AEM Assignment

Maven Lifecycle
Maven follows a structured sequence of phases to build and manage projects efficiently. The key phases in the lifecycle are:

1. **Clean** – Removes previous build artifacts to ensure a fresh build.
2. **Compile** – Transforms source code into bytecode.
3. **Test** – Executes unit tests to validate the code.
4. **Package** – Bundles the compiled code into a JAR or WAR file.
5. **Install** – Stores the generated package in the local Maven repository for future use.
6. **Deploy** – Uploads the package to a remote repository for distribution.

Each phase follows a dependency order, meaning running `mvn package` will automatically trigger compilation and testing before packaging.

What is pom.xml and Why is it Used?

The **pom.xml (Project Object Model)** file is the core configuration file in a Maven project. It defines project details, dependencies, plugins, and build instructions in XML format.

Why use it?

- Centralizes dependency management, avoiding manual downloads.
- Ensures consistent builds across different environments.
- Automates library retrieval and integration.
-

How Dependencies Work in Maven

Dependencies are external libraries required by your project. Instead of manually adding JAR files, you declare them in `pom.xml`. For example:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.20</version>
</dependency>
```

Maven fetches the required files from its repository and stores them in the local `.m2` directory. It also handles transitive dependencies (dependencies required by your dependencies) and resolves version conflicts automatically.

Building Multi-Module Projects with Maven

For projects with multiple modules, a **parent POM** manages dependencies and builds all submodules in a specific sequence. Running `mvn install` from the parent directory ensures all modules are built properly.

Can you build a specific module?
Yes! Instead of building the entire project, you can compile a single module using:

```
mvn install -pl module-name -am
```

This ensures that only the specified module and its dependencies are built.

Understanding AEM Folder Structure: ui.apps, ui.content, ui.frontend

In **Adobe Experience Manager (AEM)**, projects are structured into different folders:

- **ui.apps** – Contains AEM components, templates, and configurations.
- **ui.content** – Stores page content, assets, and content packages.
- **ui.frontend** – Handles front-end development, including CSS, JavaScript, and client-side libraries.

Each folder plays a distinct role in managing an AEM project efficiently.

Why Do We Use Run Modes in AEM?

Run modes help configure AEM based on the environment (e.g., development, staging, production). This allows different settings for each stage without affecting others.

Example:

- `author.dev` – Development environment settings.
- `author.prod` – Production-ready configurations.

This flexibility ensures optimal performance and security.

What is the Publish Environment in AEM?

The **publish environment** is where final content is made available to end users. Unlike the **author environment**, where content is created and edited, the publish instance is optimized for delivering content quickly and securely.

What is the Role of the Dispatcher in AEM?

The **Dispatcher** serves as both a caching mechanism and a security filter for AEM. It helps:

- Improve performance by caching frequently accessed content.
- Protect AEM publish instances by filtering unwanted requests.

It sits between users and the AEM publish instance to manage load efficiently.

How to Access CRXDE in AEM?

You can access AEM's **CRXDE Lite (Content Repository Explorer and Editor)** by navigating to:

```
http://localhost:4502/crx/de
```

For publish instances, replace `4502` with the appropriate port (e.g., `4503`). This tool allows you to manage JCR content, nodes, and configurations easily.