TUTORIAL 2 – CREATE JAVA WEB APPLICATION WITH JSP & SERVLET

❖ Content:

- Create a Java Enterprise web project in IntelliJ using JSP (Java Server Pages) as front-end, Servlet as back-end and Tomcat as web server

❖ Description:

- This web app follows MVC (Model-View-Controller) design pattern. Java object works as Model, JSP works as View and Servlet works as Controller.
- In general, JSP is used as Java in HTML and Servlet is used as HTML in Java.
- For further reading about JSP & Servlet, you can access the bellow website: https://www.guru99.com/difference-between-servlets-vs-jsp.html

❖ Instructions:

1. Create a new *Java Enterprise project* in IntelliJ

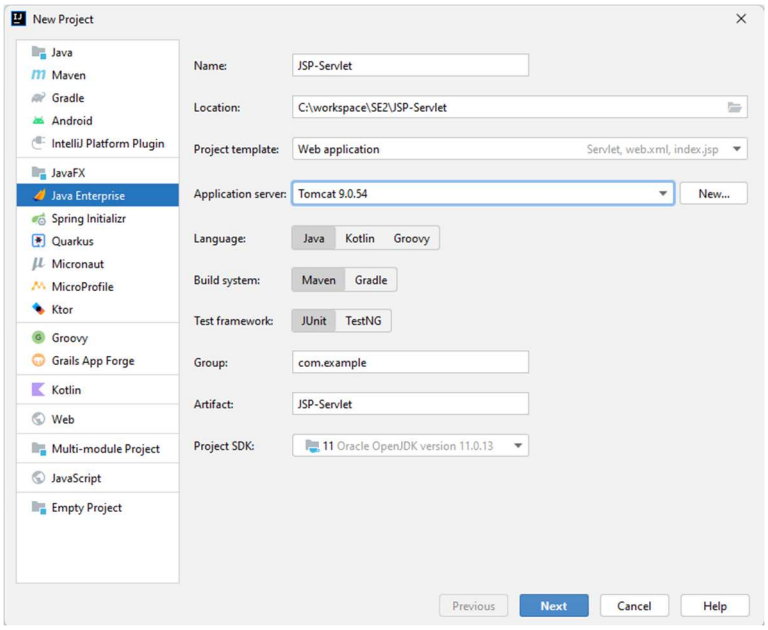

Figure 1 - Create Java Enterprise project (1)

- Project type: Java Enterprise
- Set a valid project name
- Select a suitable project location
- Set application server: Tomcat 9

Note: If you did not config Tomcat in IntelliJ, you should refer to the guide ***Environment Setup*** or move to the next step and set empty *<No application server>* at first.

- Language: Java
- Build System: Maven
- Test framework: JUnit
- Project SDK: 11
- Click **Next** to continue

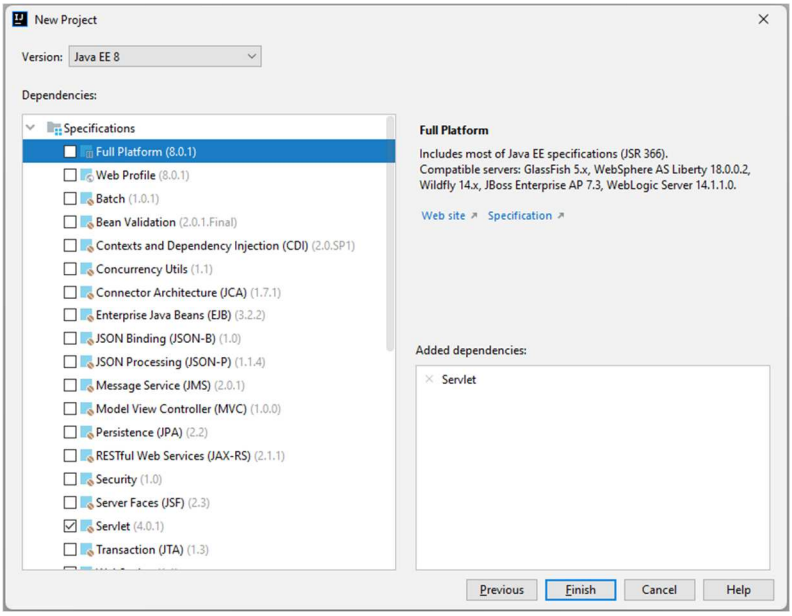

Figure 2 - Create Java Enterprise project (2)

- Version: Java EE 8
- Dependecies ⇨ Specifications: Servlet
- Click **Finish** to complete

2. Config Tomcat server in IntelliJ (ignore this if you already config before)

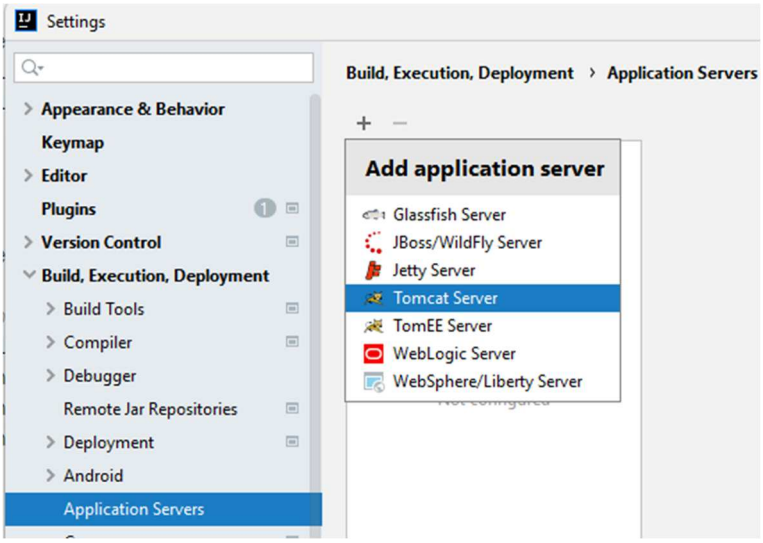

Figure 3 - Config Tomcat server (1)

- ***Note:*** *Configuration for Tomcat server is only required for the first time. From the second project, you only need to select the Tomcat server at project initialization*
- Select menu **File ⇨ Settings (Ctrl+Alt+S)**
- Select **Build, Execution, Deployment ⇨ Application Server**
- Click the **Add** button and select **Tomcat Server**

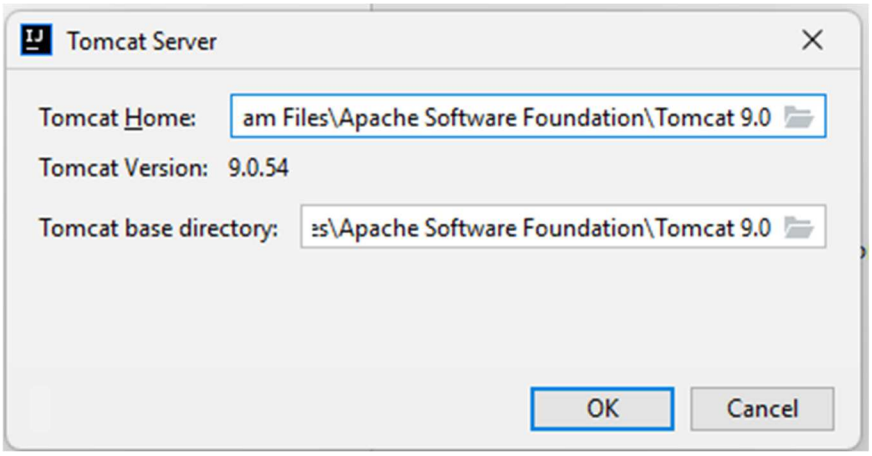

Figure 4 - Config Tomcat server (2)

- Specify the path to the Tomcat server install location

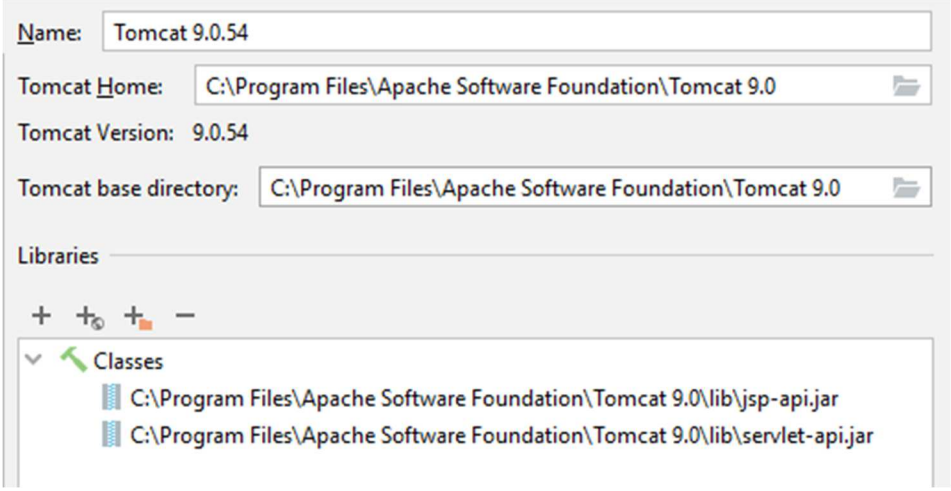

Figure 5 - Config Tomcat server (3)

- IntelliJ IDEA detects and sets the name and version appropriately

3. Config Tomcat run configuration

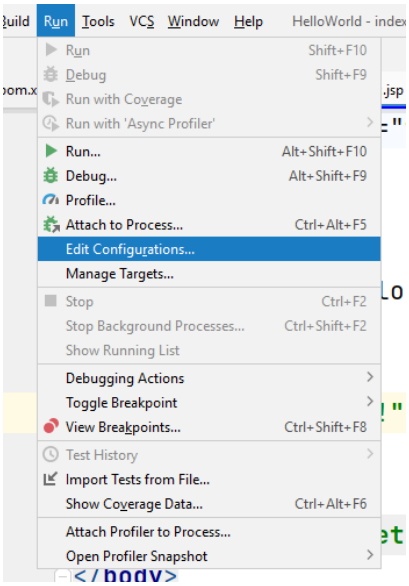

Figure 6 - Config Tomcat run configuration (1)

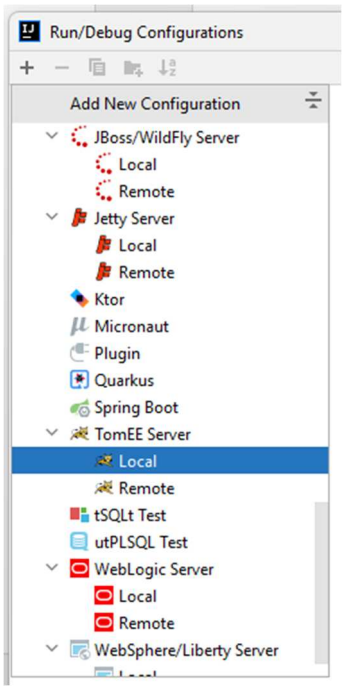

Figure 7 - Config Tomcat run configuration (2)

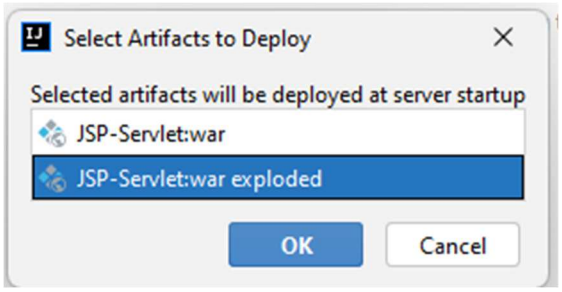

Figure 8 - Config Tomcat run configuration (3)

4. Import MySQL driver to project (*refer to Tutorial 1*)

5. Import JSTL (Java Standard Tag Library)

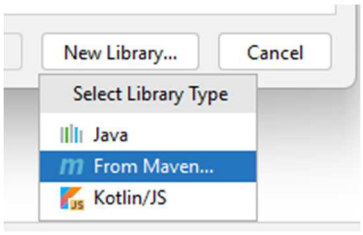

Figure 9 - Import JSTL (1)

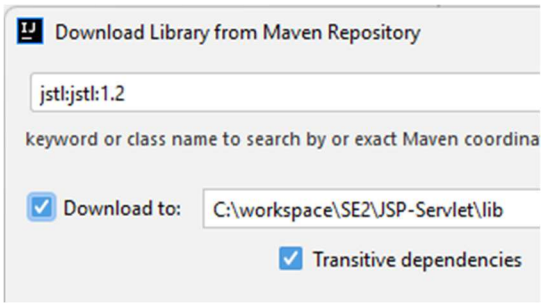

Figure 10 - Import JSTL (2)

6. Copy MySQL driver from MySQL folder & JSTL library *(jstl-1.2.jar)* from Internet to **Tomcat/lib** folder (*You need to do for only one time*)
   - JSTL link: https://repo1.maven.org/maven2/javax/servlet/jstl/1.2/jstl-1.2.jar

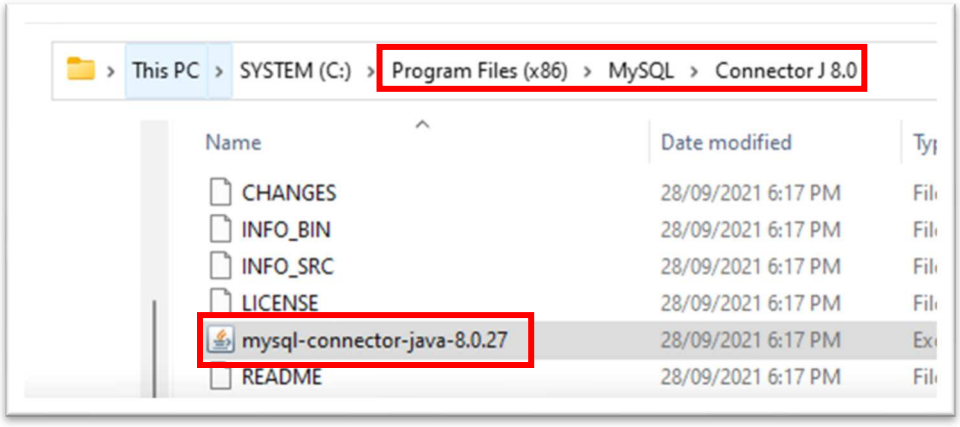

Figure 11 – Copy MySQL driver from MySQL folder

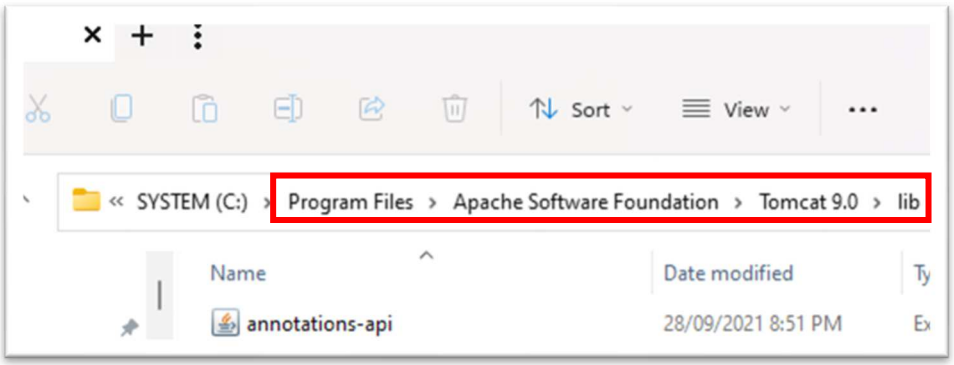

Figure 12 – Paste MySQL driver and JSTL library to Tomcat/lib folder

7. Create database with script using MySQL Workbench (*refer to Tutorial 1*)
8. Create packages and Java/Servlet class following below structure

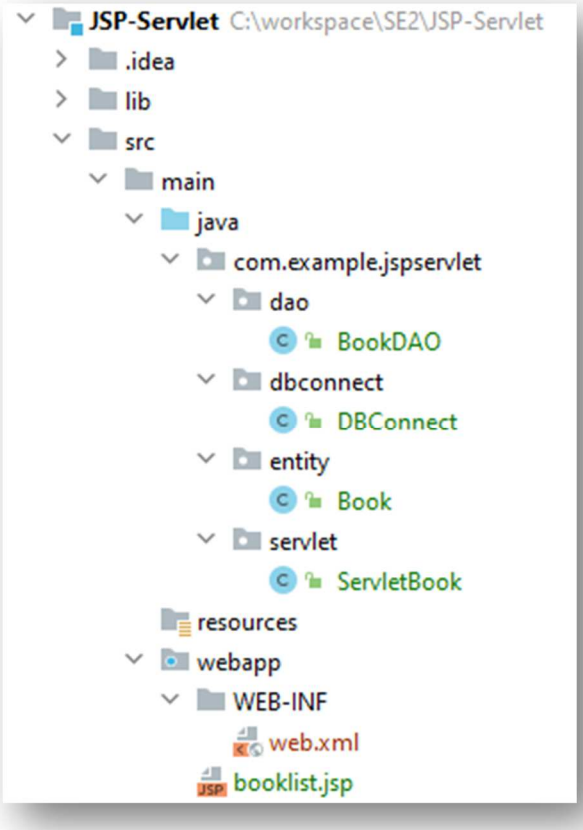

Figure 13 - Project structure

9. Create a Java class to establish connection to database

```java
public class DBConnect {
    private static final String DB_URL = "jdbc:mysql://127.0.0.1:3306/bookdb";
    private static final String DB_USERNAME = "root";
    private static final String DB_PASSWORD = "root";

    public static Connection getConnection() {
        Connection connection = null;
        try {
            connection = DriverManager.getConnection(DB_URL,DB_USERNAME, DB_PASSWORD);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return connection;
    }
}
```

Figure 14 - DBConnect.java

8. Create Java class to represent an entity for corresponding table in database

```java
public class Book {
    private int id;
    private String title;
    private String author;
    private float price;

    public Book(int id, String title, String author, float price) {
        this.id = id;
        this.title = title;
        this.author = author;
        this.price = price;
    }

    //auto-generated getters & setters
```

Figure 15 - Book.java

9. Create Java class acting as *DAO (Data Access Object)* to provide operations for the table (such as *CRUD: CREATE – READ – UPDATE – DELETE*)

```java
public class BookDAO {
    Connection connection;

    public BookDAO() {
        connection = DBConnect.getConnection();
    }

    public List<Book> selectAllBooks() {
        List<Book> bookList = new ArrayList<>();

        try {
            String sql = "SELECT * FROM book";
            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                int id = resultSet.getInt( columnIndex: 1);
                String title = resultSet.getString( columnIndex: 2);
                String author = resultSet.getString( columnLabel: "author");
                float price = resultSet.getFloat( columnLabel: "price");
                Book book = new Book(id, title, author, price);
                bookList.add(book);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return bookList;
    }
}
```

Figure 16 - BookDAO.java

10. Create Java Servlet acting as a page controller for the application to handle all requests from the client

```java
@WebServlet("/")
public class BookServlet extends HttpServlet {
    private BookDAO bookDAO;

    public void init() {
        bookDAO = new BookDAO();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
         getBookList(req, resp);
    }

    private void getBookList(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        List<Book> listBook = bookDAO.selectAllBooks();
        request.setAttribute( s: "books", listBook);
        RequestDispatcher requestDispatcher = request.getRequestDispatcher( s: "booklist.jsp");
        requestDispatcher.forward(request, response);
    }

}
```

Figure 17 - BookServlet.java

11. Create JSP page as view for corresponding Servlet inside the **webapp** folder

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!-- JSTL core -->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
    <title>Book List</title>
    <!-- Bootstrap -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
          rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
          crossorigin="anonymous">
</head>
<body>
<div class="container col-md-6 mt-3">
    <h2 class="text-success text-center">Book List</h2>
    <table class="table table-bordered text-center mt-3">
        <tr>
            <th>Book ID</th>
            <th>Book Title</th>
            <th>Book Author</th>
            <th>Book Price</th>
        </tr>
        <c:forEach var="book" items="${books}">
            <tr>
                <td> <c:out value="${book.id}" /> </td>
                <td> <c:out value="${book.title}" /></td>
                <td> <c:out value="${book.author}" /></td>
                <td> <c:out value="${book.price} $" /></td>
            </tr>
        </c:forEach>
    </table>
</div>
</body>
</html>
```

Figure 18 - booklist.jsp

12. Run the web server to test the result

Book List

Book ID	Book Title	Book Author	Book Price
1	Java Web	John	100.0 $
2	Spring Boot	David	120.0 $
3	Sofware Engineering	Tom	200.0 $

Figure 19 - Book List page

❖ What to do and submit ?

- Complete the remained operations for CRUD including CREATE, UPDATE and DELETE. You must add new methods in BookDAO and ServletBook then create new corresponding JSP files (ex: bookadd.jsp, bookedit.jsp)
- Compress whole project and submit to FIT Portal with name syntax: *FullName_StudentID_SE2_Tut2.rar*
- The complete reference source codes will be published in GitHub after homework deadline