

Book Manager - Project Documentation

COVER PAGE

Institute Name	:	Besant Technologies
Project Name	:	Book Manager
Submitted to	:	Roshiga
Submitted by	:	Nagasri Bondili
Student ID	:	25BTF41834
Batch No	:	418 Java
Co-ordinator Name	:	Roshiga
Date of Submission	:	July 30, 2025

Acknowledgement

I would like to express my sincere gratitude to my project coordinator Roshiga and Rajesh Sir of the Java department Faculty at Besant Technologies for their invaluable guidance, encouragement, and support throughout the duration of this project.

Their mentorship was instrumental in helping me understand the core concepts of Java web development and in successfully completing the Book Manager application. This project has been a significant learning experience, and I am thankful for the opportunity.

I also extend my thanks to my family and friends for their constant encouragement and support.

Sincerely,
Nagasri Bondili

Table of Contents

1. Concepts used in Project
2. Source Code
3. Description of Source Code
4. Output
5. Conclusion
6. Bibliography

Concepts Used in Project

This project is built upon several core concepts of Java web development and software design patterns:

- **Model-View-Controller (MVC) Architecture:** The project is strictly divided into three parts:
- **Model:** The Book.java bean and the BookDao.java class, which handle the data and business logic.
- **View:** The JSP files (list-books.jsp, add-book.jsp, edit-book.jsp), which are responsible for presenting the data to the user.
- **Controller:** The BookServlet.java class, which handles user requests, interacts with the Model, and selects the appropriate View to display.
- **Java Servlets:** A single servlet acts as the central controller, receiving all HTTP requests and directing the application's workflow.
- **JavaServer Pages (JSP):** Used as the template engine to create dynamic HTML pages.
- **JSP Standard Tag Library (JSTL) & Expression Language (EL):** These are used to eliminate Java code (scriptlets) from the JSP files. JSTL provides tags for loops (<c:forEach>) and conditional logic, while EL (\${...}) provides easy access to data.
- **Data Access Object (DAO) Pattern:** The BookDao class abstracts and encapsulates all data access logic, separating it from the business logic layer.
- **Singleton Pattern:** The BookDao is implemented as a Singleton to ensure there is only one instance managing the in-memory data store, preventing data inconsistencies.
- **Thread Safety:** The in-memory data store (List<Book>) is made thread-safe to handle concurrent user requests without data corruption.
- **Deployment Descriptor (web.xml):** This file is used to configure the servlet and map incoming URL patterns to the controller servlet.

Source Code

Model: Book.java

```
package com.bookmanager;

public class Book {
    private int id;
    private String name;
    private String author;
    private String status;

    public Book(String name, String author, String status) {
        this.name = name;
        this.author = author;
        this.status = status;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAuthor() {
```

```

        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
}

```

BookDao.java

```

package com.bookmanager;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.atomic.AtomicInteger;

public class BookDao {

    private static final BookDao INSTANCE = new BookDao();

    private final List<Book> bookList = Collections.synchronizedList(new
ArrayList<>());

    private final AtomicInteger idCounter = new AtomicInteger();

    private BookDao() {
        addBook(new Book("The Hobbit", "J.R.R. Tolkien", "Available"));
    }
}

```

```

        addBook(new Book("1984", "George Orwell", "Issued"));
    }

    public static BookDao getInstance() {
        return INSTANCE;
    }

    public List<Book> getAllBooks() {
        return new ArrayList<>(bookList);
    }

    public Optional<Book> getBookById(int id) {
        return bookList.stream().filter(book -> book.getId() == id).findFirst();
    }

    public void addBook(Book book) {
        book.setId(idCounter.getAndIncrement());
        bookList.add(book);
    }

    public void updateBook(Book updatedBook) {
        getBookById(updatedBook.getId()).ifPresent(book -> {
            book.setName(updatedBook.getName());
            book.setAuthor(updatedBook.getAuthor());
            book.setStatus(updatedBook.getStatus());
        });
    }

    public void deleteBook(int id) {
        bookList.removeIf(book -> book.getId() == id);
    }
}

```


BookServlet.java

```
package com.bookmanager;

import java.io.IOException;
import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/bookServlet")
public class BookServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private final BookDao bookDao = BookDao.getInstance();
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String action = request.getParameter("action");
        if (action == null) {
            action = "list";
        }
        switch (action) {
            case "new":
                showNewForm(request, response);
                break;
            case "edit":
```

```

        showEditForm(request, response);
        break;
    default:
        listBooks(request, response);
        break;
    }
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String action = request.getParameter("action");
    if (action == null) {
        response.sendRedirect(request.getContextPath() + "/bookServlet");
        return;
    }
    switch (action) {
        case "add":
            addBook(request, response);
            break;
        case "update":
            updateBook(request, response);
            break;
        case "delete":
            deleteBook(request, response);
            break;
        default:

```

```

        listBooks(request, response);
        break;
    }
}

private void listBooks(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    List<Book> bookList = bookDao.getAllBooks();
    request.setAttribute("bookList", bookList);
    RequestDispatcher dispatcher = request.getRequestDispatcher("list-
books.jsp");
    dispatcher.forward(request, response);
}

private void showNewForm(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    RequestDispatcher dispatcher = request.getRequestDispatcher("add-
book.jsp");
    dispatcher.forward(request, response);
}

private void showEditForm(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    bookDao.getBookById(id).ifPresent(book -> request.setAttribute("book",
book));
    RequestDispatcher dispatcher = request.getRequestDispatcher("edit-
book.jsp");
    dispatcher.forward(request, response);
}

```

```

private void addBook(HttpServletRequest request, HttpServletResponse
response) throws IOException {
    String name = request.getParameter("name");
    String author = request.getParameter("author");
    String status = request.getParameter("status");
    bookDao.addBook(new Book(name, author, status));
    response.sendRedirect(request.getContextPath() +
"/bookServlet?action=list");
}

private void updateBook(HttpServletRequest request, HttpServletResponse
response) throws IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    String name = request.getParameter("name");
    String author = request.getParameter("author");
    String status = request.getParameter("status");
    Book book = new Book(name, author, status);
    book.setId(id);
    bookDao.updateBook(book);
    response.sendRedirect(request.getContextPath() +
"/bookServlet?action=list");
}

private void deleteBook(HttpServletRequest request, HttpServletResponse
response) throws IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    bookDao.deleteBook(id);
    response.sendRedirect(request.getContextPath() +
"/bookServlet?action=list");
}

```

```
}
```

list-books.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<html>
```

```
<head>
```

```
    <title>Book List</title>
```

```
    <link rel="stylesheet" href="style.css">
```

```
</head>
```

```
<body>
```

```
    <h2>Book Manager</h2>
```

```
    <a href="bookServlet?action=new">Add New Book</a>
```

```
    <hr>
```

```
    <table border="1">
```

```
        <tr>
```

```
            <th>ID</th>
```

```
            <th>Name</th>
```

```
            <th>Author</th>
```

```
            <th>Status</th>
```

```
            <th>Actions</th>
```

```
        </tr>
```

```
        <c:forEach var="book" items="${bookList}">
```

```
            <tr>
```

```
                <td><c:out value="${book.id}" /></td>
```

```
                <td><c:out value="${book.name}" /></td>
```



```

<form action="bookServlet?action=add" method="post">
    <label>Book Name</label>
    <input type="text" name="name" required><br>
    <label>Author Name</label>
    <input type="text" name="author" required><br>
    <label>Available Status</label>
    <input type="text" name="status" required><br>
    <input type="submit" value="Add Book">
</form>
<br>
<a href="bookServlet?action=list">Back to List</a>
</body>
</html>

```

edit-book.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <title>Edit Book</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h2>Edit Book</h2>
    <c:if test="${book != null}">
        <form action="bookServlet?action=update" method="post">

```

```

        <input type="hidden" name="id" value="<c:out value='${book.id}' />">

        <label>Book Name</label>

        <input type="text" name="name" value="<c:out value='${book.name}'
/>" required><br>

        <label>Author Name</label>

        <input type="text" name="author" value="<c:out
value='${book.author}' />" required><br>

        <label>Available Status</label>

        <input type="text" name="status" value="<c:out value='${book.status}'
/>" required><br>

        <input type="submit" value="Update Book">

    </form>    </c:if>

    <c:if test="${book == null}">

        <p>Book not found.</p>

    </c:if>

    <br>

    <a href="bookServlet?action=list">Back to List</a>

</body>

</html>

```

web.xml

```

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">

    <servlet>

        <servlet-name>BookServlet</servlet-name>

        <servlet-class>com.bookmanager.BookServlet</servlet-class>

    </servlet>    <servlet-mapping>

        <servlet-name>BookServlet</servlet-name>

        <url-pattern>/bookServlet</url-pattern>

```



```
</servlet-mapping>
    <welcome-file-list>
        <welcome-file>bookServlet</welcome-file>
    </welcome-file-list> </web-app>
```

style.css

```
body {
    font-family: Arial, sans-serif;
    padding: 20px;
    margin: 20px;
}

input, label, table {
    display: block;
    margin-bottom: 10px;
}

table {
    border-collapse: collapse;
    width: 80%;
}

th, td {
    border: 1px solid #ccc;
    padding: 8px;
    text-align: left;
}

th {
    background-color: #f2f2f2;
}
```

Description of Source Code

Book.java: This is a simple POJO (Plain Old Java Object) that acts as the model for a book. It contains private fields for id, name, author, and status, along with public getters and setters to access and modify these properties.

BookDao.java: This class handles all data persistence logic. It uses the Singleton design pattern to ensure only one instance of the DAO exists. It stores the book data in a thread-safe `synchronizedList` to prevent issues with concurrent access. It contains methods for all CRUD operations: `addBook`, `getAllBooks`, `getBookById`, `updateBook`, and `deleteBook`.

BookServlet.java: This is the single controller for the entire application. It uses the `@WebServlet` annotation to map itself to the URL pattern `/bookServlet`. It handles both GET and POST requests and uses an action parameter to determine which operation to perform (e.g., list, show add form, add, delete). It interacts with the `BookDao` to get or modify data and then forwards the request to the appropriate JSP page for rendering the view.

JSP Files (list-books.jsp, add-book.jsp, edit-book.jsp): These files form the View layer. They contain no Java code. Instead, they use JSTL tags (like `<c:forEach>`) and EL expressions (`${...}`) to dynamically display the data passed to them by the servlet.

web.xml: The Deployment Descriptor file maps the URL `/bookServlet` to our `BookServlet` class and sets the default welcome file to launch the servlet.

Outputs

The application has three main screens:



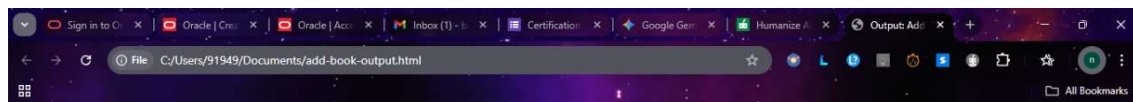
Book Manager

Add New Book				
ID	Name	Author	Status	Actions
0	The Hobbit	J.R.R. Tolkien	Available	Edit Delete
1	1984	George Orwell	Issued	Edit Delete



Book List Page (list-books.jsp)

This is the main screen. It displays a heading "Book Manager" with an "Add New Book" link below it. A table shows all the books in the system with columns for ID, Name, Author, and Status. The final column, "Actions," contains an "Edit" link and a "Delete" button for each book.



Add a New Book

Book Name

Author Name

Available Status

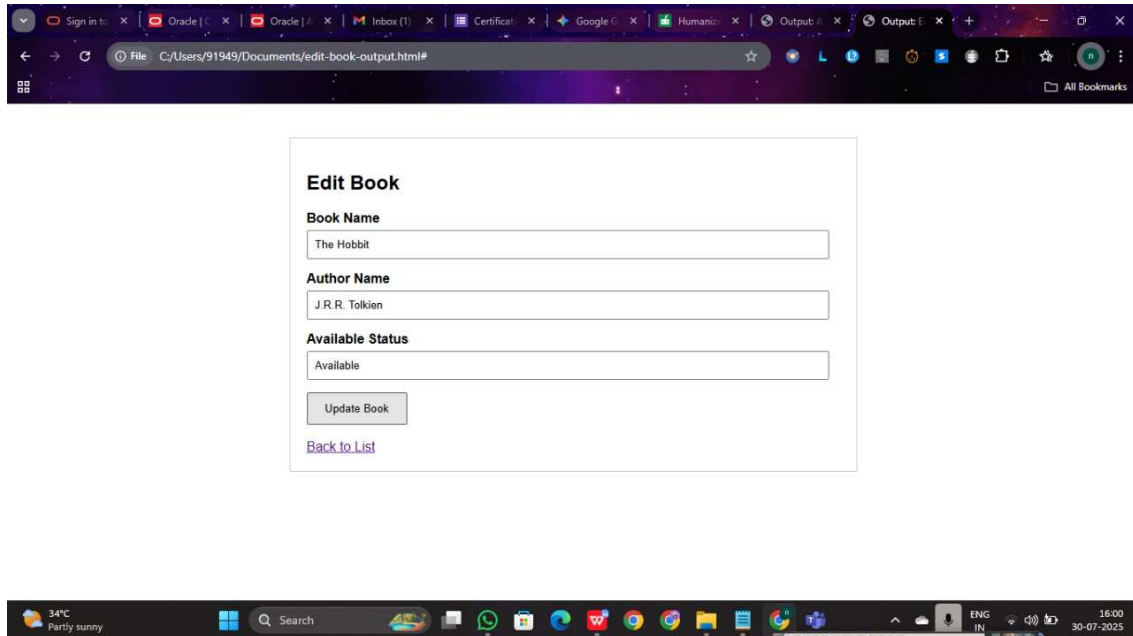
[Add Book](#)

[Back to List](#)



Add Book Page (add-book.jsp)

This page presents a form with the heading "Add a New Book". It has text input fields for "Book Name," "Author Name," and "Available Status." Below the fields is an "Add Book" submit button and a "Back to List" link.



The screenshot shows a web browser window with the address bar displaying "File C:/Users/91949/Documents/edit-book-output.html#". The browser has several tabs open, including "Sign in to...", "Oracle |...", "Oracle |...", "Inbox (1)", "Certificat...", "Google C...", "Humanit...", "Output...", and "Output...". The main content area displays a form titled "Edit Book". The form contains three text input fields: "Book Name" with the value "The Hobbit", "Author Name" with the value "J.R.R. Tolkien", and "Available Status" with the value "Available". Below these fields is a button labeled "Update Book" and a link labeled "Back to List". The Windows taskbar at the bottom shows the date and time as "10:00 30-07-2023" and the weather as "34°C Partly sunny".

Edit Book Page (edit-book.jsp)

This screen is visually similar to the "Add Book" page but is titled "Edit Book." The input fields for Name, Author, and Status are pre-populated with the details of the book selected for editing. The submit button is labeled "Update Book."

Conclusion

The Book Manager project successfully demonstrates the development of a well-structured and functional Java web application. By correctly implementing the Model-View-Controller (MVC) pattern, the application achieves a clean separation of concerns, making it easy to manage, debug, and extend.

Key achievements include the use of a single controller servlet for centralized request handling, a dedicated and thread-safe DAO for data management, and the complete removal of scriptlets from JSPs in favor of the modern JSTL and EL standard. This project serves as a solid foundation and a practical example of building robust web applications using core Java EE technologies.

Bibliography

This project was developed using standard Java EE technologies and common design patterns.

The following resources are excellent references for the concepts used:

Books:

1. Head First Servlets & JSP, 2nd Edition by Bryan Basham, Kathy Sierra & Bert Bates.
2. Core Servlets and JavaServer Pages, 2nd Edition by Marty Hall and Larry Brown.

Online Documentation & Tutorials:

1. The Official Oracle/Jakarta EE Tutorials for Servlets and JSP.

Websites like Baeldung, GeeksforGeeks, and TutorialsPoint for specific implementation details and examples.