

Module 10: Authentication with JWT and Security

Demo Document 1: Create Login and registration form to use JWT authentication

edureka!

edureka!

© Brain4ce Education Solutions Pvt. Ltd.

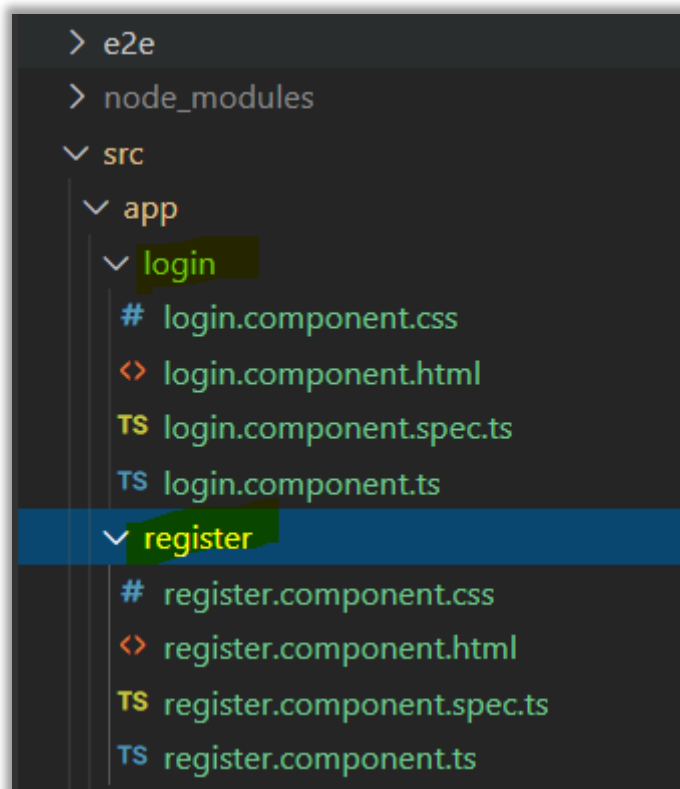
Create Login and registration form to use JWT authentication

In this demo, we will see how to create Login and registration form and store user data (registration form data) using fakebackend provider

Step 1: Create new project using command 'ng new Module10Demo1' .

Step 2: Create new login and register components using the command

'ng g c login and ng g c register', it will create reactive component



Step 3- Open index.html and add bootstrap reference as below.

```

index.html
src > index.html > html > head > link
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>Module8Demo1</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9   <link href="//cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet" />
10 </head>
11 <body>
12   <app-root></app-root>
13 </body>
14 </html>
15

```

Step 4 – Open login.component.html and add below code .

```

<h2>Login</h2>
<form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
  <div class="form-group">
    <label for="username">Username</label>
    <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
    <div *ngIf="submitted && f.username.errors" class="invalid-feedback">
      <div *ngIf="f.username.errors.required">Username is required</div>
    </div>
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" formControlName="password" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }" />
    <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
      <div *ngIf="f.password.errors.required">Password is required</div>
    </div>
  </div>
  <div class="form-group">
    <button [disabled]="loading" class="btn btn-primary">
      <span *ngIf="loading" class="spinner-border spinner-border-sm mr-
1"></span>
      Login
    </button>
    <a routerLink="/register" class="btn btn-link">Register</a>
  </div>
</form>

```

Step 5- Open login.component.ts file and add below code

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  loginForm: FormGroup;
  loading = false;
  submitted = false;
  returnUrl: string;

  constructor(private formBuilder: FormBuilder,
    private route: ActivatedRoute,
    private router: Router) { }

  ngOnInit() {
    this.loginForm = this.formBuilder.group({
      username: ['', Validators.required],
      password: ['', Validators.required]
    });

    // get return url from route parameters or default to '/'
    this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
  }

  get f() { return this.loginForm.controls; }

  onSubmit() {
    this.submitted = true;
  }
}
```

Step 6- As we are using reactive form, import ReactiveFormsModule into app.module.ts

```
src > app > TS app.module.ts > AppModule
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { ReactiveFormsModule } from '@angular/forms';
4
5  import { AppRoutingModule } from './app-routing.module';
6  import { AppComponent } from './app.component';
7  import { LoginComponent } from './login/login.component';
8  import { RegisterComponent } from './register/register.component';
9
10 @NgModule({
11   declarations: [
12     AppComponent,
13     LoginComponent,
14     RegisterComponent
15   ],
16   imports: [
17     BrowserModule,
18     ReactiveFormsModule,
19     AppRoutingModule
20   ],
21   providers: [],
22   bootstrap: [AppComponent]
23 })
24 export class AppModule { }
25
```

Step 7- open app-routing.module.ts file and add below code.

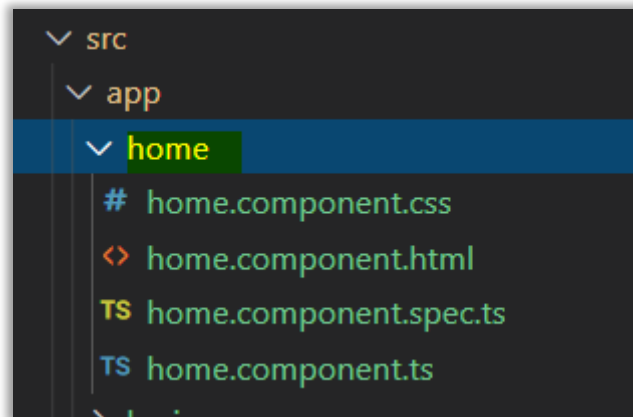
```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';

const routes: Routes = [
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

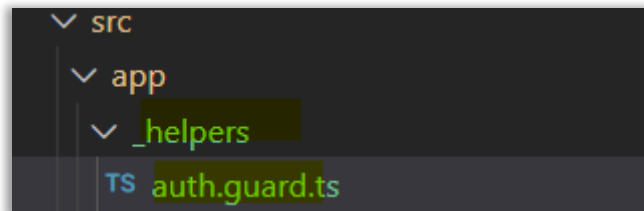
Step 8- Similarly copy html and .ts code to register.component.html and register.component.ts file

Step 9- Create home component using command 'ng g c home' as below

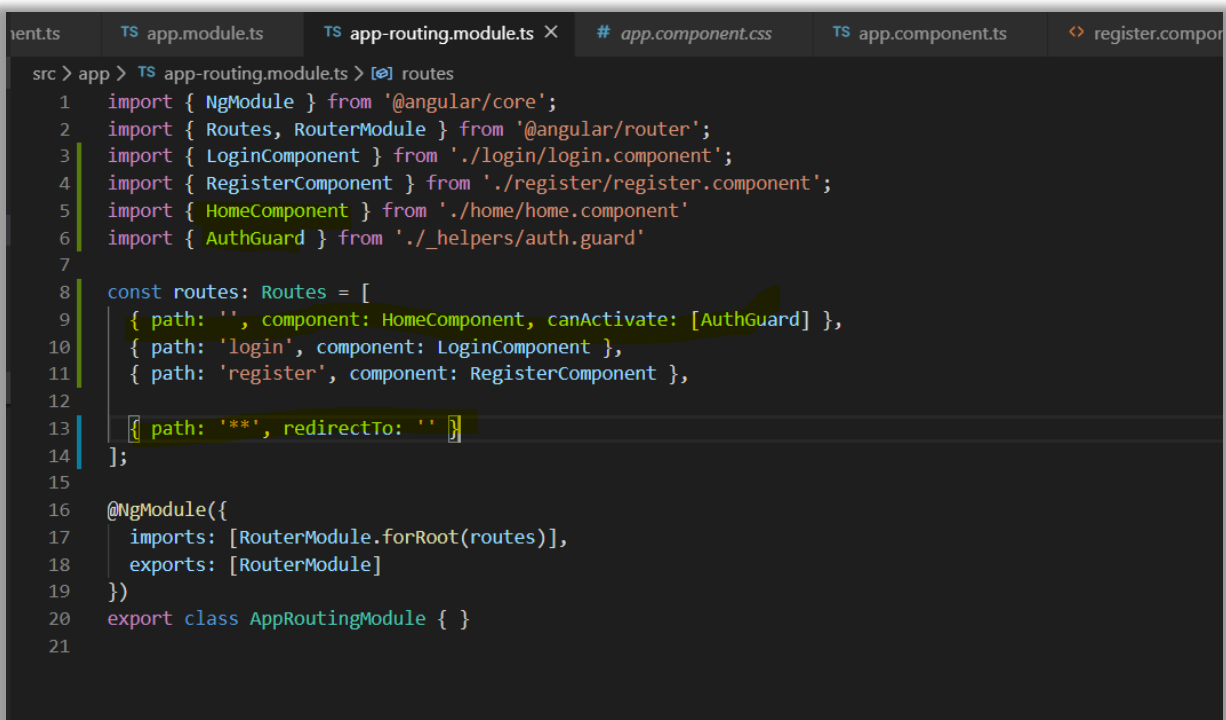


Step 10 – Right click on src/app folder and create new folder as '_helpers' to add helper files

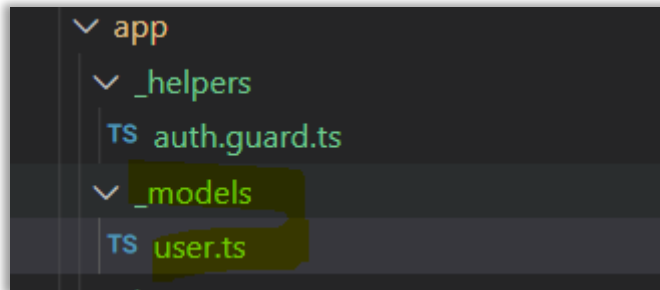
Step 11 – Right click on _helpers folder and create new file as 'auth.guard.ts'



Step 11 – Open app-routing.module.ts file and import home and authguard as below and add to route constants.

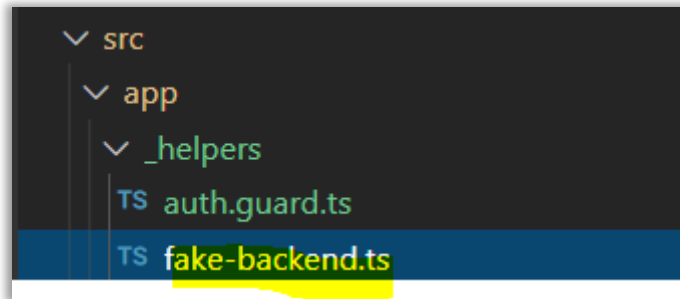


Step 12 – Right click app folder and create new folder _models, inside _models folder create new file called user.ts (user.ts will be used as model to add user information)



```
src > app > _models > TS user.ts > User
1 export class User {
2     id: number;
3     username: string;
4     password: string;
5     firstName: string;
6     lastName: string;
7     token: string;
8 }
```

Step 13 – Right click `src/app/_helpers` folder and create `fake-backend.ts` file to use as fake backend to store user information.



Code: `fake-backend.ts` file

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpResponse, HttpHandler, HttpEvent, HttpInterceptor, HTTP_INTERCEPTORS } from '@angular/common/http';
import { Observable, of, throwError } from 'rxjs';
import { delay, mergeMap, materialize, dematerialize } from 'rxjs/operators';

// array in local storage for registered users
let users = JSON.parse(localStorage.getItem('users')) || [];

@Injectable()
export class FakeBackendInterceptor implements HttpInterceptor {
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const { url, method, headers, body } = request;

    // wrap in delayed observable to simulate server api call
    return of(null)
      .pipe(mergeMap(handleRoute))
      .pipe(materialize()) // call materialize and dematerialize to ensure delay even if an error is thrown (https://github.com/Reactive-Extensions/RxJS/issues/648)
      .pipe(delay(500))
      .pipe(dematerialize());

    function handleRoute() {
      switch (true) {
        case url.endsWith('/users/register') && method === 'POST':
          return register();
        default:
          // pass through any requests not handled above
          return next.handle(request);
      }
    }
  }
}
```



```

    // route functions

    function register() {
        const user = body

        if (users.find(x => x.username === user.username)) {
            return error('Username "' + user.username + '" is already taken')
        }

        user.id = users.length ? Math.max(...users.map(x => x.id)) + 1 : 1;
        users.push(user);
        localStorage.setItem('users', JSON.stringify(users));

        return ok();
    }

    // helper functions

    function ok(body?) {
        return of(new HttpResponse({ status: 200, body }));
    }

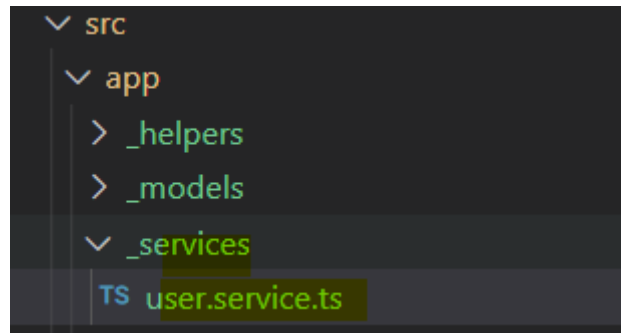
    function error(message) {
        return throwError({ error: { message } });
    }

}

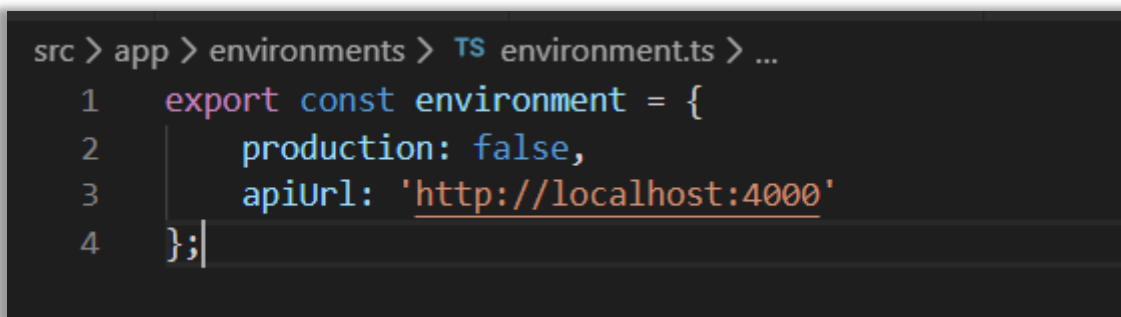
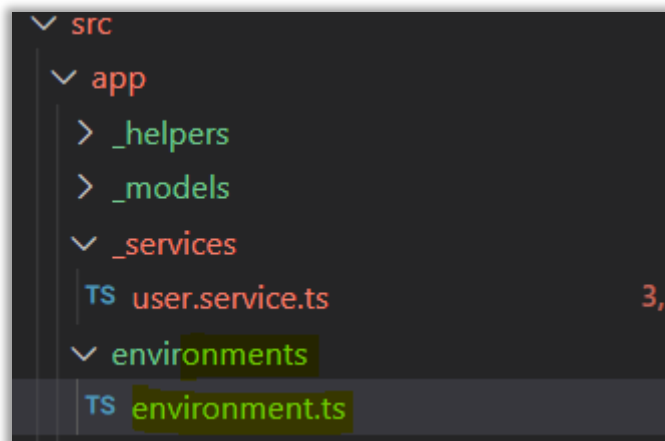
export const fakeBackendProvider = {
    // use fake backend in place of Http service for backend-less development
    provide: HTTP_INTERCEPTORS,
    useClass: FakeBackendInterceptor,
    multi: true
};

```

Step 14 – Right click on app folder and create `_services` folder, inside `_services` folder create new file called `user.service.ts` as below

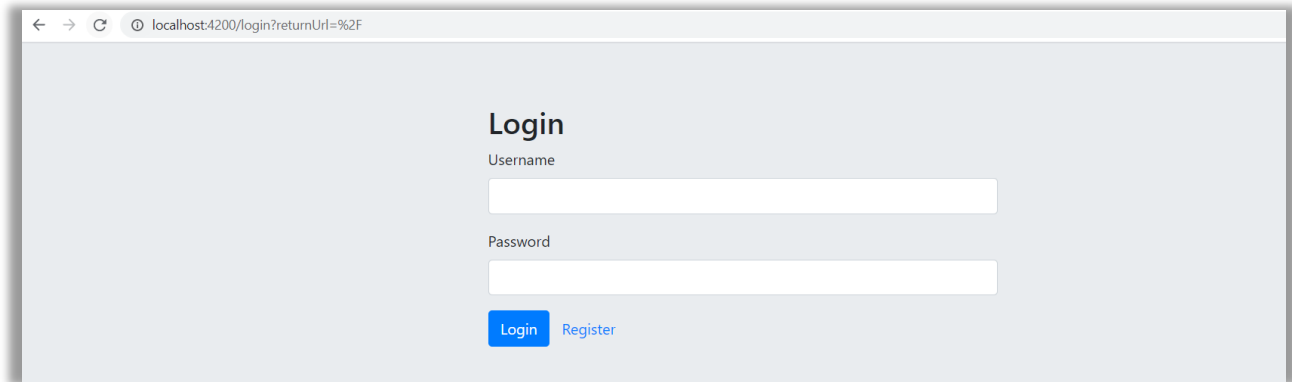


Step 15 – Right click on the app folder and create environment folder to store development environment config as below .



Step 15 – Run app using `ng serve` command

This will open below window.



← → ↻ localhost:4200/login?returnUrl=%2F

Login

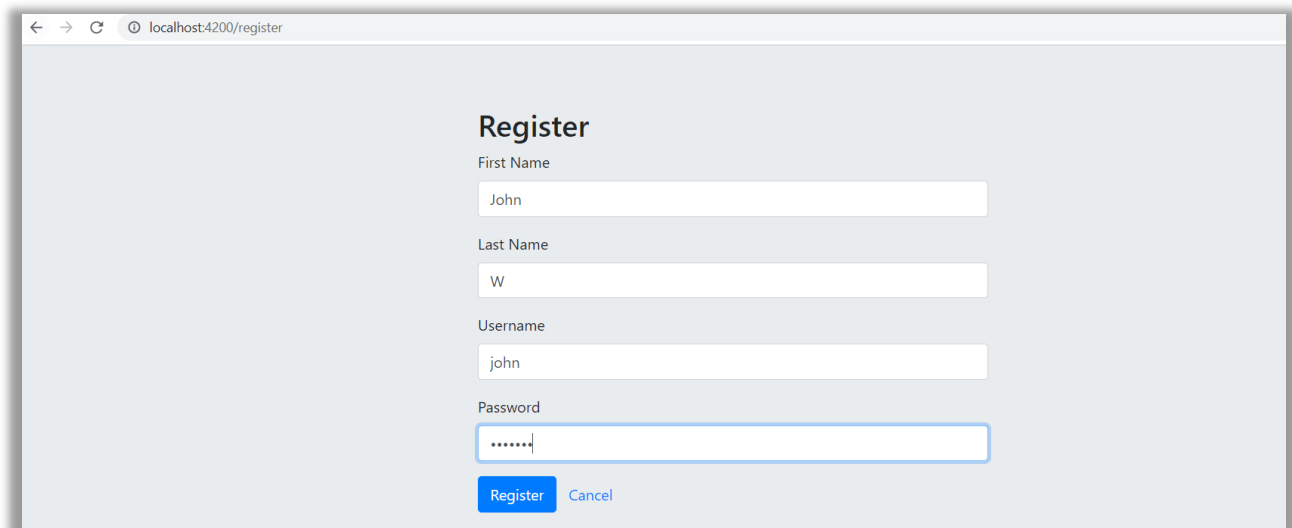
Username

Password

[Login](#) [Register](#)

Now click on Register link at the bottom.

Add user information and click Register



← → ↻ localhost:4200/register

Register

First Name

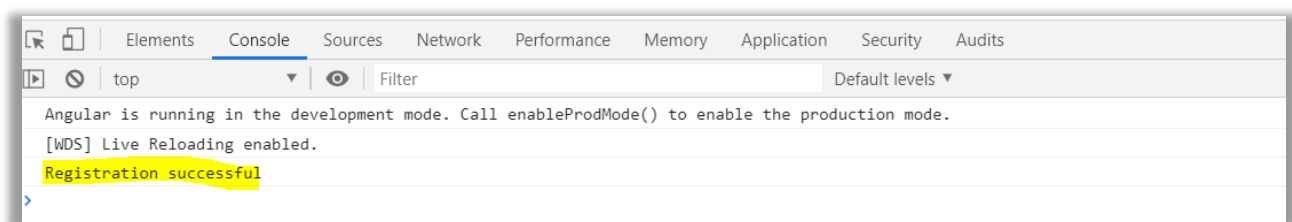
Last Name

Username

Password

[Register](#) [Cancel](#)

Once register button is clicked, check console. You will see Registration successful as a response



In this demo we create login and registration components and also created user service to register users' information using fakebackend providers. In next demo we will see how to authenticate and login using JWT.