**Pearson Candidate 2**

**Patterns Library – User Guide**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed by** | **Approved By** |
| **Name** | Nalini Krishna Chakka |  |  |
| **Role** |  |  |  |
| **Signature** |  |  |  |
| **Date** | 05-Aug-2016 |  |  |

Table of Contents

Table of Contents 2

Overview of the component 3

1.1 Question Metadata 3

1.2 Assessment Metadata 3

1.3 Bank Metadata 4

1.4 Asset Metadata (Not Ready) 4

1.5 Add an Asset: 4

1.6 Product Linking Component…………………………………………………………………………………………………………..6

1.7 Search and Select Component……………………………………………………………………………………………………….7

Usage: 5

Component initialization and launch 6

3.1 Initialize pattern library using config object: 6

3.1.1 Pattern configuration with APIGEE: 6

3.2 Create Pattern component and configure pattern: 10

3.3 Provide callback hooks from component to host application: 10

3.4 Render the component: 11

3.5 Provide any additional call backs for events: 11

FAQ 12

Example 12

# Overview of the component

Patterns Library provides a set of pluggable components that achieve the following functionality:

1. Publish metadata related to Question / Assessment / Asset / Bank – Metadata Pattern.
2. Ability to upload asset and maintain its metadata – Upload pattern
3. Search assets / Browse assets – Search Pattern

This document describes two metadata patterns:

## 1.1 Question Metadata

This component is launches metadata pattern for capturing question metadata. This patterns provides ability to capture the following metadata associated with a question:

This component is responsible for the following:

1. Retrieves the allowed values for all taxonomies
2. Renders the UI to capture metadata for a question
3. Leverages MDS api to store meatadata entered into Marklogic.
4. Provides the ability to retrieve metadata associated with a question
5. Provides the metadata captured as a JSON object to the host application (Authoring tools). Host applications can use this object for local storage.

## 1.2 Assessment Metadata

This component is launches metadata pattern for capturing Assessment metadata. This patterns provides ability to capture the following metadata associated with an assessment:

This component is responsible for the following:

1. Retrieves the allowed values for all taxonomies
2. Renders the UI to capture metadata for a question
3. Leverages MDS api to store meatadata entered into Marklogic.
4. Provides the ability to retrieve metadata associated with an assessment
5. Provides the metadata captured as a JSON object to the host application (Authoring tools). Host applications can use this object for local storage.

## 1.3 Bank Metadata

This component is launches metadata pattern for capturing Bank metadata. This patterns provides ability to capture the following metadata associated with a Bank:

This component is responsible for the following:

1. Retrieves the allowed values for all taxonomies
2. Renders the UI to capture metadata for a question
3. Leverages MDS api to store meatadata entered, into Marklogic.
4. Provides the ability to retrieve metadata associated with a Bank.
5. Provides the metadata captured as a JSON object to the host application (Authoring tools). Host applications can use this object for local storage.

## 1.4 Asset Metadata (Not Ready)

This component is launches metadata pattern for capturing Asset metadata. This patterns provides ability to capture the following metadata associated with a Asset:

This component is responsible for the following:

1. Retrieves the allowed values for all taxonomies
2. Renders the UI to capture metadata for a question
3. Leverages MDS api to store meatadata entered, into Marklogic.
4. Provides the ability to retrieve metadata associated with an Asset.
5. Provides the metadata captured as a JSON object to the host application (Authoring tools). Host applications can use this object for local storage.

## 1.5 Add an Asset:

This component is launches UI to search for an asset based on a keyword / browse the folder structure in Alfresco depending on the user permissions and also provide ability to upload an asset. User has the ability to launch “Add an Asset” component specific to a particular product.

This component is responsible for the following:

1. Allows the user to search for an asset based on key word.
2. Provides the ability to filter the search based on content type.
3. Tabs visibility is configurable using a JSON configuration at launch time.
4. Display the results after applying pagination.
5. Display the folder list for which the user has access to.
6. Up on selection of a particular folder, the contents of the folder are shown.
7. Ability to upload an asset.
8. Ability to pick a folder from among the available folder list while uploading an asset.
9. Provide an ability to pick an asset to associate it with a question / assessment.

1.6 Product Linking component:

This component provides the ability to pick a product among the list of products the user has access to. Only the folders corresponding to the user will be shown. The list of products are fetched from Alfresco. Refer to section 3.1.2 for Alfresco configuration.

The Product linking component provides the following:

1. Displays a dropdown consisting of all product names.
2. Allow the user to pick a product and click on Link
3. Provides the Product name and Alfresco Node Ref of the selection in the call back function.

1.7 Search and Select Component:

This component provides an ability to search Interactive/questions from mark logic, it provides collection of Assessment metadata based on the some search parameters. The UI will be displayed list of assessments which has workExample (SCAPI manifestation).

The Search and Select Component provides the following:

* + - 1. Host application can able to pass the search item and filter type to Search and Select component based on the parameter from host application, search and select component display list of assessments else it will display all assessments without filter type
      2. User can able to re-issue the query to filter the assessments using filter type
      3. User can control the display items per page
      4. User can able to save their search terms and they can see their save last three recent searches and saved searches
      5. User can able to sort the results which displayed on the application. When user clicks on the column names it will sort ASC/DESC
      6. Pagination functionality allows user see their next set of assessments
      7. When selecting particular record and click on select we will provide you with following information
      8. Title
      9. Type
      10. dateModified
      11. workExample
      12. taxonomicType

# Usage:

Authoring tools can integrate patterns library using the <script> tag.

<script src="/lib/PatternsLib.js"></script>

PatternsLib is a package that bundles up all the pattern library components and applications can launch context specific components. Each component will maintain its own state and hence won’t interfere with each other.

In production, use AKAMAI CDN references to use the PatternLib instead of embedding PatternsLib.js in your local work space. This will make the updates flow through seamlessly without having the need to redeploy your application.

# Component initialization and launch

Initialization of component follow factory pattern. The following steps need to be followed in order to initialize appropriate component.

## 3.1 Initialize pattern library using config object:

Pattern config object serves as a property bundle for Pattern library and includes properties such as URL for MetaData Services Api, Authorization values and so on. Please note that Pattern library should be initialized only once irrespective of the number of components being rendered.

As an example, consider the following pattern config object:

const libConfig = {'locale': 'en\_US',

'environment': 'dev',

'headers' : {'Content-Type' : 'application/json',

'Accept' : 'application/ld+json',

'X-Roles' : 'roleX,roleY,roleC',

'Authorization' : 'Basic Ymx1ZWJlcnJ5OmVAQkhSTUF2M2V5S2xiT1VjS0tAWl56Q0ZhMDRtYw=='

},

'database' : '?db=qa6',

'server' : 'https://uat.pearsonmeta.io',

‘taxonomyserver’ : ‘https://uat.pearsonmeta.io’,

'port' : '8080'

};

In certain environments, server and taxonomy server URLs may be different. Hence there are place holders for different servers.

Using libConfig object, initialize patterns library as follows:

patternsLib.setup(libConfig);

The above configuration applies only for environments where APIGEE gateway is not in place.

## 3.1.1 Pattern configuration with APIGEE:

In case of environments which have APIGEE security layer, the following additional properties need to be passed in the PatternsLib config.

**'x-apikey' :** '5x8gLqCCfkOfgPkFd9YNotcAykeldvVd',

**'X-PearsonSSOSession'** : 'AQIC5wM2LY4SfczJwDEKiLveBuH9DtGGXLkfvRlveimpxgQ.\*AAJTSQACMDIAAlNLABM0NjAwMDQ4NTYxOTkyNTk3NTU1AAJTMQACMDE.\*'

Procedure to generate x-apikey and X-PearsonSSOSession is beyond the scope of this document.

3.1.2 Pattern configuration with Alfresco:

In addition to the above libConfig, the components ProductLinking and AddAnAsset, require Alfresco configuration. Alfresco configuration is done using the following keys. Add these keys to libconfig object during configuration initialization.

alfToken = Pre authenticated token that host application can pass to component to avoid passing userid / password.

alfuname = In case Alf token is not available, user name should be sent

alfpwd = Password for the user name

nodeRef = Used only by AddAnAsset component. This corresponds to Alfresco site where the assets are uploaded and assets are picked up from.

tabVisibility = Used by AddAnAsset component only. This property is a JSON object that defines the visibility for various tabs: Images / Video / Audio and Others.

Example for tabVisibility: { "audio" : true, "image": true, "other":false, "video": true, “defaultTab”:”image”}

alfServer = Alfresco Server URL. This is typically supplied by Product Linking component depending up on the product selected. The Server URL returned in the call back from Product Linking needs to be supplied to AddAnAsset component by host application.

repoName = This attribute determines the Alfresco repository name. Ex: US PPE, UK PPE etc. This is an optional attribute and wont affect the functionality of AddAnAsset component.

workURN = Host application can optionally request for wURN and mURN of the selected asset from AddAnAsset component by specifying workURN as true in the configuration. Note that in order for this attribute to work, a special configuration needs to be done in Alfresco for the site selected. Hence it may not work for all others. If you don’t need Work URN and Manifestation URN for the selected asset, then we recommend to set this attribute as false. Refer to the sample application provided by C2 team for details. If you use workURN as true, then wURN and mURN attributes will be returned back to the host application in the select call back.

The following sections (3.1.3 and 3.1.4) talks about the additional configurations for “Add an asset” component.

3.1.3 Configuration to obtain EPS URL:

EPS URL refers to the URL using which you can access the Alfresco asset directly without going through Apigee / Alfresco authentication. A special configuration is needed in order to obtain the EPS URL from Patterns library.

Add additional attribute, epsUrl, to the tabVisibility configuration to obtain EPS URL in the call back.

For example, along with the visibility attributes set in section 3.1.2, epsUrl can be set to true or false. Default value, if not specified in the configuration, is true.

TabVisibility:

{ "audio" : true, "image": true, "other":true, "video": true,"epsUrl":true}

Once epsUrl is configured, the call back contains the EPS URL and the Alfresco URL for the thumbnail.

3.1.4 Supported MIME types:

MIME type of the asset determines which tab the asset will be displayed in Search and Browse functionalities. Here is a mapping of the Asset Mime type and the tab name:

|  |  |
| --- | --- |
| **MIME Type** | **Tab** |
| 'jpg','jpeg','png','tiff', 'gif' | Image |
| 'mp3' | Audio |
| 'mp4' | Video |
| 'docx','xls','xlsx', 'ppt','pptx','txt','pdf','csv','odg', 'odp','odt','ods','ebk','wdgt' | Other |

3.1.5 Library configuration for Product Linking:

Product Linking component has the ability to pull product names (Alfresco Sites) from various Alfresco repositories based on the configured list of repositories. Users will be presented with a consolidated list of product names in this component. In case of conflicts in the product names from different repositories, product linking component qualifies conflicted product names with the appropriate repository name.

For example: c2testsite01 (pub) – US

The above product name means that c2testsite01 is a public repository in US instance of Alfresco.

During initialization of Product Linking component, users can supply a list of repositories using repoList attribute.

libconfig.repoList = [

{ 'repo' : 'https://staging.api.pearson.com/content/cmis/ukwip', 'repoName' : 'UK'},

{ 'repo' : 'https://staging.api.pearson.com/content/cmis/ukwip', 'repoName' : 'US East'},

{ 'repo' : 'https://staging.api.pearson.com/content/cmis/uswip', 'repoName' : 'US East1'}

]

Upon selection of a product, host application gets the details of product (uuid, name etc) along with its corresponding repository information in the call back.

Host application needs to preserve this and launch AddAnAsset by passing repo (referenced as alfServer) and repoName in AddAnAsset libConfig object.

## 3.2 Create Pattern component and configure pattern:

Using the reference to pattern library, create appropriate component type.

The following table gives us a list of component types:

|  |  |
| --- | --- |
| Pattern Name | Pattern type to be used |
| Assessment metadata | patternsLib.type.ASSESSMENT |
| Question Metadata | patternsLib.type.QUESTION |
| Bank Metadata | patternsLib.type.BANKMETADATA |
| Asset Metadata | patternsLib.type.ASSETMETADATA |
| Add an Asset | patternsLib.type.AddAnAsset |

patAssesment = patternsLib.create(patternsLib.type.ASSESSMENT);

Configure the pattern object with the following information:

1. Where to render the component on the DOM.
2. Initialization data, if any (For example, UUID of the existing question / assessment)

let patAssesmentConfig = {'selector' : renderderedTagSelector};

if(assesmentUUID!==''){

patAssesmentConfig.uuid = assesmentUUID;

}

## 3.3 Provide callback hooks from component to host application:

Host application need to provide a call back function for it to be able to read the data from component.

As an example, the following piece of code is a call back provided by the host application so that it can display the JSON data it receives from the component as a table in the DOM element: ‘compResp’

// Define a callback which will receive results back from the pattern instance

let cbAssesment = (data) => {

// data is a JSON structure returned back from the pattern instance

// Here we are just displaying the stringified version of JSON structure

let e = document.getElementById('compResp');

//e.innerHTML = String(data);

let metadataContent ='<table>';

for (var key in data) {

if (data.hasOwnProperty(key)) {

let property = '<tr><td>'+key+'</td><td>'+data[key]+'</td></tr>';

if(typeof(data[key]) === 'string'){

metadataContent = metadataContent+property;

}

}

}

metadataContent =metadataContent+'</table>';

e.innerHTML = metadataContent;

};

Configure the newly created component with this call back to register call back with the component:

// Setup the instance using configuraton and callback

patAssesment.setup(patAssesmentConfig, cbAssesment);

## 3.4 Render the component:

Render the component by calling run method.

// Run the render method, processess user interactions and do teardown when finished

patAssesment.run();

## 3.5 Provide any additional call backs for events:

Communication between the host application and the component is done using event mechanism. We leverage ‘bean’ npm package to fire and listen for events.

Consider the following example where we want the Assessment metadata component to save data to MarkLogic when user clicks on Save button in QuaD application. QuaD fires an event to component and component publishes the data to Marklogic. QuaD can register a call back with the component so that it also receives a copy of the JSON object component sent to MarkLogic. In order to achieve this, QuaD has to fire an event to component and also register a call back to get copy of the data.

let SaveCallBack = (channelData) => {

// channel data is a JSON structure

// Here we are just displaying the stringified version of JSON structure

let e = document.getElementById('assesmentResp');

e.innerHTML = JSON.stringify(channelData);

};

//fire an event

onSaveAssesment = function(){

patAssesment.fire({"id":"aaa"});

}

//Register call back with the component

patAssesment.on(SaveCallBack);

# FAQ

This section will be updated incrementally as we receive questions related to usage from host applications.

**4.1.1 Will the host application receive locally stored values in the call back?**

Yes, the host application will receive locally stored values in all save call backs as well as all other event call backs.

**4.1.2 Host application should pass the Product name field while launching the component. How is it achieved?**

Name field will be leveraged only during Save operation. Hence Host application can pass the name field in the custom data that is passed to the component upon clicking Name. You can use the key as ‘name’ and pass the values so that our components pick it and send to MarkLogic during save process.

# Example

Refer to the following bitbucket URL to install a sample application that launches the components discussed above. Follow the Readme.MD file to get instructions on how to install the sample application.

<https://bitbucket.pearson.com/projects/C2AI/repos/patternslib/browse>

Pick the branch  “final”.