AEM ASSIGNMENT

Maven Life Cycle

Maven follows a set of predefined phases to manage and build projects. The key life cycle phases are:

1. Clean – Removes old build files.
2. Compile – Compiles the source code.
3. Test – Executes unit tests.
4. Package – Packages the compiled code into JAR/WAR files.
5. Install – Stores the package in the local repository.
6. Deploy – Uploads the package to a remote repository.

Each phase is dependent on the previous one, so when you execute `mvn package`, it will automatically compile and test the code before packaging.

What is pom.xml and why do we use it?

The `pom.xml` file (Project Object Model) serves as the core configuration for a Maven project. It's an XML file where you define project information, dependencies, plugins, build configurations, and other settings.
Why use it? It keeps everything structured and organized. Instead of manually managing libraries, Maven refers to `pom.xml` to fetch necessary dependencies and set up the project.

How Dependencies Work?

Dependencies are external libraries required by your project. To add them to the `pom.xml`, you define the dependency like this:

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

Maven will automatically download the dependency from the Maven repository and store it in your system's `.m2` folder. It also resolves any version conflicts and handles transitive dependencies.

How All Modules Build Using Maven?

In multi-module projects, the parent `pom.xml` connects all modules. Running `mvn install` from the parent directory will build all the modules in sequence. If one module depends on another, Maven automatically handles the build order.

Can We Build a Specific Module?

Yes, it's possible to build only a specific module without building the entire project. Use the following command to build just the desired module:
`mvn install -pl module-name -am`
This command builds the selected module and resolves its dependencies.

Role of ui.apps, ui.content, and ui.frontend Folder?

- ui.apps – Contains the AEM code (e.g., components, templates, configurations).
- ui.content – Holds content packages such as pages and assets.
- ui.frontend – Manages frontend resources like CSS, JS, and client libraries.

These folders each serve a specific purpose but together form the structure of an AEM project.

Why Are We Using Run Mode?
Run modes enable AEM to behave differently depending on the environment (development, staging, or production). You can configure settings like logging, users, and features based on the specific environment.
For example: `author.dev` and `author.prod` would represent the same AEM instance with different configurations.

What is Publish Env?
The publish environment is where content is made available to end users. Unlike the author environment (where content is created and edited), the publish environment is optimized for performance and security.

Why Are We Using Dispatcher?
Dispatcher serves as a performance enhancer and a security layer for AEM. It caches content to speed up page loading and filters requests to block malicious traffic. It acts as a reverse proxy in front of the AEM publish instance.

From Where Can We Access crx/de?
The CRXDE Lite interface can be accessed by navigating to:
`http://localhost:4502/crx/de`
This is where you manage JCR content, nodes, and configurations in AEM. For a publish instance, replace the port number with the appropriate one (e.g., 4503).