

Exideral®

WEEK 5

FINAL PROJECT

LUIS MIGUEL SÁNCHEZ FLORES

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



Thymeleaf



MVC Application

Spring Data JPA



JUnit + Mockito



Model Component

Task Entity for JPA persistence

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

Leverage **Lombok** library
to reduce boilerplate code

```
4+ import jakarta.persistence.*;■
5
6
7  @Entity // Specify that the class is an entity
8  @Table(name = "task") // Sets the database table to which a
9
10 // LOMBOK ANNOTATIONS
11 @Data // Generate getters, setters, toString and hashCode m
12 @AllArgsConstructor // Sets up a constructor that requires
13 @NoArgsConstructor // Sets up an empty constructor
14 public class Task {
15
16     // Primary key attribute
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private int ID;
20
21     private String title;
22
23     // Specify how the enum value should be persisted (in t
24     @Enumerated(EnumType.ORDINAL)
25     private Status status;
26
27     public enum Status {
28         TODO, DONE
29     }
30 }
```

Controller Component

Central hub that interacts with Model and View

```
18  
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>
2 <html xmlns="http://www.w3.org/1999/xhtml">
3   <head>
```

```
<!-- Enable Thymeleaf's attributes and tags -->
  xmlns:th="http://www.thymeleaf.org">
```

delete-task.html

```
y>
<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

```
3+ import java.util.List;[]
4+ import java.util.List;[]
10
11 public interface TaskRepository extends JpaRepository<Task, Integer> {
12
13     // Get list of tasks order by the status:
14@Query("SELECT t FROM Task t ORDER BY t.status ASC")
15     List<Task> findAllOrderByStatus();
16
17 }
```

```
3+ import java.util.List;[]
12
13 @Service
14 public class TaskServiceImpl implements TaskService{
15
16@Autowired
17     private TaskRepository taskRepository;
18
19@Override
20     public List<Task> findAllTasks() {
21         List<Task> allTasks = taskRepository.findAllOrderByStatus();
22         return allTasks;
23     }
24
25@Override
26@Transactional
27     public Task addTask(Task task) {
28         Task newTask = taskRepository.save(task);
29         return newTask;
30     }
```

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 endpoint requires user role  
            .requestMatchers("/").permitAll() // Allow anyone to access the index page  
            .anyRequest().authenticated() // Any other requests should be authenticated  
        ).
```

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         exampleTask = new Task(1, "Test 1", Task.Status.TODO);
49     }
50 }
```

```
// 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
}
```

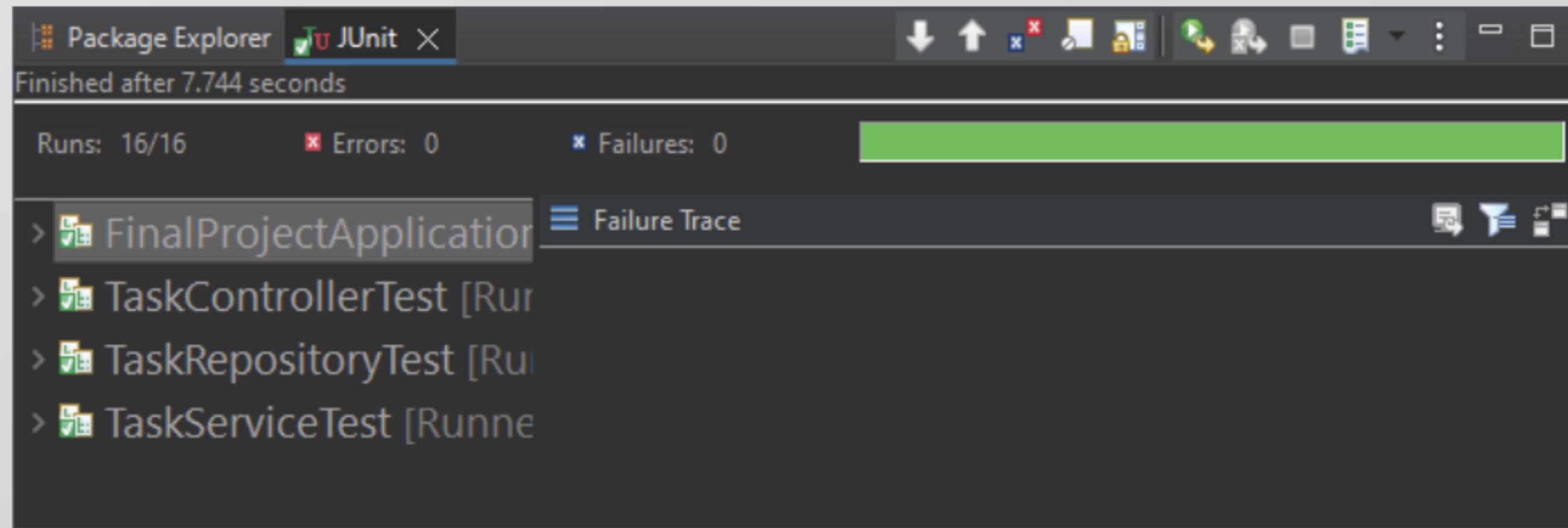
Unit testing

Checking out if the application works as intended!



```
Finished after 4.486 seconds
Runs: 1/1    ✘ Errors: 0    ✘ Failures: 1
TaskControllerTest [Run]
  testTaskDelete() (0.13s)
    ! java.lang.AssertionError: No ModelAndView found
      at org.springframework.test.util.AssertionErrors.fail()
      at org.springframework.test.web.servlet.result.MockMvcResultMatchers.assertNoModelAndView(MockMvcResultMatchers.java:110)
      at org.springframework.test.web.servlet.MockMvc$1.andExpect(MockMvc.java:170)
      at academyMty.lmsf.final_project.mvc.controller.TaskControllerTest.testTaskDelete(TaskControllerTest.java:26)
      at java.base/java.lang.reflect.Method.invoke(Method.java:52)
      at java.base/java.util.ArrayList.forEach(ArrayList.java:146)
      at java.base/java.util.ArrayList.forEach(ArrayList.java:146)
```

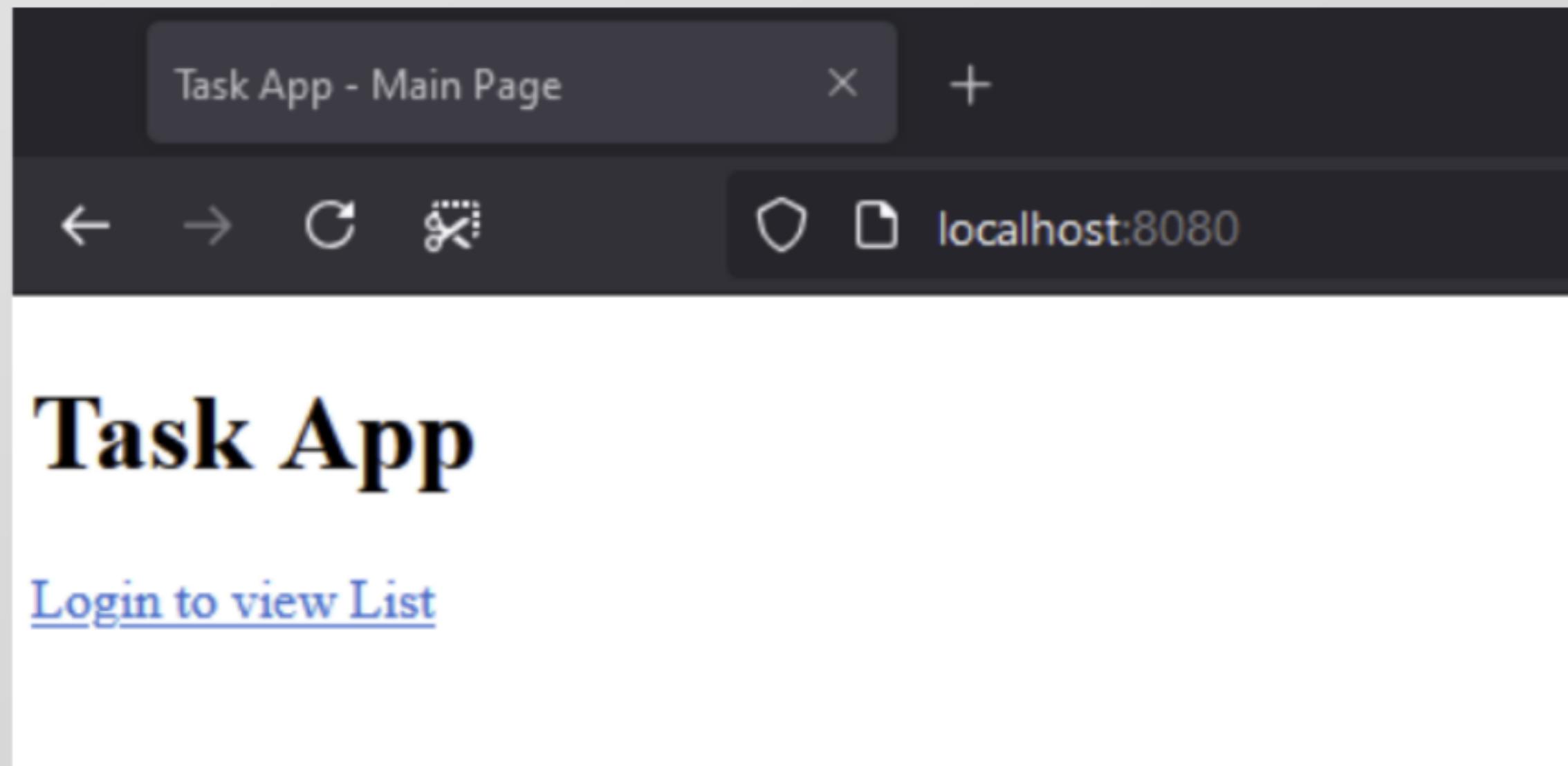
Unit testing



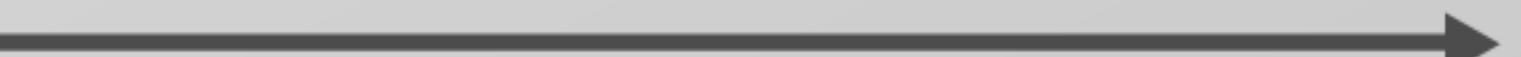
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
TaskMVC	97.9 %	969	21	990
src/main/java	95.0 %	209	11	220
academyMty.lmsf.final_proj	37.5 %	3	5	8
academyMty.lmsf.final_proj	92.9 %	52	4	56
academyMty.lmsf.final_proj	97.3 %	72	2	74
academyMty.lmsf.final_proj	100.0 %	24	0	24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

DEMONSTRATION

Index.html



Clicking on the link....

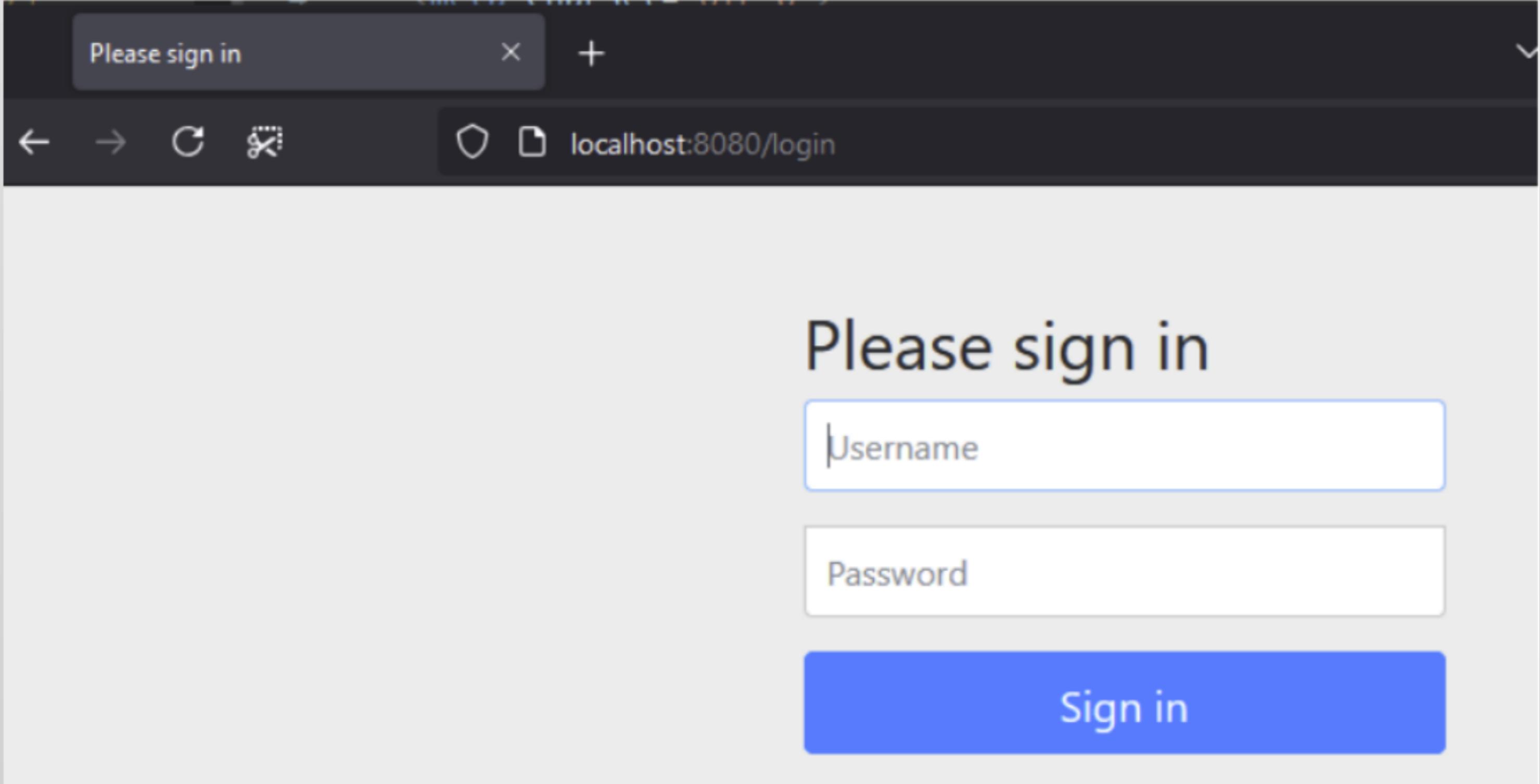


DEMONSTRATION

Prompts the login page!



Result Grid			
	username	password	enabled
▶	luis	{noop}xiderallmsf	1
*	NULL	NULL	NULL



Please sign in

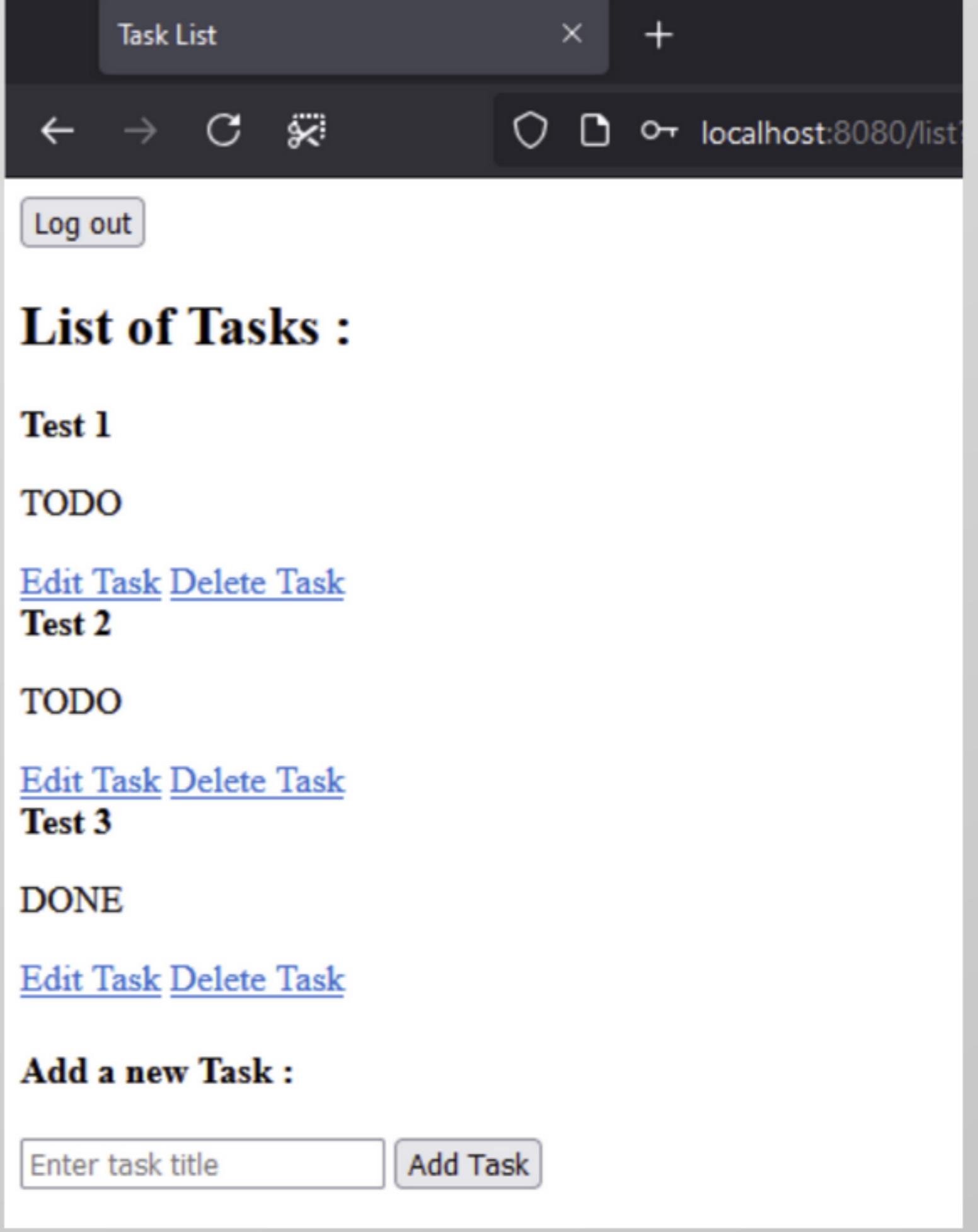
Username

Password

Sign in

DEMONSTRATION

List of Tasks (with initial data)



The screenshot shows a web browser window titled "Task List". The address bar indicates the URL is "localhost:8080/list". A "Log out" button is visible at the top left. The main content area displays a list of tasks:

- Test 1**
TODO
[Edit Task](#) [Delete Task](#)
- Test 2**
TODO
[Edit Task](#) [Delete Task](#)
- Test 3**
DONE
[Edit Task](#) [Delete Task](#)

Add a new Task :

DEMONSTRATION

ADDING A NEW TASK

Add a new Task :

List of Tasks :

Test 1

TODO

[Edit Task](#) [Delete Task](#)

Test 2

TODO

[Edit Task](#) [Delete Task](#)

Move around stuff

TODO

[Edit Task](#) [Delete Task](#)

Test 3

DONE

[Edit Task](#) [Delete Task](#)

DEMONSTRATION

UPDATING EXISTING TASK

Edit Task

Title:
Moved stuff already :)

Task Status:
DONE ▾

Save Changes

List of Tasks :

Test 1

TODO

[Edit Task](#) [Delete Task](#)

Test 2

TODO

[Edit Task](#) [Delete Task](#)

Test 3

DONE

[Edit Task](#) [Delete Task](#)

Moved stuff already :)

DONE

[Edit Task](#) [Delete Task](#)

DEMONSTRATION

DELETING TASK

Delete Task

localhost:8080/list/task/delete/3

Delete Task ?

Are you sure you want to delete the task with ID # 3

Title:
Test 3

Status:
DONE

List of Tasks :

Test 1
TODO
[Edit Task](#) [Delete Task](#)

Test 2
TODO
[Edit Task](#) [Delete Task](#)

Result Grid | Filter Rows:

	id	status	title
▶	1	0	Test 1
◀	2	0	Test 2
*	NULL	NULL	NULL

Spring Web



Spring Data JPA



REST Application

Lombok



Entities

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

User Entity:

```
13  @Entity
14  @Table(name="users") // <- Table the entity should map to
15  // LOMBOK ANNOTATIONS
16  @Data
17  @AllArgsConstructor
18  @NoArgsConstructor
19  public class User {
20
21      // UserID that represents the primary key of the table
22  •     @Id
23      @GeneratedValue(strategy = GenerationType.IDENTITY)
24      private long ID;
25
26  •     @Column(nullable = false, unique = true)
27      private String name;
28
29  •     @Column(nullable = false)
30      private String password;
31
32  •     @JsonIgnore
33     @JsonManagedReference
34     @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
35     @OrderBy("task_id ASC") // Order the list of tasks by their IDs
36     private List<Task> tasks = new ArrayList<>();
37 }
38
```

One To Many
relationship {

Entities

Task Entity:

Special
Composite
Key {

```
9
10 @Entity
11 @Table(name = "tasks")
12 @IdClass( taskId.class ) // <- Associate the Task entity with the TaskId composite key class
13 @Data
14 @AllArgsConstructor
15 @NoArgsConstructor(access = AccessLevel.PRIVATE, force=true)
16 public class Task implements Comparable<Task> {
17
18@ Id
19     @Column(name = "task_id")
20     private int tId;
21
22@ JsonIgnore // Exclude the attribute from the JSON response of the app
23     @Id
24     @Column(name = "user_id")
25     private long uId;
26
27     // No need for the @Column annotation
28     private String title;
29
30@ Enumerated( EnumType.ORDINAL )
31     private Status status;
32
33
34     // Set up the relationship with the User entity
35@ JsonBackReference
36     @ManyToOne
37     @JoinColumn(name = "user_id", referencedColumnName = "ID", insertable = false, updatable = false)
38     private User user;
39
```

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:

```
4+ import java.util.List;■
2
3 // TASK REPOSITORY
4
5 public interface TaskRepository extends JpaRepository<Task, TaskId> {
6
7     // Retrieve the list of tasks for a particular user:
8     @Query("SELECT t FROM Task t WHERE t.uId = :userId")
9     List<Task> findByUser(long userId);
0
1     // Retrieve a specific task, from a specific user:
2     @Query("SELECT t FROM Task t WHERE t.tId = :taskId AND t.uId = :userId")
3     Optional<Task> findTaskByUser(int taskId, long userId);
4
5     // Count the number of tasks for a particular user:
6     @Query("SELECT COUNT(*) FROM Task t WHERE t.uId = :userId")
7     long countTasks(long userId);
8
9 }
```

Services

Interface and class implementation for both Entities:

UserService Interface

```
7 // USER SERVICE CONTRACT
8
9 public interface UserService {
10
11     // CREATE new user
12     User createUser(User user);
13
14     // UPDATE existing user
15     User updateUser(User user);
16
17     // UPDATE password of an existing user:
18     void changePassword(User user);
19
20     // READ user's details based on the ID provided:
21     User getUserById(long id);
22
23     // READ all the users in the database:
24     List<User> getAllUsers();
25
26     // READ the number of users in the database:
27     long countUser();
28
29     // DELETE the user from the database by its ID:
30     void deleteUser(long id);
31
32
33 }
```

UserService Implementation

```
3+import java.util.List;
12
13 // CONCRETE IMPLEMENTATION OF THE USER SERVICE|
14
15 @Service
16 public class UserServiceImpl implements UserService {
17
18     // Inject the UserRepository onto the service:
19@Autowired
20     private UserRepository userRepository;
21
22
23     // Method to create a new user in the database:
24@Override
25     @Transactional
26     public User createUser(User user) {
27         User newUser = userRepository.save(user);
28
29         return newUser;
30     }
31
```

Services

Interface and class implementation for both Entities:

UserService Implementation

```
// 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 Entities:

TaskService Interface

```
3+import java.util.List;■
7
8 //TASK SERVICE CONTRACT
9
10 public interface TaskService {
11
12     // CREATE a new task for a specific user:
13     Task addTaskToUser(Task task);
14
15     // READ a specific task, from a specific user:
16     Task getTaskByUser(long userId, int taskId);
17
18     // READ the list of tasks from a particular user:
19     List<Task> getTasksOfUser(long userId);
20
21     // READ the number of tasks available for a user:
22     long countTasks(long userId);
23
24     // UPDATE a task details for a specific user:
25     Task updateTaskOfUser(Task task);
26
27     // DELETE a specific task based on its composite
28     void deleteTask(TaskId id);
29
30 }
```

TaskService Implementation

```
5 // TASK SESRVICE IMPLEMENTATION
6
7 @Service
8 public class TaskServiceImpl implements TaskService{
9
10     // Automatically inject the TaskRepository onto the service to perform the
11     @Autowired
12     private TaskRepository taskRepository;
13
14     @Transactional
15     @Override
16     public Task addTaskToUser(Task task) {
17         return taskRepository.save(task); // Use built-in JpaRepository method
18     }
19
20     @Override
21     public Task getTaskByUser(long userId, int taskId) {
22
23         // Find the specific task of a specific user by means of their ID
24         // If the task was not found, throw an exception that the entity was no
25         return taskRepository.findTaskByUser(taskId, userId).orElseThrow(
26
27             () -> new EntityNotFoundException("Task of ID #" + taskId + " f
28
29         );
30     }
31 }
```

Services

Interface and class implementation for both Entities:

TaskService Implementation

```
@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.getUserId(), 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 appropriate responses

Conversion between Java objects and JSON/XML responses

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

```
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

```
// 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

```
// 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

```
|3 //TASK REST CONTROLLER
|4
|5 @RestController
|6 @RequestMapping(path = "/api/user/{user_id}") // In order to access the entry point, a user id must be passed:
|7 public class TaskRestController {
|8
|9@    @Autowired
|0     private TaskService taskService;
|1
|2@    @Autowired
|3     private UserService userService;
|4
|5     // Get the list of tasks for a specific user:
|6@    @GetMapping("/tasks")      // Use the user_id path variable passed in the entry point:
|7     public List<Task> getAllTasks(@PathVariable("user_id") long userId) {
|8
|9         return taskService.getTasksOfUser(userId);
|0     }
|1
|2     // Get the number of tasks of a specific user:
|3@    @GetMapping("/tasks/size")
|4     public String getNumTasks(@PathVariable("user_id") long userId) {
|5         long size = taskService.countTasks(userId);
|6
|7         return "Number of tasks for User #"+userId + " : " + size;
|8     }
|9
|0     // Get a specific task, for a specific user:
|1@    @GetMapping("/task/{task_id}")
|2     public Task getTask(@PathVariable("user_id") long userId, @PathVariable("task_id") int taskId) {
```

REST Controller

TaskRestController.java

```
81
82     // Change an existing task's details of a specific user, based on the task ID and user ID:
83     @PutMapping("/task")
84     public Task updateTask(@PathVariable("user_id") long userId, @RequestBody Task task) {
85
86         // Set up the user ID, if it was not passed in the request's body:
87         task.setUserId(userId);
88
89         return taskService.updateTaskOfUser(task);
90     }
91
92     // Delete a task from the list of a specific user:
93     @DeleteMapping("/task/{task_id}")
94     public String deleteTask(@PathVariable("user_id") long userId, @PathVariable("task_id") int taskId) {
95
96         // Create the composite key, based on the task ID and user ID:
97         TaskId tId = new TaskId(taskId, userId);
98
99         taskService.deleteTask(tId);
100
101         return "Deleted Task #"+taskId+ " from User with ID #"+userId;
102     }
```

DEMONSTRATION

Initial Tasks and Users in the database:

Result Grid | Filter Rows:

	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

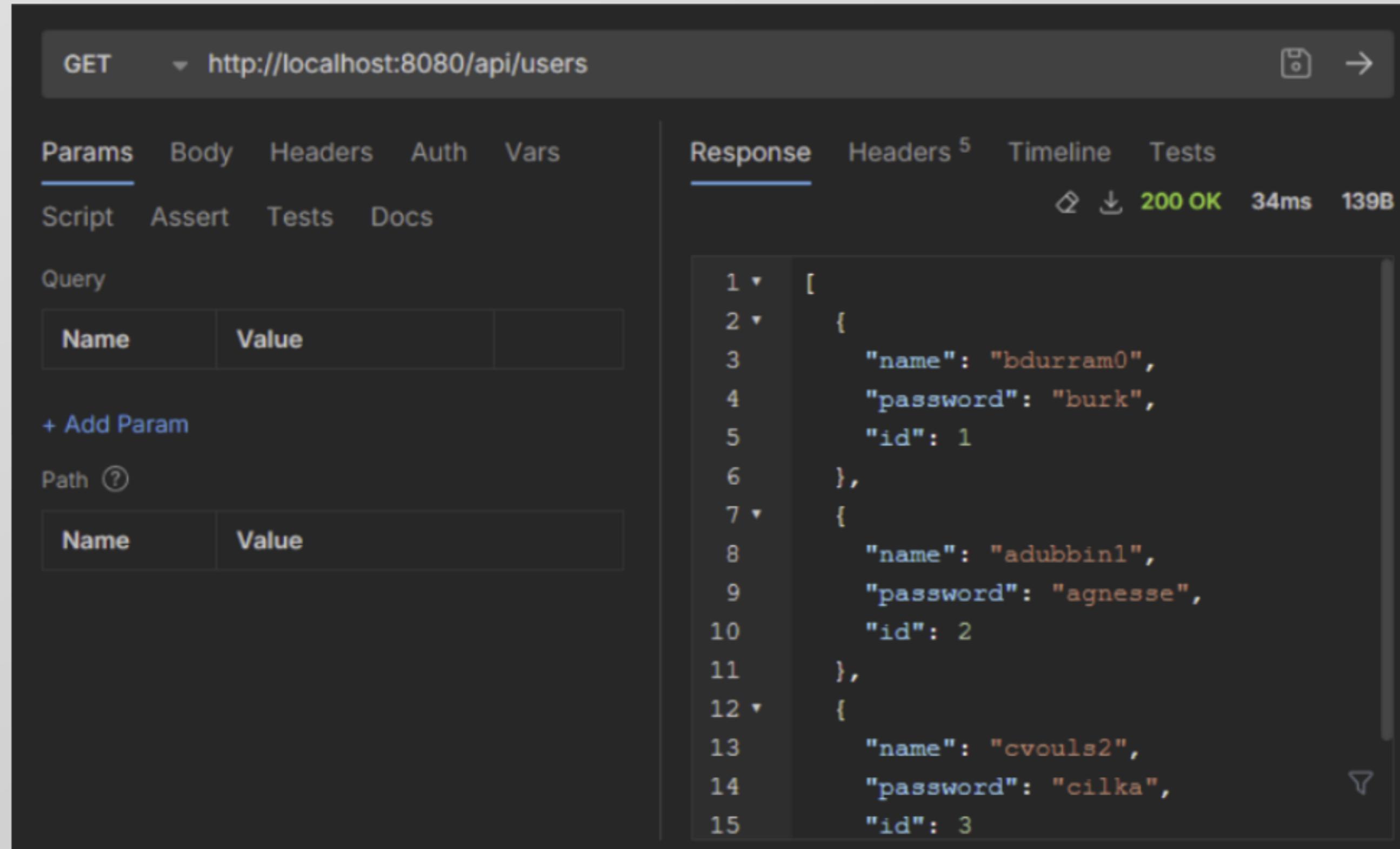
Result Grid | Filter Rows:

	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:



The screenshot shows a dark-themed API client interface. At the top, a header bar displays "GET" and the URL "http://localhost:8080/api/users". Below the header, there are tabs for "Params", "Body", "Headers", "Auth", and "Vars", with "Params" being the active tab. Under "Params", there are sections for "Query", "Script", "Assert", "Tests", and "Docs". A table for "Query" has columns "Name" and "Value", with one row currently selected. Below the table is a "+ Add Param" button. Another table for "Path" also has columns "Name" and "Value", with one row currently selected. To the right of the "Params" section, there are tabs for "Response", "Headers", "Timeline", and "Tests", with "Response" being the active tab. The "Response" tab shows a status of "200 OK" and a response time of "34ms" for a response size of "139B". The response body is a JSON array of three user objects, each with properties "name", "password", and "id".

```
1 [  
2 {  
3   "name": "bdurram0",  
4   "password": "burk",  
5   "id": 1  
6 },  
7 {  
8   "name": "adubbin1",  
9   "password": "agnesse",  
10  "id": 2  
11 },  
12 {  
13   "name": "cvouls2",  
14   "password": "cilkta",  
15   "id": 3
```

DEMONSTRATION

Create a new user into the database:

A screenshot of a REST client interface showing a POST request to `http://localhost:8080/api/users/create`. The request body is a JSON object with fields `name` and `password`. The response is a 200 OK status with a message indicating a new user was created.

POST `http://localhost:8080/api/users/create`

Params Body * Headers Auth Vars

Script Assert Tests Docs

JSON Prettify

```
1 {  
2   "name": "luis",  
3   "password": "secretpassow  
rd"  
4 }
```

Response Headers 5 Timeline Tests

200 OK 53ms 76B

```
1 "Created new user : \nUser(ID=4, na  
me=luis, password=secretpassowrd, t  
asks=[])"
```

A screenshot of MySQL Workbench showing the `Result Grid` for the `users` table. The grid displays four rows of data: id, name, and password.

	id	name	password
▶	1	bdurram0	burk
	2	adubbin1	agnesse
	3	cvouls2	cilka
	4	luis	secretpassowrd
	NULL	NULL	NULL

DEMONSTRATION

Delete user from the database

The screenshot shows a REST client interface with the following details:

Request URL: `DELETE http://localhost:8080/api/users/:userId`

Params: 1

Name	Value
userId	3

Response: 200 OK

1	"User with ID # 3 was deleted"

Result Grid:

	id	name	password
▶	1	bdurram0	burk
	2	adubbin1	agnesse
	4	luis	testpassword
*	NULL	NULL	NULL

DEMONSTRATION

Get list of tasks of a specific user:

The screenshot shows a REST client interface with the following details:

- Request:** GET http://localhost:8080/api/user/:uid/tasks
- Params:** uid = 2
- Response Headers:** 200 OK, 301ms, 130B
- Response Body:**

```
1 [  
2 {  
3   "title": "Test 1",  
4   "status": "TODO",  
5   "tid": 1  
6 },  
7 {  
8   "title": "Test 2",  
9   "status": "TODO",  
10  "tid": 2  
11 },  
12 {  
13   "title": "Test 4",  
14   "status": "TODO",  
15   "tid": 4
```

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 ②

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

The screenshot shows a REST client interface with the following details:

Request URL: `DELETE http://localhost:8080/api/user/:user/task/:task`

Params: 2

Name	Value
task	1
user	3

Response: 200 OK

1 "Deleted Task #1 from User with ID #3"

Result Grid:

	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

MYSQL SETUP

```
2
3 • CREATE USER IF NOT EXISTS 'fp'@'%' IDENTIFIED BY 'finalproject';
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`;
```

```
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 );|
```

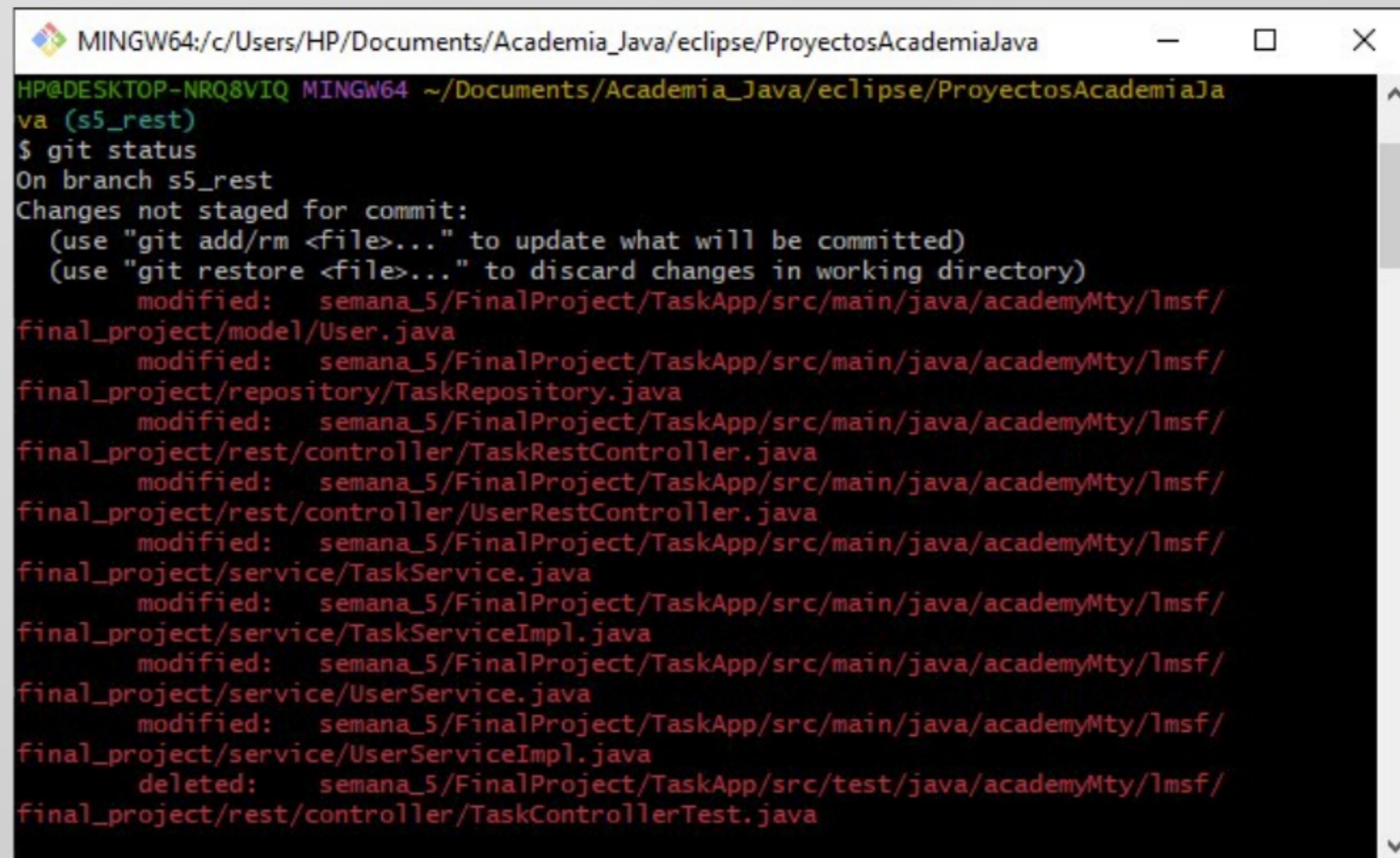
```
src/main/resources
  templates
  application.properties
  data.sql
  schema.sql
```

```
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



The screenshot shows a terminal window titled "MINGW64:/c/Users/HP/Documents/Academia_Java/eclipse/ProyectosAcademiaJava". The command \$ git status is run, showing the following output:

```
HP@DESKTOP-NRQ8VIQ MINGW64 ~/Documents/Academia_Java/eclipse/ProyectosAcademiaJava
va (s5_rest)
$ git status
On branch s5_rest
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/model/User.java
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/repository/TaskRepository.java
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/rest/controller/TaskRestController.java
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/rest/controller/UserRestController.java
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/service/TaskService.java
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/service/TaskServiceImpl.java
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/service/UserService.java
    modified:   semana_5/FinalProject/TaskApp/src/main/java/academyMty/lmsf/
final_project/service/UserServiceImpl.java
    deleted:   semana_5/FinalProject/TaskApp/src/test/java/academyMty/lmsf/
final_project/rest/controller/TaskControllerTest.java
```

Thank you for listening!