# 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.