next →

Angular 8 Tutorial

Angular community has released its latest version which is known as Angular 8. If you are familiar with previous version of Angular, it will not be difficult for you. You can easily upgrade your Angular CLI to version 8.

What is Angular 8?

Angular 8 is a client-side TypeScript based framework which is used to create dynamic web applications. It is very similar to its previous versions except having some extensive features.

Note: Dynamic web applications are simply dynamic websites i.e. *www.gmail.com*, *www.yahoo.com*, etc. which has tendency to change data/information with respect to 3 parameters:

- Time-to-time (eg. news update webs applications)
- Location-to-location (eg. Weather-report web applications)
- User-to-user (eg. Gmail, Facebook type applications)

You can see how to upgrade Angular CLI to version 8. Click Here

New Features of Angular 8

The Angular community has released its latest version Angular 8 with an impressive list of changes and improvements including the much awaited Ivy compiler as an opt-in feature.

These are the most prominent features of Angular 8:

- Angular 8 supports TypeScript 3.4
- Angular 8 supports Web Workers
- The new compiler for Angular 8 is Ivy Rendering Engine
- Angular 8 provides dynamic imports for lazy-loaded modules.
- Improvement of ngUpgrade

TypeScript 3.4

Angular 8 supports TypeScript 3.4 and it is required to run your Angular 8 project. So you have to upgrade your TypeScript version to 3.4.

Web workers class

JavaScript is single threaded, so it is common for more critical tasks like data calls to take place asynchronously. Web Workers facilitates you to run the CPU intensive computations in the background thread, freeing the main thread to update the user interface.

Web workers can also be helpful, if your application is unresponsive while processing data.

If you want to outsource such a calculation to a background, we must first create the web worker using the Angular CLI.

```
ng generate worker n-queens
```

More about Ivy and Bazel

Ivy is the new rendering engine and Bazel is the new build system. Both are ready for proper use with Angular 8. The preview of these two should be available shortly. Ivy is a new compiler/runtime of Angular and Angular 8 is a first release to offer a switch to opt-in into Ivy officially.

Ivy is supposed to be a by default rendering engine in Angular version 9.

Bazel provides one of the newest features of Angular 8 as a possibility to build your CLI application more quickly.

The main advantages of Bazel are:

- The incremental build and tests.
- It provides a chance to make your backends and frontends with a same tool.
- It has a possibility to have remote builds and cache on the build farm.

Dynamic imports for lazy-loaded modules

Angular 8 facilitates you to use standard dynamic import syntax instead of a custom string for lazy-loaded modules.

It means lazy-loaded import that looked like this:

```
{ path: '/student', loadChildren: './student/student.module#StudentModule' }
```

Will be looked like this:

```
{ path: `/student`, loadChildren: () => import(`./student/student.module`).then(s => s.StudentModule) }
```

Improved Angular CLI Workflow

The Angular CLI is continuously improving. Now, the **ng build, ng test and ng run** are equipped by 3rd-party libraries and tool. For example, AngularFire already makes use of these new capabilities with a **deploy** command.

Prerequisite for Angular 8 tutorial

- You must have installed Node.js version > 10. NPM will be updated also because it will be used by default. Here, I am using Node version 12.4.0
- You must have installed MongoDB on your system. You can see how to install MongoDB. Click here...

Workflow of Angular 8 Tutorial

Here, we will create two separate projects:

One for **front end (in Angular)** and one for **backend (in Node.js | Express | MongoDB)**. We will also create a backend API which will be used by frontend.

Here, we use the following technologies:

- Node: 12.4.0
- Angular CLI: 8.0.2
- NPM: v12.4.0
- MongoDB shell version v4.0.10
- MongoDB version v4.0.10
- Windows 10

Note: You can check your node, angular, npm, and mongoDB versions by using the following commands:

- To check **Node and Angular CLI version**, use **ng --version** command.
- To check **npm version**, use **node -v** command.
- To check **MongoDB version**, use **mongod --version** command.
- To check **MongoDB shell version**, use **mongo --version** command.

Create an Angular 8 project

Let's create an Angular 8 project by using the following command:

```
ng new angular8project
```

Here, **angular8project** is the name of the project.

Node.js command prompt

```
C:\Users\JavaTpoint>ng new angular8project
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE angular8project/angular.json (3497 bytes)
CREATE angular8project/package.json (1288 bytes)
CREATE angular8project/README.md (1032 bytes)
CREATE angular8project/tsconfig.json (438 bytes)
CREATE angular8project/tslint.json (1985 bytes)
CREATE angular8project/.editorconfig (246 bytes)
CREATE angular8project/.gitignore (629 bytes)
CREATE angular8project/browserslist (429 bytes)
CREATE angular8project/karma.conf.js (1027 bytes)
CREATE angular8project/tsconfig.app.json (210 bytes)
CREATE angular8project/tsconfig.spec.json (270 bytes)
CREATE angular8project/src/favicon.ico (5430 bytes)
CREATE angular8project/src/index.html (302 bytes)
CREATE angular8project/src/main.ts (372 bytes)
CREATE angular8project/src/polyfills.ts (2838 bytes)
CREATE angular8project/src/styles.css (80 bytes)
CREATE angular8project/src/test.ts (642 bytes)
CREATE angular8project/src/assets/.gitkeep (0 bytes)
CREATE angular8project/src/environments/environment.prod.ts (51 bytes)
CREATE angular8project/src/environments/environment.ts (662 bytes)
CREATE angular8project/src/app/app-routing.module.ts (245 bytes)
CREATE angular8project/src/app/app.module.ts (393 bytes)
CREATE angular8project/src/app/app.component.html (1152 bytes)
CREATE angular8project/src/app/app.component.spec.ts (1122 bytes)
CREATE angular8project/src/app/app.component.ts (219 bytes)
CREATE angular8project/src/app/app.component.css (0 bytes)
```

Node.js command prompt

```
CREATE angular8project/e2e/protractor.conf.js (810 bytes)
CREATE angular8project/e2e/tsconfig.json (214 bytes)
CREATE angular8project/e2e/src/app.e2e-spec.ts (644 bytes)
CREATE angular8project/e2e/src/app.po.ts (251 bytes)

> core-js@2.6.9 postinstall C:\Users\JavaTpoint\angular8project\node_modules\babel-runtime\node_modules\core-js
> node scripts/postinstall || echo "ignore"


> core-js@2.6.9 postinstall C:\Users\JavaTpoint\angular8project\node_modules\karma\node_modules\core-js
> node scripts/postinstall || echo "ignore"


> @angular/cli@8.0.3 postinstall C:\Users\JavaTpoint\angular8project\node_modules\@angular\cli
> node ./bin/postinstall/script.js

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

added 1011 packages from 1041 contributors and audited 19005 packages in 210.005s
found 0 vulnerabilities

'git' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\JavaTpoint>
```

Install Bootstrap 4 CSS framework

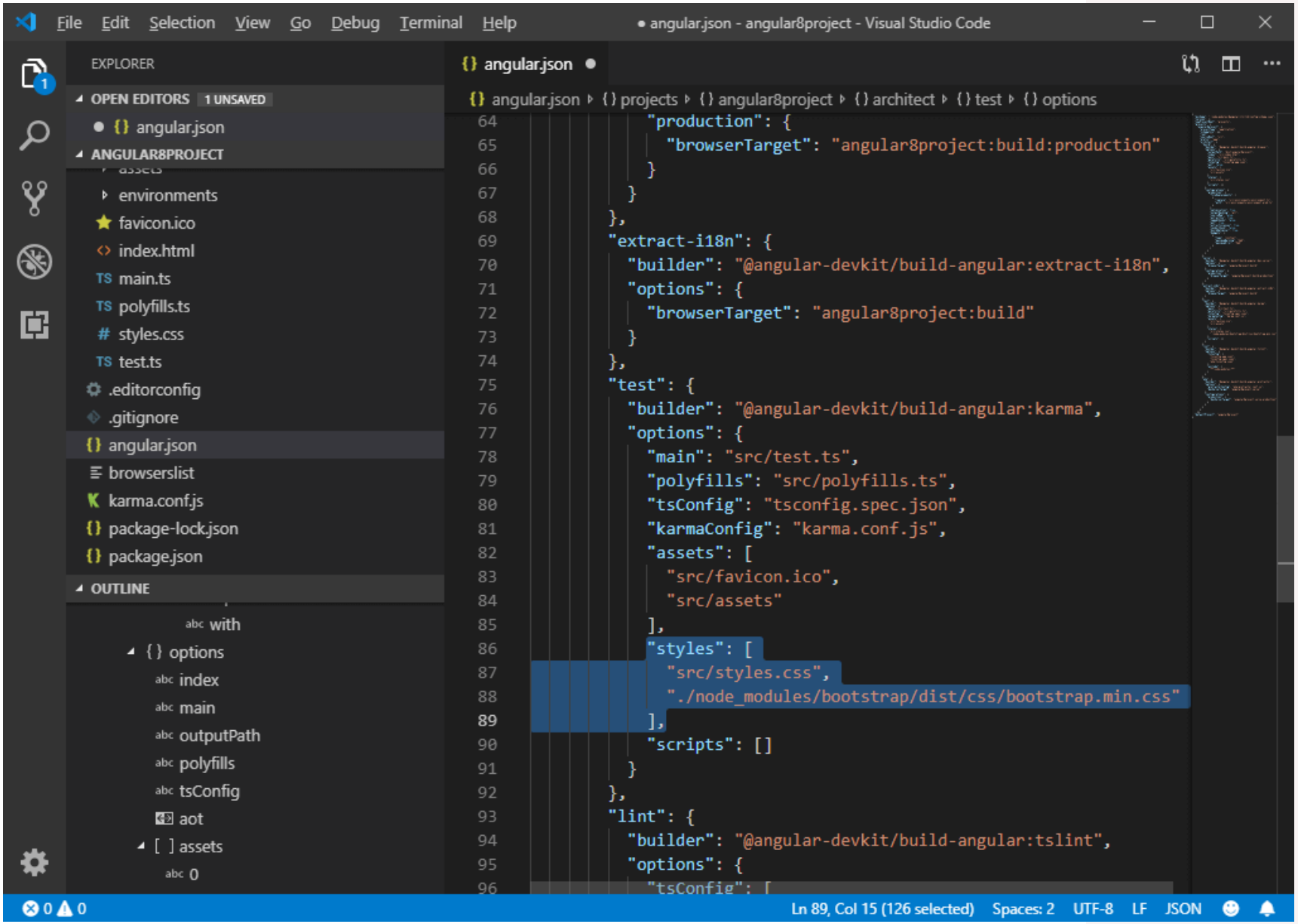
Use the following command to install bootstrap in your project.

```
npm install bootstrap --save
```

```
Select Node.js command prompt

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

added 1011 packages from 1041 contributors and audited 19005 packages in 210.005s
found 0 vulnerabilities

'git' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\JavaTpoint>cd angular8project

C:\Users\JavaTpoint\angular8project>code .

C:\Users\JavaTpoint\angular8project>npm install bootstrap --save
npm WARN bootstrap@4.3.1 requires a peer of jquery@1.9.1 - 3 but none is installed. You must install peer dependencies y
ourself.
npm WARN bootstrap@4.3.1 requires a peer of popper.js@^1.14.7 but none is installed. You must install peer dependencies
yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

+ bootstrap@4.3.1
added 1 package from 2 contributors and audited 19006 packages in 24.064s
found 0 vulnerabilities


C:\Users\JavaTpoint\angular8project>_
```

Now, add the following code inside the angular.json file.

```
"styles": [
  "src/styles.css",
  "./node_modules/bootstrap/dist/css/bootstrap.min.css"
],
```



The above CSS is used that we can use the bootstrap classes inside any file.

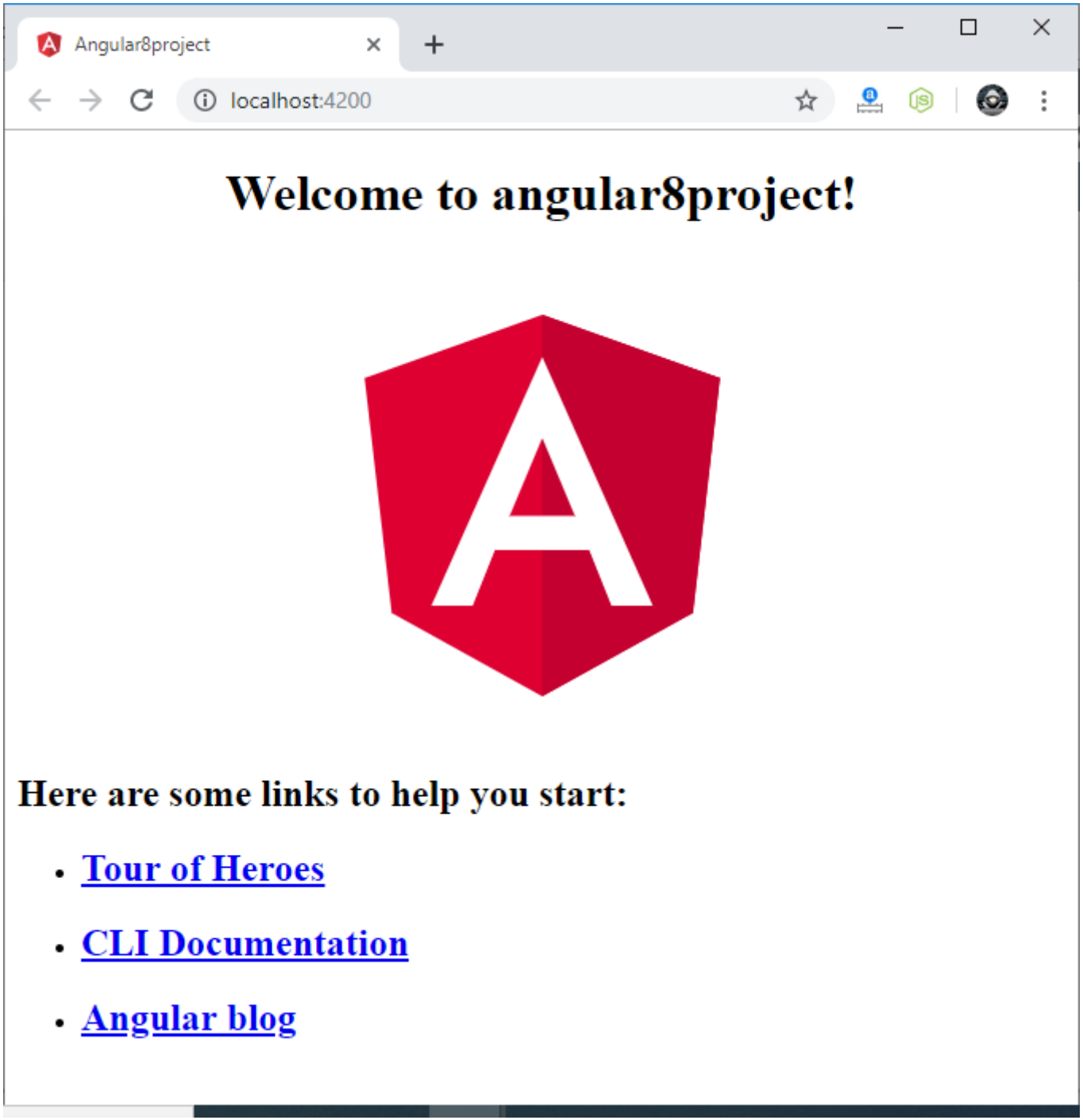
Start the Angular development server using the following command.

```
ng serve -o
```

```
Select ng serve -o
C:\Users\JavaTpoint\angular8project>npm install bootstrap --save
npm WARN bootstrap@4.3.1 requires a peer of jquery@1.9.1 - 3 but none is installed. You must install peer dependencies y
ourself.
npm WARN bootstrap@4.3.1 requires a peer of popper.js@^1.14.7 but none is installed. You must install peer dependencies
yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

+ bootstrap@4.3.1
added 1 package from 2 contributors and audited 19006 packages in 24.064s
found 0 vulnerabilities


C:\Users\JavaTpoint\angular8project>ng serve -o

Date: 2019-06-14T06:19:57.512Z
Hash: 13e1b81ffd5b36d7014a
Time: 19330ms
chunk {main} main.js, main.js.map (main) 11.4 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 248 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.3 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.94 MB [initial] [rendered]
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
i ℹ wdm ℹ: Compiled successfully.
```

The server starts at the http://localhost:4200/

You can see the output. It is the initial Angular home screen.


Angular8project
localhost:4200
Welcome to angular8project!
Here are some links to help you start:
Tour of Heroes
CLI Documentation
Angular blog

Project Description

In this project, we will facilitate users:

- To enter their ProductName, Product Description, and ProductPrice to a form and submit the form.
- Validate the form against values. If values are incorrect, then it will be validated at the frontend and form will not be submitted.
- If all values are correct, send the form to the Node.js backend API.
- Store the values inside the MongoDB database.
- Show the data on the frontend.
- Facilitates users to edit and delete the data also.
- Show the data on the MongoD CLI.
- Show the data on MongoDB Compass Community (a GUI of MongoDB database.).

Generate Angular Components

We are going to make a **CRUD operation** to create, read, and update data. So, we will create three components.

Use the following command to generate 3 Angular Components:

```
ng g c product-add --skipTests=true
ng g c product-get --skipTests=true
ng g c product-edit --skipTests=true
```

Note: The "spec" command is deprecated in Angular 8. You have to use "skipTests" instead.

```
Node.js command prompt

C:\Users\JavaTpoint\angular8project>ng g c product-add --skipTests=true
CREATE src/app/product-add/product-add.component.html (30 bytes)
CREATE src/app/product-add/product-add.component.ts (288 bytes)
CREATE src/app/product-add/product-add.component.css (0 bytes)
UPDATE src/app/app.module.ts (493 bytes)

C:\Users\JavaTpoint\angular8project>ng g c product-get --skipTests=true
CREATE src/app/product-get/product-get.component.html (30 bytes)
CREATE src/app/product-get/product-get.component.ts (288 bytes)
CREATE src/app/product-get/product-get.component.css (0 bytes)
UPDATE src/app/app.module.ts (593 bytes)

C:\Users\JavaTpoint\angular8project>ng g c product-edit --skipTests=true
CREATE src/app/product-edit/product-edit.component.html (31 bytes)
CREATE src/app/product-edit/product-edit.component.ts (292 bytes)
CREATE src/app/product-edit/product-edit.component.css (0 bytes)
UPDATE src/app/app.module.ts (697 bytes)

C:\Users\JavaTpoint\angular8project>_
```

All the above 3 components are automatically added to **app.module.ts** file. Now, we have to configure the routing of angular components inside an **app-routing.module.ts** file.

You can check the **app-routing.module.ts** file inside the src >> app folder in your project file. It is created because when we install an angular app, we permit **angular cli** to generate the routing file for us.

Now, write the following code inside an **app-routing.module.ts** file:

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ProductAddComponent } from './product-add/product-add.component';
import { ProductEditComponent } from './product-edit/product-edit.component';
import { ProductGetComponent } from './product-get/product-get.component';
```

```
const routes: Routes = [
  {
    path: 'product/create',
    component: ProductAddComponent
  },
  {
    path: 'edit/:id',
    component: ProductEditComponent
  },
  {
    path: 'products',
    component: ProductGetComponent
  }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Now, you can see inside the **app.component.html** file that directive is there. This directive helps us to render the different component based on the route URI.

Create Angular Navigation

Write the following code inside the **app.component.html** file.

```
<nav class="navbar navbar-expand-sm bg-light">
  <div class="container-fluid">
    <ul class="navbar-nav">
```

```
      <li class="nav-item">
        <a routerLink="product/create" class="nav-link" routerLinkActive="active">
          Create Product
        </a>
      </li>
      <li class="nav-item">
        <a routerLink="products" class="nav-link" routerLinkActive="active">
          Products
        </a>
      </li>
    </ul>
  </div>
</nav>
<div class="container">
  <router-outlet></router-outlet>
</div>
```

Install Angular Routing Progress Indicator

Type the following command to install the **ng2-slim-loading-bar** library.

```
npm install ng2-slim-loading-bar --save
```

If you have installed third-party packages right now, then it is not compatible with Angular 8. To solve the problem between Angular 8 and third-party packages, you have to install the following library.

```
npm install rxjs-compat --save
```

Now, import the **SlimLoadingBarModule** inside an **app.module.ts** file.

```
import { SlimLoadingBarModule } from 'ng2-slim-loading-bar';
imports: [
    ...
    SlimLoadingBarModule
],
```

Now, include the styling that comes with the library inside src >> styles.css file.

```
@import "../node_modules/ng2-slim-loading-bar/style.css";
```

Adding Router Events

Angular RouterModule gives us the following event modules.

- NavigationStart
- NavigationEnd
- NavigationError
- NavigationCancel
- Router
- Event

Now, write the following code inside the **app.component.ts** file.

```
import { Component } from '@angular/core';
import {SlimLoadingBarService} from 'ng2-slim-loading-bar';
import { NavigationCancel,
 Event,
        NavigationEnd,
        NavigationError,
        NavigationStart,
```

```
      Router } from '@angular/router';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'angular8tutorial';
  constructor(private loadingBar: SlimLoadingBarService, private router: Router) {
    this.router.events.subscribe((event: Event) => {
      this.navigationInterceptor(event);
    });
  }
  private navigationInterceptor(event: Event): void {
    if (event instanceof NavigationStart) {
      this.loadingBar.start();
    }
    if (event instanceof NavigationEnd) {
      this.loadingBar.complete();
    }
    if (event instanceof NavigationCancel) {
      this.loadingBar.stop();
    }
    if (event instanceof NavigationError) {
      this.loadingBar.stop();
    }
  }
}
```

In the above code, we have specified in Angular application that when the user navigates from one component to another component, it will show the progress result.

When the user clicks the other route, angular progress indicator start showing, and when the navigation is complete, it will stop displaying. So, it is kind of UX for the user.

The above code intercepts the routing event and add the loading bar component to every route, so that we can see the routing indication every time when we change the routes.

We have to add the **ng2-slim-loading-bar directive** inside the **app.component.html** file at the top of the page.

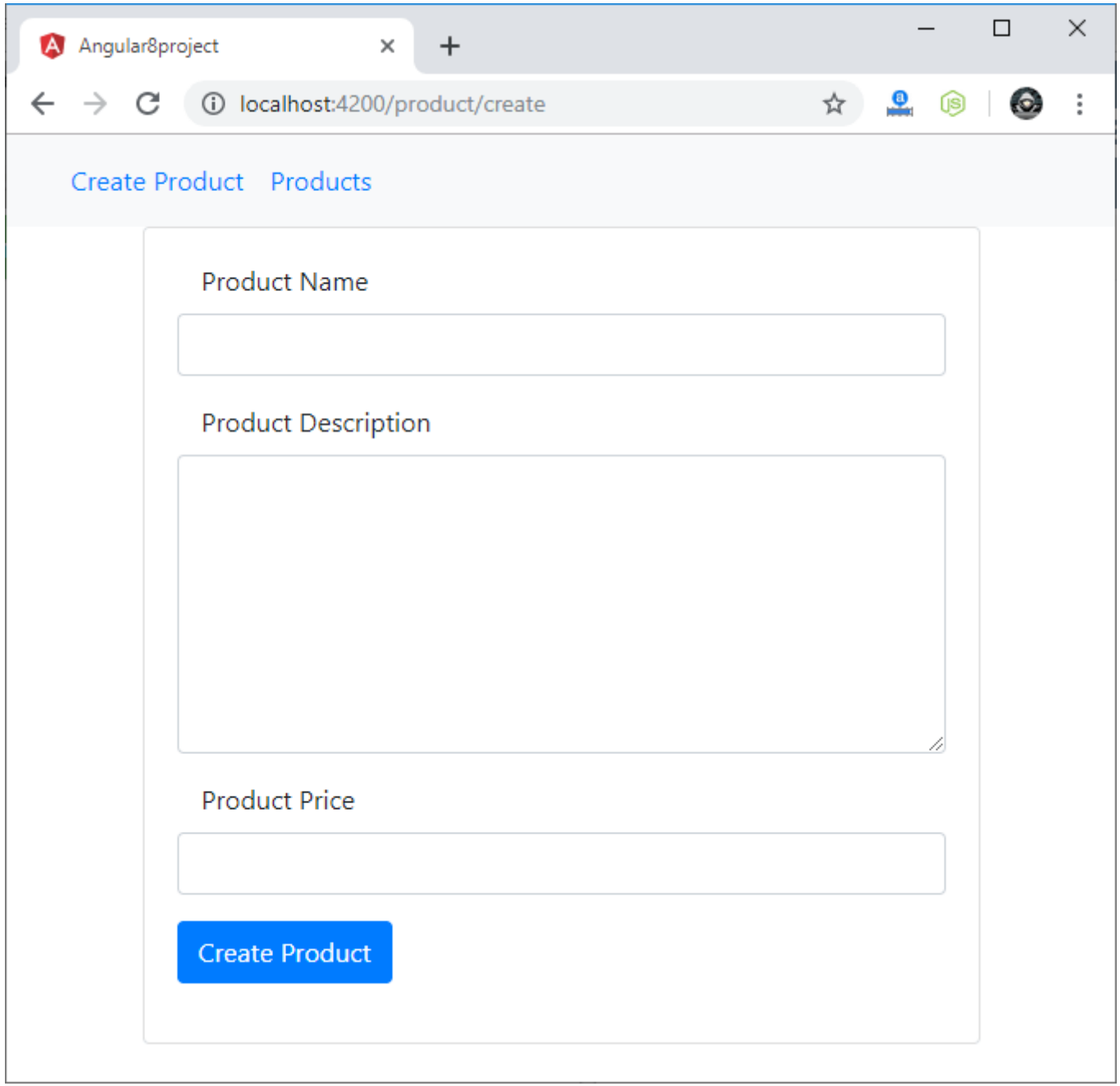
```
<!-- app.component.html -->
<ng2-slim-loading-bar color="blue"></ng2-slim-loading-bar>
<nav class="navbar navbar-expand-sm bg-light">
  <div class="container-fluid">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a routerLink="product/create" class="nav-link" routerLinkActive="active">
          Create Product
        </a>
      </li>
      <li class="nav-item">
        <a routerLink="products" class="nav-link" routerLinkActive="active">
          Products
        </a>
      </li>
    </ul>
  </div>
</nav>
<div class="container">
  <router-outlet></router-outlet>
</div>
```

Add Bootstrap Form

Add the following bootstrap 4 form Inside the product-add.component.html file.

```
<div class="card">
  <div class="card-body">
    <form>
      <div class="form-group">
        <label class="col-md-4">Product Name</label>
        <input type="text" class="form-control" />
      </div>
      <div class="form-group">
        <label class="col-md-4">Product Description </label>
        <textarea class="form-control" rows = 7 cols = "5"></textarea>
      </div>
      <div class="form-group">
        <label class="col-md-4">Product Price</label>
        <input type="text" class="form-control" />
      </div>
      <div class="form-group">
        <button type="submit" class="btn btn-primary">Create Product</button>
      </div>
    </form>
  </div>
</div>
```

Open the localhost browser and see the output. It will look like the image given below.



Add Angular 8 Form Validation

Here, we use **reactive form module** to add the validation in our form. Import the **ReactiveFormsModule** inside the **app.module.ts** file.

```
import { ReactiveFormsModule } from '@angular/forms';
imports: [
    ...
    ReactiveFormsModule
],
```

Here, you need to aware that it is not a template driven form, so you need to change in the **app.component.ts** file.

Here, we have to import the **FormGroup, FormBuilder, Validators modules** from **@angular/forms** and create a constructor and instantiate the **FormBuilder**.

So, write the following code inside the **product-add.component.ts** file.

```
import { Component, OnInit } from '@angular/core';
import { FormGroup,  FormBuilder,  Validators } from '@angular/forms';
@Component({
  selector: 'app-product-add',
  templateUrl: './product-add.component.html',
  styleUrls: ['./product-add.component.css']
})
export class ProductAddComponent implements OnInit {
  angForm: FormGroup;
  constructor(private fb: FormBuilder) {
    this.createForm();
  }
  createForm() {
    this.angForm = this.fb.group({
      ProductName: ['', Validators.required ],
      ProductDescription: ['', Validators.required ],
```

```
      ProductPrice: ['', Validators.required ]
    });
  }
  ngOnInit() {
  }
}
```

Here, we use form builder to handle all the validation. Now, in that constructor, we have to create a form with the validation rules. In our example, there are three fields. If the input text is empty, then it will give an error, and we need to display that error.

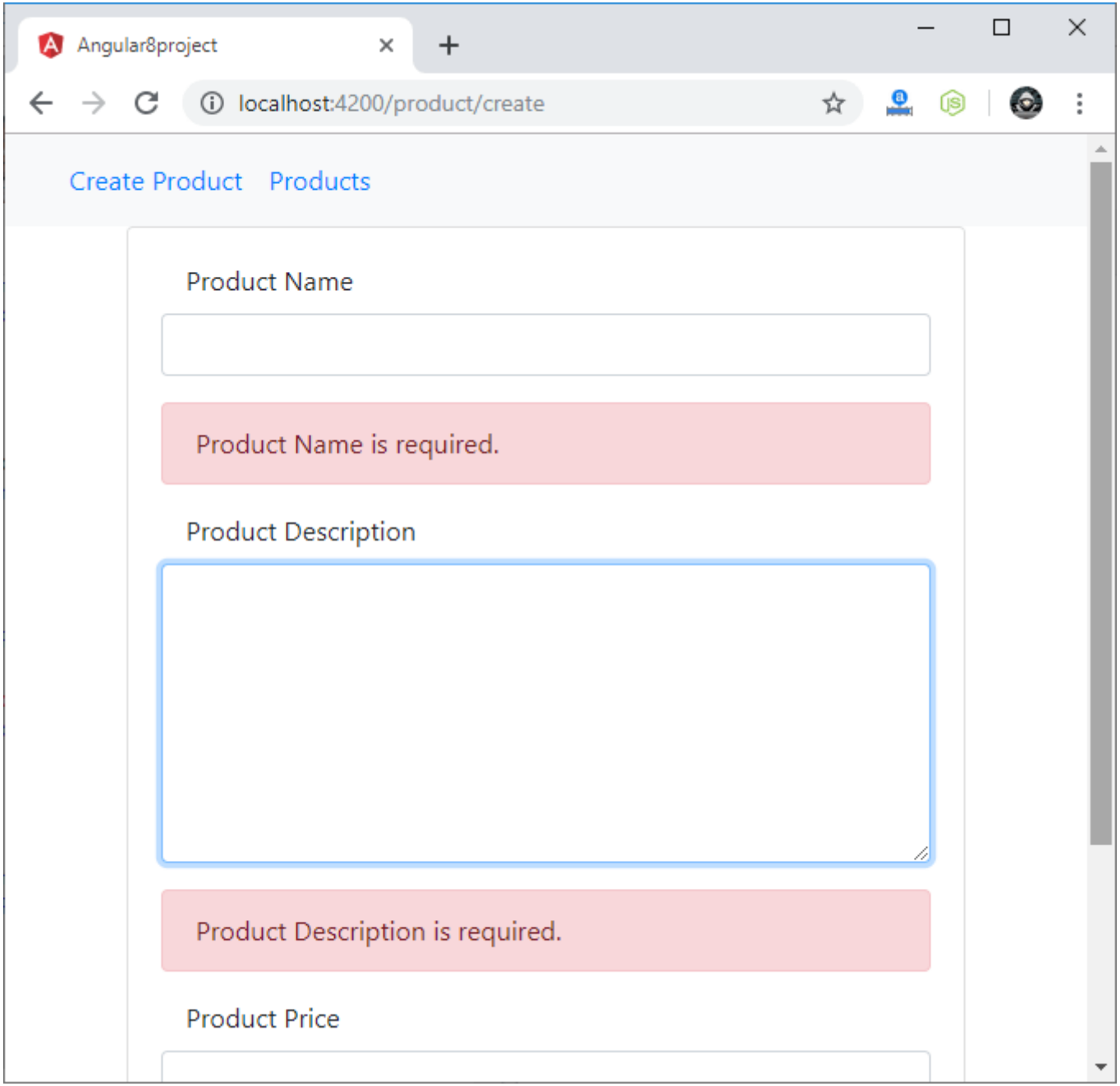
Now, write the following code inside the **product-add.component.html** file.

```
<div class="card">
  <div class="card-body">
    <form [formGroup]="angForm" novalidate>
      <div class="form-group">
        <label class="col-md-4">Product Name</label>
        <input type="text" class="form-control"
          formControlName="ProductName"
          #ProductName />
      </div>
      <div *ngIf="angForm.controls['ProductName'].invalid && (angForm.controls['ProductName'].dirty || angForm.controls['Prod
danger">
        <div *ngIf="angForm.controls['ProductName'].errors.required">
          Product Name is required.
        </div>
      </div>
      <div class="form-group">
        <label class="col-md-4">Product Description </label>
        <textarea class="form-control" rows = 7 cols = "5"
```

```
      formControlName="ProductDescription"
      #ProductDescription></textarea>
    </div>
    <div *ngIf="angForm.controls['ProductDescription'].invalid && (angForm.controls['ProductDescription'].dirty || angForm.co
danger">
      <div *ngIf="angForm.controls['ProductDescription'].errors.required">
        Product Description is required.
      </div>
    </div>
    <div class="form-group">
      <label class="col-md-4">Product Price</label>
      <input type="text" class="form-control"
        formControlName="ProductPrice"
        #ProductPrice
      />
    </div>
    <div *ngIf="angForm.controls['ProductPrice'].invalid && (angForm.controls['ProductPrice'].dirty || angForm.controls['Produ
danger">
      <div *ngIf="angForm.controls['ProductPrice'].errors.required">
        Product Price is required.
      </div>
    </div>
    <div class="form-group">
      <button type="submit" class="btn btn-primary"
      [disabled]="angForm.pristine || angForm.invalid" >
        Create Product
      </button>
    </div>
  </form>
</div>
```

```
</div>
```

Now, you can check that the form is validated.



Configure the HttpClientModule

The Front-end applications always need HTTP protocol to communicate with the backend services. Modern browsers support two different APIs for making HTTP requests: the javaHttpRequest interface and the fetch() API.

Import the HttpClientModule inside an app.module.ts file.

```
// app.module.ts
import { HttpClientModule } from '@angular/common/http';
imports: [
   ...
    HttpClientModule
 ],
```

Create a model file

Inside the src >> app folder, create one file called **Product.ts** and add the following code.

```
export default class Product {
  ProductName: string;
  ProductDescription: string;
  ProductPrice: number;
}
```

Create an Angular Service file

Type the following command to generate the service file.

```
ng g service products --skipTests=true
```

After creation, it will look like this:

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class ProductsService {

  constructor() { }
}
```

Now, import the **products.service.ts** file into the **app.module.ts** file.

```
import { ProductsService } from './products.service';
providers: [ ProductsService ],
```

Submit the data to the node server

Now, we have to write the code that will send the HTTP POST request with the data to the Node.js server and after that save the data into the MongoDB database.

Write the following code inside the **products.service.ts** file.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Injectable({
  providedIn: 'root'
})
export class ProductsService {
  uri = 'http://localhost:4000/products';
  constructor(private http: HttpClient) { }
```

```
  addProduct(ProductName, ProductDescription, ProductPrice) {
    const obj = {
      ProductName,
      ProductDescription,
      ProductPrice
    };
    console.log(obj);
    this.http.post(`${this.uri}/add`, obj)
        .subscribe(res => console.log('Done'));
  }
}
```

Note: Here, we have defined our backend API URL, but we have not created any backend yet. Let's create it.

Now, we have to add the click event to the Add Product Button. So, add the following code inside the **product-add.component.html** file.

```
<div class="form-group">
        <button (click) = "addProduct(ProductName.value, ProductDescription.value, ProductPrice.value)" type="submit" class="b
primary"
        [disabled]="angForm.pristine || angForm.invalid" >
          Create Product
        </button>
</div>
```

Now, add the addProduct() function inside the **product-add.component.ts** file. So, write the following code inside the **product-add.component.ts** file.

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

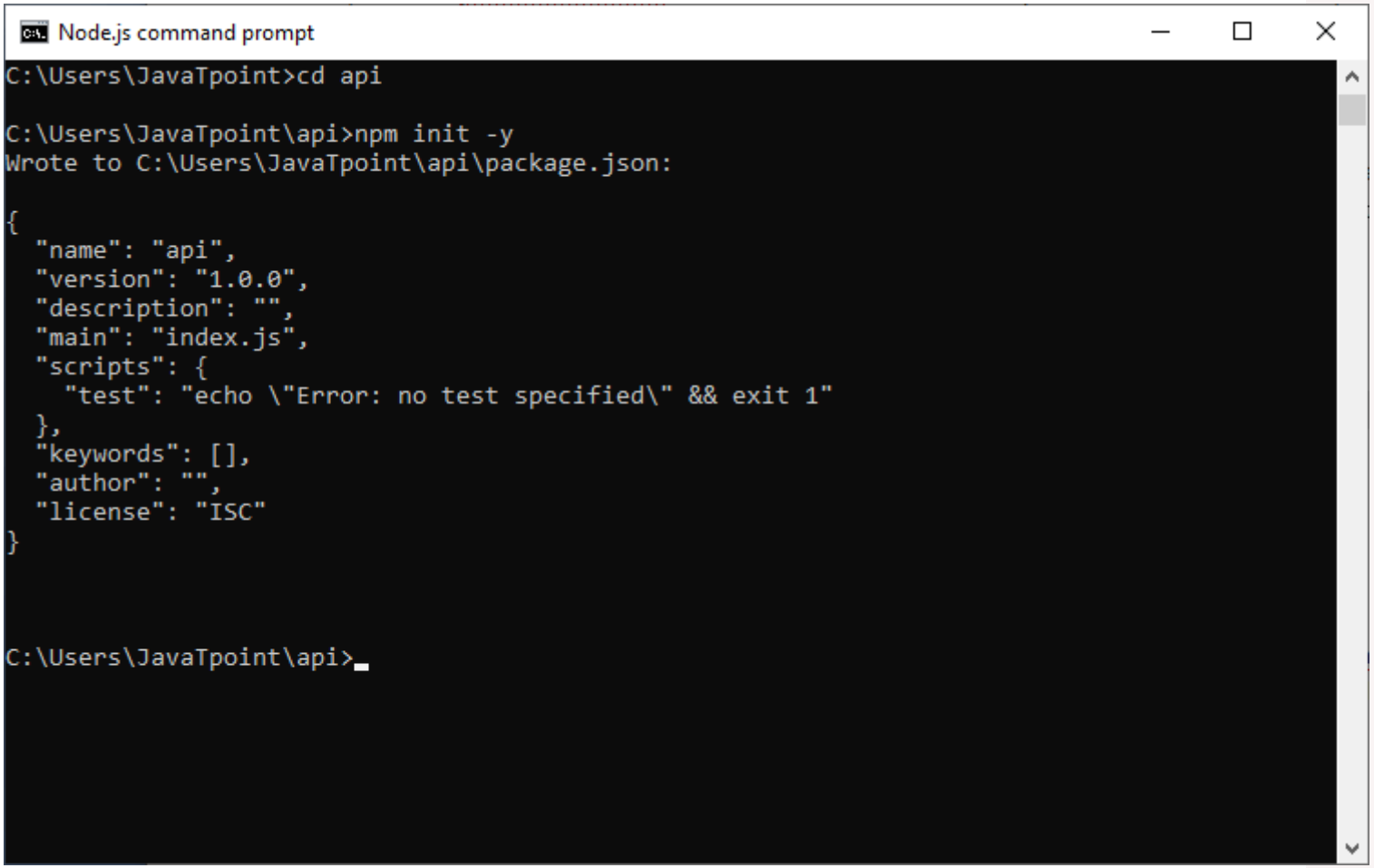
```
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { ProductsService } from '../products.service';
@Component({
  selector: 'app-product-add',
  templateUrl: './product-add.component.html',
  styleUrls: ['./product-add.component.css']
})
export class ProductAddComponent implements OnInit {

  angForm: FormGroup;
  constructor(private fb: FormBuilder, private ps: ProductsService) {
    this.createForm();
  }
  createForm() {
    this.angForm = this.fb.group({
      ProductName: ['', Validators.required ],
      ProductDescription: ['', Validators.required ],
      ProductPrice: ['', Validators.required ]
    });
  }
  addProduct(ProductName, ProductDescription, ProductPrice) {
    this.ps.addProduct(ProductName, ProductDescription, ProductPrice);
  }
  ngOnInit() {
  }
}
```

Create a Node.js backend API

Create a folder called the **api** inside an angular root folder, and go inside that folder. It will be the complete separate project from your Angular project. So, its node_modules folders are different from the Angular project.

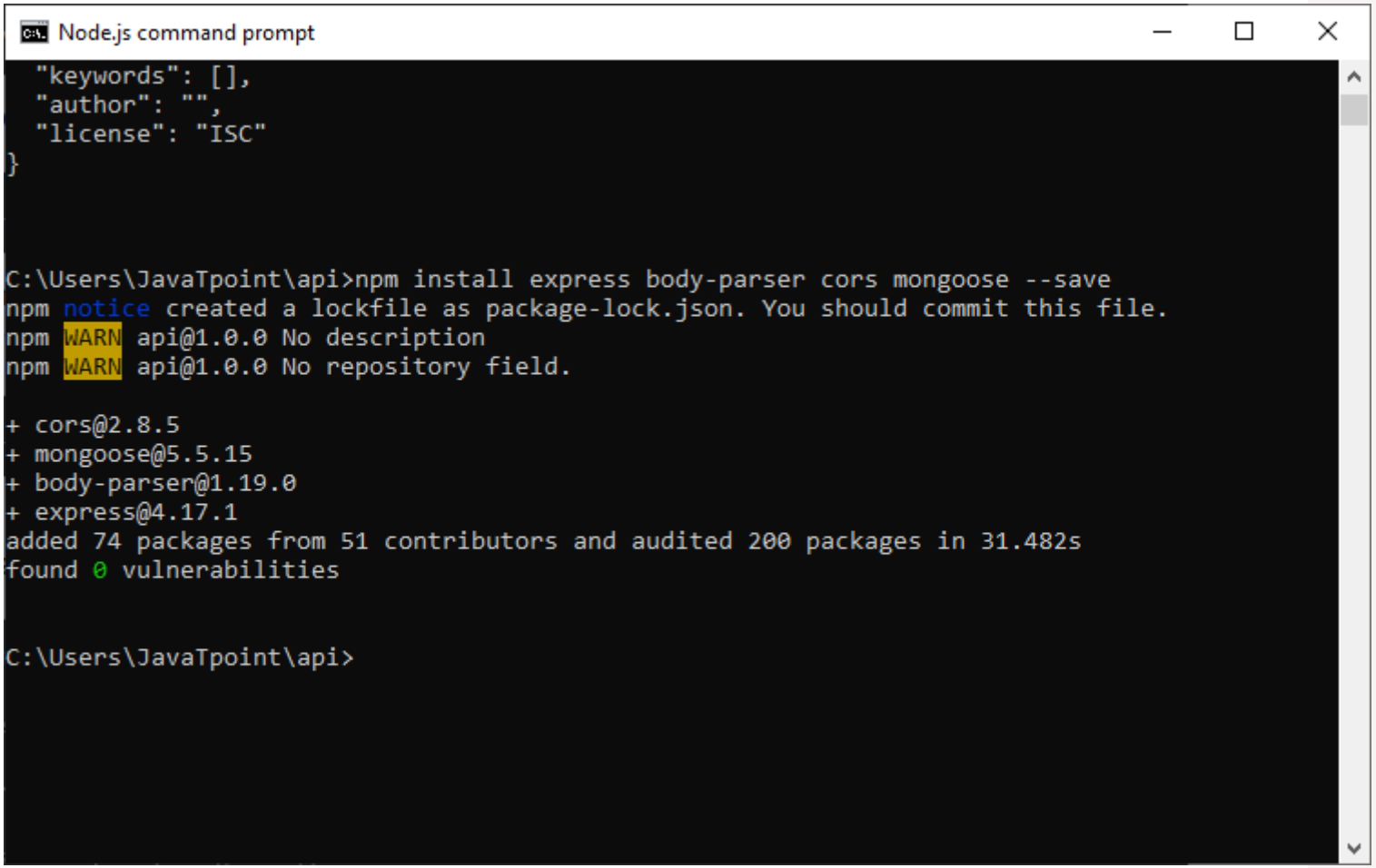
Open the terminal inside the api folder and type the following command. It will generate the package.json file using NPM.

```
npm init -y
```



Install the following node specific modules.

```
npm install express body-parser cors mongoose --save
```

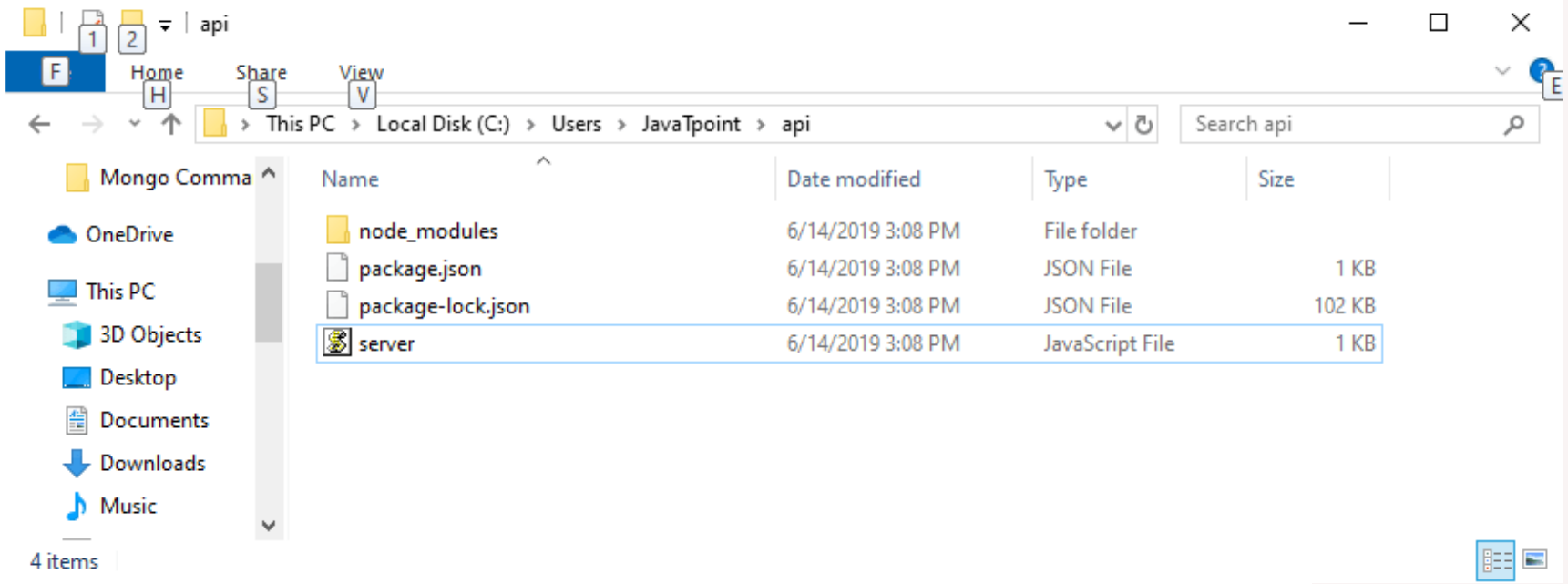


Install the **nodemon server**, it will facilitate you to not to start node server every time you make changes it. It restarts the node.js server automatically.

```
npm install nodemon --save-dev
```

```
Select Node.js command prompt
+ mongoose@5.5.15
+ body-parser@1.19.0
+ express@4.17.1
added 74 packages from 51 contributors and audited 200 packages in 31.482s
found 0 vulnerabilities


C:\Users\JavaTpoint\api>npm install nodemon --save-dev

> nodemon@1.19.1 postinstall C:\Users\JavaTpoint\api\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
 > https://opencollective.com/nodemon/donate

npm WARN api@1.0.0 No description
npm WARN api@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: want
ed {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ nodemon@1.19.1
added 216 packages from 129 contributors and audited 2436 packages in 39.759s
found 0 vulnerabilities


C:\Users\JavaTpoint\api>
```

Now, inside the api folder, create one file called the **server.js** file. Add the following code inside the server.js file.




```
const express = require('express'),
    path = require('path'),
    bodyParser = require('body-parser'),
    cors = require('cors'),
    mongoose = require('mongoose');
    const app = express();
    let port = process.env.PORT || 4000;
    const server = app.listen(function(){
        console.log('Listening on port ' + port);
    });
```

Now, connect to the MongoDB database with our node express application.

If you have not installed a MongoDB database, then install it and then start the mongodb server using the following command.

How to Install MongoDB Click Here...

```
mongod
```

Now, we have connected to the database.

Create one file called **DB.js** inside the api root project folder. Write the following code inside a **DB.js** file.

```
module.exports = {
  DB: 'mongodb://localhost:27017/ng8crud'
};
```

Import this **DB.js** file inside our **server.js** file and use the mongoose library to set up the database connection with MongoDB. Write a following code inside the **server.js** file to connect our MongoDB application to the Node.js server.

```
const express = require('express'),
    path = require('path'),
    bodyParser = require('body-parser'),
    cors = require('cors'),
    mongoose = require('mongoose'),
    config = require('./DB');
    mongoose.Promise = global.Promise;
    mongoose.connect(config.DB, { useNewUrlParser: true }).then(
      () => {console.log('Database is connected') },
      err => { console.log('Can not connect to the database'+ err)}
    );
    const app = express();
    app.use(bodyParser.json());
    app.use(cors());
    const port = process.env.PORT || 4000;
    const server = app.listen(port, function(){
     console.log('Listening on port ' + port);
    });
```

Save this **server.js** file and go to the terminal and start the node server using the following command.

```
npm start
```

So, right now, there are three servers running:

- Angular Development Server
- Node.js Server
- MongoDB server

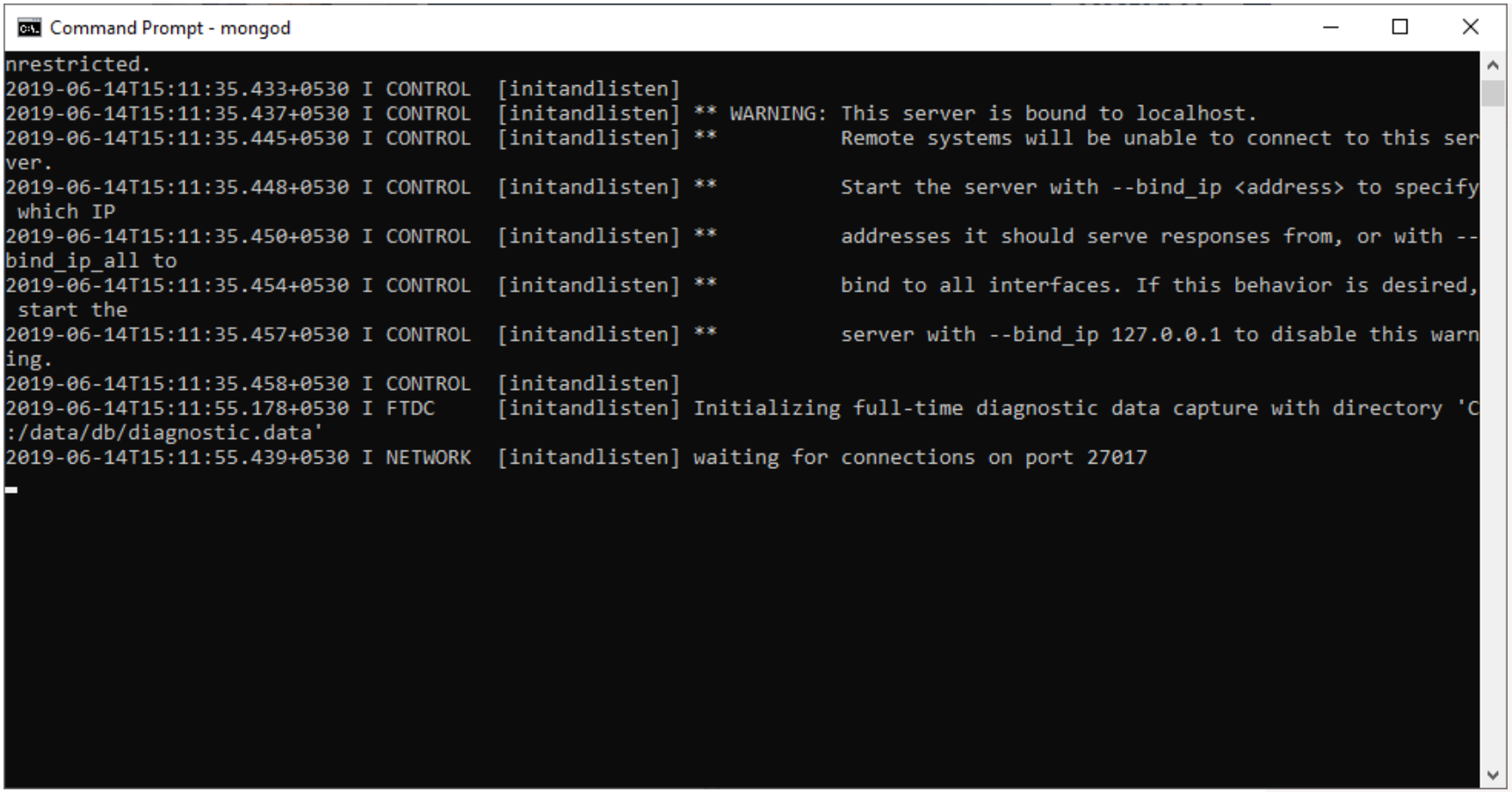
Angular Server:

```
ng serve
chunk {styles} styles.js, styles.js.map (styles) 963 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.3 MB [initial] [rendered]
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localh
ost:4200/ **
i ℹwdmℹ: Compiled successfully.
i ℹwdmℹ: Compiling...

Date: 2019-06-14T10:33:12.385Z - Hash: ad7cd86475c82d55f404 - Time: 550ms
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 27.8 kB [initial] [rendered]
i ℹwdmℹ: Compiled successfully.
i ℹwdmℹ: Compiling...

Date: 2019-06-14T10:40:10.083Z - Hash: 87b936325aa00bfe26a7 - Time: 3953ms
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 27.6 kB [initial] [rendered]
i ℹwdmℹ: Compiled successfully.
i ℹwdmℹ: Compiling...

Date: 2019-06-14T10:53:36.437Z - Hash: 87b936325aa00bfe26a7 - Time: 5075ms
5 unchanged chunks
i ℹwdmℹ: Compiled successfully.
```

Node.js Server:

```
npm
    stars, start, stop, t, team, test, token, tst, un,
    uninstall, unpublish, unstar, up, update, v, version, view,
    whoami

npm <command> -h  quick help on <command>
npm -l            display full usage info
npm help <term>   search for help on <term>
npm help npm      involved overview

Specify configs in the ini-formatted file:
    C:\Users\JavaTpoint\.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@6.9.0 C:\Program Files\nodejs\node_modules\npm


C:\Users\JavaTpoint\api>npm start

> api@1.0.0 start C:\Users\JavaTpoint\api
> node server.js

Listening on port 4000
Database is connected
```

MongoDB server:



Create model and routes for your application

Now, we have to create two folders inside the api root folder called routes and models.

Let's create one model called **Product.js** in models folder having the following code.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
// Define collection and schema for Product
let Product = new Schema({
  ProductName: {
```

```
    type: String
  },
  ProductDescription: {
    type: String
  },
  ProductPrice: {
    type: Number
  }
},{
    collection: 'Product'
});
module.exports = mongoose.model('Product', Product);
```

Here, we have defined our schema for the Product collection. We have three fields called **ProductName**, **ProductDescription**, **ProductPrice**.

In the routes folder, create one file called the product.route.js.

Write the following CRUD code inside the **product.route.js** file.

```
const express = require('express');
const app = express();
const productRoutes = express.Router();
// Require Product model in our routes module
let Product = require('../models/Product');
// Defined store route
productRoutes.route('/add').post(function (req, res) {
  let product = new Product(req.body);
  product.save()
    .then(product => {
      res.status(200).json({'Product': 'Product has been added successfully'});
```

```
    })
    .catch(err => {
    res.status(400).send("unable to save to database");
    });
});
// Defined get data(index or listing) route
productRoutes.route('/').get(function (req, res) {
  Product.find(function (err, products){
    if(err){
      console.log(err);
    }
    else {
      res.json(products);
    }
  });
});
// Defined edit route
productRoutes.route('/edit/:id').get(function (req, res) {
  let id = req.params.id;
  Product.findById(id, function (err, product){
      res.json(product);
  });
});
//  Defined update route
productRoutes.route('/update/:id').post(function (req, res) {
  Product.findById(req.params.id, function(err, product) {
    if (!product)
      res.status(404).send("Record not found");
    else {
      product.ProductName = req.body.ProductName;
```

```
      product.ProductDescription = req.body.ProductDescription;
      product.ProductPrice = req.body.ProductPrice;
 product.save().then(product => {
         res.json('Update complete');
      })
      .catch(err => {
           res.status(400).send("unable to update the database");
      });
    }
  });
});
// Defined delete | remove | destroy route
productRoutes.route('/delete/:id').get(function (req, res) {
    Product.findByIdAndRemove({_id: req.params.id}, function(err, product){
        if(err) res.json(err);
        else res.json('Successfully removed');
    });
});
module.exports = productRoutes;
```

Now, we have all the CRUD operations set up on the route file; we need to import inside the server.js file.

So, update the server.js file with the following code:

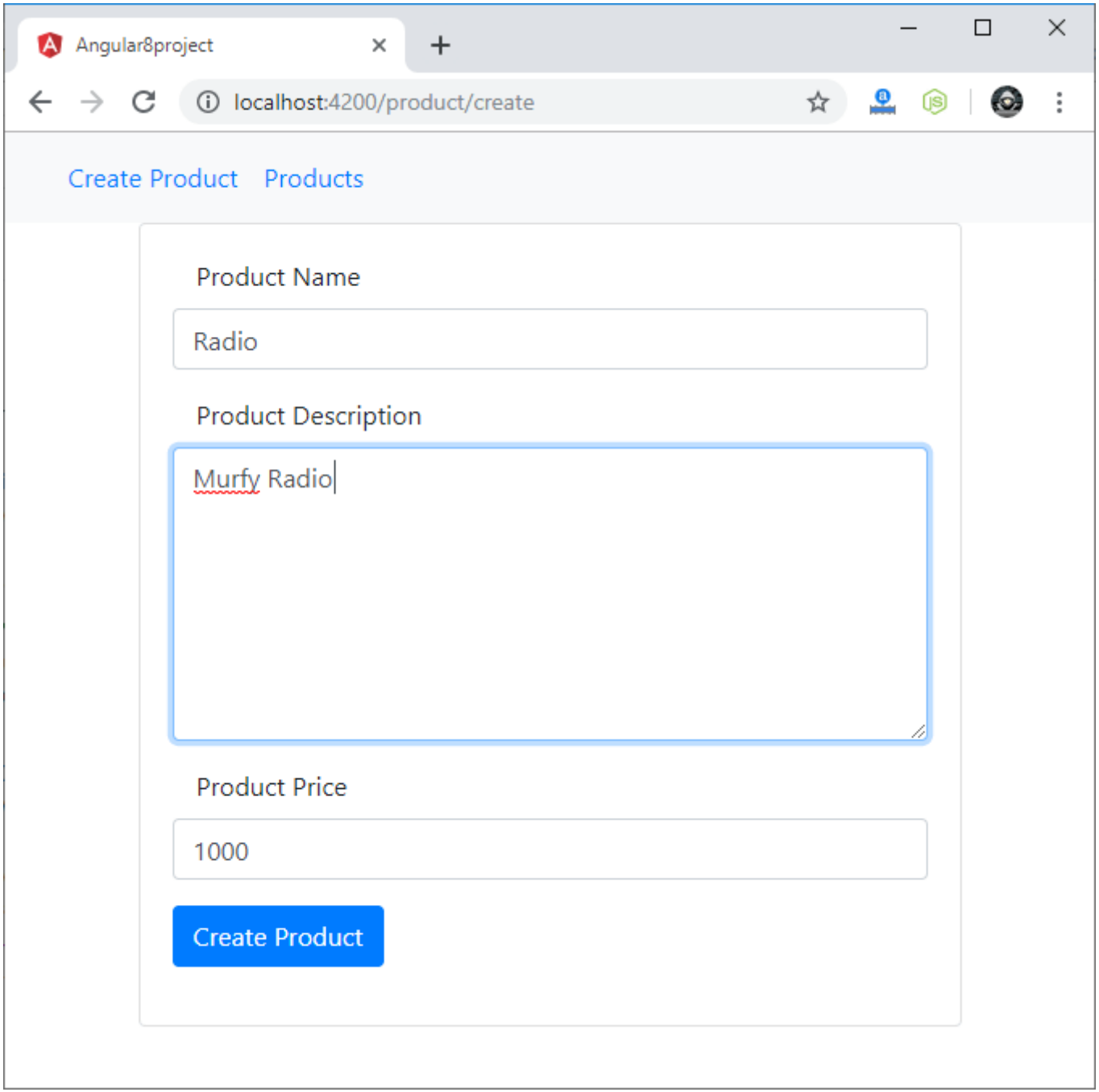
```
const express = require('express'),
    path = require('path'),
    bodyParser = require('body-parser'),
    cors = require('cors'),
    mongoose = require('mongoose'),
    config = require('./DB');
   const productRoute = require('./routes/product.route');
```

```
mongoose.Promise = global.Promise;
mongoose.connect(config.DB, { useNewUrlParser: true }).then(
  () => {console.log('Database is connected') },
  err => { console.log('Can not connect to the database'+ err)}
);
const app = express();
app.use(bodyParser.json());
app.use(cors());
app.use('/products', productRoute);
const port = process.env.PORT || 4000;
const server = app.listen(port, function(){
 console.log('Listening on port ' + port);
});
```

Now, start the node.js server, if you yet not started. If it is already started then let's check the data store functionality.

Test the data store functionality

Now, we will enter some data to test if it is storing or not.



First, open the mongo shell on the 4th tab because all the other three tabs are occupied at the moment.

Use the following command to open mongo shell:

```
mongo
```

```
Select Command Prompt - mongo
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\JavaTpoint>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4b62b143-f652-41b0-8112-af3bc4f5d1c7") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-06-13T17:54:39.068+0530 I CONTROL  [initandlisten]
2019-06-13T17:54:39.069+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-06-13T17:54:39.069+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is u
nrestricted.
2019-06-13T17:54:39.069+0530 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
```

Here, you can use

- **show dbs** command to see the databases
- use **database_name** command to use the database and **db.collection_name.find()** to find the query.

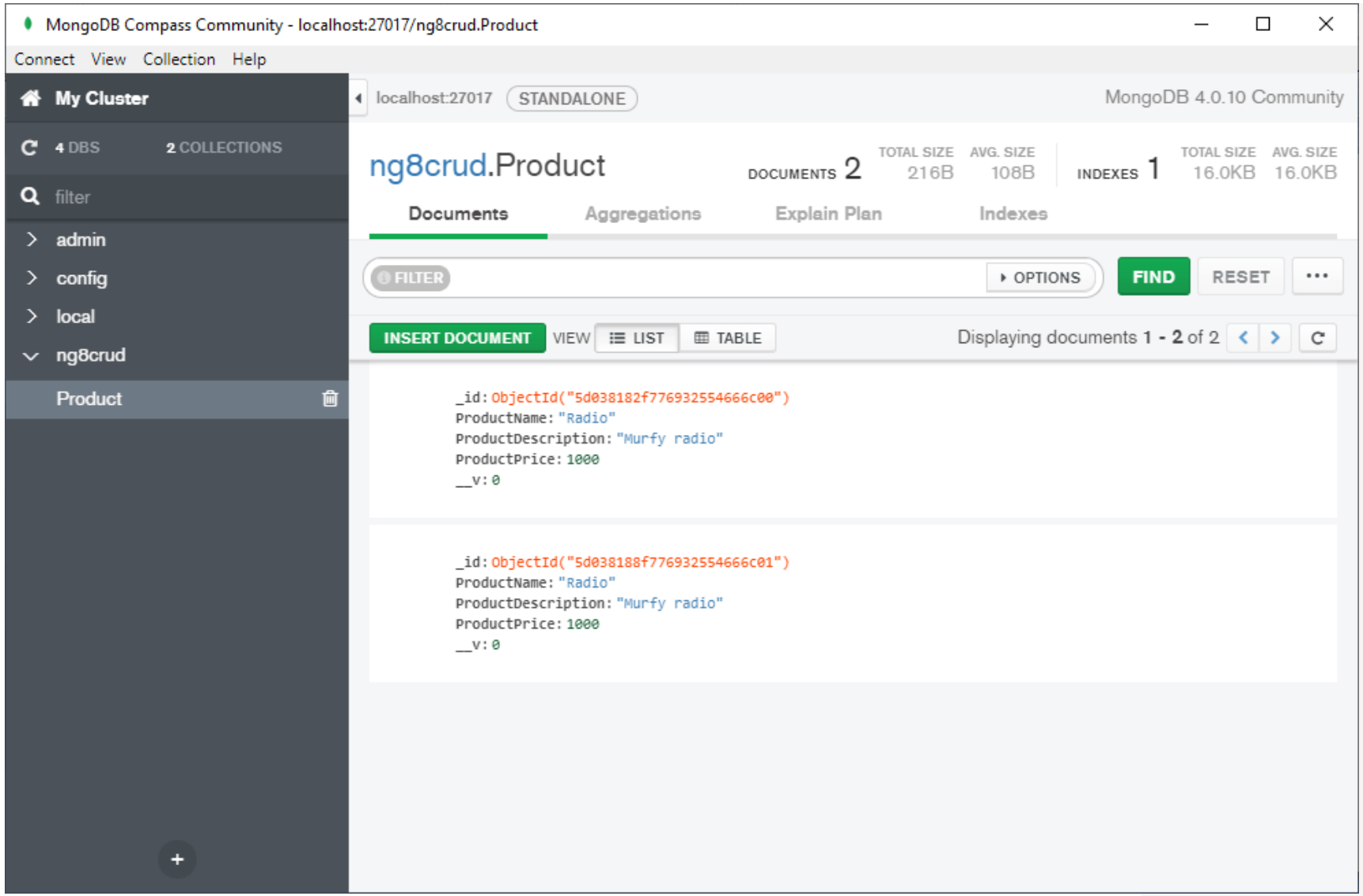
Command Prompt - mongo

```
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
ng8crud  0.000GB
> show collections;
> use ng8crud
switched to db ng8crud
> db.Product.find()
{ "_id" : ObjectId("5d038182f776932554666c00"), "ProductName" : "Radio", "ProductDescription" : "Murfy radio", "ProductPrice" : 100
0, "__v" : 0 }
{ "_id" : ObjectId("5d038188f776932554666c01"), "ProductName" : "Radio", "ProductDescription" : "Murfy radio", "ProductPrice" : 100
0, "__v" : 0 }
> db.Product.find().pretty
function () {
    this._prettyShell = true;
    return this;
}
> db.Product.find().pretty()
{
        "_id" : ObjectId("5d038182f776932554666c00"),
        "ProductName" : "Radio",
        "ProductDescription" : "Murfy radio",
        "ProductPrice" : 1000,
        "__v" : 0
}
```

Note: Here, we have used **db.collection_name.find().pretty()** command to prettify the result only.

Alternative way to see the stored items in database

You can also use the **MongoDB Compass Community** to see the databases and entries if we prefer a GUI.



Display the data on the frontend browser

If you want to display the data on the frontend browser, use the following code inside the **product-get.component.html** file.

```
<table class="table table-hover">
  <thead>
```

```
  <tr>
    <td>Product Name</td>
    <td>Product Description</td>
    <td>Product Price</td>
    <td colspan="2">Actions</td>
  </tr>
  </thead>
  <tbody>
    <tr *ngFor="let product of products">
      <td>{{ product.ProductName }}</td>
      <td>{{ product.ProductDescription }}</td>
      <td>{{ product.ProductPrice }}</td>
      <td><a [routerLink]="['/edit', product._id]" class="btn btn-primary">Edit</a></td>
      <td><a [routerLink]="" class="btn btn-danger">Delete</a></td>
    </tr>
  </tbody>
</table>
```

Now, we have to write the function inside the **products.service.ts** file that fetches the products data from the MongoDB database and display at the Angular application.

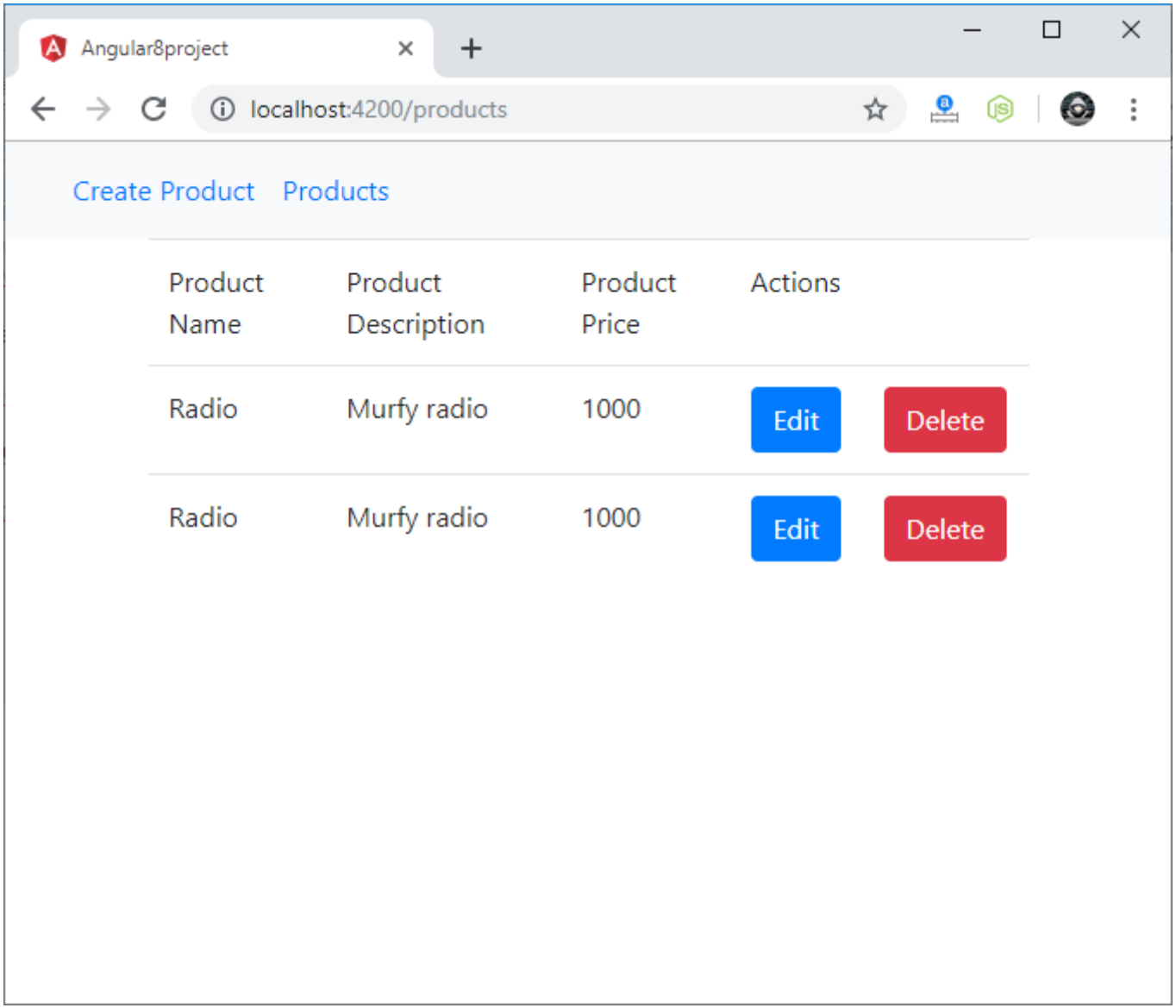
```
getProducts() {
  return this
      .http
      .get(`${this.uri}`);
  }
```

Include this **products.service.ts** file and **Product.ts** file inside the **product-get.component.ts** file.

Write the following code inside the **product-get.component.ts** file.

```
import { Component, OnInit } from '@angular/core';
import Product from '../Product';
import { ProductsService } from '../products.service';
@Component({
  selector: 'app-product-get',
  templateUrl: './product-get.component.html',
  styleUrls: ['./product-get.component.css']
})
export class ProductGetComponent implements OnInit {
  products: Product[];
  constructor(private ps: ProductsService) { }
  ngOnInit() {
    this.ps
      .getProducts()
      .subscribe((data: Product[]) => {
        this.products = data;
    });
  }
}
```

Now, save the file, go to the browser and switch to this URL: http://localhost:4200/products. You will see the following result.



Now, you have succeeded to fetch data from the database and show to the browser.

Edit and Update the Data

First, we need to fetch the _id wise data from the MongoDB database and then display that data in the **product-edit.component.html** file.

Write the following code inside the **product-edit.component.ts** file.

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { ProductsService } from '../products.service';
@Component({
  selector: 'app-product-edit',
  templateUrl: './product-edit.component.html',
  styleUrls: ['./product-edit.component.css']
})
export class ProductEditComponent implements OnInit {
  angForm: FormGroup;
 product: any = {};
  constructor(private route: ActivatedRoute, private router: Router, private ps: ProductsService, private fb: FormBuilder) {
      this.createForm();
 }
  createForm() {
    this.angForm = this.fb.group({
      ProductName: ['', Validators.required ],
      ProductDescription: ['', Validators.required ],
      ProductPrice: ['', Validators.required ]
    });
  }
  ngOnInit() {
    this.route.params.subscribe(params => {
        this.ps.editProduct(params['id']).subscribe(res => {
          this.product = res;
      });
    });
```

```
  }
}
```

Here, when the **product-edit component.ts** render, it will call the **ngOnInit** method and send an HTTP request to the node server and fetch the data from an _id to display inside the **product-edit component.html** file.

Now, inside the **products.service.ts** file, we need to code the editProduct function to send an HTTP request.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Injectable({
  providedIn: 'root'
})
export class ProductsService {
  uri = 'http://localhost:4000/products';
  constructor(private http: HttpClient) { }
  addProduct(ProductName, ProductDescription, ProductPrice) {
    console.log(ProductName, ProductDescription, ProductPrice);
    const obj = {
      ProductName,
      ProductDescription,
      ProductPrice
    };
    this.http.post(`${this.uri}/add`, obj)
        .subscribe(res => console.log('Done'));
  }
  getProducts() {
    return this
          .http
          .get(`${this.uri}`);
  }
```

```
  editProduct(id) {
    return this
          .http
          .get(`${this.uri}/edit/${id}`);
    }
}
```

At last, write the form inside the **product-edit.component.html** file.

```
<!-- product-edit.component.html -->
<div class="card">
  <div class="card-body">
    <form [formGroup]="angForm" novalidate>
      <div class="form-group">
        <label class="col-md-4">Product Name</label>
        <input type="text" class="form-control"
          formControlName="ProductName"
          #ProductName
          [(ngModel)] = "product.ProductName"/>
      </div>
      <div *ngIf="angForm.controls['ProductName'].invalid && (angForm.controls['ProductName'].dirty || angForm.controls['Prod
danger">
        <div *ngIf="angForm.controls['ProductName'].errors.required">
          Product Name is required.
        </div>
      </div>
      <div class="form-group">
        <label class="col-md-4">Product Description </label>
        <textarea class="form-control" rows = 7 cols = "5"
        formControlName="ProductDescription"
```

```
      #ProductDescription [(ngModel)] = "product.ProductDescription"></textarea>
    </div>
    <div *ngIf="angForm.controls['ProductDescription'].invalid && (angForm.controls['ProductDescription'].dirty || angForm.co
danger">
      <div *ngIf="angForm.controls['ProductDescription'].errors.required">
        Product Description is required.
      </div>
    </div>
    <div class="form-group">
      <label class="col-md-4">Product Price</label>
      <input type="text" class="form-control"
        formControlName="ProductPrice"
        #ProductPrice
        [(ngModel)] = "product.ProductPrice"
      />
    </div>
    <div *ngIf="angForm.controls['ProductPrice'].invalid && (angForm.controls['ProductPrice'].dirty || angForm.controls['Produ
danger">
      <div *ngIf="angForm.controls['ProductPrice'].errors.required">
        Product Price is required.
      </div>
    </div>
    <div class="form-group">
      <button (click) = "updateProduct(ProductName.value, ProductDescription.value, ProductPrice.value)" type="submit" class
primary"
      [disabled]="angForm.invalid" >
        Update Product
      </button>
    </div>
  </form>
```

```
  </div>
</div>
```

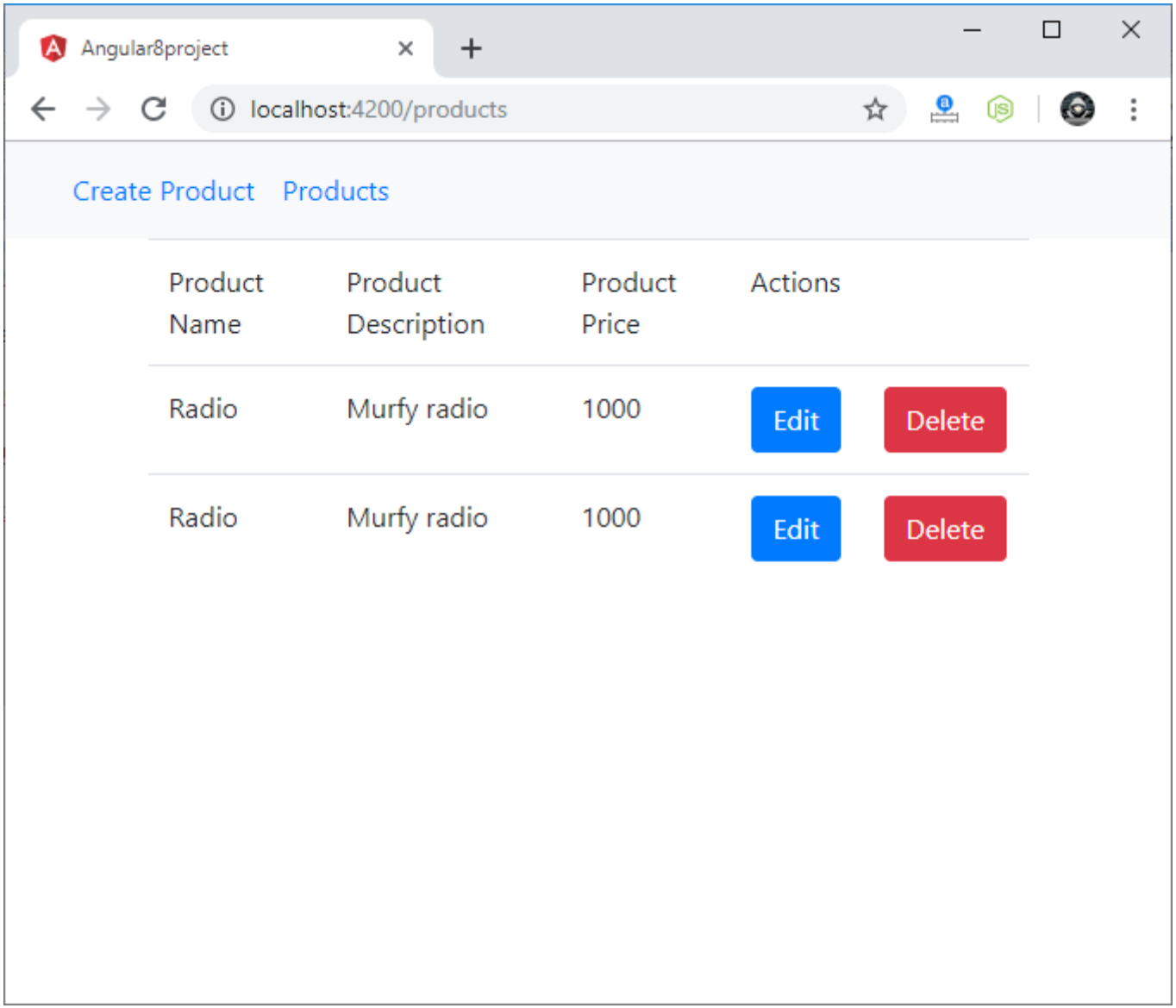
Now, use the following code to update the data inside the **products.service.ts** file, we need to write the function that updates the data.

```
updateProduct(ProductName, ProductDescription, ProductPrice, id) {
    const obj = {
      ProductName,
      ProductDescription,
      ProductPrice
    };
    this
      .http
      .post(`${this.uri}/update/${id}`, obj)
      .subscribe(res => console.log('Done'));
}
```

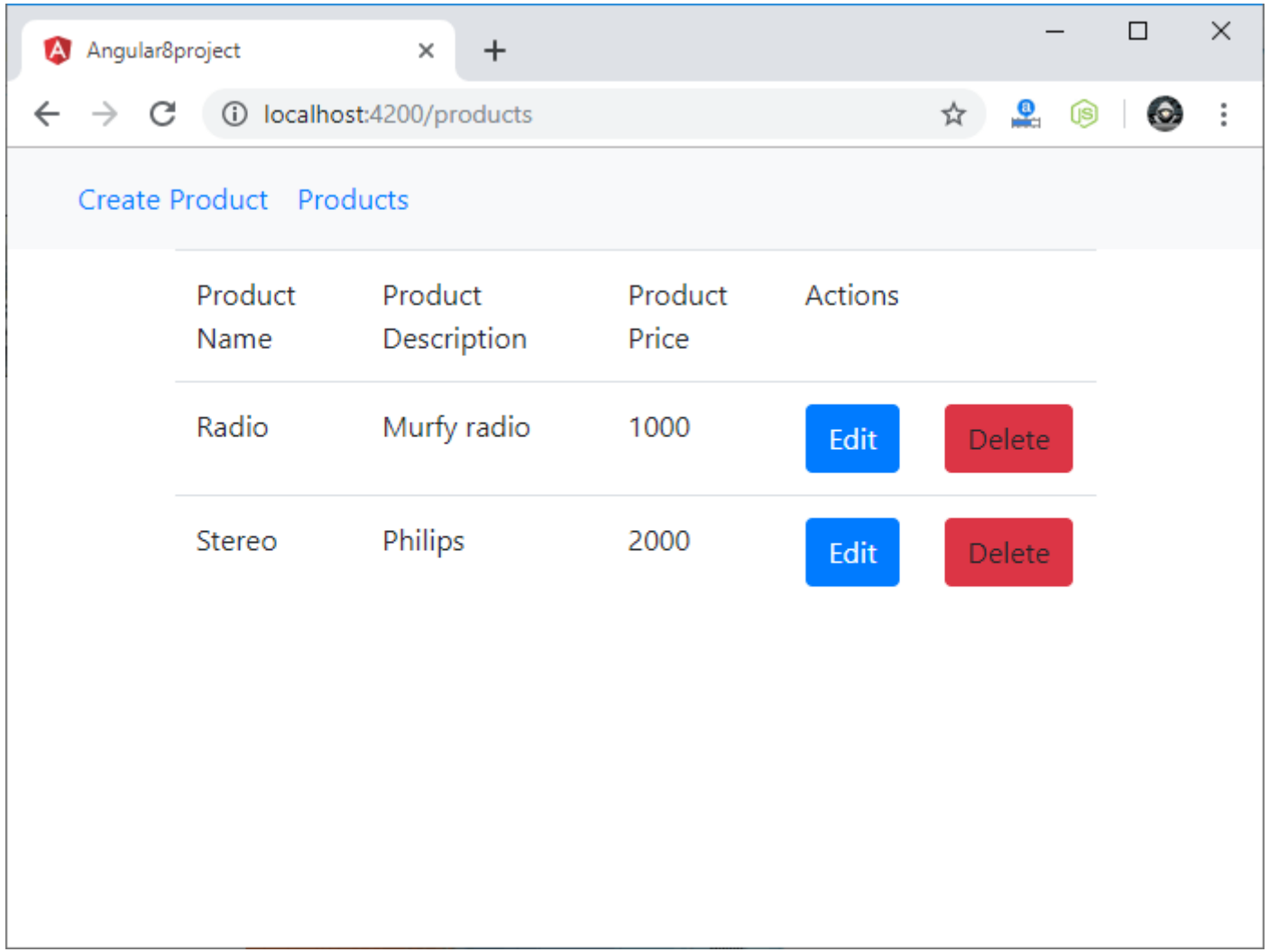
Now, write the **updateProduct() function** inside **product-edit.component.ts** file.

```
updateProduct(ProductName, ProductDescription, ProductPrice, id) {
    this.route.params.subscribe(params => {
      this.ps.updateProduct(ProductName, ProductDescription, ProductPrice, params.id);
      this.router.navigate(['products']);
    });
  }
```

Let's check the edit functionality. In the above read operation, we have seen that there are two duplicate entries of Radio like this:



Now, we will change the second entry of Radio with Stereo.



Delete the Data

This is the last step of CRUD operation. We have successfully deployed CREATE, READ, and UPDATE operation till now.

So, to make the DELETE operation possible, we have to define a click event on the delete button inside the **product-get.component.html** file.

```
<tbody>
```

```
    <tr *ngFor="let product of products">
      <td>{{ product.ProductName }}</td>
      <td>{{ product.ProductDescription }}</td>
      <td>{{ product.ProductPrice }}</td>
      <td><a [routerLink]="['/edit', product._id]" class="btn btn-primary">Edit</a></td>
      <td><a (click) = "deleteProduct(product._id)" class="btn btn-danger">Delete</a>
    </tr>
  </tbody>
```

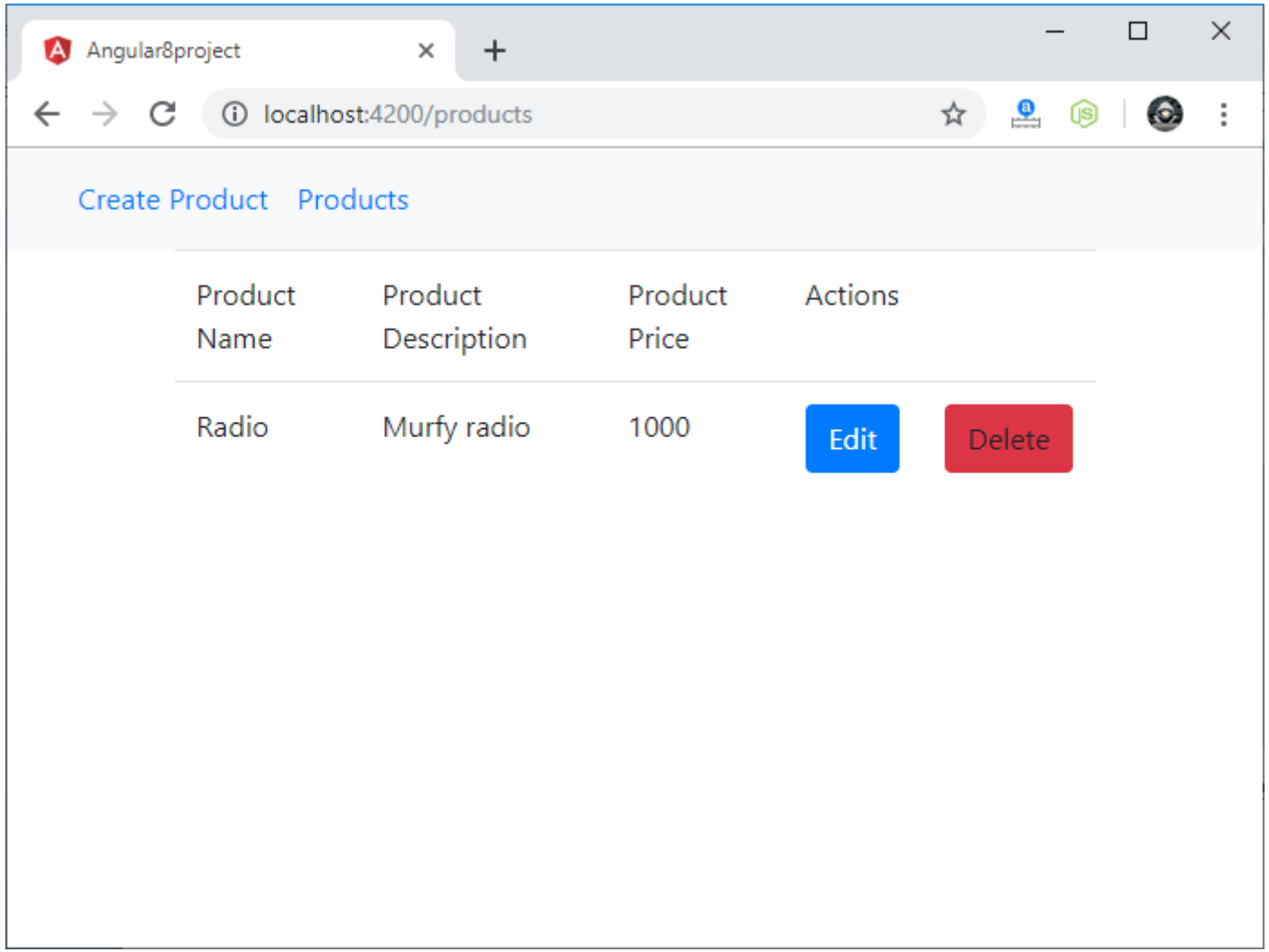
Now, we have to write a **deleteProduct() function** inside the **product-get.component.ts** file.

```
deleteProduct(id) {
  this.ps.deleteProduct(id).subscribe(res => {
    this.products.splice(id, 1);
  });
}
```

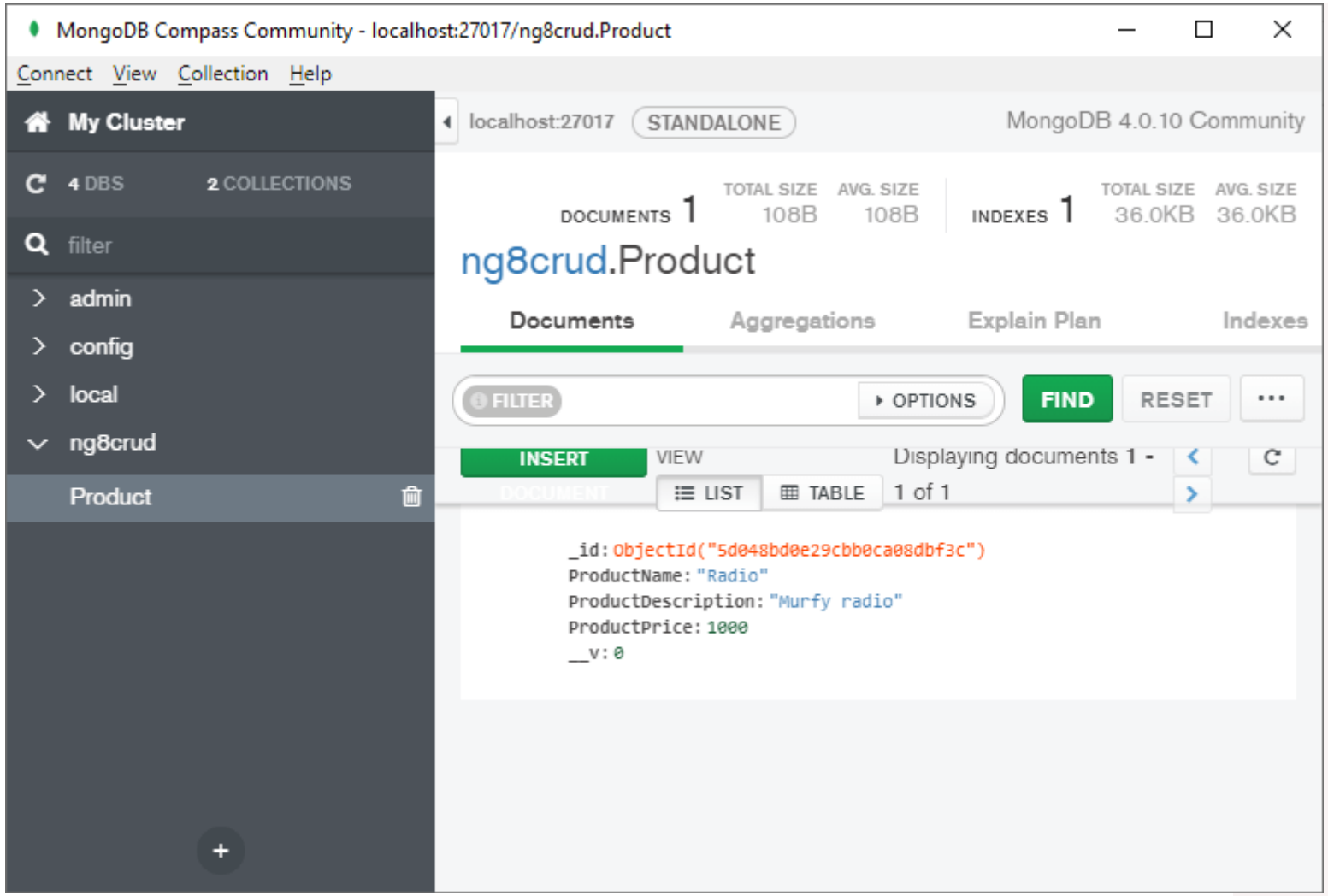
At last, create deleteProduct() function inside the product.service.ts file.

```
deleteProduct(id) {
  return this
          .http
          .get(`${this.uri}/delete/${id}`);
 }
```

DELETE operation is completed now. Let's check how it is working. Here, we delete the second item Stereo which we have edited first.



You can see that Stereo is deleted now. You can also verify it on MongoDB Compass Community GUI.



Angular 8 Tutorial Index

Angular 8 Tutorial

- Angular 8 Tutorial
- How to Upgrade Angular older versions to Angular 8

Angular 8 Data Binding

- Angular 8 Data Binding
- Property Binding
- String Interpolation

Angular + Spring

- CRUD Example
- File Upload Example
- Login & Logout Example

Next Topic How to upgrade Angular older versions to Angular 8

next →

Help Others, Please Share

Learn Latest Tutorials

UML Tutorial

UML

Artificial Neural Network Tutorial

ANN

ES6 Tutorial

ES6

Flutter Tutorial

Flutter

Selenium Python

Selenium Py

Firebase Tutorial

Firebase

Cobol Tutorial

Cobol

Ansible Tutorial

Ansible

Mockito Tutorial

Mockito

Talend Tutorial

Talend

Microsoft Azure Tutorial

Azure

Sharepoint Tutorial

SharePoint

Preparation

Aptitude

Aptitude

Logical Reasoning

Reasoning

Verbal Ability

Verbal A.

Interview Questions

Interview

Company Interview Questions

Company

Trending Technologies

Artificial Intelligence Tutorial

AWS Tutorial

AWS

Selenium tutorial

Selenium

Cloud tutorial

Cloud

Hadoop tutorial

Hadoop

AI

ReactJS Tutorial

ReactJS

Data Science Tutorial

D. Science

Angular 7 Tutorial

Angular 7

Blockchain Tutorial

Blockchain

Git Tutorial

Git

Machine Learning Tutorial

ML

DevOps Tutorial

DevOps

B.Tech / MCA

DBMS tutorial

DBMS

Data Structures tutorial

DS

DAA tutorial

DAA

Operating System tutorial

OS

Computer Network tutorial

C. Network

Compiler Design tutorial

Compiler D.

Computer Organization and Architecture

COA

Discrete Mathematics Tutorial

D. Math.

Ethical Hacking Tutorial

E. Hacking

Computer Graphics Tutorial

C. Graphics

Software Engineering Tutorial

Software E.

html tutorial

Web Tech.

Cyber Security tutorial

Cyber Sec.

Automata Tutorial

Automata

C Language tutorial

C

C++ tutorial

C++

Java tutorial

Java

.Net Framework tutorial

.Net

Python tutorial

Python

List of Programs

Programs

Control Systems tutorial

Control S.

Data Mining Tutorial

Data Mining

Javatpoint Services

JavaTpoint offers too many high quality services. Mail us on hr@javatpoint.com, to get more information about given services.

- Website Designing
- Website Development
- Java Development
- PHP Development
- WordPress
- Graphic Designing
- Logo
- Digital Marketing
- On Page and Off Page SEO
- PPC
- Content Development
- Corporate Training
- Classroom and Online Training
- Data Entry

Training For College Campus

JavaTpoint offers college campus training on Core Java, Advance Java, .Net, Android, Hadoop, PHP, Web Technology and Python. Please mail your requirement at hr@javatpoint.com.

Duration: 1 week to 2 week