



Okta Platform for Developers

Joël Franusic, Sr. Product Marketing Manager

October 17th, 2018

Agenda

- OAuth and OpenID Connect (in plain English)
- Implementing OAuth and OpenID Connect



OAuth and OpenID Connect (in plain English)

A lot of confusion around OAuth.

- × Terminology and jargon
- × Incorrect advice



Identity use cases (circa 2007)

- Simple login – forms and cookies
- Single sign-on across sites – SAML
- Mobile app login – ???
- Delegated authorization – ???



The delegated authorization problem


OR: How can I let a website access my data (without giving it my password)?

Don't do it this way!

Are your friends already on Yelp?

Many of your friends may already be here, now you can find out. Just log in and we'll display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service

☒  Hotmail ☐  ☐  AOL Mail ☒  Gmail

Your Email Address

(e.g. bob@gmail.com)

Your Gmail Password

(The password you use to log into your Gmail email)

[Skip this step](#)

[Check Contacts](#)



Don't do it this way!


Step 1
Find Friends

Step 2
Profile Information

Step 3
Profile Picture

Are your friends already on Facebook?


Many of your friends may already be here. Searching your email account is the fastest way to find your friends on Facebook.


 Gmail

Your Email:


Email Password:

Find Friends


 Facebook will not store your password.

 Yahoo!

Find Friends

 Windows Live Hotmail

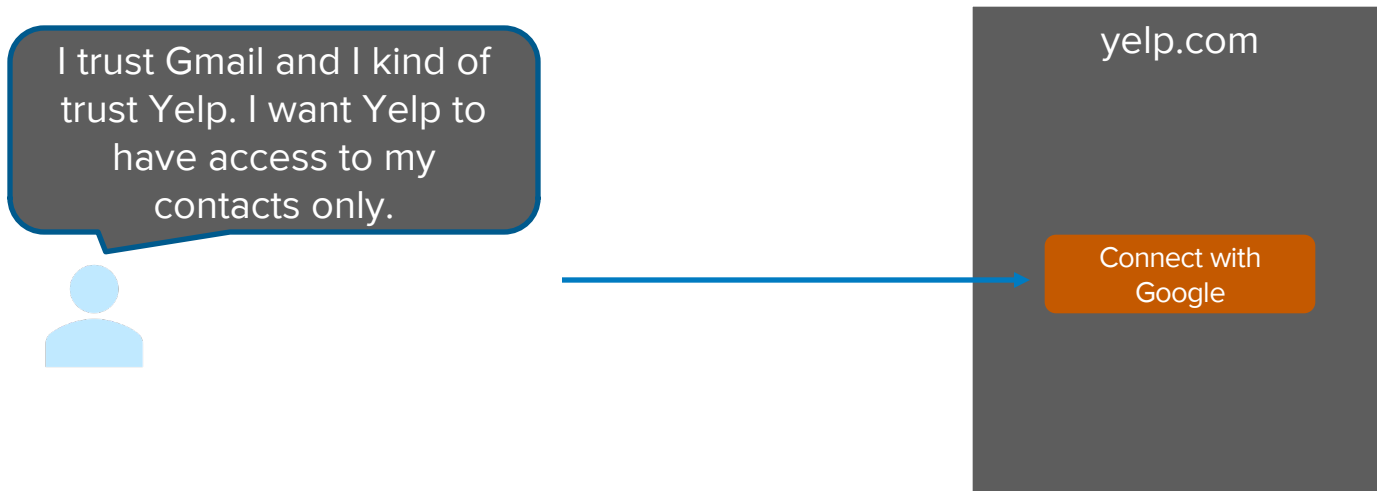
Find Friends

 Other Email Service

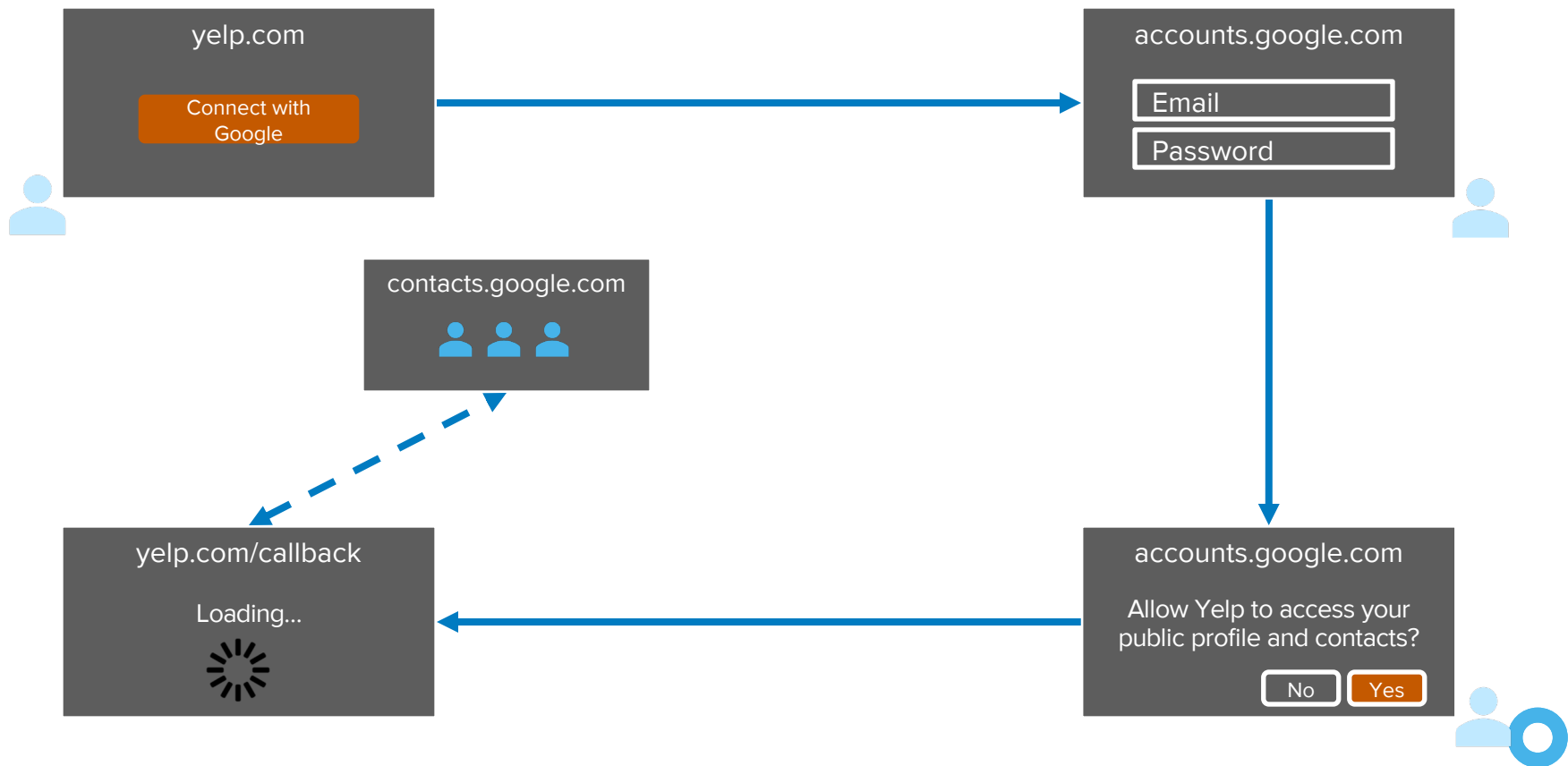
Find Friends



Delegated authorization with OAuth 2.0



Delegated authorization with OAuth 2.0

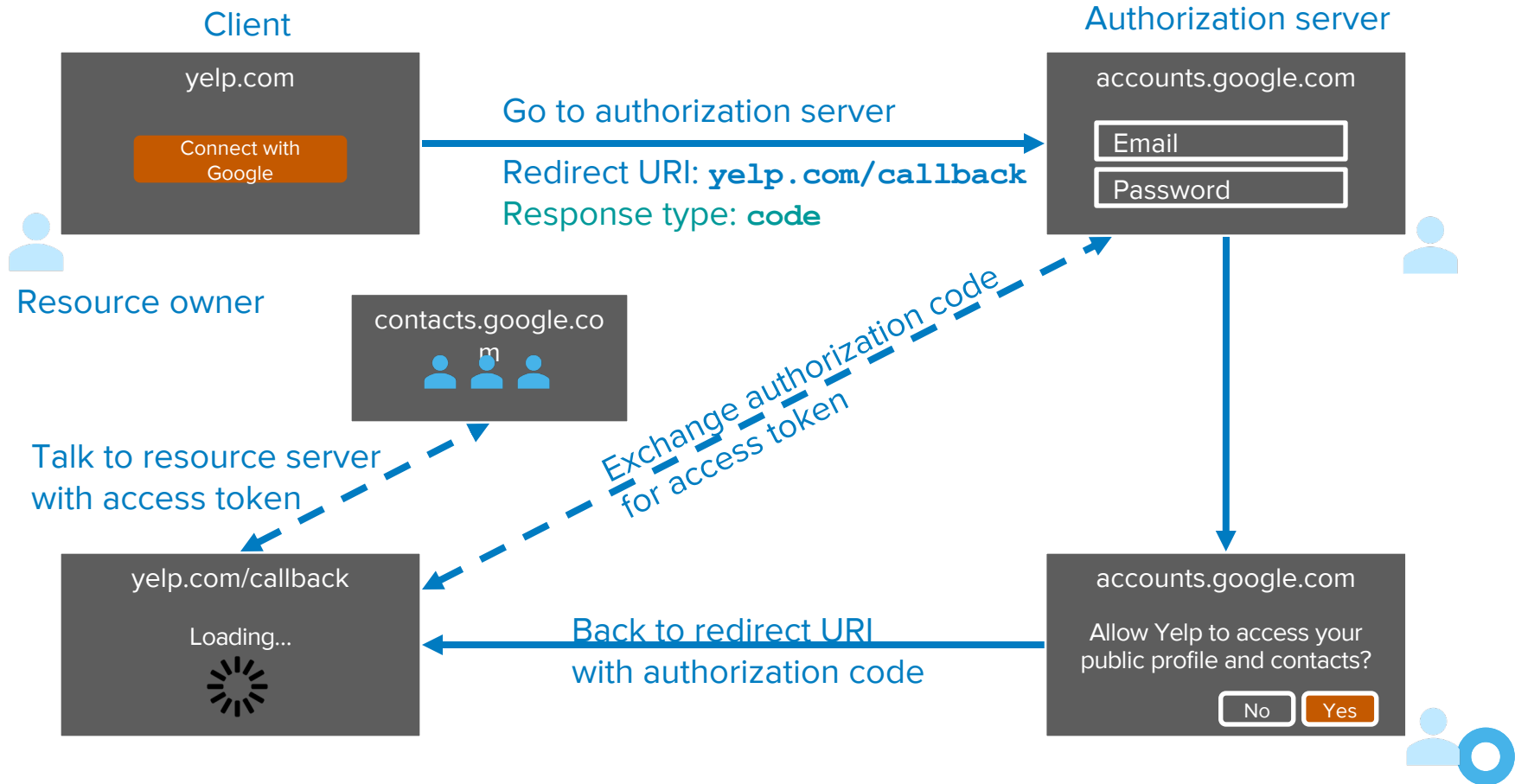


OAuth 2.0 terminology

- Resource owner
- Client
- Authorization server
- Resource server
- Authorization grant
- Access token



OAuth 2.0 authorization code flow

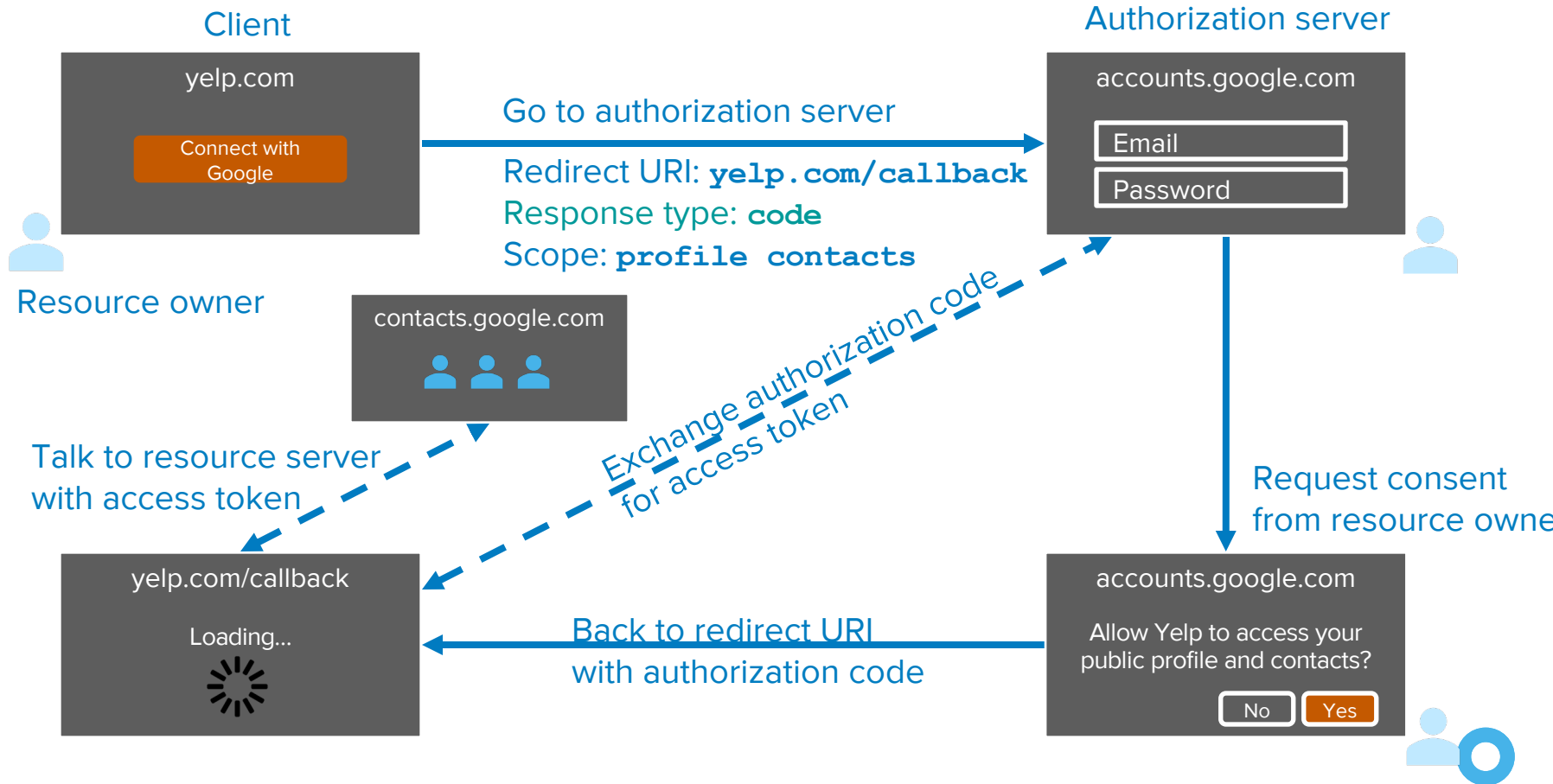


More OAuth 2.0 terminology

- Scope
- Consent



OAuth 2.0 authorization code flow



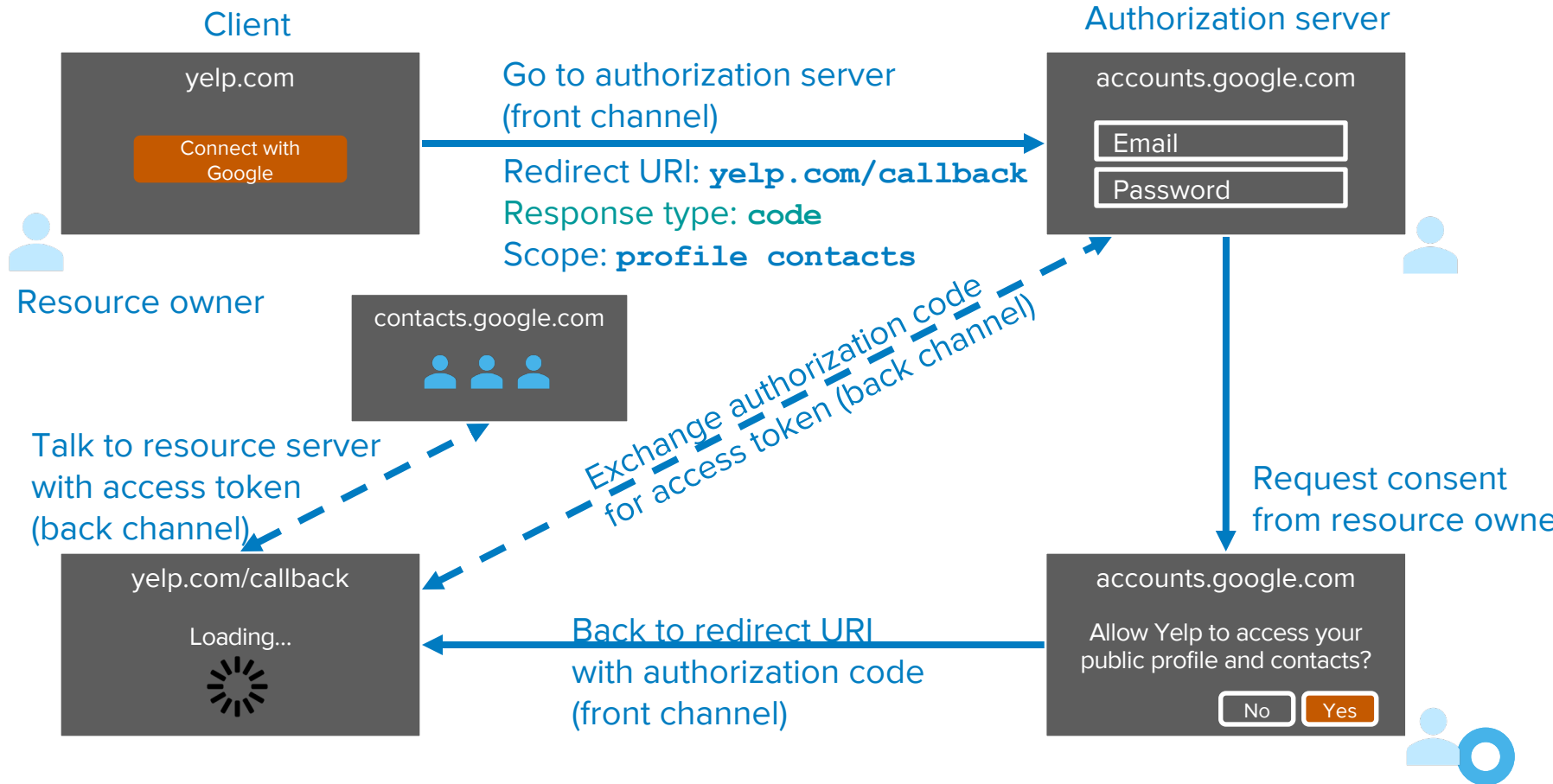
Even more OAuth 2.0 terminology

Back channel (highly secure channel)

Front channel (less secure channel)



OAuth 2.0 authorization code flow



Starting the flow

```
https://accounts.google.com/o/oauth2/v2/auth?  
  client_id=abc123&  
  redirect_uri=https://yelp.com/callback&  
  scope=profile&  
  response_type=code&  
  state=foobar
```



Calling back

```
https://yelp.com/callback?  
error=access_denied&  
error_description=The user did not consent.
```

```
https://yelp.com/callback?  
code=oMsCeLvIaQm6bTrgtp7&  
state=foobar
```



Exchange code for an access token

POST `www.googleapis.com/oauth2/v4/token`
Content-Type: `application/x-www-form-urlencoded`

`code=oMsCeLvIaQm6bTrgtp7&`
`client_id=abc123&`
`client_secret=secret123&`
`grant_type=authorization_code`



Authorization server returns an access token

```
{  
  "access_token": "fFAGRNJru1FTz70BzhT3Zg",  
  "expires_in": 3920,  
  "token_type": "Bearer",  
}
```



Use the access token

`GET api.google.com/some/endpoint`

`Authorization: Bearer fFAGRNJru1FTz70BzhT3Zg`



- Validate token
- Use token scope for authorization

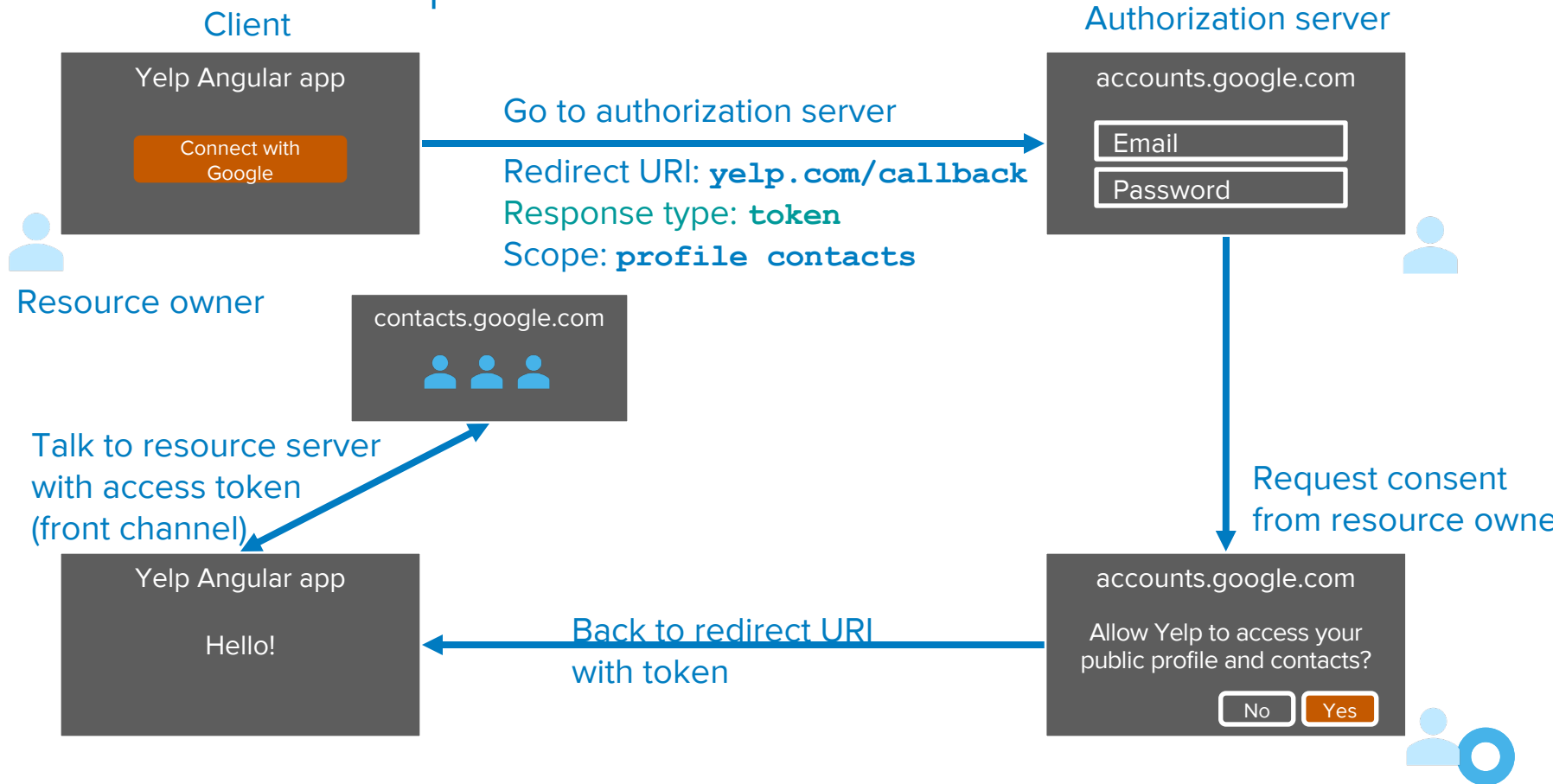


OAuth 2.0 flows

- Authorization code (front channel + back channel)
- Implicit (front channel only)
- Resource owner password credentials (back channel only)
- Client credentials (back channel only)



OAuth 2.0 implicit flow



Identity use cases (circa 2012)

Simple login	OAuth 2.0	Authentication
Single sign-on across sites	OAuth 2.0	Authentication
Mobile app login	OAuth 2.0	Authentication
Delegated authorization	OAuth 2.0	Authorization

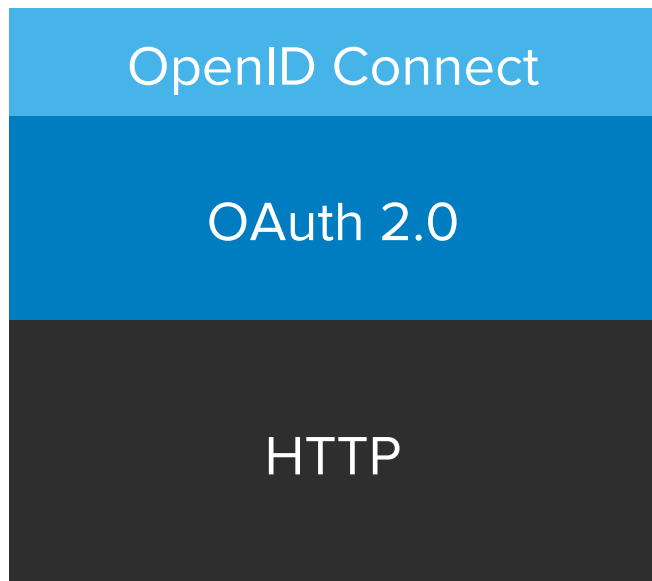


Problems with OAuth 2.0 for **authentication**

- No standard way to get the user's information
- Every implementation is a little different
- No common set of scopes



OAuth 2.0 and OpenID Connect



OpenID Connect is for
authentication

OAuth 2.0 is for
authorization

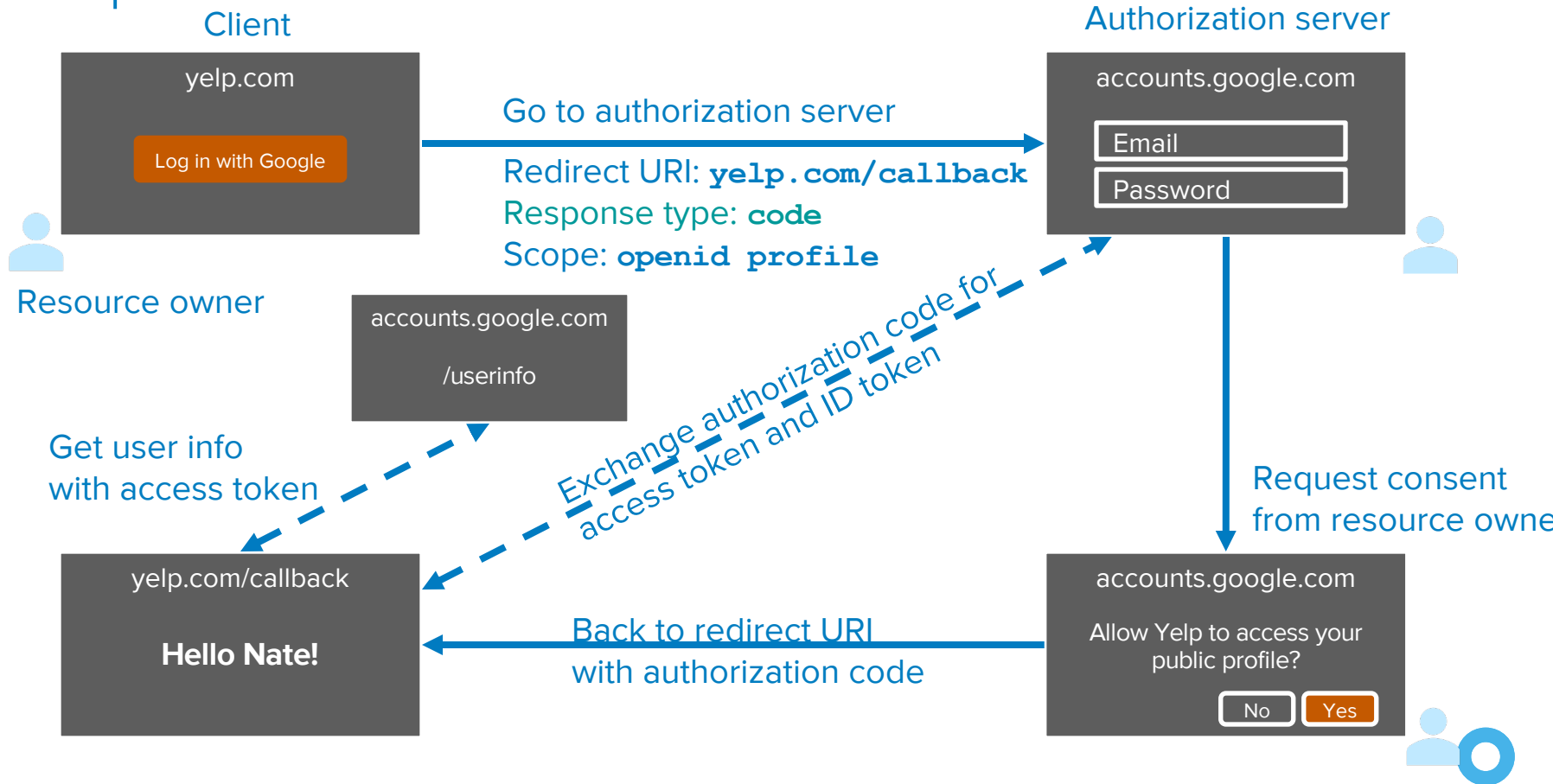


What OpenID Connect adds

- ID token
- UserInfo endpoint for getting more user information
- Standard set of scopes
- Standardized implementation



OpenID Connect authorization code flow



Starting the flow

```
https://accounts.google.com/o/oauth2/v2/auth?  
  client_id=abc123&  
  redirect_uri=https://yelp.com/callback&  
  scope=openid profile&  
  response_type=code&  
  state=foobar
```



Exchange code for access token *and* ID token

POST `www.googleapis.com/oauth2/v4/token`
Content-Type: `application/x-www-form-urlencoded`

`code=oMsCeLvIaQm6bTrgtp7&`
`client_id=abc123&`
`client_secret=secret123&`
`grant_type=authorization_code`



Authorization server returns access and ID tokens

```
{  
  "access_token": "fFAGRNJru1FTz70BzhT3Zg",  
  "id_token": "eyJraB03ds3F...",  
  "expires_in": 3920,  
  "token_type": "Bearer",  
}
```



ID token (JWT)

eyJhbGciOiJSUzI1NiIsImtpZCI6Ikp0RU1EYnh0Y25xaVJISVhuYUxubWI3UUUpfWF9rWmJyaEtBMGMifQ

Header

eyJzdWIiOiIwMHU5bzFuaWtqdK9CZzVabzBoNyIsInZlciI6MSwiaXNzIjoiaHR0cHM6Ly9kZXYtMzQxNjA3Lm9rdGFwcmV2aWV3LmNvbS9vYXV0aDIvYXVzOW84d3ZraG9ja3c5VEwwaDciLCJhdWQiOiJswFNlbkx4eFBpOGtRVmpKRTVzNCIsIm1hdCI6MTUwOTA0Tg5OCwiZXhwIjoxNTA5MDUzNDk4LCJqdGkiOiJJRC5oa2RXSXNBSXZTbnBGYVFHTVRYUGNVSmhhMkgwS2c5Yk13ZEVvVm1ZZHN3IiwiaW1yIjpbImtiYSIsIm1mYSIsInB3ZCJdLCJpZHAiOiIwMG85bzFuaWpraWpLeGNpbjBoNyIsIm5vbmNlIjoidWpwMmFzeHlqN2UiLCJhdXRoX3RpbWUiOiJE1MDkwNDk3MT19

Payload
(claims)

dv4Ek8B4BDee1PcQT_4zm7kxDEY1sRIGbLoNtlodZcSzHz-XU5GkKy16sAVmdXOIPU1AIrJAHnFQWQ-_XZLBVPjETiZE8CgNg5uqNmeXMUnYnQmvN5oWlXUZ8Gcub-GAbJ8-NQuyBmyec1j3gmGzX3wemke8NkuI6SX2L4Wj1PyvkknBtbjfiF9ud1-ERKbobaFbnjDF0FTzvL6g34SpMmZWY6uc_Hs--n4IC-ex-_Ps3FcMwRggCW_-7o2FpH6rJT0GPZYrOx44n3ZwAu2dGm6axtPI-sqU8b6sw7DaHpogD_hxsXgMIOzOBMbYsQEiczoGn71ZFz_107FiW4dH6g

Signature



The ID token (JWT)

(Header)

```
.  
{  
  "iss": "https://accounts.google.com",  
  "sub": "nate.barbettini@example.com",  
  "name": "Nate Barbettini"  
  "aud": "s6BhdRkqt3",  
  "exp": 1311281970,  
  "iat": 1311280970,  
  "auth_time": 1311280969,  
}
```

.
(Signature)



Calling the userinfo endpoint

```
GET www.googleapis.com/oauth2/v4/userinfo  
Authorization: Bearer fFAGRNJru1FTz70BzhT3Zg
```

```
200 OK
```

```
Content-Type: application/json
```

```
{  
  "sub": "nate.barbettini@example.com",  
  "name": "Nate Barbettini"  
  "profile_picture": "http://plus.g.co/123"  
}
```



Identity use cases (today)

Simple login

OpenID Connect

Authentication

Single sign-on across sites

OpenID Connect

Authentication

Mobile app login

OpenID Connect

Authentication

Delegated authorization

OAuth 2.0

Authorization



OAuth and OpenID Connect

Use **OAuth 2.0** for:

- Granting access to your API
- Getting access to user data in other systems

(Authorization)

Use **OpenID Connect** for:

- Logging the user in
- Making your accounts available in other systems

(Authentication)



Which flow (grant type) do I use?

Web application with a server backend: **authorization code flow**

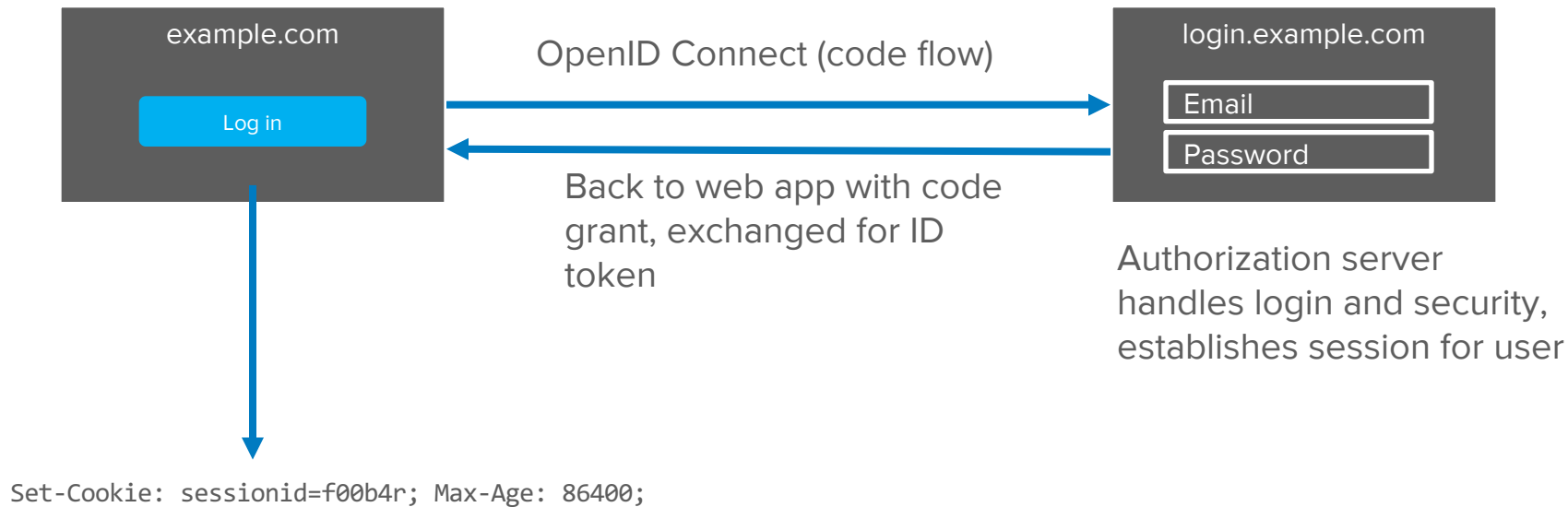
Native mobile app: **authorization code flow with PKCE**

JavaScript app (SPA) with an API backend: **implicit flow**

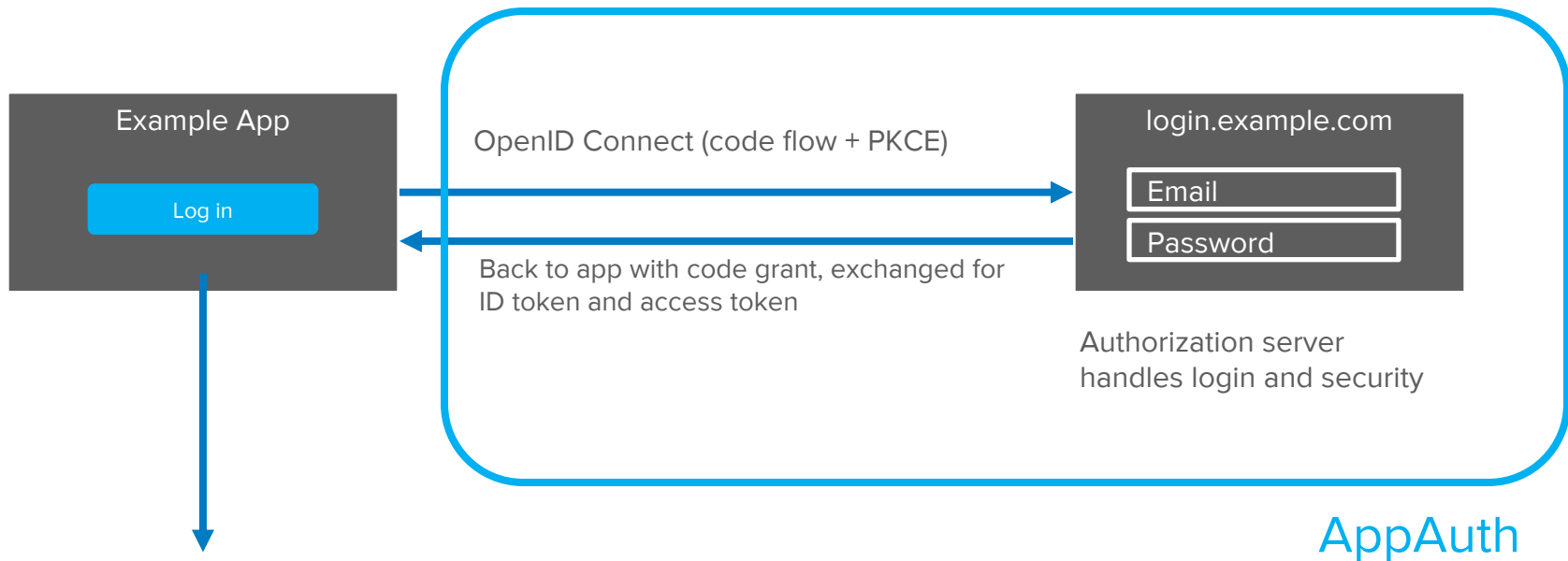
Microservices and APIs: **client credentials flow**



Example: web application with server backend



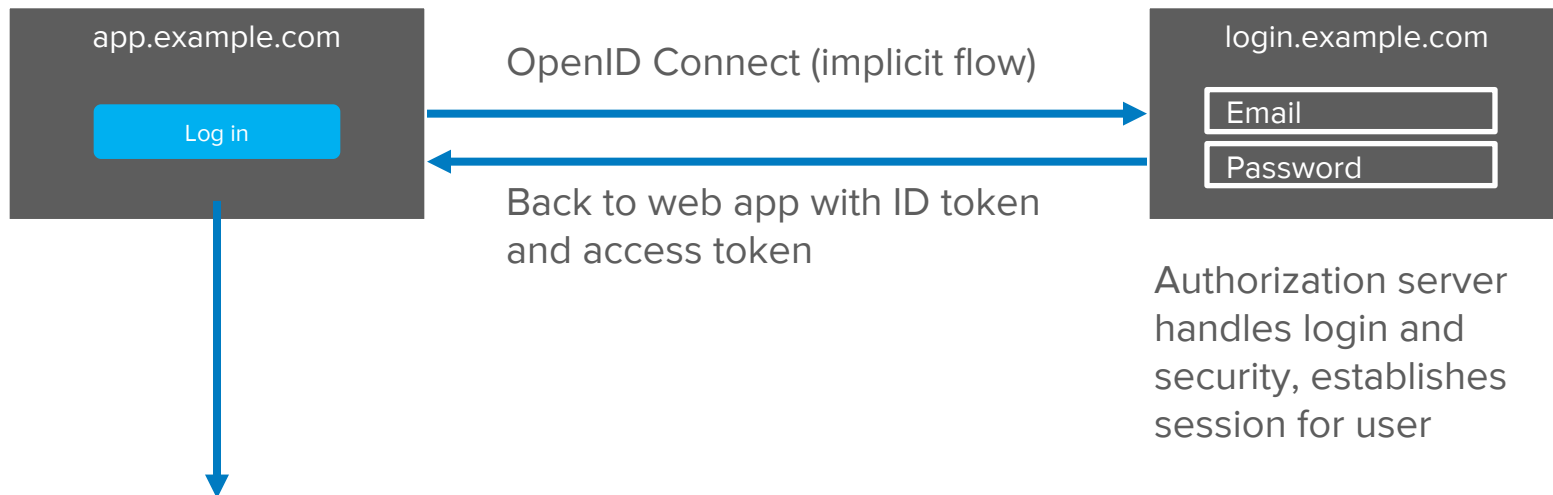
Example: native mobile app



- Store tokens in protected device storage
- Use ID token to know who the user is
- Attach access token to outgoing API requests



Example: SPA with API backend

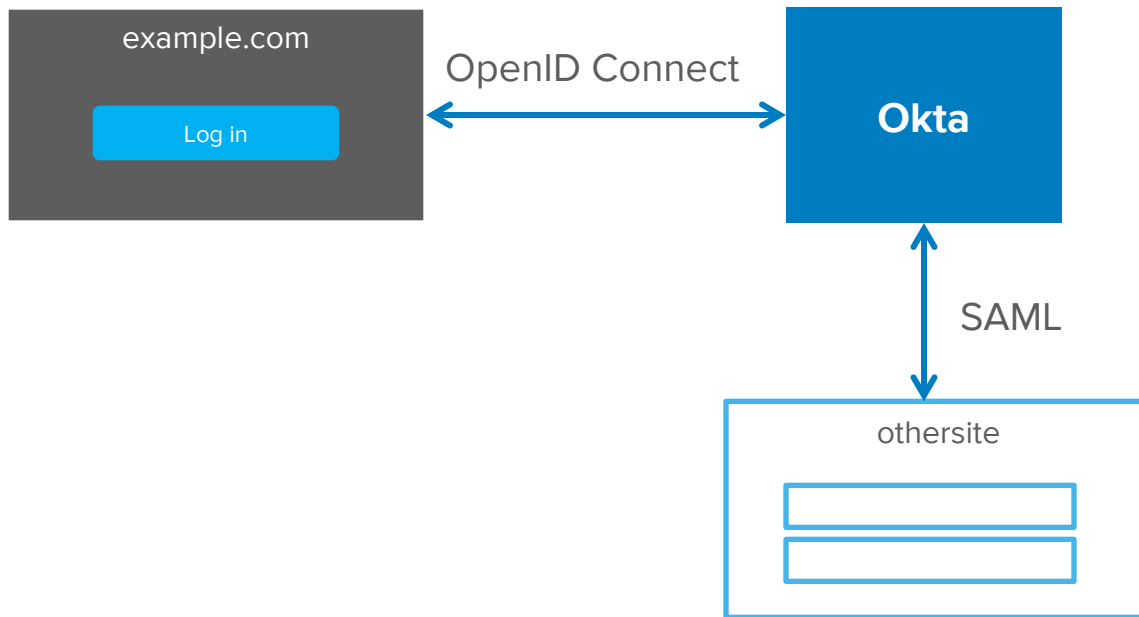


Authorization server handles login and security, establishes session for user

- Store tokens locally with JavaScript
- Use ID token to know who the user is
- Attach access token to outgoing API requests



Example: SSO with 3rd-party services



Token validation

- The fast way: local validation
 - Check expiration timestamp
 - Validate cryptographic signature
 - Validate claims
 - etc
- The strong way: introspection



Revocation

Token issued and used for
API calls

Device compromised!

What happens?

12PM

1PM

2PM

```
POST /oauth2/default/v1/revoke
Content-Type: application/x-www-form-urlencoded
```

```
token=fFAGRNJru1FTz70BzhT3Zg
&token_type_hint=access_token
&client_id=...
```



Keeping the user signed in

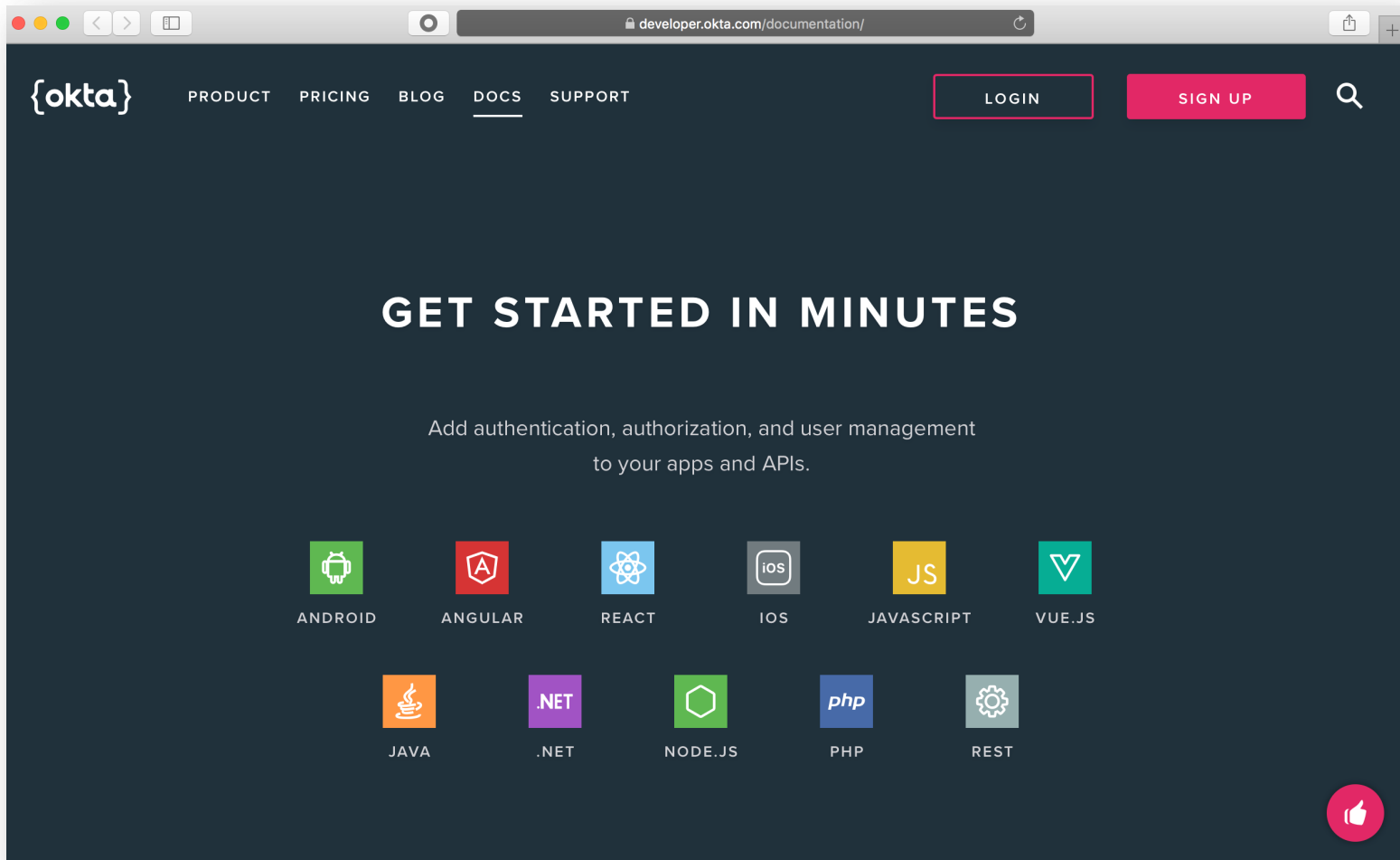
For both local validation and introspection, the token is invalid once it expires, so:

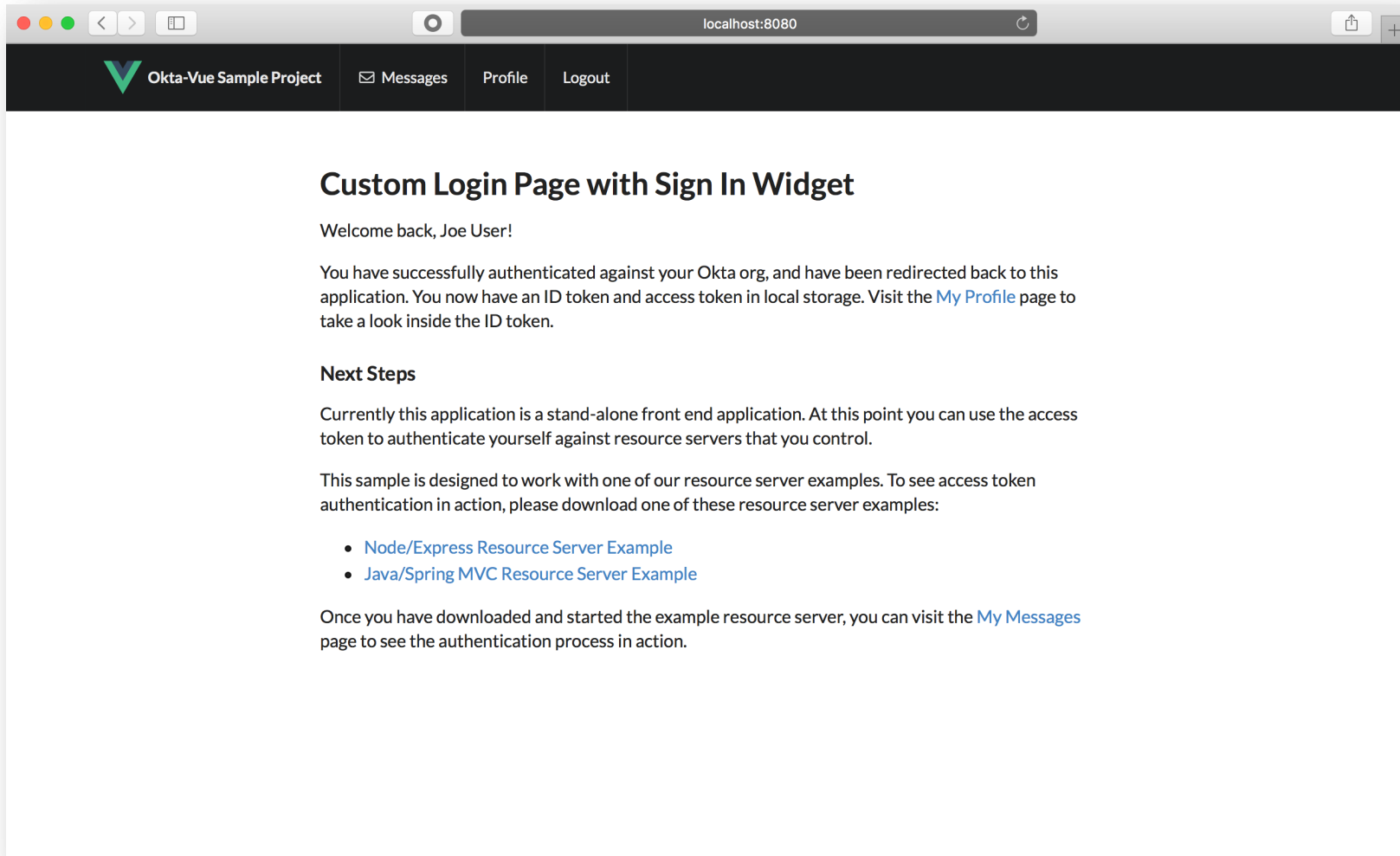
- If there's a user at the keyboard, just redirect through the authorization server again.
- If there's no user (automated tasks), request a refresh token (offline scope).

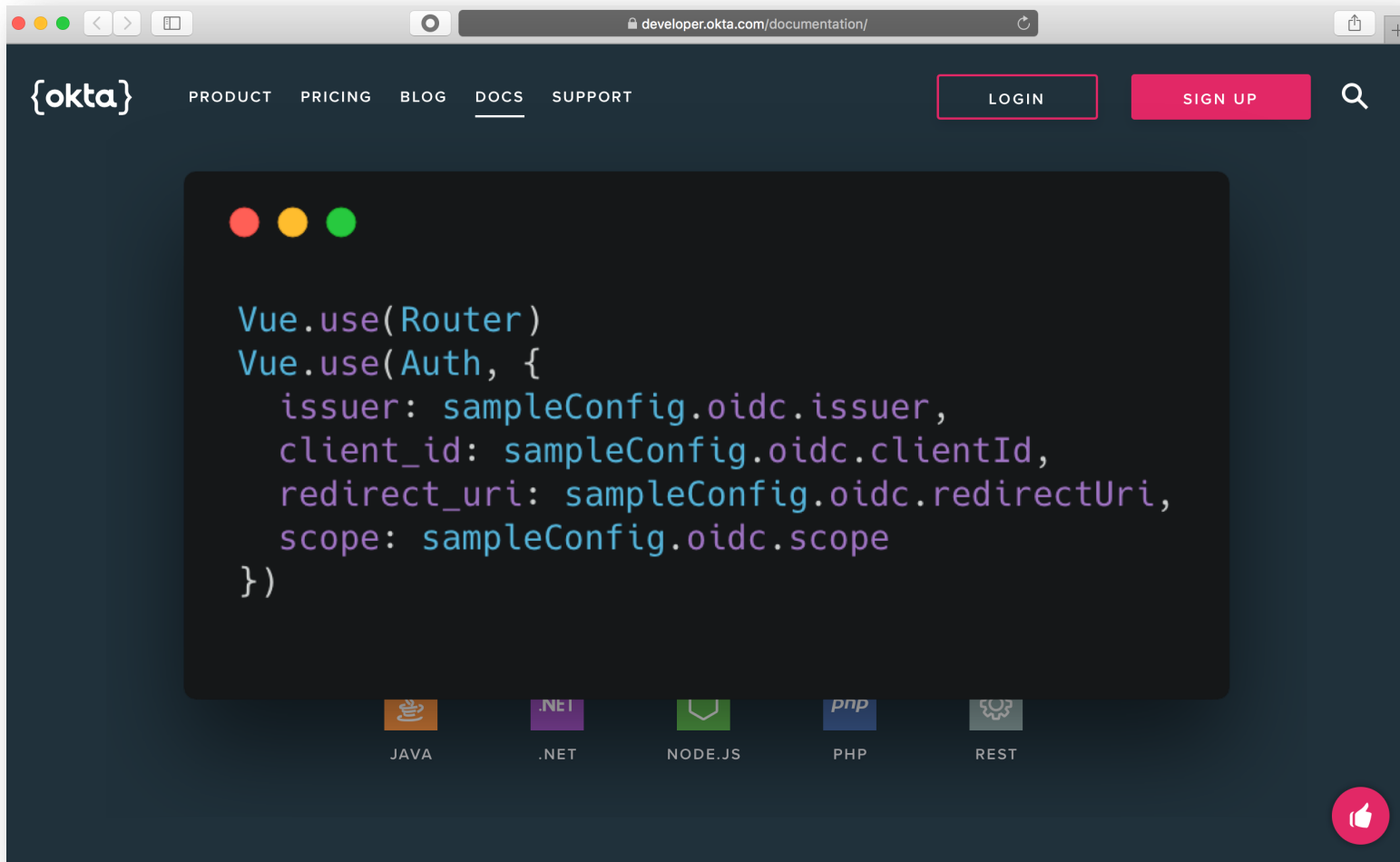


Implementing OAuth and OpenID Connect









The image shows a web browser window displaying the Okta developer documentation page. The browser's address bar shows the URL `developer.okta.com/documentation/`. The page has a dark blue header with the Okta logo on the left and navigation links for `PRODUCT`, `PRICING`, `BLOG`, `DOCS` (which is underlined), and `SUPPORT`. On the right side of the header, there are `LOGIN` and `SIGN UP` buttons, and a search icon. The main content area features a dark blue background with a large, semi-transparent dark rectangle in the center. Inside this rectangle, there is a code snippet for Vue.js authentication configuration, preceded by three colored circles (red, yellow, green). The code is as follows:

```
Vue.use(Router)
Vue.use(Auth, {
  issuer: sampleConfig.oidc.issuer,
  client_id: sampleConfig.oidc.clientId,
  redirect_uri: sampleConfig.oidc.redirectUri,
  scope: sampleConfig.oidc.scope
})
```

Below the code snippet, there is a horizontal row of five technology logos: `JAVA`, `.NET`, `NODE.JS`, `PHP`, and `REST`. In the bottom right corner of the page, there is a red circular button with a white thumbs-up icon.

My Messages

This component makes a GET request to the resource server example, which must be running at `localhost:8000/api/messages`

It attaches your current access token in the `Authorization` header on the request, and the resource server will attempt to authenticate this access token. If the token is valid the server will return a list of messages. If the token is not valid or the resource server is incorrectly configured, you will see a `401 Unauthorized` response.

This route is protected with the `onAuthRequired` navigation guard, which will ensure that this page cannot be accessed until you have authenticated and have an access token in local storage.

Date	Message
10/15/2018 5:18:57 PM	I am a robot.
10/15/2018 4:18:57 PM	Hello, world!



PRODUCT

SIGN UP



```
function authenticationRequired(req, res, next) {
  const authHeader = req.headers.authorization || '';
  const match = authHeader.match(/Bearer (.+)/);

  if (!match) {
    res.status(401);
    return next('Unauthorized');
  }

  const accessToken = match[1];

  return oktaJwtVerifier.verifyAccessToken(accessToken)
    .then((jwt) => {
      req.jwt = jwt;
      next();
    })
    .catch((err) => {
      res.status(401).send(err.message);
    });
}

app.get('/secure', authenticationRequired, (req, res) => {
  res.json(req.jwt);
});
```





```
app.get('/api/messages', authenticationRequired, (req, res) => {  
  res.json({  
    messages: [  
      {  
        date: new Date(),  
        text: 'I am a robot.'  
      },  
      {  
        date: new Date(new Date().getTime() - 1000 * 60 * 60),  
        text: 'Hello, world!'  
      }  
    ]  
  });  
});
```

Additional Resources

- <https://oauth.com>
- <https://oauth.com/playground>
- YouTube: “OAuth in plain English”
- Google: “Using OAuth and OpenID Connect in your applications”
- <https://developer.okta.com/documentation/>



The Okta logo is displayed in a white, lowercase, sans-serif font. The letters are bold and rounded, with a clean, modern aesthetic. The 'o' is a simple circle, the 'k' has a straight vertical stem and a short horizontal bar, the 't' has a thin vertical stem and a wide, flat top, and the 'a' is a simple, rounded shape. The logo is centered horizontally and vertically on the blue background.

okta

Thank you!