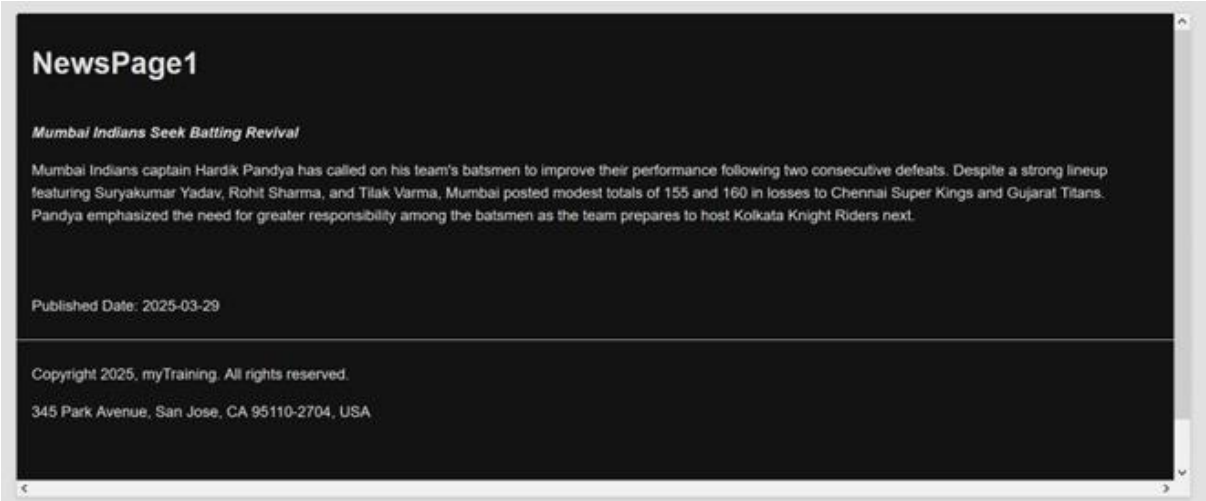
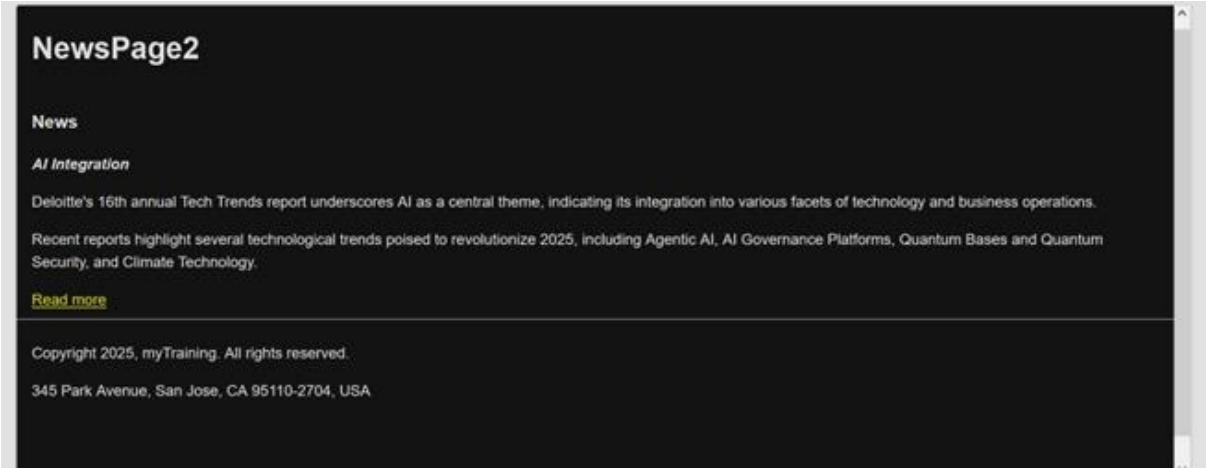
Creating a News Website in AEM

Create 5 News Article Pages under /content/us/en/news

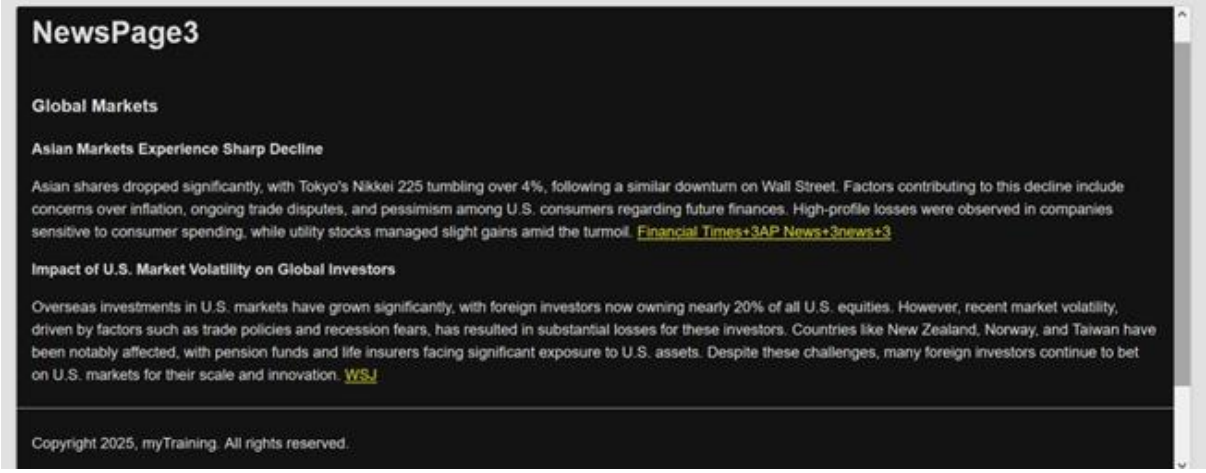
1. Navigate to Sites Console (/content/us/en/news).
2. Click Create Page → Select News Room Page Template.
3. Provide unique titles for five articles (e.g., "Tech Innovations 2025", "Political Insights", "Sports Update", etc.).
4. Use the News Component to add:
    - Title
    - News Detail
    - Published Date
5. Publish all pages.





Create Header Experience Fragment

1. Navigate to Experience Fragments Console (/content/experience-fragments/us/en).
2. Create a new Experience Fragment named Header-XF.
3. Add a Navigation Component and configure the menu:
    - News (Menu item linking to /content/us/en/news)
    - Contact Us (/content/us/en/contact)
    - About Me (/content/us/en/about-me)
4. Publish the Experience Fragment.

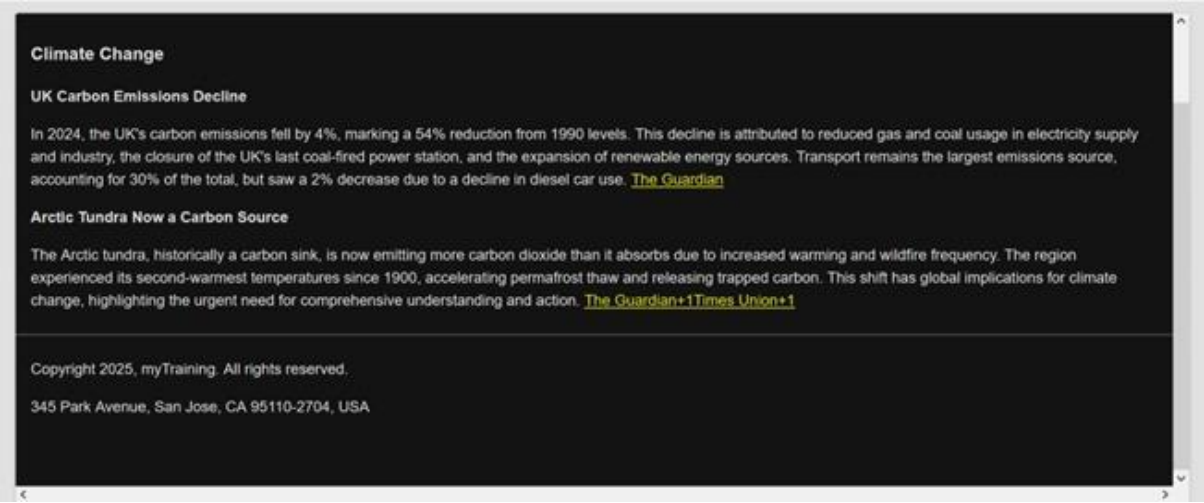


Create "About Me" and "Contact Us" Pages

1. Create a new page using the Base Page Template.
2. Add the following components:
    - Teaser Component → Add Image of Journalist + Title.
    - Text Component → Add journalist details.

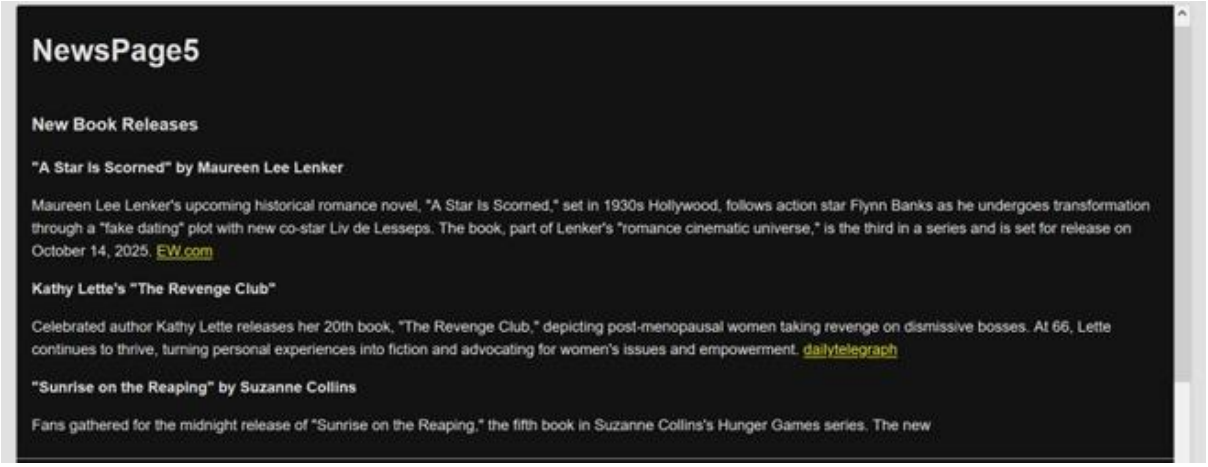
Contact Us Page (/content/us/en/contact)

1. Create a new page using the Base Page Template.
2. Add the following components:
    - Text Component → Display Phone, Email, Office Address.



Create Footer Experience Fragment

1. Navigate to Experience Fragments Console (/content/experience-fragments/us/en).
2. Create a new Experience Fragment named Footer-XF.
3. Add the following sections:
    - News Menu Section: Use List Component (Select 4 News Articles).
    - About Me Section: Use Text Component with bio details.
    - Contact Us Section: Use Text Component with contact details.
    - Social Media Section: Use List Component with social media links.
4. Publish the Experience Fragment.



Create a Custom OSGi Service to Print "Hello World"

1. Navigate to **Core Module (ui.apps.core)** in your AEM project.
2. Create a new OSGi Service (HelloWorldService.java):

```
@Designate(ocd = HelloWorldService.Config.class)
@Component(service = HelloWorldService.class, immediate = true)
public class HelloWorldService {
```

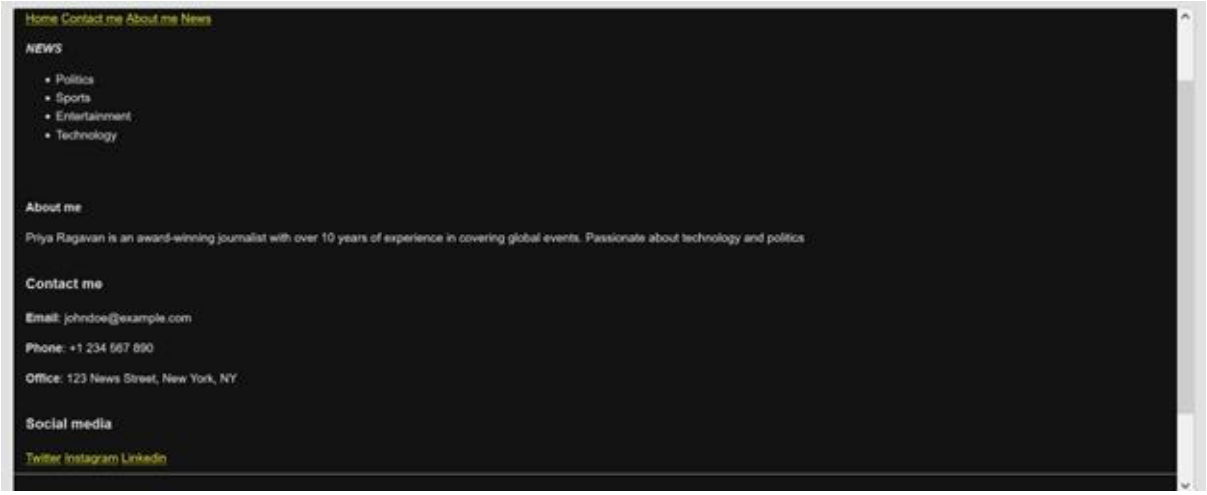
```
    @ObjectClassDefinition(name = "Hello World Configuration")
    public @interface Config {}

    public String getHelloMessage() {
        return "Hello, World!";
    }
}
```

Modify **News Component Sling Model** to call this service:

```
@Model(adaptables = SlingHttpServletRequest.class, defaultInjectionStrategy
= DefaultInjectionStrategy.OPTIONAL)
public class NewsComponentModel {
    @OSGiService
    private HelloWorldService helloWorldService;

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

    @PostConstruct
    protected void init() {
        String message = helloWorldService.getHelloMessage();
        LOG.info("Message from Service: " + message);
    }

    public String getServiceMessage() {
        return helloWorldService.getHelloMessage();
    }
}
```

Deploy the service and check **logs (error.log)** for output:

Message from Service: Hello, World!



Create Custom Configurations for 3rd Party API

1. Navigate to **OSGi Configuration (/system/console/configMgr)**.
2. Create a new Configuration **"ThirdPartyAPIConfig"**:
    - Add a field for **API URL (https://jsonplaceholder.typicode.com/posts)**.
3. Create an OSGi Service to fetch and log API data:

```
@Designate(ocd = ThirdPartyAPIConfig.Config.class)
@Component(service = ThirdPartyAPIConfig.class, immediate = true)
public class ThirdPartyAPIConfig {
    @ObjectClassDefinition(name = "Third Party API Configuration")
    public @interface Config {
        @AttributeDefinition(name = "API Endpoint")
        String apiUrl() default "https://jsonplaceholder.typicode.com/posts";
    }
```

```
    @Activate
    @Modified
    protected void activate(Config config) {
        LOG.info("Fetching Data from API: " + config.apiUrl());
        String response = fetchData(config.apiUrl());
        LOG.info("API Response: " + response);
    }

    private String fetchData(String url) {
        try {
            HttpGet request = new HttpGet(url);
            CloseableHttpClient client = HttpClients.createDefault();
            CloseableHttpResponse response = client.execute(request);
            return EntityUtils.toString(response.getEntity());
        } catch (Exception e) {
            LOG.error("Error fetching API data", e);
            return "Error";
        }
    }
}
```

Deploy the configuration and check **logs (error.log)** for API response.