# Documentation of Bookshelf Management Subsystem

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

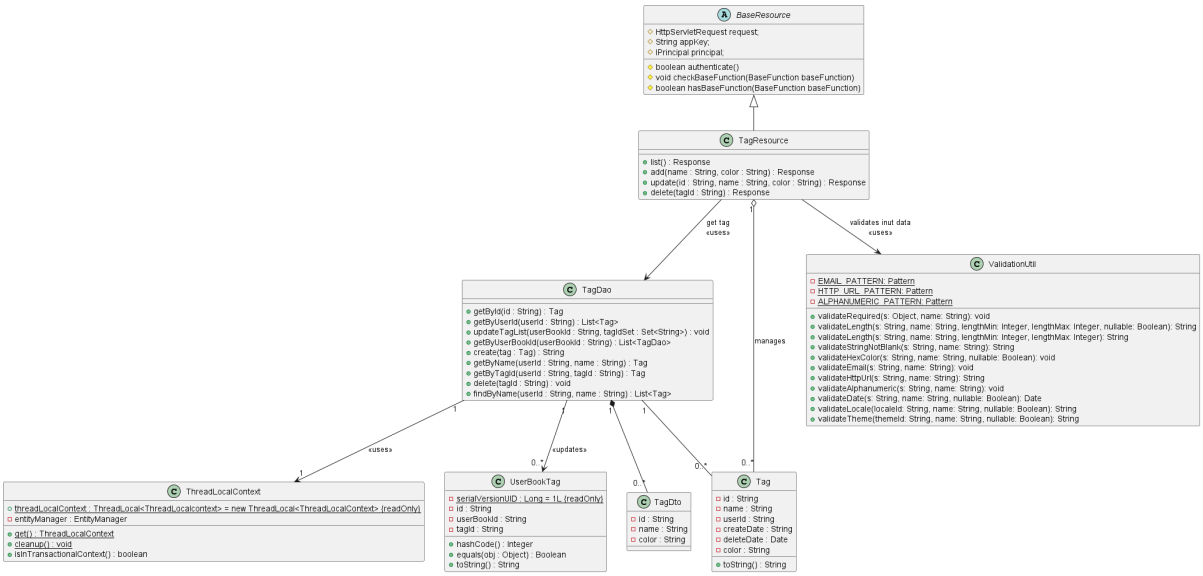


Figure 1: Class Diagram for Bookshelf Management Subsystem

The management of bookshelves is through maintaining a list of tags, each of which corresponds to a bookshelf. It uses the TagResource class as a controller, which inherits from BaseResource. It utilizes TagDao for database access and Tag and UserBookTag as the entity classes, which is mapped to the database tables.

# TagResource

The **TagResource** class is responsible for providing REST endpoints for managing tags within the application. It supports operations such as listing all tags for a user, creating new tags, updating existing tags, and deleting tags.

## Methods

- **Response list()** [**GET /tag/list**]: Retrieves a list of all tags associated with the authenticated user. It returns a JSON response containing an array of tag objects.
- **Response add(`String name, String color`)** [**PUT /tag**]: Creates a new tag with the specified name and color for the authenticated user. Validates the input data and ensures that tag names are unique within the user's scope. Returns the ID of the newly created tag.
- **Response update(`String id, String name, String color`)** [**POST /tag/{id}**]: Updates the specified tag belonging to the authenticated user. It allows changing the tag's name and color, validating the input data and ensuring that the updated name remains unique. Returns the ID of the updated tag.
- **Response delete(`String tagId`)** [**DELETE /tag/{id}**]: Deletes the specified tag by its ID for the authenticated user. Validates that the tag exists and belongs to the user before deleting it. Returns a success status upon completion.

# 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`.

# 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.

# UserBookTag

The **UserBookTag** class represents the link between a user's book and a tag within the application. It is used to categorize books into various tags for better organization and filtering.

## Attributes

- `String id`: The unique identifier of the book tag link.
- `String userBookId`: The identifier of the user book associated with this tag.
- `String tagId`: The identifier of the tag linked to the user book.

# ThreadLocalContext

The **ThreadLocalContext** class encapsulates the application context associated with a user request, making it available throughout the processing of the request via thread-local storage. It primarily manages the lifecycle of the **EntityManager** within the context of the request.

## Attributes

- `ThreadLocal<ThreadLocalContext> threadLocalContext`: A thread-local variable that stores instances of the **ThreadLocalContext** class.
- `EntityManager entityManager`: The **EntityManager** associated with the current thread's request context.

## Methods

- **static ThreadLocalContext get()**: Retrieves the current thread's instance of **ThreadLocalContext**, creating it if it does not exist. This method ensures that each thread has its own instance of the context.
- **static void cleanup()**: Cleans up the thread-local variable by removing the current thread's instance of **ThreadLocalContext**. This is typically called at the end of request processing to prevent memory leaks.
- **boolean isInTransactionalContext()**: Determines whether the current thread's context is within a transactional boundary by checking the presence of an **EntityManager**. Returns true if the **EntityManager** is not null.
- **EntityManager getEntityManager()** and **void setEntityManager(`EntityManager entityManager`)**: Gets and sets the **EntityManager** associated with the current thread's request context. These methods manage the lifecycle of the **EntityManager** within the request scope.

# TagDto

The **TagDto** class is a Data Transfer Object designed for transferring tag data between layers of the application, particularly useful in RESTful services where tags need to be serialized to and deserialized from JSON. It encapsulates the tag's ID, name, and color.

## Attributes

- `String id`: The unique identifier of the tag.
- `String name`: The name of the tag.
- `String color`: The hexadecimal color code representing the tag color for UI purposes.

## Relationships

1. **User "1.." – "0.." Book**

   Each user can have multiple books, but a book may not necessarily belong to any user.

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

   The BookResource class serves as the interface for creating, adding, and deleting books. It handles these operations via REST endpoints.

3. **TagResource "1" *– "1" TagDao**

   TagResource and TagDao have a composition relationship. TagResource uses TagDao to to retrieve/get information related to the Tag entity.

4. **TagResource –> ValidationUtil**

   TagResource utilizes ValidationUtil to validate input data before processing it further. This ensures that the input adheres to certain criteria or constraints.

5. **TagDao "1" –> "0..* UserBookTag**

   TagDao is associated with UserBookTag, indicating that it updates instances of UserBookTag when performing certain operations. This suggests that UserBookTag might be updated when tags are modified or deleted.

6. **TagDao "1" –> "1" ThreadLocalContext**

   TagDao interacts with ThreadLocalContext, implying that it uses thread-local context information during its operations.

7. **TagResource –> TagDto**

   TagResource uses TagDto for transferring tag-related data. This suggests that instances of TagDto are employed to exchange information between TagResource and other components.

8. **BookResource "1" *– "1" TagDao**

   Each instance of BookResource is composed of TagDao.

9. **BookResource "1" –> "1" ValidationUtil**

   BookResource utilizes ValidationUtil for input data validation purposes. This indicates that ValidationUtil plays a crucial role in ensuring the integrity and validity of the data processed by BookResource.

10. **TagDao "1" o– "0..*" Tag**

    TagDao and Tag have an aggregation relationship between them.