Task 3c

God Class: AppContext (books-core src/main/java/com/sismics/books/core/model/context)

Identified Code Snippet:

AppContext.java

```
/**
 * Global application context.
 *
 * This class manages the global application context, including event
buses, services, and executors.
 *
 * @author jtremeaux
 */
public class AppContext {
    /**
     * Singleton instance.
     */
    private static AppContext instance;

    /**
     * Event bus.
     */
    private EventBus eventBus;

    /**
     * Generic asynchronous event bus.
     */
    private EventBus asyncEventBus;

    /**
     * Asynchronous event bus for mass imports.
     */
    private EventBus importEventBus;

    /**
     * Service to fetch book information.
     */
    private BookDataService bookDataService;

    /**
     * Facebook interaction service.
     */
    private FacebookService facebookService;

    /**
     * Asynchronous executors.
```

```
     */
    private List<ExecutorService> asyncExecutorList;

    /**
     * Private constructor.
     *
     * Initializes the event buses and starts required services.
     */
    private AppContext() {
        resetEventBus();

        bookDataService = new BookDataService();
        bookDataService.startAndWait();

        facebookService = new FacebookService();
        facebookService.startAndWait();
    }

    /**
     * (Re)-initializes the event buses.
     */
    private void resetEventBus() {
        eventBus = new EventBus();
        eventBus.register(new DeadEventListener());

        asyncExecutorList = new ArrayList<ExecutorService>();

        asyncEventBus = newAsyncEventBus();
        asyncEventBus.register(new UserAppCreatedAsyncListener());

        importEventBus = newAsyncEventBus();
        importEventBus.register(new BookImportAsyncListener());
    }

    /**
     * Returns a single instance of the application context.
     *
     * @return Application context
     */
    public static AppContext getInstance() {
        if (instance == null) {
            instance = new AppContext();
        }
        return instance;
    }

    /**
     * Creates a new asynchronous event bus.
     *
     * @return Async event bus
     */
    private EventBus newAsyncEventBus() {
        if (EnvironmentUtil.isUnitTest()) {
            return new EventBus();
```

```
            } else {
                ThreadPoolExecutor executor = new ThreadPoolExecutor(1, 1,
                        0L, TimeUnit.MILLISECONDS,
                        new LinkedBlockingQueue<Runnable>());
                asyncExecutorList.add(executor);
                return new AsyncEventBus(executor);
            }
        }

        /**
         * Getter of eventBus.
         *
         * @return eventBus
         */
        public EventBus getEventBus() {
            return eventBus;
        }

        /**
         * Getter of asyncEventBus.
         *
         * @return asyncEventBus
         */
        public EventBus getAsyncEventBus() {
            return asyncEventBus;
        }

        /**
         * Getter of importEventBus.
         *
         * @return importEventBus
         */
        public EventBus getImportEventBus() {
            return importEventBus;
        }

        /**
         * Getter of bookDataService.
         *
         * @return bookDataService
         */
        public BookDataService getBookDataService() {
            return bookDataService;
        }

        /**
         * Getter of facebookService.
         *
         * @return facebookService
         */
        public FacebookService getFacebookService() {
            return facebookService;
        }
    }
```

Apply Manual Refactoring

Refactored into:

1. `AppContext.java`

```
/**
 * Global application context.
 *
 * This class manages the global application context, including event
buses, services, and executors.
 *
 * @author jtremeaux
 */
public class AppContext {
    /**
     * Singleton instance.
     */
    private static AppContext instance;

    /**
     * Service manager.
     */
    private ServiceManager serviceManager;

    /**
     * Event bus manager.
     */
    private EventBusManager eventBusManager;

    /**
     * Private constructor.
     *
     * <p>
     * Initializes the service manager and event bus manager.
     * </p>
     */
    private AppContext() {
        serviceManager = new ServiceManager();
        eventBusManager = new EventBusManager();
    }

    /**
     * Returns a single instance of the application context.
     *
     * @return Application context
     */
    public static AppContext getInstance() {
        if (instance == null) {
            instance = new AppContext();
```

```
            }
            return instance;
        }

        /**
         * Retrieves the main event bus.
         *
         * @return The main event bus instance
         */
        public EventBus getEventBus() {
            return eventBusManager.getEventBus();
        }

        /**
         * Retrieves the asynchronous event bus.
         *
         * @return The asynchronous event bus instance
         */
        public EventBus getAsyncEventBus() {
            return eventBusManager.getAsyncEventBus();
        }

        /**
         * Retrieves the event bus for import events.
         *
         * @return The event bus for import events instance
         */
        public EventBus getImportEventBus() {
            return eventBusManager.getImportEventBus();
        }

        /**
         * Retrieves the BookDataService instance.
         *
         * @return The BookDataService instance
         */
        public BookDataService getBookDataService() {
            return serviceManager.getBookDataService();
        }

        /**
         * Retrieves the FacebookService instance.
         *
         * @return The FacebookService instance
         */
        public FacebookService getFacebookService() {
            return serviceManager.getFacebookService();
        }
    }
```

2. ServiceManager.java

```
/**
 * Service manager.
 *
 * This class manages the services of the application.
 *
 * @author jtremeaux
 */
public class ServiceManager {
    /**
     * Instance of BookDataService.
     */
    private BookDataService bookDataService;

    /**
     * Instance of FacebookService.
     */
    private FacebookService facebookService;

    /**
     * Constructs a new ServiceManager and initializes services.
     *
     * <p>
     * Initializes instances of {@code BookDataService} and {@code
FacebookService}.
     * </p>
     */
    public ServiceManager() {
        initializeServices();
    }

    /**
     * Initializes services.
     *
     * <p>
     * Creates instances of {@code BookDataService} and {@code
FacebookService} and starts them.
     * </p>
     */
    private void initializeServices() {
        bookDataService = new BookDataService();
        bookDataService.startAndWait();

        facebookService = new FacebookService();
        facebookService.startAndWait();
    }

    /**
     * Retrieves the instance of BookDataService.
     *
     * @return The instance of BookDataService
     */
    public BookDataService getBookDataService() {
```

```
        return bookDataService;
    }

    /**
     * Retrieves the instance of FacebookService.
     *
     * @return The instance of FacebookService
     */
    public FacebookService getFacebookService() {
        return facebookService;
    }
}
```

3. EventBusManager.java

```
/**
 * Event bus manager.
 *
 * This class manages the event buses of the application.
 *
 * @author jtremeaux
 */
public class EventBusManager {
    /**
     * Instance of EventBus.
     */
    private EventBus eventBus;

    /**
     * Instance of AsyncEventBus.
     */
    private EventBus asyncEventBus;

    /**
     * Instance of ImportEventBus.
     */
    private EventBus importEventBus;

    /**
     * List of executor services for async event buses.
     */
    private List<ExecutorService> asyncExecutorList = new ArrayList<>();

    /**
     * Constructs a new EventBusManager and initializes event buses.
     *
     * <p>
     * Initializes instances of {@code EventBus}, {@code AsyncEventBus},
and {@code ImportEventBus}.
     * </p>
     */
```

```
    public EventBusManager() {
        resetEventBus();
    }

    /**
     * Initializes event buses.
     *
     * <p>
     * Creates instances of {@code EventBus}, {@code AsyncEventBus}, and
{@code ImportEventBus}.
     * Registers appropriate event listeners.
     * </p>
     */
    private void resetEventBus() {
        eventBus = new EventBus();
        eventBus.register(new DeadEventListener());

        asyncEventBus = createAsyncEventBus();
        asyncEventBus.register(new UserAppCreatedAsyncListener());

        importEventBus = createAsyncEventBus();
        importEventBus.register(new BookImportAsyncListener());
    }

    /**
     * Creates an instance of AsyncEventBus.
     *
     * <p>
     * Creates an instance of AsyncEventBus with a single-threaded executor
service.
     * Registers the executor service to the asyncExecutorList.
     * </p>
     *
     * @return The instance of AsyncEventBus
     */
    private EventBus createAsyncEventBus() {
        if (EnvironmentUtil.isUnitTest()) {
            return new EventBus();
        } else {
            ThreadPoolExecutor executor = new ThreadPoolExecutor(1, 1,
                    0L, TimeUnit.MILLISECONDS,
                    new LinkedBlockingQueue<Runnable>());
            asyncExecutorList.add(executor);
            return new AsyncEventBus(executor);
        }
    }

    /**
     * Retrieves the instance of EventBus.
     *
     * @return The instance of EventBus
     */
    public EventBus getEventBus() {
        return eventBus;
```

```
    }

    /**
     * Retrieves the instance of AsyncEventBus.
     *
     * @return The instance of AsyncEventBus
     */
    public EventBus getAsyncEventBus() {
        return asyncEventBus;
    }

    /**
     * Retrieves the instance of ImportEventBus.
     *
     * @return The instance of ImportEventBus
     */
    public EventBus getImportEventBus() {
        return importEventBus;
    }
}
```

LLM Refactoring

```
/**
 * Global application context.
 * This class manages the initialization of various components and provides
access to them.
 */
public class AppContext {
    private static AppContext instance;

    private EventBus eventBus;
    private EventBus asyncEventBus;
    private EventBus importEventBus;
    private BookDataService bookDataService;
    private FacebookService facebookService;

    // Private constructor to prevent instantiation
    private AppContext() {
        initializeEventBuses();
        initializeServices();
    }

    public static synchronized AppContext getInstance() {
        if (instance == null) {
            instance = new AppContext();
        }
        return instance;
    }

    private void initializeEventBuses() {
```

```
        eventBus = new EventBus();
        eventBus.register(new DeadEventListener());

        asyncEventBus = newAsyncEventBus();
        asyncEventBus.register(new UserAppCreatedAsyncListener());

        importEventBus = newAsyncEventBus();
        importEventBus.register(new BookImportAsyncListener());
    }

    private EventBus newAsyncEventBus() {
        if (EnvironmentUtil.isUnitTest()) {
            return new EventBus();
        } else {
            return new AsyncEventBus(new ThreadPoolExecutor(1, 1, 0L,
TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>()));
        }
    }

    private void initializeServices() {
        bookDataService = new BookDataService();
        bookDataService.startAndWait();

        facebookService = new FacebookService();
        facebookService.startAndWait();
    }

    public EventBus getEventBus() {
        return eventBus;
    }

    public EventBus getAsyncEventBus() {
        return asyncEventBus;
    }

    public EventBus getImportEventBus() {
        return importEventBus;
    }

    public BookDataService getBookDataService() {
        return bookDataService;
    }

    public FacebookService getFacebookService() {
        return facebookService;
    }
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- The manual refactoring separates the functionalities into different manager classes (`ServiceManager` and `EventBusManager`).
- Each manager class is responsible for handling a specific set of related functionalities (e.g., managing services or event buses).
- The manual refactoring introduces a clear separation of concerns, making the codebase easier to understand and maintain.
- Each manager class encapsulates its own set of methods and responsibilities.

LLM-generated Refactoring:

- The LLM-generated refactoring consolidates all functionalities into a single `AppContext` class.
- The `AppContext` class manages both services and event buses, reducing the overall complexity of the codebase.
- The LLM-generated refactoring promotes a more concise structure by centralizing related functionalities within the same class.

Evaluation of Strengths and Weaknesses:

Manual Refactoring:

Strengths:

- **Clear separation of concerns**: Each manager class handles a specific set of responsibilities, enhancing code organization and readability.
- **Easier to maintain**: Changes to one aspect of the application (e.g., services or event buses) are less likely to affect other parts of the codebase.
- **Scalability**: The modular structure facilitates the addition of new functionalities or modifications to existing ones.

Weaknesses:

- **Increased number of classes**: Introducing multiple manager classes might result in a larger number of files, potentially complicating navigation and maintenance.
- **Manual effort required**: Refactoring manually demands meticulous analysis and understanding of the codebase, which can be time-consuming.

LLM-generated Refactoring:

Strengths:

- **Centralized management**: The single `AppContext` class centralizes the management of various components, promoting simplicity and reducing duplication.
- **Reduced complexity**: By consolidating functionalities into one class, the LLM-generated refactoring simplifies the codebase and makes it easier to comprehend.
- **Efficiency**: The automated refactoring process saves time and effort, especially for initial code structuring or quick iterations.

Weaknesses:

- **Potential for reduced clarity**: Concentrating all functionalities within a single class may lead to decreased clarity, especially for larger codebases or complex projects.
- **Lack of explicit separation**: Unlike the manual approach, where responsibilities are clearly divided into separate classes, the LLM-generated refactoring might lack explicit separation of concerns.
- **Limited adaptability**: The automated refactoring might not cater to specific project requirements or domain complexities as effectively as human-driven refactoring.

Scenarios:

LLMs Excel:

- When there's a need for rapid prototyping or generating an initial code structure.
- For repetitive tasks where a standardized pattern or structure suffices.
- When minimizing code duplication and promoting reusability are primary concerns.

Manual Intervention is Preferable:

- When specific and optimized code structures tailored to project requirements are necessary.
- For complex refactoring tasks demanding deep understanding of the codebase and domain logic.
- When clarity, maintainability, and scalability are paramount, and a human-driven approach can provide better insights and decisions.

God Class: BookResource (books-web com.sismics.books.rest.resource)

Identified Code Snippet:

BookResource.java:

```
@Path("/book")
public class BookResource extends BaseResource {
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(@FormParam("isbn") String isbn) throws
JSONException {
        // Implementation
    }

    @DELETE
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response delete() throws JSONException {
        // Implementation
    }

    @PUT
    @Path("manual")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add() throws JSONException {
        // Implementation
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update() throws JSONException {
        // Implementation
    }

    @GET
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response get() throws JSONException {
        // Implementation
    }

    @GET
    @Path("{id: [a-z0-9\\-]+}/cover")
    @Produces(MediaType.APPLICATION_OCTET_STREAM)
    public Response cover() throws JSONException {
        // Implementation
```

```
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}/cover")
    @Produces(MediaType.APPLICATION_JSON)
    public Response updateCover() throws JSONException {
        // Implementation
    }

    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        // Implementation
    }

    @PUT
    @Consumes("multipart/form-data")
    @Path("import")
    public Response importFile() throws JSONException {
        // Implementation
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}/read")
    @Produces(MediaType.APPLICATION_JSON)
    public Response read() throws JSONException {
        // Implementation
    }
}
```

Apply Manual Refactoring

Refactored into:

1. `BookManagementResource.java`

```
/**
 * Book REST resources.
 *
 * This class provides RESTful endpoints for managing books.
 *
 * @author bgamard
 */
@Path("/book")
public class BookManagementResource extends BaseResource {

    /**
     * Creates a new book.
     *
     * @param isbn ISBN Number
```

```
     * @return Response containing the ID of the created book
     * @throws JSONException If there is an error in JSON processing
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(
            @FormParam("isbn") String isbn) throws JSONException {
        // Method implementation...
    }

    /**
     * Deletes a book.
     *
     * @param userBookId User book ID
     * @return Response indicating the status of the operation
     * @throws JSONException If there is an error in JSON processing
     */
    @DELETE
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response delete(
            @PathParam("id") String userBookId) throws JSONException {
        // Method implementation...
    }

    /**
     * Add a book manually.
     *
     * @param title Title
     * @param subtitle Subtitle
     * @param author Author
     * @param description Description
     * @param isbn10 ISBN-10
     * @param isbn13 ISBN-13
     * @param pageCount Page count
     * @param language Language
     * @param publishDateStr Publish date
     * @param tagList List of tags
     * @return Response containing the ID of the created book
     * @throws JSONException If there is an error in JSON processing
     */
    @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 {
        // Method implementation...
    }

    /**
     * Updates the book.
     *
     * @param userBookId User book ID
     * @param title Title
     * @param subtitle Subtitle
     * @param author Author
     * @param description Description
     * @param isbn10 ISBN-10
     * @param isbn13 ISBN-13
     * @param pageCount Page count
     * @param language Language
     * @param publishDateStr Publish date
     * @param tagList List of tags
     * @return Response containing the ID of the updated book
     * @throws JSONException If there is an error in JSON processing
     */
    @POST
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    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 {
        // Method implementation...
    }

    /**
     * Get a book.
     *
     * @param userBookId User book ID
     * @return Response containing the book details
     * @throws JSONException If there is an error in JSON processing
     */
    @GET
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response get(
            @PathParam("id") String userBookId) throws JSONException {
        // Method implementation...
    }
```

```
    /**
     * Imports books.
     *
     * @param fileBodyPart File to import
     * @return Response indicating the status of the import operation
     * @throws JSONException If there is an error in JSON processing
     */
    @PUT
    @Consumes("multipart/form-data")
    @Path("import")
    public Response importFile(
            @FormDataParam("file") FormDataBodyPart fileBodyPart) throws
JSONException {
        // Method implementation...
    }
}
```

2. BookCoverResource.java

```
@Path("/book/{id: [a-z0-9\\\\-]+}/cover")
public class BookCoverResource extends BaseResource {

    /**
     * Returns a book cover.
     *
     * @param id User book ID
     * @return Response containing the book cover image
     * @throws JSONException If there is an error in JSON processing
     */
    @GET
    @Produces(MediaType.APPLICATION_OCTET_STREAM)
    public Response cover(
            @PathParam("id") final String userBookId) throws JSONException
{
        // Method implementation...
    }

    /**
     * Updates a book cover.
     *
     * @param id User book ID
     * @param imageUrl URL of the new cover image
     * @return Response indicating the status of the cover update
     * @throws JSONException If there is an error in JSON processing
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response updateCover(
            @PathParam("id") String userBookId,
            @FormParam("url") String imageUrl) throws JSONException {
        // Method implementation...
```

```
    }
}
```

3. BookListingResource.java

```
@Path("/book/list")
public class BookListingResource extends BaseResource {

    /**
     * Returns all books.
     *
     * @param limit Maximum number of books to retrieve per page
     * @param offset Number of books to skip for pagination
     * @param sortColumn Column to sort the books by
     * @param asc Set to true for ascending order, false for descending
order
     * @param search Search query string to filter books by title, author,
or other attributes
     * @param read Set to true to filter books by whether they have been
read or not
     * @param tagName Name of the tag to filter books by
     * @return Response containing the list of books matching the criteria
     * @throws JSONException If there is an error in JSON processing
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response list(
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset,
            @QueryParam("sort_column") Integer sortColumn,
            @QueryParam("asc") Boolean asc,
            @QueryParam("search") String search,
            @QueryParam("read") Boolean read,
            @QueryParam("tag") String tagName) throws JSONException {
        // Method implementation...
    }
}
```

4. BookReadStatusResource.java

```
@Path("/book/{id: [a-z0-9\\\\-]+}/read")
public class BookReadStatusResource extends BaseResource {

    /**
     * Set a book as read/unread.
     *
     * @param userBookId User book ID
     * @param read Read state
     * @return Response indicating the status of the operation
```

```
     * @throws JSONException If there is an error in JSON processing
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response read(
            @PathParam("id") final String userBookId,
            @FormParam("read") boolean read) throws JSONException {
        // Method implementation...
    }
}
```

LLM Refactoring

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

import javax.ws.rs.*;
import javax.ws.rs.core.*;
import org.codehaus.jettison.json.*;

/**
 * Book REST resources.
 *
 * @author bgamard
 */
@Path("/book")
public class BookResource extends BaseResource {
    private final BookService bookService;

    public BookResource(BookService bookService) {
        this.bookService = bookService;
    }

    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response addBook(@FormParam("isbn") String isbn) throws
JSONException {
        return bookService.addBook(isbn, authenticate(),
principal.getId());
    }

    @DELETE
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response deleteBook(@PathParam("id") String userBookId) throws
JSONException {
        return bookService.deleteBook(userBookId, authenticate(),
principal.getId());
    }

    // Other methods like updateBook, getBook, listBooks, etc.
```

```
}
```

```
package com.sismics.books.rest.service;

import com.sismics.books.core.dao.jpa.*;
import com.sismics.books.core.model.jpa.*;
import com.sismics.rest.exception.*;
import com.sismics.rest.util.ValidationUtil;
import org.codehaus.jettison.json.*;
import javax.ws.rs.core.*;
import java.text.*;
import java.util.*;

public class BookService {
    private final BookDao bookDao;
    private final UserBookDao userBookDao;

    public BookService(BookDao bookDao, UserBookDao userBookDao) {
        this.bookDao = bookDao;
        this.userBookDao = userBookDao;
    }

    public Response addBook(String isbn, boolean authenticated, String
userId) throws JSONException {
        if (!authenticated) {
            throw new ForbiddenClientException();
        }

        ValidationUtil.validateRequired(isbn, "isbn");

        Book book = bookDao.getByIsbn(isbn);
        if (book == null) {
            // Handle book retrieval from public API and saving in the
database
            // book = retrieveBookFromApi(isbn);
            // bookDao.create(book);
        }

        UserBook userBook = userBookDao.getByBook(book.getId(), userId);
        if (userBook == null) {
            userBook = new UserBook();
            userBook.setUserId(userId);
            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();
    }

    public Response deleteBook(String userBookId, boolean authenticated,
String userId) throws JSONException {
        if (!authenticated) {
            throw new ForbiddenClientException();
        }

        UserBook userBook = userBookDao.getUserBook(userBookId, userId);
        if (userBook == null) {
            throw new ClientException("BookNotFound", "Book not found with
id " + userBookId);
        }

        userBookDao.delete(userBook.getId());

        JSONObject response = new JSONObject();
        response.put("status", "ok");
        return Response.ok().entity(response).build();
    }

    // Other methods like updateBook, getBook, listBooks, etc.
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- The manual refactoring separates the functionalities into different resource classes (BookManagementResource, BookCoverResource, BookListingResource, BookReadStatusResource).
- Each resource class is responsible for handling a specific set of related operations (e.g., adding/deleting books, managing book covers, listing books, updating read status).
- The manual refactoring introduces a clear separation of concerns, making the codebase easier to understand and maintain.
- Each resource class has its own set of methods corresponding to the HTTP methods it handles.

LLM-generated Refactoring:

- The LLM-generated refactoring consolidates all functionalities into a single BookResource class.
- The BookResource class delegates the handling of different operations to a separate service class (BookService).
- The service class encapsulates the business logic for each operation, providing a more modular and organized structure.
- The LLM-generated refactoring reduces code duplication by centralizing common functionalities within the service class.

Evaluation of Strengths and Weaknesses:

Manual Refactoring:

Strengths:

- **Clear separation of concerns**: Each resource class handles a specific set of operations, promoting code organization and readability.
- **Easier to maintain**: Changes to one set of functionalities are less likely to affect other parts of the codebase, reducing the risk of unintended consequences.
- **Scalability**: Adding new functionalities or modifying existing ones can be done more easily due to the modular structure.

Weaknesses:

- **Increased number of classes**: Having multiple resource classes might lead to a larger number of files, potentially making navigation more complex.
- **Manual effort required**: Refactoring manually requires careful analysis and understanding of the codebase, which can be time-consuming.

LLM-generated Refactoring:

Strengths:

- **Centralized business logic**: The service class centralizes the business logic, promoting code reusability and maintainability.
- **Reduced code duplication**: Common functionalities are encapsulated within the service class, minimizing redundant code.
- **Simplified resource class**: The BookResource class becomes less cluttered, focusing primarily on request handling and delegation.

Weaknesses:

- **Potential for complexity**: Concentrating all functionalities within a single class may lead to increased complexity and difficulty in understanding.
- **Lack of explicit separation**: Unlike the manual approach, where functionalities are clearly separated into different classes, the LLM-generated approach relies on comments or documentation to indicate the purpose of each method.
- **Limited control**: The LLM-generated refactoring might not always produce the most optimal or intuitive structure, as it lacks human judgment and context awareness.

Scenarios:

LLMs Excel:

- When there's a need to quickly prototype or generate initial code structure.
- For repetitive tasks where a standardized pattern or structure is sufficient.
- When there's a desire to minimize code duplication and promote reusability.

Manual Intervention is Preferable:

- When there's a need for a specific and optimized code structure tailored to the project's requirements.
- For complex refactoring tasks that require deep understanding of the codebase and domain logic.
- When clarity, maintainability, and scalability are critical factors, and a human-driven approach can provide better insights and decisions.

Lazy Class & Deficient Encapsulation: Constants (com.sismics.books.core.constant)

Identified Code Snippet:

Constants.java:

```
/**
 * Application constants.
 *
 * @author jtremeaux
 */
public class Constants {
    /**
     * Default locale.
     */
    public static final String DEFAULT_LOCALE_ID = "en";

    /**
     * Default timezone ID.
     */
    public static final String DEFAULT_TIMEZONE_ID = "Europe/London";

    /**
     * Default theme ID.
     */
    public static final String DEFAULT_THEME_ID = "default.less";

    /**
     * Administrator's default password ("admin").
     */
    public static final String DEFAULT_ADMIN_PASSWORD =
"$2a$05$6Ny3TjrW3aVAL1or2SlcR.fhuDgPKp5jp.P9fBXwVNePgeLqb4i3C";

    /**
     * Default generic user role.
     */
    public static final String DEFAULT_USER_ROLE = "user";
}
```

UserDao.java

```
/**
 * User DAO.
 *
 * @author jtremeaux
```

```
 */
public class UserDao {
    ///
    /// ... something ...
    ///

    /**
     * Creates a new user.
     *
     * @param user User to create
     * @return User ID
     * @throws Exception
     */
    public String create(User user) throws Exception {
        ///
        /// ... something ...
        ///

        user.setTheme(Constants.DEFAULT_THEME_ID);

        ///
        /// ... something ...
        ///
    }

    ///
    /// ... something ...
    ///
}
```

LocaleUtil.java

```


/**
 * Locale utilities.
 *
 * @author jtremeaux
 */
public class LocaleUtil {
    ///
    /// ... something ...
    ///

    /**
     * Extracts the ID of the locale from the HTTP Accept-Language header.
     *
     * @param acceptLanguageHeader header
     * @return Locale ID
     */
    public static String getLocaleIdFromAcceptLanguage(String
acceptLanguageHeader) {
        ///
```

```
        /// ... something ...
        ///

        if (StringUtils.isNotBlank(localeId)) {
            ///
            /// ... something ...
            ///

            localeId = Constants.DEFAULT_LOCALE_ID;

            ///
            /// ... something ...
            ///
        }
        if (StringUtils.isBlank(localeId)) {
            localeId = Constants.DEFAULT_LOCALE_ID;
        }

        ///
        /// ... something ...
        ///
    }
}
```

`UserResource.java` (Changed to `UserCrudResource.java` after God Smell refactoring)

```
/**
 * User REST resources for CRUD Operations on User.
 *
 * @author jtremeaux
 */
@Path("/user")
public class UserCrudResource extends BaseResource {

    /**
     * Returns the information about the connected user.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        JSONObject response = new JSONObject();
        if (!authenticate()) {

            // something

            if (adminUser != null && adminUser.getDeleteDate() == null) {
                response.put("is_default_password",
Constants.DEFAULT_ADMIN_PASSWORD.equals(adminUser.getPassword()));
```

```
            }
        } else {
            // something

            response.put("is_default_password",
hasBaseFunction(BaseFunction.ADMIN) &&
Constants.DEFAULT_ADMIN_PASSWORD.equals(user.getPassword()));
        }

        return Response.ok().entity(response).build();
    }

    /**
     * Creates a new user.
     *
     * @param username User's username
     * @param password Password
     * @param email E-Mail
     * @param localeId Locale ID
     * @return Response
     * @throws JSONException
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email) throws JSONException {

        // ... something ...

        // Create the user
        User user = new User();
        user.setRoleId(Constants.DEFAULT_USER_ROLE);

        // ... something ...

        user.setLocaleId(Constants.DEFAULT_LOCALE_ID);

        // ... something ...
    }

    // something

}
```

RequestContextFilter.java:

```
/**
 * Filter used to process a couple things in the request context.
```

```
 *
 * @author jtremeaux
 */
public class RequestContextFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Force the locale in order to not depend on the execution
environment
        Locale.setDefault(new Locale(Constants.DEFAULT_LOCALE_ID));

        // something
    }

    // ...other methods...
}
```

TokenBasedSecurityFilter.java:

```
/**
 * This filter is used to authenticate the user having an active session
via an authentication token stored in database.
 * The filter extracts the authentication token stored in a cookie.
 * If the ocokie exists and the token is valid, the filter injects a
UserPrincipal into a request attribute.
 * If not, the user is anonymous, and the filter injects a
AnonymousPrincipal into the request attribute.
 *
 * @author jtremeaux
 */
public class TokenBasedSecurityFilter implements Filter {
    // something

    /**
     * Inject an anonymous user into the request attributes.
     *
     * @param request HTTP request
     */
    private void injectAnonymousUser(HttpServletRequest request) {
        // something


anonymousPrincipal.setDateTimeZone(DateTimeZone.forID(Constants.DEFAULT_TIM
EZONE_ID));

        // something
    }
}
```

Apply Manual Refactoring

1. Removed `Constants.java`

`UserDao.java`

```
/**
 * User DAO.
 *
 * @author jtremeaux
 */
public class UserDao {

    private String DEFAULT_THEME_ID = "default.less";

    ///
    /// ... something ...
    ///

    /**
     * Creates a new user.
     *
     * @param user User to create
     * @return User ID
     * @throws Exception
     */
    public String create(User user) throws Exception {
        ///
        /// ... something ...
        ///

        user.setTheme(DEFAULT_THEME_ID);

        ///
        /// ... something ...
        ///
    }

    ///
    /// ... something ...
    ///
}
```

`LocaleUtil.java`

```
/**
```

```
 * Locale utilities.
 *
 * @author jtremeaux
 */
public class LocaleUtil {
    ///
    /// ... something ...
    ///

    private static String DEFAULT_LOCALE_ID = "en";

    /**
     * Extracts the ID of the locale from the HTTP Accept-Language header.
     *
     * @param acceptLanguageHeader header
     * @return Locale ID
     */
    public static String getLocaleIdFromAcceptLanguage(String
acceptLanguageHeader) {
        ///
        /// ... something ...
        ///

        if (StringUtils.isNotBlank(localeId)) {
            ///
            /// ... something ...
            ///

            localeId = DEFAULT_LOCALE_ID;

            ///
            /// ... something ...
            ///
        }
        if (StringUtils.isBlank(localeId)) {
            localeId = DEFAULT_LOCALE_ID;
        }

        ///
        /// ... something ...
        ///
    }
}
```

`UserResource.java` (Changed to `UserCrudResource.java` after God Smell refactoring)

```


/**
 * User REST resources for CRUD Operations on User.
 *
 * @author jtremeaux
 */
```

```
@Path("/user")
public class UserCrudResource extends BaseResource {

    private String DEFAULT_ADMIN_PASSWORD =
"$2a$05$6Ny3TjrW3aVAL1or2SlcR.fhuDgPKp5jp.P9fBXwVNePgeLqb4i3C";
    private String DEFAULT_USER_ROLE = "user";
    private String DEFAULT_LOCALE_ID = "en";

    /**
     * Returns the information about the connected user.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        JSONObject response = new JSONObject();
        if (!authenticate()) {

            // something

            if (adminUser != null && adminUser.getDeleteDate() == null) {
                response.put("is_default_password",
DEFAULT_ADMIN_PASSWORD.equals(adminUser.getPassword()));
            }
        } else {
            // something

            response.put("is_default_password",
hasBaseFunction(BaseFunction.ADMIN) &&
DEFAULT_ADMIN_PASSWORD.equals(user.getPassword()));
        }

        return Response.ok().entity(response).build();
    }

    /**
     * Creates a new user.
     *
     * @param username User's username
     * @param password Password
     * @param email E-Mail
     * @param localeId Locale ID
     * @return Response
     * @throws JSONException
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email) throws JSONException {
```

```
        // ... something ...

        // Create the user
        User user = new User();
        user.setRoleId(DEFAULT_USER_ROLE);

        // ... something ...

        user.setLocaleId(DEFAULT_LOCALE_ID);

        // ... something ...
    }

    // something

}
```

RequestContextFilter.java:

```
/**
 * Filter used to process a couple things in the request context.
 *
 * @author jtremeaux
 */
public class RequestContextFilter implements Filter {
    private String DEFAULT_LOCALE_ID = "en";

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Force the locale in order to not depend on the execution
environment
        Locale.setDefault(new Locale(DEFAULT_LOCALE_ID));

        // something
    }

    // ...other methods...
}
```

TokenBasedSecurityFilter.java:

```
/**
 * This filter is used to authenticate the user having an active session
via an authentication token stored in database.
 * The filter extracts the authentication token stored in a cookie.
 * If the ocokie exists and the token is valid, the filter injects a
```

```
UserPrincipal into a request attribute.
 * If not, the user is anonymous, and the filter injects a
AnonymousPrincipal into the request attribute.
 *
 * @author jtremeaux
 */
public class TokenBasedSecurityFilter implements Filter {
    // something
    private String DEFAULT_TIMEZONE_ID = "Europe/London";

    /**
     * Inject an anonymous user into the request attributes.
     *
     * @param request HTTP request
     */
    private void injectAnonymousUser(HttpServletRequest request) {
        // something


anonymousPrincipal.setDateTimeZone(DateTimeZone.forID(DEFAULT_TIMEZONE_ID))
;

        // something
    }
}
```

LLM Refactoring:

Constants.java:

```
/**
 * Application constants.
 *
 * @author jtremeaux
 */
public class Constants {
    private Constants() {
        // private constructor to prevent instantiation
    }

    private static final String DEFAULT_LOCALE_ID = "en";
    private static final String DEFAULT_TIMEZONE_ID = "Europe/London";
    private static final String DEFAULT_THEME_ID = "default.less";
    private static final String DEFAULT_ADMIN_PASSWORD =
"$2a$05$6Ny3TjrW3aVAL1or2SlcR.fhuDgPKp5jp.P9fBXwVNePgeLqb4i3C";
    private static final String DEFAULT_USER_ROLE = "user";

    public static String getDefaultLocaleId() {
        return DEFAULT_LOCALE_ID;
    }

    public static String getDefaultTimezoneId() {
        return DEFAULT_TIMEZONE_ID;
    }

    public static String getDefaultThemeId() {
        return DEFAULT_THEME_ID;
    }

    public static String getDefaultAdminPassword() {
        return DEFAULT_ADMIN_PASSWORD;
    }

    public static String getDefaultUserRole() {
        return DEFAULT_USER_ROLE;
    }
}
```

UserDao.java:

```
/**
 * User DAO.
 *
 * @author jtremeaux
 */
public class UserDao {
```

```
    ///
    /// ... something ...
    ///

    /**
     * Creates a new user.
     *
     * @param user User to create
     * @return User ID
     * @throws Exception
     */
    public String create(User user) throws Exception {
        ///
        /// ... something ...
        ///

        user.setTheme(Constants.getDefaultThemeId());

        ///
        /// ... something ...
        ///
    }

    ///
    /// ... something ...
    ///
}
```

LocaleUtil.java:

```
/**
 * Locale utilities.
 *
 * @author jtremeaux
 */
public class LocaleUtil {
    ///
    /// ... something ...
    ///

    /**
     * Extracts the ID of the locale from the HTTP Accept-Language header.
     *
     * @param acceptLanguageHeader header
     * @return Locale ID
     */
    public static String getLocaleIdFromAcceptLanguage(String
acceptLanguageHeader) {
        ///
        /// ... something ...
        ///
```

```
        String localeId = ""; // declare localeId here

        if (StringUtils.isNotBlank(localeId)) {
            ///
            /// ... something ...
            ///

            localeId = Constants.getDefaultLocaleId();

            ///
            /// ... something ...
            ///
        }
        if (StringUtils.isBlank(localeId)) {
            localeId = Constants.getDefaultLocaleId();
        }

        ///
        /// ... something ...
        ///
        return localeId;
    }
}
```

`UserCrudResource.java` (Changed to `UserCrudResource.java` after God Smell refactoring):

```
/**
 * User REST resources for CRUD Operations on User.
 *
 * @author jtremeaux
 */
@Path("/user")
public class UserCrudResource extends BaseResource {

    /**
     * Returns the information about the connected user.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        JSONObject response = new JSONObject();
        if (!authenticate()) {

            // something

            if (adminUser != null && adminUser.getDeleteDate() == null) {
                response.put("is_default_password",
Constants.getDefaultAdminPassword().equals(adminUser.getPassword()));
            }
```

```
            } else {
                // something

                response.put("is_default_password",
    hasBaseFunction(BaseFunction.ADMIN) &&
    Constants.getDefaultAdminPassword().equals(user.getPassword()));
            }

            return Response.ok().entity(response).build();
        }

        /**
         * Creates a new user.
         *
         * @param username User's username
         * @param password Password
         * @param email E-Mail
         * @param localeId Locale ID
         * @return Response
         * @throws JSONException
         */
        @PUT
        @Produces(MediaType.APPLICATION_JSON)
        public Response register(
            @FormParam("username") String username,
            @FormParam("password") String password,
            @FormParam("locale") String localeId,
            @FormParam("email") String email) throws JSONException {

            // ... something ...

            // Create the user
            User user = new User();
            user.setRoleId(Constants.getDefaultUserRole());

            // ... something ...

            user.setLocaleId(Constants.getDefaultLocaleId());

            // ... something ...
        }

        // something

    }
```

RequestContextFilter.java:

```
    /**
     * Filter used to process a couple things in the request context.
     *
     * @author jtremeaux
```

```
 */
public class RequestContextFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Force the locale in order to not depend on the execution
environment
        Locale.setDefault(new Locale(Constants.getDefaultLocaleId()));

        // something
    }

    // ...other methods...
}
```

TokenBasedSecurityFilter.java:

```
/**
 * This filter is used to authenticate the user having an active session
via an authentication token stored in database.
 * The filter extracts the authentication token stored in a cookie.
 * If the ocokie exists and the token is valid, the filter injects a
UserPrincipal into a request attribute.
 * If not, the user is anonymous, and the filter injects a
AnonymousPrincipal into the request attribute.
 *
 * @author jtremeaux
 */
public class TokenBasedSecurityFilter implements Filter {
    // something

    /**
     * Inject an anonymous user into the request attributes.
     *
     * @param request HTTP request
     */
    private void injectAnonymousUser(HttpServletRequest request) {
        // something


anonymousPrincipal.setDateTimeZone(DateTimeZone.forID(Constants.getDefaultT
imezoneId()));

        // something
    }
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- `Removal of Constants Class`: The manual refactoring involved removing the Constants class and directly using the constants within the classes where they were needed.
- `Constants as Instance Variables`: Constants were converted into instance variables within each class where they were used, such as UserDao, LocaleUtil, UserCrudResource, RequestContextFilter, and TokenBasedSecurityFilter.
- `Duplication of Constants`: Some constants were duplicated across different classes, leading to potential inconsistencies and maintenance overhead.

LLM Refactoring:

- `Introduction of Constants Class`: The LLM-generated refactoring reintroduced the Constants class to encapsulate the constants.
- `Constants as Static Methods`: Constants were defined as static methods within the Constants class, providing a centralized location for accessing them.
- `Elimination of Duplication`: By using a centralized Constants class, duplication of constants across multiple classes was avoided, ensuring consistency and reducing maintenance overhead.
- `Method-level Encapsulation`: Constants were encapsulated within methods, providing clear access points and promoting readability and maintainability.

Evaluation:

Manual Refactoring:

Strengths:

- `Direct control over refactoring process`: Manual refactoring allows developers to have full control over the refactoring process, making it easier to implement specific design decisions or optimizations.
- `Flexibility`: Developers can tailor the refactoring process to suit the specific needs and constraints of the project.
- `Immediate understanding`: Since developers are directly involved in the refactoring process, they have a clear understanding of the changes made and their implications.

Weaknesses:

- `Prone to human error`: Manual refactoring relies on developers to make changes correctly, which can introduce human errors, such as forgetting to update all occurrences of a constant or inadvertently introducing new bugs.
- `Time-consuming`: Manual refactoring can be time-consuming, especially for large codebases or complex refactorings, as developers need to carefully analyze the code and make changes manually.
- `Potential inconsistency`: Without strict guidelines or automated checks, manual refactoring can lead to inconsistencies or deviations from best practices across different parts of the codebase.

LLM Refactoring:

Strengths:

- `Automation`: LLM-generated refactoring automates the process of identifying and implementing refactorings, saving developers time and effort.
- `Consistency`: LLMs ensure consistency by applying refactorings uniformly across the codebase, reducing the risk of inconsistencies or errors introduced by manual refactoring.
- `Encapsulation`: LLM-generated refactorings often encapsulate constants within methods or classes, promoting better encapsulation and code organization.
- `Reduced cognitive load`: LLMs can handle repetitive or mundane refactorings, freeing developers to focus on more complex or creative tasks.

Weaknesses:

- `Lack of context awareness`: LLMs may lack contextual understanding, leading to suboptimal refactorings or failure to consider specific project requirements or constraints.
- `Limited customization`: LLMs may not offer the same level of customization or flexibility as manual refactoring, making it challenging to tailor the refactoring process to suit specific needs.
- `Potential for over-engineering`: LLM-generated refactorings may introduce unnecessary complexity or abstraction if not carefully controlled or reviewed by human developers.

Scenarios:

LLMs Excel:

- `Routine Refactorings`: LLMs excel at performing routine refactorings, such as extracting constants or methods, renaming variables, or optimizing imports, freeing developers from repetitive tasks.
- `Consistency Enforcement`: LLMs are valuable for enforcing coding standards and ensuring consistency across large codebases, where manual enforcement would be impractical.
- `Codebase Initialization`: LLMs can be useful for quickly setting up a new codebase or applying standardized patterns and practices, saving time during project initialization.

Manual Intervention is Preferable:

- `Complex Refactorings`: For complex or domain-specific refactorings requiring deep understanding of the codebase or project domain, manual intervention is often preferable to ensure optimal results.
- `Strategic Design Decisions`: When refactoring involves making strategic design decisions or trade-offs, such as choosing between different architectural patterns or optimization strategies, manual intervention allows developers to exercise judgment and expertise.
- `Legacy Codebases`: In legacy codebases with convoluted or poorly documented code, manual intervention may be necessary to untangle dependencies, clarify intent, and ensure that refactorings align with the broader project goals.

In conclusion, while LLMs offer significant advantages in terms of automation, consistency, and efficiency, manual intervention remains essential for complex refactorings, strategic design decisions, and navigating legacy codebases where human expertise and judgment are indispensable. Combining the strengths of LLMs with human oversight and intervention can lead to optimal refactorings that improve code readability, maintainability, and efficiency

Missing Abstraction: UserBookDao (books-web com.sismics.books.rest.resource)

Identified Code Snippet:

UserBookDao.java:

```
package com.sismics.books.core.dao.jpa;

import com.sismics.books.core.dao.jpa.criteria.UserBookCriteria;
import com.sismics.books.core.dao.jpa.dto.UserBookDto;
import com.sismics.books.core.model.jpa.UserBook;
import com.sismics.books.core.util.jpa.PaginatedList;
import com.sismics.books.core.util.jpa.SortCriteria;
import com.sismics.util.context.ThreadLocalContext;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import java.util.UUID;

/**
 * User book DAO.
 */
public class UserBookDao {

    /**
     * Creates a new user book.
     *
     * @param userBook UserBook
     * @return New ID
     */
    public String create(UserBook userBook) {
        // Implementation
    }

    /**
     * Deletes a user book.
     *
     * @param id User book ID
     */
    public void delete(String id) {
        // Implementation
    }

    /**
     * Return a user book.
     *
     * @param userBookId User book ID
     * @param userId User ID
     * @return User book
```

```
     */
    public UserBook getUserBook(String userBookId, String userId) {
        // Implementation
    }

    /**
     * Return a user book by ID.
     *
     * @param userBookId User book ID
     * @return User book
     */
    public UserBook getUserBook(String userBookId) {
        // Implementation
    }

    /**
     * Return a user book by book ID and user ID.
     *
     * @param bookId Book ID
     * @param userId User ID
     * @return User book
     */
    public UserBook getByBook(String bookId, String userId) {
        // Implementation
    }

    /**
     * Searches user books by criteria.
     *
     * @param paginatedList List of user books (updated by side effects)
     * @param criteria Search criteria
     * @param sortCriteria Sort criteria
     */
    public void findByCriteria(PaginatedList<UserBookDto> paginatedList,
UserBookCriteria criteria, SortCriteria sortCriteria) {
        // Implementation
    }
}
```

Apply Manual Refactoring

Refactored into:

1. BaseDao.java:

```
/*
* Interface BaseDao which is used by the class UserBookDao
*/
public interface BaseDao<T> {
    String create(T entity);
    void delete(String id);
    T getById(String id);
}
```

2. UserBookDao.java:

```
public class UserBookDao implements BaseDao<UserBook> {

@Override
public String create(UserBook userBook) {
    // Create the UUID
    userBook.setId(UUID.randomUUID().toString());

    // Create the user book
    EntityManager em = ThreadLocalContext.get().getEntityManager();
    em.persist(userBook);

    return userBook.getId();
}

@Override
public void delete(String id) {
    EntityManager em = ThreadLocalContext.get().getEntityManager();

    // Get the user book
    Query q = em.createQuery("select ub from UserBook ub where ub.id =
:id and ub.deleteDate is null");
    q.setParameter("id", id);
    UserBook userBookDb = (UserBook) q.getSingleResult();

    // Delete the user book
    Date dateNow = new Date();
    userBookDb.setDeleteDate(dateNow);
}

@Override
public UserBook getById(String id) {
    EntityManager em = ThreadLocalContext.get().getEntityManager();
    Query q = em.createQuery("select ub from UserBook ub where ub.id =
```

```
:userBookId and ub.deleteDate is null");
    q.setParameter("userBookId", id);
    try {
        return (UserBook) q.getSingleResult();
    } catch (NoResultException e) {
        return null;
    }
}

/*Implementation remains the same*/

public UserBook getUserBook(String userBookId, String userId) {
    // Simplified to use getById for demonstrating purpose
    return this.getById(userBookId); // This is a simplification for
illustration
}

public UserBook getUserBook(String userBookId) {
    // Delegating to getById demonstrates use of inherited methods
    return this.getById(userBookId); // Adjusted to highlight use of
BaseDao method
}

/*Implementation remains the same*/

public UserBook getByBook(String bookId, String userId) {
    /*Implementation remains the same*/
}

public void findByCriteria(PaginatedList<UserBookDto> paginatedList,
UserBookCriteria criteria, SortCriteria sortCriteria) throws Exception
{
    /*Implementation remains the same*/
    }
}
```

LLM Refactoring

```
public class UserBookDao {

    /* Implementation remains the same for create, delete, getById methods
*/

    public class QueryBuilder {
        private final StringBuilder query;
        private final Map<String, Object> parameters;

        public QueryBuilder(String baseQuery) {
            this.query = new StringBuilder(baseQuery);
            this.parameters = new HashMap<>();
        }

        public QueryBuilder addCondition(String condition) {
            if (query.toString().contains("where")) {
                query.append(" and ");
            } else {
                query.append(" where ");
            }
            query.append(condition);
            return this;
        }

        public QueryBuilder addParameter(String name, Object value) {
            parameters.put(name, value);
            return this;
        }

        public String getQuery() {
            return query.toString();
        }

        public Map<String, Object> getParameters() {
            return parameters;
        }
    }

    public interface ResultMapper<T> {
        T map(Object result);
    }

    public void findByCriteria(PaginatedList<UserBookDto> paginatedList,
UserBookCriteria criteria, SortCriteria sortCriteria) {
        QueryBuilder queryBuilder = new QueryBuilder("select ub from
UserBook ub join T_BOOK b on ub.bookId = b.id where ub.deleteDate is null")
                .addParameter("userId", criteria.getUserId());

        if (!Strings.isNullOrEmpty(criteria.getSearch())) {
            queryBuilder.addCondition("b.title like :search or b.subtitle
```

```
like :search or b.author like :search")
                        .addParameter("search", "%" + criteria.getSearch()
+ "%");
        }

        // Add other criteria...

        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query query = em.createQuery(queryBuilder.getQuery());
        queryBuilder.getParameters().forEach(query::setParameter);

        List<UserBookDto> dtos = query.getResultList().stream()
                                      .map(result ->
mapToUserBookDto(result))
                                      .collect(Collectors.toList());

        paginatedList.setResultList(dtos);
    }

    private UserBookDto mapToUserBookDto(Object[] result) {
        // Mapping logic here
        return new UserBookDto(); // Simplified for demonstration
    }

    /* Other methods remain unchanged */
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- The manual refactoring separates the functionalities into different resource classes (UserCrudResource, UserListResource, UserLoginResource, UserLogoutResource, UserSessionResource).
- Each resource class is responsible for handling a specific set of related operations (e.g., CRUD operations, login/logout, session management).
- The manual refactoring introduces a clear separation of concerns, making the codebase easier to understand and maintain.
- Each resource class has its own set of methods corresponding to the HTTP methods it handles.

LLM-generated Refactoring:

- The LLM-generated refactoring consolidates all functionalities into a single UserResource class. The UserResource class delegates the handling of different operations to a separate service class (UserManagementService).
- The service class encapsulates the business logic for each operation, providing a more modular and organized structure.
- The LLM-generated refactoring reduces code duplication by centralizing common functionalities within the service class.

Evaluation of Strengths and Weaknesses:

Manual Refactoring:

Strengths:

- `Clear separation of concerns`: Each resource class handles a specific set of operations, promoting code organization and readability.
- `Easier to maintain`: Changes to one set of functionalities are less likely to affect other parts of the codebase, reducing the risk of unintended consequences.
- `Scalability`: Adding new functionalities or modifying existing ones can be done more easily due to the modular structure.

Weaknesses:

- `Increased number of classes`: Having multiple resource classes might lead to a larger number of files, potentially making navigation more complex.
- `Manual effort required`: Refactoring manually requires careful analysis and understanding of the codebase, which can be time-consuming.

LLM-generated Refactoring:

Strengths:

- `Centralized business logic`: The service class centralizes the business logic, promoting code reusability and maintainability.
- `Reduced code duplication`: Common functionalities are encapsulated within the service class, minimizing redundant code.
- `Simplified resource class`: The UserResource class becomes less cluttered, focusing primarily on request handling and delegation.

Weaknesses:

- `Potential for complexity`: Concentrating all functionalities within a single class may lead to increased complexity and difficulty in understanding.
- `Lack of explicit separation`: Unlike the manual approach, where functionalities are clearly separated into different classes, the LLM-generated approach relies on comments or documentation to indicate the purpose of each method.
- `Limited control`: The LLM-generated refactoring might not always produce the most optimal or intuitive structure, as it lacks human judgment and context awareness.

Scenarios:

LLMs Excel:

- When there's a need to quickly prototype or generate initial code structure.
- For repetitive tasks where a standardized pattern or structure is sufficient.
- When there's a desire to minimize code duplication and promote reusability.

Manual Intervention is Preferable:

- When there's a need for a specific and optimized code structure tailored to the project's requirements.
- For complex refactoring tasks that require deep understanding of the codebase and domain logic.
- When clarity, maintainability, and scalability are critical factors, and a human-driven approach can provide better insights and decisions.

Speculative Generality

Identified Code Snippet:

1. `UserAppDao.java`:

```
/**
 * User app DAO.
 *
 * @author jtremeaux
 */
public class UserAppDao {
    /**
     * Create a new connection.
     *
     * @param userApp User app
     * @return ID
     */
    public String create(UserApp userApp) {
        // Implementation
        return null; // Placeholder return
    }

    /**
     * Delete a connection.
     *
     * @param id User app ID
     */
    public void delete(String id) {
        // Implementation
    }

    /**
     * Deletes connection linked to a user and an application.
     *
     * @param userId User ID
     * @param appId App ID
     */
    public void deleteByUserIdAndAppId(String userId, String appId) {
        // Implementation
    }

    /**
     * Search and returns a non-deleted connection.
     *
     * @param id User app ID
     * @return User app
     */
    public UserApp getActiveById(String id) {
        // Implementation
```

```
        return null; // Placeholder return
    }

    /**
     * Search and returns a non-deleted connection by user and app.
     *
     * @param userId User ID
     * @param appId App ID
     * @return User app
     */
    public UserApp getActiveByUserIdAndAppId(String userId, String
appId) {
        // Implementation
        return null; // Placeholder return
    }

    /**
     * Search and returns application list with connection status.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findByUserId(String userId) {
        // Implementation
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns user's connected applications.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findConnectedByUserId(String userId) {
        // Implementation
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns applications connected to a user.
     *
     * @param appId Application ID
     * @return User app list
     */
    public List<UserAppDto> findByAppId(String appId) {
        // Implementation
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Updates a connection.
     *
     * @param userApp User app
     * @return Updated user app
```

```
     */
    public UserApp update(UserApp userApp) {
        // Implementation
        return null; // Placeholder return
    }
}
```

2. `UserContactDao.java`:

```
/**
 * User's contact DAO.
 *
 * @author jtremeaux
 */
public class UserContactDao {
    /**
     * Create a new contact.
     *
     * @param userContact UserContact
     * @return ID
     */
    public String create(UserContact userContact) {
        //implementation
    }

    /**
     * Returns users' contact.
     *
     * @param userId User ID
     * @param appId Application ID
     * @return Users' contact
     */
    @SuppressWarnings("unchecked")
    public List<UserContactDto> findByUserIdAndAppId(String userId,
String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select uc.USC_ID_C,
uc.USC_EXTERNALID_C ");
        sb.append(" from T_USER_CONTACT uc");
        sb.append(" where uc.USC_IDUSER_C = :userId and uc.USC_IDAPP_C
= :appId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userId", userId);
        q.setParameter("appId", appId);
        List<Object[]> l = q.getResultList();
        List<UserContactDto> userContactDtoList = new
ArrayList<UserContactDto>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
```

```
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDtoList.add(userContactDto);
        }

        return userContactDtoList;
    }

    /**
     * Updates last check user contact date.
     *
     * @param userId User ID
     * @param appId Application ID
     */
    public void updateByUserIdAndAppId(String userId, String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("update T_USER_CONTACT
uc");
        sb.append(" set uc.USC_UPDATEDATE_D = :updateDate ");
        sb.append(" where uc.USC_IDUSER_C = :userId and uc.USC_IDAPP_C
= :appId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("updateDate", new Date());
        q.setParameter("userId", userId);
        q.setParameter("appId", appId);
        q.executeUpdate();
    }

    /**
     * Delete a contact.
     *
     * @param id User contact ID
     */
    public void delete(String id) {
        //implementation
    }

    /**
     * Search user's contacts.
     *
     * @param criteria Search criteria
     * @param paginatedList Paginated list (filled by side effect)
     */
    public void findByCriteria(PaginatedList<UserContactDto>
paginatedList, UserContactCriteria criteria) {
        Map<String, Object> parameterMap = new HashMap<String, Object>
();

        StringBuilder sb = new StringBuilder("select uc.USC_ID_C,
uc.USC_EXTERNALID_C, uc.USC_FULLNAME_C ");
        sb.append(" from T_USER_CONTACT uc");
```

```
        // Add search criterias
        List<String> criteriaList = new ArrayList<String>();
        criteriaList.add("uc.USC_DELETEDATE_D is null");
        if (criteria.getAppId() != null) {
            criteriaList.add("uc.USC_IDAPP_C = :appId");
            parameterMap.put("appId", criteria.getAppId());
        }
        if (criteria.getUserId() != null) {
            criteriaList.add("uc.USC_IDUSER_C = :userId");
            parameterMap.put("userId", criteria.getUserId());
        }
        if (!Strings.isNullOrEmpty(criteria.getQuery())) {
            criteriaList.add("lower(uc.USC_FULLNAME_C) like
:fullName");
            parameterMap.put("fullName", "%" +
criteria.getQuery().toLowerCase() + "%");
        }

        if (!criteriaList.isEmpty()) {
            sb.append(" where ");
            sb.append(Joiner.on(" and ").join(criteriaList));
        }

        sb.append(" order by uc.USC_FULLNAME_C, uc.USC_ID_C");

        // Perform the search
        QueryParam queryParam = new QueryParam(sb.toString(),
parameterMap);
        List<Object[]> l =
PaginatedLists.executePaginatedQuery(paginatedList, queryParam);

        // Build results
        List<UserContactDto> userContactDtoList = new
ArrayList<UserContactDto>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDto.setFullName((String) o[i++]);
            userContactDtoList.add(userContactDto);
        }
        paginatedList.setResultList(userContactDtoList);
    }
}
```

3. `UserAppCreatedAsyncListener.java`:

```
/**
 * User app created listener.
 *
```

```
 * @author jtremeaux
 */
public class UserAppCreatedAsyncListener {
    /**
     * Logger.
     */
    private static final Logger log =
LoggerFactory.getLogger(UserAppCreatedAsyncListener.class);

    /**
     * Process event.
     *
     * @param userAppCreatedEvent Event
     */
    @Subscribe
    public void onUserCreated(final UserAppCreatedEvent
userAppCreatedEvent) {
        if (log.isInfoEnabled()) {
            log.info("UserApp created event: " +
userAppCreatedEvent.toString());
        }

        final UserApp userApp = userAppCreatedEvent.getUserApp();

        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                try {
                    AppId appId = AppId.valueOf(userApp.getAppId());
                    switch (appId) {
                    case FACEBOOK:
                        // Synchronize friends contact
                        final FacebookService facebookService =
AppContext.getInstance().getFacebookService();
                        String facebookAccessToken =
userApp.getAccessToken();

facebookService.synchronizeContact(facebookAccessToken,
userApp.getUserId());
                        break;
                    }
                } catch (Exception e) {
                    if (log.isErrorEnabled()) {
                        log.error("Error synchronizing App contacts",
userAppCreatedEvent, e);
                    }
                }
            }
        });
    }
}
```

4. `ConnectResource.java`:

```
@Path("/connect")
public class ConnectResource extends BaseResource {

    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        //implementation
    }

    @POST
    @Path("{id: [a-z]+}/add")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(
            @PathParam("id") String appIdString,
            @FormParam("access_token") String accessToken) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Get application to add
        AppId appId = getAppId(appIdString);

        switch (appId) {
        case FACEBOOK:
            // Specific application logic
            //implementation
            break;
        }

        // Always return OK
        //implementation
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}/remove")
    @Produces(MediaType.APPLICATION_JSON)
    public Response remove(@PathParam("id") String appIdString) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Get application to remove
        AppId appId = getAppId(appIdString);

        // Delete user app for this application
        //implementation

        // Always return OK
        //implementation
```

```
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}/update")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(
            @PathParam("id") String appIdString,
            @FormParam("sharing") boolean sharing) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Get application to update
        AppId appId = getAppId(appIdString);

        // Check if the user is connected to this application
        //implementation

        // Always return OK
        //implementation
    }

    @GET
    @Path("{id: [a-z0-9\\-]+}/contact/list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response contactList(
            @PathParam("id") String appIdString,
            @QueryParam("query") String query,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) throws JSONException
{
        if (!authenticate()) {
            //implementation
        }

        // Get application
        AppId appId = getAppId(appIdString);

        // Check if the user is connected to the application
        //implementation

        // Search contacts
        //implementation

        return Response.ok().entity(response).build();
    }

    private AppId getAppId(String appIdString) throws JSONException {
        AppId appId = null;
        try {
            appId = AppId.valueOf(StringUtils.upperCase(appIdString));
        } catch (Exception e) {
            throw new ClientException("AppNotFound", "Application not
```

```
found: " + appIdString);
        }
        return appId;
    }
}
```

Apply Manual Refactoring

Refactored code:

1. `UserAppDao.java`:

```
/**
 * User app DAO.
 *
 * @author jtremeaux
 */
public class UserAppDao {
    /**
     * Create a new connection.
     *
     * @param userApp User app
     * @return ID
     */
    public String create(UserApp userApp) {
        // Implementation remains same
        return userApp.getId();
    }

    /**
     * Delete a connection.
     *
     * @param id User app ID
     */
    public void delete(String id) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();

        // Get the user app
        Query q = em.createQuery("select ua from UserApp ua where
ua.id = :id and ua.deleteDate is null");
        q.setParameter("id", id);
        UserApp userAppFromDb = (UserApp) q.getSingleResult();

        // Delete the user app
        userAppFromDb.setDeleteDate(new Date());
    }

    /**
     * Deletes connection linked to a user and an application.
     *
     * @param userId User ID
     */
    public void deleteByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("update UserApp ua set ua.deleteDate
= :deleteDate where ua.userId = :userId and ua.deleteDate is null");
        q.setParameter("deleteDate", new Date());
```

```
            q.setParameter("userId", userId);
            q.executeUpdate();
        }

        /**
         * Search and returns a non-deleted connection.
         *
         * @param id User app ID
         * @return User app
         */
        public UserApp getActiveById(String id) {
            // Implementation remains same
            try {
                // Process Query
                return (UserApp) q.getSingleResult();
            } catch (NoResultException e) {
                return null;
            }
        }

        /**
         * Search and returns a non-deleted connection by user
         *
         * @param userId User ID
         * @return User app
         */
        public UserApp getActiveByUserId(String userId) {
            EntityManager em =
    ThreadLocalContext.get().getEntityManager();
            StringBuilder sb = new StringBuilder("select distinct ua from
    UserApp ua");
            sb.append(" where ua.userId = :userId ");
            sb.append(" and ua.deleteDate is null ");
            sb.append(" order by ua.createDate desc ");
            Query q = em.createQuery(sb.toString());
            try {
                q.setParameter("userId", userId);
                return (UserApp) q.getSingleResult();
            } catch (NoResultException e) {
                return null;
            }
        }

        /**
         * Search and returns application list with connection status.
         *
         * @param userId User ID
         * @return User app list
         */
        public List<UserAppDto> findByUserId(String userId) {
            List<UserAppDto> userAppDtoList = new ArrayList<UserAppDto>();
            // Implementation remains same
            return userAppDtoList;
        }
```

```
    /**
     * Search and returns user's connected applications.
     *
     * @param userId User ID
     * @return User app list
     */
    @SuppressWarnings("unchecked")
    public List<UserAppDto> findConnectedByUserId(String userId) {
        List<UserAppDto> userAppDtoList = new ArrayList<UserAppDto>();
        // Implementation remains same
        return userAppDtoList;
    }

    /**
     * Updates a connection.
     *
     * @param userApp User app
     * @return Updated user app
     */
    public UserApp update(UserApp userApp) {
        //Implementation remains same
        return userApp;
    }
}
```

2. `UserContactDao.java`:

```
/**
 * User's contact DAO.
 *
 * This class focuses on managing user contacts independently of any
specific application context.
 *
 * @author jtremeaux
 */
public class UserContactDao {
    /**
     * Create a new contact.
     *
     * @param userContact UserContact
     * @return ID
     */
    public String create(UserContact userContact) {
        //Implementation remains same
        return userContact.getId()

    }

    /**
     * Returns users' contact.
```

```
     *
     * @param userId User ID
     * @return Users' contact
     */
    @SuppressWarnings("unchecked")
    public List<UserContactDto> findByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select uc.USC_ID_C,
uc.USC_EXTERNALID_C ");
        sb.append(" from T_USER_CONTACT uc");
        sb.append(" where uc.USC_IDUSER_C = :userId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userId", userId);
        List<Object[]> l = q.getResultList();
        List<UserContactDto> userContactDtoList = new
ArrayList<UserContactDto>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDtoList.add(userContactDto);
        }

        return userContactDtoList;
    }

    /**
     * Updates last check user contact date.
     *
     * @param userId User ID
     */
    public void updateByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("update T_USER_CONTACT
uc");
        sb.append(" set uc.USC_UPDATEDATE_D = :updateDate ");
        sb.append(" where uc.USC_IDUSER_C = :userId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("updateDate", new Date());
        q.setParameter("userId", userId);
        q.executeUpdate();
    }

    /**
     * Delete a contact.
     *
     * @param id User contact ID
     */
    public void delete(String id) {
```

```
        // Implementation remains same
    }

    /**
     * Search user's contacts based on criteria.
     *
     * @param criteria Search criteria
     * @param paginatedList Paginated list (filled by side effect)
     */
    public void findByCriteria(PaginatedList<UserContactDto>
paginatedList, UserContactCriteria criteria) {
        // The implementation of this method is specific to the
refactoring effort and addresses the removal of app-specific contexts.
        // Therefore, its detailed implementation is kept as an
example of adapting search functionality to focus on user-centric
criteria.
        Map<String, Object> parameterMap = new HashMap<>();

        StringBuilder sb = new StringBuilder("SELECT uc.USC_ID_C,
uc.USC_EXTERNALID_C, uc.USC_FULLNAME_C ");
        sb.append("FROM T_USER_CONTACT uc ");

        List<String> criteriaList = new ArrayList<>();
        criteriaList.add("uc.USC_DELETEDATE_D IS NULL");
        if (criteria.getUserId() != null) {
            criteriaList.add("uc.USC_IDUSER_C = :userId");
            parameterMap.put("userId", criteria.getUserId());
        }
        if (!Strings.isNullOrEmpty(criteria.getQuery())) {
            criteriaList.add("LOWER(uc.USC_FULLNAME_C) LIKE
:fullName");
            parameterMap.put("fullName", "%" +
criteria.getQuery().toLowerCase() + "%");
        }

        if (!criteriaList.isEmpty()) {
            sb.append("WHERE ");
            sb.append(String.join(" AND ", criteriaList));
        }

        sb.append(" ORDER BY uc.USC_FULLNAME_C, uc.USC_ID_C");

        QueryParam queryParam = new QueryParam(sb.toString(),
parameterMap);
        List<Object[]> l =
PaginatedLists.executePaginatedQuery(paginatedList, queryParam);

        List<UserContactDto> userContactDtoList = new ArrayList<>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDto.setFullName((String) o[i]);
```

```
            userContactDtoList.add(userContactDto);
        }
        paginatedList.setResultList(userContactDtoList);
    }
}
```

3. `UserAppCreatedAsyncListener.java`:

```
/**
 * User app created listener.
 *
 * @author jtremeaux
 */
public class UserAppCreatedAsyncListener {
    /**
     * Logger.
     */
    private static final Logger log =
LoggerFactory.getLogger(UserAppCreatedAsyncListener.class);

    /**
     * Process event.
     *
     * @param userAppCreatedEvent Event
     */
    @Subscribe
    public void onUserCreated(final UserAppCreatedEvent
userAppCreatedEvent) {
        if (log.isInfoEnabled()) {
            log.info("UserApp created event: " +
userAppCreatedEvent.toString());
        }

        final UserApp userApp = userAppCreatedEvent.getUserApp();

        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                try {

                    // Synchronize friends contact
                    final FacebookService facebookService =
AppContext.getInstance().getFacebookService();
                    String facebookAccessToken =
userApp.getAccessToken();

facebookService.synchronizeContact(facebookAccessToken,
userApp.getUserId());

                } catch (Exception e) {
```

```
                    if (log.isErrorEnabled()) {
                        log.error("Error synchronizing App contacts",
userAppCreatedEvent, e);
                    }
                }
            }
        });
    }
}
```

4. ConnectResource.java:

```
/**
 * Connected application REST resource.
 *
 * @author jtremeaux
 */
@Path("/connect")
public class ConnectResource extends BaseResource {
    /**
     * Returns current user's connected applications.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Search connected applications
        UserAppDao userAppDao = new UserAppDao();
        List<UserAppDto> userAppList =
userAppDao.findByUserId(principal.getId());

        List<JSONObject> items = new ArrayList<JSONObject>();
        for (UserAppDto userAppDto : userAppList) {
            JSONObject userApp = new JSONObject();
            userApp.put("connected", userAppDto.getId() != null &&
userAppDto.getAccessToken() != null);
            userApp.put("username", userAppDto.getUsername());
            userApp.put("sharing", userAppDto.isSharing());
            items.add(userApp);
        }

        JSONObject response = new JSONObject();
        response.put("apps", items);
        return Response.ok().entity(response).build();
    }
```

```
    /**
     * Add a connected application.
     *
     * @param authToken OAuth authorization token
     * @return Response
     * @throws JSONException
     */
    @POST
    @Path("{id: [a-z]+}/add")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(@FormParam("access_token") String accessToken)
throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Validate input data
        accessToken =
ValidationUtil.validateStringNotBlank(accessToken, "access_token");

        UserAppDao userAppDao = new UserAppDao();
        UserApp userApp = null;
        // Delete old connection to this application
        userAppDao.deleteByUserId(principal.getId());

        // Exchange the short lived token (2h) for a long lived one
(60j)
        final FacebookService facebookService =
AppContext.getInstance().getFacebookService();
        String extendedAccessToken = null;
        try {
            extendedAccessToken =
facebookService.getExtendedAccessToken(accessToken);
        } catch (AuthenticationException e) {
            throw new ClientException("InvalidAuthenticationToken",
"Error validating authentication token", e);
        }

        // Check permissions
        try {
            facebookService.validatePermission(extendedAccessToken);
        } catch (PermissionException e) {
            throw new ClientException("PermissionNotFound",
e.getMessage(), e);
        }

        // Create the connection to the application
        userApp = new UserApp();
        userApp.setAccessToken(extendedAccessToken);
        userApp.setUserId(principal.getId());
        userApp.setSharing(true);

        // Get user's personnal informations
```

```
        facebookService.updateUserData(extendedAccessToken, userApp);

        userAppDao.create(userApp);



        // Raise a user app created event
        UserAppCreatedEvent userAppCreatedEvent = new
UserAppCreatedEvent();
        userAppCreatedEvent.setUserApp(userApp);

AppContext.getInstance().getAsyncEventBus().post(userAppCreatedEvent);

        // Always return OK
        JSONObject response = new JSONObject();
        response.put("status", "ok");
        return Response.ok().entity(response).build();
    }

    /**
     * Remove a connected application.
     *
     * @param appIdString App ID
     * @return Response
     * @throws JSONException
     */
    @POST
    @Path("{id: [a-z0-9\\-]+}/remove")
    @Produces(MediaType.APPLICATION_JSON)
    public Response remove(
            @PathParam("id") String appIdString) throws JSONException
{
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Delete user app for this application
        UserAppDao userAppDao = new UserAppDao();
        userAppDao.deleteByUserId(principal.getId());

        // Always return OK
        JSONObject response = new JSONObject();
        response.put("status", "ok");
        return Response.ok().entity(response).build();
    }

    /**
     * Updates connected application.
     *
     * @param appIdString App ID
     * @param sharing If true, share on this application
     * @return Response
     * @throws JSONException
     */
```

```
    @POST
    @Path("{id: [a-z0-9\\-]+}/update")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(
            @PathParam("id") String appIdString,
            @FormParam("sharing") boolean sharing) throws
JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Check if the user is connected to this application
        UserAppDao userAppDao = new UserAppDao();
        UserApp userApp =
userAppDao.getActiveByUserId(principal.getId());
        if (userApp == null) {
            throw new ClientException("AppNotConnected",
java.text.MessageFormat.format("You are not connected to the app {0}",
appIdString));
        }

        // Update the user app
        userApp.setSharing(sharing);
        userAppDao.update(userApp);

        // Always return OK
        JSONObject response = new JSONObject();
        response.put("status", "ok");
        return Response.ok().entity(response).build();
    }

    /**
     * Returns contact list on a connected application.
     *
     * @param appIdString App ID
     * @param limit Page limit
     * @param offset Page offset
     * @return Response
     * @throws JSONException
     */
    @GET
    @Path("{id: [a-z0-9\\-]+}/contact/list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response contactList(
            @PathParam("id") String appIdString,
            @QueryParam("query") String query,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) throws JSONException
{
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Check if the user is connected to the application
```

```
        UserAppDao userAppDao = new UserAppDao();
        UserApp userApp =
userAppDao.getActiveByUserId(principal.getId());
        if (userApp == null) {
            throw new ClientException("AppNotConnected",
java.text.MessageFormat.format("You are not connected to the app {0}",
appIdString));
        }

        JSONObject response = new JSONObject();
        List<JSONObject> contacts = new ArrayList<JSONObject>();

        // Search contacts
        PaginatedList<UserContactDto> paginatedList =
PaginatedLists.create(limit, offset);
        UserContactCriteria criteria = new UserContactCriteria();
        criteria.setUserId(principal.getId());
        criteria.setQuery(query);
        UserContactDao userContactDao = new UserContactDao();
        userContactDao.findByCriteria(paginatedList, criteria);
        for (UserContactDto userContactDto :
paginatedList.getResultList()) {
            JSONObject userContact = new JSONObject();
            userContact.put("id", userContactDto.getId());
            userContact.put("external_id",
userContactDto.getExternalId());
            userContact.put("full_name",
userContactDto.getFullName());
            contacts.add(userContact);
        }

        response.put("total", paginatedList.getResultCount());
        response.put("contacts", contacts);
        return Response.ok().entity(response).build();
    }

    /**
     * Get application ID.
     *
     * @param appIdString Application ID (string format)
     * @return App ID
     * @throws JSONException
     */
    private AppId getAppId(String appIdString) throws JSONException {
        AppId appId = null;
        try {
            appId = AppId.valueOf(StringUtils.upperCase(appIdString));
        } catch (Exception e) {
            throw new ClientException("AppNotFound",
MessageFormat.format("Application not found: {0}", appIdString));
        }
        return appId;
    }
}
```

LLM Refactoring

1. `UserAppDao.java`:

```
/**
 * UserAppDao manages operations on the UserApp entities, ensuring no
direct dependency
 * on the App entity itself. Operations involving app IDs use the
identifier stored within
 * the UserApp entity, maintaining a clear separation from the App
entity.
 */
public class UserAppDao {

    /**
     * Create a new UserApp entity.
     *
     * @param userApp User app to be created.
     * @return The ID of the created UserApp entity.
     */
    public String create(UserApp userApp) {
        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        // Perform the create operation
        return userApp.getId();
    }

    /**
     * Deletes a UserApp entity by its ID.
     *
     * @param id ID of the UserApp to delete.
     */
    public void delete(String id) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        UserApp userApp = em.find(UserApp.class, id);
        if (userApp != null) {
            userApp.setDeleteDate(new Date());
            em.merge(userApp);
        }
    }

    /**
     * Deletes UserApp entities linked to a specific user ID and app
ID.
     * Note: This method operates on the relationship between UserApp
and the
     * identifier of an App, not the App entity directly.
     *
     * @param userId ID of the user.
     * @param appId ID of the app linked to the user.
     */
```

```
    public void deleteByUserIdAndAppId(String userId, String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("UPDATE UserApp ua SET ua.deleteDate
= :deleteDate WHERE ua.userId = :userId AND ua.appId = :appId AND
ua.deleteDate IS NULL");
        q.setParameter("deleteDate", new Date());
        q.setParameter("userId", userId);
        q.setParameter("appId", appId);
        q.executeUpdate();
    }

    /**
     * Retrieves an active (non-deleted) UserApp by its ID.
     *
     * @param id ID of the UserApp.
     * @return The found UserApp, or null if not found.
     */
    public UserApp getActiveById(String id) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        try {
            return em.createQuery("SELECT ua FROM UserApp ua WHERE
ua.id = :id AND ua.deleteDate IS NULL", UserApp.class)
                    .setParameter("id", id)
                    .getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Retrieves active (non-deleted) UserApp entities based solely on
the user ID.
     * This approach abstracts away from any direct linkage to
specific applications,
     * focusing instead on the user's engagement within the system.
     *
     * @param userId The unique identifier of the user.
     * @return List of active UserApp entities for the user, or an
empty list if none are found.
     */
    public List<UserApp> getActiveByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        List<UserApp> userApps = em.createQuery("SELECT ua FROM
UserApp ua WHERE ua.userId = :userId AND ua.deleteDate IS NULL ORDER
BY ua.createDate DESC", UserApp.class)
                                   .setParameter("userId", userId)
                                   .getResultList();
        return userApps;
    }

    /**
```

```
     * Search and returns application list with connection status.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findByUserId(String userId) {
        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns user's connected applications.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findConnectedByUserId(String userId) {
        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns all active UserApp entities as DTOs.
     * The parameter appId is taken but not used.
     *
     * @return List of UserAppDto without filtering by appId, as appId
no longer exists.
     */
    @SuppressWarnings("unchecked")
    public List<UserAppDto> findAllActiveUserApps(String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("SELECT ua.USA_ID_C,
ua.USA_IDUSER_C, ua.USA_ACCESSTOKEN_C, ua.USA_USERNAME_C,
ua.USA_SHARING_B ");
        sb.append("FROM T_USER_APP ua ");
        sb.append("WHERE ua.USA_DELETEDATE_D IS NULL "); // Removed
the appId filter
        sb.append("ORDER BY ua.USA_CREATEDATE_D ");
        Query q = em.createNativeQuery(sb.toString());

        List<Object[]> resultList = q.getResultList();
        List<UserAppDto> userAppDtoList = new ArrayList<>();
        for (Object[] record : resultList) {
            int i = 0;
            UserAppDto userAppDto = new UserAppDto();
            userAppDto.setId((String) record[i++]);
            userAppDto.setUserId((String) record[i++]);
            userAppDto.setAppId((String) record[i++]); // Consider if
this is still relevant
            userAppDto.setAccessToken((String) record[i++]);
            userAppDto.setUsername((String) record[i++]);
```

```
            Boolean sharing = (Boolean) record[i];
            userAppDto.setSharing(sharing != null && sharing);
            userAppDtoList.add(userAppDto);
        }

        return userAppDtoList;
    }

    /**
     * Updates a connection.
     *
     * @param userApp User app
     * @return Updated user app
     */
    public UserApp update(UserApp userApp) {

        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        // Perform the update operation
        return userApp;
    }


    /* The methods for findByUserId, findConnectedByUserId, and update
    remain conceptually unchanged from the original, as they do not
directly interact
    with the App entity but rather work with identifiers or attributes
within UserApp. */
}
```

2. UserContactDao.java:

```
/**
 * User's contact DAO.
 *
 * This class has been refactored to remove any dependencies on
specific application contexts,
 * focusing purely on user contact management.
 *
 * @author jtremeaux
 */
public class UserContactDao {
    /**
     * Create a new contact.
     *
     * @param userContact UserContact
     * @return ID
     */
    public String create(UserContact userContact) {
        /* Implementation remains same because the method is not
directly connected to App Entity */
```

```
    }

    /**
     * Returns all contacts for a given user.
     *
     * @param userId User ID
     * @return List of UserContactDto for the specified user
     */
    @SuppressWarnings("unchecked")
    public List<UserContactDto> findByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("SELECT uc.USC_ID_C,
uc.USC_EXTERNALID_C ");
        sb.append("FROM T_USER_CONTACT uc ");
        sb.append("WHERE uc.USC_IDUSER_C = :userId ");
        sb.append("AND uc.USC_DELETEDATE_D IS NULL");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userId", userId);
        List<Object[]> result = q.getResultList();
        List<UserContactDto> userContactDtoList = new ArrayList<>();
        for (Object[] record : result) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) record[i++]);
            userContactDto.setExternalId((String) record[i++]);
            userContactDtoList.add(userContactDto);
        }

        return userContactDtoList;
    }

    /**
     * Update last check date for all contacts of a user.
     *
     * @param userId User ID
     */
    public void updateByUserId(String userId) {
        /* Implementation remains same because the method is not
directly connected to App Entity */
    }

    /**
     * Delete a contact by its ID.
     *
     * @param id User contact ID
     */
    public void delete(String id) {
        /* Implementation remains same because the method is not
directly connected to App Entity */
    }

    /**
     * Search user's contacts based on criteria.
```

```
     *
     * Note: The search criteria no longer includes appId to ensure
independence from any app-specific context.
     *
     * @param paginatedList Paginated list (filled by side effect)
     * @param criteria Search criteria
     */
    public void findByCriteria(PaginatedList<UserContactDto>
paginatedList, UserContactCriteria criteria) {
        // The implementation details for findByCriteria are important
for understanding how the method adapts to the removal of appId,
        // so we will keep this method's implementation visible as it
directly reflects the refactoring effort.
        Map<String, Object> parameterMap = new HashMap<>();

        StringBuilder sb = new StringBuilder("SELECT uc.USC_ID_C,
uc.USC_EXTERNALID_C, uc.USC_FULLNAME_C ");
        sb.append("FROM T_USER_CONTACT uc ");

        List<String> criteriaList = new ArrayList<>();
        criteriaList.add("uc.USC_DELETEDATE_D IS NULL");
        if (criteria.getUserId() != null) {
            criteriaList.add("uc.USC_IDUSER_C = :userId");
            parameterMap.put("userId", criteria.getUserId());
        }
        if (!Strings.isNullOrEmpty(criteria.getQuery())) {
            criteriaList.add("LOWER(uc.USC_FULLNAME_C) LIKE
:fullName");
            parameterMap.put("fullName", "%" +
criteria.getQuery().toLowerCase() + "%");
        }

        if (!criteriaList.isEmpty()) {
            sb.append("WHERE ");
            sb.append(String.join(" AND ", criteriaList));
        }

        sb.append(" ORDER BY uc.USC_FULLNAME_C, uc.USC_ID_C");

        Query q = em.createNativeQuery(sb.toString());
        for (Map.Entry<String, Object> entry :
parameterMap.entrySet()) {
            q.setParameter(entry.getKey(), entry.getValue());
        }

        List<Object[]> results = q.getResultList();
        List<UserContactDto> userContactDtoList = new ArrayList<>();
        for (Object[] result : results) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) result[i++]);
            userContactDto.setExternalId((String) result[i++]);
            if (result.length > i) { // Check if fullName is part of
the result set
```

```
                userContactDto.setFullName((String) result[i]);
            }
            userContactDtoList.add(userContactDto);
        }
        paginatedList.setResultList(userContactDtoList);
    }

    // Additional methods if needed for UserContactDao operations,
focusing solely on user-centric data management.
}
```

3. `UserAppCreatedAsyncListener.java`:

```
/**
 * Listener for user app creation events.
 * This version abstracts away from specific app handling, focusing on
generic event processing.
 *
 * @author jtremeaux
 */
public class UserAppCreatedAsyncListener {
    private static final Logger log =
LoggerFactory.getLogger(UserAppCreatedAsyncListener.class);

    /**
     * Process event.
     *
     * @param userAppCreatedEvent Event
     */
    @Subscribe
    public void onUserCreated(final UserAppCreatedEvent
userAppCreatedEvent) {
        if (log.isInfoEnabled()) {
            log.info("UserApp created event: " +
userAppCreatedEvent.toString());
        }

        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                try {
                    // Handle user app creation generically without
depending on specific app logic

handleUserAppCreation(userAppCreatedEvent.getUserApp());
                } catch (Exception e) {
                    if (log.isErrorEnabled()) {
                        log.error("Error handling user app creation",
e);
                    }
                }
            }
```

```
        });
    }

    /**
     * Handles the creation of a user app in a generic manner.
     * This method can be expanded or modified to introduce specific
logic based on app type
     * without directly coupling to any specific app's API or logic
here.
     *
     * @param userApp The user app involved in the creation event.
     */
    private void handleUserAppCreation(UserApp userApp) {
        // Implement generic handling logic here
        // For example, notify other parts of the system, log the
event, update related records, etc.
        log.info("Handling creation of user app: " + userApp.getId());

        // Optionally, delegate to a service that can handle specific
app types internally

AppServiceHandler.getInstance().processUserAppCreated(userApp);
    }
}
```

4. `ConnectResource.java`:

```
@Path("/connect")
public class ConnectResource extends BaseResource {

    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        //implementation
    }

    @POST
    @Path("/add")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(@FormParam("access_token") String accessToken)
throws JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Generic application addition logic
        // The specific app handling logic is removed.
        // Implement generic add application logic here
        //implementation

        // Always return OK
```

```
        //implementation
    }

    @POST
    @Path("/remove")
    @Produces(MediaType.APPLICATION_JSON)
    public Response remove(@FormParam("appIdentifier") String
appIdentifier) throws JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Generic application removal logic
        // The specific app handling logic is removed.
        // Implement generic remove application logic here
        //implementation

        // Always return OK
        //implementation
    }

    @POST
    @Path("/update")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(
            @FormParam("appIdentifier") String appIdentifier,
            @FormParam("sharing") boolean sharing) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Generic application update logic
        // The specific app handling logic is removed.
        // Implement generic update application logic here
        //implementation

        // Always return OK
        //implementation
    }

    @GET
    @Path("/contact/list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response contactList(
            @QueryParam("query") String query,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) throws JSONException
{
        if (!authenticate()) {
            //implementation
        }

        // Generic contact list fetching
```

```
        // The specific app handling logic is removed.
        // Implement generic contact list fetching logic here
        //implementation

        return Response.ok().entity(response).build();
    }

    // Removed the getAppId method as it's no longer needed in this
generic approach
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- Manual refactoring has introduced a method deleteByUserId which simplifies deletion by user ID, avoiding direct references to app IDs.
- It has also streamlined the getActiveByUserId method to focus solely on user IDs, removing app ID considerations.
- The manual version seems to simplify and directly address the operations without considering app IDs, aligning with a focus on user-centric data management.

LLM-generated Refactoring:

- The LLM-refactored code maintains methods that originally referenced app IDs but modifies their implementations to ignore or bypass these references, reflecting an attempt to abstract away from the App entity while still keeping the method signatures and general structure intact.
- It includes a method findAllActiveUserApps that attempts to generalize fetching of user apps without filtering by app ID, even though it retains an unused appId parameter, indicating an incomplete abstraction from the app ID concept.

Evaluation of Strengths and Weaknesses:

Manual Refactoring:

Strengths:

- Simplification and Focus: Concentrates on user-related operations, making the DAO more straightforward and easier to maintain.
- Reduction of Irrelevant Parameters: Eliminates unnecessary parameters (e.g., appId), which could simplify future development and reduce confusion.

Weaknesses:

- Potential Loss of Functionality: By removing the ability to filter by app ID, it might limit the flexibility of the DAO in scenarios where application-specific data retrieval is necessary.
- Lack of Generalization: Does not introduce generalized methods that could accommodate future requirements without reintroducing app-specific logic.

LLM-generated Refactoring:

Strengths:

- Attempt at Generalization: Tries to create more generalized methods that could potentially accommodate a wider range of use cases.
- Preservation of Method Signatures: Keeps method signatures that reference app IDs, which might ease the transition for existing codebases relying on these methods, even if the actual dependency on app IDs is minimized.

Weaknesses:

- Incomplete Abstraction: Retains unused parameters and method signatures that suggest app-specific operations, which could lead to confusion or the impression of incomplete refactoring.
- Complexity in Understanding: The presence of app ID references, even in a minimized role, may complicate understanding for new developers or those unfamiliar with the historical context of the refactoring effort.

Scenarios:

LLMs Excel:

- In situations where a quick refactoring is needed to prototype or migrate away from certain dependencies while maintaining a resemblance of the original structure.
- When there's a need to balance between removing specific dependencies and retaining API compatibility for a transitional period.

Manual Intervention is Preferable:

- For targeted, purpose-driven refactoring efforts that aim to simplify and clarify the codebase with a clear understanding of future requirements and constraints.
- When the refactoring aims not just to remove dependencies but also to optimize and streamline the codebase for maintainability and scalability.
- Both approaches have their merits, depending on the goals and constraints of the refactoring effort. The choice between manual and LLM-generated refactoring should be guided by the specific needs of the project, including considerations of maintainability, flexibility, and the ease of future development.

Feature Envy : TagDao.java (books-core com.sismics.books.core.dao.jpa)

Original Code:

```
/**
 * Tag DAO.
 *
 * @author bgamard
 */
public class TagDao {
    /**
     * Gets a tag by its ID.
     *
     * @param id Tag ID
     * @return Tag
     */
    public Tag getById(String id) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        try {
            return em.find(Tag.class, id);
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Returns the list of all tags.
     *
     * @return List of tags
     */
    @SuppressWarnings("unchecked")
    public List<Tag> getByUserId(String userId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.userId =
:userId and t.deleteDate is null order by t.name");
        q.setParameter("userId", userId);
        return q.getResultList();
    }

    /**
     * Update tags on a user book.
     *
     * @param userBookId
     * @param tagIdSet
     */
    public void updateTagList(String userBookId, Set<String> tagIdSet) {
        // Delete old tag links
        EntityManager em = ThreadLocalContext.get().getEntityManager();
```

```
        Query q = em.createQuery("delete UserBookTag bt where bt.userBookId
= :userBookId");
        q.setParameter("userBookId", userBookId);
        q.executeUpdate();

        // Create new tag links
        for (String tagId : tagIdSet) {
            UserBookTag userBookTag = new UserBookTag();
            userBookTag.setId(UUID.randomUUID().toString());
            userBookTag.setUserBookId(userBookId);
            userBookTag.setTagId(tagId);
            em.persist(userBookTag);
        }

    }

    /**
     * Returns tag list on a user book.
     * @param userBookId
     * @return
     */
    @SuppressWarnings("unchecked")
    public List<TagDto> getByUserBookId(String userBookId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select t.TAG_ID_C,
t.TAG_NAME_C, t.TAG_COLOR_C from T_USER_BOOK_TAG bt ");
        sb.append(" join T_TAG t on t.TAG_ID_C = bt.BOT_IDTAG_C ");
        sb.append(" where bt.BOT_IDUSERBOOK_C = :userBookId and
t.TAG_DELETEDATE_D is null ");
        sb.append(" order by t.TAG_NAME_C ");

        // Perform the query
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userBookId", userBookId);
        List<Object[]> l = q.getResultList();

        // Assemble results
        List<TagDto> tagDtoList = new ArrayList<TagDto>();
        for (Object[] o : l) {
            int i = 0;
            TagDto tagDto = new TagDto();
            tagDto.setId((String) o[i++]);
            tagDto.setName((String) o[i++]);
            tagDto.setColor((String) o[i++]);
            tagDtoList.add(tagDto);
        }
        return tagDtoList;
    }

    /**
     * Creates a new tag.
     *
     * @param tag Tag
     * @return New ID
```

```
     * @throws Exception
     */
    public String create(Tag tag) {
        // Create the UUID
        tag.setId(UUID.randomUUID().toString());

        // Create the tag
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        tag.setCreateDate(new Date());
        em.persist(tag);

        return tag.getId();
    }

    /**
     * Returns a tag by name.
     * @param userId User ID
     * @param name Name
     * @return Tag
     */
    public Tag getByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.name = :name
and t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", name);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Returns a tag by ID.
     * @param userId User ID
     * @param tagId Tag ID
     * @return Tag
     */
    public Tag getByTagId(String userId, String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.id = :tagId
and t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("tagId", tagId);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Deletes a tag.
```

```
     *
     * @param tagId Tag ID
     */
    public void delete(String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();

        // Get the tag
        Query q = em.createQuery("select t from Tag t where t.id = :id and
t.deleteDate is null");
        q.setParameter("id", tagId);
        Tag tagDb = (Tag) q.getSingleResult();

        // Delete the tag
        Date dateNow = new Date();
        tagDb.setDeleteDate(dateNow);

        // Delete linked data
        q = em.createQuery("delete UserBookTag bt where bt.tagId =
:tagId");
        q.setParameter("tagId", tagId);
        q.executeUpdate();
    }

    /**
     * Search tags by name.
     *
     * @param name Tag name
     * @return List of found tags
     */
    @SuppressWarnings("unchecked")
    public List<Tag> findByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.name like
:name and t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", "%" + name + "%");
        return q.getResultList();
    }
}
```

Manually Refactored Code:

```
public class TagDao {
    private final EntityManager em;

    public TagDao() {
        this.em = ThreadLocalContext.get().getEntityManager();
    }

    public Tag getById(String id) {
        return findTagById(id);
    }

    private Tag findTagById(String id) {
        try {
            return em.find(Tag.class, id);
        } catch (NoResultException e) {
            return null;
        }
    }

    public List<Tag> getByUserId(String userId) {
        return findTagsByUserId(userId);
    }

    private List<Tag> findTagsByUserId(String userId) {
        Query q = em
                .createQuery("select t from Tag t where t.userId = :userId
and t.deleteDate is null order by t.name");
        q.setParameter("userId", userId);
        return q.getResultList();
    }

    public void updateTagList(String userBookId, Set<String> tagIds) {
        deleteOldTagLinks(userBookId);
        createNewTagLinks(userBookId, tagIds);
    }

    private void deleteOldTagLinks(String userBookId) {
        Query q = em.createQuery("delete UserBookTag bt where bt.userBookId
= :userBookId");
        q.setParameter("userBookId", userBookId);
        q.executeUpdate();
    }

    private void createNewTagLinks(String userBookId, Set<String> tagIds) {
        for (String tagId : tagIds) {
            UserBookTag userBookTag = new UserBookTag();
            userBookTag.setId(UUID.randomUUID().toString());
            userBookTag.setUserBookId(userBookId);
            userBookTag.setTagId(tagId);
```

```
            em.persist(userBookTag);
        }
    }

    public List<TagDto> getByUserBookId(String userBookId) {
        return assembleTagDtosFromUserBookId(userBookId);
    }

    private List<TagDto> assembleTagDtosFromUserBookId(String userBookId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select t.TAG_ID_C,
t.TAG_NAME_C, t.TAG_COLOR_C from T_USER_BOOK_TAG bt ");
        sb.append(" join T_TAG t on t.TAG_ID_C = bt.BOT_IDTAG_C ");
        sb.append(" where bt.BOT_IDUSERBOOK_C = :userBookId and
t.TAG_DELETEDATE_D is null ");
        sb.append(" order by t.TAG_NAME_C ");

        // Perform the query
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userBookId", userBookId);
        List<Object[]> l = q.getResultList();

        // Assemble results
        List<TagDto> tagDtoList = new ArrayList<TagDto>();
        for (Object[] o : l) {
            int i = 0;
            TagDto tagDto = new TagDto();
            tagDto.setId((String) o[i++]);
            tagDto.setName((String) o[i++]);
            tagDto.setColor((String) o[i++]);
            tagDtoList.add(tagDto);
        }
        return tagDtoList;
    }

    public String create(Tag tag) {
        tag.setId(UUID.randomUUID().toString());
        tag.setCreateDate(new Date());
        em.persist(tag);
        return tag.getId();
    }

    public Tag getByName(String userId, String name) {
        return findTagByNameAndUserId(userId, name);
    }

    private Tag findTagByNameAndUserId(String userId, String name) {
        Query q = em.createQuery(
                "select t from Tag t where t.name = :name and t.userId =
:userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", name);
        try {
            return (Tag) q.getSingleResult();
```

```
        } catch (NoResultException e) {
            return null;
        }
    }

    public Tag getByTagId(String userId, String tagId) {
        return findTagByIdAndUserId(userId, tagId);
    }

    private Tag findTagByIdAndUserId(String userId, String tagId) {
        Query q = em
                .createQuery("select t from Tag t where t.id = :tagId and
t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("tagId", tagId);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    public void delete(String tagId) {
        Tag tagDb = findTagById(tagId);
        if (tagDb != null) {
            Date dateNow = new Date();
            tagDb.setDeleteDate(dateNow);
            deleteLinkedData(tagId);
        }
    }

    private void deleteLinkedData(String tagId) {
        Query q = em.createQuery("delete UserBookTag bt where bt.tagId =
:tagId");
        q.setParameter("tagId", tagId);
        q.executeUpdate();
    }

    public List<Tag> findByName(String userId, String name) {
        return findTagsByNameAndUserId(userId, name);
    }

    private List<Tag> findTagsByNameAndUserId(String userId, String name) {
        Query q = em.createQuery(
                "select t from Tag t where t.name like :name and t.userId =
:userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", "%" + name + "%");
        return q.getResultList();
    }
}
```

LLM Given Code:

```
/**
 * Data Access Object (DAO) for managing Tag entities.
 * Handles CRUD operations and queries related to tags.
 * Author: bgamard
 */
public class TagDao {

    /**
     * Retrieves a tag by its ID.
     *
     * @param id Tag ID
     * @return Tag object if found, null otherwise
     */
    public Tag getById(String id) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        try {
            return em.find(Tag.class, id);
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Retrieves all tags associated with a specific user.
     *
     * @param userId User ID
     * @return List of tags associated with the user
     */
    @SuppressWarnings("unchecked")
    public List<Tag> getByUserId(String userId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.userId =
:userId AND t.deleteDate IS NULL ORDER BY t.name");
        q.setParameter("userId", userId);
        return q.getResultList();
    }

    /**
     * Updates tags associated with a user book.
     *
     * @param userBookId User book ID
     * @param tagIdSet Set of tag IDs to be associated with the user book
     */
    public void updateTagsForUserBook(String userBookId, Set<String>
tagIdSet) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        // Delete old tag links
        Query deleteQuery = em.createQuery("DELETE FROM UserBookTag bt
WHERE bt.userBookId = :userBookId");
        deleteQuery.setParameter("userBookId", userBookId);
```

```
            deleteQuery.executeUpdate();

            // Create new tag links
            for (String tagId : tagIdSet) {
                UserBookTag userBookTag = new UserBookTag();
                userBookTag.setId(UUID.randomUUID().toString());
                userBookTag.setUserBookId(userBookId);
                userBookTag.setTagId(tagId);
                em.persist(userBookTag);
            }
        }

        /**
         * Retrieves tag list associated with a user book.
         *
         * @param userBookId User book ID
         * @return List of TagDto objects representing tags associated with the
    user book
         */
        @SuppressWarnings("unchecked")
        public List<TagDto> getTagsByUserBookId(String userBookId) {
            EntityManager em = ThreadLocalContext.get().getEntityManager();
            StringBuilder sb = new StringBuilder("SELECT t.TAG_ID_C,
    t.TAG_NAME_C, t.TAG_COLOR_C FROM T_USER_BOOK_TAG bt ");
            sb.append("JOIN T_TAG t ON t.TAG_ID_C = bt.BOT_IDTAG_C ");
            sb.append("WHERE bt.BOT_IDUSERBOOK_C = :userBookId AND
    t.TAG_DELETEDATE_D IS NULL ");
            sb.append("ORDER BY t.TAG_NAME_C ");

            // Perform the query
            Query q = em.createNativeQuery(sb.toString());
            q.setParameter("userBookId", userBookId);
            List<Object[]> resultList = q.getResultList();

            // Assemble results
            List<TagDto> tagDtoList = new ArrayList<>();
            for (Object[] result : resultList) {
                int index = 0;
                TagDto tagDto = new TagDto();
                tagDto.setId((String) result[index++]);
                tagDto.setName((String) result[index++]);
                tagDto.setColor((String) result[index++]);
                tagDtoList.add(tagDto);
            }
            return tagDtoList;
        }

        /**
         * Creates a new tag.
         *
         * @param tag Tag object to be created
         * @return New ID of the created tag
         */
        public String createTag(Tag tag) {
```

```
        // Generate UUID
        tag.setId(UUID.randomUUID().toString());

        // Create the tag
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        tag.setCreateDate(new Date());
        em.persist(tag);

        return tag.getId();
    }

    /**
     * Retrieves a tag by its name.
     *
     * @param userId User ID
     * @param name Tag name
     * @return Tag object if found, null otherwise
     */
    public Tag getByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.name = :name
AND t.userId = :userId AND t.deleteDate IS NULL");
        q.setParameter("userId", userId);
        q.setParameter("name", name);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Retrieves a tag by its ID.
     *
     * @param userId User ID
     * @param tagId Tag ID
     * @return Tag object if found, null otherwise
     */
    public Tag getByTagId(String userId, String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.id = :tagId
AND t.userId = :userId AND t.deleteDate IS NULL");
        q.setParameter("userId", userId);
        q.setParameter("tagId", tagId);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Deletes a tag.
     *
```

```
     * @param tagId Tag ID
     */
    public void deleteTag(String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();

        // Get the tag
        Query selectQuery = em.createQuery("SELECT t FROM Tag t WHERE t.id
= :id AND t.deleteDate IS NULL");
        selectQuery.setParameter("id", tagId);
        Tag tagDb = (Tag) selectQuery.getSingleResult();

        // Delete the tag
        Date dateNow = new Date();
        tagDb.setDeleteDate(dateNow);

        // Delete linked data
        Query deleteQuery = em.createQuery("DELETE FROM UserBookTag bt
WHERE bt.tagId = :tagId");
        deleteQuery.setParameter("tagId", tagId);
        deleteQuery.executeUpdate();
    }

    /**
     * Searches tags by name.
     *
     * @param userId User ID
     * @param name Tag name (or part of it) to search for
     * @return List of found tags
     */
    @SuppressWarnings("unchecked")
    public List<Tag> findByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.name LIKE
:name AND t.userId = :userId AND t.deleteDate IS NULL");
        q.setParameter("userId", userId);
        q.setParameter("name", "%" + name + "%");
        return q.getResultList();
    }
}
```

ChatGPT Chat Link: https://chat.openai.com/share/3b246ad1-11ce-427a-9b11-b2d2b8e19158

Differences in Refactoring Approach

1. **Entity Manager Initialization:**
    - **Manual:** Uses a class-level `EntityManager`, reducing redundancy.
    - **LLM:** Keeps the original method-level `EntityManager` access.
2. **Method Structuring:**
    - **Manual:** Extracts private helper methods for common queries, improving code reuse.
    - **LLM:** Executes queries directly in public methods, simpler but less reusable.
3. **Code Readability and Maintainability:**
    - **Manual:** Enhances readability and maintainability by centralizing query logic.
    - **LLM:** Each method is self-contained, straightforward but potentially harder to maintain.
4. **Efficiency Considerations:**
    - **Both:** No significant efficiency difference, though manual might slightly edge out due to centralized `EntityManager` management.

Scenarios Where LLMs Excel vs. Manual Intervention

- **LLMs Excel:**
    - Quick boilerplate code generation.
    - Immediate, contextually relevant suggestions.
    - Maintaining existing coding style consistency.
- **Manual Intervention Preferable:**
    - Deep architectural refactoring.
    - Adherence to specific coding or architectural patterns.
    - Complex refactoring involving performance, readability, and maintainability trade-offs.

Conclusion

While manual refactoring is better for deep, maintainable changes, LLMs are great for quick fixes and maintaining style consistency. A combination of both methods often works best.

God Class: UserResource (books-web com.sismics.books.rest.resource)

Identified Code Snippet:

UserResource.java:

```
/**
 * User REST resources.
 *
 * @author jtremeaux
 */
@Path("/user")
public class UserResource extends BaseResource {
    /**
     * Creates a new user.
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register() throws JSONException {
            // Implementation
    }

    /**
     * Updates user informations.
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response update() Boolean firstConnection) throws JSONException
{

            // implementation
    }

    /**
     * Updates user informations.
     */
    @POST
    @Path("{username: [a-zA-Z0-9_]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update() throws JSONException {
            // implementation
    }

    /**
     * Checks if a username is available. Search only on active accounts.
     */
    @GET
```

```
    @Path("check_username")
    @Produces(MediaType.APPLICATION_JSON)
    public Response checkUsername() throws JSONException {

            // implementation

    }

    /**
     * This resource is used to authenticate the user and create a user
session.
     * The "session" is only used to identify the user, no other data is
stored in the session.
     */
    @POST
    @Path("login")
    @Produces(MediaType.APPLICATION_JSON)
    public Response login() throws JSONException {

            // implementation
    }

    /**
     * Logs out the user and deletes the active session.
     */
    @POST
    @Path("logout")
    @Produces(MediaType.APPLICATION_JSON)
    public Response logout() throws JSONException {
            // implementation

    }

    /**
     * Delete a user.
     */
    @DELETE
    @Produces(MediaType.APPLICATION_JSON)
    public Response delete() throws JSONException {
            // implementation
    }

    /**
     * Deletes a user.
     */
    @DELETE
    @Path("{username: [a-zA-Z0-9_]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response delete(@PathParam("username") String username) throws
JSONException {
            // implementation
    }

    /**
```

```
     * Returns the information about the connected user.
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
            // implementation
    }

    /**
     * Returns the information about a user.
     */
    @GET
    @Path("{username: [a-zA-Z0-9_]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response view(@PathParam("username") String username) throws
JSONException {
            // implementation
    }

    /**
     * Returns all active users.
     */
    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
            // implementation
    }

    /**
     * Returns all active sessions.
     */
    @GET
    @Path("session")
    @Produces(MediaType.APPLICATION_JSON)
    public Response session() throws JSONException {
            // implementation
    }

    /**
     * Deletes all active sessions except the one used for this request.
     */
    @DELETE
    @Path("session")
    @Produces(MediaType.APPLICATION_JSON)
    public Response deleteSession() throws JSONException {
            // implementation
    }
}
```

Apply Manual Refactoring

Refactored into:

1. `UserCrudResource.java`:

```
/**
 * User REST resources for CRUD Operations on User.
 *
 * @author jtremeaux
 */
@Path("/user")
public class UserCrudResource extends BaseResource {

    /**
     * Returns the information about the connected user.
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        // Implementation
    }

    /**
     * Creates a new user.
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register() throws JSONException {
        // Implementation
    }

    /**
     * Updates user informations.
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response update() throws JSONException {
        // Implementation
    }

    /**
     * Checks if a username is available. Search only on active
accounts.
     */
    @GET
    @Path("check_username")
    @Produces(MediaType.APPLICATION_JSON)
    public Response checkUsername() throws JSONException {
        // Implementation
    }

    /**
```

```
     * Delete a user.
     */
    @DELETE
    @Produces(MediaType.APPLICATION_JSON)
    public Response delete() throws JSONException {
        // Implementation
    }
}
```

2. UserListResource.java:

```
@Path("/user/list")
public class UserListResource extends BaseResource {

    /**
     * Returns all active users.
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        // Implementation
    }
}
```

3. UserLoginResource.java:

```
@Path("/user/login")
public class UserLoginResource extends BaseResource {
    /**
     * This resource is used to authenticate the user and create a
user session.
     * The "session" is only used to identify the user, no other data
is stored in the session.
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response login() throws JSONException {
        // Implementation
    }
}
```

4. UserLogoutResorce.java:

```
@Path("/user/logout")
```

```
public class UserLogoutResorce extends BaseResource {

    /**
     * Logs out the user and deletes the active session.
     */
    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response logout() throws JSONException {
        // Implementation
    }
}
```

5. UserSessionResource.java:

```
@Path("/user/session")
public class UserSessionResource extends BaseResource {

    /**
     * Returns all active sessions.
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response session() throws JSONException {
        // Implementation
    }

    /**
     * Deletes all active sessions except the one used for this
request.
     */
    @DELETE
    @Produces(MediaType.APPLICATION_JSON)
    public Response deleteSession() throws JSONException {
        // Implementation
    }
}
```

LLM Refactoring

```
@Path("/user")
public class UserResource extends BaseResource {
    private final UserManagementService userManagementService;

    public UserResource(UserManagementService userManagementService) {
        this.userManagementService = userManagementService;
    }

    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register() throws JSONException {
        return userManagementService.register();
    }

    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(Boolean firstConnection) throws JSONException {
        return userManagementService.update(firstConnection);
    }

    @POST
    @Path("{username: [a-zA-Z0-9_]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response updateUser() throws JSONException {
        return userManagementService.update();
    }

    @GET
    @Path("check_username")
    @Produces(MediaType.APPLICATION_JSON)
    public Response checkUsername() throws JSONException {
        return userManagementService.checkUsername();
    }

    @POST
    @Path("login")
    @Produces(MediaType.APPLICATION_JSON)
    public Response login() throws JSONException {
        return userManagementService.login();
    }

    @POST
    @Path("logout")
    @Produces(MediaType.APPLICATION_JSON)
    public Response logout() throws JSONException {
        return userManagementService.logout();
    }

    @DELETE
    @Produces(MediaType.APPLICATION_JSON)
```

```
    public Response delete() throws JSONException {
        return userManagementService.delete();
    }

    @DELETE
    @Path("{username: [a-zA-Z0-9_]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response deleteUser(@PathParam("username") String username)
throws JSONException {
        return userManagementService.deleteUser(username);
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        return userManagementService.info();
    }

    @GET
    @Path("{username: [a-zA-Z0-9_]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response viewUser(@PathParam("username") String username) throws
JSONException {
        return userManagementService.view(username);
    }

    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        return userManagementService.list();
    }
}

public class UserManagementService {
    public Response register() {
        // implementation
    }

    public Response update(Boolean firstConnection) {
        // implementation
    }

    public Response update() {
        // implementation
    }

    public Response checkUsername() {
        // implementation
    }

    public Response login() {
        // implementation
    }
```

```
    public Response logout() {
        // implementation
    }

    public Response delete() {
        // implementation
    }

    public Response deleteUser(String username) {
        // implementation
    }

    public Response info() {
        // implementation
    }

    public Response view(String username) {
        // implementation
    }

    public Response list() {
        // implementation
    }
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- The manual refactoring separates the functionalities into different resource classes (UserCrudResource, UserListResource, UserLoginResource, UserLogoutResource, UserSessionResource).
- Each resource class is responsible for handling a specific set of related operations (e.g., CRUD operations, login/logout, session management).
- The manual refactoring introduces a clear separation of concerns, making the codebase easier to understand and maintain.
- Each resource class has its own set of methods corresponding to the HTTP methods it handles.

LLM-generated Refactoring:

- The LLM-generated refactoring consolidates all functionalities into a single UserResource class. The UserResource class delegates the handling of different operations to a separate service class (UserManagementService).
- The service class encapsulates the business logic for each operation, providing a more modular and organized structure.
- The LLM-generated refactoring reduces code duplication by centralizing common functionalities within the service class.

Evaluation of Strengths and Weaknesses:

Manual Refactoring:

Strengths:

- `Clear separation of concerns`: Each resource class handles a specific set of operations, promoting code organization and readability.
- `Easier to maintain`: Changes to one set of functionalities are less likely to affect other parts of the codebase, reducing the risk of unintended consequences.
- `Scalability`: Adding new functionalities or modifying existing ones can be done more easily due to the modular structure.

Weaknesses:

- `Increased number of classes`: Having multiple resource classes might lead to a larger number of files, potentially making navigation more complex.
- `Manual effort required`: Refactoring manually requires careful analysis and understanding of the codebase, which can be time-consuming.

LLM-generated Refactoring:

Strengths:

- `Centralized business logic`: The service class centralizes the business logic, promoting code reusability and maintainability.
- `Reduced code duplication`: Common functionalities are encapsulated within the service class, minimizing redundant code.
- `Simplified resource class`: The UserResource class becomes less cluttered, focusing primarily on request handling and delegation.

Weaknesses:

- `Potential for complexity`: Concentrating all functionalities within a single class may lead to increased complexity and difficulty in understanding.
- `Lack of explicit separation`: Unlike the manual approach, where functionalities are clearly separated into different classes, the LLM-generated approach relies on comments or documentation to indicate the purpose of each method.
- `Limited control`: The LLM-generated refactoring might not always produce the most optimal or intuitive structure, as it lacks human judgment and context awareness.

Scenarios:

LLMs Excel:

- When there's a need to quickly prototype or generate initial code structure.
- For repetitive tasks where a standardized pattern or structure is sufficient.
- When there's a desire to minimize code duplication and promote reusability.

Manual Intervention is Preferable:

- When there's a need for a specific and optimized code structure tailored to the project's requirements.
- For complex refactoring tasks that require deep understanding of the codebase and domain logic.
- When clarity, maintainability, and scalability are critical factors, and a human-driven approach can provide better insights and decisions.

Abhhinav Reddy Boddu

1. Created UML diagram of User Management Sub-System
2. Design smell God Class of UserResource.java
3. Design smell Deficient Encapsulation of Constants.java

Rohit Gowlapalli

1. Created UML diagram of Book Addition and Display Sub-System
2. Design smell God Class of BookResource.java

Vempati Siva Koti Reddy

1. Created UML diagram of Book Addition and Display Sub-System
2. Design smell Feature Envy of TagDao.java

Sasidhar Chavali

1. Created UML diagram of Book Management Sub-System
2. Design smell Speculative Generality of App.java

Devisetti Sai Asrith

1. Design Smell God Class of AppContext.java
2. Design smell Missing Abstraction of UserBookDao.java