ASSIGNMENT 3 : PARTS 1 + 2 + 3 (OF 5)

Progressive Web Apps
2023

You must create a Progressive Web App with the functionality described in this document.

The PWA will be hosted on a server provided for you and will be served via HTTPS , it must work offline, and be installable.

Features of Assessment:

Work Offline with Service Workers

Cache Management

Multithreaded with Web Workers

DOM Scripting

Web Services

Objects / Closures

Part 1: Create an App Shell (Tip: refine - Assignment 1)

Part 2: Create a page to Search Flickr

Part3: Create a page to Search a large JSON object (see attached – Json file)

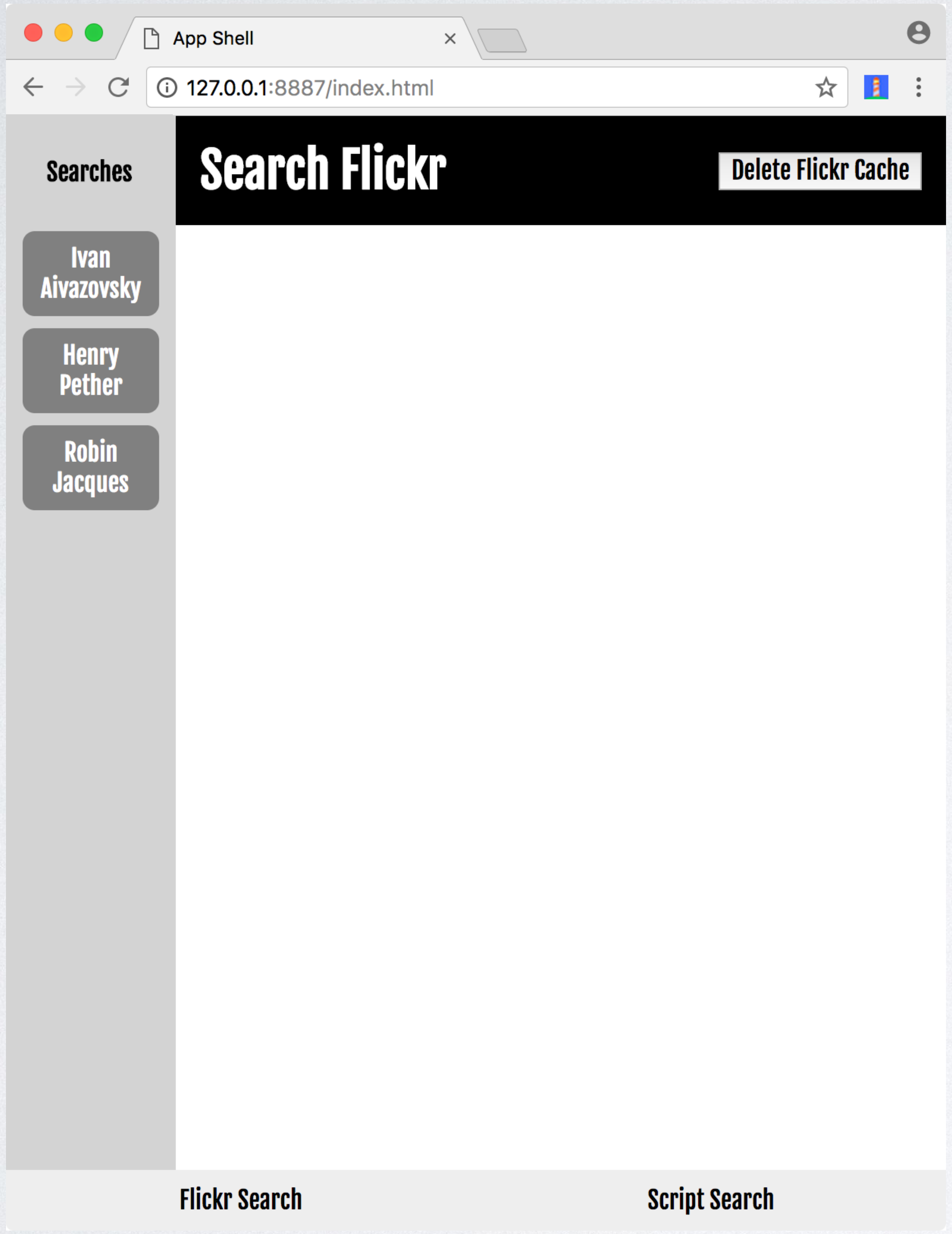
Part 4: Make your site Installable as app (i.e. a PWA)

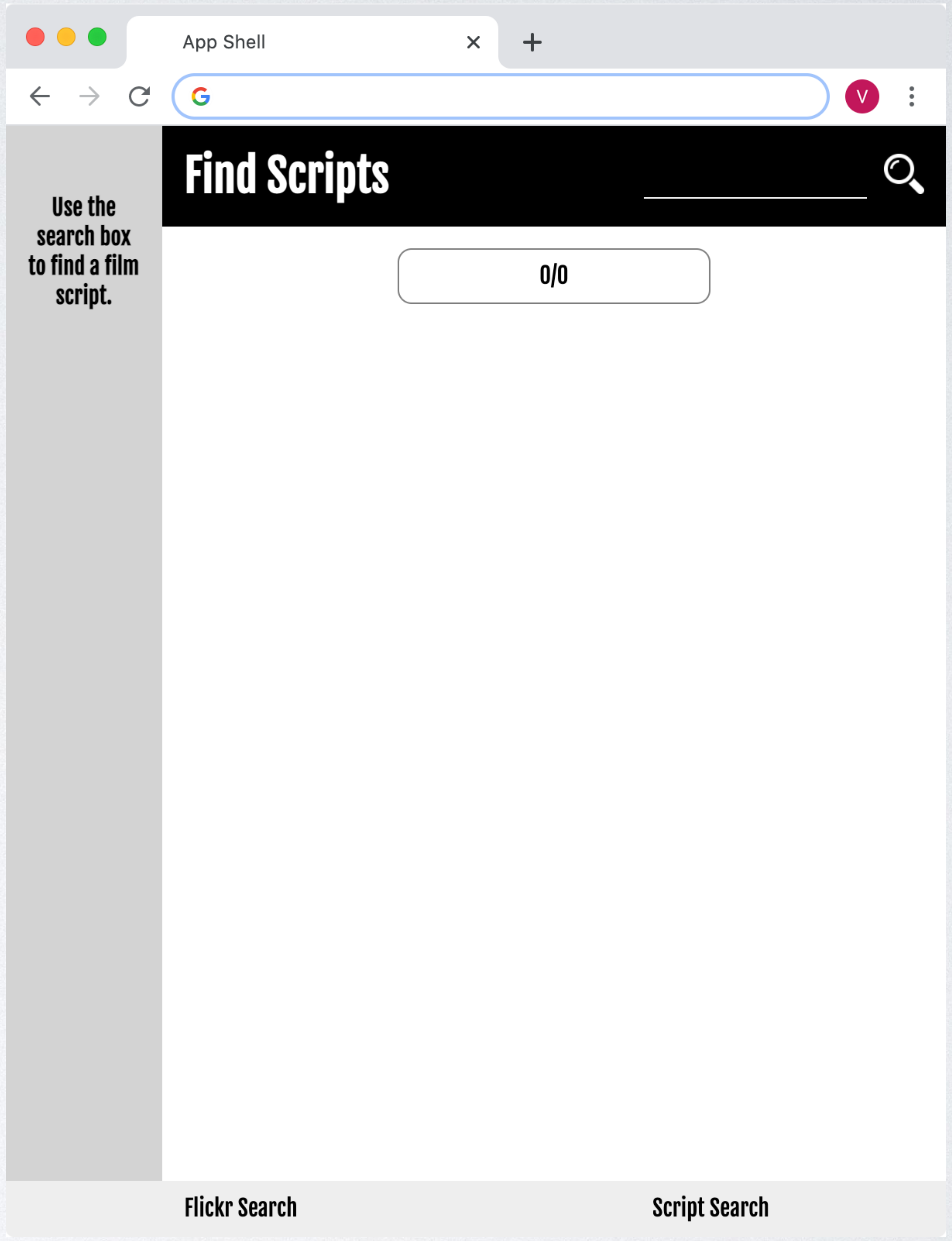
Part 5: Implement the provided Caching Policies

Part 1: App Shell

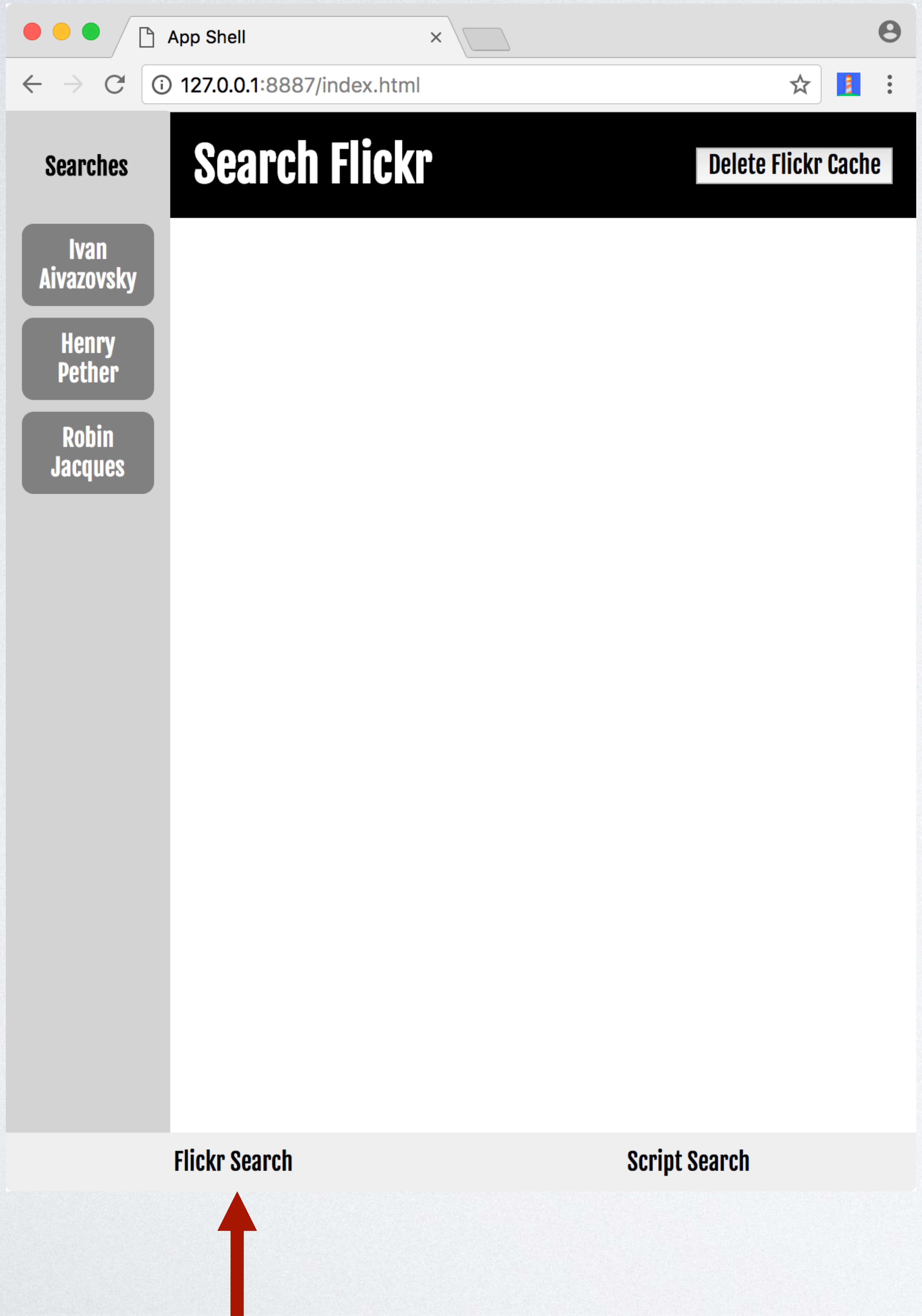
Your app will consist of two pages/screens.

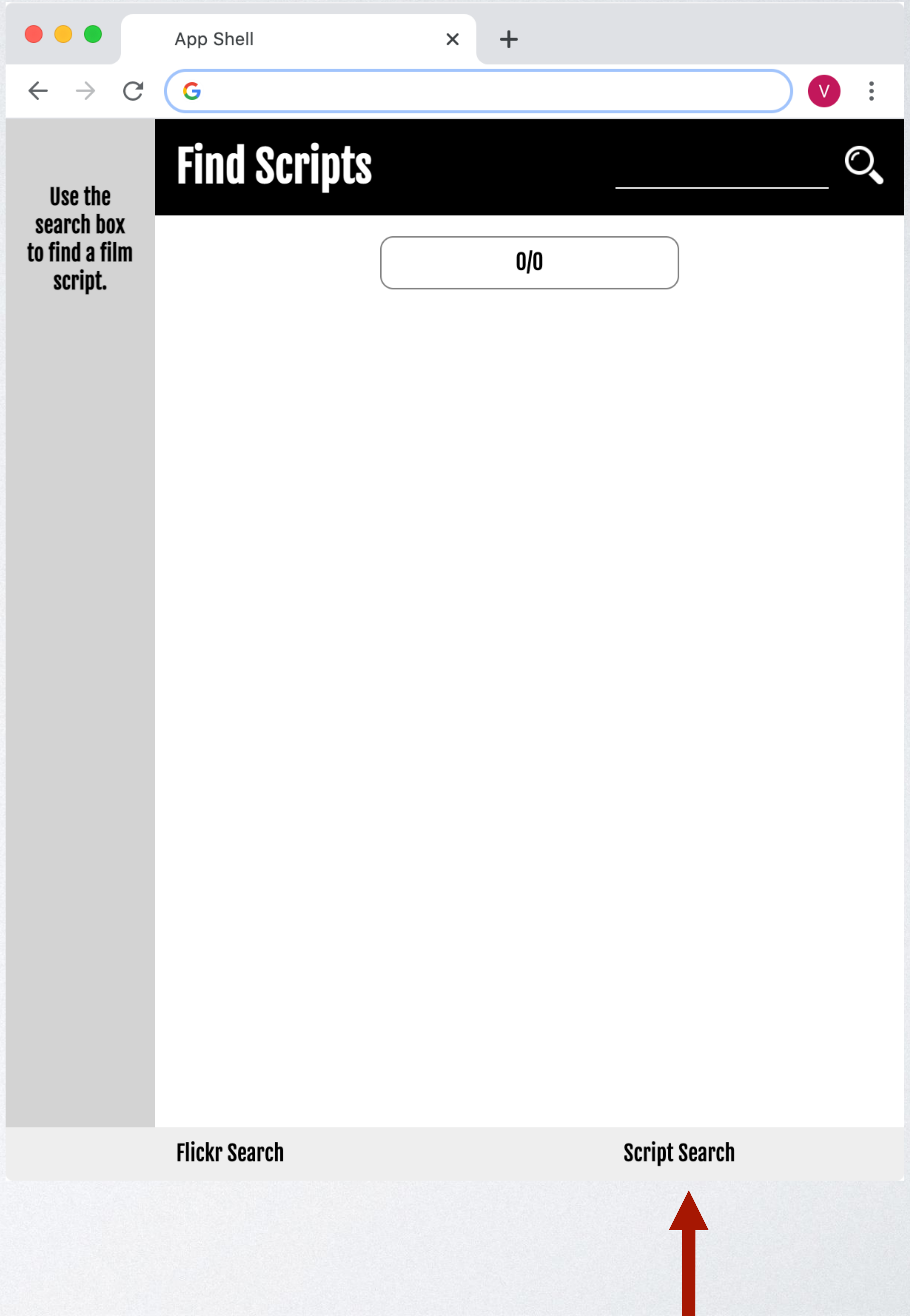
You must first create an **app shell** (based on your first assignment) that will be served from your cache, enabling your app to load if offline.





The links at the bottom of the page bring you to one of the 2 pages/screens.





Your app will have two main functions:

1) Search Flickr (using web service)

2) Search for Movie Scripts (using external JSON file)

These can be 2 separate pages.

Part 2: Search Flickr

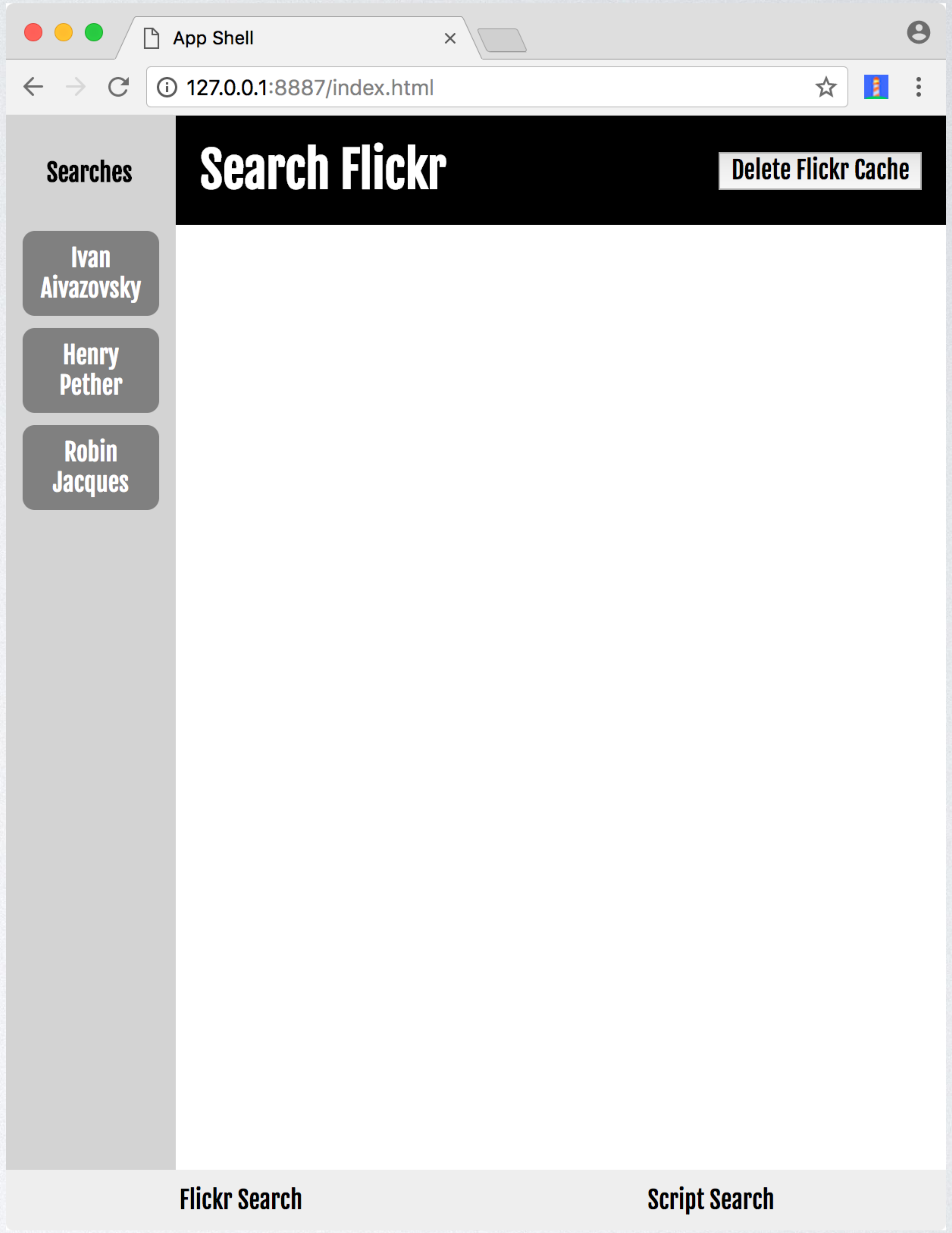
Note: slides in a red border like this can be left until you start converting your app to a PWA.

Your page should have 3 buttons that initiate searches of Flickr.

The search terms should be contained in an array and the buttons dynamically created from that array using DOM scripting.

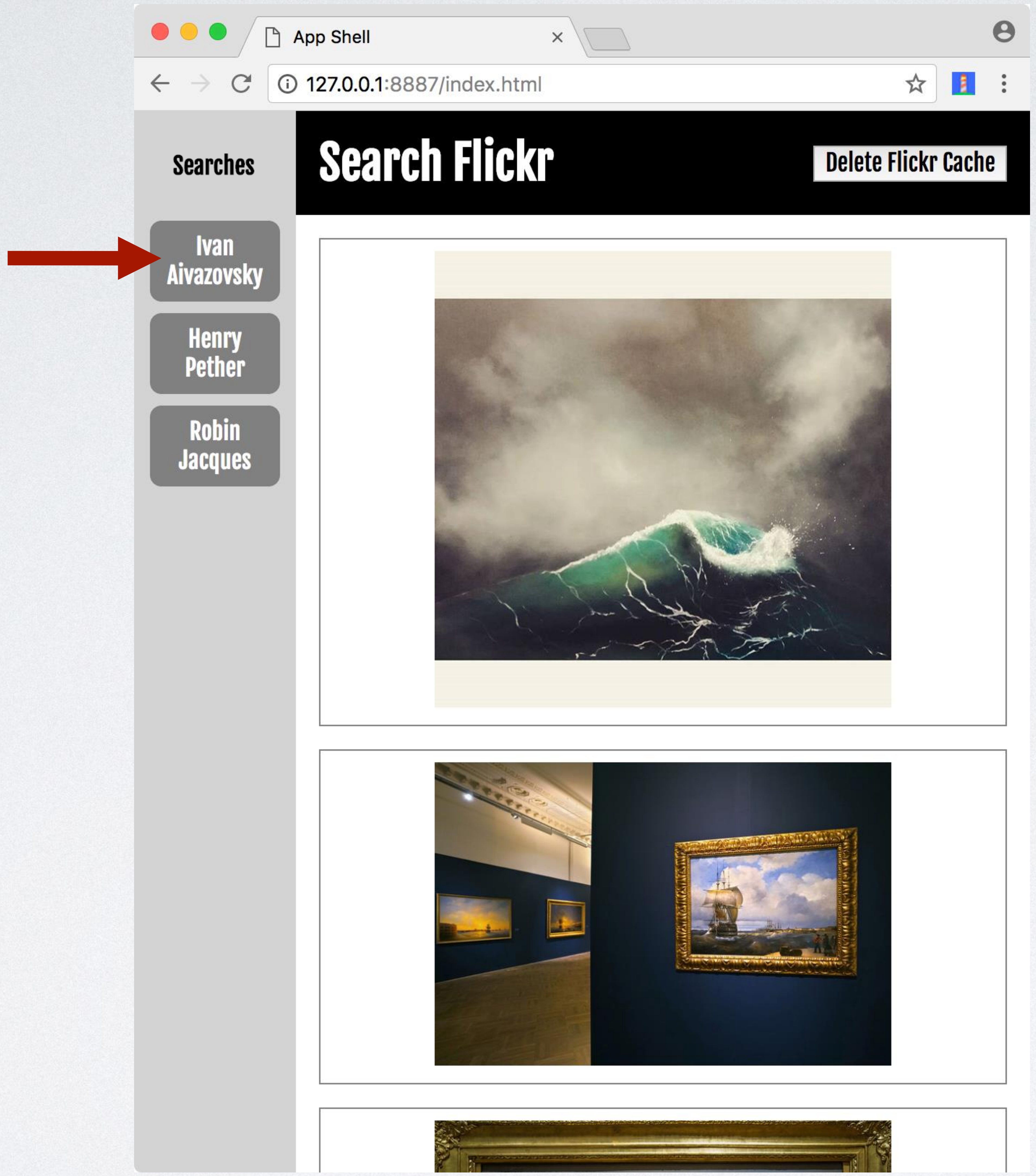
E.g.

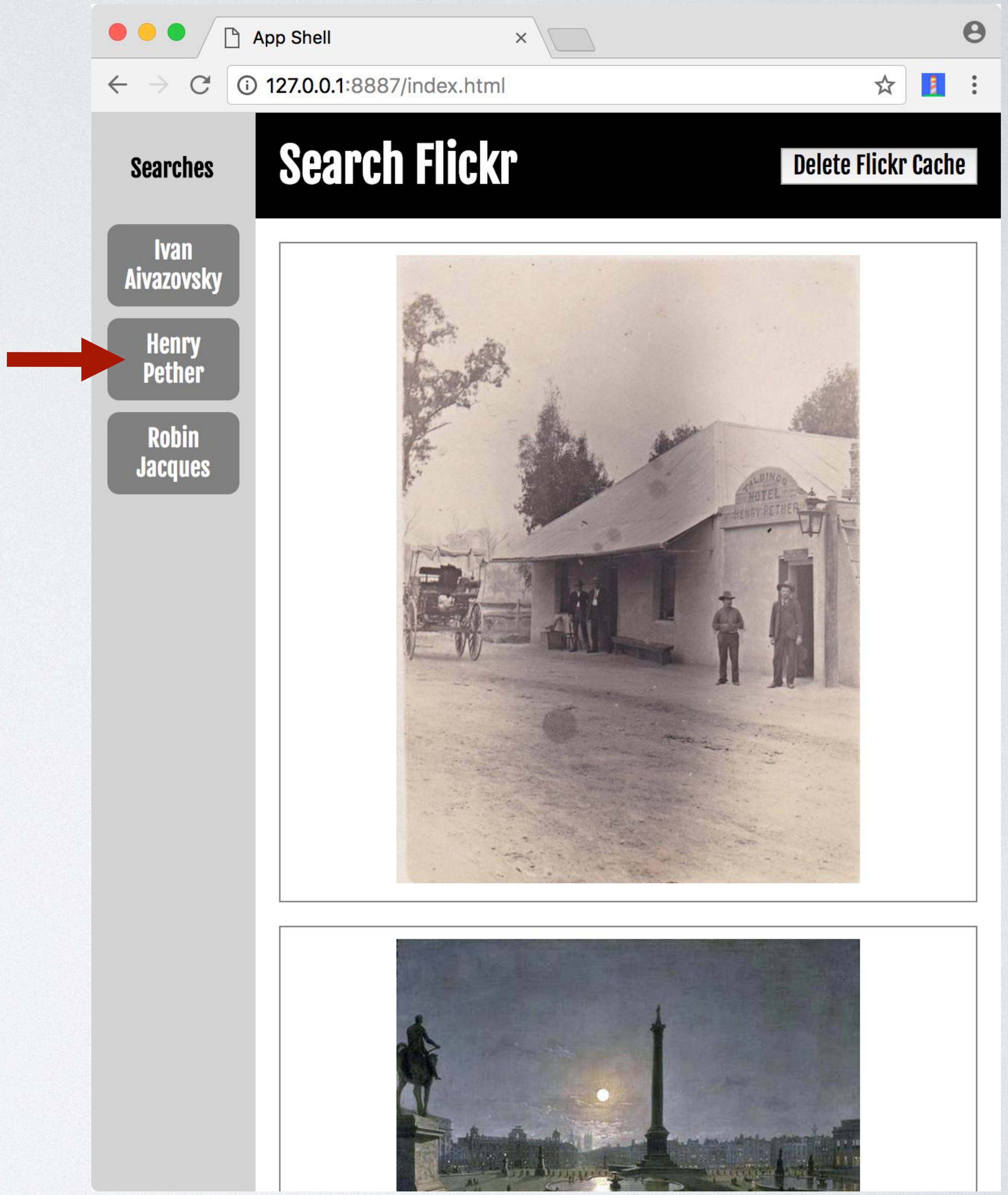
```
 var searchTerms = [
"Ivan Aivazovsky",
"Henry Pether",
"Robin Jacques"
];
```



If you click on one of the search terms you should search Flickr (with JSON-P) for that term and display the images that Flickr sends back (to a maximum of 10).

See the earlier Lab for information on searching Flickr.

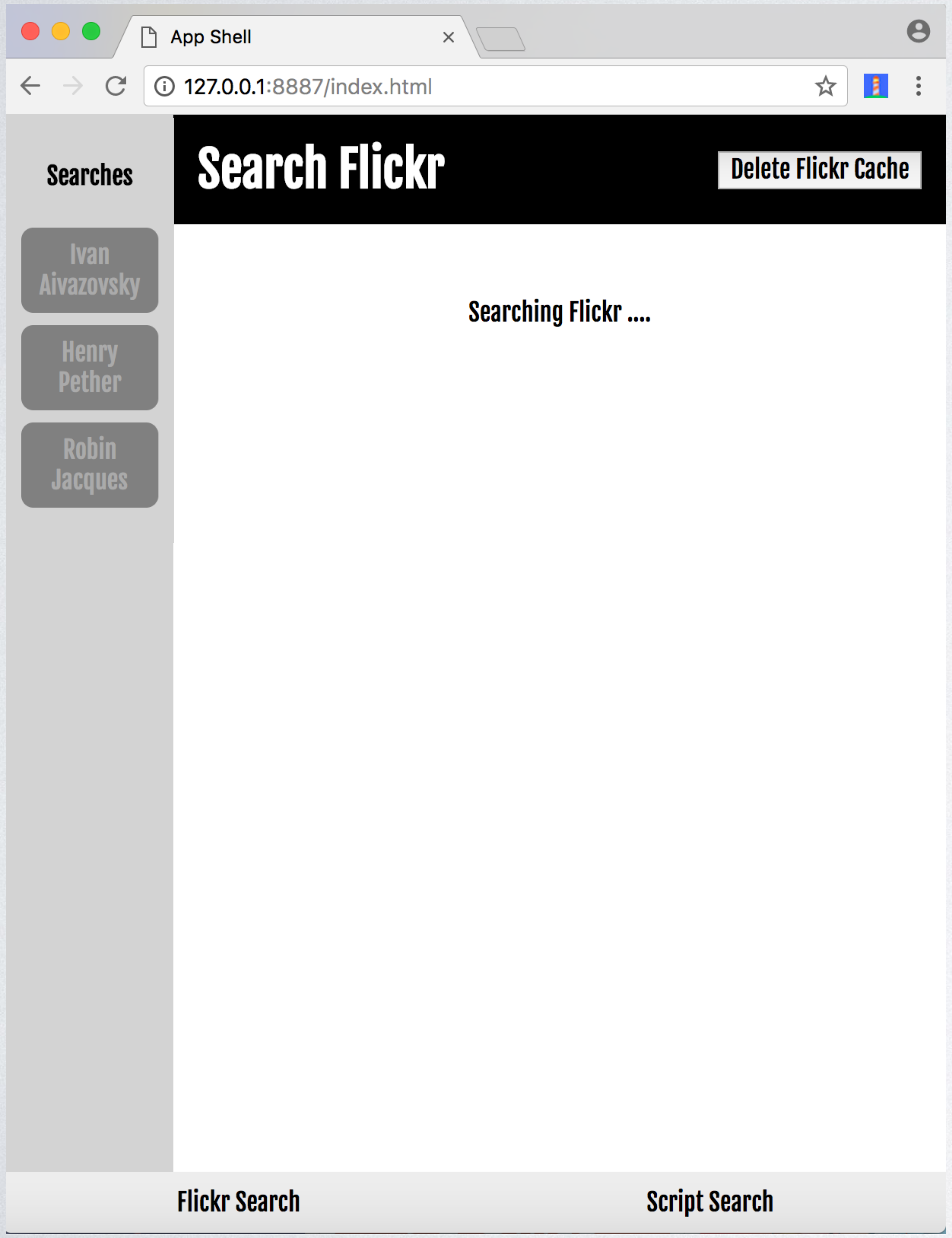


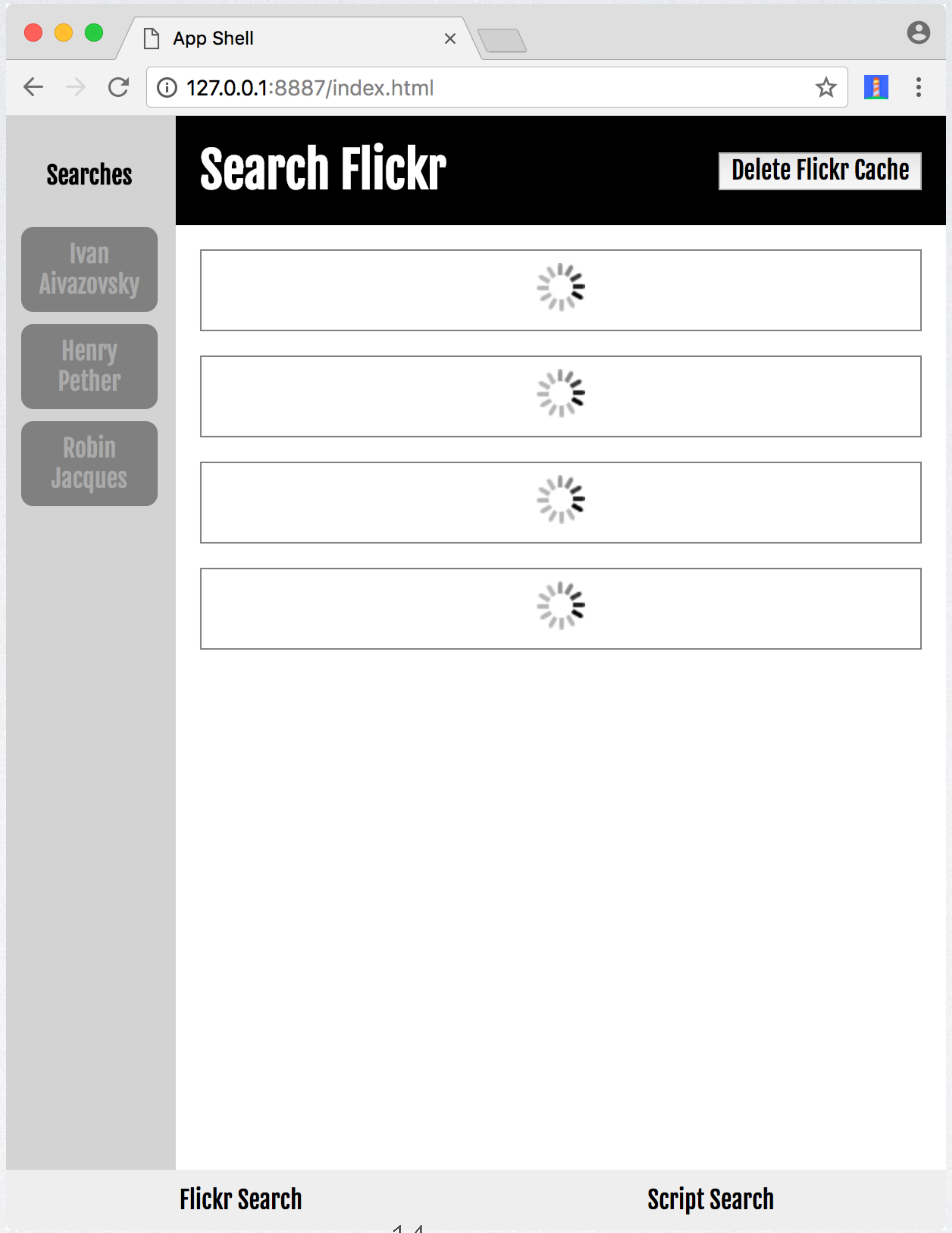


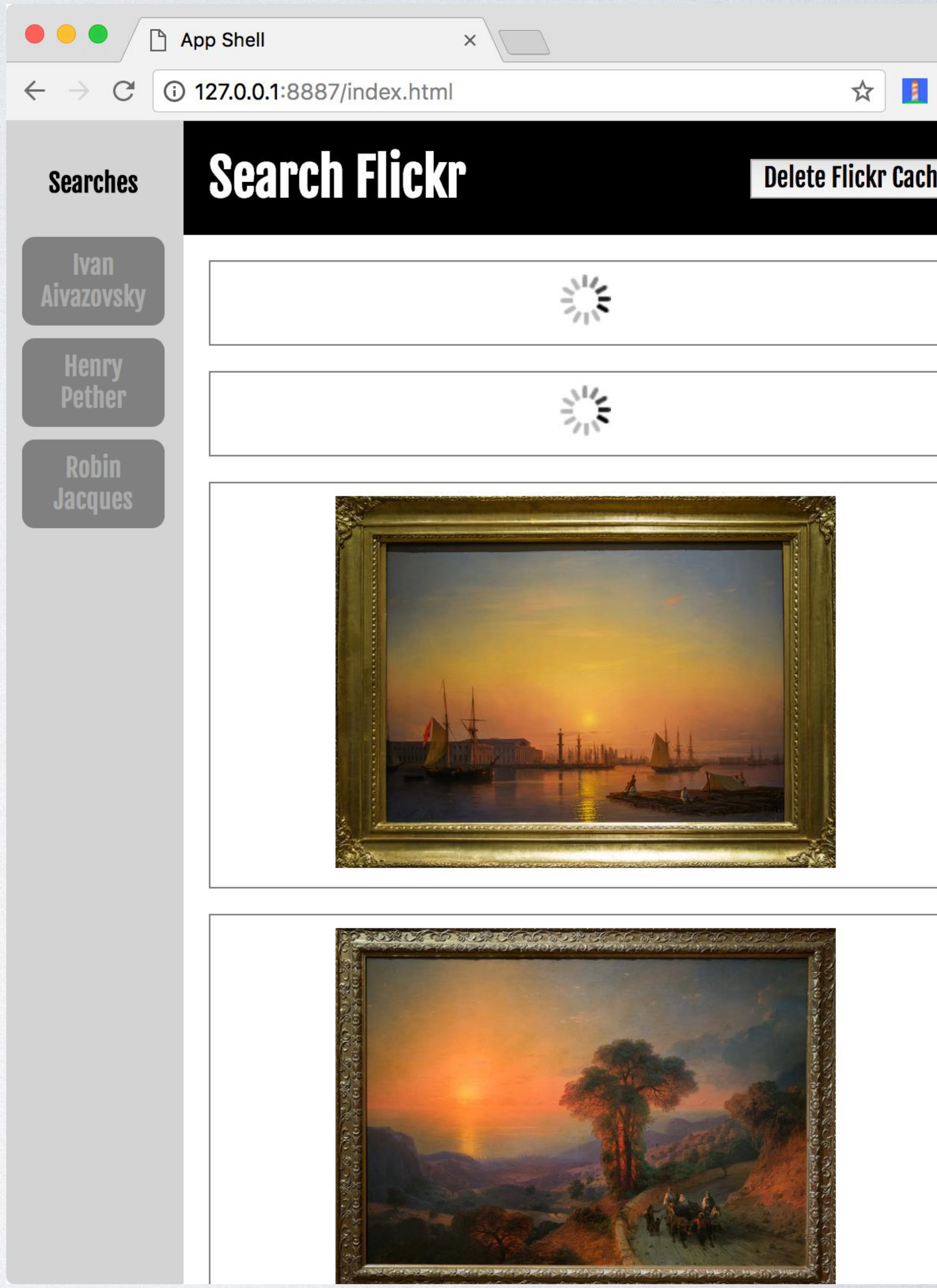
You should alert the user that the data is being retrieved.

Once you generate the URLS. each image has a loading gif displayed until it downloads fully.

The loading gifs are eventually *individually* replaced by the *fully* downloaded images.



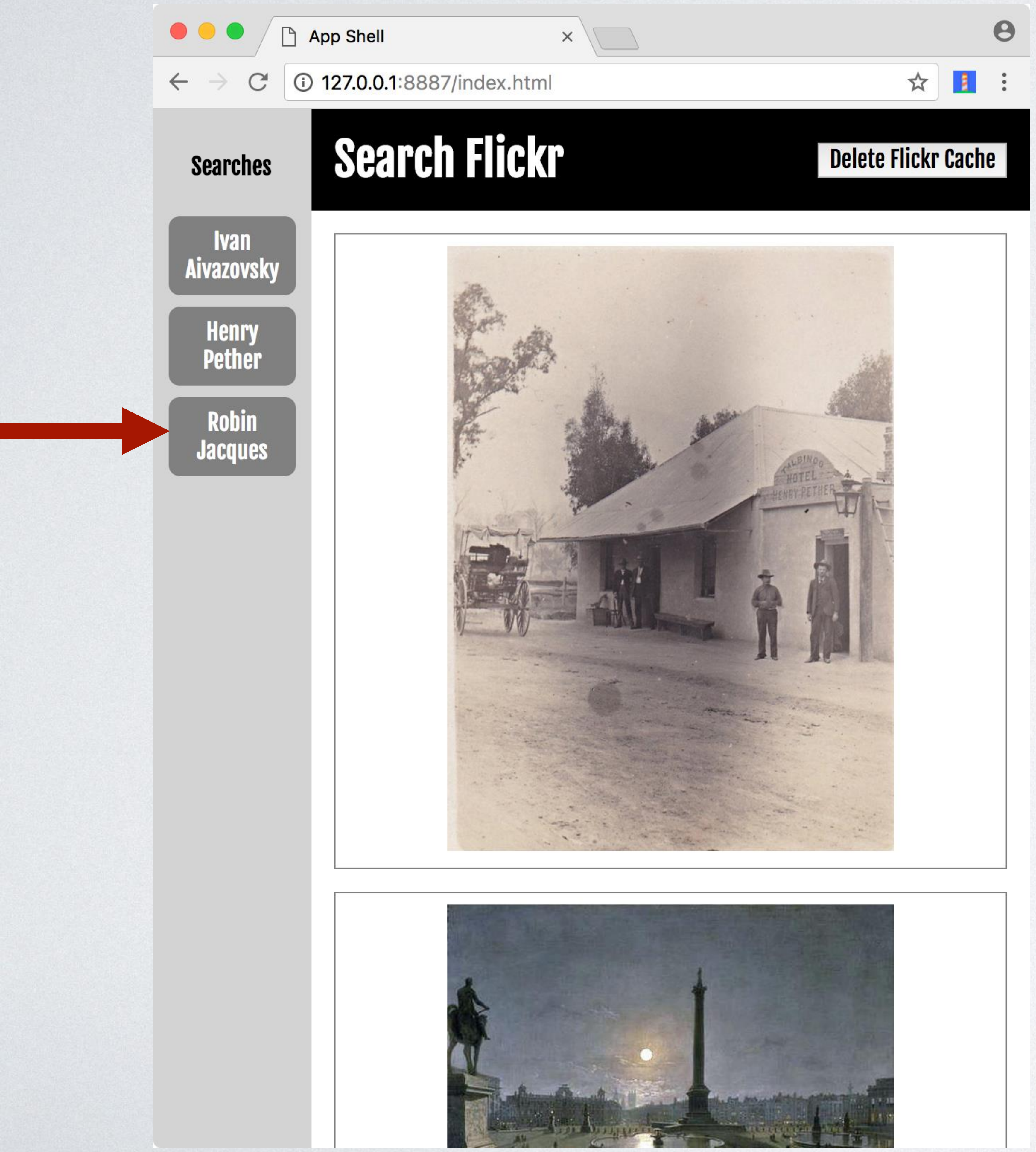


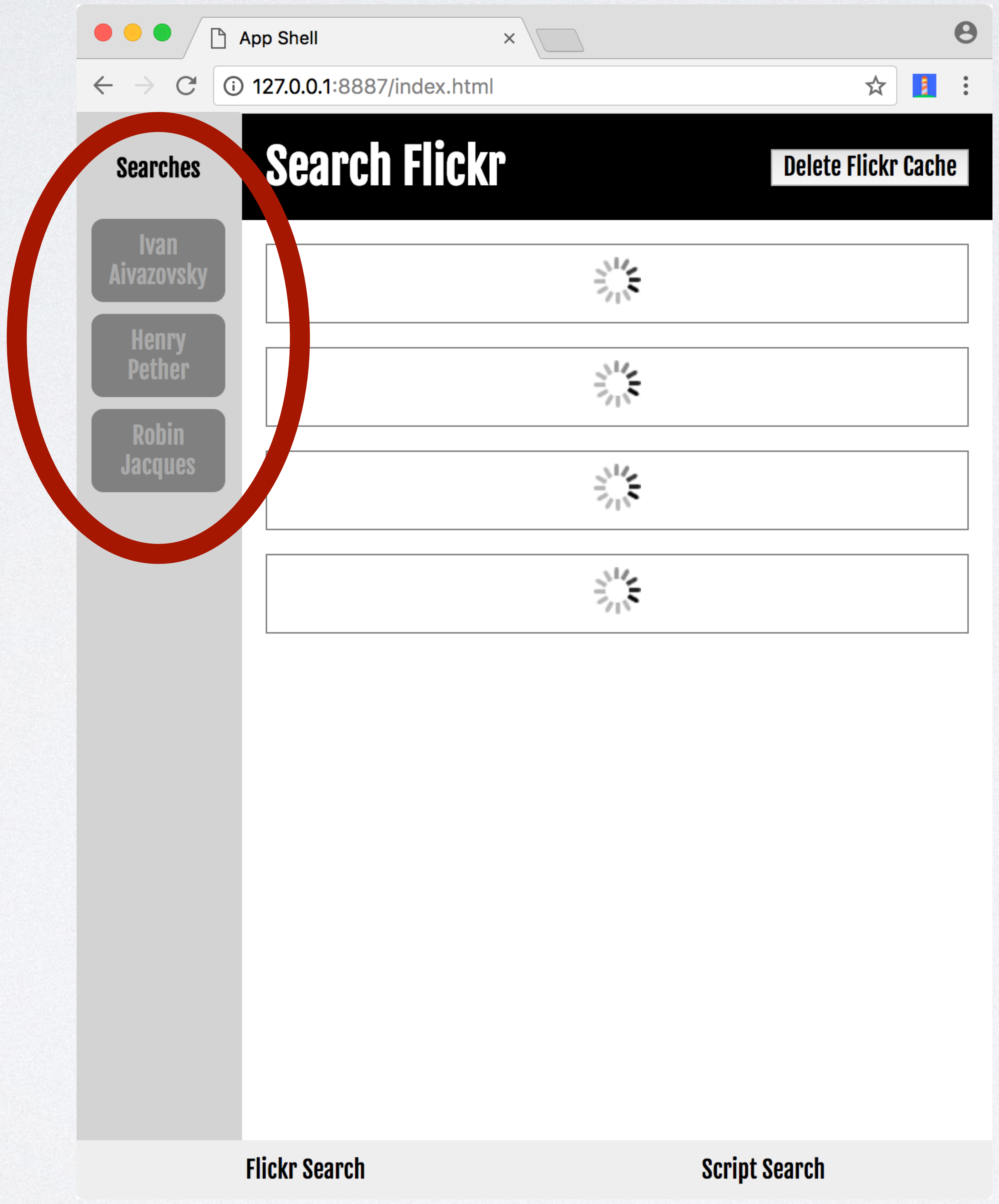


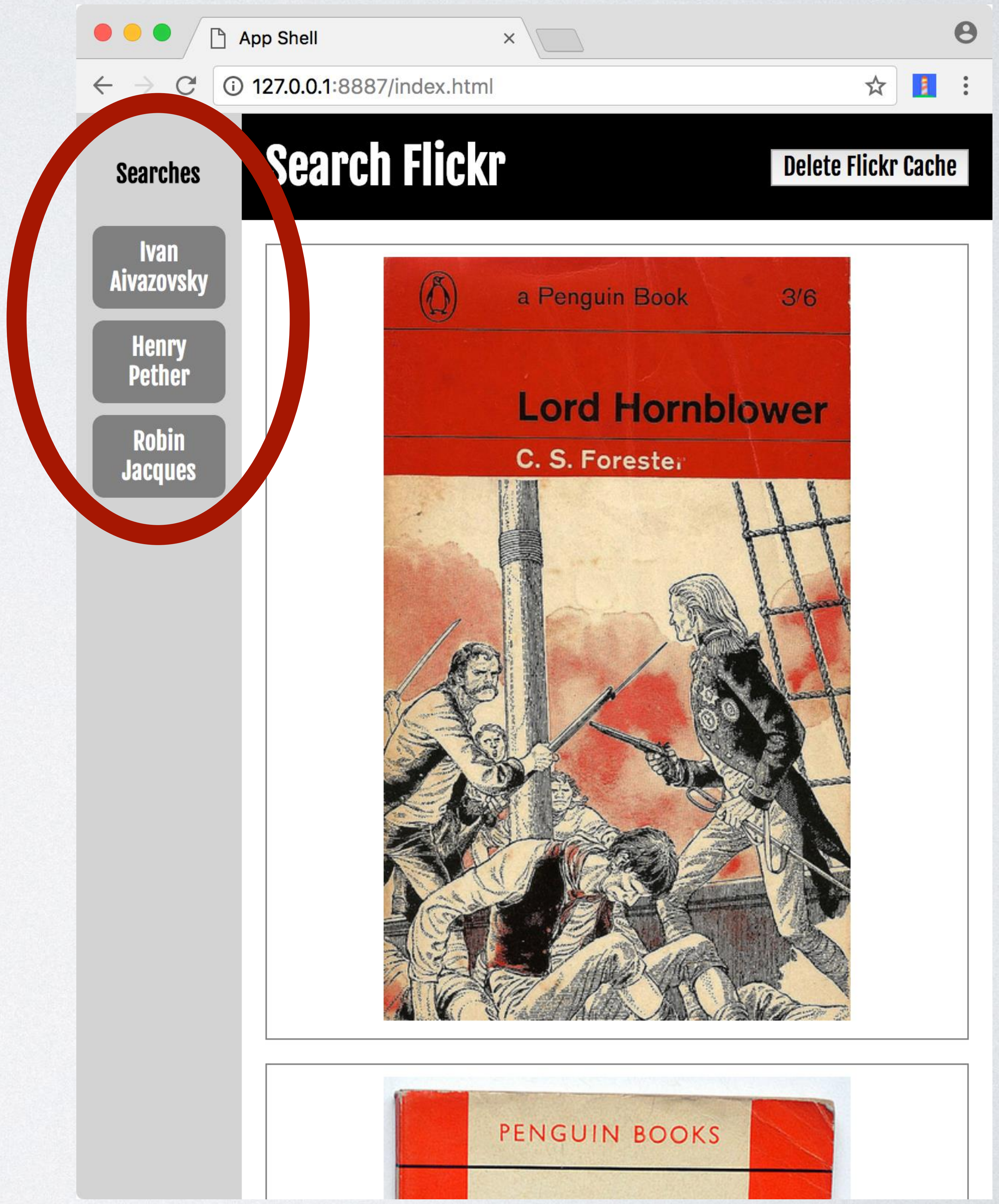
Note that the buttons are disabled when the searches started.

You must represent his visually somehow. The text is darkened in this example.

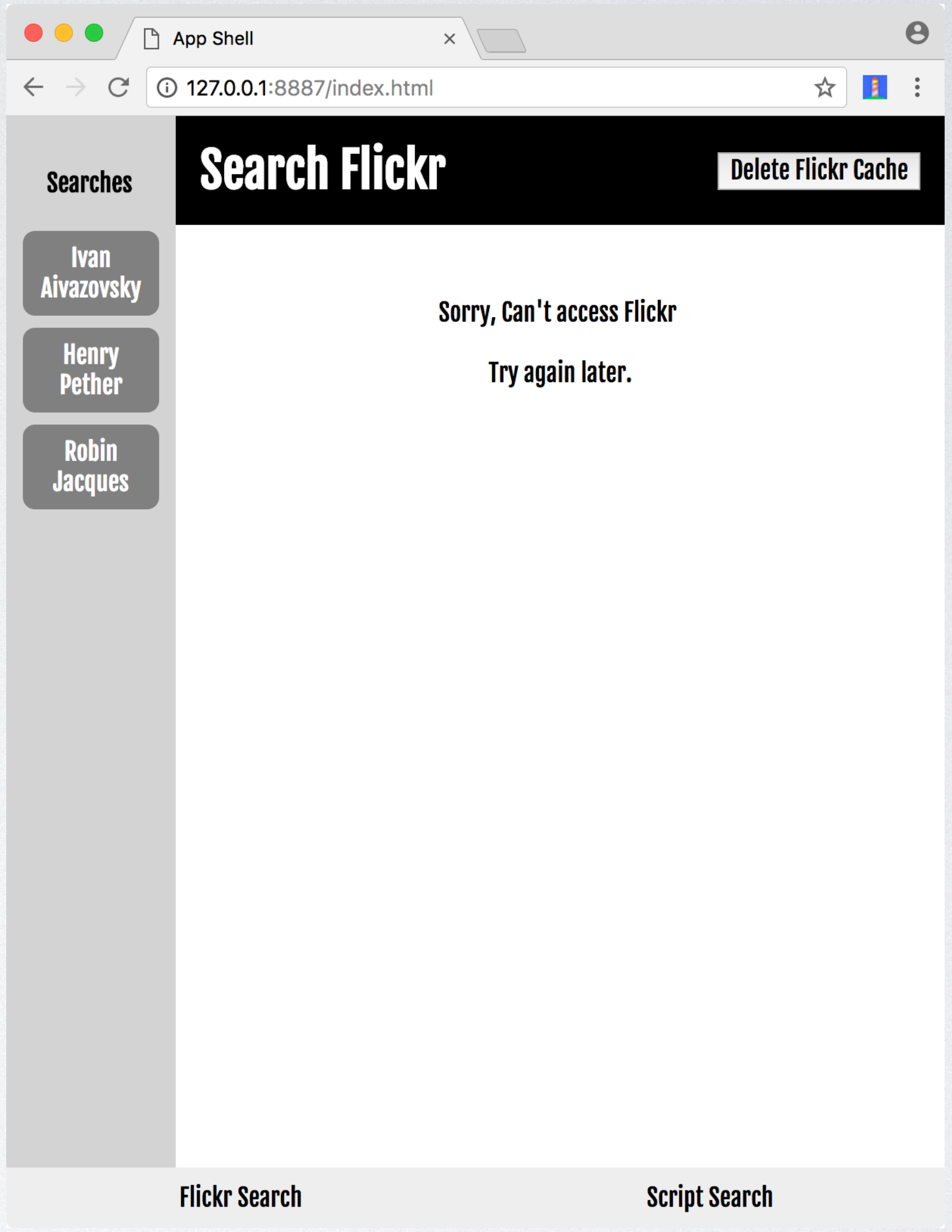
When **ALL** images have downloaded the buttons are reenabled.







If you can't access Flickr (e.g. you are offline) you must let the user know gracefully.

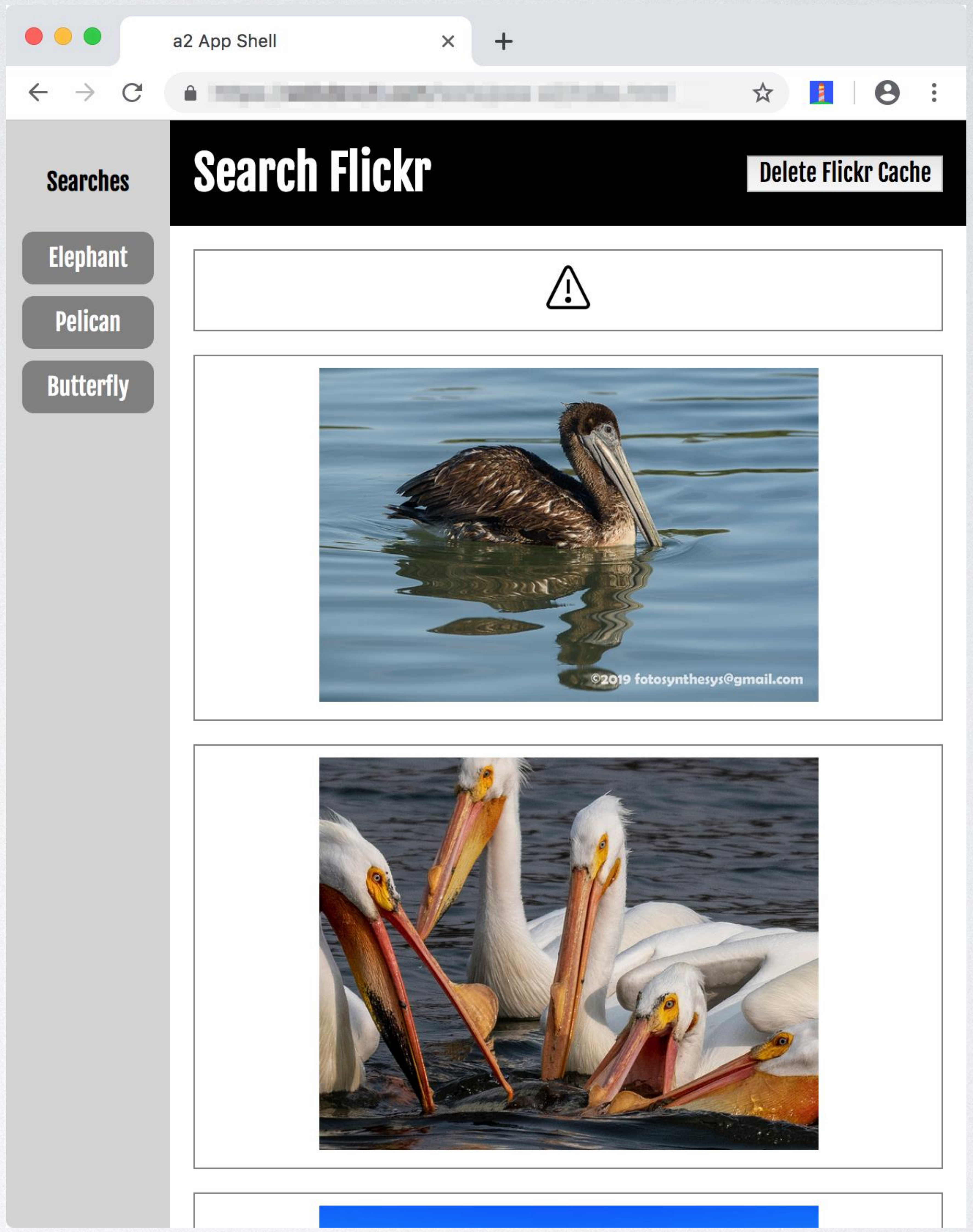


Sorry, Can't access Flickr

Try again later.

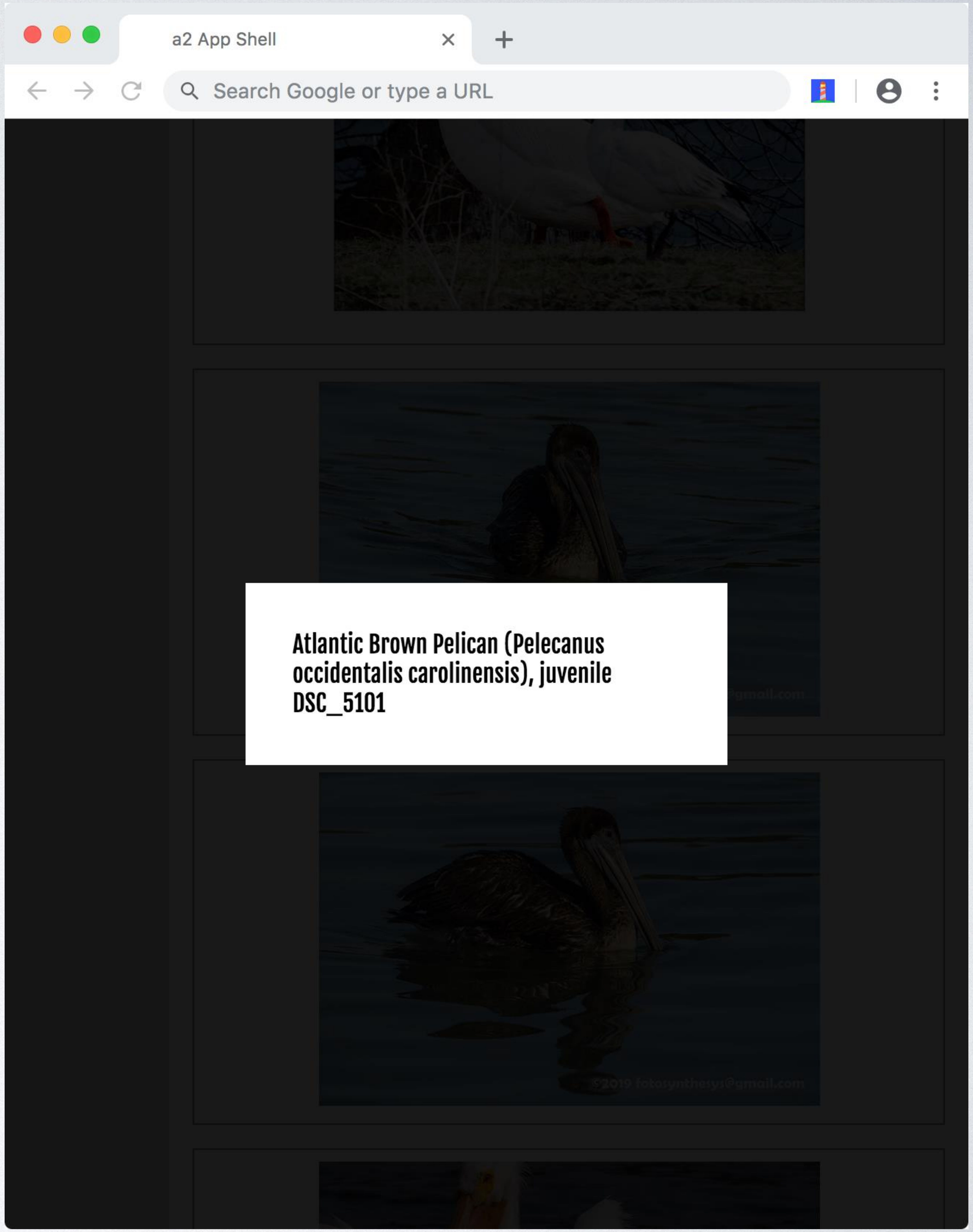
Similarly, your script should gracefully handle the cases where individual images don't load.

(I.e. they can't be accessed online and are not in the cache.)



Each image should have an **onclick** event handler so that when you click on the image it displays the title in a centered element that covers the entire window (as shown).

Clicking anywhere should hide the title and show the main page again (see screencast).



Atlantic Brown Pelican (Pelecanus occidentalis carolinensis), juvenile DSC_5101

Note: You should use Promises to detect when individual images download. And Promises.all() to detect when they have all downloaded.

Tip: You should arrange it so that If an image doesn't download **it should still resolve** (i.e. **not reject**) so that you can re-enable the buttons. (I.e. this is important because Promise.all() won't resolve if any of the Promises you pass to it reject.)

localStorage

For the purposes of this assignment we will store data in the (blocking) **localStorage** API.

(Usually we would use the non-blocking IndexedDB.)

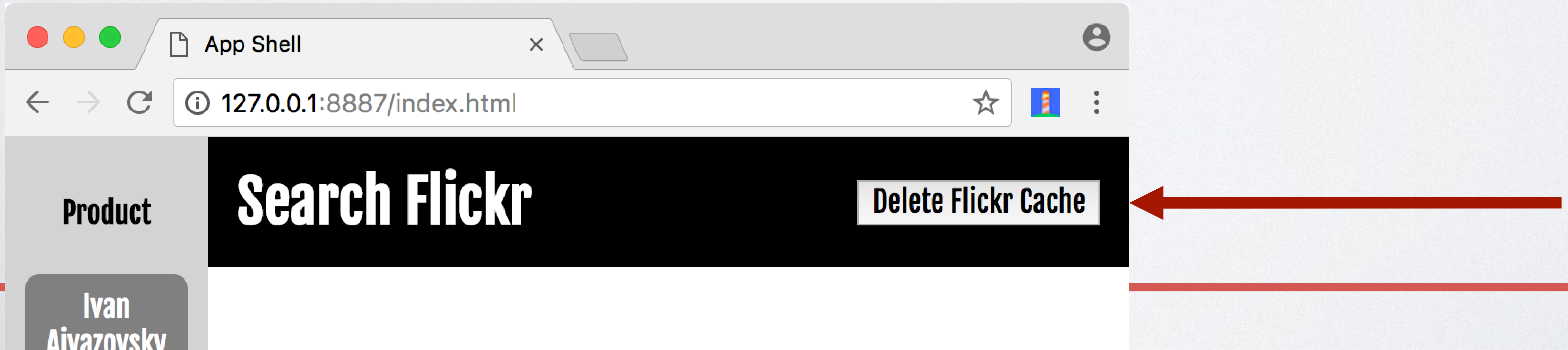
We will be storing an array of filenames of the last set of images we retrieved whenever we perform a search (i.e. we should always have the urls of the last search we performed in **localStorage**).

Since the images will be cached, we can then retrieve this array (if it is present), and add the images to the page immediately when we startup up the app (even if we are offline - since the images will be in the cache).

Managing the Cache

The Caching Policy for the app is discussed later.

But you must add a button to your page that allows you delete all the files you downloaded from Flickr (and ***only*** those files) from the cache. You shouldn't touch the other files such as those used for the app shell.



However, **you shouldn't delete the images of the last search** assuming there was one.

(You will know these images since you stored them in an array. i.e. we have the filenames stored in localStorage - See last section).

By leaving ***just*** these files in the cache there will be something on screen when users launch the app (assuming its not he first time they launched the app).

Implementation: Objects & Promises

To help implement loading the images from Flickr you should create a constructor function to create objects representing those images.

You should be able to pass the constructor a filename/url which it stores:

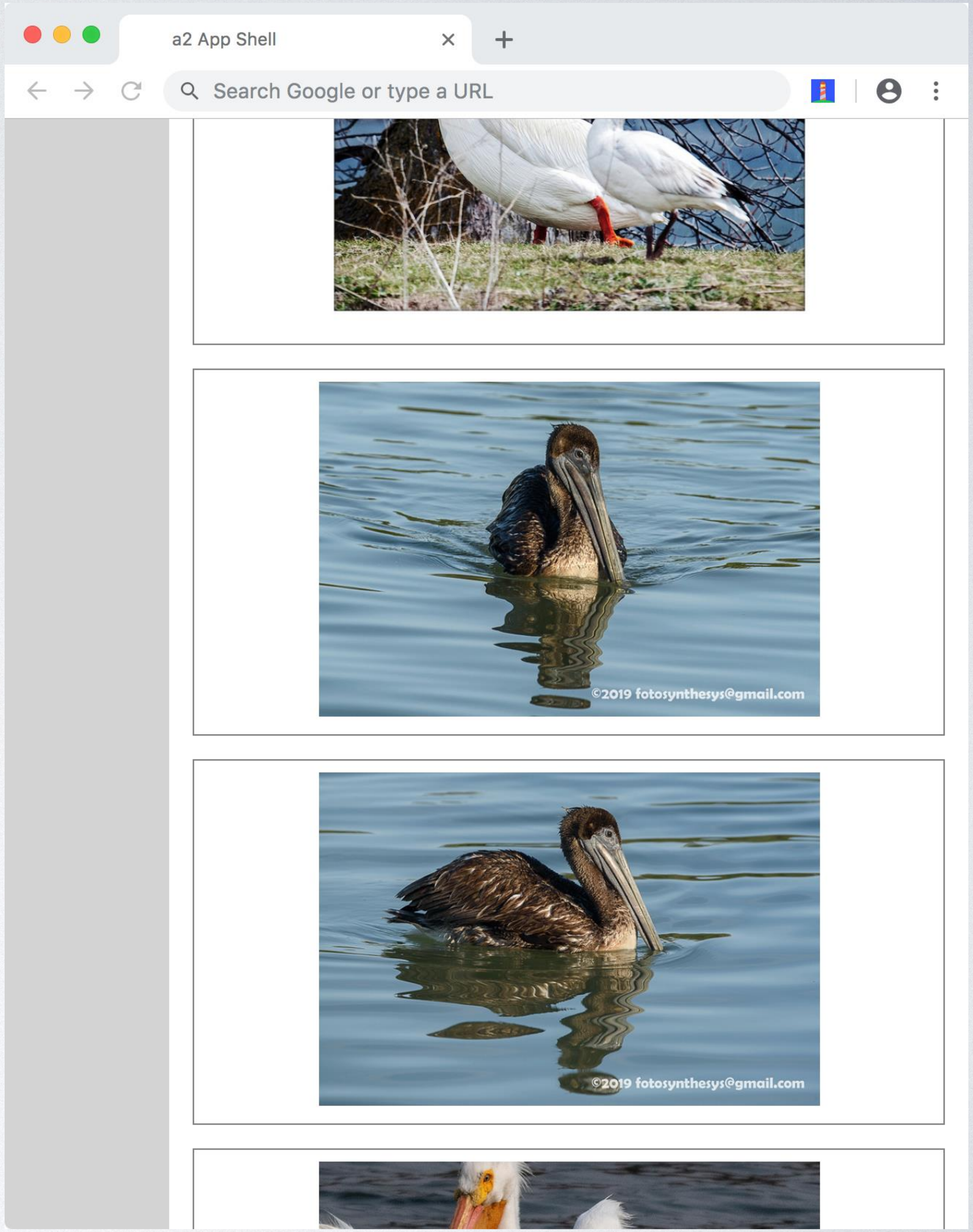
```
var x = new ImageLoader(imageName);
```

You should also be able to pass the title of the image.

```
var x = new ImageLoader(imageName, imageTitle);
```

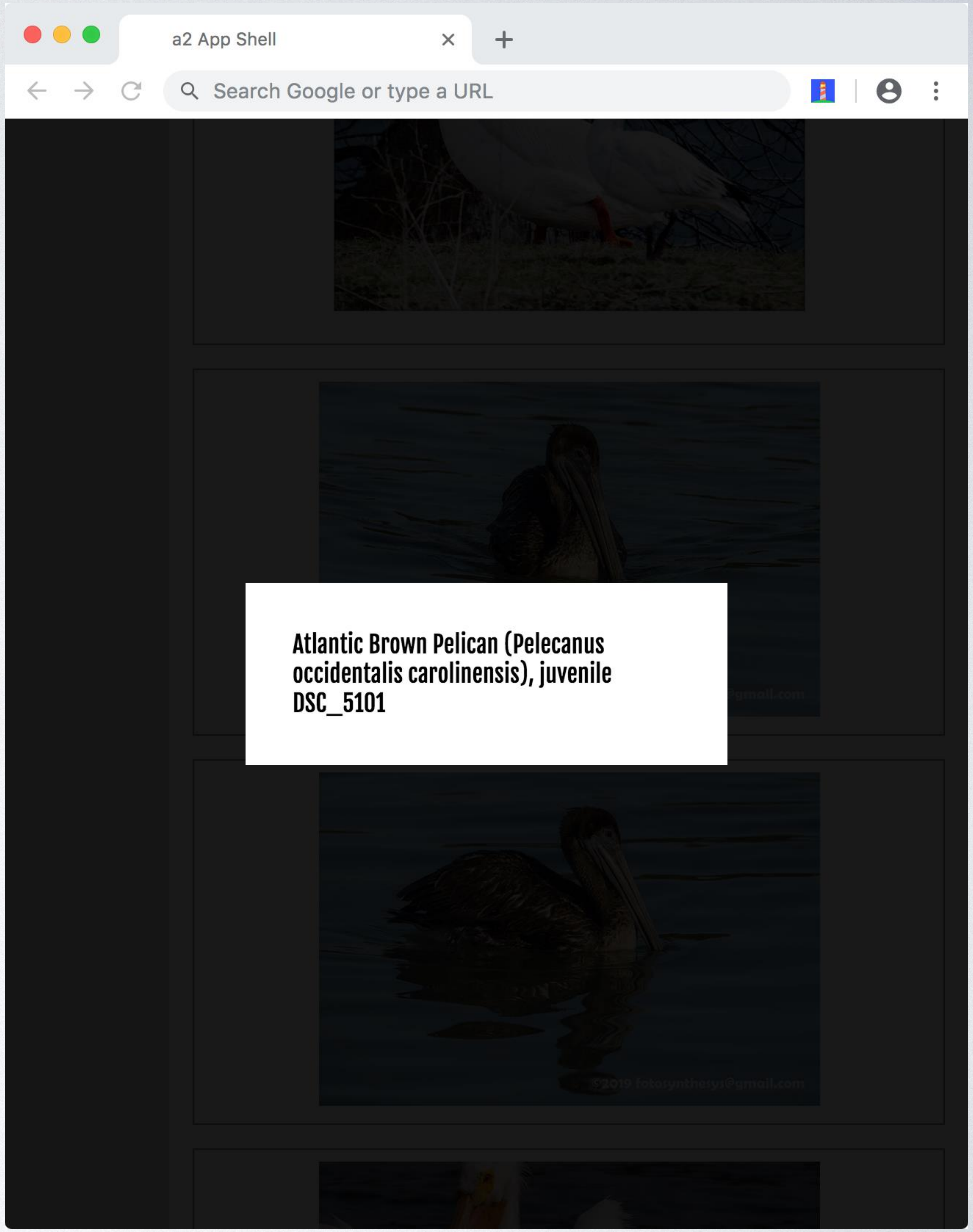
Then you can write a method that returns a Promise since it will be asynchronous. It should create an Image (DOM) Object from that filename and - once it finishes downloading - appends it to the page (and resolves the Promise).

Alternatively it could return that image object in the Promise (i.e. resolve with the Image object) when it finishes downloading.


a2 App Shell
Search Google or type a URL
©2019 fotosynthesys@gmail.com
©2019 fotosynthesys@gmail.com

Each image object should have an **onclick** event handler as discussed previously, i.e. clicking on the image displays the title. Therefore, it will need access to the image's title.

Closures may help here.



Atlantic Brown Pelican (Pelecanus occidentalis carolinensis), juvenile
DSC_5101

Part 3: Search JSON Object (of Film Scripts)

This part involves searching a large JSON object (with 9254 properties) for specific data.

It is stored in a .js file using the JSON-P format.

We will use an arbitrary data file for the purposes of this assignment.

In this case a large file of films and where to find their scripts.

https://github.com/matthewfdaniels/scripts/

https://github.com/matthewfdaniels/scripts/blob/graphs/movieObj.js

```
processFilms({ '1':
   { scrape_id: '1',
     title: '10 things i hate about you',
     source: 'cornell',
     year: '1999',
     link: 'http://www.dailyscript.com/scripts/10Things.html',
     imdb_match: 'tt0147800',
     cornellId: 'm0' },
  '2':
   { scrape_id: '2',
     title: '1492: conquest of paradise',
     source: 'cornell',
     year: '1992',
     link: 'http://www.hundland.org/scripts/1492-ConquestOfParadise.txt',
     imdb_match: 'tt0103594',
     cornellId: 'm1' },
  '3':
   { scrape_id: '3',
     title: '15 minutes',
     source: 'cornell',
     year: '2001',
     link: 'http://www.dailyscript.com/scripts/15minutes.html',
     imdb_match: 'tt0179626',
     cornellId: 'm2' },
  '4':
   { scrape_id: '4',
     title: '2001: a space odyssey',
     source: 'cornell',
     year: '1968'
```

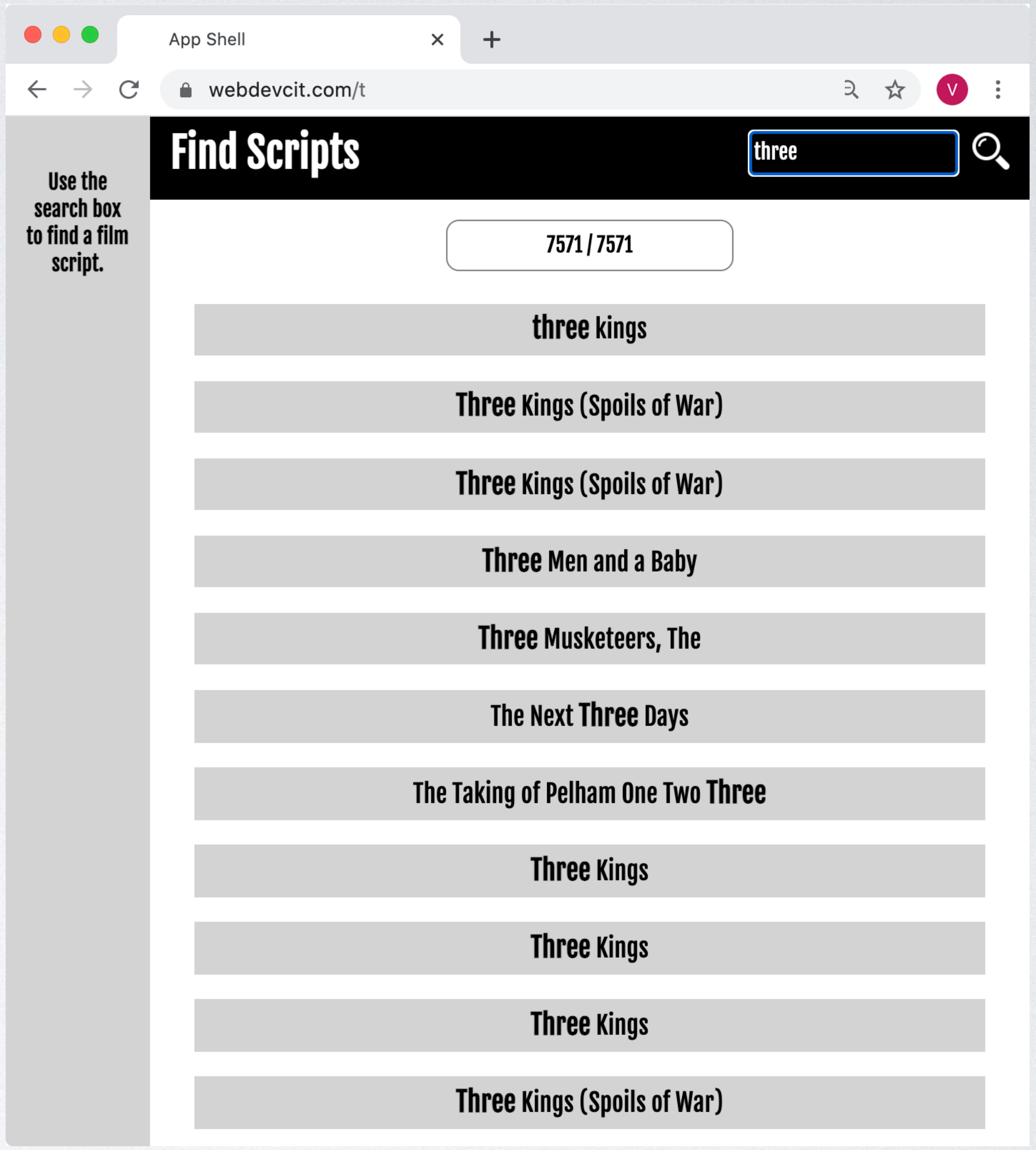
This data will be provided for you.

It was adapted slightly from the original to support **JSON-P**

```
processFilms( '1':
  { scrape_id: '1',
    title: '10 things i hate about you',
    source: 'cornell',
    year: '1999',
    link: 'http://www.dailyscript.com/scripts/10Things.html',
    imdb_match: 'tt0147800',
    cornellId: 'm0' },
 '2':
  { scrape_id: '2',
    title: '1492: conquest of paradise',
    source: 'cornell',
    year: '1992',
    link: 'http://www.hundland.org/scripts/1492-ConquestOfParadise.txt',
    imdb_match: 'tt0103594',
    cornellId: 'm1' },
 '3':
  { scrape_id: '3',
    title: '15 minutes',
    source: 'cornell',
    year: '2001',
    link: 'http://www.dailyscript.com/scripts/15minutes.html',
    imdb_match: 'tt0179626',
    cornellId: 'm2' },
 '4':
  { scrape_id: '4',
    title: '2001: a space odyssey',
    source: 'cornell',
```

I.e. the function call to **processFilms()** was added.

You can adapt the app shell HTML/CSS from assignment I as before.


App Shell

webdevcit.com/t

Find Scripts

three

Use the search box to find a film script.

7571 / 7571

three kings

Three Kings (Spoils of War)

Three Kings (Spoils of War)

Three Men and a Baby

Three Musketeers, The

The Next Three Days

The Taking of Pelham One Two Three

Three Kings

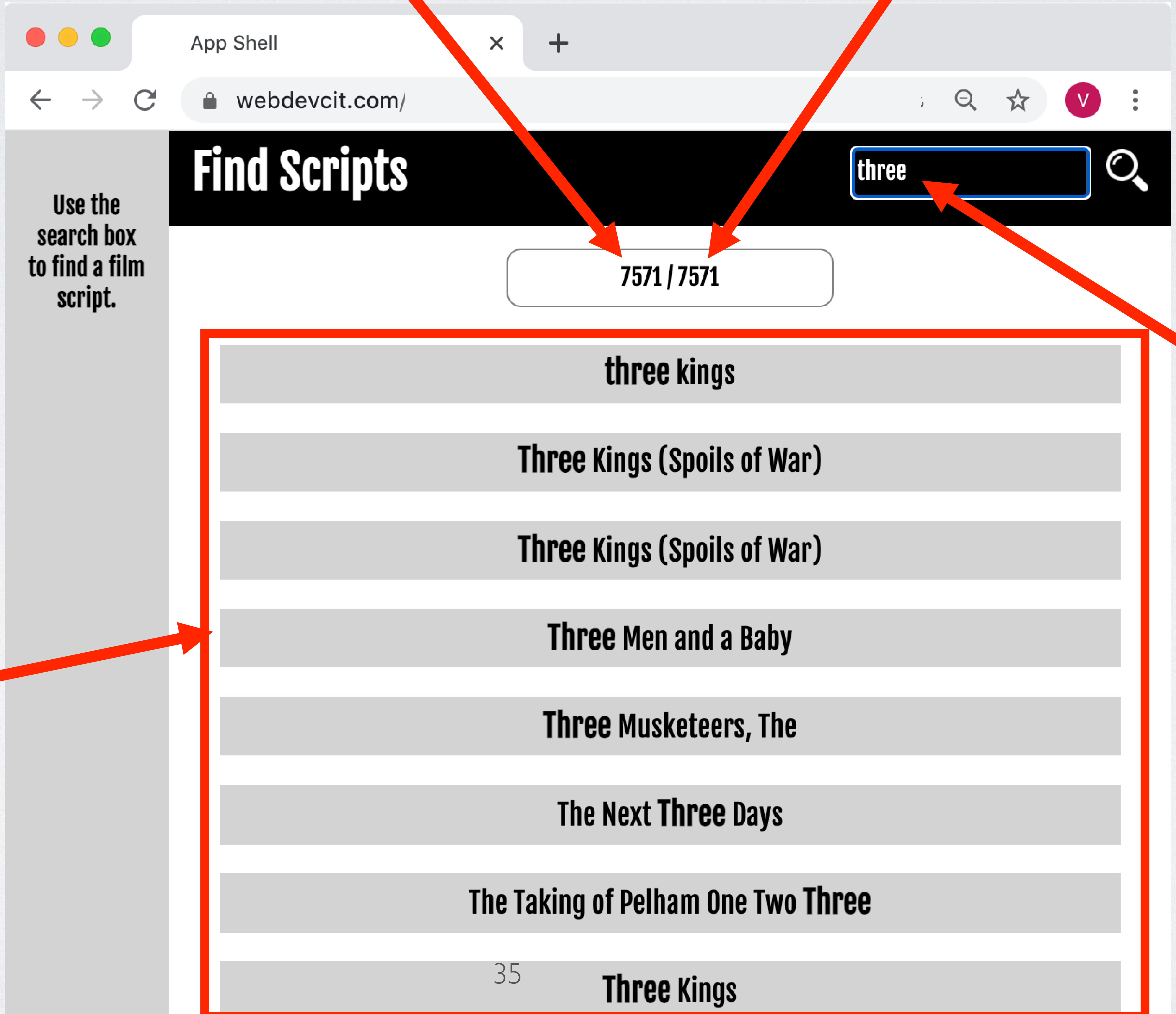
Three Kings

Three Kings

Three Kings (Spoils of War)

Total Number of items that have been processed (i.e. searched through) currently

Total number of items to be processed


App Shell

webdevcit.com/

Find Scripts

three

Use the search box to find a film script.

7571 / 7571

three kings

Three Kings (Spoils of War)

Three Kings (Spoils of War)

Three Men and a Baby

Three Musketeers, The

The Next Three Days

The Taking of Pelham One Two Three

Three Kings

Search Term

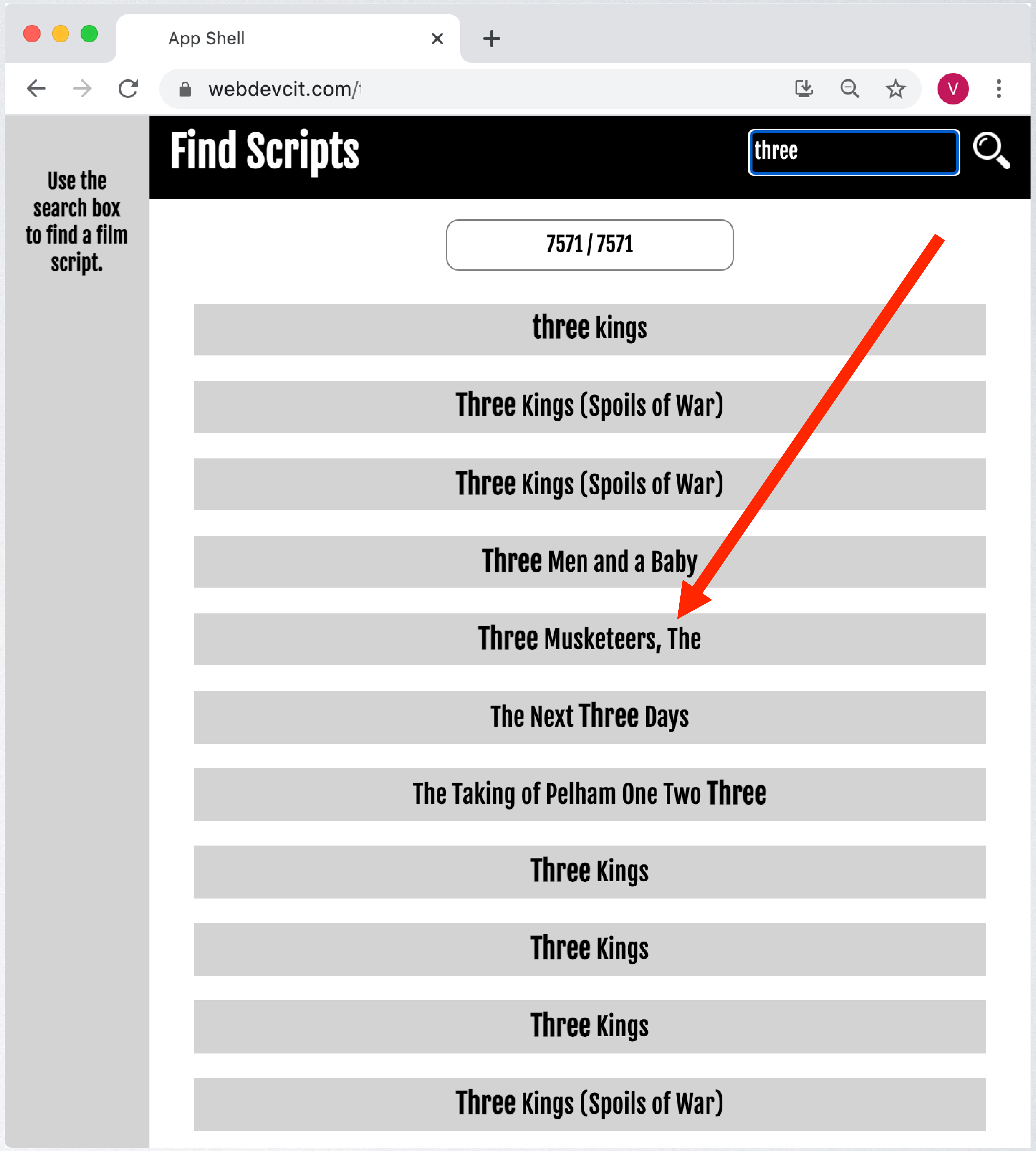
Results

The results are added to the page via DOM scripting as buttons (or similar elements).

If you click on a button it will open a new window/tab with that script.

```
window.open(url);
```

Note, not all urls in the dataset are still live. That is not an issue for this assignment.



Find Scripts

Use the search box to find a film script.

7571 / 7571

three kings

Three Kings (Spoils of War)

Three Kings (Spoils of War)

Three Men and a Baby

Three Musketeers, The

The Next Three Days

The Taking of Pelham One Two Three

Three Kings

Three Kings

Three Kings

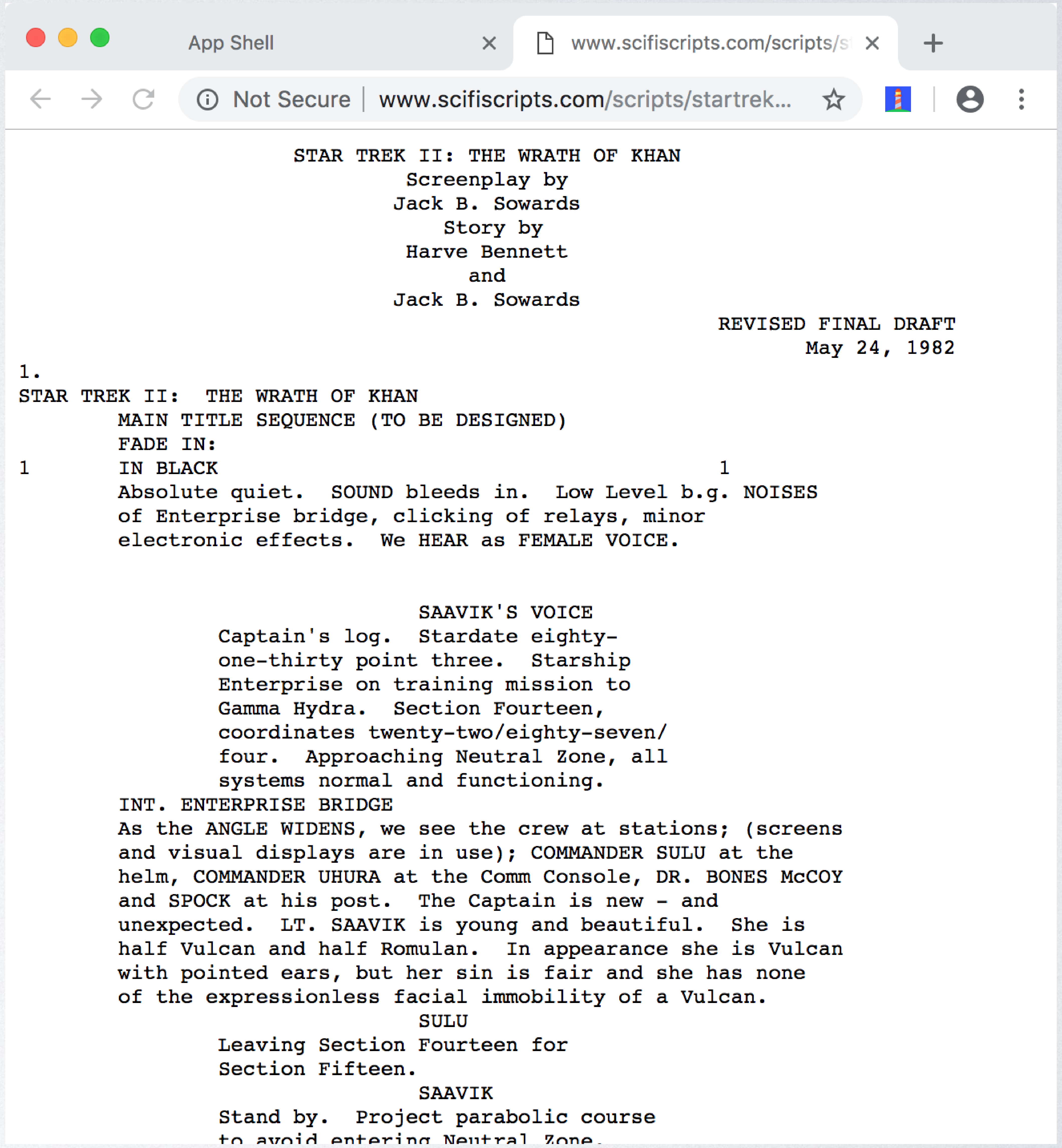
Three Kings (Spoils of War)

The results are added to the page via DOM scripting as buttons (or similar elements).

If you click on a button it will open a new window/tab with that script.

`window.open(url);`

Note, not all urls in the dataset are still live. That is not an issue for this assignment.


App Shell
www.scifiscripts.com/scripts/s
Not Secure
www.scifiscripts.com/scripts/startrek...

```
                         STAR TREK II: THE WRATH OF KHAN
                                  Screenplay by
                                 Jack B. Sowards
                                    Story by
                                  Harve Bennett
                                       and
                                 Jack B. Sowards
                                                          REVISED FINAL DRAFT
                                                                 May 24, 1982
1.
STAR TREK II:  THE WRATH OF KHAN
        MAIN TITLE SEQUENCE (TO BE DESIGNED)
        FADE IN:
1       IN BLACK                                        1
        Absolute quiet.  SOUND bleeds in.  Low Level b.g. NOISES
        of Enterprise bridge, clicking of relays, minor
        electronic effects.  We HEAR as FEMALE VOICE.


                                SAAVIK'S VOICE
                Captain's log.  Stardate eighty-
                one-thirty point three.  Starship
                Enterprise on training mission to
                Gamma Hydra.  Section Fourteen,
                coordinates twenty-two/eighty-seven/
                four.  Approaching Neutral Zone, all
                systems normal and functioning.
        INT. ENTERPRISE BRIDGE
        As the ANGLE WIDENS, we see the crew at stations; (screens
        and visual displays are in use); COMMANDER SULU at the
        helm, COMMANDER UHURA at the Comm Console, DR. BONES McCOY
        and SPOCK at his post.  The Captain is new - and
        unexpected.  LT. SAAVIK is young and beautiful.  She is
        half Vulcan and half Romulan.  In appearance she is Vulcan
        with pointed ears, but her sin is fair and she has none
        of the expressionless facial immobility of a Vulcan.
                                SULU
                Leaving Section Fourteen for
                Section Fifteen.
                                SAAVIK
                Stand by.  Project parabolic course
```

The details (the film names and URLs) come from the open data source we mentioned.

When **searching the data** you should check if the search term is contained in the title of a film (looping through all 7500+ films).

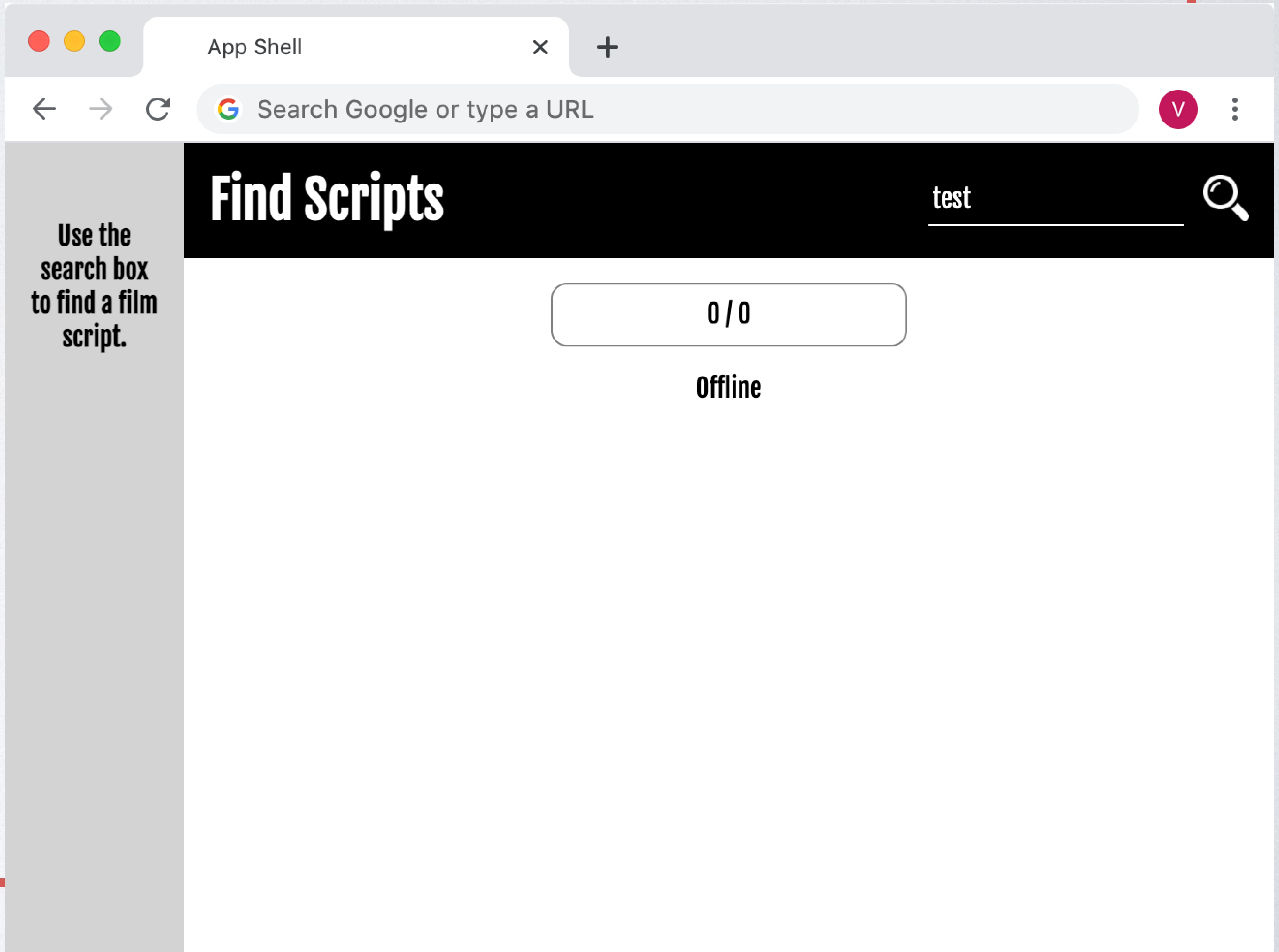
If you find a match you should arrange for the **title** and **url** to be added to the page.

0

The page should still appear when offline.

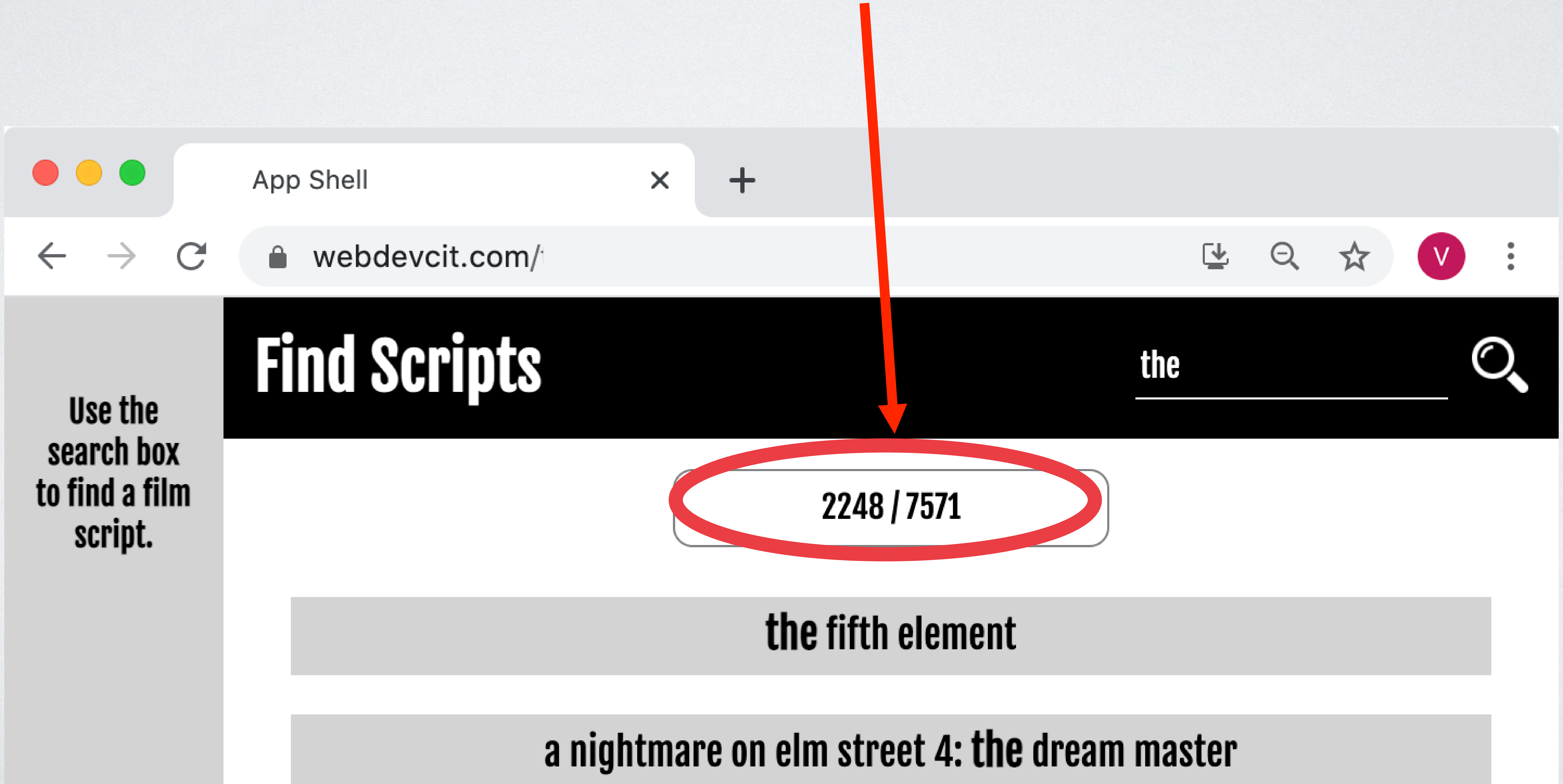
However **if you try a search** it should **inform you that it is offline** instead of showing results.

NOTE: we could have cached the JSON-P file but we are emulating a web service so **we will deliberately avoid using a cached version**.


App Shell
Search Google or type a URL
V
Find Scripts
test
Use the search box to find a film script.
0 / 0
Offline

As you perform the search (and dynamically add the result buttons), you show how many of the movies you have already searched on the page.

See included screencasts.



Find Scripts

the

Use the search box to find a film script.

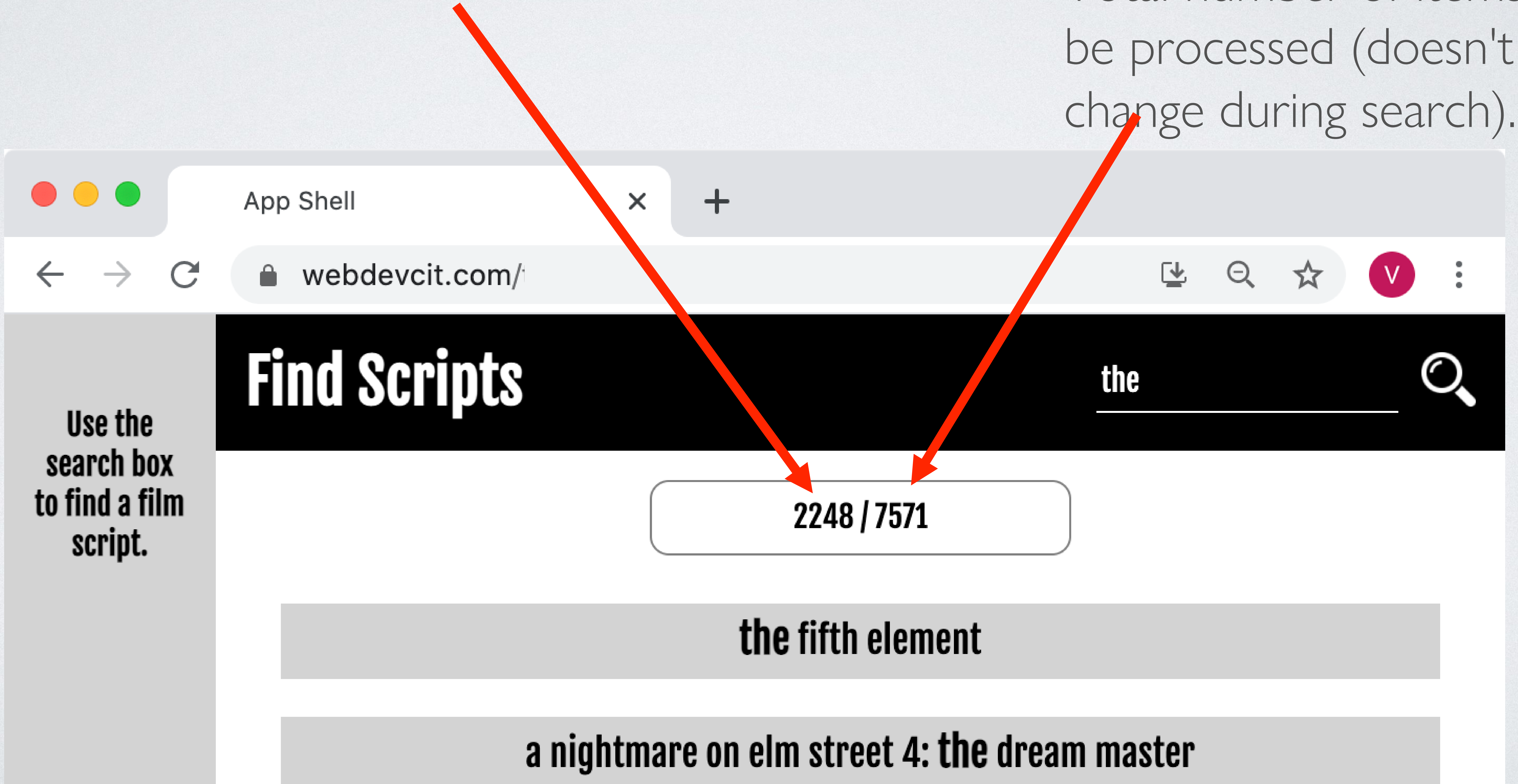
2248 / 7571

the fifth element

a nightmare on elm street 4: the dream master

Total Number of items that have been processed (i.e. searched through) currently (changes during search)

Total number of items to be processed (doesn't change during search).



Find Scripts

the

Use the search box to find a film script.

2248 / 7571

the fifth element

a nightmare on elm street 4: the dream master

Since the constantly changing search count will be making a lot of use of the single thread in your page (to update the UI) **you will search the file in a separate thread via web workers.**

The web worker should do the following in response to a search term sent to it:

Automatically **search though each film** (i.e. loop through the array) in the JSON object

For each film, **check the title** for occurrences of the search term

If you find a match **send the (altered) title and the link** for that film back to your main script (see next slide for how the title is altered).

As your search continues it should **update your main script on its progress** so it can update the search count. I.e. you should send it the number of films you have already searched after each film is checked (as well as the total number of films it will be searching) .

You should also inform the main script **when the search is finished**.

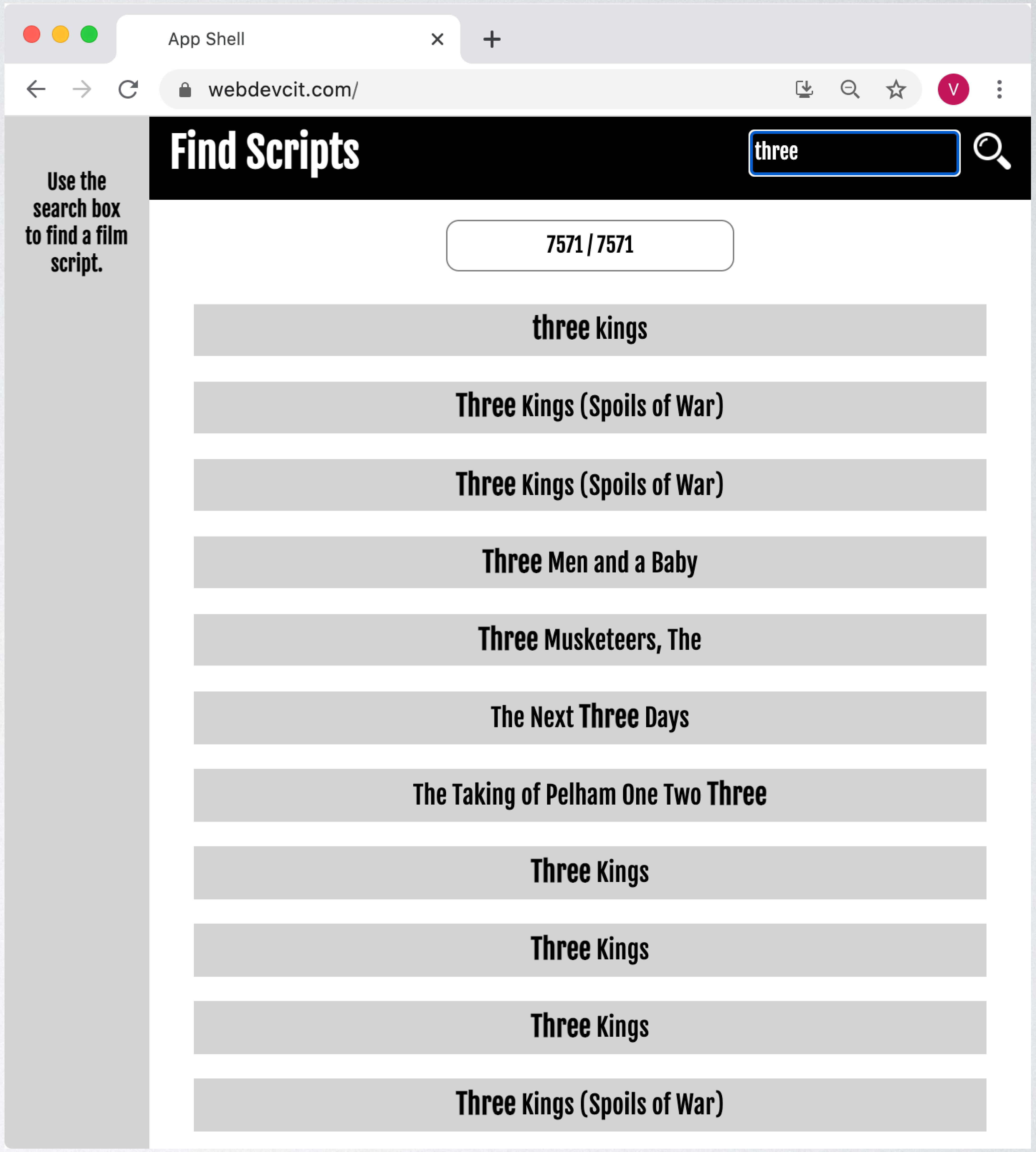
You should convert the title of any film that matches the search term to highlight the match with a <mark> tag **before you send it to the main script**.

E.g. Assuming a search for "Night" you would convert the following:

```
      night of the living dead
```

to:

```
     <mark>night</mark> of
the living dead
```



Use the search box to find a film script.

7571 / 7571

three kings

Three Kings (Spoils of War)

Three Kings (Spoils of War)

Three Men and a Baby

Three Musketeers, The

The Next Three Days

The Taking of Pelham One Two Three

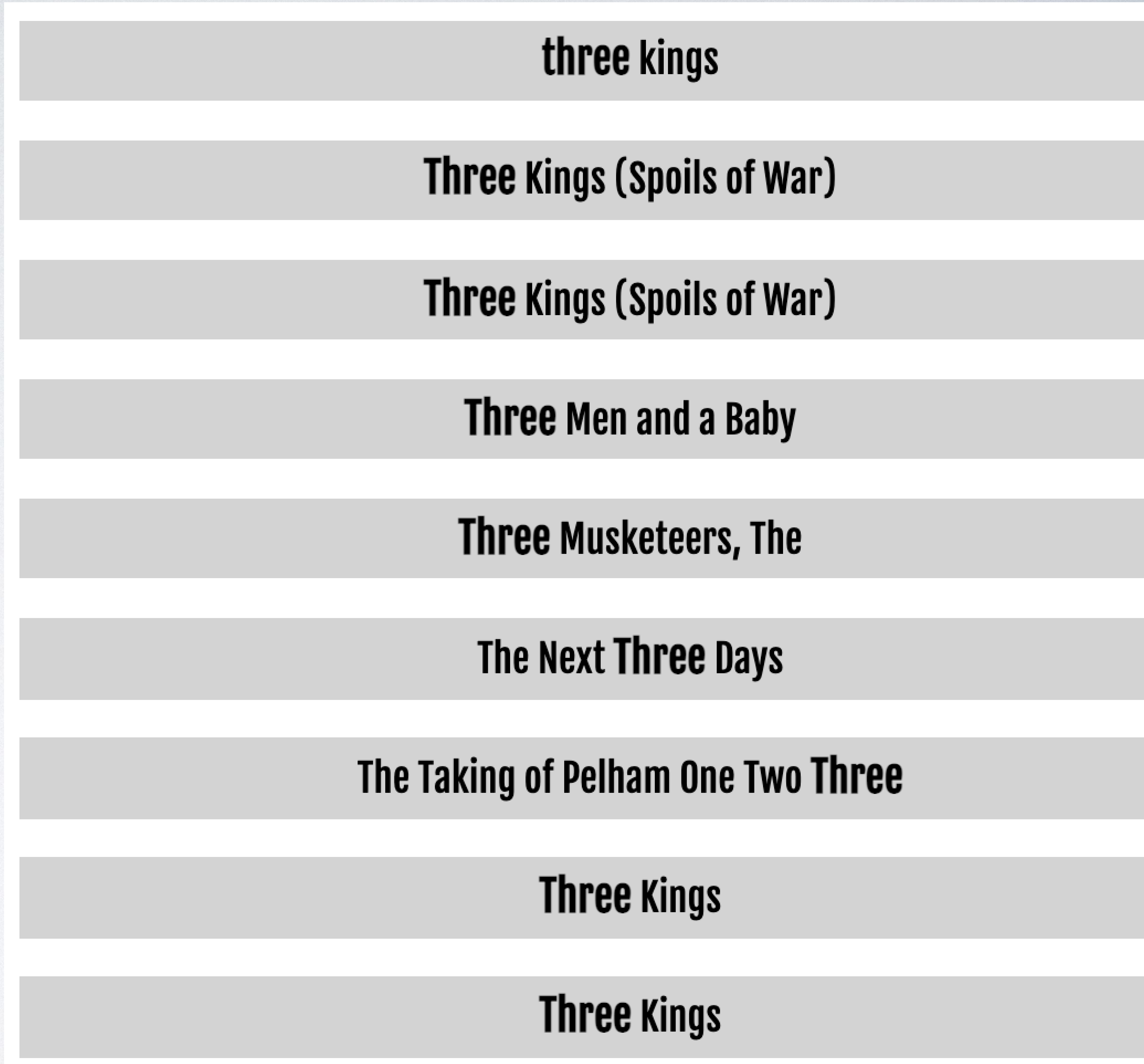
Three Kings

Three Kings

Three Kings

Three Kings (Spoils of War)

Note the word **three** is highlighted here in bold via CSS (applied to the <mar.> tags).



three kings

Three Kings (Spoils of War)

Three Kings (Spoils of War)

Three Men and a Baby

Three Musketeers, The

The Next **Three** Days

The Taking of Pelham One Two **Three**

Three Kings

Three Kings

Development Tips

The processing of over 7000 objects may take time when testing your code.

To help with your development you might find it useful to **arrange to only return a maximum of 100 results while testing** (so you don't have to wait for the entire search to finish everytime)

It may also help to speed up your work flow to **use a file with far fewer objects** when testing your code at the beginning.

Reduce Number of Messages

Communication between Web Workers and your main thread can slow your code.

If you are sending count values to the main thread after checking each film this will slow down your page. I.e. the communication is taking up a lot of time. Most of the updates can't even be seen it is replaced so quickly.

For the purposes of this assignment this is fine. But to speed up the functionality you can reduce the messages as shown in the notes.

JSON-P in Webworkers

Your webworker can use **importScripts()** to download the JSON-P file which will in turn call the **processFilms()** function when it finishes downloading.

```
onmessage = function(e){

   searchTerm = e.data;

   importScripts("movieObj.js");

}




function processFilms(data)
{

     < Perform Search>

}
```

Using regular expressions to wrap text around a given match.

```
var regex = new RegExp("word",'ig');

var text = "There is a word here";

newText = text.replace(regex , '<mark>$&</mark>');

console.log(newText);
```

Options string:
i: case insensitive
g: global (i.e. match every occurrence, not just the first)

text we are looking for.

```
var regex = new RegExp("word",'ig');
```

text we will be searching

```
var text = "There is a word here";

newText = text.replace(regex , '<mark>$&</mark>');

console.log(newText);
```

text we are searching in

We will make a copy of the text we are searching in with any changes made.

Regular expression specifying the match we are looking for.

```
newText = text.replace(regex , '<mark>$&</mark>');

console.log(newText);
```

This is what we will replace the matched text with.

We want to replace the matched text with the same text but with <mark> and </mark> tags around it.

$& will contain the matched text ("word" as per the regular expression)

```
newText = text.replace(regex , '<mark>$&</mark>');

console.log(newText);
```

```
var regex = new RegExp("word",'ig');

var text = "There is a word here";

newText = text.replace(regex , '<mark>$&</mark>');

console.log(newText);
```

$\longrightarrow$ There is a <mark>word</mark> here

Communication

Note that the communication from the WebWorker back to the main thread should be via JSON objects. This way we can send titles and descriptions with one message.