Sismics Books
Project-1
Team_11

Strategy Employed for Project Completion

+ Distribute tasks following team discussion.
+ Collaboratively install required project tools.
+ Identify classes with team coordination.
+ Draw UML diagrams for each Subsystem.
+ Detect design smells using SonarQube.
+ Analyze existing code with Code Metrics.
+ Refactor code individually for design smells.
+ Create GitHub issues, assign within team.
+ Review and merge code into master.
+ Reanalyze code using CodeMR.
+ Compare manual and LLM refactoring.
+ Simultaneously completed bonus tasks.

Distribute Tasks Following Team Ddiscussion

Task No	Task Name	Task Description	Member Name	%
Task 1	1. Book Addition & Display Subsystem	For each subsystem separately, Identify relevant Classes, Document Functionality and Behavior, Create UML Diagrams, and Provide Observations and Comments, Along with any applicable Assumptions.	**Vilal Ali** **Madan NS**	**50** **50**
	2. Bookshelf Management Subsystem		**Vilal Ali** **Madan NS**	**60** **40**
	3. User Management Subsystem		**Vilal Ali** **Madan NS**	**60** **40**
Task 2	2a. Detect 5-7 design smells	Sonarqube as the primary tool to identify 5-7 design smells in code, focusing on structures and patterns indicative of violations to fundamental design principles, recognizing the potential for recurring issues in development.	**Vilal Ali** **Shriom Tyagi**	**60** **40**
	2b. Code Metrics Analysis	We have used popular tools like **CodeMR** to compute the code quality metrics on the Books project. **CodeMR** along with the code metric, generates comprehensive visualizations on it.	**Vilal Ali**	**100**
Task 3	3a. Refactor 5-7 design smells	In Task 2a, we identified and analyzed 7+ design smells within the existing codebase. Now, the objective is to rectify these issues through code refactoring without fundamentally altering the structure of the code.	**Vilal Ali** **Shriom Tyagi** **Hanuma** **Madan NS**	**35** **30** **20** **15**
	3b. Code Metrics: After Refactoring	Post-refactoring, analyze code metrics using CodeMR, which not only provides metrics but also generates comprehensive visualizations for a more thorough understanding of the codebase.	**Vilal Ali**	**100**
	3c. Leveraging LLM for Refactoring	Leverage advanced language models like GPT-3.5 or bard for refactoring code snippets identified with design smells. After manually refactoring the snippets, use the language models to generate alternative versions, comparing clarity, conciseness, and adherence to best practices	**Vilal Ali** **Hanuma**	**60** **40**
Bonus	Automated Refactoring Pipeline	design and implementation of an automated pipeline for detecting design smells in a GitHub repository, refactoring the identified issues, and generating pull requests for the changes.	**Hanuma**	**100**

Collaboratively Install Required Project Tools

Tools Installed for Project Development

+ Installed Java Version.
+ Set up Sonarqube for analysis.
+ Integrated CodeMR for code metrics.
+ Configured IntelliJ for development.

Challenges Encountered During Project Setup and Execution

+ New to Java projects before this endeavor.
+ Project utilizes Java 11; Sonarqube and CodeMR require the Java 17.
+ Switched to Java 17; Sonarqube ran but project malfunctioned. Reverting to Java 11, Sonar and CodeMetrics failed, causing frustration.
+ Implemented SDK for Java Version Control.
+ While using LLM for code refactoring, encountered the issue of fixing one problem only to cause another. After numerous trials and learning from the experiences, successfully resolved the challenges.

Initial impressions upon reviewing the code repository.

Identify Classes With Team Coordination

Identified Classes In Book Addition & Display Subsystem

+ AppContext, AppResource, BaseResource, Book, BookDao, BookDataService, BookImportAsyncListener, BookImportedEvent, BookResource, TagDao, TagDto, TagResource, UserBook, UserBookDao, UserBookDto, UserBookTag, AnonymousPrincipal

Identified Classes In Bookshelf Management Subsystem

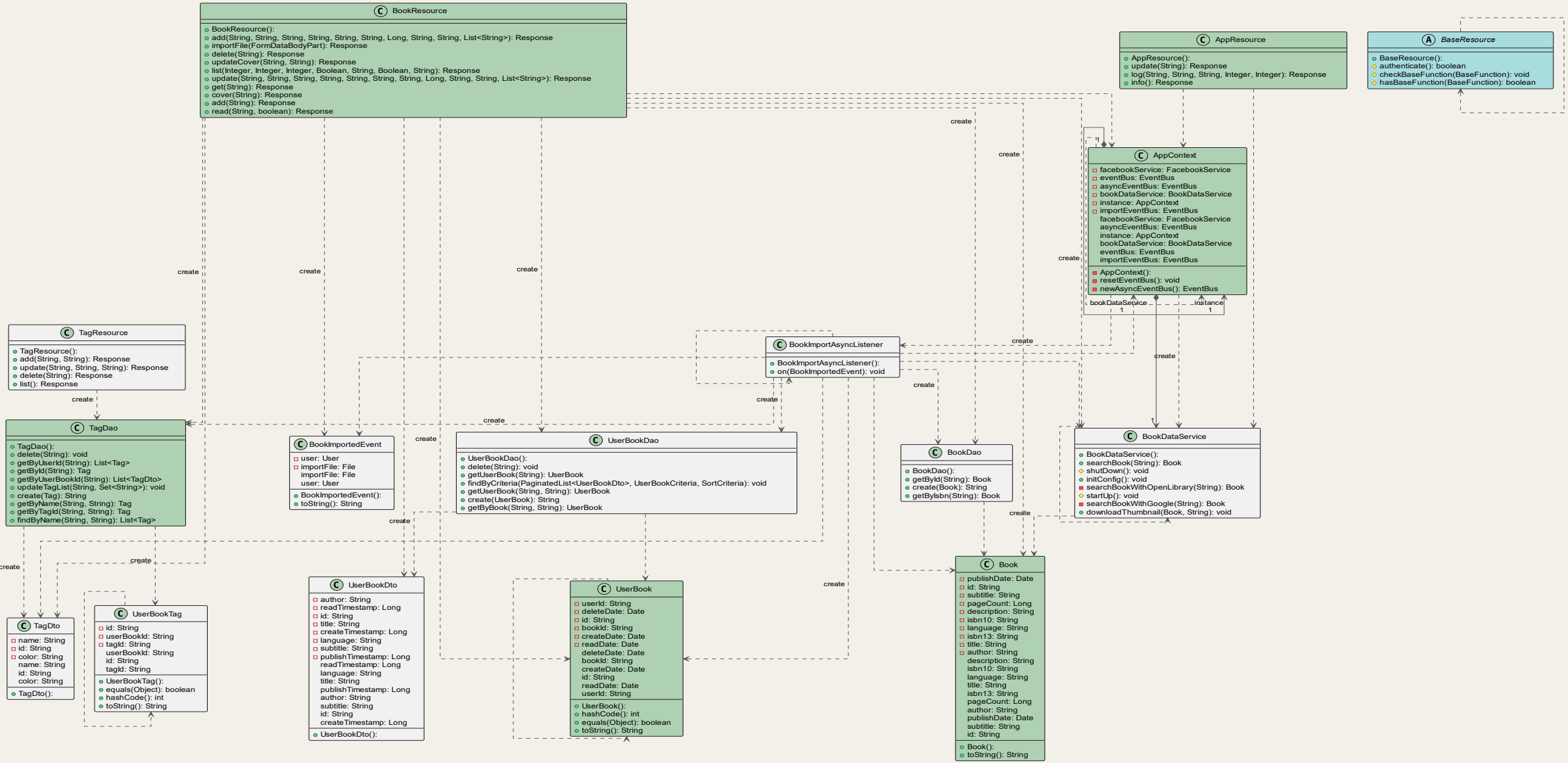
+ AppContext, AppResource, BaseResource, Book, BookDao, BookDataService, BookImportAsyncListener, BookImportedEvent, BookResource, SortCriteria, UserBookDao, UserBookCriteria.

Identified Classes In User Management Subsystem

+ AppContext, AppResource, BaseResource, ConnectResource, FacebookService, QueryParam, AuthenticationToken and AuthenticationTokenDao, User, UserResource, UserApp, UserAppCreatedEvent, UserAppCreatedAsyncListener, UserContact, UserDao, UserDto , UserAppDao, UserAppDto ,UserContactDao, UserContactDto, UserContactCriteria

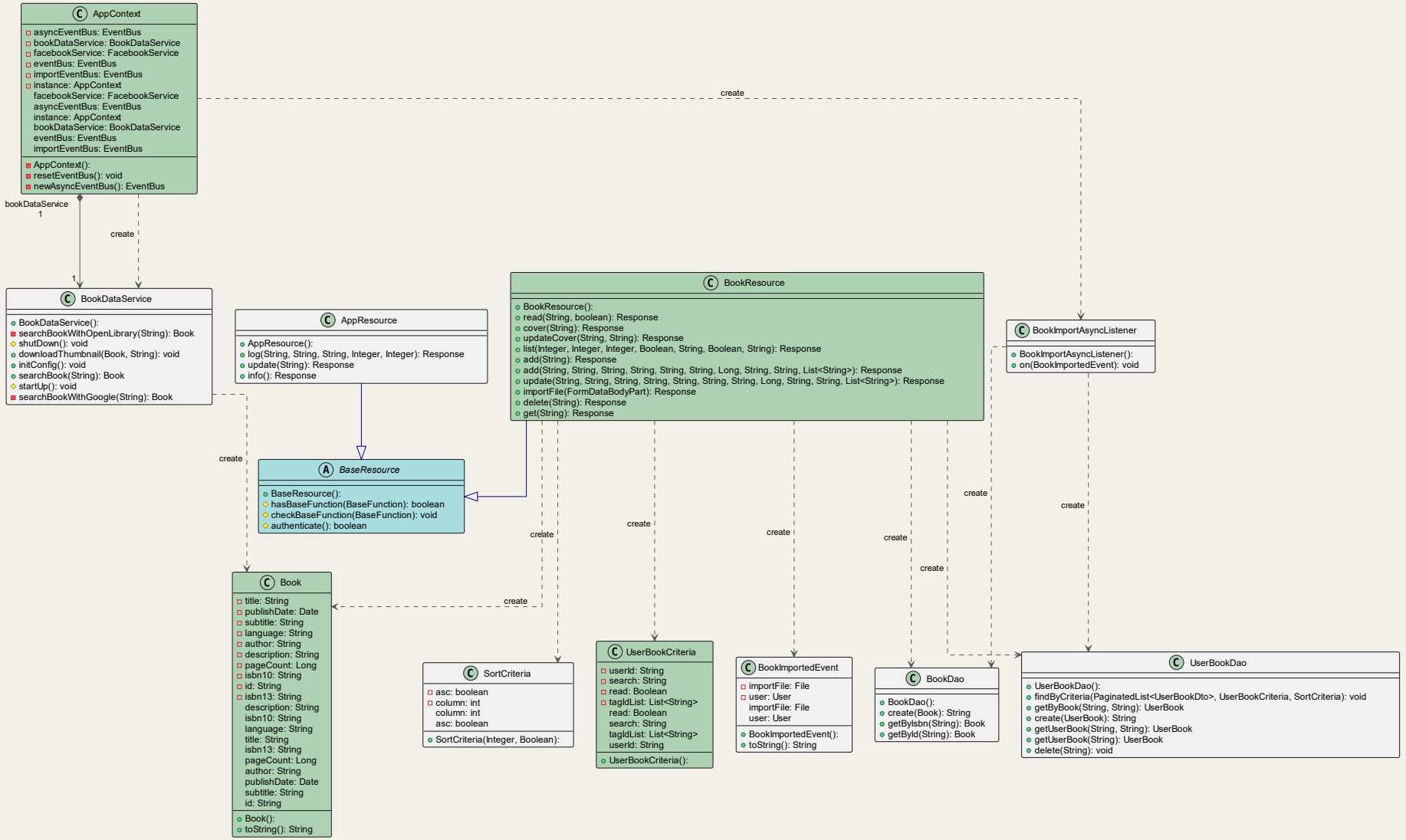
UML Diagrams for Each Subsystem

Book Addition Display Subsystem



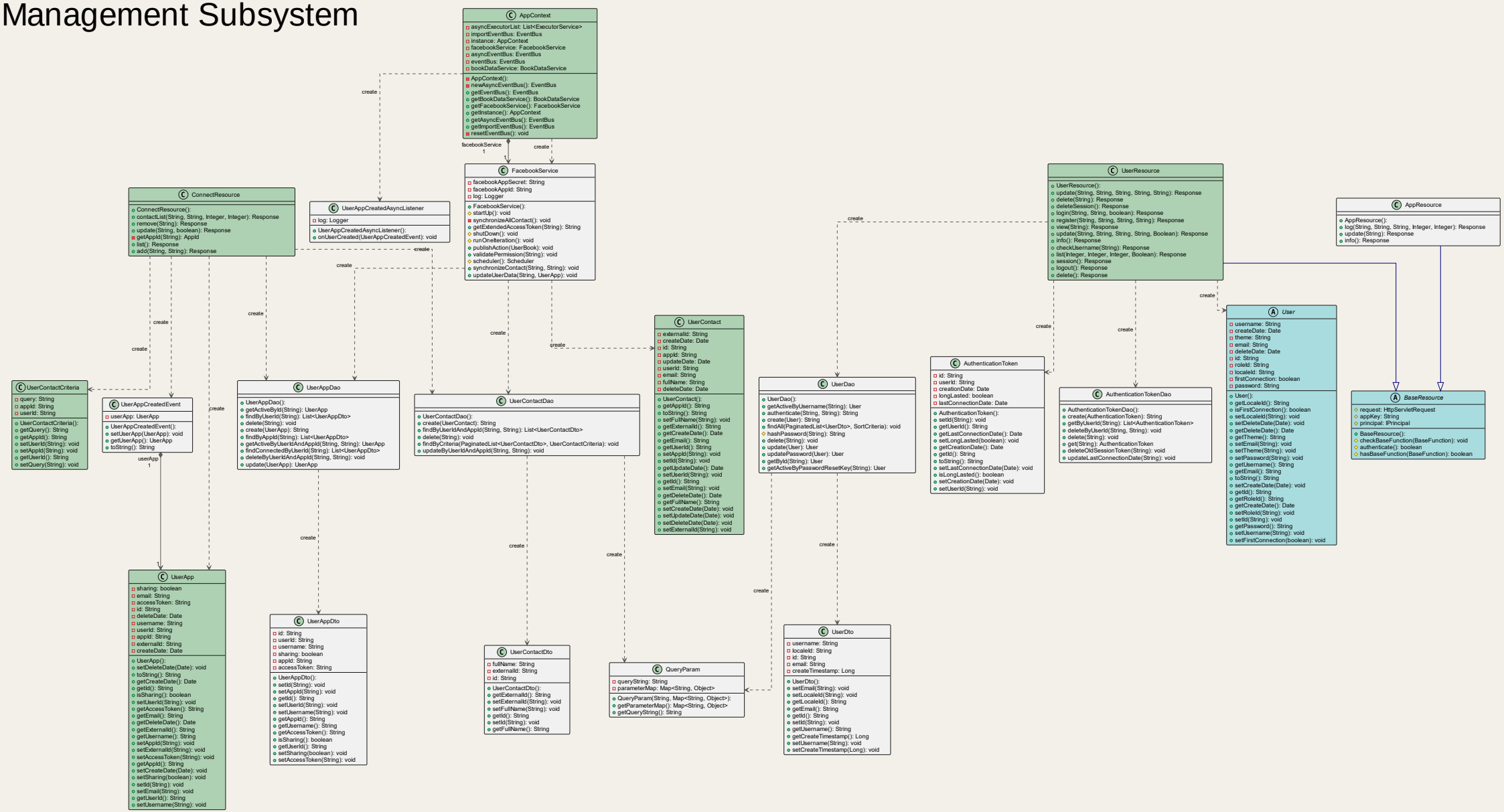
UML Diagrams for Each Subsystem ...

Bookshelf Management Subsystem



UML Diagrams for Each Subsystem...

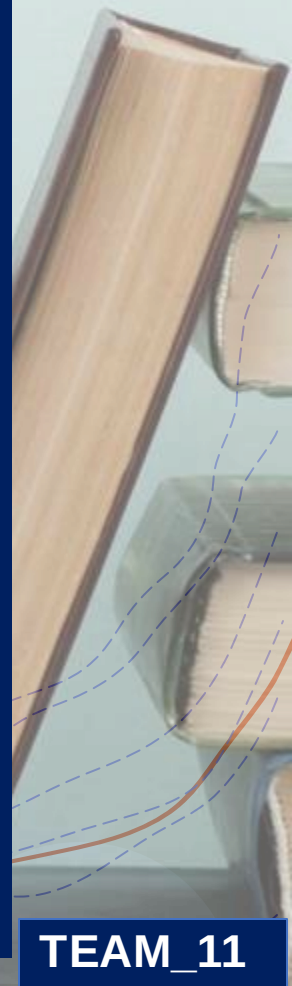
User Management Subsystem



Detect Design Smells help of SonarQube

Where is the design smell?

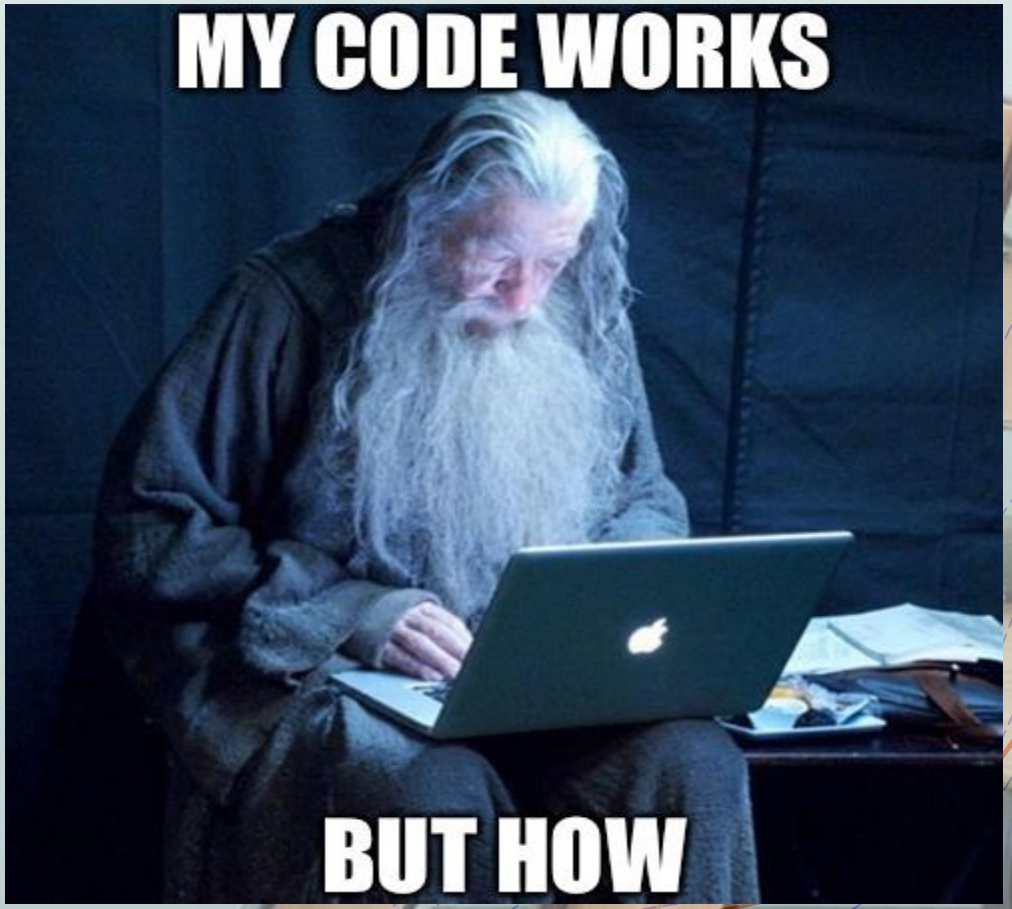
+ **books-web/src/main/java/com/sismics/books/rest/resource/BookResource.java**
  - **Large Class, Long Method, Lack of Dependency Injection**
  - Solution: Based on the identified design smells, We will refactor the code by breaking down long methods, introducing dependency injection, and replacing magic strings with constants. Here's the refactored version of the BookResource.java
+ **books-core/src/main/java/com/sismics/books/core/constant/Constants.java**
  - Lack of Encapsulation, Global State
  - Solution: we can refactor the Constants.java class to use a more object-oriented approach. One way to do this is to create separate classes for each group of related constants, encapsulating them within those classes.
+ **books-core/src/main/java/com/sismics/books/core/listener/async/BookImportAsyncListener.java**
  - Large Class, Long Method, Lack of Dependency Injection, Switch Statements, Duplication
  - Solution: We can refactor the code by extracting methods, and simplifying the logic.
+ **books-core/src/main/java/com/sismics/util/ResourceUtil.java**
  - Feature Envy, Long Method, Data Clumps, Switch Statements
+ **books-core/src/main/java/com/sismics/util/HttpUtil.java**
  - Resource Management, Many resources in Java need be closed after they have been used.
  - Solution: Try-with-Resources: The BufferedReader is declared within the parentheses of the try statement
+ **books-web/src/main/java/com/sismics/books/rest/resource/BaseResource.java**
  - Redundant Null Check, Unthrown Exception in Method Signature.
  - Solution: Redundant Check Elimination: The superfluous null checks preceding the instanceof operator have been eliminated from the authenticate() method.
+ **books-web/src/main/java/com/sismics/books/rest/resource/BaseResource.java**
  - Large Class, Feature Envy, Lack of Encapsulation.
  - Solution: We can extract the authentication logic into a separate AuthenticationService class to adhere to SRP.



TEAM_11

Refactor Code Individually for Design Smells

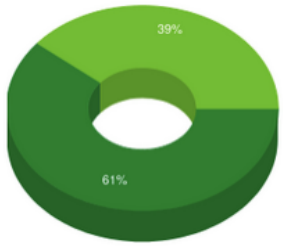
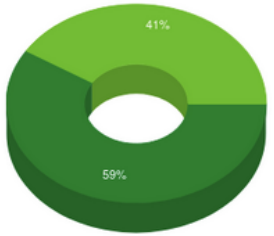
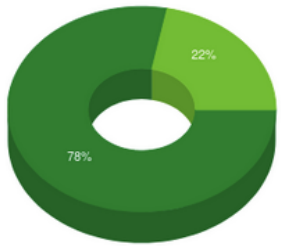
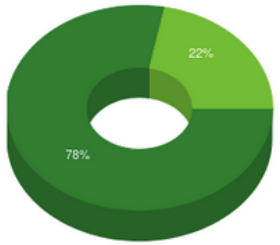
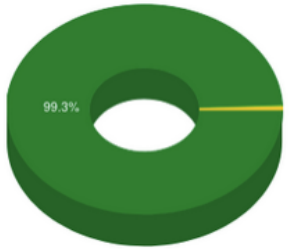
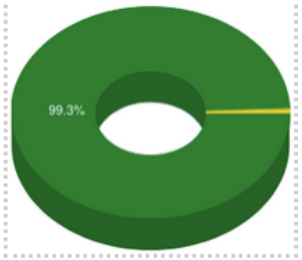
For further details, kindly refer to the project report by clicking here

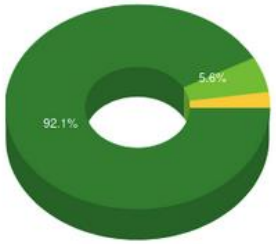
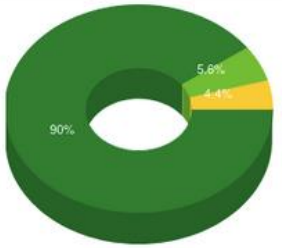
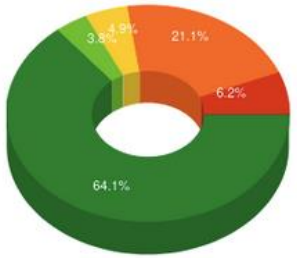
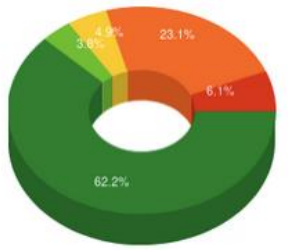
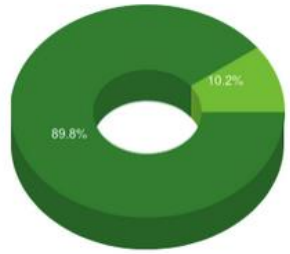
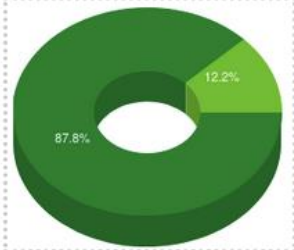


List of Created issues and PR on GitHub

S. No.	Design Smell	Task Description	#Issue	Assignee	PR	Reviewed
01	Large Classes or Large Methods	Refactor 'Brain Method' to reduce LOC, complexity, nesting, and variables.	#15	Vilal	#26	Shriom
02	Encapsulation	Add a private constructor to hide the implicit public one.	#10	Vilal	#18	Shriom
03	Resource Leak	Changing Try to try-with-resources	#12	Madan	#26	Vilal
04	Modularity, Encapsulation	Encapsulates the creation of the EntityManager and handles exceptions., Removed unnecessary checks since they were already performed earlier.	#16	Hanuma	#31	Vilal
05	Large Classes or Large Methods	Refactor this method to reduce its Cognitive Complexity from 52 to the 15 allowed.	#14	Shriom	#19	Vilal
06	Large Classes or Large Methods	Refactor the code to bring down its Cognitive Complexity from 25 to the permissible limit of 15.	#24	Shriom	#27	Vilal
07	Encapsulation	The visibility of private constructor change to protected.	#20	Shriom	#19	Vilal

Conduct Code Metrics Analysis using CodeMR
Reevaluate the Code Post-Refactoring using CodeMR

S. N.	Metrics	Before Refactoring: Code Metrics	After Refactoring: Code Metrics
01	Cyclomatic Complexity Very High High Medium-high Low-medium Low		
02	Depth of Inheritance Tree Very High High Medium-high Low-medium Low		
03	Number of Children of a Class Very High High Medium-high Low-medium Low		

S. N.	Metrics	Before Refactoring: Code Metrics	After Refactoring: Code Metrics
04	Coupling Between Object Classes Very High High Medium-high Low-medium Low		
05	Lack of Cohesion on Methods Very High High Medium-high Low-medium Low		
06	Response for a Class Very High High Medium-high Low-medium Low		

Conduct Code Metrics Analysis using CodeMR, For further details, by clicking here.

Reevaluate the Code Post-Refactoring using CodeMR, For further details, by clicking here.

TEAM_11

Compare Manual and LLM Refactoring

- **Where is the design smell?**
- books-core/src/main/java/com/sismics/books/core/model/context/AppContext.java
- Link to the ChatGPT chat
- **Original Smells detected by us:** Singleton Pattern Misuse, God Object, Static Method, Hardcoded Values, Mutable Static State. Please refer to section 2.a for more details.
- **Smells detected by ChatGPT:**
- **Singleton Pattern**: While the singleton pattern ensures there's only one instance of **AppContext**, it also introduces tight coupling and makes it hard to test.
- **Resource Management**: The code directly creates and starts services (**BookDataService**, **FacebookService**) within the constructor, which tightly couples the initialization of services with the **AppContext**.
- **Complex Initialization Logic**: The **resetEventBus()** method has complex logic for initializing event buses and executors, making it hard to understand and maintain.
- **Changes in the code. Red refers to the code generated by ChatGPT, Green is the refactored code by us. Please see the changes below**

For further details, by clicking here.

- **Removed Singleton Instance**

```
26    private BookDataService bookDataService;
27    private FacebookService facebookService;
28
29    private AppContext() {
30        initializeServices();
31        initializeEventBuses();
32    }
33
34    public static AppContext getInstance() {
35        if (instance == null) {
36            instance = new AppContext();
37        }
38        return instance;
39    }
40
41    private void initializeServices() {


42        bookDataService = new BookDataService();
43        bookDataService.startAndWait();
```

```
48    private BookDataService bookDataService;
49
50    /**
51     * Facebook interaction service.
52     */
53    private FacebookService facebookService;
54
55    /**
56     * Asynchronous executors.
57     */
58    private List<ExecutorService> asyncExecutorList;
59
60    /**
61     * Private constructor.
62     */
63    private AppContext() {
64        resetEventBus();
65
66        bookDataService = new BookDataService();
67        bookDataService.startAndWait();
```

- **Separated Service Initialization**

```
77    public EventBus getImportEventBus() {
78        return importEventBus;
79    }
80
81    public BookDataService getBookDataService() {




82        return bookDataService;
83    }
84




85    public FacebookService getFacebookService() {
86        return facebookService;
87    }
88 }
```

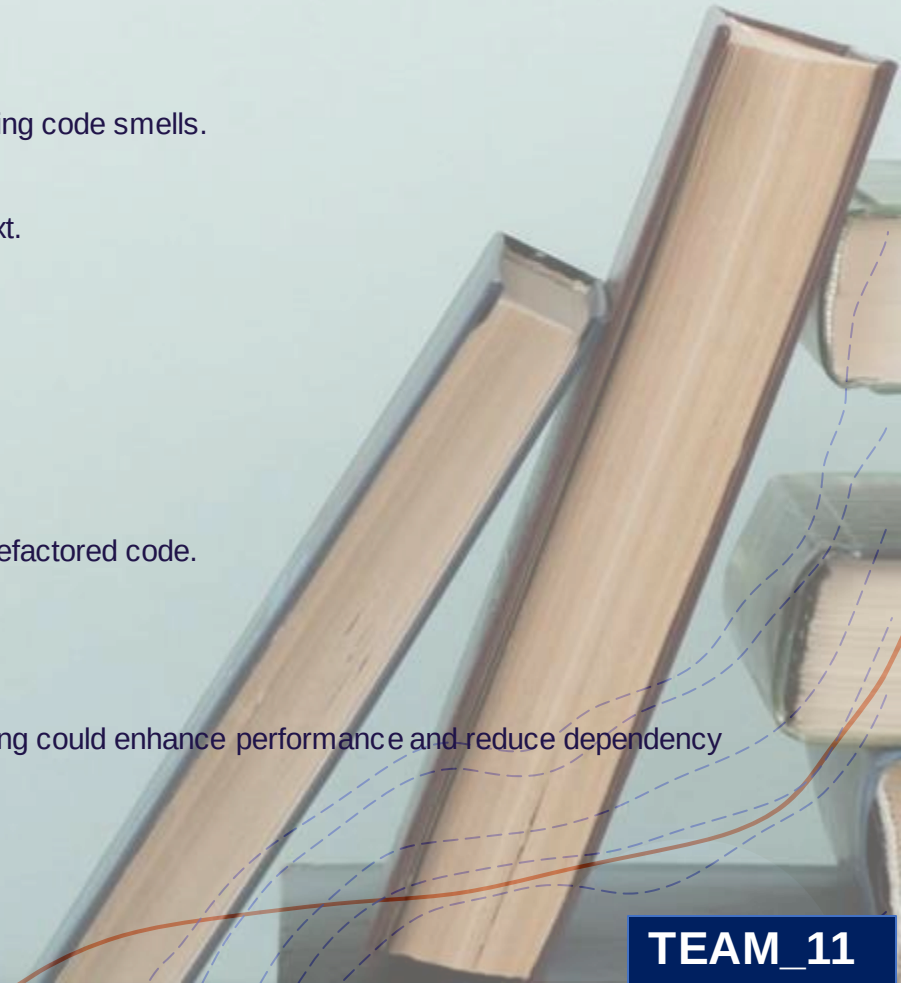
```
140     */
141    public EventBus getImportEventBus() {
142        return importEventBus;
143    }
144
145    /**
146     * Getter of bookDataService.
147     *
148     * @return bookDataService
149     */
150    public BookDataService getBookDataService() {
151        return bookDataService;
152    }
153
154    /**
155     * Getter of facebookService.
156     *
157     * @return facebookService
158     */
159    public FacebookService getFacebookService() {
160        return facebookService;
161    }
162 }
```

TEAM_11

Bonus Tasks

Findings

- Performance with Code Context Length
  - Longer code contexts resulted in inconsistency and reduced effectiveness in identifying and refactoring code smells.
  - In scenarios with smaller code contexts, ChatGPT exhibited better performance.
  - Continuous prompting after each interaction proved to be a viable workaround for maintaining context.
- Prompt Dependency
  - The prompt's nature significantly impacted the output generated by ChatGPT.
  - Prompt-driven interactions were crucial for steering ChatGPT towards the desired outcomes.
- Manual Intervention Requirement
  - While ChatGPT showed promise, manual intervention was often necessary to ensure the quality of refactored code.
  - Smaller codebases required less manual intervention and yielded excellent results.
- Performance Enhancement through Fine-Tuning
  - Focusing on training ChatGPT-like models specifically tailored for code smell detection and refactoring could enhance performance and reduce dependency on manual intervention.



Bonus Tasks...

1. **Introduction:** This report documents the design and implementation of an automated pipeline for detecting design smells in a GitHub repository, refactoring the identified issues, and generating pull requests for the changes.

2. **Pipeline Overview:**
   - The pipeline consists of three main components:
     - Automated Design Smell Detection
     - Automated Refactoring
     - Pull Request Generation

3. **Code Snippets Explanation:**
   - **Refactoring Code with ChatGPT**
     - This function (refactor code with chatgpt) takes the path to a Java file in a GitHub repository and refactors the code using the OpenAI ChatGPT model. The refactored code is then written to a new file in the repository.
   - **Creating Branch**
     - Function (create branch) creates a new branch in the local repository to push the refactored code changes.
   - **Main Function**
     - The main function (main) orchestrates the entire pipeline by cloning the repository, refactoring the code, committing and pushing the changes to a new branch, and creating a pull request.

For further details, by clicking here.

How it Works !!

1. Set-up: API

```
# Set up OpenAI API
API_KEY = 'sk-YiI7WHB66e[redacted]2uBQNlNimvK'
# Your GitHub Personal Access Token
GITHUB_TOKEN = 'ghp_3ZiYc7[redacted]MehT4aTH4W'
# Your GitHub Username
GITHUB_USERNAME = 'bhaskarahanuma'
# Repository owner and name
REPO_OWNER = 'serc-courses'
REPO_NAME = 'se-project-1--_11'
# Branch where the refactored code will be pushed
NEW_BRANCH = 'refactor-branch'
```

2. Code Refactoring with OpenAI's ChatGPT

- The **refactor_code_with_chatgpt** function is defined to refactor Java code using OpenAI's ChatGPT.
- It reads the code from a specified file, constructs a prompt for analysis and refactoring, sends the code to OpenAI's API, and retrieves the refactored code from the response.
- Prompt:

```
# Construct prompt for code analysis and refactoring
prompt = f"Review the following Java code and refactor the code in
        all ways possible and give the refactored code:\n\n{code}\n\n"
```

- Model:

```
# Send code to OpenAI API for analysis and refactoring
openai.api_key = API_KEY
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "system", "content": prompt}],
    max_tokens=800,  # Adjust max tokens as per requirement
    temperature=0.7,  # Adjust temperature as per requirement
    top_p=1
)
```

3. The refactored code is then written to a new file with a name prefixed by "refactored_".

- Pushing the code:

```
def create_branch(repo_path, branch_name):
    # Change directory to the repository
    os.chdir(repo_path)
    # Initialize git repository object
    repo = Repo(repo_path)
    if branch_name in repo.branches:
        print(f"Branch {branch_name} already exists.")
    else:
        try:
            repo.git.checkout('-b', branch_name)
            print(f"Created branch {branch_name}")
        except GitCommandError as e:
            print(f"An error occurred while creating branch {branch_name}: {e}")
```

TEAM_11

thank you!