CS6.401 Software Engineering
Spring 2024
Project - 1

Team - 22

Gunank Singh Jakhar (2022201057)

M Elamparithy (2022801014)

Rishabh Gupta (2022202011)

Piyush Singh (2022201032)

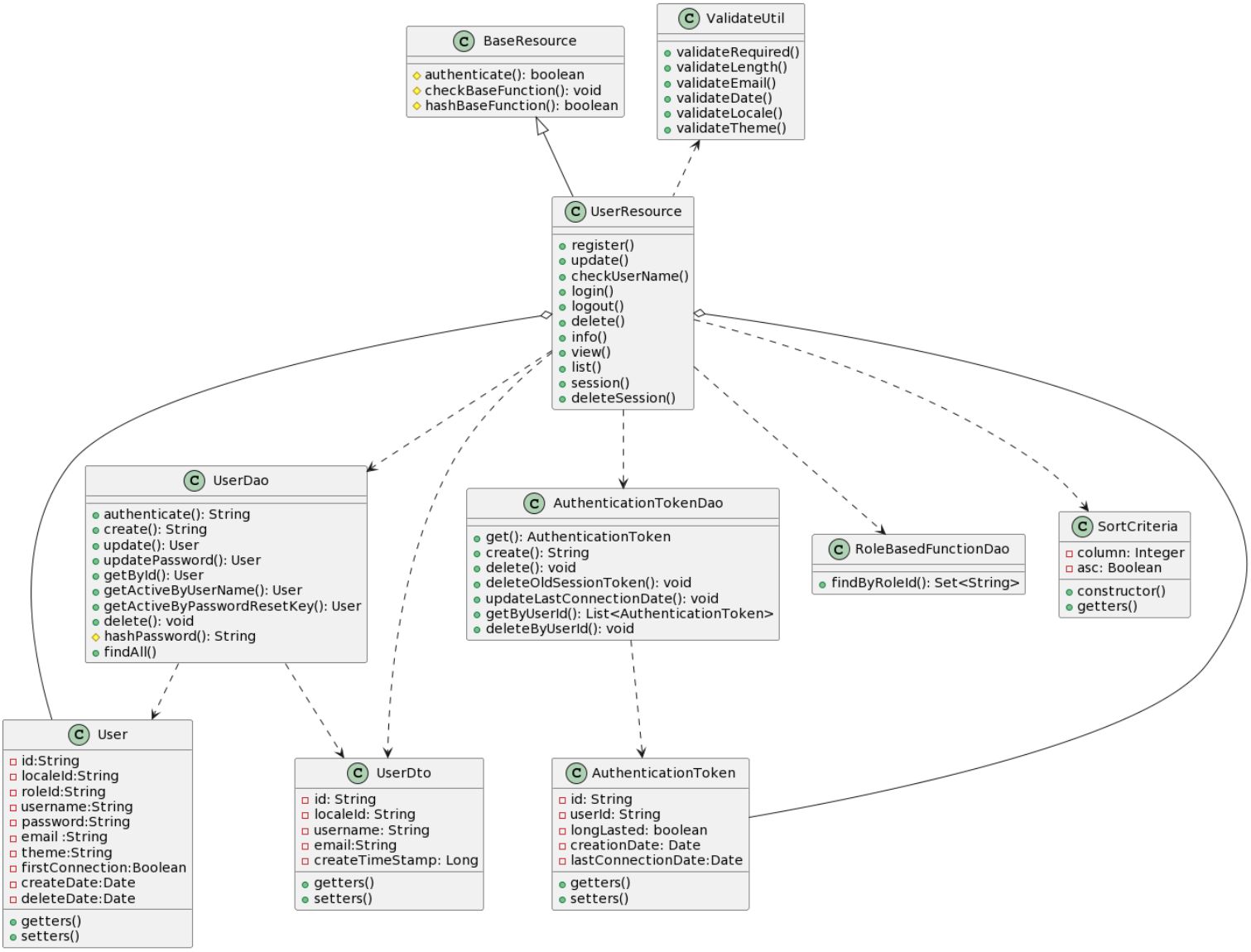
Prashant Kumar (2022201058)

Task-1 Mining the repository

The codebase follows the architectural pattern of MVC (Model View Controller). In this book's system codebase Resource classes (example: BookResource, UserResource) act as controllers for the respective subsystems, and the model classes are as usual (example: Book and User).

Created UMLs and documented their functionality and behaviors.

- **User Management Subsystem**



BaseResource:

Base class of REST resources.

authenticate(): Used to check if the user is authenticated and return true if the user is authenticated and not anonymous.

checkBaseFunction(): Check if the user has a base function.

UserResource:

register(): Takes username, password, and email and creates a new user.

update(): Update user's information like username, password, email, theme, localeid, firstconnection.

checkUserName(): Check user is available or not and it will search for active accounts.

login(): To identify the user.

logout(): Logout of the user and delete the active session of that particular user

delete(): Delete a user and all related information.

info(): Give the information about the connected user.

view(): Return the information about a user

list(): Return list of all active users .

session(): Return all active sessions.

deleteSession(): It deletes all active sessions except the one used for this request.

ValidateUtil:

This class is used to validate all the parameters.

validateRequired(): Checks whether the argument is NULL or not.

validate Length(): Check the length of the string. Length should be between lenghtMin and lengthMax.

validateEmail(): Check string type is an email.

validateDate(): Validates and parses a date.

validateLocate(): validate locateId.

validateTheme(): ThemeId of the theme to validate

User:

It's a user entity, which uses get and set functions to fetch and set the information of the user.

Each user has id, localeId, roleId, username, password, email, theme, firstConnection, createDate, deleteDate and there are getters and setters to access.

UserDao:

Its class for User Data Access object creation which further connects to the database for storing the information on it.

authenticate(): Takes username and password and authenticates a user.

create(): This creates a new user.

update(): This updates a user

updatePassword(): This method updates the user's password.

getById(): Return a user by its ID.

getActiveByUsername(): It is used for getting the active user by its username.

getActiveByPasswordResetKey(): Gets an active user by its password recovery token.

delete(): This deletes a user.

hashPassword(): Used to hash the user's password.

findAll(): It is used to return the list of all users.

UserDto:

This is the class where a data transfer object is created which is accessed by other classes as per their requirements. This class uses constructor, getter, and setter functions to do this.

This class has id, localeId, username, email, createTimestamp, and each member has its own get and set functions.

AuthenticationTokenDao:

It's a class where a Data Object for the authentication token is created and the authentication details are then saved to the database.

get(): This method takes an id of the authentication token ID and returns the authentication token.

create(): Creates a new authentication token.

delete(): This deletes the authentication token.

deleteOldSessionToken(): This deletes old short-lived tokens.

updateLastConnectionDate(): This renews the last date when the connection was established by the last authentication token.

getByUserId(): Returns a list of authentication tokens of a user.

deleteByUserId(): Deletes all authentication tokens of a user.

AuthenticationToken:

It's a class where the entity for the authentication token is created by using get and set functions.

This class has id, userId, londLasted, creationDate, lastConnectionsDate and each member has its own get and set functions.

RoleBaseFunctionDao:

findByRoleId(): This method finds the set of base functions of a role.

sortCriteria:

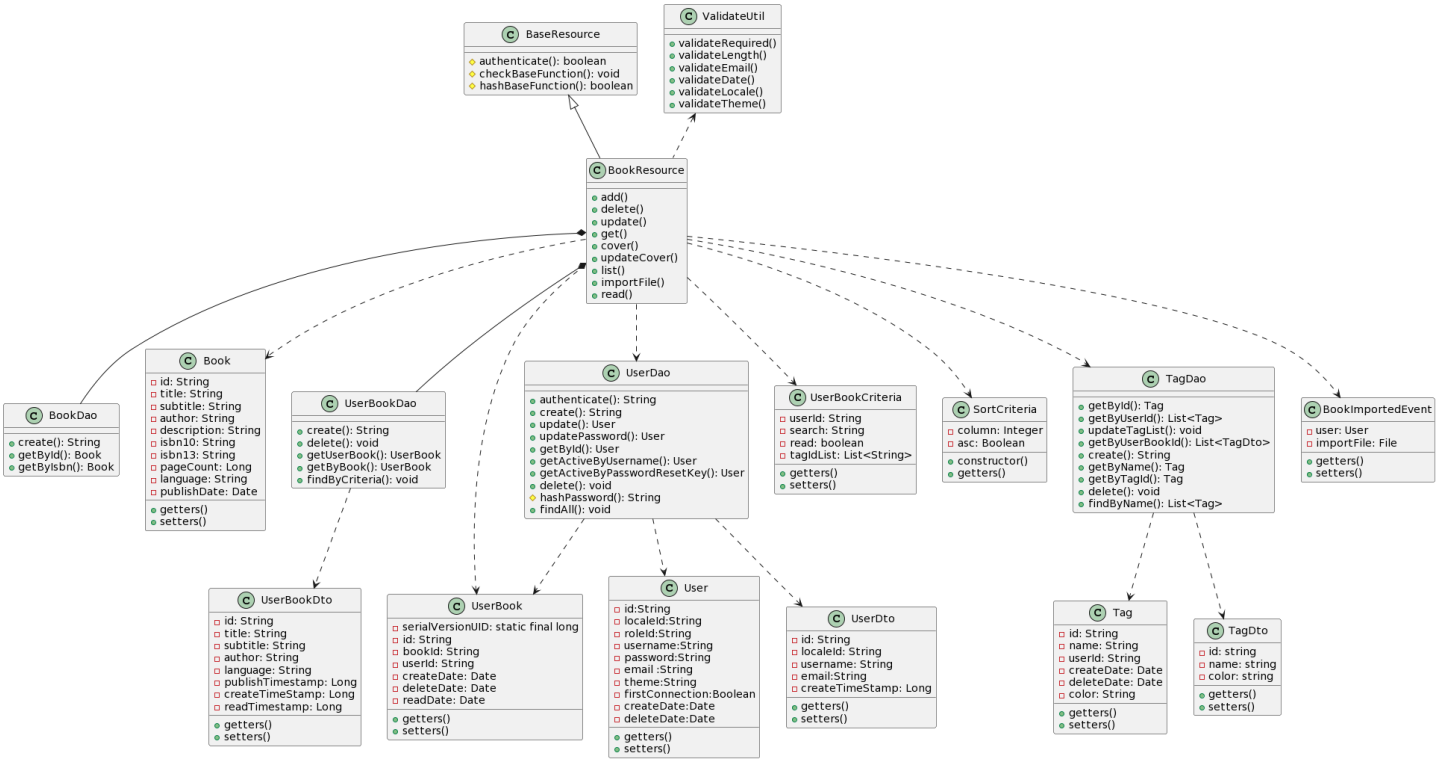
This class is used to sort the criteria of a query.

column: Index of the column to sort (first is 0).

asc: Sort in increasing order (or else decreasing).

SortCriteria(): Constructor of sortCriteria.

- **Book Addition and Display Subsystem**



BookResource:

add(): This method takes ISBN as a parameter and creates a new book. We can add a book manually also by taking different parameters like title, author, isbn10, publish_date, etc.

delete(): This deletes a book.

update(): Used to update the book.

get(): Takes user book id and returns the book.

cover(): This method returns a book cover.

updateCover(): This updates a book cover.

list(): This returns all books.

importFile(): This method is used to import books.

read(): Set a book as read/unread.

Book:

It's a book entity that uses get and set functions to fetch and set information about books.

Each book has id, title, subtitle, author, description, isbn10, isbn13, pageCount, language, publishDate, and each member has its own get and set methods.

BookDao:

create(): This creates a new book.

getbyId(): Returns a book by its ID.

getByIsbn(): Returns a book by its ISBN (10 or 13).

UserBook:

It is a class for user book entities.

UserBook has static serialVersionUID, id, bookId, userId, createDate, deleteDate, readDate, and own get and set methods.

UserBookDao:

Its class for user book Data Access Object which is connected to the database for storing the information on it.

create(): This creates a new user book.

delete(): This deletes a user book.

getUserBook(): This method takes different types of parameters and based on that returns the user book.

This function has overridden like based on userBookId, userId, and userBookId return user book.

getByBook(): This returns a user book by book ID and user ID.

findByCriteria(): This Searches user books by criteria.

UserBookCriteria:

It is a class for user book criteria and has userId, search, read, and tagIdList attributes.

TagDao:

getById(): This method returns a tag by its ID.

getByUserId(): Returns the list of all tags.

updateTagList(): It updates tags on a user book.

getByUserBookId(): It returns a list of tags on a user book.

create(): This method creates a new tag.

getByName(): Returns a tag by name.

getByTagId(): Returns a tag by ID.

delete(): It deletes a tag.

findByName(): This methods search tags by name and returns a list of tags.

Tag:

It's a class for tag entities and has id, name, userId, createDate, deleteDate, and color and each has its own get and set methods.

TagDto:

This class has id, name, and color attributes and each has its getters and setters.

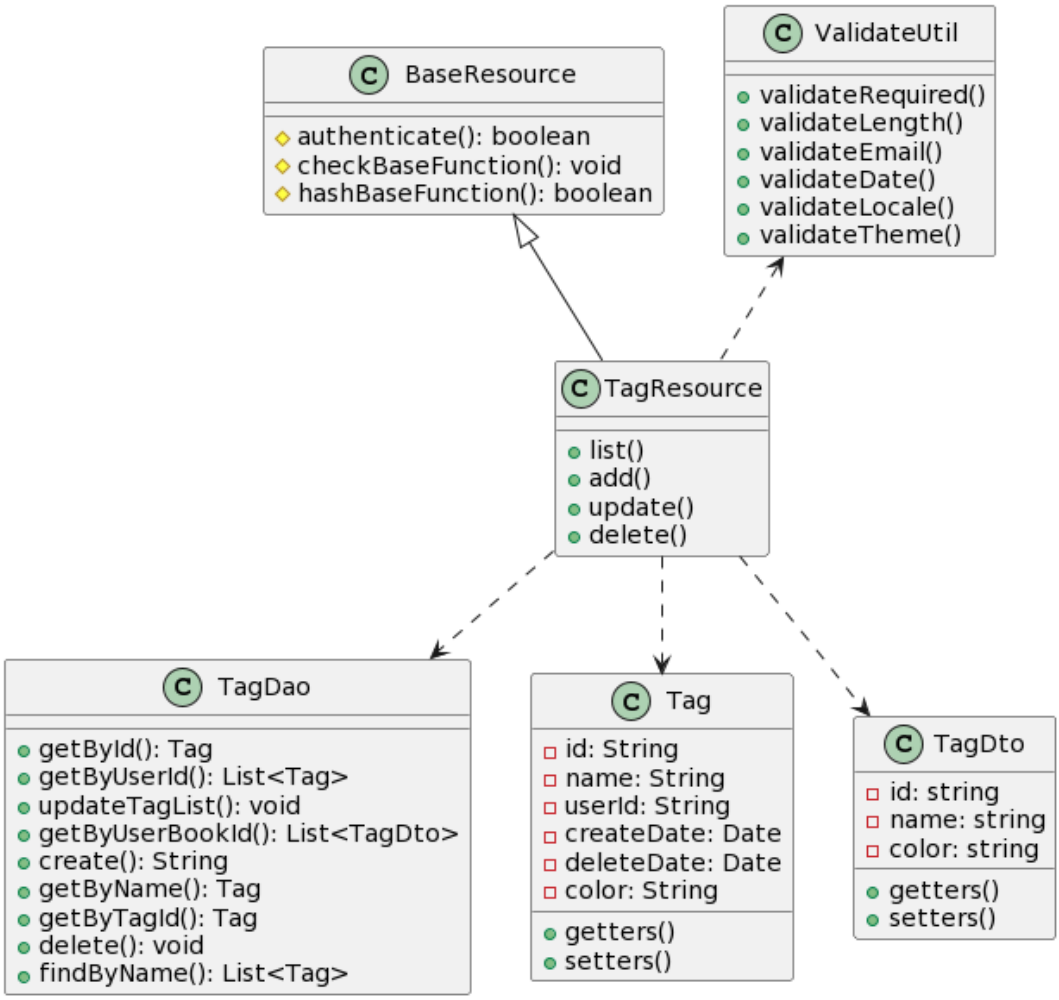
BookImportedEvent:

This event was raised on request to import books.

user: user requesting the import.

importFile: temporary file to import.

- **Bookshelf Management Subsystem:**



TagResource: This class manages the bookshelf management system.

- **Strength and Weakness of the System:**

  Strength:

  Functionality: The system has clear and specific functions organized into different categories, making it easy for users to understand what each function does. The user interface is carefully designed and includes features like adding and importing books, as well as the ability to summarize metadata. It also displays information about the number of books and user preferences, making the overall experience with book management better.

  Security: Server logs can only viewed by the admin which is one of the good practices for secure software development.

  Compatibility: The system is compatible with different external sources, enabling users to bring in books from platforms such as Goodreads, personal computers, google APIs, or open library APIs. This flexibility also applies to file formats, supporting both single-book imports and the bulk import of books in .csv format, improving overall ease of use and compatibility.

  Comprehensive Cataloging: The book management system effectively organizes a wide variety of books, giving users well-arranged and easy-to-explore libraries. Users can search, filter, and categorize books using different criteria, creating a user-friendly and efficient book browsing experience.

Multi-User Support: The system can handle many users, each with their profiles and preferences. This allows for personalized book recommendations, tracking reading progress, and maintaining individualized book lists. It improves the overall experience for avid readers and book enthusiasts.

Weakness

Complexity: The code base has quite several classes where there are unnecessary long methods that decrease the readability of the code. There was a case of unnecessary hierarchy and also missing hierarchy both of which are design smells associated with increased complexity.

Performance and Reliability: Slow performance and long response times worsen the user experience, making the overall system less usable. In some time it does not show any confirmation message for CRUD operations example: there is no message when we add a book by importing in CSV format which makes it less user-friendly.

Ineffective Search: The book management system has problems with its search function. Users struggle to find specific books because the search capabilities are limited, lacking advanced filters or sorting options. This hampers the system's ability to offer a smooth and efficient book discovery process.

Limited User Collaboration: The system doesn't have features that allow users to share book recommendations, and reviews, or work together on reading lists. The absence of these collaborative elements reduces the social aspect of book management, limiting engagement and interaction among users in the system.

User Interaction Challenges: The placement of the logout option on the login page causes confusion and may lead to accidental logouts, detracting from the user experience and lack of a visible signup option on the front page complicates the onboarding process for new users, potentially reducing user acquisition rates and hindering system usability.

Security: We have identified a vulnerability in this system, allowing unrestricted access without logging in.

- **Assumptions**

  While constructing the UML diagram, we ignored the classes that assist in working the codebase framework and just represented the classes that are core to the business logic. This was done to simplify the UML diagram.

Task-2(a) Design Smells

- **Smell-1 Deficient Encapsulation**

  A utility class only contains static members and should not be instantiated and we can use the default constructor which is public in Java making it instantiable.

TransactionUtil.java

```java
package com.sismics.books.core.util;

import com.sismics.util.context.ThreadLocalContext;

/**
 * Database transaction utils.
 *
 * @author jtremeaux
 */
public class TransactionUtil {


    /**
     * Logger.
     */
    private static final Logger log = LoggerFactory.getLogger(TransactionUtil.class);

```

- **Smell-2 Unnecessary Hierarchy**

PermissionException.java

```java
package com.sismics.books.core.service.facebook;
/**
 * Exception raised on a Facebook permission not found.
 *
 * @author jtremeaux
 */
public class PermissionException extends Exception {
    /**
     * Serial version UID.
     */
    private static final long serialVersionUID = 1L;
    /**
     * Constructor of PermissionException.
     *
     * @param message Message
     */
    public PermissionException(String message) {
        super(message);
    }
}

```

AuthenticationException.java

```java
package com.sismics.books.core.service.facebook;
/**
 * Exception raised on a Facebook authentication error.
 *
 * @author jtremeaux
 */
public class AuthenticationException extends Exception {
    /**
     * Serial version UID.
     */
    private static final long serialVersionUID = 1L;
    /**
     * Constructor of AuthenticationException.
     *
     * @param message Message
     */
    public AuthenticationException(String message) {
        super(message);
    }
}
```

Both the above classes are the same which defeats the purpose of having both of them in our system.

- **Smell-3 Insufficient Modularization:**

BookImportAsynListener

BookImportAsyncListener.java

```java
public class BookImportAsyncListener {
    /**
     * Logger.
     */
    private static final Logger log = LoggerFactory.getLogger(BookImportAsyncListener.class);

    /**
     * Process the event.
     */
    @Subscribe
    public void on(final BookImportedEvent bookImportedEvent) throws Exception {
        if (log.isInfoEnabled()) {
            log.info(MessageFormat.format("Books import requested event: {0}", bookImportedEvent.toString()));
        }

        // Create books and tags
        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                CSVReader reader = null;
                BookDao bookDao = new BookDao();
                UserBookDao userBookDao = new UserBookDao();
                TagDao tagDao = new TagDao();
                try {
                    reader = new CSVReader(new FileReader(bookImportedEvent.getImportFile()));
                } catch (FileNotFoundException e) {
                    log.error("Unable to read CSV file", e);
                }

                // Goodreads date format
                DateTimeFormatter formatter = DateTimeFormat.forPattern("yyyy/MM/dd");

                String [] line;
                try {
                    while ((line = reader.readNext()) != null) {
                        if (line[0].equals("Book Id")) {
                            // Skip header
                            continue;
                        }

                        // Retrieve ISBN number
                        String isbn = Strings.isNullOrEmpty(line[6]) ? line[5] : line[6];
                        if (Strings.isNullOrEmpty(isbn)) {
                            log.warn("No ISBN number for Goodreads book ID: " + line[0]);
                            continue;
                        }

                        // Fetch the book from database if it exists
                        Book book = bookDao.getByIsbn(isbn);
                        if (book == null) {
                            // Try to get the book from a public API
                            try {
                                book = AppContext.getInstance().getBookDataService().searchBook(isbn);
                            } catch (Exception e) {
                                continue;
```

BookImportAsyncListener.java

```
95                          }
96                          // Save the new book in database
97                          bookDao.create(book);
98                      }
99                      // Create a new user book if needed
100                     UserBook userBook = userBookDao.getByBook(book.getId(), bookImportedEvent.getUser().getId());
101                     if (userBook == null) {
102                         userBook = new UserBook();
103                         userBook.setUserId(bookImportedEvent.getUser().getId());
104                         userBook.setBookId(book.getId());
105                         userBook.setCreateDate(new Date());
106                         if (!Strings.isNullOrEmpty(line[14])) {
107                             userBook.setReadDate(formatter.parseDateTime(line[14]).toDate());
108                         }
109                         if (!Strings.isNullOrEmpty(line[15])) {
110                             userBook.setCreateDate(formatter.parseDateTime(line[15]).toDate());
111                         }
112                         userBookDao.create(userBook);
113                     }
114                     // Create tags
115                     String[] bookshelfArray = line[16].split(",");
116                     Set<String> tagIdSet = new HashSet<String>();
117                     for (String bookshelf : bookshelfArray) {
118                         bookshelf = bookshelf.trim();
119                         if (Strings.isNullOrEmpty(bookshelf)) {
120                             continue;
121                         }
122                         Tag tag = tagDao.getByName(bookImportedEvent.getUser().getId(), bookshelf);
123                         if (tag == null) {
124                             tag = new Tag();
125                             tag.setName(bookshelf);
126                             tag.setColor(MathUtil.randomHexColor());
127                             tag.setUserId(bookImportedEvent.getUser().getId());
128                             tagDao.create(tag);
129                         }
130                         tagIdSet.add(tag.getId());
131                     }
132                     // Add tags to the user book
133                     if (tagIdSet.size() > 0) {
134                         List<TagDto> tagDtoList = tagDao.getByUserBookId(userBook.getId());
135                         for (TagDto tagDto : tagDtoList) {
136                             tagIdSet.add(tagDto.getId());
137                         }
138                         tagDao.updateTagList(userBook.getId(), tagIdSet);
139                     }
140                     TransactionUtil.commit();
141                 }
142             } catch (Exception e) {
143                 log.error("Error parsing CSV line", e);
144             }
145         }
```

This class is unnecessarily long, it’s doing more than what is expected as there is the opportunity to extract a service. This is a sign of insufficient modularization.

BookResource

BookResource.java

```java
public class BookResource extends BaseResource {
    /**
     * Creates a new book.
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(
            @FormParam("isbn") String isbn) throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Validate input data
        ValidationUtil.validateRequired(isbn, "isbn");

        // Fetch the book
        BookDao bookDao = new BookDao();
        Book book = bookDao.getByIsbn(isbn);
        if (book == null) {
            // Try to get the book from a public API
            try {
                book = AppContext.getInstance().getBookDataService().searchBook(isbn);
            } catch (Exception e) {
                throw new ClientException("BookNotFound", e.getCause().getMessage(), e);
            }

            // Save the new book in database
            bookDao.create(book);
        }

        // Create the user book if needed
        UserBookDao userBookDao = new UserBookDao();
        UserBook userBook = userBookDao.getByBook(book.getId(), principal.getId());
        if (userBook == null) {
            userBook = new UserBook();
            userBook.setUserId(principal.getId());
            userBook.setBookId(book.getId());
            userBook.setCreateDate(new Date());
            userBookDao.create(userBook);
        } else {
            throw new ClientException("BookAlreadyAdded", "Book already added");
        }

        JSONObject response = new JSONObject();
        response.put("id", userBook.getId());
        return Response.ok().entity(response).build();
    }
```

BookResource.java

```java
    /**
     * Add a book book manually.
     *
     * @param title Title
     * @param description Description
     * @return Response
     * @throws JSONException
     */
    @PUT
    @Path("manual")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(
            @FormParam("title") String title,
            @FormParam("subtitle") String subtitle,
            @FormParam("author") String author,
            @FormParam("description") String description,
            @FormParam("isbn10") String isbn10,
            @FormParam("isbn13") String isbn13,
            @FormParam("page_count") Long pageCount,
            @FormParam("language") String language,
            @FormParam("publish_date") String publishDateStr,
            @FormParam("tags") List<String> tagList) throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Validate input data
        title = ValidationUtil.validateLength(title, "title", 1, 255, false);
        subtitle = ValidationUtil.validateLength(subtitle, "subtitle", 1, 255, true);
        author = ValidationUtil.validateLength(author, "author", 1, 255, false);
        description = ValidationUtil.validateLength(description, "description", 1, 4000, true);
        isbn10 = ValidationUtil.validateLength(isbn10, "isbn10", 10, 10, true);
        isbn13 = ValidationUtil.validateLength(isbn13, "isbn13", 13, 13, true);
        language = ValidationUtil.validateLength(language, "language", 2, 2, true);
        Date publishDate = ValidationUtil.validateDate(publishDateStr, "publish_date", false);

        if (Strings.isNullOrEmpty(isbn10) && Strings.isNullOrEmpty(isbn13)) {
            throw new ClientException("ValidationError", "At least one ISBN number is mandatory");
        }

        // Check if this book is not already in database
        BookDao bookDao = new BookDao();
        Book bookIsbn10 = bookDao.getByIsbn(isbn10);
        Book bookIsbn13 = bookDao.getByIsbn(isbn13);
        if (bookIsbn10 != null || bookIsbn13 != null) {
            throw new ClientException("BookAlreadyAdded", "Book already added");
        }

        // Create the book
        Book book = new Book();
        book.setId(UUID.randomUUID().toString());

```

BookResource.java

```
202
203         if (title != null) {
204             book.setTitle(title);
205         }
206         if (subtitle != null) {
207             book.setSubtitle(subtitle);
208         }
209         if (author != null) {
210             book.setAuthor(author);
211         }
212         if (description != null) {
213             book.setDescription(description);
214         }
215         if (isbn10 != null) {
216             book.setIsbn10(isbn10);
217         }
218         if (isbn13 != null) {
219             book.setIsbn13(isbn13);
220         }
221         if (pageCount != null) {
222             book.setPageCount(pageCount);
223         }
224         if (language != null) {
225             book.setLanguage(language);
226         }
227         if (publishDate != null) {
228             book.setPublishDate(publishDate);
229         }
230
231         bookDao.create(book);
232
233         // Create the user book
234         UserBookDao userBookDao = new UserBookDao();
235         UserBook userBook = new UserBook();
236         userBook.setUserId(principal.getId());
237         userBook.setBookId(book.getId());
238         userBook.setCreateDate(new Date());
239         userBookDao.create(userBook);
240
241         // Update tags
242         if (tagList != null) {
243             TagDao tagDao = new TagDao();
244             Set<String> tagSet = new HashSet<>();
245             Set<String> tagIdSet = new HashSet<>();
246             List<Tag> tagDbList = tagDao.getByUserId(principal.getId());
247             for (Tag tagDb : tagDbList) {
248                 tagIdSet.add(tagDb.getId());
249             }
250             for (String tagId : tagList) {
251                 if (!tagIdSet.contains(tagId)) {
252                     throw new ClientException("TagNotFound", MessageFormat.format("Tag not found: {0}", tagId));
```

BookResource.java

```java
    public Response update(
            @PathParam("id") String userBookId,
            @FormParam("title") String title,
            @FormParam("subtitle") String subtitle,
            @FormParam("author") String author,
            @FormParam("description") String description,
            @FormParam("isbn10") String isbn10,
            @FormParam("isbn13") String isbn13,
            @FormParam("page_count") Long pageCount,
            @FormParam("language") String language,
            @FormParam("publish_date") String publishDateStr,
            @FormParam("tags") List<String> tagList) throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Validate input data
        title = ValidationUtil.validateLength(title, "title", 1, 255, true);
        subtitle = ValidationUtil.validateLength(subtitle, "subtitle", 1, 255, true);
        author = ValidationUtil.validateLength(author, "author", 1, 255, true);
        description = ValidationUtil.validateLength(description, "description", 1, 4000, true);
        isbn10 = ValidationUtil.validateLength(isbn10, "isbn10", 10, 10, true);
        isbn13 = ValidationUtil.validateLength(isbn13, "isbn13", 13, 13, true);
        language = ValidationUtil.validateLength(language, "language", 2, 2, true);
        Date publishDate = ValidationUtil.validateDate(publishDateStr, "publish_date", true);

        // Get the user book
        UserBookDao userBookDao = new UserBookDao();
        BookDao bookDao = new BookDao();
        UserBook userBook = userBookDao.getUserBook(userBookId, principal.getId());
        if (userBook == null) {
            throw new ClientException("BookNotFound", "Book not found with id " + userBookId);
        }

        // Get the book
        Book book = bookDao.getById(userBook.getBookId());

        // Check that new ISBN number are not already in database
        if (!Strings.isNullOrEmpty(isbn10) && book.getIsbn10() != null && !book.getIsbn10().equals(isbn10)) {
            Book bookIsbn10 = bookDao.getByIsbn(isbn10);
            if (bookIsbn10 != null) {
                throw new ClientException("BookAlreadyAdded", "Book already added");
            }
        }

        if (!Strings.isNullOrEmpty(isbn13) && book.getIsbn13() != null && !book.getIsbn13().equals(isbn13)) {
            Book bookIsbn13 = bookDao.getByIsbn(isbn13);
            if (bookIsbn13 != null) {
                throw new ClientException("BookAlreadyAdded", "Book already added");
            }
        }
```

BookResource.java

```java
        // Update the book
        if (title != null) {
            book.setTitle(title);
        }
        if (subtitle != null) {
            book.setSubtitle(subtitle);
        }
        if (author != null) {
            book.setAuthor(author);
        }
        if (description != null) {
            book.setDescription(description);
        }
        if (isbn10 != null) {
            book.setIsbn10(isbn10);
        }
        if (isbn13 != null) {
            book.setIsbn13(isbn13);
        }
        if (pageCount != null) {
            book.setPageCount(pageCount);
        }
        if (language != null) {
            book.setLanguage(language);
        }
        if (publishDate != null) {
            book.setPublishDate(publishDate);
        }

        // Update tags
        if (tagList != null) {
            TagDao tagDao = new TagDao();
            Set<String> tagSet = new HashSet<>();
            Set<String> tagIdSet = new HashSet<>();
            List<Tag> tagDbList = tagDao.getByUserId(principal.getId());
            for (Tag tagDb : tagDbList) {
                tagIdSet.add(tagDb.getId());
            }
            for (String tagId : tagList) {
                if (!tagIdSet.contains(tagId)) {
                    throw new ClientException("TagNotFound", MessageFormat.format("Tag not found: {0}", tagId));
                }
                tagSet.add(tagId);
            }
            tagDao.updateTagList(userBookId, tagSet);
        }

        // Returns the book ID
        JSONObject response = new JSONObject();
        response.put("id", userBookId);
        return Response.ok().entity(response).build();
```

DbOpenHelper

DbOpenHelper.java

```java
public abstract class DbOpenHelper {
    /**
     * Logger.
     */
    private static final Logger log = LoggerFactory.getLogger(DbOpenHelper.class);

    private final ConnectionHelper connectionHelper;

    private final SqlStatementLogger sqlStatementLogger;

    private final List<Exception> exceptions = new ArrayList<Exception>();

    private Formatter formatter;

    private boolean haltOnError;

    private Statement stmt;

    public DbOpenHelper(ServiceRegistry serviceRegistry) throws HibernateException {
        final JdbcServices jdbcServices = serviceRegistry.getService(JdbcServices.class);
        connectionHelper = new SuppliedConnectionProviderConnectionHelper(jdbcServices.getConnectionProvider());

        sqlStatementLogger = jdbcServices.getSqlStatementLogger();
        formatter = (sqlStatementLogger.isFormat() ? FormatStyle.DDL : FormatStyle.NONE).getFormatter();
    }

    public void open() {
        log.info("Opening database and executing incremental updates");

        Connection connection = null;
        Writer outputFileWriter = null;

        exceptions.clear();

        try {
            try {
                connectionHelper.prepare(true);
                connection = connectionHelper.getConnection();
            } catch (SQLException sqle) {
                exceptions.add(sqle);
                log.error("Unable to get database metadata", sqle);
                throw sqle;
            }

            // Check if database is already created
            Integer oldVersion = null;
            try {
                stmt = connection.createStatement();
                ResultSet result = stmt.executeQuery("select c.CFG_VALUE_C from T_CONFIG c where c.CFG_ID_C='DB_VERSION'");
                if (result.next()) {
                    String oldVersionStr = result.getString(1);
                    oldVersion = Integer.parseInt(oldVersionStr);
                }
            } catch (Exception e) {
                if (e.getMessage().contains("object not found")) {
                    log.info("Unable to get database version: Table T_CONFIG not found");
                } else {
                    log.error("Unable to get database version", e);
                }
            } finally {
                if (stmt != null) {
                    stmt.close();
                    stmt = null;
                }
            }

```

BookDataService

BookDataService.java

```java
*/
public class BookDataService extends AbstractIdleService {
    /**
     * Logger.
     */
    private static final Logger log = LoggerFactory.getLogger(BookDataService.class);

    /** Google Books API Search URL. */
    private static final String GOOGLE_BOOKS_SEARCH_FORMAT = "https://www.googleapis.com/books/v1/volumes?q=isbn:%s&key=%s";

    /** Open Library API URL. */
    private static final String OPEN_LIBRARY_FORMAT = "http://openlibrary.org/api/volumes/brief/isbn/%s.json";

    /** Executor for book API requests. */
    private ExecutorService executor;

    /** Google API rate limiter. */
    private RateLimiter googleRateLimiter = RateLimiter.create(20);

    /** Open Library API rate limiter. */
    private RateLimiter openLibraryRateLimiter = RateLimiter.create(0.33332);

    /** API key Google. */
    private String apiKeyGoogle = null;

    /** Parser for multiple date formats; */
    private static DateTimeFormatter formatter;

    static {

    @Override
    protected void startUp() throws Exception {
        initConfig();
        executor = Executors.newSingleThreadExecutor();
        if (log.isInfoEnabled()) {
            log.info("Book data service started");
        }
    }

    /** Initialize service configuration. */
    public void initConfig() {
        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                apiKeyGoogle = ConfigUtil.getConfigStringValue(ConfigType.API_KEY_GOOGLE);
            }
        });
    }

    /** Search a book by its ISBN. */
    public Book searchBook(String rawIsbn) throws Exception {
        // Sanitize ISBN (keep only digits)
        final String isbn = rawIsbn.replaceAll("[^\\d]", "");

        // Validate ISBN
        if (Strings.isNullOrEmpty(isbn)) {
            throw new Exception("ISBN is empty");
        }
        if (isbn.length() != 10 && isbn.length() != 13) {
            throw new Exception("ISBN must be 10 or 13 characters long");
        }

        Callable<Book> callable = new Callable<Book>() {

            @Override
            public Book call() throws Exception {
                try {
```

BookResource, DbOpenHelper, and BookDataService all have long methods which make the code less readable and difficult to understand whereas there could easily have been helper functions throughout. This is a sign of insufficient modularization.

- **Smell-4 Missing Hierarchy:**

ResourceUtil.java

```java
public class ResourceUtil {
    /**
     * List files inside a directory. The path can be a directory on the filesystem, or inside a JAR.
     */
    public static List<String> list(Class<?> clazz, String path, FilenameFilter filter) throws URISyntaxException, IOException {
        // Path is a directory on the filesystem
        URL dirUrl = clazz.getResource(path);
        if (dirUrl != null && dirUrl.getProtocol().equals("file")) {
            return Arrays.asList(new File(dirUrl.toURI()).list(filter));
        }
        // Path is a directory inside the same JAR as clazz
        if (dirUrl == null) {
            String className = clazz.getName().replace(".", "/") + ".class";
            dirUrl = clazz.getClassLoader().getResource(className);
        }
        if (dirUrl.getProtocol().equals("jar")) {
            if (path.startsWith("/")) {
                path = path.substring(1);
            }
            if (!path.endsWith("/")) { path = path + "/";
            }
            // Extract the JAR path
            String jarPath = dirUrl.getPath().substring(5, dirUrl.getPath().indexOf("!"));
            JarFile jar = new JarFile(URLDecoder.decode(jarPath, "UTF-8"));
            Set<String> fileSet = new HashSet<String>();
            try {
                Enumeration<JarEntry> entries = jar.entries();
                while (entries.hasMoreElements()) {
                    // Filter according to the path
                    String entryName = entries.nextElement().getName();
                    if (!entryName.startsWith(path)) {
                        continue;
                    }
                    String name = entryName.substring(path.length());
                    if (!"".equals(name)) {
                        // If it is a subdirectory, just return the directory name
                        int checkSubdir = name.indexOf("/");
                        if (checkSubdir >= 0) {
                            name = name.substring(0, checkSubdir);
                        }
                        if (filter == null || filter.accept(null, name)) {
                            fileSet.add(name);
                        }
                    }
                }
            } finally {  jar.close();}
            return Lists.newArrayList(fileSet);
        }
        throw new UnsupportedOperationException(MessageFormat.format("Cannot list files for URL {0}", dirUrl));
    }
    /**
     * List files inside a directory. The path can be a directory on the filesystem, or inside a JAR.
     */
    public static List<String> list(Class<?> clazz, String path) throws URISyntaxException, IOException {
        return list(clazz, path, null);
    }
```

Similar code blocks exist for handling file system and jar resources whereas there is an opportunity to make a common interface that can be implemented by concrete classes that deal with specific types of resources.

- **Smell-5 Speculative Generality:**

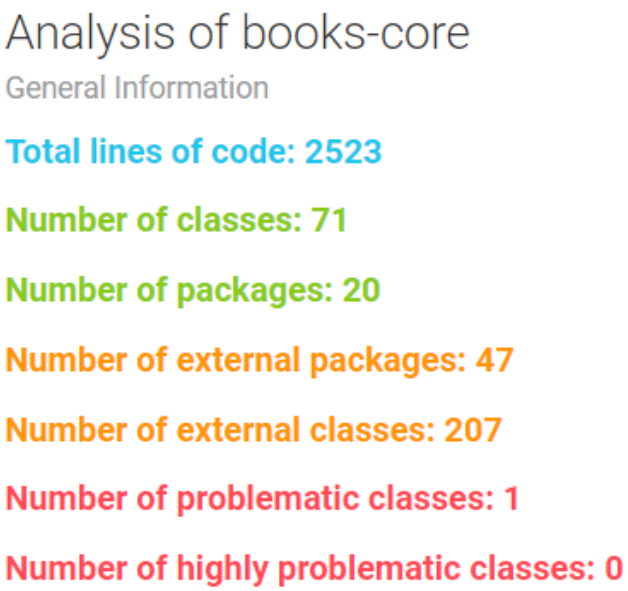
*FacebookService.java ×

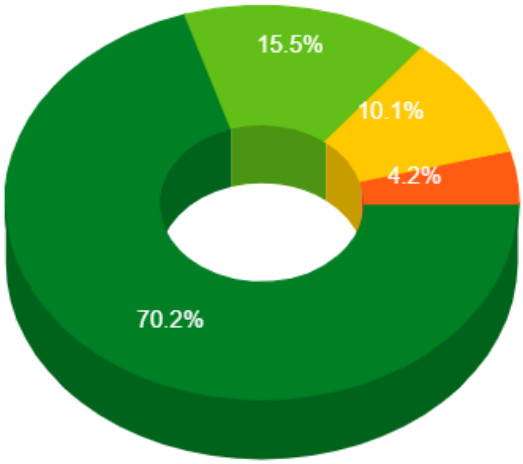
```java
    /**
     * Publish a book on Facebook.
     *
     * @param userBook User book to publish
     */
    public void publishAction(final UserBook userBook) {
//        FacebookClient facebookClient = new DefaultFacebookClient();

        // TODO Publish a user book
//        final String trackUrl = UrlUtil.getTrackUrl(track.getId());
//        String activityId = track.getActivity().getId();
//        String connection = "me/fitness.runs";
//        if (ActivityId.RUNNING.name().equals(activityId)) {
//            connection = "me/fitness.runs";
//        } else if (ActivityId.CYCLING.name().equals(activityId)) {
//            connection = "me/fitness.bikes";
//        } else if (ActivityId.WALKING.name().equals(activityId) || ActivityId.HIKING.name().equals(activityId)) {
//            connection = "me/fitness.walks";
//        }
//        try {
//            FacebookType publishResponse = facebookClient.publish(connection, FacebookType.class, Parameter.with("course", trackUrl));
//
//            if (log.isInfoEnabled()) {
//                log.info(MessageFormat.format("Published Facebook action: {0} for user {1}, item id = {2}", connection, user.getUsername(), publishResponse.getId()));
//            }
//        } catch (FacebookException e) {
//            log.error("Error publishing Facebook action", e);
//        }
    }
}
```

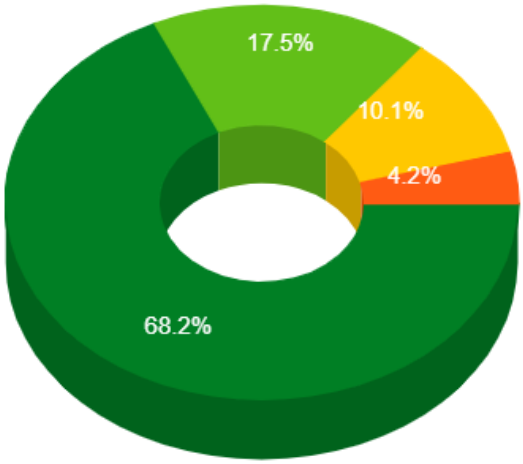
PublishAction function is dead code and seems to be a future feature that hasn’t been materialized yet. This exhibits the "Speculative Generality" smell. This refers to writing code for hypothetical, uncertain needs, leading to unnecessary complexity and potential maintenance burden.

Task-2(b) Code Metric Analysis

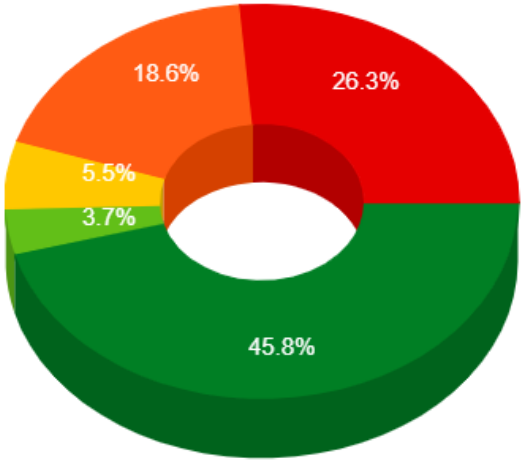




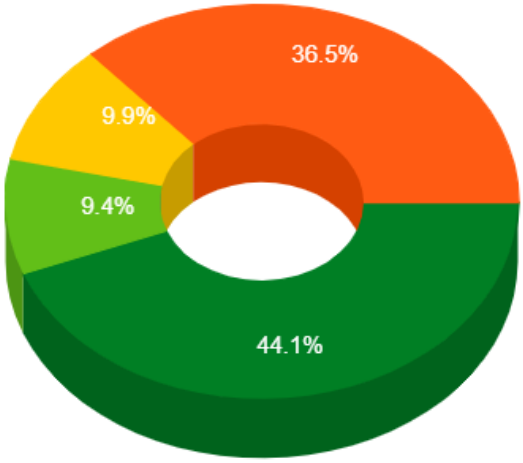
Coupling Between Object Classes



Coupling



Lack of Tight Class Cohesion



Lack of Cohesion of Methods

Note: We have only reported change in the above 4 metrics as these metrics had major issues. Most of the other metrics already had either low or low-medium issues.

Task-3 Refactoring

Task-3(a) Rectifies Design Smells

- **1. Deficient Encapsulation:**

  We have refactored all the util classes by providing a private constructor which ensures that these utility classes cannot be instantiated.

  Made changes in the below files

  e.g. Constants.java, Config.Util, DirectoryUtil.java, EntityManagerUtil.java, StreamUtil.java, TransactionUtil.java, PaginatedLists.java, QueryUtil.java, MathUtil.java, MimeType.java, MimeTypeUtil.java,EnvironmentUtil.java, HttpUtil.java, LocaleUtil.java,ResourceUtil.java

TransactionUtil.java

```java
package com.sismics.books.core.util;

import com.sismics.util.context.ThreadLocalContext;

/**
 * Database transaction utils.
 *
 * @author jtremeaux
 */
public class TransactionUtil {


    private TransactionUtil() {

    }
```

- **2. Unnecessary Hierarchy:**

  We have removed unnecessary classes (PermissionException and AuthenticationException) and created a common FacebookServiceException.

FacebookServiceException.java

```java
package com.sismics.books.core.service.facebook;

/**
 * Exception raised on a Facebook authentication error.
 *
 * @author jtremeaux
 */
public class FacebookServiceException extends Exception {

    /**
     * Serial version UID.
     */
    private static final long serialVersionUID = 1L;

    /**
     * Constructor of FacebookServiceException.
     *
     * @param message Message
     */
    public FacebookServiceException(String message) {
        super(message);
    }
}
```

- **3. Insufficient Modularization**

  A new function- connectionSetUp() is introduced which is used in both searchBookWithGoogle() and searchBookWithOpenLibrary()

BookDataService.java

```java
    /**
     * helper function for fetching data using isbn
     */
    private JsonNode connectionSetUp(String isbn) throws Exception {
        googleRateLimiter.acquire();

        URL url = new URL(String.format(Locale.ENGLISH, GOOGLE_BOOKS_SEARCH_FORMAT, isbn, apiKeyGoogle));
        URLConnection connection = url.openConnection();
        connection.setRequestProperty("Accept-Charset", "utf-8");
        connection.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gec
        connection.setConnectTimeout(10000);
        connection.setReadTimeout(10000);
        InputStream inputStream = connection.getInputStream();
        ObjectMapper mapper = new ObjectMapper();
        JsonNode rootNode = mapper.readValue(inputStream, JsonNode.class);

        return rootNode;
    }
```

A new class- BookImportService is added

BookImportService.java

```java
package com.sismics.books.core.service;

import java.io.File;

public class BookImportService extends AbstractIdleService{

    /**
     * Logger.
     */
    private static final Logger log = LoggerFactory.getLogger(BookImportService.class);

    private BookDao bookDao = new BookDao();
    private UserBookDao userBookDao = new UserBookDao();
    private TagDao tagDao = new TagDao();

    @Override
    protected void startUp() throws Exception {
        if (log.isInfoEnabled()) {
            log.info("Book import service started");
        }
    }
```

DbOpenHelper

DbOpenHelper.java

```java
package com.sismics.util.jpa;

import com.google.common.base.Strings;

/**
 * A helper to update the database incrementally.
 *
 * @author jtremeaux
 */

public abstract class DbOpenHelper {
    /**
     * Logger.
     */
    private static final Logger log = LoggerFactory.getLogger(DbOpenHelper.class);

    private final ConnectionHelper connectionHelper;

    private final SqlStatementLogger sqlStatementLogger;

    private final List<Exception> exceptions = new ArrayList<Exception>();

    private Formatter formatter;

```

BookResource

BookResource.java

```java
package com.sismics.books.rest.resource;

import java.io.File;

/**
 * Book REST resources.
 *
 * @author bgamard
 */
@Path("/book")
public class BookResource extends BaseResource {
    private void authenticateOrThrowForbidden() throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }
    }
    private Book findOrCreateBook(String isbn) throws JSONException {
        BookDao bookDao = new BookDao();
        Book book = bookDao.getByIsbn(isbn);

        if (book == null) {
            try {
                book = AppContext.getInstance().getBookDataService().searchBook(isbn);
            } catch (Exception e) {
                throw new ClientException("BookNotFound", e.getCause().getMessage(), e);
```

- **4. Missing Hierarchy:**

  We have created a common interface (ResourceHandler to deal with listing the files in a resource). This interface is implemented by **FileSystemResourceHandler** and **JarResourceHandler.** This makes are system more decoupled so that if there is any need of handling another type of resource. It can be achieved by just implementing ResourceHandler interface.

FileSystemResourceHandler.java

```java
package com.sismics.util.resource;

import java.io.File;

public class FileSystemResourceHandler implements ResourceHandler {
        private File directory;

        public FileSystemResourceHandler(URL dirUrl) throws URISyntaxException {
            this.directory = new File(dirUrl.toURI());
        }

        @Override
        public List<String> listFiles(FilenameFilter filter) throws IOException {
            return Arrays.asList(this.directory.list(filter));
        }
    }
```

JarResourceHandler.java

```java
package com.sismics.util.resource;

import java.io.FilenameFilter;

public class JarResourceHandler implements ResourceHandler {
    private JarFile jarFile;
    private String path;

    public JarResourceHandler(URL jarUrl, String path) throws URISyntaxException, IOException {
        // Extract the JAR path
        String jarPath = jarUrl.getPath().substring(5, jarUrl.getPath().indexOf("!"));
        this.jarFile = new JarFile(URLDecoder.decode(jarPath, "UTF-8"));
        this.path = path;
    }

    @Override
    public List<String> listFiles(FilenameFilter filter) throws IOException {
        Set<String> fileSet = new HashSet<>();
```

ResourceHandler.java

```java
package com.sismics.util.resource;

import java.io.FilenameFilter;

public interface ResourceHandler {
    List<String> listFiles(FilenameFilter filter) throws IOException;
}
```

- **5. Speculative Generality:**

Removed *publishAction(),* It had a Dead code.

FacebookService.java

```java

    /**
     * Updates user's personal informations.
     *
     * @param accessToken Access token
     * @param userApp User app (updated by side effect)
     */
    public void updateUserData(String accessToken, UserApp userApp) {
        FacebookClient facebookClient = new DefaultFacebookClient(accessToken);
        User user = facebookClient.fetchObject("me", User.class);
        userApp.setExternalId(user.getId());
        userApp.setEmail(user.getEmail());
        userApp.setUsername(user.getUsername());
    }
}
```

Task-3(b) Code Metrics Analysis

Analysis of books-core

General Information

Total lines of code: 2493

Number of classes: 73

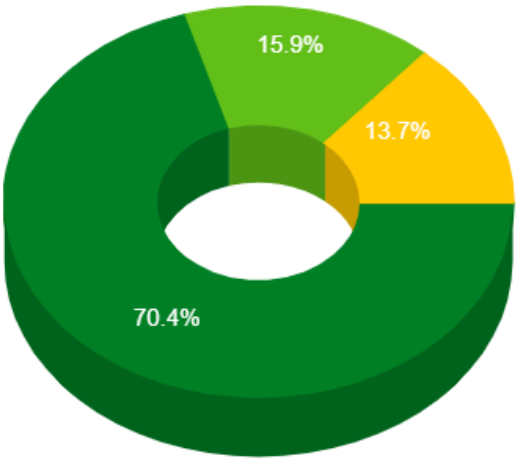
Number of packages: 21

Number of external packages: 43

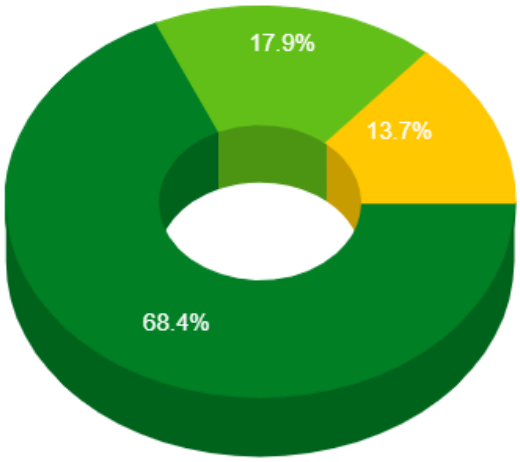
Number of external classes: 193

Number of problematic classes: 1

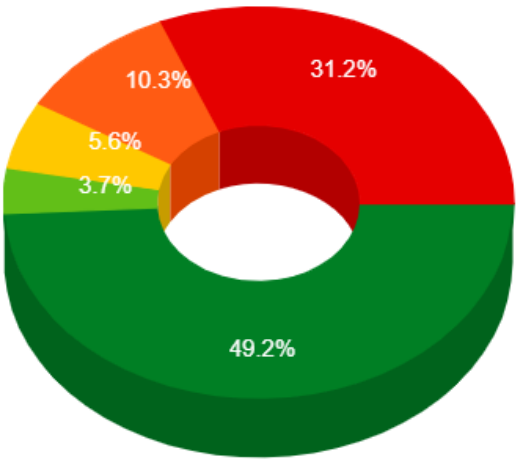
Number of highly problematic classes: 0



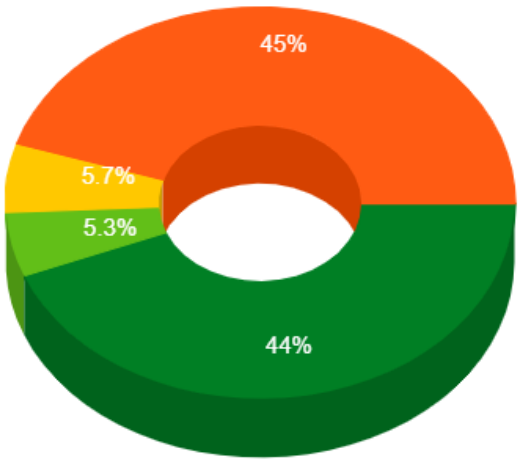
Coupling Between Object Classes



Coupling



Lack of Tight Class Cohesion



Lack of Cohesion of Methods

Note: We have only reported change in the above 4 metrics as these metrics had major issues. Most of the other metrics already had either low or low-medium issues.

Task-3(c) Leveraging Large Language Models for refactoring

Link - https://chat.openai.com/share/7da3c95d-211c-4779-83b6-096fdc715767

Observations-

- Refactoring suggested by the LLM is often incomplete and has comments indicating the implementation of missing code.
- In the case of BookImportAsyncListner, the LLM suggests a very minimal refactoring whereas we create a separate book import service (BookImportService) which does the database operations using the DAO object.
- In the case of ResourceUtil, the LLM suggests separate functions for handling the respective resources. We handle this issue differently, by creating a common interface (ResourceHandler) to deal with listing the files in a resource. This interface is implemented by the classes FileSystemResourceHandler and JarResourceHandler.

Contribution of team members :

1. Gunank Singh Jakhar -

- Setting up the codebase and debugging environment (in Eclipse) to facilitate the understanding of the specific subsystems.
- Understanding **Book Addition, Display Subsystem, and Bookshelf Management Subsystem**, and creating their corresponding UML class diagrams.
- Identifying Insufficient Modularization design smell in the **BookImportAsyncListener** class and refactoring it.
- Identifying Missing Hierarchy design smell in the **ResourceUtil** class and refactoring it.
- Using **CodeMR** to identify relevant Metrics.

2. M Elamparithy -

- Understanding **Book Addition, Display Subsystem, and Bookshelf Management Subsystem**, and creating their corresponding UML class diagrams.
- Identifying Insufficient Modularization design smell in the **BookDataService** class and refactoring it.
- Identifying Unnecessary Hierarchy design smell in **AuthenticationException** and **PermissionException** and refactoring it.
- Using **CodeMR** to identify relevant Metrics.

3. Rishabh Gupta -

- Understanding **User Management Subsystems** and creating their corresponding UML class diagrams.
- Identifying Deficient Encapsulation Design smell in all the **utility classes** and refactoring it.
- Identifying Speculative Generality Design smell and refactoring it.

4. Piyush Singh -

- Understanding **User Management Subsystems** and creating their corresponding UML class diagrams.
- Identifying Insufficient Modularization design smell in the **DbOpenHelper** class and refactoring it.
- **Documentation** of the processes involved in Refactoring and relevant matrices.

5. Prashant Kumar -

- Understanding **Book Addition, Display Subsystem, and Bookshelf Management Subsystem**, and creating their corresponding UML class diagrams.
- Identifying Insufficient Modularization design smell in **BookResource** class and refactoring it.
- **Documentation** of the processes involved in Refactoring and relevant matrices.