TECH MAHINDRA COE TRAINING

AEM TASKS DOCUMENTATION(18/03/2025 - 28/03/2025)

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.

19/03/2025 - TASKS

Table of Contents

---

1. What is DAM and Why We Use It?

Digital Asset Management (DAM) in AEM

DAM (**Digital Asset Management**) is used to store, organize, retrieve, and manage digital assets like images, videos, and documents within Adobe Experience Manager (AEM).

Why Do We Use DAM?

- Centralized storage for digital assets
- Metadata management for organizing assets
- Image renditions for automatic optimization
- Version control for tracking changes
- Workflow integration for approvals before publishing

Location in AEM

All assets are stored inside:

/content/dam

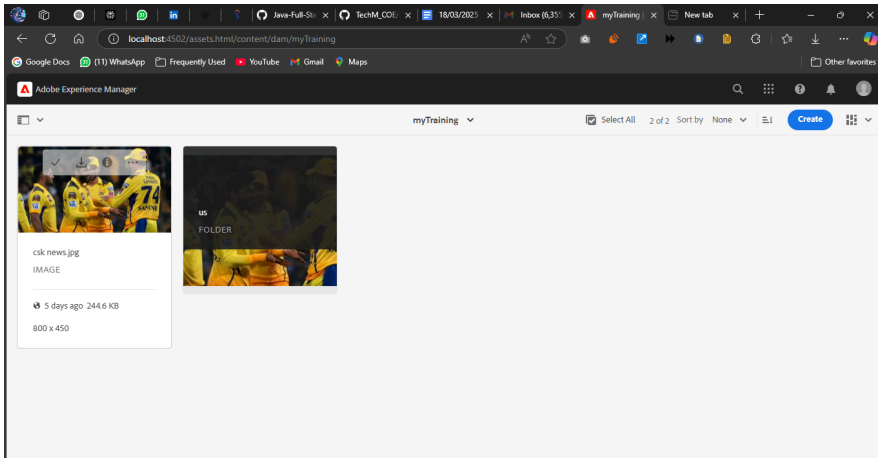
2. Create Folder and Upload Images

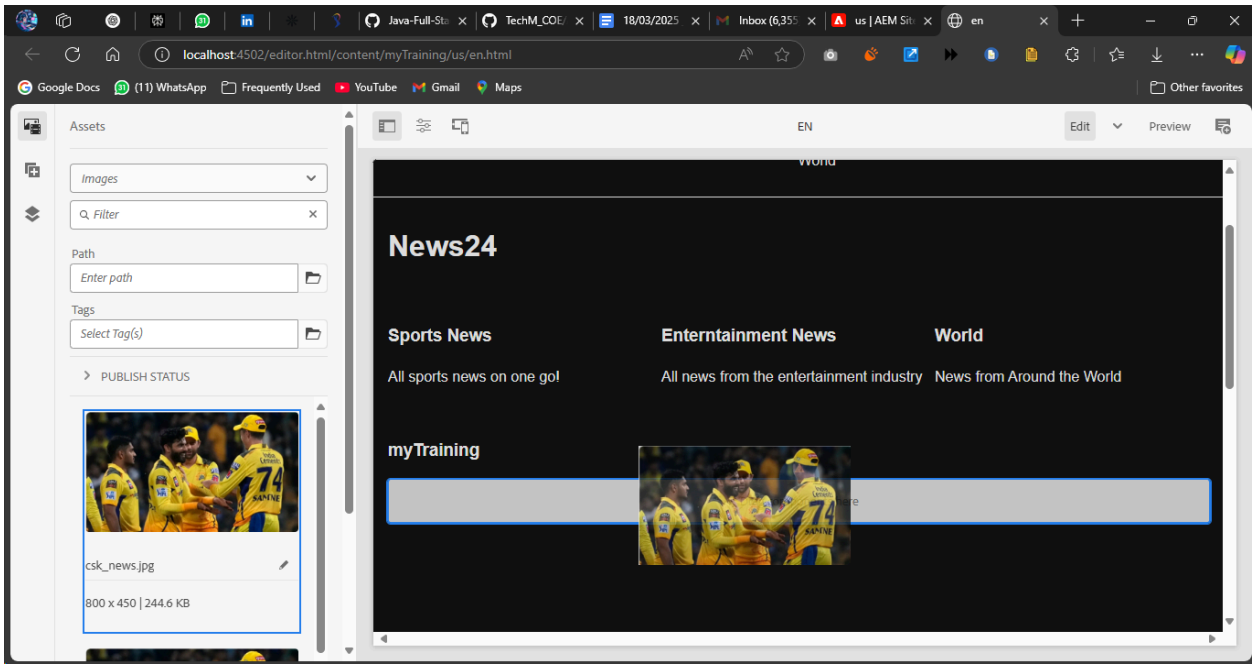
Steps to Create Folder & Upload Images

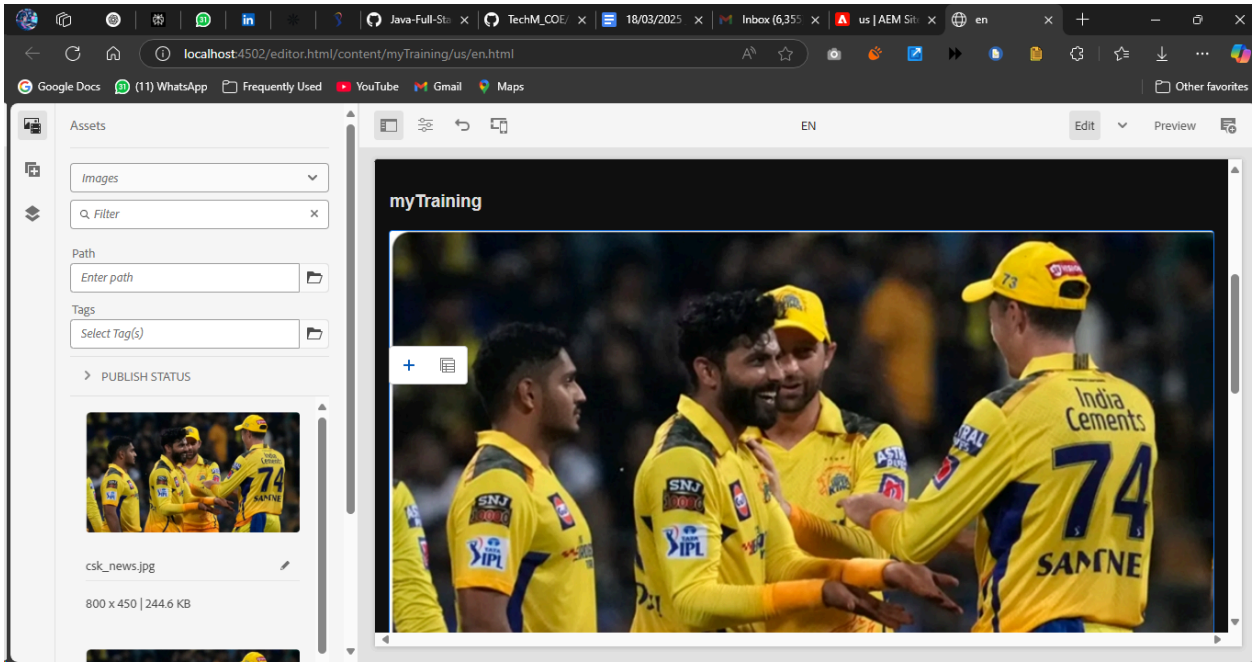
Open AEM **DAM Console**

http://localhost:4502/assets.html/content/dam

1. Create a Folder
    - Click **Create** → **Folder**
    - Enter **Folder Name**: `myTraining`
    - Inside `myTraining`, create another folder: `us/en-us`
2. Upload Images
    - Open `us/en-us` folder
    - Click **Upload** → Select 2 images from your system
    - Click **Save**
3. Add Image to a Page


localhost:4502/assets.html/content/dam/myTraining
Adobe Experience Manager
myTraining
Select All
Create
us
FOLDER
csk news.jpg
IMAGE
5 days ago 244.6 KB
800 x 450


localhost:4502/editor.html/content/myTraining/us/en.html
Assets
Images
Filter
Path
Enter path
Tags
Select Tag(s)
PUBLISH STATUS
csk_news.jpg
800 x 450 | 244.6 KB
EN
Edit
Preview
News24
Sports News
All sports news on one go!
Enterntainment News
All news from the entertainment industry
World
News from Around the World
myTraining



Open **Sites Console**:

http://localhost:4502/sites.html/content/myTraining

- Edit your page and drag **Image Component** from the Side Panel
- Select one of the uploaded images and **Save the Page**

3. Check Renditions in AEM

What are Renditions?

Renditions are different versions of an image automatically created when you upload an asset to DAM. These help optimize images for different screen sizes and devices.

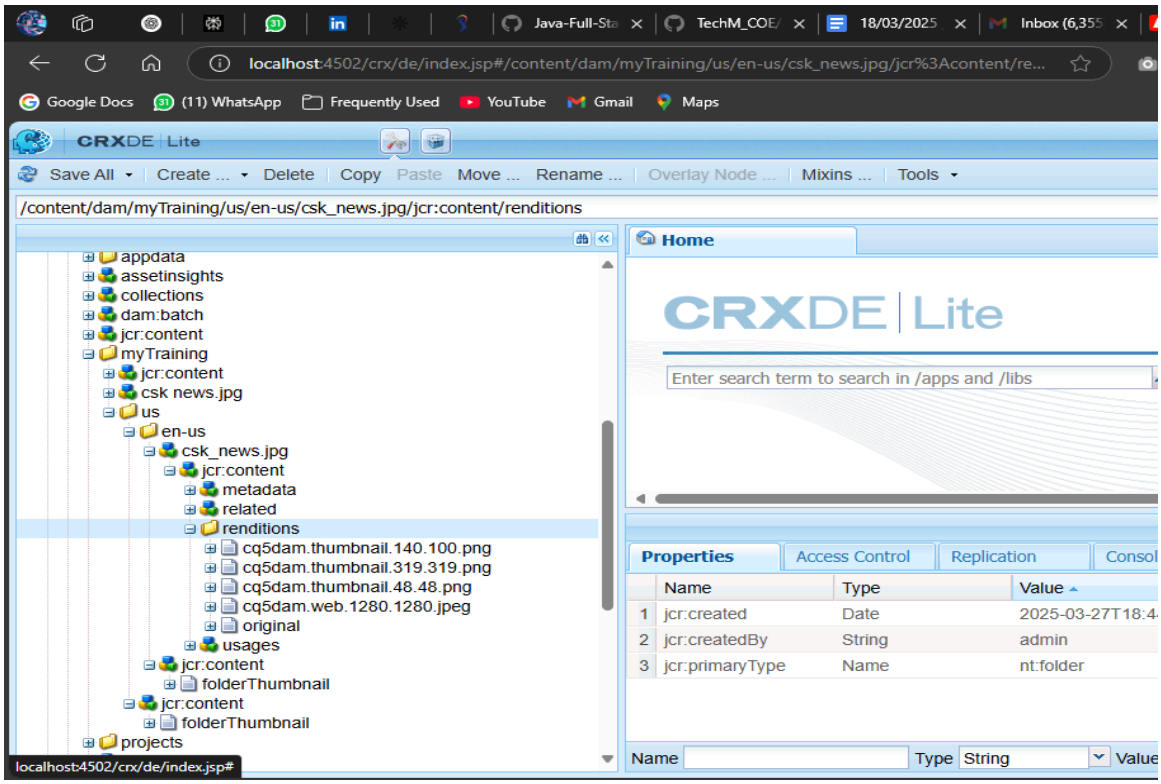
Steps to Check Renditions

Open **DAM Console**:

http://localhost:4502/assets.html/content/dam/myTraining/us/en-us

1. Click on an **uploaded image**
2. Click **View Properties → Renditions Tab**

3. You will see multiple renditions such as:
    - `cq5dam.thumbnail.48.48.png` (48x48 px thumbnail)
    - `cq5dam.web.1280.1280.jpeg` (Web-optimized version)



4. Modify HelloWorld Component

Steps to Add FirstName & LastName Fields

Open **CRXDE Lite**

http://localhost:4502/crx/de/

Navigate to:

/apps/myTraining/components/helloworld

1. Update `cq:dialog` to add input fields:

<firstName
jcr:primaryType="nt:unstructured"
sling:resourceType="granite/ui/components/coral/foundation/form/textfield"

```
  fieldLabel="First Name"
  name="./firstName"/>

<lastName
  jcr:primaryType="nt:unstructured"
  sling:resourceType="granite/ui/components/coral/foundation/form/textfield"
  fieldLabel="Last Name"
  name="./lastName"/>
```

4. Update `helloWorld.html` to display values:

```
<div>
  <h2>Hello World Component</h2>
  <p>First Name: ${properties.firstName}</p>
  <p>Last Name: ${properties.lastName}</p>
</div>
```

5. Save and Deploy the Component

5. Use @ValueMapValue in HelloWorldModel

Open Java Model File:

/core/src/main/java/com/myTraining/core/models/HelloWorldModel.java

1. Update `HelloWorldModel.java` to fetch values:

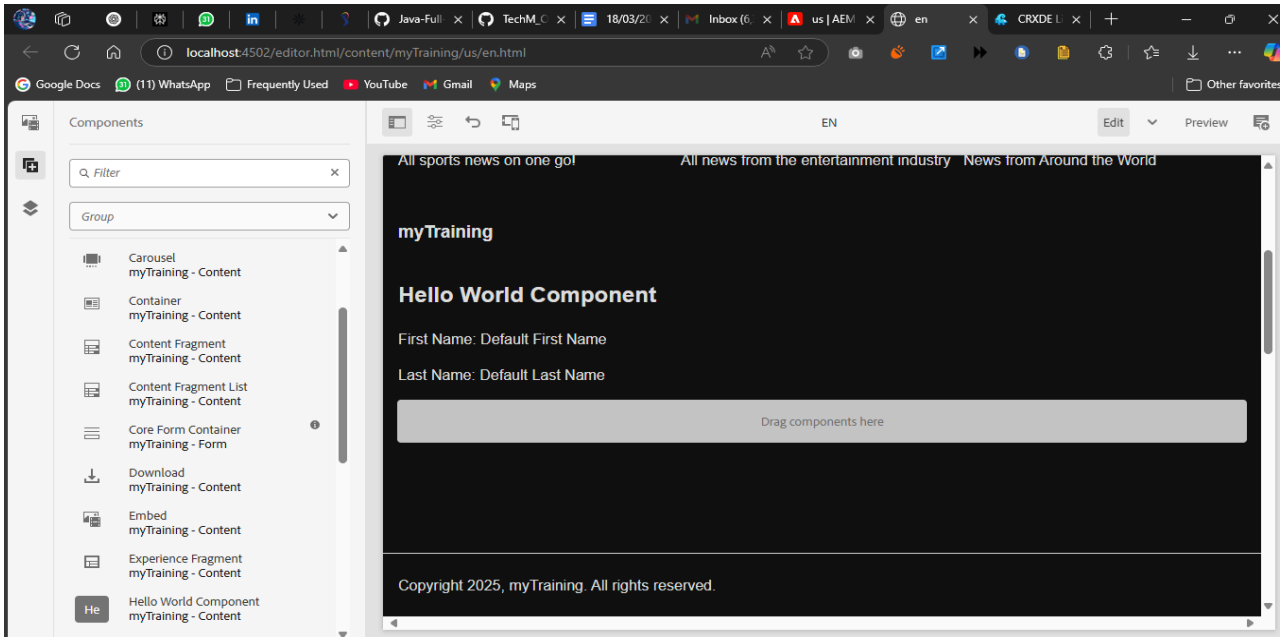
```
package com.myTraining.core.models;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.models.annotations.DefaultInjectionStrategy;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;

@Model(adaptables = Resource.class, defaultInjectionStrategy =
DefaultInjectionStrategy.OPTIONAL)
public class HelloWorldModel {

  @ValueMapValue
  private String firstName;

  @ValueMapValue
  private String lastName;
```

```
    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

3. Update `helloWorld.html` to use Sling Model:

```
<div data-sly-use.model="com.myTraining.core.models.HelloWorldModel">
  <h2>Hello World Component</h2>
  <p>First Name: ${model.firstName}</p>
  <p>Last Name: ${model.lastName}</p>
</div>
```

4. Save & Build the Project



6. Create AEM Packages Using Package Manager

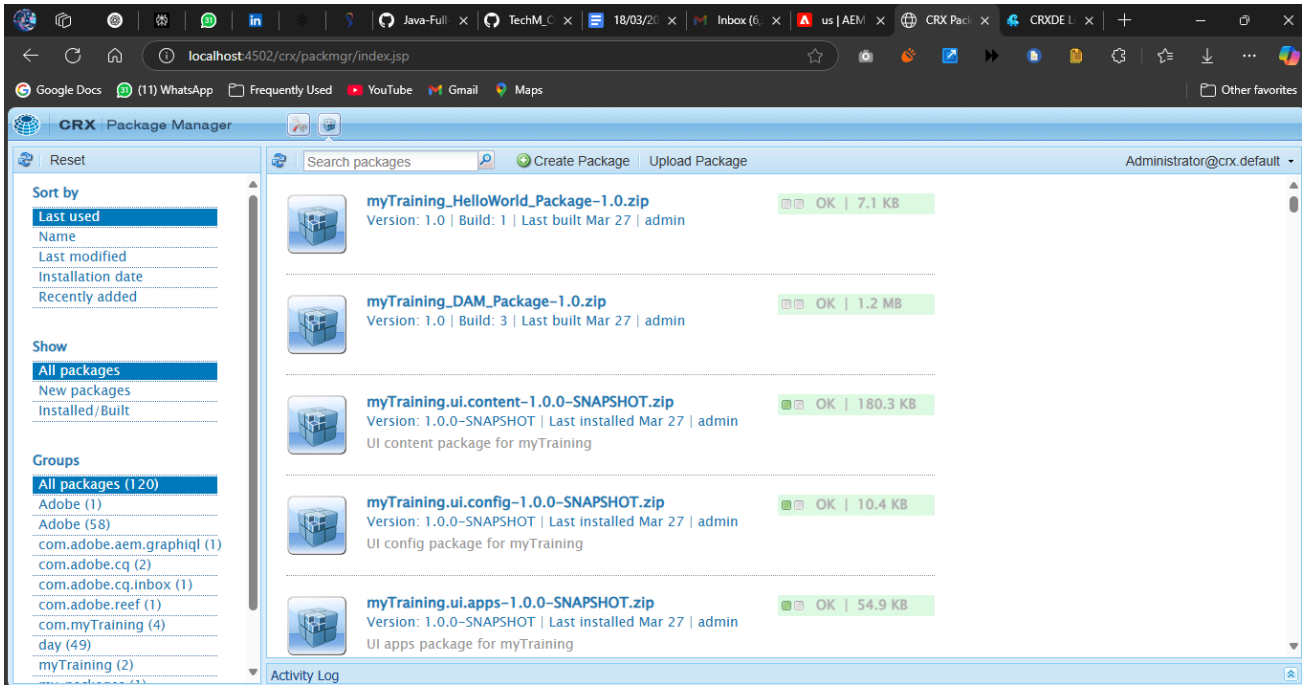
Open **Package Manager**

http://localhost:4502/crx/packmgr/index.jsp

1. Create DAM Package

- Name: `myTraining_DAM_Package`
- Group: `myTraining`
- Path: `/content/dam/myTraining/us/en-us`
- Click **Build & Download**

2. Create HelloWorld Package
   - Name: `myTraining_HelloWorld_Package`
   - Group: `myTraining`
   - Path: `/apps/myTraining/components/helloworld`
   - Click **Build & Download**



7. Configure Replication Agent & Publish Page

Open **Replication Agents Console**

http://localhost:4502/etc/replication/agents.author.html

1. Edit **Default Replication Agent**
    - **Name:** Default Agent
    - **Transport URI:** `http://localhost:4503/bin/receive`
    - **User:** `admin`
    - **Password:** `admin`
    - Click **Test Connection** → Should show **Successful**
2. Publish Page to 4503

Open **Sites Console**:

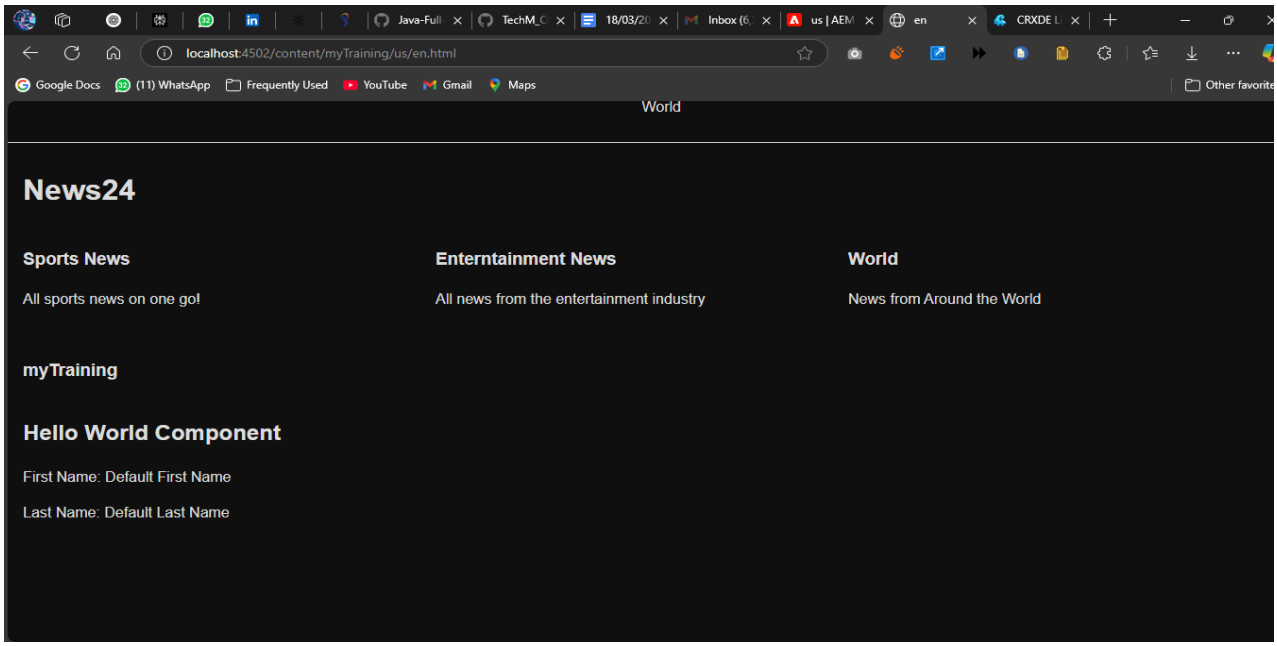
http://localhost:4502/sites.html/content

- Select your page → Click **Publish**

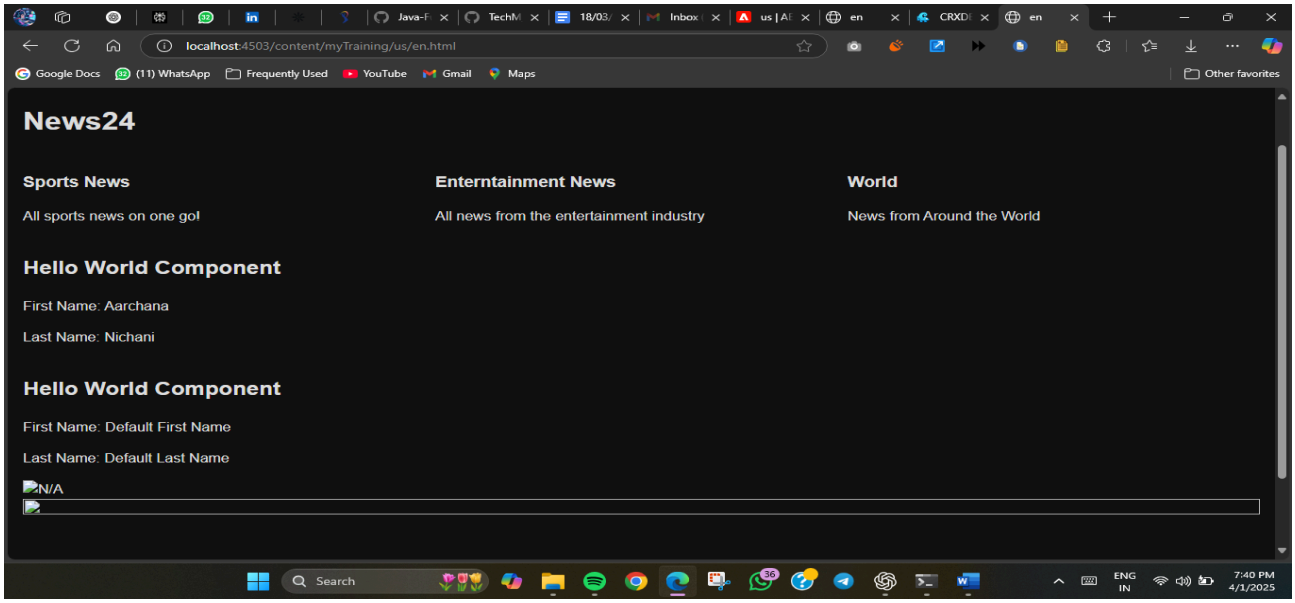
Verify the page in the **Publish Instance (4503)**:

http://localhost:4503/content/myTraining/us/en.html

AUTHOR INSTANCE:



PUBLISH INSTANCE:

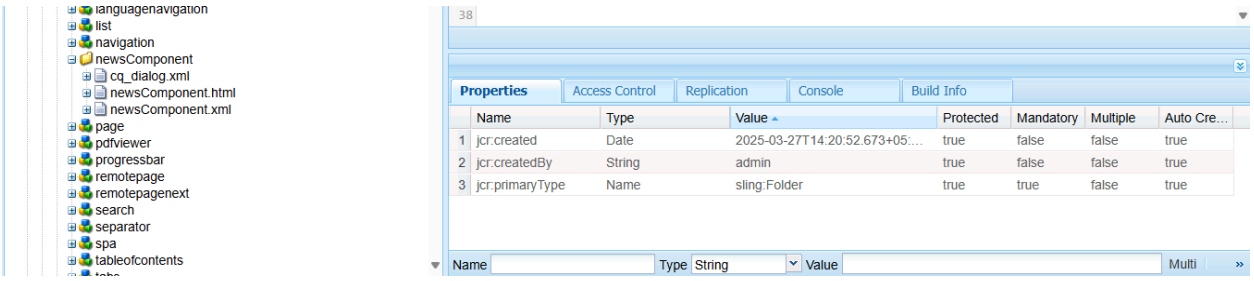


20/03/2025 - TASKS

1. Create a New Component for News

Steps:

1. Navigate to **CRXDE Lite** → `/apps/myTraining/components/`
2. Create a new component folder named `newsComponent`
3. Inside the `newsComponent` folder, create the following files:
    - `newsComponent.html` → **HTL template for rendering the component**
    - `newsComponent.xml` → **Defines the component properties**
    - `_cq_dialog.xml` → **Configures the dialog for authoring**
    - `NewsComponent.java` → **Sling Model (in core bundle)**



NewsComponent.java (Sling Model)

```
package com.myTraining.core.models;
import com.adobe.cq.sightly.WCMUsePojo;
import com.adobe.cq.wcm.core.components.models.Component;
import org.apache.sling.models.annotations.Default;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;
import javax.inject.Inject;

@Model(adaptables = { Component.class })
public class NewsComponent {

    @Inject
    @Default(values = "Default Title")
    private String title;

    @Inject
    @Default(values = "Default News Detail")
    private String newsDetail;

    @Inject
    @Default(values = "2025-03-20")
    private String publishedDate;

    @Inject
    @Default(values = "Default Source")
    private String source;

    public String getTitle() {
        return title;
    }
```

```
    public String getNewsDetail() {
        return newsDetail;
    }

    public String getPublishedDate() {
        return publishedDate;
    }

    public String getSource() {
        return source;
    }
}
```

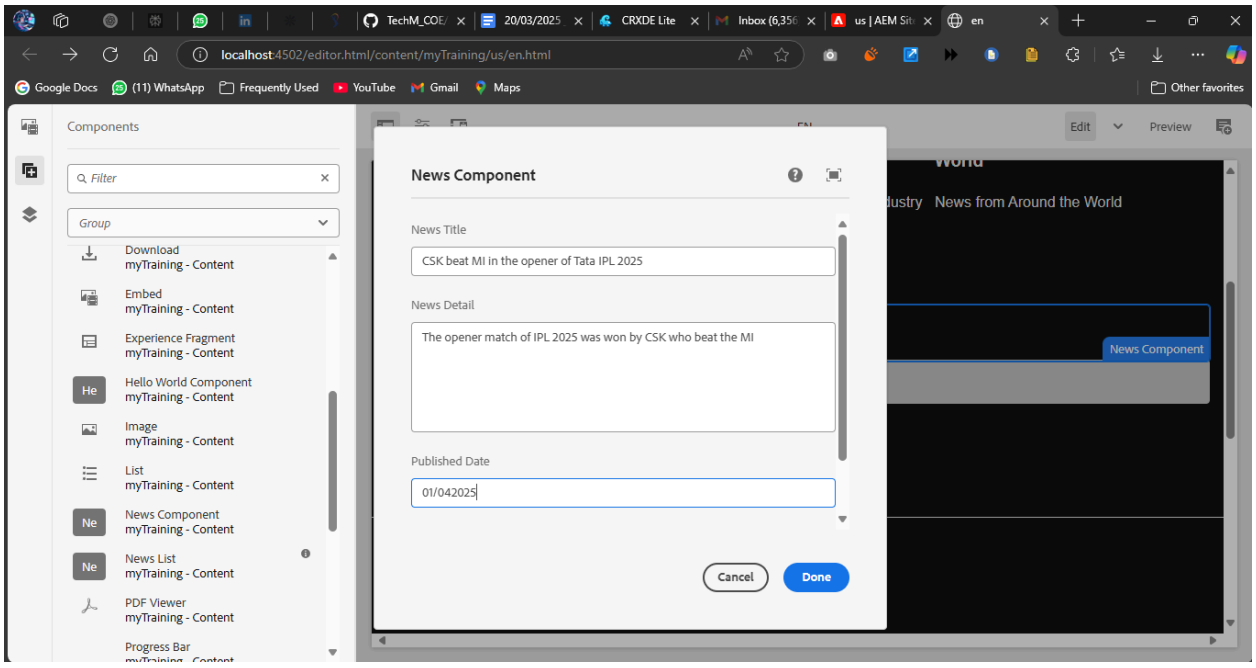
newsComponent.html (HTL Template)

```
<div class="cop-news-component">
    <h2 class="news-title">${properties.title}</h2>
    <p class="news-detail">${properties.newsDetail}</p>
    <p class="news-date">Published on: ${properties.publishedDate}</p>
    <p class="news-source">Source: ${properties.source}</p>
</div>
```

cq_dialog.xml (Dialog for Authoring)

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root
    xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
    xmlns:cq="http://www.day.com/jcr/cq/1.0"
    xmlns:jcr="http://www.jcp.org/jcr/1.0"
    jcr:primaryType="cq:Dialog"
    cq:dialogMode="floating"
    title="News Component"
    helpPath="help/using/newsComponent.html">
    <items jcr:primaryType="cq:Panel">
        <items jcr:primaryType="cq:WidgetCollection">
            <title
                jcr:primaryType="cq:Widget"
                fieldLabel="News Title"
                name="./title"
                xtype="textfield"/>
```

```
        <newsDetail
            jcr:primaryType="cq:Widget"
            fieldLabel="News Detail"
            name="./newsDetail"
            xtype="textarea"/>
        <publishedDate
            jcr:primaryType="cq:Widget"
            fieldLabel="Published Date"
            name="./publishedDate"
            xtype="datetime"/>
        <source
            jcr:primaryType="cq:Widget"
            fieldLabel="Source"
            name="./source"
            xtype="textfield"/>
      </items>
   </items>
</jcr:root>
```



2. Create a Multifield Component (Multiple News)

Steps:

1. Create a new component folder **`multiNewsComponent`**

2. Add a `multiNewsComponent.html` file.

3. Modify the **Sling Model** to support a multifield.

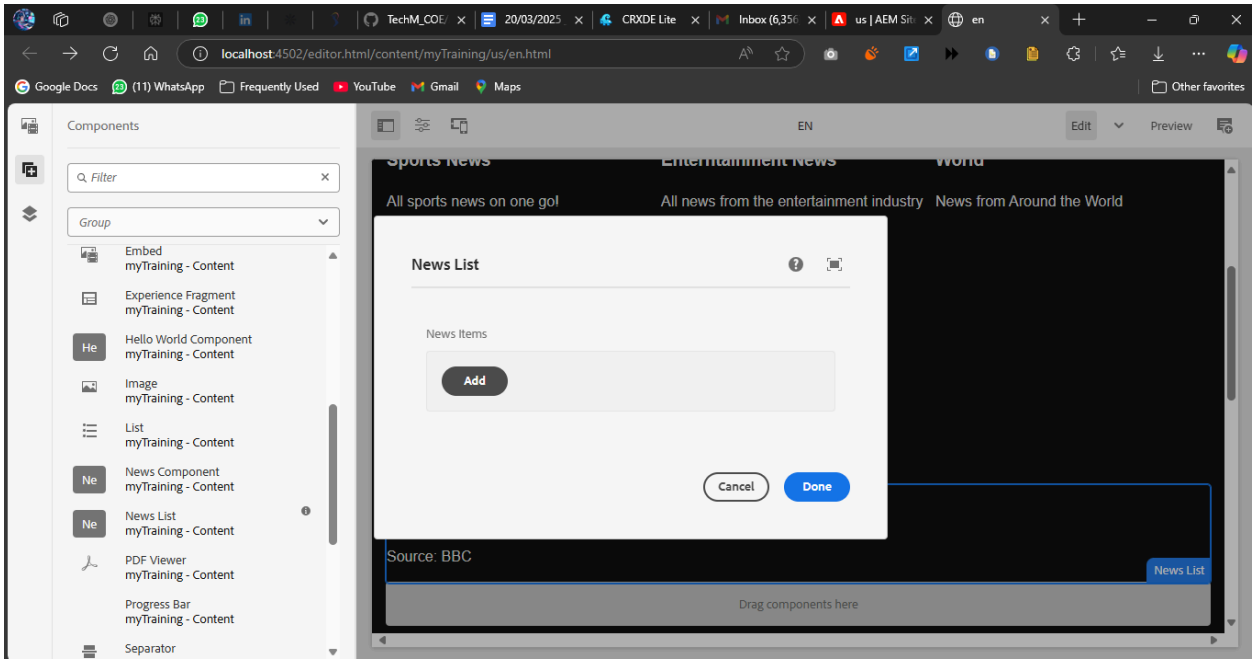
MultiNewsComponent.java

```
package com.myTraining.core.models;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.injectorspecific.ChildResource;
import javax.inject.Inject;
import java.util.List;
@Model(adaptables = Resource.class)
public class MultiNewsComponent {
    @ChildResource
    private List<NewsItem> newsList;

    public List<NewsItem> getNewsList() {
        return newsList;
    }
    public static class NewsItem {
        @Inject
        private String title;

        @Inject
        private String source;

        public String getTitle() {
            return title;
        }
        public String getSource() {
            return source;
        }
    }
}
```

multiNewsComponent.html

```
<div class="multi-news-component">
```

```
    <ul>
        <sly data-sly-list.news="${resource.children}">
            <li>
                <h2>${news.title}</h2>
                <p>Source: ${news.source}</p>
            </li>
        </sly>
    </ul>
</div>
```



3. Create Clientlibs and Apply Green colour to the heading(h2) and yellow colour to news detail(<p) and date should be black in colour.

Create a clientlib folder:

/apps/myTraining/clientlibs/newsComponent

clientlibs/css/style.css

```
.cop-news-component .news-title {
    color: green;
}

.cop-news-component .news-detail {
    color: yellow;
```

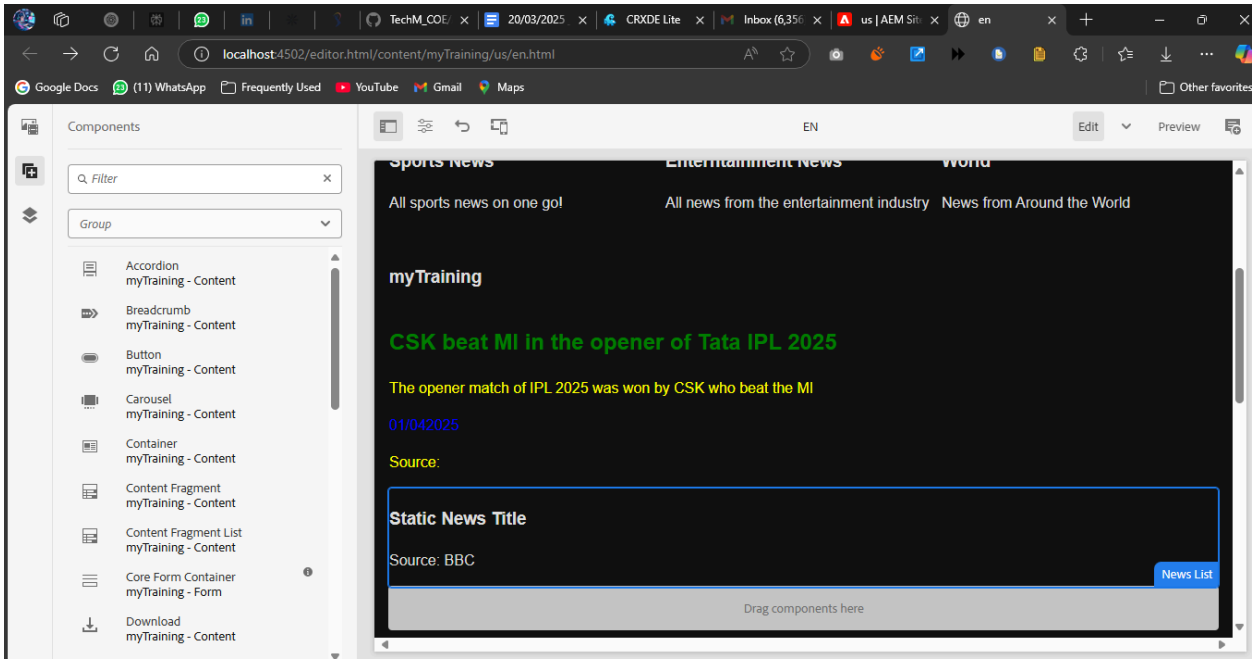
```
}

.cop-news-component .news-date {
    color: black;}
```

clientlibs/js/script.js

```
console.log("News Component Loaded");
```

clientlibs/.content.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root
    xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
    xmlns:jcr="http://www.jcp.org/jcr/1.0"
    jcr:primaryType="cq:ClientLibraryFolder"
    categories="myTraining.newsComponent"
    dependencies="cq.jquery">
</jcr:root>
```



4. Create a Base Page Component

Create a **basePage** component.

basePage.html

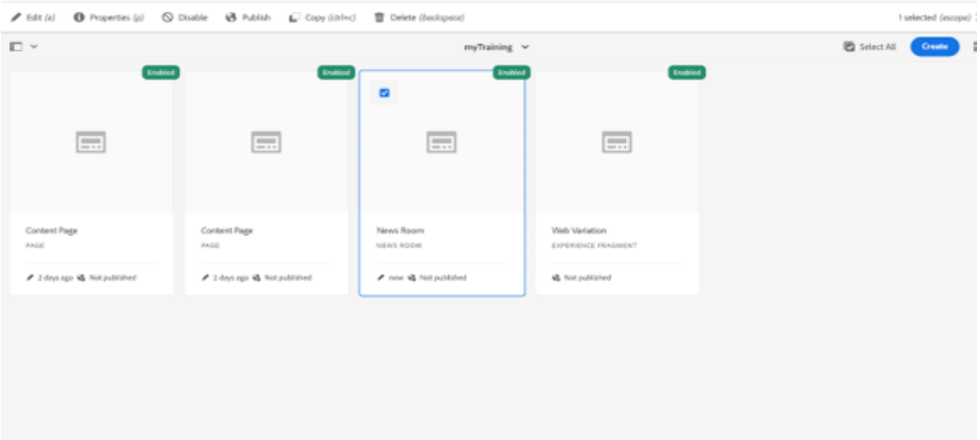
```
<head>
   <meta property="og:title" content="${properties.ogTitle}">
   <meta property="og:description" content="${properties.ogDescription}">
   <meta property="og:image" content="${properties.ogImage}">
</head>
```

5. Create Global Page Properties

Modify the page properties to include OG metadata.

_cq_dialog.xml

```
<items jcr:primaryType="cq:WidgetCollection">
   <ogTitle jcr:primaryType="cq:Widget" fieldLabel="OG Title" name="./ogTitle"
xtype="textfield"/>
   <ogDescription jcr:primaryType="cq:Widget" fieldLabel="OG Description"
name="./ogDescription" xtype="textarea"/>
   <ogImage jcr:primaryType="cq:Widget" fieldLabel="OG Image Path"
name="./ogImage" xtype="pathfield"/>
</items>
```



6. What is `extraClientLibs`?

- `extraClientLibs` is an **AEM feature** that allows including additional client libraries at runtime.
- It is useful when you need to **load specific client libraries dynamically**.
- You can add it in **multiNewsComponent**.

Adding extraClientLibs to Multifield Component

Modify `_cq_dialog.xml`:

```
<extraClientLibs
   jcr:primaryType="nt:unstructured"
   extraClientLibs="[myTraining.newsComponent]"
   xtype="checkbox"
   fieldLabel="Include Extra Clientlibs"/>
```

21/03/2025 - TASKS

- **Create a News room page component using the base page component**
- **Create a custom page property as NEWS configurations where I can author the default news image. Read More CTA for News Article**
- **Create a News room template type by using the news room page component**
- **Create a News Room Template using the news room template type.**
- **Apply the styling to the news/ hello world component from ui. frontend folder**
- **Additionally, you can create a custom style to apply the above styling to Hello World or the news component.**

---

1. Create News Room Page Component Using Base Page Component

1.1. Folder Structure

- Navigate to `ui.apps/src/main/content/jcr_root/apps/myTraining/components/structure/`.

- Create a new folder named `newsroom-page`.

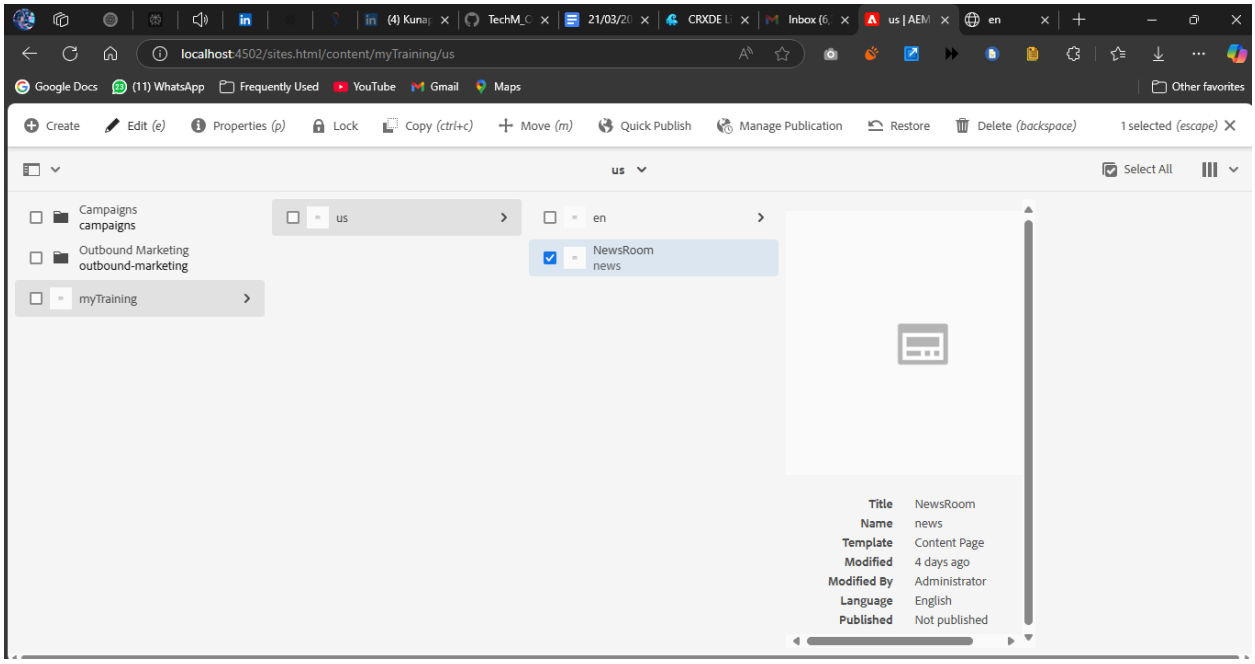
1.2. Create News Room Page HTML File

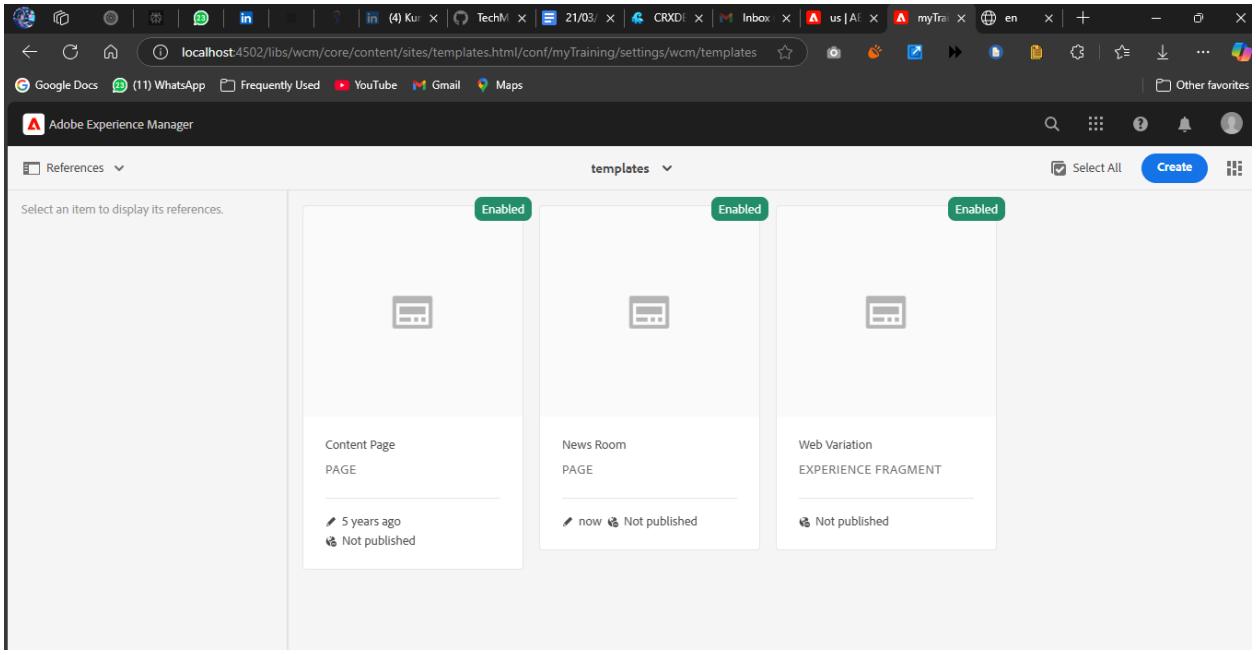
Inside the `newsroom-page` folder, create a new file named `newsroom-page.html` and extend the `base-page` component:

<sly data-sly-use.basePage="com.myTraining.core.models.BasePageModel">

```
  <div class="newsroom-page">
    <sly data-sly-include="@basePage.html"/>
  </div>
</sly>
```

- This code imports the `basePage` component and wraps it inside a div with the class `newsroom-page`. It ensures that the News Room page inherits the structure and styling from the `base-page` component.





2. Create Custom Page Property: NEWS Configurations

2.1. Create CQ Dialog for News Configuration

- Navigate to `ui.apps/src/main/content/jcr_root/apps/myTraining/components/structure/newsroom-page/`.

- Create a new dialog file named `cq:dialog` to allow authors to configure the page's news-specific properties such as default news image and the "Read More" CTA text.

```
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
        xmlns:cq="http://www.day.com/jcr/cq/1.0"
        xmlns:jcr="http://www.jcp.org/jcr/1.0"
        jcr:primaryType="cq:Dialog"
        cq:dialogMode="floating">
    <content jcr:primaryType="nt:unstructured"
            sling:resourceType="granite/ui/components/coral/foundation/container">
        <items jcr:primaryType="nt:unstructured">
            <newsConfig jcr:primaryType="nt:unstructured"
                    sling:resourceType="granite/ui/components/coral/foundation/form/fieldset"
                    jcr:title="NEWS Configurations">
                <items jcr:primaryType="nt:unstructured">
                    <defaultImage jcr:primaryType="nt:unstructured"

sling:resourceType="granite/ui/components/coral/foundation/form/fileupload"
```

```
                        fieldLabel="Default News Image"
                        name="./defaultNewsImage"/>
                <readMoreCTA jcr:primaryType="nt:unstructured"

sling:resourceType="granite/ui/components/coral/foundation/form/textfield"
                        fieldLabel="Read More CTA"
                        name="./readMoreCTA"/>
            </items>
        </newsConfig>
    </items>
  </content>
</jcr:root>
```

- **defaultImage** allows the author to upload a default image for the news article.
- **readMoreCTA** enables the author to specify the "Read More" call-to-action text.

3. Create News Room Template Type Using News Room Page Component

3.1. Folder Structure for Template Types

- Navigate to `ui.apps/src/main/content/jcr_root/conf/myTraining/settings/wcm/template-types/`.
- Create a folder named `newsroom-template-type`.

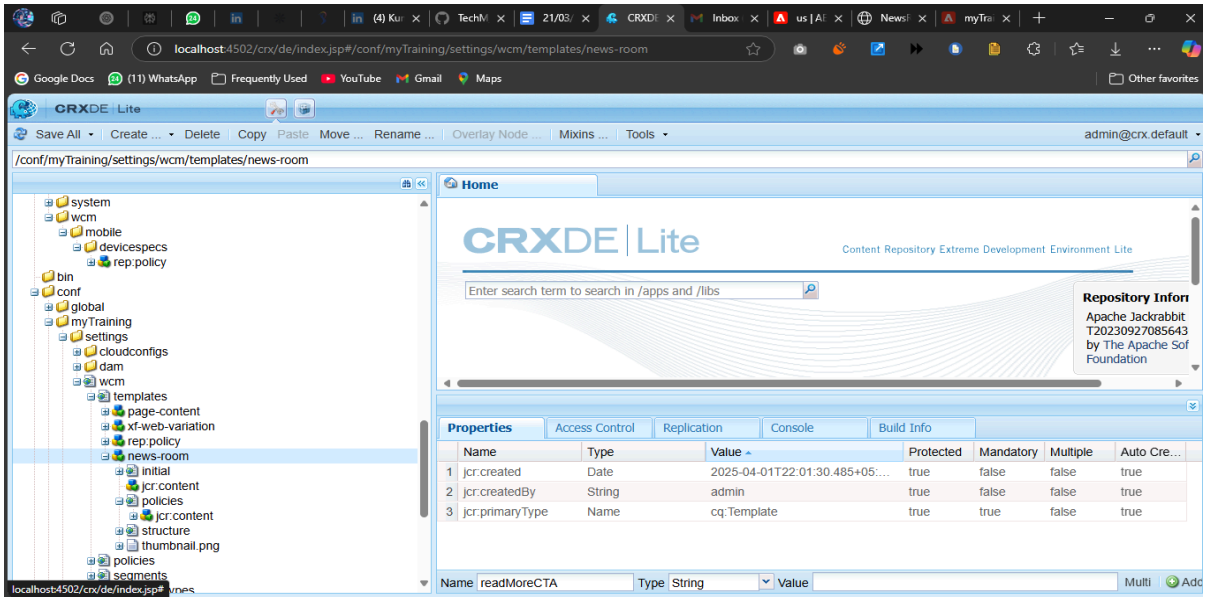
3.2. Create `.content.xml` for Template Type

Inside the `newsroom-template-type` folder, create a file named `.content.xml` with the following content:

<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0"

```
xmlns:cq="http://www.day.com/jcr/cq/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0"
jcr:primaryType="cq:TemplateType"
jcr:title="News Room Template Type"
allowedPaths="/content/myTraining(/.*)?"
allowedChildren="/apps/myTraining/components/structure/newsroom-page"/>
```

- This defines a new template type, `News Room Template Type`, and specifies that it can be used in paths under `/content/myTraining`. It also indicates that the template type allows children of the `newsroom-page` component.



4. Create News Room Template Using News Room Template Type

4.1. Folder Structure for Templates

- Navigate to `ui.apps/src/main/content/jcr_root/conf/myTraining/settings/wcm/templates/`.

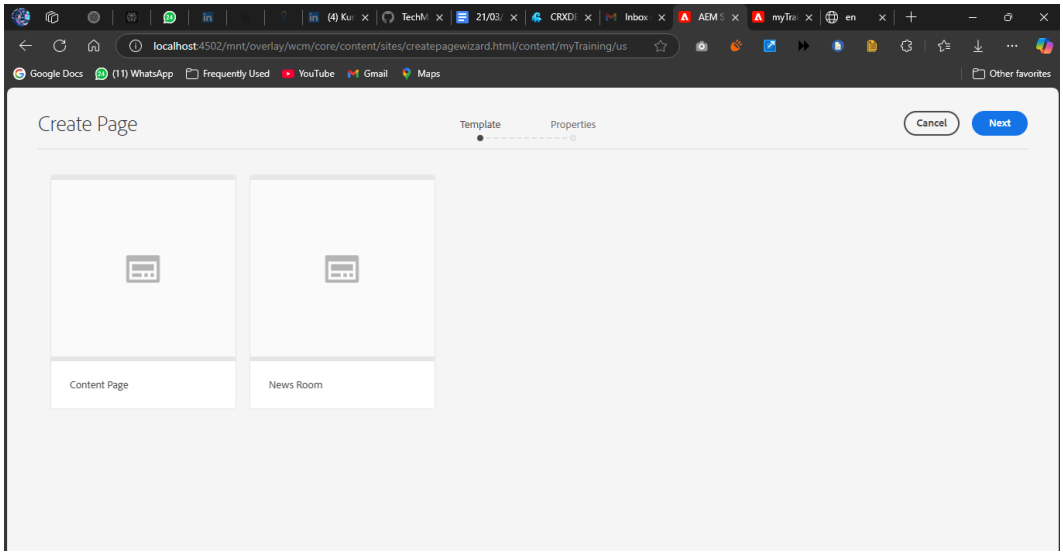
- Create a folder named `newsroom-template`.

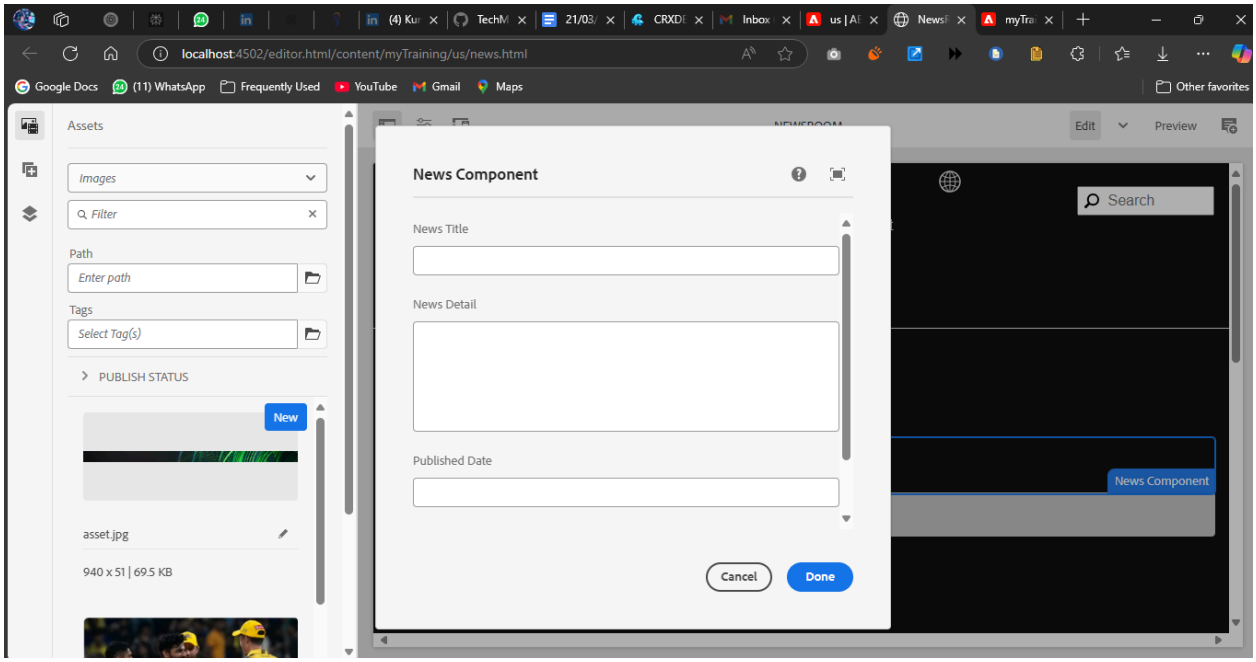
4.2. Create `.content.xml` for Template

Inside the `newsroom-template` folder, create a file named `.content.xml` with the following content:

```
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
        xmlns:cq="http://www.day.com/jcr/cq/1.0"
        xmlns:jcr="http://www.jcp.org/jcr/1.0"
        jcr:primaryType="cq:Template"
        jcr:title="News Room Template"
        jcr:description="Template for News Room Pages"
        allowedPaths="/content/myTraining(/.*)?"

cq:templateType="/conf/myTraining/settings/wcm/template-types/newsroom-template-type"/>
```

- This creates a `News Room Template` using the `newsroom-template-type` defined earlier, allowing authors to select this template for pages in the `/content/myTraining` path.





5. Apply Styling to News/Hello World Component from `ui.frontend`

5.1. Locate the News Component Styles

Navigate to `ui.frontend/src/main/webpack/components/news/` and locate the file `news.scss`. Add the following styles:

```
.news-component {
    h2 {
        color: green;
    }
    p {
        color: yellow;
    }
    .date {
        color: black;
    }
}
```

- These styles apply green to the title (h2), yellow to the news detail (p), and black to the date.

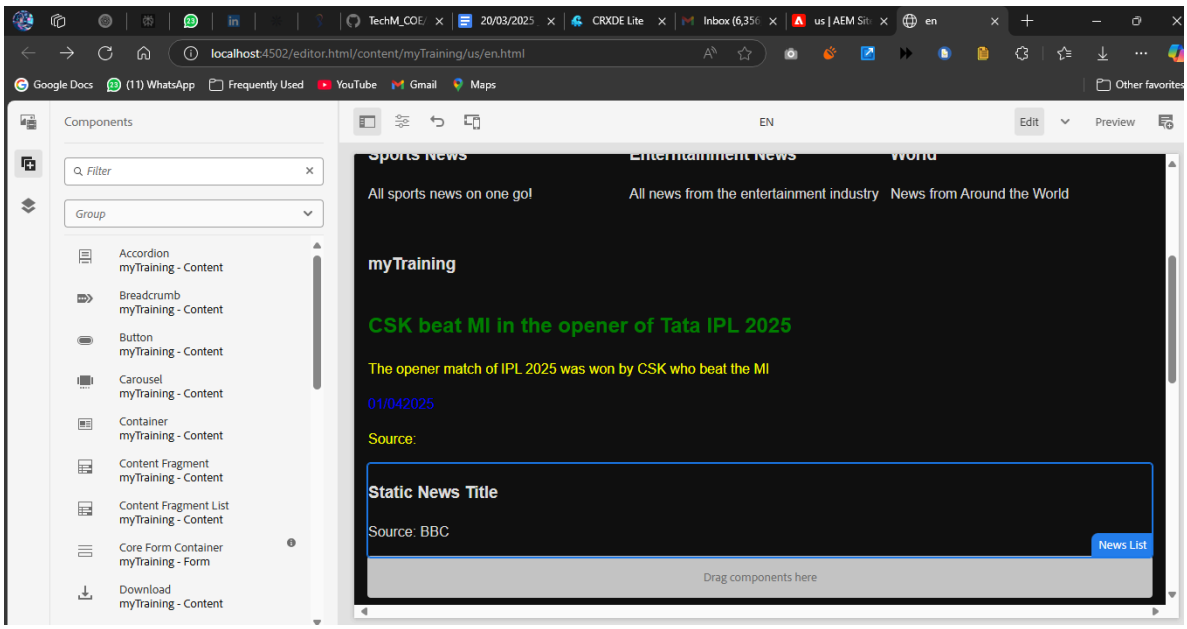
6. Create Custom Style to Apply Above Styling to Hello World or News Component

6.1. Define Custom Styles for News Component

Navigate to `ui.apps/src/main/content/jcr_root/apps/myTraining/components/news/` and create a new `cq:style` configuration inside `.content.xml`:

```
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
        xmlns:cq="http://www.day.com/jcr/cq/1.0"
        xmlns:jcr="http://www.jcp.org/jcr/1.0"
        jcr:primaryType="nt:unstructured">
   <cq:styleClasses jcr:primaryType="nt:unstructured">
      <style1 jcr:primaryType="nt:unstructured"
            cq:styleLabel="Green Header"
            cq:styleId="green-header"/>
      <style2 jcr:primaryType="nt:unstructured"
            cq:styleLabel="Yellow News Detail"
            cq:styleId="yellow-news-detail"/>
   </cq:styleClasses>
</jcr:root>
```

- This configuration allows users to apply specific styles like `Green Header` and `Yellow News Detail` through AEM's style system.



24/03/2025 - TASKS

Table of Contents

---

1. Create 5 News Article Pages

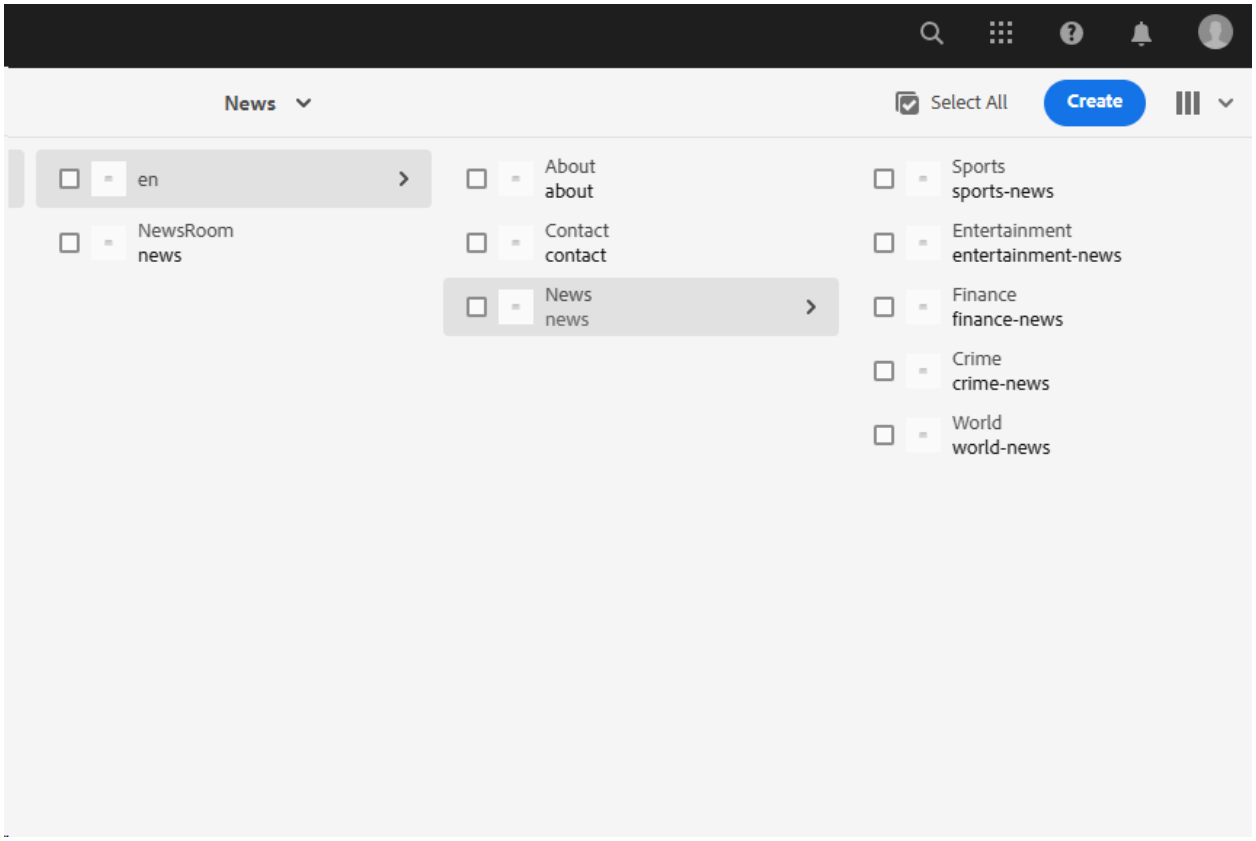
Objective: Create 5 unique news article pages within AEM to display content.

Steps:

1. **Navigate to Content Folder**:
    - Go to **/content/us/en/news** in AEM's CRX/DE or AEM Author instance.
2. **Create News Article Pages**:
    - Right-click on the `/news` folder and create 5 new pages.
    - Each page should have a unique title, for example:
        - News Article 1
        - News Article 2
        - News Article 3
        - News Article 4

- News Article 5

- Ensure each page has distinct content for each article.



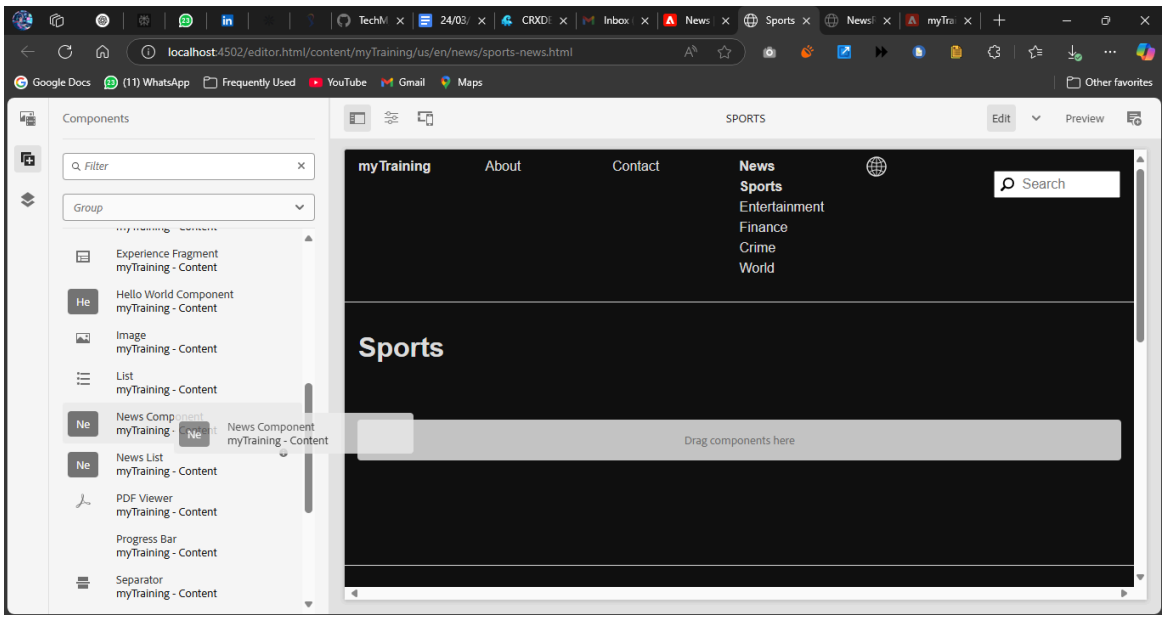
2. Use News Component

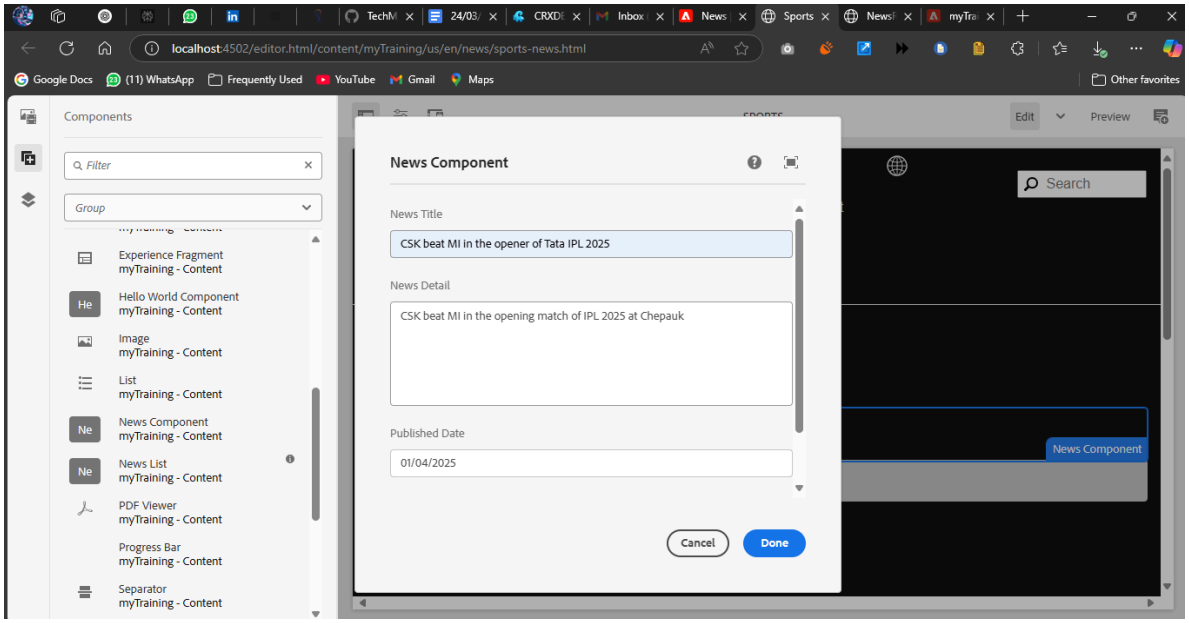
Objective: Add a previously created **News Component** to each news page to display content like title, detail, and published date.

Steps:

1. **Create or Locate the News Component**:
    - If not already created, ensure the **News Component** is available under `/apps/myTraining/components/structure/news`.
2. **Add Component to Each News Page**:
    - Edit each of the 5 news article pages.

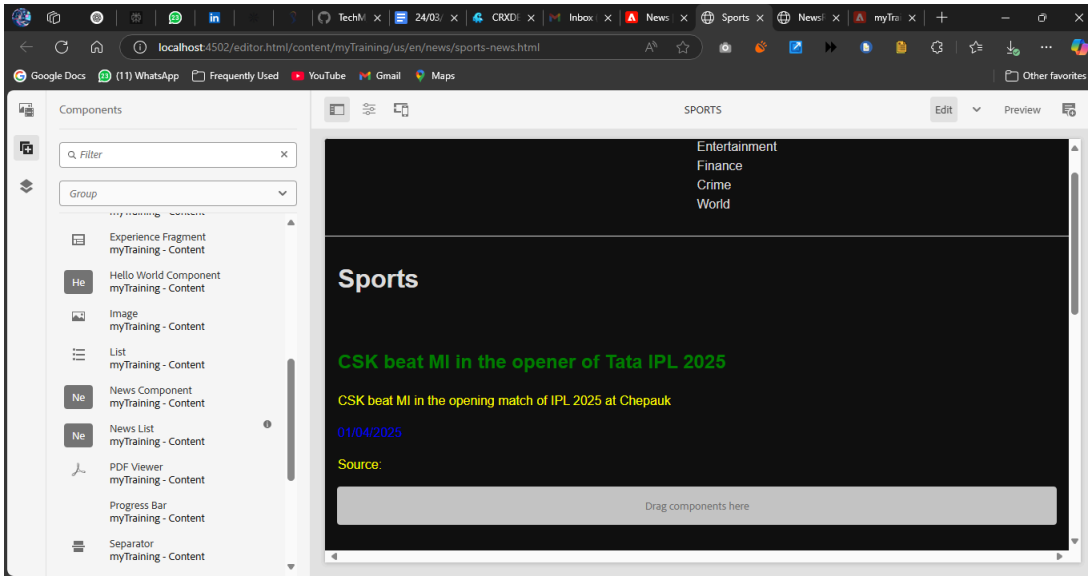
- Add the **News Component** to the page by dragging it from the sidekick (or adding it via the component tab in the page editor).





3. **Customize the News Component**:

```
<div class="news-item">

  <h2 style="color: green;">${title}</h2>
  <p style="color: yellow;">${newsDetail}</p>
  <span style="color: black;">Published on: ${publishedDate}</span></div>
```



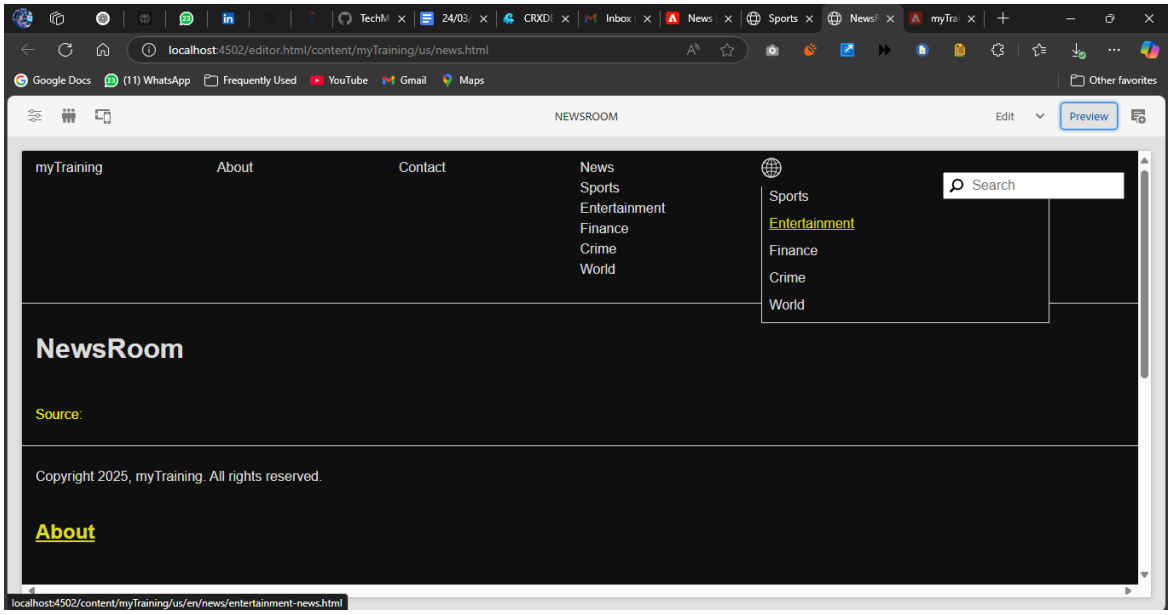
---

3. Create Header Experience Fragment

Objective: Design a **Header Experience Fragment** to contain the navigation menu and important links.

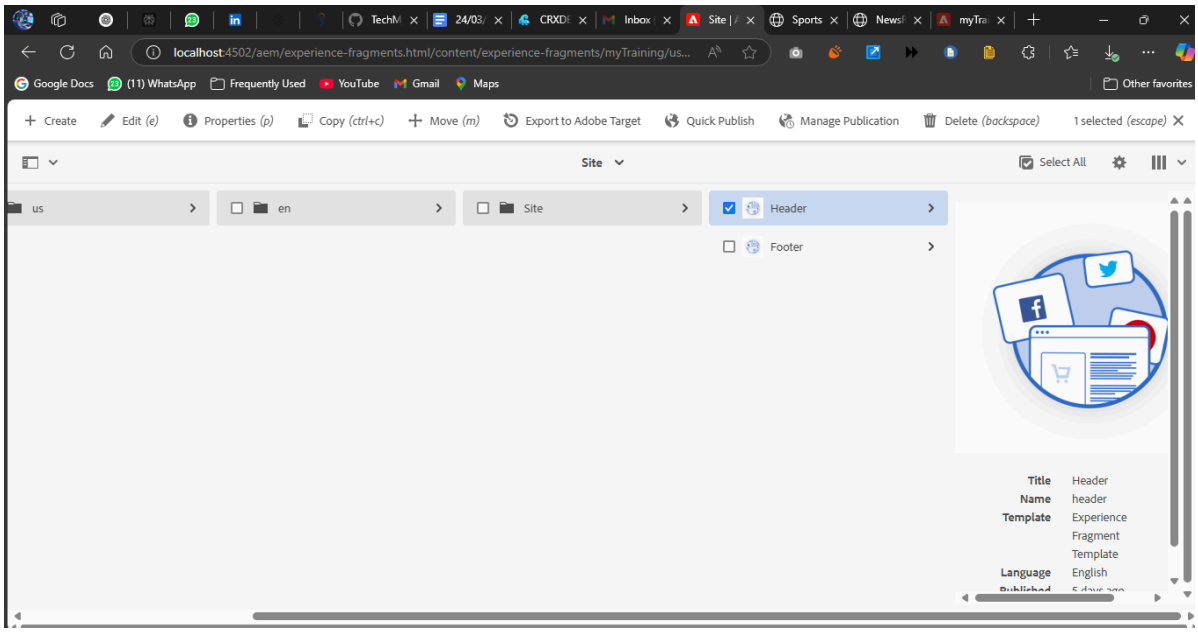
Steps:

1. **Navigate to Experience Fragments**:
    - Go to **/content/experience-fragments** in AEM.
    - Create a new experience fragment for the header (e.g., `header-fragment`).
2. **Design the Header**:
    - Add a **Navigation Menu** component.
    - Link the following pages:
        - News Menu (Links to news pages)
        - Contact Us Page
        - About Me Page



3. **Publish the Experience Fragment**:

   - Once designed, make sure the experience fragment is published and ready for use across pages.

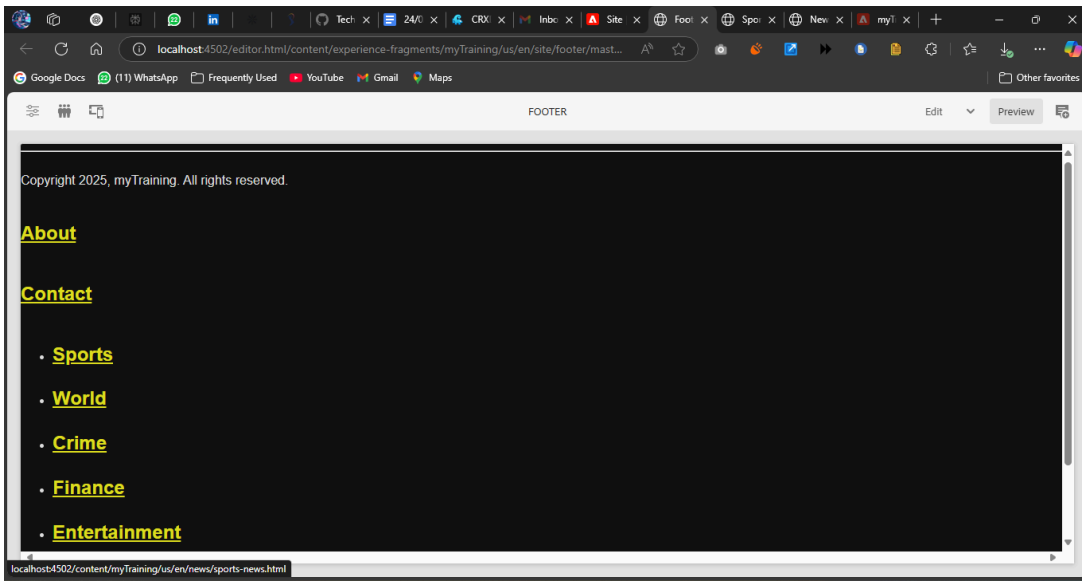


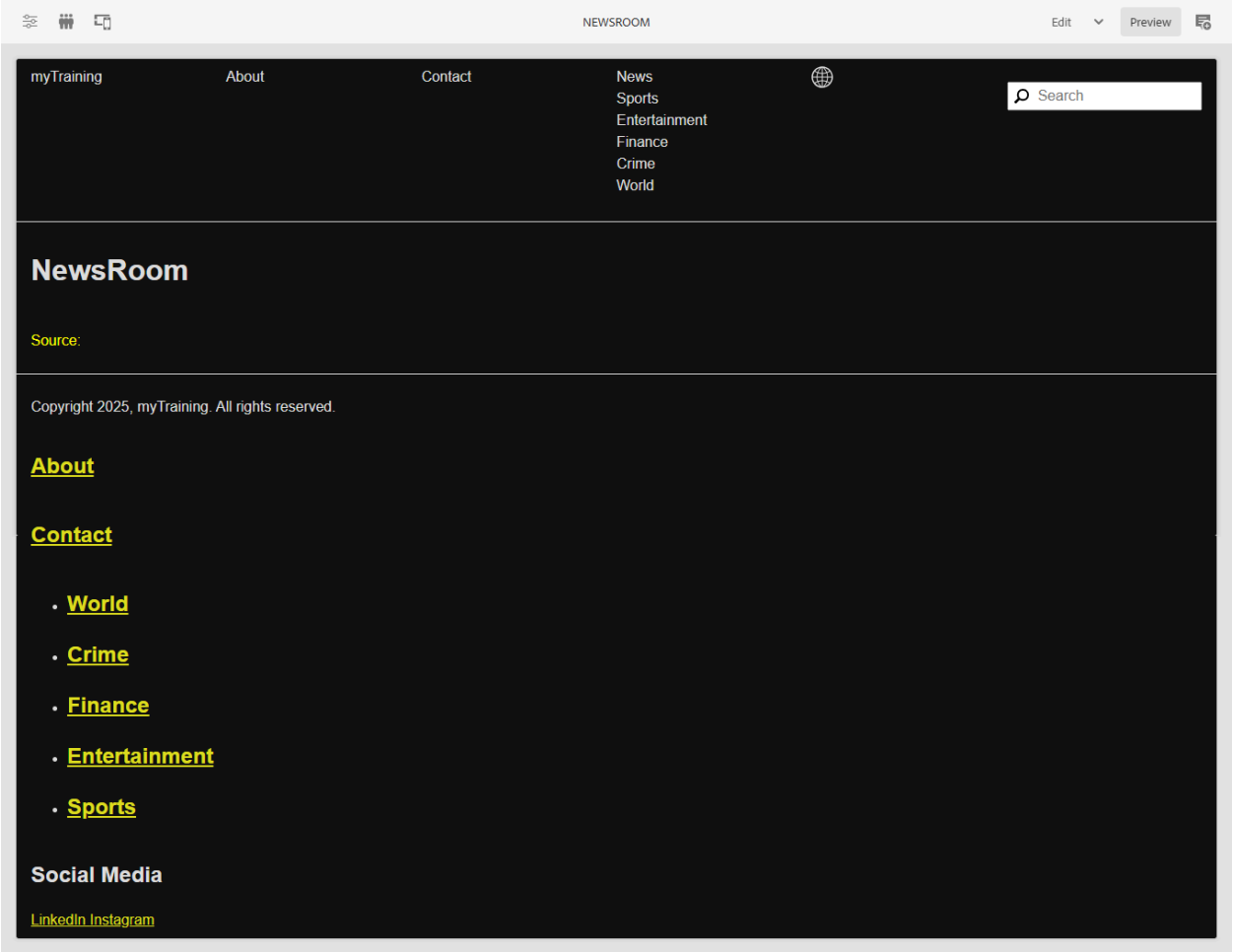
4. Create Footer Experience Fragment

Objective: Create a **Footer Experience Fragment** with multiple sections, including news articles, contact information, and social media links.

Steps:

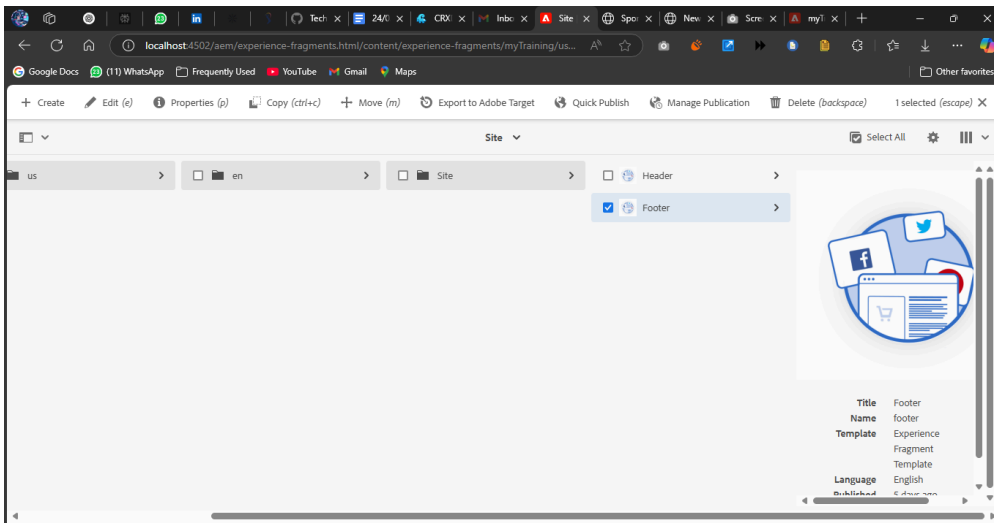
1. **Navigate to Experience Fragments**:
    - Go to **/content/experience-fragments** in AEM.
    - Create a new experience fragment for the footer (e.g., `footer-fragment`).
2. **Design the Footer**:
    - Add the following sections using appropriate components:
        - **News Menu Section**: Add a **List Component** to display the 4 most recent news articles.
        - **About Me Section**: Add a **Text Component** to provide brief information about the journalist.
        - **Contact Us Section**: Add a **Text Component** to list the contact details (email, phone, office address).
        - **Social Media Section**: Add a **List Component** for links to social media accounts.





3. **Publish the Experience Fragment**:

   - Once complete, publish the footer experience fragment.



5. Create Custom Service

Objective: Develop a **Custom Service** in AEM that prints `Hello World` and is called within the **News Component's Sling Model**.

Steps:

1. **Create a Service Interface**:

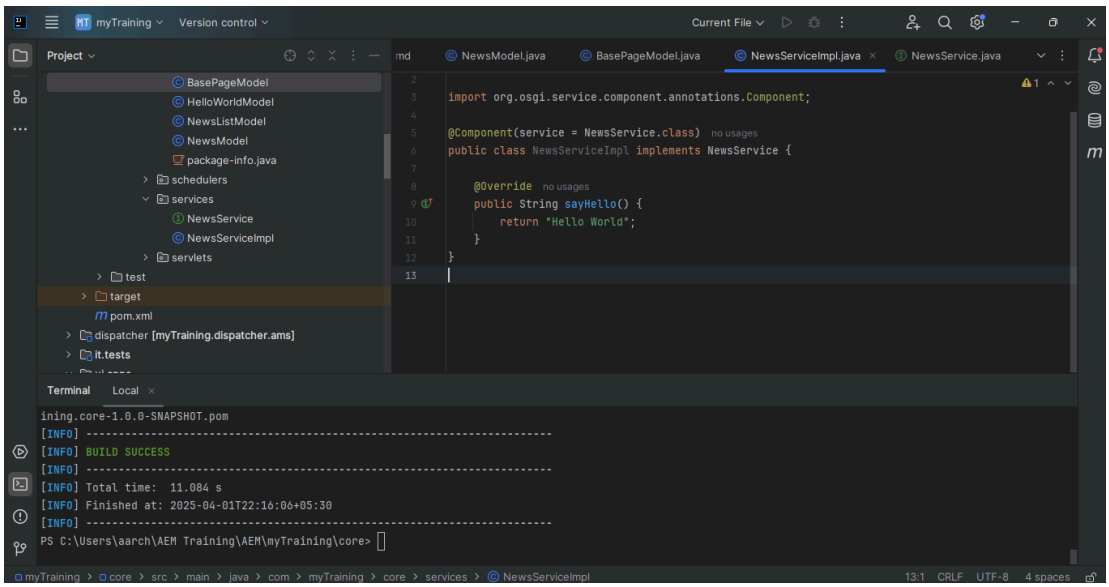
In your `core` module, create a service interface like `NewsService`:

```
 public interface NewsService {
    String sayHello();
}
```

2. **Create the Service Implementation**:

Implement the service interface in a new class `NewsServiceImpl`:

```
 @Component(service = NewsService.class)
public class NewsServiceImpl implements NewsService {
    @Override
    public String sayHello() {
        return "Hello World";
    }
}
```



3. **Inject and Call the Service in Sling Model**:

In the `BasePageModel` or a new `NewsComponentModel`, inject and use the service:

```
 @Inject
private NewsService newsService;

public String getGreeting() {
    return newsService.sayHello();
}
```

4. **Log the Output**:

Log the service output in the AEM logs to confirm it's working:

```
 @Activate
@Modified
public void logGreeting() {
    String greeting = getGreeting();
    LOGGER.info(greeting);  // Logs "Hello World"
}
```

6. Create Custom Configuration

Objective: Create a **Custom Configuration** to store a third-party API URL and fetch JSON data from it.

Steps:

1. **Create the Configuration Interface**:

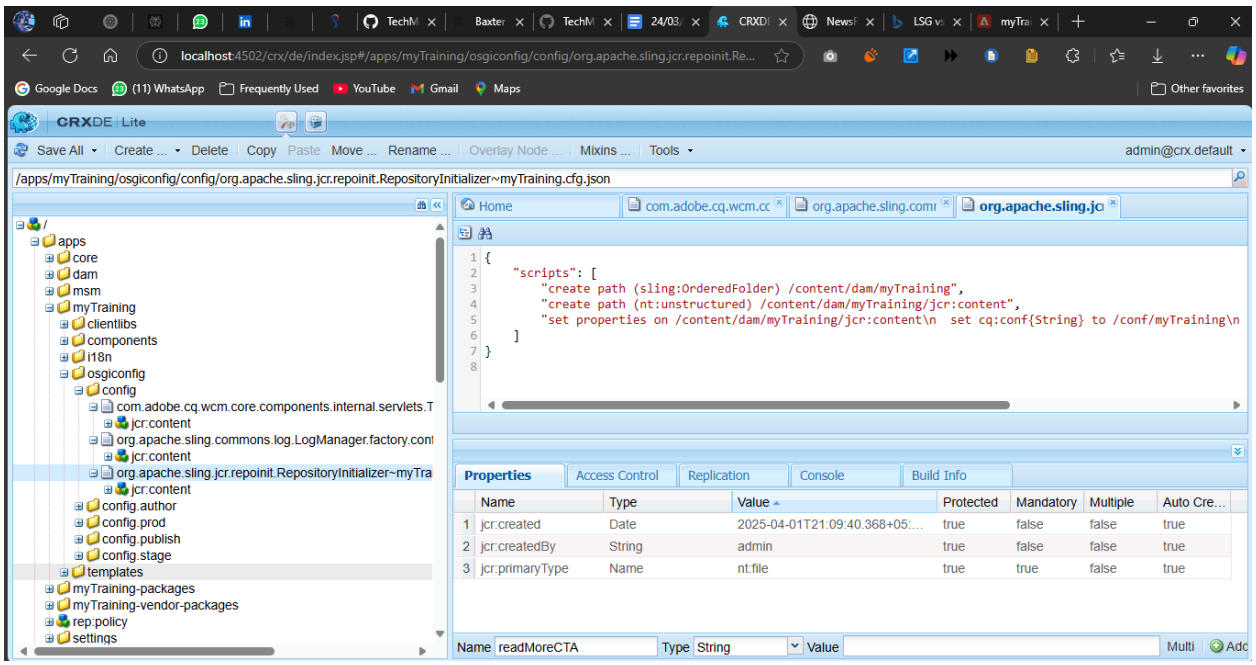
Define a Sling Model or OSGi configuration to store the API URL.

```
 @Designate(ocd = MyConfig.class)
public class MyConfig {
    @Activate
    @Modified
    public void activate() {
        String apiUrl = config.apiUrl();
        LOGGER.info("Configured API URL: " + apiUrl);
    }
```

```
    @Activate
    @Modified
    @Property
    private String apiUrl;
}
```

2. **Create the Configuration Dialog**:

    - Add a configuration dialog under `/apps/myTraining/configs` where you can input the third-party API URL, such as `https://jsonplaceholder.typicode.com/posts`.



3. **Fetch API Data**:

In the service or Sling model, fetch data from the configured API URL:

```
 HttpClient client = HttpClients.createDefault();
HttpGet request = new HttpGet(apiUrl);
HttpResponse response = client.execute(request);
String jsonResponse = EntityUtils.toString(response.getEntity());
LOGGER.info("Fetched API Response: " + jsonResponse);
```

25/03/2025 - TASKS

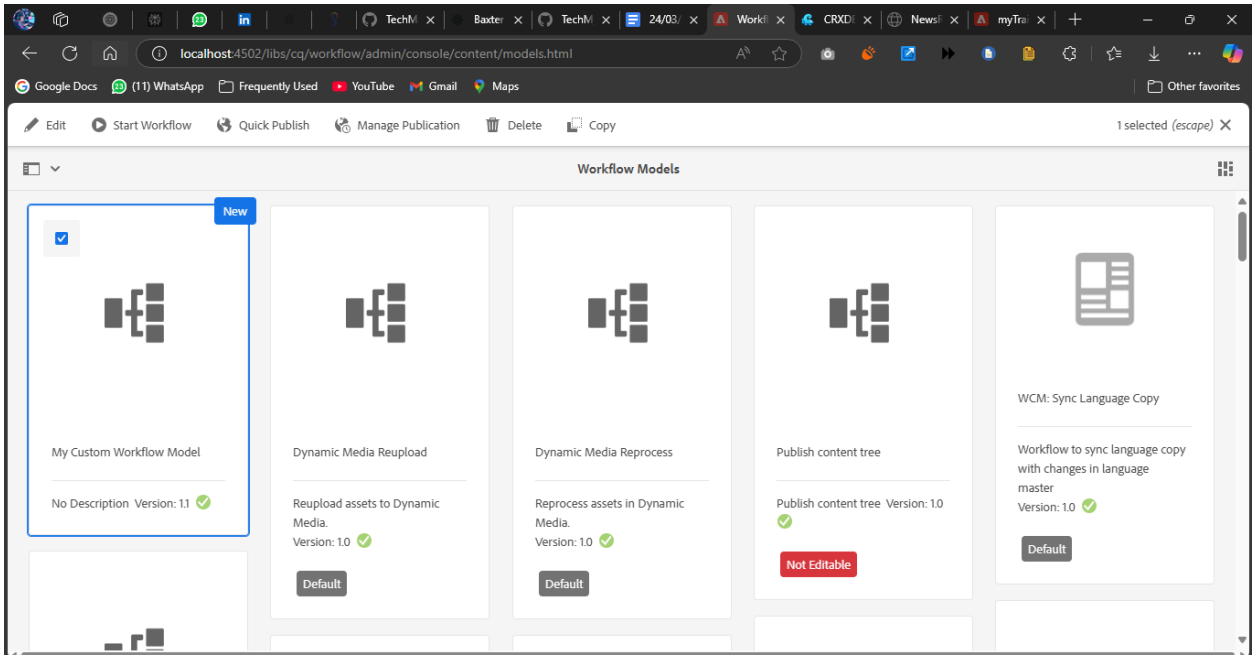
Table of Contents:

Create Custom Workflow Model

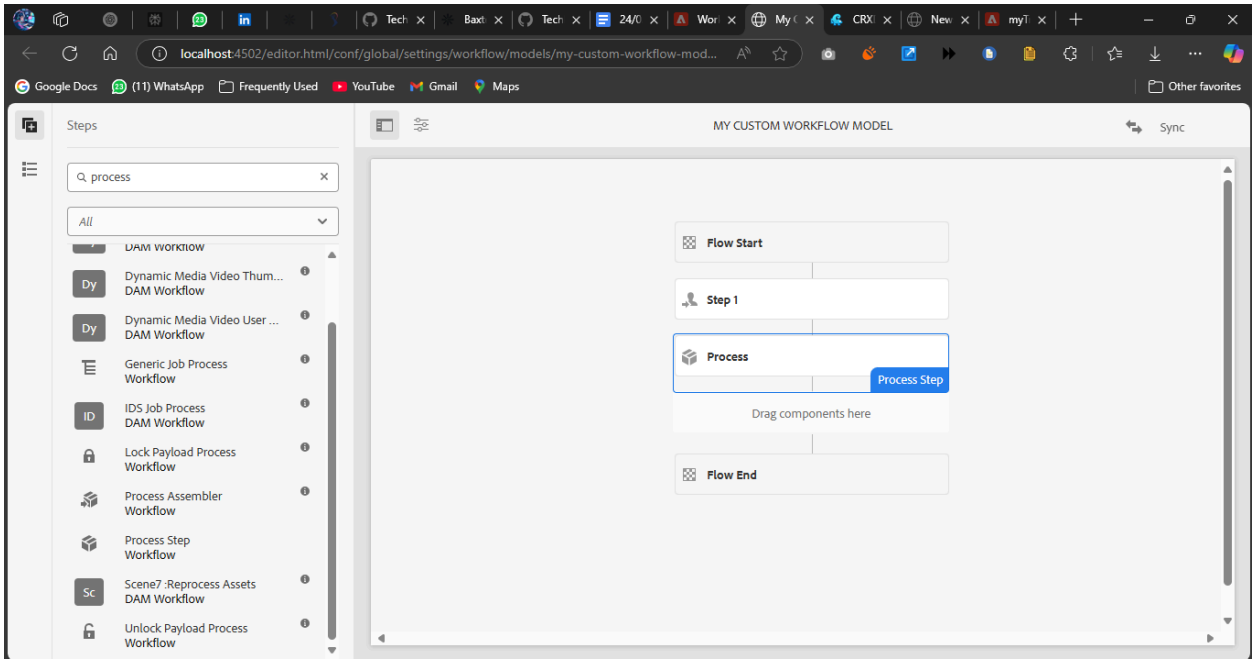
Steps:

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

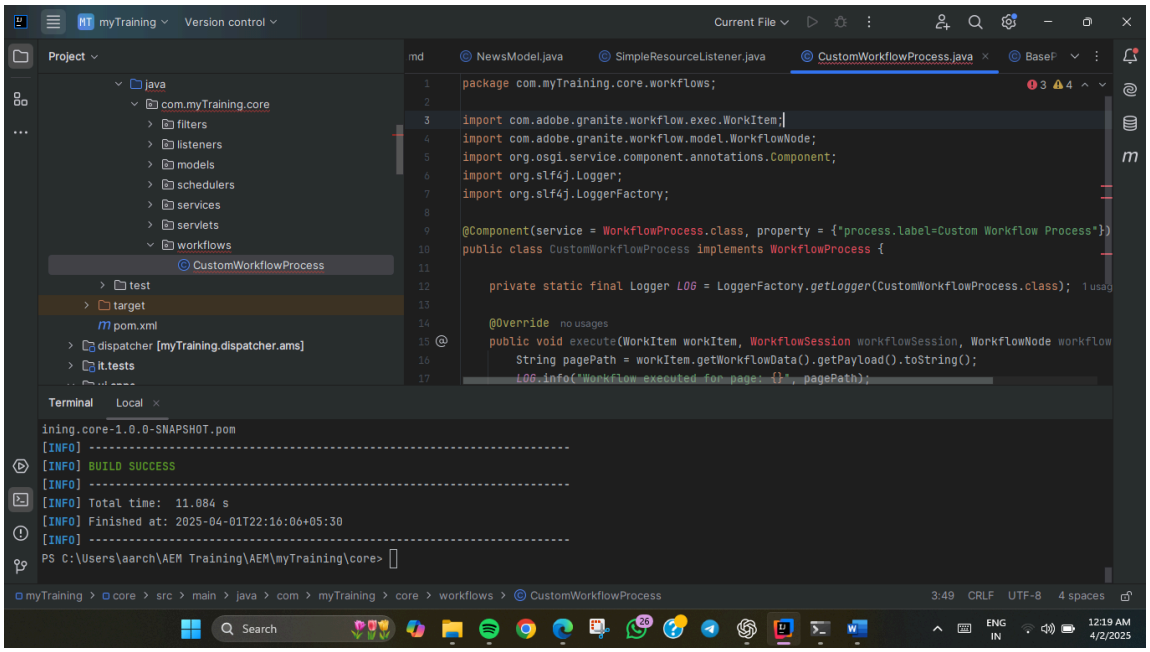
4. Open the model and click **Edit**.

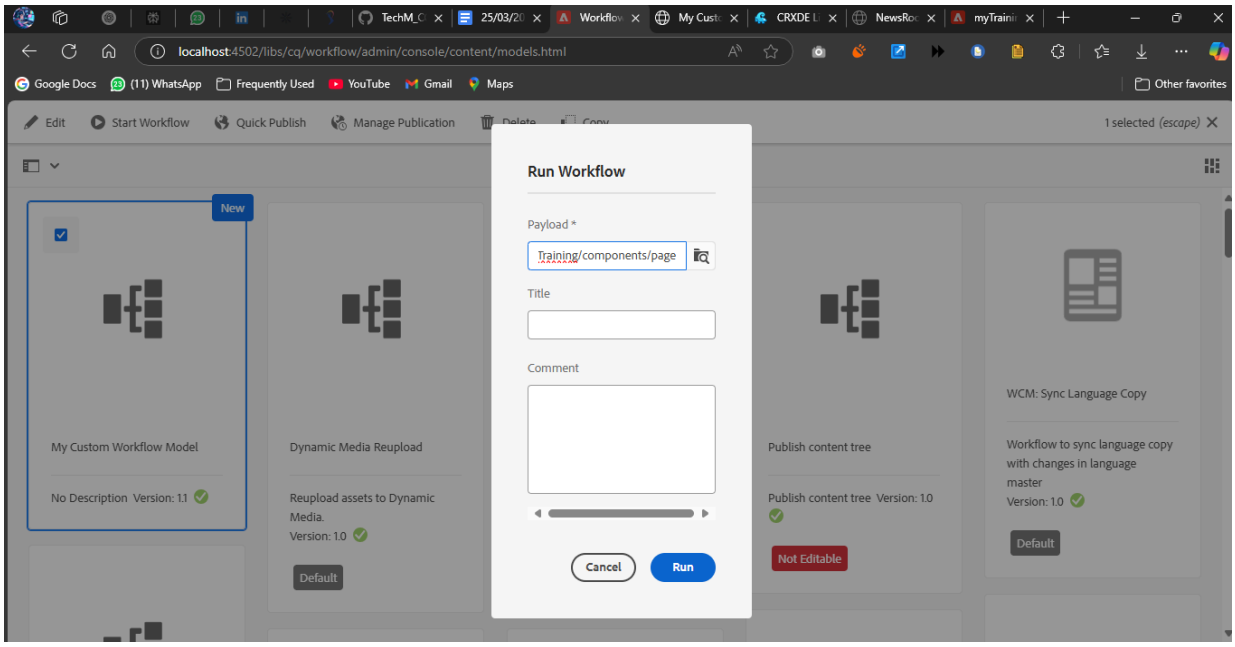


5. Drag and drop a **Process Step**.



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.

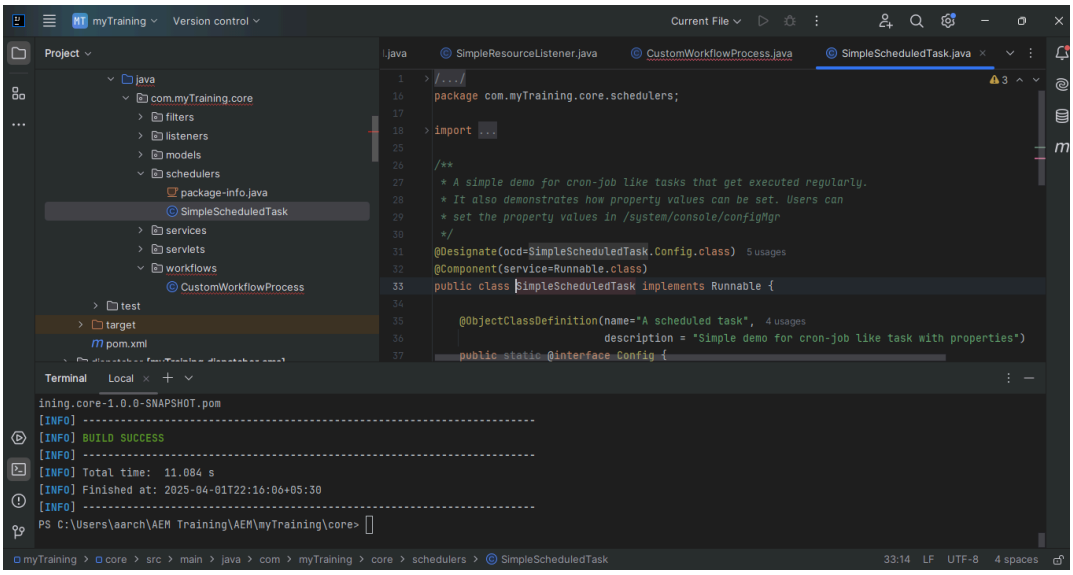
Create Scheduler

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!");
    }
```

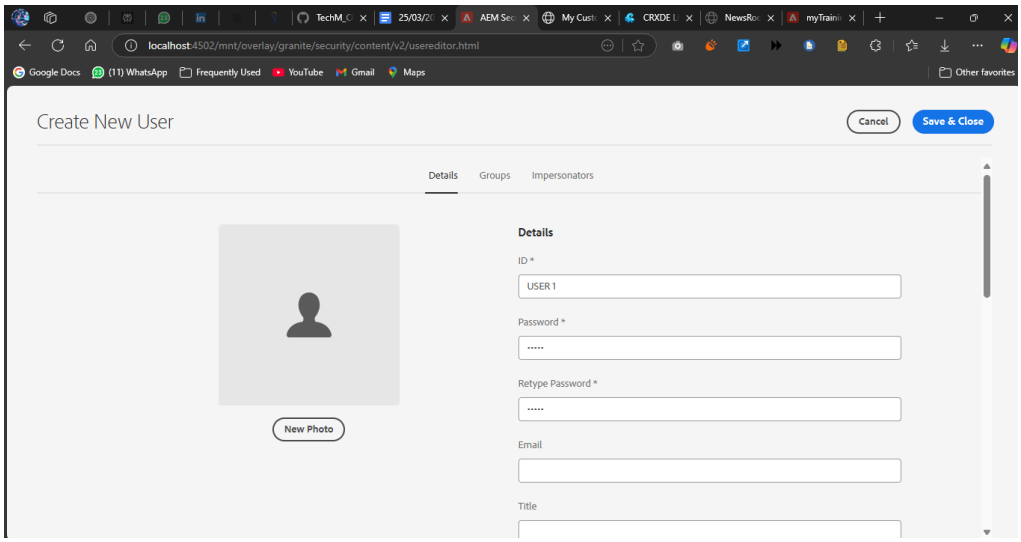


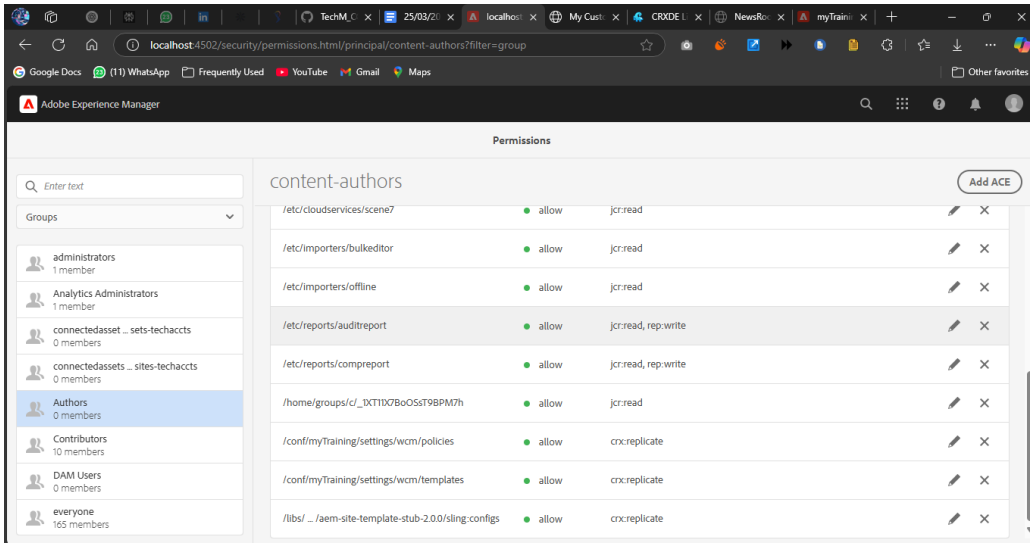

```
}
```

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.





26/03/2025 - TASKS

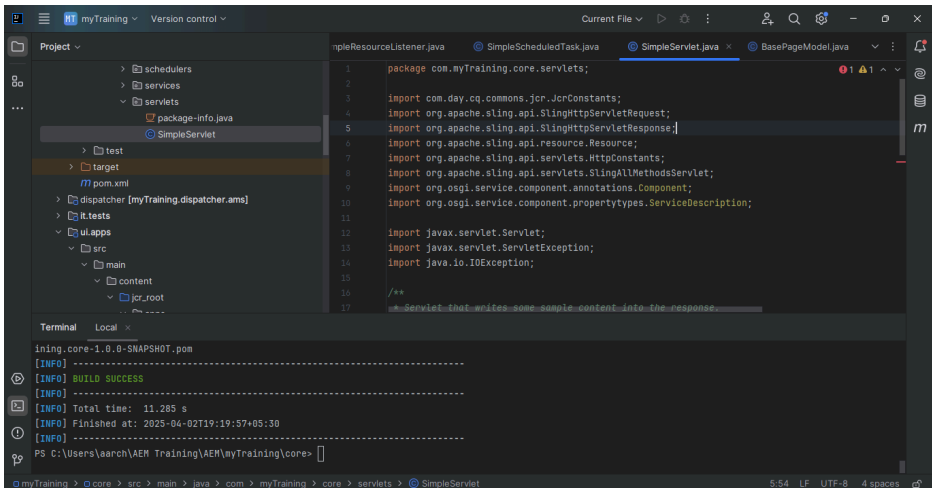
Table of Contents

1. Create `SampleServlet`

- Create a servlet that responds with a JSON message.
- Register the servlet using **resourceType**.

Implementation

1. Extend `SlingAllMethodsServlet`.
2. Use `@SlingServletResourceTypes` annotation to register the servlet.
3. Handle `GET` requests and return a JSON response.



```
@Designate(ocd = SampleServlet.Config.class)
@SlingServletResourceTypes(resourceTypes = "myTraining/components/sample", methods =
{"GET"}, extensions = "json")
public class SampleServlet extends SlingAllMethodsServlet {
   @Override
   protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
throws ServletException, IOException {
      response.setContentType("application/json");
      response.getWriter().write("{\"message\": \"Hello from SampleServlet!\"}");
   }}
```

2. Create `CreatePageServlet`

Objective

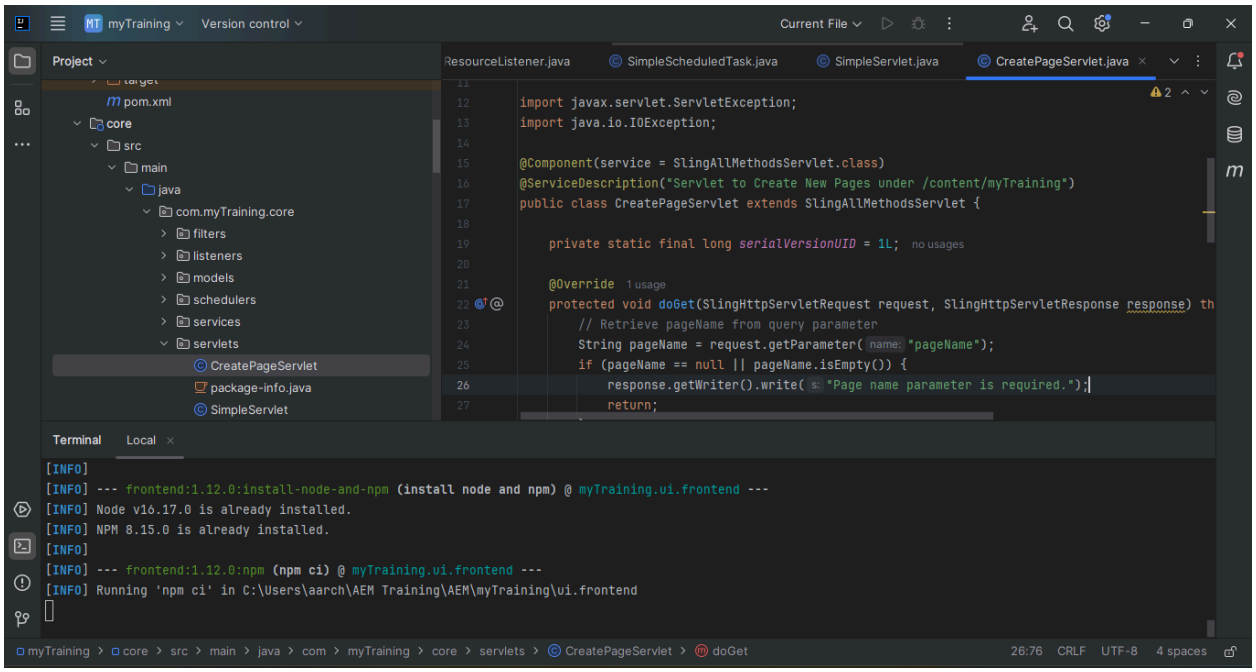
- Create a servlet to generate new pages dynamically in AEM.
- Register the servlet using a **path-based approach**.

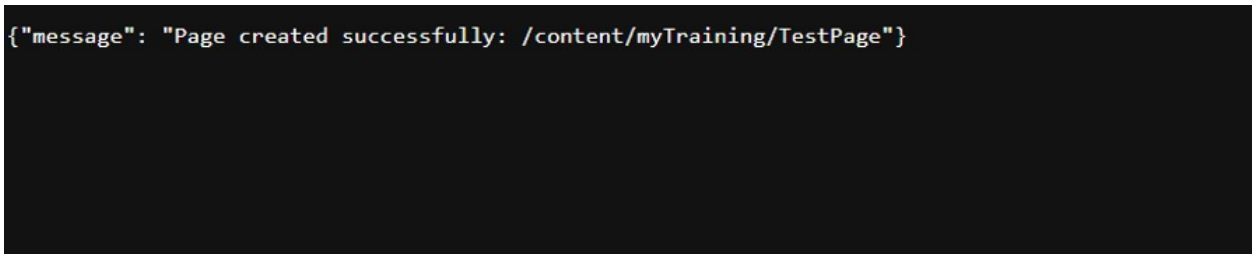
Implementation

1. Extend `SlingSafeMethodsServlet`.
2. Use `@SlingServletPaths` to define a unique servlet path.
3. Accept a page name from the user and create a new page in AEM.

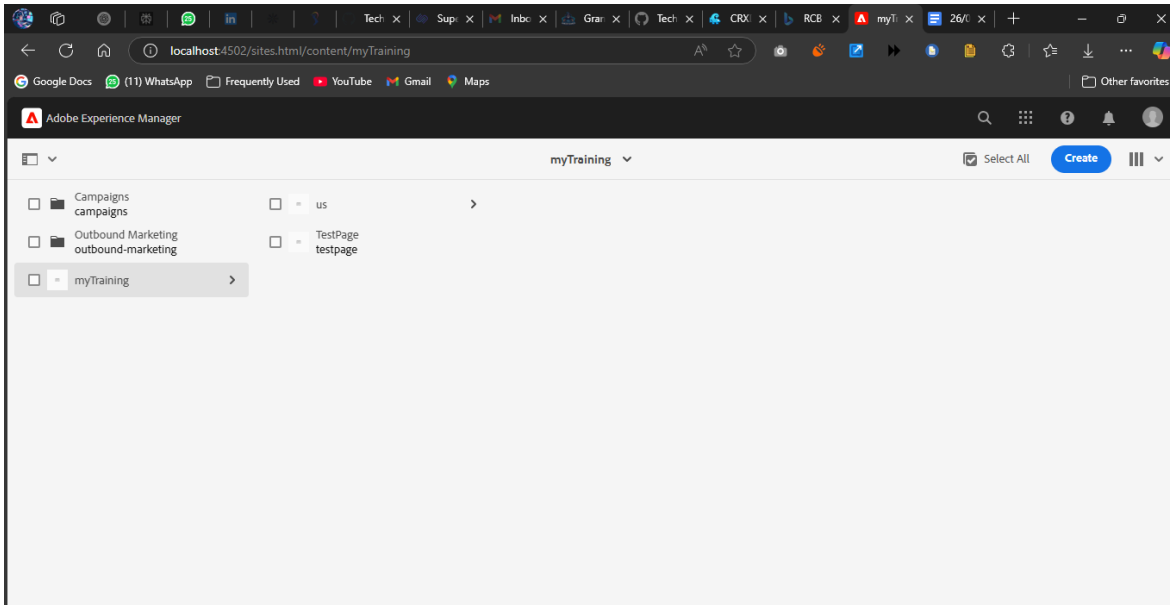
```
@SlingServletPaths("/bin/createpage")
public class CreatePageServlet extends SlingSafeMethodsServlet {
   @Reference
   private ResourceResolverFactory resolverFactory;

   @Override
   protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
throws ServletException, IOException {
      String pageName = request.getParameter("pageName");
      try (ResourceResolver resolver = resolverFactory.getServiceResourceResolver(null)) {
         PageManager pageManager = resolver.adaptTo(PageManager.class);
         if (pageManager != null) {
```

```
            Page newPage = pageManager.create("/content/us/en", pageName,
"/apps/wknd/components/page", pageName);
            response.getWriter().write("Page created successfully: " + newPage.getPath());
        }
    } catch (Exception e) {
        response.getWriter().write("Error: " + e.getMessage());
    }
  }
}
```





Page Created Succesfully:



3. Create Pages in AEM via User Input

Objective

- Allow users to specify a page name and dynamically generate pages.

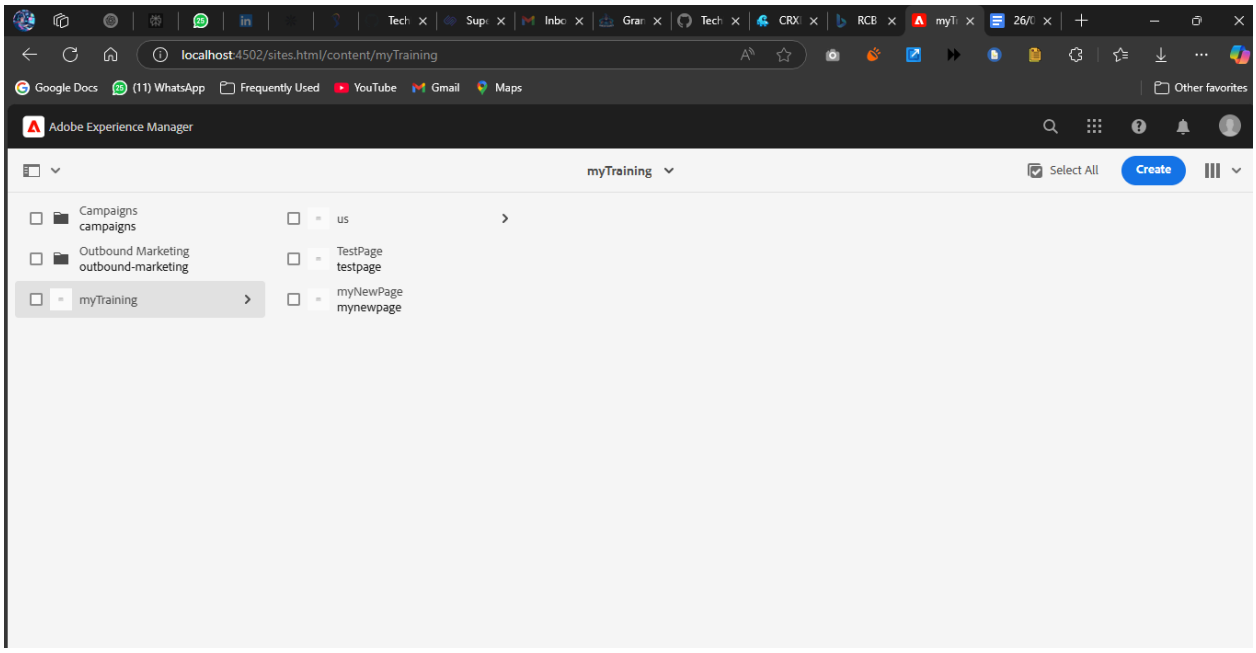
Steps

1. The servlet reads the `pageName` from the request parameter.
2. Uses the `PageManager` API to create a new page inside `/content/us/en`.
3. Returns success or error messages as responses.

Access the Servlet via:

```bash
http://localhost:4502/bin/createpage?pageName=myNewPage
```

Replace `myNewPage` with your desired page name.



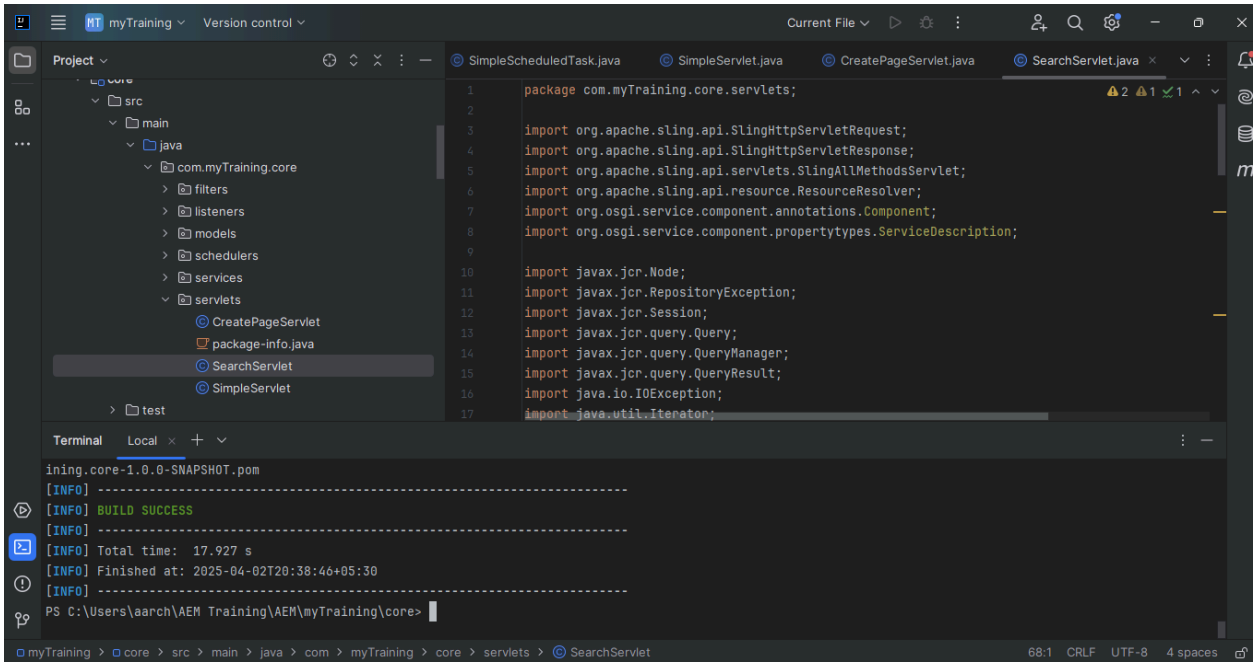
4. Implement `SearchServlet` Using PredicateMap

Objective

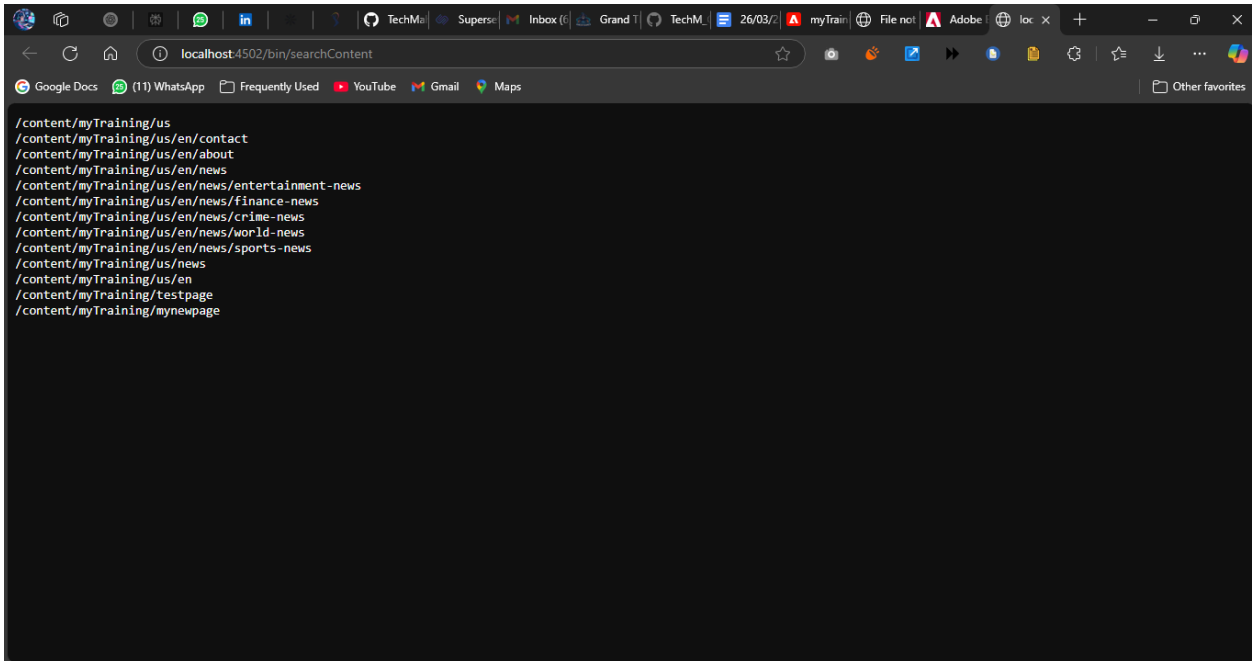
- Implement a search functionality in AEM.
- Use **QueryBuilder API** and `PredicateMap` to filter results.

Implementation

1. Extend `SlingSafeMethodsServlet`.
2. Use `@SlingServletPaths` to define the endpoint `/bin/searchcontent`.
3. Construct a `PredicateMap` with search criteria.
4. Fetch search results and return matching page paths.



Search Results:



27/03/2025 - TASKS

Table of Contents

1. Understanding MSM in AEM

Multi-Site Management (MSM) is an AEM feature that allows you to manage content across multiple websites efficiently. It enables you to create a master site (Blueprint) and generate localized or region-specific copies (Live Copies), making the process of managing and maintaining large, multi-regional websites more streamlined.

Components in MSM:

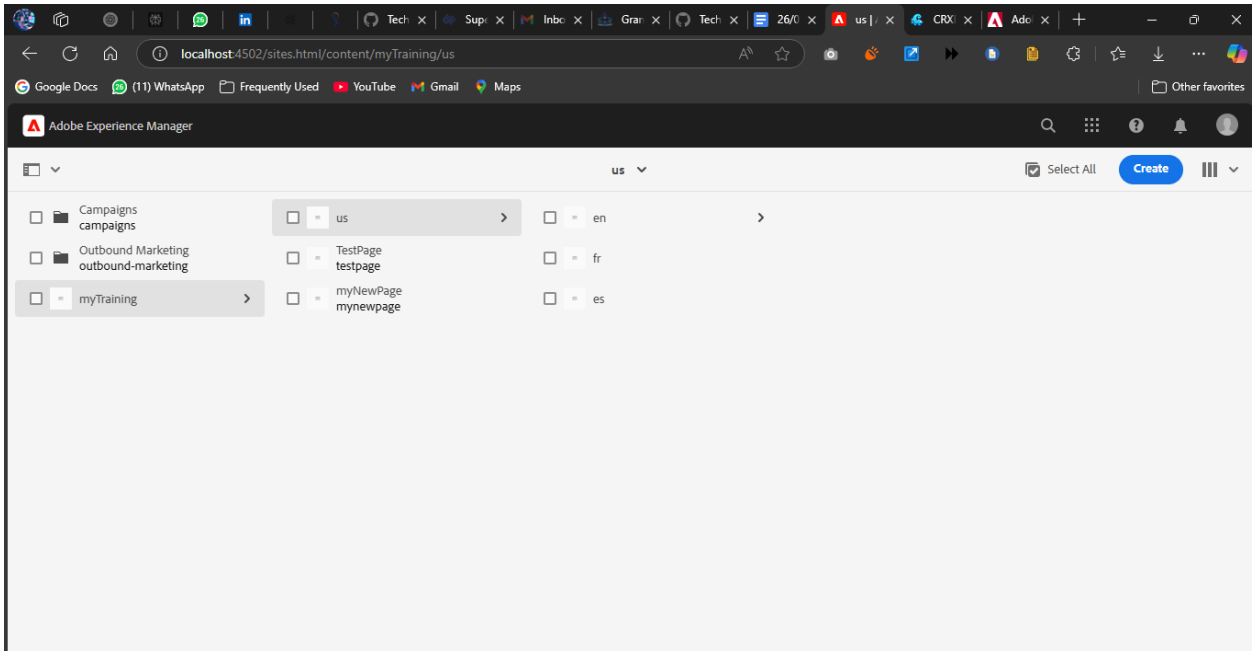
- **Blueprint**: The master site that holds the original content.
- **Live Copy**: A copy of the Blueprint. These pages inherit the structure and content from the Blueprint.
- **Rollout**: The action of pushing changes from the Blueprint to the Live Copy, ensuring that the copies are kept up-to-date with the original content.

2. Language Copy in AEM

A **Language Copy** is used to create a version of your website in different languages. Instead of manually duplicating content for each language, AEM allows you to manage multiple language versions of the site through **Language Copies**.

Example:

- `/content/us/en`: English version
- `/content/us/fr`: French version
- `/content/us/es`: Spanish version



These language copies allow you to manage content in multiple languages and translate or localize them efficiently.

---

3. Creating a Site Using MSM (US EN as Source)

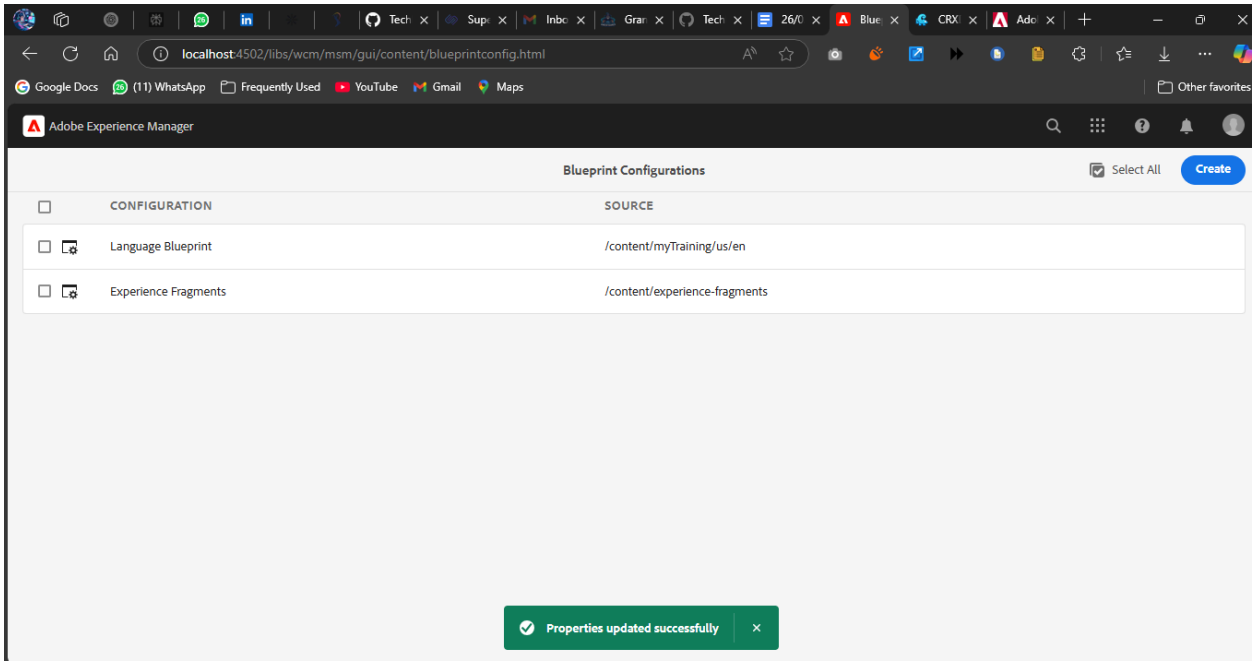
Steps to Create MSM Site:

1. **Navigate to MSM Control Center**:
    - Go to **AEM Sites → Tools → MSM Control Center**.
2. **Create a Blueprint Configuration**:
    - Select `/content/us/en` as the source for your blueprint configuration.
3. **Create a Live Copy**:

- Navigate to **Sites** and create a Live Copy using the newly created Blueprint. The Live Copy will automatically inherit content and structure from the Blueprint.

4. **Configure Rollout Settings**:

   - In the MSM Control Center, configure the rollout rules that define how content will propagate from the Blueprint to the Live Copy.



---

4. Creating McDonald's "About Our Food" Page in AEM

Steps to Author the Page:

1. **Create a New Page**:

   - Go to **AEM Sites** and create a new page using the **We.Retail template**.

2. **Add Components to the Page**:

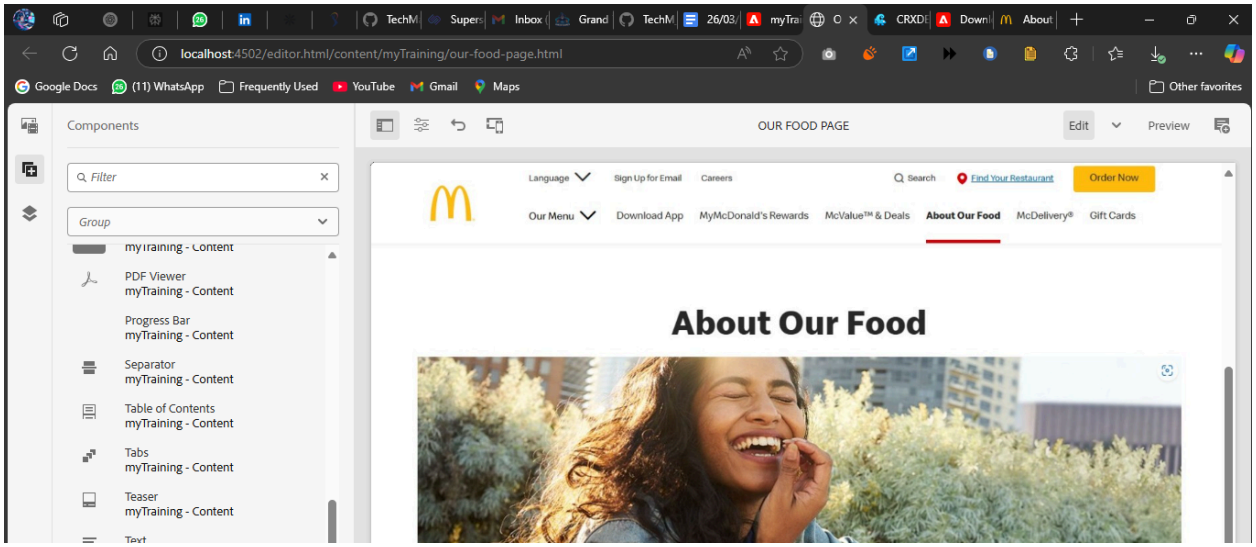
   - **Image Component**: Add images from McDonald's website.

   - **Text Component**: Copy relevant text content from the McDonald's About page.

- **Carousel Component**: Use a carousel for showcasing banner images or slides.

3. **Structure the Page Layout**:

   - Arrange the components in a layout similar to McDonald's "About Our Food" page. Refer to McDonald's page here.



By using AEM's built-in components, you can create a similar layout and design while customizing content as needed.

---

5. Setting Up Dispatcher on Local

The **Dispatcher** is an important AEM component used for caching and load balancing. It helps optimize performance by caching content on the web server side and controlling how the request is forwarded to AEM.

Steps to Install and Configure Dispatcher:

1. **Download Dispatcher Module**:

   - Obtain the dispatcher module from the official Adobe website: Adobe Dispatcher Download.

2. **Install Apache Web Server**:

   - On your local machine, install Apache web server if it is not already installed:

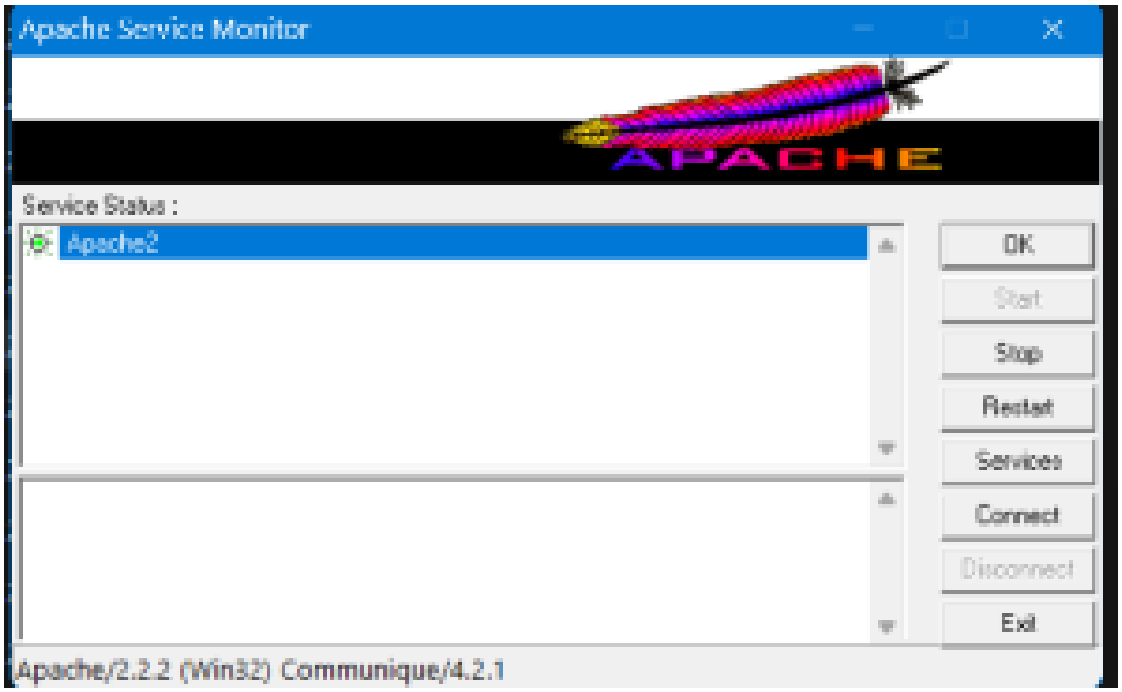
sudo apt update

sudo apt install apache2

3. **Configure Dispatcher**:
    - Extract the Dispatcher module and configure it by modifying the `dispatcher.any` configuration file to include caching rules, security configurations, and URL mapping.
    - Modify `httpd.conf` to ensure that Apache serves requests to AEM correctly.
4. **Test Dispatcher**:
    - Once the dispatcher is configured, restart the Apache server and test if the content is being served from the cache using:

curl -I http://localhost:80/content/we-retail.html

5. This command will verify if the dispatcher is working by checking the HTTP response header.



28/03/2025 - TASKS

Table of Contents

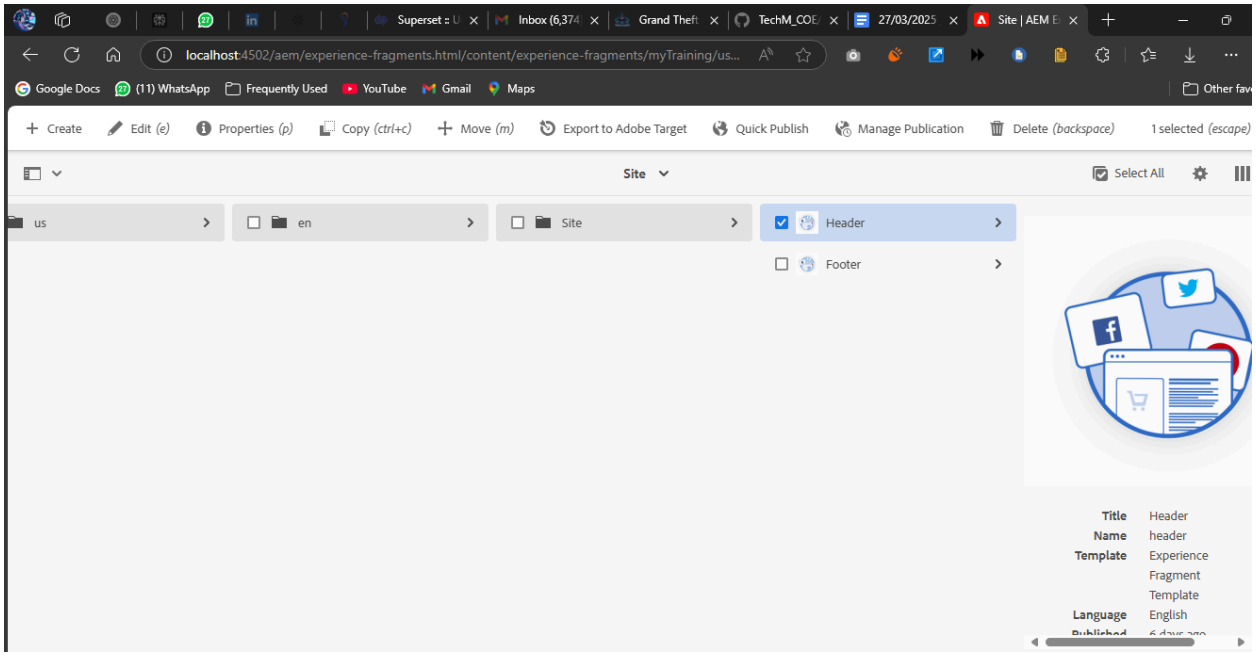
---

1. Header Setup

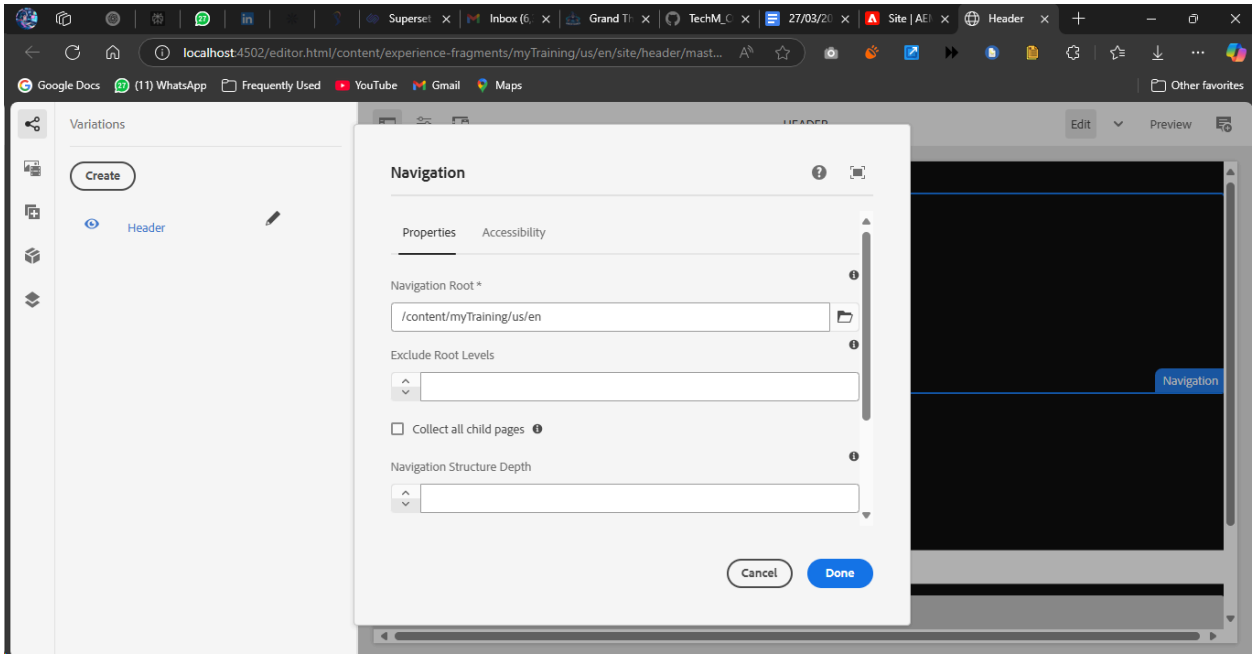
To set up the header for the Newsroom portfolio website, we will use the out-of-the-box (OOTB) header component in AEM.

Steps:

1. **Use the OOTB Header Component**:
    - Open the AEM Sites console and navigate to the page template or the Home page.
    - Use the **Header** component available in AEM to add the header to the template.

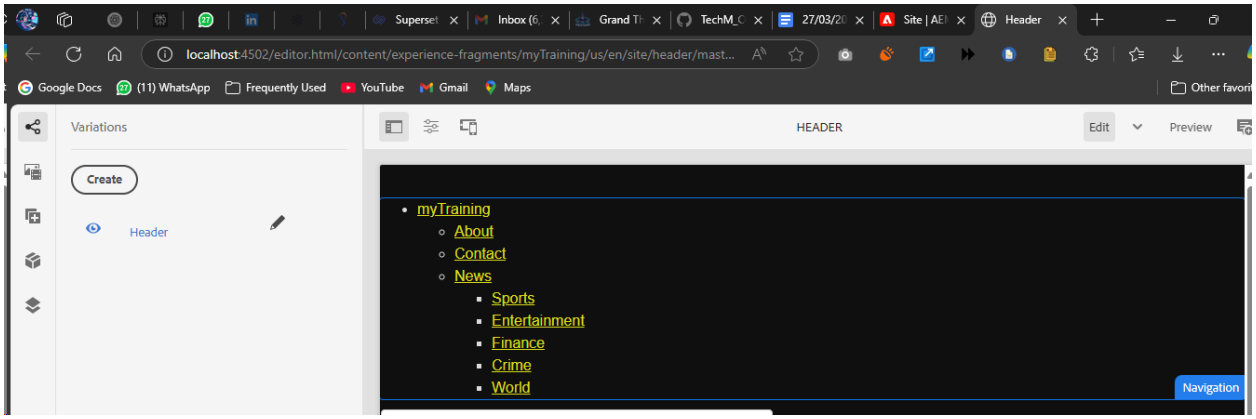


2. **Create Navigation Links**:
    - The navigation links in the header should include:
        - News (with a dropdown containing 5 news items)
        - About Me
        - Contact Me
        - Homepage



3. **Configure Drop-Down for News**:

    - For the **News** link, configure a drop-down that will display the latest 5 news articles. This can be done by adding a **teaser** component or **list component** to show news articles from the `/content/news` folder.

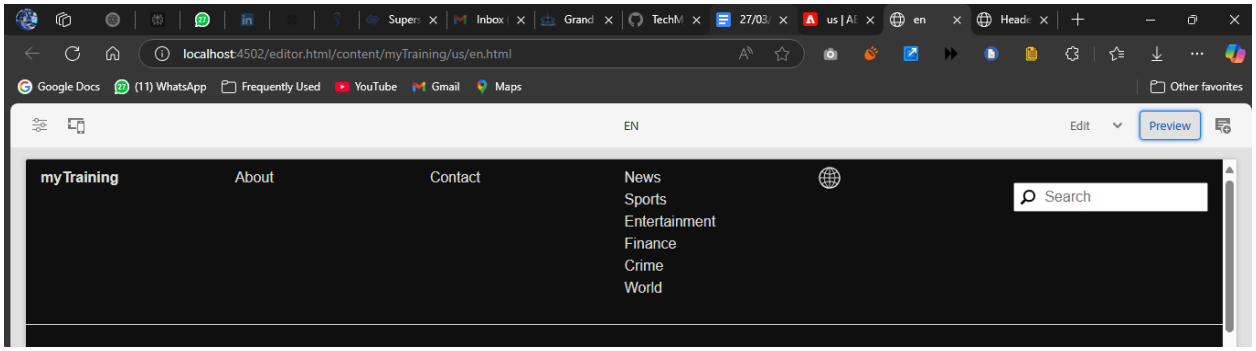


4. **Styling the Header**:

    - Use AEM's **Style System** or **CSS** to ensure that the header has proper styling, such as alignment, spacing, and colors.

5. **Ensure Navigation Links Work**:
    - Verify that the links are functioning and correctly navigate to their respective pages.

HEADER VIEW:

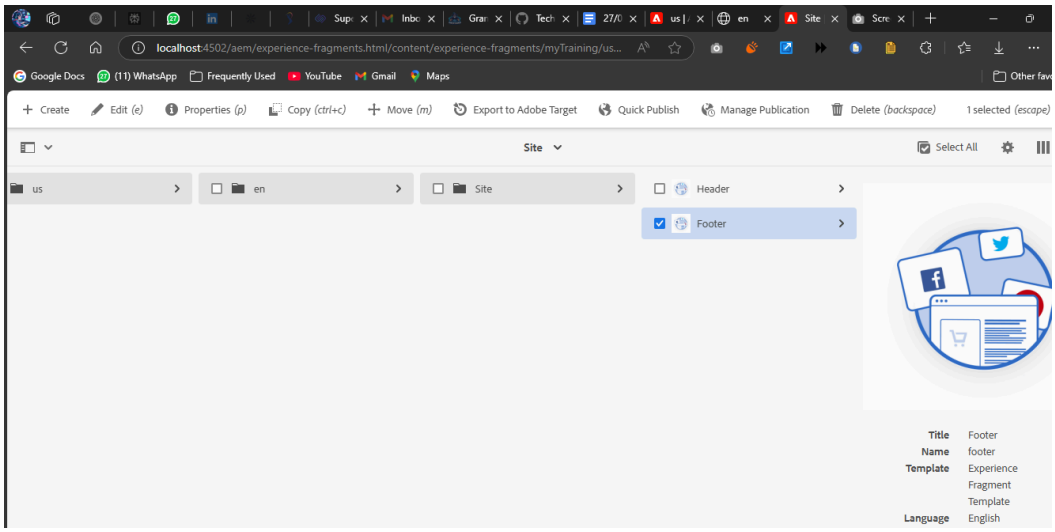


2. Footer Setup

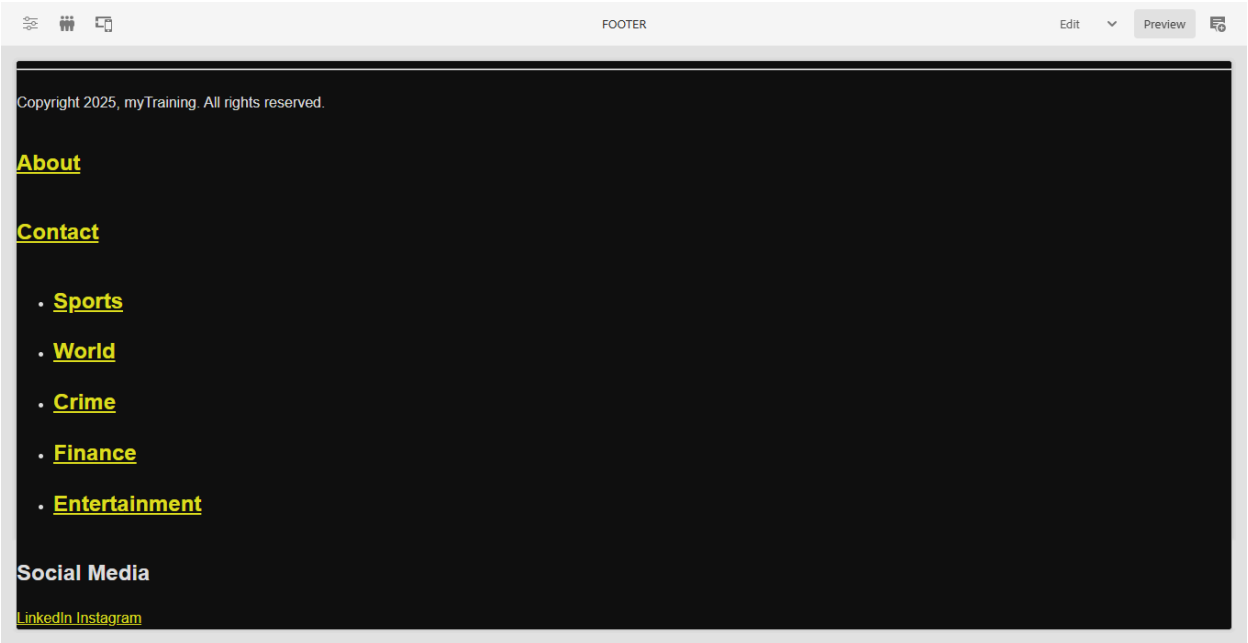
The footer should contain key links and information about the website, such as the About Me, Contact Me, and Latest News.

Steps:

1. **Add Footer to Template**:
    - Use the Footer component in AEM or create a custom footer by using text or links components to display information at the bottom of every page.



2. **Footer Sections**:
    - **About Me**: Link to the About Me page.
    - **Contact Me**: Link to the Contact Me page.
    - **Latest 5 News**: Display the latest 5 news articles with links. These should open in a new tab when clicked.
3. **Social Media Section**:
    - Add social media links (Facebook, Twitter, Instagram, LinkedIn, etc.) using link components.



3. Home Page

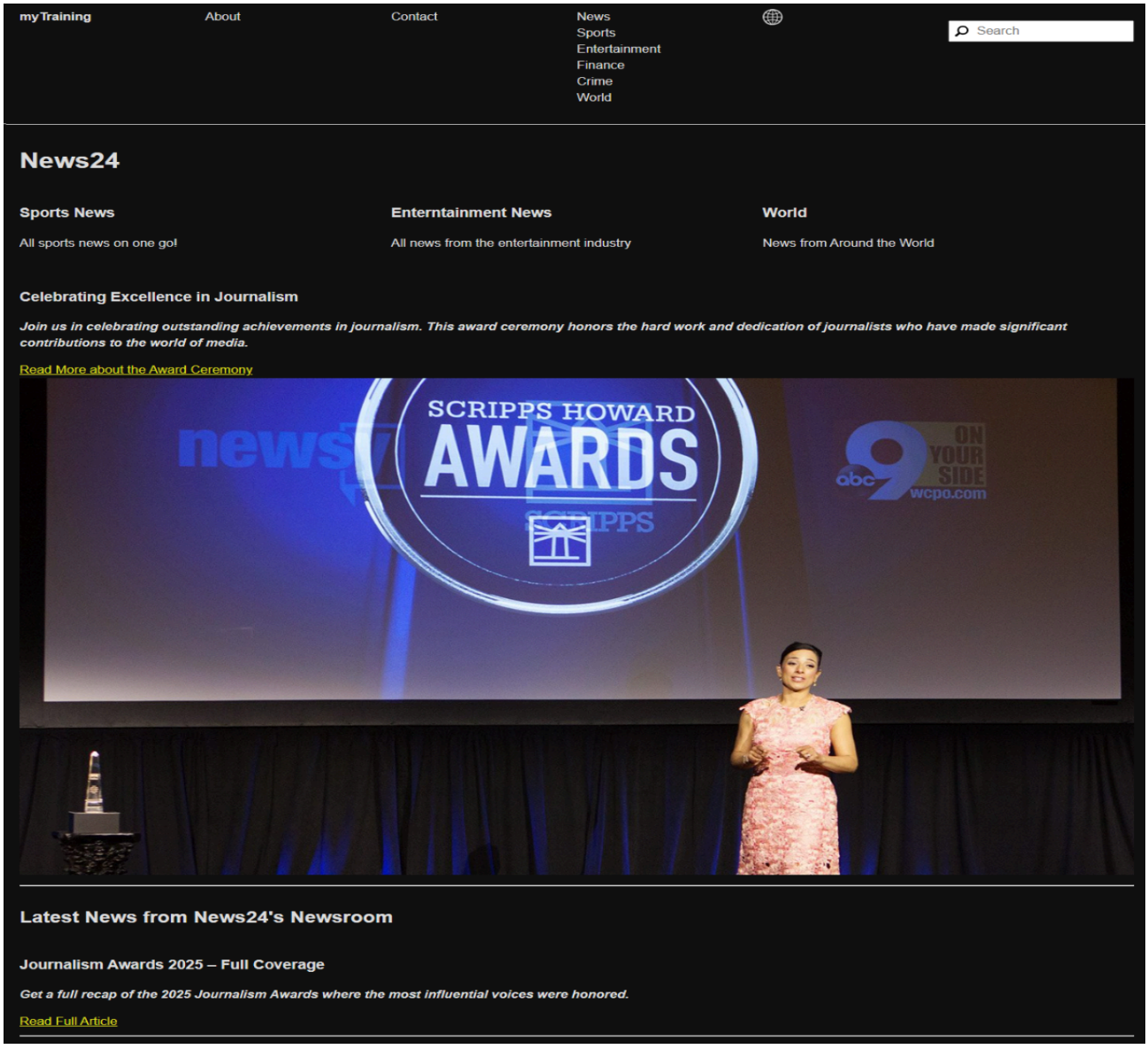
The home page will highlight a specific award ceremony for journalists using a teaser component.

Steps:

1. **Teaser Component for Award Ceremony**:
    - Use the **Teaser** component to display an award ceremony for journalists.
    - The teaser should have:

- **Image**: Display an image of the award ceremony.
- **Title**: Title of the award.
- **Description**: A brief description of the award.
- **CTA (Call to Action)**: Link to a page with more details about the award. Create a new page to author the full news article and link it to the CTA.

2. **Style and Layout**:
    - Use AEM's style system or custom CSS to ensure the layout and design match the desired outcome for the homepage.



4. News Page

For the News page, we need to create a **NewsRoom** component that will dynamically fetch and display news articles.

Steps:

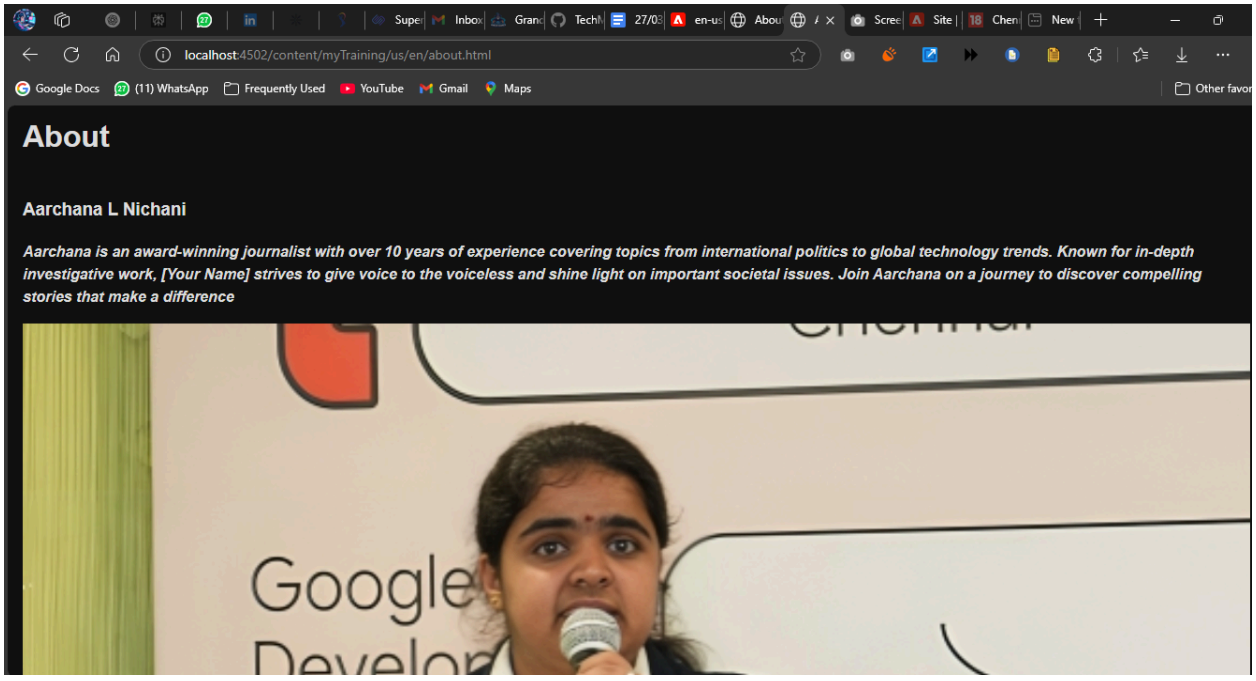
1. **Create NewsRoom Component**:
    - The **NewsRoom** component should fetch all the news articles from the `/content/news` folder.
    - The component should display the articles as cards, each with:
        - **Title**
        - **News Detail**
        - **Image**

5. About Me Page

The About Me page will display information about the journalist using a custom **Journalist Component**.

Steps:

1. **Use Journalist Component**:
    - Use the **Journalist** component to display information about the journalist. This component should show:
        - **Name**
        - **Bio**
        - **Image**
2. **Design Layout**:
    - The layout should be simple and clean, displaying relevant information about the journalist.

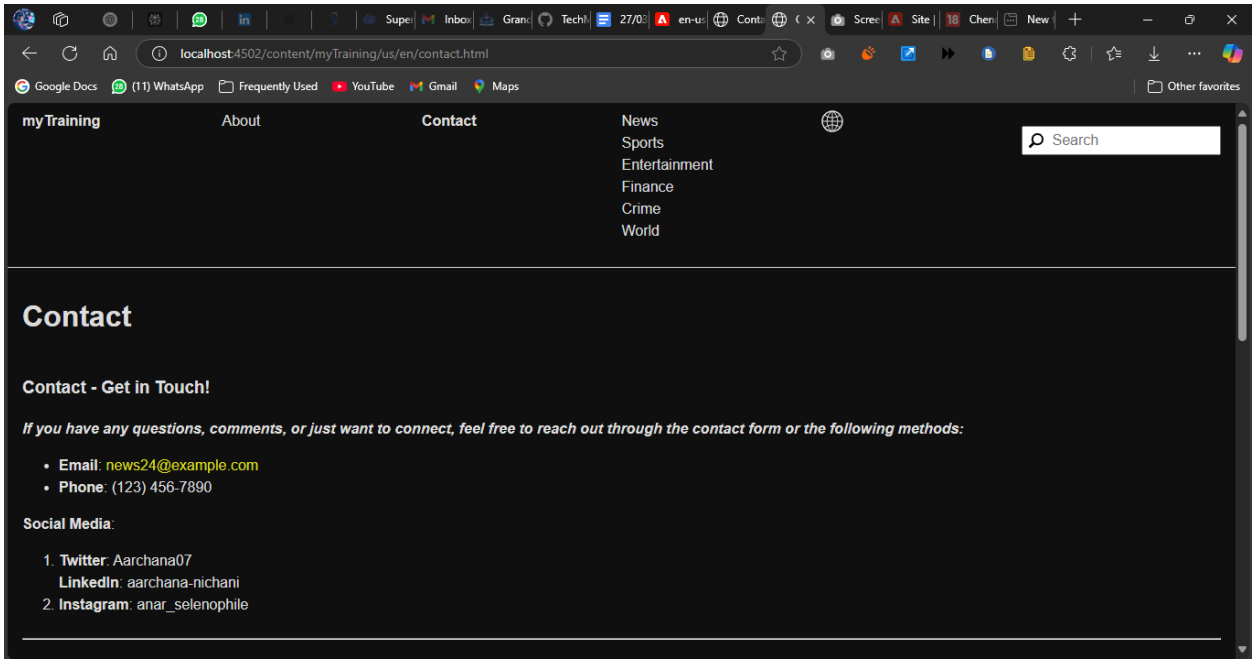


6. Contact Me Page

The Contact Me page will allow users to reach out to the journalist. This can include basic contact details and social media profiles.

Steps:

1. **Add Contact Information**:
    - Use the **Teaser** or **Text** component to add the contact details.
2. **Social Media Links**:
    - Include social media links (such as Twitter, LinkedIn, etc.) where users can follow the journalist.



7. Template Setup

Both the header and footer should be part of the template so that they can be reused on multiple pages.

Steps:

1. **Create a Template**:
    - Use the OOTB **Content Page Template** for the homepage.

- Ensure the header and footer are placed at the template level so they appear on all pages.

2. **Set Template for Pages**:

   - When creating pages, make sure they are using this template to automatically include the header and footer sections.

CREATED THE NEWSROOM TEMPLATE: USES HEADER AND FOOTER BY DEFAULT: