AEM Assignment 4

---

1. Create 5 News Article Pages Under `/content/us/en/news`

Steps:

1. Go to **AEM > Sites > We.Retail (or your project name) >** `us/en/news/`.
2. Click **Create > Page** and select **News Room Template**.
3. Create **5 unique pages**, for example:
   - **Article 1**: "AI Revolution in 2025"
   - **Article 2**: "New SpaceX Mission Announced"
   - **Article 3**: "Global Warming Report 2025"
   - **Article 4**: "Tech Giants Battle for AI Dominance"
   - **Article 5**: "Electric Cars: Future of Transportation"

Modify Each Page Using News Component

1. Open each page in **AEM Editor** and add the **News Component**.
2. Fill in the details:
   - **Title**: Headline of the news
   - **News Detail**: Short description
   - **Published Date**: Current date

---

2. Create Header Experience Fragment (XF)

We need a **global header XF** to include navigation to:

- **News** (menu with news pages)
- **About Me**
- **Contact Us**

Steps to Create Header XF

1. Go to **AEM > Experience Fragments > Create XF**.
2. Name it **"Header"** and use **Site Template**.
3. Open XF Editor and add a **Navigation Component**:
    - Link `/content/us/en/news` under **News Menu**.
    - Add two more links: `/content/us/en/about-me` & `/content/us/en/contact-us`.
4. Save and publish.

---

3. Create "About Me" and "Contact Us" Pages

About Me Page (`/content/us/en/about-me`)

1. **Create a new page** using the Base Page Template.
2. Add **Text Component** with:
    - **Name**: Journalist Name
    - **Bio**: Short introduction
    - **Image Component**: Add a picture

Contact Us Page (`/content/us/en/contact-us`)

1. **Create a new page** using the Base Page Template.
2. Add **Text Component** with:
    - **Office Address**
    - **Email Address**
    - **Mobile Number**

---

4. Create Footer XF (Experience Fragment)

Footer XF should contain **4 sections**:

1. **News Section** → Use **List Component** to display 4 latest news articles.
2. **About Me Section** → Use **Text Component** for a short bio.
3. **Contact Us Section** → Use **Text Component** with contact details.
4. **Social Media Section** → Use **List Component** for social media links.

Steps to Create Footer XF

1. Go to **AEM > Experience Fragments > Create XF**.
2. Name it **"Footer"** and use **Site Template**.
3. Open XF Editor and add components:
   - **List Component** (News Menu Section)
   - **Text Component** (About Me + Contact Us)
   - **List Component** (Social Media)
4. Save and publish.

---

5. Create a Custom Service to Print "Hello World"

We will create an **OSGi Service** to print `"Hello World"` and call it from the **News Component Sling Model**.

Steps:

1. **Create a Java Interface (`HelloService.java`)**

```
package com.myTraining.core.services;

public interface HelloService {
    String getMessage();
}
```

2. **Implement the Service (`HelloServiceImpl.java`)**

```
package com.myTraining.core.services.impl;

import com.myTraining.core.services.HelloService;
import org.osgi.service.component.annotations.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component(service = HelloService.class, immediate = true)
public class HelloServiceImpl implements HelloService {

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

```
    @Override
    public String getMessage() {
        String message = "Hello World from Custom Service!";
        LOG.info(message);
        return message;
    }
}
```

3. **Call the Service in News Component Sling Model (`NewsModel.java`)**

```
package com.myTraining.core.models;

import com.myTraining.core.services.HelloService;
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.OSGiService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.inject.Inject;

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

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

    @OSGiService
    private HelloService helloService;

    @Inject
    private String newsTitle;

    @Inject
    private String newsDetail;

    @Inject
    private String newsDate;

    public String getNewsTitle() {
        return newsTitle;
    }
    public String getNewsDetail() {
        return newsDetail;
    }

    public String getNewsDate() {
        return newsDate;
    }

    public String getHelloMessage() {
        String message = helloService.getMessage();
        LOG.info("News Component - Service Message: {}", message);
        return message;
    }
}
```

4. **Modify `news.html` to Print the Message**

```
<div>
    <h2>${newsModel.newsTitle}</h2>
    <p>${newsModel.newsDetail}</p>
    <span class="date">${newsModel.newsDate}</span>
    <p>Service Message: ${newsModel.helloMessage}</p>
</div>
```

---

6. Create Custom Configuration for a 3rd Party API

Steps to Create OSGi Configuration

1. **Create a Java Interface (`ApiConfigService.java`)**

```
package com.myTraining.core.services;

public interface ApiConfigService {
    String getApiUrl();
}
```

2. **Implement the Service (`ApiConfigServiceImpl.java`)**

```java
package com.myTraining.core.services.impl;

import com.myTraining.core.services.ApiConfigService;
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.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;

@Component(service = ApiConfigService.class, immediate = true)
@Designate(ocd = ApiConfigServiceImpl.Config.class)
public class ApiConfigServiceImpl implements ApiConfigService {

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

    private String apiUrl;

    @ObjectClassDefinition(name = "API Configuration")
    public @interface Config {
        @AttributeDefinition(name = "API URL")
        String api_url() default "https://jsonplaceholder.typicode.com/posts";
    }

    @Activate
    @Modified
    protected void activate(Config config) {
        this.apiUrl = config.api_url();
        LOG.info("Configured API URL: {}", apiUrl);
    }

    @Override
    public String getApiUrl() {
        return apiUrl;
    }
}
```

3. **Call API and Print in Logs (`ApiClient.java`)**

```java
package com.myTraining.core.services.impl;

import com.myTraining.core.services.ApiConfigService;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;

@Component(service = Runnable.class, immediate = true)
public class ApiClient implements Runnable {

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

    @Reference
    private ApiConfigService apiConfigService;

    @Override
    public void run() {
        try {
            URL url = new URL(apiConfigService.getApiUrl());
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            Scanner scanner = new Scanner(conn.getInputStream());
            while (scanner.hasNext()) {
                LOG.info(scanner.nextLine());
            }
            scanner.close();
        } catch (IOException e) {
            LOG.error("Error fetching API data", e);
        }
    }
}
```