Lesson 13

Advanced APIs:
Fetching pictures from the 500px photography site

Lesson Objectives

- Understand and explain the OAuth flow
- Get access to a third-party API that requires authentication, by using OAuth and by getting the user to log into the third-party API
- Write an app to pull information from a third-party API, using the user's location, and display that information in the page
- Search documentation needed to make and customize third-party API requests.

Justifications for and examples of closures

https://medium.com/written-in-code/practical-uses-for-closures-c65640ae7304#.356c092rj

https://howtonode.org/why-use-closure

http://reallifejs.com/the-meat/getting-closure/

https://www.sitepoint.com/javascript-closures-demystified/

https://javascriptweblog.wordpress.com/2010/10/25/understanding-javascript-closures/

Quick recap

- API’s are services on the internet that you can make requests to retrieve or change data, via URLs called “endpoints”.
- They are commonly provided by SaaS companies (Github, Instagram, Google)
- They commonly use JSON to communicate the data back and forth.

But what about sensitive endpoints…

- Sometimes these APIs have endpoints that expose things like user data.
- It would be bad if everyone could just hit these API endpoints and retrieve whatever data they want.
- To mitigate this, APIs commonly implement things called:
  - **authentication** (identifying the user as who they say they are, typically with a username/password combo) and
  - **authorization** (limiting access to certain resources based on the identity of the user)
  - The combo is often just called “**auth**” for short

OAuth

Say you're writing a little app, called AwesomeApp…

- AwesomeApp wants to access the user's account on 500px to get some of their pictures, or to get some pictures that are only available to authenticated users
- AwesomeApp can't just log into 500px with the user's 500px username and password, because
  - the user's not going to trust AwesomeApp with those
  - AwesomeApp doesn't want to deal with them anyway
  - That's just not the way it works
- **OAuth** to the rescue.

OAuth in a nutshell

1. AwesomeApp wants to show the user some pictures from 500px, and asks 500px for the pictures.

2. 500px says to AwesomeApp, “Who the **** are you? You’re not an authenticated user! Plus if I’ve never heard of you, how awesome can you be?”

3. AwesomeApp asks the user, “Hey, can you please vouch for me with 500px, and tell them it’s OK if I ask for the pictures?”

4. The user tells 500px, “I know AwesomeApp. They’re pretty cool. You can let them see the pictures”

5. 500px gives AwesomeApp the key to the pictures

OAuth in a nutshell

The slightly more technical version…

1. AwesomeApp wants to get some pictures from the 500px API, using a restricted endpoint at 500px that a user would have to be logged in if they wanted to access.

2. AwesomeApp opens a popup in the browser that directly accesses the 500px API.

3. AwesomeApp types their 500px username and password into the popup, logging into 500px and proving their identity and their right to access the pictures.

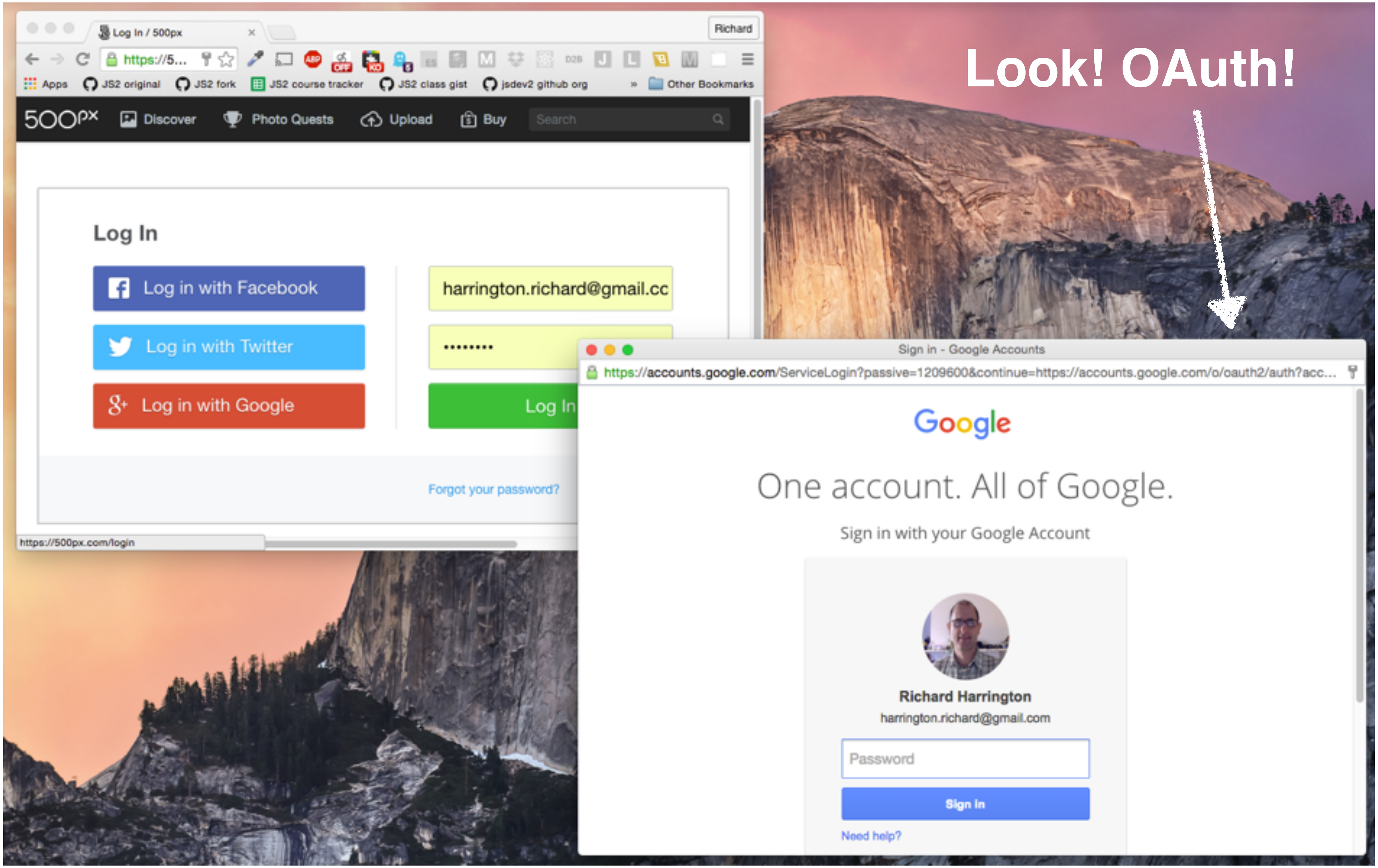
4. 500px redirects the user's browser back to AwesomeApp's web-page and also sends along something called an **access token**.

5. AwesomeApp now has an **access token** that it can use to get the pictures from 500px, on behalf of the user.

Another common use of OAuth: Just keeping track of users' identities

- A lot of sites will use OAuth just to authenticate people and get basic info about them, like email addresses
- Because they want a user to be able to log into their site
  - So they can send them to the right place
  - To keep track of who's doing what and store that info
- But they don't want to deal with usernames and passwords, or the user doesn't trust them with them

This is probably how you've seen Oauth the most:


Look! OAuth!

Log In / 500px

Log In

Log in with Facebook

Log in with Twitter

Log in with Google

harrington.richard@gmail.cc

Log In

Forgot your password?

https://500px.com/login

Sign in - Google Accounts

https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://accounts.google.com/o/oauth2/auth?acc...

Google

One account. All of Google.

Sign in with your Google Account

Richard Harrington

harrington.richard@gmail.com

Password

Sign in

Need help?

LocalLandscapes app

- Let’s get building!
- Our “LocalLandscapes app will show the user a bunch of the most popular landscape shots from 500px, based on the user’s location.

Basic outline of the steps:

1. Get our 500px developer credentials (maybe done already)

2. Create our initial page view which will have a way for the user to initiate an OAuth process with 500px

3. Get the user to log in with 500px, who will then grant us an access token

4. Get the user's location

5. Ping the 500px endpoint with the user's location and the access token

6. Parse the AJAX response for image urls, and put the images into another page view

Which 500px endpoint do we hit?

- Let’s find out if our idea is feasible.
- We want to find landscape photos based on location.
- Is this possible?
- To the documentation, Batman!

Recap: Getting the credentials

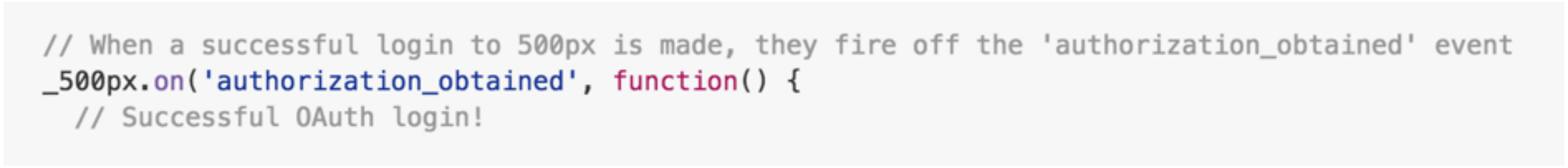
- https://gist.github.com/richardharrington/6dbdee3d94000079a4f0de6cf4326f5d

What is an SDK?

- An SDK (Software Development Kit) is a small library put out by a software maker in order to make it easier for developers to use their software
    - (in this case, the 500px API)
- We will use the 500px SDK methods for things like
    - AJAX calls
    - handling the OAuth access token
- The 500px SDK has docs on their site
- Also the code for the 500px SDK is pretty short, and sort of readable, if we need to look there to figure out how something works

Custom event systems

- The 500px SDK has a custom event system that comes with it.
- You listen for special events using the `on` method of the main `_500px` object.
- It’s a lot like the other main event system we’ve used (the one in the DOM).
- We’ll explain it when we go over it.



```
// When a successful login to 500px is made, they fire off the 'authorization_obtained' event
_500px.on('authorization_obtained', function() {
  // Successful OAuth login!
```

To the code!

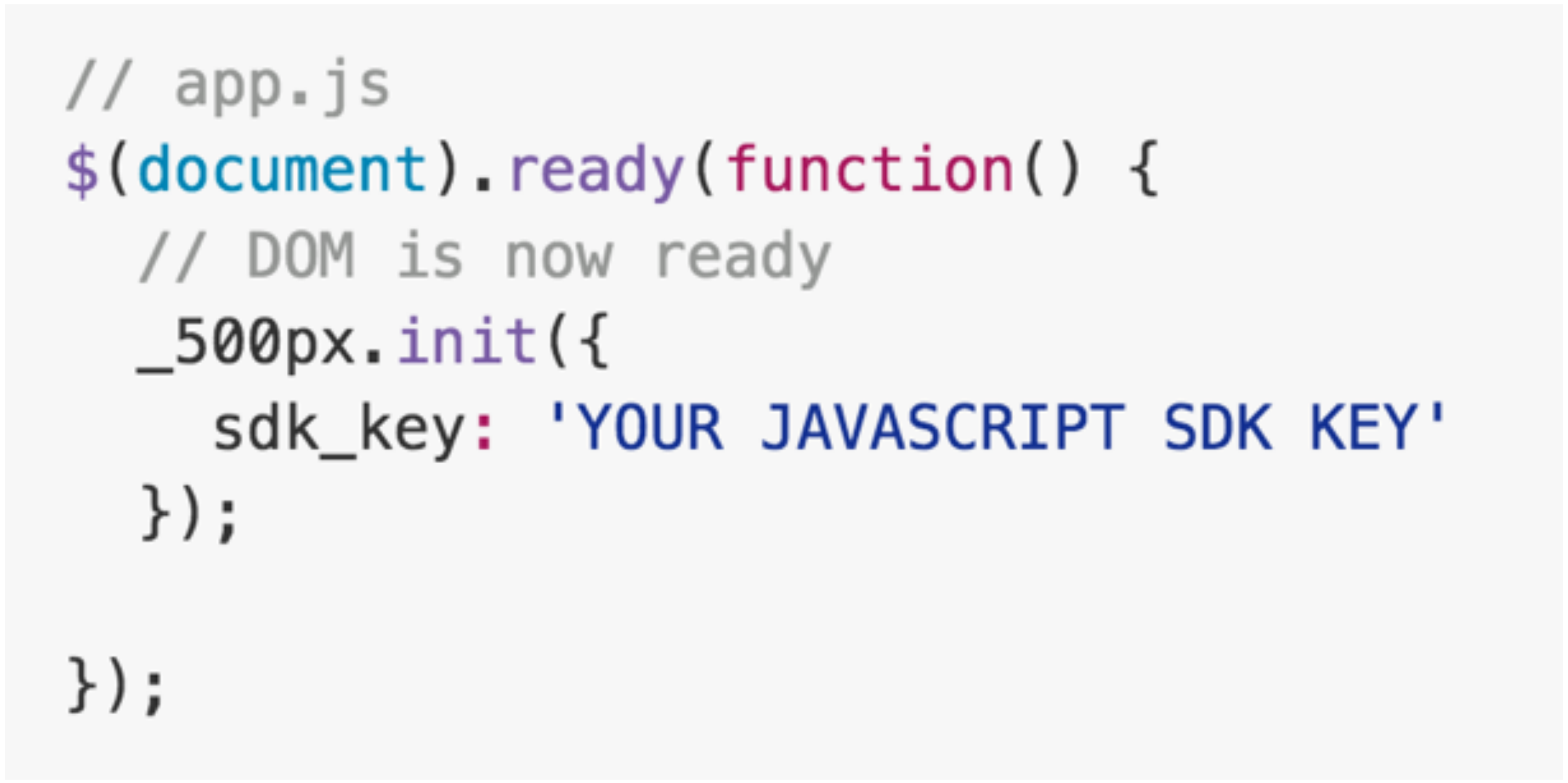
- Download the zip file at http://bit.ly/jsdev2-lesson13-starter
- Unzip it (should be a “lesson13” folder)
- Put it in your **~/GA-JS** folder

Running a simple server

- In order for the OAuth redirect to work, we have to run our app from a server instead of just opening a file from the browser
- Install a program that runs a server:
    - `**npm install -g http-server**`
- Now in our terminal, from the path of our app (i.e. **~/GA-JS/lesson13/**) we can run the command
    - `**http-server -p 3000**`.
    - This command simply says start up a server on port 3000 for the code that is in the current path (don't worry about ports are for now).
    - Then visit `**http://localhost:3000**` from your browser

Set up the login - 1

- Initialize the SDK with a config object (just the **sdk_key**)



```
// app.js
$(document).ready(function() {
  // DOM is now ready
  _500px.init({
    sdk_key: 'YOUR JAVASCRIPT SDK KEY'
  });

});
```

Set up the login - 2

- When the user clicks on the Login button, launch the OAuth process using the 500px SDK's **login** method

```
...
      <button class="btn btn-default">
        <a href="#" id="login">Login to 500px</a>
      </button>
...
```

```
// app.js
...
$('#login').click(function() {
  _500px.login();
});
```

Set up the login - 3

- Upon completion of the OAuth process at login, the custom event **authorization_obtained** is fired off.
- This means our app has now received the access token.
  - (the 500px SDK will keep track of the access token under the hood for us).
- Then the login window will close.
- We can assign an event handler to listen for this event and execute the rest of our code

Set up the login - 4

```
// app.js
$(function() {
  // DOM is now ready
  _500px.init({
    sdk_key: 'YOUR JAVASCRIPT SDK KEY'
  });

  // If the user clicks the login link, log them in
  $('#login').click(function() {
    _500px.login();
  });

  // When a successful login to 500px is made, the SDK fires
  // off the 'authorization_obtained' event
  _500px.on('authorization_obtained', function() {
    // Successful OAuth login!
    // What should we do now?

  });

});
```

Independent practice: setting up the conditional showing of the 2 views

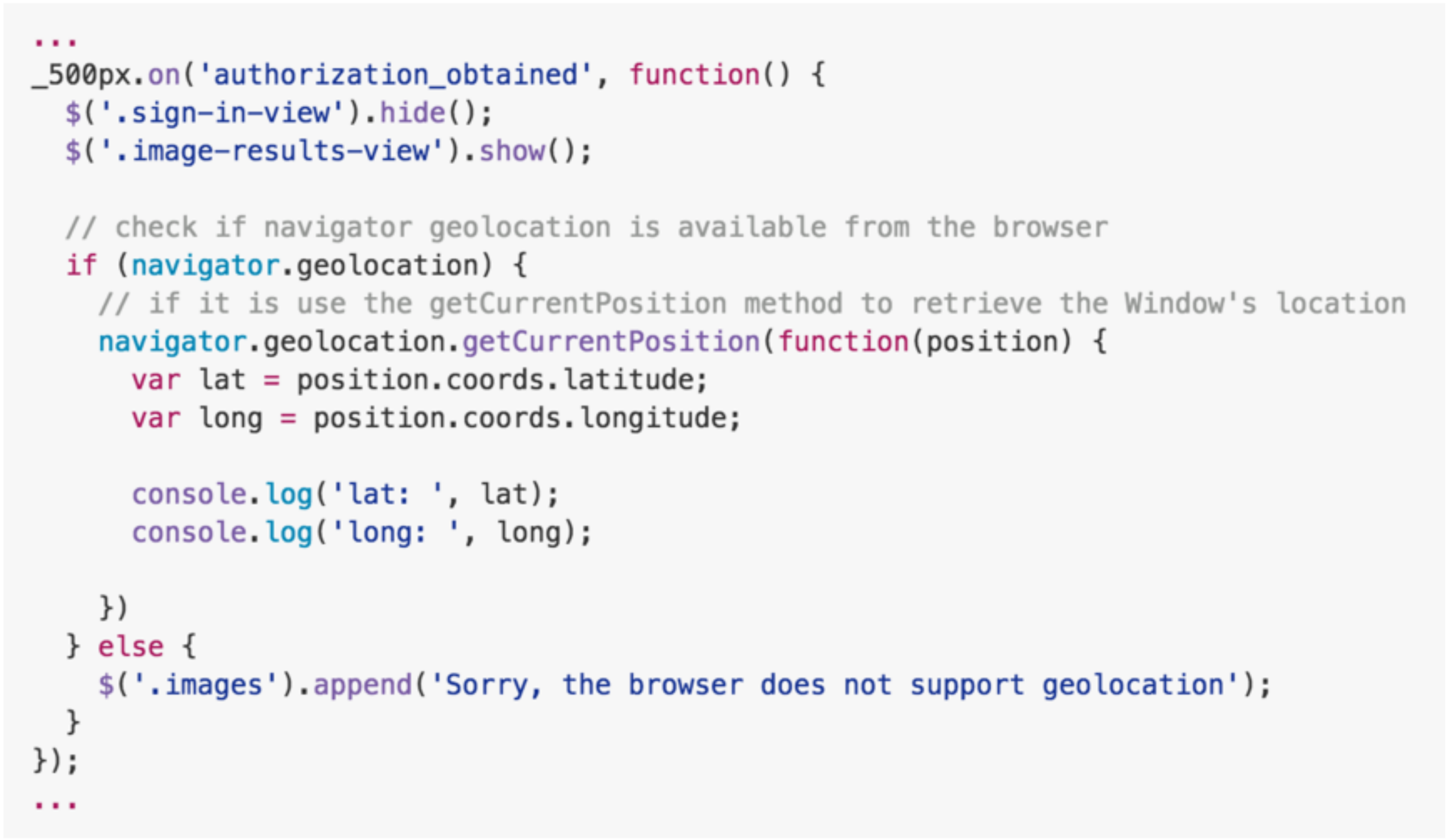
- We want the html showing the pictures to be hidden when we start
- Then we want to show that and hide the login html, once we get the access token back from 500px
- So set that up, starting with this:

```
_500px.on('authorization_obtained', function() {

   // … Your code here

});
```

Next: Get the user’s location

- We want to find photos based on the user’s location
- From the docs, we know we can do a search for photos using latitude, longitude, radius, and category
- We can get the user’s location from the browser’s **navigator** object
- **navigator.geolocation.getCurrentPosition()** should do the trick
- But check first if **navigator.geolocation** is available
- What kind of thing does **.getCurrentPosition()** return? Let’s check!

Getting the user's location



```
...
_500px.on('authorization_obtained', function() {
  $('.sign-in-view').hide();
  $('.image-results-view').show();

  // check if navigator geolocation is available from the browser
  if (navigator.geolocation) {
    // if it is use the getCurrentPosition method to retrieve the Window's location
    navigator.geolocation.getCurrentPosition(function(position) {
      var lat = position.coords.latitude;
      var long = position.coords.longitude;

      console.log('lat: ', lat);
      console.log('long: ', long);

    })
  } else {
    $('.images').append('Sorry, the browser does not support geolocation');
  }
});
...
```

Now we have all the pieces of the puzzle, we can make the API call!

- We have obtained the user’s location and gotten 500px to give us an OAuth access code
- We can now make the API call to get the pictures, with the user’s location and the category of picture we want.
- The 500px SDK will take care of sending the OAuth access code along with the API call

Now we have all the pieces of the puzzle, we can make the API call!

```
...
  var lat = position.coords.latitude;
  var long = position.coords.longitude;

  console.log('lat: ', lat);
  console.log('long: ', long);

  // Feel free to adjust the search radius as you see fit
  var radius = '25mi';

  var searchOptions = {
    geo: lat + ',' + long + ',' + radius,
    only: 'Landscapes', // We only want landscape photos
    image_size: 3 // This isn't neccessary but by default the images are thumbnail sized
  };

  _500px.api('/photos/search', searchOptions, function(response) {
    if (response.data.photos.length == 0) {
      alert('No photos found!');
    } else {
      handleResponseSuccess(response); // This function has not been written yet

    }
  });
...
```

Independent practice: handle the response

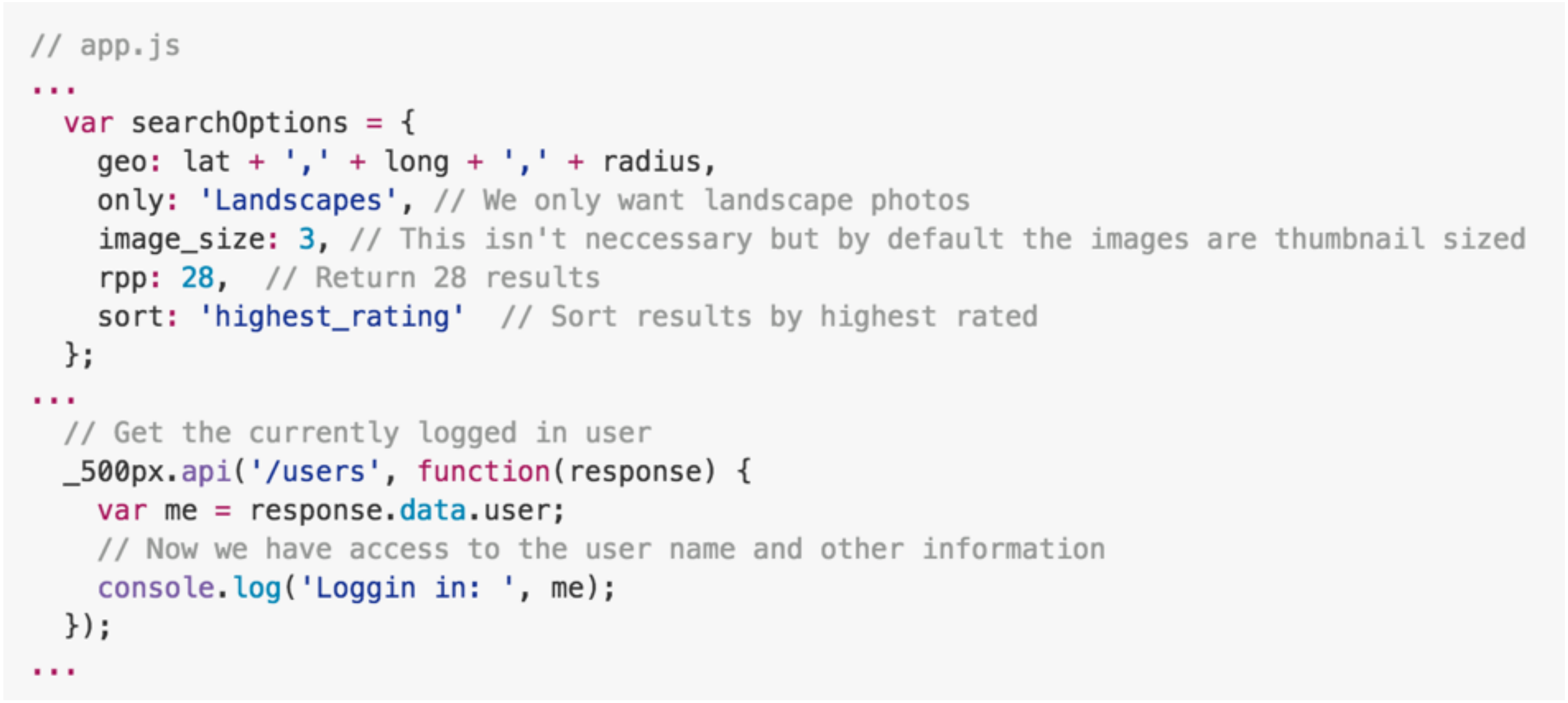
- Write the function **handleResponseSuccess** to handle the response.
- Your function should iterate through the response data and
  - Create an image element on the page, for each image url in the response data.
  - Add the class **image** to the image
  - Append it to the existing **.images** element in the DOM.
- Hint: The data you're looking for is somewhere in **response.data**

Voilà!

Independent Practice: API Exploration

- Get some practice reading API docs. Modify our request to also:
  - Sort photo results by highest rated first
  - Return 28 photos instead of the default 20
- **BONUS:** Display the current user's information on the site after a successful login.
  - Hint: look on the API docs for another API endpoint, besides **/search**

API Exploration Review



```
// app.js
...
  var searchOptions = {
    geo: lat + ',' + long + ',' + radius,
    only: 'Landscapes', // We only want landscape photos
    image_size: 3, // This isn't neccessary but by default the images are thumbnail sized
    rpp: 28,  // Return 28 results
    sort: 'highest_rating'  // Sort results by highest rated
  };
...
  // Get the currently logged in user
  _500px.api('/users', function(response) {
    var me = response.data.user;
    // Now we have access to the user name and other information
    console.log('Loggin in: ', me);
  });
...
```

Conclusion

- What is the OAuth flow?
- Why do some third-party APIs require authentication?
- Go ahead and refactor your code as much as possible. Make it DRY (Do Not Repeat Yourself)!