Here’s a set of AEM (Adobe Experience Manager) interview questions tailored for someone with 3 years of experience. Each question is followed by a concise answer.

Basic Questions

1. **What is AEM, and why is it used?**  
   **Answer:**  
   Adobe Experience Manager (AEM) is a comprehensive content management solution for building websites, mobile apps, and forms. It allows organizations to create, manage, and deliver personalized digital experiences across various channels.
2. **What are the key features of AEM?**  
   **Answer:**
   * Digital Asset Management (DAM)
   * Content Fragments and Experience Fragments
   * Editable Templates
   * Integration with Adobe Marketing Cloud
   * Multi-site Management
   * Personalization with Target and Analytics
3. **What is the role of JCR in AEM?**  
   **Answer:**  
   The Java Content Repository (JCR) is a hierarchical content repository used in AEM to store and retrieve content. It supports versioning, access control, and structured and unstructured content storage.

**Sling and OSGi**

1. **What is the Sling framework in AEM?**  
   **Answer:**  
   Sling is a web framework used in AEM to map HTTP requests to content resources based on the request's URL. It operates on the principle of RESTful architecture and simplifies access to the JCR.
2. **What are Sling Models? How are they used?**  
   **Answer:**  
   Sling Models are POJOs (Plain Old Java Objects) annotated with @Model to map the properties of AEM resources or requests to Java objects. They are used to encapsulate business logic and simplify backend development for HTL templates.
3. **What is OSGi, and how is it used in AEM?**  
   **Answer:**  
   OSGi (Open Services Gateway Initiative) is a modular framework that AEM uses to manage dependencies and services dynamically. OSGi components are declared as bundles, which can be started, stopped, or updated independently.

**Development and Configuration**

1. **What is the difference between WCMUsePojo and Sling Models? Why should WCMUsePojo be avoided?**  
   **Answer: A screenshot of a web page

   Description automatically generated**

**Why Should WCMUsePojo Be Avoided?**

1. **Deprecated in Modern AEM Versions**
2. **Boilerplate Code :** overriding methods like activate() or init()
3. **Tight Coupling**
4. **Poor Testability**
5. **Performance**
6. **Lack of Flexibility**

**Advantages of Sling Models Over WCMUsePojo**

* **Declarative and Cleaner Code**
* **Integration**
* **Future-Proof**
* **Enhanced Functionality**

1. **How do you create a custom OSGi service in AEM?**  
   **Answer:**

**Define the Service Interface**

* + Create a Java interface to define the service's contract. This interface outlines the methods your service will expose.

**Implement the Service**

* + Create a class that implements the service interface. Use the @**Component** annotation to register the class as an OSGi service and the @**Service** annotation to specify the interface it implements.

**Inject the Service Where Needed**

* + Use @Reference to inject the service into other components or servlets.

1. **How do you configure a dialog in Touch UI?**  
   **Answer:**
   * Use the cq:dialog node under the component's folder in CRXDE.
   * Define fields using Coral UI components (e.g., text, number, dropdown).
   * Save the dialog as a cq:dialog XML or in code repositories.

**Advanced Questions**

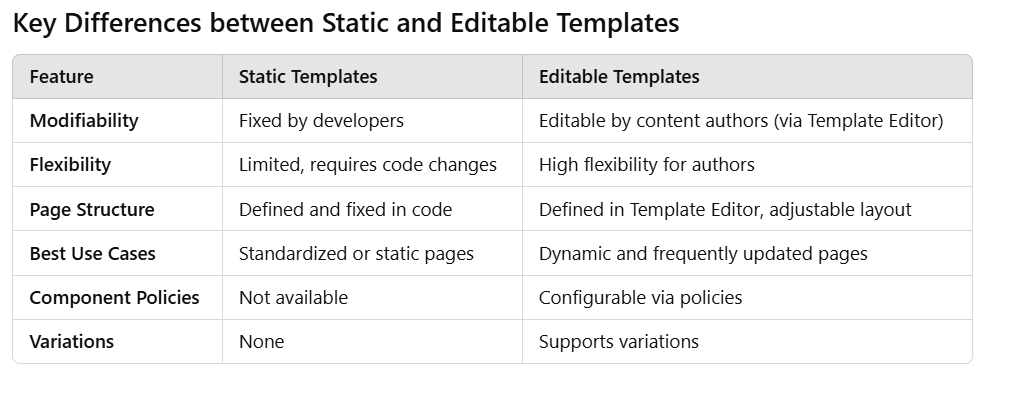
1. **What is the purpose of editable templates in AEM?**  
   **Answer:**  
   Editable templates allow authors to define the structure of a page, including allowed components and layouts, while giving flexibility to modify content on individual pages without affecting the template.

**Static Templates**

* **Overview**: Static templates are predefined and coded by developers using the **HTL (HTML Template Language)** or **JSP** and stored within AEM’s codebase. They are created in the traditional **JCR** (Java Content Repository) and are only modifiable by developers, not by content authors.
* **Structure**: Each static template defines a fixed structure for a page, with specified components and layout.
* **Usage**: Since these templates are not editable, they are best suited for pages where the layout and components need to remain consistent, such as certain landing pages or standardized pages.
* **Customization**: Content authors cannot customize or modify the structure of static templates. Any changes require developer intervention.
* **Limitations**:
  + Limited flexibility for authors.
  + Cannot be modified through the AEM Touch UI editor, only through code.
  + Not recommended for projects needing frequent layout adjustments.
* **Example**:
  + A static home page with a banner, text, and footer section, where the layout remains unchanged across instances.

**Editable Templates (Dynamic Templates with Touch UI)**

* **Overview**: Editable templates were introduced with AEM’s Touch UI and are managed through the **Template Editor**. They allow content authors to create and customize page layouts in a more flexible and user-friendly way.
* **Structure**: Editable templates are divided into **Initial Content** and **Layout**:
  + **Initial Content**: Defines the default components (text, images, etc.) that will appear when a new page is created.
  + **Layout**: Defines the structure of the page and can be locked by template authors if certain sections should remain static.
* **Usage**: Editable templates are ideal for dynamic and flexible page layouts where content authors need the ability to adjust sections, add or remove components, and create variations.
* **Customization**: Template authors can control which components are available and which areas of the page can be edited by content authors. Layouts can be reused across multiple pages.
* **Variations**:
  + Content authors can create **variations** of editable templates to cater to specific needs (e.g., different layouts for desktop vs. mobile).
* **Policies**:
  + Editable templates come with **policies** that allow defining rules or properties for components, such as maximum image size, allowed text characters, etc.
* **Example**:
  + A content-rich page template where authors can customize layout sections, insert components like banners, text, and images, and create tailored experiences for each page.



**\*What are the differences between Content Fragments and Experience Fragments?\***  
**Answer:**

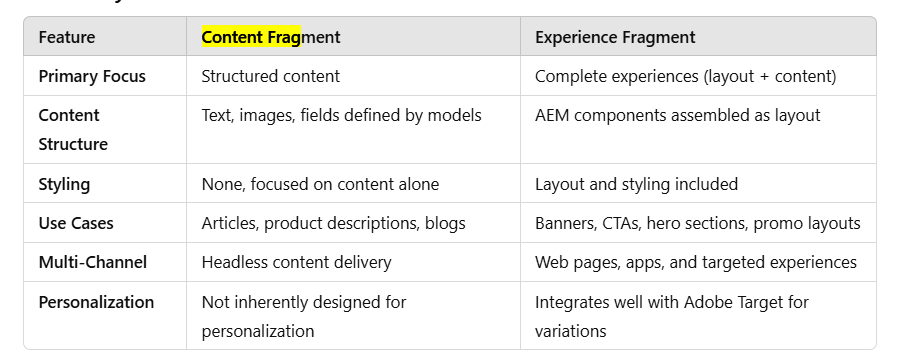
* + **Content Fragments:** Structured content, primarily text and associated metadata, reusable across channels.
  + **Experience Fragments:** Visual and layout-specific content that includes components, images, and text, designed for specific channels.

**1. Content Fragment**

* + **Purpose**: Primarily used for creating and managing structured, reusable text-based content.
  + **Usage**: Ideal for situations where you need consistent, modular content that can be used across multiple channels (web, mobile apps, etc.).
  + **Structure**: Based on a predefined **Content Fragment Model**, which defines fields (text, images, dates, etc.).
  + **Content Focus**: Allows authors to create pure content without layout or style, focusing on the text and media itself.
  + **Channels**: Suitable for multi-channel delivery. Content can be consumed via APIs (e.g., GraphQL or JSON), making it versatile for headless delivery.
  + **Use Cases**:
  + Blog posts, product descriptions, articles.
  + Content meant for delivery across various platforms and devices.
  + **Authoring Experience**: Provides in-line editing and flexibility for creating multiple variations within a fragment for different needs or locales.

**2. Experience Fragment**

* + **Purpose**: Used for creating complete page experiences that are reusable and styled.
  + **Usage**: Ideal for scenarios where content needs to include layout, style, and potentially complex components, such as images, buttons, or forms.
  + **Structure**: Experience fragments are assembled using AEM components, similar to a web page layout, but saved as reusable fragments.
  + **Content & Layout Focus**: Contains content and layout/styling, as well as links to AEM components like banners, CTAs, etc.
  + **Channels**: Can be used within AEM pages, mobile apps, or pushed to other channels. Can integrate with Adobe Target for personalization.
  + **Use Cases**:
  + Reusable page sections, such as hero banners, promotional sections, or calls-to-action.
  + Variations for personalized experiences, such as different layouts or images for various customer segments.
  + **Authoring Experience**: Allows authors to create entire page sections that are visually structured and style-complete for various channels or audiences.



1. **How do you manage user permissions in AEM?**  
   **Answer:4**  
   User permissions are managed through user groups in CRXDE or AEM's Admin Console, where read, write, modify, and delete permissions are granted at the node level.
2. **What is a dispatcher in AEM?**  
   **Answer:**  
   The dispatcher is AEM's caching and load balancing tool. It stores cached content to reduce load on the publish instance and ensures efficient delivery of content.

**Practical Scenario Questions**

1. **How would you debug a situation where an AEM component does not load properly?**  
   **Answer:**
   * Check the browser console for JavaScript errors.
   * Review the server logs (error.log or request.log).
   * Validate the resource type mapping and check if the correct component path is referenced.
   * Verify the Sling Model or backend logic for errors.
2. **Explain the steps for migrating components from Classic UI to Touch UI.**  
   **Answer:**
   * Identify components using Classic UI.
   * Recreate dialogs using Touch UI standards.
   * Update client libraries to Coral UI framework.
   * Test and validate dialogs in authoring mode.
3. **How do you ensure no duplicate links are present in an AEM-generated report?**  
   **Answer:**  
   Use a Set collection in Java to store links, ensuring uniqueness. Before writing links to the report, check if they already exist in the set.

**Behavioral and Teamwork**

1. **Describe a challenging AEM migration issue you solved.**  
   **Answer:**  
   Example: Resolving compatibility issues during an upgrade by identifying deprecated APIs, replacing them with newer alternatives, and performing thorough testing to ensure all functionality worked as expected.
2. **How do you prioritize tasks when working on multiple AEM projects?**  
   **Answer:**  
   Prioritize tasks based on deadlines, criticality, and dependencies. Use Agile tools like Jira to track progress and ensure efficient time management.
3. **How do you ensure code quality in AEM projects?**  
   **Answer:**
   * Follow coding standards (e.g., Clean Code principles).
   * Perform regular code reviews.
   * Use tools like SonarQube for static code analysis.
   * Write unit tests and integration tests.

**Tricky Questions**

1. **What would you do if an AEM servlet does not return the expected response?**  
   **Answer:**
   * Check the servlet's mapping (sling.servlet.paths or sling.servlet.resourceTypes).
   * Debug the service logic for errors.
   * Verify permissions and ensure the servlet has access to required resources.
2. **Explain the difference between replication and reverse replication in AEM.**  
   **Answer:**
   * **Replication:** Content is pushed from the author instance to the publish instance.
   * **Reverse Replication:** Content is pulled from the publish instance back to the author instance, often used for user-generated content.  
       
       
       
     **What is CMS (Content Management System)?**
   * A **Content Management System (CMS)** is a software platform that allows users to create, manage, and modify digital content (such as text, images, videos, and documents) on a website or application without requiring extensive technical knowledge.  
       
     **Key Features of a CMS:**
   * **Content Creation:** Easy-to-use interfaces for creating and editing content, such as WYSIWYG editors.
   * **Content Management:** Tools for organizing, storing, and retrieving content.
   * **Publishing:** Workflow mechanisms to publish approved content.
   * **Customization:** Support for templates, themes, and plugins for design and functionality.
   * **Version Control:** Tracks changes and maintains previous versions of content.
   * **User Roles and Permissions:** Manage access for authors, editors, and administrators.
3. **What** **are Sling Jobs in AEM?**
   * Sling Jobs in **Adobe Experience Manager (AEM)** are like tasks or work items that run in the background. They are part of the **Apache Sling Event and Job Handling** framework, which is designed to handle asynchronous operations.
   * Think of Sling Jobs as a to-do list for AEM where tasks are queued up and processed one by one (or concurrently, depending on the configuration). These tasks are executed outside the main user interface, allowing heavy or time-consuming processes to run without slowing down the system.
   * **Why Use Sling Jobs?**
   * You use Sling Jobs when you need AEM to perform tasks like:
   * Sending emails in response to user actions.
   * Processing or resizing images.
   * Syncing data between AEM and another system.
   * Performing scheduled maintenance tasks (like cleanup).
   * Indexing or reindexing large amounts of content.
   * These tasks don't have to be completed immediately while the user is waiting—they can be done in the background.
   * **How Do Sling Jobs Work?**

**Job Creation**:

* + A developer creates a Sling Job and adds it to a queue.
  + Each job has a **type** (what kind of task it is) and **properties** (data the job needs to run).

**Job Queues**:

* + Jobs are placed into **queues**, which are managed by the Sling framework.
  + Queues determine how and when jobs are executed.

For example:

* + **Parallel Queue**: Runs multiple jobs at the same time.
  + **Ordered Queue**: Runs jobs one after the other in the order they were added.

**Job Execution**:

* + A worker thread picks up the job from the queue and processes it.
  + The job runs the logic defined by the developer (e.g., sending an email or resizing an image).

**Job Completion**:

* + Once the job finishes, it is marked as completed.
  + If it fails, the job can be retried automatically based on the queue configuration.

**Example of a Sling Job in AEM:**

* + Let’s say you have a website where users upload profile pictures. You want AEM to resize these pictures for faster loading on your site.
  + **Step 1**: When a user uploads an image, a Sling Job is created with the image details.
  + **Step 2**: The job is added to the “image-processing” queue.
  + **Step 3**: The queue picks up the job and runs a script to resize the image.
  + **Step 4**: Once done, the resized image is saved, and the job is marked as complete.

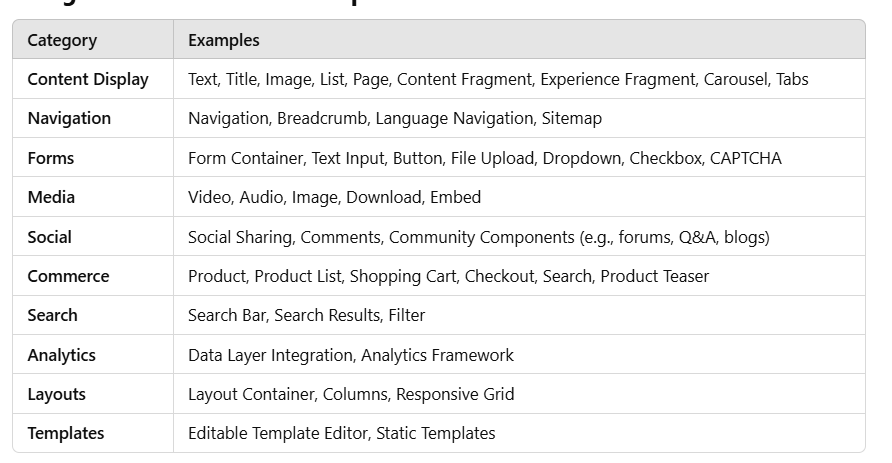
**Key Features of Sling Jobs:**

* + **Asynchronous Processing**: Jobs run in the background, freeing up the user interface.
  + **Retry Mechanism**: Failed jobs can be retried automatically.
  + **Scalable**: Supports parallel job execution for faster processing.
  + **Flexible Configuration**: You can configure queues to suit your needs (e.g., set priorities or limits).
  + **Where Are Sling Jobs Useful in AEM?**
  + Automated workflows like sending notifications or emails.
  + Backend processes like syncing with external systems.
  + Maintenance tasks, such as purging old data.
  + High-volume operations like generating reports or processing large batches of content.

1. What are the out of the box components in AEM ?

Out-of-the-box (OOTB) components in AEM are pre-built components provided by Adobe Experience Manager that can be used directly or customized to create web pages and other digital experiences. These components reduce development time and ensure consistency in design and functionality.

**Categories of AEM OOTB Components**

****

**Notable OOTB Components**

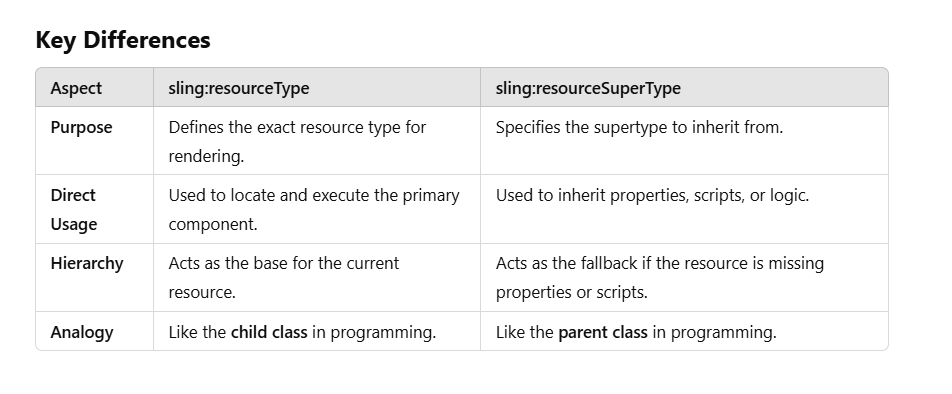
1. **Text Component**: For adding rich text with WYSIWYG editing capabilities.
2. **Image Component**: For adding and resizing images with support for metadata and renditions.
3. **Carousel/Slideshow**: For displaying rotating content like banners.
4. **Tabs/Accordion**: To organize content into collapsible or tabbed sections.
5. **Form Components**: For creating forms with validation and submission logic.
6. **Experience Fragment**: To reuse content across different pages and channels.
7. **Content Fragment**: For managing and reusing structured content (e.g., article summaries, descriptions).
8. **Navigation**: For rendering menus and site structure.
9. **Language Navigation**: To enable multilingual site navigation.
10. **Social Sharing**: For integrating sharing buttons for platforms like Facebook, Twitter, and LinkedIn.

**Advantages of OOTB Components**

1. **Time-Saving**: Reduces the need for custom development.
2. **Consistency**: Ensures uniform styling and behavior.
3. **Customizable**: Can be extended or modified to meet specific project needs.
4. **Integration**: Often integrates with other Adobe products and tools seamlessly.

**How to Use OOTB Components**

1. Drag and drop OOTB components into a page in the AEM authoring environment.
2. Configure the component properties using the dialog box.
3. Style them using CSS or customize them as per your requirements.
   1. These components are part of AEM’s **Core Components** or **Foundation Components**, depending on the version of AEM you're using. Core Components are recommended for new projects as they are more flexible, feature-rich, and regularly updated by Adobe.
4. What is the difference between sling:resourcetype and sling:resourceSuperType in AEM ?

**Question:In which scenario you have worked in schedulars in AEM? give me scenario which i can explain in the interview .**

**Scenario: Automating Weekly Email Reports**

**1. Start with the Objective (Why it was needed):**

"At my previous project, the team required a system to automate weekly reporting of activities in our Adobe Experience Manager (AEM) platform. This included tracking new pages created, updates to existing pages, and assets uploaded to the Digital Asset Manager (DAM). The goal was to ensure stakeholders received timely and accurate insights without manual intervention."

**2. Explain Your Approach (How you solved it):**

**"I implemented an AEM Scheduler that generates and emails a weekly activity report. Here's how it worked:**

* Scheduling: I used AEM's Scheduler service with a cron expression to automatically run every Monday at 8:00 AM.
* Report Generation: The job gathers weekly statistics, such as new pages, updates, and uploads, and compiles them into a structured text-based report.
* Email Integration: The report is sent to relevant stakeholders via email using AEM's MailService.
* Logging and Error Handling: I integrated detailed SLF4J logs to ensure we could monitor the job execution and troubleshoot any failures."

**3. Highlight Your Contribution (Technical Implementation):**

"I wrote the job class using OSGi annotations for dependency injection and modularity. I used the following tools and practices:

* MailService API: For sending emails programmatically.
* SLF4J Logging: To track execution flow and identify errors in production.
* Cron Expression: To define the precise schedule for the job.
* Code Scalability: The logic was designed to be easily extended—for example, adding new data points to the report or enhancing email formatting."

**4. Discuss the Impact (What it achieved):**

"This solution automated a repetitive task, saving several hours of manual work every week. It also ensured consistency in reporting and improved visibility into our content management activities. Stakeholders received accurate reports on time, enabling better decision-making."

package com.example.aem.schedulers;

import org.apache.sling.api.servlets.SlingAllMethodsServlet;

import org.apache.sling.api.resource.Resource;

import org.apache.sling.api.resource.ResourceResolver;

import org.apache.sling.api.servlets.Servlet;

import org.osgi.framework.Constants;

import org.osgi.service.component.annotations.Activate;

import org.osgi.service.component.annotations.Component;

import org.osgi.service.component.annotations.Reference;

import org.osgi.service.scheduler.Job;

import org.osgi.service.scheduler.JobContext;

import org.osgi.service.scheduler.ScheduleOptions;

import org.osgi.service.scheduler.Scheduler;

import org.apache.sling.api.mail.MailService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.text.SimpleDateFormat;

import java.util.Calendar;

@Component(service = Job.class,

immediate = true,

property = {

Constants.SERVICE\_DESCRIPTION + "=Weekly Report Email Sender",

"scheduler.expression=0 0 8 ? \* MON" // Cron expression: every Monday at 8:00 AM

})

public class WeeklyReportScheduler implements Job {

private static final Logger LOG = LoggerFactory.getLogger(WeeklyReportScheduler.class);

@Reference

private MailService mailService;

@Reference

private Scheduler scheduler;

@Activate

protected void activate() {

LOG.info("WeeklyReportScheduler activated.");

// Scheduler is triggered automatically based on the expression

}

@Override

public void execute(JobContext context) {

LOG.info("WeeklyReportScheduler execution started.");

try {

// Step 1: Gather data for the report

LOG.debug("Generating the weekly report...");

String report = generateWeeklyReport();

LOG.info("Weekly report generated successfully.");

// Step 2: Send the report via email

LOG.debug("Sending the weekly report via email...");

sendEmailReport(report);

LOG.info("Weekly report sent successfully.");

} catch (Exception e) {

// Log failure

LOG.error("Error generating or sending the report.", e);

}

}

private String generateWeeklyReport() {

LOG.debug("Starting report generation...");

StringBuilder reportBuilder = new StringBuilder();

reportBuilder.append("Weekly Report\n");

reportBuilder.append("===================\n");

// Sample content generation

reportBuilder.append("New Pages Created: 5\n");

reportBuilder.append("Pages Updated: 3\n");

reportBuilder.append("Assets Uploaded: 2\n");

// Include the date for clarity

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

reportBuilder.append("\nReport Generated on: ").append(dateFormat.format(Calendar.getInstance().getTime()));

LOG.debug("Report generation completed.");

return reportBuilder.toString();

}

private void sendEmailReport(String reportContent) {

String subject = "Weekly AEM Report";

String[] recipients = {"team@example.com"};

try {

LOG.debug("Sending email to recipients: {}", (Object) recipients);

mailService.sendMail("no-reply@example.com", recipients, subject, reportContent);

LOG.debug("Email sent successfully.");

} catch (Exception e) {

LOG.error("Error sending email.", e);

}

}

}

Certainly! Here's the flow of the **WeeklyReportScheduler**:

1. **Component Setup**:
   * The @Component annotation marks this class as an OSGi component.
   * It is configured to run as a job (service = Job.class) and is activated immediately (immediate = true).
   * A scheduler.expression is provided to trigger the job every Monday at 8:00 AM based on the cron expression.
2. **Activation Phase**:
   * The activate method logs the activation of the scheduler.
   * The job will automatically execute at the defined schedule, so no manual setup is needed here.
3. **Job Execution**:
   * The execute method is triggered when the scheduler runs. It serves as the entry point for the weekly report generation and email-sending process.
   * The flow begins by logging the start of the job.
4. **Report Generation**:
   * The generateWeeklyReport method is called to prepare the weekly report.
   * It constructs the report as a plain-text string with details like:
     + Number of new pages created.
     + Number of pages updated.
     + Number of assets uploaded.
   * The current date and time are appended to indicate when the report was generated.
   * The process is logged at every step for debugging and monitoring.
5. **Email Sending**:
   * The sendEmailReport method is responsible for sending the report via email.
   * It uses the AEM MailService API to send an email with:
     + A subject line ("Weekly AEM Report").
     + The list of recipients (team@example.com in this case).
     + The content of the generated report.
   * Logs indicate the start, success, or failure of this step.
6. **Error Handling**:
   * The try-catch blocks in the execute and sendEmailReport methods ensure any errors during report generation or email sending are logged with detailed error messages.
   * This aids in troubleshooting if the scheduler encounters issues.
7. **Logging**:
   * SLF4J is used throughout the class to log various stages of execution:
     + INFO logs for high-level progress like activation and successful execution.
     + DEBUG logs for detailed insights into internal operations, such as generating reports and sending emails.
     + ERROR logs to capture and describe any issues.

**Overall Flow**: When the scheduler runs (every Monday at 8:00 AM):

1. The execute method is triggered.
2. A report is generated using generateWeeklyReport.
3. The report is sent via email using sendEmailReport.
4. Logs track the progress, successes, or failures at every step.

**Test the Scheduler**

Once deployed to AEM, the scheduler will automatically run every Monday at 8:00 AM, generating the report and sending it via email to the configured recipients.

**Question :How you schedualr linked with email?**

After the report is generated, the job uses **AEM's Mail Service** to send the report to the people who need it. The report content is set as the body of the email. The system automatically sends the email to a predefined list of recipients, like your team or managers.

**Question: In osgi service what are the inputs you have given in AEM ?**

 Service Interface:  
Think of this as a blueprint or contract. It defines what the service will do, like a list of tasks the service promises to handle.

 Service Implementation:  
This is where the actual work gets done. It’s the class that takes the blueprint (interface) and turns it into real functionality.

 @Designate Annotation:  
This allows you to link configurations to your service. It’s like saying, “Here’s the setup file this service needs to follow.”

 @Activate Annotation:  
This marks the starting point of your service. When the service is turned on, this method runs to set things up.

 @Deactivate Annotation:  
This is the cleanup method. It’s called when the service is turned off, ensuring everything is tidied up properly.

 @Reference Annotation:  
This is like plugging in a charger. It connects your service to other services it needs to work, automatically providing the required dependencies.

**Question : what are the things required to create an editable template ?**

1. **Template Type**: A predefined template type that specifies the structure and allowed components for templates (e.g., Page Template or Content Fragment Template).
2. **Policy Configuration**: Define policies to control the behavior of components, such as styles, allowed components, and other settings.
3. **Structure**: The static framework of the page, including header, footer, and placeholders (defined in the **Layout Container**).
4. **Initial Content**: Default content to prefill the template, such as logos, titles, or text.
5. **Editable Areas**: Layout containers where authors can add, edit, or remove components while creating pages.
6. **Permissions**: Ensure proper user permissions for creating and managing templates (usually granted to template authors or admins).
7. **Component Mapping**: Use the **Sling Resource Type** to map components to the template.

**Steps Overview:**

1. Go to **Tools > Templates** in AEM.
2. Create a new **Template Type** if one doesn’t exist.
3. Add a **New Template** using the template type.
4. Define the structure, policies, and initial content.
5. Enable the template for use.

**Annotations :-**

**AEM Sling Models**

1. **@Model**
   * Declares the class as a Sling Model.
   * Specifies adaptable types such as Resource or SlingHttpServletRequest.

@Model(adaptables = Resource.class)

public class MyModel {

// ...

}

1. **@Inject**
   * Used to inject properties, services, or OSGi configurations into a model.

@Inject

private String title;

1. **@ValueMapValue**
   * Retrieves properties directly from the JCR using a ValueMap.

@ValueMapValue

private String description;

1. **@Default**
   * Provides a default value if the injected property is null or missing.

@Inject

@Default(values = "Default Title")

private String title;

1. **@Optional**
   * Marks a property as optional to avoid exceptions if it is missing.

@Inject

@Optional

private String subtitle;

1. **@Self**
   * Injects the adaptable itself (e.g., Resource or SlingHttpServletRequest).

@Self

private Resource resource;

1. **@ScriptVariable**
   * Injects global objects from the current request context (e.g., currentPage, resourceResolver).

@ScriptVariable

private Page currentPage;

1. **@Source**
   * Specifies a different injector source (e.g., @Source("osgi-services")).

java

Copy code

@Inject

@Source("osgi-services")

private MyService myService;

1. **@Exporter**
   * Used for exporting Sling Models as JSON objects.

java

Copy code

@Model(adaptables = Resource.class, resourceType = "example/components/content")

@Exporter(name = "jackson", extensions = "json")

public class MyModel {

// ...

}

1. **@PostConstruct**
   * Marks a method to be executed after the model is initialized and all dependencies are injected.

java

Copy code

@PostConstruct

private void init() {

// Initialization logic

}

1. **@ChildResource**
   * Injects child resources under the current resource.

java

Copy code

@ChildResource

private List<Resource> items;

1. **@Via**
   * Specifies an alternative path to resolve the injected value.

java

Copy code

@Inject

@Via("resource")

private String pageTitle;

**Annotations in OSGi Components**

1. **@Component**
   * Marks a class as an OSGi component.
   * Used to register services in the OSGi container.

@Component(service = MyService.class)

public class MyServiceImpl implements MyService {

// ...

}

1. **@Service**
   * Deprecated in OSGi R6. Use @Component instead to specify the service.
2. **@Reference**
   * Injects an OSGi service into another service or component.

@Reference

private MyService myService;

1. **@Activate, @Deactivate**
   * Lifecycle methods for OSGi services.

@Activate

protected void activate() {

// Initialization logic

}

@Deactivate

protected void deactivate() {

// Cleanup logic

}

1. **@Property**
   * Deprecated. Use @Designate with OSGi annotations for configurations.
2. **@Designate**
   * Links a configuration class to an OSGi service.

@Designate(ocd = MyConfig.class)

@Component

public class MyService {

@Activate

protected void activate(MyConfig config) {

// Access config values

}

}