18/03/2025 - TASKS

- Maven life cycle
- What is pom.xml file and why we use it
- How dependencies work?
- Check the maven repository.
- How all modules build using maven
- Can we build specific module?
- Role of ui.apps and ui.content and ui.frontend folder?
- Why we are using run mode?
- What is publish env?
- Why we are using dispatcher?
- From where can access the crx/de?

Maven Lifecycle

Maven follows a structured lifecycle with different phases that execute in sequence:

1. **Validate** – Checks if the project structure and configurations are correct.
2. **Compile** – Compiles the Java source code.
3. **Test** – Runs unit tests to verify the functionality.
4. **Package** – Packages the compiled code into a JAR or WAR file.
5. **Verify** – Ensures that the package meets quality standards.
6. **Install** – Installs the packaged build into the local repository.
7. **Deploy** – Deploys the packaged application to a remote repository for use in other projects.

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

The pom.xml (Project Object Model) file is the main configuration file for a Maven project. It defines:

- **Project metadata** (name, version, description)
- **Dependencies** (external libraries required for the project)
- **Plugins** (tools that extend Maven's functionality)
- **Build settings** (compilation, testing, and packaging instructions)

It allows for **automatic dependency management**, ensuring that all required libraries are downloaded and integrated.

How Dependencies Work?

Dependencies are external libraries that a project requires to function. These are declared inside pom.xml under the <dependencies> section. Maven automatically downloads them from the **Maven repository** when the project is built.

Example of a dependency declaration in pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.apache.sling</groupId>
    <artifactId>org.apache.sling.api</artifactId>
    <version>2.20.0</version>
  </dependency>
</dependencies>
```

Checking the Maven Repository

Maven dependencies are retrieved from repositories, which can be:

- **Local Repository** – Located at ~/.m2/repository on your system. Maven first checks here before downloading from external sources.
- **Central Repository** – The default online repository, available at Maven Central.
- **Remote Repository** – Additional repositories can be specified in pom.xml to fetch dependencies from third-party sources.

How Are All Modules Built Using Maven?

AEM projects are modular, typically containing multiple submodules like core, ui.apps, and ui.content. The **all module** aggregates all these submodules. To build all modules together, run the following command in the project's root directory:

```
mvn clean install
```

This ensures that all modules are compiled, tested, and packaged together.

Can We Build a Specific Module?

Yes, you can build a specific module by navigating to its directory and running:

```
mvn clean install
```

For example, to build only the ui.apps module:

```
cd ui.apps
mvn clean install
```

This builds only that module and its dependencies without affecting other modules.

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

- **ui.apps** – Contains AEM-specific code like components, templates, and OSGi configurations.
- **ui.content** – Stores content packages such as pages, assets, and configurations.
- **ui.frontend** – Handles frontend resources like JavaScript, CSS, and client libraries used for rendering the UI.

Why Are We Using Run Modes?

Run modes allow AEM to adapt to different environments (e.g., development, testing, production) by enabling specific configurations. Instead of modifying the code, you can set environment-specific settings dynamically.

Example command to start AEM with specific run modes:

-Dsling.run.modes=author,dev

This tells AEM to use the **author instance** in **development mode**.

What is the Publish Environment?

AEM has two environments:

- **Author Environment** – Where content authors create and manage content.
- **Publish Environment** – The live instance where end-users access the published content.

After content is **activated (published)** from the author environment, it becomes available in the publish environment.

Why Are We Using Dispatcher?

The **dispatcher** is a caching and security layer in AEM that:

- **Caches pages** to reduce load on the publish instance and improve performance.
- **Secures the publish instance** by restricting direct access.
- **Balances load** between multiple publish instances.

It acts as an intermediary between users and the publish environment.

From Where Can We Access CRX/DE?

CRX/DE is AEM's developer interface for managing the **Java Content Repository (JCR)**. You can access it using:

- **Author Instance** – http://localhost:4502/crx/de
- **Publish Instance** – http://localhost:4503/crx/de

CRX/DE allows developers to browse, edit, and manage repository nodes and properties.