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.