

# 25/03/2025 - TASKS

## Table of Contents:

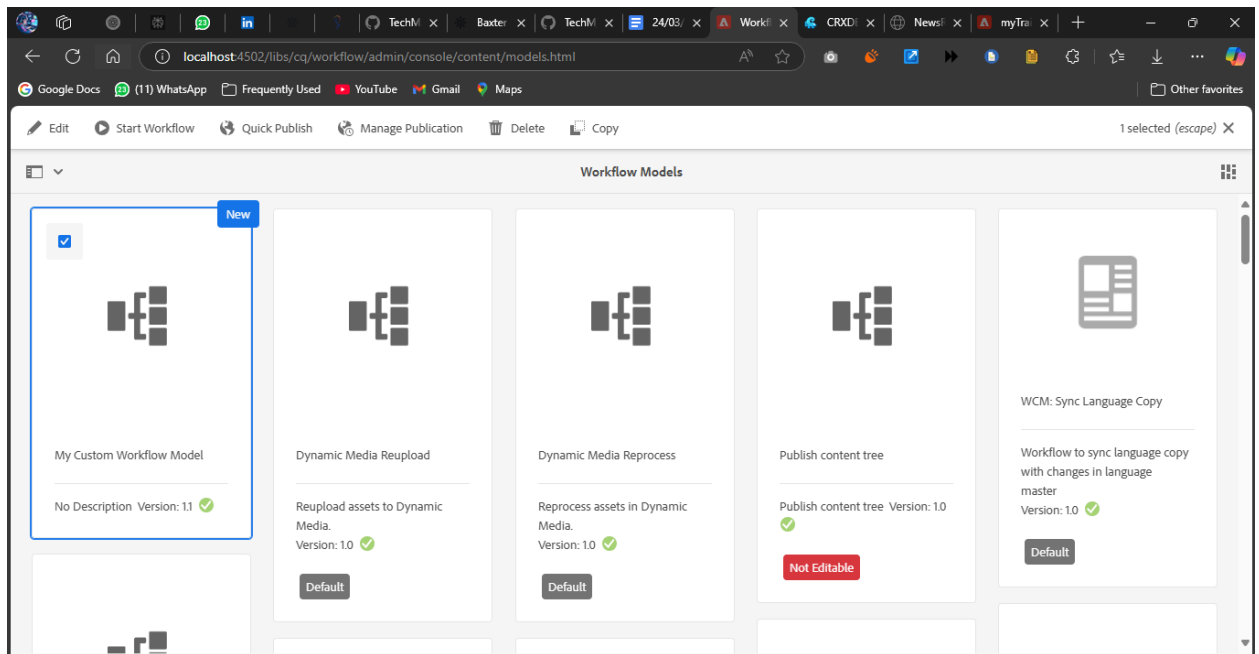
1. **Create Custom Workflow Model**
  2. **Create Custom Workflow Process**
  3. **Create Event Handler**
  4. **Create Sling Job**
  5. **Create Scheduler**
  6. **Create Users & Group with Permissions**
  7. **Test and Troubleshoot Custom Workflow, Event Handler, Sling Job, and Scheduler**
  8. **Best Practices for Workflow and Event Handling in AEM**
- 

## Create Custom Workflow Model

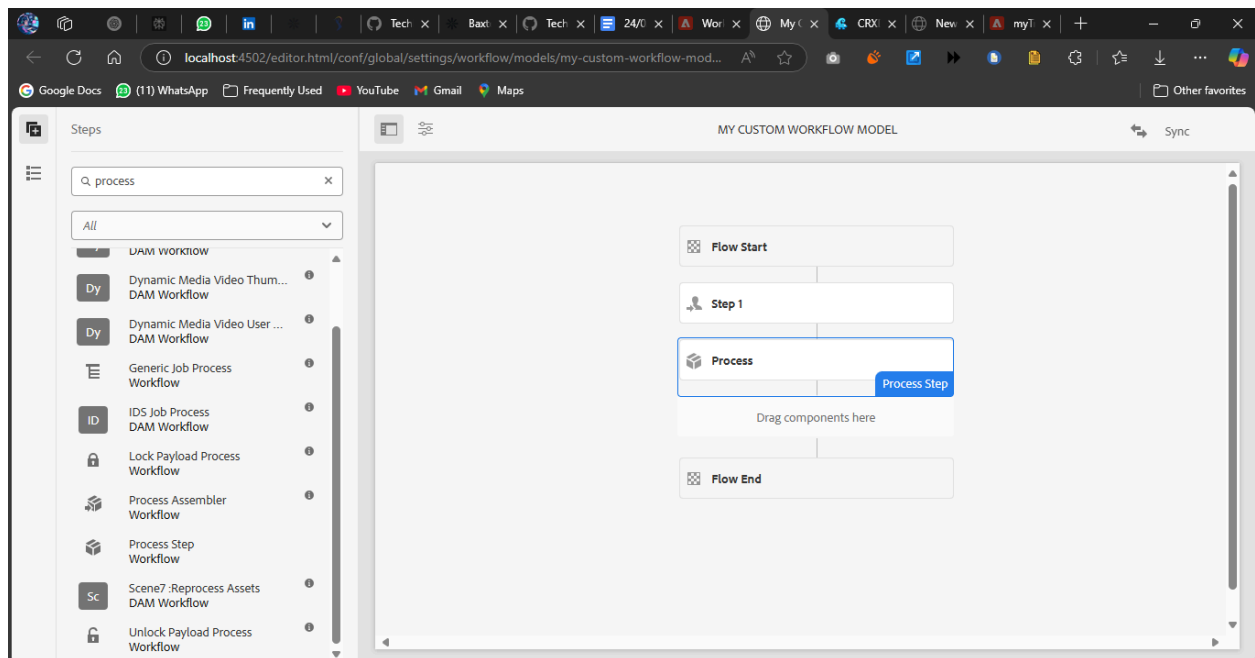
### Steps:

1. Go to **AEM Start** → **Tools** → **Workflow** → **Models**.
2. Click on **Create** → **Create Model**.
3. Set the **Title** as **My Custom Workflow Model**.

4. Open the model and click **Edit**.

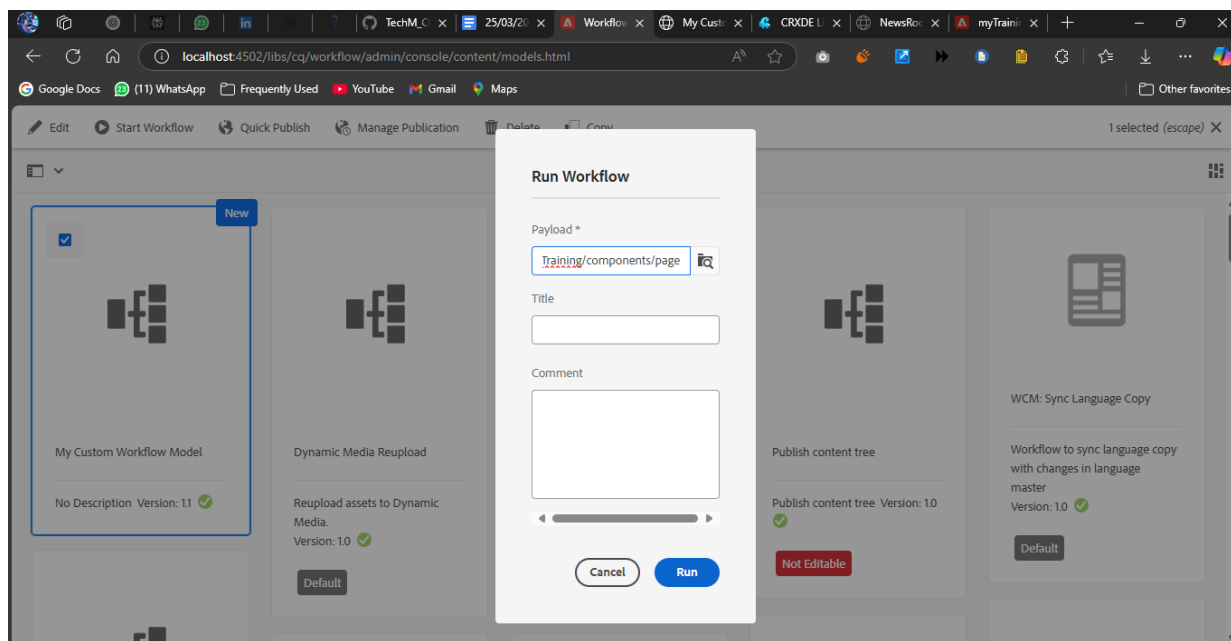
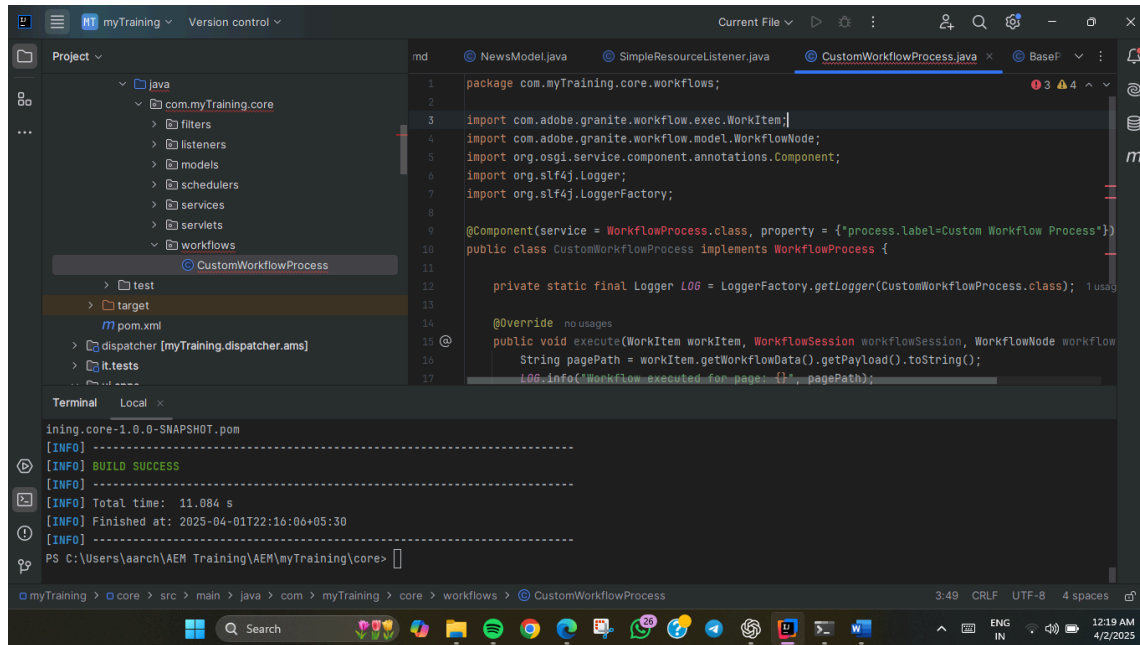


5. Drag and drop a **Process Step**.



6. In the Process Step configuration:

- Set **Title** to **Custom Workflow Process**.
- Set **Process** to the newly created **com.example.core.workflows.CustomWorkflowProcess**.



7. Save the model and close.

## Create Event Handler

### Steps:

1. Create a new Java class `CustomEventHandler.java` inside `com.example.core.listeners`.

```
package com.example.core.listeners;

import org.apache.sling.api.SlingConstants;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.event.Event;
import org.osgi.service.event.EventHandler;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component(service = EventHandler.class, immediate = true,
    property = {"event.topics=" + SlingConstants.TOPIC_RESOURCE_ADDED})
public class CustomEventHandler implements EventHandler {

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

    @Override
    public void handleEvent(Event event) {
        LOG.info("Resource added at: {}", event.getProperty("path"));
    }
}
```

2. Deploy and verify the logs when new resources are created in AEM.

---

## Create Sling Job

### Steps:

1. Create a new class `CustomSlingJob.java` inside `com.example.core.jobs`.

```
package com.example.core.jobs;
```

```

import org.apache.sling.event.jobs.Job;
import org.apache.sling.event.jobs.consumer.JobConsumer;
import org.osgi.service.component.annotations.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

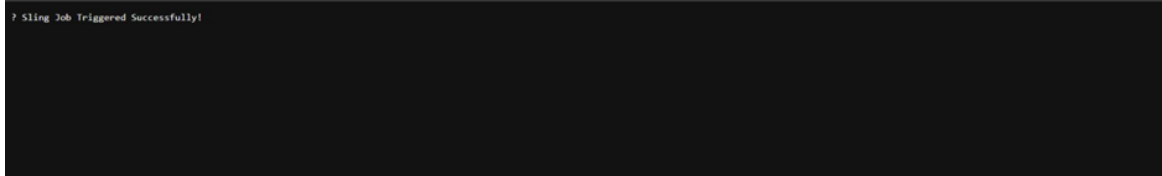
@Component(service = JobConsumer.class, property = {"job.topics=custom/job/helloworld"})
public class CustomSlingJob implements JobConsumer {

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

    @Override
    public JobResult process(Job job) {
        LOG.info("Hello World from Sling Job!");
        return JobResult.OK;
    }
}

```

2. Deploy the Sling Job and trigger it to test.



## Create Scheduler

### Steps:

1. Create the class `CustomScheduler.java` inside `com.example.core.schedulers`.

```

package com.example.core.schedulers;

import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Modified;
import org.osgi.service.component.annotations.ConfigurationPolicy;
import org.osgi.service.metatype.annotations.AttributeDefinition;
import org.osgi.service.metatype.annotations.ObjectClassDefinition;
import org.osgi.service.metatype.annotations.Designate;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.concurrent.Executors;

```

```
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
```

```
@Component(service = Runnable.class, configurationPolicy = ConfigurationPolicy.REQUIRE,
immediate = true)
```

```
@Designate(ocd = CustomScheduler.Config.class)
public class CustomScheduler implements Runnable {
```

```
    private static final Logger LOG = LoggerFactory.getLogger(CustomScheduler.class);
    private final ScheduledExecutorService scheduler =
Executors.newSingleThreadScheduledExecutor();
```

```
    @ObjectClassDefinition(name = "Custom Scheduler")
    public @interface Config {
        @AttributeDefinition(name = "Cron Expression")
        String cronExpression() default "0 0/5 * * * ?"; // Every 5 minutes
    }
```

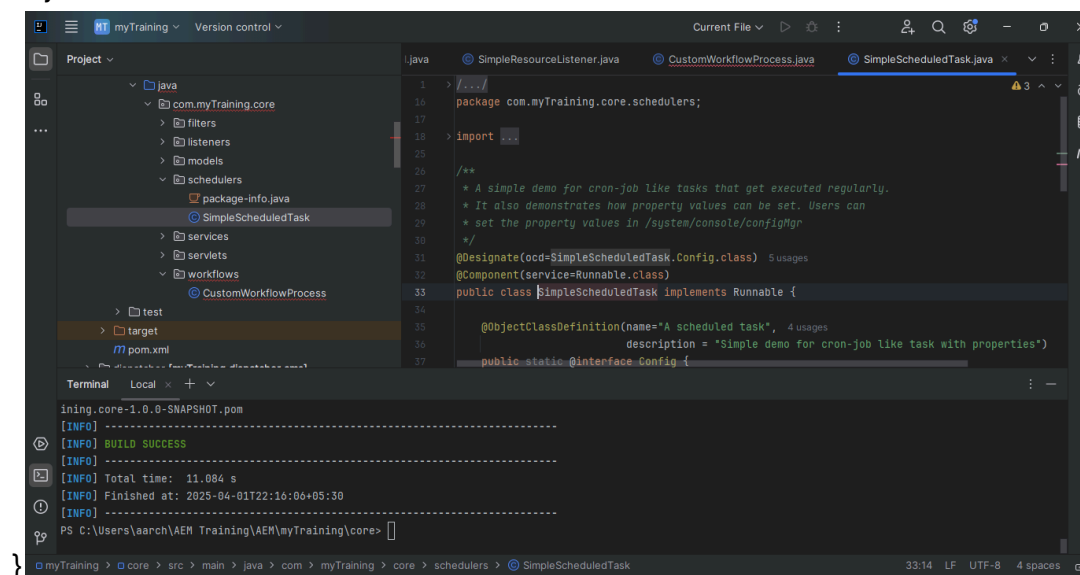
```
    @Activate
```

```
    @Modified
```

```
    protected void activate(Config config) {
        scheduler.scheduleAtFixedRate(this, 0, 5, TimeUnit.MINUTES);
    }
```

```
    @Override
```

```
    public void run() {
        LOG.info("Yellow World from Scheduler!");
    }
```



## Create Users & Group with Permissions

### Steps:

1. Navigate to **AEM Start** → **Security** → **Users**.
2. Create three users: **user1**, **user2**, **user3**.
3. Navigate to **Groups** → **Create Group**: **Dev Author**.
4. Add the created users to the **Dev Author** group.

The screenshot shows the 'Create New User' form in Adobe Experience Manager. The form has tabs for 'Details', 'Groups', and 'Impersonators'. The 'Details' tab is active, showing fields for ID, Password, Retype Password, Email, and Title. The ID field is filled with 'USER 1'. There is a 'New Photo' button below a placeholder image. At the top right, there are 'Cancel' and 'Save & Close' buttons.

The screenshot shows the 'Permissions' page in Adobe Experience Manager. The page title is 'content-authors'. On the left, there is a sidebar with a search bar and a list of groups. The 'Groups' dropdown is set to 'content-authors'. The main table lists permissions for various resources, including paths like /etc/cloudservices/scene7, /etc/importers/bulkeditor, and /etc/importers/offline. The permissions are categorized by 'allow', 'deny', and 'inherit'. The 'Add ACE' button is at the top right.

Resource	Permission	Action
/etc/cloudservices/scene7	allow	jcr:read
/etc/importers/bulkeditor	allow	jcr:read
/etc/importers/offline	allow	jcr:read
/etc/reports/auditreport	allow	jcr:read, rep:write
/etc/reports/compreport	allow	jcr:read, rep:write
/home/groups/cj_DXTID7BoO5t98PM7h	allow	jcr:read
/conf/myTraining/settings/wcm/policies	allow	crx:replicate
/conf/myTraining/settings/wcm/templates	allow	crx:replicate
/libs/_/aem-site-template-stub-2.0.0/sling.configs	allow	crx:replicate