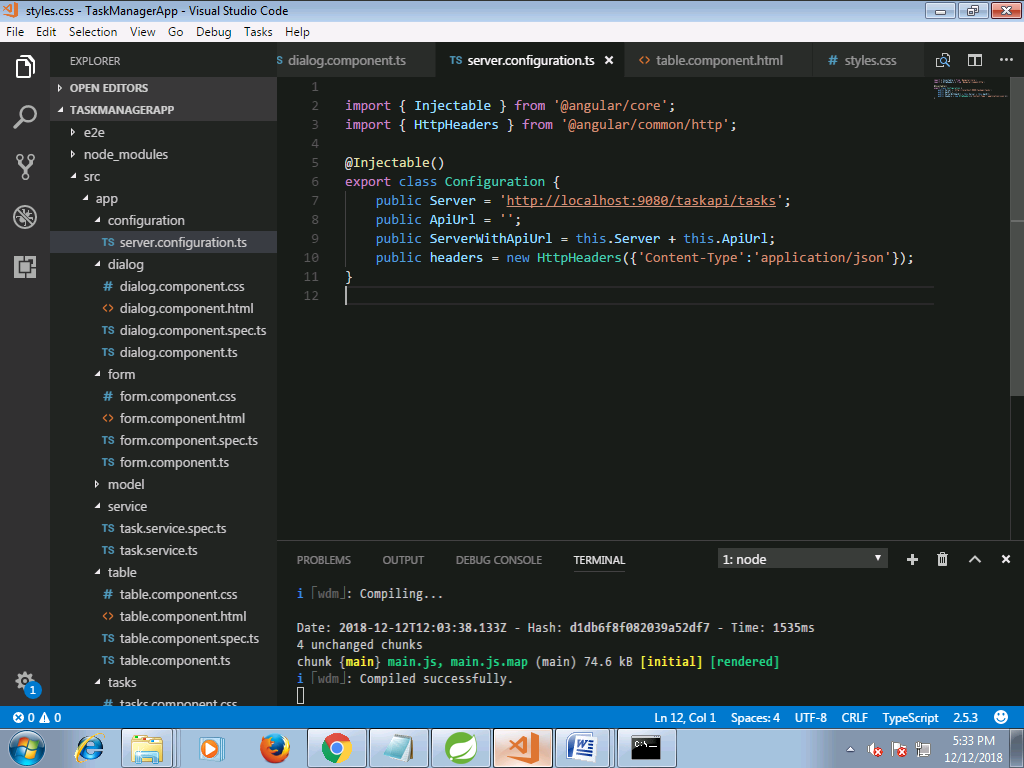
Project Structure



server.configuration.ts

import { Injectable } from '@angular/core';

import { HttpHeaders } from '@angular/common/http';

@Injectable()

export class Configuration {

public Server = 'http://localhost:9080/taskapi/tasks';

public ApiUrl = '';

public ServerWithApiUrl = this.Server + this.ApiUrl;

public headers = new HttpHeaders({'Content-Type':'application/json'});

}

dialog.component.css

/\*\*\*Dialog Box\*\*/

@media (min-width: 768px){

.col-md-4 {

flex: 0 0 100%;

max-width: 100%;

}

}

dialog.component.html

/\*\*\*Dialog Box\*\*/

@media (min-width: 768px){

.col-md-4 {

flex: 0 0 100%;

max-width: 100%;

}

}

form.component.css

.calIcon {

width: 40px;

padding-top: 25px;

cursor: pointer;

}

form.component.html

<div class="p-3 bg-light" [ngClass]="isModalUpdateForm ?'':'mat-elevation-z8 w-50'">

<form [formGroup]="angularForm" >

<div class="row">

<div class="col-xs-12 col-12 col-sm-10 form-group">

<label for="taskValue">Task:

<input class="form-control form-control-sm" formControlName="taskValue">

</label>

</div>

</div>

<div \*ngIf="angularForm.controls['taskValue'].invalid && (angularForm.controls['taskValue'].dirty || angularForm.controls['taskValue'].touched)" class="alert alert-danger col-xs-12 col-12 col-sm-6">

<div \*ngIf="angularForm.controls['taskValue'].errors.required">

Task is required.

</div>

</div>

<div class="row">

<div class="col-xs-12 col-12 col-sm-10 form-group">

<label for="priorityValue">Priority: <span class="display-box">{{angularForm.controls['priorityValue'].value}}</span>

<input type="range" class="form-control form-control-sm" min="0" max="30" formControlName="priorityValue">

</label>

</div>

</div>

<!--<slider-formatting ></slider-formatting>-->

<div class="row">

<div class="col-xs-12 col-12 col-sm-10 form-group">

<label for="parentTaskValue">Parent Task:

<input class="form-control form-control-sm" formControlName="parentTaskValue">

</label>

</div>

</div>

<div class="row">

<div class="col-xs-12 col-12 col-sm-10 form-group">

<label for="startDate">Start Date:

<input type="text" class="form-control" placeholder="YYYY-MM-DD, HH:mm:ss"

#dp1 = "bsDatepicker"

bsDatepicker

[minDate]="startMinDate"

[maxDate]="startMaxDate"

(onHidden)="openEndDatePicker('onShown')"

placement = "top"

[bsConfig]="{ dateInputFormat: 'YYYY-MM-DD, HH:mm:ss' }"

formControlName="startDate"/>

</label>

</div>

<div class="col-xs-12 col-12 col-sm-2 form-group">

<img src="../../assets/images/calendar-icon.svg" class="calIcon" (click)="dp1.toggle()" [attr.aria-expanded]="dp1.isOpen">

</div>

</div>

<div \*ngIf="angularForm.controls['startDate'].invalid && (angularForm.controls['startDate'].dirty || angularForm.controls['startDate'].touched)" class="alert alert-danger col-xs-12 col-12 col-sm-6">

<div \*ngIf="angularForm.controls['startDate'].errors.required">

Start Date is required.

</div>

</div>

<div class="row">

<div class="col-xs-12 col-12 col-sm-10 form-group">

<label for="endDate">End Date:

<input type="text" class="form-control" placeholder="YYYY-MM-DD, HH:mm:ss"

#dp = "bsDatepicker"

bsDatepicker

[minDate]="endMinDate"

[maxDate]="endMaxDate"

placement = "top"

[bsConfig]="{ dateInputFormat: 'YYYY-MM-DD, HH:mm:ss' }"

formControlName="endDate"/>

</label>

</div>

<div class="col-xs-12 col-12 col-sm-2 form-group">

<img src="../../assets/images/calendar-icon.svg" class="calIcon" (click)="dp.toggle()" [attr.aria-expanded]="dp.isOpen">

</div>

</div>

<div \*ngIf="angularForm.controls['endDate'].invalid && (angularForm.controls['endDate'].dirty || angularForm.controls['endDate'].touched)" class="alert alert-danger col-xs-12 col-12 col-sm-6">

<div \*ngIf="angularForm.controls['endDate'].errors.required">

End Date is required.

</div>

</div>

<div class="form-group" \*ngIf="!isModalUpdateForm">

<div class="padRight10 float-left">

<button type="submit" (click) ="onSubmit($event);"

[disabled]="angularForm.pristine || angularForm.invalid || isSubmitted" class="btn btn-primary">

{{submitButtonLabel}}

</button>

</div>

<div>

<button type="submit" (click)="resetClick();" class="btn btn-secondary">

Reset

</button>

</div>

</div>

<div class="modal-footer" \*ngIf="isModalUpdateForm">

<button type="button" class="btn btn-primary" (click) ="onUpdate($event);" [disabled]="valueChnages">UPDATE</button>

<button type="button" class="btn btn-outline-dark" (click)="cancelClick($event)">CANCEL</button>

</div>

</form>

</div>

form.component.ts

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

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

import { Task} from '../model/task';

@Component({

selector:'add-task-form',

styleUrls:['./form.component.css'],

templateUrl : './form.component.html'

})

export class FormComponent implements OnInit ,OnChanges{

// startDatepickerModel :Date; //Selected date in start date

startMinDate :Date; //Set Start date min as current date

endMinDate :Date; // Set start date end as after date selected in start date

@Input('taskToUpdate') task :Task;

\_tempTask :Task;

valueChnages :boolean =true;

@Input() submitButtonLabel :string;

//To check if form submitted

@Input('isSubmitted') isSubmitted :boolean = false;

//For Modal Form for Update

@Input('isModalUpdateForm') isModalUpdateForm : boolean;

//Ends

angularForm = new FormGroup ({

taskValue: new FormControl(),

priorityValue : new FormControl(),

parentTaskValue: new FormControl(),

startDate: new FormControl(),

endDate: new FormControl(),

taskId: new FormControl(),

parenttaskId: new FormControl()

});

constructor(private fb: FormBuilder){

this.startMinDate = new Date();

}

ngOnInit(){

this.createForm();

this.submitButtonLabel = "Add Task";

console.log('Input Changes------ >');

if(this.isModalUpdateForm){

this.onChanges();

}

}

onChanges(): void {

this.angularForm.valueChanges.subscribe(val => {

this.valueChnages = false;

});

}

ngOnChanges(){

console.log("Is submitted :"+this.isSubmitted);

}

//This handler will be called when end date date picker will be shown

openEndDatePicker(value:string) :void{

let endateMin :Date = this.angularForm.controls["startDate"].value;

endateMin.setDate(endateMin.getDate() + 7) ;

this.endMinDate = endateMin;

}

private createForm() {

if(this.task != null){

this.\_tempTask = this.task;

this.angularForm = this.fb.group({

taskValue: [this.\_tempTask.taskName,Validators.required ],

parentTaskValue: this.\_tempTask.parentTask != null ? this.task.parentTask.parentTaskName:null,

priorityValue:this.\_tempTask.priority,

startDate:this.\_tempTask.startDate,

endDate:this.\_tempTask.endDate,

taskId:this.\_tempTask.taskId,

parentTaskId:this.\_tempTask.parentTask != null ? this.task.parentTask.parentId:null,

});

}else{

this.angularForm = this.fb.group({

taskValue: ['',Validators.required ],

parentTaskValue: '',

priorityValue:0,

startDate:['',Validators.required],

endDate:['',Validators.required]

});

}

}

//Emit the event in Parent Component

@Output() outPutToParent: EventEmitter<FormGroup> = new EventEmitter();

public onSubmit($event){

this.isSubmitted = true;

this.outPutToParent.emit(this.angularForm);

this.angularForm.disable();

}

public onUpdate($event){

console.log('Update Form Clicked'+$event);

this.isSubmitted = true;

this.outPutToParent.emit(this.angularForm);

this.angularForm.disable();

}

@Output() cancelUpdate: EventEmitter<any> = new EventEmitter();

public cancelClick($event){

this.cancelUpdate.emit($event);

}

public resetClick($event){

console.log('Reset Form Clicked'+$event);

this.angularForm.reset();

this.\_tempTask = this.task;

this.isSubmitted = false;

this.angularForm.enable();

}

}

task.ts

import { ParentTask } from "./parenttask";

export class Task{

taskId :number;

taskName :string;

priority :number;

parentTask :ParentTask;

startDate:string;

endDate:string;

isActive:boolean;

}

parenttask.ts

export class ParentTask{

parentId :number;

parentTaskName :string;

taskId :number;

isActive:boolean;

}

taskvo.ts

export class TaskVo{

id :number;

taskName :string;

priority :number;

parentTask :string;

startDate:Date;

endDate:Date;

isActive:boolean;

}

taskservice.ts

import { Injectable} from '@angular/core';

import {Logger} from '../utillity/logger.service';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import {Configuration} from '../configuration/server.configuration';

import { DomSanitizer } from '@angular/platform-browser';

import {Task} from '../model/task';

@Injectable()

export class TasksService{

private actionUrl: string;

private headers :HttpHeaders;

constructor(private \_logger: Logger,private http: HttpClient, public sanitizer: DomSanitizer, \_configuration: Configuration) {

this.actionUrl = \_configuration.ServerWithApiUrl;

this.headers = \_configuration.headers;

}

// Get all Tasks

getTasks(): Promise<Task[]> {

console.log(this.actionUrl);

return this.http.get(this.actionUrl)

.toPromise()

.then(response => response as Task[])

.catch(this.handleError);

}

getTask(id: number): Promise<Task> {

return this.http.get(`${this.actionUrl}/${id}`)

.toPromise()

.then(response => response as Task)

.catch(this.handleError);

}

createTask(task: Task): Promise<Task> {

this.\_logger.log('Task Name :'+JSON.stringify(task));

return this.http

.post(this.actionUrl, JSON.stringify(task),{headers:this.headers})

.toPromise()

.then(res => res as Task)

.catch(this.handleError);

}

updateTask(task: Task): Promise<Task> {

const url = `${this.actionUrl}/${task.taskId}`;

return this.http

.put(url, JSON.stringify(task), {headers: this.headers})

.toPromise()

.then(() => task)

.catch(this.handleError);

}

deleteTask(id: number): Promise<void> {

const url = `${this.actionUrl}/${id}`;

return this.http.delete(url, {headers: this.headers})

.toPromise()

.then(() => null)

.catch(this.handleError);

}

private handleError(error: any): Promise<any> {

//this.\_logger.error('Error'); // for demo purposes only

return Promise.reject(error.message || error);

}

}

table.component.html

<div class="mat-elevation-z8 p-3 bg-light">

<form class="">

<div class="row col-xs-12 col-12">

<div class="col-6 form-group no-padding-left">

<label>Task :</label>

<input type="text" class="w-77 ml-1" (keyup)="applyFilterByTaskName($event.target.value)">

</div>

<div class="col-6 form-group no-padding-left">

<label>Parent Task :</label>

<input type="text" class="w-77 ml-1" (keyup)="applyFilterByParentTaskName($event.target.value)">

</div>

</div>

<div class="row col-xs-12 col-12 marTop20">

<div class="col-3 form-group no-padding">

<label>Priority From :

<input type="number" class="w-29" min="0" max="30" value="0" [(ngModel)]="priorityFrom" name="priorityFrom" (change) = "applyFilterByPriorityRange($event.target.value)">

</label>

</div>

<div class="col-3 form-group no-padding">

<label>Priority To :

<input type="number" class="w-29" min="0" max="30" value="30" [(ngModel)]="priorityTo" name="priorityTo" (change) = "applyFilterByPriorityRange($event.target.value)">

</label>

</div>

<div class="col-3 form-group no-padding">

<label for="startDate">Start Date: </label>

<input type="text" class="w-48" placeholder="YYYY-MM-DD"

#dp = "bsDatepicker"

bsDatepicker [(ngModel)] = "startDate"

[minDate]="startMinDate"

[maxDate]="startMaxDate"

(onHidden)="openEndDatePicker(startDate)"

[bsConfig]="{ dateInputFormat: 'YYYY-MM-DD' }"

name="startDate"/>

<img src="../../assets/images/calendar-icon.svg" class="calIcon" (click)="dp.toggle()">

</div>

<div class="col-3 form-group no-padding">

<label for="endDate">End Date: </label>

<input type="text" class="w-48" placeholder="YYYY-MM-DD"

#dp1 = "bsDatepicker"

bsDatepicker [(ngModel)] = "endDate"

[minDate]="endMinDate"

[maxDate]="endMaxDate"

(onHidden) ="applyFilterByEndDate(endDate)"

[bsConfig]="{ dateInputFormat: 'YYYY-MM-DD' }"

name="endDate"/>

<img src="../../assets/images/calendar-icon.svg" class="calIcon" (click)="dp1.toggle()">

</div>

</div>

</form>

</div>

<div class="dataContainer">

<table mat-table [dataSource]="dataSource" matSort matSortDisableClear class="mat-elevation-z8">

<!-- Progress Column -->

<ng-container matColumnDef="taskName">

<th mat-header-cell \*matHeaderCellDef mat-sort-header> Task </th>

<td mat-cell \*matCellDef="let row"> {{row.taskName}} </td>

</ng-container>

<!-- Name Column -->

<ng-container matColumnDef="priority">

<th mat-header-cell \*matHeaderCellDef mat-sort-header> Priority </th>

<td mat-cell \*matCellDef="let row"> {{row.priority}} </td>

</ng-container>

<!-- Color Column -->

<ng-container matColumnDef="parentTask">

<th mat-header-cell \*matHeaderCellDef mat-sort-header> Parent Task </th>

<td mat-cell \*matCellDef="let row" > {{row.parentTask}} </td>

</ng-container>

<!-- Start Date Column -->

<ng-container matColumnDef="startDate">

<th mat-header-cell \*matHeaderCellDef mat-sort-header> Start Date </th>

<td mat-cell \*matCellDef="let row" > {{row.startDate | date: 'dd-MM-yyyy'}} </td>

</ng-container>

<!-- End date Column -->

<ng-container matColumnDef="endDate">

<th mat-header-cell \*matHeaderCellDef mat-sort-header> End Date </th>

<td mat-cell \*matCellDef="let row" > {{row.endDate | date: 'dd-MM-yyyy'}} </td>

</ng-container>

<!-- Action Column -->

<ng-container matColumnDef="customColumn">

<th mat-header-cell \*matHeaderCellDef> Actions </th>

<td mat-cell \*matCellDef="let row" >

<button [disabled]="!row.isActive" class="btn btn-success" (click) = "editOrEndTask(row.id,'edit');"> Edit Task</button>

<button [disabled]="!row.isActive" class="btn btn-primary marLeft20" (click)="editOrEndTask(row.id,'end');">End Task</button>

</td>

</ng-container>

<tr mat-header-row \*matHeaderRowDef="displayedColumns"></tr>

<tr mat-row \*matRowDef="let row; columns: displayedColumns;"></tr>

</table>

<mat-paginator [pageSizeOptions]="[5, 10, 25, 50]"></mat-paginator>

</div>

table.component.css

table {

width: 100%;

}

.mat-form-field {

font-size: 14px;

width: 100%;

}

td, th {

width: 25%;

}

th.mat-sort-header-sorted {

color: black;

}

.calIcon {

width: 30px;

cursor: pointer;

}

.w-77 {

width: 77% !important;

}

.w-29 {

width: 29% !important;

}

.w-48 {

width: 48% !important;

}

.marTop20 {

margin-top: 20px !important;

}

table.component.ts

import {Component, OnInit, ViewChild, Input, Output, EventEmitter} from '@angular/core';

import {MatPaginator, MatSort, MatTableDataSource, MatSortable} from '@angular/material';

import { Task } from '../model/task';

import { TaskVo } from '../model/taskvo';

import { Logger } from '../utillity/logger.service';

import { NgxSpinnerService } from 'ngx-spinner';

import { DatePipe } from '@angular/common';

/\*\*

\* @title Data table with sorting, pagination, and filtering.

\*/

@Component({

selector: 'table-task',

templateUrl: './table.component.html',

styleUrls:['./table.component.css'],

})

export class TableTaskComponent implements OnInit {

displayedColumns: string[] = ['taskName', 'priority', 'parentTask','startDate','endDate','customColumn'];

dataSource: MatTableDataSource<TaskVo>;

public tasks : Array<TaskVo>;

startMinDate :Date; //Set Start date min as current date

endMinDate :Date; // Set start date end as after date selected in start date

startDate :Date;

priorityFrom : number = 0;

priorityTo : number =30;

@ViewChild(MatPaginator) paginator: MatPaginator;

@ViewChild(MatSort) sort: MatSort;

//bind this property from parent injected data List on change of this

// ngOnChanges will be called which is invild feature of Angular 2

@Input('data') data : Task[];

constructor(private datePipe: DatePipe, private spinner: NgxSpinnerService,private \_logger : Logger) {

}

ngOnInit() {

this.sort.sort(<MatSortable>{

id: 'startDate',

start: 'asc'

}

);

}

// Called when "data" is changed in Parent Component

ngOnChanges(){

this.tasks = this.getTasksView(this.data);

this.dataSource = new MatTableDataSource(this.tasks);

this.dataSource.paginator = this.paginator;

this.dataSource.sort = this.sort;

//this.spinner.hide();

}

//This handler will be called when end date date picker will be shown

openEndDatePicker(filterValue: string) :void{

let endateMin :Date = this.startDate;

endateMin.setDate(endateMin.getDate()) ;

this.endMinDate = endateMin;

this.applyFilterByStartDate(filterValue);

}

applyFilterByTaskName(filterValue: string) {

this.dataSource.filterPredicate =

(data: TaskVo, filter: string) => data.taskName.indexOf(filter) != -1;

this.dataSource.filter = filterValue.trim();

if (this.dataSource.paginator) {

this.dataSource.paginator.firstPage();

}

}

applyFilterByParentTaskName(filterValue: string) {

this.dataSource.filterPredicate =

(data: TaskVo, filter: string) => data.parentTask.indexOf(filter) != -1;

this.dataSource.filter = filterValue.trim();

if (this.dataSource.paginator) {

this.dataSource.paginator.firstPage();

}

}

applyFilterByStartDate(filterValue: string) {

console.log(filterValue);

this.dataSource.filterPredicate =

(data: TaskVo, filter: string) :boolean =>{

//console.log("Date Date :"+data.startDate);

let inputValue = this.datePipe.transform(filterValue,'yyyy-MM-dd')

let dataValue = this.datePipe.transform(data.startDate,'yyyy-MM-dd')

return inputValue == dataValue;

}

this.dataSource.filter = filterValue;

if (this.dataSource.paginator) {

this.dataSource.paginator.firstPage();

}

}

applyFilterByEndDate(filterValue: string) {

this.dataSource.filterPredicate =

(data: TaskVo, filter: string) :boolean =>{

//console.log("Date Date :"+data.startDate);

let inputValue = this.datePipe.transform(filterValue,'yyyy-MM-dd')

let dataValue = this.datePipe.transform(data.endDate,'yyyy-MM-dd')

return inputValue == dataValue;

}

this.dataSource.filter = filterValue;

if (this.dataSource.paginator) {

this.dataSource.paginator.firstPage();

}

}

applyFilterByPriorityRange(filterValue: string) {

this.dataSource.filterPredicate =

(data: TaskVo, filter: string) :boolean =>{

//console.log("Date Date :"+data.startDate);

let pFrom = this.priorityFrom;

let pTo = this.priorityTo;

let retValue = data.priority >= pFrom && data.priority <= pTo;

console.log("Return Value :"+retValue);

return retValue;

}

this.dataSource.filter = filterValue;

if (this.dataSource.paginator) {

this.dataSource.paginator.firstPage();

}

}

//Emit the event in Parent Component

@Output() updateTask: EventEmitter<any> = new EventEmitter();

public editOrEndTask(id:number, action:string){

this.updateTask.emit({action:action,id:id});

}

public getTasksView(data: Task[]): Array<TaskVo> {

let taskVoList = new Array<TaskVo>();

let taskVo : TaskVo;

if(data != null){

for (let i = 0; i < data.length; i++){

taskVo = new TaskVo();

taskVo.id = data[i].taskId;

taskVo.taskName = data[i].taskName.trim();

taskVo.priority = data[i].priority;

taskVo.parentTask = data[i].parentTask != null ? data[i].parentTask.parentTaskName.trim():'';

taskVo.startDate = data[i].startDate != null ? new Date(data[i].startDate):null;

taskVo.endDate = data[i].endDate != null ? new Date(data[i].endDate):null;

taskVo.isActive = data[i].isActive;

taskVoList.push(taskVo);

}

}

//this.spinner.hide();

return taskVoList;

}

}

tasks.component.css

.taskContainer{

padding:10px 5%;

}

.display-box{

background-color:antiquewhite;

width: 20px;

height: 20px;

}

task.component.html

<div class="taskContainer">

<!-- <h3>{{app\_heading}}</h3> -->

<div class="taskContent">

<tabset>

<tab heading="View Task">

<table-task [data]="data" (updateTask)="updateTask($event);"></table-task>

</tab>

<tab heading="Add Task">

<div>

<add-task-form [isSubmitted] = "isSubmitted" (outPutToParent)="addUpdateTask($event);"

></add-task-form>

</div>

</tab>

</tabset>

<div>

</div>

</div>

<dialog-form-tag (initiateDialogContent) ="initiateDialogContent($event)" (okClickEmitter) = "okClickCall($event)"

[deleteConfirmBtn] ="deleteConfirmBtn"

[popupHeading]="popupHeading" [popupMessage]="popupMessage"

[updateForm] ="updateForm"

[taskToUpdate] = "task"

(updateTaskEvent) = "updateTaskDetails($event);"></dialog-form-tag>

</div>

<ngx-spinner

bdColor="rgba(51,51,51,0.8)"

size="medium"

color="#fff"

loadingText="Loading..."

type="ball-scale-multiple">

</ngx-spinner>

task.component.spec.ts

import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { TasksComponent } from './tasks.component';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { CUSTOM\_ELEMENTS\_SCHEMA, NO\_ERRORS\_SCHEMA } from '@angular/core';

import { NgxSpinnerService } from 'ngx-spinner';

import { Logger } from '../utillity/logger.service';

import { TasksService } from '../service/task.service';

import { HttpClient, HttpHandler } from '@angular/common/http';

import { Configuration } from '../configuration/server.configuration';

import { DatePipe } from '@angular/common';

describe('TasksComponent', () => {

let component: TasksComponent;

let fixture: ComponentFixture<TasksComponent>;

beforeEach(async(() => {

TestBed.configureTestingModule({

declarations: [ TasksComponent ],

schemas:[CUSTOM\_ELEMENTS\_SCHEMA,NO\_ERRORS\_SCHEMA],

imports:[FormsModule,ReactiveFormsModule],

providers:[NgxSpinnerService,Logger,TasksService,HttpClient,HttpHandler,Configuration,DatePipe]

})

.compileComponents();

}));

beforeEach(() => {

fixture = TestBed.createComponent(TasksComponent);

component = fixture.componentInstance;

fixture.detectChanges();

});

it('should create', () => {

expect(component).toBeTruthy();

});

});

tasks.component.ts

import { Component, OnInit, ViewChild, AfterViewInit, Input, Output, ElementRef, TemplateRef, ViewChildren, Renderer2} from "@angular/core";

import {TasksService} from "../service/task.service";

import { FormGroup } from "@angular/forms";

import {Logger} from '../utillity/logger.service';

import {TableTaskComponent} from '../table/table.component';

import { NgbModalRef } from '@ng-bootstrap/ng-bootstrap';

import { Task } from "../model/task";

import { ParentTask } from "../model/parenttask";

import { DatePipe } from "@angular/common";

import { NgxSpinnerService } from "ngx-spinner";

import { TabsetComponent } from "ngx-bootstrap";

import { DialogComponent } from "../dialog/dialog.component";

@Component({

moduleId:"task-module",

selector:'app-tasks',

templateUrl : './tasks.component.html',

styleUrls:['./tasks.component.css']

})

export class TasksComponent implements OnInit {

currentUrl :any;

public app\_heading :string;

isSubmitted : boolean;

deleteConfirmBtn :boolean;

updateForm : boolean;

popupHeading :string;

popupMessage :string;

taskId : number;

\_logger :Logger = new Logger();

data :Task[];

@ViewChild(TableTaskComponent) tableComp;

@ViewChild(TabsetComponent) tabSet;

@ViewChild(DialogComponent) dialogContent :DialogComponent;

task :Task;

angularForm :FormGroup;

private modalRef: NgbModalRef;

constructor(private spinner: NgxSpinnerService,

private tasksService:TasksService,

private datePipe: DatePipe){

this.app\_heading ="Task Manager";

this.deleteConfirmBtn = false;

}

ngOnInit(){

this.spinner.show();

this.getTasks();

setTimeout(() => {

/\*\* spinner ends after 5 seconds \*/

this.spinner.hide();

}, 1000);

}

//Edit or Delete Popup Form and Message

public updateTask(passValues?:any){

this.\_logger.log("Action NAme-->"+passValues.action);

if(passValues.action == 'edit'){

this.\_logger.log('Contents :'+this.dialogContent);

this.popupHeading = "Edit Task details:";

this.updateForm = true;

this.tasksService.getTask(passValues.id).then((res) =>{

this.task = res as Task;

this.\_logger.log("Task Fetched :"+this.task.taskName);

this.dialogContent.openFormModal();

})

}

if(passValues.action == 'end'){

this.\_logger.log('Task Ended :');

this.taskId = passValues.id;

this.deleteConfirmBtn = true;

this.updateForm = false;

this.popupHeading = "End Task Confirmation!";

this.popupMessage = "Do you want to mark this task as completed ?"

this.dialogContent.openFormModal();

}

}

private getTasks(){

this.\_logger.log("Calling get All Tasks");

this.spinner.show();

return this.tasksService.getTasks().then(tasks =>

{

this.data = tasks;

this.tableComp.getTasksView();

this.spinner.hide();

},(error) => {

this.\_logger.log(error);

});

}

public addUpdateTask(form : FormGroup){

this.angularForm = form;

this.spinner.show();

this.\_logger.log('Task Added!'+form);

let parentTask :ParentTask;

if(form.controls["parentTaskValue"].value != null && form.controls["parentTaskValue"].value != ''){

parentTask = {

parentId:null,

parentTaskName:form.controls["parentTaskValue"].value,

taskId: null,

isActive:true

};

}

this.task = {

taskId:null,

taskName:form.controls["taskValue"].value,

priority :form.controls["priorityValue"].value,

parentTask: parentTask != null ? parentTask:null,

startDate:form.controls["startDate"].value != null ?this.datePipe.transform(form.controls["startDate"].value,'yyyy-MM-dd HH:mm:ss'):'',

endDate:form.controls["endDate"].value != null ? this.datePipe.transform(form.controls["endDate"].value,'yyyy-MM-dd HH:mm:ss'):'',

isActive:true

};

this.tasksService.createTask(this.task).then((res) => {

this.spinner.hide(); // end loading

this.popupHeading = "Task Added Successfully!";

this.popupMessage = "Task has been added within task list."

this.dialogContent.openFormModal(); //open successpopup

})

}

public updateTaskDetails(form : FormGroup){

this.angularForm = form;

this.spinner.show();

this.\_logger.log('Task Added Successfully!'+form);

let parentTask :ParentTask;

if(form.controls["parentTaskValue"].value != null && form.controls["parentTaskValue"].value != ''){

parentTask = {

parentId:form.controls["parentTaskId"].value != null?form.controls["parentTaskId"].value:null,

parentTaskName:form.controls["parentTaskValue"].value,

taskId: null,

isActive:true

};

}else{

parentTask = null;

}

this.task = {

taskId:form.controls["taskId"].value,

taskName:form.controls["taskValue"].value,

priority :form.controls["priorityValue"].value,

parentTask: parentTask != null ? parentTask:null,

startDate:form.controls["startDate"].value != null ?this.datePipe.transform(form.controls["startDate"].value,'yyyy-MM-dd HH:mm:ss'):'',

endDate:form.controls["endDate"].value != null ? this.datePipe.transform(form.controls["endDate"].value,'yyyy-MM-dd HH:mm:ss'):'',

isActive:true

};

this.\_logger.log(this.task);

this.tasksService.updateTask(this.task).then((res) => {

this.spinner.hide(); // end loading

this.popupHeading = "Task Updated Successfully!";

this.popupMessage = "Task has been updated with in task list."

//this.dialogContent.openFormModal(); //open successpopup

this.okClickCall({action:'OK'});

})

}

initiateDialogContent(content :any){

this.dialogContent = content;

}

//called on Ok button clik on confirmation popup.

okClickCall(actionValues?:any){

this.\_logger.log("Action name :"+actionValues.action);

if(actionValues.action == 'OK'){

this.angularForm.reset(); // Reset Add Form

this.isSubmitted = false; // Check flag for form submission click

this.angularForm.enable(); // Enable form

// this.modalRef.close();

//window.location.reload();

this.tabSet.tabs[0].active = true;

// this.spinner.show();

this.getTasks();

}else if(actionValues.action == 'Yes'){

//this.modalRef.close();

this.deleteConfirmBtn = false;

this.tasksService.deleteTask(this.taskId).then((res) => {

this.\_logger.log("dleted now");

this.getTasks();

})

}

else if(actionValues.action == 'Cancel'){

//this.modalRef.close();

this.deleteConfirmBtn = false;

this.taskId = null;

}else if(actionValues.action == 'Update'){

//this.modalRef.close();

this.deleteConfirmBtn = false;

this.taskId = null;

}

}

}

logger.service.ts

export class Logger {

log(msg: any) { console.log(msg); }

error(msg: any) { console.error(msg); }

warn(msg: any) { console.warn(msg); }

}

app.component.html

<!--The content below is only a placeholder and can be replaced.-->

<div style="text-align:center">

<h1>

{{ title }}

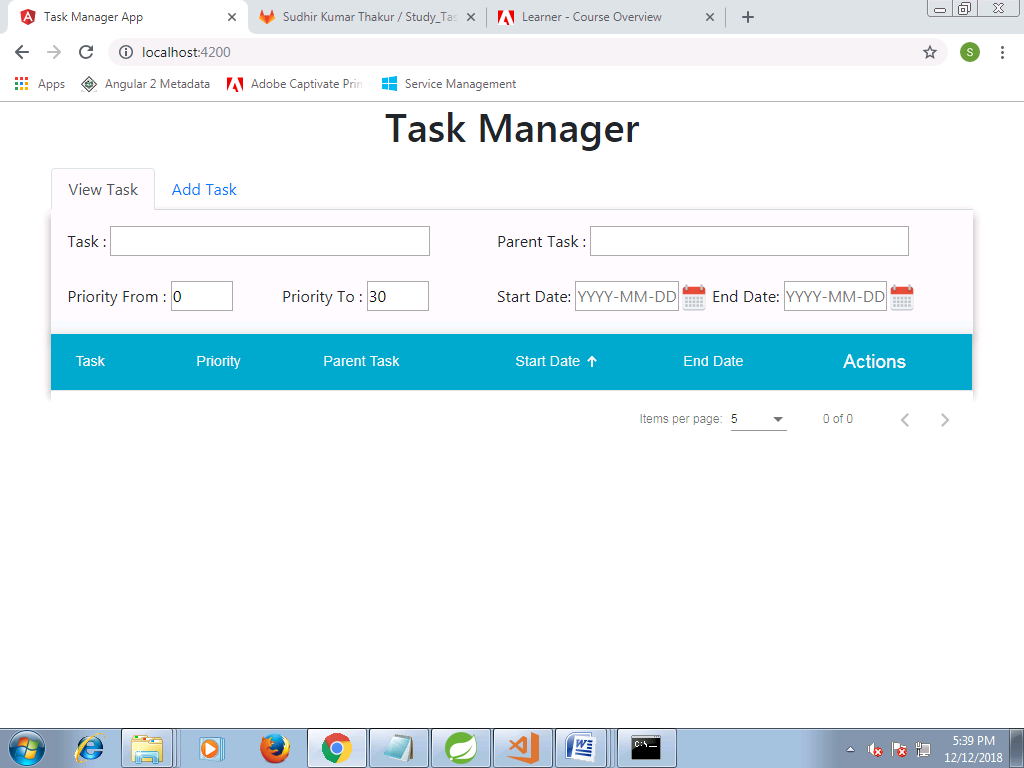
</h1>

</div>

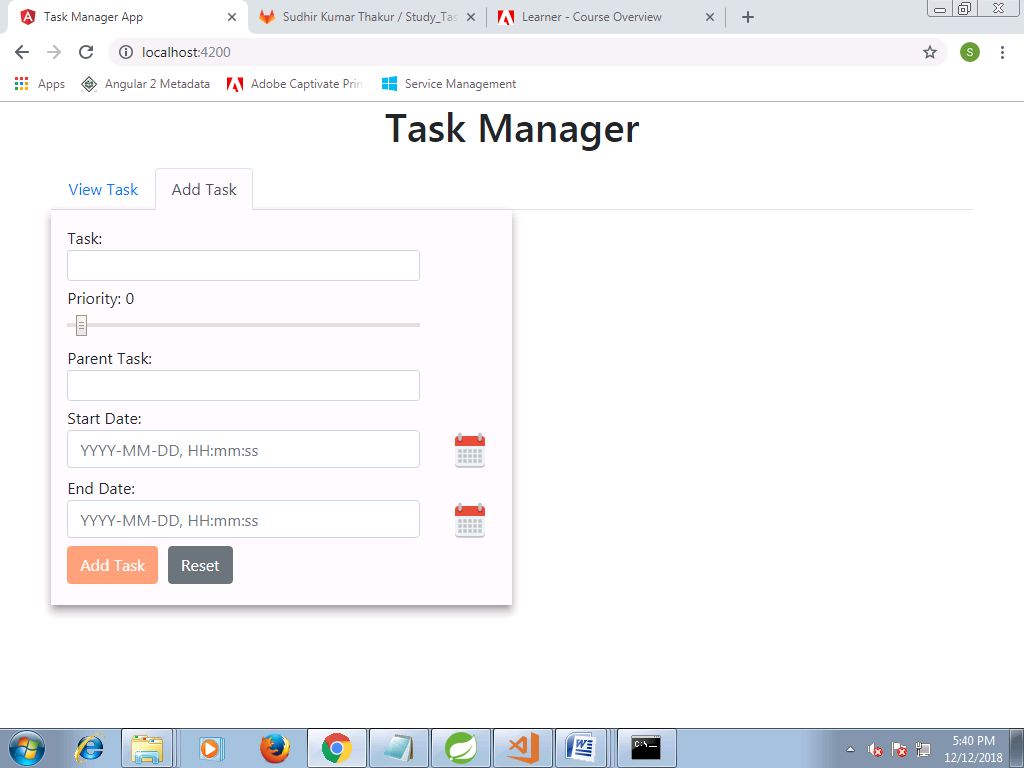
<app-tasks></app-tasks>

**Screenshots**

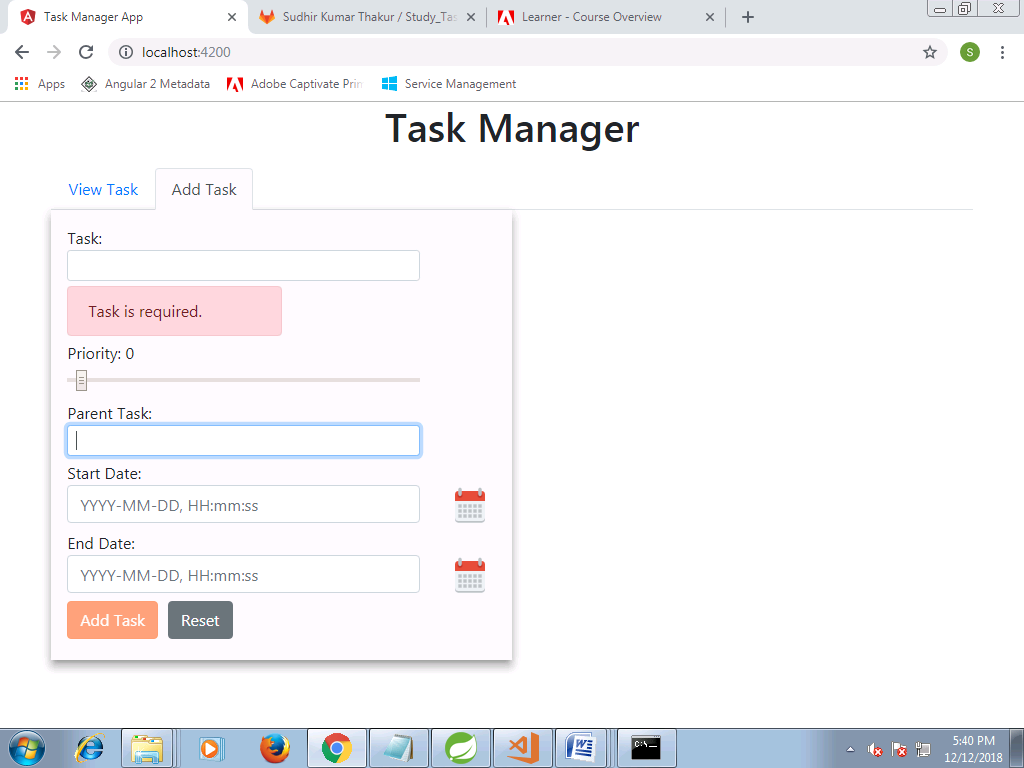
**View Tasks**

****

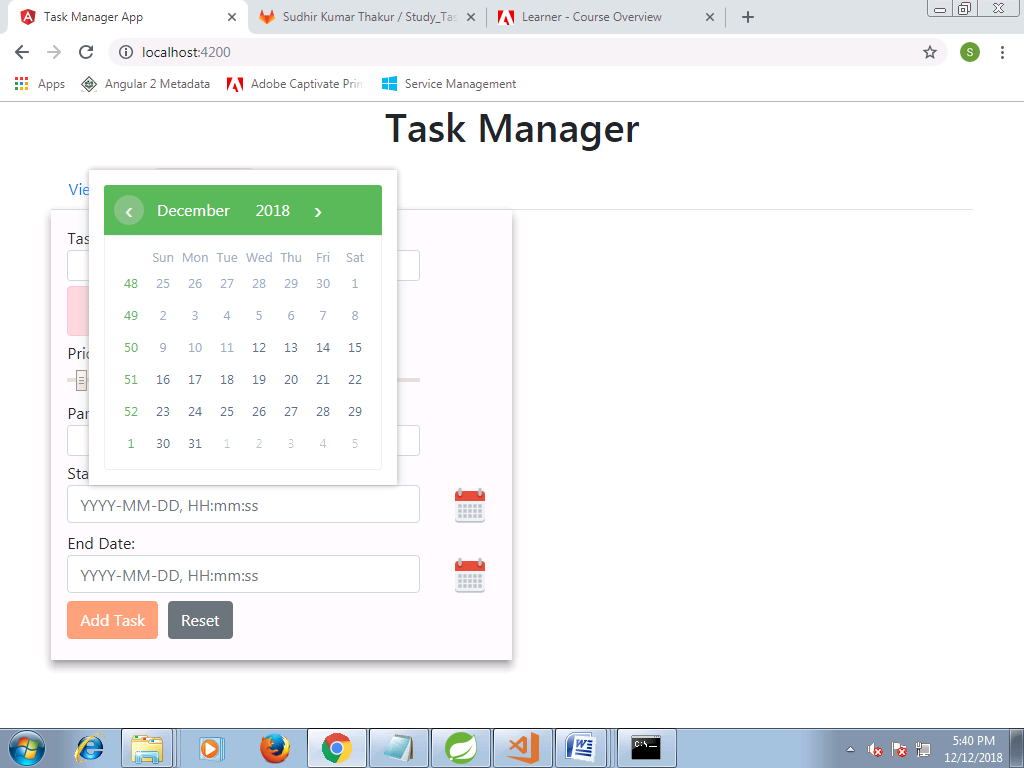
**Add Tasks**

****

**Validation msgs**

****

**Calendar popup**

****