Overview

You're on a mission to help the local Library to build an MVP books records system to replace the current paper records they have been using.

This is an MVP for us to gather more feedback. As a result, we have omitted a login feature as we are not sure how we are going to control usage nor what kind of authentication mechanism we will be using.

We have done out initial backlog review and came up with this prioritized backlog of User Stories. The PO has kick started the first sprint and has **prioritized User Stories 1 and 2 as one that MUST be completed**.

Please make sure that your implementations is **backed by a robust and complete set of unit tests.**

You must use Git and host your repository on GitHub or other similar services.

The repository must have a top-level README.md that briefly explains how to start up the application locally.

You must use **SpringBoot and React** as your main tech stack, **PostgreSQL** for the database.

To facilitate automated testing, please follow all specified requirements and structure.

USER STORY 1: Back End Implementation (Prioritized)

I want to have a back end set up and ready to go for Front-End's implementation.

Task 1:

You have been given a set of postgreSQL queries that will create tables in your entity in the database.

You are now tasked to implement the ORM model using JPA ORM and the provided set of queries for reference. Then link the tables based on their relevant relationship; ie. Many to Many, Single to Many etc. So that it will be a working schema.

Acceptance criteria:

- Entity should follow the provided postgreSQL definition.

Task 2:

Your next task will be to create a SQL Query that will interact and pull the specific results from the database.

Your SQL query should be able to return the following the results:

1. Top 3 rented books in the world
2. For each book, the Top 3 people with the most amount of that book rented in a specific country. (Specified countries using country codes **SG, MY, US**)
    - Sorted by descending order

Use the most optimized way you can to retrieve the results.

Acceptance criteria:

- Query should return the correct results
- Response time of the query will be compared with a baseline

Task 3:

Now, you are to implement a GET API endpoint that uses the data retrieved from Task 2 and return a JSON with the following structure:

```
[
	{
		"author": "author_value",
		"name": "book_name_value",
		"borrower": [
			"top_1_borrower_value",
			"top_2_borrower_value",
			"top_3_borrower_value"
		]
	}
]
```

The API endpoint must follow this url with the parameters: (/getTop3ReadBooks?country_code=XX)

Acceptance Criteria

- Results from the SQL query in Task 2 should be converted to conform to the specified JSON structure.
- Validation to the country_code parameter. When parameter is invalid, a code 400 with error message of {message: “invalid parameter”} should be returned.
- If there are no returned results from the query, an error message of {message: “no results”} will be returned

USER STORY 2: Front End Implementation (Prioritized)

I want to have a user facing, functional web page interface where I should be able to do and view the following:

- A list of the top 3 most checked-out books
- A list of the top 3 customers that have the highest count of checked-out books
- Filtered results in my selected country of choice.

Task 1:

You have been given a FIGMA link by the UI UX team.

(You need to create an account or log in with your google account to view the specifications.)

https://www.figma.com/file/EgUOn8iE9xfmFV9TRFjd28/Easy-Frontend-UI?node-id=63%3A36&t=9JLACpFAA71OvNeu-0

Your task is to create a functional web page based on the given FIGMA link.

It is decided that you should follow exact specifications and dimensions of the given FIGMA. Attached to some of the elements are notes from the UIUX team, and you are required to conform to the specifications they are requesting.

To facilitate automated testing, please do the following:

1. Taking that the size of the web page is a standard 1920 x 1080. Follow all of the dimensions and layout in the FIGMA. What you see in the FIGMA is exactly what you should be building.
2. Follow the notes attached into some of the elements, they contain specific class/id names.

Acceptance Criteria:

- Designs and specifications are followed exactly as what is shown in the FIGMA
- Specified Id and Class names are implemented to their respective elements.

Task 2:

After creating a web page, you are to implement the functionalities.

You are to be able to call the API (/getTop3ReadBook) to retrieve the data. Using the components you created in Task 1, display the results.

For reference, this is the JSON response when GET /getTop3ReadBook


```
[
    {
        "author": "author_value",
        "name": "book_name_value",
        "borrower": [
            "top_1_borrower_value",
            "top_2_borrower_value",
            "top_3_borrower_value"
        ]
    }
]
```


Acceptance Criteria:

- Data retrieved from the API successfully and bound to the components created in Task 1
- Implementation of Book-Toggle event handling as per commented in the FIGMA

Task 3:

You are to create a “Get Country” button to call the API (/getRandomCountry) to get a random country code. When the button is triggered, use the “country_code” as a query parameter to call /getTop3ReadBook?country_code=SG, then update the UI component with the filtered data.

The page UI should display an error message text of “No Data Found” if there is no data retrieved from the API /getTop3ReadBook . This is shown in the FIGMA’s frame 3.

For reference, the JSON format for GET /getRandomCountry is:

```
{
	"country" : {
		"full_name": "country_name_value",
		"country_code": "country_code_value"
	}
}
```

Acceptance Criteria:

- The button should conform to the one shown in the FIGMA. (button at the top left side of the screen)
- Button should call 2 APIs upon trigger. (/getRandomCountry and /getTop3ReadBook)
- Upon a successfully API call, render the new data on to the components
- Upon a failed API call, display the error message text on the page as shown in frame 3