FLAME Admin Portal UI – Developer Guide

Version 1.0.0

05 February 2021

Notices

Following are policies pertaining to proprietary rights and trademarks.

Proprietary Rights

Trademarks

Trademark notices and symbols used in this document reflect the registration status of Mastercard trademarks in the United States. Please consult with the Customer Operations Services team or the Mastercard Law Department for the registration status of particular product, program, or service names outside the United States.

All third-party product and service names are trademarks or registered trademarks of their respective owners.

Disclaimer

Mastercard makes no representations or warranties of any kind, express or implied, with respect to the contents of this document. Without limitation, Mastercard specifically disclaims all representations and warranties with respect to this document and any intellectual property rights subsisting therein or any part thereof, including but not limited to any and all implied warranties of title, non-infringement, or suitability for any purpose (whether or not Mastercard has been advised, has reason to know, or is otherwise in fact aware of any information) or achievement of any particular result. Without limitation, Mastercard specifically disclaims all representations and warranties that any practice or implementation of this document will not infringe any third-party patents, copyrights, trade secrets or other rights.

05 February 2021

Table of Contents

1 Introduction

1.1 Purpose and Scope

This document provides all key functionalities and steps required to build the Masterpass Wallet application front-end components using Masterpass Wallet Builder (MWB) API services.

1.2 Revision History

Date	Version	Description	Author
05 Feb 2021	1.0	Initial release	Mastercard

1.3 Intended Audience

- This document is intended for software developers who are developing MWB Admin Portal UI application front-end components using Admin Portal backend API services. The document is written with the assumption that the reader is familiar with the following:
- MWB Admin Portal UI

1.4 Pre-requisites

The document assumes that the reader is well versed with AngularJS front-end web application framework. It is assumed that the reader has read and understood the Reference documents.

1.5 References

- flame_mwb_admin_portal_user_guide_v1_0.pdf
- flame_cws_admin_portal_application_installation_guide_v2_2_1
- flame_admin_portal_server_workspace_setup_guide.docx
- flame_admin_portal_server_developer_guide.docx

1.6 Glossary

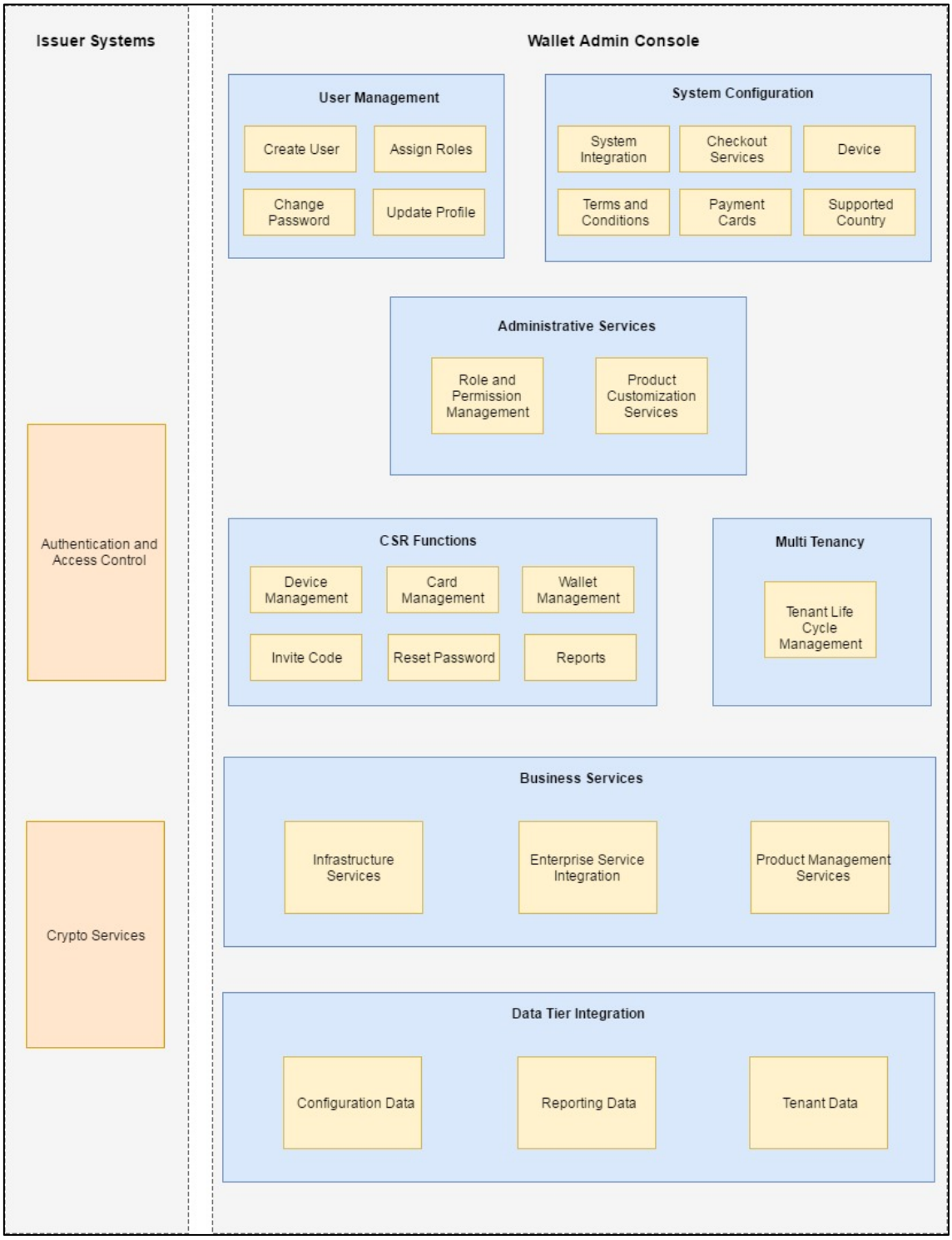
This section explains and defines words and acronyms that are used throughout this document.

Term/Acronym	Description
MWB	Masterpass Wallet Builder
API	Application Programming Interface
HTTP	Hyper Text Transfer Protocol
HTTPs	Secure Hyper Text Transfer Protocol (HTTP/SSL)
JSON	Java Script Object Notation

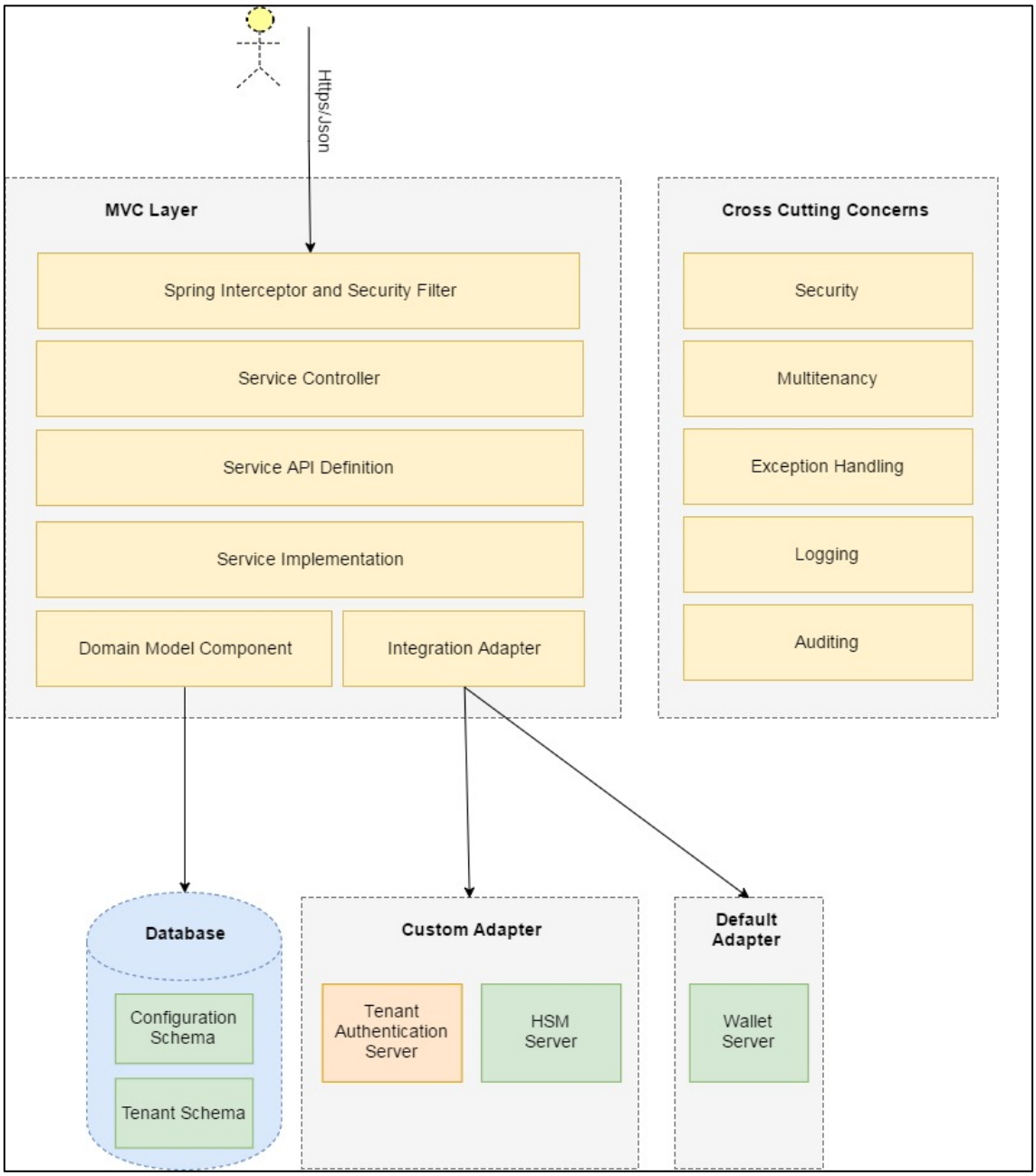
Term/Acronym	Description
OWASP	Open Web Application Security Protocol
REST	Representational State Transfer
UI	User Interface
URL	Uniform Resource Locator
W3	World Wide Web
WS	Wallet Server. Also known as CWS (Corporate Wallet Server)
CaaS	Crypto as a Service
CSR	Customer Support Representative

2 Technical Overview

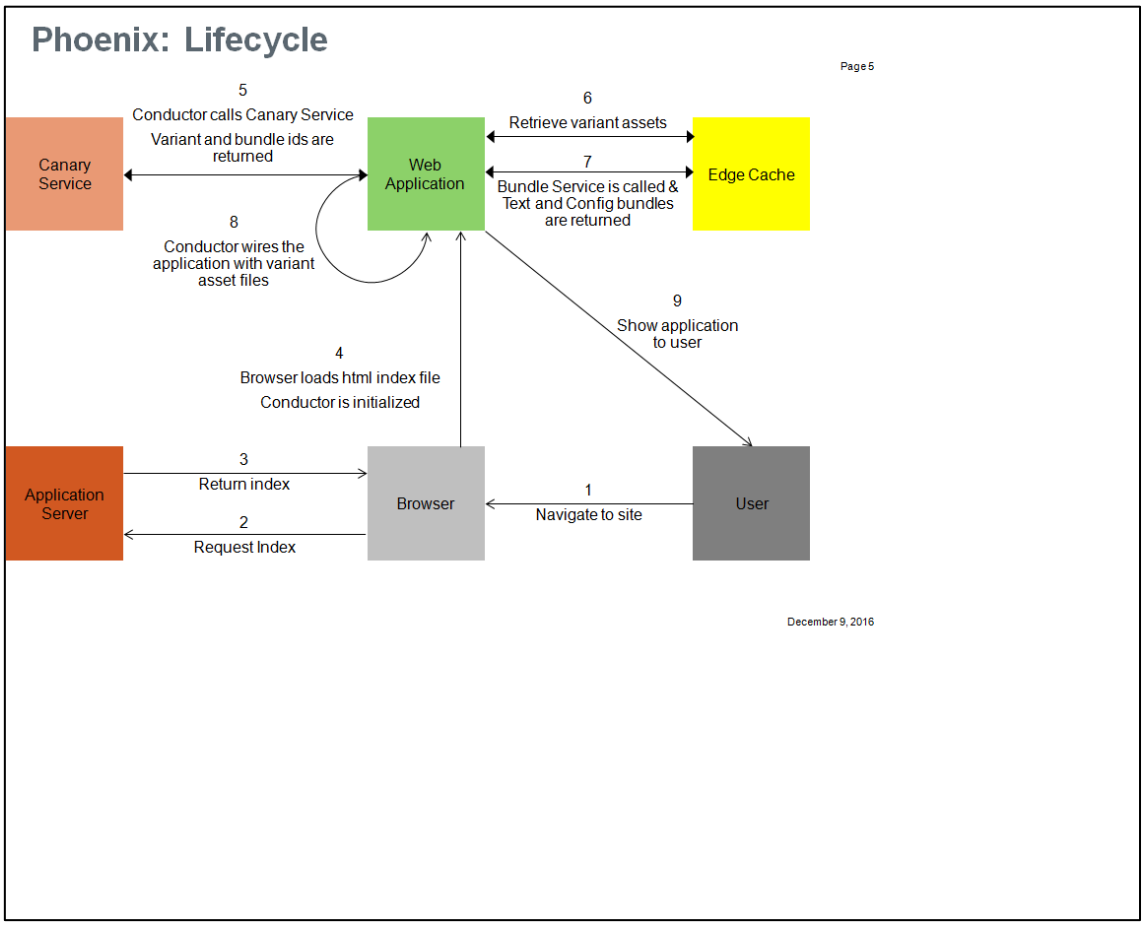
2.1 Functional Overview



2.2 Architecture Overview



2.3 High Level Diagram of Execution Flow



3 Development Environment Setup

3.1 Overview

To get started, setup the development environment as described in the subsequent chapters.

3.2 Pre-requisites

To setup the development environment, developer should configure the machine with given hardware configurations:

- 8 GB RAM
- Windows 7 and above, Linux operating system
- 2.80 GHz Processor

To set up development workspace, developer should have following tools installed in the system:

IDE (Integrated Development Environment)	Any IDE like Visual Studio Code, Eclipse Luna 4.4.2, WebStorm.
Application Server	JBoss-Eap-6.4.13 and above
Programming Language	Angular JS (1.x)
Build Tool	Maven 3.0, Gradle 3.3
Database	Oracle 11g and above or PostgreSql 9.4
Repository	Bitbucket

3.3 Setup and Dependency

3.3.1 Setup Environment Variables

1. Set JAVA_HOME path to <JDK_HOME_PATH>.
2. Set M2_HOME path to <MAVEN_HOME_PATH>
3. Set GRADLE_HOME path to <GRADLE_HOME_PATH>

3.4 Workspace Setup on IDE

1. Install XCode
    - can get this through the Apple Store or Self Service
2. Install node.js version 6.10.0 via terminal: ~> **nvm install v6.10.0 -g**
    - Install through Node Version Manager (nvm) so you can easily switch node version - https://github.com/creationix/nvm
    - Install from brew - http://blog.teamtreehouse.com/install-node-js-npm-mac (via brew tap homebrew/versions && homebrew/versions/node4-lts not just 'brew install node' so you can get 6.x)
    - Install directly from - https://nodejs.org/en/ (not recommended)
3. Download source code zip file from bundle folder.
2. Unzip the folder and cd into source folder via terminal.
3. disable npm ssl validation: **npm config set strict-ssl false** (or perhaps allow MC CA https://docs.npmjs.com/misc/config#ca)
4. Set the config registry to npmjs.org via:- **npm config set registry https://registry.npmjs.org/**

   Verify the registry via:- **npm get registry**
5. run **npm install -g bower gulp@3.8.11** nodemon (may need to run as sudo user)
    - this will install above node dependencies globally
6. run the **npm install -g bower-art-resolver**
    - this will install bower dependencies globally
7. run **npm install** (may need to run as sudo user)
    - this will install node and bower dependencies
8. [optional] When prompt by bower to choose a version of angular, 'Unable to find a suitable version for angular, please choose one:' choose number 2 which is angular#~1.3.8. Some version of angular doesn't work.
9. Run **'gulp clean-code'**,

   'gulp build' or **'gulp serve'** to verify that project is working.
10. 'gulp or gulp help' will list all gulp task.
    - Refer to these instructions on how to not require sudo

3.4.1 Troubleshooting steps

In case you face any error while setting-up workspace, try with fresh installation by removing cache and previously installed modules from the system.

Some of the useful commands like,

1. Uninstall nvm and npm from the system (mac)
2. Remove cache (references can be at multiple places)
    a. /Users/<username>/.cache/bower
    b. /Users/<username>/.npm/
    c. /Users/<username>/.nvm/
    d. /private/var/folders/9m/h1vlwvx12kx3vjghfsmw6l0j6k07fg/T/npm-*
3. Re-install nvm-sh again
    a. curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.37.2/install.sh | bash
4. Re-install 6.10.0 version
    a. nvm install v6.10.0 -g (it will take time)
5. Follow the steps from developer guide - section 3.4 - (PDF attached)
6. Try with “npm install --loglevel verbose”
    a. In case you get error in npm install, try again by removing “node_modules” and “bower_components” folders.

3.5 Running the application

3.5.1 Tests

- Run the unit tests using `gulp test` (via karma, mocha and sinon).
- Run the project with `gulp serve`
- Opens it in a browser and updates the browser with any file’s changes.
- Build the optimized project using `gulp build`
- This creates the optimized code for the project and puts it in the build folder
- The build folder is not ignored since it contains the deployable artifacts.
- Run the optimize project from the build folder with `gulp serve-build`
- `gulp help`

3.6 Gulp Tasks

3.6.1 Task Listing

Displays all of the available gulp tasks.

3.6.2 Testing

- `gulp test`

Runs all unit tests using karma runner, mocha, chai and sinon with phantomjs. Depends on vet task, for code analysis. More information on test task below.

3.6.3 Serving Development Code

- `gulp serve`

Serves the development code and launches it in a browser. The goal of building for development is to do it as fast as possible, to keep development moving efficiently. This task serves all code from the source folders and compiles less to css in a temp folder.

3.6.4 Serving Production Code

- `gulp serve-build`

Serve the optimized code from the build folder and launch it in a browser.

- `gulp serve-build --debug`

Launch debugger with node-inspector.

3.7 Code Commit and Tagging in Repository

A central repository is necessary to manage a project where multiple developers are developing some functionalities. It is recommended to use <TBA> as a repository and below are some processes being followed while developing an application:

The main application development should be done in Master Branch. The Master Branch is the main branch and should always be up to date.

For developing a new feature, there should be a feature branch. The feature branch is used for development of features and should be merged with Master Branch once the feature is developed and tested successfully.

The code base should be tagged for all major or minor releases. The tags are used for releases and every release should have a tag. The development should not be done in a tag workspace and release should not be done from feature or Master Branch.

Issues occurred in a feature developed in older releases should not be fixed in the Master Branch directly. A new feature branch should be created from the tag and issue should be fixed in that branch. The fix should be merged in Master Branch after successful development and testing done.

Once the merging with Master Branch is completed it should be tested again.

4 Development Process

To create a module in project folder follow below path

admin_portal_console_forked>src >client >app >variant > default >modules

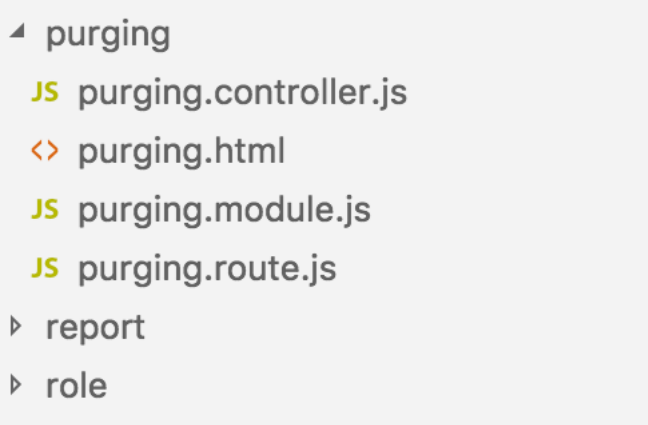
Create a folder and place below files: -

- **module-name.module.js**

```
country.controller.js
(function () {
    'use strict';

    angular
        .module('country')
        .controller('CountryController', CountryController);

    function CountryController(logger, countryFactory, $state, DTOptionsBuilder, DTColumnDefBuilder, displayError, modal, $filter, lang) {

        var vm = this;
        vm.dataTableOptions = DTOptionsBuilder.newOptions().withPaginationType('full_numbers');
        vm.dataTableColumnDefs = [
            //Below code will remove sorting from action column
            DTColumnDefBuilder.newColumnDef(11).notSortable()
        ];
        vm.viewDetailPage = viewDetailPage;
        vm.editDetailPage = editDetailPage;
        vm.deleteCountryDetail = deleteCountryDetail;

        activate();

        /** @function activate
         * @description Default executable
         */
        function activate(){
            getCountryCode();
        }
```

- **module-name.route.js**
- **module-name.controller.js**
- **module-name.html**

  For eg:- Purging Module



4.1 Adding CSS

To add styles follow below path

admin_portal_console_forked>src >client >app >variant > default >styles>custom.css

All the styles define by developer at the time of development is required to be defined in custom.css file only.

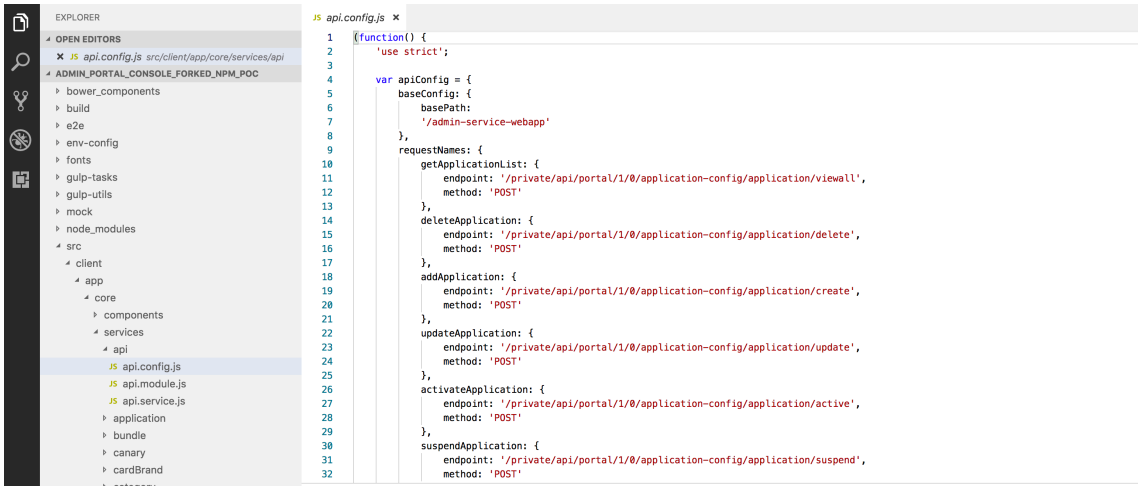
```
@font-face {
    font-family: "Mark-MC";
    src: url("/fonts/MarkForMCNrw.ttf") format('truetype');
}
.logo-container {
    padding: 7em 0em 1.5em 0em;
}
.login-div {
    background-color: #404040;
    color: #FFF;
    text-align: center;
    padding: 1.7em;
}
.login-div h3 {
    margin-top: 5px !important;
    margin-bottom: 15px !important;
}
```

4.2 Making Data Configurable

To add config data follow below path

admin_portal_console_forked>src >client >app >core > services >api>api.config.js

All the API URL and the data or function that need to be configurable and can be used throughout the application is needed to be defined in config file.



4.3 Configurable Headings and Label's

The labels and headings appearing on the application screen are kept configurable i.e those can be easily changed just by making a change in one single file throughout the application without editing any of the HTML code.

We have an **app.text** file which maps all the static data that appear on screen in form key value pair. The key contains the name of variable and the value contains the exact data that needs to be shown on the screen. (http://screen.by/)

For localization support files follow below path

admin_portal_console_forked>mock >bundle >text.json

```
{
    "loginUserID": "User ID",
    "loginPassword": "Password",
    "loginSubmitButton": "Login",
    "loginForgotPasswordLink": "Forgot Password?",
    "loginUserIDRequiredErrorMessage": "User ID is required",
    "loginUserIDPatternErrorMessage": "Provide valid User ID",
    "loginPasswordRequiredErrorMessage": "Password is required"
}
```

To access these defines variables in we need to bind these in HTML.

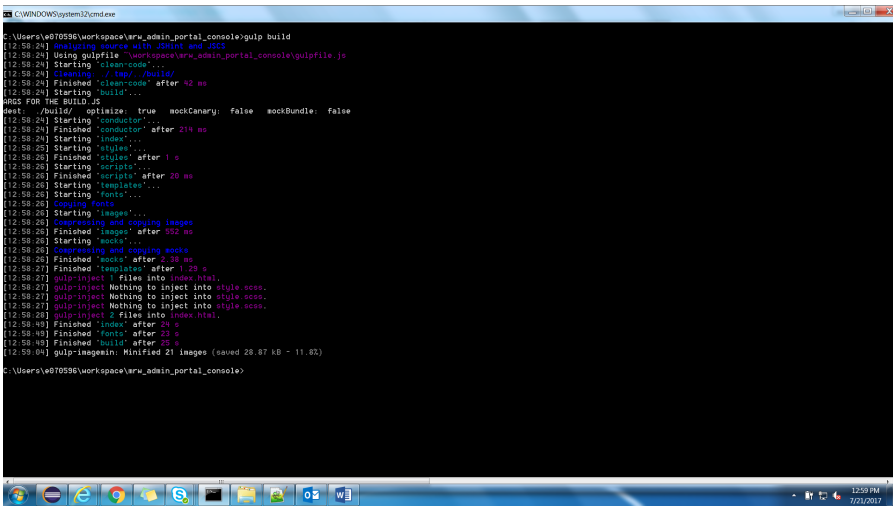
```
<div id="wrapper">
<!-- Page Content -->
    <div id="page-content-wrapper">
        <p>{{app.text.loginUserId}}</p>
    </div>
</div>
```

5 UI Build Process

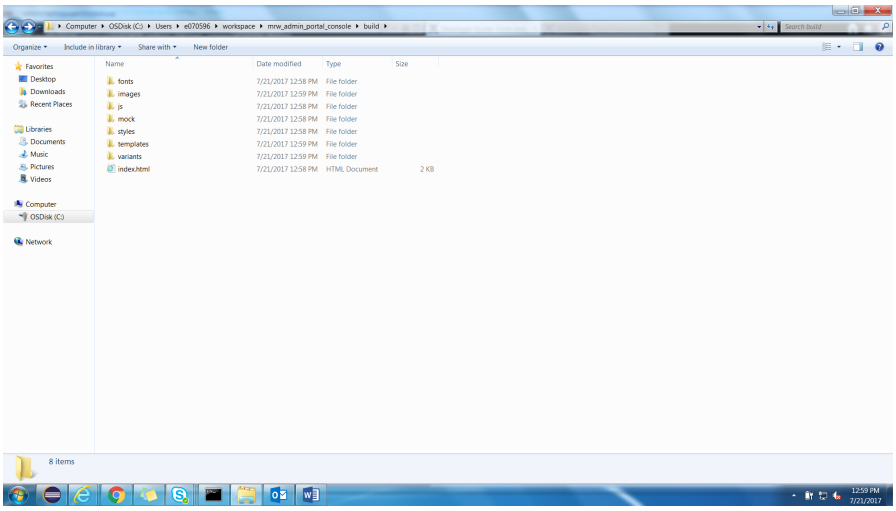
Note - If project path needs to change then update is required in gulp.config.js file, change should happen on basepath variable value. Current value is 'adminweb/' of basepath.

Eg. basepath: 'adminweb/',

To create build we need to execute "**gulp build**" command and it will create build folder inside project directory



Use build folder to create war file



To export the war via terminal use below steps:-

>cd build

> Rename index-<random_number>.html file to index.html

> jar -cvf adminweb.war *

The above command creates the deployable war file

Export the build in .war file.