SOFTWARE ENGINEERING LAB

EXERCISE – 5

TOPIC – 3

CREATING MAVEN WEB PROJECT USING ECLIPSE AND PUSH INTO TO GITHUB

Note: At every step take screenshots and save in a document

1. Introduction to Maven Web Projects

What is a Maven Web Project? A Maven web project is a setup for building Java-based web applications. It's structured to support servlets, JSP (JavaServer Pages), and other web resources. Maven simplifies web development by handling project structure, dependencies, and deployment tasks.

Why Use Maven for Web Projects?

- **Dependency Management**: Automatically adds the libraries your web app needs.
- **Organized Structure**: Provides a standard layout specifically for web projects.
- **Automated Builds**: Compiles, tests, and packages the project with a few commands.
- **Smooth Deployment**: Integrates with servers like Apache Tomcat, simplifying the deployment process.

2. Setting Up a Maven Web Project in Eclipse

1. **Creating a New Maven Web Project**:
    - In Eclipse, go to **File > New > Maven Project**.
    - Choose the **maven-archetype-webapp** archetype from the list.

Why Use the `webapp` Archetype?

- **Purpose**: The `webapp` archetype provides a pre-configured structure for web applications, with essential folders like `webapp/WEB-INF`.
- **Benefits**: This structure includes:
  - A `/src/main/webapp` directory for web resources (HTML, JSP, etc.).
  - A `WEB-INF` folder for configurations and dependencies hidden from direct web access.
- Using this archetype saves setup time and ensures a consistent environment for developing web applications.
- Set the **Group ID** and **Artifact ID** (identifiers for your project).
- Click **Finish** to generate the web project.

2. **Understanding Reverse Domain Naming**:
   - Maven uses the *reverse domain naming convention* to create unique project identifiers.
   - **How It Works**: If your company's domain is `mycompany.com`, you reverse it to `com.mycompany` for the **Group ID**.
   - **Example**: For a project by `example.com`, the Group ID could be `com.example.webapp`.
   - **Purpose**: This naming method is industry-standard, ensuring unique and recognizable project names.
3. **Key Files and Folders in a Web Project**:
   - **`pom.xml`**: The main configuration file for your Maven project, defining dependencies and settings.
   - **`src/main/webapp`**: Folder for your web resources, like HTML and JSP files.
   - **`WEB-INF`**: Located under `webapp`, it contains configurations and libraries that shouldn't be accessible via URL.

3. Adding Dependencies to `pom.xml`

To create a basic web application, you'll need the `servlet-api` dependency. This dependency provides the necessary classes to create servlets, which handle web requests and responses in Java.

1. **Why Add `servlet-api` from the Central Maven Repository?**
   - **Purpose**: The `servlet-api` dependency includes essential libraries for handling HTTP requests and responses, which are fundamental to web applications.
   - **Source**: By adding this dependency to `pom.xml`, Maven automatically downloads it from the *Central Maven Repository*, saving you the effort of finding and installing it manually.
2. **How to Add the `servlet-api` Dependency**:
   - Open `pom.xml` in Eclipse.
   - Under the `<dependencies>` section, add the following:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
</dependency>
```

   - **Explanation**:
     - **`groupId`** and **`artifactId`**: Identify the servlet library in Maven's central repository.
     - **`version`**: Specifies the servlet API version.
     - **`scope: provided`**: Specifies that the server (e.g., Tomcat 9) provides this library at runtime, so it doesn't need to be bundled with the application.
3. **Updating the Project**:
   - Right-click on the project, select **Maven > Update Project**. This action downloads and includes the `servlet-api` from the central Maven repository into your project.

4. Running Essential Maven Goals for a Web Project

Maven's *goals* are specific tasks that help you build and prepare your web project.

1. **Clean**:
   - **Purpose**: Deletes any old compiled files, ensuring each build is fresh.
   - **How to Run**: Right-click the project, select **Run As > Maven Clean**.
2. **Install**:
   - **Purpose**: Compiles the project, packages it as a `.war` file (Web Application Archive), and adds it to your local Maven repository.
   - **How to Run**: Right-click the project, select **Run As > Maven Install**.
3. **Package**:
   - **Purpose**: Packages the project as a `.war` file, which is needed for web deployment.
   - **How to Run**: Right-click the project, select **Run As > Maven Package**.
4. **Running All Goals Together**:
   - Enter `Clean Install Package` in the Goals field to perform these tasks in sequence, then click **Apply** and **Run**.

5. Deploying the Project on Tomcat 9

Apache Tomcat 9 is a widely-used server for running Java web applications. Here's how to deploy your Maven web project on Tomcat 9.

1. **Setting Up Tomcat 9 in Eclipse**:
   - In Eclipse, go to **Servers > New > Server**.
   - Select **Apache Tomcat v9.0** and specify the location of your Tomcat installation.
   - Complete the setup to add Tomcat 9 to your Eclipse environment.
2. **Running the Project on Tomcat 9**:
   - Right-click the project, select **Run As > Run on Server**.
   - Choose **Tomcat v9.0** as the server and start it.

- Your application will be deployed on Tomcat 9, and you can access it in a browser at a URL like `http://localhost:8080/your_project_name`.

6. Pushing the Project to GitHub

Why Use GitHub for Web Projects? GitHub allows you to store your project online, manage versions, and collaborate with others.

Steps for Pushing a Web Project to GitHub:

1. **Create a Repository on GitHub**:
   - Go to GitHub, create a new repository for your project, and select **Maven** as the `.gitignore` template.
2. **Why Use `.gitignore` for Web Projects?**
   - **Purpose**: The `.gitignore` file tells Git to ignore unnecessary files and folders, like `/target/`, which are created during the build process.
   - **Benefit**: This keeps your repository clean by excluding files that aren't needed on GitHub.
3. **Copy the Repository URL**:
   - Copy the HTTPS link to link your local project to GitHub.
4. **Push the Project Using Git Bash**:
   - Open Git Bash, navigate to your project folder.
   - Use these commands to push your project:

```
git add .                  # Adds all files
git commit -m "Initial commit"   # Commits changes with a description
git push origin main     # Pushes the project to GitHub
```

   - Refresh your GitHub page to confirm the project has been successfully uploaded.