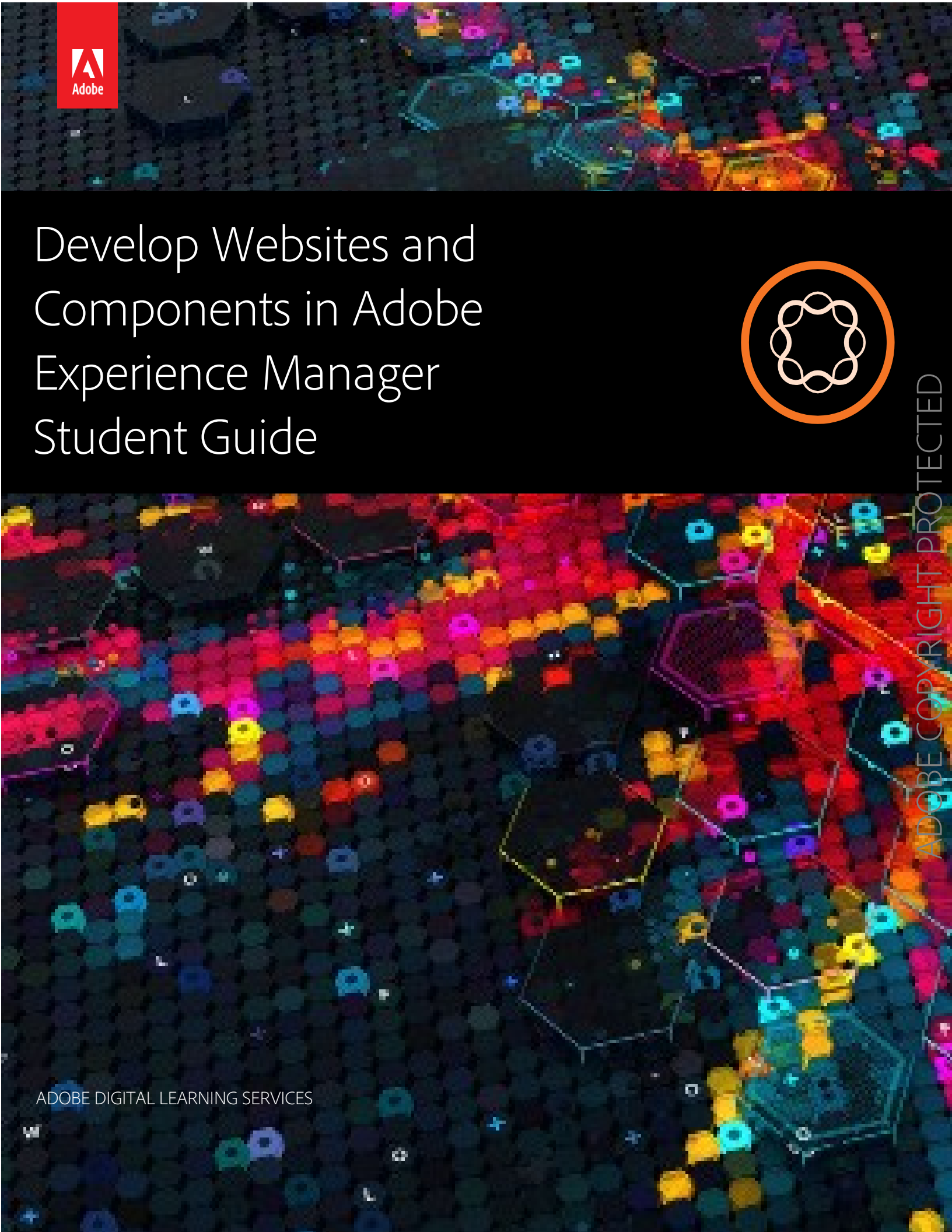
Develop Websites and Components in Adobe Experience Manager Student Guide



ADOBE DIGITAL LEARNING SERVICES

Develop Websites and Components in Adobe Experience Manager

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

December 20, 2017

TABLE OF CONTENTS

TABLE OF CONTENTS ..... **Error! Bookmark not defined.**
1 ARCHITECTURE OVERVIEW 8
What is Adobe Experience Manager (AEM)? 8
Infrastructure-Level Functions 8
Basics of the Architecture Stack 9
Introduction to the Granite Platform 9
OSGi Framework 9
Introduction to the Java Content Repository (JCR) 10
Introduction to Apache Sling 11
The Functional Building Blocks of AEM 11
Application-Level Functions 11
2 INSTALLATION 12
Exercise 16
2.1.1 Task - Start an AEM Author Instance 16
2.1.2 Task - Start an AEM Publish instance 18
2.1.3 (Optional) Task - Start AEM using the command line 20
3 AUTHORING BASICS 21
Exercise 25
3.1.1 Task – Create a page in AEM 25
3.1.2 Task – Edit a page in AEM 27
4 DEVELOPER TOOLS 31
Exercise 36
4.1.1 Task - Install a package in AEM 36
4.1.2 Task - Create, Build, and Download a Package 38
5 INTRODUCTION TO CONTENT RENDERING 40
Exercise - I 43
5.1.1 Task – Create the project structure in /apps 43

Exercise – II 46
5.1.2 Task – Create a page-rendering component 46
5.1.3 Task – Create content to be rendered 48
Exercise – III 54
5.1.4 Task – Basic Sling resource resolution and hunt for a rendering script 54
5.1.5 Task – Selector Manipulation 56
6 TEMPLATES 57
Exercise - I 58
6.1.1 Task – Create a Template 58
6.1.2 Task – Test the contentpage template 60
6.1.3 Task – Restrict Template Use 62
6.1.4 Task – Add Content Structure to the Template 63
Exercise – II 69
6.1.5 Task – Create the pages for the site 69
7 INTRODUCTION TO HTL 71
Exercise - I 75
7.1.1 Task – Render Basic Page Content 75
7.1.2 Task – Modularize the contentpage Component 77
8 INHERITANCE 81
Exercise 82
8.1.1 Task – Investigate the contentpage sling:resourceSuperType property 82
8.1.2 Task – Investigate the Foundation Page Component 83
8.1.3 Task – Delete the contentpage.html script 85
8.1.4 (Optional) Extra Credit Task – Build the rendering chain 86
9 DESIGN AND STYLES 87
Exercise 88
9.1.1 Task - Define a design and use clientlibs 88
9.1.2 Task – Modify the contentpage component to call in the design 90
10 DEVELOPING STRUCTURE COMPONENTS 93
Exercise - I 95
10.1.1 Task – Create a simple Top Navigation component 95
10.1.2 Task – Make the Navigation Component responsive 99
10.1.3 Task – Create a complex Navigation component with a Java Helper 101
10.1.4 Extra Credit task 104

10.1.5 Task – Create a complex Navigation component with a JavaScript Helper 105
Exercise - II 110
10.1.6 Task – Create a custom Log File 110
10.1.7 Task – Use log statements in the topnav Component 112
10.1.8 Extra Credit task – Add logging to the Java helper 114
Exercise - III 117
10.1.9 Task – Create a Title Component 117
10.1.10 Task – Create a dialog box for the Title Component 120
10.1.11 Task – Use the Title Dialog 122
Exercise - IV 125
10.1.12 Task – Create editConfig for the Title Component 125
Exercise - V 127
10.1.13 Task – Create an initial Hero Image Component 127
10.1.14 Task – Create a Dialog for Author Input 130
10.1.15 Task – Modify the text feature of the Hero Component 131
10.1.16 Task – Test the Hero Component 132
10.1.17 Task – Modify the Hero Component to display an Image 134
10.1.18 Extra Credit Task – Use the DAM to configure images in the Hero component 139
10.1.19 Task – Assign a Design to We.Train 144
10.1.20 Task – Create a Design Dialog for the Hero Component 146
11 THE RESPONSIVE GRID 152
Exercise 155
11.1.1 Task – Add the Responsive Grid Container 155
11.1.2 Task – Add Components to the Responsive Grid 157
11.1.3 Task – Enable the Responsive Emulator 160
11.1.4 Task – Use a Custom Emulator 162
11.1.5 Task – Integrate Layout Mode 164
11.1.6 Task - Authoring a Responsive Page with Layout Mode 167
12 ADVANCED SLING FUNCTIONALITY 171
Exercise I– Sling Selectors 171
12.1.1 Task – Investigate the Print Friendly button 171
12.1.2 Task – Create the print.html script 173
Exercise II– Sling Resource Merger 176
12.1.3 Task – Modify a Navigation Button 176

12.1.4 (Optional) Extra Credit Task – Hide Nav Button 178
12.1.5 (Optional) Extra Credit Task – Add a New Nav Button 179
Exercise III– Custom Error Handlers 180
12.1.6 Task – Create a custom 404 Error Handler 180
Exercise IV– Sling Redirect 182
12.1.7 Task – Make we-train.html redirect to another page 182
12.1.8 Task – Manage redirects from the Authoring interface 183
13 INTERNATIONALIZATION (GLOBALIZATION/LOCALIZATION) 184
Exercise 185
13.1.1 Task – Create the content to be localized 185
13.1.2 Task – Create the localization information 187
13.1.3 Task – Test the localized content 191
14 CONTENT COMPONENTS 192
Exercise I - Create a Content Component 193
14.1.1 Task – Create the initial StockPlex component 193
14.1.2 Task – Enable the StockPlex component 195
14.1.3 Task – Add a placeholder 197
14.1.4 Task – Build out the dialog box 199
Exercise II – Client Libraries 207
14.1.5 Task – Add the client libraries 207
14.1.6 Task – Determine whether your client library is defined in AEM 211
14.1.7 Task – Determine which client libraries are loaded with the page 212
14.1.8 Task – Extend the Dialog box to provide the author with more options 214
14.1.9 Task – Add a Design Dialog box 221
Exercise III (Optional) - Integrate Stockplex component with Live Data 223
14.1.10 Task – Upload and Install the Stockplex Backend Package 223
14.1.11 Task – Permission the Service User 225
14.1.12 Task – Modify stockplex.js to use imported data 227
Exercise IV – Search Component 228
14.1.13 Task – Create a search component 228
Exercise V – (Optional) Using JQuery and Ajax with an AEM component 236
14.1.14 Task – Create a User Grid component 236
14.1.15 Task – Create a placeholder dialog 238
14.1.16 Task – Define the User Grid client library 240

14.1.17 Task – Add the Ajax callback 243
15 DEBUGGING AND TESTING 247
Exercise 250
15.1.1 Task – Monitor component-based timing 250
15.1.2 Task – Create the Hobbes Testing Suite 252
15.1.3 Task – Run the Hobbes testing suite 254
16 USING BRACKETS FOR DEVELOPMENT 256
Exercise 258
16.1.1 Task - Install Brackets and the AEM Extension 258
16.1.2 Task – Package up your We.Train work in this course 260
16.1.3 Task – Make changes to the repository using Brackets 263
17 AEM ENVIRONMENT 266
Exercise 269
17.1.1 Task – Monitor Page Response 269
17.1.2 Task – Examine Page Request Performance 270
APPENDIX 276
Classic UI Dialog Boxes 276

1 ARCHITECTURE OVERVIEW

What is Adobe Experience Manager (AEM)?

Adobe Experience Manager is a content management system for building websites, content services, and forms. It is a web-based client-server system made up of a number of infrastructure-level and application-level functions. You can use these infrastructure and application-level building blocks to create customized solutions by building the required applications.

Infrastructure-Level Functions

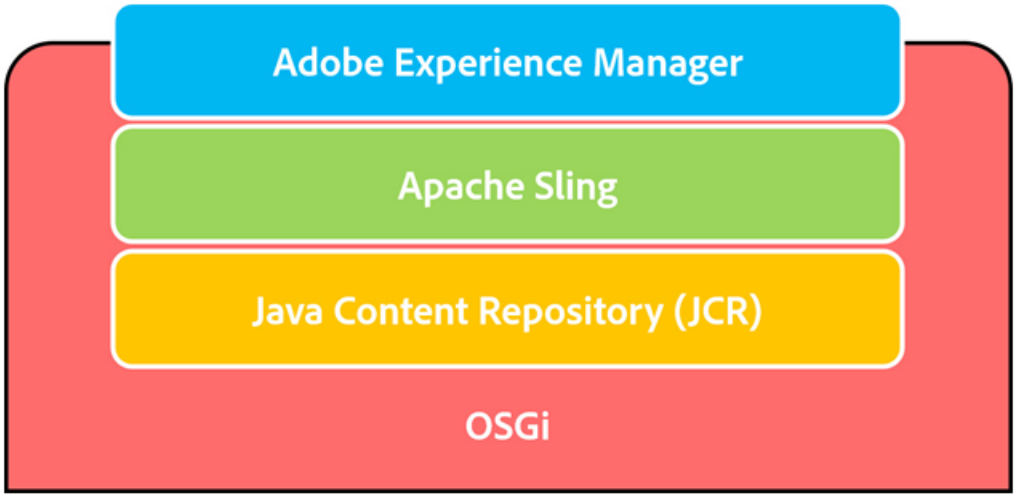
At the infrastructure-level, Adobe Experience Manager has:

- Web Application Server: You can deploy Adobe Experience Manager in standalone mode or as a web application within a third-party application server, such as WebLogic and WebSphere. The standalone mode includes an integated Jetty web server.
- Web Application Framework: Adobe Experience Manager has a Sling web application framework that helps simplify the writing of REST, content-oriented web applications.
- Content Repository: Adobe Experience Manager includes a Java Content Repository (JCR), a type of hierarchical database designed specifically for unstructured and semi-structured data.

Basics of the Architecture Stack

Adobe Experience Manager is a Java web application, and is based on technologies such as Open Service Gateway Initiative (OSGi), Java Content Repository (JCR), and Apache Sling.

The following diagram is a high-level view of the architecture stack.



Introduction to the Granite Platform

Granite is a general purpose platform for building robust scalable applications; it supports an "open architecture," which is based on both "open standards" (JCR and OSGi) and "open source" projects (Apache Sling and Apache Jackrabbit). Granite is Adobe's open web stack and Adobe Experience Manager is built on the Granite platform. Granite is open development, but not open source.

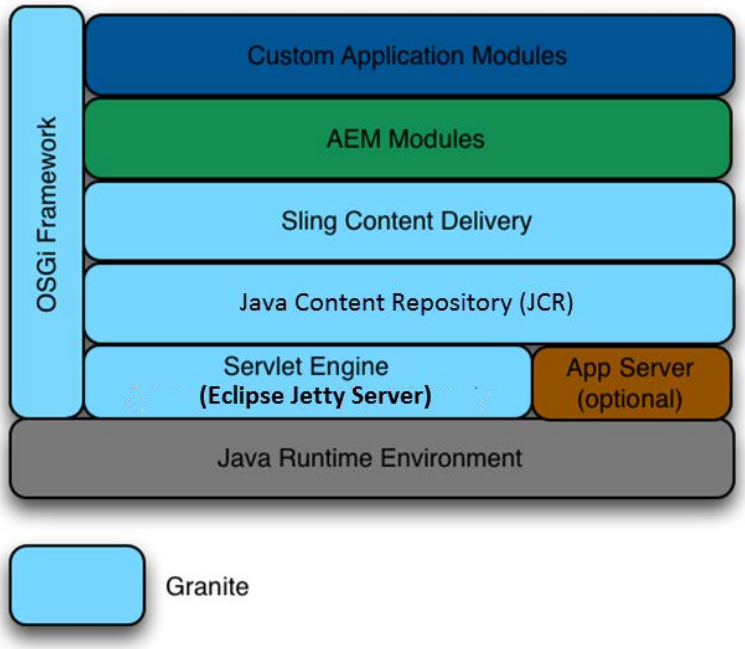
Technically, at the core, Granite provides:

- An application launcher: Quickstart for standalone Java Application or a Web Application Archive for deployment in existing Servlet Containers or Application Servers.
- An OSGi Framework into which everything is deployed.
- A number of OSGi Compendium Services to support building applications: Log Service, Http Service, Event Admin Service, Configuration Admin Service, Declarative Services, and Metatype Service.
- A comprehensive Logging Framework providing various logging APIs: SLF4J, Log4F, Apache Commons Logging, and OSGi Log Service.
- Repository based on Apache Jackrabbit Oak and JSR-283
- The Apache Sling Web Framework.

OSGi Framework

OSGi enables a collaborative and modular environment, where each application may be built and implemented as a small bundle. Each of these bundles is a collection of tightly coupled, dynamically loadable classes, JAR files, and configuration files that explicitly declare their external dependencies.

All content is stored in the content repository, which means backup is done at the repository level. OSGi runtime hosts Java applications that can access the repository using the JCR API. As part of the application runtime, you get Apache Sling, a RESTful web application framework that exposes the full repository content using HTTP and other protocols.



Apache Felix is an open source implementation of the OSGi for the Adobe Experience Manager framework. Apache Felix provides a dynamic runtime environment, where code and content bundles can be loaded, unloaded, and reconfigured at runtime.

Introduction to the Java Content Repository (JCR)

The JCR, specifically JSR-283 (Java Specification Request-283), is a database that supports structured and unstructured content, versioning, and observation. All data pertaining to Adobe Experience Manager such as HTML, CSS, JavaScript/Java, images, and videos are stored in the JCR object database. It is built with Apache Jackrabbit Oak, an open-source project. The Adobe implementation of JSR-283 used to be known as the Content Repository eXtreme (CRX). Hence, the naming of "CRX" you will see in some tools and interfaces in AEM. However, the CRX as a feature of AEM is being phased out. In its place is the Granite platform, and the use of Apache Jackrabbit Oak.

Adobe Experience Manager also works with other JCR repositories, such as Apache Jackrabbit 2.x, and with a number of non-JCR data stores through connectors. Because Adobe Experience Manager is built on top of this standard, it is capable of pulling content not just from its built-in Oak repository, but from any JCR-compliant source, such as a third-party repository (for example, Jackrabbit 2.x) or a connector that exposes legacy storage through JCR.

Advantages of using JCR:

- **JCR provides a generic application data store for structured and unstructured content. While file systems provide excellent storage for unstructured, hierarchical content, and databases provide storage for structured data, JCR provides the best of both data storage architectures.**

- **JCR supports namespaces. Namespaces prevent naming collisions among items and node types that come from different sources and application domains. JCR namespaces are defined with a prefix, delimited by a single colon (:). For example: jcr:title. This means that this title property is defined in the jcr namespace.**

Introduction to Apache Sling

Apache Sling is a web application framework for content-centric applications, and uses a Java Content Repository (such as Apache Jackrabbit Oak) to store and manage content. Apache Sling is based on Representational State Transfer (REST) principles, and helps build applications as a series of OSGi bundles.

Advantages of Apache Sling:

- Apache Open Source
- Sling is based on REST principles
- applications are built as a series of OSGi bundles
- is resource-oriented (every resource has a URI), and maps to JCR nodes

The embedded Apache Felix OSGi framework and console provides a dynamic runtime environment, where code and content bundles can be loaded, unloaded, and reconfigured at runtime. Sling makes it easy to implement simple applications, while providing an enterprise-level framework for more complex applications.

Sling assumes "Everything is a Resource". That is, Sling is resource-oriented, where the resources have URLs, and each resource is mapped into a JCR node. A request URL is first resolved to a resource, and then based on the resource, it selects the Servlet or script to handle that request. Servlets and scripts are handled uniformly in that they are represented as resources themselves and are accessible by a resource path. This means every script, Servlet, filter, error handler, and so on is available from the ResourceResolver just like normal content—providing data to be rendered on request.

The Functional Building Blocks of AEM

The Adobe Experience Manager application modules such as Sites, Assets, Communities, Forms, and Apps, sit on top of the Adobe Experience Manager shared framework. This shared framework is known as Granite, and includes all the application layer functionality such as mobile functionality, multisite manager, taxonomy management, and workflow. These functionalities are shared among all the application modules. In addition to these functionalities, these Adobe Experience Manager applications share the same infrastructure and UI framework, and are very tightly integrated with each other. All of this is encapsulated within the OSGi container.

Application-Level Functions

At the application-level, Adobe Experience Manager has the following functions to manage:

- Websites
- Content Services
- Forms
- Digital Assets
- Communities
- Online Commerce

2 INSTALLATION

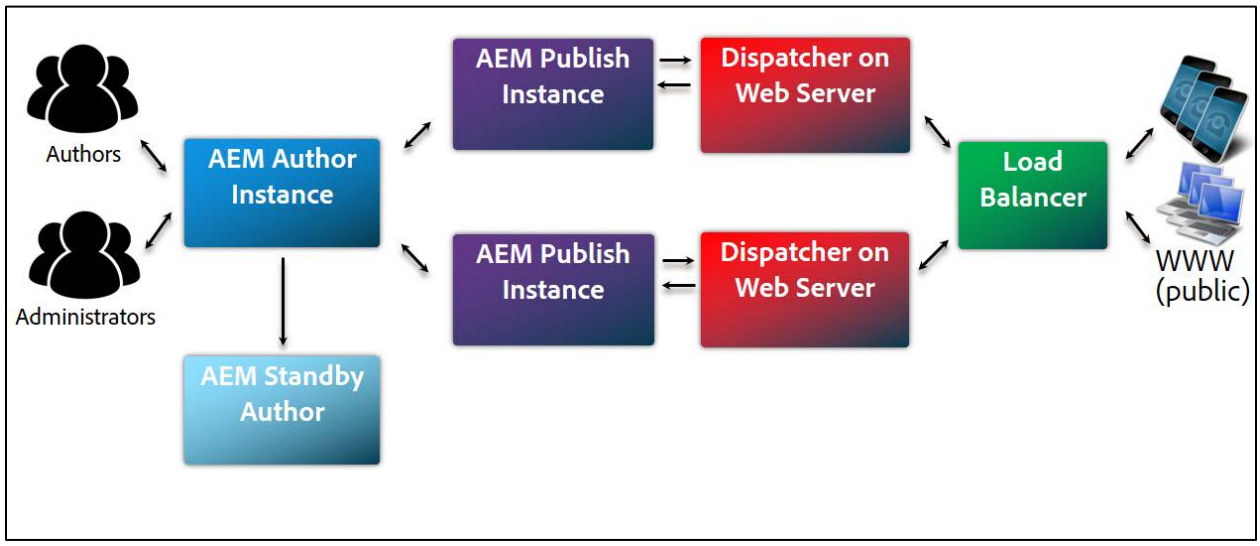
Adobe Experience Manager Instances

Adobe Experience Manager runs on most operating systems that support the Java platform. All client interactions with Adobe Experience Manager are done through a web browser.

In Adobe Experience Manager terminology, an "instance" is a copy of Adobe Experience Manager running on a server. Adobe Experience Manager installations usually involve at least two instances running on separate computers and a dispatcher:

- Author: An Adobe Experience Manager instance used to create, upload, and edit content, and administer the website. After content is ready to go live, it is replicated to the Publish Instance.
- Publish: An Adobe Experience Manager instance that serves the published content to the public.
- Dispatcher: A static web server (Apache httpd, Microsoft IIS, and so on) augmented with the Adobe Experience Manager Dispatcher module. It caches webpages produced by the Publish instance to improve performance.

NOTE: The author and publish instances are the same software stack but two different run modes.



Installation Prerequisites

To install Adobe Experience Manager, you need:

- Adobe Experience Manager installation and startup JAR file
- A valid Adobe Experience Manager license key properties file
- JDK version 1.8
- Approximately 4 GB of free space per instance
- Approximately 4 GB of RAM (at the **very minimum!)**

The Adobe Experience Manager installation and startup JAR file is also known as the "quickstart" file. You use the file to install Adobe Experience Manager. Once installed, the file is referred to as the Adobe Experience Manager startup file. During installation, you will notice the JAR file creates a root folder called **crx-quickstart**.

You also need to set environment variables as a part of your JDK 1.8 setup.

NOTE: You can download the latest JDK version from the following link:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Installing Adobe Experience Manager on Your System

In general, when you want to install Adobe Experience Manager on your system, you would follow this procedure:

Create two specific folder structures for your Adobe Experience Manager instances:

1. **Author instance**
   For Windows: C:/adobe/AEM/author
   For Mac OS or *x: /opt/adobe/AEM/author OR /Applications/AEM/author
2. **Publish instance**
   For Windows: C:/adobe/AEM/publish
   For Mac OS or *x: /opt/adobe/AEM/publish OR /Applications/AEM/publish

Add the **AEM Quickstart JAR** file along with the license.properties file to each folder which you created earlier.

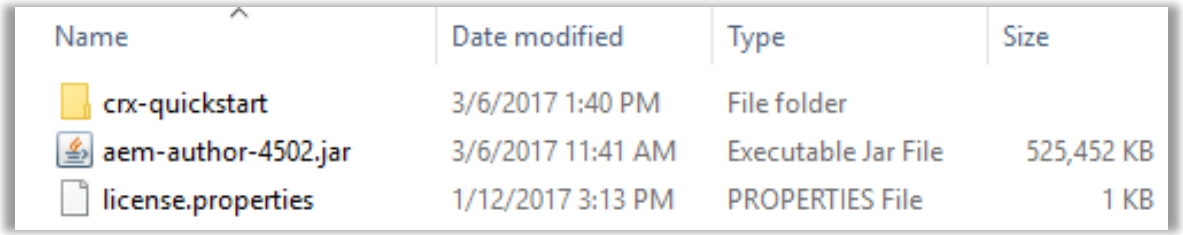
Rename the jar file to include the run mode as well as the port number. That is, rename the file to the format: **aem-<run mode>-<port number>.jar.**

For example,
Author instance: `aem-author-4502`
Publish instance: `aem-publish-4503`

You can therefore control the way Adobe Experience Manager is installed by defining properties via file name.

The first time you double-click the jar file, Adobe Experience Manager will install on your system, creating a root folder called `crx-quickstart`, which serves as your repository.

A sample folder structure for an Author instance is shown below.



NOTE: The Adobe Experience Manager quickstart file is renamed for installation purposes. When running for the first time, the quickstart file will notice that it has to install Adobe Experience Manager. By renaming the file, you use a convention of passing the instance name (Webpathcontext) and port number through the file name so no user interaction is needed during the installation process. If no port number is provided in the file name, Adobe Experience Manager will select the first available port from the following list in this specific order: 1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, or a random port.

NOTE: If you have multiple author and multiple publish instances, a best practice to consider is using an even/odd numbering paradigm for port numbers. So, you author instances would be 4502, 4504, 4506, and so on. Your publish instances would be 4503, 4505, 4507, and so on.

Starting an Adobe Experience Manager Instance

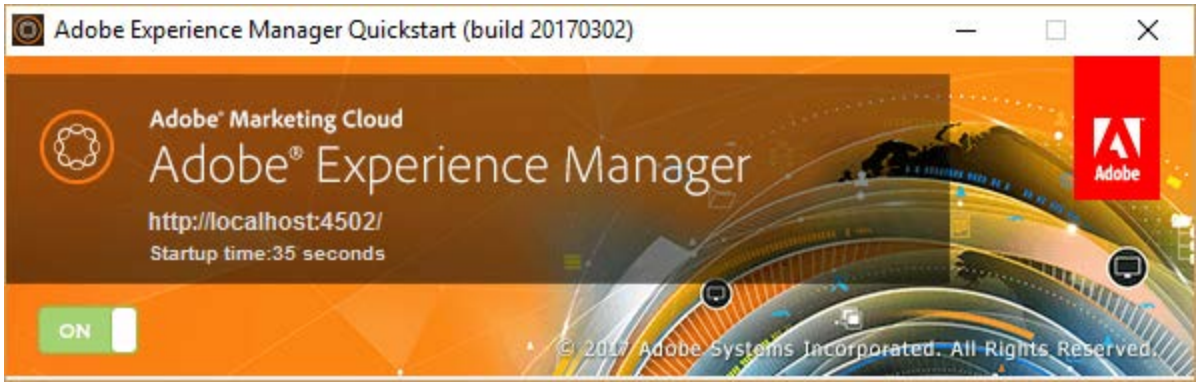
There are many ways of starting an Adobe Experience Manager instance, two of which are—graphical and by command line. The latter is more powerful because you have the possibility of providing additional performance-tuning parameters to the Java Virtual Machine (JVM).

Using the *.jar file to Start an Adobe Experience Manager Instance

In a Windows or Mac OS environment, you can double-click the aem-author-4502.jar file to start an Author instance (or the aem-publish-4503.jar file for a Publish instance).

Installation will take approximately 5-7 minutes, depending on your system's capabilities.

A dialog window will pop up similar to the following (this is known as the GUI):



After Adobe Experience Manager starts, your default browser will open automatically, pointing to Adobe Experience Manager's start URL (where the port number is the one you defined on installation).

Using the Command Line to Start an Adobe Experience Manager Author Instance

Prior to the installation, you may want to know which parameters are available to configure quickstart. Enter the following command to display a complete list of optional parameters:

```
java -jar aem-author-4502.jar -h
```

The Adobe Experience Manager quickstart installer will show all available command-line options without starting the server. In addition, you need to tune the JVM used for running Adobe Experience Manager. Tuning the JVM is an important and delicate task and requires a more realistic environment in terms of resources (hardware, operating system, and so on) and workload (content, requests, and so on). For now, it will be enough to know that you can start your instance (Author or Publish) using the following parameters:

-Xms --> assigns the initial heap size

Default value	64 MB for a JVM running on 32-bit machines, or 83 MB for 64-bit machines
Recommended	Specific to physical memory available and expected traffic
Syntax	-Xms512m (sets the initial heap size to 512 MB)

-Xmx --> assigns the maximum size the heap can grow

Default value	64 MB for a JVM running on 32-bit machines, or 83 MB for 64-bit machines
Recommended	Specific to physical memory available and expected traffic, but should be equal or greater than the initial size. To run Adobe Experience Manager, it is recommended to allocate at least 1024 MB of heap size.
Syntax	-Xmx1024m (sets the maximum size for the heap. In the example, we are letting it grow to 1024 MB; however, in production, this should be higher because Adobe Experience Manager consumes a lot of resources).

You can now install and start Adobe Experience Manager from the command line together with increasing the Java heap size, which will improve performance.

Using the Command Line to Start Adobe Experience Manager Publish Instance

If you wanted to start AEM using a command prompt, navigate to the directory containing your quickstart jar file (such as /adobe/AEM/publish), and enter the following command to install the publish instance:

```
java -jar aem-publish-4503.jar
```

Exercise

2.1.1 Task - Start an AEM Author Instance

NOTE: If you are attending a VILT class using ReadyTech, steps 1 through 3 were completed for you.

1. Create a folder structure on your file system where you will store, install, and start your Adobe Experience Manager author instance. For example:

   Windows: C:/adobe/AEM/author

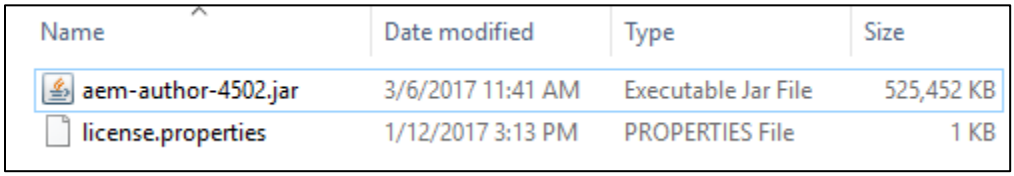
   MacOS X: /Applications/adobe/AEM/author or *x: /opt/adobe/AEM/author

2. Copy the `aem-quickstart-6.3.0.jar` and `license.properties` files from the Exercise_Files zip to your newly created directory.
3. Rename the `aem-quickstart-6.3.0.jar` file to `aem-author-4502.jar`:

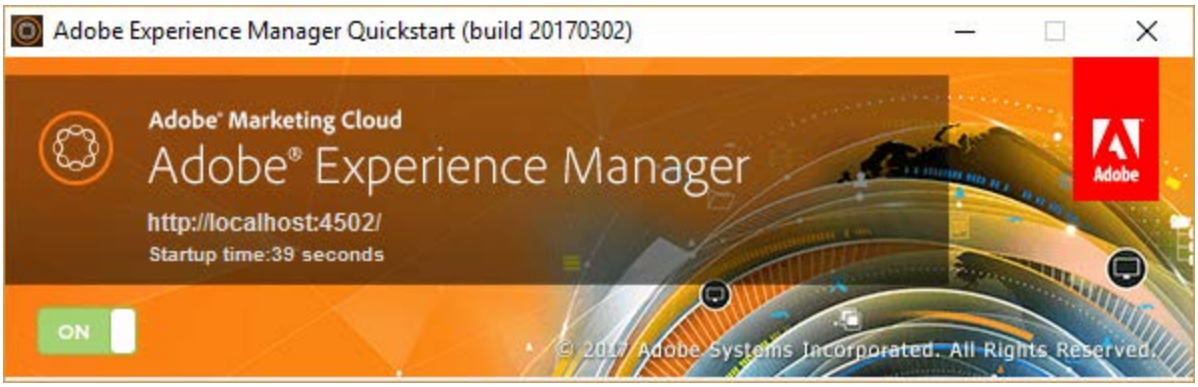
   aem = Application

   author = Web Content Management (WCM) mode it will run in (in this case, Author)

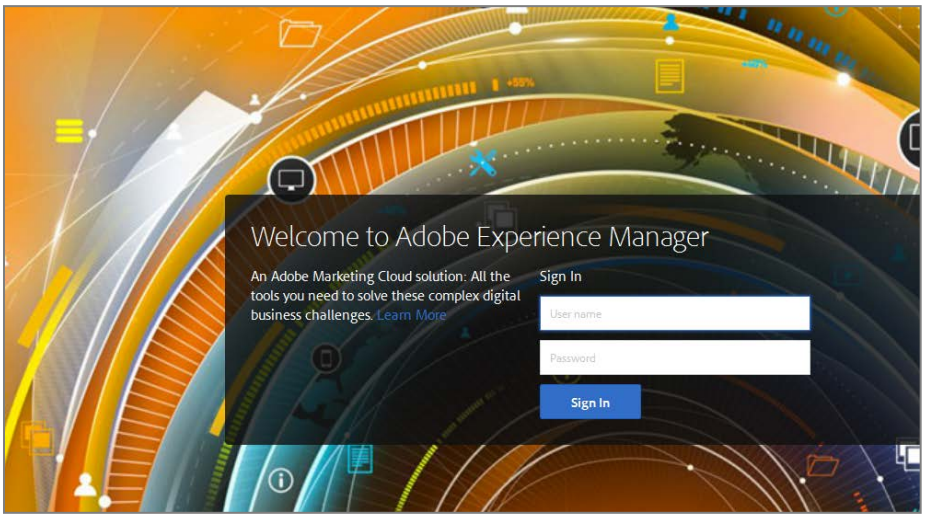
   4502 = Port it will run in



4. In a Windows or MacOS X environment, double-click the `aem-author-4502.jar` file. Installation will take approximately 5–7 minutes depending on your system's capabilities.
5. After Adobe Experience Manager Author instance has started successfully, the start-up screen (the GUI) will change to something similar to the following:



6. In addition, after Adobe Experience Manager starts, your default browser will automatically open to Adobe Experience Manager's start URL (where the port number is the one you defined on installation); for example: http://localhost:4502. A Sign In screen will be displayed:



NOTE: A crx-quickstart directory is also created on your machine:

Name	Date modified	Type	Size
crx-quickstart	3/6/2017 1:40 PM	File folder	
aem-author-4502.jar	3/6/2017 11:41 AM	Executable Jar File	525,452 KB
license.properties	1/12/2017 3:13 PM	PROPERTIES File	1 KB

2.1.2 Task - Start an AEM Publish instance

NOTE: If you are attending a VILT class using ReadyTech, steps 1 through 3 were completed for you.

1. Create a folder structure on your file system where you will store, install, and start your Adobe Experience Manager publish instance. For example:

Windows: C:/adobe/AEM/publish

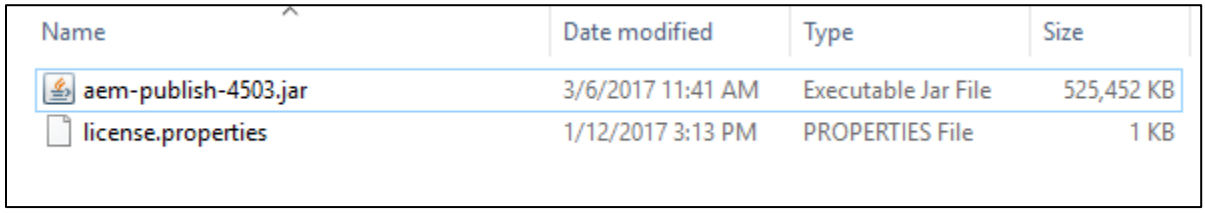
MacOS X: /Applications/adobe/AEM/author or *x: /opt/adobe/AEM/publish

2. Copy the `aem-quickstart-6.3.0.jar` and `license.properties` files from the Exercise_Files zip to your newly created directory.
3. Rename the `aem-quickstart-6.3.0.jar` file to `aem-publish-4503.jar`:

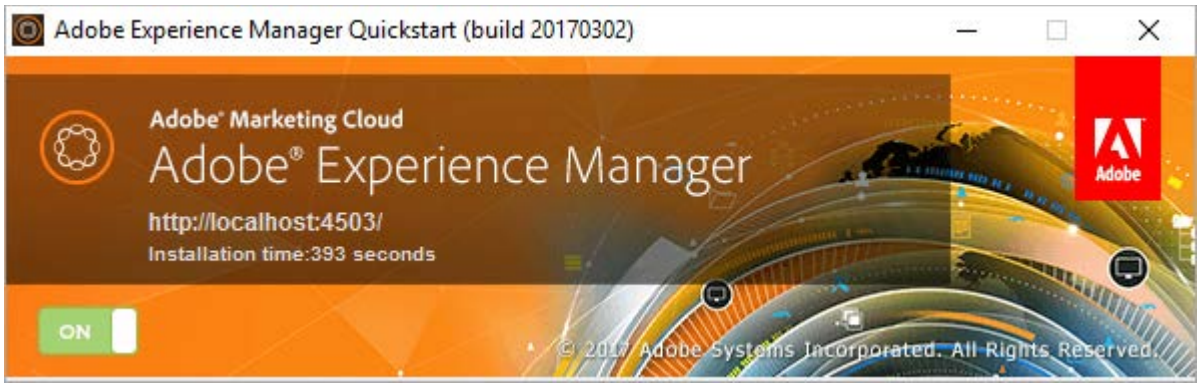
   aem = Application

   publish = Web Content Management (WCM) mode it will run in (in this case, Author)

   4503 = Port it will run in



4. **In a Windows or MacOS X environment, double-click the `aem-publish-4503.jar` file. Installation will take approximately 5–7 minutes depending on your system's capabilities.**
5. After Adobe Experience Manager Publish instance has started successfully, the start-up screen will change to something similar to the following:



6. In addition, the Adobe Experience Manager login page opens from your default browser (where the port number is the one you defined on installation); for example, http://localhost:4503.

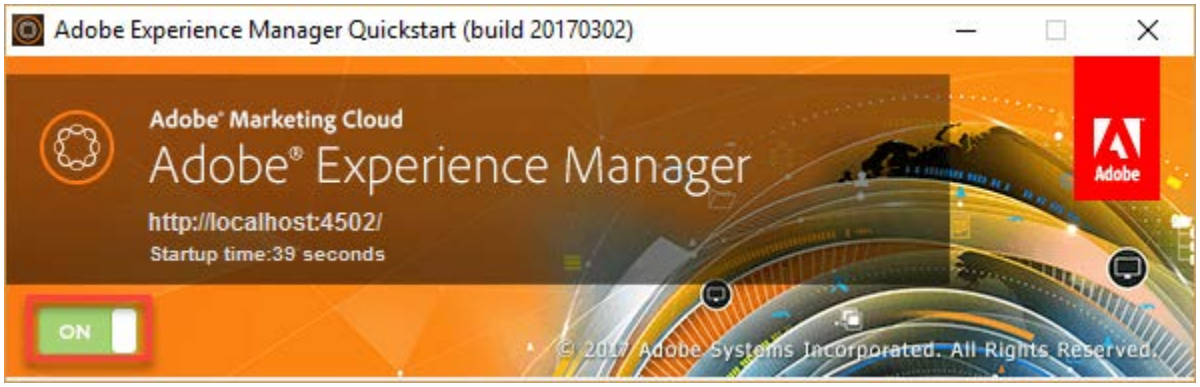
7. The following screen appears once the Publish instance is up and running:

NOTE: There is no need to sign in. The publish instance loads the We.Retail reference site immediately.

NOTE: We.Retail is a reference implementation that illustrates the recommended way of setting up an online presence with Adobe Experience Manager. While We.Retail illustrates a retail vertical, the way the site is set up can be applied to any vertical. Only the product catalog and cart features are retail-specific.

You have now successfully installed and started Adobe Experience Manager Author and Publish instances on localhost.

To stop an Adobe Experience Manager instance, click the "on / off" toggle button in the GUI window:



To start Adobe Experience Manager in the future, double-click the renamed `aem-quickstart-6.3.0.jar` file; for example, `aem-author-4502.jar`.

2.1.3 (Optional) Task - Start AEM using the command line

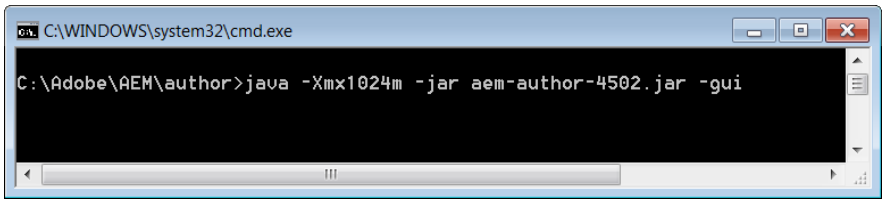
You already have an author instance and a publish instance running. Perform this task only when necessary or as an add-on exercise to try out beyond this class.

This is a powerful method because you can provide additional performance-tuning parameters to the Java Virtual Machine (JVM). On Windows, MacOS X, or *x, you can install or start Adobe Experience Manager from the command line, while increasing the Java heap size, which improves performance.

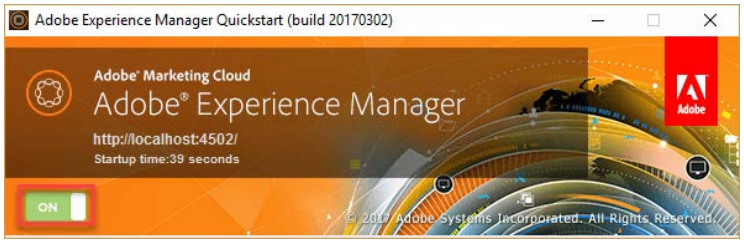
A typical command line to start AEM by setting the Java heap size will have the following:

```
java -Xmx1024m -jar aem-author-4502.jar -v
```

This example command below starts AEM author runmode with a specific memory allocation to the JVM and the GUI window "on":



1. Stop your author instance by clicking the **On** button in the GUI window:



2. In your command prompt, navigate to the Adobe\AEM\author directory (or the directory where your author *.jar file is), and use the following command to start Adobe Experience Manager the very first time without installing the We.Retail reference site:

```
java -jar aem-author-4502.jar -r author, nosamplecontent -gui
```

NOTE: You would run with "nosamplecontent" if you are performing a production installation, in which case the sample content is not needed. Also, note the "nosamplecontent" option is only available upon first starting the instance.

TIP: To open a directory in Windows Explorer in the command-line, select the directory, hold down the Shift key, and right-click. Then, you will see an option to open that directory in a command-line window.

Summary

You should now be able to:

- Install and run the Adobe Experience Manager Author instance
- Install and run the Adobe Experience Manager Publish instance

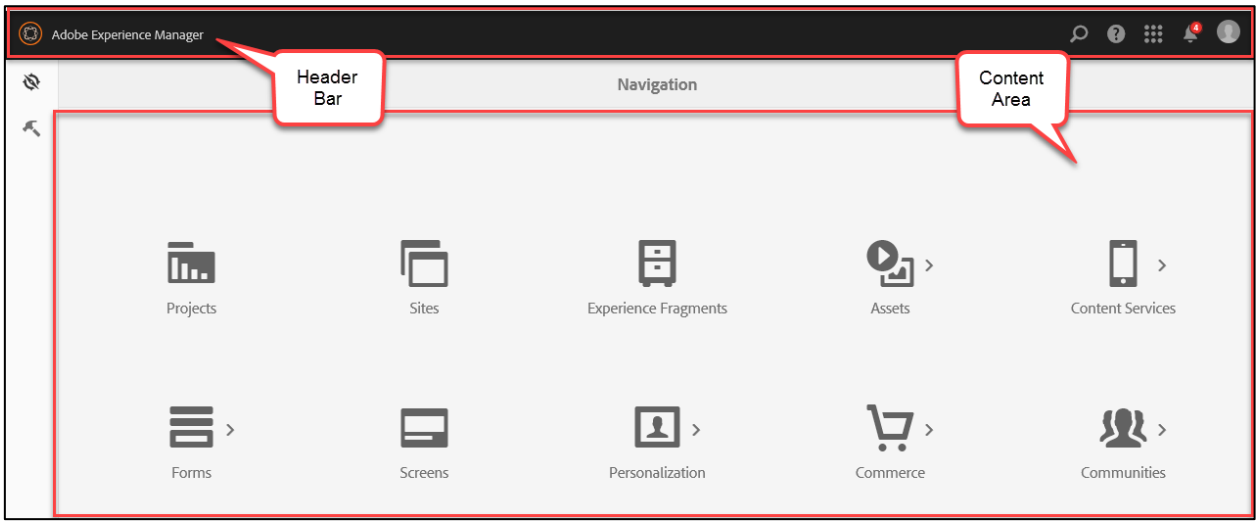
3 AUTHORING BASICS

Introduction to Touch UI

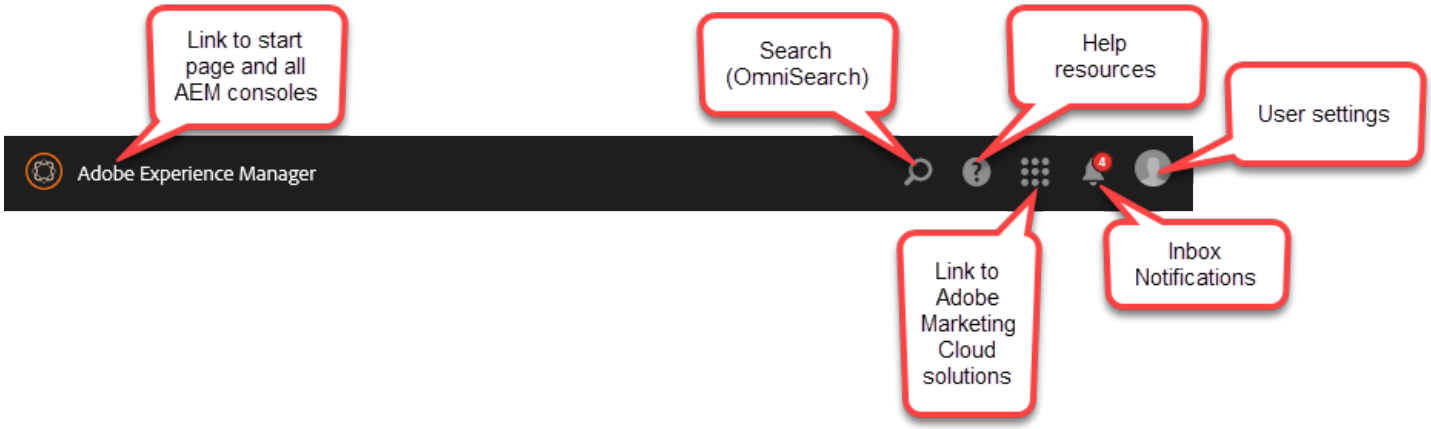
The Adobe Experience Manager User Interface (UI) combines the advantages of a web interface with the fluidity and responsiveness that is usually associated with desktop applications.

Touch UI Actions	Desktop UI Actions
Tap	Click
Touch-and-hold	Double-click
Swipe actions	Hover

Let's familiarize ourselves with the key areas of the Adobe Experience Manager Touch UI. When you load the author application, you are presented with a start screen that includes the header bar and the content area. All of the capabilities of AEM are available (Projects, Sites, Experience Fragments, Assets, and so on).

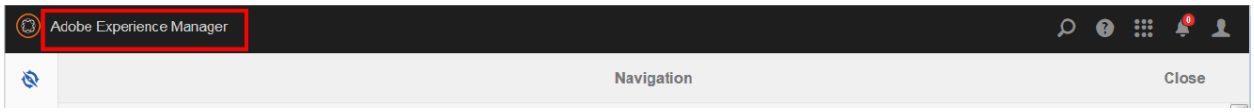


The Header bar has the following elements available globally:

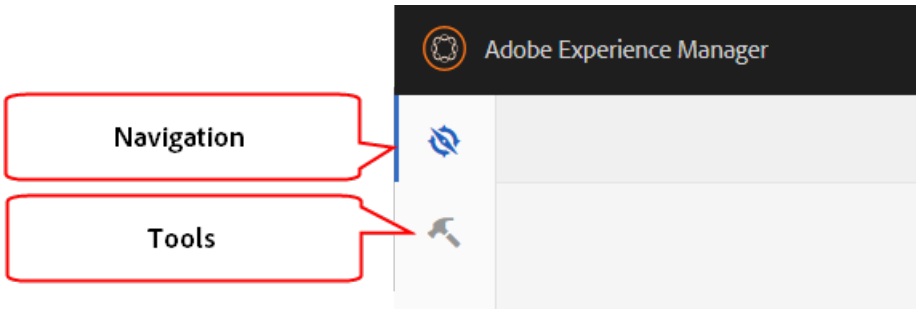


Navigating to Consoles and Tools

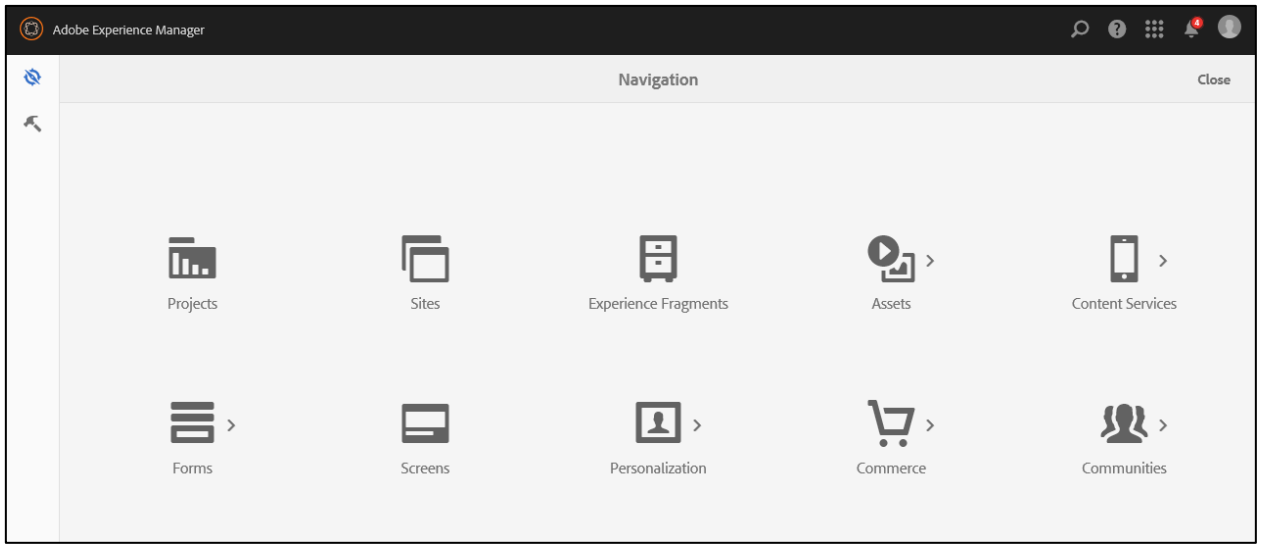
Two major consoles you need to access during this training are the start (consoles) and Tools. You can access these links by clicking Adobe Experience Manager in the upper-left corner of the screen.



You are presented with the following options:



Navigation—This option provides access to all AEM capabilities such as Projects, Sites, Assets, and so on.



Tools—This option provides access to a number of specialized tools and consoles that help you develop and administer your websites, digital assets, and other aspects of your content repository.

Creating and Editing Pages

A page in Adobe Experience Manager is similar to a web page, and contains components such as text, image, videos, and many more. These pages are created from templates that define the structure of the page—including where each component has to be placed.

Creating Pages

To organize a website within Adobe Experience Manager, you need to first create and name your content pages, which allows for easy accessibility.

Before you add content, you must create a page within a site. Two key fields are used while creating a page:

Title: This is displayed to the user in the console and shown at the top of the content page when editing. This field is mandatory.

Name: This is used to generate the URI. User input for this field is optional. If not specified, the name is derived from the title.

- Name of the page should be in lowercase, as it's a recommended practice for URLs (some web servers are case-sensitive while others are not).
- NOTE: Only the following characters are allowed in the Name field: 'a' through 'z', 'A' through 'Z', '0' through '9', '-'

We.Retail

Recall at the end of the last exercise, you explored the We.Retail reference site that comes with AEM. We.Retail completely replaces the Geometrixx reference sites in prior versions of AEM.The We.Retail site is built with the following best practices of Adobe Experience Manager:

Localized site structure, with language masters live-copied into country-specific sites

Content fragments

Responsive layout for all pages

Editable templates for all templates

Based on Maven Archetype 10

HTML Template Language (HTL) for all components

eCommerce capabilities with product catalog

Communities sites for site visitors

You can download the latest release of the We.Retail site using the following link:
https://github.com/Adobe-Marketing-Cloud/aem-sample-we-retail/releases

The We.Retail site is built using Maven Archetype 10. This is the latest archetype available from Adobe.

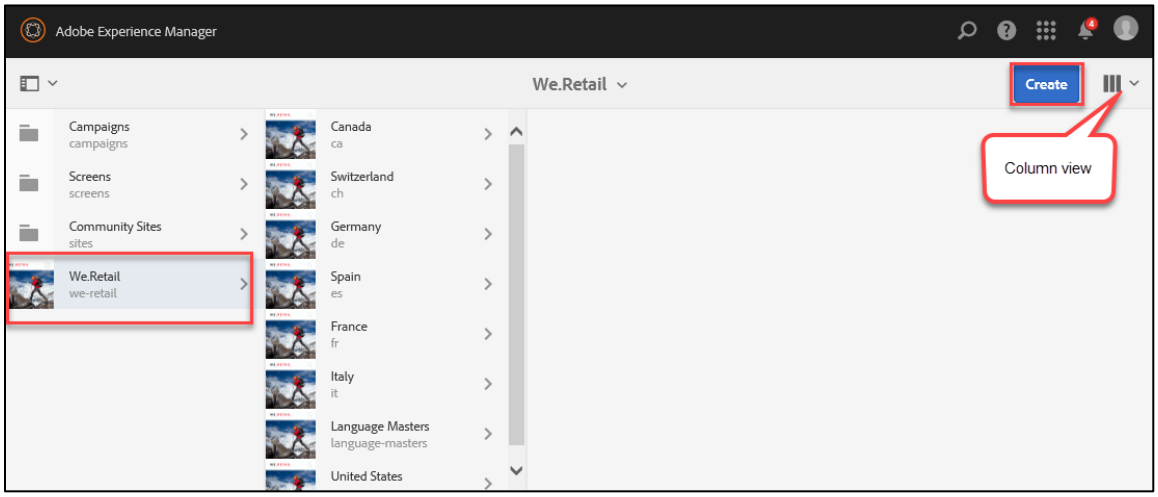
With this course, you will work with a site called We.Train that is based on the We.Retail site and thereby adheres to the best practices. We.Train is also created using Archetype 10.

Exercise

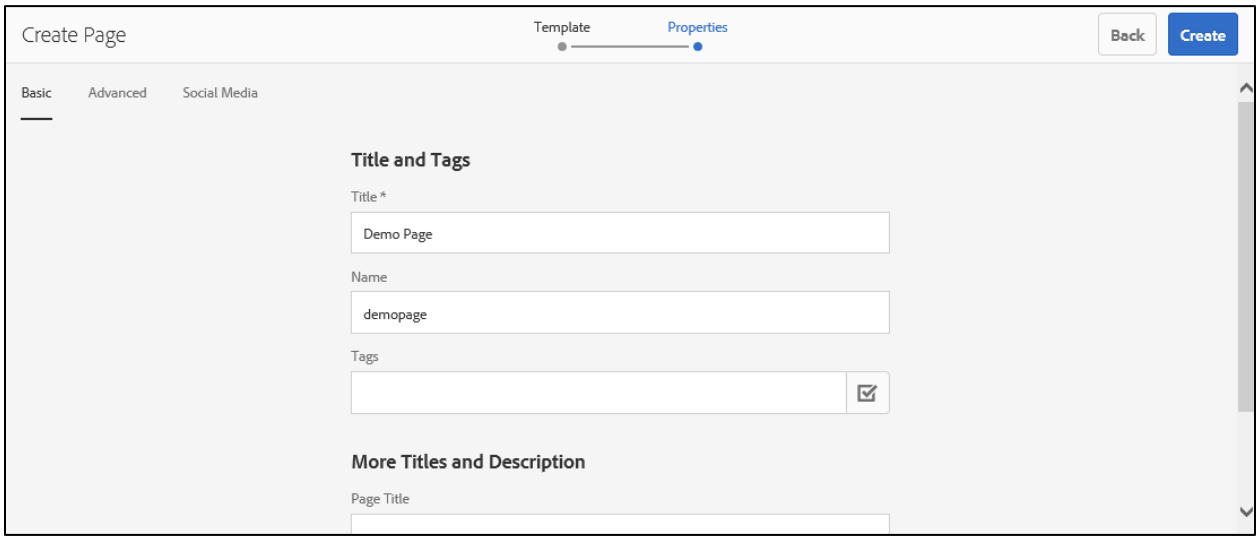
3.1.1 Task – Create a page in AEM

Steps to create a page:

1. Start your author instance of AEM. Refer to the exercise from the previous module for instructions.
2. Log in to Adobe Experience Manager, and navigate to the **Sites** console.
3. While in the Column view, select **We.Retail**, and click **Create** > **Page**.



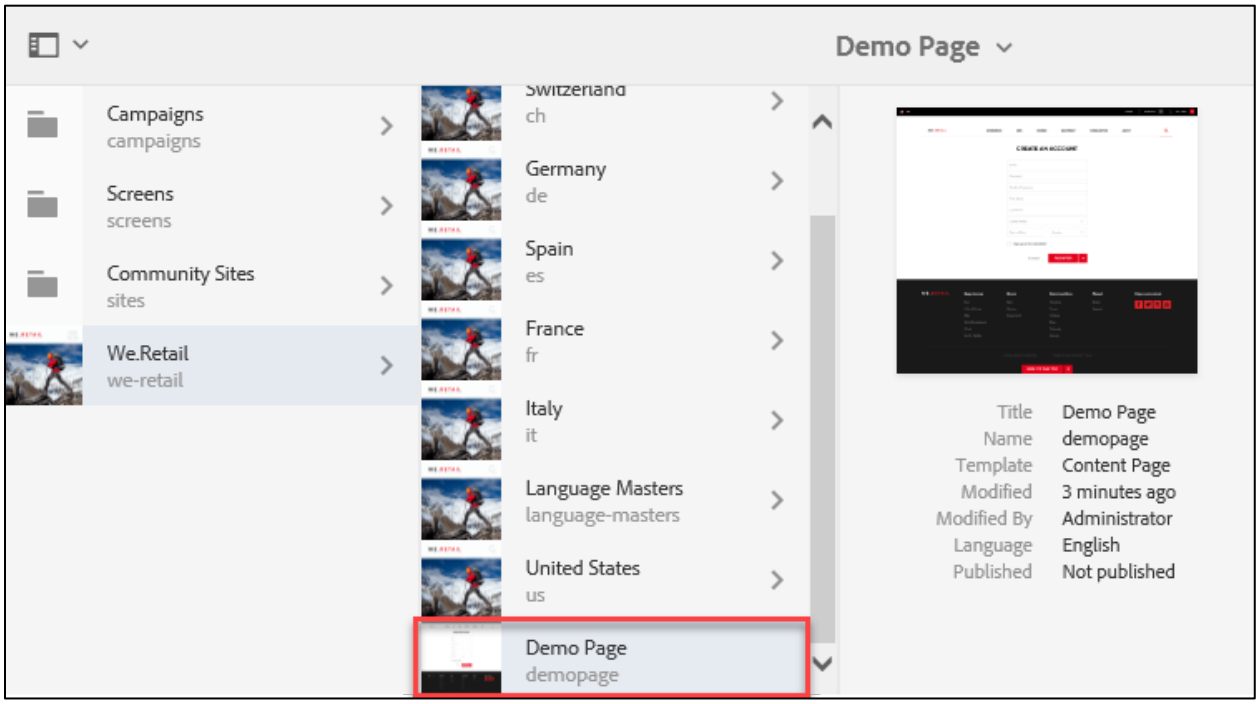
4. Click **Create** > **Page** from the actions bar.
5. In the Template wizard, select the **Content Page** template, and then click **Next** to proceed.
6. In the Properties wizard, enter these values for the following fields:
   - **Title**: Demo Page
   - **Name:** demopage



7. Click **Create** to complete the process.
8. Click **Done** from the Page Created dialog box.

The new page will appear as a child of the We.Retail page. Notice that the page name “Demo Page” appears in the column view as well as the page title (“demopage”) below it.

You can create subpages of any page using the same process.



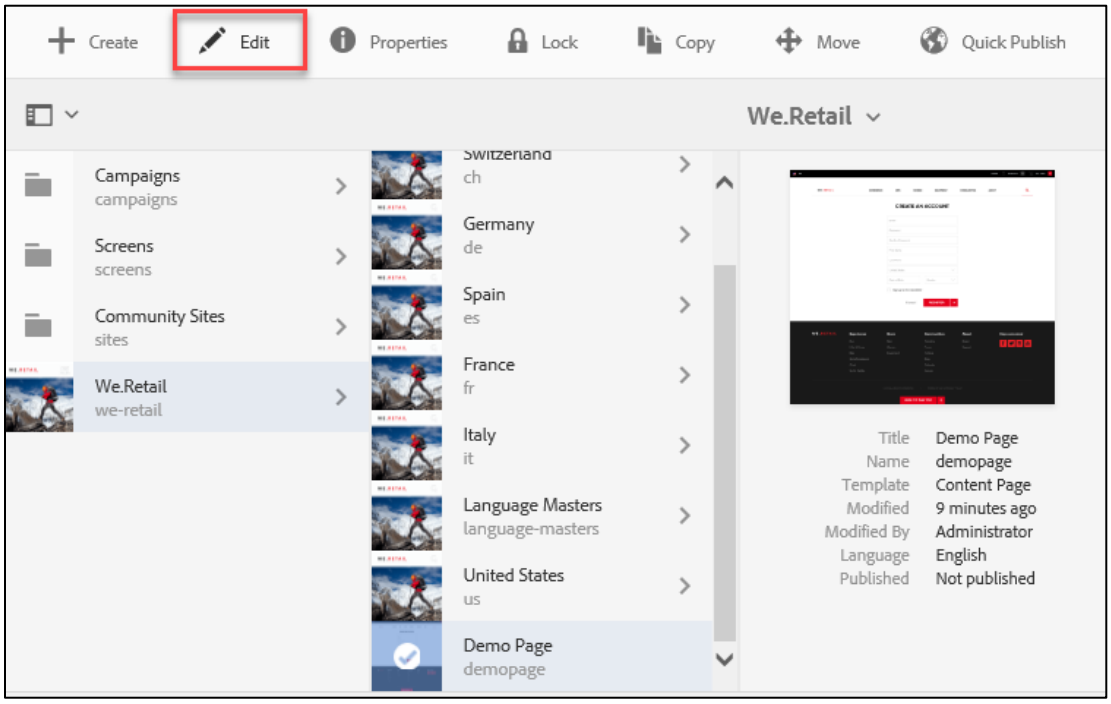
3.1.2 Task – Edit a page in AEM

Once the page is created, you can edit the content of the page. Content is added using components (content placeholders), which you can drag and drop onto the page. You can add, edit and delete components from the page.

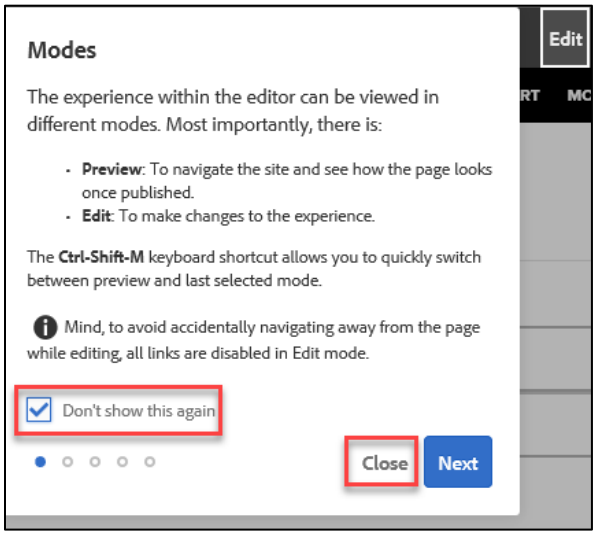
Let's edit the page and add content using components such as Text and Image on the page you just created.

Steps to add text to the page:

1. Select the page you just created by clicking it to use the selection mode and click **Edit** from the actions bar. Be patient as it may take a few minutes to load.

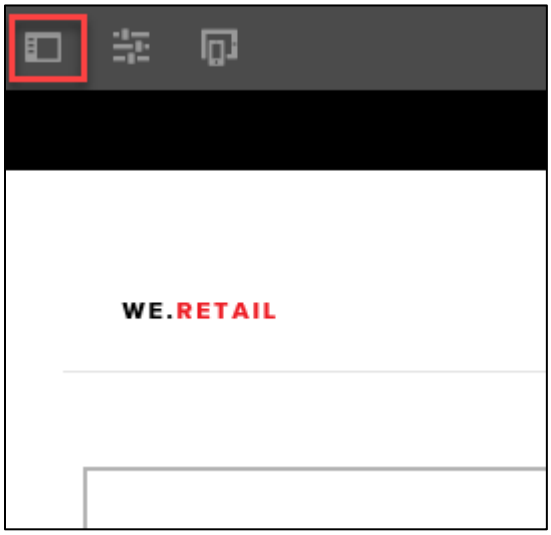


2. When you first open the page in edit mode, you will most likely see a dialog guiding you through the different modes. You may close this dialog and choose **Don't show this again**:

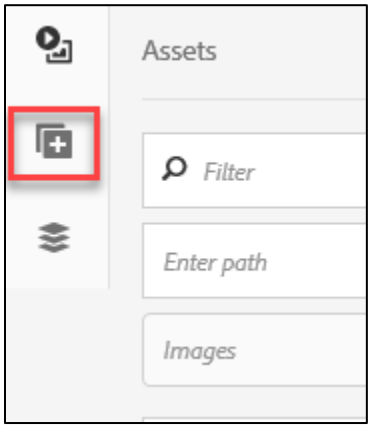


3. Make sure the page is opened in Edit mode by looking in the upper-right and seeing the **Edit** mode.

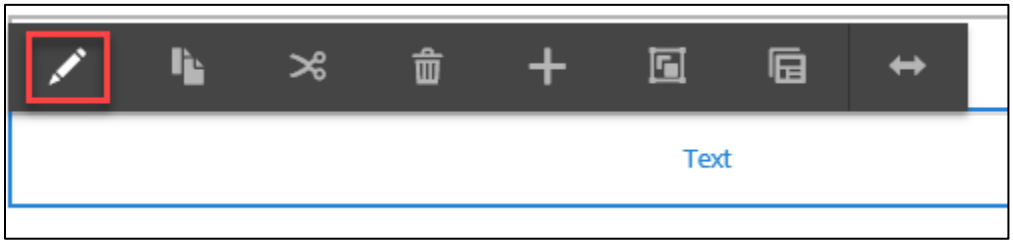
4. Next, add a Text component to the page. Click the **Toggle Side Panel** icon in the upper-left:



5. Click **Components** in the side panel:



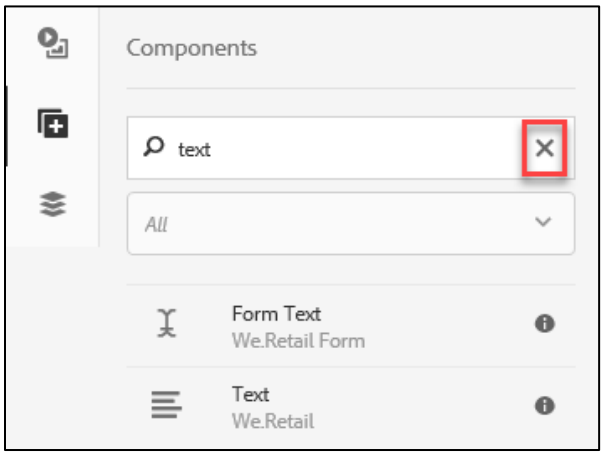
6. In the Filter field, enter text (to search for the Text component) and **press Enter**. The search yields two results, which all contain the word "text".
7. Drag and drop the Text component into the **Drag components here** area. The Drag components here area changes to the Text area.
8. Click the **Text** component, and then click **Edit** (pencil icon) from the component toolbar. A text editor opens in the same tab.



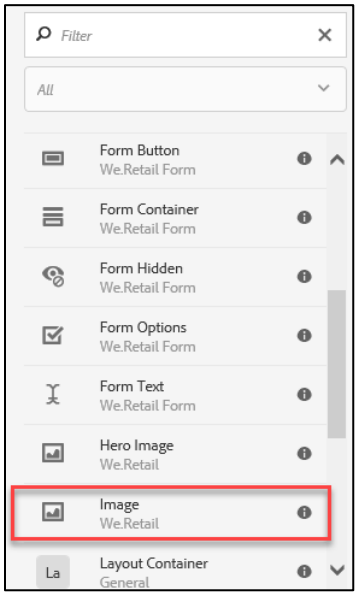
9. Add sample text and click **Done (checkmark)** to save the changes. The text is added to the page.

Steps to add an image to the page:

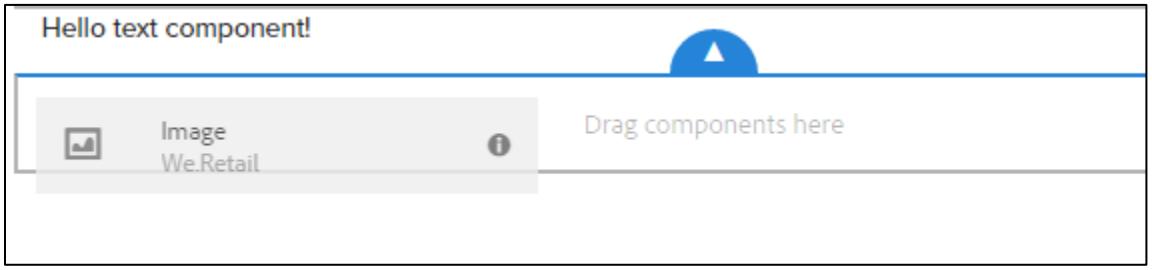
1. In the side panel, click **x** to clear the search.



2. Scroll down and select **Image** from the list of components:

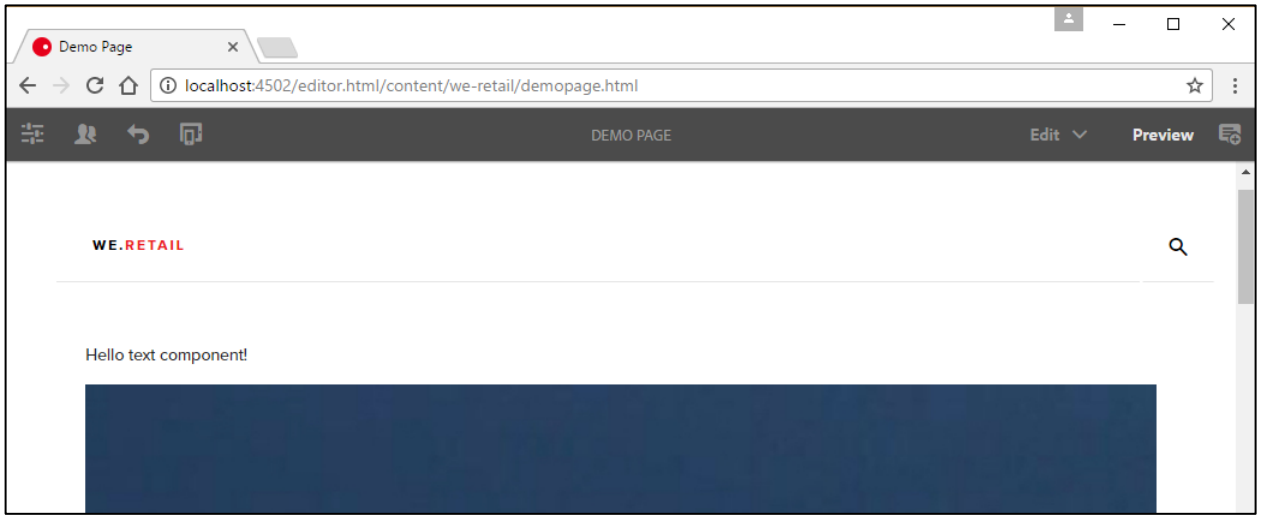


3. Drag and drop the **Image** component onto your page in the Drag components here area.



4. Click **Assets** in the side panel.

5. Choose any image from the Assets tab and drag and drop the image onto the **Image** component. Adobe Experience Manager will add the image to the page.
6. Click **Preview** mode at upper-right to view the changes to your page:

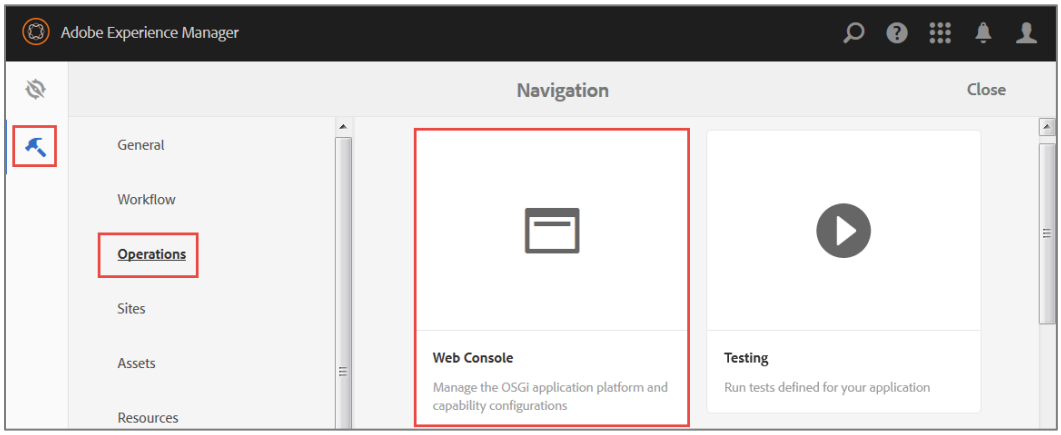


4 DEVELOPER TOOLS

The Web Console

The Web Console in Adobe Experience Manager is based on the Apache Felix Web Management Console. It is used to manage various bundles and configurations. Any changes made through this console are automatically applied to the running system, without the need to restart the instance.

You can access the console at: http://localhost:4502/system/console or by navigating to Tools > Operations > Web Console, as shown:



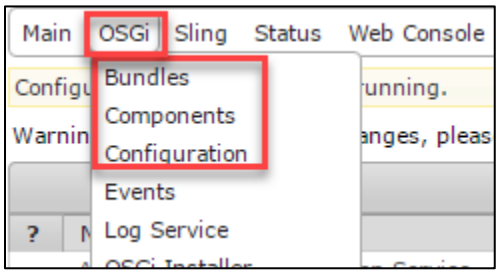
If you use the navigation in AEM, the Web Console opens in a new tab in your browser.

The most important menu selections under the OSGi tab are:

Bundles—used for installing and managing bundles.

Components—used for managing and controlling the status of components required for Adobe Experience Manager.

Configuration—used for configuring OSGi bundles, and is the underlying mechanism for configuring Adobe Experience Manager parameters.

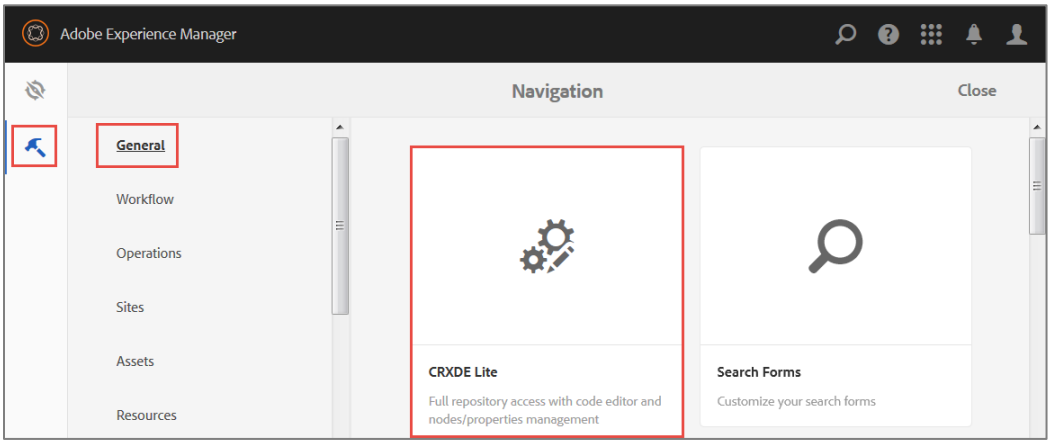


CRXDE Lite

The CRXDE Lite console is embedded into Adobe Experience Manager and allows you to perform common development and administration tasks in the browser. Because it is embedded in the server and is always available, CRXDE Lite is often the preferred tool for administrators and developers for working with nodes and properties in the JCR. It gives quick and direct access to the repository for monitoring, configuration, and development.

NOTE: You will use this interface frequently in this training.

You can access this console through http://localhost:4502/crx/de/index.jsp or by navigating to Tools > General > CRXDE Lite from the Touch UI, as shown:



With CRXDE Lite, you can create a project, create and edit files (like .jsp and .java), folders, templates, components, dialogs, nodes, properties, and bundles. CRXDE Lite is recommended for most repository-level administration tasks as well as many light weight development tasks.

The CRXDE Lite console contains:

Top switcher bar: Enables you to quickly switch between CRXDE Lite, Package Manager, and Package Share.

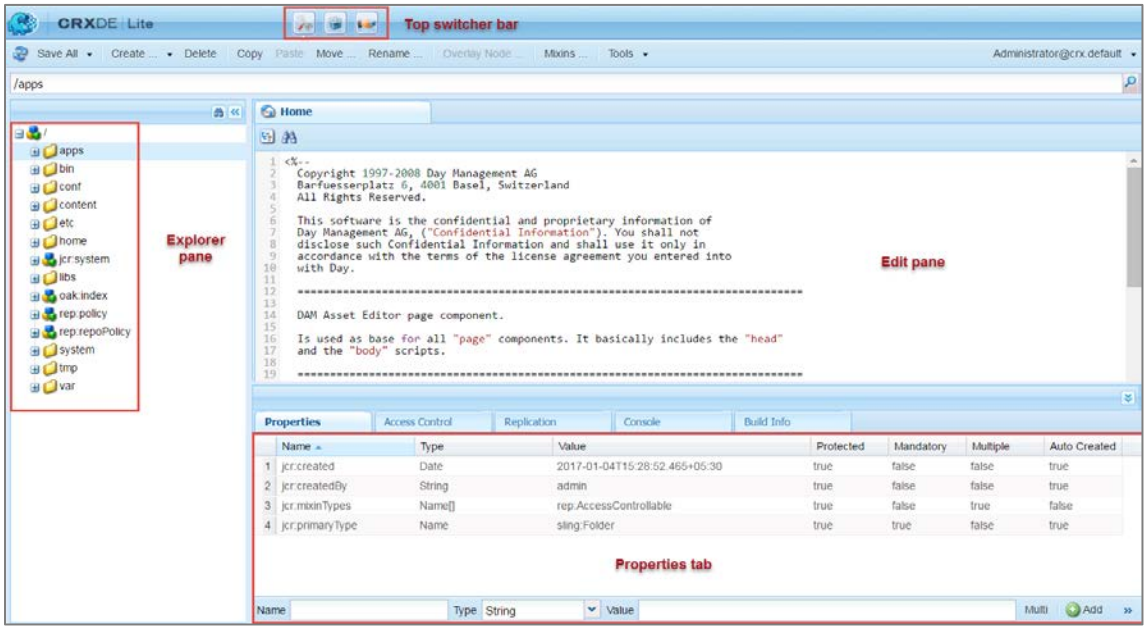
Explorer pane: Displays a tree of all the nodes in the repository.

You can perform the following actions on a node in the tree:

- Select the node and view its properties in the Properties tab. Examine all the JCR properties of different nodes.
- Right-click the node and perform an action on it, such as renaming the node, creating a new node, creating a folder, creating a file.

Edit pane: Double-click a file in the Explorer pane to display its content. For example, a .jsp or a .html file. You can then modify the code and save the changes.

Properties tab: Displays the properties of the node that you selected. You can add new properties or delete existing ones.



Working with Packages

A package is a zip file that holds repository content in the form of a file-system serialization called "vault" serialization. Packages provide an easy-to-use-and-edit representation of files such as pages, assets, and folders.

Throughout this training, you will be working with packages; either by creating your own or installing an available package. These packages enable you to import and export repository content. A package may contain:

- Page-related content
- Project-related content
- Assets-related content
- Vault meta information such as filter definitions and import configuration information
- Other package information such as package settings, package filters, package screenshots, and package icons

Packages enable you to import and export repository content.

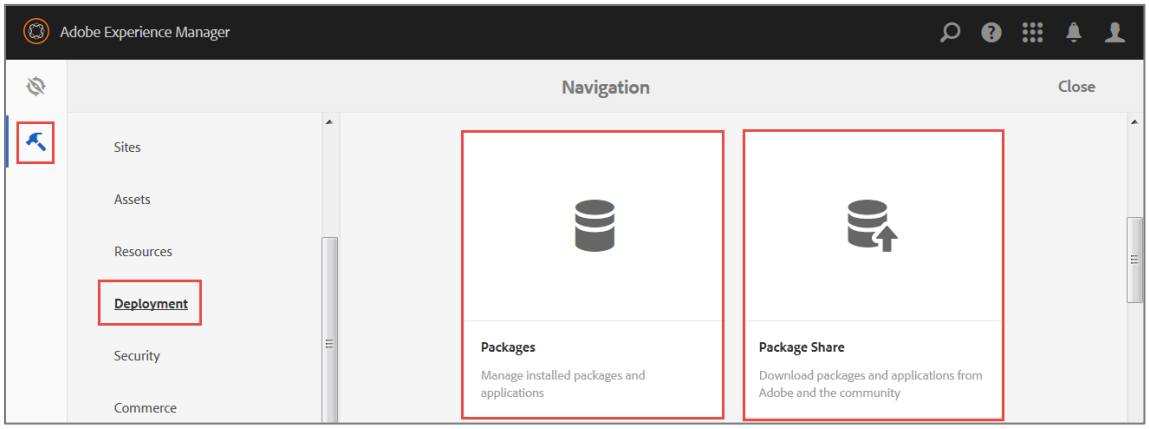
You commonly use packages for any of the following:

- Install new functionality
- Transfer content between instances
- Back-up repository content
- Export content to the local file system

You can work with packages in two ways:

1. **Package Manager**—can be used to import or export content, transfer content between instances, and back-up repository content. Using filters, you can create a package to contain page content, or project-related content. You can access this console through the following link: http://localhost:4502/crx/packmgr/index.jsp. With the Package Manager, you can perform the following common tasks:
    - Create, build, and download content packages
    - Upload and install packages
    - Modify existing packages
    - View package information
2. **Package Share**—a centralized server that contains both public and private packages available across all instances. Public packages may include hotfixes, new functionality, documentation, and so on. You can access this console through the following link: http://localhost:4502/crx/packageshare/index.html.

    To access Package Manager and Package Share from the Touch UI, navigate to **Tools** > **Deployment** > **Packages/ Package Share** as shown:



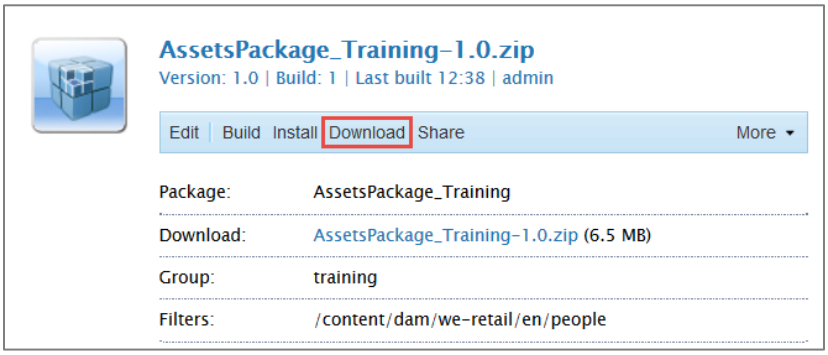
Creating and Building New Packages

When creating packages using Package Manager, you can apply rules and filters to determine the content package should extract from the repository. After you define the content, you can build it. A package is created in a .zip file and you can download it to your local file system. You can test the contents of the package before building it.
There are many options in Package Manager to work with packages, such as:

- Rebuild: Helps rebuild the package if there is a change in the repository content.
- Edit: Helps edit filters or rules applied to the package.
- Test: Helps perform a dry run of the installation.
- Rewrap: Helps recreate the package with additional information such as thumbnails and icons.

Downloading packages to your file system

You can download a package by clicking the download link. This link displays when the package details are expanded. After downloading the package, you can unzip the contents of the package onto your local system.



Typically, a content package contains the following folders:

- **jcr_root:** Represents the root node of the repository and contains the actual content of the package.
- **META-INF**: Contains metadata regarding node definitions and the filter.xml file that gives directions to FileVault about the paths to include.

Using the Package Share

Package Share is a centralized server where public and private packages are made available. These packages may be hotfixes, feature sets, updates, or Adobe Experience Manager content generated by other users. You can search, download, and install any package either to your instance or to your local file system.

Within the Package Share, you have access to the following:

- Adobe packages provided by Adobe
- Shared packages provided by others companies and made public by Adobe

  Your company packages that are private

  You can access the Package Share with this link: http://localhost:4502/crx/packageshare/login.html
- You can directly access Package Share (without using CRXDE Lite) through the following link: https://www.adobeaemcloud.com/content/packageshare.html

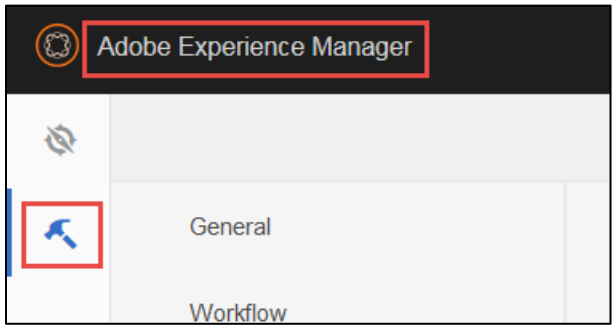
---

NOTE: You must have an Adobe ID to use the Package Share. For more information, refer to the online documentation about Package Share.

Exercise

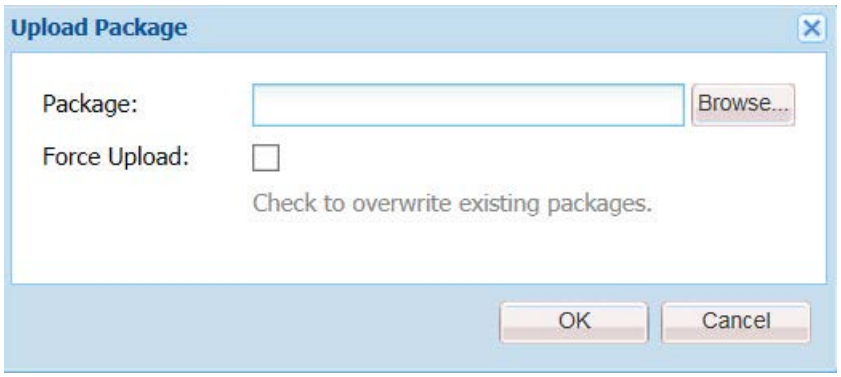
4.1.1 Task - Install a package in AEM

1. Navigate to the Projects console, or click the following link: **http://localhost:4502**.
2. Click Adobe Experience Manager in the upper-left, then click the **Tools** icon.

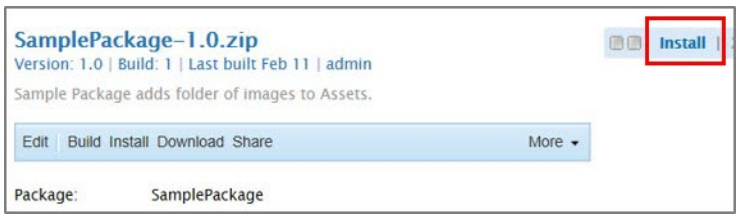


3. Under the Deployment section, select **Packages**. This opens the Package Manager of CRXDE Lite. As an alternative, click the following link to open the page directly: http://localhost:4502/crx/packmgr/index.jsp
4. Click **Upload Package**.

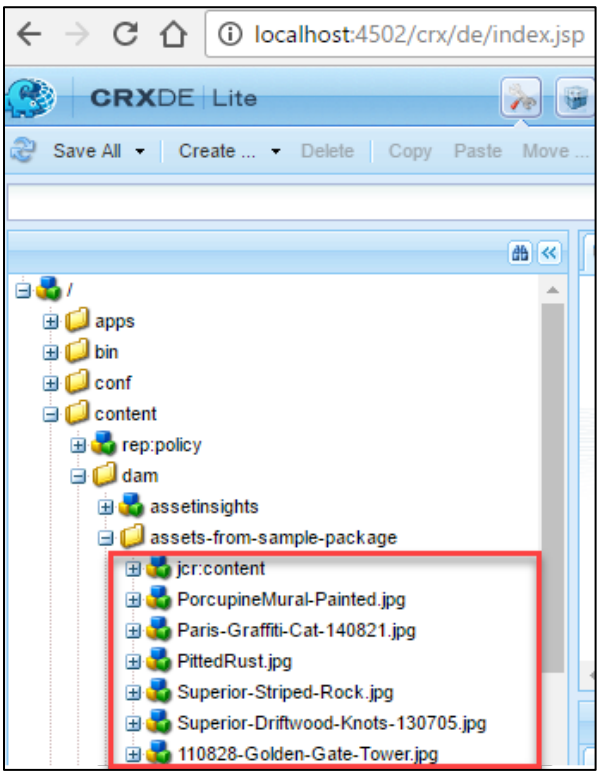
5. In the Upload Package dialog box, click Browse, and select the **SamplePackage.zip** package from the Exercise_Files provided to you. Click OK.



6. **After the package is uploaded, click Install.**

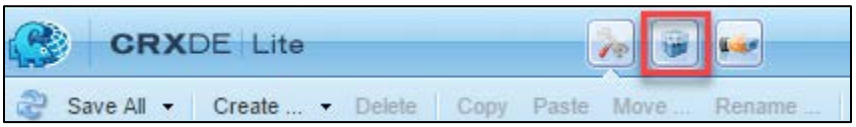


7. **In the Install Package dialog box, ignore the Advanced Settings area and click Install.**
8. **Check the Activity Log. You can see the content was added from the package.**
9. **Check the /content/dam/assets-from-sample-package folder in CRXDE Lite to see the changes reflected there. The contents of the package are added to the repository.**



4.1.2 Task - Create, Build, and Download a Package

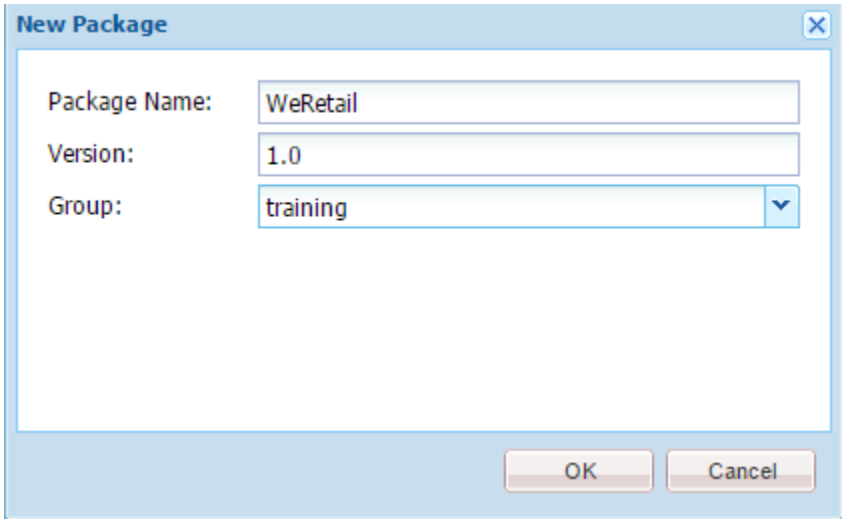
1. **Click the Package Manager button in CRXDE Lite. This takes you back to the Package Manager.**



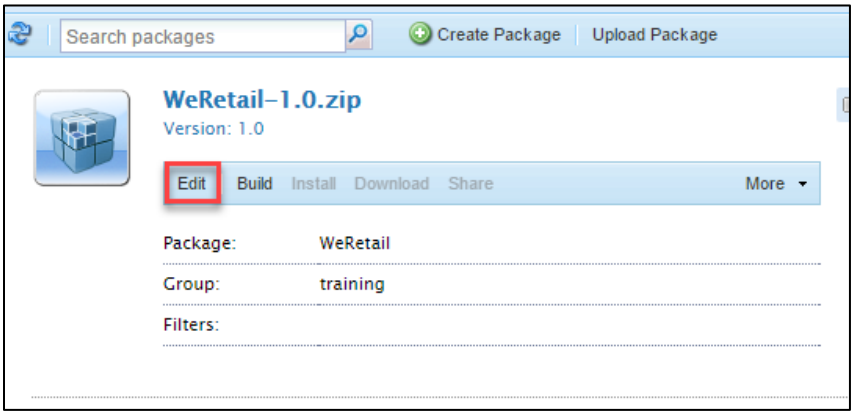
2. **As an alternative, click the following link to open the page directly: http://localhost:4502/crx/packmgr/index.jsp**
3. **Click Create Package.**

4. **In the New Package dialog box, enter the following details:**

Property	Value
Package Name	WeRetail
Version	1.0
Group	training

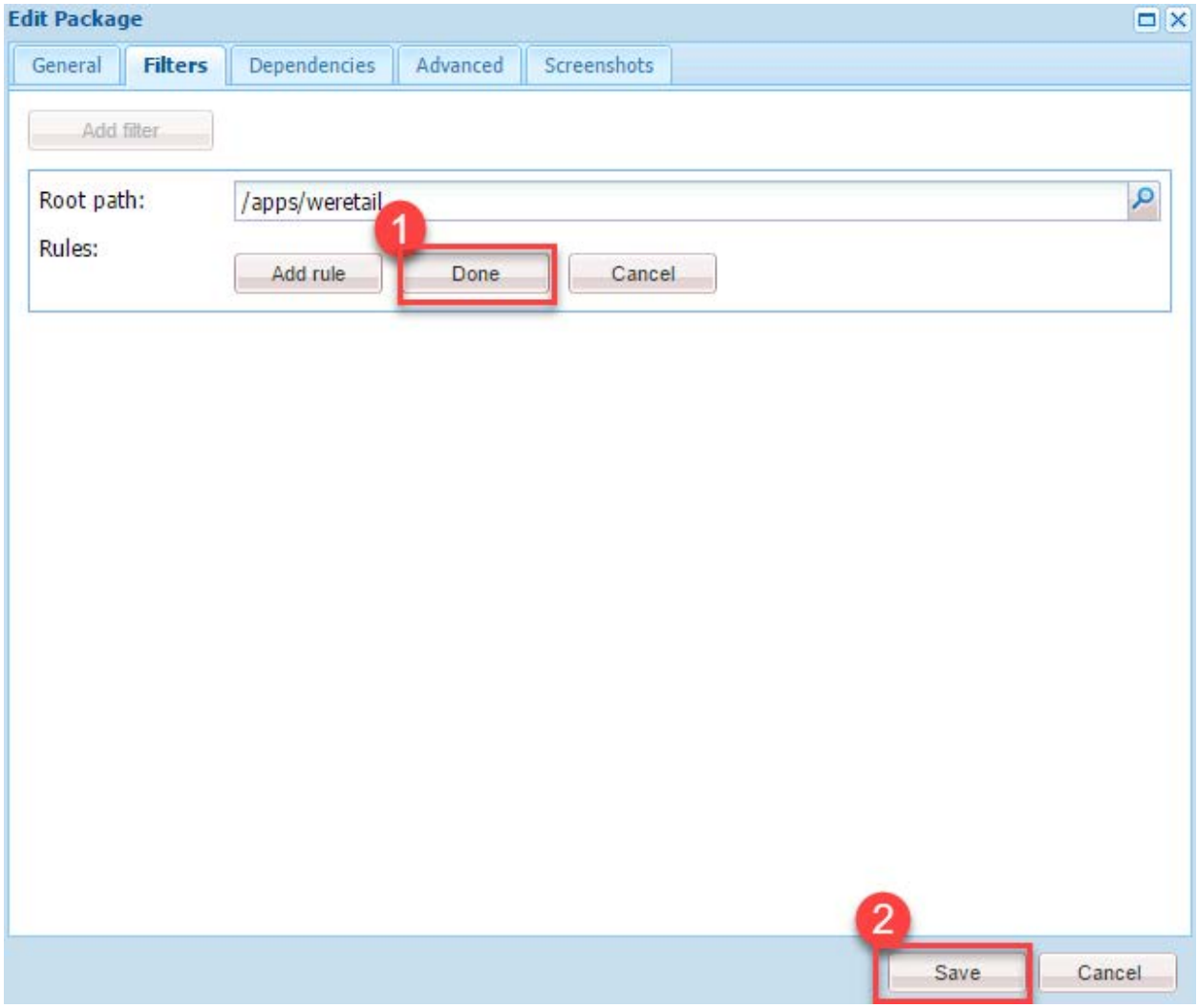


5. **Click OK.**
6. **Click Edit on the newly-created package.**

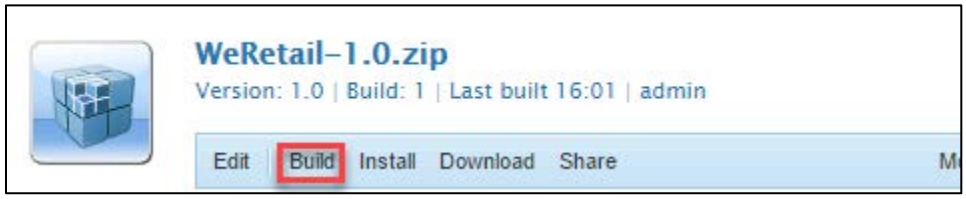


7. **To add filters to the package, click the Filters tab, and click Add Filter.**

8. **For the Root path, browse and select the WeRetail project folder under apps, and click OK.**
9. **Click Done, and then click Save.**



10. **Click Build to build the package, and then click Build again in the confirmation dialog box.**



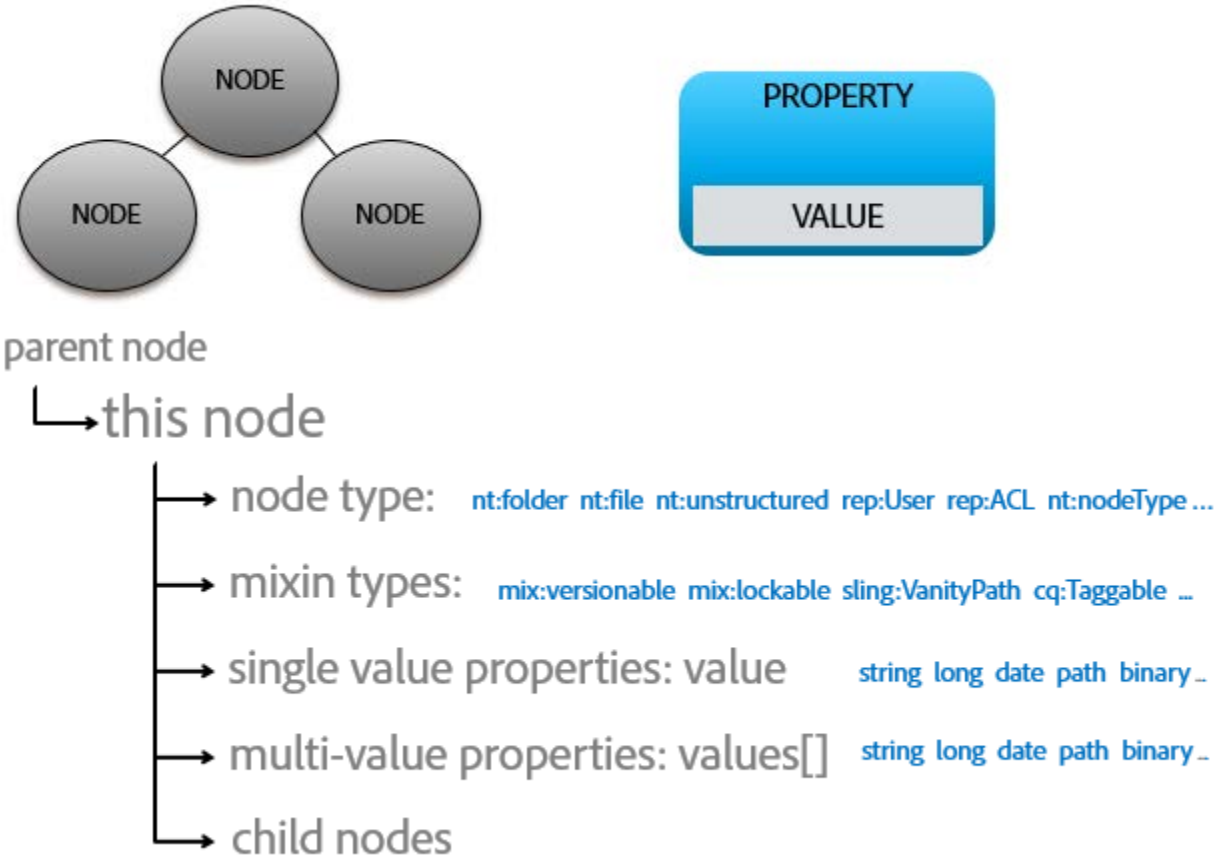
11. **The package is now ready for download. Click the Download link to download a copy of the package to your computer.**

5 INTRODUCTION TO CONTENT RENDERING

Nodes and Properties

You can think of the JCR as a hierarchical tree of two types of items:

1. Nodes: provide the structure. Nodes can have parent and child nodes which are represented with paths.
2. Properties: store the data. All properties have a name and a value.



Looking more closely at a node (example above), there are a number of other features worth noting:

- Nodes have a type. The node type sets rules about what kinds of properties and child nodes a node can or must have. For example, a node of type cq:page must always have a child node named jcr:content of type cq:PageContent.

- Nodes can also have zero to many mixin types. A mix-in type is an add-on; it specifies additional properties and child nodes the node must or may have. Common mixin types include mix:versionable and mix:referenceable.

Looking at the properties, a node may have any number of properties for storing information. Nodes can be single or multi-valued. A multi-valued property is notated and behaves much like an array. Multi-valued properties usually have plural names. So, here (above) we see a single value property named value and a multi-valued property named values[ ].

Namespaces

Common Namespaces for node types and properties:

- jcr: basic data storage (part of jcr spec)
- nt: foundation node types (part of jcr spec)
- rep: repository internals (part of jcr spec)
- mix: standard mixin node types (part of jcr spec)
- sling: added by Sling framework
- cq: added by the AEM application

Common JCR Node Types

Node Type	Description
nt:file	Represents a file, as in a filesystem.
nt:folder	Represents a folder, as in a filesystem
nt:unstructured	• Allows any combination of child nodes • Allows any combination of properties • Supports client-orderable child nodes • Used to store unstructured content • Commonly used for touch-enabled UI dialogs

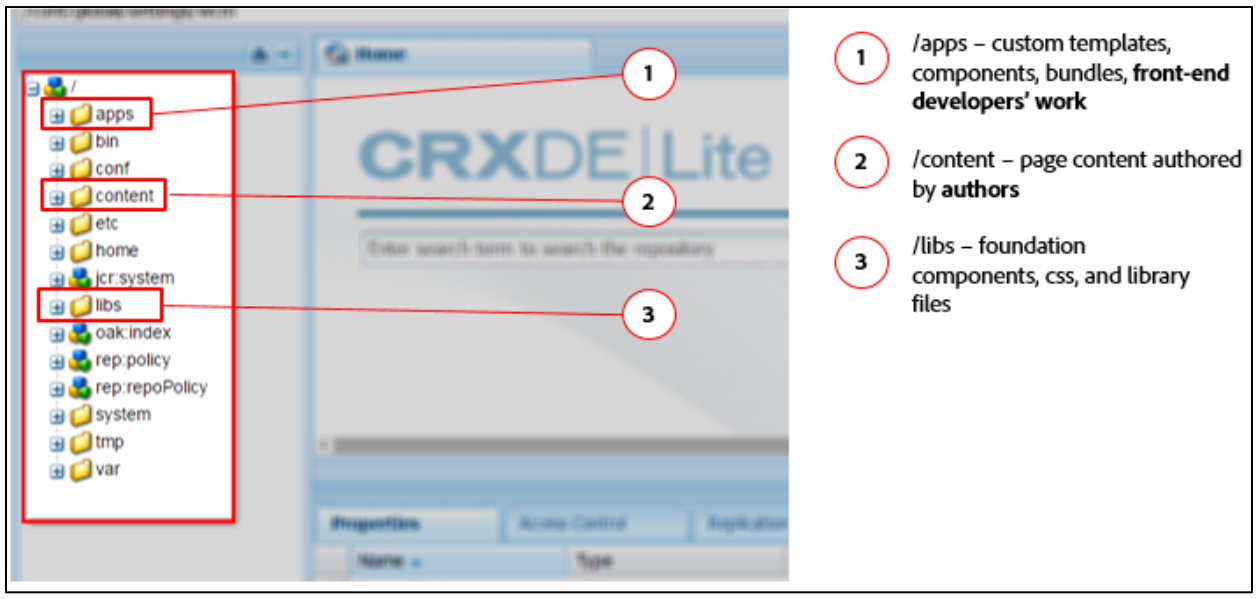
Common AEM Node Types

Node Type	Description
cq:Page	Stores content and properties for a page in a website
cq:Template	Defines a template used to create pages
cq:ClientLibraryFolder	Defines a library of client-side JavaScript CSS
cq:EditConfig	Defines editing configuration for a component including drag and drop and in-place editing.
cq:InplaceEditingConfig	Defines an in-place editing configuration for a component. Always seen as a child of cq:EditConfig

Folder Structure of the Repository

You have already seen how a project created using the Archetype 10 is structured. We will create a similar structure for our site. It is good practice to set up a specific folder structure that clearly defines and organizes the various elements of your site into its respective folders. These elements include your templates, components, OSGi bundles, and static files. You should create all custom projects under the /apps folder in CRXDE Lite.

Before we begin building our site, let's have a quick look at the folder structure in CRXDE Lite.



Here's what each of the folders contain:

/apps—stores all custom templates, components, and any other definitions related to your site are stored here. When you inherit foundation components, it is done from the libs folder. Always copy the components from the libs folder to the apps folder, and then modify it in the apps folder. By doing this, no code will be lost when you upgrade Adobe Experience Manager. You just need to update the apps folder.

NOTE: A best practice is to use a feature known as Sling Resource Merger (which will be discussed in Module 12 – Advanced Sling Functionality), and just duplicate the required /libs folder structure (empty folders) under /apps. You can then make modifications to the node or property, or wherever the changes are required.

/libs— stores all libraries and definitions that belong to Adobe Experience Manager code are stored here. It includes out-of-the-box components, templates, and other Adobe Experience Manager features such as search or replication. It is also referred to as the foundation components. **Avoid making any modifications to any of the components in libs**.

/content—all content of your website is stored in this folder.

/conf—contains all configurations for your site. This folder is used to store the dynamic templates and policies for your site.

/etc—contains all resources related to utilities and tools.

/home—contains all information related to users and groups.

/tmp—serves as a temporary working area.

/var—contains files that change and are updated by the system, such as audit logs, statistics, and event handling.

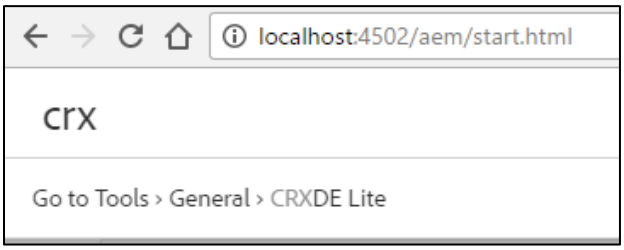
/oak:index—is a node that contains Jackrabbit Oak index definitions. Each node specifies the details of one index. Standard indexes for the AEM application are visible and additional custom indexes can be created.

Exercise - I

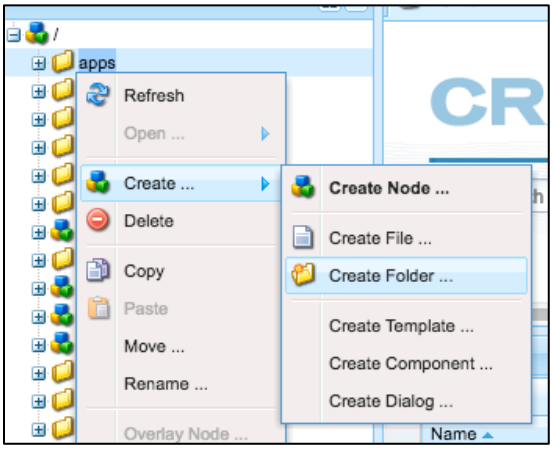
In one of the earlier modules, we touched on Apache Sling, the web framework. Now let's go a bit deeper and see how AEM uses Apache Sling to resolve a resource and then map that resource to a rendering script.

5.1.1 Task – Create the project structure in /apps

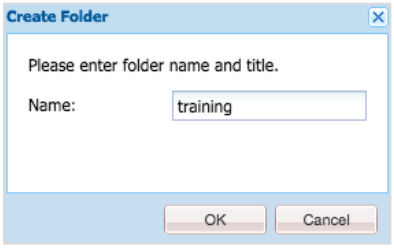
1. **Navigate to CRXDE Lite. http://localhost:4502/crx/de. Or from any console, enter "crx" into the OmniSearch toolbar and click on CRXDE Lite from the search results. You may load the OmniSearch at any time by typing "/" in AEM.**



2. **Right-click on /apps and select Create…> Create Folder.**



3. **Enter training as the name of the folder. Click OK and then click Save All in the upper-left.**

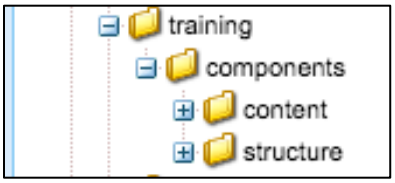


NOTE: Node names are a restricted set. Here are some best practice guidelines for node names:

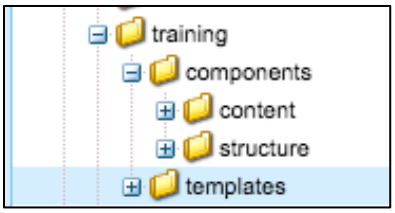
- No whitespaces or special characters.
- Use lower case.
- Do **NOT** use an "_" (underscore) character in node names; instead, use hyphens ("-") if you must separate two words or phrases in the name of a node. This also helps with SEO (Search Engine Optimization) best practices.
- More information: https://wiki.apache.org/jackrabbit/NodeNamingConventions

4. **Right-click on /apps/training and select Create…> Create Folder. Enter "components" as the folder name.**
5. **Save your changes using Save All. From this point forward in this course, you will click Save All to save any and all changes in CRXDE Lite. It is a best practice to click Save All often!**

6. **Now use what you learned to create two child folders of /apps/training/components named "content" and "structure". Save your changes.**



7. **To finish creating the project structure, right-click on /apps/training and create a folder named templates. Save your changes.**



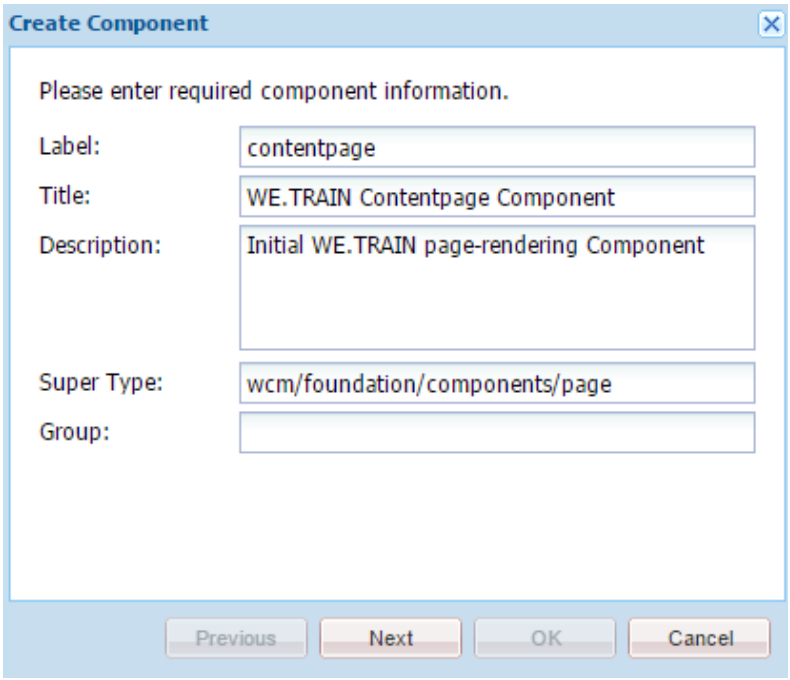
Page Rendering Components

- Components are modular, reusable units that implement specific functionality or logic to render the content of your website.
- A component is a collection of scripts (for example, HTL files, JSPs, Java servlets, and so on) that completely realize a specific function.
- Every component has a default script file identical to the name of the component. If needed, you can remove this script and create your own.
- The template sling:resourceType property must match the path to a page rendering component.

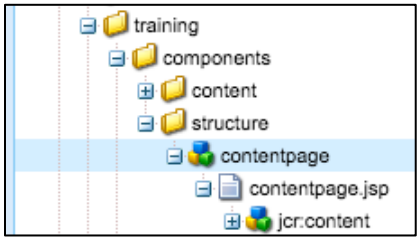
5.1.2 Task – Create a page-rendering component

1. **Right-click /apps/training/components/structure and select Create…> Create Component.**
2. **Fill in the following values:**

   - **Label**: contentpage
   - **Title**: WE.TRAIN Contentpage Component
   - **Description**: Initial WE.TRAIN page-rendering Component
   - **SuperType**: wcm/foundation/components/page (This indicates the Authoring UI is enabled)



3. **Click Next.**
4. **Click OK, and save your changes. Your folder structure should look like this:**



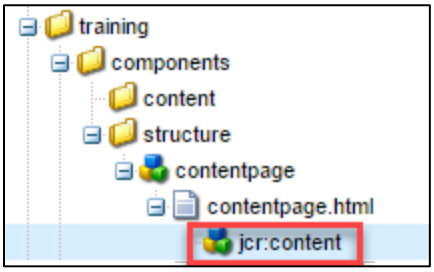
5. **Right-click on contentpage.jsp to rename it to contentpage.html. You are renaming this because you are populating this with HTML.**
6. **Save All.**
7. **Double-click on contentpage.html to open the script for editing.**
8. **Replace all sample code and comments with the HTML code from the Exercise Files for this course. The file is provided in the subfolder …/Module 5 Introduction to Content Rendering/Exercise - II/ Task - Create a Page-rendering Component/.**

NOTE: There are two ways you can replace HTML, JS, and any other code in CRXDE Lite in this entire course (OPTION A and OPTION B):

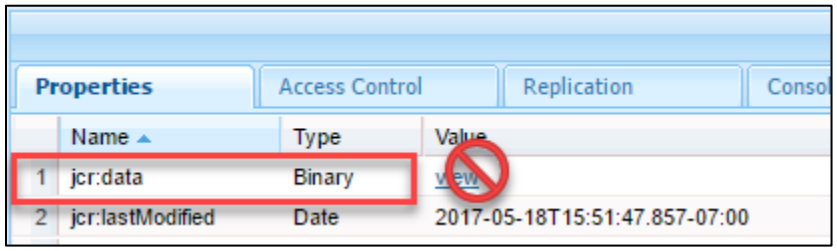
OPTION A: Open the code files from the extracted Exercise_Files directories in Notepad++ (or similar text editor), copy all the contents of the files, and paste the contents into CRXDE Lite. Save All!

OPTION B: Use CRXDE Lite's method of populating code in the jcr:data node. The steps are as follows. You may follow these steps as-is to complete Step 8:

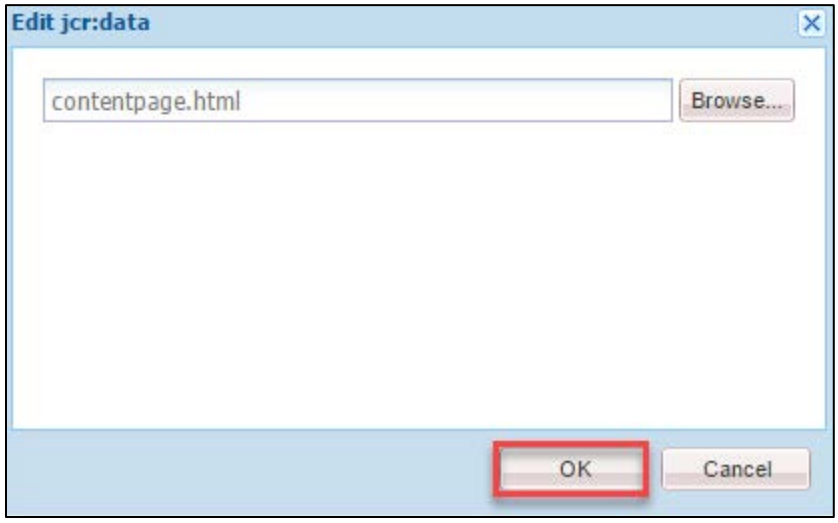
a. Under the renamed (or newly created) file (in this case, **contentpage.html**), find a **jcr:content** node. Expand this node:



b. **Double-click on the jcr:data property within this node. Do NOT click on the View link:**



c. **In the edit dialog that appears, click Browse and select the file in the extracted Exercise_Files directory on your computer or ReadyTech instance you want to use to replace the file in CRXDE Lite (in this case, .../Module 5 Introduction to Content Rendering/Exercise - II/ Task - Create a Page-rendering Component/contentpage.html):**



d. **Click Ok. Save All in CRXDE Lite.**

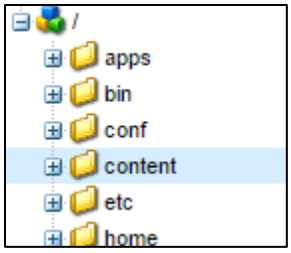
NOTE: You may use this method to replace the contents of files as well. You can use this method for all situations where you create a new file, or replace the contents of files. If you are creating a new blank file, remember to first save the file after it is initially created. Then use the method of double-clicking on the jcr:data property.

5.1.3 Task – Create content to be rendered

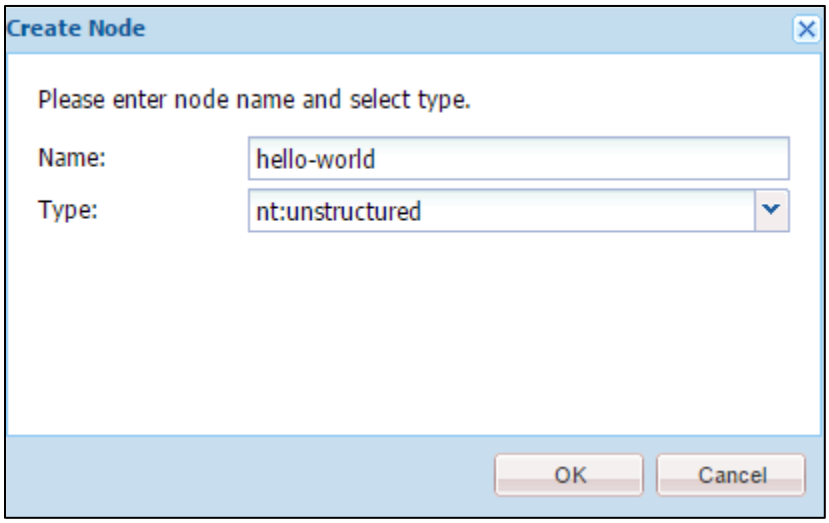
Now you have a component that will render content, we need some actual content to render.

1. **Navigate in CRXDE Lite to /content.**

   NOTE: This is the at the root, not within your **/training** project structure.



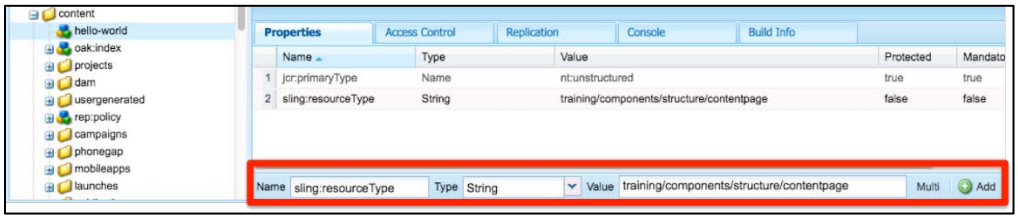
2. **Right-click on /content and select Create…> Create Node.**
3. **Create a node named "hello-world" of type "nt:unstructured":**



4. **Save your changes.**
5. **In your node, add a property with the following attributes, and click Add.**

Name	Type	Value
sling:resourceType	String	training/components/structure/contentpage

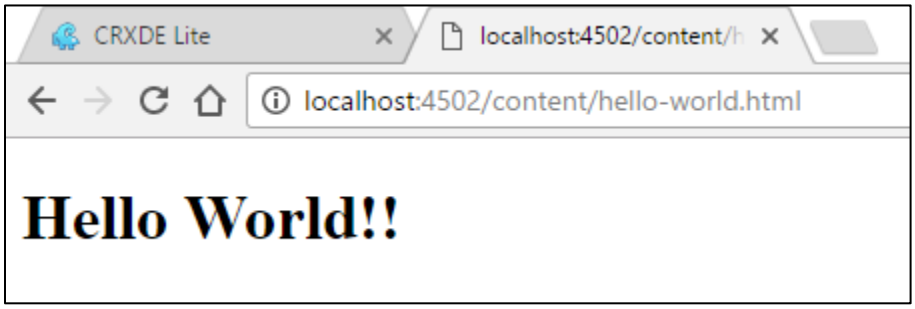
Notice the sling:resourceType property value is the path to the page-rendering component that we just defined.



Pro Tip: By convention, property names are camel case, excluding the namespace prefix (sling:). Camel case means the first word in the property name starts with a lowercase letter and each additional word in the property name starts with an uppercase letter (in the above, **resourceType** follows the camel case syntax).

6. **Save your changes.**

Open http://localhost:4502/content/hello-world.html in the browser.



7. What was the sequence of events that caused the Hello World text to display?

Understanding the Sling Resolution Process

- Sling is resource-oriented.
- All resources are maintained in the form of a virtual tree.
- A resource is usually mapped to a JCR node, but can also be mapped to a file system or database.
- Common properties that a resource can have:
    - Path
    - Name
    - Resource Type

URL ➔ Resource = JCR Structure

Basic Steps of Processing Requests

Each content item in the JCR repository is exposed as an HTTP resource.

After the content is determined, the script or servlet to be used to handle the request is determined through:

- Properties of the content item itself
- The HTTP method used to make the request
- A simple naming convention within the URL that provides secondary information

For every URL request, the following steps are performed to get it resolved:

1. Decompose the URL
2. Search for a servlet or a vanity URL redirect
3. Search for a node indicated by the URL
4. Resolve the Resource
5. Resolve the rendering script/servlet
6. Create rendering chain
7. Invoke rendering chain

Decomposing the URL

Consider the following example:
URL : `http://myhost/tools/spy.printable.a4.html/a/b?x=12`

The above URL can be decomposed into the following components:

protocol	Host	Content path	Selector(s)	Extension		Suffix		Param(s)
http://	myhost	tools/spy	.printable.a4	html	/	a/b	?	x=12

where:

- **Protocol:** Hypertext transfer protocol (HTTP)
- **Host:** Name of the website
- **Content path:** Path specifying the content to be rendered.

- **Selector(s):** Used for alternative methods of rendering the content
- **Extension:** Content format; also specifies the script to be used for rendering.
- **Suffix:** Can be used to specify additional information
- **Param(s):** Any parameters required for dynamic content

Resolving Requests to Resources

Let's take a look at a couple of examples:

Example 1:
URL : `http://myhost/tools/spy.html`

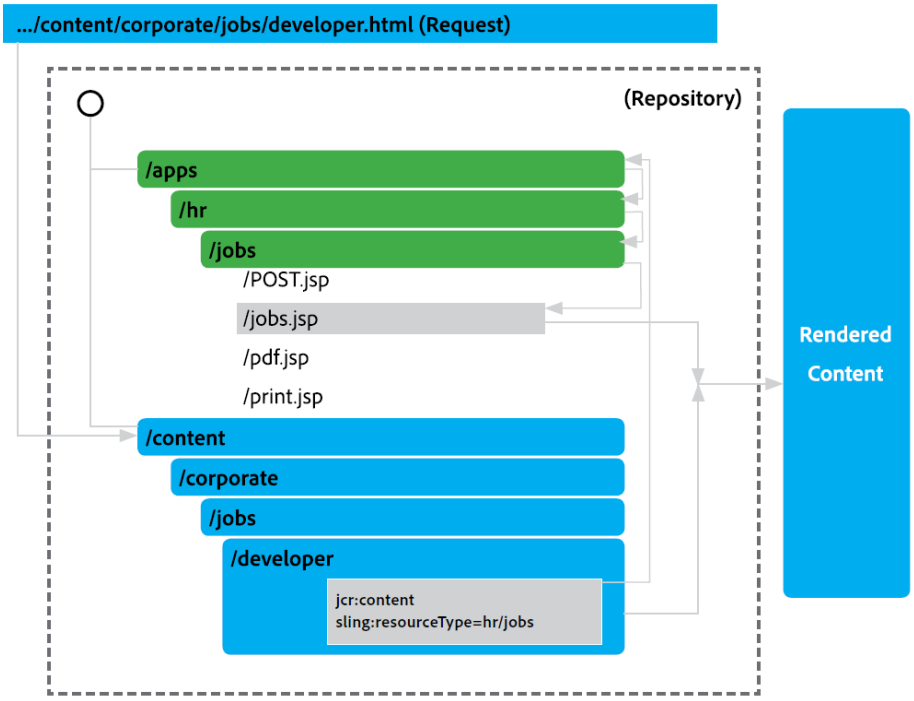
Once the URL is decomposed, the content node is located from the content path. This node is identified as the resource, and performs the following steps to map to the request.

Steps performed to map request:

1. The Sling Resource Resolver process first looks for a redirection rule (such as a vanity URL), or a servlet and resolves to this. If this does not exist or is not found, then the script resolution process begins.
2. As part of the script resolution process, Sling searches for the **spy** node.
3. If a node is found, then the sling resource type for that node is extracted, and used to locate the script to be used for rendering the content.
4. If no node is found, then Sling will return the http code 404 (Not Found).

Example 2:

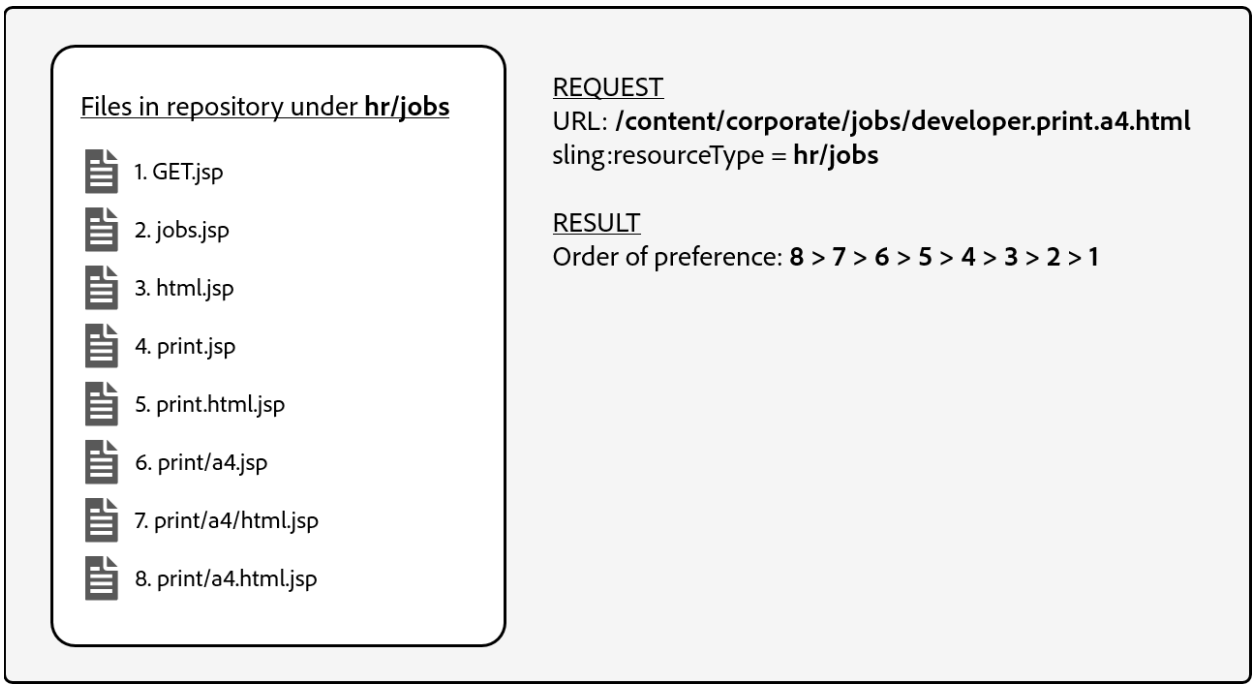
Consider the URL request: **http://myhost/content/corporate/jobs/developer.html** and corresponding diagram. Notice the properties of the developer node are used to provide the rendered content:



Locating and Rendering Scripts

When the resource is identified from the URL, its resource type property is located and the value extracted. This value is either an absolute or a relative path that points to the location of the script to be used for rendering the content. All scripts are stored in either the /apps or /libs folder, and are searched in the same order. If no matching script is found in either of the folders, then the default script is rendered.

For multiple matches of the script, the script name with the best match is selected. The more selector matches, the better.

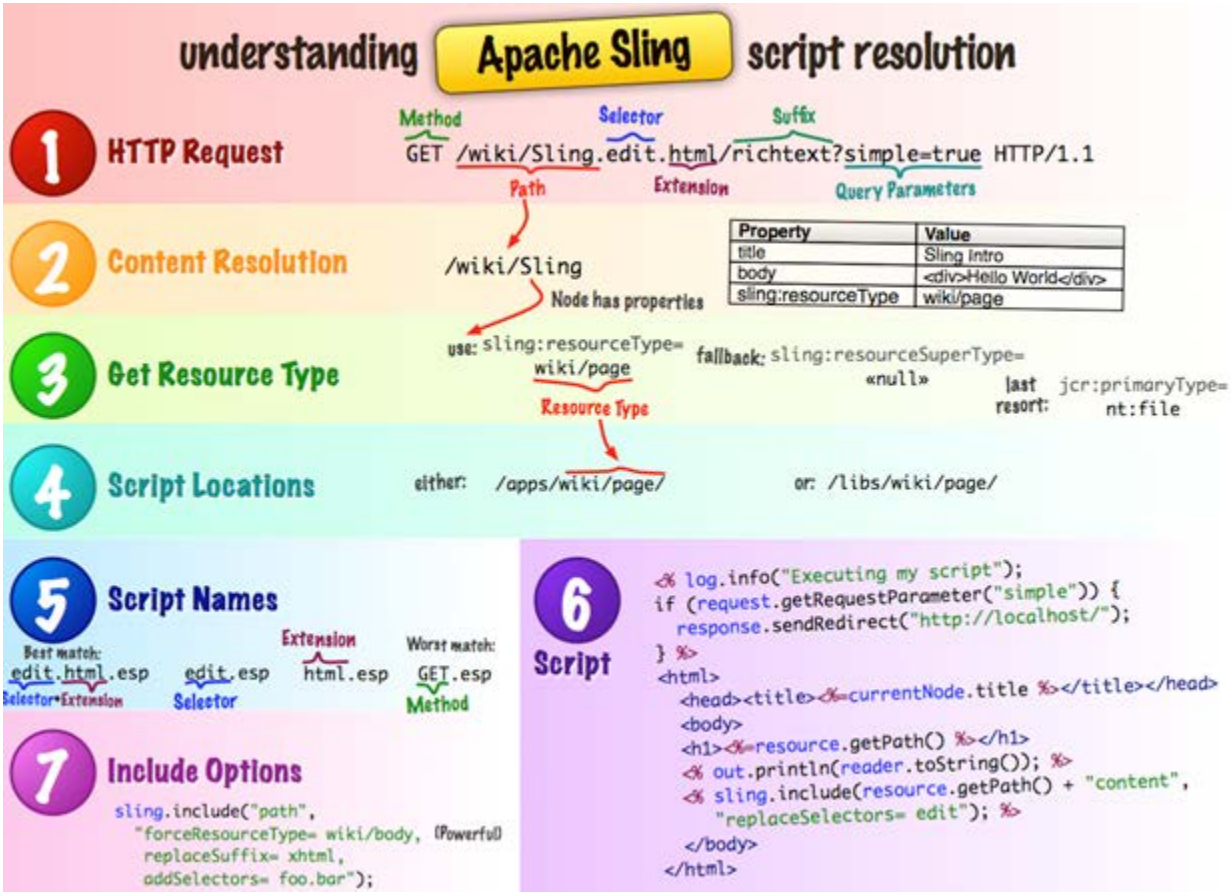


Understanding URL Decomposition

Each content item in the JCR repository is exposed as an HTTP resource, so the request URL addresses the data to be processed, not the procedure that does the processing. After the content is determined, the script or servlet to be used to handle the request is determined in cascading resolution that checks, in order of priority:

1. Properties of the content item itself
2. The HTTP method used to make the request
3. A simple naming convention within the URL that provides secondary information

Therefore, it must be noted again that chronologically, the **Sling Resource Resolver first** tries to resolve the URL to a servlet or redirect. If this does not exist or is not successful, then the **script resolution process** described below takes place:



If no node is found at all, a 404 error is returned.

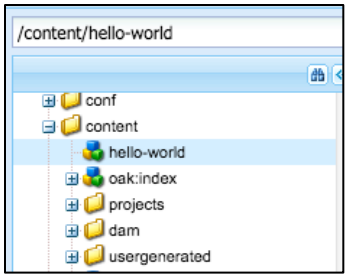
5.1.4 Task – Basic Sling resource resolution and hunt for a rendering script

When presented with a request (URL), Apache Sling will take the following actions:

1. **Strip off the protocol, server, and port.**
2. **Strip off any suffix.**
3. **Examine the remaining portion of the URL and use the information contained therein to assist with resolving the resource and finding its associated rendering script.**

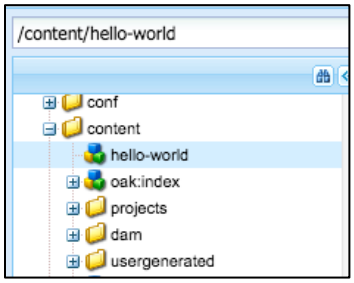
Let us follow Sling's actions to see how our hello-world resource was rendered.

1. **Consider the request (URL) http://localhost:4502/content/hello-world.html**
2. **Disregard http://localhost:4502.**
3. **In CRXDE Lite, navigate to /content/hello-world.html. Notice that path does not exist in the repository.**



4. **At this point, Sling will back off the end of the remaining part of the request until the first ".". Everything after the "." is considered the extension. Now Sling will attempt to locate "/content/hello-world".**

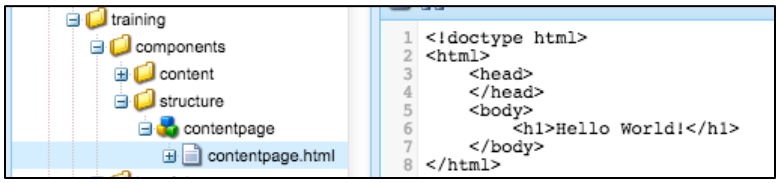
5. **In CRXDE Lite, navigate to /content/hello-world. Yay! We found a node in the repository that matches the resource in the request. What we just did is called "resolving the resource". We successfully resolved a request URL to a resource that is represented by a node in the repository.**



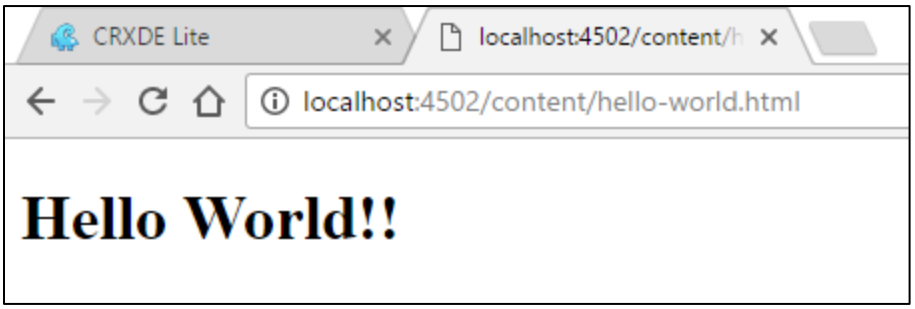
NOTE: Resources could also map to items in relational databases and/or file system. The Apache Sling specification does not mandate a JCR database.

6. **Now we need to find the rendering script. On the /content/hello-world node, find the sling:resourceType property. The value of the sling:resourceType property is what Sling uses to begin its search for a rendering script.**
7. **Notice the sling:resourceType property points to the page-rendering script that we created previously.**

8. **Navigate to: /apps/training/components/structure/contentpage**
9. **Notice there is one script: contentpage.html. This is called the default script because the script's name matches the name of the folder/component in which the script resides. Later, we will explore the many options that Sling might use in selecting the "right" script. But right now, we have the default script. Given the specified request and the available selection of scripts, the default script is the best match and Sling chooses it to render the resource.**

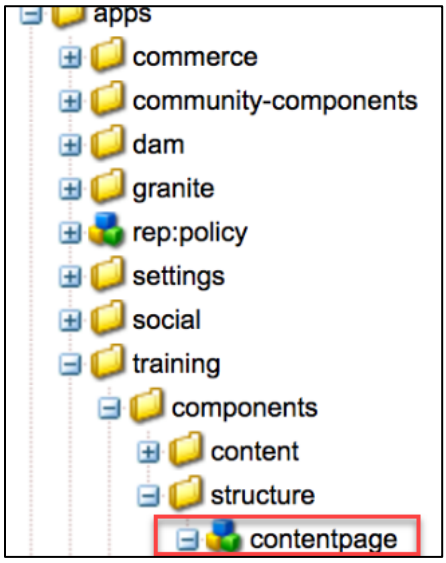


10. **The script outputs "Hello World!". We have now successfully resolved the resource, found the rendering script, and invoked the rendering chain.**
11. **You have just seen how the request "http://localhost:4502/content/hello-world.html" results in the following browser output:**



5.1.5 Task – Selector Manipulation

1. **Using CRXDE Lite, right-click on /apps/training/components/structure/contentpage.**

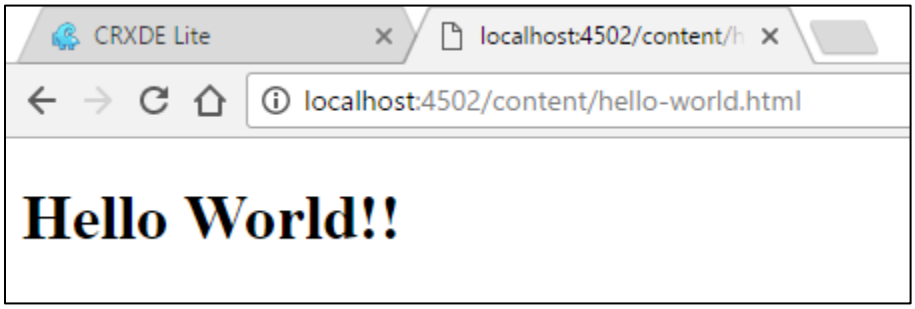


2. **Select Create… > Create File.**
3. **Enter blue.html as the file name.**
4. **Click OK. Save your changes.**
5. **Open the file blue.html and enter the following statement. You will find the code in the Exercise Files as well.**

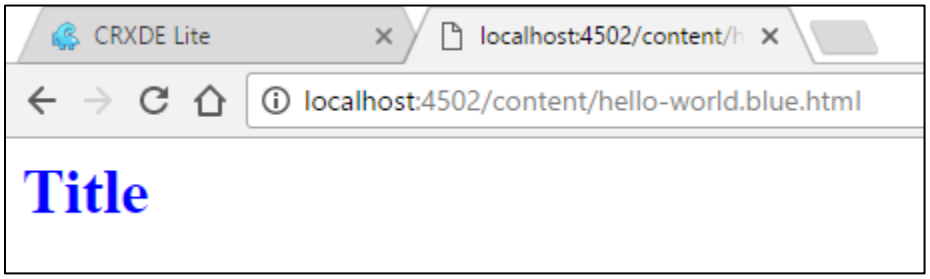
   <h1 style="color:blue">Title</h1>

6. **Save your changes.**

   To test our selectors, enter the URL in your browser that will cause our hello-world node to be rendered: **http://localhost:4502/content/hello-world.html.**



7. **Now enter http://localhost:4502/content/hello-world.blue.html. Notice the difference. The code from blue.html is chosen, as the script name matches the selector in the URL.**



We will continue to learn more about Apache Sling URL decomposition and resource resolution throughout the course.

6 TEMPLATES

What is a template?

Templates are used to create pages. When an author creates a page, the nodes and properties of the template are copied to the location of the new page. When pages are created, the author must choose a template to base the page on.

When the *sling:resourceType* property of the template is copied to the new page location a connection is made between the page and its rendering component. Other properties of the template can control where the template may be used and even the order it will appear in the Create Page dialog.

The template can also contain initial content for the page in a *jcr:content* node tree.

Properties of a Template

AEM templates have the following attributes and properties (not a complete list but includes the most important properties):

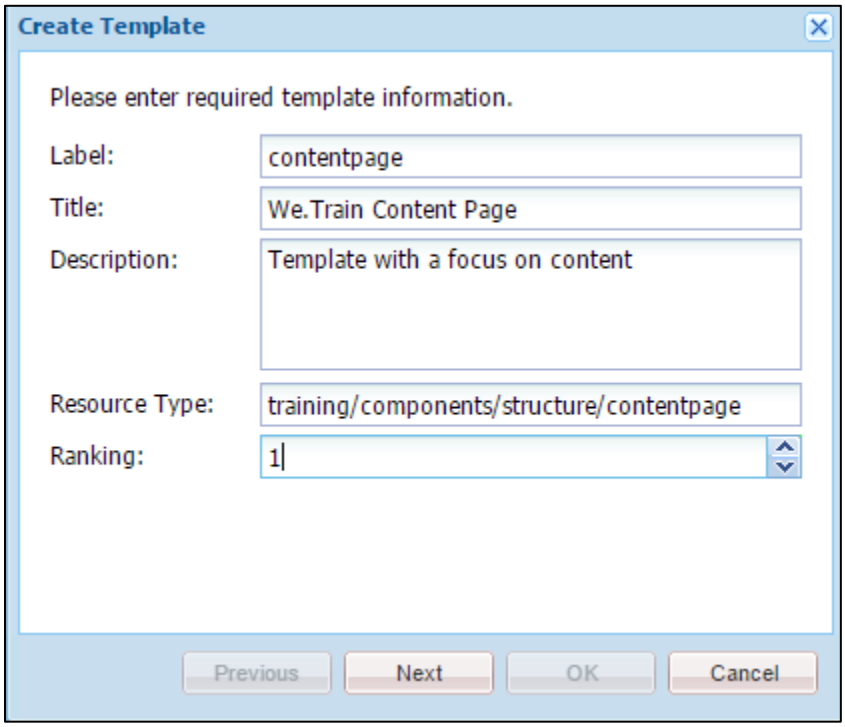
- label – becomes the node name
- sling:resourceType – defines the path where the rendering script search begins
- jcr:title – the string used by the Consoles to display this template
- jcr:description – documentation/description for this template
- ranking – defines the order in which this template will appear, as compared to other available templates in the "Create" dialog
- allowedPaths – paths where this template can be used
- allowedParents – paths of templates that are allowed to be parent to this template
- allowedChildren – paths of templates that are allowed to be children to this template
- thumbnail.png – a thumbnail image that helps the author identify this template

Exercise - I

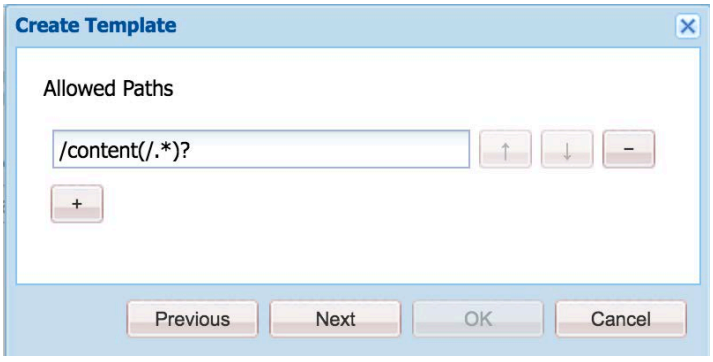
Templates play many parts within the AEM ecosystem. One important part is to be in the list of available templates when the content author creates a page.

6.1.1 Task – Create a Template

1. **In CRXDE Lite, navigate to /apps/training/templates.**
2. **Right click /apps/training/templates and select Create…> Create Template.**
3. **Enter the following values:**
   - **Label**: contentpage
   - **Title**: We.Train Content Page
   - **Description**: Template with a focus on content
   - **Resource Type**: training/components/structure/contentpage
   - Ranking: 1



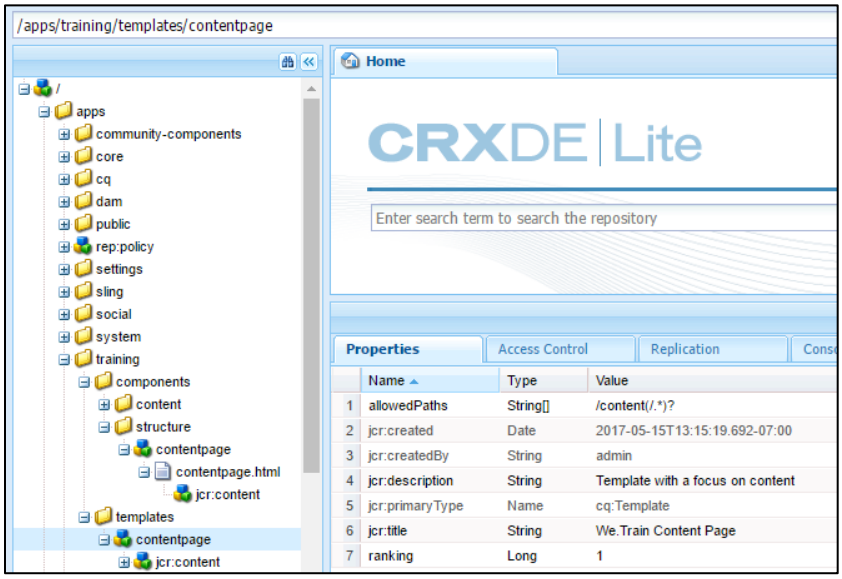
4. **Click Next.**
5. **Click the + and enter /content(/.*)? for Allowed Paths.**



6. **Click Next.**
   NOTE: Do not click on the "+" for Allowed Parents.

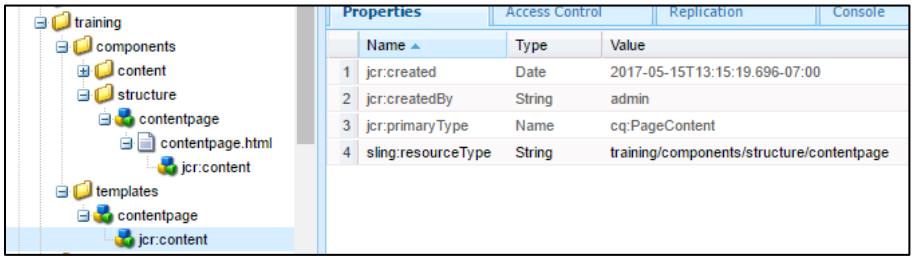
7. **Click Next.**
   NOTE: Do not click on the "+" for Allowed Children.
8. **Click OK.**
9. **Save all changes.**

Your project node structure should look like this:



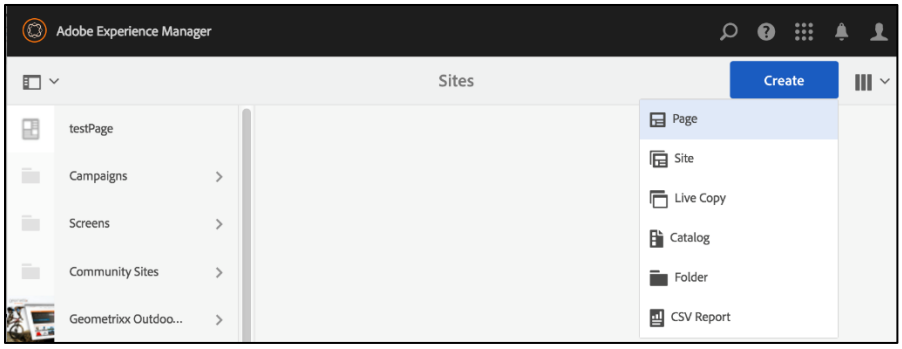
10. Take a look at the properties that were created on the /apps/training/templates/contentpage node:
    - allowedPaths property specifies where the template can be used to create pages and came from the Allowed Paths pane in the Create Template wizard.
    - jcr:description came from the Description entry in the Create Template wizard.
    - jcr:title came from the Title entry in the Create Template wizard
    - ranking property specifies where in the list of templates this template will appear and came from the Ranking entry in the Create Template wizard. Ours will be first, as we have assigned a ranking of "1".

Notice that the value from the Resource Type entry in the Create Template wizard does not appear. This value becomes the sling:resourceType property, which does not appear at this level. We will find the sling:resourceType property on the jcr:content child node of the contentpage template node.

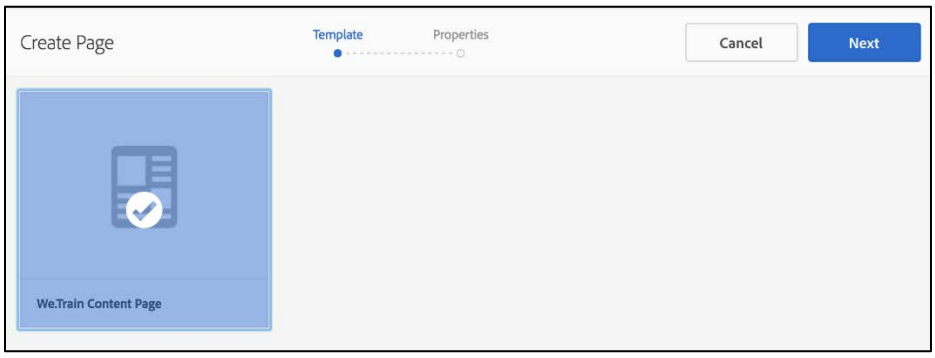


6.1.2 Task – Test the contentpage template

1. **Navigate to the Sites Console by entering http://localhost:4502/sites.html or by typing "/", entering "sites" into the OmniSearch toolbar, and choosing Go to Sites.**
2. **Click Create and choose Page.**



3. **Select the We.Train Content Page template and click Next.**

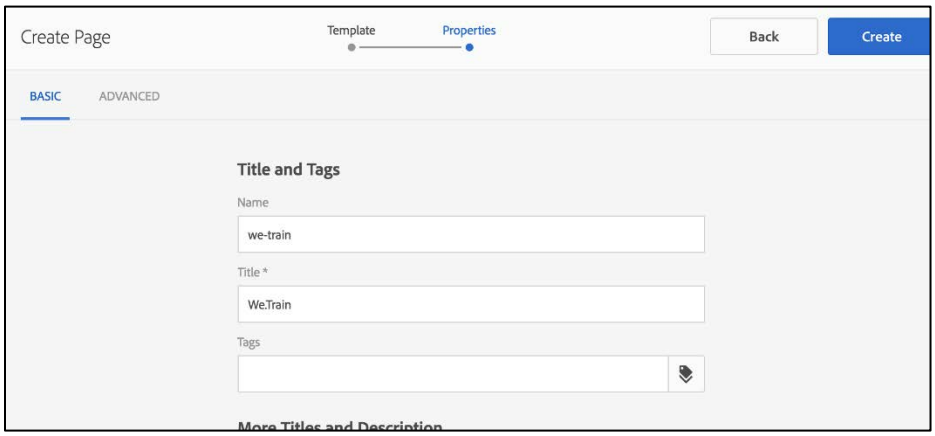


Troubleshooting tip: If your template does not appear in the list, check the allowedPaths property on the /apps/training/templates/contentpage template. The value should be: "**/content(/.*)?**". Any other value will prohibit the template from appearing in the list of available templates.

4. **Enter the following properties:**

Name: we-train

Title: We.Train



5. **Click Create.**
6. **Click Open (This opens the page in Edit mode). It should appear like this:**

Troubleshooting tip: If your page is blank, check the **sling:resourceType** property on the /apps/training/templates/contentpage/jcr:content node. It should exactly match the path to the contentpage page-rendering component. For example, the relative path might be training/components/structure/contentpage, or an absolute path might be /apps/training/components/structure/contentpage.

If the sling:resourceType path is incorrect, Apache Sling will not be able to find a rendering script and that is why you are seeing a blank page. Correcting the sling:resourceType property on the template will correct the problem for all new pages. Now you have two choices: 1) you can delete the page you created and recreate it or 2) correct the sling:resourceType property on the jcr:content child of the page you created.

Pro Tip: When a blank page or blank component occurs, look in the error log, located at <AEM instance root>/crx-quickstart/logs/error.log. You will find an entry for the request that states "… could not find renderer for "<path to the request>. This error message tells you that Sling could not find a rendering script for that request. Carefully examine the path specified in the error message and then look in the repository to fix the problem.

6.1.3 Task – Restrict Template Use

You can restrict where a template can be used in two ways. We have already seen that the allowedPaths property on the /apps/training/templates/contentpage template will define the paths where this template can be used to create pages. This restricts the template from the template's point of view.

1. **Navigate to the AEM Sites Console. Click Create Page.**
2. **Notice that the We.Train template appears.**
3. **Do the same with Sites>Screens. Notice the We.Train template appears.**

NOTE: We probably want the We.Train template to appear as an available template only for the We.Train site.

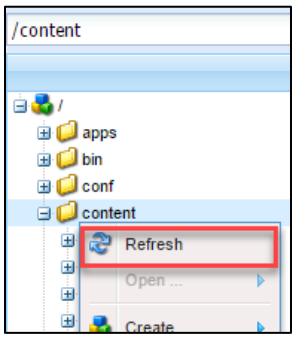
4. **Using CRXDE Lite, navigate to /apps/training/templates/contentpage.**
5. **Modify the value of the allowedPaths property (by double-clicking on this property) to be:**

   /content/we-train(/.*)?

6. **Repeat steps 1-3. Notice that this time the We.Train template no longer appears as a choice for Screens.**
7. **Navigate to We.Train.**
8. **Click Create Page. Notice that the We.Train Content Page template is available.**

Another way to restrict templates is to specify which templates may be used within a specified path. We can accomplish this using the cq:allowedTemplates property on a page.

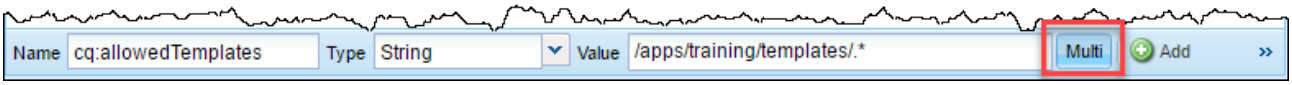
9. **Refresh CRXDE Lite by clicking on the /content folder and choosing Refresh:**



10. **Using CRXDE Lite, navigate to /content/we-train/jcr:content.**
11. **Add the following property:**

    - Name: cq:allowedTemplates
    - Type: String[]
    - Value: /apps/training/templates/.*

    Tip: To enter a String Array (String[]), choose String and click the Multi button. Beware! The Multi button stays clicked until you unclick it.



12. **Save your changes.**
13. **Test out the restriction by going to the AEM Sites Console. Using the same logic as we used previously in this task, make sure that the We.Train template is still available within the We.Train site.**

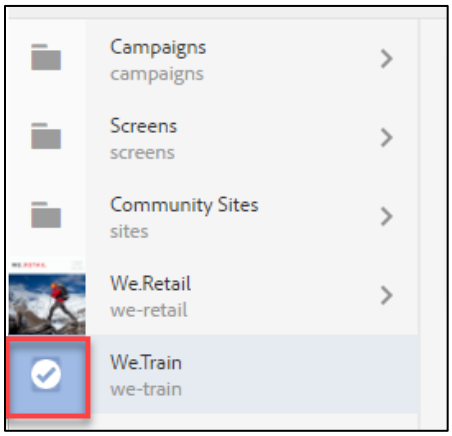
These two mechanisms allow us to control where templates are made available for use. The allowedPaths property allows the developer to decide where to use templates. The cq:allowedTemplates property allows the website owner to decide which templates to use within the site.

6.1.4 Task – Add Content Structure to the Template

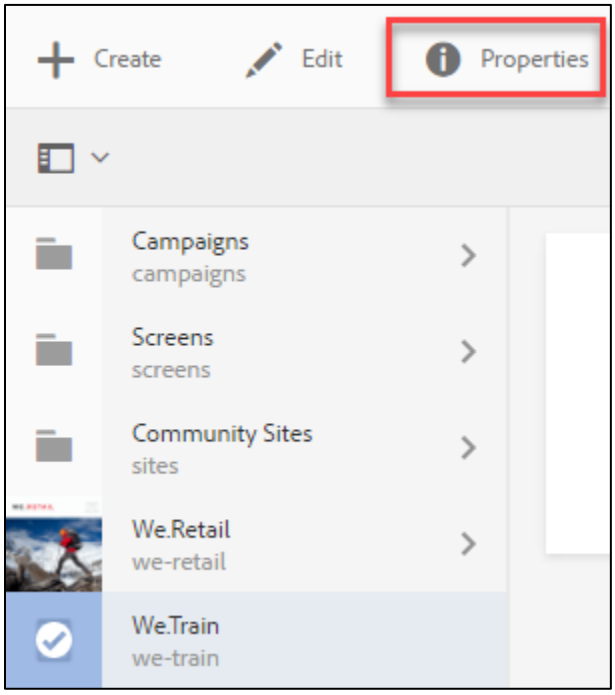
As mentioned earlier, the template plays many roles. One role that we experienced earlier is to appear in the list of available templates when creating a page. Another role that the template plays is to "front-load" content onto a page created from that template.

So, let's say we wanted every page created by the contentpage template in the We-train site to have the same thumbnail image. Here's how to achieve this:

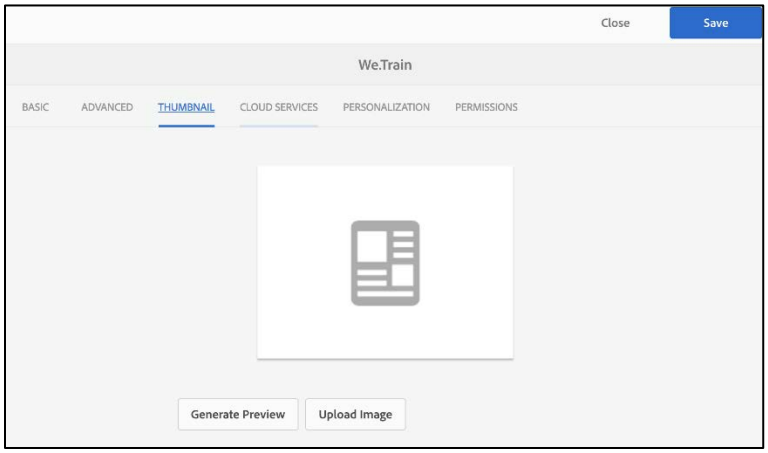
1. **Using the Sites Console, navigate to We.Train.**
2. **Select We.Train:**



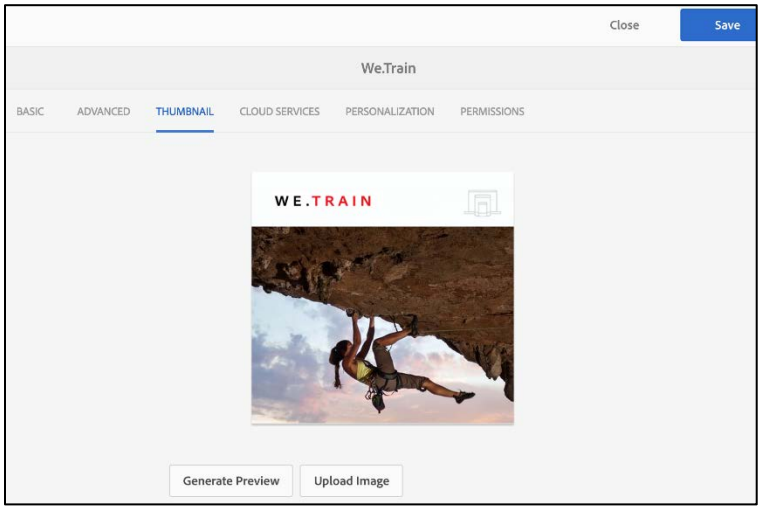
3. **Click Properties.**



4. **Click the Thumbnail tab.**

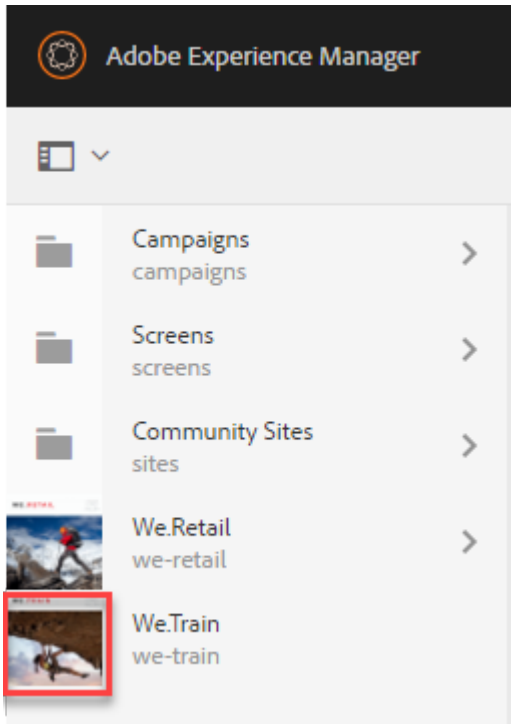


5. **Click on Upload Image and upload We-train.png from the exercise files provided to you for this exercise into the thumbnail.**



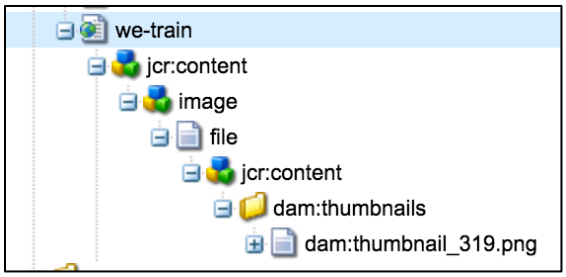
6. **Click Save & Close in the upper right.**

7. **Notice that your Site now has a thumbnail image associated with it:**



8. **Now we have created the thumbnail structure under the /content/we-train page that we want to have on every page. Let's modify the definition of the /apps/training/templates/contentpage template so that every page created from that template has this thumbnail.**
9. **Using CRXDE Lite, navigate to /content/we-train/jcr:content.**

   Notice the newly defined structure under the jcr:content node. **NOTE:** You may need to refresh CRXDE Lite.

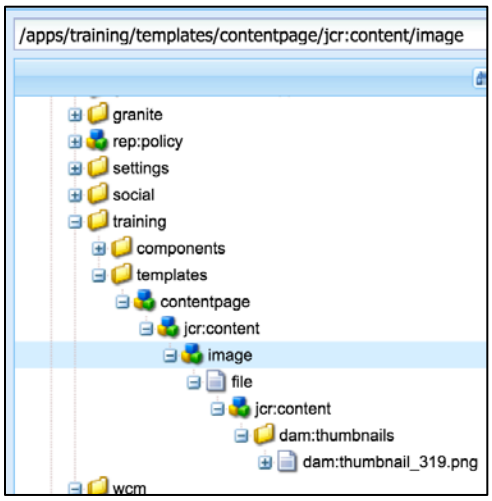


10. **Right-click on the image node and select Copy.**

NOTE: Do not copy the "dam:thumbnail_319.png" node. Instead, ensure you are copying the image node directly under jcr:content.

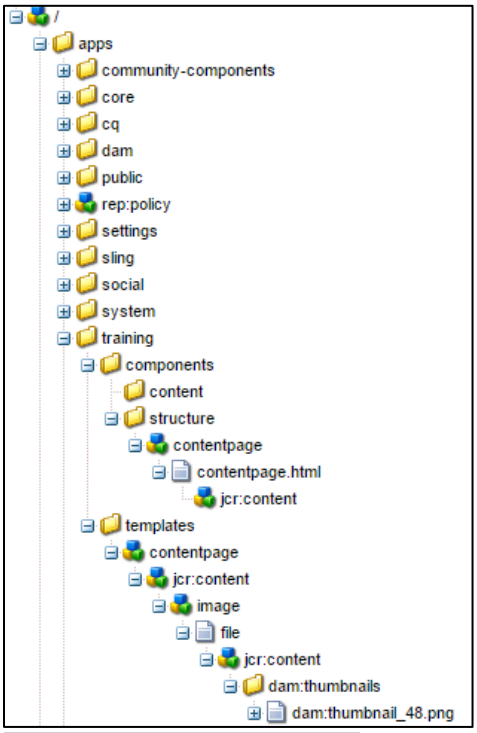
11. **Navigate to /apps/training/templates/contentpage/jcr:content.**

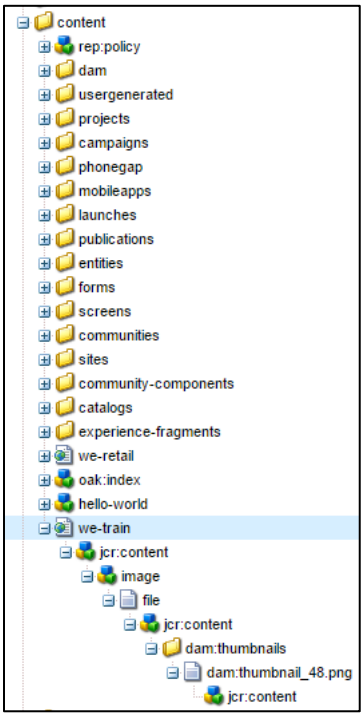
12. **Right-click the jcr:content node and select Paste. Your node structure should now look like this (NOTE: The "319" number at the end of the *.png filename may be different in your instance):**



NOTE: When a page is created from a template, AEM writes the cq:Page parent and then copies the template's jcr:content child as the child to the cq:Page node. Now, every page we create from the contentpage template will have the thumbnail.

NOTE: Your /apps and /content node structures (respectively) should look like this in CRXDE Lite:





Creating the Website Structure

At a very basic level, creating a page structure is how a website structure is set up. A page is where content authors create and edit content that most likely will be published and viewed by site visitors.

A page is many things:

- Website content container
- Instance of a template
- cq:Page JCR Node type (has a mandatory jcr:content child node)

When creating a page, the content that authors enter in the dialog box becomes the nodes and associated properties for that page. The Page Creation wizard allows you to enter or select the following information, which is necessary to create the complete page structure.

- Name: cq:Page node name
- Title: jcr:title property
- Template: cq:template property

The template's *sling:resourceType* property is added as the page's *sling:resourceType* property, so the page knows where its rendering script is. When working with pages, you can use the following WCM APIs:

- com.day.cq.wcm.api.Page
- com.day.cq.wcm.api.PageManager
- com.day.cq.wcm.api.PageFilter

6.1.5 Task – Create the pages for the site

Now it is time to create our website structure. It won't be a very interesting website yet, as all the pages will be identical. But we will continue to build out our page-rendering component and create other components to fill out the page content.

1. **Using the Sites Console, navigate to We.Train.**
2. **Click Create Page.**
3. **Choose the We.Train template and click Next.**
4. **Fill in the Name and Title:**

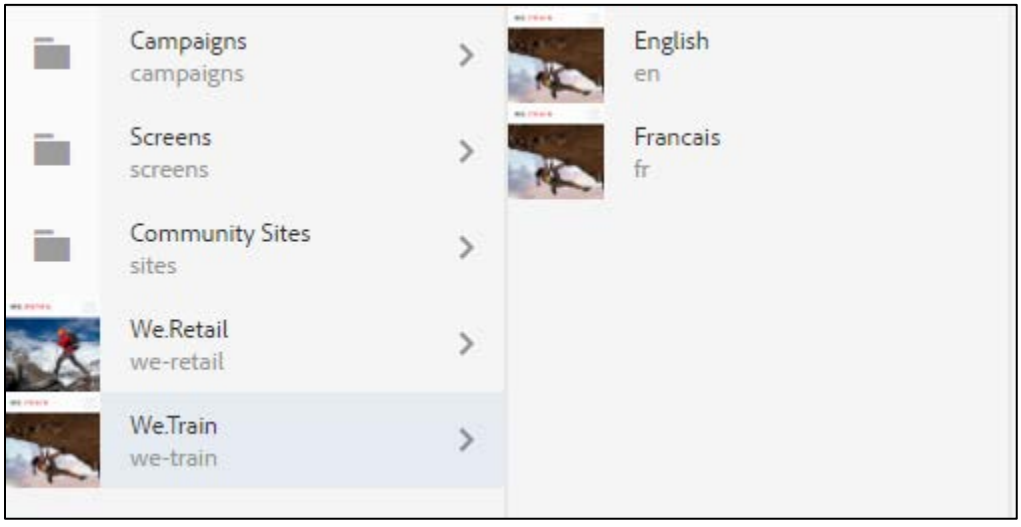
- Name: en
- Title: English

  (This page will be the English language root.)

5. **Click Create, and then click Done.**
6. **At the same level, create the French language root page:**

   - Name: fr
   - Title: Francais

   Notice the thumbnail image appearing under the English and French language roots:



This came from the content that we placed under the template in the last exercise.

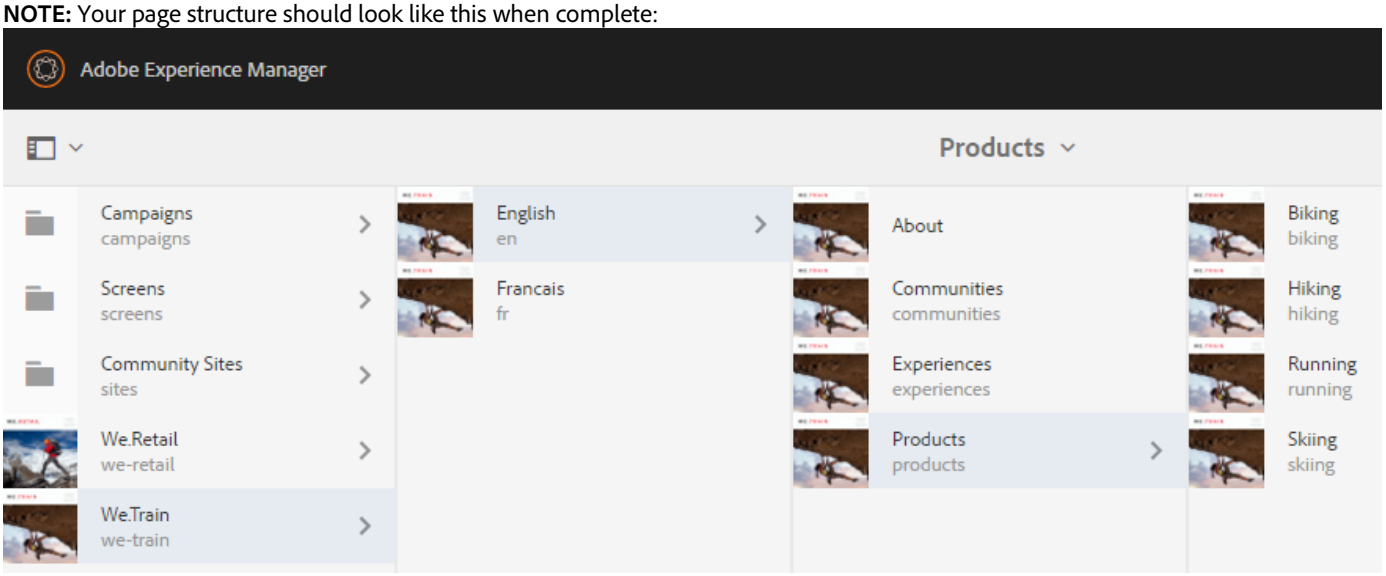
7. **Under the English root, create the following website page structure.**

   NOTE: Use a naming best practice by naming all your pages in *lowercase* and use hyphens for two or more words:

   en (Title: English) >

   - about (Title: About)
   - communities (Title: Communities)
   - experiences (TItle: Experiences)
   - products (TItle: Products)>
     - biking (Title: Biking)
     - hiking (Title: Hiking)
     - running (Title: Running)
     - skiing (Title: Skiing)

NOTE: Your page structure should look like this when complete:



7 INTRODUCTION TO HTL

Working with HTL

HTL (HTML Template Language) is a templating language developed by Adobe. It is the recommended language to use for developing components using Adobe Experience Manager. The best practice is to use HTML instead of JSP. The AEM Sites reference site, We.Retail, is built using HTL.

An HTL template defines an HTML output stream by specifying the presentation logic and the values to be dynamically inserted into the stream based on some background business logic. HTL differs from other templating systems in four main ways:

1. **HTL is HTML5:** A template created in HTL is a valid HTML5 file. All HTL-specific syntax is expressed within a data attribute or within HTML text. Any HTL file opened as HTML in an editor will automatically benefit from features such as auto-completion and syntax highlighting, which are provided by the editor for regular HTML.

```
<div class="HTL-example">
    <header data-sly-include="header.html"></header>
    <h1 data-sly-test="${properties.title}">${properties.title}</h1>
    <section data-sly-use-navigation="Navigation">
        <h1>
            ${navigation.breadcrumb}
        </h1>
    </section>
    <ul data-sly-list-child="${resource.listChildren}">
        <li>${child.name}</li>
    </ul>
</div>

```

2. **Separation of concerns (Web designer versus web developer):** The expressiveness of the HTL markup language is purposely limited so that only relatively simple presentation logic can be embedded in the actual markup. All complex business logic must be placed in an external helper class. The HTL's Use API defines the structure of the external helper.
3. **Secure by default:** HTL automatically filters and escapes all text being output to the presentation layer to prevent cross-site-scripting vulnerabilities.
4. Compatible with JSP or ESP.

HTL - Two Main Goals:

Goal 1: Simplified development ("Beautiful markup")

- ✓ Less code
- ✓ Cleaner code
- ✓ More intuitive code
- ✓ More efficient

Goal 2: Security

- ✓ Cross-site scripting protection
- ✓ Built-In
- ✓ Automatic
- ✓ Context-sensitive

Comparing HTL to JSP

HTL (less code, more secure)

```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

JSP (more code, less secure)

```
<a href="<%= xssAPI.getValidHref(properties.get("link", "#")) %>" <%
   String title = properties.get("jcr:title", "");
   if (title.length() > 0) {
      %>title="<%= xssAPI.encodeForHTMLAttr(title) %>"<%
   } %>>
   <%= xssAPI.encodeForHTML(properties.get("jcr:description", "")) %>
</a>
```

Both code blocks above achieve the same functional result.

HTL Syntax

There are two different types of syntaxes:

1. HTL Block Statements - To define structural elements within the template, HTL employs the HTML data attribute, which is HTML5 attribute syntax purposely intended for custom use by third-party applications. All HTL-specific attributes are prefixed with `data-sly-`.

2. HTL Expressions - HTL expressions are delimited by characters ${ and }. At runtime, these expressions are evaluated and their value is injected into the outgoing HTML stream. They can occur within the HTML text nodes or within the attribute values. In other words, you include HTL expressions within HTML tags.

The following is a list of a few of the basic HTL statements, expressions, and tags.

Comments (HTL comments are HTML comments with additional syntax. They are delimited like shown here.**)**
Example: `<!--/* An HTL Comment */-->`

Expressions
Examples: `${true}`, `${properties.text}`

URI Manipulation
Example: `${'example.com/path/page.html' @ scheme='http'}` (Resulting output: http://example.com/path/page.html)

Enumerable objects
Examples: `pageProperties`, `properties`, `inheritedPageProperties`

HTL Block Statements

- ✓ use: `<div data-sly-use.nav="navigation.js">${nav.foo}</div>`
- ✓ list: `<dl data-sly-list="${currentPage.listChildren}">`
- ✓ `data-sly-include, data-sly-repeat, etc.`

Special HTML tags
Example: `<sly>`

Expressions contain two types:

1. Literals

- Boolean: `${true} ${false}`
- Strings: `${'foo'} ${"answer"}`
  - NOTE: All of the global.jsp objects are available automatically!
- Positive integers: `${42}`
- Arrays: `${[42, true, 'Hello World']}`

2. Variables

`${myVar}`
Variables are accessed as: `${properties.myVar}`

Page-Rendering Scripts

When you create a component, a rendering script is also created by default. When you view a page, the output is displayed from the rendering script.

Adobe recommends you use HTL as the rendering script. Apart from using the HTL tags, you just need to change the extension of the script from .jsp to .html. As you will see in later exercises (starting in Module 10), you will be doing this task quite often in this course.

Use APIs to Display Basic Page Content

This section is an introduction to rendering the content from the repository, which is a similar concept to querying and displaying data from a database. In the repository, the nodes define the structure and the properties hold the data. To render content on any page, you need to render the data from those properties. Initially, start with the basic properties associated with every page.

Enumerable Objects and Java-Backed Objects: Summary

- Enumerable objects can be iterated-through using data-sly-list.
  - properties
  - pageProperties
  - inheritedPageProperties
- Java-backed objects
  - component – componentContext
  - currentDesign
  - currentPage
  - currentSession
  - request - response
  - resource – resourceDesign – resourcePage
  - wcmmode
  - etcetera…

Here are some of the ways that you can access the JCR in Adobe Experience Manager:

The **currentPage** object (Java-backed)
The currentPage object is an instance of the page (see AEM API) class, which provides some methods to access content. For example: `${currentPage.Title}`

The **properties** object (Enumerable)
The properties object is an instance of the ValueMap (see Sling API) class and contains all properties of the current resource. For example: `<p> Title : ${properties.jcr:title}</p>`

The **currentNode** object (Java-backed)
The currentNode object is an instance of the Node (see JCR API) class, which provides access to content using the getProperty() method. For example: `${currentNode.Name}`

Modularize Page Components

It is important to modularize a component into multiple scripts and include them at runtime—promoting component or script reuse. You use the HTL include tags to support modularization. There are different ways in which you can include a file to the script.

For example, you can include a file at runtime using HTL as: `<div data-sly-include=”myScript.html”/>`

HTL Resources

Adobe Documentation and Tutorials:
https://docs.adobe.com/docs/en/htl/overview.html and https://docs.adobe.com/docs/en/htl/docs/getting-started.html

HTL Spec

The HTML Template Language Specification GitHub Repository is available as a reference for all HTL attributes, behavior and syntax:
https://github.com/Adobe-Marketing-Cloud/htl-spec

https://github.com/Adobe-Marketing-Cloud/htl-spec/blob/master/SPECIFICATION.md

Exercise - I

7.1.1 Task – Render Basic Page Content

At this point, all of the pages in the We.Train website are identical. Now we will begin to use HTL to render the content residing in properties.

1. **Using CRXDE Lite, navigate to the /apps/training/components/structure/contentpage node.**
2. **Open contentpage.html.**
3. **Replace the existing code in CRXDE Lite with the code from the Exercise Files and save your changes.**

   NOTE: The code is provided as part of the Exercise_Files under **\Module 7 Introduction to HTL\Task - Render Basic Page Content**. Copy and paste the code from the exercise files to CRXDE Lite. Do not copy from this exercise book. The following is for illustration purposes only.

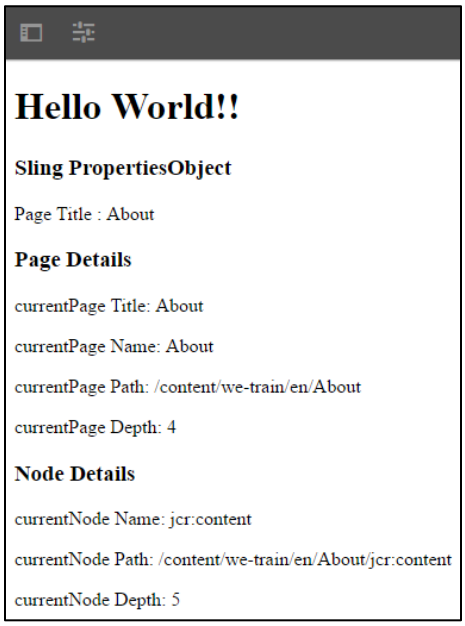
   NOTE: You may employ the method of replacing/populating contents of files from within CRXDE Lite described in **Exercise 5.1.2 (Create a page-rendering component)** if you like. Remember that this method may be used in all cases where you need to replace or populate new code in a file in CRXDE Lite (something you will be doing very often).

   contentpage.html

```
<!DOCTYPE html!>
<!--/* A simple HTL script */-->
<html>
       <head>
       <meta charset="utf-8"/>
    </head>
    <body>
       <h1>Hello World!!</h1>
        <h3>Sling PropertiesObject</h3>
                    <p>Page Title : ${properties.jcr:title}</p>

              <h3>Page Details</h3>
                    <p>currentPage Title: ${currentPage.Title}</p>
                    <p>currentPage Name: ${currentPage.Name}</p>
                    <p>currentPage Path: ${currentPage.Path}</p>
                    <p>currentPage Depth: ${currentPage.Depth}</p>

        <h3> Node Details </h3>
                    <p>currentNode Name: ${currentNode.Name}</p>
                    <p>currentNode Path: ${currentNode.Path}</p>
                    <p>currentNode Depth: ${currentNode.Depth}</p>
    </body>
</html>
```

4. **Using the Sites Console, navigate to Sites > We.Train > English and open the page for editing (HINT: Use the Edit button).**

5. **Examine the rendered content.**
6. **Open the About child page of English in Edit mode.**



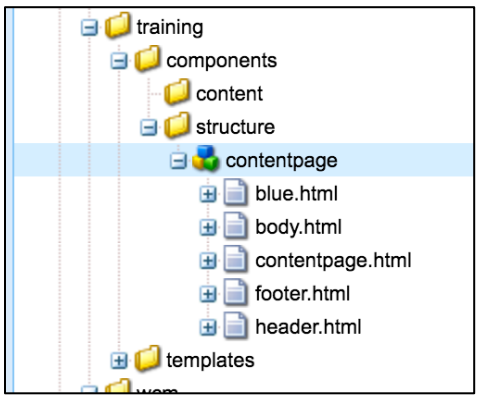
7. **Examine the rendered content. Notice the differences as compared to the English page. Each page is now unique because we are rendering (for each page) the properties associated with that page.**

What we have just done consists of the basics for page rendering. As we have mentioned before, in the repository, the nodes provide the structure and the properties hold the data. By rendering the property values associated with the page, we have rendered the page content.

7.1.2 Task – Modularize the contentpage Component

Modularization of components enables reuse of common structures and reduces code redundancy.

1. **Using CRXDE Lite, right-click /apps/training/components/structure/contentpage and select Create…> Create File.**
2. **Enter body.html for the file name. Save your changes.**
3. **Repeat steps 1 and 2 to create header.html and footer.html.**



4. **Examine the code for body.html, header.html, and footer.html.**

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12" data-sly-
include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-GridColumn--default-
-12" style="padding-top: 100px;">
                <div>Hero Component</div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--default--12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
      <div>Breadcrumb</div>
                            <div class="we-Header">Title Component</div>
                            <div>Responsive Content Area</div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--default--12" data-
sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

header.html

```
<div class="navbar navbar-inverse navbar-fixed-top hidden-xs">
    <div class="container-fluid">
        <nav style="color: white;" >Language Navigation</nav>
        <ul class="nav navbar-nav navbar-right" style="color: white;" >
            <sly>Toolbar</sly>
        </ul>
    </div>
</div>
<div >Site Navigation</div>
```

footer.html

```
<footer class="we-Footer width-full">
    <div class="container">
        <div class="row">
            <div class="row we-Footer-section we-Footer-section--sub">
                <div class="we-Footer-section-item">
                    <div class="we-Logo we-Logo--big">
                        we.<strong>train</strong>
                    </div>
                    <div class="we-Footer-nav">
                        <h2>Footer Toolbar</h2>
                    </div>
                </div>
            </div>
            <div class="row we-Footer-section we-Footer-section--sub">
                i18n Coded Content
            </div>
            <div class="row">
                <div class="col-md-12">
                    <div class="text-center">
                        <a href="#top" class="btn btn-primary">Back to the top</a>
                    </div>
                </div>
            </div>
        </div>
    </div>
</footer>
```

5. **Using the code from the Exercise_Files, paste the code for each of the three scripts into the three empty HTML files that you created.**

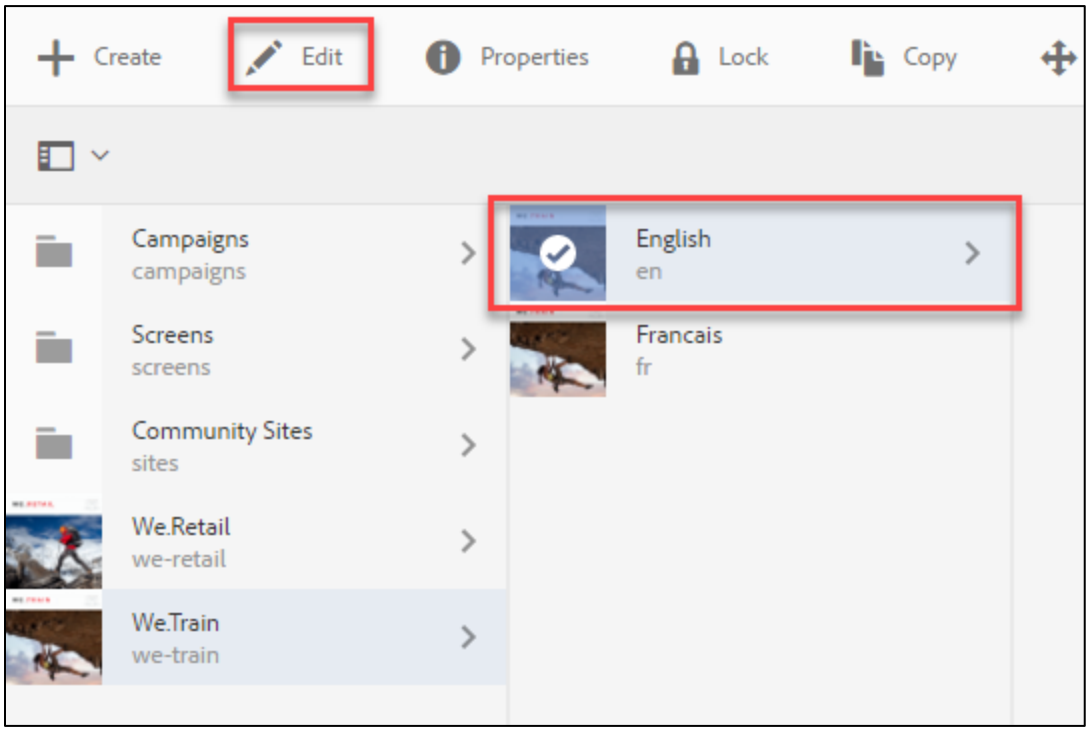
   NOTE: The code is provided as part of the Exercise_Files under **\Module 7 Introduction to HTL\Task - Modularize Contentpage Component**. Copy and paste the code from the exercise files to CRXDE Lite. Do not copy from this exercise book.

6. **Also, modify the contents of contentpage.html using the content from the Exercise_Files (same directory).**

contentpage.html

```
<!DOCTYPE html!>
<!--/* A simple HTL script */-->
<html>
 <head>
   <title>${properties.jcr:title}</title>
  </head>
  <body>
    <div data-sly-include="body.html"></div>
  </body>
</html>
```

7. **Using the Sites Console, open Sites > We.Train > English.**



It should look something like this:

Notice that there are placeholders for components and other elements that we will be implementing throughout the course. There is no design applied yet, so the class definitions that we referenced in the code are not active yet. We will define and apply the design later in the course.

8 INHERITANCE

Inheriting Foundation Components

Components can be given a hierarchical structure to implement the inheritance of script files, dialog boxes, and so on. Therefore, it is possible for a specific 'page' component (or any component) to inherit from a 'base' component. For example, allowing inheritance of a script file for a specific part of the page, for example, the <head> tag.

During this course, you will inherit a few of the foundation components such as:

- (Foundation) page component (Module 5)
- Responsive grid component (also known as a paragraph system component) (Module 11)

Components within AEM are subject to three different hierarchies:

1. Resource Type Hierarchy

This is used to extend components using the property **sling:resourceSuperType**. This enables the component to inherit from a 'base' component. For example, a text component will inherit various attributes from the foundation text component, including:

Scripts (resolved by Sling)

Dialog boxes

Descriptions (including thumbnail images, icons, and so on)

It is important to note that a local copy or instance of a component element (for example, body.html) will take precedence over an inherited element.

2. Container Hierarchy

This is used to populate configuration settings to the child component and is most commonly used in a responsive grid scenario. For example, configuration settings for the edit bar buttons, control set layout (edit bars, rollovers, and so on), and dialog box layout (inline, floating, and so on) can be defined on the parent component and propagated to the child components. Configuration settings (related to edit functionality) in cq:editConfig and cq:childEditConfig are propagated. These topics are covered in later modules.

3. Include Hierarchy

This is imposed at runtime by the sequence of includes. It is typically used by the designer, which in turn acts as the base for various design aspects of the rendering—including layout. Example: Information, CSS information, the available components in a responsive grid, and so on.

Exercise

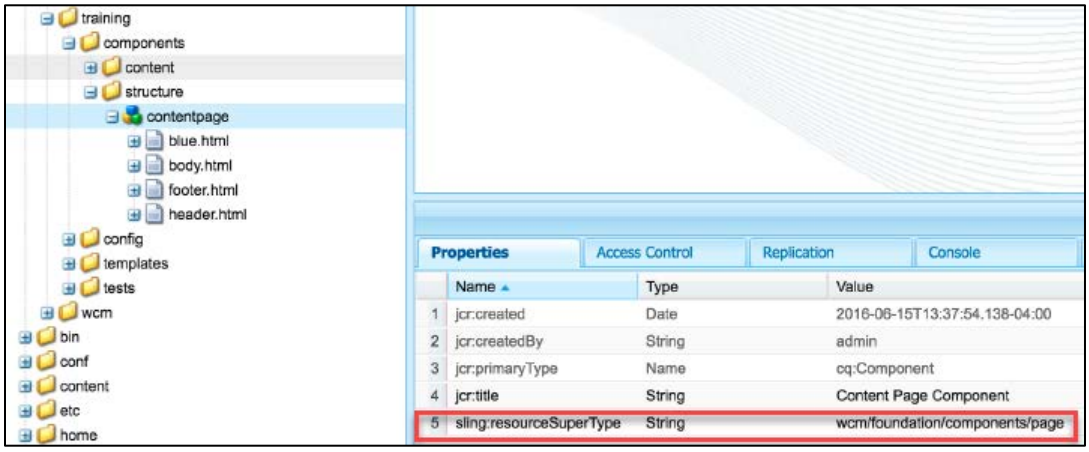
Review: When the /apps/training/components/structure/contentpage component was defined, we declared the Foundation Page component as the superType of our page-rendering component. By doing this, we allowed our contentpage component to initialize the WCM in order to take full advantage of the authoring environment. By declaring the Foundation Page component as its superType, the contentpage component will also inherit and make use properties, scripts, and other features of the Foundation Page component. Use of Foundation Page also ensures the Authoring UI is enabled.

8.1.1 Task – Investigate the contentpage sling:resourceSuperType property

1. **In CRXDE Lite, navigate to /apps/training/components/structure/contentpage.**

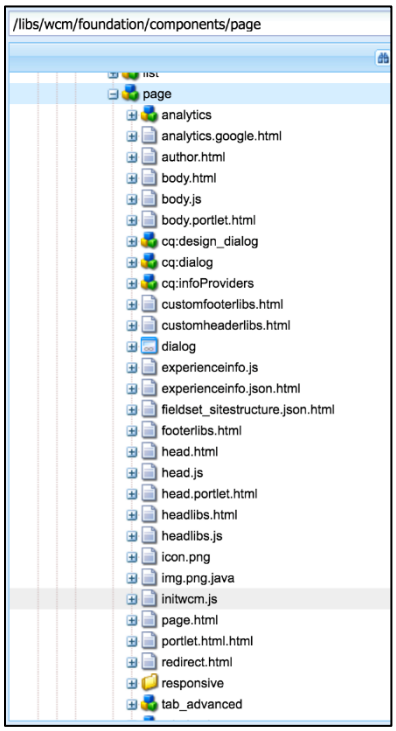
   Notice the sling:resourceSuperType property.

   Defining the sling:resourceSuperType property enables inheritance from a super type or, as it is sometimes called, a base type. Once you define the sling:resourceSuperType property with the value of wcm/foundation/components/page, the /apps/training/components/structure/contentpage component may now inherit scripts, properties, dialog box structures, and other elements from the Foundation Page component.



8.1.2 Task – Investigate the Foundation Page Component

1. **Using CRXDE Lite, navigate to /libs/wcm/foundation/components/page. You will note that the Foundation page component contains all the pieces necessary to be a complete page-rendering component.**



2. **Take particular notice of the default scripts: page.html and head.html:**

page.html

```
…
<head data-sly-include="head.html"></head>
    <body data-sly-use.body="body.js"
class="${body.cssClasses}"
            data-sly-include="body.html"></body>
…
```

head.html

```
…
<meta http-equiv="content-type" content="text/html; charset=UTF-8"
          data-sly-use.head="head.js"
          data-sly-use.headlibRenderer="headlibs.html"
          data-sly-
use.clientLib="/libs/granite/sightly/templates/clientlib.html"/>
    <meta data-sly-test.keywords="${head.keywords}" name="keywords"
content="${keywords}"/>
    <meta data-sly-test.description="${head.description}"
name="description" content="${description}"/>
    <sly data-sly-call="${headlibRenderer.headlibs @
designPath=head.designPath}" />
    <sly data-sly-include="author.html" />
    <sly data-sly-
include="/libs/cq/cloudserviceconfigs/components/servicelibs/servicelibs.j
sp" />
```

```
    <sly data-sly-include="customheaderlibs.html" />
    <sly data-sly-test.faviconpath="${head.faviconPath}">
    <link rel="icon" type="image/vnd.microsoft.icon"
href="${faviconpath}">
    <link rel="shortcut icon" type="image/vnd.microsoft.icon"
href="${faviconpath}">
    </sly>
    <title>${head.title}</title>
```

8.1.3 Task – Delete the contentpage.html script

1. **Using CRXDE Lite, delete /apps/training/components/structure/contentpage/contentpage.html script by right-clicking it and selecting Delete.**
2. **Save changes.**
3. **Using the Sites Console, open Sites > We.Train > English**

Notice that the page renders just fine, even without the default script, contentpage.html. The proper rendering of the page is possible because the contentpage component is inheriting several scripts from the Foundation Page component.

8.1.4 (Optional) Extra Credit Task – Build the rendering chain

Use CRXDE Lite to follow the Sling URL decomposition and resource resolution process.

Use CRXDE Lite to find the initial rendering scripts and build the rendering chain for the **Sites** > **We.Train** > **English** page.

9 DESIGN AND STYLES

Adding a Design to your Site

Designing your site involves creating a package of CSS files, client libraries, images, fonts, and anything else that provides the overall look and feel of your site. This helps with consistency and adherence to branding. Creating and assigning a design(er) in Adobe Experience Manager allows you to enforce a consistent look and feel across your website, as well as share global content. This is commonly referred to as a "Designer" in AEM, though it is equivalent to a site's design.

The pages that use the same design(er) will have access to common CSS files, defining the formats of specific areas or components, and images that you use for features such as backgrounds and buttons.

In Adobe Experience Manager, designs are stored in the /etc/designs folder.

Web Accessibility Guidelines

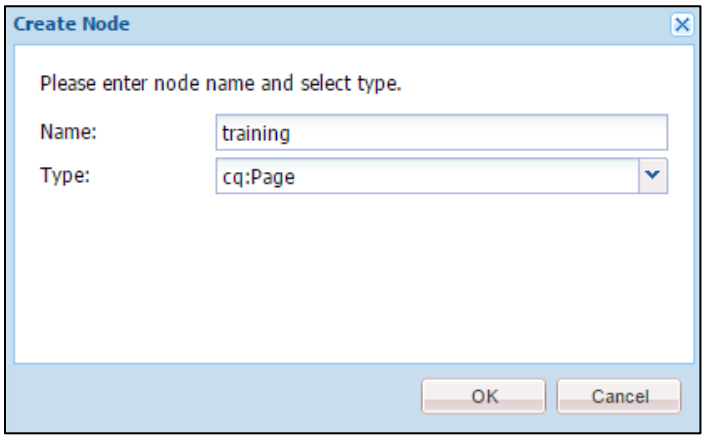
Adobe Experience Manager was developed to maximize compliance with Web Accessibility Guidelines. Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the web, and they can contribute to the web. This can include measures such as providing textual alternatives to images or any non-text item. These can then be used to help people with sight impairment by outputting the text on a Braille keypad, or by using a text to speech program or utility. Such measures can also benefit people with slow Internet connections, or any Internet user—when the measures offer the user more information.

These mechanisms must be carefully planned and designed to ensure that they provide the information required for the user to successfully understand and use the content. Certain aspects are integral to Adobe Experience Manager, whereas other aspects must be realized during your project development.

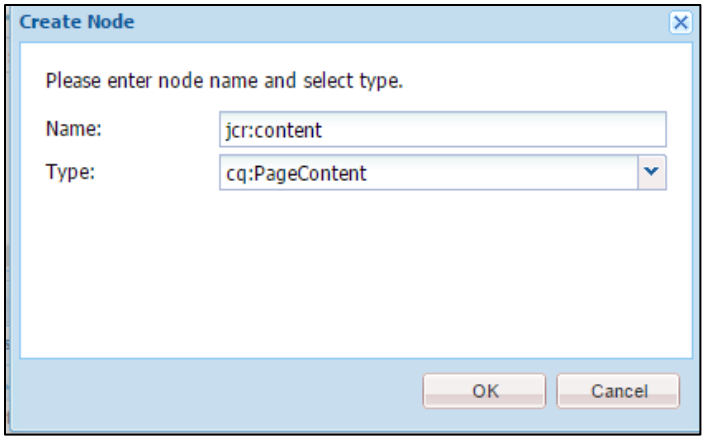
Exercise

9.1.1 Task - Define a design and use clientlibs

1. **Using CRXDE Lite, navigate to /etc/designs.**
2. **Create a node named training of type cq:Page:**

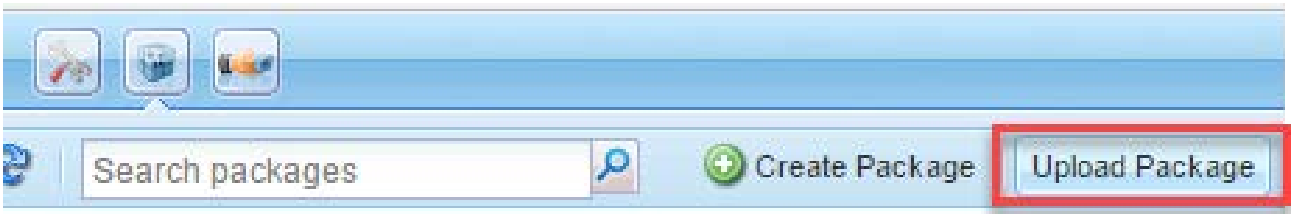


3. In your newly created **training** node, create a child node named **jcr:content** of type **cq:PageContent:**



The client libraries that make up the look and feel of the site will be stored here (/etc/designs/training). Due to time constraints and for training purposes, we will upload an already created design.

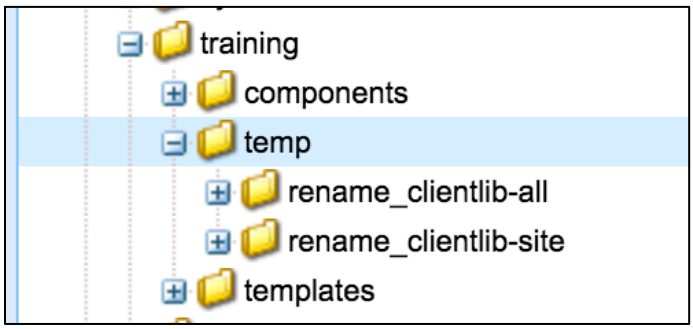
4. **Enter http://localhost:4502/crx/packmgr/index.jsp or click on the package icon in the toolbar on the CRXDE Lite page.**
5. **Click on Upload Package.**



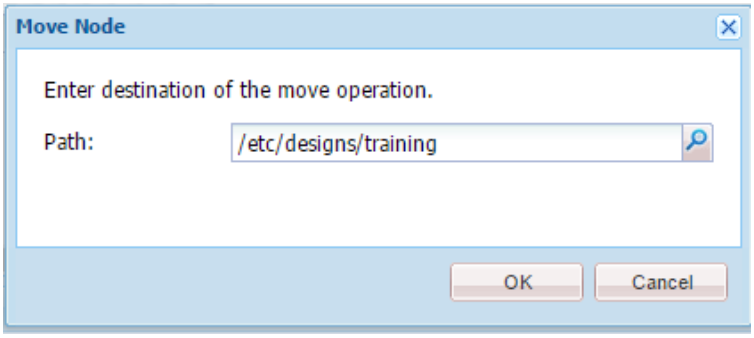
6. **In the file upload dialog box, browse to the Exercise_Files directory for this module (/Module 9 Design and Styles/Task – Define Design), and select we.train-design-package.zip.**
7. **Click OK and then Install the package. There is no need to force upload.**

Let's check out what happened in the repository.

8. **To navigate back to CRXDE Lite, click the Develop icon in the top toolbar and navigate to /apps/training/temp. You will notice two temporary client library directories that will make up our design:**



9. **Right click on each client library, and select Move.**
10. **Click the search icon, and select the path as /etc/designs/training. Save changes.**

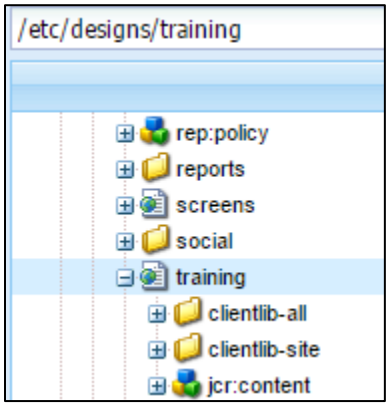


11. **Right-click on each client library, rename each by removing "rename_" from the start, and save changes.**

rename_clientlib-site becomes **clientlib-site**

rename_clientlib-all becomes **clientlib-all**

Your newly created node structure under **/etc/designs** should look like this; notice the **training** node and **jcr:content** node you created:

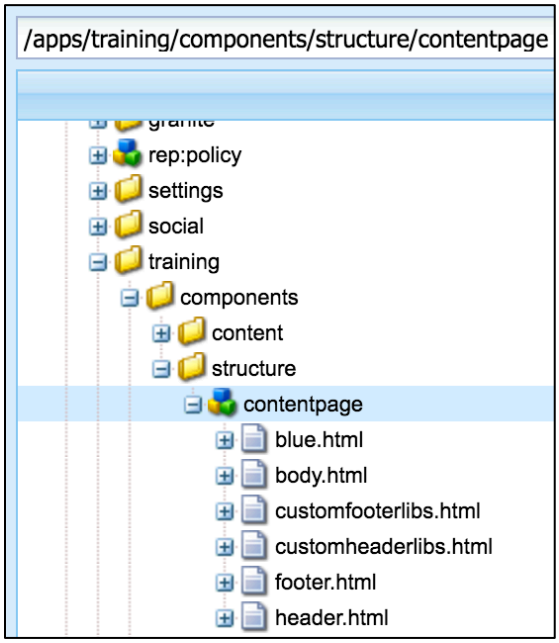


Now we have a design defined. The association of the design with the We.Train website will be achieved in an exercise in the next module (Module 10 – Exercise V).

9.1.2 Task – Modify the contentpage component to call in the design

1. **Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.**
2. **Create two new files here in the contentpage component:**
   - customfooterlibs.html
   - customheaderlibs.html

Your node should now look like this:



3. **Using the code from the Exercise_Files for this module, enter or paste in the contents of customfooterlibs.html and customheaderlibs.html. Do not copy from this exercise book. The following is for illustration purposes only.**

customfooterlibs.html

```
<!--/* Include the site client libraries (loading only the JS in the footer, CSS
was loaded in the header) */-->

<sly data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html" data-
sly-call="${clientLib.js @ categories='we.train.all'}"/>
```

customheaderlibs.html

```
<!--/* Include the site client libraries (loading only the CSS in the header, JS
will be loaded in the footer) */-->

<sly data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html" data-
sly-call="${clientLib.css @ categories='we.train.all'}" />
```

Notice that, as suggested by best practices, the *.css files are called in at the top of the page (header) and the JavaScript files are called in at the end of the page (footer).

4. **Using the code from the Exercise_Files, modify footer.html to call the customfooterlibs.html. Do not copy from this exercise book. The following is for illustration purposes only.**

Alternatively, you could add one line of HTL to the existing footer.html (line 28) as the third to the last line in the HTML file:

<sly data-sly-include="customfooterlibs.html" />

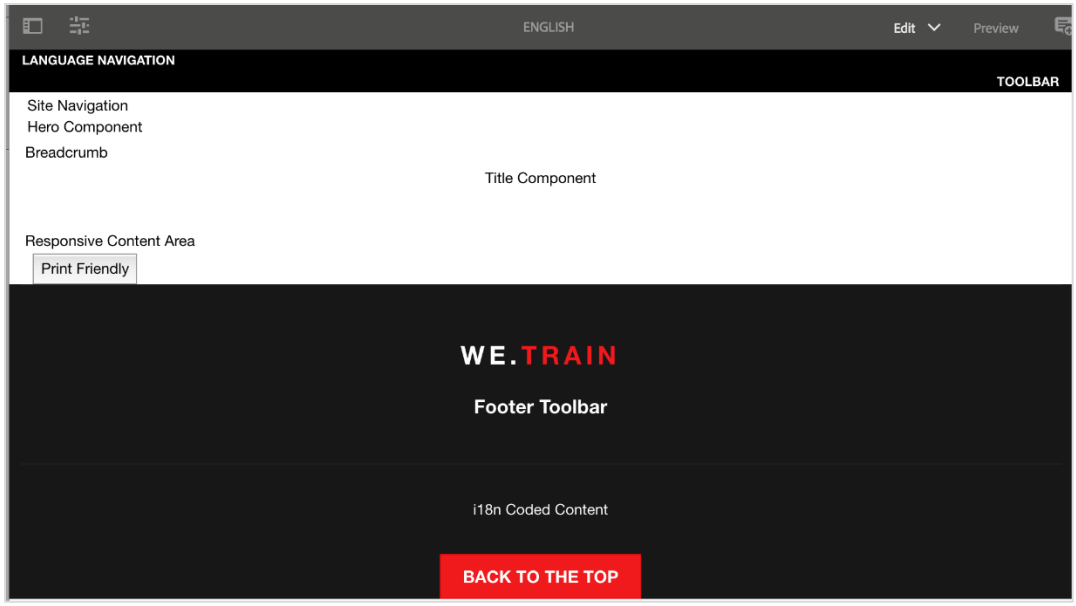
footer.html

```
<footer class="we-Footer width-full">
    <div class="container">

        <div class="row">
            <div class="row we-Footer-section we-Footer-section--sub">
                <div class="we-Footer-section-item">
                    <div class="we-Logo we-Logo--big">
                        we.<strong>train</strong>
                    </div>
                    <div class="we-Footer-nav">
                        <h2>Footer Toolbar</h2>
                    </div>
                </div>
            </div>

            <div class="row we-Footer-section we-Footer-section--sub">
                i18n Coded Content
            </div>

            <div class="row">
                <div class="col-md-12">
                    <div class="text-center">
                        <a href="#top" class="btn btn-primary">Back to the top</a>
                    </div>
                </div>
            </div>
        </div>
          <sly data-sly-include="customfooterlibs.html" />
    </div>
</footer>
```

Now let's see the result.

Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.

Open the **English** page in Edit mode. It should look like this now:



Notice that the English page is now using clientlibs (or "client libraries") to apply the design and the header and footer are now more functional and look better. This topic is discussed later in this course (Module 14 – Content Components).

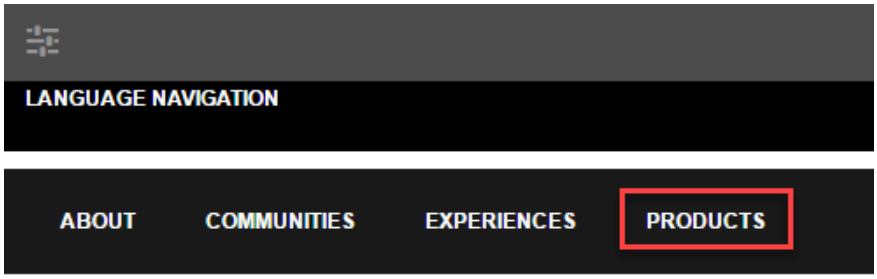
10 DEVELOPING STRUCTURE COMPONENTS

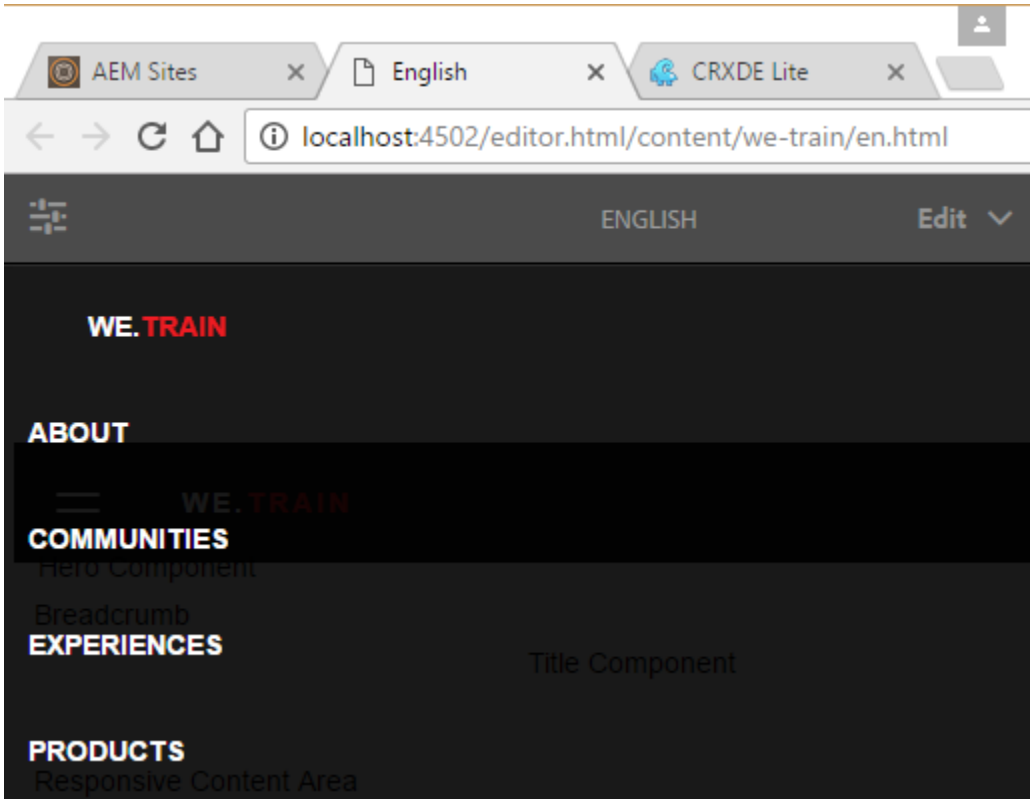
Structure Components: Guidelines

- Authors cannot remove structure components
- Authors can edit and confgure structure components
- Authors CAN remove content components

Navigation Component: Example

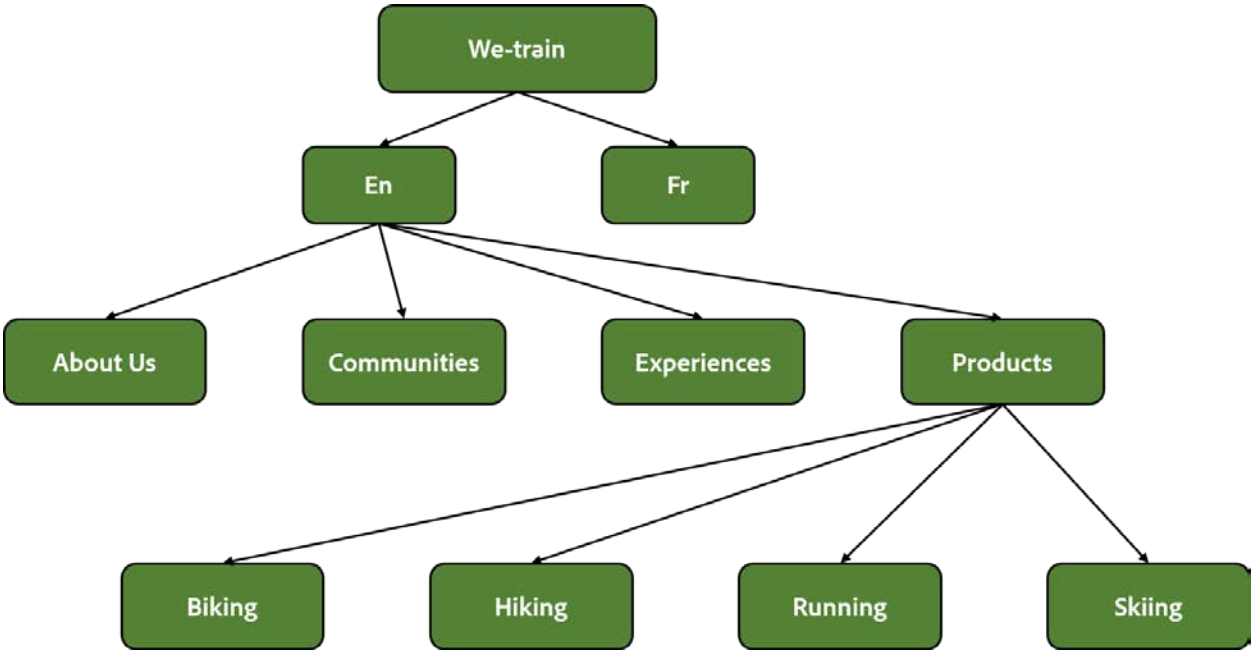
The navigation component is a responsive horizontal menu at the top of the page that provides links to navigate to a specific page in the site.





Creating a Top Navigation Component using HTL

To demonstrate the component creation process, this module's exercises will involve creating a dynamic text-based navigation component, allowing for the website structure to be easily modified, represented, and navigated in real time. The following image shows the exercises "We.Train" example simple website structure that has four levels:



For example, in the above structure, if you were in the Products page, the navigation component would list Biking, Hiking, Running, and Skiing. However, for a page with no child pages, the navigation component would be blank. For example, navigating to the About Us page would have no pages listed in the navigation component, as it has no child pages. To overcome this issue, you could have a component that always lists the child pages of the root page.

Let's see how you can create a navigation component, and then see how to optimize that component.

How Do I Create Dynamic Navigation?

Providing dynamic navigation capabilities and allowing for the easy addition and removal of pages is one of the most important (and sometimes difficult) tasks you can do as a developer in Adobe Experience Manager.

You can include a component from within the script using the data-sly-resource tag. The following code includes an HTL component named *topnav* to the script. The resourceType provides the location of the component. It also provides the name of the component that appears in Design mode.

```
<div data-sly-resource="${'topnav'@ resourceType='training/components/topnav'}"></div>
```

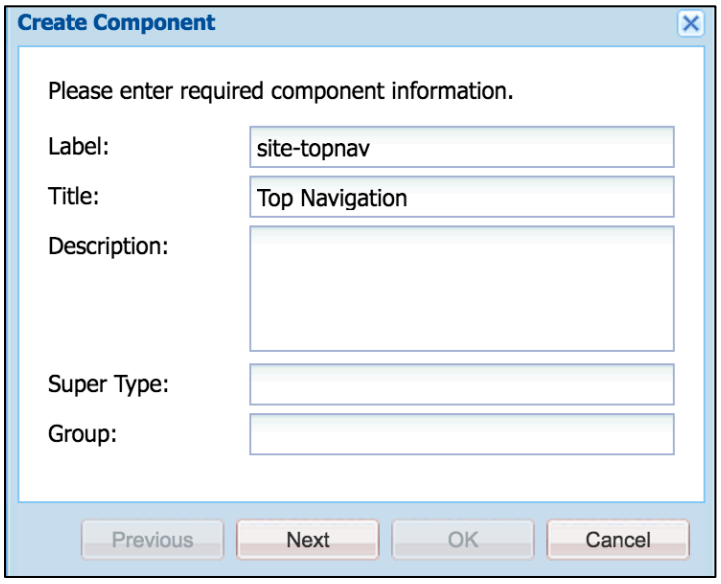
The following code snippet is used to extract the child pages of the current page:

```
<ul class="topnav" data-sly-list="${currentPage.listChildren}">
<li><a href="${item.path}.html">${item.title}</a></li>
</ul>
```

To get child pages of the root page, add a JavaScript file with the method: `currentPage.getAbsoluteParent()`

10.1.1 Task – Create a simple Top Navigation component

1. **Using CRXDE Lite, navigate to /apps/training/components/structure.**
2. **Right-click and select Create… > Create Component.**
3. **Enter the following values in the dialog:**
    - Label: site-topnav
    - Title: Top Navigation



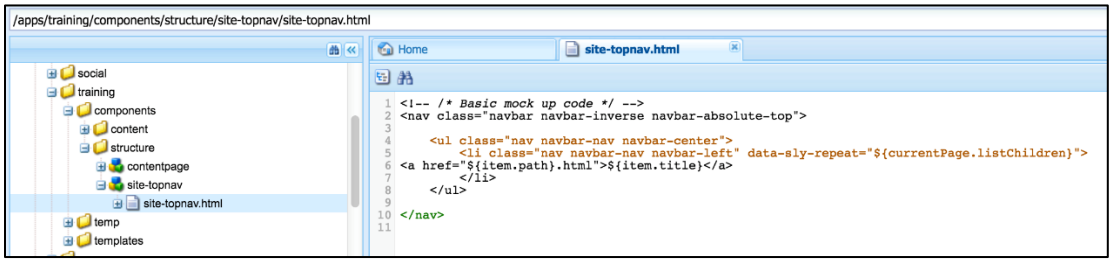
4. **Click Next. Click OK.**
5. **Save your changes.**
6. **Rename the default script to site-topnav.html.**
7. **Using the code from the Exercise_Files for this module, enter or paste in the contents of site-topnav.html. Do not copy from this exercise book. The following is for illustration purposes only.**

site-topnav.html

```
<!-- /* Basic mock up code */ -->
<nav class="navbar navbar-inverse navbar-absolute-top">
      <ul class="nav navbar-nav navbar-center">
            <li class="nav navbar-nav navbar-left" data-sly-
            repeat="${currentPage.listChildren}">
                  <a href="${item.path}.html">${item.title}</a>
            </li>
      </ul>
</nav>
```

You will notice that the code provides a simple navigation that just provides a list of the children of the current page.

8. **Save the changes. Your node structure should look like this:**



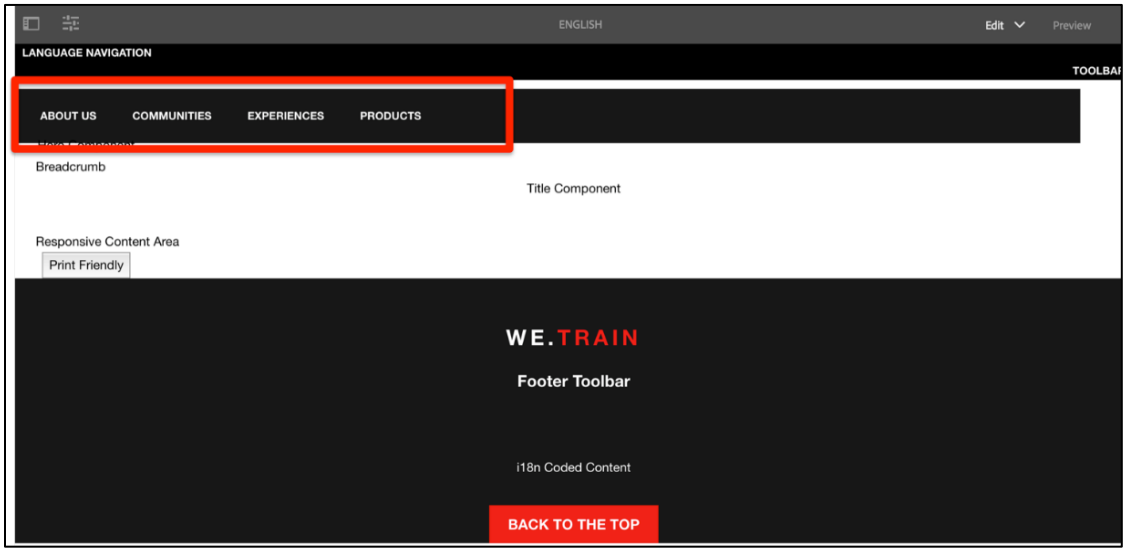
9. **Navigate up to /apps/training/structure/contentpage.**
10. **Using the code from the Exercise_Files for this module, enter or paste in the contents of header.html. Do not copy from this exercise book. The following is for illustration purposes only.**

header.html

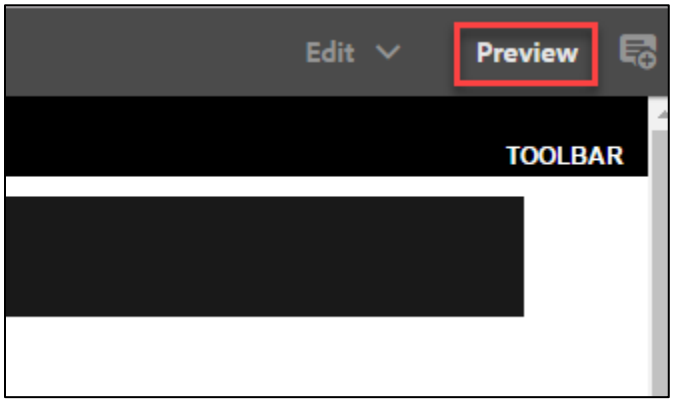
```
<div class="navbar navbar-inverse navbar-fixed-top hidden-xs">
       <div class="container-fluid">
              <nav style="color: white;" >Language Navigation</nav>
              <ul class="nav navbar-nav navbar-right" style="color: white;" >
                     <sly>Toolbar</sly>
              </ul>
       </div>
</div>
<div data-sly-resource="${'site-topnav' @
resourceType='training/components/structure/site-topnav'}"></div>
```

NOTE: The purpose of the HTL code here is to call the Top Navigation component. Technically, the only thing altered here was the last tag.

11. **Using the Sites Console, open the Sites > We.Train > English page in edit mode. Notice the new Top Navigation element:**

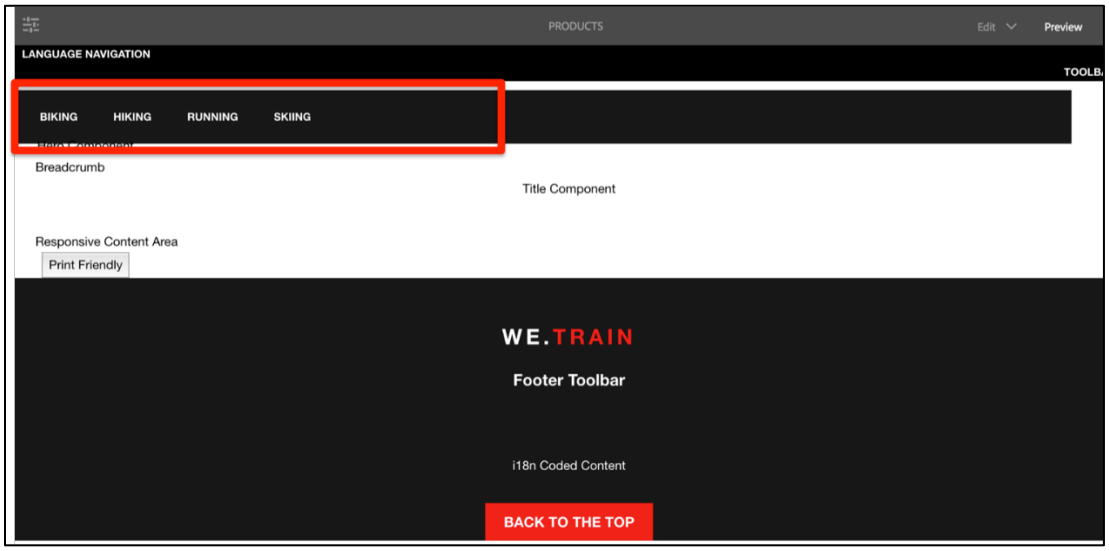


12. **To facilitate testing, switch to Preview mode in the upper right:**



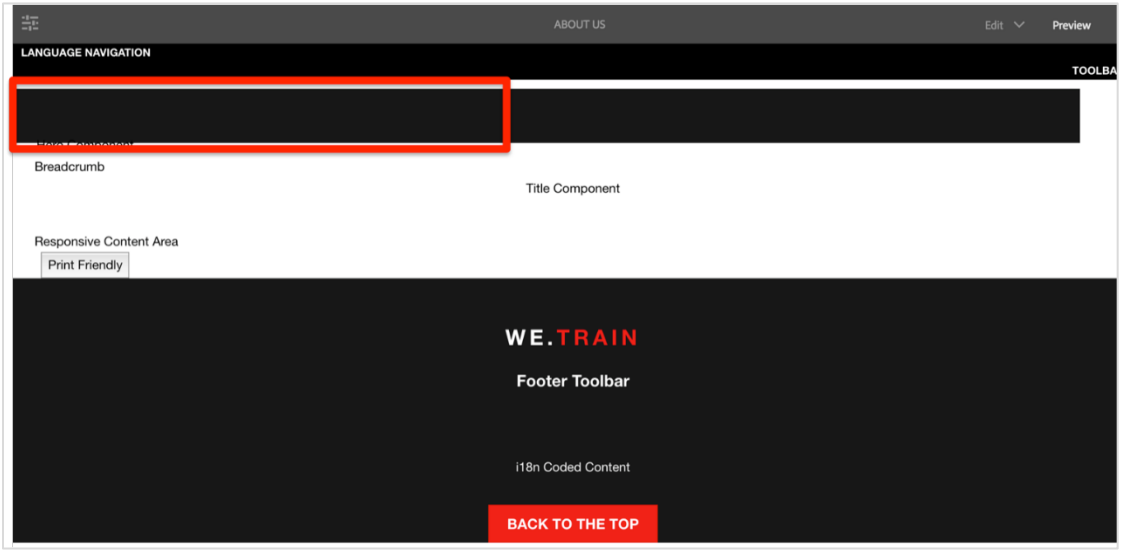
13. **In your newly created Top Navigation, click Products.**

Notice how the contents of your Top navigation has changed:



NOTE: This is probably not what we really want for a top navigation which should not typically change as we navigate through the site.

14. **Navigate to Sites > We.Train > English > About. Notice that for pages with no children, the navigation menu is empty.**



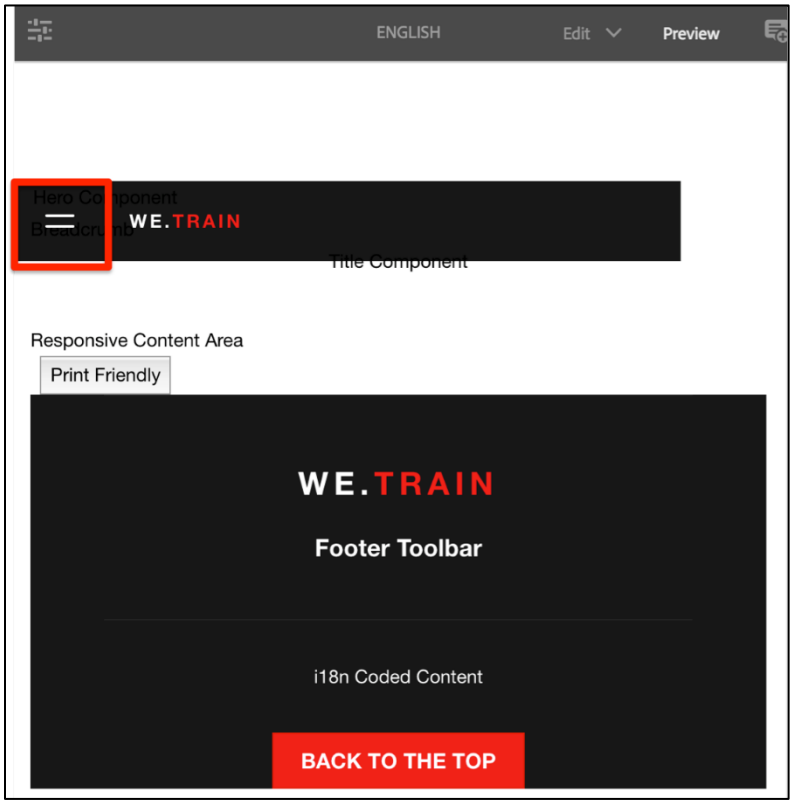
10.1.2 Task – Make the Navigation Component responsive

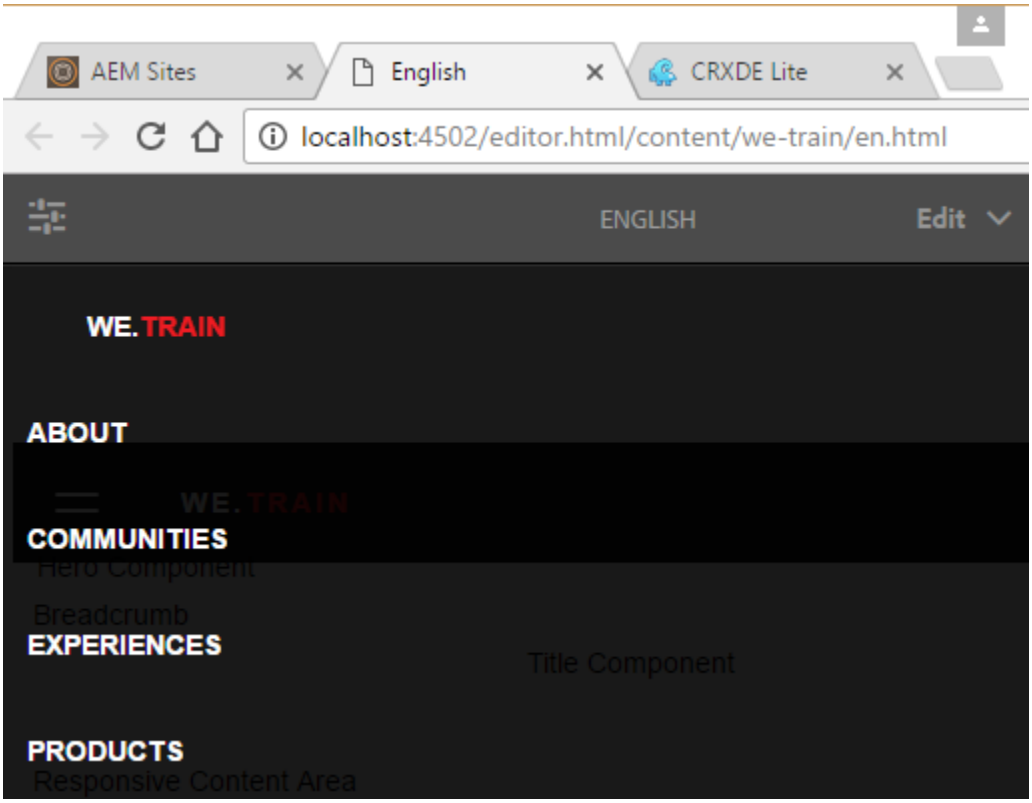
1. **Using the code from the Exercise_Files for this module, paste in the new contents of site-topnav.html. Do not copy from this exercise book. The following is for illustration purposes only.**

site-topnav.html

```
<!-- /* Add the full responsive design */ -->
<div class="container we-Container--top-navbar">
<nav class="navbar navbar-inverse navbar-absolute-top">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#we-example-navbar-collapse-inverse" aria-expanded="false">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span> <span class="icon-bar"></span>
</button>
<button type="button" class="navbar-toggle navbar-toggle-close collapsed" data-
toggle="collapse" data-target="#we-example-navbar-collapse-inverse" aria-
expanded="false">
<span class="sr-only">Toggle navigation</span>
</button>
<a class="navbar-brand" href="#">we.<strong>train</strong></a>
<div class="pull-right visible-xs"></div>
</div>
<!-- /.navbar-header -->
<div class="collapse navbar-collapse width" id="we-example-navbar-collapse-inverse">
<ul class="nav navbar-nav navbar-center">
<li class="visible-xs"><a href="#">we.<strong class="text-
primary">train</strong></a></li>
<!-- /* Basic mock up code */ -->
<li class="nav navbar-nav navbar-left" data-sly-
repeat="${currentPage.listChildren}">
<a href="${item.path}.html">${item.title}</a>
</li>
<li class="visible-xs divider" role="separator"></li>
</ul>
</div>
<span style="height: 0px;" class="navbar-shutter"></span>
</nav>
<!-- /.navbar -->
</div>
```

2. **Open Sites > We.Train > English and reduce the size of the browser window. Notice that the menu changes to become a list.**





So now we have a responsive navigation menu, but it is still just a simple list of the children of the current page. It still changes on every page and displays no items when the page has no children.

10.1.3 Task – Create a complex Navigation component with a Java Helper

Typically, the top navigation is the same on all pages. So let's create a top navigation component that is more typical. For this task, we will be using a Java helper servlet to perform the business logic of constructing a navigation menu.

NOTE: The Java class that we will be implementing will be local to the site-topnav component. In a production deployment of AEM, that Java class would typically be deployed in an application bundle.

1. **Create a file named TopNav.java under /apps/training/components/structure/site-topnav.**
2. **Using the code from the Exercise_Files for this module, paste in the contents of TopNav.java. Do not copy from this exercise book. The following is for illustration purposes only.**

TopNav.java

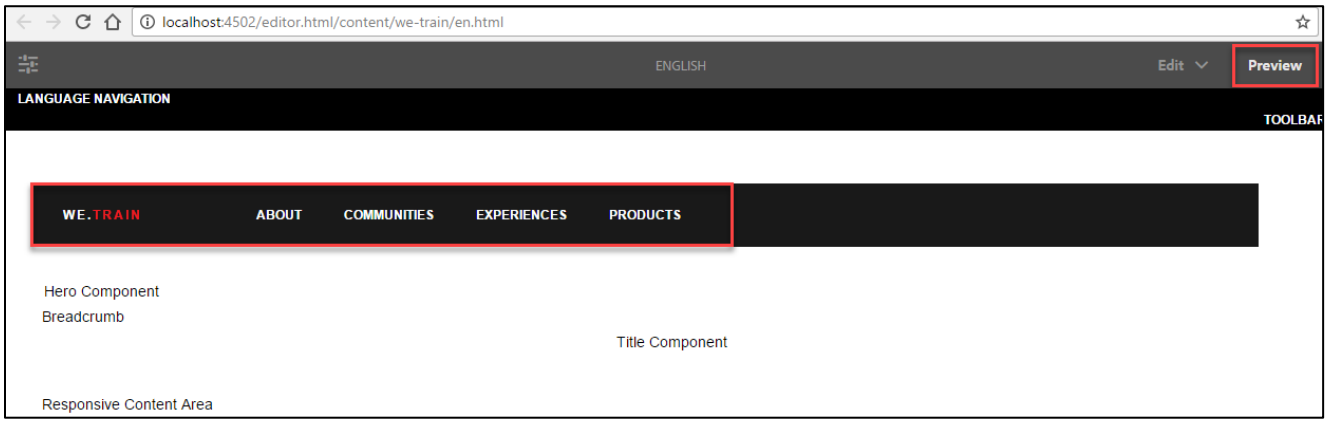
```
package apps.training.components.structure.site_topnav;
import java.util.*;
import java.util.Iterator;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageFilter;
import com.adobe.cq.sightly.WCMUsePojo;
public class TopNav extends WCMUsePojo{
private List<Page> items = new ArrayList<Page>();
private Page rootPage;
// Initializes the navigation
@Override
public void activate() throws Exception {
rootPage = getCurrentPage().getAbsoluteParent(2);
if (rootPage == null) {
rootPage = getCurrentPage();
}
Iterator<Page> childPages = rootPage.listChildren(new PageFilter(getRequest()));
while (childPages.hasNext()) {
items.add(childPages.next());
}
}
// Returns the navigation items
public List<Page> getItems() {
return items;
}
// Returns the navigation root
public Page getRoot() {
return rootPage;
}
}
```

3. **Using the code from the Exercise_Files for this module, paste in the contents of site-topnav.html. Do not copy from this exercise book. The following is for illustration purposes only. Notice the inclusion in bold of the TopNav class. Now site-topnav.html uses TopNav.java to perform the business logic.**

site-topnav.html

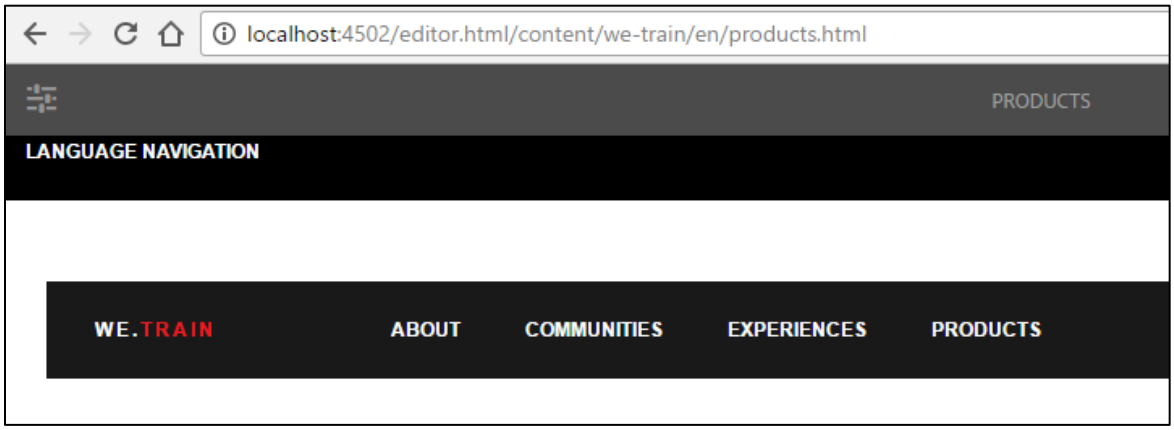
```
<!-- /* Add the business logic*/ -->
<div data-sly-use.topnav="TopNav" class="container we-Container--top-navbar">
<nav class="navbar navbar-inverse navbar-absolute-top">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#we-example-navbar-collapse-inverse" aria-expanded="false">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span> <span class="icon-bar"></span>
</button>
<button type="button" class="navbar-toggle navbar-toggle-close collapsed" data-
toggle="collapse" data-target="#we-example-navbar-collapse-inverse" aria-
expanded="false">
<span class="sr-only">Toggle navigation</span>
</button>
<a class="navbar-brand"
href="${topnav.root.path}.html">we.<strong>train</strong></a>
<div class="pull-right visible-xs"></div>
</div>
<!-- /.navbar-header -->
<div class="collapse navbar-collapse width" id="we-example-navbar-collapse-
inverse">
<ul class="nav navbar-nav navbar-center">
<li class="visible-xs"><a href="${topnav.root.path}.html">we.<strong class="text-
primary">train</strong></a></li>
<!-- /* Nav with business logic */ -->
<li class="nav navbar-nav navbar-left" data-sly-repeat="${topnav.items}">
<a href="${item.path}.html">${item.title}</a>
</li>
<li class="visible-xs divider" role="separator"></li>
</ul>
</div>
<span style="height: 0px;" class="navbar-shutter"></span>
</nav>
<!-- /.navbar -->
</div>
```

4. **Using the Sites Console, navigate to Sites > We.Train > English. Open the page in Preview mode:**



5. **Notice the navigation menu.**

6. Click **Products**. Notice that the Navigation menu values are **unchanged:**



7. Click other pages such as Communities, Experiences, and so on. Notice that the Navigation menu values are **unchanged**. Now the top navigation menu is the same on all pages.

10.1.4 Extra Credit task

1. **Why does the navigation menu show the children of the English page (level 2)?**
2. **Navigate to Sites > We.Train and open the page. Why does the navigation menu show the children of We.Train (level 1)?**

10.1.5 Task – Create a complex Navigation component with a JavaScript Helper

As we discussed, there are two ways to provide more complex business logic than possible with HTML alone. We already explored one way by using Java helpers. For this task, we will be using a JavaScript helper to perform the business logic of constructing a navigation menu.

1. **Using CRXDE Lite, create a file named topnav.js under /apps/training/components/structure/site-topnav.**
2. **Using the code from the Exercise_Files for this module, paste in the contents of topnav.js. Do not copy from this exercise book. The following is for illustration purposes only.**

topnav.js

```
// Server-side JavaScript for the topnav logic
use(function () {
var items = [];
var root = currentPage.getAbsoluteParent(2);
//make sure that we always have a valid set of returned items
//if navigation root is null, use the currentPage as the navigation root
if(root == null){
root = currentPage;
}
var it = root.listChildren(new Packages.com.day.cq.wcm.api.PageFilter());
while (it.hasNext()) {
var page = it.next();
items.push(page);
}
return {
items: items,
root: root
};
});
```

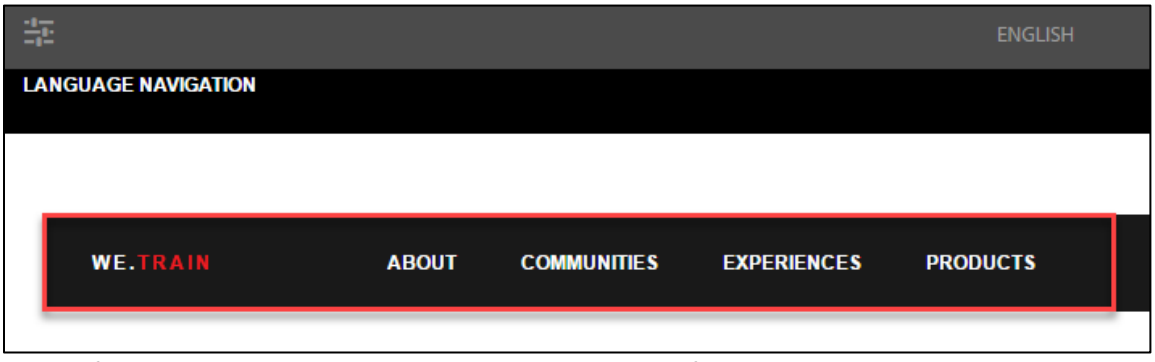
3. **Alter *one line of code* in site-topnav.html (shown in bold below). All that is changing in the HTML is the use of topnav.js instead of TopNav.java. Now site-topnav.html uses topnav.js to perform the business logic.**

site-topnav.html

```
<!-- /* Add the business logic*/ -->
<div data-sly-use.topnav="topnav.js" class="container we-Container--top-navbar">
    <nav class="navbar navbar-inverse navbar-absolute-top">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#we-example-navbar-collapse-inverse" aria-
expanded="false">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span> <span class="icon-bar"></span>
            </button>
```

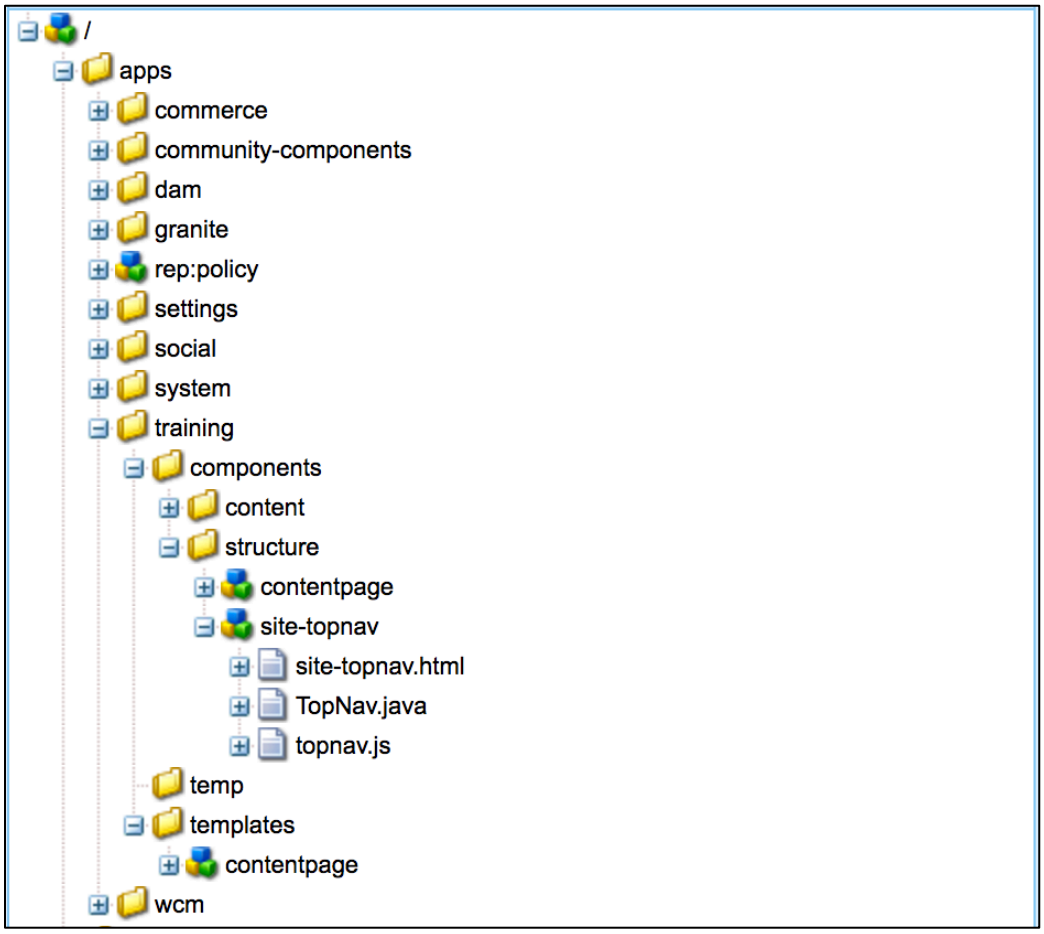
```
            <button type="button" class="navbar-toggle navbar-toggle-close
collapsed" data-toggle="collapse" data-target="#we-example-navbar-collapse-
inverse" aria-expanded="false">
                <span class="sr-only">Toggle navigation</span>
            </button>
            <a class="navbar-brand"
href="${topnav.root.path}.html">we.<strong>train</strong></a>
            <div class="pull-right visible-xs"></div>
        </div>
        <!-- /.navbar-header -->
        <div class="collapse navbar-collapse width" id="we-example-navbar-
collapse-inverse">
            <ul class="nav navbar-nav navbar-center">
                <li class="visible-xs"><a
href="${topnav.root.path}.html">we.<strong class="text-
primary">train</strong></a></li>
                <!-- /* Nav with business logic */ -->
                <li class="nav navbar-nav navbar-left" data-sly-
repeat="${topnav.items}">
                    <a href="${item.page.path}.html">${item.page.name}</a>
                </li>
                <li class="visible-xs divider" role="separator"></li>
            </ul>
        </div>
        <span style="height: 0px;" class="navbar-shutter"></span>
    </nav>
    <!-- /.navbar -->
</div>
```

4. **Using the Sites Console, navigate to Sites > We.Train > English. Notice the navigation menu is the same as when we used the Java helper:**



5. **Verify that the navigation menu is the same on all children of English.**

FYI: Your node structure should look like this:



Logging

Adobe Experience Manager offers you the possibility to configure:

- global parameters for the central logging service.
- request data logging (a specialized logging configuration for request information).
- specific settings for the individual services -- for example, an individual log file and format for the log messages.

Some Adobe Experience Manager log files provide detailed information about the current system state. In addition to the default system log files, you can also create and customize your own log files. They can help you better track messages produced by your own applications and separate them from the default log entries.

Adding log messages to a component script will allow you to easily debug various scripts you might be working on. In the daily life of a developer, it is often crucial to monitor the values of variables assigned or used. There are several possibilities, in various usability levels. Adobe Experience Manager and CRXDE Lite make your life a little easier by implementing the popular Log4j framework, which is designed to provide an easy-to-use logging solution. The initialization of a Logger object, called *log*, has already been accomplished during the inclusion of global.jsp in whatever component you may be working on, as well as initialized automatically in HTL. The log file entries are formatted according to the Sling configuration.

Two pieces of information are required to append an entry to the log file:

1. **log level**: This is provided by the corresponding method call. For example, a log.*debug*(<message>) produces a message with a log level of *debug* while a log.*info*(<message>) produces a message with log level of *info*.

Possible methods (logging levels) of the Logger object include (from most verbose to least verbose):

- trace()
- debug()
- info()
- warn()
- error()

2. **message:** The message itself is provided as a parameter to the method call. For example:

log.debug("This is the log message");

This appends the message "This is the log message," with a log level of "debug" to the error.log file.

Log files are located on the server at:

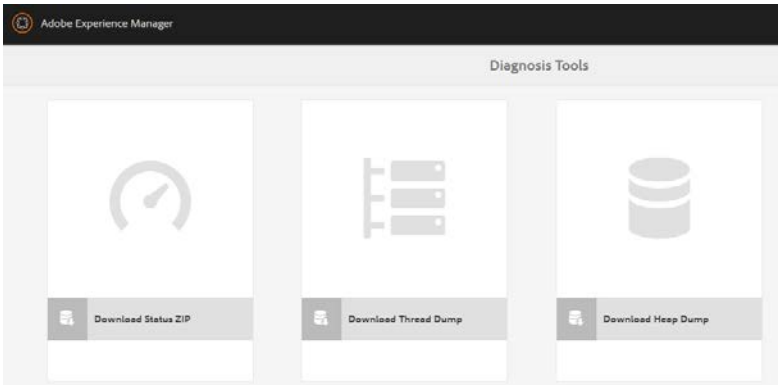
.../crx-quickstart/logs/error.log

.../crx-quickstart/logs/request.log

.../crx-quickstart/logs/access.log

You can download log files remotely from the server:

1. Go to Tools > Operations > Diagnosis OR OmniSearch (/): Diagnosis
2. Click Download Status Zip.



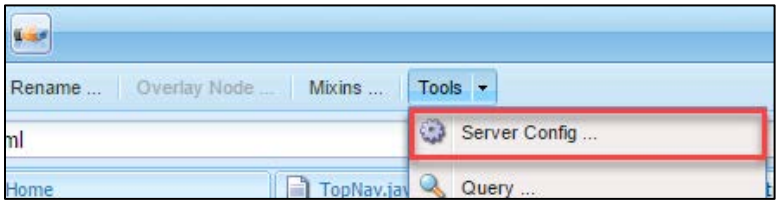
Create custom log files at:

http://localhost:4502/system/console/slinglog

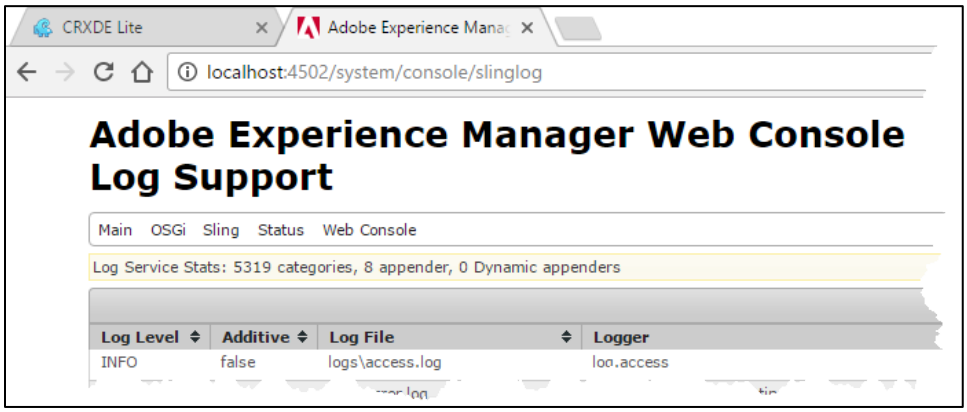
10.1.6 Task – Create a custom Log File

1. **Open the AEM System Console: http://localhost:4502/system/console**

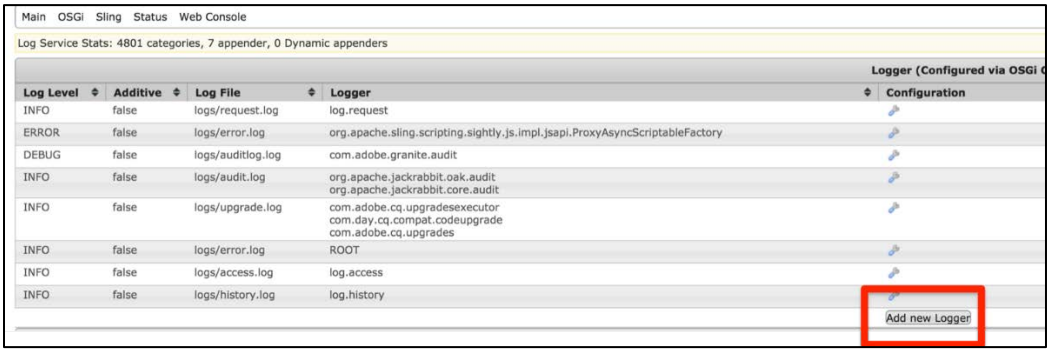
NOTE: You may also click **Tools** > **Server Config** in CRXDE Lite to navigate here.



2. **Navigate to Sling > Log Support:**



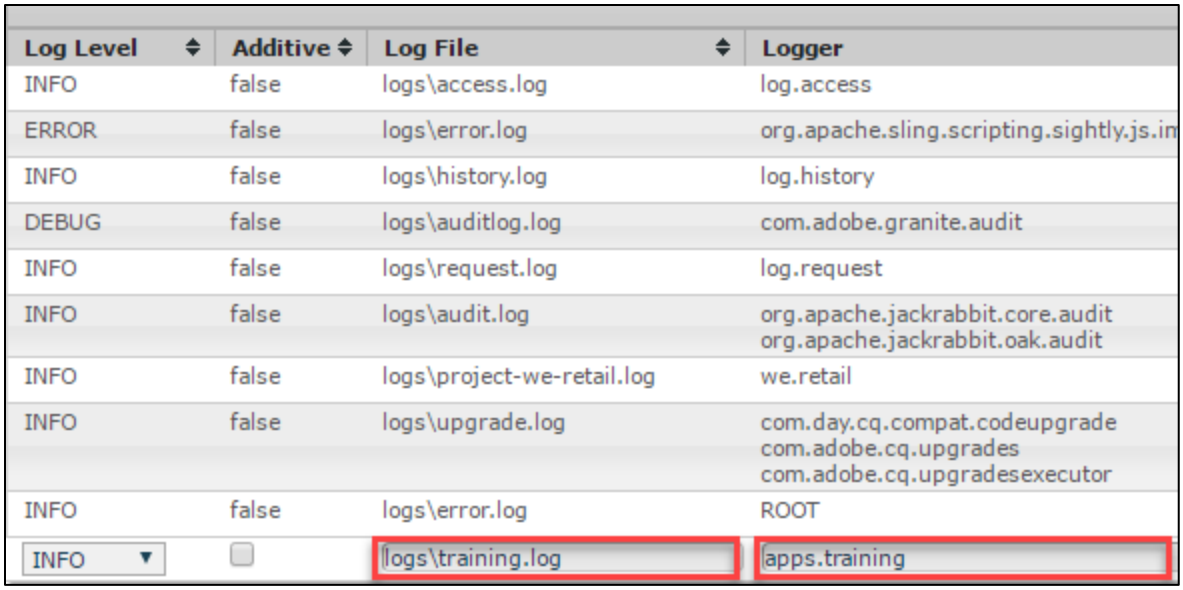
3. **Click Add new Logger at the bottom of the top panel.**



4. **Enter the following in the input fields:**

   - Log File: logs\training.log
   - Logger: apps.training

   NOTE: Do not change any other field values.



Log Level	Additive	Log File	Logger
INFO	false	logs\access.log	log.access
ERROR	false	logs\error.log	org.apache.sling.scripting.sightly.js.in
INFO	false	logs\history.log	log.history
DEBUG	false	logs\auditlog.log	com.adobe.granite.audit
INFO	false	logs\request.log	log.request
INFO	false	logs\audit.log	org.apache.jackrabbit.core.audit org.apache.jackrabbit.oak.audit
INFO	false	logs\project-we-retail.log	we.retail
INFO	false	logs\upgrade.log	com.day.cq.compat.codeupgrade com.adobe.cq.upgrades com.adobe.cq.upgradesexecutor
INFO	false	logs\error.log	ROOT
INFO		logs\training.log	apps.training

5. **Click Save in the Configuration column on the right side.**

10.1.7 Task – Use log statements in the topnav Component

1. **Using CRXDE Lite, navigate to /apps/training/components/structure/site-topnav.**
2. **Open topnav.js.**
3. **Using the code from the Exercise_Files for this module, paste in the contents of topnav.js. Do not copy from this exercise book. The following is for illustration purposes only. Notice the addition of a logging message (in bold below):**

topnav.js

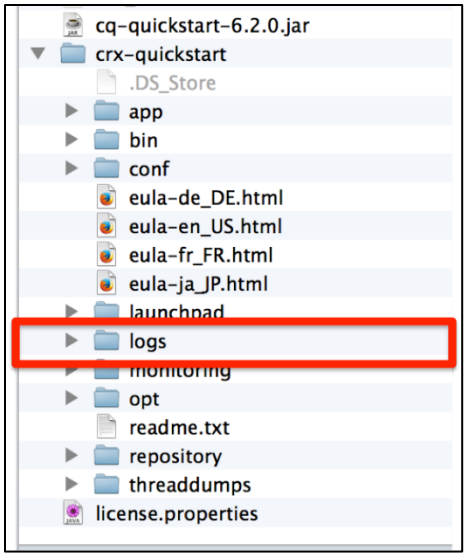
```
// Server-side JavaScript for the topnav logic
use(function () {
    var items = [];
    var root = currentPage.getAbsoluteParent(2);

    //make sure that we always have a valid set of returned items
    //if navigation root is null, use the currentPage as the the navigation root
    if(root == null){
       root = currentPage;
    }

    //Logging Message
    log.info("########[JS] Root page is: {}",  root.getTitle());

    var it = root.listChildren(new Packages.com.day.cq.wcm.api.PageFilter());
    while (it.hasNext()) {
        var page = it.next();
        items.push(page);
    }

    return {
        items: items,
        root: root
    };
});
```

Take note of the logging statement:

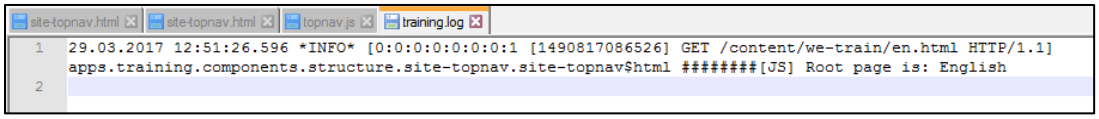
```
//Logging Message
    log.info("########[JS] Root page is: {}",  root.getTitle());
```

We discussed the global objects before. The Logger object is one of the global objects that arepredefined and available to components.

4. **From the Sites Console, open (or refresh) Sites > We.Train > English.**
5. **In your file system, navigate to <AEM author instance installation directory>/crx-quickstart/logs.**



6. **Open training.log.**
7. **Find the message logged in training.log:**



10.1.8 Extra Credit task – Add logging to the Java helper

Using the same concepts as the previous tasks, enable logging using the Java Helper.

Remember you will have to modify the contents of site-topnav.html to call the Java helper.

Component Dialog Boxes

A touch UI dialog box gathers user input using a "form", potentially validates it, and then makes that input available for further use (storage in the JCR, configuration, and so on).

Understanding the types of Dialog Boxes

With Adobe Experience Manager, there are two types of dialog boxes available—the classic dialog box, and the touch-optimized (also known as "Touch UI") dialog box. For a better user experience and for more features/functionality, it is recommended you use the touch-optimized dialog box (cq:dialog).

NOTE: This course's focus is on touch-optimized dialog boxes. A discussion of Classic dialog boxes and the differences between Touch and Classic dialog boxes is provided in the Appendix for this course.

Touch-Optimized UI Dialog Boxes

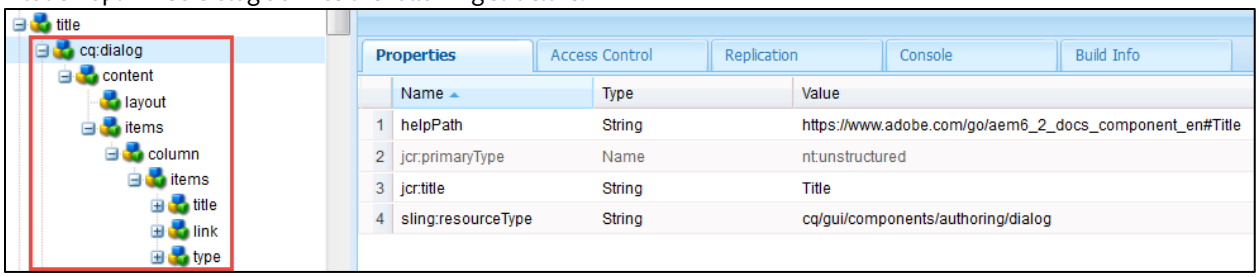
The touch-optimized UI makes use of Granite.js. All the elements you can use for creating the dialog boxes are saved in the /libs/granite/ui/components/foundation/form directory. The root node of a dialog box should extend cq/gui/components/authoring/dialog.

In addition, you may copy cq:dialog nodes from /libs/wcm/foundation/components/ in order to build/use your own Dialogs. This will be explained in the exercise.

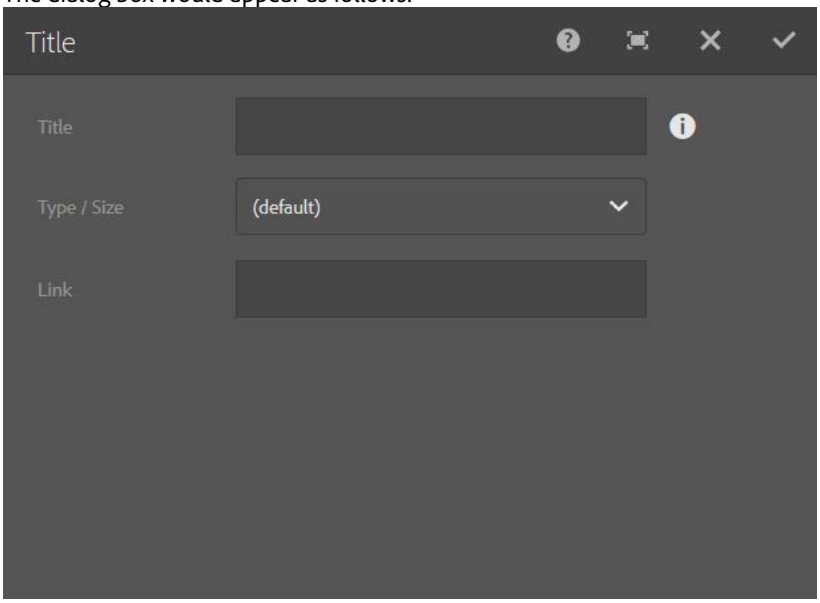
A touch-optimized UI dialog box is defined by using nodes of type nt:unstructured. To define the type of control used, you need to set the node's sling:resourceType property. For example, to define a text field on a Touch UI dialog, set the sling:resourceType property to granite/ui/components/foundation/form/textfield.

Example Node Structure of a Dialog Box

A touch-optimized dialog box has the following structure:



The dialog box would appear as follows:



Take a look at what each node means:

The **cq:dialog** node produces the outer border and the title bar.

The **content** node provides the container holding the dialog box content.

The **layout** node defines the layout, in this case, "fixed columns".

The **column** node defines the container for the widgets.

The **items** node is the parent of the input forms (widgets).

Here's a description of a few of the properties of the widget:

jcr:primaryType: Defines the Node type, in this case, nt:unstructured

fieldLabel: Tells the author what to do, in this case, "Title".

fieldDescription: Gives the author more information, in this case, "Leave empty to use the page title".

name: Tells the JS code that backs this widget what property name to write, in this case, a property named 'title'.

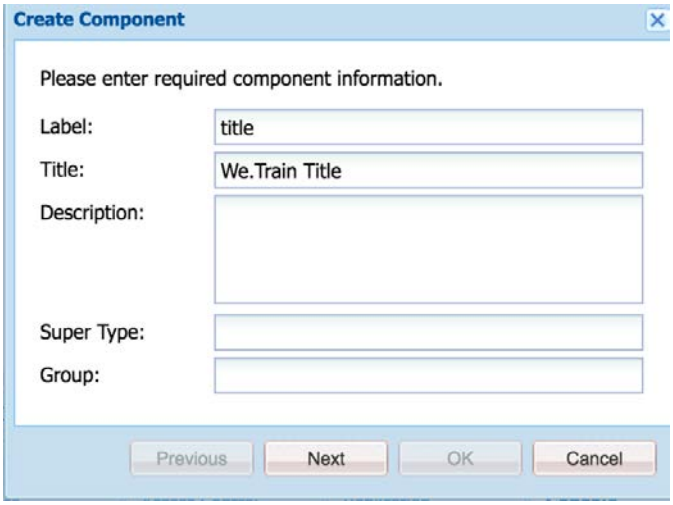
sling:resourceType: Defines the type of input form, in this case, a 'textfield' - single line, alphanumeric

10.1.9 Task – Create a Title Component

1. **Using CRXDE Lite, navigate to /apps/training/components/structure.**
2. **Right-click on /apps/training/components/structure and select Create... > Create Component.**
3. **Enter the following values in the dialog:**

Label: title

Title: We.Train Title



4. **Click Next, and then click OK.**
5. **Save your changes.**
6. **Rename the default script title.jsp to title.html.**
7. **Replace all code in title.html with one <h1> tag as shown here:**

title.html

```
<h1 data-sly-use.title="title.js">${title.text}</h1>
```

8. **Create another file in the same location as title.html named title.js.**
9. **Using the code from the Exercise_Files for this module, paste in the contents of title.js. Do not copy from this exercise book. The following is for illustration purposes only.**

title.js

```
"use strict";

use(function () {

    var CONST = {
        PROP_TITLE: "jcr:title",
        PROP_TAG_TYPE: "type"
    }

    var title = {};
```

```
// The actual title content retrieved from the property title
// or, the pageProperties, or, the currentPage.name
    title.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;

    return title;

});
```

10. **Using the code from the Exercise_Files, modify the code in the body.html file you created earlier under the contentpage node. Notice the modified code below in bold:**

NOTE: body.html is located in: /apps/training/components/structure/contentpage/

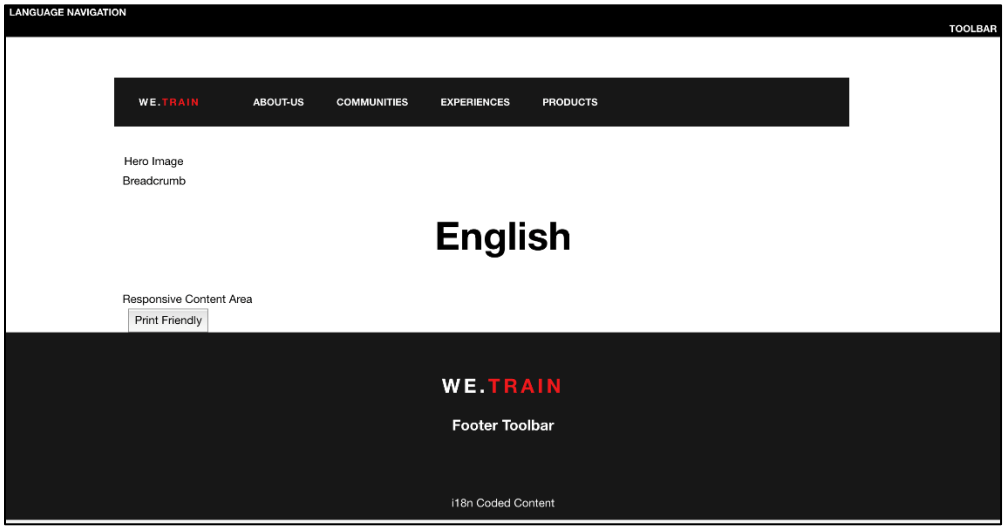
Do not copy from this exercise book. The following is for illustration purposes only.

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12" data-
sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-GridColumn--
default--12" style="padding-top: 150px;">
                 <div>Hero Image </div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--default--
12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div>Breadcrumb</div>
                            <div class="we-Header">
                                 <div data-sly-resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                                                </div>
                            <div>Responsive Content Area</div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--default--12"
data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
```

```
</div>
```

11. **Using the Sites Console, navigate to Sites > We.Train > English.**
12. **Open (or refresh) the English page. Notice that now instead of "Title Component" it renders the title property:**



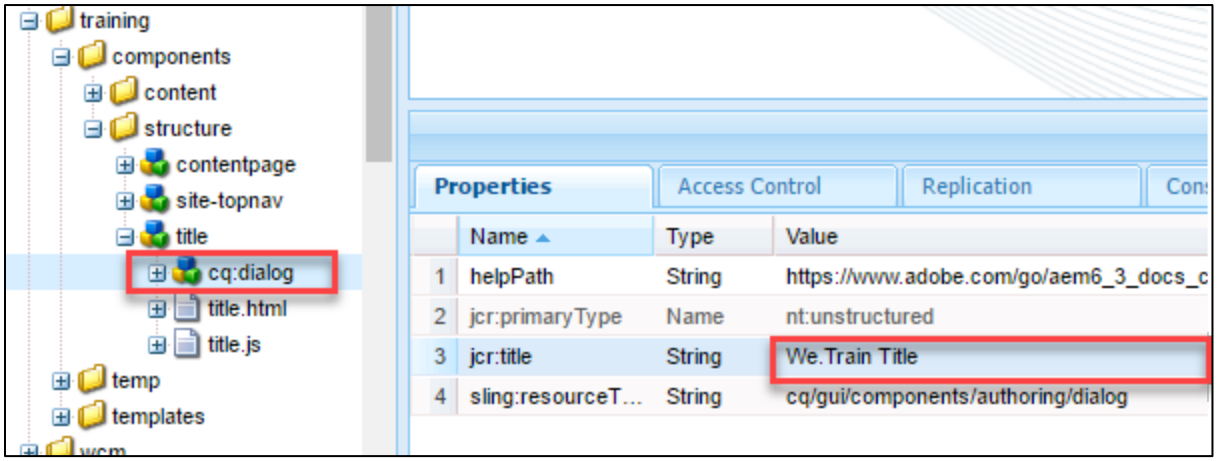
10.1.10 Task – Create a dialog box for the Title Component

Now we have a title component that displays the default page title, but no way to customize the rest of the title. We need to add a component dialog box to the component so that authors can enter a custom title.

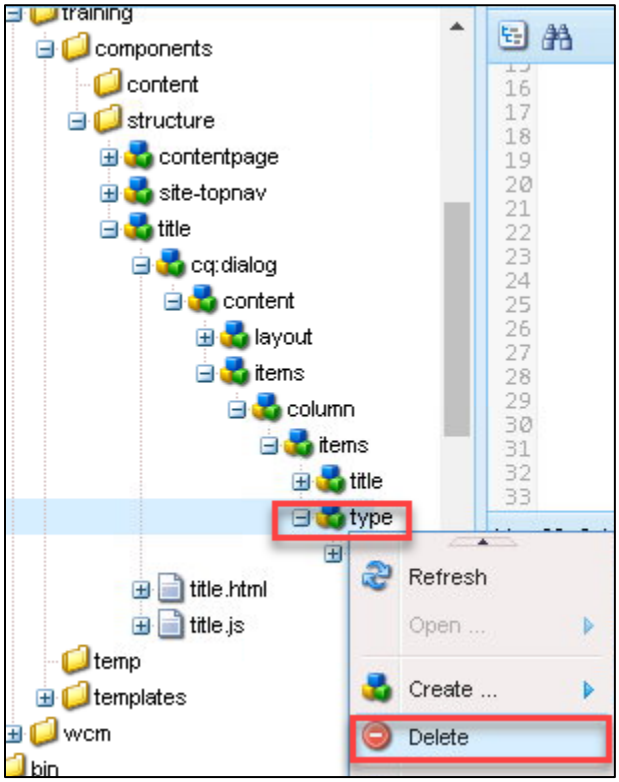
1. **Navigate to /libs/wcm/foundation/components/title.**
2. **Select the cq:dialog node, right-click and copy it.**
3. **Navigate to /apps/training/components/structure/title.**
4. **Paste the copied node.**

Tip: You must name the Touch-optimized dialog box **cq:dialog** and be aware it is case-sensitive.

5. **Select the cq:dialog node.**
6. **Modify the jcr:title property to be We.Train Title:**

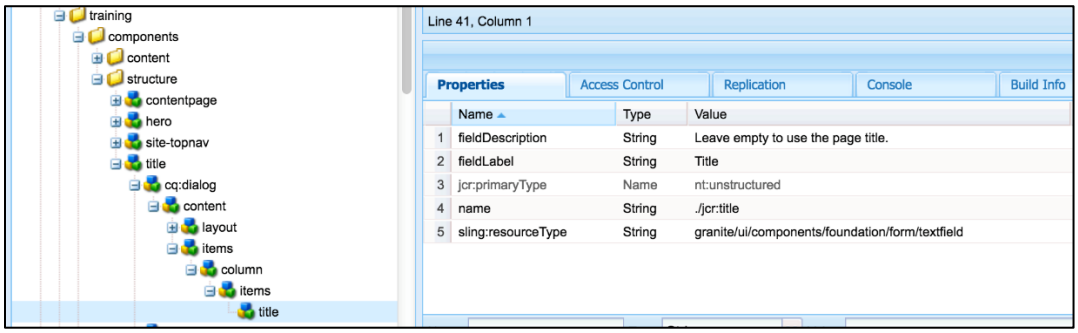


7. **Save your changes.**
8. **Navigate to …/content/items/column/items under cq:dialog.**
9. **Delete the type node:**



10. **Save your changes.**

11. **Select the remaining title node. Take notice of the properties on the node:**



fieldDescription: Informational text when author clicks, taps, or mouses over the "information" icon

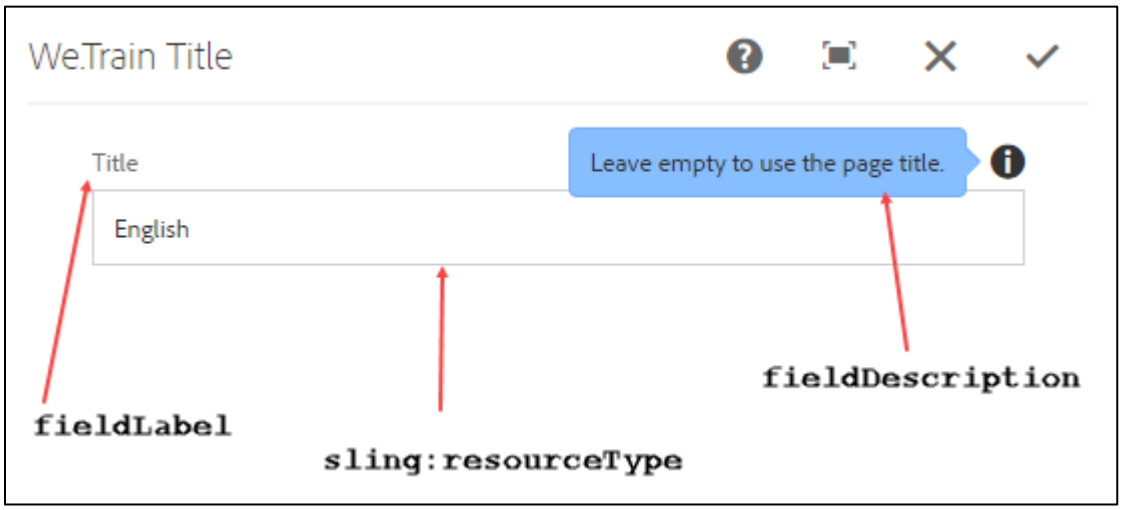
fieldLabel: Label of the input field visible to the author

name: Name of the property the JavaScript will write when the author clicks OK in the dialog box

sling:ResourceType: Definition of the input field type

If the dialog was used by an author, it would look like this screenshot below. The screenshot also shows the use / purpose of each of the properties described here.

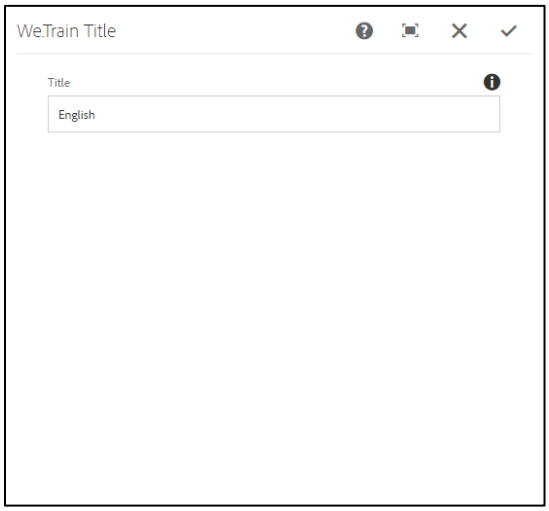
NOTE: You will test your dialog in the next exercise.



10.1.11 Task – Use the Title Dialog

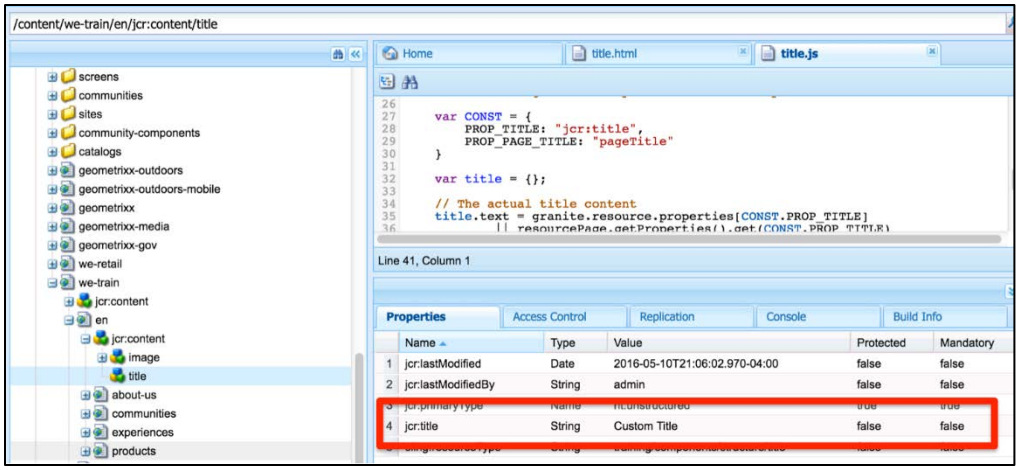
We are now ready to test out our new dialog and enter some custom content.

1. **Open the We.Train English page again in Edit mode. Now you will be able to edit the Title component.**
2. **Double-click on the Title component to render the dialog. Enter a custom title of your choosing and click on the check mark to save the new title. Alternatively, you may click the "English" title component and click the "wrench" (configure) button to render the dialog.**



Now let's see what happened in the repository.

3. **Using CRXDE Lite, navigate to /content/we-train/en/jcr:content.**
4. **Notice that a title node has appeared.**
5. **Look at the properties on the title node. When you entered a custom title, it was stored on the title node in the jcr:title property.**



Using cq:editConfig to Enhance a Component

cq:editConfig nodes are used to configure content input actions when the dialog box is not in control. For example:

- Drag and drop from the Content Finder
- In-place (inline) editing
- Refresh of a dialog box or page after an author action

To add inline editing functionality, the following are required:

- Node of type **cq:editConfig**
- cq:InplaceEditing node

NOTE: You may copy from the existing foundation cq:editConfig node found at /libs/wcm/foundation/components/title. This node can be copied as-is and pasted to your title component you want to enable it for.

Similarly, to add drag-and-drop functionality from the Content Finder, the following are required:

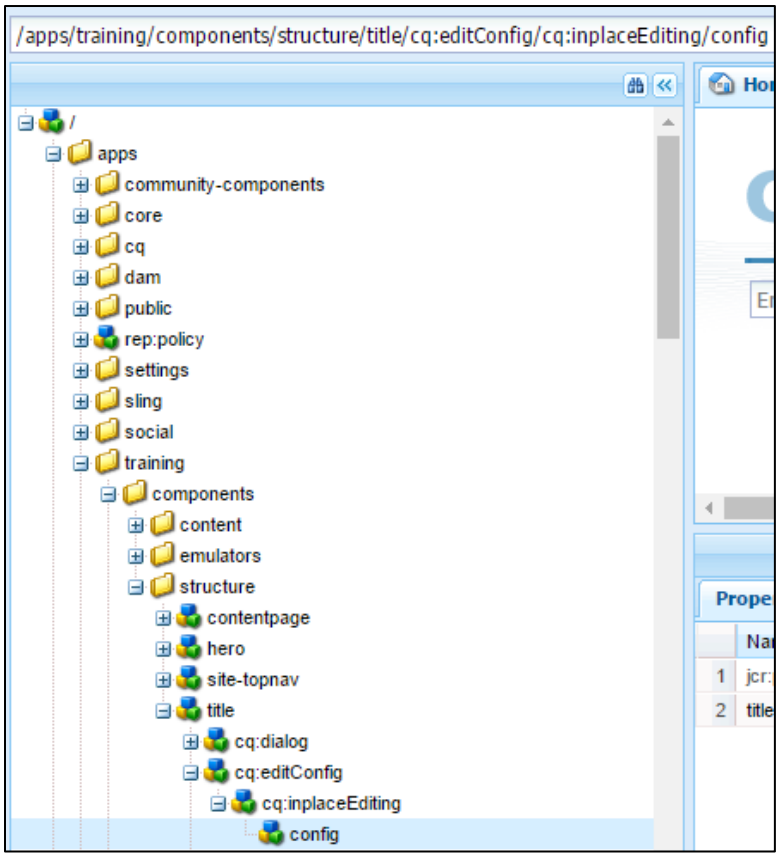
- Node of type **cq:editConfig** that is a child node to the **cq:component** node
- Configuration node that is a child of the **cq:EditConfig** node. This node defines what action you are configuring.
- Asset node that is a child of the **cq:dropTargets** node. This node defines what types of assets the paragraph (responsive grid) will accept. In this example, assets of type image.

10.1.12 Task – Create editConfig for the Title Component

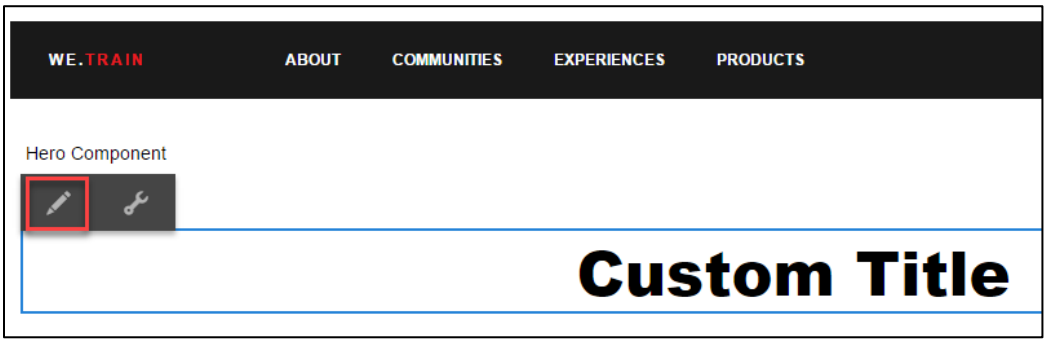
Now we need to create the editConfig node to define the inline editor. Conveniently, we may utilize the existing foundation component in AEM as-is to do this instead of manually creating nodes.

1. **Navigate to /libs/wcm/foundation/components/title.**
2. **Select the cq:editConfig node, right-click and copy it.**
3. **Navigate to /apps/training/components/structure/title.**
4. **Paste the copied node. Save all.**

NOTE: Your expanded node structure should look like this:



5. **Refresh (or reopen) the We.Train English page again in Edit mode.**
6. **Now you should be able to edit your title using inline editing. Click on your component title and edit it using the pencil icon:**



Use Design Dialog Boxes for Global Content

You can use a Design(er) to enforce a consistent look and feel across your website, as well as share global content. This global content is editable when using a Design dialog box. So once again, the many pages that use the same Design(er) will have access to this global content.

Properties of Design Dialog boxes:

- Global Content
- Stored in */etc/designs* instead of the local node of the page
- Root node of the design is of type **cq:Page**
- Child node jcr:content of type **cq:PageContent**
- sling:resourceType = wcm/designer

You can access the Design(er) values by the currentStyle object which is an AEM global object in HTL. This is different from using the properties object as we have done so far. Design dialog boxes are almost identical to component dialog boxes with the following exceptions:

- name: **cq:design_dialog** instead of cq:dialog
- content storage: **/etc/designs** instead of /content/website
- availability: Design mode instead of Edit mode

The Hero Component

Hero Components are considered typical in modern website design.

- Consists of a large eye catching image (or "carousel" of images)
- Includes informational text that draws a visitor in

You can add a hero image using the touch UI dialog box (and the cq:dropTargets node).

- You can add hero text using the touch dialog box. Hero text design should be able to be:
  - Uppercase only
  - Exactly how the author inputs it
- The Hero component should respond to the correct device size.

The FileUpload Form Field

The FileUpload Form field is a field component that is used to upload files. You will see in the rest of exercise activities in this Module that follow how to create a component that accepts binary files as inputs. You can upload an image by both dragging and dropping the image from the Asset browser directly onto the component or through its Configure dialog box.

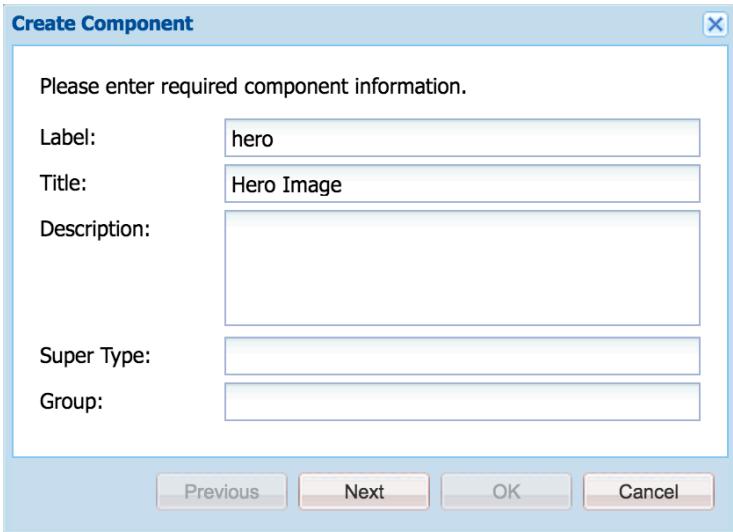
BEST PRACTICE: Use the Assets feature to include the assets to their repositories.

10.1.13 Task – Create an initial Hero Image Component

1. **Using CRXDE Lite, navigate to /apps/training/components/structure.**
2. **Right-click on /apps/training/components/structure and select Create… > Create Component.**
3. **Enter the following values in the dialog:**

Label: hero

Title: Hero Image



4. **Click Next, and then click OK.**
5. **Save your changes.**
6. **Rename the default script hero.jsp to hero.html.**
7. **Replace the sample code in hero.html with this HTL in a tag shown below.**

hero.html

```
<div data-sly-use.hero="hero.js">
    <strong>${hero.text}</strong>
</div>
```

8. **Create a file named hero.js under the hero component node.**
9. **Using the code from the Exercise_Files, paste the code into hero.js. Do not copy from this exercise book. The following is for illustration purposes only.**

hero.js

```
"use strict";
use(function() {
    var CONST = {
        PROP_TITLE: "jcr:title",
    }

    var hero = {}
```

```
    //Get the title text
    hero.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;

    return hero;
});
```

If you take a look at the code, you will see that the initial hero.html renders ${hero.text}. Since we have not created a dialog yet, it will display the page Title.

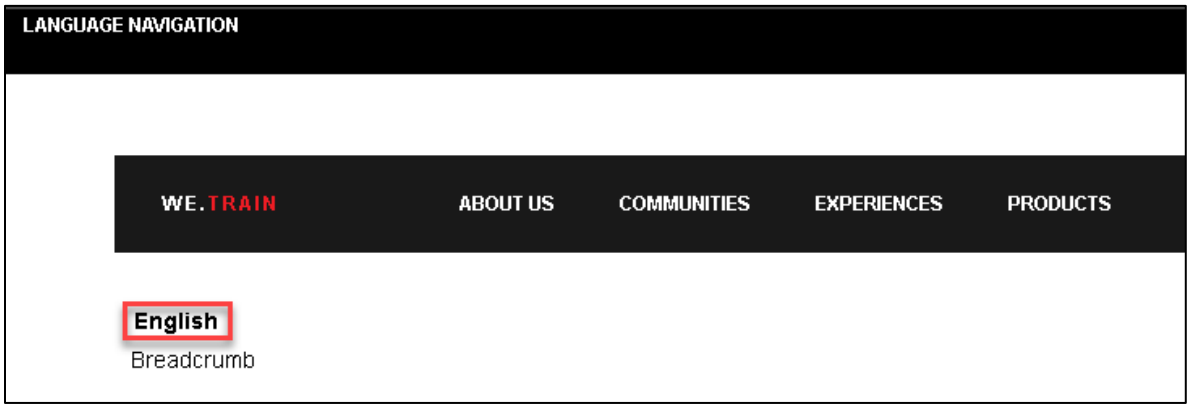
10. **Navigate to /apps/training/components/structure/contentpage.**
11. **Open body.html for editing. We need to replace the existing simplistic in the body.html to include the Hero component on the page. See below in bold. Do not copy from this exercise book. The following is for illustration purposes only.**
12. **Using the code from the Exercise_Files, paste the code into line 12 of body.html (the tag that now includes the hero component):**

body.html

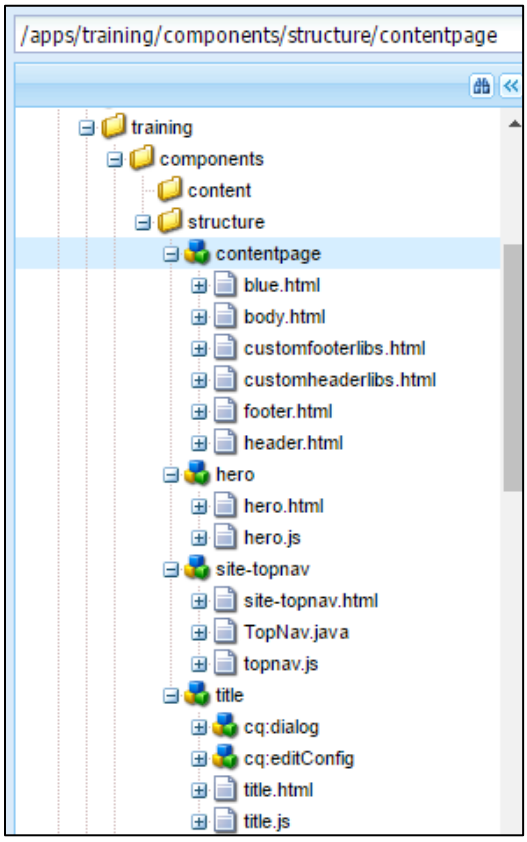
```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12" data-
sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-GridColumn--
default--12" style="padding-top: 150px;">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--default--
12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">


                                                <div class="aem-
breadcrumb">Breadcrumb</div>
                            <div class="we-Header" data-sly-resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div>Responsive Content Area</div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--default--12"
data-sly-include="footer.html"></div>
                </div>
```

```
            </div>
        </div>
    </div>
</div>
```

13. **Save your changes.**
14. **Using the Sites Console, navigate to or refresh Sites > We.Train > English. You will notice that the page Title now shows up in place of the Hero Component:**



NOTE: Your structure components should look like this so far:



10.1.14 Task – Create a Dialog for Author Input

We now need to define a component dialog so the Author can enter a custom string for the Hero Component.

Because the Hero component deals with an image, we need a dialog structure that knows how to deal with binary data (files). So we are going to copy the dialog from the Foundation Image component.

1. **Using CRXDE Lite, navigate to /libs/wcm/foundation/components/image.**
2. **Copy the cq:dialog node.**
3. **Paste to /apps/training/components/structure/hero.**

10.1.15 Task – Modify the text feature of the Hero Component

In order to use our dialog, to modify the jcr:title property, we need to define editConfig. We are also going to add the ability to drag and drop images from the AEM DAM (Digital Asset Manager) into the Hero component.

1. **Using CRXDE Lite, right-click on /apps/training/components/structure/hero node.**

Select **Create…** > **Create Node**. Enter the following values:

Name: cq:editConfig

Type: cq:EditConfig

2. **Click OK and Save.**
3. **Right-click on the cq:editConfig node you created and select Create… > Create Node. Enter the following values:**

Name: cq:dropTargets

Type: nt:unstructured

4. **Save your changes.**
5. **Right-click on the cq:dropTargets node and select Create… > Create Node. Enter the following values:**

Name: image

Type: cq:DropTargetConfig

6. **Save your changes.**
7. **Add the following three properties on the image node:**

Name	Type	Value
accept	String[] (Multi)	image/.*
groups	String[] (Multi)	media
propertyName	String	./fileReference

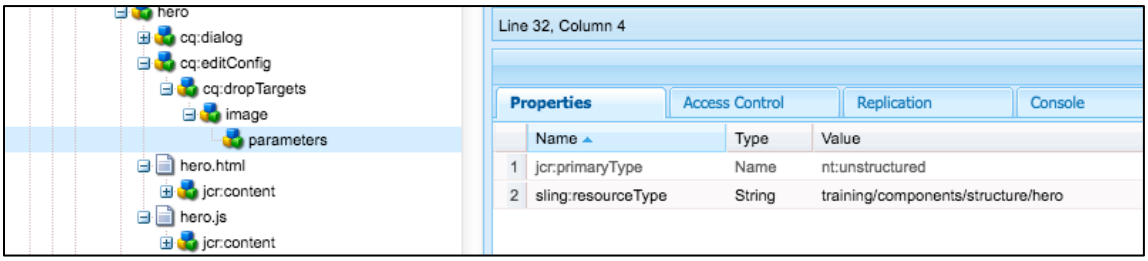
8. **Save your changes.**
9. **Right-click on the image node and select Create… > Create Node. Enter the following values:**

Name: parameters

Type: nt:unstructured

Add the following properties on the parameters node:

Name	Type	Value
sling:resourceType	String	training/components/structure/hero

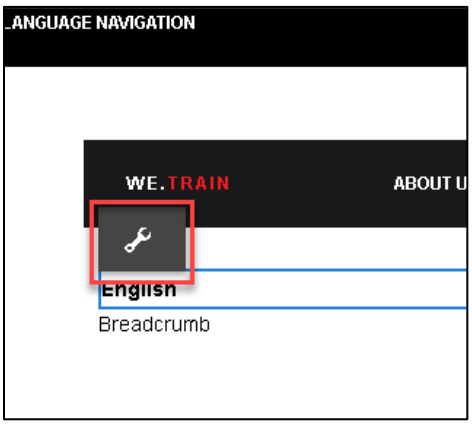


10. **Save your changes.**

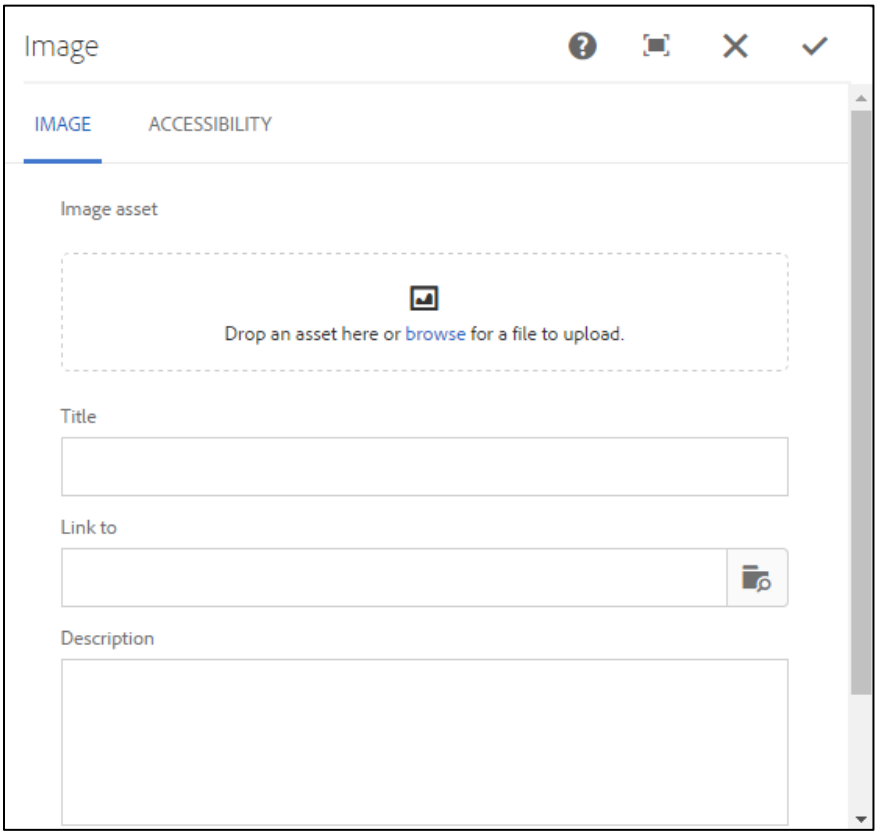
10.1.16 Task – Test the Hero Component

Now test your Hero Component.

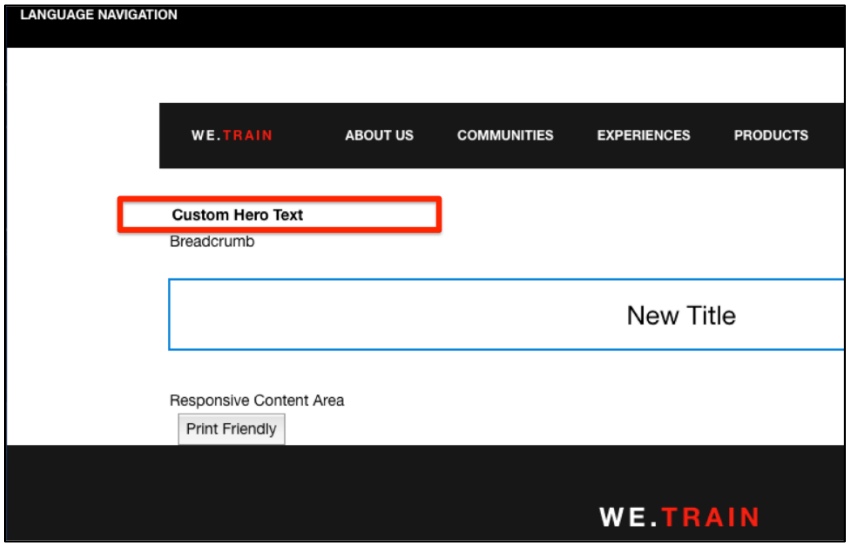
1. **Using the Sites Console, navigate to or refresh Sites > We.Train > English.**
2. **Click to select the Hero component and click on the wrench:**



3. **Verify this dialog appears:**



4. **In the dialog, enter a value as shown below in Title (Custom Hero Text) and click the checkmark in the upper right to save. The new value should appear:**



10.1.17 Task – Modify the Hero Component to display an Image

1. **Using CRXDE Lite, open: /apps/training/components/structure/hero/hero.js**
2. **Using the code from the Exercise_Files for this module, paste in the new contents of hero.js. Do not copy from this exercise book. The following is for illustration purposes only. This will add the functionality to *capture* an uploaded image.**

hero.js

```
"use strict";
use(function() {
    var CONST = {
        PROP_TITLE: "jcr:title",
        PROP_REF_HERO_IMAGE: "fileReference",
        PROP_UPLOAD_HERO_IMAGE: "file"
    }

    var hero = {}

    //Get the title text
    hero.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;



    //Check for file reference from the DAM
    var image = properties.get(CONST.PROP_REF_HERO_IMAGE, String.class);
    if(image == "undefined"){
       //Check for file upload
       var res = resource.getChild(CONST.PROP_UPLOAD_HERO_IMAGE);
       if(res != null){
              image = res.getPath();
       }
    }
    if(image != "undefined"){
       hero.style = "background-image:url(" + image + ");";
    }
    return hero;
});
```

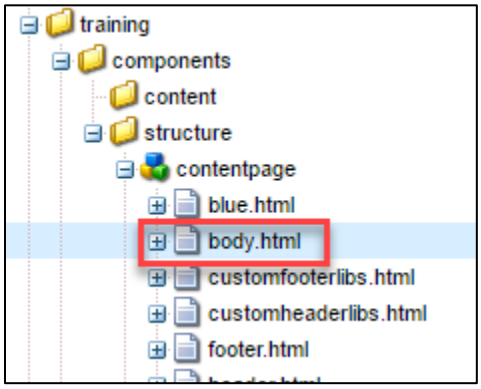
3. **Save your changes.**

4. **Using the code from the Exercise_Files for this module, paste the new contents of hero.html. Do not copy from this exercise book. The following is for illustration purposes only. This will add the functionality to *display* an uploaded image.**

hero.html

```
<div data-sly-use.hero="hero.js" class="we-HeroImage width-full ratio-16by9"
style="${hero.style @ context='styleString'}">
        <div class="container cq-dd-image">
            <div class="we-HeroImage-wrapper">
                <strong class="we-HeroImage-title">${hero.text}</strong>
            </div>
        </div>
    </div>
```

5. **Save your changes.**
6. **Navigate up to the contentpage node and open body.html for editing.**



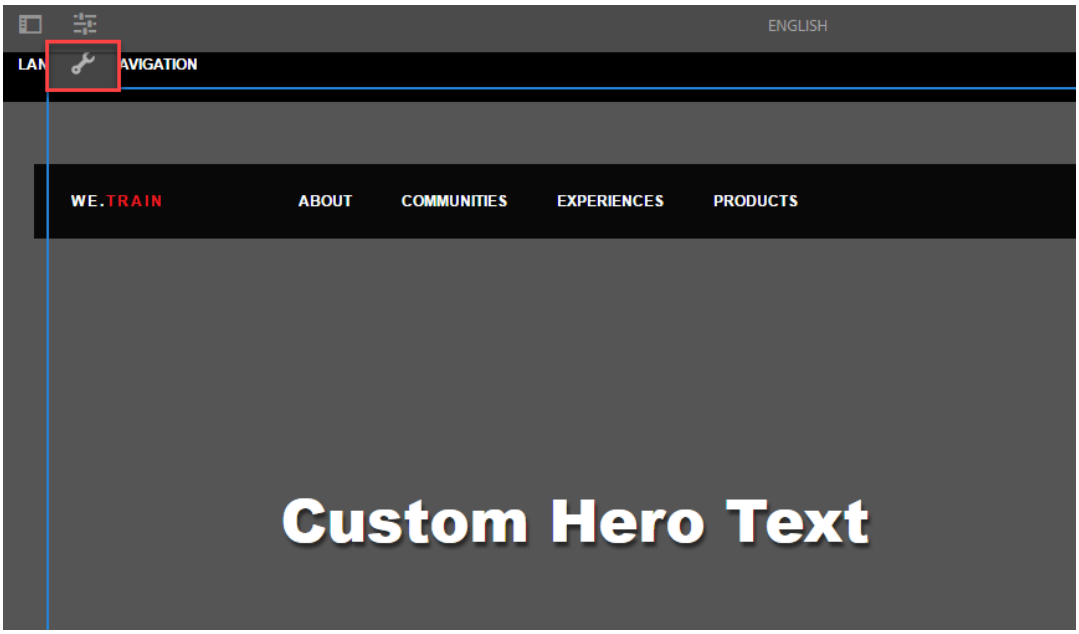
7. **Modify the contents of body.html using the code from the Exercise_Files.**

body.html

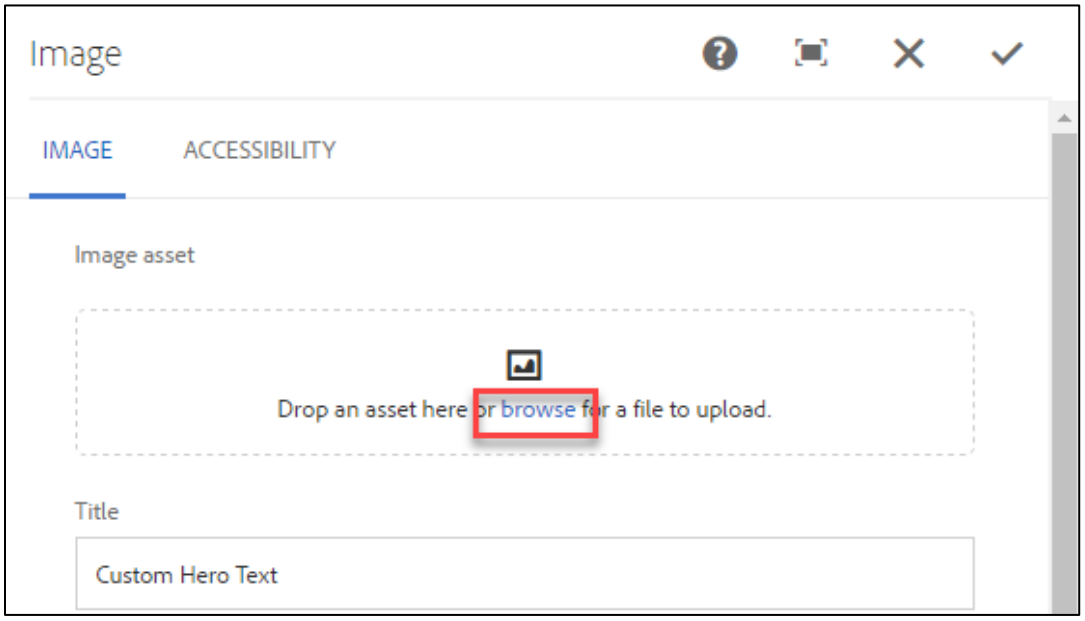
```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12" data-
sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-GridColumn--
default--12">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--default--
12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div class="aem-
breadcrumb">Breadcrumb</div>
                            <div class="we-Header" data-sly-resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div>Responsive Content Area</div>
```

```
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--default--12"
data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

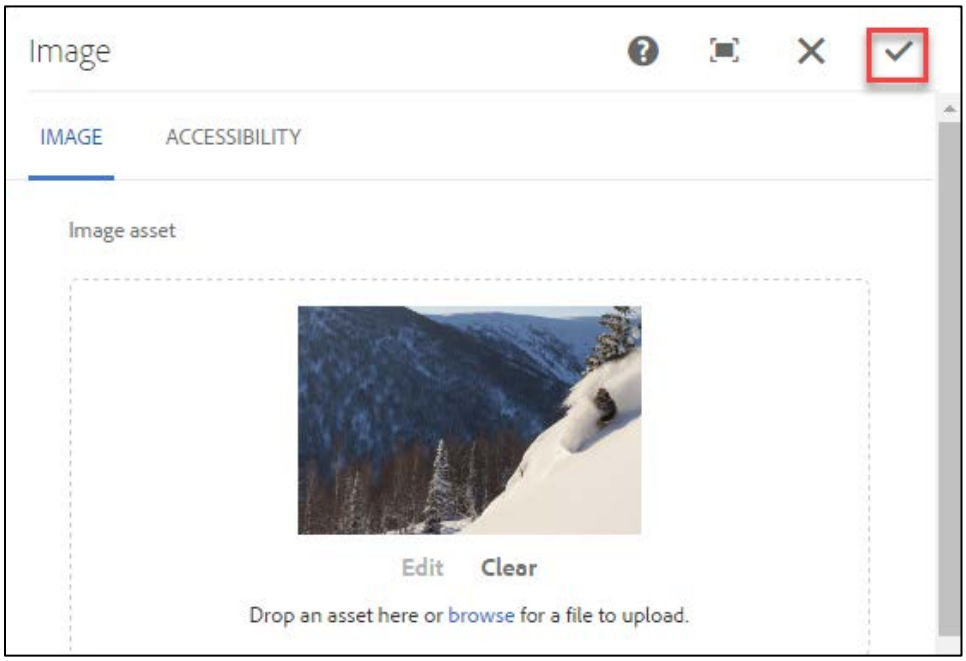
8. **Save your changes.**
9. **Using the Sites Console, open or refresh Sites > We.Train > English in Edit mode. Notice the new Hero component.**
10. **Click on the Hero Component Configure button (wrench) to render the dialog:**



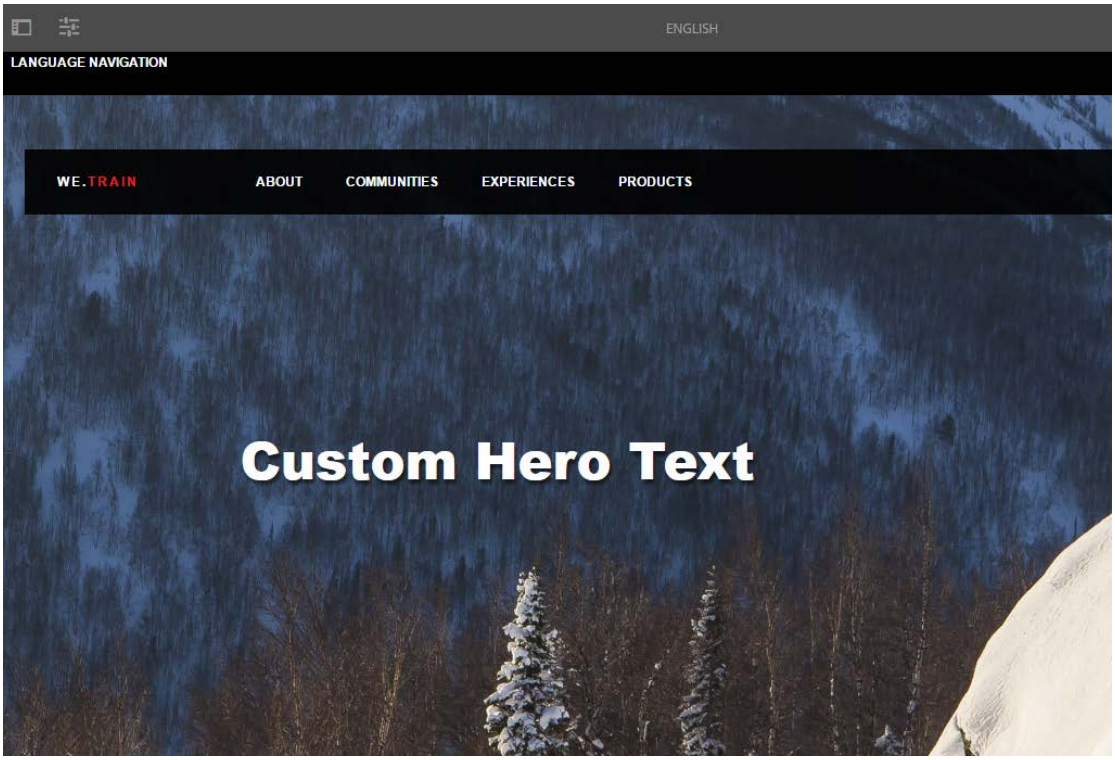
11. **Click browse:**



12. **Upload one of the images from the Exercise_Files into the dialog.**
13. **Click Done.**



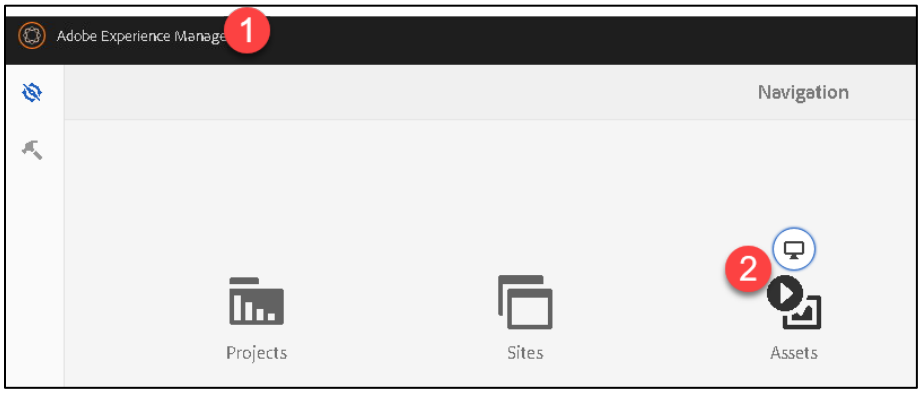
14. **Your hero image should now display like this (with a large, high-quality hero image):**



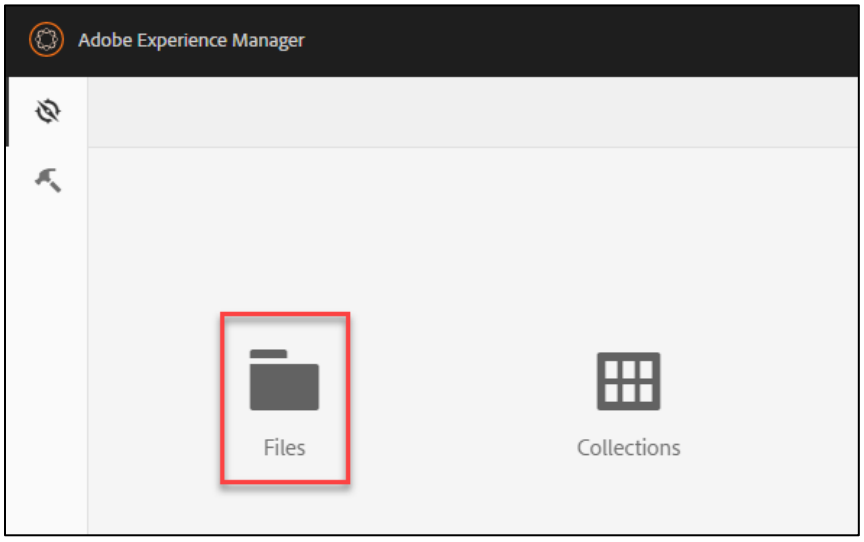
10.1.18 Extra Credit Task – Use the DAM to configure images in the Hero component

Upload all the supplied images into the DAM. This method allows you to drag hero images of your choosing from the Assets selector in the sidebar instead of uploading manually each image.

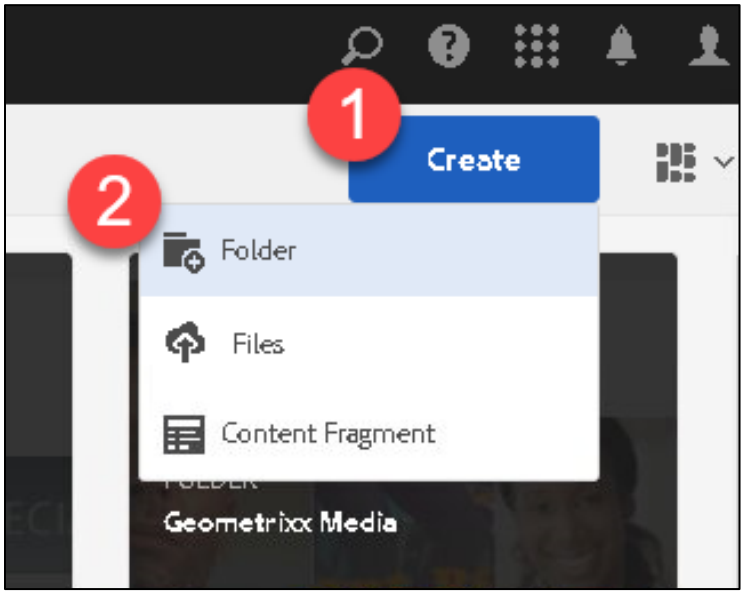
1. **In AEM, navigate to the Assets console:**



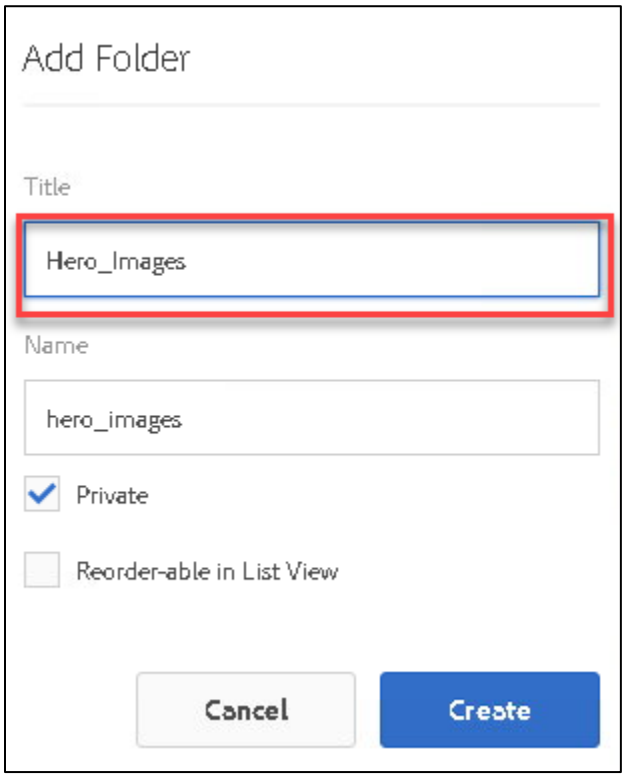
2. **Click Files.**



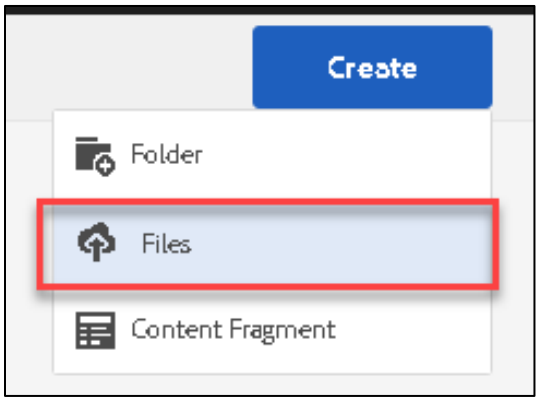
3. **Click Create > Folder:**



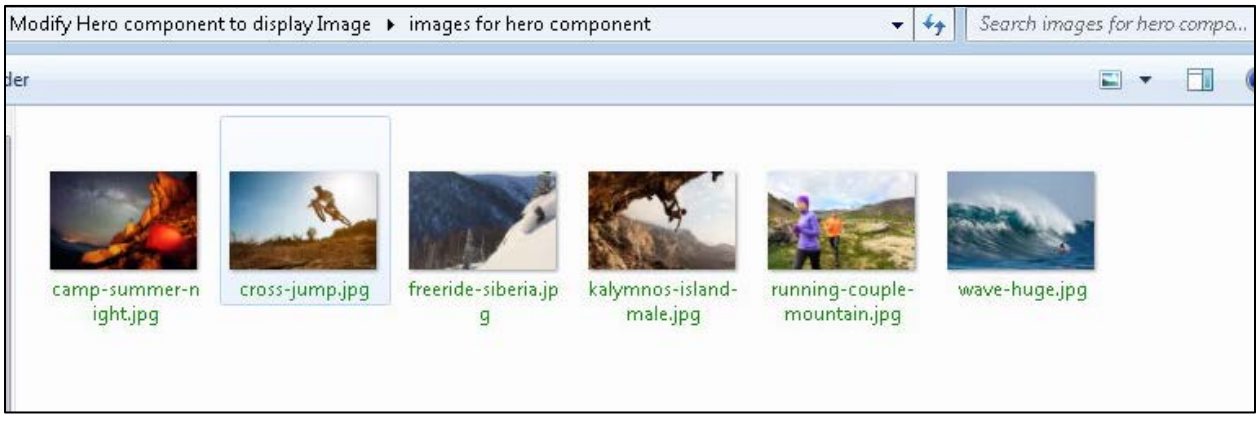
4. **Enter Hero_Images as the title for your folder and click Create:**



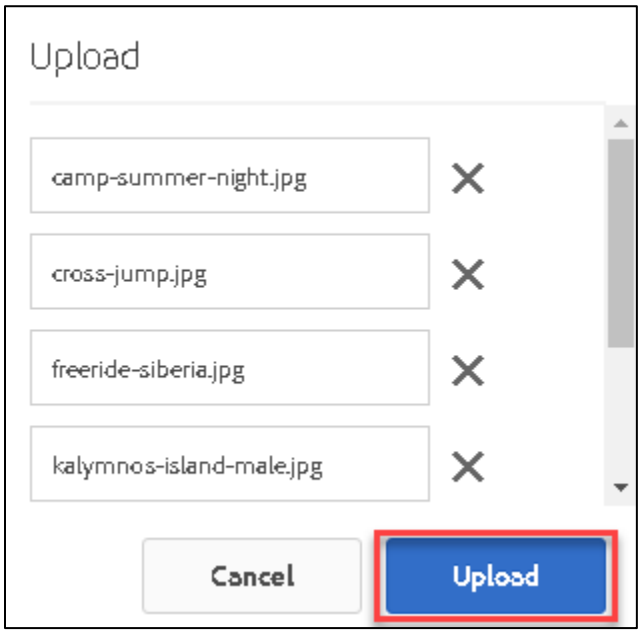
5. **Select your folder and open it. Click Create > Files:**



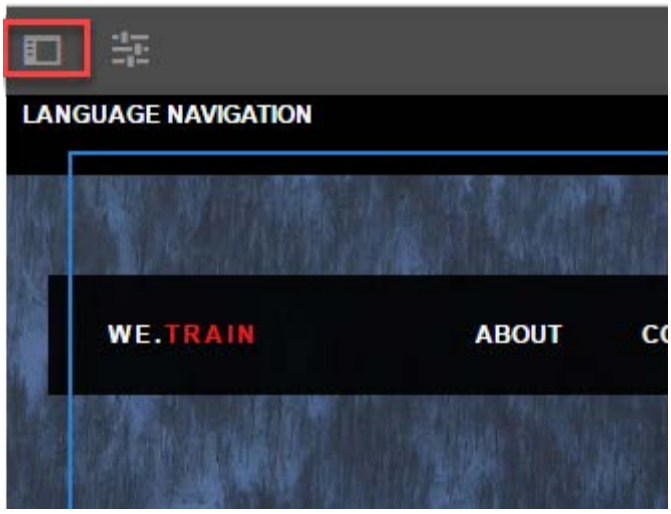
6. **Select all the images from the Exercise Files ("Modify Hero component to display Image") and add them to the DAM:**



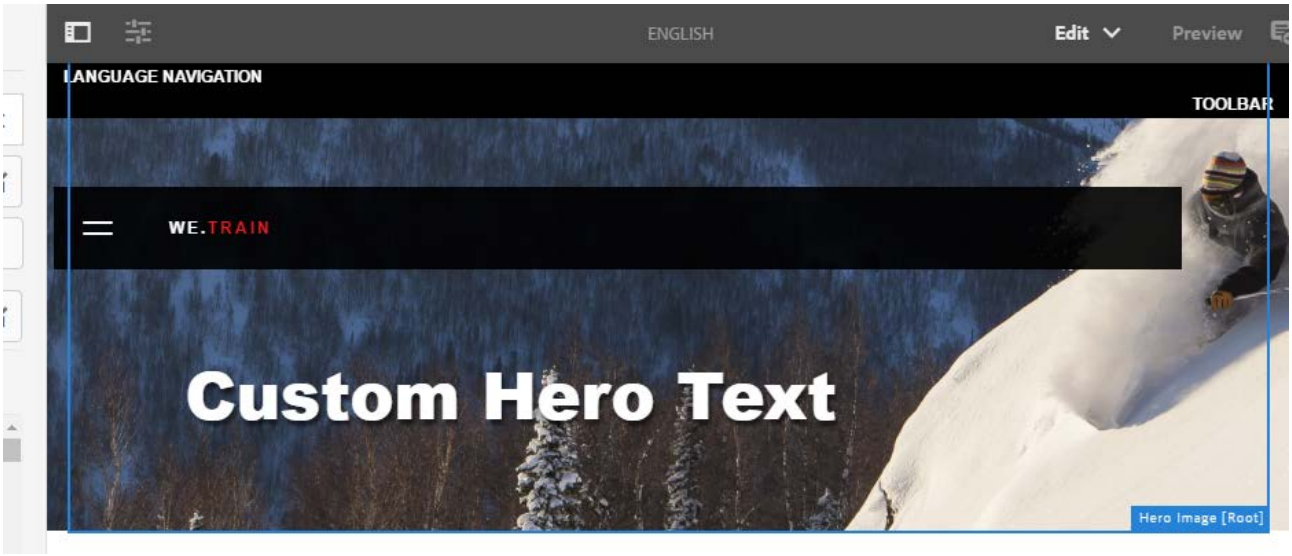
7. **Click Upload:**



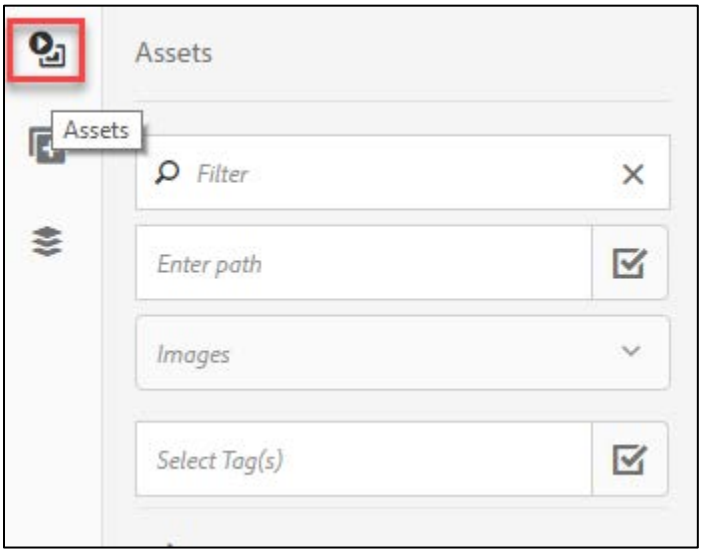
8. **Using the Sites Console, open or refresh Sites > We.Train > English in Edit mode.**
9. **Toggle the Side Panel On:**



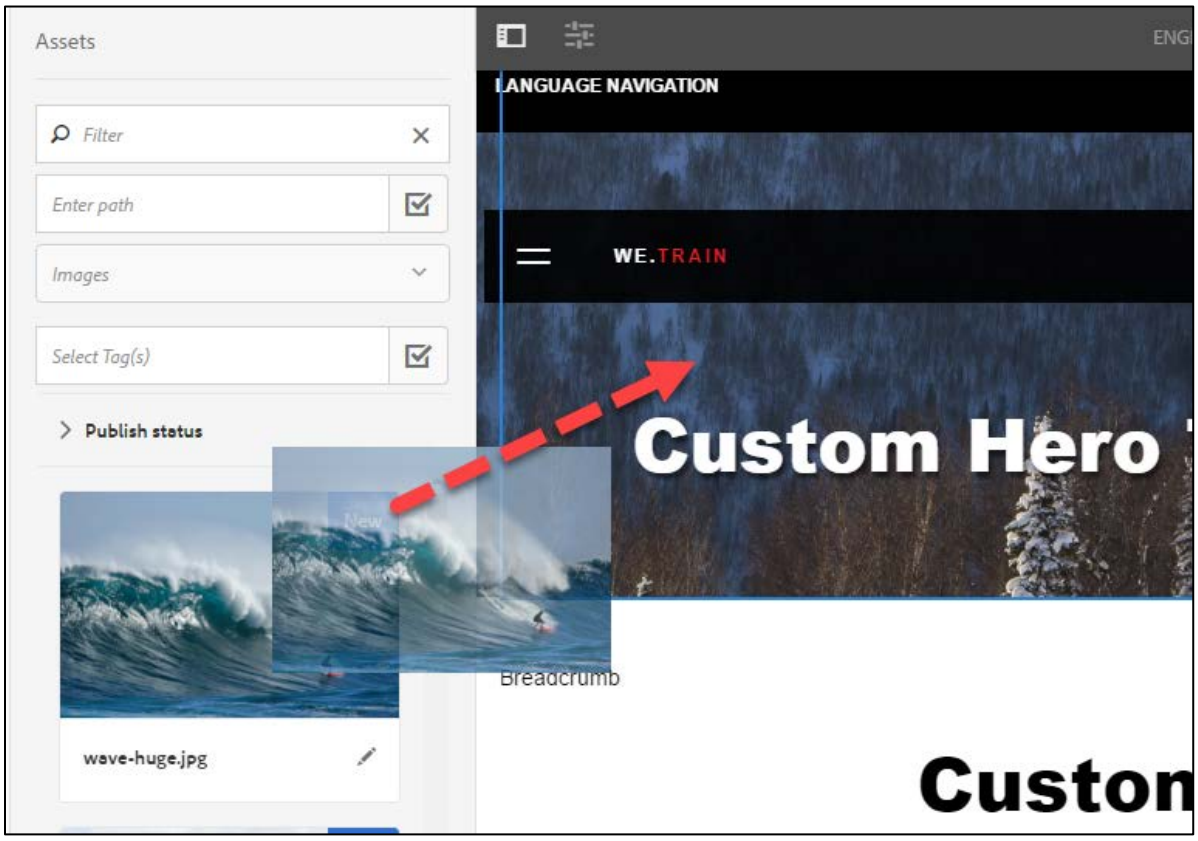
10. **Select the Hero Image Component:**



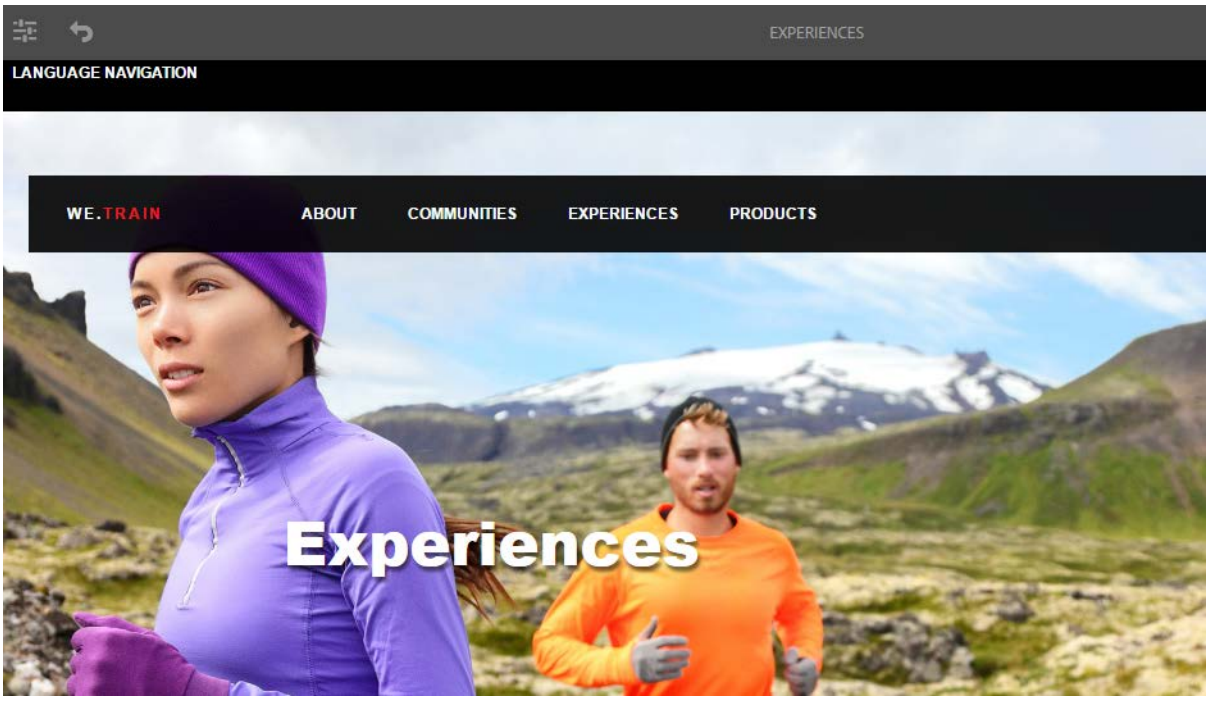
11. **Click Assets.**



12. **Drag your images from the Assets selector in the Side Panel to the blue "bar" in your hero component:**



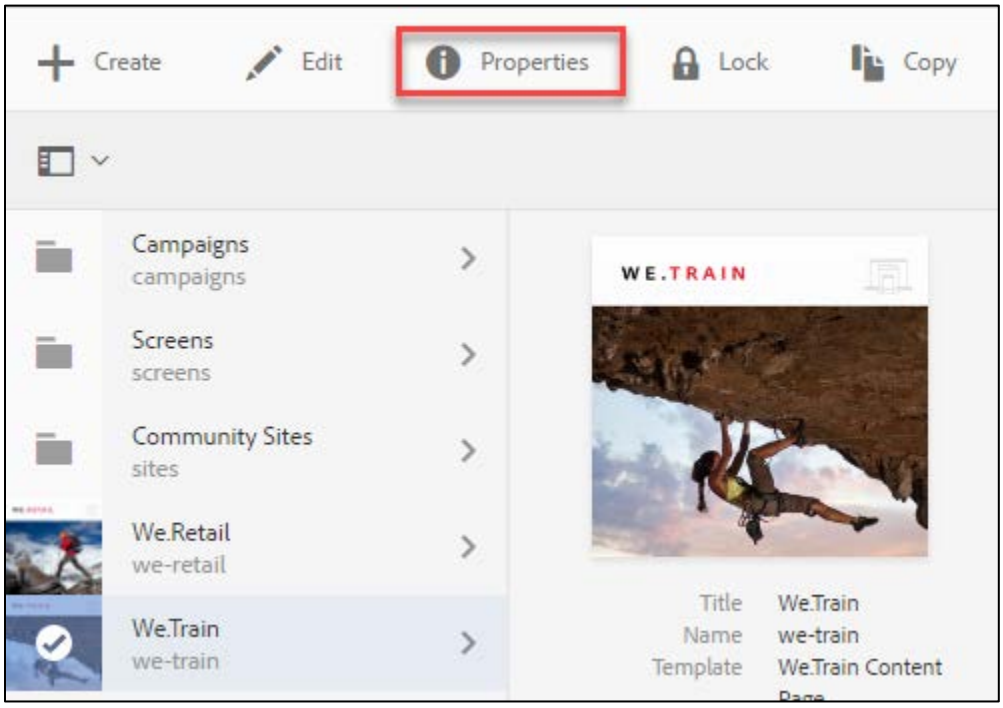
13. **Try the same for a few other pages in We.Train:**



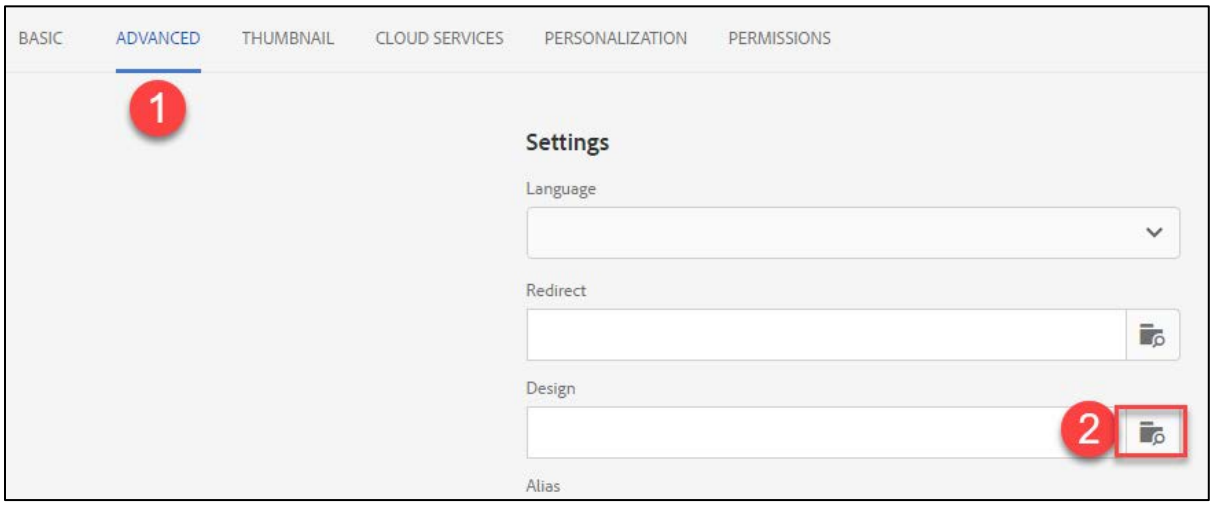
10.1.19 Task – Assign a Design to We.Train

Review: In Module 9, **Exercise 9.1.1,** we defined a Design. At this point, we need to call that design as it will be propogated to our Design Dialog for the Hero Component that we are building in the next exercise. We need this design to successfully save our design dialog properties to the designer.

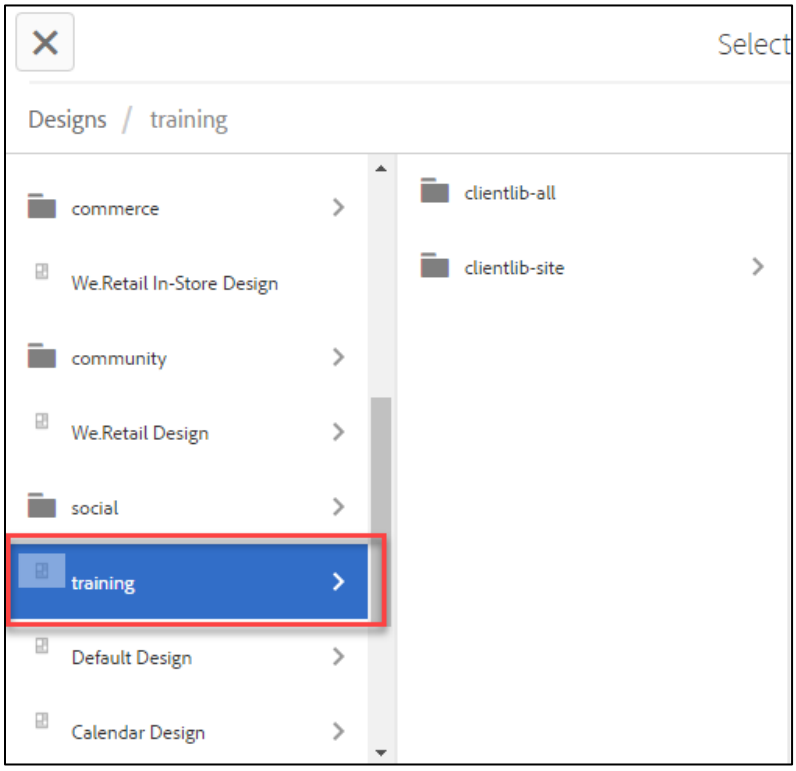
1. **Using the Sites Console, navigate to Sites > We.Train.**
2. **Select the We.Train page, and click Properties:**



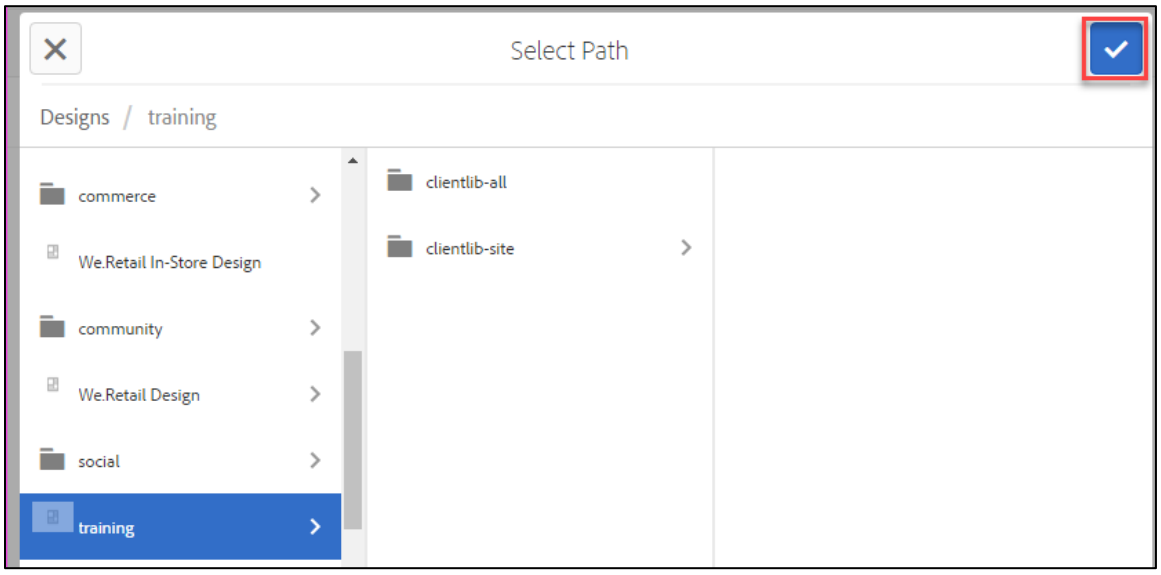
3. **Click the Advanced tab (1), then click Browse to browse for a Design (2):**



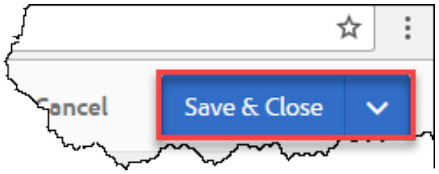
4. **Scroll down and select training:**



5. **Click the checkmark in the upper right to save your selection:**



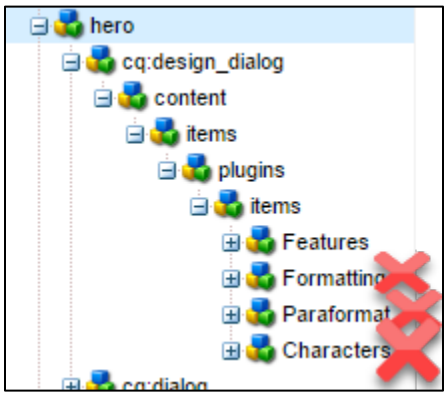
6. **Click Save & Close in the upper right:**



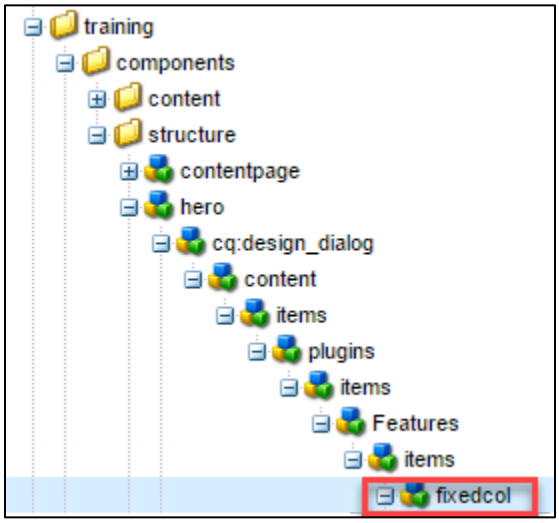
10.1.20 Task – Create a Design Dialog for the Hero Component

Now to add a design dialog that allows the author to specify the styling of the text overlaying the Hero image.

1. **Copy /libs/foundation/components/text/cq:design_dialog to /apps/training/components/structure/hero.**
2. **Delete the Formatting, Paraformat, and the Characters nodes as shown:**

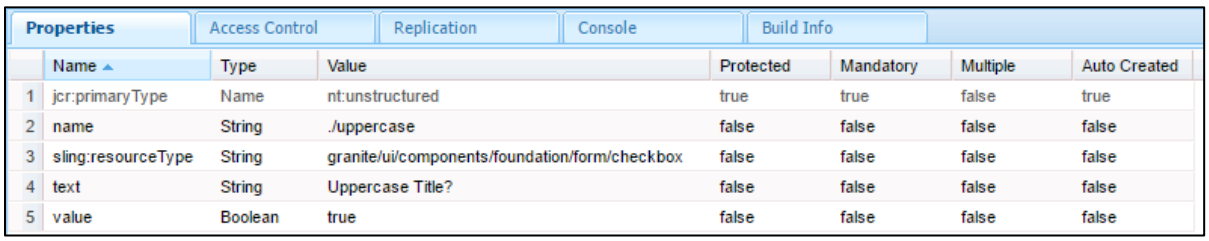


3. **Delete the fixedcol node under …/Features/items:**



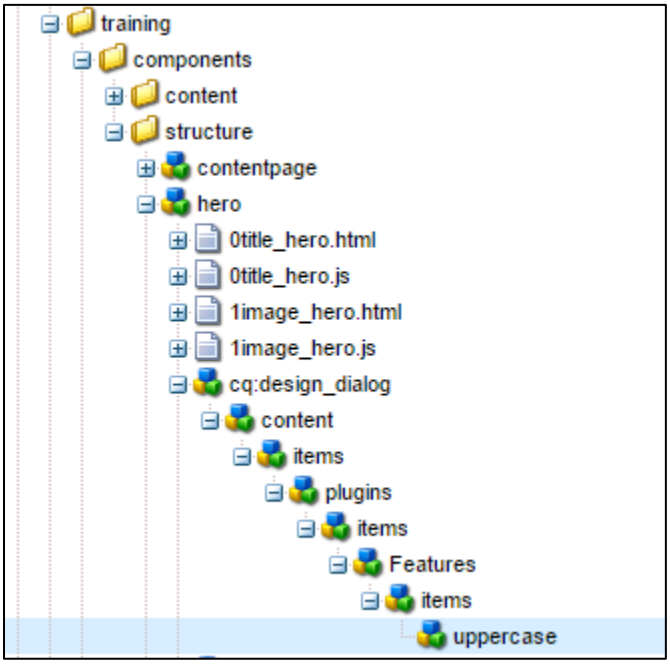
4. **Save your changes.**
5. **In …/Features/items, create a new node: uppercase [nt:unstructured] and then add the following four properties (making sure to save your changes after adding each property):**

Name	Type	Value
name	String	./uppercase
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	Uppercase Title?
value	Boolean	true



NOTE: Ensure your node is placed in:

/apps/training/components/structure/hero/cq:design_dialog/content/items/plugins/items/Features/items/uppercase



6. **Now let's modify the code to take advantage of the new information from the design dialog.**

7. Navigate to and open /apps/training/components/structure/hero/hero.js.

8. **Using the code from the Exercise_Files for this module, paste the new contents of hero.js. Do not copy from this exercise book. The following is for illustration purposes only.**

Notice that the new code (highlighted in **bold** below) will extract the styling information from the **currentStyle** object. The JavaScript behind the design dialog will store the information entered in the design dialog in the design. This design information populates the **currentStyle** object.

hero.js

```
"use strict";
use(["/libs/wcm/foundation/components/utils/AuthoringUtils.js"], function
(AuthoringUtils) {
    var CONST = {
        PROP_TITLE: "jcr:title",
        PROP_REF_HERO_IMAGE: "fileReference",
        PROP_UPLOAD_HERO_IMAGE: "file",
        PROP_UPPERCASE: "uppercase"
    }

    var hero = {}

    //Get the title text
    hero.text = properties.get(CONST.PROP_TITLE)
```

```
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;

    //Check for file reference from the DAM
    var image = properties.get(CONST.PROP_REF_HERO_IMAGE, String.class);
    if(image == "undefined"){
       //Check for file upload
       var res = resource.getChild("file");
       if(res != null){
             image = res.getPath();
       }
    }
    if(image != "undefined"){
       hero.style = "background-image:url(" + image + ");";
    }
    //Add uppercase design to hero text
    if(currentStyle.get(CONST.PROP_UPPERCASE)){
       hero.uppercase = "text-transform:uppercase;";
    }
    return hero;
});
```

9. **Save your changes.**
10. **Navigate to and open /apps/training/components/structure/hero/hero.html**
11. **Using code from the Exercise_Files for this module, paste in the new contents of hero.html. Do not copy from this exercise book. The following is for illustration purposes only. Notice that the new code (highlighted in bold below) will render the text using the styling information returned by hero.js.**

hero.html

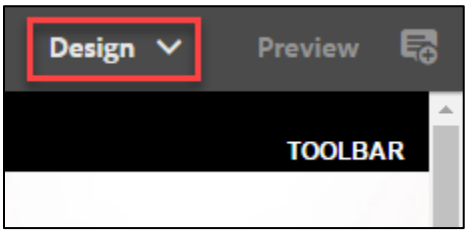
```
<div data-sly-use.hero="hero.js" class="we-HeroImage width-full ratio-16by9"
style="${hero.style @ context='styleString'}">
        <div class="container cq-dd-image">
            <div class="we-HeroImage-wrapper">


                <strong class="we-HeroImage-title" style="${hero.uppercase @
context='styleString'}">${hero.text}</strong>


            </div>
        </div>
    </div>
```

12. Save your changes.

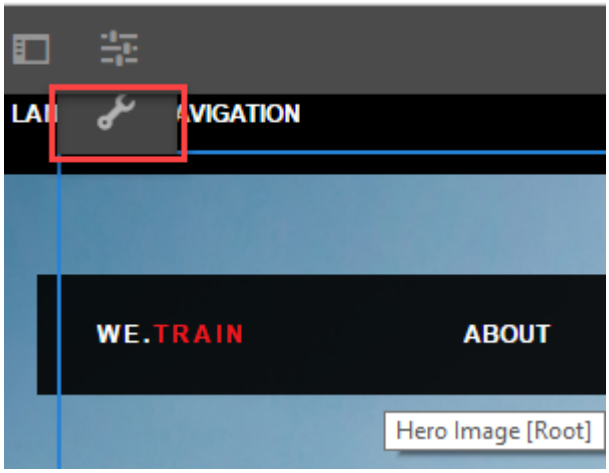
We can now test our design dialog and code.

13. **Using the Sites Console, navigate to or refresh Sites > We.Train > English.**

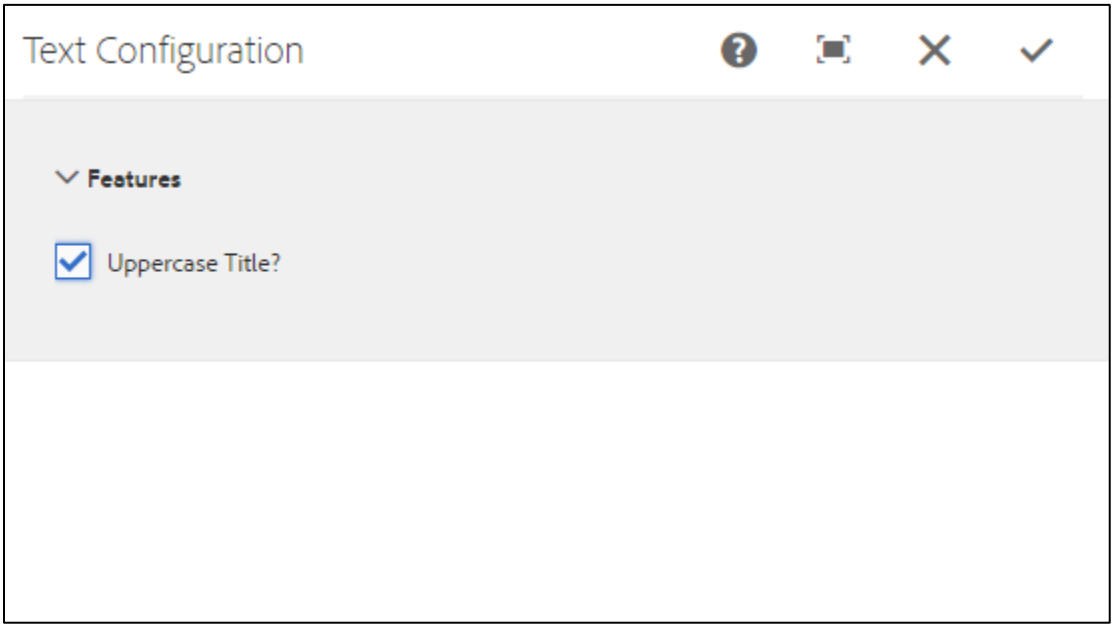
14. **Using the mode dropdown at upper-right, select Design mode:**



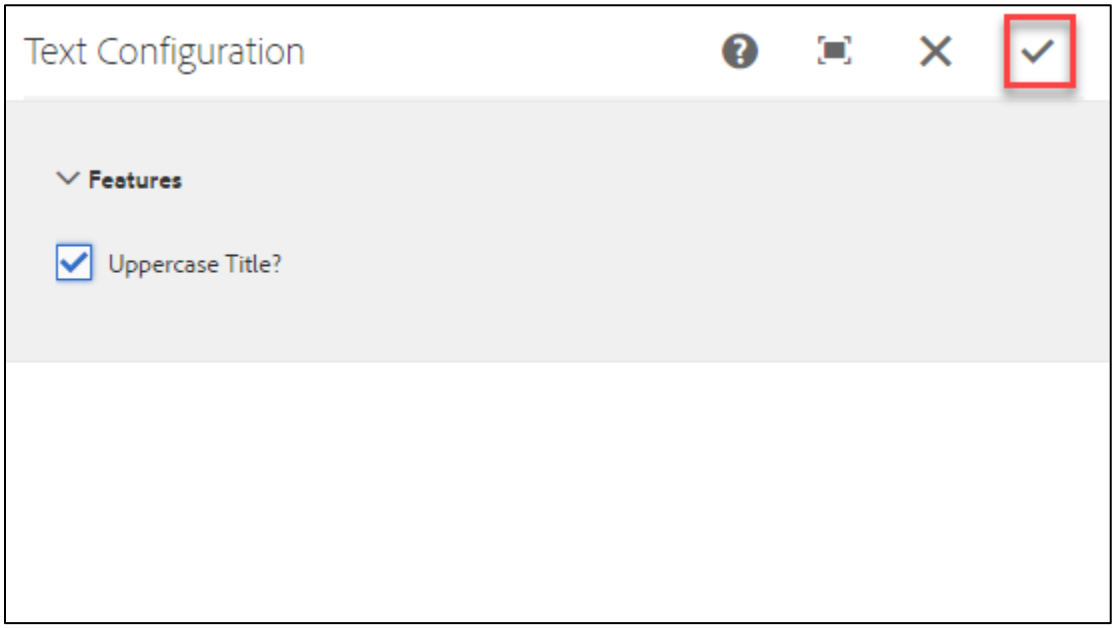
15. **Click on the Hero component**
16. **Click on the wrench to render the dialog.**



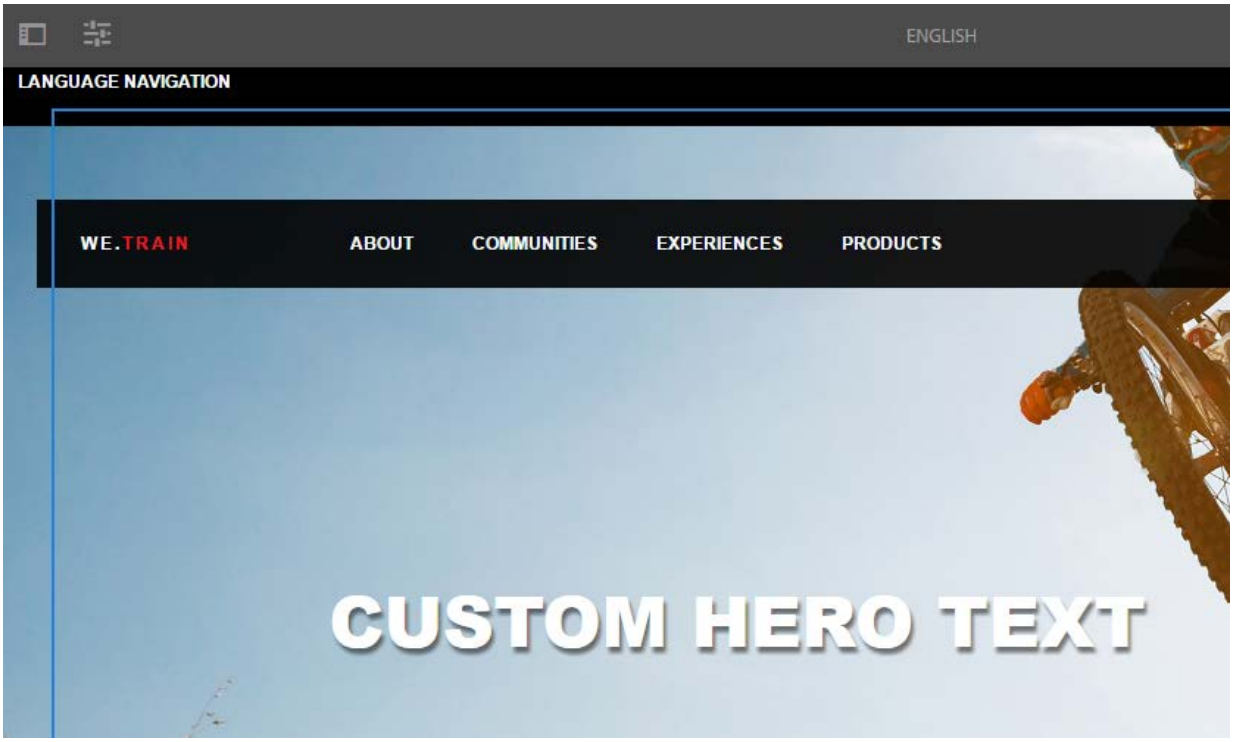
17. **Click to expand Features and select Uppercase Title?**



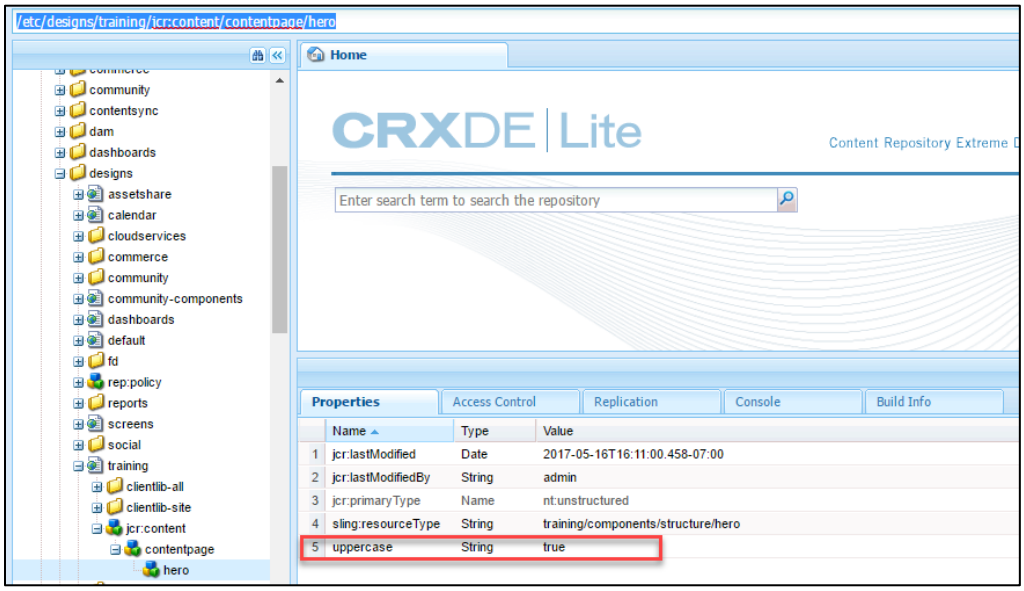
18. **Click Done to save your changes.**



19. **Check that your hero component text is now in uppercase:**



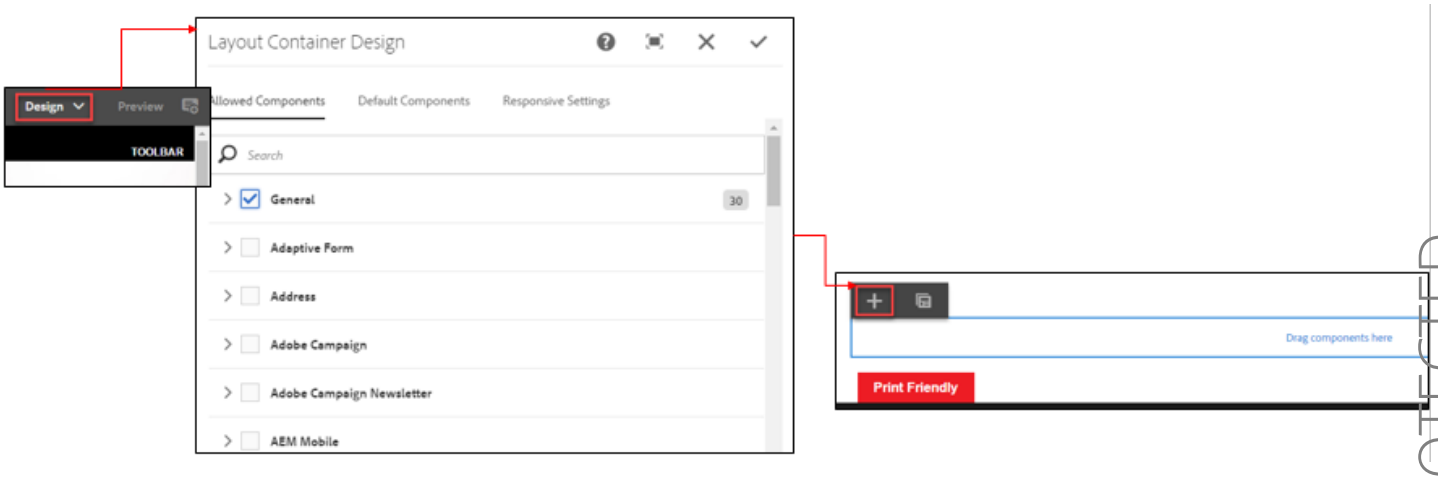
20. Check that the value is persisted. In CRXDE Lite, navigate to **/etc/designs/training/jcr:content/contentpage/hero.** Check that a property with the value **true** was added:



11 THE RESPONSIVE GRID

The Responsive Grid in AEM

The purpose of the Responsive Grid is to provide a way for authors to drag and drop (or add) the components they will need onto a page. The Responsive Grid appears as a "Layout Container" component in AEM. As a developer you may control the types of components you want to make available to authors to use by using Design Mode.



Responsive Design

The website you created is optimized to view on desktops and laptops. However, your users will want to view your website on handheld devices, such as tablets and mobile phones. Therefore, you need to ensure that the website is optimized to view on all devices to provide your users with a better experience and consistency across devices and screen sizes. The guideline with responsive design is that the site should be build for "mobile web"; that is, it will change and respond to fit any screen size by using client-side feature detection via media queries. AEM contains both client-side and server-side device detection libraries.

Responsive Design allows you to create sites that provide an optimal viewing experience across various mobile devices. You are provided with emulators in AEM to view the site on various devices.

Responsive design has become a common practice for all websites today.

Goals of Responsive Design

- Easy reading and access to content
- Clear and easy navigation
- Minimum of resizing, panning, and scrolling
- Design for a wide range of devices (desktop and mobile) with varying:
    - Screen sizes
    - Memory capacity
    - Network speeds
    - CPU speeds

Responsive Design is not a single piece of technology or tool, but rather, a collection of techniques and ideas that allow the site to identify and then respond to the browsing environment or device through which they are being viewed.

Tenets of Responsive Design

- ✓ A single design for all devices to ensure a consistent experience across devices
- ✓ A single URL for all devices
- ✓ A single code base for your website
- ✓ Less expensive to maintain

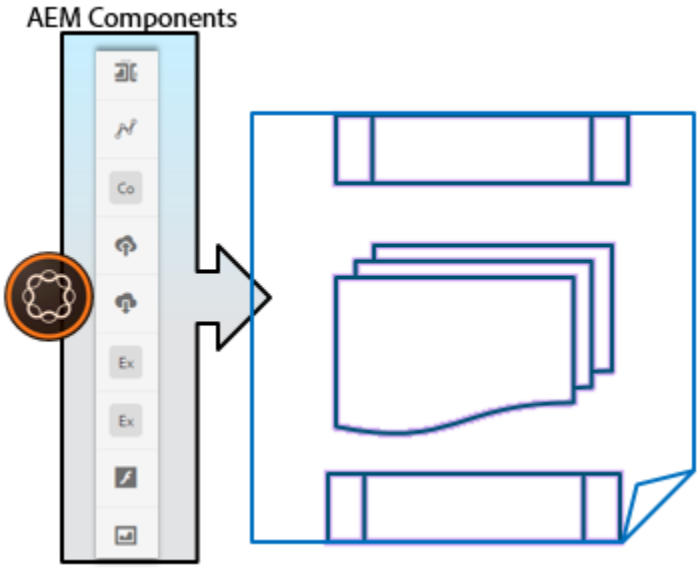
The Responsive Paragraph System in AEM

The purpose of a paragraph system is to organize content on a page. Goal is to aid the author in dragging components (content) onto a page, or adding components to specified areas.

Each area of the page can be thought of as a "paragraph":

- Header components
- Layout Container (Responsive Grid)
- Footer components

To put this into context, you can think of a "paragraph system" as a general term for the area where you drag paragraphs (components) into. If you think of the page like a paper, each paragraph contains an idea that can be edited, moved around, cut, copied, and deleted. The intro paragraph lays out what will happen in the paper (header components) and the conclusion sums up the paper (footer components). The paper's body (paragraph system) really makes/breaks the paper, it contains the substance of the paper and is managed by whoever is in charge of the content in the paper (author).



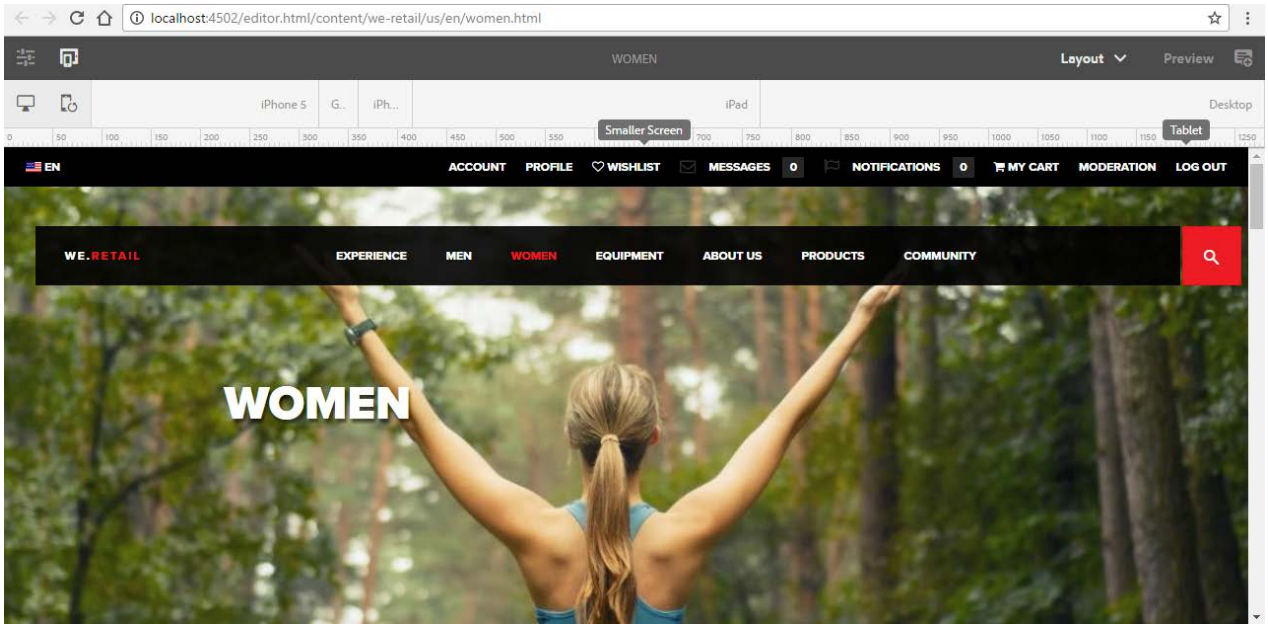
AEM has three out of the box components that provide a paragraph system:

1. `parsys` (Paragraph System)
2. `iparsys` (Inherited Paragraph System)
3. `responsivegrid` (Layout Container, aka "Responsive Grid")

As a best practice, the responsivegrid should be used to maximize the Touch-Optimized UI and provide flexibility to authors. The parsys and iparsys are AEM's original legacy paragraph system. While they can be used, they have to explicitly made responsive, and therefore the work involves more design effort.

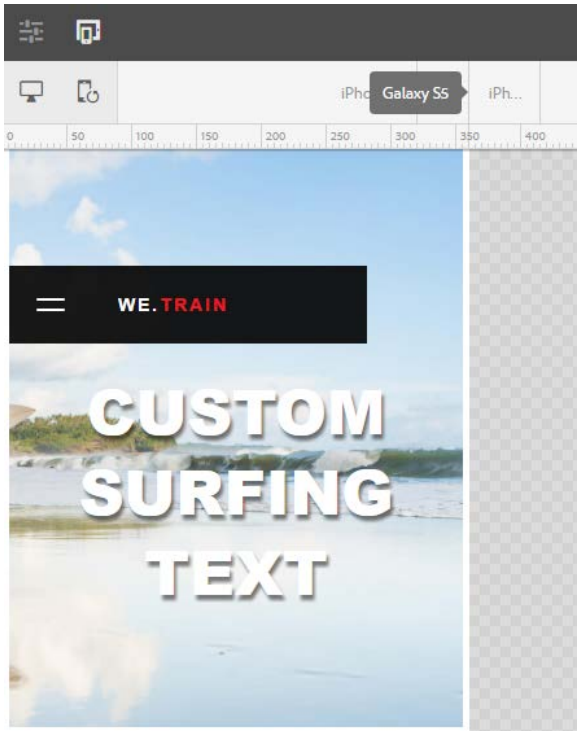
Layout Mode

Layout mode allows you to create and edit your responsive layout and simulate its appearance on devices. Layout mode is only available with the `responsivegrid` component. Use the cq:responsive node from We.Retail to enable Layout mode. Authors use Layout mode to position content on the page within the responsive grid.



Emulators

AEM comes with an emulator feature that allows you to simulate how devices/screens resize your content. The emulator example shown below allows you to mimic what a user on a Samsung Galaxy S5 would experience:

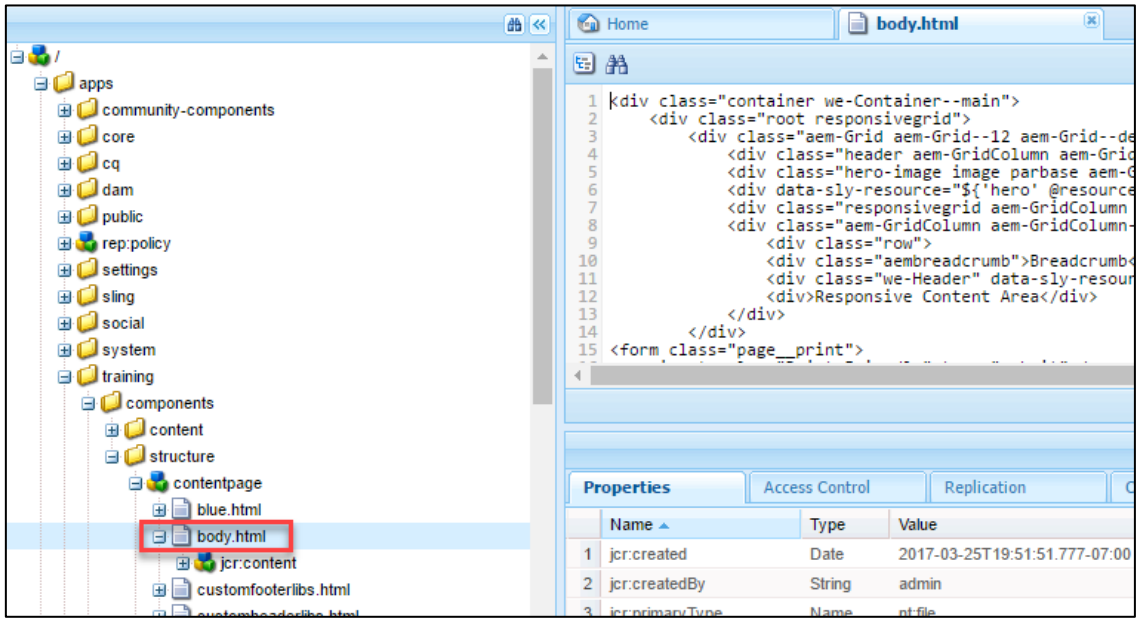


Exercise

11.1.1 Task – Add the Responsive Grid Container

This will eventually become the free content area where authors can add new content to the responsive page.

1. **Using CRXDE Lite, navigate to and open /apps/training/components/structure/contentpage/body.html:**



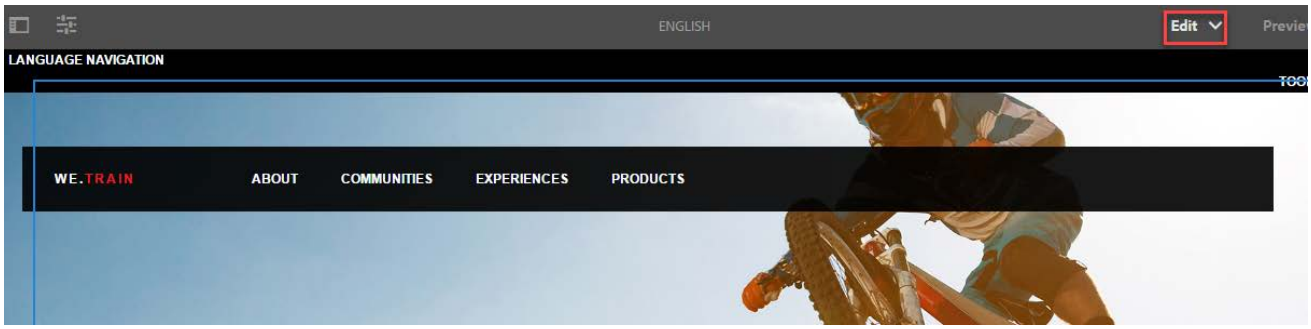
2. **Using code from the Exercise_Files for this module, paste in the new contents of body.html. Do not copy from this exercise book. The following is for illustration purposes only. Notice that the new HTL code (highlighted in bold below) includes the responsive grid container:**

body.html

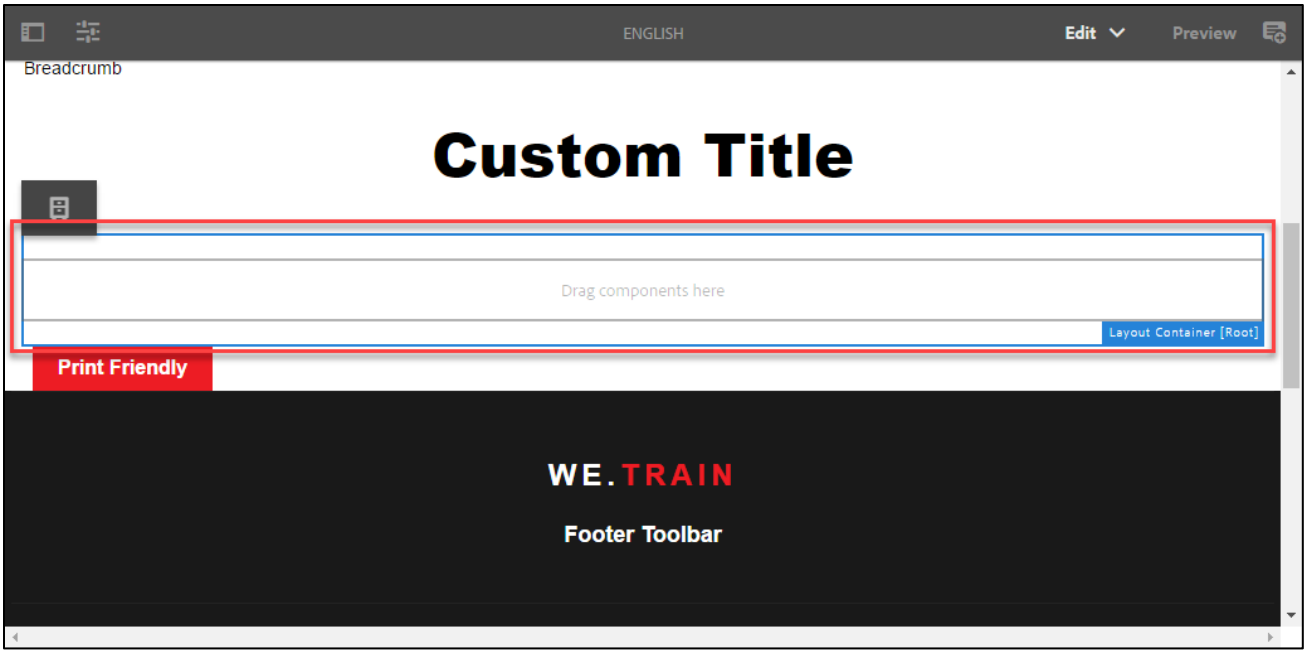
```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12" data-
sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-GridColumn--
default--12">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--default--
12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div class="aem-
breadcrumb">Breadcrumb</div>
                            <div class="we-Header" data-sly-resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div data-sly-resource="${'responsivegrid' @
resourceType='wcm/foundation/components/responsivegrid'}"></div>
```

```
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--default--12"
data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

3. **Save your changes.**
4. **Using the Sites Console, open or refresh Sites > We.Train > English in Edit mode:**



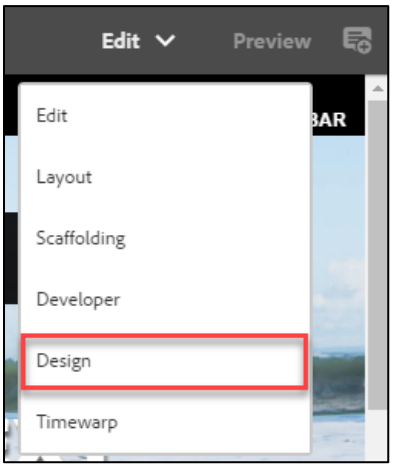
5. **Scroll down and notice how there is a new Layout Container available:**



11.1.2 Task – Add Components to the Responsive Grid

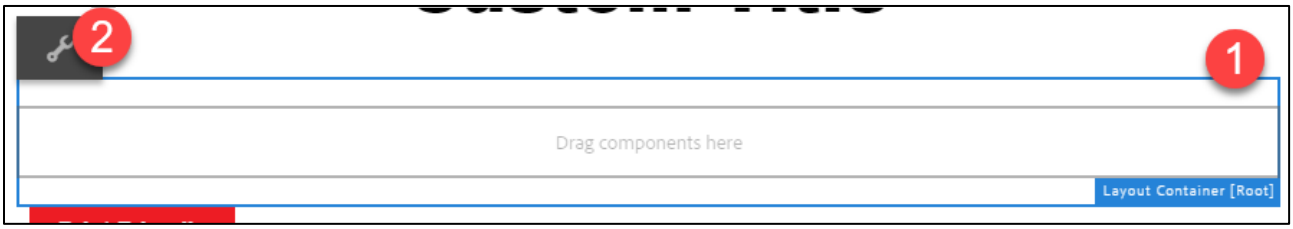
At this stage, you may notice you cannot actually add any components in Edit mode. So, now you need to enable the components you want to be made available for authors to drag and drop into the Responsive Grid.

1. **Select Design Mode at upper-right:**

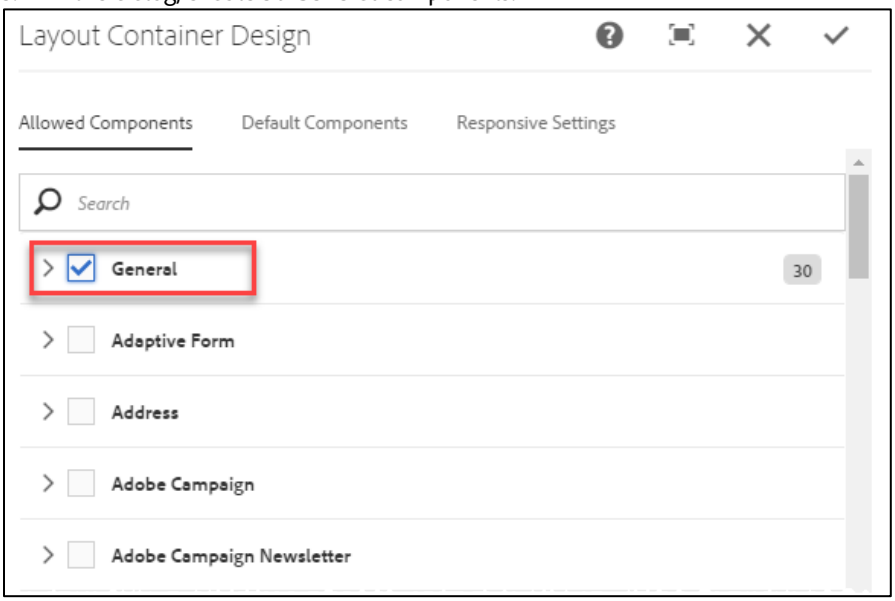


2. Select the **Layout Container** (1) and then click **Configure** (2).

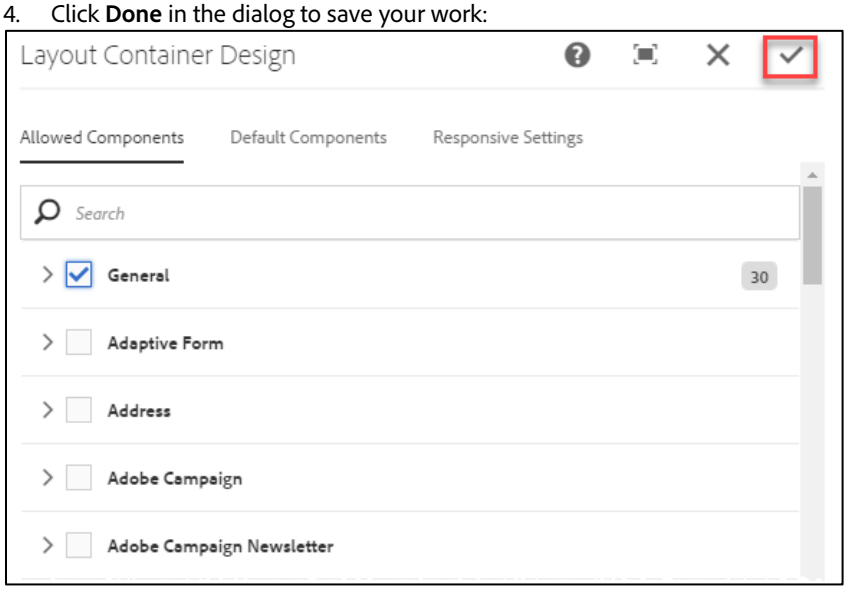
NOTE: You need to select the outer "root" to enable the "wrench":



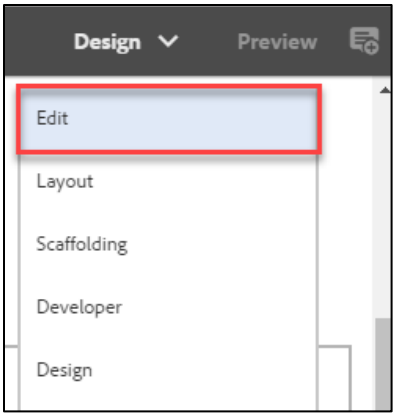
3. In the dialog, enable 30 **General** components:



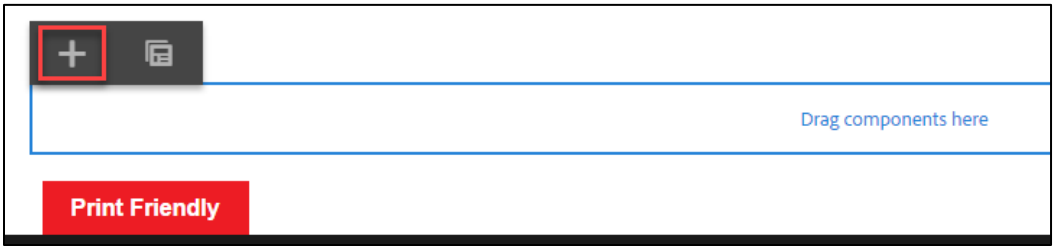
4. Click **Done** in the dialog to save your work:

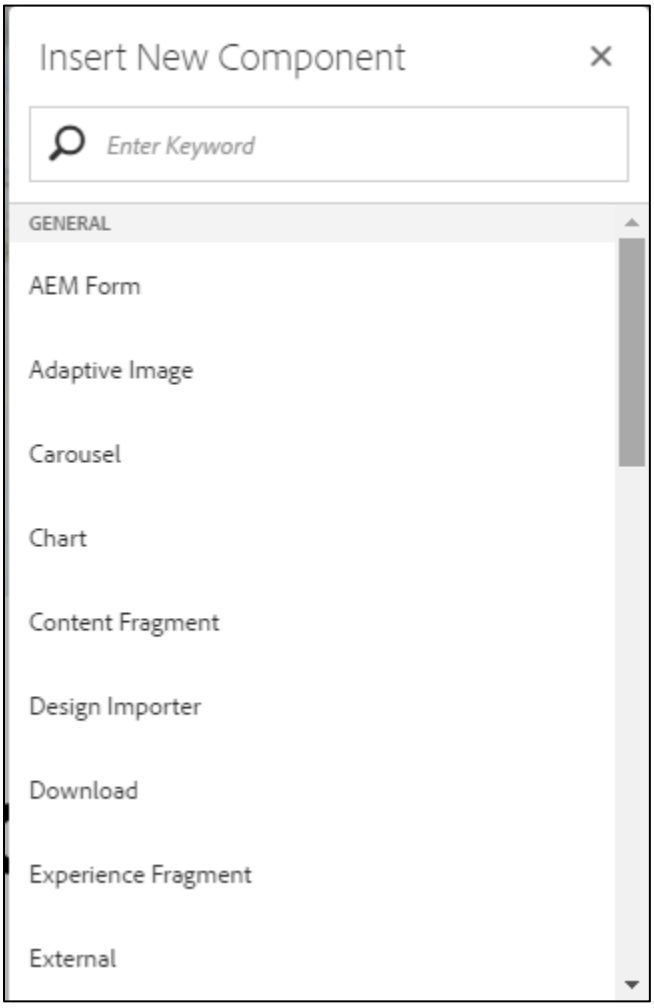


5. Switch back to **Edit** mode:



6. Click the "inner" layout container and confirm you can add a component:

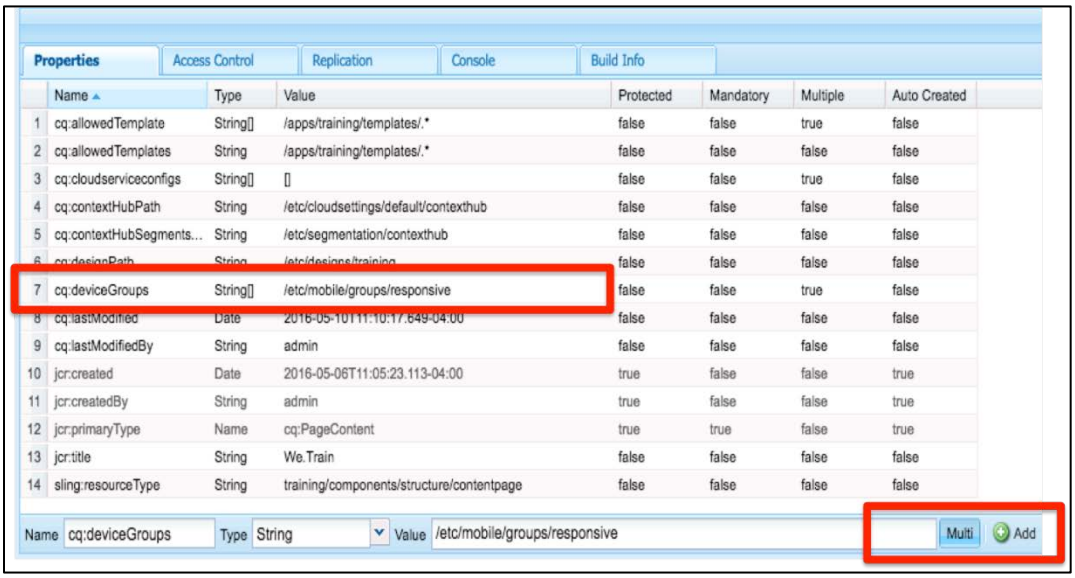




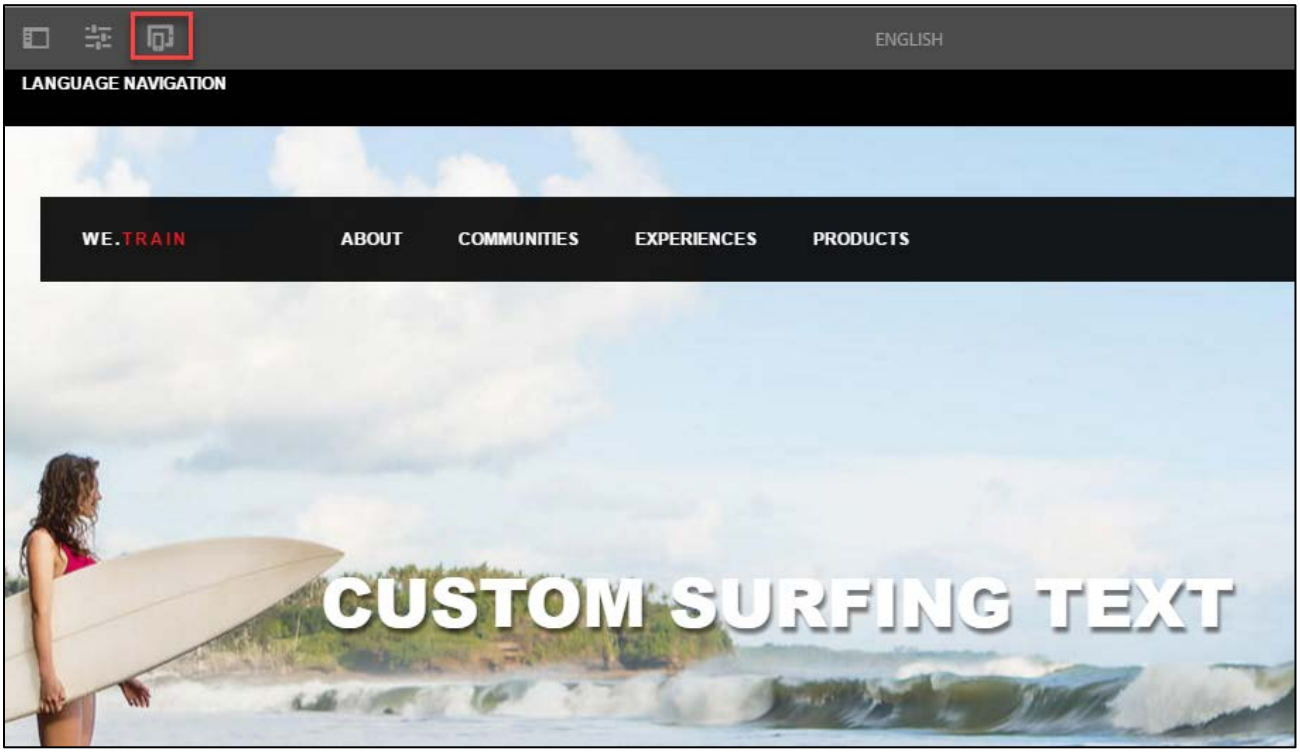
11.1.3 Task – Enable the Responsive Emulator

1. **Navigate to /content/we-train/jcr:content.**
2. **Enable the emulator for We-Train and specify the device groups that will appear in the devices list by adding the following property to the jcr:content node:**

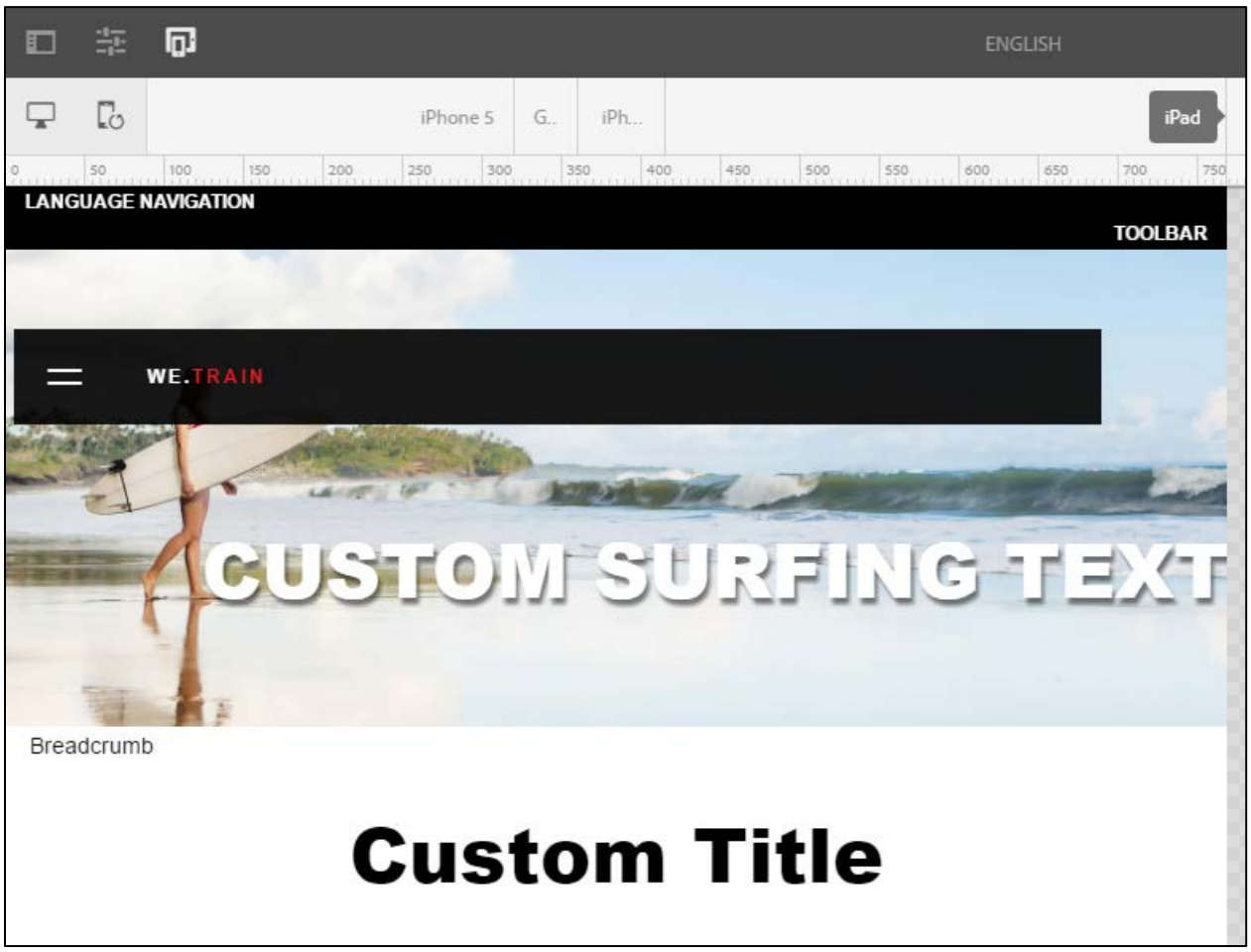
Name	Type	Value
cq:deviceGroups	String[] *(Multi)*	/etc/mobile/groups/responsive



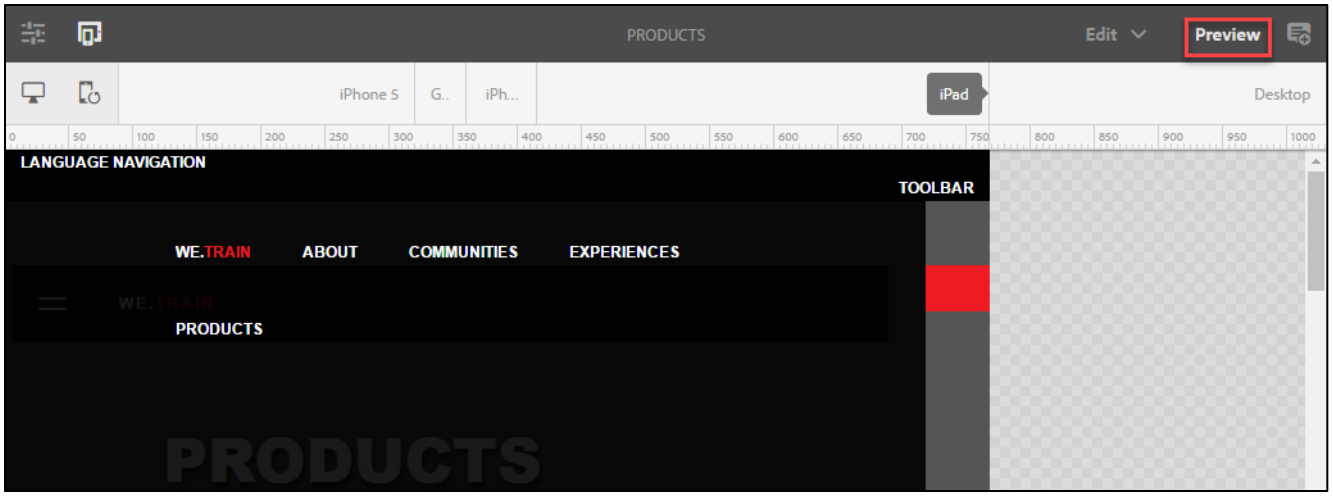
3. **Save your changes.**
4. **Using the Sites Console, navigate to or Refresh Sites > We.Train > English.**
5. **Click on the Emulator icon:**



6. In the emulator, you can select the various devices for which you want to view the page. The following example shows the English page as seen on an iPad device. This multi-screen experience helps you to optimize your CSS further.



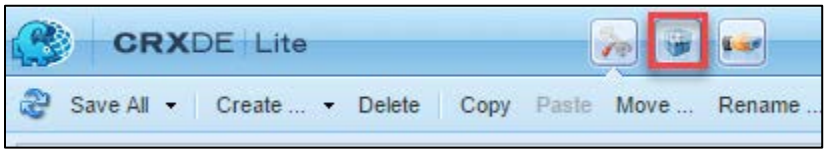
7. Switch to **Preview** mode to simulate the navigation on an iPad:



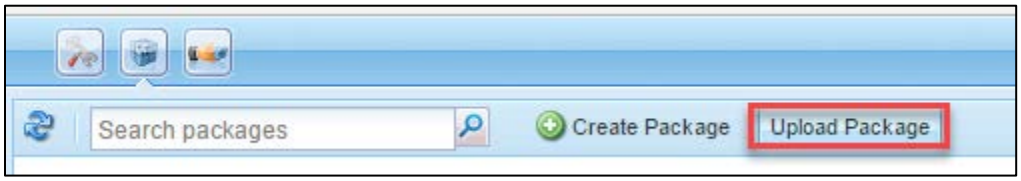
11.1.4 Task – Use a Custom Emulator

In addition to using the emulator provided with Adobe Experience Manager, you may use a custom responsive emulator. In this exercise step we will import an emulator for a **Samsung Galaxy S7**, which is not provided with AEM 6.3.

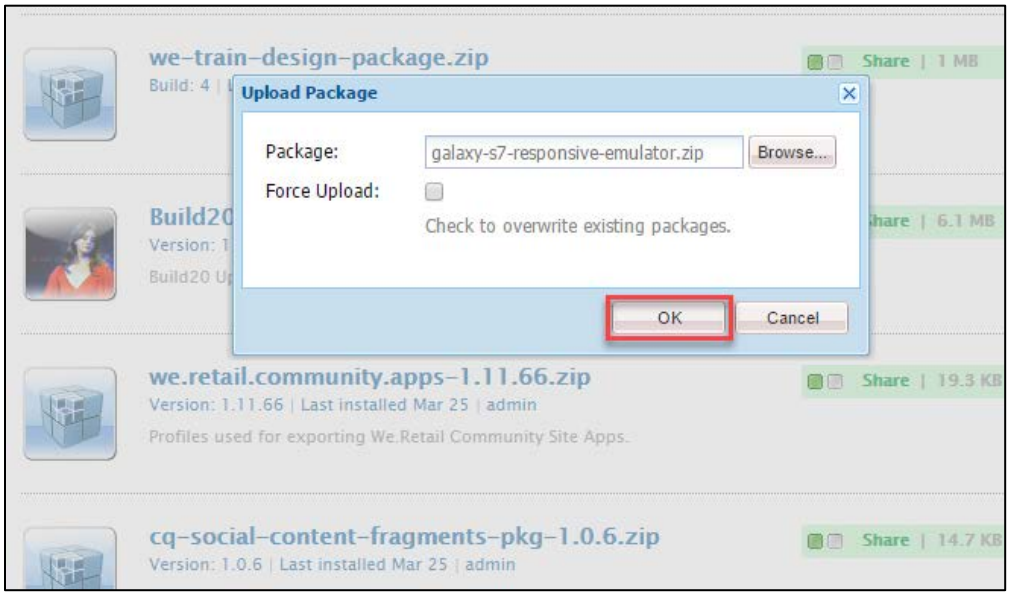
1. **Navigate to the Package <anager by clicking the package button in CRXDE Lite:**



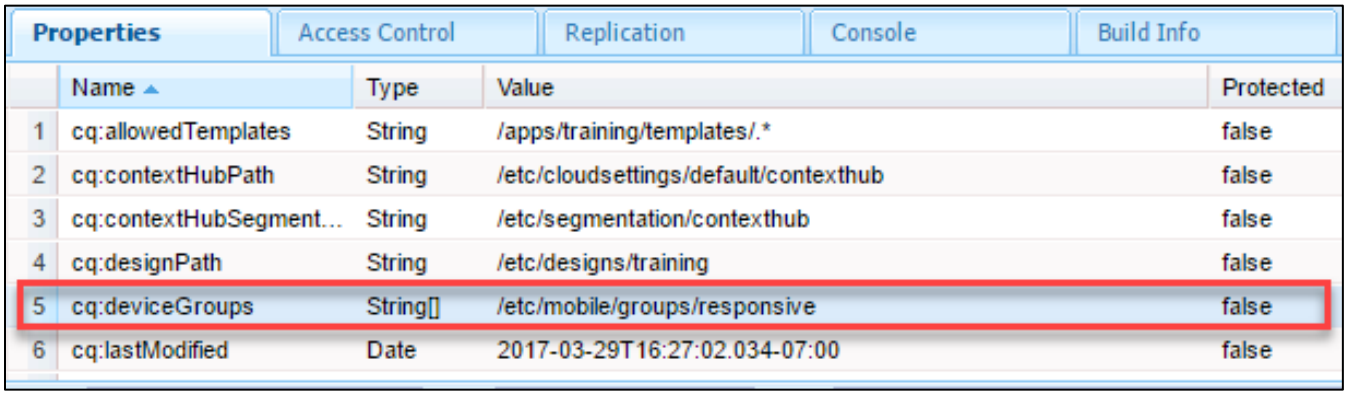
2. **Upload and Install the galaxy-s7-responsive-emulator.zip package provided in the Exercise_Files for this Module (\Module 11 Responsive Grid\):**



NOTE: If you need a refresher on how to install a package, refer to Exercise 4.1.2 in the **Developer Tools** module. This exercise assumes you know how to install a package.

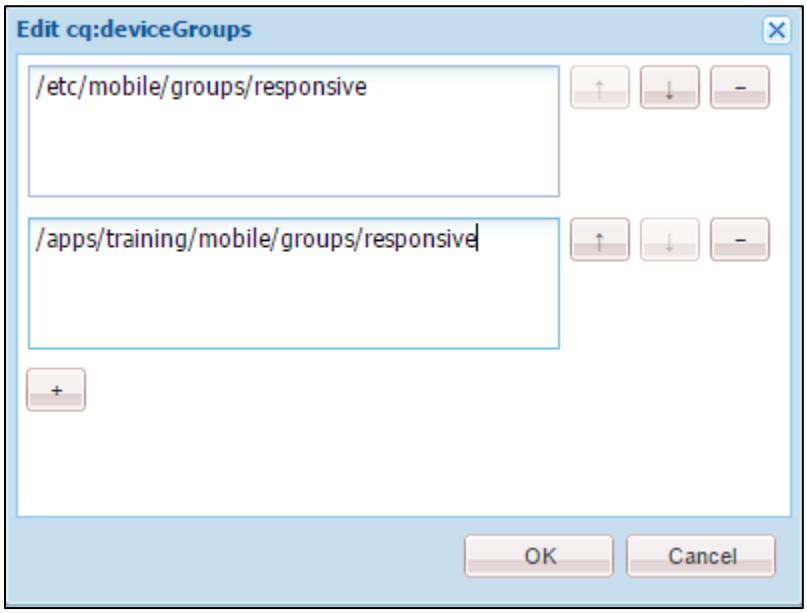


3. **Navigate to the /content/we-train/jcr:content node.**
4. **Double-click the following property in the jcr:content node:**



	Name	Type	Value	Protected
1	cq:allowedTemplates	String	/apps/training/templates/.*	false
2	cq:contextHubPath	String	/etc/cloudsettings/default/contexthub	false
3	cq:contextHubSegment...	String	/etc/segmentation/contexthub	false
4	cq:designPath	String	/etc/designs/training	false
5	cq:deviceGroups	String[]	/etc/mobile/groups/responsive	false
6	cq:lastModified	Date	2017-03-29T16:27:02.034-07:00	false

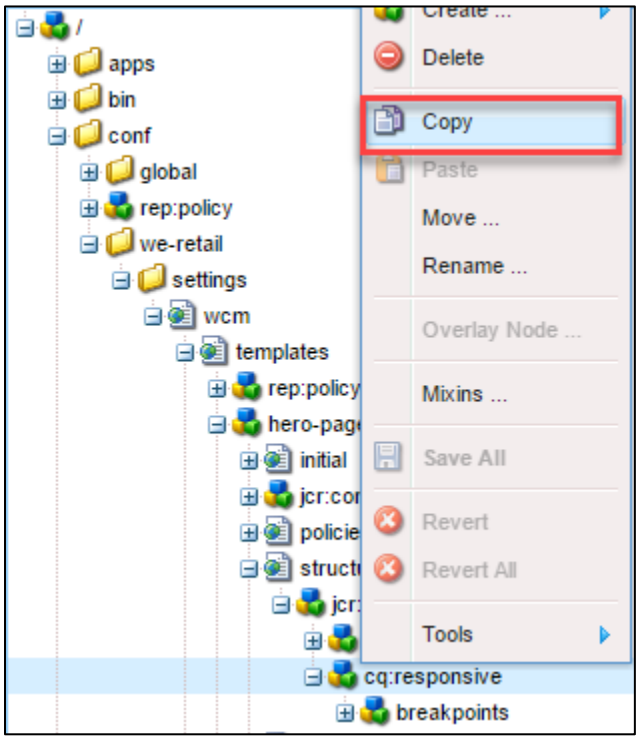
5. **Add the following value. Recall we can do this because the type is an array (String[]):**



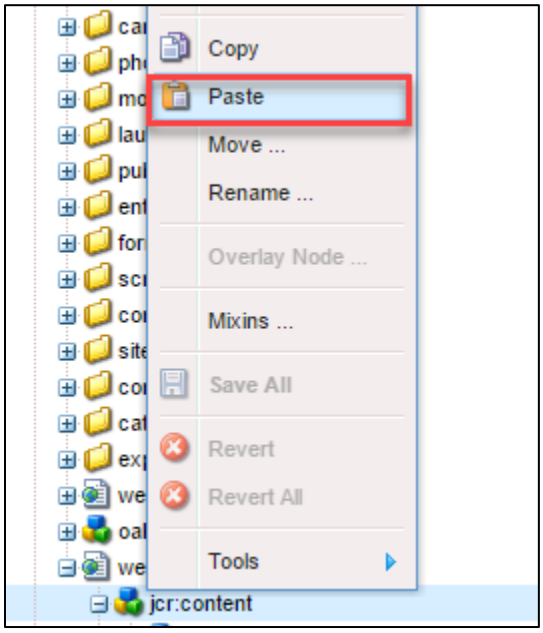
6. **Click Ok.**
7. **Save your changes.**
8. **Using the Sites Console, Refresh Sites > We.Train > English.**
9. **Your emulator should now include a Galaxy S7. To check for this, click Select device:**

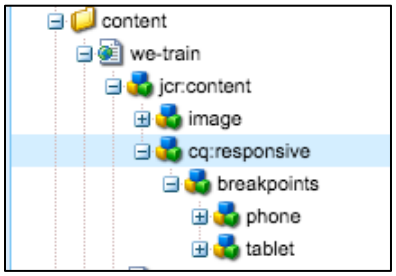
11.1.5 Task – Integrate Layout Mode

1. **Using CRXDE Lite, navigate to /conf/we-retail/settings/wcm/templates/hero-page/structure/jcr:content/.**
2. **Copy the cq:responsive node:**



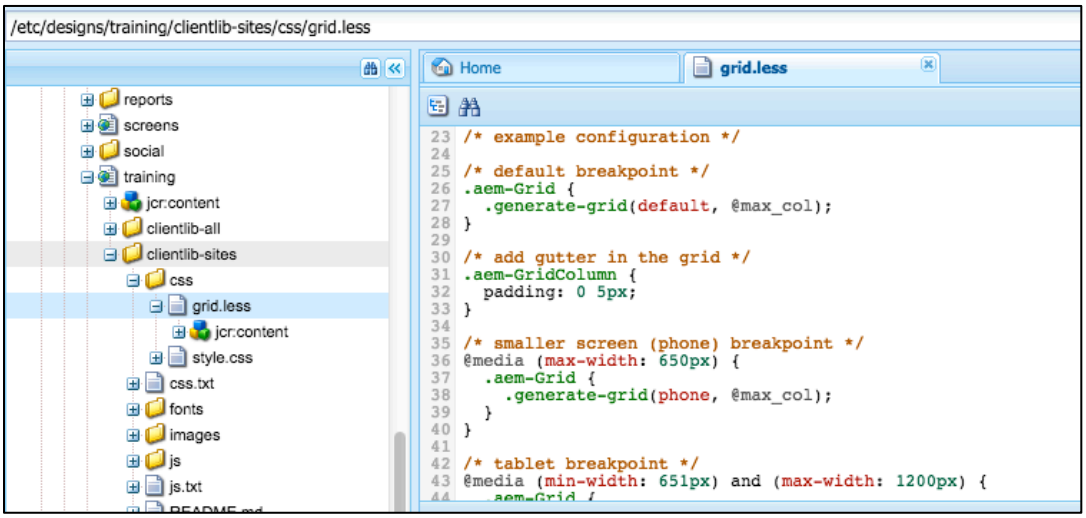
3. **Navigate to /content/we-train/jcr:content and paste the cq:responsive node.**





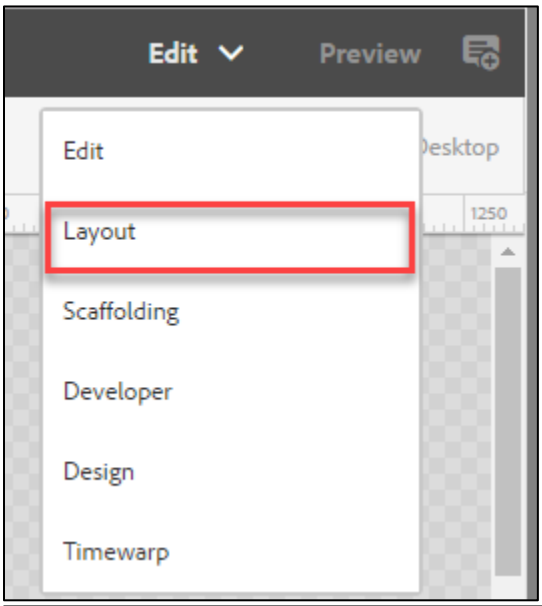
We already added the responsive grid, but need to manage styling to allow authoring of responsive content.

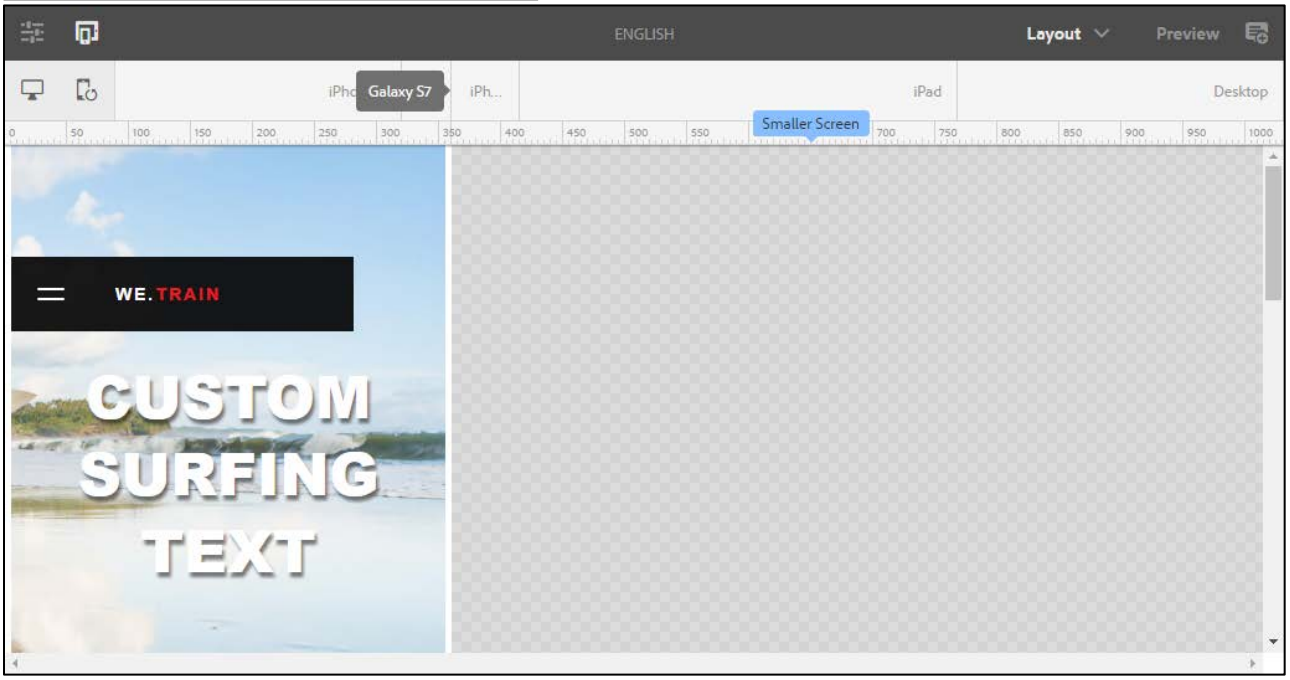
4. **Navigate to /etc/designs/training/clientlib-site/css.**
5. **Open grid.less and investigate the contents.**



6. **Using the Sites Console, navigate to or refresh Sites > We.Train > English.**

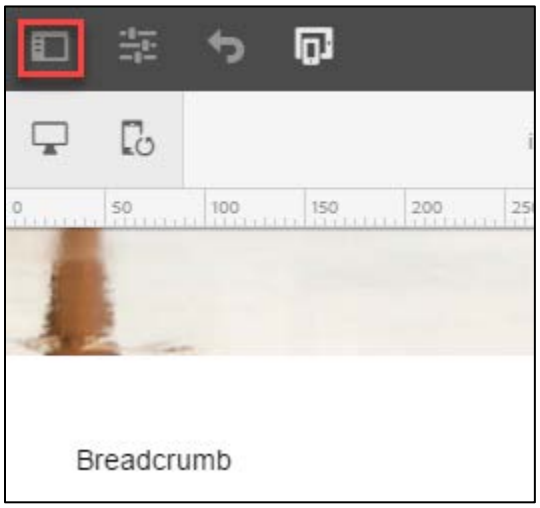
7. **Notice how Layout mode is now available.**



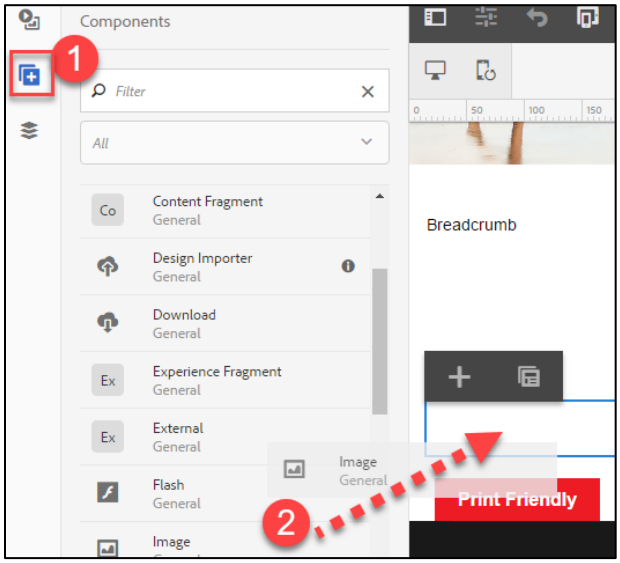


11.1.6 Task - Authoring a Responsive Page with Layout Mode

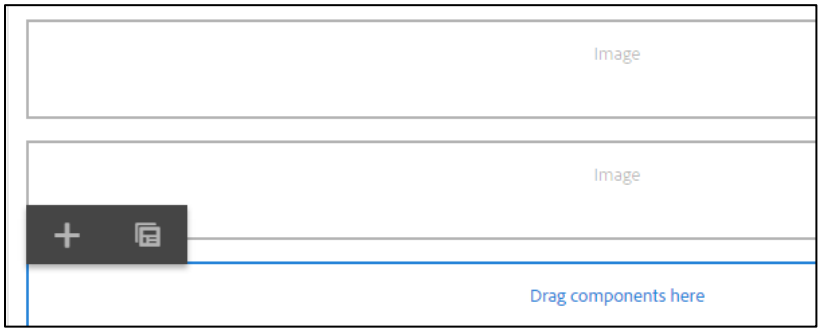
1. Switch back to Edit mode.
2. Enable the Side Panel:



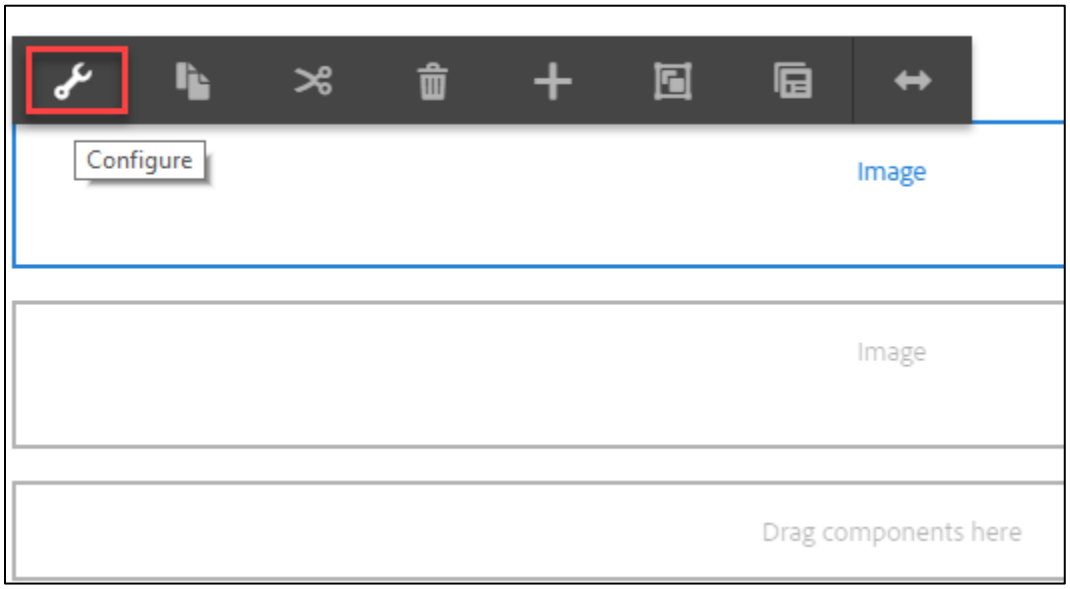
3. Drag and drop an image component to your responsive grid (to Drag components here):



4. Drag and drop another image component again (repeat step 3).
5. Verify your Layout Container looks like this:

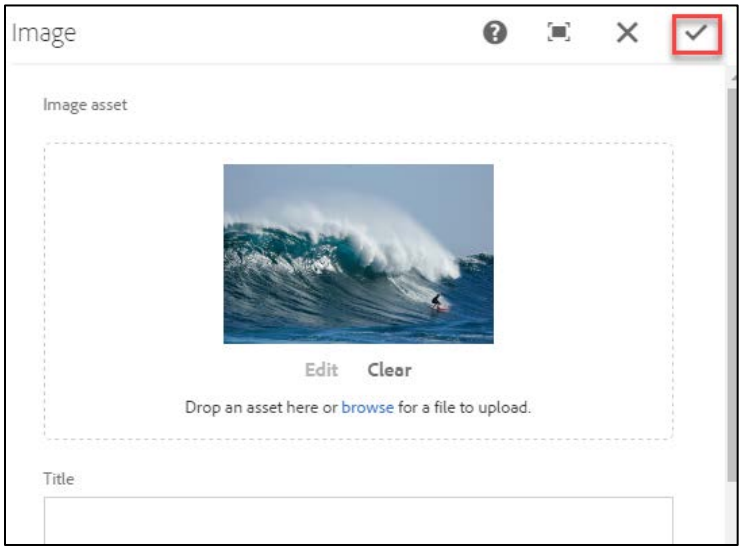


6. **Now, designate two images of your choice for your two image components. With one of your image components selected, click Configure:**

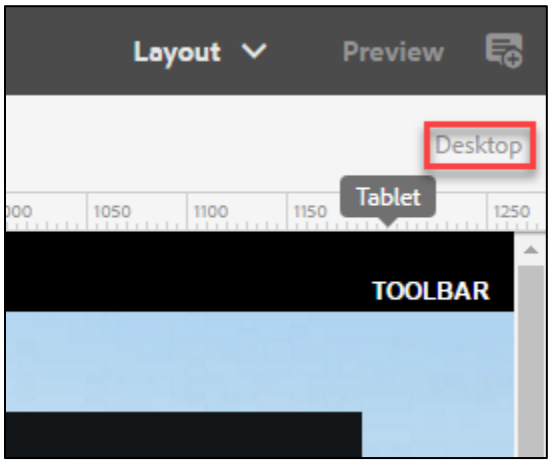


7. **Click browse to upload images of your choice to use here. You may use the images provided for the last module (Module 10 – Developing Structure Components) provided in the Exercise_Files:**
    - **\Module 10 Developing Structure Components\Exercise - V\10.1.17 Task - Modify Hero component to display Image\images for hero component**

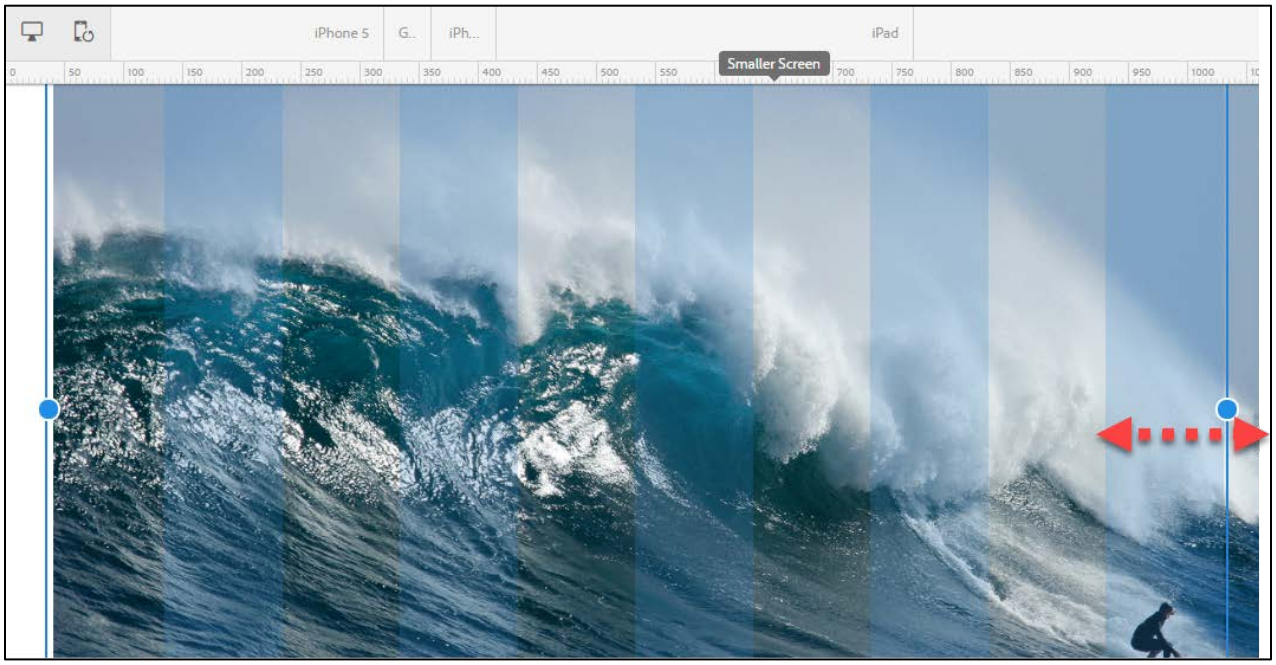
8. **Click Done to save your images:**



9. **Switch to Layout mode and click the Desktop Emulator by clicking Desktop in the upper right:**

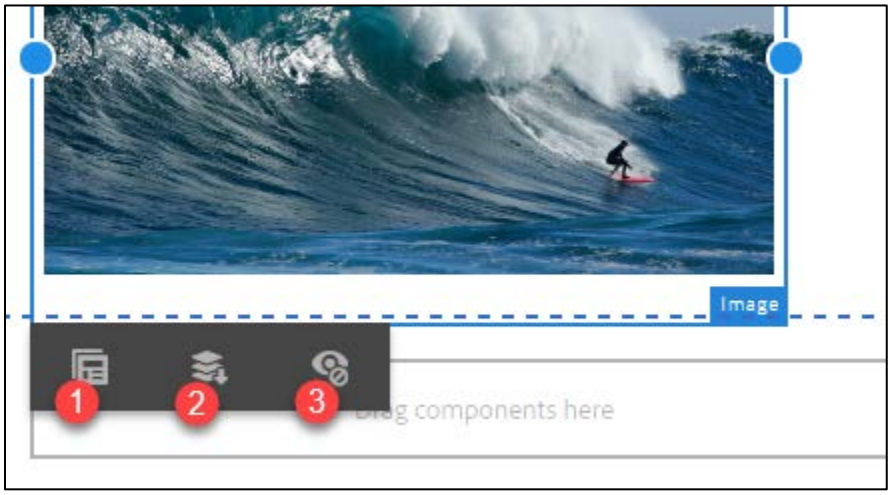


10. **Notice you may now use the "blue dot" to horizontally resize your image components. You can also span the images to a column "grid"as shown here (five columns):**



Lastly, you may also use features to experiment with the grids and layouts, such as:

1. **Parent (move to parent *layout container)***
2. **Float to new line (moves the component to a new line within the layout, and prevents haphazard content flow)**
3. **Hide Component (hides the component for that layout alone (such as iPhone 4))**



12 ADVANCED SLING FUNCTIONALITY

Exercise I– Sling Selectors

12.1.1 Task – Investigate the Print Friendly button

Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.
Using the code from the Exercise_Files, replace the code in body.html.

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12" data-sly-
include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-GridColumn--
default--12">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--default--
12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div class="aem-breadcrumb" data-sly-
resource="${'breadcrumb' @ resourceType='foundation/components/breadcrumb'}"></div>
                            <div class="we-Header" data-sly-resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div data-sly-resource="${'responsivegrid' @
resourceType='wcm/foundation/components/responsivegrid'}"></div>
                        </div>
                    </div>
                    <form class="page__print" action="${currentPage.Path @
selectors='print'}.html">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--default--12"
data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Look at the following statement (line 30 of body.html):

```
…
<form class="page__print" action="${currentPage.Path @ selectors='print'}.html">
              <input value="Print Friendly" type="submit" />
```

```
</form>
...
```

When the user clicks the Print Friendly button, the following statement is sent as a request:

```
${currentPage.Path @ selectors='print'}.html"
```

If the currentPage is /content/we-train/en, then the request is: http://localhost:4502/content/we-train/en.print.html.

12.1.2 Task – Create the print.html script

Think back to the previous discussion in Module 5 of Sling URL decomposition, resource resolution and how Sling finds the rendering script. Recall that when a URL contains a selector, Sling will first attempt to find a script to match that selector.

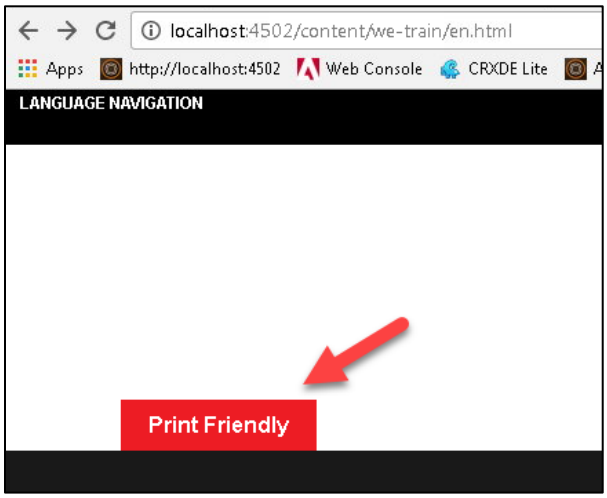
1. **Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.**
2. **Right-click on the contentpage node and select Create… > Create File.**
3. **Name the file print.html, click OK and Save All.**
4. **Using the code from the Exercise_Files, paste the code into print.html.**

print.html

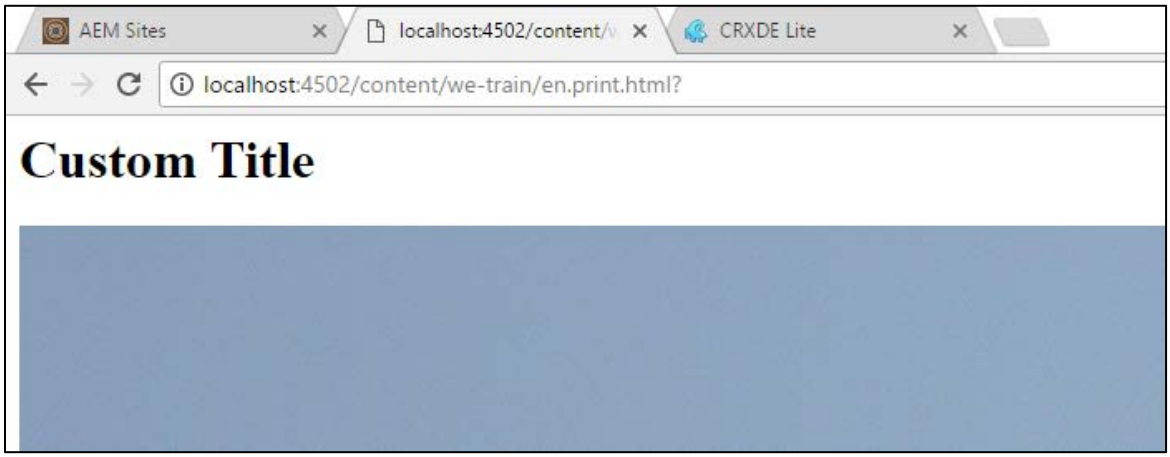
```
<div  data-sly-resource="${'title' @
resourceType='wcm/foundation/components/title'}"></div>

<div  data-sly-resource="${'responsivegrid' @
resourceType='wcm/foundation/components/responsivegrid'}"></div>
```

5. **Using the Sites Console navigate to Sites > We.Train > English or any page with content on it.**
6. **Remove editor.html from the URL. For example, http://localhost:4502/content/we-train/en.html**
7. **Click the Print Friendly button near the bottom of your page.**

NOTE: You may have to scroll down on your page to locate this button, depending on how you added images to your page in an earlier exercise.



8. **Notice that only the items in the Responsive Grid (sometimes called a paragraph system) are displayed and there is no design. If you examine the code in print.html, you will see why the display looks the way it does.**



Overlays and Sling Resource Merger

In Adobe Experience Manager, the UI (user interface) is based on a set of nodes in the JCR repository, and its underlying structure. These nodes reside in the **/libs** folder. When you work on your project, any customization required is done by overlaying the corresponding content node in the repository, under the **/apps** folder. This is done by overlaying the content node in the repository.

There are two methods of performing these customizations:

1. **Overlays**
2. **Sling Resource Merger**

Overlays

Sling's Resource Resolver will search for resources in a list of specific paths, as defined by the Apache Sling JCR Resource Resolver. The default search path is first **/apps** and then **/libs**. As a result, you can change the out-of-the-box functionality as provided in **/libs** by adding a resource or file at the same path in **/apps**. This ability to override default functionality is called an overlay.

To use overlays, follow this general process:

1. Recreate the node structure from **/libs** under **/apps**
2. Copy the content node that you want to change in **/apps**, and then work on this copy.
3. When a resource is requested, it gets resolved according to the search path logic.
4. It retrieves the resource from **/apps**, failing which, it looks at **/libs**.

To summarize, an overlay involves taking the predefined functionality and imposing your own definitions over that to override the standard functionality.

Sling Resource Merger

Sling Resource Merger is applicable only to the touch-optimized UI (Touch UI), and enables you to customize consoles and page authoring. With Sling Resource Merger, you can overlay nodes without replicating all of their ancestors. It uses diff mechanisms along with resource resolver search paths to merge overlays of resources.

The Sling Resource Merger provides services to access and merge resources. It merges overlays of resources using resource resolver search paths and diff mechanisms. A customized sling vocabulary is used to use the resource merger, allowing you to manage overrides of nodes and their properties. With this, the overlay resources/properties (defined in /apps) are merged with the original resources/properties (from /libs).

Sling Resource Merger is used to ensure that all changes are made in /apps, thus avoiding the need to make any changes in /libs. Once the structure is created, you can add your own properties to the nodes under /apps. The content of /apps has a higher priority than that of /libs. The properties defined in /apps indicate how the content merged from /libs are to be used.

Comparing Overlays with Sling Resource Merger

The following table describes the basic difference between using Overlays and Sling Resource Merger. However, it must be noted that both concepts are supported in AEM 6.3.

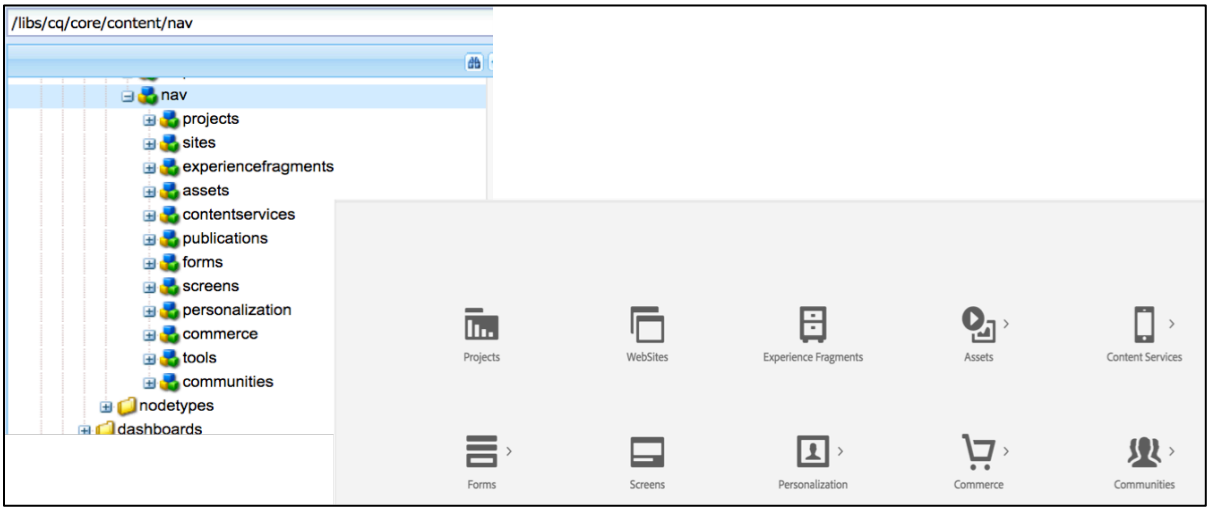
Overlays	Sling Resource Merger
Based on search paths (/libs + /apps)	Based on Resource Type Hierarchy
Need to copy the whole subtree	Extends within an almost empty subtree
All the properties are duplicated	Only required properties are overlaid

When upgrades are done to the /libs folder, these changes have to be manually recreated under /apps.	As properties are not copied, only the structure, upgrades are automatically reflected in /apps.

NOTE: After you copy a system file from /libs to /apps to overlay it with custom code, your custom code will not pick up any modifications to the system component/file/script/dialog box, which result from the application of a hotfix, feature pack, or upgrade. A careful examination of the release notes of any upgrade, feature pack, or hotfix should be a step in your upgrade plan. This way, you will be able to evaluate any changes and make a plan for incorporating them into your application.

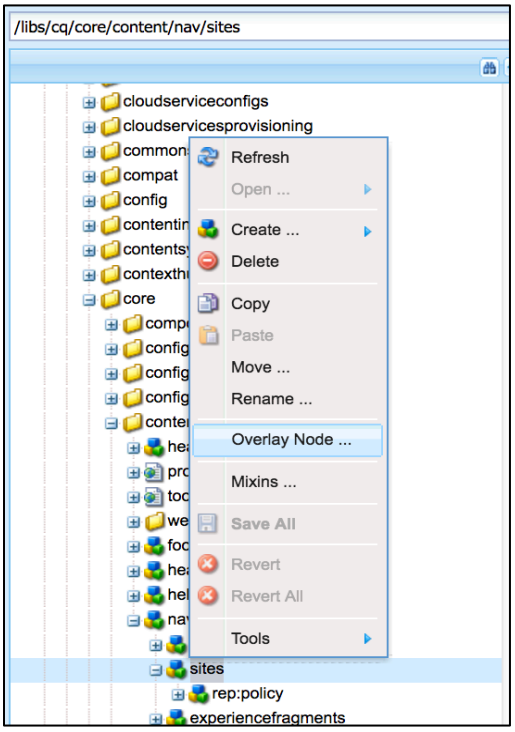
12.1.3 Task – Modify a Navigation Button

1. **Using CRXDE Lite, navigate to /libs/cq/core/content/nav.**
2. **Expand the nav node. You will notice that the child nodes of nav are the definition of the global navigation buttons (consoles) for AEM.**

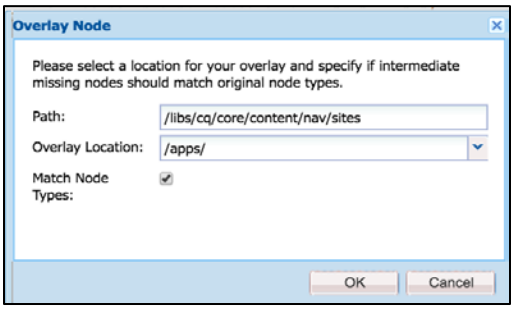


To modify the navigation icons, hide icons, or add a new icon, you need to change the list under **cq/core/content/nav**. Since you know never to modify **/libs** you need a way to make the change in **/apps**. This is where the Sling Resource Merger comes in.

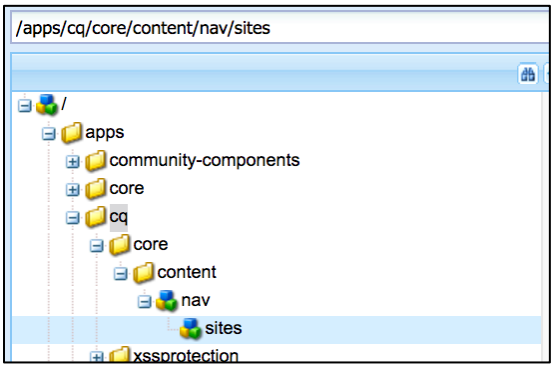
3. **Right-click on /libs/cq/core/content/nav/sites.**
4. **Select Overlay node … from the context menu.**



5. **In the pop-up dialog, make sure that the Overlay Location is /apps/.**



6. **Select Match Node Types and click OK.**
7. **Save.**
8. **Navigate to /apps/cq and expand the cq node all the way to the bottom.**



Notice that **/apps/cq/core/content/nav/sites** has been created.

NOTE: Refresh the /apps path if necessary to see the change.

NOTE: The overlay node structure has been reproduced, but the properties have not been created. Remember, the Sling Resource Merger will merge properties, as well as nodes.

9. **Add the following property to the /apps/cq/core/content/nav/sites node:**

Name	Type	Value
jcr:title	String	Websites

10. **Save.**
11. **Using the Global Navigation, notice that the Sites button is now titled "Websites".**

12.1.4 (Optional) Extra Credit Task – Hide Nav Button

1. **Using CRXDE Lite, navigate to /apps/cq/core/content/nav.**
2. **Right-click on nav and select Create > Create Node**

Name	Type
communities	nt:unstructured

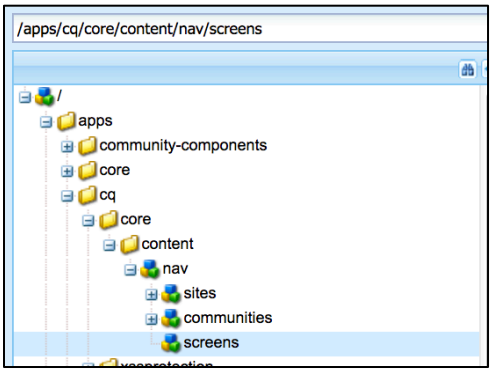
3. **Save.**
4. **Add the following property to the communities node:**

Name	Type	Value
sling:hideResource	Boolean	true

5. **Save.**
6. **Using Global Navigation, notice that the Communities icon has disappeared.**

12.1.5 (Optional) Extra Credit Task – Add a New Nav Button

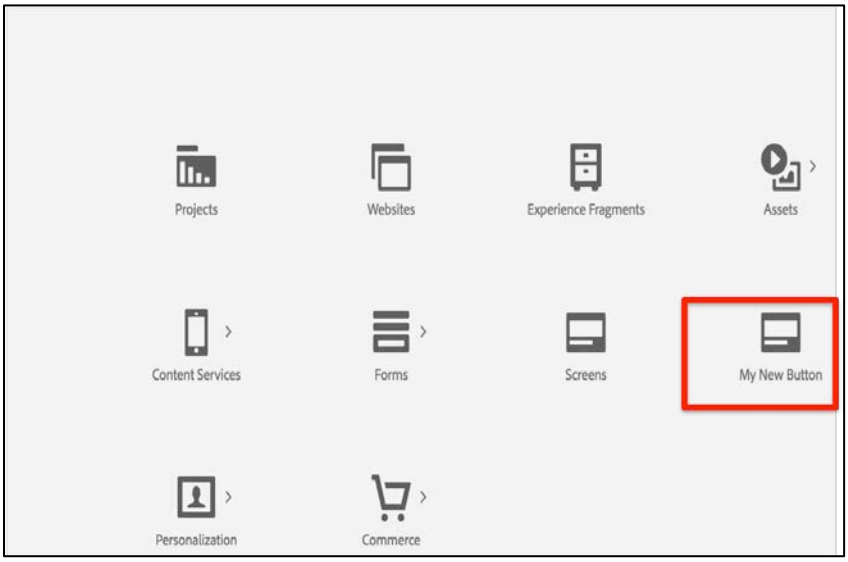
1. **Using CRXDE Lite, navigate to /libs/cq/core/content/nav.**
2. **Right-click on the screens node and select Copy from the context menu.**
3. **Paste the screens node at /apps/cq/core/content/nav.**



4. **Save All.**
5. **Rename the screens node to myButton.**
6. **Save All.**
7. **Modify the following properties of the myButton node.**

Name	Type	Value
jcr:title	String	My New Button
id	String	cq-myButton
href	String	/sites.html

8. **Using Global Navigation, notice how the new icon has shown up.**



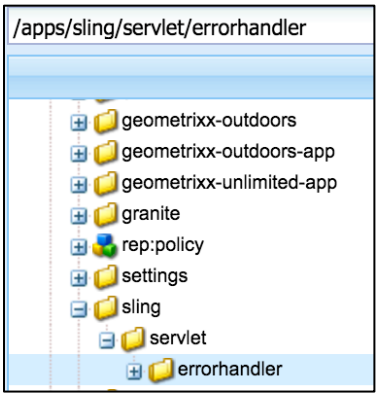
Exercise III– Custom Error Handlers

Another common use case for the Sling Resource Merger is the definition of custom error handlers.

12.1.6 Task – Create a custom 404 Error Handler

1. **Using CRXDE Lite, navigate to /apps/sling.**
2. **Right-click on /sling and select Create… > Create Folder.**
3. **Create the following folder structure:**

sling > **servlet** > **errorhandler**



Tip: Remember AEM is case sensitive.

NOTE: You could also use **Create Overlay …** of **/libs/sling/servlet/errorhandler** from the context menu to create the folder structure.

4. **Save after creating each folder.**
5. **In the errorhandler folder, create two files: 404.html and ResponseStatus.java. Save all.**
6. **Using the code from the Exercise_Files, paste the code into 404.html.**

404.html

```
<html data-sly-use.responseStatus="apps.sling.servlet.errorhandler.ResponseStatus">
<head>
<title>File not found</title>
</head>
<body>
    <p>A custom errorhandler for 404 responses</p>
</body>
</html>
```

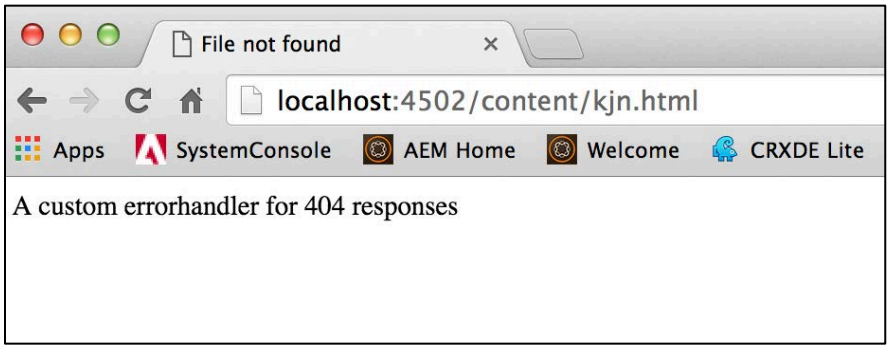
7. **Using the code from the Exercise_Files, paste the code into ResponseStatus.java.**

ResponseStatus.java

```
package apps.sling.servlet.errorhandler;

import com.adobe.cq.sightly.WCMUsePojo;

public class ResponseStatus extends WCMUsePojo {
```

```
    @Override
    public void activate() throws Exception {
        getResponse().setStatus(404);
        getResponse().setContentType("text/html");
    }
}
```

8. **Save your changes.**
9. **Type a non-valid AEM URL into your browser. For example: http://localhost:4502/content/<yourinitials>.html**



12.1.7 Task – Make we-train.html redirect to another page

1. **Using CRXDE Lite, navigate to /content/we-train/jcr:content.**
2. **Define the following three properties on the jcr:content node and save.**

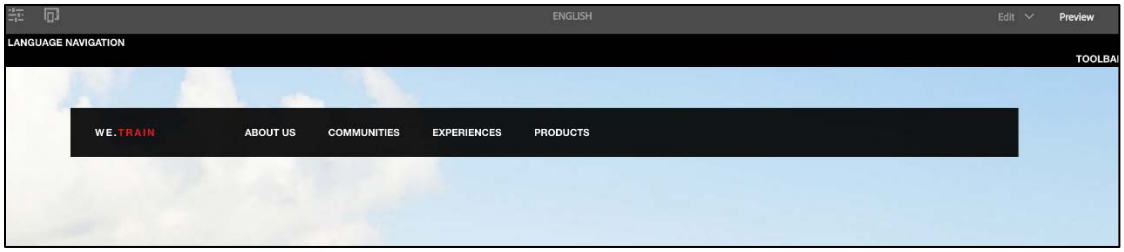
Name	Type	Value
redirectTarget	String	/content/we-train/en
sling:redirect	Boolean	true
sling:redirectStatus	Long	302

3. **Modify the sling;resourceType property to point to the Foundation redirect component.**

Name	Type	Value
sling:resourceType	String	foundation/components/redirect

4. **Using the Sites Console, navigate to Sites > We.Train. This should be the result:**

5. **Switch to Preview mode and reload the page. The English page should load.**

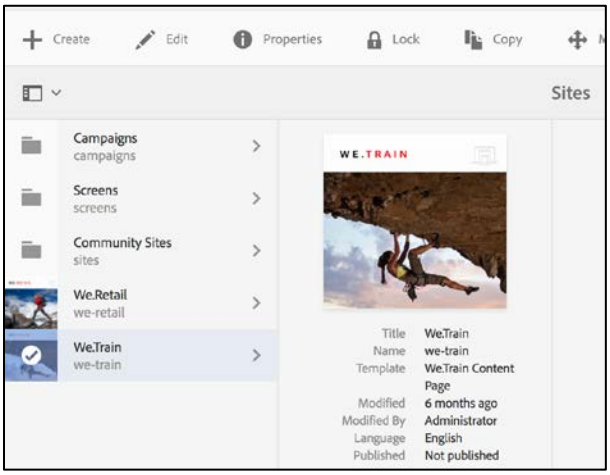


Notice how /content/we-train.html URL redirected to /content/we-train/en.

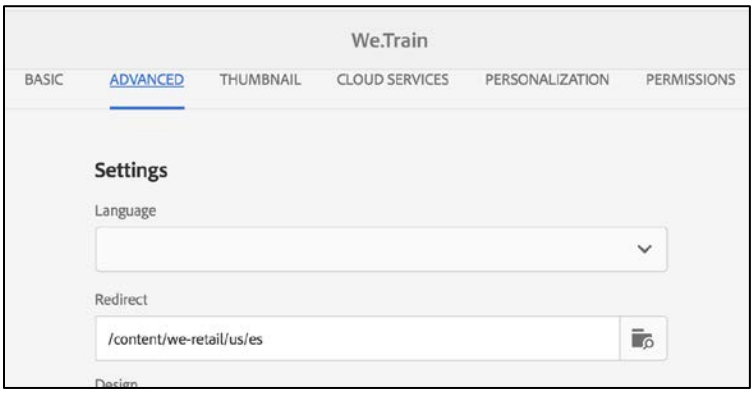
12.1.8 Task – Manage redirects from the Authoring interface

Authors can also manage page redirects using the following method.

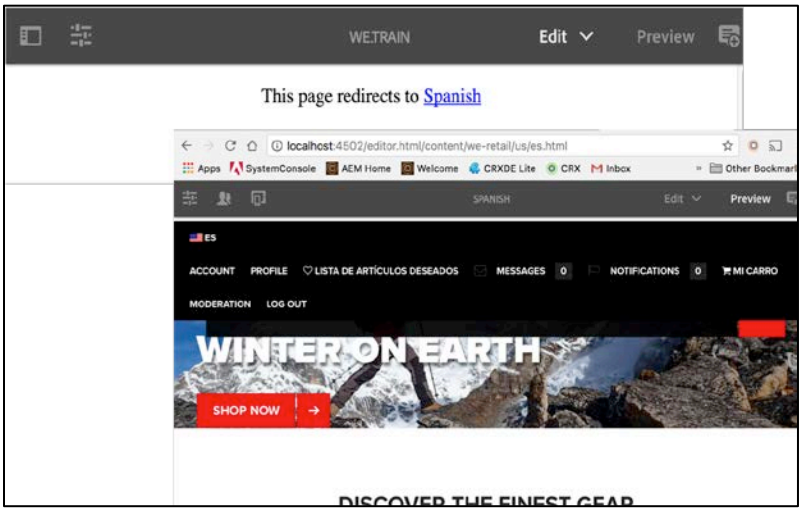
1. **Return to the Sites Console and select the We.Train page.**



2. **Click the Properties icon in the action bar and click the Advanced tab.**
3. **Modify the Redirect property to contain the following value: /content/we-retail/us/es.**



4. **Click Save & Close.**
5. **Using the Site Console, navigate to Sites > We.Train and open the page in both Edit and Preview modes.**



13 INTERNATIONALIZATION (GLOBALIZATION/LOCALIZATION)

Internationalizing the Authoring Interface

The Adobe Experience Manager Authoring interface ships in *seven languages*, allowing authors to enter and manage content in these seven languages. When you create your own components and dialog boxes, as we learned earlier, you can provide those dialog boxes in multiple languages as well.

1. DE (German) (Deustch)
2. ES (Spanish) (Español)
3. FR (French) (Français)
4. IT (Italian) (Italiano)
5. PT-BR (Portuguese (Brazil)) (Português (Brasil))
6. ZH-CN (Chinese (China))
7. ZN-TW (Chinese (Taiwan))

Internationalization of the authoring interface allows you to provide dynamic messaging based on the authors' language preference. Internationalization message bundles are stored in the repository under nodes (node type sling:Folder) named **i18n**. The children nodes (node type sling:Folder + mixin mix:language) of the i18n node represent languages and are named using the ISO code of the languages that are supported, for example, en, de, and fr. Each code is a two or four character representation of a language, such as "es" for Español (Spanish) or "pt-br" for Portuguese (Brazil). Below these language nodes are the message bundle nodes (node type sling:MessageEntry), which will contain a simple key message pair.

The location of the i18n node within the repository determines the scope of the message bundles. If located in a project directory, for example, /apps/training, it should contain only messages related to the current project. However, if located in a component hierarchy, it should contain only component-specific translations. Globally used messages should be place in /apps/i18n.

```
<h1>${“English string” @ i18n}</h1>
```

Mixin Nodes

Every node has one and only one primary node type and zero or more mixins. The primary node type, which is assigned at node creation, defines the structure of the node: for example, mandatory properties, allowed child node types, mandatory child nodes, etcetera.

Mixin node types specify additional properties or child nodes related to functionality being added to the node. For example, ***mix:created*** adds two properties:jcr:createdBy and jcr:created to the node type definition. The mixin ***mix:language*** adds the jcr:language property to the node type definition.

Exercise

13.1.1 Task – Create the content to be localized

1. **Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.**
2. **Using the code from the Exercise_Files, modify the code in footer.html**

footer.html

```
<footer class="we-Footer width-full">
    <div class="container">


        <div class="row">
            <div class="row we-Footer-section we-Footer-section--sub">
                <div class="we-Footer-section-item">
                    <div class="we-Logo we-Logo--big">
                        we.<strong>train</strong>
                    </div>
                    <div class="we-Footer-nav">
                        <h2 data-sly-resource="${'toolbar' @
resourceType='foundation/components/toolbar'}"></h2>
                    </div>
                </div>
            </div>


            <div class="row we-Footer-section we-Footer-section--sub">
                <div class="we-Footer-section-item">
                    <span class="text-uppercase text-muted">${"&copy; {0} We.Train.
All rights reserved." @ i18n, format='2017', context='html'}</span>
                </div>
                <div class="we-Footer-section-item">
                    <a href="#" class="text-uppercase text-muted">Terms of use &amp;
privacy policy</a>
                </div>
            </div>


            <div class="row">
                <div class="col-md-12">
                    <div class="text-center">
                        <a href="#top" class="btn btn-primary">Back to the top</a>
                    </div>
                </div>
            </div>
        </div>
        <sly data-sly-include="customfooterlibs.html" />
    </div>
```

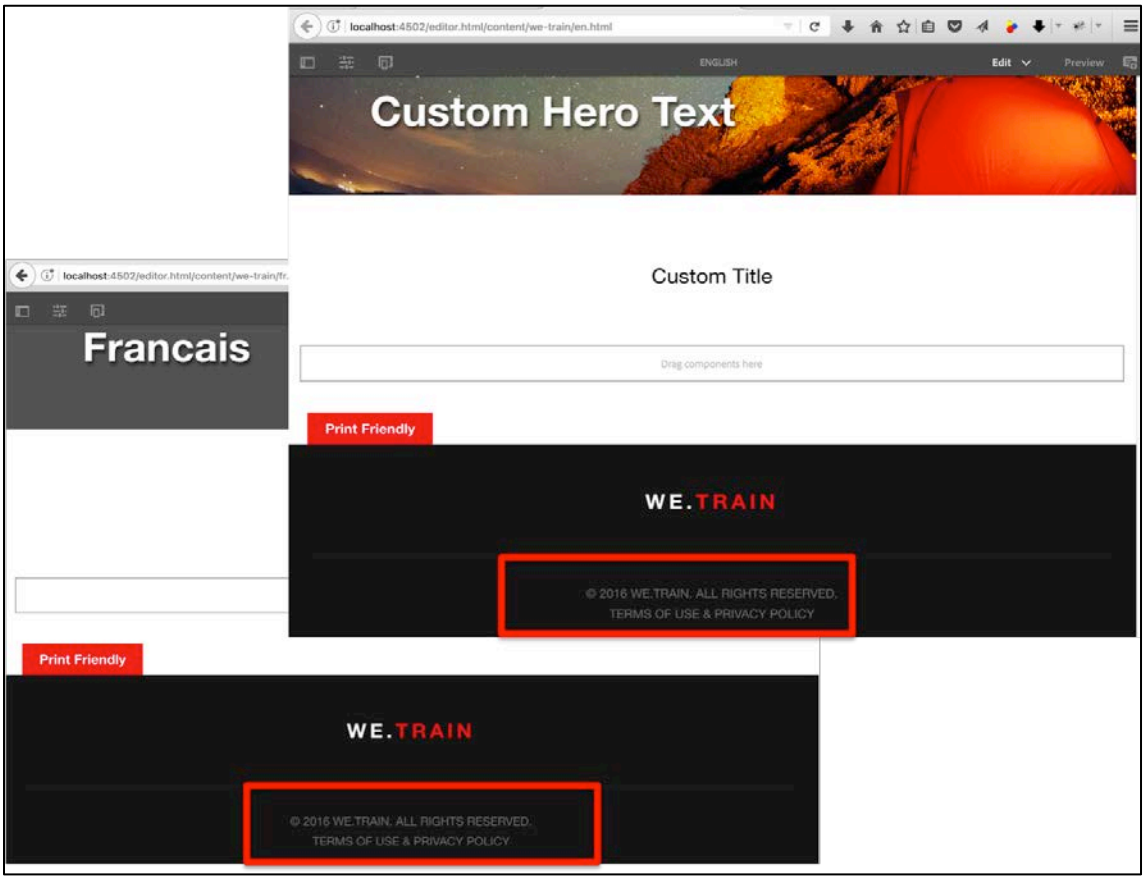
```
</footer>
```

3. **Let's look at the HTL code that specifies the internationalization process:**

```
<span class="text-uppercase text-muted">${"&copy; {0} We.Train. All rights reserved." @ i18n,
format='2017', context='html'}</span>
```

- *i18n* - specifies that the string is an i18n key
- *format='2017'* - inserts the value at "{0}". This way we can change the year as often as we need to without changing the sling:key value.
- *context = 'html'* - specifies the context of this string for security purposes

4. **Using the Sites Console, navigate to the We.Train English and French pages. Scroll to the bottom of each page and notice the updated Copyright statement with the current year:**

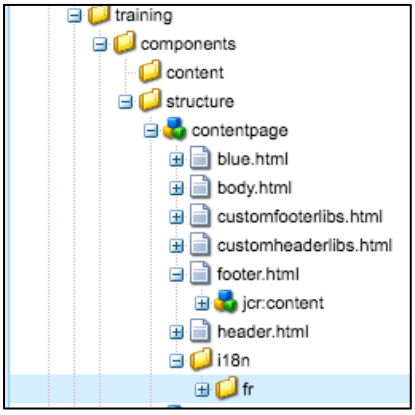


5. **Notice that both the English and French pages have the same wording in the footer. The default string is English.**

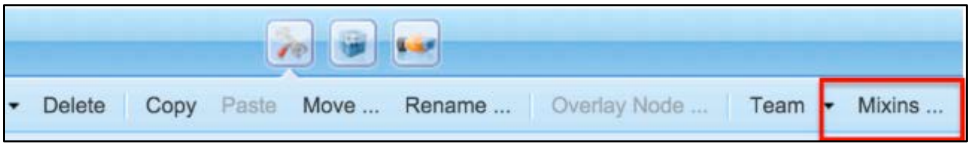
13.1.2 Task – Create the localization information

The purpose of this exercise is to internationalize the copyright statement on the footer of a website.

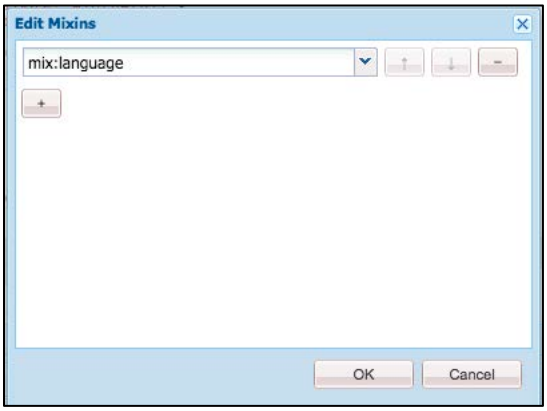
1. **Using CRXDE Lite, navigate to /apps/training/components/structure.**
2. **Right-click the contentpage node and select Create… > Create Folder.**
3. **Name the folder i18n.**
4. **Save your changes.**
5. **Create a child folder to i18n, named fr and save.**



6. **Select the fr folder and click the Mixins button displayed on the toolbar.**

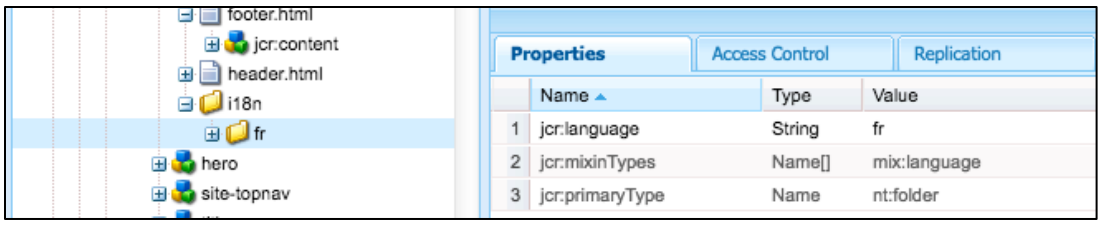


7. **Click on the "+" symbol and select mix:language and click OK.**

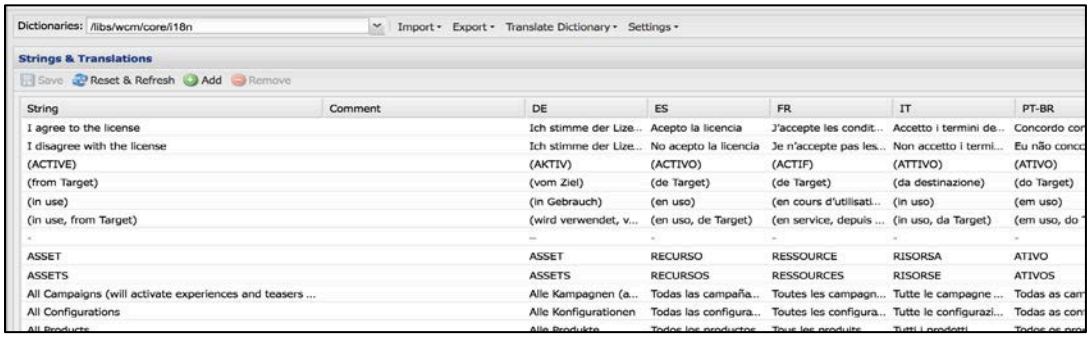


8. **Add the following property to the fr folder and save.**

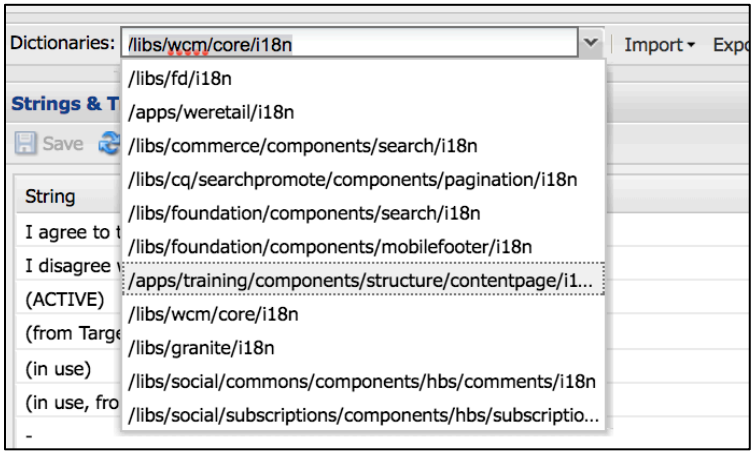
Name	Type	Value
jcr:language	String	fr



9. **Use your browser to navigate to http://localhost:4502/libs/cq/i18n/translator.html.**



10. **From the drop-down list, open the translation dictionary for the contentpage component: /apps/training/components/structure/contentpage/i18n.**

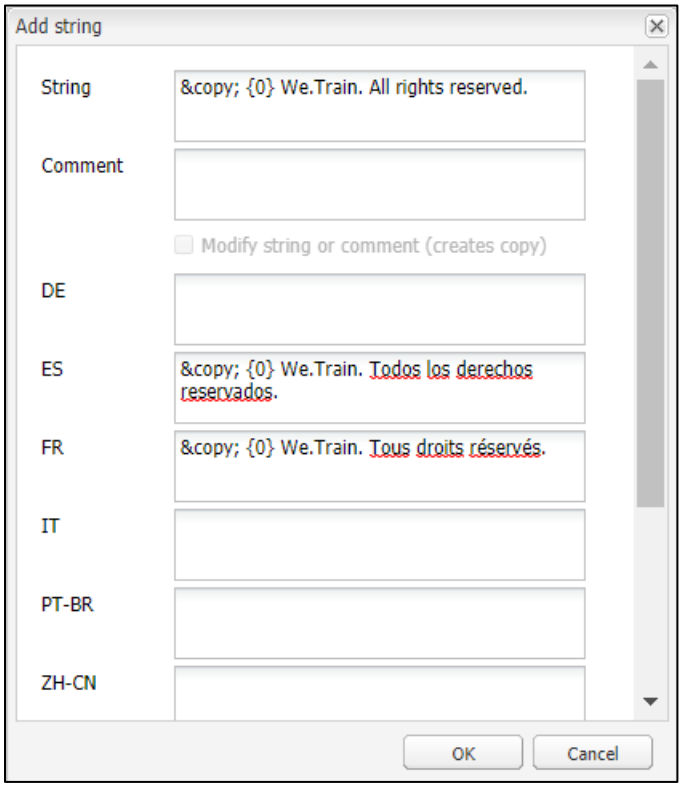


11. **Click Add.**
12. **Enter the following information in the appropriate places in the dialog box and click OK.**

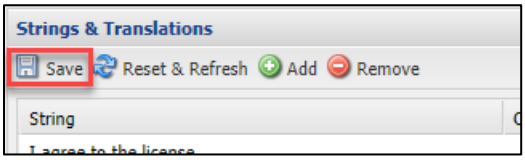
NOTE: To ensure special characters in French are used, copy and paste the French values from the Exercise_Files text file into the dialog.

Name	Type
String	© {0} We.Train. All rights reserved.
ES	© {0} We.Train. Todos los derechos reservados.
FR	© {0} We.Train. Tous droits réservés.

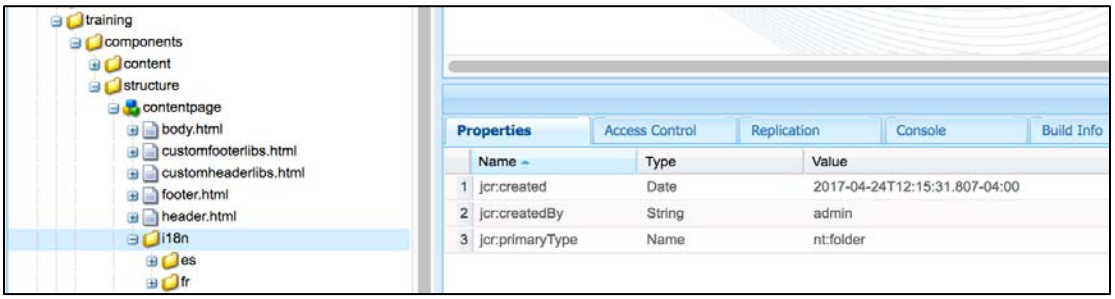
NOTE: Ensure your dialog appears like this:



13. **Save your changes at upper-left using the Save button:**



14. **Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage/i18n.**

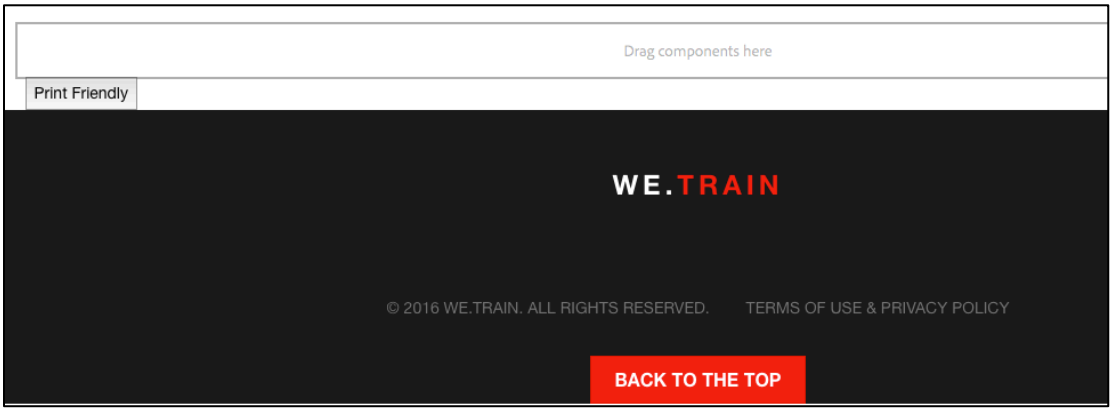


15. **Notice that a new child node of fr has appeared. Examine its properties. You may need to refresh the node in CRXDE Lite to see the new node. Right-click on the i18n node and choose Refresh if needed.**

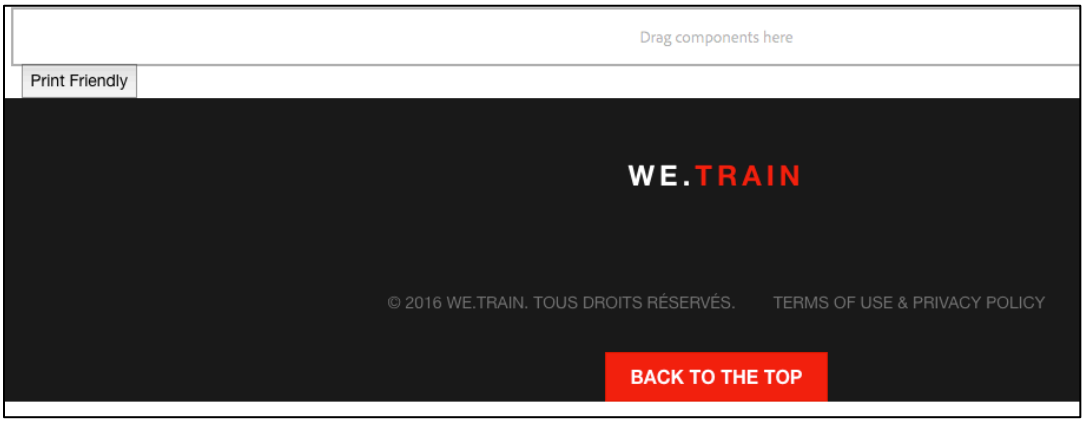
16. **Notice that an es node has also appeared. Examine the properties of the es node.**

13.1.3 Task – Test the localized content

1. **Using the Sites Console, refresh the Sites > We.Train > English page and examine the footer.**



2. **Refresh the Sites > We.Train > French page and examine the footer.**
3. **Notice that the copyright is in French.**



Extra Credit: Create a Spanish (Español) root page. HINT: Use "es" as the name. What do you see in the footer?

14 CONTENT COMPONENTS

So far in this journey, each lesson has taught us an important concept. We have had small, neat components that served a single purpose to demonstrate that concept. But in real life you will have many requirements and will need to design a component to meet them.

What if …. you had the following requirements:

- A component that will return stock information.
- A component that fits in with the website look and feel.
- Allows the author to specify:
    - which stock is of interest
    - validating the stock is a valid stock symbol
    - a summary about the stock feed
- Allows the author to choose to display:
    - The date the request was made
    - The time the request was made
    - Whether the stock price is up or down
    - The stock's opening price
    - The high and low ranges
    - The stock volume
- A component that allows the user to download stock history
- A component that can be dragged into a layout container (responsive grid)

NOTE: Due to environment constraints, our component will use canned data.

14.1.1 Task – Create the initial StockPlex component

First, let's create a simple html component with static rendering.

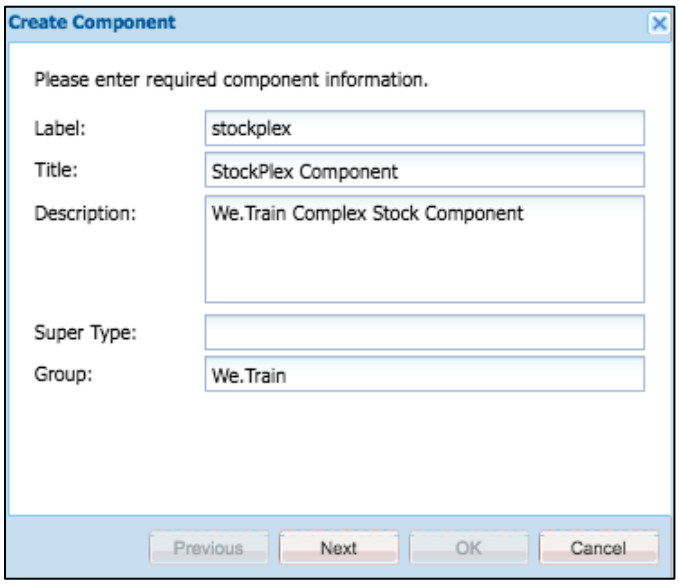
1. **Using CRXDE Lite, navigate to /apps/training/components/content.**
2. **Right-click on /apps/training/components/content and select Create... > Create Component.**
3. **Enter the following values in the dialog:**

   Label: stockplex

   Title: StockPlex Component

   Description: We.Train Complex Stock Component

   Group: We.Train



4. **Click Next, and then click OK.**
5. **Save your changes.**
6. **Rename the default script stockplex.jsp to stockplex.html.**
7. **Using the code from the Exercise_Files for this module, replace the sample code in stockplex.html.**

   stockplex.html

```
<div id="stockplex">
    <div >
        <table id="stockPrice">
            <tr>
                <h1 id="symbol" scope="row" width="48%">STOCK</h1>
                <h1 id="currentPrice">Current Value: Value</h1>
            </tr>
        </table>

        <div id="summary">
            <h3>Summary: </h3>
            <p>Value</p>
```

```
        </div>

        <table id="items"></table>
    </div>
</div>
```

We need to define a **cq:editConfig** node so the component will show up in the list of Available Components in the layout container (responsive grid) design.

8. **Right-click on the stockplex component node.**
9. **Select Create… > Create Node.**
10. **Enter the following in the dialog:**

    Name: **cq:editConfig**

    Type: **cq:EditConfig**

    Tip: The node name is a reserved word. It must be named cq:editConfig and be case-sensitive.

    We need to create a cq:dialog node to enable component delete functionality and to provide a means for the author to enter content.

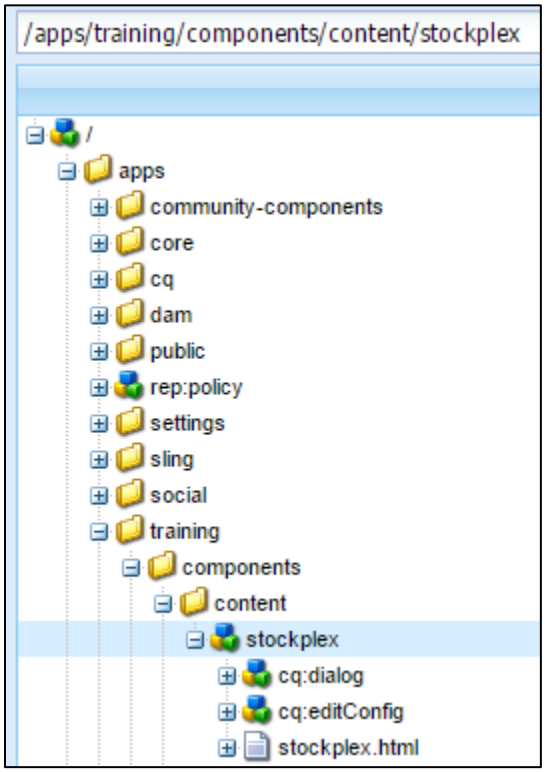
11. **Right-click on the stockplex component.**
12. **Select Create… > Create Node.**
13. **Enter the following in the dialog:**

    Name: **cq:dialog**

    Type: **nt:unstructured**

    Tip: The node name is a reserved word. It must be named cq:dialog and it is case sensitive.

Hint: Your node structure should look like this:



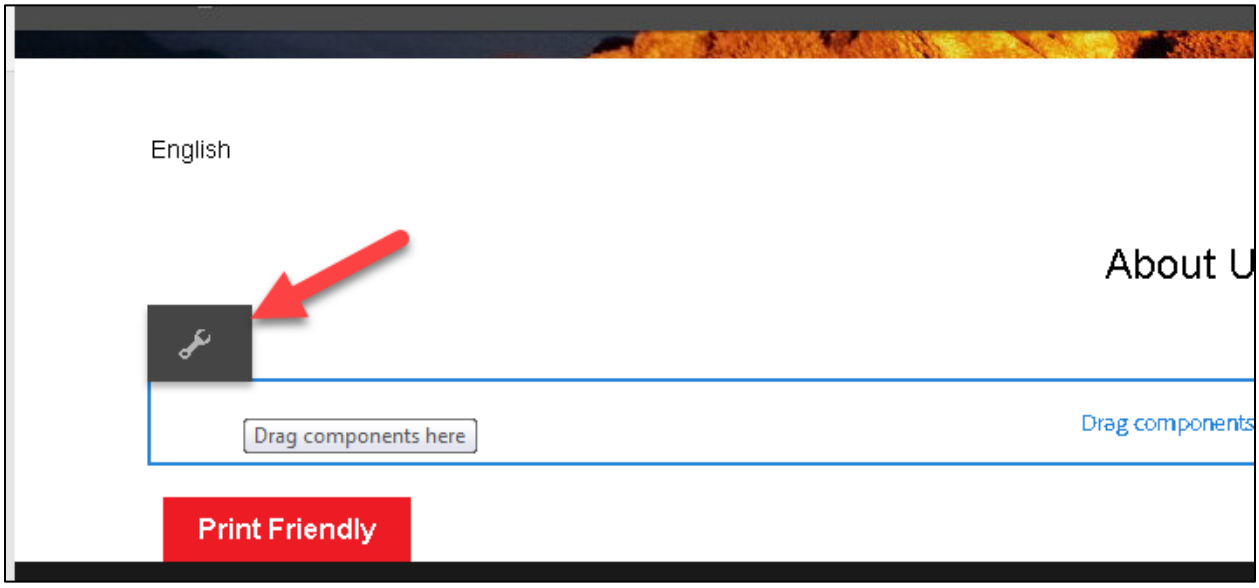
14.1.2 Task – Enable the StockPlex component

We need to make the new component available to the layout container design.

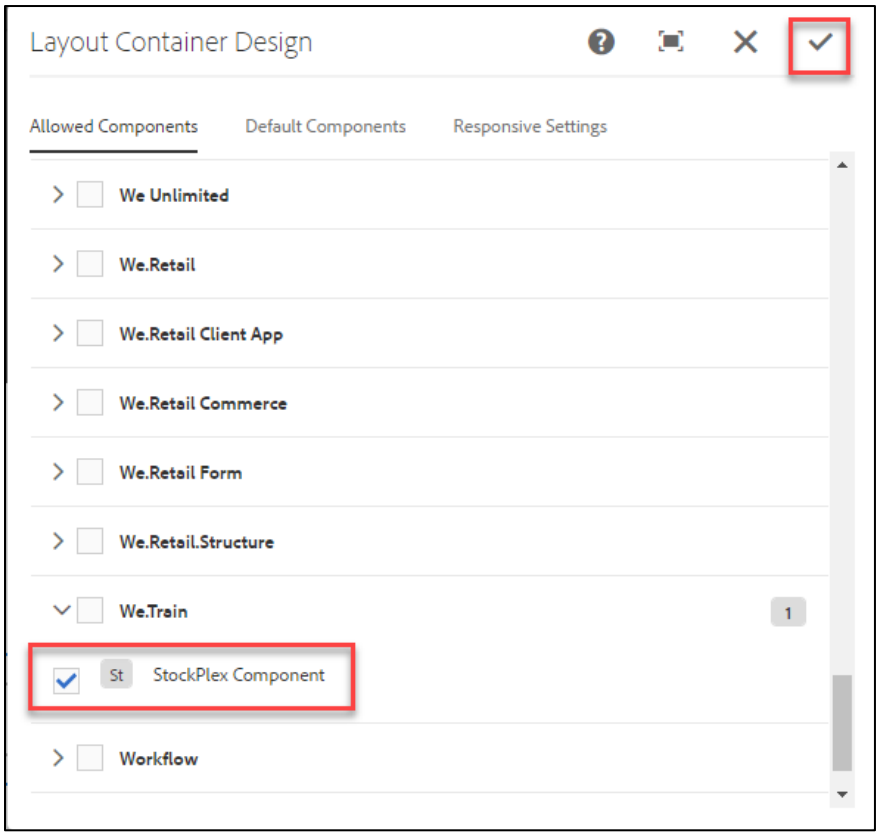
1. **Using the Sites Console, navigate to Sites > We.Train > English > About.**

NOTE: In your instance, this may be named "About" instead of "About Us".

2. **Open the page for Edit and switch to Design Mode.**
3. **Click on the layout container, and then click Configure.**

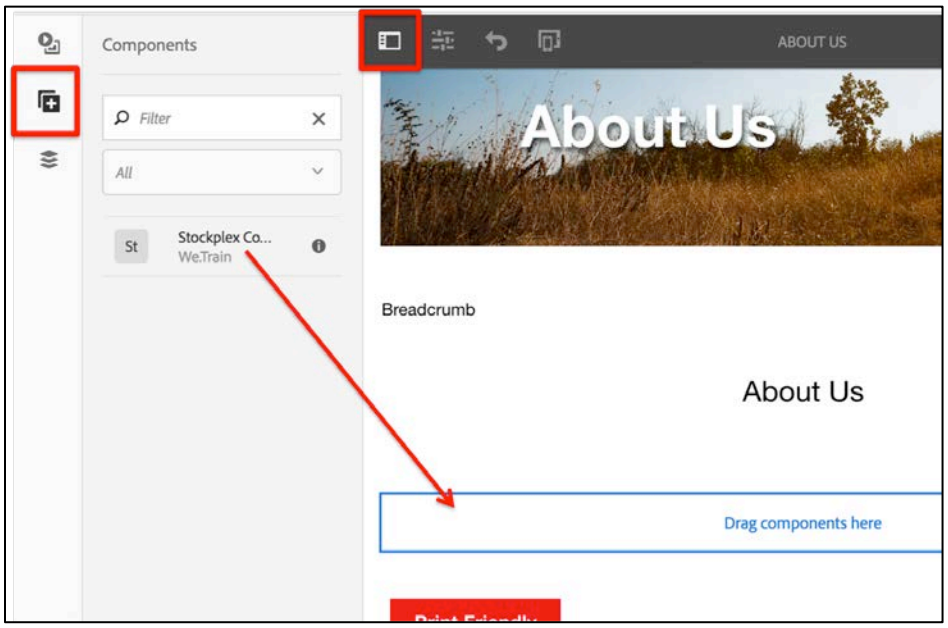


4. **In the dialog of Available Components, scroll down, select the We.Train group and select the StockPlex component. Click the checkmark to save your choices.**

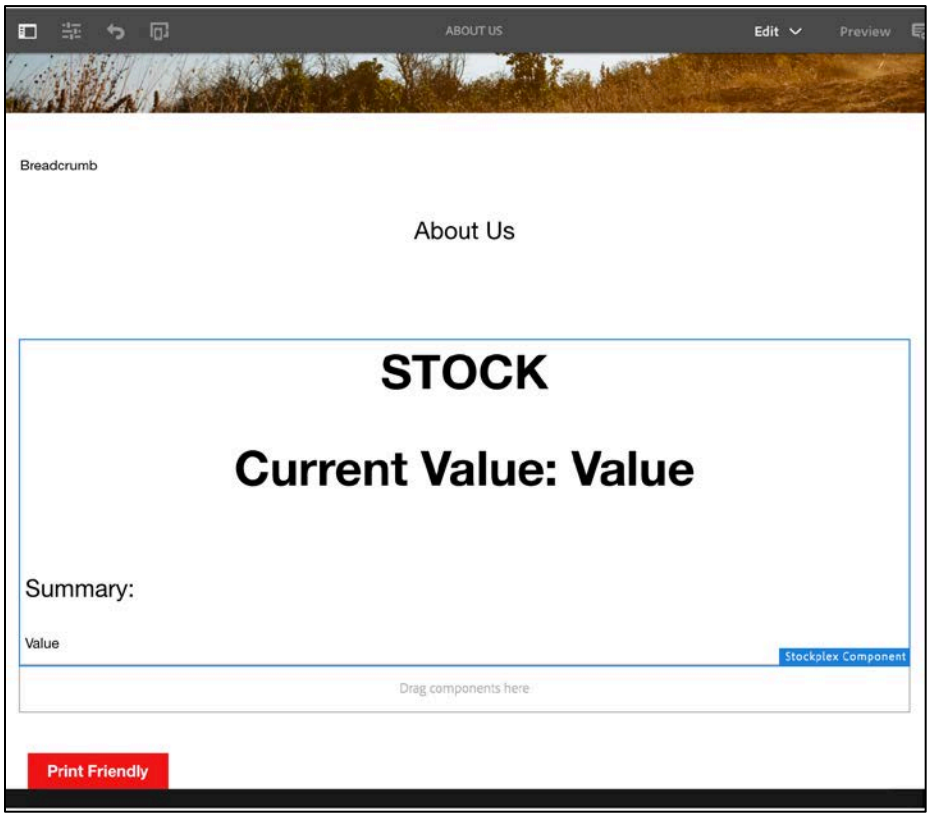


Pro Tip: You can check the We.Train group to enable all (current and future) components in the We.Train group.

5. **Now let's drag the StockPlex component into the layout container.**
6. **Switch to Edit mode.**
7. **Click on the Side Panel toggle, and then click on the Components tab. Drag the StockPlex component into the layout container (the Drag components here area).**



The result should look like this:



14.1.3 Task – Add a placeholder

Now let's add a bit more sophisticated code.

1. **Using CRXDE Lite, navigate to /apps/training/components/content and open stockplex.html.**
2. **Using the code from the Exercise_Files, replace the code in stockplex.html.**

stockplex.html

```
<div id="stockplex" data-sly-use.stock="stockplex.js" class="${stock.cssClass}"
data-emptytext="StockPlex Component" >
    <div data-sly-test="${stock.symbol}">
        <table id="stockPrice">
            <tr>
                <h1 id="symbol" scope="row" width="48%">STOCK</h1>
                <h1 id="currentPrice">Current Value: Value</h1>
            </tr>
        </table>

        <div id="summary">
            <h3>Summary: </h3>
            <p>Value</p>
        </div>

        <table id="items"></table>
    </div>
</div>
```

NOTE: The html code is rendering the stock symbol and current stock price.

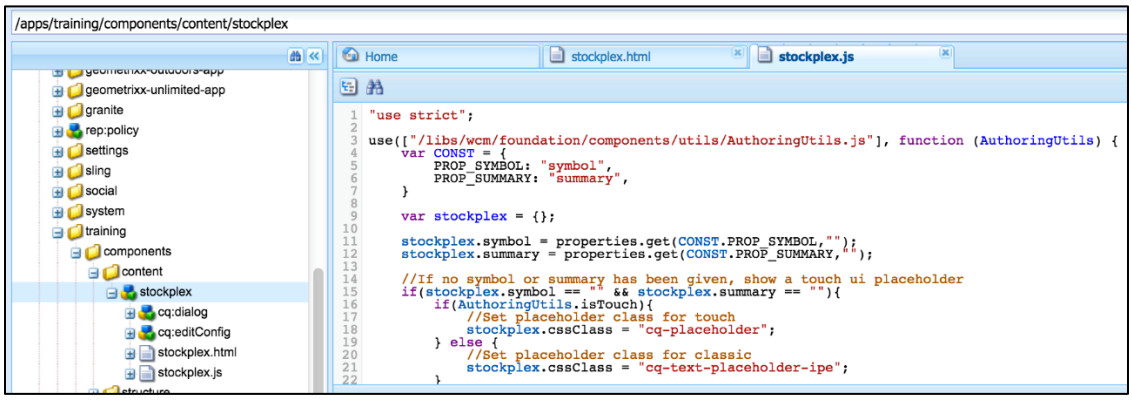
3. **Right-click on the stockplex component and select Create… > Create File.**
4. **Name the file stockplex.js and use the code from the Exercise_Files.**

stockplex.js

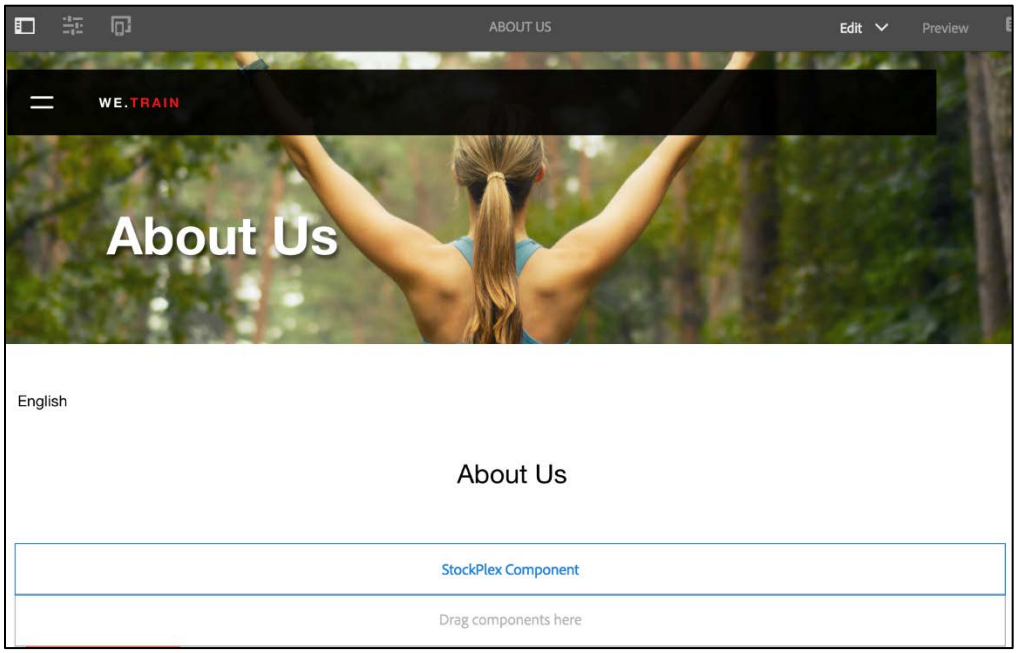
```
"use strict";


use(["/libs/wcm/foundation/components/utils/AuthoringUtils.js"], function
(AuthoringUtils) {
       var CONST = {
        PROP_SYMBOL: "symbol",
        PROP_SUMMARY: "summary",
    }


    var stockplex = {};
       stockplex.symbol = properties.get(CONST.PROP_SYMBOL,"");
    stockplex.summary = properties.get(CONST.PROP_SUMMARY,"");
```

```
    //If no symbol or summary has been given, show a touch ui placeholder
       if(stockplex.symbol == "" && stockplex.summary == ""){
              if(AuthoringUtils.isTouch){
                     //Set placeholder class for touch
                     stockplex.cssClass = "cq-placeholder";
              } else {
                     //Set placeholder class for classic
                     stockplex.cssClass = "cq-text-placeholder-ipe";
              }
       }
    return stockplex;
});
```

NOTE: The code is getting the value of the stock symbol and summary from the properties written by the dialog box, but still just displays static text, as we have not yet built out the dialog box for author input.



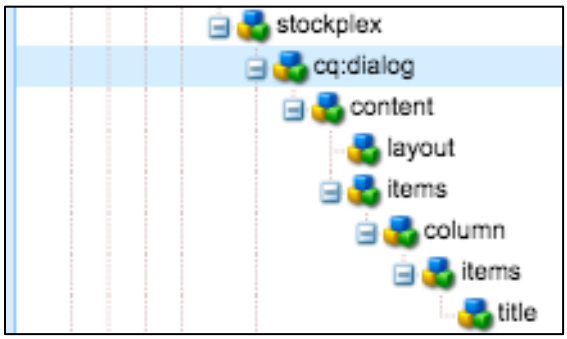
5. Reload the About Us page.



14.1.4 Task – Build out the dialog box

Now let's build out the dialog box to accept and store input from the author. To make things easier, we will start by copying a dialog from a Foundation component and then modifying it to have the elements that we need.

1. **Navigate to apps/training/components/content/stockplex.**
2. **Delete the cq:dialog node.**
3. **Save your changes.**
4. **Navigate to /apps/training/components/structure/title.**
5. **Right-click on the cq:dialog node and select Copy.**
6. **Navigate to /apps/training/components/content/stockplex.**
7. **Right-click on stockplex, and paste the copied node.**
8. **Save your changes.**
9. **Modify the jcr:title on the cq:dialog node to be We.Train StockPlex.**
10. **Save your changes.**
11. **Expand the nodes of the dialog that you just pasted.**



We will now modify what was a simple title dialog to be a much more complex tabbed dialog box.

12. **Delete the title node at the bottom of the dialog structure and Save.**
13. **Modify the sling:resourceType property of the cq:dialog/content/layout node so that the layout is a tab.**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/layouts/**tabs**

14. **To modify cq:dialog/content/items/column node to create a section, add a jcr:title property to the cq:dialog/content/items/column node and modify the sling:resourceType property to be a section.**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/section
jcr:title	String	Stock

15. **Right-click the items node at the bottom of the dialog structure (cq:dialog/content/items/column/items) and select Create… > Create Node.**
16. **Create the following two nodes as children to the items node at the bottom of the dialog structure. All nodes are of type nt:unstructured.**

Pro Tip: You can create one node and then copy and paste the others. Just change the node name and property values.

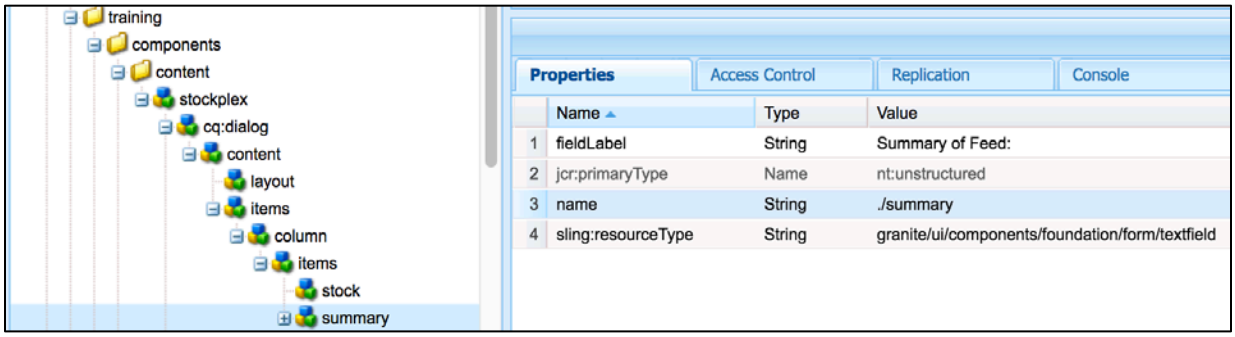
NOTE: Do not forget to click **Save All** after adding each property!

Node Name: **stock**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/textfield
fieldLabel	String	Stock Symbol:
name	String	./symbol

Node Name: summary

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/textfield
fieldLabel	String	Summary of Feed:
name	String	./summary



17. **Replace the code in stockplex.html using the code from the Exercise_Files.**

stockplex.html

```
<div id="stockplex" data-sly-use.stock="stockplex.js" class="${stock.cssClass}"
data-emptytext="StockPlex Component" >
    <div data-sly-test="${stock.symbol}">
        <table id="stockPrice">
            <tr>
                <h1 id="symbol" scope="row" width="48%">${stock.symbol}</h1>
                <h1 id="currentPrice">Current Value: ${stock.currentPrice}</h1>
            </tr>
        </table>

        <div id="summary">
            <h3>Summary: </h3>
            <p>${stock.summary}</p>
        </div>

        <table id="items"></table>
    </div>
</div>
```

18. **Replace the code in stockplex.js using the code from the Exercise_Files.**

stockplex.js

```
"use strict";


use(["/libs/wcm/foundation/components/utils/AuthoringUtils.js"], function
(AuthoringUtils) {
```

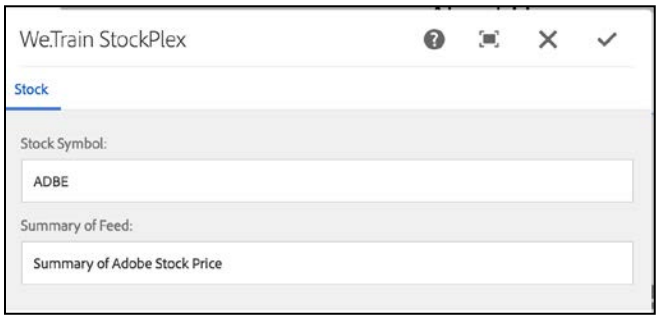
```
            var CONST = {
                PROP_SYMBOL: "symbol",
                PROP_SUMMARY: "summary",
            }

            var SHOW_PROP = {
                REQUESTDATE: "showRequestDate",
                UPDOWN: "showUpDown",
                OPENPRICE: "showOpenPrice",
                RANGEHIGH: "showRangeHigh",
                RANGELOW: "showRangeLow",
                VOLUME: "showVolume",
            }

            var INFO_PROP = {
                CURRENTPRICE: "currentPrice",
                REQUESTDATE: "requestDate",
                UPDOWN: "upDown",
                OPENPRICE: "openPrice",
                RANGEHIGH: "rangeHigh",
                RANGELOW: "rangeLow",
                VOLUME: "volume"
            }

            var NAME = {
                CURRENTPRICE: "Current Price",
                REQUESTDATE: "Request Date",
                UPDOWN: "Up/Down",
                OPENPRICE: "Open Price",
                RANGEHIGH: "Range High",
                RANGELOW: "Range Low",
                VOLUME: "Volume"
            }

            var stockplex = {};
            stockplex.items = []; //each item holds the [Name, Value] to be
        displayed on the page

           stockplex.symbol = properties.get(CONST.PROP_SYMBOL,"");
            stockplex.summary = properties.get(CONST.PROP_SUMMARY,"");

            //If no symbol or summary has been given, show a touch ui
        placeholder
```

```
    if(stockplex.symbol == "" && stockplex.summary == ""){
        if(AuthoringUtils.isTouch){
            //Set placeholder class for touch
            stockplex.cssClass = "cq-placeholder";
        } else {
            //Set placeholder class for classic
            stockplex.cssClass = "cq-text-placeholder-ipe";
        }
    }


    //Get the current stock
    stockplex.currentPrice = properties.get(INFO_PROP.CURRENTPRICE,
'100');


    //Add the appropriate properties to the item array to be
displayed
    if(properties.get(SHOW_PROP.REQUESTDATE)){
      stockplex.items.push([NAME.REQUESTDATE,
properties.get(INFO_PROP.REQUESTDATE, '01/01/2016')]);
    }
    if(properties.get(SHOW_PROP.UPDOWN)){
      stockplex.items.push([NAME.UPDOWN,
properties.get(INFO_PROP.UPDOWN, 10)]);
    }
    if(properties.get(SHOW_PROP.OPENPRICE)){
      stockplex.items.push([NAME.OPENPRICE,
properties.get(INFO_PROP.OPENPRICE, 90)]);
    }
    if(properties.get(SHOW_PROP.RANGEHIGH)){
      stockplex.items.push([NAME.RANGEHIGH,
properties.get(INFO_PROP.RANGEHIGH, 105)]);
    }
    if(properties.get(SHOW_PROP.RANGELOW)){
      stockplex.items.push([NAME.RANGELOW,
properties.get(INFO_PROP.RANGELOW, 90)]);
    }
    if(properties.get(SHOW_PROP.VOLUME)){
      stockplex.items.push([NAME.VOLUME,
properties.get(INFO_PROP.VOLUME, 2712018)]);
    }


    //Get the download button boolean from the Designer
    stockplex.downloadButton = currentStyle.get("downloadButton",
false);


    return stockplex;
```

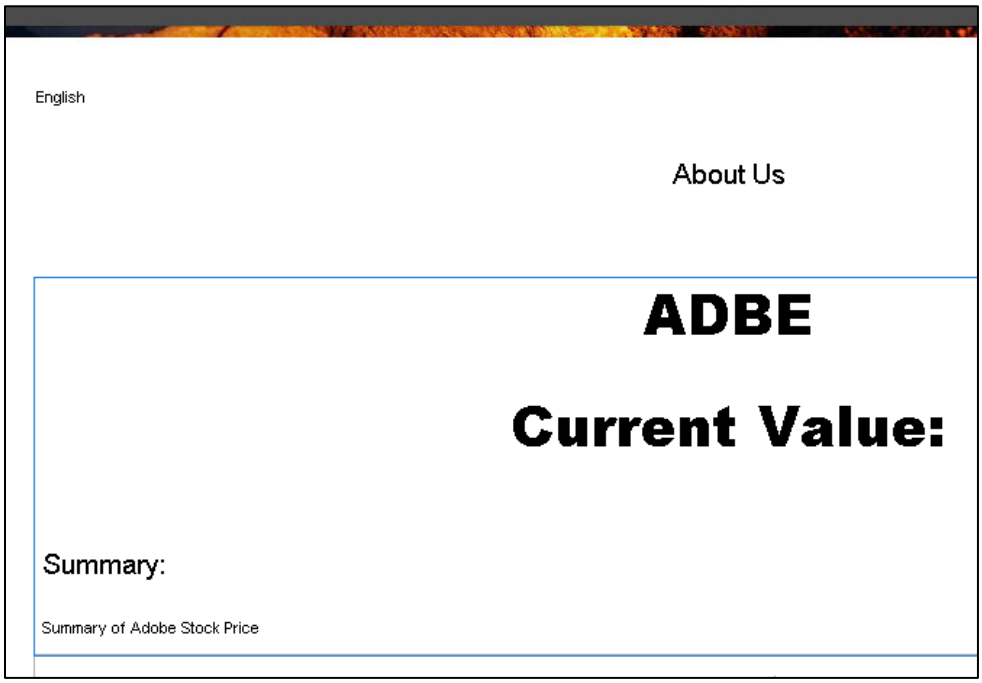
```
        });
```

19. **Using the Sites Console, reload the About Us page and ensure you are in Edit mode.**
20. **Click on the StockPlex component and open it to configure it:**

21. **Enter the Adobe stock symbol and a Feed Summary (Stock Symbol: ADBE, Summary of Adobe Stock Price:):**



22. **Click the checkmark to save.**



Notice that the Stock Symbol and the Summary text appear, but they are not very nicely formatted.

Using Client Libraries

In today's modern web development, comprehensive JavaScript libraries, with HTML and CSS, are responsible for some exciting web experiences. Managing these client-side assets can become quite cumbersome, especially because Adobe Experience Manager allows authors to select components and templates at their own convenience. Developers cannot really plan when and where client-side assets will be used. Another challenge is that many components and templates require client-side assets.

Client or HTML Libraries

Adobe Experience Manager has introduced a very interesting way to define and manage client libraries, also known as 'clientlibs'. Client libraries are defined as 'folders' (nodes of node-type cq:ClientLibraryFolder) that contain the client-side assets, CSS and JS files, and required resources, for example, images or fonts. There can be unlimited number of client library folders. The folder content can be loaded individually and at any given time.

Adobe Experience Manager has a tool called Dumplibs that lists all the defined client libraries and its properties. You can access it from the following link: http://localhost:4502/libs/granite/ui/content/dumplibs.html

Client Library Conventions

Create client libraries under /etc/clientlibs or within the component folder. A client library 'folder' is created as a node with the Node type cq:ClientLibraryFolder, with the following properties:

categories: An array of names to identify the client library

NOTE: this is a mandatory property, as it uniquely identifies this library

dependencies: An array of categories of client libraries this library depends on

NOTE: This property is used only if this library has dependencies

Dependent libraries are loaded by the HTML Library Manager, if they have not already been loaded. Each dependent library will result in an additonal include statement and access will be checked for each included library.

embed: An array of client libraries to be embedded in this library

NOTE: This property is used only if you wish to embed libraries in this library

The libraries of categories listed in 'embed' will be included in the HTML page as well. If the libraries have already been included, they are ignored. Embedded libraries do not result in an additional include statement, as they are embedded in the 'outer' library. As a result, access is checked only on the 'outer' library.

The client library files themselves are stored in subordinate folders to the client library folder. For example a folder named js or scripts to hold javascript files and/or a folder named css or styles to hold css files.

Including Client Libraries

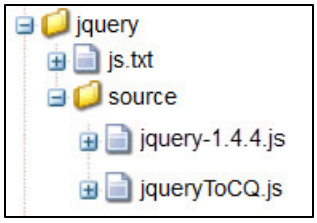
Before the client libraries can be included in your code, the 'clientlib' object must be declared. Typically, you will find the declaration in the head.html. The following is an example declaration from the foundation page component /libs/wcm/foundation/components/page/head.html:

```
!--/* declare a template */
< data-sly-use.clientLib=/libs/granite/sightly/template/clientlib.html />
```

Client libraries are loaded with the data-sly-call="${clientlib …} statement. This statement includes a client library, which can be a JS or CSS,library. For multiple inclusions of different types, for example, JS and CSS included at different locations in the script, this statement needs to be used multiple times in the HTL script.

Examples of Client Libraries

An example to define jQuery:



NOTE: The example used here describes a client library that contains only javascript files. You can define client libraries that include javascript, css and other relevant files.

Properties of the jQuery client library folder:

jcr:primaryType = cq:ClientLibraryFolder

categories = cq.jquery (dot-notated names are allowed)

The js.txt file contains:

```
#base=source
jquery-1.4.4.js
jqueryToCQ.js
```

The #base statement defines the relative path where the JS files can be found. After the base statement, the files are listed in the desired order and will be concatenated in that order.

Once defined, the client library can be used as an independent library. It can also be defined as a 'dependent' library to or defined as 'embedded' in another client library

Include Client Libraries

In HTL, use data-sly-call tag to include client libraries. Examples of statements that include client libraries in HTL:

```
< data-sly-call="${clientlib.all @ categories='yourClientLib'}" />

< data-sly-call="${clientlib.js @ categories='clientlib1,clientlib2'}" />

< data-sly-call="${clientlib.css @ categories='clientlib1,clientlib2'}" />
```

The commonly used options are:

clientlib.all: A list of comma-separated client library categories. This will include all JavaScript and CSS libraries for the given categories. The theme name is extracted from the request.

clientlib.js: A list of comma-separated client library categories. This will include all JavaScript libraries from the listed categories.

clientlib.css: A list of comma-separated client library categories. This will include all CSS libraries from the listed categories.

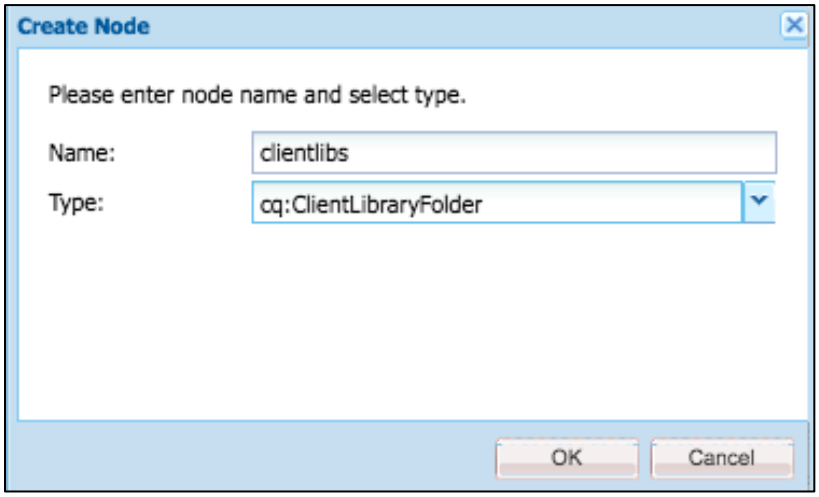
Using these options, you can include the entire client library at once or include selected parts of the library. For example, you can include the CSS at the top of the page and the JS at the bottom of the page.

14.1.5 Task – Add the client libraries

1. **Using CRXDE Lite, navigate to /apps/training/components/content.**
2. **Right-click on the stockplex node and select Create… > Create Node.**
3. **Enter the following values and save.**

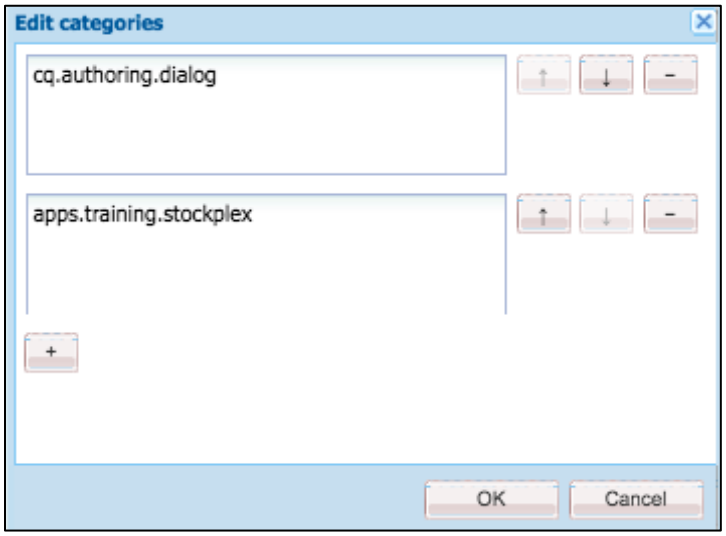
   Name: clientlibs

   Type: cq:ClientLibraryFolder



4. **Add the following multi-valued property on the clientlibs folder node. Remember: use the Multi button to create a multi-valued (array) property.**

Name	Type	Value
categories	String[]	cq.authoring.dialog apps.training.stockplex



5. **Click OK, and then Save.**
6. **Right-click again on the clientlibs node and select Create… > Create Folder.**
7. **Name the folder css.**
8. **Click OK and then Save. This is the location where we will put the css for this component, which is a component-specific client library.**

9. **Right-click on the css folder and select Create… > Create File.**
10. **Enter the name of the file as table.css.**
11. **Using the code from the Exercise_Files, paste the code into table.css. NOTE: The file is located in \clientlibs\css.**
12. **Save your changes.**

Next we are going to add some javascript for validation of the Stock symbol. This will ensure that the author enters in a 4-letter stock symbol into the dialog box. For training purposes, we will only validate for four-letter stock symbols.

13. **Right-click on the clientlibs node and select Create... > Create Folder.**
14. **Name the folder js.**
15. **Click OK and then Save. This is the location where we will put the js for this component.**
16. **Right-click on the js folder and select Create… > Create File.**
17. **Name the file as validation.js.**
18. **Using the code from the Exercise_Files (located in the clientlibs folder), paste the code into validation.js.**

validation.js

```
(function (document, $, ns) {
    "use strict";

    $(document).on("click", ".cq-dialog-submit", function (e) {
        e.stopPropagation();
        e.preventDefault();

        var $form = $(this).closest("form.foundation-form"),
            symbolid = $form.find("[name='./symbol']").val(),
               message, clazz = "coral-Button ",
         patterns = {
             symboladd: /^([a-z][a-z][a-z][a-z])\.?$/i
        };

        if(symbolid != "" && !patterns.symboladd.test(symbolid) &&
(symbolid != null)) {
                ns.ui.helpers.prompt({
                title: Granite.I18n.get("Invalid Input"),
                message: "Please Enter a valid 4 Letter Stock Symbol",
                actions: [{
                    id: "CANCEL",
                    text: "CANCEL",
                    className: "coral-Button"
                }],
            callback: function (actionId) {
                if (actionId === "CANCEL") {
                }
            }
        });
```

```
        }else{
                $form.submit();
        }
    });
})(document, Granite.$, Granite.author);
```

19. **Save your changes.**
20. **Create two additional files under the clientlibs node: css.txt and js.txt.**
21. **Using the code from the Exercise_Files, paste the code into the appropriate file (or enter the code below).**

css.txt

```
#base=css
table.css
```

js.txt

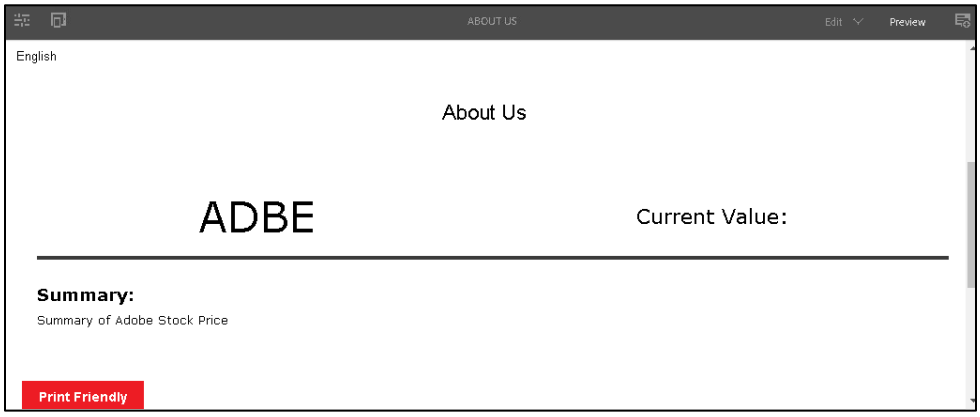
```
#base=js
validation.js
```

22. **Save your changes.**
23. **Modify stockplex.html to take advantage of the new client library using code from the Exercise_Files.**

stockplex.html

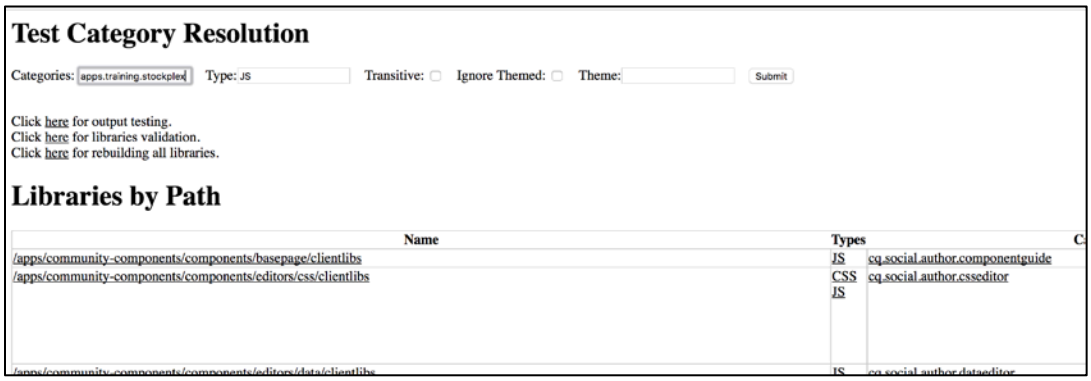
```
<sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-
sly-call="${clientlib.css @ categories='apps.training.stockplex'}" />


<div id="stockplex" data-sly-use.stock="stockplex.js" class="${stock.cssClass}"
data-emptytext="StockPlex Component" >
    <div data-sly-test="${stock.symbol}">
        <table id="stockPrice">
            <tr>
                <h1 id="symbol" scope="row" width="48%">${stock.symbol}</h1>
                <h1 id="currentPrice">Current Value: ${stock.currentPrice}</h1>
            </tr>
        </table>

        <div id="summary">
            <h3>Summary: </h3>
            <p>${stock.summary}</p>
        </div>

        <table id="items"></table>
    </div>
</div>
```

24. **Using the Sites Console, reload About Us and notice the formatted output.**

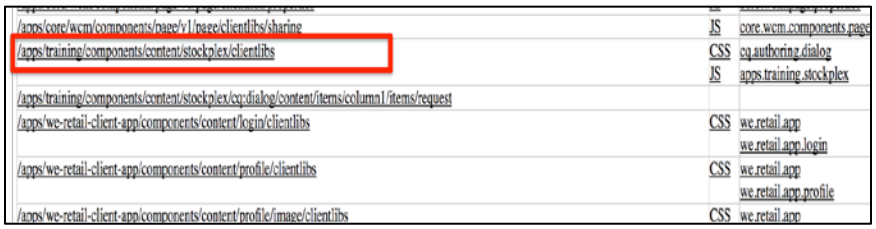


14.1.6 Task – Determine whether your client library is defined in AEM

1. Navigate to http://localhost:4502/libs/granite/ui/content/dumplibs.html.



2. Using your browser's keyword search, search for "stockplex":



Notice that when you locate it, you may click on it to directly navigate to that node folder in CRXDE Lite.

14.1.7 Task – Determine which client libraries are loaded with the page

The browser developer tools will assist you in determining whether your client library was loaded with the page.

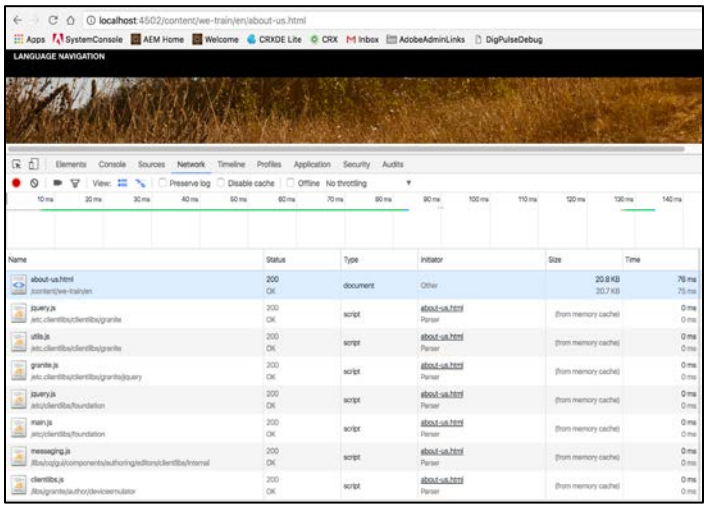
1. Navigate to http://localhost:4502/content/we-train/en/about-us.html. Notice that "editor.html" was removed from the URL. NOTE: You may have set this up as "About" instead of "about-us". In that case, navigate to: http://localhost:4502/editor.html/content/we-train/en/About.html. Remember, AEM is case-sensitive.

2. Depending on which browser you have, the method for finding the Developer tools is different.

CHROME:

1. Click on the three stacked dots in the upper right of the browser and then select **More Tools** > **Developer Tools** from the Menu.

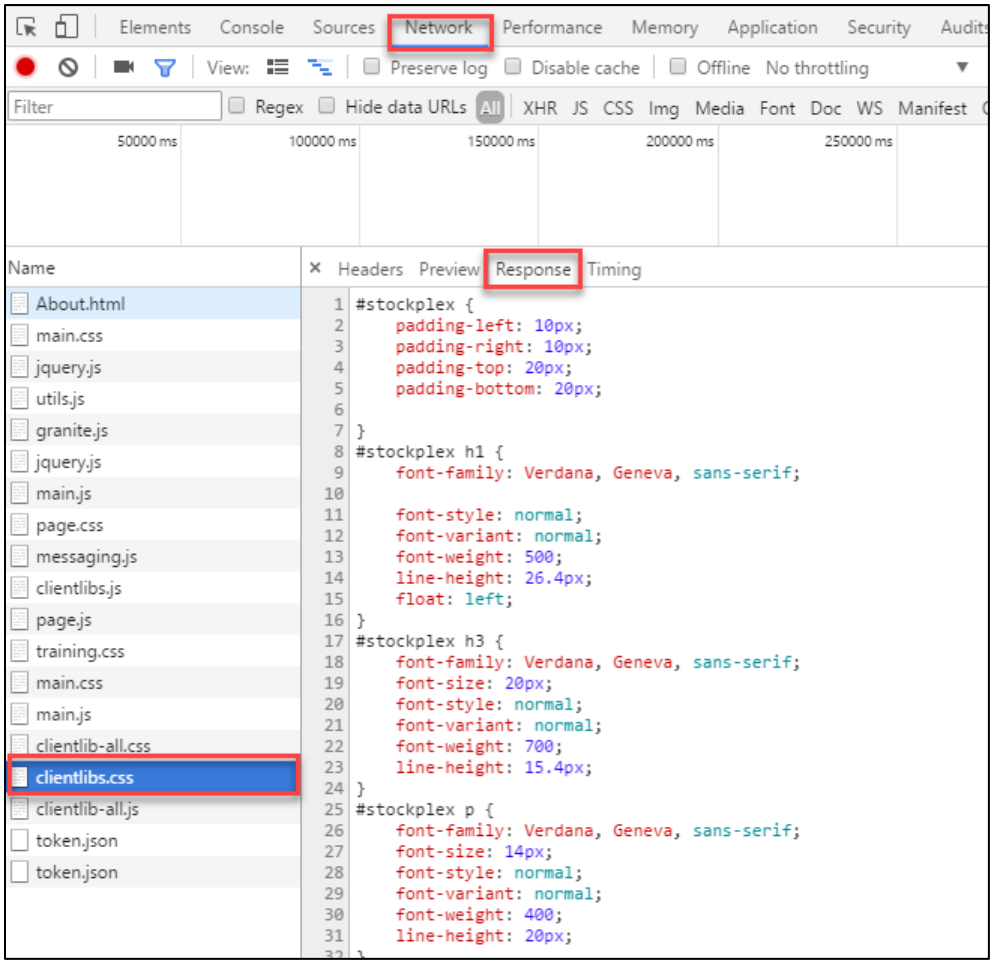
2. Click the **Network** tab.



NOTE: In the Chrome developer tools, you may have to click on the funnel icon to select All.

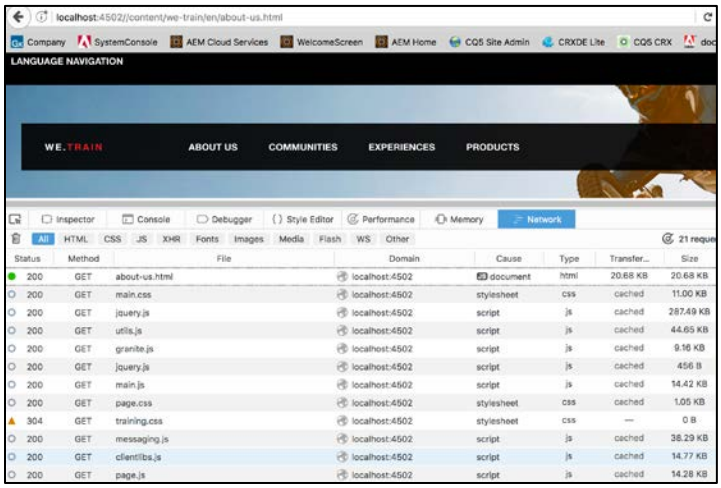
3. Reload the About Us page.

4. Find the clientlibs.css entry for stockplex and click on it. If you click on the Response tab, you will see the css file contents.



FIREFOX:

1. Click on the three stacked lines in the upper right of the browser and select Developer > Network.



2. Make sure that "All" is selected. If you do not see the JavaScript and css files listed, reload the page.

14.1.8 Task – Extend the Dialog box to provide the author with more options

1. Using CRXDE Lite, navigate to /apps/training/components/content/stockplex/cq:dialog/content/items/column.
2. Right-click the column node, and rename it to **column1.**
3. Right-click the **items** child node of the newly renamed **column1** node (cq:dialog/content/items/column1/items) and select Create... > Create Node.
4. Create the following two nodes as children to the items node. All nodes are of type nt:unstructured. The tables below define the new properties you need to add to the nodes once created.
   Pro Tip: You can create one node and then copy and paste the others. Just change the node name and property values.

   Node Name: **requestDate**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	Request Date
name	String	./showRequestDate
value	Boolean	true

5. Navigate up the node structure to **/apps/training/components/content/stockplex/cq:dialog/content/items**.
6. Right-click on the **items** node that is the parent of the **column1** node (**cq:dialog/content/items**) and select **Create…** > **Create Node**.

   Name: column2

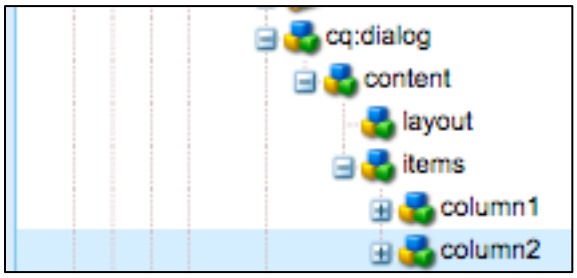
   Type: nt:unstructured

   Add the following properties on the **column2** node.

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/section
jcr:title	String	Other

7. Right-click on the **column2** node and select **Create…** > **Create Node**.
   Name: items

   Type: nt:unstructured



8. Right-click the items node and select **Create...** > **Create Node**. Create the following five nodes as children to the items node. All nodes are of type **nt:unstructured**. The table below defines the new properties you need to add to the nodes once created.

   Pro Tip: You can create one node and then copy and paste the others. Just change the node name and property values.

   Node Name: **upDown**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	Up/Down
name	String	./showUpDown
value	Boolean	true

NOTE: The remaining nodes are not needed to complete this task, but are nice to see full functionality of the StockPlex component. If there is time, feel free to add the remaining checkbox form fields.

Node Name: **openPrice**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	**Open Price**
name	String	**./showOpenPrice**
value	Boolean	true

Node Name: **rangeHigh**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	**Range High**
name	String	**./showRangeHigh**
value	Boolean	true

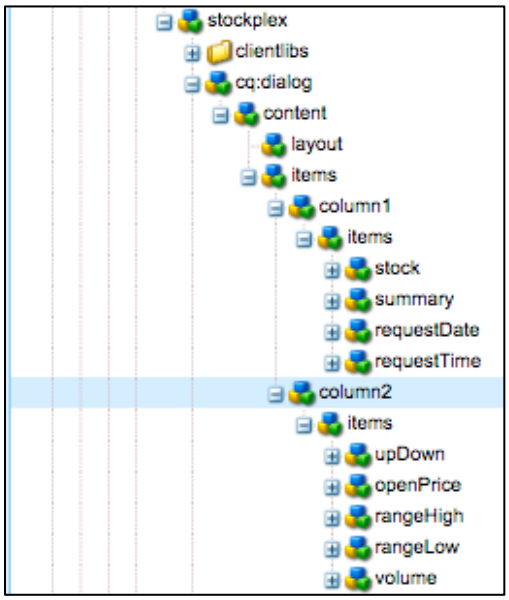
Node Name: **rangeLow**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	**Range Low**
name	String	**./showRangeLow**
value	Boolean	true

Node Name: **volume**

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	**Volume**
name	String	**./showVolume**
value	Boolean	true

Your node structure should look like this:



NOTE: If you do not have the node hierarchy/structure correct, you may drag and drop nodes to their correct location(s).

9. To take advantage of the new author input, replace the code for **stockplex.js** using the code from the Exercise_Files.

stockplex.js

```
"use strict";

use(["/libs/wcm/foundation/components/utils/AuthoringUtils.js"], function (AuthoringUtils)
{
       var CONST = {
         PROP_SYMBOL: "symbol",
         PROP_SUMMARY: "summary",
    }

    var SHOW_PROP = {
        REQUESTDATE: "showRequestDate",
        UPDOWN: "showUpDown",
        OPENPRICE: "showOpenPrice",
        RANGEHIGH: "showRangeHigh",
        RANGELOW: "showRangeLow",
        VOLUME: "showVolume",
    }

    var INFO_PROP = {
        CURRENTPRICE: "currentPrice",
        REQUESTDATE: "requestDate",
        UPDOWN: "upDown",
        OPENPRICE: "openPrice",
        RANGEHIGH: "rangeHigh",
        RANGELOW: "rangeLow",
        VOLUME: "volume"
    }

    var NAME = {
        CURRENTPRICE: "Current Price",
        REQUESTDATE: "Request Date",
        UPDOWN: "Up/Down",
        OPENPRICE: "Open Price",
        RANGEHIGH: "Range High",
        RANGELOW: "Range Low",
        VOLUME: "Volume"
```

```
    }

    var stockplex = {};
    stockplex.items = []; //each item holds the [Name, Value] to be displayed on the page

       stockplex.symbol = properties.get(CONST.PROP_SYMBOL,"");
    stockplex.summary = properties.get(CONST.PROP_SUMMARY,"");

    //If no symbol or summary has been given, show a touch ui placeholder
       if(stockplex.symbol == "" && stockplex.summary == ""){
              if(AuthoringUtils.isTouch){
                     //Set placeholder class for touch
                     stockplex.cssClass = "cq-placeholder";
              } else {
                     //Set placeholder class for classic
                     stockplex.cssClass = "cq-text-placeholder-ipe";
              }
       }

    //Get the current stock
    stockplex.currentPrice = properties.get(INFO_PROP.CURRENTPRICE, '100');

    //Add the appropriate properties to the item array to be displayed
    if(properties.get(SHOW_PROP.REQUESTDATE)){
      stockplex.items.push([NAME.REQUESTDATE, properties.get(INFO_PROP.REQUESTDATE,
'01/01/2016')]);
    }
    if(properties.get(SHOW_PROP.UPDOWN)){
      stockplex.items.push([NAME.UPDOWN, properties.get(INFO_PROP.UPDOWN, 10)]);
    }
    if(properties.get(SHOW_PROP.OPENPRICE)){
      stockplex.items.push([NAME.OPENPRICE, properties.get(INFO_PROP.OPENPRICE, 90)]);
    }
    if(properties.get(SHOW_PROP.RANGEHIGH)){
      stockplex.items.push([NAME.RANGEHIGH, properties.get(INFO_PROP.RANGEHIGH, 105)]);
    }
    if(properties.get(SHOW_PROP.RANGELOW)){
      stockplex.items.push([NAME.RANGELOW, properties.get(INFO_PROP.RANGELOW, 90)]);
    }
    if(properties.get(SHOW_PROP.VOLUME)){
      stockplex.items.push([NAME.VOLUME, properties.get(INFO_PROP.VOLUME, 2712018)]);
    }

    //Get the download button boolean from the Designer
    stockplex.downloadButton = currentStyle.get("downloadButton", false);

    return stockplex;


});
```

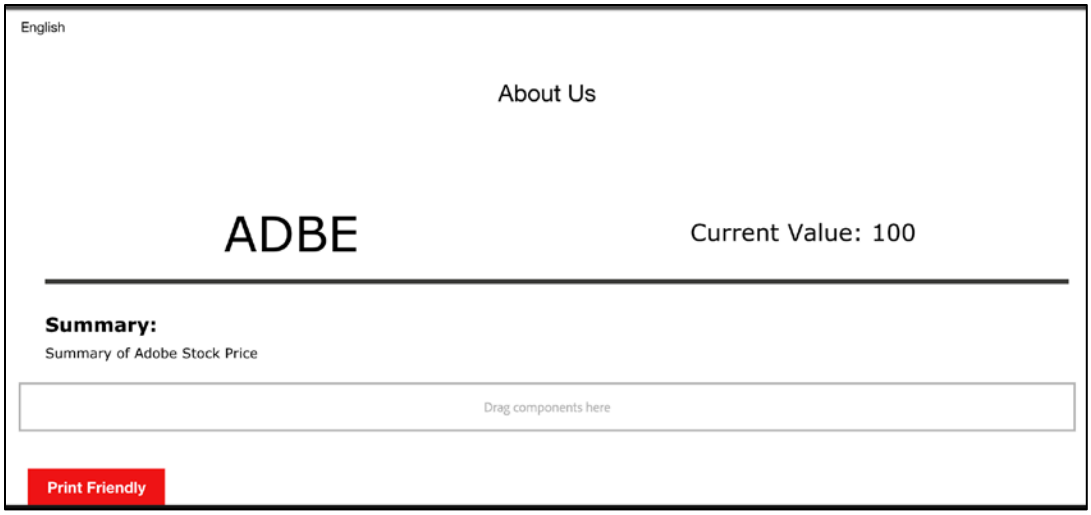
10. To take advantage of the new author input, replace the code for **stockplex.html** using the code from the Exercise_Files.

stockplex.html

```
<sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-sly-
call="${clientlib.all @ categories='apps.training.stockplex'}" />


<div id="stockplex" data-sly-use.stock="stockplex.js" class="${stock.cssClass}" data-
emptytext="StockPlex Component" >

    <div data-sly-test="${stock.symbol}">

        <table id="stockPrice">

            <tr>
```

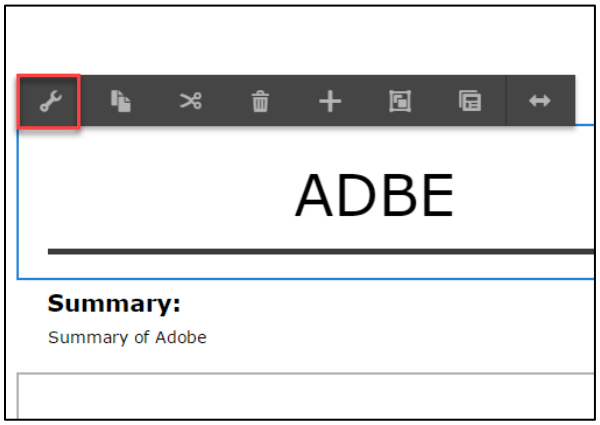
```
                <h1 id="symbol" scope="row" width="48%">${stock.symbol}</h1>
                <h1 id="currentPrice">Current Value: ${stock.currentPrice}</h1>
            </tr>
        </table>

        <div id="summary">
            <h3>Summary: </h3>
            <p>${stock.summary}</p>
        </div>

        <div id="download" data-sly-test="${stock.downloadButton}">
            <a
href="${'http://finance.google.com/finance/historical?q=NASDAQ%3A{0}&ei=TeGIWdnKI8LujAGbza3
YCw&output=csv' @ format=stock.symbol, context='uri'}"><Button>Download Stock
History</Button></a>
        </div>

        <div id="values" data-sly-list="${stock.items}">
            <table id="items">
                <tbody>
                    <tr>
                        <th >${item[0]}</th>
                        <td >${item[1]}</td>
                    </tr>
                </tbody>
            </table>
        </div>



    </div>
</div>
```

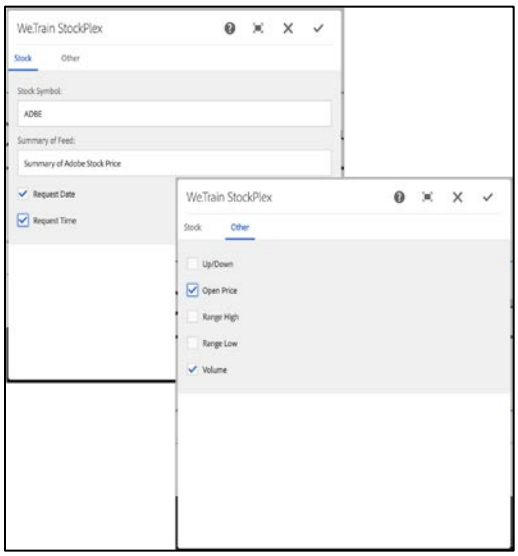
11. Using the Sites Console, reload the **About Us** page.



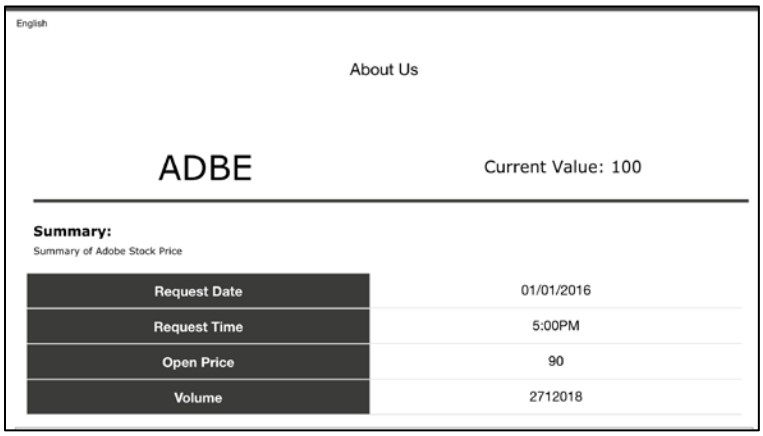
12. Notice that not much has changed. The current value has been filled in, but everything else is the same.

13. To get the additional values to appear, open the stockplex component dialog (in Edit mode) and check a few boxes.





14. Click the checkmark to save your choices.

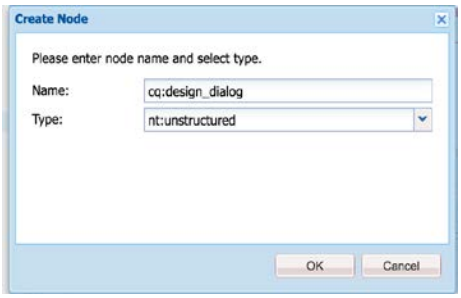


14.1.9 Task – Add a Design Dialog box

1. Using CRXDE Lite, navigate to **/apps/training/components/content/stockplex**.
2. Right-click on the stockplex component and select Create... > Create Node.

Node Name: cq:design_dialog

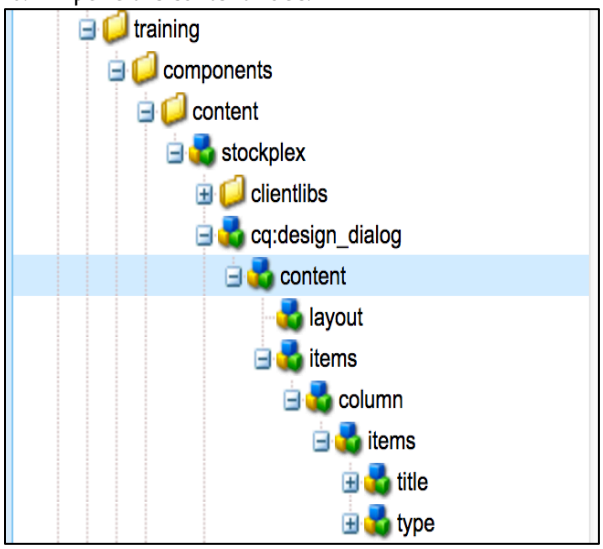
Node Type: nt:unstructured



3. Save.
4. Click OK.
5. Add the following two properties to the cq:design_dialog node.

Name	Type	Value
sling:resourceType	String	cq/gui/components/authoring/dialog
jcr:title	String	We.Train Stockplex

6. Save.
7. Navigate to **/libs/wcm/foundation/components/title/cq:dialog**.
8. Copy the content node and paste under **/apps/training/components/content/stockplex/cq:design_dialog**
9. Save.
10. Expand the content node.



11. Delete the title and type nodes at the bottom of the dialog structure.
12. Right-click on the **items** node at the bottom of the hieriarchy and select **Create… > Create Node.**

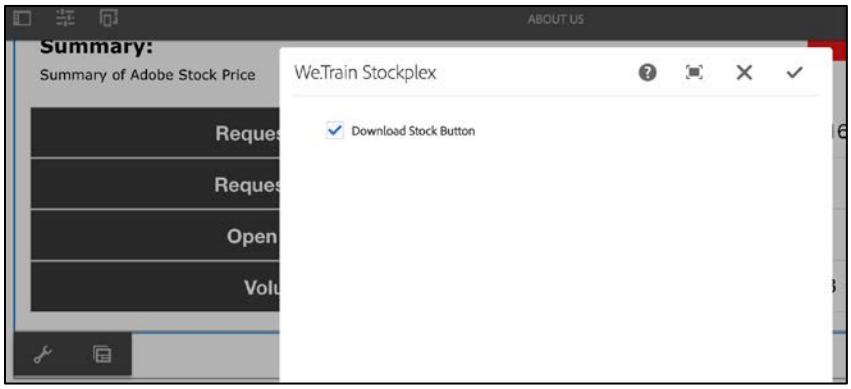
Name: downloadBox

Type: nt:unstructured

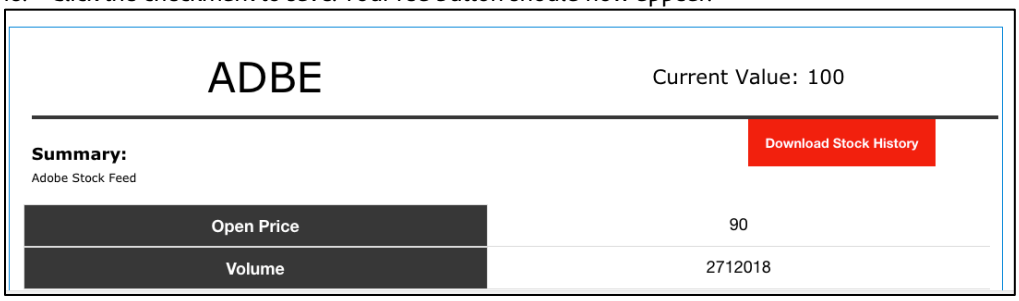
13. Define the following four additional properties on the downloadBox node.

Name	Type	Value
sling:resourceType	String	granite/ui/components/foundation/form/checkbox
text	String	Download Stock Button
name	String	./downloadButton
value	Boolean	true

14. Using the Sites Console, reload the About Us page.
15. Switch to **Design** mode and click on the StockPlex component.
16. Click on the wrench.
17. Select **Download Stock Button:**



18. Click the checkmark to save. Your red button should now appear:



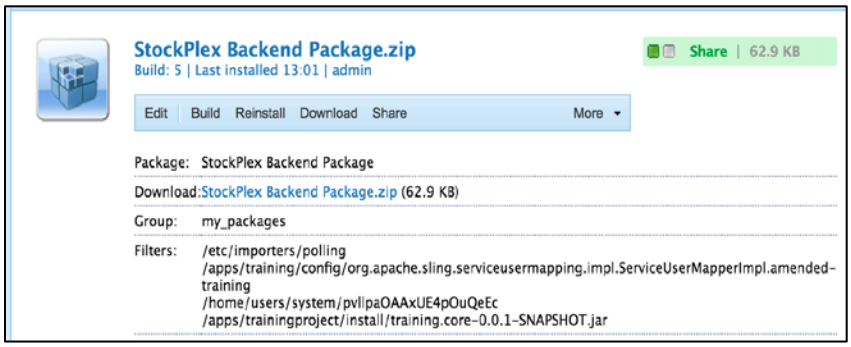
19. Take a moment to go back to the requirements and see if we have met them.

Exercise III (Optional) - Integrate Stockplex component with Live Data

The previous stockplex exercises have used data included in the javascript file. To integrate the stockplex component with live data, you must add business logic that queries an outside source. This is done by creating that business logic in a java class and packaging it into an OSGi bundle. For the purposes of this exercise, you will upload a content package that contains the OSGi bundle.

14.1.10 Task – Upload and Install the Stockplex Backend Package

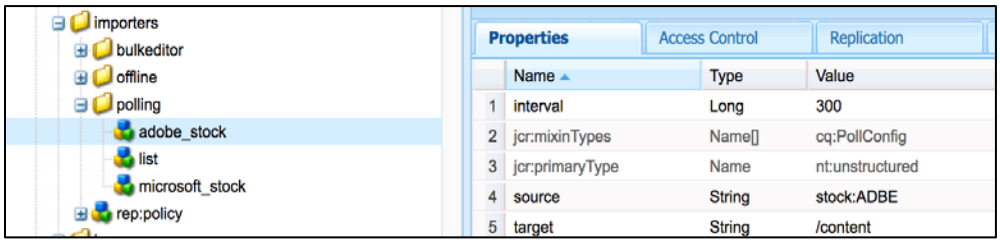
1. Using the package manager (http://localhost:4502/crx/packmgr/index.jsp), upload and install the content package "Stockplex Backend Package.zip" from the Exercise_Files.



Now let's investigate what the package installed.

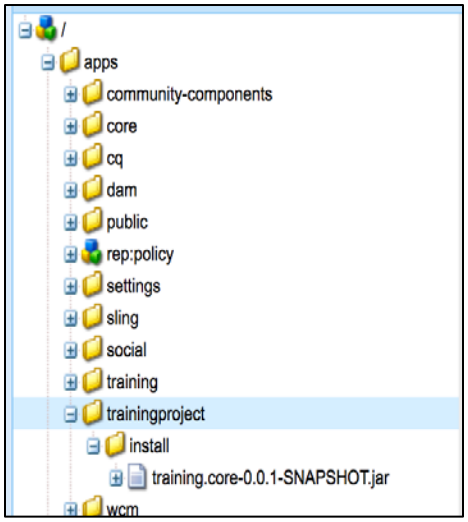
2. Using CRXDE Lite, navigate to **/etc/importers/polling**.

   The **adobe_stock** and **microsoft_stock** nodes in **/etc/importers/polling** configure the AEM polling importer with the custom importer. We import ADBE and MSFT stock with these values.



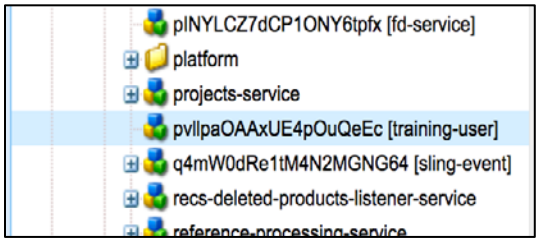
3. Navigate to **/apps/trainingproject/install**.

   This jar file is the OSGi bundle that contains the backend services for our custom importer and a custom model called StockModel that makes it easier to retrieve information from the custom importer.

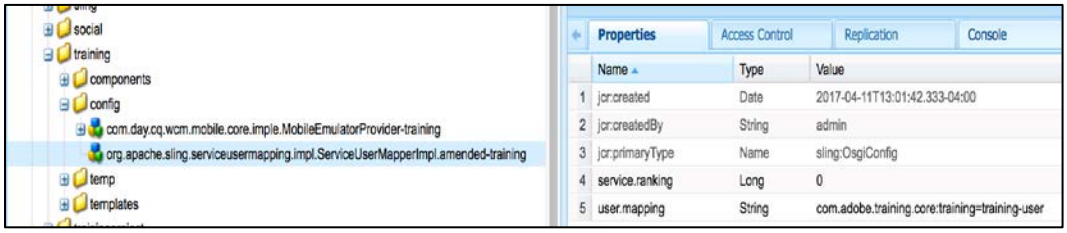


4. Navigate to **/home/users/system**. Scroll down to the nodes starting with the letter "p".

   The package installed a System User, named **training-user**, that is used to provide proper permissioning for the classes in the bundle.



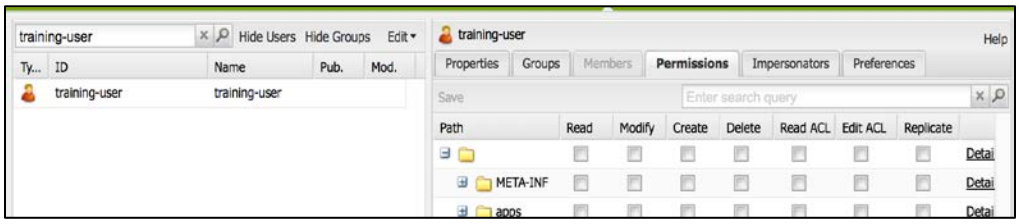
5. Navigate to **/apps/training/config.**
   Inside the config folder you will find a configuration that associates the training-user with the service. In this way, the permissions granted to the System User are associated with the service.



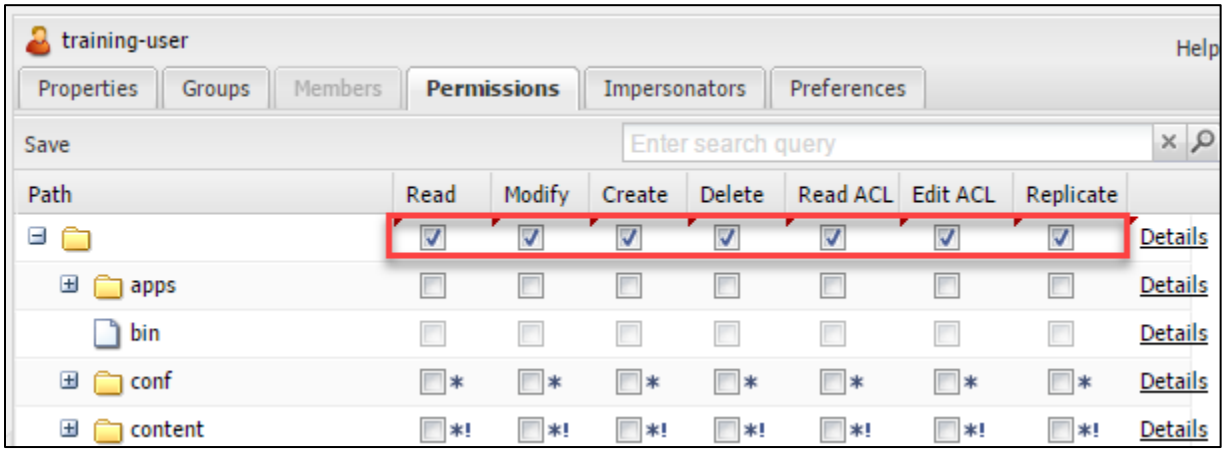
14.1.11 Task – Permission the Service User

Now you need to set permissions for the service user associated with the back-end services. Since the services run in the context of the service user, this will allow the back-end services to do their job.

1. Navigate to http://localhost:4502/useradmin, or use the Global Navigation: **Tools** > **Security** > **Permissions.**
2. Search for "training-user" and double-click on the user to bring it into context.

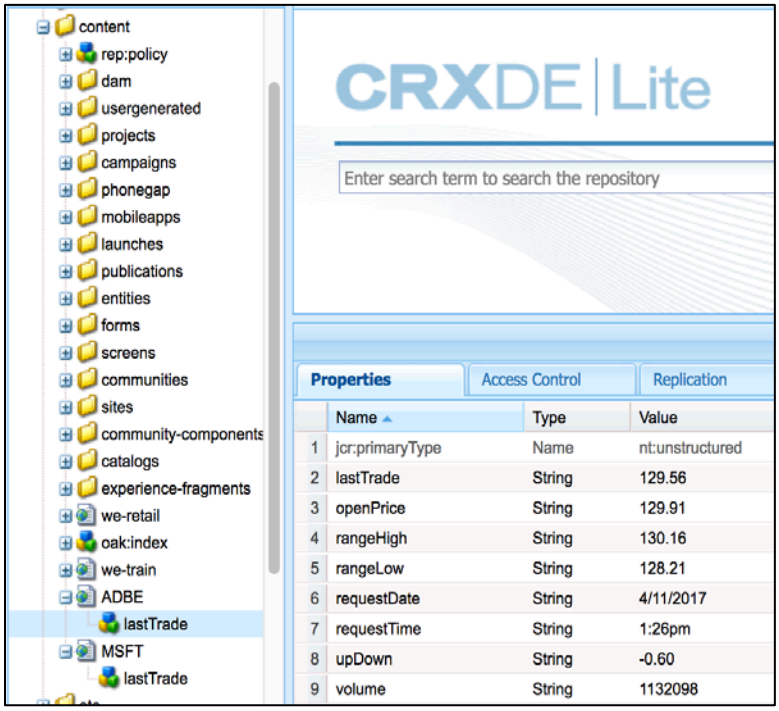


3. Click on the **Permissions** tab.
4. Select all permissions at the root.



5. Save.

6. Using CRXDE Lite, navigate to **/content/ADBE/lastTrade**. Notice the property values.
NOTE: You may need to refresh CRXDE Lite completely to see the new ADBE and MSFT nodes.

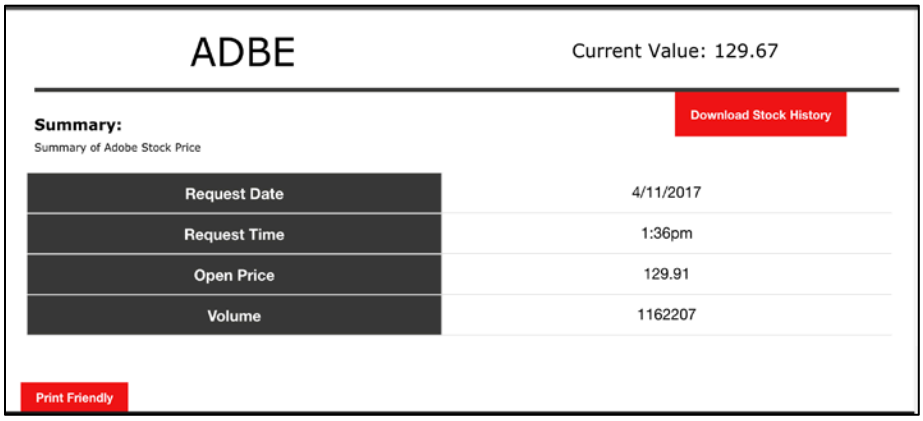


7. Navigate to **/content/MSFT/lastTrade**. Notice the property values.

NOTE: You must wait five minutes before the first data import. This is the default minimum wait time for importers.

14.1.12 Task – Modify stockplex.js to use imported data

1. Using CRXDE Lite, navigate to **/apps/training/components/content/stockplex**..
2. Replace the code in **stockplex.js** with the code from the Exercise_Files..
3. Using the Sites Console, navigate to **Sites** > **We.Train** > **English** and open the About Us page. (Remember, this page is where you placed the Stockplex component..)



Now the Stockplex component is using live data retrieved by the polling importer. NOTE: All request times are Eastern Time (UTC -5).

Exercise IV – Search Component

The following instructions explain how to create a component that will allow visitors to search the content of the website/repository. This exercise will demonstrate the differences among the multiple Search APIs. The search component can be placed in the responsive grid of any page, and has the ability to search the content of the website based on a query string provided in the request.

14.1.13 Task – Create a search component

1. Navigate to **/apps/training/components/structure**.
2. Right-click structure, and select **Create…** > **Create Component**.
3. Enter the following values:

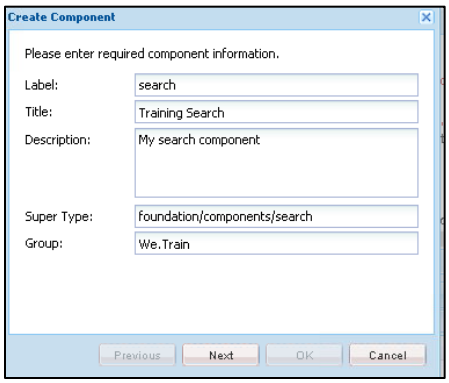
   Label: search

   Title: Training Search

   Description: My search component

   Super Type: foundation/components/search

   Group: We.Train



4. Click **Next**, and then click **OK**.
5. Save all.
6. Expand the **search** component. Rename the **search.jsp** file to **search.html**.
7. Replace the contents of **search.html** with the contents of **search-jcr.html** from the Exercise_Files.

search.html

```
<center data-sly-use.search="search-jcr.js">
    <form action="${currentPage.path}.html">
    <input name="q" value="${search.queryString}" />
          <input value="Search" type="submit" />
       </form>
</center>
<br />

<div class="searchresult" data-sly-list="${search.query}">
       <a href="${item.path}.html">
       <p style="color: rgb(51,102,255);">${item.title}
        </p>
    </a>
</div>
```

```
<div class="searchresult" data-sly-test="${!search.query}">No results found ...
Please try again ...</div>
```

8. Create a new file under the **search** component node, and name it **search-jcr.js**.
9. Save all.
10. Replace the file contents with that of **search-jcr.js** from the Exercise_Files.

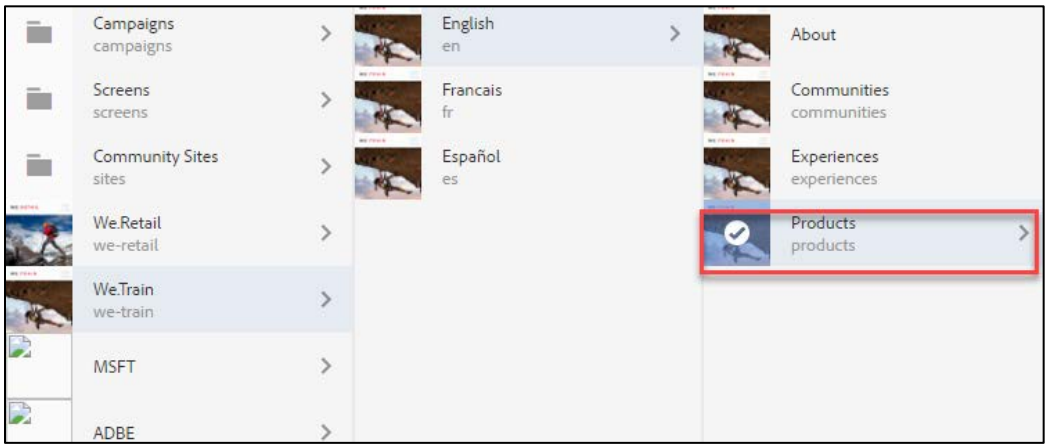
search-jcr.js

```
"use strict";


use(function(){
      var queryString = (request.getParameter("q") != null) ?
request.getParameter("q") : "";
    var items = [];


      if (request.getParameter("q") != null) {
        var stmt = "select * from cq:Page where jcr:path like '/content/we-train/%'
and contains(*, '" + request.getParameter("q") + "') order by jcr:score desc";
        var query =
currentNode.getSession().getWorkspace().getQueryManager().createQuery(stmt,
javax.jcr.query.Query.SQL);
        var results = query.execute();
        if (results.getNodes() != null && results.getNodes().hasNext()) {
            var it = results.getNodes();
            while (it.hasNext()) {
                var node = it.nextNode();
                var npath = node.getPath();
                var contentPage =
pageManager.getContainingPage(resource.getResourceResolver().getResource(npath));
                items[items.length] = contentPage;
            }
        }
    }


    return{
        queryString: queryString,
        query: items
    };
});
```

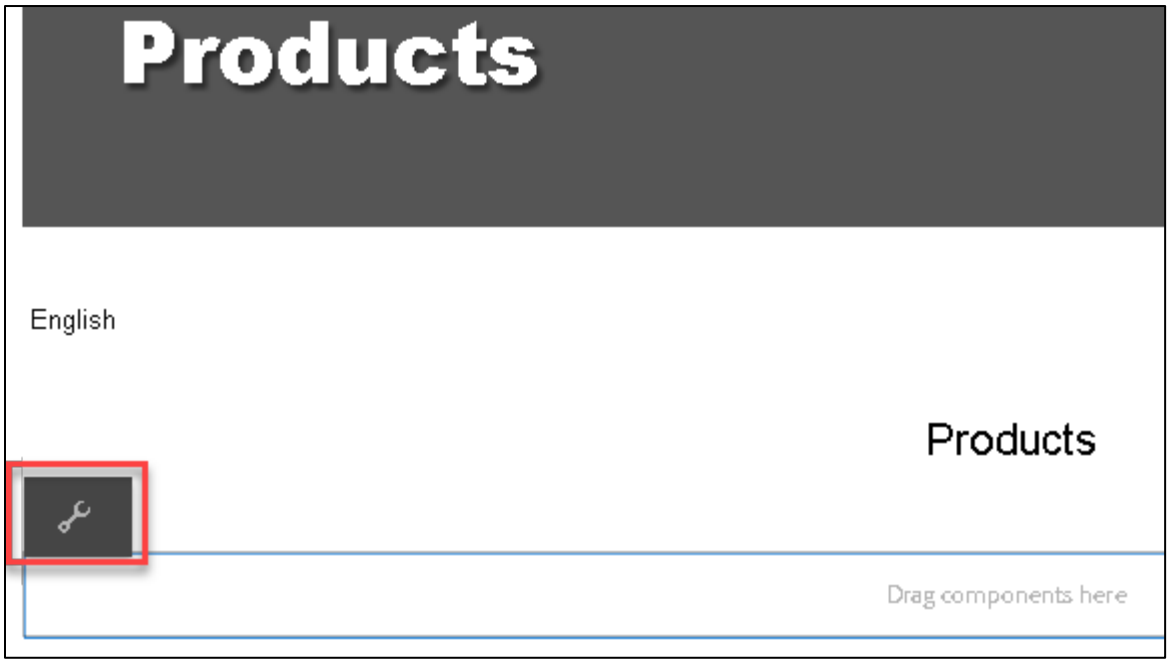
NOTE: The search path in **search-jcr.js** (line 8, in bold above) matches the path to your training site (**/content/we-train**).

11. Save your changes.

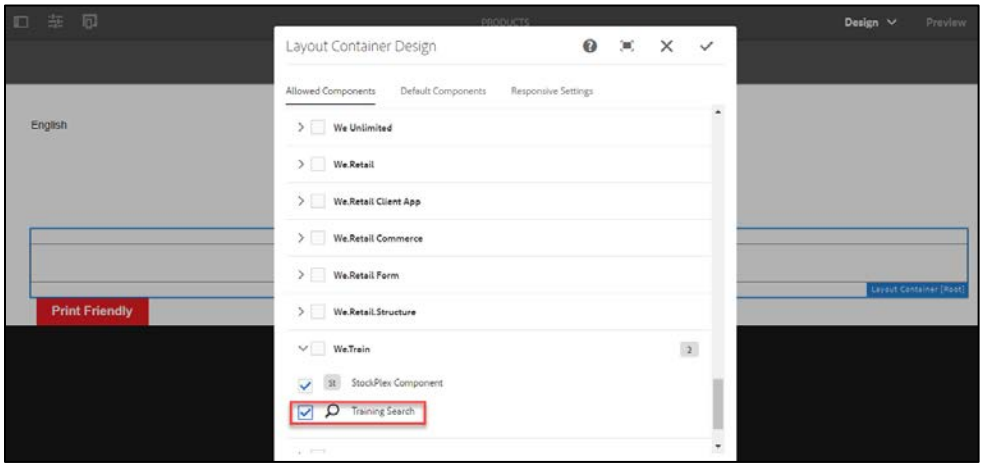
12. Open the **Products** page of your site in **Design mode.** Recall that you navigate to this page using this path shown below. Notice that you may see the MSFT and ADBE pages if you did the previous optional exercise (Exercise III):



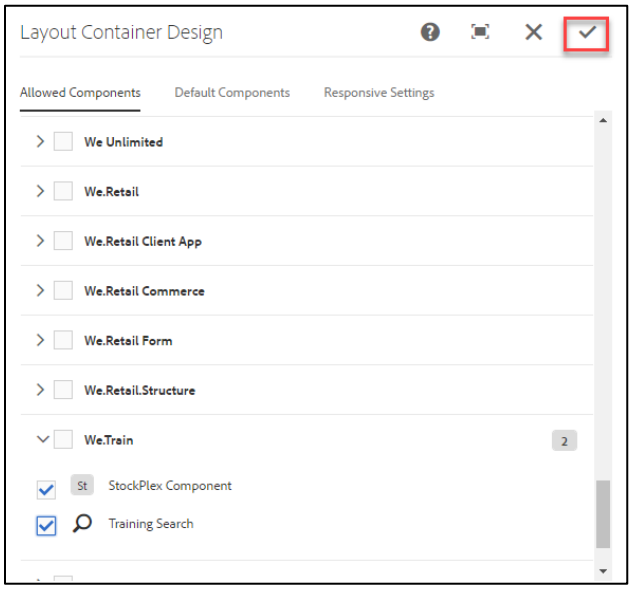
13. Click the **layout container** (responsivegrid) near the middle of your page. Click **Configure** (the wrench):



14. Scroll to the bottom of the dialog and add your search component to the layout container component. Notice how the component will be found in the 'We.Train' group. This was defined during the creation of the component.



15. Click **Done** at the upper right of the dialog to save your changes.



16. Go to **Edit Mode**, and test your work by adding the **Training Search** component to the layout container component of the **Products** page.
NOTE: The process of adding a component to the responsive grid is covered in Exercise I of this module (**Task 14.1.2**). Refer to those instructions if you need to review how to add components in Edit mode. If successful, you should see the default search component.

17. Switch to **Preview mode** and search for a word you know exists on a separate page in the We.Train website structure. You may need to perform this search in page preview mode to ensure a clean request. For example, here is the result of searching for "about" which locates the "About Us" page:

18. Now replace the contents of **search.html** with the contents of **search-wcm.html** from the Exercise_Files.

search.html

```
<center data-sly-use.search="search-wcm.js">
    <form action="${currentPage.path}.html">
              <input size="41" maxlength="2048" name="q"
value="${search.queryString}" />
              <input value="${search.searchButton}" type="submit" />
       </form>
</center>

<br />

<!--/* Displays the results for this page */-->
<div class="searchresult" data-sly-list="${search.hits}">
       <a href="${item.uRL}">${item.title}</a>
    <br/>
</div>
<div class="searchresult" data-sly-test="${!search.hits}">${search.noResults}</div>

<br/><br/>

<!--/* Displays the page links on the bottom of the component */-->
<a data-sly-test="${search.prevPage}" href="${search.prevPage.uRL ||
'#'}">${search.previousText}</a>

<sly data-sly-list.resultPage="${search.resultPages}">
       <sly data-sly-
test.curPage="${resultPage.isCurrentPage}">${resultPageList.count}</sly>
       <a data-sly-test="${!curPage}"
href="${resultPage.uRL}">${resultPageList.count}</a>
</sly>

<a data-sly-test="${search.nextPage}" href="${search.nextPage.uRL ||
'#'}">${search.nextText}</a>
```

19. Examine the code to see the differences between the JCR search API and the WCM search APIs.
20. Create a new file under the **search** component and name it **search-wcm.js**.
21. Replace the contents of the file with that of **search-wcm.js** from the Exercise_Files.

search-wcm.js

```
"use strict";


use(function(){
       var CONST = {
        PROP_NO_RESULTS_TEXT: "noResultsText",
        PROP_SEARCH_BUTTON: "searchButtonText",
        PROP_PREV_TEXT: "previousText",
        PROP_NEXT_TEXT: "nextText"
    }


    var hits;
    var noResultsText = granite.resource.properties[CONST.PROP_NO_RESULTS_TEXT] ||
"No Results";
       var searchButtonText = granite.resource.properties[CONST.PROP_SEARCH_BUTTON]
|| "Search";
    var previousText = granite.resource.properties[CONST.PROP_PREV_TEXT] || "back";
    var nextText = granite.resource.properties[CONST.PROP_NEXT_TEXT] || "next";
    var prevPage;
    var nextPage;


       /* initialize CQ5 WCM Search */
    var search = new com.day.cq.wcm.foundation.Search(request);


       /* process search results */
    var result = search.getResult();


    /* handle no results */
    if (result == null || result.getHits() == null) {
              queryEmpty = currentPage.properties.noResultsText;


    /* handle result list */
    } else {
              hits = result.getHits();


        /* pagination */
        var resultPages = result.getResultPages();
        if (!resultPages.isEmpty()) {
            if (result.getPreviousPage() != null) {
                prevPage = result.getPreviousPage();
            }
            if (result.getNextPage() != null) {
                nextPage = result.getNextPage();
            }
```

```
            }
    }

    return{
            noResults: noResultsText,
        searchButton: searchButtonText,
        previousText: previousText,
        prevPage: prevPage,
        nextText: nextText,
        nextPage: nextPage,
        hits: hits,
        resultPages: resultPages
    };
});
```

22. Save all changes, refresh your browser, and try the search again using the Preview mode .

Congratulations! You successfully created a search component that queries the content in your website structure. You can further enhance this component by adding widgets to the dialog box to output default messages written by a content author if the search was successful or unsuccessful.

Using Ajax in Components

Ajax is used to retrieve data from AEM. For example, retrieve a list of users from AEM and display them within a component. This executes a callback to AEM to retrieve JSON-formatted data, which involves providing a JSON request handler. This process is suggested when dealing with dynamic content on a static site.

jQuery Flexigrid Plug-In

Flexigrid is a lightweight but rich data grid with resizable columns and a scrolling data to match the headers, plus an ability to connect to an xml/json based data source using Ajax to load the content. Flexigrid is available at https://github.com/paulopmx/Flexigrid.

    - Has resizable columns
    - Creates scrolling data to match the headers
    - Can connect to an XML or JSON data source using Ajax to load the content
- Similar in concept with the Ext Grid, but is pure jQuery
- Follows the jQuery mantra of running with the least amount of configuration

Exercise V – (Optional) Using JQuery and Ajax with an AEM component

Create a custom component that will display a dynamic grid of user accounts. This component will make use of the jQuery plugin that uses Ajax to retrieve data from AEM. The jQuery Flexigrid plugin is the target for this exercise. The plugin will execute a callback to AEM to retrieve JSON-formatted data.

14.1.14 Task – Create a User Grid component

1. Using CRXDE Lite, navigate to **/apps/training/components/content**.
2. Right-click on the content folder and select **Create… > Create Component**.
3. Enter the following information in the Create Component dialog and click **Next** and then **OK**.

Name	Value
Label	user-grid
Title	User Account Grid
Description	A custom component to display a dynamic grid of user accounts
Group	We.Train

4. Save.
5. Expand the user-grid component and rename user-grid.jsp to user-grid.html.
6. Replace the contents of user-grid.html with the contents of user-grid-0.html from your Exercise_Files.

user-grid-0.html

```
<!--/*
            Step 1 of User-Grid. Shows the basic table.
*/-->

<table class="user-table">
      <thead>
        <tr>
            <th width="100">Col 1</th>
            <th width="100">Col 2</th>
            <th width="100">Col 3 is a long header name</th>
            <th width="300">Col 4</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>This is data 1 with overflowing content</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
```

```
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
    </tbody>
    </table>
```

7. Save.

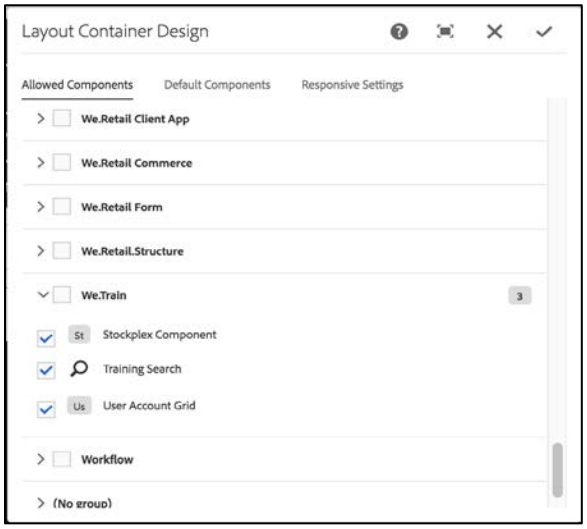
14.1.15 Task – Create a placeholder dialog

Now create a placeholder dialog.

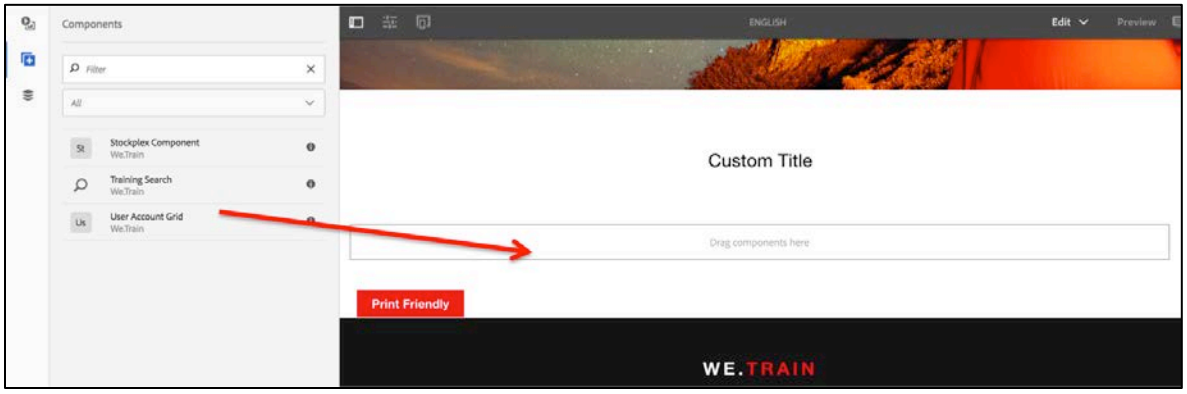
1. Right-click on the user-grid component node and select Create … Create Node from the context menu.

Name	Type
cq:dialog	nt:unstructured

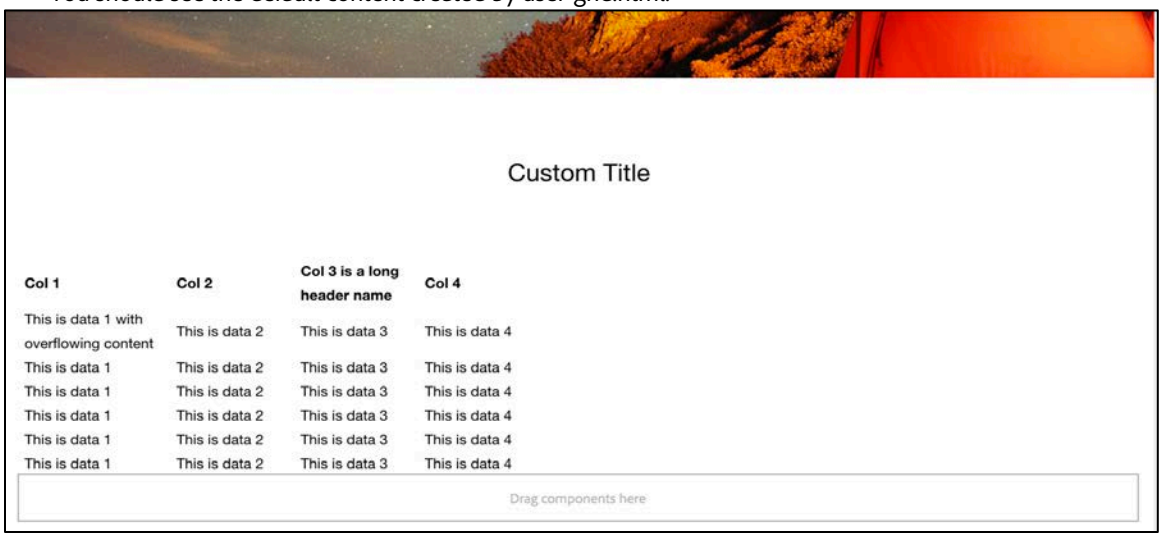
2. Save.
3. Using the Sites Console, navigate to the We.Train site and, within Design mode, enable the **user-grid** component (User Account Grid) in the responsive grid.



4. Drag the user-grid component on to an empty page.



You should see the default content created by user-grid.html.



14.1.16 Task – Define the User Grid client library

In order to control the look and feel of the user-grid component, you need to define a client library.

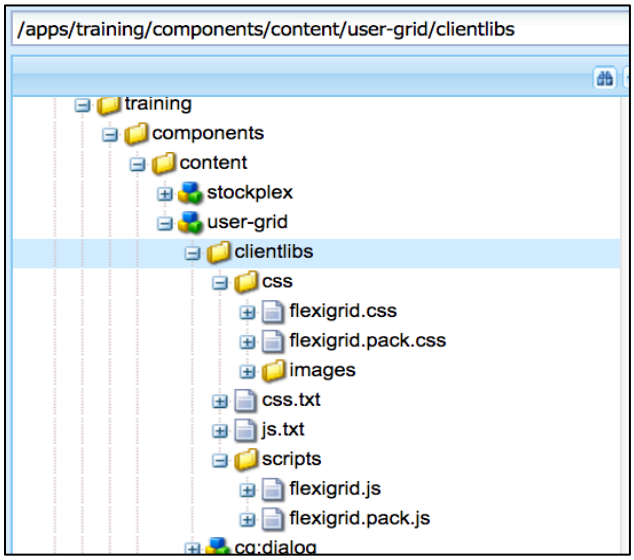
1. To create a client library folder under the user-grid component, right-click the user-grid component and select **Create…** > **Create Node**.

Name	Type
clientlibs	cq:ClientLibraryFolder

2. Add the following properties to the clientlibs node:

Name	Type	Value
categories	String[]	componentlab
dependencies	String[]	cq.jquery

3. Using the package manager, upload and install the **user-grid-libs-1.0.zip** package from the Exercise_Files. This package uploads the assets, that you need to develop the user-grid component, into the repository. The user-grid client library files will be installed at **/apps/training/custom**.
4. Refresh the /apps/training folder to see the uploaded files.
5. Copy the contents of the custom folder that you just uploaded to the /apps/training/components/content/usergrid/clientlibs folder. The directory structure should look as shown below:



6. Examine the files in the client library folder.

Now modify the user-grid.html to use the client library.

7. Replace the contents of the user-grid.html with that of user-grid-1.html from the Exercise_Files.

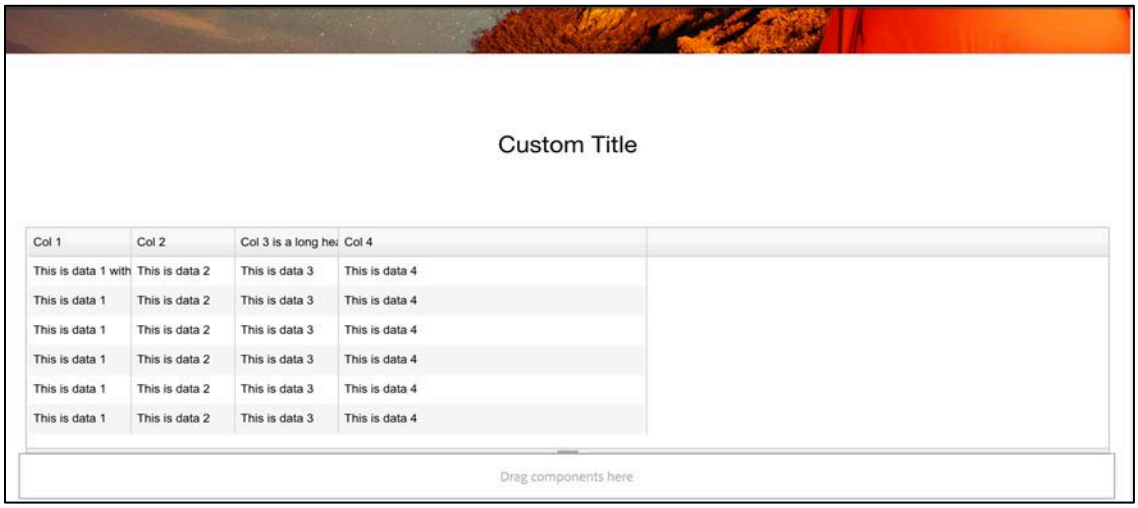
user-grid-1.html

```
<!--/*
            Step 2 of User-Grid. Shows the basic table with the flexgrid enabled.
*/-->


<sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"
     data-sly-call="${clientlib.all @ categories='componentlab'}"/>

<script type="text/javascript">
```

```
$CQ(function () {
    $CQ('.user-table').flexigrid();
});


</script>

<table class="user-table">
      <thead>
        <tr>
            <th width="100">Col 1</th>
            <th width="100">Col 2</th>
            <th width="100">Col 3 is a long header name</th>
            <th width="300">Col 4</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>This is data 1 with overflowing content</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
        <tr>
            <td>This is data 1</td>
            <td>This is data 2</td>
            <td>This is data 3</td>
            <td>This is data 4</td>
        </tr>
    </tbody>
    </table>
```

8. Save.

9. Refresh the page with the user-grid component.



14.1.17 Task – Add the Ajax callback

Now add the Ajax callback to populate the grid with content from the repository.

1. Replace the contents of user-grid.html with that of user-grid-2.html from the Exercise_Files.

user-grid-2.html

```
<!--/*
            Step 3 of User-Grid. Shows the AEM user accounts with the flexgrid
enabled.
*/-->

<sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"
       data-sly-call="${clientlib.all @ categories='componentlab'}"/>

<script type="text/javascript">

    /* Grab the JCR path to the content entry that calls this component
     * with Sling, you cannot call a script, you must call the jcr content
     * node that resolves to the representation (script).
     */
    var baseURL = "${currentNode.path @ context='scriptString'}";

    $CQ(function () {
        $CQ('.user-table').flexigrid({
            url: baseURL + '.json', // This will trigger a POST request back to CQ
            dataType: 'json', // The expected response will be JSON formatted data
            colModel : [ {
                display : 'User ID', name : 'id', width : 215, sortable : true, align
: 'left', hide: false
            }, {
                display : 'First Name', name : 'givenName', width : 100, sortable :
true, align : 'left', hide: false
            }, {
                display : 'Last Name', name : 'familyName', width : 100, sortable :
true, align : 'left', hide: false
            },{
                display : 'Email', name : 'email', width : 215, sortable : true, align
: 'left', hide: false
            }],
            buttons : [
                {name: 'Add', bclass: 'add', onpress : userAdd},
                {name: 'Edit', bclass: 'edit', onpress : userEdit},
                {name: 'Delete', bclass: 'delete', onpress : userDelete},
                {separator: true}
                ],
           searchitems : [
                {display: 'User ID', name : 'user_id',isdefault: true},
                {display: 'First Name', name : 'givenName'},
                {display: 'Last Name', name : 'familyName'}
                ],
            sortname: "id",
            sortorder: "asc",
            usepager: true,
            title: "User Account Grid",
            useRp: true,
            rp: 15,
            showTableToggleBtn: false,
            singleSelect: true,
            width: 700,
            height: 200
        });
    });
```

```
    function userAdd() {
      alert("Add button clicked");
    }

    function userEdit() {
            alert("Edit button clicked.");
    }

    function userDelete() {
      alert("Delete button clicked.");
    }

</script>

<table class="user-table">

</table>
```

2. Create a new file, named user-grid.json.POST.html under the user-grid component.
3. Paste the contents of user-grid.json.POST.html, from the Exercise_files into the file you just created.

```
<sly data-sly-use.users="UserGrid"/>
```

4. Create a new file named UserGrid.java under the user-grid component.
5. Paste the contents of UserGrid.java from the Exercise_Files into the file you just created.

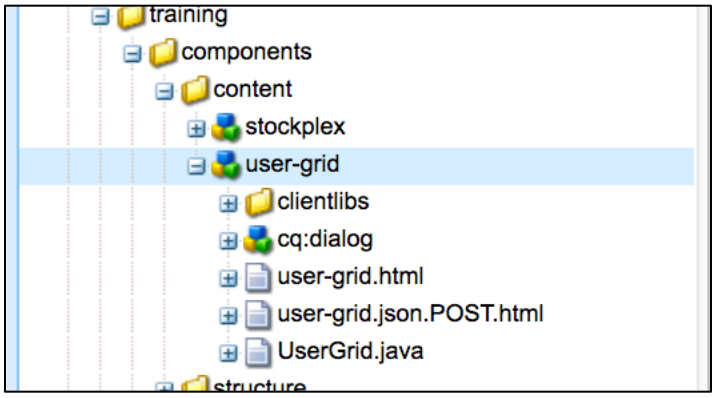
UserGrid.java

```
package apps.training.components.content.user_grid;

import com.adobe.cq.sightly.WCMUsePojo;
import org.apache.sling.jcr.api.SlingRepository;
import com.day.cq.commons.TidyJSONWriter;

import org.apache.jackrabbit.api.JackrabbitSession;
import org.apache.jackrabbit.api.security.user.Authorizable;
import org.apache.jackrabbit.api.security.user.UserManager;
import org.apache.jackrabbit.api.security.user.User;
import org.apache.jackrabbit.api.security.user.Query;
import org.apache.jackrabbit.api.security.user.QueryBuilder;

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;

import javax.jcr.*;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class UserGrid extends WCMUsePojo {
      private final Logger logger = LoggerFactory.getLogger(getClass());
    @Override
    public void activate() throws Exception{

      final SlingRepository repos =
getSlingScriptHelper().getService(SlingRepository.class);

        JackrabbitSession jcrSession = null;
```

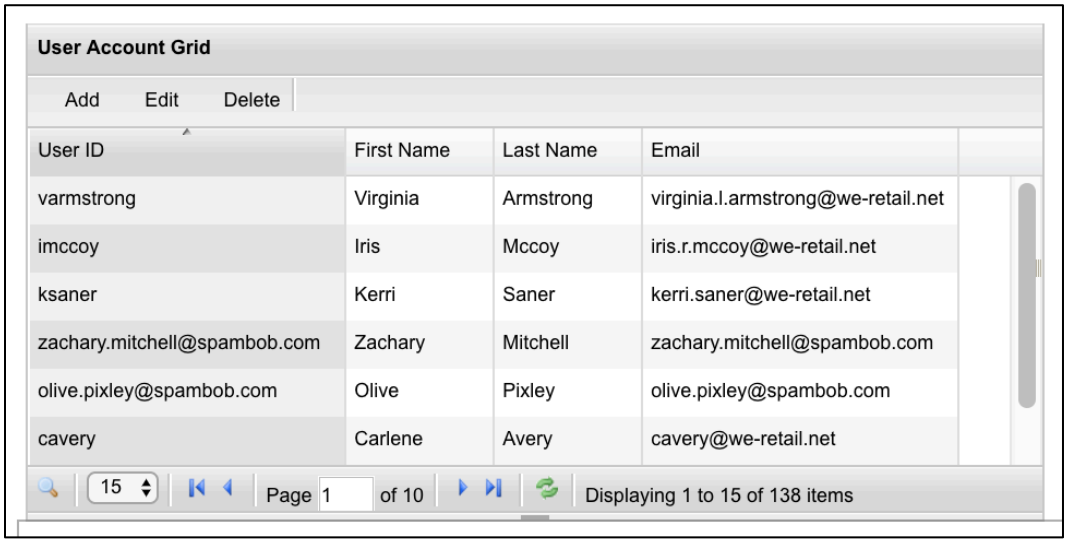
```
        Iterator<Authorizable> authorizableIterator = null;
        try
        {
            // Ensure that the currently logged on user has admin privileges.
             jcrSession = (JackrabbitSession)repos.loginAdministrative(null);

            final UserManager um = jcrSession.getUserManager();
            final TidyJSONWriter writer = new
TidyJSONWriter(getResponse().getWriter());

            authorizableIterator = um.findAuthorizables(new Query() {
                public  void build(QueryBuilder builder) {
                    builder.setSelector(User.class);
                }
            });

            List<Authorizable> users = new ArrayList<Authorizable>();

            // copy iterator into a List for additional manipulations.
            Authorizable tmpUser;
            while(authorizableIterator.hasNext())
            {
                tmpUser = authorizableIterator.next();
                users.add((User)tmpUser);

            }

            writer.setTidy("true".equals(getRequest().getParameter("tidy")));
            writer.object();
            writer.key("page").value(1);
            writer.key("total").value(users.size());
            writer.key("rows").array();

            for(int i=0; i < users.size(); i++)
            {

                Authorizable aUser = users.get(i);

                if(aUser.hasProperty("./profile/givenName") &&
                          aUser.hasProperty("./profile/familyName") &&
                          aUser.hasProperty("./profile/email")){

                   writer.object();
                    writer.key("id").value(aUser.getID());
                    writer.key("cell").array();

                    writer.value(aUser.getID());

writer.value(aUser.getProperty("./profile/givenName")[0].getString());

writer.value(aUser.getProperty("./profile/familyName")[0].getString());
                    writer.value(aUser.getProperty("./profile/email")[0].getString());

                    writer.endArray();
                    writer.endObject();

                }

            }

                writer.endArray();
                writer.endObject();
                jcrSession.logout();
        }
```

```
        catch (Exception e)
        {
            logger.error("myajaxsample Exception Occured: " + e.getMessage());
        }
        finally
        {
             jcrSession.logout();
             jcrSession = null;
        }
    }
}
```

6. At the end of these steps, you should have the following node structure:



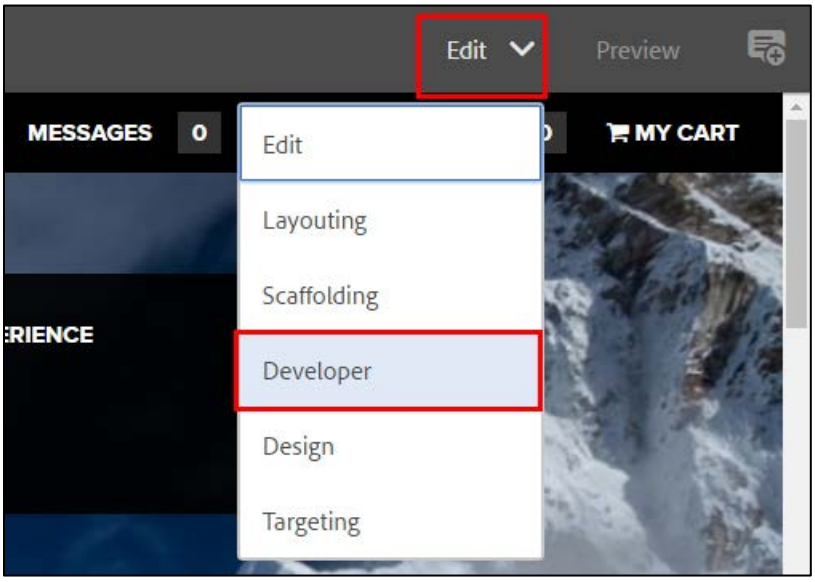
7. Test your component by refreshing the page that contains the user-grid component.



15 DEBUGGING AND TESTING

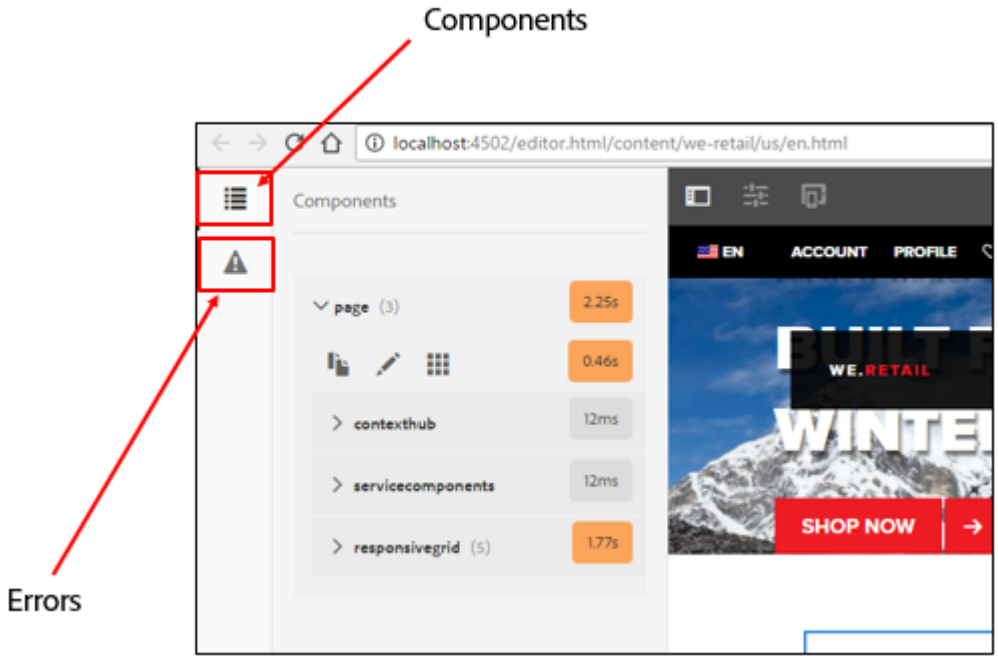
Developer Mode

You can access Developer mode as shown below:



The Developer mode provides you with two tools:

1. Components (server side computation time)
2. Errors (details of any errors /broken components on a page)



Components

The Components tab provides you with a components tree of used components on the page. For each component, it provides you with the associated script. You can click the Edit icon to navigate directly to the script in CRXDE Lite. The component tree also provides you with the server-side computation time for the component. It helps you to identify the slow and heavy components that need further optimization.

Error

The Error tab displays the error details of the broken components. You can see the required details without inspecting the log files.

Parameter Debugging

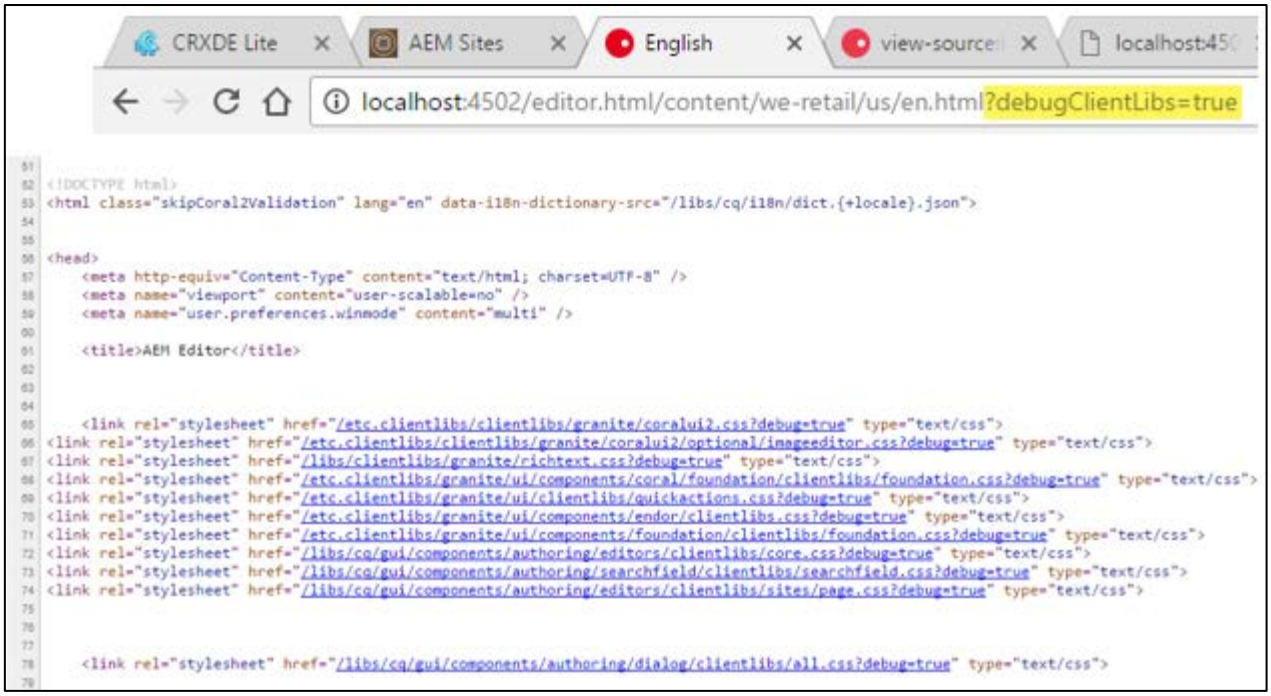
- Append the `?debug=layout` parameter to get information about renderers, selectors, resources, and so on.

For example:

?debugClientLibs=true

- Use this parameter to display the list of **client libraries** loaded onto the page shown in the page source HTML.

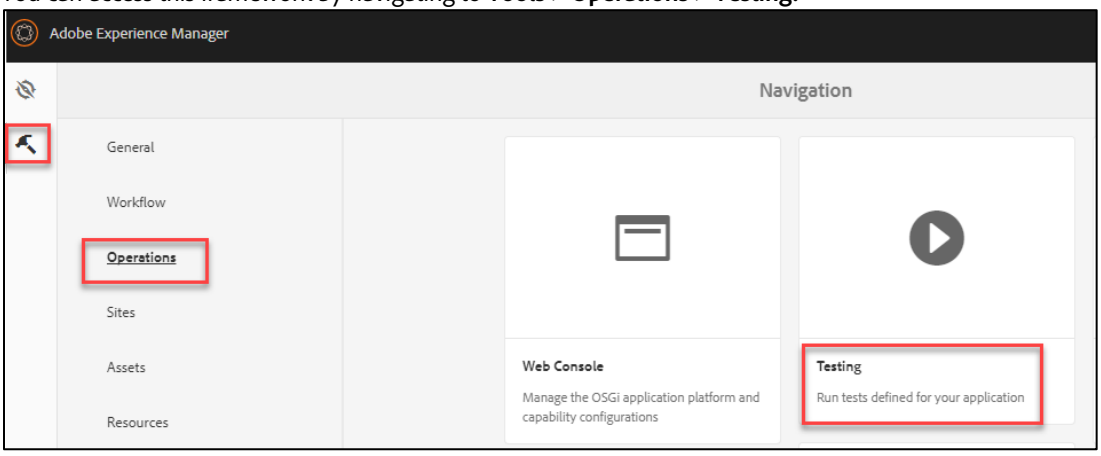
For example:



Performing Hobbes Tests

Functional testing is a form of testing that is dedicated to the user experience. Before you can publish your site, you need to ensure that every component in it is fixed and in full working order. Adobe Experience Manager has an out-of-the-box functional testing framework embedded in its UI, and it uses the Hobbes.JS testing framework as its base. This framework provides a streamlined experience for ensuring the usability of interactive elements on your site.

You can access this framework by navigating to **Tools** > **Operations** > **Testing:**



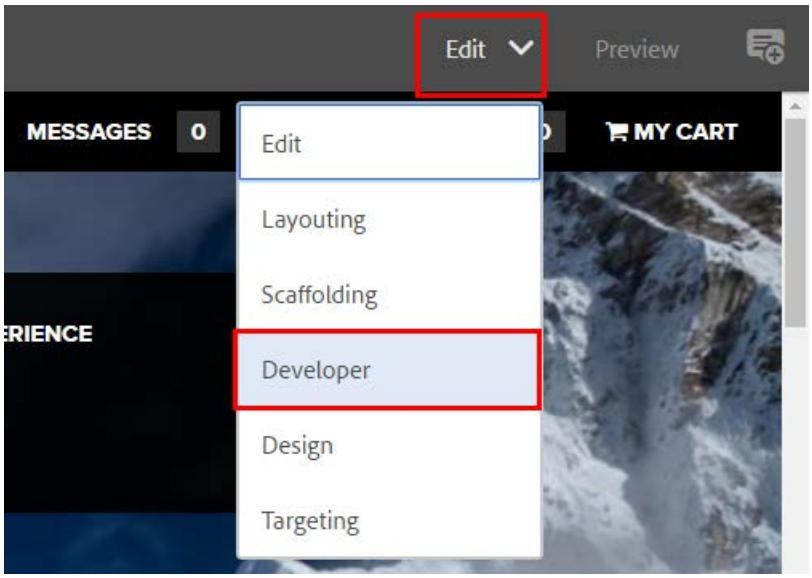
For more information, visit: https://docs.adobe.com/docs/en/aem/6-3/develop/ref/test-api/index.html.

Exercise

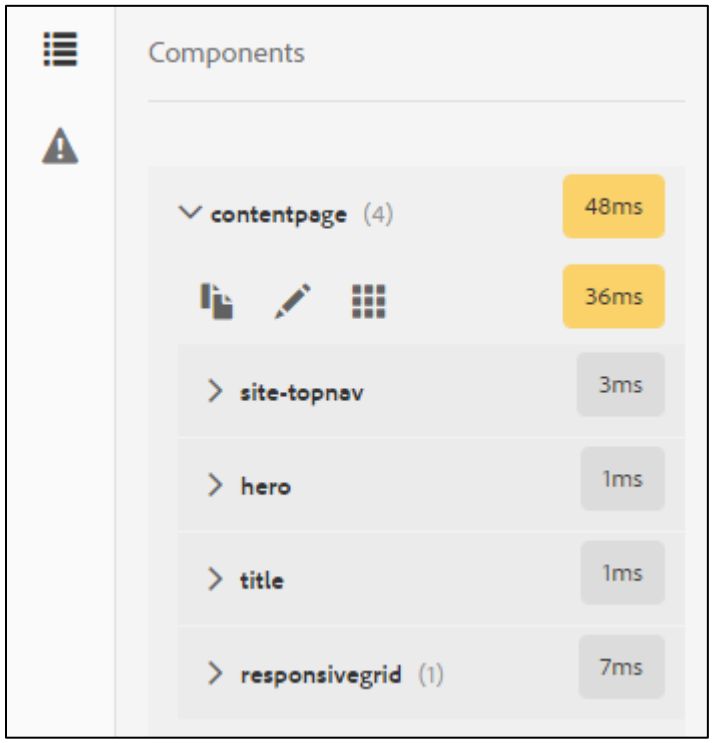
15.1.1 Task – Monitor component-based timing

In Developer mode of a page, you can view the time taken by the components to load.

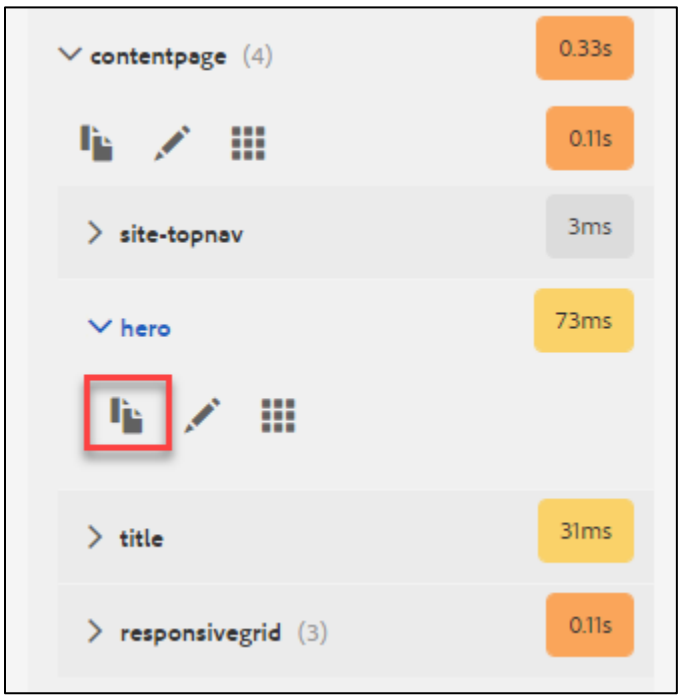
1. **Open the We.Train English page (or any page of your choosing) and select Developer mode.**

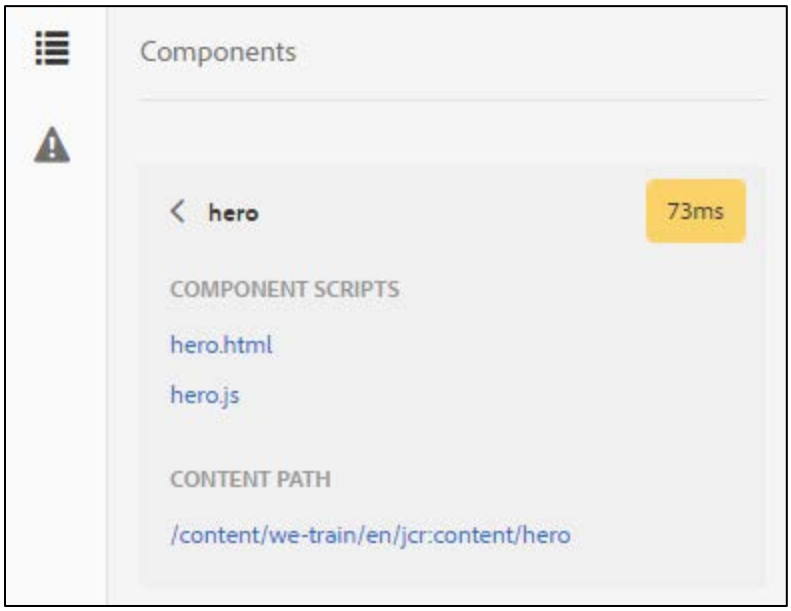


2. **Go to the Components tab. This tab displays a component tree that provides you with the server-side computation time for each component.**



3. **Click to expand one of the components and then click Details to view the component scripts:**





15.1.2 Task – Create the Hobbes Testing Suite

Hobbes Test Suites are created as modified client libraries inside the project. The client library has specific category and dependency values. Each Test Suite is a JavaScript file within the client library.

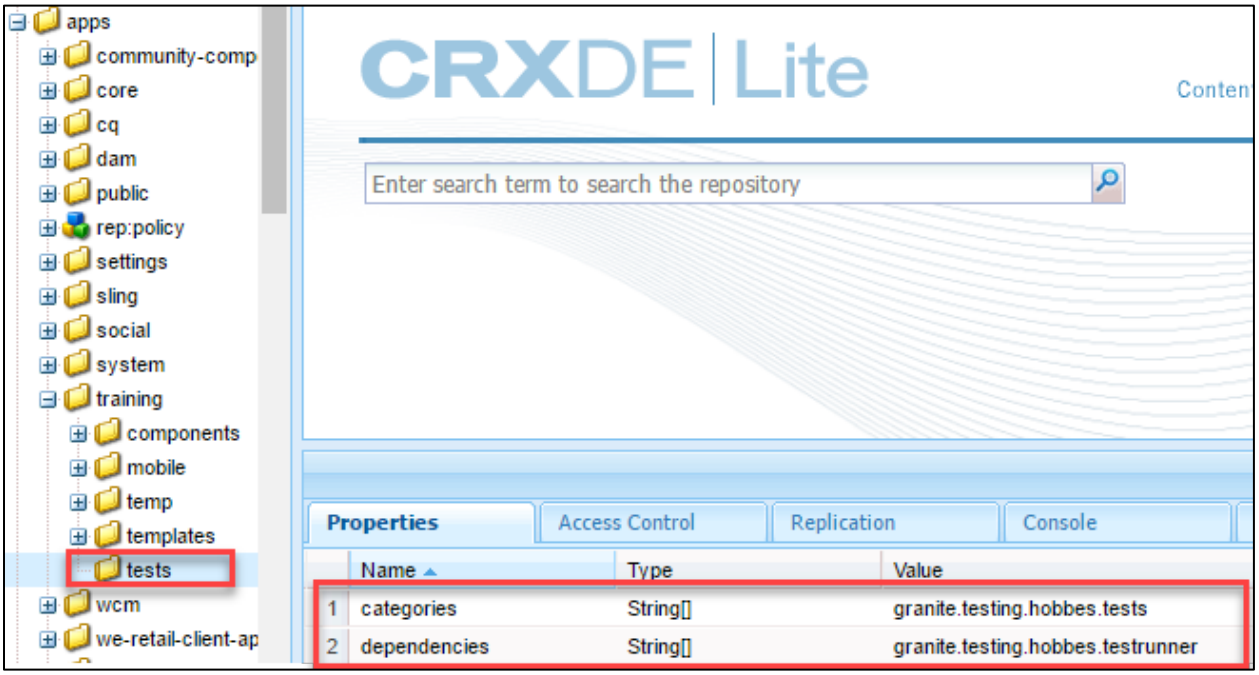
1. **Using CRXDE Lite, navigate to /apps/training/.**
2. **Right-click on the training node and select Create… > Create Node.**

   Name: tests

   Type: cq:ClientLibraryFolder

3. **Save your changes.**
4. **Define the following two array properties on your tests folder:**

Name	Type	Value
categories	String[]	granite.testing.hobbes.tests
dependencies	String[]	granite.testing.hobbes.testrunner



5. **Save your changes.**
6. **Right-click on the tests folder and select Create… > Create file.**
7. **Name the file SampleTests.js.**
8. **Save your changes.**
9. **Using the code from the Exercise_Files, paste the code into SampleTests.js.**

   SampleTests.js

```
/*…comments*/
```

```
new hobs.TestSuite("We.Train Tests", {path:"/apps/training/tests/SampleTests.js",
register: true})


    .addTestCase(new hobs.TestCase("Hero component on en.html")
        .navigateTo("/content/we-train/en.html")
        .asserts.location("/content/we-train/en.html", true)
        .asserts.visible(".hero-image", true)
    )

    .addTestCase(new hobs.TestCase("Hero component on fr.html")
        .navigateTo("/content/we-train/fr.html")
        .asserts.location("/content/we-train/fr.html", true)
        .asserts.visible(".hero-image", true)
    )

    .addTestCase(new hobs.TestCase("Print Selector inserted on en.html")
              .navigateTo("/content/we-train/en.html")
              .asserts.location("/content/we-train/en.html", true)
              .asserts.exists("form.page__print[action='/content/we-
train/en.print']",true)
       );
```

10. **Save your changes.**
11. **Right-click on the tests folder and select Create… > Create File.**
12. **Name the file as js.txt.**
13. **Populate the code in js.txt as shown below to reference your JavaScript file:**

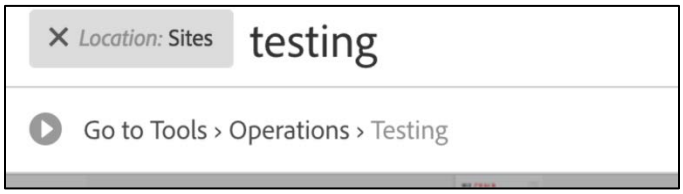
```
//Add Hobbes tests to the clientlib
SampleTests.js
```

14. **Save your changes.**

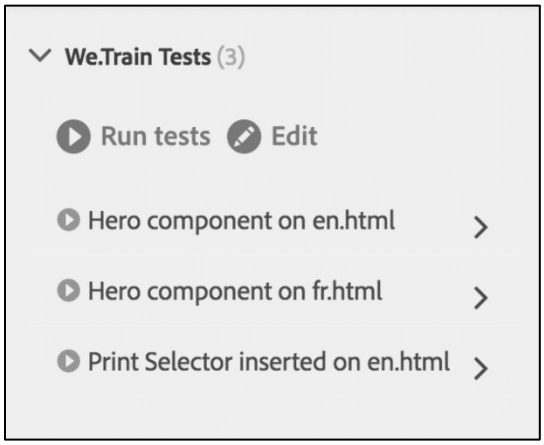
15.1.3 Task – Run the Hobbes testing suite

Think back to the previous discussion in Module 5 of Sling URL decomposition, resource resolution and how Sling finds the rendering script. Recall that when a URL contains a selector, Sling will first attempt to find a script to match that selector.

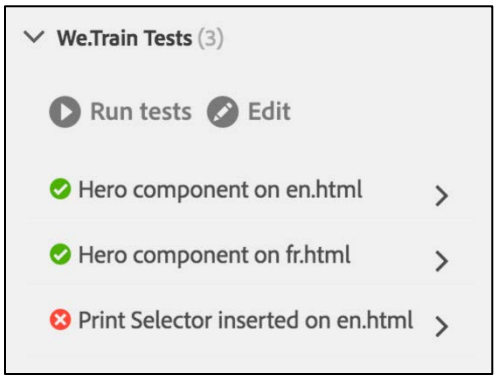
1. **Navigate to the Hobbes Testing Area (Tools > Operations > Testing) (http://localhost:4502/libs/granite/testing/hobbes.html). Or from any Console, type "/" and enter "testing" into the OmniSearch toolbar and click on Testing.**



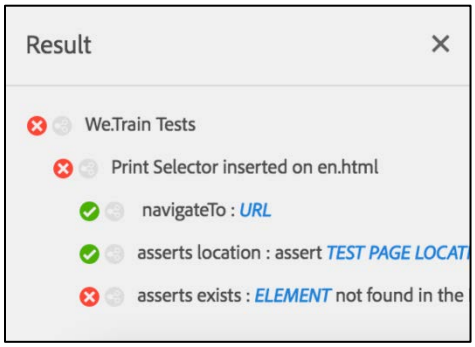
2. **Notice how the We.Train Tests now shows up in the Testing console:**



3. **Click Run tests (Run tests) to run the Hobbes testing suite. Be patient as the tests may take up to 10-15 seconds to complete. Notice how two tests pass successfully and the last one fails.**



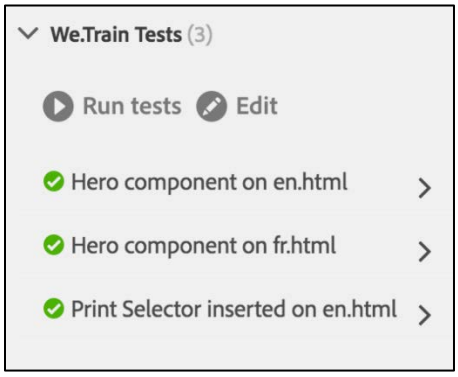
4. **Open up the failed test (Print Selector inserted on en.html). This is a test to make sure that the correct URL was programmatically inserted to the page for the print button.**



5. **If you open up the source HTML of http://localhost:4502/content/we-train/en.html, you will notice that the form action URL is correctly inserting the print selector to the URL, so our test is coded wrong.**
6. **Since our test is incorrect, we need to fix it. Click on the Edit (Edit) button to go back to CRXDE Lite and open the testing suite (/apps/training/tests/SampleTests.js).**
7. **Update the JS code to fix the test. Add .html to the last assert.exists statement and save your changes:**

```
...
.asserts.exists("form.page__print[action='/content/we-train/en.print.html']",true)
...
```

8. **Go back to the Testing console and rerun the tests to see all tests pass.**



NOTE: If your tests still fail, you may have to run another browsing tab in "Incognito Mode" in Google Chrome if you are using that browser. This may be due to a caching issue. If you are using IE, use File > New Session.

16 USING BRACKETS FOR DEVELOPMENT

Working with Brackets

Brackets is developed by Adobe, and is an open source code editor for HTML, HTL, CSS, and JavaScript. You can download the latest version of Brackets from: **http://brackets.io/**.

Features of Brackets:

- Live preview
- In-line editors
- Pre-processor support
- A collection of add-on extensions

Available Plugins for Adobe Experience Manager Developers:

- AEM Brackets Extension
- Extract for Brackets

Installing the AEM Brackets Extension

You can install the AEM Brackets Extension within Brackets using the Extension Manager. Brackets with the AEM extension is the preferred tool for front-end developers building components with HTL.

Features of AEM Brackets Extension:

- Automatic, bidirectional synchronization with the content repository
- Full content-package synchronization of the project.
- HTL syntax highlighting
- HTL code completion

For more information on this extension, refer to https://docs.adobe.com/content/docs/en/dev-tools/aem-brackets.html.

Configuring Your Project

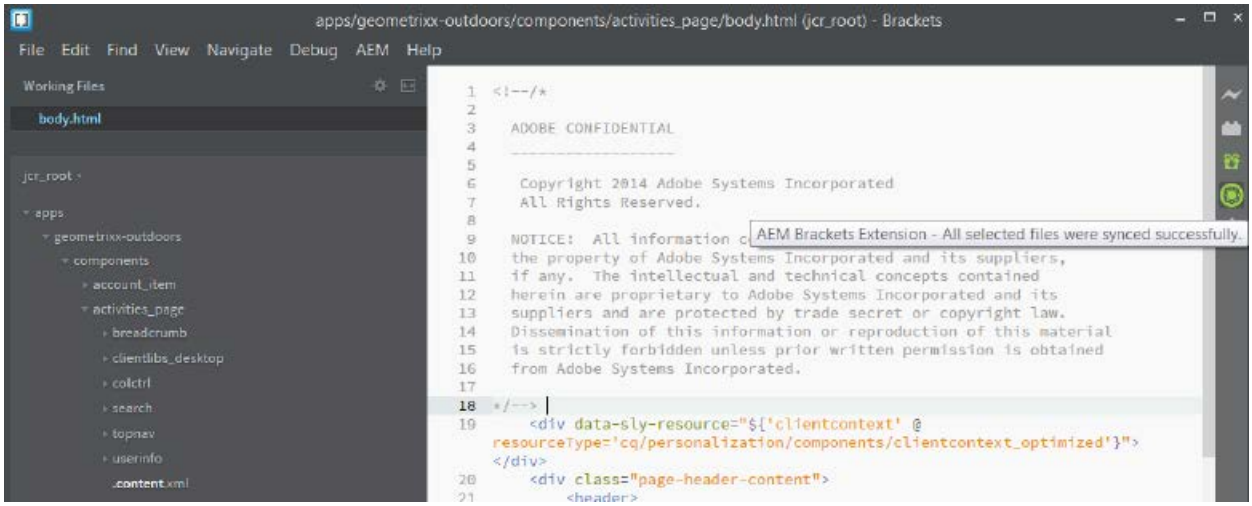
You already learned how to create a package using the Package Manager. You can import the same package to Brackets and begin working on it, or if you already have a project package, you must ensure it has the following:

- A jcr_root folder
- A filter.xml file

How to set up your Brackets project based on an existing front-end development project in your repository:

1. Generate a package of your project work in CRX Package Manager
2. Extract the contents of the package
3. Open the **jcr_root** folder of the package
4. Configure the project settings to connect to Adobe Experience Manager development author instance

Your project is now ready for development. Any time you save your changes, it will be automatically synchronized with the Adobe Experience Manager content repository. This is indicated by a green icon in the toolbar on the right pane.



Either you can use the Live Preview to review your changes, or you can do it through the Adobe Experience Manager browser instance.

Exercise

16.1.1 Task - Install Brackets and the AEM Extension

NOTE: You can install Brackets with or without an internet connection.

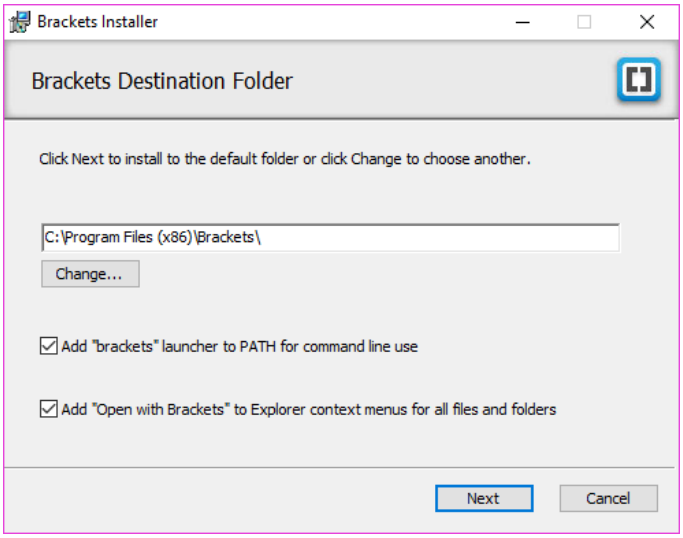
NOTE: If you are attending a VILT class using ReadyTech, the installation of Brackets has already been performed for you. You may skip ahead to step #3.

1. **If you have an Internet connection, download the latest version of Brackets from: http://brackets.io/. If you do not have network connectivity, obtain the standalone installation files from the Distribution_Files folder for this course.**

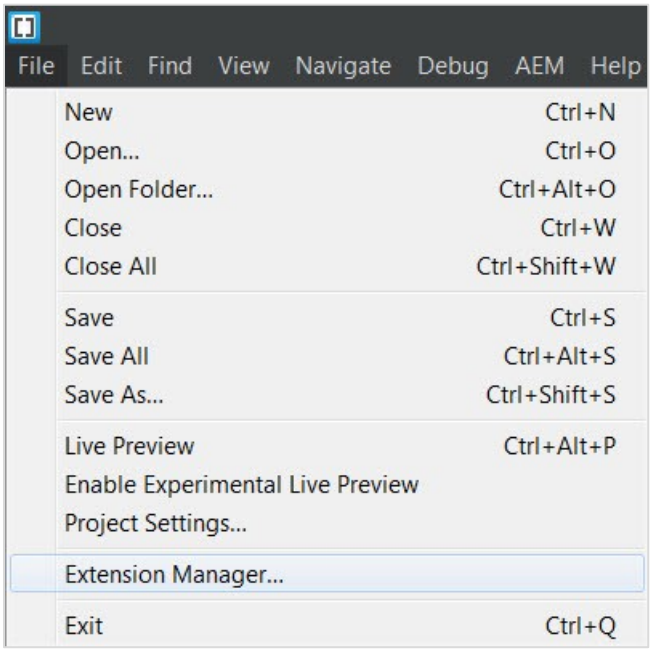
NOTE: This is different than the Exercise_Files you have been using for this course. Installers are available in the Distribution_Files for both Mac and Windows (Windows: *Brackets.Release.1.9.msi*, Mac: *Brackets.Release.1.9.dmg*).

2. **Install Brackets.**

3. **When going through the Brackets installation wizard, accept all defaults.**

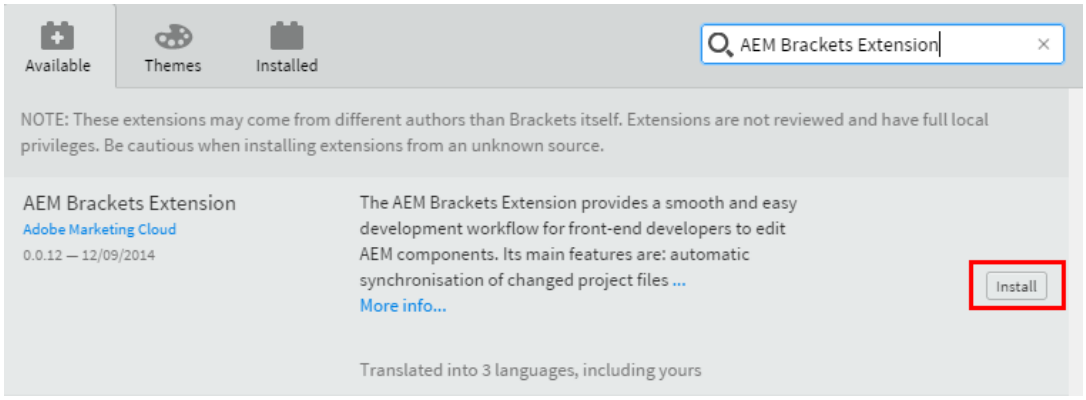


4. **Open Brackets. Click File > Extension Manager...**



5. **On the Available tab, search for AEM Brackets Extension. You can also drag the extension zip file (aem-brackets-extension-master.zip) from the Distribution_Files to the area displayed at the bottom of the pop-up window. Notice you will have to drag the file if you do not have internet connectivity.**

6. **Click Install to install the extension.**



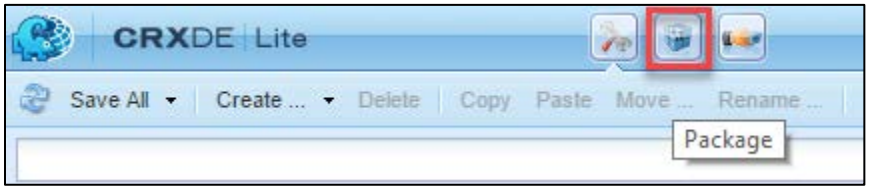
7. **After a successful installation, a success message appears on the screen.**

8. **Restart Brackets.**

16.1.2 Task – Package up your We.Train work in this course

In a previous exercise (Module 4), you learned how to work with packages by creating a simple package. In this exercise, you will follow a similar process, but will include all your work and configurations you performed in your environment to date in this course into an exportable package. Review Module 4 (Developer Tools) if you need to further understand more about packages.

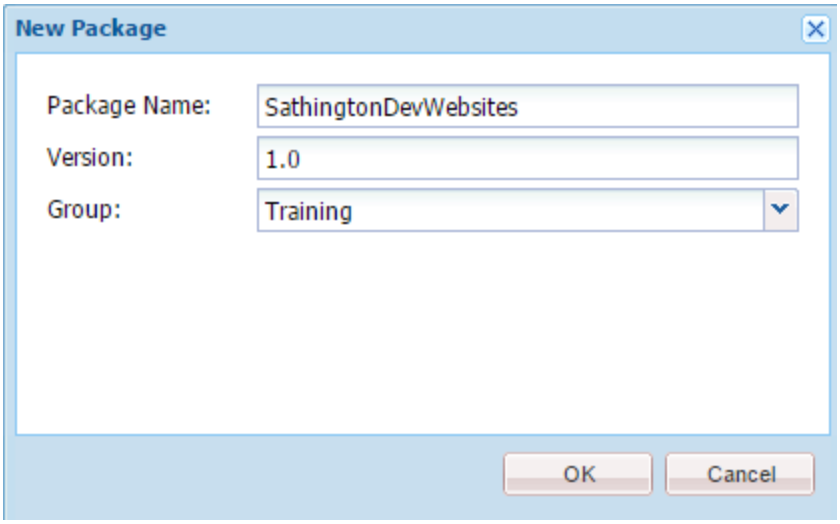
1. **In CRXDE Lite, navigate to Package Manager:**



2. **Click Create Package.**

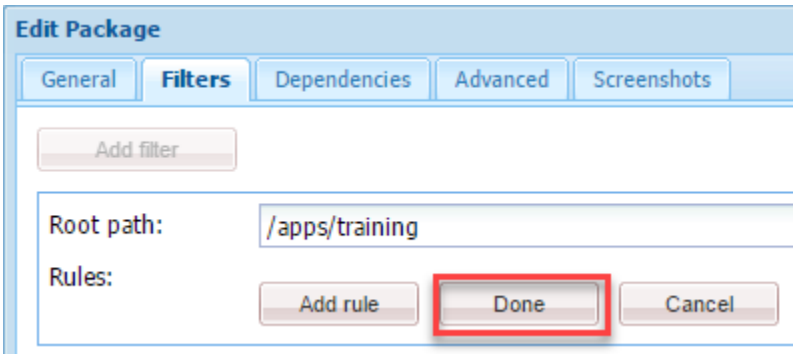
3. **In the New Package dialog box, enter the following details:**

Property	Value
Package Name	<YourName>DevWebsites
Version	1.0
Group	Training



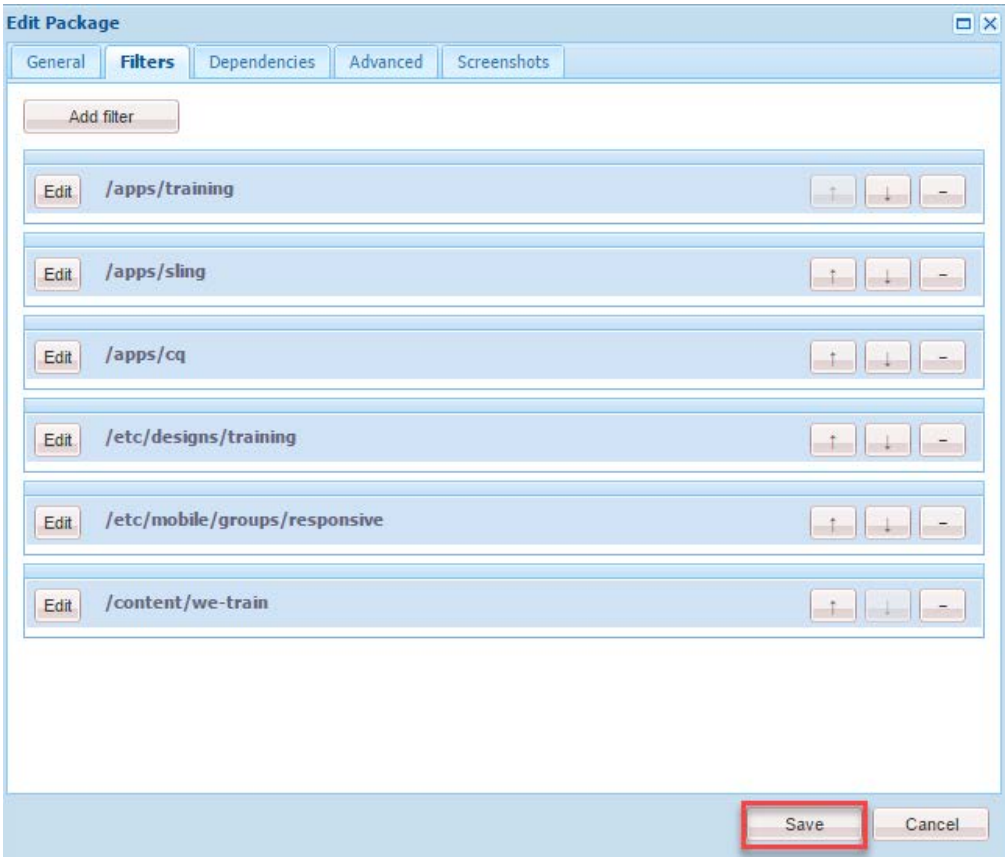
4. **Click OK.**
5. **Click Edit on the newly-created package.**
6. **To add filters to the package, click the Filters tab, and click Add Filter.**
7. **For the Root path, browse and select /apps/training project folder and click OK.**

8. **Click Done.**

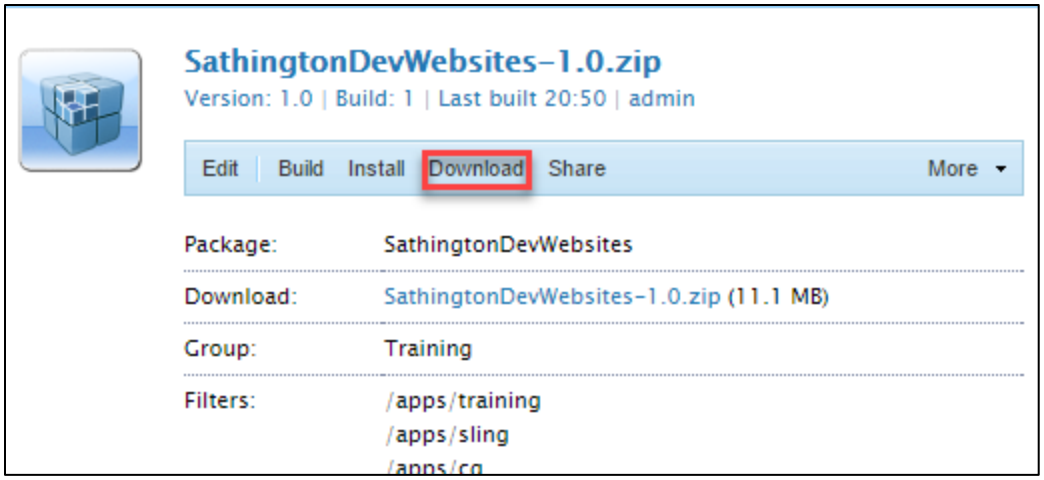


9. **Add the following additional filters (click Done after adding each new filter):**
   - **/apps/sling**
   - **/apps/cq**
   - **/etc/designs/training/**
   - **/etc/mobile/groups/responsive**
   - **/content/we-train**

10. **Click Save:**



11. **Click Build to build the package, and then click Build again in the confirmation dialog box. The package is now ready for download.**
12. **Click the Download link to download a copy of the package to your computer.**



NOTE: If you are using a ReadyTech environment, you may upload your package for use after class to a file-sharing tool such as the Adobe Document Cloud (https://cloud.acrobat.com). Use your Adobe ID to access the Document Cloud. If you do not have an Adobe ID, you may register for one for free.

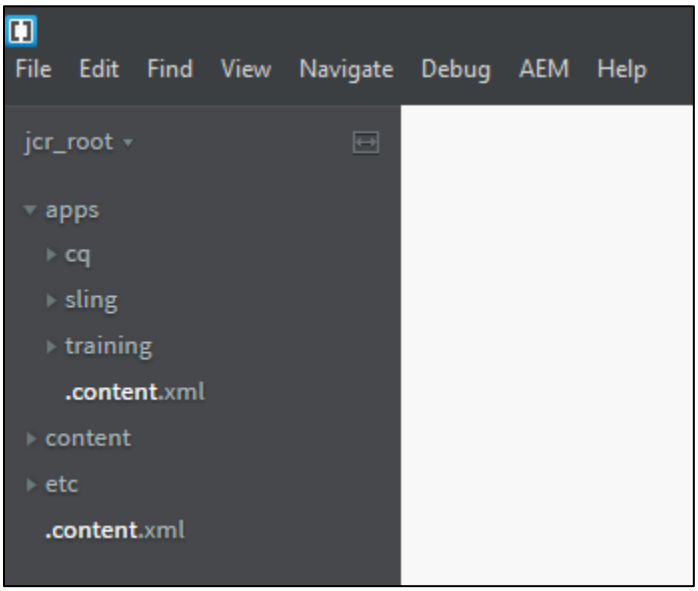
Alternatively, you can place the package file in c:\ReadyTech\Outbox and your instructor can retrieve them and email them to you at the end of class.

16.1.3 Task – Make changes to the repository using Brackets

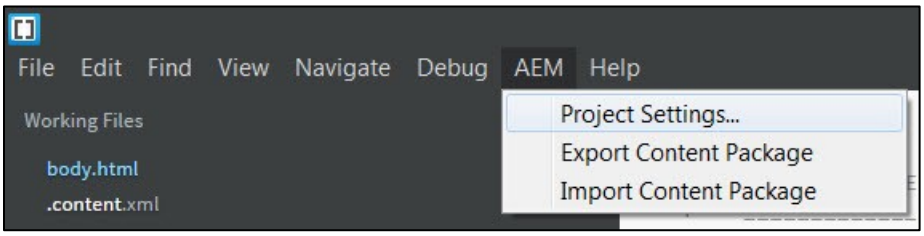
NOTE: The goal of this task is to edit the contents of your project in Brackets. After completing this task, you can update the package content and share it with team members so they also see the same content in their Adobe Experience Manager instances.

You already created and downloaded a package containing your work in this course in the previous task.
You will import this package into Brackets in order to see how your repository can be configured in Brackets and then see the changes propogated to the repository in one of your We.Train pages.

1. **Navigate to the downloaded zipped package you created and downloaded to your computer. Extract the file to a local directory of your choosing.**
2. **Open Brackets.**
3. **Navigate to File > Open Folder...**
4. **Locate the jcr_root folder of the unzipped (extracted) package, and click Select Folder (or click Open on a Mac computer). The contents of the folder are listed in the left panel. Expand the apps folder:**



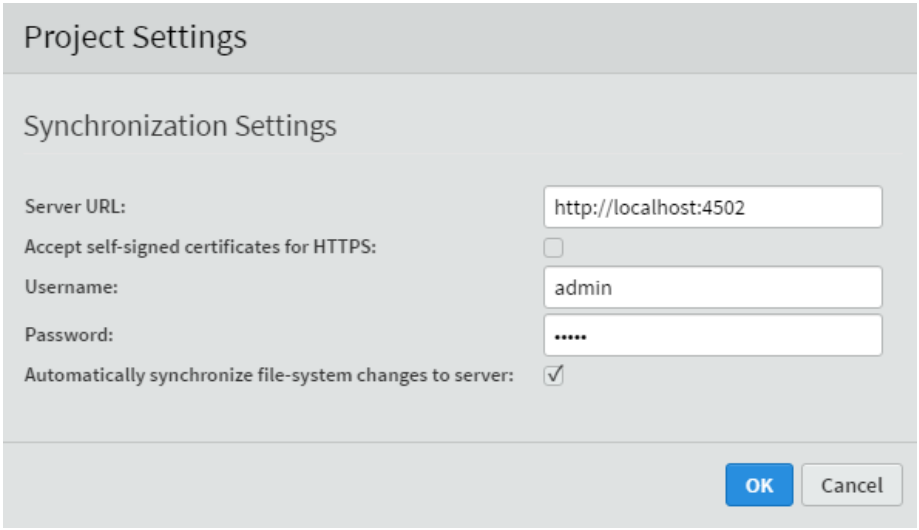
5. **Configure the Project Settings. On the menu bar, click** AEM**, and select** Project Settings...



6. **In the** Project Settings **dialog box, enter the following details, and click** OK.

NOTE: These settings may already be populated; if so, confirm they are correct.

Label	Value
ServerURL	http://localhost:4502
Username	Admin
Password	Admin



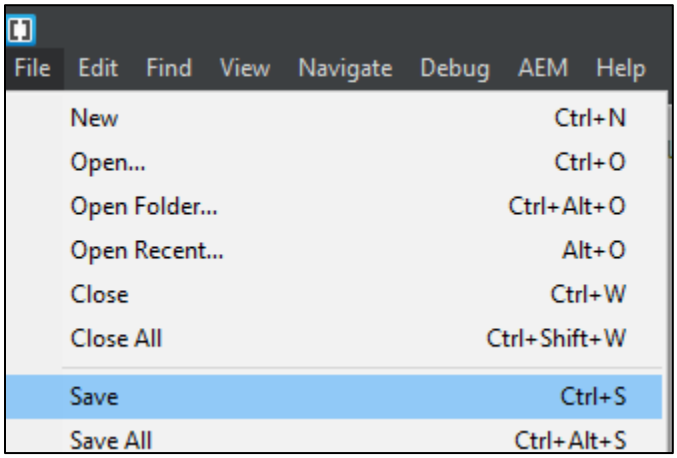
NOTE: The "Automatically synchronize file-system changes to server" option should be already selected.

These details are provided to Brackets, so it can interact with the Adobe Experience Manager author development instance. In a real-life scenario, your URL and credentials might differ.

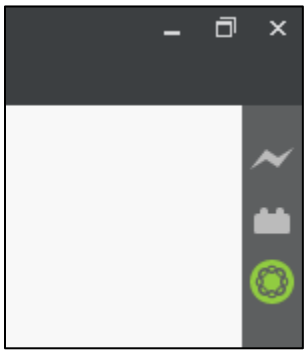
7. **In the left panel, expand the apps node and navigate to** /apps/training/components/structure/title/title.html.
8. **When you select** title.html**, the file opens in the right panel. On line 2, add a line of simple HTML like:**

```
<h2>Hello, I'm from Brackets!</h2>
```

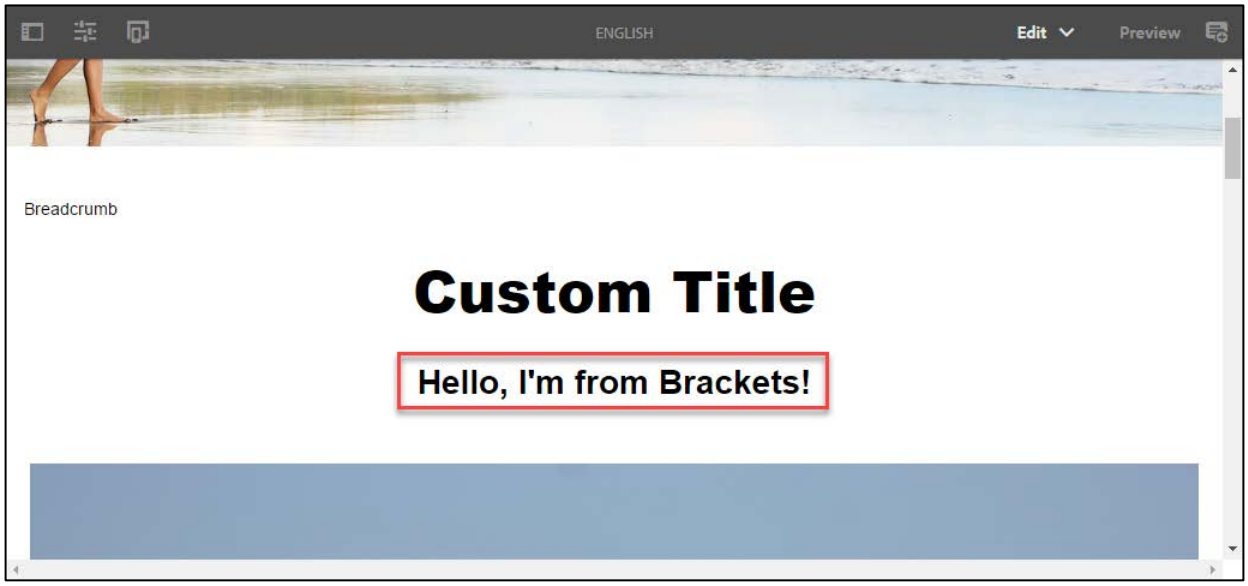
9. **Save your changes. Use CTRL+S (Windows) or File > Save.**



10. **This automatically synchronizes the file with Adobe Experience Manager. Notice how the AEM extension tool turns green in the upper-right corner:**



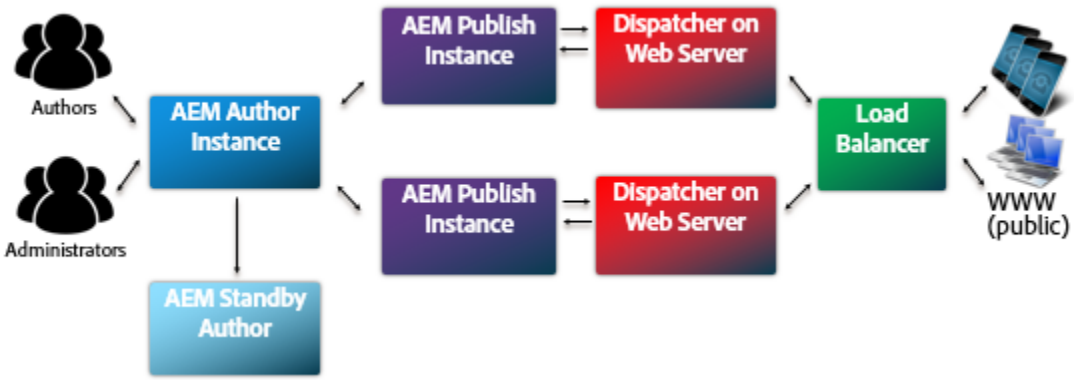
11. **To test the synchronization, open the English page in We.Train. Your HTML should appear in your title component:**



17 AEM ENVIRONMENT

Review: AEM Instances

In Module 1, we explored how there are two instances of AEM – Author and Publish. This is how AEM works *typically* in production. Author and Publish instances share same the code base, different 'run mode'.



In Production, several instances are involved. Because this is a production scenario (above) it is important to note that the users involved are not developers. Once AEM is in production, you would not need development resources (in most cases). In production, all development and code is finalized. Authors are the primary users of the production environment and administrators maintain and update the production environment.

In Adobe Experience Manager terminology, an "instance" is a copy of Adobe Experience Manager running on a server. Adobe Experience Manager installations usually involve multiple instances, typically running on separate machines:

- Author: An Adobe Experience Manager instance used to create, upload, and edit content, and to administer the website. After content is ready to go live, it is replicated to the Publish Instance.
- Publish: An Adobe Experience Manager instance that serves the published content to the public. These instances are identical in terms of installed software. They are differentiated only by the configuration.

In addition, most installations use Dispatchers and Load Balancers to handle performance:

- Dispatchers are static web servers (such as Apache httpd, Microsoft IIS, and so on) augmented with the Adobe Experience Manager Dispatcher module. Dispatchers cache webpages produced by the Publish Instance to improve performance.

AEM Standby is what is known as AEM cold standby, a feature of AEM. It replicates all content from the primary author instance and then if the primary ever goes down, the cold standby instance (secondary) can be rebooted as the primary.

More information on Cold Standby: https://docs.adobe.com/docs/en/aem/6-3/deploy/recommended-deploys/tarmk-cold-standby.html

Performance Considerations

A key issue is the time your website takes to respond to visitor requests. Although this value will vary for each request, an average target value can be defined. When this value is proven achievable and maintainable, it can be used by authors, who add and update the content, use this instance, monitor the performance of the website, and indicate the development of potential problems.

The response times you will be aiming for will be different on the author and publish instances, reflecting the different characteristics of the target audience.

It is important to know the operating norms of your environment. Otherwise, it can be difficult to locate the problem when performance deteriorates. Spend some time investigating your system when it is running smoothly. Ensure that collecting performance information is an ongoing task.

- Performance tuning *must* be part of every project.
- Do not optimize early in the development cycle.
- Performance is only as good as the weakest link.
- Always think about capacity vs. volume.
- Optimize important things first.
- Never optimize without *realistic* goals.

Author Instance

Authors who add and update the content use this instance. It must cater for a small number of users— who generate a high number of performance-intensive requests—when updating content pages and individual elements on those pages.

Publish Instance

This instance contains content that you make publicly available on the web, where the number of requests is even greater and the speed is just as vital. Because the nature of the requests is less dynamic, additional performance-enhancing mechanisms can be leveraged, such as the caching content and/or using load balancing.

Performance Optimization Methodology

A performance optimization methodology for AEM projects can be summarized into five simple rules to avoid performance issues from the start. These rules, to a large degree, apply to web projects in general, and are relevant to project managers and system administrators to ensure that their projects will not face performance challenges when launch time comes.

1. Plan for Optimization

Around 10% of the project effort should be planned for the performance optimization phase. Of course, the actual performance optimization requirements will depend on a project's level of complexity and the experience of the development team. While your project may ultimately not require all the allocated time, it is a good practice to always
plan for performance optimization in that suggested range.

Whenever possible, a project should first be soft-launched to a limited audience to gather real-life experience and perform further optimizations, without the additional pressure that follows a full announcement. When you are live, performance optimization is not over. This is the point in time when you experience the real load on your system. It is important to plan for additional adjustments after the launch.

Because your system load changes and the performance profiles of your system shifts over time, a performance tune-up or 'health-check' should be scheduled at 6–12 month intervals.

2. Simulate Reality

If you go live with a website and find out after the launch that you run into performance issues, there is only one reason for that—your load and performance tests did not simulate reality close enough. Simulating reality is difficult and how much effort you will reasonably want to invest into getting real depends on the nature of your project. Real means not just real code and real traffic, but also real content, especially regarding content size and structure. Keep in mind that your templates may behave completely different depending on the size and structure of the repository.

3. Establish Solid, Realistic Goals

The importance of properly establishing performance goals is not to be underestimated. Often, when people are focused on specific performance goals, it is hard to change these goals later, even if they are based on wild assumptions.
Establishing good, solid performance goals is one of the trickiest areas. It is often best to collect real-life logs and benchmarks from a comparable website (for example, the new website's predecessor).

4. Stay Relevant

It is important to optimize one bottleneck at a time. If you do things in parallel without validating the impact of one optimization, you will lose track of which optimization measure actually helped.

5. Iterate

Performance tuning is an iterative process that involves measuring, analysis, optimization, and validation until the goal is reached. To consider this aspect properly, implement an agile validation process in the optimization phase rather than a more heavyweight testing process after each iteration. This largely means that the developer implementing the optimization should have a quick way to tell if the optimization has already reached the goal, which is valuable
information, because when the goal is reached, optimization is over.

Basic Performance Guidelines

Generally, keep your uncached html requests to less than 100ms. More specifically, the following may serve as a guideline:

- 70% of the requests for pages should be responded to in less than 100ms.
- 25% of the requests for pages should get a response within 100ms–300ms.
- 4% of the requests for pages should get a response within 300ms–500ms.
- 1% of the requests for pages should get a response within 500ms–1,000ms.
- No pages should respond slower than 1 second.

The above numbers assume the following conditions:

- Measured on publish (no authoring environment and/or CFC overhead)
- Measured on the server (no network overhead)
- Not cached (no AEM-output cache, no Dispatcher cache)
- Only for complex items with many dependencies (HTML, JS, PDF, and so on)
- No other load on the system

Certain issues frequently contribute to performance issues, which mainly revolve around (a) dispatcher caching inefficiency and (b) the use of queries in normal display templates. JVM- and OS-level tuning usually do not lead to big leaps in performance and should therefore be performed at the tail end of the optimization cycle. Your best friends during a usual performance optimization exercise are the request.log, component-based timing, and lastly, a Java profiler.

Exercise

17.1.1 Task – Monitor Page Response

1. **Navigate to and open the file request.log located at <aem-install-dir>/crx-quickstart/logs. For a ReadyTech environment, this is located at C:/adobe/AEM/author/crx-quickstart/logs/request.log.**
2. **Request a page that utilizes your Training Template and components. For example, /content/we-train/en/products.**
3. **In the log file, review the response times directly related to the previous step's request. (A page request of the page: /content/we-train/en/products). Your log should look like this:**

```
150  20/Apr/2017:07:38:21 -0700 [1642] -> GET
     /mnt/override/libs/cq/activitymap/content/settings.json?path=/content/we-train/en/products&_=1492699100037 HTTP/1.1
151  20/Apr/2017:07:38:21 -0700 [1640] <- 200 application/json 11ms
152  20/Apr/2017:07:38:21 -0700 [1642] <- 200 application/json 2ms
153  20/Apr/2017:07:38:21 -0700 [1643] -> GET /resource-status/editor/content/we-train/en/products.1.json?_=1492699100038 HTTP/1.1
154  20/Apr/2017:07:38:21 -0700 [1643] <- 200 application/json 3ms
```

4. **You successfully reviewed the response time of a page using request.log. Again, this will aid your development in being able to monitor response times of pages that implement custom templates and components, and comparing said time to your project goals.**

17.1.2 Task – Examine Page Request Performance

1. **Go to Tools > Operations > Diagnosis > Request Performance. The Request Performance page appears. The page displays requests in the order of duration taken. Notice you may click to sort on each column (Request Time, Resource/Method, Duration).**
2. **Which page requests took the longest? The shortest?**

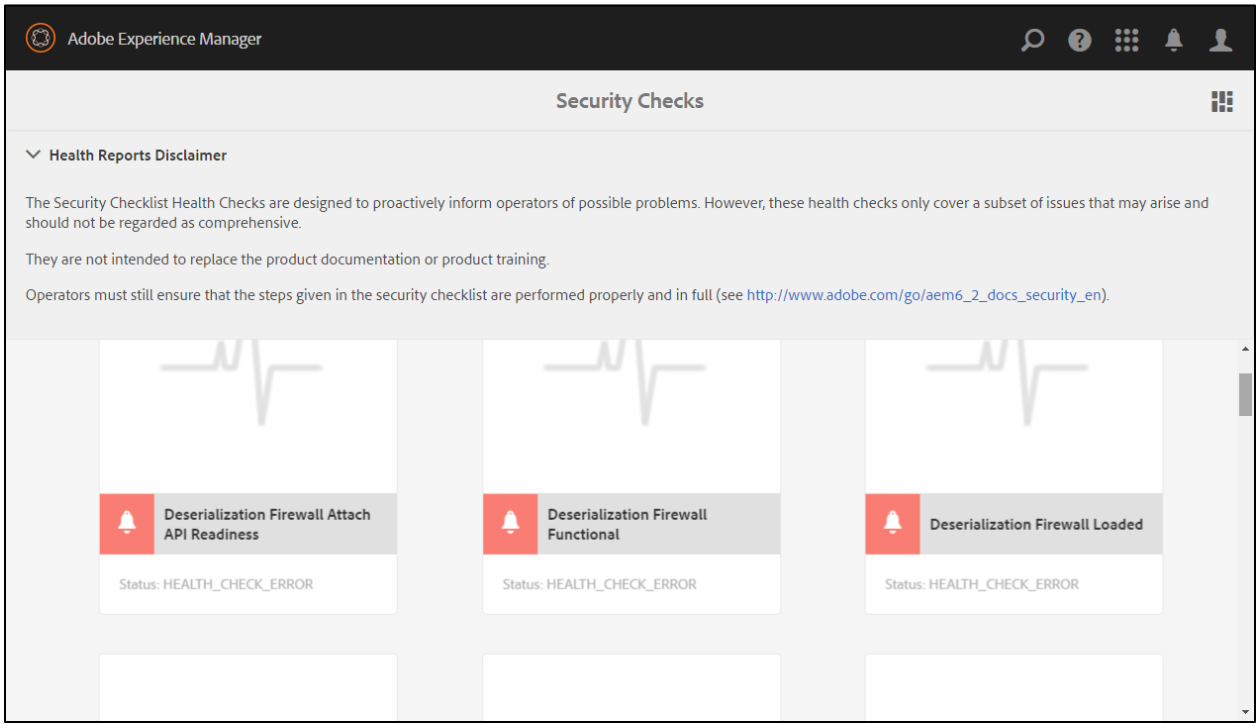
You can use this information to further investigate the cause of long responses.

You successfully found and displayed long-lasting requests/responses in AEM. Again, this is just one of many tools to help you meet your project's performance goals.

Performing Security Checks

One of the most important health reports available in the dashboard is the Security Checks. It is recommended that you check the status of all the security health checks before going live with your production instance.

Tools > Operations > Health Reports > Security Checks

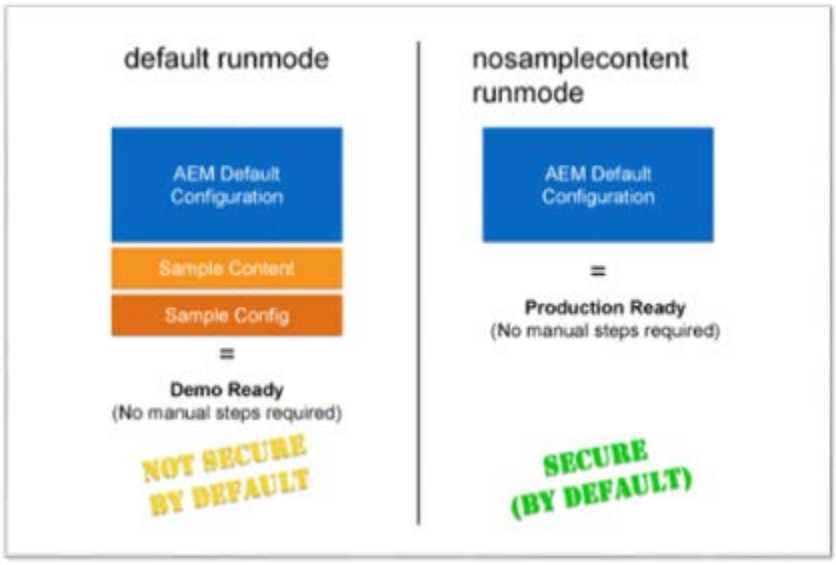


The following are a few of the checks that are essential for a secure instance:

- **Authorizable Node Name Generation** – Checks whether the AuthorizableNodeName implementation exposes the authorizable ID.
- **CRXDE Support** – Fails if the CRXDE Support bundle is active.
- **Default Login Accounts** - Fails if the default login accounts have not been disabled.
- **Sling Get Servlet** - Fails if the default Sling Get Servlet configuration is not following the security guidelines.
- **CQ Dispatcher Configuration** - Checks the basic configuration of the Dispatcher component.
- **Example Content Packages**—Fails if the sample content packages are found.
- **CQ HTML Library Manager Config**—Checks if the default CQ HTML Library Manager configuration follows the security guidelines.
- **Replication and Transport Users**—Checks the replication and transport users.
- **Sling Java Script Handler**—Checks if the Sling Java Script Handler configuration follows the security guidelines.
- **Sling JSP Script Handler**—Checks if the Sling JSP Script Handler configuration follows the security guidelines.
- **Sling Referrer Filter**—Checks if the Sling Referrer Filter is configured in order to prevent CSRF attacks.
- **User Profile Default Access**—Checks if everyone has read access to user profiles.
- **WCM Filter Config**—Checks if the default WCM Filter configuration follows the security guidelines.
- **WebDav Access**—Checks if the WebDav Access bundle is active.

- **Web Server Configuration**—Checks if the web server sends the X-FRAME-OPTIONS HTTP header set to SAMEORIGIN. In order for it to function, this Health Check needs to be configured with the public server address that the dispatcher is configured with.

AEM in Production-Ready Mode



Generally, the "sample content" consists of the We.Retail reference site that comes with AEM. This site can be used for demos and is a fully functional, demo-ready site. The We.Retail reference site is covered in more detail in Module 2 (Installation).

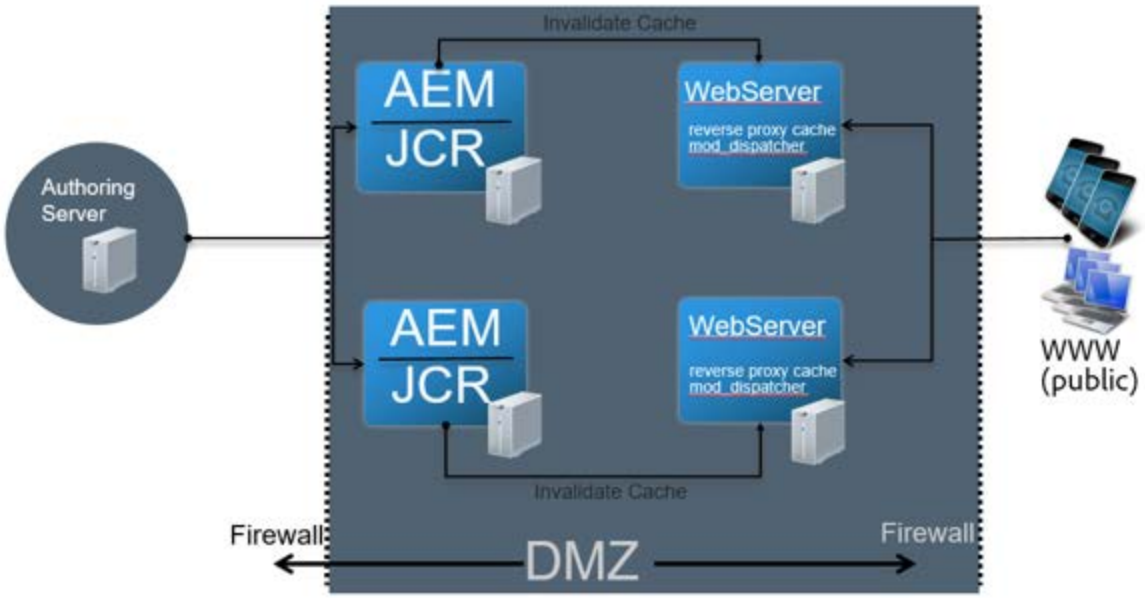
Security Checklist Review

1. Run Adobe Experience Manager in Production Ready Mode
2. Enable HTTPS for transport layer security
3. Install Security Hotfixes
4. Change Default Passwords For the AEM and OSGi Console Admin Accounts.
5. Change the Adobe Experience Manager admin user password
6. Change the OSGi web console admin password
7. Implement Custom Error Handler
8. Complete Dispatcher Security Checklist
9. Verification Steps
    a. Configure replication and transport users
    b. Check the Operations Dashboard Security Health Checks
    c. Check if Example Content is Present
    d. Check if the CRX development bundles are present
    e. Check if the Sling development bundle is present
10. Protect against Cross-Site Request Forgery
11. OSGi Settings
12. Mitigate Denial of Service (DoS) Attacks
13. Disable WebDAV

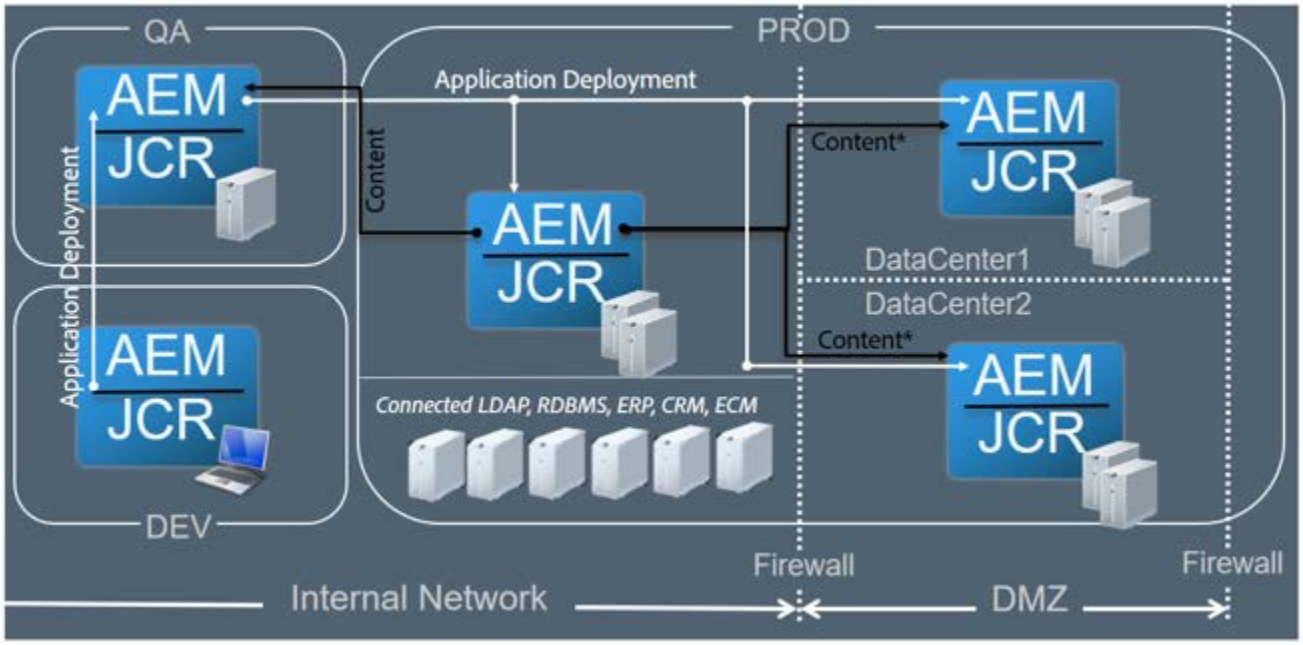
More info:

https://docs.adobe.com/docs/en/aem/6-3/administer/security/security-checklist.html

DMZ Detail



Example Physical Layout/Deployment

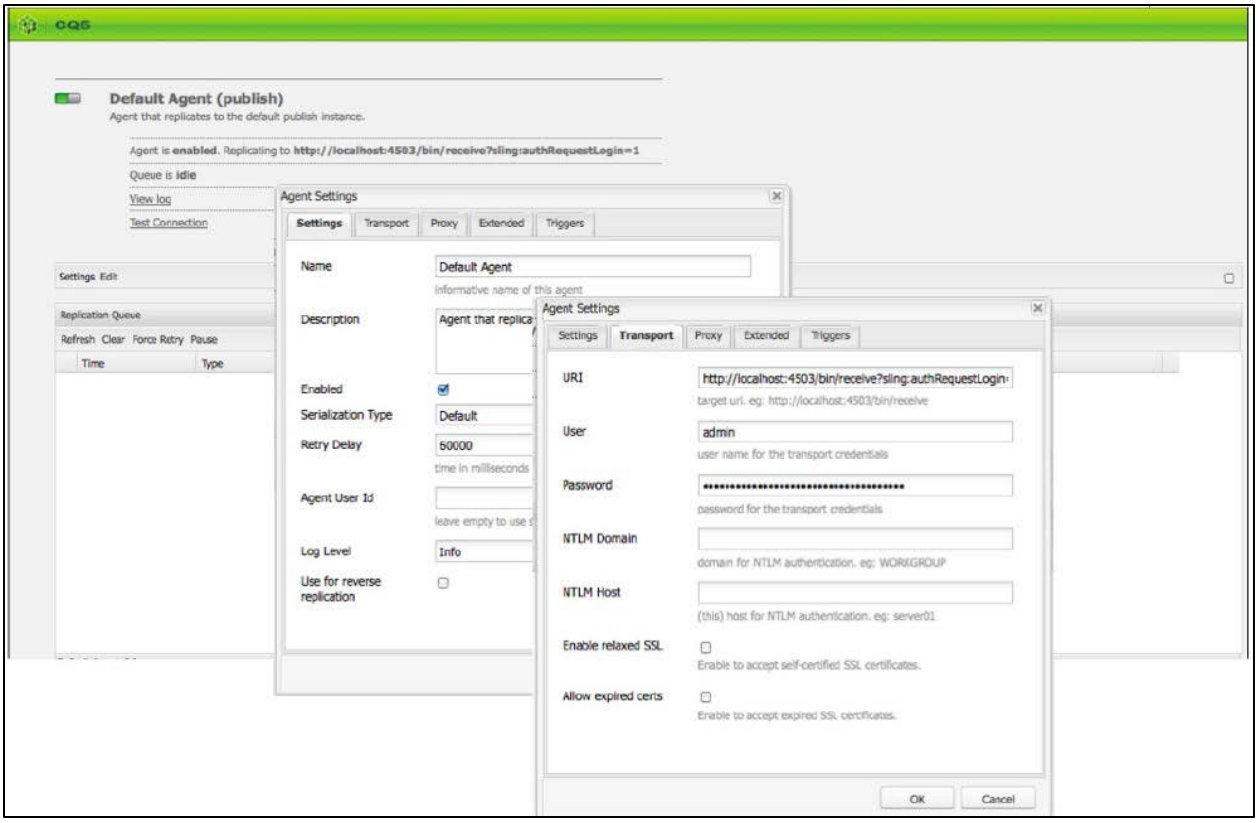


Replication Agents

Content and code often move in different directions and get to their destination through different mechanisms. Content typically is moved by use of Replication Agents. Content will move forward from Author to Publish, but also may move back to QA. This allows testing to be done on real data.

Deployment of code and configuration information is typically done through automated processes that use content packages, Replication Agents, and/or FileVault.

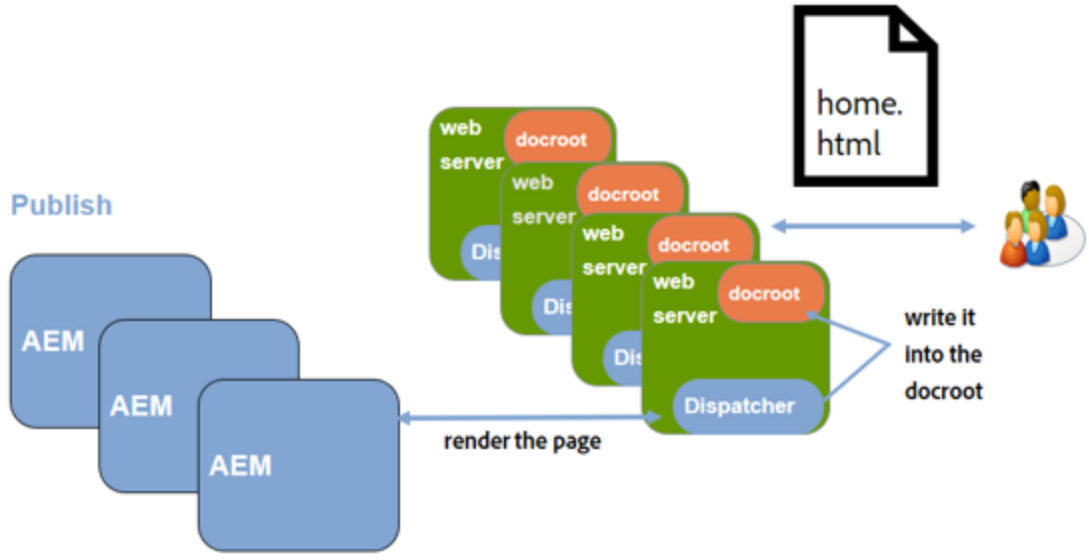
Activation or publication of content is handled by the Replication Agents. Each Replication Agent replicates content from the current instance to a destination. So, if you have four publish instances, you need four Replication Agents.



Details on the creation and management of Replication Agents can be found in the documentation at:
https://docs.adobe.com/docs/en/aem/6-2/deploy/configuring.html

Dispatcher

In the production delivery environment, the publish instance is joined by a web server module, called the Dispatcher. The Dispatcher is the AEM caching and/or load balancing tool.



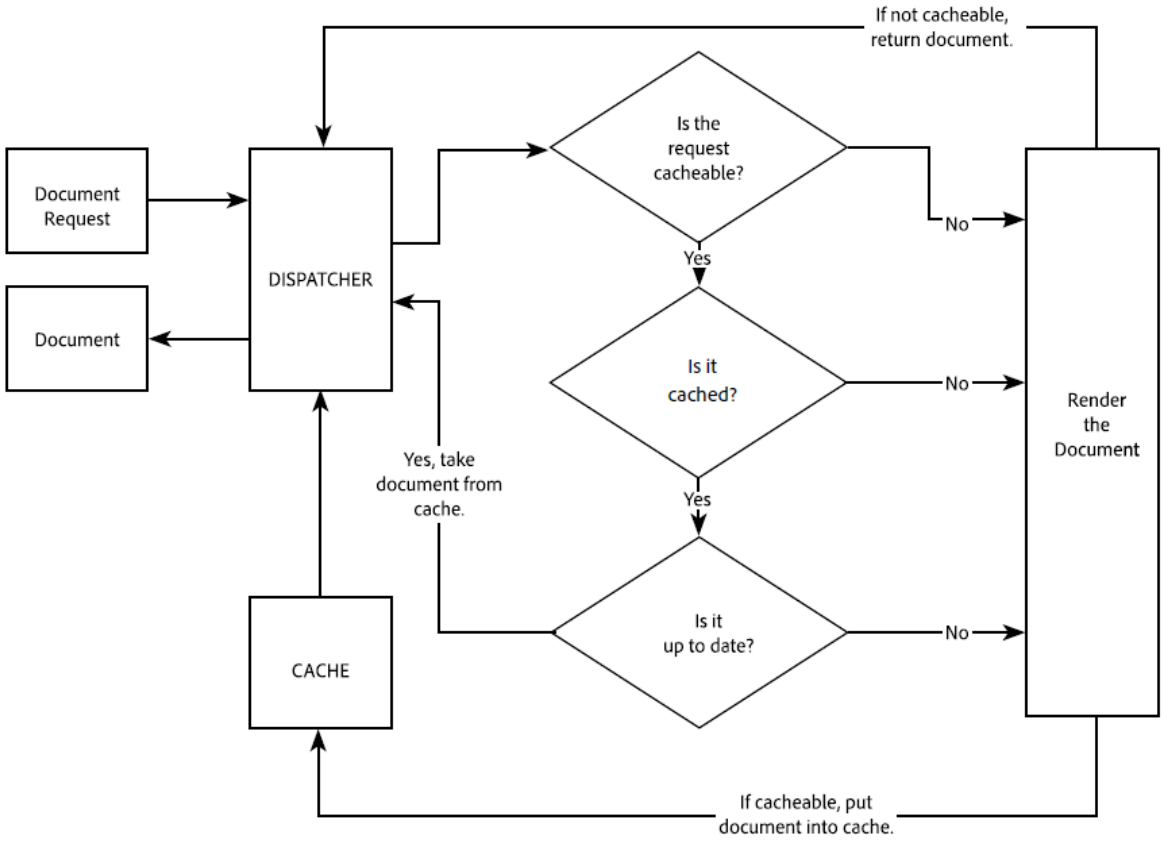
The Dispatcher helps realize an environment that is fast and dynamic. It works as part of a static HTML server, such as Apache, with the aim of:

- storing (or caching) as much of the site content as possible, in the form of a static website
- accessing the layout engine as little as possible

This means that:

- static content is handled with exactly the same speed and ease as on a static web server. Additionally, you can use the administration and security tools available for your static web server(s).
- dynamic content is generated as needed, without slowing the system any more than absolutely necessary

The Dispatcher contains mechanisms to generate and update static HTML, based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically. The figure below demonstrates the algorithm that the Dispatcher uses to determine whether an object should be served from the web server cache or rendered dynamically.



Details regarding the management and configuration of the Dispatcher can be found at: http://docs.adobe.com/docs/en/dispatcher.html

APPENDIX

Classic UI Dialog Boxes

Review: Dialog boxes are small temporary windows that are used to enable the user to provide additional inputs. These inputs are either used to perform some action, or are stored in the repository.

A dialog box in Adobe Experience Manager is similar to other dialog boxes that you have used or created in the past—it gathers user input using a 'form', potentially validates it, and then makes that input available for further use (storage, configuration, and so on).

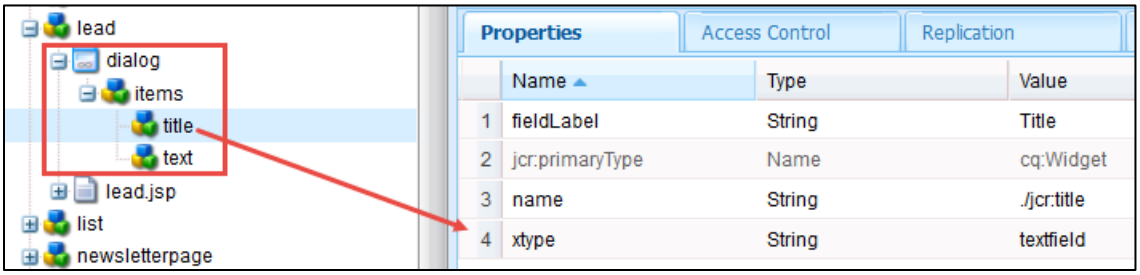
This Appendix discusses the Classic UI dialog boxes only. For a discussion of Touch UI Dialog Boxes, which are used in this course, consult Module 10.

Classic UI Dialog Box

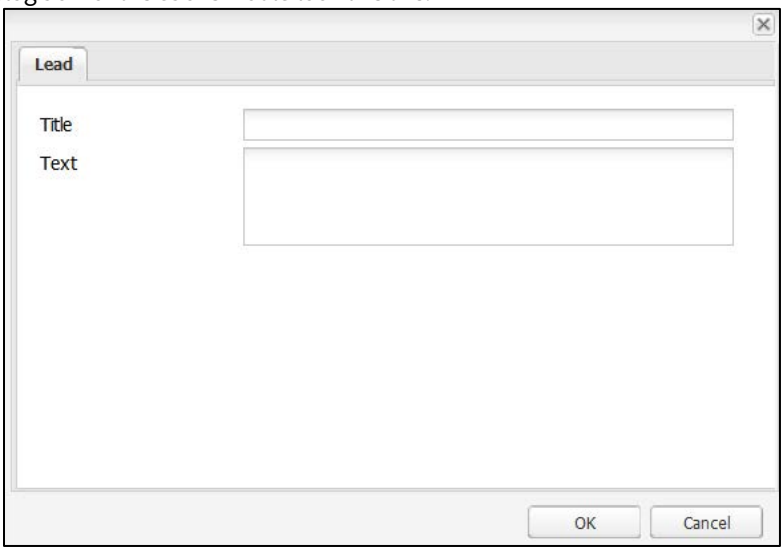
This is the default dialog box that is used in the absense of a touch-optimimzed (Touch UI) dialog box. The Classic UI uses a widget library called ExtJS. ExtJS is a cross-browser JavaScript library for building interactive web applications. It provides UI elements that work across all the most important browsers and allow the creation of desktop-grade UI experiences. The popular ExtJS JavaScript framework gives developers the ability to easily create Rich Internet Applications (RIA) using Ajax. It includes:

- High-performance, customizable UI widgets
- A well-designed and extensible component model
- An intuitive, easy-to-use API

For example, a simple classic dialog box would have the following structure and properties:



The dialog box for the above would look like this:



Difference Between Granite UI (Touch UI) and Classic UI Dialog Boxes:

Granite UI (Touch-optimized UI)	Classic UI
Defined as nt:unstructured node with resourceType property	Defined as cq:Dialog node with xtype property
Node structure describes actual fields of the dialog box	Contains node structure defining Widgets (cq:widgets)
Dialog boxes are rendered on the server-side (as Sling components), based on their content structure	Dialog boxes are rendered on the client-side out of a JSON object coming from the server
To customize a component, you only need to know how to develop components	To customize, you need to know and use the ExtJS code style
NOTE: By default, if a component has no touch-optimized dialog box defined, then the classic UI dialog box is used.	

A quick glance at the following table summarizes the two UIs.

	Touch-optimized UI	Classic UI
Framework	Granite UI	EXTJS
Rendering	resource types (server-side)	xtypes (client-side)
Dialog node	cq:dialog	dialog
Terminology	dialog / fields	dialog / widgets