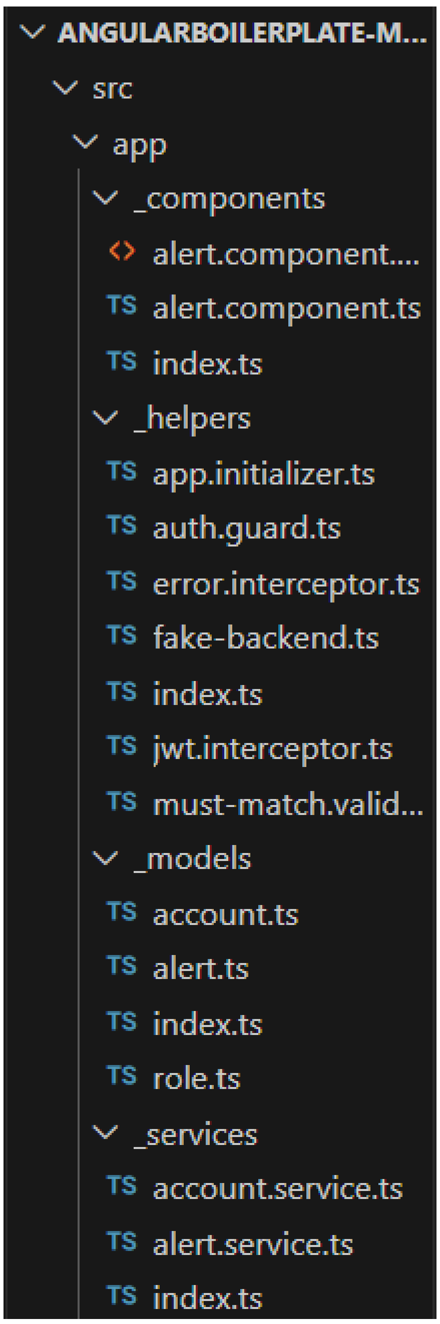
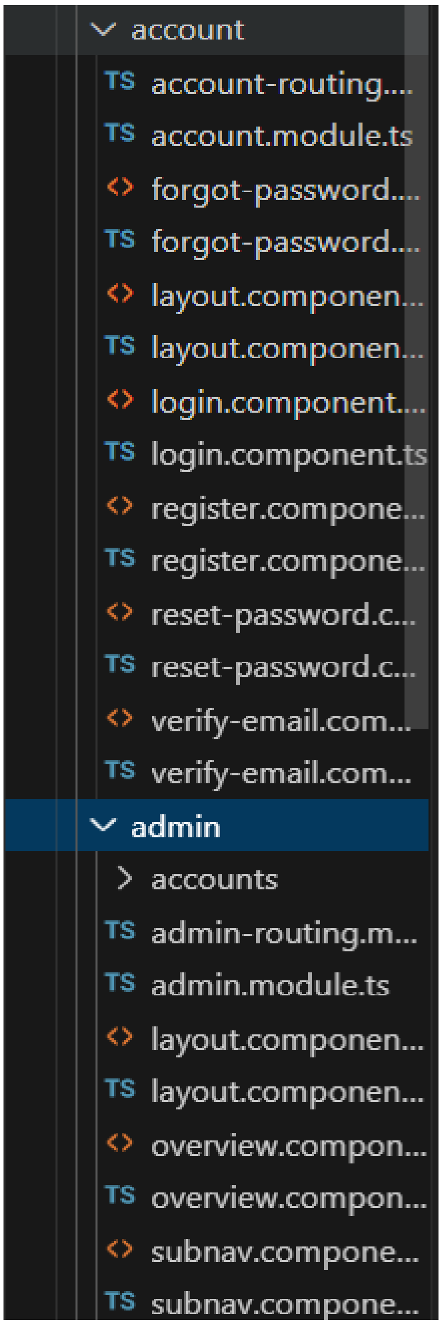
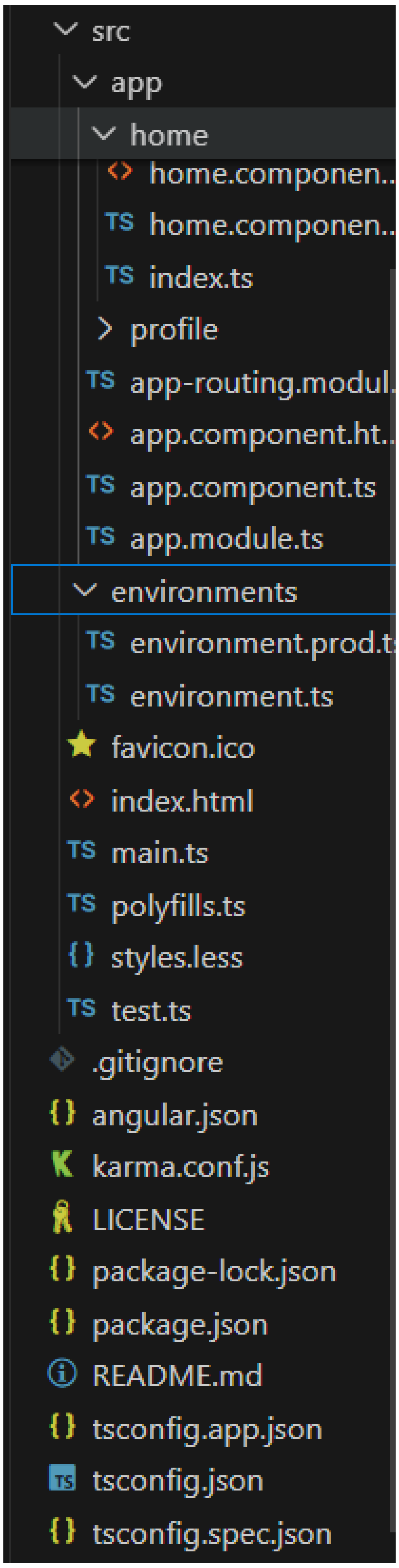


```
v ANGULARBOILERPLATE-M...
  v src
    v app
      v _components
        <> alert.component....
        TS alert.component.ts
        TS index.ts
      v _helpers
        TS app.initializer.ts
        TS auth.guard.ts
        TS error.interceptor.ts
        TS fake-backend.ts
        TS index.ts
        TS jwt.interceptor.ts
        TS must-match.valid...
      v _models
        TS account.ts
        TS alert.ts
        TS index.ts
        TS role.ts
      v _services
        TS account.service.ts
        TS alert.service.ts
        TS index.ts
```

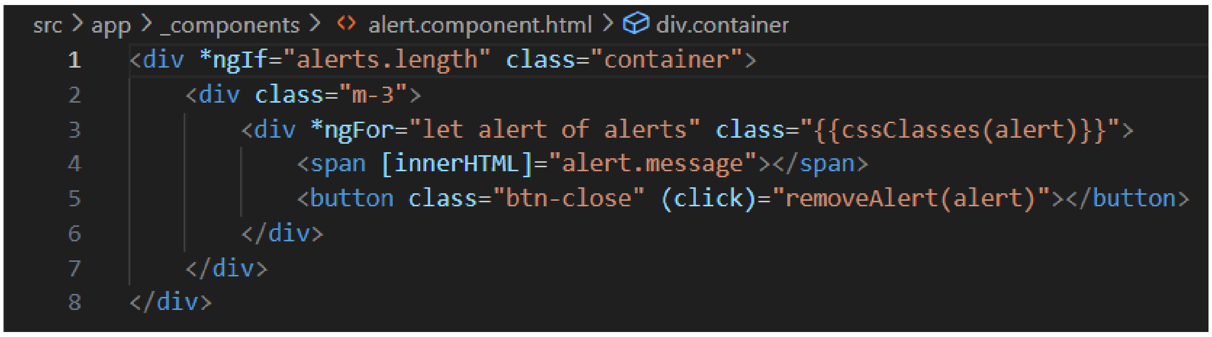


```
      v account
        TS account-routing....
        TS account.module.ts
        <> forgot-password....
        TS forgot-password....
        <> layout.componen...
        TS layout.componen...
        <> login.component....
        TS login.component.ts
        <> register.compone...
        TS register.compone...
        <> reset-password.c...
        TS reset-password.c...
        <> verify-email.com...
        TS verify-email.com...
      v admin
        > accounts
        TS admin-routing.m...
        TS admin.module.ts
        <> layout.componen...
        TS layout.componen...
        <> overview.compon...
        TS overview.compon...
        <> subnav.compone...
        TS subnav.compone...
```



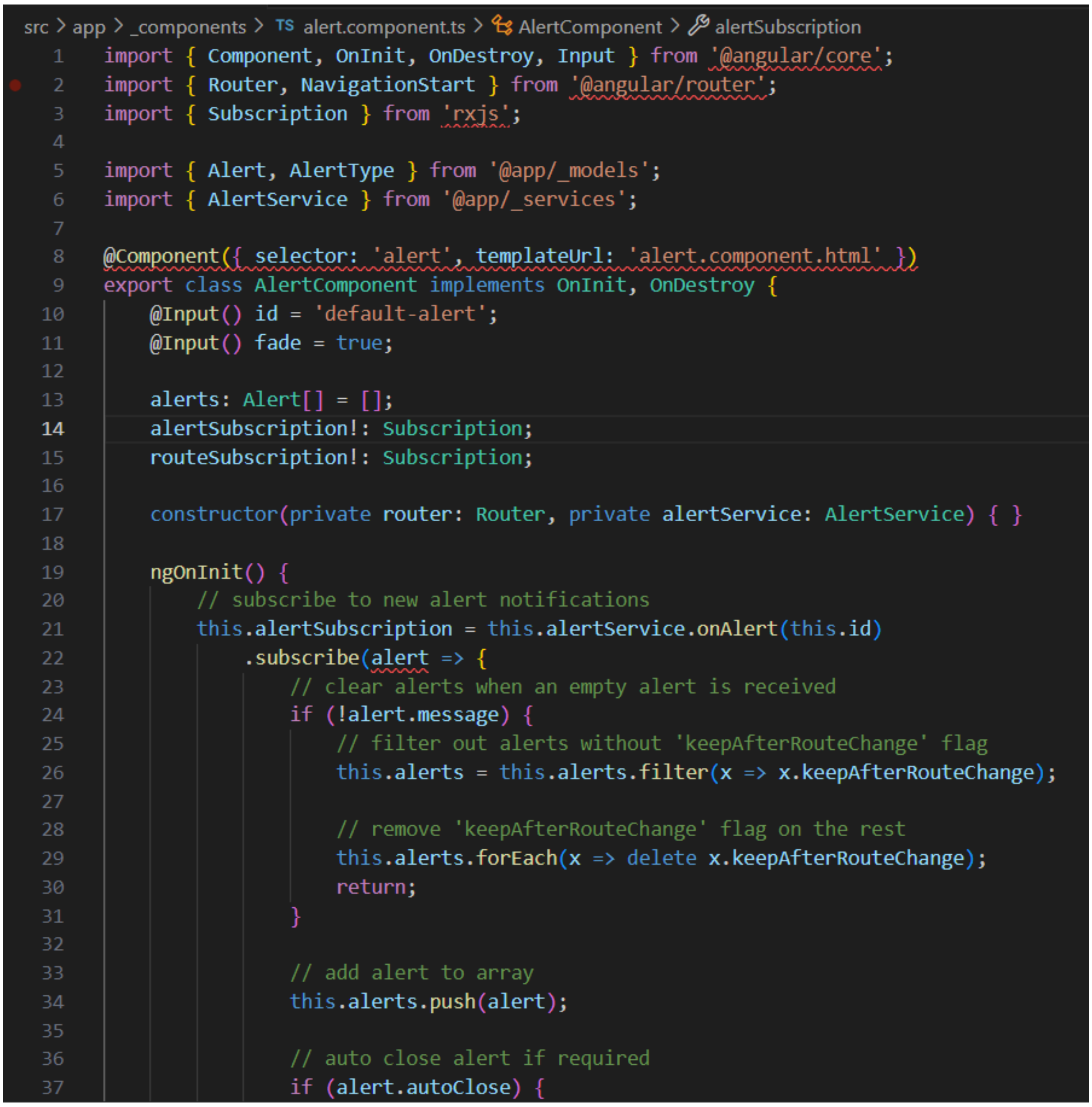
```
src
  app
    home
      home.componen..
      home.componen..
      index.ts
    profile
    app-routing.modul.
    app.component.ht..
    app.component.ts
    app.module.ts
  environments
    environment.prod.t
    environment.ts
  favicon.ico
  index.html
  main.ts
  polyfills.ts
  styles.less
  test.ts
.gitignore
angular.json
karma.conf.js
LICENSE
package-lock.json
package.json
README.md
tsconfig.app.json
tsconfig.json
tsconfig.spec.json
```

src > app > _component > alert.component.html

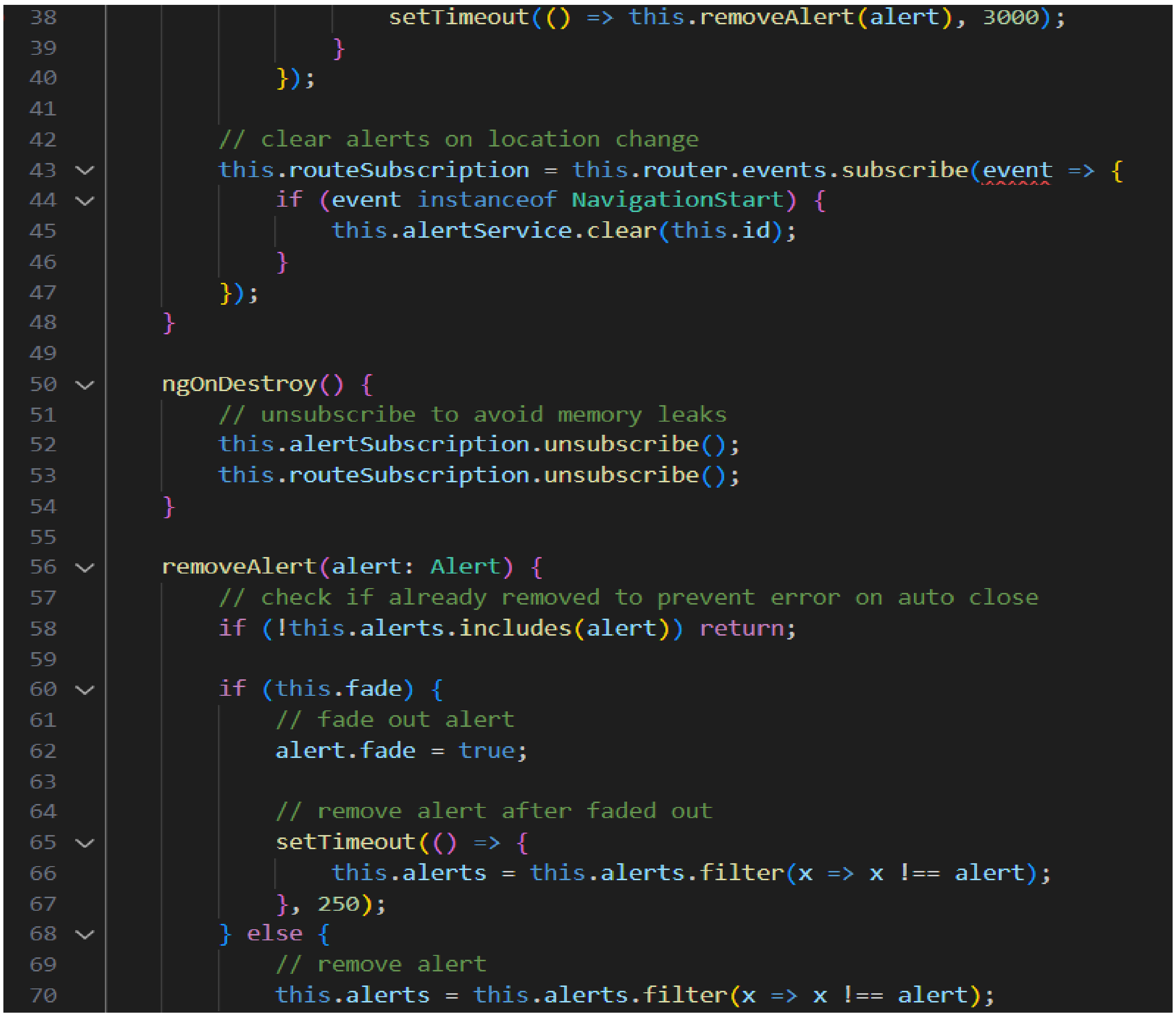

src > app > _components > alert.component.html > div.container

```html
<div *ngIf="alerts.length" class="container">
    <div class="m-3">
        <div *ngFor="let alert of alerts" class="{{cssClasses(alert)}}">
            <span [innerHTML]="alert.message"></span>
            <button class="btn-close" (click)="removeAlert(alert)"></button>
        </div>
    </div>
</div>
```

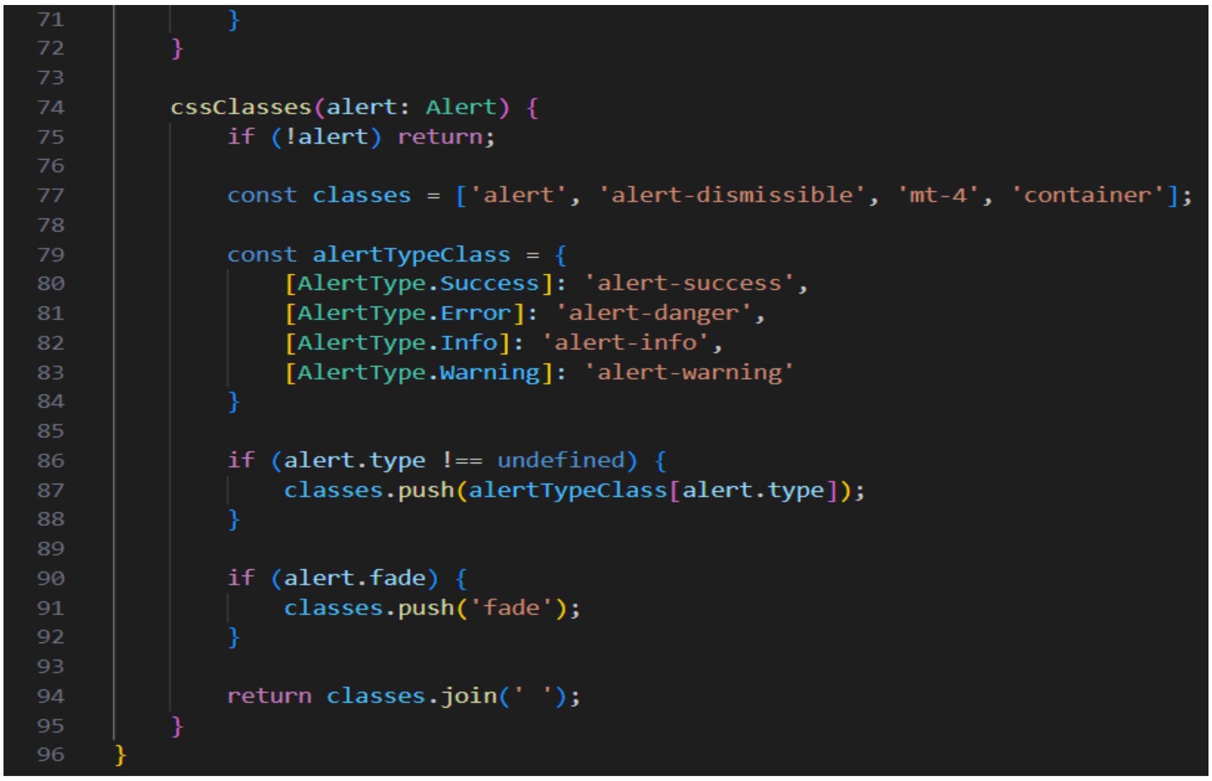
src > app > _component > alert.component.ts


src > app > _components > TS alert.component.ts > AlertComponent > alertSubscription

```typescript
import { Component, OnInit, OnDestroy, Input } from '@angular/core';
import { Router, NavigationStart } from '@angular/router';
import { Subscription } from 'rxjs';

import { Alert, AlertType } from '@app/_models';
import { AlertService } from '@app/_services';

@Component({ selector: 'alert', templateUrl: 'alert.component.html' })
export class AlertComponent implements OnInit, OnDestroy {
    @Input() id = 'default-alert';
    @Input() fade = true;

    alerts: Alert[] = [];
    alertSubscription!: Subscription;
    routeSubscription!: Subscription;

    constructor(private router: Router, private alertService: AlertService) { }

    ngOnInit() {
        // subscribe to new alert notifications
        this.alertSubscription = this.alertService.onAlert(this.id)
            .subscribe(alert => {
                // clear alerts when an empty alert is received
                if (!alert.message) {
                    // filter out alerts without 'keepAfterRouteChange' flag
                    this.alerts = this.alerts.filter(x => x.keepAfterRouteChange);

                    // remove 'keepAfterRouteChange' flag on the rest
                    this.alerts.forEach(x => delete x.keepAfterRouteChange);
                    return;
                }

                // add alert to array
                this.alerts.push(alert);

                // auto close alert if required
                if (alert.autoClose) {
```

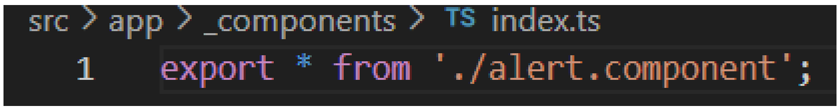


```
                    setTimeout(() => this.removeAlert(alert), 3000);
                }
            });

        // clear alerts on location change
        this.routeSubscription = this.router.events.subscribe(event => {
            if (event instanceof NavigationStart) {
                this.alertService.clear(this.id);
            }
        });
    }

    ngOnDestroy() {
        // unsubscribe to avoid memory leaks
        this.alertSubscription.unsubscribe();
        this.routeSubscription.unsubscribe();
    }

    removeAlert(alert: Alert) {
        // check if already removed to prevent error on auto close
        if (!this.alerts.includes(alert)) return;

        if (this.fade) {
            // fade out alert
            alert.fade = true;

            // remove alert after faded out
            setTimeout(() => {
                this.alerts = this.alerts.filter(x => x !== alert);
            }, 250);
        } else {
            // remove alert
            this.alerts = this.alerts.filter(x => x !== alert);
```



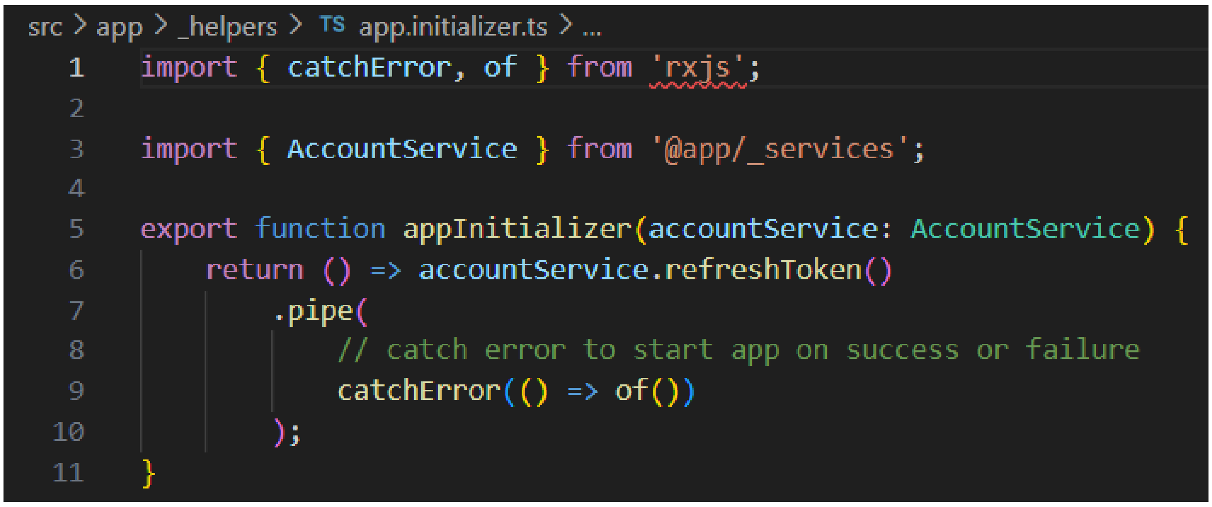
```
        }
    }

    cssClasses(alert: Alert) {
        if (!alert) return;

        const classes = ['alert', 'alert-dismissible', 'mt-4', 'container'];

        const alertTypeClass = {
            [AlertType.Success]: 'alert-success',
            [AlertType.Error]: 'alert-danger',
            [AlertType.Info]: 'alert-info',
            [AlertType.Warning]: 'alert-warning'
        }

        if (alert.type !== undefined) {
            classes.push(alertTypeClass[alert.type]);
        }

        if (alert.fade) {
            classes.push('fade');
        }

        return classes.join(' ');
    }
}
```

src > app >_components > index.ts



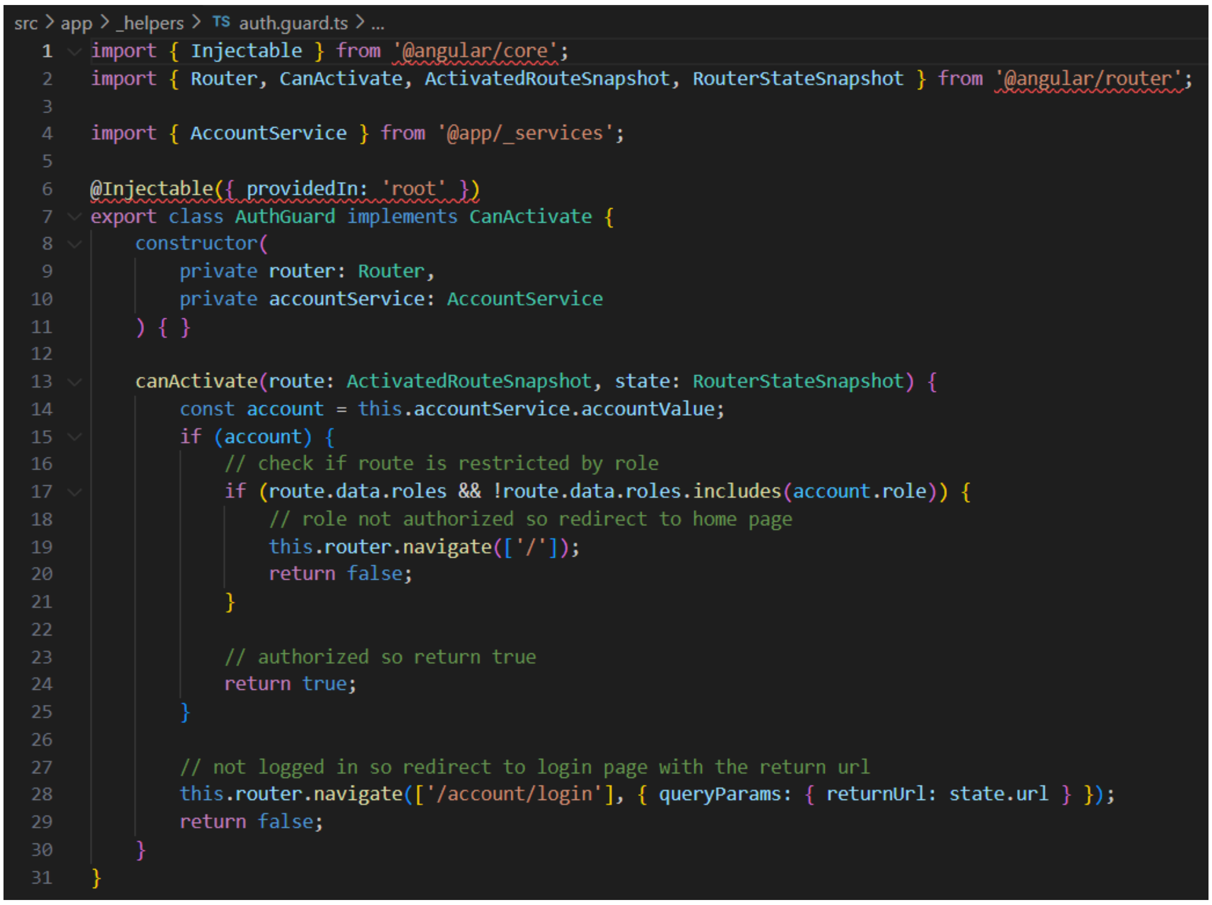
src > app > _components > index.ts

```
export * from './alert.component';
```



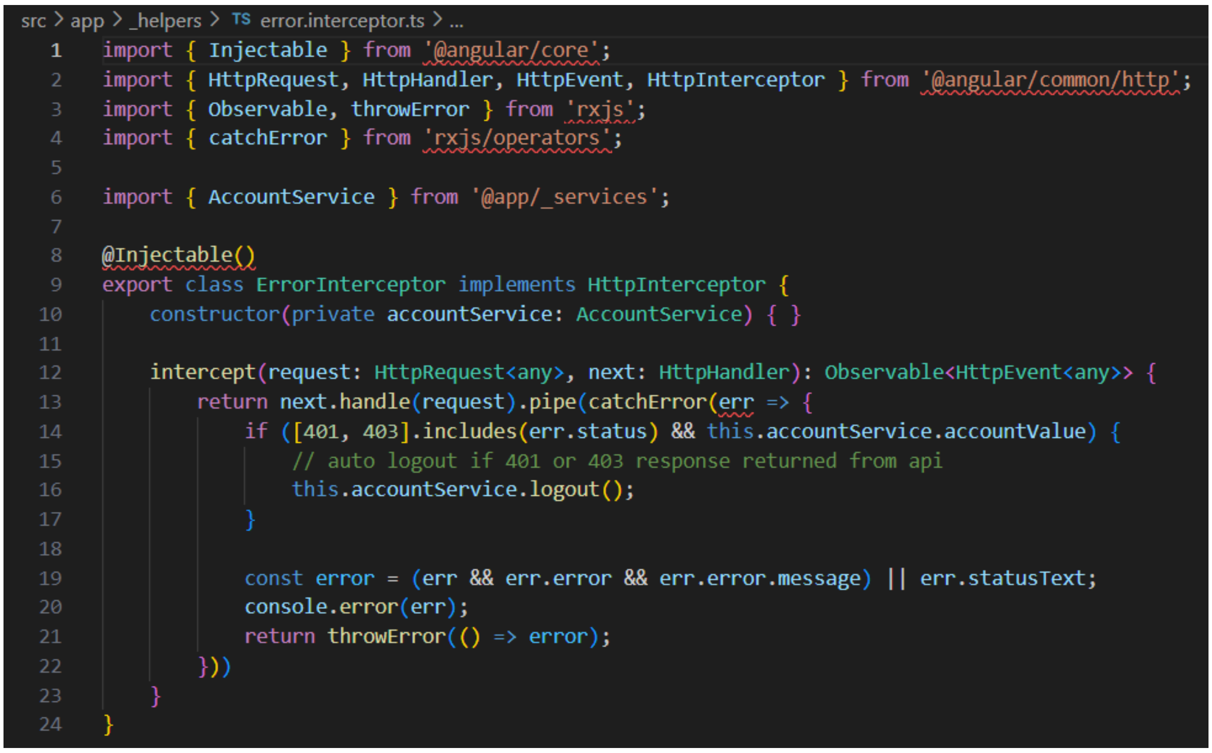
src > app > _helpers > app.initializer.ts > ...

```
import { catchError, of } from 'rxjs';

import { AccountService } from '@app/_services';

export function appInitializer(accountService: AccountService) {
    return () => accountService.refreshToken()
        .pipe(
            // catch error to start app on success or failure
            catchError(() => of())
        );
}
```

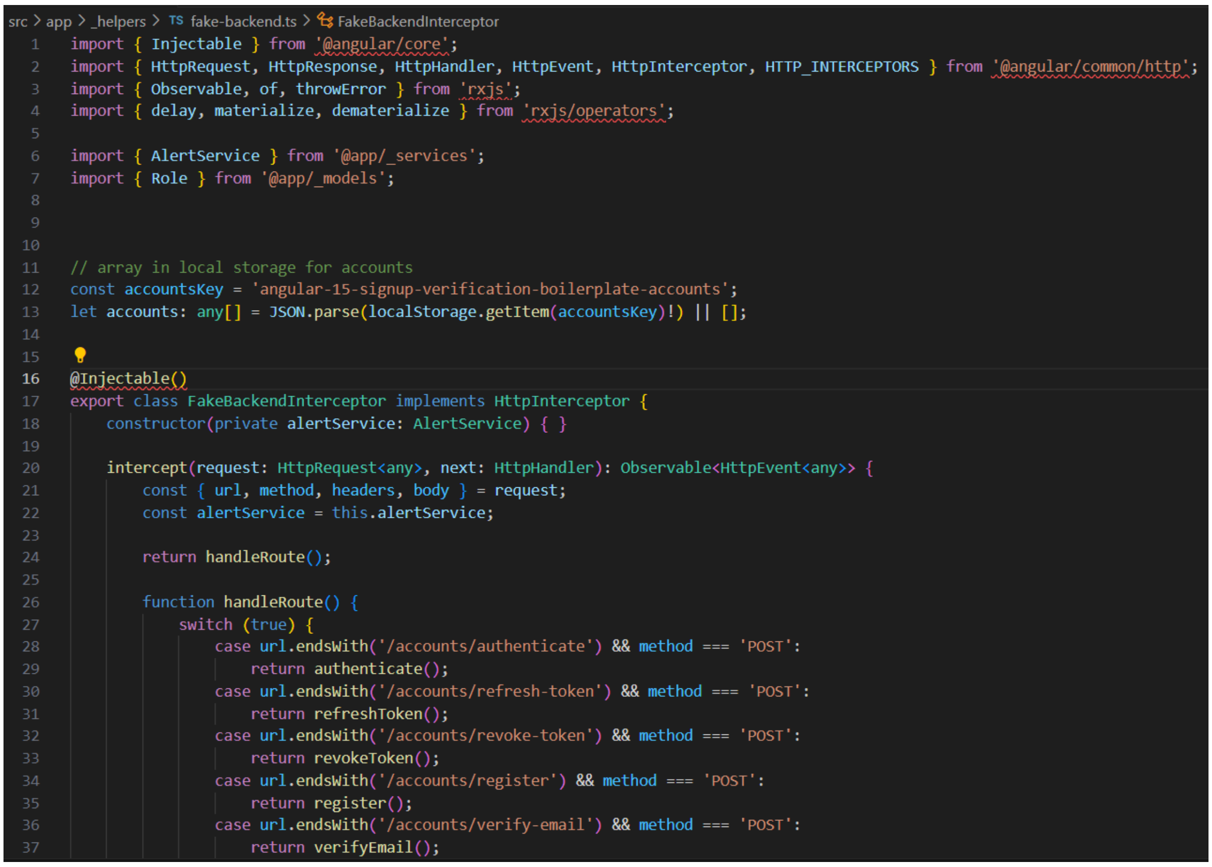


src > app > _helpers > auth.guard.ts > ...

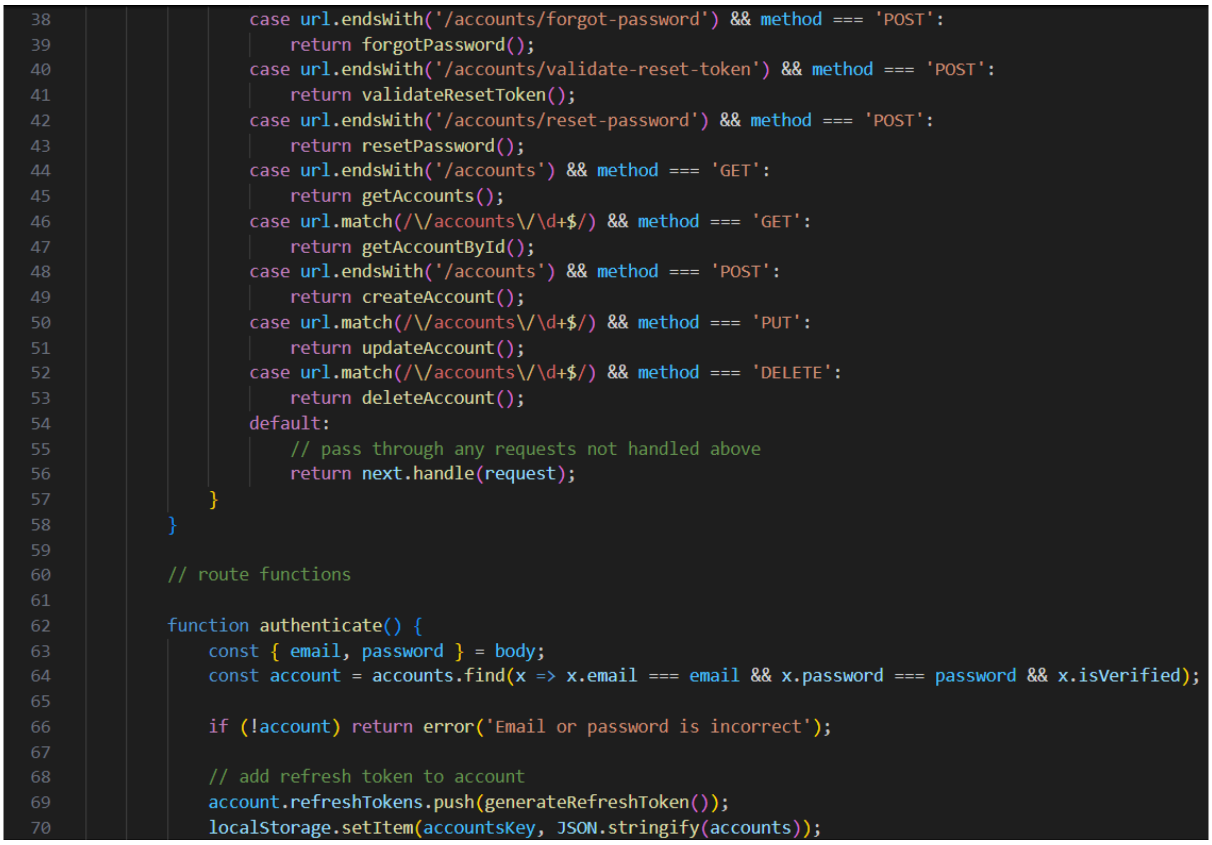
```
import { Injectable } from '@angular/core';
import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

import { AccountService } from '@app/_services';

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

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const account = this.accountService.accountValue;
        if (account) {
            // check if route is restricted by role
            if (route.data.roles && !route.data.roles.includes(account.role)) {
                // role not authorized so redirect to home page
                this.router.navigate(['/']);
                return false;
            }

            // authorized so return true
            return true;
        }

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


src > app > _helpers > TS error.interceptor.ts > ...

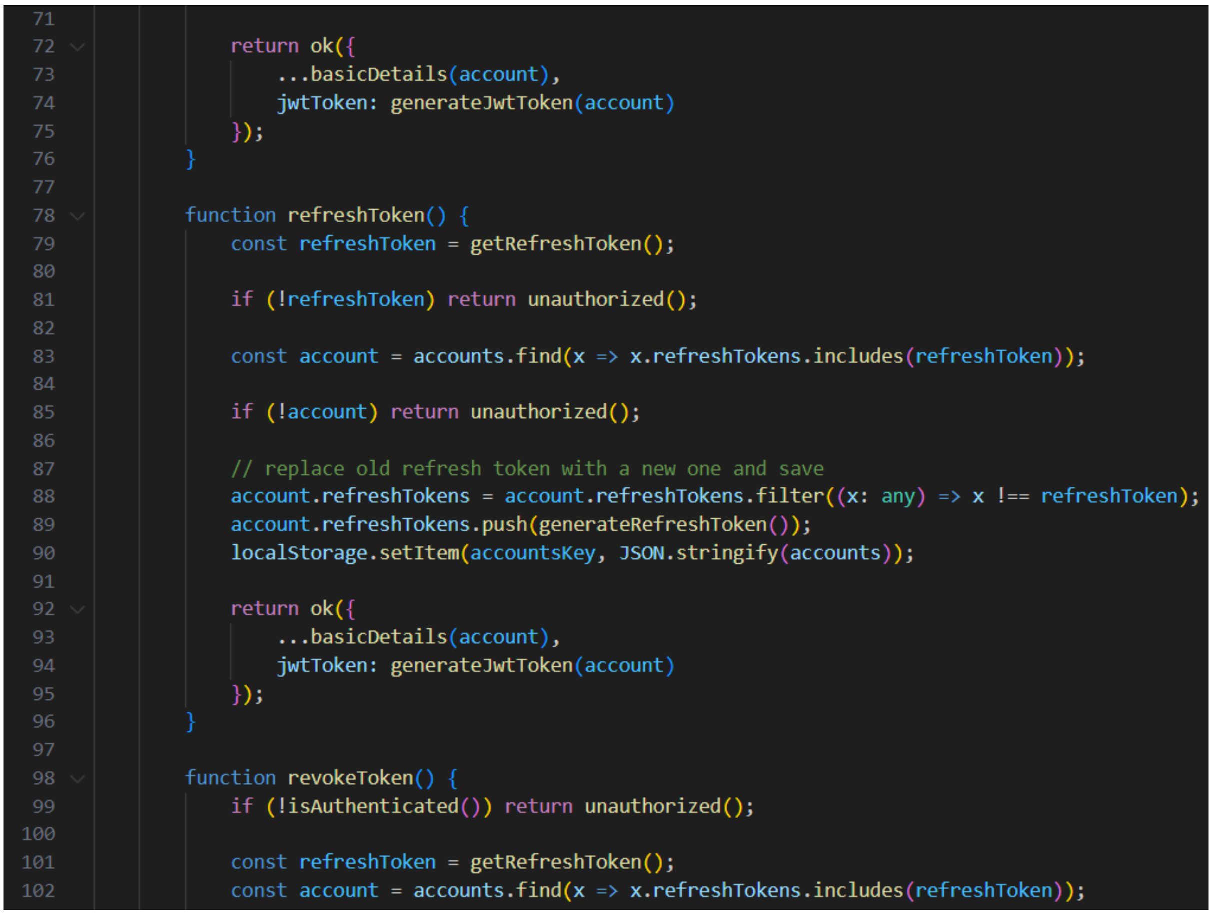
```typescript
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

import { AccountService } from '@app/_services';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
    constructor(private accountService: AccountService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        return next.handle(request).pipe(catchError(err => {
            if ([401, 403].includes(err.status) && this.accountService.accountValue) {
                // auto logout if 401 or 403 response returned from api
                this.accountService.logout();
            }

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


src > app > _helpers > TS fake-backend.ts > FakeBackendInterceptor

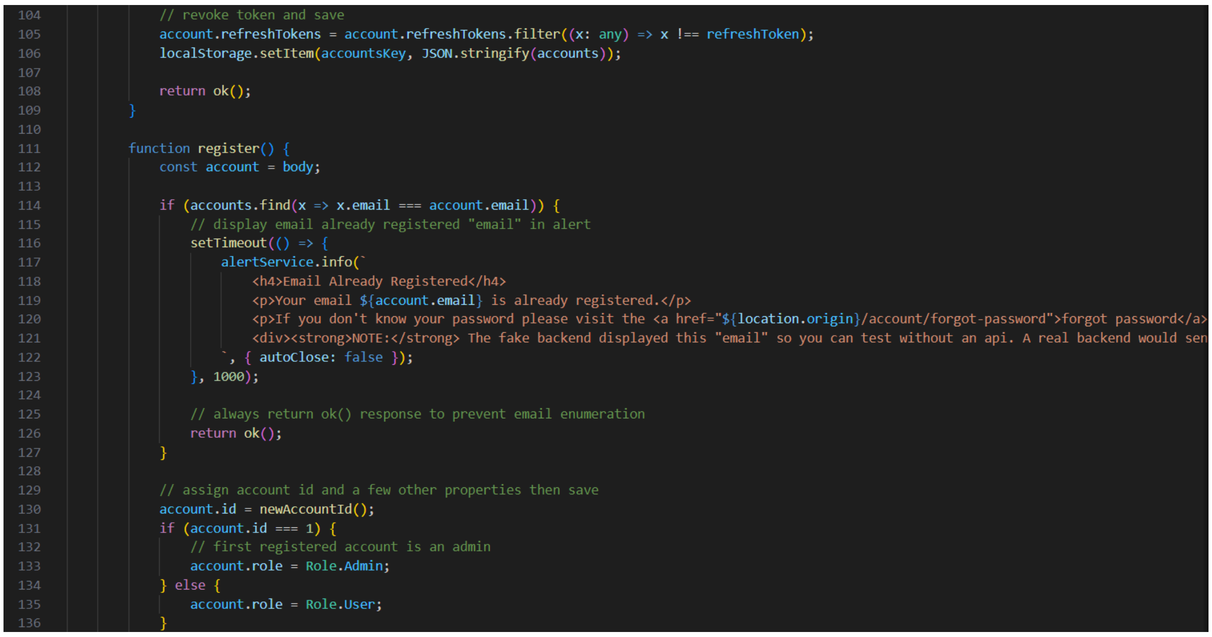
```typescript
import { Injectable } from '@angular/core';
import { HttpRequest, HttpResponse, HttpHandler, HttpEvent, HttpInterceptor, HTTP_INTERCEPTORS } from '@angular/common/http';
import { Observable, of, throwError } from 'rxjs';
import { delay, materialize, dematerialize } from 'rxjs/operators';

import { AlertService } from '@app/_services';
import { Role } from '@app/_models';



// array in local storage for accounts
const accountsKey = 'angular-15-signup-verification-boilerplate-accounts';
let accounts: any[] = JSON.parse(localStorage.getItem(accountsKey)!) || [];


@Injectable()
export class FakeBackendInterceptor implements HttpInterceptor {
    constructor(private alertService: AlertService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const { url, method, headers, body } = request;
        const alertService = this.alertService;

        return handleRoute();

        function handleRoute() {
            switch (true) {
                case url.endsWith('/accounts/authenticate') && method === 'POST':
                    return authenticate();
                case url.endsWith('/accounts/refresh-token') && method === 'POST':
                    return refreshToken();
                case url.endsWith('/accounts/revoke-token') && method === 'POST':
                    return revokeToken();
                case url.endsWith('/accounts/register') && method === 'POST':
                    return register();
                case url.endsWith('/accounts/verify-email') && method === 'POST':
                    return verifyEmail();
```



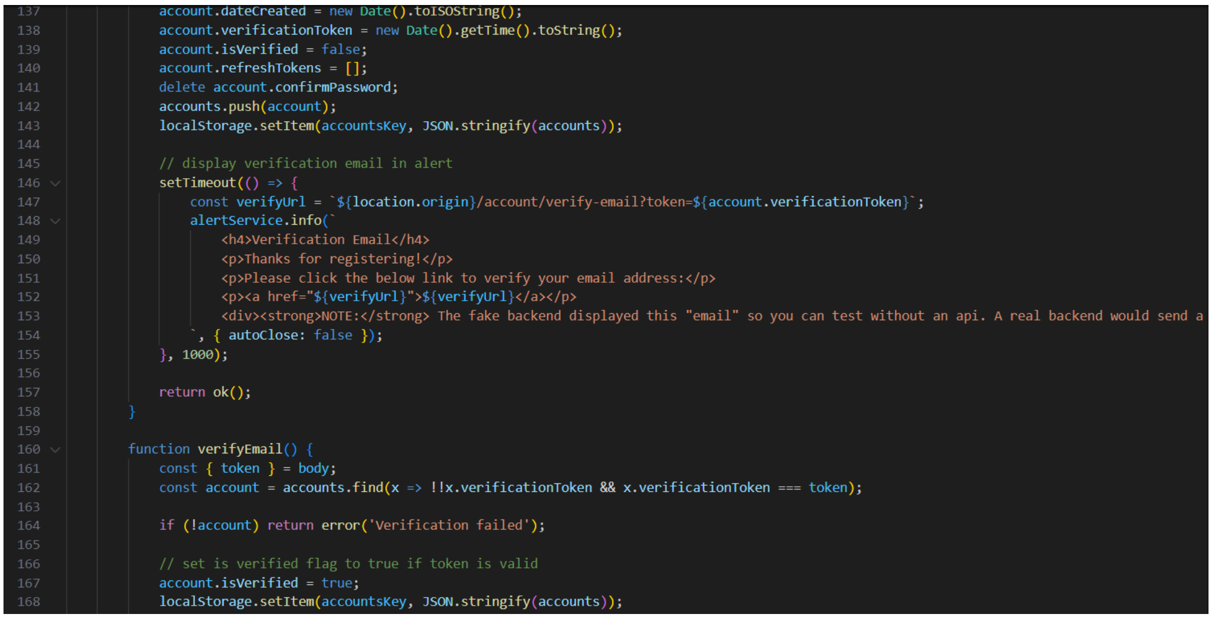
```
                case url.endsWith('/accounts/forgot-password') && method === 'POST':
                    return forgotPassword();
                case url.endsWith('/accounts/validate-reset-token') && method === 'POST':
                    return validateResetToken();
                case url.endsWith('/accounts/reset-password') && method === 'POST':
                    return resetPassword();
                case url.endsWith('/accounts') && method === 'GET':
                    return getAccounts();
                case url.match(/\/accounts\/\d+$/) && method === 'GET':
                    return getAccountById();
                case url.endsWith('/accounts') && method === 'POST':
                    return createAccount();
                case url.match(/\/accounts\/\d+$/) && method === 'PUT':
                    return updateAccount();
                case url.match(/\/accounts\/\d+$/) && method === 'DELETE':
                    return deleteAccount();
                default:
                    // pass through any requests not handled above
                    return next.handle(request);
            }
        }

        // route functions

        function authenticate() {
            const { email, password } = body;
            const account = accounts.find(x => x.email === email && x.password === password && x.isVerified);

            if (!account) return error('Email or password is incorrect');

            // add refresh token to account
            account.refreshTokens.push(generateRefreshToken());
            localStorage.setItem(accountsKey, JSON.stringify(accounts));
```



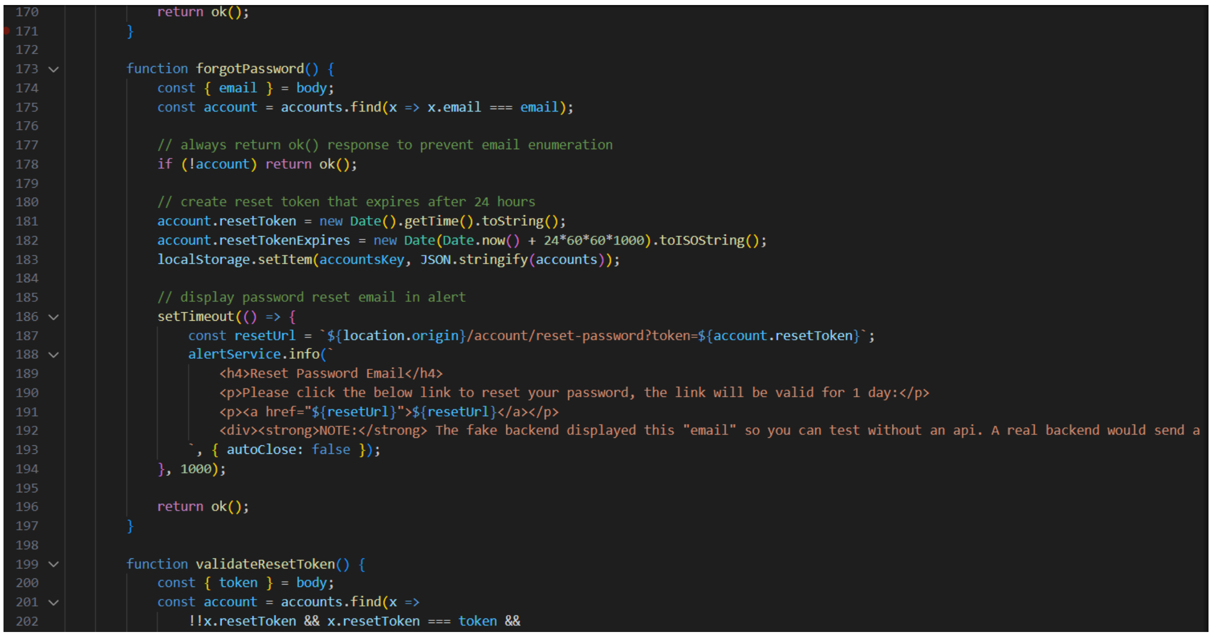
```

            return ok({
                ...basicDetails(account),
                jwtToken: generateJwtToken(account)
            });
        }

        function refreshToken() {
            const refreshToken = getRefreshToken();

            if (!refreshToken) return unauthorized();

            const account = accounts.find(x => x.refreshTokens.includes(refreshToken));

            if (!account) return unauthorized();

            // replace old refresh token with a new one and save
            account.refreshTokens = account.refreshTokens.filter((x: any) => x !== refreshToken);
            account.refreshTokens.push(generateRefreshToken());
            localStorage.setItem(accountsKey, JSON.stringify(accounts));

            return ok({
                ...basicDetails(account),
                jwtToken: generateJwtToken(account)
            });
        }

        function revokeToken() {
            if (!isAuthenticated()) return unauthorized();

            const refreshToken = getRefreshToken();
            const account = accounts.find(x => x.refreshTokens.includes(refreshToken));
```



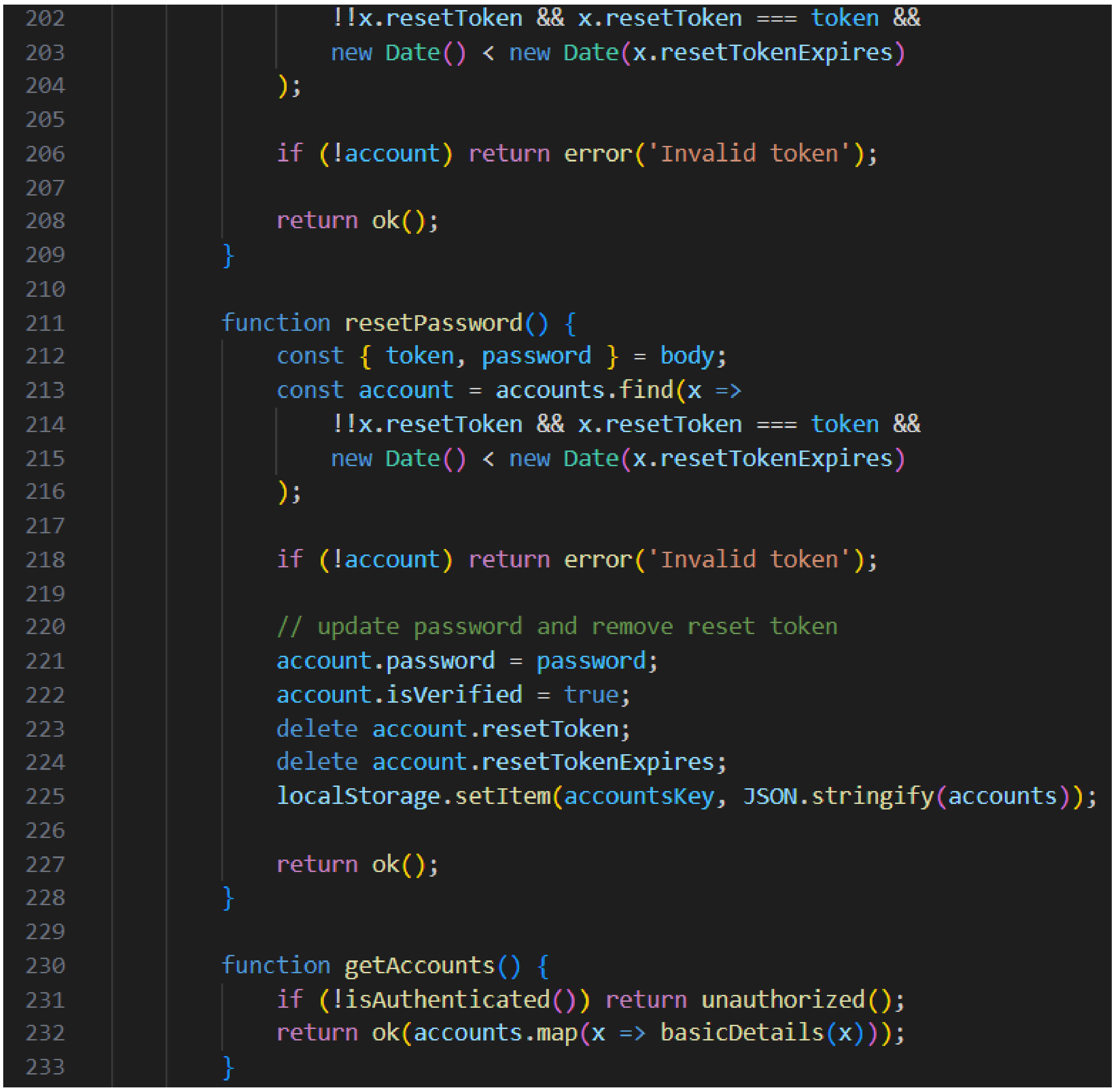
```
            // revoke token and save
            account.refreshTokens = account.refreshTokens.filter((x: any) => x !== refreshToken);
            localStorage.setItem(accountsKey, JSON.stringify(accounts));

            return ok();
        }

        function register() {
            const account = body;

            if (accounts.find(x => x.email === account.email)) {
                // display email already registered "email" in alert
                setTimeout(() => {
                    alertService.info(`
                        <h4>Email Already Registered</h4>
                        <p>Your email ${account.email} is already registered.</p>
                        <p>If you don't know your password please visit the <a href="${location.origin}/account/forgot-password">forgot password</a>
                        <div><strong>NOTE:</strong> The fake backend displayed this "email" so you can test without an api. A real backend would sen
                    `, { autoClose: false });
                }, 1000);

                // always return ok() response to prevent email enumeration
                return ok();
            }

            // assign account id and a few other properties then save
            account.id = newAccountId();
            if (account.id === 1) {
                // first registered account is an admin
                account.role = Role.Admin;
            } else {
                account.role = Role.User;
            }
```



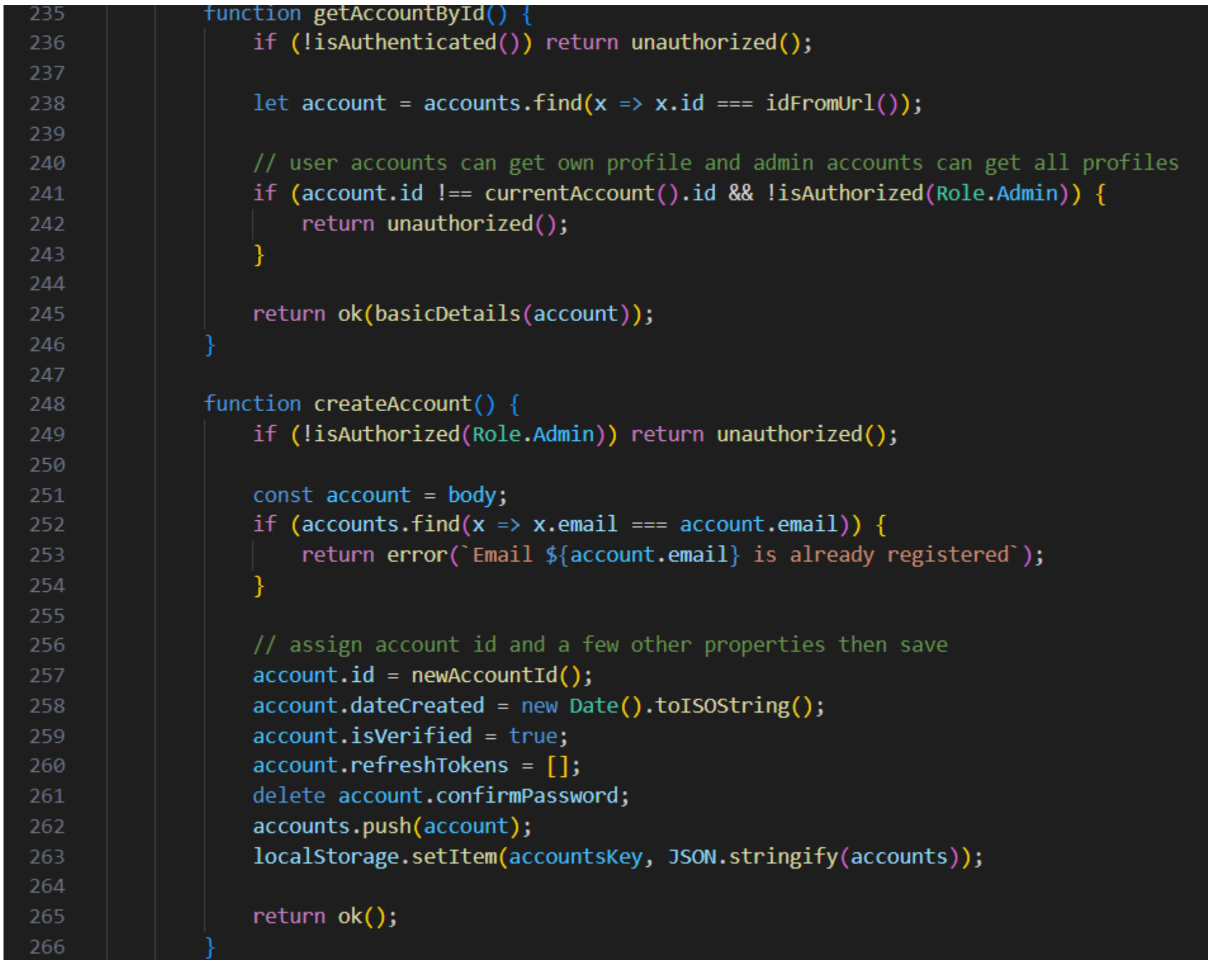
```
            account.dateCreated = new Date().toISOString();
            account.verificationToken = new Date().getTime().toString();
            account.isVerified = false;
            account.refreshTokens = [];
            delete account.confirmPassword;
            accounts.push(account);
            localStorage.setItem(accountsKey, JSON.stringify(accounts));

            // display verification email in alert
            setTimeout(() => {
                const verifyUrl = `${location.origin}/account/verify-email?token=${account.verificationToken}`;
                alertService.info(`
                    <h4>Verification Email</h4>
                    <p>Thanks for registering!</p>
                    <p>Please click the below link to verify your email address:</p>
                    <p><a href="${verifyUrl}">${verifyUrl}</a></p>
                    <div><strong>NOTE:</strong> The fake backend displayed this "email" so you can test without an api. A real backend would send a
                `, { autoClose: false });
            }, 1000);

            return ok();
        }

        function verifyEmail() {
            const { token } = body;
            const account = accounts.find(x => !!x.verificationToken && x.verificationToken === token);

            if (!account) return error('Verification failed');

            // set is verified flag to true if token is valid
            account.isVerified = true;
            localStorage.setItem(accountsKey, JSON.stringify(accounts));
```

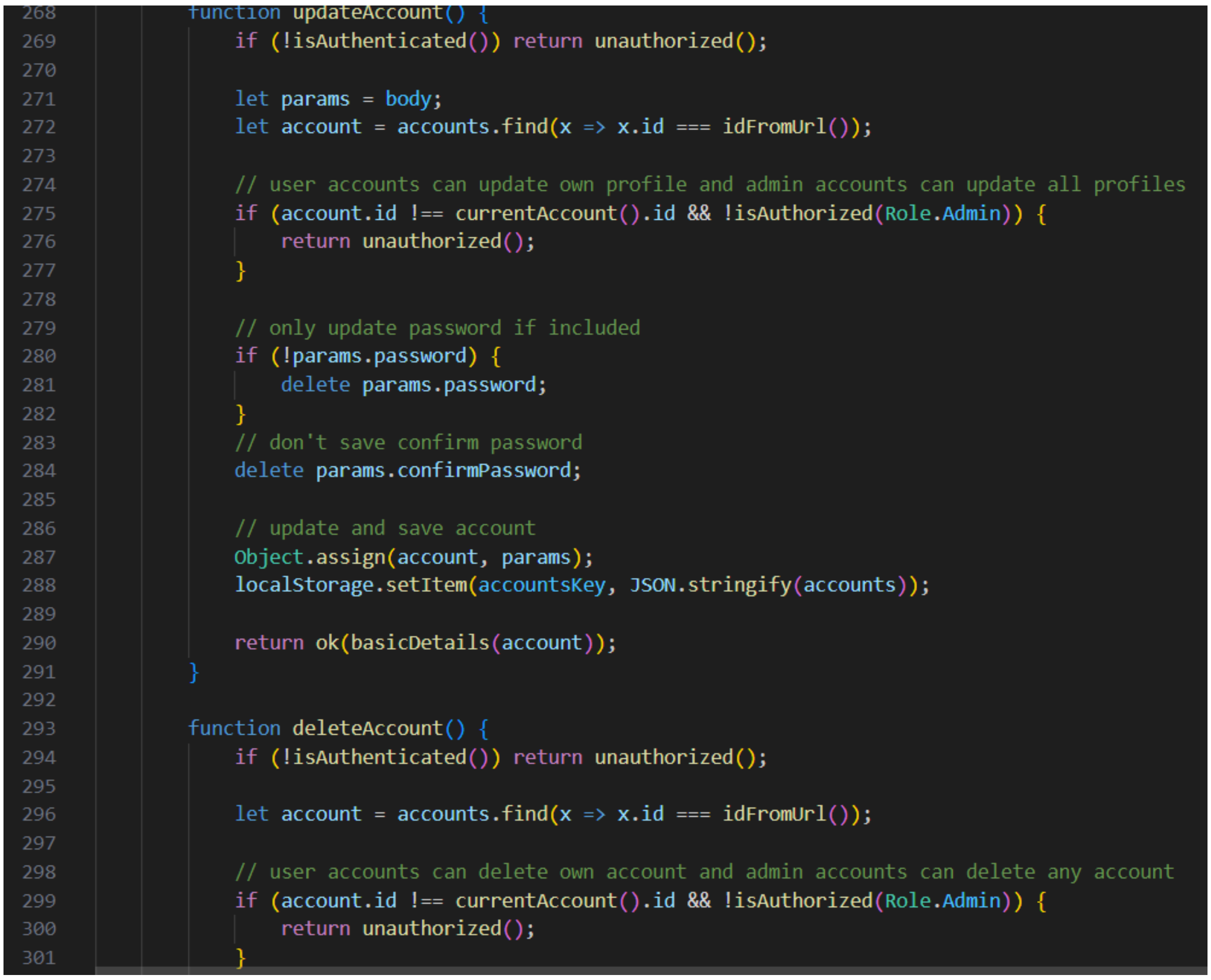


```
        return ok();
    }

    function forgotPassword() {
        const { email } = body;
        const account = accounts.find(x => x.email === email);

        // always return ok() response to prevent email enumeration
        if (!account) return ok();

        // create reset token that expires after 24 hours
        account.resetToken = new Date().getTime().toString();
        account.resetTokenExpires = new Date(Date.now() + 24*60*60*1000).toISOString();
        localStorage.setItem(accountsKey, JSON.stringify(accounts));

        // display password reset email in alert
        setTimeout(() => {
            const resetUrl = `${location.origin}/account/reset-password?token=${account.resetToken}`;
            alertService.info(`
                <h4>Reset Password Email</h4>
                <p>Please click the below link to reset your password, the link will be valid for 1 day:</p>
                <p><a href="${resetUrl}">${resetUrl}</a></p>
                <div><strong>NOTE:</strong> The fake backend displayed this "email" so you can test without an api. A real backend would send a
            `, { autoClose: false });
        }, 1000);

        return ok();
    }

    function validateResetToken() {
        const { token } = body;
        const account = accounts.find(x =>
            !!x.resetToken && x.resetToken === token &&
```

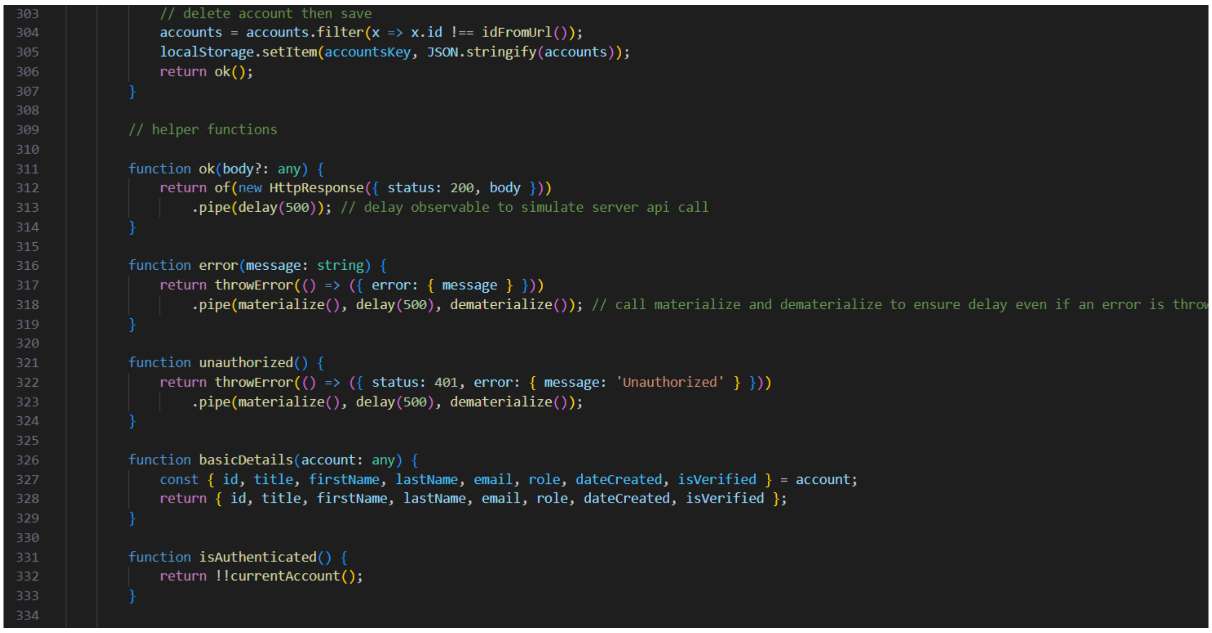


```
            !!x.resetToken && x.resetToken === token &&
            new Date() < new Date(x.resetTokenExpires)
        );

        if (!account) return error('Invalid token');

        return ok();
    }

    function resetPassword() {
        const { token, password } = body;
        const account = accounts.find(x =>
            !!x.resetToken && x.resetToken === token &&
            new Date() < new Date(x.resetTokenExpires)
        );

        if (!account) return error('Invalid token');

        // update password and remove reset token
        account.password = password;
        account.isVerified = true;
        delete account.resetToken;
        delete account.resetTokenExpires;
        localStorage.setItem(accountsKey, JSON.stringify(accounts));

        return ok();
    }

    function getAccounts() {
        if (!isAuthenticated()) return unauthorized();
        return ok(accounts.map(x => basicDetails(x)));
    }
```

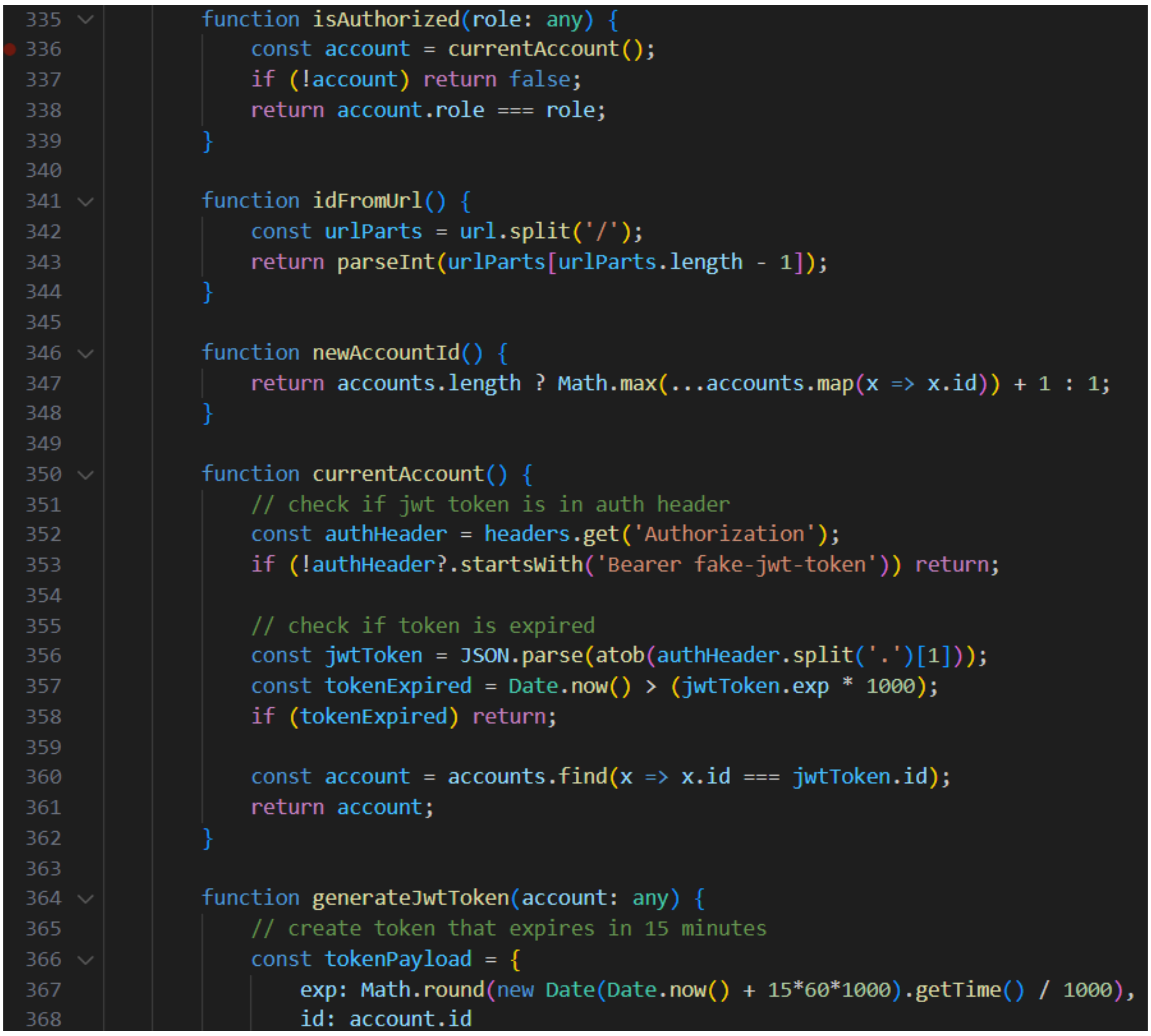

```
    function getAccountById() {
        if (!isAuthenticated()) return unauthorized();

        let account = accounts.find(x => x.id === idFromUrl());

        // user accounts can get own profile and admin accounts can get all profiles
        if (account.id !== currentAccount().id && !isAuthorized(Role.Admin)) {
            return unauthorized();
        }

        return ok(basicDetails(account));
    }

    function createAccount() {
        if (!isAuthorized(Role.Admin)) return unauthorized();

        const account = body;
        if (accounts.find(x => x.email === account.email)) {
            return error(`Email ${account.email} is already registered`);
        }

        // assign account id and a few other properties then save
        account.id = newAccountId();
        account.dateCreated = new Date().toISOString();
        account.isVerified = true;
        account.refreshTokens = [];
        delete account.confirmPassword;
        accounts.push(account);
        localStorage.setItem(accountsKey, JSON.stringify(accounts));

        return ok();
    }
```



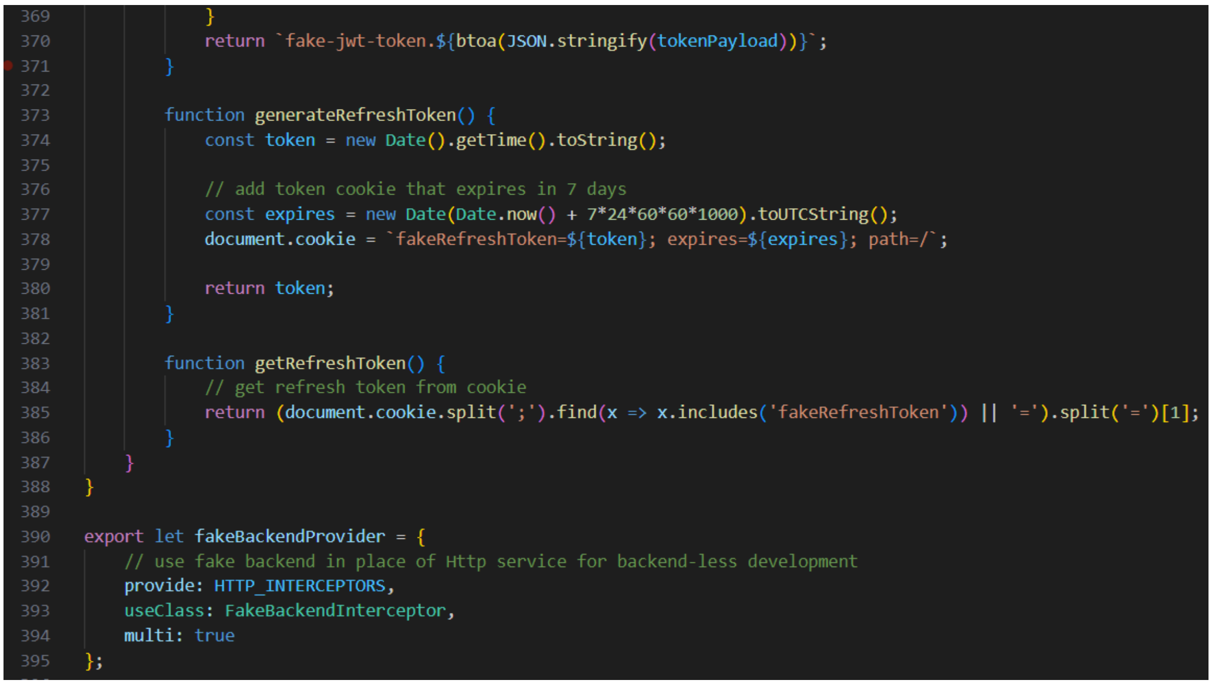
```
        function updateAccount() {
            if (!isAuthenticated()) return unauthorized();

            let params = body;
            let account = accounts.find(x => x.id === idFromUrl());

            // user accounts can update own profile and admin accounts can update all profiles
            if (account.id !== currentAccount().id && !isAuthorized(Role.Admin)) {
                return unauthorized();
            }

            // only update password if included
            if (!params.password) {
                delete params.password;
            }
            // don't save confirm password
            delete params.confirmPassword;

            // update and save account
            Object.assign(account, params);
            localStorage.setItem(accountsKey, JSON.stringify(accounts));

            return ok(basicDetails(account));
        }

        function deleteAccount() {
            if (!isAuthenticated()) return unauthorized();

            let account = accounts.find(x => x.id === idFromUrl());

            // user accounts can delete own account and admin accounts can delete any account
            if (account.id !== currentAccount().id && !isAuthorized(Role.Admin)) {
                return unauthorized();
            }
```



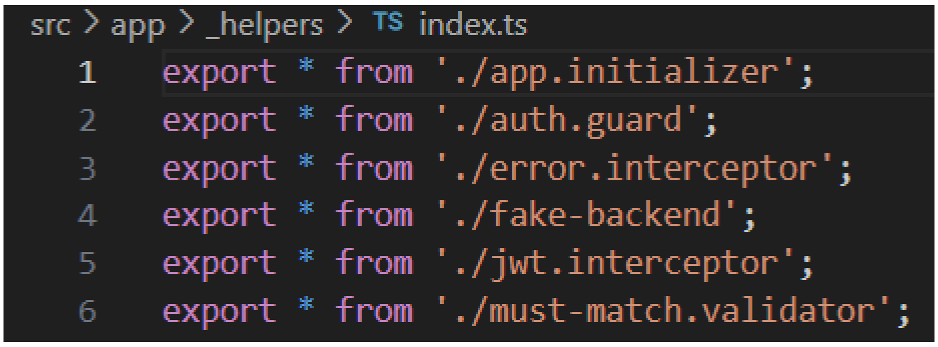
```
            // delete account then save
            accounts = accounts.filter(x => x.id !== idFromUrl());
            localStorage.setItem(accountsKey, JSON.stringify(accounts));
            return ok();
        }

        // helper functions

        function ok(body?: any) {
            return of(new HttpResponse({ status: 200, body }))
                .pipe(delay(500)); // delay observable to simulate server api call
        }

        function error(message: string) {
            return throwError(() => ({ error: { message } }))
                .pipe(materialize(), delay(500), dematerialize()); // call materialize and dematerialize to ensure delay even if an error is throw
        }

        function unauthorized() {
            return throwError(() => ({ status: 401, error: { message: 'Unauthorized' } }))
                .pipe(materialize(), delay(500), dematerialize());
        }

        function basicDetails(account: any) {
            const { id, title, firstName, lastName, email, role, dateCreated, isVerified } = account;
            return { id, title, firstName, lastName, email, role, dateCreated, isVerified };
        }

        function isAuthenticated() {
            return !!currentAccount();
        }

```



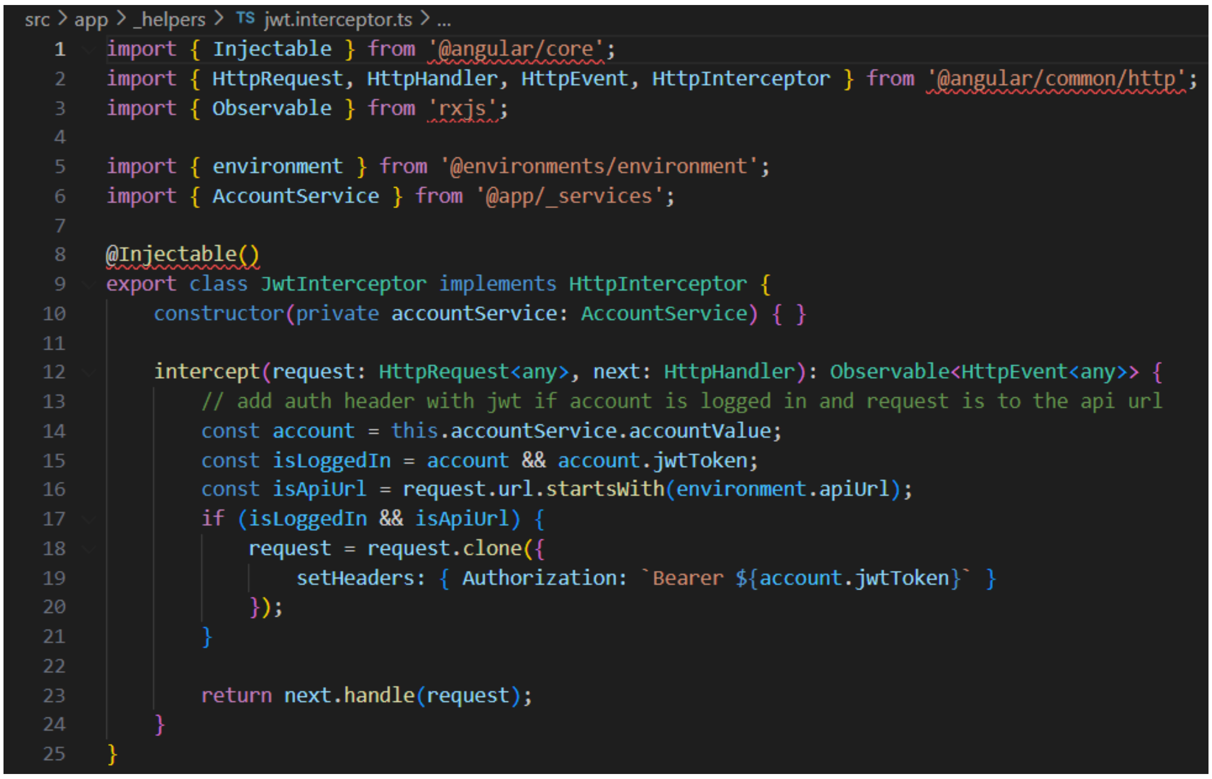
```
        function isAuthorized(role: any) {
            const account = currentAccount();
            if (!account) return false;
            return account.role === role;
        }

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

        function newAccountId() {
            return accounts.length ? Math.max(...accounts.map(x => x.id)) + 1 : 1;
        }

        function currentAccount() {
            // check if jwt token is in auth header
            const authHeader = headers.get('Authorization');
            if (!authHeader?.startsWith('Bearer fake-jwt-token')) return;

            // check if token is expired
            const jwtToken = JSON.parse(atob(authHeader.split('.')[1]));
            const tokenExpired = Date.now() > (jwtToken.exp * 1000);
            if (tokenExpired) return;

            const account = accounts.find(x => x.id === jwtToken.id);
            return account;
        }

        function generateJwtToken(account: any) {
            // create token that expires in 15 minutes
            const tokenPayload = {
                exp: Math.round(new Date(Date.now() + 15*60*1000).getTime() / 1000),
                id: account.id
```



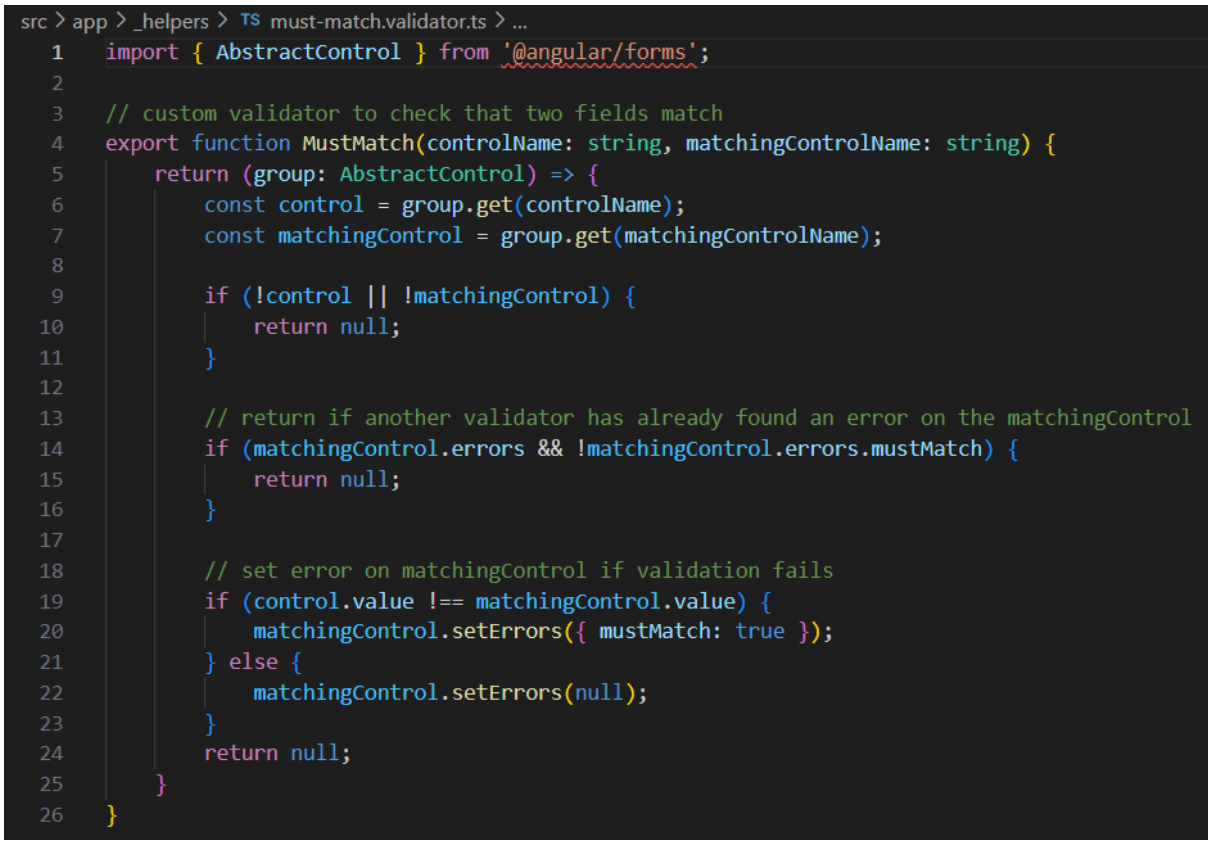
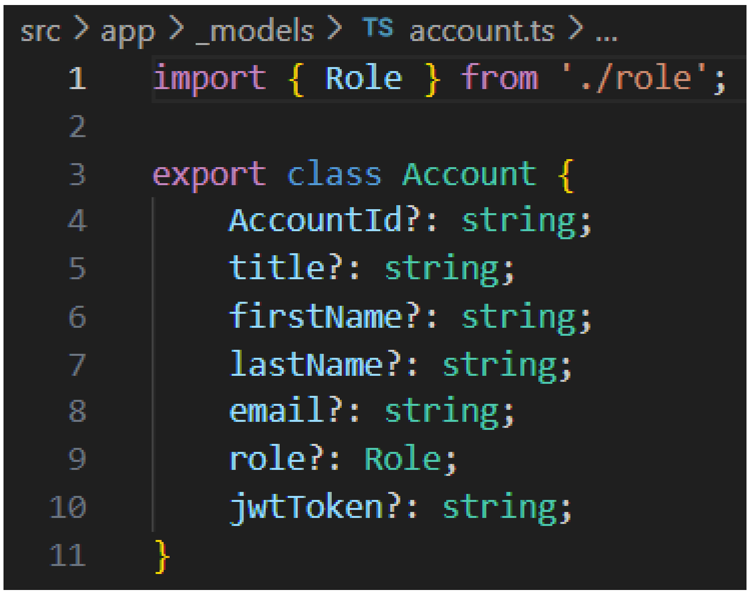
```
            }
            return `fake-jwt-token.${btoa(JSON.stringify(tokenPayload))}`;
        }

        function generateRefreshToken() {
            const token = new Date().getTime().toString();

            // add token cookie that expires in 7 days
            const expires = new Date(Date.now() + 7*24*60*60*1000).toUTCString();
            document.cookie = `fakeRefreshToken=${token}; expires=${expires}; path=/`;

            return token;
        }

        function getRefreshToken() {
            // get refresh token from cookie
            return (document.cookie.split(';').find(x => x.includes('fakeRefreshToken')) || '=').split('=')[1];
        }
    }
}

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

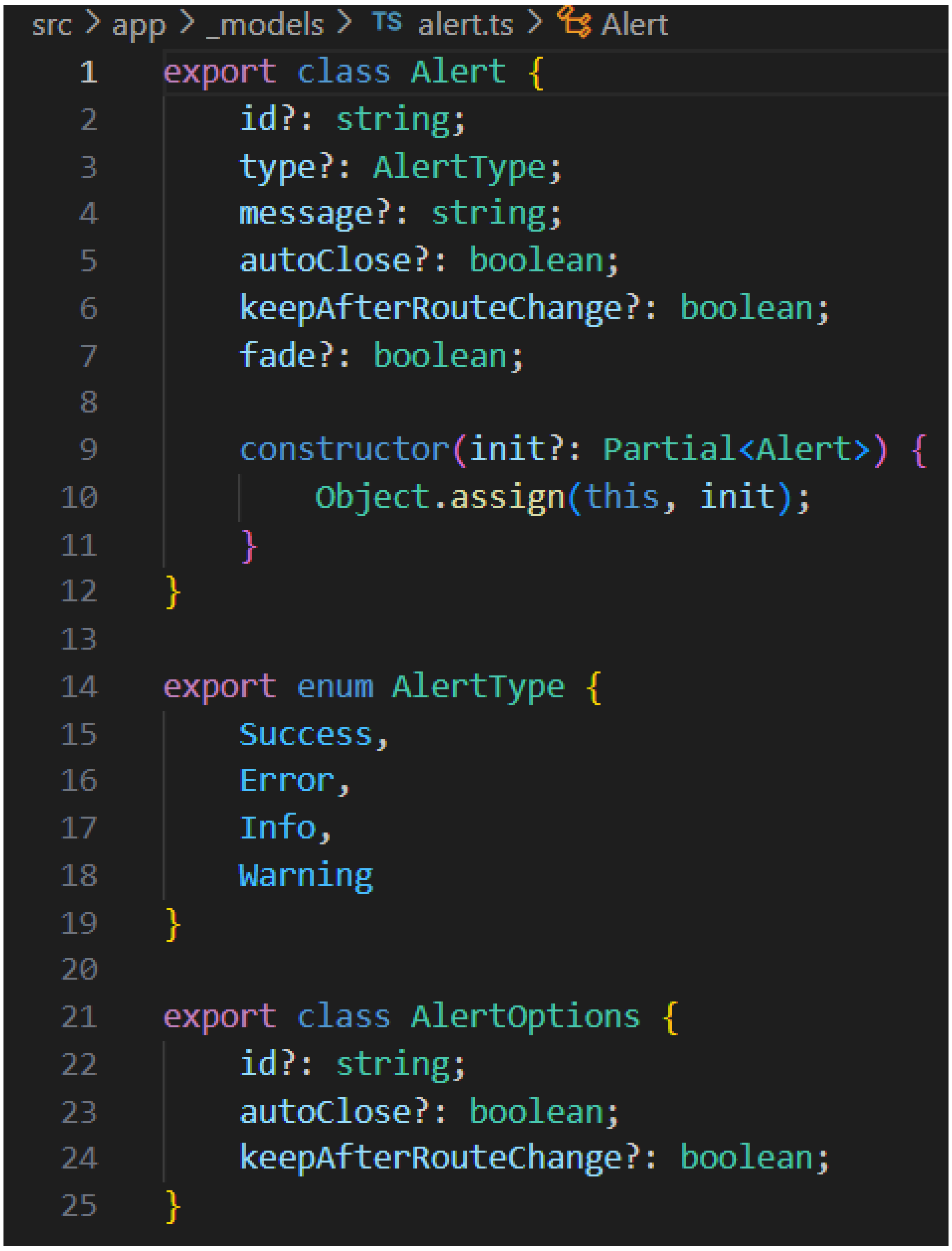

src > app > _helpers > TS index.ts

```
export * from './app.initializer';
export * from './auth.guard';
export * from './error.interceptor';
export * from './fake-backend';
export * from './jwt.interceptor';
export * from './must-match.validator';
```

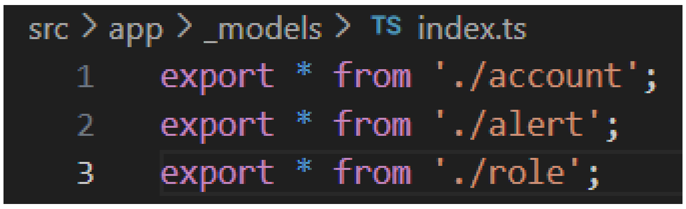

src > app > _helpers > TS jwt.interceptor.ts > ...

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

import { environment } from '@environments/environment';
import { AccountService } from '@app/_services';

@Injectable()
export class JwtInterceptor implements HttpInterceptor {
    constructor(private accountService: AccountService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        // add auth header with jwt if account is logged in and request is to the api url
        const account = this.accountService.accountValue;
        const isLoggedIn = account && account.jwtToken;
        const isApiUrl = request.url.startsWith(environment.apiUrl);
        if (isLoggedIn && isApiUrl) {
            request = request.clone({
                setHeaders: { Authorization: `Bearer ${account.jwtToken}` }
            });
        }

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

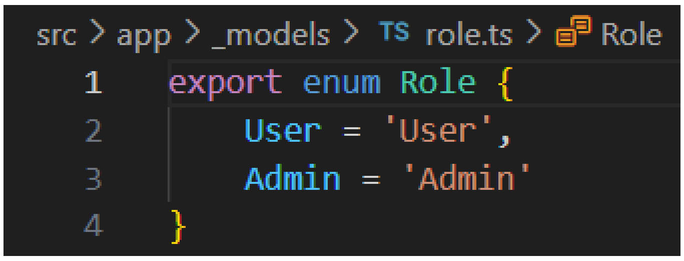

src > app > _helpers > TS must-match.validator.ts > ...
```typescript
import { AbstractControl } from '@angular/forms';

// custom validator to check that two fields match
export function MustMatch(controlName: string, matchingControlName: string) {
    return (group: AbstractControl) => {
        const control = group.get(controlName);
        const matchingControl = group.get(matchingControlName);

        if (!control || !matchingControl) {
            return null;
        }

        // return if another validator has already found an error on the matchingControl
        if (matchingControl.errors && !matchingControl.errors.mustMatch) {
            return null;
        }

        // set error on matchingControl if validation fails
        if (control.value !== matchingControl.value) {
            matchingControl.setErrors({ mustMatch: true });
        } else {
            matchingControl.setErrors(null);
        }
        return null;
    }
}
```

src > app > _models > TS account.ts > ...
```typescript
import { Role } from './role';

export class Account {
    AccountId?: string;
    title?: string;
    firstName?: string;
    lastName?: string;
    email?: string;
    role?: Role;
    jwtToken?: string;
}
```


src > app > _models > TS alert.ts > Alert

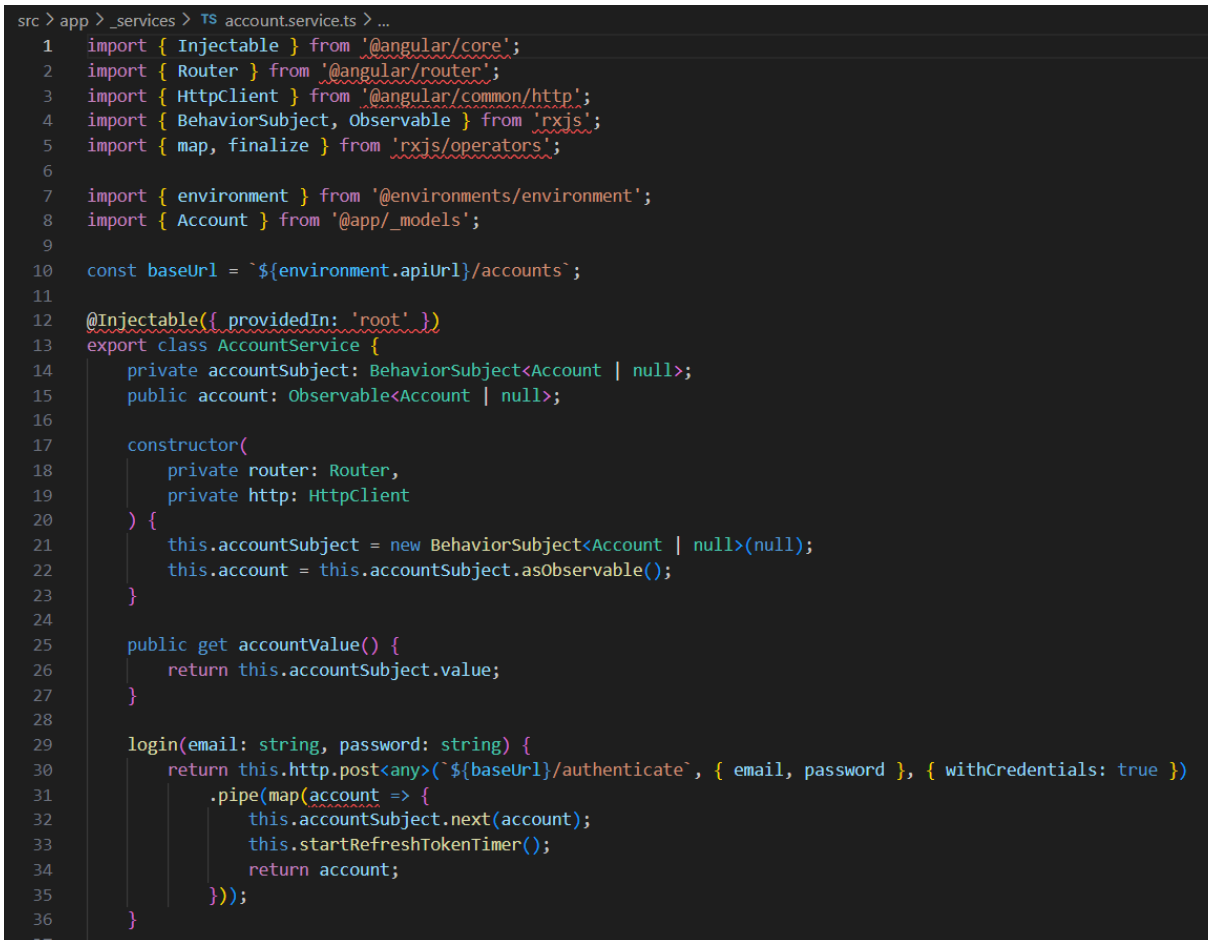
```typescript
export class Alert {
    id?: string;
    type?: AlertType;
    message?: string;
    autoClose?: boolean;
    keepAfterRouteChange?: boolean;
    fade?: boolean;

    constructor(init?: Partial<Alert>) {
        Object.assign(this, init);
    }
}

export enum AlertType {
    Success,
    Error,
    Info,
    Warning
}

export class AlertOptions {
    id?: string;
    autoClose?: boolean;
    keepAfterRouteChange?: boolean;
}
```


src > app > _models > TS index.ts

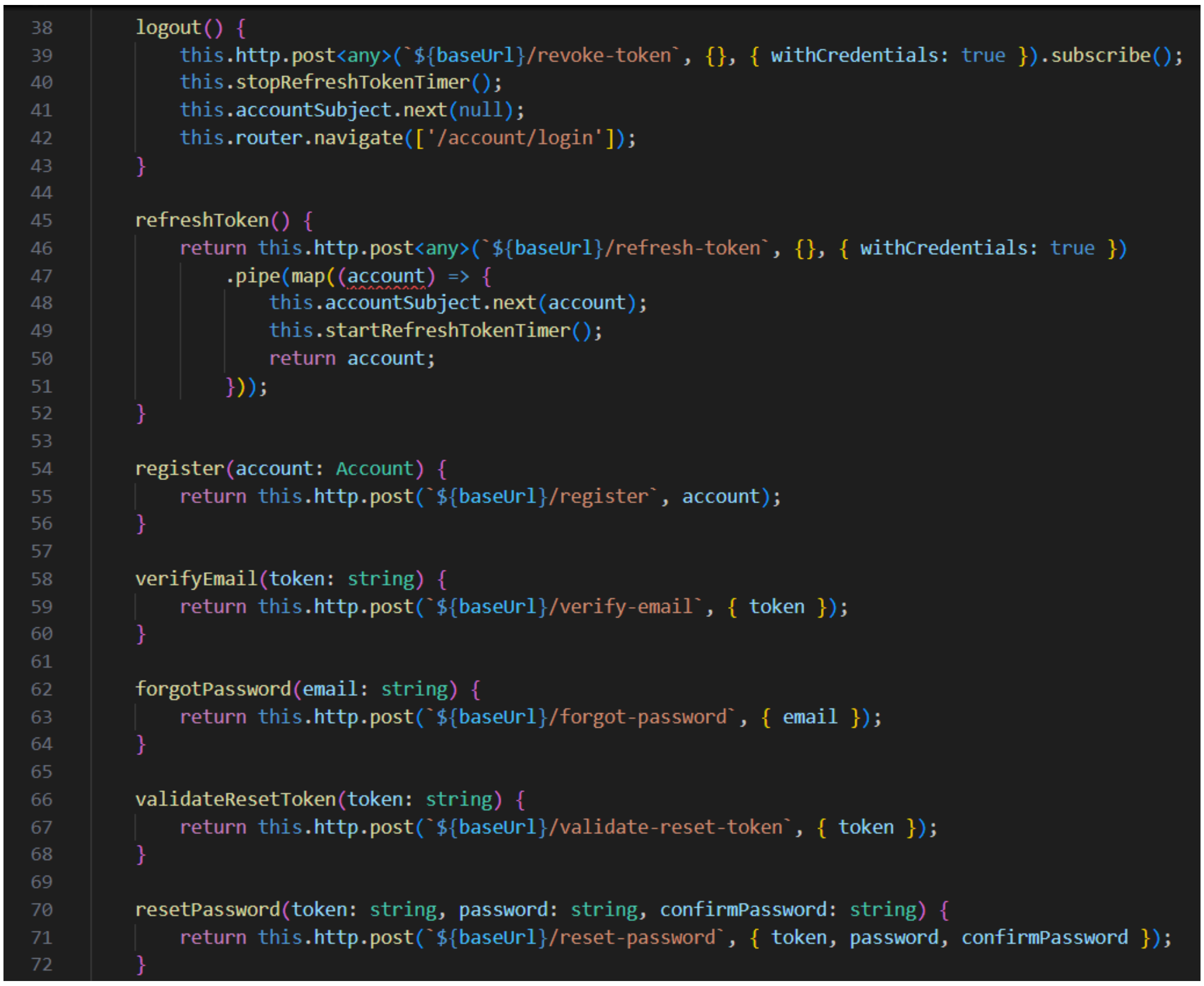
```typescript
export * from './account';
export * from './alert';
export * from './role';
```


src > app > _models > TS role.ts > Role

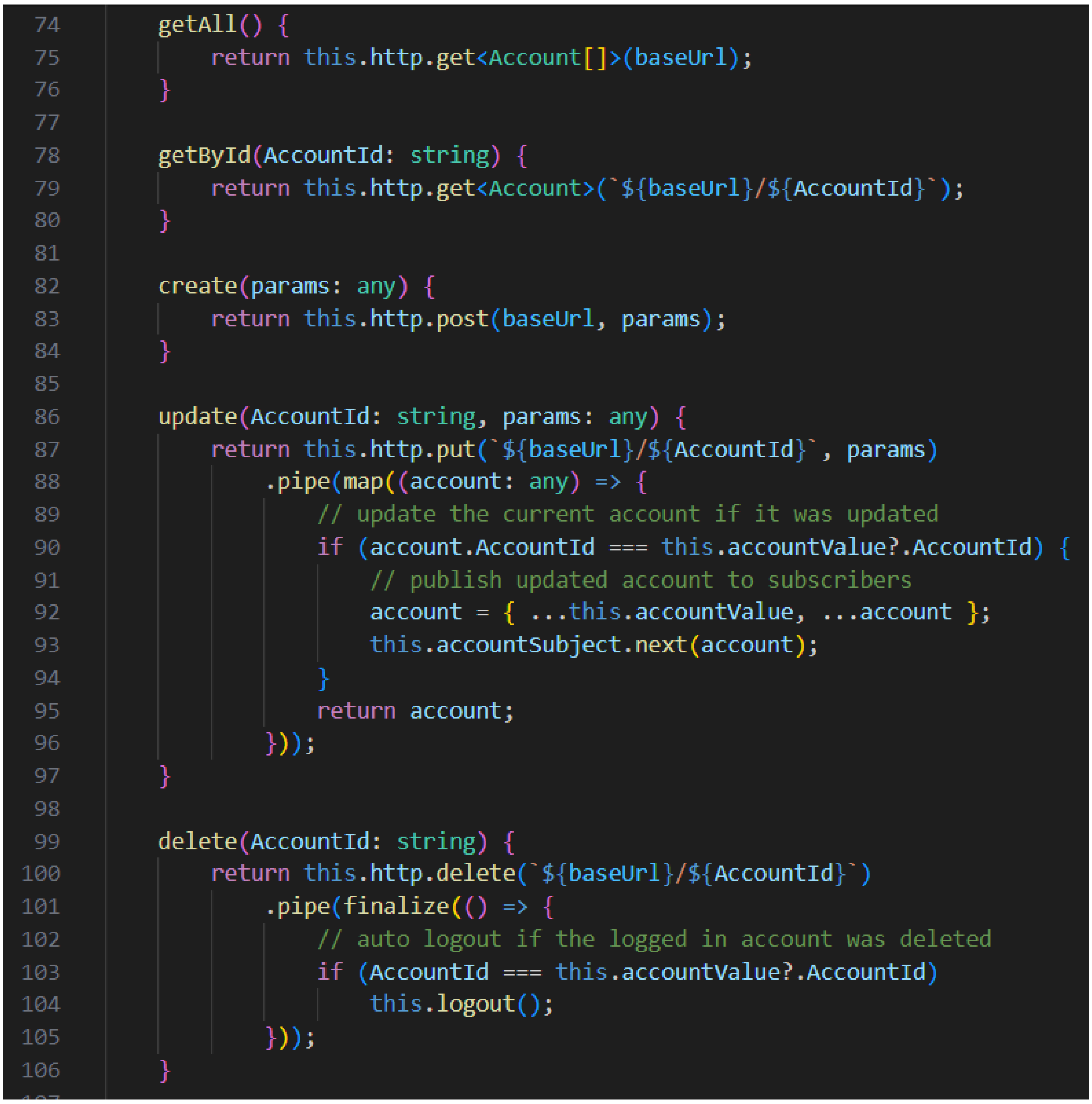
```typescript
export enum Role {
    User = 'User',
    Admin = 'Admin'
}
```


src > app > _services > TS account.service.ts > ...

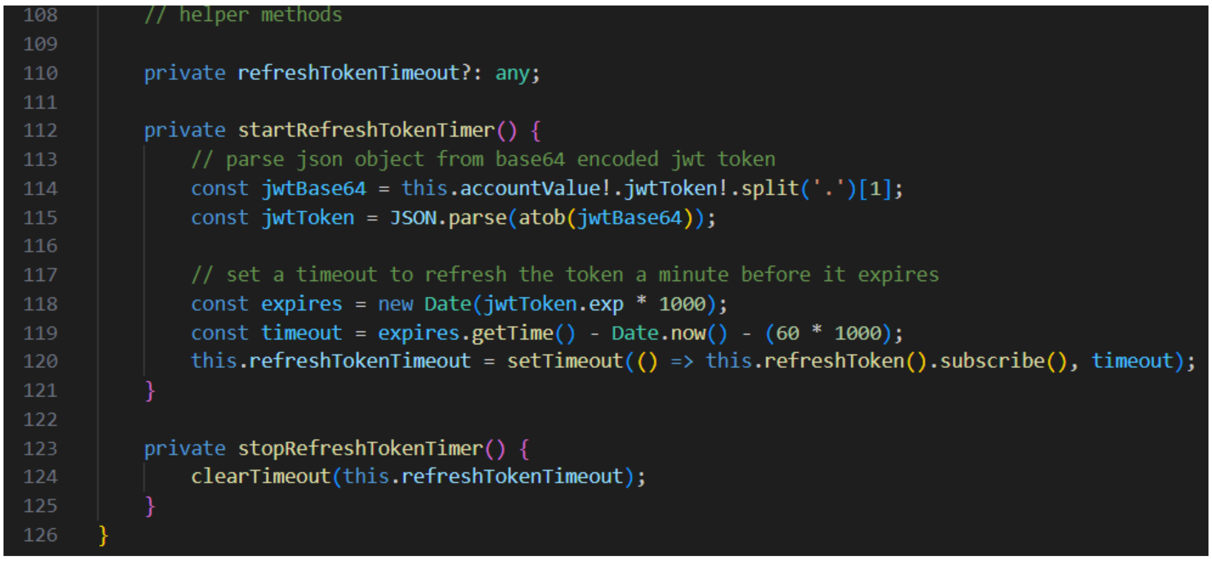
```typescript
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map, finalize } from 'rxjs/operators';

import { environment } from '@environments/environment';
import { Account } from '@app/_models';

const baseUrl = `${environment.apiUrl}/accounts`;

@Injectable({ providedIn: 'root' })
export class AccountService {
    private accountSubject: BehaviorSubject<Account | null>;
    public account: Observable<Account | null>;

    constructor(
        private router: Router,
        private http: HttpClient
    ) {
        this.accountSubject = new BehaviorSubject<Account | null>(null);
        this.account = this.accountSubject.asObservable();
    }

    public get accountValue() {
        return this.accountSubject.value;
    }

    login(email: string, password: string) {
        return this.http.post<any>(`${baseUrl}/authenticate`, { email, password }, { withCredentials: true })
            .pipe(map(account => {
                this.accountSubject.next(account);
                this.startRefreshTokenTimer();
                return account;
            }));
    }
```



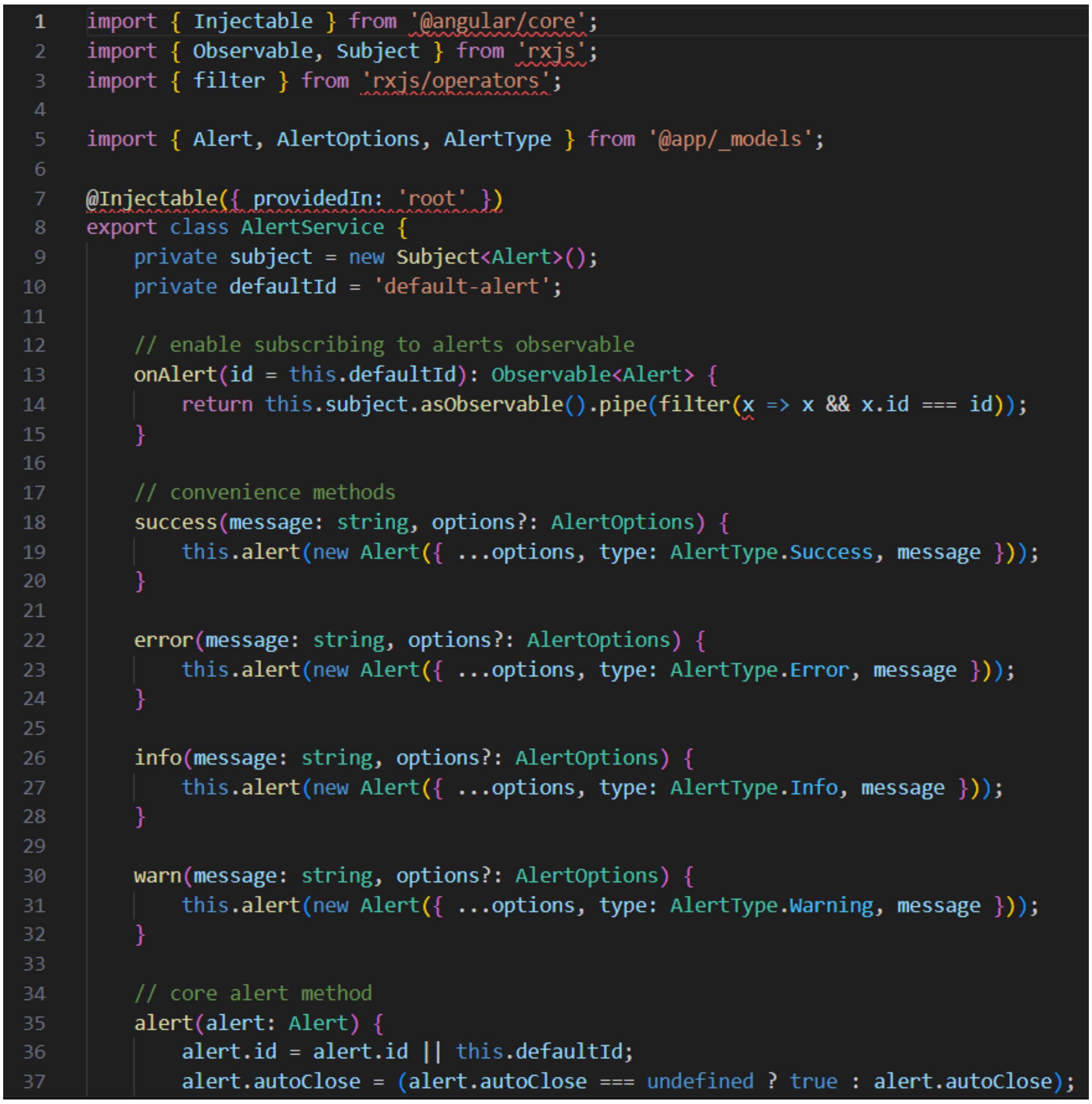
```
    logout() {
        this.http.post<any>(`${baseUrl}/revoke-token`, {}, { withCredentials: true }).subscribe();
        this.stopRefreshTokenTimer();
        this.accountSubject.next(null);
        this.router.navigate(['/account/login']);
    }

    refreshToken() {
        return this.http.post<any>(`${baseUrl}/refresh-token`, {}, { withCredentials: true })
            .pipe(map((account) => {
                this.accountSubject.next(account);
                this.startRefreshTokenTimer();
                return account;
            }));
    }

    register(account: Account) {
        return this.http.post(`${baseUrl}/register`, account);
    }

    verifyEmail(token: string) {
        return this.http.post(`${baseUrl}/verify-email`, { token });
    }

    forgotPassword(email: string) {
        return this.http.post(`${baseUrl}/forgot-password`, { email });
    }

    validateResetToken(token: string) {
        return this.http.post(`${baseUrl}/validate-reset-token`, { token });
    }

    resetPassword(token: string, password: string, confirmPassword: string) {
        return this.http.post(`${baseUrl}/reset-password`, { token, password, confirmPassword });
    }
```



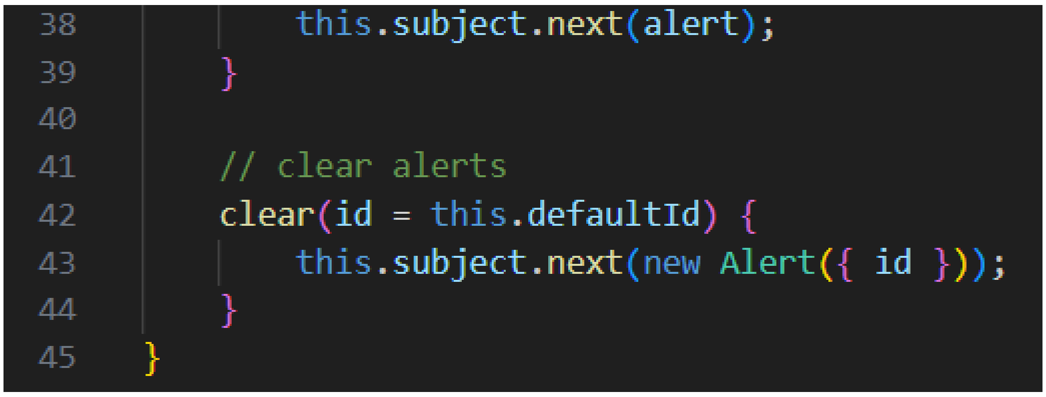
```
    getAll() {
        return this.http.get<Account[]>(baseUrl);
    }

    getById(AccountId: string) {
        return this.http.get<Account>(`${baseUrl}/${AccountId}`);
    }

    create(params: any) {
        return this.http.post(baseUrl, params);
    }

    update(AccountId: string, params: any) {
        return this.http.put(`${baseUrl}/${AccountId}`, params)
            .pipe(map((account: any) => {
                // update the current account if it was updated
                if (account.AccountId === this.accountValue?.AccountId) {
                    // publish updated account to subscribers
                    account = { ...this.accountValue, ...account };
                    this.accountSubject.next(account);
                }
                return account;
            }));
    }

    delete(AccountId: string) {
        return this.http.delete(`${baseUrl}/${AccountId}`)
            .pipe(finalize(() => {
                // auto logout if the logged in account was deleted
                if (AccountId === this.accountValue?.AccountId)
                    this.logout();
            }));
    }
```



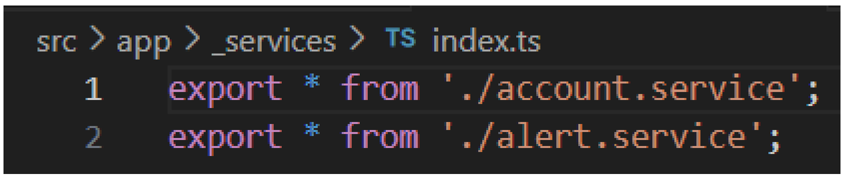
```
    // helper methods

    private refreshTokenTimeout?: any;

    private startRefreshTokenTimer() {
        // parse json object from base64 encoded jwt token
        const jwtBase64 = this.accountValue!.jwtToken!.split('.')[1];
        const jwtToken = JSON.parse(atob(jwtBase64));

        // set a timeout to refresh the token a minute before it expires
        const expires = new Date(jwtToken.exp * 1000);
        const timeout = expires.getTime() - Date.now() - (60 * 1000);
        this.refreshTokenTimeout = setTimeout(() => this.refreshToken().subscribe(), timeout);
    }

    private stopRefreshTokenTimer() {
        clearTimeout(this.refreshTokenTimeout);
    }
}
```



```
import { Injectable } from '@angular/core';
import { Observable, Subject } from 'rxjs';
import { filter } from 'rxjs/operators';

import { Alert, AlertOptions, AlertType } from '@app/_models';

@Injectable({ providedIn: 'root' })
export class AlertService {
    private subject = new Subject<Alert>();
    private defaultId = 'default-alert';

    // enable subscribing to alerts observable
    onAlert(id = this.defaultId): Observable<Alert> {
        return this.subject.asObservable().pipe(filter(x => x && x.id === id));
    }

    // convenience methods
    success(message: string, options?: AlertOptions) {
        this.alert(new Alert({ ...options, type: AlertType.Success, message }));
    }

    error(message: string, options?: AlertOptions) {
        this.alert(new Alert({ ...options, type: AlertType.Error, message }));
    }

    info(message: string, options?: AlertOptions) {
        this.alert(new Alert({ ...options, type: AlertType.Info, message }));
    }

    warn(message: string, options?: AlertOptions) {
        this.alert(new Alert({ ...options, type: AlertType.Warning, message }));
    }

    // core alert method
    alert(alert: Alert) {
        alert.id = alert.id || this.defaultId;
        alert.autoClose = (alert.autoClose === undefined ? true : alert.autoClose);
```

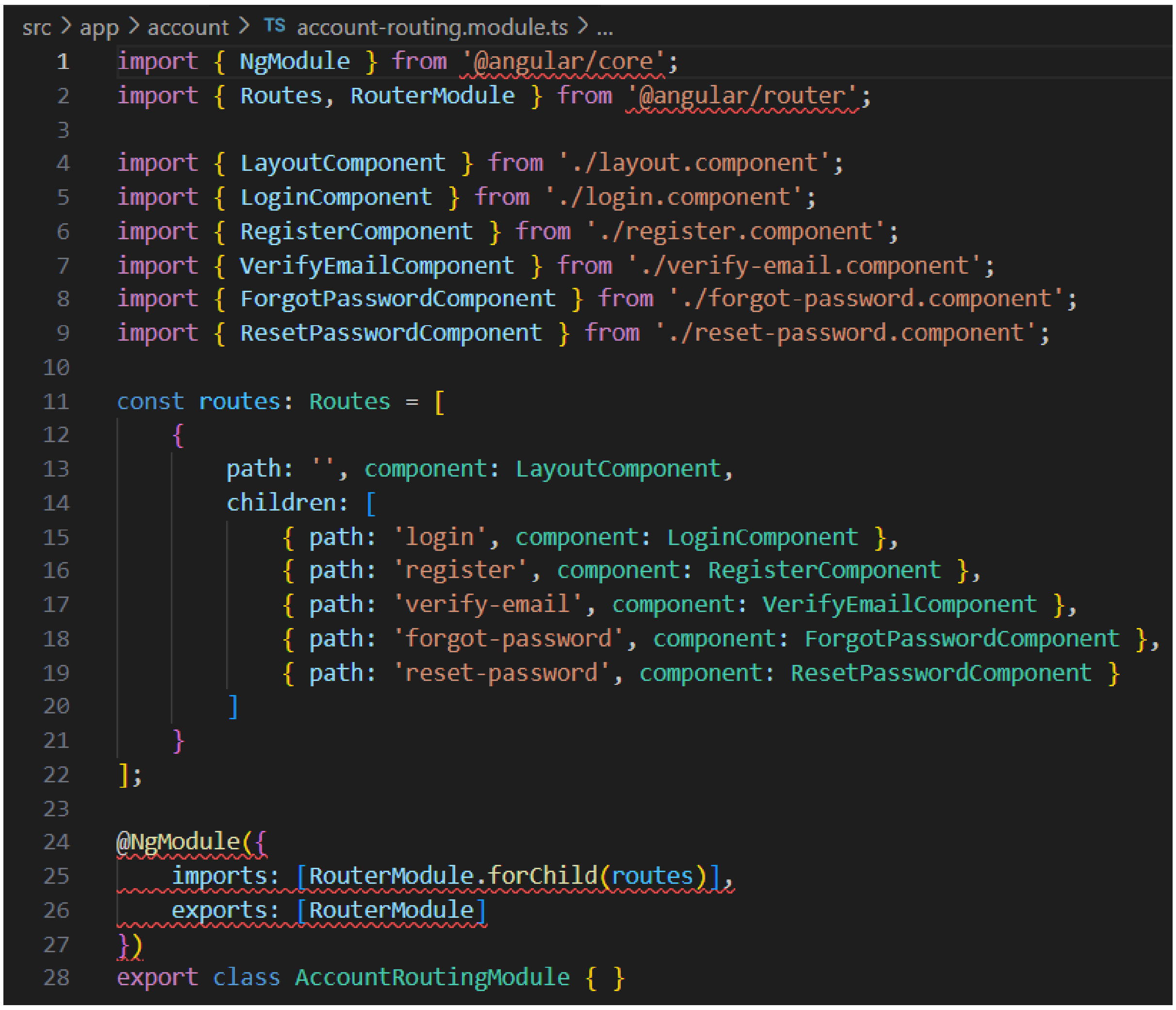


```
        this.subject.next(alert);
    }

    // clear alerts
    clear(id = this.defaultId) {
        this.subject.next(new Alert({ id }));
    }
}
```

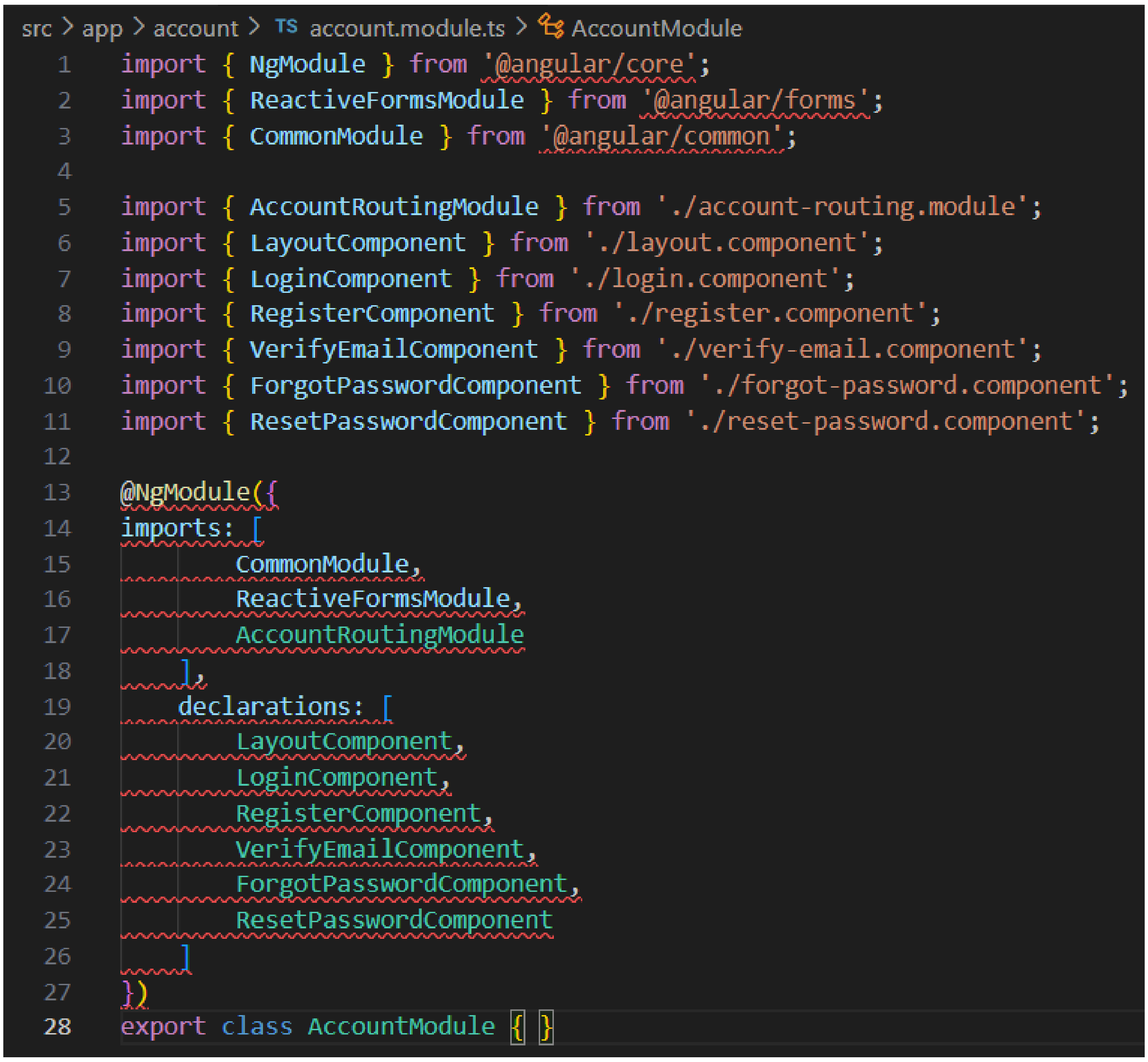


src > app > _services > TS index.ts

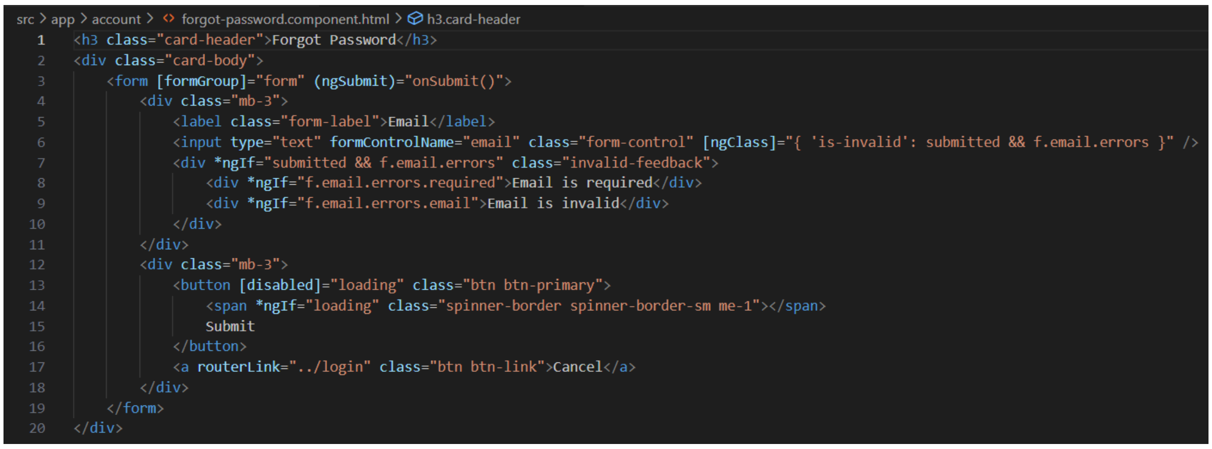
```
export * from './account.service';
export * from './alert.service';
```


src > app > account > TS account-routing.module.ts > ...

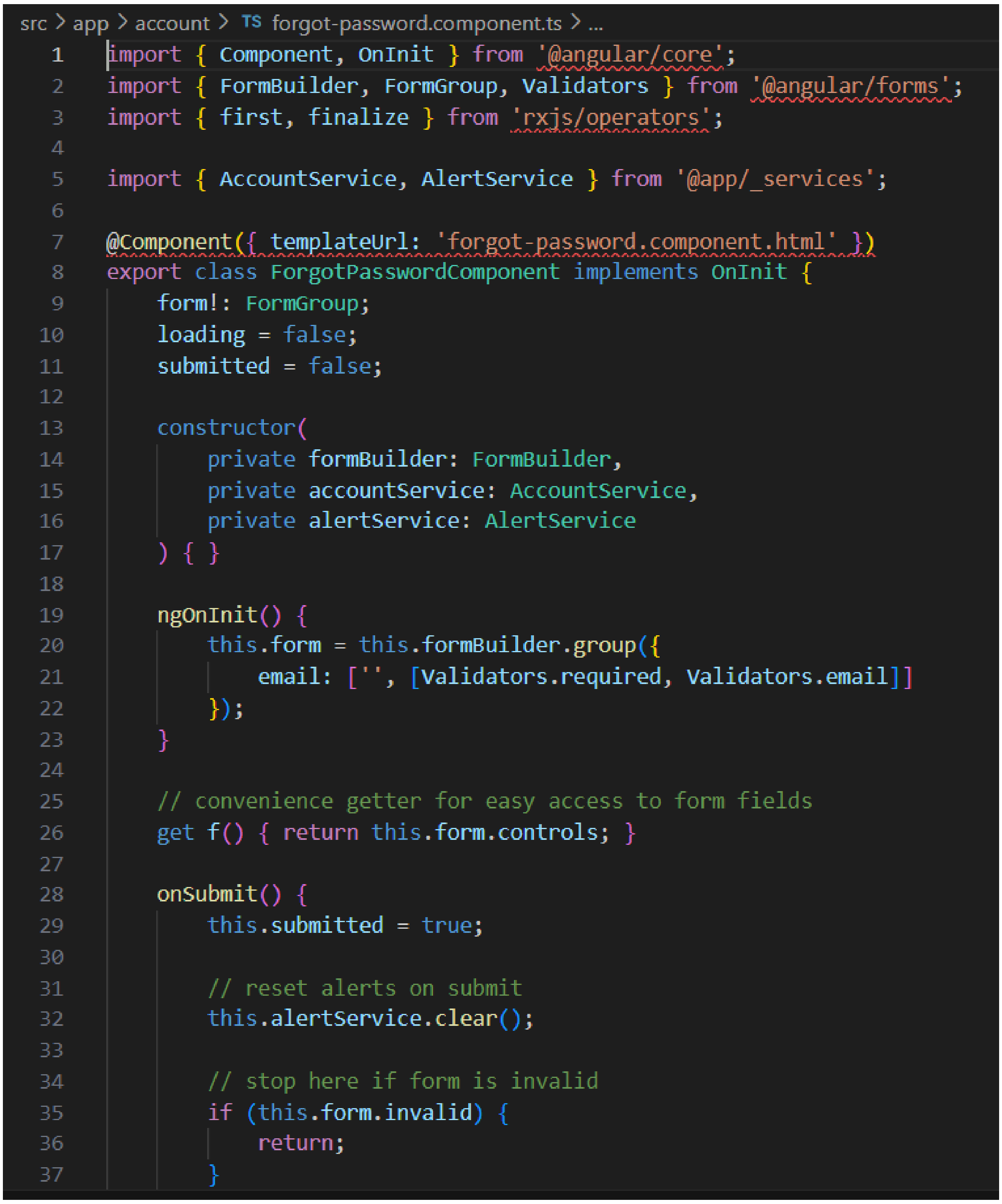
```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { LayoutComponent } from './layout.component';
import { LoginComponent } from './login.component';
import { RegisterComponent } from './register.component';
import { VerifyEmailComponent } from './verify-email.component';
import { ForgotPasswordComponent } from './forgot-password.component';
import { ResetPasswordComponent } from './reset-password.component';

const routes: Routes = [
    {
        path: '', component: LayoutComponent,
        children: [
            { path: 'login', component: LoginComponent },
            { path: 'register', component: RegisterComponent },
            { path: 'verify-email', component: VerifyEmailComponent },
            { path: 'forgot-password', component: ForgotPasswordComponent },
            { path: 'reset-password', component: ResetPasswordComponent }
        ]
    }
];

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



```typescript
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';

import { AccountRoutingModule } from './account-routing.module';
import { LayoutComponent } from './layout.component';
import { LoginComponent } from './login.component';
import { RegisterComponent } from './register.component';
import { VerifyEmailComponent } from './verify-email.component';
import { ForgotPasswordComponent } from './forgot-password.component';
import { ResetPasswordComponent } from './reset-password.component';

@NgModule({
imports: [
            CommonModule,
            ReactiveFormsModule,
            AccountRoutingModule
        ],
        declarations: [
            LayoutComponent,
            LoginComponent,
            RegisterComponent,
            VerifyEmailComponent,
            ForgotPasswordComponent,
            ResetPasswordComponent
        ]
})
export class AccountModule { }
```

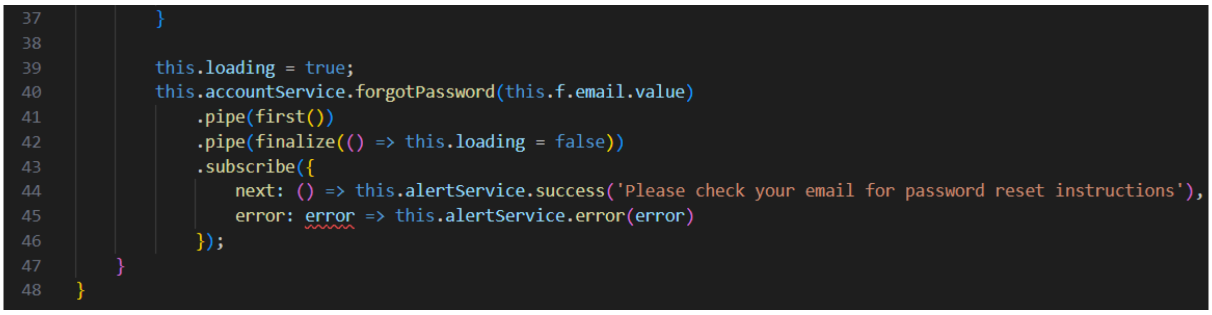

src > app > account > forgot-password.component.html > h3.card-header

```html
<h3 class="card-header">Forgot Password</h3>
<div class="card-body">
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
        <div class="mb-3">
            <label class="form-label">Email</label>
            <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }" />
            <div *ngIf="submitted && f.email.errors" class="invalid-feedback">
                <div *ngIf="f.email.errors.required">Email is required</div>
                <div *ngIf="f.email.errors.email">Email is invalid</div>
            </div>
        </div>
        <div class="mb-3">
            <button [disabled]="loading" class="btn btn-primary">
                <span *ngIf="loading" class="spinner-border spinner-border-sm me-1"></span>
                Submit
            </button>
            <a routerLink="../login" class="btn btn-link">Cancel</a>
        </div>
    </form>
</div>
```

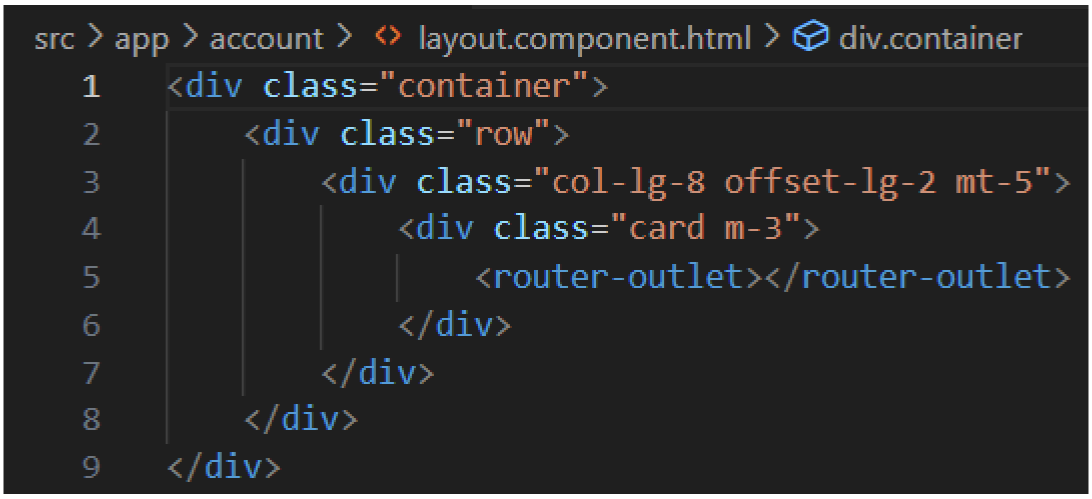


src > app > account > TS forgot-password.component.ts > ...

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

import { AccountService, AlertService } from '@app/_services';

@Component({ templateUrl: 'forgot-password.component.html' })
export class ForgotPasswordComponent implements OnInit {
    form!: FormGroup;
    loading = false;
    submitted = false;

    constructor(
        private formBuilder: FormBuilder,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            email: ['', [Validators.required, Validators.email]]
        });
    }

    // convenience getter for easy access to form fields
    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        // reset alerts on submit
        this.alertService.clear();

        // stop here if form is invalid
        if (this.form.invalid) {
            return;
        }
```

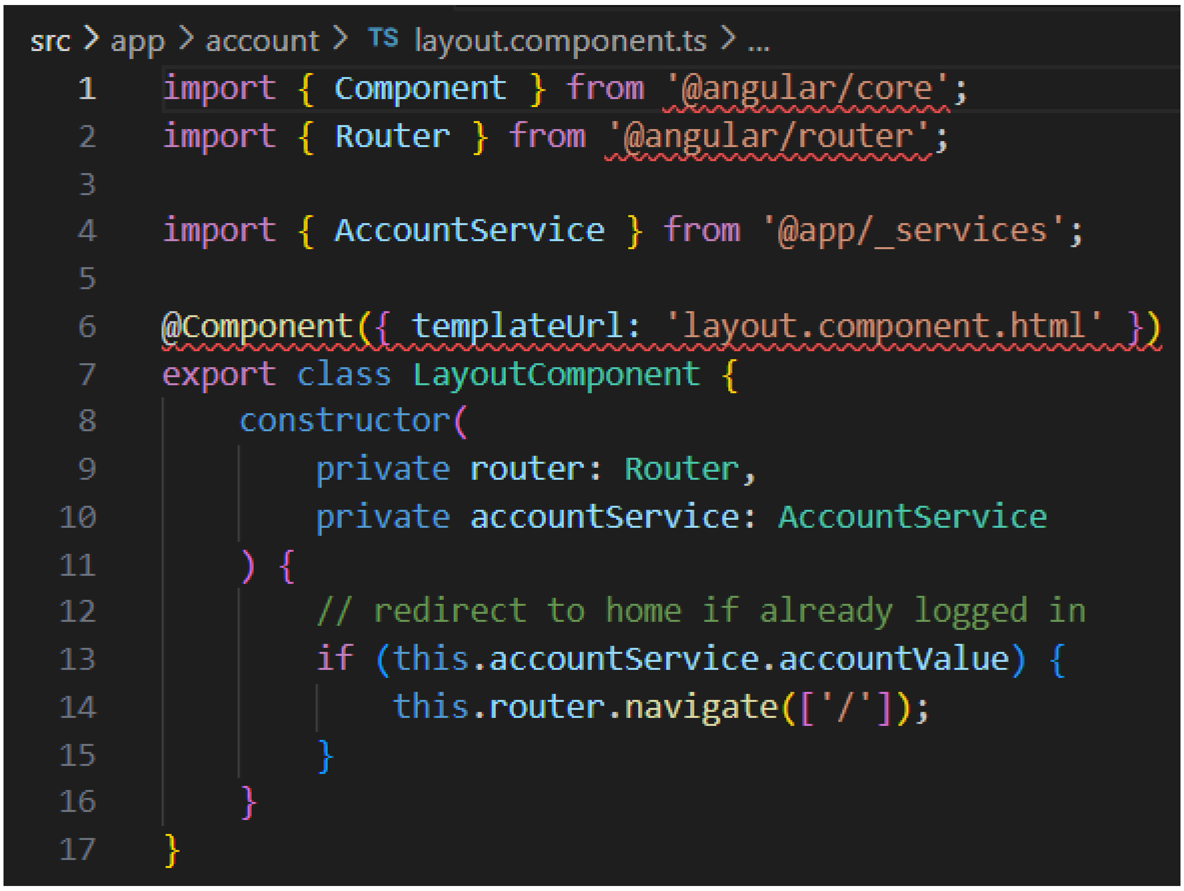


```typescript
        }

        this.loading = true;
        this.accountService.forgotPassword(this.f.email.value)
            .pipe(first())
            .pipe(finalize(() => this.loading = false))
            .subscribe({
                next: () => this.alertService.success('Please check your email for password reset instructions'),
                error: error => this.alertService.error(error)
            });
    }
}
```



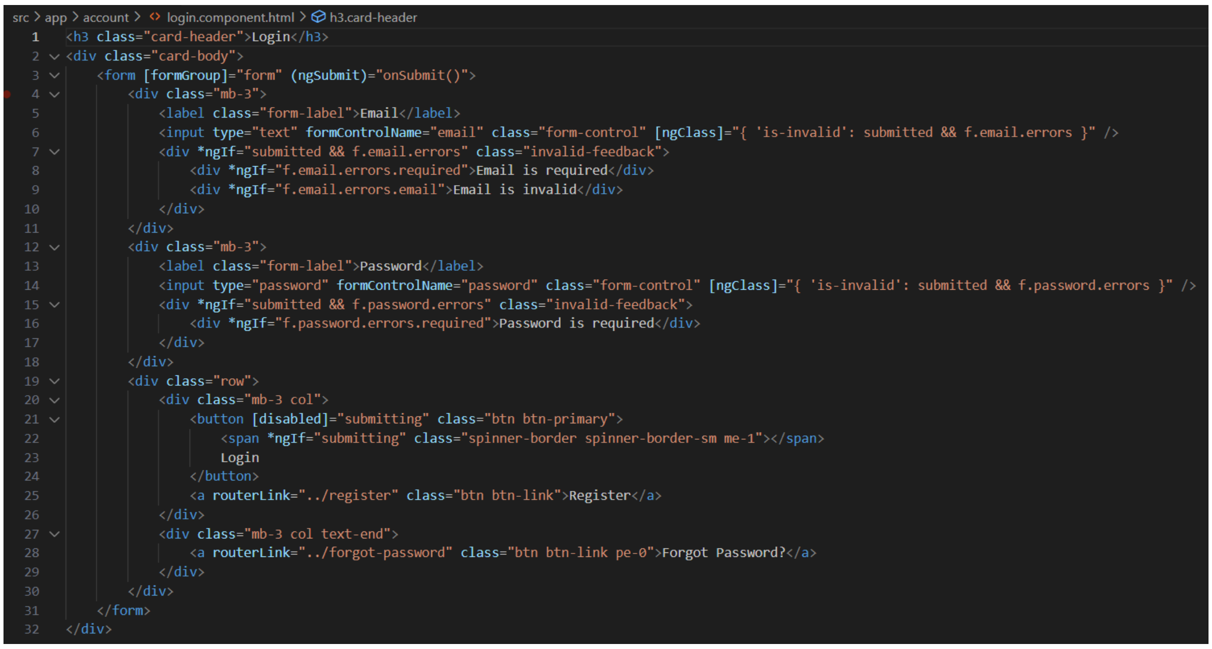
src > app > account > layout.component.html > div.container

```html
<div class="container">
    <div class="row">
        <div class="col-lg-8 offset-lg-2 mt-5">
            <div class="card m-3">
                <router-outlet></router-outlet>
            </div>
        </div>
    </div>
</div>
```



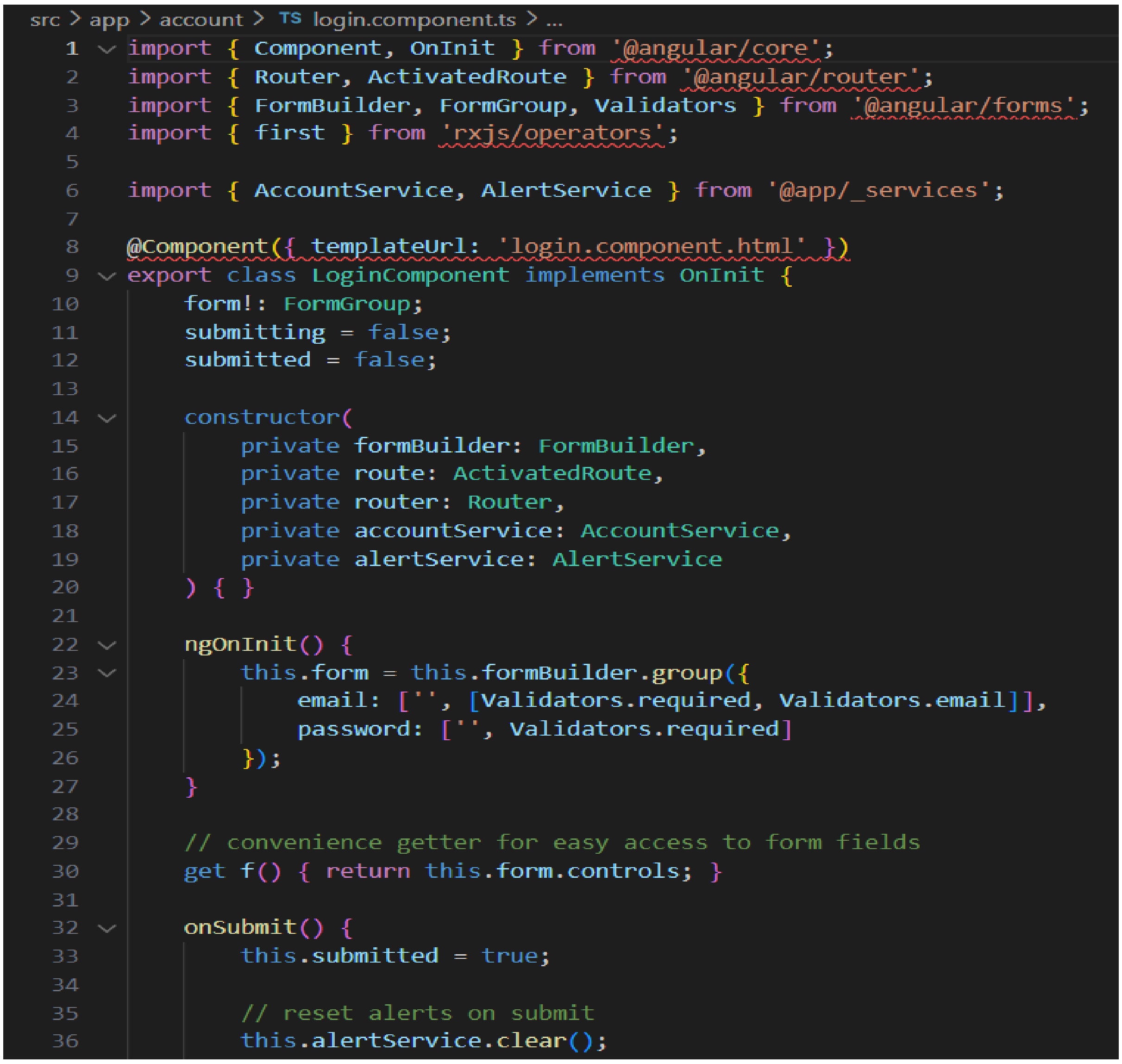
src > app > account > TS layout.component.ts > ...

```typescript
import { Component } from '@angular/core';
import { Router } from '@angular/router';

import { AccountService } from '@app/_services';

@Component({ templateUrl: 'layout.component.html' })
export class LayoutComponent {
    constructor(
        private router: Router,
        private accountService: AccountService
    ) {
        // redirect to home if already logged in
        if (this.accountService.accountValue) {
            this.router.navigate(['/']);
        }
    }
}
```



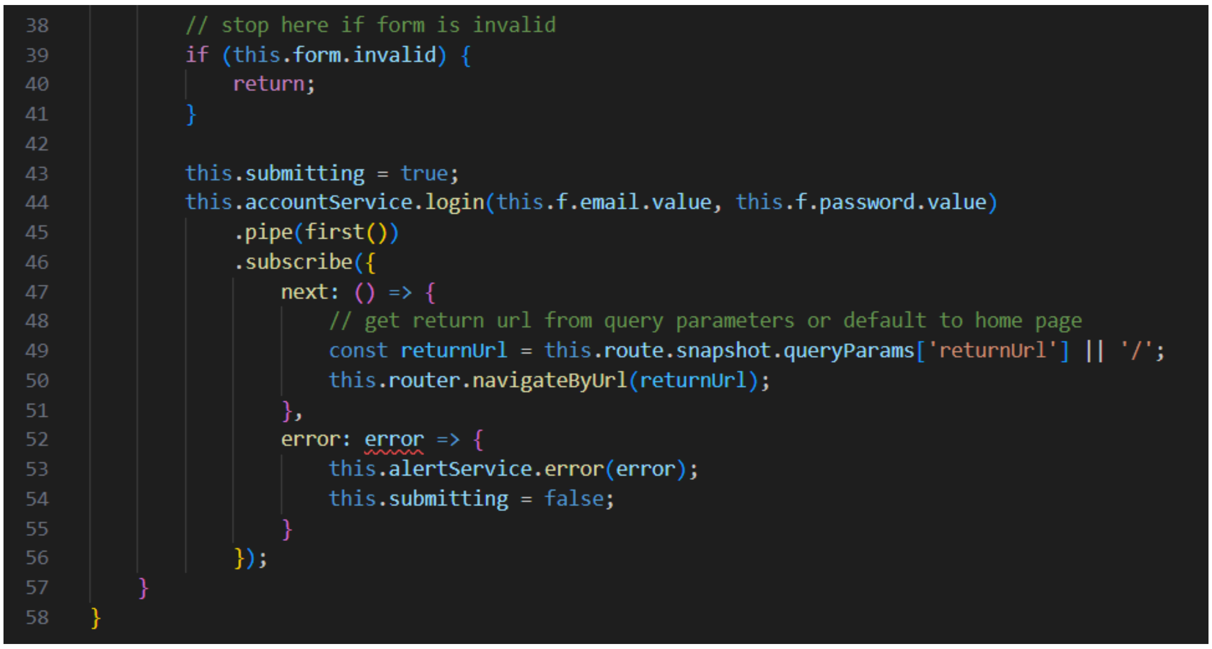
src > app > account > login.component.html > h3.card-header

```html
<h3 class="card-header">Login</h3>
<div class="card-body">
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
        <div class="mb-3">
            <label class="form-label">Email</label>
            <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }" />
            <div *ngIf="submitted && f.email.errors" class="invalid-feedback">
                <div *ngIf="f.email.errors.required">Email is required</div>
                <div *ngIf="f.email.errors.email">Email is invalid</div>
            </div>
        </div>
        <div class="mb-3">
            <label class="form-label">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="row">
            <div class="mb-3 col">
                <button [disabled]="submitting" class="btn btn-primary">
                    <span *ngIf="submitting" class="spinner-border spinner-border-sm me-1"></span>
                    Login
                </button>
                <a routerLink="../register" class="btn btn-link">Register</a>
            </div>
            <div class="mb-3 col text-end">
                <a routerLink="../forgot-password" class="btn btn-link pe-0">Forgot Password?</a>
            </div>
        </div>
    </form>
</div>
```

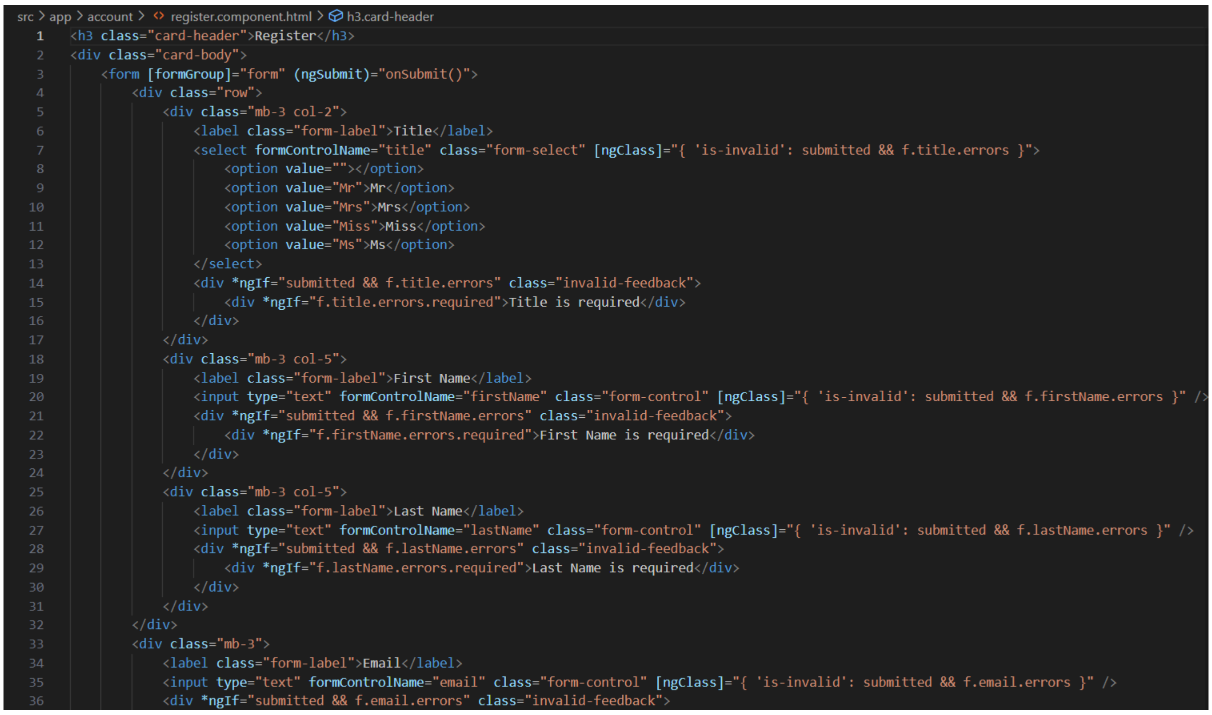


src > app > account > TS login.component.ts > ...

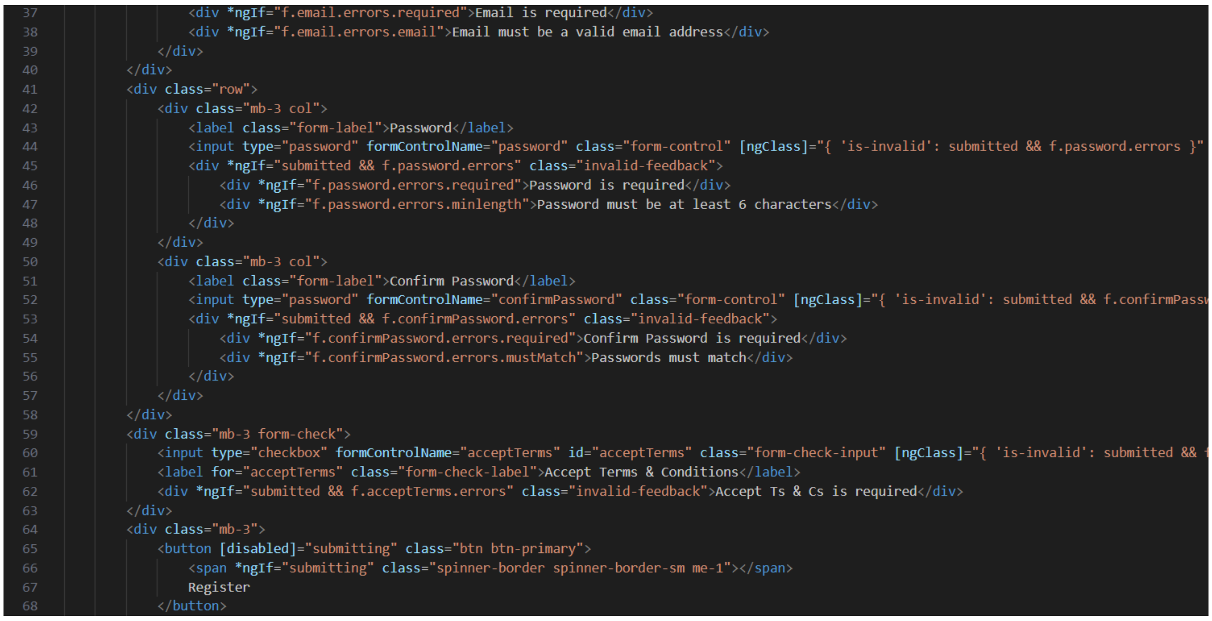
```typescript
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';

@Component({ templateUrl: 'login.component.html' })
export class LoginComponent implements OnInit {
    form!: FormGroup;
    submitting = false;
    submitted = false;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

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

    // convenience getter for easy access to form fields
    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        // reset alerts on submit
        this.alertService.clear();
```



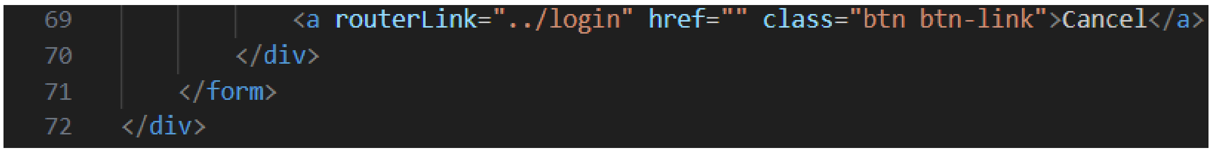
```
        // stop here if form is invalid
        if (this.form.invalid) {
            return;
        }

        this.submitting = true;
        this.accountService.login(this.f.email.value, this.f.password.value)
            .pipe(first())
            .subscribe({
                next: () => {
                    // get return url from query parameters or default to home page
                    const returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
                    this.router.navigateByUrl(returnUrl);
                },
                error: error => {
                    this.alertService.error(error);
                    this.submitting = false;
                }
            });
    }
}
```


src > app > account > register.component.html > h3.card-header

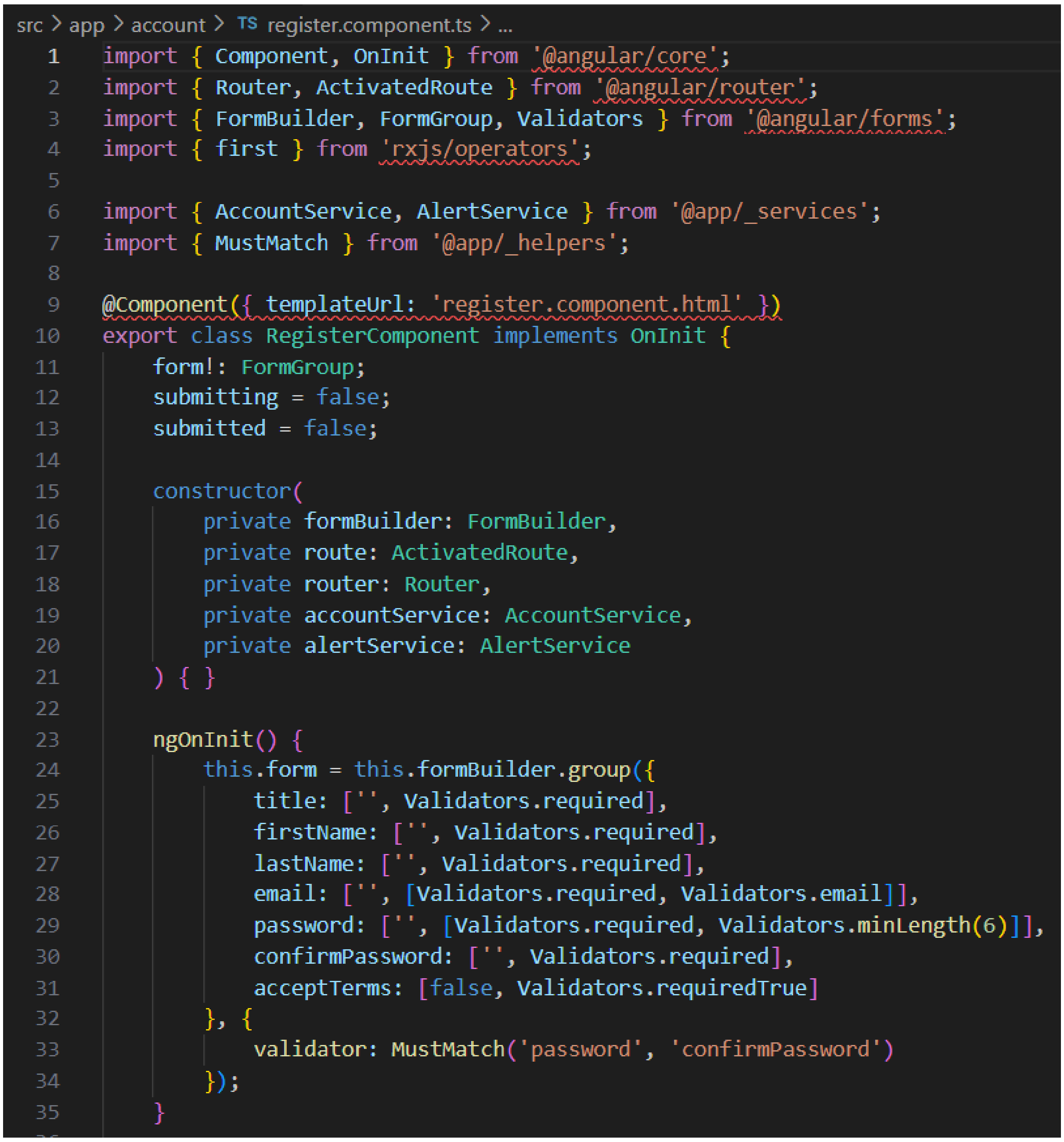
```
<h3 class="card-header">Register</h3>
<div class="card-body">
    <form [formGroup]="form" (ngSubmit)="onSubmit()">
        <div class="row">
            <div class="mb-3 col-2">
                <label class="form-label">Title</label>
                <select formControlName="title" class="form-select" [ngClass]="{ 'is-invalid': submitted && f.title.errors }">
                    <option value=""></option>
                    <option value="Mr">Mr</option>
                    <option value="Mrs">Mrs</option>
                    <option value="Miss">Miss</option>
                    <option value="Ms">Ms</option>
                </select>
                <div *ngIf="submitted && f.title.errors" class="invalid-feedback">
                    <div *ngIf="f.title.errors.required">Title is required</div>
                </div>
            </div>
            <div class="mb-3 col-5">
                <label class="form-label">First Name</label>
                <input type="text" formControlName="firstName" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.firstName.errors }" />
                <div *ngIf="submitted && f.firstName.errors" class="invalid-feedback">
                    <div *ngIf="f.firstName.errors.required">First Name is required</div>
                </div>
            </div>
            <div class="mb-3 col-5">
                <label class="form-label">Last Name</label>
                <input type="text" formControlName="lastName" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.lastName.errors }" />
                <div *ngIf="submitted && f.lastName.errors" class="invalid-feedback">
                    <div *ngIf="f.lastName.errors.required">Last Name is required</div>
                </div>
            </div>
        </div>
        <div class="mb-3">
            <label class="form-label">Email</label>
            <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }" />
            <div *ngIf="submitted && f.email.errors" class="invalid-feedback">
```



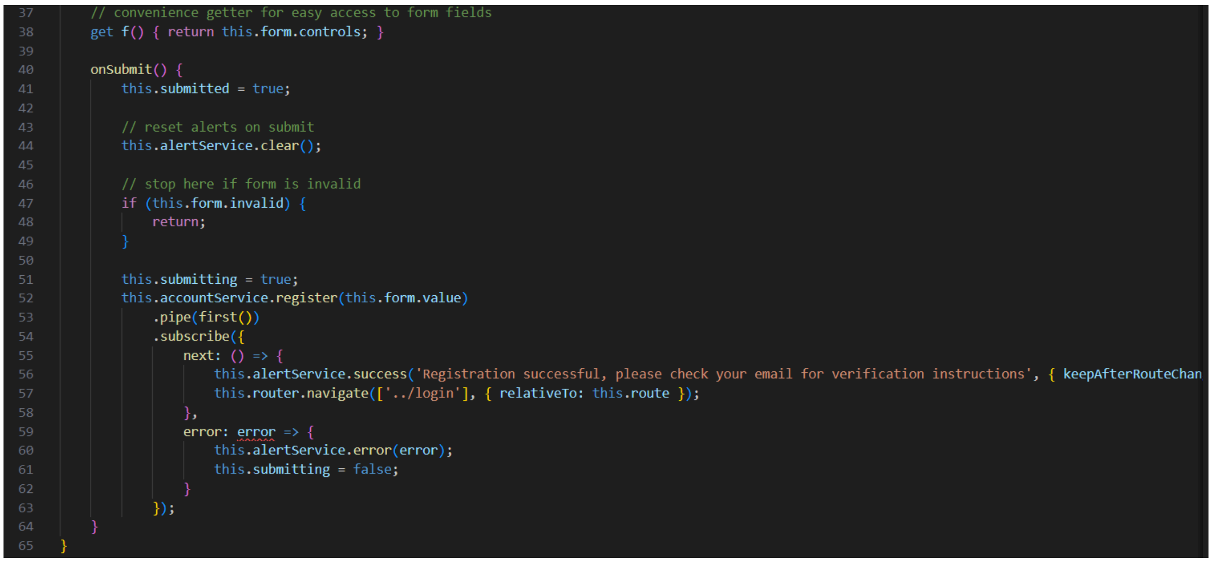
```html
                <div *ngIf="f.email.errors.required">Email is required</div>
                <div *ngIf="f.email.errors.email">Email must be a valid email address</div>
            </div>
        </div>
        <div class="row">
            <div class="mb-3 col">
                <label class="form-label">Password</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 *ngIf="f.password.errors.minlength">Password must be at least 6 characters</div>
                </div>
            </div>
            <div class="mb-3 col">
                <label class="form-label">Confirm Password</label>
                <input type="password" formControlName="confirmPassword" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.confirmPassw
                <div *ngIf="submitted && f.confirmPassword.errors" class="invalid-feedback">
                    <div *ngIf="f.confirmPassword.errors.required">Confirm Password is required</div>
                    <div *ngIf="f.confirmPassword.errors.mustMatch">Passwords must match</div>
                </div>
            </div>
        </div>
        <div class="mb-3 form-check">
            <input type="checkbox" formControlName="acceptTerms" id="acceptTerms" class="form-check-input" [ngClass]="{ 'is-invalid': submitted && f
            <label for="acceptTerms" class="form-check-label">Accept Terms & Conditions</label>
            <div *ngIf="submitted && f.acceptTerms.errors" class="invalid-feedback">Accept Ts & Cs is required</div>
        </div>
        <div class="mb-3">
            <button [disabled]="submitting" class="btn btn-primary">
                <span *ngIf="submitting" class="spinner-border spinner-border-sm me-1"></span>
                Register
            </button>
```

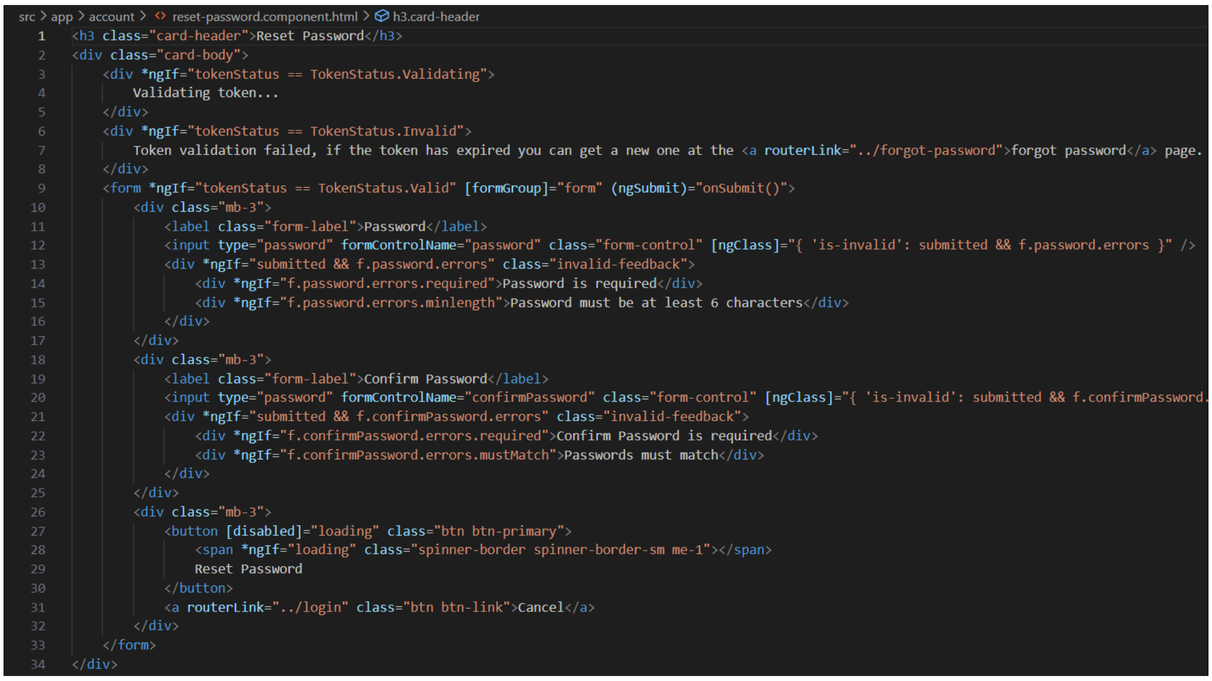


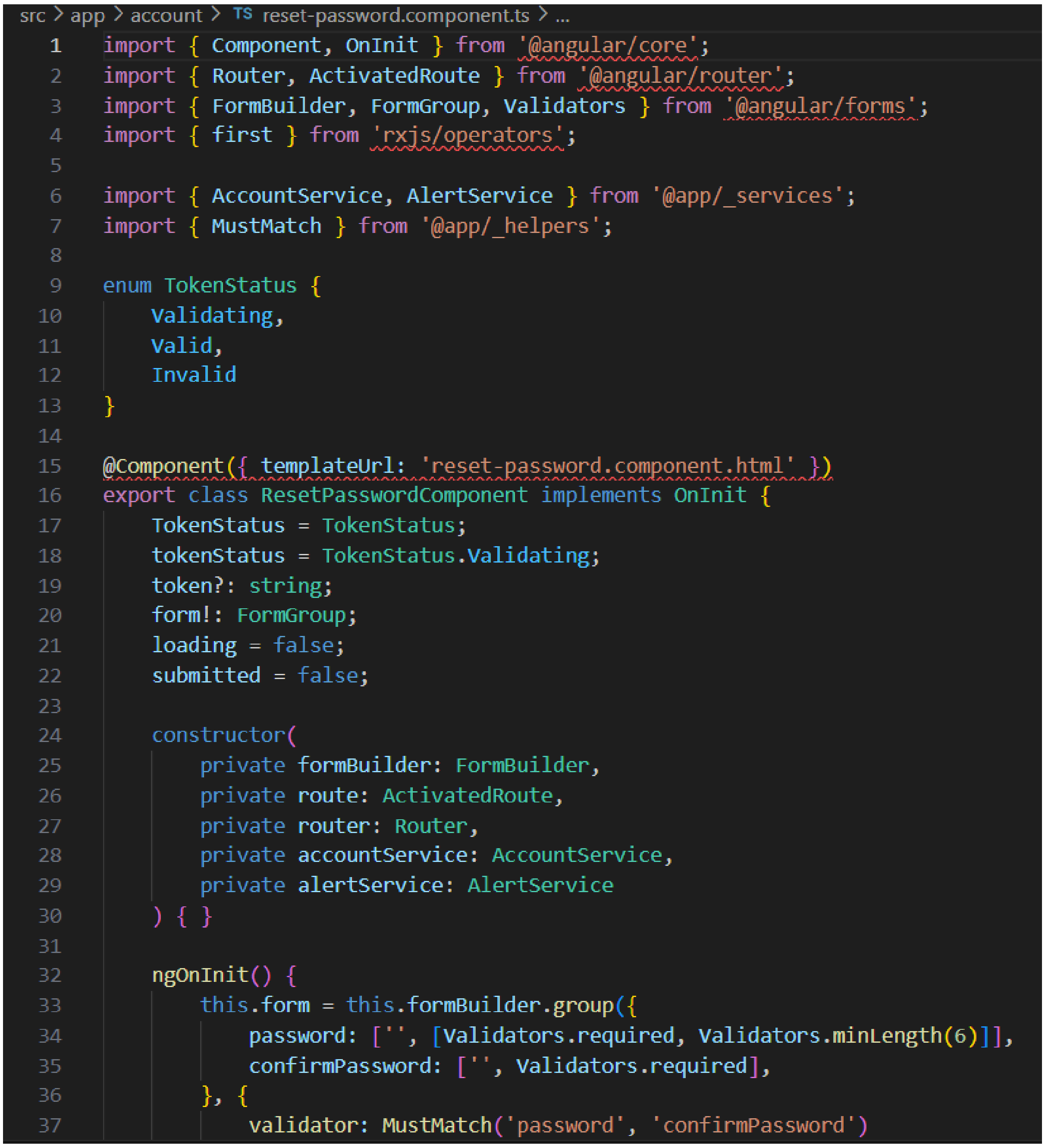
```html
            <a routerLink="../login" href="" class="btn btn-link">Cancel</a>
        </div>
    </form>
</div>
```


src > app > account > TS register.component.ts > ...

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

import { AccountService, AlertService } from '@app/_services';
import { MustMatch } from '@app/_helpers';

@Component({ templateUrl: 'register.component.html' })
export class RegisterComponent implements OnInit {
    form!: FormGroup;
    submitting = false;
    submitted = false;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            title: ['', Validators.required],
            firstName: ['', Validators.required],
            lastName: ['', Validators.required],
            email: ['', [Validators.required, Validators.email]],
            password: ['', [Validators.required, Validators.minLength(6)]],
            confirmPassword: ['', Validators.required],
            acceptTerms: [false, Validators.requiredTrue]
        }, {
            validator: MustMatch('password', 'confirmPassword')
        });
    }
```

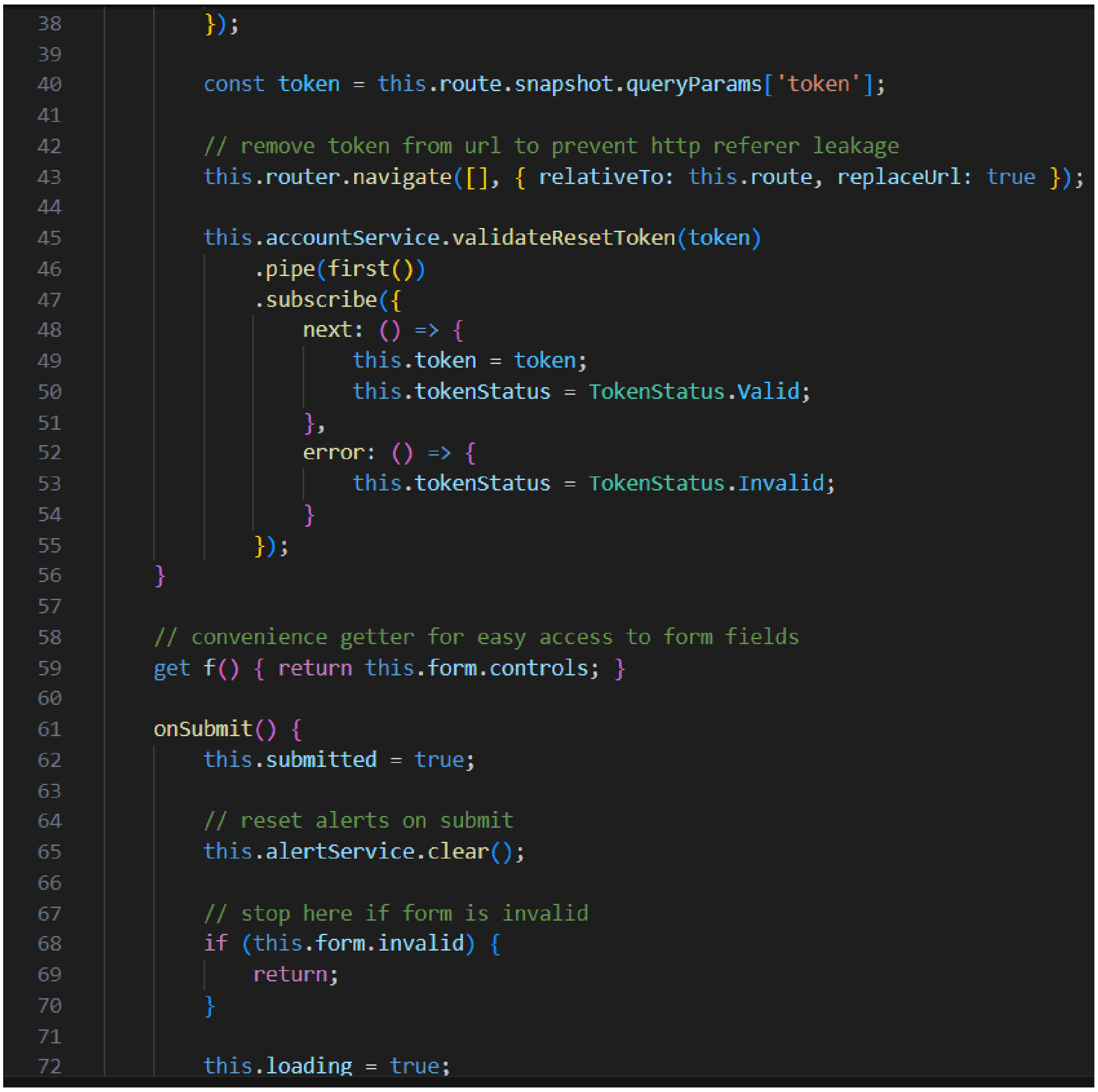

```
    // convenience getter for easy access to form fields
    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        // reset alerts on submit
        this.alertService.clear();

        // stop here if form is invalid
        if (this.form.invalid) {
            return;
        }

        this.submitting = true;
        this.accountService.register(this.form.value)
            .pipe(first())
            .subscribe({
                next: () => {
                    this.alertService.success('Registration successful, please check your email for verification instructions', { keepAfterRouteChan
                    this.router.navigate(['../login'], { relativeTo: this.route });
                },
                error: error => {
                    this.alertService.error(error);
                    this.submitting = false;
                }
            });
    }
}
```


src > app > account > reset-password.component.html > h3.card-header
```
<h3 class="card-header">Reset Password</h3>
<div class="card-body">
    <div *ngIf="tokenStatus == TokenStatus.Validating">
        Validating token...
    </div>
    <div *ngIf="tokenStatus == TokenStatus.Invalid">
        Token validation failed, if the token has expired you can get a new one at the <a routerLink="../forgot-password">forgot password</a> page.
    </div>
    <form *ngIf="tokenStatus == TokenStatus.Valid" [formGroup]="form" (ngSubmit)="onSubmit()">
        <div class="mb-3">
            <label class="form-label">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 *ngIf="f.password.errors.minlength">Password must be at least 6 characters</div>
            </div>
        </div>
        <div class="mb-3">
            <label class="form-label">Confirm Password</label>
            <input type="password" formControlName="confirmPassword" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.confirmPassword.
            <div *ngIf="submitted && f.confirmPassword.errors" class="invalid-feedback">
                <div *ngIf="f.confirmPassword.errors.required">Confirm Password is required</div>
                <div *ngIf="f.confirmPassword.errors.mustMatch">Passwords must match</div>
            </div>
        </div>
        <div class="mb-3">
            <button [disabled]="loading" class="btn btn-primary">
                <span *ngIf="loading" class="spinner-border spinner-border-sm me-1"></span>
                Reset Password
            </button>
            <a routerLink="../login" class="btn btn-link">Cancel</a>
        </div>
    </form>
</div>
```

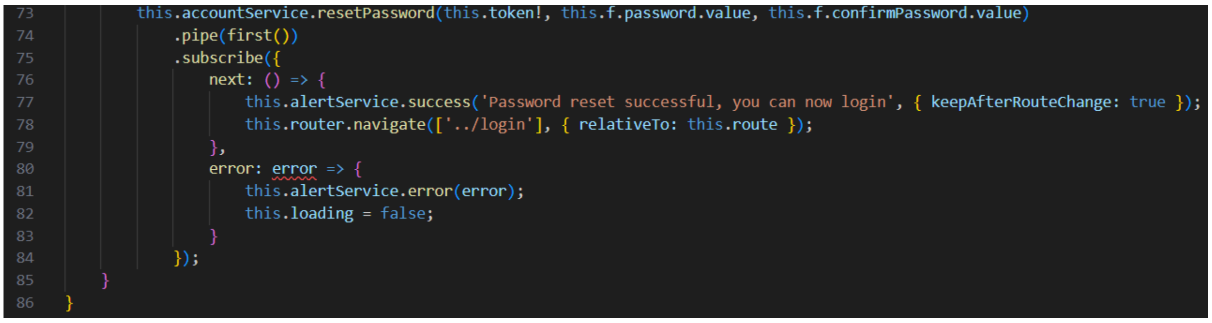


src > app > account > TS reset-password.component.ts > ...

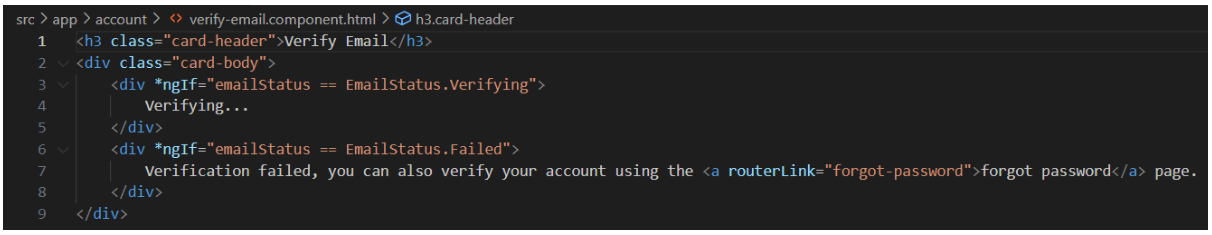
```typescript
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';
import { MustMatch } from '@app/_helpers';

enum TokenStatus {
    Validating,
    Valid,
    Invalid
}

@Component({ templateUrl: 'reset-password.component.html' })
export class ResetPasswordComponent implements OnInit {
    TokenStatus = TokenStatus;
    tokenStatus = TokenStatus.Validating;
    token?: string;
    form!: FormGroup;
    loading = false;
    submitted = false;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            password: ['', [Validators.required, Validators.minLength(6)]],
            confirmPassword: ['', Validators.required],
        }, {
            validator: MustMatch('password', 'confirmPassword')
```



```
        });

        const token = this.route.snapshot.queryParams['token'];

        // remove token from url to prevent http referer leakage
        this.router.navigate([], { relativeTo: this.route, replaceUrl: true });

        this.accountService.validateResetToken(token)
            .pipe(first())
            .subscribe({
                next: () => {
                    this.token = token;
                    this.tokenStatus = TokenStatus.Valid;
                },
                error: () => {
                    this.tokenStatus = TokenStatus.Invalid;
                }
            });
    }

    // convenience getter for easy access to form fields
    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        // reset alerts on submit
        this.alertService.clear();

        // stop here if form is invalid
        if (this.form.invalid) {
            return;
        }

        this.loading = true;
```

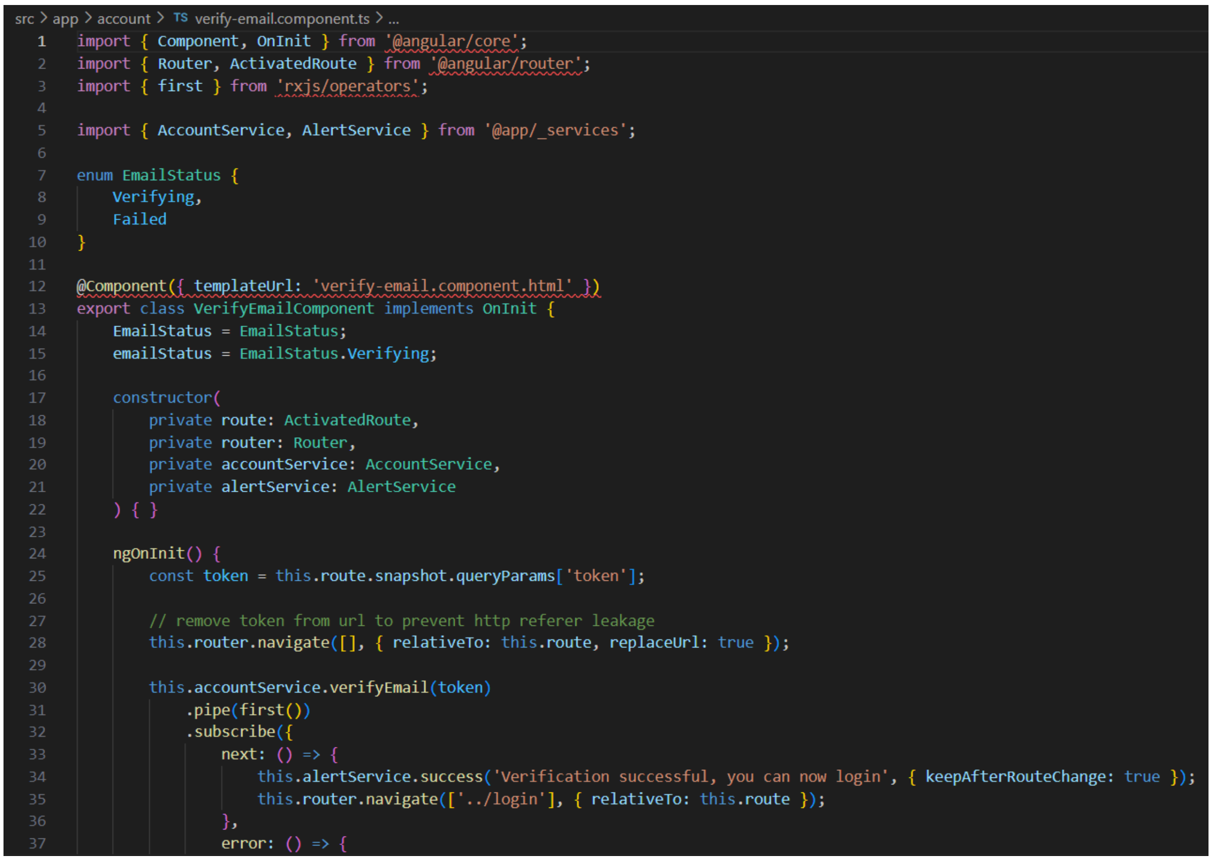


```
        this.accountService.resetPassword(this.token!, this.f.password.value, this.f.confirmPassword.value)
            .pipe(first())
            .subscribe({
                next: () => {
                    this.alertService.success('Password reset successful, you can now login', { keepAfterRouteChange: true });
                    this.router.navigate(['../login'], { relativeTo: this.route });
                },
                error: error => {
                    this.alertService.error(error);
                    this.loading = false;
                }
            });
    }
}
```

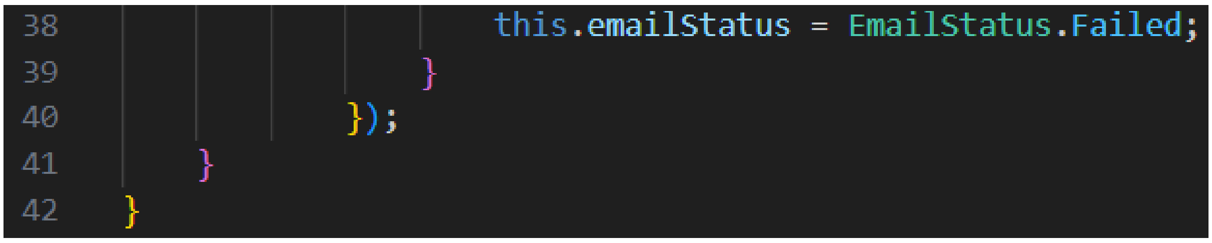


src > app > account > verify-email.component.html > h3.card-header

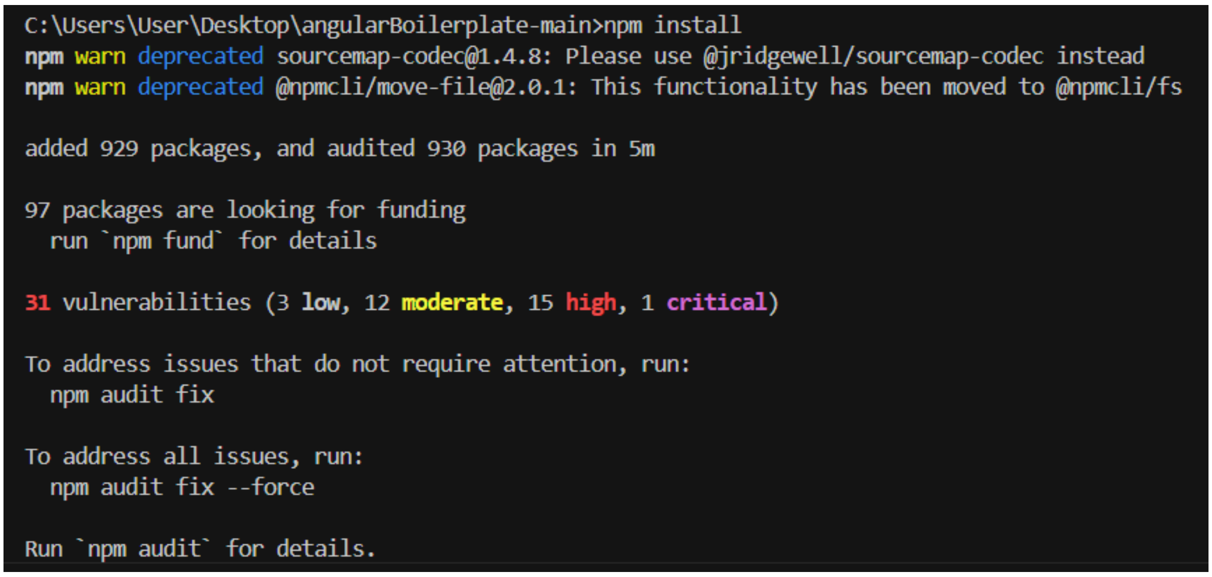
```
<h3 class="card-header">Verify Email</h3>
<div class="card-body">
    <div *ngIf="emailStatus == EmailStatus.Verifying">
        Verifying...
    </div>
    <div *ngIf="emailStatus == EmailStatus.Failed">
        Verification failed, you can also verify your account using the <a routerLink="forgot-password">forgot password</a> page.
    </div>
</div>
```



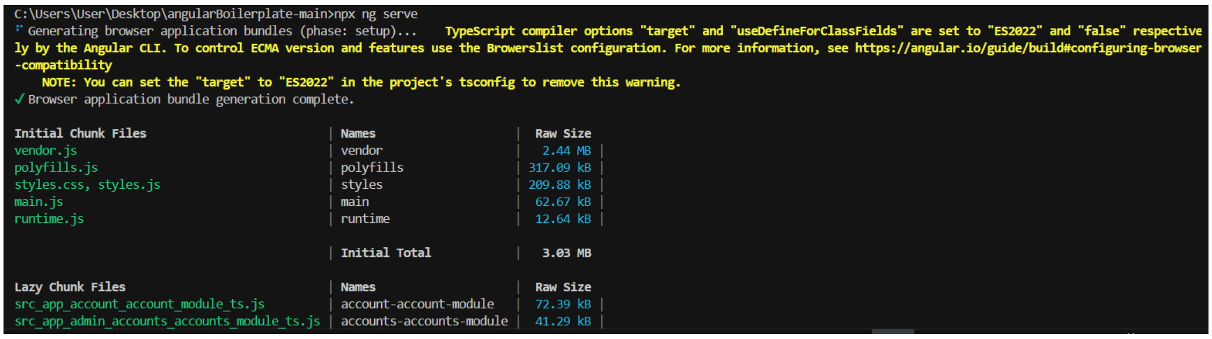
```typescript
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';

enum EmailStatus {
    Verifying,
    Failed
}

@Component({ templateUrl: 'verify-email.component.html' })
export class VerifyEmailComponent implements OnInit {
    EmailStatus = EmailStatus;
    emailStatus = EmailStatus.Verifying;

    constructor(
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        const token = this.route.snapshot.queryParams['token'];

        // remove token from url to prevent http referer leakage
        this.router.navigate([], { relativeTo: this.route, replaceUrl: true });

        this.accountService.verifyEmail(token)
            .pipe(first())
            .subscribe({
                next: () => {
                    this.alertService.success('Verification successful, you can now login', { keepAfterRouteChange: true });
                    this.router.navigate(['../login'], { relativeTo: this.route });
                },
                error: () => {
```



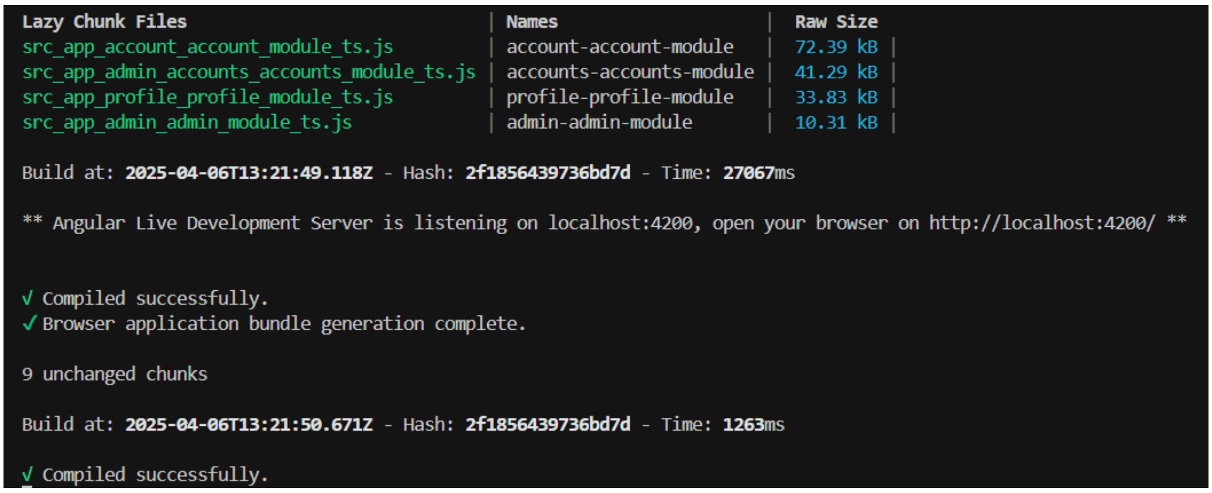
```typescript
                    this.emailStatus = EmailStatus.Failed;
                }
            });
    }
}
```



```
C:\Users\User\Desktop\angularBoilerplate-main>npm install
npm warn deprecated sourcemap-codec@1.4.8: Please use @jridgewell/sourcemap-codec instead
npm warn deprecated @npmcli/move-file@2.0.1: This functionality has been moved to @npmcli/fs

added 929 packages, and audited 930 packages in 5m

97 packages are looking for funding
  run `npm fund` for details

31 vulnerabilities (3 low, 12 moderate, 15 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues, run:
  npm audit fix --force

Run `npm audit` for details.
```



```
C:\Users\User\Desktop\angularBoilerplate-main>npx ng serve
⠋ Generating browser application bundles (phase: setup)...    TypeScript compiler options "target" and "useDefineForClassFields" are set to "ES2022" and "false" respective
ly by the Angular CLI. To control ECMA version and features use the Browerslist configuration. For more information, see https://angular.io/guide/build#configuring-browser
-compatibility
    NOTE: You can set the "target" to "ES2022" in the project's tsconfig to remove this warning.
✓ Browser application bundle generation complete.

Initial Chunk Files                              | Names                     |  Raw Size
vendor.js                                        | vendor                    |   2.44 MB |
polyfills.js                                     | polyfills                 | 317.09 kB |
styles.css, styles.js                            | styles                    | 209.88 kB |
main.js                                          | main                      |  62.67 kB |
runtime.js                                       | runtime                   |  12.64 kB |

                                                 | Initial Total             |   3.03 MB

Lazy Chunk Files                                 | Names                     |  Raw Size
src_app_account_account_module_ts.js             | account-account-module    |  72.39 kB |
src_app_admin_accounts_accounts_module_ts.js     | accounts-accounts-module  |  41.29 kB |
```



```
Lazy Chunk Files                                 | Names                     |  Raw Size
src_app_account_account_module_ts.js             | account-account-module    |  72.39 kB |
src_app_admin_accounts_accounts_module_ts.js     | accounts-accounts-module  |  41.29 kB |
src_app_profile_profile_module_ts.js             | profile-profile-module    |  33.83 kB |
src_app_admin_admin_module_ts.js                 | admin-admin-module        |  10.31 kB |

Build at: 2025-04-06T13:21:49.118Z - Hash: 2f1856439736bd7d - Time: 27067ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
✓ Browser application bundle generation complete.

9 unchanged chunks

Build at: 2025-04-06T13:21:50.671Z - Hash: 2f1856439736bd7d - Time: 1263ms

√ Compiled successfully.
```