|  |  |  |
| --- | --- | --- |
| **Setting up MEAN with Angular 2 and Angular CLI** | | |
|  | | |
| MEAN simply stands for MongoDB, ExpressJS, Angular and NodeJS, and is a concept that came about a few years ago with the growing popularity of all those technologies. | | |
| To explain the different parts, | | |
|  | | |
| MongoDB | | usually acts as the database for your application, in case you need to persist data. It's where we store rec |
| ExpressJS | | is a web framework for nodejs, usually used as a backend for web apps in the MEAN stack. |
| Angular | | is usually the client side MVC web framework. In this case, we will be using Angular 2.\*. |
| NodeJS | | powers express, and will be the layer our server run on. |
|  | |  |
| **Steps for Setup** | | |
|  | | |
|  | Prerequisites | |
|  | We will need the Angular CLI. | |
|  | $ npm install -g angular-cli | |
|  |  | |
|  | Creating the Angular App | |
|  | It will create the Angular App. | |
|  | $ ng new mean-app | |
|  | To serve the app, simply run. | |
|  | $ ng serve | |
|  |  | |
|  | Adding Express | |
|  | Angular CLI comes with a command ng build, which bundles your angular app into a dist folder, or a folder that you may specify in the angular-cli.json file. | |
|  | Install express and body-parser as dependecies. | |
|  | $ npm install --save express body-parser | |
|  |  | |
|  | Then create a file server.js and a folder server in the root of our angular project. The server.js file will have the server code, that will point to the server folder, where the rest of the server implementation is. | |
|  | **server.js** | |
|  |  | |
|  | // Get dependencies  const express = require('express');  const path = require('path');  const http = require('http');  const bodyParser = require('body-parser');  // Get our API routes  const api = require('./server/routes/api');  const app = express();  // Parsers for POST data  app.use(bodyParser.json());  app.use(bodyParser.urlencoded({ extended: false }));  // Point static path to dist  app.use(express.static(path.join(\_\_dirname, 'dist')));  // Set our api routes  app.use('/api', api);  // Catch all other routes and return the index file  app.get('\*', (req, res) => {  res.sendFile(path.join(\_\_dirname, 'dist/index.html'));  });  /\*\*  \* Get port from environment and store in Express.  \*/  const port = process.env.PORT || '3000';  app.set('port', port);  /\*\*  \* Create HTTP server.  \*/  const server = http.createServer(app);  /\*\*  \* Listen on provided port, on all network interfaces.  \*/  server.listen(port, () => console.log(`API running on localhost:${port}`)); | |
|  | The above code sets up a simple express app, with an /api route and all other routes are directed towards the dist/index.html page. This catch all route, denoted with \*, **MUST** come last after all other API routes have been define. | |
|  | The /api route points to a file ./server/routes/api.js. Let's create this file. | |
|  | **server/routes/api.js** | |
|  |  | |
|  | const express = require('express');  const router = express.Router();  /\* GET api listing. \*/  router.get('/', (req, res) => {  res.send('Hello Radhe Krishna…');  });  module.exports = router; | |
|  |  | |
|  | One last thing before we run this. Since the catch all route is pointing to dist/index.html, we need to do a build of the angular app. | |
|  | $ ng build | |
|  |  | |
|  | This creates the dist folder with the angular 2 app built files. Now we can serve the app with express. | |
|  | $ node server.js | |
|  |  | |
|  | Going to http://localhost:3000 should load the app, and http://localhost:3000/api should show as below. | |
|  | **angular app (**[**http://localhost:3000**](http://localhost:3000)**)** | |
|  | **App works!!!** | |
|  |  | |
|  | **express api (**[**http://localhost:3000/api**](http://localhost:3000/api)**)** | |
|  | **Api works!!!** | |
|  |  | |
|  | **Server Data** | |
|  | We have the api set up. Let's quickly mock up some data for three route endpoints. We'll call the [jsonplaceholder](https://jsonplaceholder.typicode.com) mock api to respond with some data. | |
|  | First add [axios](https://github.com/mzabriskie/axios) for making http requests. | |
|  | $ npm install --save axios | |
|  |  | |
|  | Then, update the api.js file to have the following content. | |
|  | **server/routes/api.js** | |
|  |  | |
|  | const express = require('express');  const router = express.Router();  // declare axios for making http requests  const axios = require('axios');  const API = 'https://jsonplaceholder.typicode.com';  /\* GET api listing. \*/  router.get('/', (req, res) => {  res.send('api works');  });  // Get all posts  router.get('/posts', (req, res) => {  // Get posts from the mock api  // This should ideally be replaced with a service that connects to MongoDB  axios.get(`${API}/posts`)  .then(posts => {  res.status(200).json(posts.data);  })  .catch(error => {  res.status(500).send(error)  });  });  module.exports = router; | |
|  |  | |
|  | If we now stop the server and run it again, node server.js, we should see json data when we go to | |
|  | <http://localhost:3000/api/posts> | |
|  |  | |
|  | Angular Route, Component and Provider | |
|  | We'll add an angular component, then add a route that display this component's template. | |
|  | Add an angular component with the Angular CLI | |
|  | $ ng generate component posts | |
|  |  | |
|  | This adds a new folder in the src/app directory, called posts. If you've done a little getting started with Angular 2, the generated content of the component should be clear. We'll edit them when the time comes. | |
|  | The above command also imports the generated PostComponent in the src/app/app.module.ts file, and adds it to the declarations property of the @NgModule decorator. | |
|  | **src/app/app.module.ts** | |
|  |  | |
|  | // Imports commented out for brevity  import { PostsComponent } from './posts/posts.component';  @NgModule({  declarations: [  AppComponent,  PostsComponent // Posts Component injected here  ],  imports: [  BrowserModule,  FormsModule,  HttpModule  ],  providers: [],  bootstrap: [AppComponent]  })  export class AppModule { } | |
|  |  | |
|  | we'll add a posts route. There are a couple of recommended ways to add routes to your angular 2 apps, and that is out of scope for this guide. We'll use the simplest and most straight foreward. | |
|  | **src/app/app.module.ts** | |
|  |  | |
|  | // Imports commented out for brevity  import { RouterModule } from '@angular/router';  import { AppComponent } from './app.component';  import { PostsComponent } from './posts/posts.component';  // Define the routes  const ROUTES = [  {  path: '',  redirectTo: 'posts',  pathMatch: 'full'  },  {  path: 'posts',  component: PostsComponent  }  ];  @NgModule({  declarations: [  AppComponent,  PostsComponent  ],  imports: [  BrowserModule,  FormsModule,  HttpModule,  RouterModule.forRoot(ROUTES) // Add routes to the app  ],  providers: [],  bootstrap: [AppComponent]  })  export class AppModule { } | |
|  |  | |
|  | We are simply telling our router that whenever the root route / is visited, redirect to /posts. We then decalre the /posts route. | |
|  | One final thing to complete our routing is to first make sure that we have a <base href="/"> in the src/index.html head tag. And then add a router-outlet where the route should be rendered. We'll add this in the | |
|  | **src/app/app.component.html** | |
|  |  | |
|  | <h1>  {{title}}  </h1>  <router-outlet></router-outlet> | |
|  |  | |
|  | We need to do a build and serve the app, we could do | |
|  | $ ng build && node server.js | |
|  |  | |
|  | Or just create an npm script within the package.json. | |
|  | **package.json** | |
|  |  | |
|  | {  "name": "mean-app",  // meta data  "scripts": {  // Other scripts  "build": "ng build && node server.js"  },  "private": true,  "dependencies": {  ...  },  "devDependencies": {  ...  }  } | |
|  |  | |
|  | Then simply run. | |
|  | $ npm run build | |
|  |  | |
|  | Going to http://localhost:3000 should redirect you to http://locahost:3000/posts, based on the instructions we gave to our router. | |
|  | App works!!!  Posts works!!! | |
|  |  | |
|  | Connecting component to Express API | |
|  | Angular 2 recommend defining a provider or service to handle the http calls. So, we'll generate one with the Angular CLI. | |
|  | $ ng generate service posts | |
|  |  | |
|  | This creates a posts.service.ts in the src/app directory. We then need to add it in the providers section of our module declaration. | |
|  | **src/app/app.module.ts** | |
|  |  | |
|  | // Imports commented out for brevity  import { PostsService } from './posts.service';  // Routes  @NgModule({  declarations: [  AppComponent,  PostsComponent  ],  imports: [  BrowserModule,  FormsModule,  HttpModule,  RouterModule.forRoot(ROUTES)  ],  providers: [PostsService], // Add the posts service  bootstrap: [AppComponent]  })  export class AppModule { } | |
|  |  | |
|  | Then make the http call within the service to our express server. | |
|  | **src/app/posts.service.ts** | |
|  |  | |
|  | import { Injectable } from '@angular/core';  import { Http } from '@angular/http';  import 'rxjs/add/operator/map';  @Injectable()  export class PostsService {  constructor(private http: Http) { }  // Get all posts from the API  getAllPosts() {  return this.http.get('/api/posts')  .map(res => res.json());  }  } | |
|  |  | |
|  | Then import our service in the post component. | |
|  | **src/app/posts/posts.component.ts** | |
|  |  | |
|  | import { Component, OnInit } from '@angular/core';  import { PostsService } from '../posts.service';  @Component({  selector: 'app-posts',  templateUrl: './posts.component.html',  styleUrls: ['./posts.component.css']  })  export class PostsComponent implements OnInit {  // instantiate posts to an empty array  posts: any = [];  constructor(private postsService: PostsService) { }  ngOnInit() {  // Retrieve posts from the API  this.postsService.getAllPosts().subscribe(posts => {  this.posts = posts;  });  }  } | |
|  |  | |
|  | And finally, we'll just display the posts in the view. | |
|  | **src/app/posts/posts.component.html** | |
|  |  | |
|  | <div class="container">  <div class="row" \*ngFor="let post of posts">  <div class="card card-block">  <h4 class="card-title">{{ post.title }}</h4>  <p class="card-text">{{post.body}}</p>  <a href="#" class="card-link">Card link</a>  <a href="#" class="card-link">Another link</a>  </div>  </div>  </div> | |
|  | This is just a [bootstrap 4 card](https://v4-alpha.getbootstrap.com/components/card/), looping through our posts and binding the title and the body properties. I also added the bootstrap cdn to the src/index.html page in the head tag. | |
|  | <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-alpha.5/ css/bootstrap.c | |
|  |  | |
|  | Run the app. | |
|  | $ npm run build | |