Experimental Report

Session and Session Technologies

Student Name: Mohammad Abdullah

Student ID: 202322240356

Course: Java Web

Date: 2 December 2025

Instructor: D.cora

Department of Computer Science
China West Normal University
Academic Year: 2025

Contents

Abstract

This experiment implements a small servlet-based web application using Java Servlets and JSP, developed in Visual Studio Code with Maven and deployed on Apache Tomcat 10.1. The application models a simple Bookstore backed by an in-memory data store and supports three core operations: listing all books, viewing a single book, and simulating a purchase. The work emphasises correct use of the Jakarta Servlet API 5, Maven-based build configuration, servlet–JSP interaction via request attributes and JSP Expression Language (EL), and manual WAR deployment. Several practical issues were encountered, including HTTP 404 context-path errors and JSP compilation failures caused by incorrect scriptlet usage and character encoding. These were resolved through systematic debugging, namespace alignment, and a transition from scriptlets to EL. The final result is a fully functioning servlet-based application with a reproducible build and deployment workflow.

Keywords: Java Servlets; JSP; Tomcat 10.1; Jakarta EE; Maven; VS Code; Web Application

1 Objective

The objective of this experiment was to design, implement, and deploy a servlet-based Java web application using Visual Studio Code and Apache Tomcat. Specifically, the goals were:

- To construct a Maven-based web project that follows standard Java web application structure.
- To implement the domain model (`Book`, `BookDB`) and three servlets: `ListBookServlet`, `BookServlet`, and `PurchaseServlet`.
- To integrate JSP pages for list, detail, and purchase confirmation views using JSP EL.
- To deploy the application as a WAR file to Apache Tomcat 10.1 and verify correct behaviour via browser-based testing.
- To diagnose and resolve practical issues such as context-path 404 errors, servlet/JSP configuration problems, and JSP compilation errors caused by Unicode content.

2 Environment Setup and Tools

2.1 Software Environment

The experiment was conducted using the following software stack:

Component	**Specification**
Operating System	Windows 11
Integrated Development Environment	Visual Studio Code (Version 1.94 or similar)
Java Development Kit	Eclipse Adoptium JDK 17.0.15
Application Server	Apache Tomcat 10.1.20 (Jakarta Servlet 5.0)
Build Automation Tool	Apache Maven 3.9.11
Version Control (optional)	Git (local repository)

Table 1: Software environment specifications.

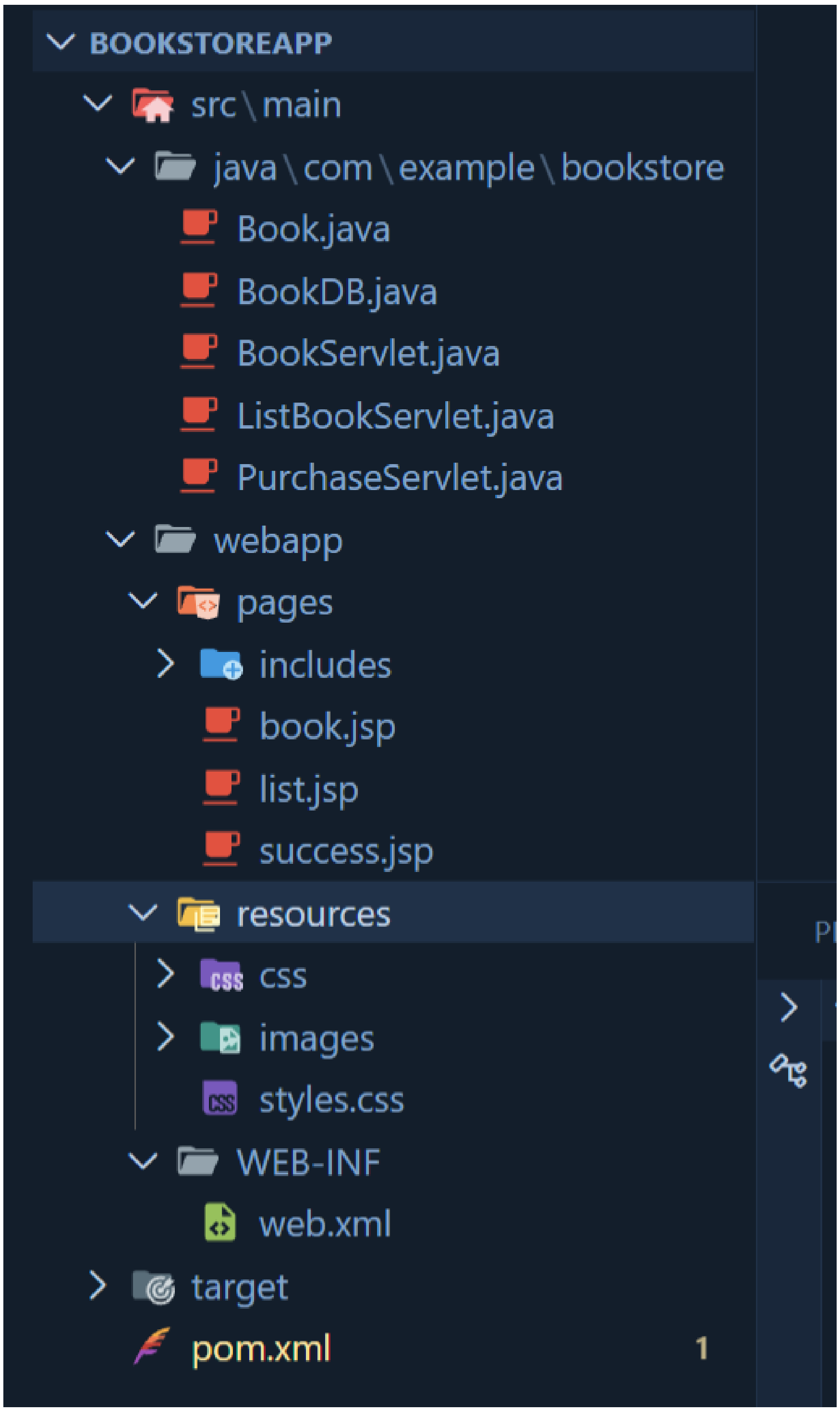
2.2 VS Code Extensions

The following VS Code extensions were used to support development:

- **Extension Pack for Java** (Microsoft) — Java language support and debugging.
- **Maven for Java** — Integration of Maven goals within VS Code.
- **Community Server Connectors** (Red Hat) — Management of the Tomcat server from within the IDE.

2.3 Project Structure Screenshot

Figure 1 shows the overall project structure in VS Code, including Java sources, web resources, configuration files, and the Maven `pom.xml`.



```
BOOKSTOREAPP
  src\main
    java\com\example\bookstore
      Book.java
      BookDB.java
      BookServlet.java
      ListBookServlet.java
      PurchaseServlet.java
    webapp
      pages
        includes
        book.jsp
        list.jsp
        success.jsp
      resources
        css
        images
        styles.css
      WEB-INF
        web.xml
  target
  pom.xml                              1
```

Figure 1: VS Code Explorer showing the Bookstore servlet project structure.

3 System Design and Implementation

3.1 Project Initialization

A new Maven web project was created using the `maven-archetype-webapp` archetype. The project was scaffolded via the command line:

Listing 1: Maven command for creating the Bookstore web project.

```
mvn archetype:generate ^
  -DgroupId=com.example ^
  -DartifactId=bookstore ^
  -DarchetypeGroupId=org.apache.maven.archetypes ^
  -DarchetypeArtifactId=maven-archetype-webapp ^
  -DarchetypeVersion=1.4 ^
  -DinteractiveMode=false
```

The generated directory structure follows Maven conventions: Java sources under `src/main/java`, web resources under `src/main/webapp`, and configuration files such as `web.xml` under `WEB-INF`.

3.2 Maven Configuration

The `pom.xml` file was configured to match Tomcat 10.1, which implements Jakarta Servlet 5.0 and JSP 3.0. The final configuration is summarised below:

Listing 2: Relevant sections of pom.xml for the Bookstore application.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>bookstore</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <!-- Servlet API 5.0 (Tomcat 10.x) -->
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>5.0.0</version>
      <scope>provided</scope>
    </dependency>

    <!-- JSP API 3.0 (Tomcat 10.x) -->
    <dependency>
      <groupId>jakarta.servlet.jsp</groupId>
      <artifactId>jakarta.servlet.jsp-api</artifactId>
      <version>3.0.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>bookstore</finalName>
    <pluginManagement>
      <plugins>
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>3.2.2</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

3.3 Web Application Descriptor

The application descriptor `web.xml` adopts the Jakarta EE 9 namespace and Servlet 5.0 schema, and sets the `list.jsp` page as the welcome file:

Listing 3: web.xml configuration for the Bookstore application.

```
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
                      https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
  version="5.0">

  <display-name>Bookstore</display-name>

  <welcome-file-list>
    <welcome-file>pages/list.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

Individual servlets are configured using the `@WebServlet` annotation rather than explicit mappings in `web.xml`, which keeps the descriptor concise.

3.4 Domain Model: Book and BookDB

The domain model consists of a simple `Book` class and an in-memory `BookDB` store:

Listing 4: Book and BookDB domain classes with Chinese titles.

```
public class Book implements Serializable {
    private static final long serialVersionUID = 1L;
    private String id;
    private String name;

    public Book(String id, String name) {
        this.id = id;
        this.name = name;
    }
    public String getId()  { return id; }
    public String getName(){ return name; }
}

public class BookDB {
    private static Map<String, Book> books = new LinkedHashMap<>();
    static {
        books.put("1", new Book("1", "javaweb"));
        books.put("2", new Book("2", "jdbc"));
        books.put("3", new Book("3", "java"));
        books.put("4", new Book("4", "struts"));
        books.put("5", new Book("5", "spring"));
    }

    public static Collection<Book> getAll() {
        return books.values();
    }

    public static Book getBook(String id) {
        return books.get(id);
    }
}
```

3.5 Servlet Design

Three servlets implement the core application logic:

- **`ListBookServlet`** — retrieves all books and forwards to `list.jsp`.
- **`BookServlet`** — retrieves a single `Book` by ID and forwards to `book.jsp`.
- **`PurchaseServlet`** — simulates a purchase and forwards to `success.jsp`.

A representative example, `ListBookServlet`, is shown below:

Listing 5: ListBookServlet using the Jakarta Servlet API.

```
@WebServlet("/listBooks")
public class ListBookServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        Collection<Book> books = BookDB.getAll();
        req.setAttribute("books", books);
        req.getRequestDispatcher("/pages/list.jsp").forward(req, resp);
    }
}
```

The `BookServlet` and `PurchaseServlet` follow the same pattern: they obtain parameters from the request, query `BookDB`, store a `book` attribute, and forward to the appropriate JSP.

3.6 JSP Views and Interface Design

The view layer consists of three JSP pages and two shared layout fragments:

- `list.jsp` — displays a card-style list of all books with "View" links.
- `book.jsp` — shows detailed information about a selected book and provides a "Purchase" button.
- `success.jsp` — confirms the simulated purchase.
- `header.jsp` and `footer.jsp` — reusable layout for navigation and footer content.

To avoid scriptlet-related compilation errors and improve readability, JSP EL is used instead of Java code inside the pages. For example, `book.jsp` uses expressions such as `${book.name}` and `${book.id}` after the servlet places the `book` object in the request scope.

A simplified snippet from `book.jsp` is given below:

Listing 6: JSP snippet using Expression Language in book.jsp.

```
<h3>
  ${book != null ? book.name : 'Book not found'}
</h3>
<p class="text-muted">
  ID: ${book != null ? book.id : 'N/A'}
</p>
<form action="${pageContext.request.contextPath}/purchase" method="post">
  <input type="hidden" name="id" value="${book != null ? book.id : ''}" />
  <button class="btn btn-primary" type="submit">Purchase</button>
</form>
```

Styling is provided through a dedicated `styles.css` file, which defines a modern card layout and colour palette for the Bookstore interface.

3.7 UI Screenshots

Figure 2 and Figure 3 show the rendered list and detail views in the browser.

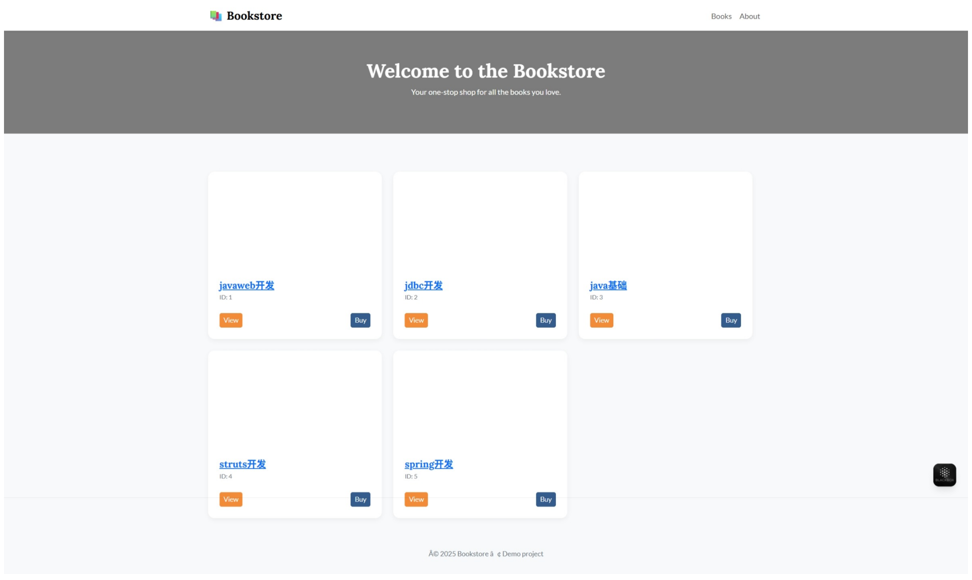
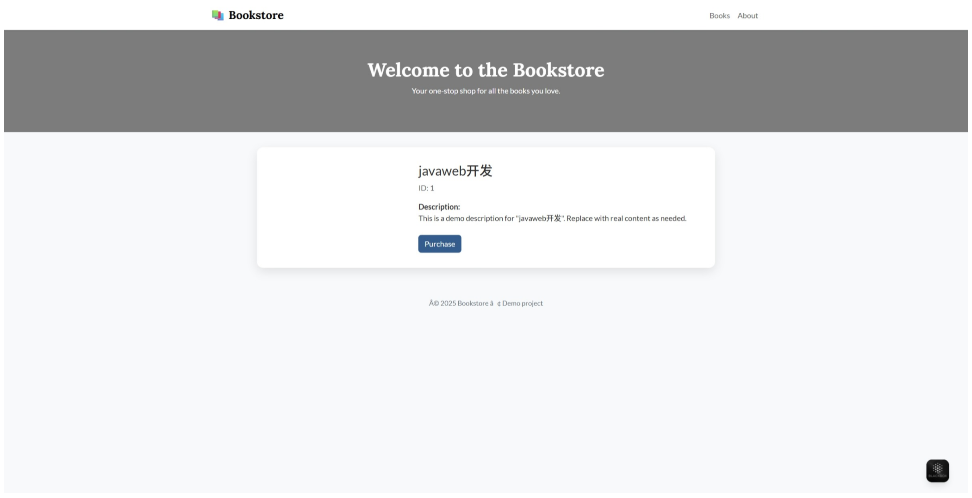


Figure 2: Book list view rendered in the browser.


Bookstore
Books About
Welcome to the Bookstore
Your one-stop shop for all the books you love.
javaweb开发
ID: 1
Description:
Purchase
Â© 2025 Bookstore â ¢ Demo project

This is a demo description for "javaweb开发". Replace with real content as needed.

Figure 3: Book detail view for `id=2`.

4 Deployment and Execution

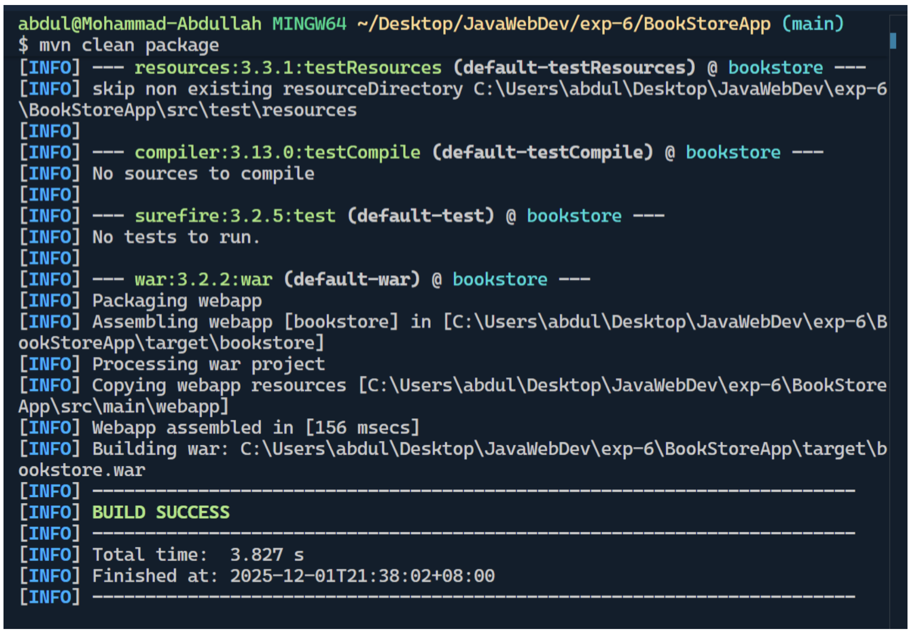
4.1 WAR Packaging

The application was packaged as a WAR file using Maven. In the project root, the following command was executed:

Listing 7: Maven build command for packaging the WAR.

```
mvn clean package
```

Upon success, Maven generated `target/bookstore.war`, where the `finalName` matches the desired context path.



```
abdul@Mohammad-Abdullah MINGW64 ~/Desktop/JavaWebDev/exp-6/BookStoreApp (main)
$ mvn clean package
[INFO] --- resources:3.3.1:testResources (default-testResources) @ bookstore ---
[INFO] skip non existing resourceDirectory C:\Users\abdul\Desktop\JavaWebDev\exp-6
\BookStoreApp\src\test\resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ bookstore ---
[INFO] No sources to compile
[INFO]
[INFO] --- surefire:3.2.5:test (default-test) @ bookstore ---
[INFO] No tests to run.
[INFO]
[INFO] --- war:3.2.2:war (default-war) @ bookstore ---
[INFO] Packaging webapp
[INFO] Assembling webapp [bookstore] in [C:\Users\abdul\Desktop\JavaWebDev\exp-6\B
ookStoreApp\target\bookstore]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\Users\abdul\Desktop\JavaWebDev\exp-6\BookStore
App\src\main\webapp]
[INFO] Webapp assembled in [156 msecs]
[INFO] Building war: C:\Users\abdul\Desktop\JavaWebDev\exp-6\BookStoreApp\target\b
ookstore.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.827 s
[INFO] Finished at: 2025-12-01T21:38:02+08:00
[INFO] ------------------------------------------------------------------------
```

Figure 4: VS Code terminal showing a successful Maven build.

4.2 Manual Deployment to Tomcat

Deployment to Apache Tomcat 10.1.20 was performed manually:

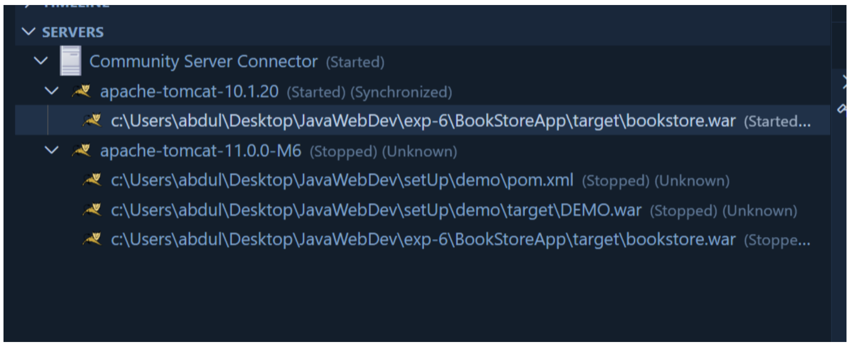
1. Stop Tomcat if it is running.
2. Copy `target/bookstore.war` into the `webapps/` directory of the Tomcat installation.
3. Start Tomcat using `startup.bat` (Windows) or `startup.sh` (Linux/macOS).
4. Verify that Tomcat automatically expands the WAR into a `webapps/bookstore/` directory.

The application root was then accessible at:

```
http://localhost:8080/bookstore/
```

4.3 Tomcat Management Screenshot

Figure 5 shows the Tomcat management or browser view confirming that the `bookstore` application is deployed and accessible.



```
SERVERS
  Community Server Connector (Started)
    apache-tomcat-10.1.20 (Started) (Synchronized)
      c:\Users\abdul\Desktop\JavaWebDev\exp-6\BookStoreApp\target\bookstore.war (Started...
    apache-tomcat-11.0.0-M6 (Stopped) (Unknown)
      c:\Users\abdul\Desktop\JavaWebDev\setUp\demo\pom.xml (Stopped) (Unknown)
      c:\Users\abdul\Desktop\JavaWebDev\setUp\demo\target\DEMO.war (Stopped) (Unknown)
      c:\Users\abdul\Desktop\JavaWebDev\exp-6\BookStoreApp\target\bookstore.war (Stoppe...
```

Figure 5: Tomcat showing the deployed `bookstore` application.

5 Testing and Results

5.1 Test Plan

Functional testing focused on the main use cases: viewing the book list, viewing a single book, and purchasing a book. Additional tests covered invalid input and direct access to the root context.

Test Case	**URL**	**Expected Result**	**Status**
List all books	`/bookstore/listBooks`	List of all 5 books	PASS
Application root	`/bookstore/`	Redirect/welcome to list view	PASS
View book detail	`/bookstore/book?id=2`	Detail view of book with ID 2	PASS
Purchase book	`/bookstore/purchase` (POST)	Success page showing book name	PASS
Invalid book ID	`/bookstore/book?id=999`	“Book not found” message	PASS

Table 2: Summary of functional test cases and results.

5.2 Representative Test Cases

5.2.1 Book List View

```
URL: http://localhost:8080/bookstore/listBooks
HTTP Method: GET
Response Code: 200 OK
Expected Output: HTML page listing all books
```

5.2.2 Book Detail View

```
URL: http://localhost:8080/bookstore/book?id=2
HTTP Method: GET
Response Code: 200 OK
Expected Output: Detailed view of the book with ID = 2
```

5.2.3 Purchase Flow

```
URL: http://localhost:8080/bookstore/purchase
HTTP Method: POST
Form Data: id=2
Response Code: 200 OK
Expected Output: Confirmation page naming the purchased book
```

5.3 Technical Validation

The testing phase confirmed the following:

- Servlet mappings via `@WebServlet` annotations function correctly.
- Request attributes are successfully transmitted from servlets to JSP pages.
- JSPs compile without scriptlet-related errors when using JSP EL.
- The WAR build and manual deployment process is reliable and repeatable.

6 Problems Encountered and Resolutions

6.1 Context Path 404 Errors

Problem: Initially, navigation to `http://localhost:8080/bookstore/` resulted in an HTTP 404 “resource not available” error, even though Tomcat was running.

Cause: The WAR name and the URL context path were inconsistent, and in some attempts the WAR was not correctly copied into the Tomcat `webapps/` directory.

Resolution:

- The <`finalName`> element in `pom.xml` was set to `bookstore` so that Maven produced `bookstore.war`.
- The WAR file was manually copied to the correct `webapps/` folder of Tomcat 10.1.20.
- The application was accessed explicitly via `/bookstore/`, matching the WAR name.

6.2 Servlet and Jakarta Version Alignment

Problem: Earlier attempts reused a configuration targeting Tomcat 11 and Jakarta Servlet 6.0, leading to confusion when switching to Tomcat 10.1.

Cause: The server version and the declared dependencies were misaligned (Servlet 6 API with a Servlet 5 container).

Resolution:

1. The Tomcat version was fixed at 10.1.20.
2. The servlet and JSP dependencies were changed to Jakarta Servlet 5.0.0 and JSP 3.0.0.
3. The `web.xml` descriptor was updated to use the Servlet 5.0 schema.

6.3 JSP Compilation Errors and Unicode Issues

Problem: Accessing the book detail and success pages generated HTTP 500 errors with messages such as `book cannot be resolved to a variable` and syntax errors in JSP files. Additionally, Chinese characters in `BookDB` caused LaTeX compilation issues when documenting the code.

Causes:

- Scriptlets referenced a local variable `book` that was never declared in the JSP.
- In one case, a comment such as "Get the Book object that BookServlet put into the request" was merged into the same line as Java code, causing nonsensical expressions.
- The initial LaTeX template used pdfLaTeX and legacy font packages that could not handle CJK characters inside code listings.

Resolution:

- Scriptlet blocks were removed and replaced by JSP EL expressions like `${book.name}` and `${book.id}`.
- Comments were kept separate from Java code, and only valid Java statements were used inside any scriptlet blocks.
- The report was migrated to XeLaTeX with `fontspec` and `xeCJK`, allowing Chinese book titles to appear directly in the listings.

6.4 Memory Leak Detection Warnings

Problem: Tomcat logs emitted warnings about `ThreadLocal` and RMI target memory leak detection requiring additional `-add-opens` JVM arguments.

Resolution: It was confirmed that these warnings are informational and do not affect application correctness. They were acknowledged but not treated as blockers for this experiment.

7 Conclusion

7.1 Experimental Summary

This experiment successfully delivered a servlet-based Java web application using Visual Studio Code, Maven, and Apache Tomcat 10.1. The application implements a simple Bookstore workflow for listing books, viewing details, and simulating purchases, demonstrating the fundamentals of server-side Java web development.

7.2 Key Findings

- **Servlet–JSP Integration:** Using request attributes in combination with JSP EL offers a clean and robust way to transfer data from servlets to views.
- **Environment Alignment:** Matching container versions (Tomcat 10.1) with appropriate Jakarta dependencies and `web.xml` schemas is critical for successful deployment.
- **Deployment Reliability:** Manual WAR deployment, while simple, provides transparency and control that is helpful for learning and debugging.
- **Unicode Handling:** XeLaTeX with `xeCJK` enables professional documentation of Java code containing Chinese characters.
- **Debugging Practice:** Real-world problems such as 404 context errors and JSP compilation failures provided valuable experience in systematic troubleshooting.

7.3 Technical Validation

Technical Aspect	Status
Jakarta Servlet 5.0 Compatibility	Verified
JSP Compilation (EL-based views)	Verified
WAR Build and Packaging via Maven	Verified
Tomcat 10.1 Deployment	Verified
Application Routing and Context Path	Verified
Core Use-Case Functionality	Verified

Table 3: Summary of technical validation results.

7.4 Practical Implications

The procedures and configurations developed in this experiment form a reusable template for future Java web applications. The combination of VS Code, Maven, and Tomcat provides a modern, flexible environment for servlet-based development. Lessons learned about version compatibility, project structure, Unicode handling, and JSP best practices will directly support subsequent coursework and larger projects.

7.5 Conclusion Statement

By carefully aligning environment configuration, build management, and application code, it is possible to develop and deploy a robust Java Servlet application using contemporary Jakarta technologies and modern tooling. The experiment demonstrates not only the mechanics of Java web development but also the importance of methodical troubleshooting and clear separation between presentation and business logic.