Task 1

1. Book Addition & Display Subsystem:

BookResource

- Extends the BaseResource class which has basic stuff for handling HTTP requests, sockets, etc.
- Implements methods to add, remove, and edit books that exist for the user.
- Interacts with several DAO objects to perform the same

PaginatedList

- simple representation of a paginated list, commonly used in applications dealing with large datasets that need to be retrieved in smaller, manageable portions.
- defines a generic class `PaginatedList<T>`, which is used to handle paginated lists of objects of type `T`.
- Used for retrieving large data in chunk or pages. Four instance variables: limit, offset, resultCount, resultList.

`limit` - max number of records a page can contain
`offset` - starting point in a list of records for the page
`resultCount` - total number of records in the list
`resultList` - List of results

- Considering that pagination parameters are typically set once upon instantiation and remain constant, making them immutable (e.g., through constructor parameters without corresponding setters) could enhance the class's reliability and maintainability.

PaginatedLists

- Creates and handles PaginatedList object
- Provides utilities for working with paginated lists in the context of Java Persistence API (JPA).
- It allows the creation of paginated lists with specified page size and offset, ensuring efficient retrieval of data from large result sets.

BookImportedEvent

- Represents an event raised when a request is made to import books.
- Generates a string of the form BookImportedEvent{user=value, importFile=value} for pushing onto the Import bus

UserBookCriteria

- Represents criteria for filtering user book data.
- Stores userId, search query, read state, tagIdList for storing the criteria.
- Validation logic for ensuring the correctness of criteria values could be added to setter methods to prevent invalid criteria from being set.

SortCriteria

- Stores the sorting parameter (which column to sort, asc/desc)
- The class lacks mutator methods (setters) for its attributes, which promotes immutability. Once created, an instance of `SortCriteria` cannot be modified, ensuring consistency and predictability.

UserDAO

- Serves as a Data Access Object for managing user-related data in the application's database. It provides methods to interact with the `User` entity, including authentication, creation, retrieval, updating, and deletion of user records.
- **Methods**:
  1. `authenticate(String username, String password)`: Authenticates a user by verifying the provided username and password against the database records. It uses BCrypt for secure password hashing.
  2. `create(User user)`: Creates a new user record in the database. It generates a UUID for the user ID, checks for username uniqueness, hashes the password, sets default values for certain attributes, and persists the user object.
  3. `update(User user)`: Updates an existing user record in the database. It retrieves the user from the database, updates the relevant attributes, and returns the updated user object.
  4. `updatePassword(User user)`: Updates the password of an existing user in the database. It retrieves the user from the database, updates the password with a hashed version, and returns the updated user object.
  5. `getById(String id)`: Retrieves a user record from the database by its unique identifier (`id`).
  6. `getActiveByUsername(String username)`: Retrieves an active user record from the database by its username.
  7. `getActiveByPasswordResetKey(String passwordResetKey)`: Retrieves an active user record from the database by its password reset token.
  8. `delete(String username)`: Deletes a user record from the database by its username. It marks the user as deleted and deletes any associated data, such as authentication tokens.
  9. `hashPassword(String password)`: Hashes the provided password using the BCrypt hashing algorithm for secure storage in the database.
  10. `findAll(PaginatedList<UserDto> paginatedList, SortCriteria sortCriteria)`: Retrieves a paginated list of all user records from the database. It constructs and executes a query to fetch user data, assembles the results into DTOs (Data Transfer Objects), and populates the provided paginated list.
- **Dependencies**:
  - `EntityManager`: Manages the lifecycle of user entities within the persistence context.
  - `ThreadLocalContext`: Provides access to the `EntityManager` associated with the current thread.
  - `BCrypt`: Library for secure password hashing.
  - `Constants`: Contains application-wide constant values.
  - `PaginatedList`, `PaginatedLists`, `QueryParam`, `SortCriteria`: Utility classes for handling pagination, query parameters, and sorting in JPA queries.

UserDto

- Serves as a lightweight representation of user data, facilitating data transfer between different layers of the application. Here's a brief overview:
- Enables efficient transfer of user data between application layers.
- Encapsulates essential user information for specific operations.
- Typically employed for transferring user details to presentation layer components. Enhances maintainability and performance by controlling data granularity.

UserBookDAO

- A Data Access Object (DAO) responsible for interacting with the database to perform CRUD (Create, Read, Update, Delete) operations related to user books.
- Relies on the entity manager injected via the `ThreadLocalContext`, ensuring proper management of database transactions and resources.
- The method `findByCriteria` is quite complex due to the dynamic construction of the query based on multiple criteria. While this offers flexibility, it can make maintenance challenging, especially when modifying or extending the search logic.
- Pagination is implemented using the `PaginatedList` and `PaginatedLists` utility classes, enhancing performance and usability when dealing with large result sets.

BookDAO

- Responsible for CRUD operations related to books in the database.
- The class provides methods for creating a new book and retrieving books by their ID or ISBN number.
- The `create` method persists a new book entity to the database using the entity manager obtained from the thread-local context. It then returns the ID of the newly created book.
- The `getById` method retrieves a book by its ID using the `EntityManager.find` method, which is efficient for primary key lookups.
- The `getByIsbn` method retrieves a book by its ISBN number (either ISBN-10 or ISBN-13) using a JPQL query.
- Relies on the entity manager injected via the `ThreadLocalContext`, ensuring proper management of database transactions and resources.
- Proper error handling is implemented using try-catch blocks to handle potential `NoResultException` when retrieving books by ID or ISBN.

TagDAO

- The class provides methods for creating, reading, updating, and deleting tags.
- The `create` method persists a new tag entity to the database, generating a UUID for the tag ID and setting the creation date.
- Methods like `getById`, `getByUserBookId`, `getByName`, and `getByTagId` retrieve tags based on different criteria such as tag ID, user ID, or tag name.
- The `delete` method marks a tag as deleted by setting the deletion date and also deletes any linked data (user book tag associations) to ensure data integrity.
- The class contains methods like `getByUserBookId` that convert raw database query results into DTOs (Data Transfer Objects), making it easier to work with the data in the application layer.
- The `updateTagList` method allows updating the tags associated with a user book. It first deletes the existing tag links and then creates new tag links based on the provided tag IDs.

BookDataService

- Responsible for fetching book information from external APIs like Google Books and Open Library.
- The class uses `RateLimiter` from Guava to limit the rate of API requests made to Google Books and Open Library APIs. This prevents hitting API rate limits and potential service degradation.
- The `searchBook()` method searches for book information using the provided ISBN. It first attempts to search using the Google Books API and falls back to the Open Library API if no results are found.
- API requests are made asynchronously using `ExecutorService` and `FutureTask`. This allows for concurrent execution of API requests, improving performance and responsiveness.
- The class relies on external libraries such as Guava, Jackson, and Joda-Time for functionality related to rate limiting, JSON parsing, and date parsing, respectively.

DirectoryUtil

- Provides utility methods for accessing various directories used by the application.
- The `getBaseDataDirectory()` method returns the base data directory used by the application. It checks if the application is running in a webapp environment and if the `books.home` property is set. If not, it defaults to using a directory named "sismicsbooks" in the webapp root directory or falls back to the system's temporary directory.
- EnvironmentUtil is used to detect whether the application is running in a webapp environment and to retrieve the webapp root directory.
- For the theme directory, if the application is not running in a webapp environment, it attempts to load the directory from the classpath.

Book

- Represents the entity model for books in the application's database.
- It defines attributes such as `id`, `title`, `subtitle`, `author`, `description`, `isbn10`, `isbn13`, `pageCount`, `language`, and `publishDate`, representing various properties of a book.

User

- Represents a user entity in the JPA context.
- Contains fields representing user attributes.
- Has storage of user-specific settings, such as language and theme.

Tag

- Represents the entity model for tags in the application's database.
- It defines attributes such as `id`, `name`, `userId`, `createDate`, `deleteDate`, and `color`, representing various properties of a tag.
- Getter and setter methods are provided for each attribute to access and modify the state of the object.

AppContext

- Has various buses (EventBus, asyncEventBus, importEventBus, bookDataService, etc.)
- Has Singleton pattern (private constructor)

BookDao

- This Java class BookDao is a Data Access Object (DAO) responsible for handling database operations related to the Book entity in a Java Persistence API (JPA) environment.
- Some of the methods are as follows:
    1. create(Book book): This method creates a new book entry in the database. It persists the provided Book entity using the entity manager obtained from the thread-local context and returns the ID of the newly created book.
    2. getById(String id): This method retrieves a book from the database by its ID. It uses the entity manager to find the Book entity with the specified ID and returns it. If no book is found, it returns null.
    3. getByIsbn(String isbn): This method retrieves a book from the database by its ISBN number (either ISBN-10 or ISBN-13). It constructs a JPQL query to select the book with the specified ISBN number and executes it. If a book matching the ISBN is found, it returns it; otherwise, it returns null.

UserBook

- This class represents the association between a user and a book within the application's database.
- It stores metadata about the relationship, such as the creation date, deletion date, and read date.
- The main fields are id, bookId, userId, createDate, deleteDate, readDate.
- The hashCode() and equals() methods are overridden to provide custom equality comparison based on the bookId and userId fields. This ensures proper comparison and identification of UserBook instances.

BaseResource (abstract class)

- Implements authentication and other checks for the user so that he is not anonymous
- Can be extended by resource classes, supplying common functionalities and applying security measures
- Implements checks to verify if the user has the authority to perform tasks

UserBookTag

- This class represents a mapping between a user book and a tag within a database. It allows for the association of multiple tags with a user book and provides methods for accessing and manipulating these associations within a JPA-based application.
- The main fields are id, userBookId and tagId.
- The hashCode() and equals() methods are overridden to provide custom equality comparison based on the userBookId and tagId fields. This ensures proper comparison and identification of UserBookTag instances.

---

2. Bookshelf Management Subsystem:

BaseResource (abstract class)

- Implements authentication and other checks for the user so that he is not anonymous

- Can be extended by resource classes, supplying common functionalities and applying security measures
- Implements checks to verify if the user has the authority to perform tasks

Tag

- Represents tags in the system, storing information such as their name, associated user ID, creation date, deletion date (if applicable), and color.
- Ensures proper mapping of the class to the database schema, facilitating persistence operations.
- Attributes are encapsulated with private access modifiers and accessed through public getter and setter methods, adhering to the principle of encapsulation.
- The class does not include any validation logic for attributes like `id`, `name`, and `color`. Adding validation logic to ensure that these attributes meet certain criteria (e.g., length constraints) can enhance data integrity.

TagResource

- Provides endpoints for listing, adding, updating, and deleting tags.
- Implements standard RESTful endpoints for CRUD operations on tags, providing a clear and predictable interface for clients.
- Properly handles authentication to ensure only authenticated users can perform operations on tags.
- Utilizes data validation to ensure the integrity of tag attributes.
- Direct interaction with `TagDao` within the resource methods tightly couples the resource layer with the data access layer, potentially leading to issues like code duplication and difficulty in testing.
- **Code Duplication**: Some validation and error-checking logic is duplicated across methods. Refactoring common functionality into helper methods or aspects could improve maintainability and readability.

TagDao

- Responsible for interacting with the database to perform CRUD (Create, Read, Update, Delete) operations on tags.
- Utilizes parameterized queries to prevent SQL injection.
- Supports transactional operations for data consistency.
- Lack of input validation in some methods, potentially leading to unexpected behavior.
- Direct interaction with the database layer might result in tight coupling with the persistence framework.

TagDto

- Serves as a Data Transfer Object (DTO) for transferring tag information between different layers of the application.
- Encapsulates tag attributes within a single object, improving maintainability and readability.
- Lack of validation or data transformation logic, which might lead to inconsistencies if not handled properly when converting between DTOs and domain objects.

UserBookDao

- Provides data access methods for managing user books

- **Search**: A method for searching user books by criteria such as search text, tag IDs, and read status.
- **Pagination**: Supports paginated retrieval of user books to efficiently handle large datasets.
- Uses parameterized queries to prevent SQL injection attacks and ensure data integrity.Breaks down complex queries into manageable parts, enhancing readability and maintainability.
- **Lack of Abstraction**: Directly couples database-specific details with application logic, potentially reducing portability and making it harder to switch database providers.
- **Limited Reusability**: Methods are tightly coupled with the `UserBookDto` class, limiting their reusability in other parts of the application.

UserBookDto

- Serves as a data transfer object (DTO) representing user book information. It contains properties corresponding to various attributes of a user book, such as title, subtitle, author, language, publication date, creation date, and read date.
- Provides a straightforward representation of user book data with simple getter and setter methods for each attribute.
- Does not include validation logic for ensuring the correctness of data during setting of properties, potentially leading to inconsistencies or errors in the application.
- Uses `Long` data type for timestamps instead of `Date` or `LocalDateTime`, which may require additional conversion and handling in certain scenarios.

UserBookCriteria

- Represents criteria used for searching user books. It encapsulates various filtering options such as user ID, search query, read state, and tag IDs.
- Offers a simple interface with getter and setter methods for each criteria property, making it easy to understand and use.
- The absence of default values for criteria properties may require additional handling to prevent null pointer exceptions or unexpected behavior when certain criteria are not set.

BookResource

- This class acts as an intermediary between the Android application and a backend server that provides data about books. It encapsulates the logic for making HTTP requests to the server's API endpoints and handling the responses.
- The list() method retrieves a list of books from the server, specifying parameters for sorting and pagination.
- The info() method fetches detailed information about a specific book identified by its unique ID.
- The add() method allows the application to add a new book to the system by providing its ISBN.

UserBookTag

- This class represents a mapping between a user book and a tag within a database. It allows for the association of multiple tags with a user book and provides methods for accessing and manipulating these associations within a JPA-based application.
- The main fields are id, userBookId and tagId.
- The hashCode() and equals() methods are overridden to provide custom equality comparison based on the userBookId and tagId fields. This ensures proper comparison and identification of

UserBookTag instances.

BookImportedEvent

- This class serves as a container for data related to a book import event, allowing for the transfer of information about the import request within the application's event-driven architecture.
- The main fields are the user and the importFile which contains the csv file to be imported.
- This event class encapsulates information about a book import request, including the user initiating the request and the file containing the books to be imported.
- It facilitates passing relevant data about the import request between different components of the application.

UserBook

- This class represents the association between a user and a book within the application's database.
- It stores metadata about the relationship, such as the creation date, deletion date, and read date.
- The main fields are id, bookId, userId, createDate, deleteDate, readDate.
- The hashCode() and equals() methods are overridden to provide custom equality comparison based on the bookId and userId fields. This ensures proper comparison and identification of UserBook instances.

BookImportAsyncListener

- This Java class is a listener component responsible for handling events related to book import requests asynchronously.
- `@Subscribe` indicates that the method on(BookImportedEvent bookImportedEvent) subscribes to events of type BookImportedEvent published by the event bus.
- `on(BookImportedEvent bookImportedEvent)`: This method is invoked when a BookImportedEvent is published. It processes the event by importing books from a CSV file provided in the event.
- This adds the books to the database if they don't already exist and then adds the tags to the user corresponding to each book.

Book

- Represents the entity model for books in the application's database.
- It defines attributes such as `id`, `title`, `subtitle`, `author`, `description`, `isbn10`, `isbn13`, `pageCount`, `language`, and `publishDate`, representing various properties of a book.

PaginatedList

- Simple representation of a paginated list, commonly used in applications dealing with large datasets that need to be retrieved in smaller, manageable portions.
- Defines a generic class `PaginatedList<T>`, which is used to handle paginated lists of objects of type `T`.
- Used for retrieving large data in chunk or pages. Four instance variables: limit, offset, resultCount, resultList.
- Considering that pagination parameters are typically set once upon instantiation and remain constant, making them immutable (e.g., through constructor parameters without corresponding

setters) could enhance the class's reliability and maintainability.

PaginatedLists

- Creates and handles PaginatedList object
- Provides utilities for working with paginated lists in the context of Java Persistence API (JPA).
- It allows the creation of paginated lists with specified page size and offset, ensuring efficient retrieval of data from large result sets.

---

3. User Management Subsystem:

BaseFunction

- Defines the fundamental functionalities that can be performed in the associated base function
- Defines a role (eg: admin) and the corresponding base function
- Controls access based on the role of a user

BaseResource (abstract class)

- Implements authentication and other checks
- Can be extended by resource classes, supplying common functionalities and applying security measures

RoleBaseFunctionDao

- Data Access Object for base functions that have a role associated with them

Constants

- Has constants that are used all over the application, increasing efficiency and eliminating the need to hardcode them
- Stores the default locale, timezone, theme, admin password, and user role

AuthenticationTokenDao

- Data Access Object for Authentication Tokens
- Has methods to get, retrieve, create, and delete tokens

TokenBasedSecurityFilter

- Manages user authentication via authentication tokens in cookies
- Checks token validiy and expiration status to determine user authentication status.
- Injects either authenticated or anonymous user principal into request attributes
- Utilizes DAO classes (AuthenticationTokenDao, UserDao, RoleBaseFunctionDao) for database interactions

ThreadLocalContext

- Manages context associated with a user request and stores it in a ThreadLocal.

- Utilizes a ThreadLocal variable named threadLocalContext for storage.
- Includes an EntityManager field for managing database transactions.
- Follows a singleton pattern to ensure one instance per thread.
- Provides methods to initialize and clean up the thread context (get() and cleanup()).
- Offers a isInTransactionalContext() method to detect if the entity manager is defined, indicating a transactional scope.

ValidationUtil

- A utility class for validating parameters in a RESTful context.
- Includes methods for checking if parameters are required and validating their lengths.
- Provides validation for strings, including not blank, hex color, email, HTTP URL, and alphanumeric patterns.
- Validates and parses dates.
- Validates locale and theme IDs against database entries.
- Throws `ClientException` with appropriate error messages for validation failures.
- Utilizes DAO classes (`ThemeDao`, `LocaleDao`) to validate theme and locale IDs against database entries.

SortCriteria

- Represents sort criteria for a query.
- Includes an index of the column to sort (starting from 0) and a flag indicating ascending or descending order.
- Provides a constructor to initialize the sort criteria with optional parameters for column index and sort order.
- Contains getters to retrieve the column index and sort order.

PaginatedList

- Represents a paginated list of records.
- Includes properties for page size, offset, total result count, and a list of records for the current page.
- Provides constructors to initialize page size and offset.
- Contains getters and setters for accessing and modifying the properties of the paginated list.

PaginatedLists

- Provides utilities for handling paginated lists in JPA contexts.
- Can create paginated lists with configurable page size and offset.
- Supports executing count queries to determine the total number of results.
- Executes queries to retrieve data for the current page based on pagination parameters.
- Offers methods for executing paginated queries with sorting criteria.
- Utilizes `QueryUtil` and `QueryParam` classes for query execution and parameter management.
- Implements default page size and maximum page size constants for consistency and limiting purposes.

User

- Represents a user entity in the JPA context.

- Contains fields representing user attributes.
- Has storage of user-specific settings, such as language and theme.

UserDao

- Serves as a Data Access Object for managing user-related data in the application's database. It provides methods to interact with the `User` entity, including authentication, creation, retrieval, updating, and deletion of user records.

- **Methods**:

  1. `authenticate(String username, String password)`: Authenticates a user by verifying the provided username and password against the database records. It uses BCrypt for secure password hashing.

  2. `create(User user)`: Creates a new user record in the database. It generates a UUID for the user ID, checks for username uniqueness, hashes the password, sets default values for certain attributes, and persists the user object.

  3. `update(User user)`: Updates an existing user record in the database. It retrieves the user from the database, updates the relevant attributes, and returns the updated user object.

  4. `updatePassword(User user)`: Updates the password of an existing user in the database. It retrieves the user from the database, updates the password with a hashed version, and returns the updated user object.

  5. `getById(String id)`: Retrieves a user record from the database by its unique identifier (`id`).

  6. `getActiveByUsername(String username)`: Retrieves an active user record from the database by its username.

  7. `getActiveByPasswordResetKey(String passwordResetKey)`: Retrieves an active user record from the database by its password reset token.

  8. `delete(String username)`: Deletes a user record from the database by its username. It marks the user as deleted and deletes any associated data, such as authentication tokens.

  9. `hashPassword(String password)`: Hashes the provided password using the BCrypt hashing algorithm for secure storage in the database.

  10. `findAll(PaginatedList<UserDto> paginatedList, SortCriteria sortCriteria)`: Retrieves a paginated list of all user records from the database. It constructs and executes a query to fetch user data, assembles the results into DTOs (Data Transfer Objects), and populates the provided paginated list.

- **Dependencies**:

  - `EntityManager`: Manages the lifecycle of user entities within the persistence context.
  - `ThreadLocalContext`: Provides access to the `EntityManager` associated with the current thread.
  - `BCrypt`: Library for secure password hashing.

    - `Constants`: Contains application-wide constant values.
    - `PaginatedList`, `PaginatedLists`, `QueryParam`, `SortCriteria`: Utility classes for handling pagination, query parameters, and sorting in JPA queries.

Overall, the `UserDao` class encapsulates the logic for interacting with user data in the database, providing essential functionalities for user management within the application.

UserDto

- Serves as a lightweight representation of user data, facilitating data transfer between different layers of the application. Here's a brief overview:
- Enables efficient transfer of user data between application layers.
- Encapsulates essential user information for specific operations.
- Typically employed for transferring user details to presentation layer components. Enhances maintainability and performance by controlling data granularity.

UserResource

- Extends BaseResource and gives resource implementation for managing user-related operations such as registration, login, logout, profile updates, and user deletion.
- It includes functionality for managing user sessions, including login, logout, and session deletion.
- Input data is validated before processing, ensuring data integrity and preventing common security vulnerabilities like injection attacks.
- Some code segments, particularly validation logic, appear to be duplicated across different methods. Refactoring common logic into utility methods or shared components can improve maintainability.
- The class may benefit from further modularization, breaking down complex methods into smaller, more focused functions to improve readability and maintainability.

Iprincipal

- The `IPrincipal` interface defines the contract for principal objects in the application. It extends the `java.security.Principal` interface and provides additional methods to retrieve user-related information such as ID, locale, timezone, and email.
- Tight coupling with `DateTimeZone` from Joda-Time library, which may limit flexibility in using other date/time libraries.

AnonymousPrincipal

- The `AnonymousPrincipal` class represents an anonymous user within the application.
- It implements the `IPrincipal` interface and provides methods to retrieve information about the anonymous user such as locale and timezone.
- This class is typically used to represent unauthenticated users or users with limited access.
- Lack of flexibility in representing different types of anonymous users with varying characteristics.

UserPrincipal

- The `UserPrincipal` class represents an authenticated user within the application.
- It implements the `IPrincipal` interface and provides methods to retrieve information about the authenticated user such as ID, username, locale, timezone, email, and base functions.

- This class is typically used to represent users who have successfully authenticated and have access to various functionalities within the application.
- Provides a comprehensive representation of authenticated users within the application, including essential user information and base functions.

> The UML Diagrams are attached in the same order as the above subsystems.