**AEM**

**Introduction:**

1. AEM Frontend:

1. Basic HTML, CSS & JavaScript

w3schools.com

2. AEM Backend:

1. Core java

1. UI/UX = design - html/css/js

2. AEM Dev = components/dialog I

CMS

-

content management system

create/manage the content

bhasaka.com

-> AWS S3

thehindu.com

-

100 articles (web page

-

html/css/js/images/document/videos)

without CMS:

* UI/UX - 4 hour - 400 hours
* 50 UI/UX

Time - 4 hours

* features (search/login/permissions/sco)

AEM 6.5 + SP 20

AEMASC

1. UI/UX 🡪 design - html/css/js

2. AEM Dev 🡪 components/dialog

3. AEM QA 🡪 testing

4. AEM CAT 🡪 Content Authoring

5. AEM Admin 🡪 - server

6. Marketing team 🡪 SEO/Analytics/Target

**AEMASC admin**

Cloud Manager

new server - 20 mins

**AEM 6.5 🡪2 days**

1. linux server setup

2. java

3. author

4. publish

5. apache web server

6. dispatcher

7. SSL

8. Domain configuration

1. jenkins

2. git

**Installation:**

A diagram of a company

Description automatically generated with medium confidence

**Content - web pages, assets, cf,xf,tags**

1. Sites

2. Assets

3. Page Editor

4. package manager - Dev/QA/CA

localhost:4502/crx/packmgr

localhost:4502 -> start screen -> tools -> deployment -> packages

prod author - 100 pages

localhost:4502 - 10 pages

- content package

- download

localhost:4502

5. CRX De

- VS Code / IntelliJ / Eclipse

JCR - Java Content Repository

- unstructured/tree database

- nodes/properties

localhost:4502/crx/de

localhost:4502 -> tools -> general -> crx de

/

apps - components,clientlibs,osgiconfigs,18n

libs - components,clientlibs,osgiconfigs,18n

etc  - clientlibs,osgiconfigs,18n

content - web pages, assets, cf, xf, tags

conf    -  templates, content (text,asset references)

AEM 6.5 vs AEM Cloud:

---------------------

1. admin is not required in AEMAsc

1. remote server

2. java

3. author

4. publish

5. ssl/domain

6. apache web server

7. dispatcher

8. replication configuration

1. creation

2. maintenance

3. pipeline/git

4. Auto Scalability

AEM 6.5 Prod

1 - author

3 - publish -> 20K -> 50K

8 - publish -> 50k

Dev

- forms module

- Sling Repo Init

- SCD  - Cloud Services

AEM 6.5 jar   -

AEM Cloud SDK - localhost:4502 -> code -> test code -> push the code into GIT

  - Cloud Manager CI/CD pipelines -> remote servers (dev,qa,stage,prod)

Cloud Manager

- domains/SSL

- fastly CDN

AEM Cloud:

Dev

- author   - domain

- devauthor.destaco.com/crx/de - read only

- code - CI/CD Deployment

- content - CURD

- publish  - dev.destaco.com

QA

- author   - domain

- publish  - qa.destaco.com

Stage

- author   - domain

- publish  - uat.destaco.com

Prod

- author   - domain

- publish  - www.destaco.com

content sync

AEM As Cloud -  5L

AEM 6.5 + 19 -  1.7L

1+2 - admins

**Consoles:**

**In AEMACS:**

1. author & publish server installation

2. connect author with publish

3. install wknd sample project

1. UI/UX 🡪 html/css/js

2. AEM Dev - component (code html, css, js, java)

3. AEM QA - testing

4. AEM CA content creation

1. Sites - - pages

localhost:4502/sites

replicate, activate, publish 🡪 move the content from author to publish

2. Page Editor – CURD operation on page

3. Assets Console

4. Package Manager

**Local Project Setup: Google > search > Adobe AEM project archetype**

mvn -B org.apache.maven.plugins:maven-archetype-plugin:3.2.1:generate -D archetypeGroupId=com.adobe.aem -D archetypeArtifactId=aem-project-archetype -D archetypeVersion=48 -D appTitle="News Portal" -D appId="newsportal" -D groupId="com.bhasaka.newsportal"

**groupid should be > com.<companyname>.<projectname>**

**Recommended to run the above code in command prompt as a “Run as administrator”**

**Create a folder where you want this project setup and go to the folder and run the below code:**



newsportal

frontend:   -> editors (VS Code/Eclipse/IntelliJ)

all

newsportal.all.<vid>.zip

newsportal.ui.apps.<vid>.zip

newsportal.ui.config.<vid>.zip

newsportal.ui.content.<vid>.zip

ui.apps

newsportal.ui.apps.<vid>.zip (Content Packages)

ui.config

newsportal.ui.config.<vid>.zip

ui.content

newsportal.ui.content.<vid>.zip

ui.frontend

localhost:4502/crx/packmgr

http://localhost:4502/crx/de

backend: -> editors (VS Code/Eclipse/IntelliJ)

core - 100 java files

newsportal.core.<vid>.jar  (Bundle)

localhost:4502/system/console

Go to the project folder and run the below command:

mvn clean install -🡪 To build a package(content and core)

mvn clean install -PautoInstallPackage  (Frontend/Backend code)       -> localhost:4502

mvn clean install -PautoInstallPackagePublish  -> localhost:4503

mvn clean install -PautoInstallBundle (Backend code)         -> localhost:4502

login

- components,java -> localhost:4502

file system  -> push -> Remote GIT Repo -> CI/CD -> Remote AEM Servers (dev,qa,stage,prod)

**Use VS code for front end code writing/editing (**Add AEM sync plugin under extension to sync between local file system to local AEM server**)**

**IntelliJ for back-end code writing/editing** (sync code using command mvn clean install -PautoInstallBundle or mvn clean install -PautoInstallPackage)

**article-details.html**

<h1 class="title"> Jailer Movie Trailer </h1>

<img src="/content/dam/newsportal/images/jailer-trailer.png"/>

<div class="desc">Release date of Rajinikanth and Mohanlal starrer 'Jailer' is  2023-08-11.</div>

A screenshot of a computer

Description automatically generated

**Content:** text, asset/page reference

**Code :** <h1 class="title"> </h1>

1. Node type of component is **cq:Component**
2. /apps/<projectname>/components

**Node Names:**

article-details

article\_details

articleDetails

Component Properties:

1. jcr:title
2. componentGroup

Component Child Node

cq:dialog

cq:editConfig

Allowed Component List  - template level configuration

**Components:**

JCR - Java Content Repository

- Storage (Database)

- unstructure/tree database 🡪 nodes and properties

localhost:4502/crx/de

code: apps (Custom component)

libs – Out of the box component

etc -

content:

custom code (components)

content -> web pages, assets, cf, xf, tags, forms

conf -> configuration & templates

var -> run time data

**article-details.html (HTML + HTL (Sightly))**

<h1> ${properties.articleTitle} </h1>

<div> ${properties. articleDesc} </div>

<img src="${properties.articleImage}" />

/apps/newsportal/components/article-details

sling: resourceSuperType 🡪 newsportal/components/helloworld

/apps/newsportal/components/helloworld

A screenshot of a computer

Description automatically generated

**Touch UI Dialog:**

Granite UI - collection of components to build the dialog and it's fields

ex: textfied

textarea

select

checkbox

pathfield

Coral UI

A screenshot of a web page

Description automatically generated

**Dynamic/Editable Templates:**

- N pages

Class:- Student

object

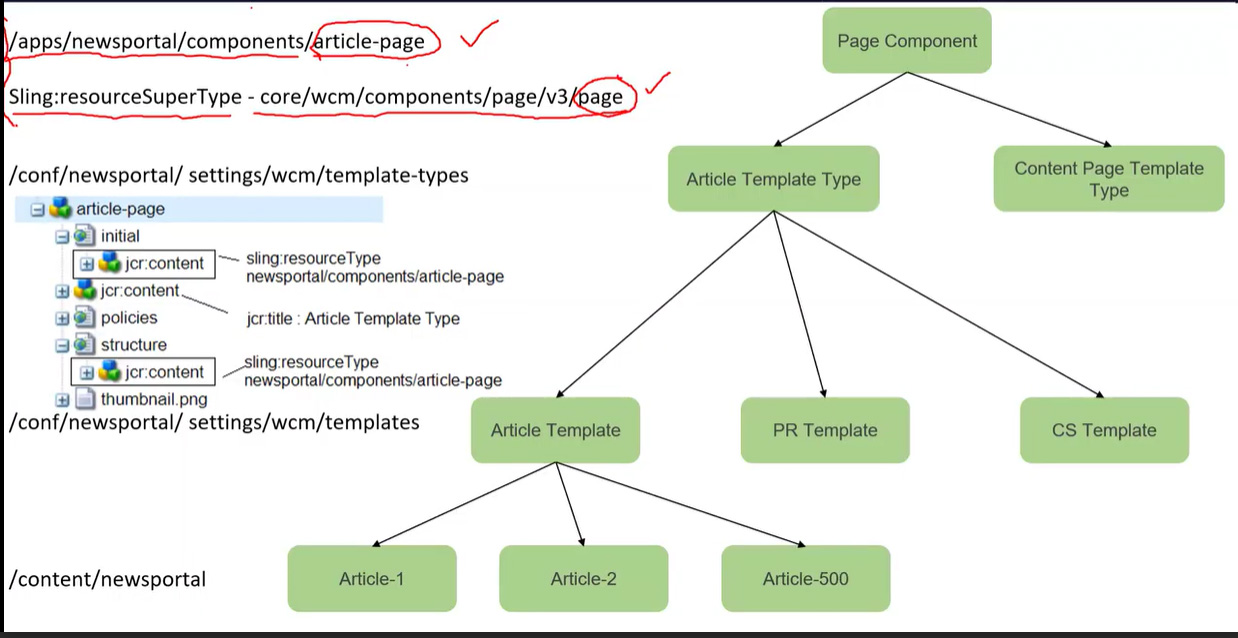
Student st1 = new Student (); Student st1 = new Student ();

1) Structure

define all the common components (ex: header and footer)

Article Template

20000 article pages



A diagram of a website

Description automatically generated

**Experience Fragment (XF):**

-------------------------

- small part of the web page

- cq:Page

- ex: header and footer

- define global/common content

/content

/experience-fragments

/newsportal

/us

/en

/site

/header (cq:Page)

master

jcr:content

root

top-bar

brand-logo

navigation

cs-header

jcr:content

root

top-bar

brand-logo

navigation

/footer

master

jcr:content

root

footer

footer-list-1

footer-list-2

footer-list-3

footer-list-4

footer-list-5

/conf

/newsportal

/settings

/wcm

/templates

/article-template

/structure

jcr:content

root

experience-fragment

fragmentContentPath : path of header master variation

/product-template

/structure

jcr:content

root

experience-fragment

fragmentContentPath : path of header master variation

/cs-template

/structure

jcr:content

root

experience-fragment

fragmentContentPath : path of header master variation

/newsportal

/us

/en

articles

article-1 (cq:Page)

jcr:content

root

container

experience-fragment

path - /content/experience-fragment/us/en/sites/header

   content page  -> editable template -> template type -> page component

   XF content

   -> XF Editable Tempalte : /conf/newsportal/settings/wcm/templates/xf-web-variation

   -> XF Template Type     : /conf/newsportal/settings/wcm/template-types/xf

   -> xfpage component     : /apps/newsportal/components/xfpage

2. article pages - 20000

ravi     - 5000

ramu     - 2500

ramya    - 7500

nishee   - 5000

article-1

writer-info

first name : ravi

last name  : kongari

email      : ravi.k@gmail.com

profile pic

/content/experience-fragment/us/en/sites/writer-info

master

- writre-profile

ramya

ramu

nishee

destaco.com/content/experience-fragments/newsportal/us/en/site/header/product-header.plain.html

**Content Fragments (CF):**

---------------------------------------------------

1. CF is an asset

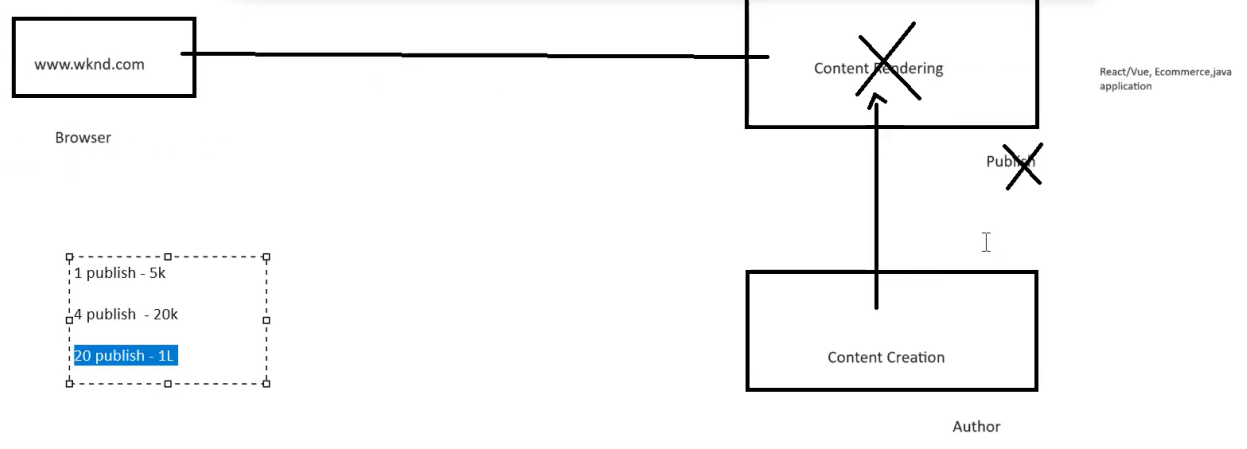
2. /content/dam

3. Primary Type: dam:Asset

1. Headless Approach

2. Headfull Approach

3. Hybrid Approach (headless + headfull)



Archiving Headless, using below 2 implementations in AEM:

1. Asset HTTP API

localhost:4502/content/dam/newsportal/content-fragments/author-profiles

http://localhost:4502/api/assets/newsportal/content-fragments/author-profiles.json

2. GraphQL

Tools > General > GraphQL > Click on create and select newsportal.

A screenshot of a computer

Description automatically generated

1. Editable Templates -> Web pages

2. CF Models (templates - dialog - (name,email,dob))  -> CF

/Content

1. web pages

2. assets

3. CF

4. XF

5. tags

-------- CF Vs XF --------------------

1. We can use CF & XF for global content (common)

2. CF & XF are having variations

3. CF & XF support to export data (cf- .json, xf - .html)

**Difference:**

CF – Render plain content without design

XF – Render content with design (styles, ex: header,footer)

CF – headless approach

XF - personalization (target), marketing (mobile, email social,web) global content (ex: header and footer)

new-user

vip-user

activate-user

super-activate

product-a

mobile

email

social

web

12 fields - CF Model

v1 - 10

v2 - 11

0. type -> type1, type2

1. mobile

2. phone

3. email

http://localhost:4502/content/assets/newsportal/content-fragments/author-profiles.mobiledata.json

**HTL: (HTML Template Language) (Sightly is a OLD name)**

* Model View Controller (MVC pattern)

A diagram of a web page

Description automatically generated with medium confidence

**HTL**

Advantages:

✓ Security

✓ Development efficiency

✓ Reduced Costs

✔Java knowledge is not required to write HTL

Expression:

The HTML Template Language uses an expression language to access the data structures that provide the dynamic elements of the HTML output. These expressions are delimited by characters **${ and }.** To avoid malformed HTML, expressions can only be used in attribute values, in element content, or in comments.

**<h1 class="${component.name}">${properties.jcr:title}</h1>**

**<h1 class="article-title" data-authorName="chamu"> AEM Cloud new fetures </h1>**

HTML Element- h1/sly

Attribute

* class/data-sly-test
* data-authorName

Text/Content -- AEM Cloud new fetures / ${}

**There are 13 HTL Custom Attributes/Block Statements: (First 8 are IMPORTANT out of 13)**

1) **data-sly-use** 🡪 create object for java, js, html and content nodes

Student st = new Student ();

**article-details**

article-details.html (html + htl)

<div data-sly-use.st="Student"> </div>

OR

<div data-sly-use.st=”com.bhasaka.newsportal.model.Student”></div>

2) **data-sly-include 🡪** to **i**nclude one html file into another html file

**article-banner**

writer-info.html

**article-details**

article-details.html 50 lines

<div data-sly-include="newsportal/components/article-banner/writer-info.html"></div>

<div data-sly-include="article-tags.html"></div>

writer-info.html - 25 lines html

article-tags.html

A screen shot of a computer

Description automatically generated

3) **data-sly-resource** 🡪 to include one component into another component

**article-details**

article-details.html

<div data-sly-resource="${'np-button' @resourceType @resourceType ='newsportal/components/button' }"></div>



A screenshot of a computer

Description automatically generated

**4) data-sly-test** 🡪 Similar to if condition. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If the title is available, then only the inside code will be executed. If the title is BLANK or NULL, the inside code will not be executed.

A screenshot of a computer

Description automatically generated

5) **data-sly-list** 🡪 Iterate list of objects (eg. Array, Any collection object (ArrayList, HashSet, HashMap))\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

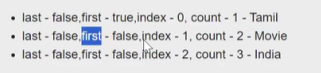
A screenshot of a computer code

Description automatically generated

* **If we not mentioned the variable, It takes variable name as “item” by default.**

A screenshot of a computer program

Description automatically generated



6) **data-sly-repeat** 🡪 It will repeat the host element and child element\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A screenshot of a computer code

Description automatically generated

**7) data-sly-template** 🡪 It is used to define the block of reusable content. Similar to function/method. Reuse the block wherever needed. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Define the template without passing the arguments. It’s recommended.

<div data-sly-template.myCode>

<h1>title</h1>

<p>description</p>

</div>

2. Define the template with passing the arguments. Arguments begin with **@.**

<div data-sly-template.myCode=”${@fName, lName, eMail }”>

<p>Firstname: ${fName}</p>

<p>Lastname: ${lName}</p>

<p>Email: ${eMail}</p>

</div>

A close-up of a white background

Description automatically generated

**For Dynamic:**

data-sly-call=”${writerInfo @fName=properties.firstName,@lName=properties.lastName}”

**IF THE TEMPLATE CODE/BLOCK IN DIFFERENT HTML FILE, WE HAVE TO CREATE AN OBJECT TO CALL THE TEMPLATE CODE/BLOCK.**

A computer screen shot of a computer code

Description automatically generated

**8) data-sly-call** 🡪 Used to call a defined template code/block \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<div data-sly-call=”${Name}”></div>

<div data-sly-call="${myDetails @fName='Anbu', lName='Ramu', eMail='anburuku@gmail.com'}"></div>

<div data-sly-template.myDetails="${@fName, lName, eMail}">

<p>FirstName:${fName}</p>

<p>LastName: ${lName}</p>

<p>Email:${eMail}</p>

</div>

**We can add clientlibs in 2 ways:**

1. Template policy
2. HTL

HTL

/libs/core/wcm/components/commons/v1/templates/clientlib.html

Templates:

js

CSS

all

<div data-sly-use.clientlib="core/wcm/components/commons/v1/templates/clientlib.html"></div>

<div data-sly-call="${clientlib.js @categories='newsportal.desktop'}"></div>

clientlib-desktop.js

<div data-sly-call="${clientlib.css @categories='newsportal.desktop'}"></div>

clientlib-desktop.css

<div data-sly-call="${clientlib.all @categories='newsportal.desktop'}"></div>

clientlib-desktop.css

clientlib-desktop.js

**9.data-sly-element:** Replaces the element name of the host element. Ie. replace the html element in dynamically from the content node value.

<h1 data-sly-element="${properties.titleType}">${properties.articleTitle}</h1>

**10. data-sly-text:** Replaces the content of its host element with the specified text.

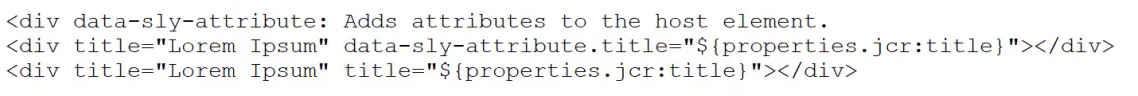
Ex: <p data-sly-text="${properties.jcr:description}">Lorem ipsum</p>

**Both are same:**



**11. data-sly-attribute:** Adds attributes to the host element.

Ex: <div title="Lorem Ipsum" data-sly-attribute.title="${properties.jcr:title}"></div>



**12. data-sly-unwrap:** removes html element from rendering 🡪 Instead of this we use <sly>

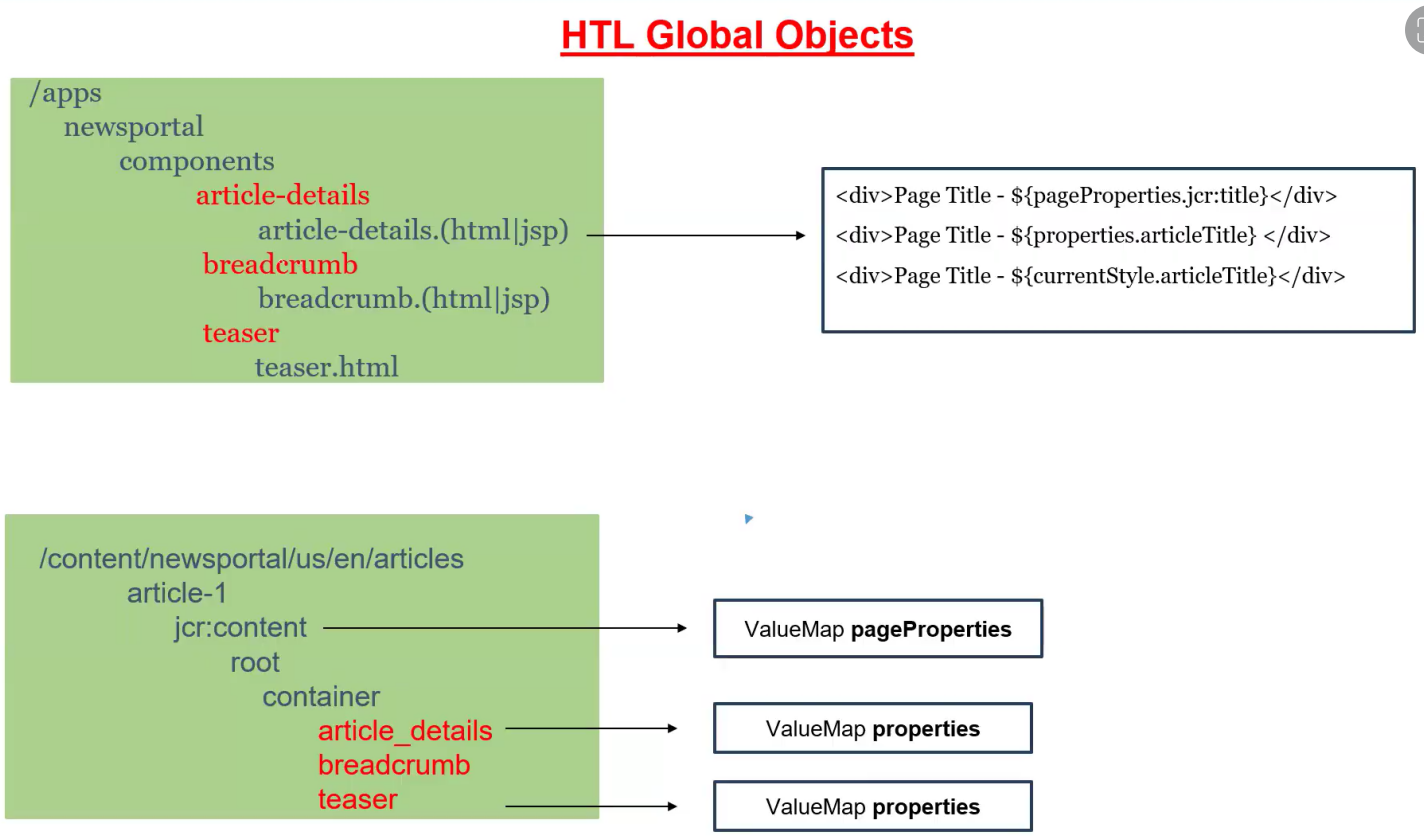
<p data-sly-use.nav="navigation.js">Hello World</p>

**13. data-sly-set:** define variable and assign values

<sly data-sly-set.title="${properties.articleTitle}" />

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a diagram

Description automatically generated

**HTL Context: (By default is @context=’text’)**

<h1> ${properties. articleTitle @context='html'} </h1>

jailer <b>movie</b> trailer

<a href="${properites. article Page @context='href'}"> ${properties.articleTitle @context='text'} </h1>

JS/CSS - context is mandatory

<script>

var articleTitle = ${properties.articleTitle @context='scriptString'};

var ${properties.variableName @context='scriptToken'} =’test’;

</script>

<style>

.article-titl {

content: ${properties.articleTitle @context='styleString'}

${properties.type @context='styleToken'}:'red'

</style>

1. JCR - Java Content Repository (Specification)

- Apache Jackrabbit is an implementation

2. Apache Sling - Request Processing

3. OSGI - Java Framework

- Specification

- Apache Felix is an implementation (Apache is a company, Felix is a project)

A 1.0.jar

A 2.0.jar

B.1.0.jar

A.1.0.jar

C.1.0.jar

A.2.0.jar

----------------------------------------------------------------------------------------------------------------

newsportal

core (100 java) -> newsportal.core.<VID>.jar (9.15 AM)

-> newsportal.core.<VID>.jar (9.30 AM)

jar file contains the below 3 things:

1. compiled java class (.class)

2. manifest.mf (metadata information)

3. resources (xml, json, small icon images, properties)

> mvn clean install 🡪 It will create a jar inside a target folder. If already jar file available on the target folder, new jar will be replaced.

**> mvn clean install -PautoInstallBundle > Move it localhost:4502**

**BundleActivator -> OSGI Interface**

**BundleActivatorImpl implement BundleActivator**

**1) start**

**2) stop**

**manifest.mf**

**Export-Package**

**Import-Package**

**Bundle-Activator**

**Private-Package**

**core.jar**

com.bhasaka.newsportal.core.services. Product InfoImpl.java

**Import-Package** com.bhasaka.newsportal.payment.services (Call the allowed package)

**payment.jar**

com.bhasaka.newsportal.payment.services. PaymentServiceImpl.java

**Export-Package** com.bhasaka.newsportal.payment.services (It allow to use the package to other.)

**Private-Package** : com.bhasaka.newsportal.payment.history (It will not allow to use/import other)

Difference between jar and bundle:

* Both are having .jar extension
* Bundle contains additional metadata headers eg. Export-Package, Import-Package, Private-Package.

**OSGI Components & Services:**

----------------------------------------------------------------------

OSGI Component:

- It is a Java Class

- OSGI Component can listen lifecycle events (activate, deactivate, modified)

OSGI Annotations:

-----------------

**Class Level:**

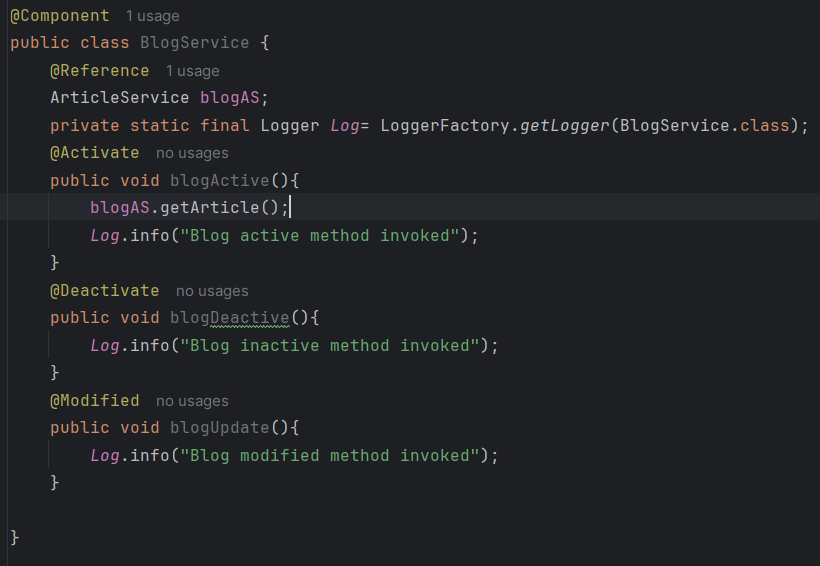
1. @Component  🡪 It will register a normal java class into OSGi component.

**Method Level:**

2) @Activate

3) @Deactivate

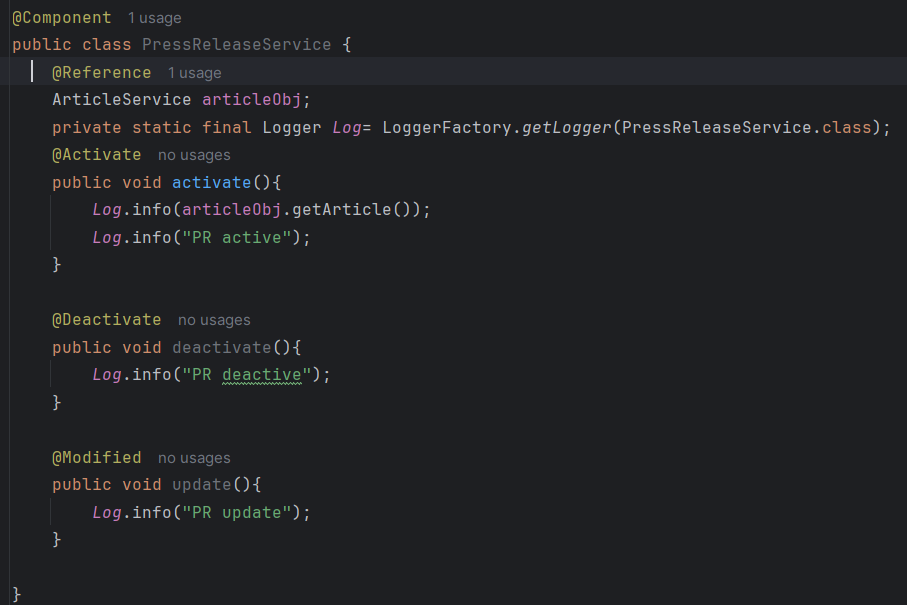
4) @Modified



5) @Reference – We call one component into another

@Reference will assign a dynamic object that is created by osgi container to

ArticleService articleObj;



If you want to use once OSGi component into another OSGi component, we have to convert that component into OSGi service component.

Eg.

@Component(service=ArticleService.class)

We can’t define OSGI service, without OSGI component.

OSGi service is an OSGi component. But OSGi component may or may not be an OSGi service.

ArticleService

@Activate

- activate()

@Deactivate

- deactivate()

@Modified

- update()

PressReleaseService

- activate

- deactivate

- update

newsportal.core.vid.jar

- activate

- Container Identify all OSGI Component java classes based on XML files inside jar file

- Register OSGI component java classes with OSGI container

- Creates Object for all OSGI Component Java classes

- Triggers activate event and execute activate method.

**IMPORTANT:**

1. All OSGI components are singleton () java classes.

2. We can't create object for OSGI components.

Browser - Client

  Request - <https://gorest.co.in/public/v2/posts>

Java

- Client

- Request

- Response

- convert response into string

**OSGi Dynamic Configuration:**

OSGi annotation:

**Class level:**

@Designate > Link the configuration class file with osgi component “ArticleService.class” file

@ObjectClassDefinition > used to create a dynamic osgi configuration class

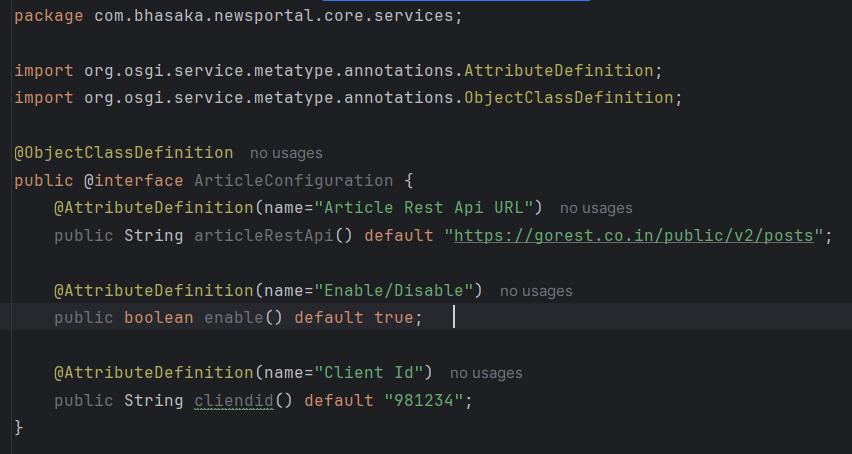
**Property level:**

@AttributeDefinition

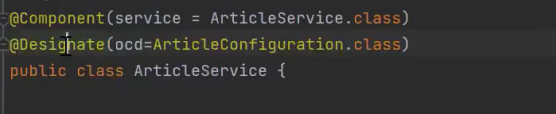
**Step 1:** Create a configuration class and define 3 properties that are make it dynamic.

A screenshot of a computer

Description automatically generated



Step 2: Link the configuration class file with osgi ArticleService.class file using @Designate



Step 3: We have to pass the dynamic configuration class as an argument to all lifecycle method (Activate, Deactivate, Modified)

Using the dynamic object **config**, call the method from ArticleConfiguration class. And assign it to instance variable.

A screen shot of a computer program

Description automatically generated



A screen shot of a computer screen

Description automatically generated

Step 4: We have to create a dynamic configuration file (JSON) under ui.config module.

ui.config

/apps

/newsportal

/osgiconfig

/config

com.bhasaka.newsportal.core.services.ArticleService.cfg.json (//cfg is a selector and json is an extension)

{ // These key : value pair taken from ArticleConfiguration.class file. Key is a dynamic property, go configuration class file take it from there. Value we can define.

“articleRestApi”:”https://www.google.com”,

“enable”: “true”,

“clientid”:”989898”

}

Terminal:

>cd ui.config

ui.config> cd mvn clean install -PautoInstallPackage

cd ../core

core> mvn clean install -PautoInstallBundle

Update Dynamic Configuratin:

----------------------------

1. CRX De

2. System Console Configuration

ui.config

/apps

/newsportal

/osgicongis

/config

com.bhasaka.newsportal.core.services.ArticleService.cfg.json

{

"articleRestAPI":"https://gorest.co.in/public/v2/posts",

"enable":"true",

"clientId":"8908978"

}

Runmodes:

---------

Standard/Installation/OOTB/Primary Runmodes:

--------------------------------------------

1. author

2. publish

3. samplecontent    - we-retail

4. nosamplecontent

aem-author-p4502.jar

java -jar aem-author-p4502.jar -gui -r nosamplecontent

author,samplecontent

author,nosamplecontent

publish,samplecontent

publish,nosamplecontent

Custom/Secondary Runmodes:

--------------------------

AEM 6.5:

--------

ex: dev,qa,stage,uat,prelive,live,prod,sit ....

AEM Cloud:

----------

dev,stage,prod,rde

1. To identify the server

2. To select runmode specific configuration.

1. java -jar aem-author-p4502.jar -Dsling.run.modes=dev

2. author

crx-quickstart

conf

  sling.properties (file)

sling.run.modes=qa

3. java -jar aem-author-p4502.jar -r prod