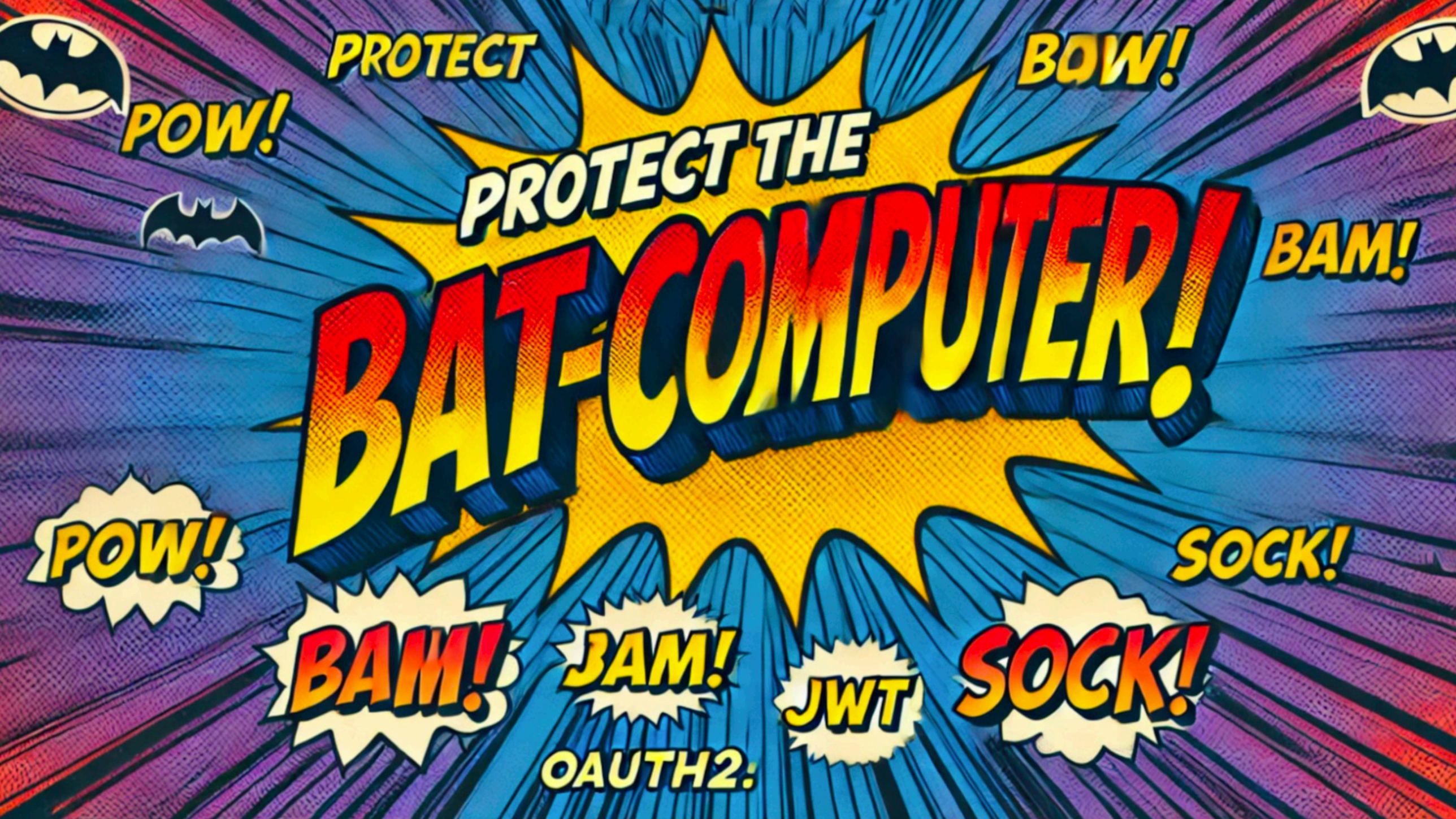

PROTECT
POW!
BOW!
PROTECT THE
BAT-COMPUTER!
BAM!
POW!
SOCK!
BAM!
JAM!
JWT
SOCK!
OAUTH2.

Kelly Morrison, Ph.D. Computer Engineering

The University of Georgia

TEAM Daugherty Est. 1985

CNN

The Home Depot

COLORBUS

CGI

Auburn University

Turner

Digital Output Magazine

THE CLIMATE CORPORATION

NASA Marshall Space Flight Center

PATHFIRE Enabling Digital Media

Edward Jones MAKING SENSE OF INVESTING

Performance MATTERS

BAYER

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.

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

But your birth certificate **authenticates** you as Harleen Quinzel.

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?

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.

Thanks, Wonder Woman!

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!


TARGET
SYSTEM STATUS
1
2
BRICKBUILDERS101

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

Thanks, Wonder Woman!

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

Client

Authorization Code

Resource Server

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…

Resource Server

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

Authorization Code

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

Resource Server

Client

Hooray!

Resource

Client

ID Token

Access Token

Resource


TARGET
SYSTEM STATUS
1
2
BRICKBUILDERS101

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

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.

Here’s an example ID token.

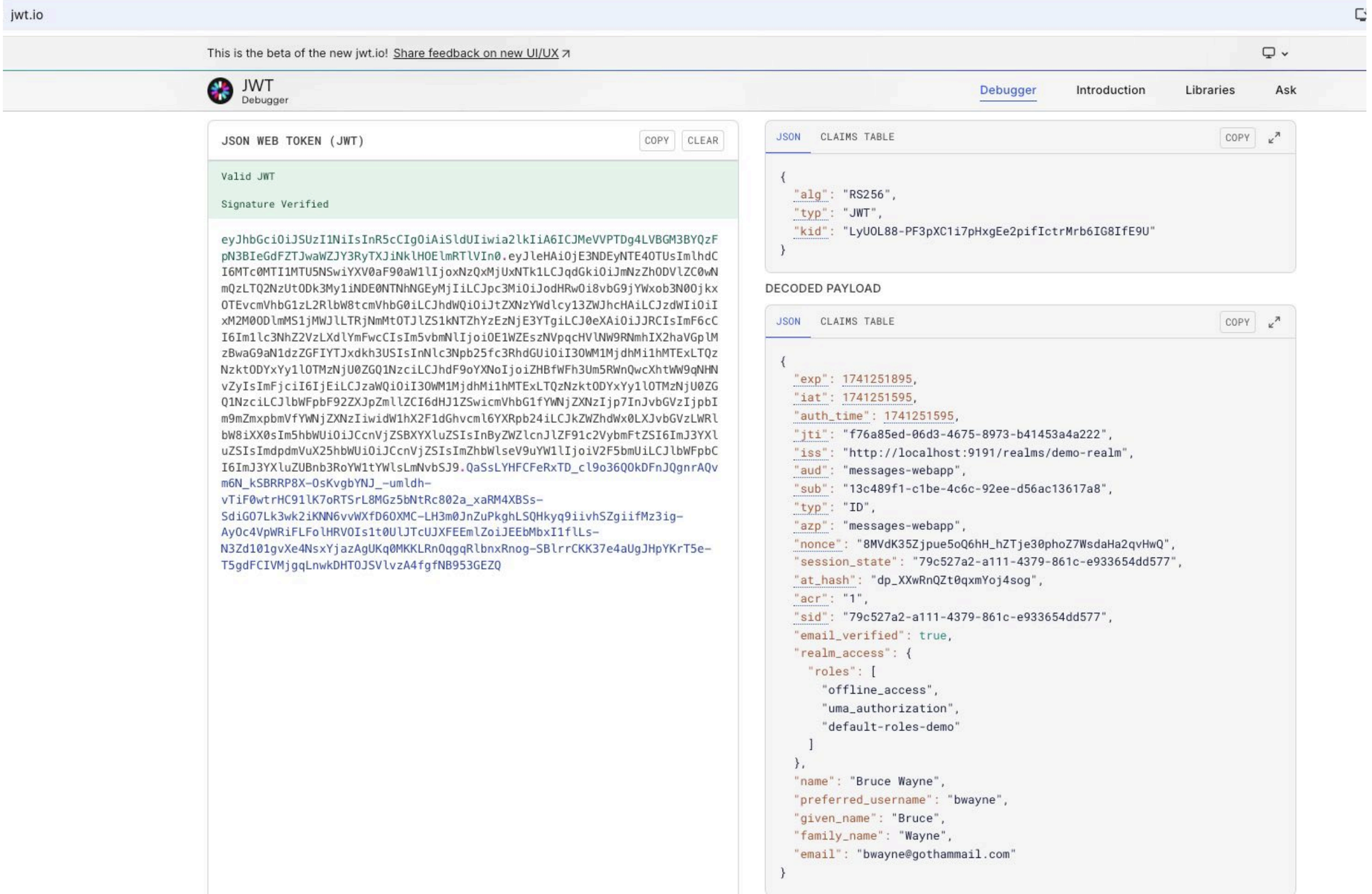
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vbXktZG9tYWluLmF1dGgwLmNvbSIsInN1YiI6ImF1dGgwfDEyMzQ1NiIsImF1ZCI6IjEyMzRhYmNkZWYiLCJleHAiOjEzMTEyODE5NzAsImlhdCI6MTMx
MTI4MDk3MCwibmFtZSI6IkphbmUgRG9lIiwiZ2l2ZW5fbmFtZSI6IkphbmUiLCJmYW1pbHlfbmFtZSI6IkRvZSJ9.bql-jxlG9B_bielkqOnjTY9Di9FillFb6IMQINXoYsw

And here’s an example access token.

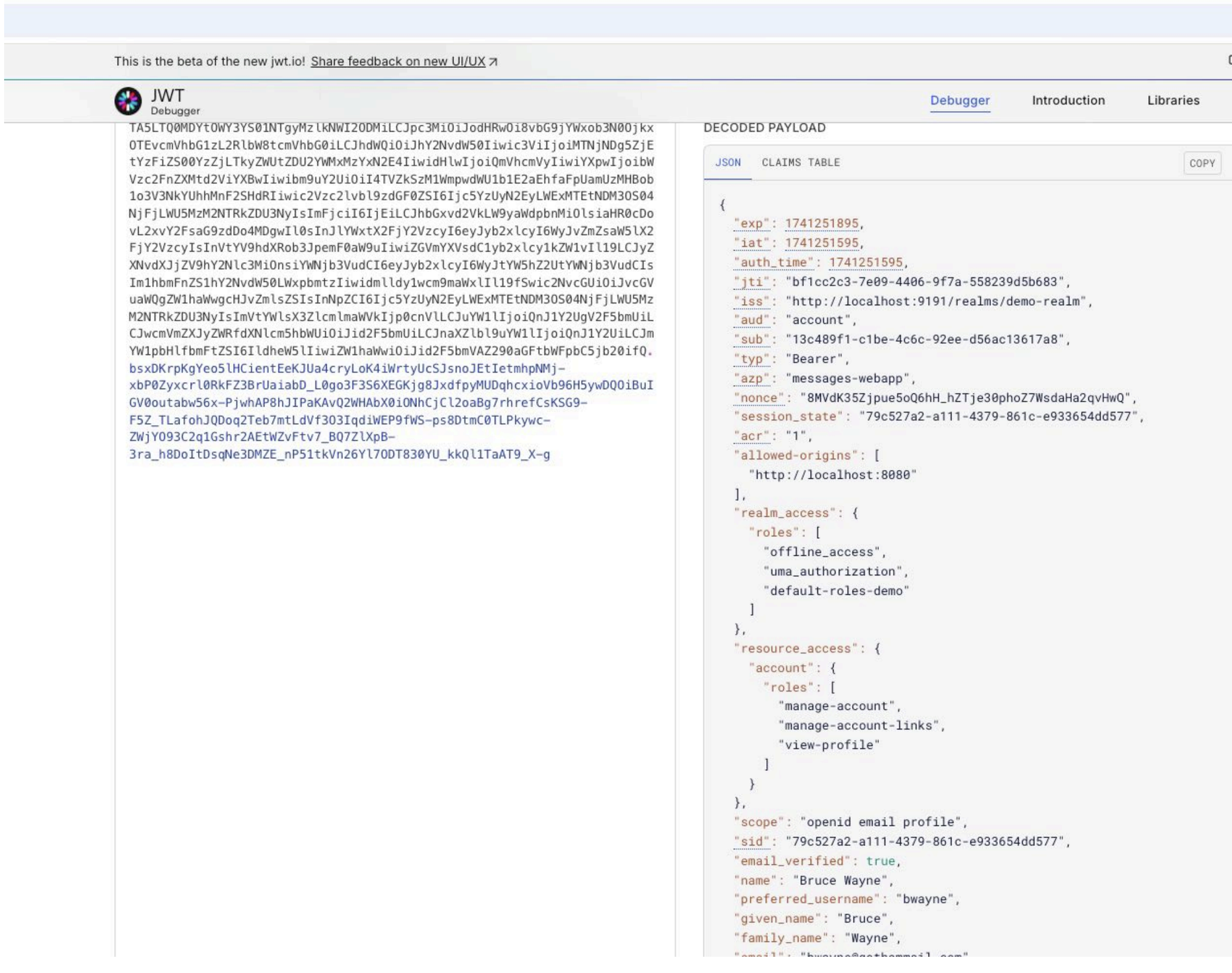
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

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

ID Token


jwt.io
This is the beta of the new jwt.io! Share feedback on new UI/UX
JWT
Debugger
Debugger
Introduction
Libraries
Ask
JSON WEB TOKEN (JWT)
COPY
CLEAR
Valid JWT
Signature Verified
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJMeVVPTDg4LVBGM3BYQzFpN3BIeGdFZTJwaWZJY3RyTXJiNklHOElmRTlVIn0.eyJleHAiOjE3NDEyNTE4OTUsImlhdCI6MTc0MTI1MTU5NSwiYXV0aF90aW1lIjoxNzQxMjUxNTk1LCJqdGkiOiJmNzZhODVlZC0wNmQzLTQ2NzUtODk3My1iNDE0NTNhNGEyMjIiLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjkxOTEvcmVhbG1zL2RlbW8tcmVhbG0iLCJhdWQiOiJtZXNzYWdlcy13ZWJhcHAiLCJzdWIiOiIxM2M0ODlmMS1jMWJlLTRjNmMtOTJlZS1kNTZhYzEzNjE3YTgiLCJ0eXAiOiJJRCIsImF6cCI6Im1lc3NhZ2VzLXdlYmFwcCIsIm5vbmNlIjoiOE1WZEszNVpqcHVlNW9RNmhIX2haVGplMzBwaG9aN1dzZGFIYTJxdkh3USIsInNlc3Npb25fc3RhdGUiOiI3OWM1MjdhMi1hMTEzLTQzNzktODYxYy1lOTMzNjU0ZGQ1NzciLCJhdF9oYXNoIjoiZHBfWFh3UmpRWnQwcXhtWW9qNHNvZyIsImFjciI6IjEiLCJzaWQiOiI3OWM1MjdhMi1hMTEzLTQzNzktODYxYy1lOTMzNjU0ZGQ1NzciLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbIm9mZmxpbmVfYWNjZXNzIiwidW1hX2F1dGhvcml6YXRpb24iLCJkZWZhdWx0LXJvbGVzLWRlbW8iXX0sInJlc291cmNlX2FjY2VzcyI6eyJhY2NvdW50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2NvdW50IiwibWFuYWdlLWFjY291bnQtbGlua3MiLCJ2aWV3LXByb2ZpbGUiXX19LCJuYW1lIjoiQnJ1Y2UgV2F5bmUiLCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJid2F5bmUiLCJnaXZlbl9uYW1lIjoiQnJ1Y2UiLCJmYW1pbHlfbmFtZSI6IldheW5lIiwiZW1haWwiOiJid2F5bmVAZ290aGFtbWFpbC5jb20ifQ.QaSsLYHFCFeRxTD_cl9o36QOkDFnJQgnrAQvm6N_kSBRRP8X-OsKvgbYNJ_-umldh-vTiF0wtrHC91lK7oRTSrL8MGz5bNtRc802a_xaRM4XBSs-SdiGO7Lk3wk2iKNN6vvWXfD6OXMC-LH3m0JnZuPkghLSQHkyq9iivhSZgiifMz3ig-AyOc4VpWRiFLFolHRVOIs1t0UlJTcUJXFEEmlZoiJEEbMbxI1flLs-N3Zd101gvXe4NsxYjazAgUKq0MKKLRnOqgqRlbnxRnog-SBlrrCKK37e4aUgJHpYKrT5e-T5gdFCIVMjgqLnwkDHTOJSVlvzA4fgfNB953GEZQ
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"
}

Access Token

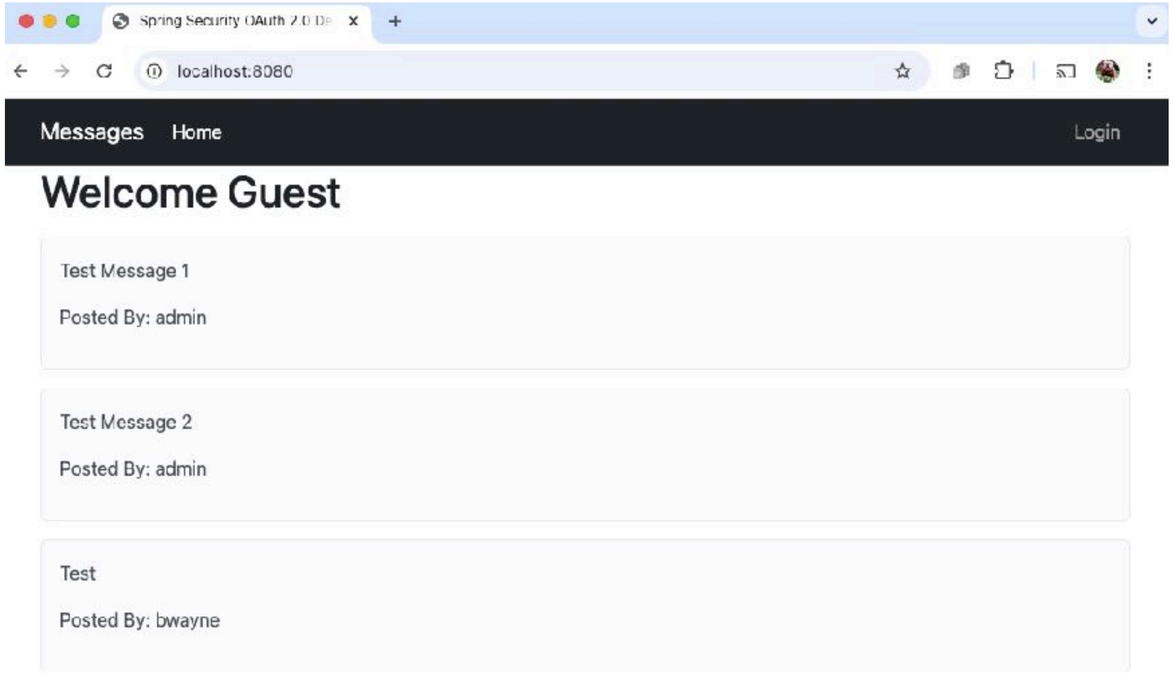


Access Tokens

- Access Tokens are sent as Authorization Headers.
- The format is:
  - Authorization Bearer *jwt*

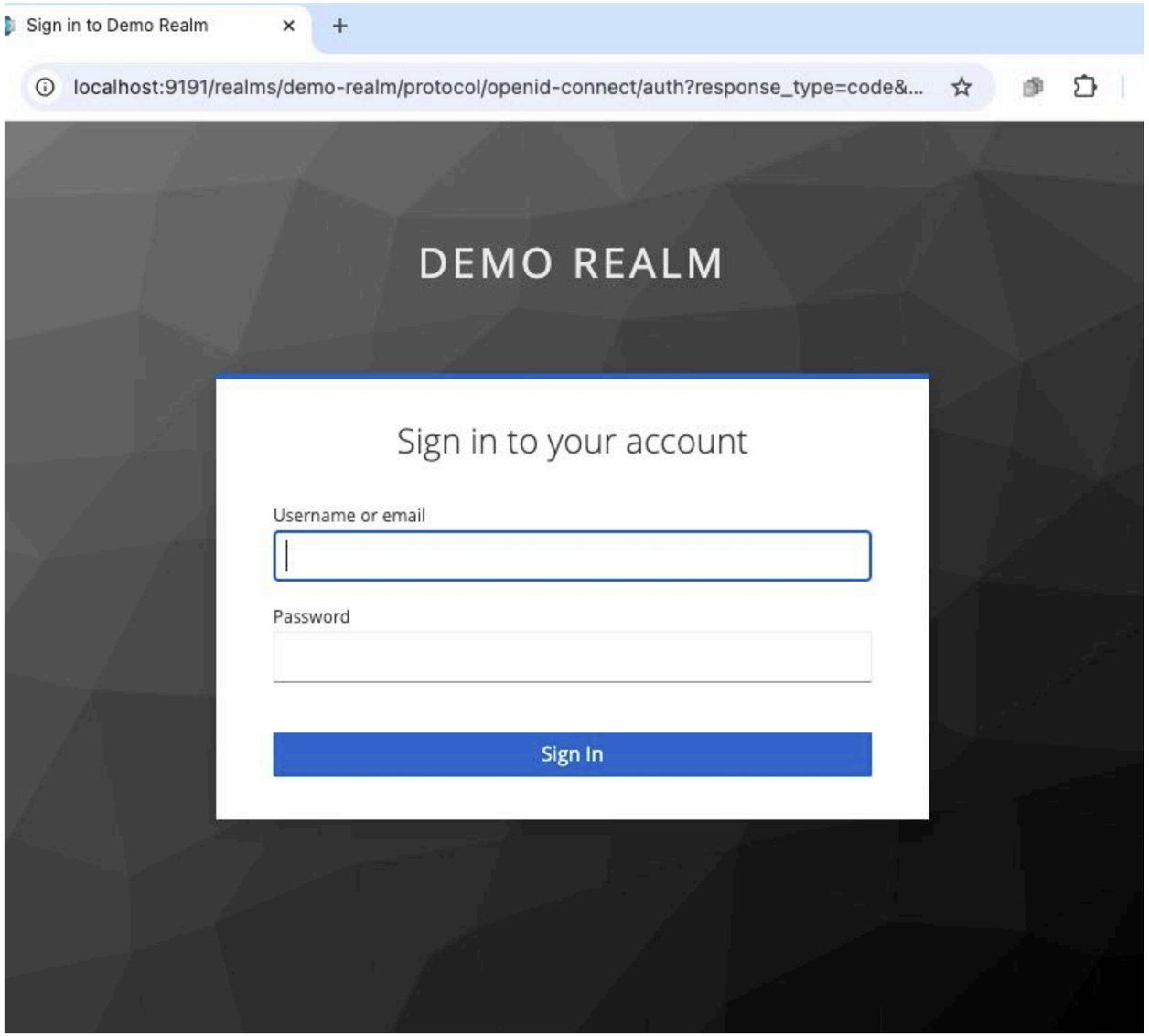
Let’s go through this one last time,

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.

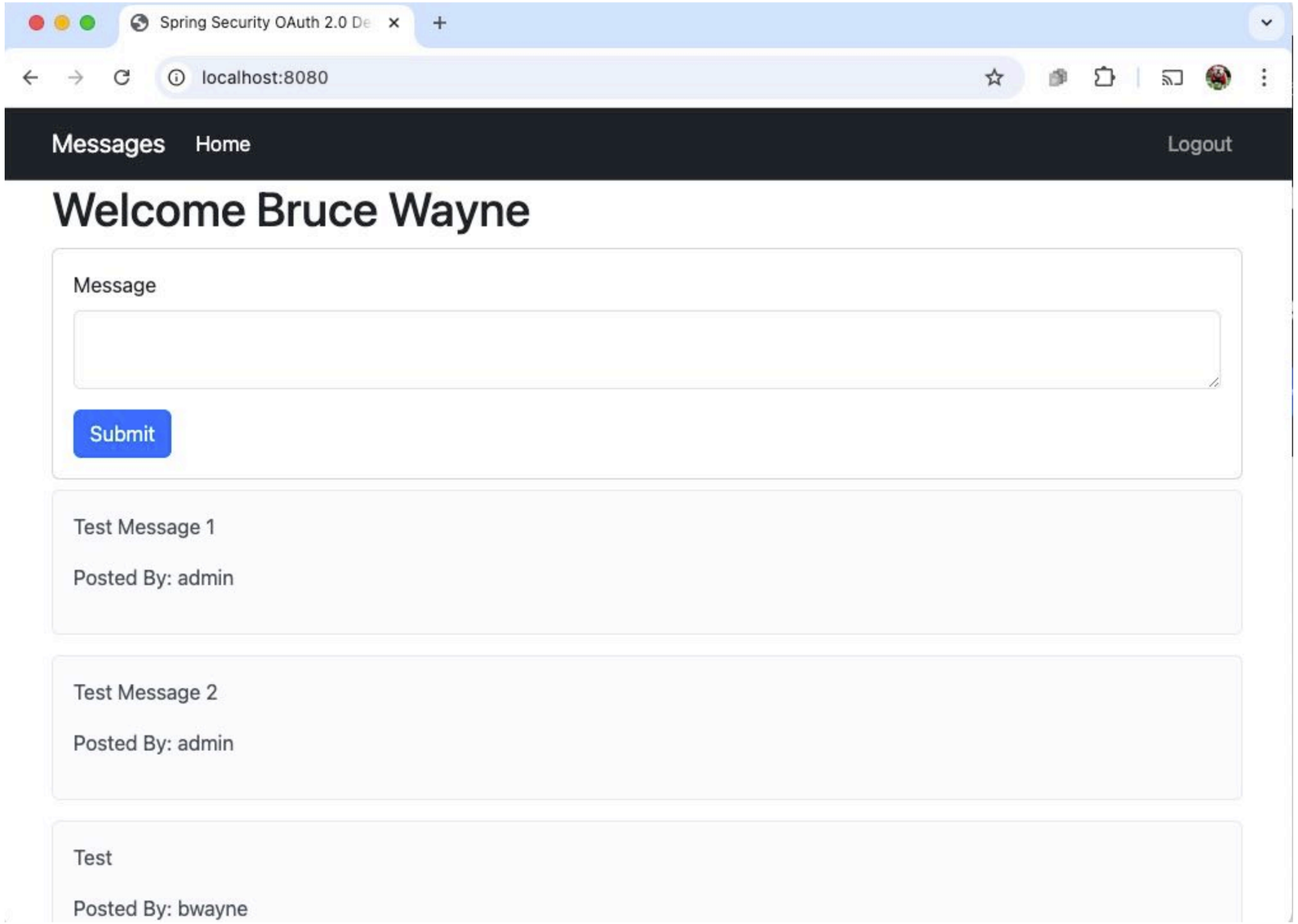
The user enters their credentials and hits “Sign In”.

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.



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!

KEYCLOAK

GitHub Stars 26k | Guides | Docs | Downloads | Community | Blog

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

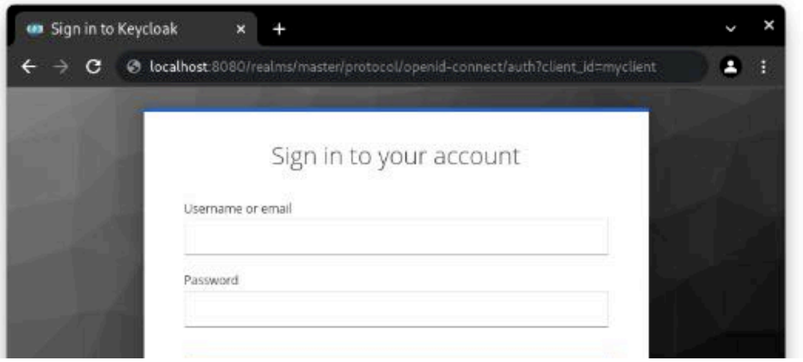
28 Feb Keycloak 26.1.3 released

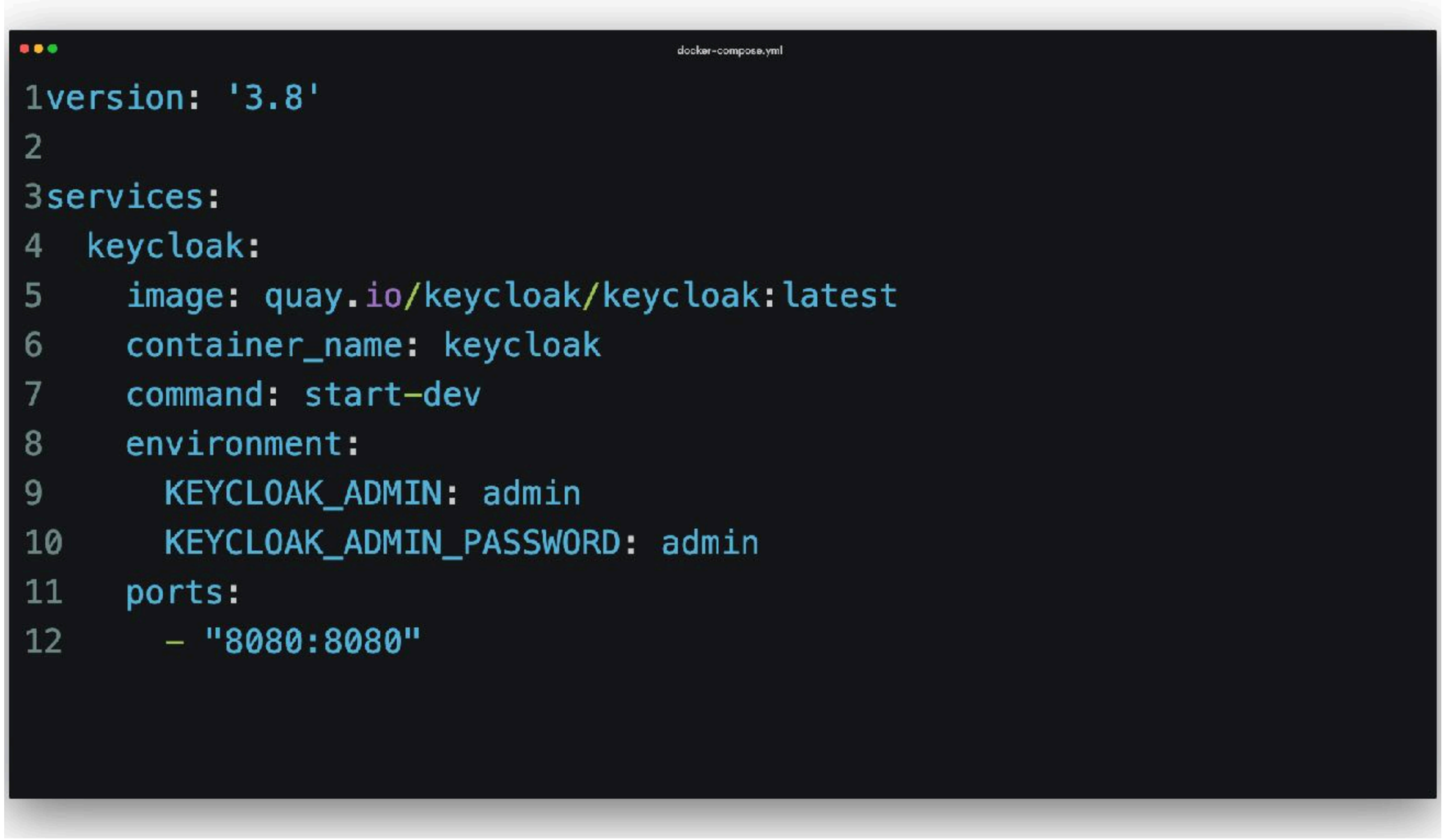
25 Feb New videos about OpenID Connect and Keycloak from FOSDEM 2025

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.

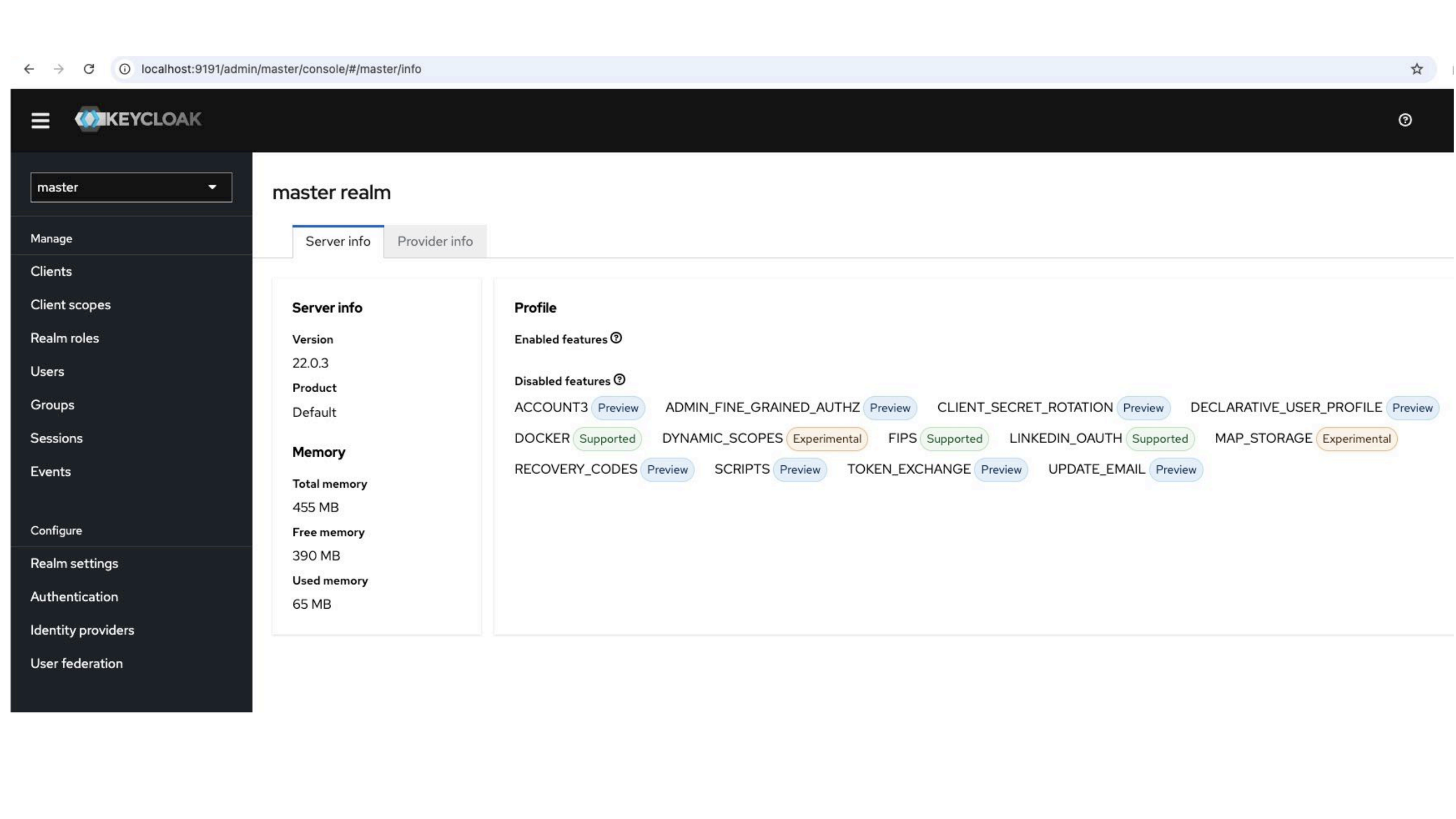




Save this as docker-compose.yml

Run docker compose up -d

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


localhost:9191/admin/master/console/#/master/info
KEYCLOAK
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

Title

```
OAUTH_SERVER=http://localhost:9191/realms/demo-realm

spring.security.oauth2.client.registration.messages-webapp.client-id=messages-webapp
spring.security.oauth2.client.registration.messages-webapp.client-secret=qVcg0foCUNyYbgF0Sg52zeIhLYyOwXpQ
spring.security.oauth2.client.registration.messages-webapp.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.messages-webapp.scope=openid, profile
spring.security.oauth2.client.registration.messages-webapp.redirect-uri={baseUrl}/login/oauth2/code/messages-webapp

spring.security.oauth2.client.provider.messages-webapp.issuer-uri=${OAUTH_SERVER}

```

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.CorsConfigurer;
import org.springframework.security.config.annotation.web.configurers.CsrfConfigurer;
import
 org.springframework.security.oauth2.client.oidc.web.logout.OidcClientInitiatedLogoutSuccessHand
 ler;
import org.springframework.security.oauth2.client.registration.ClientRegistrationRepository;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    private final ClientRegistrationRepository clientRegistrationRepository;

    public SecurityConfig(ClientRegistrationRepository clientRegistrationRepository) {
        this.clientRegistrationRepository = clientRegistrationRepository;
    }

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(c ->
                    c.requestMatchers("/").permitAll()
                    .anyRequest().authenticated()
            )
            .cors(CorsConfigurer::disable)
            .csrf(CsrfConfigurer::disable)
            //.oauth2Login(Customizer.withDefaults())
            .oauth2Login(oauth2 ->
                oauth2.userInfoEndpoint(userInfo -> userInfo
                        .userAuthoritiesMapper(new KeycloakAuthoritiesMapper())))
            .logout(logout -> logout
                    .clearAuthentication(true)
                    .invalidateHttpSession(true)
                    .logoutSuccessHandler(oidcLogoutSuccessHandler())
            );
        return http.build();
    }

```

SecurityConfig.java

Title

```java
class KeycloakAuthoritiesMapper implements GrantedAuthoritiesMapper {

    @Override
    public Collection<? extends GrantedAuthority> mapAuthorities(
            Collection<? extends GrantedAuthority> authorities) {
        Set<GrantedAuthority> mappedAuthorities = new HashSet<>();

        authorities.forEach(authority -> {
            if (authority instanceof SimpleGrantedAuthority) {
                mappedAuthorities.add(authority);
            }
            else if (authority instanceof OidcUserAuthority oidcUserAuthority) {
                OidcIdToken idToken = oidcUserAuthority.getIdToken();
                Map<String, Object> claims = idToken.getClaims();
                Map<String,Object> realm_access = (Map<String, Object>) claims.get("realm_access");
                if(realm_access != null && !realm_access.isEmpty()) {
                    List<String> roles = (List<String>) realm_access.get("roles");
                    var list = roles.stream()
                            .filter(role -> role.startsWith("ROLE_"))
                            .map(SimpleGrantedAuthority::new).toList();
                    mappedAuthorities.addAll(list);
                }
            } else if (authority instanceof OAuth2UserAuthority oauth2UserAuthority) {
                Map<String, Object> userAttributes = oauth2UserAuthority.getAttributes();
                // Map the attributes found in userAttributes
                // to one or more GrantedAuthority's and add it to mappedAuthorities
            }
        });
        return mappedAuthorities;
    }
}
```

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientService;
import org.springframework.security.oauth2.client.authentication.OAuth2AuthenticationToken;
import org.springframework.security.oauth2.core.oidc.OidcUserInfo;
import org.springframework.security.oauth2.core.oidc.user.DefaultOidcUser;
import org.springframework.stereotype.Service;

import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Service
public class SecurityHelper {
    private static final Logger log = LoggerFactory.getLogger(SecurityHelper.class);

    private final OAuth2AuthorizedClientService authorizedClientService;

    public SecurityHelper(OAuth2AuthorizedClientService authorizedClientService) {
        this.authorizedClientService = authorizedClientService;
    }

    public String getAccessToken() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        if(!(authentication instanceof OAuth2AuthenticationToken oauthToken)) {
            return null;
        }
        OAuth2AuthorizedClient client = authorizedClientService.loadAuthorizedClient(
                oauthToken.getAuthorizedClientRegistrationId(), oauthToken.getName());

        log.info("Access token = " + client.getAccessToken().getTokenValue());
        return client.getAccessToken().getTokenValue();
    }

    public static Map<String, Object> getLoginUserDetails() {
        Map<String, Object> map = new HashMap<>();
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        if(!(authentication instanceof OAuth2AuthenticationToken)) {
            return null;
        }
        DefaultOidcUser principal = (DefaultOidcUser) authentication.getPrincipal();

        log.info("ID token = " + principal.getIdToken().getTokenValue());
        Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities();
        List<String> roles = authorities.stream().map(GrantedAuthority::getAuthority).toList();
        OidcUserInfo userInfo = principal.getUserInfo();

        map.put("id", userInfo.getSubject());
        map.put("fullName", userInfo.getFullName());
        map.put("email", userInfo.getEmail());
        map.put("username", userInfo.getPreferredUsername());
        map.put("roles", roles);
        return map;
    }
}
```

```html
<!DOCTYPE html>
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity">

            <ul class="navbar-nav mb-2 mb-lg-0">
              <li class="nav-item" sec:authorize="!isAuthenticated()">
                  <a class="nav-link" href="/oauth2/authorization/messages-webapp">Login</a>
              </li>
              <li class="nav-item" sec:authorize="isAuthenticated()">
                  <a class="nav-link" href="/logout">Logout</a>
              </li>
            </ul>

    <div sec:authorize="isAuthenticated()">
      <div class="card">
          <div class="card-body">
              <form method="post" action="/messages">
                  <div class="mb-3">
                      <label for="content" class="form-label">Message</label>
                      <textarea class="form-control" id="content" name="content"></textarea>
                  </div>
                  <button type="submit" class="btn btn-primary">Submit</button>
              </form>
          </div>
      </div>
   </div>

```

Title

```java
@Service
public class MessageServiceClient {
    private static final Logger log = LoggerFactory.getLogger(MessageServiceClient.class);
    private static final String MESSAGE_SVC_BASE_URL = "http://localhost:8181";

    private final SecurityHelper securityHelper;
    private final RestTemplate restTemplate;

    public MessageServiceClient(SecurityHelper securityHelper, RestTemplate restTemplate) {
        this.securityHelper = securityHelper;
        this.restTemplate = restTemplate;
    }

    public void createMessage(Message message) {
        try {
            String url = MESSAGE_SVC_BASE_URL + "/api/messages";
            HttpHeaders headers = new HttpHeaders();
            headers.add("Authorization", "Bearer " + securityHelper.getAccessToken());
            HttpEntity<?> httpEntity = new HttpEntity<>(message, headers);
            ResponseEntity<Message> response = restTemplate.exchange(
                    url, HttpMethod.POST, httpEntity,
                    new ParameterizedTypeReference<>() {});
            log.info("Create message response code: {}", response.getStatusCode());
        } catch (Exception e) {
            log.error("Error while creating message", e);
        }
    }
```

Let’s look at the backend.

application.properties in src/main/resources

Title

```
spring.application.name=messages-service
server.port=8181
OAUTH_SERVER=http://localhost:9191/realms/demo-realm
spring.security.oauth2.resourceserver.jwt.issuer-uri=${OAUTH_SERVER}
```

Title

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(c ->
                c
                    .requestMatchers(HttpMethod.GET, "/api/messages").permitAll()
                    .requestMatchers(HttpMethod.POST, "/api/messages/archive").hasAnyRole("ADMIN", "ADMIN_JOB")
                    .anyRequest().authenticated()
            )
            .sessionManagement(c -> c.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .cors(CorsConfigurer::disable)
            .csrf(CsrfConfigurer::disable)
            .oauth2ResourceServer(oauth2 ->
                //oauth2.jwt(Customizer.withDefaults())
                oauth2.jwt(jwt -> jwt.jwtAuthenticationConverter(new KeycloakJwtAuthenticationConverter()))
            );
        return http.build();
    }
}
```

Some tips for debugging OAuth2 / OIDC

- Check and recheck your config files. Spring relies heavily on them, and getting them wrong can be hard to fix.
- Set the Spring oauth2 and security packages to TRACE level logging.
- Use a local identity server and verify that you can hit the endpoints. Look for active sessions.
- Try using curl to access the identify server outside of your app.
- Print out the ID token and access token and verify they’re correct.
- Make sure your redirect URLs are correct.
- Make sure you’re using the correct grant type.

Slides on GitHub

LinkedIn

CGI

kellyivymorrison@gmail.com