

# BookResource - Brain Method

---

[Link to chat 1](#)

[Link to chat 2](#)

[Manual Refactoring](#)

file: books-web/src/main/java/com/sismics/books/rest/resource/BookResource.java

## Initial code

```
// Validate input data
title = ValidationUtil.validateLength(title, "title", 1, 255,
false);
subtitle = ValidationUtil.validateLength(subtitle, "subtitle", 1,
255, true);
author = ValidationUtil.validateLength(author, "author", 1, 255,
false);
description = ValidationUtil.validateLength(description,
"description", 1, 4000, true);
isbn10 = ValidationUtil.validateLength(isbn10, "isbn10", 10, 10,
true);
isbn13 = ValidationUtil.validateLength(isbn13, "isbn13", 13, 13,
true);
language = ValidationUtil.validateLength(language, "language", 2,
2, true);
Date publishDate = ValidationUtil.validateDate(publishDateStr,
"publish_date", false);

if (Strings.isNullOrEmpty(isbn10) && Strings.isNullOrEmpty(isbn13))
{
    throw new ClientException("ValidationError", "At least one ISBN
number is mandatory");
}

// Check if this book is not already in database
BookDao bookDao = new BookDao();
Book bookIsbn10 = bookDao.getByIsbn(isbn10);
Book bookIsbn13 = bookDao.getByIsbn(isbn13);
if (bookIsbn10 != null || bookIsbn13 != null) {
    throw new ClientException("BookAlreadyAdded", "Book already
added");
}

// Create the book
Book book = new Book();
book.setId(UUID.randomUUID().toString());

if (title != null) {
    book.setTitle(title);
}
```

```

    }
    if (subtitle != null) {
        book.setSubtitle(subtitle);
    }
    if (author != null) {
        book.setAuthor(author);
    }
    if (description != null) {
        book.setDescription(description);
    }
    if (isbn10 != null) {
        book.setIsbn10(isbn10);
    }
    if (isbn13 != null) {
        book.setIsbn13(isbn13);
    }
    if (pageCount != null) {
        book.setPageCount(pageCount);
    }
    if (language != null) {
        book.setLanguage(language);
    }
    if (publishDate != null) {
        book.setPublishDate(publishDate);
    }

    bookDao.create(book);

    // Create the user book
    UserBookDao userBookDao = new UserBookDao();
    UserBook userBook = new UserBook();
    userBook.setUserId(principal.getId());
    userBook.setBookId(book.getId());
    userBook.setCreateDate(new Date());
    userBookDao.create(userBook);

    // Update tags
    if (tagList != null) {
        TagDao tagDao = new TagDao();
        Set<String> tagSet = new HashSet<>();
        Set<String> tagIdSet = new HashSet<>();
        List<Tag> tagDbList = tagDao.getByUserId(principal.getId());
        for (Tag tagDb : tagDbList) {
            tagIdSet.add(tagDb.getId());
        }
        for (String tagId : tagList) {
            if (!tagIdSet.contains(tagId)) {
                throw new ClientException("TagNotFound",
                    MessageFormat.format("Tag not found: {0}", tagId));
            }
            tagSet.add(tagId);
        }
        tagDao.updateTagList(userBook.getId(), tagSet);
    }

```

```

        // Returns the book ID
        JSONObject response = new JSONObject();
        response.put("id", userBook.getId());
        return Response.ok().entity(response).build();
    }

    /**
     * Updates the book.
     *
     * @param title Title
     * @param description Description
     * @return Response
     * @throws JSONException
     */
    @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 {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Validate input data
        title = ValidationUtil.validateLength(title, "title", 1, 255,
true);
        subtitle = ValidationUtil.validateLength(subtitle, "subtitle", 1,
255, true);
        author = ValidationUtil.validateLength(author, "author", 1, 255,
true);
        description = ValidationUtil.validateLength(description,
"description", 1, 4000, true);
        isbn10 = ValidationUtil.validateLength(isbn10, "isbn10", 10, 10,
true);
        isbn13 = ValidationUtil.validateLength(isbn13, "isbn13", 13, 13,
true);
        language = ValidationUtil.validateLength(language, "language", 2,
2, true);
        Date publishDate = ValidationUtil.validateDate(publishDateStr,
"publish_date", true);

        // Get the user book
        UserBookDao userBookDao = new UserBookDao();

```

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

        // Get the book
        Book book = bookDao.getById(userBook.getBookId());

        // Check that new ISBN number are not already in database
        if (!Strings.isNullOrEmpty(isbn10) && book.getIsbn10() != null &&
!book.getIsbn10().equals(isbn10)) {
            Book bookIsbn10 = bookDao.getByIsbn(isbn10);
            if (bookIsbn10 != null) {
                throw new ClientException("BookAlreadyAdded", "Book already
added");
            }
        }

        if (!Strings.isNullOrEmpty(isbn13) && book.getIsbn13() != null &&
!book.getIsbn13().equals(isbn13)) {
            Book bookIsbn13 = bookDao.getByIsbn(isbn13);
            if (bookIsbn13 != null) {
                throw new ClientException("BookAlreadyAdded", "Book already
added");
            }
        }

        // Update the book
        if (title != null) {
            book.setTitle(title);
        }
        if (subtitle != null) {
            book.setSubtitle(subtitle);
        }
        if (author != null) {
            book.setAuthor(author);
        }
        if (description != null) {
            book.setDescription(description);
        }
        if (isbn10 != null) {
            book.setIsbn10(isbn10);
        }
        if (isbn13 != null) {
            book.setIsbn13(isbn13);
        }
        if (pageCount != null) {
            book.setPageCount(pageCount);
        }
        if (language != null) {
            book.setLanguage(language);
        }
    }
}
```

```

    }
    if (publishDate != null) {
        book.setPublishDate(publishDate);
    }

    // Update tags
    if (tagList != null) {
        TagDao tagDao = new TagDao();
        Set<String> tagSet = new HashSet<>();
        Set<String> tagIdSet = new HashSet<>();
        List<Tag> tagDbList = tagDao.getByUserId(principal.getId());
        for (Tag tagDb : tagDbList) {
            tagIdSet.add(tagDb.getId());
        }
        for (String tagId : tagList) {
            if (!tagIdSet.contains(tagId)) {
                throw new ClientException("TagNotFound",
MessageFormat.format("Tag not found: {0}", tagId));
            }
            tagSet.add(tagId);
        }
        tagDao.updateTagList(userBookId, tagSet);
    }

    // Returns the book ID
    JSONObject response = new JSONObject();
    response.put("id", userBookId);
    return Response.ok().entity(response).build();
}

/**
 * Get a book.
 *
 * @param id User book ID
 * @return Response
 * @throws JSONException
 */
@GET
@Path("/{id: [a-z0-9\\-]+}")
@Produces(MediaType.APPLICATION_JSON)
public Response get(
    @PathParam("id") String userBookId) throws JSONException {
    if (!authenticate()) {
        throw new ForbiddenClientException();
    }

    // Fetch the user book
    UserBookDao userBookDao = new UserBookDao();
    UserBook userBook = userBookDao.getUserBook(userBookId,
principal.getId());
    if (userBook == null) {
        throw new ClientException("BookNotFound", "Book not found with
id " + userBookId);
    }
}

```

```

// Fetch the book
BookDao bookDao = new BookDao();
Book bookDb = bookDao.getById(userBook.getBookId());

// Return book data
JSONObject book = new JSONObject();
book.put("id", userBook.getId());
book.put("title", bookDb.getTitle());
book.put("subtitle", bookDb.getSubtitle());
book.put("author", bookDb.getAuthor());
book.put("page_count", bookDb.getPageCount());
book.put("description", bookDb.getDescription());
book.put("isbn10", bookDb.getIsbn10());
book.put("isbn13", bookDb.getIsbn13());
book.put("language", bookDb.getLanguage());
if (bookDb.getPublishDate() != null) {
    book.put("publish_date", bookDb.getPublishDate().getTime());
}
book.put("create_date", userBook.getCreateDate().getTime());
if (userBook.getReadDate() != null) {
    book.put("read_date", userBook.getReadDate().getTime());
}

// Add tags
TagDao tagDao = new TagDao();
List<TagDto> tagDtoList = tagDao.getByUserBookId(userBookId);
List<JSONObject> tags = new ArrayList<>();
for (TagDto tagDto : tagDtoList) {
    JSONObject tag = new JSONObject();
    tag.put("id", tagDto.getId());
    tag.put("name", tagDto.getName());
    tag.put("color", tagDto.getColor());
    tags.add(tag);
}
book.put("tags", tags);

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

```

## Chat 1: Before being told that it was a brain method design smell:

Certainly! Here are some design smells I've identified in the code snippet:

1. **Repeated Code:** There's a significant amount of repetition, particularly in the validation and update logic for book attributes.
2. **Business Logic in Resource Methods:** Business logic, such as validation and database operations, is mixed with resource methods, violating the Single Responsibility Principle.

3. **Data Access Object (DAO) Instantiation:** DAOs are directly instantiated within resource methods, making it hard to test and violating the Dependency Inversion Principle.
4. **Lack of Error Handling:** Error handling is scattered throughout the code and could be better organized.
5. **Inefficient Tag Handling:** Tag handling involves multiple database operations and could be optimized.

To refactor the code, we'll create separate classes for validation, database access, and error handling. We'll also consolidate common operations and improve error handling. Here's a refactored version:

```
// ValidationUtil.java
public class ValidationUtil {
    public static String validateLength(String value, String fieldName, int
minLength, int maxLength, boolean allowNull) {
        // Validation logic
    }

    public static Date validateDate(String value, String fieldName, boolean
allowNull) {
        // Validation logic
    }
}

// BookService.java
public class BookService {
    private BookDao bookDao;
    private UserBookDao userBookDao;
    private TagDao tagDao;

    public BookService() {
        this.bookDao = new BookDao();
        this.userBookDao = new UserBookDao();
        this.tagDao = new TagDao();
    }

    public String createBook(...) {
        // Validation logic
        // Create book logic
        // Update tags logic
        // Return book ID
    }

    public String updateBook(...) {
        // Validation logic
        // Fetch user book
        // Fetch book
        // Check ISBN uniqueness
        // Update book logic
        // Update tags logic
        // Return book ID
    }
}
```

```

    }

    public JSONObject getBook(String userBookId) {
        // Fetch user book
        // Fetch book
        // Fetch tags
        // Construct JSON response
        // Return JSON response
    }
}

// BookResource.java
@Path("/books")
public class BookResource {
    private BookService bookService;

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

    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public Response createBook(...) {
        // Call BookService.createBook method
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response updateBook(...) {
        // Call BookService.updateBook method
    }

    @GET
    @Path("{id: [a-z0-9\\-]+}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getBook(@PathParam("id") String userBookId) {
        // Call BookService.getBook method
    }
}

```

In this refactored version, we've separated concerns by creating a **BookService** class to handle business logic, a **ValidationUtil** class for validation, and DAO classes for data access. The **BookResource** class now acts as a controller, delegating tasks to the **BookService** class. This separation of concerns makes the code more modular, easier to maintain, and testable.

---

## Chat 2: After being told that it was a brain method design smell:

Sure, the provided code snippet could benefit from refactoring to improve readability and maintainability. One approach to address the brain method design smell is to break down the large methods into smaller, more focused ones. Here's a refactored version of the code:



```

public class BookService {

    private BookDao bookDao = new BookDao();
    private UserBookDao userBookDao = new UserBookDao();
    private TagDao tagDao = new TagDao();

    public Response createBook(String title, String subtitle, String
author, String description,
                                String isbn10, String isbn13, Long
pageCount, String language,
                                String publishDateStr, List<String> tagList)
throws JSONException {

        validateInputData(title, subtitle, author, description, isbn10,
isbn13, language, publishDateStr);

        checkUniqueISBN(isbn10, isbn13);

        Book book = createBookObject(title, subtitle, author, description,
isbn10, isbn13, pageCount, language, publishDateStr);

        saveBookToDatabase(book);

        String userBookId = createUserBook(book);

        updateTags(tagList, userBookId);

        return Response.ok().entity(createResponse(userBookId)).build();
    }

    private void validateInputData(String title, String subtitle, String
author, String description,
                                String isbn10, String isbn13, String
language, String publishDateStr) {
        // Validation logic
    }

    private void checkUniqueISBN(String isbn10, String isbn13) {
        // Check if ISBN numbers are unique
    }

    private Book createBookObject(String title, String subtitle, String
author, String description,
                                String isbn10, String isbn13, Long
pageCount, String language,
                                String publishDateStr) {
        // Create and populate Book object
        return new Book.Builder()
            .setTitle(title)
            .setSubtitle(subtitle)
            .setAuthor(author)
            .setDescription(description)

```

```

        .setIsbn10(isbn10)
        .setIsbn13(isbn13)
        .setPageCount(pageCount)
        .setLanguage(language)
        .setPublishDate(ValidationUtil.validateDate(publishDateStr,
"publish_date", false))
        .build();
    }

    private void saveBookToDatabase(Book book) {
        bookDao.create(book);
    }

    private String createUserBook(Book book) {
        UserBook userBook = new UserBook();
        userBook.setUserId(principal.getId());
        userBook.setBookId(book.getId());
        userBook.setCreateDate(new Date());
        userBookDao.create(userBook);
        return userBook.getId();
    }

    private void updateTags(List<String> tagList, String userBookId) {
        if (tagList != null) {
            Set<String> tagSet = validateAndGetTagIds(tagList);
            tagDao.updateTagList(userBookId, tagSet);
        }
    }

    private Set<String> validateAndGetTagIds(List<String> tagList) {
        // Validate and get tag IDs
        return tagDao.validateAndGetTagIds(principal.getId(), tagList);
    }

    private JSONObject createResponse(String userBookId) throws
JSONException {
        JSONObject response = new JSONObject();
        response.put("id", userBookId);
        return response;
    }

    // Other methods (update and get) can follow a similar pattern of
    breaking down functionality into smaller methods.
}

```

In this refactored version:

1. Each major functionality (validation, book creation, database operations, etc.) is extracted into its own method, improving readability and facilitating easier maintenance.
2. Logic related to validation, database operations, and business logic is separated, adhering to the Single Responsibility Principle.
3. Error handling and response creation are handled uniformly across methods.

#### 4. The code is organized in a more modular and understandable manner.

---

Here, we see that the LLM was not able to figure out that it was a brain method design smell, and thus was only able to make minor changes for the refactoring - that to in completely unrelated parts.

However, when the issues (brain method design smell) and the location were highlighted, the LLM performed much better and was able to show successful rectification of the issue.

When the AI model was not told about the design smell, it couldn't recognize it, and just gave almost the same skeleton of the code, whereas when it was given about the design smell, it did refactor it the way expected. Although, the code has various bugs, like it replaced the conditional checking of the various parameters of the book class while setting them to a single statement, without checking for null in them. There are various other functions where the implementation doesn't exactly match what was done in the initial code.