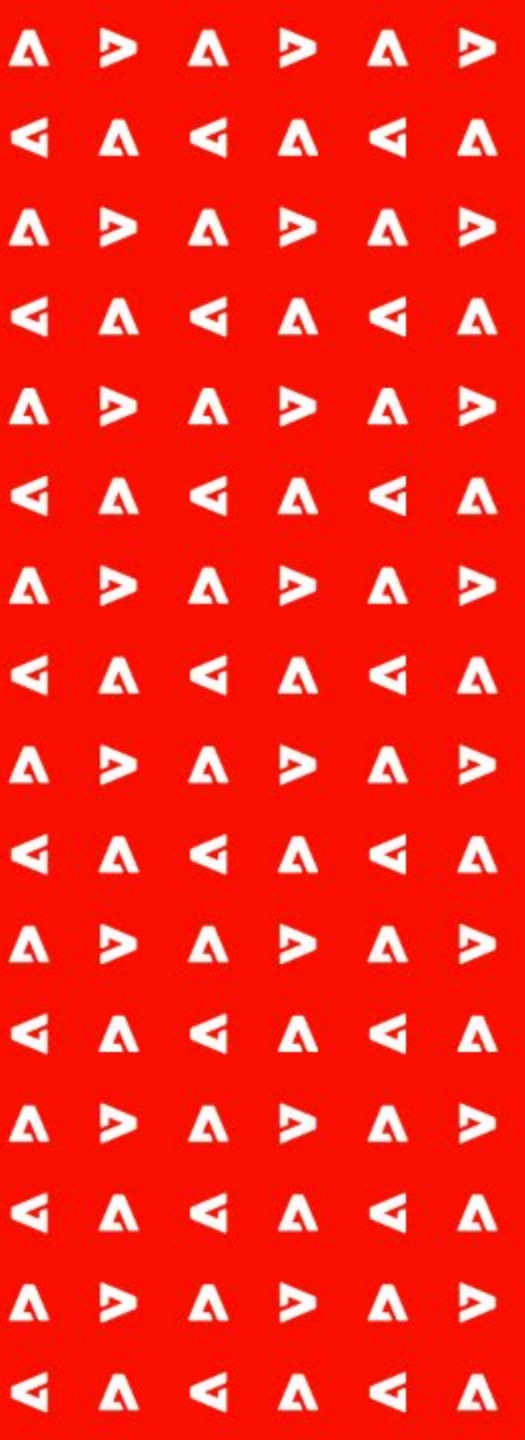
Adobe Experience Cloud

Develop SPA with React in AEM



Adobe

Legal and Confidentiality Information

Every effort was made to ensure that the information in this document was complete and accurate at the time of printing. However, information is subject to change and Adobe cannot assume responsibility for any errors or omissions. Changes or corrections to the information contained in this document may be incorporated in future issues.

This copyrighted guide along with other materials for this course may not be recorded, reproduced, duplicated, reverse engineered or adapted.

Training is intended only for those individuals who have registered and paid for it; training access URLs, user names and passwords, or access of any kind may not be shared.

This document and other materials for this course are Adobe confidential and proprietary information which must be retained in confidence by the recipient.

Introducing Your Adobe Digital Learning Services Instructor

Instructor Name
Job Title, Adobe Digital Learning Services

- Interesting fact
- Your experience with Adobe Marketing Cloud
- Something awesome about you

Student Introductions

Please introduce yourself:

- Name
- Role
- Company
- Location
- Experience with the product
- Goals for this course

Course Checklist

- Save course materials on your computer:
  - PowerPoint Slide Guide (PDF)
  - Exercise Guide (PDF)
- Use two monitors, if possible
- Enable browser pop-ups
- Google Chrome browser is recommended

Supported browsers:

Browser	Supported Version
Apple Safari	Latest version
Google Chrome	Latest version
Microsoft Edge	Latest version
Mozilla Firefox	Latest version

Course Agenda

Module 1: Introduction to AEM SPA Editor
Module 2: SPA Editor Project
Module 3: Integrate a SPA
Module 4: Map SPA Components
Module 5: Navigation and Routing
Module 6: Create a Custom Component
Module 7: Extend a Core Component

What You'll Learn

After completing this training, you should be able to:

- Install the AEM SPA Editor JS SDK and implement React components for both image and text components.
- Modify JS and CSS files and see those changes reflected in real time in the browser.
- Develop SPA Model Routing to allow navigation between different views of the app.
- Query Content Fragments from a SPA App using GraphQL

Module 1

Introduction to AEM SPA Editor

Prerequisites and Necessary Skills

For successful completion of this short introduction, the following skills are recommended:

- Maven or other build tool
- Eclipse, IntelliJ or other Java IDE
- Node.js and npm

Course Structure

Goals

- Implement a Single Page Application (SPA) in React that can be edited in Adobe Experience Manager (AEM) with the AEM SPA Editor.
- After completing this course a developer will understand the end to end creation of the SPA and the integration with AEM.

Based on WKND Project

- WKND is a fictional online magazine and blog that focuses on nightlife, activities, and events in several international cities.

Course Resources

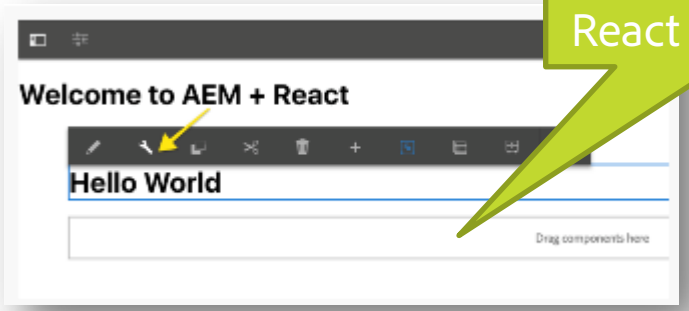
- Hands-on instructions and content provided by Instructor

- Source Code for each Module also available on Github (future updates can be obtained here):
  - https://docs.adobe.com/content/help/en/experience-manager-learn/spa-react-tutorial/overview.html

- Hands-on instructions for Angular that follow the same Topic Flow can be found at:
  - https://docs.adobe.com/content/help/en/experience-manager-learn/spa-angular-tutorial/overview.html

Topics covered in this course

Mapping AEM components to React components



SPA Routing

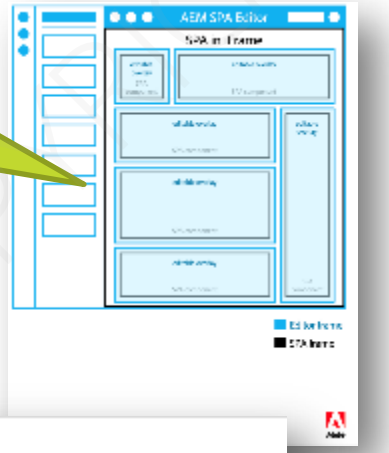
```
// psuedo code in JSX to represent the Routing aproach

<BrowserRouter>
    <App>
        <Route path="/content/wknd-events/react/home">
            <WkndPage cqPath="/content/wknd-events/react/home" />
        </Route>
        <Route path="/content/wknd-events/react/home/first-article">
            <WkndPage cqPath="/content/wknd-events/react/home/first-article" />
        </Route>
        <Route path="/content/wknd-events/react/home/second-article">
            <WkndPage cqPath="/content/wknd-events/react/home/second-article" />
        </Route>
    </App>
</BrowserRouter>
```

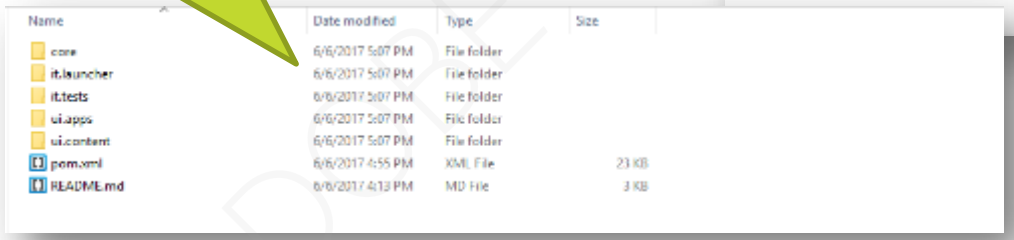
Integrating the SPA app into an AEM Page

```
<!--/*
Custom Headerlibs for React Site
*/-->
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<meta property="cq:datatype" data-sly-test="${wcmmode.edit || wcmmode.preview}" content="JSON"/>
<meta property="cq:wcmmode" data-sly-test="${wcmmode.edit}" content="edit"/>
<meta property="cq:wcmmode" data-sly-test="${wcmmode.preview}" content="preview"/>
<meta property="cq:pagemodel_root_url"
    data-sly-use.page="com.adobe.aem.guides.wkndevents.core.models.HierarchyPage"
    content="${page.rootUrl}"/>
<sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html">
<sly data-sly-call="${clientlib.css @ categories='wknd-events.react'}"/>
```

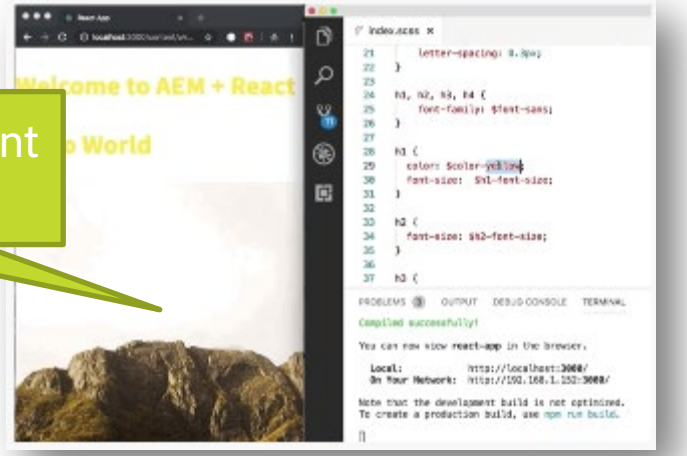
Authoring with the SPA Editor



Maven AEM Project Archetype 27+



Front End development outside of AEM



Course Agenda

Module 1: Introduction & overview – AEM & SPA architecture

Module 2: Project Setup - Maven Multi Module Project with a dedicated module for SPA development, and the AEM SPA Editor JS SDK

Module 3: Integrate the SPA - Explore project and code structure for a React SPA app that will be editable via AEM Author server.

Module 4: Map SPA Components - Map React components to AEM components to support authoring content for the SPA in AEM's page authoring environment.

Course Agenda (cont.…)

Module 5: Navigation and Routing - Development of SPA Model Routing to allow navigation between different views of the App.

Module 6: Custom component – Create a custom React component with author dialog and Sling Model to be used with the AEM SPA Editor

Module 7: Extend a component – Learn to extend an existing Core Component for use in the SPA Editor

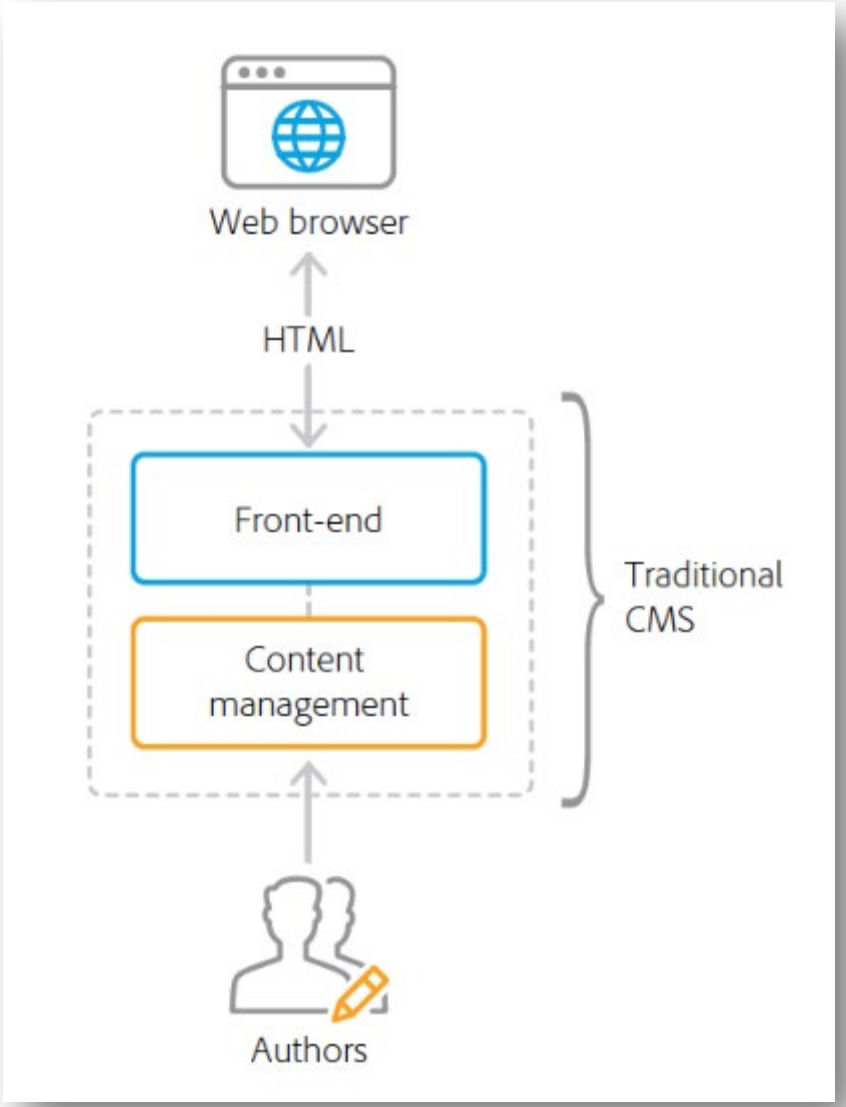
Why use Single Page Applications in AEM?

AEM SPA Editor

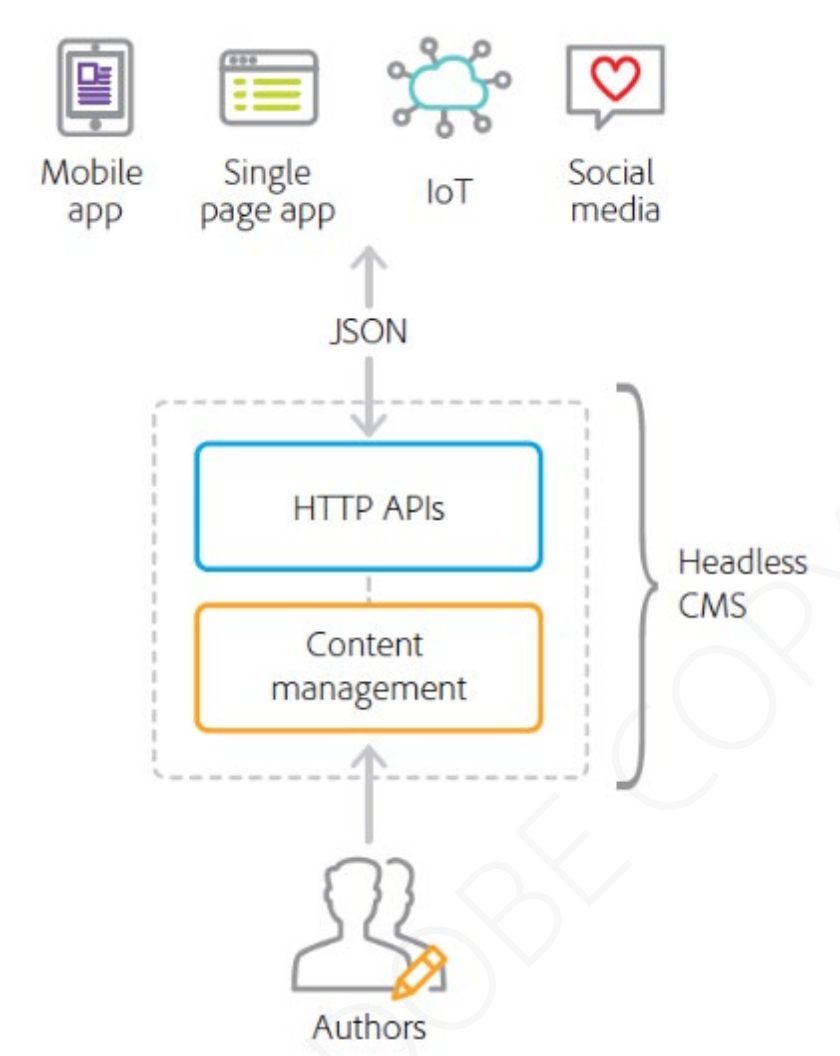
- Allows front-end developers to create SPAs that can be integrated into an AEM site
  - allowing the content authors to edit the SPA content as easily as any other AEM content
- Most or all page content is retrieved once in a single page load
  - additional resources loaded as needed based on user interaction with the page
- Reduces the need for page refreshes and presents an experience to the user that is seamless, fast, and feels more like a native app experience.

Traditional CMS Architecture

- Manages and delivers content on a single technology stack
- Minimizes total cost of ownership with efficient system maintenance and training
- Controls all of the templating and presentation logic
- Outputs fully formatted HTML
- Often tightly coupled with a single channel
  - Results in a monolithic software architecture
  - Can make multichannel content delivery difficult and costly.



Headless CMS Architecture

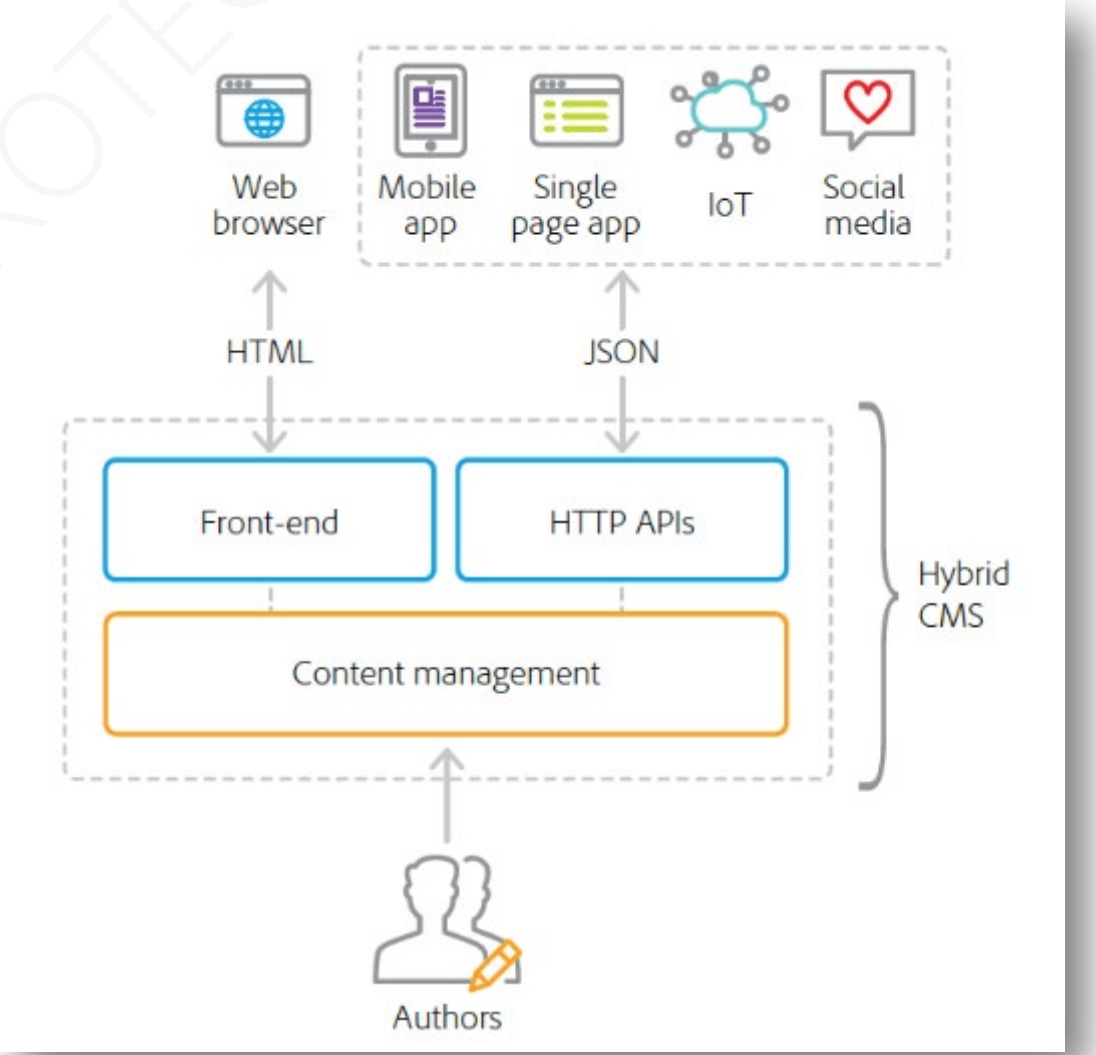


- Does not deliver HTML or formatted content directly
- Decouples content from presentation
- Enables content to be used by a variety of front-end technologies.
- Exposes content through well-defined HTTP APIs

Note: The term 'headless' originates from the idea that the front-end presentation layer is the "head" of the application

Hybrid CMS Architecture

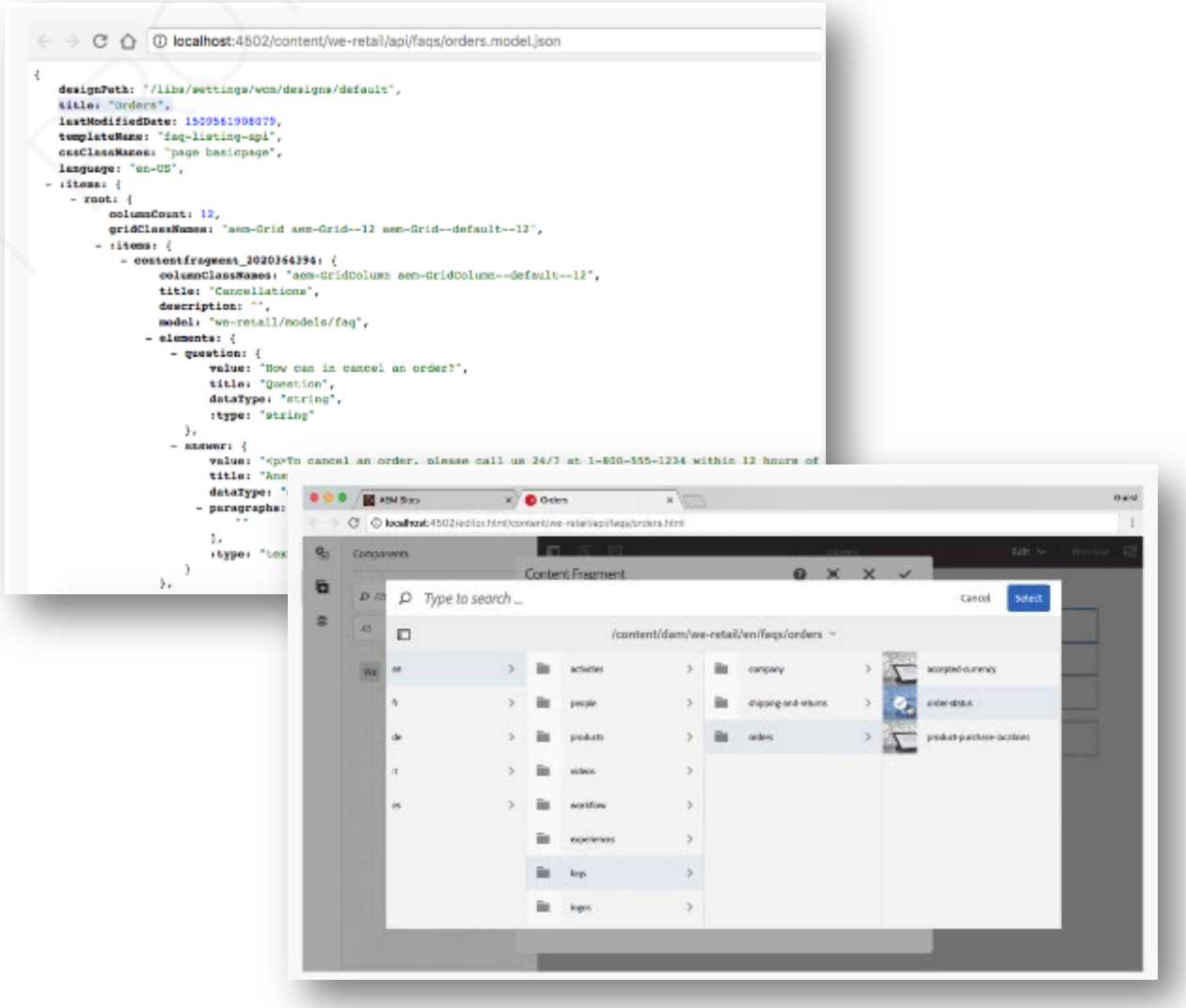
- AEM takes a hybrid approach
  - Efficiency and ease of use of a traditional CMS
  - Flexibility and scalability of a headless development framework
- Provides a central hub to house and distribute content and experiences to any channel
- Delivers content either as fully formatted HTML or as JavaScript Object Notation (JSON) over HTTP APIs
- Brands can choose between traditional or headless delivery and can mix and match capabilities to meet their business needs



Hybrid CMS Architecture (cont.)

AEM includes features that express content in a variety of formats through API endpoints:

- **Content services**
  - Exposes content via HTTP APIs using a standard JSON schema, enabling brands to expose content to any channel without coding.
- **Sling Model Exporter**
  - Allows developers to quickly render any Experience Manager content into JSON using custom business logic.

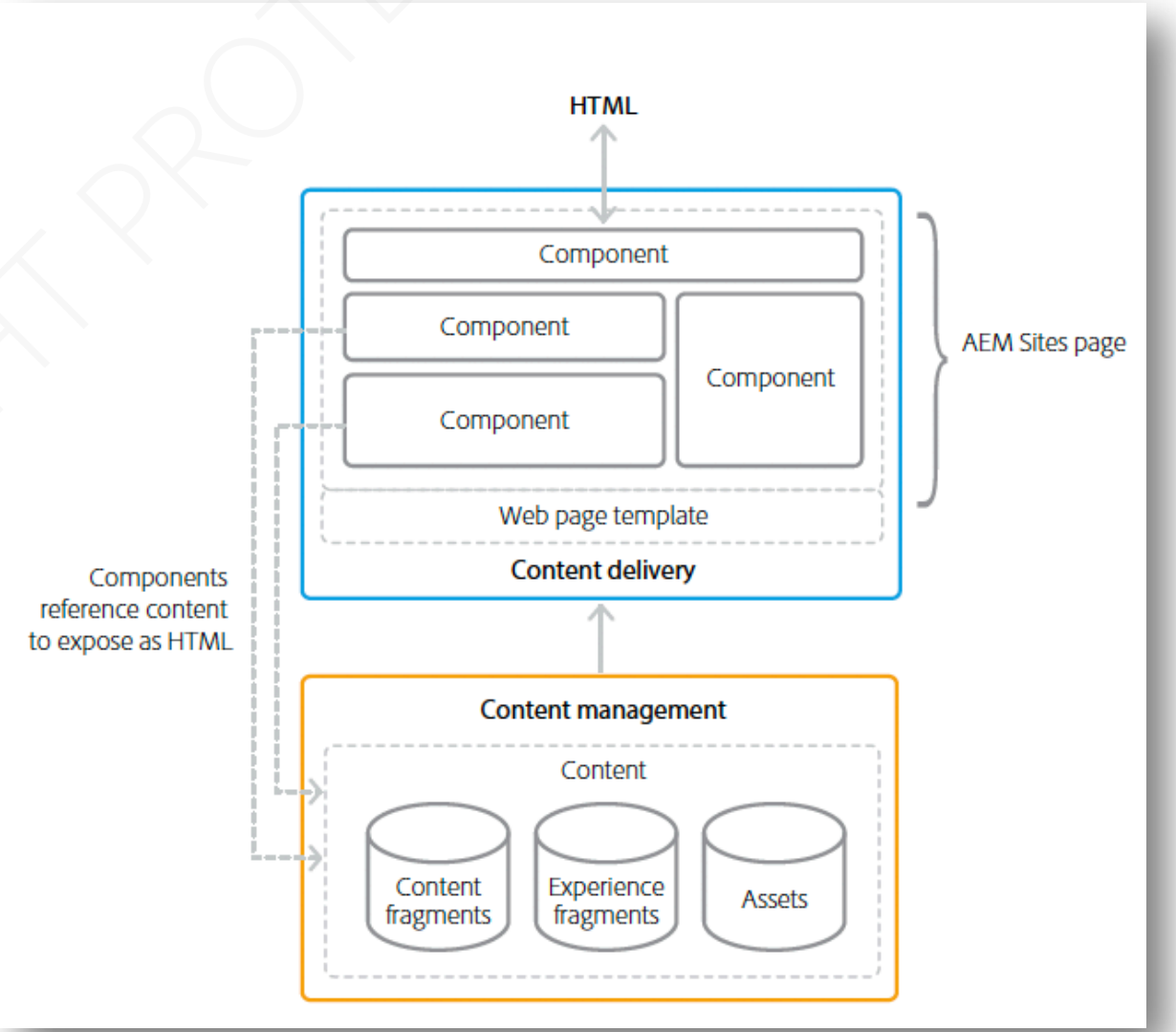


Multi-Channel Content Delivery and Content Services

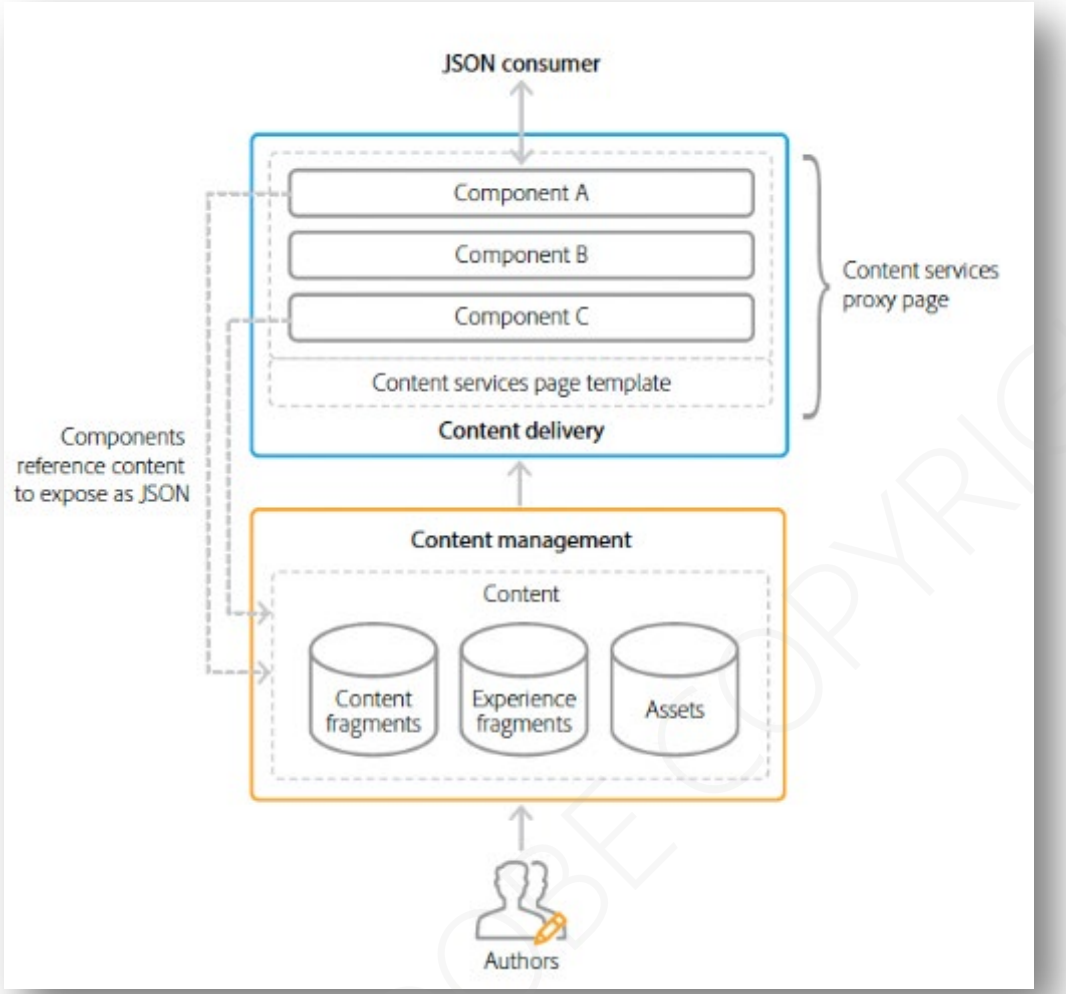
To support all channels a CMS must expose the managed content in a multi-format and scalable way.

Standard formats for exposing web content:

- HTML – language of the web browser
- JSON – lightweight delivery format that powers modern connected applications



AEM Delivery of HTML and JSON



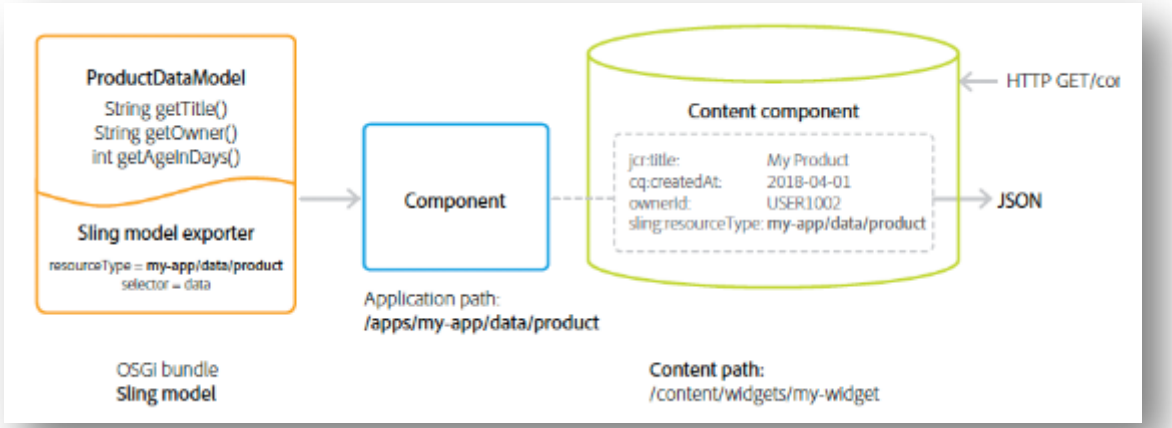
- AEM uses templates, pages, and components to compose content and deliver it via both HTML and JSON
- Components can reference and reuse content abstraction features within the same channel or across channels :
  - Content fragments
  - Experience fragments
  - Assets

Sling Models

- Preferred method for implementing the business logic of an AEM component.
- Supports both, HTML components and headless Content Management System (CMS).
- Java objects that can be mapped to Sling resources.
- Can export JSON objects
  - JSON objects are then used by programmatic web consumers such as other web services and JavaScript applications

AEM Sling Model Exporter

- Builds on the Sling HTTP web framework
- Extension of Sling Models
- Provides a mechanism for serializing Sling Model Java objects into any JSON format
- Sling Models
  - Developed in Java as annotated POJOs
  - Represent some logical set of content or data



AEM Sling Model Exporter

- Facilitates the automatic serialization of the Sling Model object into JSON using the FasterXML Jackson library
- Binds to content using the content's sling:resourceType and a selector
  - Allows developers to attach distinct JSON representations to different content
  - HTTP requests to the content resource retrieve the content in its JSON format
- Content resource has the matching sling:resourceType set and a customs selector and extension:

  HTTP GET /path/to/content.<selector>.json

  for example: *HTTP GET /content/site/en/products.details.json*

Enabling SPA with AEM Site SPA Editor

- Developers can build SPA experiences rapidly with popular JavaScript frameworks such as React and Angular
  - Typically using decoupled or “headless” CMS
- AEM Sites
  - Enables developers by integrating with familiar frameworks and development tools
  - SPA Editor offers a WYSIWYG user interface that enables marketers and authors to make in-context changes to content, layout, and presentation, just as they would with traditional web pages

Summary: HTML Delivery vs. JSON Delivery

	HTML delivery	JSON delivery
Capability or feature	Sites, Screens, Forms	Content services
Underlying technology	Sling HTTP web framework	Sling Model Exporter
Templates	Responsive layouts that define the structure and appearance of a web page	JSON schema definitions
Pages	Traditional pages forming an Experience Manager website, based on templates and components	HTTP API JSON endpoints adhering to the JSON schema definition that includes templates and components
Components	Modules that collect and present content on a page	Modules that collect and transform content into normalized JSON
Referenced content	Content fragments, experience fragments, Assets	Content fragments, Assets

Introduction to the AEM Repository

Introduction to the Java Content Repository

Java Content Repository (JCR) is a database that looks like a file system.

- It is unstructured and enables versioning and observation.
- It provides services such as full-text search, indexing, access control, and event monitoring.

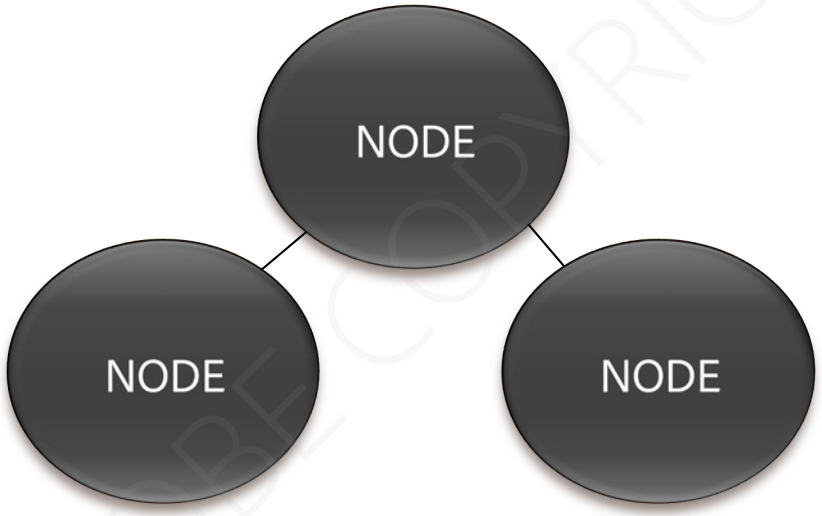
JSR-283 key principles:

- A common programmatic interface to content repositories
- An API not tied directly to the underlying architecture, data source, or protocol
- Content organization in a repository model - Hierarchical modeling
- The JCR is implemented by Apache Jackrabbit Oak

JCR Nodes and Properties

The JCR has a hierarchical tree structure with two item types: nodes and properties.

- Nodes: Provide the structure. They can have parent and child nodes which are represented by paths.
- Properties: Store the data. All properties have a name and a value.



Common Node Types

NT

Node Type	Description
nt:file	Represents a file in a filesystem
nt:folder	Represents a folder in a filesystem
nt:unstructured	Allows any combination of child nodes and properties. It also supports client-orderable child nodes and stores unstructured content and commonly used (for touch UI) dialog boxes.

AEM

Node Type	Description
cq:Page	Stores the 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 the editing configuration for a component including drag-and-drop and in-place editing
cq:InplaceEditingConfig	Defines an in-place editing configuration for a component. It is a child of cq:EditConfig

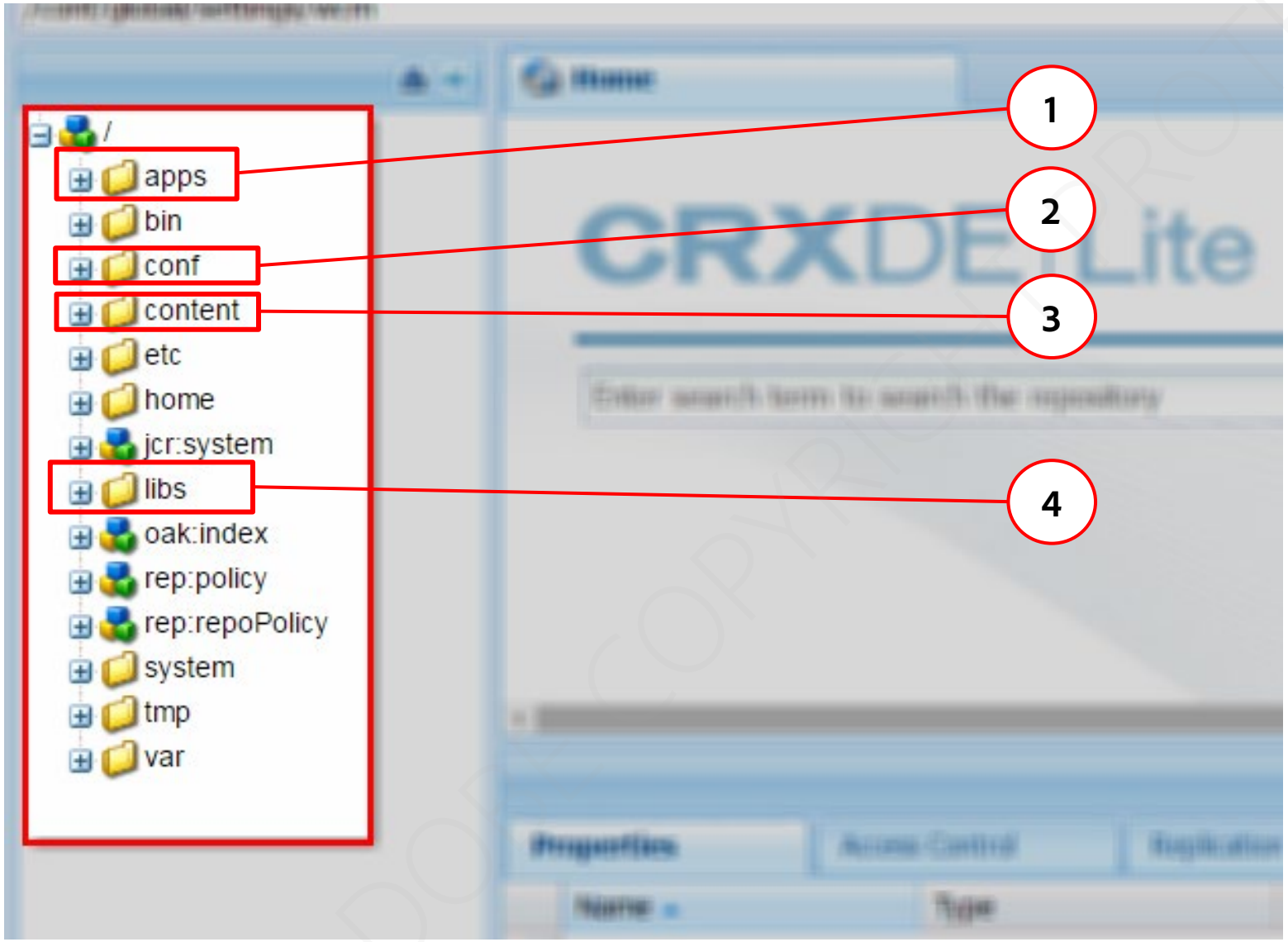
JCR Nodes and Properties (cont.)

parent node

└→ this node

- → node type: nt:folder nt:file nt:unstructured rep:User rep:ACL nt:nodeType …
- → mixin types: mix:versionable mix:lockable sling:VanityPath cq:Taggable …
- → single value properties: value — string long date path binary …
- → multi-value properties: values[] — string long date path binary …
- → child nodes

Folder Structure of the Repository



1. /apps – clientlibs, components, bundles, **front-end developer's work**
2. /conf – context-aware and global configurations such as editable templates, metadata schemas, metadata profiles, search filters, and cloud configurations
3. /content – page content authored by **authors**
4. /libs – foundation components, css, and library files

Client-Side Libraries

In AEM, you can include CSS and JS libraries just like in traditional site development, but….

- *This may not be a good idea* because:
  - It increases page overhead—files are included in all pages whether they are needed or not.
  - Components will not be portable.
    - You have to manually examine/modify the new site templates to ensure that the component has all necessary libraries.

Instead…

Convenience wrapper for com.adobe.granite.ui.clientlibs.impl.HtmlLibraryManagerImpl service interface

- Manages client-side inclusions like JS and CSS
- Determines which files should be included on the page
- Ensures duplicate files are not sent

Client-Side Libraries (cont.)

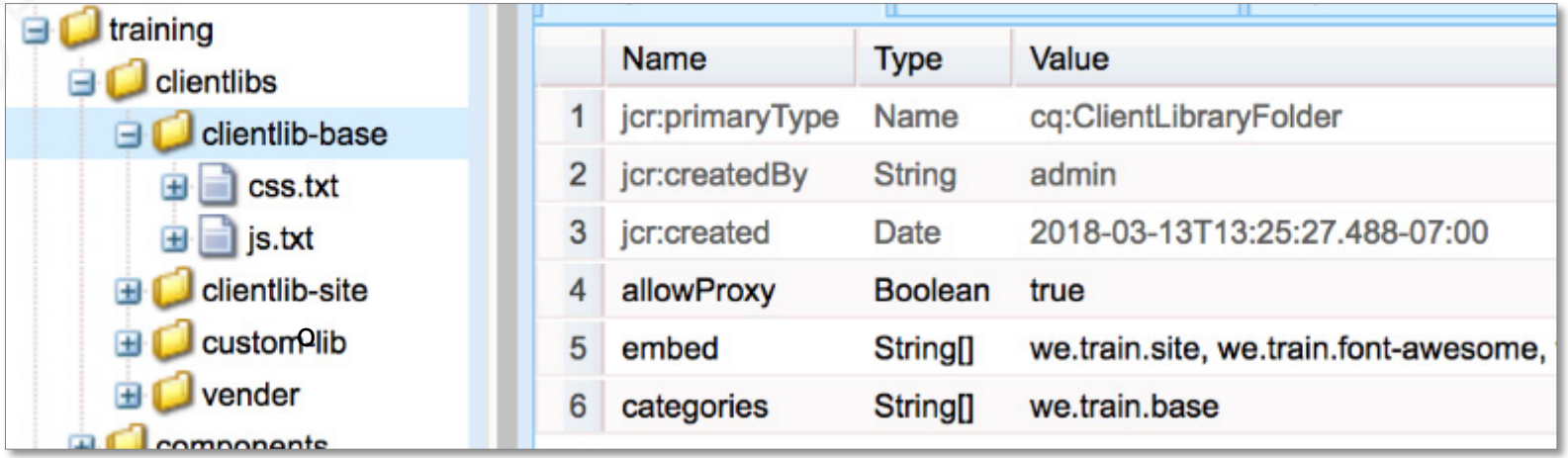
- AEM uses client-side library folders to organize the client-side libraries logically.
- The basic goals of client-side libraries are to:
  - Store CSS/JS in small discrete files for easier development and maintenance
  - Manage dependencies on third-party frameworks in an organized fashion
  - Minimize the number of client-side requests by concatenating CSS/JS into one or two requests
  - Minimize CSS/JS that is delivered to optimize speed/performance of a site

Structure of Client-Side Libraries

A client-side library folder is a repository node of type `cq:ClientLibraryFolder`.

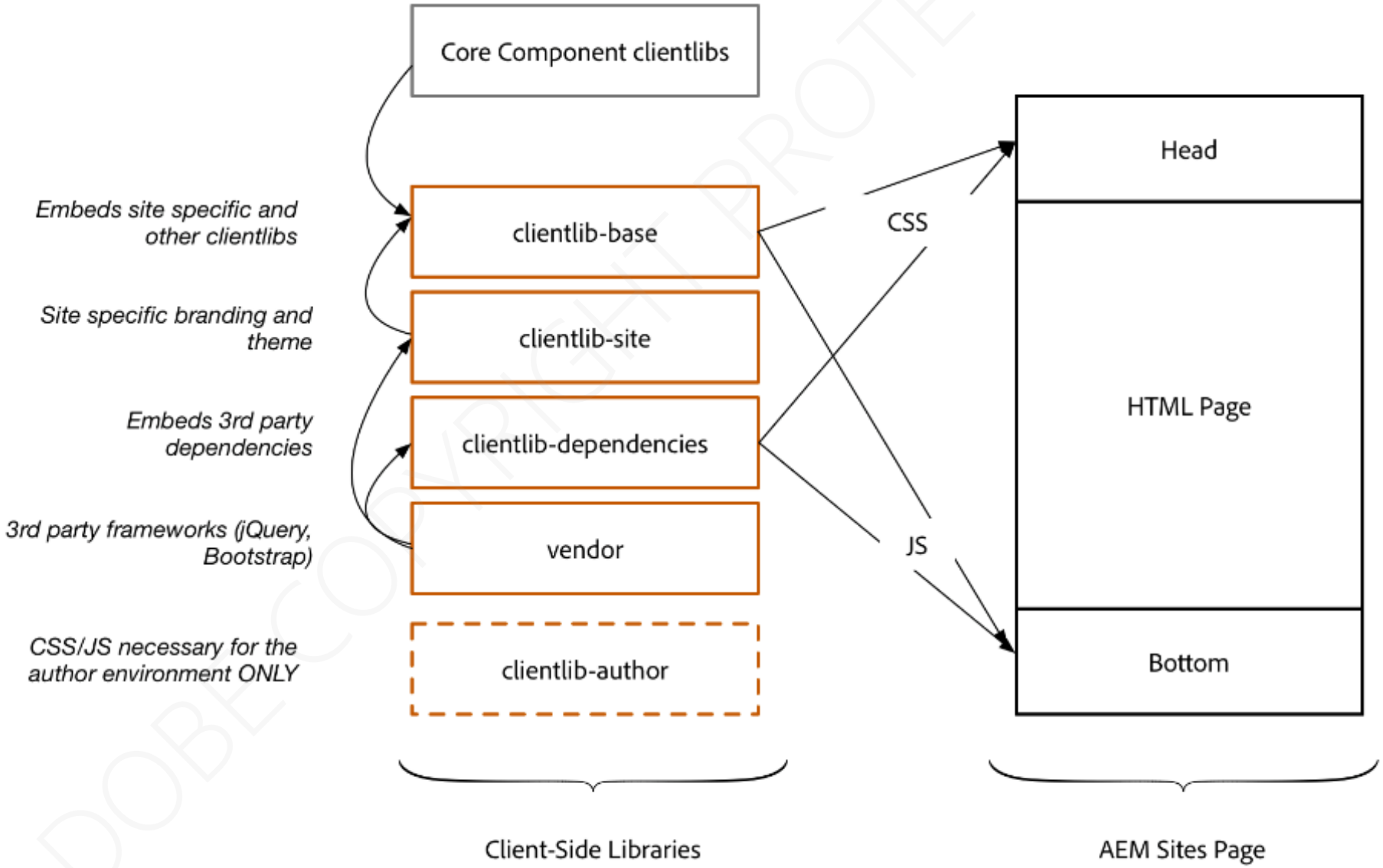
The definition in Compact Namespace and Node Type Definition (CND) notation is:

```
[cq:ClientLibraryFolder] - jcr:primaryType
- categories (String[])
- embed (String[])
- dependencies (String[])
+ css.txt (nt:file)
+ js.txt (nt:file)
```



	Name	Type	Value
1	jcr:primaryType	Name	cq:ClientLibraryFolder
2	jcr:createdBy	String	admin
3	jcr:created	Date	2018-03-13T13:25:27.488-07:00
4	allowProxy	Boolean	true
5	embed	String[]	we.train.site, we.train.font-awesome,
6	categories	String[]	we.train.base

Clientlibs Structure

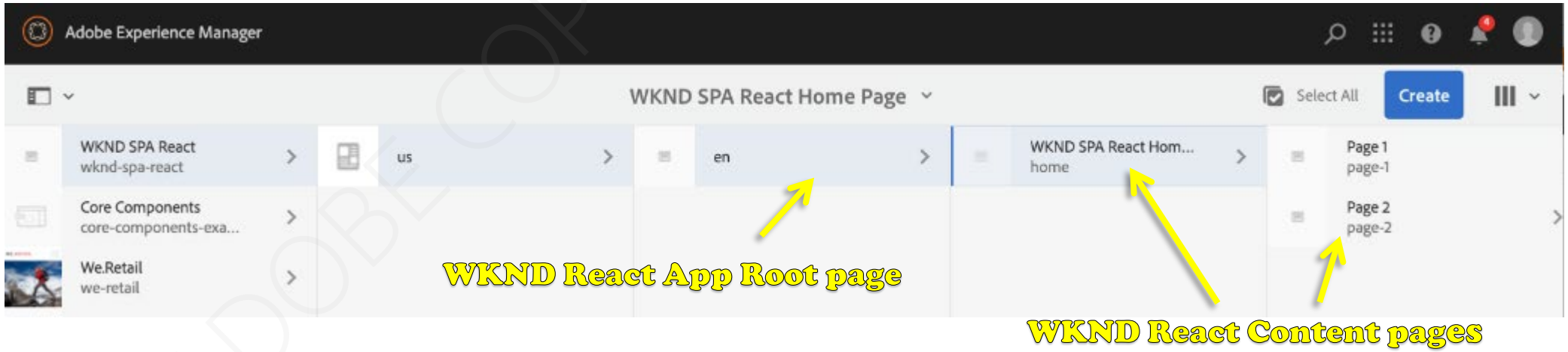


Referencing Client-Side Libraries

- You can include the client-side libraries in HTML Template Language (HTL).
- In HTL, the client-side libraries are loaded through a helper template provided by AEM.
- You can access the template through data-sly-use.
- Three templates are available and can be called through `data-sly-call`:
  - css: Loads only the CSS files of the referenced client libraries.
  - js: Loads only the JavaScript files of the referenced client libraries.
  - all: Loads all files of the referenced client libraries (both CSS and JavaScript).

Basic Content Structure for this Course

- Two editable templates
  - WKND App Root Template - represents the root of the application
  - WKND React Page Template - normal content page(s) for SPA application
  - Other Templates you may create – additional content page(s) for SPA application
- Page Structure
  - <Customer Specific URL base> / <React App Root > / Home Page / SPA Content Pages



Module 2

SPA Editor Project

Module 2 – Pre-requisites

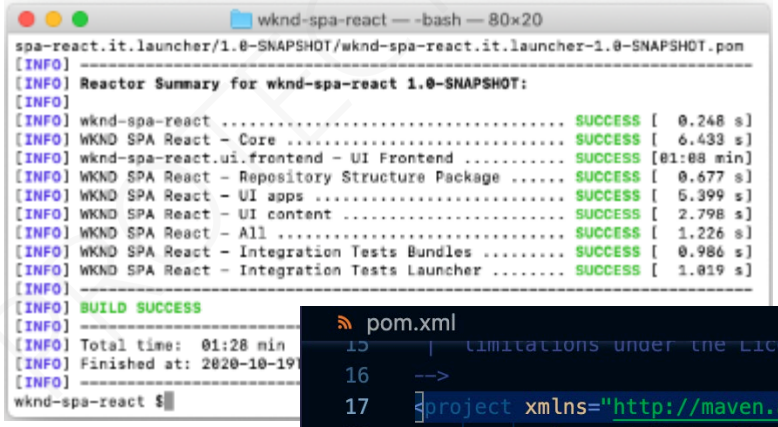
- Local development environment
- Software
  - Java 1.8 or 1.11
  - AEM as a Cloud Service, AEM 6.5.4+ or 6.4.8+
  - Apache Maven (3.3.9 or newer)
  - Node.js (v12.x recommended)
  - npm 6+
  - Eclipse or other IDE

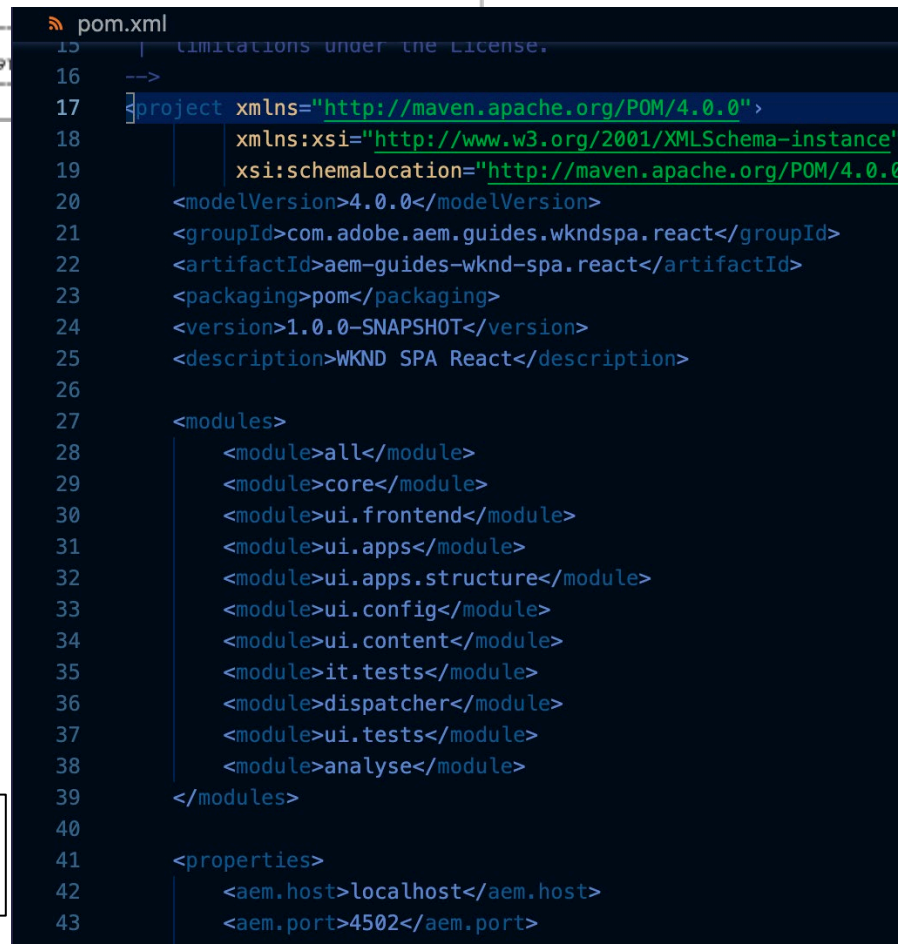
Apache Maven

- Software project management and comprehension tool
- Based on the concept of a project object model (POM)
  - Manage a project's build, reporting and documentation from a central piece of information

```
mvn -B archetype:generate \
 -D archetypeGroupId=com.adobe.aem \
 -D archetypeArtifactId=aem-project-archetype \
 -D archetypeVersion=27 \
 -D appTitle="WKND SPA React" \
 -D appId="wknd-spa-react" \
 -D artifactId="aem-guides-wknd-spa.react" \
 -D groupId="com.adobe.aem.guides.wkndspa.react" \
 -D frontendModule="react" \
 -D aemVersion="cloud"
```

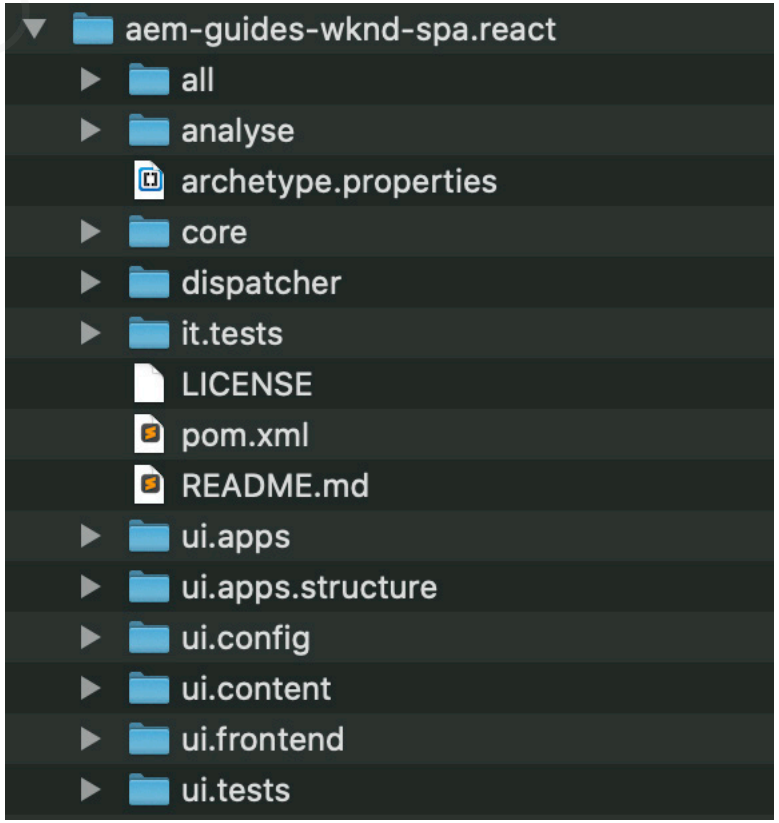
```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
```





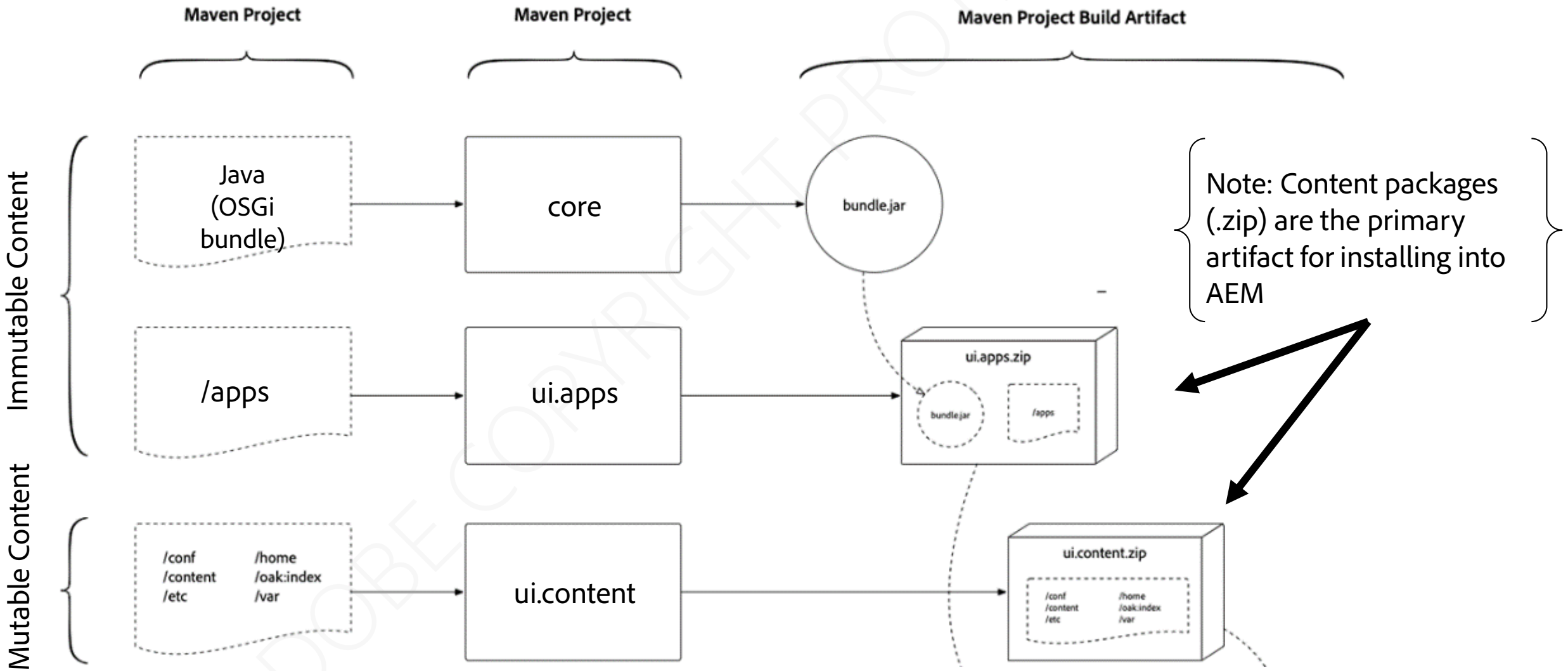
AEM Archetype

- Minimal AEM project
- Standard for best practice
  - Cloud packaging (compatible with earlier versions too)
  - Mutable/immutable separation
  - Core Components
  - Editable templates (page and XF)
  - Layout Containers
  - XF header/footer concept
  - Site content structure
  - Java examples
- Optional Modules
  - dispatcher
  - ui.frontend
- And more...

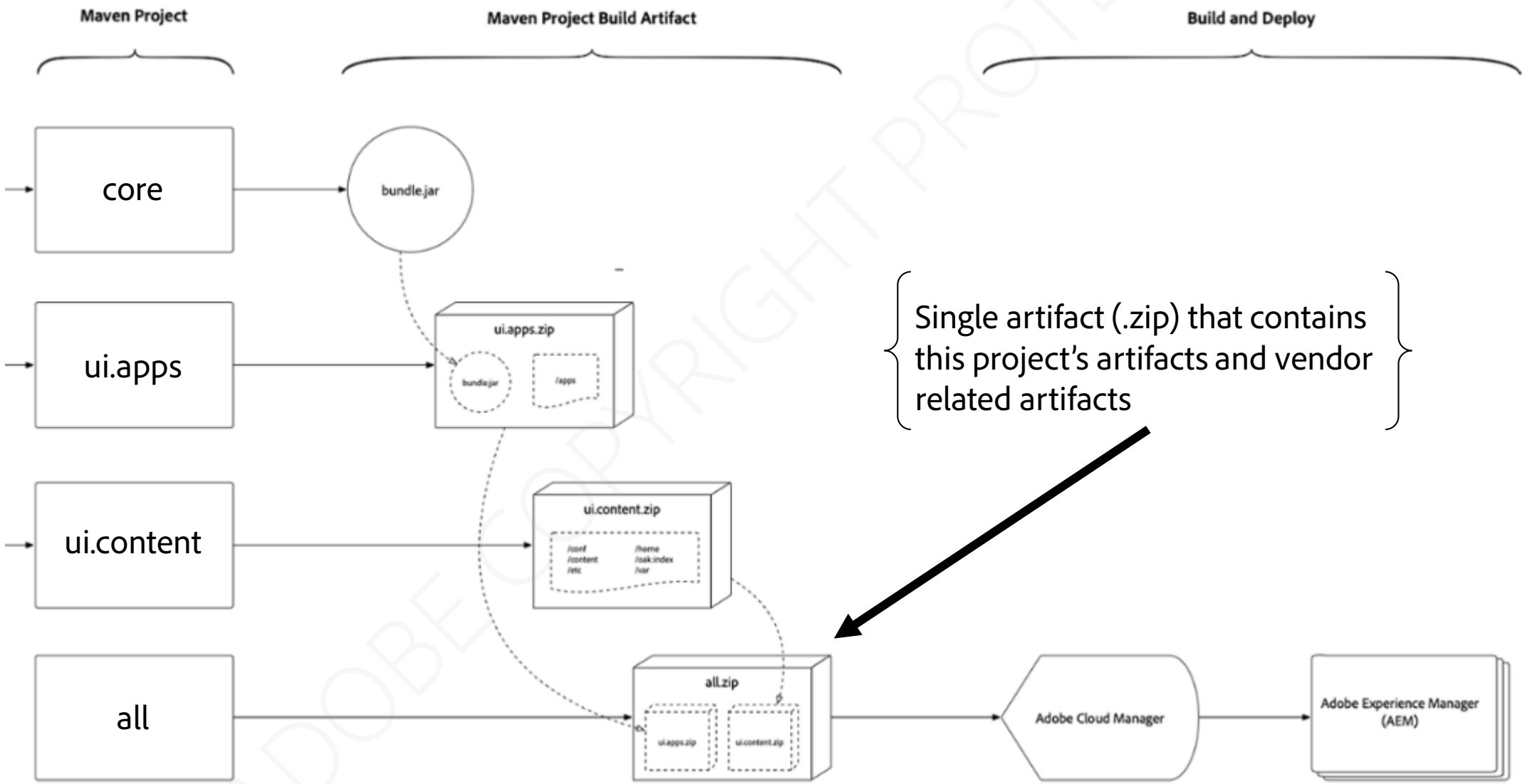


https://github.com/adobe/aem-project-archetype/releases

Primary Artifacts for Development



Single Content Package (all)



Create a Project with Maven

- Adobe's nexus repo needs to be added to settings.xml
  - located in the .m2 directory
- Once updated, a project can be generated
  - Only first 4 lines are required
  - The rest of the parameters can be answered with interactive mode
- Project can then be installed into AEM
  - `$ mvn clean install -PautoInstallSinglePackage`

```
mvn -B archetype:generate \
 -D archetypeGroupId=com.adobe.aem \
 -D archetypeArtifactId=aem-project-archetype \
 -D archetypeVersion=27 \
 -D appTitle="WKND SPA React" \
 -D appId="wknd-spa-react" \
 -D artifactId="aem-guides-wknd-spa.react" \
 -D groupId="com.adobe.aem.guides.wkndspa.react" \
 -D frontendModule="react" \
 -D aemVersion="cloud"
```


45

UberJar (AEM 6.5) and aem-sdk-api (cloud)

- Special JAR files provided by Adobe
  - Used to reduce the number of dependencies.
  - AEM 6.5 requires the UberJar, whereas Cloud requires the aem-sdk-api.
- UberJar and aem-sdk-api contain:
  - All public Java APIs exposed by AEM.
  - Limited external libraries—all public APIs available in AEM that comes from Apache Sling, Apache Jackrabbit, Apache Lucene, Google Guava, and two libraries used for image processing.
  - Interfaces and classes exported by an OSGi bundle in AEM.
  - MANIFEST.MF file with the correct package export versions for all the exported packages.

UberJar (AEM 6.5)

- UberJar is a special JAR file provided by Adobe
  - It is used to reduce the number of dependencies.
- UberJar file contains:
  - All public Java APIs exposed by AEM.
  - Limited external libraries—all public APIs available in AEM that comes from Apache Sling, Apache Jackrabbit, Apache Lucene, Google Guava, and two libraries used for image processing.
  - Interfaces and classes exported by an OSGi bundle in AEM.
  - MANIFEST.MF file with the correct package export versions for all the exported packages.

“Parent” pom.xml

Project versioning

Sub-modules.
Order matters for build of complete project

```xml
<groupId>com.adobe.aem.guides.wkndspa.react</groupId>
<artifactId>aem-guides-wknd-spa.react</artifactId>
<packaging>pom</packaging>
<version>1.0.0-SNAPSHOT</version>
<description>WKND SPA React</description>

<modules>
    <module>all</module>
    <module>core</module>
    <module>ui.frontend</module>
    <module>ui.apps</module>
    <module>ui.apps.structure</module>
    <module>ui.config</module>
    <module>ui.content</module>
    <module>it.tests</module>
    <module>dispatcher</module>
    <module>ui.tests</module>
    <module>analyse</module>
</modules>
```

```xml
<!-- Development profile: install
<profile>
    <id>autoInstallBundle</id>
    <!--
 <profile>
    <id>autoInstallPackage</id>
    <activation>
 <profile>
    <id>adobe-public</id>

    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
```

build profiles defined here ‘globally’, then tie into specific sub-modules as needed

```xml
<plugin>
    <groupId>com.github.eirslett</groupId>
    <artifactId>frontend-maven-plugin</artifactId>
    <version>1.7.6</version>
    <configuration>
        <nodeVersion>v10.13.0</nodeVersion>
        <npmVersion>6.9.0</npmVersion>
    </configuration>
    <executions>
        <execution>
            <id>install node and npm</id>
            <goals>
                <goal>install-node-and-npm</goal>
            </goals>
```

Plugins & Dependencies, with versioning, that tie to specific sub-modules as needed.
Shown plugin for React module

Verify *front-end-maven-plugin* configuration – parent pom.xml

Specify *front-end-maven-plugin*, *node* and *npmVersion* in parent reactor POM match versions on your development environment.

```
<plugin>
    <groupId>com.github.eirslett</groupId>
    <artifactId>frontend-maven-plugin</artifactId>
    <version>1.7.6</version>
    <configuration>
        <nodeVersion>v10.13.0</nodeVersion>
        <npmVersion>6.9.0</npmVersion>
    </configuration>
    <executions>
        <execution>
            <id>install node and npm</id>
            <goals>
                <goal>install-node-and-npm</goal>
            </goals>
        </execution>
        <execution>
            <id>npm install</id>
            <goals>
                <goal>npm</goal>
            </goals>
```

ui.apps Module

ui.apps module includes all rendering code beneath /apps Deployed as AEM package.

```
<!-- ====================================================================== -->
<!-- P A R E N T  P R O J E C T  D E S C R I P T I O N                      -->
<!-- ====================================================================== -->
<parent>
    <groupId>com.adobe.aem.guides.wkndspa.react</groupId>
    <artifactId>aem-guides-wknd-spa.react</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <relativePath>../pom.xml</relativePath>
</parent>
```

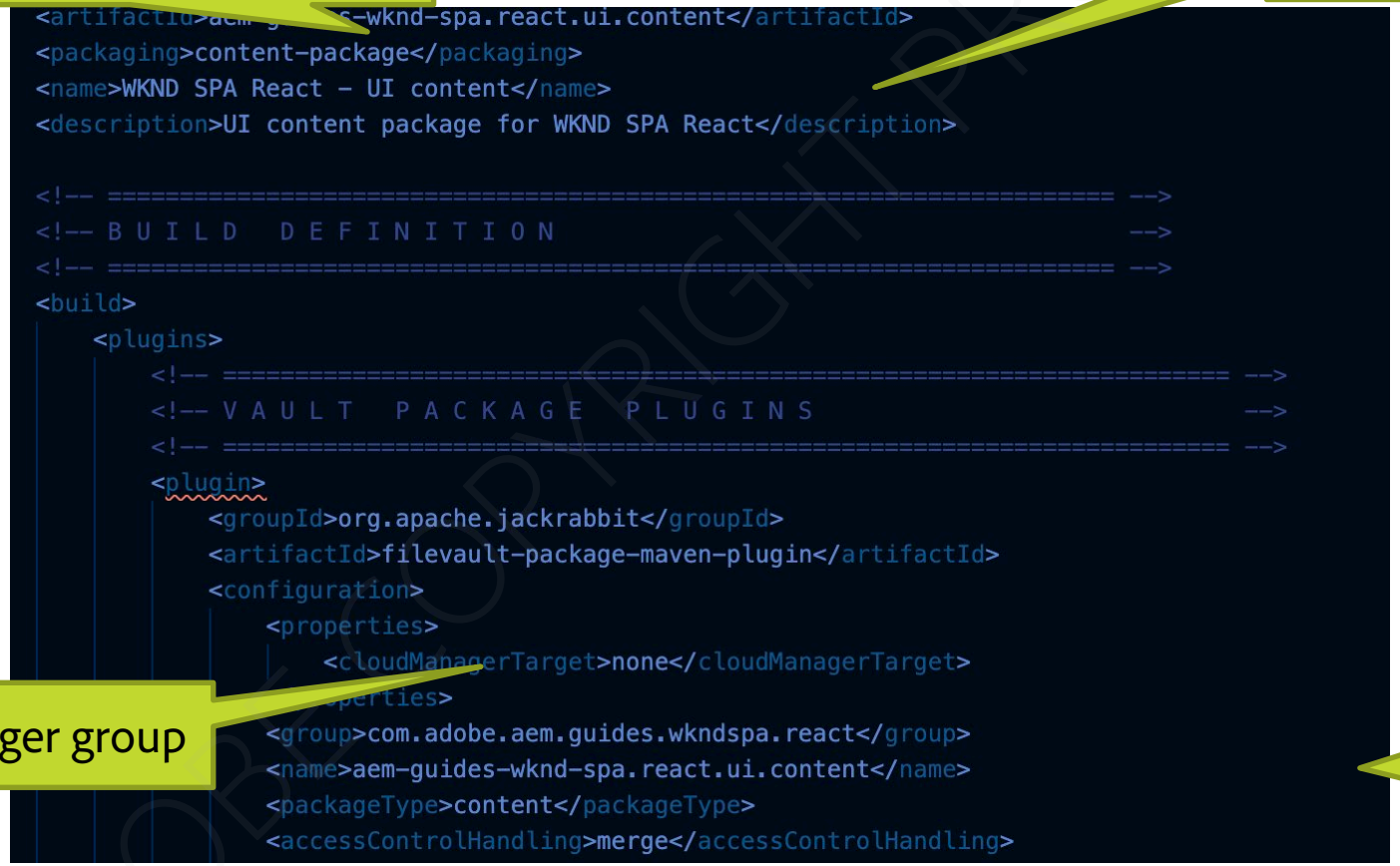
Includes dependencies to include spa project core and heirarchy page

```
<!-- SPA Project Core (includes hierarchy page) -->
<dependency>
    <groupId>com.adobe.aem</groupId>
    <artifactId>spa.project.core.ui.apps</artifactId>
    <type>zip</type>
</dependency>
<dependency>
    <groupId>com.adobe.aem</groupId>
    <artifactId>spa.project.core.core</artifactId>
</dependency>
```

ui.content Module

ui.content module contains the content beneath /content and configurations defined in /conf

compiled into a content package

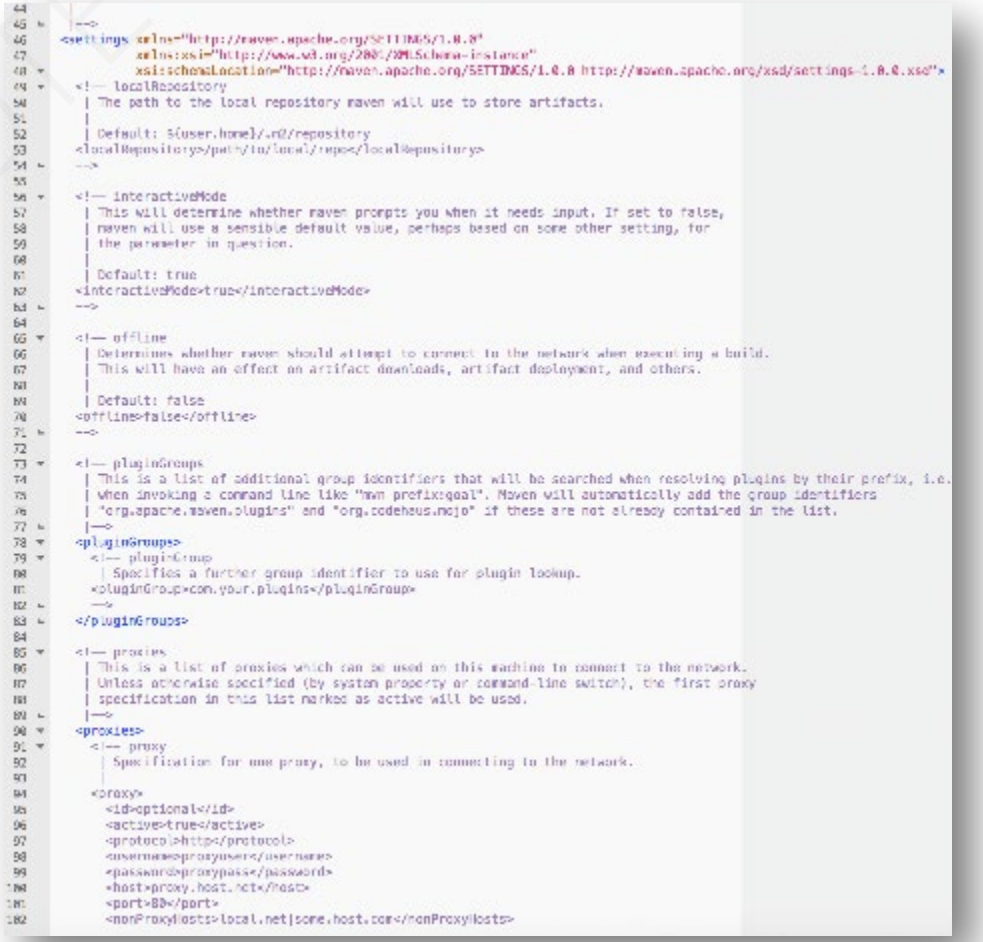


package manager group

Preserves access control information for editable templates

settings.xml

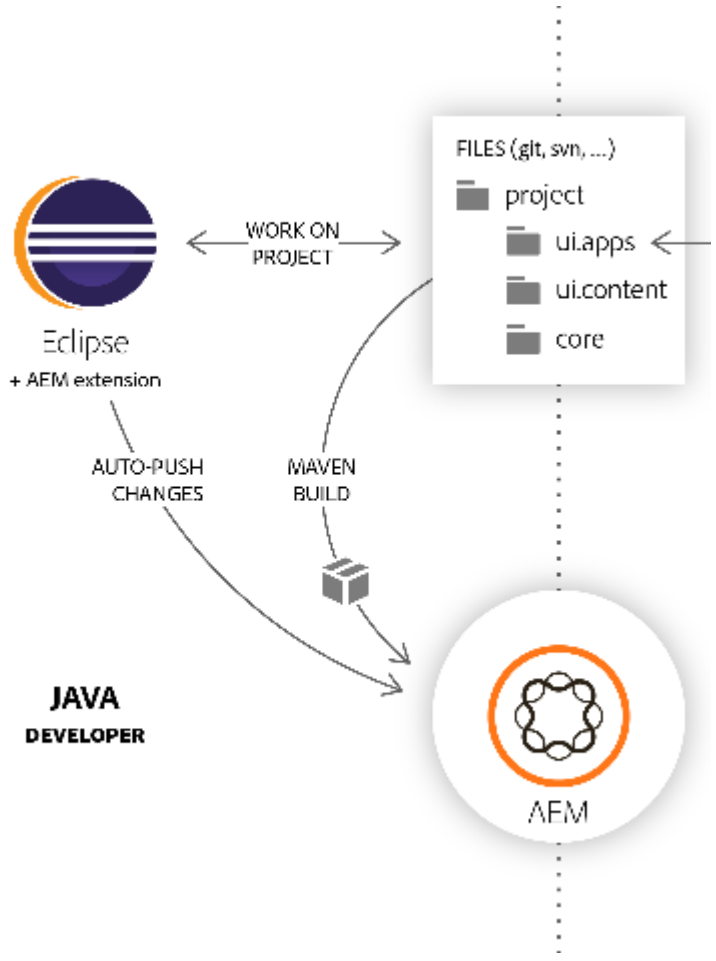
- Contains elements used to define values which configure Maven test execution in various ways, like the pom.xml, but are not associated with any specific project
  - Local repository location, alternate remote repository servers, authentication information
- User/Project-specific settings.xml can be created in the local .m2 directory



Exercise: Build starter SPA project

Using Maven for AEM Project development

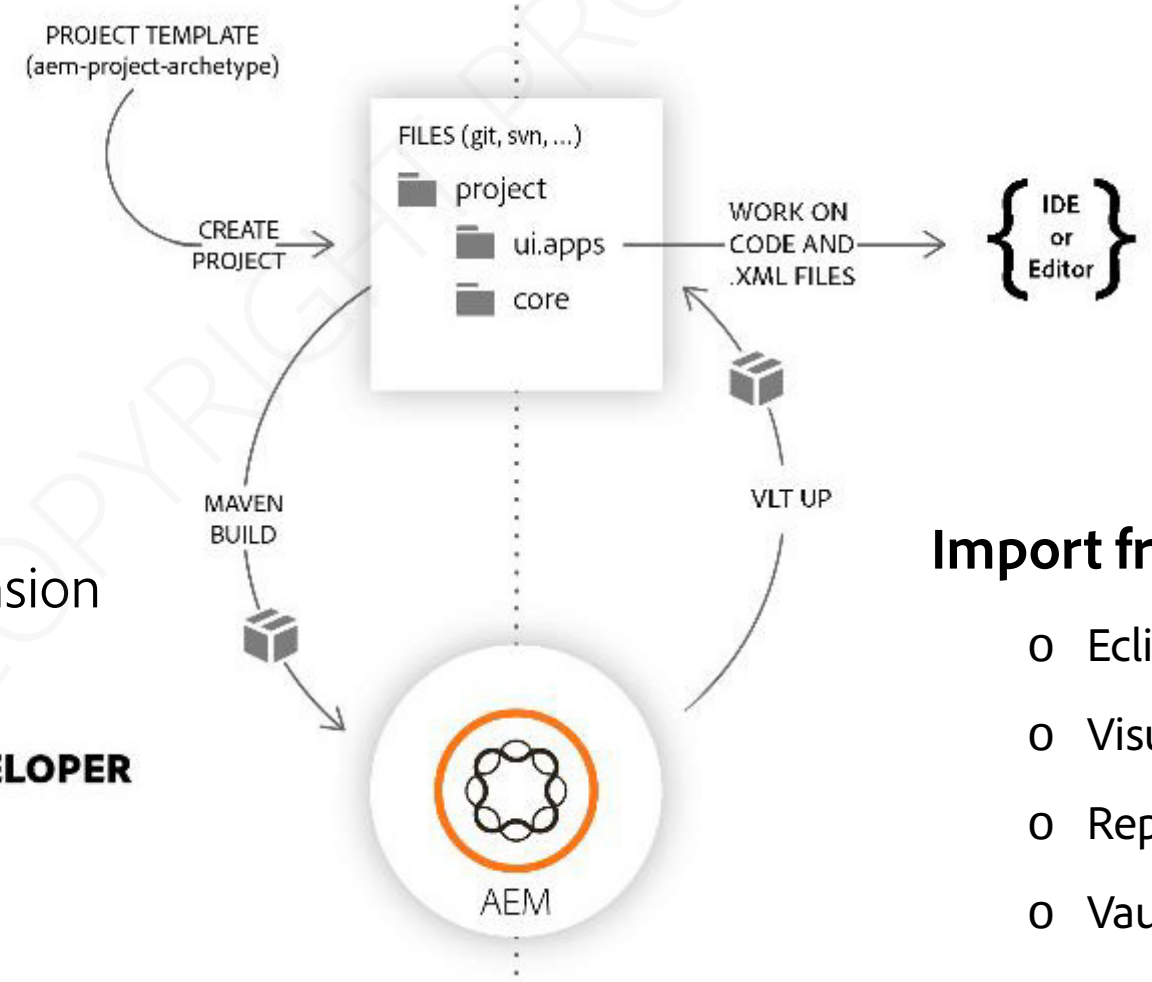
- Maven Profiles to
  - Build and install the OSGi bundle only
  - Build and deploy ui.apps and/or ui.content modules
  - Build and deploy the “all” module for the entire project
- IDEs for editing the code the project contains
  - IDEs may contain
    - plugins to execute Maven profiles
    - plugins to synchronize code to local AEM directly
  - Alternatively – run Maven profiles from command line



Tools to Export/Import Locally

Export to AEM

- Entire project or module
  - Maven command
  - Maven IDE extension
- Individual Files
  - Eclipse sync extension
  - Visual Studio sync extension
  - AEMFED command line
  - Repo command line

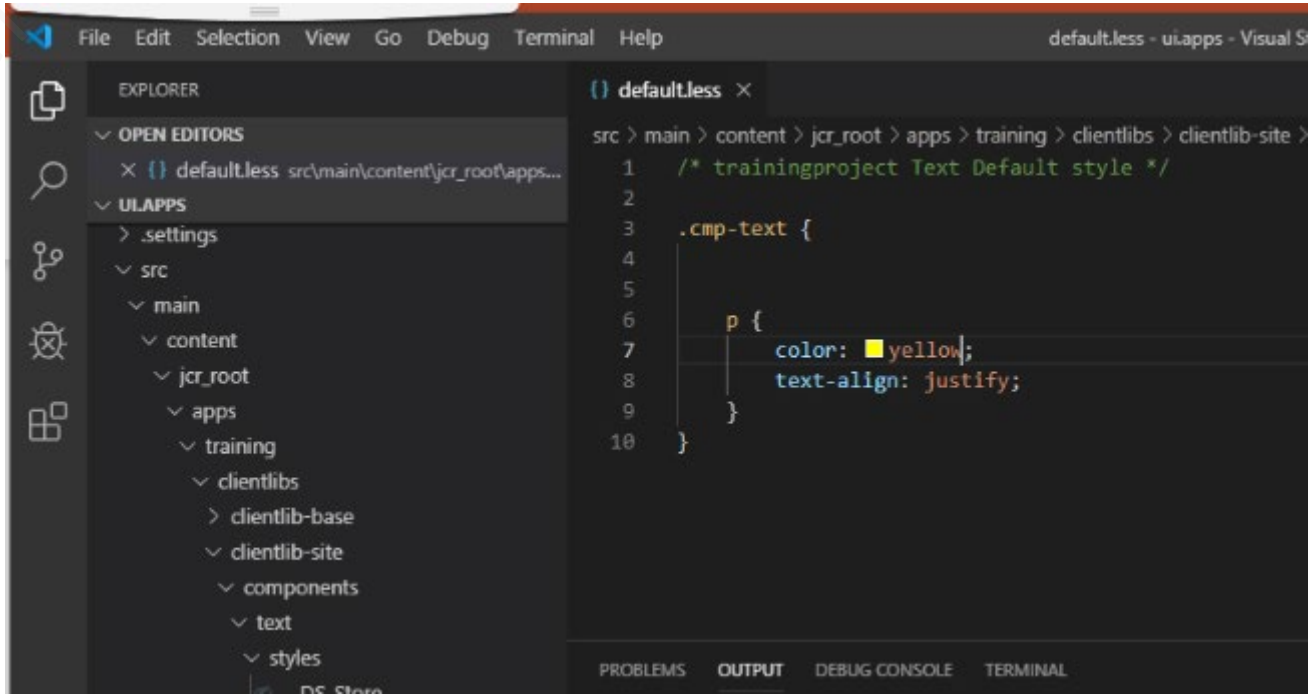


Import from AEM

- Eclipse Sync extension
- Visual Studio Sync extension
- Repo command line
- Vault command line

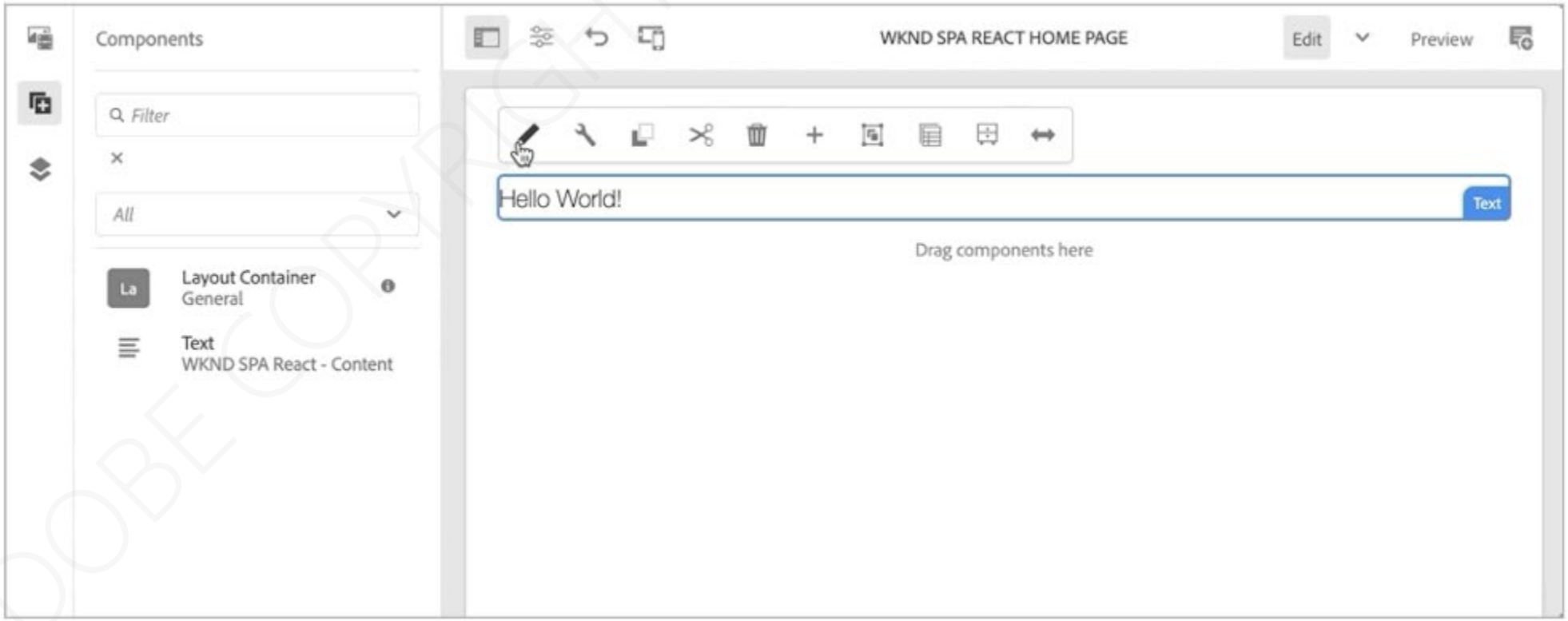
Visual Studio Code

- Lightweight and a powerful source code editor
- Has built-in support for JavaScript, TypeScript, and Node.js
- Extensions available for Java development
- VSCode AEM sync Extension to sync:
  - Files
  - Folders
  - Nodes to AEM



SPA starter app on AEM

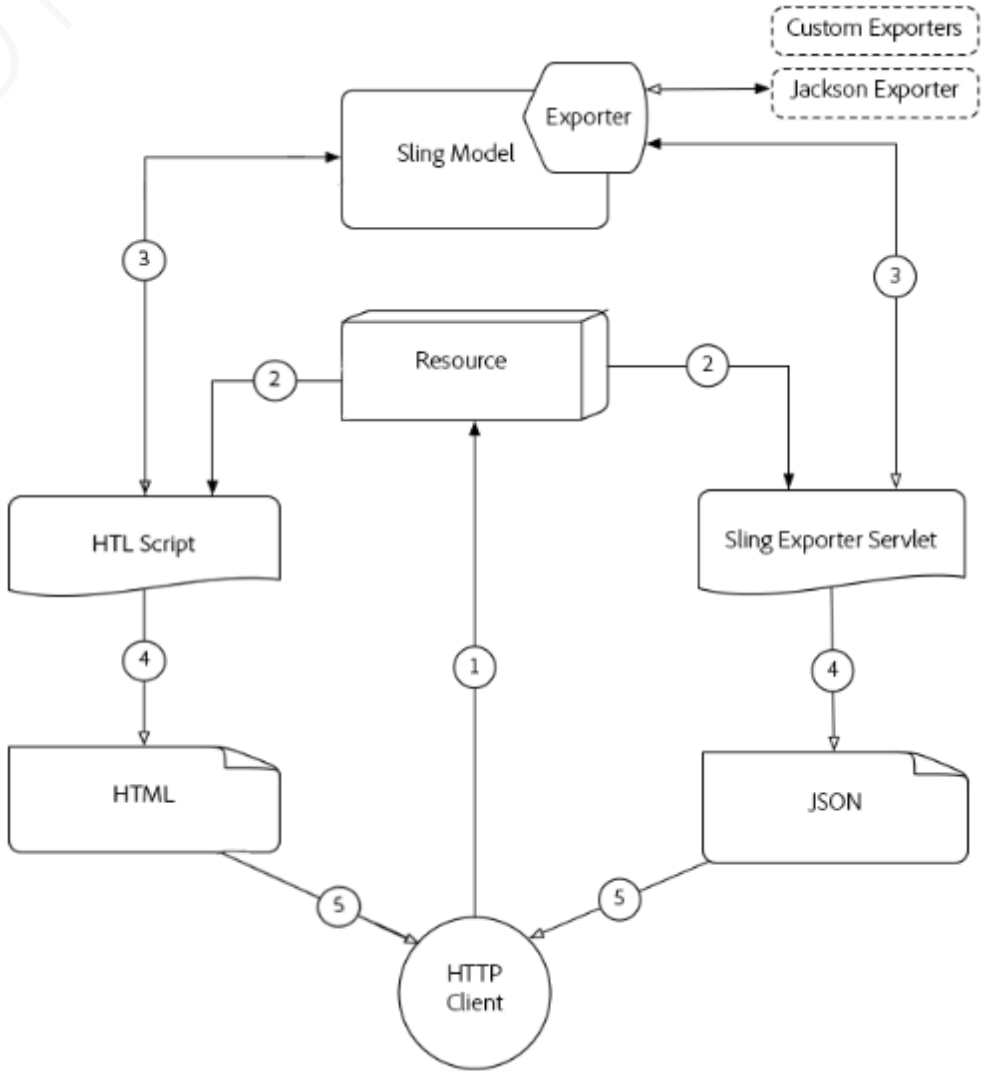
- AEM SPA Editor JS SDK deployed as part of build
  - Enables Author functions for mapped components *(more later)*
  - React .js files handle rendering of json content AEM sends



Sling Model Exporter

- Enables AEM authored content *(Resource)* to be accessed as JSON
- OOTB triggered via request to an AEM server ending in */content/path/to/page.model.json*
- AEM SPA Editor JS SDK handles automatic triggering of Model Exporter framework for content built into our SPA app.
- All AEM Core Components backed by a Sling Model

- more detail on Models and SPA Editor SDK to come….



Exercise: Author Content

Exercise: Inspect SPA code on AEM

Module 3

Integrate a SPA

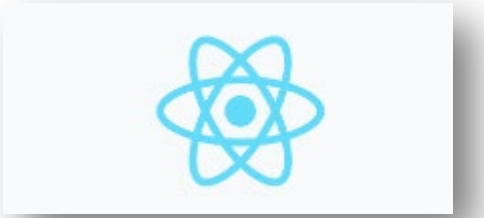
Node.js

- Asynchronous, event-driven JavaScript runtime
- Open-source
- Cross platform
- Executes JavaScript outside of a browser
  - Command line tools
  - Server-side scripting to produce dynamic web page content

npm

- Default package manager for Node.js
- Consists of 3 components
  - Website – discover packages, set up profiles, manage npm
  - Command line interface (CLI) – interact with npm
  - Registry – public database of JavaScript (JS) software and meta-information

ReactJS



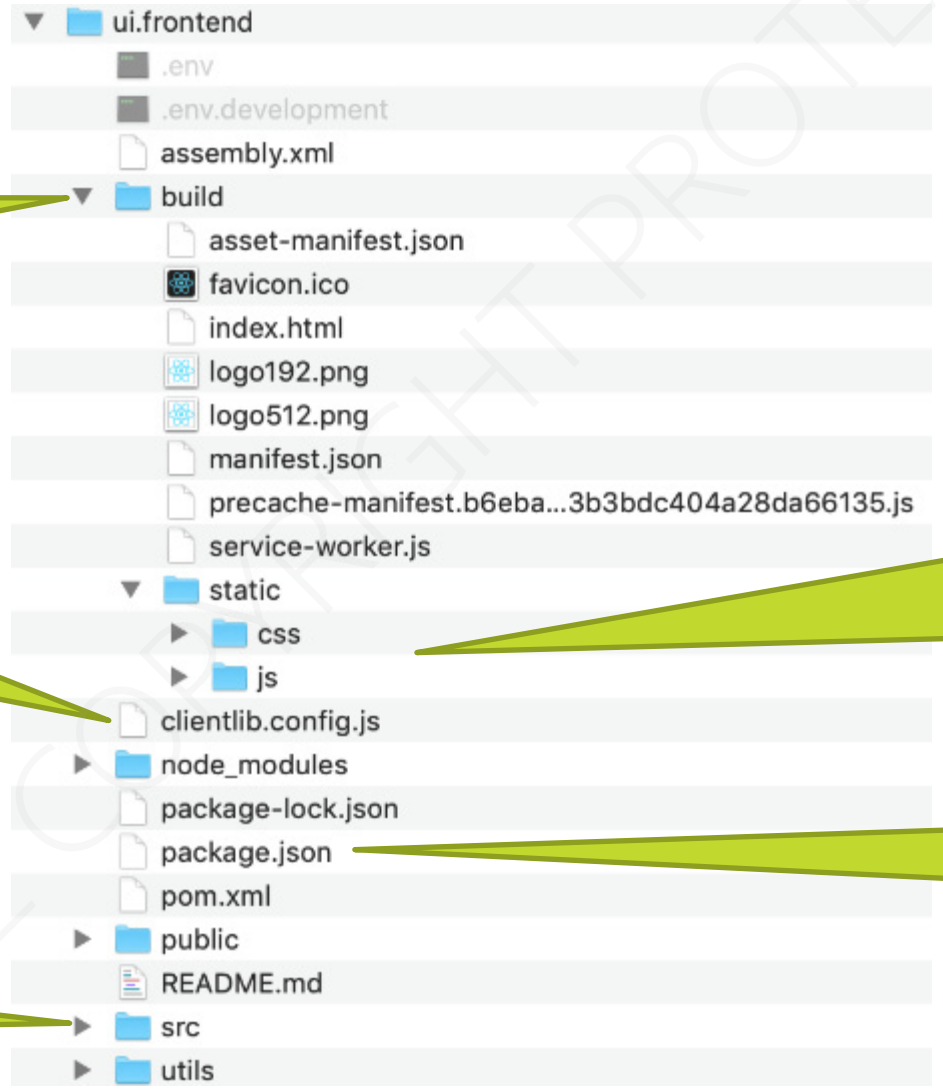
- JavaScript Library for building user interfaces
- Maintained by Facebook
- Base for development of SPA or Mobile applications

React build

Files created by **npm run build**

Defines how Client Library will be generated

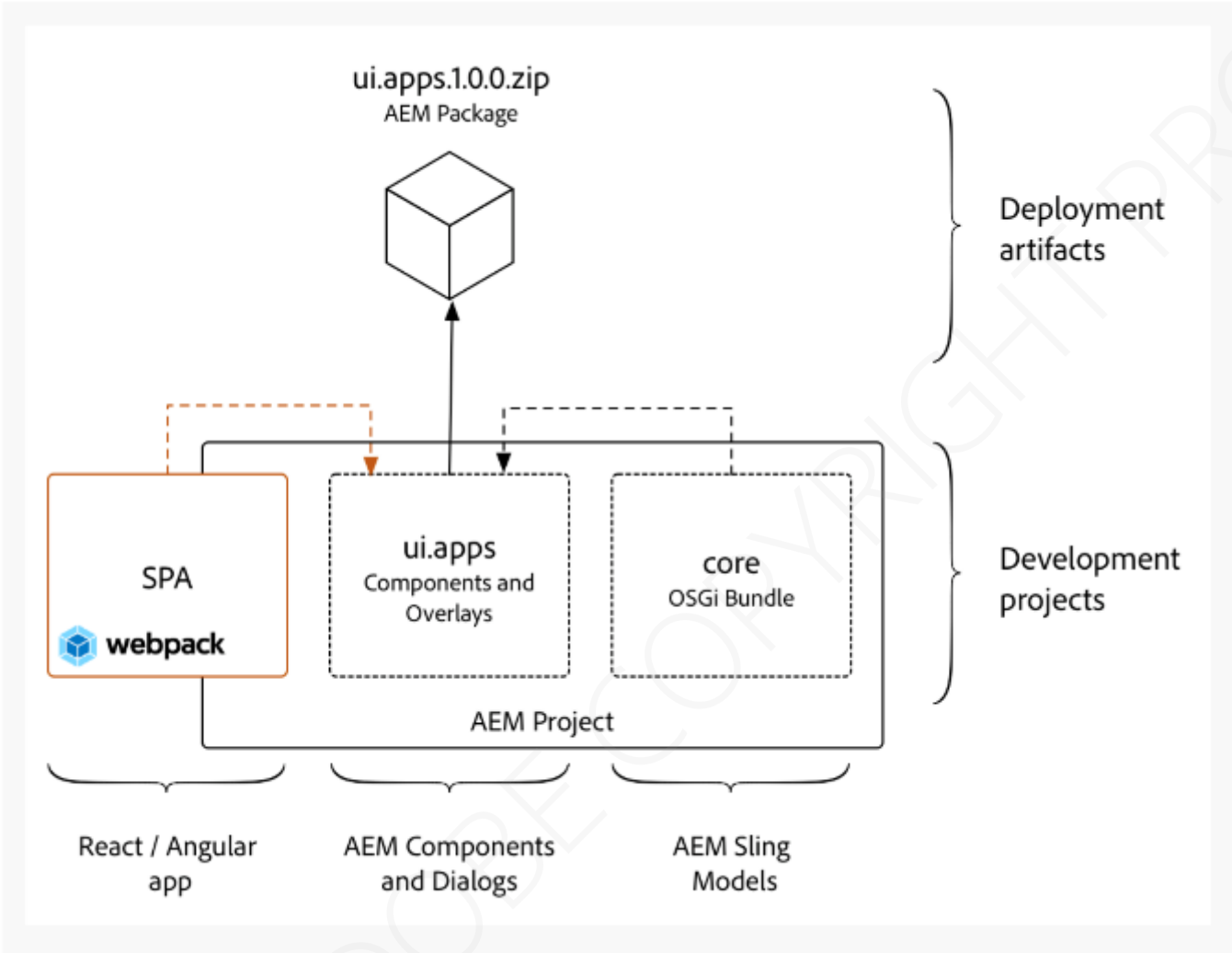
React component *.js* and *.css* files



.js and ***.css*** files will be copied into an AEM clientlib by **npm** build process

Defines dependencies & version. Defines scripts to run during build.

Integration Approach



- Development done in Webpack project. (react-app)
- Compiled SPA copied into ui.apps as a client-side library
- Deployed to AEM as a content package

Configure client libraries - */react-app/clientlib.config.js*

Create a client library in the ui.apps module at the specified path

Specify client library name, categories, + other jcr properties

Include all .css and .js files beneath ui.frontend/build/static

```
const CLIENTLIB_DIR = path.join(
  __dirname,
  '..','ui.apps','src','main','content','jcr_root','apps','wknd-spa-react','clientlibs'
);
const ASSET_MANIFEST_PATH = path.join(BUILD_DIR, 'asset-manifest.json');

const entrypoints = getEntrypoints(ASSET_MANIFEST_PATH);

// Config for `aem-clientlib-generator`
module.exports = {
  context: BUILD_DIR,
  clientLibRoot: CLIENTLIB_DIR,
  libs: {
    name: 'clientlib-react',
    allowProxy: true,
    categories: ['wknd-spa-react.react'],
    serializationFormat: 'xml',
    cssProcessor: ['default:none', 'min:none'],
    jsProcessor: ['default:none', 'min:none'],
    assets: {
      // Copy entrypoint scripts and stylesheets into the respective ClientLib
      // directories (in the order they are in the entrypoints arrays). They
      // will be bundled by AEM and requested from the HTML. The remaining
      // chunks (placed in `resources`) will be loaded dynamically
      js: entrypoints.filter(fileName => fileName.endsWith('.js')),
      css: entrypoints.filter(fileName => fileName.endsWith('.css')),

      // Copy all other files into the `resources` ClientLib directory
      resources: {
```

AEM SPA Editor JS SDK dependencies

npm modules provided by Adobe
These 3 modules make up the AEM SPA Editor JS SDK

```
"dependencies": {
  "@adobe/aem-core-components-react-base": "1.1.3",
  "@adobe/aem-core-components-react-spa": "1.1.3",
  "@adobe/aem-react-editable-components": "~1.1.2",
  "@adobe/aem-spa-component-mapping": "~1.1.0",
  "@adobe/aem-spa-page-model-manager": "~1.3.3",
  "custom-event-polyfill": "^1.0.7",
  "dompurify": "^2.0.7",
  "history": "^4.10.1",
  "react": "^16.12.0",
  "react-app-polyfill": "^1.0.5",
```

AEM SPA Editor JS SDK

- Collection of JS libraries
- Framework for editing contents of a SPA deployed in AEM
- AEM delivers content as JSON and SPA Editor JS SDK maps the JSON to React components
- Available via five npm modules:
  - @adobe/aem-spa-component-mapping - provides helpers to map AEM Components to SPA components. This module is not tied to a specific SPA framework.
  - @adobe/aem-spa-page-model-manager - provides the API to manage the model representation of the AEM Pages that are used to compose a SPA. This module is not tied to a specific SPA framework.
  - @adobe/aem-react-editable-components - provides generic React helpers and components to support AEM authoring. This module also wraps the cq-spa-page-model-manager and cq-spa-component-mapping to make these available to the React framework.
  - @adobe/aem-core-components-react-base & @adobe/aem-core-components-react-spa -set of re-usable UI components that map to out of the box AEM components. These are designed to be used as is and styled to meet your project's needs

Sites vs SPA

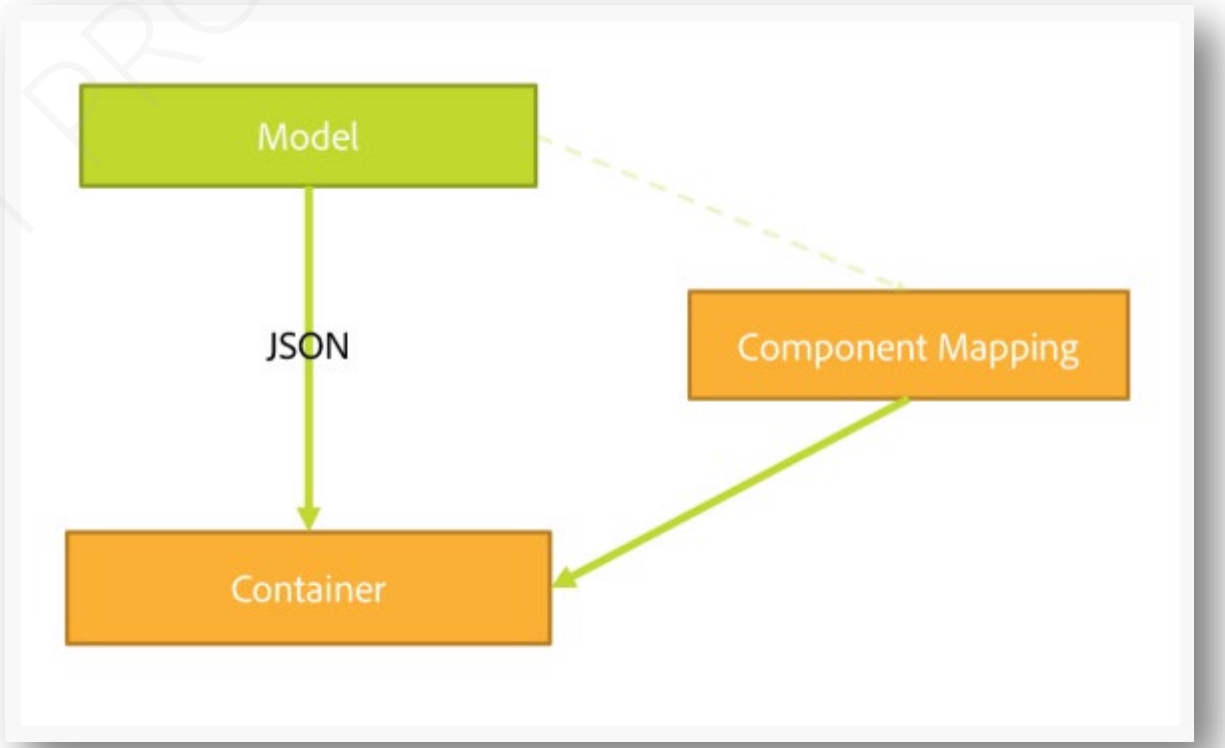
- Sites built using common SPA frameworks such as React and Angular
  - load their content via dynamic JSON
  - do not provide the HTML structure necessary for the AEM Page Editor to be able to place edit controls
- Enabling editing of SPAs within AEM requires a mapping between the JSON output of the SPA and the content model in the AEM repository
- SPA support in AEM introduces a thin JS layer that interacts with the SPA JS code when loaded in the Page Editor
  - events can be sent
  - location for the edit controls can be activated to allow in-context editing
- Builds upon the Content Services API Endpoint concept

Page Component Differences

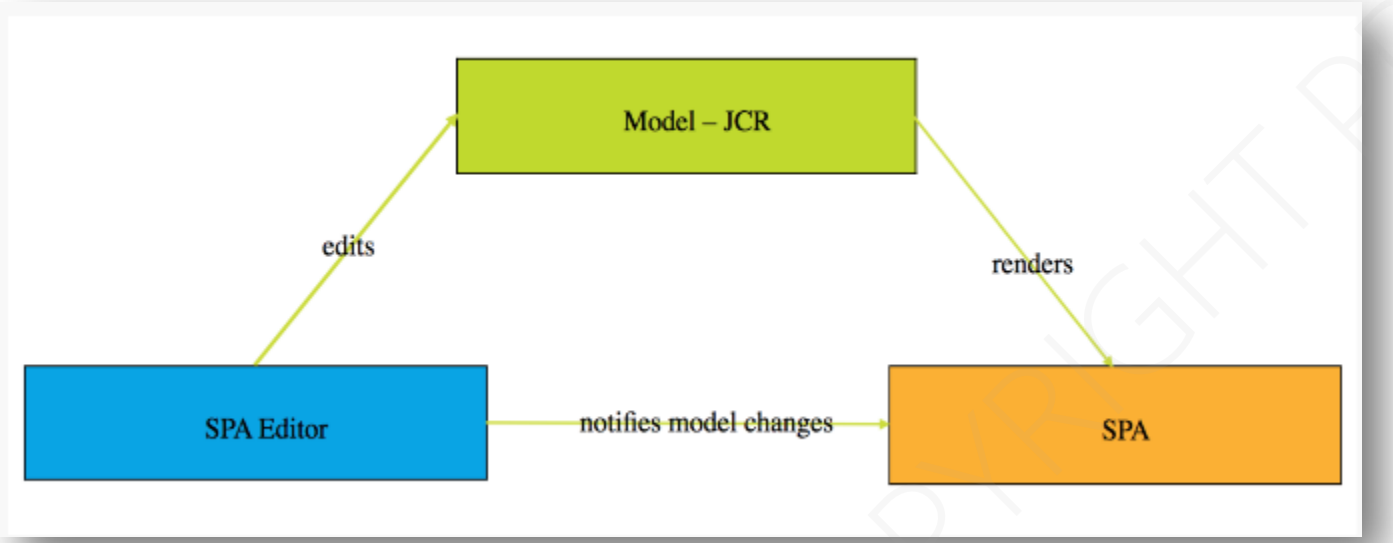
- Page component for a SPA doesn't provide the HTML elements of its child components via the JSP or HTL file
  - These operations delegated to the SPA framework
- Representation of child components or model fetched as a JSON data structure from the repository
- SPA components added to the page using this structure

Page Model Management (cont.)

- For each resource in the exported model the SPA maps an actual component which will do the
rendering.
- The model, represented as JSON, is then rendered using the component mappings within a container.



Communication Data Type

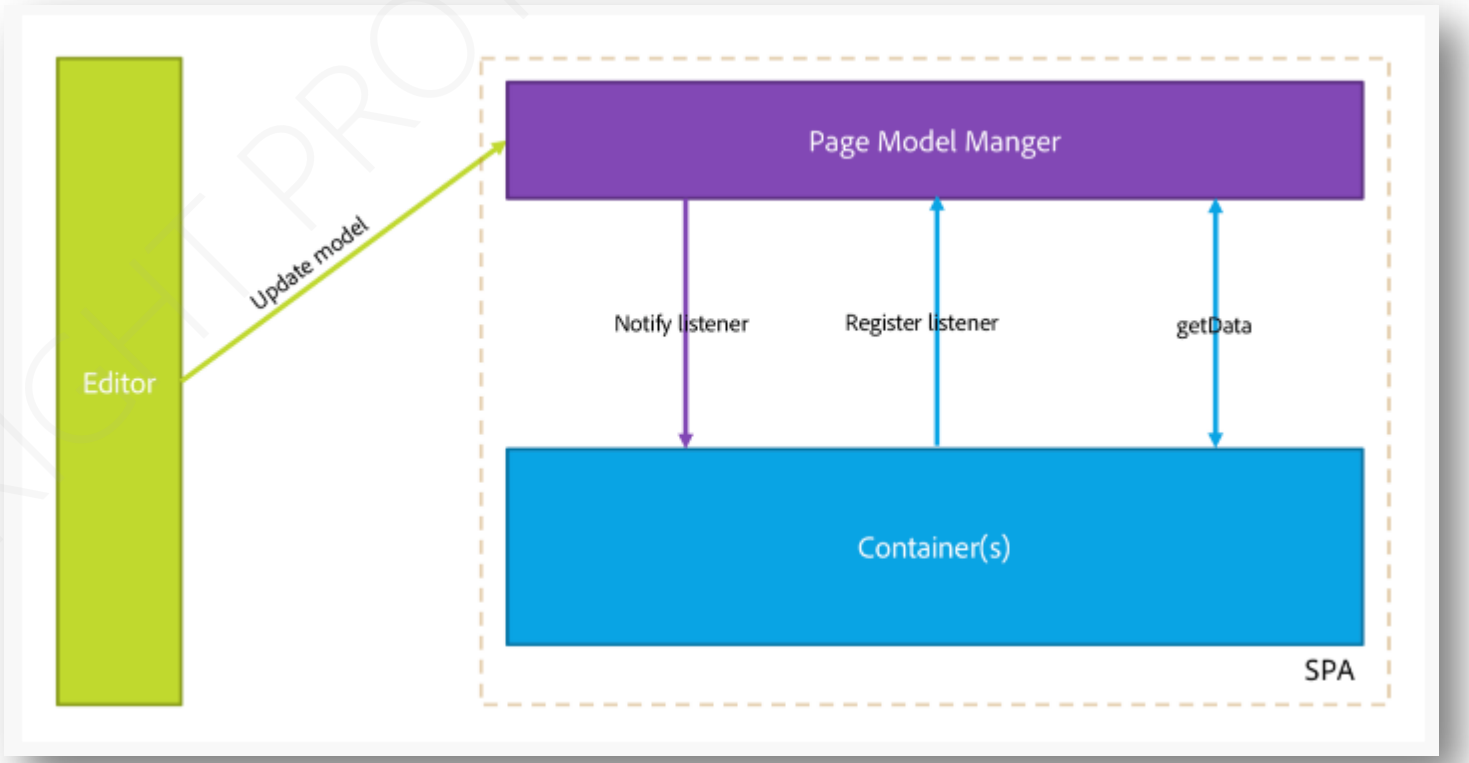


- cq.authoring.pagemodel.messaging client library sends a message to the Page Editor to establish the JSON communication data type
- When the communication data type is set to JSON, GET requests will communicate with the Sling Model end-points of a component.

- After an update occurs in the page editor, a JSON representation of the updated component is sent to the Page Model library
- Page Model library then informs the SPA of updates.

Flow of Interaction Between SPA and AEM

- SPA Editor is mediator
- Communication between Page Editor and SPA is JSON, not HTML
- Page Editor provides latest version of the page model to the SPA via iframe and messaging API
- Page Model Manager notifies editor that it is ready and passes page model as a JSON structure



Editor does not alter or access the DOM structure of the page being authored, rather it provides latest page model.

Exercise: Integrate SPA module.

Heart of the React App

- *Index.js* is the React app entry point and initiates connection to app CSS and the DOM
  - Initializes connection to AEM json data for the app
  - ModelManager is provided by the AEM SPA Editor JS SDK. It is responsible for calling and injecting the pageModel (the JSON content) into the application

- *App.js* handles the view generation and component inclusion:
  - Main app JavaScript component
  - Container for all other components
  - Extends React's Page class to define a page component

Integrate AEM SPA Editor JS SDK – *react-app/src/index.js*

index.js

index.js 〉 No Selection

```
23
24 import { Constants, ModelManager } from '@adobe/cq-spa-page-model-manager';
25 import { createBrowserHistory } from 'history';
26 import React from 'react';
27 import { render } from 'react-dom';
28 import { Router } from 'react-router-dom';
29 import App from './App';
30 import './components/import-components';
31 import './index.css';
32
33 document.addEventListener('DOMContentLoaded', () => {
34   ModelManager.initialize().then(pageModel => {
35     const history = createBrowserHistory();
36     render(
37       <Router history={history}>
38         <App
39           history={history}
40           cqChildren={pageModel[Constants.CHILDREN_PROP]}
41           cqItems={pageModel[Constants.ITEMS_PROP]}
42           cqItemsOrder={pageModel[Constants.ITEMS_ORDER_PROP]}
43           cqPath={pageModel[Constants.PATH_PROP]}
44           locationPathname={window.location.pathname}
45         />
46       </Router>,
47       document.getElementById('spa-root')
48     );
```

Initialize the App with AEM JSON Model

Integrate AEM SPA Editor JS SDK – *react-app/src/App.js*

App.js extends Page class from ***@adobe/cq-react-editable-components***

App.js

App.js 〉 No Selection

```
1  import { Page, withModel } from '@adobe/cq-react-editable-components';
2  import React from 'react';
3
4  // This component is the application entry point
5  class App extends Page {
6    render() {
7      return (
8        <div>
9          {this.childComponents}
10         {this.childPages}
11       </div>
12     );
13   }
14 }
15
16 export default withModel(App);
```

render() function ***this.childComponents*** and ***this.childPages*** include React components driven by JSON model

Integrate AEM SPA Editor JS SDK – *react-app/src/components/page/page.js*

Page.js — Edited

Page.js 〉 No Selection

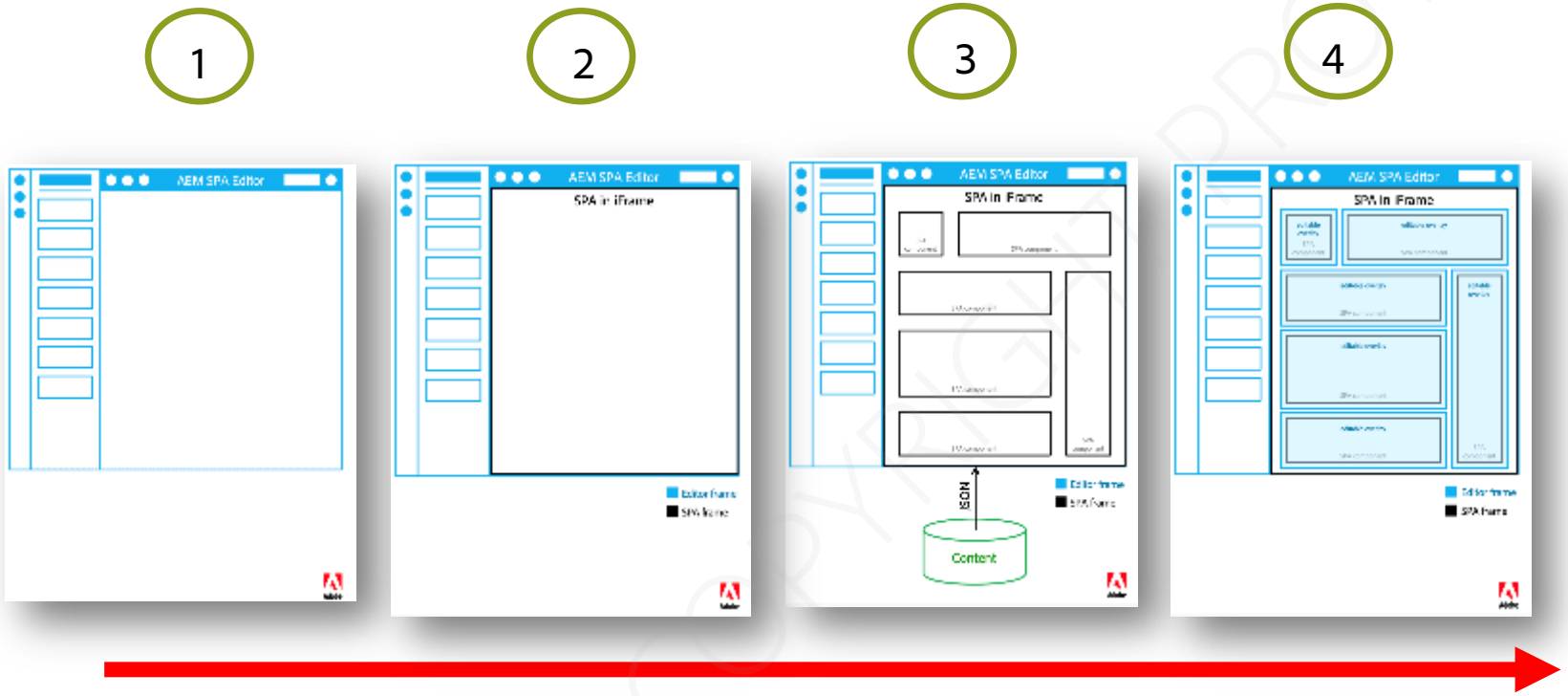
```
17 import {MapTo, Page, withComponentMappingContext}
18     from '@adobe/cq-react-editable-components';
19 import { withRoute } from '../RouteHelper/RouteHelper';
20
21 require('./Page.css');
22
23 // This component is a variant of a Page component mapped to the
24 // "wknd-spa-react/components/page" resource type. For now, the rendering is
25 // the same as the RootPage, except 1 illustrative injection of a unique
26 // class value, "WkndPage"
27 class AppPage extends Page {
28   get containerProps() {
29     let attrs = super.containerProps;
30     attrs.className =
31       (attrs.className || '') + ' WkndPage ' + (this.props.cssClassNames ||
             '');
32     return attrs;
33   }
34 }
35
36 export default MapTo('wknd-spa-react/components/page')(
37   withComponentMappingContext(withRoute(AppPage))
38 );
```

Map AEM resource type

wknd-spa-react/components/page

to a React component ***WKNDPage***

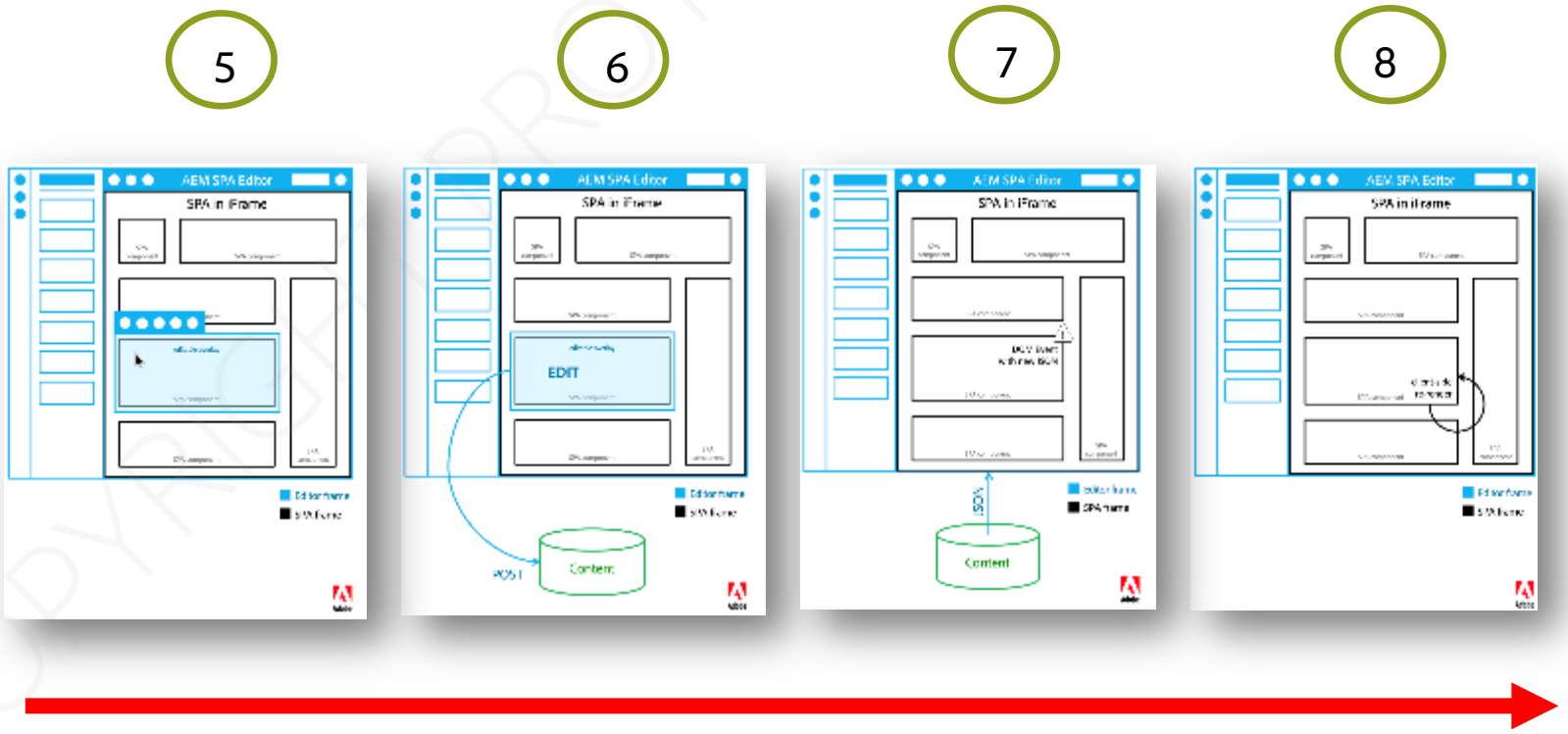
How Editing a SPA Appears to the Author



1. SPA Editor loads.
2. SPA is loaded in a separate frame.
3. SPA requests JSON content and renders components client-side.
4. SPA Editor detects rendered components and generates overlays.

How Editing a SPA Appears to the Author (cont.)

5. Author clicks overlay, displaying the component's edit toolbar.
6. SPA Editor persists edits with a POST request to the server.
7. SPA Editor requests updated JSON to the SPA Editor, which is sent to the SPA with a DOM Event.
8. SPA re-renders the concerned component, updating its DOM.



Integrating the React App on the Page – *customheaderlibs.html*

customheaderlibs.html

customheaderlibs.html 〉 No Selection

```
16 <sly data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html"
17      data-sly-call="${clientlib.css @ categories='wknd-spa-react.base'}"/>
18 <!--/*
19     SPA-specific meta tags
20 */-->
21 <meta name="theme-color" content="#000000" />
22 <link rel="icon"
      href="/etc
      .clientlibs/wknd-spa-react/clientlibs/clientlib-react/resources/favicon
      .ico" />
23 <link rel="apple-touch-icon"
      href="/etc
      .clientlibs/wknd-spa-react/clientlibs/clientlib-react/resources/logo192
      .png" />
24 <link rel="manifest"
      href="/etc
      .clientlibs/wknd-spa-react/clientlibs/clientlib-react/resources/manifest
      .json" />
25
26 <!-- AEM page model -->
27 <meta
28     property="cq:pagemodel_root_url"
29     data-sly-use.page="com.adobe.aem.spa.project.core.models.Page"
30     content="${page.hierarchyRootJsonExportUrl}"
31 />
```

Load ***wknd-spa-react*** client libraries

Used by AEM SPA editor to identify root page JSON

Integrating the React App on the Page – *customfooterlibs.html*

```
<!--/*
Custom footer React libs
*/-->
<sly data-sly-use.clientLib="${'/libs/granite/sightly/templates/clientlib.html'}"></sly>
<sly data-sly-test="${wcmmode.edit || wcmmode.preview}"
    data-sly-call="${clientLib.js @ categories='cq.authoring.pagemodel.messaging'}"></sly>
```

Allows SPA editing capabilities using the AEM Sites editor

Exercise: Add a Header Component

- Add a Header Component

Rapid Development Approaches

Proxy JSON

- Configure a Proxy between static React development server and AEM instance

Mock JSON

- Use a static or mock JSON file to simulate AEM responses

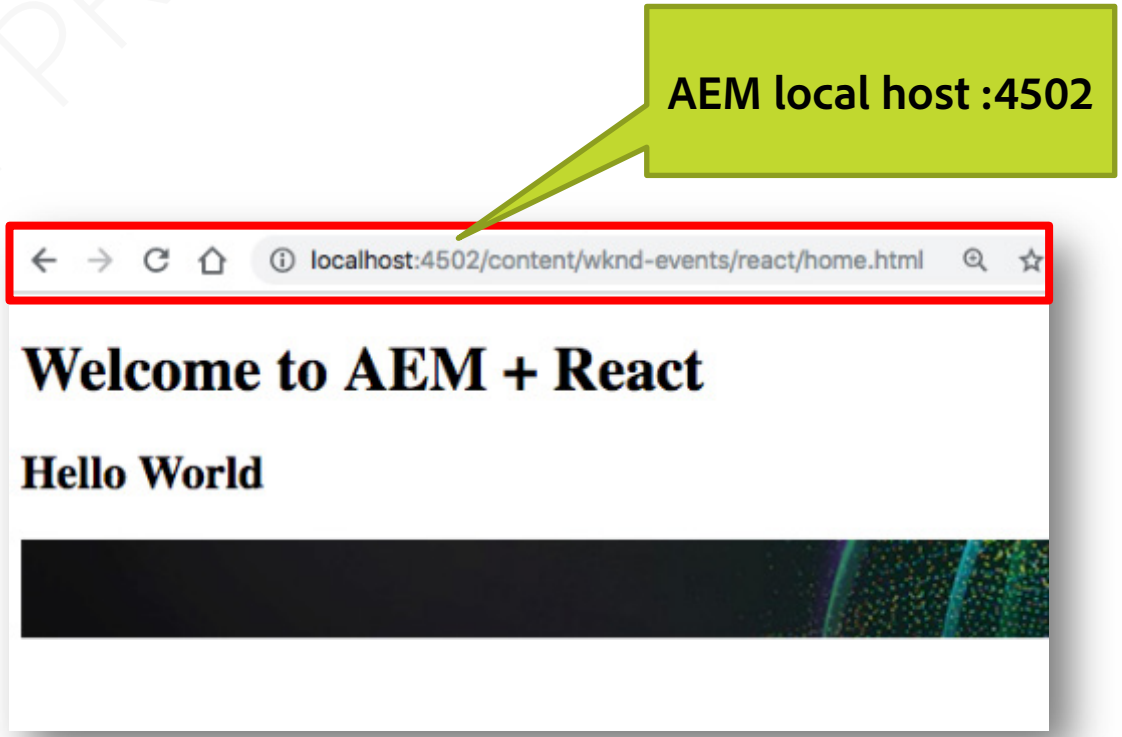
Proxy JSON Approach – *react-app/package.json*

Project automatically configures a proxy between the React development server and a local AEM instance.

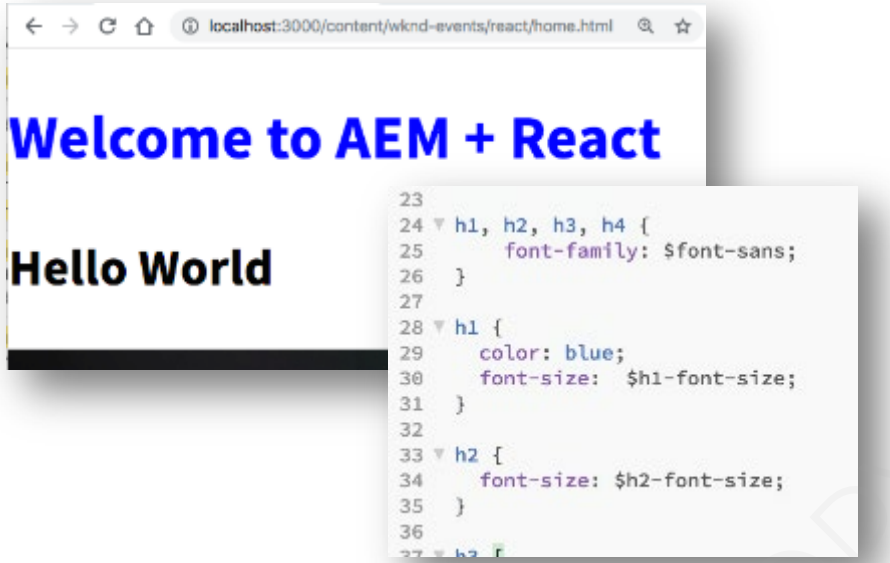
- React App uses proxy to server all unknown requests

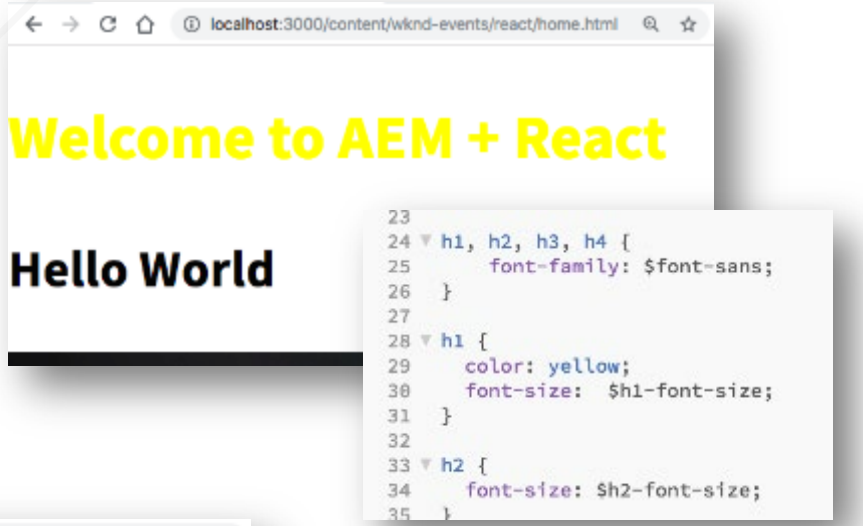
Requests to AEM content, for example the JSON Model and images, will be served as if the request(s) had originated on the Webpack development server

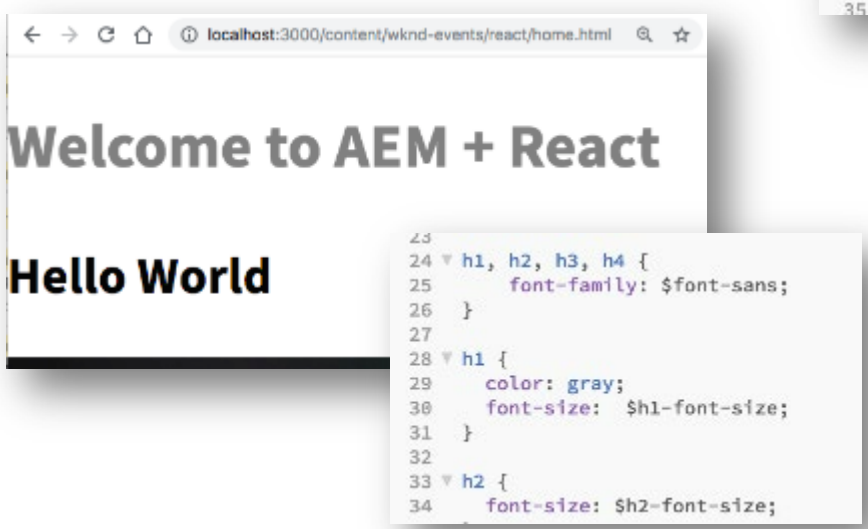
AEM Successfully Proxied



Make JS or Style Changes Outside of AEM





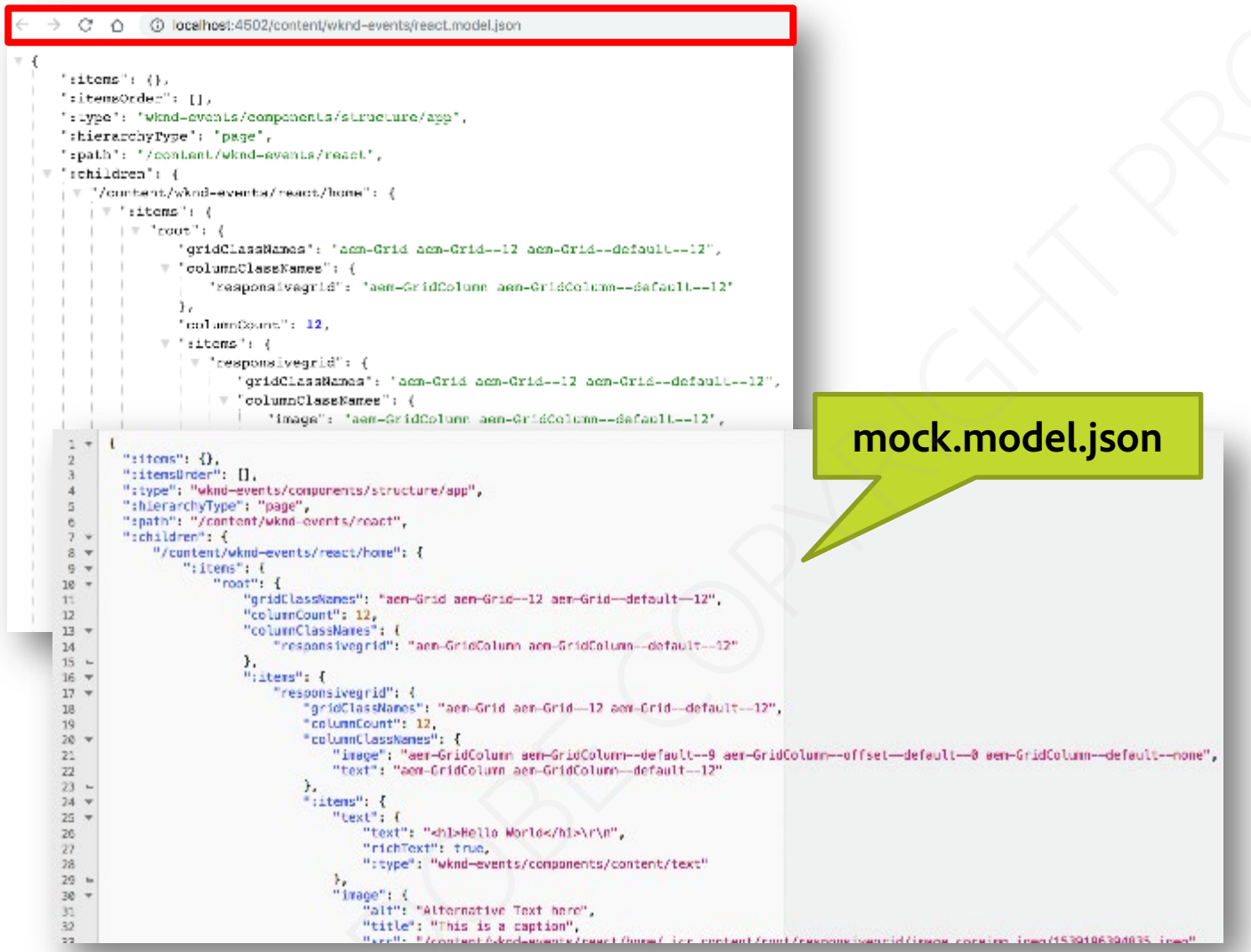


Update client files in *react-app/src*

Exericse: Proxy JSON Approach

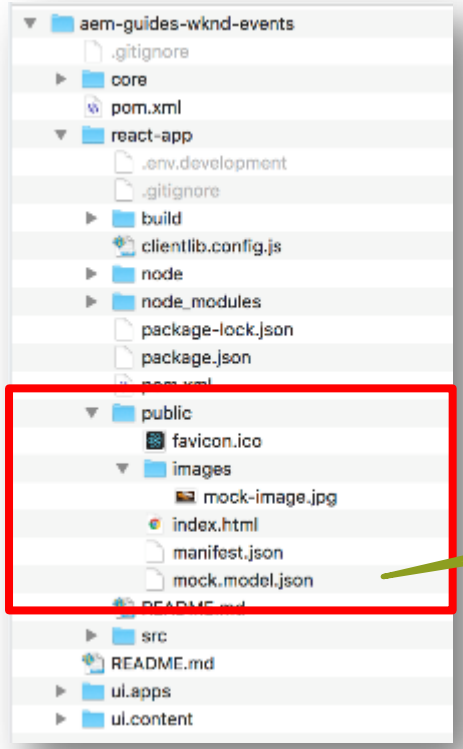
- Proxy JSON Approach

Mock JSON Approach – *public/mock-content/mock.model.json*

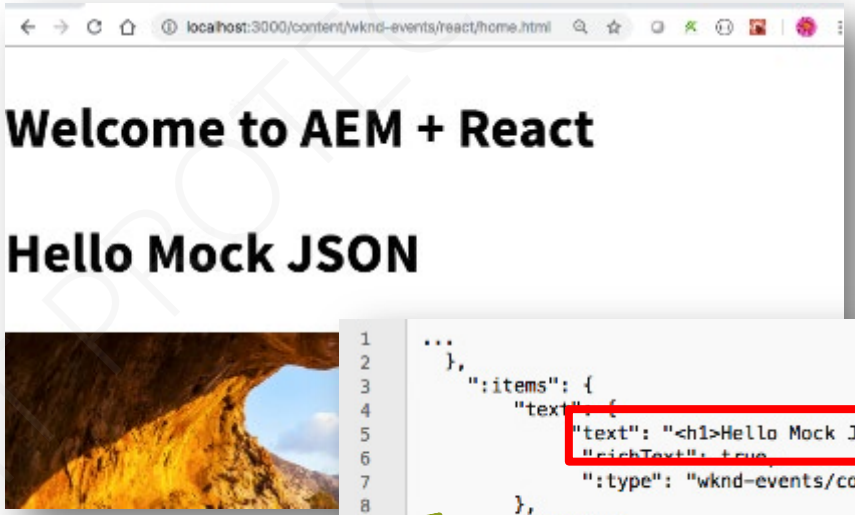


- Use a static or mock JSON file
- Removes dependency on a local AEM instance
- Allows the Front End developer to update the JSON
  - test functionality
  - mock new JSON responses without dependency on a Back End developer

Mock AEM Responses for Testing

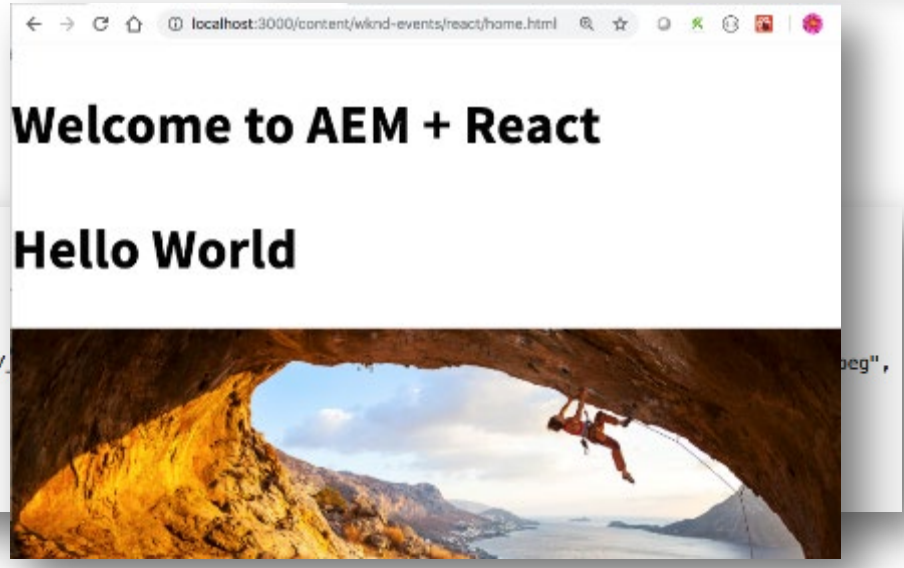


mock.model.json




```
...
  },
    ":items": {
        "text": {
            "text": "<h1>Hello Mock JSON</h1>\r\n",
            "richText": true,
            ":type": "wknd-events/components/content/text"
        },
        "image": {
            "alt": "Alternative Text here",
...
```

```
...
{
    "image": {
        "alt": "Alternative Text here",
        "src": "/images/mock-image.jpg",
        "lazyEnabled": false,
        "widths": [],
        ":type": "wknd-events/components/content/image"
    }
},
...
```



Exercise: Mock JSON Approach

Module 4

Map SPA Components

AEM SPA Editor JS SDK

- Collection of JS libraries
- Framework for editing contents of a SPA deployed in AEM
- AEM delivers content as JSON and SPA Editor JS SDK maps the JSON to React components
- Available via five npm modules:
  - @adobe/aem-spa-component-mapping - provides helpers to map AEM Components to SPA components. This module is not tied to a specific SPA framework.
  - @adobe/aem-spa-page-model-manager - provides the API to manage the model representation of the AEM Pages that are used to compose a SPA. This module is not tied to a specific SPA framework.
  - @adobe/aem-react-editable-components - provides generic React helpers and components to support AEM authoring. This module also wraps the cq-spa-page-model-manager and cq-spa-component-mapping to make these available to the React framework.
  - @adobe/aem-core-components-react-base & @adobe/aem-core-components-react-spa -set of re-usable UI components that map to out of the box AEM components. These are designed to be used as is and styled to meet your project's needs

AEM Page Editor and the SPA

- Goal:
  - Edits the data exposed by a Single Page Application (SPA) framework

- Requirement:
  - Project must be able to interpret the structure of the data stored for an application within the AEM repository

- Solution:
  - Two framework-agnostic libraries are provided:
    - **PageModelManager**
    - **ComponentMapping**

Page Model

- Content structure of a page is stored in AEM
- Model of the page is used to map & instantiate SPA components
- Leverages the JSON Model Exporter for a Page built in AEM
- SPA developers create SPA components which they map to AEM components
  - Use the *resource type* (or path to the AEM component) as a unique key
- SPA components must be in synch with the page model and be updated with any changes to its content

ComponentMapping

- Library provided as an NPM package
- Stores front-end components
- Provides a way for the SPA to map front-end components to AEM resource types:
  - Enables a dynamic resolution of components when parsing the JSON model of the application
  - Each item present in the model contains a **:type** field that exposes an AEM resource type
- Front-end component can render itself using the fragment of model it has received from the underlying libraries

Meta fields

Exportable sling models expose the following fields in order to enable the underlying libraries interpret the data model:

- **:type**: Type of the AEM resource (default = resource type)
- **:children**: Hierarchical children of the current resource. Children are not part of the inner content of the current resource (can be found on items representing a page)
- **:hierarchyType**: Hierarchical type of a resource. The **PageModelManager** currently supports the page type
- **:items**: Child content resources of the current resource (nested structure, only present on containers)
- **:itemsOrder**: Ordered list of the children. The JSON map object doesn't guaranty the order of its fields. By having both the map and the current array the consumer of the API has the benefits of both structures
- **:path**: Content path of an item (present on items representing a page)

(Review) – *ui.frontend/src/App.js*

App.js extends Page class from ***@adobe/cq-react-editable-components***

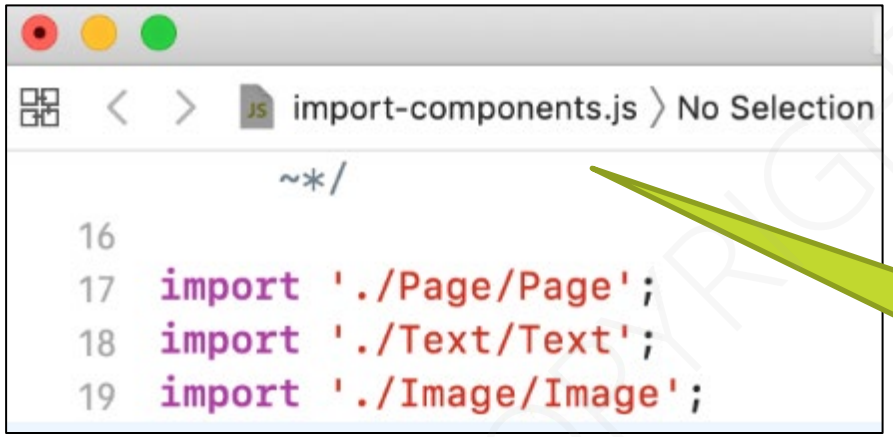
App.js

App.js 〉 No Selection

```
1  import { Page, withModel } from '@adobe/cq-react-editable-components';
2  import React from 'react';
3
4  // This component is the application entry point
5  class App extends Page {
6    render() {
7      return (
8        <div>
9          {this.childComponents}
10         {this.childPages}
11       </div>
12     );
13   }
14 }
15
16 export default withModel(App);
```

render() function ***this.childComponents*** and ***this.childPages*** include React components driven by JSON model

Integrate AEM SPA Editor JS SDK – *src/components/import-components.js*



A dedicated file to identify all React components that map to an AEM component

Text Component

Sets up a default editConfig for the Text component

AEM properties available as React 'props'

Maps the AEM Text component to a React component that allows rich text authoring as part of the app

Text.js — Edited

Text.js 〉 No Selection

```
21  import { MapTo } from '@adobe/cq-react-editable-components';
22  require('./Text.scss');
23
24  const TextEditConfig = {
25    emptyLabel: 'Text',
26
27    isEmpty: function(props) {
28      return !props || !props.text || props.text.trim().length < 1;
29    }
30  };
31
32  class Text extends Component {
33    get richTextContent() {
34      return (
35        <div
36          id={extractModelId(this.props.cqPath)}
37          data-rte-editelement
38          dangerouslySetInnerHTML={{
39            __html: DOMPurify.sanitize(this.props.text)
40          }}
41        />
42      );
43    }
44
45    get textContent() {
46      return <div>{this.props.text}</div>;
47    }
48
49    render() {
50      return this.props.richText ? this.richTextContent : this.textContent;
51    }
52  }
53
54  export default MapTo('wknd-spa-react/components/text')(
```

Text Component Model JSON Export

MapTo functionality exposes the JSON values of Text component properties

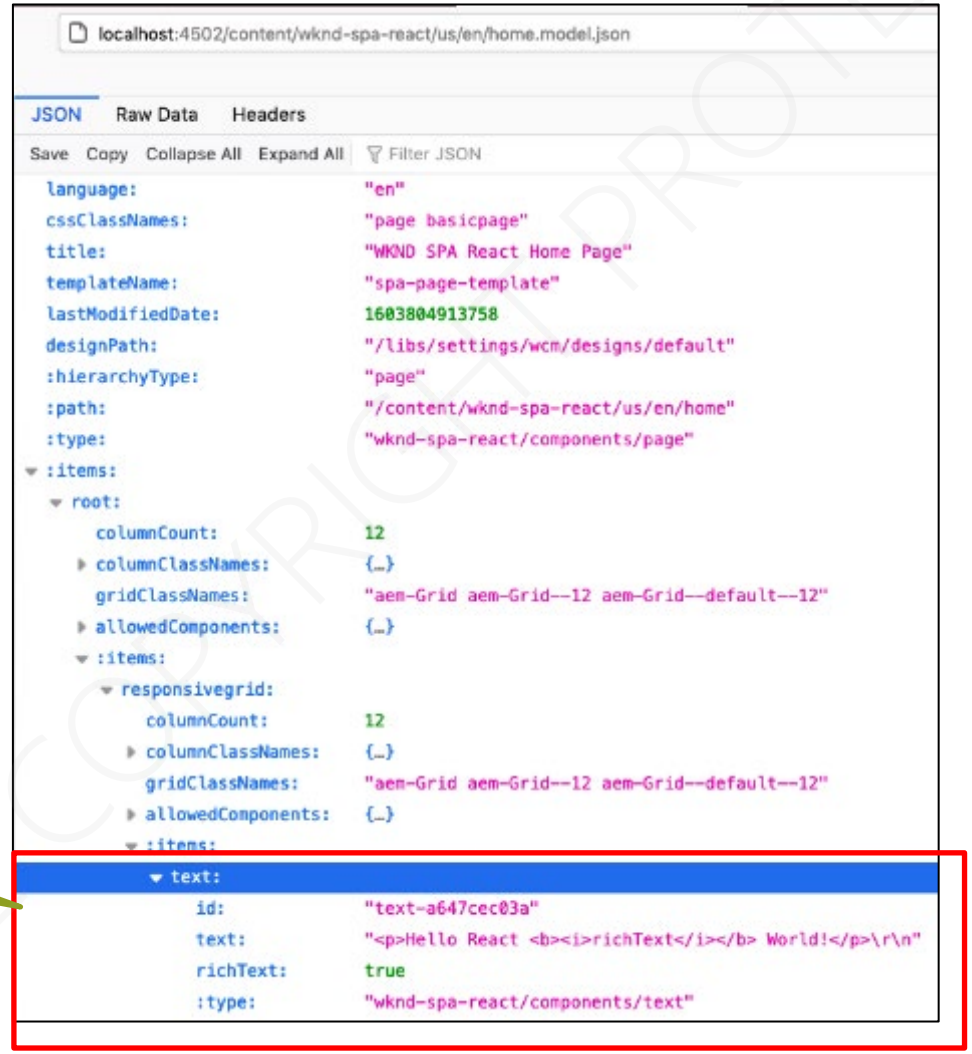


Image Component

Set up a default editConfig for the Image component

AEM properties available as React 'props'

Map the AEM Image component to a React component that allows users to include an image in the app

Image.js

Image.js 〉 No Selection

```
6  export const ImageEditConfig = {
7
8      emptyLabel: 'Image',
9
10     isEmpty: function(props) {
11         return !props || !props.src || props.src.trim().length < 1;
12     }
13 };
14
15 export default class Image extends Component {
16
17     get content() {
18         return <img
19                 className="Image-src"
20                 src={this.props.src}
21                 alt={this.props.alt}
22                 title={this.props.title ? this.props.title : this.props.alt}
23                     />;
24     }
25
26     render() {
27         if(ImageEditConfig.isEmpty(this.props)) {
28             return null;
29         }
30
31         return (
32                 <div className="Image">
33                     {this.content}
34                 </div>
35         );
36     }
37 }
38
39 MapTo('wknd-spa-react/components/image')(Image, ImageEditConfig);
```

Image Component Model JSON Export

MapTo functionality exposes the JSON values of alt, title, src that can then be used by the React Image component.

localhost:4502/content/wknd-events/react/home.model.json

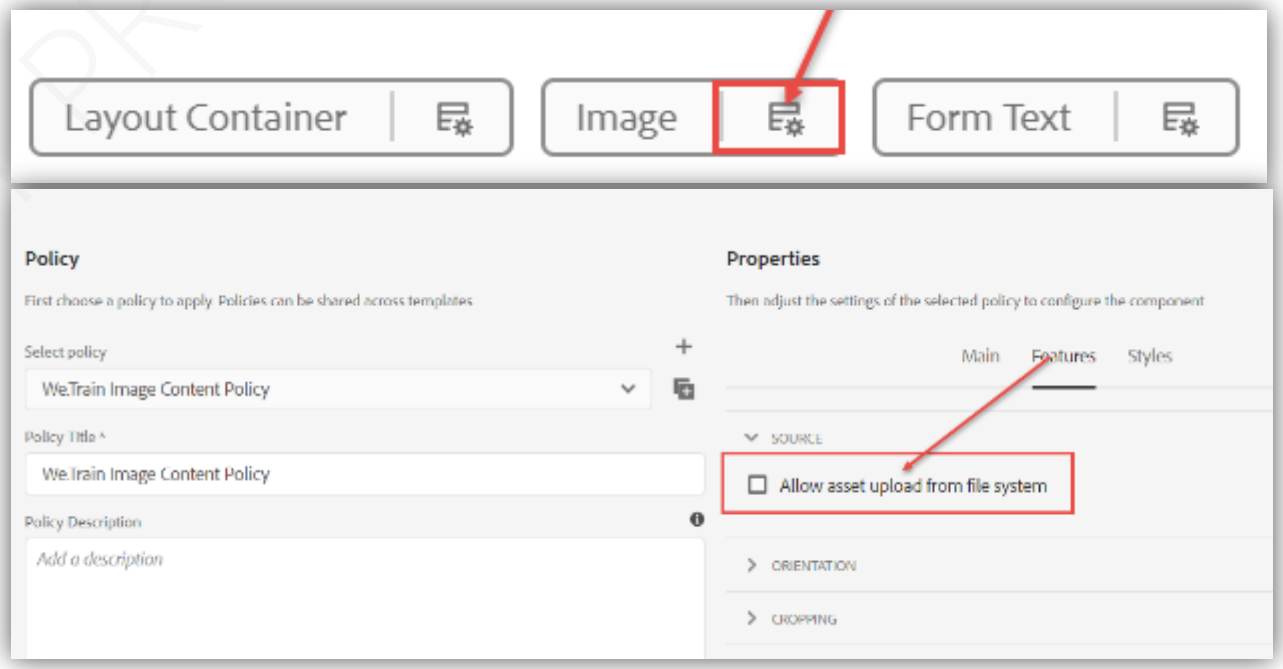

```
"gridClassNames": "aem-Grid aem-Grid--12 aem-Grid--default--12",
":items": {
  "responsivegrid": {
    "columnCount": 12,
    "columnClassNames": {
      "image": "aem-GridColumn aem-GridColumn--default--12",
      "text": "aem-GridColumn aem-GridColumn--default--12"
    },
    "gridClassNames": "aem-Grid aem-Grid--12 aem-Grid--default--12",
    ":items": {
      "text": {
        "text": "<h2><b>Hello World</b></h2>\r\n",
        "richText": true,
        ":type": "wknd-events/components/content/text"
      },
      "image": {
        "alt": "Arctic Surfing in Lofoten, Northern Norway, Europe",
        "title": "Northern Lights",
        "src": "/content/wknd-
        events/react/home/_jcr_content/root/responsivegrid/image.coreimg.jpeg/1545147707682.jpeg",
        "srcUriTemplate": "/content/wknd-
        events/react/home/_jcr_content/root/responsivegrid/image.coreimg{.width}.jpeg/1545147707682.jpeg",
        "areas": [],
        "uuid": "24be0df2-92ae-4cd5-91f2-ca5a7fbf3dff",
        "widths": [],
        "lazyEnabled": false,
        ":type": "wknd-events/components/content/image"
      }
    },
```

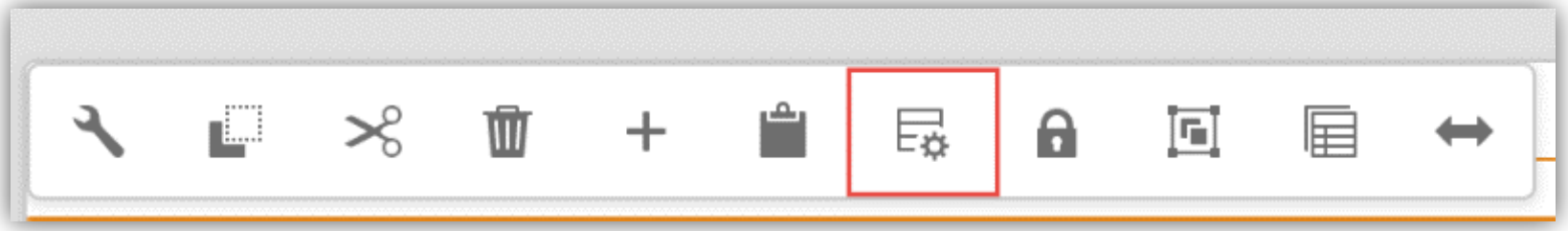
Exercise: Inspect the Text Component

Content Policies

- Global content stored in **conf/<project>/settings/wcm/policies/**
  - component/relative/path/policy_<UID>
  - Effect the options available to Authors

- Can be accessed through:
  - The template editor
  - Structure mode
  - Content components in the Layout Container

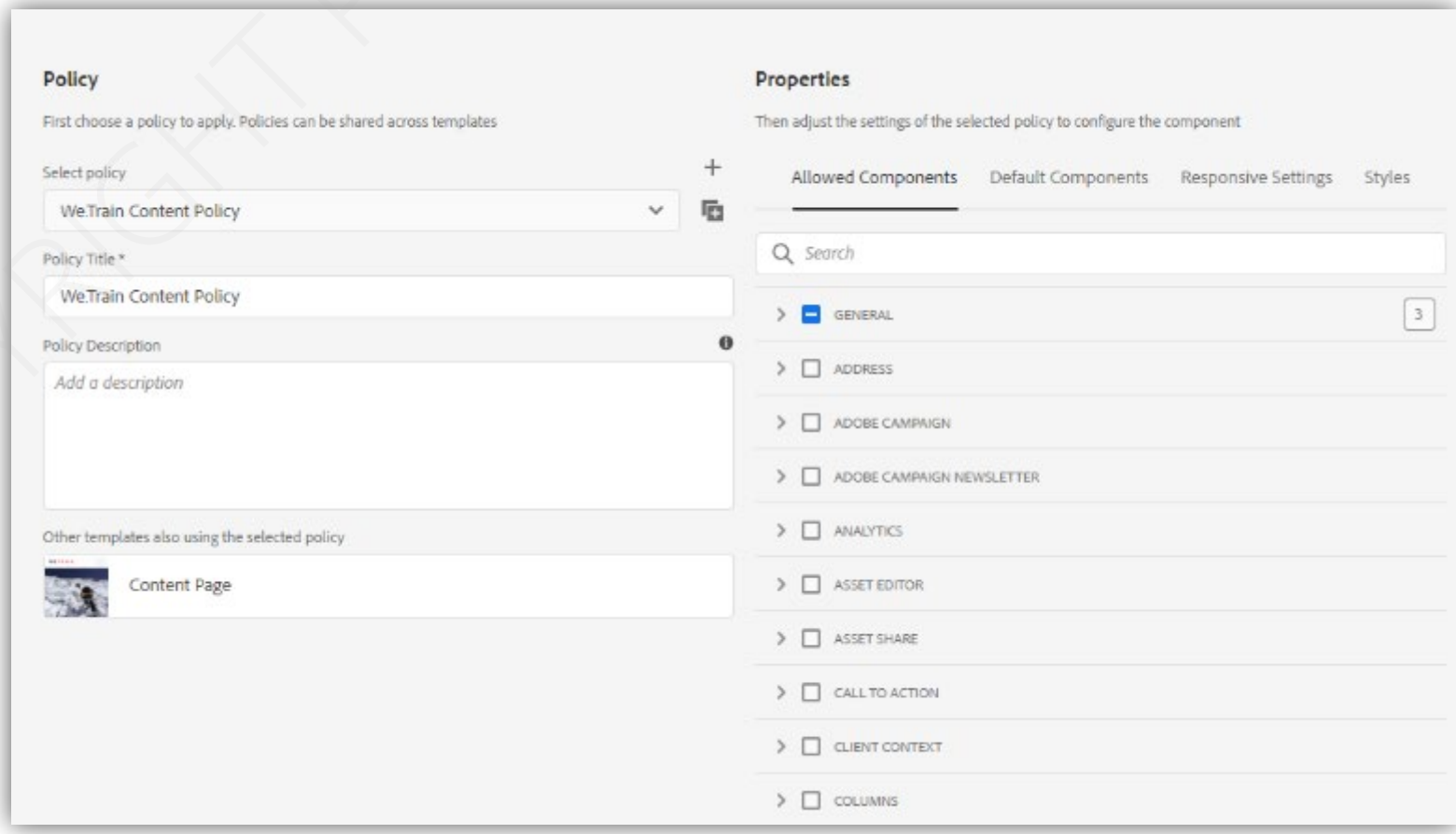




Core Component Policies

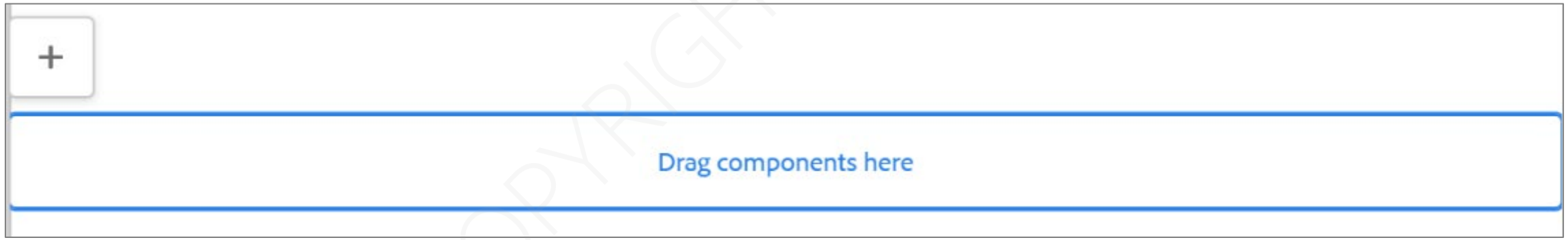
Component content policies:

- Are highly configurable
- Reusable between templates
- Can be configured per Layout Container
- Are fully extensible



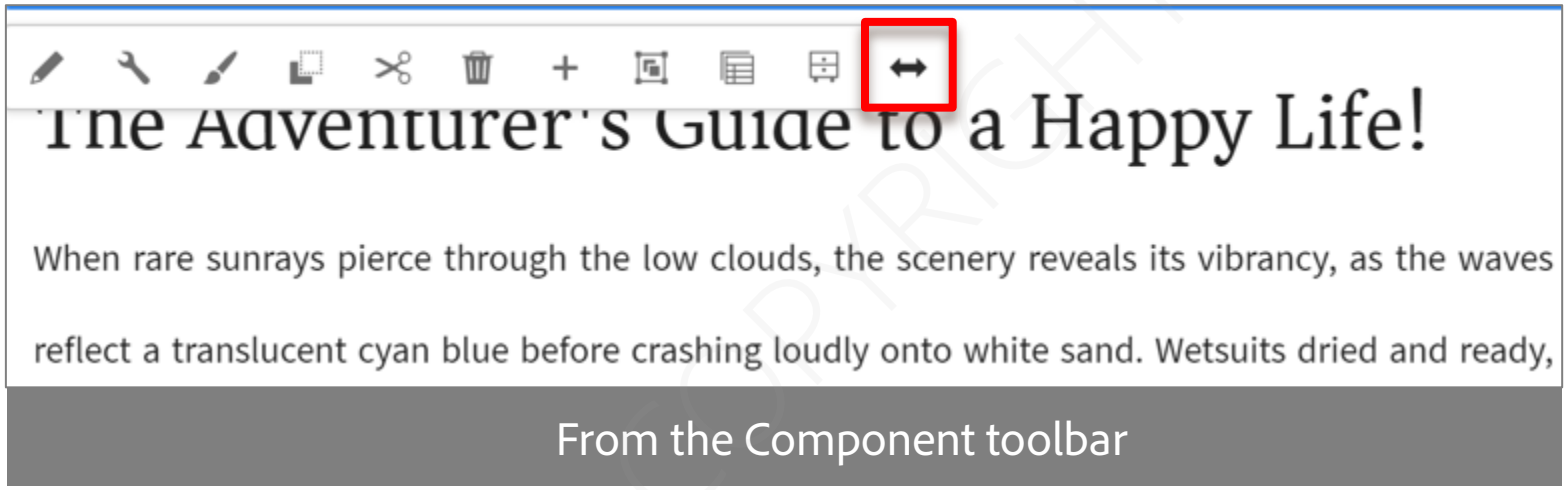
Layout Container Component

- Contains other content components
- Creates a graphical area to add other components

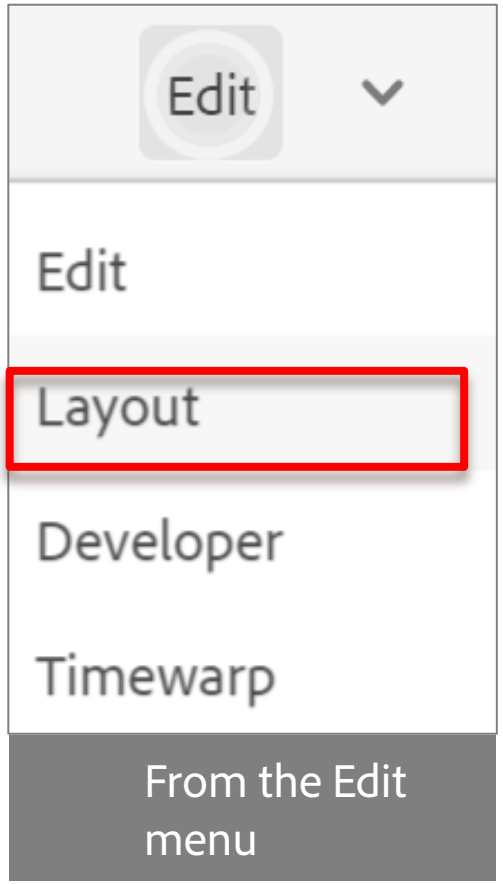


Access Layout Mode

Access the component layout in two ways:



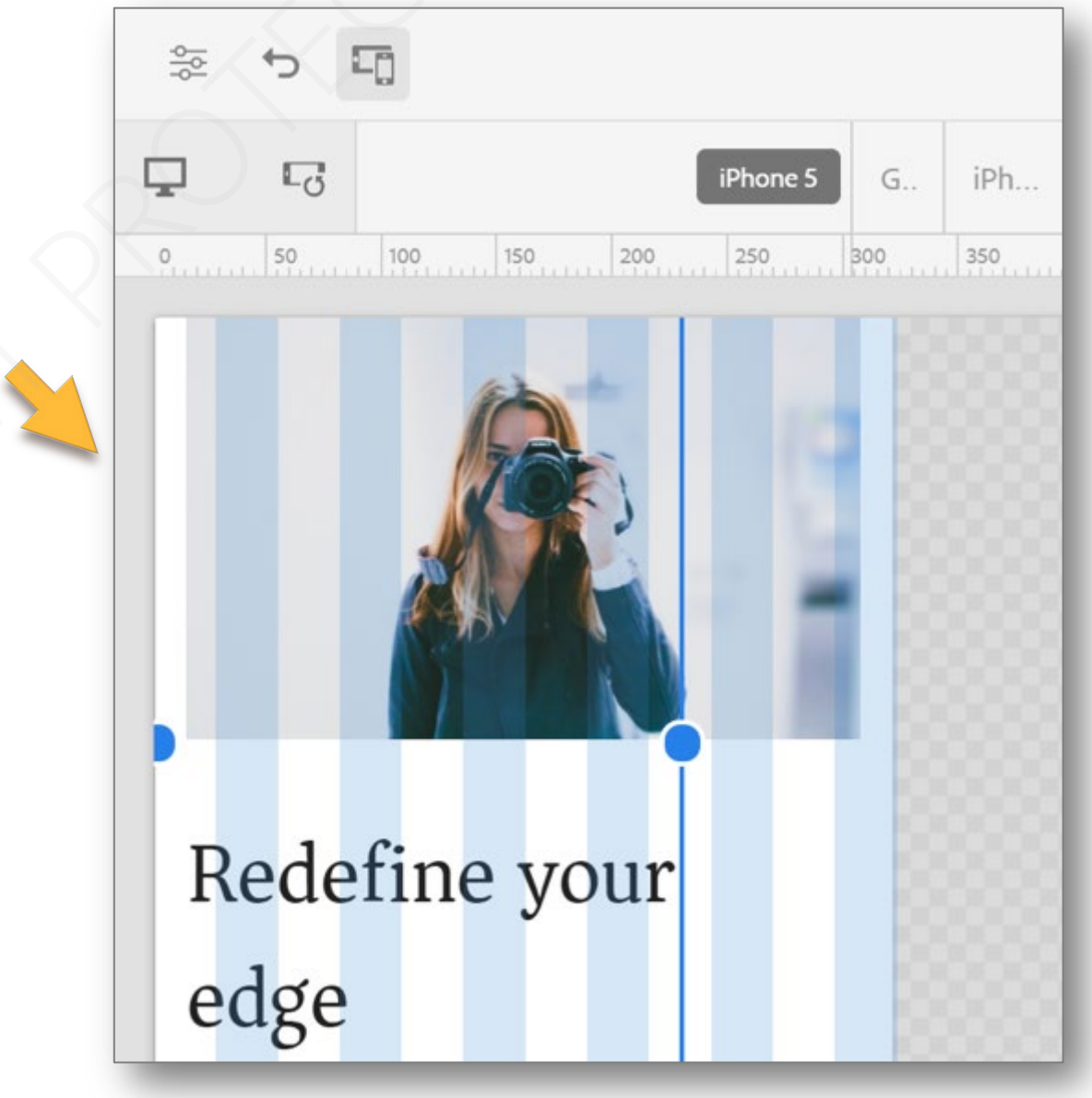
From the Component toolbar



From the Edit menu

Modify the Component Layout

- Use the blue drag circles to adjust the component width on a page
- Choose various layouts per screen



Exercise: Update Policies in AEM

Module 5

Navigation and Routing

Routing in a Single Page Application

Routing enables navigating the app through links

Behavior that is expected in normal web applications:

- Browser should change the URL when you navigate to a different screen
- Deep linking should work
  - Clicking on a hyperlink to the URL for a page on the website, should render that page, not the website/application home page
- Browser back and forward buttons should work as expected

Routing links your application navigation with the navigation features offered by the browser:

- address bar
- navigation buttons

SPA Routing Approach

- Map different views within the SPA to respective pages within AEM
  - Easy to manage multiple parts of the application
  - Content authors can edit individual views

```
1   // psuedo code in JSX to represent the Routing approach
2
3   <BrowserRouter>
4       <App>
5           <Route path="/content/wknd-events/react/home">
6               <WkndPage cqPath="/content/wknd-events/react/home" />
7           </Route>
8           <Route path="/content/wknd-events/react/home/child1">
9               <WkndPage cqPath="/content/wknd-events/react/home/child1" />
10          </Route>
11          <Route path="/content/wknd-events/react/home/child2">
12              <WkndPage cqPath="/content/wknd-events/react/home/child2" />
13          </Route>
14      </App>
15  </BrowserRouter>
```

<Route /> elements for each page shown in the JSX pseudo code

- Each AEM Page represented in the App will be wrapped in a <Route> with a path of the AEM page

Meta Properties

cq:wcmmode: WCM mode of the editors (e.g. edit, preview)

cq:pagemodel_root_url: URL of the root model of the App

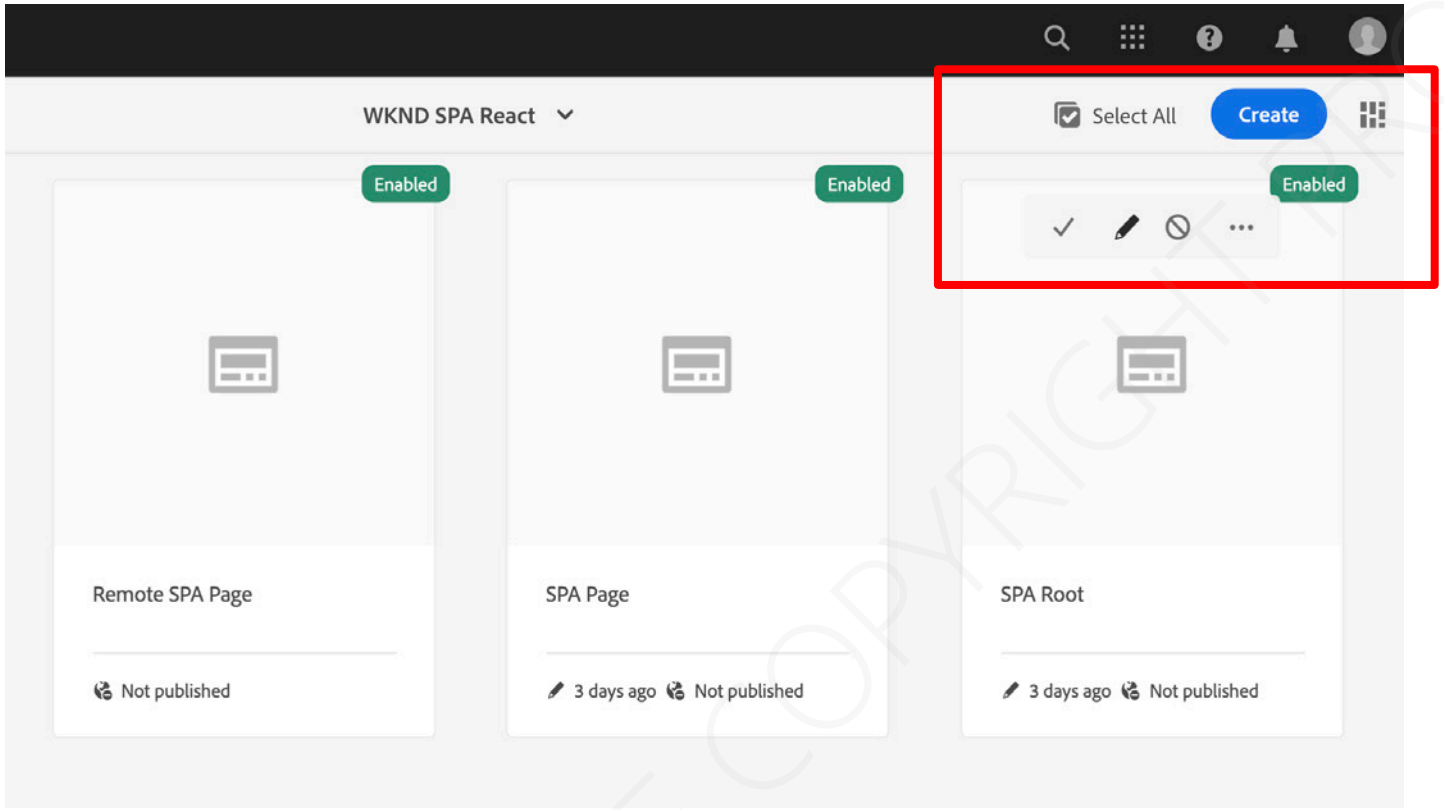
cq:pagemodel_router: Enable or disable the ModelRouter of the PageModelManager library

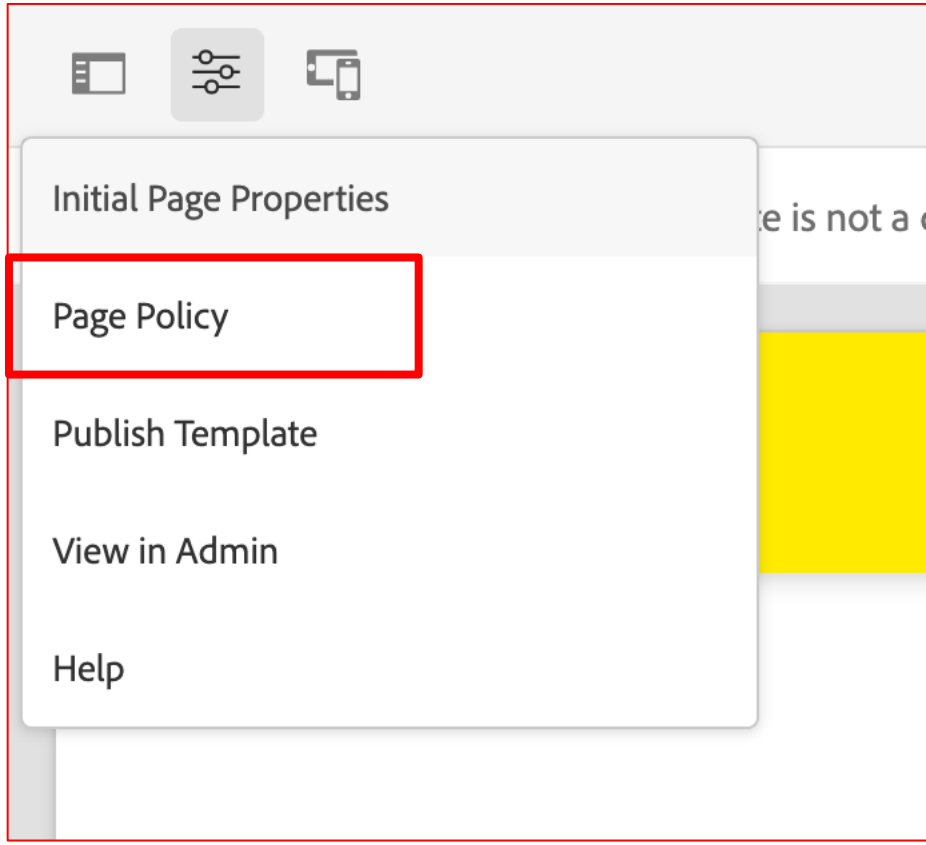
cq:pagemodel_route_filters: Comma separated list or regular expressions to provide routes that the ModelRouter must ignore.

Sling Model JSON Exported Structure Configuration

- When routing is enabled, assume that the JSON export of the SPA contains the different routes of the application
  - Due to the JSON export of the AEM navigation component
- JSON output of the AEM navigation component can be configured in the SPA's root page content policy through the following two properties:
  - **structureDepth**: Number corresponding to the depth of the tree exported
  - **structurePatterns**: Regex of array of regexes corresponding to the page to be exported
- These are set in the Policies of the SPA Root editable template

Setting exported Sling Model data via SPA Root Template





React Router

- Standard routing library for React
  - Collection of navigational components
  - Provide and manage different views of the application
- Keeps UI in sync with the URL
- Simple API with powerful features built in:
  - lazy code loading
  - dynamic route matching
  - location transition handling

React Router Modules

Library of 3 Modules:

- **react-router:** core package for the router
- **react-router-dom:** DOM bindings for React Router, router components for the web app development environment ***
- **react-router-native:** native bindings for React Router, router components for an app development environment using React Native

*** since this app will be deployed to the web, we will be using ***react-router-dom***

Routing Components - */src/components/RouteHelper/RouteHelper.js*

RouteHelper.js — Edited

RouteHelper.js 〉 No Selection

```
17 import React, { Component } from 'react';
18 import { Route } from 'react-router-dom';
19
20 export const withRoute = (WrappedComponent, extension) => {
21   return class CompositeRoute extends Component {
22     render() {
23       let routePath = this.props.cqPath;
24       if (!routePath) {
         return <WrappedComponent {...this.props} />;
       }

       extension = extension || 'html';

       // Context path + route path + extension
       return (
         <Route
           key={routePath}
           exact
35         path={'(.*)' + routePath + '(.' + extension + ')?'}
36         render={routeProps => {
37           return <WrappedComponent {...this.props} {...routeProps} />;
38         }}
39       />
```

import {Route}

withRoute **can wrap any React component. Primarily used to wrap Page component to provide routing between pages**

our React Page component will be returned as a <Route> object, allowing use of <Link> and History retention

Wrap AppPage component - */src/components/page/Page.js*

Page.js

Page.js 〉 No Selection

```
17 import {
18   MapTo,
19   Page,
20   withComponentMappingContext
21 } from '@adobe/cq-react-editable-components';
22 import { withRoute } from '../RouteHelper/RouteHelper';
23
24 require('./Page.css');
25
26 // This component is a variant of a Page component mapped to the
27 // "wknd-spa-react/components/page" resource type. For now, the rendering is
28 // the same as the RootPage; this is more for illustration purposes
29 class AppPage extends Page {
30   get containerProps() {
31     let attrs = super.containerProps;
32     attrs.className =
33       (attrs.className || '') + ' page ' + (this.props.cssClassNames || '');
34     return attrs;
35   }
36 }
37
38 export default MapTo('wknd-spa-react/components/page')(
39   withComponentMappingContext(withRoute(AppPage))
40 );
```

Import withRoute

Wrap AppPage via withRoute in RouteHelper.js

Wrapping the App - */src/index.js*

index.js

index.js 〉 No Selection

```
28 import { Router } from 'react-router-dom';
29 import App from './App';
30 import './components/import-components';
31 import './index.scss';
32
33 document.addEventListener('DOMContentLoaded', () => {
34   ModelManager.initialize().then(pageModel => {
35     const history = createBrowserHistory();
36     render(
37       <Router history={history}>
38         <App
39           history={history}
40           cqChildren={pageModel[Constants.CHILDREN_PROP]}
41           cqItems={pageModel[Constants.ITEMS_PROP]}
42           cqItemsOrder={pageModel[Constants.ITEMS_ORDER_PROP]}
43           cqPath={pageModel[Constants.PATH_PROP]}
44           locationPathname={window.location.pathname}
45         />
46       </Router>,
47       document.getElementById('spa-root')
48     );
49   });
50 });
```

wrap the **App** with **Router** to track history

Routing Review

- AEM resource wknd-spa-react/components/page
  - Represents an AEM Page
  - Will be mapped to the React component AppPage
- Addition of **withRoute** causes all pages to be wrapped as a **Route**
  - You can now navigate to any page

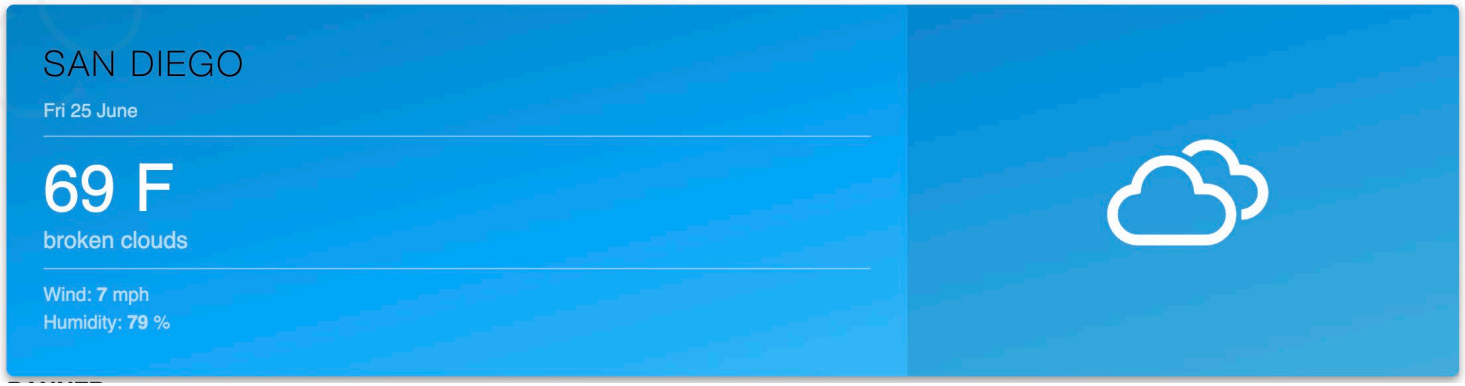
Exercise: Implement Routing & Navigation

Module 6

Create a Custom Component

Creating a Custom Component

- Creating a net-new component from scratch
- Objectives
  - Understand the role of Sling Models in manipulating JSON Models provided by AEM
  - Understand how to create new AEM component dialogs
  - Learn to create a custom AEM component that will be compatible with the SPA editor framework
- What you will build:



Create the .content.xml under ui.apps

- jcr:primaryType="cq:Component" - identifies that this node will be an AEM component
- jcr:title is the value that will be displayed to Content Authors
- componentGroup determines the grouping of components in the authoring UI

ui.apps > src > main > content > jcr_root > apps > wknd-spa-react > components > open-weather > .content.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0"
3      jcr:primaryType="cq:Component"
4      jcr:title="Open Weather"
5      componentGroup="WKND SPA React - Content"/>
6
```

Create the .content.xml for the component's dialog

- Inner <message> XML node contains the required textfield widgets
- These textfields allow the author to edit the content via the component
- This label textfield persists the authored value to a property named label for later retrieval by the component on the page

```
<items jcr:primaryType="nt:unstructured">
    <label
        jcr:primaryType="nt:unstructured"
        sling:resourceType="granite/ui/components/coral/foundation/form/textfield"
        fieldDescription="The label to display for the component"
        fieldLabel="Label"
        name="./label"/>
```

Create the Sling Model

- Sling models are annotation driven POJOs that allow JCR Data to be mapped to java variables
- Encapsulates business logic for AEM components
- Exposes a component's content via a JSON Model through a Sling Model Exporter
- Leverages an interface pattern:

```
public class OpenWeatherModelImpl implements OpenWeatherModel {

    @ValueMapValue
```

Implements the OpenWeatherModel interface

```
// public getter method to expose value of private variable `label`
// adds additional logic to default the label to "(Default)" if not set.
@Override
public String getLabel() {
    return StringUtils.isNotBlank(label) ? label : "(Default)";
}
```

getLabel() exposes the value of the author dialog through the JSON model.

Create the Sling Model

- OpenWeatherModelImpl implements the OpenWeather interface
  - @Model identifies the Java class as a Sling Model
  - @Exporter enables the Java class to be serialized and exported through the Sling Model Exporter.

```java
public class OpenWeatherModelImpl implements OpenWeatherModel {

    @ValueMapValue
    private String label; //maps variable to jcr property named "label" persisted by Dialog

    @ValueMapValue
    private double lat; //maps variable to jcr property named "lat"

    @ValueMapValue
    private double lon; //maps variable to jcr property named "lon"

    // points to AEM component definition in ui.apps
    static final String RESOURCE_TYPE = "wknd-spa-react/components/open-weather";

    // public getter method to expose value of private variable `label`
    // adds additional logic to default the label to "(Default)" if not set.
    @Override
    public String getLabel() {
        return StringUtils.isNotBlank(label) ? label : "(Default)";
    }
```

Relative RESOURCE_TYPE path identifying the component.

Implements the getLabel method to return the authored message content.

Update the React component

- The react custom component code responsible for rendering the content on the page
- Uses the returned JSON provided by the Sling Model Exporter to get the message property

```
    return (
        <div className="cmp-open-weather">
            <ReactWeather
                isLoading={isLoading}
                errorMessage={errorMessage}
                data={data}
                lang="en"
                locationLabel={props.label} // passed in from AEM JSON
                unitsLabels={{ temperature: 'F', windSpeed: 'mph' }}
                showForecast={false}
            />
        </div>
    );
}
```

Properties returned from sling model JSON mapped to message in the react component

Maps our React component to the custom AEM component

```
// Map OpenWeather to AEM component
MapTo('wknd-spa-react/components/open-weather')(OpenWeather, OpenWeatherEditConfig);
```

Custom Component Review

- React component maps to wknd-spa-react/components/open-weather resource and renders the message property
- AEM component created via cq:component and dialog .content.xml under ui.apps
- Sling Model Exporter manipulates and return's the JSON content to be used by the custom react component code.

Exercises

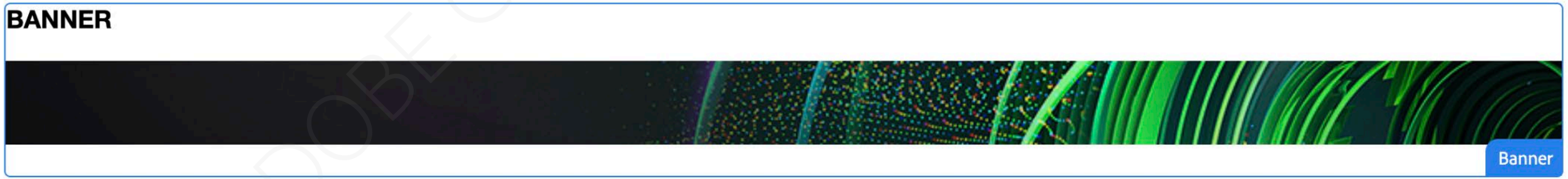
- Update the SPA
- Update the Template policy
- Author the Open Weather Component

Module 7

Extend a Core Component

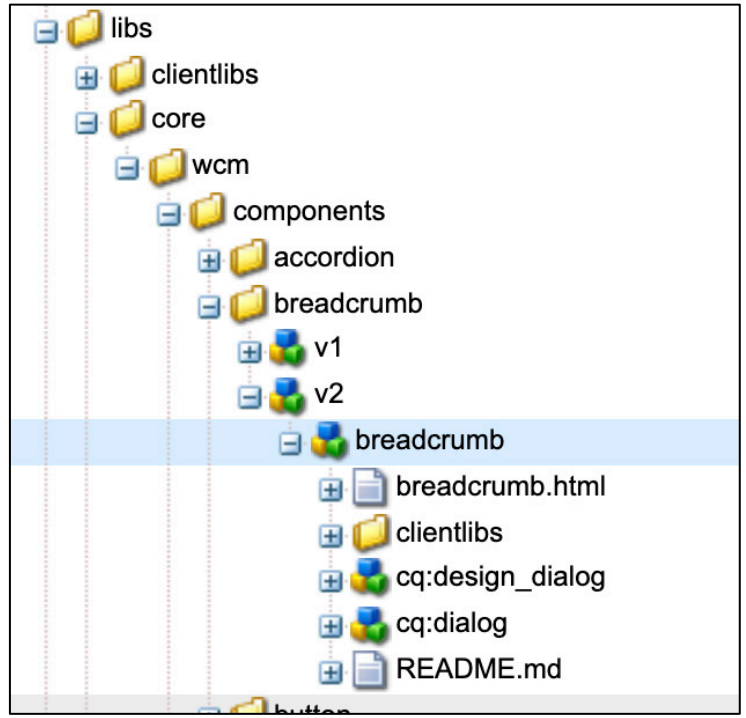
Extend a Core Component

- Extending an existing Core Component to be use with the AEM Spa Editor.
- Objectives
  - Extend an existing Core Component with additional properties and content.
  - Understand the basic of Component Inheritance with the use of sling:resourceSuperType.
  - Learn how to leverage the Delegation Pattern for Sling Models to re-use existing logic and functionality.
- What you will build:



Core Components

- Make page authoring more flexible and customizable
- Are developed in and delivered through GitHub
- Are in the JCR: `/libs/core/wcm/components`
  - `(Under /apps/.. In 6.5).`
  - Flexible configuration
  - Frequent incremental functionality improvements
  - Periodic release of content packages for component upgrades
  - Component versioning, modern implementation, and lean markup
  - Capability to serialize as JSON for headless CMS use cases



In Cloud Service core components are stored under /libs and are always up to date

Inspect the banner component implementation

- Extends the WKND image proxy component under ui.apps
- Inherits all functionality from the WKND SPA Image component via the sling:resourceSuperType property

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0"
    jcr:primaryType="cq:Component"
    jcr:title="Banner"
    sling:resourceSuperType="wknd-spa-react/components/image"
    componentGroup="WKND SPA React - Content"/>
```

Extends the image proxy component…

Inspect the SPA Image Proxy component implementation

- Extends the Core image component via the sling:resourceSuperType property
- Core components should not be used directly and should be extended as proxy components
  - Proxy pattern: Sling resource inheritance allows child components to inherit functionality and extend/override behavior when desired.
  - Sling inheritance supports multiple levels of inheritance, so ultimately the new Card component inherits functionality of the Core Component Image.

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0"
    jcr:primaryType="cq:Component"
    jcr:title="Image"
    sling:resourceSuperType="core/wcm/components/image/v2/image"
    componentGroup="WKND SPA React - Content"/>
```

… which extends the image Core Component.

Update the Banner Sling Model

- Populate the JSON for the Card component by extending a core component
  - Expose the value of bannerText via the sling model and JSON output
  - Reuse and return values from the image class to return title, alt and src.

```
// public getter to expose the value of `ba
@Override
public String getBannerText() {
    return bannerText;
}
```

```
// Re-use the Image class for all other methods:

@Override
public String getSrc() {
    return null != image ? image.getSrc() : null;
}

@Override
public String getAlt() {
    return null != image ? image.getAlt() : null;
}

@Override
public String getTitle() {
    return null != image ? image.getTitle() : null;
}
```

Update the React component

Now that the JSON model is populated with properties for bannerText, src, title and alt we can update the React component to display these.

```
export default class Banner extends Component {

    get content() {
        return <img
                    className="Image-src"
                    src={this.props.src}
                    alt={this.props.alt}
                    title={this.props.title ? this.props.title : this.props.alt} />;
    }

    // display our custom bannerText property!
    get bannerText() {
        if(this.props.bannerText) {
            return <h4>{this.props.bannerText}</h4>;
        }

        return null;
    }
```

Banner Component Review

Core components

- Cannot be used directly, they must be extended as proxy components
- Sling inheritance supports multiple levels of inheritance
  - The new Banner component inherits functionality of the Core Component Image
- Allows our react component to support additional properties and content

Exercises

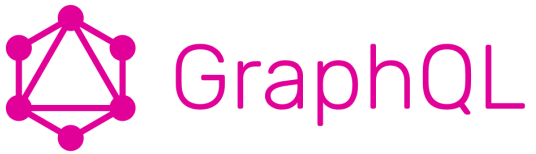
- Inheritance with Sling Resource Super Type
- Extend the Dialog
- Implement SPA Component
- Add Java Interface
- Implement Sling Model

Module 8

(Optional) GraphQL and Remote SPA

GraphQL

- An API query language and server-side runtime for executing using a developer defined type system.
- Not tied to any specific database of data storage- backed by existing code and data.
- A GraphQL service is created by defining types and fields on those types, then providing functions for each field on each type.
- Used in Experience Manager for ecommerce integration or querying data exposed via Content Fragments
- More information: https://graphql.org/



AEM's GraphQL APIs for Content Fragments

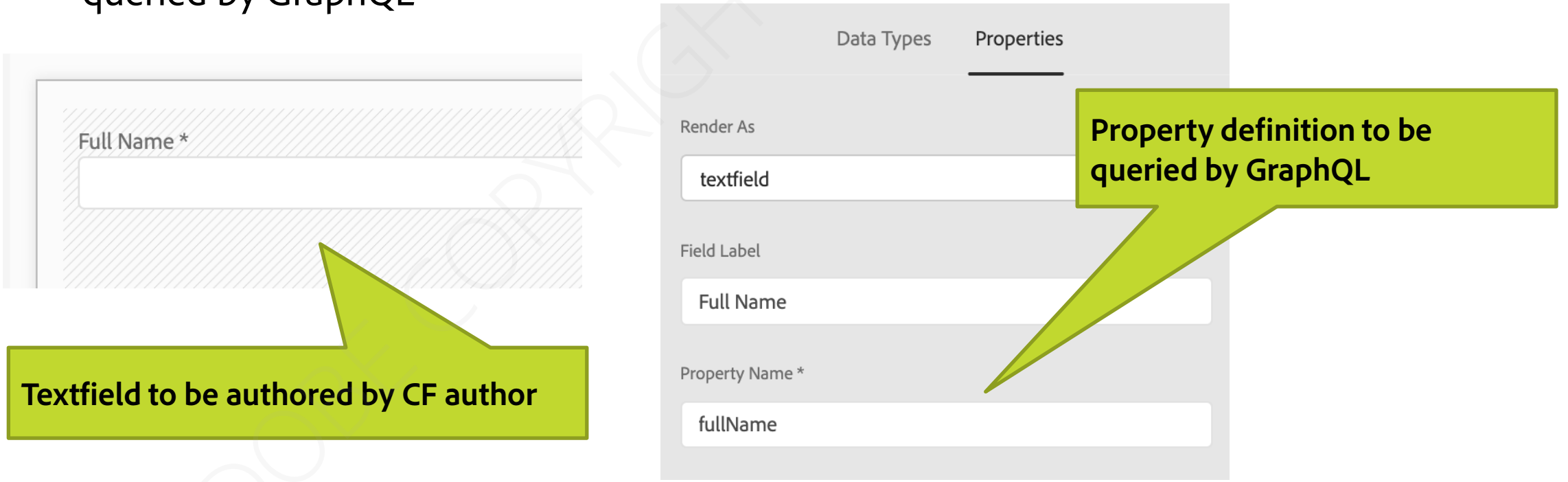
- Supports headless CMS scenarios where external client applications render experiences using content managed in Experience Manager.
- Using a Rest API Introduces challenges:
  - Large number of requests, fetching one object at a time
  - Over-delivering content, the app receives more than it needs
- Graph QL provides a query-based API to query AEM for only the content it needs via a single API call
- React App consumes Content Fragments' data via AEM's GraphQL APIs.

AEM GraphQL vs. AEM Content Services

	AEM GraphQL APIs	AEM Content Services
Schema definition	Structured Content Fragment Models	AEM Components
Content	Content Fragments	AEM Components
Content discovery	By GraphQL query	By AEM Page
Delivery format	GraphQL JSON	AEM ComponentExporter JSON

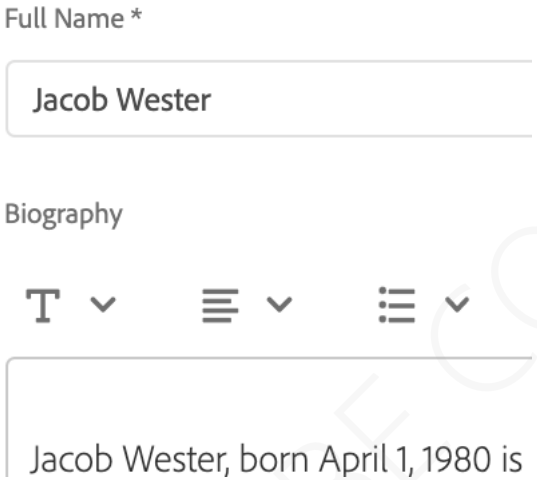
Content Fragment Model

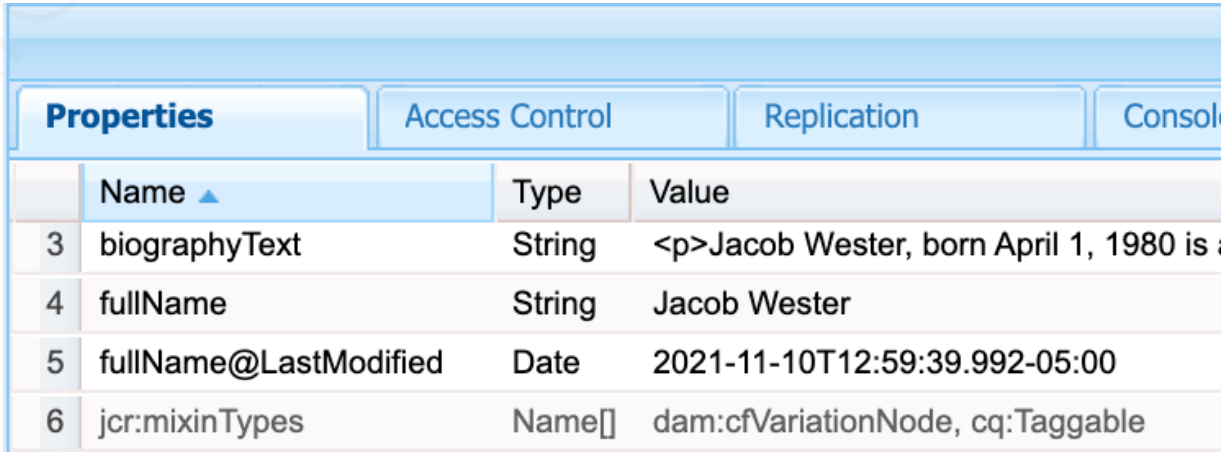
- Defines the GraphQL data schema and authoring template for a content fragment
  - Content Fragment Model editor creates the model and drives the data definition, to later be queried by GraphQL



Content Fragment Authoring

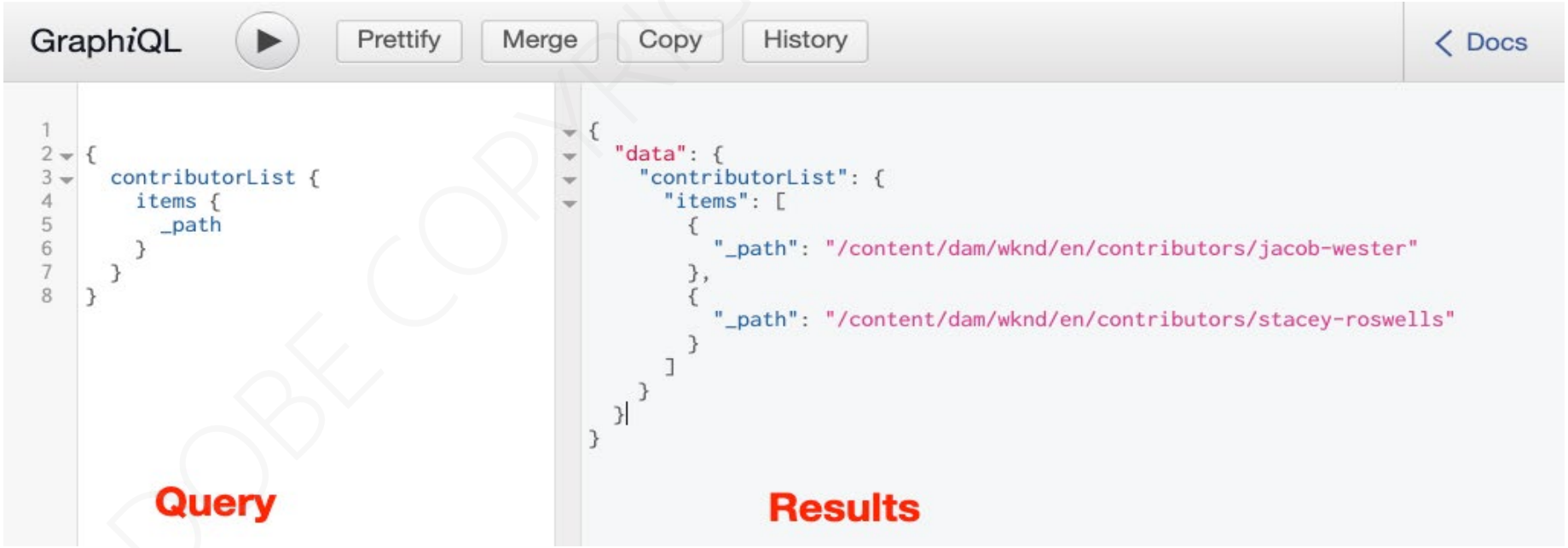
- Content Fragment is populated by content author based on newly defined model
- Provides the data to be queried by the GraphQL Schema





GraphiQL

- Integrated Dev environment for GraphQL
- Useful for testing GraphQL queries against Content Fragments
- Available as an AEM plugin via the Software Distribution Portal



Example GraphQL Queries

GraphiQL ▶ Prettify Merge Copy History Docs

```
{
 contributorByPath(_path: "/content/dam/wknd/en/contributors/scott-reynolds") {
    item {
      _path
      fullName
      biographyText {
        html
      }
    }
  }
}
```

```
{
  "data": {
    "contributorByPath": {
      "item": {
        "_path": "/content/dam/wknd/en/contributors/scott-reynolds",
        "fullName": "Scott Reynolds",
        "biographyText": {
          "html": "<p>I'm Scott Reynolds!</p>\n"
        }
      }
    }
  }
}
```

Query contributor at a given path. Return full name, occupation, biography text (as HTML).

Example GraphQL Queries (cont.)

Query the list of contributors, return full name, occupation. Filter based on occupation as Photographer.

Example GraphQL Queries (cont.)

GraphiQL ▶ Prettify Merge Copy History Docs

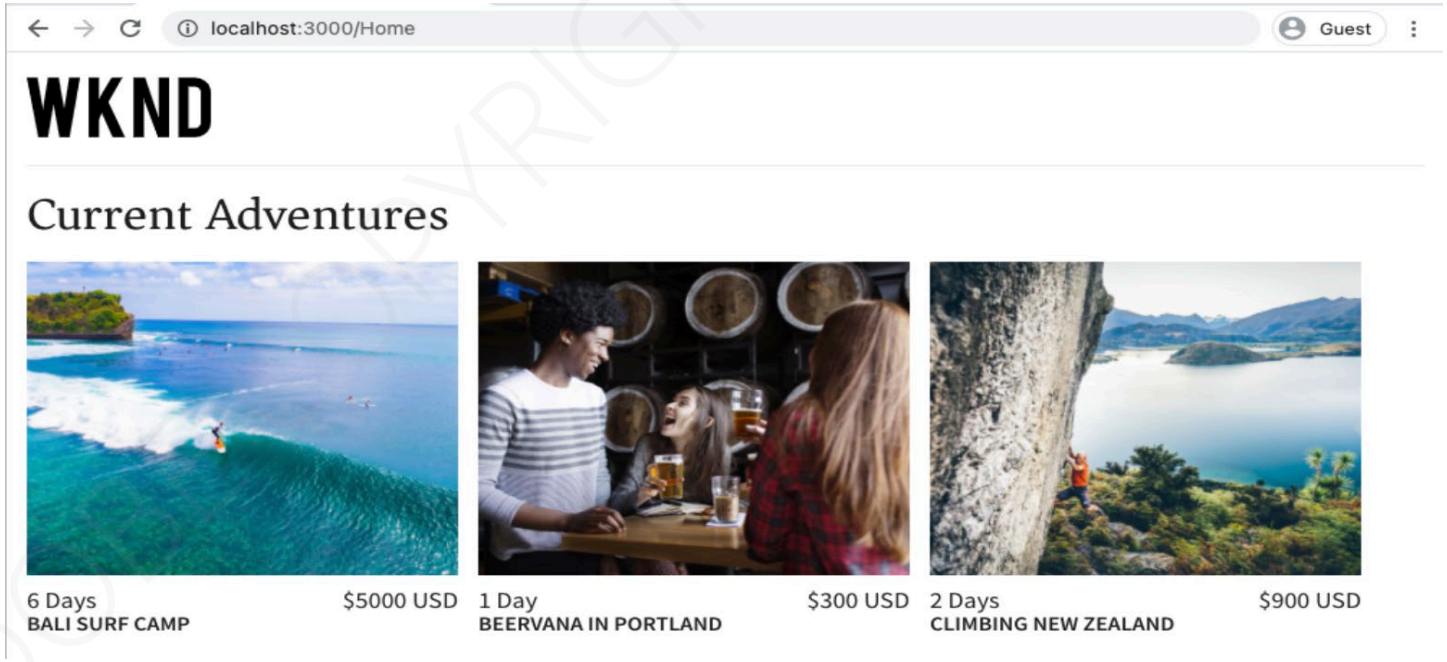
```
{
  contributorList {
    items {
      _path
      fullName
      occupation
      biographyText {
        html
      }
      pictureReference {
        ... on ImageRef {
          _path
          width
          height
        }
      }
    }
  }
}
```


```
{
  "data": {
    "contributorList": {
      "items": [
        {
          "_path": "/content/dam/wknd/en/contributors/jacob-wester",
          "fullName": "Jacob Wester",
          "occupation": "Writer",
          "biographyText": {
            "html": "<p>&lt;p&gt;Jacob Wester, born April 1, 1980 is an
American writer for the lifestyle magazine, the WKND. Jacob is also a
skier who competed for the United Sates in the 2016 Winter Olympics in
the Alpine Skiing and Ski Jump events. Jacob finished first in the 5000
meter Alpine solem with a time of 98 seconds, besting Stefan Sprinter
from Switzerland who finished second with a time of 100.02 seconds.
Jacob then finished second in the Skip Jump event with a score of 9.6,
losing to Jean-Claude Jagger of Germany.&amp;nbsp;&lt;/p&gt;
<br>&lt;p&gt;Jacob Wester was born in Grand Rapids, MI, and grew up
Traverse City, where he attended City High School and competed in
varsity Ski Club. Jacob was the speediest kid on the team and helped the
```


Query the list of contributors, return full name, occupation, biography text (as HTML). Return picture reference data (including width and height).

Querying from External React App

- Experience Manager's GraphQL APIs can be used to drive the experience in an external application (in this case, our react app)
- Calls are made from the app to AEM's GraphQL end-points
- GraphQL query filters a list of adventures Content Fragments by activity



Querying from External React App (cont.)

- useGraphQL.js listens for changes to the app's query
  - Contains React Effect Hook, on change makes a POST request to the AEM GraphQL end-point
  - Returns the JSON Response to the app
- Any time the Reactapp needs to make a GraphQL query, it invokes this custom useGraphQL(query) hook, passing in the GraphQL to send to AEM
- Response is inspected to see if it includes an errors object (sent if there issues with the GraphQL query, such as undefined fields based on the schema.

Querying from External React App – src/api/useGraphQL.js

```
function useGraphQL(query, path) {
    let [data, setData] = useState(null);
    let [errorMessage, setErrors] = useState(null);

    useEffect(() => {
      const sdk = new AEMHeadless({ endpoint: REACT_APP_GRAPHQL_ENDPOINT })
      const request = query ? sdk.runQuery.bind(sdk) : sdk.runPersistedQuery.bind(sdk);

      request(query || path)
        .then(({ data, errors }) => {
          //If there are errors in the response set the error message
          if(errors) {
            setErrors(mapErrors(errors));
          }
          //If data in the response set the data as the results
          if(data) {
            setData(data);
          }
        })
        .catch((error) => {
          setErrors(error);
        });
    }, [query, path]);
```

Query passed in and posted to the GraphQL endpoint defined in .env.development

JSON response parsed as data, errors caught from output

Querying from External React App – src/api/components/Adventures.js

```
//    const [query, setQuery] = useState('');
const [query, setQuery] = useState(allAdventuresQuery);

//backup
const persistentQuery = 'wknd/adventures-all';
//Use a custom React Hook to execute the GraphQL query
const { data, errorMessage } = useGraphQL(query, persistentQuery);
```

Query passed into useGraphQL.js

```
const allAdventuresQuery = `
{
    adventureList {
      items {
        _path
        adventureTitle
        adventurePrice
        adventureTripLength
        adventurePrimaryImage {
        ... on ImageRef {
            _publishUrl
            _path
            mimeType
            width
            height
          }
        }
      }
  }
}
`;
```

What You've Learned

You should now be able to:

- Install the AEM SPA Editor JS SDK and implement React components for both image and text components.
- Modify JS and CSS files and see those changes reflected in real time in the browser.
- Develop SPA Model Routing to allow navigation between different views of the app.
- Query Content Fragments from a SPA App using GraphQL

Adobe