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`:

```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`:

```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.