Routing & Securing Angular Applications

Agenda

Advanced Routing – Interceptors, Route Guards, Global Error Handling

App Initialization

Routing & NgRx

SPA Security Big Picture

Auth Standards / Terms

Cloud Auth (Firebase, Azure)

Angular Auth Building Blocks

Implementing Token Based Authentication using NgRx

Advanced Routing Concepts

Location Service

The Angular Location service is used to interact with the browser URL & history

We can use it to track the URL changes, to read the current URL, modify the URL, go back or forward through the browser history,

Common methods:

- back(), forward()
- path()
- go() – only for path that are available in history
- onUrlChange()

Route Guards Recap

Route Guards allow or deny route execution before a Routing event takes place

Implement one of the following Interfaces:

- CanActivate
- CanActivateChild
- CanDeactivate
- CanLoad
- Resolve (Data Prefetch)

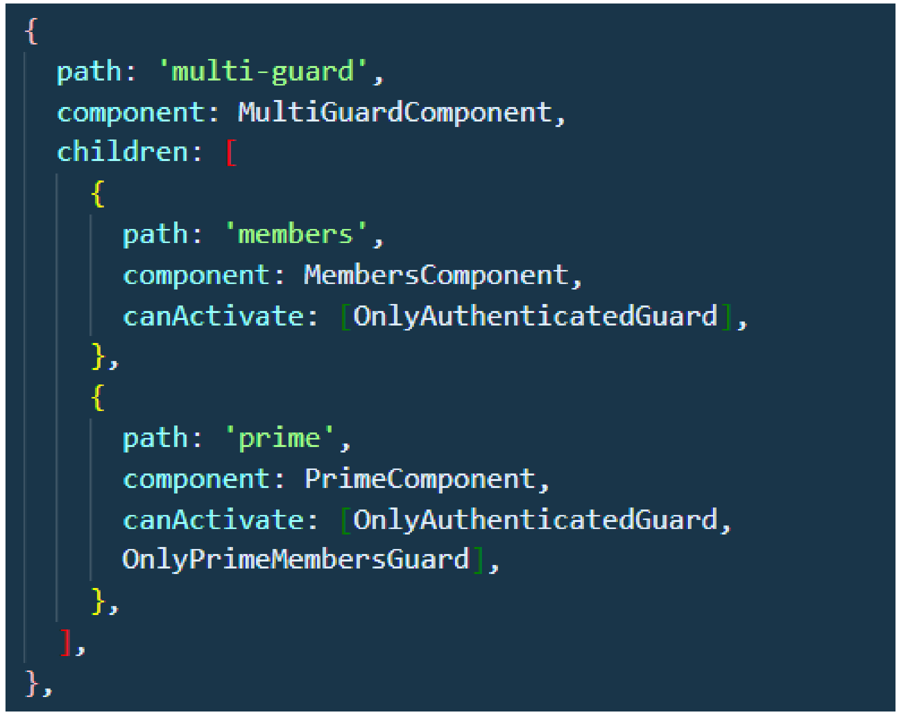
```
ng g guard demos/samples/multi-guard/OnlyAuthenticated --implements CanActivate
```

Using multible Route Guards

It is possible to combine multiple guards to protect a route

Might reduce complexibility in a specific guard

If one guard fails the later ones are not executed



```
{
  path: 'multi-guard',
  component: MultiGuardComponent,
  children: [
    {
      path: 'members',
      component: MembersComponent,
      canActivate: [OnlyAuthenticatedGuard],
    },
    {
      path: 'prime',
      component: PrimeComponent,
      canActivate: [OnlyAuthenticatedGuard,
      OnlyPrimeMembersGuard],
    },
  ],
},
```

Interceptors

Interceptors are used in Angular to Intercept HttpRequest / HttpResponse

Several Interceptor Use Cases

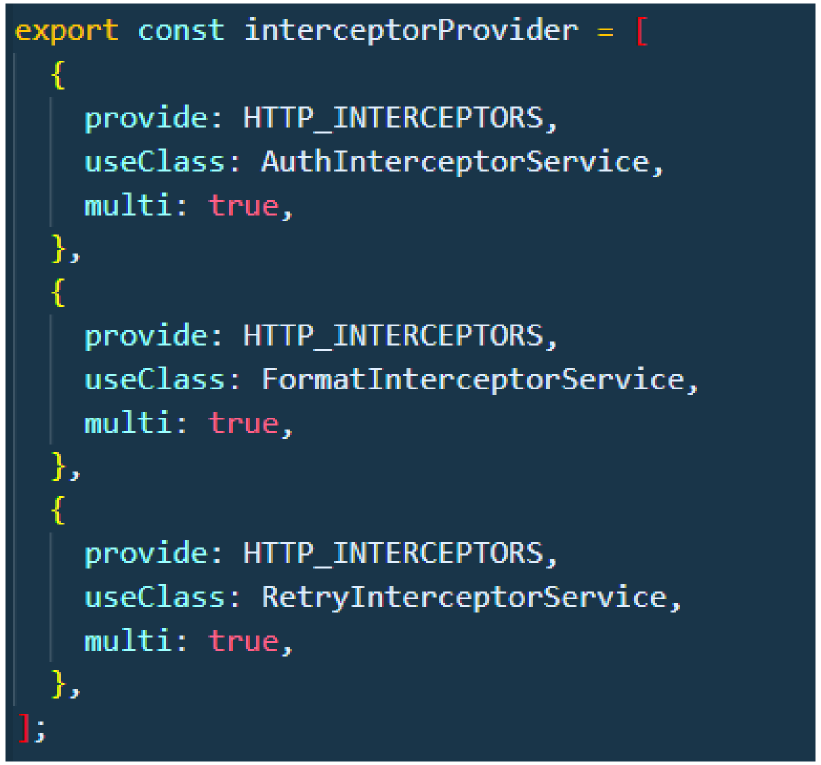
- Authentication / Session Interceptor
  - Adds the token to the HttpHeader
- Request Format Interceptor
- AJAX animation interceptor
- Notify error interceptor
- TimeStamp interceptor
- Retry Request Interceptor

Using multible Interceptors

Multiple Interceptors can be added in Angular with each Interceptor having different purpose

Could be done in one Interceptor but reduces complexity

Interceptors can be combined using a simple class



```
export const interceptorProvider = [
  {
    provide: HTTP_INTERCEPTORS,
    useClass: AuthInterceptorService,
    multi: true,
  },
  {
    provide: HTTP_INTERCEPTORS,
    useClass: FormatInterceptorService,
    multi: true,
  },
  {
    provide: HTTP_INTERCEPTORS,
    useClass: RetryInterceptorService,
    multi: true,
  },
];
```

Global Error Handling

Handling error is an important part of the application design

We distinguish:

- Http Errors
- Javascript / TypeScript Errors
  - When we reference a non-existent variable.
  - The value provided is not in the range of allowed values.
  - When Interpreting syntactically invalid code
  - When a value is not of the expected type
  - Internal errors in the JavaScript engine

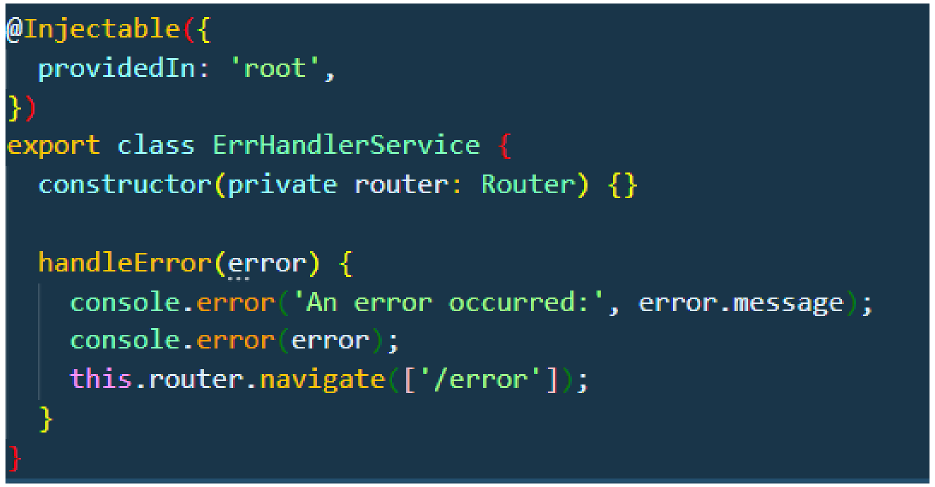
Implementing Global Error Handling

The default Error handling in Angular is handled by ErrorHandler class and is part of @angular/core

Has to be registered as a provider: { provide: ErrorHandler, useClass: ErrHandlerService },

It advisable to create our own global error handler as a service to be able to

- Log Errors
- Display a msg to the use or,
- Redirect the user to an Error page



```
@Injectable({
  providedIn: 'root',
})
export class ErrHandlerService {
  constructor(private router: Router) {}

  handleError(error) {
    console.error('An error occurred:', error.message);
    console.error(error);
    this.router.navigate(['/error']);
  }
}
```

Catch error globally using HTTP Interceptor

Whenever the error occurs in an HTTP operation, the Angular wraps it in an HttpErrorResponse

Global HTTP error handling is done using the Angular HTTP Interceptor

We can catch the HTTP Errors at three different places.

- Component
- Service
- Globally

App Initialization

Angular Injector

Is responsible instantiating the dependency and injecting into the component or service

Looks for the dependency in the Angular Providers using the token

The Angular Providers array returns the Provider, which contains the information about how to create the instance of the dependency

The Injector creates the instance and injects it into Component or service

Dependency Injection Recap

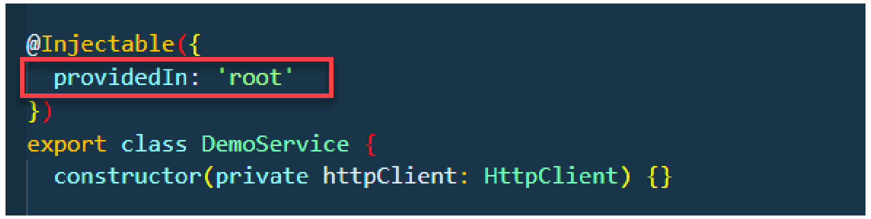
The Angular injector uses the DI token to locate the dependencies in the Angular Provider using:

- `providers: [ProductService]`
- `providers :[{ provide: token, useClass: SomeService }]`

Several Token Types:

- Type, String, Inject, Factory

Many times in modern ngApps you are not working with providers because you use the providedIn flag



```
@Injectable({
  providedIn: 'root'
})
export class DemoService {
  constructor(private httpClient: HttpClient) {}
```

App Initializer

Angular has a hook in its process of initialization called App Initializer

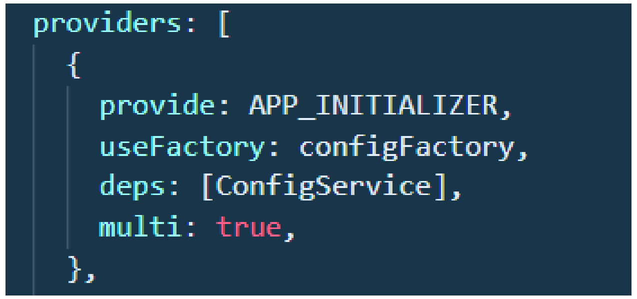
- App Initializer is part of the Angular Dependency Injection model

The provided functions are injected at application startup and executed during app initialization

The State of App Initialization is provided by ApplicationInitStatus (returns Promise)

Use Cases

- Config Injection from Optimization Module of this class
- Preload Lookup Tables on App Start
- Create a factory function that loads language data
- …



```
providers: [
  {
    provide: APP_INITIALIZER,
    useFactory: configFactory,
    deps: [ConfigService],
    multi: true,
  },
```

Dynamic Component Loading

Dynamic Components

Component templates are not always fixed and can be loaded during runtime

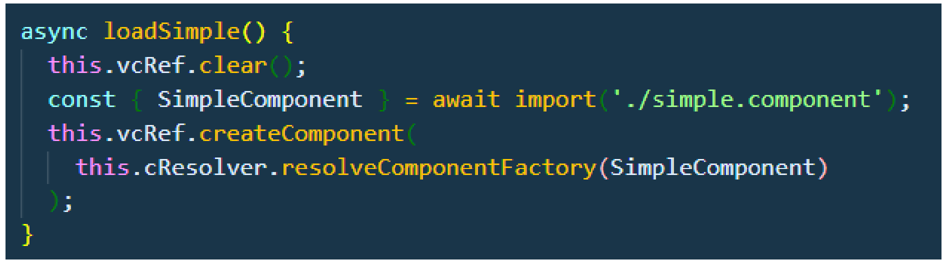
ComponentFactoryResolver to add components dynamically

Components that can be lazy loaded can be created using Angular CLI

- ng g c NAME --flat --skip-import --skip-selector

Loading is done using:

- ViewContainerRef
- ComponentFactoryResolver



```
async loadSimple() {
  this.vcRef.clear();
  const { SimpleComponent } = await import('./simple.component');
  this.vcRef.createComponent(
    this.cResolver.resolveComponentFactory(SimpleComponent)
  );
}
```

ViewContainerRef

Represents a container where one or more views can be attached to a component.

It can contain host views (created by instantiating a component with the createComponent() method), and embedded views (created by instantiating a TemplateRef with the createEmbeddedView() method).

A view container instance can contain other view containers, creating a view hierarchy

- createEmbeddedView() instantiates an embedded view and inserts it into this container.
- createComponent() instantiates a single component and inserts its host view into this container at a specified index.

ComponentFactory & ComponentFactoryResolver

ComponentFactory

- Base class for a factory that can create a component dynamically.
- Instantiate a factory for a given type of component with resolveComponentFactory().
- Use the resulting ComponentFactory.create() method to create a component of that type.

ComponentFactoryResolver

- A simple registry that maps Components to generated ComponentFactory classes that can be used to create instances of components.
- Use to obtain the factory for a given component type, then use the factory's create() method to create a component of that type

Routing & NgRx

@ngrx/router-store

Bindings to connect the Angular Router with Store

Allows listening to Router Events & Params using NgRx

Configured in Root Module

- StoreRouterConnectingModule.forRoot()

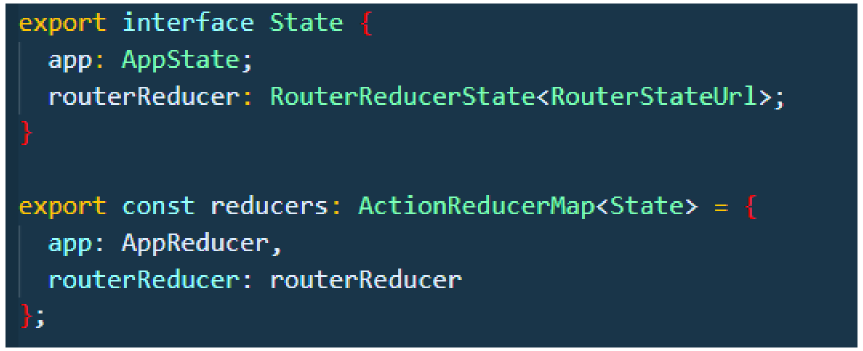
Can be added using:

- ng add @ngrx/router-store

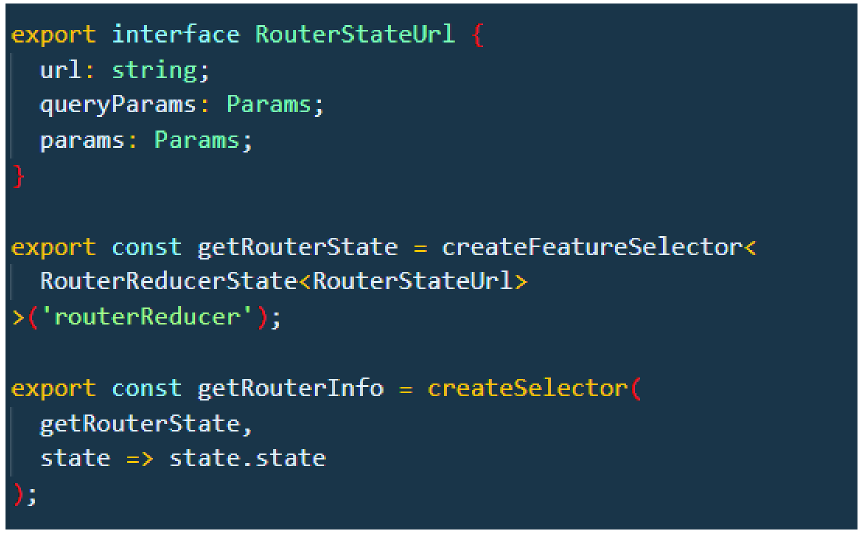
Setup Router State

In Order to bind Router State to State:

- Implement an Interface representing your desired State
- Create Feature and further Selectors
- Create a custom Serializer for your Router State
- Add "routerReducer" (required name) to global App State



```
export interface State {
  app: AppState;
  routerReducer: RouterReducerState<RouterStateUrl>;
}

export const reducers: ActionReducerMap<State> = {
  app: AppReducer,
  routerReducer: routerReducer
};
```



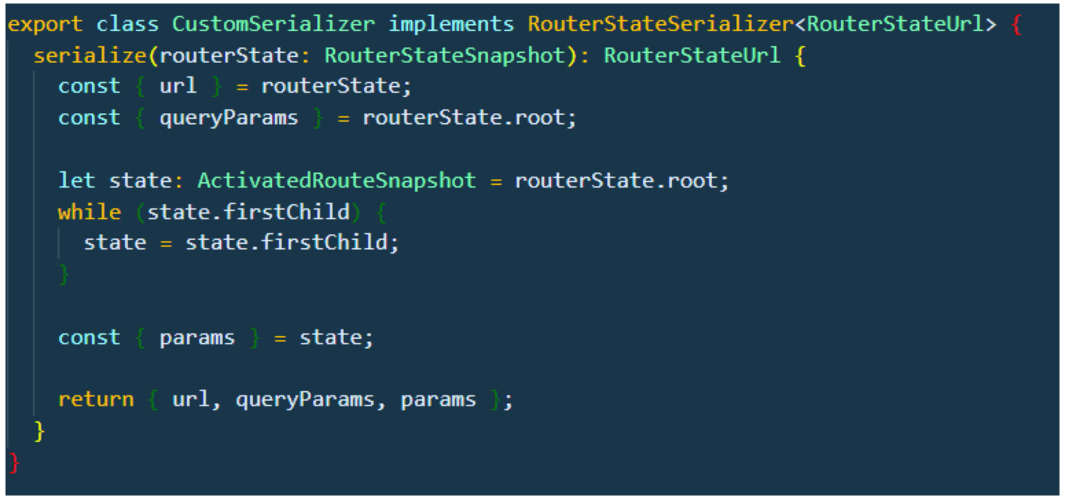
```
export interface RouterStateUrl {
  url: string;
  queryParams: Params;
  params: Params;
}

export const getRouterState = createFeatureSelector<
  RouterReducerState<RouterStateUrl>
>('routerReducer');

export const getRouterInfo = createSelector(
  getRouterState,
  state => state.state
);
```

Router State Serializer

This router state serializer, serializes the URL together with the ActivatedRouteSnapshot from Angular Router.

Default Implementation provided

Override using a shape data that fits your needs



```
export class CustomSerializer implements RouterStateSerializer<RouterStateUrl> {
  serialize(routerState: RouterStateSnapshot): RouterStateUrl {
    const { url } = routerState;
    const { queryParams } = routerState.root;

    let state: ActivatedRouteSnapshot = routerState.root;
    while (state.firstChild) {
      state = state.firstChild;
    }

    const { params } = state;

    return { url, queryParams, params };
  }
}
```

Navigation Actions

Now that you are using NgRx for Routing you can also create & dispatch your own Routing Actions:

- Go Forward
- Go Back
- Go To a spcific URL with id parm
- Go To error page
- ....

SPA Security Big Picture

Authentication

The process of verifying that "you are who you say you are"

Authentication in SPAs typically uses Token Based Auth

- JWT (Json Web Token) defines a token format
- OAuth 2.0 defines a protocol, i.e. specifies how tokens are transferred
  - Can use Jwt

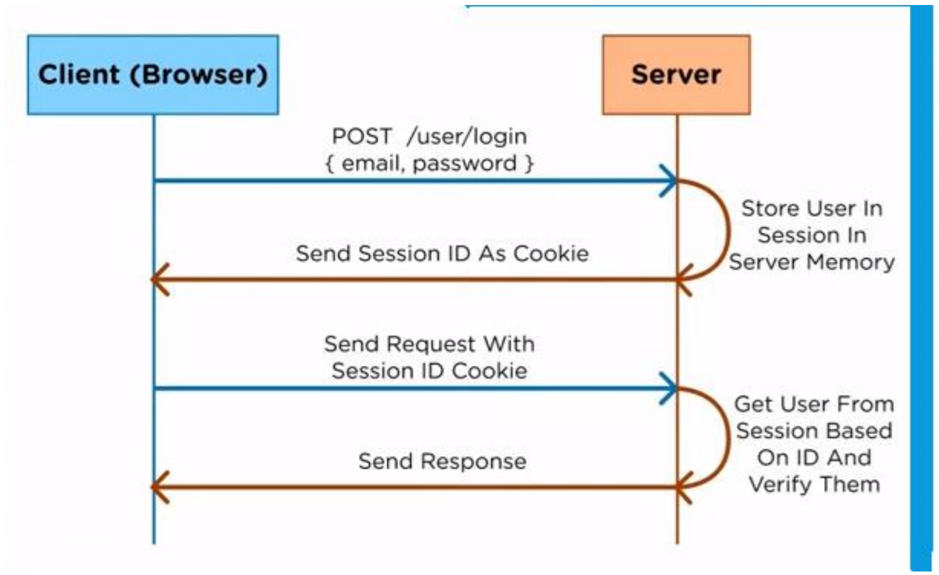
OAuth 2.0 is supported by almost all cloud services like

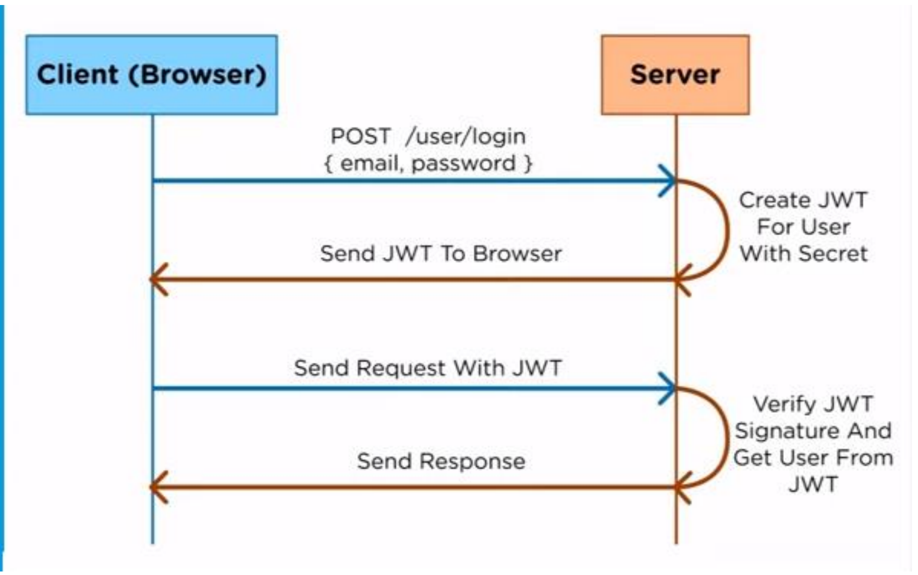
- Facebook
- Google
- Microsoft (Azure AD, Office 365)

Authentication: Classic vs SPAs

SPAs use Token-based-Auth instead of Sessions / Cookies

Traditional Server Side Sec not optimal in SPAs - No Page Change


Client (Browser)
Server
POST /user/login
{ email, password }
Store User In Session In Server Memory
Send Session ID As Cookie
Send Request With Session ID Cookie
Get User From Session Based On ID And Verify Them
Send Response



Authorization

Authorization is the function of specifying access rights/privileges to resources

Authorization takes place when

- The user wants to navigate from one page (Route) to another
  - Typically done by using Route Guards
- Angular wants to consume an Api
  - Authentication between Angular + Api can be archieved using Tokens
  - Use Interceptors to automate Token sending

Transport Security

Achieved using SSL (TLS)

Free Root Trust Certificates available using Let‘s encrypt

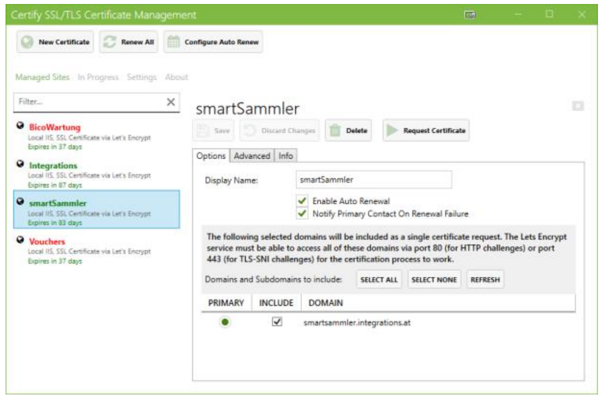
- https://letsencrypt.org/

Certbot

- Linux based Cert request / renewal

Certify SSL Manager

- Easy to use IIS Tool with Automatic Certificate Renewal
- https://certifytheweb.com/



BicoWartung
Local IIS, SSL Certificate via Let's Encrypt
Expires in 37 days

Integrations
Local IIS, SSL Certificate via Let's Encrypt
Expires in 87 days

smartSammler
Local IIS, SSL Certificate via Let's Encrypt
Expires in 83 days

Vouchers
Local IIS, SSL Certificate via Let's Encrypt
Expires in 37 days

smartSammler

Display Name: smartSammler

[✓] Enable Auto Renewal
[✓] Notify Primary Contact On Renewal Failure

The following selected domains will be included as a single certificate request. The Lets Encrypt service must be able to access all of these domains via port 80 (for HTTP challenges) or port 443 (for TLS-SNI challenges) for the certification process to work.

Domains and Subdomains to include:

PRIMARY	INCLUDE	DOMAIN
●	✓	smartsammler.integrations.at

Auth Standards & Terms

Identity Provider

System that does Identity Management

- Traditionally Active Directory

In a more Cloud based Approach

- Social Logins
- Cloud based Logins
  - Azure AD
  - Firebase
  - …

Cloud Options for Token-based Auth

Firebase (Google Cloud)

- Easy to learn, free – good for learning, small Apps
- npm: firebase, @angular/fire

Azure AD

- Commercial, might be in use alreadey (Azure, Office 365)
- 2 SDKs available: ADAL, MSAL

OpenID Connect

A simple identity layer on top of the OAuth 2.0 protocol, which allows computing clients to verify the identity of an end-user based on the authentication performed by an authorization server

Enables Single Sign-on

Has become the leading standard for single sign-on and identity provision on the Internet by using:

- simple JSON-based identity tokens (JWT),
- delivered via the OAuth 2.0 protocol

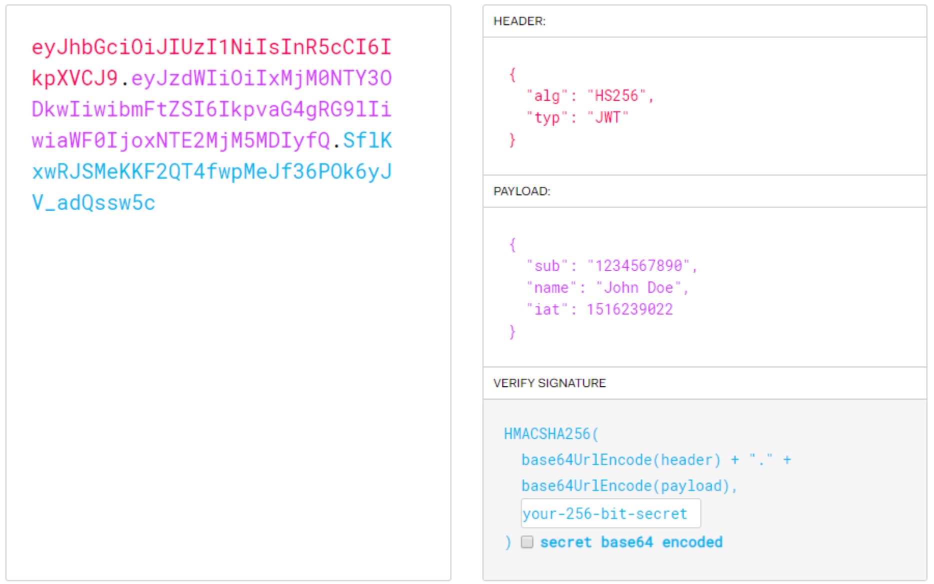
JSON Web Tokens

An open, industry standard for representing claims securely between two parties

Can be sent through a URL, POST parameter, or inside an HTTP header

Contains all the required information about the user

Doku @ https://jwt.io/



```
eyJhbGciOiJIUzI1NiIsInR5cCI6I
kpXVCJ9.eyJzdWIiOiIxMjM0NTY3O
DkwIiwibmFtZSI6IkpvaG4gRG9lIi
wiaWF0IjoxNTE2MjM5MDIyfQ.SflK
xwRJSMeKKF2QT4fwpMeJf36POk6yJ
V_adQssw5c
```

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Token Types

ID Token

- JWT encoded Identity Information about the user

Access Token

- Used to access 3rd-party ressources without any further Identification
  - ie: MS Graph, SharePoint, Azure Blob Storage, ...
- Attached to the http-request in the header: Manually | Using Interceptor

Refresh Token

- Not used in Angular because of Implict Flow – Silent Renew is used

OAuth 2.0

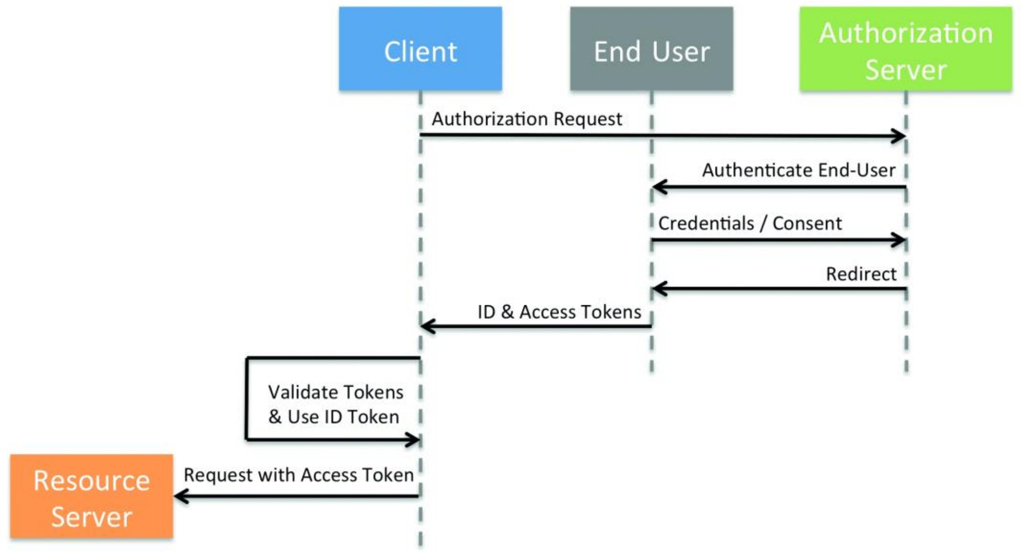
OAuth 2.0 is the industry-standard protocol for authorization

Defines Token Flows - How you get your tokens

- Authorization Code Grant
- Implicit Flow
- Client Credentials Flow
- ...

Flow Depends on Use Case

- Web / Mobile



OpentID Connect Flows

Defines how Auth Proces is implemented

- Authorization Code Flow
  - Involves an initial browser redirection to / from the OP for user authentication and consent
  - Then a second back-channel request to retrieve the ID token
- Implicit Flow
  - ID token is received directly with the redirection response from the OP
- Hybrid

Authorization Code & Hybrid use persisted Secret

- Not secure to persist Secret in JavaScript App (Dev Tools)
- -> Implicit Flow used in Angular

Firebase Auth

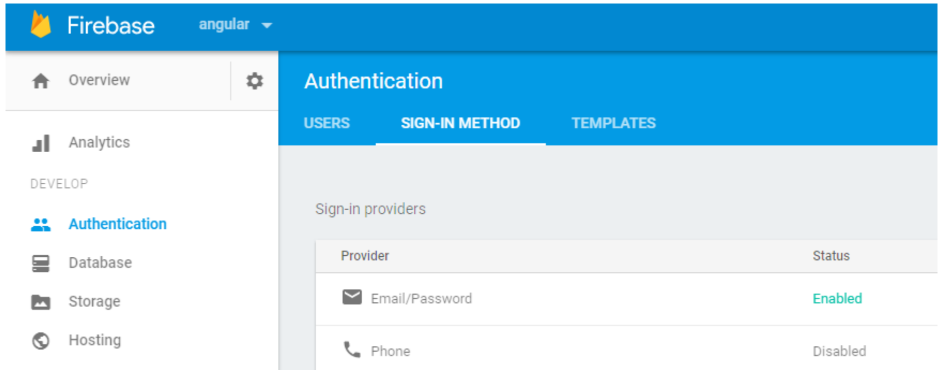
Firebase

Firebase is part of Googles Cloud Platform

Firebase provides an easy to understand approach to get into using Tokens

Other Features:

- Cloud Messaging
- ML
- Hosting
- Cloud Functions



Provider	Status
Email/Password	Enabled
Phone	Disabled

Firebase SDK

Provides the tools and infrastructure you need to develop, grow, and earn money from your app.

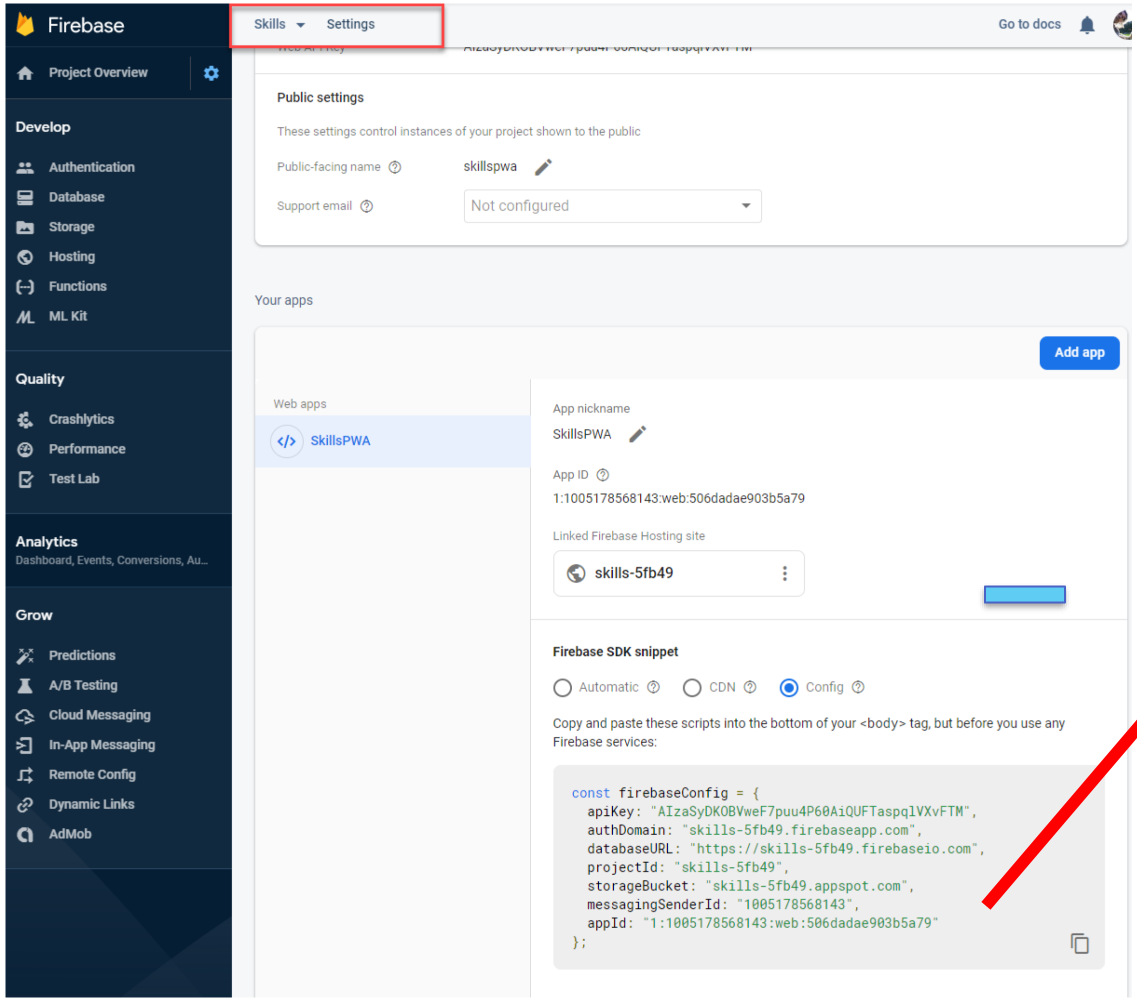
npm packages:

- firebase@6.x
- @angular/fire@6.x (Angular Wrapper)

Consists of:

- Firebase Realtime Database / Cloud Firestore
- Firebase Storage
- Firebase Cloud Messaging
- Firebase Authentication

Firebase Config



Public settings

These settings control instances of your project shown to the public

Public-facing name: skillspwa

Support email: Not configured

Your apps

Web apps

SkillsPWA

App nickname: SkillsPWA

App ID: 1:1005178568143:web:506dadae903b5a79

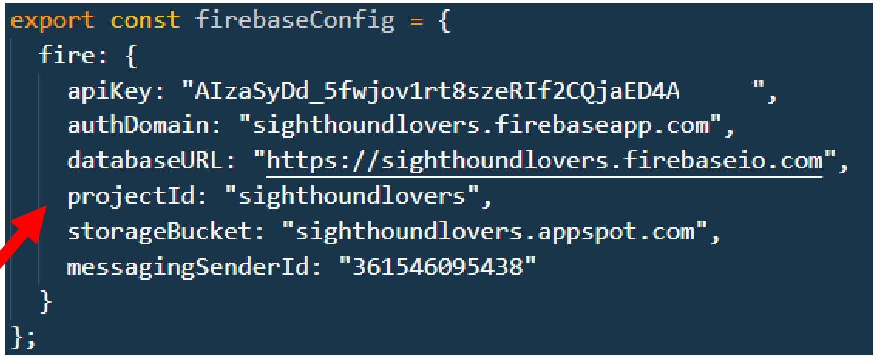
Linked Firebase Hosting site: skills-5fb49

Firebase SDK snippet

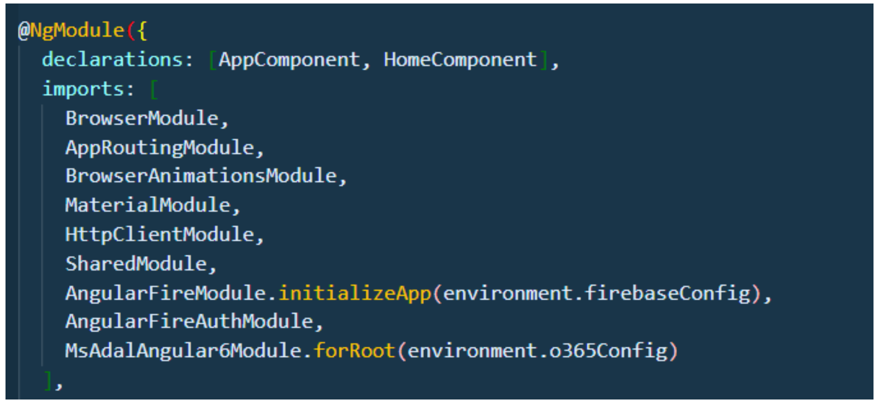
Automatic CDN [✓ Config]

Copy and paste these scripts into the bottom of your <body> tag, but before you use any Firebase services:

```
const firebaseConfig = {
  apiKey: "AIzaSyDKOBVweF7puu4P60AiQUFTaspqlVXvFTM",
  authDomain: "skills-5fb49.firebaseapp.com",
  databaseURL: "https://skills-5fb49.firebaseio.com",
  projectId: "skills-5fb49",
  storageBucket: "skills-5fb49.appspot.com",
  messagingSenderId: "1005178568143",
  appId: "1:1005178568143:web:506dadae903b5a79"
};
```



```
export const firebaseConfig = {
  fire: {
    apiKey: "AIzaSyDd_5fwjov1rt8szeRIf2CQjaED4A     ",
    authDomain: "sighthoundlovers.firebaseapp.com",
    databaseURL: "https://sighthoundlovers.firebaseio.com",
    projectId: "sighthoundlovers",
    storageBucket: "sighthoundlovers.appspot.com",
    messagingSenderId: "361546095438"
  }
};
```



```
@NgModule({
  declarations: [AppComponent, HomeComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MaterialModule,
    HttpClientModule,
    SharedModule,
    AngularFireModule.initializeApp(environment.firebaseConfig),
    AngularFireAuthModule,
    MsAdalAngular6Module.forRoot(environment.o365Config)
  ],
```

Steps to implement

Import Firebase Modules to app.module.ts

- Initialize with Firebase Config

Create AuthService, implement

- createUserWithEmailAndPassword()
- signInWithEmailAndPassword()
- signOut()

Create Login, Register Page

Sign-In methods

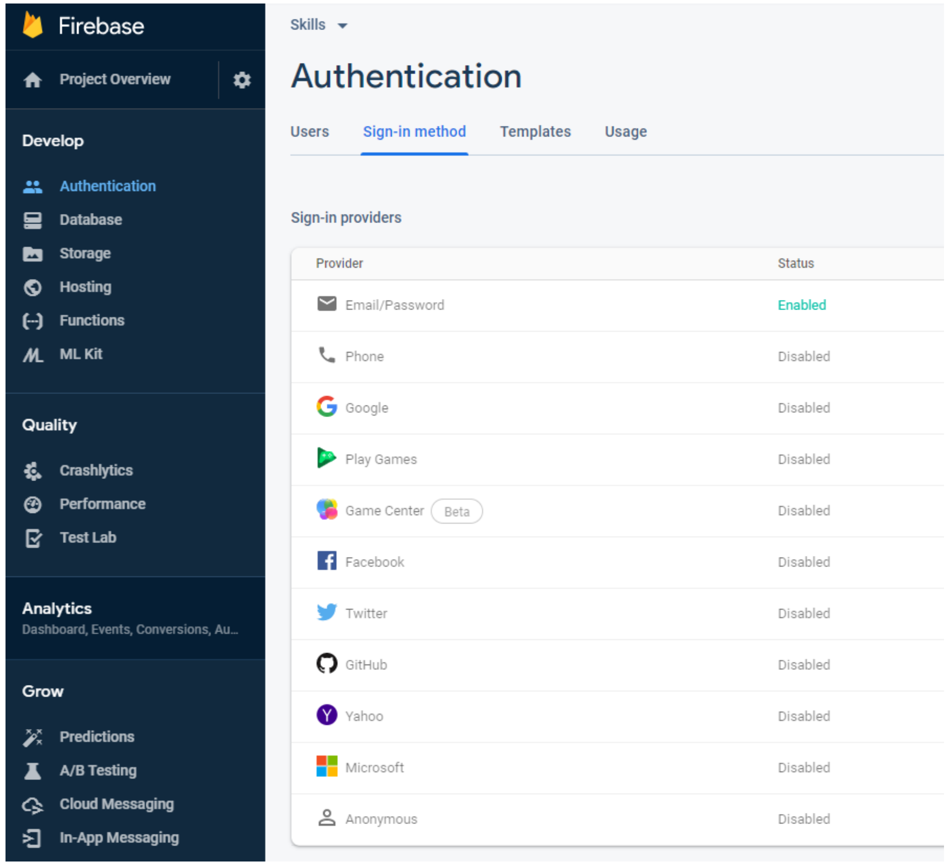
Default Method is Email / Password Auth

Allows Facebook & Google Login as a Sign-In method

Account Management is still done in Firebase

Requires AppID & AppSecret

- Configure at https://developers.facebook.com/



Authentication

Sign-in providers

Provider	Status
Email/Password	Enabled
Phone	Disabled
Google	Disabled
Play Games	Disabled
Game Center Beta	Disabled
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Yahoo	Disabled
Microsoft	Disabled
Anonymous	Disabled

Building Blocks

FireBase User

- Can have custom claims

AngularFireAuth

- Firebase Auth Service abstraction
  - createUserWithEmailAndPassword
  - signInWithEmailAndPassword / signOut

AngularFireAuthGuard

- Provides a prebuilt canActivate Router Guard
- Probides helper methods: hasCustomClaim, redirectLoggedInTo, ...

Azure AD Auth

Azure Active Directory

Is Microsoft’s multi-tenant cloud based directory and identity management service.

Allows sync / forward login requests with on-premises AD

Allows social identities using Azure AD B2C

Can be used for:

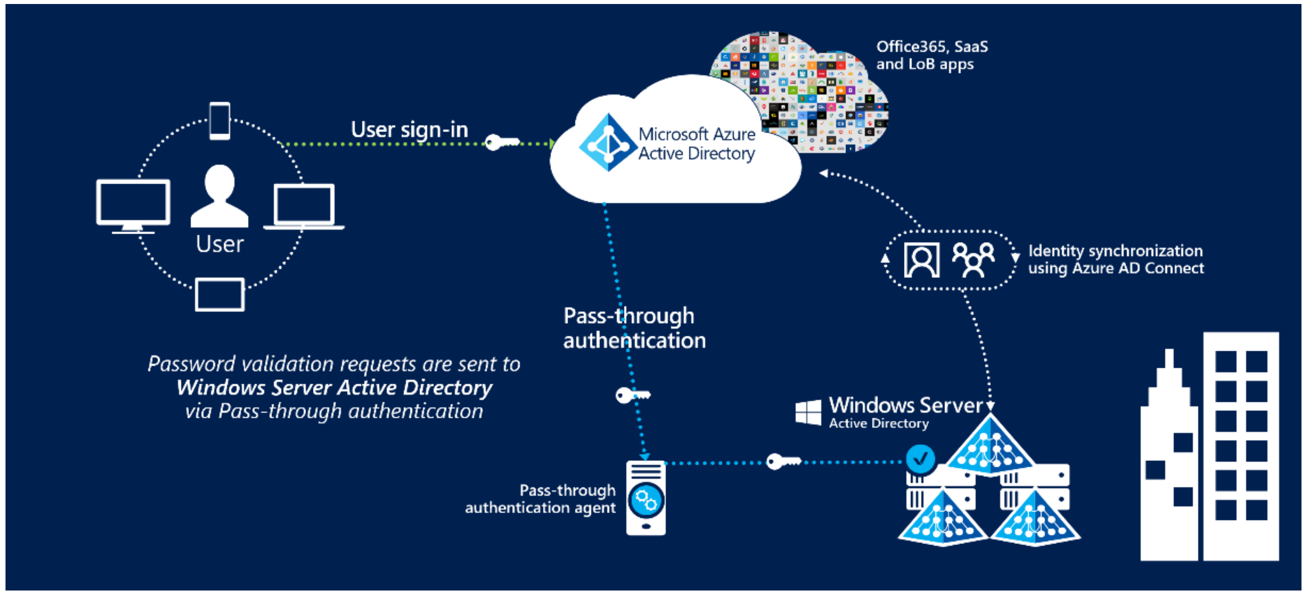
- Simple Auth
- Access to Azure / O365 ressources
  - Needs additional permissions

Connect Active Directory with Azure AD

Most companies have their accounts in their local AD

Three choices

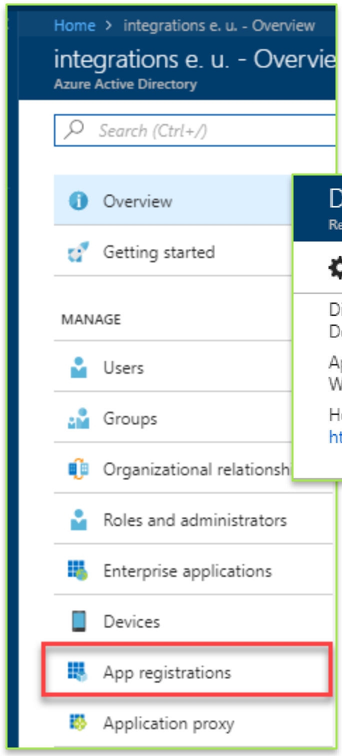
- ADFS
- Synchronized identity
- Azure AD Pass Through Auth


Office365, SaaS and LoB apps
User sign-in
Microsoft Azure Active Directory
User
Identity synchronization using Azure AD Connect
Pass-through authentication
Password validation requests are sent to *Windows Server Active Directory* via Pass-through authentication
Windows Server Active Directory
Pass-through authentication agent

App Registration

Enables Authentication

Provides additional permissions that can be granted (O365, Graph, ...)



Home > integrations e. u. - Overview

integrations e. u. - Overvie

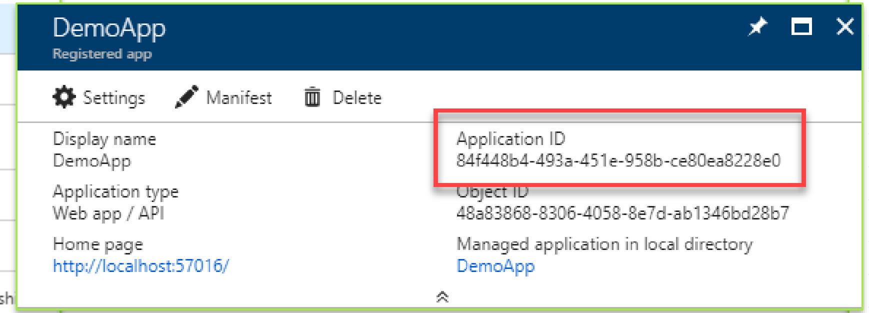
Azure Active Directory

Search (Ctrl+/)

- Overview
- Getting started

MANAGE

- Users
- Groups
- Organizational relationsh
- Roles and administrators
- Enterprise applications
- Devices
- App registrations
- Application proxy

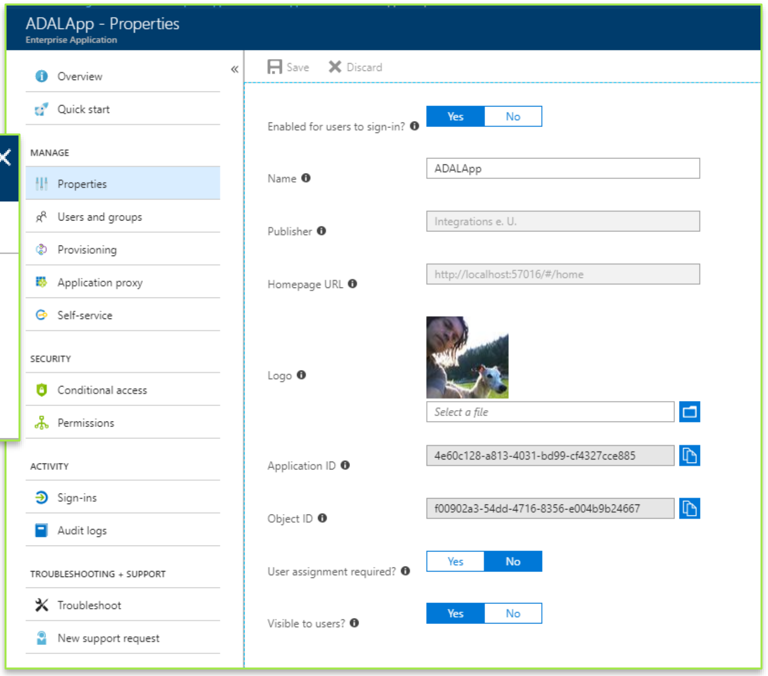


DemoApp

Registered app

Settings | Manifest | Delete

Display name DemoApp	Application ID 84f448b4-493a-451e-958b-ce80ea8228e0
Application type Web app / API	Object ID 48a83868-8306-4058-8e7d-ab1346bd28b7
Home page http://localhost:57016/	Managed application in local directory DemoApp



ADALApp - Properties

Enterprise Application

- Overview
- Quick start

MANAGE

- Properties
- Users and groups
- Provisioning
- Application proxy
- Self-service

SECURITY

- Conditional access
- Permissions

ACTIVITY

- Sign-ins
- Audit logs

TROUBLESHOOTING + SUPPORT

- Troubleshoot
- New support request

Save | Discard

Enabled for users to sign-in?	[✓ Yes] No
Name	ADALApp
Publisher	Integrations e. U.
Homepage URL	http://localhost:57016/#/home
Logo	Select a file
Application ID	4e60c128-a813-4031-bd99-cf4327cce885
Object ID	f00902a3-54dd-4716-8356-e004b9b24667
User assignment required?	Yes [✓ No]
Visible to users?	[✓ Yes] No

Microsoft Authentication Lib - MSAL

100% OpenID Connect compliant

- npm package: @azure/msal-angular
- https://github.com/AzureAD/microsoft-authentication-library-for-js

Native Angular Wrapper available

- https://github.com/AzureAD/microsoft-authentication-library-for-js/blob/dev/lib/msal-angular/README.md

Active Directory Auth Lib - ADAL

With Angular / Ionic use: adal-angular5

- adal-angular5 is an Angular wrapper around adal.js

Where to use it:

- JavaScript Applications accessing Office 365 Ressources (or not)!!!
- Can be used for Singel Page Applications (SPA) Authentication

Move to MSAL - why:

- ADAL does not support Social Logins
- MSAL Token flow 100% OAuth compatible

DEPRECATED

Securing .NET Core

Authorization

Authorization Types

- Simple Authorization – using Authorize attribute
- Claim / Token based Authorization

Ressources to Protect

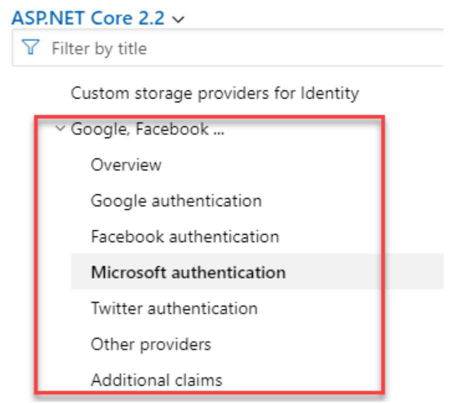
- Angular App
- Web Api [Controller]

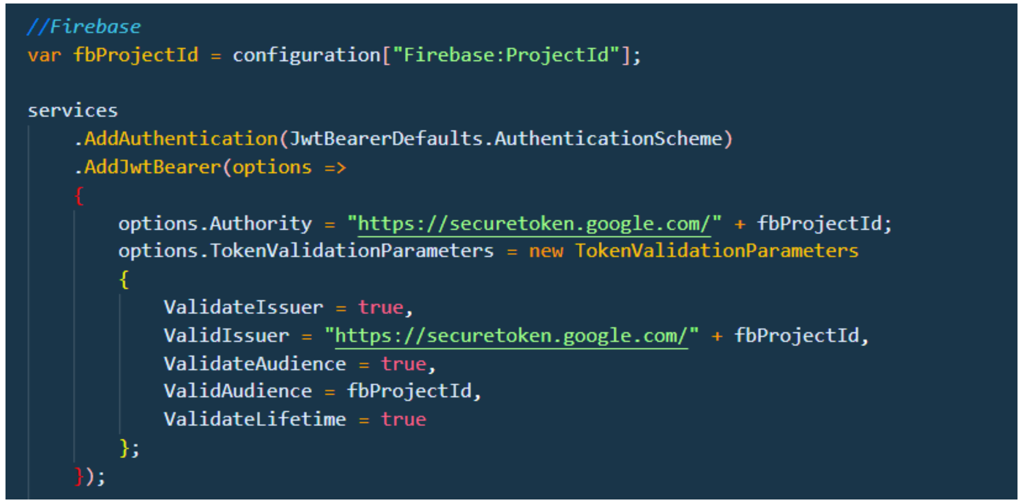
Auth Provider

Auth Provider has to be registered in Startup.cs

Config Snippets availabe @ docs.microsoft.com

- Link in readme.md





```
//Firebase
var fbProjectId = configuration["Firebase:ProjectId"];

services
    .AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.Authority = "https://securetoken.google.com/" + fbProjectId;
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidIssuer = "https://securetoken.google.com/" + fbProjectId,
            ValidateAudience = true,
            ValidAudience = fbProjectId,
            ValidateLifetime = true
        };
    });
```

Authorize Attribute

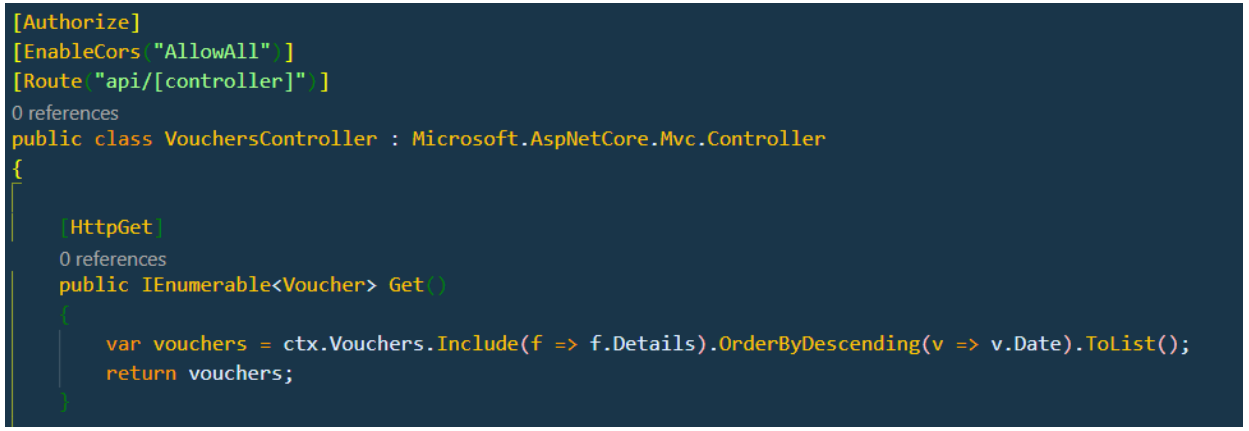
The [Autorize]-Attribute is used to require autorization for controlers

Implemented in Microsoft.AspNet.Authorization

Can be configured globally in Startup.cs -> Configure Services

Can be set on

- an individual method
- a class



```
[Authorize]
[EnableCors("AllowAll")]
[Route("api/[controller]")]
0 references
public class VouchersController : Microsoft.AspNetCore.Mvc.Controller
{

    [HttpGet]
    0 references
    public IEnumerable<Voucher> Get()
    {
        var vouchers = ctx.Vouchers.Include(f => f.Details).OrderByDescending(v => v.Date).ToList();
        return vouchers;
    }
```

Using Social Authentication

Social Login

Provides Authentication using external (social) Services like

- Facebook,
- Google,
- …

Why Social Login?

- Users don‘t need to register „another“ account for external Services

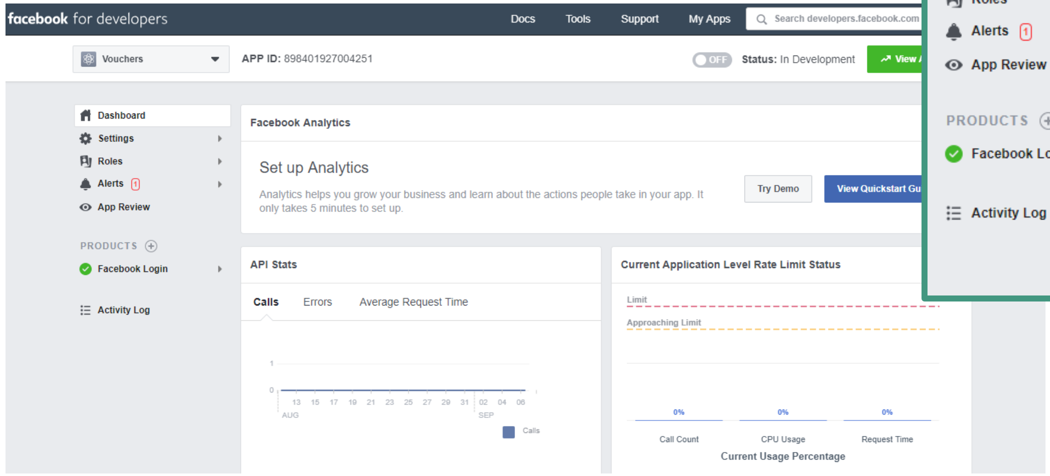
How:

- Use Wrapper-Libs instead of implementing everything on your ows – less error-prone
- i.e. angular5-social-login @ https://www.npmjs.com/package/angular5-social-login

Facebook

Register as a developer at: https://developers.facebook.com/

Get AppID & Secret here



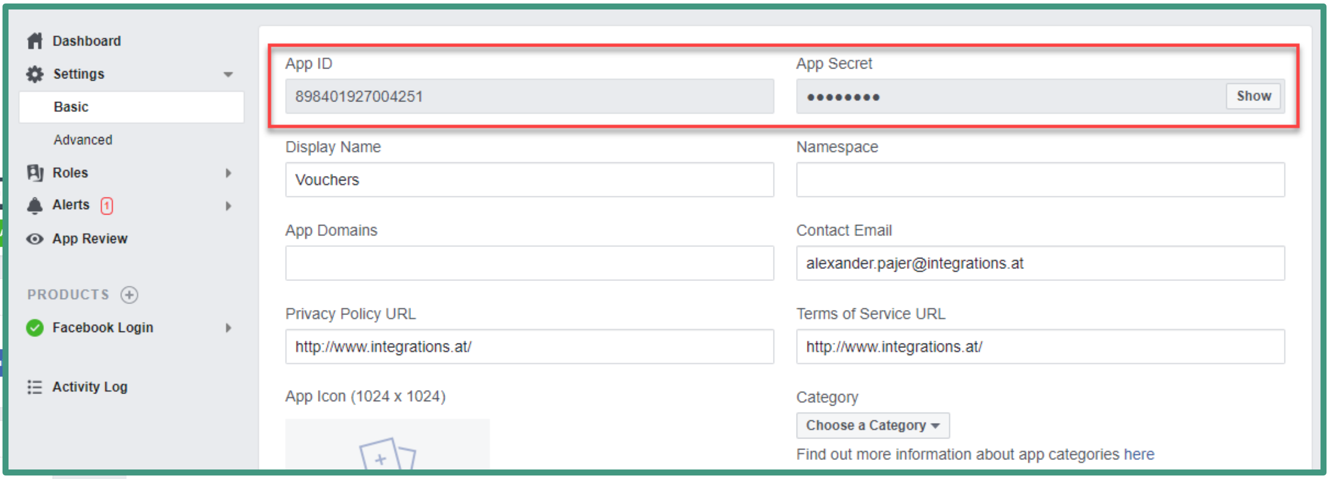
APP ID: 898401927004251

Status: In Development

Facebook Analytics

Set up Analytics

Analytics helps you grow your business and learn about the actions people take in your app. It only takes 5 minutes to set up.



App ID: 898401927004251

App Secret: ••••••••

Display Name: Vouchers

Namespace:

App Domains:

Contact Email: alexander.pajer@integrations.at

Privacy Policy URL: http://www.integrations.at/

Terms of Service URL: http://www.integrations.at/

App Icon (1024 x 1024)

Category

Find out more information about app categories here

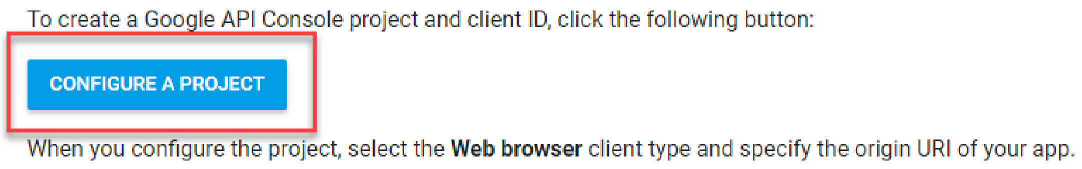
Google

Register as Google Developer @ https://developers.google.com

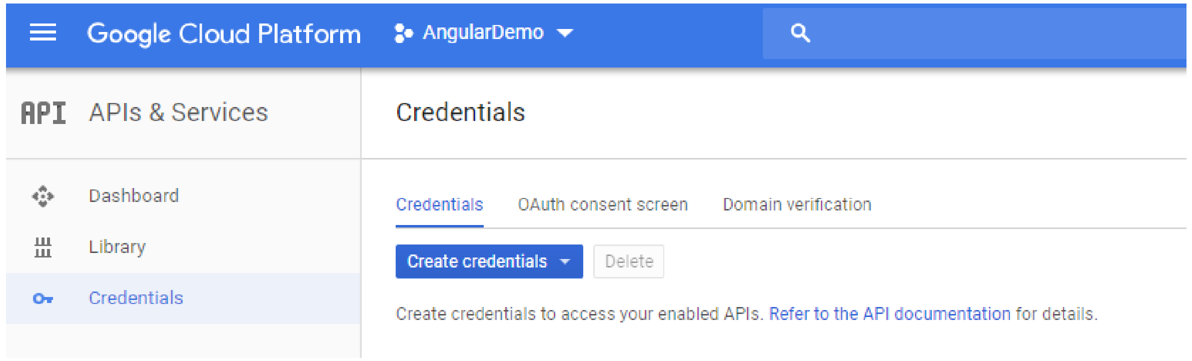
Create project / enable Sign-In for a project

Manage ongoing projects & Get Key:

- https://console.cloud.google.com/
- Enable Auth here using APIs & Services


To create a Google API Console project and client ID, click the following button:

When you configure the project, select the **Web browser** client type and specify the origin URI of your app.



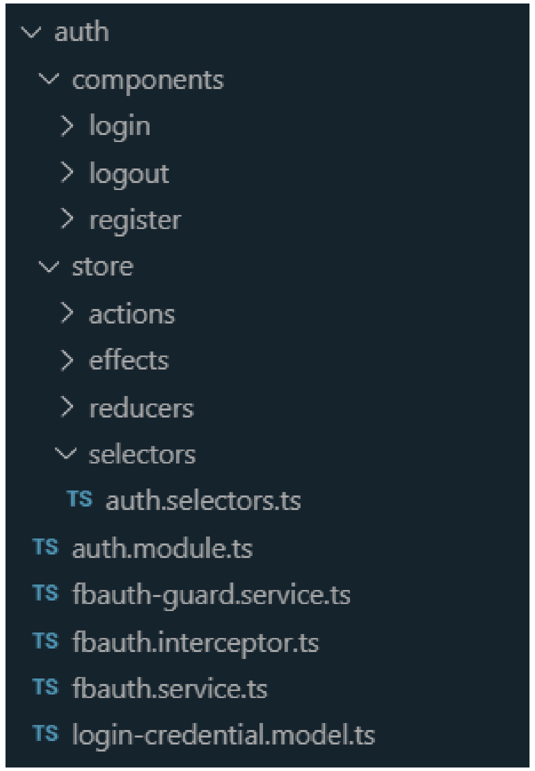
Create credentials to access your enabled APIs. Refer to the API documentation for details.

Angular Security

Building Blocks

Angular offers three major building block to enable Auth

- Statefull Service / NgRx Store
  - Stores: Token, User, ...
  - Selectors can use Parama taken from NgRx RouterState
- Route Guard
  - Allows Access to Routes
  - Params & State can be consumed by using RouterState
- Interceptor
  - Adds Tokens to the Headers of the request
  - Takes Token out of RouterState

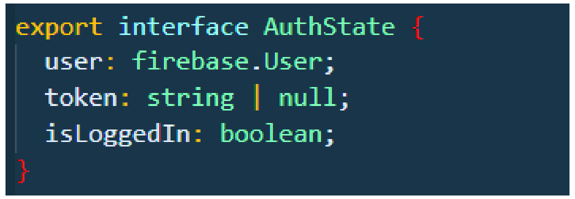


```
auth
  components
    login
    logout
    register
  store
    actions
    effects
    reducers
    selectors
      auth.selectors.ts
  auth.module.ts
  fbauth-guard.service.ts
  fbauth.interceptor.ts
  fbauth.service.ts
  login-credential.model.ts
```

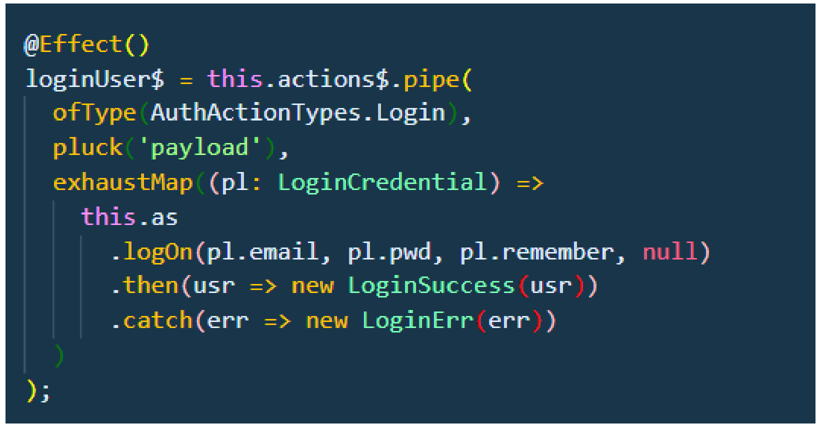
NgRx based Authentication

Uses:

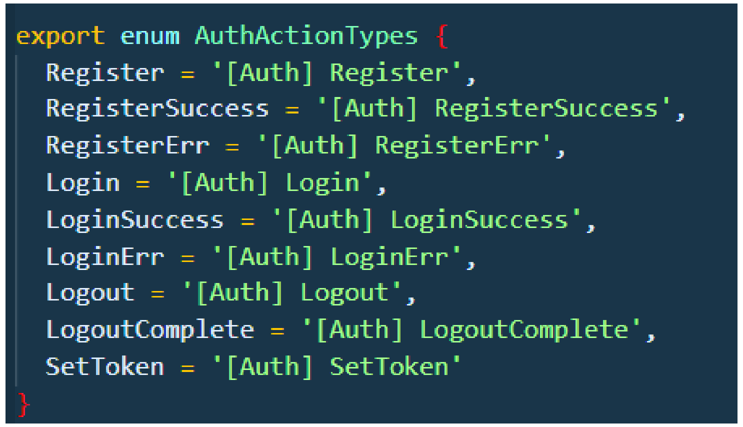
- Service for the Identity Provider of your choice (Azure AD, Google, ...)
- AuthState
- Actions, Reducers, Effects, Selectors
- Register, Login, Logout



```
export interface AuthState {
  user: firebase.User;
  token: string | null;
  isLoggedIn: boolean;
}
```



```
@Effect()
loginUser$ = this.actions$.pipe(
  ofType(AuthActionTypes.Login),
  pluck('payload'),
  exhaustMap((pl: LoginCredential) =>
    this.as
      .logOn(pl.email, pl.pwd, pl.remember, null)
      .then(usr => new LoginSuccess(usr))
      .catch(err => new LoginErr(err))
  )
);
```



```
export enum AuthActionTypes {
  Register = '[Auth] Register',
  RegisterSuccess = '[Auth] RegisterSuccess',
  RegisterErr = '[Auth] RegisterErr',
  Login = '[Auth] Login',
  LoginSuccess = '[Auth] LoginSuccess',
  LoginErr = '[Auth] LoginErr',
  Logout = '[Auth] Logout',
  LogoutComplete = '[Auth] LogoutComplete',
  SetToken = '[Auth] SetToken'
}
```

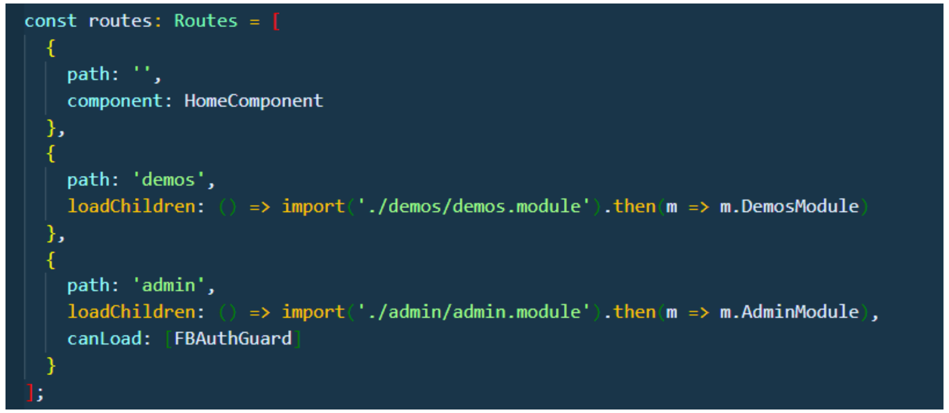
NgRx & Route Guard

Route Guards allow or deny route execution before a Routing event takes place

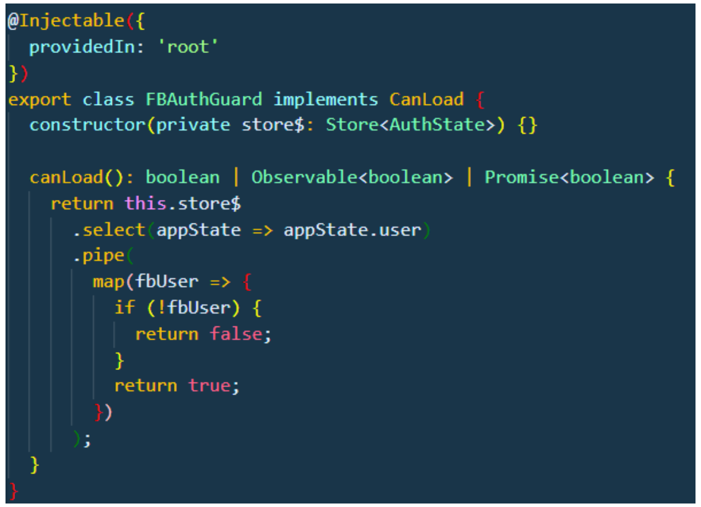
Easy to implement in custom class

Implement one of the following Interfaces:

- CanActivate
- CanActivateChild
- CanDeactivate
- CanLoad



```
const routes: Routes = [
  {
    path: '',
    component: HomeComponent
  },
  {
    path: 'demos',
    loadChildren: () => import('./demos/demos.module').then(m => m.DemosModule)
  },
  {
    path: 'admin',
    loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule),
    canLoad: [FBAuthGuard]
  }
];
```

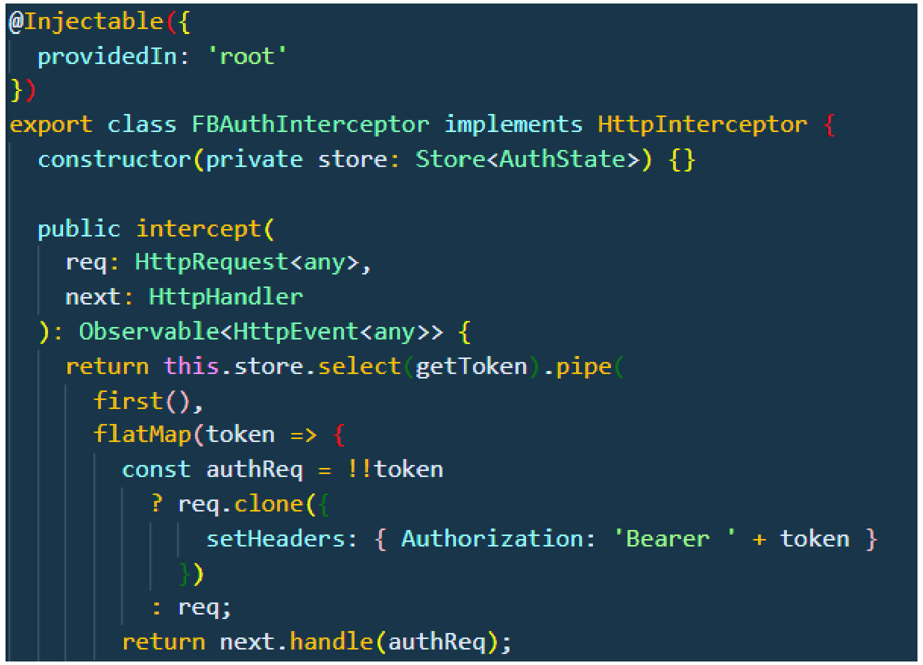


```
@Injectable({
  providedIn: 'root'
})
export class FBAuthGuard implements CanLoad {
  constructor(private store$: Store<AuthState>) {}

  canLoad(): boolean | Observable<boolean> | Promise<boolean> {
    return this.store$
      .select(appState => appState.user)
      .pipe(
        map(fbUser => {
          if (!fbUser) {
            return false;
          }
          return true;
        })
      );
  }
}
```

NgRx & Interceptor

Interceptors are used to "modify" Request befor it is sent: ie Add Token

Token is taken out of NgRx Store



```
@Injectable({
  providedIn: 'root'
})
export class FBAuthInterceptor implements HttpInterceptor {
  constructor(private store: Store<AuthState>) {}

  public intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    return this.store.select(getToken).pipe(
      first(),
      flatMap(token => {
        const authReq = !!token
          ? req.clone({
              setHeaders: { Authorization: 'Bearer ' + token }
            })
          : req;
        return next.handle(authReq);
```