AEM Assignment

1. Maven Lifecycle

Maven follows a structured lifecycle to build and manage Java-based applications. The lifecycle consists of multiple phases and each has its own purposes ,

- **validate**: Ensures the project is correct and all required information is available.
- **compile**: Compiles the source code.
- **test**: Runs unit tests on the compiled code.
- **package**: Packages the project into a deployable format, such as a JAR or WAR file.
- **Install**: Install the packaged build into the local repository.
- **deploy**: Deploy the package to a remote repository for sharing.

2. POM.xml File and Its Purpose

The pom.xml (Project Object Model) file is the core configuration file in a Maven project. It defines project dependencies, plugins, goals, and build configurations.

Uses:

- Manages project dependencies.
- Controls the build process.
- Defines plugins for testing, packaging, and deployment.
- Helps maintain consistent builds across environments.

3. How Dependencies Work?

Dependencies are external libraries required for a project. They are declared inside pom.xml, and Maven automatically fetches these dependencies from central or remote repositories and adds them into the project. It also resolves transitive dependencies (dependencies of dependencies).

4. Checking the Maven Repository

Maven dependencies are downloaded from Maven Central Repository. Users can search for dependencies, check their versions, and add them to their project.

5. How All Modules Build Using Maven?

Maven supports multi-module projects. A parent pom.xml can define common configuration and list child modules using the <modules> tag. When you run a Maven

build on the parent POM, all the listed modules are built in the specified order.
To Build all modules . we can use the command (mvn clean install).

6. Can We Build a Specific Module?

Yes , Maven allows you to build a specific module using the following command ;

mvn install -pl module-name -am

pl → Project List
am → also make

7. Role of ui.apps, ui.content, and ui.frontend Folders in AEM

In AEM (Adobe Experience Manager), these folders play specific roles in Maven projects:

- ui.apps: Contains Java code, components, services, and configurations.
- ui.content: Stores site content like templates, pages, assets, and configurations.
- ui.frontend: Holds frontend files such as CSS, JavaScript, and client-side libraries.

8. Why We Use Run Modes?

Run modes define different environments (e.g., development, staging, production) in AEM. Run modes allow configurations to be environment-specific such as,

- author - For authoring instances.
- publish - For publishing content.
- dev, prod, staging - Custom modes for different environments**.**

9. What is a Publish Environment?

The publish environment in AEM is responsible for delivering content to end users. It serves as the publicly accessible version of an AEM site.

- It's paired with the author environment, where editors create and manage content before publishing it.
- Users can access the published site but cannot edit content.

10. Why We Use Dispatcher?

The Dispatcher is AEM’s caching and load-balancing tool, improving performance and security.

- Improves performance by caching pages.
- Protects AEM instances by filtering requests.

- Balances loads across multiple AEM publish instances. It ensures secure, fast, and efficient delivery of content to users.

11. How to Access CRX/DE?

CRX/DE (Content Repository Extreme Development Environment) is AEM's web-based development environment to manage JCR (Java Content Repository).

Steps to Access:

1. Open a browser.
2. Navigate to:
    - For author instance: http://localhost:portNumber/crx/de
    - For publish instance: http://localhost:portNumber/crx/de
3. Log in with AEM credentials.