```
export interface AuthData {
  email: string;
  password: string;
}


// Auth interceptor

import {
  HttpInterceptor,
  HttpRequest,
  HttpHandler
} from "@angular/common/http";
import { Injectable } from "@angular/core";

import { AuthService } from "./auth.service";

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const authToken = this.authService.getToken();
    const authRequest = req.clone({
      headers: req.headers.set("Authorization", "Bearer " + authToken)
    });
    return next.handle(authRequest);
  }
}

// Auth guard

import {
  CanActivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  Router
} from "@angular/router";
import { Injectable } from "@angular/core";
import { Observable } from "rxjs";

import { AuthService } from "./auth.service";

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): boolean | Observable<boolean> | Promise<boolean> {
    const isAuth = this.authService.getIsAuth();
    if (!isAuth) {
      this.router.navigate(['/login']);
    }
    return isAuth;
  }
}

// Auth Service
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { Router } from "@angular/router";
import { Subject } from "rxjs";

import { environment } from "../../environments/environment";
import { AuthData } from "./auth-data.model";

const BACKEND_URL = environment.apiUrl + "/user/";

@Injectable({ providedIn: "root" })
export class AuthService {
  private isAuthenticated = false;
  private token: string;
  private tokenTimer: any;
  private userId: string;
```

```
private authStatusListener = new Subject<boolean>();

constructor(private http: HttpClient, private router: Router) {}

getToken() {
  return this.token;
}

getIsAuth() {
  return this.isAuthenticated;
}

getUserId() {
  return this.userId;
}

getAuthStatusListener() {
  return this.authStatusListener.asObservable();
}

createUser(email: string, password: string) {
  const authData: AuthData = { email: email, password: password };
  this.http.post(BACKEND_URL + "/signup", authData).subscribe(
    () => {
      this.router.navigate(["/"]);
    },
    error => {
      this.authStatusListener.next(false);
    }
  );
}

login(email: string, password: string) {
  const authData: AuthData = { email: email, password: password };
  this.http
    .post<{ token: string; expiresIn: number; userId: string }>(
      BACKEND_URL + "/login",
      authData
    )
    .subscribe(
      response => {
        const token = response.token;
        this.token = token;
        if (token) {
          const expiresInDuration = response.expiresIn;
          this.setAuthTimer(expiresInDuration);
          this.isAuthenticated = true;
          this.userId = response.userId;
          this.authStatusListener.next(true);
          const now = new Date();
          const expirationDate = new Date(
            now.getTime() + expiresInDuration * 1000
          );
          console.log(expirationDate);
          this.saveAuthData(token, expirationDate, this.userId);
          this.router.navigate(["/"]);
        }
      },
      error => {
        this.authStatusListener.next(false);
      }
    );
}

autoAuthUser() {
  const authInformation = this.getAuthData();
  if (!authInformation) {
    return;
  }
  const now = new Date();
  const expiresIn = authInformation.expirationDate.getTime() - now.getTime();
  if (expiresIn > 0) {
    this.token = authInformation.token;
    this.isAuthenticated = true;
    this.userId = authInformation.userId;
    this.setAuthTimer(expiresIn / 1000);
    this.authStatusListener.next(true);
```

```
    }
  }

  logout() {
    this.token = null;
    this.isAuthenticated = false;
    this.authStatusListener.next(false);
    this.userId = null;
    clearTimeout(this.tokenTimer);
    this.clearAuthData();
    this.router.navigate(["/"]);
  }

  private setAuthTimer(duration: number) {
    console.log("Setting timer: " + duration);
    this.tokenTimer = setTimeout(() => {
      this.logout();
    }, duration * 1000);
  }

  private saveAuthData(token: string, expirationDate: Date, userId: string) {
    localStorage.setItem("token", token);
    localStorage.setItem("expiration", expirationDate.toISOString());
    localStorage.setItem("userId", userId);
  }

  private clearAuthData() {
    localStorage.removeItem("token");
    localStorage.removeItem("expiration");
    localStorage.removeItem("userId");
  }

  private getAuthData() {
    const token = localStorage.getItem("token");
    const expirationDate = localStorage.getItem("expiration");
    const userId = localStorage.getItem("userId");
    if (!token || !expirationDate) {
      return;
    }
    return {
      token: token,
      expirationDate: new Date(expirationDate),
      userId: userId
    };
  }
}
```