

WEA5 Übung Angular: Minimales Keycloak Setup WS 2024

Anstatt den in der Cloud gehosteten IdentityServer von Manfred Steyrer zu verwenden kann man beispielsweise das Open Source Identity and Access Management System [Keycloak](#) verwenden.

Mit Docker

Quelle: <https://www.keycloak.org/getting-started/getting-started-docker>

```
docker run -p 8080:8080 -e KEYCLOAK_ADMIN=wea5admin -e  
KEYCLOAK_ADMIN_PASSWORD=wae5adminPass quay.io/keycloak/keycloak:26.0.5  
start-dev
```

Konfigurierten Docker Container speichern (optional):

```
docker commit <container_id> (erhält man mit docker ps -a)  
docker images -> IMAGE ID kopieren  
docker tag <IMAGE ID> wae5/keycloak  
oder  
docker commit <container_id> wae5/keycloak
```

Container wieder starten:

```
docker run -it -p 8080:8080 <neuer-name>
```

Später bei Bedarf den Container danach noch benennen:

```
docker rename <container_id> wae5-keycloak-2024
```



Server Initialization (Admin-Account festlegen)

Am Anfang müssen wir einen Admin-Account festlegen. Die aktuelle Doku findet man [hier](https://www.keycloak.org/docs/latest/server_admin/index.html) (https://www.keycloak.org/docs/latest/server_admin/index.html).

- In der lokalen Installation muss nur <http://localhost:8080/admin> aufgerufen und zuerst der Admin Benutzer konfiguriert werden.
- Wir geben den Initial Admin ein:
 - **Username:** **wae5admin**
 - **Password:** **wae5adminPass**

Server Administration (Realm/Bereich, Benutzer und Apps festlegen)

Nun legen wir mindestens einen Benutzer, einen Realm/Bereich und unsere SPA an. Die aktuelle Doku findet man [hier](#).

- Mit Browser verbinden wir uns zu <http://localhost:8080/admin>
- Einloggen mit dem **Admin-User** von oben:
 - Username: **wea5admin**
 - Password: **wea5adminPass**
- Neuen Realm/Bereich "wea5" anlegen
 - Name: wea5
 - Display name: wea5 Auth Demo
 - HTML Display name: wea5 Auth Demo
- Neuen User anlegen für Realm **wea5** (**Achtung: jeder Realm hat seine eigenen User!!:**)
 - User: **wea5user**
- Credentials für User "wea5user" festlegen
 - Username: **wea5user** (Speichern, dann zu Reiter „Credentials“ wechseln)
 - Password: **wea5userPass**
 - Das "Temporary" Flag auf "off" setzen, damit wir nicht beim Login ein neues Passwort einstellen müssen
- Auf der Details-Seite vom Benutzer "wea5user"
 - Required User Actions: sollte leer sein
 - Fertig
- Clients (= unsere App) hinzufügen
 - Client ID: wea5-demo
 - Name: WEA5 Angular Demo mit OAuth
 - Client Protocol: openid-connect
 - Dann
 - "Implicit Flow" aktivieren (on/off Schalter)
 - Root URL: <http://localhost:4200>
 - Valid Redirect Request: http://localhost:4200/*

Hinweis: man kann auch Clients und Realms exportieren und diese im neuen Server importieren (sofern man schon einen bestehenden Server hat), aber das würde zu weit führen 😊

Angular-Anwendung konfigurieren

Eine wesentliche Rolle bei der Konfiguration der Angular-Anwendung kommt die Einstellung der **Endpoints** zu, d.h. wie meine App mit dem Identity Management System kommunizieren muss. Die Dokumentation zu Keycloak findet man hier: <https://www.keycloak.org/securing-apps/oidc-layers>

- Im Verzeichnis app des Angular-Projekts legt man sich eine auth.config.ts- Datei an, die folgenden Inhalt hat:

```
import { AuthConfig } from 'angular-oauth2-oidc';

export const authConfig: AuthConfig = {
  issuer: 'http://localhost:8080/realm/wea5',
  loginUrl: 'http://localhost:8080/realm/wea5/protocol/openid-connect/auth',
  logoutUrl: 'http://localhost:8080/realm/wea5/protocol/openid-connect/logout',
  tokenEndpoint: 'http://localhost:8080/realm/wea5/protocol/openid-
connect/token',
  sessionCheckIFrameUrl: 'http://localhost:8080/realm/wea5/protocol/openid-
connect/login-status-iframe.html',
  userinfoEndpoint: 'http://localhost:8080/realm/wea5/protocol/openid-
connect/userinfo',
  clientId: 'wea5-demo',
  redirectUri: window.location.origin + '/index.html',
  silentRefreshRedirectUri: window.location.origin + '/silent-refresh.html',
  scope: 'profile email',
  silentRefreshTimeout: 5000, // For faster testing
  timeoutFactor: 0.25, // For faster testing
  sessionChecksEnabled: true,
  showDebugInformation: true, // Also requires enabling "Verbose" level in
devtools
  clearHashAfterLogin: false, // https://github.com/manfredsteyer/angular-
oauth2-oidc/issues/457#issuecomment-431807040
};
```

- Die Endpunkte kann man wie folgt über einen Browser abfragen:
<http://localhost:8080/realm/wea5/.well-known/openid-configuration>
- Das schaut dann ungefähr so aus, wenn man die dann entsprechend formatiert:

```
http://localhost:8080/realm/wea5/.well-known/openid-configuration
{
  "issuer": "http://localhost:8080/realm/wea5",
  "authorization_endpoint": "http://localhost:8080/realm/wea5/protocol/openid-connect/auth",
  "token_endpoint": "http://localhost:8080/realm/wea5/protocol/openid-connect/token",
  "introspection_endpoint": "http://localhost:8080/realm/wea5/protocol/openid-
connect/token/introspect",
  "userinfo_endpoint": "http://localhost:8080/realm/wea5/protocol/openid-connect/userinfo",
  "end_session_endpoint": "http://localhost:8080/realm/wea5/protocol/openid-connect/logout",
```

```

    "frontchannel_logout_session_supported": true,
    "frontchannel_logout_supported": true,
    "jwks_uri": "http://localhost:8080/realm/wea5/protocol/openid-connect/certs",
    "check_session_iframe": "http://localhost:8080/realm/wea5/protocol/openid-connect/login-
status-iframe.html",
    "grant_types_supported": [
        "authorization_code",
        "implicit",
        "refresh_token",
        "password",
        "client_credentials",
        "urn:ietf:params:oauth:grant-type:device_code",
        "urn:openid:params:grant-type:ciba"
    ],
],

```

- `app.module.ts` braucht folgende Erweiterung:

```

import { provideOAuthClient } from 'angular-oauth2-oidc';

export const appConfig: ApplicationConfig = {
    providers: [provideZoneChangeDetection({ eventCoalescing: true }),
        provideHttpClient(withInterceptorsFromDi()),
        provideRouter(routes),
        provideOAuthClient()]
};

```

- `app.component.ts` braucht folgende Erweiterung:

```

import { authConfig } from './auth.config';
import { OAuthService } from 'angular-oauth2-oidc';

constructor(private oauthService: OAuthService) {
    this.configureWithNewConfigApi();
}

private configureWithNewConfigApi() {
    this.oauthService.configure(authConfig);
    this.oauthService.loadDiscoveryDocumentAndTryLogin();
}

```

- Und unser `authentication.service.ts`, welches letztendlich dann den gewünschten Flow umsetzt, könnte ungefähr so aussehen:

```

import { Injectable } from '@angular/core';
import { OAuthService } from 'angular-oauth2-oidc';

@Injectable({
    providedIn: 'root'
})

```

```
export class AuthenticationService {

  constructor(private oauthService: OAuthService) { }

  login(username: string, password: string): boolean {
    this.oauthService.initCodeFlow(); // is recommended nowadays
    return true;
  }

  isLoggedIn() {
    return this.oauthService.hasValidAccessToken() &&
      this.oauthService.hasValidIdToken();
  }
}
```