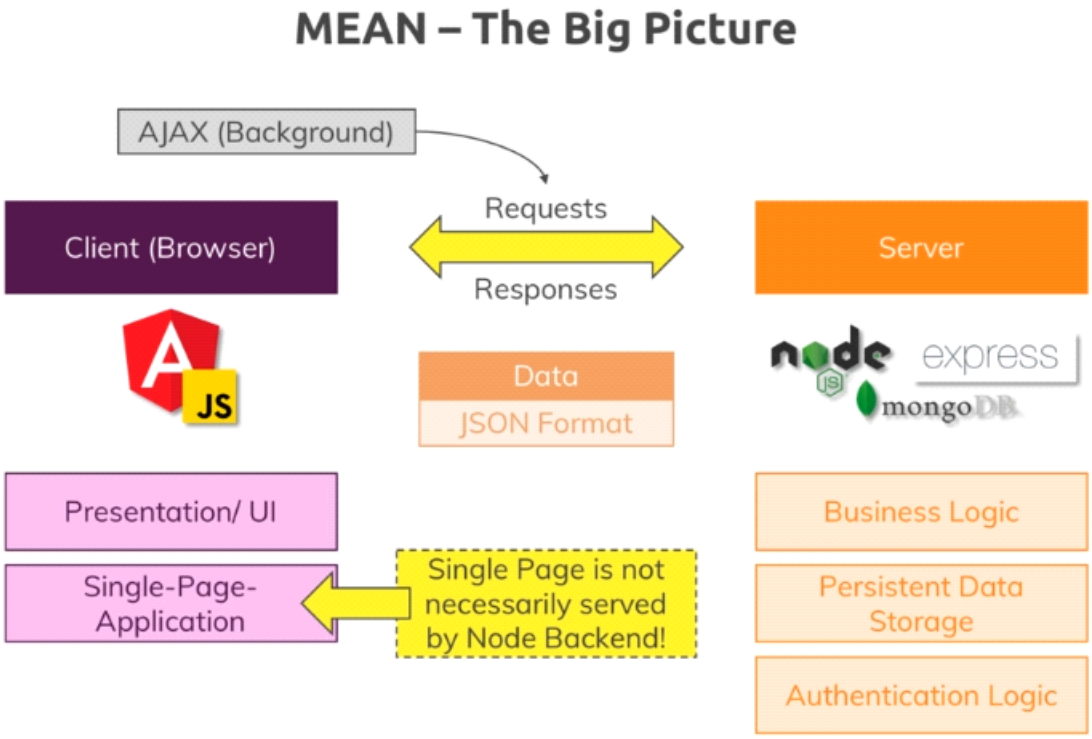
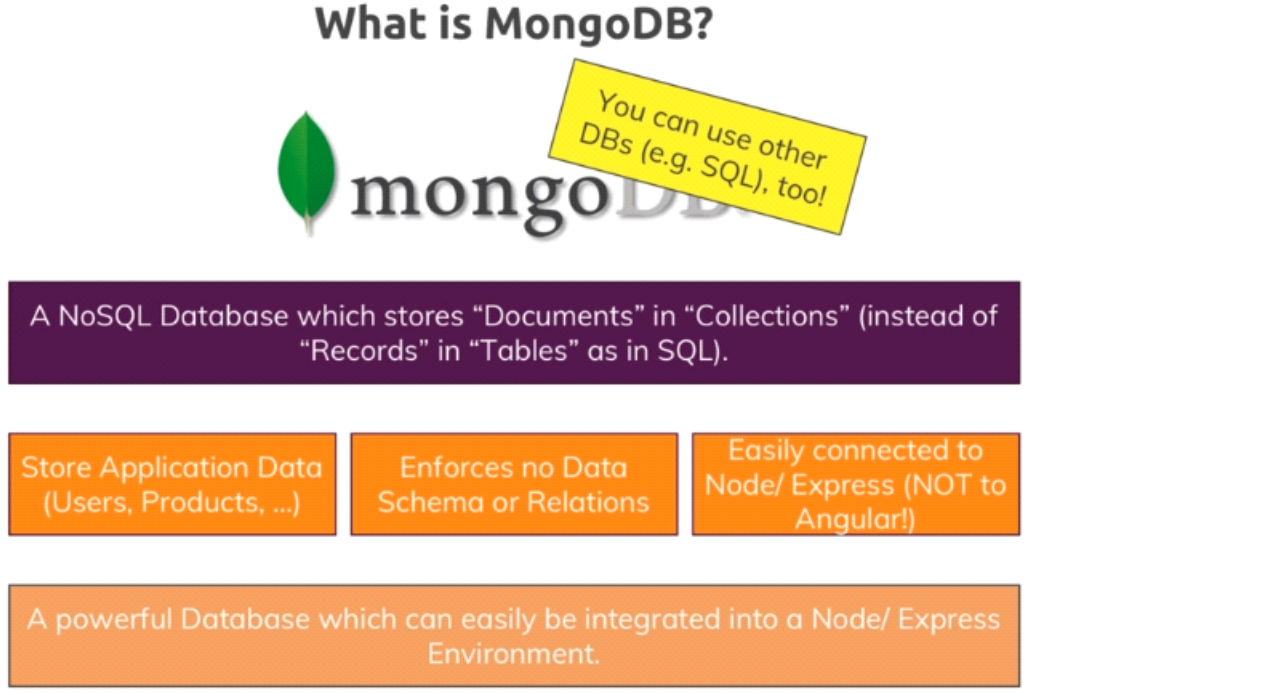
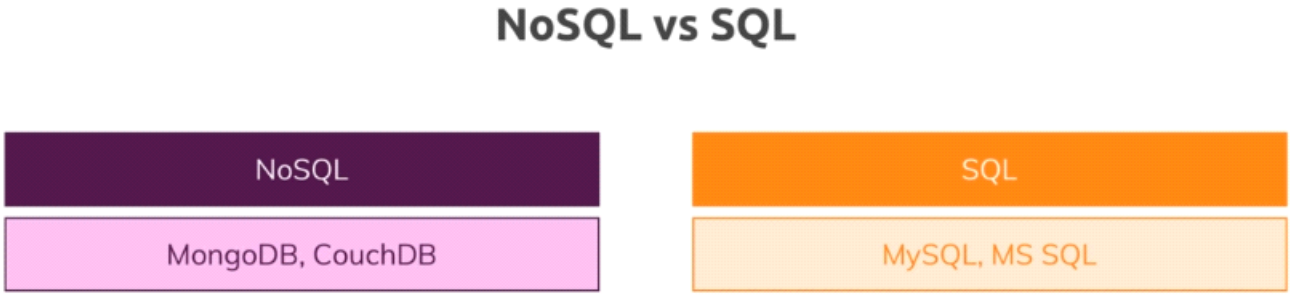
MongoDB Introduction in Project

16 April 2022 11:52

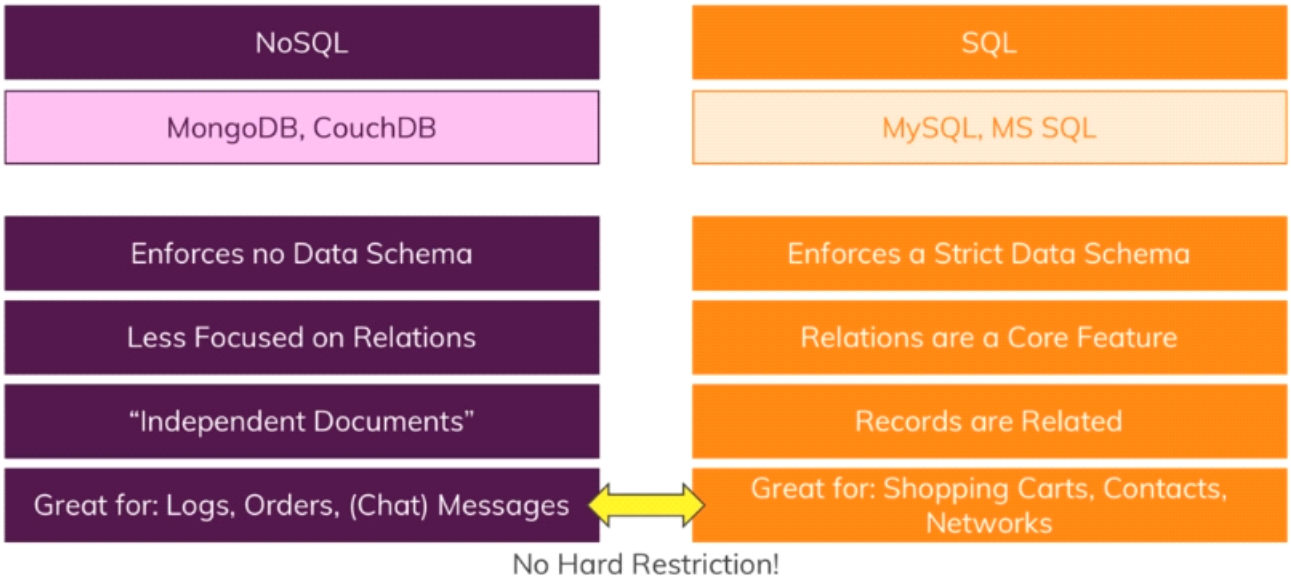


- We are going to store and persist the user data from the front end , so user will not lose his data whenever he reloads the page. And the browser persists the user data.





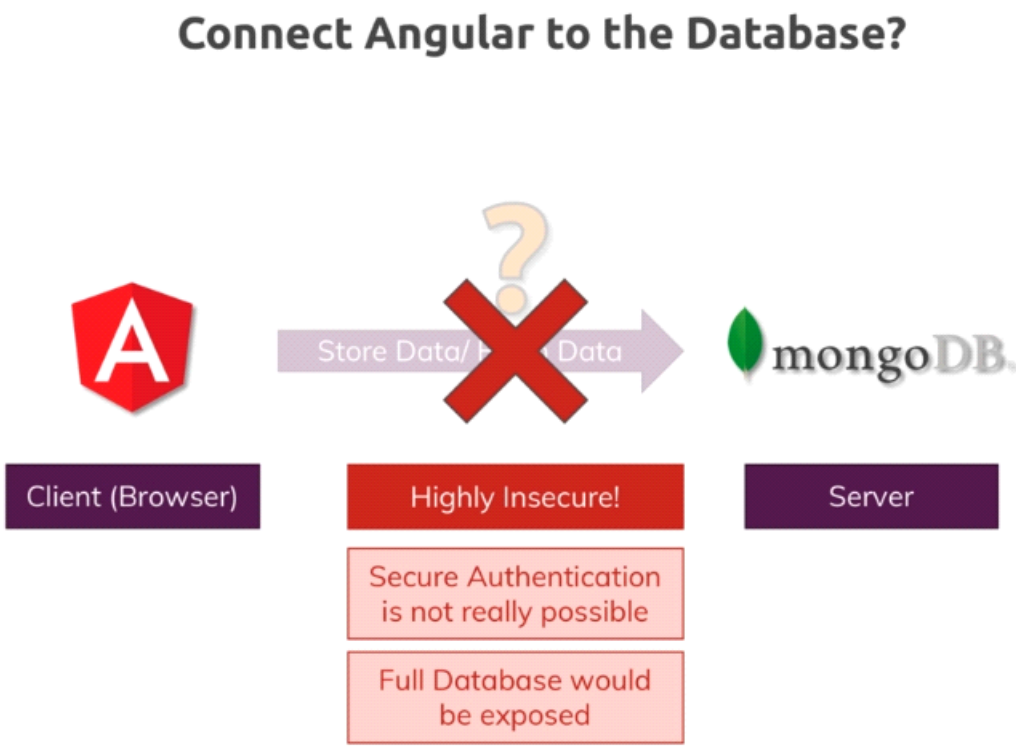
NoSQL vs SQL



No Hard Restriction!

Connect Angular to the Database

16 April 2022 16:43

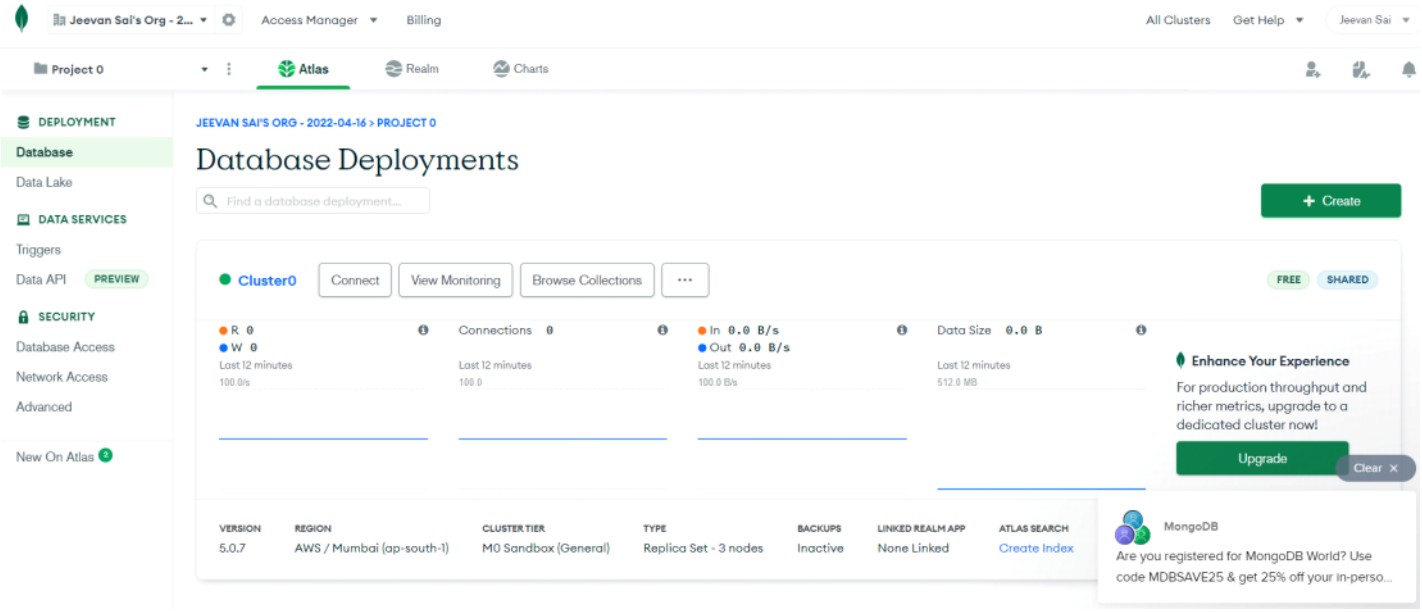


- We could connect angular to the mongoDB database directly , but it is not preferable because . If we connect it through the JavaScript all the front end code and the JavaScript code is exposed to the users and secure authentication is not really possible because , user can see the JavaScript code and surpass the security with the loop wholes and can inject into our database and our passwords got exposed. And hence it is not preferable to connect mongoDB to the angular directly. So , node and express are acting as middleware's for the mongoDB and angular, where all the authentication and database connection happens in the middleware and this middleware is not exposed to the users as it locates on the server.
- Client(Browser) <=> Middleware's (node and express) <=> Database (MongoDB)
- All node and express and database are under the backend section.

Adding Mongoose

16 April 2022 17:36

- We use mongoose for accessing mongoDB, it is a third party package that is build on mongoDB driver. It helps us , easy in accessing mongoDB in easier way and much efficient.
- But mongoose can define schema whereas original mongoDB doesn't care about schema.
- Mongoose can also work with unstructured data
- The following node package manager is used to install mongoose
- **npm install --save mongoose**
- For utilizing the mongoDB , we are using a free tier mongoDB version. Where we created a cluster with AWS server and created user name as "Jeevan" and password as "ngyCMRtgbjegrYOi" to access the mongoDB, our system local ip we given for accessing the DB is "103.155.30.14/32"
- This is the cluster that we created.

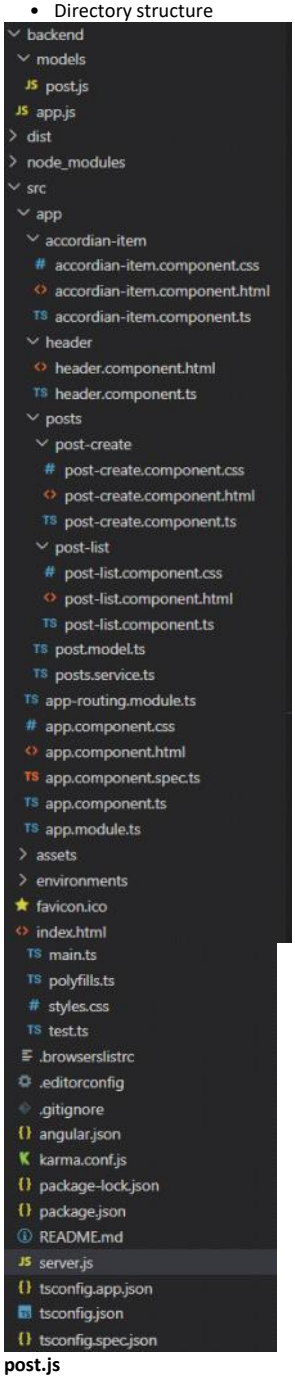


Follow this link to get a good clarity on how to connect the node express backend to the mongoDB using mongoose

(Full Project)Adding MongoDB to the Angular through Node, ExpressJS

16 April 2022 23:35

- Additionally from the previous project , here we created a extra file called post.js under models directory. And rest , we modified from the existing files. The core functionality presents in models/post.js , app.js and posts.service.ts
- Directory structure



post.js

- post.js is the base for the mongodb connection to the application.
- The created schema's and models and collections are used by the furthur files in this application.
- We created Post as the model name , and posts as the collection name, database name will be created automatically in app.js

post.js

backend > models > JS post.js > ...

```
//mainly contains mongodb related code
//import mongoose
const mongoose = require('mongoose');
//we create a schema or a blueprint on how a data must look like
//but in reality mongoDB doesn't require any schema, but mongoose requires
// for data authentication and structure
//required in the sense , that is a required input from the user
//to add into mongoDB database
const postSchema = mongoose.Schema({
    title:{ type:String , required:true },
    content:{type:String , required:true }
});
//If we want to convert schema into a model we do the following
module.exports = mongoose.model('Post',postSchema);
//If the model name is Post then the collection name of the database that we create
//Is the plural form of the model name i.e. Post => posts
//posts is the name of collection
```

app.js

app.js

backend > JS app.js > app.delete("/api/posts/:id") callback > then() callback

```
//Here we write express code
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
//Post instance is used to access mongodb
const Post = require('./models/post');
const app = express();
//we are using mongoose to connect to the mongoDB server, myFirstDatabase in this link is the database name
//The following then catch works as the if else case for the mongoose connect
//If the connection is successfull then will executed
//if the connection failed then catch will be executed
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!");
});
//added new middlewares for parsing of the body
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended:false}));
//Giving permissions
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,POST,PATCH,DELETE,OPTIONS');
    next();
});
//If we execute post operation on the database on the front end,
//the following api path route generated and do the following operation
//remeber in angular we try to do single page applications, so rest all the requests are done using API's
//This function was triggered when post-create executes which trigger posts.service.ts and which trigger's this post function of express backend
//and perform following functions, post local constant variable is used to store the user title and content in specified format
//And save it using save method to the mongoDB
app.post("/api/posts",(req,res,next)=>{
    //body is a method provided by the body parser, which helps us to convert into intended format
    // const post = req.body;
    const post = new Post({
        title:req.body.title,//accessing title and the content using request object and by parsing it through body parser
        content:req.body.content
    });
    //saves the data into the "myFirstDatabase" database,  "posts" collection
    //If the saving process done successfully then, the new post that is created is stored in createdPost
    //the response is returned to the posts.service.ts
    //which contain a message and post_id which is created by mongoDB after storing the new post
    //And this id is used to update id of local angular application, so there will be consistency between front end and backend database
    //If this operation is not done , the database data manipulations will not work properly
```

```
    post.save().then(createdPost=>{
        // console.log(result);
        //console.log(post);
        //sending the response to posts.service.ts which update local Post model id
        res.status(201).json({
            message:'Post added successfully',
            postId:createdPost._id
        });
    });
})
//app.use and app.get both function as the same
//app.get is used by the post-list.component
app.get("/api/posts",(req,res,next)=>{
    // const posts = [
    //     {
    //         id:"fadf124211",
    //         title:"First server-side post",
    //         content:"This is coming from the server"
    //     },
    //     {
    //         id:"ksajflaj132",
    //         title:"Second server-side post",
    //         content:"This is coming from the server!"
    //     }
    // ];
    // Post.find((err,documents)=>{});
    //All these functions are asynchronous in nature, so we are arranging the response inside the find operation
    //So success response will only send to the request if  the find operation is successful
    //or else response may send earlier than find operation as these operations are asynchronous in nature
    Post.find().then(documents => {
        // console.log(documents);
        res.status(200).json({
            message:'Posts fetched successfully!',
            posts:documents
        });
    });

});
//This is used for delete functionality which is called by accordian-item.component.ts => posts.service.js =>app.js
//We pass the id to delete from the front end accordian-item =>posts.service.js =>app.js
//Which uses connection of the database and pass the id parameter to the database dynamically and delete it
app.delete("/api/posts/:id",(req,res,next)=>{
    // console.log(req.params.id);
    //params is an inbuild method to fetch all the parameters, form that we use id attribute
    Post.deleteOne({_id:req.params.id}).then(result => {
        console.log(result);
        res.status(200).json({message:"Post Deleted!"});
    });
});
module.exports=app;
```

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
Go to component
<mat-expansion-panel>
    <mat-expansion-panel-header>
        {{item.title}}
    </mat-expansion-panel-header>
    {{item.content}}
    <mat-action-row>
        <button mat-button color="primary">EDIT</button>
        <!-- onDelete operation is added new in this application to connect to the database and delete a post-->
        <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

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

    }
    constructor(public postsService:PostsService){ }
    onDelete(postId:string){
        //send request to delete to posts.service.ts
        this.postsService.deletePost(postId);
    }
}
```

header.component.html

header.component.html

```html
<mat-toolbar color="primary">My Messages</mat-toolbar>
```

header.component.ts

header.component.ts

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

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

post-create.component.css

post-create.component.css

```css
mat-form-field,textarea{
    width:40%;
}

```

post-create.component.html

post-create.component.html

```html
<mat-card>
    <form (submit)="onAddPost(postForm)" #postForm="ngForm">
        <mat-form-field [style.width.vw]=70>
            <input matInput type="text" name="title" ngModel 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 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.ts

post-create.component.ts

```ts
import { Component, Output } from "@angular/core";
import { NgForm } from "@angular/forms";
import { PostsService } from "../posts.service";
@Component({
    selector:'app-post-create',
    templateUrl:'./post-create.component.html'
})
export class PostCreateComponent{
    enteredTitle = "";
    enteredContent = "";

    onAddPost(form: NgForm){
        if(form.invalid){
            return;
        }
        this.postService.addPost(form.value.title,form.value.content);
        form.resetForm();
    }
    constructor(public postService:PostsService){}
}
```

post-list.component.css

post-list.component.css

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

post-list.component.html

```html
<mat-accordion multi="true" *ngIf="greaterThan(items.length) else nopost">
    <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">No Posts are available</p></ng-template>
```

post-list.component.ts

```typescript
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[]=[];
    private postsSub: Subscription = new Subscription;
    // items=[
    //     {title:"First Post",content:"This is the first post\'s content"},
    //     {title:"Second Post",content:"This is the second post\'s content"},
    //     {title:"Third Post",content:"This is the third post\'s content"},
    // ];
    greaterThan(n:any){
        if(n>0){
            return true;
        }
        return false;
    }

    constructor(public postsService:PostsService){ }

    ngOnInit(): void {
        //Posts are retreived from an API that we declared using the backend server(localhost:3000/api/hosts).
        this.postsService.getPosts();
        //those posts are retrieved as an observable and updated the local variable items
        //So the front end automatically gets updated as this items private variable is passed to accordian-item and
        //gets printed on the front end
        this.postsSub=this.postsService.getPostUpdateListener().subscribe((posts:Post[])=>{
            this.items=posts;
        });
    }
    ngOnDestroy(): void {
        this.postsSub.unsubscribe();
    }
}
```

post.model.ts

```typescript
//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';

@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    private postsUpdated = new Subject<Post[]>();
    constructor(private http:HttpClient){}
    //Now we established a connection to the database and so that we can get all existing posts
    //And send them for mapping into the pipes,
    //local private variable get's accumulated as usual after retrieving data through get
    getPosts(){
        this.http.get<{message:string,posts:any[]}>('http://localhost:3000/api/posts')
        .pipe(map((postData)=>{ //from backend we are getting all the posts in an array with id in it and an attached message with it
            return postData.posts.map(post => { //now we are using only the posts part of it. And iterating through it and mapping it to the new pattern
                return{
                    title:post.title,
                    content:post.content,
                    id:post._id
                };
            });
        }))//all the above process is used to map the data of the database to the pattern of local
        //And after transforming we subscribe it and save the transformedPosts to the local , to display it on frontend
        .subscribe(transformedPosts => {
            this.posts=transformedPosts;
            this.postsUpdated.next([...this.posts]);
        });
    }
    getPostUpdateListener(){
        return this.postsUpdated.asObservable();
    }
    //we are calling the post method with it's path declared in app.js
    //And if the API call is successful a message is returned from app.js and store it in
    //responseData and print that in the console and add the new post locally
    //We can see after adding data the change occured in database and the database logic
    //is written on app.js which is written in express
    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)=>{
            // console.log(responseData.message);
            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]);
        });
    }
    deletePost(postId : string){
        this.http.delete("http://localhost:3000/api/posts/"+postId)
        .subscribe(()=>{
            // console.log('Deleted!');
            //updating local posts array to display changes on the front end
            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

```html
<app-header></app-header>
<main>
  <app-post-create></app-post-create>
  <app-post-list></app-post-list>
</main>
```

app.component.ts

app.component.ts

```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

```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 {HttpClientModule} from "@angular/common/http";

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

angular.json

```
"styles": [
  "./node_modules/@angular/material/prebuilt-themes/indigo-pink.css",
  "src/styles.css"
],
```

package.json

```
{
  "name": "angular-course",
  "version": "0.0.0",
  ▷ Debug
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "start:server": "nodemon server.js"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~13.3.0",
    "@angular/cdk": "^13.3.2",
    "@angular/common": "~13.3.0",
    "@angular/compiler": "~13.3.0",
    "@angular/core": "~13.3.0",
    "@angular/forms": "~13.3.0",
    "@angular/material": "^13.3.2",
    "@angular/platform-browser": "~13.3.0",
    "@angular/platform-browser-dynamic": "~13.3.0",
    "@angular/router": "~13.3.0",
    "body-parser": "^1.20.0",
    "express": "^4.17.3",
    "mongoose": "^6.3.0",
    "rxjs": "~7.5.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.11.4"
  },
```

server.js

server.js

server.js > onError

```
1   //Here we write nodejs code
2   const http = require('http');
3   const debug = require("debug")("node-angular");
4   const { listen } = require('./backend/app');
5   const app = require('./backend/app');
6   const normalizePort = val => {
7       var port = parseInt(val,10);
8       if(isNaN(port)){
9           //named pipe
10          return val;
11      }
12      if(port>=0){
13          //port number
14          return port;
15      }
16      return false;
17  };
18  const onError = error => {
19      if(error.syscall !==listen){
20          throw error;
21      }
22      const bind = typeof addr ==="string" ? "pipe"+addr:"port"+port;
23      switch(error.code){
24          case "EACCES":
25              console.error(bind+" requires  elevated privileges ");
26              process.exit(1);
27              break;
28          case "EADDRINUSE":
29              console.error(bind+" is already in use");
30              process.exit(1);
31              break;
32          default:
33              throw error;
34      }
35  };
36  const onListening = () => {
37      const addr = server.address;
38      const bind = typeof addr ==="string" ? "pipe"+addr:"port"+port;
39      debug("Listening on "+bind);
40  };
41  const port = normalizePort(process.env.PORT || "3000");
42  app.set('port',port);
43  const server = http.createServer(app);
44  server.on("error",onError);
45  server.on("listening",onListening);
46  server.listen(port);
```