UCLL HOGESCHOOL	Number of pages: 5	Lecturer: J. Pieck, B. Van Impe course unit (OPO/OLA): Back-End Development Code OPO/OLA: MBI03j The **standard time** for this exam is **2h20min**, but each student is given 30% extra time, so **3 hours total**
Date: 01/09/2025 Group: **1BCS**		Student name: Student number: Time of submission: .....hours.....min.

Allowed tools

- The exam is done on your own laptop.
- No smartphone, nor smartwatch.
- Documentation you may use on your laptop:
  - Learning materials we made available on Toledo
  - Pre-made summaries
  - Code your own project and exercises
- Other documentation/tools you may *not* use: internet applications, messaging apps, AI tools (both online and offline)

Composition of the exam

The exam consists of two parts:

- A theoretical part as an ANS test (at 4 points).
- A practical part as an ANS test (on 12 points) including:
  - This assignment as a PDF
  - A zip with additional code
- Access code ANS is projected during the exam.

Startup

- Connect to the wifi network 'ans'. The password is 'exam001'.
- **!! Important:** make sure you are logged out of Postman before the exam. Otherwise, you will not be able to use Postman during the exam.
- Download the zip with additional code from ANS. This contains a first version of the domain classes and schema.sql from which you can start. Place these in the appropriate package of your start code.

Submitting

- You submit your code in one zip file named "LastName_FirstName.zip"
- Submit by uploading this zip to the ANS test.
- Once everything is uploaded, check again in the test to make sure your zip is there. Download your submission to another folder, unzip, and check that you submitted the correct one.
- Hand over this assignment bundle and sign the attendance sheet.

Evaluation

- You can score points for each (partial) question.
- Some (partial) questions are related to others, some are not.
- The functionality must work. Otherwise, you will not get points for that (sub)question.
- End-points must be implemented exactly as described in the API table. Otherwise, you will not receive points for that (sub)question.
- If the functionality works, the code must be implemented as was taught in class. Otherwise you can lose points for a (sub)question.
- You do not need to write tests unless explicitly requested.
- Work in small steps, do not do more for a question than is explicitly asked.

Practical Exam

We are developing an API for a task management system. Users must be able to manage projects and track tasks within those projects. In addition, notifications must be generated for important task changes.

You will use the starting code you were given for the exam, along with the additional code you will get with this exam. You do not need User's existing end-points on this exam, but feel free to leave them in.
The additional code you get on the exam is incomplete and you will need to expand.

Question 1 (2 points)

For this first query, implement an endpoint that allows updating the details of an existing project (Project) based on ID. The request body contains the fields to be updated. The name field is always mandatory, description is optional. If description is not provided, the previous value is retained.

- Code according to the principles of layered architecture and use JPA.
- Make sure the Project entity has the name (String) and description (String) fields.
- Add test data with DbInitializer so there are projects to update.
- Implement input validation: name should not be empty if provided, description should not be longer than 25 characters if provided.

Request Type	PUT
URL	http://localhost:8080/projects/{projectId}
Path Variables	projectId
Request params	None

Request Body	<pre>{
	  "name": "New API Development",
	  "description": "Updated description for V2 API."
	}</pre>

Response code	200
Output success	The updated Project object with an ID.
Response code error	400
Output error	• Project not found: <pre>{ "error": "Project with id 99 not found."}</pre>• Empty name: <pre>{ "name": "Name cannot be empty."}</pre>• Description length: <pre>{ "description": "Description cannot exceed 25 characters."}</pre>

Question 2 (4 points)

Implement an end-point that retrieves all Task objects belonging to a specific Project. This end-point should filter all tasks of a specific project based on two request parameters: status and dueBefore:

1. status: Mandatory. The status to be filtered on ("pending", "doing", "completed", case-insensitive). You do not need to handle the error thrown when this parameter is not supplied.
2. dueBefore: Mandatory. A date: only tasks whose deadline is between today and before this date should be returned. So, for example, you can use this to retrieve all tasks of a project that are still in progress and need to be finished between today and the day after tomorrow.

A Project can have multiple Tasks and a Task can only belong to one Project.
The answer is a list of Tasks that meet the criteria. A Task may not contain information about the Project.

Customize DbInitializer to create multiple Tasks with different statuses and deadlines associated with different Projects.

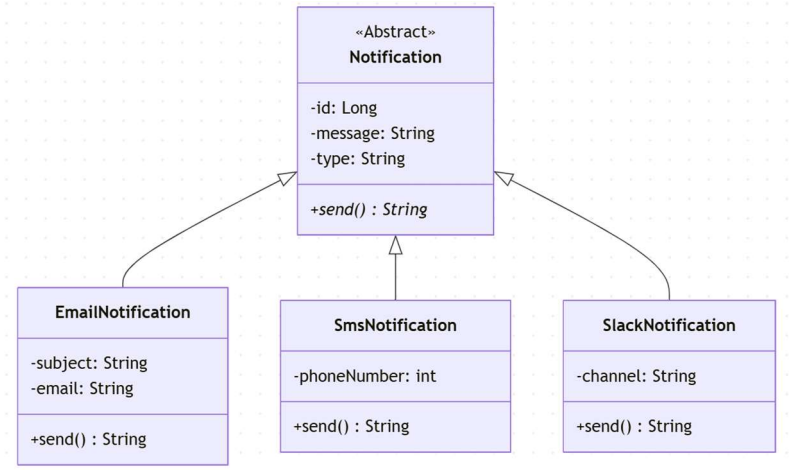
Request Type	GET
URL	http://localhost:8080/projects/{projectId}/tasks?status={status}&dueBefore={dueBefore}
Path Variables	projectId
Request params	• status: required; "pending", "doing", "completed" • dueBefore: required; e.g. "2025-09-04"
Request Body	None
Response code	200

Output success	<pre>[
	  {
	  "id": 1,
	  "title": "Setup database",
	  "status": "pending",
	  "deadline": "2025-09-02"
	  },
	  {
	  "id": 4,
	  "name": "Write tests",
	  "status": "pending",
	  "deadline": "2025-09-03"
	  }
	]</pre>

Question 3 (5 points).

In this question, we set up a notification system: creating different types of notifications (Notification). Linking to Task does not need to be done in this question.

a) Implement the domain of Notifications according to the UML diagram below:



The output of the *send* method varies by Notification type:

- EmailNotification: "Sending mail with subject '{subject}' to '{email}': {message}"
  Example: *"Sending mail with subject 'URGENT: Task Deadline' to 'john.doe@mail.com' : Deadline of task 1 is today."*
- SmsNotificaton: "Sending sms to *{phoneNumber}*: {message}"
  Example: *"Sending text message to* 32493879865: Deadline of task 1 is today."
- SlackNotification: "Posting to Slack channel *#{channel}*: {message}"
  Example: *"Posting to Slack channel #alerts: Deadline of task 1 is today."*

b) Using DBInitializer, create at least one instance of each Notification type.

c) Provide an end-point to retrieve all Notifications by type. The response is a list of Notification objects.

Request Type	GET
URL	http://localhost:8080/notifications/{type}
Path Variables	type: String; values: email, sms or slack
Request params	None
Request Body	None
Response code	200

Output success	[
	  {
	  "id": 1,
	  "message": "Deadline of task 1 is today",
	  "type": "slack",
	  "channel": "alerts"
	  },
	  {
	  "id": 2,
	  "message": "Task 2 is marked as completed.",
	  "type": "slack",
	  "channel": "news"
	  },
	]

Question 4 (2 points)

a) Write the unit test below for domain class Project. Place the test in the appropriate package.

- `givenProjectWithInvalidName_whenValidated_thenConstraintViolation`

b) Write the unit test below for the service class Task Service . Make use of mocks here.

- `givenValidProjectId_whenFindTasksByStatusWithDueDate_thenTasksBetweenTodayAndDueDateAreReturned`

Question 5 (2 points)

Write an integration test for the end-point of question 1: updating a Project:

- `givenExistingProject_whenUpdateProjectWithValidData_thenProjectIsUpdated`