**Estructura de la Solución**

La solución se divide en dos bloques:

1. **Frontend / Componente:**
   * **HTL:** Vista del componente (dynamic-search-cards.html).
   * **Diálogo de Configuración:** Definido en \_cq\_dialog.xml para configurar título y endpoint.
   * **Sling Model:** DynamicSearchCardsModel.java para encapsular la lógica del componente.
   * **ClientLibs:** JavaScript (ES6+) para filtrar y buscar en tiempo real, y CSS para estilos.
2. **Backend Avanzado:**
   * **Servicio OSGi:** ExternalApiService e implementación ExternalApiServiceImpl para consumir el endpoint externo.
   * **Servlet:** DynamicSearchDataServlet que expone los datos en JSON.
   * **Configuración OSGi:** Para definir parámetros como el endpoint y el timeout, con ejemplo de configuración en /apps/<proyecto>/config.
   * **Sling Model:** Complementa la lógica del componente, integrando la configuración y datos del servicio.

**Paso a Paso del Desarrollo**

**Paso 1: Crear la Estructura del Proyecto AEM**

* Utiliza el arquetipo oficial AEM (aem-project-archetype) para generar un proyecto multi-módulo que incluya:
  + proyecteve.core: Contendrá la lógica Java y los servicios OSGi.
  + proyecteve.ui.apps: Para componentes, clientLibs y configuración de UI.
  + proyecteve.ui.content: Para empaquetar el contenido (plantillas, diálogos).
  + proyecteve.all: Módulo que agrupa e integra todo en un paquete final para desplegar en AEM.

**Paso 2: Desarrollo del Componente Frontend**

**2.1. Crear la Vista HTL**

* Ubica el componente en /apps/proyecteve/components/dynamic-search-cards/.
* Crea el archivo dynamic-search-cards.html con la siguiente estructura:

html

Copiar

<sly data-sly-use.model="com.jadeveloper.core.models.DynamicSearchCardsModel"

data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html" />

<sly data-sly-call="${clientLib.css @ categories='ja-developer.dynamicsearchcards'}" />

<div class="cmp-dynamic-search-cards"

data-endpoint="${model.externalApiEndpoint @ context='unsafe'}"

role="region"

aria-label="Búsqueda dinámica de tarjetas">

<h2 data-sly-text="${model.title}">Título por defecto</h2>

<div class="cmp-search-input">

<label for="search-input" class="visually-hidden">Buscar tarjetas</label>

<input type="text"

id="search-input"

placeholder="Buscar tarjetas..."

aria-label="Buscar contenido" />

</div>

<nav id="tag-filters" class="cmp-tag-filters"

role="navigation"

aria-label="Filtrar por categoría">

<!-- Filtros generados dinámicamente -->

</nav>

<div id="cards-loading" class="cmp-cards-loading" aria-hidden="true">

<div class="spinner"></div>

<span>Cargando tarjetas...</span>

</div>

<div id="cards-container" class="cmp-cards-container" role="list">

<!-- Tarjetas renderizadas dinámicamente -->

</div>

</div>

<sly data-sly-call="${clientLib.js @ categories='ja-developer.dynamicsearchcards'}" />

**2.2. Crear el Diálogo de Configuración (\_cq\_dialog.xml)**

Define el diálogo para configurar el título y el endpoint:

xml

Copiar

<?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"

xmlns:jcr="http://www.jcp.org/jcr/1.0"

jcr:primaryType="nt:unstructured"

jcr:title="Dynamic Search Cards"

sling:resourceType="cq/gui/components/authoring/dialog">

<content jcr:primaryType="nt:unstructured">

<items jcr:primaryType="nt:unstructured">

<titleField

jcr:primaryType="nt:unstructured"

sling:resourceType="granite/ui/components/coral/foundation/form/textfield"

fieldLabel="Título principal"

name="./title"/>

<endpointField

jcr:primaryType="nt:unstructured"

sling:resourceType="granite/ui/components/coral/foundation/form/textfield"

fieldLabel="Endpoint del servicio externo"

name="./externalApiEndpoint"/>

</items>

</content>

</jcr:root>

**2.3. ClientLibs y JavaScript**

* En la carpeta clientlibs/js, crea dynamic-search.js con la lógica de filtrado y búsqueda en tiempo real (incluyendo debouncing si es necesario).
* En la carpeta clientlibs/css, define dynamic-searchcards.css para los estilos.

**Paso 3: Desarrollo del Backend Avanzado**

**3.1. Sling Model (DynamicSearchCardsModel.java)**

Utiliza Sling Models para exponer la configuración del componente:

java

Copiar

package com.jadeveloper.core.models;

import javax.inject.Inject;

import org.apache.sling.api.resource.Resource;

import org.apache.sling.models.annotations.Default;

import org.apache.sling.models.annotations.DefaultInjectionStrategy;

import org.apache.sling.models.annotations.Model;

import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;

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

public class DynamicSearchCardsModel {

@ValueMapValue

@Default(values = "Dynamic Search Cards")

private String title;

@ValueMapValue

@Default(values = "/content/proyecteve")

private String searchBasePath;

@ValueMapValue

@Default(values = "https://api.example.com/data")

private String externalApiEndpoint;

public String getTitle() {

return title;

}

public String getSearchBasePath() {

return searchBasePath;

}

public String getExternalApiEndpoint() {

return externalApiEndpoint;

}

}

**3.2. Servicio OSGi (ExternalApiService)**

Define la interfaz y su implementación para consumir el endpoint externo.

**Interfaz:**

java

Copiar

package com.jadeveloper.core.services;

public interface ExternalApiService {

String getEndpointUrl();

int getTimeout();

String fetchData();

}

**Implementación:**

java

Copiar

package com.jadeveloper.core.services.impl;

import com.jadeveloper.core.services.ExternalApiService;

import org.apache.commons.io.IOUtils;

import org.osgi.service.component.annotations.Activate;

import org.osgi.service.component.annotations.Component;

import org.osgi.service.component.annotations.Modified;

import org.osgi.service.metatype.annotations.AttributeDefinition;

import org.osgi.service.metatype.annotations.Designate;

import org.osgi.service.metatype.annotations.ObjectClassDefinition;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.io.InputStream;

import java.net.HttpURLConnection;

import java.net.URL;

import java.nio.charset.StandardCharsets;

@Component(

service = ExternalApiService.class,

configurationPolicy = org.osgi.service.component.annotations.ConfigurationPolicy.REQUIRE

)

@Designate(ocd = ExternalApiServiceImpl.Config.class)

public class ExternalApiServiceImpl implements ExternalApiService {

private static final Logger log = LoggerFactory.getLogger(ExternalApiServiceImpl.class);

@ObjectClassDefinition(name = "JADeveloper - External API Service", description = "Consume un servicio externo")

public @interface Config {

@AttributeDefinition(name = "API Endpoint URL", description = "URL del servicio externo")

String endpointUrl();

@AttributeDefinition(name = "Timeout (ms)", description = "Timeout de conexión en milisegundos")

int timeout() default 5000;

}

private String endpointUrl;

private int timeout;

@Activate

@Modified

protected void activate(Config config) {

this.endpointUrl = config.endpointUrl();

this.timeout = config.timeout();

log.info("ExternalApiServiceImpl activado con endpointUrl={} y timeout={}", endpointUrl, timeout);

}

@Override

public String fetchData() {

if (endpointUrl == null || endpointUrl.trim().isEmpty()) {

log.error("Endpoint URL no configurado");

return "{\"error\":\"Endpoint URL no configurado\"}";

}

HttpURLConnection conn = null;

try {

URL url = new URL(endpointUrl);

conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setConnectTimeout(timeout);

conn.setReadTimeout(timeout);

int status = conn.getResponseCode();

if (status != HttpURLConnection.HTTP\_OK) {

log.error("Error al conectar con la API externa. Código de estado: {}", status);

return "{\"error\":\"Error al llamar a la API: " + status + "\"}";

}

try (InputStream is = conn.getInputStream()) {

return IOUtils.toString(is, StandardCharsets.UTF\_8);

}

} catch (Exception e) {

log.error("Excepción al llamar a la API: {}", e.getMessage(), e);

return "{\"error\":\"No se pudo obtener datos del servicio externo: " + e.getMessage() + "\"}";

} finally {

if (conn != null) {

conn.disconnect();

}

}

}

@Override

public String getEndpointUrl() {

return this.endpointUrl;

}

@Override

public int getTimeout() {

return this.timeout;

}

}

**3.3. Servlet para Exponer Datos JSON (DynamicSearchDataServlet.java)**

Crea un servlet que llame al servicio y exponga los datos en JSON:

java

Copiar

package com.jadeveloper.core.servlets;

import com.jadeveloper.core.services.ExternalApiService;

import org.apache.sling.api.SlingHttpServletRequest;

import org.apache.sling.api.SlingHttpServletResponse;

import org.apache.sling.api.servlets.SlingSafeMethodsServlet;

import org.json.JSONArray;

import org.json.JSONObject;

import org.osgi.service.component.annotations.Component;

import org.osgi.service.component.annotations.Reference;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import javax.servlet.Servlet;

import java.io.IOException;

import java.io.PrintWriter;

import static org.osgi.framework.Constants.SERVICE\_DESCRIPTION;

import static org.osgi.framework.Constants.SERVICE\_VENDOR;

@Component(

service = Servlet.class,

property = {

"sling.servlet.methods=GET",

"sling.servlet.paths=/bin/searchcards",

SERVICE\_DESCRIPTION + "=Servlet para obtener tarjetas desde un servicio externo",

SERVICE\_VENDOR + "=JA Developer"

}

)

public class DynamicSearchDataServlet extends SlingSafeMethodsServlet {

private static final Logger log = LoggerFactory.getLogger(DynamicSearchDataServlet.class);

private static final long serialVersionUID = 1L;

@Reference

private ExternalApiService externalApiService;

@Override

protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws IOException {

response.setContentType("application/json");

response.setCharacterEncoding("UTF-8");

JSONArray cards = new JSONArray();

try {

// Llama al servicio externo

String jsonData = externalApiService.fetchData();

log.info("Datos externos recibidos: {}", jsonData);

// Se asume que la respuesta contiene un array de resultados

JSONObject externalResponse = new JSONObject(jsonData);

JSONArray results = externalResponse.optJSONArray("results");

if (results != null) {

for (int i = 0; i < results.length(); i++) {

JSONObject game = results.getJSONObject(i);

JSONObject card = new JSONObject();

card.put("title", game.optString("name", "Sin título"));

card.put("description", "Fecha: " + game.optString("released", "N/A") +

", Rating: " + game.optDouble("rating", 0.0));

card.put("image", game.optString("background\_image", ""));

JSONArray tags = new JSONArray();

if (game.has("genres")) {

for (int j = 0; j < game.getJSONArray("genres").length(); j++) {

JSONObject genre = game.getJSONArray("genres").getJSONObject(j);

tags.put(genre.optString("name", "Sin categoría"));

}

}

card.put("tags", tags);

cards.put(card);

}

}

} catch (Exception e) {

log.error("Error en DynamicSearchDataServlet: {}", e.getMessage(), e);

response.setStatus(SlingHttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR);

JSONObject error = new JSONObject();

error.put("error", "No se pudo obtener datos del servicio externo");

error.put("detail", e.getMessage());

try (PrintWriter writer = response.getWriter()) {

writer.write(error.toString(2));

}

return;

}

try (PrintWriter out = response.getWriter()) {

out.write(cards.toString(2));

}

}

}

**Paso 4: Configuración en AEM**

1. **Archivo de Configuración de Ejemplo:**  
   Crea un archivo de configuración en /apps/proyecteve/config (por ejemplo, com.jadeveloper.core.services.ExternalApiService.config) con propiedades como:

ini

Copiar

endpointUrl=https://api.example.com/data

timeout=5000

1. **Desplegar el Proyecto:**  
   Compila el proyecto con mvn clean install y despliega el paquete final en AEM. Verifica que el servicio OSGi se active y que el servlet esté disponible en la ruta /bin/searchcards.

**Conclusión**

Este documento detalla paso a paso cómo crear un componente avanzado en AEM que:

* Muestra tarjetas filtrables con título, descripción, imagen y categorías.
* Permite búsqueda en tiempo real.
* Consume datos de un servicio externo (configurable vía OSGi).
* Expone un servlet para retornar datos en JSON.
* Utiliza Sling Models, HTL, JavaScript moderno, y sigue buenas prácticas de accesibilidad y performance.