

# Project

Book

1.0

Version

G. Jongen, J. Pieck, E. Steegmans, B. Van Impe

Authors

## CONTENT

---

<b>1 INTRODUCTION.....</b>	<b>3</b>
<b>2 BEFORE YOU START.....</b>	<b>5</b>
2.1 CREATE A TEAM .....	5
2.2 CREATE REPO IN GITHUB CLASSROOM ASSIGNMENT .....	5
<b>3 ASKING THE LIBRARY FOR DATA (GET REQUEST) .....</b>	<b>6</b>
3.1 CREATE A SPRING BOOT PROJECT AND PUSH IT TO THE GITHUB CLASSROOM .....	6
3.2 CREATE A CLASS DIAGRAM AND IMPLEMENT IT.....	6
3.3 MAKE A REST CONTROLLER.....	7
3.4 FRONT END .....	7
<b>4 STORING DATA IN DATABASE.....</b>	<b>8</b>
<b>5 VALIDATION.....</b>	<b>10</b>
<b>6 MERGE PROJECT “BOOK” WITH LABO “USER” .....</b>	<b>11</b>
<b>7 CRUD .....</b>	<b>14</b>
<b>8 TEST DRIVEN DEVELOPMENT .....</b>	<b>15</b>

# 1 Introduction

---

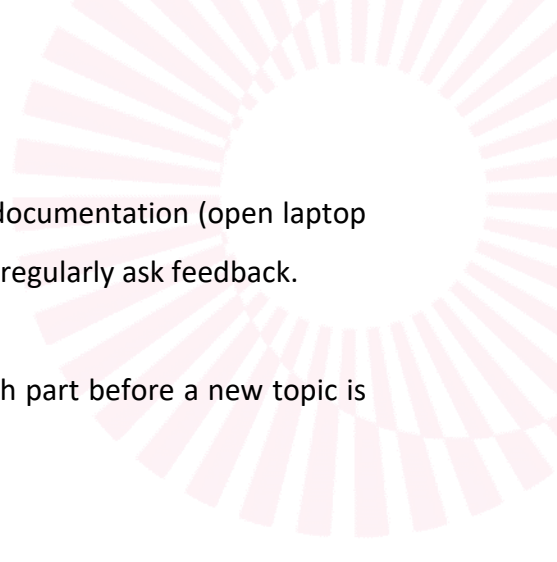
OPO	Back-End Development	
Place in the OPO	Overall project	<b>Team (2 students)</b>
Documentation	Learning material in Toledo Labs of the User application	
Prerequisites	<ul style="list-style-type: none"><li>• Part 1: none</li><li>• Part 2: both Book app (part 1) and User app (labs) are finished</li></ul>	
Learning Goals	<ul style="list-style-type: none"><li>• You can implement an application with all technologies presented in this OPO</li><li>• You can analyze and implement user stories</li><li>• You can work in team (2 students)</li><li>• You can manage feedback</li><li>• You are mastering the concepts of OO programming learned in the course</li><li>• You can write clean code according to design principles introduced in the course</li><li>• You can use GitHub as version control system</li><li>• You can debug your code on an efficient way using the debugging tool of the editor</li></ul>	
Product Goals	Part 1: You create the back-end for a book registration service. Books can be added, deleted, updated, searched.... Part 2: You create the back-end for a library service. Users can borrow books, bring them back, ...	

The project consists of 2 modules

1. Book: An application to register, update, search, ... books of a library
2. Library: An application to manage library facilities: a registered user can borrow registered books, bring them back etc.

Module 1 (Book) runs parallel to the user-labs. In the labs (user) you exercise specific competences in addition to the theory lessons and demo code. The book project is more open.

In module 2, the User application of the labs meets the Book application of module 1. That will be impossible if you haven't completed both.



During the exam, it is allowed to use your self-written code as documentation (open laptop exam). Therefore, it is important to complete the project and to regularly ask feedback.

The project is divided in several parts. Ideally you complete each part before a new topic is introduced.

## 2 Before you start

---

### 2.1 Create a team

A team consists of 2 students of the same class group.

Register your team at Toledo.

In case you have a very good reason to get an exception to pairing, contact your lector.

We expect that team members work together and preferably simultaneously (pair programming). If the cooperation becomes impossible or too difficult, contact your lector.

We will try to find a solution.

All students are expected to complete all tasks, even when they work individually.

### 2.2 Create repo in GitHub Classroom Assignment

You are expected to share your code with your teammate and lector via repo created as assignment in our GitHub classroom.

Link to the assignment: <https://classroom.github.com/a/qpVFIsF9>

See manual “Create and share a project”, part 1.

Name of your group is the same as your Toledo-team.

### 3 Asking the library for data (get request)

OPO	Back-End Development	
Place in the OPO	Java Basics	Team
Documentation	<ul style="list-style-type: none"><li>• Demo</li><li>• Course Material</li></ul>	
Prerequisites	<ul style="list-style-type: none"><li>• You finished user labo 1 – 3</li></ul>	
Learning Goals	<ul style="list-style-type: none"><li>• domain – service – rest API with GET request</li><li>• You can create a Spring Boot Project from scratch</li><li>• You can analyze a user story and design a corresponding class diagram</li><li>• You can write all necessary code to implement the back end as a Spring Boot project of a given user story</li><li>• You can link your project to a given front end</li><li>• You can share a project on GitHub and collaborate on it</li></ul>	
Product Goals	<ul style="list-style-type: none"><li>• A Spring Boot Project with name as described in the manual</li><li>• All given tests run (<i>only minor adjustments are allowed, e.g. port number of local host</i>)</li><li>• You have implemented story 1-3 (including front-end)</li><li>• Your project is pushed to our GitHub classroom</li></ul>	

#### 3.1 Create a Spring Boot Project and push it to the GitHub classroom

We created an assignment in a GitHub classroom. Before you start coding, follow step 2-4 in the tutorial "Create and share your project" to create a (shared) Spring Boot project connected to the repo you created via the GitHub classroom.

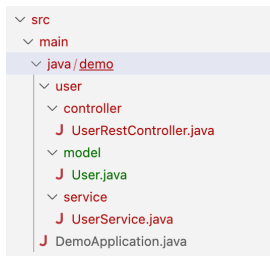
#### 3.2 Create a class diagram and implement it

Read story 1-3. Create a class diagram necessary for their implementation.

You already know that you have to put the Java classes in the folder `"src/main/java/ownPath"` because we are working with the Spring Boot Framework. In Java, there are conventions to further structure files. We choose the following structure:

- Parent package `"book"` for the book project
- Subpackage `"model"` for the domain class `"Book"`
- `"service"` for `"BookService"`
- `"controller"` for the restcontroller

Create those packages in VS Code (as "New Folder"). Remark that the class "DemoApplication" with main method is in the root package "demo" and not in a subpackage.



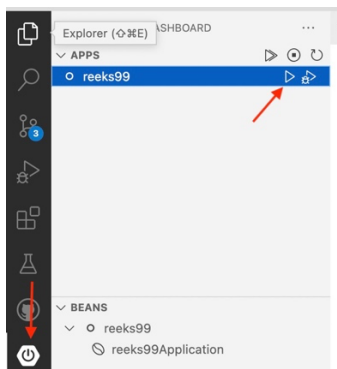
Implement all classes and methods of your class diagram. Run the given test classes until they pass. You can find the test classes on Toledo.

### 3.3 Make a Rest Controller

Write the necessary code so that the url's of the stories work.

Test them in Thunder Client. Check the necessary parameters and the obtained responses.

Before you can test in Thunder Client, you must start your application:



### 3.4 Front End

Display your library in the browser according to the stories. Use the front end <https://github.com/UCLLBackEndDevelopment/FrontEndBook/releases/tag/1.0> with basic functionalities. Make them work. It is not allowed to change url's in JS.

## 4 Storing data in Database

---

Up to now, all books are stored in the service class itself. Every time you restart the project, all books are deleted. Therefore, it is more convenient to store all data in a database.

You've already noticed at the user lab that you're going to have to thoroughly rework your code. Therefore, you should create a version or release in your GitHub project of your last commit. Alternatively, you can start a new branch. That way, you can later easily find the point where stories 1-3 worked with in-memory data (ArrayList).

This part of the project consists of two components.

1. Refactor the code you created so far so that the data is stored in a database. You will find the reworked user stories (denoted with v2) attached to this assignment. Note that in some stories the output of the rest controller has changed.
2. Add extra functionalities (story 4-6)

Download new resources (front-end, test classes and stories) from

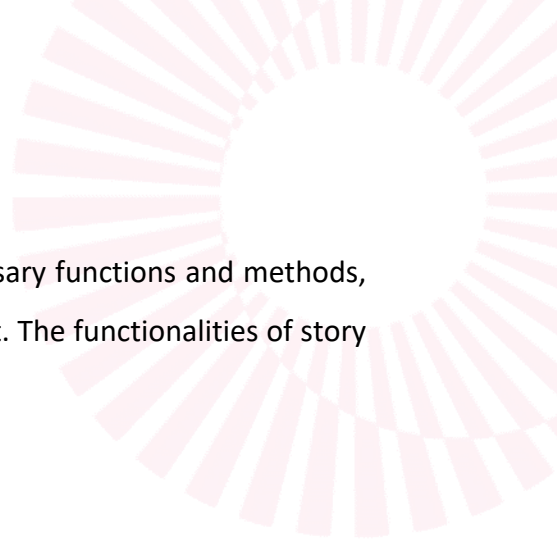
<https://github.com/UCLLBackEndDevelopment/startercode/tree/main/Project/deel-2> .

They replace the previous stories, test classes, front-end.

For component 1, follow the same steps as for the lab User.

- Add the necessary Maven dependencies
- Prepare your project for the h2-database: update the file application.properties
- Refactor your Book class to match v2 of the stories 1-3. Comment or remove the "old" BookTest and download the new one. Refactor your code until all tests pass.
- Refactor the class BookService such that all data will be stored in and queried from the database (put the repository in package "repo"). Make sure all methods have the correct return value. These may be different from the previous version. Test with the new BookServiceTest.
- Refactor the restcontroller where necessary.
- Test first with your Thunder Client collection and next with the new front-end.





For component 2, complete the class diagram. Write the necessary functions and methods, run the tests until they pass. Test the url's first in Thunder Client. The functionalities of story 6 are not used in the given front-end.

## 5 Validation

---

Add validation to the Book project as described in stories 3 (v3), 4 (v2), 5 (v2) and 6 (v2). Implement it as shown in class, user demo and presentations on Toledo.

Download new resources (front-end, test classes and stories) from

<https://github.com/UCLLBackEndDevelopment/startercode/tree/main/Project/deel-2b>

They replace the previous stories, test classes, front-end.

In the previous assignments, you were not allowed to adjust the tests. From now on, you are allowed to, albeit limited and only the `bookServiceTest`. Indeed, the mocking in the `serviceTests` explicitly depends on the way you query the repo e.g.

```
when(bookRepository.findBooksByTitleAndPrice("xyz", 50)).thenReturn(anExtraBook);
```

But there are different ways to query the repo that still give the same result, e.g. `findBooksByTitleAndPrice()` vs `findBooksByPriceAndTitle()`. If that is the case, you may refactor the test to your own situation.

Test all url's in your Thunder Client collection.

## 6 Merge project “Book” with labo “User”

---

Until now, the "book" project and the "user" lab were two separate applications. As promised at the beginning of this document, now comes the time to bring them together.

### Preparations

Make the following preparations

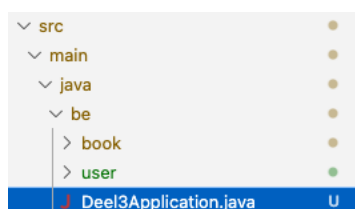
- Carefully test the code of both projects. Make sure all tests color green. Run your request-collection in Thunder Client one more time and assure yourself that everything works as requested.
- Push the code of both projects to their respective GitHub repo. Label those commits (release) e.g. with v2. That way, if it all goes wrong, you can be sure you can go back to a point where your code was still fine.
- If you are working in pairs:
  - Select one user-project that you will continue with.
  - Decide on whose laptop you will perform the merge. Don't do it partly on one, partly on the other laptop because that's guaranteed to cause major merge conflicts.

### Workflow

In what follows, we start from project "Book" and add the code of labo “User” to this project. After that, we will work only with the (extended) project Book and leave the labo “User” for what it is.

Start now and open project Book.

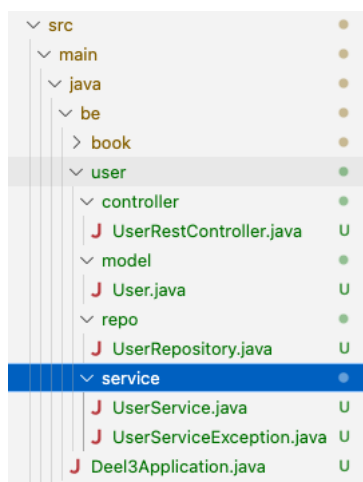
### Root Directory



Your project's basic document structure may be different from ours.

In what follows, we will call the directory where the java-file with annotation `@SpringBootApplication` (and the main method) resides the "root directory". In the screenshot above the java-file is called "Deel3Application.java" and "main/java/be" is the root directory. If your project looks different, e.g. "BookApplication.java" in "main/java/be/series1-2-3", you may leave it as it is. If the main method is now in the package "book", you must drag it one level higher, otherwise your application will not recognize the classes in the package user.

### Create packages for user classes and import them into the "Book" project



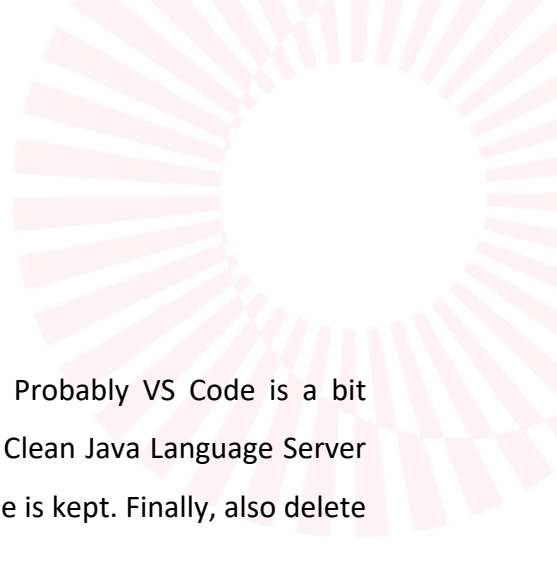
Create in your root directory the package "user" and subpackages "controller", "model", "repo" and "service".

Import (copy) the corresponding user classes to the appropriate package. Refactor the classes if necessary (e.g. initial package declaration, imports).

Rename the two ServiceException classes to respectively UserServiceException and BookServiceException.

### Import Test Classes for User





Do the same for the user test classes in the root test directory.

### **Clean your Java Workspace**

You made a lot of fundamental adjustments to your project. Probably VS Code is a bit confused. Therefore, clear all of VS Code's cache: choose "F1 > Clean Java Language Server Workspace". Also delete the "data" folder where the h2 database is kept. Finally, also delete the folder "target" containing the .class files.

### **Run your project**

Fingers crossed and run your project. All tests should pass. Run the request-collections "User" and "Book" in Thunder Client. No errors 500 or 405 should appear.

Test also both front-ends.

### **Commit to GitHub**

Commit the code to GitHub. Your teammate can pull the code to their own computer and test the functionalities.

## 7 CRUD

---

In computer programming, create, read, update, and delete (often referred to via the acronym CRUD) are the four basic operations of persistent storage. CRUD is also sometimes used to describe user interface conventions that facilitate viewing, searching, and changing information using computer-based forms and reports.

[https://en.wikipedia.org/wiki/Create,\\_read,\\_update\\_and\\_delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)

As for the project “Book”, we have already coded “create”, “read” and “delete”. We are left with the update functionality. Implement story 7 so that the book app includes all CRUD operations.

Don’t forget to complete the class diagram.

There is no front-end to test this functionality. Create an extra request in your Thunder Client collection.

## 8 Test Driven Development

Testing is important to keep your code bug-free, both when developing and when refactoring your code.

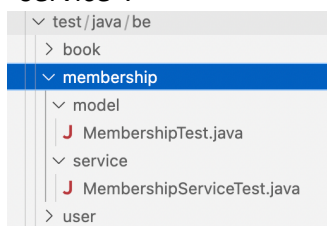
Until now, you got the tests with the starter code. From now on, you have to write your own tests.

To practice this, create a new class "Membership". Write the tests first, then the code. That way, you first think about what you need to code: what is the purpose, what special cases are there, what does the API expect from you, ...

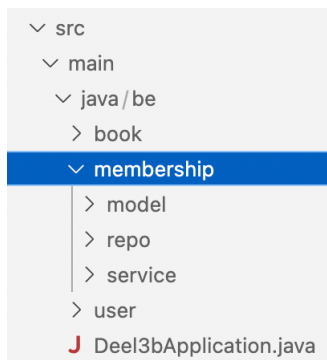
The "Membership" class stands alone for now. You won't get an API in this assignment either. So you should not write a REST controller. In the next part, we will connect the different classes of the project ("User", "Book" and "Membership") with each other and add API.

### Workflow

- Open the (merged) project.
- Test code: Create a new package "membership". Create the packages "model" and "service".



- Main code: Create the same packages here. Also add the package 'repo'.



- Read story 8.
  - Add the new class to your class diagram.
  - Write the tests as requested under "Technical Requirements". Then deploy the code so that the tests turn green.
- Complete the other stories.

If a new story requires adjustments to the code you already wrote, all previous tests should continue to color green without adjustments.