




Web Application Report

Ida Åberg, Erik Dreifeldt, Madeleine Larsson, Cecilia Nordén Elgh

March 2025

BookShelf				
<div>Search</div> <div>Title</div> <div>Add Book</div> <div>Profile</div>				
Cover	Title	Author	Status	Rating
	Frankenstein	Mary Shelly	Have Read	4
	Treasure Island	Robert Louis Stevenson	Reading	N/A
	The Great Gatsby	F.Scott Fitzgerald	Want to Read	N/A

localhost:5173/edit/4

Contents

1	Introduction	1
2	User Manual	2
2.1	How to install	2
2.2	How to use	3
3	Design	9
3.1	Descriptions of components	9
3.1.1	Components	9
3.1.2	Pages	13
3.2	Specification of API	15
3.2.1	Book	15
3.2.2	User	18
3.3	Er-diagram	19
4	Responsibilities	21
4.1	Everybody	21
4.2	Ida	21
4.3	Erik	22
4.4	Cecilia	22
4.5	Madeleine	22

1 Introduction

We have created a web application for keeping track of your books in a digital book shelf. The user can add the books that they have read, are currently reading and want to read. The books are then displayed in the users bookshelf, and can be edited upon request. When adding a book the user can insert the title, author, status (have read, reading, want to read), rating (optional) and comments (optional). The user can also filter their books digitally.

The app has a simple and intuitive design that makes it easy for anyone to use. The goal with the application is to help readers track their progress and encourage healthy reading habits. It aims to help readers reflect on their reading experiences by allowing notes and ratings, as well as a simple and structured way to keep track of all their books.

Github link

Book cover in screen shots:

`Image by freepik `

2 User Manual

This is a user manual for our BookShelf application.

2.1 How to install

In order to use the app you need to install the following:

- Beekeeper Studio
- Docker Desktop
- Your preferred IDE

Start by cloning the GitHub repository.

In the root file, add an `.env` file. This is where you put secret passwords and keys. Add **SESSION_SECRET** and **DB_PASSWORD** and choose appropriate passwords for both. Make sure this file is included in the `.gitignore`.

To set up the database you need to create a docker using the following command, replace the `<password>` with the same password you created in the `.env` file:

```
docker run --env POSTGRES_USER=app_db_user \
  --env POSTGRES_PASSWORD=<password> \
  --publish 54321:5432 --name web_apps_db \
  --detach postgres:17
```

Make sure it is running in Docker Desktop or with this command:

```
docker ps
```

To connect to the database, press import from URL in Beekeeper Studio and paste in the following, again replace the `<password>`:

```
postgres://app_db_user:<password>@localhost:54321
```

fill in the missing fields and press connect.

If you are having issues with setting up the database you can use the in memory database instead. To do this you need to add **NODE_ENV=test** in your `.env` file.

Before running the web application you need to install all the dependencies using the following command:

```
npm install
```

in both the **client** and **server** directory.

Finally, run this command in both the **client** and **server** directory (using two different terminals):

```
npm run dev
```

click the localhost link that is displayed in the client terminal to open the webapp.

2.2 How to use

When opening the application, the user is first met with a Log-In screen. On this page, the user has the option to register a new user or log in using an existing one. If you try to log in using a username that hasn't been registered or the wrong password, the application will let you know by giving you an error message as seen in figure 1.

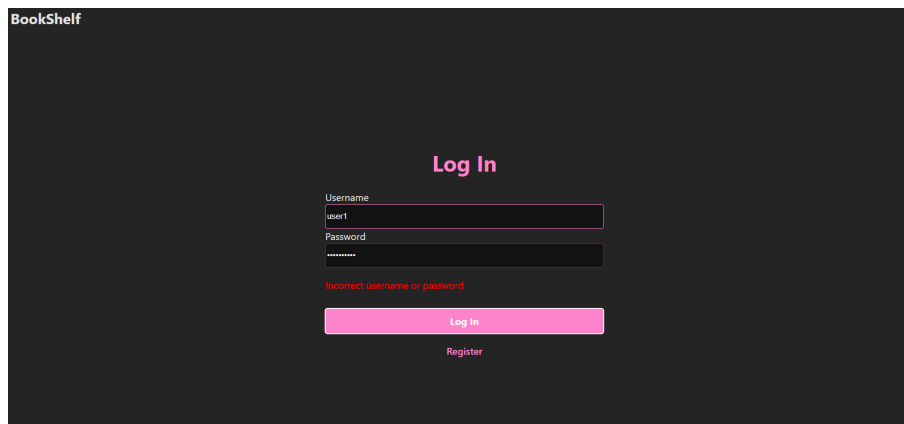


Figure 1: A failed login attempt

If you click on the register button it will take the user to a register page, similar to the login page, where you have to fill out a username and password to be able to sign up. If you try to register a user with an already taken username or have a password that is shorter than 8 characters, the application will not let you register as shown in figure 2. If you have a unique username and long password, the register button takes you back to the login page. If you change your mind and don't want to sign up, you can also use the cancel button to get back to the login page.

BookShelf

Register

Username
user

Username is already taken

Password

Password must be at least 8 characters long

Register

Cancel

Figure 2: A failed register attempt

Once registered and logged in, you will be met by your own personal bookshelf figure 3. If you log in for the first time, it will be empty until you add books. From here you can see all your added books with title, author, status of the book and rating. You can also see a bookcover, unfortunately the cover feature has not been implemented and it's just a placeholder. In the menu bar there is a search field that let's you filter your books with the help of the drop-down next to it. The user can filter using title, author, status and rating as demonstrated in Figure 4. There is also a Add Book button that let's you add your books as well as a Profile button that redirects you to your profile page.

BookShelf				
<div> <input type="text" value="Search"/> <div>Title</div> <div>Add Book</div> <div>Profile</div> </div>				
Cover	Title	Author	Status	Rating
	Frankenstein	Mary Shelly	Have Read	4
	Treasure Island	Robert Louis Stevenson	Reading	N/A
	The Great Gatsby	F.Scott Fitzgerald	Want to Read	N/A

Figure 3: Your personal bookshelf with books added

The screenshot shows the BookShelf application interface. At the top, there is a header with the 'BookShelf' logo, a search bar containing the letter 'F', and buttons for 'Author', 'Add Book', and 'Profile'. Below the header is a table with the following columns: Cover, Title, Author, Status, and Rating. The table contains one row for the book 'The Great Gatsby' by F. Scott Fitzgerald, with a status of 'Want to Read' and a rating of 'N/A'. The book cover image shows a silhouette of a head with the word 'POETRY' above it.


Cover	Title	Author	Status	Rating
	The Great Gatsby	F.Scott Fitzgerald	Want to Read	N/A

Figure 4: Bookshelf filtered by authors on F

If you click the Add Book button you will see a page where you can fill in the title, author, status, rating as well as adding a comment about the book as in figure 5. The title and author field has to be filled in for a book to be created but the rating and comment field can be left empty. If you don't want to add a book you can go back to the bookshelf page by using the cancel button or the BookShelf logo in the top left corner. Once you are satisfied with your book details you click the Submit button, and it takes you back to your bookshelf where you can view your new book. Adding a book will also trigger a message at the bookshelf page that lets you know that your book has been successfully added.

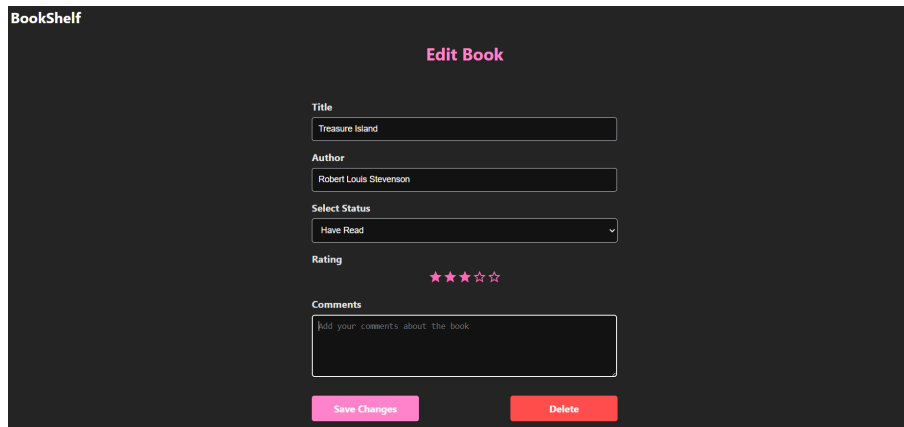
The screenshot shows the 'Add Book' page in the BookShelf application. The page has a dark background with a pink header. The title 'Add Book' is centered at the top. Below the title are several form fields: 'Title' (filled with 'Frankenstein'), 'Author' (filled with 'Mary Shelly'), 'Select Status' (a dropdown menu with 'Have Read' selected), 'Rating' (a row of five stars, with the first four filled in pink), and 'Comments' (a text area filled with 'Very good book!'). At the bottom of the form are two buttons: 'Submit' (pink) and 'Cancel' (pink).

Figure 5: Add book page with fields filled in

You can also edit your books. If for example, you've added a book with status reading and want to change it to have read or have made a typo. All you have to do is click on the book you want to edit in your bookshelf and it will take you to the Edit Book page like in figure 6. All the information about your book is visible and you can change it however you like. If you don't want to make any changes you can go back to the bookshelf page by using the BookShelf logo

in the top left corner. Once you're done with your changes you click the Save Changes button and it will redirect you back to the bookshelf page where you can view your new changes like in figure 7. It will also give you a message to show that your changes have been saved.

From the Edit Book page you can also delete books you no longer want in your library. To do this just click the delete button. A pop-up will then be shown where you are asked if you are sure you want to delete the book as a safety net as shown in figure 8. To delete you press the delete button but you can also go back by pressing the cancel button. If you decide to delete your book you will come back to the bookshelf page where it can no longer be seen in your library figure 9. You will also get a confirmation message that your book has been successfully deleted.



BookShelf

Edit Book

Title
Treasure Island

Author
Robert Louis Stevenson

Select Status
Have Read

Rating
★★★★☆

Comments
Add your comments about the book

Save Changes Delete

Figure 6: Edit book page with the details previously filled in from Add Book page




BookShelf				
<input type="text" value="Search"/> <input type="button" value="Title"/> <input type="button" value="Add Book"/> <input type="button" value="Profile"/>				
Cover	Title	Author	Status	Rating
	Frankenstein	Mary Shelly	Have Read	4
	Treasure Island	Robert Louis Stevenson	Have Read	3
	The Great Gatsby	F.Scott Fitzgerald	Want to Read	N/A

Figure 7: Bookshelf updated after edits have been made on a book, compare to figure 3

Edit Book

Title

Author

Select Status

Rating

Comments

Are you sure you want to delete this book?

Figure 8: The delete book safety net



BookShelf				
<div> <input type="text" value="Search"/> <div> <div>Title</div> <div>-</div> <div>Add Book</div> </div> <div>Profile</div> </div>				
Cover	Title	Author	Status	Rating
	Frankenstein	Mary Shelly	Have Read	4
	Treasure Island	Robert Louis Stevenson	Have Read	3

Figure 9: The library after a book has been deleted

From the bookshelf page you can also reach your profile page figure 10. There you can see your username and "password". The password feature hasn't been implemented and is only a placeholder. At the profile page you can log out, which will take you back to the log in page or you can go back to your bookshelf using the back button or the BookShelf logo.

BookShelf

Profile

Username: user1

Password: *****

Log Out

Back

Figure 10: The profile page

3 Design

3.1 Descriptions of components

3.1.1 Components

- **AuthForm**

The purpose of **AuthForm** is to be a blueprint for the login and register pages. It handles user input for username and password, validates errors, and submits the data to the provided authentication function.

Props

- *title* (string): The title displayed at the top of the form.
- *buttonText* (string): The text displayed on the submit button.
- *onSubmit* (function): A function that takes username and password and handles authentication.
- *linkText* (string): The text displayed on the link that allows navigation to another form.
- *linkTo* (string): The path to another form.
- *errors* (errors): An object containing potential error messages related to username and password input.

State

- *username* (string): Stores the current value of the username input field.
- *password* (string): Stores the current value of the password input field.

Backend Calls

- The component does not make any backend calls itself, but relies on the *onSubmit* function prop to handle authentication. Instead it is the components or pages that uses **AuthForm** that makes the call to the backend.

- **Book**

The purpose of **Book** is to display a book's information such as name, author, rating and state.

Props

- *book* (object of type *Book*): The book object containing details such as:
 - * *id* (string or number): Unique id for the book.
 - * *title* (string): The book's title.
 - * *author* (string): The author's name.

- * *state* (string): The current status of the book (Want to read, reading, have read).
- * *rating* (number or null): The book's rating, or null if no rating has been chosen.

State

- This component is stateless

Backend Calls

- This component does not make any backend calls

• BookForm

The purpose of **BookForm** is to be a blueprint for the `addBook` and `editBook` pages. It allows users to input book-information. In `editBook`, it also allows for the book to be deleted.

Props

- *initialBook* (optional, type *Book*): A pre-existing book object used when editing a book. If not provided, the form starts with default values.
- *onSubmit* (function): A function that handles form submission by saving the book's details. It returns a *Promise<void>* and is expected to interact with the backend.
- *isEditing* (optional, boolean, default: false): Determines whether the form is in editing mode. When true, the form loads *initialBook* data and includes a delete button.

State

- *book* (object of type *Book*): The book object containing details such as:
 - * *id* (string or number): Unique id for the book.
 - * *title* (string): The book's title.
 - * *author* (string): The author's name.
 - * *state* (string): The current status of the book (Want to read, reading, have read).
 - * *rating* (number or null): The book's rating, or null if no rating has been chosen.
- *popupOpen* (boolean): Controls the visibility of the delete confirmation popup.

Backend Calls

- The component does not make any backend calls itself, but relies on the *onSubmit* function prop to save book details. Instead it is the components or pages that uses **BookForm** that makes the call to the backend.

- **BookShelf**

The purpose of the **BookShelf** is to display a list of books in a structured layout. Each book is represented by the *Book*-component.

Props

- *books* (type `Book[]`): An array of book objects that the component renders.

State

- This component is stateless

Backend Calls

- This component does not directly interact with the backend.

- **DeletePopup**

The purpose of **DeletePopup** is to display a confirmation popup before deleting a book.

Props

- *bookID* (number): The ID of the book to be deleted.
- *setPopupOpen* ((*open*: boolean) → void): A function to control whether the popup is open or closed.

State

- This component is stateless, but relies on its parent to control visibility

Backend Calls

- Calls *deleteBook(bookID)*, which sends a request to remove the book from the backend.

- **Header**

The purpose of **Header** is to display the "BookShelf" title as a clickable element.

Props

- This component does not receive any props.

State

- This component is stateless.

Backend Calls

- This component does not interact directly with the backend.

- **Navbar**

The purpose of the **Navbar** is to display the navigation bar for the application. It includes the "BookShelf" logo, a search bar with filtering options, and links to add a new book and view the user profile.

Props

- *search* (string): The current search query.
- *setSearch* ((value: string) → void): A function to update the search query.
- *setFilterBy* ((filter: "title" | "author" | "state" | "rating") → void): A function to update the filter criteria.

State

- This component is stateless, but it relies on its parents to manage the search and filter states.

Backend Calls

- This component does not interact directly with the backend.

• Rating

The purpose of **Rating** is to allow users to provide rating for books, using MUI's star-rating component.

Props

- *value* (number| null): The current rating value, and is null if no rating is selected.
- *onChange* (function): A function to update the rating value when the user selects a different rating.

State

- This component is stateless

Backend Calls

- This component does not interact directly with the backend.

• SearchFilter

The purpose of **SearchFilter** is to provide a search bar and a dropdown filter for the search-result to the user.

Props

- *search* (string): The current search term provided by the user.
- *setSearch* (function): A function to update the *search* state in the parent component.
- *setFilterBy* (function): A function to update the filter criteria in the parent component.

State

- *selectedFilter* (string, default: "Filter"): Keeps track of the currently selected filter option.
- *isDropdownOpen* (boolean): Manages the dropdown state (open/closed) to visually indicate when the user interacts with it

Backend Calls

- The components does not interact directly with the backend.

- **SuccessPopup**

The pupose of **SuccessPopup** is to display a temporary success message when adding, editing or deleting a book.

Props

- *message* (string): The success message to display inside the popup.
- *onClose* (function): A callback function that gets called when the popup should be removed from the UI.

State

- *fade* (boolean): Controls the fade-in and fade-out animation effects.

Backend Calls

- This components does not interact directly with the backend.

3.1.2 Pages

- **addBook**

The purpose of **addBook** is to provide a page where users can add a new book. It renders a *BookForm* and handles the submission by calling an API function to store the book's details.

Props

- This page does not receive any props.

State

- This page is stateless.

Backend Calls

- *addBook* (title, author, state, rating, comment): Calls *addBook* from the API to add a new book to the backend when the form is submitted.

- **editBook**

The purpose of **editBook** is to provide a page where users can edit an already existing book. It fetches the books details from the backend using its ID and pre-fills a *BookForm* for the user to modify.

Props

- This page does not receive any props.

State

- **book** (Book|null): Stores the book details retrieved from the backend. Initially set to *null* until the data is fetched.

Backend Calls

- *getBookByID* (id): Fetches book details from the backend using the book ID from the URL.
- *editBook* (id, title, author, state, rating, comment): Sends an update request to modify the book details in the backend when the form is submitted.

- **home**

The purpose of *home* is to serve as the main page of the application, displaying a list of books retrieved from the backend.

Props

- This page does not receive any props.

State

- *books* (Book[]): Stores all books retrieved from the backend.
- *filteredBooks* (Book[]): Stores books filtered based on user input.
- *search* (string): Holds the search term entered by the user.
- *filteredBy* ("title"|"author"|"state"|"rating"): Determines the filter category.
- *showPopup* (boolean): Controls the visibility of the success message popup.
- *message* (string): Stores the success message to be displayed in the popup.

Backend Calls

- *getBooks* (): Fetches the list of books from the backend when the component mounts.

Additional features

- Reads *localStorage* for flags (*editSuccess*, *bookAdded*, *bookDeleted*) to display success messages when a book is modified, added or deleted.

- **login**

The purpose of *login* is to provide a user authentication page where the users can log in with their username and password. It validates input fields and attempts to authenticate the user with the backend. If successful, the user is redirected to the home page; otherwise, errors are displayed.

Props

- This page does not receive any props.

State

- *errors* (Errors): Stores validation errors for the username and password fields.

Backend Calls

- *login* (username, password): Sends login credentials to the backend for authentication.

- **profile**

The purpose of *profile* is to provide a page to display user’s profile information: their username and a placeholder for password. It also provides a log out button.

Props

- This page does not receive any props.

State

- *username* (string): Stores the username of the logged-in user. Initially set to an empty string, it is populated once the *getUsername* API call is successful.

Backend Calls

- *getUsername* (): Fetches the logged-in user’s username from the backend when the component mounts.
- *logout* (): Logs the user out by calling the backend API.

- **register**

The purpose of *register* is to provide a page where user can create a new account by providing a username and password.

Props

- This page does not receive any props.

State

- *errors* (Errors): Stores validation errors for the username and password fields.

Backend Calls

- *checkUsername* (username): Checks if the username already exists in the backend.
- *register* (username, password): Registers the new user with the backend.

3.2 Specification of API

3.2.1 Book

1. GET /book

- Description: Retrieves all books for the logged-in user.
- Request:
 - Verb: GET

- Endpoint: /book
 - Headers: Session cookie required
 - Responses:
 - 200 OK: Returns an array of books
 - 401 Unauthorized: User is not logged in
 - 500 Internal Server Error
2. GET /book/:id
- Description: Retrieves a specific book by ID.
 - Request:
 - Verb: GET
 - Endpoint: /book/:id
 - Headers: Session cookie required
 - Responses:
 - 200 OK: Returns the book object
 - 400 Bad Request: Invalid book ID format
 - 401 Unauthorized: User is not logged in
 - 404 Not Found: Book with the given ID does not exist
 - 500 Internal Server Error
3. POST /book
- Description: Adds a new book to the user's collection.
 - Request:
 - Verb: POST
 - Endpoint: /book
 - Headers: Session cookie required
 - Body (JSON):


```
{
  "title": "Book Title",
  "author": "Author Name",
  "state": "Reading",
  "rating": 4,
  "comment": "Great book!"
}
```
 - Responses:
 - 201 Created: Returns the newly created book
 - 400 Bad Request: Title, author, and state are required fields, Field 'state' must be a valid BookState, Invalid rating value
 - 401 Unauthorized: User is not logged in

- 500 Internal Server Error

4. PATCH /book/:id

- Description: Updates an existing book.
- Request:
 - Verb: PATCH
 - Endpoint: /book/:id
 - Headers: Session cookie required
 - Body (JSON):

```
{  
  "title": "Book Title",  
  "author": "Author Name",  
  "state": "Reading",  
  "rating": 4,  
  "comment": "Great book!"  
}
```
- Responses:
 - 200 OK: Book has been updated
 - 400 Bad Request: Missing id param, Field 'state' must be a BookState, id number must be a non-negative integer
 - 401 Unauthorized: User is not logged in
 - 404 Not Found: Book with given ID does not exist
 - 500 Internal Server Error

5. DELETE /book/:id

- Description: Deletes a book by ID.
- Request:
 - Verb: DELETE
 - Endpoint: /book/:id
 - Headers: Session cookie required
- Responses:
 - 200 OK: Book deleted successfully
 - 400 Bad Request: Invalid book ID
 - 401 Unauthorized: User is not logged in
 - 404 Not Found: No book with given ID
 - 500 Internal Server Error

3.2.2 User

1. POST /user

- Description: Creates a new user.
- Request:
 - Verb: POST
 - Endpoint: /user
 - Body (JSON):

```
{  
  "username": "user123",  
  "password": "securepassword"  
}
```
- Responses:
 - 201 Created: User created successfully
 - 400 Bad Request: Username already taken
 - 500 Internal Server Error

2. POST /user/login

- Description: Logs in a user.
- Request:
 - Verb: POST
 - Endpoint: /user/login
 - Body (JSON):

```
{  
  "username": "user123",  
  "password": "securepassword"  
}
```
- Responses:
 - 200 OK: Logged in
 - 401 Unauthorized: No such username or password
 - 500 Internal Server Error

3. POST /user/logout

- Description: Logs out the current user.
- Request:
 - Verb: POST
 - Endpoint: /user/login
 - Headers: Session cookie required

- Responses:
 - 200 OK: Logged out
 - 500 Internal Server Error

4. GET /user/exists

- Description: Checks if a username exists.
- Request:
 - Verb: GET
 - Endpoint: /user/exists
 - Headers: Session cookie required
 - Query Parameters: username (required)
- Responses:
 - 200 OK:


```
{
  "exists": true
}
or
{
  "exists": false
}
```
 - 400 Bad Request: Username query parameter is required
 - 500 Internal Server Error

5. GET /user

- Description: Retrieves the logged-in user's username.
- Request:
 - Verb: GET
 - Endpoint: /user
 - Headers: Session cookie required
- Responses:
 - 200 OK: Returns "username": "user123"
 - 401 Unauthorized: User not logged in
 - 500 Internal Server Error

3.3 Er-diagram

User:

PK: *id* (INT, auto-increment)

username (STRING, unique, not empty)

password (STRING, not empty)

Book:

PK: *id* (INT, auto-increment)

title (STRING, not null)

author (STRING, not null)

state (ENUM: 'Have Read', 'Want to Read', 'Reading', not null)

rating (INT, nullable, min:1, max:5)

comment (STRING, nullable)

FK: *userId* (references User.id)

The relationship between the two tables is a One-to-Many as one user can have many books and is implemented through the foreign key 'userId' in the Book table.

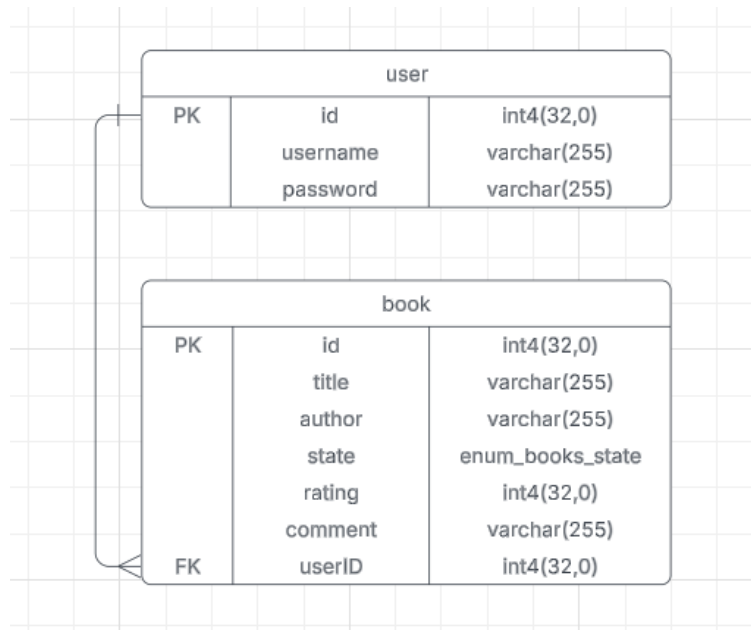


Figure 11: Entity- Relationship diagram of our database

4 Responsibilities

4.1 Everybody

We used Liveshare a lot when we sat together and worked on the labs. Things we did together include:

- First HTML and CSS mockup (lab 1)
- First basic backend (lab2)
- React components for lab3 (Book, Bookshelf, AddBook, EditBook)
- Users and sessions, styled login and register, 5 frontend tests, more backend tests (lab4)
- Modified the code to use a database, added interfaces, changed the services, db connection (lab5)
- Final project fixes, css, accessibility

4.2 Ida

- Fixed so rating and comment could be undefined, which didn't work after lab2, and then again after lab 5
- Fixed so the passwords are hashed
- Created login and register react components and connected them to api
- Fixed so we used .env file for SESSION_SECRET, DB_PASSWORD and NODE_ENV when testing
- Added the search and filter functionality
- Refactored code to be less duplicated, for example AddBook and EditBook – > BookForm, Login and Register – > AuthForm
- Did a lot of tests in both backend and frontend
- Fixed auto id incrementation issue in db
- Added validation and accessibility of the usernames and passwords in frontend and backend
- Created a profile page (With Erik)
- Added delete book functionality (With Erik)
- Added popup to delete button (With Erik)
- Wrote the How to install part of User Manual and the Specification of API, in report

4.3 Erik

- Wrote the Descriptions of Components part of the report
- Managed a lot of the CSS, especially when trying to place the different pieces correctly using flex.
- Created a profile page (with Ida)
- Added delete book functionality (With Ida)
- Added popup to delete button (With Ida)

4.4 Cecilia

- Changed the filter button into a select to fix issues with the header as well as CSS to make it look like the other buttons.
- Added path back to bookshelf from the BookShelf logo in Add Book, Edit Book and Profile page as well as CSS.
- Wrote the 'How to use' part of the User Manual, did the ER-diagram and database relations, and the introduction in the report.

4.5 Madeleine

- I participated in close to all sessions when we sat together and worked. I followed everything we did and have understood on an overall level and on a mid-level but maybe not all details in the programming. I did not create code independently since I was not the most skilled programmer.
- I tried to create a function for the user to upload a picture in the end, but did not make it work.
- I created a first draft of the final presentation that we then improved all of us in a group meeting. I did not participate in creating the final report.