**Maven Lifecycle**

## Lightbox

Maven follows a predefined **build lifecycle** consisting of several phases:

1. **Validate** – Checks if the project is correct and all necessary information is available.
2. **Compile** – Compiles the source code of the project.
3. **Test** – Runs unit tests using a testing framework.
4. **Package** – Bundles compiled code into a distributable format (JAR, WAR, etc.).
5. **Verify** – Runs checks to ensure package quality.
6. **Install** – Installs the package into the local repository.
7. **Deploy** – Uploads the package to a remote repository for sharing.

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

pom.xml (Project Object Model) is the core configuration file in a Maven project. It defines:

* Project metadata (name, version, description).
* Dependencies (third-party libraries).
* Build plugins and configurations.
* Repository information.
* Build lifecycle settings.

It helps automate project builds and manage dependencies efficiently.

**How do dependencies work?**

Dependencies in Maven are managed using <dependencies> in pom.xml. Maven fetches required libraries from remote repositories and adds them to the classpath.

Example:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.9</version>

</dependency>

</dependencies>

Maven checks dependencies in this order:

1. **Local Repository** (.m2/repository/)
2. **Central Repository** (Maven Central)
3. **Remote Repository** (if configured)

**Check the Maven Repository**

Maven's official repository: [**https://mvnrepository.com**](https://mvnrepository.com)  
Use the following command to check local repository dependencies:

mvn dependency:tree

**How are all modules built using Maven?**

Maven builds multi-module projects using a **parent POM**. Running mvn install from the parent project builds all sub-modules in order.

**Can we build a specific module?**

Yes, By using:

mvn install -pl module-name -am

-pl specifies the module, -am (also make) ensures dependencies are built.

**Role of ui.apps, ui.content, and ui.frontend folders**

* **ui.apps** – Contains code, configurations, and components deployed in AEM.
* **ui.content** – Holds website content, templates, and assets.
* **ui.frontend** – Manages frontend dependencies (React, Angular, JavaScript, CSS).

**Why do we use Run Modes?**

Run modes in AEM allow different configurations for different environments (author, publish, dev, prod). Example:

-Dsling.run.modes=author,dev

**What is Publish Environment?**

The **publish** environment in AEM is where content is live and accessible to end-users. It delivers content optimized for performance.

**Why do we use Dispatcher?**

AEM Dispatcher is a caching and security layer used to:

* Cache pages for performance optimization.
* Protect AEM instances from excessive requests.
* Improve load balancing.

**From where can we access crx/de?**

We can access the **CRXDE (Content Repository Extensible Development Environment)** using:

<http://localhost:4502/crx/de/>

This allows us to browse and modify AEM repository content.