Client-side web APIs (Part 2)

Assoc. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Dept. of Computer Engineering
Khon Kaen University

What are third party APIs?

- Third party APIs are APIs provided by third parties — generally companies such as Facebook, Twitter, or Google — to allow you to access their functionality via JavaScript and use it on your own site
- Third party APIs have a slightly different permissions system — they tend to use key codes to allow developers access to the API functionality. Look again at the URL of the Google Maps API library we linked to

```
1 | https://maps.google.com/maps/api/js?key=AIzaSyDDuGt0E5IEGkcE6ZfrKfUtE9Ko_de66pA
```

- The URL parameter provided at the end of the URL is a developer key

Why using a developer key?

- The point of requiring a key is so that not just anybody can use API functionality without any kind of accountability
- When the developer has registered for a key, they are then known to the API provider
  - Action can be taken if they start to do anything malicious with the API (tracking people's location or trying to spam the API with loads of requests to stop it working, for example)
    - The easiest action would be to just revoke their API privileges

A RESTful API - NYTimes

- Let's look at an API example – the New York Times API
- This API allows you to retrieve New York Times news story information and display it on your site
- This type of API is known as a **RESTful API**
  - Getting data by making HTTP requests to specific URLs

An approach for using third-party APIs

1. Find the documentation
2. Get a developer key
3. Connect the API to your app
4. Request data from the API
5. Display the data

Find the documentation

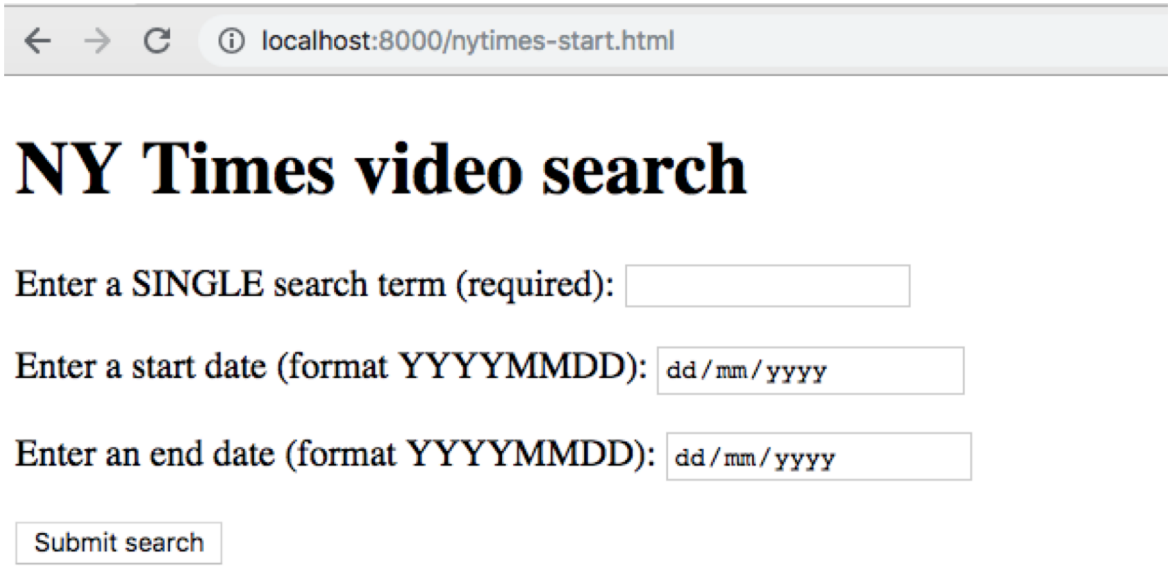
- When you want to use a third party API, it is essential to find out where the documentation is
- You can find out what features the API has, how you use them, etc.
- For example, the New York Times API documentation is at https://developer.nytimes.com/.

Get a developer key

- Most APIs require you to use some kind of developer key, for reasons of security and accountability
- To sign up for an NYTimes API key, you need to go to https://developer.nytimes.com/signup
- Let's request a key for the "Article Search API"
  - Fill in the form
  - Select the API you want to use
  - Wait to get the key from your email

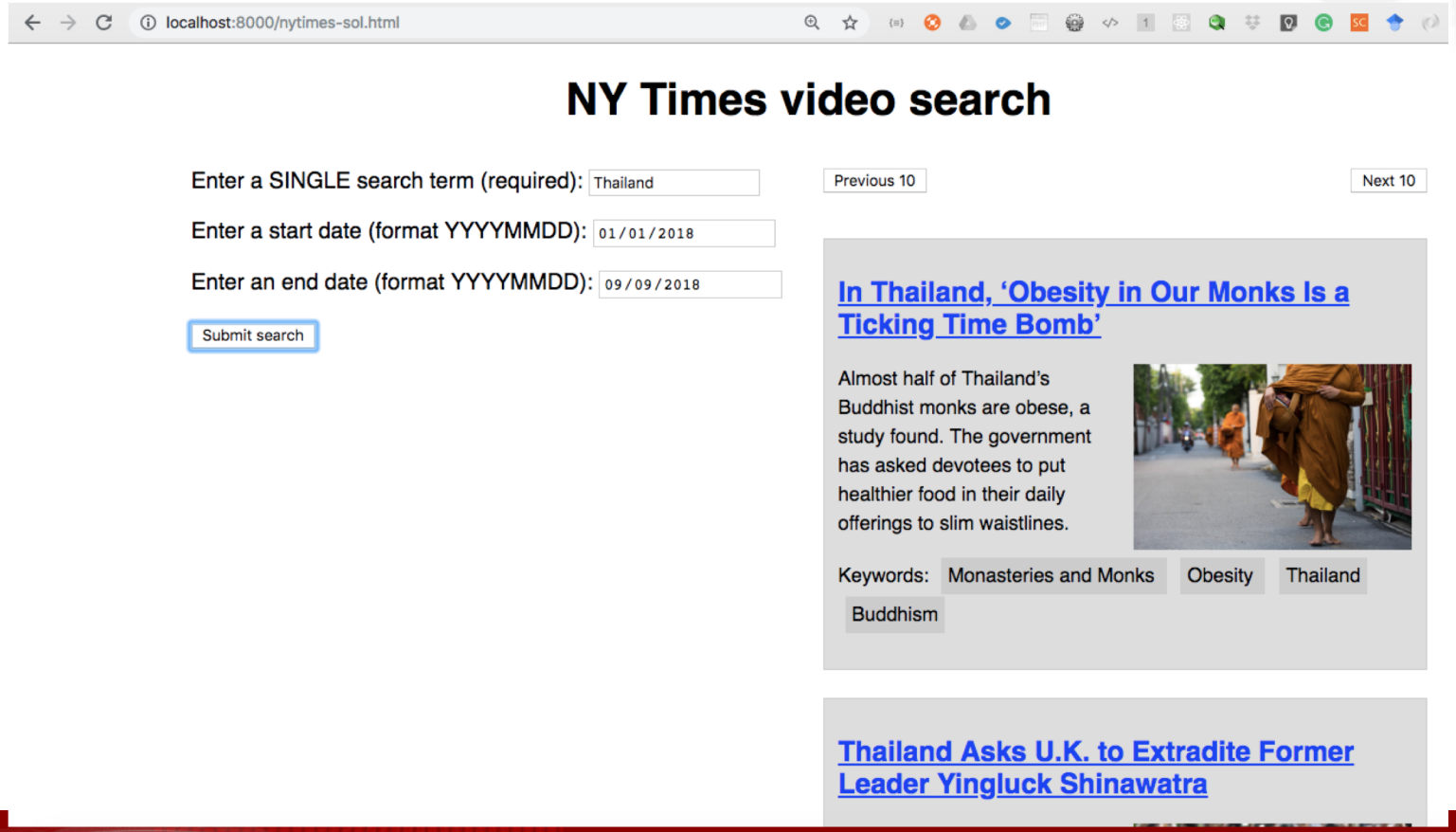
Start the example

- Download nytimes_start.html and nytimes.css given in Google classroom
- Then run the nytimes_start.html and you should get the page as shown below



The output of calling NYTimes

The app will end up allowing you to type in a search term and optional start and end dates, which it will then use to query the Article Search API and display the search results.



Calling NYTimes Tutorial

1. Replace INSERT-YOUR-API-KEY-HERE with the actual API key you got in the previous section

```
var key = 'INSERT-YOUR-API-KEY-HERE';
```

2. Add the following line to your JavaScript, below the "//Event listeners to control the functionality" comment

- This runs a function called fetchResults() when the form is submitted (the button is pressed)

```
searchForm.addEventListener('submit', submitSearch);
```

Calling NYTimes Tutorial

- Now add the submitSearch() and fetchResults() function definitions, below the previous line

```
function submitSearch(e) {
  pageNumber = 0;
  fetchResults(e);
}

function fetchResults(e) {
  // Use preventDefault() to stop the form submitting
  e.preventDefault();

  // Assemble the full URL
  url = baseURL + '?api-key=' + key + '&page=' + pageNumber

  if(startDate.value !== '') {
    url += '&begin_date=' + startDate.value;
  };

  if(endDate.value !== '') {
    url += '&end_date=' + endDate.value;
  };

}
```

Method submitSearch()

- submitSearch() sets the page number back to 0 to begin with, then calls fetchResults()
- This first calls preventDefault() on the event object, to stop the form actually submitting (which would break the example)
- Next, we use some string manipulation to assemble the full URL that we will make the request to.

Method fetchResults()

- The base URL (taken from the baseURL variable).
- The API key, which has to be specified in the api-key URL parameter (the value is taken from the key variable).
- The page number, which has to be specified in the page URL parameter (the value is taken from the pageNumber variable).
- The search term, which has to be specified in the q URL parameter (the value is taken from the value of the searchTerm text <input>).
- The document type to return results for, as specified in an expression passed in via the fq URL parameter. In this case, we just want to return articles.
- Example:

https://api.nytimes.com/svc/search/v2/articlesearch.json?api-key=4f3c267e125943d79b0a3e679f608a78&page=0&q=cats&fq=document_type:("article")&begin_date=20170301&end_date=20170312

Requesting data from the API

- Now we've constructed our URL, let's make a quest to it
- We'll do this using the Fetch API
- Add the following code block inside the fetchResults() function

```
1 | // Use fetch() to make the request to the API
2 | fetch(url).then(function(result) {
3 |   return result.json();
4 | }).then(function(json) {
5 |   displayResults(json);
6 | });
```

Displaying the data

- The while loop is a common pattern used to delete all of the contents of a DOM element, in this case, the <section> element
- We keep checking to see if the <section> has a first child, and if it does, we remove the first child. The loop ends when <section> no longer has any children.

```
function displayResults(json) {
  while (section.firstChild) {
      section.removeChild(section.firstChild);
  }
```

Displaying the data (1/6)

- Next, we set the articles variable to equal json.response.docs
  - This is the array holding all the objects that represent the articles returned by the search

```
var articles = json.response.docs;
```

Displaying the data (2/6)

- The first if() block checks to see if 10 articles are returned (the API returns up to 10 articles at a time.)
- If so, we display the <nav> that contains the *Previous 10*/*Next 10* pagination buttons
- If less than 10 articles are returned, they will all fit on one page, so we don't need to show the pagination buttons
- We will wire up the pagination functionality in the next section.

```
if(articles.length === 10) {
  nav.style.display = 'block';
} else {
  nav.style.display = 'none';
}
```

Displaying the data (3/6)

- The next if() block checks to see if no articles are returned
- If so, we don't try to display any — we just create a <p> containing the text "No results returned." and insert it into the <section>

```
if(articles.length === 0) {
  var para = document.createElement('p');
  para.textContent = 'No results returned.'
  section.appendChild(para);
} else {
```

Displaying the data (4/6)

- If some articles are returned, we, first of all, create all the elements that we want to use to display each news story, insert the right contents into each one, and then insert them into the DOM at the appropriate places

```
for(var i = 0; i < articles.length; i++) {
  var article = document.createElement('article');
  var heading = document.createElement('h2');
  var link = document.createElement('a');
  var img = document.createElement('img');
  var para1 = document.createElement('p');
  var para2 = document.createElement('p');
  var clearfix = document.createElement('div');
```

```
    article.appendChild(heading);
    heading.appendChild(link);
    article.appendChild(img);
    article.appendChild(para1);
    article.appendChild(para2);
    article.appendChild(clearfix);
    section.appendChild(article);
  }
 }
};
```

Displaying the data (5/6)

We used a for loop

(for(var j = 0; j < current.keywords.length; j++) { ... })

to loop through all the keywords associated with each article, and insert each one inside its own , inside a <p>. This was done to make it easy to style each one.

```
var current = articles[i];
console.log(current);

link.href = current.web_url;
link.textContent = current.headline.main;
para1.textContent = current.snippet;
para2.textContent = 'Keywords: ';
for(var j = 0; j < current.keywords.length; j++) {
  var span = document.createElement('span');
  span.textContent += current.keywords[j].value + ' ';
  para2.appendChild(span);
}
```

Displaying the data (6/6)

- We used an if() block (if(current.multimedia.length > 0) { ... }) to check whether each article actually has any images associated with it (some stories don't.) We display the first image only if it actually exists (otherwise an error would be thrown)
- We gave our element a class of "clearfix", so we can easily apply clearing to it (this technique is needed at the time of writing to stop floated layouts from breaking.)

```
if(current.multimedia.length > 0) {
  img.src = 'http://www.nytimes.com/' + current.multimedia[0].url;
  img.alt = current.headline.main;
}

clearfix.setAttribute('class','clearfix');
```

Wiring up the pagination buttons (1/2)

- To make the pagination buttons work, we will increment (or decrement) the value of the pageNumber variable, and then re-rerun the fetch request with the new value included in the page URL parameter
- This works because the NYTimes API only returns 10 results at a time
  - If more than 10 results are available, it will return the first 10 (0-9)
  - If the page URL parameter is set to 0 (or not included at all — 0 is the default value), the next 10 (10-19)
  - If it is set to 1, and so on

Wiring up the pagination buttons (2/2)

- Below your previous addition, let's define the two functions — add this code now

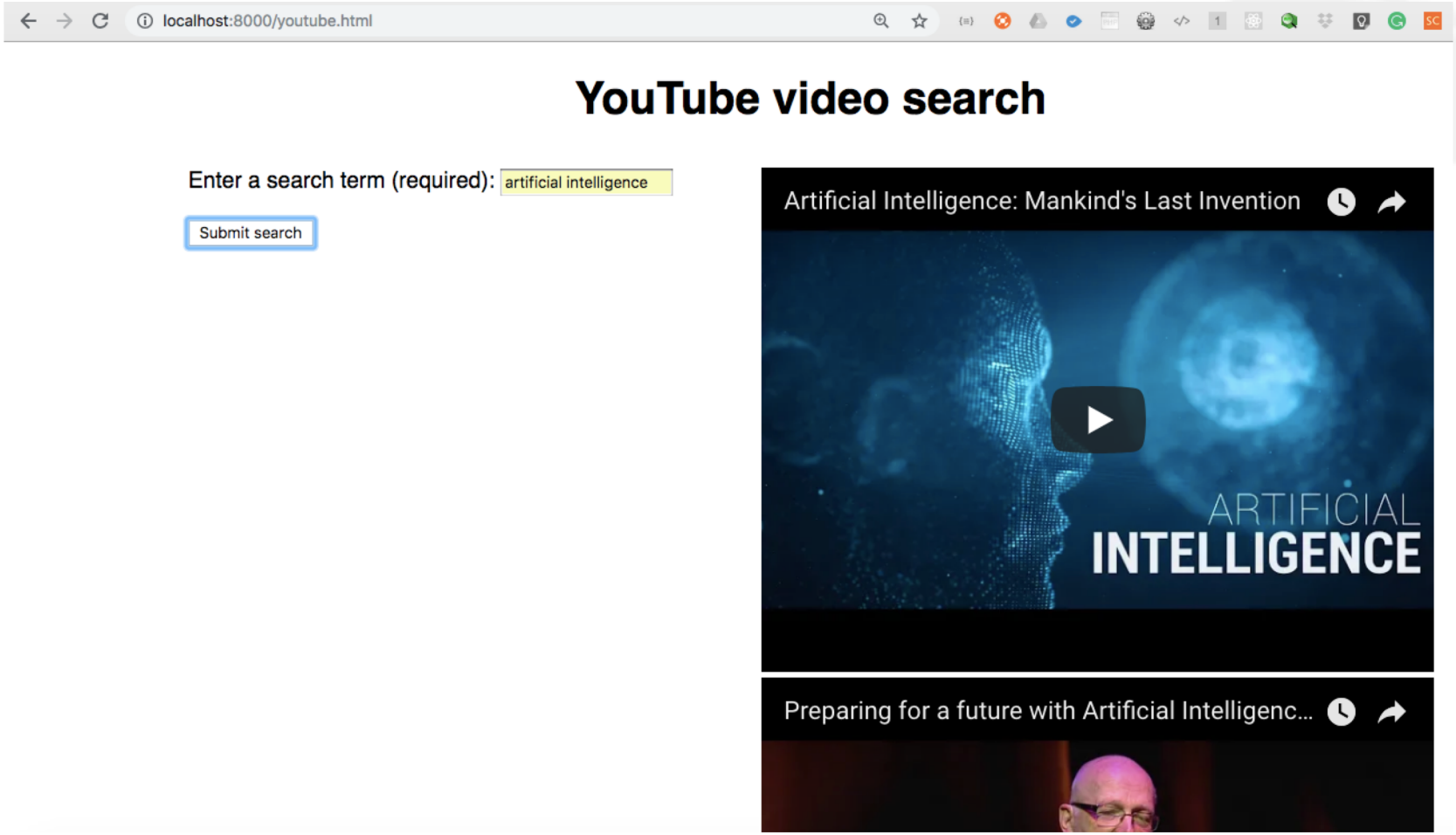
```
function nextPage(e) {
  pageNumber++;
  fetchResults(e);
};

function previousPage(e) {
  if(pageNumber > 0) {
    pageNumber--;
  } else {
    return;
  }
  fetchResults(e);
};
```

YouTube example

- We also built another example for you to study and learn from — see our YouTube video search example
- This uses two related APIs:
  - The YouTube Data API to search for YouTube videos and return results.
  - The YouTube IFrame Player API to display the returned video examples inside IFrame video players so you can watch them.

The output of YouTube search



YouTube Search (1/5)

```
 8  <body>
 9    <h1>YouTube video search</h1>
10
11    <div class="wrapper">
12
13      <div class="controls">
14        <form>
15          <p>
16            <label for="search">Enter a search term (required): </label>
17            <input type="text" id="search" class="search" required>
18          </p>
19          <p>
20            <button class="submit">Submit search</button>
21          </p>
22        </form>
23      </div>
24
25      <div class="results">
26        <section>
27        </section>
28      </div>
29
30    </div>
```

YouTube Search (2/5)

```
32    <!--
33      Apply both necessary API scripts to your HTML. The first is for the YouTube
        Data
34      API, and the second is for the YouTube Iframe Player API
35    -->
36    <script src="https://apis.google.com/js/client.js" type="text/javascript">
      </script>
37    <script src="https://www.youtube.com/iframe_api" type="text/javascript">
      </script>
38
39    <script>
40      // Get references to DOM elements we need to manipulate
41      var searchTerm = document.querySelector('.search');
42      var searchForm = document.querySelector('form');
43      var submitBtn = document.querySelector('.submit');
44      var section = document.querySelector('section');
45      // When the window (tab) has finished loading, run onClientLoad() to get
46      // It all started
47      window.onload = onClientLoad;
48      // Load and inialize the API, then run the onYouTubeApiLoad() function once
        this is done
49      function onClientLoad() {
50        gapi.client.load('youtube', 'v3', onYouTubeApiLoad);
51      }
```

YouTube Search (3/5)

```
53      function onYouTubeApiLoad() {
54        // To get a key for your own application:
55        // 1. Go to https://console.cloud.google.com/apis/dashboard
56        // 2. Create a new project if you've not already got one
57        // 3. Click the Enable API button
58        // 4. Choose YouTube Data API
59        // 5. Click the Enable button
60        // 6. Click create credentials
61        // 7. Select "Web browser (JavaScript)" from the second dropdown
62        // 8. Click the "Public data" radio button
63        // 9. Click the "What credentials do I need?" button
64        // 10. Copy your API key and paste it in below
65        gapi.client.setApiKey('AIzaSyB6B_1tjwrSuM_NYS2HbqINbaapkEi2ThA');
66        // Attach an event listener to the form so that a search is carried out
67        // when it is submitted – the search() function
68        searchForm.addEventListener('submit', search);
69      }
70      function search(e) {
71        // use preventDefault() to stop form actually submitting
72        e.preventDefault();
73        // create a search request using the Data API;
74        var request = gapi.client.youtube.search.list({
75          // set what kind of data the response will include
76          part: 'snippet',
77          // set the number of results that will be returned
78          maxResults: 10,
79          // set the search query to search for
80          q: searchTerm.value
81        });
82        // send the request, specify a function to run when the response returns
83        request.execute(onSearchResponse);
```

YouTube Search (4/5)

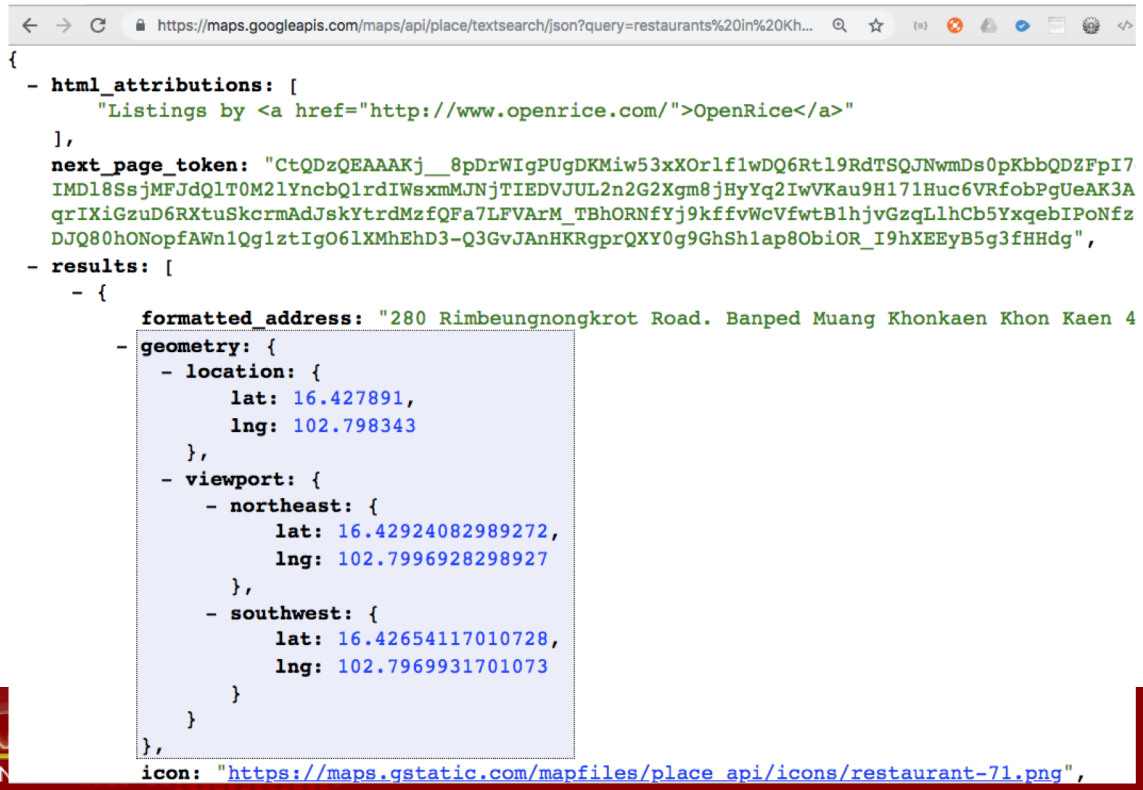
```
85        // This function automatically has the response passed in as a parameter
86        function onSearchResponse(response) {
87          // Empty the <section> element
88          while (section.firstChild) {
89              section.removeChild(section.firstChild);
90          }
91          // Store the actual results of the search in a variable
92          var results = response.items;
93          // loop through the results and run displayVideo() on each
94          for(var i = 0; i < results.length; i++) {
95            displayVideo(results[i], i);
96          }
97        }
98        function displayVideo(result, i) {
99          // Create a div with a unique ID for each video, and append it to the
            <section>
100         // The YouTube Iframe Player API will replace each one with
101         // an <iframe> containing the corresponding video
102         var vid = document.createElement('div');
103         vidId = 'vid' + i;
104         vid.id = vidId;
105         section.appendChild(vid);
```

YouTube Search (5/5)

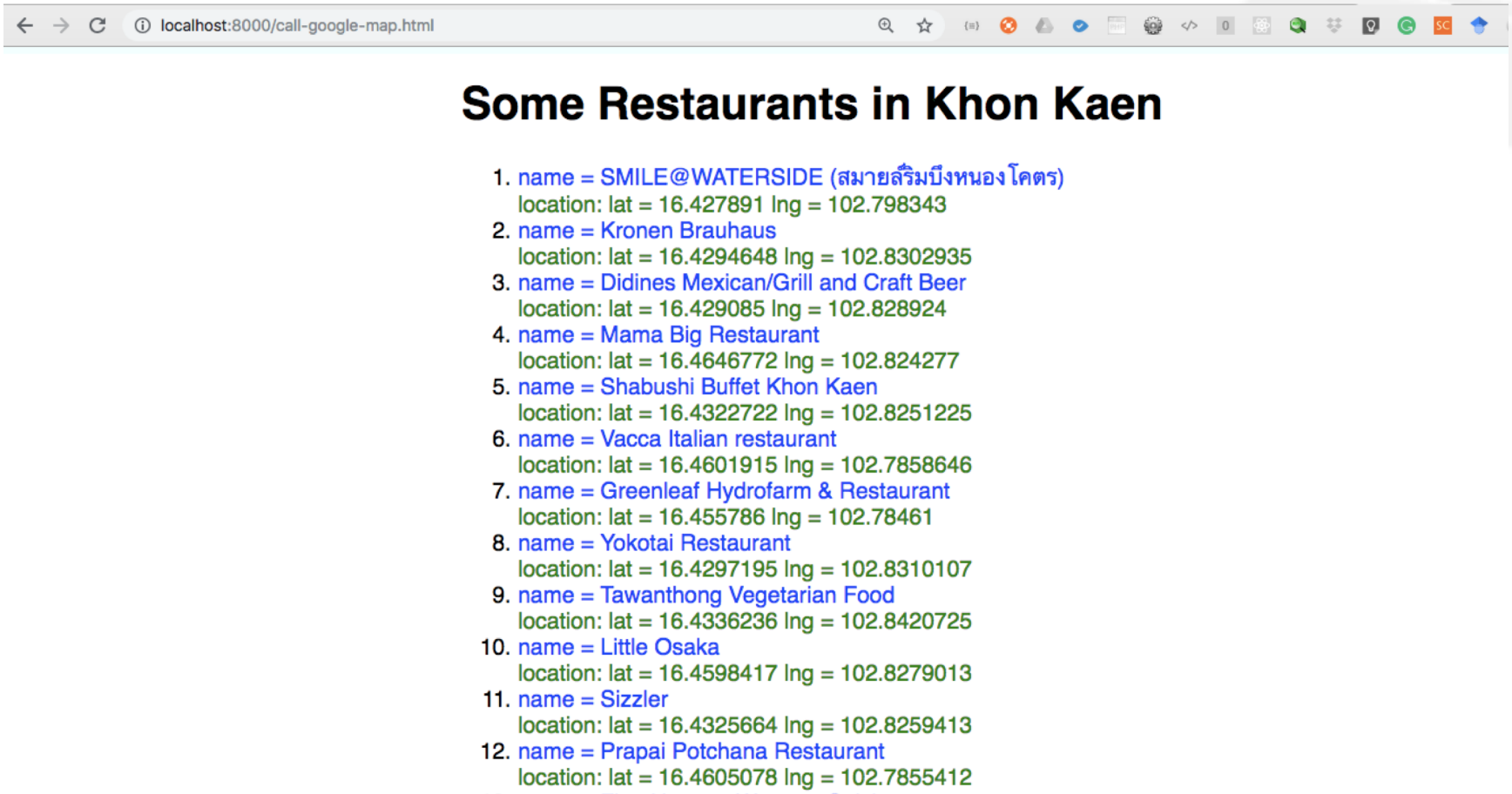
```
106         // Use the YT.Player() constructor to create a new video player object,
107         // Specifying the ID of the element to be replaced by it (the <div>),
108         // A height and width, and an event handler to handle the custom onReady
            event
109         var player = new YT.Player(vidId, {
110           height: '360',
111           width: '480',
112           videoId: result.id.videoId,
113           events: {
114             'onReady': onPlayerReady
115           }
116         });
117         // The onPlayerReady() handler grabs the ID of each video, and checks its
            duration
118         // If the duration is 0, the video can't be played, so we just delete it
119         function onPlayerReady(e) {
120           var myId = e.target.a.id;
121           var duration = e.target.getDuration();
122           if(duration === 0) {
123             console.log('Video ' + myId + ' cannot be played, so it was
              deleted.');
124             section.removeChild(e.target.a);
125           } else {
126             var myId = e.target.a.id;
127             console.log('Video ' + myId + ' ready to play.');
128           }
129         }
130       }
131     </script>
```

Exercise 1: Calling Google Map API

https://maps.googleapis.com/maps/api/place/textsearch/json?query=restaurants%20in%20Khon%20Kaen&key=AIzaSyC1IxT1nhPYaa8WIqS5N9UdqmW7nCZr42A



Exercise 1 Output



References

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
- Image source: Overloaded plug socket by The Clear Communication People, on Flickr.