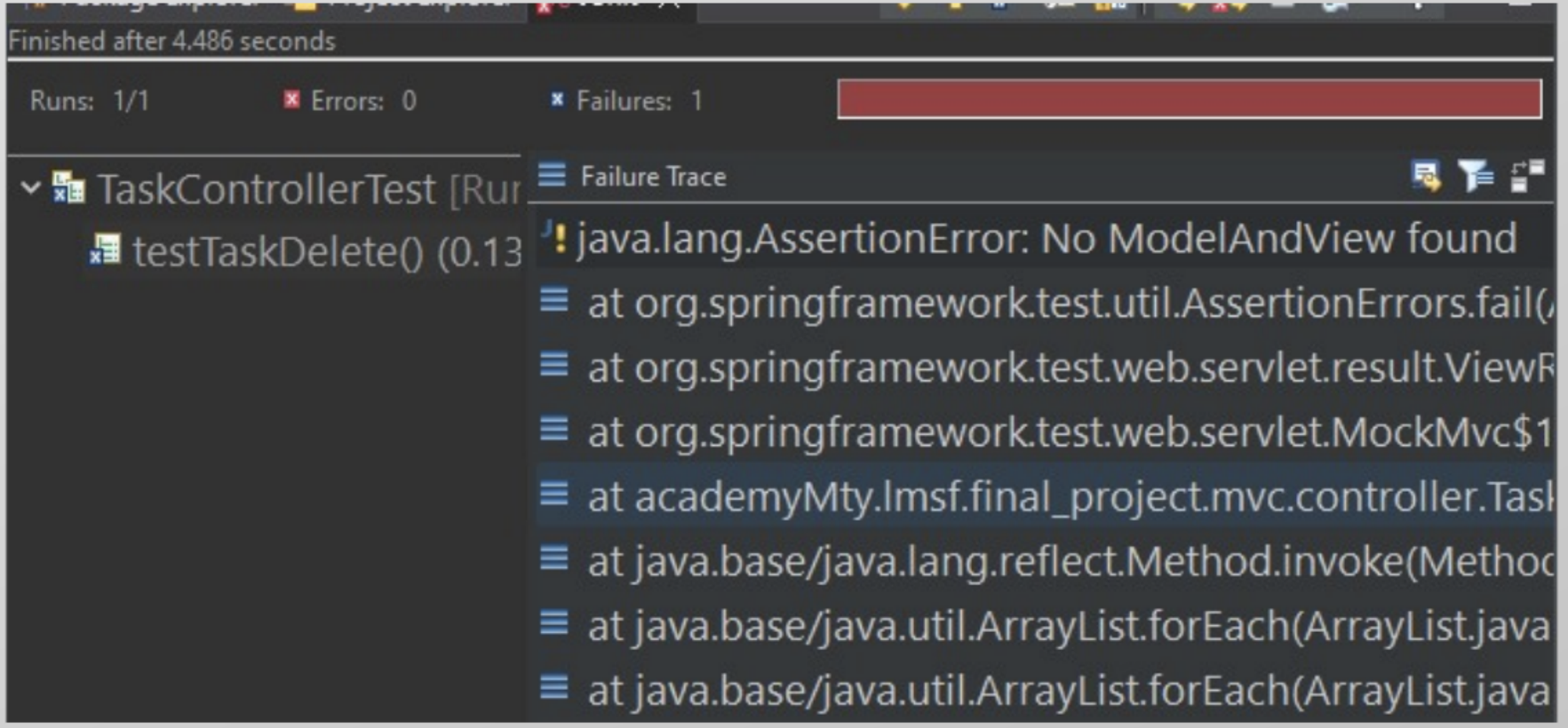
Checking out if the application works as intended!

```
34 @WebMvcTest(TaskController.class)
35 @AutoConfigureMockMvc(addFilters = false)
36 class TaskControllerTest {
37
38     @Autowired
39     private MockMvc mockMvc;
40
41     @MockBean
42     private TaskServiceImpl taskService;
43
44     Task exampleTask;
45
46     @BeforeEach
47     void setUp() throws Exception {
48
49         exampleTask = new Task(1, "Test 1", Task.Status.TODO);
50     }
51
```

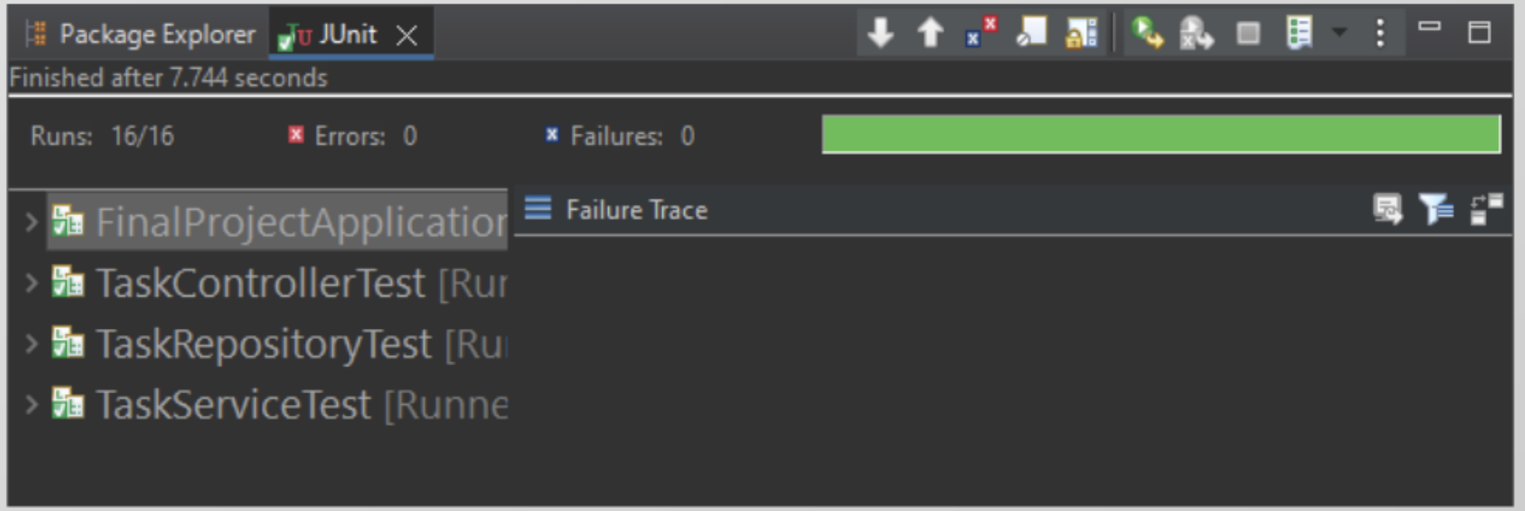
```
// Test out the list view
@Test
void testListView() throws Exception {

    List<Task> exampleTaskList = List.of(

            exampleTask,
            new Task(2, "Test 2", Task.Status.DONE),
            new Task(3, "Test 3", Task.Status.TODO)
            );

    // Mock findAllTasks() by returning the create exampleTaskList:
    when(taskService.findAllTasks()).thenReturn(exampleTaskList);

    mockMvc.perform(get("/list")) // <- perform the GET request at "/lis
    .andExpect(model().attribute("curList", exampleTaskList)) // The mod
    .andExpect(view().name("list")) // Return the "list" view
    .andExpect(status().isOk()); // The status is OK

}
```

Unit testing

Checking out if the application works as intended!



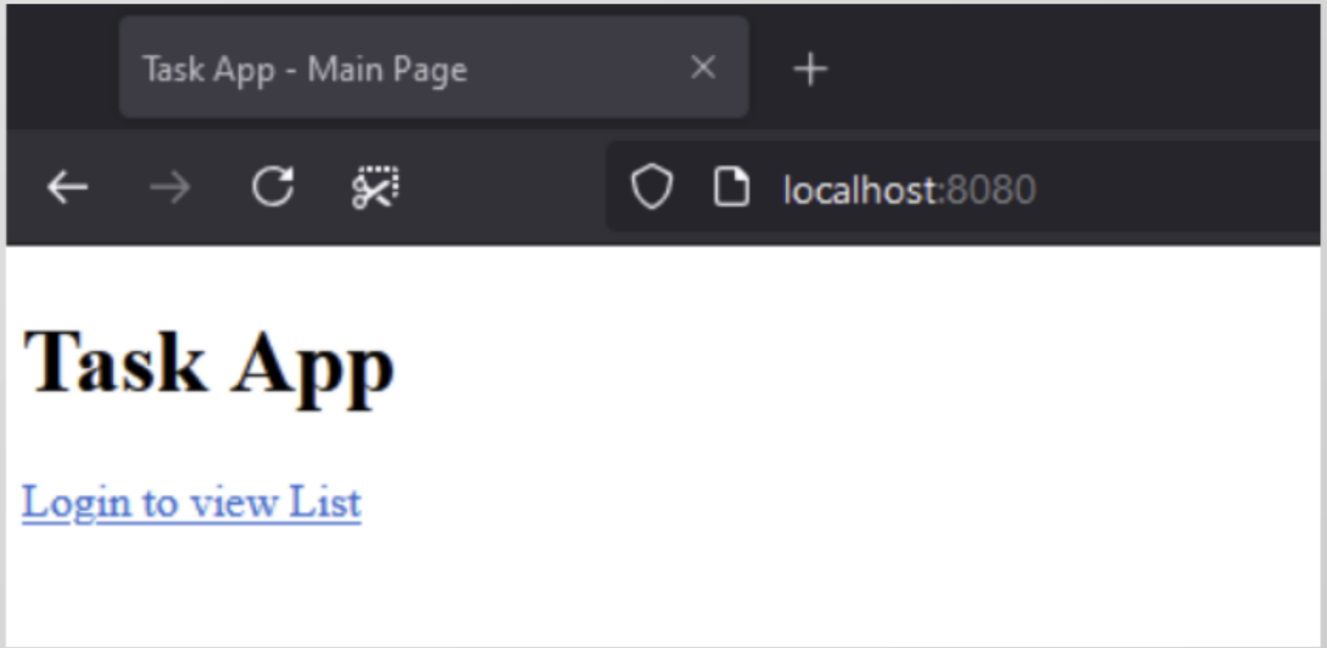
Unit testing



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
TaskMVC	97.9 %	969	21	990
src/main/java	95.0 %	209	11	220
academyMty.lmsf.final_proj	37.5 %	3	5	8
academyMty.lmsf.final_proj	92.9 %	52	4	56
academyMty.lmsf.final_proj	97.3 %	72	2	74
academyMty.lmsf.final_proj	100.0 %	24	0	24
academyMty.lmsf.final_proj	100.0 %	58	0	58
src/test/java	98.7 %	760	10	770

DEMONSTRATION

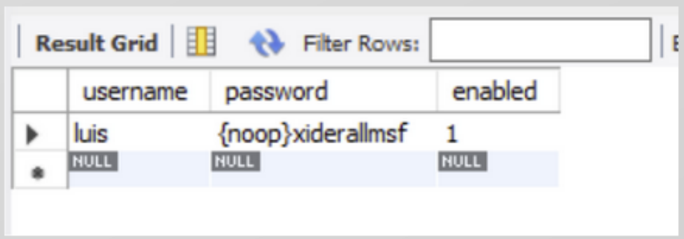
Index.html

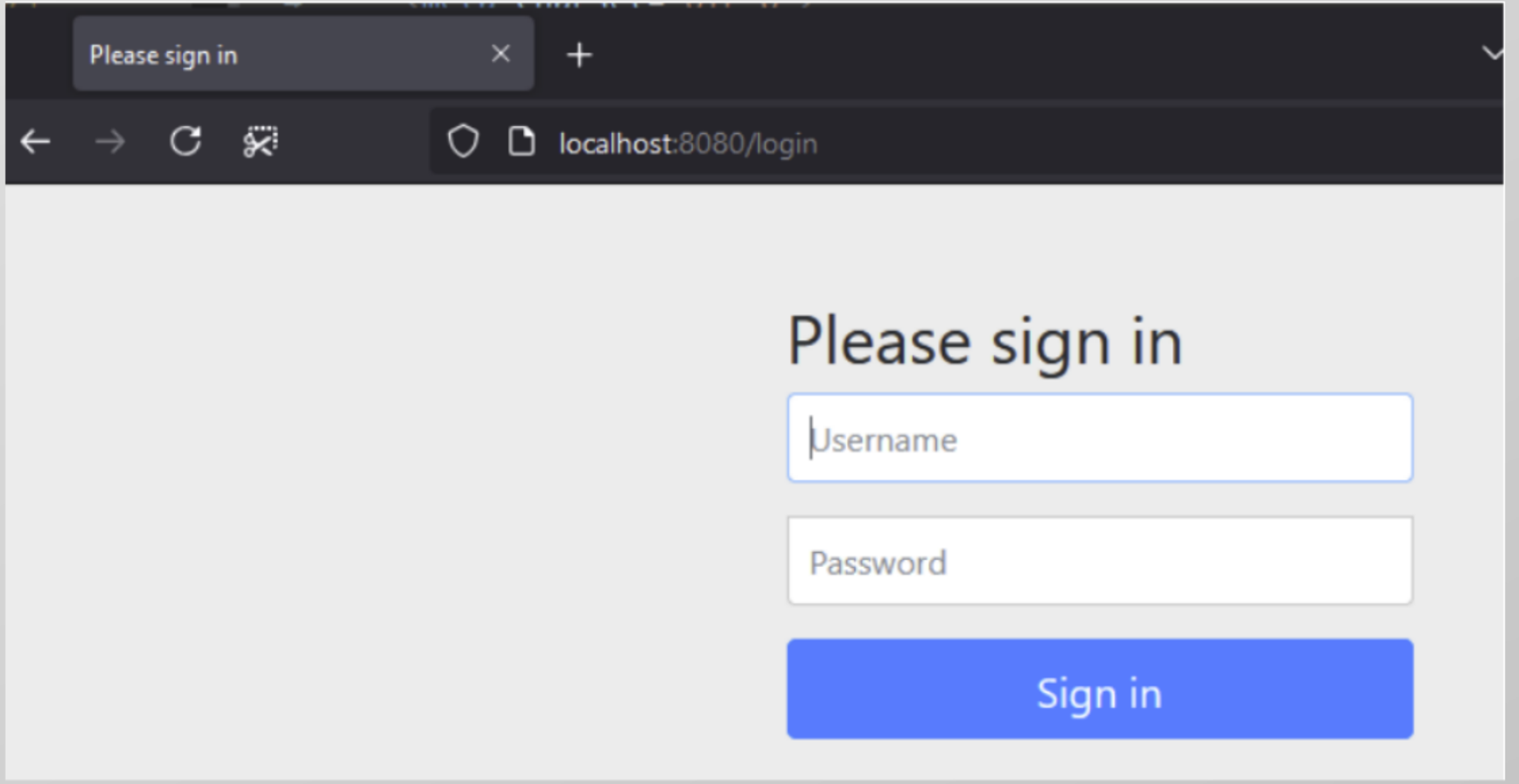


Clicking on the link....

DEMONSTRATION

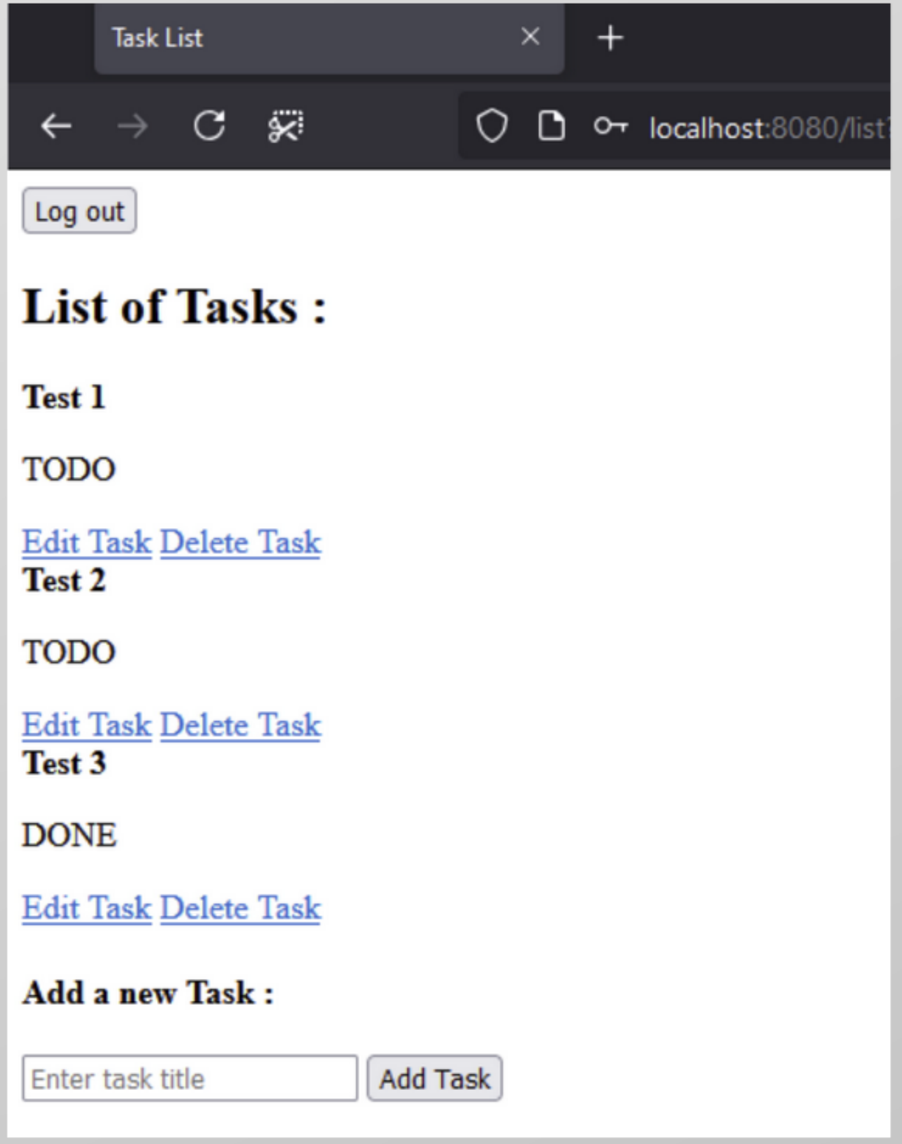
Prompts the login page!





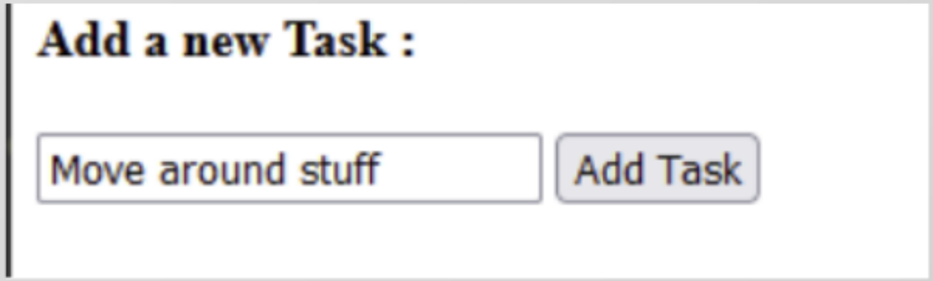
DEMONSTRATION

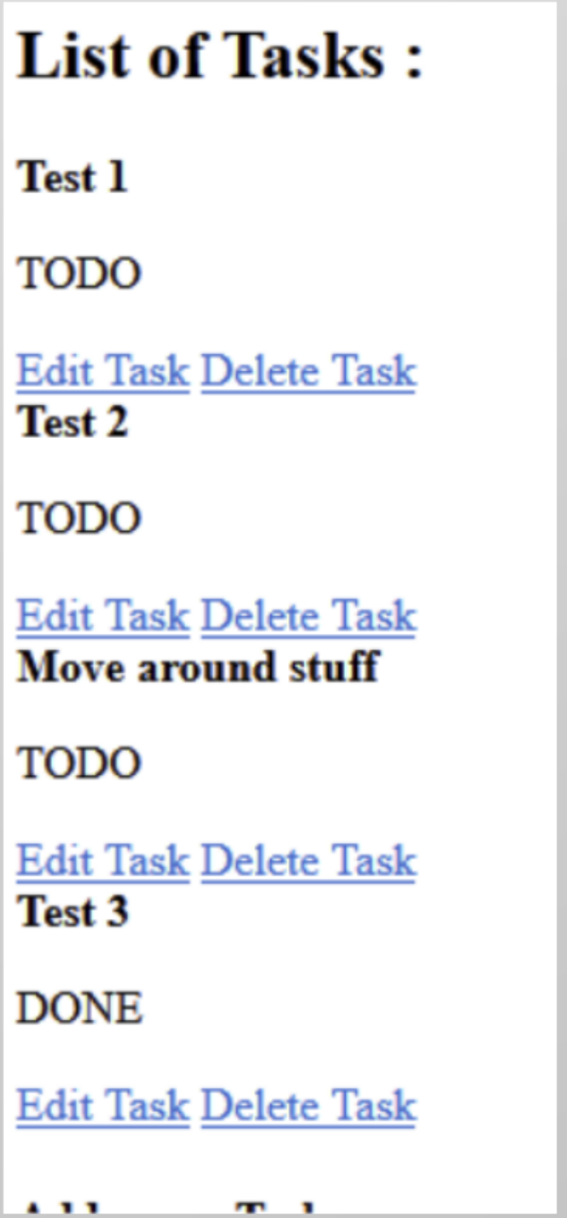
List of Tasks (with initial data)



DEMONSTRATION

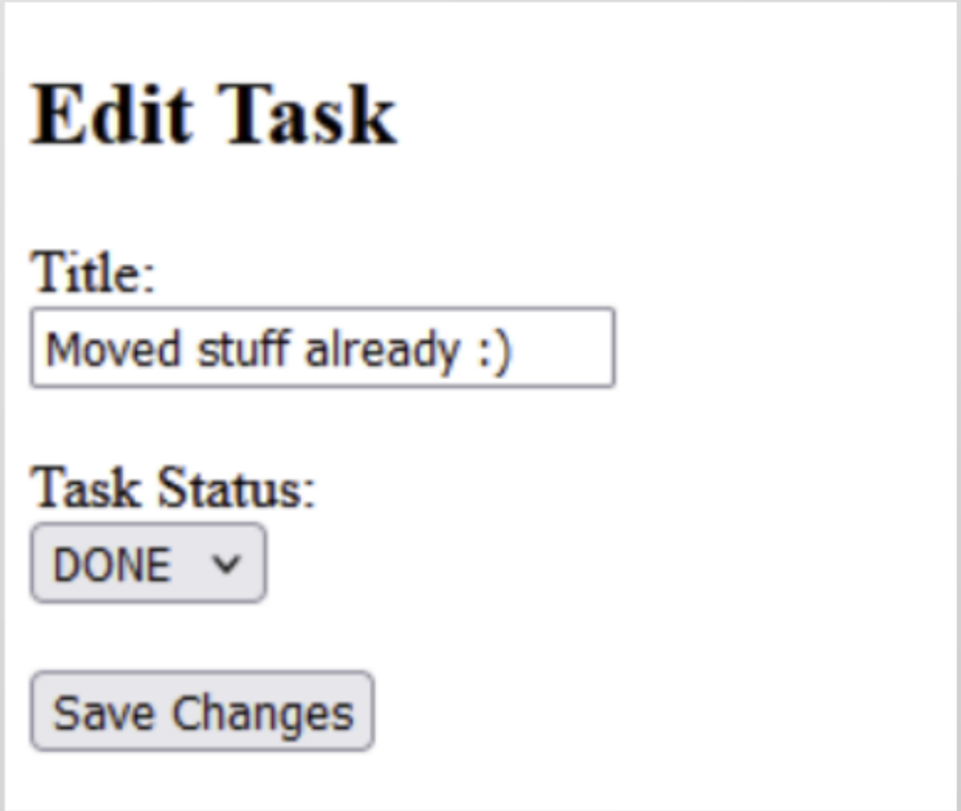
ADDING A NEW TASK

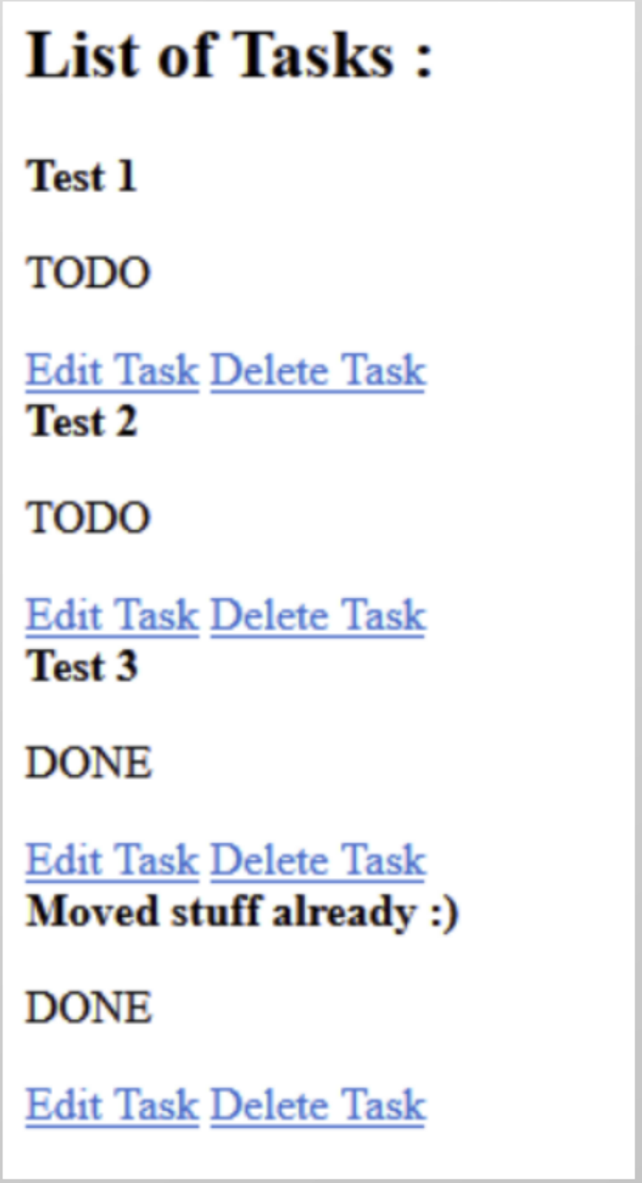




DEMONSTRATION

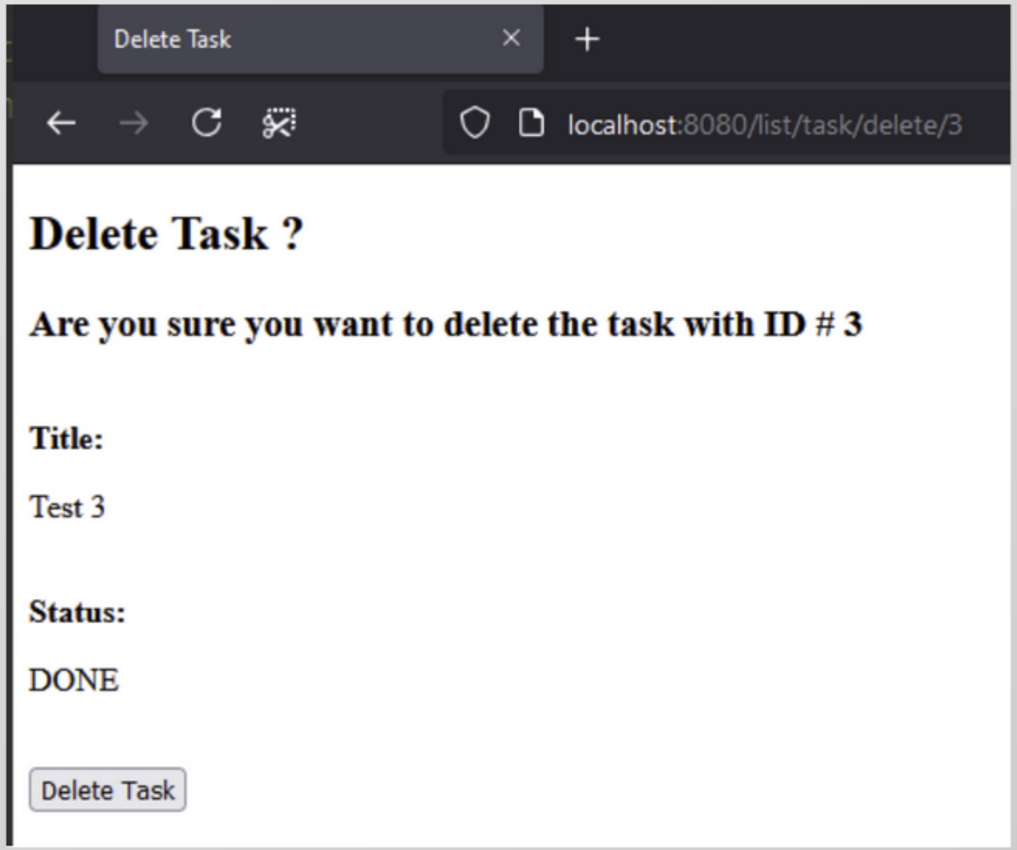
UPDATING EXISTING TASK

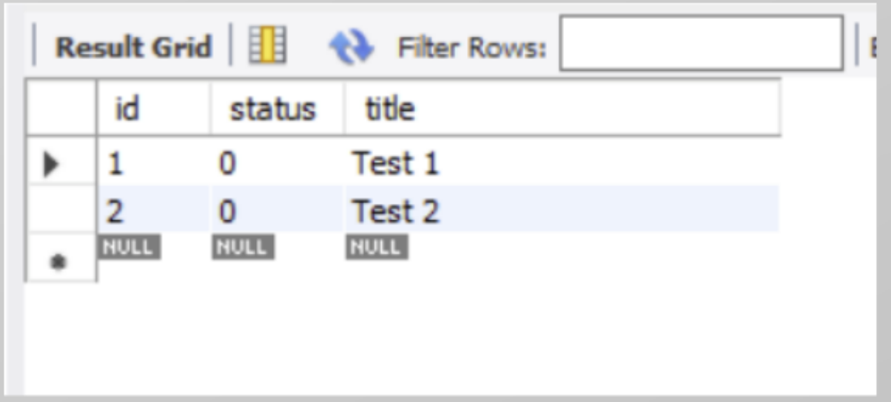




DEMONSTRATION

DELETING TASK





Spring Web

Spring Data JPA

REST Application

Lombok

Entities

Two for this REST project: **USER** and **TASK**

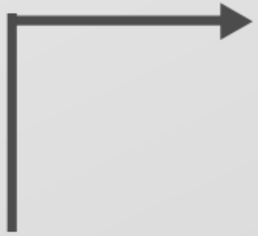
User Entity:

```java
@Entity
@Table(name="users") // <- Table the entity should map to
// LOMBOK ANNOTATIONS
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {

    // UserID that represents the primary key of the table
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long ID;

    @Column(nullable = false, unique = true)
    private String name;

    @Column(nullable = false)
    private String password;

    @JsonIgnore
    @JsonManagedReference
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    @OrderBy("task_id ASC") // Order the list of tasks by their IDs
    private List<Task> tasks = new ArrayList<>();
}
```

One To Many relationship

Entities

Task Entity:



Special Composite Key

```java
@Entity
@Table(name = "tasks")
@IdClass(TaskId.class) // <- Associate the Task entity with the TaskId composite key class
@Data
@AllArgsConstructor
@NoArgsConstructor(access = AccessLevel.PRIVATE, force=true)
public class Task implements Comparable<Task> {

    @Id
    @Column(name = "task_id")
    private int tId;

    @JsonIgnore // Exclude the attribute from the JSON response of the app
    @Id
    @Column(name = "user_id")
    private long uId;

    // No need for the @Column annotation
    private String title;

    @Enumerated(EnumType.ORDINAL)
    private Status status;


    // Set up the relationship with the User entity
    @JsonBackReference
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "ID", insertable = false, updatable = false)
    private User user;
```

Repositories

User Repository:

```
4  import org.springframework.data.jpa.repository.JpaRepository;
7
8  // USER REPOSITORY
9
10 public interface UserRepository extends JpaRepository<User, Long> {
11
12
13
14 }
```

Spring automatically adds basic CRUD operations thanks to Spring Data JPA

Repositories

Task Repository:

Applies JPQL with the @Query annotation to set custom queries easily:

```java
import java.util.List;

// TASK REPOSITORY

public interface TaskRepository extends JpaRepository<Task, TaskId> {

    // Retrieve the list of tasks for a particular user:
    @Query("SELECT t FROM Task t WHERE t.uId = :userId")
    List<Task> findByUser(long userId);

    // Retrieve a specific task, from a specific user:
    @Query("SELECT t FROM Task t WHERE t.tId = :taskId AND t.uId = :userId")
    Optional<Task> findTaskByUser(int taskId, long userId);

    // Count the number of tasks for a particular user:
    @Query("SELECT COUNT(*) FROM Task t WHERE t.uId = :userId")
    long countTasks(long userId);

}
```

Services

Interface and class implementation for both Entitites:

UserService Interface

```java
// USER SERVICE CONTRACT

public interface UserService {

    // CREATE new user
    User createUser(User user);

    // UPDATE existing user
    User updateUser(User user);

    // UPDATE password of an existing user:
    void changePassword(User user);

    // READ user's details based on the ID provided:
    User getUserById(long id);

    // READ all the users in the database:
    List<User> getAllUsers();

    // READ the number of users in the database:
    long countUser();

    // DELETE the user from the database by its ID:
    void deleteUser(long id);


}
```

UserService Implementation

```java
import java.util.List;

// CONCRETE IMPLEMENTATION OF THE USER SERVICE

@Service
public class UserServiceImpl implements UserService {

    // Inject the UserRepository onto the service:
    @Autowired
    private UserRepository userRepository;


    // Method to create a new user in the database:
    @Override
    @Transactional
    public User createUser(User user) {
        User newUser = userRepository.save(user);

        return newUser;
    }

```

Services

Interface and class implementation for both Entitites:

UserService Implementation

```java
// Method to update an existing user:
@Transactional
@Override
public User updateUser(User user) {
    return userRepository.save(user);
}


// Retrieve a user from the database through its ID:
@Override
public User getUserById(long id) {
    return userRepository.findById(id).orElseThrow(
            () -> new EntityNotFoundException("No user with ID #"+id+" was found..."));
}

// Retrieve all users of the database:
@Override
public List<User> getAllUsers() {
    return userRepository.findAll(); // <- Example of JpaRepository built-in method
}

// Count the # of users in the database:
@Override
public long countUser() {
    return userRepository.count();
}

// Delete the user from the database with its ID:
@Transactional
@Override
public void deleteUser(long id) {
    userRepository.deleteById(id);
}
```

Services

Interface and class implementation for both Entitites:

TaskService Interface

```java
import java.util.List;

//TASK SERVICE CONTRACT

public interface TaskService {

    // CREATE a new task for a specific user:
    Task addTaskToUser(Task task);

    // READ a specific task, from a specific user:
    Task getTaskByUser(long userId, int taskId);

    // READ the list of tasks from a particular user:
    List<Task> getTasksOfUser(long userId);

    // READ the number of tasks available for a user:
    long countTasks(long userId);

    // UPDATE a task details for a specific user:
    Task updateTaskOfUser(Task task);

    // DELETE a specific task based on its composite
    void deleteTask(TaskId id);

}
```

TaskService Implementation

```java
// TASK SESRVICE IMPLEMENTATION

@Service
public class TaskServiceImpl implements TaskService{

    // Automatically inject the TaskRepository onto the service to perform the
    @Autowired
    private TaskRepository taskRepository;

    @Transactional
    @Override
    public Task addTaskToUser(Task task) {
        return taskRepository.save(task); // Use built-in JpaRepository method
    }

    @Override
    public Task getTaskByUser(long userId, int taskId) {

        // Find the specific task of a specific user by means of their ID
        // If the task was not found, throw an exception that the entity was no
        return taskRepository.findTaskByUser(taskId, userId).orElseThrow(

                () -> new EntityNotFoundException("Task of ID #" + taskId + " f

                );
    }
```

Services

Interface and class implementation for both Entitites:

TaskService Implementation

```java
@Override
public long countTasks(long userId) {
    return taskRepository.countTasks(userId);
}

@Override
public List<Task> getTasksOfUser(long userId) {
    return taskRepository.findByUser(userId);
}

@Transactional
public Task updateTaskOfUser(Task task) {

    // Check if the task exists for the specific user:
    Task oldTask = getTaskByUser(task.getUId(), task.getTId());

    // If that's the case, then update the title and status of the existing task:
    oldTask.setTitle(task.getTitle());
    oldTask.setStatus(task.getStatus());

    // Save the existing task:
    return taskRepository.save(oldTask);

}

@Transactional
@Override
public void deleteTask(TaskId id) {
    taskRepository.deleteById(id);
```

REST Controller

Acts as the bridge between the client and the application

Handles HTTP requests and send the appropiate responses

Conversion between Java objects and JSON/XML responses

```
1  // USER REST CONTROLLER
2
3  @RestController // <- Tells Spring that the class sho
4  @RequestMapping("/api/users") // Entry point
5  public class UserRestController {
6
```

```
25 public class UserRestController {
26
27     @Autowired
28     private UserService userService;
29
30     // Retrieve the list of users of the database:
31     @GetMapping
32     public List<User> getAllUsers() {
33         return userService.getAllUsers();
34     }
35
```

REST Controller

UserRestController.java

GET request for retrieving a specific user

```java
// Get a specific user by its ID:
@GetMapping("/{user_id}")
public User getUser(@PathVariable("user_id") long id) {
    return userService.getUserById(id);
}
```

POST request to create a new user into the database

```java
// Create a new user into the database, using the HTTP request body data for the instance attributes:
@PostMapping("/create")
public String createUser(@RequestBody User newUser) {

    // Set the ID as 0, and let the repository handle the automatic assignment of the new user ID:
    newUser.setID(0);

    userService.createUser(newUser);

    // Show the following message, if successful:
    return "Created new user : \n"+ newUser;

}
```

REST Controller

TaskRestController.java

```java
//TASK REST CONTROLLER

@RestController
@RequestMapping(path = "/api/user/{user_id}") // In order to access the entry point, a user id must be passed:
public class TaskRestController {

    @Autowired
    private TaskService taskService;

    @Autowired
    private UserService userService;

    // Get the list of tasks for a specific user:
    @GetMapping("/tasks")     // Use the user_id path variable passed in the entry point:
    public List<Task> getAllTasks(@PathVariable("user_id") long userId) {

        return taskService.getTasksOfUser(userId);
    }

    // Get the number of tasks of a specific user:
    @GetMapping("/tasks/size")
    public String getNumTasks(@PathVariable("user_id") long userId) {
        long size = taskService.countTasks(userId);

        return "Number of tasks for User #"+userId + " : " + size;
    }

    // Get a specific task, for a specific user:
    @GetMapping("/task/{task_id}")
    public Task getTask(@PathVariable("user_id") long userId, @PathVariable("task_id") int taskId) {

```

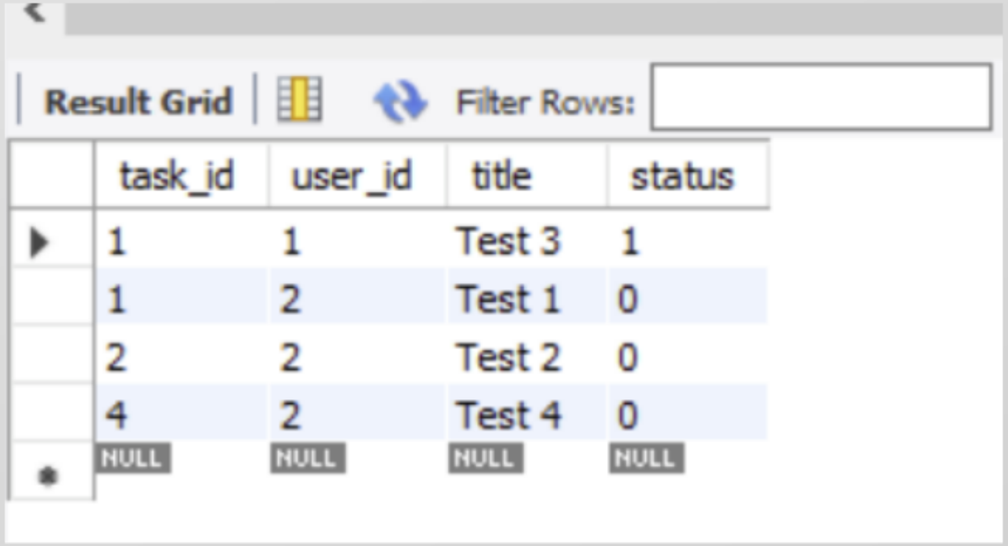
REST Controller

TaskRestController.java

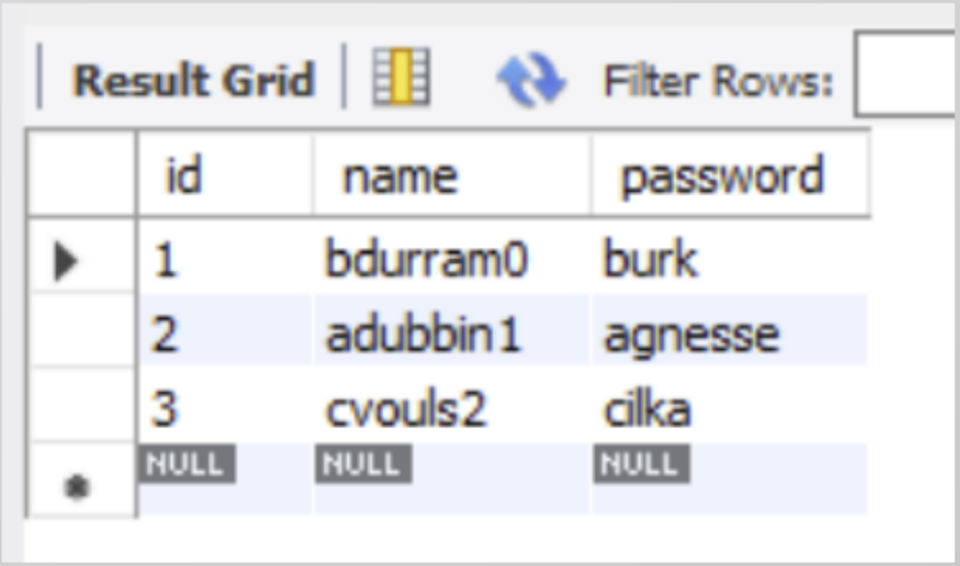
```java
    // Change an existing task's details of a specific user, based on the task ID and user ID:
    @PutMapping("/task")
    public Task updateTask(@PathVariable("user_id") long userId, @RequestBody Task task ) {

        // Set up the user ID, if it was not passed in the request's body:
        task.setUId(userId);

        return taskService.updateTaskOfUser(task);
    }

    // Delete a task from the list of a specific user:
    @DeleteMapping("/task/{task_id}")
    public String deleteTask(@PathVariable("user_id") long userId, @PathVariable("task_id") int taskId) {

        // Create the composite key, based on the task ID and user ID:
        TaskId tId = new TaskId(taskId, userId);

        taskService.deleteTask(tId);

        return "Deleted Task #"+taskId+ " from User with ID #"+userId;
    }
```

DEMONSTRATION

Initial Tasks and Users in the database:



task_id	user_id	title	status
1	1	Test 3	1
1	2	Test 1	0
2	2	Test 2	0
4	2	Test 4	0
NULL	NULL	NULL	NULL

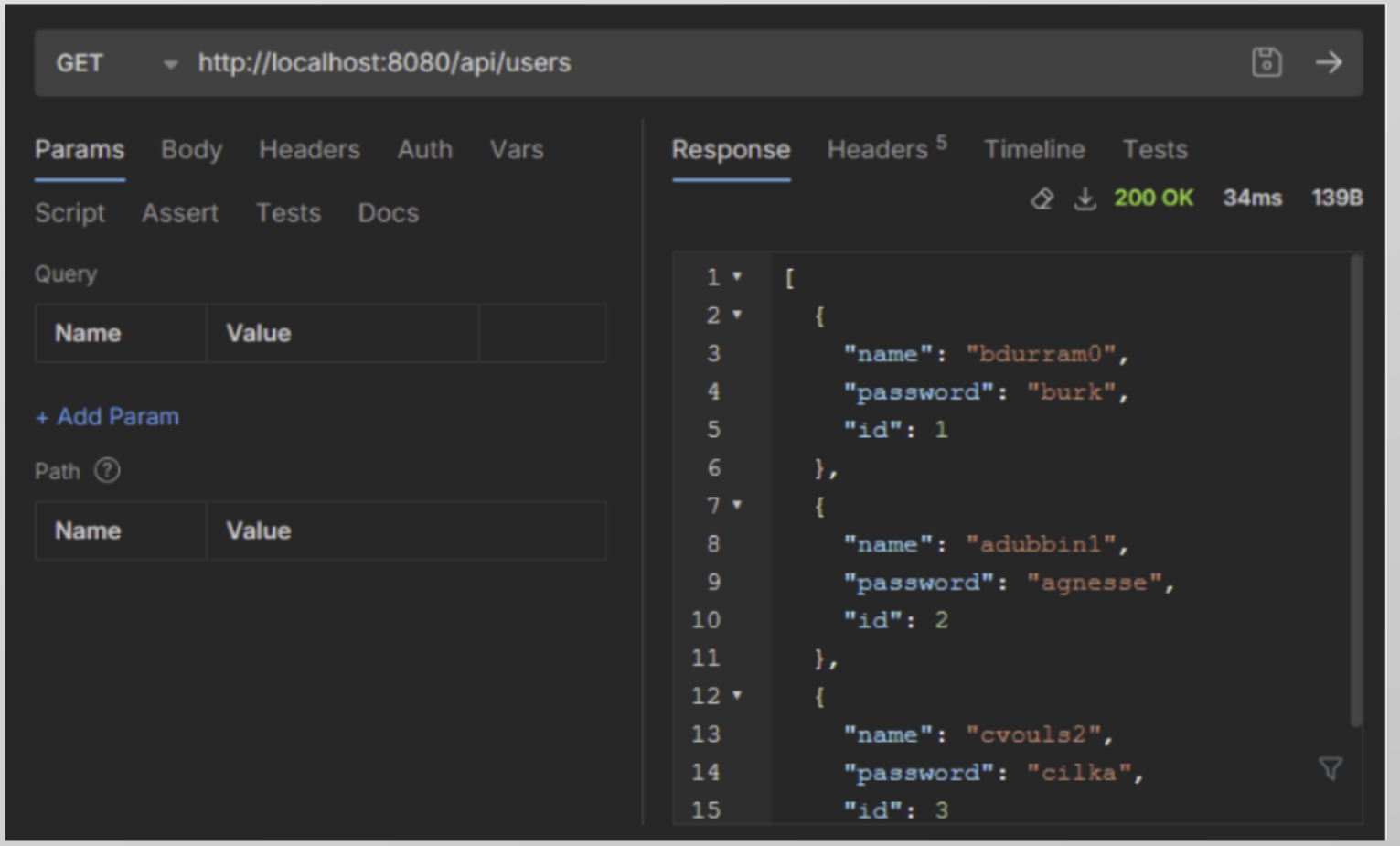


id	name	password
1	bdurram0	burk
2	adubbin1	agnesse
3	cvouls2	cilka
NULL	NULL	NULL

DEMONSTRATION

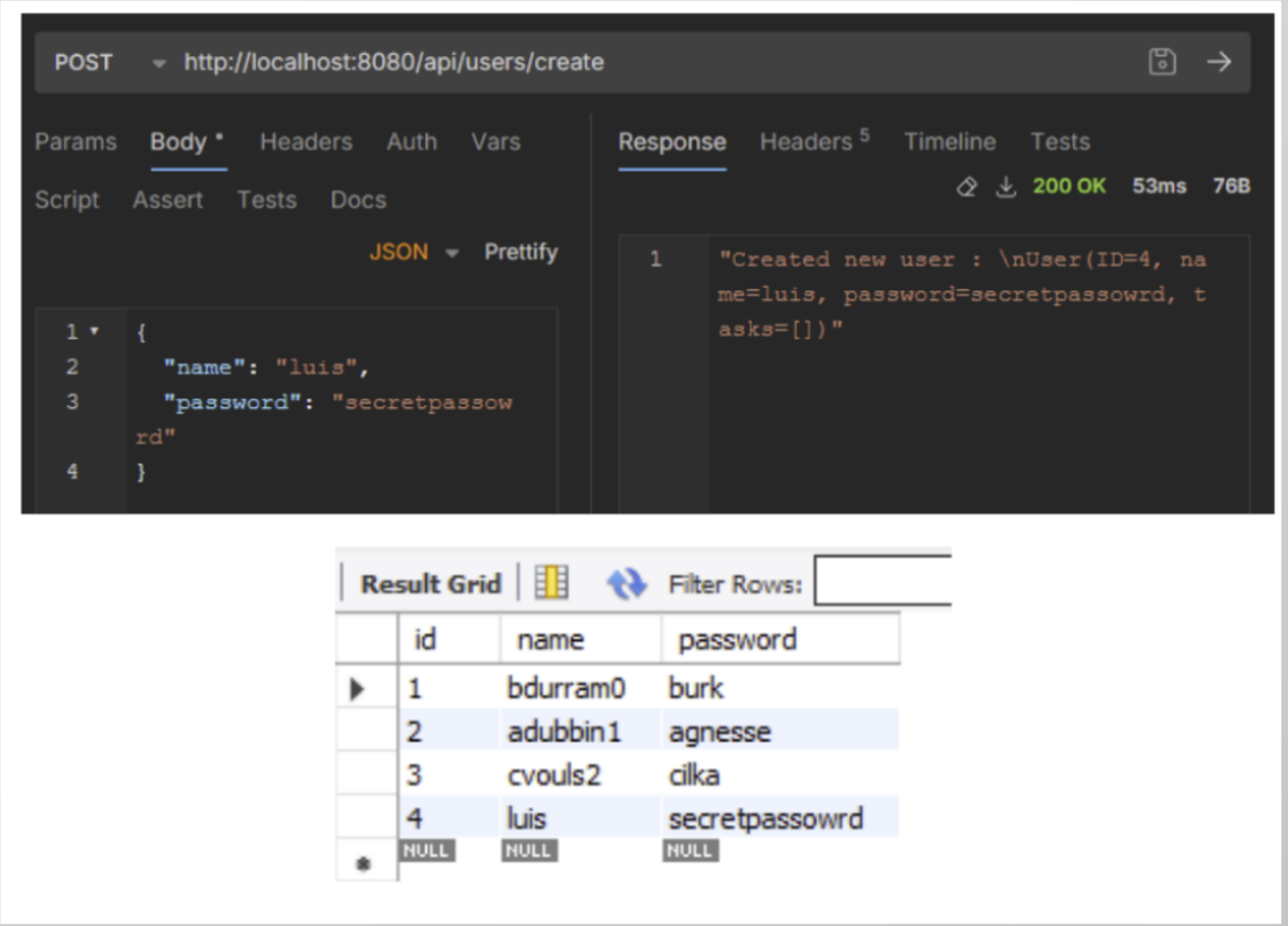
Using an API client like Postman / Bruno, we can send out HTTP requests to test the RESTFul application, performing operations such as:

List of users in the database:



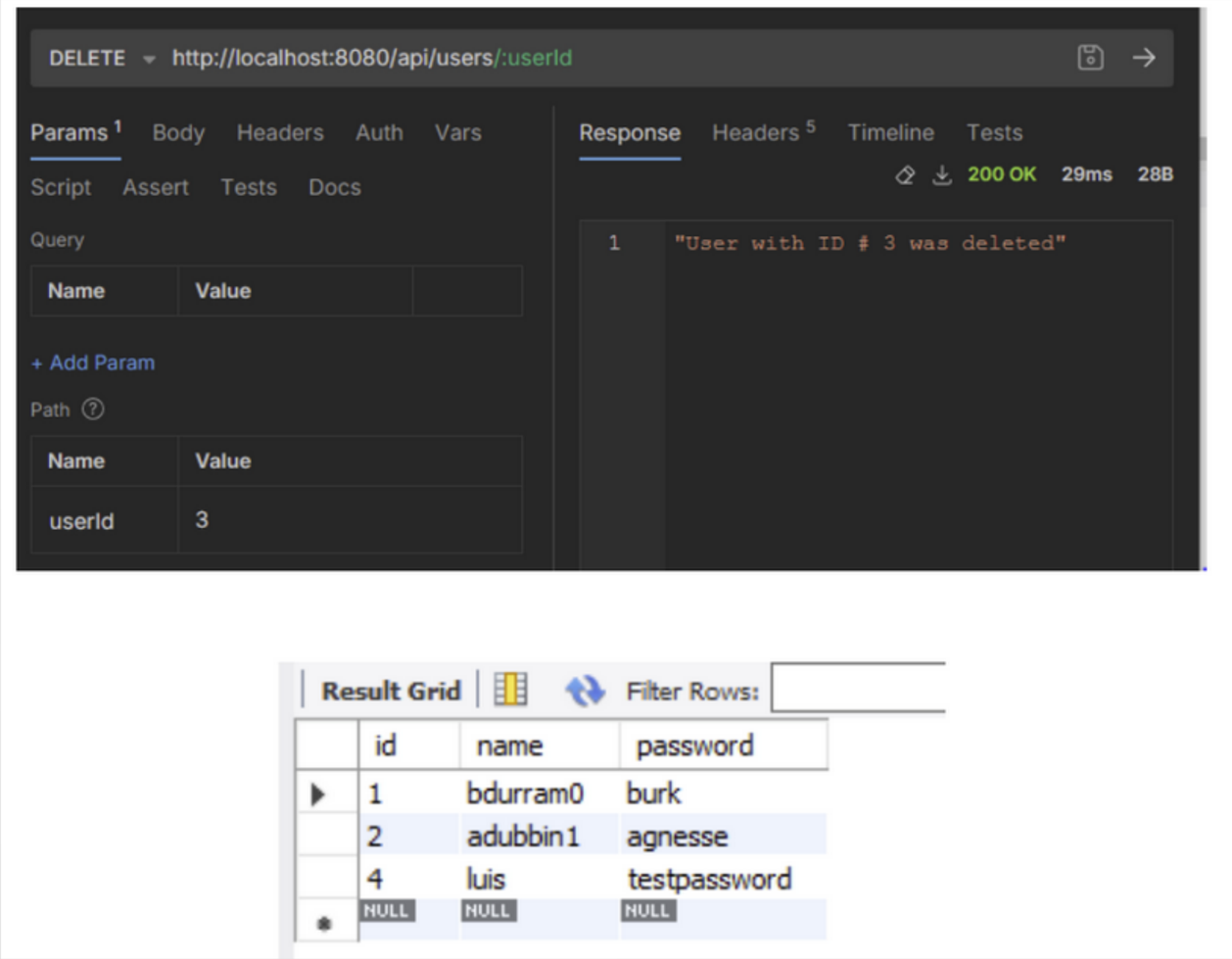
DEMONSTRATION

Create a new user into the database:



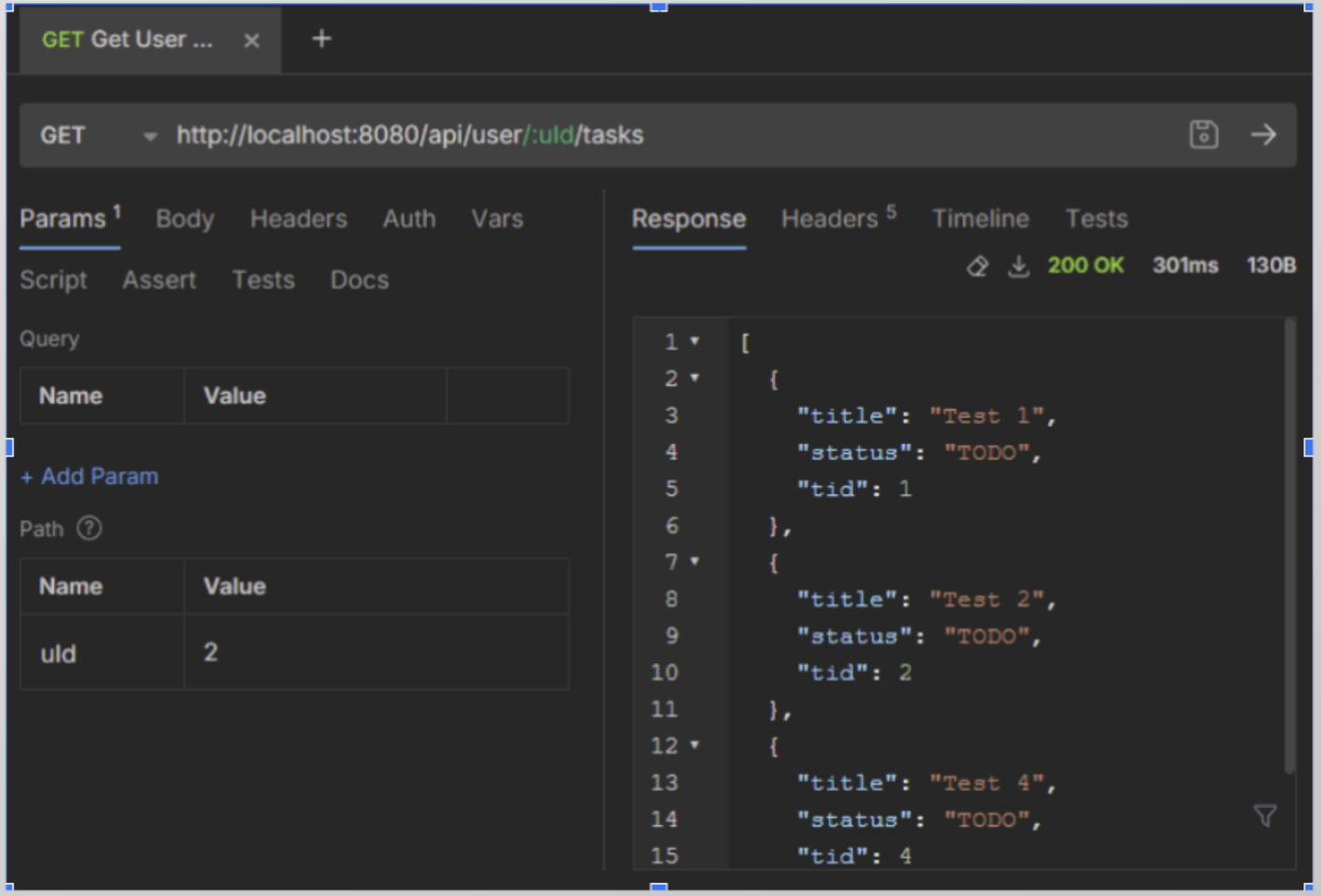
DEMONSTRATION

Delete user from the database



DEMONSTRATION

Get list of tasks of a specific user:



DEMONSTRATION

Create new task for User #3

POST http://localhost:8080/api/user/:uid/task

Params 1 Body • Headers Auth

Vars Script Assert Tests Docs

JSON Prettify

```
1 ▾ {
2       "title": "New task",
3       "status": 0
4   }
```

Response Headers 5 Timeline Tests

200 OK 167ms 44B

```
> POST http://localhost:8080/api/user/3/task
> content-type: application/json
> data
{
    "title": "New task",
    "status": 0
}

< 200
< content-type: application/json
< transfer-encoding: chunked
< date: Tue, 17 Sep 2024 16:54:27 GMT
< keep-alive: timeout=60
< connection: keep-alive
```

Result Grid | Filter Rows:

task_id	user_id	title	status
1	1	Test 3	1
1	2	Test 1	0
1	3	New task	0
2	2	Test 2	0
4	2	Test 4	0
NULL	NULL	NULL	NULL

DEMONSTRATION

Edit task #1 for User #3

PUT http://localhost:8080/api/user/:uld/task

Params 1 | Body * | Headers | Auth

Vars | Script | Assert | Tests | Docs

Query

Name	Value

+ Add Param

Path

Name	Value
uld	3

Response | Headers 5 | Timeline | Tests

200 OK 37ms 48B

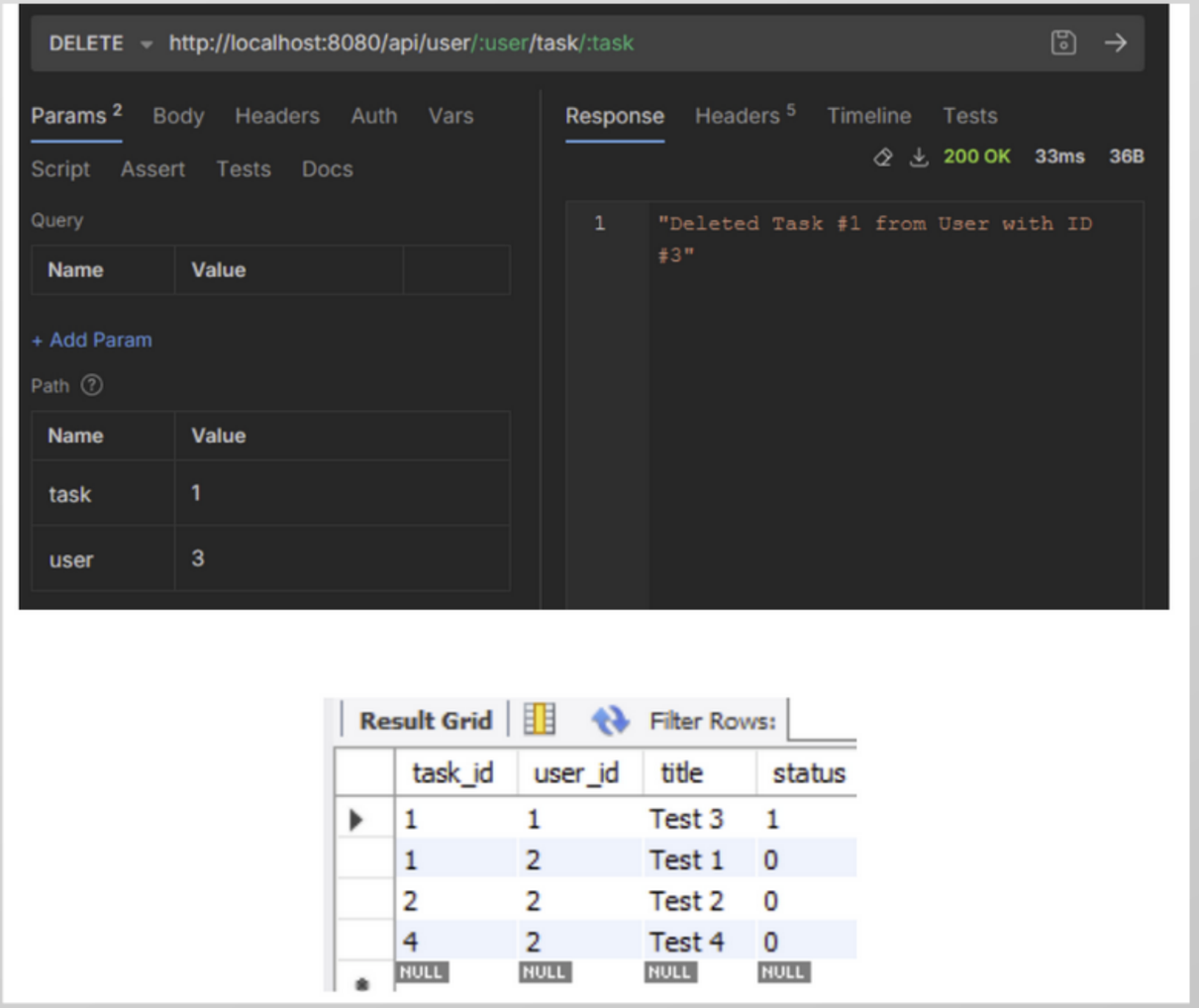
```
> PUT http://localhost:8080/api/user/3/task
> content-type: application/json
> data
{
    "tid": 1,
    "title": "Changed task",
    "status": 1
}

< 200
< content-type: application/json
< transfer-encoding: chunked
< date: Tue, 17 Sep 2024 16:56:04 GMT
< keep-alive: timeout=60
< connection: keep-alive
```

Result Grid | Filter Rows:

task_id	user_id	title	status
1	1	Test 3	1
1	2	Test 1	0
1	3	Changed task	1

DEMONSTRATION

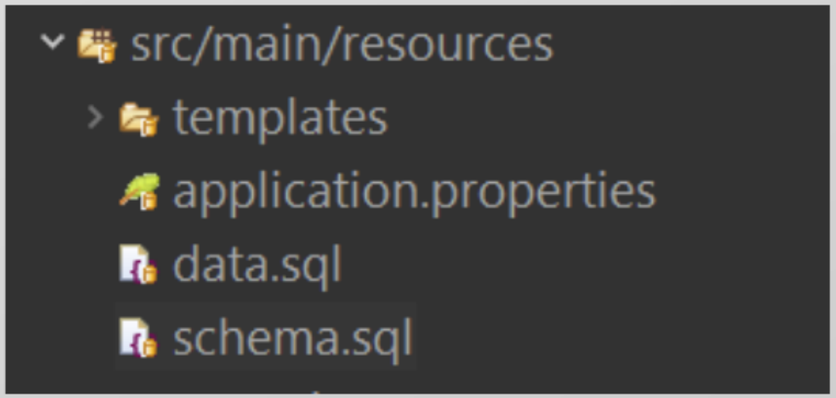
Delete task #1 for User #3



MYSQL SETUP

```
3  CREATE USER IF NOT EXISTS 'fp'@'%' IDENTIFIED BY 'finalpro
4
5  GRANT ALL PRIVILEGES ON * . * TO 'fp'@'%';
6
7  -- Database for MVC example
8  CREATE DATABASE IF NOT EXISTS `mvc_db`;
9
10 -- Database for REST example
11 CREATE DATABASE IF NOT EXISTS `rest_db`;
12
```

```
1  DROP TABLE IF EXISTS `authorities`;
2  DROP TABLE IF EXISTS `users`;
3
4  CREATE TABLE `users` (
5      `username` varchar(50) NOT NULL,
6      `password` varchar(50) NOT NULL,
7      `enabled` tinyint NOT NULL,
8
9      PRIMARY KEY (`username`)
10 );
11
12
13 CREATE TABLE `authorities` (
14     `username` varchar(50) NOT NULL,
15     `authority` varchar(50) NOT NULL,
16
17     UNIQUE (`username`, `authority`),
18
19     CONSTRAINT `authorities_ibfk_1`
20     FOREIGN KEY (`username`)
21     REFERENCES users (`username`)
22 );
```

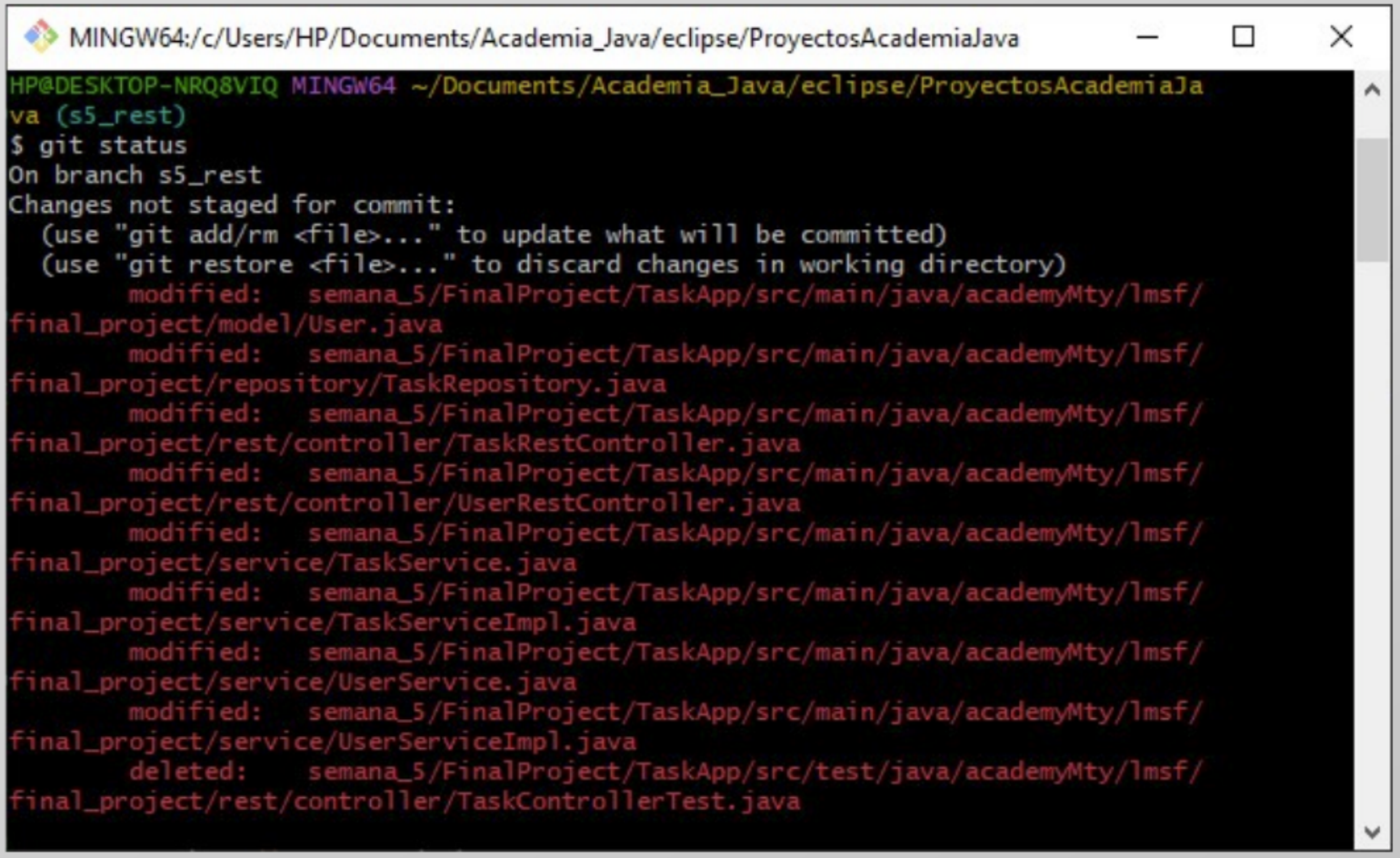



```
1  spring.application.name=Final_Project
2  spring.datasource.url=jdbc:mysql://localhost:3306/mvc_db
3  spring.datasource.username=fp
4  spring.datasource.password=finalproject
5  spring.sql.init.schema-locations=classpath:schema.sql
6  spring.sql.init.data-locations=classpath:data.sql
7  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
8  spring.jpa.hibernate.ddl-auto=create
```

USE OF GIT

Developed projects in separate branches (s5_mvc, s5_rest)

Once done, merge into semana_5, then main



Thank you for listening!