API Documentation and Setup Instructions

Agenda:

1) API Documentation.
2) Setup Instructions.

■ API Documentation:

1) Login API:

- ❒ Suffix Path: /user/login
- ❒ Operation Type: POST
- ❒ Purpose: it enables the user who wants to login to get a JWT token that he can used later in accessing other APIs.
- ❒ Is JWT needed: No
- ❒ Input Body fields:
  - "email": the email of user who wants to login.
  - "password": the password of user who wants to login
- ❒ Output Body fields:
  - "token": the JWT for the user, generated using his email, role (got from DB), and the login date-time to ensure its uniqueness.
- ❒ Successful Status Code: 200

2) User Registration API:

- ❒ Suffix Path: /user/register
- ❒ Operation Type: POST
- ❒ Purpose: Creating a user record in the DB as follows:
  - If no JWT token provided in authorization header of the request, a user record of role 'borrower' will be created.
  - If JWT token is provided in authorization header of the request and the role decoded from JWT equals 'admin', a user record of role 'librarian' will be created.
- ❒ Input Body fields:
  - "firstName": the first name of the user to be added.
  - "lastName": the last name of the user to be added.
  - "email": the mail of the user to be added.
  - "password": the password of the user to be added.
- ❒ Output Body fields:
  - "message": sent if the user record is successfully added.
- ❒ Successful Status Code: 201
- ❒ Precautions:
  - "firstName" and "lastName" fields should be compose of only letters, hyphens, and spaces only.
  - The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.

3) User Update API:

- ❒ Suffix Path: /user/update
- ❒ Operation Type: PUT
- ❒ Purpose: Updating a user record in the DB.
- ❒ Input Body fields:
  - "firstName": the first name of the user to be added.
  - "lastName": the last name of the user to be added.
  - "password": the password of the user to be added.
- ❒ Output Body fields:
  - "message": sent if the user record is successfully updated.
- ❒ Successful Status Code: 200
- ❒ Precautions:
  - "firstName" and "lastName" fields should be compose of only letters, hyphens, and spaces only.
  - The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.
  - If you wish not to update a field, you can either send its current value or do not send this field at all.
- ❒ Notes:
  - The user email would be decoded from the JWT token provided in authorization header.
  - User Role and Email are assumed not to be editable

4) User Deletion API:

- ❒ Suffix Path: /user/delete/:emailToBeDeleted
- ❒ Operation Type: DELETE
- ❒ Purpose: deleting a user record in the DB.
- ❒ Path Parameter: the email of the user whose record to be deleted.
- ❒ Output Body fields:
  "message": sent if the user record is successfully deleted.
- ❒ Successful Status Code: 200
- ❒ Precautions:
  The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.
- ❒ Notes: Only the System admin who can use this API

5) Listing all Borrowers APIs:

- There are 2 APIs created to get all borrowers in a Paginated manner:

A) Less Recent Added Borrowers API:

❒ Suffix Path: /user/older/:pageSize?timestamp=

❒ Purpose: Getting at most {pageSize} of borrowers that are registered before certain timestamp sent as a query parameter.

B) More Recent Added Borrowers API:

❒ Suffix Path: /user/newer/:pageSize?timestamp=

❒ Purpose: Getting at most {pageSize} of borrowers that are registered after certain timestamp sent as a query parameter.

- Such 2 APIs share the following aspects:

❒ Operation Type: GET

❒ Path Parameter:

":pageSize": specifying the max. number of records to be retrieved.

❒ Query Parameter:

"timestamp": a threshold to get the borrowers registered before/after

❒ Output Body fields:

A JSON Array of Name "borrowers" where each item has the following fields:

- "first_name": borrower first name.
- "last_name": borrower last name.
- "email": borrower email.
- "registered_at": borrower regiseration date-time

❒ Successful Status Code: 200

❒ Precautions:

- The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.
- Timestamp query parameter should be provided in GMT time in ISO format like "2024-12-18T19:45:19.000Z".
- If Timestamp is not provided, it will be defaulted to the current GMT time.

❒ Notes:

- Only the System admin who can use this API.
- These APIs gets all borrowers registered in the system regardless they currently have books or not.
- The list of returned borrowers will be sorted in a descending order according to the borrower registration time.

6) Book Addition API:

- ❒ Suffix Path: /book/add
- ❒ Operation Type: POST
- ❒ Purpose: Creating a book record in the DB.
- ❒ Input Body fields:
  - "title": the title of the book to be added.
  - “author”: the author’s name of the book to be added.
  - “isbn”: the ISBN of the book to be added.
  - “quantity”: the available quantity of the book to be added.
  - "section": the library section of the book to be added.
  - “bay”: the bay number of the book to be added within the section.
  - “shelf”: the shelf number of the book to be added within the bay.
- ❒ Output Body fields:
  - “message”: sent if the user record is successfully added.
- ❒ Successful Status Code: 201
- ❒ Precautions:
  - “author” field should be composed of only letters, hyphens, and spaces only.
  - “isbn” field length should not exceed 13 characters.
  - “quantity” field value should be non-negative.
  - “shelf” field value should be a letter from ‘A’ to ‘Z’.
  - The JWT token is provided in authorization header should be prefixed by the word “Bearer ”, without quotations, then token itself.
- ❒ Notes: only users with role librarians can use this API

7) Book Update API:

- ❒ Suffix Path: /book/update
- ❒ Operation Type: PUT
- ❒ Purpose: Updating a book record in the DB.
- ❒ Input Body fields:
  - "title": the title of the book to be added.
  - “author”: the author’s name of the book to be added.
  - “isbn”: the ISBN of the book to be added.
  - “quantity”: the available quantity of the book to be added.
  - "section": the library section of the book to be added.
  - “bay”: the bay number of the book to be added within the section.
  - “shelf”: the shelf number of the book to be added within the bay.
- ❒ Output Body fields:
  - “message”: sent if the user record is successfully updated.

❒ Successful Status Code: 200

❒ Precautions:

- "author" field should be composed of only letters, hyphens, and spaces only.
- "isbn" field length should not exceed 13 characters.
- "quantity" field value should be non-negative.
- "shelf" field value should be a letter from 'A' to 'Z'.
- The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.

❒ Notes:

- Book "isbn" field is assumed not to be editable.
- only users with role librarians can use this API.

8) Book Deletion API:

❒ Suffix Path: /book/delete/:isbn

❒ Operation Type: DELETE

❒ Purpose: deleting a book record in the DB.

❒ Path Parameter: the isbn of the book whose record to be deleted.

❒ Output Body fields:

"message": sent if the user record is successfully deleted.

❒ Successful Status Code: 200

❒ Precautions:

The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.

❒ Notes:

- Book "isbn" field is assumed not to be editable.
- only users with role librarians can use this API.

9) Listing All Books APIs:

- There are 2 APIs created to get all books in a Paginated manner:

A) Less Recent Added Books API:

❒ Suffix Path: /book/older/:pageSize?timestamp=

❒ Purpose: Getting at most {pageSize} of boks that are imported before certain timestamp sent as a query parameter.

B) More Recent Added Books API:

❒ Suffix Path: /book/newer/:pageSize?timestamp=

❒ Purpose: Getting at most {pageSize} of books that are imported after certain timestamp sent as a query parameter.

- Such 2 APIs share the following aspects:
  - ❒ Operation Type: GET
  - ❒ Path Parameter:
    ":pageSize": specifying the max. number of records to be retrieved.
  - ❒ Query Parameter:
    "timestamp": a threshold to get the books imported before/after
  - ❒ Output Body fields:
    A JSON Array of Name "books" where each item has all the book fields as in Book Addition API.
  - ❒ Successful Status Code: 200
  - ❒ Precautions:
    - The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.
    - Timestamp query parameter should be provided in GMT time in ISO format like "2024-12-18T19:45:19.000Z".
    - If Timestamp is not provided, it will be defaulted to the current GMT time.
  - ❒ Notes:
    - Any System Registered User can use this API.
    - These APIs gets all books registered in the system regardless their available quantity.
    - The list of returned books will be sorted in a descending order according to the book import time.

10) Book Search APIs:

- The 2 APIs used in listing books is also used here. You just need to add extra query parameters as follows: /book/older/5?timestamp=&isbn=&title=&author=
- You can leave the unnecessary parameter without any value or remove it at all if it is not considered in the search criteria.

11) PassKey Generation API:

- ❒ Suffix Path: /borrow/passKey
- ❒ Operation Type: GET
- ❒ Purpose: Generating a pass-key to be used on borrowing/returning books:
  - This pass-key will be stored in DB for associated with user email.
  - The librarian will use this pass-key in creating borrow/return request.
  - If and only if the actual user is the one that the librarian is entering his mail in borrow/return request, the used passkey will match the one stored in DB within 2 minutes ago and the request get approved.

❒ Output Body fields:

- "passkey": sent if the passkey is successfully generated and stored in DB.

❒ Successful Status Code: 200

❒ Precautions:

The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.

❒ Notes:

- After 2 minutes of calling this API, the passkey becomes invalid.
- Only a user of role 'borrower' can use this API.

12) Book Borrowing API:

❒ Suffix Path: /borrow/checkout

❒ Operation Type: POST

❒ Purpose: When a borrower borrows a book at the library, this does the following:

- Decrement book available quantity in book DB.
- Adds a new record in borrow table with necessary info.

❒ Input Body fields:

- "borrower_mail": the mail of the book borrower.
- "book_isbn": the ISBN of the book being borrowed.
- "due_date": the date-time in which the book will be returned.
- "passkey": the passkey string generated by the borrower.

❒ Output Body fields:

"message": sent if the above purpose is full filled.

❒ Successful Status Code: 201

❒ Precautions:

- The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.
- "due_date" field should be provided in GMT time in ISO format like "2024-12-18T19:45:19.000Z".

❒ Notes:

- The System deletes the passkey at the end of this API call.
- Only a user of role 'borrower' can use this API.
- Each user can borrow at most one copy of a certain book:
  - To speed up the Call to book return API
  - It's a design decision: why should someone borrow 2 copies of the same book?

13) Book Return API:

- ❒ Suffix Path: /borrow/return
- ❒ Operation Type: PUT
- ❒ Purpose: When a borrower returns a book to the library, this does the following:
  - Increment book available quantity in book DB.
  - updates the record add by the Borrow Book API in borrow table with book return date.
- ❒ Input Body fields:
  - "borrower_mail": the mail of the book borrower.
  - "book_isbn": the ISBN of the book being borrowed.
  - "passkey": the passkey string generated by the borrower.
- ❒ Output Body fields:

  "message": sent if the above purpose is full filled.
- ❒ Successful Status Code: 200
- ❒ Precautions:

  The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.
- ❒ Notes:
  - The System deletes the passkey at the end of this API call.
  - Only a user of role 'borrower' can use this API.

14) Book Tracking API:

- ❒ Suffix Path: /borrow/bookStatus/:isbn
- ❒ Operation Type: GET
- ❒ Purpose: Tracking a certain book using its ISBN
- ❒ Path Parameters:
  - "isbn": the ISBN of the book to track.
- ❒ Output Body fields:
  - "availableQuantity": the available book quantity in the library.
  - "borrowerDetails": a JSON Array of borrowers, each item has fields:
    - "first_name": the first name of the book borrower
    - "last_name": the last name of the book borrower
    - "email": the email address of book borrower
    - "borrow_date": the date-time the borrower borrowed the book
    - "due_date": the date-time in which the borrower should return the book.
- ❒ Successful Status Code: 200

❒ Precautions:

The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.

❒ Notes: Only a user of role 'librarian' can use this API.

15) Getting Borrower Books API:

❒ Suffix Path: /borrow/myBooks

❒ Operation Type: GET

❒ Purpose: used by borrower to show the books he currently has.

❒ Output Body fields:

"borrowedBooks": a JSON Array of borrowers, each item has fields:

- "title": the title of the borrowed book
- "author": the author of the borrowed book
- "isbn": the ISBN of the borrowed book
- "borrow_date": the date-time the borrower borrowed the book
- "due_date": the date-time in which the borrower should return the book.

❒ Successful Status Code: 200

❒ Precautions:

The JWT token is provided in authorization header should be prefixed by the word "Bearer ", without quotations, then token itself.

❒ Notes:

- Only a user of role 'borrower' can use this API.
- Books are returned in descending order according to "borrow_date"

16) Listing All Borrowed Books API:

- There are 2 APIs created to get all books in a Paginated manner:

A) Less Recent Added Books API:

❒ Suffix Path: /borrow/older/:pageSize?timestamp=

❒ Purpose: Getting at most {pageSize} of borrowed books that are imported before certain timestamp sent as a query parameter.

B) More Recent Added Books API:

❒ Suffix Path: /borrow/newer/:pageSize?timestamp=

❒ Purpose: Getting at most {pageSize} of borrowed books that are imported after certain timestamp sent as a query parameter.

- Such 2 APIs share the following aspects:

❒ Operation Type: GET

❒ Path Parameter:

":pageSize": specifying the max. number of records to be retrieved.

❒ Query Parameter:

“timestamp”: a threshold to get the borrowed books imported before/after

❒ Output Body fields:

A JSON Array of Name “borrowedBooks” where each item has the following fields:

- "title": title of the borrowed book
- "author": author name of the borrowed book.
- "isbn": ISBN of the borrowed book.
- "borrow_date": the date-time the borrower borrowed the book
- "due_date": the date-time the borrower should return the book,
- "first_name": borrower first name
- "last_Name": borrower last name
- "email": borrower last name

❒ Successful Status Code: 200

❒ Precautions:

- The JWT token is provided in authorization header should be prefixed by the word “Bearer ”, without quotations, then token itself.
- Timestamp query parameter should be provided in GMT time in ISO format like “2024-12-18T19:45:19.000Z”.
- If Timestamp is not provided, it will be defaulted to the current GMT time.

❒ Notes:

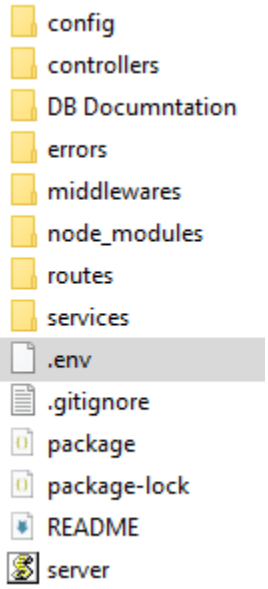
- Any User of role “librarian” can use this API.
- The list of returned books will be sorted in a descending order according to the book “borrow_date”.

17) Listing OverDue Borrowed Books API:

Similar to the previous APIs. Just add another query parameter called overdue and set its value to true. Any other value gets the same results of the previous API.

■ Setup Instructions:

- Just Run the associated setupScript.sql to setup DB schema
- The system depends on the presence of an admin user who is authorized to do specific tasks such as registering librarians, ...
  - There is no need to insert a record for this admin user while DB setup.
  - Just attach an .env file in the application source code as shown below and the admin user record would be inserted once the server starts:
    - Make sure the admin name has no digits, just letters, spaces and hyphes. Otherwise, the admin record would be not updatable.
    - Make sure to update The admin password once the server starts for security issues.
    - Make sure to add all these variables stored in the below env file.



.env

```
1   MYSQL_HOST=localhost
2   MYSQL_USER=root
3   MYSQL_PASSWORD=password
4   MYSQL_DATABASE=Library
5   PORT=3000
6
7   ADMIN_Name=admin
8   ADMIN_MAIL=admin@library.com
9   ADMIN_PASSWORD=admin12345
10
11  JWT_SECRET_KEY=a1eQMPLwcBnoExIRzamTpjMYu3m2KsM4wJrRTLmxoQf65Gwsyup
```