White Label Using Drupal Content Management

Summary

The main objective of this solution is to provide an ability for a non-technical, authorized user / customer to create and manage some aspects of a branded site.

Currently, supported configuration components are:

- site primary color (start and end)
- site secondary start
- site secondary end

The values above will be used for merging with a common css template to generate a brand specific style sheet file.

Flow

CSS Creation

- quartz job periodically polls Drupal for content type: WHITE_LABEL, filters on the ones that are in edit state and changed since last checkpoint. quartz poll interval is defined by variable
  drupal.client.wl.poll_interval=180000 (current value 3 min in milliseconds)
  backend will compare time white label content was changed to the time CSS compiled.
  If the difference more than 30 seconds CSS needs to be recompiled.

  There are four environments, DEV, TEST, STAGE and PROD. Each of them potentially have Quartz turned on. Since there is only one instance of Drupal it is not necessary to have them all running and triggering White Label recompilation. John Kaldor suggested to keep only one Quartz scheduler running in Production, since it is the most stable environment. There is a switch in properties file which allows to turn off the quartz scheduler for specific platform:

  drupal.client.wl.quartz_enabled=true
- if a WHITE_LABEL spec matched above, the job above will send a POST request to Ruby server to compile a css based on the params in the selected content node.
- Ruby gets the params from the compile call, merges params into the standard css template, compiles it into a css and returns the css as a stream back to the caller.
- The caller (java code) created MD5 hash on the received file, creates a name for that file e.g. [hash-code]_[web_brand_id]_style.css where web_brand_id is a field on the WHITE_LABEL content node; then, java code calls a REST API to store the file in IBM ObjectStorage using the generated name;
- java code then writes name back to Drupal into the same WHITE_LABEL node; this node must be in edit state otherwise the call will fail.

  REST call to recompile CSS:

```
curl -X GET \
http://localhost:8080/content/tag/white_label \
-H 'cm_env: stage'
```

Real-time CSS Fetch

- Ruby calls the content API providing web brand id and an input param.
- Java REST service pulls the flattened content structure; it also gets WHITE_LABEL content for the given brand id; results are merged; results include full URL to the css file.
- Ruby server uses the URL to insert into the link's href.
- Page renders.

Return JSON with URL:

```
// Ruby calls Java REST
curl -X GET \
  'http://localhost:8080/content/tag/wl_stylesheet?brand=DELL' \
  -H 'cm_env: prod' \

// Java responds with
{
    "status": {
        "code": 0,
        "message": "Success",
        "errors": []
    },
    "content": {
        "component": [
            {
                "web_brand_id": "DELL",
                "css_location":
"https://stg-account.optumbank.com/whitelabel-css/DELL_Sxoaliu67XxNFYSmq
cQQbA==.css",
                "moderation_state": "edit",
                "cm_id": "TEST_WL",
                "nid": "53",
                "type": "stylesheet"
            }
        ]
    }
}
```

Object Storage and Caching

IBM ObjectStore will have the actual css file and will make it available on HTTP GET. This will be fronted by NGINX where NGINX will route css requests to the ObjectStorage HTTP endpoint. NGINX will cache the retrieve file for future requests. Each file edit will have a different "hash" prefix and thus insure that the right content is shown.

Ruby CSS Compile Endpoint

`drupal.client.api.wl.cssCompilerUrl=https://dev-web-engageui-ofsnonprod.ose-elr-dmz.optum.com`

ObjectStore Spec

There is a production Object Store bucket created for Content Management:

drupal.client.api.object_store.host=s3api-dmz.uhc.com - host where Object Store is locationed
drupal.client.api.object_store.bucket=cap-whitelabel-css - name of the bucket in Object Store for white label CSS stored
drupal.client.api.object_store.access_key=p93MvdB4xIfks7Xcvwjl - access key. necessary to build Authorization Header. This key created when bucket is created
drupal.client.api.object_store.secret_key=PKXmbGL2AnlxOwwzXR5I0ORFfe45ca8KtKrR8qbY - secret key, necessary to connect to Object Store. This key is created when bucket created
drupal.client.api.object_store.bucket_location=/whitelabel-css - name of the bucket to use when Front End needs to reference CSS
drupal.client.api.object_store.bucket_server=https://stg-account.optumbank.com - this server name is only used on DEV and TEST environments so Front End can develop using absolute path. STAGE and PROD are going to be using relative path to the bucket.

Stage VS Prod Envs

For css to be successfully compiled and propagated through the system the content node must be in edit state. ALL versions of css will be stored in the ObjectStorage. However, for staging calls, due to a env header in the API call, java code will return the latest revision that is edit state or published if the last one. The prod calls will always get the last revision that is in "published" state. ASSUMPTION: the domain name and path of the CSS resource will be THE SAME for all instances (PROD/STAGE) and all brands.