SEARCH COMPONENT

1-->Here we search the 'Digiquad' word as Request Parameter as below

"htp://localhst:4502/content/myite/us/en/ab.html?RequestParameter[pagePath]=Digiquad"

2→Through HTl this request goes to backend

<div data-sly-use.abc ="com.adobe.aem.mysite.site.core.models.PageSearchModel" >

3→ In order to read the request param which is given by the user in frontend, we have to declare the variable in model class as below

private String pagePath;

- **Retrieve the pagePath parameter from the request**
  pagePath=request.getParameter("pagePath");

4→If pagepath is not null , It process the request further .

```
if (pagePath != null && !pagePath.isEmpty()) {
            LOG.info("Received request with pagePath: {}", pagePath);
        } else {
            LOG.error("No pagePath parameter found in the request");

        }
```

- **our requirement is "if I search any word using request parameter ,it render all the pages related to that word"**
- **To read the pages related information or other information we need jcr access , We can't directly call the properties from backend to jcr for that we use Resoue Resolver .**
- **Resource Resolver provide access to 'Resources' , Resource represents 'content' in AEM , Resource Resolver Factory creates Resource Resolver .**
- **For doing all this things , We have to create System User and provide the necessary permissions to read the jcr**

```
Impo rt org.apache.sling.api.resource.Resource;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.api.resource.ResourceResolverFactory;
import javax.jcr.Session;
```

- Now ,We have a access to call the JCR

- Now our task is , if I request anything in the request parameter , it go to jcr and search as per our request.

5→ Once request is passed , Call performQuery on pageservice to get foundpagepaths and pagetitles .

```
try {

     ResourceResolver resolver = newResolver(resolverFactory);


      pageService.performQuery(pagePath);
      foundPages = pageService.getFoundPages();


        } catch (Exception e) {
            LOG.error("Error occurred while processing the request",
e);

        }
    }
```

6→In the above 5th point we have two methods

```
      //pageService.performQuery(pagePath);
      //foundPages = pageService.getFoundPages();
```

This 2 methods defined in pageService class and perform the 'performQuery()' method accordingly .

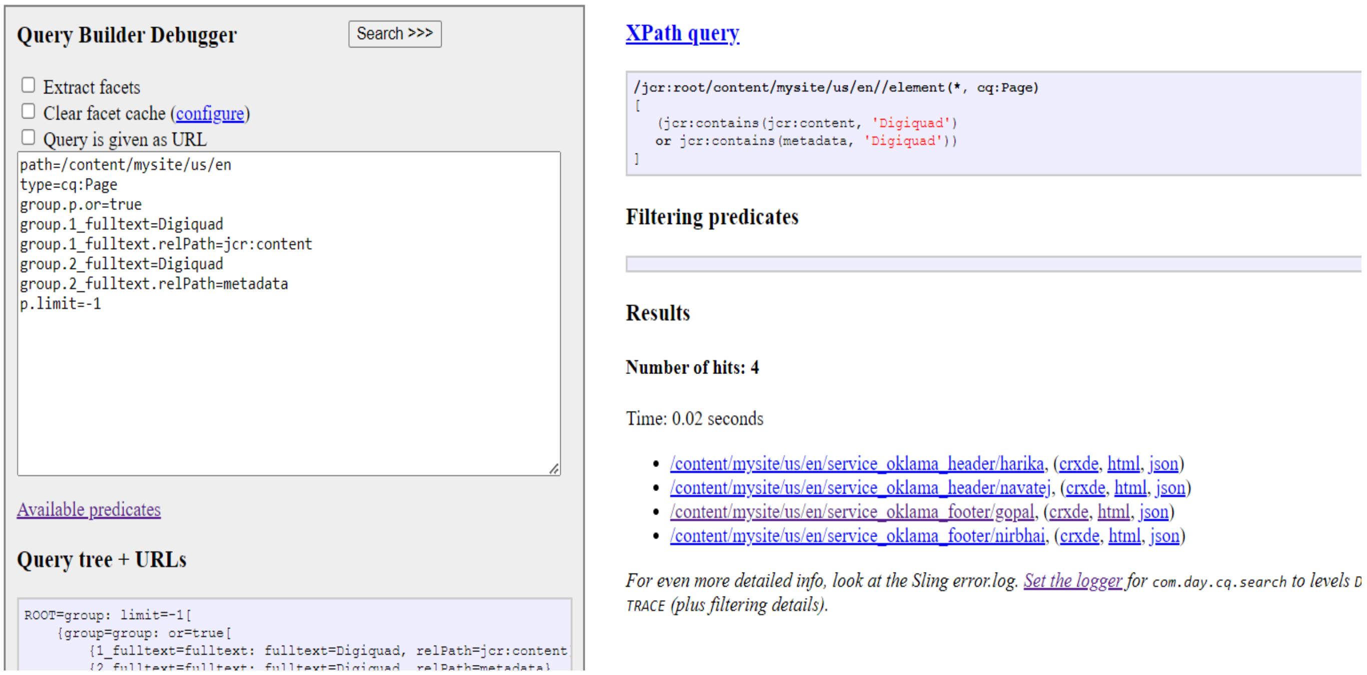
```
public void performQuery(String pagePath) {



}
```

7→Based on the request received, performQuery(String pagePath) method will create a query using queryBuilder.

Query Builder:

If i want to search for a word “digiquad” in the path ‘content/mysite/us/en’, I have to go to this path and check manually which takes so much time.Instead of searching in hard way aem provides queryBuilder feature. By using this we can retrieve all the jcr data easily.

- In this case I searched one word called ‘Digiquad’ , It went to this “/content/mysite/us/en” path and giving the result. Based on the query gaven it has given 4 pages containing the word ‘Digiquad’.



```
path=/content/mysite/us/en
type=cq:Page
group.p.or=true
group.1_fulltext=Digiquad
group.1_fulltext.relPath=jcr:content
group.2_fulltext=Digiquad
group.2_fulltext.relPath=metadata
p.limit=-1
```

```
ROOT=group: limit=-1[
    {group=group: or=true[
        {1_fulltext=fulltext: fulltext=Digiquad, relPath=jcr:content
        {2_fulltext=fulltext: fulltext=Digiquad, relPath=metadata}
```

```
/jcr:root/content/mysite/us/en//element(*, cq:Page)
[
  (jcr:contains(jcr:content, 'Digiquad')
  or jcr:contains(metadata, 'Digiquad'))
]
```

Number of hits: 4

Time: 0.02 seconds

- /content/mysite/us/en/service_oklama_header/harika, (crxde, html, json)
- /content/mysite/us/en/service_oklama_header/navatej, (crxde, html, json)
- /content/mysite/us/en/service_oklama_footer/gopal, (crxde, html, json)
- /content/mysite/us/en/service_oklama_footer/nirbhai, (crxde, html, json)

For even more detailed info, look at the Sling error.log. Set the logger for `com.day.cq.search` *to levels* `D`
`TRACE` *(plus filtering details).*

8→Based on the requestparam, we have wrote queries to retrieve the information from the jcr.

```
Map<String, String> map = new HashMap<>();
map.put("path", "/content/mysite/us/en");
map.put("type", "cq:Page");
map.put("group.p.or", "true");
map.put("group.1_fulltext", pagePath);
map.put("group.1_fulltext.relPath", "jcr:content");
map.put("group.2_fulltext", pagePath);
map.put("group.2_fulltext.relPath", "metadata");
map.put("p.limit", "-1");
```

MAP FUNCTION:
In order to store the pair values we can only use the Map data structure. Map datastructure helps us to store key pair values in it,
syntax:- `Map<String, String> map = new HashMap<>();`
By using put function we are giving the values to get stored in the Map.

QUERY EXPLAINATION:

1. path=/content/mysite/us/en
2. type=cq:Page
3. group.p.or=true
4. group.1_fulltext=Digiquad
5. group.1_fulltext.relPath=jcr:content
6. group.2_fulltext=Digiquad
7. group.2_fulltext.relPath=metadata
8. p.limit=-1

1.path=/content/mysite/us/en

path=/content/mysite/us/en ------- In this path parameter , This parameter specifies the starting path for the search. In this case, it starts the search from the **/content/mysite/us/en** location in the AEM repository.

2.type=cq:Page

type=cq:Page --------- We are telling that in this type we want only 'cq:page' type path only It ensures that only pages are included in the search results.

4.group.1_fulltext=Digiquad

group.1_fulltext=<your_page_path> ----------- This parameter specifies the full-text search query. We should**<page name >** replaced with the actual text you want to search for within the content of the pages.

5.group.1_fulltext.relPath=jcr:content

group.1_fulltext.relPath=@jcr:content ----------------- This parameter specifies the relative path within each page where the first full-text search should be performed. It indicates that the search should be performed within the **jcr:content** node of each page, which typically contains the main content of a page.

`6.group.2_fulltext=Digiquad`

group.2_fulltext=<your_page_path>   same as the above information given in the group_1 we used to same input or different when we give same input means it gives the same result on the query debugger --- but when we used to give the different output it will give the different result of the query

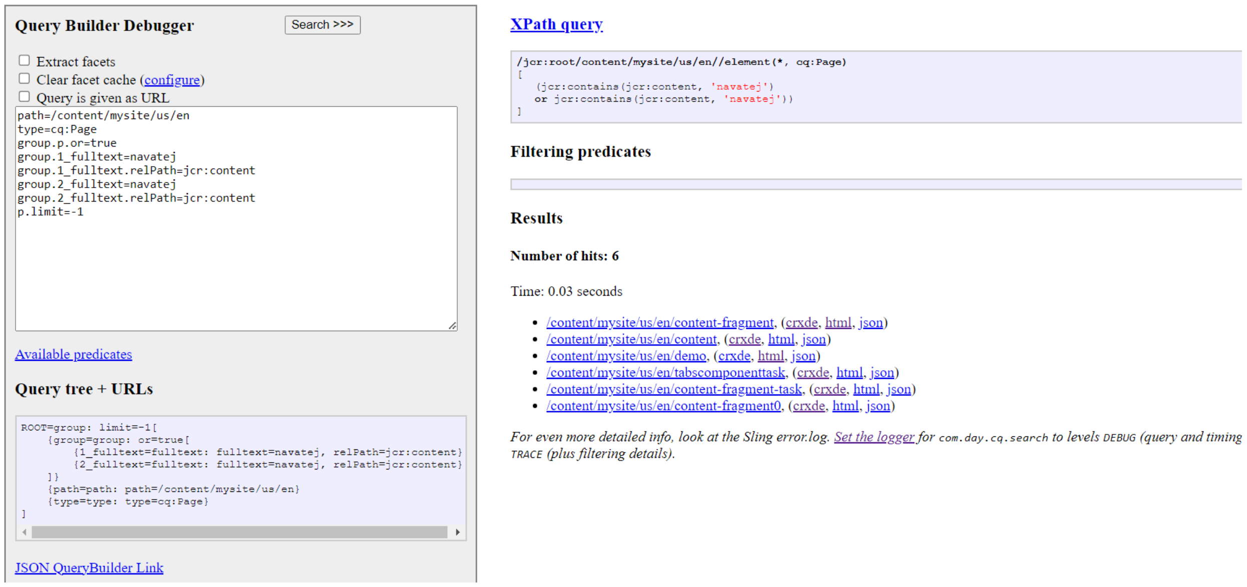
`7.group.2_fulltext.relPath=metadata`

group.2_fulltext.relPath=@jcr:content ----------------- This parameter specifies the relative path within each page where the first full-text search should be performed. It indicates that the search should be performed within the **jcr:content** node of each page, which typically contains the main content of a page.

`8.p.limit=-1`

p.limit=-1 -------- we use p.limit=-1 used to represents This parameter specifies the maximum number of search results to return. Setting it to **-1** means that there is no limit on the number of results, and all matching pages will be returned.

group.p.or=true



```
path=/content/mysite/us/en
type=cq:Page
group.p.or=true
group.1_fulltext=navatej
group.1_fulltext.relPath=jcr:content
group.2_fulltext=navatej
group.2_fulltext.relPath=jcr:content
p.limit=-1
```

Query tree + URLs

```
ROOT=group: limit=-1[
    {group=group: or=true[
        {1_fulltext=fulltext: fulltext=navatej, relPath=jcr:content}
        {2_fulltext=fulltext: fulltext=navatej, relPath=jcr:content}
    ]}
    {path=path: path=/content/mysite/us/en}
    {type=type: type=cq:Page}
]
```

XPath query

```
/jcr:root/content/mysite/us/en//element(*, cq:Page)
[
   (jcr:contains(jcr:content, 'navatej')
   or jcr:contains(jcr:content, 'navatej'))
]
```

Filtering predicates

Results

Number of hits: 6

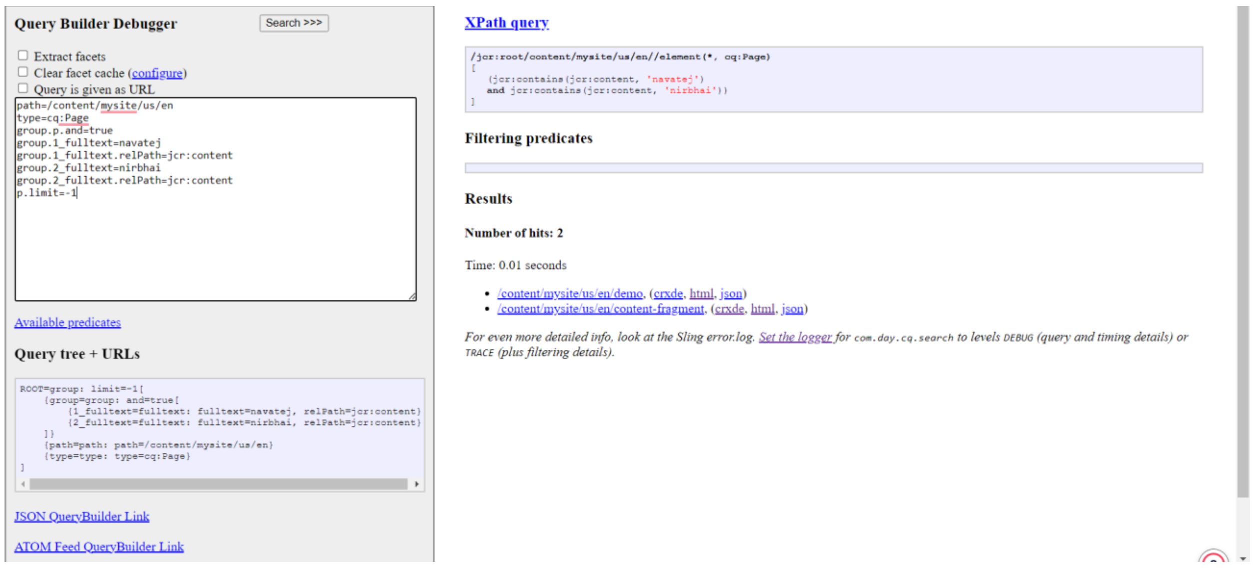
Time: 0.03 seconds

- /content/mysite/us/en/content-fragment, (crxde, html, json)
- /content/mysite/us/en/content, (crxde, html, json)
- /content/mysite/us/en/demo, (crxde, html, json)
- /content/mysite/us/en/tabscomponenttask, (crxde, html, json)
- /content/mysite/us/en/content-fragment-task, (crxde, html, json)
- /content/mysite/us/en/content-fragment0, (crxde, html, json)

For even more detailed info, look at the Sling error.log. Set the logger for `com.day.cq.search` *to levels* `DEBUG` *(query and timing* `TRACE` *(plus filtering details).*

From the above image when we used to give the same details in group_1 and group_2 it should render the parameters of that pagetitle. when we use group.p.or=true

group.p.or=false



```
path=/content/mysite/us/en
type=cq:Page
group.p.and=true
group.1_fulltext=navatej
group.1_fulltext.relPath=jcr:content
group.2_fulltext=nirbhai
group.2_fulltext.relPath=jcr:content
p.limit=-1
```

Query tree + URLs

```
ROOT=group: limit=-1[
    {group=group: and=true[
        {1_fulltext=fulltext: fulltext=navatej, relPath=jcr:content}
        {2_fulltext=fulltext: fulltext=nirbhai, relPath=jcr:content}
    ]}
    {path=path: path=/content/mysite/us/en}
    {type=type: type=cq:Page}
]
```

XPath query

```
/jcr:root/content/mysite/us/en//element(*, cq:Page)
[
   (jcr:contains(jcr:content, 'navatej')
   and jcr:contains(jcr:content, 'nirbhai'))
]
```

Filtering predicates

Results

Number of hits: 2

Time: 0.01 seconds

- /content/mysite/us/en/demo, (crxde, html, json)
- /content/mysite/us/en/content-fragment, (crxde, html, json)

For even more detailed info, look at the Sling error.log. Set the logger for `com.day.cq.search` *to levels* `DEBUG` *(query and timing details) or* `TRACE` *(plus filtering details).*

when **group.p.or=false** and different inputs are provided in **group.1_fulltext** and **group.2_fulltext**, the query debuger will only return pages that contain both "navatej" and "nirbhai" within their content (**jcr:content**). Both conditions must be met for a page to be included in the search results.

9→ **In order to perform query , We need 'QueryBuilder' dependency .'@Reference' used to inject the Querybuilder dependency into service class .**

```
@Reference
private QueryBuilder queryBuilder;
```

- **This piece of code creates the query**
- `Query query = queryBuilder.createQuery(PredicateGroup.create(map), resourceResolver.adaptTo(Session.class));`

- **This piece of code Executes the query**
- `SearchResult result = query.getResult();`

- **This piece of code Processes the results**

- By using a Iterator we will iterate over result(which is obtained from query executed) by using getResources(); and store it in the resourceIterator.

```
Iterator<Resource> resourceIterator = result.getResources();
```

- Enters a loop that iterates over each resource as long as there is a next resource available.

```
while (resourceIterator.hasNext()) {
```

- Retrieves the next resource from the iterator.

```
Resource pageResource = resourceIterator.next();
```

- Adapts the **pageResource** to a **Node** object. In AEM, a **Resource** represents content in the repository, and a **Node** represents a JCR node.

```
Node pageNode = pageResource.adaptTo(Node.class);
```

- Gets the child node named "jcr:content" under the current **pageNode**. This is commonly done to access the content node of a page in AEM.

```
pageNode = pageNode.getNode("jcr:content");
```

- Reads the value of the "jcr:title" property from the **pageNode**. This property typically represents the title of the content.

```
        String pageTitle =
pageNode.getProperty("jcr:title").getValue().getString();
                LOG.info("Found page title: {}", pageTitle);
```

- Initializes a new HashMap to store page information

```
Map<String, String> pageInfo = new HashMap<>();
```

- Puts the page title (**pageTitle**) into the map with the key "title"

```
pageInfo.put("title", pageTitle);
```

- Puts the path of the page resource (**pageResource.getPath()**) into the map with the key "path". This retrieves the path of the current resource

```
pageInfo.put("path", pageResource.getPath());
```

- Now we are adding this pageInfo in the foundPages variables which is initialized in the service class

```
          foundPages.add(pageInfo);
        }

    } catch (Exception e) {
        LOG.error("Error occurred while performing query", e);
    }
```

10→ Now we are having both the PageTitles and PagePaths in 'foundPages', coming to our requirement when i give a word in req param it should renders the page titles which is containing that word.

now our page titles are stored in the ‘foundPages’ variable, now in order to call this variable into frontend we have to define this variable in the model class.

```
private List<Map<String, String>> foundPages;
public List<Map<String, String>> getFoundPages() {
     return foundPages;
     }
```

11→ To call this foundPages variable in the frontend we have to define in the HTML file.

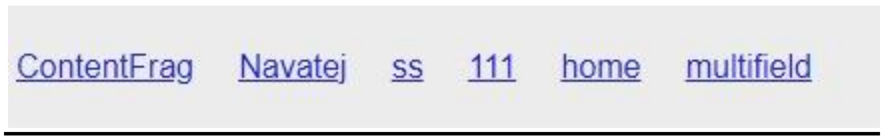
```
    <div data-sly-list.item="${usernameModel.foundPages}">

        <a href="${item.path}.html">${item.title}</a><br>

    </div>
```

data-sly-list:

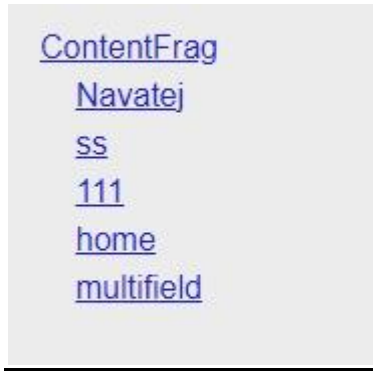
data-sly-list is used to iterate over a list of elements like array or a list, in our case it is foundPages. We have used a reference ‘item’ and stored all the pagePaths in it and printed .

<br> tag:

If we don’t define <br> tag in htl then the output look like below image



If we define <br> tag in htl then the output look like this image given below



ContentFrag
- Navatej
- ss
- 111
- home
- multifield