**Web Applications Project Assignment**

***Part II, Frontend***

Table of Contents

[Project Description 2](#_Toc51943559)

[Public Part 2](#_Toc51943560)

[Homepage 2](#_Toc51943561)

[Register 2](#_Toc51943562)

[Login 3](#_Toc51943563)

[Logout 3](#_Toc51943564)

[Private Part 3](#_Toc51943565)

[Retrieve all books 3](#_Toc51943566)

[View individual book 3](#_Toc51943567)

[Borrow a book 4](#_Toc51943568)

[Return a book 4](#_Toc51943569)

[Read book reviews 4](#_Toc51943570)

[Create book review 4](#_Toc51943571)

[Update book review 4](#_Toc51943572)

[Delete book review 5](#_Toc51943573)

[Rate book 5](#_Toc51943574)

[Like reviews 5](#_Toc51943575)

[Reading Points/Gamification 5](#_Toc51943576)

[Administration Part 6](#_Toc51943577)

[CRUD any books/reviews 6](#_Toc51943578)

[Ban users 6](#_Toc51943579)

[Delete users 6](#_Toc51943580)

[General Requirements 6](#_Toc51943581)

[Frontend Requirements 6](#_Toc51943582)

[Teamwork Guidelines 7](#_Toc51943583)

[Projects Defenses 7](#_Toc51943584)

[Give Feedback about Your Teammate 7](#_Toc51943585)

# Project Description

Design and implement the **Frontend** of the new **Telerik Academy Library System**, where the users can borrow and return books, rate them, write a review about a book they have borrowed, read all the reviews about a book and rate the reviews.

Functional Requirements

Each requirement is categorized in one of three categories – **must**, **should** or **could**.

* “Must” requirements have the highest priority and should be addressed first.
* “Should” requirements have medium priority and should be addressed after all or most must requirements have been implemented and tested.
* “Could” requirements are best left for the last.

The application should have:

* **public part** (accessible without authentication)
* **private part** (available for registered users)
* **administration part** (available for admin users only)

Any additional features are welcome if you have covered all the listed requirements below.

## Public Part

The **public part** of your library system should be **accessible** **without authentication**.

This section **must** support the following functionalities:

### Homepage

The homepage must be the landing page for the library where the user can get acquainted with the system in some way along with corresponding functionality for user register and user login. The homepage is the face of your system. It should have something engaging for the user (public books, video, carousel widget, etc.) and should show some kind of navigation through the application (navbar, side menu, etc.) as well as points to the register/login functionalities or pages.

### Register

* **Requirements**

Create a POST request to register a user.

* **Description**

The register functionality may be done on a separate route **or** within a widget (like a modal). It should contain the desired fields with the desired validations that can tell the user immediately that his field is invalid and a button that will make a request to the API with the user’s input. After the response you can redirect or show a notification to the user.

### Login

* **Requirements**

Create a POST request to obtain a token and start a client-side session.

* **Description**

The login functionality may be done on a separate route or within a widget (like modal). It should contain the desired fields with the desired validations that can tell the user immediately that his field is invalid and a button that will make a request to the API with the user’s input. If the server returns a successful response with token, you should save the token in the local/session/cookie storage and use it to access the private API resources. You can redirect or show a notification to the user.

### Logout

* **Requirements**  
  Remove the token from the storage and (if such endpoint **exists**) communicate to the backend to **invalidate** the token.
* **Description**

The logout functionality may be done with just a button that sends a request (if such endpoint **exists**) to the API and on successful response deletes the token from the storage. You can client-side redirect or show a notification to the user.

## Private Part

The **private part** of your library system should be **accessible** **for registered users**.

This section **must** support the following functionality:

### Retrieve all books

* **Requirements**

Create a functionality that display all books from the backend.

**Should:**

* Provide input to **filter** books.
* Provide options for **pagination** or infinite scroll.
* Provide functionality to **sort** the books by **title**.
* **Description**

The view all books functionality should make a call to the API and visualize the returned books in some way. You can add some additional features like client-side pagination, filtering, or sorting. You can display a “borrow” button on books that are available (not borrowed and not unlisted).

### View individual book

* **Requirements**

GET book by id or other unique property and display on the client.

* **Description**

The view individual book functionality should make a call to the API and visualize the returned book in some way. You can either visualize it in a new view or as a modal as part of the ‘all books’ page

### Borrow a book

* **Requirements**

POST a ‘borrow-a-book’ request to the backend.

* **Description**

The borrow a book functionality should make a call to the API and visualize the returned book in some way.

You may decide to display a button on each book that is borrow-able or some other way for the user to borrow a book.

### Return a book

* **Requirements**

POST a ‘return-a-book’ request to the backend.

* **Description**

The return a book functionality should make a call to the API and visualize the returned book in some way.

You may visualize a button on each book that is borrowed by the current user with the option to return that book.

### Read book reviews

* **Requirements**

GET reviews for a certain book.

* **Description**

The ‘view individual book’ view should be extended to feature the book’s reviews. You may load them *on demand* or *on page load.*

### Create book review

* **Requirements**

POST a new review to the server.

* **Description**

The create book review functionality can be done in the ‘view individual book’ view or in the ‘view all books’ view with a button somewhere in the book. The feature should enable the user to create a review for the desired book with a form that contains the review’s fields with the desired validations that can tell the user immediately that his field is invalid and a button that will make a request to the API with the review’s data. You can add some additional features that will allow the user to make the font of the text bold/italic or create a hyperlink. On response you can redirect or show a notification to the user.

### Update book review

* **Requirements**

PUT a review to the server.

* **Description**

The update book review functionality can be done in the ‘view individual book’ view or in the ‘view all books’ view (where you show the reviews) with a button somewhere in the review. The feature should enable the user to edit the review’s content with a form that contains the desired fields with the desired validations that can tell the user immediately that his field is invalid and a button that will make a request to the API with the updated review’s data. On response you can redirect or show a notification to the user.

Make sure that the functionality to update a review is visible to the current user only if the review belongs to them!

### Delete book review

* **Requirements**

DELETE a review.

* **Description**

The delete book review functionality can be done in the ‘view individual book’ view or in the ‘view all books’ view (where you show the reviews) with a button somewhere in the book. You can consider showing a popup to the user to confirm if he really wants to delete his review. The delete review feature should make a request to the API and on response you can redirect or show a notification.

Make sure that the functionality to delete a review is visible to the current user only if the review belongs to them!

This section **should** support the following functionality:

### Rate book

* **Requirements**

PUT a book rating.

* **Description**

The rate book functionality should exist on the book view, it should show if the user has rated the book or not, and it should make a call to the API when the user selects a rating. Upon successful return of the API call the overall rating of the book should be updated (the average of all ratings of the book by different users).

### Like reviews

* **Requirements**

PUT/POST/PATCH a review like/dislike.

* **Description**

The user should be able see the votes count and click a button to like/dislike a review in the view.

### Reading Points/Gamification

* **Requirements**

If the backend supports ‘reading points’ then display those for the current user.

* **Description**

The user profile should also display your reading points next to your username.

## Administration Part

The **administration part** of your library system should be **accessible** **only for admin users**.

This section **should** support the following functionality:

### CRUD any books/reviews

* **Requirements**

Use the endpoints that the backend provides for books/reviews administration.

* **Description**

The administration page should be accessible only to admins. You can display books in a grid-like interface, along with buttons to delete, update individual books and a form to create a new book.

### Ban users

* **Requirements**

PUT/POST/PATCH a ban request to the server.

* **Description**

The administrator should be able to view all users along with a menu for banning/unbanning individual users. Make sure that the view is only visible to authenticated administrators.

### Delete users

* **Requirements**

Send a DELETE request for a user.

* **Description**

Extend the users administration page to support deleting alongside banning.

# General Requirements

This section **must** support the following functionality:

* + You **must** use **Git** to keep your source code and for team collaboration.
  + You **must** use **Git Kanban** for project management.
* You **must** use **ESLint** to write consistently styled code.
* You **must** follow **OOP** **principles**and other principles you**have heard of**such as**Single Responsibility**.
* You **must** use correct naming and write clean, **self-documenting code**.

# Frontend Requirements

This section **must** support the following functionality:

* + You **must** use **React**.
  + You **must** use the **Hooks** **API.**
  + You **must** use the **React Router**
  + You **must** apply proper **client-side** **data validation.** All user input should be validated to mitigate the number of unnecessary requests and to provide immediate feedback.
  + You **must** apply proper **error handling.**
  + You **must** have usable and somewhat responsive UI.

# Teamwork Guidelines

Refer to the teamwork guidelines document found along with the project requirements.

### Projects Defenses

When you finish both parts of the web application you will present it.

### Give Feedback about Your Teammate

You will be invited to provide feedback about all your teammates, their attitude about this project, their technical skills, their team working skills, their contributions to the project, etc. **The feedback is important part of the project evaluation so take it seriously and be honest.**