AEM Task 1

Maven Lifecycle

Maven has three built-in lifecycles:

1. **Clean:** Cleans the project (mvn clean).
2. **Default (Build):** Compiles, tests, and packages the project (mvn install).
3. **Site:** Generates project documentation (mvn site).

Each lifecycle has **phases** like validate, compile, test, package, verify, install, and deploy.

What is the pom.xml File and Why Do We Use It?

pom.xml (Project Object Model) is the core Maven configuration file that defines the project structure, dependencies, plugins, and build process.

◆ **Why use pom.xml?**

- Defines **project metadata** (name, version, packaging).
- Manages **dependencies** automatically.
- Configures **plugins** for building, testing, and deploying.
- Standardizes project structure.

How Do Dependencies Work?

Dependencies are external JAR files required by the project. They are defined in pom.xml under <dependencies>.

◆ **How it works:**

1. Maven fetches dependencies from **Maven Central** or custom repositories.
2. It **downloads and caches** them locally in the .m2 folder.
3. Dependencies are automatically added to the classpath.

Example:

xml

```
<dependencies>

        <dependency>
        <groupId>org.apache.sling</groupId>

        <artifactId>org.apache.sling.models.api</artifactId>

        <version>1.4.0</version>

        </dependency>

</dependencies>
```

How to Check the Maven Repository?

- **Local Repository:** ~/.m2/repository/ (stores downloaded JARs).
- **Remote Repository:** Maven Central (default source for dependencies).
- **Project-Specific Repository:** Defined in settings.xml.

How Are All Modules Built Using Maven?

Maven builds **multi-module projects** using the **parent-child** structure.

1. The **parent module** contains the root pom.xml.
2. The **child modules** inherit configurations from the parent.
3. Running mvn install in the parent **builds all modules**.

Can We Build a Specific Module?

Yes, using:

sh

```
mvn clean install -pl <module-name> -am
```

- -pl: Specifies the module to build.
- -am: Builds required dependencies of that module.

Role of ui.apps, ui.content, and ui.frontend Folders

1. **ui.apps** → Contains OSGi bundles, templates, and components (/apps).
2. **ui.content** → Stores content structure, DAM assets (/content).
3. **ui.frontend** → Holds frontend assets (JS, CSS, React/Vue code).

Why Are We Using Run Modes?
Run modes allow **AEM to behave differently** based on the environment (e.g., dev, stage, prod).

🔹 **Example Run Modes:**

- author → Authoring instance (4502).
- publish → Publish instance (4503).
- dev, stage, prod → Environment-specific settings.

How to set run mode?

- Via sling.properties:

sling.run.modes=author,dev

- Or via JVM option:

-Dsling.run.modes=publish,prod

What is the Publish Environment?

The **publish environment** is the **public-facing** instance where content is delivered to users (4503).

🔹 **Key features:**

- Only **approved** content is available.
- No direct editing (authoring is done in 4502).
- Integrated with **Dispatcher** for caching and security.

Why Are We Using Dispatcher?

Dispatcher is AEM's caching and security tool that improves performance and protects the publish instance.

🔹 **Why use it?**

- **Caching:** Reduces server load by serving cached content.
- **Load Balancing:** Distributes traffic across multiple instances.
- **Security:** Blocks unauthorized access, prevents attacks.

From Where Can We Access crx/de?

CRXDE Lite (AEM Developer Console) is accessed via:

http://localhost:4502/crx/de

or for publish:

http://localhost:4503/crx/de

It allows developers to **browse, edit, and manage** repository content (/apps, /content, /etc)