Module 10: Authentication with JWT and Security

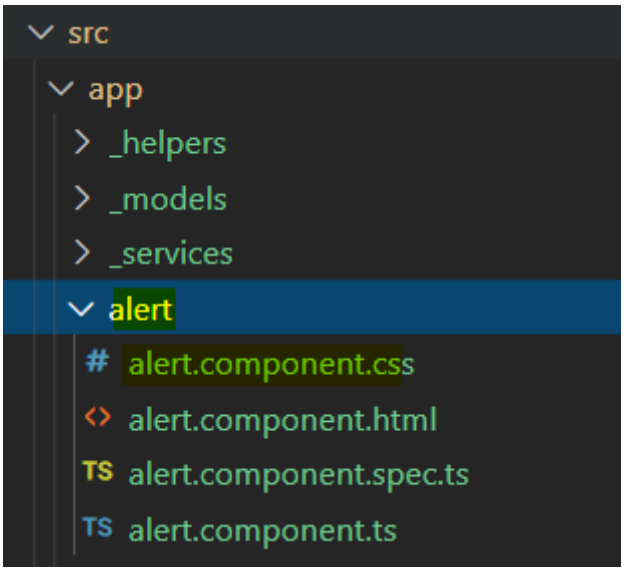
Demo Document 2: Use JWT authentication for login form

Use JWT authentication for login form created in demo1 of module 10

In this demo, we will see how authenticate user using JWT, we have already created login and register components in demo 1 of module 10, we will continue to use same module code to create JWT authentication

Step 1: Open visual studio and open folder Module10Demo1.

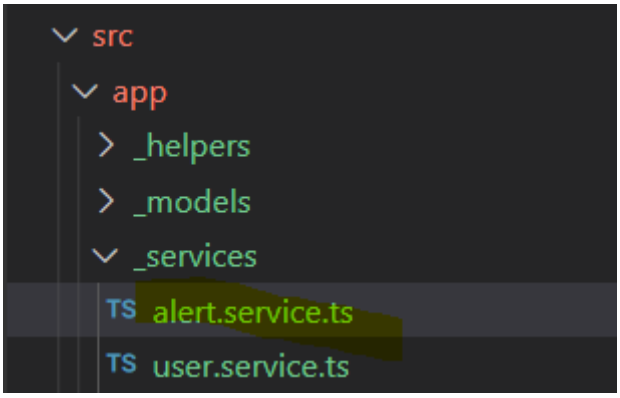
Step 2: Create alert component using command ng g c alert inside src/app folder alert component is just used to display error or success messages.




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

import {AlertService} from '../_services/alert.service'

@Component({
  selector: 'app-alert',
  templateUrl: './alert.component.html',
  styleUrls: ['./alert.component.css']
})
export class AlertComponent implements OnInit,OnDestroy {
  private subscription: Subscription;
  message: any;

  constructor(private alertService:AlertService) { }

  ngOnInit() {
    this.subscription = this.alertService.getAlert()
    .subscribe(message => {
        switch (message && message.type) {
            case 'success':
```

```
                message.cssClass = 'alert alert-success';
                break;
            case 'error':
                message.cssClass = 'alert alert-danger';
                break;
        }

        this.message = message;
    });
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
}

}
```

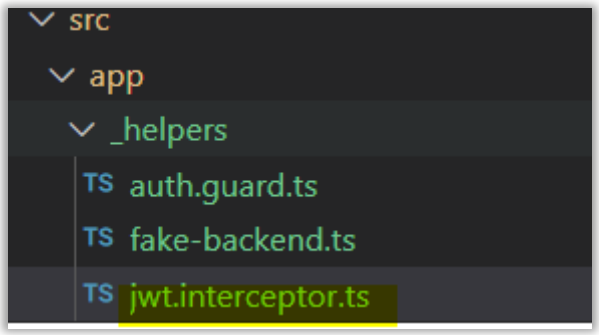
Step 3- Create alert service inside _services folder




```
import { Injectable } from '@angular/core';
import { Router, NavigationStart } from '@angular/router';
import { Observable, Subject } from 'rxjs';

@Injectable({ providedIn: 'root' })
export class AlertService {
    private subject = new Subject<any>();
    private keepAfterRouteChange = false;

    constructor(private router: Router) {
        // clear alert messages on route change unless 'keepAfterRouteChange' flag
 is true
        this.router.events.subscribe(event => {
            if (event instanceof NavigationStart) {
                if (this.keepAfterRouteChange) {
                    // only keep for a single route change
                    this.keepAfterRouteChange = false;
                } else {
```

```
                    // clear alert message
                    this.clear();
                }
            }
        });
    }

    getAlert(): Observable<any> {
        return this.subject.asObservable();
    }

    success(message: string, keepAfterRouteChange = false) {
        this.keepAfterRouteChange = keepAfterRouteChange;
        this.subject.next({ type: 'success', text: message });
    }

    error(message: string, keepAfterRouteChange = false) {
        this.keepAfterRouteChange = keepAfterRouteChange;
        this.subject.next({ type: 'error', text: message });
    }

    clear() {
        // clear by calling subject.next() without parameters
        this.subject.next();
    }
}
```

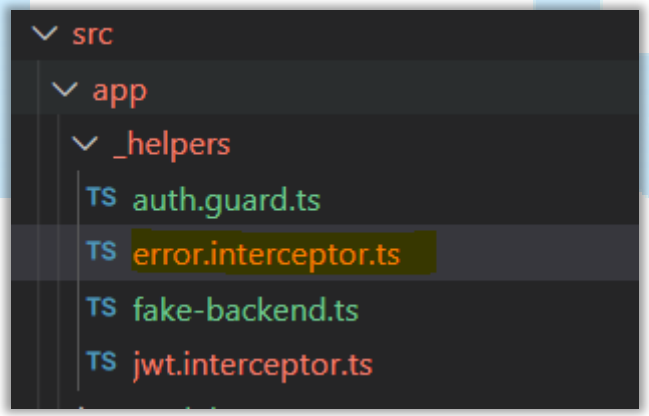
Step 4 – Create jwt interceptor to generate jwt token. Create new file inside _helpers as 'jwt.interceptor.ts'




```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/com
mon/http';
import { Observable } from 'rxjs';

import { AuthenticationService } from '../_services/authentication.service'

@Injectable()
```

```
export class JwtInterceptor implements HttpInterceptor {
    constructor(private authenticationService: AuthenticationService) {}

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<
any>> {
        // add authorization header with jwt token if available
        let currentUser = this.authenticationService.currentUserValue;
        if (currentUser && currentUser.token) {
            request = request.clone({
                setHeaders: {
                    Authorization: `Bearer ${currentUser.token}`
                }
            });
        }

        return next.handle(request);
    }
}
```

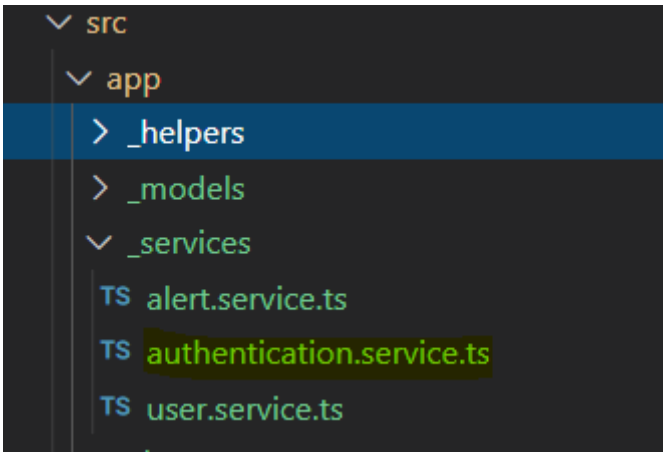
Step 5- Create error interceptor inside _helpers folder for errors




```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/com
mon/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

import { AuthenticationService } from '../_services/authentication.service';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
    constructor(private authenticationService: AuthenticationService) {}

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<
any>> {
        return next.handle(request).pipe(catchError(err => {
```

```
            if (err.status === 401) {
                // auto logout if 401 response returned from api
                this.authenticationService.logout();
                location.reload(true);
            }

            const error = err.error.message || err.statusText;
            return throwError(error);
        }))
    }
}
```

Step 6- Create authentication service inside _services folder.



```
import { Injectable } from '@angular/core';

import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '
@angular/router';


import { AuthenticationService } from '../_services/authentication.service'

@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
    constructor(
        private router: Router,
        private authenticationService: AuthenticationService
    ) {}

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const currentUser = this.authenticationService.currentUserValue;

        if (currentUser) {
            // authorised so return true
            return true;
        }
        // not logged in so redirect to login page with the return url
        this.router.navigate(['/login'], { queryParams: { returnUrl: state.url }})
;
        return false;
    }
}
```

Step 7- In the user.service.ts add the below code

```
src > app > _services > TS user.service.ts > UserService
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { User } from '../_models/user'
import {environment} from '../environments/environment'

@Injectable({ providedIn: 'root' })
export class UserService {
    constructor(private http: HttpClient) { }

    register(user: User) {
        return this.http.post(`${environment.apiUrl}/users/register`, user);
    }

    getAll() {
        return this.http.get<User[]>(`${environment.apiUrl}/users`);
    }

    delete(id: number) {
        return this.http.delete(`${environment.apiUrl}/users/${id}`);
    }


}
```

Step 8- Modify home component both html and .ts file and add code from demo folder

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

import { User } from '../_models/user'
import {  AuthenticationService } from '../_services/authentication.service';
import {UserService} from '../_services/user.service'

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  currentUser: User;
  users = [];

  constructor( private authenticationService: AuthenticationService,
    private userService: UserService) {
      this.currentUser = this.authenticationService.currentUserValue;
    }

  ngOnInit() {
    this.loadAllUsers();
  }

  deleteUser(id: number) {
    this.userService.delete(id)
        .pipe(first())
        .subscribe(() => this.loadAllUsers());
}

private loadAllUsers() {
    this.userService.getAll()
        .pipe(first())
        .subscribe(users => this.users = users);
}

}
```

Similarly modify login component, register component and app component.to add updated code from demo zip file

Step 12 – open app.module.ts file and import JWT and error interceptors as below

```
src > app > TS app.module.ts > ...
import { HomeComponent } from './home/home.component';

import { HttpClientModule,HTTP_INTERCEPTORS } from '@angular/common/http';
import { fakeBackendProvider } from './_helpers/fake-backend';
import { AlertComponent } from './alert/alert.component'

import { JwtInterceptor } from './_helpers/jwt.interceptor';
import { ErrorInterceptor } from './_helpers/error.interceptor'

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    RegisterComponent,
    HomeComponent,
    AlertComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },
    fakeBackendProvider],
  bootstrap: [AppComponent]
})
```

Step 13 – Open fake-backend.ts file:

```
mport { 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 d
elay even if an error is thrown (https://github.com/Reactive-
Extensions/RxJS/issues/648)
            .pipe(delay(500))
```

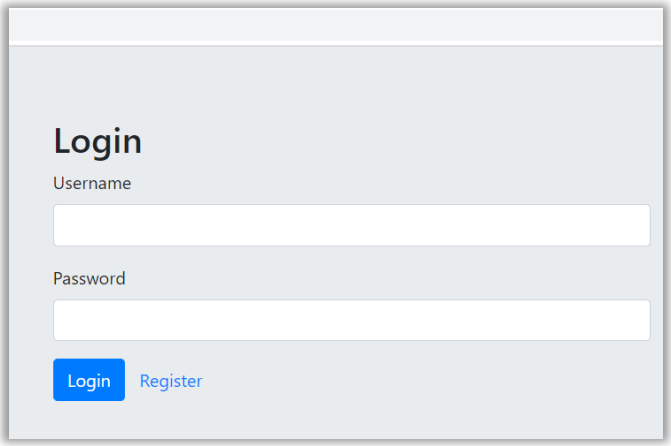
```
            .pipe(dematerialize());

        function handleRoute() {
            switch (true) {
                case url.endsWith('/users/register') && method === 'POST':
                    return register();
                case url.endsWith('/users/authenticate') && method === 'POST':
                    return authenticate();
                case url.endsWith('/users') && method === 'GET':
                        return getUsers();
                case url.match(/\/users\/\d+$/) && method === 'DELETE':
                        return deleteUser();
                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();
        }

        function authenticate() {
            const { username, password } = body;
            const user = users.find(x => x.username === username && x.password ===
password);
            if (!user) return error('Username or password is incorrect');
            return ok({
                id: user.id,
                username: user.username,
                firstName: user.firstName,
                lastName: user.lastName,
                token: 'fake-jwt-token'
            })
        }
```

```
        function getUsers() {
            if (!isLoggedIn()) return unauthorized();
            return ok(users);
        }

        function deleteUser() {
            if (!isLoggedIn()) return unauthorized();

            users = users.filter(x => x.id !== idFromUrl());
            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 } });
        }

        function unauthorized() {
            return throwError({ status: 401, error: { message: 'Unauthorised' } })
;
        }

        function isLoggedIn() {
            return headers.get('Authorization') === 'Bearer fake-jwt-token';
        }

        function idFromUrl() {
            const urlParts = url.split('/');
            return parseInt(urlParts[urlParts.length - 1]);
        }


    }
}

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

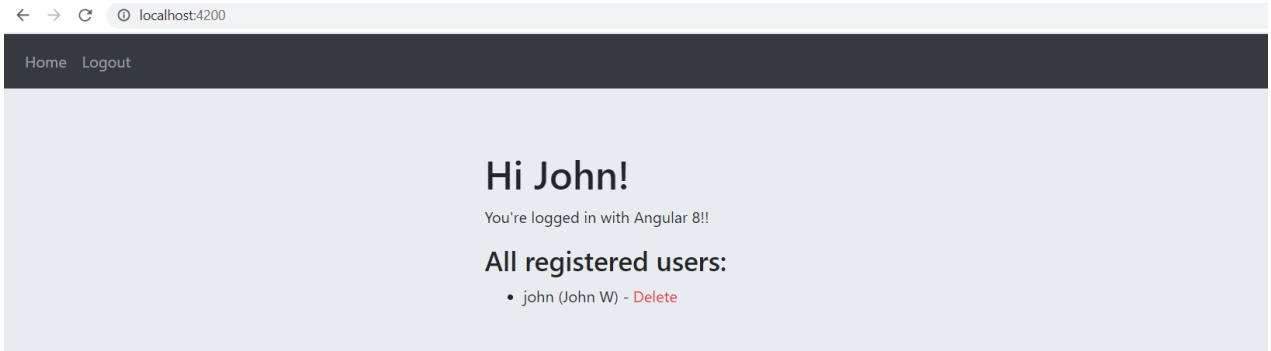
Step 15 – Open auth.guard.ts

```
import { Injectable } from '@angular/core';
import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '
@angular/router';


import { AuthenticationService } from '../_services/authentication.service'

@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
    constructor(
        private router: Router,
        private authenticationService: AuthenticationService
    ) {}

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const currentUser = this.authenticationService.currentUserValue;

        if (currentUser) {
            // authorised so return true
            return true;
        }
        // not logged in so redirect to login page with the return url
        this.router.navigate(['/login'], { queryParams: { returnUrl: state.url }})
;
        return false;
    }
}
```

Step 14 – Run app using ng serve command.



Now use the same user which we have registered in previous demo. (e.g. I used John), and try to login using same credentials



Application successfully logged in

You can create different users and register.