Library Management System

Handling User Registration: User End

1. **Username Handling**: When a user enters a username, we use the fetch api of javascript to fetch from a post request to an endpoint. The endpoint will check usernames in registration table which are in pending or approved state. If the username matches the retrieved usernames then the endpoint will return false else true. As a result we allow the username if the return value is true.
2. **Main Form handling**: After the username is allowed one can sign up for the given fields. Here Three Hidden forms will be used to send paid:unpaid, transaction:null, rejectionmsg:null
3. **Payment Handling**: Use a Form to first select the payment mode. If the payment mode is offline then it will say to visit the user and submit the form with paid:unpaid, transaction:null, rejectionmsg:null. Else if the payment mode is online it will ask to submit the transaction id and send data as paid:paid, transaction:transactionid, rejectionmsg:null.

Handling User Registration: Admin End

1. **Checking for pending PAID requests**: In the admin panel there will be two sections one of them is PAID PENDING APPROVALS. The Admin checks the transaction id and matches it with the trasactions manually to approve. If rejected then it will be handled in rejection handler.
2. **Checking for pending UNPAID requests:** The next section in the admin panel will be PAID PENDING APPROVALS. When the user visits the admin for offline payment, he will check this section to approve the user directly.
3. **Rejection Handler:** If the admin rejects some user, he will write a message as a reason. 1st, the user will be rejected by updating the approved column. 2nd, The message will be sent to his given email address.

Routes:

The routes in the `AdminController` class are:

1. GET: "`/admin/viewpending/paid`" - This route maps to the viewPaidRegistration() method and displays pending registration requests with "paid" category.
2. GET: "`/admin/viewpending/unpaid`" - This route maps to the viewUnPaidRegistration() method and displays pending registration requests with "unpaid" category.
3. POST: "`/admin/approveuser`" - This route maps to the approveUser() method and is used to approve a user registration request. It takes in parameters rsid (registration request ID) and pay (payment status).
4. POST: "`/admin/rejectuser`" - This route maps to the rejectUser() method and is used to reject a user registration request. It takes in parameters rsid (registration request ID), message (rejection reason), and pay (payment status).

The routes in the class `BooksController` are:

1. POST `/books` - Endpoint to insert a new book with a thumbnail file.
2. GET `/books` - Endpoint to get the book form.

3. GET `/books/search` - Endpoint to search for books by a search parameter.
4. GET `/books/author` - Endpoint to search for books by author.
5. GET `/books/publisher` - Endpoint to search for books by publisher.
6. GET `/books/title` - Endpoint to search for books by title.

in the `JournalsController` class, the routes (URL paths) handled by this controller are as follows:

1. GET " `/journals` ": This route maps to the method and is responsible for rendering the journal form page.
2. POST " `/journals` ": This route maps to the method and is responsible for inserting a new journal into the system.
3. GET " `/journals/search` ": This route maps to the method and is responsible for handling search requests for journals based on a search parameter.
4. GET " `/journals/editor` ": This route maps to the method and is responsible for handling requests to search for journals by editor/author.
5. GET " `/journals/publisher` ": This route maps to the method and is responsible for handling requests to search for journals by publisher.
6. GET " `/journals/title` ": This route maps to the method and is responsible for handling requests to search for journals by title.

The routes (URL paths) handled by the `MagazinesController` class are as follows:

1. GET " `/magazines` ": This route maps to the method and is responsible for rendering the magazine form page.
2. POST " `/magazines` ": This route maps to the method and is responsible for inserting a new magazine into the system.
3. GET " `/magazines/search` ": This route maps to the method and is responsible for handling search requests for magazines based on a search parameter.
4. GET " `/magazines/title` ": This route maps to the method and is responsible for handling requests to search for magazines by title.

The routes (URL paths) handled by the `SoftCopyController` class are as follows:

1. GET " `/softcopy` ": This route maps to the method and is responsible for rendering the soft copy form page.
2. POST " `/softcopy` ": This route maps to the method and is responsible for inserting a new soft copy into the system.
3. GET " `/softcopy/search` ": This route maps to the method and is responsible for handling search requests for soft copies based on a search parameter.
4. GET " `/softcopy/title` ": This route maps to the method and is responsible for handling requests to search for soft copies by title.

The routes (URL paths) handled by the `ThesesController` class are as follows:

1. GET " `/theses` ": This route maps to the method and is responsible for rendering the theses form page.
2. POST " `/theses` ": This route maps to the method and is responsible for inserting a new thesis into the system.
3. GET " `/theses/search` ": This route maps to the method and is responsible for handling search requests for theses based on a search parameter.
4. GET " `/theses/title` ": This route maps to the method and is responsible for handling requests to search for theses by title.

the following routes are defined in the `RegistrationController` class:

1. `GET` request to `/registration` - mapped to `getRegistrationForm()` method, which returns a view name "personalDetailsForm".
2. `POST` request to `/registration/checkusername` - mapped to `checkUsername()` method, which takes a `username` as a request parameter, checks if it exists in the list of pending and approved usernames using `registrationServices.findPendingApprovedUsernames()`, and returns a `ResponseEntity` containing a boolean response ("true" if the username is not found and "false" if the username is found) with HTTP status code 200 (OK).
3. `POST` request to `/registration/submitregister` - mapped to `submitPersonalDetails()` method, which takes a `Registration` object, a `MultipartFile` for profile picture, and a `Model` object as parameters. It inserts the registration details into the database using `registrationServices.insertOneRegistration()`, sets a success message based on whether the payment is "paid" or not, adds the message to the model, and returns a view name "registrationRequestSubmitted".

the following routes are defined in the `LoginController` class:

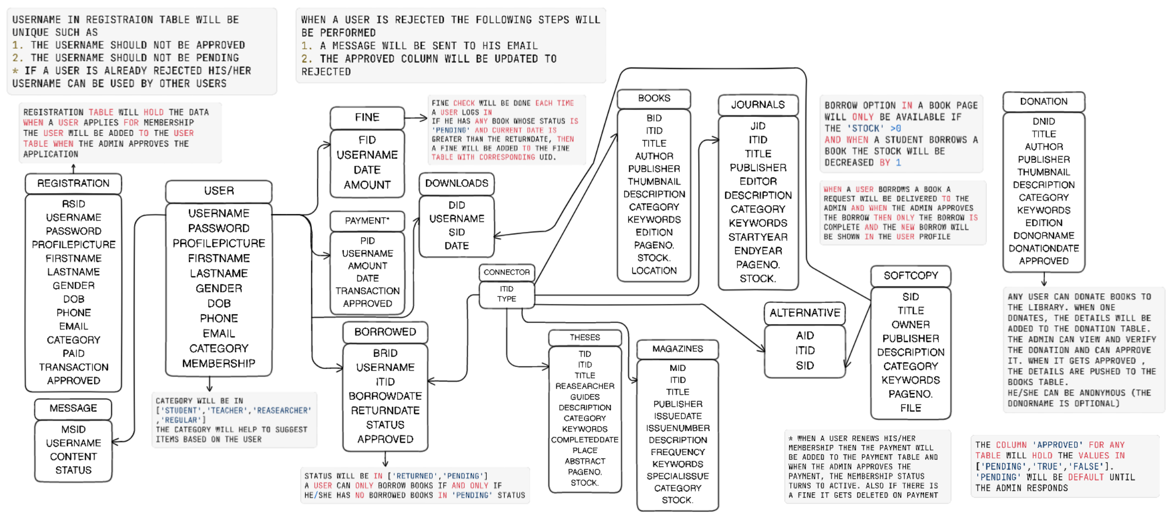
1. `GET` request to `/login/users` - mapped to `getUserLoginForm()` method, which returns a view name "userLogin".
2. `POST` request to `/login/users` - mapped to `userLogin()` method, which takes `username`, `password`, and `HttpServletRequest` as parameters. It checks if the provided `username` and `password` match with any user in the database using `usersServices.findUserByUsernamePassword()` method, and if found, sets a session using `SessionHandler.setSession()` method and redirects to "/user" route. If not found, it redirects back to "/login/users" route.

The `SearchController` has several endpoints defined using the `@GetMapping` annotation, each corresponding to a specific type of search. For example:

- `/books`: This endpoint handles a search for all books and returns a list of `Books` objects as the search results.
- `/searchitem`: This endpoint handles a general search query based on a search parameter. It searches for items such as books, journals, magazines, theses, and soft copies that match the search parameter. The search results are then added to a `Model` object and passed to the "searchResult" view.

- `/author` : This endpoint handles a search query based on the author's name and returns a list of books that match the author's name.
- `/researcher` : This endpoint handles a search query based on the researcher's name and returns a list of theses that match the researcher's name.
- `/editor` : This endpoint handles a search query based on the editor's name and returns a list of journals that match the editor's name.
- `/owner` : This endpoint handles a search query based on the owner's name and returns a list of soft copies that match the owner's name.
- `/publisher` : This endpoint handles a search query based on the publisher's name and returns a list of books, journals, magazines, and soft copies that match the publisher's name.
- `/category` : This endpoint handles a search query based on the category and returns a list of books, journals, magazines, theses, and soft copies that match the category.
- `/title` - Handles a GET request for searching items by title.
- `/category/{category}` - Handles a GET request for searching items by category.
- `/books/{bookId}` - Handles a GET request for retrieving details of a single book by its ID.
- `/journals/{jid}` - Handles a GET request for retrieving details of a single journal by its ID.
- `/magazines/{mid}` - Handles a GET request for retrieving details of a single magazine by its ID.
- `/theses/{tid}` - Handles a GET request for retrieving details of a single thesis by its ID.
- `/softcopy/{sid}` - Handles a GET request for retrieving details of a single soft copy by its ID.

Library Management System



USERNAME IN REGISTRAION TABLE WILL BE UNIQUE SUCH AS
1. THE USERNAME SHOULD NOT BE APPROVED
2. THE USERNAME SHOULD NOT BE PENDING
* IF A USER IS ALREADY REJECTED HIS/HER USERNAME CAN BE USED BY OTHER USERS

WHEN A USER IS REJECTED THE FOLLOWING STEPS WILL BE PERFORMED
1. A MESSAGE WILL BE SENT TO HIS EMAIL
2. THE APPROVED COLUMN WILL BE UPDATED TO REJECTED

REGISTRATION TABLE WILL HOLD THE DATA WHEN A USER APPLIES FOR MEMBERSHIP THE USER WILL BE ADDED TO THE USER TABLE WHEN THE ADMIN APPROVES THE APPLICATION

FINE CHECK WILL BE DONE EACH TIME A USER LOGS IN
IF HE HAS ANY BOOK WHOSE STATUS IS 'PENDING' AND CURRENT DATE IS GREATER THAN THE RETURNDATE, THEN A FINE WILL BE ADDED TO THE FINE TABLE WITH CORRESPONDING UID.

BORROW OPTION IN A BOOK PAGE WILL ONLY BE AVAILABLE IF THE 'STOCK' >0
AND WHEN A STUDENT BORROWS A BOOK THE STOCK WILL BE DECREASED BY 1

WHEN A USER BORRDWS A BOOK A REQUEST WILL BE DELIVERED TO THE ADMIN AND WHEN THE ADMIN APPROVES THE BORROW THEN ONLY THE BORROW IS COMPLETE AND THE NEW BORROW WILL BE SHOWN IN THE USER PROFILE

CATEGORY WILL BE IN ['STUDENT','TEACHER','REASEARCHER','REGULAR']
THE CATEGORY WILL HELP TO SUGGEST ITEMS BASED ON THE USER

STATUS WILL BE IN ['RETURNED','PENDING']
A USER CAN ONLY BORROW BOOKS IF AND ONLY IF HE/SHE HAS NO BORROWED BOOKS IN 'PENDING' STATUS

ANY USER CAN DONATE BOOKS TO THE LIBRARY. WHEN ONE DONATES, THE DETAILS WILL BE ADDED TO THE DONATION TABLE. THE ADMIN CAN VIEW AND VERIFY THE DONATION AND CAN APPROVE IT. WHEN IT GETS APPROVED , THE DETAILS ARE PUSHED TO THE BOOKS TABLE.
HE/SHE CAN BE ANONYMOUS (THE DONORNAME IS OPTIONAL)

* WHEN A USER RENEWS HIS/HER MEMBERSHIP THEN THE PAYMENT WILL BE ADDED TO THE PAYMENT TABLE AND WHEN THE ADMIN APPROVES THE PAYMENT, THE MEMBERSHIP STATUS TURNS TO ACTIVE. ALSO IF THERE IS A FINE IT GETS DELETED ON PAYMENT

THE COLUMN 'APPROVED' FOR ANY TABLE WILL HOLD THE VALUES IN ['PENDING','TRUE','FALSE']. 'PENDING' WILL BE DEFAULT UNTIL THE ADMIN RESPONDS