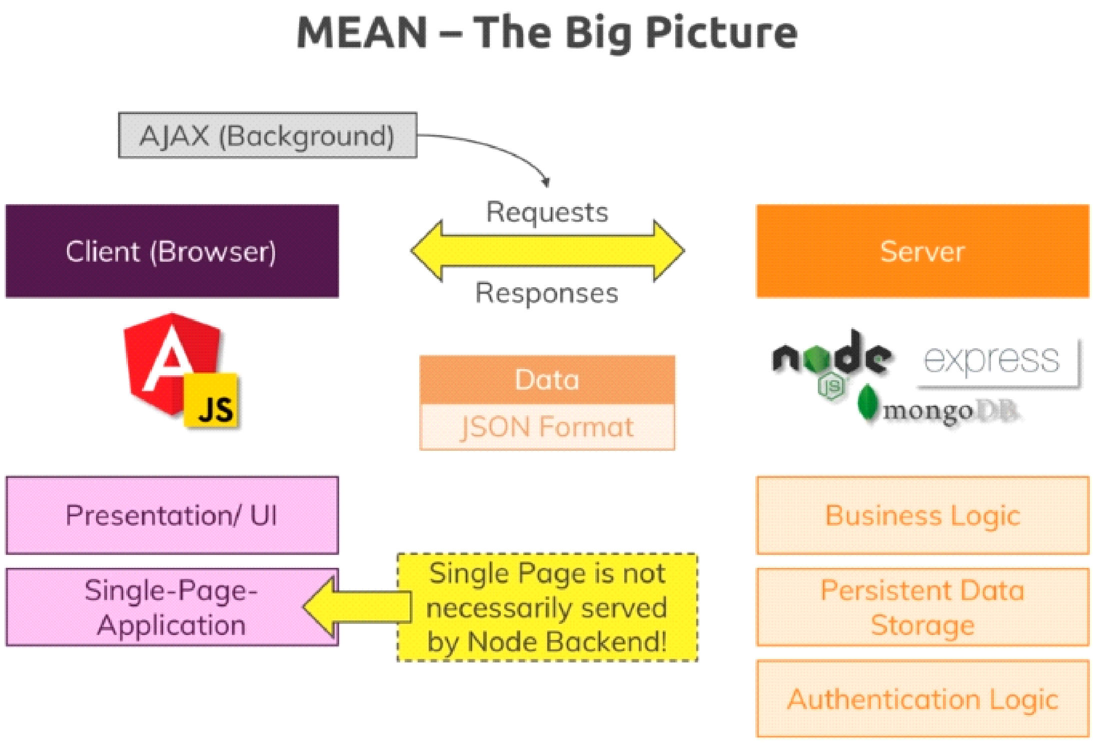
Introduction

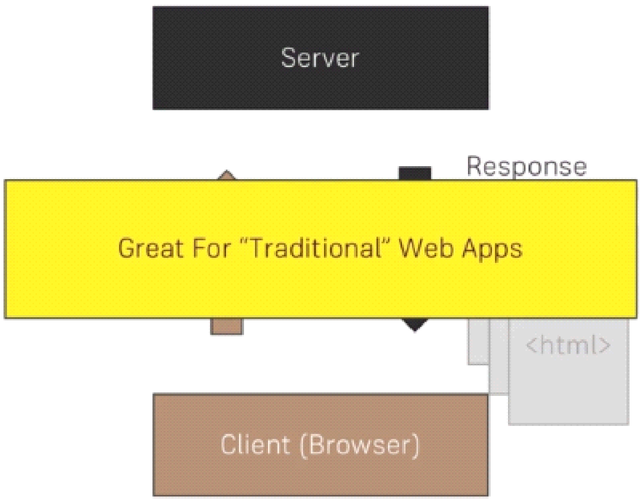
15 April 2022 16:57

- Ng Serve is used to connect the angular to local server . Ng serve gives us a development server.
- Backend uses a different server but not same as angular.



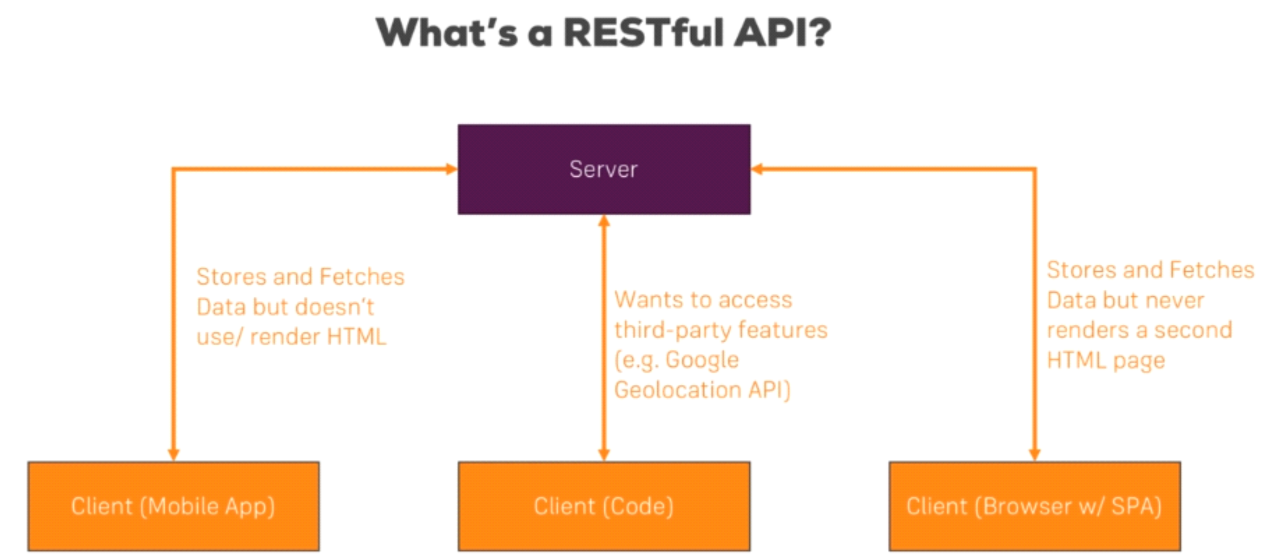
- We are having a node express application which has it's own server and we are having some url sections where we can utilize those to connect the front end. Or we can execute them on individual servers , one for the front end and one for the backend. Both the methods are similar but they differ only at the deployment stage.
- Angular uses HTTP requests to the node express backend and those requests must be handled by the node and express and send the response back to the front end.
- There is no direct connection between front and backend they are connected by the restful API. And it is build by the node js.
- REST mean representational state transfer.

For traditional apps

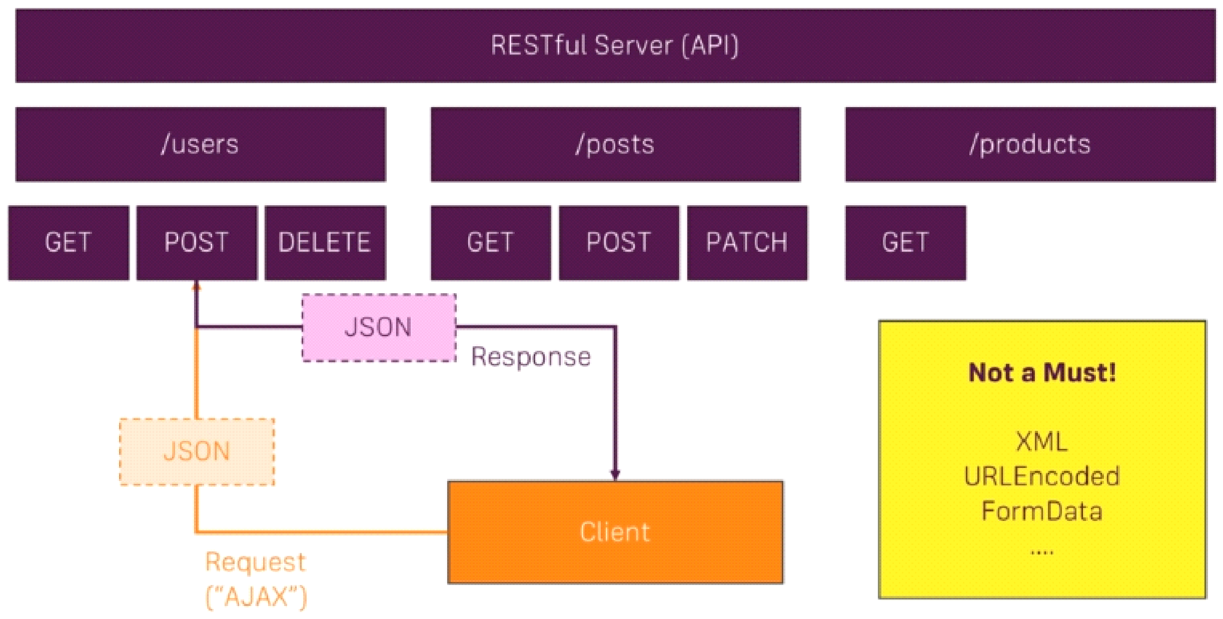


- Browser sends the request to the server and that server sends the html pages as the response in the traditional web apps.

For angular apps


What's a RESTful API?
Server
Stores and Fetches Data but doesn't use/ render HTML
Wants to access third-party features (e.g. Google Geolocation API)
Stores and Fetches Data but never renders a second HTML page
Client (Mobile App)
Client (Code)
Client (Browser w/ SPA)

- In angular apps , there is no necessity of handling web pages by the server . Because everything about webpages are handled by the JavaScript it self . We just need to transaction in data from front end to the backend using restful API.
- Restful API serves differently for different clients.
- Restful API's are stateless Backends.



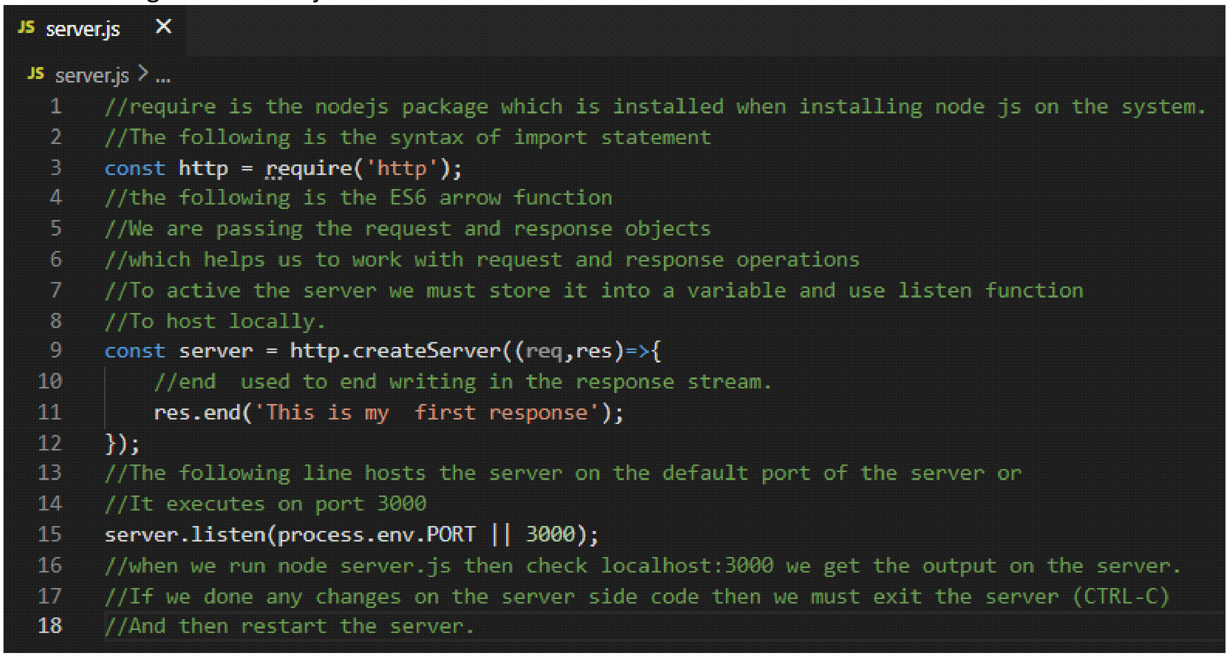
- Here we are going to create the Restful API through node backend hence, we can define our own path URLs and what permissions and methods to give access for those paths. And client who is using that API will access the API through AJAX and the data format that is used for data transaction is JSON format , we can use other formats but that is not mandatory . It is preferred to use JSON file format.

Adding Node and Express Backend

15 April 2022 17:46

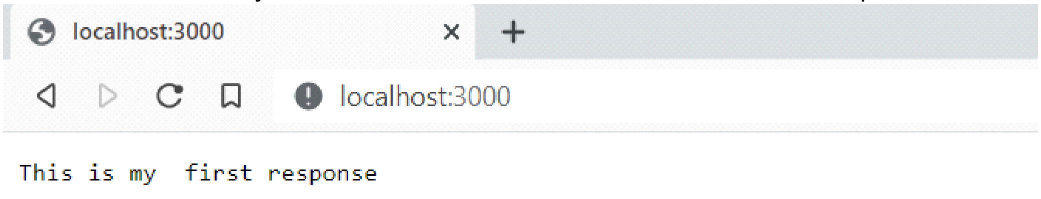
Create a new directory backend in the current project and a file called server.js to execute the node application.
Node will be already installed on the system as we have downloaded node.js on the system during angular installation.

The following is the server.js code



```
//require is the nodejs package which is installed when installing node js on the system.
//The following is the syntax of import statement
const http = require('http');
//the following is the ES6 arrow function
//We are passing the request and response objects
//which helps us to work with request and response operations
//To active the server we must store it into a variable and use listen function
//To host locally.
const server = http.createServer((req,res)=>{
    //end  used to end writing in the response stream.
    res.end('This is my  first response');
});
//The following line hosts the server on the default port of the server or
//It executes on port 3000
server.listen(process.env.PORT || 3000);
//when we run node server.js then check localhost:3000 we get the output on the server.
//If we done any changes on the server side code then we must exit the server (CTRL-C)
//And then restart the server.
```

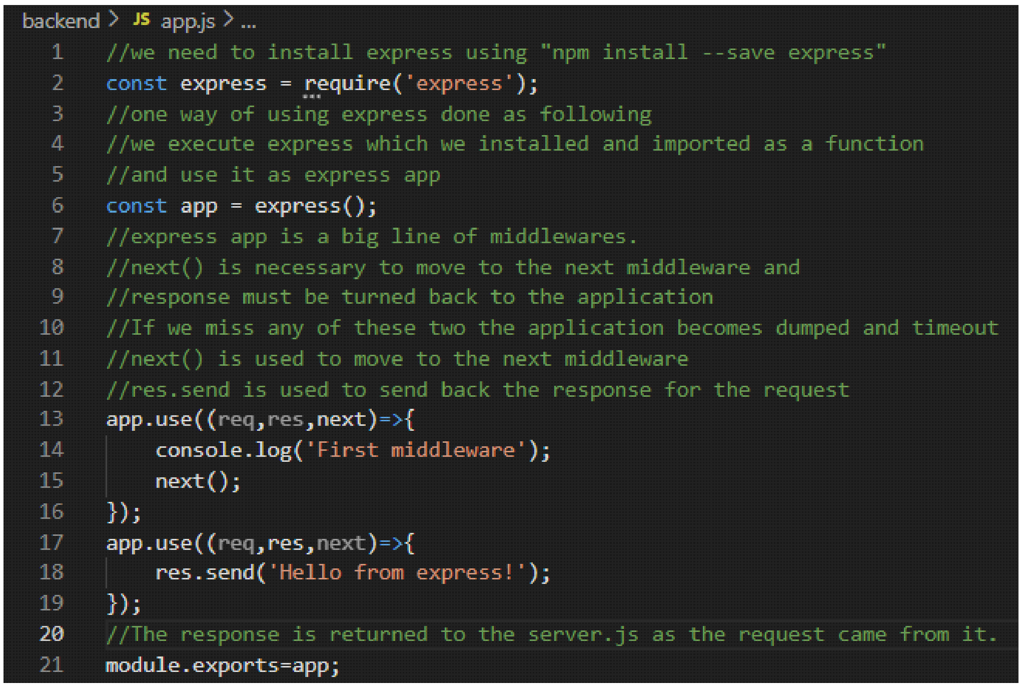
Execute "node server.js" in the terminal and check the browser for the output on localhost:3000



```
This is my  first response
```

- To install express on our system, we have to use the following command "npm install --save express"
- And then create a file called app.js under the backend folder .

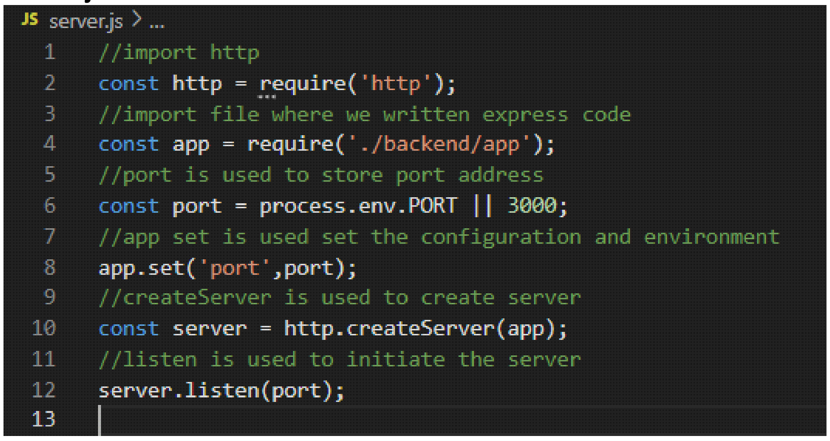
backend/app.js


backend > app.js > ...

```javascript
//we need to install express using "npm install --save express"
const express = require('express');
//one way of using express done as following
//we execute express which we installed and imported as a function
//and use it as express app
const app = express();
//express app is a big line of middlewares.
//next() is necessary to move to the next middleware and
//response must be turned back to the application
//If we miss any of these two the application becomes dumped and timeout
//next() is used to move to the next middleware
//res.send is used to send back the response for the request
app.use((req,res,next)=>{
    console.log('First middleware');
    next();
});
app.use((req,res,next)=>{
    res.send('Hello from express!');
});
//The response is returned to the server.js as the request came from it.
module.exports=app;
```

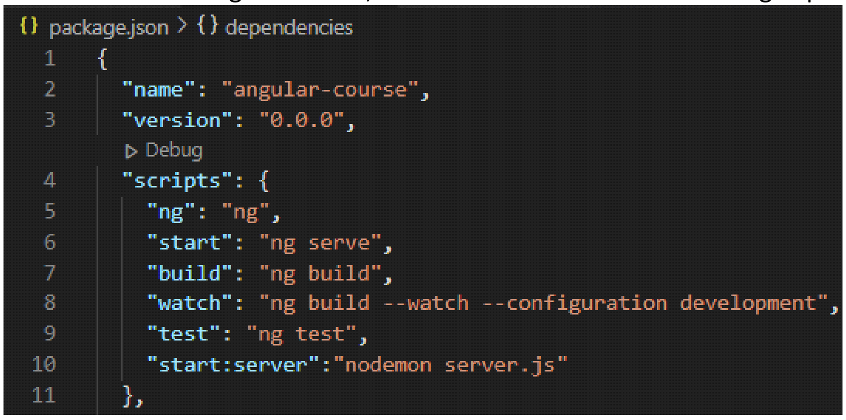
- app.js sends the response for the request which is generated by the server.js
- App.js is written in express and server.js is written in node.js
- Express is a framework of node.js
- We connect express to the node using port numbers that we declared on the server.js

server.js


server.js > ...

```javascript
//import http
const http = require('http');
//import file where we written express code
const app = require('./backend/app');
//port is used to store port address
const port = process.env.PORT || 3000;
//app set is used set the configuration and environment
app.set('port',port);
//createServer is used to create server
const server = http.createServer(app);
//listen is used to initiate the server
server.listen(port);

```

- To prevent repeated exiting and entering the node and express server , we use another dependency ,which will be installed using "**npm install --save-dev nodemon**"
- After installing nodemon , declare "start:server" as following in package.json

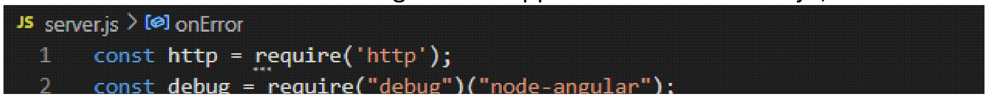

package.json > {} dependencies

```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"
  },
```

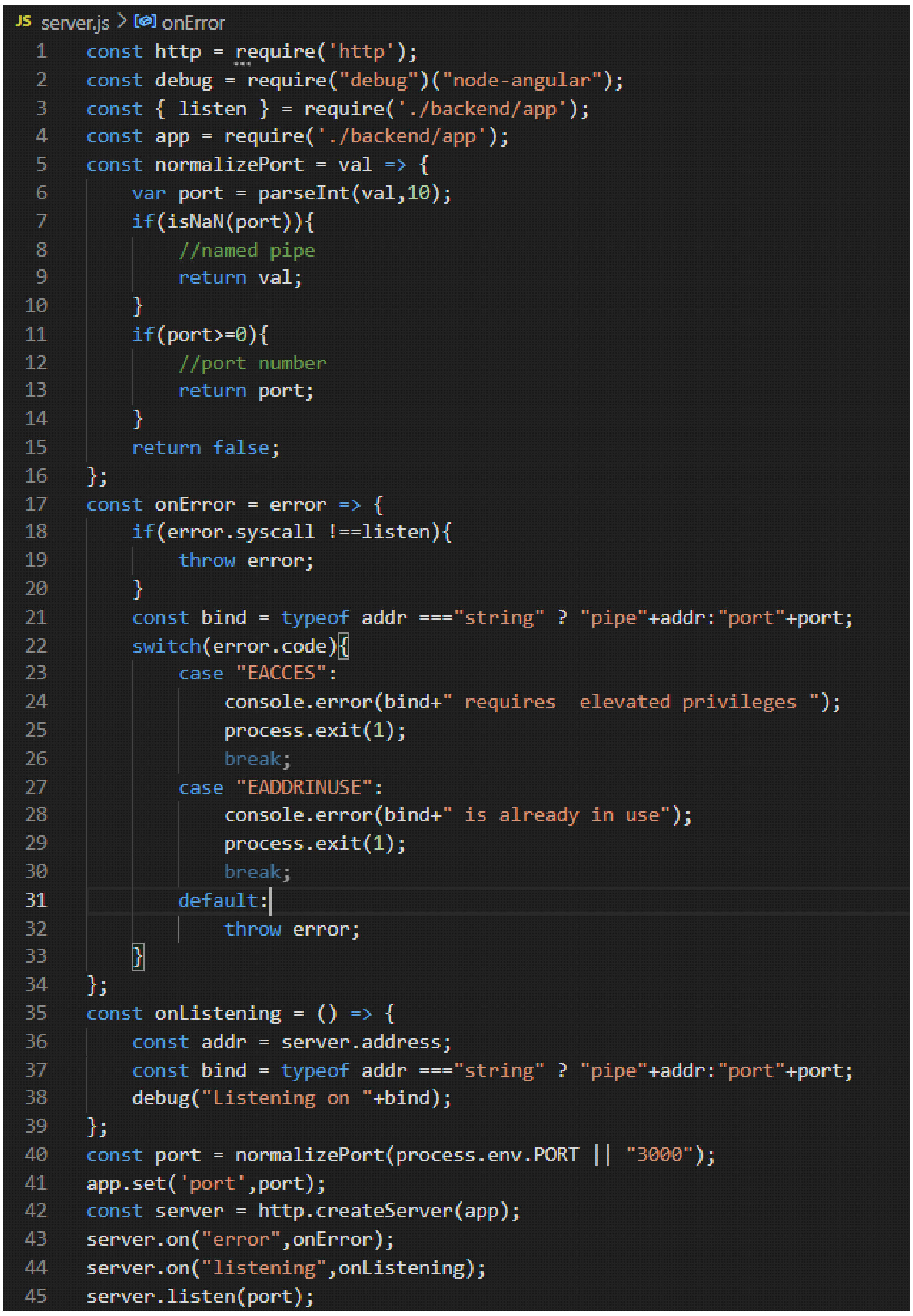
- To start the nodemon we need to declare in such a way in package.json and run the following command to start the node server "npm run start:server" then node application that we created will run.

Server.js

- We added some backend logic to our application under server.js , which is coded on node.js


server.js > onError

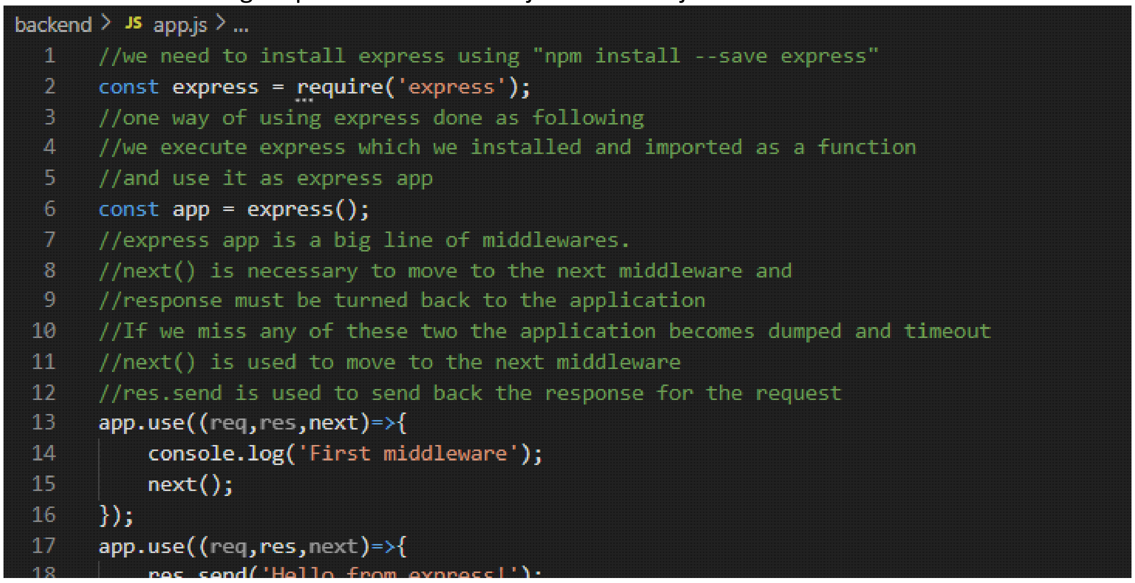
```javascript
const http = require('http');
const debug = require("debug")("node-angular");
```



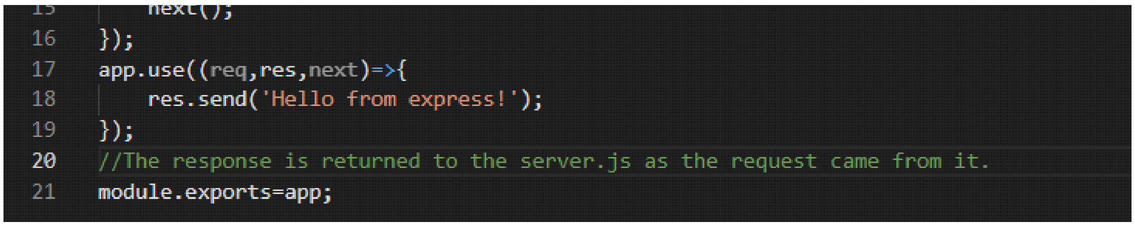
```
const http = require('http');
const debug = require("debug")("node-angular");
const { listen } = require('./backend/app');
const app = require('./backend/app');
const normalizePort = val => {
    var port = parseInt(val,10);
    if(isNaN(port)){
        //named pipe
        return val;
    }
    if(port>=0){
        //port number
        return port;
    }
    return false;
};
const onError = error => {
    if(error.syscall !==listen){
        throw error;
    }
    const bind = typeof addr ==="string" ? "pipe"+addr:"port"+port;
    switch(error.code){
        case "EACCES":
            console.error(bind+" requires  elevated privileges ");
            process.exit(1);
            break;
        case "EADDRINUSE":
            console.error(bind+" is already in use");
            process.exit(1);
            break;
        default:
            throw error;
    }
};
const onListening = () => {
    const addr = server.address;
    const bind = typeof addr ==="string" ? "pipe"+addr:"port"+port;
    debug("Listening on "+bind);
};
const port = normalizePort(process.env.PORT || "3000");
app.set('port',port);
const server = http.createServer(app);
server.on("error",onError);
server.on("listening",onListening);
server.listen(port);
```

App.js

- We are sending response to the server.js and server.js executes under localhost:3000



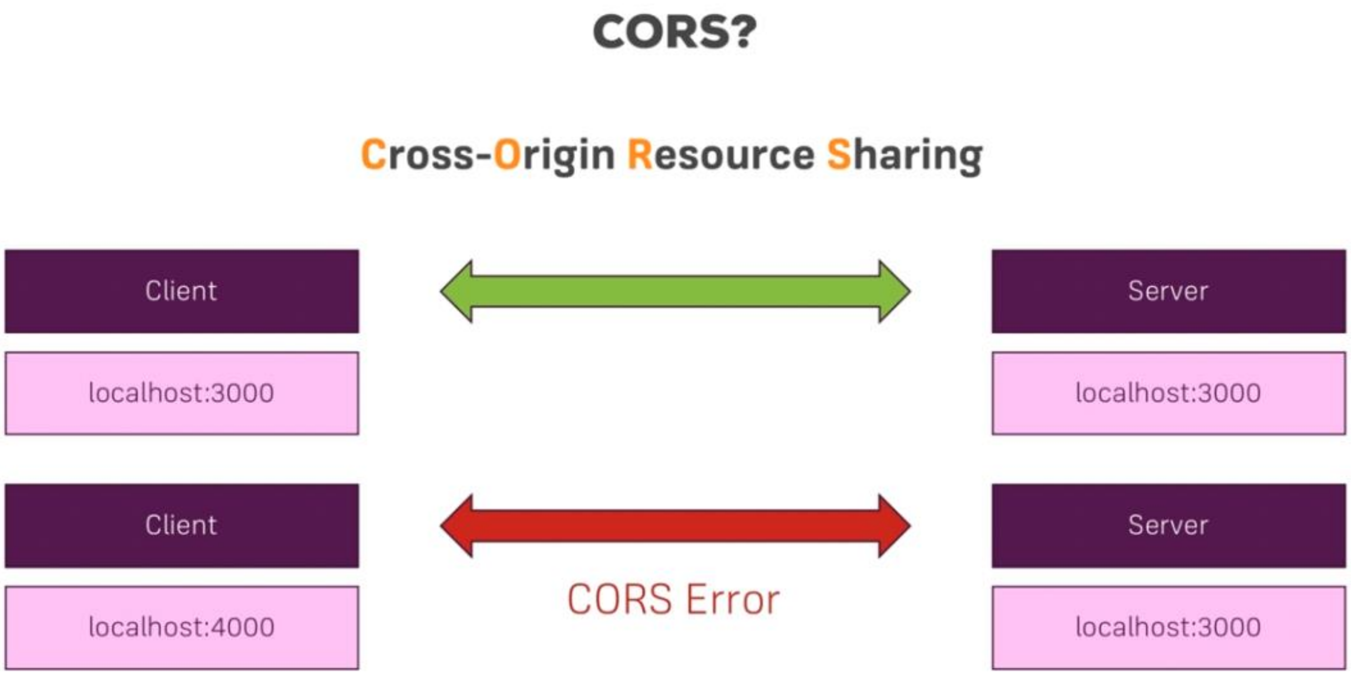
```
//we need to install express using "npm install --save express"
const express = require('express');
//one way of using express done as following
//we execute express which we installed and imported as a function
//and use it as express app
const app = express();
//express app is a big line of middlewares.
//next() is necessary to move to the next middleware and
//response must be turned back to the application
//If we miss any of these two the application becomes dumped and timeout
//next() is used to move to the next middleware
//res.send is used to send back the response for the request
app.use((req,res,next)=>{
    console.log('First middleware');
    next();
});
app.use((req,res,next)=>{
    res.send('Hello from express!');
```



```
        next();
});
app.use((req,res,next)=>{
    res.send('Hello from express!');
});
//The response is returned to the server.js as the request came from it.
module.exports=app;
```

Now go and check localhost:3000 after executing node server.js or npm run start:server

- Sometimes we would get CORS error , it mean Cross Origin Resource sharing error, it will generally occurred when we try to connect the backend to the frontend, or while connecting any two independent data sources through and API or when we try to connect different servers (localhost:4200 for angular & localhost:3000 for server).

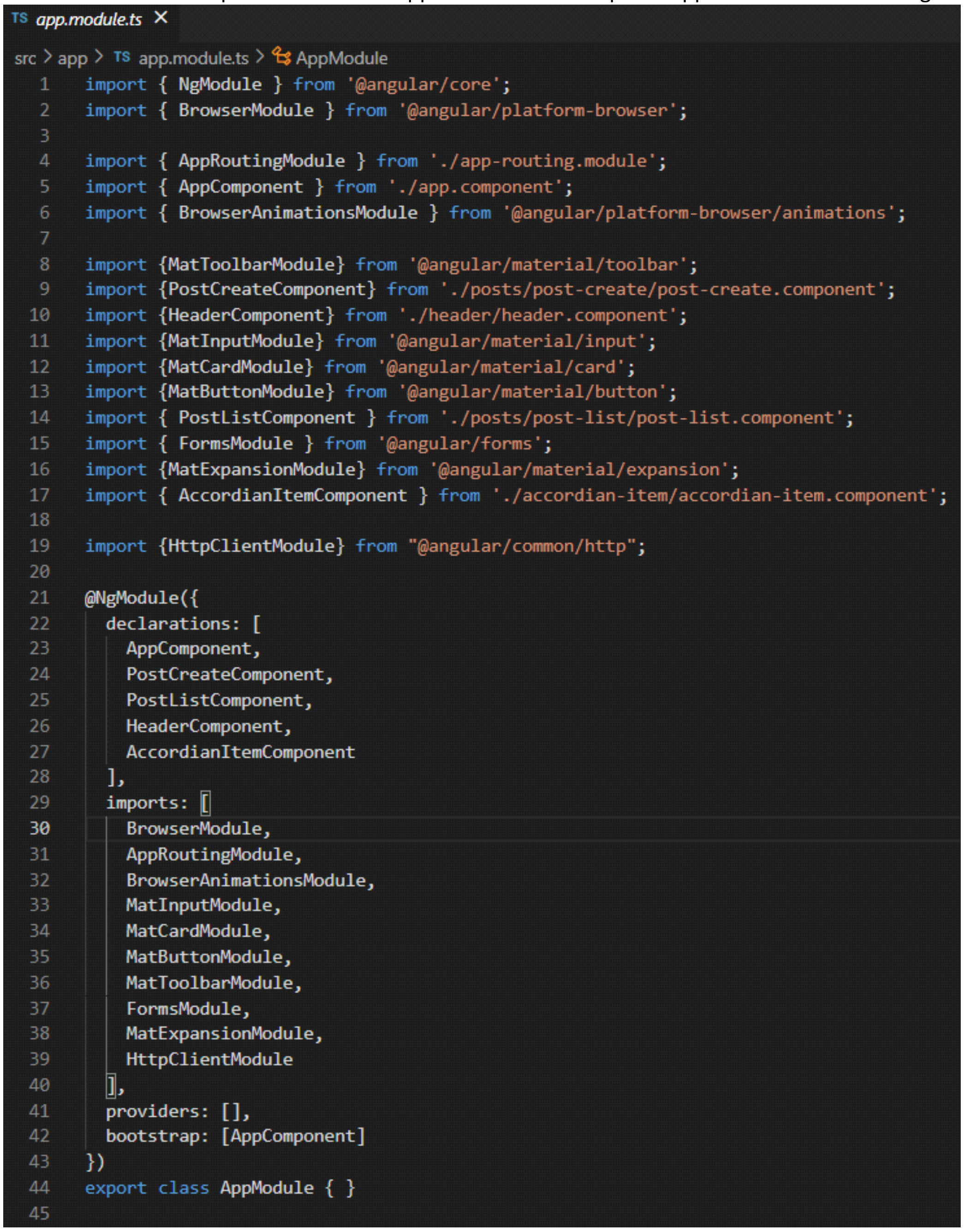


- To prevent this type of errors we need to make it of manually in the backend code as the following.

Connect API calls of Express -> Angular

15 April 2022 22:48

- To utilize the http modules on our application we must update app.module.ts as following



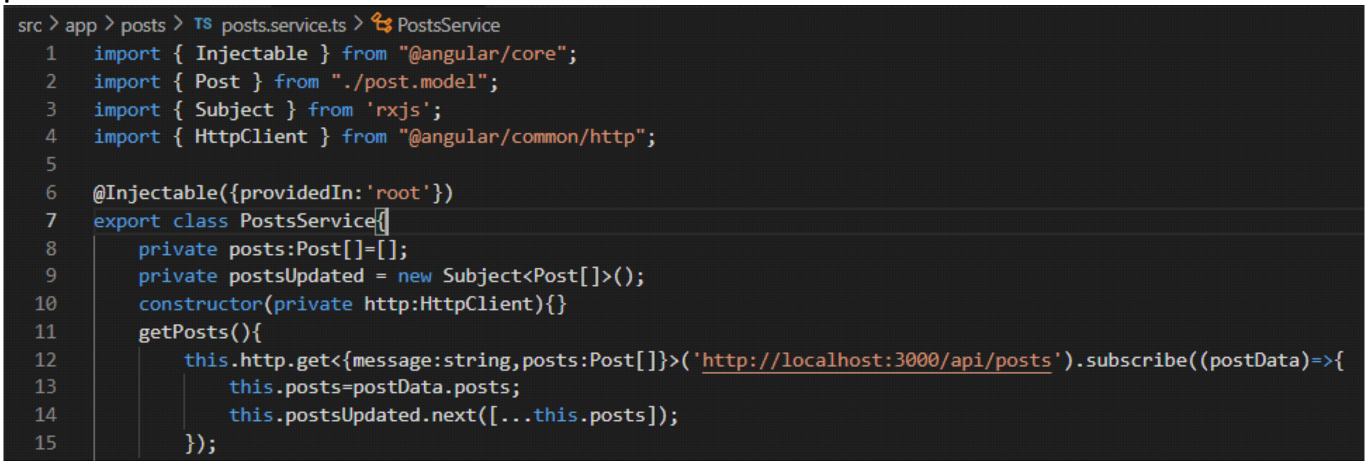
app.module.ts ×

src > app > app.module.ts > AppModule

```typescript
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 { }
```

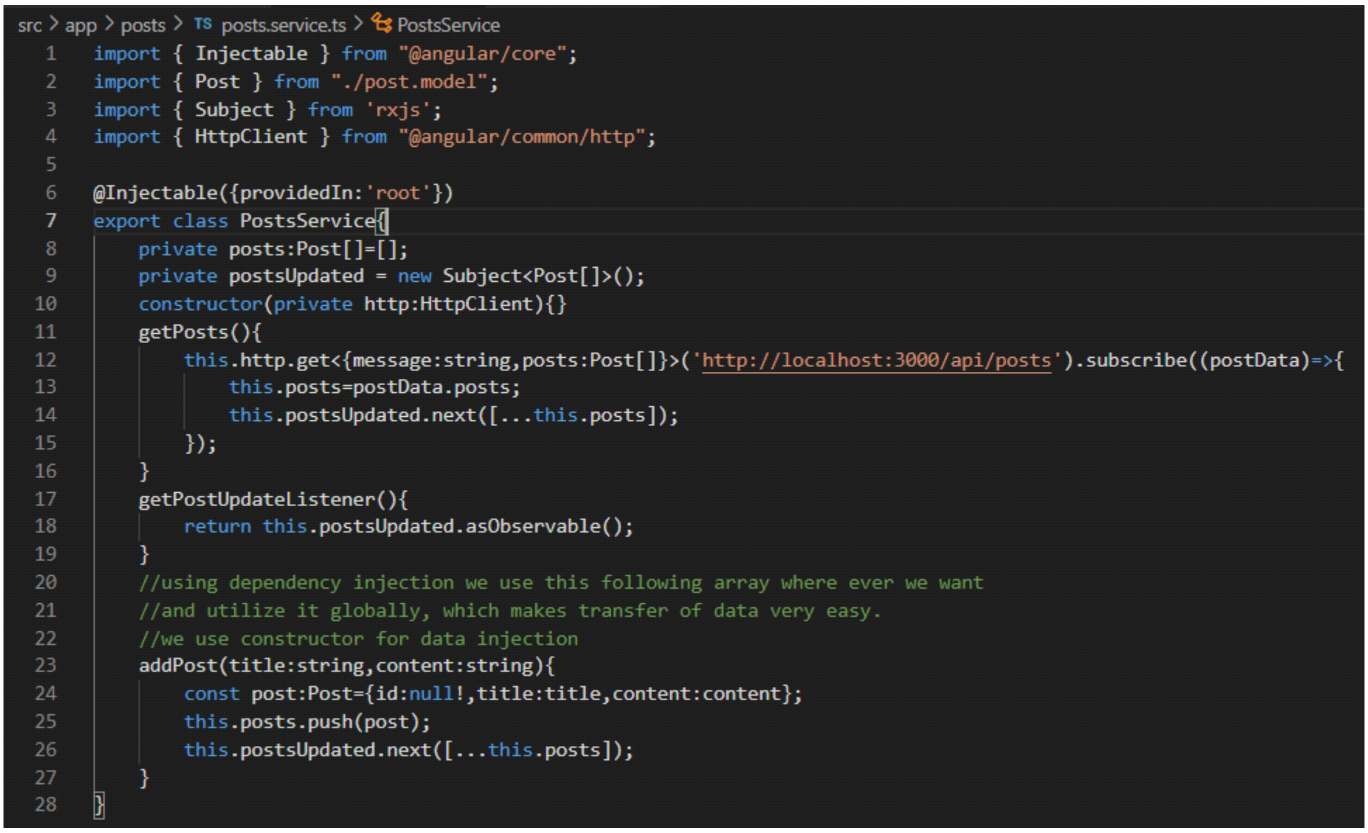
- After installing the whole HttpClient module we use what ever we need.

posts.service.ts



src > app > posts > posts.service.ts > PostsService

```typescript
import { Injectable } from "@angular/core";
import { Post } from "./post.model";
import { Subject } from 'rxjs';
import { HttpClient } from "@angular/common/http";

@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    private postsUpdated = new Subject<Post[]>();
    constructor(private http:HttpClient){}
    getPosts(){
        this.http.get<{message:string,posts:Post[]}>('http://localhost:3000/api/posts').subscribe((postData)=>{
            this.posts=postData.posts;
            this.postsUpdated.next([...this.posts]);
        });
```

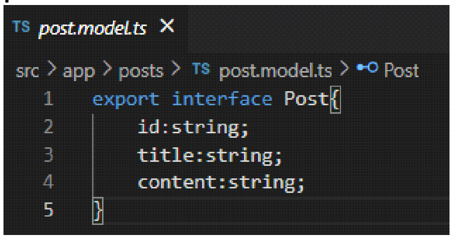


src > app > posts > TS posts.service.ts > PostsService

```typescript
import { Injectable } from "@angular/core";
import { Post } from "./post.model";
import { Subject } from 'rxjs';
import { HttpClient } from "@angular/common/http";

@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    private postsUpdated = new Subject<Post[]>();
    constructor(private http:HttpClient){}
    getPosts(){
        this.http.get<{message:string,posts:Post[]}>('http://localhost:3000/api/posts').subscribe((postData)=>{
            this.posts=postData.posts;
            this.postsUpdated.next([...this.posts]);
        });
    }
    getPostUpdateListener(){
        return this.postsUpdated.asObservable();
    }
    //using dependency injection we use this following array where ever we want
    //and utilize it globally, which makes transfer of data very easy.
    //we use constructor for data injection
    addPost(title:string,content:string){
        const post:Post={id:null!,title:title,content:content};
        this.posts.push(post);
        this.postsUpdated.next([...this.posts]);
    }
}
```

- Here in posts service typescript file, we use HttpClient and to initialize an instance for it , we define it in the constructor by creating a instance of HttpClient named as http. And using that we are using a get function of it to get the contents of given api and subscribing it and store the contents of it in postData. postData contains a json comprises of message and posts. So we are defining what type of input we are going to get in get function type declaration and storing that json inside the postData and further storing posts of postData inside the local private variable posts. And to utilize that variable as an observable , we are storing it in the postsUpdated Subject Observable. That observable is returned by accessing getPostUpdateListener. Remember we are updating both the posts array and postsUpdated observable. Observable destroy whenever component disappear.
- This getPosts and getPostUpdateListener will be accessed in post list component.

post.model.ts



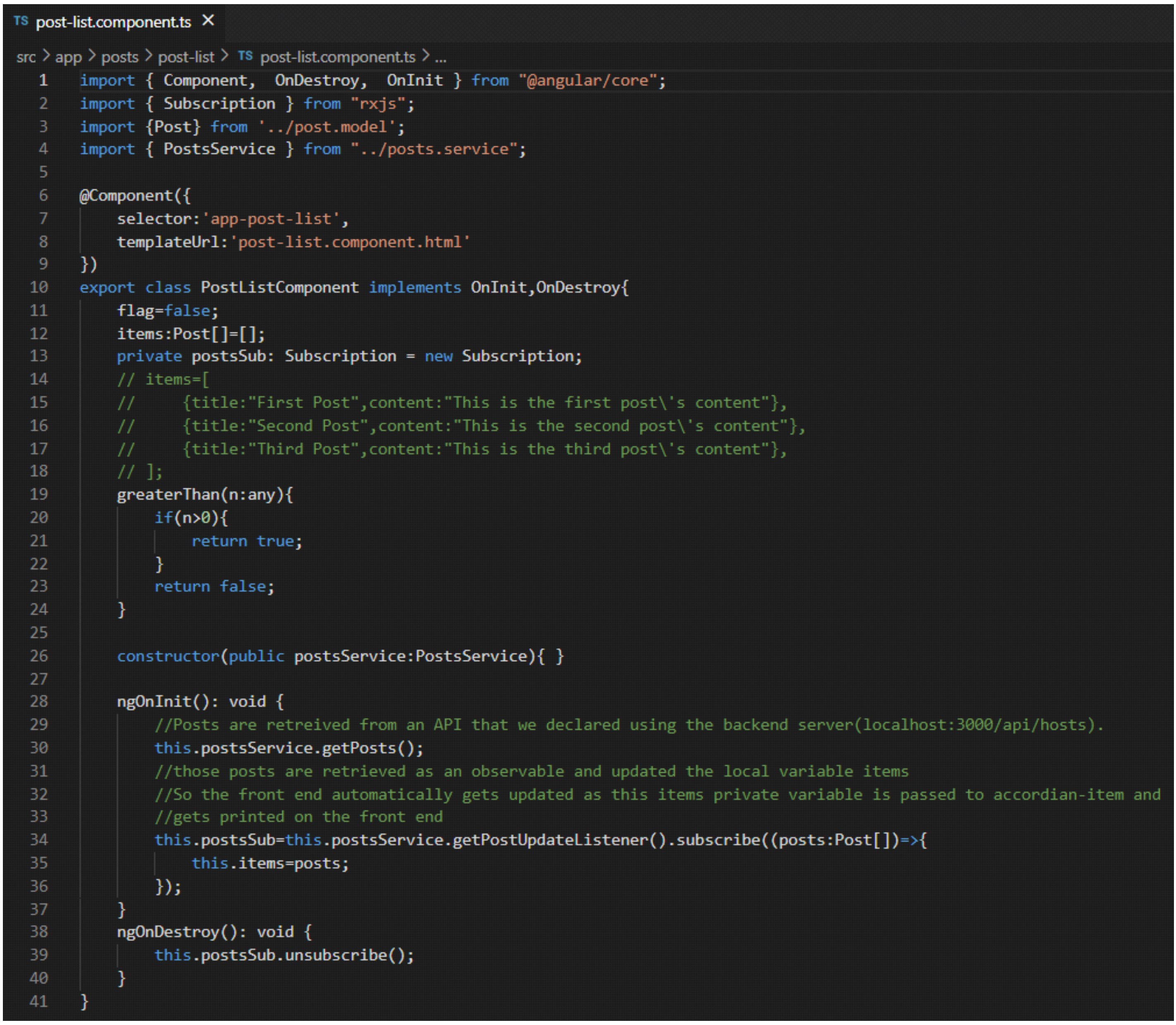
TS post.model.ts ✕

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

```typescript
export interface Post{
    id:string;
    title:string;
    content:string;
}
```

- The post interface structure is updated

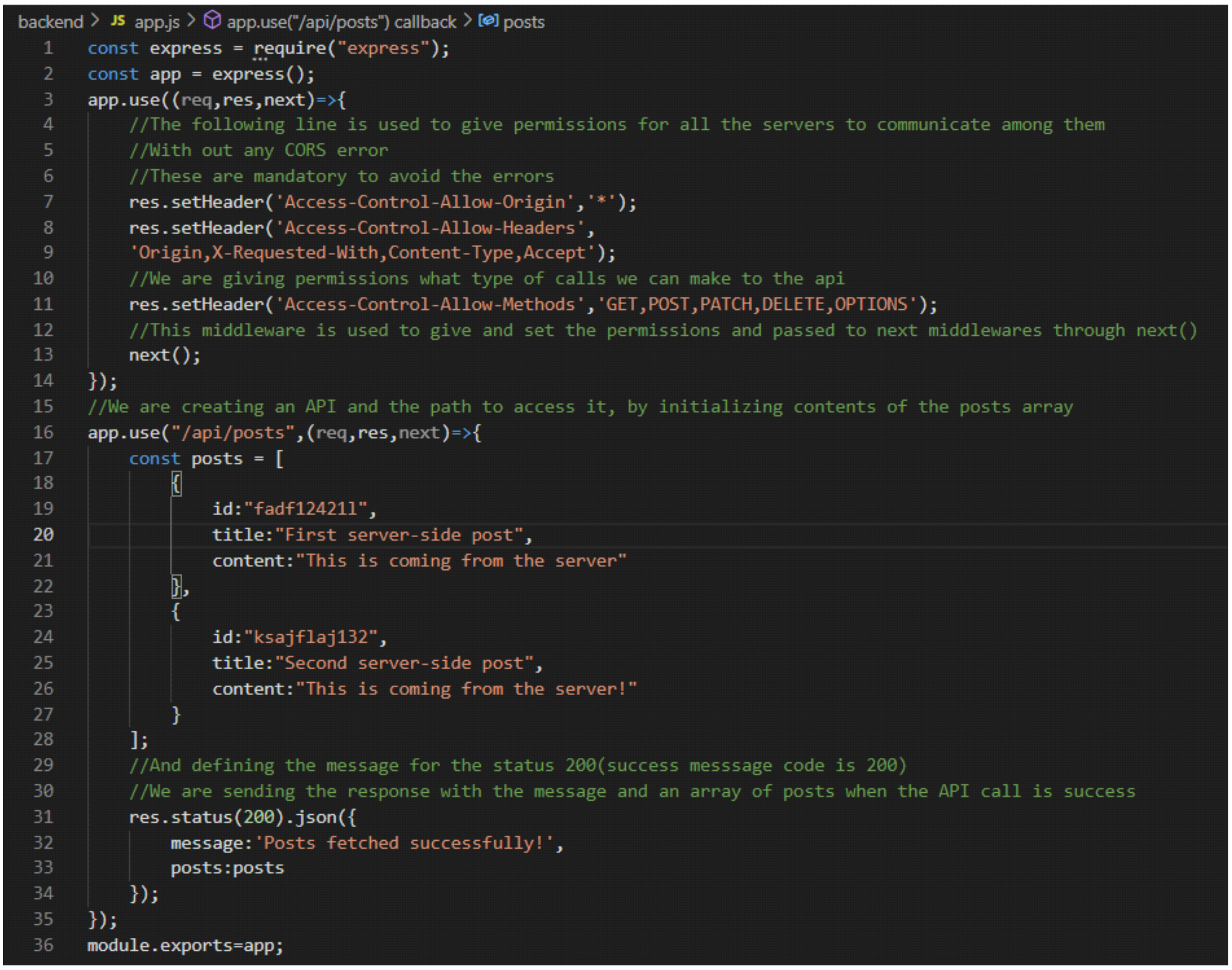
post-list.component.ts


post-list.component.ts
src > app > posts > post-list > 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();
    }
}
```

- Creating API through the backend and defining the path to access it and permissions of utilizing it like get , post requests. Here below we are using first middleware to give permissions and second middleware to create the api and send the response back for the request that is send by the posts.service.ts.
- Here posts.service.ts acts as the data passage between the post-create and post-list and API calls

app.js



backend > app.js > app.use("/api/posts") callback > posts

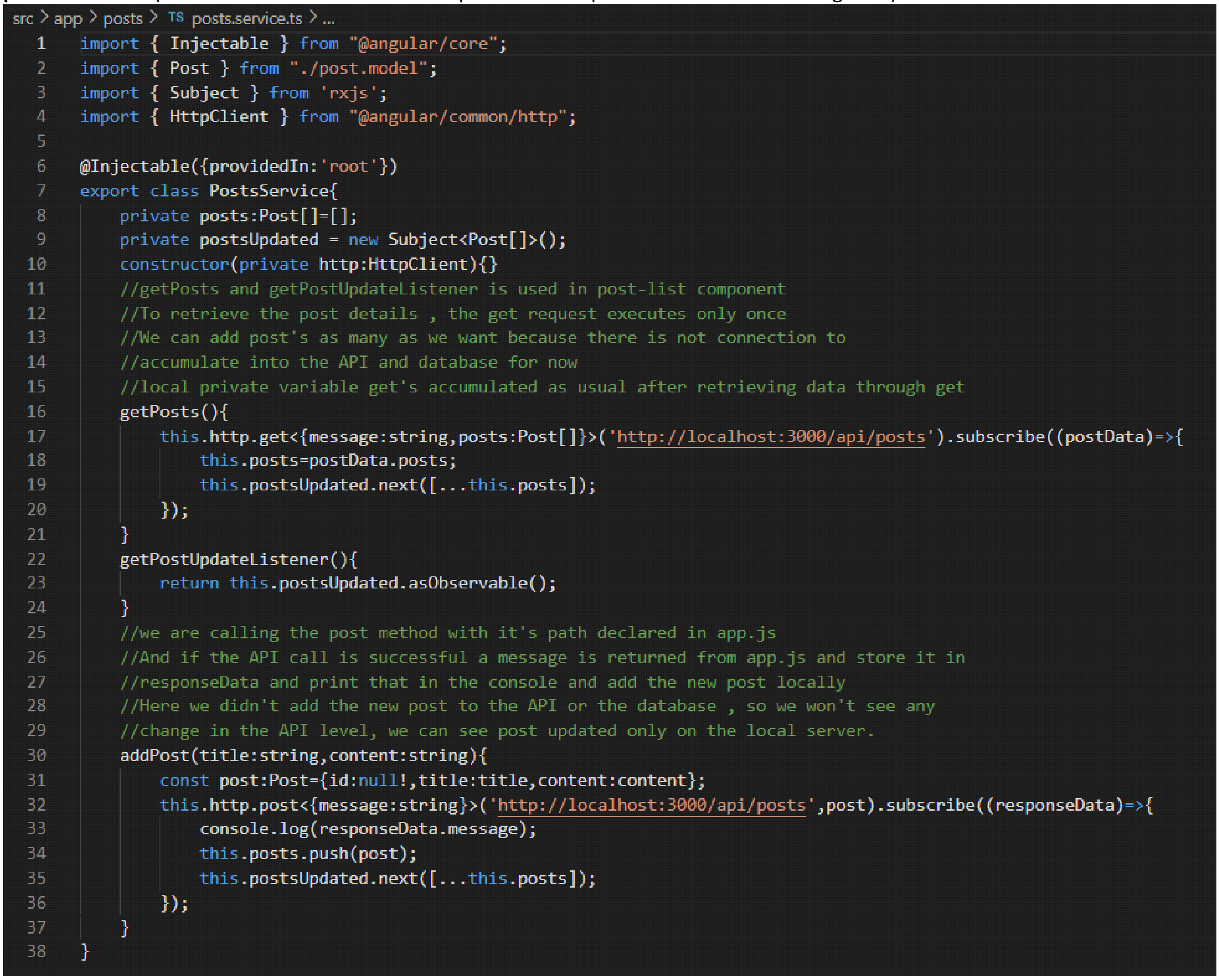
```
const express = require("express");
const app = express();
app.use((req,res,next)=>{
    //The following line is used to give permissions for all the servers to communicate among them
    //With out any CORS error
    //These are mandatory to avoid the errors
    res.setHeader('Access-Control-Allow-Origin','*');
    res.setHeader('Access-Control-Allow-Headers',
    'Origin,X-Requested-With,Content-Type,Accept');
    //We are giving permissions what type of calls we can make to the api
    res.setHeader('Access-Control-Allow-Methods','GET,POST,PATCH,DELETE,OPTIONS');
    //This middleware is used to give and set the permissions and passed to next middlewares through next()
    next();
});
//We are creating an API and the path to access it, by initializing contents of the posts array
app.use("/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!"
        }
    ];
    //And defining the message for the status 200(success messsage code is 200)
    //We are sending the response with the message and an array of posts when the API call is success
    res.status(200).json({
        message:'Posts fetched successfully!',
        posts:posts
    });
});
module.exports=app;
```

Post operation from the front end

16 April 2022 00:04

- Generally, we send data from the front end to the API and then route to the database using post request , as now we are not including the database inside the application. We are just consoling the output.
- When we are posing the data using the post method we need to parse the data that we send to the json format to store it into the database . So instead of doing it manually and for safety purpose we use a body-parser to do that operation.
- Install the body-parser using npm "npm install --save body-parser"
- Start angular server and node express server to see the output's and establish a connection between two servers.
- For the post operation to be included , three files must be modified app.js , post-create.component.ts and posts.service.ts

posts.service.ts (establish a connection between post-create & post-list & database through API)

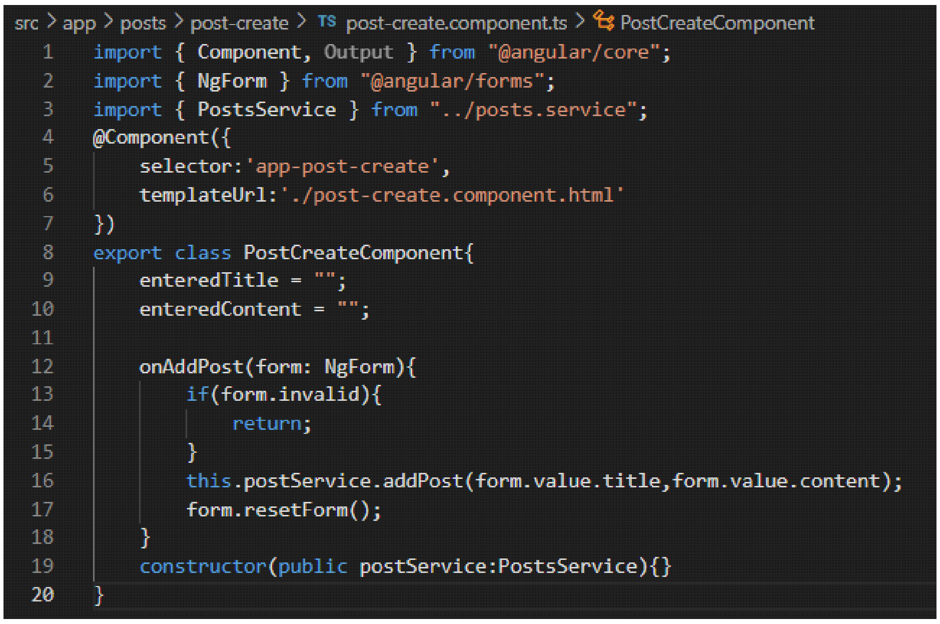


src > app > posts > TS posts.service.ts > ...

```typescript
import { Injectable } from "@angular/core";
import { Post } from "./post.model";
import { Subject } from 'rxjs';
import { HttpClient } from "@angular/common/http";

@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    private postsUpdated = new Subject<Post[]>();
    constructor(private http:HttpClient){}
    //getPosts and getPostUpdateListener is used in post-list component
    //To retrieve the post details , the get request executes only once
    //We can add post's as many as we want because there is not connection to
    //accumulate into the API and database for now
    //local private variable get's accumulated as usual after retrieving data through get
    getPosts(){
        this.http.get<{message:string,posts:Post[]}>('http://localhost:3000/api/posts').subscribe((postData)=>{
            this.posts=postData.posts;
            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
    //Here we didn't add the new post to the API or the database , so we won't see any
    //change in the API level, we can see post updated only on the local server.
    addPost(title:string,content:string){
        const post:Post={id:null!,title:title,content:content};
        this.http.post<{message:string}>('http://localhost:3000/api/posts',post).subscribe((responseData)=>{
            console.log(responseData.message);
            this.posts.push(post);
            this.postsUpdated.next([...this.posts]);
        });
    }
}
```

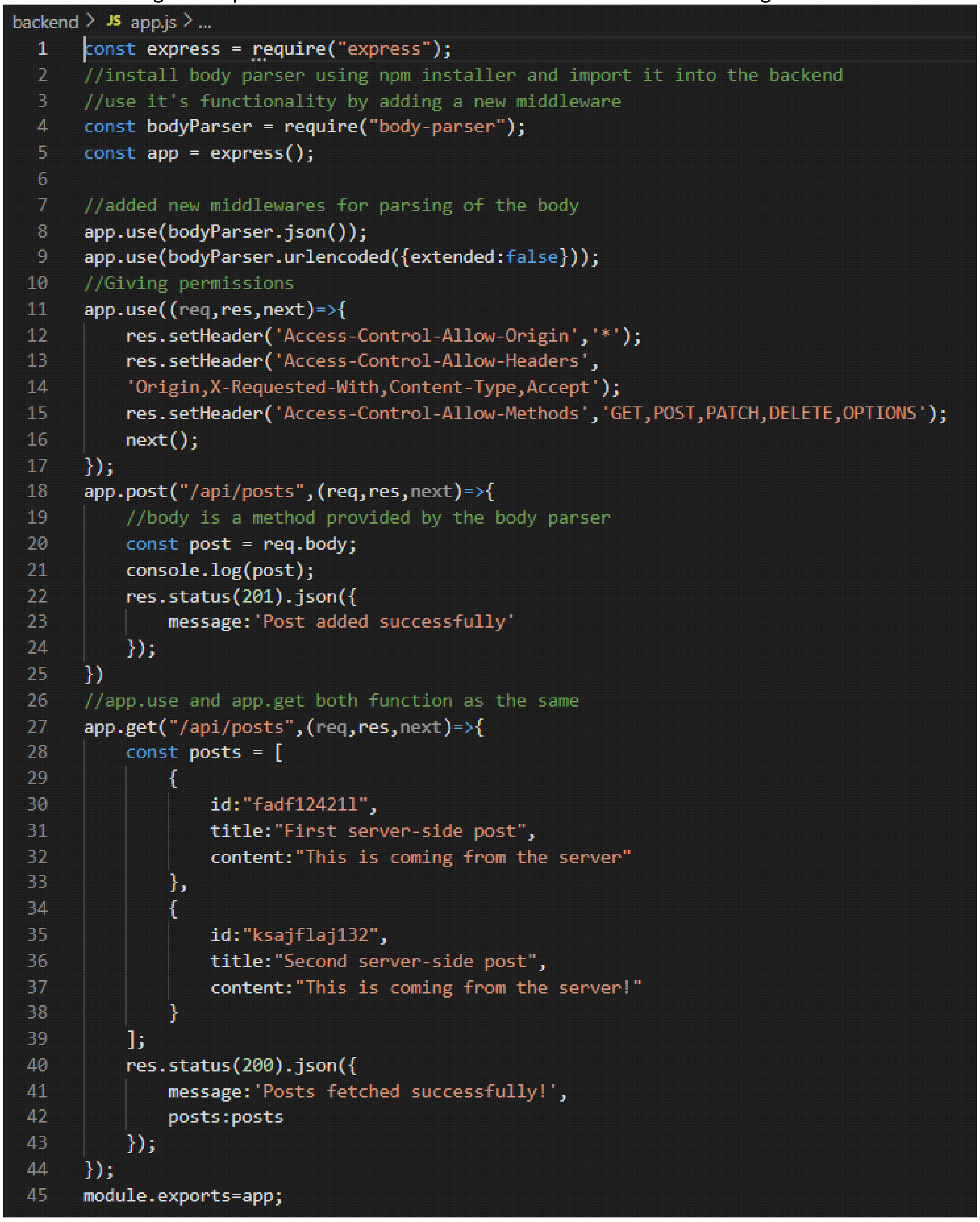
post-create.component.ts

- This post method is called when we execute post-create component's onAddPost then this function calls the addPost function of posts.service.ts then this calls the post method from this file.
- Post-create.component.ts remain's unchanged for now


src > app > posts > post-create > TS post-create.component.ts > PostCreateComponent
```
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){}
}
```

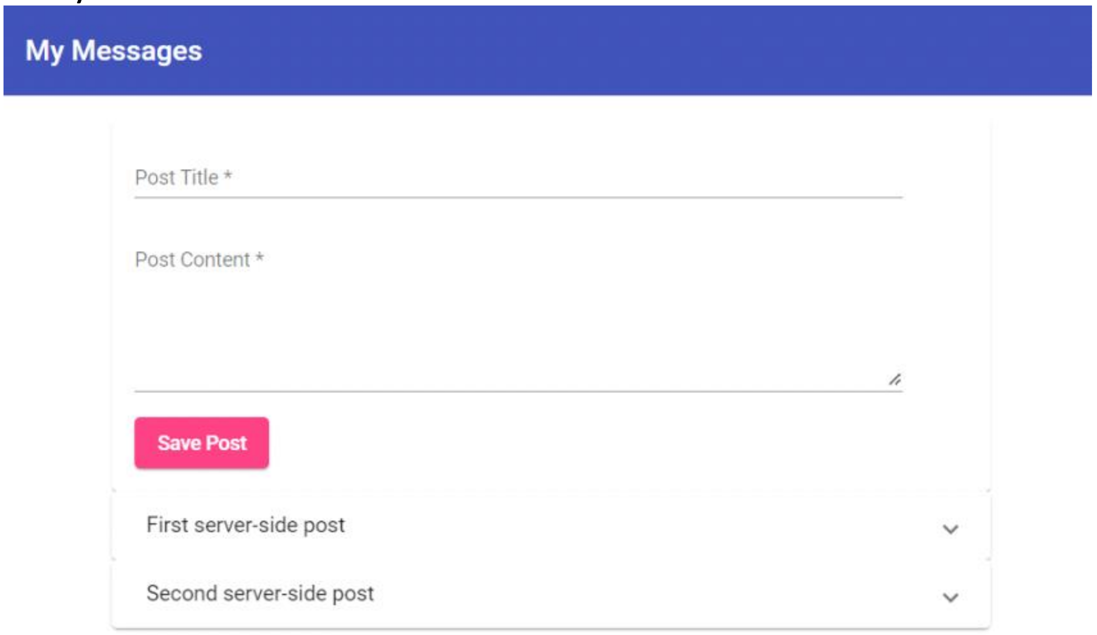
app.js

- Both the get and post methods and the API is declared here as following

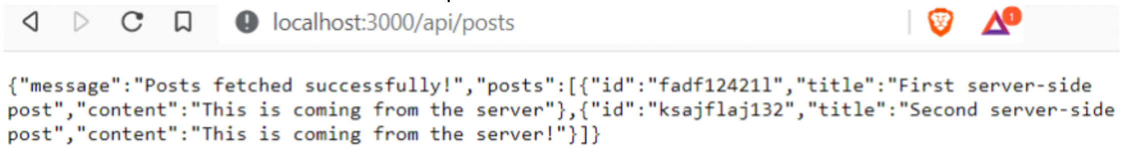

backend > JS app.js > ...
```
const express = require("express");
//install body parser using npm installer and import it into the backend
//use it's functionality by adding a new middleware
const bodyParser = require("body-parser");
const app = express();

//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();
});
app.post("/api/posts",(req,res,next)=>{
    //body is a method provided by the body parser
    const post = req.body;
    console.log(post);
    res.status(201).json({
        message:'Post added successfully'
    });
})
//app.use and app.get both function as the same
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!"
        }
    ];
    res.status(200).json({
        message:'Posts fetched successfully!',
        posts:posts
    });
});
module.exports=app;
```

- Remember to see the output you must execute both servers of front end and the backend.

Initially:



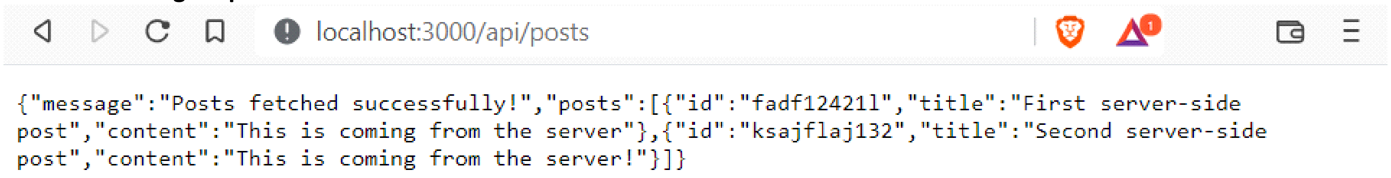
- Data that we declared in the API is present



```
{"message":"Posts fetched successfully!","posts":[{"id":"fadf12421l","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!"}]}
```

- Webpage will get updated when we save a post, but the data in the backend API and database will not be changed. In current example we didn't include database.

Backend didn't get updated



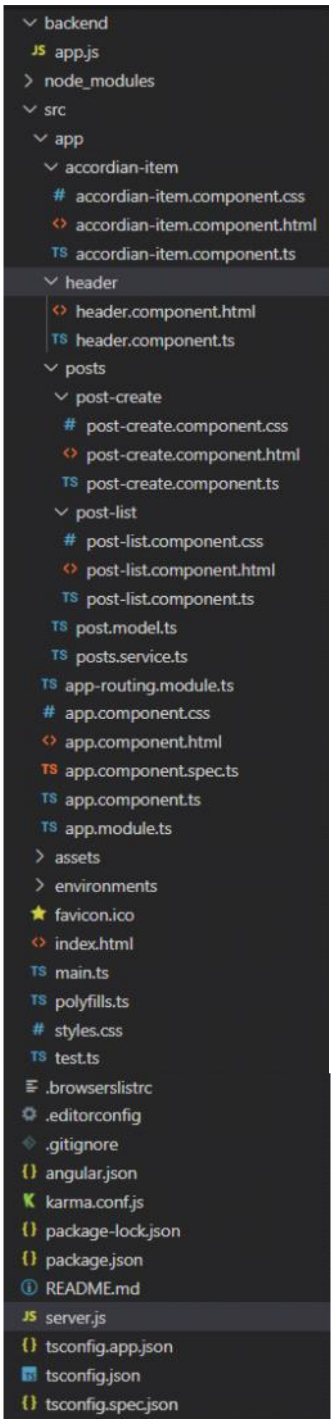
```
{"message":"Posts fetched successfully!","posts":[{"id":"fadf12421l","title":"First server-side
post","content":"This is coming from the server"},{"id":"ksajflaj132","title":"Second server-side
post","content":"This is coming from the server!"}]}
```

Complete Code (Angular + Express and Node)

16 April 2022 11:25

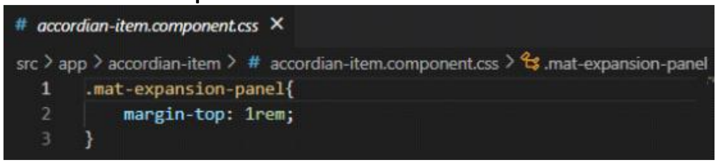
Complete File link: (https://drive.google.com/file/d/1V4leathsPcDHK1VOAiuGumP-zBiT13zN/view?usp=sharing)

Directory :



```
backend
  app.js
node_modules
src
  app
    accordian-item
      accordian-item.component.css
      accordian-item.component.html
      accordian-item.component.ts
    header
      header.component.html
      header.component.ts
    posts
      post-create
        post-create.component.css
        post-create.component.html
        post-create.component.ts
      post-list
        post-list.component.css
        post-list.component.html
        post-list.component.ts
      post.model.ts
      posts.service.ts
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
  assets
  environments
  favicon.ico
  index.html
  main.ts
  polyfills.ts
  styles.css
  test.ts
.browserslistrc
.editorconfig
.gitignore
angular.json
karma.conf.js
package-lock.json
package.json
README.md
server.js
tsconfig.app.json
tsconfig.json
tsconfig.spec.json
```

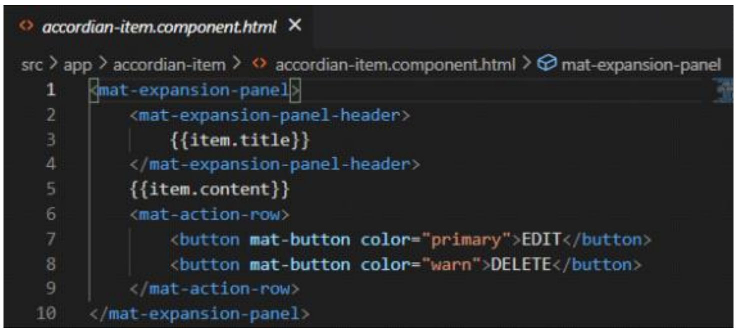
accordian-item.component.css

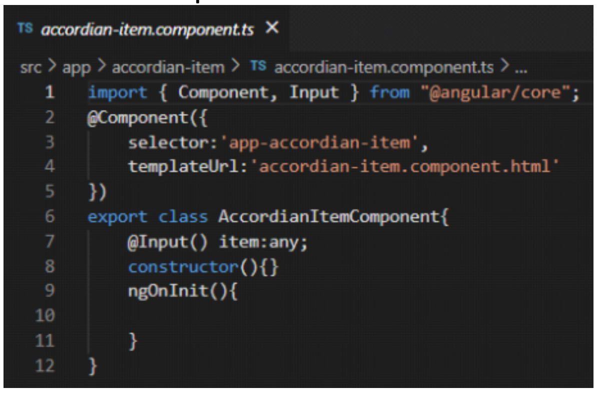


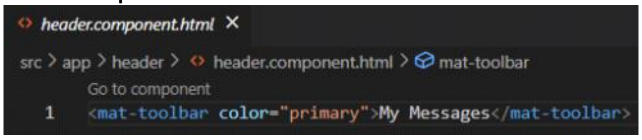
accordian-item.component.css ×
src > app > accordian-item > # accordian-item.component.css > .mat-expansion-panel

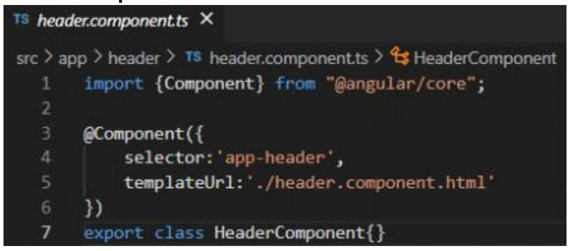
```css
.mat-expansion-panel{
    margin-top: 1rem;
}
```

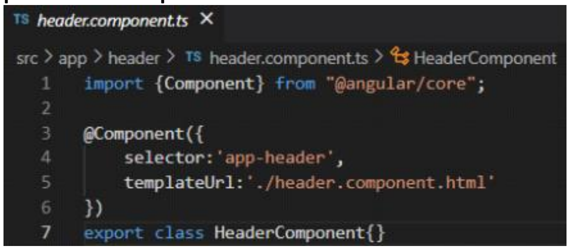
accordian-item.component.html


accordian-item.component.html ×
src > app > accordian-item > accordian-item.component.html > mat-expansion-panel
```html
<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>
        <button mat-button color="warn">DELETE</button>
    </mat-action-row>
</mat-expansion-panel>
```
accordian-item.component.ts

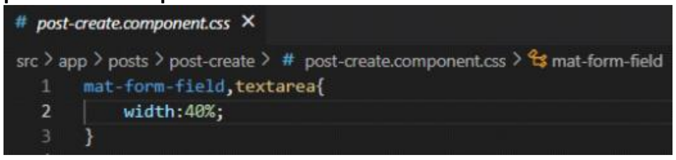

accordian-item.component.ts ×
src > app > accordian-item > accordian-item.component.ts > ...
```typescript
import { Component, Input } from "@angular/core";
@Component({
    selector:'app-accordian-item',
    templateUrl:'accordian-item.component.html'
})
export class AccordianItemComponent{
    @Input()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

    }
}
```
header.component.html

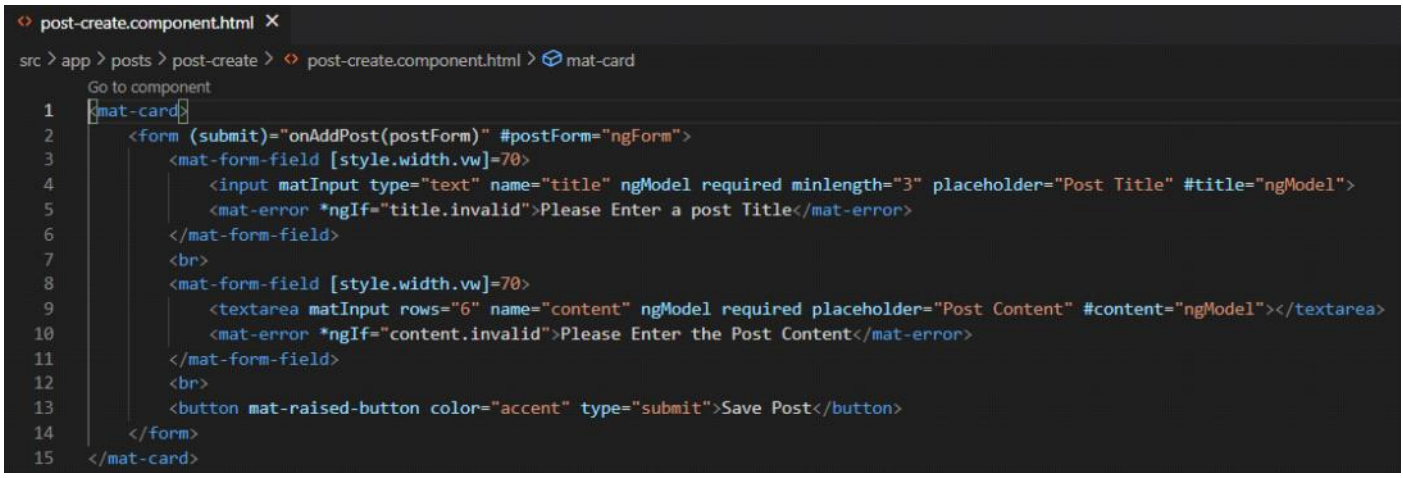

header.component.html ×
src > app > header > header.component.html > mat-toolbar
Go to component
```html
<mat-toolbar color="primary">My Messages</mat-toolbar>
```
header.component.ts


header.component.ts ×
src > app > header > header.component.ts > HeaderComponent
```typescript
import {Component} from "@angular/core";

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


header.component.ts ×
src > app > header > header.component.ts > HeaderComponent
```typescript
import {Component} from "@angular/core";

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

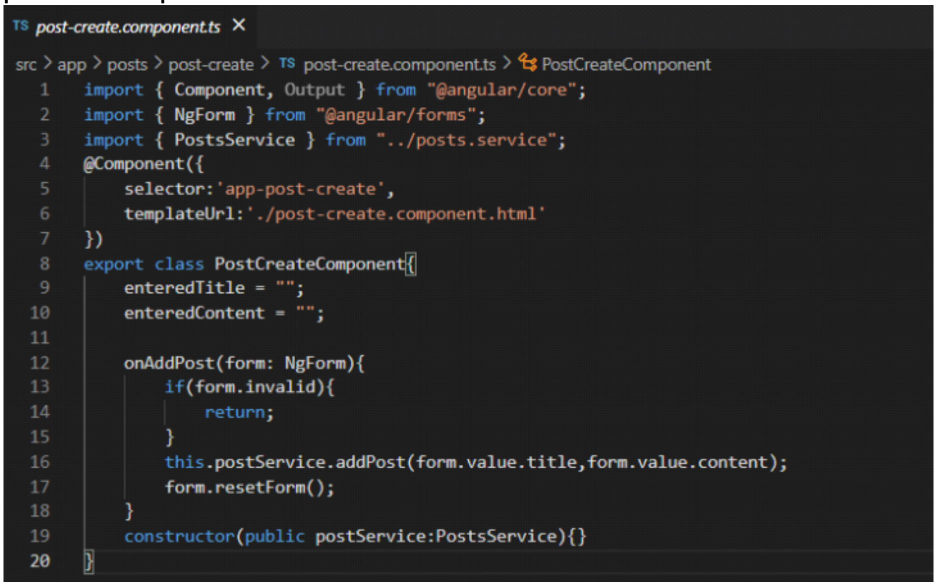
post-create.component.css


post-create.component.css ×
src > app > posts > post-create > post-create.component.css > mat-form-field
```css
mat-form-field,textarea{
    width:40%;
}
```
post-create.component.html


post-create.component.html
src > app > posts > post-create > post-create.component.html > mat-card
Go to component

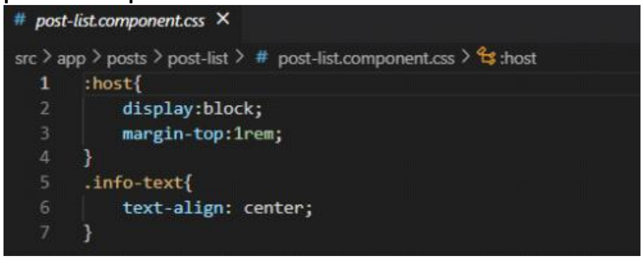
```html
<mat-card>
    <form (submit)="onAddPost(postForm)" #postForm="ngForm">
        <mat-form-field [style.width.vw]=70>
            <input matInput type="text" name="title" ngModel required minlength="3" placeholder="Post Title" #title="ngModel">
            <mat-error *ngIf="title.invalid">Please 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
src > app > posts > post-create > post-create.component.ts > PostCreateComponent

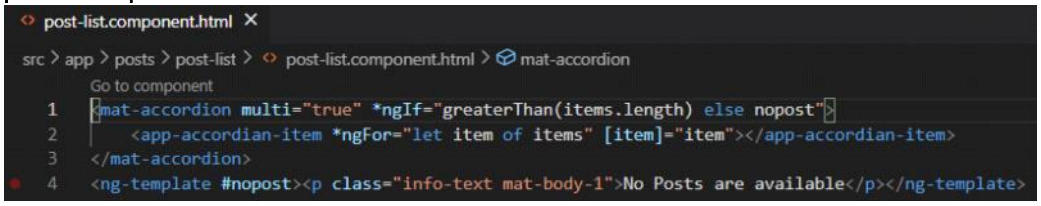
```typescript
import { Component, Output } from "@angular/core";
import { NgForm } from "@angular/forms";
import { PostsService } from "../posts.service";
@Component({
    selector:'app-post-create',
    templateUrl:'./post-create.component.html'
})
export class PostCreateComponent{
    enteredTitle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

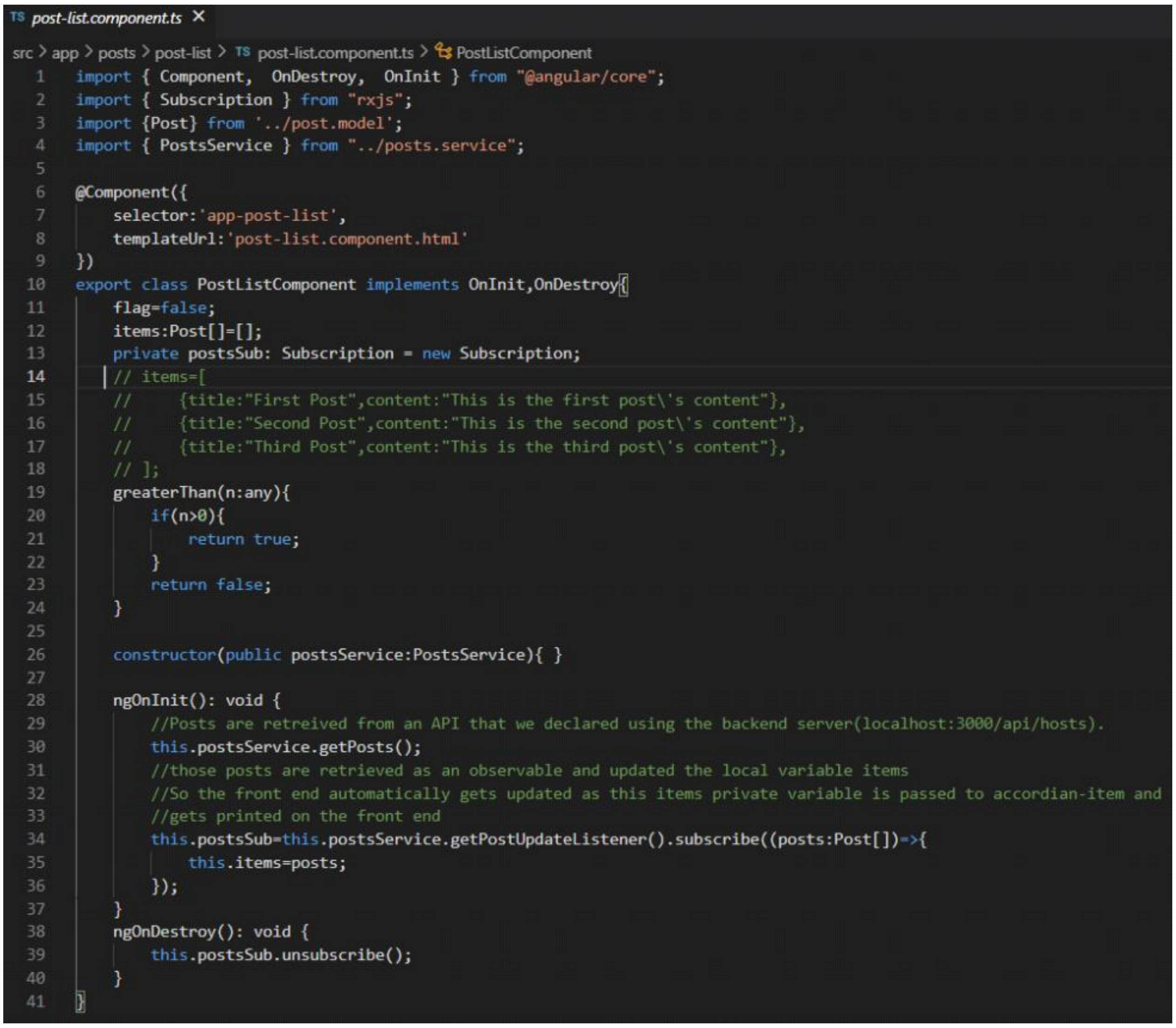
```css
:host{
    display:block;
    margin-top:1rem;
}
.info-text{
    text-align: center;
}
```

post-list.component.html


post-list.component.html
src > app > posts > post-list > post-list.component.html > mat-accordion
Go to component

```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

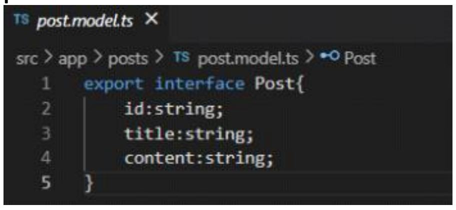


TS post-list.component.ts ×

src > app > posts > post-list > TS post-list.component.ts > PostListComponent

```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

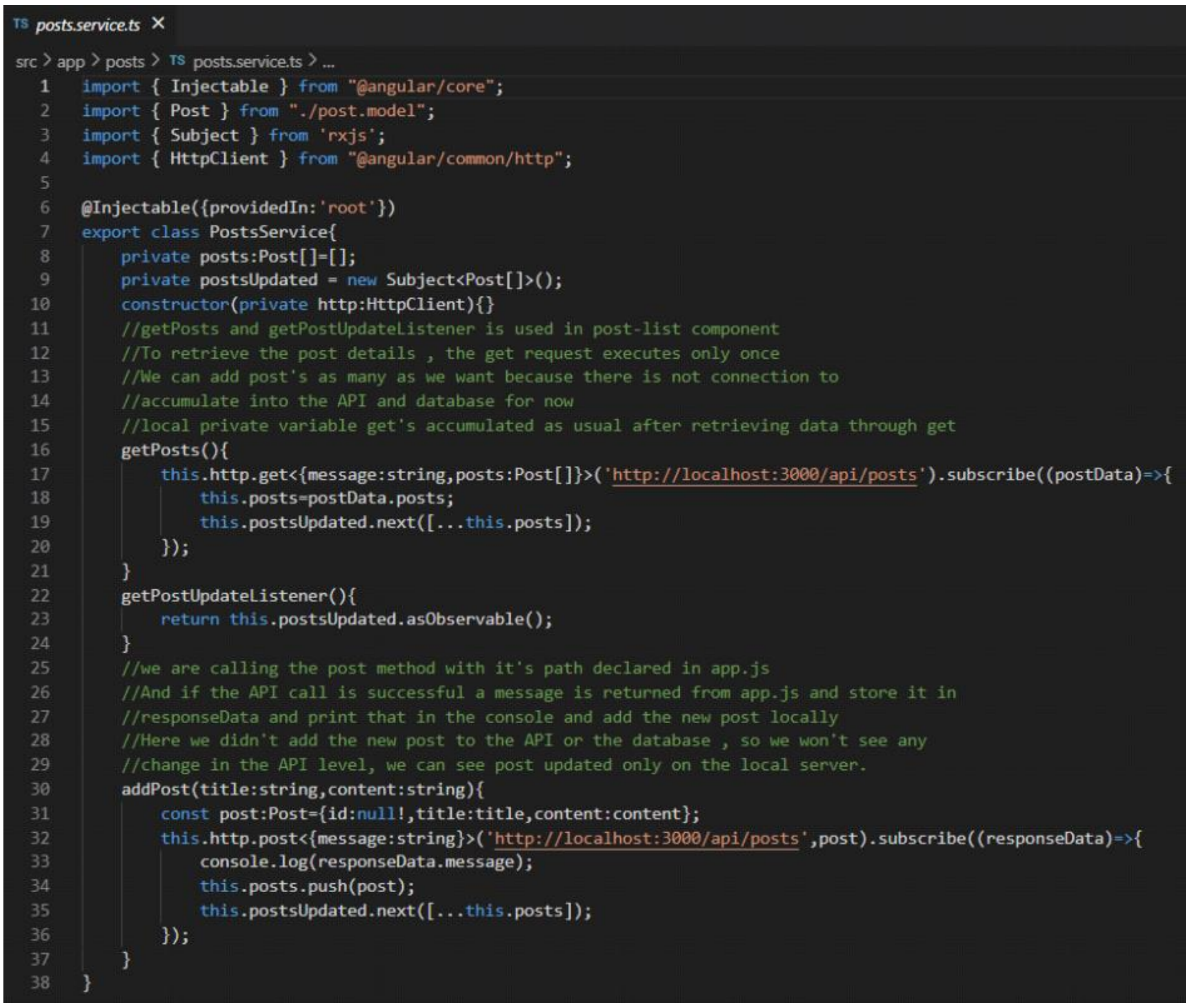


TS post.model.ts ×

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

```typescript
export interface Post{
    id:string;
    title:string;
    content:string;
}
```

posts.service.ts

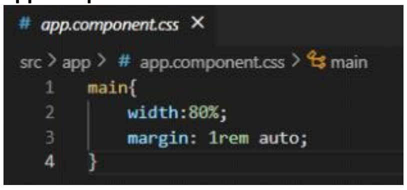


posts.service.ts ×

src > app > posts > posts.service.ts > ...

```typescript
import { Injectable } from "@angular/core";
import { Post } from "./post.model";
import { Subject } from 'rxjs';
import { HttpClient } from "@angular/common/http";

@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    private postsUpdated = new Subject<Post[]>();
    constructor(private http:HttpClient){}
    //getPosts and getPostUpdateListener is used in post-list component
    //To retrieve the post details , the get request executes only once
    //We can add post's as many as we want because there is not connection to
    //accumulate into the API and database for now
    //local private variable get's accumulated as usual after retrieving data through get
    getPosts(){
        this.http.get<{message:string,posts:Post[]}>('http://localhost:3000/api/posts').subscribe((postData)=>{
            this.posts=postData.posts;
            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
    //Here we didn't add the new post to the API or the database , so we won't see any
    //change in the API level, we can see post updated only on the local server.
    addPost(title:string,content:string){
        const post:Post={id:null!,title:title,content:content};
        this.http.post<{message:string}>('http://localhost:3000/api/posts',post).subscribe((responseData)=>{
            console.log(responseData.message);
            this.posts.push(post);
            this.postsUpdated.next([...this.posts]);
        });
    }
}
```

app.component.css

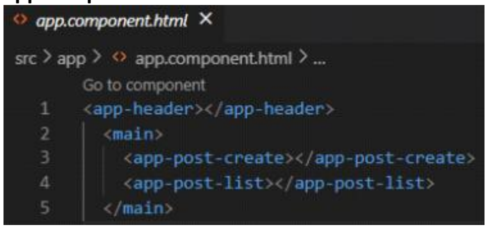


app.component.css ×

src > app > app.component.css > main

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

app.component.html



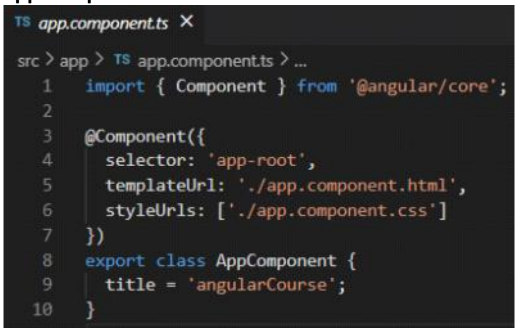
app.component.html ×

src > app > app.component.html > ...

Go to component

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

app.component.ts

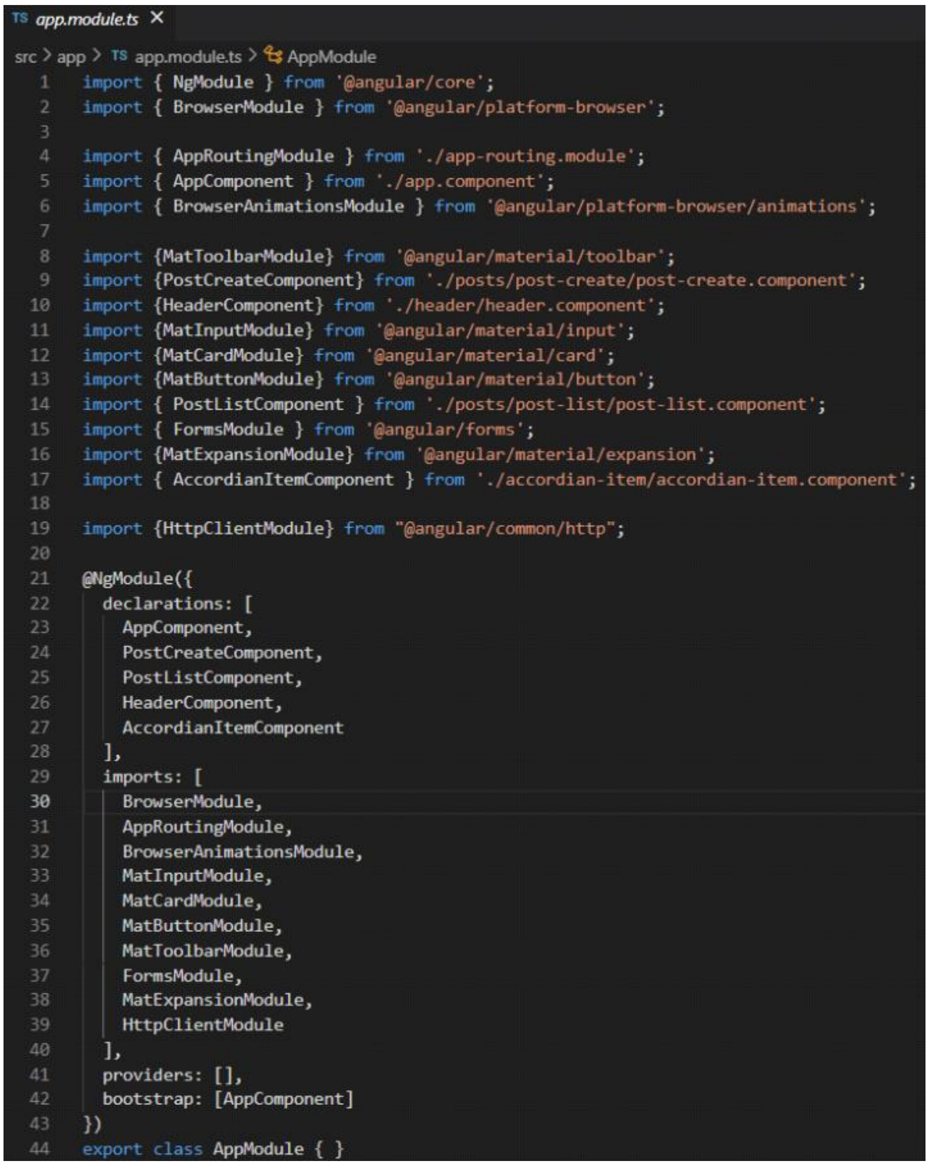


app.component.ts ×

src > app > app.component.ts > ...

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

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

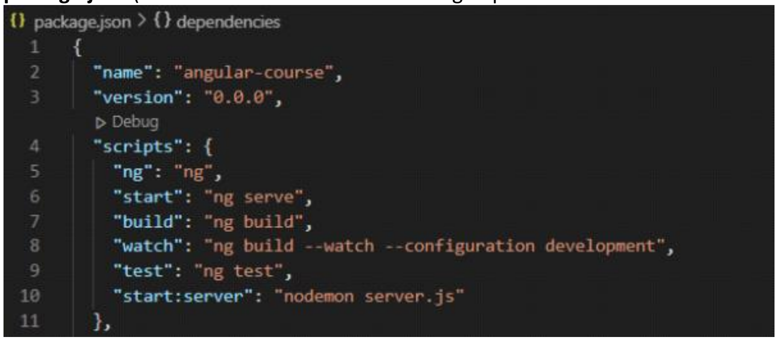
app.module.ts



app.module.ts
src > app > app.module.ts > AppModule

```typescript
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 { }
```

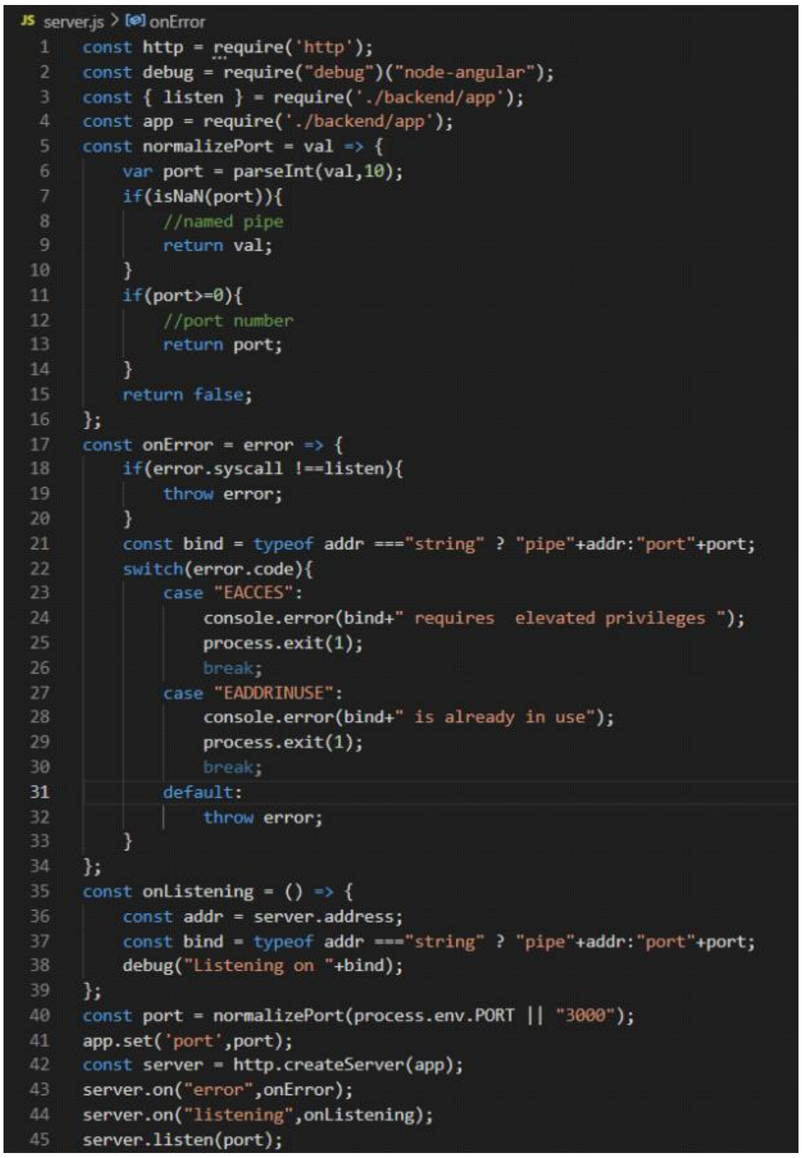
package.json (nodemon must be installed through npm and we need to declare it here)



package.json > dependencies

```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"
  },
```

server.js



JS server.js > onError

```javascript
const http = require('http');
const debug = require("debug")("node-angular");
const { listen } = require('./backend/app');
const app = require('./backend/app');
const normalizePort = val => {
    var port = parseInt(val,10);
    if(isNaN(port)){
        //named pipe
        return val;
    }
    if(port>=0){
        //port number
        return port;
    }
    return false;
};
const onError = error => {
    if(error.syscall !==listen){
        throw error;
    }
    const bind = typeof addr ==="string" ? "pipe"+addr:"port"+port;
    switch(error.code){
        case "EACCES":
            console.error(bind+" requires  elevated privileges ");
            process.exit(1);
            break;
        case "EADDRINUSE":
            console.error(bind+" is already in use");
            process.exit(1);
            break;
        default:
            throw error;
    }
};
const onListening = () => {
    const addr = server.address;
    const bind = typeof addr ==="string" ? "pipe"+addr:"port"+port;
    debug("Listening on "+bind);
};
const port = normalizePort(process.env.PORT || "3000");
app.set('port',port);
const server = http.createServer(app);
server.on("error",onError);
server.on("listening",onListening);
server.listen(port);
```