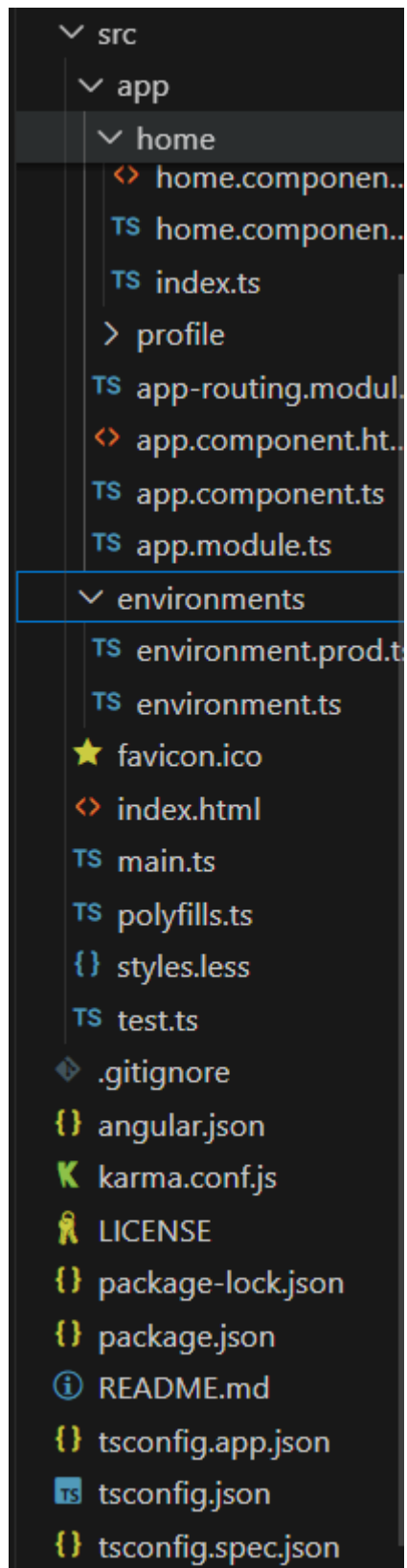


- ▼ ANGULARBOILERPLATE-M...
- ▼ src
  - ▼ app
    - ▼ \_components
      - <> alert.component....
      - TS alert.component.ts
      - TS index.ts
    - ▼ \_helpers
      - TS app.initializer.ts
      - TS auth.guard.ts
      - TS error.interceptor.ts
      - TS fake-backend.ts
      - TS index.ts
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

- ▼ 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...
- ▼ 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 > \_component > alert.component.html

```
src > app > _components > alert.component.html > div.container
1 <div *ngIf="alerts.length" class="container">
2   <div class="m-3">
3     <div *ngFor="let alert of alerts" class="{{cssClasses(alert)}}">
4       <span [innerHTML]="alert.message"></span>
5       <button class="btn-close" (click)="removeAlert(alert)"></button>
6     </div>
7   </div>
8 </div>
```

src > app > \_component > alert.component.ts

```
src > app > _components > TS alert.component.ts > AlertComponent > alertSubscription
1 import { Component, OnInit, OnDestroy, Input } from '@angular/core';
2 import { Router, NavigationStart } from '@angular/router';
3 import { Subscription } from 'rxjs';
4
5 import { Alert, AlertType } from '@app/_models';
6 import { AlertService } from '@app/_services';
7
8 @Component({ selector: 'alert', templateUrl: 'alert.component.html' })
9 export class AlertComponent implements OnInit, OnDestroy {
10   @Input() id = 'default-alert';
11   @Input() fade = true;
12
13   alerts: Alert[] = [];
14   alertSubscription!: Subscription;
15   routeSubscription!: Subscription;
16
17   constructor(private router: Router, private alertService: AlertService) { }
18
19   ngOnInit() {
20     // subscribe to new alert notifications
21     this.alertSubscription = this.alertService.onAlert(this.id)
22       .subscribe(alert => {
23         // clear alerts when an empty alert is received
24         if (!alert.message) {
25           // filter out alerts without 'keepAfterRouteChange' flag
26           this.alerts = this.alerts.filter(x => x.keepAfterRouteChange);
27
28           // remove 'keepAfterRouteChange' flag on the rest
29           this.alerts.forEach(x => delete x.keepAfterRouteChange);
30           return;
31         }
32
33         // add alert to array
34         this.alerts.push(alert);
35
36         // auto close alert if required
37         if (alert.autoClose) {
```

```

38         setTimeout(() => this.removeAlert(alert), 3000);
39     }
40 });
41
42 // clear alerts on location change
43 this.routeSubscription = this.router.events.subscribe(event => {
44     if (event instanceof NavigationStart) {
45         this.alertService.clear(this.id);
46     }
47 });
48 }
49
50 ngOnDestroy() {
51     // unsubscribe to avoid memory leaks
52     this.alertSubscription.unsubscribe();
53     this.routeSubscription.unsubscribe();
54 }
55
56 removeAlert(alert: Alert) {
57     // check if already removed to prevent error on auto close
58     if (!this.alerts.includes(alert)) return;
59
60     if (this.fade) {
61         // fade out alert
62         alert.fade = true;
63
64         // remove alert after faded out
65         setTimeout(() => {
66             this.alerts = this.alerts.filter(x => x !== alert);
67         }, 250);
68     } else {
69         // remove alert
70         this.alerts = this.alerts.filter(x => x !== alert);

```

```

71     }
72 }
73
74 cssClasses(alert: Alert) {
75     if (!alert) return;
76
77     const classes = ['alert', 'alert-dismissible', 'mt-4', 'container'];
78
79     const alertTypeClass = {
80         [AlertType.Success]: 'alert-success',
81         [AlertType.Error]: 'alert-danger',
82         [AlertType.Info]: 'alert-info',
83         [AlertType.Warning]: 'alert-warning'
84     };
85
86     if (alert.type !== undefined) {
87         classes.push(alertTypeClass[alert.type]);
88     }
89
90     if (alert.fade) {
91         classes.push('fade');
92     }
93
94     return classes.join(' ');
95 }
96 }

```

src > app > \_components > index.ts

```
src > app > _components > TS index.ts
1   export * from './alert.component';
```

src > app > \_helpers > TS app.initializer.ts > ...

```
1   import { catchError, of } from 'rxjs';
2
3   import { AccountService } from '@app/_services';
4
5   export function appInitializer(accountService: AccountService) {
6       return () => accountService.refreshToken()
7           .pipe(
8               // catch error to start app on success or failure
9               catchError(() => of())
10          );
11  }
```

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

```
1   import { Injectable } from '@angular/core';
2   import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
3
4   import { AccountService } from '@app/_services';
5
6   @Injectable({ providedIn: 'root' })
7   export class AuthGuard implements CanActivate {
8       constructor(
9           private router: Router,
10          private accountService: AccountService
11      ) { }
12
13      canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
14          const account = this.accountService.accountValue;
15          if (account) {
16              // check if route is restricted by role
17              if (route.data.roles && !route.data.roles.includes(account.role)) {
18                  // role not authorized so redirect to home page
19                  this.router.navigate(['/']);
20                  return false;
21              }
22
23              // authorized so return true
24              return true;
25          }
26
27          // not logged in so redirect to login page with the return url
28          this.router.navigate(['/account/login'], { queryParams: { returnUrl: state.url } });
29          return false;
30      }
31  }
```

```

src > app > _helpers > TS error.interceptor.ts > ...
1  import { Injectable } from '@angular/core';
2  import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
3  import { Observable, throwError } from 'rxjs';
4  import { catchError } from 'rxjs/operators';
5
6  import { AccountService } from '@app/_services';
7
8  @Injectable()
9  export class ErrorInterceptor implements HttpInterceptor {
10     constructor(private accountService: AccountService) { }
11
12     intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
13         return next.handle(request).pipe(catchError(err => {
14             if ([401, 403].includes(err.status) && this.accountService.accountValue) {
15                 // auto logout if 401 or 403 response returned from api
16                 this.accountService.logout();
17             }
18
19             const error = (err && err.error && err.error.message) || err.statusText;
20             console.error(err);
21             return throwError(() => error);
22         })))
23     }
24 }

```

```

src > app > _helpers > TS fake-backend.ts > FakeBackendInterceptor
1  import { Injectable } from '@angular/core';
2  import { HttpRequest, HttpResponse, HttpHandler, HttpEvent, HttpInterceptor, HTTP_INTERCEPTORS } from '@angular/common/http';
3  import { Observable, of, throwError } from 'rxjs';
4  import { delay, materialize, dematerialize } from 'rxjs/operators';
5
6  import { AlertService } from '@app/_services';
7  import { Role } from '@app/_models';
8
9
10
11 // array in local storage for accounts
12 const accountsKey = 'angular-15-signup-verification-boilerplate-accounts';
13 let accounts: any[] = JSON.parse(localStorage.getItem(accountsKey)!) || [];
14
15
16 @Injectable()
17 export class FakeBackendInterceptor implements HttpInterceptor {
18     constructor(private alertService: AlertService) { }
19
20     intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
21         const { url, method, headers, body } = request;
22         const alertService = this.alertService;
23
24         return handleRoute();
25
26         function handleRoute() {
27             switch (true) {
28                 case url.endsWith('/accounts/authenticate') && method === 'POST':
29                     return authenticate();
30                 case url.endsWith('/accounts/refresh-token') && method === 'POST':
31                     return refreshToken();
32                 case url.endsWith('/accounts/revoke-token') && method === 'POST':
33                     return revokeToken();
34                 case url.endsWith('/accounts/register') && method === 'POST':
35                     return register();
36                 case url.endsWith('/accounts/verify-email') && method === 'POST':
37                     return verifyEmail();

```

```

38     case url.endsWith('/accounts/forgot-password') && method === 'POST':
39         return forgotPassword();
40     case url.endsWith('/accounts/validate-reset-token') && method === 'POST':
41         return validateResetToken();
42     case url.endsWith('/accounts/reset-password') && method === 'POST':
43         return resetPassword();
44     case url.endsWith('/accounts') && method === 'GET':
45         return getAccounts();
46     case url.match(/\saccounts\s\d+$/) && method === 'GET':
47         return getAccountById();
48     case url.endsWith('/accounts') && method === 'POST':
49         return createAccount();
50     case url.match(/\saccounts\s\d+$/) && method === 'PUT':
51         return updateAccount();
52     case url.match(/\saccounts\s\d+$/) && method === 'DELETE':
53         return deleteAccount();
54     default:
55         // pass through any requests not handled above
56         return next.handle(request);
57     }
58 }
59
60 // route functions
61
62 function authenticate() {
63     const { email, password } = body;
64     const account = accounts.find(x => x.email === email && x.password === password && x.isVerified);
65
66     if (!account) return error('Email or password is incorrect');
67
68     // add refresh token to account
69     account.refreshTokens.push(generateRefreshToken());
70     localStorage.setItem(accountsKey, JSON.stringify(accounts));

```

```

71
72     return ok({
73         ...basicDetails(account),
74         jwtToken: generateJwtToken(account)
75     });
76 }
77
78 function refreshToken() {
79     const refreshToken = getRefreshToken();
80
81     if (!refreshToken) return unauthorized();
82
83     const account = accounts.find(x => x.refreshTokens.includes(refreshToken));
84
85     if (!account) return unauthorized();
86
87     // replace old refresh token with a new one and save
88     account.refreshTokens = account.refreshTokens.filter((x: any) => x !== refreshToken);
89     account.refreshTokens.push(generateRefreshToken());
90     localStorage.setItem(accountsKey, JSON.stringify(accounts));
91
92     return ok({
93         ...basicDetails(account),
94         jwtToken: generateJwtToken(account)
95     });
96 }
97
98 function revokeToken() {
99     if (!isAuthenticated()) return unauthorized();
100
101     const refreshToken = getRefreshToken();
102     const account = accounts.find(x => x.refreshTokens.includes(refreshToken));

```

```

104 // revoke token and save
105 account.refreshTokens = account.refreshTokens.filter((x: any) => x !== refreshToken);
106 localStorage.setItem(accountsKey, JSON.stringify(accounts));
107
108 return ok();
109 }
110
111 function register() {
112   const account = body;
113
114   if (accounts.find(x => x.email === account.email)) {
115     // display email already registered "email" in alert
116     setTimeout(() => {
117       alertService.info(`
118         <h4>Email Already Registered</h4>
119         <p>Your email ${account.email} is already registered.</p>
120         <p>If you don't know your password please visit the <a href="${location.origin}/account/forgot-password">forgot password</a>
121         <div><strong>NOTE:</strong> The fake backend displayed this "email" so you can test without an api. A real backend would send a
122         `, { autoClose: false });
123     }, 1000);
124
125     // always return ok() response to prevent email enumeration
126     return ok();
127   }
128
129   // assign account id and a few other properties then save
130   account.id = newAccountId();
131   if (account.id === 1) {
132     // first registered account is an admin
133     account.role = Role.Admin;
134   } else {
135     account.role = Role.User;
136   }
137
138   account.dateCreated = new Date().toISOString();
139   account.verificationToken = new Date().getTime().toString();
140   account.isVerified = false;
141   account.refreshTokens = [];
142   delete account.confirmPassword;
143   accounts.push(account);
144   localStorage.setItem(accountsKey, JSON.stringify(accounts));
145
146   // display verification email in alert
147   setTimeout(() => {
148     const verifyUrl = `${location.origin}/account/verify-email?token=${account.verificationToken}`;
149     alertService.info(`
150       <h4>Verification Email</h4>
151       <p>Thanks for registering!</p>
152       <p>Please click the below link to verify your email address:</p>
153       <p><a href="${verifyUrl}">${verifyUrl}</a></p>
154       <div><strong>NOTE:</strong> The fake backend displayed this "email" so you can test without an api. A real backend would send a
155       `, { autoClose: false });
156     }, 1000);
157
158   return ok();
159 }
160
161 function verifyEmail() {
162   const { token } = body;
163   const account = accounts.find(x => !!x.verificationToken && x.verificationToken === token);
164
165   if (!account) return error('Verification failed');
166
167   // set is verified flag to true if token is valid
168   account.isVerified = true;
169   localStorage.setItem(accountsKey, JSON.stringify(accounts));

```



```

170         return ok();
171     }
172
173     function forgotPassword() {
174         const { email } = body;
175         const account = accounts.find(x => x.email === email);
176
177         // always return ok() response to prevent email enumeration
178         if (!account) return ok();
179
180         // create reset token that expires after 24 hours
181         account.resetToken = new Date().getTime().toString();
182         account.resetTokenExpires = new Date(Date.now() + 24*60*60*1000).toISOString();
183         localStorage.setItem(accountsKey, JSON.stringify(accounts));
184
185         // display password reset email in alert
186         setTimeout(() => {
187             const resetUrl = `${location.origin}/account/reset-password?token=${account.resetToken}`;
188             alertService.info(
189                 <h4>Reset Password Email</h4>
190                 <p>Please click the below link to reset your password, the link will be valid for 1 day:</p>
191                 <p><a href="${resetUrl}">${resetUrl}</a></p>
192                 <div><strong>NOTE:</strong> The fake backend displayed this "email" so you can test without an api. A real backend would send a
193                 , { autoClose: false });
194         }, 1000);
195
196         return ok();
197     }
198
199     function validateResetToken() {
200         const { token } = body;
201         const account = accounts.find(x =>
202             !!x.resetToken && x.resetToken === token &&

```

```

202             !!x.resetToken && x.resetToken === token &&
203             new Date() < new Date(x.resetTokenExpires)
204         );
205
206         if (!account) return error('Invalid token');
207
208         return ok();
209     }
210
211     function resetPassword() {
212         const { token, password } = body;
213         const account = accounts.find(x =>
214             !!x.resetToken && x.resetToken === token &&
215             new Date() < new Date(x.resetTokenExpires)
216         );
217
218         if (!account) return error('Invalid token');
219
220         // update password and remove reset token
221         account.password = password;
222         account.isVerified = true;
223         delete account.resetToken;
224         delete account.resetTokenExpires;
225         localStorage.setItem(accountsKey, JSON.stringify(accounts));
226
227         return ok();
228     }
229
230     function getAccounts() {
231         if (!isAuthenticated()) return unauthorized();
232         return ok(accounts.map(x => basicDetails(x)));
233     }

```

```
235 function getAccountById() {
236   if (!isAuthenticated()) return unauthorized();
237
238   let account = accounts.find(x => x.id === idFromUrl());
239
240   // user accounts can get own profile and admin accounts can get all profiles
241   if (account.id !== currentAccount().id && !isAuthorized(Role.Admin)) {
242     return unauthorized();
243   }
244
245   return ok(basicDetails(account));
246 }
247
248 function createAccount() {
249   if (!isAuthorized(Role.Admin)) return unauthorized();
250
251   const account = body;
252   if (accounts.find(x => x.email === account.email)) {
253     return error(`Email ${account.email} is already registered`);
254   }
255
256   // assign account id and a few other properties then save
257   account.id = newAccountId();
258   account.dateCreated = new Date().toISOString();
259   account.isVerified = true;
260   account.refreshTokens = [];
261   delete account.confirmPassword;
262   accounts.push(account);
263   localStorage.setItem(accountsKey, JSON.stringify(accounts));
264
265   return ok();
266 }
```

```

268 function updateAccount() {
269     if (!isAuthenticated()) return unauthorized();
270
271     let params = body;
272     let account = accounts.find(x => x.id === idFromUrl());
273
274     // user accounts can update own profile and admin accounts can update all profiles
275     if (account.id !== currentAccount().id && !isAuthorized(Role.Admin)) {
276         return unauthorized();
277     }
278
279     // only update password if included
280     if (!params.password) {
281         delete params.password;
282     }
283     // don't save confirm password
284     delete params.confirmPassword;
285
286     // update and save account
287     Object.assign(account, params);
288     localStorage.setItem(accountsKey, JSON.stringify(accounts));
289
290     return ok(basicDetails(account));
291 }
292
293 function deleteAccount() {
294     if (!isAuthenticated()) return unauthorized();
295
296     let account = accounts.find(x => x.id === idFromUrl());
297
298     // user accounts can delete own account and admin accounts can delete any account
299     if (account.id !== currentAccount().id && !isAuthorized(Role.Admin)) {
300         return unauthorized();
301     }

```

```

303     // delete account then save
304     accounts = accounts.filter(x => x.id !== idFromUrl());
305     localStorage.setItem(accountsKey, JSON.stringify(accounts));
306     return ok();
307 }
308
309 // helper functions
310
311 function ok(body?: any) {
312     return of(new HttpResponse({ status: 200, body }))
313         .pipe(delay(500)); // delay observable to simulate server api call
314 }
315
316 function error(message: string) {
317     return throwError(() => ({ error: { message } }))
318         .pipe(materialize(), delay(500), dematerialize()); // call materialize and dematerialize to ensure delay even if an error is thrown
319 }
320
321 function unauthorized() {
322     return throwError(() => ({ status: 401, error: { message: 'Unauthorized' } }))
323         .pipe(materialize(), delay(500), dematerialize());
324 }
325
326 function basicDetails(account: any) {
327     const { id, title, firstName, lastName, email, role, dateCreated, isVerified } = account;
328     return { id, title, firstName, lastName, email, role, dateCreated, isVerified };
329 }
330
331 function isAuthenticated() {
332     return !!currentAccount();
333 }
334

```

```

335  function isAuthorized(role: any) {
336      const account = currentAccount();
337      if (!account) return false;
338      return account.role === role;
339  }
340
341  function idFromUrl() {
342      const urlParts = url.split('/');
343      return parseInt(urlParts[urlParts.length - 1]);
344  }
345
346  function newAccountId() {
347      return accounts.length ? Math.max(...accounts.map(x => x.id)) + 1 : 1;
348  }
349
350  function currentAccount() {
351      // check if jwt token is in auth header
352      const authHeader = headers.get('Authorization');
353      if (!authHeader?.startsWith('Bearer fake-jwt-token')) return;
354
355      // check if token is expired
356      const jwtToken = JSON.parse(atob(authHeader.split('.')[1]));
357      const tokenExpired = Date.now() > (jwtToken.exp * 1000);
358      if (tokenExpired) return;
359
360      const account = accounts.find(x => x.id === jwtToken.id);
361      return account;
362  }
363
364  function generateJwtToken(account: any) {
365      // create token that expires in 15 minutes
366      const tokenPayload = {
367          exp: Math.round(new Date(Date.now() + 15*60*1000).getTime() / 1000),
368          id: account.id

```

```

369      }
370      return `fake-jwt-token.${btoa(JSON.stringify(tokenPayload))}`;
371  }
372
373  function generateRefreshToken() {
374      const token = new Date().getTime().toString();
375
376      // add token cookie that expires in 7 days
377      const expires = new Date(Date.now() + 7*24*60*60*1000).toUTCString();
378      document.cookie = `fakeRefreshToken=${token}; expires=${expires}; path=/`;
379
380      return token;
381  }
382
383  function getRefreshToken() {
384      // get refresh token from cookie
385      return (document.cookie.split(';').find(x => x.includes('fakeRefreshToken')) || '=').split('=')[1];
386  }
387  }
388  }
389
390  export let fakeBackendProvider = {
391      // use fake backend in place of Http service for backend-less development
392      provide: HTTP_INTERCEPTORS,
393      useClass: FakeBackendInterceptor,
394      multi: true
395  };

```

```
src > app > _helpers > TS index.ts
```

```
1 export * from './app.initializer';
2 export * from './auth.guard';
3 export * from './error.interceptor';
4 export * from './fake-backend';
5 export * from './jwt.interceptor';
6 export * from './must-match.validator';
```

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


```
1 import { Injectable } from '@angular/core';
2 import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4
5 import { environment } from '@environments/environment';
6 import { AccountService } from '@app/_services';
7
8 @Injectable()
9 export class JwtInterceptor implements HttpInterceptor {
10     constructor(private accountService: AccountService) { }
11
12     intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
13         // add auth header with jwt if account is logged in and request is to the api url
14         const account = this.accountService.accountValue;
15         const isLoggedIn = account && account.jwtToken;
16         const isApiUrl = request.url.startsWith(environment.apiUrl);
17         if (isLoggedIn && isApiUrl) {
18             request = request.clone({
19                 setHeaders: { Authorization: `Bearer ${account.jwtToken}` }
20             });
21         }
22
23         return next.handle(request);
24     }
25 }
```

src > app > \_helpers > TS must-match.validator.ts > ...

```
1  import { AbstractControl } from '@angular/forms';
2
3  // custom validator to check that two fields match
4  export function MustMatch(controlName: string, matchingControlName: string) {
5      return (group: AbstractControl) => {
6          const control = group.get(controlName);
7          const matchingControl = group.get(matchingControlName);
8
9          if (!control || !matchingControl) {
10             return null;
11          }
12
13          // return if another validator has already found an error on the matchingControl
14          if (matchingControl.errors && !matchingControl.errors.mustMatch) {
15             return null;
16          }
17
18          // set error on matchingControl if validation fails
19          if (control.value !== matchingControl.value) {
20             matchingControl.setErrors({ mustMatch: true });
21          } else {
22             matchingControl.setErrors(null);
23          }
24          return null;
25      }
26  }
```

src > app > \_models > TS account.ts > ...


```
1  import { Role } from './role';
2
3  export class Account {
4      accountId?: string;
5      title?: string;
6      firstName?: string;
7      lastName?: string;
8      email?: string;
9      role?: Role;
10     jwtToken?: string;
11 }
```

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

```
1  export class Alert {
2      id?: string;
3      type?: AlertType;
4      message?: string;
5      autoClose?: boolean;
6      keepAfterRouteChange?: boolean;
7      fade?: boolean;
8
9      constructor(init?: Partial<Alert>) {
10         Object.assign(this, init);
11     }
12 }
13
14 export enum AlertType {
15     Success,
16     Error,
17     Info,
18     Warning
19 }
20
21 export class AlertOptions {
22     id?: string;
23     autoClose?: boolean;
24     keepAfterRouteChange?: boolean;
25 }
```

src > app > \_models > TS index.ts

```
1  export * from './account';
2  export * from './alert';
3  export * from './role';
```

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

```
1  export enum Role {
2      User = 'User',
3      Admin = 'Admin'
4  }
```

```
src > app > _services > TS account.service.ts > ...
1  import { Injectable } from '@angular/core';
2  import { Router } from '@angular/router';
3  import { HttpClient } from '@angular/common/http';
4  import { BehaviorSubject, Observable } from 'rxjs';
5  import { map, finalize } from 'rxjs/operators';
6
7  import { environment } from '@environments/environment';
8  import { Account } from '@app/_models';
9
10 const baseUrl = `${environment.apiUrl}/accounts`;
11
12 @Injectable({ providedIn: 'root' })
13 export class AccountService {
14     private accountSubject: BehaviorSubject<Account | null>;
15     public account: Observable<Account | null>;
16
17     constructor(
18         private router: Router,
19         private http: HttpClient
20     ) {
21         this.accountSubject = new BehaviorSubject<Account | null>(null);
22         this.account = this.accountSubject.asObservable();
23     }
24
25     public get accountValue() {
26         return this.accountSubject.value;
27     }
28
29     login(email: string, password: string) {
30         return this.http.post<any>(`${baseUrl}/authenticate`, { email, password }, { withCredentials: true })
31             .pipe(map(account => {
32                 this.accountSubject.next(account);
33                 this.startRefreshTokenTimer();
34                 return account;
35             }));
36     }
37 }
```



```
38     logout() {
39         this.http.post<any>(`${baseUrl}/revoke-token`, {}, { withCredentials: true }).subscribe();
40         this.stopRefreshTokenTimer();
41         this.accountSubject.next(null);
42         this.router.navigate(['/account/login']);
43     }
44
45     refreshToken() {
46         return this.http.post<any>(`${baseUrl}/refresh-token`, {}, { withCredentials: true })
47             .pipe(map((account) => {
48                 this.accountSubject.next(account);
49                 this.startRefreshTokenTimer();
50                 return account;
51             }));
52     }
53
54     register(account: Account) {
55         return this.http.post(`${baseUrl}/register`, account);
56     }
57
58     verifyEmail(token: string) {
59         return this.http.post(`${baseUrl}/verify-email`, { token });
60     }
61
62     forgotPassword(email: string) {
63         return this.http.post(`${baseUrl}/forgot-password`, { email });
64     }
65
66     validateResetToken(token: string) {
67         return this.http.post(`${baseUrl}/validate-reset-token`, { token });
68     }
69
70     resetPassword(token: string, password: string, confirmPassword: string) {
71         return this.http.post(`${baseUrl}/reset-password`, { token, password, confirmPassword });
72     }
```

```

74     getAll() {
75         return this.http.get<Account[]>(baseUrl);
76     }
77
78     getById(AccountId: string) {
79         return this.http.get<Account>(`${baseUrl}/${AccountId}`);
80     }
81
82     create(params: any) {
83         return this.http.post(baseUrl, params);
84     }
85
86     update(AccountId: string, params: any) {
87         return this.http.put(`${baseUrl}/${AccountId}`, params)
88             .pipe(map((account: any) => {
89                 // update the current account if it was updated
90                 if (account.AccountId === this.accountValue?.AccountId) {
91                     // publish updated account to subscribers
92                     account = { ...this.accountValue, ...account };
93                     this.accountSubject.next(account);
94                 }
95                 return account;
96             }));
97     }
98
99     delete(AccountId: string) {
100         return this.http.delete(`${baseUrl}/${AccountId}`)
101             .pipe(finally(() => {
102                 // auto logout if the logged in account was deleted
103                 if (AccountId === this.accountValue?.AccountId)
104                     this.logout();
105             }));
106     }

```

```

108     // helper methods
109
110     private refreshTokenTimeout?: any;
111
112     private startRefreshTokenTimer() {
113         // parse json object from base64 encoded jwt token
114         const jwtBase64 = this.accountValue!.jwtToken!.split('.')[1];
115         const jwtToken = JSON.parse(atob(jwtBase64));
116
117         // set a timeout to refresh the token a minute before it expires
118         const expires = new Date(jwtToken.exp * 1000);
119         const timeout = expires.getTime() - Date.now() - (60 * 1000);
120         this.refreshTokenTimeout = setTimeout(() => this.refreshToken().subscribe(), timeout);
121     }
122
123     private stopRefreshTokenTimer() {
124         clearTimeout(this.refreshTokenTimeout);
125     }
126 }

```

```

1  import { Injectable } from '@angular/core';
2  import { Observable, Subject } from 'rxjs';
3  import { filter } from 'rxjs/operators';
4
5  import { Alert, AlertOptions, AlertType } from '@app/_models';
6
7  @Injectable({ providedIn: 'root' })
8  export class AlertService {
9      private subject = new Subject<Alert>();
10     private defaultId = 'default-alert';
11
12     // enable subscribing to alerts observable
13     onAlert(id = this.defaultId): Observable<Alert> {
14         return this.subject.asObservable().pipe(filter(x => x && x.id === id));
15     }
16
17     // convenience methods
18     success(message: string, options?: AlertOptions) {
19         this.alert(new Alert({ ...options, type: AlertType.Success, message }));
20     }
21
22     error(message: string, options?: AlertOptions) {
23         this.alert(new Alert({ ...options, type: AlertType.Error, message }));
24     }
25
26     info(message: string, options?: AlertOptions) {
27         this.alert(new Alert({ ...options, type: AlertType.Info, message }));
28     }
29
30     warn(message: string, options?: AlertOptions) {
31         this.alert(new Alert({ ...options, type: AlertType.Warning, message }));
32     }
33
34     // core alert method
35     alert(alert: Alert) {
36         alert.id = alert.id || this.defaultId;
37         alert.autoClose = (alert.autoClose === undefined ? true : alert.autoClose);

```

```

38         this.subject.next(alert);
39     }
40
41     // clear alerts
42     clear(id = this.defaultId) {
43         this.subject.next(new Alert({ id }));
44     }
45 }

```

```
src > app > _services > TS index.ts
```

```

1  export * from './account.service';
2  export * from './alert.service';

```

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

```
1  import { NgModule } from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3
4  import { LayoutComponent } from './layout.component';
5  import { LoginComponent } from './login.component';
6  import { RegisterComponent } from './register.component';
7  import { VerifyEmailComponent } from './verify-email.component';
8  import { ForgotPasswordComponent } from './forgot-password.component';
9  import { ResetPasswordComponent } from './reset-password.component';
10
11  const routes: Routes = [
12    {
13      path: '', component: LayoutComponent,
14      children: [
15        { path: 'login', component: LoginComponent },
16        { path: 'register', component: RegisterComponent },
17        { path: 'verify-email', component: VerifyEmailComponent },
18        { path: 'forgot-password', component: ForgotPasswordComponent },
19        { path: 'reset-password', component: ResetPasswordComponent }
20      ]
21    }
22  ];
23
24  @NgModule({
25    imports: [RouterModule.forChild(routes)],
26    exports: [RouterModule]
27  })
28  export class AccountRoutingModule { }
```

```

src > app > account > TS account.module.ts > AccountModule
1  import { NgModule } from '@angular/core';
2  import { ReactiveFormsModule } from '@angular/forms';
3  import { CommonModule } from '@angular/common';
4
5  import { AccountRoutingModule } from './account-routing.module';
6  import { LayoutComponent } from './layout.component';
7  import { LoginComponent } from './login.component';
8  import { RegisterComponent } from './register.component';
9  import { VerifyEmailComponent } from './verify-email.component';
10 import { ForgotPasswordComponent } from './forgot-password.component';
11 import { ResetPasswordComponent } from './reset-password.component';
12
13 @NgModule({
14   imports: [
15     CommonModule,
16     ReactiveFormsModule,
17     AccountRoutingModule
18   ],
19   declarations: [
20     LayoutComponent,
21     LoginComponent,
22     RegisterComponent,
23     VerifyEmailComponent,
24     ForgotPasswordComponent,
25     ResetPasswordComponent
26   ]
27 })
28 export class AccountModule { }

```

```


src > app > account > forgot-password.component.html > h3.card-header
1  <h3 class="card-header">Forgot Password</h3>
2  <div class="card-body">
3    <form [formGroup]="form" (ngSubmit)="onSubmit()">
4      <div class="mb-3">
5        <label class="form-label">Email</label>
6        <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }" />
7        <div *ngIf="submitted && f.email.errors" class="invalid-feedback">
8          <div *ngIf="f.email.errors.required">Email is required</div>
9          <div *ngIf="f.email.errors.email">Email is invalid</div>
10       </div>
11     </div>
12     <div class="mb-3">
13       <button [disabled]="loading" class="btn btn-primary">
14         <span *ngIf="loading" class="spinner-border spinner-border-sm me-1"></span>
15         Submit
16       </button>
17       <a routerLink="../login" class="btn btn-link">Cancel</a>
18     </div>
19   </form>
20 </div>

```

```
src > app > account > Ts forgot-password.component.ts > ...
```

```
1  import { Component, OnInit } from '@angular/core';
2  import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3  import { first, finalize } from 'rxjs/operators';
4
5  import { AccountService, AlertService } from '@app/_services';
6
7  @Component({ templateUrl: 'forgot-password.component.html' })
8  export class ForgotPasswordComponent implements OnInit {
9      form!: FormGroup;
10     loading = false;
11     submitted = false;
12
13     constructor(
14         private formBuilder: FormBuilder,
15         private accountService: AccountService,
16         private alertService: AlertService
17     ) { }
18
19     ngOnInit() {
20         this.form = this.formBuilder.group({
21             email: ['', [Validators.required, Validators.email]]
22         });
23     }
24
25     // convenience getter for easy access to form fields
26     get f() { return this.form.controls; }
27
28     onSubmit() {
29         this.submitted = true;
30
31         // reset alerts on submit
32         this.alertService.clear();
33
34         // stop here if form is invalid
35         if (this.form.invalid) {
36             return;
37         }
```

```
37     }
38
39     this.loading = true;
40     this.accountService.forgotPassword(this.f.email.value)
41         .pipe(first())
42         .pipe(finalize(() => this.loading = false))
43         .subscribe({
44             next: () => this.alertService.success('Please check your email for password reset instructions'),
45             error: error => this.alertService.error(error)
46         });
47     }
48 }
```

```
src > app > account > <> layout.component.html >  div.container
1  <div class="container">
2    <div class="row">
3      <div class="col-lg-8 offset-lg-2 mt-5">
4        <div class="card m-3">
5          <router-outlet></router-outlet>
6        </div>
7      </div>
8    </div>
9  </div>
```

```
src > app > account > TS layout.component.ts > ...
1  import { Component } from '@angular/core';
2  import { Router } from '@angular/router';
3
4  import { AccountService } from '@app/_services';
5
6  @Component({ templateUrl: 'layout.component.html' })
7  export class LayoutComponent {
8    constructor(
9      private router: Router,
10     private accountService: AccountService
11   ) {
12     // redirect to home if already logged in
13     if (this.accountService.accountValue) {
14       this.router.navigate(['/']);
15     }
16   }
17 }
```

```

src > app > account > login.component.html > h3.card-header
1 <h3 class="card-header">Login</h3>
2 <div class="card-body">
3 <form [formGroup]="form" (ngSubmit)="onSubmit()"
4 <div class="mb-3">
5 <label class="form-label">Email</label>
6 <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }" />
7 <div *ngIf="submitted && f.email.errors" class="invalid-feedback">
8 <div *ngIf="f.email.errors.required">Email is required</div>
9 <div *ngIf="f.email.errors.email">Email is invalid</div>
10 </div>
11 </div>
12 <div class="mb-3">
13 <label class="form-label">Password</label>
14 <input type="password" formControlName="password" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }" />
15 <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
16 <div *ngIf="f.password.errors.required">Password is required</div>
17 </div>
18 </div>
19 <div class="row">
20 <div class="mb-3 col">
21 <button [disabled]="submitting" class="btn btn-primary">
22 <span *ngIf="submitting" class="spinner-border spinner-border-sm me-1"></span>
23 Login
24 </button>
25 <a routerLink="../register" class="btn btn-link">Register</a>
26 </div>
27 <div class="mb-3 col text-end">
28 <a routerLink="../forgot-password" class="btn btn-link pe-0">Forgot Password?</a>
29 </div>
30 </div>
31 </form>
32 </div>

```

```

src > app > account > TS login.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { Router, ActivatedRoute } from '@angular/router';
3 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
4 import { first } from 'rxjs/operators';
5
6 import { AccountService, AlertService } from '@app/_services';
7
8 @Component({ templateUrl: 'login.component.html' })
9 export class LoginComponent implements OnInit {
10     form!: FormGroup;
11     submitting = false;
12     submitted = false;
13
14     constructor(
15         private formBuilder: FormBuilder,
16         private route: ActivatedRoute,
17         private router: Router,
18         private accountService: AccountService,
19         private alertService: AlertService
20     ) { }
21
22     ngOnInit() {
23         this.form = this.formBuilder.group({
24             email: ['', [Validators.required, Validators.email]],
25             password: ['', Validators.required]
26         });
27     }
28
29     // convenience getter for easy access to form fields
30     get f() { return this.form.controls; }
31
32     onSubmit() {
33         this.submitted = true;
34
35         // reset alerts on submit
36         this.alertService.clear();

```



```

38     // stop here if form is invalid
39     if (this.form.invalid) {
40         return;
41     }
42
43     this.submitting = true;
44     this.accountService.login(this.f.email.value, this.f.password.value)
45         .pipe(first())
46         .subscribe({
47             next: () => {
48                 // get return url from query parameters or default to home page
49                 const returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
50                 this.router.navigateByUrl(returnUrl);
51             },
52             error: error => {
53                 this.alertService.error(error);
54                 this.submitting = false;
55             }
56         });
57     }
58 }

```

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

```

1  <h3 class="card-header">Register</h3>
2  <div class="card-body">
3      <form [formGroup]="form" (ngSubmit)="onSubmit()">
4          <div class="row">
5              <div class="mb-3 col-2">
6                  <label class="form-label">Title</label>
7                  <select formControlName="title" class="form-select" [ngClass]="{ 'is-invalid': submitted && f.title.errors }">
8                      <option value=""></option>
9                      <option value="Mr">Mr</option>
10                     <option value="Mrs">Mrs</option>
11                     <option value="Miss">Miss</option>
12                     <option value="Ms">Ms</option>
13                 </select>
14                 <div *ngIf="submitted && f.title.errors" class="invalid-feedback">
15                     <div *ngIf="f.title.errors.required">Title is required</div>
16                 </div>
17             </div>
18             <div class="mb-3 col-5">
19                 <label class="form-label">First Name</label>
20                 <input type="text" formControlName="firstName" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.firstName.errors }" />
21                 <div *ngIf="submitted && f.firstName.errors" class="invalid-feedback">
22                     <div *ngIf="f.firstName.errors.required">First Name is required</div>
23                 </div>
24             </div>
25             <div class="mb-3 col-5">
26                 <label class="form-label">Last Name</label>
27                 <input type="text" formControlName="lastName" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.lastName.errors }" />
28                 <div *ngIf="submitted && f.lastName.errors" class="invalid-feedback">
29                     <div *ngIf="f.lastName.errors.required">Last Name is required</div>
30                 </div>
31             </div>
32         </div>
33         <div class="mb-3">
34             <label class="form-label">Email</label>
35             <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }" />
36             <div *ngIf="submitted && f.email.errors" class="invalid-feedback">

```

```

37         <div *ngIf="f.email.errors.required">Email is required</div>
38         <div *ngIf="f.email.errors.email">Email must be a valid email address</div>
39     </div>
40 </div>
41 <div class="row">
42     <div class="mb-3 col">
43         <label class="form-label">Password</label>
44         <input type="password" formControlName="password" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }"
45         <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
46             <div *ngIf="f.password.errors.required">Password is required</div>
47             <div *ngIf="f.password.errors.minlength">Password must be at least 6 characters</div>
48         </div>
49     </div>
50     <div class="mb-3 col">
51         <label class="form-label">Confirm Password</label>
52         <input type="password" formControlName="confirmPassword" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.confirmPassw
53         <div *ngIf="submitted && f.confirmPassword.errors" class="invalid-feedback">
54             <div *ngIf="f.confirmPassword.errors.required">Confirm Password is required</div>
55             <div *ngIf="f.confirmPassword.errors.mustMatch">Passwords must match</div>
56         </div>
57     </div>
58 </div>
59 <div class="mb-3 form-check">
60     <input type="checkbox" formControlName="acceptTerms" id="acceptTerms" class="form-check-input" [ngClass]="{ 'is-invalid': submitted && f
61     <label for="acceptTerms" class="form-check-label">Accept Terms & Conditions</label>
62     <div *ngIf="submitted && f.acceptTerms.errors" class="invalid-feedback">Accept Ts & Cs is required</div>
63 </div>
64 <div class="mb-3">
65     <button [disabled]="submitting" class="btn btn-primary">
66         <span *ngIf="submitting" class="spinner-border spinner-border-sm me-1"></span>
67         Register
68     </button>

```

```

69         <a routerLink="../login" href="" class="btn btn-link">Cancel</a>
70     </div>
71 </form>
72 </div>

```

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

```
1  import { Component, OnInit } from '@angular/core';
2  import { Router, ActivatedRoute } from '@angular/router';
3  import { FormBuilder, FormGroup, Validators } from '@angular/forms';
4  import { first } from 'rxjs/operators';
5
6  import { AccountService, AlertService } from '@app/_services';
7  import { MustMatch } from '@app/_helpers';
8
9  @Component({ templateUrl: 'register.component.html' })
10 export class RegisterComponent implements OnInit {
11     form!: FormGroup;
12     submitting = false;
13     submitted = false;
14
15     constructor(
16         private formBuilder: FormBuilder,
17         private route: ActivatedRoute,
18         private router: Router,
19         private accountService: AccountService,
20         private alertService: AlertService
21     ) { }
22
23     ngOnInit() {
24         this.form = this.formBuilder.group({
25             title: ['', Validators.required],
26             firstName: ['', Validators.required],
27             lastName: ['', Validators.required],
28             email: ['', [Validators.required, Validators.email]],
29             password: ['', [Validators.required, Validators.minLength(6)]],
30             confirmPassword: ['', Validators.required],
31             acceptTerms: [false, Validators.requiredTrue]
32         }, {
33             validator: MustMatch('password', 'confirmPassword')
34         });
35     }
36 }
```

```

37 // convenience getter for easy access to form fields
38 get f() { return this.form.controls; }
39
40 onSubmit() {
41   this.submitted = true;
42
43   // reset alerts on submit
44   this.alertService.clear();
45
46   // stop here if form is invalid
47   if (this.form.invalid) {
48     return;
49   }
50
51   this.submitting = true;
52   this.accountService.register(this.form.value)
53     .pipe(first())
54     .subscribe({
55       next: () => {
56         this.alertService.success('Registration successful, please check your email for verification instructions', { keepAfterRouteChange: true });
57         this.router.navigate(['../login'], { relativeTo: this.route });
58       },
59       error: error => {
60         this.alertService.error(error);
61         this.submitting = false;
62       }
63     });
64 }
65 }

```

```

src > app > account > reset-password.component.html > h3.card-header
1 <h3 class="card-header">Reset Password</h3>
2 <div class="card-body">
3   <div *ngIf="tokenStatus == TokenStatus.Validating">
4     Validating token...
5   </div>
6   <div *ngIf="tokenStatus == TokenStatus.Invalid">
7     Token validation failed, if the token has expired you can get a new one at the <a routerLink='../forgot-password'>forgot password</a> page.
8   </div>
9   <form *ngIf="tokenStatus == TokenStatus.Valid" [formGroup]="form" (ngSubmit)="onSubmit()">
10     <div class="mb-3">
11       <label class="form-label">Password</label>
12       <input type="password" formControlName="password" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }" />
13       <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
14         <div *ngIf="f.password.errors.required">Password is required</div>
15         <div *ngIf="f.password.errors.minlength">Password must be at least 6 characters</div>
16       </div>
17     </div>
18     <div class="mb-3">
19       <label class="form-label">Confirm Password</label>
20       <input type="password" formControlName="confirmPassword" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.confirmPassword.errors }" />
21       <div *ngIf="submitted && f.confirmPassword.errors" class="invalid-feedback">
22         <div *ngIf="f.confirmPassword.errors.required">Confirm Password is required</div>
23         <div *ngIf="f.confirmPassword.errors.mustMatch">Passwords must match</div>
24       </div>
25     </div>
26     <div class="mb-3">
27       <button [disabled]="loading" class="btn btn-primary">
28         <span *ngIf="loading" class="spinner-border spinner-border-sm me-1"></span>
29         Reset Password
30       </button>
31       <a routerLink='../login' class="btn btn-link">Cancel</a>
32     </div>
33   </form>
34 </div>

```

```
src > app > account > TS reset-password.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { Router, ActivatedRoute } from '@angular/router';
3  import { FormBuilder, FormGroup, Validators } from '@angular/forms';
4  import { first } from 'rxjs/operators';
5
6  import { AccountService, AlertService } from '@app/_services';
7  import { MustMatch } from '@app/_helpers';
8
9  enum TokenStatus {
10     Validating,
11     Valid,
12     Invalid
13 }
14
15 @Component({ templateUrl: 'reset-password.component.html' })
16 export class ResetPasswordComponent implements OnInit {
17     TokenStatus = TokenStatus;
18     tokenStatus = TokenStatus.Validating;
19     token?: string;
20     form!: FormGroup;
21     loading = false;
22     submitted = false;
23
24     constructor(
25         private formBuilder: FormBuilder,
26         private route: ActivatedRoute,
27         private router: Router,
28         private accountService: AccountService,
29         private alertService: AlertService
30     ) { }
31
32     ngOnInit() {
33         this.form = this.formBuilder.group({
34             password: ['', [Validators.required, Validators.minLength(6)]],
35             confirmPassword: ['', Validators.required],
36         }, {
37             validator: MustMatch('password', 'confirmPassword')
```

```

38     });
39
40     const token = this.route.snapshot.queryParams['token'];
41
42     // remove token from url to prevent http referer leakage
43     this.router.navigate([], { relativeTo: this.route, replaceUrl: true });
44
45     this.accountService.validateResetToken(token)
46       .pipe(first())
47       .subscribe({
48         next: () => {
49           this.token = token;
50           this.tokenStatus = TokenStatus.Valid;
51         },
52         error: () => {
53           this.tokenStatus = TokenStatus.Invalid;
54         }
55       });
56   }
57
58   // convenience getter for easy access to form fields
59   get f() { return this.form.controls; }
60
61   onSubmit() {
62     this.submitted = true;
63
64     // reset alerts on submit
65     this.alertService.clear();
66
67     // stop here if form is invalid
68     if (this.form.invalid) {
69       return;
70     }
71
72     this.loading = true;

```

```

73     this.accountService.resetPassword(this.token!, this.f.password.value, this.f.confirmPassword.value)
74       .pipe(first())
75       .subscribe({
76         next: () => {
77           this.alertService.success('Password reset successful, you can now login', { keepAfterRouteChange: true });
78           this.router.navigate(['../login'], { relativeTo: this.route });
79         },
80         error: error => {
81           this.alertService.error(error);
82           this.loading = false;
83         }
84       });
85   }
86 }

```

```

src > app > account > verify-email.component.html > h3.card-header
1   <h3 class="card-header">Verify Email</h3>
2   <div class="card-body">
3     <div *ngIf="emailStatus == EmailStatus.Verifying">
4       Verifying...
5     </div>
6     <div *ngIf="emailStatus == EmailStatus.Failed">
7       Verification failed, you can also verify your account using the <a routerLink="forgot-password">forgot password</a> page.
8     </div>
9   </div>

```

```

src > app > account > TS verify-email.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { Router, ActivatedRoute } from '@angular/router';
3  import { first } from 'rxjs/operators';
4
5  import { AccountService, AlertService } from '@app/_services';
6
7  enum EmailStatus {
8      Verifying,
9      Failed
10 }
11
12 @Component({ templateUrl: 'verify-email.component.html' })
13 export class VerifyEmailComponent implements OnInit {
14     EmailStatus = EmailStatus;
15     emailStatus = EmailStatus.Verifying;
16
17     constructor(
18         private route: ActivatedRoute,
19         private router: Router,
20         private accountService: AccountService,
21         private alertService: AlertService
22     ) { }
23
24     ngOnInit() {
25         const token = this.route.snapshot.queryParams['token'];
26
27         // remove token from url to prevent http referer leakage
28         this.router.navigate([], { relativeTo: this.route, replaceUrl: true });
29
30         this.accountService.verifyEmail(token)
31             .pipe(first())
32             .subscribe({
33                 next: () => {
34                     this.alertService.success('Verification successful, you can now login', { keepAfterRouteChange: true });
35                     this.router.navigate(['../login'], { relativeTo: this.route });
36                 },
37                 error: () => {
38                     this.emailStatus = EmailStatus.Failed;
39                 }
40             });
41     }
42 }

```

```

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" respectively by the Angular CLI. To control ECMA version and features use the Browserslist 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.