1. What is the purpose of the core module in AEM?

The core module defines the structure and logic of an application in AEM. It handles the backend operations, allowing components to interact and function properly within the application.

2. What kind of files and code can be found in the core folder?

The core folder contains the pom.xml file, which manages dependencies, configurations, and modules required to build the project. It also has a models folder for Java classes and an OSGi config folder for component configurations.

3. Explain the role of ui.apps in AEM projects.

The ui.apps module contains all the AEM components, client libraries, and dialog configurations required for the application's front-end.

4. How are components structured in the ui.apps folder?

The component structure typically looks like:

• /apps/project/components/helloworld/ → Component folder.

• /apps/project/components/helloworld/helloworld.html → HTL (HTML Template Language) script.

• /apps/project/components/helloworld/_cq_dialog.xml → Dialog configuration file.

5. Hello World Component:

• Where is the Hello World component located in both core and ui.apps?

The Java file (backend logic) for Hello World is located in core/models, while the front-end HTML and dialog configuration files are in ui.apps/components.

• Explain the Java class (in core) for the Hello World component.

The Java class, HelloWorldModel, has a private variable called message and a method called init(). This method appends the current resource path to the message variable. The method is protected, allowing only subclasses to access it.

• How does the HTML script work in ui.apps for Hello World?

The HTML script acts as a front-end template using HTL, which loads content from the core Java files and displays it on the web page.

• How are properties and dialogs defined for this component?

Properties and dialogs are defined using the _cq_dialog.xml file, which configures fields like text input, drop-downs, or other editable content for the component.

6. What are the different types of AEM modules (core, ui.apps, ui.content, etc.)?

The different AEM modules are:

• all – Combines all modules to create a complete package.

• core – Contains the backend Java logic.

• ui.frontend – Manages front-end frameworks like React, Angular, etc.

• ui.apps – Handles AEM components and client libraries.

• ui.apps.structure – Defines folder structures for content.

• ui.config – Contains configuration settings.

• ui.content – Manages the content structure.

• it.tests – Handles integration testing.

• dispatcher – Manages content delivery via a web server.

• ui.tests – Performs user interface testing.

7. How does Maven build these modules?

Maven uses the pom.xml file in the root directory, which contains all modules and their dependencies. Maven automates the building process, ensuring that all modules are compiled and packaged correctly.

8. Explain the build lifecycle of Maven in the context of AEM.

The Maven build lifecycle includes:

• clean – Clears previous build files.

• compile – Converts source code into class files.

• package – Bundles the compiled files into a deployable format.

• install – Installs the package into the local repository.

• deploy – Uploads the package to the AEM server.

9. How are dependencies managed in pom.xml?

Dependencies are managed using the <dependencies> section in the pom.xml file. Maven automatically downloads and installs the required libraries during the build process.

10. Why is Maven used instead of other build tools?

Maven is preferred due to its vast library support, easy configuration through pom.xml, and ability to build projects on any platform with minimal configuration.

11. What advantages does Maven offer for AEM development?

Maven simplifies the AEM build process, automates dependency management, and offers a structured approach for building scalable and compatible applications.

12. How does Maven help in managing dependencies and plugins in AEM projects?

Maven uses the pom.xml file to organize and install dependencies and plugins in a structured manner, ensuring that the required libraries and plugins are available for the build process.

13. What does mvn clean install do in an AEM project?

The mvn clean install command cleans the previous build files, installs all necessary dependencies, and compiles the project using the pom.xml file. It then packages the application and deploys it to the local repository.

14. How to deploy packages directly to AEM using Maven commands?

You can deploy packages directly to AEM using the command:

nginx

CopyEdit

```
mvn clean install -PautoInstallPackage
```

This will clean, build, and deploy the package to the AEM instance.

15. Explain the purpose of different Maven profiles in AEM (autoInstallPackage, autoInstallBundle).

• autoInstallPackage – Deploys the entire package, including components and content.

• autoInstallBundle – Deploys only the backend Java code (core).

16. What is the purpose of dumplibs in AEM?

Dumplibs are used to debug and test client libraries (JS and CSS) in AEM projects.

17. How can you view client libraries using dumplibs?

You can access the dumplibs page via:

ruby

CopyEdit

http://localhost:4502/libs/granite/ui/content/dumplibs.html

This page shows all client libraries used in the project.

18. Explain how client libraries are structured in AEM.

The client libraries are structured like this:

swift

CopyEdit

/apps/project/clientlibs

├── js.txt → Lists JavaScript files.

├── css.txt → Lists CSS files.

├── allowProxy → Allows public access via `/etc.clientlibs/`.

The allowProxy option ensures that client libraries are accessible through a secure URL