AEM ASSIGNMENT 1

1) Maven life cycle
   The Maven lifecycle is structured into three distinct lifecycles: Clean, Build (Default), and Site. Each lifecycle contains specific phases, representing steps in the software development process. These lifecycles ensure a clean, standardized, and automated workflow for building, testing, and deploying applications.

2) What is pom.xml file and why we use it?
   The pom.xml (Project Object Model) file is the core configuration file in a Maven-based project. It defines project dependencies, build plugins, and other settings required for compilation, testing, and deployment. Maven uses this file to manage dependencies automatically, ensuring version control and reducing manual effort in project setup.

3) How do dependencies work?
   In Maven, dependencies are external libraries or modules that a project needs to function. These are declared in the pom.xml file under the <dependencies> section. When you build the project, Maven automatically downloads the required dependencies from a central repository and stores them in the local repository (.m2 folder). It also resolves transitive dependencies—if a dependency requires other dependencies, Maven fetches them automatically, avoiding conflicts and ensuring compatibility.

4) Check the maven repository
   The Maven repository is a storage location for libraries, dependencies, and plugins. There are three types:
   1. Local Repository : On the user's machine (default: `~/.m2/repository`), storing downloaded artifacts.
   2. Central Repository : A remote public repository (e.g., Maven Central) for common libraries.
   3. Remote Repository : Custom repositories defined in `pom.xml`. Maven checks the local repository first, then remote ones if needed.

5) How all modules build using maven?
   In a multi-module Maven project, all modules are managed under a parent pom.xml, which defines shared dependencies and configurations. Each module has its own pom.xml but inherits settings from the parent. The parent POM lists all modules in the <modules> section, ensuring they are built together. Running mvn clean install from the root directory compiles and builds all modules in the correct order based on dependencies. This approach simplifies project management, ensures consistency, and streamlines the build process.

6) Can we build specific module?
   We can build a specific module in a multi-module Maven project without building the entire project. This is useful when working on a single module to save time. You can either navigate to the module's directory and run mvn clean install or specify the module from the root using mvn clean install -pl module-name. If the module depends on other modules, adding -am ensures those dependencies are built as well. This approach optimizes the build process by focusing only on the necessary modules.

7) Role of ui.apps and ui.content and ui.frontend folder?
   In an AEM project, the `ui.apps`, `ui.content`, and `ui.frontend` folders serve distinct roles. The `ui.apps` module contains essential code, including components, templates, clientlibs, and OSGi configurations, which are deployed under `/apps`. The `ui.content` module stores initial or sample content, such as pages, assets, and configurations, placed under `/content` to set up the site structure. The `ui.frontend` module manages the front-end code, including CSS, JavaScript, and build tools like Webpack, ensuring optimized and modern styling and scripting for the AEM site.

8) Why we are using run mode?
   Run modes in AEM allow environment-specific configurations (e.g., `author`, `publish`, `dev`, `prod`). They enable the system to load different settings or content based on the environment by matching run mode-specific config files (e.g., `config.author` or `config.publish`), ensuring flexibility and separation of concerns.

9) What is publish env?

In AEM, the publish environment is where the final website is delivered to end users. It is responsible for serving content to visitors, ensuring high performance, scalability, and security. Unlike the author environment, where content is created and managed, the publish environment only allows viewing of approved and activated content. Once content is published from the author instance, it becomes available on the publish instance, which is typically secured behind a dispatcher to optimize caching and protect against unauthorized access.

10) Why are we using dispatcher?

The dispatcher in AEM acts as a caching and load-balancing layer between the publish instance and end users. It caches static content to improve performance, handles security by filtering requests, and distributes load across multiple publish instances, ensuring scalability and speed.

11) From where can access the crx/de?

In AEM, you can access CRX/DE (Content Repository eXtreme Developer Edition), the web-based interface for managing the JCR (Java Content Repository), using the following URL-
http://localhost:4502/crx/de