Package.json

"dev": "lite-server",

"build-prod": "node --max\_old\_space\_size=5048 ./node\_modules/@angular/cli/bin/ng build --env=prod --aot"

"@swimlane/ngx-charts": "^7.0.1",

"ngx-permissions": "^3.1.0",

"ngx-webstorage": "^2.0.1",

Angular Cli.json

"assets": [

"assets",

{ "glob": "app.config.json", "input": "./app/config/", "output": "./config/" },

"favicon.ico",

"web.config"

],

"styles": [

"themes/styles.scss",

"themes/themes.scss",

"../node\_modules/@covalent/core/common/platform.scss"

],

"defaults": {

"styleExt": "scss",

"component": {},

"serve": {

"port": 4200,

"host": "localhost"

}

Angular Web.Config

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="Cache-Control" value="no-cache" />

</customHeaders>

</httpProtocol>

<staticContent>

<clientCache cacheControlMode="DisableCache" />

</staticContent>

<rewrite>

<rules>

<rule name="Angular" stopProcessing="true">

<match url="^api/?" negate="true" />

<conditions logicalGrouping="MatchAll">

<add input="{REQUEST\_FILENAME}" matchType="IsFile" negate="true" />

<add input="{REQUEST\_FILENAME}" matchType="IsDirectory" negate="true" />

</conditions>

<action type="Rewrite" url="/" />

</rule>

</rules>

</rewrite>

</system.webServer>

</configuration>

Loader

<aiim-app>

<div class="brand-title" id="loader-text">AIIM</div>

<div id="loader"></div>

</aiim-app>

Index.html

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="X-UA-Compatible" content="IE=11">

<meta http-equiv="cache-control" content="no-cache, must-revalidate, post-check=0, pre-check=0">

<meta http-equiv="expires" content="0">

<meta http-equiv="pragma" content="no-cache">

<title>Metlife Investments AIIM</title>

<base href="/">

<link rel="icon" type="image/x-icon" href="favicon.ico">

<script>

\_\_Zone\_enable\_cross\_context\_check = true;

</script>

Themes

$foreground: map-get($theme, foreground);

$background: map-get($theme, background);

@import "variables";

@import "custom-styles";

@import "loader";

Environment

export const environment = {

production: true,

configFilePath: 'config/app.config.json'

};

Route Change

private router: Router,

private route: ActivatedRoute) {

this.router.navigate([this.subRoutes[0].route]);

this.appConstantService.appConstants.appRoutesTitles.subRoutes.filter( s => s.key === item.route)[0].routes;

providers: [

{ provide: ErrorHandler, useClass: CustomErrorHandler },

],

static forRoot(): ModuleWithProviders {

return {

ngModule: AppCoreModule,

providers: [

AuthenticationService, AuthGuard,

AppConstantService, UtilitiesService, AppNotificationService, ConfigService,

CurrencyPipe, DecimalPipe, DatePipe, PercentPipe, AppRoutesTitles, {

provide: APP\_INITIALIZER,

useFactory: configFactory,

deps: [HttpClient, Injector, ConfigService],

multi: true

}

]

};

export class ConfigService {

public config: AppConfig;

public constructor() {}

}

export function configFactory(http: HttpClient, injector: Injector, configSrv: ConfigService) {

const configFileName = environment.configFilePath;

return () => new Promise<any>((resolve: any) => {

const locationInitialized = injector.get(LOCATION\_INITIALIZED, Promise.resolve(null));

locationInitialized.then(() => {

http.get(configFileName).subscribe(res => {

configSrv.config = <AppConfig>res;

resolve(null);

}, error => {

console.error('INITIALIZATION:', error);

resolve(null);

});

});

});

}

const routes: Routes = [

{ path: 'packages-config', redirectTo: '/packages-config', pathMatch: 'full' },

{ path: 'feeds-config', redirectTo: '/feeds-config', pathMatch: 'full' },

{ path: 'file-queue-monitor', redirectTo: '/file-queue-monitor', pathMatch: 'full' },

{ path: 'products', redirectTo: '/products', pathMatch: 'full' },

{ path: 'subscriptions', redirectTo: '/subscriptions', pathMatch: 'full' },

{ path: 'dashboard/:productId', redirectTo: '/dashboard/:productId', pathMatch: 'full' },

{ path: 'dashboard', redirectTo: '/dashboard', pathMatch: 'full' },

{ path: '', redirectTo: '/dashboard', pathMatch: 'full'},

{path: '404', component: PageNotFoundComponent},

{path: 'error', component: DefaultErrorPageComponent},

{path: '\*\*', redirectTo: '/404'}

];

constructor(injector: Injector) {

this.appConstantService = injector.get(AppConstantService);

this.utilitiesService = injector.get(UtilitiesService);

export interface AppRoute {

title: string;

route: string;

icon: string;

tooltip: string;

}

this.routes = [{

title: 'Dashboard',

route: 'dashboard',

icon: 'dashboard',

tooltip: 'Dashboard',

mainTitle: 'Dashboards'

}, {

this.subRoutes = [

{

key: 'dashboard',

routes: [{

title: 'Dashboard',

route: 'dashboard',

icon: 'dashboard',

tooltip: 'AIIM Dashboard'

}]

}, {

key: 'file-queue-monitor',

routes: [{

title: 'File Queue Monitor',

route: 'file-queue-monitor',

icon: 'assessment',

tooltip: 'Monitor incoming file'

}]

}, {

Service base error

public extractData(res: Response) {

return res || {};

}

public handleError(error: Response) {

let errMsg = '';

console.log(errMsg);

if (error instanceof Response) {

const body = error.json() || '';

const err = body.error || body;

this.loggedError = new LoggedError();

try {

this.loggedError.message = err.message;

this.loggedError.messageDetail = err.messageDetail;

} catch (error) {

this.loggedError.message = 'Error occurred';

this.loggedError.messageDetail = err;

}

if (error.status !== 0) {

errMsg = `${error.status} - ${error.statusText || ''} ${err}`;

}

}

if (errMsg !== '') {

let toast: Toast;

toast = {

title: this.loggedError.message,

bodyOutputType: BodyOutputType.TrustedHtml,

body: `<b><i>${this.loggedError.messageDetail}</i></b>`,

type: AppNotificationType[AppNotificationType.error],

toastContainerId: 501

};

this.appNotificationService ? this.appNotificationService.generateNotification(toast) : console.log(error);

return;

}

return Observable.throw(errMsg);

}

export class KeyValuePair<K, V>

{

public key : K;

public value : V;

}

public getLoggedInUserProfile(): Observable<User> {

return this.httpUtilities.httpGet<User>(this.webApiUrls.loggedInUserProfile);

}

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> {

return Observable.of(true);

// return this.userProfileService.isAuthenticatedUser()

// .map(

// (result) => {

// if (result === true) {

// this.isActiveUser = true;

// } else {

// this.isActiveUser = false;

// }

// if (!this.isActiveUser) {

// this.router.navigate(['/access-denied']);

// }

// return result;

// }

// )

// .catch((err, res) => {

// console.log(err);

// this.router.navigate(['/error', { code: 'API404' }]);

// return Observable.throw(err);

// });

}

canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> {

return this.canActivate(route, state);

}

}

import {Observable} from 'rxjs/Observable';

import {merge} from 'rxjs/observable/merge';

import {of as observableOf} from 'rxjs/observable/of';

import {catchError} from 'rxjs/operators/catchError';

import {map} from 'rxjs/operators/map';

import {startWith} from 'rxjs/operators/startWith';

import {switchMap} from 'rxjs/operators/switchMap';

export { AppNotificationService } from './notification-handler/app-notification.service';

export \* from './constants/index';

export \* from './utilities/index';

Routes

this.routes = [{

title: 'Dashboard',

route: '',

icon: 'dashboard',

tooltip: 'Dashboard',

mainTitle: 'Dashboard',

hasSubRoutes: false

}, {

title: 'All Events',

route: '',

icon: 'assessment',

tooltip: 'Monitor incoming file',

mainTitle: 'File Queue Monitor',

hasSubRoutes: true,

subRoutes: [{

title: 'Type',

route: '/events/event-type',

icon: 'assessment',

tooltip: 'Monitor incoming file',

hasSubRoutes: false,

}, {

title: 'Assigned To',

route: '/events/assigned-to',

icon: 'assessment',

tooltip: 'Monitor incoming file',

hasSubRoutes: false

}]

}, {

const routes: Routes = [

{path: 'admin', loadChildren: 'app/admin/admin.module#AdminModule'},

{path: 'events/:event-type', redirectTo: '/events', pathMatch: 'full'},

{path: 'events', redirectTo: '/events', pathMatch: 'full'},

{path: 'tasks', redirectTo: '/tasks', pathMatch: 'full'},

{path: 'dashboard', redirectTo: '/dashboard', pathMatch: 'full'},

{path: 'faq', component: FaqComponent},

{ path: '', redirectTo: '/dashboard', pathMatch: 'full'}

// {path: '404', component: PageNotFoundComponent},

// {path: 'error', component: DefaultErrorPageComponent},

// {path: '\*\*', redirectTo: '/404'}

];

import \* as $ from 'jquery';

import 'jquery';

import 'metismenu';

{ path: 'access-denied', component: AccessDeniedComponent },

{ path: 'property', loadChildren: 'app/views/property/property.module#PropertyModule',

canActivate: [AuthGuard], canActivateChild: [AuthGuard], },

{ path: 'valuation', loadChildren: 'app/views/valuation/valuation.module#ValuationModule',

canActivate: [AuthGuard], canActivateChild: [AuthGuard], },

export class AuthGuard implements CanActivate, CanActivateChild {

imports: [RouterModule.forRoot(routes,{preloadingStrategy: PreloadAllModules})],

public alphaRegex = `^[a-zA-Z ]+$`;

public alphaNumeric = `^[0-9a-zA-Z]+$`;

public currencyRegEx = `^(?=.{0,18}$)\\$?([+]?[0-9]{1,9}(?:,?[0-9]{0,9})\*(?:\\.{0,1}[0-9]{0,2})?)$`;

public currencyRegExWithMinus = `^(?=.{0,18}$)([+-]?|\\()\\$?([0-9]{1,9}(?:,?[0-9]{0,9})\*(?:\\.{0,1}[0-9]{0,2})?)\\)?$`;

public decimalRegEx = `^(?=.{0,20}$)\\$?([+-]?[0-9]{1,9}(?:,?[0-9]{0,9})\*(?:\\.{0,1}[0-9]{0,4})?)$`;

public decimalRegExPositiveOnly = `^(?=.{0,20}$)\\$?([+]?[0-9]{1,9}(?:,?[0-9]{0,9})\*(?:\\.{0,1}[0-9]{0,4})?)$`;

public decimalRegExUpto1000 = `^(?=.{0,6}$)([+-]?[0-9]{0,3}?(?:\\.?{0,1}[0-9]{0,2})$`;

public decimalRegExWithoutDecimal = `^(?=.{0,18}$)\\$?([+-]?[0-9]{1,9}(?:,?[0-9]{0,9})\*)$`;

public decimalRegExPositiveWithoutDecimal = `^(?=.{0,18}$)\\$?([+]?[0-9]{1,9}(?:,?[0-9]{0,9})\*)$`;

// Gopi latest backup

// public currencyRegEx = `^(?=.{0,18}$)[+]?\\$?([0-9]{1,9}(?:,?[0-9]{0,9})\*(?:\\.{0,1}[0-9]{0,2})?)$`;

// public currencyRegExWithMinus = `^(?=.{0,18}$)([+-]?|\\()\\$?([0-9]{1,9}(?:,?[0-9]{0,9})\*(?:\\.{0,1}[0-9]{0,2})?)\\)?$`;

// public decimalRegEx = `^(?=.{0,20}$)([+-]?|\\()\\$?([0-9]{1,9}(?:,?[0-9]{0,9})\*(?:\\.{0,1}[0-9]{0,4})?)?\\)?$`;

// public decimalRegExUpto1000 = `^(?=.{0,6}$)[+-]?[0-9]{0,3}?(?:\.{0,1}[0-9]{0,2})$`;

// public decimalRegExWithoutDecimal = `^(?=.{0,18}$)([+-]?|\\()\\$?([0-9]{1,9}(?:,?[0-9]{0,9})\*)\\)?$`;

// public decimalRegExPositiveWithoutDecimal = `^(?=.{0,18}$)[+]?\\$?([0-9]{1,9}(?:,?[0-9]{0,9})\*)$`;

public emailRegEx = `^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+.[a-zA-Z]$`;

public lessThanTen = `^[+-]?([0-9]{0,1}(?:\\.[0-9]{1,3})?)$|^[+-]?(10(?:\\.[0]{1,3})?)$`;

//public lessThanTen = `^[+-]?[0-9]{0,1}(?:\\.[0-9]{1,3})?|^10(?:\\.[0]{1,3})?$`;

public numberMaxFourDigits = `^[0-9]{0,4}$`;

public numberMaxSevenDigits = `^[0-9]{0,7}$`;

public onlyYear = `^[0-9]{4,4}$`;

//public year19002100 = `^[0-9]{4,4}$`;

public year19002100 = `^19[0-9][0-9]$|^20[0-9][0-9]$|^2100$`;

public yearValue="^19\d\d|2[0-1]\d\d$";

public percentageRegEx = `^[+-]?[0-9]{0,3}(\\.[0-9]{1,2})? \*%?$`;

public percentageRegExUpto100 = '^[0-9]{0,2}(\\.[0-9]{1,2})? \*%?$|^(100|100.00)? \*%?$'; // FINAL

public postiveNumberRegex = '^[0-9]\*$';

public percentageRegWithouDecExUpto100 = '^[0-9]{0,2}? \*%?$|^100? \*%?$'; //Nithya

public decimalValueRegExUpto1000 = '^[0-9]{0,4}(\\.[0-9]{1,4})? \*?$|^1000|1000.00 \*?$'; // Gautam

public usersUpdate = 'users?applicationName=EMS&updateUserRoles={updateUserRoles}';

public userAvailableRoles = 'roles/all?metnetId={metnetId}';

public userAssignedRoles = 'roles/metnet/selected?metNetId={metnetId}';

public userLeadAMRoles = 'users/metnet/roles/leadAssestManager?metNetId={metnetId}';

public userAccessByRole = 'users/metnet/isAllowAccess?metNetId={metnetId}&roleSK={roleSK}';

public userAccessByWorkflowRole = 'users/workflowrole/isAllowAccess?metNetId={metnetId}&workflowRoleSK={workflowRoleSK}';

public appUIRoutes: IDictionary<string, PageRouteAndTitle>[];

this.appUIRoutes = [];

this.appUIRoutes[this.pageRoutes\_Dashboard] = new PageRouteAndTitle(

{

pageRouterLink: this.pageRoutes\_Dashboard,

pageMainTitle: 'Dashboard',

pageTitle: '',

pageSubTitle: '',

breadCrumbCaption: 'Dashboard',

pageRouterLinkCaption: 'Dashboard',

pageRouterBreadcrumbUrl: '/' + this.pageRoutes\_Dashboard

})

export interface IDictionary<K,V>

{

0: K,

1: V

};

providers: [

{ provide: APP\_BASE\_HREF, useValue: '/' },

Append to body directive

import { Directive, ElementRef, Input } from '@angular/core';

@Directive({ selector: '[ems-append-to-body]' })

export class AppendToBodyDirective{

constructor(el: ElementRef) {

document.body.appendChild(el.nativeElement);

}

}

import { Directive, OnInit, OnDestroy, Output, EventEmitter, ElementRef, OnChanges, SimpleChanges, Input } from '@angular/core';

@Directive({ selector: '[ems-confirm-dialog-appendTo]' })

export class ConfirmDialogAppendToDirective implements OnChanges{

@Input() appendToComponent: any;

@Input() appendToBody: boolean;

constructor(public el: ElementRef) {

}

public ngOnChanges(changes: SimpleChanges) {

if (this.appendToBody) {

document.body.appendChild(this.el.nativeElement);

return;

}

if(this.appendToComponent)

this.appendToComponent.appendChild(this.el.nativeElement);

}

}

import { Directive, forwardRef, Attribute } from '@angular/core';

import { Validator, AbstractControl, NG\_VALIDATORS } from '@angular/forms';

@Directive({

selector: '[validateDate][formControlName],[validateDate][formControl],[validateDate][ngModel]',

providers: [

{ provide: NG\_VALIDATORS, useExisting: forwardRef(() => InvalidDateValidator), multi: true }

]

})

export class InvalidDateValidator implements Validator {

constructor( @Attribute('validateDate') public validateDate: string) {}

validate(c: AbstractControl): { [key: string]: any } {

if (isNaN(new Date(this.validateDate).getDate())) return {

validateDate: false

}

return null;

}

}

Validation error

<template #popTemplate>

<div \*ngIf="showValidation"

class="alert alert-danger">

<div [hidden]="!showRequiredValidation">

{{requiredErrorMsg}}

</div>

<div [hidden]="!showMinLengthValidation">

{{minLengthErrorMsg}}

</div>

<div [hidden]="!showMaxLengthValidation">

{{maxLengthErrorMsg}}

</div>

<div [hidden]="!showPatterValidation">

{{patternErrorMsg}}

</div>

</div>

</template>

<div

[popover]="popTemplate"

triggers="" #pop="bs-popover"

placement="bottom"

triggers="focus">

</div>

import { Component, OnInit, Input, OnChanges, ViewChild, SimpleChanges } from '@angular/core';

@Component({

selector: 'ems-validation-error-display',

templateUrl: 'validation-error-display.component.html'

})

export class ValidationErrorDisplayComponent implements OnInit,OnChanges {

@Input() showValidation?: boolean;

@Input() errors?: any;

@Input() requiredErrorMsg: any;

@Input() minLengthErrorMsg: any;

@Input() maxLengthErrorMsg: any;

@Input() patternErrorMsg: any;

@ViewChild('pop') pop: any;

public showRequiredValidation: boolean;

public showMinLengthValidation: boolean;

public showMaxLengthValidation: boolean;

public showPatterValidation: boolean;

constructor() { }

ngOnInit() {

this.showValidation =false;

this.showRequiredValidation =false;

this.showMinLengthValidation =false;

this.showMaxLengthValidation =false;

this.showPatterValidation =false;

}

ngOnChanges(changes:SimpleChanges) {

this.showRequiredValidation =false;

this.showMinLengthValidation =false;

this.showMaxLengthValidation =false;

this.showPatterValidation =false;

if (this.showValidation)

{

this.pop.show();

} else {

this.pop.hide();

}

if (this.errors) {

if (this.errors.required) {

this.showRequiredValidation = this.errors.required;

}

if (this.errors.minlength) {

this.showMinLengthValidation = this.errors.minlength;

}

if (this.errors.maxlength) {

this.showMaxLengthValidation = this.errors.maxlength;

}

if (this.errors.pattern) {

this.showPatterValidation = this.errors.pattern;

}

}

}

}

<form #capitalTrackingForm="ngForm" (ngSubmit)="onSubmit()">

<input type="text" class="form-control ems-invalid " style="text-align: right;" name="{{'capitalSpentYTD\_b\_' + rowData.projectNum}}"

disabled="{{!isGridEditable ||

(((rowData.projectDescription === null) ? true : false) && isGridEditable)}}"

[(ngModel)]="rowData.capitalSpentYTD"

[pattern]="appConstants.validationRegExs.decimalRegExPositiveWithoutDecimal"

(blur)="capitalTrackingValueEdit($event.target.value,rowData)"

#capitalSpentYTD="ngModel">

<ems-validation-error-display [showValidation]="capitalSpentYTD.errors && (capitalSpentYTD.dirty || capitalSpentYTD.touched)"

[errors]="capitalSpentYTD.errors" [patternErrorMsg]="validationMessages.validPositiveNumeric"></ems-validation-error-display>

<form #capitalTrackingProjectsForm="ngForm" autocomplete="off">

ReActive

<form (ngSubmit)="onSubmit()" \*ngIf="active" [formGroup]="attributeForm" autocomplete="off" novalidate>

Switch Case

<ng-container [ngSwitch]="col.key">

<ng-container \*ngSwitchCase="col.key === consts.columns\_ItemCode || col.key === consts.columns\_ItemName || col.key === consts.columns\_Status || col.key === consts.columns\_DisplayOrderNumber ? col.key: 'default'">

<!--already handled above , So Ignore-->

</ng-container>

<ng-container \*ngSwitchCase="consts.columns\_CustomListIndText">

Form control name.

<div class="col-md-5">

<mdl-textfield type="text" name="col.key" [class.is-invalid]="attributeForm.get(col.key).invalid"

label={{col.value}} aria-required="true" floating-label disableNativeValidityChecking

[error-msg]="formErrors[col.key]" formControlName={{col.key}}></mdl-textfield>

</div>

import { Component, OnInit, Input, Injector, ViewChild, EventEmitter, Output, OnChanges, Inject,SimpleChanges } from "@angular/core";

import { FormBuilder, FormGroup, Validators } from "@angular/forms";

import { ValidatorFn, AbstractControl } from "@angular/forms";

public attributeForm: FormGroup;

public constructor(public injector: Injector,

public referenceDataAttributesService: AttributesService,

public userService: UsersService,

public fb: FormBuilder) {

super(injector);

}

public generateFormGroup(attributeToBind: ListItemType) {

this.attributeForm = this.fb.group({

itemCode: [{value: attributeToBind.itemCode, disabled: this.itemCodeDisabled}, [Validators.required,

Validators.maxLength(this.attributeTypeMetaData.itemCodeMaxLength),, this.isItemCodeExisting()]],

itemName: [{value: attributeToBind.itemName, disabled: this.itemNameDisabled}, [Validators.required, Validators.maxLength(this.attributeTypeMetaData.itemNameMaxLength)]],

activeInd: [attributeToBind.activeInd],

status: [attributeToBind.status, [Validators.required]],

displayOrderNumber: [{value:attributeToBind.displayOrderNumber, disabled: this.displayOrderNumberDisabled},

[Validators.required, Validators.pattern(this.appConstants.validationRegExs.numberMaxFourDigits)]],

});

this.attributeForm.valueChanges

.subscribe(data => this.onValueChanged(data));

this.onValueChanged(); // (re)set validation messages now

}

public populateDropdownList()

{

this.yesNoValues = this.commonUtilities.getYesNoDropdownValues();

this.statusValues = this.commonUtilities.getStatusDropdownValues();

}

public onValueChanged(data?: any) {

if (!this.attributeForm) { return; }

const form = this.attributeForm;

for (const field in this.formErrors) {

// clear previous error message (if any)

this.formErrors[field] = "";

const control = form.get(field);

if (control && control.dirty && !control.valid) {

const messages = this.attributeFormValidationMessages[field];

for (const key in control.errors) {

this.formErrors[field] += messages[key] + ' ';

}

}

}

}

public onSubmit() {

if (!this.attributeForm) {return;}

let value = this.attributeForm.getRawValue();

this.isBusy=true;

if (!this.attributeForm.valid) {

this.showError(this.validationMessages.refDataAdmin\_ValidationError);

this.isBusy = false;

return;

}

if (this.attributeTypeMetaData.attributeTypeKey === this.consts.attributeType\_NCREIFPropertyType)

{

if (this.selectedFunctionType[0].id === '' || this.selectedFunctionType[0].id === null) {

this.showError(this.validationMessages.refDataAdmin\_PropertyTypeRequired);

this.isBusy = false;

return;

}

}

public generateFormValidationMessages() {

if (!this.attributeTypeMetaData.attributeTypeKey) { return; }

let itemCodeDisplayValue = this.attributeTypeColumns.filter(d => d.key === this.consts.columns\_ItemCode).map(v => v.value);

let itemNameDisplayValue = this.attributeTypeColumns.filter(d => d.key === this.consts.columns\_ItemName).map(v => v.value);

this.attributeFormValidationMessages = {

"itemCode": {

"required": itemCodeDisplayValue + " is required.",

'maxlength': itemCodeDisplayValue + ' cannot be more than ' + this.attributeTypeMetaData.itemCodeMaxLength.toString() + ' characters long.',

'invalidItemCode': 'Item Code already exists'

},

'itemName': {

'required': itemNameDisplayValue + ' is required.',

'maxlength': itemNameDisplayValue + ' cannot be more than ' + this.attributeTypeMetaData.itemNameMaxLength.toString() + ' characters long.'

},

'displayOrderNumber': {

'required': this.appConstants.validationMessages.refDataAdmin\_DisplayOrderNumber\_Required,

'pattern': this.appConstants.validationMessages.refDataAdmin\_DisplayOrderNumber\_Pattern,

},

'endCalendarYear': {

'pattern': this.appConstants.validationMessages.refDataAdmin\_endCalendarYear\_Pattern,

}

};

};

public isItemCodeExisting(): ValidatorFn {

return (control: AbstractControl): {[key: string]: any} => {

if(control.hasError && (control.dirty || control.touched)) {

const itemCode : string = control.value;

const no = this.attributes

.filter(attrs=> attrs.itemCode.toLocaleLowerCase() !== this.attribute.itemCode.toLocaleLowerCase())

.find(attrs => attrs.itemCode.toLocaleLowerCase() === itemCode.toLocaleLowerCase()) ? true : false;

return no ? {'invalidItemCode': itemCode + ' already exists'} : null;

} else {

return null;

}

};

}

public showError(errorMessage: string) {

this.toast = { title: this.attributeTypeMetaData.attributeCategory + " " +

this.appConstants.referenceDataAdmin\_Header + " - " +

this.attributeTypeMetaData.attributeTypeValue , body: errorMessage, type: AppNotificationType[AppNotificationType.error] };

this.appNotificationService.generateNotification(this.toast);

}

public showSuccess(successMessage: string) {

this.toast = { title: this.attributeTypeMetaData.attributeCategory + " " +

this.appConstants.referenceDataAdmin\_Header + " - " +

this.attributeTypeMetaData.attributeTypeValue, body: successMessage, type: AppNotificationType[AppNotificationType.success] };

this.appNotificationService.generateNotification(this.toast);

}

protected void Application\_BeginRequest()

{

if (Request.Headers.AllKeys.Contains("Origin") && Request.HttpMethod == "OPTIONS")

{

Response.Flush();

}

}

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="Access-Control-Allow-Origin" value="http://localhost:4400" />

<add name="Access-Control-Allow-Headers" value="Content-Type,Accept" />

<add name="Access-Control-Allow-Methods" value="GET, POST, PUT, DELETE, OPTIONS" />

<add name="Access-Control-Allow-Credentials" value="true" />

</customHeaders>

</httpProtocol>

public class CustomAuthenticationFilterAttribute : Attribute, IAuthenticationFilter

{

public async Task AuthenticateAsync(HttpAuthenticationContext context, CancellationToken cancellationToken)

{

await Task.Run(() =>

{

IPrincipal incomingPrincipal = context.ActionContext.RequestContext.Principal;

string userName = incomingPrincipal.Identity.Name;

//SecurityLogic securityLogic = new SecurityLogic();

//bool isActiveUser = securityLogic.isActiveUser(userName);

bool isActiveUser = true;

if (!isActiveUser)

{

context.ErrorResult = new AuthenticationFailureResult("Invalid user", null);

}

});

}

public async Task ChallengeAsync(HttpAuthenticationChallengeContext context, CancellationToken cancellationToken)

{

await Task.Run(() =>

{

IPrincipal incomingPrincipal = context.ActionContext.RequestContext.Principal;

//Debug.WriteLine(String.Format("Incoming principal in custom auth filter ChallengeAsync method is authenticated: {0}", incomingPrincipal.Identity.IsAuthenticated));

});

}

public bool AllowMultiple

{

get { return false; }

}

}

public class AuthenticationFailureResult : IHttpActionResult

{

public AuthenticationFailureResult(string reasonPhrase, HttpRequestMessage request)

{

ReasonPhrase = reasonPhrase;

Request = request;

}

public string ReasonPhrase { get; private set; }

public HttpRequestMessage Request { get; private set; }

public Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken)

{

return Task.FromResult(Execute());

}

private HttpResponseMessage Execute()

{

HttpResponseMessage response = new HttpResponseMessage(HttpStatusCode.Unauthorized);

response.RequestMessage = Request;

response.ReasonPhrase = ReasonPhrase;

return response;

}

}

config.EnableCors();

// Web API routes

config.MapHttpAttributeRoutes();

GlobalConfiguration.Configuration.Formatters.Remove(GlobalConfiguration.Configuration.Formatters.XmlFormatter);

var jsonformatter = GlobalConfiguration.Configuration.Formatters.JsonFormatter;

jsonformatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();

jsonformatter.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;

// [Authorize]

[RoutePrefix("Event/Types")]

public class EventTypesController : ApiController

{

private readonly IEventTypesLogic \_EventTypesLogic;

public EventTypesController(IEventTypesLogic EventTypesLogic)

{

\_EventTypesLogic = EventTypesLogic;

}

/// <summary>

/// Get all event types

/// </summary>

/// <returns>All event types</returns>

[Route("")]

[ResponseType(typeof(IEnumerable<EventType>))]

public IHttpActionResult GetEventTypes()

{

return Ok(\_EventTypesLogic.GetEventTypes());

}

var jsonformatter = GlobalConfiguration.Configuration.Formatters.JsonFormatter;

jsonformatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();

// Register Dependecy resolver

UnityConfig.RegisterComponents();

config.Services.Add(typeof(IExceptionLogger), new ApiExceptionLogger()); //Global Logger removed

// config.Services.Replace(typeof(IExceptionHandler), new ApiExceptionHandler());

config.Filters.Add(new ApiExceptionFilter());

GlobalConfiguration.Configuration.Filters.Add(new CustomAuthenticationFilterAttribute());

public class ApiExceptionLogger : ExceptionLogger

{

public override void LogCore(ExceptionLoggerContext context)

{

IExceptionHandlerLogic exceptionHandlerLogic = BuildUnityContainer.Container.Resolve<IExceptionHandlerLogic>();

var errorCode = exceptionHandlerLogic.LogExceptionDetals(context.ExceptionContext.Exception, Thread.CurrentPrincipal.Identity.Name,context.ExceptionContext.Request.RequestUri.ToString());

Error err = new Error

{

Message = ExceptionMessages.SystemErrorOccured,

MessageDetail = ExceptionMessages.CUSTOM\_ERROR\_MESSAGE\_WITHERRORCODE + errorCode

};

throw new HttpResponseException(context.Request.CreateResponse(HttpStatusCode.InternalServerError, err));

}

}

public class ApiExceptionFilter : ExceptionFilterAttribute

{

public override void OnException(HttpActionExecutedContext actionExecutedContext)

{

IExceptionHandlerLogic exceptionHandlerLogic = BuildUnityContainer.Container.Resolve<IExceptionHandlerLogic>();

var errorCode = exceptionHandlerLogic.LogExceptionDetals(actionExecutedContext.Exception, Thread.CurrentPrincipal.Identity.Name, actionExecutedContext.Exception.HelpLink);

Error err = new Error

{

Message = ExceptionMessages.SystemErrorOccured,

MessageDetail = ExceptionMessages.CUSTOM\_ERROR\_MESSAGE\_WITHERRORCODE + errorCode

};

throw new HttpResponseException(actionExecutedContext.Request.CreateResponse(HttpStatusCode.InternalServerError, err));

}

}

public class CustomAuthenticationFilterAttribute : Attribute, IAuthenticationFilter

{

public async Task AuthenticateAsync(HttpAuthenticationContext context, CancellationToken cancellationToken)

{

await Task.Run(() =>

{

IPrincipal incomingPrincipal = context.ActionContext.RequestContext.Principal;

//IPrincipal genericPrincipal = new GenericPrincipal(new GenericIdentity("Andras", "CustomIdentification"), new string[] { "Admin", "PowerUser" });

//context.Principal = genericPrincipal;

SecurityLogic securityLogic = new SecurityLogic();

string userName = incomingPrincipal.Identity.Name;

bool isActiveUser = securityLogic.isActiveUser(userName);

if (!isActiveUser)

{

context.ErrorResult = new AuthenticationFailureResult("Invalid user", null);

//Debug.WriteLine(String.Format("Incoming principal in custom auth filter AuthenticateAsync method is not authenticated: {0}", incomingPrincipal.Identity.IsAuthenticated));

}

});

}

public async Task ChallengeAsync(HttpAuthenticationChallengeContext context, CancellationToken cancellationToken)

{

await Task.Run(() =>

{

IPrincipal incomingPrincipal = context.ActionContext.RequestContext.Principal;

//Debug.WriteLine(String.Format("Incoming principal in custom auth filter ChallengeAsync method is authenticated: {0}", incomingPrincipal.Identity.IsAuthenticated));

});

}

public bool AllowMultiple

{

get { return false; }

}

}

public class AuthenticationFailureResult : IHttpActionResult

{

public AuthenticationFailureResult(string reasonPhrase, HttpRequestMessage request)

{

ReasonPhrase = reasonPhrase;

Request = request;

}

public string ReasonPhrase { get; private set; }

public HttpRequestMessage Request { get; private set; }

public Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken)

{

return Task.FromResult(Execute());

}

private HttpResponseMessage Execute()

{

HttpResponseMessage response = new HttpResponseMessage(HttpStatusCode.Unauthorized);

response.RequestMessage = Request;

response.ReasonPhrase = ReasonPhrase;

return response;

}

}

public class ExceptionHandlerLogic : IExceptionHandlerLogic

{

private IExceptionHandlerData \_exceptionHandlerData;

public ExceptionHandlerLogic(IExceptionHandlerData exceptionHandlerData)

{

\_exceptionHandlerData = exceptionHandlerData;

}

#region "Exception Logging"

public string LogExceptionDetals(ExceptionDetails ExDetails, string UserName)

{

string Error\_ID=string.Empty;

Error\_ID = \_exceptionHandlerData.ExpectionLogging(ExDetails.ErrorDescription, ExDetails.StackTrace, ExDetails.ClassName, ExDetails.MethodName, ExDetails.ApplicationURL,UserName);

return Error\_ID;

}

public string LogExceptionDetals(Exception ex,string userName,string requestUrl)

{

string Error\_ID = string.Empty;

Error\_ID = \_exceptionHandlerData.ExpectionLogging(ex.Message, ex.StackTrace, ex.TargetSite.ReflectedType.Name, ex.TargetSite.Name, requestUrl, userName);

return Error\_ID;

}

#endregion

}

public class ApiExceptionLogger : ExceptionLogger

{

public override void LogCore(ExceptionLoggerContext context)

{

IExceptionHandlerLogic exceptionHandlerLogic = BuildUnityContainer.Container.Resolve<IExceptionHandlerLogic>();

var errorCode = exceptionHandlerLogic.LogExceptionDetals(context.ExceptionContext.Exception, Thread.CurrentPrincipal.Identity.Name,context.ExceptionContext.Request.RequestUri.ToString());

Error err = new Error

{

Message = ExceptionMessages.SystemErrorOccured,

MessageDetail = ExceptionMessages.CUSTOM\_ERROR\_MESSAGE\_WITHERRORCODE + errorCode

};

throw new HttpResponseException(context.Request.CreateResponse(HttpStatusCode.InternalServerError, err));

}

}

public class ApiExceptionFilter : ExceptionFilterAttribute

{

public override void OnException(HttpActionExecutedContext actionExecutedContext)

{

IExceptionHandlerLogic exceptionHandlerLogic = BuildUnityContainer.Container.Resolve<IExceptionHandlerLogic>();

var errorCode = exceptionHandlerLogic.LogExceptionDetals(actionExecutedContext.Exception, Thread.CurrentPrincipal.Identity.Name, actionExecutedContext.Exception.HelpLink);

Error err = new Error

{

Message = ExceptionMessages.SystemErrorOccured,

MessageDetail = ExceptionMessages.CUSTOM\_ERROR\_MESSAGE\_WITHERRORCODE + errorCode

};

throw new HttpResponseException(actionExecutedContext.Request.CreateResponse(HttpStatusCode.InternalServerError, err));

}

}

public class ApiHttpResponseException : IApiHttpResponseException

{

public IHttpActionResult ThrowResponseException(HttpRequestMessage request, Exception ex, HttpStatusCode statusCode, string message, bool shouldLog)

{

if (shouldLog)

{

IExceptionHandlerLogic exceptionHandlerLogic = BuildUnityContainer.Container.Resolve<IExceptionHandlerLogic>();

var errorCode = exceptionHandlerLogic.LogExceptionDetals(ex, Thread.CurrentPrincipal.Identity.Name, ex.HelpLink);

message = message + ", Reference Error Code : " + errorCode;

}

string messageDetail = string.Empty;

if (ex.Data["ErrorDetail"] != null)

{

messageDetail = ex.Data["ErrorDetail"].ToString();

}

Error err = new Error

{

Message = message,

MessageDetail = messageDetail

};

var errorResponse = request.CreateResponse(statusCode, err);

throw new HttpResponseException(errorResponse);

}

public IHttpActionResult ThrowResponseException(HttpRequestMessage request, Exception ex, HttpStatusCode statusCode, string message)

{

return ThrowResponseException(request, ex, statusCode, message, false);

}

}

public class ExceptionHandler : IExceptionHandler

{

public virtual Task HandleAsync(ExceptionHandlerContext context,

CancellationToken cancellationToken)

{

if (!ShouldHandle(context))

{

return Task.FromResult(0);

}

return HandleAsyncCore(context, cancellationToken);

}

public virtual Task HandleAsyncCore(ExceptionHandlerContext context,

CancellationToken cancellationToken)

{

HandleCore(context);

return Task.FromResult(0);

}

public virtual void HandleCore(ExceptionHandlerContext context)

{

}

public virtual bool ShouldHandle(ExceptionHandlerContext context)

{

// return context.ExceptionContext.IsOutermostCatchBlock;

return true;

}

}

public class ExceptionLogger : IExceptionLogger

{

public object LoggedByKey { get; private set; }

public virtual Task LogAsync(ExceptionLoggerContext context,

CancellationToken cancellationToken)

{

if (!ShouldLog(context))

{

return Task.FromResult(0);

}

return LogAsyncCore(context, cancellationToken);

}

public virtual Task LogAsyncCore(ExceptionLoggerContext context,

CancellationToken cancellationToken)

{

LogCore(context);

return Task.FromResult(0);

}

public virtual void LogCore(ExceptionLoggerContext context)

{

}

public virtual bool ShouldLog(ExceptionLoggerContext context)

{

LoggedByKey = "MS\_LoggedBy";

IDictionary exceptionData = context.ExceptionContext.Exception.Data;

if (!exceptionData.Contains("MS\_LoggedBy"))

{

exceptionData.Add("MS\_LoggedBy", new List<object>());

}

ICollection<object> loggedBy = ((ICollection<object>)exceptionData[LoggedByKey]);

if (!loggedBy.Contains(this))

{

loggedBy.Add(this);

return true;

}

else

{

return false;

}

}

}

public interface IApiHttpResponseException

{

IHttpActionResult ThrowResponseException(HttpRequestMessage request, Exception ex, HttpStatusCode statusCode, string message);

IHttpActionResult ThrowResponseException(HttpRequestMessage request, Exception ex, HttpStatusCode statusCode, string message, bool shouldLog);

}