Documentation of Book Addition and Book Display

Note - Getter, Setter functions, and Constructors have their usual meanings and are not mentioned here.

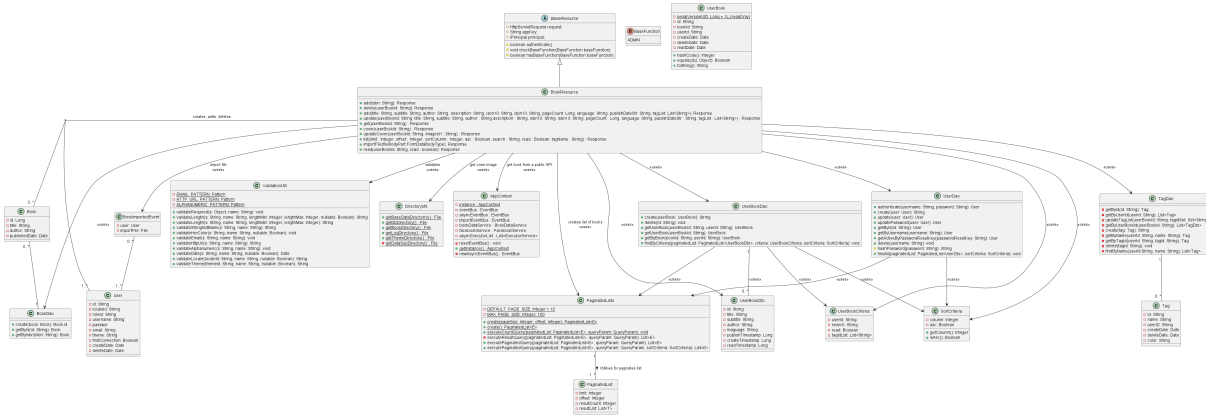

Figure 1: Class Diagram for Book Addition and Display Subsystem

The Bookresource Managenent system manages addition, deletion of books with respect to users. It is managed by the BookResource controller which inherits BaseResource. It uses BookDao for database access. It uses BookDto for data transfer in the backend. It uses Book class as the entity class which is mapped with the database table.

BookResource

The **BookResource** class is responsible for managing book-related operations in the application. It extends the **BaseResource** class, utilizing common functionalities such as authentication checks. This class provides endpoints for adding new books, deleting books, updating book information, fetching book details, managing book covers, and importing books.

Methods

- **Response add(`String isbn`)** [**PUT /book**]: Adds a new book to the database using the provided ISBN number. Validates the input and fetches or creates the book and user book records.
- **Response delete(`String userBookId`)** [**DELETE /book/{id}**]: Deletes a book identified by the user book ID. Validates user authentication and ownership before deletion.
- **Response addManual(`String title, String subtitle, String author, String description, String isbn10, String isbn13, Long pageCount, String language, String publishDateStr, List<String> tagList`)** [**PUT /book/manual**]: Adds a book manually with detailed information. Validates each input field and creates a new book and user book record in the database.
- **Response update(`String userBookId, String title, String subtitle, String author, String description, String isbn10, String isbn13, Long pageCount, String language, String publishDateStr, List<String> tagList`)** [**POST /book/{id}**]: Updates the information of an existing book using the user book ID. Validates user authentication and provided details before updating the book and user book records.
- **Response get(`String userBookId`)** [**GET /book/{id}**]: Fetches the details of a book using the user book ID. Validates user authentication and ownership before returning book details.
- **Response cover(`String userBookId`)** [**GET /book/{id}/cover**]: Retrieves the cover image of a book using the user book ID. Validates the existence of a cover image and serves it, or defaults to a placeholder image if not available.
- **Response updateCover(`String userBookId, String imageUrl`)** [**POST /book/{id}/cover**]: Updates the cover image of a book using the user book ID and a new image URL. Validates user authentication and downloads the new image to replace the existing cover.
- **Response list(`Integer limit, Integer offset, Integer sortColumn, Boolean asc, String search, Boolean read, String tagName`)** [**GET /book/list**]: Lists all books based on pagination parameters, sorting, and filtering criteria such as search term, read status, and tag name. Validates user authentication and applies filters accordingly.
- **Response importFile(`FormDataBodyPart fileBodyPart`)** [**PUT /book/import**]: Imports books from a provided file. Validates the file input and initiates the import process, handling any errors encountered during import.
- **Response read(`String userBookId, boolean read`)** [**POST /book/{id}/read**]: Marks a book as read or unread using the user book ID and the read state. Validates user authentication and updates the read status accordingly.

BaseResource

The **BaseResource** class serves as the abstract base class for REST resources in the application, providing common functionalities such as authentication checks and permission validations.

Attributes

- `protected HttpServletRequest request`: Injects the HTTP request to allow access to request information within resource classes.
- `protected String appKey`: A query parameter that might be used for application-level authentication or API key validation.
- `protected IPrincipal principal`: Represents the principal of the authenticated user, providing an interface to user identity and roles.

Methods

- **boolean authenticate()**: Checks if the user is authenticated and not anonymous. It retrieves the user principal from the request attributes, set by a security filter, and checks if the user is not marked as anonymous.
- **void checkBaseFunction(**`BaseFunction baseFunction`**)**: Validates if the authenticated user has a specific base function (role or permission). Throws `JSONException` if the user does not have the required base function.
- **boolean hasBaseFunction(**`BaseFunction baseFunction`**)**: Checks if the authenticated user has a specific base function. Returns a boolean indicating if the user has the specified base function. Throws `JSONException`.

Book

The **Book** class represents a book entity.

Attributes

- `id`: Book ID.
- `title`: Title of the book.
- `subtitle`: Subtitle of the book (optional).
- `author`: Author of the book.
- `description`: Description of the book (optional).
- `isbn10`: ISBN-10 of the book (optional).
- `isbn13`: ISBN-13 of the book (optional).
- `pageCount`: Page count of the book (optional).
- `language`: Language of the book (ISO 639-1 code).
- `publishDate`: Publication date of the book.

User

The **User** class represents a user entity in the system, containing information such as user ID, username, password, email, and creation date.

Attributes

- `String id`: User ID.
- `String localeId`: Locale ID.
- `String roleId`: Role ID.
- `String username`: User's username.
- `String password`: User's password.
- `String email`: Email address.
- `String theme`: Theme.
- `boolean firstConnection`: True if the user hasn't dismissed the first connection screen.
- `Date createDate`: Creation date.
- `Date deleteDate`: Deletion date (optional).

Methods

- **`boolean isFirstConnection()`**: Returns true if the user hasn't dismissed the first connection screen.

BookDao

The **BookDao** class is a data access object, designed for managing the persistence and retrieval operations related to **Book** entities within the application. It utilizes the Java Persistence API (JPA) EntityManager for database interactions, supporting operations such as creating new books, fetching books by ID, and retrieving books by their ISBN numbers.

Methods

- **String create(`Book book`)**: Persists a new book entity to the database. The `book` parameter is the **Book** entity to be persisted. Returns the ID of the newly created book.
- **Book getById(`String id`)**: Retrieves a book entity based on its ID. The `id` parameter is the unique identifier of the book. Returns the **Book** entity if found, or `null` if no book is found with the given ID.
- **Book getByIsbn(`String isbn`)**: Fetches a book entity based on its ISBN number, which could be either ISBN-10 or ISBN-13. The `isbn` parameter is the ISBN number of the book. Returns the **Book** entity if found, or `null` if no book matches the given ISBN.

UserDao

The **UserDao** class is a data access object, responsible for managing user-related database operations, such as authentication, creation, updating, deletion, and retrieval of user information.

Methods

- **String authenticate**(`String username`, `String password`): Authenticates a user given the username and password. Returns the ID of the authenticated user or null if authentication fails.
- **String create**(`User user`): Creates a new user in the database. Throws an exception if the username already exists. Returns the ID of the newly created user.
- **User update**(`User user`): Updates the information of an existing user in the database.
- **User updatePassword**(`User user`): Updates the password of an existing user in the database.
- **User getById**(`String id`): Retrieves a user from the database based on the user ID.
- **User getActiveByUsername**(`String username`): Retrieves an active user from the database based on the username.
- **User getActiveByPasswordResetKey**(`String passwordResetKey`): Retrieves an active user from the database based on the password reset token.
- **void delete**(`String username`): Deletes a user from the database based on the username.
- **void findAll**(`PaginatedList<UserDto> paginatedList`, `SortCriteria sortCriteria`): Retrieves a paginated list of all users from the database, including their ID, username, email, creation timestamp, and locale ID.

UserBookDao

The **UserBookDao** class is a data access object, responsible for handling all database interactions related to **UserBook** entities within the application. It provides functionality for creating new user books, deleting existing ones, and querying user books based on various criteria.

Methods

- **String create(**`UserBook userBook`**)**: Persists a new **UserBook** entity to the database, assigning it a unique identifier. Returns the identifier of the newly created user book.
- **void delete(**`String id`**)**: Removes a **UserBook** entity from the database by marking it as deleted, using its unique identifier.
- **UserBook getUserBook(**`String userBookId, String userId`**)**: Retrieves a single **UserBook** entity based on its identifier and the user's identifier, ensuring that it has not been marked as deleted.
- **UserBook getUserBook(**`String userBookId`**)**: Fetches a **UserBook** entity by its identifier, without filtering by user, provided it has not been marked as deleted.
- **UserBook getByBook(**`String bookId, String userId`**)**: Finds a **UserBook** entity based on the book's identifier and the user's identifier, ensuring that the book is associated with the user and has not been marked as deleted.
- **void findByCriteria(**`PaginatedList<UserBookDto> paginatedList, UserBookCriteria criteria, SortCriteria sortCriteria`**)**: Executes a search for **UserBook** entities based on a set of criteria including pagination and sorting parameters. Updates the provided `paginatedList` with the results.

UserBookDto

The **UserBookDto** class represents a Data Transfer Object (DTO) used for transferring user book data between different layers of the application.

Attributes

- `id`: User book ID.
- `title`: Title of the book.
- `subtitle`: Subtitle of the book.
- `author`: Author of the book.
- `language`: Language of the book (ISO 639-1).
- `publishTimestamp`: Timestamp representing the publication date of the book.
- `createTimestamp`: Timestamp representing the creation date of the user book entry.
- `readTimestamp`: Timestamp representing the date when the user read the book.

UserBook

The **UserBook** entity class represents the relationship between a user and a book within the application. It includes information such as the unique ID of the user book, the ID of the book, the ID of the user, creation date, deletion date, and read date.

Attributes

- `String id`: The unique identifier for the user book.
- `String bookId`: The identifier for the associated book.
- `String userId`: The identifier for the user who owns the book.
- `Date createDate`: The date when the user book relationship was created.
- `Date deleteDate`: The date when the user book relationship was deleted (soft delete).
- `Date readDate`: The date when the book was marked as read by the user.

UserBookCriteria

The **UserBookCriteria** class is utilized as a criteria object for filtering and querying user book records. It encapsulates various criteria such as user ID, search query, read state, and tag IDs to support flexible searching and filtering operations on user books.

Attributes

- `String userId`: The ID of the user to which the books belong.
- `String search`: A search query to filter books based on text matching in titles, authors, or descriptions.
- `Boolean read`: The read state of the books, where `true` represents books that have been read and `false` represents books that have not been read.
- `List<String> tagIdList`: A list of tag IDs to filter books associated with specific tags.

SortCriteria

The **SortCriteria** class represents the sort criteria of a query.

Attributes

- `int column`: Index of the column to sort (starting from 0).
- `boolean asc`: Indicates whether to sort in increasing order (`true`) or decreasing order (`false`).

Methods

- **getColumn(): int**: Returns the index of the column to sort.
- **isAsc(): boolean**: Returns `true` if sorting is in increasing order, `false` if sorting is in decreasing order.

PaginatedList<T>

The **PaginatedList<T>** class represents a paginated list of items.

Attributes

- `int limit`: The size of a page.
- `int offset`: The offset of the page (in number of records).
- `int resultCount`: The total number of records.
- `List<T> resultList`: The list of records of the current page.

PaginatedLists

The **PaginatedLists** class provides utilities for working with paginated lists.

Attributes

- `int DEFAULT_PAGE_SIZE`: The default size of a page.
- `int MAX_PAGE_SIZE`: The maximum size of a page.

Methods

- **create(**`Integer pageSize, Integer offset`**)**: Constructs a paginated list with the specified page size and offset. If `pageSize` is `null`, the default page size is used. If `offset` is `null`, the offset is set to 0.
- **create():** Constructs a paginated list with default parameters.
- **executeCountQuery(**`PaginatedList<E> paginatedList, QueryParam queryParam`**)**: Executes a native `count(*)` request to count the number of results and sets the result count in the `paginatedList` object.
- **executePaginatedQuery(**`PaginatedList<E> paginatedList, QueryParam queryParam`**)**: Executes a paginated request with two native queries (one to count the number of results and one to return the page) and returns the list of results.
- **executePaginatedQuery(**`PaginatedList<E> paginatedList, QueryParam queryParam, SortCriteria sortCriteria`**)**: Executes a paginated request with two native queries (one to count the number of results and one to return the page) and returns the list of results sorted according to the `sortCriteria`.

Tag

The **Tag** class represents the tag entity within the application. It is used to categorize books by assigning them tags for easier searching and sorting. Tags are associated with users, allowing for personalized categorization.

Attributes

- `String id`: The unique identifier for the tag.
- `String name`: The name of the tag.
- `String userId`: The ID of the user to whom the tag belongs.
- `Date createDate`: The creation date of the tag.
- `Date deleteDate`: The deletion date of the tag, used to soft delete the tag.
- `String color`: The color associated with the tag for UI representation.

TagDao

The **TagDao** class provides methods for handling database operations related to tags.

Methods

- **Tag getById(String id)**: Gets a tag by its ID.
- **List<Tag> getByUserId(String userId)**: Returns the list of all tags associated with a user.
- **void updateTagList(String userBookId, Set<String> tagIdSet)**: Updates tags on a user book.
- **List<TagDto> getByUserBookId(String userBookId)**: Returns the list of tags associated with a user book.
- **String create(Tag tag)**: Creates a new tag.
- **Tag getByName(String userId, String name)**: Returns a tag by name.
- **Tag getByTagId(String userId, String tagId)**: Returns a tag by ID.
- **void delete(String tagId)**: Deletes a tag.
- **List<Tag> findByName(String userId, String name)**: Searches tags by name.

ValidationUtil

The **ValidationUtil** class provides utility methods for validating parameters, such as strings, emails, URLs, and dates. It also includes methods for validating locales and themes.

Methods

- **void validateRequired**(`Object s`, `String name`): Checks that the argument is not null.
- **String validateLength**(`String s`, `String name`, `Integer lengthMin`, `Integer lengthMax`, `boolean nullable`): Validates a string length.
- **String validateLength**(`String s`, `String name`, `Integer lengthMin`, `Integer lengthMax`): Validates a string length. The string mustn't be empty.
- **String validateStringNotBlank**(`String s`, `String name`): Checks if the string is not null and is not only whitespaces.
- **void validateHexColor**(`String s`, `String name`, `boolean nullable`): Checks if the string is a hexadecimal color.
- **void validateEmail**(`String s`, `String name`): Checks if the string is an email.
- **String validateHttpUrl**(`String s`, `String name`): Validates that the provided string matches an URL with HTTP or HTTPS scheme.
- **void validateAlphanumeric**(`String s`, `String name`): Checks if the string uses only alphanumerical or underscore characters.
- **Date validateDate**(`String s`, `String name`, `boolean nullable`): Validates and parses a date.
- **String validateLocale**(`String localeId`, `String name`, `boolean nullable`): Validates a locale.
- **String validateTheme**(`String themeId`, `String name`, `boolean nullable`): Validates a theme.

DirectoryUtil

The **DirectoryUtil** class provides utilities to gain access to the storage directories used by the application.

Methods

- **File getBaseDataDirectory()**: Returns the base data directory.
- **File getDbDirectory()**: Returns the database directory.
- **File getBookDirectory()**: Returns the book covers directory.
- **File getLogDirectory()**: Returns the log directory.
- **File getThemeDirectory()**: Returns the themes directory.
- **File getDataSubDirectory(`String subdirectory`)**: Returns a subdirectory of the base data directory.

AppContext

The **AppContext** class represents the global application context, providing access to event buses, services, and executors.

Fields

- `instance` (**AppContext**): Singleton instance of the application context.
- `eventBus` (**EventBus**): Event bus for synchronous events.
- `asyncEventBus` (**EventBus**): Generic asynchronous event bus.
- `importEventBus` (**EventBus**): Asynchronous event bus specifically for mass imports.
- `bookDataService` (**BookDataService**): Service to fetch book information.
- `facebookService` (**FacebookService**): Facebook interaction service.
- `asyncExecutorList` (**List<ExecutorService>**): List of asynchronous executors.

Methods

- **private void resetEventBus()**: (Re)-initializes the event buses.
- **public static AppContext getInstance()**: Returns a single instance of the application context.
- **private EventBus newAsyncEventBus()**: Creates a new asynchronous event bus.

BookImportedEvent

The **BookImportedEvent** class represents an event raised on request to import books. It contains information about the user requesting the import and the temporary file to import.

Fields

- `user` (**User**): User requesting the import.
- `importFile` (**File**): Temporary file to import.

Methods

- **public String toString()**: Returns a string representation of the object, including user and import file information.

Relationships

1. **User ”1..*” – ”0..*” Book**

   Represents a one-to-many relationship between users and books, where a user can have multiple books, and a book can belong to multiple users.

2. **(User, Book) .. BookResource**

   BookResource interacts with both User and Book entities, performing operations like creating, adding, and deleting books.

3. **BookResource –> ValidationUtil: validates**

   BookResource utilizes ValidationUtil for validating input data.

4. **BookResource ”1” *– ”1” BookDao**

   BookResource and BookDao have a composition relationship. BookResource uses BookDao to access information related to the Book entity from the database.

5. **Book ”0..*” –o ”1” BookDao**

   BookDao interacts with Book entities to persist data in the database.

6. **BookResource –> AppContext**:

   BookResource utilizes AppContext to fetch books from a public API.

7. **BookResource ”1” *– ”1” UserBookDao**

   BookResource and UserBookDao have a composition relationship. BookResource uses UserBookDao to access information related to the UserBook entity from the database.

8. **UserBookDao ”1” o– ”0..*” UserBook**

   Represents an aggregation relationship between UserBookDao and UserBook.

9. **UserBookDao ”1” – ”0..*” UserBookDto**

   UserBookDao utilizes UserBookDto for data transfer related to user-book entities.

10. **UserBookDao –> UserBookCriteria**

    UserBookDao utilizes UserBookCriteria to define criteria for querying user-book data.

11. **UserBookDao –> PaginatedLists**

    UserBookDao utilizes PaginatedLists for paginating lists of user-book data.

12. **UserBookDao –> SortCriteria**

    UserBookDao utilizes SortCriteria for specifying sorting criteria.

13. **BookResource ”1” *– ”1” UserDao**

    BookResource and UserDao have a composition relationship. BookResource uses UserDao to access information related to the User entity from the database.

14. **BookResource ”1” *– ”1” TagDao**

    BookResource and TagDao have a composition relationship. BookResource uses TagDao to access information related to the Tag entity from the database.

15. **TagDao ”1” o– ”0..*” Tag**

    Represents an aggregation relationship between TagDao and Tag.

16. **BookResource –> DirectoryUtil**

    BookResource uses DirectoryUtil to retrieve cover images.

17. **BookResource –> PaginatedLists**:

    BookResource utilizes PaginatedLists for creating paginated lists.

18. **PaginatedLists "1" – "1..*" PaginatedList**

    PaginatedLists is responsible for managing one or more PaginatedList instances.

19. **BookResource –> SortCriteria**

    BookResource uses SortCriteria for specifying sorting criteria.

20. **BookResource –> UserBookCriteria** BookResource utilizes UserBookCriteria for defining criteria.

21. **BookResource –> UserBookDto**

    BookResource utilizes UserBookDto for data transfer related to user-book entities.

22. **BookResource –> BookImportedEvent**:

    BookResource generates BookImportedEvent to handle the import of files, indicating a user's action.

23. **BookImportedEvent "0..*" – "1..*" User**

    BookImportedEvent is associated with multiple User entities, indicating that multiple users may trigger import events.

24. **UserDao –> PaginatedLists**

    UserDao utilizes PaginatedLists for handling paginated lists.

25. **UserDao –> SortCriteria**

    UserDao uses SortCriteria for specifying sorting criteria.

26. **(BookResource, TagDao) .. TagDto**:

    Both BookResource and TagDao are associated with TagDto, indicating that TagDto is used for data transfer between these components.