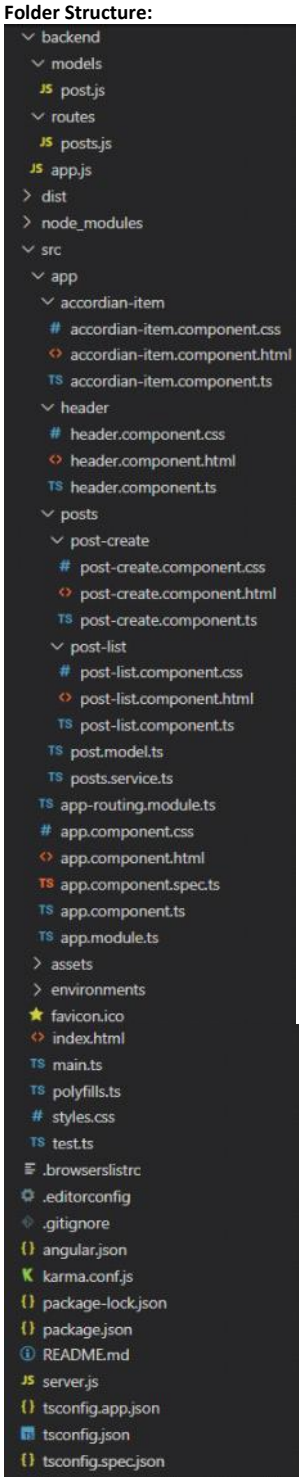
Integrating Mean Stack (Project Based)

22 April 2022 07:42

Folder Structure:



models>post.js (used to import mongoDB and create a schema and collection for the database)

```
backend > models > JS post.js > ...
1   //import statement
2   const mongoose = require('mongoose');
3   //schema creating with type declaration
4   const postSchema = mongoose.Schema({
5       title:{ type:String , required:true },
6       content:{type:String , required:true }
7   });
8   //Exporting schema into a model to utilize it into other files
9   module.exports = mongoose.model('Post',postSchema);
```

app.js (used to establish a connection to the database and create a body of API mapping which we created in another file called routes>posts.js, which we import it into app.js)

backend > app.js > ...

```
//import express for connecting front end to the database
//body parser to parse the body of the dataset to the json
//routes>posts.js contain the api calls declaration
//It contain for which type of call (get,put,post) we need to call which type of api
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const postsRoutes = require("./routes/posts");
const app = express();
//connecting to the database
mongoose.connect("mongodb+srv://JeevanSai:TdZSmDVP1j62OiV6@cluster0.i6gf1.mongodb.net/myFirstDatabase?retryWrites=true&w=majority").then(()=>{
    console.log("Connected to Database!");
}).catch(()=>{
    console.log("Connection failed!");
});
//adding middlewares
app.use(bodyParser.json());
//as we need to see the USER data on the url , we keep urlencoded as false
app.use(bodyParser.urlencoded({extended:false}));
//Giving extra permissions for the app.js
app.use((req,res,next)=>{
    res.setHeader('Access-Control-Allow-Origin','*');
    res.setHeader('Access-Control-Allow-Headers',
    'Origin,X-Requested-With,Content-Type,Accept');
    res.setHeader('Access-Control-Allow-Methods','GET,PUT,POST,PATCH,DELETE,OPTIONS');
    next();
});
//Adding API routes
app.use(postsRoutes);
//To utilize we need to import
module.exports=app;
```

routes>posts.js (For which type of request to the server - which API must be called)

- Here we are using API for the data transfer from frontend to the backend

posts.js

backend > routes > posts.js > ...

```
const express = require("express");
const router = express.Router();
const Post = require("../models/post");
router.post("/api/posts",(req,res,next)=>{
    //while posting we are creating a similar syntax to add into the database as the database
    const post = new Post({
        _id:req.body.id,
        title:req.body.title,//accessing title and the content using request object and by parsing it through body parser
        content:req.body.content
    });
    //If the post method is successful and post added successfully to the database
    //Then return back the response to the posts.service.ts , because call to this post function will come
    //from the posts.service.ts , 201 is the success response.
    post.save().then(createdPost=>{
        res.status(201).json({
            message:'Post added successfully',
            postId:createdPost._id
        });
    });
})
//put is used for updating the posts, while updating we send id of the database to the call
router.put("/api/posts/:id",(req,res,next)=>{
    const post = new Post({
        _id: req.body.id,
        title:req.body.title,
        content:req.body.content
    });
    //using that id we update the post
    Post.updateOne({_id:req.params.id},post).then(result => {
        console.log(result);
        res.status(200).json({message:"Update Successful!"});
    });
});
//get is used to retrieve a particular post
router.get("/api/posts/:id",(req,res,next)=>{
    //after getting updated we need to get that post based on id
    Post.findById(req.params.id).then(post =>{
        if(post){
            res.status(200).json(post);
        }else{
            res.status(404).json({message:'Post not found!'});
        }
    });
});
```

```
//This get method is used to retrieve all the posts , irrespective of ID
router.get("/api/posts",(req,res,next)=>{
    Post.find().then(documents => {
        res.status(200).json({
            message:'Posts fetched successfully!',
            posts:documents
        });
    });
});
//Deleting the post with respective to ID
router.delete("/api/posts/:id",(req,res,next)=>{
    Post.deleteOne({_id:req.params.id}).then(result => {
        console.log(result);
        res.status(200).json({message:"Post Deleted!"});
    });
});
module.exports = router;
```

app-routing.module.ts (we declare for which url link which component must be rendered)

app-routing.module.ts

src > app > app-routing.module.ts > routes

```
import { NgModule } from '@angular/core';
//This import is necessary for routing operations
import { RouterModule, Routes } from '@angular/router';
import { PostCreateComponent } from './posts/post-create/post-create.component';
import { PostListComponent } from './posts/post-list/post-list.component';
//Routes is a javascript objects where we define for which url which app must be represented
//This array holds the mapping
const routes: Routes = [
  //after reload we are defining to load it to the post list component page
  //localhost:4200/create renders PostCreateComponent
  {  path:'', component:PostListComponent  },
  { path:'create', component:PostCreateComponent}, //localhost:4200/create
  { path: 'edit/:postId',component:PostCreateComponent} //localhost:4200/edit/id
];

@NgModule({
  //Here we are configuring our router module by importing RouterModule and using forRoot function
  //After configuring we are exporting the configured RouterModule
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

accordian-item.component.css

accordian-item.component.css

src > app > accordian-item > # accordian-item.component.css > .mat-expansion-panel

```
.mat-expansion-panel{
    margin-top: 1rem;
}
```

accordian-item.component.html

accordian-item.component.html

src > app > accordian-item > accordian-item.component.html > mat-expansion-panel > mat-action-row

```
<mat-expansion-panel>
    <mat-expansion-panel-header>
        {{item.title}}
    </mat-expansion-panel-header>
    {{item.content}}
    <mat-action-row>
        <a mat-button color="primary" [routerLink]="['/edit',item.id]">EDIT</a>
        <button mat-button color="warn" (click)="onDelete(item.id)">DELETE</button>
    </mat-action-row>
</mat-expansion-panel>
```

accordian-item.component.ts

accordian-item.component.ts

src > app > accordian-item > accordian-item.component.ts > ...

```
import { Component, Input } from "@angular/core";
import { PostsService } from "../posts/posts.service";
@Component({
    selector:'app-accordian-item',
    templateUrl:'accordian-item.component.html',
    styleUrls:['accordian-item.component.css']
})
export class AccordianItemComponent{
    @Input() item:any;
    ngOnInit(){

    }
    constructor(public postsService:PostsService){ }
    onDelete(postId:string){
        this.postsService.deletePost(postId);
    }
}
```

header.component.html

header.component.html

src > app > header > header.component.html > mat-toolbar > span.spacer

Go to component

```
<mat-toolbar color="primary">
    <!-- RouterLinks are used to route the links to the pages according to the map -->
    <!-- That we declared at app-routing.module -->
    <!-- We should not give the same link paths for routing module and backend  -->
    <!-- Cliend side routing is all about reading the url and re rendering part's of the page -->
    <!-- Server side routing is about handling incoming requests and sending back the response -->
    <span><a routerLink="/">My Messages</a></span>
    <!-- spacer class is used to push new post section to the extreme right  -->
    <span class="spacer"></span>
    <!-- routerLinkActive is used to style the css when routerLink is Active -->
    <!-- Angular Router routes the pages but not do the operations of backend which are defined in app.js -->
    <!-- Both these files are necessary app-routing.module.ts and app.js -->
    <ul>
        <li>
            <a mat-button routerLink="/create" routerLinkActive="mat-accent">NewPost</a>
        </li>
    </ul>
</mat-toolbar>
```

header.component.css

src > app > header > # header.component.css > .spacer

```
ul{
    list-style: none;
    padding:0;
    margin:0;
}
a{
    text-decoration: none;
    color: white
}
.spacer{
    flex: 1 1 auto;
}
```

header.component.ts

src > app > header > header.component.ts > HeaderComponent

```
import {Component} from "@angular/core";

@Component({
    selector:'app-header',
    templateUrl:'./header.component.html',
    styleUrls:['./header.component.css']
})
export class HeaderComponent{}
```

post-create.component.html

src > app > posts > post-create > post-create.component.html > ...

Go to component

```
<mat-card>
    <!-- mat spinner is used to display loading gif while the contents are loading , uploading , or retrieving -->
    <!-- The logic for this will be in the typescript files -->
    <!-- The following code tells that If the current status of the page is loading then don't display the form -->
    <!-- And if it is not loading i.e. all the operations are done at the backend , show the form for the new input -->
    <!-- style.width.vw is an angular way of declaration of styles , we can do it in css way also -->
    <!-- ? in a variable tells us that , if we are having a not null value in that variable then present it,
        or else keep it blank even we are having required condition -->
    <mat-spinner *ngIf="isLoading"></mat-spinner>
    <form (submit)="onSavePost(postForm)" #postForm="ngForm" *ngIf="!isLoading">
        <mat-form-field [style.width.vw]=70>
            <input matInput type="text" name="title" [ngModel]="post?.title" required minlength="3" placeholder="Post Title" #title="ngModel">
            <mat-error *ngIf="title.invalid">Please Enter a post Title</mat-error>
        </mat-form-field>
        <br>
        <mat-form-field [style.width.vw]=70>
            <textarea matInput rows="6" name="content" [ngModel]="post?.content" required placeholder="Post Content" #content="ngModel"></textarea>
            <mat-error *ngIf="content.invalid">Please Enter the Post Content</mat-error>
        </mat-form-field>
        <br>
        <button mat-raised-button color="accent" type="submit">Save Post</button>
    </form>
</mat-card>

```

post-create.component.css

post-create.component.css

```
mat-form-field,textarea{
    width:40%;
}
mat-spinner{
    margin:auto;
}
```

post-create.component.ts

```
import { Component, OnInit, Output } from "@angular/core";
import { NgForm, NgModel } from "@angular/forms";
import { ActivatedRoute, ParamMap } from "@angular/router";

import { Post } from "../post.model";
import { PostsService } from "../posts.service";
@Component({
    selector:'app-post-create',
    templateUrl:'./post-create.component.html'
})
export class PostCreateComponent implements OnInit{
    enteredTitle = "";
    enteredContent = "";
    private mode = 'create';
    private postId:any;
    isLoading=false;
    post: Post = {id:null!,title:null!,content:null!};
    onSavePost(form: NgForm){
        if(form.invalid){
            return;
        }
        this.isLoading=true;
        if(this.mode==='create'){
            this.postService.addPost(form.value.title,form.value.content);
        }
        else{
            this.postService.updatePost(this.postId,form.value.title,form.value.content);
        }
        form.resetForm();
    }
    ngOnInit() {
        //paramMap is an Observable that we can subscribe
        //Here we are observing that is there any changes in the URL and if any change in the URL
        //Angular tries to make changes in part of the component rather than the whole component
        this.route.paramMap.subscribe((paramMap:ParamMap)=>{
            if(paramMap.has('postId')){
                this.mode = 'edit';
                this.postId = paramMap.get('postId');
                this.isLoading=true;
                this.postService.getPost(this.postId).subscribe(postData =>{
                    this.isLoading=false;
                    this.post = { id:postData._id,title:postData.title,content:postData.content};
                });
                console.log(this.mode,this.postId,this.post.title,this.post.content);
                //The following two lines are no need , if we use elvis operator on input fields
                //We can use elvis instead of binding the data conditionally
                // this.enteredTitle=this.post.title;
                // this.enteredContent=this.post.content;
            }else{
                this.mode = 'create';
                this.postId = null;
            }
        });
    }
    constructor(public postService:PostsService , public route: ActivatedRoute){}
}
```

post-list.component.html

```
<mat-spinner *ngIf="isLoading"></mat-spinner>
<mat-accordion multi="true" *ngIf="items.length>0 && !isLoading">
    <app-accordian-item *ngFor="let item of items" [item]="item"></app-accordian-item>
</mat-accordion>
<ng-template #nopost><p class="info-text mat-body-1" *ngIf="items.length<=0 && !isLoading">No Posts are available</p></ng-template>
```

post-list.component.css

post-list.component.css

src > app > posts > post-list > # post-list.component.css > mat-spinner

```css
:host{
    display:block;
    margin-top:1rem;
}
.info-text{
    text-align: center;
}
mat-spinner{
    margin:auto;
}
```

post-list.component.ts

TS post-list.component.ts

src > app > posts > post-list > TS post-list.component.ts > PostListComponent > ngOnInit > subscribe() callback

```ts
import { Component,  OnDestroy,  OnInit } from "@angular/core";
import { Subscription } from "rxjs";
import {Post} from '../post.model';
import { PostsService } from "../posts.service";

@Component({
    selector:'app-post-list',
    templateUrl:'post-list.component.html'
})
export class PostListComponent implements OnInit,OnDestroy{
    flag=false;
    items:Post[]=[];
    isLoading=false;
    private postsSub: Subscription = new Subscription;
    greaterThan(n:any){
        if(n>0){
            return true;
        }
        return false;
    }

    constructor(public postsService:PostsService){ }

    ngOnInit(): void {
        this.isLoading=true;
        this.postsService.getPosts();
        this.postsSub=this.postsService.getPostUpdateListener().subscribe((posts:Post[])=>{
            this.isLoading=false;
            this.items=posts;
        });
    }
    ngOnDestroy(): void {
        this.postsSub.unsubscribe();
    }
}
```

post.model.ts

TS post.model.ts

src > app > posts > TS post.model.ts > Post

```ts
//id is used to store the id of the post
export interface Post{
    id:string;
    title:string;
    content:string;
}
```

posts.service.ts

posts.service.ts

```ts
import { Injectable } from "@angular/core";
import { Post } from "./post.model";
import { Subject } from 'rxjs';
import { HttpClient } from "@angular/common/http";
import {map} from 'rxjs/operators';
import { Router } from "@angular/router";
//Injectable is used add the post-service to app.module.ts
@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    private postsUpdated = new Subject<Post[]>();
    constructor(private http:HttpClient, private router:Router){}
    getPosts(){
        this.http.get<{message:string,posts:any[]}>('http://localhost:3000/api/posts')
        .pipe(map((postData)=>{
            return postData.posts.map(post => {
                return{
                    title:post.title,
                    content:post.content,
                    id:post._id
                };
            });
        }))
        .subscribe(transformedPosts => {
            this.posts=transformedPosts;
            this.postsUpdated.next([...this.posts]);
        });
    }
    getPostUpdateListener(){
        return this.postsUpdated.asObservable();
    }
    getPost(id:string){
        return this.http.get<{_id:string,title:string,content:string}>("http://localhost:3000/api/posts/"+id);
    }
    addPost(title:string,content:string){
        const post:Post={id:null!,title:title,content:content};
        this.http.post<{message:string,postId:string}>('http://localhost:3000/api/posts',post).subscribe((responseData)=>{
            const id=responseData.postId;//the id of the added post is retrived from database and stored locally, for consistency
            post.id=id;
            this.posts.push(post);
            this.postsUpdated.next([...this.posts]);
            this.router.navigate(["/"]);//after adding a post we need to redirect to the '/'
            //it mean we declared it to redirect to postListComponent in app-routing.module.ts
        });
    }
    updatePost(id:string , title:string , content: string){
        const post: Post = {id:id, title:title, content:content};
        this.http.put("http://localhost:3000/api/posts/"+id,post)
        .subscribe( response => {
            console.log(response);
            //This is not necessary because we already connected to database
            //So it will be fetched anyways, but to fetch the changes locally
            //on the backend server we do this way
            const updatedPosts = [...this.posts];
            const oldPostIndex = updatedPosts.findIndex(p => p.id === post.id);
            updatedPosts[oldPostIndex]=post;
            this.posts=updatedPosts;
            this.postsUpdated.next([...this.posts]);
            this.router.navigate(["/"]);//after updating it will be redirected to the postListComponent
        });
    }
    deletePost(postId : string){
        this.http.delete("http://localhost:3000/api/posts/"+postId)
        .subscribe(()=>{
            const updatedPosts = this.posts.filter(post=> post.id!==postId);
            this.posts=updatedPosts;
            this.postsUpdated.next([...this.posts]);
        });
    }
}
```

app.component.css

app.component.css

```css
main{
    width:80%;
    margin: 1rem auto;
}
```

app.component.html

app.component.html

```
<app-header></app-header>
  <main>
    <!-- To change pages according to the router links we need to use route-outlet -->
    <router-outlet></router-outlet>
  </main>
```

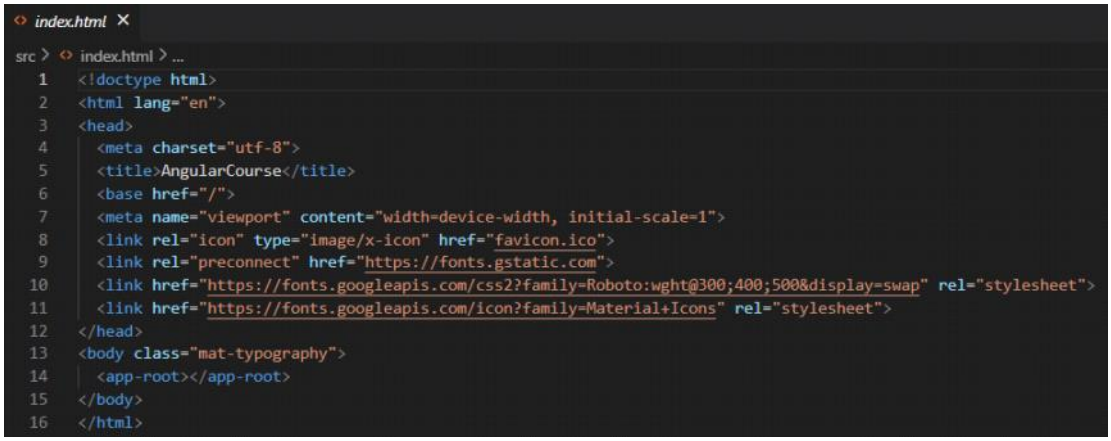
app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'angularCourse';
}
```

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import {MatToolbarModule} from '@angular/material/toolbar';
import {PostCreateComponent} from './posts/post-create/post-create.component';
import {HeaderComponent} from './header/header.component';
import {MatInputModule} from '@angular/material/input';
import {MatCardModule} from '@angular/material/card';
import {MatButtonModule} from '@angular/material/button';
import { PostListComponent } from './posts/post-list/post-list.component';
import { FormsModule } from '@angular/forms';
import {MatExpansionModule} from '@angular/material/expansion';
import { AccordianItemComponent } from './accordian-item/accordian-item.component';
import {MatProgressSpinnerModule} from '@angular/material/progress-spinner';
import {HttpClientModule} from "@angular/common/http";

@NgModule({
  declarations: [
    AppComponent,
    PostCreateComponent,
    PostListComponent,
    HeaderComponent,
    AccordianItemComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MatInputModule,
    MatCardModule,
    MatButtonModule,
    MatToolbarModule,
    FormsModule,
    MatExpansionModule,
    HttpClientModule,
    MatProgressSpinnerModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

index.html



Check the files of index.html , styles.css , angular.json , package.json , server.js from the pervious notebook of MongoDB (Project Based)