**Angular 6 Web Theme**

**How to start**

In order to start the project use :

* npm install
* npm start

**Folder Structure**

* All service, model, interceptor, global function and custom validator are existing inside of core module.
* All modules that can access after login are existing inside of layout module.
* app module is the root module of all modules.

Eg.

app/core/services/service-name.ts

app/core/models/model-name.ts

app/layout/module-name

**Code scaffolding**

Run `ng g module module-name –routing` to generate module.

Run `ng g component component-name –module module-name`

to generate component.

Run `ng g service service-name` to generate service.

Eg.

#module

ng g module layout/example –routing

#component

ng g component layout/example –module example

#service

ng g service core/services/sample

**Routing**

If you add new menu, you need to add routing to layout-routing-module.ts.

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

#component routing

If you add new component , you need to add routing to specific routing module.

Eg. const routes: Routes = [

{path: '', component: AdminComponent}

];

**Global function**

**messageDialogBox function**

1. import Dialog Module in your module.

@NgModule({

imports: [DialogModule],

declarations: [LoginComponent]

})

2.Declare global function in your component.ts.

public globalFunction: Globalfunction;

constructor(private dialogService: DialogService) {

this.globalFunction = new Globalfunction(dialogService);

}

3.Use message dialog box

this.globalfunction.messageDialogBox(‘message’,’title’);

**Validation**

**Form Validation**

**-**use single validation

Eg. editForm: FormGroup = new FormGroup({

‘name’ : new FormControl(‘’,Validators.required);

});

-use multiple validation

We can use `Validators.compose` to use more than one validation.

Eg. editForm: FormGroup = new FormGroup({

‘name’ : new FormControl(‘’,{validators: Validators.compose([

Validators.required,

Validators.pattern([a-zA-Z]+)]

)]

)}

});

-use custom validation

We can also use custom validation in form control.

Create a function that include custom validation logic and add this function to form control.

Eg. editForm: FormGroup = new FormGroup({

‘name’ : new FormControl(‘’,custom-validation-function-name) });

**HTTP Client**

**Api service call**

* All common api service calls are written in api.service.ts.
* If you want to create http request , you need to inherence api.service.

Eg. Component => your service => api.service

**Http Interceptor**

* Add provider to core module.

Eg. providers: [

{ provide: HTTP\_INTERCEPTORS, useClass: HttpTokenInterceptor, multi: true } ]

* Put http.token.interceptorl.ts at .core/services/interceptors

Notes:

All both service request and response will be passed http.token.interceptor.

**User Permission**

**Ngx Permission**

* Put permission-guard.service.ts file at ./core/services.
* Add “PermissionGuardService” to your routing module by using with canActive property.
* const routes: Routes = [
* {path: '', component: AdminComponent, canActivate: [PermissionGuardService] },
* ];
* add NgxPermissionModule and PermissionGuardService to @NgModule of your routing module.
* @NgModule({
* imports: [RouterModule.forChild(routes), NgxPermissionsModule.forRoot()],
* exports: [RouterModule, NgxPermissionsModule],
* providers: [ PermissionGuardService ]
* })
* Add \*ngxPermissionOnly , allow us to check permission, to your html page.

<a class="btn" \*ngxPermissionsOnly="'edit'" (click)="edit(dataItem)" title="edit"><i class="fa fa-refresh"></i></a>

**Controls**

**File upload**

* Add kendo-upload tab to html.
* <kendo-upload required
* [batch]="true"
* [multiple]="true"
* [restrictions]="uploadRestrictions"
* [autoUpload]="false"
* (select)="selectEventHandler($event)"
* (clear)="clearEventHandler($event)"
* (remove)="removeEventHandler($event)"
* (complete)="completeEventHandler($event)"
* (upload)="uploadEventHandler($event)">
* <ng-template kendoUploadFileTemplate let-files let-state="state">
* <div>Name: {{files[0].name}} Size: {{files[0].size}} bytes</div>
* <button \*ngIf="state"
* (click)="remove(myUpload, files[0].uid)"
* class="k-button" style="position: absolute; right: .2em;">
* Remove
* </button>
* </ng-template>
* </kendo-upload>
* Add event handler functions for kendo upload to component.ts.

Eg.

public removeEventHandler(e: RemoveEvent): void {

const index = this.imagePreviews.findIndex(item => item.uid === e.files[0].uid);

if (index >= 0) {

this.imagePreviews.splice(index, 1);

}

}

public uploadRestrictions: FileRestrictions = {

allowedExtensions: ['.jpg', '.png']

};

* Create a folder path in your local device that are written in dotnet core api’s appsetting.json.

Eg. "uploadAdminPath": "D:\\MFI Need File\\Photo\\Admin\\"

**Some config for kendo upload**

[multiple] = “true” => allow to select multiple files.

[restrictions]=”uploadrestrictions” => to restrict upload files.

[autoUpload]=”false” => to prevent auto file upload.

(select) = “selectedfunction($event)”

* “selectedfunction” will be trigger when selected upload file.

(upload) = “uploadfunction($event)”

* “uploadfunction “ will be trigger when selected files are uploaded.

**Tree View**

**ngx-treeview**

* Put treeviewservice.service.ts at ./core/services.
* Add <ngx-treeview> tab to html.

<ngx-treeview [config]="config" [items]="items" [itemTemplate]="itemTemplate" (selectedChange)="onSelectedChange($event)" #treeview>

</ngx-treeview>

* Add “treeview-service-call” to your component.

this.items = [new TreeviewItem(this.treeviewService.createSingleDataTreeView(x.data, 0, []))];

* we need to add “selectedChange event” to get selected node in treeview.
* @Component({
* selector: 'app-adminlevel',
* templateUrl: './adminlevel.component.html',
* styleUrls: ['./adminlevel.component.scss'],
* providers: [
* { provide: TreeviewEventParser, useClass: OrderDownlineTreeviewEventParser },
* ]
* })

onSelectedChange(downlineItems: DownlineTreeviewItem[]) {

this.idList = [0];

downlineItems.forEach(downlineItem => {

const item = downlineItem.item;

const value = item.value;

const texts = [item.text];

let parent = downlineItem.parent;

while (!isNil(parent)) {

const id = parent.item.value;

if (this.idList.findIndex(x => x == id) === -1) {

this.idList.push(id);

}

parent = parent.parent;

}

if (this.idList.findIndex(x => x == value) === -1) {

this.idList.push(value);

}

});

}

* ngx treeview mainly consist of 2 attributes.

1.items : to add data set for tree view

2.config : To configure tree view

config = TreeviewConfig.create({

hasAllCheckBox: false,

hasFilter: false,

hasCollapseExpand: false,

decoupleChildFromParent: false,

maxHeight: 190,

});

Kendo Grid

* Add <kendo-grid> tab in html.
* Add Page size, pageable, skip attributes to enable pagination for grid.
* Add filter and filterable attribute to enable filter.
* Add sort and sortable attribute to enable sorting.
* Add dataStateChange attribute to trigger statechange event.

Eg.

<kendo-grid style="height: 300px;"

            [data]="view | async"

            [height]="533"

            [pageSize]="gridState.take"

            [pageable]="{

                buttonCount: 5,

                info: true,

                type: 'input',

                pageSizes: true,

                previousNext: true

             }"

            [skip]="gridState.skip"

            [sort]="gridState.sort"

            [filter]="gridState.filter"

            [filterable]="true"

            [sortable]="true"

            (dataStateChange)="onStateChange($event)">

Reference:

Folder structure:

<https://github.com/gothinkster/angular-realworld-example-app/tree/master/src/app/core>

File upload:

<https://www.telerik.com/kendo-angular-ui/components/upload/>

Grid:

<https://www.telerik.com/kendo-angular-ui/components/grid/>

Ngx-Treeview:

<https://www.npmjs.com/package/ngx-treeview>

Ngx-Permission:

<https://www.npmjs.com/package/ngx-permissions>

Form validation:

<https://angular.io/api/forms/Validators>