

9 Relationships

In this exercise we extend our application to a real library application where users can become member of the library and rent books.

Therefore, we must create relationship between the classes you already implemented: User, Book and Membership and a new class Profile.

The implementation needs a lot of refactoring. Create a release on GitHub so that afterwards you can easily revert to a working version before refactoring.

The first two stories use the class User and a new (fourth) class Profile. The other stories also use Book and Membership. So, it is important to complete all these classes before you go on.

As the stories progress, the User and Book classes are extended. In the API of a given story, we list only those fields that you should have implemented at that time. At the end, when all stories are finished, the information of User and Book in *all* stories 1-22 should contain the following fields:

User

- id, name, age, email, password,
- profile with fields {id, bio, place},
- list of memberships with fields {id, startDate, endDate, type, freeLoans},
- list of rented books with fields {id, title, numberInStock, price, inColor, availableCopies}

Book

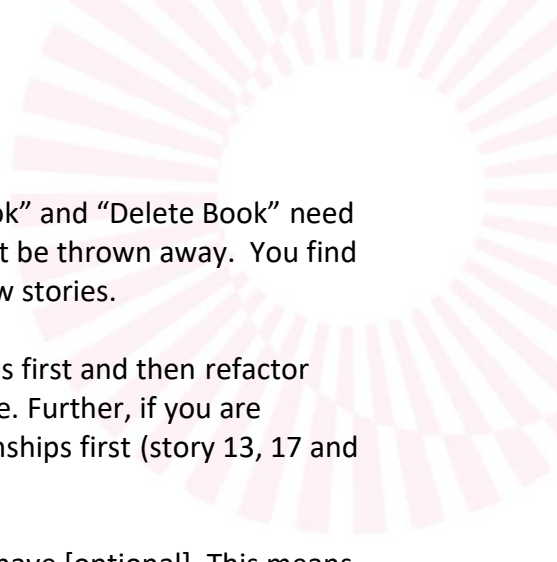
- id, title, numberInStock, price, inColor, availableCopies

Complete your Class Diagram.

Work test first: For each functionality, write at least one happy scenario and the necessary unhappy cases.

Test the requests in Thunder Client. A functionality only “works” if it complies with the request as described in the API: the URL must be exactly the same (incl parameter naming), the fields in the request body must match and the output fields (incl error messages) must be identically the same as requested.

When a user is registered, he has neither profile nor membership. So, the story concerning creating user remains unchanged. The story “Update User” still holds since the update of profile and membership have their own API. The story “Delete User” needs some modification as you will read in the acceptance criteria of the new stories.



In case of Book, “Create Book” remains unchanged. “Update Book” and “Delete Book” need some modifications since a (copy of a) book that is rented cannot be thrown away. You find more information about this in the acceptance criteria of the new stories.

We recommend implementing the stories related to relationships first and then refactor update and delete User and Book, unless a story states otherwise. Further, if you are pressed for time, it is a good idea to focus on making the relationships first (story 13, 17 and 20) and ignore the unhappy cases for a while.

For some (parts of) stories you will notice that some validations have [optional]. This means that you can choose whether to implement (that part of) the story. Usually those validations are more challenging and we see that as an extra part of the learning.

A few tips:

- Add the necessary requests to your Thunder Client Collection.
- Run regular (existing) test classes and requests and check that you haven't broken the code you wrote before.

9.1 One to One: A User has a Profile

Make it possible that a user creates a profile. Implement story 13 and 14. Story 15 and 16 are optional.

9.2 One to Many: A User has Memberships

A user can only rent books from the library when he has a valid membership. All memberships are kept, including those that have expired, so that the user's background can be traced. Implement story 17-19.

9.3 Many to Many: A User can rent a Book

A user with valid membership can rent books from the library. The library mostly has several copies of the same book (field “numberInStock”). Therefore, a book can be rented by several users. Implement story 20-22.