Software Development notes

Content Management

Content management is built on top of existing Engage Core (Harmony) project: https://bitbucket.org/engageft/backend/src/master/

There is RestController which is responsible processing request from Front End:

backend/onbudget/src/main/java/com/ob/ws/ContentManagementWS.java

There are two endpoints which service Front End request:

`com.ob.ws.ContentManagementWS#getContentByTag` - return all content (array) which is labeled with same CM Tag

`com.ob.ws.ContentManagementWS#getComponentById` - return single piece of content which is referenced by CM_ID

Both Endpoints fully configured with Swagger.

URL to access content:

Select Content by Tag:

GET /content/tag/test?platform_type=EV1& product_type=FSA& payer=Else& employer=SAIC& brand=TEST HTTP/1.1
Host: localhost:8080
cm_env: stage

Call above will select content which is marked with CM_Tag `test` (case insensitive). It will filter out content based on query parameters.

Method: `com.ob.ws.ContentManagementWS#getContentByTag`

Select Content by ID:

GET /content/id/TEST_LINK?platform_type=EV1& product_type=FSA& payer=Else& employer=SAIC& brand=TEST HTTP/1.1
Host: localhost:8080
cm_env: stage

Method: `com.ob.ws.ContentManagementWS#getComponentById`

Both calls give same response:

```
{
 "status": {
  "code": 0,
  "message": "Success",
  "errors": []
 },
 "content": {
  "component": [{
   "title": "TEST_LINK",
   "moderation_state": "edit",
   "nid": "35",
   "cm_id": "TEST_LINK",
   "text": "test/url/updated",
   "url": "/test/url/stage",
   "open_new_window": false,
   "type": "cm_link",
   "web_brand_id": "TEST",
   "class": "test class"
  }]
 }
}
```


Drupal

There are two Drupal instances configured for this project. When Content Management runs on local machine it connects to Drupal on AWS. It is considered as Local Drupal. Another Drupal instance is installed on Optum side and it is Optum Drupal.

When Content Management is running on local machine (developer's laptop), it connects to `Local Drupal`:

drupal.client.api.server.location=http://54.91.247.52/drupal2
drupal.client.api.version=/api/v1/
drupal.client.api.username= check local properties file

drupal.client.api.password= check local properties file

It is accessible from anywhere.

`Optum Instance` is only accessible from Optum VM.

drupal.client.api.server.location=http://drupal-engagecore-ofsnonprod.ose-elr-core.optum.com
drupal.client.api.version=/api/v1/
drupal.client.api.username= check dev properties file
drupal.client.api.password= check dev properties file

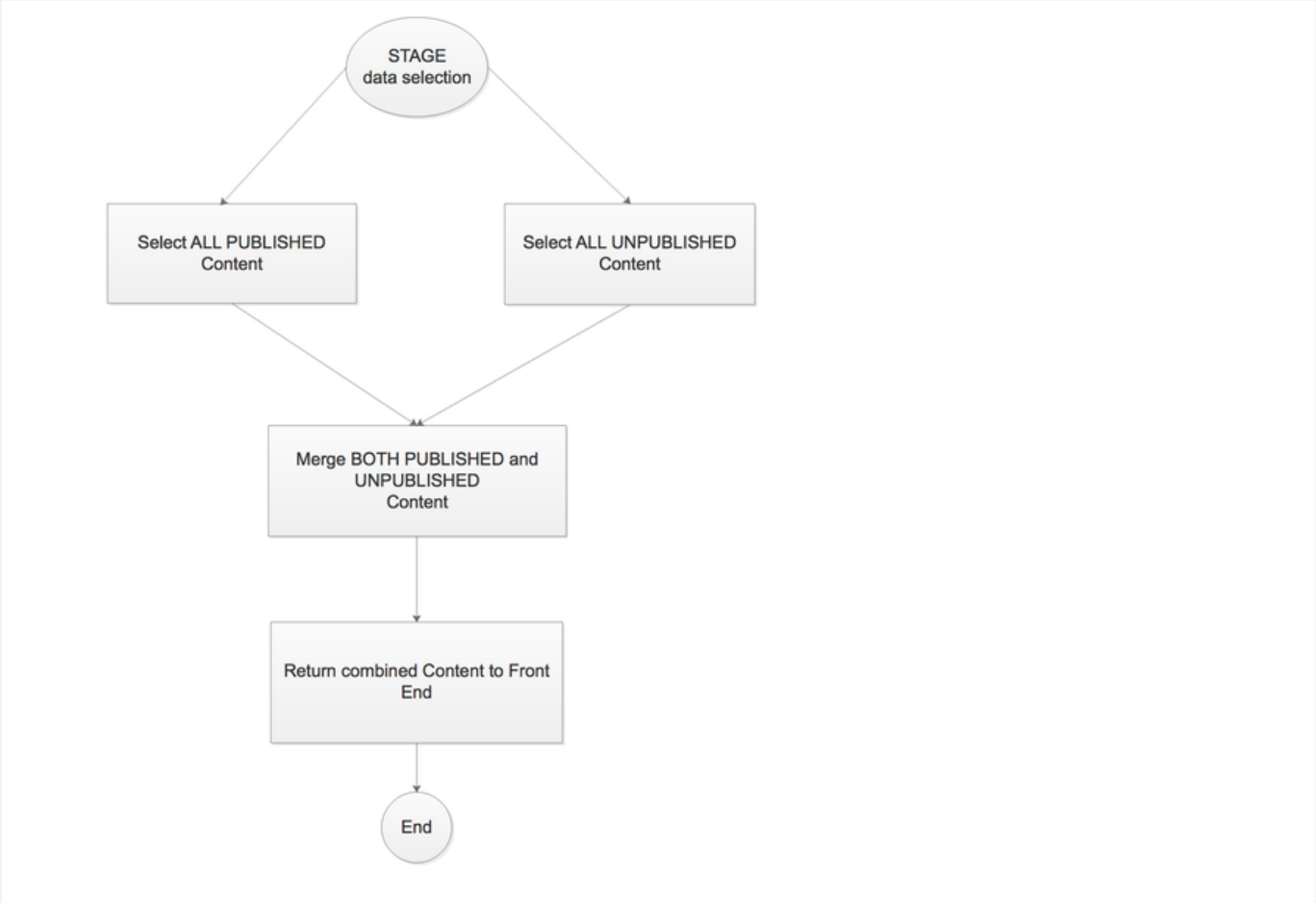
Both `Local` and `Optum` instances are configured the same. The difference is the content and cm tags. While `Local` instance is useful to test different content combinations, `Optum` instance is where Front End does its work creating custom content.

Selecting STAGE content vs PROD

Drupal has only one version of the content. In our case we need to serve latest version to `STAGE` and latest `published` version to `PROD`. They we are doing it is checking if component is `published`. If so, return `published` version. If the content not `published`, we are going into `Revision History` and look for latest `published` content there. There is HTTP header `cm_env` which defines if server returns `PROD or STAGE content.`

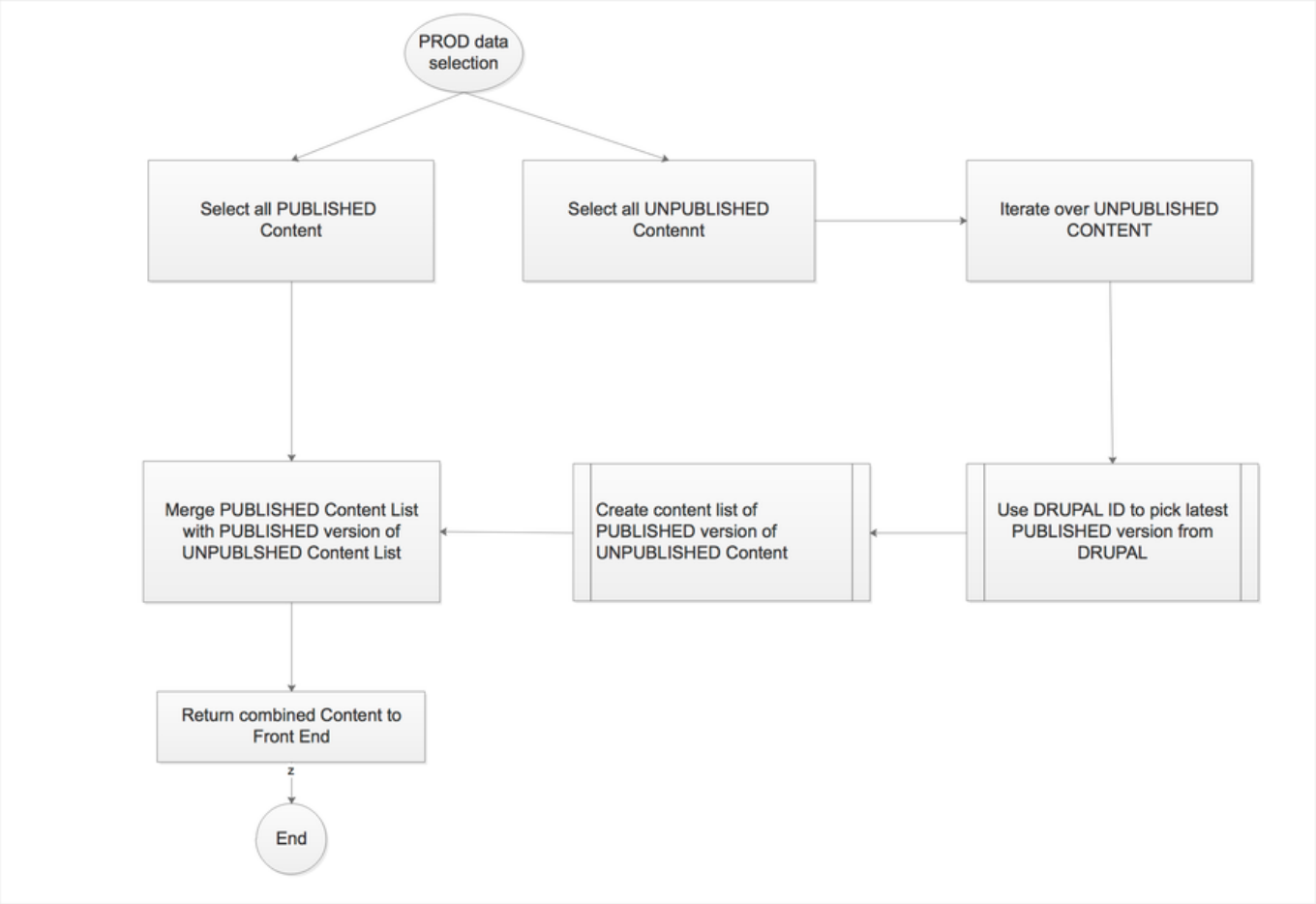
Stage data selection (includes unpublished):

When `cm_env: stage` is passed from UI, execute the following data selection.



Prod data selection (only published content):

When `cm_env: prod` is passed from UI, execute the following: data selection.

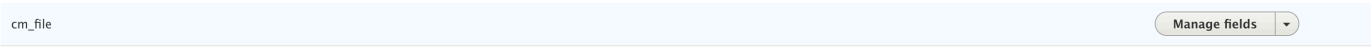


How to add field to component in Drupal

1. Login as Admim

2. Change to http://54.91.247.52/drupal2/admin/structure/types (points to AWS instance of Drupal)

3. Pick the content you want to edit. Click Manage Fields.



4. Add field to content type

5. Pick appropriate field type. More than likely the request is going to be from Front End team (Zack). Check with Zack what field type does he need.

6. Name the field starting with cm_******

For example, the image above shows the field 'cm_class'. Also record 'machine name'. For example if you just created field cm_class the machine name will be field_cm_class (like in example above).

You will need this name to map in Backend.

7. In Engage core project open application.xml file and find the mappings for the content type you are changing. For example, if you are changing cm_file, find the following map:

```
<util:map id="cm_file_map" map-class="java.util.LinkedHashMap">
<entry key="moderation_state" value="moderation_state" />
<entry key="nid" value="nid"/>
<entry key="type" value="type"/>
<entry key="field_cm_id" value="cm_id"/>
<entry key="field_cm_platform_type" value="platform" />
<entry key="field_cm_product_type" value="product" />
<entry key="field_cm_web_brand_id" value="web_brand_id"/>
<entry key="field_cm_file_description" value="file_description"/>
<entry key="field_cm_employer_id" value="employer_id" />
<entry key="field_cm_file_title" value="file_title" />
<entry key="field_cm_class" value="class"/>
</util:map>
```

For example, if you would need to add field cm_class you would add entry with key field_cm_class (what is Drupal machine name) and value = class. The value is what the backend send to Front End.