**DOCUMENTATION**

**LIBRARY MANAGEMENT SYSTEM**

**Problem Statement**

The objective is to develop a library management page that allows users to view a list of books and perform filtering based on various criteria such as title, author, subject, and publish date. Additionally, the page should implement paging functionality, displaying a limited number of books per page or providing the option to load more books on scroll. The page should also include the ability to display the count of books corresponding to each filtering criteria, enabling users to easily identify the number of books available based on their preferences.

**Tech Stack**

* Angular
* HTML
* CSS
* TypeScript

**Component**

A component is a reusable and self-contained unit that combines the behavior and presentation of a specific part of the user interface. It consists of a TypeScript class for the component's logic and an HTML template for defining its structure.

**Components used**

**Book-details**

Description: A component that displays detailed information about a specific book.Dependencies: NoneProperties: 1. book: Holds the book object with properties like title, author, genre, description, etc.HTML Template:Displays the book details, including the title, author, genre, description, and availability.Provides options to borrow or return the book based on its availability.

**Book-report**

Description: A component that displays a summary of borrowed books and allows users to pay fines or return books.Dependencies: 1. BookRequestService: Manages book request-related services.Properties: 1. books: Holds an array of Book objects representing the borrowed books.Methods: 1. ngOnInit(): Fetches the book report on component initialization. 2. fetchBookReport(): Retrieves the book report from the BookRequestService and updates the books array. 3. isDueDateExceeded(): Checks if the due date for a book has been exceeded. 4. payFine(bookId): Calculates and stores the fine amount for a book in the local storage. 5. returnBook(bookId): Sends a request to the BookRequestService to return a book.HTML Template:Displays a table with columns for book title, author, genre, status, issued date, due date, and actions.Iterates over the books array and populates the table rows with the respective book details.Provides buttons to pay fines or return books based on the book status.Shows a message if no books are borrowed and provides an option to explore more books.

**Confirmation**

Description: A component that displays the payment success confirmation message after a successful payment.Properties: 1. fineAmount: Holds the fine amount paid by the user (in INR). 2. bookId: Holds the ID of the book for which the payment was made.Methods: 1. ngOnInit(): Retrieves the fine amount and book ID from the local storage.HTML Template:Shows the payment success message along with the book ID and fine amount.Provides a button to discover additional books.

**Login**

Description: A component responsible for handling user login functionality.Dependencies: 1. Router: Used for navigation between pages. 2. AuthService: Provides authentication-related services. 3. BookRequestService: Manages book request-related services.Form Controls: 1. email: Tracks the email input field with required and email validation. 2. password: Tracks the password input field with required and minimum length validation.Methods: 1. ngOnInit(): Initializes the login form by creating a new FormGroup and assigning the email and password form controls. 2. login(): Retrieves the email and password values from the login form and performs the login process.HTML Template:Represents a login form with email and password input fields.Validates the form using Angular reactive forms and displays error messages if the fields are not filled or do not meet the validation criteria.Provides a login button to trigger the login process.Offers a link to register for new users.

**Navbar**

Description: A component that represents the navigation bar of the Library Management System.Dependencies: AuthService (for user authentication)Properties: 1. searchOptions: An array of search filter options. 2. selectedOption: Holds the currently selected search filter option. 3. isSearchFormVisible: Determines the visibility of the search form.

**Payment**

Description: A component that represents the payment page of the Library Management System.

Properties: 1. fineAmount: The fine amount to be paid for the book. 2. bookId: The ID of the book for which the payment is being made. 3. paymentDetails: An object that holds the payment details entered by the user.Methods: 1. ngOnInit(): A lifecycle hook method that initializes the component and retrieves the fine amount and book ID from local storage. 2. get formValid(): A getter method that checks if the payment form is valid. 3. onSubmit(): A method that is called when the payment form is submitted. It performs payment logic if the form is valid, otherwise displays an error. 4. isCardNumberValid(): A method that checks if the card number entered is valid. 5. isExpirationDateValid(): A method that checks if the expiration date entered is valid. 6. iscvvValid(): A method that checks if the CVV entered is valid.

**Register**

Description: A component that represents the registration page of the Library Management System.Properties: 1. registerForm: A FormGroup instance that holds the form controls for registration. 2. email: A FormControl instance that represents the email input field.password: A FormControl instance that represents the password input field. 3. confirmPassword: A FormControl instance that represents the confirm password input field.Methods: 1. ngOnInit(): A lifecycle hook method that initializes the component and sets up the registerForm with its form controls. 2. getEmailError(): A method that returns the error message for the email input field based on the validation rules. 3. getPasswordError(): A method that returns the error message for the password input field based on the validation rules. 4. getConfirmPasswordError(): A method that returns the error message for the confirm password input field based on the validation rules. 5. onSubmit(): A method that is called when the registration form is submitted. It performs registration logic, validates the form inputs, and stores the user data in local storage.

**Single-book**

Description: A component that displays the details of a single book and allows the user to request the book.Properties: 1. book: A Book object that holds the details of the book to be displayed. 2. successMessage: A string that represents the success message displayed when the book is successfully requested. 3. errorMessage: A string that represents the error message displayed when there is an error in requesting the book.books: An array of Book objects that holds the list of books (possibly for future use).Methods: 1. ngOnInit(): A lifecycle hook method that initializes the component and calls the getBookDetails() method to retrieve the book details. 2. getBookDetails(): A method that retrieves the book details based on the book ID from the route parameters. It calls the getBookById() method of the BookDetailsService to fetch the book details. 3. requestBook(): A method that is called when the user clicks the "Request Book" button. It calls the requestBook() method of the BookRequestService to request the book. If the request is successful, it updates the local storage with the borrowed book details, sets the success message, and navigates to the book report page. If there is an error, it sets the error message.

**User-details**

Description: A component that displays the user's email, their borrowed books, and recommended books based on their borrowing history.Properties: 1. userEmail: A string that holds the user's email address.bookReport: An array of Book objects that holds the user's borrowed books. 2. recommendedBooks: An array of Book objects that holds the recommended books based on the user's borrowing history.Methods: 1. ngOnInit(): A lifecycle hook method that initializes the component. It retrieves the user's email from the local storage, calls the fetchBookHistory() method to fetch the user's borrowed books, and calls the recommendBooks() method to recommend books based on the user's borrowing history. 2. fetchBookHistory(): A method that fetches the user's borrowed books. It calls the getBookReport() method of the BookRequestService to retrieve the book report and filters the result to get only the books with the "Issued" status. 3. recommendBooks(): A method that recommends books based on the user's borrowing history. It calculates the most frequent book category from the book report, calls the getBooksByCategories() method of the BookDetailsService to retrieve books of that category, filters out the books already borrowed by the user, and selects the top three recommended books.

**Interface**

An interface is a TypeScript construct used to define the structure and type of objects, providing a contract for consistent and type-safe coding practices. It is commonly used to define data models, API responses, and component inputs.

**Interface Used**

**Model**

Description: Represents a book entity with various properties.Properties: 1. daysLeft (optional): A number representing the number of days left for the book to be returned. 2. bookId: A string representing the unique identifier of the book.title: A string representing the title of the book. 3. author: A string representing the author of the book. 4. publicationDate: A string representing the publication date of the book. 5. summary: A string representing the summary or description of the book.image (optional): A string representing the URL or path of the book's image. 6. category: A string representing the category or genre of the book.status (optional): A string representing the current status of the book, such as "Available", "Issued", etc. 7. issuedDate (optional): A Date object representing the date when the book was issued. 8. dueDate (optional): A Date object representing the due date for returning the book. 9. fine (optional): A number representing the fine amount for returning the book late. 10. borrower (optional): A string representing the email or identifier of the borrower who has borrowed the book. 11. returnedBy (optional): A string representing the email or identifier of the user who returned the book. 12. returnedDate (optional): A Date object representing the date when the book was returned.

**Service**

A service is a class that encapsulates reusable functionality and data, allowing components and other services to share and access common resources and perform business logic. It promotes modularity, separation of concerns, and facilitates dependency injection for easy testability and maintainability.

**Services Used**

**Auth**

The AuthService is an Angular service that handles authentication-related functionality. It manages the user's login state, provides methods for logging in and out, and checks if the user is currently logged in. It relies on the Router module to handle navigation.

**Book-details**

Import statements: The code imports various modules, functions, and the Book model from different sources. These imports are necessary for the functionality of the service.Injectable decorator: The @Injectable decorator is used to declare that the BookDetailsService class is injectable and can be used as a dependency in other components or services.Class definition: The BookDetailsService class contains methods and properties related to retrieving book details.default and books properties: These properties store default values for an image and an array of Book objects, respectively. The Book model is expected to have properties such as bookId, title, author, publicationDate, summary, image, and category.

**Book-request**

1. It maintains an array bookReport to keep track of issued books and their details.
2. The service loads the bookReport from local storage when initialized.It provides methods to request a book, update book status, and return a book.
3. The simulateServerRequest method simulates a server request to issue a book. It updates the book's status, issued date, and due date, and adds it to the bookReport.
4. The service saves the bookReport to local storage after any changes.
5. It also provides methods to clear the bookReport when the user changes and update the days left for a book.
6. The getRecommendedBooks method takes a list of categories and returns recommended books based on the user's book history. It retrieves books by category using the bookDetailsService and filters out books that are already in the user's history.