

# **CS6.401 Software Engineering Spring 2024**

## **Project- 1 Report**

### **Team 23**

#### **Team Members:**

Name	Roll Number
Karan Bhatt	2022202003
Madhusree Bera	2022202007
Vedashree Ranade	2022201073
Yash Maheshwari	2022201074
Piyush Rana	2022202012

## **Relevant Classes Identification and**

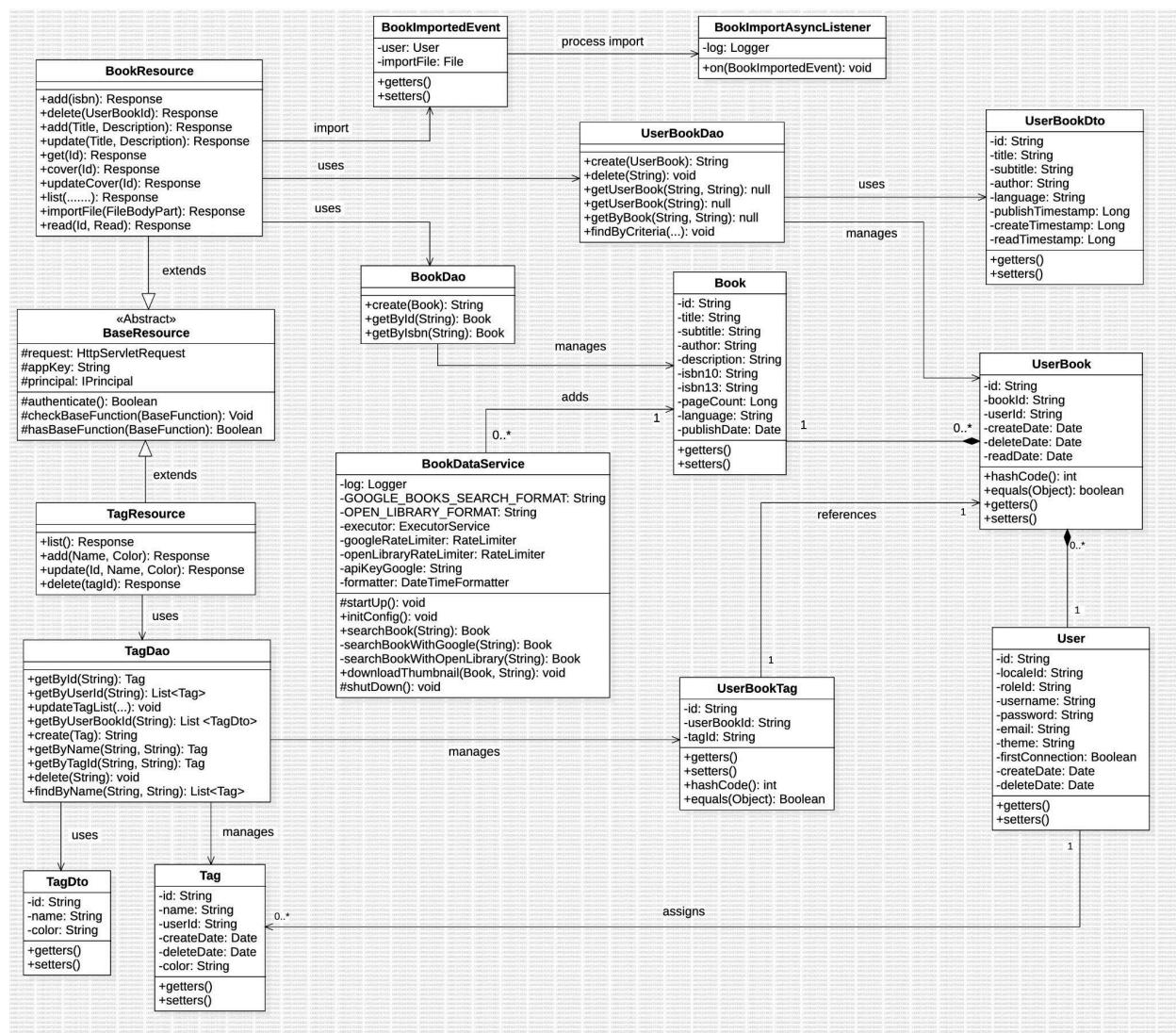
## **UML Class Diagrams**

# 1. Book Addition & Display Subsystem:

This is where new characters (books) enter our literary world. Picture a multi-modal approach, barcode scanners on Android and iOS – like magical portals, and the use of ISBNs. Google and OpenLibrary databases are our trusted encyclopedias, and we can even import data from Goodreads for a richer narrative.

This system also displays the books' cover art, detailed descriptions, author biographies – it's like opening a treasure chest of literary information. Users can even interact with the books, marking them as read or completed.

## UML Class Diagram



## List of Relevant Classes

### 1.1 Book class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/Book.java
- **Functionalities:**
  - a. This class stores the book model
  - b. It has all the parameters of a book which are set private
  - c. It also has getters and setters to access these parameters
- **Relevance:**

Addition of a new book requires the details of the book to be stored. This can be done using an object of the Book class.

### 1.2 BookDao class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/BookDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the Book objects
  - b. Its main functionality includes
    - i. Creating a new book
    - ii. Searching for a book by Id and returning it
    - iii. Searching for a book by ISBN and returning it

### 1.3 BaseResource abstract class

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/BaseResource.java
- **Functionalities:**
  - a. This class is the base class for REST resources.
  - b. Its main functionalities include checking the authenticity of the user and checking if the user has a base function.
  - c. This is an abstract class and is inherited by many other resource classes.

## 1.4 BookResource class

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/BookResource.java
- **Functionalities:**
  - a. This class extends the base class BaseResource
  - b. Used for CRUD operations such as
    - i. creating a new book
    - ii. deleting existing book
    - iii. adding a book manually
    - iv. update any parameter of an existing book
    - v. Fetch an existing book
    - vi. Fetch a book cover
    - vii. Update book cover
    - viii. Return a list of all books
    - ix. Import a book file
    - x. Set book as read or unread
  - c. It makes use of other objects of type BookDao, Book, UserBookDao, TagDao, BookImportedEvent, UserDao
- **Relevance:**

The class provides a method to create a book, add a book manually, update the book's parameter, display and update book cover, import book from file, mark book as read / unread.

## 1.5 BookImportedEvent class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/event/BookImportedEvent.java
- **Functionalities:**
  - a. This class is used to raise an event for a request to import books
  - b. It encapsulates together the user and the imported book file
- **Relevance:**

The class allows importing books which can be used for addition of books.

## 1.6 BookImportAsyncListener class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/listener/async/BookImportAsyncListener.java
- **Functionalities:**
  - a. This class is listener for book import requests

- b. It uses BookImportedEvent, CSVReader, BookDao, UserBookDao and TagDao objects
- c. This is used to import data from GoodReads by reading csv files
- **Relevance**

The class allows to import GoodReads file using CSVReader

## 1.7 BookDataService class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/service/BookDataService.java
- **Functionalities:**
  - a. This class inherits from the base class AbstractIdleService
  - b. Its main functionality is to run services for fetching book information such as
    - i. Searching for books by ISBN
    - ii. Searching for a book by ISBN using Google Books
    - iii. Searching for a book by ISBN using Open Library
    - iv. Downloading and overwriting the thumbnail for a book
  - c. It uses the Book class to store the fetched books
- **Relevance:**  
The class is required to fetch the information of the books to be displayed from Google Books, Open Library.

## 1.8 User class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/User.java
- **Functionalities:**
  - a. This class stores the User model which is used to create users
  - b. It has all the parameters of the user such as id, localeId, roleId, username, password, email, theme, firstConnection, createDate and deleteDate. All these are set as private attributes
  - c. It also has getters and setters to access these parameters
- **Relevance:**  
The User class is a relevant class for book addition because a new book can only be added by a user when logged in. Also, it is relevant for books display because the books can be displayed only for a specific user.

## 1.9 UserBook class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/Tag.java

- **Functionalities:**
  - a. This class stores information of both the user and the book being read by him.
  - b. It has all the parameters related to a user's book such as id, userId, bookId, createDate, deleteDate, readDate. All these are set as private attributes
  - c. It also has getters and setters to access these parameters

## 1.10 UserBookDao class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/UserBookDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the UserBook objects
  - b. Its main functionalities include
    - i. Creating a new UserBook
    - ii. Deleting a UserBook
    - iii. Get a tag by its UserBook ID, UserBook ID and user ID, book ID and user ID
    - iv. Searching books by criteria

## 1.11 UserBookDto class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/dto/UserBookDto.java
- **Functionalities:**
  - a. This class is DTO class for transferring parameters of UserBook and Book
  - b. It contains getters and setters for its attributes which are private

## 1.12 Tag class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/Tag.java
- **Functionalities:**
  - a. This class stores the tag model which is used to create virtual bookshelves
  - b. It has all the parameters of tag such as name, color, create date, delete date and the user id of the creator. All these are set private attributes
  - c. It also has getters and setters to access these parameters
- **Relevance:**

The class helps to store the details about the tag related to the book to be added or the book to be displayed.

### 1.13 TagDao class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/TagDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the Tag objects
  - b. It's main functionality includes
    - i. Creating a new tag
    - ii. Get a tag by its tag ID, tag name, tag Id and user Id
    - iii. Return a list of all tags
    - iv. Update tags on a user book in the database
    - v. Returns tag list on a particular user book.
    - vi. Deleting a tag
    - vii. Searching tags by name

### 1.14 TagDto class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/dto/TagDto.java
- **Functionalities:**
  - a. This class is DTO class for transferring id, name and color of tag
  - b. It contains getters and setters for its attributes

### 1.15 TagResource class

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/TagResource.java
- **Functionalities:**
  - a. This class extends the base class BaseResource
  - b. Used for CRUD operations such as
    - i. creating a new tag
    - ii. deleting existing tag
    - iii. update any parameter of an existing tag
    - iv. Fetch list of all tags
  - c. It makes use of other objects of type TagDao and Tag
- **Relevance:**

The class is relevant because while adding a new book manually, the tag for the book also needs to be provided.

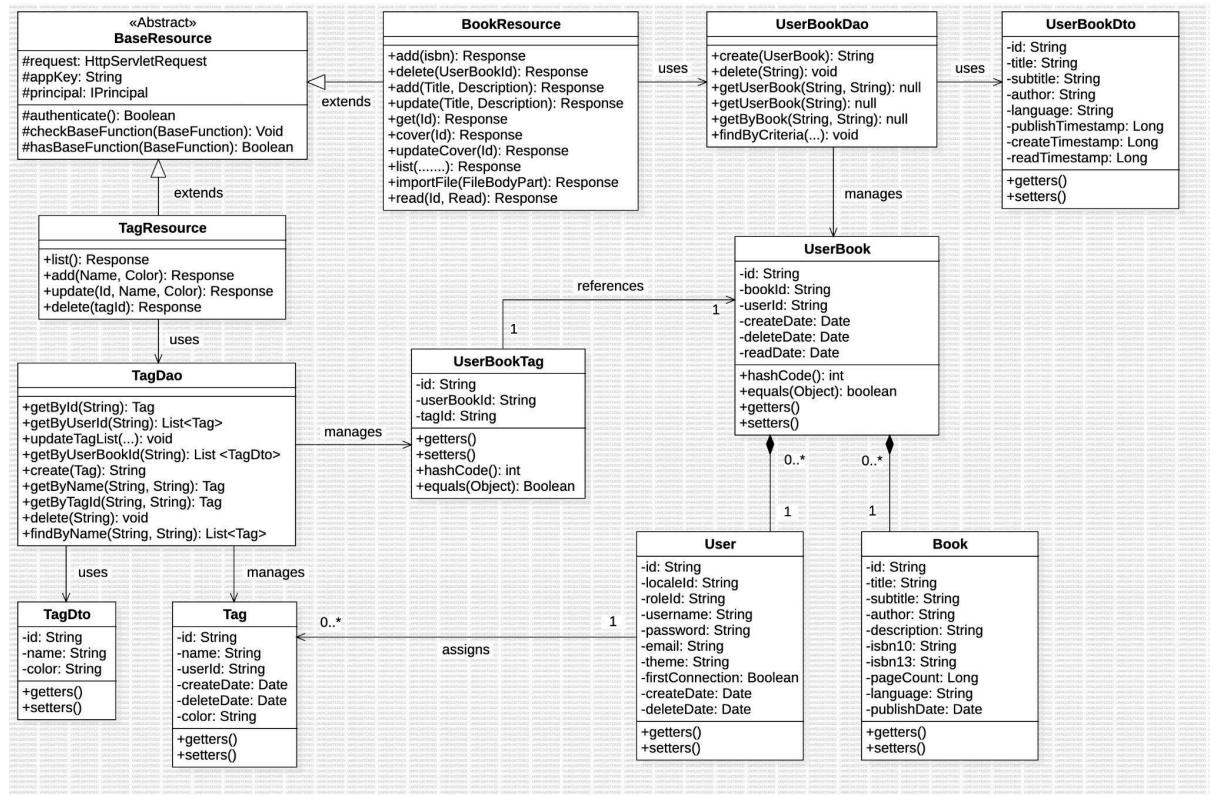
## 1.16 UserBookTag class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/UserBookTag.java
- **Functionalities:**
  - a. This class encapsulates the UserBook and Tag. It has attributes like id, userBookID, tagID
  - b. It also has getters and setters to access these parameters

## 2. Bookshelf Management Subsystem:

In the vast library of our app, users can create their own bookshelves, like personal book genres. Imagine organizing books into these virtual bookshelves, making it easy to navigate through the literary landscape.

### UML Class Diagram



### List of Relevant Classes

#### 2.1 Tag class

- File path:** books-core/src/main/java/com/sismics/books/core/model/jpa/Tag.java
- Functionalities:**
  - This class stores the tag model which is used to create virtual bookshelves
  - It has all the parameters of tag such as name, color, create date, delete date and the user id of the creator. All these are set private attributes
  - It also has getters and setters to access these parameters

- **Relevance:**

The class helps to store the details about the tag related to the book.

## 2.2 TagDao class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/TagDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the Tag objects
  - b. It's main functionality includes
    - i. Creating a new tag
    - ii. Get a tag by its tag ID, tag name, tag Id and user Id
    - iii. Return a list of all tags
    - iv. Update tags on a user book in the database
    - v. Returns tag list on a particular user book.
    - vi. Deleting a tag
    - vii. Searching tags by name

## 2.3 TagDto class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/dto/TagDto.java
- **Functionalities:**
  - a. This class is DTO class for transferring id, name and color of tag
  - b. It contains getters and setters for its attributes

## 2.4 BaseResource abstract class

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/BaseResource.java
- **Functionalities:**
  - a. This class is the base class for REST resources.
  - b. Its main functionalities include checking the authenticity of the user and checking if the user has a base function.
  - c. This is an abstract class and is inherited by many other resource classes.

## 2.5 TagResource class

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/TagResource.java
- **Functionalities:**
  - a. This class extends the base class BaseResource
  - b. Used for CRUD operations such as
    - i. creating a new tag
    - ii. deleting existing tag
    - iii. update any parameter of an existing tag
    - iv. Fetch list of all tags
  - c. It makes use of other objects of type TagDao and Tag
- **Relevance:**

The class is relevant because while adding or displaying a book, the tag for the book also needs to be provided.

## 2.6 BookResource class

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/BookResource.java
- **Functionalities:**
  - a. This class extends the base class BaseResource
  - b. Used for CRUD operations such as
    - i. creating a new book
    - ii. deleting existing book
    - iii. adding a book manually
    - iv. update any parameter of an existing book
    - v. Fetch an existing book
    - vi. Fetch a book cover
    - vii. Update book cover
    - viii. Return a list of all books
    - ix. Import a book file
    - x. Set book as read or unread
  - c. It makes use of other objects of type BookDao, Book, UserBookDao, TagDao, BookImportedEvent, UserDao

## 2.7 Book class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/Book.java
- **Functionalities:**
  - a. This class stores the book model
  - b. It has all the parameters of a book which are set private
  - c. It also has getters and setters to access these parameters

## 2.8 User class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/User.java
- **Functionalities:**
  - a. This class stores the User model which is used to create users
  - b. It has all the parameters of the user such as id, localeId, roleId, username, password, email, theme, firstConnection, createDate and deleteDate. All these are set as private attributes
  - c. It also has getters and setters to access these parameters
- **Relevance:**

The User class is a relevant class for bookshelf management because a bookshelf is specific to a particular user. The user has the capability to create and manage bookshelves.

## 2.9 UserBook class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/Tag.java
- **Functionalities:**
  - a. This class stores information of both the user and the book being read by him.
  - b. It has all the parameters related to a user's book such as id, userId, bookId, createDate, deleteDate, readDate. All these are set as private attributes
  - c. It also has getters and setters to access these parameters

## 2.10 UserBookDao class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/UserBookDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the UserBook objects

- b. It's main functionalities include
  - i. Creating a new UserBook
  - ii. Deleting a UserBook
  - iii. Get a tag by its UserBook ID, UserBook ID and user ID, book ID and user ID
  - iv. Searching books by criteria

## 2.11 UserBookDto class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/dto/UserBookDto.java
- **Functionalities:**
  - a. This class is DTO class for transferring parameters of UserBook and Book
  - b. It contains getters and setters for its attributes which are private

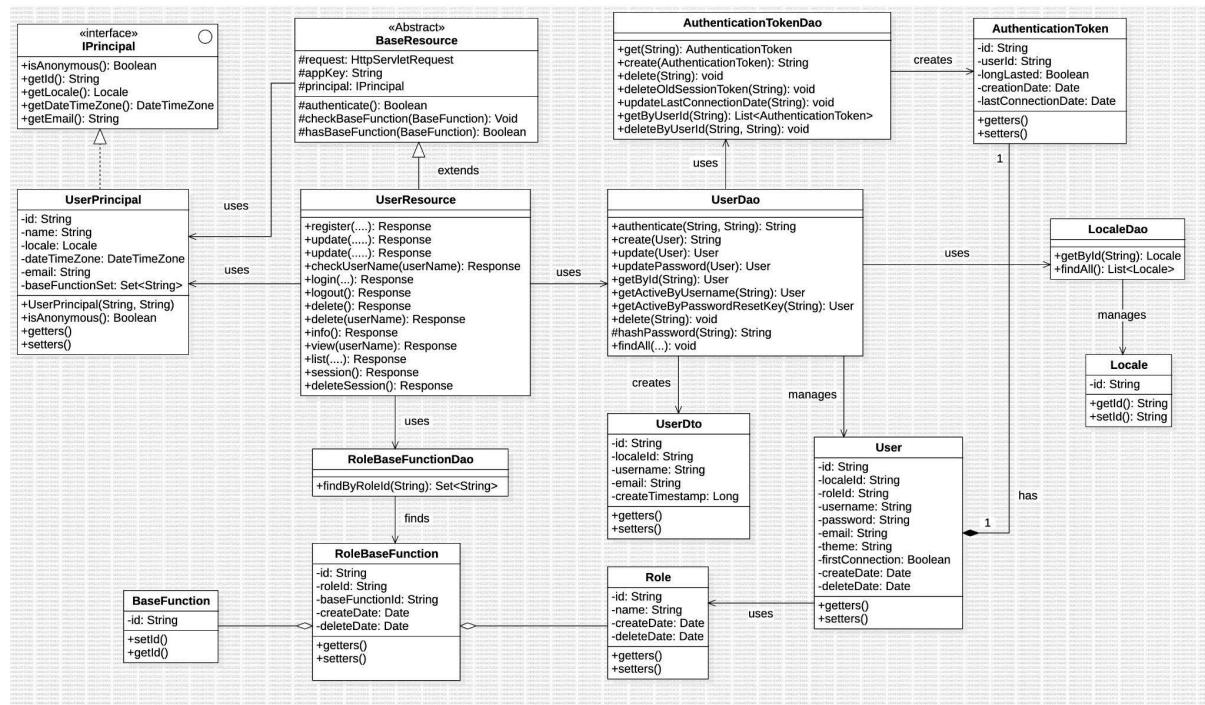
## 2.12 UserBookTag class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/UserBookTag.java
- **Functionalities:**
  - a. This class encapsulates the UserBook and Tag. It has attributes like id, userBookID, tagID
  - b. It also has getters and setters to access these parameters

### 3. User Management Subsystem:

Every literary kingdom needs its guardians. This subsystem empowers admins to add users, change passwords, and maintain order by clearing other opened sessions.

### UML Class Diagram



### List of Relevant Classes

#### 3.1 IPrincipal interface

- File path:** books-web-common/src/main/java/com/sismics/security/IPrincipal.java
- Functionalities:**
  - This is the interface of Principals
  - Its main functionality are shown below
    - Check if the principal is anonymous.
    - Returns the ID of the connected user, or null if the user is anonymous
    - Returns the locale of the principal.
    - Returns the timezone of the principal.
    - Returns the email of the principal.

## 3.2 UserPrincipal class

- **File path:**  
books-web-common/src/main/java/com/sismics/security/UserPrincipal.java
- **Functionalities:**
  - a. This is a class for authenticated UserPrincipals.
  - b. It implements IPrincipal interface
  - c. It has private attributes such as id, name, locale, dateTimeZone, email, baseFunctionSet
  - d. Through the baseFunctionSet, the admin functionalities are specified.
  - e. It has getters and setters for the private attributes
  - f. The isAnonymous function of UserPrinciple will always return false

## 3.3 User class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/User.java
- **Functionalities:**
  - a. This class stores the User model which is used to create users
  - b. It has all the parameters of user such as id, localeId, roleId, username, password, email, theme, firstConnection, createDate and deleteDate. All these are set as private attributes
  - c. It also has getters and setters to access these parameters

## 3.4 UserDao class

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/UserDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the User objects
  - b. Its main functionality includes
    - i. Authenticating an user
    - ii. Creating a new user
    - iii. Update user object
    - iv. Update password
    - v. Get a user by its user ID/ username /password recovery token
    - vi. Return a list of all users
    - vii. Hash the password
    - viii. Deleting a user

### **3.5 UserDto class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/dto/UserDto.java
- **Functionalities:**
  - a. This class is DTO class for transferring parameters of User
  - b. It contains getters and setters for its attributes which are private

### **3.6 AuthenticationToken class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/AuthenticationToken.java
- **Functionalities:**
  - a. This class stores the AuthenticationToken model which is used for maintaining sessions
  - b. It has all the parameters such as id, userId, longLasted, creationDate and lastConnectionDate. All these are set as private attributes
  - c. It also has getters and setters to access these parameters

### **3.7 AuthenticationTokenDao class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/AuthenticationTokenDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the AuthenticationToken objects
  - b. Its main functionality includes
    - i. Fetching an auth token
    - ii. Creating a new auth token
    - iii. Deleting auth token
    - iv. Deleting old short lived tokens
    - v. Update last connection date
    - vi. Fetch all authentication tokens of an user.
    - vii. Deletes all authentication tokens of an user.

### **3.8 Locale class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/Locale.java
- **Functionalities:**
  - a. This class is used for storing the model of Locale
  - b. It has a ID which is private
  - c. It has getters and setters for id and toString() method

### **3.9 LocaleDao class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/LocaleDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the Locale objects
  - b. It's main functionality includes
    - i. Get a locale by its id from database
    - ii. Find all locales from the database

### **3.10 Role class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/Role.java
- **Functionalities:**
  - a. This class is used for storing the model of Role
  - b. It has private attributes of role.
  - c. It has getters, setters and toString() methods
  - d. This is used to store the Admin role.
  - e. Any other future roles can be created using this role

### **3.11 BaseFunction class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/BaseFunction.java
- **Functionalities:**
  - a. This class is used for storing the model of BaseFunction
  - b. It has a private attribute of id.
  - c. It has getters, setters and toString() methods
  - d. This is used to store the base function of the Admin role.

### **3.12 RoleBaseFunction class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/model/jpa/RoleBaseFunction.java
- **Functionalities:**
  - a. This class is used for storing the model of RoleBaseFunction
  - b. It aggregates roleId and basefunctionId and has other private attributes
  - c. It has getters, setters and toString() methods
  - d. This is used to store the Admin role and its base function.
  - e. Any other future roles can be related to their base function using this

### **3.13 RoleBaseFunctionDao class**

- **File path:**  
books-core/src/main/java/com/sismics/books/core/dao/jpa/RoleBaseFunctionDao.java
- **Functionalities:**
  - a. This class is DAO class for accessing the RoleBaseFunction objects
  - b. Its main functionality includes
    - i. Finding set of base functions of a role from the database

### **3.14 BaseResource abstract class**

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/BaseResource.java
- **Functionalities:**
  - a. This class is the base class for REST resources.
  - b. Its main functionalities include checking the authenticity of the user and checking if the user has a base function.
  - c. This is an abstract class and is inherited by many other

### **3.15 UserResource class**

- **File path:**  
books-web/src/main/java/com/sismics/books/rest/resource/UserResource.java
- **Functionalities:**
  - a. This class extends the base class BaseResource
  - b. This class allows the admin to manage the users
  - c. Its main functionalities include
    - i. Allows admin to register new user

- ii. Allows user to update their data
- iii. Checks if username is active
- iv. Login and logout functionalities that creates and ends a session
- v. Allows admin to delete a user
- vi. Fetch information about connected user
- vii. Allows admin to view all active users
- viii. Allows admin to view user details
- ix. Allows admin to see all active sessions
- x. Allows admin to delete any active session
- d. It makes use of other objects of type AuthenticationTokenDao and AuthenticationToken for session handling, User, UserDao for handling users

## **Observations and Comments:**

### **Strength:**

- The code is well modularized by creating classes for various independent entities.
- Authentication is checked by assigning role for Admin and Users
- Enum and constants are used according to standard practices

### **Weakness:**

- There are some unnecessary classes created for defining only a single function which is not really required.
- Some child classes are inherited which are actually not using any function of the parent class.
- Repetition of code exists in case of some functions having common code.
- For checking Admin role, many tables are created such as Base function table, Role table, Role base function table which can be done in a more optimised way.
- Too many functions are present in UserResource for every role (User and Admin)
- In every function for the admin role, “if” condition is used to check if the user is admin or not.

# **Design Smells Detection and Refactoring**

## 1.Imperative Abstraction in EntityManagerUtil

**Path:**

books-core/src/main/java/com/sismics/books/core/util/EntityManagerUtil.java

**Description:**

Imperative Abstraction is indicated when an operation is turned into a class that has only one method defined in it. In the given class "EntityManagerUtil", the static method flush() is calling the javax.persistence.EntityManager.flush() method. For just one line of operation, a separate class is not required.

This static method of EntityManagerUtil class is not even used in any part of code.

**Tool used:**

SonarQube Code Smells report. It indicated “Add a private constructor to hide the implicit public one.” as code smell.

**Our Solution:**

Included the “flush()” method from EntityManagerUtil to ThreadLocalContext



The screenshot shows a code editor with a dark theme. A file named 'ThreadLocalContext.java' is open. The code is annotated with SonarQube's 'diff' tool, showing changes made to the file. The changes are as follows:

```
@@ -72,4 +72,11 @@ public EntityManager getEntityManager() {  
 72 72     public void setEntityManager(EntityManager entityManager) {  
 73 73         this.entityManager = entityManager;  
 74 74     }  
 75 +    /**  
 76 +     * Flush the entity manager session.  
 77 +     */  
 78 +    public static void flush() {  
 79 +        get().getEntityManager().flush();  
 80 +    }  
 81 +}  
 75 82 }
```

The new code adds a static flush() method that calls the EntityManager's flush() method via reflection ('get().getEntityManager().flush();').

## ChatGPT Solution:



ChatGPT

The provided code snippet appears to be a simple utility class for flushing an entity manager session. However, one potential design smell is the tight coupling and direct reliance on the `ThreadLocalContext` to obtain the entity manager. This can introduce potential issues and make the code less modular and testable.

Design smells are subjective, and what might be considered a design smell in one context may be acceptable in another. However, in this case, the direct dependency on `ThreadLocalContext` might be a concern. It tightly couples the utility class to a specific context handling mechanism, making it less flexible and harder to test in isolation.

A more flexible and testable design would involve injecting the entity manager or the entity manager factory as a parameter, allowing the class to work with different contexts or be easily testable with mock implementations. This would decouple the utility class from the specific context management strategy and make it more adaptable to changes or different environments.

KB

You

Isn't there imperative design smell in EntityManagerUtil class as it has only one method and that too not used anywhere in the project



ChatGPT

Yes, you are correct. The `EntityManagerUtil` class, as presented, exhibits a form of the Imperative Abstraction smell. This smell occurs when a class or method is defined but lacks a clear purpose or utility in the context of the project. In this case, if the `flush` method is not used anywhere in the project, it suggests that the class may not be serving a meaningful purpose.

To address this smell, you have a few options:

### 1. Remove Unused Code:

If the `flush` method is not used anywhere in the project, consider removing the entire `EntityManagerUtil` class. Unused or unnecessary code can lead to confusion and maintenance overhead.

java

Copy code

```
// Remove the entire EntityManagerUtil class.
```

**2. Review and Refactor:**

If there's a reason for keeping the `EntityManagerUtil` class but the `flush` method is not needed, consider reviewing the class's purpose and refactor it accordingly. If there are other methods or responsibilities that could be added to make it useful, that might be an option.

**3. Documentation:**

If there's a reason for keeping the class even though it seems unused (e.g., future use, specific design decisions), consider adding comments or documentation explaining the rationale.

It's generally a good practice to keep the codebase free from unused or unnecessary elements to enhance readability and maintainability.

## 2. Broken Modularization in MimeTypeUtil & MimeType

### Path:

books-core/src/main/java/com/sismics/books/core/util/mime

### Description:

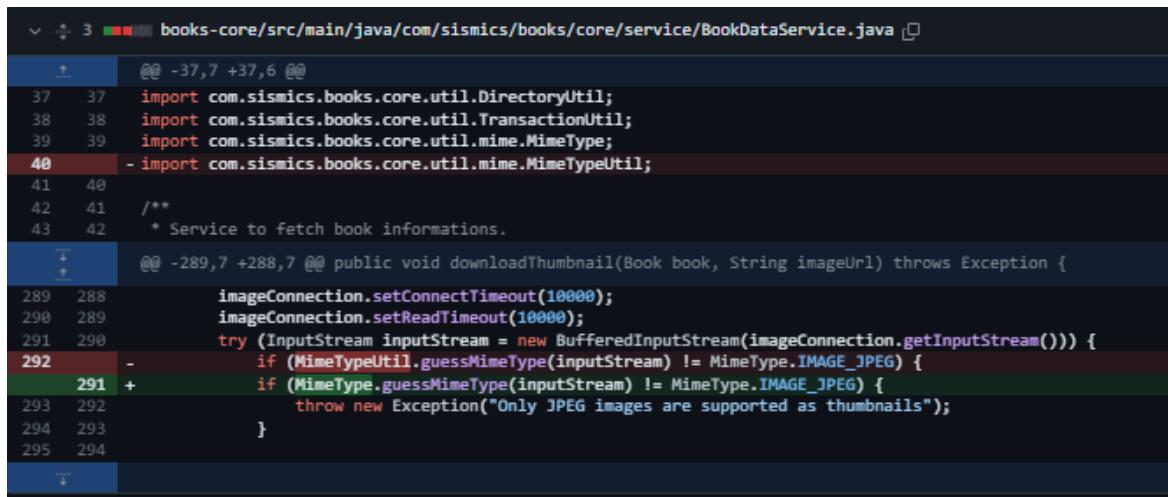
This smell arises when data and/or methods that ideally should have been localized into a single abstraction are separated and spread across multiple abstractions.

### Tool Used:

DesigniteJava report

### Our Solution:

MimeTypeUtil methods are included in MimeType class instead of a separate util class.



```
@@ -37,7 +37,6 @@
37 37 import com.sismics.books.core.util.DirectoryUtil;
38 38 import com.sismics.books.core.util.TransactionUtil;
39 39 import com.sismics.books.core.util.mime.MimeType;
40 -import com.sismics.books.core.util.mime.MimeTypeUtil;
41 40 /**
42 41 * Service to fetch book informations.
43 42 */
@@ -289,7 +288,7 @@ public void downloadThumbnail(Book book, String imageUrl) throws Exception {
289 288     imageConnection.setConnectTimeout(10000);
290 289     imageConnection.setReadTimeout(10000);
291 290     try (InputStream inputStream = new BufferedInputStream(imageConnection.getInputStream())) {
292 -         if (MimeTypeUtil.guessMimeType(inputStream) != MimeType.IMAGE_JPEG) {
291 +         if (MimeType.guessMimeType(inputStream) != MimeType.IMAGE_JPEG) {
293 292             throw new Exception("Only JPEG images are supported as thumbnails");
294 293         }
295 294     }

```

```
~ 51 books-core/src/main/java/com/sismics/books/core/util/mime/MimeType.java □
...
@@ -1,5 +1,8 @@
1  package com.sismics.books.core.util.mime;
2
3 + import java.io.InputStream;
4 + import java.io.UnsupportedEncodingException;
5 +
6  /**
7   * A collection of MIME types.
8   */
@@ -18,4 +21,52 @@ public class MimeType {
18  public static final String APPLICATION_ZIP = "application/zip";
19
20  public static final String APPLICATION_PDF = "application/pdf";
21
22
23
24  /**
25   * Try to guess the MIME type of a file by its magic number (header).
26   *
27   * @param is Stream to inspect
28   * @return MIME type
29   * @throws Exception
30   */
31
32  public static String guessMimeType(InputStream is) throws Exception {
33  byte[] headerBytes = new byte[64];
34  is.mark(headerBytes.length);
35  int readCount = is.read(headerBytes);
36  is.reset();
37
38  if (readCount <= 0) {
39    throw new Exception("Cannot read input file");
40  }
41
42  return guessMimeType(headerBytes);
43}
44
45  /**
46   * Try to guess the MIME type of a file by its magic number (header).
47   *
48   * @param headerBytes File header (first bytes)
49   * @return MIME type
50   * @throws UnsupportedEncodingException
51   */
52  public static String guessMimeType(byte[] headerBytes) throws UnsupportedEncodingException {
53  String header = new String(headerBytes, "US-ASCII");
54
55  if (header.startsWith("PK")) {
56    return MimeType.APPLICATION_ZIP;
57  } else if (header.startsWith("GIF87a") || header.startsWith("GIF89a")) {
58    return MimeType.IMAGE_GIF;
59  } else if (headerBytes[0] == ((byte) 0xFF) && headerBytes[1] == ((byte) 0xD8)) {
60    return MimeType.IMAGE_JPEG;
61  } else if (headerBytes[0] == ((byte) 0x89) && headerBytes[1] == ((byte) 0x50) && headerBytes[2] == ((byte) 0x4E) && headerBytes[3] == ((byte) 0x47) &&
62  headerBytes[4] == ((byte) 0x0D) && headerBytes[5] == ((byte) 0x0A) && headerBytes[6] == ((byte) 0x1A) && headerBytes[7] == ((byte) 0x0A)) {
63    return MimeType.IMAGE_PNG;
64  } else if (headerBytes[0] == ((byte) 0x00) && headerBytes[1] == ((byte) 0x00) && headerBytes[2] == ((byte) 0x01) && headerBytes[3] == ((byte) 0x00)) {
65    return MimeType.IMAGE_X_ICON;
66  } else if (headerBytes[0] == ((byte) 0x25) && headerBytes[1] == ((byte) 0x50) && headerBytes[2] == ((byte) 0x44) && headerBytes[3] == ((byte) 0x46)) {
67    return MimeType.APPLICATION_PDF;
68  }
69
70  return null;
71}
72}
```

## ChatGPT Solution:



ChatGPT

There are a couple of design smells in the provided code snippets:

### 1. Code Duplication:

The magic number checks for different file types are repeated in the `guessMimeType` method. This can lead to maintenance issues, as any changes or additions to supported file types would require updates in multiple places. Consider refactoring this logic to a separate method or a data structure to avoid redundancy and make it easier to maintain.

### 2. Static Utility Class with Constants:

The `MimeType` class contains only constant strings and serves as a static repository for MIME types. While having constants is reasonable, it may be beneficial to consider whether this class needs to be instantiated or if it could be made more modular and extensible. If there's a need for future extensions or customizations, a more object-oriented approach with non-static methods and the possibility of subclassing might be considered.

### 3. Exception Handling:

The `guessMimeType` method in `MimeTypeUtil` throws a generic `Exception`, which can make it challenging for users of this method to understand the specific issues that might occur. It is recommended to use more specific exceptions or create a custom exception class to provide better information about the nature of the problem.

### 4. Limited Extensibility:

The file type detection in `MimeTypeUtil` is hard-coded for specific file types. If there's a need to support additional file types or dynamically add/remove types, the current design would require modifying the code. Consider a more extensible approach, perhaps using a configuration or a strategy pattern, to allow easier adaptation to changing requirements without modifying the core logic.

Addressing these design smells would improve the maintainability, flexibility, and readability of the code.

KB

You

Can I include methods of `MimeTypeUtil` class into `MimeType` class

ChatGPT

Yes, you can consider including the methods from the `'MimeTypeUtil'` class into the `'MimeType'` class, especially if the methods in `'MimeTypeUtil'` are closely related to the functionality of the `'MimeType'` class. This can lead to a more cohesive design, where related functionality is grouped together.

However, before making such a change, consider the following:

1. **Single Responsibility Principle (SRP):** Ensure that each class has a clear and single responsibility. If the inclusion of methods from `'MimeTypeUtil'` aligns with the responsibility of `'MimeType'`, it can be a reasonable design.
2. **Code Cohesion:** Evaluate whether the methods from `'MimeTypeUtil'` logically belong to the `'MimeType'` class. If they are closely related and work together, it can enhance code cohesion.
3. **Dependencies:** Consider if there are any dependencies in `'MimeTypeUtil'` that might not be appropriate in the `'MimeType'` class. If there are dependencies that are specific to utility methods, separating them might still be a good idea.
4. **Testing:** Assess the impact on testing. If the methods in `'MimeTypeUtil'` have their own set of test cases, ensure that these tests are still valid and applicable when the methods are moved into the `'MimeType'` class.
5. **Code Readability:** Evaluate the impact on code readability. If including the methods in `'MimeType'` makes the code more readable and maintains a clear structure, it could be a beneficial change.

Here's a basic example of what the integration might look like:

java

 Copy code

```
public class MimeType {  
    // Existing MimeType constants...  
  
    public static String guessMimeType(InputStream is) throws Exception {  
        // Implementation from MimeTypeUtil...  
    }  
  
    public static String guessMimeType(byte[] headerBytes) throws UnsupportedEncodingException  
        // Implementation from MimeTypeUtil...  
    }  
}
```

### 3. Imperative Abstraction in SessionUtil

**Path:**

books-core/src/main/java/com/sismics/util/jpa

**Description:**

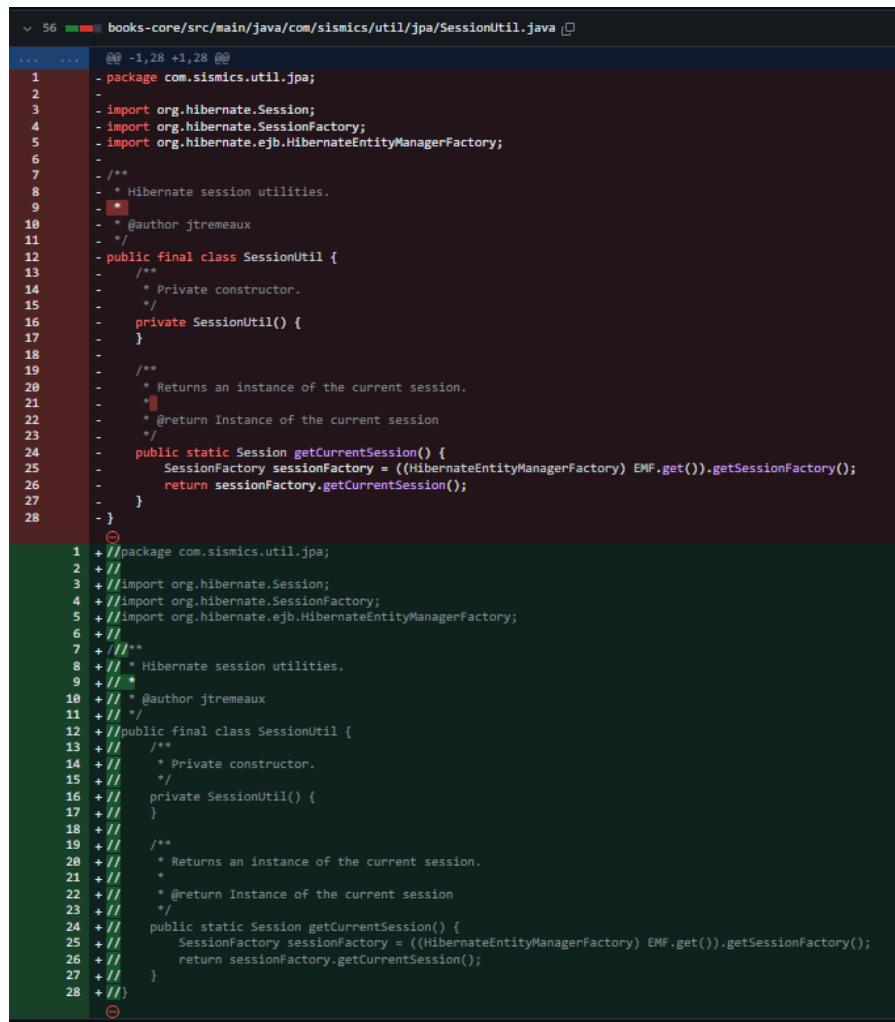
Imperative Abstraction is indicated when an operation is turned into a class that has only one method defined in it (that is used nowhere). In the given class "SessionUtil", there is only one static method that is getCurrentSession(). So here a separate class is not required. Hence it is an Imperative Abstraction Design Smell.

**Tool Used:**

DesigniteJava report

**Our Solution:**

Commented the code



The screenshot shows a code editor with the file 'SessionUtil.java' open. The code is annotated with numerous comments from the DesigniteJava tool. The annotations include:

- Line 1: `@@ -1,28 +1,28 @@`
- Line 2: `- package com.sismics.util.jpa;`
- Line 3: `+ //`
- Line 4: `- import org.hibernate.Session;`
- Line 5: `- import org.hibernate.SessionFactory;`
- Line 6: `- import org.hibernate.ejb.HibernateEntityManagerFactory;`
- Line 7: `- //`
- Line 8: `- * Hibernate session utilities.`
- Line 9: `- *`
- Line 10: `- public final class SessionUtil {`
- Line 11: `+ /**`
- Line 12: `+ * Private constructor.`
- Line 13: `+ */`
- Line 14: `- private SessionUtil() {`
- Line 15: `- }`
- Line 16: `- //`
- Line 17: `- * Returns an instance of the current session.`
- Line 18: `+ //`
- Line 19: `- * @return Instance of the current session`
- Line 20: `+ //`
- Line 21: `- public static Session getCurrentSession() {`
- Line 22: `+ SessionFactory sessionFactory = ((HibernateEntityManagerFactory) EMF.get()).getSessionFactory();`
- Line 23: `+ return sessionFactory.getCurrentSession();`
- Line 24: `- }`
- Line 25: `+ //`
- Line 26: `+ //package com.sismics.util.jpa;`
- Line 27: `+ //`
- Line 28: `+ //import org.hibernate.Session;`
- Line 29: `+ //import org.hibernate.SessionFactory;`
- Line 30: `+ //import org.hibernate.ejb.HibernateEntityManagerFactory;`
- Line 31: `+ //`
- Line 32: `+ //**`
- Line 33: `+ * Hibernate session utilities.`
- Line 34: `+ //`
- Line 35: `+ * @author jtremeaux`
- Line 36: `+ //`
- Line 37: `+ public final class SessionUtil {`
- Line 38: `+ //`
- Line 39: `+ * Private constructor.`
- Line 40: `+ */`
- Line 41: `- private SessionUtil() {`
- Line 42: `- }`
- Line 43: `+ //`
- Line 44: `+ * Returns an instance of the current session.`
- Line 45: `+ *`
- Line 46: `+ * @return Instance of the current session`
- Line 47: `+ */`
- Line 48: `- public static Session getCurrentSession() {`
- Line 49: `+ SessionFactory sessionFactory = ((HibernateEntityManagerFactory) EMF.get()).getSessionFactory();`
- Line 50: `+ return sessionFactory.getCurrentSession();`
- Line 51: `+ }`
- Line 52: `+ //`

## ChatGPT Solution:



ChatGPT

The main design smell in this code snippet is the tight coupling with Hibernate-specific classes, which could limit the reusability and maintainability of the code. Specifically:

### 1. **Hibernate Dependency:**

The `SessionUtil` class directly depends on Hibernate classes such as `Session`, `SessionFactory`, and `HibernateEntityManagerFactory`. This creates a tight coupling with the Hibernate framework. While it's common to have utility classes that abstract away certain framework details, it's generally advisable to minimize dependencies on specific frameworks, as it can make the code less flexible and harder to switch to another persistence solution in the future.

### 2. **Static Utility Class:**

The class is declared as a final class with a private constructor, indicating that it's intended to be used as a static utility class. Static utility classes with hard dependencies on specific frameworks can be challenging to test and make it difficult to substitute alternative implementations.

Consider using dependency injection or creating an interface for the utility class to make it more flexible and testable.

### 3. **Unused Code:**

If this code snippet is not used anywhere in the project, it represents dead code. Unused code can clutter the codebase and make it harder to understand and maintain. It's generally a good practice to remove unused code to keep the codebase clean and reduce confusion.

To improve the design, you might consider decoupling the code from Hibernate dependencies, making it more modular and independent. If the code is not used, it might be a candidate for removal unless it is expected to be used in the future. If Hibernate-specific functionality is needed, it could be encapsulated behind an interface to enable easier substitution or extension with alternative persistence solutions.

## **4. Imperative Abstraction in StreamUtil**

### **Path:**

books-core/src/main/java/com/sismics/books/core/util

### **Description:**

It contains only one method and has no usage.

### **Tool Used:**

DesigniteJava report

## Our Solution:

Commented the code

```
 60 books-core/src/main/java/com/sismics/books/core/util/StreamUtil.java □
...
1  @@ -1,33 +1,33 @@
2  - package com.sismics.books.core.util;
3  + // package com.sismics.books.core.util;
4
5  - import java.io.IOException;
6  - import java.io.InputStream;
7  - import java.io.PushbackInputStream;
8  - import java.util.zip.GZIPInputStream;
9
10 - // import java.io.IOException;
11 - // import java.io.InputStream;
12 - // import java.io.PushbackInputStream;
13 - // import java.util.zip.GZIPInputStream;
14
15 - /**
16 -  * Detects if the stream is gzipped, and returns a uncompressed stream according to this.
17 -  *
18 -  * @param is InputStream
19 -  * @return InputStream
20 -  * @throws IOException
21 -  */
22 public static InputStream detectGzip(InputStream is) throws IOException {
23     PushbackInputStream pb = new PushbackInputStream(is, 2);
24     byte [] signature = new byte[2];
25     pb.read(signature);
26     pb.unread(signature);
27     if(signature[0] == (byte) GZIPInputStream.GZIP_MAGIC && signature[1] == (byte) (GZIPInputStream.GZIP_MAGIC >> 8)) {
28         return new GZIPInputStream(pb);
29     } else {
30         return pb;
31     }
32 }
33
34 /**
35  * Detects if the stream is gzipped, and returns a uncompressed stream according to this.
36  *
37  * @param is InputStream
38  * @return InputStream
39  * @throws IOException
40  */
41 public static InputSteam detectGzip(InputSteam is) throws IOException {
42     PushbackInputStream pb = new PushbackInputStream(is, 2);
43     byte [] signature = new byte[2];
44     pb.read(signature);
45     pb.unread(signature);
46     if(signature[0] == (byte) GZIPInputStream.GZIP_MAGIC && signature[1] == (byte) (GZIPInputStream.GZIP_MAGIC >> 8)) {
47         return new GZIPInputStream(pb);
48     } else {
49         return pb;
50     }
51 }
52
53 + //
```

## ChatGPT Solution:



ChatGPT

The provided code snippet appears to be a utility class for detecting whether an `InputStream` is gzipped and, if so, returning an uncompressed stream. While the code itself is simple and serves a specific purpose, there are a couple of considerations:

### 1. Unused Code:

As you mentioned, if this code is not used anywhere in the project, it represents unused or dead code. Unused code should generally be removed from the codebase to keep it clean and avoid confusion.

### 2. Limited Flexibility:

The method `detectGzip` assumes that the only compression format it needs to handle is gzip. If there's a possibility of encountering other compression formats in the future, the design could be enhanced to handle different compression algorithms more flexibly. This might involve a more extensible approach, allowing for the addition of new detectors or strategies.

### 3. Checked Exception:

The `detectGzip` method declares that it throws an `IOException`. While this is a valid approach, it might be worth considering whether it makes sense for the utility method to expose this exception or if it can be handled internally. Checked exceptions can sometimes make the API more cumbersome to use, and in this case, it might be reasonable to handle the `IOException` within the method and possibly log or wrap it in an unchecked exception if needed.

In summary, while the provided code is functional, it might be a candidate for removal if it's not used. If it is intended to be used in the future, consider making the design more flexible and handling exceptions in a way that aligns with the overall goals and requirements of the codebase.

## **5. Unutilized Abstraction in ConnectResource**

### **Path:**

books-web/src/main/java/com/sismics/books/rest/resource

### **Description:**

This smell arises when an abstraction is left unused (either not directly used or not reachable).

No usages of ConnectResource anywhere.

### **Tool Used:**

DesigniteJava report

### **Our Solution:**

Commented the code

## ChatGPT Solution:



ChatGPT

The provided code snippet doesn't exhibit any significant design smell. However, there are a few considerations:

**1. Unused Code:**

As you mentioned, if this class is not used anywhere in the project, it represents unused or dead code. Removing unused code is generally a good practice to keep the codebase clean and maintainable.

**2. Authentication Logic Duplication:**

The authentication logic (``if (!authenticate())``) is repeated in each method. Consider extracting this logic into a shared method or a filter to avoid code duplication and ensure consistent authentication handling across different methods.

**3. Validation Util Class:**

The `ValidationUtil` class is referenced for validating a string, but its implementation is not provided in the code snippet. Make sure that this utility class is appropriately implemented and handles validation requirements effectively.

**4. Exception Handling:**

The exception handling is quite detailed and specific, which is good for providing clear error messages. However, you might consider using more specific exception types or creating custom exception classes to represent different error scenarios more precisely.

**5. Method Size:**

Some methods are relatively long, such as the `add` method. Consider breaking down large methods into smaller, more focused methods to improve readability and maintainability.

In summary, while the code is generally well-structured and follows best practices, you might want to address the unused code, consider refactoring the authentication logic, and ensure that referenced utility classes are appropriately implemented.

## 6. Broken Hierarchy in LocaleResource

### Path:

books-web/src/main/java/com/sismics/books/rest/resource

### Description:

This smell arises when a supertype and its subtype conceptually do not share an "IS-A" relationship resulting in broken substitutability.

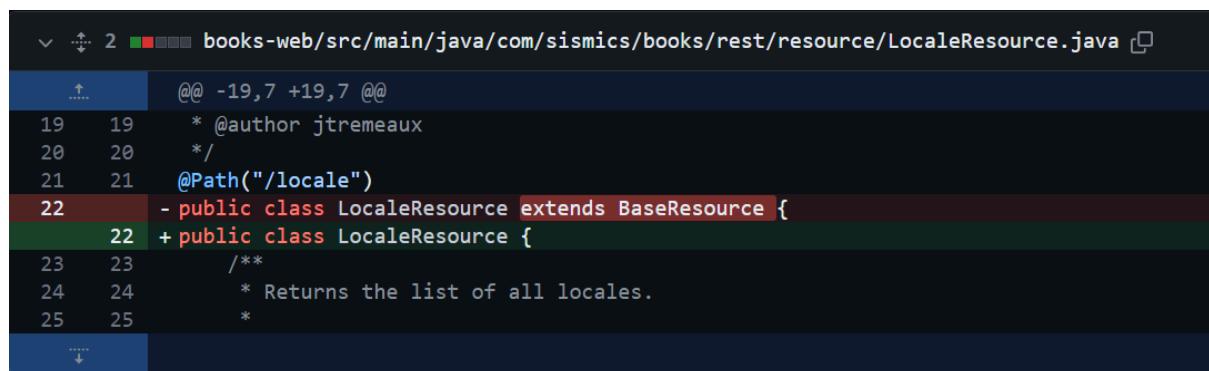
LocaleResource extends BaseResource but does not utilize any function of BaseResource. Hence it is not needed to extend it.

### Tool Used:

DesigniteJava report

### Our Solution:

Removed "extends BaseResource"



```
@@ -19,7 +19,7 @@
19   19     * @author jtremieux
20   20     */
21   21     @Path("/locale")
22 - public class LocaleResource extends BaseResource {
22 + public class LocaleResource {
23   23     /**
24   24     * Returns the list of all locales.
25   25     *
```

### ChatGPT Solution:

The `BaseResource` class combines responsibilities related to authentication (`authenticate()`, `checkBaseFunction()`, etc.) and resource-specific functionality. While it's common to have a base class for shared functionality, mixing authentication concerns with resource-specific logic might lead to a lack of separation of concerns. Consider extracting authentication-related methods into a separate class or filter to improve modularity.



## ChatGPT

The provided code does not exhibit a clear example of a broken hierarchy. The inheritance hierarchy is relatively straightforward: `LocaleResource` extends `BaseResource`, and there is no apparent violation of the Liskov Substitution Principle (LSP) or other issues that typically indicate a broken hierarchy.

However, the usage of the term "broken hierarchy" might be context-dependent, and it could refer to issues such as inappropriate use of inheritance, violation of the Liskov Substitution Principle, or other problems related to the design of class hierarchies.

In the provided code, the `BaseResource` seems to serve its purpose as a base class for common functionality shared among different resource classes. It contains methods for authentication and authorization checks that can be reused by other resource classes.

## **7. Imperative Abstraction in LocaleResource**

**Path:**

books-web/src/main/java/com/sismics/books/rest/resource

**Description:**

Only 1 method present in class and no where used

**Tool Used:**

DesigniteJava report

**Our Solution:**

Commented the code

```
 88 books-web/src/main/java/com/sismics/books/rest/resource/LocaleResource.java
```

```
... ... @@ -1,44 +1,44 @@
1 package com.sismics.books.rest.resource;
2
3 import com.sismics.books.core.dao.jpa.LocaleDao;
4 import com.sismics.books.core.model.jpa.Locale;
5 import org.codehaus.jettison.json.JSONException;
6 import org.codehaus.jettison.json.JSONObject;
7
8 import javax.ws.rs.GET;
9 import javax.ws.rs.Path;
10 import javax.ws.rs.Produces;
11 import javax.ws.rs.core.MediaType;
12 import javax.ws.rs.core.Response;
13 import java.util.ArrayList;
14 import java.util.List;
15
16 /**
17 * Locale REST resources.
18 */
19 /**
20 * @author jtremieux
21 */
22 @Path("/locale")
23 public class LocaleResource {
24     /**
25      * Returns the list of all locales.
26      *
27      * @return Response
28      * @throws JSONException
29      */
30     @GET
31     @Produces(MediaType.APPLICATION_JSON)
32     public Response list() throws JSONException {
33         LocaleDao localeDao = new LocaleDao();
34         List<Locale> localeList = localeDao.findAll();
35         JSONObject response = new JSONObject();
36         List<JSONObject> items = new ArrayList<>();
37         for (Locale locale : localeList) {
38             JSONObject item = new JSONObject();
39             item.put("id", locale.getId());
40             items.add(item);
41         }
42         response.put("locales", items);
43         return Response.ok().entity(response).build();
44     }
45 }
1+ //package com.sismics.books.rest.resource;
2+
3+ //import com.sismics.books.core.dao.jpa.LocaleDao;
4+ //import com.sismics.books.core.model.jpa.Locale;
5+ //import org.codehaus.jettison.json.JSONException;
6+ //import org.codehaus.jettison.json.JSONObject;
7+
8+ //import javax.ws.rs.GET;
9+ //import javax.ws.rs.Path;
10+ //import javax.ws.rs.Produces;
11+ //import javax.ws.rs.core.MediaType;
12+ //import javax.ws.rs.core.Response;
13+ //import java.util.ArrayList;
14+ //import java.util.List;
15+
16+ /**
17+ * Locale REST resources.
18+ */
19+ /**
20+ * @author jtremieux
21+ */
22+ //public class LocaleResource {
23+     /**
24+      * Returns the list of all locales.
25+      *
26+      * @return Response
27+      * @throws JSONException
28+      */
29+     @GET
30+     @Produces(MediaType.APPLICATION_JSON)
31+     public Response list() throws JSONException {
32+         LocaleDao localeDao = new LocaleDao();
33+         List<Locale> localeList = localeDao.findAll();
34+         JSONObject response = new JSONObject();
35+         List<JSONObject> items = new ArrayList<>();
36+         for (Locale locale : localeList) {
37+             JSONObject item = new JSONObject();
38+             item.put("id", locale.getId());
39+             items.add(item);
40+         }
41+         response.put("locales", items);
42+         return Response.ok().entity(response).build();
43+     }
44+ }
```

## ChatGPT Solution:

If the `LocaleResource` class is not used anywhere in the codebase, it could be considered as a form of "Dead Code," which is a design smell. Dead code refers to code that is no longer executed or used, and it can clutter the codebase, making it harder to understand and maintain. Here are steps you can take to address this issue:

- **Remove Unused Code:**
  - If it's confirmed that the class is not used and serves no purpose, consider removing it from the codebase. This can help reduce code clutter and improve code maintainability.
- **Documentation:**
  - If there's a reason to keep the class for future use or as a placeholder, add a comment explaining its purpose and indicating that it's intentionally not used at the moment.
- **Refactoring:**
  - If the class was meant to serve a specific purpose but is not used, consider whether its functionality can be integrated elsewhere or if it needs to be redesigned to better fit the current architecture.

## 8. Broken Hierarchy in ThemeResource

### Path:

books-web/src/main/java/com/sismics/books/rest/resource/ThemeResource.java

### Description:

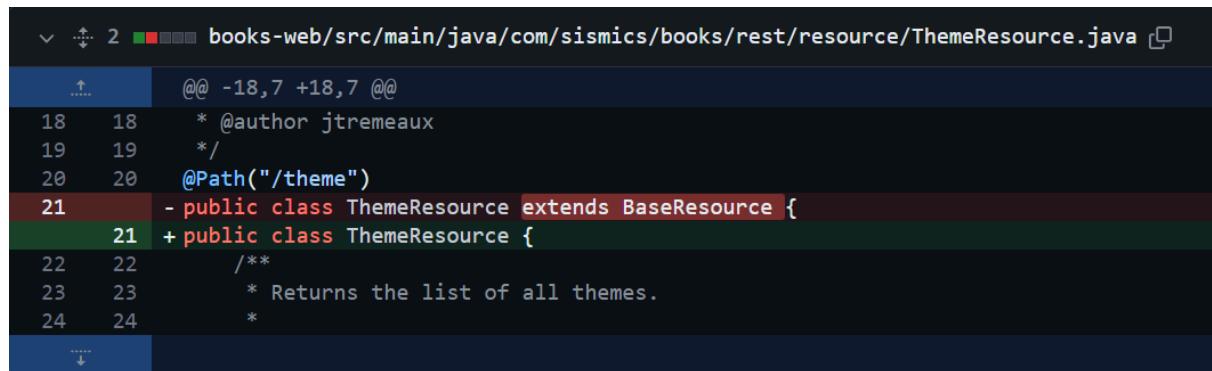
LocaleResource extends BaseResource but does not utilize any function of BaseResource. Hence it is not needed to extend it.

### Tool Used:

DesigniteJava report

### Our Solution:

Removed “extends BaseResource”



The screenshot shows a code diff interface comparing two versions of the file books-web/src/main/java/com/sismics/books/rest/resource/ThemeResource.java. The left column shows the original code, and the right column shows the modified code. The line '- public class ThemeResource extends BaseResource {' is highlighted in red, indicating it has been removed. The line '+ public class ThemeResource {' is highlighted in green, indicating it has been added. The code also includes annotations like @Path("/theme") and @author jtremeaux.

```
@@ -18,7 +18,7 @@
18   18     * @author jtremeaux
19   19   */
20   20   @Path("/theme")
21 - public class ThemeResource extends BaseResource {
21 + public class ThemeResource {
22   22     /**
23   23     * Returns the list of all themes.
24   24     *
```

## ChatGPT Solution:



ChatGPT

In the provided code, if the `'ThemeResource'` class does not use any functionality from `'BaseResource'` and simply extends it without a clear reason, it could be considered a potential design smell. This situation might indicate an inappropriate use of inheritance, violating the principle of favoring composition over inheritance when there is no real "is-a" relationship between the classes.

Here are a few considerations:

1. **Unused Inheritance:**

If `'ThemeResource'` does not benefit from the functionality provided by `'BaseResource'` and does not override or use any of its methods or attributes, then the inheritance might be unnecessary. Inheritance should generally represent an "is-a" relationship, and if it doesn't make sense in this context, it might be better to refactor or rethink the class hierarchy.

2. **Composition Instead of Inheritance:**

If `'ThemeResource'` needs certain functionality, it might be more appropriate to use composition (favoring composition over inheritance). Instead of extending `'BaseResource'`, you could encapsulate the desired functionality in a separate class and use an instance of that class within `'ThemeResource'`.

3. **Code Duplication:**

If the inheritance is unnecessary, it could lead to code duplication and make the codebase harder to maintain. Code that is not contributing to the functionality of a class should be avoided to keep the codebase clean and clear.

Consider reviewing the design and class hierarchy to ensure that the use of inheritance aligns with the principles of object-oriented design and that each class extends another only when there is a clear need for shared behavior or specialization. If there is no apparent need for `'ThemeResource'` to extend `'BaseResource'`, it might be worth reconsidering the design.

## 9. Missing Interface hierarchy in LocaleResource and ThemeResource

### Path:

books-web/src/main/java/com/sismics/books/rest/resource/ThemeResource.java

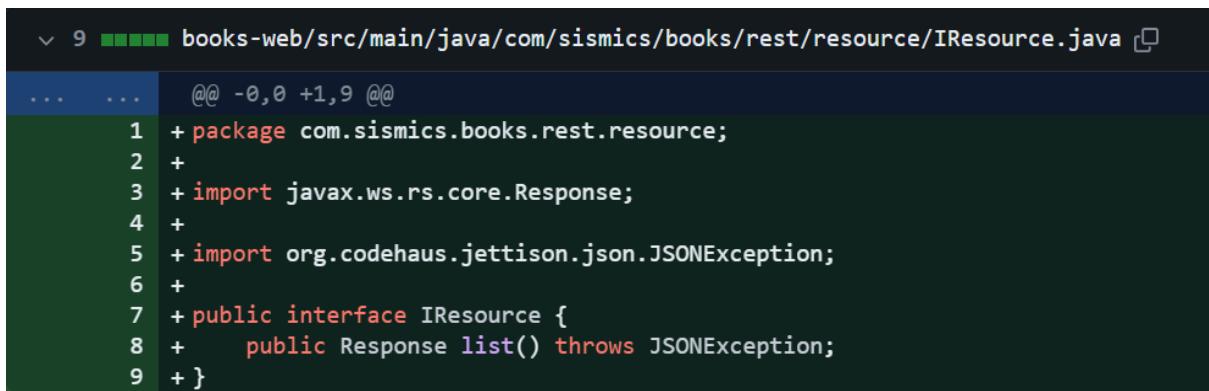
### Path:

books-web/src/main/java/com/sismics/books/rest/resource/LocaleResource.java

### Description:

The classes LocaleResource and ThemeResource both have only one method list(). This shows that they both can implement a common interface. In future, such common functions of both ThemeResource and LocaleResource can be added to the same interface.

### Our Solution:



A screenshot of a code editor showing a diff view. The top bar indicates the file is 'books-web/src/main/java/com/sismics/books/rest/resource/IResource.java'. The diff shows the following changes:

```
@@ -0,0 +1,9 @@
+ package com.sismics.books.rest.resource;
+
+ import javax.ws.rs.core.Response;
+
+ import org.codehaus.jettison.json.JSONException;
+
+ public interface IResource {
+     public Response list() throws JSONException;
+ }
```

```
✓ 88  books-web/src/main/java/com/sismics/books/rest/resource/LocaleResource.java □
1  @@ -1,44 +1,44 @@
2  - package com.sismics.books.rest.resource;
3  - /**
4  -  * Import com.sismics.books.core.dao.jpa.LocaleDao;
5  -  * Import com.sismics.books.core.model.jpa.Locale;
6  -  * Import org.codehaus.jettison.json.JSONException;
7  -  * Import org.codehaus.jettison.json.JSONObject;
8  -  */
9  - Import javax.ws.rs.GET;
10 - Import javax.ws.rs.Path;
11 - Import javax.ws.rs.Produces;
12 - Import javax.ws.rs.core.MediaType;
13 - Import javax.ws.rs.core.Response;
14 - Import java.util.ArrayList;
15 - Import java.util.List;
16 - /**
17 -  * Locale REST resources.
18 -  *
19 -  * @author jtremieux
20 -  */
21 - @Path("/locale")
22 - public class LocaleResource {
23 -     /**
24 -      * Returns the list of all locales.
25 -      *
26 -      * @return Response
27 -      * @throws JSONException
28 -     */
29 -     @GET
30 -     @Produces(MediaType.APPLICATION_JSON)
31 -     public Response list() throws JSONException {
32 -         LocaleDao localeDao = new LocaleDao();
33 -         List<Locale> localeList = localeDao.findAll();
34 -         JSONObject response = new JSONObject();
35 -         List<JSONObject> items = new ArrayList<>();
36 -         for (Locale locale : localeList) {
37 -             JSONObject item = new JSONObject();
38 -             item.put("id", locale.getId());
39 -             items.add(item);
40 -         }
41 -         response.put("locales", items);
42 -         return Response.ok().entity(response).build();
43 -     }
44 - }
1 + package com.sismics.books.rest.resource;
2 +
3 + Import com.sismics.books.core.dao.jpa.LocaleDao;
4 + Import com.sismics.books.core.model.jpa.Locale;
5 + Import org.codehaus.jettison.json.JSONException;
6 + Import org.codehaus.jettison.json.JSONObject;
7 +
8 + Import javax.ws.rs.GET;
9 + Import javax.ws.rs.Path;
10 + Import javax.ws.rs.Produces;
11 + Import javax.ws.rs.core.MediaType;
12 + Import javax.ws.rs.core.Response;
13 + Import java.util.ArrayList;
14 + Import java.util.List;
15 +
16 + /**
17 + * Locale REST resources.
18 + *
19 + * @author jtremieux
20 + */
21 + @Path("/locale")
22 + public class LocaleResource implements IResource{
23 +     /**
24 +      * Returns the list of all locales.
25 +      *
26 +      * @return Response
27 +      * @throws JSONException
28 +     */
29 +     @GET
30 +     @Produces(MediaType.APPLICATION_JSON)
31 +     public Response list() throws JSONException {
32 +         LocaleDao localeDao = new LocaleDao();
33 +         List<Locale> localeList = localeDao.findAll();
34 +         JSONObject response = new JSONObject();
35 +         List<JSONObject> items = new ArrayList<>();
36 +         for (Locale locale : localeList) {
37 +             JSONObject item = new JSONObject();
38 +             item.put("id", locale.getId());
39 +             items.add(item);
40 +         }
41 +         response.put("locales", items);
42 +         return Response.ok().entity(response).build();
43 +     }
44 + }
```

```
v ⌂ 2 books-web/src/main/java/com/sismics/books/rest/resource/ThemeResource.java □
.. @@ -18,7 +18,7 @@
18   18     * @author jtremeaux
19   19     */
20   20     @Path("/theme")
21 - public class ThemeResource {
21 + public class ThemeResource implements IResource{
22   22     /**
23   23     * Returns the list of all themes.
24   24     *
...  
↓
```

## **Code Metrics Analysis**

## **Before Refactoring (2b)**

### **Code Metrics Analysis :**

During the Initial Stage of our Project, we did a comprehensive Code metrics Analysis. After the first analysis, we find the following Code Metrics.

### **Code Metrics Used, their Meaning, and the Impact on Software Quality:**

#### **1. Line of Code**

- a. Meaning: Line of Code counts the number of lines in the source code of a software project.
- b. Impact on Software Quality:
  - i. Quality and Maintainability: A higher LOC can indicate a larger, potentially more complex project, which may be harder to maintain. However, LOC alone isn't a reliable measure of complexity or quality. It's possible to have a small codebase that's poorly written or a large one that's well-organized and maintainable.
  - ii. Potential Performance Issues: While LOC doesn't directly impact performance, a larger codebase might lead to longer compile times and potentially more bugs, which can indirectly affect performance.

#### **2. Reference for class**

- a. Meaning: This metric tells us how often a class is referenced or used within a project. It could be an indicator of a class's centrality or importance in the application architecture.
- b. Impact on Software Quality:
  - i. Quality and Maintainability: Classes that are heavily referenced might be critical to the application and could represent high-risk areas for changes. If such a class has issues, it might affect many parts of the system.
  - ii. Potential Performance Issues: High interdependence (many references to a class) can lead to tight coupling, making the system less flexible and possibly leading to performance bottlenecks.

### **3. Cyclomatic Complexity**

- a. Meaning: Cyclomatic Complexity measures the number of linearly independent paths through a program's source code. It's calculated using the control flow graph of the program, where each node represents a block of code, and edges represent control flow paths.
- b. Impact on Software Quality:
  - i. Quality and Maintainability: High cyclomatic complexity indicates a more complex and potentially less maintainable codebase. Complex code is harder to understand, test, and modify, increasing the risk of bugs.
  - ii. Potential Performance Issues: While not directly related to performance, complex code can lead to inefficient algorithms that degrade performance. Also, complex code might be more challenging to optimize.

### **4. Lack of cohesion of methods**

- a. Meaning: LCOM measures the cohesiveness of methods in a class, based on the methods of the same class should operate on the same class attributes. High LCOM indicates low cohesion, where class methods are not related to each other, suggesting that the class might be doing too many unrelated things.
- b. Impact on Software Quality:
  - i. Quality and Maintainability: Classes with low cohesion (high LCOM) can be harder to maintain and understand. Such classes might need to be refactored into smaller, more focused classes, improving modularity and maintainability.
  - ii. Potential Performance Issues: Indirectly, low cohesion can lead to unnecessary dependencies and overhead, as objects of the class may carry more data and behavior than required for particular operations, potentially affecting performance.

### **5. Coupling Between Object Classes**

- a. Meaning: CBO measures how many classes are coupled to a given class, either through method calls, variable accesses, or inheritance. High CBO indicates a class is highly dependent on other classes.
- b. Impact on Software Quality:
  - i. Quality and Maintainability: High coupling makes a system more brittle and harder to change because modifications in one class can require changes in all coupled classes. Lower coupling enhances modularity, making the system easier to understand, modify, and maintain.
  - ii. Potential Performance Issues: High coupling can lead to performance issues since changes in one part of the system

may have widespread effects, requiring more extensive testing and potentially leading to less efficient code execution.

## Tools Used :

1. CodeMR
  - a. CodeMR is an architectural software quality and static code analysis tool
2. Designite
  - a. Designite is a powerful tool for assessing the quality of Java code, providing valuable insights through metrics analysis. It helps identify and address issues to enhance the overall performance and maintainability of the codebase.

## Implications Discussion :

### Distribution of some other Quality Attributes



## Color Code Key:

- Very High
- High
- Medium-high
- Low-medium
- Low

### 1. Lines Of Code

Before refactoring we have 4505 lines of code. And we didn't consider Files located in any 'test' directories within the 'src' directory are ignored.  
Files with names that end in 'Test'.  
Files with names that begin with 'Test'.

General Information

**Total lines of code: 4505**

**Number of classes: 113**

**Number of packages: 36**

**Number of external packages: 17**

**Number of external classes: 128**

**Number of problematic classes: 0**

**Number of highly problematic classes: 0**

Here, we haven't considered any test files, and only JAVA files are considered for LOC.

ModelName:

se-project-1--\_23

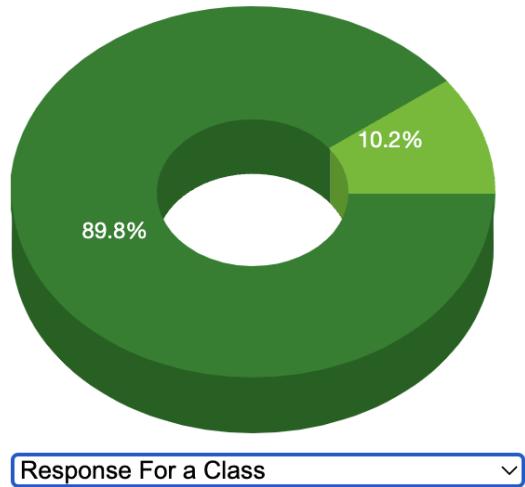
Generate CodeMR graphs:

Ignore files with path contains '\*/src/test/\*'

Ignore files with name '\*Test':

Ignore files with name 'Test\*':

## 2. Reference for a class



"Response for a Class" metric measures the number of methods that can be invoked in response to a message to the object or, more simply, the count of the number of methods in a class plus the number of methods accessible to an object class due to a method call. So, our aim is to decrease the classes with RFC.

## 3. Cyclomatic Complexity

"Cyclomatic Complexity" is a code metric used to measure the complexity of a program's control flow. It quantifies the number of linearly independent paths through a method or function, providing insights into its complexity, maintainability, and testability.

We find Cyclomatic complexity using DesigniteJava and wanted to reduce it after Refactoring.

Cyclomatic complexity is calculated using a formula based on the number of decision points in the code. For example, the formula might be:

$$CC = E - N + 2P$$

Where:

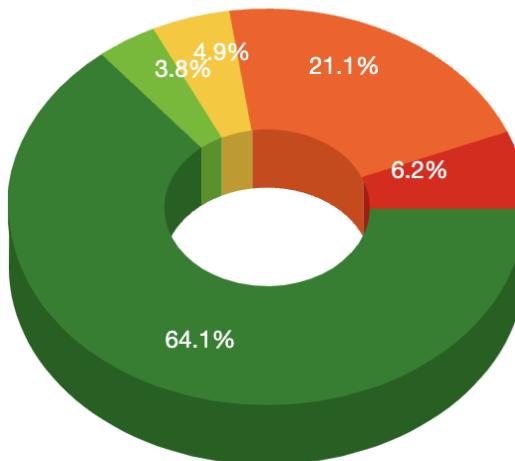
CC = Cyclomatic Complexity

E = Number of edges in the control flow graph

N = Number of nodes in the control flow graph

P = Number of connected components (typically 1 for most programs)

#### 4. Lack of cohesion of methods

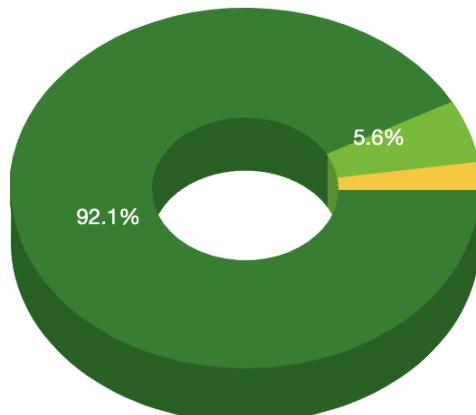


Lack of Cohesion of Methods ▾

"Lack of Cohesion of Methods" is used to measure the cohesion within a class in OOPS programming. Cohesion refers to the degree to which the responsibilities of a module or class are related and focused.

When a class has a low cohesion of methods, it means that its methods are not tightly related to each other in terms of functionality. Instead, the class may contain unrelated or loosely related sets of methods. So, our aim is to decrease this metric during Refactoring.

#### 5. Coupling Between Object Classes



Coupling Between Object Classes ▾

"Coupling Between Object Classes" (CBO) is used to measure the level of dependency between different classes in an object-oriented system. It quantifies how many other classes a particular class is directly coupled to. Lowering the Coupling between different Object Classes should be one of the major focus area while refactoring.

## After Refactoring (3b)

Re-evaluating the metrics previously assessed in task 2b.

### Analysis and Comparison

#### Dashboard Of CodeMR



#### 1. Line Of Code

Lines of Code (LOC) calculated by counting all the lines in the source files of the project, except for those that have been explicitly excluded based on the specified filters:

Files located in any 'test' directories within the 'src' directory are ignored.

Files with names that end in 'Test'.

Files with names that begin with 'Test'.

## After Refactoring:

After Refactoring the lines of code are decreased

General Information

**Total lines of code: 4471**

**Number of classes: 108**

**Number of packages: 36**

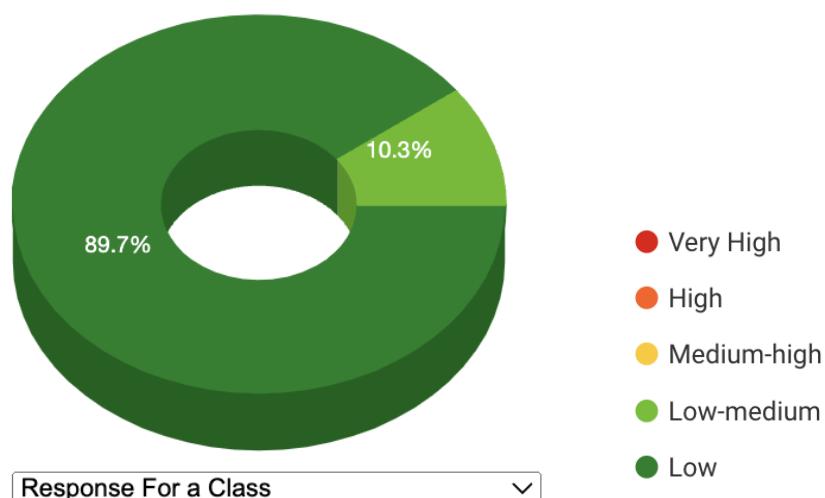
**Number of external packages: 17**

**Number of external classes: 125**

**Number of problematic classes: 0**

**Number of highly problematic classes: 0**

## 2. Reference for class



## Analysis:

In the Original code it shows that 89.8% of the classes have a certain level of RFC, and in the Refactored code after changes, it shows 89.7%. This indicates a very slight decrease in the RFC value.

## Change Imply:

The Change in Responsiveness is very minimal (0.1%), indicating there was reduced overall method calls within or outside the class. This is because methods being removed, refactored.

This could be an effort to simplify the code or improve its design.

### **Alignment with expectations:**

The decrease in RFC supports our expectations, indicating a slight enhancement in maintainability and quality, primarily attributed to the removal of redundant or unused code.

## **3. Cyclomatic Complexity**

### **Analysis:**

We use DesigniteJava for finding Cyclomatic Complexity, and all its components. You can find it in the below Sheet.

#### **Code Metrics**

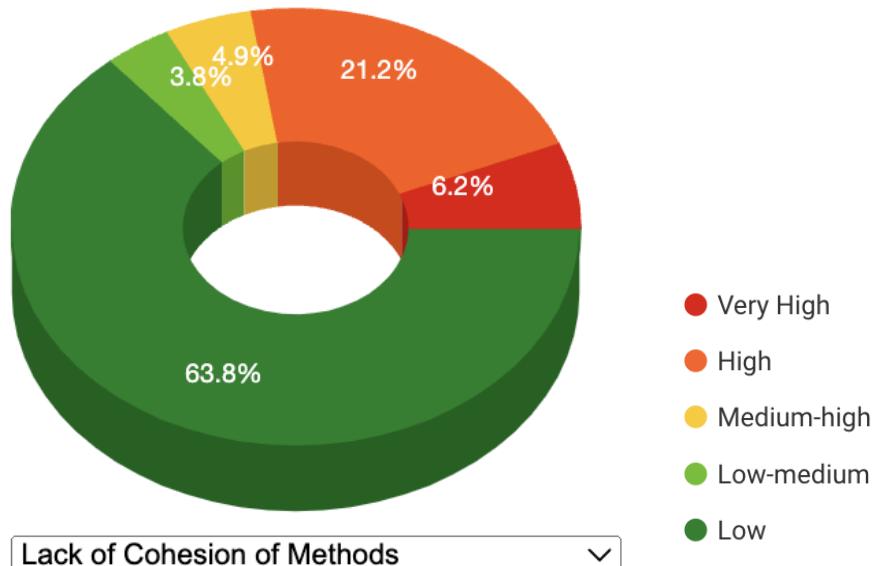
### **Change Imply:**

There is no Significant Change in Cyclomatic Complexity.

### **Alignment with expectations:**

Since, there is no significant Change in “Cyclomatic Complexity”, it doesn’t align with our Expectations.

## **4. Lack of cohesion of methods**



**Lack of Cohesion of Methods (LCOM)** measures the dissimilarity of methods within a class by looking at the sets of instance variables that each method uses.

A lower LCOM value is **generally desired** as it suggests that a class has a higher degree of cohesion. In other words, its methods are more likely to be related.

to one another in some meaningful way, which is considered a positive characteristic in object-oriented design.

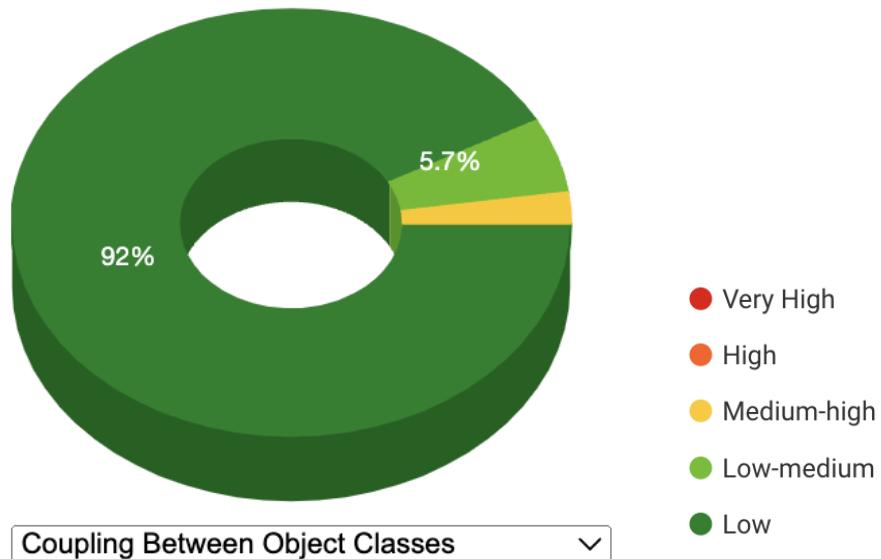
### **Changes Imply:**

Methods might have been removed, or their logic have been altered to use more common attributes of the class, thus slightly reducing the LCOM.

### **Expectations:**

The reduction in LCOM suggests an improvement in the cohesion of your classes, which is generally a positive development.

## **5. Coupling Between Object Classes**



**Coupling Between Object Classes (CBO):** refers to the degree to which classes are dependent on one another. Low coupling is generally preferred because it indicates that classes can be modified independently without affecting others, which leads to better maintainability and flexibility in the code.

### **Why It Happens:**

Before Refactoring we have 113 classes after refactor we have total 108 class out which the number of high medium remains while we remove some class which low coupling and those were redundant thus reducing the number of classes with low CBOC

### **Our Expectations:**

We also thought that the changes are not too significant that they might affect the “Coupling Between Object Classes” metric.

# Contributions

Name	Roll Number	Contribution
Karan Bhatt	2022202003	UML, Design Smell, Code Refactor, Report
Madhusree Bera	2022202007	UML, Design Smell, Code Refactor, Report
Vedashree Ranade	2022201073	UML, Design Smell, Code Refactor, Report
Yash Maheshwari	2022201074	Code Metrics, Design Smell, Code Refactor, Report
Piyush Rana	2022202012	Code Metrics, Design Smell, Code Refactor, Report