High Level Design Document

TF Form Validations Flow

Mar 10, 2020

**Updated: 03/12/20**

**Overview**

This document provides the high level design of the Tax Factory Form Validations as per the high level analysis of Tax Factory Application pages.

**Scope**

* Define Metadata for form validations.

**Assumptions**

* Existing Data model from will be used for Permissions storage.
* Browser local storage mechanism would be used for User Preferences.
* Security Service needs to enhance to TF authentication support.

**Constraints**

* Module Descriptor needs to provide valid information to support module in MAC.
* Individual module would provide bundled/generated build file and any other artifacts to support the module at run time along with Module Descriptor.

**Definitions**

* UI – User Interface
* MD – Module Descriptor
* MAC – Module Application Container
* MVC – Model View Architecture
* JSON – JavaScript Object Notation
* CF – Compliance Factory
* TF – Tax Factory
* SWS – Security Web Service

**Areas Impacted**

* Security Web Service needs to updated to support TF module authentication (Not a part of this design)

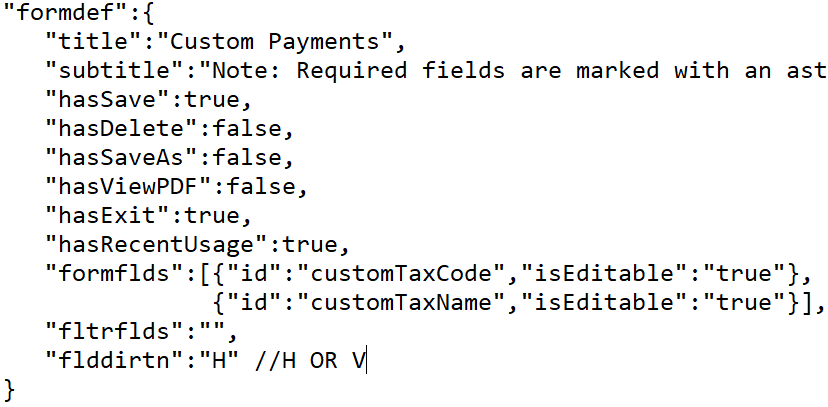
**High Level Design**

**Form Validations Flow**

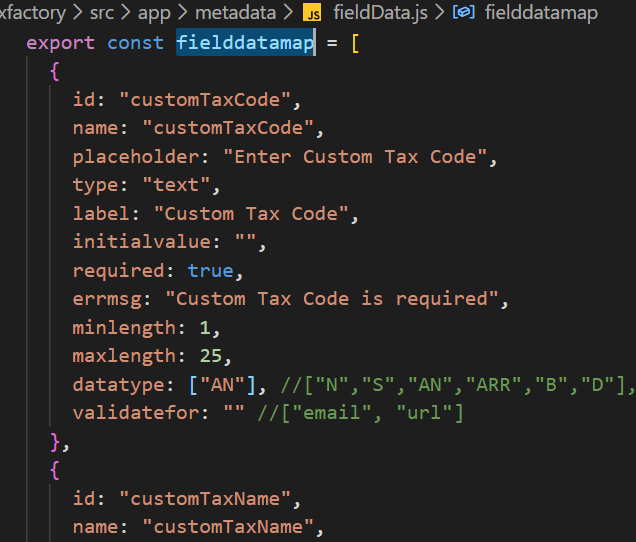
Form Validations flow would utilize Grid Metadata with all read only columns. Following diagram illustrates [Form & Validation Metadata](#RO_Component_Flow_Ills).

**Form Validations Prerequisite**

* Form & validation JSON Metadata definition. For e.g.: [Metadata Sample.](file:///\\ntsrv\common\Strategic%20Solutions\Designs%20&%20Specs\TF%20New%20Arch\Analysis\JSON-Metadata-Schema\HLD-Flow\Type1-Grid-with-Form-data.txt)
* Form Validation Library Bundle (.css, images, .js etc.) For e.g.: bsiformvalidation.js
  1. Metadata for the page has “**formdef”** (Form definition) attribute defined with “**formflds:[]**” and or “**fltrflds:[]**” for CRUD form and Form filter if needed along with the other required buttons, title, subtitle etc. For e.g.:

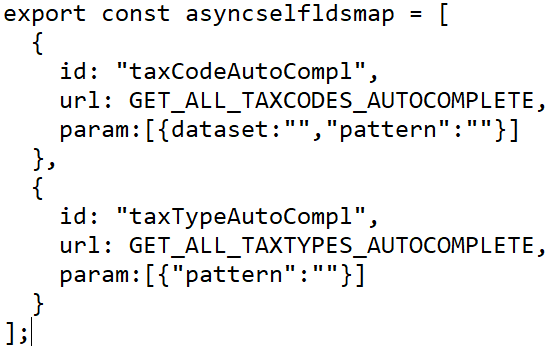


* 1. Library bundle should have all (variety of fields types) the fields defined in field data map. For e.g.:



**Form Validation Flow**

* When User click Edit on Grid component or click like Filter form pass form definition from the metadata as per pgid to <***ReusableForm***/> modal component.
* Based on the ***formflds*** or ***fltrflds*** load the field definition for the all the fields from **fielddatamap** and (for e.g.: FieldGenerator.js) generate fields for display and render fields.
  1. Based on “***flddirtn***” value (H or V) render form layout to Vertical or Horizontal direction.
  2. For every field id get the field details from **fielddatamap** and render.
  3. If ***type*** of field is “select” and ***isasync*** is true then load options data using url from **asyncselfldsmap** based on field id. For e.g.:

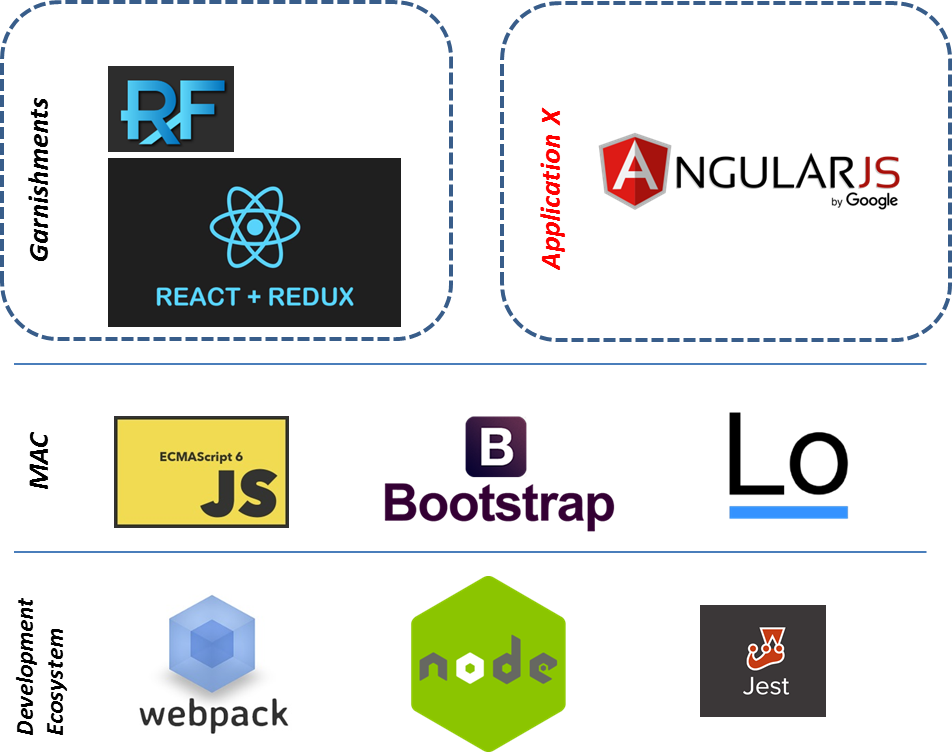


* 1. If ***type*** is select & ***options*** are provided in **fielddatamap** for the field then use that data for options rendering.
  2. If ***type*** is select & ***autocomplete*** is true then render appropriate select component with autocomplete on.
* Utilize initial values for the all the available fields and generate initial value object for form when loading form with ***mode*** as **New**.
* For **Edit** mode set values from the props/state to generated fields.
* Generate validation schema object based on validation requirement of available fields from metadata and (FieldSchemaBuilder.js) generate validation schema using Yup validation library.
* If required attribute is ***false*** for the fields then skip validation schema generation.
* If required attribute is ***true*** for the fields but datatype validation is not required and or any specific ***validationfor*** attribute is not present then generate basic validation schema and supply it to the form.
* Set initial Values and validation schema on the form.

**Form Validation Flow Illustration**



**UI Technology Stacks**

****

**Service/Batch Process Technology Stacks**

****