

Angular 14

Phase 1 – Project Setup

Step 1: Install Node JS

Step 2: Install Visual Studio Code

Step 3: Install VS Plug-in “Angular Language”

Step 4: Choose a “Workspace Folder”

Step 5: Open Visual Studio and Select the workspace folder

Step 6: Open Terminal

Step 7: Install Angular CLI Version 14

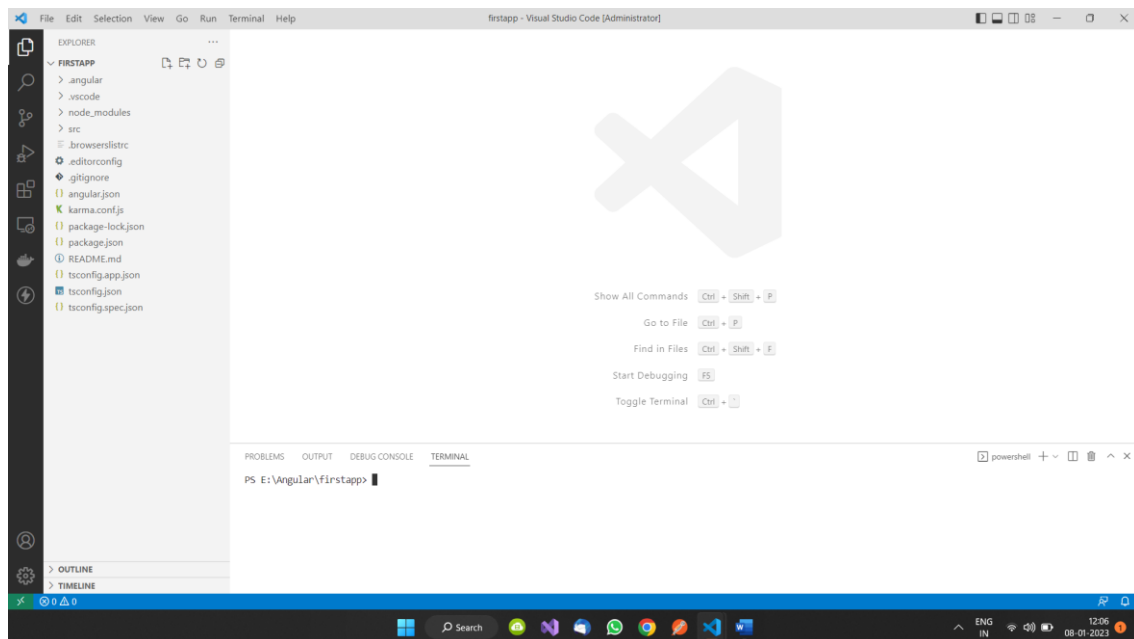
```
E:\Angular> npm i -g @angular/cli@14.2.0
```

Step 8: Create your first angular project

```
E:\Angular> ng new
```

Enter Project Name: firstapp

Step 9: Explore the project folder “firstapp” in VS Code. This is how the project structure looks like



Step 10: change to that folder in terminal

```
E:\Angular> cd .\firstapp\
```

```
E:\Angular\firstapp>
```

Phase 2 – Dashboard Home Page with Live Data

Step 1: Create component “Home” and place the static content of dashboard HTML into home.component.html

E:\Angular\firstapp> ng g c home

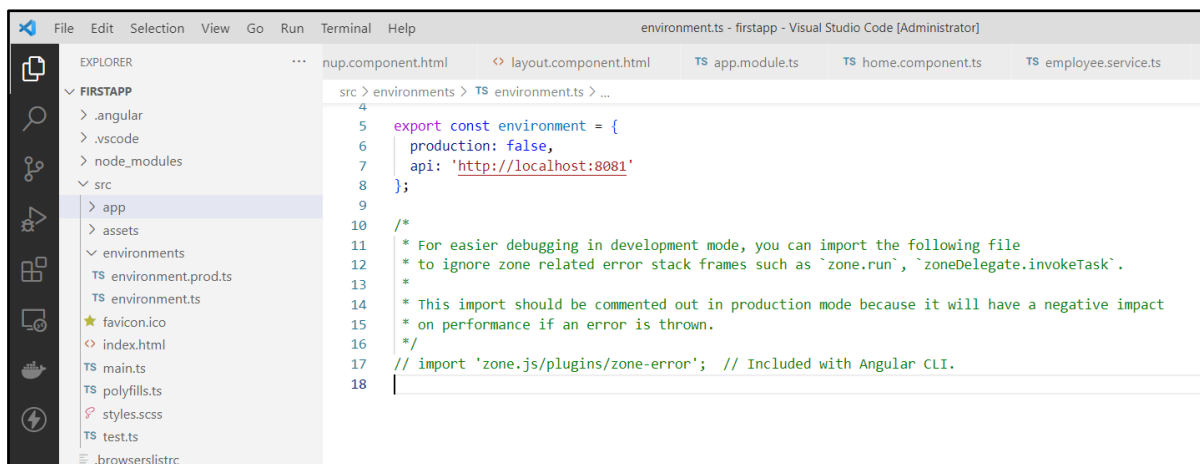


The screenshot shows the Visual Studio Code interface with the Angular CLI command `ng g c home` executed in the terminal. The file explorer on the left shows the project structure, including the `src/app/home` directory. The main editor displays the `home.component.html` file, which contains the following HTML structure:

```
1 <!-- start page title -->
2 <div class="row">
3   <div class="col-12">
4     <div class="page-title-box">
5       <h4 class="page-title">Dashboard</h4>
6       <div class="page-title-right">
7         <ol class="breadcrumb m-0">
8           <li class="breadcrumb-item"><a href="#">Shreyu</a></li>
9           <li class="breadcrumb-item active">Dashboard</li>
10        </ol>
11      </div>
12    </div>
13  </div>
14 </div>
15 <!-- end page title -->
16 <!-- end page title -->
17
18 <!-- stats + charts -->
19 <div class="row">
20   <div class="col-xl-3">
21     <div class="card">
22       <div class="card-body p-0">
23         <div class="p-3">
24           <div class="dropdown float-end">
25             <a href="#" class="dropdown-toggle arrow-none text-muted" data-bs-toggle="dropdown"
26               aria-expanded="false">
27               <i class="uil uil-ellipsis-v"></i>
28             </a>
29           </div>
30         </div>
31       </div>
32     </div>
33   </div>
34 </div>
```

The terminal at the bottom shows the command `ng g c home` and the output `Compiled successfully.`

Step 2: Here we are going to populate a table (Employee Details) in home page using data from server using Angular Service (“EmployeeService”). So first and foremost we are going to define server “Base URL” in “environment.ts”



The screenshot shows the Visual Studio Code interface with the `environment.ts` file open. The file contains the following configuration:

```
1 export const environment = {
2   production: false,
3   api: 'http://localhost:8081'
4 };
5
6 /*
7  * For easier debugging in development mode, you can import the following file
8  * to ignore zone related error stack frames such as `zone.run`, `zoneDelegate.invokeTask`.
9  *
10  * This import should be commented out in production mode because it will have a negative impact
11  * on performance if an error is thrown.
12  */
13 // import 'zone.js/plugins/zone-error'; // Included with Angular CLI.
```

Step 3: Now we need to create “EmployeeService” in “app/services” folder

E:\Angular\firstapp> ng g s services/Employee

Step 4: Before writing Service, we need to add the following modules to this application ***“app.module.ts”*** file to facilitate ***“HTTP/REST Communication”*** and also ***“Reactive Angular Forms”*** further while dealing with upcoming Forms in this application

```

8  import { LayoutComponent } from './layout/layout.component';
9  import { EmployeeListComponent } from './pages/employee-list/employee-list.component';
10 import { NewEmployeeComponent } from './pages/new-employee/new-employee.component';
11 import { NewDepartmentComponent } from './pages/new-department/new-department.component';
12 import { DepartmentListComponent } from './pages/department-list/department-list.component';
13 import { SidebarComponent } from './layout/sidebar/sidebar.component';
14 import { HttpClientModule } from '@angular/common/http'; //Form REST Communication
15 import { FormsModule, ReactiveFormsModule } from '@angular/forms'; //Reactive Form Generation
16
17 @NgModule({
18   declarations: [
19     AppComponent,
20     LoginComponent,
21     SignupComponent,
22     HomeComponent,
23     LayoutComponent,
24     EmployeeListComponent,
25     NewEmployeeComponent,
26     NewDepartmentComponent,
27     DepartmentListComponent,
28     SidebarComponent
29   ],
30   imports: [
31     BrowserModule,
32     AppRoutingModule,
33     HttpClientModule, //Form REST Communication
34     FormsModule, //Reactive Form Generation
35     ReactiveFormsModule //Reactive Form Generation
36   ],

```

Step 5: Now we are going to write REST communication for ***“Employee”*** module inside generated ***“employee.service.ts”***

```

1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { environment } from 'src/environments/environment';
4
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class EmployeeService {
9
10   constructor(private http: HttpClient) { } //Constructor Injection of HttpClient
11   getAll() { //Hitting server endpoint /employees which returns JSON of all employees
12     return this.http.get(environment.api + "/employees");
13   }
14   add(body: any) { //Hitting Post end point /employee and passing body params to the URI
15     return this.http.post(environment.api + "/employee", body);
16   }
17 }
18

```

In the above service, first we injected ***“HttpClient”*** module into the constructor. Then we wrote two service endpoints for ***“/employees”*** which is a GET and ***“/employee”*** which is a POST endpoints in server.

Step 6: Now we need to bind this **“EmployeeService”** to the page that want to load data from this service which is **“home.component.html”**. We need to follow this inside **“home.component.ts”** to bind data returned from the service to the above HTML file.

```

src > app > home > TS home.components.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { EmployeeService } from '../services/employee.service'; //Import Employee Service
3
4  @Component({
5    selector: 'app-home',
6    templateUrl: './home.component.html',
7    styleUrls: ['./home.component.scss']
8  })
9  export class HomeComponent implements OnInit {
10    employees: any[] = [] //Initiating array of employees to be sent to home component
11    constructor(private employeeService : EmployeeService) { } //Constructor Injection of EmployeeService
12    //Binding the response JSON of employeeService getAll API to the home component
13    ngOnInit(): void {
14      this.employeeService.getAll().subscribe((res: any) => {
15        this.employees = res;
16      }, err => {
17      });
18    }
19  }
20
21 }
22

```

Step 7: Now finally we need to load the **“employees”** array bind by **“EmployeeService”** to this page into the table using **“ngFor”**.

```

93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
<thead>
  <tr>
    <th scope="col">Employee#</th>
    <th scope="col">Full Name</th>
    <th scope="col">Job</th>
    <th scope="col">Salary</th>
    <th scope="col">Date of Join</th>
  </tr>
</thead>
<tbody>
  <tr *ngFor="let data of employees">
    <td>{{data?.empid}}</td>
    <td>{{data?.fullname}}</td>
    <td>{{data?.job}}</td>
    <td>{{data?.salary|currency: 'INR'}}</td>
    <td>{{data?.dateofjoin|date}}</td>
  </tr>
</tbody>
</table>
</div> <!-- end table-responsive-->
</div> <!-- end card-body-->
</div> <!-- end card-->

```

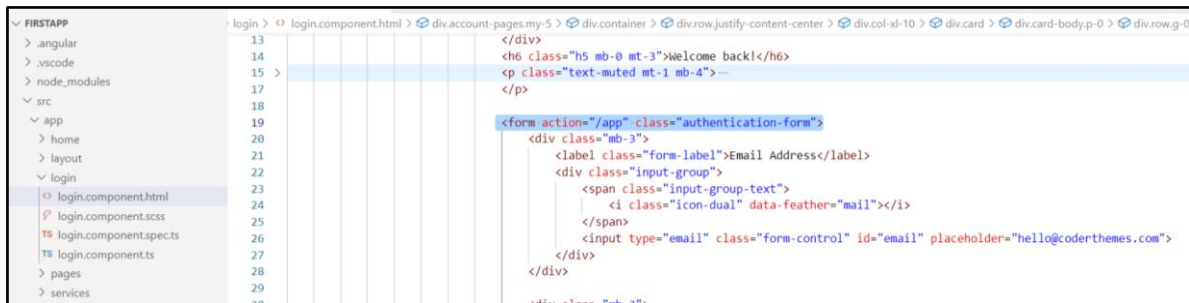
Step 8: Now Let’s check our application routes in **“app-routing.module.ts”**

```

src > app > TS app-routing.module.ts > routes > children
8  import { NewEmployeeComponent } from './pages/new-employee/new-employee.component';
9  import { LoginComponent } from './login/login.component';
10 import { SignupComponent } from './signup/signup.component';
11
12 const routes: Routes = [
13   {
14     path: '',
15     component: LoginComponent
16   },
17   {
18     path: 'signup',
19     component: SignupComponent
20   },
21   {
22     path: 'home',
23     component: HomeComponent
24   },
25   {
26     path: 'app',
27     component: LayoutComponent,
28     children: [
29       {
30         path: '',
31         redirectTo: 'home',
32         pathMatch: 'full'
33       },
34       {
35         path: 'home',
36         component: HomeComponent
37       },

```

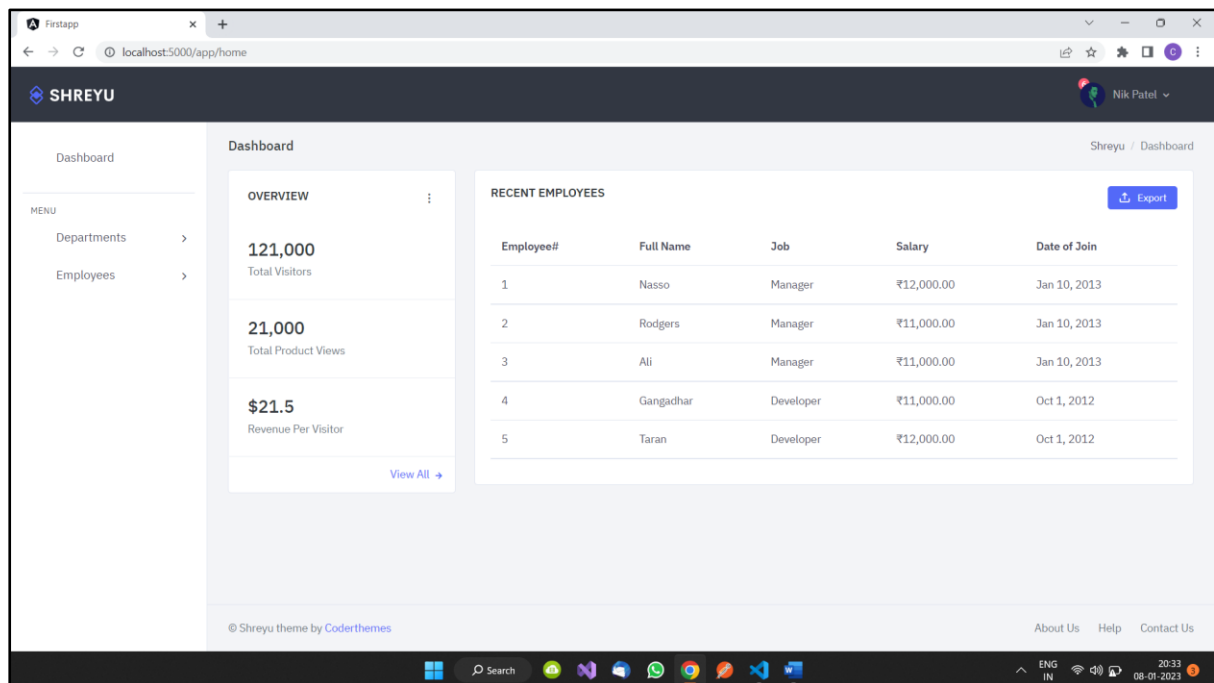
Step 9: That is it! We are ready to go. Now let us bind the home page to the **“Login”** page without any validation as of now.



```
13 </div>
14 <h6 class="h5 mb-0 mt-3">Welcome back!</h6>
15 <p class="text-muted mt-1 mb-4">--</p>
16 </div>
17 <form action="/app" class="authentication-form">
18 <div class="mb-3">
19 <label class="form-label">Email Address</label>
20 <div class="input-group">
21 <span class="input-group-text">
22 <i class="icon-dual data-feather="mail"></i>
23 </span>
24 <input type="email" class="form-control" id="email" placeholder="hello@coderthemes.com">
25 </div>
26 </div>
27 </div>
28 </div>
29 </div>
```

Step 10: Finally Let’ test the app. Run **“ng server”** and once login page opens click on submit which will redirect to the homepage directly.

E:\Angular\firstapp> ng serve --open --port 5000

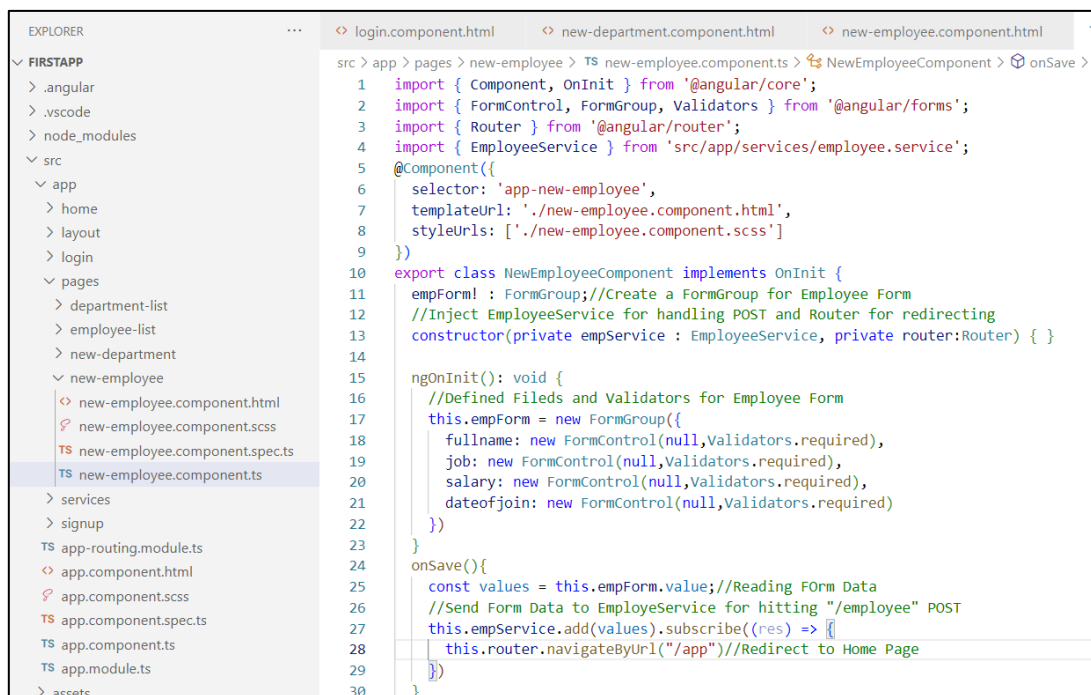


Phase 3 – Handling Form Input (POST)

Step 1: Let us create an input form component for storing employee details. We already wrote code for POST to “/employee” endpoint in the server. So let us create a component “new-employee”. From now onwards all sub page components let us put inside “app/pages” folder.

E:\Angular\firstapp> ng g c pages/new-employee

Step 2: First define “**FormGroup**” with name “**empForm**” and fields required for “**new-employee.component.html**”. Then Define a method “**onSave()**” for saving form data.



```
EXPLORER
  FIRSTAPP
    .angular
    .vscode
    node_modules
    src
      app
        home
        layout
        login
        pages
          department-list
          employee-list
          new-department
          new-employee
            new-employee.component.html
            new-employee.component.scss
            new-employee.component.spec.ts
            new-employee.component.ts
          services
          signup
        app-routing.module.ts
        app.component.html
        app.component.scss
        app.component.spec.ts
        app.component.ts
        app.module.ts
      assets

src > app > pages > new-employee > TS new-employee.component.ts > NewEmployeeComponent > onSave >
1  import { Component, OnInit } from '@angular/core';
2  import { FormControl, FormGroup, Validators } from '@angular/forms';
3  import { Router } from '@angular/router';
4  import { EmployeeService } from 'src/app/services/employee.service';
5  @Component({
6    selector: 'app-new-employee',
7    templateUrl: './new-employee.component.html',
8    styleUrls: ['./new-employee.component.scss']
9  })
10 export class NewEmployeeComponent implements OnInit {
11    empForm!: FormGroup; // Create a FormGroup for Employee Form
12    // Inject EmployeeService for handling POST and Router for redirecting
13    constructor(private empService: EmployeeService, private router: Router) {}
14
15    ngOnInit(): void {
16      // Defined Fields and Validators for Employee Form
17      this.empForm = new FormGroup({
18        fullname: new FormControl(null, Validators.required),
19        job: new FormControl(null, Validators.required),
20        salary: new FormControl(null, Validators.required),
21        dateofjoin: new FormControl(null, Validators.required)
22      });
23    }
24    onSave() {
25      const values = this.empForm.value; // Reading Form Data
26      // Send Form Data to EmployeeService for hitting "/employee" POST
27      this.empService.add(values).subscribe((res) => {
28        this.router.navigateByUrl("/app") // Redirect to Home Page
29      });
30    }
}
```

Step 3: Now map “**FormGroup: empForm**”, then it’s “**fields**” and also “**onSave()**” method to the form action



```
src > app > pages > new-employee > new-employee.component.html > div.row > div.col-lg-12.col-sm-12.col-md-12 > div.card > div.card-body > form
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58

<form [formGroup]="empForm" (ngSubmit)="onSave()">
  <div class="mb-3">
    <label class="form-label" for="employee-name">Name</label>
    <input type="text" id="employee-name" formControlName="fullname" class="form-control" placeholder="Name">
  </div>
  <div class="mb-3">
    <label class="form-label" for="employee-number">Employee No.</label>
    <input type="text" id="employee-number" formControlName="empname" class="form-control" placeholder="Employee">
  </div>
  <div class="mb-3">
    <label class="form-label" for="Join Date">Join Date</label>
    <input type="text" id="basic-datepicker" class="form-control" formControlName="dateofjoin" placeholder="Basic">
  </div>
  <div class="mb-3">
    <label class="form-label" for="employee-job">Job</label>
    <input type="text" id="employee-job" class="form-control" formControlName="job" placeholder="Job">
  </div>
  <div class="mb-3">
    <label class="form-label" for="employee-salary">Salary</label>
    <input type="number" min="0" id="employee-salary" class="form-control" formControlName="salary" placeholder="Salary">
  </div>
  <div class="d-flex justify-content-end">
    <button class="btn btn-primary">Submit</button>
  </div>
</form>
</div>
```

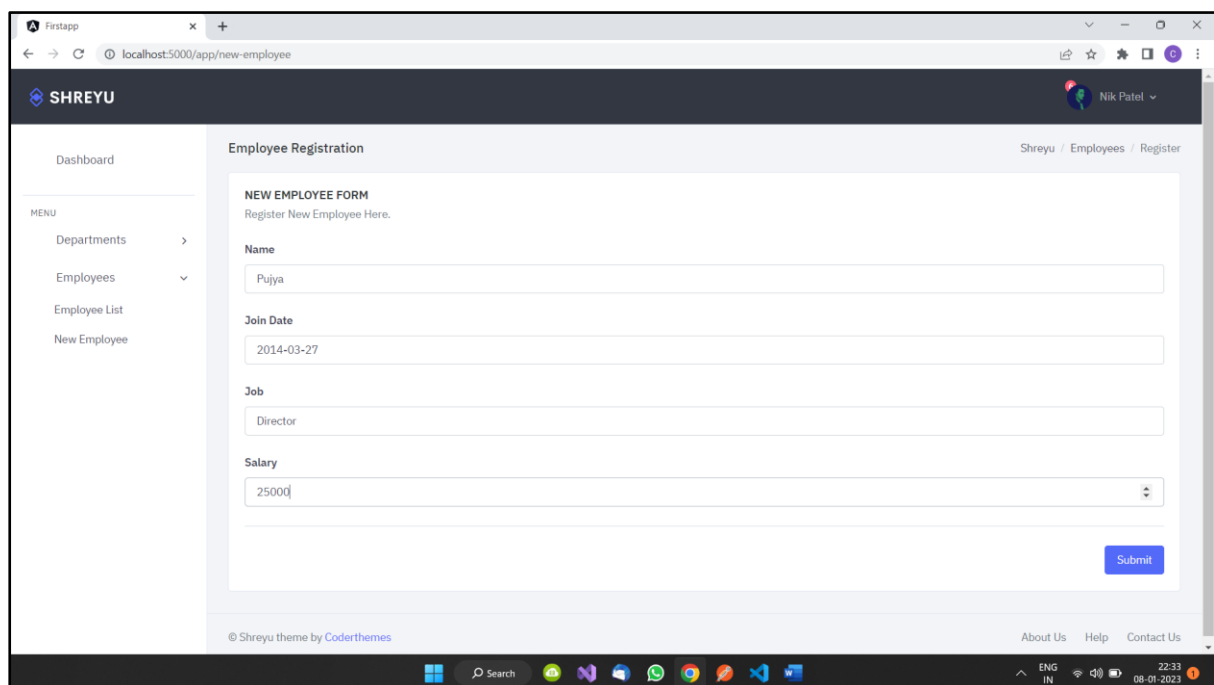
Step 4: Now, update anchor tags in **“sidebar.component.html”** to map to this **“new-employee.component.html”**

```
33 </ul>
34 </div>
35 </li>
36 </li>
37 <a href="#sidebarEmployees" data-bs-toggle="collapse">
38 <i data-feather="user"></i>
39 <span> Employees </span>
40 <span class="menu-arrow"></span>
41 </a>
42 <div class="collapse" id="sidebarEmployees">
43 <ul class="nav-second-level">
44 <li>
45 <a routerLink="/app/employee-list"
46 <i data-feather="list"></i> Employee List</a>
47 </li>
48 </li>
49 <li>
50 <a routerLink="/app/new-employee">
51 <i data-feather="user-plus"></i> New Employee</a>
52 </li>
53 </li>
54 </ul>
55 </div>
56 </li>
57 <!-- ./custom menu list -->
58 </ul>
59 </div>
```

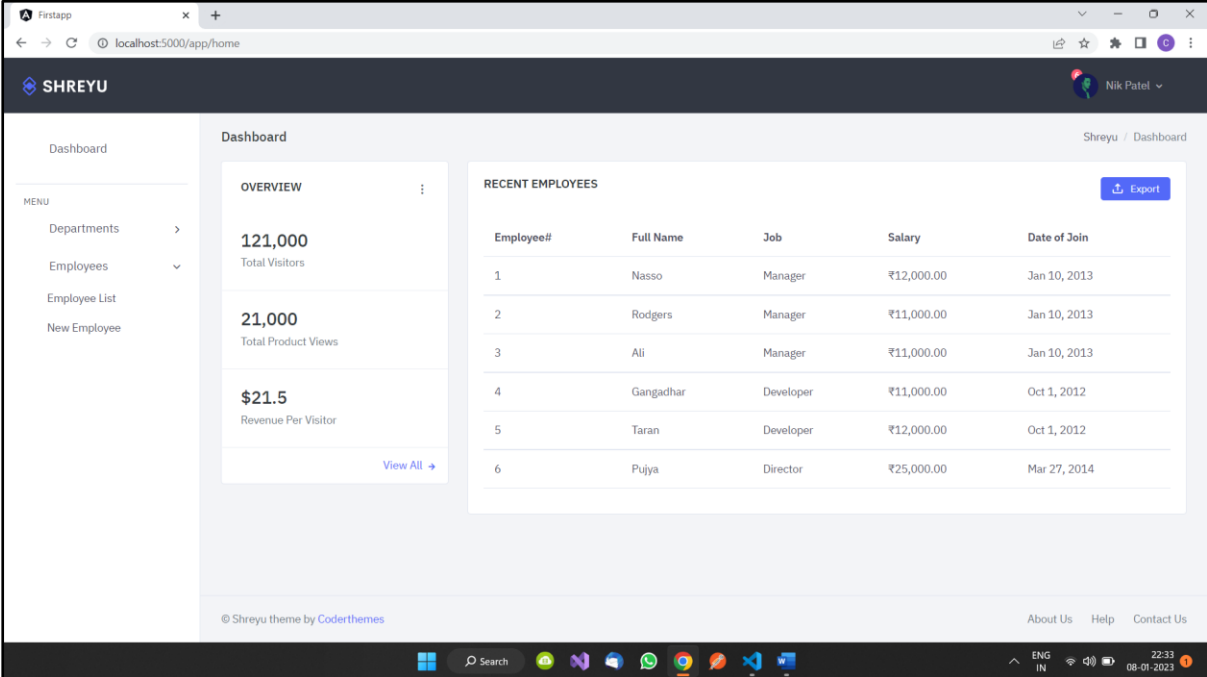
Step 5: We are ready with the new employee form. Let us build the app.

E:\Angular\firstapp> ng serve --open --port 5000

Again click on submit button in **“Login screen”** and go to the **“home page”**. Then Click on **“New Employee”** link to open the employee form.



After submission page will redirect to home page and we can see the record already saved.

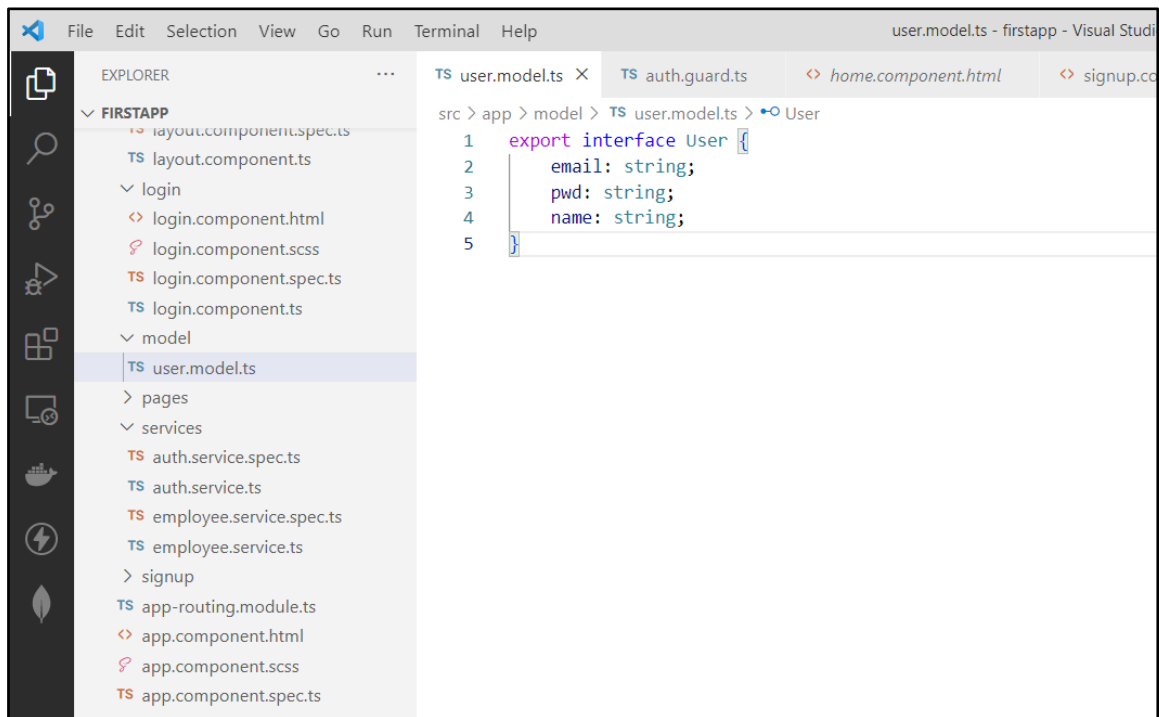


The screenshot displays the Shreyu dashboard interface. The left sidebar contains a menu with options: Dashboard, Departments, Employees, Employee List, and New Employee. The main content area is divided into two sections. The 'OVERVIEW' section on the left shows three key metrics: 121,000 Total Visitors, 21,000 Total Product Views, and \$21.5 Revenue Per Visitor, each with a 'View All' link. The 'RECENT EMPLOYEES' section on the right features a table with columns for Employee#, Full Name, Job, Salary, and Date of Join, listing six employees. An 'Export' button is located at the top right of this table. The footer includes copyright information, a search bar, and links for About Us, Help, and Contact Us.

Employee#	Full Name	Job	Salary	Date of Join
1	Nasso	Manager	₹12,000.00	Jan 10, 2013
2	Rodgers	Manager	₹11,000.00	Jan 10, 2013
3	Ali	Manager	₹11,000.00	Jan 10, 2013
4	Gangadhar	Developer	₹11,000.00	Oct 1, 2012
5	Taran	Developer	₹12,000.00	Oct 1, 2012
6	Pujiya	Director	₹25,000.00	Mar 27, 2014

Phase 4 – Adding Guard to Handle JWT Authentication

Step 1: First let us create a “model” with the name “user” for transporting “user data”.

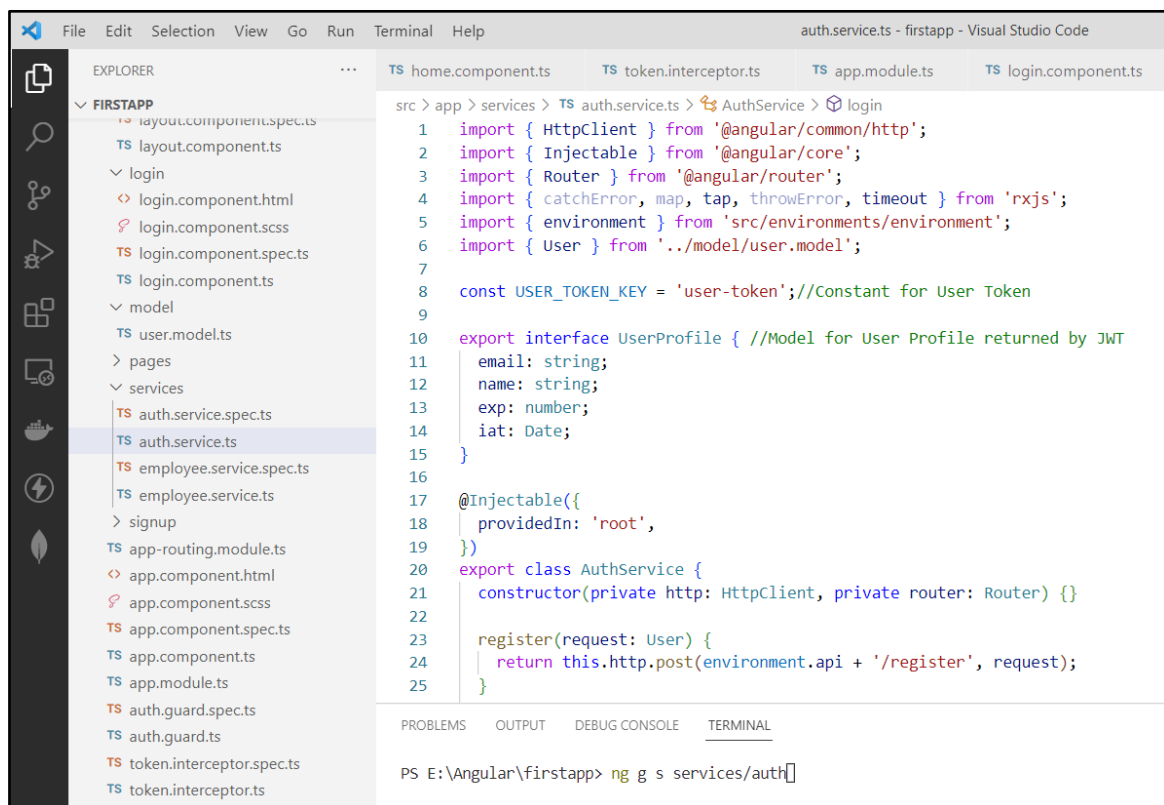


The screenshot shows the Visual Studio Code interface with the Explorer panel on the left and the editor on the right. The Explorer panel shows a project structure with a 'model' folder containing 'user.model.ts'. The editor shows the content of 'user.model.ts' with the following code:

```
src > app > model > TS user.model.ts > User
1  export interface User {
2      email: string;
3      pwd: string;
4      name: string;
5  }
```

Step 2: We are going to add a service for handling Authentication

E:\Angular\firstapp> ng g s services/auth



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left and the editor on the right. The Explorer panel shows a project structure with a 'services' folder containing 'auth.service.ts'. The editor shows the content of 'auth.service.ts' with the following code:

```
src > app > services > TS auth.service.ts > AuthService > login
1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { Router } from '@angular/router';
4  import { catchError, map, tap, throwError, timeout } from 'rxjs';
5  import { environment } from 'src/environments/environment';
6  import { User } from '../model/user.model';
7
8  const USER_TOKEN_KEY = 'user-token'; //Constant for User Token
9
10 export interface UserProfile { //Model for User Profile returned by JWT
11     email: string;
12     name: string;
13     exp: number;
14     iat: Date;
15 }
16
17 @Injectable({
18     providedIn: 'root',
19 })
20 export class AuthService {
21     constructor(private http: HttpClient, private router: Router) {}
22
23     register(request: User) {
24         return this.http.post(environment.api + '/register', request);
25     }
26 }
```

The terminal at the bottom shows the command: PS E:\Angular\firstapp> ng g s services/auth[]

The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left displaying the project structure. The main editor area shows the `login` method in `auth.service.ts`. The code implements a login function that sends a POST request to the API, sets session storage items for the token and its plain value, and navigates to the app page. Below the code, the terminal shows the command `ng g s services/auth` being executed.

```
src > app > services > TS auth.service.ts > AuthService > login
27 login(request: User) {
28   return this.http.post(environment.api + '/token', request).pipe(
29     timeout(3000),
30     tap((res: any) => {
31       sessionStorage.setItem(USER_TOKEN_KEY, JSON.stringify(res));
32       sessionStorage.setItem(USER_TOKEN_KEY + '_plain', res['token']);
33       this.router.navigateByUrl('/app');
34     })),
35   );
36 }
37
38 isLoggedIn(): boolean {
39   let isAuthenticated = false;
40   const token = this.getAccessToken();
41   if (token) {
42     isAuthenticated = true;
43   }
44   return isAuthenticated;
45 }
46
47 logout() {
48   sessionStorage.removeItem(USER_TOKEN_KEY);
49   this.router.navigateByUrl('/');
50 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS E:\Angular\firstapp> ng g s services/auth

The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left. The main editor area shows the `getAccessToken` and `getUserProfile` methods in `auth.service.ts`. The `getAccessToken` method retrieves the token from session storage and returns it or null. The `getUserProfile` method uses the token to fetch the user profile and returns it as a JSON object.

```
src > app > services > TS auth.service.ts > AuthService > login
51
52 getAccessToken() {
53   const token = JSON.parse(sessionStorage.getItem(USER_TOKEN_KEY) || '{}');
54   return token['token'] || null;
55 }
56
57 getUserProfile(): UserProfile {
58   let token: string = this.getAccessToken();
59
60   let tokenArr = token.split('.');
61   const profile = atob(tokenArr[1]);
62
63   console.log(profile);
64   return JSON.parse(profile) as any;
65 }
66 }
```

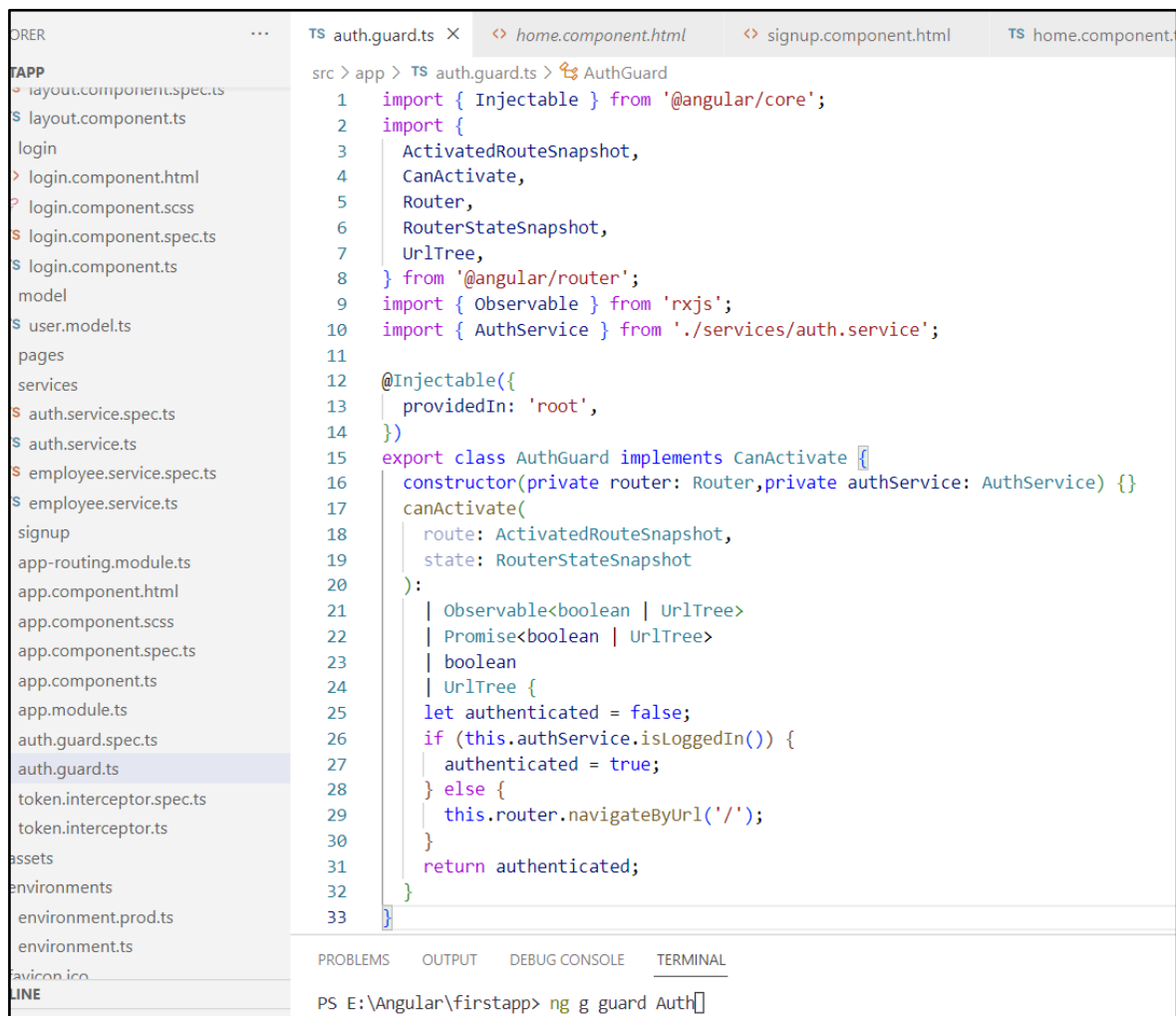
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS E:\Angular\firstapp> ng g s services/auth

Step 3: We are going to create a guard with the name **“Auth”** for globally handling **“JWT Authentication”** for all the routes.

E:\Angular\firstapp> ng g guard Auth

Step 4: We can see *“auth-guard.ts”* create after the above command



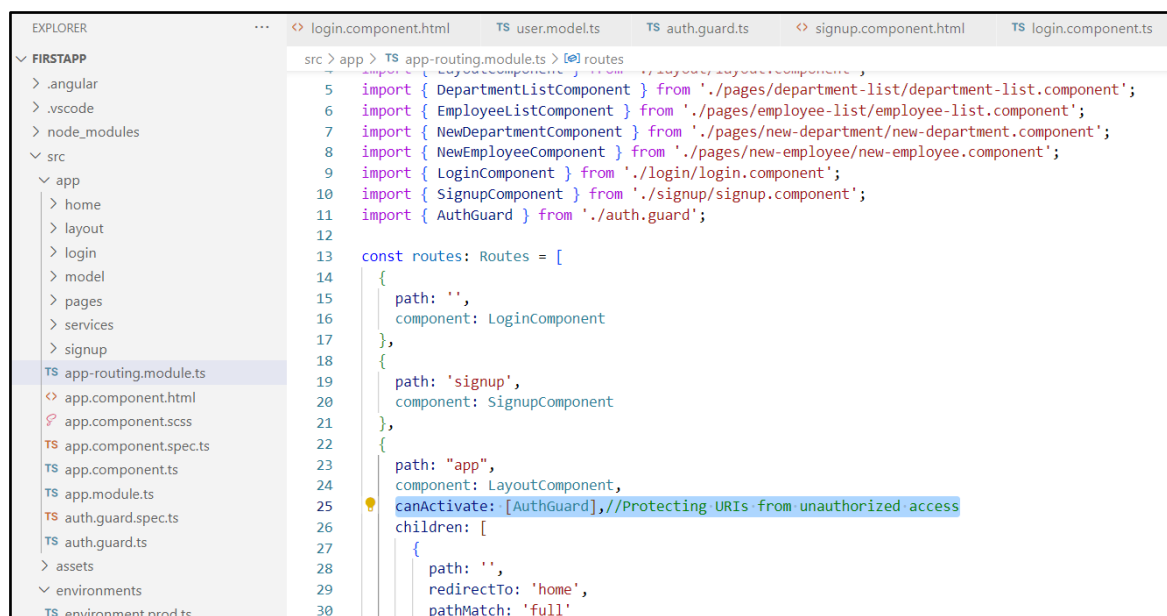
The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a file tree for a project named 'FIRSTAPP'. The file 'auth.guard.ts' is highlighted in the list. The main editor area shows the content of 'auth.guard.ts', which is a TypeScript file defining an 'AuthGuard' class. The class is decorated with '@Injectable' and implements the 'CanActivate' interface. It has a constructor that takes 'router' and 'authService' as parameters. The 'canActivate' method checks if the user is logged in using 'authService.isLoggedIn()' and either allows access or redirects to the home page. The bottom status bar shows the command prompt 'PS E:\Angular\firstapp> ng g guard Auth'.

```
1 import { Injectable } from '@angular/core';
2 import {
3   ActivatedRouteSnapshot,
4   CanActivate,
5   Router,
6   RouterStateSnapshot,
7   UrlTree,
8 } from '@angular/router';
9 import { Observable } from 'rxjs';
10 import { AuthService } from '../services/auth.service';
11
12 @Injectable({
13   providedIn: 'root',
14 })
15 export class AuthGuard implements CanActivate {
16   constructor(private router: Router, private authService: AuthService) {}
17   canActivate(
18     route: ActivatedRouteSnapshot,
19     state: RouterStateSnapshot
20   ):
21     | Observable<boolean | UrlTree>
22     | Promise<boolean | UrlTree>
23     | boolean
24     | UrlTree {
25     let authenticated = false;
26     if (this.authService.isLoggedIn()) {
27       authenticated = true;
28     } else {
29       this.router.navigateByUrl('/');
30     }
31     return authenticated;
32   }
33 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS E:\Angular\firstapp> ng g guard Auth

Step 5: Now let us protect our routes in *“app-routing.module.ts”* with the help of this *“AuthGuard”* from unauthorized access.



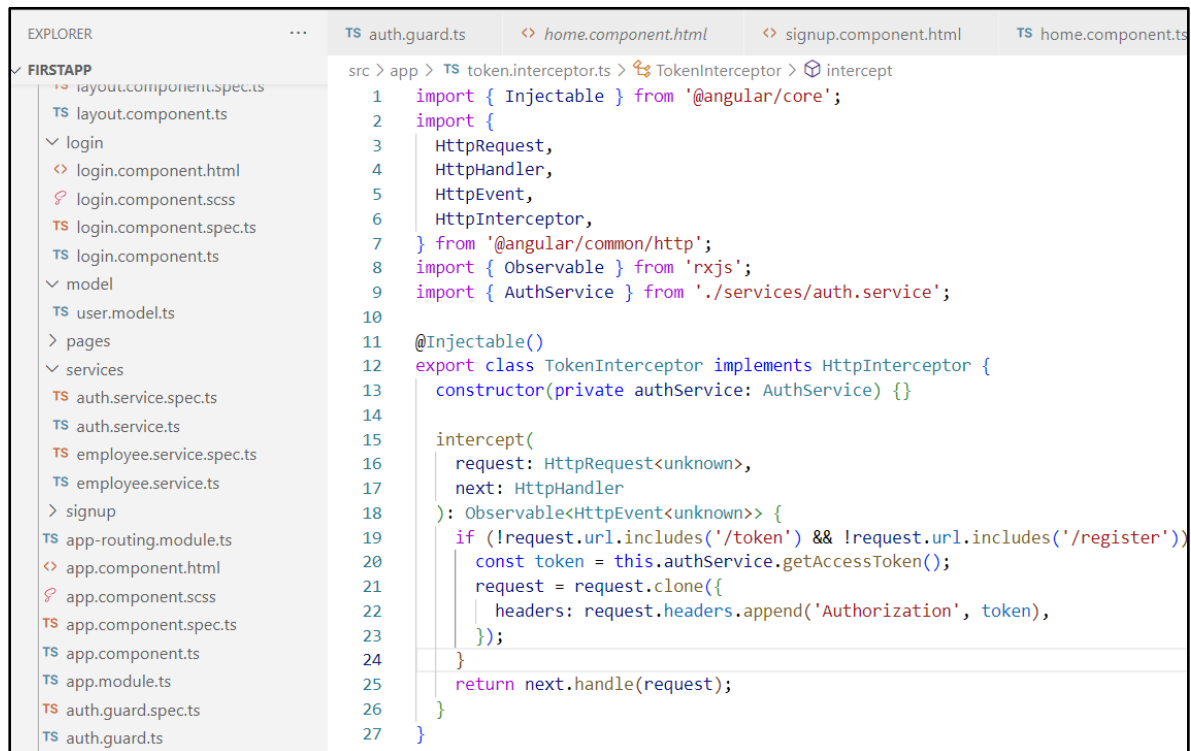
The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows the file tree with 'app-routing.module.ts' selected. The main editor displays the code for 'app-routing.module.ts'. It imports various components and the 'AuthGuard'. The 'routes' array is defined with three routes: a root route pointing to 'LoginComponent', a 'signup' route pointing to 'SignupComponent', and an 'app' route pointing to 'LayoutComponent'. The 'app' route is configured with 'canActivate: [AuthGuard]' and a 'children' array containing a root route that redirects to 'home' if not authenticated. A tooltip is visible over the 'canActivate' property, displaying the text: 'Protecting URIs from unauthorized access'.

```
5 import { DepartmentListComponent } from './pages/department-list/department-list.component';
6 import { EmployeeListComponent } from './pages/employee-list/employee-list.component';
7 import { NewDepartmentComponent } from './pages/new-department/new-department.component';
8 import { NewEmployeeComponent } from './pages/new-employee/new-employee.component';
9 import { LoginComponent } from './login/login.component';
10 import { SignupComponent } from './signup/signup.component';
11 import { AuthGuard } from '../auth.guard';
12
13 const routes: Routes = [
14   {
15     path: '',
16     component: LoginComponent
17   },
18   {
19     path: 'signup',
20     component: SignupComponent
21   },
22   {
23     path: 'app',
24     component: LayoutComponent,
25     canActivate: [AuthGuard], //Protecting URIs from unauthorized access
26     children: [
27       {
28         path: '',
29         redirectTo: 'home',
30         pathMatch: 'full'
31       }
26
```

Step 6: Now Let's put Token generation part separately in **"Token Interceptor"**

E:\Angular\firstapp>ng g interceptor

Enter the name for interceptor: **token**

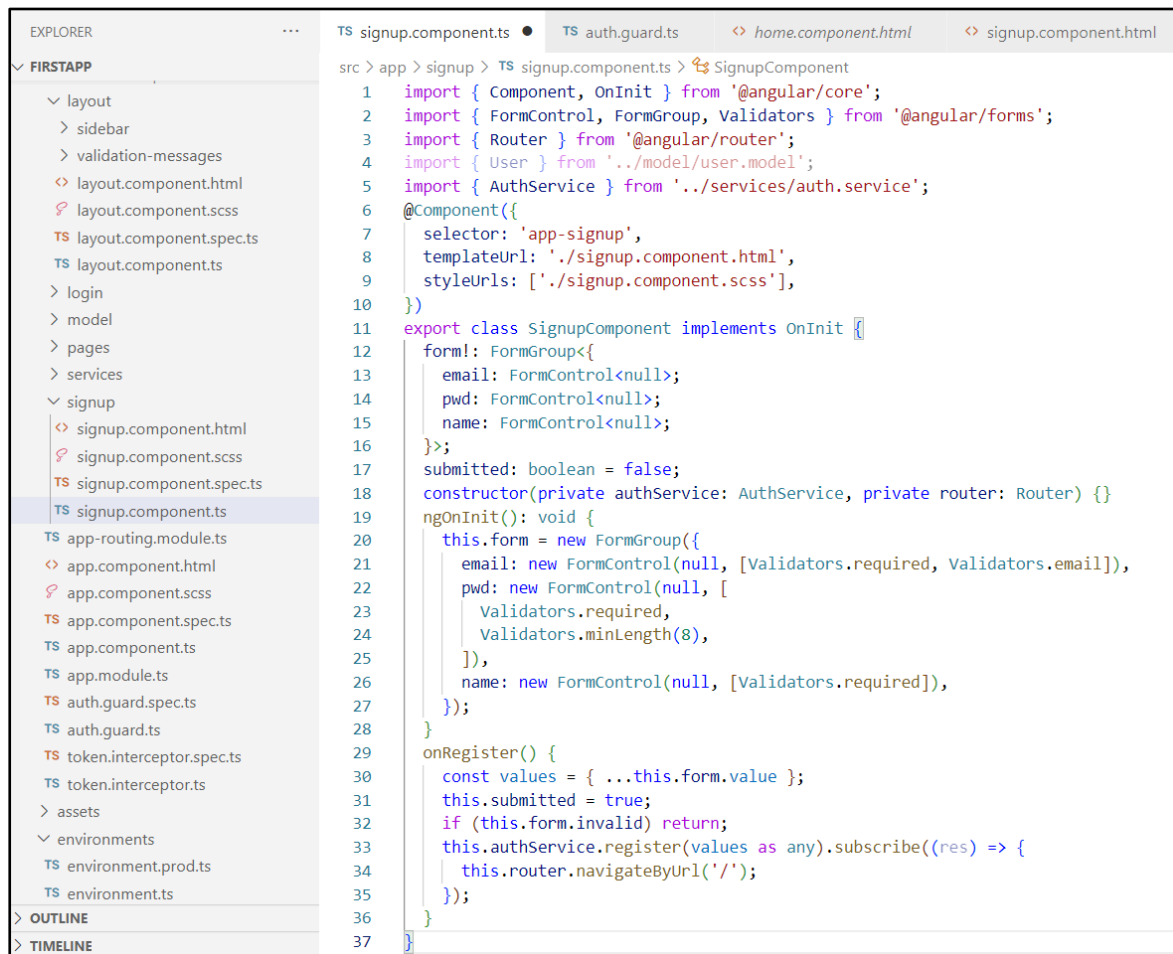


```
src > app > TS token.interceptor.ts > TokenInterceptor > intercept
1  import { Injectable } from '@angular/core';
2  import {
3    HttpRequest,
4    HttpHandler,
5    HttpEvent,
6    HttpInterceptor,
7  } from '@angular/common/http';
8  import { Observable } from 'rxjs';
9  import { AuthService } from './services/auth.service';
10
11  @Injectable()
12  export class TokenInterceptor implements HttpInterceptor {
13    constructor(private authService: AuthService) {}
14
15    intercept(
16      request: HttpRequest<unknown>,
17      next: HttpHandler
18    ): Observable<HttpEvent<unknown>> {
19      if (!request.url.includes('/token') && !request.url.includes('/register'))
20        const token = this.authService.getAccessToken();
21        request = request.clone({
22          headers: request.headers.append('Authorization', token),
23        });
24      }
25      return next.handle(request);
26    }
27  }
```

Note: Here we can see that we exempted **"/register"** the Signup API and **"/token"** the login API from **"AuthGuard"** as well as **"Interceptor"**

Phase 5 – Adding Signup Page

Step 1: Add the following code for adding **“Reactive Form”** and hitting **“/register”** endpoint using **“Auth-Service”**



```
src > app > signup > TS signup.component.ts > SignupComponent
1  import { Component, OnInit } from '@angular/core';
2  import { FormControl, FormGroup, Validators } from '@angular/forms';
3  import { Router } from '@angular/router';
4  import { User } from '../model/user.model';
5  import { AuthService } from '../services/auth.service';
6  @Component({
7    selector: 'app-signup',
8    templateUrl: './signup.component.html',
9    styleUrls: ['./signup.component.scss'],
10 })
11 export class SignupComponent implements OnInit {
12   form!: FormGroup<
13     email: FormControl<null>;
14     pwd: FormControl<null>;
15     name: FormControl<null>;
16   >;
17   submitted: boolean = false;
18   constructor(private authService: AuthService, private router: Router) {}
19   ngOnInit(): void {
20     this.form = new FormGroup({
21       email: new FormControl(null, [Validators.required, Validators.email]),
22       pwd: new FormControl(null, [
23         Validators.required,
24         Validators.minLength(8),
25       ]),
26       name: new FormControl(null, [Validators.required]),
27     });
28   }
29   onRegister() {
30     const values = { ...this.form.value };
31     this.submitted = true;
32     if (this.form.invalid) return;
33     this.authService.register(values as any).subscribe((res) => {
34       this.router.navigateByUrl('/');
35     });
36   }
37 }
```

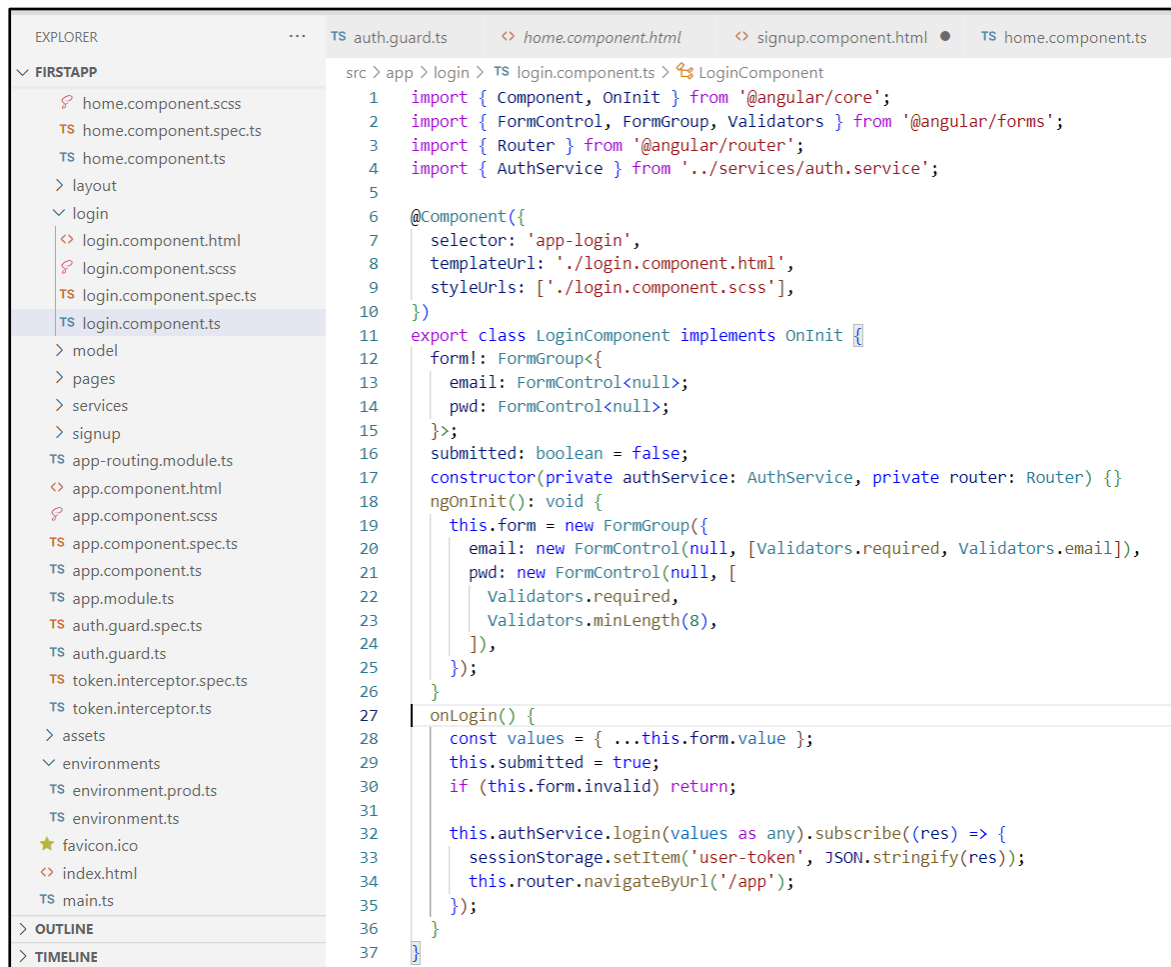
Step 2: Do the Following changes in **“signup.component.ts”** for defining **“Reactive Form”**



```
18 <form [formGroup]="form" class="authentication-form" (ngSubmit)="onRegister()">
19   <div class="mb-3">
20     <label class="form-label">Name</label>
21     <div class="input-group">
22       <span class="input-group-text">
23         <i class="icon-dual" data-feather="user"></i>
24       </span>
25       <input type="text" formControlName="name" class="form-control" id="name" placeholder="Your full name">
26     </div>
27     <app-validation-messages *ngIf="submitted" [formGroup]="form" controlName="name"></app-validation-messages>
28   </div>
29   <div class="mb-3">
30     <label class="form-label">Email Address</label>
31     <div class="input-group">
32       <span class="input-group-text">
33         <i class="icon-dual" data-feather="mail"></i>
34       </span>
35       <input type="email" formControlName="email" class="form-control" id="email" placeholder="hello@codethemes.com">
36     </div>
37     <app-validation-messages *ngIf="submitted" [formGroup]="form" controlName="email"></app-validation-messages>
38   </div>
39   <div class="mb-3">
40     <label class="form-label">Password</label>
41     <div class="input-group">
42       <span class="input-group-text">
43         <i class="icon-dual" data-feather="lock"></i>
44       </span>
45       <input type="password" formControlName="pwd" class="form-control" id="password" placeholder="Enter your password">
46     </div>
47     <app-validation-messages *ngIf="submitted" [formGroup]="form" controlName="pwd"></app-validation-messages>
48   </div>
```

Phase 6 – Adding Login Page and Bringing Token Params to Layout

Step 1: Add the following code for adding “**Reactive Form**” for Login and hitting “**/token**” endpoint using “**Auth-Service**”



```
src > app > login > TS login.component.ts > LoginComponent
1  import { Component, OnInit } from '@angular/core';
2  import { FormControl, FormGroup, Validators } from '@angular/forms';
3  import { Router } from '@angular/router';
4  import { AuthService } from '../services/auth.service';
5
6  @Component({
7    selector: 'app-login',
8    templateUrl: './login.component.html',
9    styleUrls: ['./login.component.scss'],
10  })
11  export class LoginComponent implements OnInit {
12    form!: FormGroup<{
13      email: FormControl<null>;
14      pwd: FormControl<null>;
15    }>;
16    submitted: boolean = false;
17    constructor(private authService: AuthService, private router: Router) {}
18    ngOnInit(): void {
19      this.form = new FormGroup({
20        email: new FormControl(null, [Validators.required, Validators.email]),
21        pwd: new FormControl(null, [
22          Validators.required,
23          Validators.minLength(8),
24        ]),
25      });
26    }
27    onLogin() {
28      const values = { ...this.form.value };
29      this.submitted = true;
30      if (this.form.invalid) return;
31
32      this.authService.login(values as any).subscribe((res) => {
33        sessionStorage.setItem('user-token', JSON.stringify(res));
34        this.router.navigateByUrl('/app');
35      });
36    }
37  }
```