### ****Frontend****

### **All methods and fields in the system have public access by default.**

#### ****Components****

1. **LoginComponent**

**Attributes:**

* props: Credentials – Stores the credentials data used for login (email and password).

**Methods:**

* render(): JSX – Renders the login form.
* handleLogin(): void – Handles the login process, likely triggers an API call to authenticate.

1. **RegisterComponent**

**Attributes:**

* props: UserDTO – Stores the user data required for registration (name, email, etc.).

**Methods:**

* render(): JSX – Renders the registration form.
* handleRegister(): void – Handles the registration process by sending the user data to an API.

1. **ProfileFormComponent**

**Attributes:**

* props: ProfileFormProps – Stores the properties for the profile form (user data for editing).

**Methods:**

* render(): JSX – Renders the profile editing form.
* handleSubmit(): void – Handles form submission for updating user profile.

1. **BookSearchComponent**

**Attributes:**

* props: BookSearchProps – Stores the properties related to the book search (e.g., filters, search terms).

**Methods:**

* render(): JSX – Renders the book search form/UI.
* handleSearch(): void – Handles the book search process based on user input.

1. **BookViewerComponent**

**Attributes:**

* props: BookViewerProps – Stores properties for the book viewer (e.g., current book, page number).

**Methods:**

* render(): JSX – Renders the book viewer UI.
* handlePageChange(): void – Handles page changes while reading the book.
* handleBookmark(): void – Handles bookmarking the current page or book.

1. **PaymentFormComponent**

**Attributes:**

* props: PaymentFormProps – Stores properties related to the payment form (e.g., payment details).

**Methods:**

* render(): JSX – Renders the payment form.
* handleSubmit(): void – Handles the payment form submission.

1. **ReviewFormComponent**

**Attributes:**

* props: ReviewFormProps – Stores properties related to submitting a review (e.g., review text, rating).

**Methods:**

* render(): JSX – Renders the review form.
* handleReviewSubmit(): void – Handles submitting the review to the system.

1. **AdminBookListComponent**

**Attributes:**

* props: AdminBookListProps – Stores properties for managing the list of books in the admin view.

**Methods:**

* render(): JSX – Renders the admin book list UI.
* handleBookAction(): void – Handles actions related to books (edit, delete, publish).

1. **AdminUserListComponent**

**Attributes:**

* props: AdminUserListProps – Stores properties for managing the user list in the admin view.

**Methods:**

* render(): JSX – Renders the admin user list UI.
* handleUserAction(): void – Handles user-related actions (block, delete, change role).

1. **EditorDraftListComponent**

**Attributes:**

* props: EditorDraftListProps – Stores properties for managing drafts in the editor's view.

**Methods:**

* render(): JSX – Renders the editor draft list UI.
* handleSubmitDraft(): void – Handles submitting the draft for review or publishing.

**Pages**

1. **AuthPage**

**Attributes:**

* props: AuthPageProps – Stores properties related to the authentication page (e.g., form fields, error messages).
* formData: Object – Stores the form data entered by the user for login or registration.

**Methods:**

* useEffect() – Performs side effects such as initializing form data or checking authentication status.
* handleSubmit(): void – Handles form submission (login or registration).
* render(): JSX – Renders the authentication page UI.
* dispatch(): void – Dispatches actions to the Redux store (e.g., login or registration actions).

1. **ProfilePage**

**Attributes:**

* props: ProfilePageProps – Stores properties related to the profile page (e.g., user data, editing state).
* isEditing: boolean – Indicates whether the user is in edit mode for the profile.
* reduxData: UsersState – Stores state data related to users from the Redux store.

**Methods:**

* useEffect() – Runs side effects such as fetching user data or checking access permissions.
* checkAccess(): boolean – Checks if the current user has permission to access the profile.
* render(): JSX – Renders the profile page UI.
* dispatch(): void – Dispatches actions related to user data and profile management.

1. **BookSearchPage**

**Attributes:**

* props: BookSearchPageProps – Stores properties related to the book search page (e.g., search query, filters).
* searchQuery: string – Stores the current search query entered by the user.
* reduxData: BookReaderState – Stores the state of the book reader from the Redux store.

**Methods:**

* useEffect() – Runs side effects such as fetching books based on search query.
* fetchBooks(): void – Fetches books matching the search query.
* render(): JSX – Renders the book search page UI.
* dispatch(): void – Dispatches actions to the Redux store related to book search.

1. **BookViewerPage**

**Attributes:**

* props: BookViewerPageProps – Stores properties related to the book viewer page (e.g., book, current page).
* currentPage: number – Stores the current page number being viewed in the book.
* reduxData: BookReaderState – Stores the state related to book reading and progress from Redux.

**Methods:**

* useEffect() – Runs side effects such as checking user access to the book or setting initial page.
* checkAccess(): boolean – Checks if the user has permission to view the book.
* handleBookmark(): void – Handles bookmarking the current page or book.
* render(): JSX – Renders the book viewer page UI.
* dispatch(): void – Dispatches actions to the Redux store related to the book viewer.

1. **PaymentPage**

**Attributes:**

* props: PaymentPageProps – Stores properties related to the payment page (e.g., payment details, user data).
* paymentData: PaymentDTO – Stores the payment data entered by the user.
* reduxData: PaymentsState – Stores the state data related to payments from Redux.

**Methods:**

* useEffect() – Runs side effects related to the payment process, like fetching payment details.
* submitPayment(): void – Submits the payment data.
* render(): JSX – Renders the payment page UI.
* dispatch(): void – Dispatches actions related to the payment process.

1. **ReviewPage**

**Attributes:**

* props: ReviewPageProps – Stores properties related to the review page (e.g., book details, review form).
* reviewText: string – Stores the text entered by the user for the review.
* rating: number – Stores the rating given by the user.
* reduxData: ReviewsState – Stores the state data related to reviews.

**Methods:**

* useEffect() – Runs side effects like fetching existing reviews.
* submitReview(): void – Submits the review data.
* render(): JSX – Renders the review page UI.
* dispatch(): void – Dispatches actions related to the review process.

1. **AdminDashboardPage**

**Attributes:**

* reduxData: BookReaderState – Stores state data related to book reading and management.

**Methods:**

* useEffect() – Runs side effects such as loading dashboard data.
* moderateBook(): void – Handles moderation tasks for books (e.g., approve, reject).
* render(): JSX – Renders the admin dashboard page UI.
* dispatch(): void – Dispatches actions related to book management.

1. **OwnerDashboardPage**

**Attributes:**

* reduxData: UsersAdministrationState – Stores state data related to user administration (e.g., roles, user list).

**Methods:**

* useEffect() – Runs side effects like loading the dashboard data.
* manageAdmin(): void – Manages administrative actions (e.g., role changes).
* render(): JSX – Renders the owner dashboard page UI.
* dispatch(): void – Dispatches actions related to user and role management.

1. **EditorDashboardPage**

**Attributes:**

* draftData: BookDTO – Stores data related to the book draft being edited.
* reduxData: BookEditorState – Stores state data related to book editing.

**Methods:**

* useEffect() – Runs side effects such as fetching draft data.
* submitBook(): void – Submits the book for review or publication.
* render(): JSX – Renders the editor dashboard page UI.
* dispatch(): void – Dispatches actions related to book editing.

### Redux

### 20. ****AuthSlice****

* **Attributes:**
* state: AuthState – Holds the authentication state.
* **Methods:**
* login(email: String, password: String): void – Authenticates the user with email and password.
* logout(): void – Logs the user out.
* register(userData: UserDTO): void – Registers a new user.
* refreshToken(rt: String): void – Refreshes the access token.
* sso(method: SSOMethods): void – Handles authentication via third-party SSO.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 21. ****BookAdminSlice****

* **Attributes:**
* state: BookAdminState – Holds the book admin state.
* **Methods:**
* updateBook(bookId: String, bookData: BookDTO): void – Updates book details.
* deleteBook(bookId: String): void – Deletes a book.
* previewBook(bookId: String): void – Previews a book.
* publishBook(bookId: String): void – Publishes a book.
* rejectBook(bookId: String): void – Rejects a book.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 22. ****BookEditorSlice****

* **Attributes:**
* state: BookEditorState – Holds the book editor state.
* **Methods:**
* updateBook(bookId: String, bookData: BookDTO): void – Updates book details.
* deleteBook(bookId: String): void – Deletes a book.
* addAudioVersion(audio: any): void – Adds an audio version of the book.
* suggestBook(bookData: BookDTO): void – Suggests a new book.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 23. ****BookReaderSlice****

* **Attributes:**
* state: BookReaderState – Holds the book reader state.
* **Methods:**
* setCurrentPage(page: number): void – Sets the current page.
* setPageSize(pageSize: number): void – Sets the page size.
* listenAudioBook(bookId: String): void – Starts listening to an audio book.
* downloadBook(type: FileType): void – Downloads a book in a specified file type.
* addBookToList(type: ListType, bookId: String): void – Adds a book to a list.
* deleteBookFromList(type: ListType, bookId: String): void – Removes a book from a list.
* buyBook(bookId: String): void – Buys a book.
* setSearchFilter(searchParam: String): void – Sets the search filter.
* setSortOrder(order: SortOrder): void – Sets the sorting order.
* searchBooks(searchParam: String): void – Searches for books.
* getBook(bookId: String): void – Retrieves a book by its ID.
* getBoughtBooks(userId: String): void – Retrieves a list of books bought by a user.
* getAllBooks(authorId: String): void – Retrieves all books by a specific author.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 24. ****ReviewsSlice****

* **Attributes:**
* state: ReviewsState – Holds the reviews state.
* **Methods:**
* fetchReviews(bookId: String): void – Fetches reviews for a book.
* addReview(reviewData: ReviewDTO): void – Adds a new review.
* deleteReview(reviewId: String): void – Deletes a review.
* editReview(reviewId: String, reviewData: ReviewDTO): void – Edits an existing review.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 25. ****PaymentsSlice****

* **Attributes:**
* state: PaymentsState – Holds the payments state.
* **Methods:**
* processPayment(paymentData: PaymentDTO): void – Processes a payment.
* getPaymentDetails(paymentId: String): void – Retrieves payment details.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 26. ****UsersAdministrationSlice****

* **Attributes:**
* state: UsersAdministrationState – Holds the users administration state.
* **Methods:**
* fetchUser(userId: String): void – Fetches a user by ID.
* getUsers(ids: String[]): void – Retrieves a list of users by their IDs.
* changeRole(userId: String, role: Role): void – Changes the user's role.
* deleteUser(userId: String): void – Deletes a user.
* blockUser(userId: String): void – Blocks a user.
* allowEditor(userId: String): void – Grants editor access to a user.
* rejectEditor(userId: String): void – Rejects editor access for a user.
* updateProfile(userData: UserDTO): void – Updates a user's profile.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 27. ****UsersSlice****

* **Attributes:**
* state: UsersState – Holds the users state.
* **Methods:**
* fetchUser(userId: String): void – Fetches a user by ID.
* startSubscription(cardNumber: String): void – Starts a subscription for a user.
* stopSubscription(): void – Stops the user's subscription.
* becomeEditor(data: EditorForm): void – Allows a user to become an editor.
* getList(type: ListType): void – Retrieves a list of users based on the type.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 28. ****BookmarksSlice****

* **Attributes:**
* state: BookmarksState – Holds the bookmarks state.
* **Methods:**
* fetchBookmarksByUser(userId: String): void – Fetches bookmarks by user ID.
* fetchBookmarksByBook(bookId: String): void – Fetches bookmarks by book ID.
* createBookmark(bookmarkData: BookmarkDTO, bookId: String): void – Creates a bookmark for a book.
* editBookmark(bookmarkData: BookmarkDTO, bookmarkId: String): void – Edits an existing bookmark.
* deleteBookmark(bookmarkId: String): void – Deletes a bookmark.
* getBookmarkById(bookmarkId: String): void – Retrieves a bookmark by its ID.
* setLoading(isLoading: boolean): void – Sets the loading state.
* setError(error: String): void – Sets the error message.

### 29. ****AuthApi****

* **Attributes:**
* None
* **Methods:**
* login(cred: Credentials): Tokens – Authenticates the user with credentials (email and password).
* register(userData: UserDTO): void – Registers a new user.
* refreshToken(rt: String): Tokens – Refreshes the access token.
* logout(): void – Logs the user out.
* sso(method: SSOMethods): Tokens – Authenticates via third-party SSO services.

### 30. ****BooksApi****

* **Attributes:**
* None
* **Methods:**
* getAllBooks(authorId: String): List<BookDTO> – Retrieves all books for a given author.
* searchBooks(searchParam: String): List<BookDTO> – Searches for books based on search parameters.
* getBook(bookId: String): BookDTO – Retrieves a book by its ID.
* suggestBook(bookData: BookDTO): void – Suggests a new book.
* buyBook(bookId: String): BookDTO – Purchases a book by ID.
* previewBook(bookId: String): BookDTO – Previews a book.
* publishBook(bookId: String): void – Publishes a book.
* rejectBook(bookId: String): void – Rejects a book.
* updateBook(bookId: String, bookData: BookDTO): void – Updates book details.
* deleteBook(bookId: String): void – Deletes a book.
* getBoughtBooks(userId: String): List<BookDTO> – Retrieves all books purchased by a user.
* addAudioVersion(audio: any): BookDTO – Adds an audio version of a book.
* listenAudioBook(bookId: String): mp3 – Plays the audio version of a book.
* downloadBook(type: FileType): File – Downloads a book in a specified file type.
* addBookToList(type: ListType, bookId: String): void – Adds a book to a list.
* deleteBookFromList(type: ListType, bookId: String): void – Removes a book from a list.

### 31. ****ReviewsApi****

* **Attributes:**
* None
* **Methods:**
* getReviews(bookId: String): List<ReviewDTO> – Retrieves reviews for a book by ID.
* addReview(bookId: String, reviewData: ReviewDTO): void – Adds a review for a book.
* deleteReview(reviewId: String): void – Deletes a review by ID.
* editReview(reviewId: String, reviewData: ReviewDTO): void – Edits an existing review by ID.

### 32. ****PaymentApi****

* **Attributes:**
* None
* **Methods:**
* processPayment(paymentData: PaymentDTO): void – Processes a payment.
* getPaymentDetails(paymentId: String): PaymentDTO – Retrieves payment details by ID.

### 33. ****UsersApi****

* **Attributes:**
* None
* **Methods:**
* getUser(userId: String): UserDTO – Retrieves user information by ID.
* changeInfo(userData: UserDTO): void – Changes user information.
* changeRole(userId: String, role: String): void – Changes a user's role.
* deleteUser(userId: String): void – Deletes a user.
* blockUser(userId: String): void – Blocks a user.
* startSubscription(cardNumber: String): void – Starts a subscription for a user.
* stopSubscription(): void – Stops a user's subscription.
* getList(type: String): List<String> – Retrieves a list of users based on type.
* getUsers(ids: String[]): List<UserDTO> – Retrieves a list of users by their IDs.
* becomeEditor(data: EditorForm): void – Allows a user to become an editor.
* allowEditor(userId: String): void – Grants editor access to a user.
* rejectEditor(userId: String): void – Denies editor access to a user.

### 34. ****BookmarksApi****

* **Attributes:**
* None
* **Methods:**
* getBookmarksByUser(userId: String): List<BookmarkDTO> – Retrieves bookmarks for a user by ID.
* getBookmarksByBook(bookId: String): List<BookmarkDTO> – Retrieves bookmarks for a book by ID.
* createBookmark(bookmarkData: BookmarkDTO, bookId: String): BookmarkDTO – Creates a bookmark for a book.
* editBookmark(bookmarkData: BookmarkDTO, bookmarkId: String): BookmarkDTO – Edits an existing bookmark.
* deleteBookmark(bookmarkId: String): void – Deletes a bookmark.
* getBookmarkById(bookmarkId: String): BookmarkDTO – Retrieves a bookmark by its ID.

### 35. ****LocalStorage****

* **Attributes:**
* None
* **Methods:**
* setItem(key: String, value: String): void – Stores data in localStorage.
* getItem(key: String): String – Retrieves data by key from localStorage.
* removeItem(key: String): void – Removes data by key from localStorage.
* clear(): void – Clears all data in localStorage.

**Types**

### 1. ****SSOMethods**** (Enum)

* **Values:**
* FACEBOOK – Represents Facebook as an SSO method.
* GOOGLE – Represents Google as an SSO method.
* APPLE – Represents Apple as an SSO method.

### 2. ****UserDTO****

* **Attributes:**
* name: String – The name of the user.
* email: String – The email of the user.
* password: String – The password of the user.

### 3. ****Credentials****

* **Attributes:**
* email: String – The email used for authentication.
* password: String – The password used for authentication.

### 4. ****Tokens****

* **Attributes:**
* accessToken: String – The access token for authenticating requests.
* refreshToken: String – The refresh token used to obtain a new access token.

### 5. ****Role**** (Enum)

* **Values:**
* OWNER – Represents the owner role.
* ADMIN – Represents the admin role.
* EDITOR – Represents the editor role.
* USER – Represents the user role.
* GUEST – Represents the guest role.

### 6. ****ErrorCodes**** (Enum)

* **Values:**
* USER\_NOT\_FOUND – Error when the user is not found.
* INVALID\_CREDENTIALS – Error for invalid user credentials.
* BOOK\_NOT\_FOUND – Error when the book is not found.
* PAYMENT\_FAILED – Error when the payment fails.
* UNAUTHORIZED\_ACCESS – Error for unauthorized access.
* REFRESH\_TOKEN\_EXPIRED – Error when the refresh token expires.
* ACCESS\_TOKEN\_EXPIRED – Error when the access token expires.
* INVALID\_PASSWORD – Error for an invalid password.
* EMAIL\_ALREADY\_EXISTS – Error when the email already exists.
* USER\_ALREADY\_EXISTS – Error when the user already exists.
* BOOK\_ALREADY\_EXISTS – Error when the book already exists.
* INSUFFICIENT\_FUNDS – Error for insufficient funds.
* FORBIDDEN\_ACCESS – Error for forbidden access.
* BAD\_REQUEST – Error for a bad request.
* NOT\_FOUND – Error for a resource that was not found.
* INTERNAL\_SERVER\_ERROR – General internal server error.

### 7. ****BookDTO****

* **Attributes:**
* title: String – The title of the book.
* author: String – The author of the book.
* genre: String – The genre of the book.
* cover: String – The cover image of the book.
* audioVersion: mp3 – The audio version of the book.
* text: String – The text content of the book.
* isPaid: bool – Indicates if the book is paid.
* price: float – The price of the book.

### 8. ****FileType**** (Enum)

* **Values:**
* EPUB – EPUB file format.
* FB2 – FB2 file format.
* TXT – TXT file format.
* PDF – PDF file format.

### 9. ****BookmarkDTO****

* **Attributes:**
* color: String – The color of the bookmark.
* text: String – The text associated with the bookmark.

### 10. ****ReviewDTO****

* **Attributes:**
* rating: int – The rating of the book (usually between 1 and 5).
* text: String – The review text for the book.

### 11. ****PaymentDTO****

* **Attributes:**
* userId: String – The user ID associated with the payment.
* amount: float – The amount of the payment.
* cardNumber: String – The card number used for the payment.

### 12. ****ListType**** (Enum)

* **Values:**
* LIKED – Indicates the book is liked.
* WANT – Indicates the book is on the user's "want to read" list.
* READING – Indicates the book is currently being read.
* READ – Indicates the book has been read.

### 13. ****LoginMode**** (Enum)

* **Values:**
* LOGIN – Represents the login mode.
* REGISTER – Represents the registration mode.

### 14. ****EditorForm****

* **Attributes:**
* alias: String – The alias of the editor.
* mainGenre: String – The main genre the editor specializes in.
* whyBecomeEditor: String – The reason for wanting to become an editor.
* experience: int – The number of years of experience the editor has.

### 15. ****SortOrder**** (Enum)

* **Values:**
* ASCENDING – Represents the ascending sort order.
* DESCENDING – Represents the descending sort order.

### ****ComponentProps****

### 16. ****ProfileFormProps****

* **Attributes:**
* userProfile: UserDTO – The user's profile data.
* onSubmit: (data: UserDTO) => void – A function to submit the user data.

### 17. ****BookSearchProps****

* **Attributes:**
* filters: String – The filters applied to the book search.
* searchQuery: String – The search query for books.
* onSearch: (query: String) => void – A function to handle the search action.

### 18. ****BookViewerProps****

* **Attributes:**
* book: BookDTO – The book to be viewed.
* currentPage: number – The current page of the book.
* onPageChange: (page: number) => void – A function to handle page changes.

### 19. ****PaymentFormProps****

* **Attributes:**
* paymentData: PaymentDTO – The payment data to be submitted.
* onSubmit: (paymentData: PaymentDTO) => void – A function to submit the payment data.

### 20. ****ReviewFormProps****

* **Attributes:**
* bookId: String – The ID of the book being reviewed.
* reviewText: String – The review text for the book.
* rating: number – The rating for the book.
* onSubmit: (reviewData: ReviewDTO) => void – A function to submit the review.

### 21. ****AdminBookListProps****

* **Attributes:**
* books: List<BookDTO> – A list of books.
* onUpdateBook: (bookId: String, bookData: BookDTO) => void – A function to update a book.
* onDeleteBook: (bookId: String) => void – A function to delete a book.

### 22. ****AdminUserListComponent****

* **Attributes:**
* users: List<UserDTO> – A list of users.
* onChangeRole: (userId: String, role: Role) => void – A function to change a user's role.
* onDeleteUser: (userId: String) => void – A function to delete a user.

### 23. ****EditorDraftListProps****

* **Attributes:**
* draftBooks: List<BookDTO> – A list of draft books.
* onSubmitBook: (bookId: String) => void – A function to submit a draft book.

### ****PageProps****

### 24. ****AuthPageProps****

* **Attributes:**
* mode: LoginMode – The login or registration mode.

### 25. ****PaymentPageProps****

* **Attributes:**
* userRole: Role – The role of the user (ADMIN, USER, etc.).

### 26. ****ReviewPageProps****

* **Attributes:**
* bookId: string – The ID of the book to be reviewed.

### 27. ****ProfilePageProps****

* **Attributes:**
* userRole: Role – The role of the user (ADMIN, USER, etc.).

### 28. ****BookSearchPageProps****

* **Attributes:**
* filters: string – The filters applied to the book search.

### 29. ****BookViewerPageProps****

* **Attributes:**
* bookId: string – The ID of the book to be viewed.
* userRole: Role – The role of the user viewing the book.