

PROTECT

POW!

PROTECT THE

BATCOMPUTER!

BOW!

BAM!

POW!

BAM!

BAM!

OAUTH2:

JWT

SOCK!

SOCK!

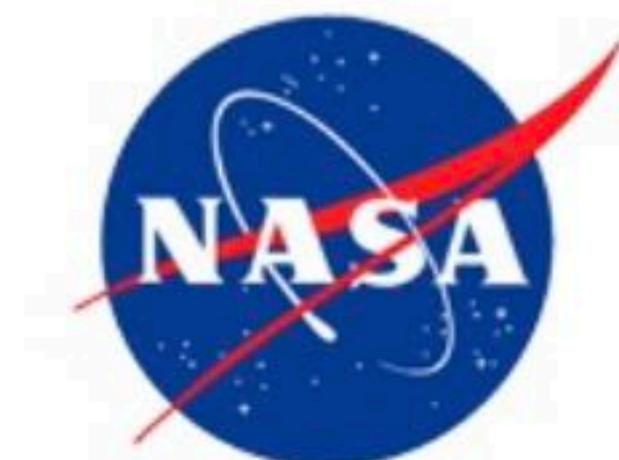
Kelly Morrison, Ph.D. Computer Engineering



The University of Georgia®



THE CLIMATE
CORPORATION



Marshall Space
Flight Center



Edward Jones
MAKING SENSE OF INVESTING



**Problem: How do
we add security
and authorization
to our apps?**



ChatGPT can generate the security code you need, if you just ask it.

But I don't know what to ask it!



**Today, we'll go over the concepts
so you can tell your generative AI
what to generate for you.**



One of the most commonly used authorization methods today is OAuth 2. It's used for web and desktop apps, mobile devices, and “living room devices”.

OAuth stands for “Open Authorization”.

You can read about the protocol at oath.net



Waitaminnit... what's “authorization”?

Authorization means determining what you can access, and what you have permission to do.



Yes.

So, like determining whether I can go in the vault at Gotham First National Bank?





Does it also verify who I am?

No, that's authentication, which means verifying your true identity.



Yes.

Like my secret identity?

I don't like that.



Authentication verifies that you're
Harleen Quinzel.

Authorization says that you can
access your bank account at an
ATM, but you can't walk into the
bank vault.



But your birth certificate
authenticates you as
Harleen Quinzel.

Hey! I'm Harley Quinn,
not that Quinzy
person!



OAuth2 is for authorization.

But there's another protocol called OpenID Connect (OIDC) for authentication.

OIDC is built on top of OAuth2.



OAuth2 defines several flows.

- **Authorization code.** Web apps where you log in.
- **Client credentials.** For services talking to other services.
- **Resource owner password flow.** For apps where redirects cannot be used.
- **Authorization code flow with PKCE (Proof of Key Code Exchange, pronounced “pixy”).** For native mobile apps and single page apps.
- **Device authorization flow.** For mobile devices.
- There is also ~~implicit flow~~, but it is deprecated.



I'm getting bored.

And you still haven't shown me how
to add security to my app.



**Fair enough. Let's take a look at
how OAuth2 works.**

**First, we'll look at it without any
code, so you can get a feel for
how it works.**



**Meet Batman.
He has a Bat-Computer.
He is very protective of it.**

**Can I use the
Bat-Computer?**



**No. I don't know
who you are!**



Do you know Wonder Woman? She'll vouch for me.



*Hmm... I DO know Wonder Woman. If she says **this caped weirdo** is trustworthy, then I suppose I do, too.*



Sure. Tell Wonder Woman to give me some proof that you're trustworthy, and I'll let you use the Bat-Computer.



**Hi, Wonder Woman!
Can you tell Batman
that he can trust me?**

**Thanks, Wonder
Woman!**

**Sure. Here's a card that
says I trust you. And I put a
secret word on it so that
he'll know it's from me:
“Martha”. Give it to him.**



Here's a card that Wonder Woman gave me that says you can trust me.



*Hmm... the card says
“Dear Batman. I vouch for
Superman. The secret word
is “Martha”.*

*This looks authentic, but
give me a moment to verify
it with Wonder Woman...*



Hi, Wonder Woman. Superman gave me this card that says you vouch for him, and that the secret word is “Martha”.



That's right. You can trust him.



OK. Wonder Woman says you're all right, so you can use the Bat-Computer. Here's a ticket to get into the Bat-Computer room.



Hooray!





You've just seen how the OAuth2 authorization code flow works.

Let's go through it again, but with OAuth2 terms added.

Client



Authorization Server



Resource Server



Resource



**Can I use the
Bat-Computer?**



Client
(Web app)

**No. I don't know
who you are!**



Resource Server
(Service)



Resource
(An API)

Do you know Wonder Woman? She'll vouch for me.



Client



Authorization Server

*Hmm... I DO know Wonder Woman. If she says **this caped weirdo** is trustworthy, then I suppose I do, too.*



Resource Server

Sure. Tell Wonder Woman to give me some proof that you're trustworthy, and I'll let you use the Bat-Computer.



Client

Hi, Wonder Woman!
Can you tell Batman
that he can trust me?

Sure. Here's a card that
says I trust you. And I put a
secret word on it so that
he'll know it's from me:
“Martha”. Give it to him.



Authorization Code

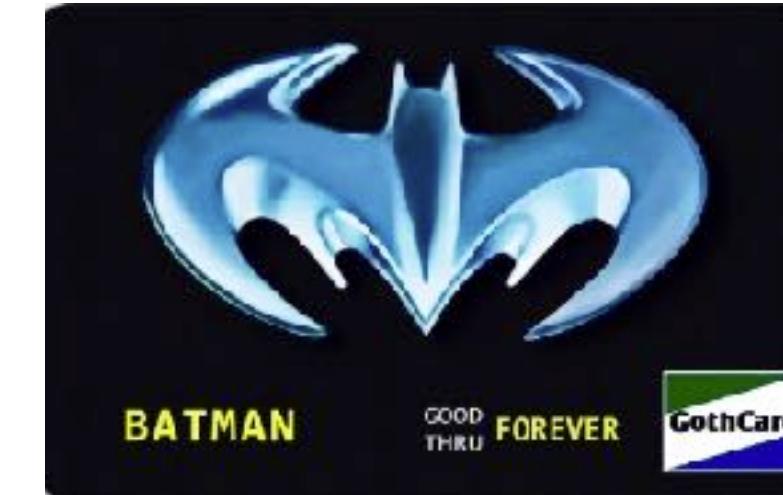


Authorization Server

Here's a card that Wonder Woman gave me that says you can trust me.



Client



Authorization Code



*Hmm... the card says
“Dear Batman. I vouch for
Superman. The secret word
is “Martha”.*

Resource Server

This looks authentic, but give me a moment to verify it with Wonder Woman...



Resource
Server



Authorization Code

Hi, Wonder Woman. Superman gave me this card that says you vouch for him, and that the secret word is “Martha”.

That's right. You can trust him.



Authorization Server

OK. Wonder Woman says you're all right, so you can use the Bat-Computer. Here's a ticket to let Alfred know who you are, and a ticket to get in the Bat-Computer Room.



ID Token



Access Token



Client

Hooray!



Resource
Server



Resource
Server



Client



ID Token



Access Token



Resource





Let's talk about that ID token and
that access token.

A stylized illustration of Alfred Pennyworth, the butler to Bruce Wayne. He is shown from the chest up, wearing his signature brown suit, white shirt, and blue bow tie. He has a serious, slightly weary expression. His hair is dark and receding. The background behind him is plain white.

The ID token says that the user has been authenticated. It's for determining identity within a client. E.g., “Alfred, this is Superman.”

The access token says what access has been authorized. It's for use by an API to determine what can be accessed. E.g., “Alfred, access to the Bat-Computer room has been granted.”

They are typically JWTs (Json Web Tokens).

The smug cat is right.



JAY DOUBLE YOU TEE!



JOT

Here's an example ID token.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRw0i8vbXktZG9tYWluLmF1dGgwLmNvbSIsInN1YiI6ImF1dGgwfDEyMzQ1NiIsImF1ZCI6IjEyMzRhYmNkZWYiLCJleHAiOjEzMTEyODE5NzAsImlhCI6MTMxMTI4MDk3MCwibmFtZSI6IkphbmUgRG9lIiwiZ2l2ZW5fbmFtZSI6IkphbmUiLCJmYW1pbHlfbmFtZSI6IkRvZSJ9.bql-jxlG9B_bielkq0njTY9Di9FillFb6IMQINXoYsw

And here's an example access token.

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiawF0IjoxNTE2MjM5MDIyfQ.SfLKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c

They are Base64 strings. You can decode them at jwt.io.



JSON WEB TOKEN (JWT)

COPY

CLEAR

Valid JWT

Signature Verified

```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJMeVVPTDg4LVBGM3BYQzF
pN3BIeGdFZTJwaWZJY3RyTXJiNklHOElmRTlVIn0.eyJleHAiOjE3NDEyNTE40TU
sImhdCI6MTc0MTI1MTU5NSwiYXV0aF90aW1lIjoxNzQxMjUxNTk1LCJqdGkiOiJmNzZhODVlZC0wNm
QzLTQ2NzUtODk3My1iINDE0NTNhNGEyMjIiLCJpc3MiOiJodHRw0i8vbG9jYWxob3N00jkx
0TEvcnVhbG1zL2RlbW8tcmVhbG0iLCJhdWQiOjItZXNzYWdlcy13ZWJhcHAiLCJzdWI0iI
xM2M00DlmMS1jMWJlLTrjNmMtOTJlZS1kNTZhYzEzNjE3YTgiLCJ0eXAiOiJJRCIsImF6cCI
6Im1lc3NhZ2VzLXd1YmFwcCIsIm5vbmlIjoi0E1WZEszNVpqcHvlnW9RNmhIX2haVGplM
zBwaG9aN1dzZGFIYTJxdkh3USIsInNlc3Npb25fc3RhdGUi0iI30WM1MjdhMi1hMTEXLTQz
NzktODYxYy1lOTMzNjU0ZGQ1NzciLCJhdF9oYXNoIjoiZHBfWFh3Um5RWnQwcXhtlW9qNH
vZyIsImFjciI6IjEiLCJzaWQiOii30WM1MjdhMi1hMTEXLTQzNzktODYxYy1lOTMzNjU0ZG
Q1NzciLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpBI
m9mZmxpbmVfYWNjZXNzIiwidW1hX2F1dGhvcmI6YXRpb24iLCJkZWZhdWx0LXJvbGVzLWRl
bW8iXX0sIm5hbWUi0iJCcnVjZSBXYXluZSIisInByZwZlcnJlZF91c2VybmFtZSI6ImJ3YXl
uZSIisImdpdmVuX25hbWUi0iJCcnVjZSIisImZhbWlseV9uYW1lIjoiV2F5bmUiLCJlbWFpbCI
6ImJ3YXluZUBnb3RoYW1tYWlsLmNvbSJ9.QaSsLYHFCFeRxTD_c19o36Q0kDFnJQgnrAQvm
6N_kSBRRP8X-OsKvgbYNJ_-umldh-
vTiF0wtrHC91lK7oRTSrL8MGz5bNtRc802a_xaRM4XBss-
SdiG07Lk3wk2iKNN6vvWXFD60XMC-LH3m0JnZuPkghLSQHkyq9iivhSZgiifMz3ig-
Ay0c4VpWRiFLFolHRV0Is1t0UlJTcUJXFEEmlZoiJEEbMbxI1fls-
N3Zd101gvXe4NsxYjazAgUKq0MKKLrn0qqqRlbnxRnog-SBlrrCKK37e4aUgJHpYKrT5e-
T5gdFCIVMjgqLnwkDHT0JSVlvzA4fgfNB953GEZQ
```

JSON CLAIMS TABLE

COPY

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "LyUOL88-PF3pXC1i7pHxgEe2pifIctrMrb6IG8IfE9U"
}
```

DECODED PAYLOAD

JSON CLAIMS TABLE

COPY

```
{
  "exp": 1741251895,
  "iat": 1741251595,
  "auth_time": 1741251595,
  "jti": "f76a85ed-06d3-4675-8973-b41453a4a222",
  "iss": "http://localhost:9191/realms/demo-realm",
  "aud": "messages-webapp",
  "sub": "13c489f1-c1be-4c6c-92ee-d56ac13617a8",
  "typ": "ID",
  "azp": "messages-webapp",
  "nonce": "8MVdK35Zjpue5oQ6hH_hZTje30phoZ7WsdaHa2qvHwQ",
  "session_state": "79c527a2-a111-4379-861c-e933654dd577",
  "at_hash": "dp_XXwRnQZt0qxmYoj4sog",
  "acr": "1",
  "sid": "79c527a2-a111-4379-861c-e933654dd577",
  "email_verified": true,
  "realm_access": {
    "roles": [
      "offline_access",
      "uma_authorization",
      "default-roles-demo"
    ]
  },
  "name": "Bruce Wayne",
  "preferred_username": "bwayne",
  "given_name": "Bruce",
  "family_name": "Wayne",
  "email": "bwayne@gothammail.com"
}
```

ID Token

Access Token

This is the beta of the new jwt.io! [Share feedback on new UI/UX ↗](#)

 **JWT**
Debugger

TA5LTQ0MDYt0WY3YS01NTgyMzLkNWI20DMiLCJpc3Mi0iJodHRw0i8vbG9jYWxob3N00jkx0TEvcmVhbG1zL2RlbW8tcmVhbG0iLCJhdWQi0iJhY2NvdW50Iiwic3ViIjoiMTNjNDg5ZjEtYzFizZ500YzzjLTkyZWUtZDU2YWMxMzYxN2E4IiwidHlwIjoiQmVhcmVyIiwiYXpwIjoibWVzc2FnZXMt2ViYXBwIiwibm9uY2Ui0iI4TVzkSzM1WmpwdWU1b1E2aEhfaFpUamUzMHB0b1o3V3NkYUhhMnF2SHdRIiwic2Vzc2lvbl9zdGF0ZSI6Ijc5YzUyN2EyLWExMTEtNDM30S04NjFjLWU5Mz2NTRkZDU3NyIsImFjciI6Ijc5IjEiLCJhbGxvd2VkLW9yaWdpbnMi0lsiaHR0cDoVL2xvY2FsaG9zdDo4MDgwIl0sInJlYWxt2FjY2VzcyI6eyJyb2xlclyI6WYJvZmZsaW5lx2FjY2VzcyIsInVtYV9hdXR0b3JpemF0aW9uIiwizGVmYXVsdC1yb2xlcly1kZW1vIl19LCJyZXNvdXJjZV9hY2Nlc3MiOnsiYWNjb3VudCI6eyJyb2xlclyI6WYJtYW5hZ2UtYWNjb3VudCIsIm1hbmFnZS1hY2NvdW50LWxpbtzIiwidmlldy1wcm9maWxlIl19fSwic2NvcGUi0iJvcGVuaWQgZw1haWwgchJvZmlsZSIIsInNpZCI6Ijc5YzUyN2EyLWExMTEtNDM30S04NjFjLWU5MzM2NTRkZDU3NyIsImVtYWlsX3ZlcmlmaWVkJp0cnVllCJuYW1lIjoiQnJ1Y2UgV2F5bmUiLCJmCJwcmVmZXJyZWRfdXNlc5hbWUi0iJid2F5bmUiLCJnaXZlbl9uYW1lIjoiQnJ1Y2UiLCJmYw1pbH1fbmFtZSI6IldheW5lIiwizW1haWwi0iJid2F5bmVAZ290aGftbwFpbC5jb20ifQ.bsxDKrpKgYeo5lHClientEeKJUa4cryLoK4iWrtyUcSJsn0JEtIetmhpnMj-xbP0Zycr10RkFZ3BrUaiabD_L0go3F3S6XEGKjg8JxdpyMUDqhcgioVb96H5ywDQ0iBuIGV0outabw56x-PjwhAP8hJIPaKAvQ2WHAbX0iONhCjCl2oaBg7rhrefCsKSG9-F5Z_TLafohJQDoq2Teb7mtLdVf303IqdiWEf9fWS-ps8DtmC0TPkywc-ZwjY093C2q1Gshr2AEtWzvFtv7_BQ7zlXpB-3ra_h8DoItDsne3DMZE_nP51tkVn26Yl70DT830YU_kkQl1TaAT9_X-g

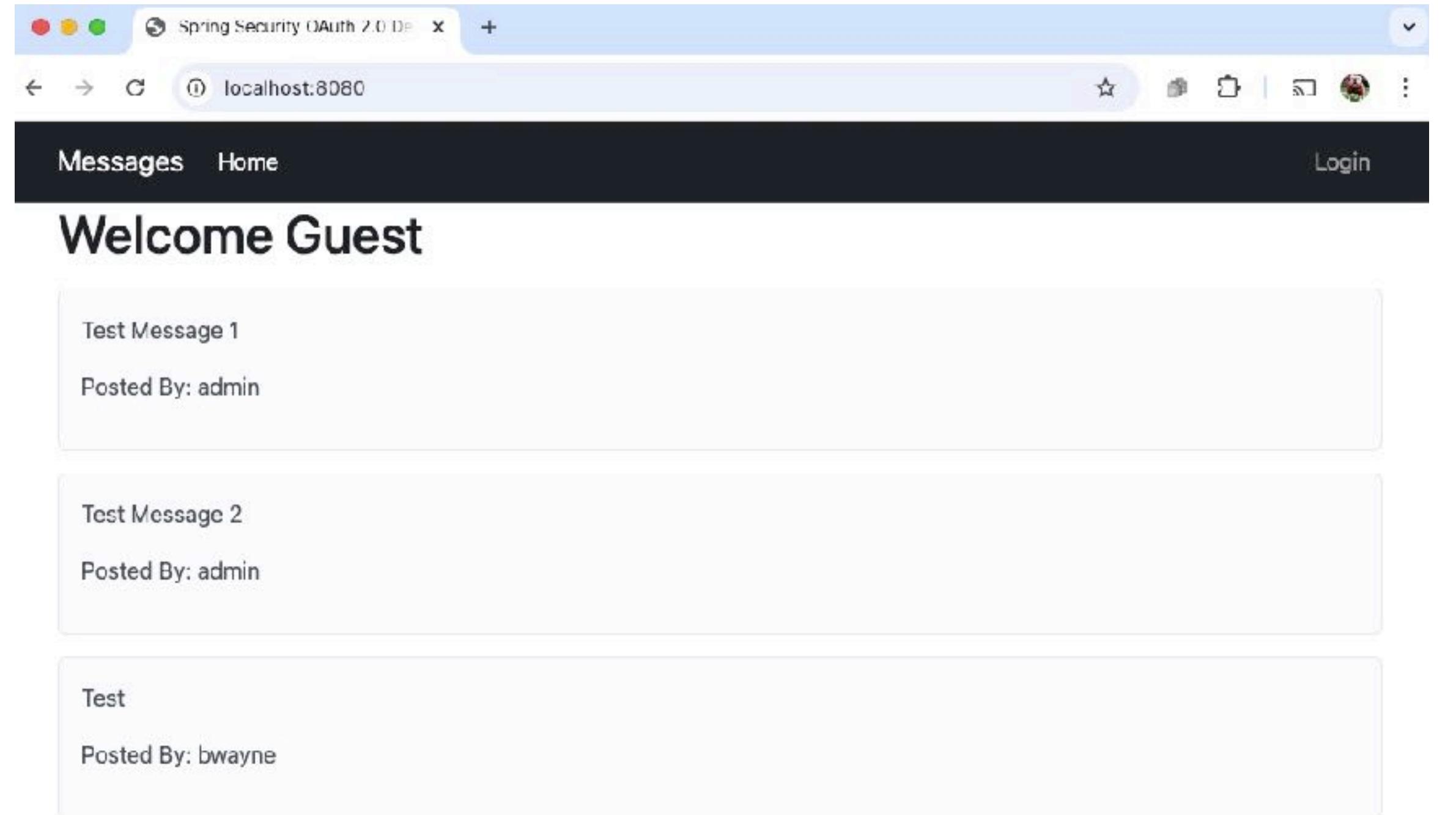
DECODED PAYLOAD

[JSON](#) [CLAIMS TABLE](#) [COPY](#)

```
{  
  "exp": 1741251895,  
  "iat": 1741251595,  
  "auth_time": 1741251595,  
  "jti": "bf1cc2c3-7e09-4406-9f7a-558239d5b683",  
  "iss": "http://localhost:9191/realms/demo-realm",  
  "aud": "account",  
  "sub": "13c489f1-c1be-4c6c-92ee-d56ac13617a8",  
  "typ": "Bearer",  
  "azp": "messages-webapp",  
  "nonce": "8MVdk35Zjpue5oQ6hH_hZTje30phoZ7WsdaHa2qvHwQ",  
  "session_state": "79c527a2-a111-4379-861c-e933654dd577",  
  "acr": "1",  
  "allowed_origins": [  
    "http://localhost:8080"  
,  
  "realm_access": {  
    "roles": [  
      "offline_access",  
      "uma_authorization",  
      "default-roles-demo"  
    ]  
,  
  "resource_access": {  
    "account": {  
      "roles": [  
        "manage-account",  
        "manage-account-links",  
        "view-profile"  
      ]  
,  
    },  
    "scope": "openid email profile",  
    "sid": "79c527a2-a111-4379-861c-e933654dd577",  
    "email_verified": true,  
    "name": "Bruce Wayne",  
    "preferred_username": "bwayne",  
    "given_name": "Bruce",  
    "family_name": "Wayne",  
    "email": "bwayne@batcave.com"  
  }  
},  
}
```



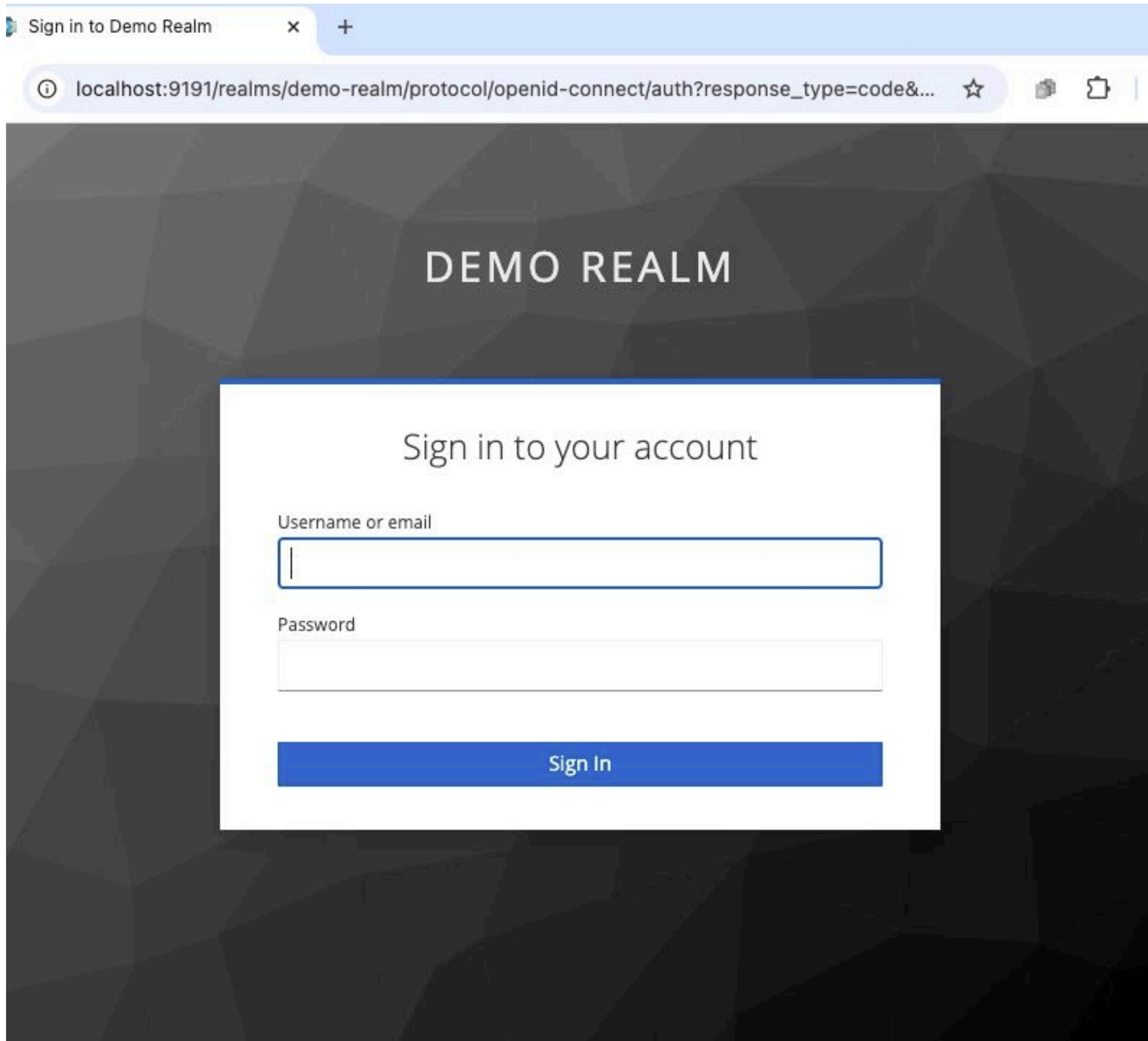
**Let's go through this one last time,
this time showing the final flow.**



Click on Login in the client.

The user is redirected to the authorization server.

A login screen is presented.



The user enters their credentials and hits “Sign In”.

The user is redirected to the application, with the authorization server supplying a one-time authorization code.

The app receives the authorization code and sends it to the authorization server.

The authorization server generates an ID token and access token and sends to the app.

The app can now use the access token to gain access to the APIs provided by the service.

A screenshot of a web browser window titled "Spring Security OAuth 2.0 De". The address bar shows "localhost:8080". The page has a dark header with "Messages" and "Home" links, and a "Logout" link on the right. Below the header, the text "Welcome Bruce Wayne" is displayed. A form for posting a message is present, with a "Message" input field and a blue "Submit" button. Three messages are listed below the form: "Test Message 1" posted by "admin", "Test Message 2" posted by "admin", and "Test" posted by "bwayne".

The app can now pass the access token to the service to be used to authorize access for various operations.



I'd love to try this out, but I don't
have an authorization server at home.



You're in luck! There's an open source
authorization server named **Keycloak**
that you can use!

Open Source Identity and Access Management

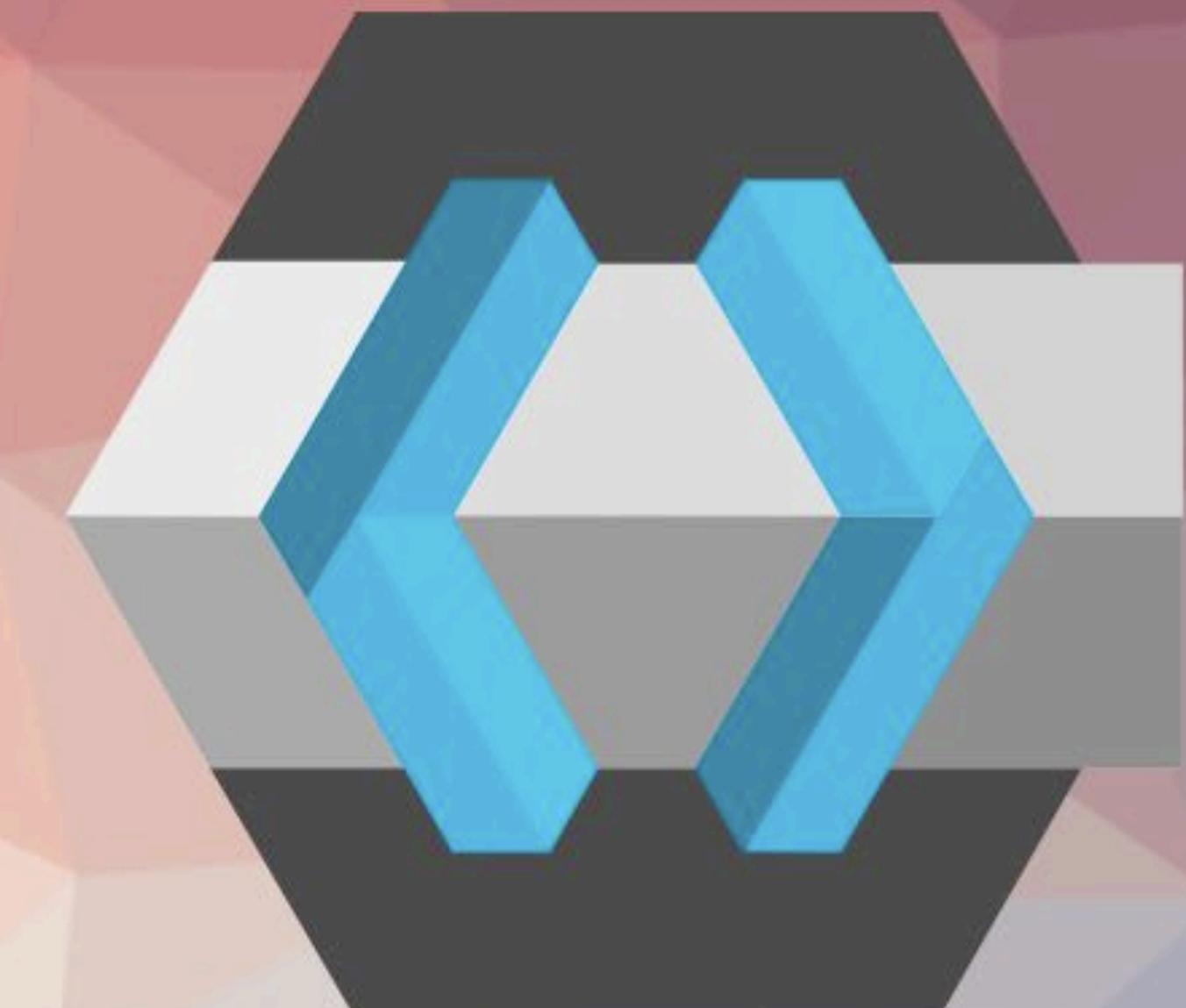
Add authentication to applications and secure services with minimum effort.

No need to deal with storing users or authenticating users.

Keycloak provides user federation, strong authentication, user management, fine-grained authorization, and more.

[Get Started](#)[Download](#)

Latest release 26.1.3



News

05 Mar Introducing the Keycloak Austria User Group

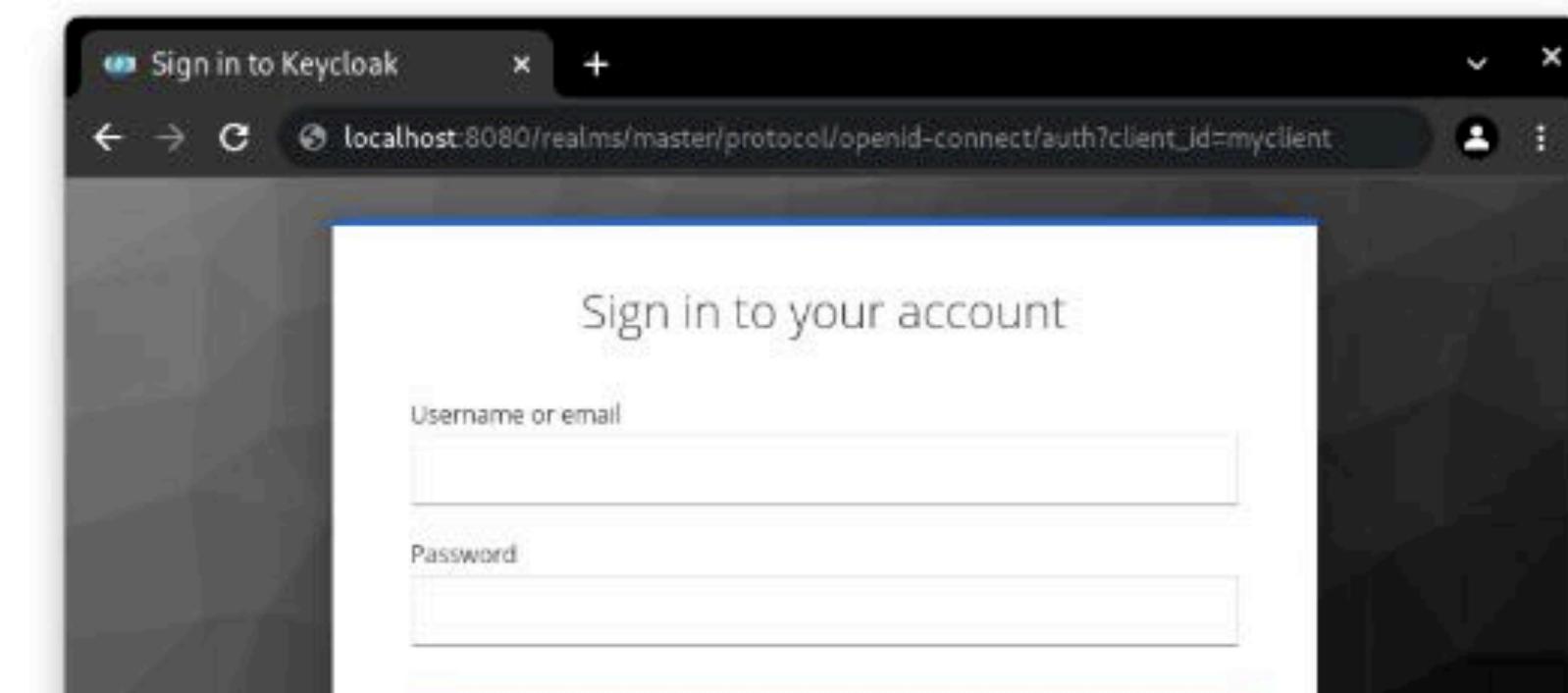
28 Feb Keycloak 26.1.3 released

25 Feb New videos about OpenID Connect and Keycloak from FOSDEM 2025

Single-Sign On

Users authenticate with Keycloak rather than individual applications. This means that your applications don't have to deal with login forms, authenticating users, and storing users. Once logged-in to Keycloak, users don't have to login again to access a different application.

This also applies to logout. Keycloak provides single-sign out, which means users only have to logout once to be logged-out of all applications that use Keycloak.



```
... docker-compose.yml
1version: '3.8'
2
3services:
4  keycloak:
5    image: quay.io/keycloak/keycloak:latest
6    container_name: keycloak
7    command: start-dev
8    environment:
9      KEYCLOAK_ADMIN: admin
10     KEYCLOAK_ADMIN_PASSWORD: admin
11   ports:
12     - "8080:8080"
```

Save this as docker-compose.yml

Run docker compose up -d

Bring up localhost:9191 in a browser and log in with admin / admin



master

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Configure

Realm settings

Authentication

Identity providers

User federation

master realm

Server info

Provider info

Server info

Version

22.0.3

Product

Default

Memory

Total memory

455 MB

Free memory

390 MB

Used memory

65 MB

Profile

Enabled features

Disabled features

ACCOUNT3 Preview

ADMIN_FINE_GRAINED_AUTHZ Preview

CLIENT_SECRET_ROTATION Preview

DECLARATIVE_USER_PROFILE Preview

DOCKER Supported

DYNAMIC_SCOPES Experimental

FIPS Supported

LINKEDIN_OAUTH Supported

MAP_STORAGE Experimental

RECOVERY_CODES Preview

SCRIPTS Preview

TOKEN_EXCHANGE Preview

UPDATE_EMAIL Preview



**Let's look at Spring Boot.
First, we'll look at a web app,
then we'll look at the backend.**

**Keycloak will run on port 9191.
Web app will run on port 8080.
Backend will run on port 8181.**

I'm using a slightly modified version of this GitHub project:

<https://github.com/sivaprasadreddy/spring-security-oauth2-microservices-demo>



Let's look at the web app.
It'll be a Spring Boot project serving
up a Thymeleaf web app.

application.properties in src/main/resources

Keycloak is running at localhost:9191

We're using a realm called demo-realm

```
1 OAUTH_SERVER=http://localhost:9191/realm/demo-realm
2
3 spring.security.oauth2.client.registration.messages-webapp.client-id=messages-webapp
4 spring.security.oauth2.client.registration.messages-webapp.client-
   secret=qVcg0foCUNyYbgF0Sg52zeIhLYy0wXpQ
5 spring.security.oauth2.client.registration.messages-webapp.authorization-grant-
   type=authorization_code
6 spring.security.oauth2.client.registration.messages-webapp.scope=openid, profile
7 spring.security.oauth2.client.registration.messages-webapp.redirect-uri=
   {baseUrl}/login/oauth2/code/messages-webapp
8
9 spring.security.oauth2.client.provider.messages-webapp.issuer-uri=${OAUTH_SERVER}
10
```

```
1 import org.springframework.context.annotation.Bean;
2 import org.springframework.context.annotation.Configuration;
3 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
4 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
5 import org.springframework.security.config.annotation.web.configurers.CorsConfigurer;
6 import org.springframework.security.config.annotation.web.configurers.CsrfConfigurer;
7 import
8     org.springframework.security.oauth2.client.oidc.web.logout.OidcClientInitiatedLogoutSuccessHandler;
9 import org.springframework.security.oauth2.client.registration.ClientRegistrationRepository;
10 import org.springframework.security.web.SecurityFilterChain;
11 import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;
12
13 @Configuration
14 @EnableWebSecurity
15 public class SecurityConfig {
16     private final ClientRegistrationRepository clientRegistrationRepository;
17
18     public SecurityConfig(ClientRegistrationRepository clientRegistrationRepository) {
19         this.clientRegistrationRepository = clientRegistrationRepository;
20     }
21
22     @Bean
23     SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
24         http
25             .authorizeHttpRequests(c ->
26                 c.requestMatchers("/").permitAll()
27                 .anyRequest().authenticated()
28             )
29             .cors(CorsConfigurer::disable)
30             .csrf(CsrfConfigurer::disable)
31             // .oauth2Login(Customizer.withDefaults())
32             .oauth2Login(oauth2 ->
33                 oauth2.userInfoEndpoint(userInfo -> userInfo
34                     .userAuthoritiesMapper(new KeycloakAuthoritiesMapper()))
35                 .logout(logout -> logout
36                     .clearAuthentication(true)
37                     .invalidateHttpSession(true)
38                     .logoutSuccessHandler(oidcLogoutSuccessHandler())
39             );
40         return http.build();
41     }

```

SecurityConfig.java

```
1 class KeycloakAuthoritiesMapper implements GrantedAuthoritiesMapper {  
2  
3     @Override  
4     public Collection<? extends GrantedAuthority> mapAuthorities(  
5             Collection<? extends GrantedAuthority> authorities) {  
6         Set<GrantedAuthority> mappedAuthorities = new HashSet<>();  
7  
8         authorities.forEach(authority -> {  
9             if (authority instanceof SimpleGrantedAuthority) {  
10                 mappedAuthorities.add(authority);  
11             }  
12             else if (authority instanceof OidcUserAuthority oidcUserAuthority) {  
13                 OidcIdToken idToken = oidcUserAuthority.getIdToken();  
14                 Map<String, Object> claims = idToken.getClaims();  
15                 Map<String, Object> realm_access = (Map<String, Object>) claims.get("realm_access");  
16                 if(realm_access != null && !realm_access.isEmpty()) {  
17                     List<String> roles = (List<String>) realm_access.get("roles");  
18                     var list = roles.stream()  
19                         .filter(role -> role.startsWith("ROLE_"))  
20                         .map(SimpleGrantedAuthority::new).toList();  
21                     mappedAuthorities.addAll(list);  
22                 }  
23             } else if (authority instanceof OAuth2UserAuthority oauth2UserAuthority) {  
24                 Map<String, Object> userAttributes = oauth2UserAuthority.getAttributes();  
25                 // Map the attributes found in userAttributes  
26                 // to one or more GrantedAuthority's and add it to mappedAuthorities  
27             }  
28         });  
29         return mappedAuthorities;  
30     }  
31 }
```

```
1 import org.slf4j.Logger;
2 import org.slf4j.LoggerFactory;
3 import org.springframework.security.core.Authentication;
4 import org.springframework.security.core.GrantedAuthority;
5 import org.springframework.security.core.context.SecurityContextHolder;
6 import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
7 import org.springframework.security.oauth2.client.OAuth2AuthorizedClientService;
8 import org.springframework.security.oauth2.client.authentication.OAuth2AuthenticationToken;
9 import org.springframework.security.oauth2.core.OidcUserInfo;
10 import org.springframework.security.oauth2.core.oidc.user.DefaultOidcUser;
11 import org.springframework.stereotype.Service;
12
13 import java.util.Collection;
14 import java.util.HashMap;
15 import java.util.List;
16 import java.util.Map;
17
18 @Service
19 public class SecurityHelper {
20     private static final Logger log = LoggerFactory.getLogger(SecurityHelper.class);
21
22     private final OAuth2AuthorizedClientService authorizedClientService;
23
24     public SecurityHelper(OAuth2AuthorizedClientService authorizedClientService) {
25         this.authorizedClientService = authorizedClientService;
26     }
27
28     public String getAccessToken() {
29         Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
30         if(!(authentication instanceof OAuth2AuthenticationToken oauthToken)) {
31             return null;
32         }
33         OAuth2AuthorizedClient client = authorizedClientService.loadAuthorizedClient(
34             oauthToken.getAuthorizedClientRegistrationId(), oauthToken.getName());
35
36         log.info("Access token = " + client.getAccessToken().getTokenValue());
37         return client.getAccessToken().getTokenValue();
38     }
39
40     public static Map<String, Object> getLoginUserDetails() {
41         Map<String, Object> map = new HashMap<>();
42         Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
43         if(!(authentication instanceof OAuth2AuthenticationToken)) {
44             return null;
45         }
46         DefaultOidcUser principal = (DefaultOidcUser) authentication.getPrincipal();
47
48         log.info("ID token = " + principal.getIdToken().getTokenValue());
49         Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities();
50         List<String> roles = authorities.stream().map(GrantedAuthority::getAuthority).toList();
51         OidcUserInfo userInfo = principal.getUserInfo();
52
53         map.put("id", userInfo.getSubject());
54         map.put("fullName", userInfo.getFullName());
55         map.put("email", userInfo.getEmail());
56         map.put("username", userInfo.getPreferredUsername());
57         map.put("roles", roles);
58     }
59 }
60 }
```

```
1 <!DOCTYPE html>
2 <html lang="en"
3     xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:th="http://www.thymeleaf.org"
5     xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity">
6
7     <ul class="navbar-nav mb-2 mb-lg-0">
8         <li class="nav-item sec:authorize='!isAuthenticated()'>
9             <a class="nav-link" href="/oauth2/authorization/messages-webapp">Login</a>
10        </li>
11        <li class="nav-item sec:authorize='isAuthenticated()'>
12            <a class="nav-link" href="/logout">Logout</a>
13        </li>
14    </ul>
15
16    <div sec:authorize="isAuthenticated()">
17        <div class="card">
18            <div class="card-body">
19                <form method="post" action="/messages">
20                    <div class="mb-3">
21                        <label for="content" class="form-label">Message</label>
22                        <textarea class="form-control" id="content" name="content"></textarea>
23                    </div>
24                    <button type="submit" class="btn btn-primary">Submit</button>
25                </form>
26            </div>
27        </div>
28    </div>
29
```

```
1 @Service
2 public class MessageServiceClient {
3     private static final Logger log = LoggerFactory.getLogger(MessageServiceClient.class);
4     private static final String MESSAGE_SVC_BASE_URL = "http://localhost:8181";
5
6     private final SecurityHelper securityHelper;
7     private final RestTemplate restTemplate;
8
9     public MessageServiceClient(SecurityHelper securityHelper, RestTemplate restTemplate) {
10         this.securityHelper = securityHelper;
11         this.restTemplate = restTemplate;
12     }
13
14     public void createMessage(Message message) {
15         try {
16             String url = MESSAGE_SVC_BASE_URL + "/api/messages";
17             HttpHeaders headers = new HttpHeaders();
18             headers.add("Authorization", "Bearer " + securityHelper.getAccessToken());
19             HttpEntity<?> httpEntity = new HttpEntity<>(message, headers);
20             ResponseEntity<Message> response = restTemplate.exchange(
21                 url, HttpMethod.POST, httpEntity,
22                 new ParameterizedTypeReference<>() {});
23             log.info("Create message response code: {}", response.getStatusCode());
24         } catch (Exception e) {
25             log.error("Error while creating message", e);
26         }
27     }
```



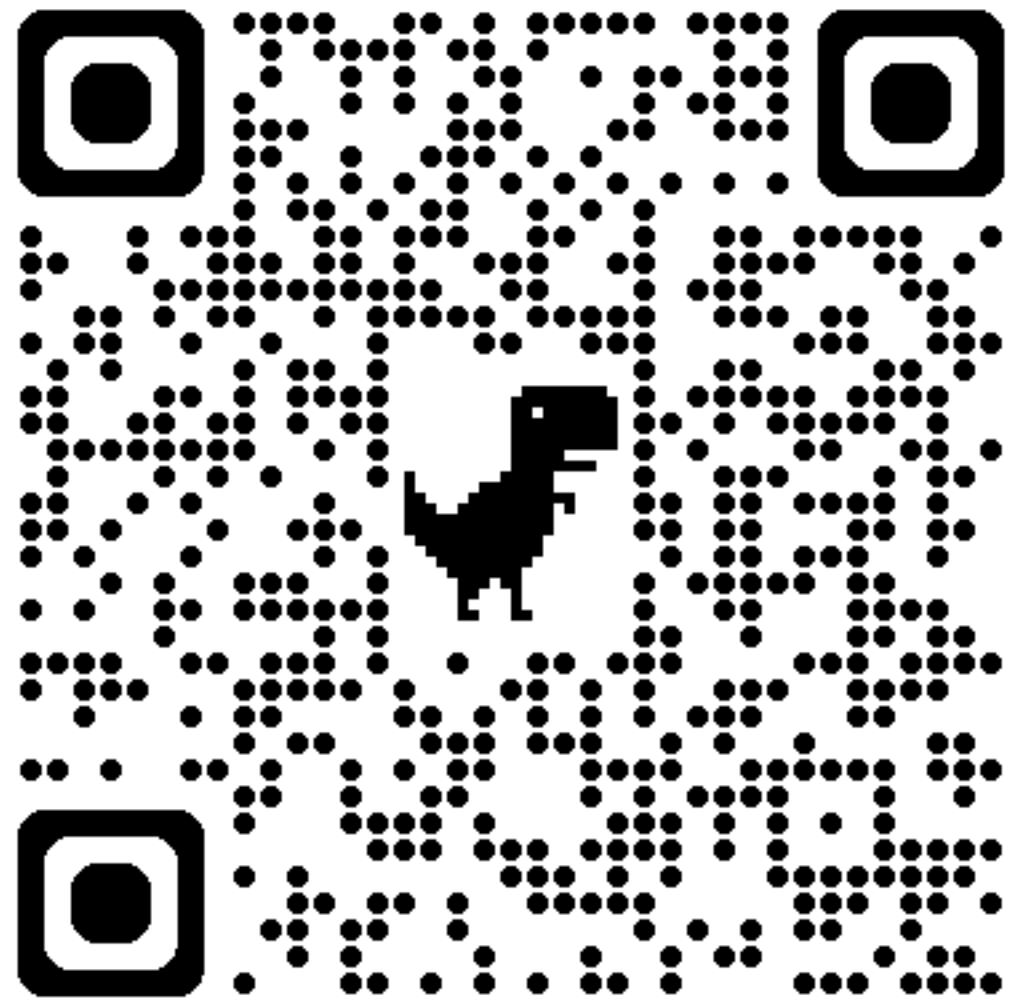
Let's look at the backend.

application.properties in src/main/resources

```
1 spring.application.name=messages-service
2 server.port=8181
3 OAUTH_SERVER=http://localhost:9191/realmns/demo-realm
4 spring.security.oauth2.resource-server.jwt.issuer-uri=${OAUTH_SERVER}
```

```
1 @Configuration
2 @EnableWebSecurity
3 public class SecurityConfig {
4
5     @Bean
6     SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
7         http
8             .authorizeHttpRequests(c ->
9                 c
10                .requestMatchers(HttpMethod.GET, "/api/messages").permitAll()
11                .requestMatchers(HttpMethod.POST, "/api/messages/archive").hasAnyRole("ADMIN", "ADMIN_JOB")
12                .anyRequest().authenticated()
13            )
14            .sessionManagement(c -> c.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
15            .cors(CorsConfigurer::disable)
16            .csrf(CsrfConfigurer::disable)
17            .oauth2ResourceServer(oauth2 ->
18                //oauth2.jwt(Customizer.withDefaults())
19                oauth2.jwt(jwt -> jwt.jwtAuthenticationConverter(new KeycloakJwtAuthenticationConverter())))
20            );
21         return http.build();
22     }
23 }
```

Slides on GitHub



[LinkedIn](#)



CGI

kellyivymorrison@gmail.com

