

IN4MATX 133: User Interface Software

Lecture 9:
Server-Side Development,
Authentication, & Authorization

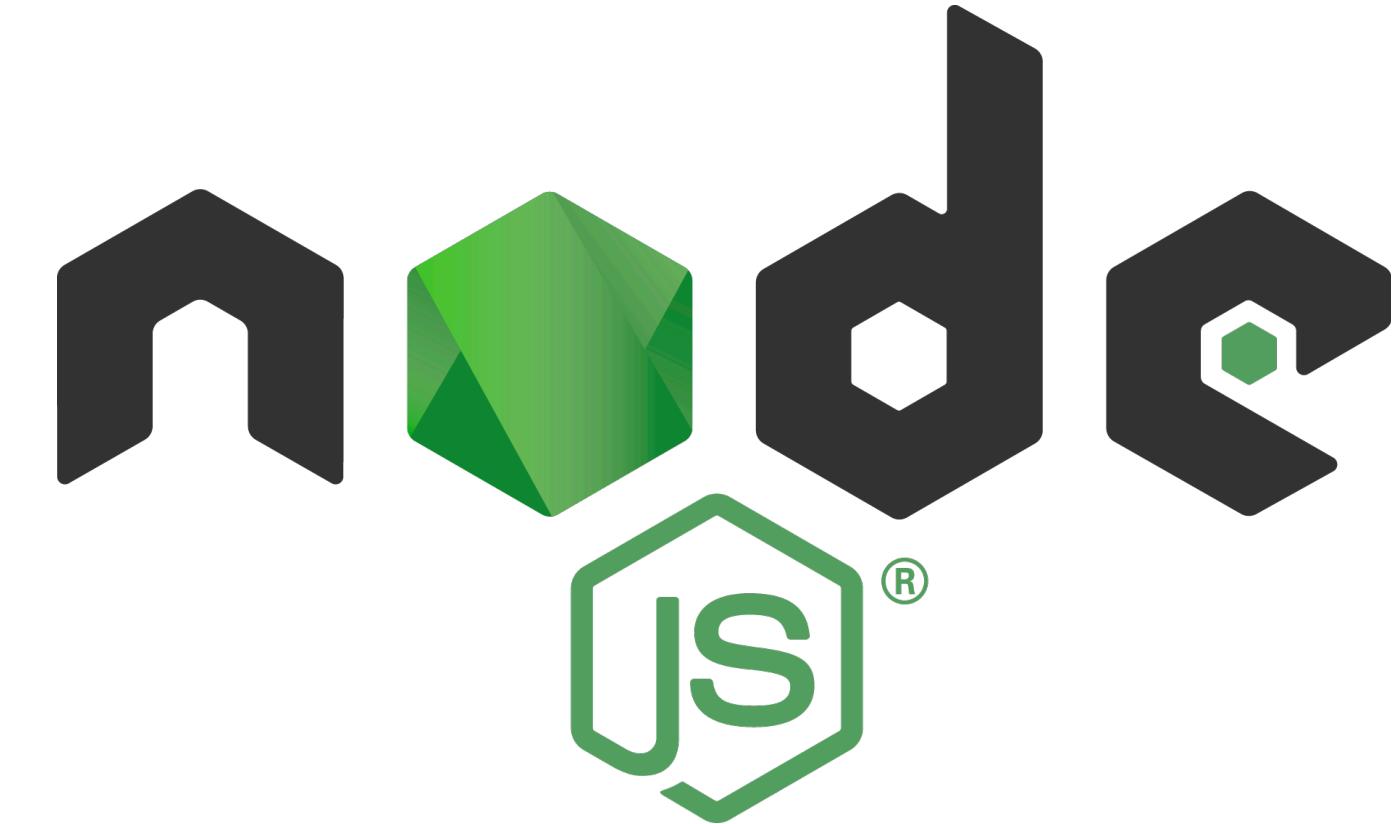
Today's goals

By the end of today, you should be able to...

- Explain the advantages and disadvantages of different tools for server-side development
- Differentiate authentication from authorization
- Describe the utility of supporting authentication and authorization in interfaces
- Explain and implement the different stages to authenticating via OAuth
- Describe the advantages and disadvantages of OpenId

Server-side development: Node.js

- Event-driven, non-blocking I/O model makes it efficient
- Best for highly-interactive pages
 - When a lot of computation is required, other frameworks are better
 - Event-driven loops are inefficient
- Lower threshold for us:
we're already learning JavaScript!



Other server-side environments

- Ruby, via Ruby on Rails
- Python, via Django or web2py
- These days, you can
create a dynamic website
in almost any language



Node package manager (npm)

- Included in the download of Node
- Originally libraries specifically for Node
- Now includes many JavaScript packages



Node.js hello world

```
var http = require('http'); ←Require the http library
```

Node.js hello world

```
var http = require('http'); //←Require the http library
var server = http.createServer(function(req, res) {
});
```

↑
Anonymous function with
request and response parameters

Node.js hello world

```
var http = require('http'); //←Require the http library
var server = http.createServer(function(req, res) {
  res.writeHead(200); //↑
  res.end('Hello World');
}); //↑
// "Ok" status in the header,
// write hello world text
```

Anonymous function with
request and response parameters

Node.js hello world

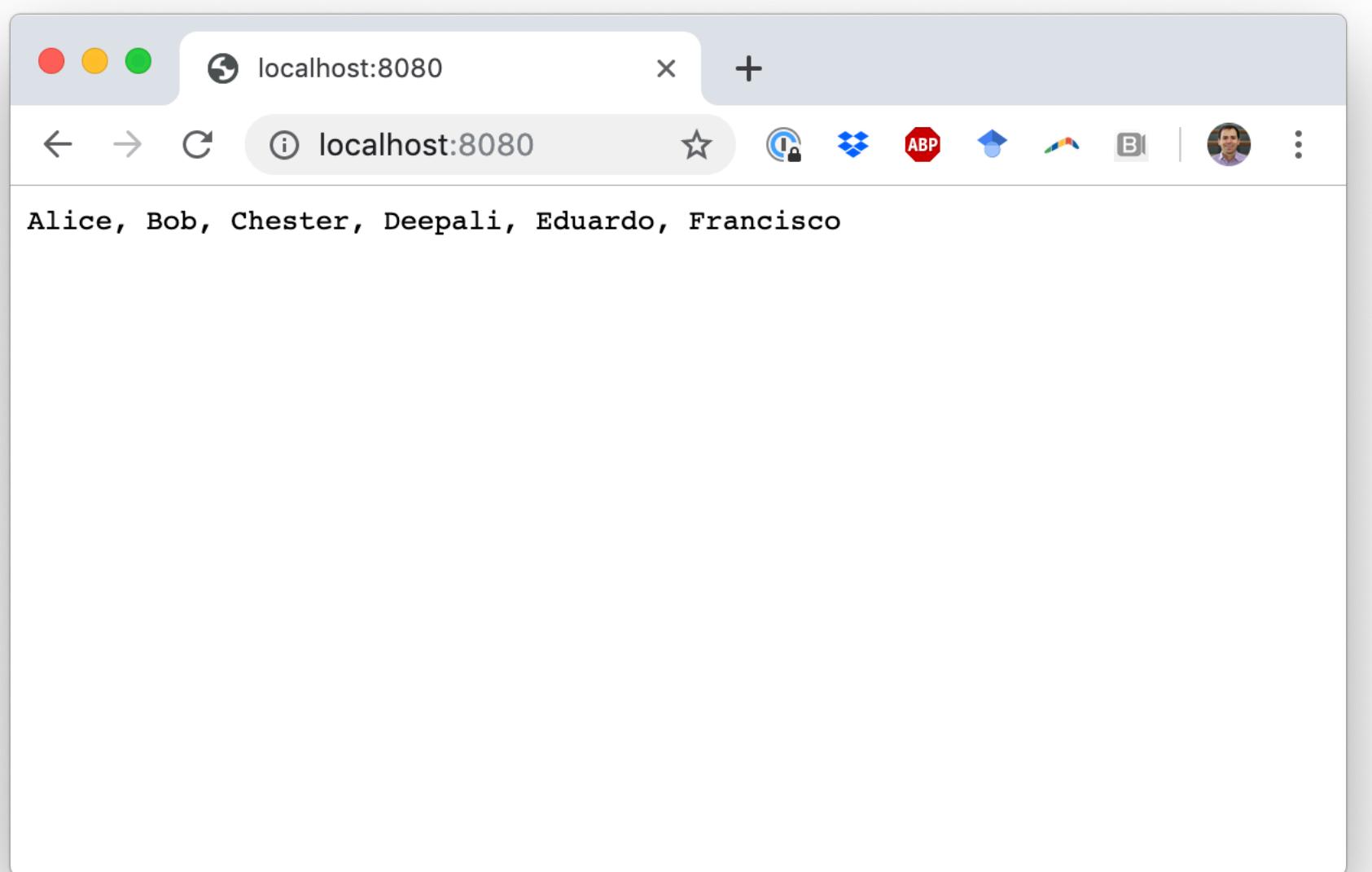
```
var http = require('http'); //←Require the http library
var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Hello World');
}); //↑ “Ok” status in the header,
server.listen(8080); //↑ write hello world text
Listen on port 8080
```

Anonymous function with request and response parameters

Running Node.js

- node file.js

Node.js



**Remember,
Node.js is server-side JavaScript**

Where is the JavaScript running?

Server-side

```
node hello.js
```

hello.js:

```
var http = require('http');
var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Hello World');
});
server.listen(8080);
console.log('Hello, console');
```

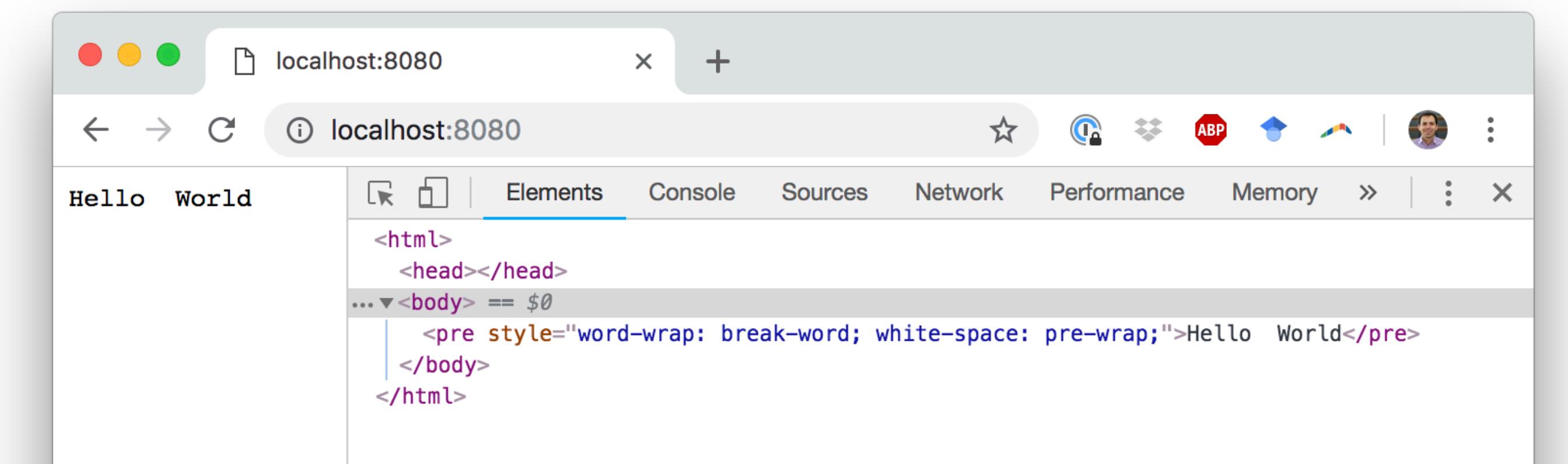
Node is listening on port 8080.

But the JavaScript is not
running in the browser.

It's running in the console.



A screenshot of a Mac OS X terminal window titled "l11_server_side_development — node hello.js — 96x15". The window shows the command "node hello.js" being run, and the output "Hello, console" appearing in the terminal.



Where is the JavaScript running?

Client-side

live-server

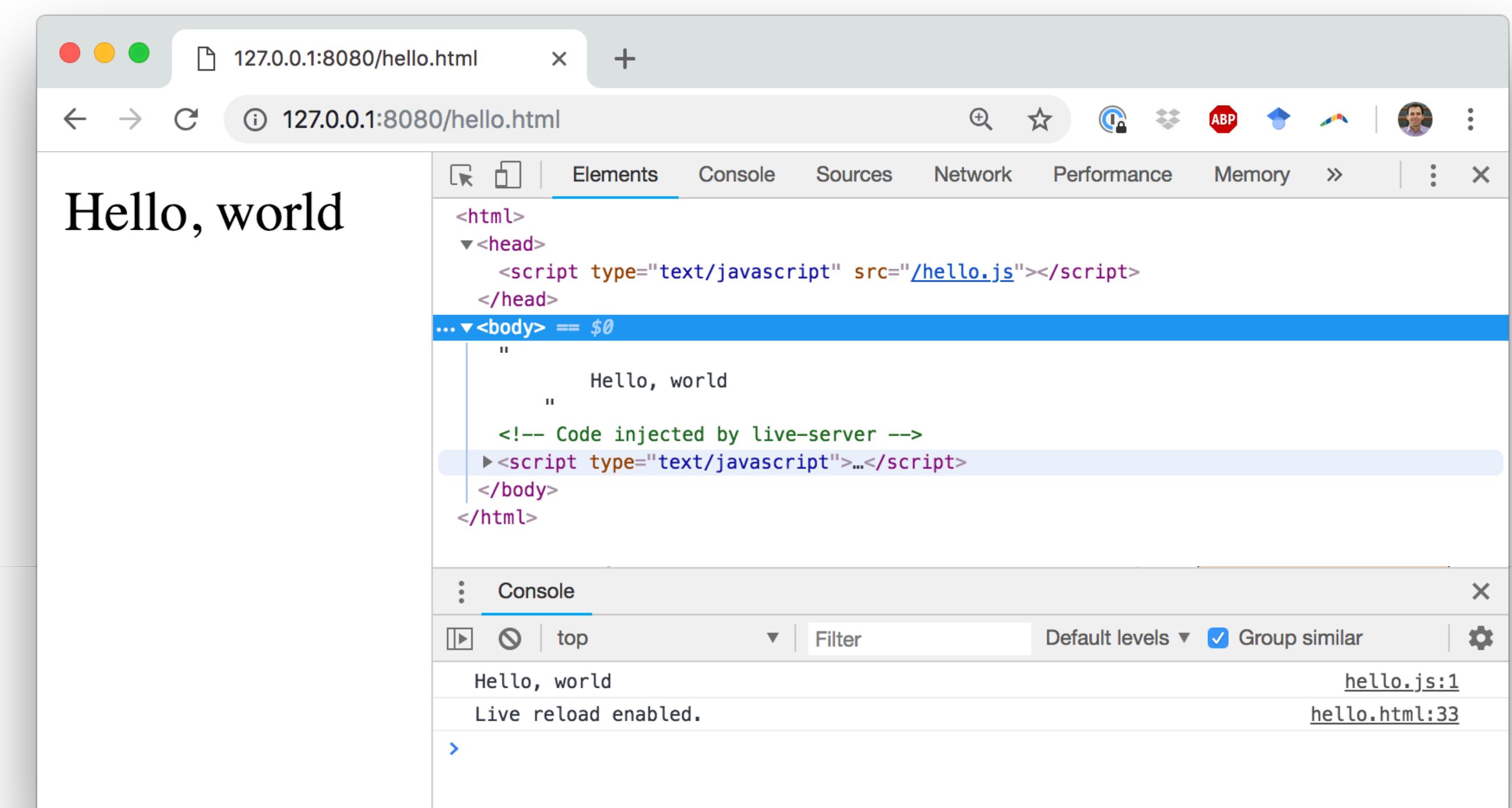
hello.html:

```
<html>
  <head>
    <script src="./hello.js"></script>
  </head>
  <body>
    Hello, world
  </body>
</html>
```

Live-server is listening on port 8080. The JavaScript is running in the browser.

hello.js:

```
console.log('Hello, world');
```



What does Node.js add?

- OS-level functionality like reading and writing files
- Tools for importing and managing packages
- The ability to listen on a port as a web server
- But it's just JavaScript, and it's pretty basic as a web framework

What does a “good” server-side web framework need?

- To speak in HTTP
 - Accept connections, handle requests, send replies
- Routing
 - Map URLs to the webserver function for that URL
- Middleware support
 - Add data processing layers
 - Make it easy to add support for user sessions, security, compression, etc.

What does a “good” server-side web framework need?

- Node.js has the capabilities of a “good” web framework
 - To speak in HTTP
 - Routing
 - Middleware support
- But they’re somewhat difficult to use.
- People have written extensions to Node (like Express) to make server-side development easier
 - This is why Node made a package manager (npm)!

Switching topics: authentication & authorization

What is authentication?

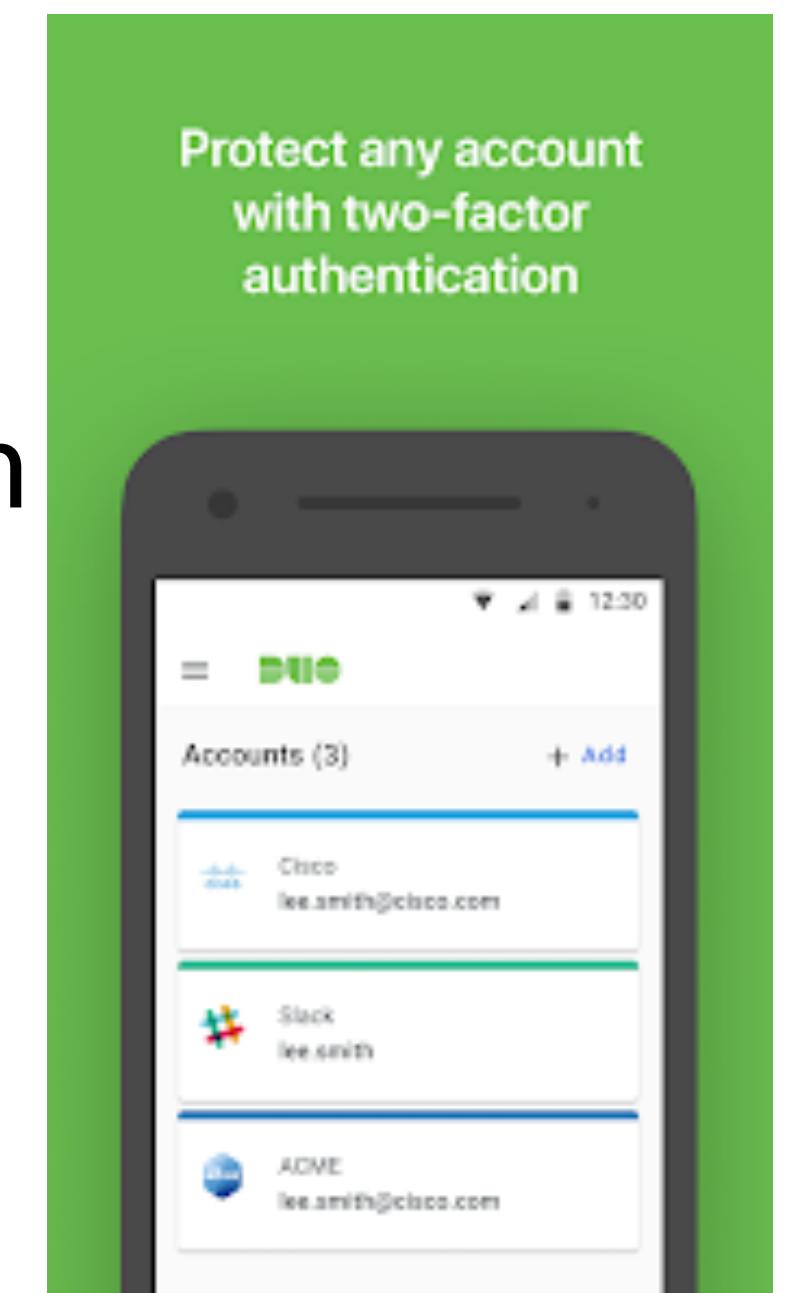
- The process of establishing and verifying identity
- Identification: who are you? (username, account number, etc.)
- Authentication: prove it! (password, PIN, etc.)

What is authorization?

- Once we know a user's identify, we must decide what they are allowed to access or modify
- One way is the app defines permissions upfront based on a user's role
 - A student can access their own grades, but not modify them
 - A TA and a professor can access and modify everyone's grades
- Another way is for the app to request the user grant certain permissions
 - A Twitter app may ask, "can I Tweet on your behalf?"

Multi-factor authentication

- Should be a mix of things that you *have/possess* and things that you *know*
- ATM machine: 2-factor authentication
 - ATM card: something you *have*
 - PIN: something you *know*
- Password + code delivered via SMS/Push: 2-factor authentication
 - Password: something you *know*
 - Code: validates that you *possess* your phone
- Two passwords != Two-factor authentication



Question



Which of these is an example of “good” two-factor authentication?

- A government agency requiring a birth certificate and a passport
- A store requiring a membership card and a PIN
- A website requiring a password and a security question
- Two of the above
- All of the above

A government agency requiring a birth certificate and a passport

0%

A store requiring a membership card and a PIN

0%

A website requiring a password and a security question

0%

Two of the above

0%

All of the above

0%

Question

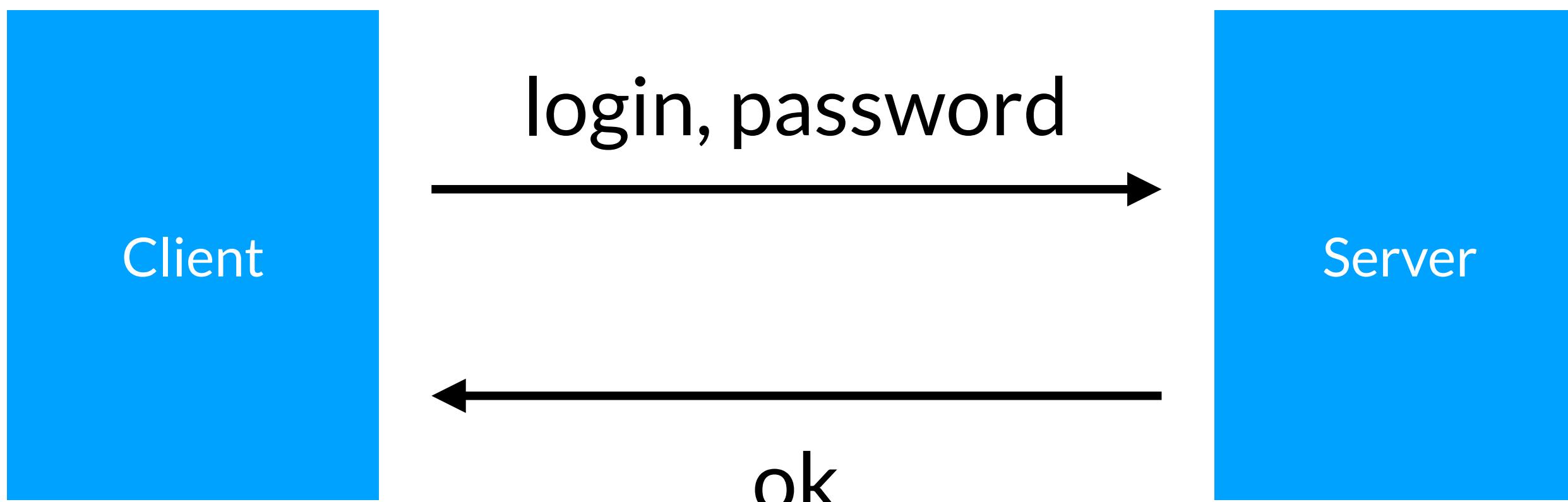


Which of these is an example of “good” two-factor authentication?

- A government agency requiring a birth certificate and a passport
- A store requiring a membership card and a PIN
- A website requiring a password and a security question
- Two of the above
- All of the above

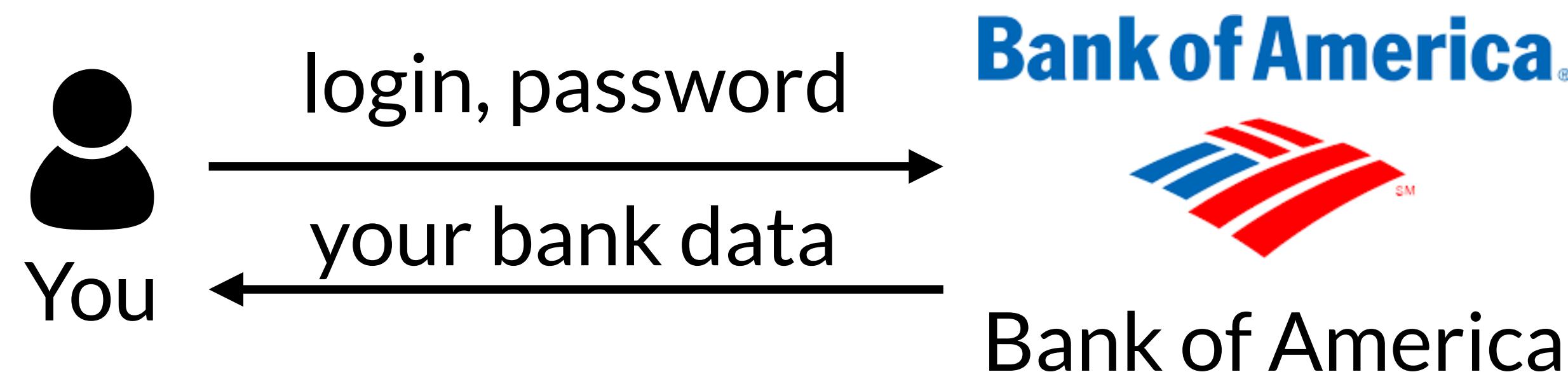
Password protocol

- Send a login and a password to a server
- Server checks your credentials and okays you
- Need to trust that the server is storing your password securely



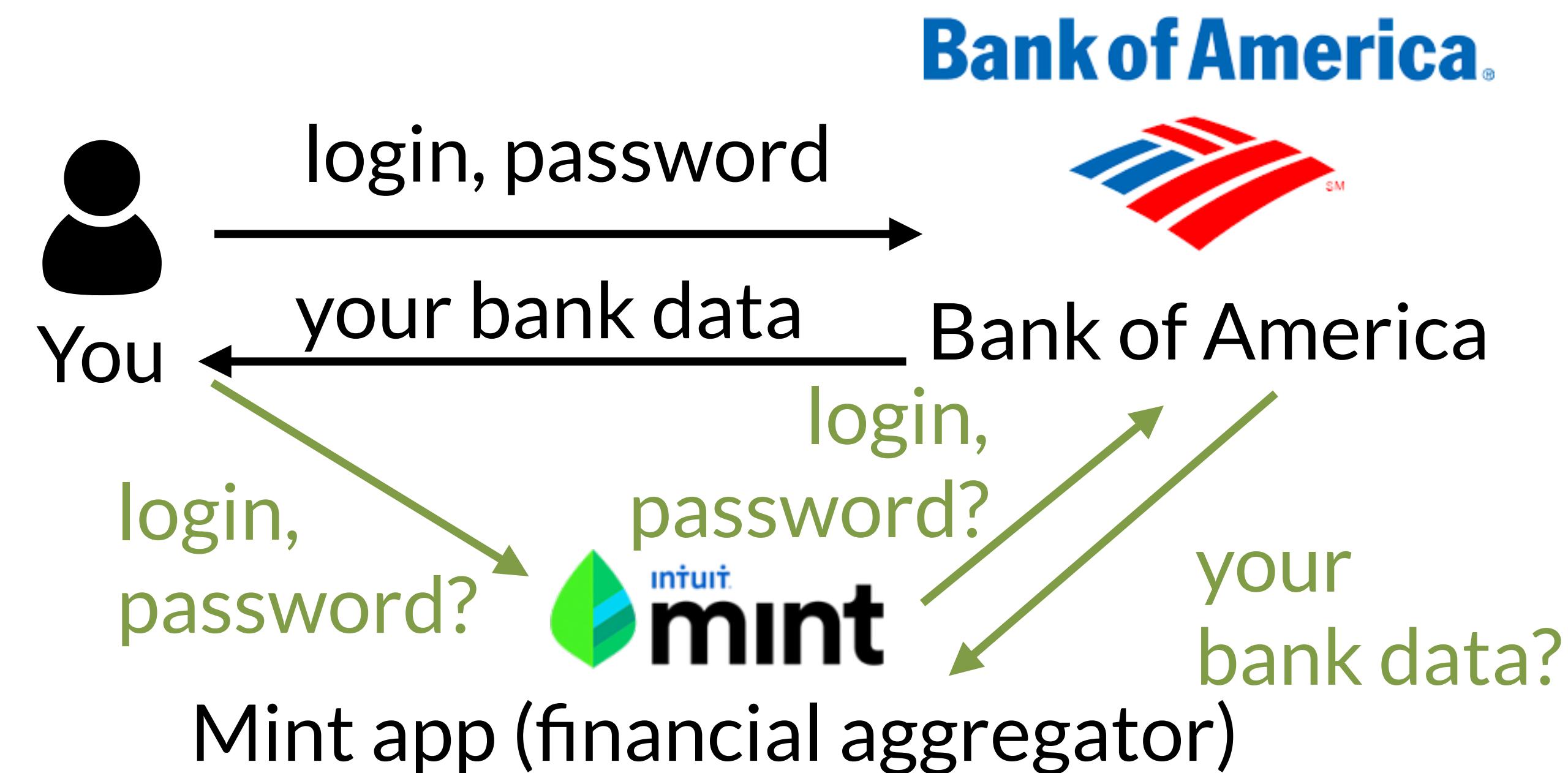
Password protocol: sending data

- Once you've logged in,
the server can send you whatever data you're allowed to see



Sending data to a third party

- You want to send data that a server has to a third party
 - You could give them your username and password...
 - Why is this a bad idea?



Sending data to a third party

- Now you have to trust *another* service to manage your password
- What if you don't want them to have full access?
 - e.g., you want Mint to load your savings account but not your checking account
- What if you want to revoke access later?
 - Can change your password, but that's not a good solution

Oauth 2.0

- Open authentication
- Goal: support users in granting access to third-party applications
 - Do not require users to share their passwords with the third-party applications
 - Allow users to revoke access from the third parties at any time

Oauth 2.0 history

- There was a 1.0
 - It was complex (worse than 2.0)
 - It had security vulnerabilities
 - It shouldn't be used anymore
- Google, Twitter, & Yahoo! teamed up to propose 2.0
- 2.0 is not compatible with 1.0

https://en.wikipedia.org/wiki/OAuth#OAuth_2.0

Oauth 2.0 terminology

- Client
 - Third-party app who wants to access resources owned by the *resource owner* (e.g., app you develop)
- Resource owner (user)
 - Person whose data is being accessed, which is stored on the *resource server*
- Resource server
 - App that stores the resources (e.g., Spotify, Google, Facebook)
- Authorization and Token endpoints
 - URIs from where a resource owner authorizes requests

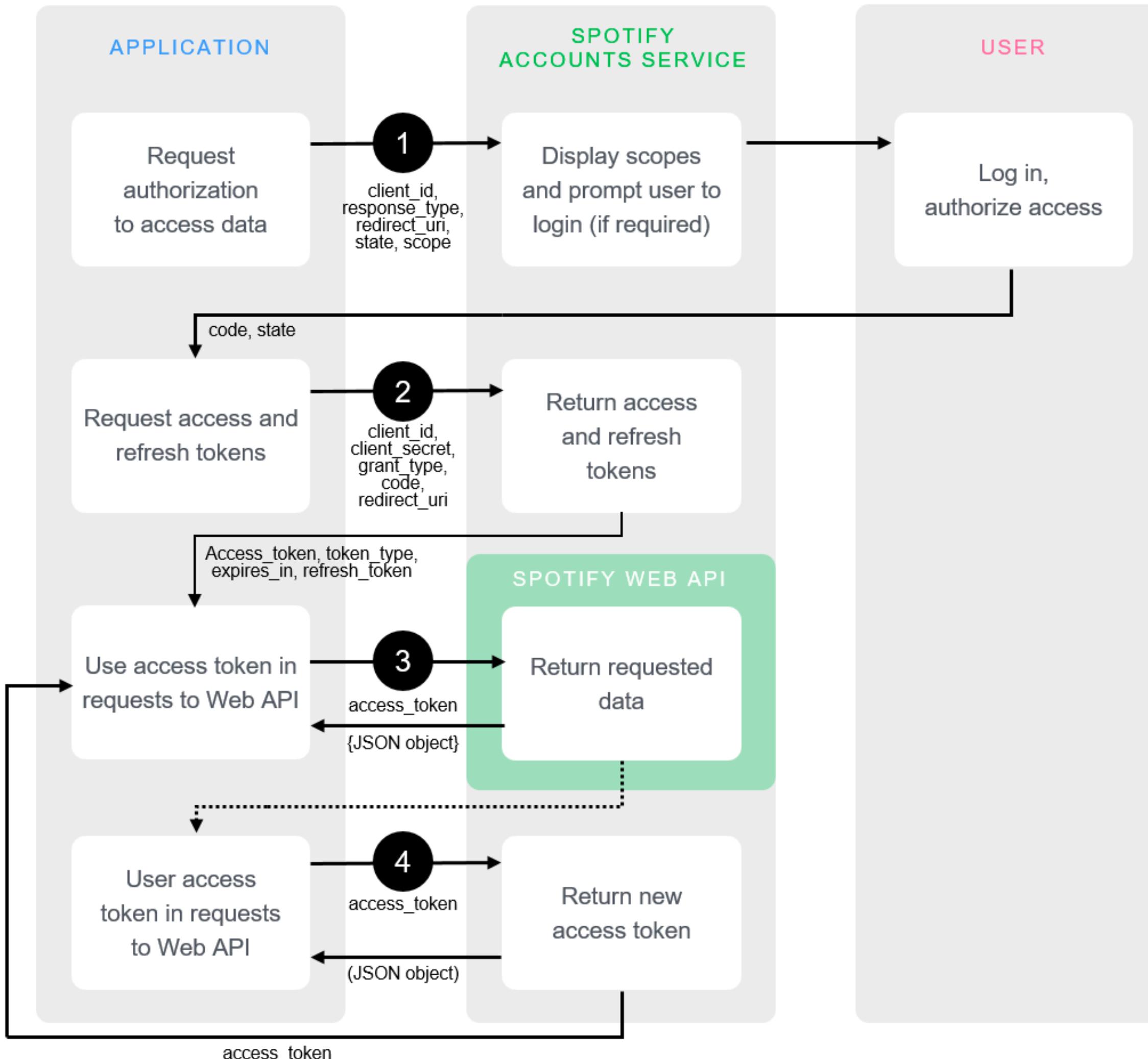
Oauth 2.0 terminology

- Authorization code
 - A string the client uses to request access tokens
- Access token
 - A string the client uses to access resources (e.g., songs on Spotify, Tweets, etc.)
 - Expires after some amount of time
- Refresh token
 - Once the access token expires, can be exchanged for a new access token

Oauth 2.0 steps

1. Request authorization
2. Get access token
3. Make API calls
4. Refresh access token

Oauth 2.0 steps



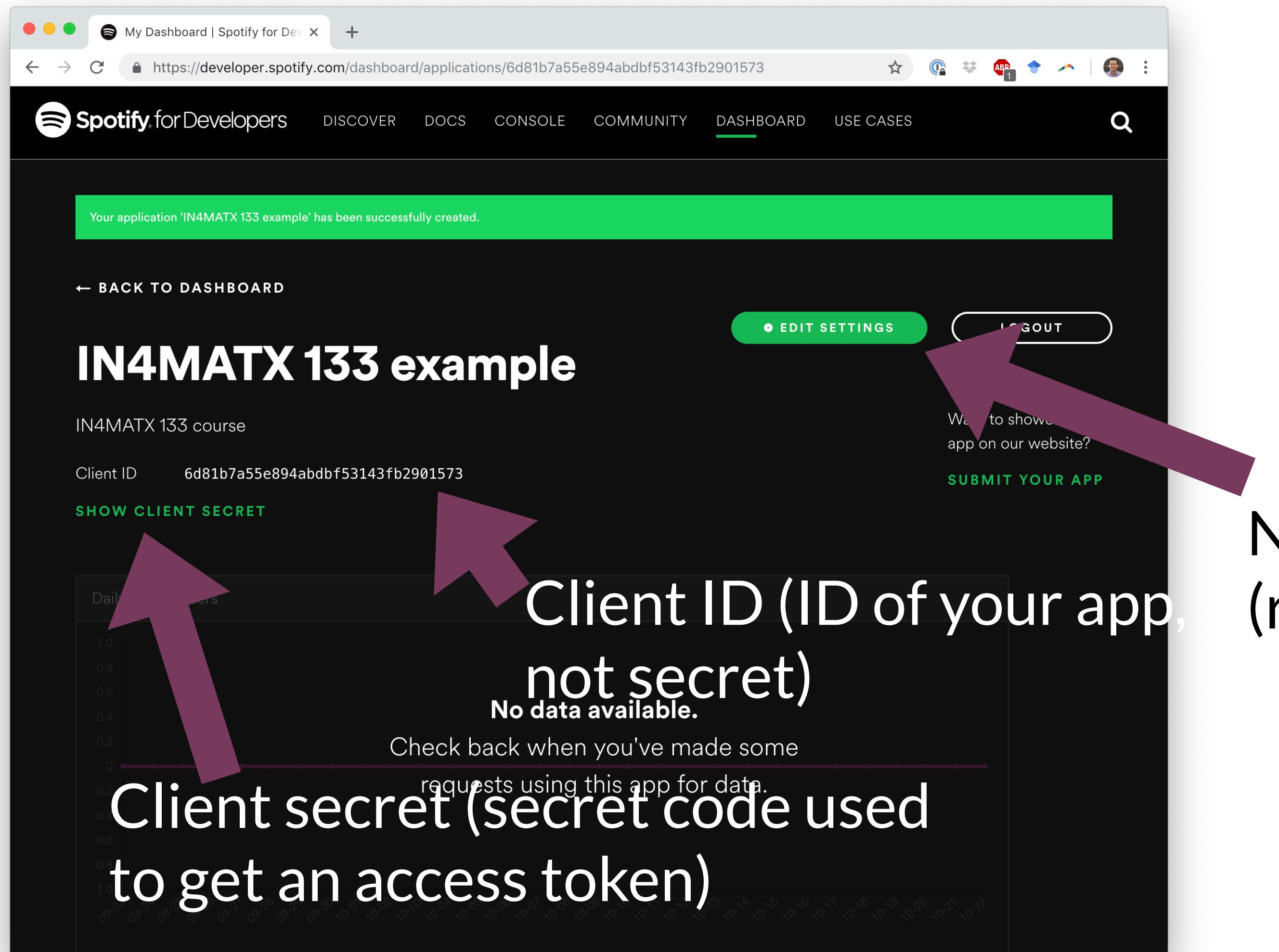
<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Oauth 2.0 and Spotify

The screenshot shows the Spotify for Developers Dashboard at <https://developer.spotify.com/dashboard/>. The page has a dark header with the Spotify logo, navigation links like DISCOVER, DOCS, CONSOLE, COMMUNITY, DASHBOARD (which is underlined), and USE CASES. Below the header is a large 'Dashboard' title. On the left, there's a green card for a client application named 'IN4MATX 133' with a placeholder Client ID: 'e9334de09e8842f7b0827dbb3ab3025e'. To the right of the card is a button labeled 'CREATE A CLIENT ID' with a purple arrow pointing to it. Below the button is a section titled 'My New App' with some placeholder data. At the bottom of the dashboard are links for DOCS, COMMUNITY, USE CASES, SUPPORT, and another USE CASES link.

<https://developer.spotify.com/dashboard/>

Oauth 2.0 and Spotify



<https://developer.spotify.com/dashboard/>

Need to specify what URI to return to
(redirect URI)

Oauth 2.0 on server-side JavaScript

- This example will walk through the Oauth flow for server-side JavaScript (like Node.js/Express)
- There are browser-side ways of doing (some parts of) Oauth
- For A3, you'll send all browser-side requests to a Node.js/Express server

Assignment 3

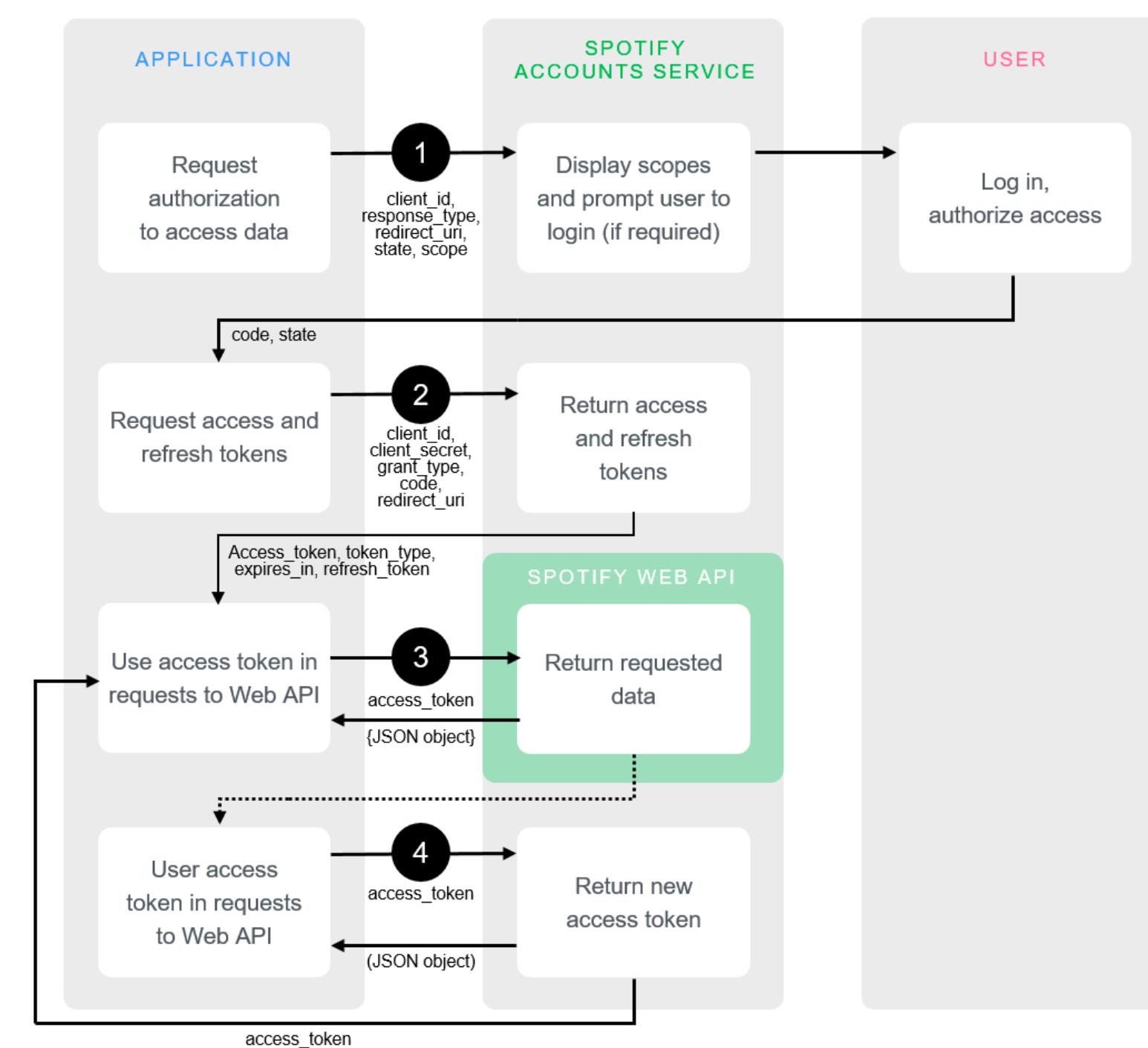


A screenshot of a web browser window titled "Spotify browser" showing a Spotify interface. The address bar indicates the page is at `localhost:4200`. The left side shows a user profile for "Logged in user: Daniel" with a photo and a button to "Open profile on Spotify". The right side shows a "Search Spotify" results page for "carly rae jepsen" set to "artist". The search result features a photo of Carly Rae Jepsen and her name below it.

Assignment 3

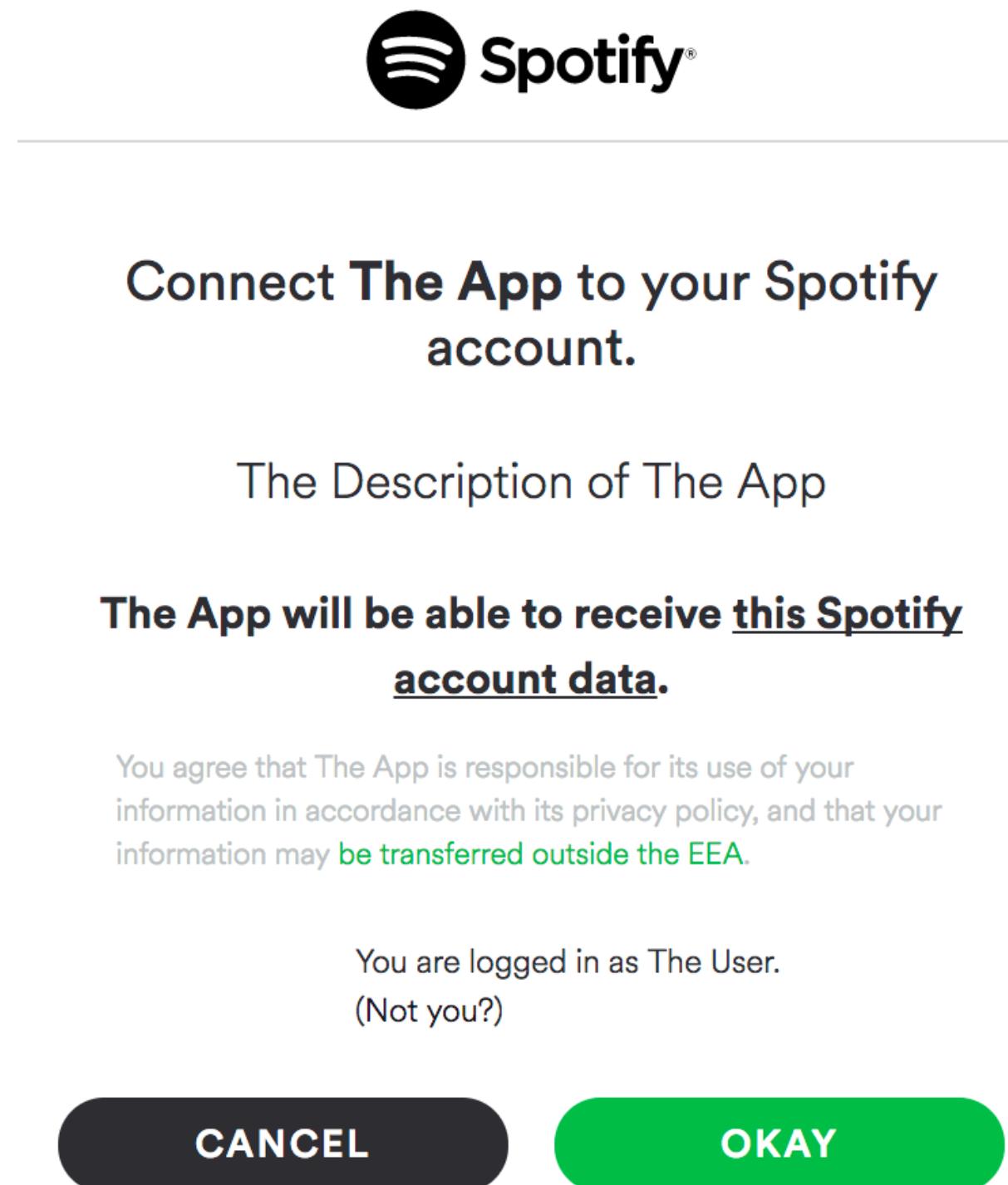
- There's an #assignment3 channel on Slack
- You can use it to look for partners,
as well as ask questions about the assignment

Step 1: request authorization to access data



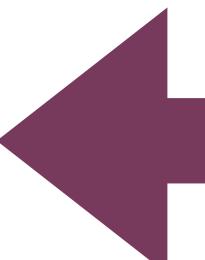
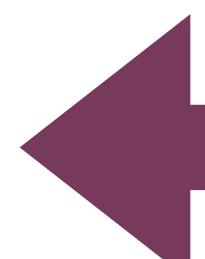
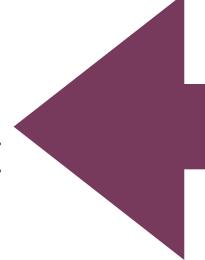
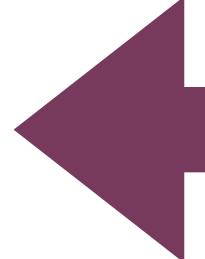
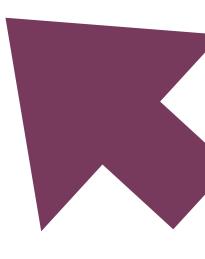
Requesting authorization

- Make a page with links to Spotify's authorization endpoint (<https://accounts.spotify.com/authorize/>)
- Pass arguments in the query string
 - Client ID (public ID of your app)
 - Response type (string "code")
 - Redirect URI (where to return to)
 - Scope (what permissions to ask for)



<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

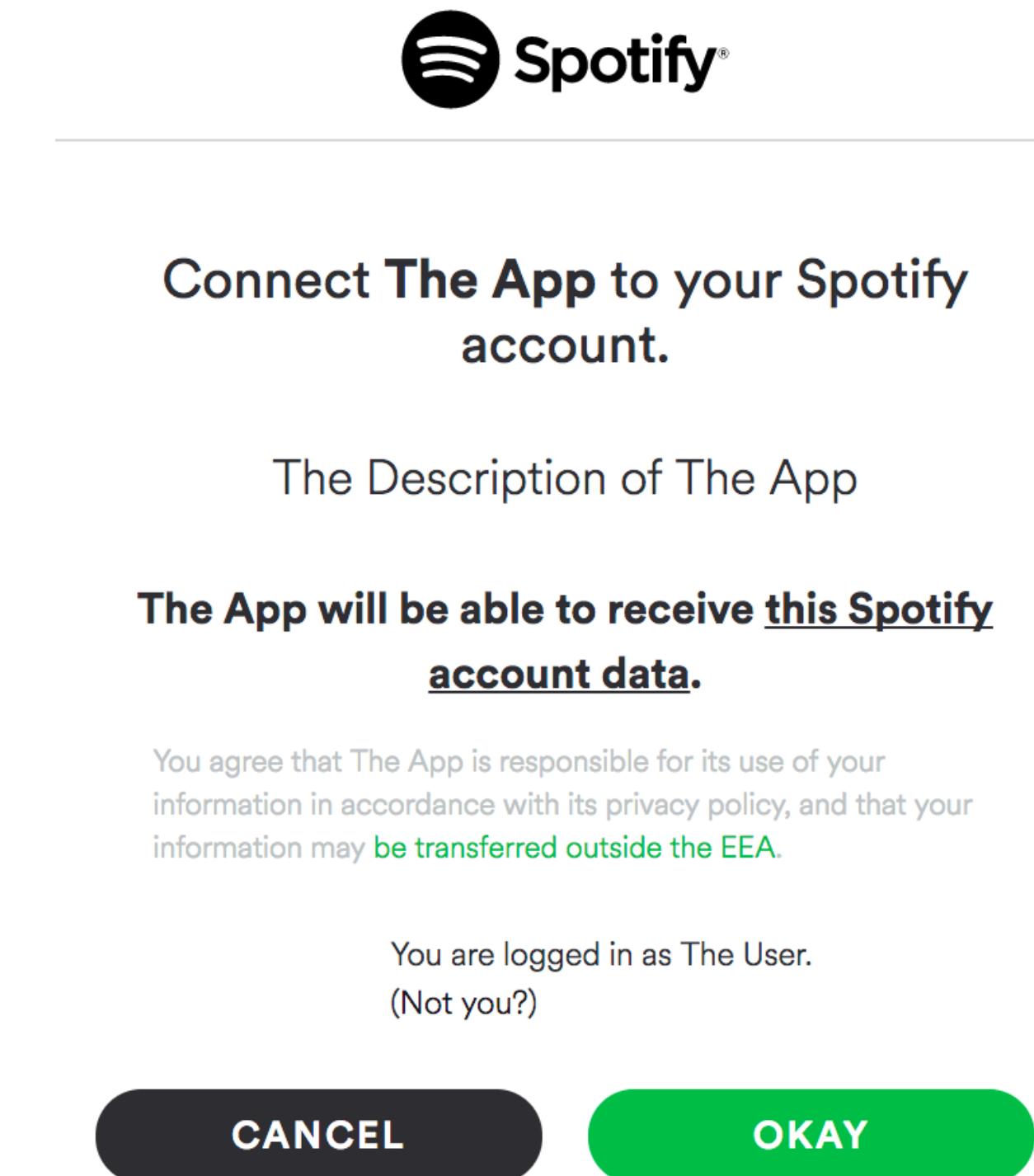
Requesting authorization

- `https://accounts.spotify.com/authorize?`  **Endpoint**
- `response_type=code&`  **“code” response type**
- `client_id=6d81b7a55e894abdbf53143fb2901573&`  **Client id for app**
- `scope=user-read-private%20user-read-email &`  **Scope**
- `redirect_uri=http%3A%2F%2Flocalhost%3A8888%2Fcallback`  **URI to redirect to:
http://localhost:8888/callback**
- **Escaping characters:** `encodeURIComponent()`

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent
<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

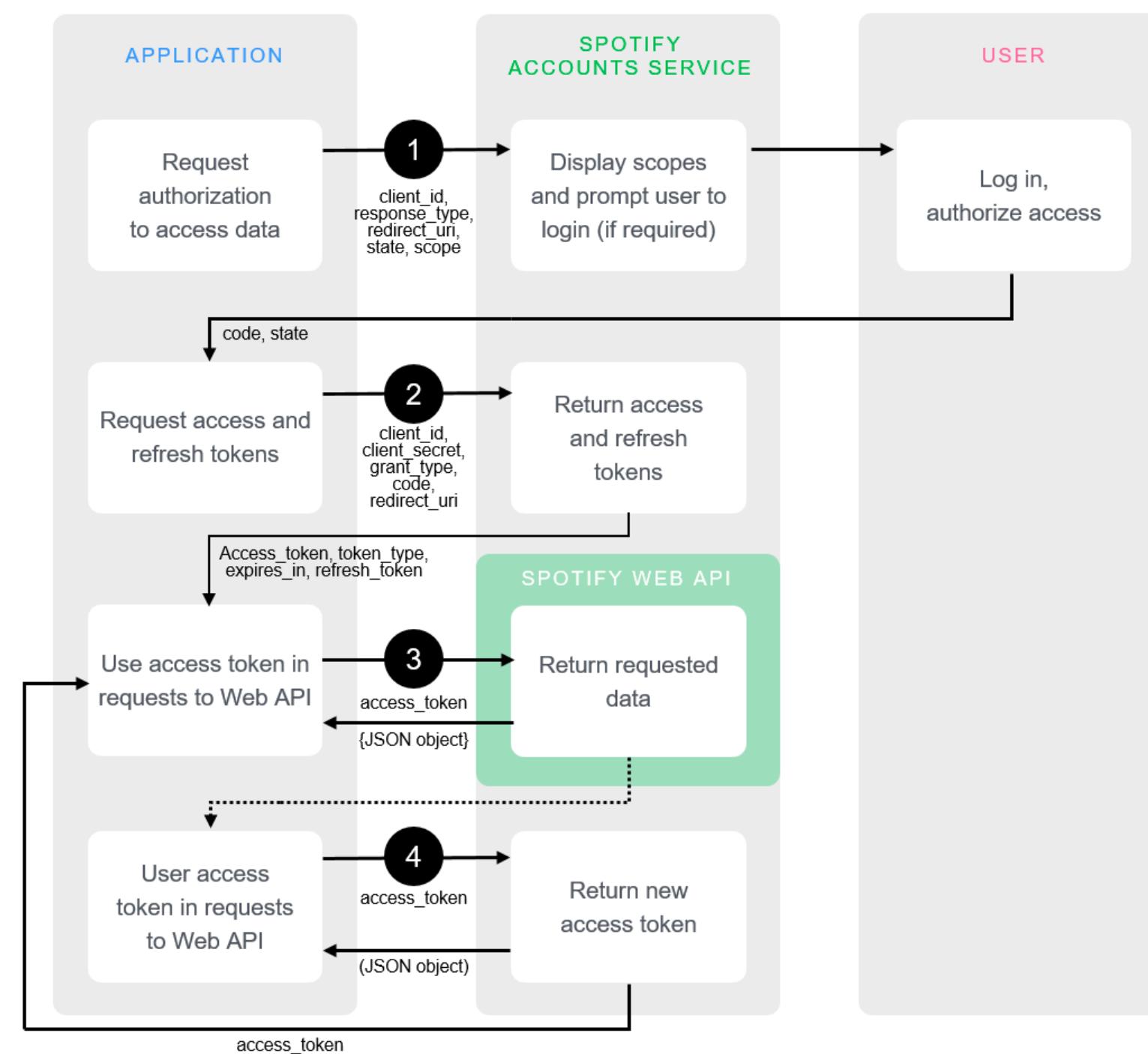
Handling response

- User clicks “okay”, browser then redirects back to your server
- The response contains additional parameters in the URL
- `http://localhost:8888/callback?code=...`
- In Express, code can be accessed through `req.query`



<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Step 2: request access and refresh tokens



Requesting an access token

- Our goal: trade code for an access token
 - An access token needs to be included in API requests
- Why do we need to do this?
 - The user has granted permission for the ID we created on Spotify to access resources
 - But any website could send a user to that URL: client IDs, etc. is all public information
 - How can we verify our app uses the client ID we created on Spotify?

<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Requesting an access token

- We make a POST request with our client's secret code and ask for an access token
 - Endpoint: <https://accounts.spotify.com/api/token>
- Why a POST request rather than a GET?
 - POST sends content in the body of an HTTP request (cannot be read by someone watching your web traffic)
 - GET sends content in the URI
 - `https://accounts.spotify.com/authorize?response_type=code&client_id=6d81b7a55e894abdbf53143fb2901573`

IN4MATX 133 example

IN4MATX 133 course

Client ID 6d81b7a55e894abdbf53143fb2901573

[SHOW CLIENT SECRET](#)

<https://security.stackexchange.com/questions/33837/get-vs-post-which-is-more-secure>
<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Requesting an access token

- Body of POST request requires 3 parameters
 - Grant type (string “authorization_code”)
 - Code (returned as a parameter in the response from the authorization request)
 - Redirect URI (must be the same as before)
- Header of POST request requires 2 parameters
 - Authorization (concatenation of client ID and client secret, as a Buffer)
 - Encoding (via Content-Type, as “*application/x-www-form-urlencoded*”)

<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Requesting an access token

- Making the body: URLSearchParams

- `params = new URLSearchParams();`
- `params.append('grant_type', 'authorization_code');` etc.

- Header: a dictionary

- `'Content-Type': 'application/x-www-form-urlencoded'`
- `'Authorization': 'Basic ' + Buffer.from(my_client_id + ':' + my_client_secret).toString('base64')`

https://www.w3schools.com/nodejs/met_buffer_from.asp

<https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams>

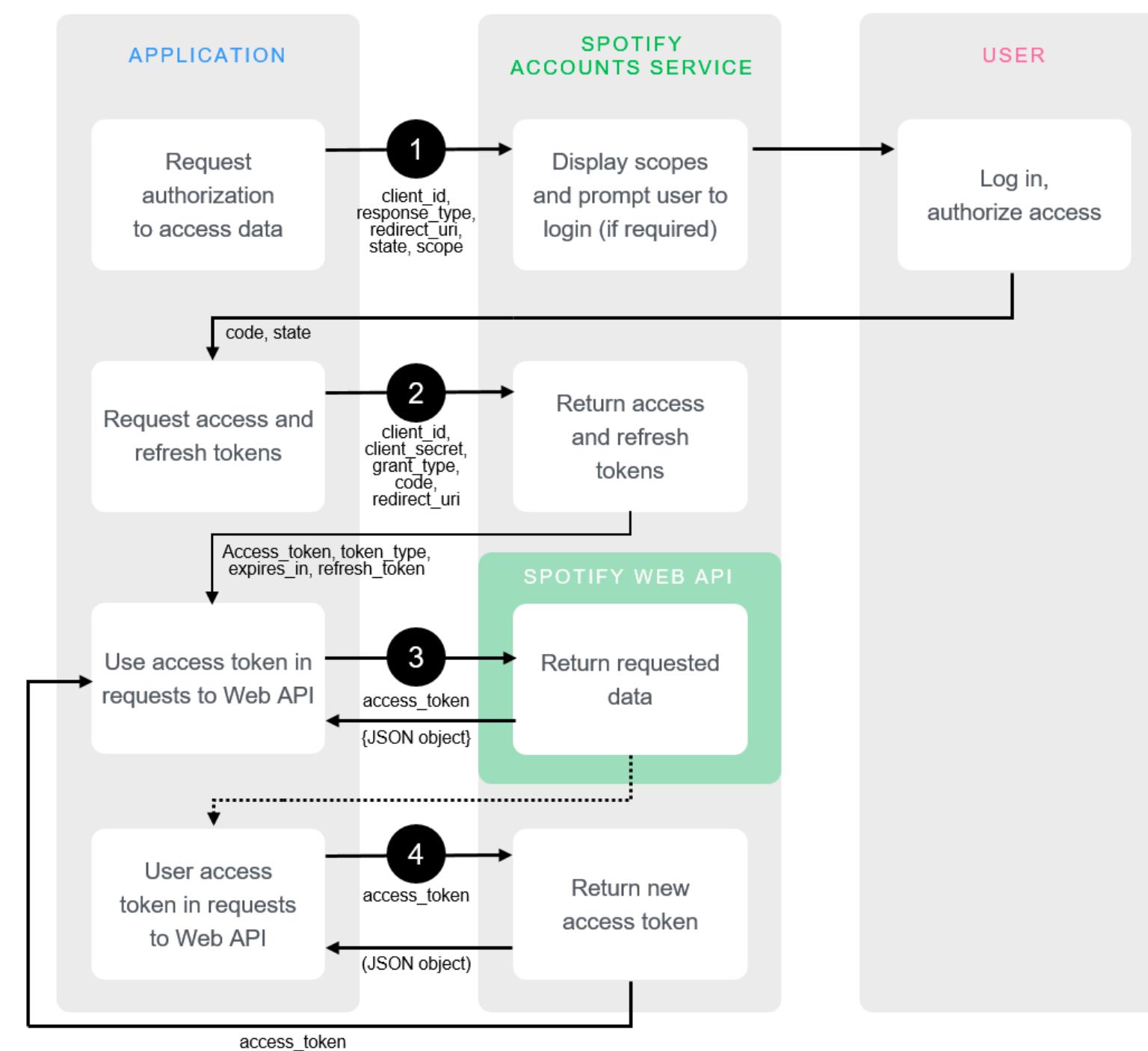
<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Handling response

- In the response body, Spotify sends back:
 - Access Token (needed to make API calls)
 - Expires in (how long the access token is good for)
 - Refresh Token (once the Access Token expires, this can be used to get a new one)
- What would you do with these tokens?
 - Store them in a database for later access
 - In A3, we'll store them in a text file (bad form, but easier)

<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Step 3: use access token in requests to web API



Making an API request

- Pass the access token in the header
 - Much like the client id and secret, but no need to convert it
 - 'Authorization': 'Bearer ' + access_token
- Make a GET request to one of the API endpoints
 - e.g., <https://api.spotify.com/v1/me>
 - Will return a JSON object with the requested resource
 - e.g., birthdate, email, a profile image

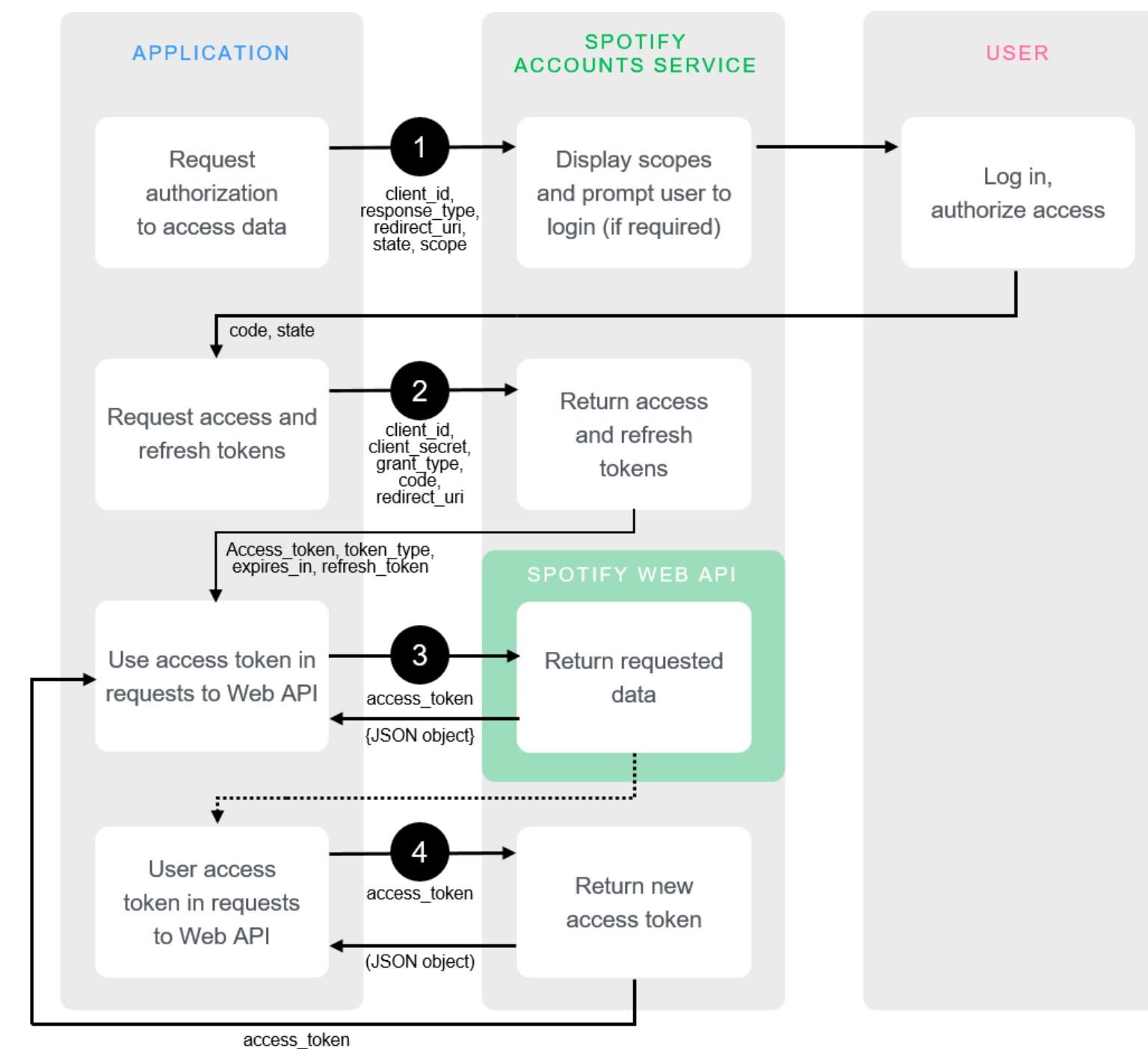
<https://developer.spotify.com/documentation/web-api/reference/users-profile/get-current-users-profile/>
<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Making an API request

- Spotify has endpoints for artists, albums, tracks, and more
- Often specify a subresource in the URI
 - e.g., `https://api.spotify.com/v1/albums/{id}` for a specific album

<https://developer.spotify.com/documentation/web-api/reference/>

Step 4: refresh access token



Refresh token

- Tokens typically expire after a fixed amount of time
 - One hour for Spotify tokens
 - After that time, all API requests will return with code 401 (Unauthorized)
- A user can use the refresh token to get a new token
- Why do tokens expire?
 - To allow a user to revoke their privileges

<https://developer.spotify.com/documentation/web-api/>

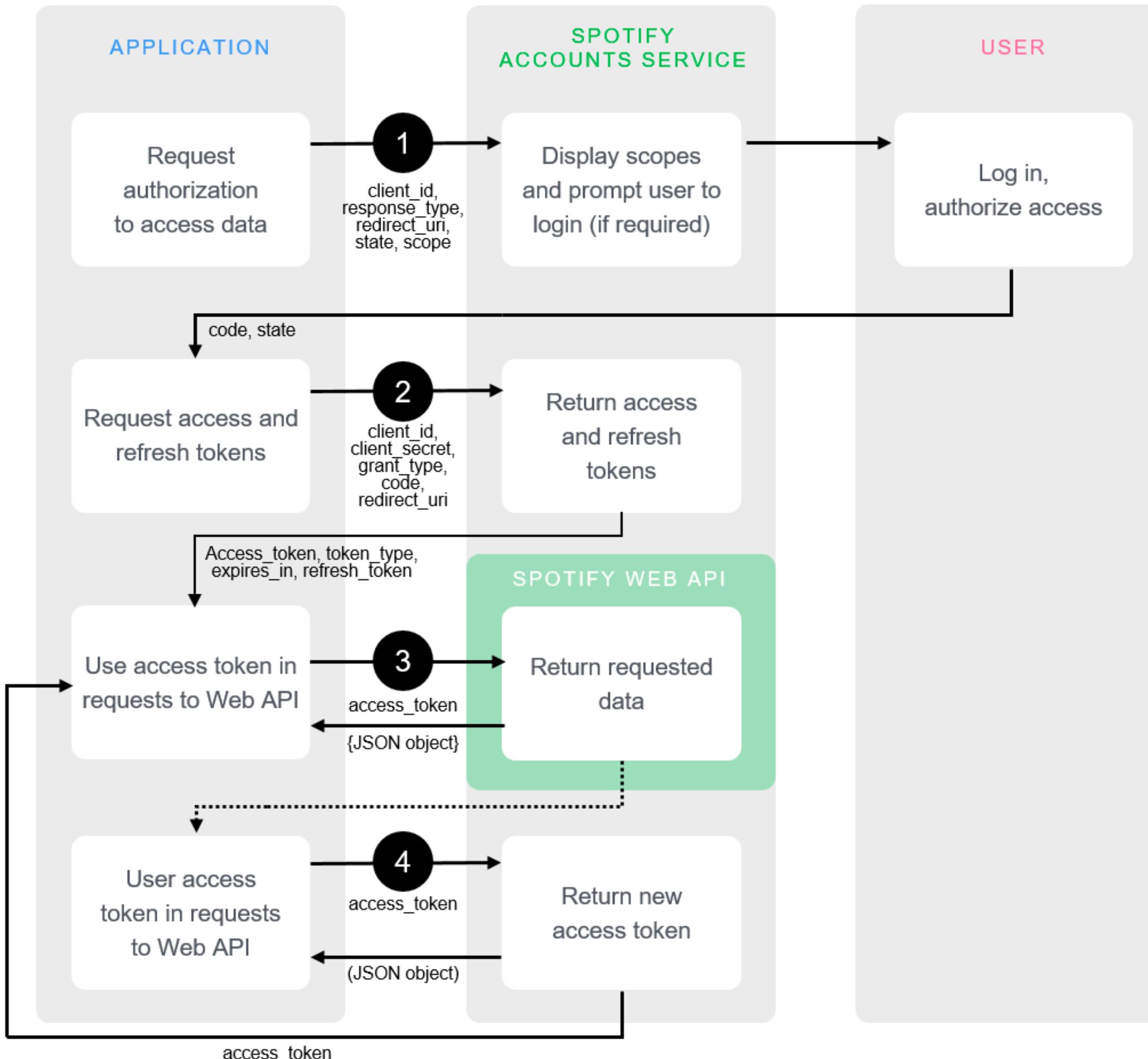
<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Refresh token

- Same endpoint as requesting an access token
 - Endpoint: <https://accounts.spotify.com/api/token>
- Similar parameters; header with encoding and authorization
 - 'Content-Type': 'application/x-www-form-urlencoded'
 - 'Authorization': 'Basic ' + Buffer.from(my_client_id + ':' + my_client_secret).toString('base64')
- Different body parameters
 - "refresh_token" as "grant_type", the token itself as "refresh_token"

<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Oauth 2.0 steps

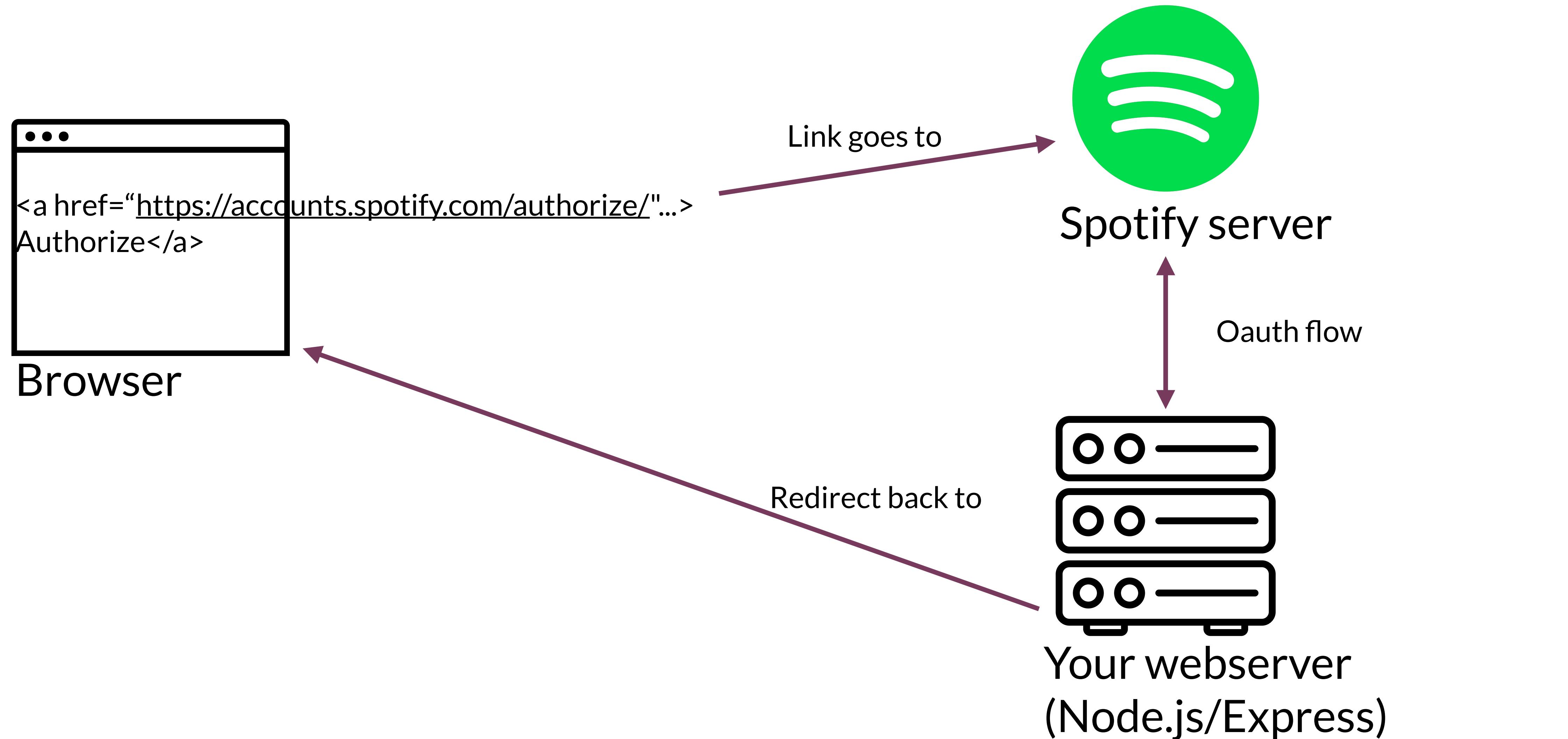


<https://developer.spotify.com/documentation/general/guides/authorization-guide/>

Authorizing from the browser

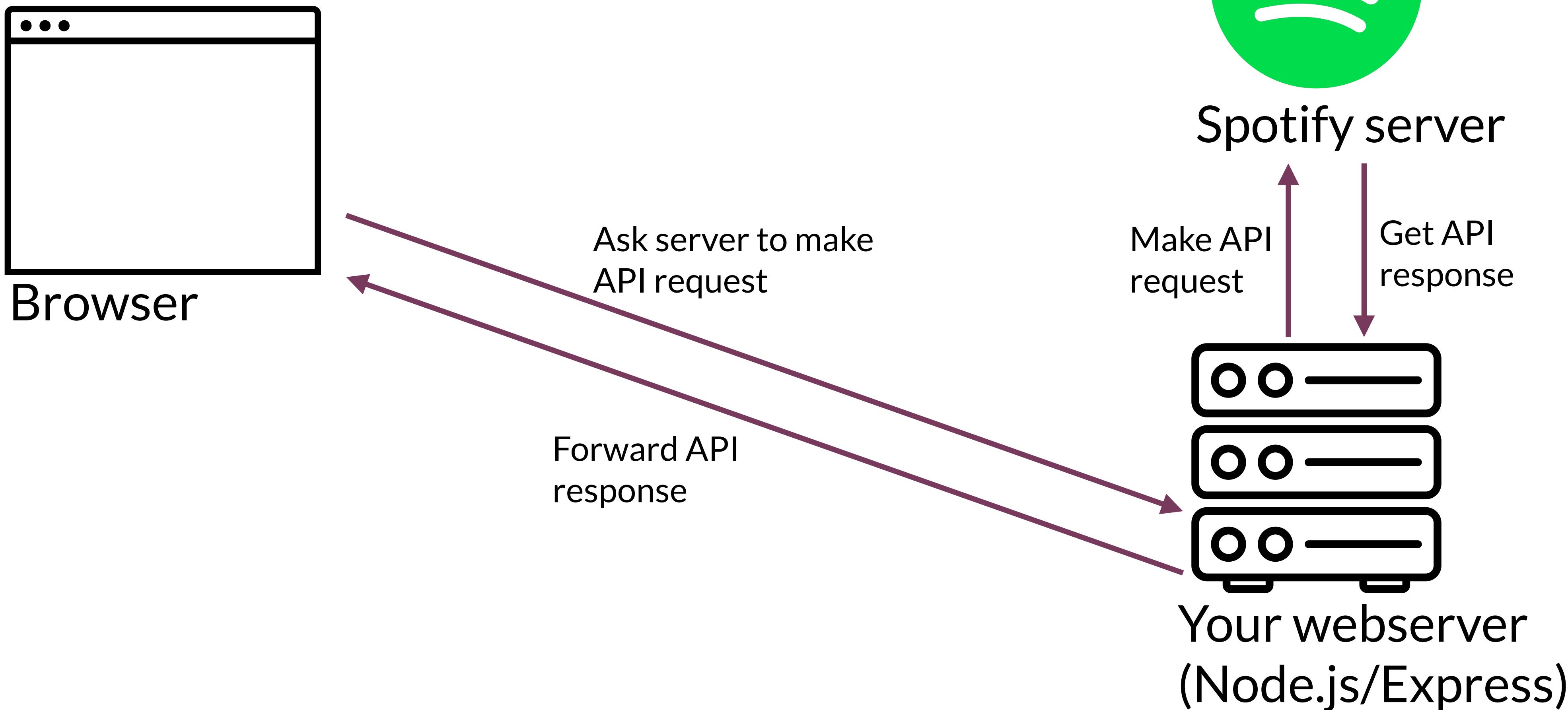
- Create a link to the authorization endpoint
(<https://accounts.spotify.com/authorize/>)
 - Which will redirect to your server-side JavaScript
- Once tokens have been received, redirect back to client-side JavaScript

Authorizing from the browser



Making an API request from the browser

After authorizing



Making an API request from the browser

- How does the browser indicate that it wants the server to make an API request?
 - All web servers communicate in HTTP
 - Make an HTTP request to the server, asking it to make the API request
 - It returns the response

Question



Which can make an HTTP request
to the Spotify API?

(Assume the browser uses default settings)

A 4

B 1, 4

C 1, 2, 4

D 1, 3, 4

E 1, 2, 3, 4

- (1) A browser open to spotify.com
- (2) A browser with client-side JavaScript at localhost:8888
- (3) A browser with server-side JavaScript at localhost:8888
- (4) A server running in the Spotify domain

Which can make an HTTP request to the Spotify API?

(Assume the browser uses default settings)

- A 4
- B 1, 4
- C 1, 2, 4
- D 1, 3, 4
- E 1, 2, 3, 4

- (1) A browser open to spotify.com
- (2) A browser with client-side JavaScript at localhost:8888
- (3) A browser with server-side JavaScript at localhost:8888
- (4) A server running in the Spotify domain

A

0%

B

0%

C

0%

D

0%

E

0%

Question



Which can make an HTTP request
to the Spotify API?

(Assume the browser uses default settings)

A 4

B 1, 4

C 1, 2, 4

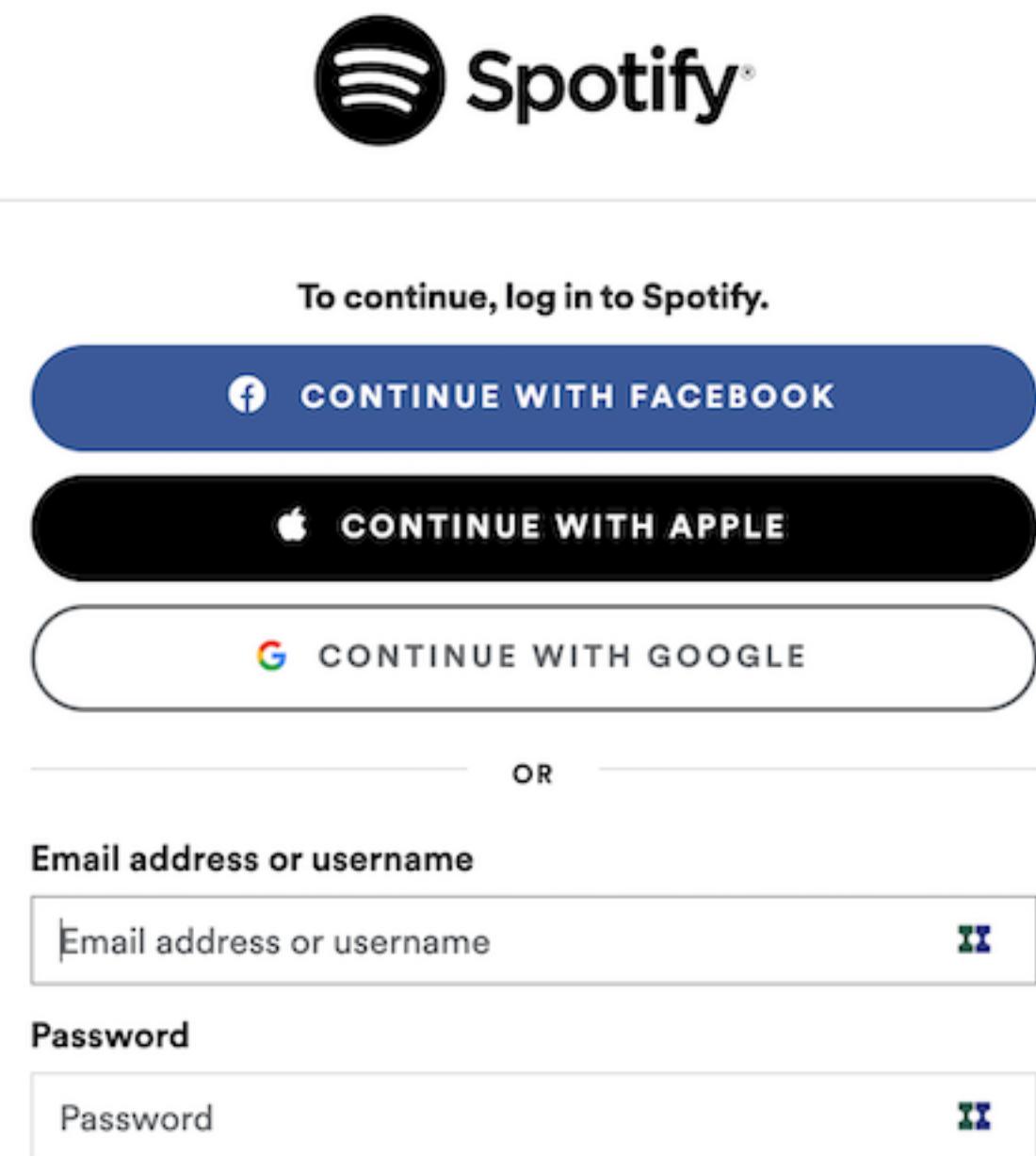
D 1, 3, 4

E 1, 2, 3, 4

- (1) A browser open to spotify.com
- (2) A browser with client-side JavaScript at localhost:8888
- (3) A browser with server-side JavaScript at localhost:8888
- (4) A server running in the Spotify domain

OpenID Connect

- Ever seen a button with “sign in with Google”, etc.?
- Implemented with OpenID Connect
 - Added layer on top of Oauth



<https://openid.net/connect/>

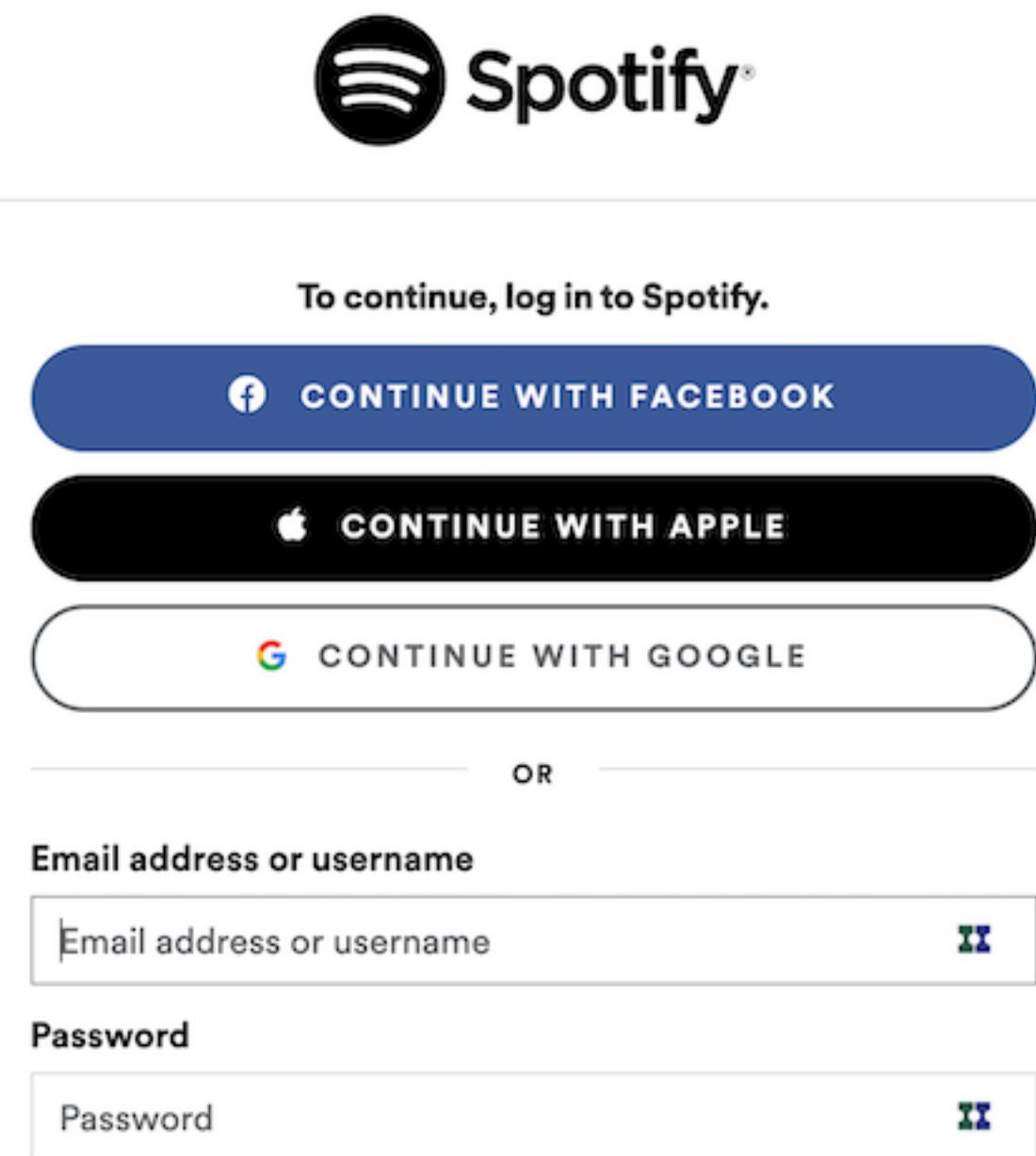
OpenID Connect

- Benefits:

- No need to get an ID for every service
- Only one password to remember/store

- Drawbacks

- Facebook/Google/Apple/etc. gather (more) information about you and the websites you go to



Today's goals

By the end of today, you should be able to...

- Explain the advantages and disadvantages of different tools for server-side development
- Differentiate authentication from authorization
- Describe the utility of supporting authentication and authorization in interfaces
- Explain and implement the different stages to authenticating via OAuth
- Describe the advantages and disadvantages of OpenId

IN4MATX 133: User Interface Software

Lecture 9:
Server-Side Development,
Authentication, & Authorization

Same origin policy*

- Sites running in the same domain but with different ports are technically different origins
- So our live server would typically not be able to access data from another script running on a different port (like the Twitter proxy)

Compared URL	Outcome	Reason
<code>http://www.example.com/dir/page2.html</code>	Success	Same scheme, host and port
<code>http://www.example.com/dir2/other.html</code>	Success	Same scheme, host and port
<code>http://username:password@www.example.com/dir2/other.html</code>	Success	Same scheme, host and port
<code>http://www.example.com:81/dir/other.html</code>	Failure	Same scheme and host but different port
<code>https://www.example.com/dir/other.html</code>	Failure	Different scheme
<code>http://en.example.com/dir/other.html</code>	Failure	Different host
<code>http://example.com/dir/other.html</code>	Failure	Different host (exact match required)
<code>http://v2.www.example.com/dir/other.html</code>	Failure	Different host (exact match required)
<code>http://www.example.com:80/dir/other.html</code>	Depends	Port explicit. Depends on implementation in browser.

https://en.wikipedia.org/wiki/Same-origin_policy

Same origin policy*

- However, the Twitter Proxy (and the Spotify Server in A3) allow for connections from other ports
 - This can be configured in Express
- That means if these were publicly available on the web (versus running on your computer), anyone would be able to use your credentials to make Twitter/Spotify API requests

```
// CORS
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
  res.header("Access-Control-Allow-Headers", "X-Requested-With");
  if (req.method === 'OPTIONS') return res.send(200);
  next();
});
```

<https://github.com/leftlogic/twitter-proxy>

More on Node and Express

Node file system

```
var fs = require('fs'); //Require the file system library  
  
fs.readFile("/path/to/file", function(err, data) {  
  console.log(data);  
});
```

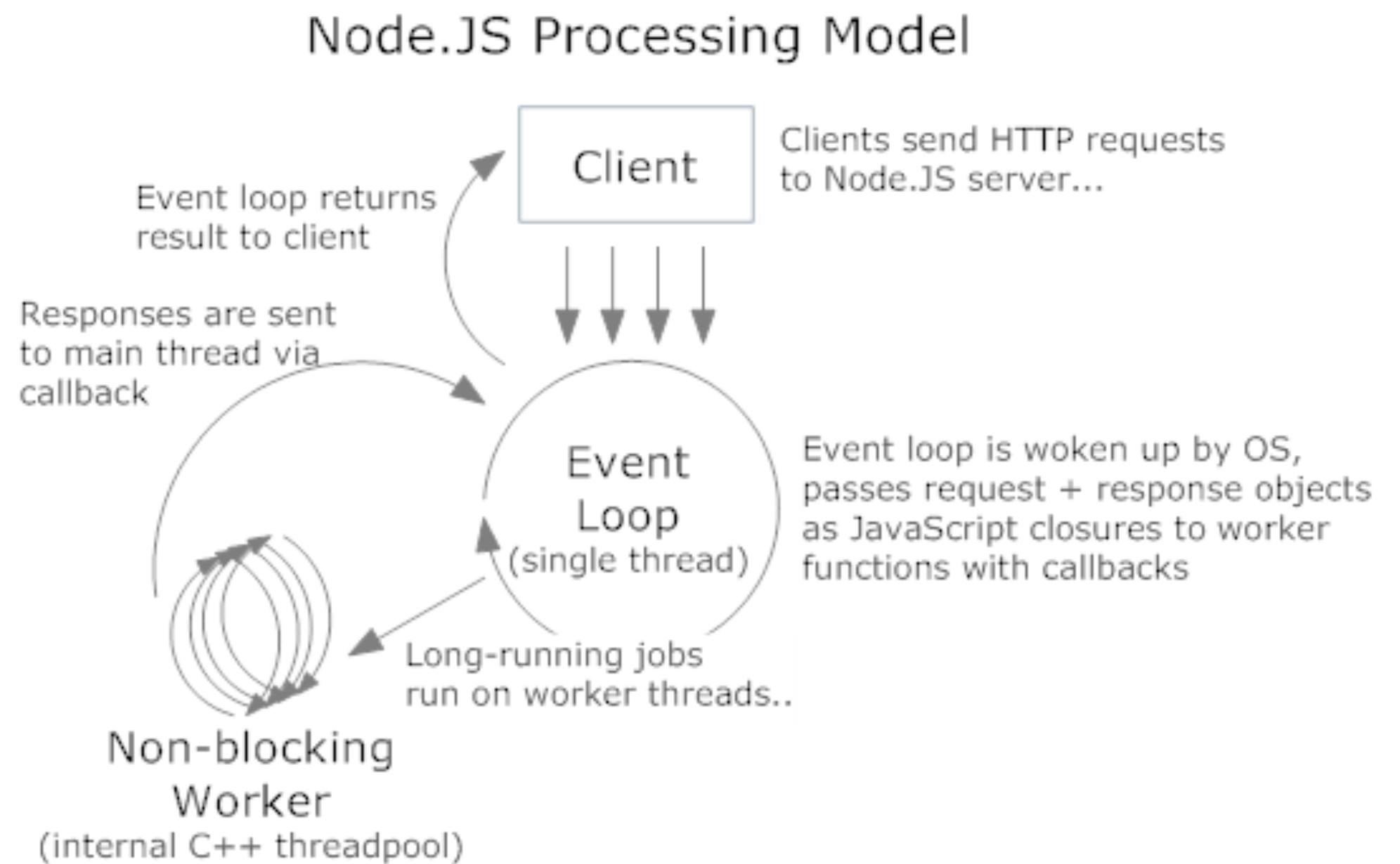
↑
Read file, wait for
asynchronous response

Node file system

```
var http = require('http');
var fs = require('fs');
var server = http.createServer(function(req, res) {
  fs.readFile(__dirname + req.url, function (err,data) {
    if (err) {
      res.writeHead(404);
      res.end(JSON.stringify(err));
      return;
    }
    res.writeHead(200);
    res.end(data);
  });
});
server.listen(8080);
```

Node processing model

- Requests are handled in a single-threaded event loop
 - Every time someone loads a page node manages, it's added to this loop
- Requests are then processed asynchronously
 - When the work a request asks for is done, responses are returned to the client



Express.js

- A fairly minimal web framework that improves Node.js functionality
 - Can route HTTP requests, render HTML, and configure middleware

```
var expressApp = express();
```

```
expressApp.get('/', function (httpRequest, httpResponse)
{
  httpResponse.send('hello world');
});
expressApp.listen(3000);
```

Express installation

- `npm install express`
 - Will save it to your `node_modules` folder

Express routing

- By HTTP method

```
expressApp.get(urlPath, requestProcessFunction);  
expressApp.post(urlPath, requestProcessFunction);  
expressApp.put(urlPath, requestProcessFunction);  
expressApp.delete(urlPath, requestProcessFunction);  
expressApp.all(urlPath, requestProcessFunction);
```

- urlPath may contain parameters (e.g., '/user/:user_id')

httpRequest object

```
expressApp.get('/user/:user_id', function (httpRequest, httpResponse) ...
```

- Has a lot of properties
 - Middleware can add properties
 - `request.params`: object containing url route params (e.g., `user_id`)
 - `request.query`: object containing query params (e.g., `&foo=9 => {foo: '9'}`)
 - `request.body`: object containing the parsed body (e.g., if a JSON object was sent)

httpResponse object

```
expressApp.get('/user/:user_id', function (httpRequest, httpResponse) ...
```

- Has a lot of methods for setting HTTP response fields
 - response.write(content): build up the response body with content
 - response.status(code): set the HTTP status code for the reply
 - response.end(): end the request by responding to it (the only actual response!)
 - response.send(content): write content and then end
- Methods should be chained

```
response.status(code).write(content1).write(content2).end();
```

Middleware

- Give other software the ability to manipulate requests

```
expressApp.all(urlPath, function (request, response,  
next) {  
    // Do whatever processing on request (or setting  
    response)  
    next(); // pass control to the next handler  
});
```

Middleware

- Middleware examples:
 - Check to see if a user is logged in, otherwise send error response and don't call `next()`
 - Parse the request body as JSON and attach the object to `request.body` and call `next()`
 - Session and cookie management, compression, encryption, etc.

Example Express server

```
var express = require('express');
var app = express(); // Creating an Express "App"
app.use(express.static(__dirname)); // Adding middleware
app.get('/', function (request, response) { // A simple request
  handler
  response.send('Simple web server of files from ' + __dirname);
});
app.listen(3000, function () { // Start Express on the requests
  console.log('Listening at http://localhost:3000 exporting the
  directory ' +
  __dirname);
});
```

Example Express user list

```
app.get('/students/list', function (request, response) {
  response.status(200).send(in4matx133.enrolledStudents());
  return;
}) ;
app.get('/students/:id', function (request, response) {
  var id = request.params.id;
  var user = in4matx133.isEnrolled(id);
  if (user === null) {
    console.log('Student with _id:' + id + ' not found.');
    response.status(400).send('Not found');
    return;
  }
  response.status(200).send(user);
  return;
}) ;
```

Express generator

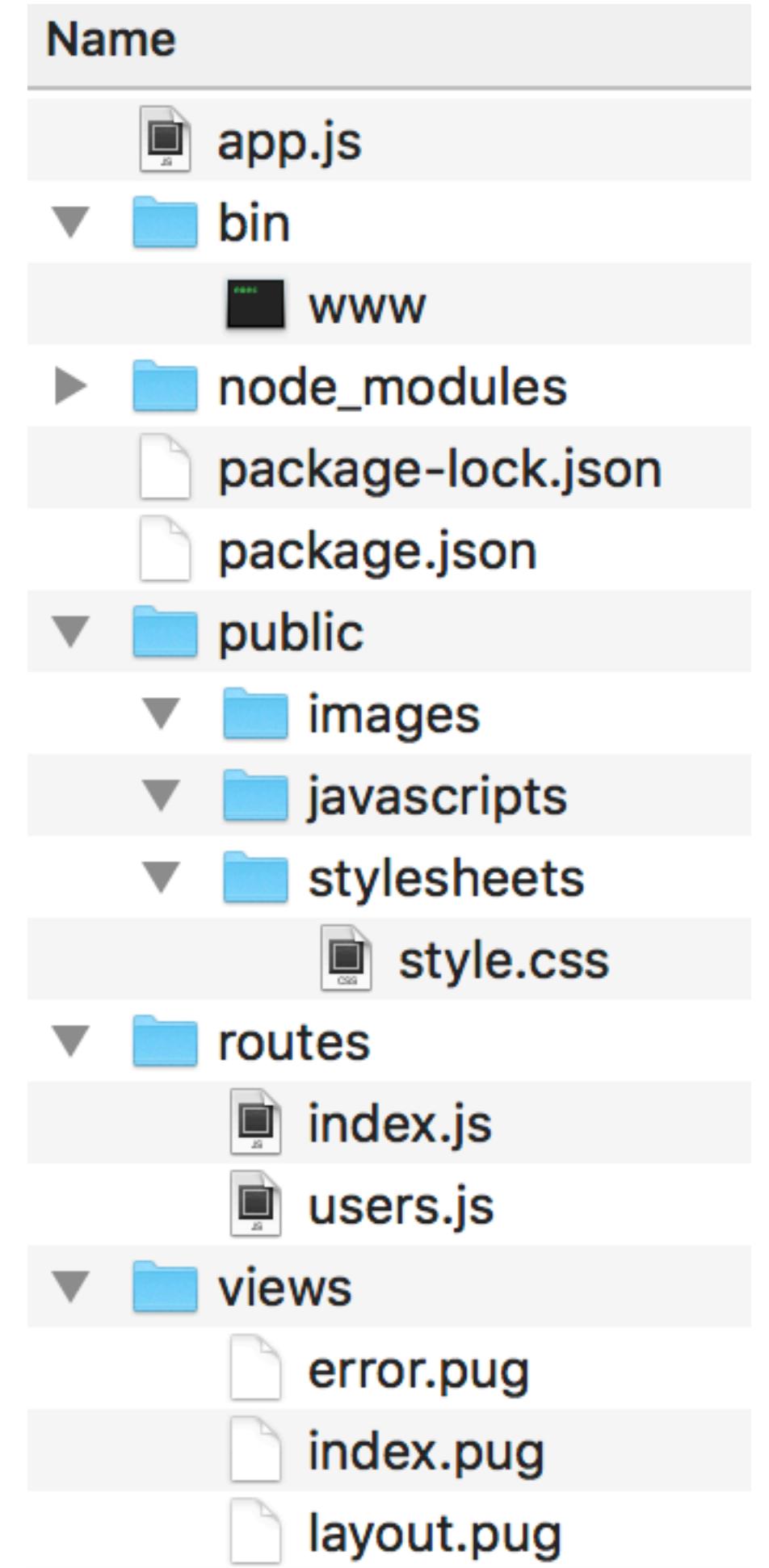
- Express provides a tool that can create and initialize an application skeleton
 - Sets up a directory structure for isolating different components
 - Your app doesn't have to be built this way, but it's a useful starting point

<https://expressjs.com/en/starter/generator.html>

Express generator

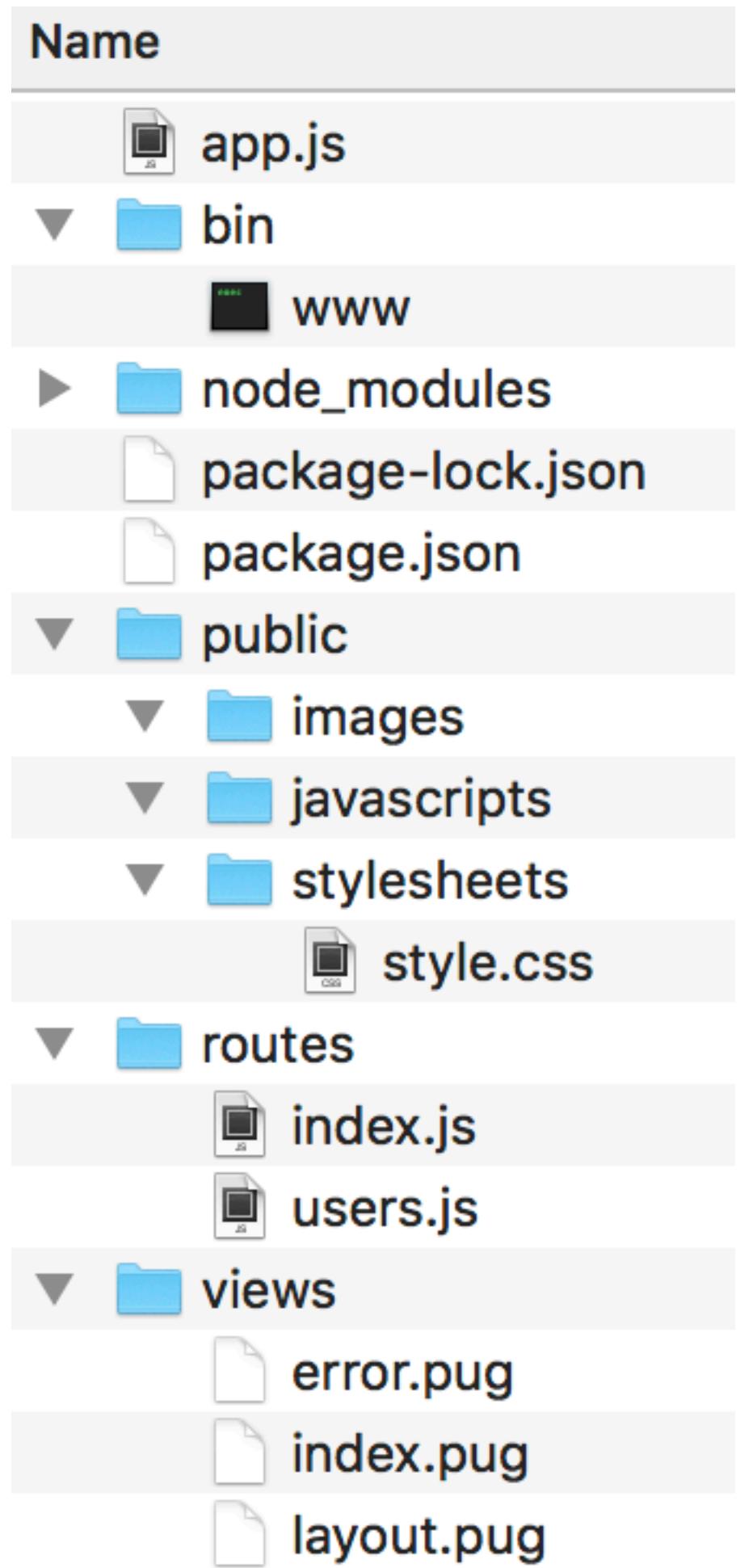
- `npm install express-generator -g`
- Can be invoked on command line with `express`
- Adds some boilerplate code and commonly used dependencies
- Install dependencies with `npm install`
 - `cd` into project directory first
- Run with `npm start`

<https://expressjs.com/en/starter/generator.html>



Express generator

- package.json, package-lock.json, and node_modules folder: library management and installed libraries
- public folder: all public-facing images, stylesheets, and JavaScript files



Express generator

- Routes folder: files which handle your URL mappings

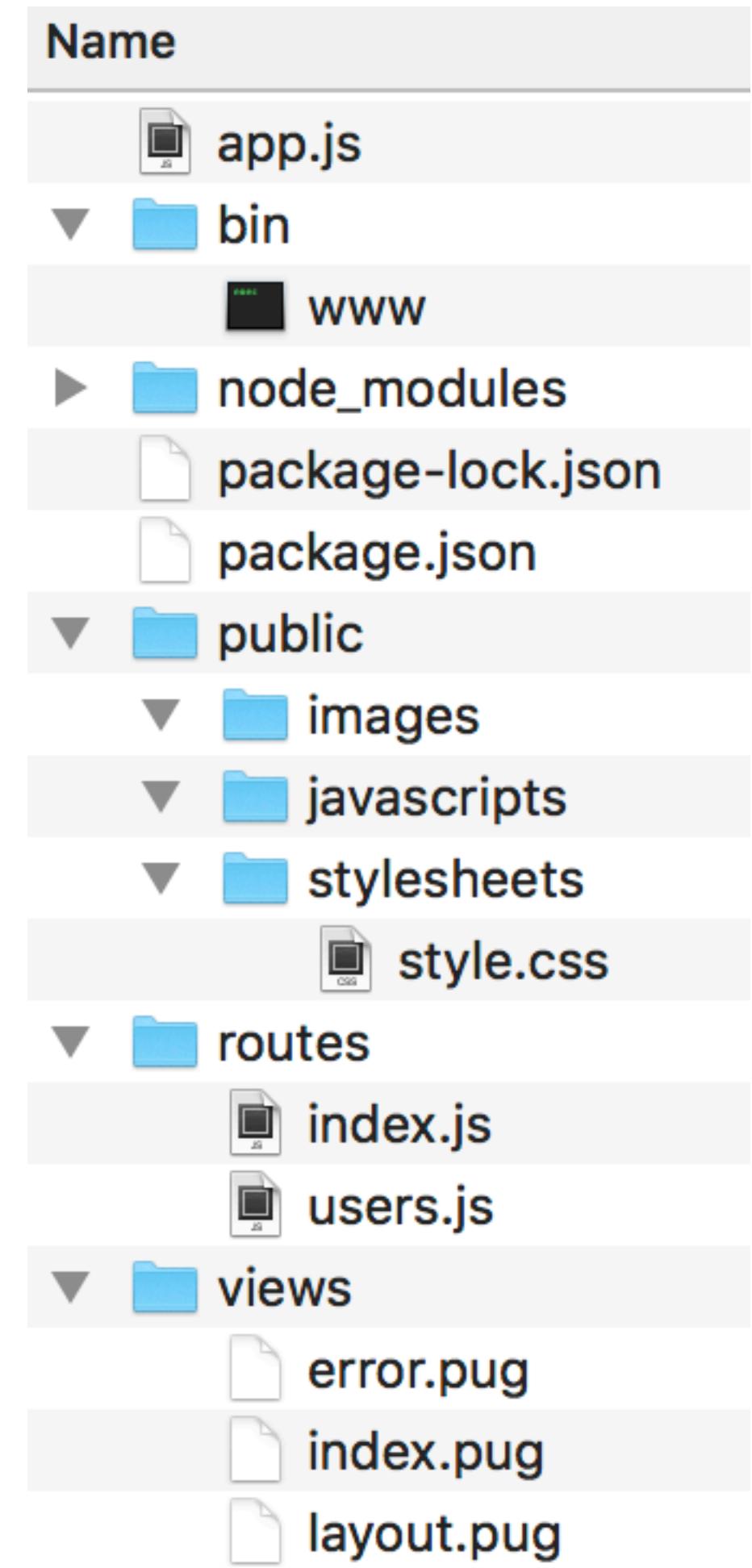
```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

↑
Variable passed to renderer

↑
So another page can import
your router



Express generator

- Views folder: any webpages which need to be rendered
- Uses a *view engine*, Pug, which generates HTML

Name
app.js
bin
www
node_modules
package-lock.json
package.json
public
images
javascripts
stylesheets
style.css
routes
index.js
users.js
views
error.pug
index.pug
layout.pug

Pug view engine

- layout.pug

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/
stylesheets/style.css')
  body
    block content
```

- index.pug

extends layout ← Imports other file

```
block content
  h1= title
  p Welcome to #{title} ← Parses variable passed
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Express</title>
    <link rel="stylesheet" href="/
stylesheets/style.css">
  </head>
  <body>
    <h1>Express</h1>
    <p>Welcome to Express</p>
  </body>
</html>
```

<https://pugjs.org/api/getting-started.html>

Express generator

- app.js: sets up middleware, routers, etc.

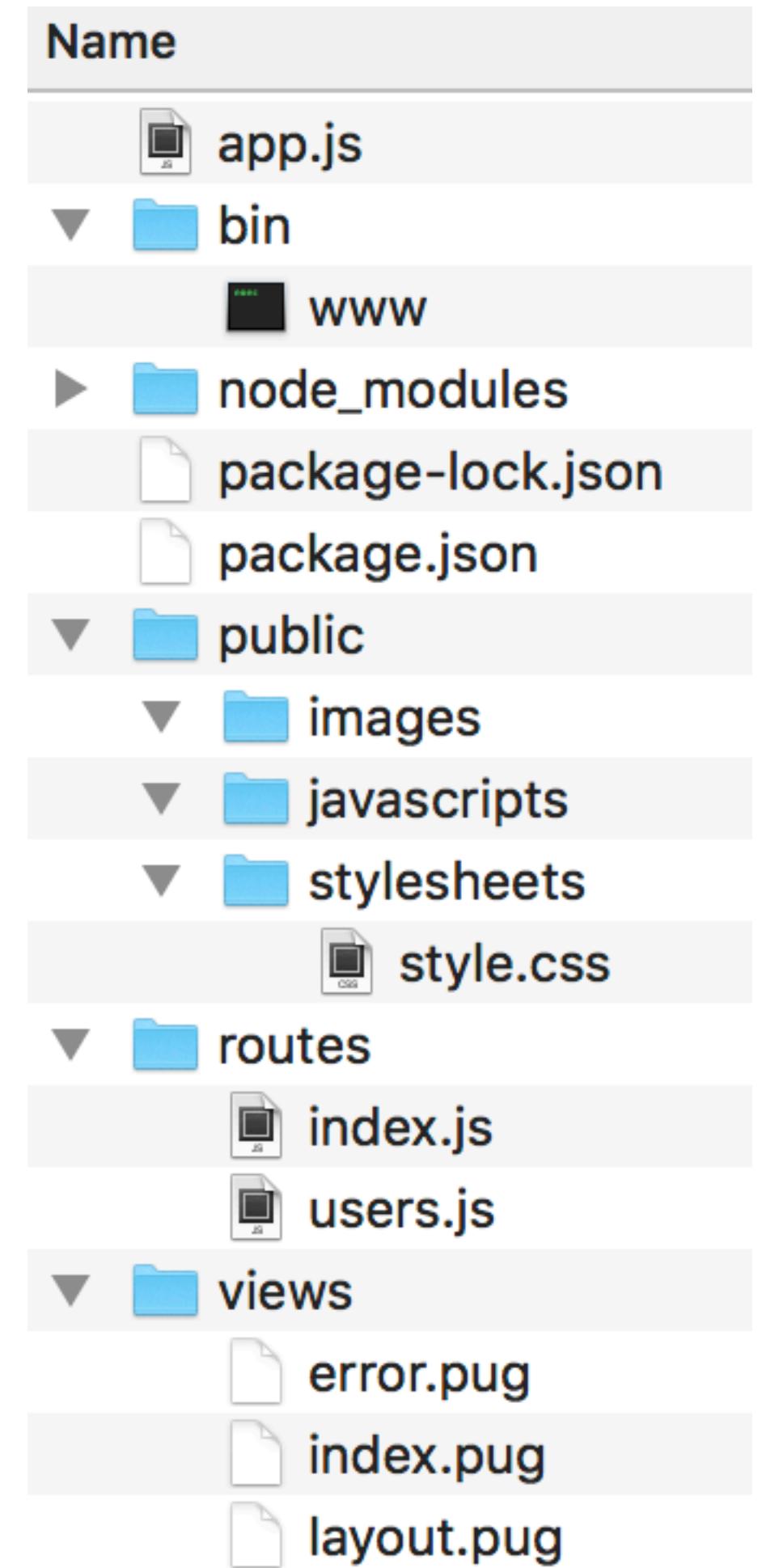
```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();  
  
app.use(express.json()); ← To parse content as json
app.use(express.urlencoded({ extended: false })); ← To encode URLs
app.use(cookieParser()); ← To handle cookies (user state)
app.use(express.static(path.join(__dirname, 'public')));  
  
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

Import route files

To treat the public folder as static content

Middleware



Use route files

Express generator

- bin/www: set up what port to listen on
- File that is run with `npm start`

```
var app = require('../app');
var http = require('http');

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
var server = http.createServer(app);

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```