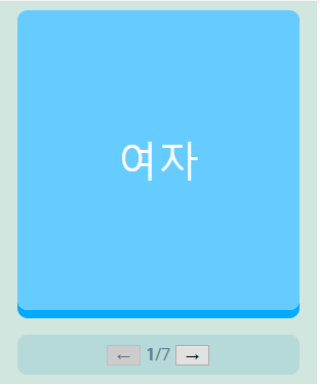
Schedule

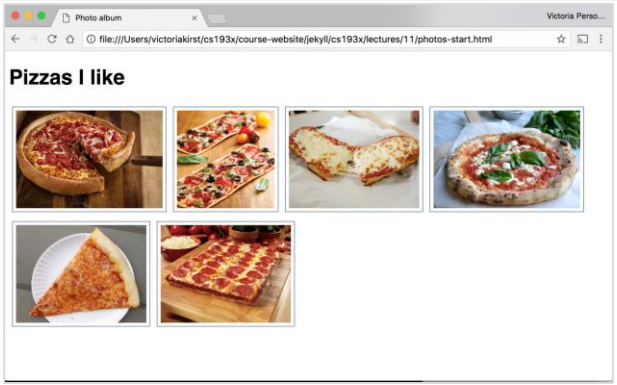
Today:

- API
- Loading data from files
  - Fetch API
  - Promises - High-level!
  - JSON
- CORS
- Fetch API JSON
- Selected topic:
  - Form submit

Back to our client applications



Lecture: Presents, Photo album
Tutorials: Playing Cards, Flash cards
Assignment: Dog type – Personality



WHERE IS THE DATA FROM?

Card boards

Enjoy your presents!

You got: Excellent judge of character

Your derpy exterior belies the keen bloodhound within, an
nose, both for sniffing out people's desires and for finding
when you're confused and tilt your head when things are f
your favorite characters are the ones who have complex st
winding subplots. However, you like short sentences. You l
start a garden. Unfortunately, plants are not as easy to und
the last time you took care of a flower, it didn't turn out so
very good care of your friends, and they are lucky to have you.

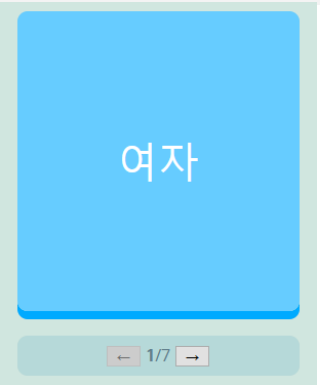
You got: A positive force in the world

You are optimistic and are confident in your ability to change something for the better, whether that is a group, a process, or yourself. It is difficult to label you as an introvert or an extrovert. You have many books that you have been meaning to read, and next week you will start one. When you wake up in the morning, It is hard for you to remember what you dreamed about. You like the sound of rain but do not enjoy getting wet. You are more fearless than the average person, especially in terms of sharing who you are. At the same time, you tend to be very aware of how your actions affect others. You fear that you are not doing enough, but to everyone else, you are!

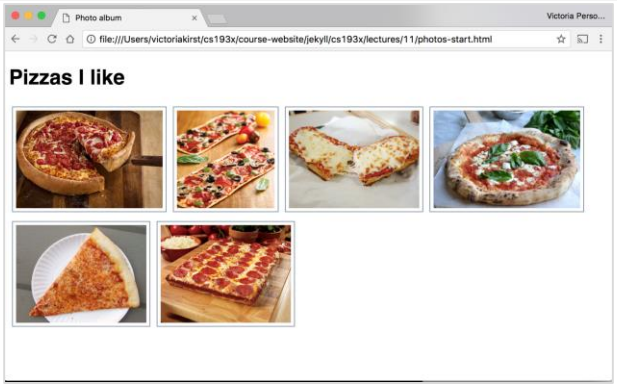
Open-minded and big-hearted

Harry Potter Pup. The pup chosen to right the wrongs in the world and
y back to Middle Earth. You enjoy free-range chicken, reading
graphies and retaking the Myers Briggs Personality test to reaffirm that
lity type is as rare as a snowflake landing on a jaguar's backside. You
owers, but I would highly recommend not putting that on your resume,
le might not agree and (2) you are a dog and seeking traditional
employment might already be difficult.

Back to our client applications



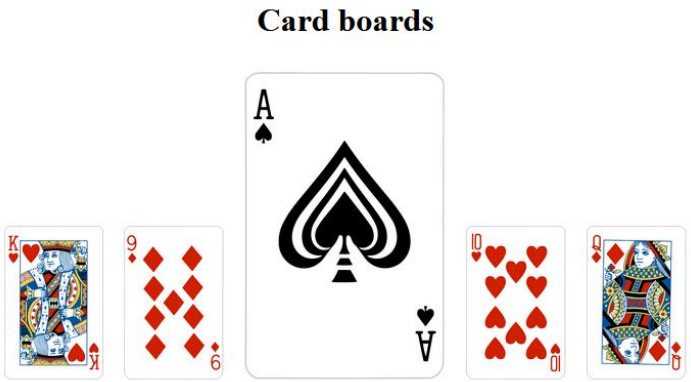
Lecture: Presents, Photo album
Tutorials: Playing Cards, Flash cards
Assignment: Dog type – Personality



WHERE IS THE DATA FROM?

Files

photo-list.js,
present-sources.js,
data.js, …



You got: Excellent judge of character

Your derpy exterior belies the keen bloodhound within, an
nose, both for sniffing out people's desires and for finding
when you're confused and tilt your head when things are f
your favorite characters are the ones who have complex st
winding subplots. However, you like short sentences. You
start a garden. Unfortunately, plants are not as easy to und
the last time you took care of a flower, it didn't turn out so
very good care of your friends, and they are lucky to have you.

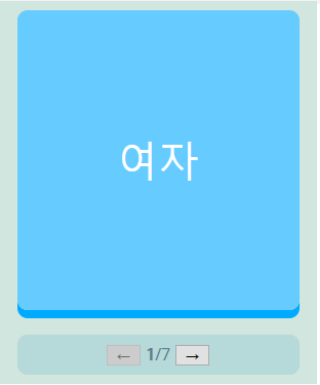
You got: A positive force in the world

You are optimistic and are confident in your ability to change something for the better, whether that is a group, a process, or yourself. It is difficult to label you as an introvert or an extrovert. You have many books that you have been meaning to read, and next week you will start one. When you wake up in the morning, It is hard for you to remember what you dreamed about. You like the sound of rain but do not enjoy getting wet. You are more fearless than the average person, especially in terms of sharing who you are. At the same time, you tend to be very aware of how your actions affect others. You fear that you are not doing enough, but to everyone else, you are!

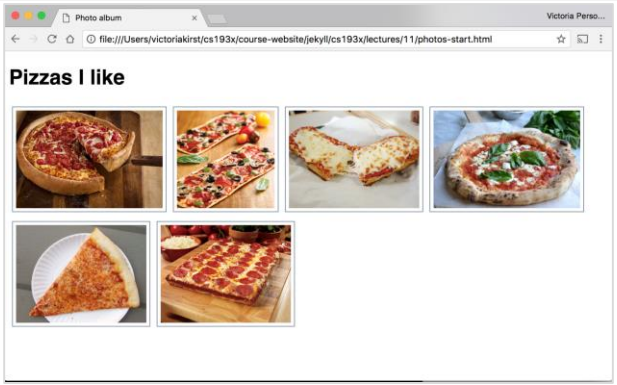
Open-minded and big-hearted

Harry Potter Pup. The pup chosen to right the wrongs in the world and
y back to Middle Earth. You enjoy free-range chicken, reading
graphies and retaking the Myers Briggs Personality test to reaffirm that
lity type is as rare as a snowflake landing on a jaguar's backside. You
owers, but I would highly recommend not putting that on your resume,
le might not agree and (2) you are a dog and seeking traditional
employment might already be difficult.

Back to our client applications

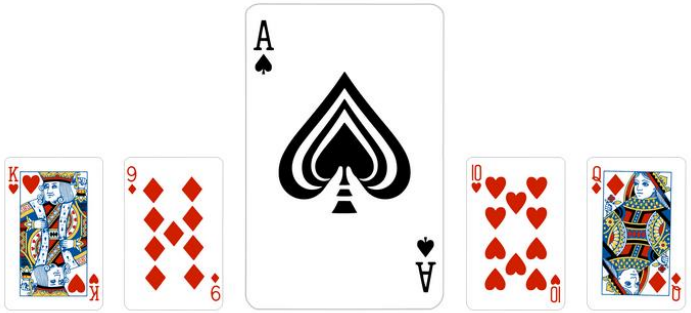


Lecture: Presents, Photo album
Tutorials: Playing Cards, Flash cards
Assignment: Dog type – Personality



WHERE IS THE DATA FROM?
Files
FIXED

Card boards



Enjoy your presents!

You got: Excellent judge of character

Your derpy exterior belies the keen bloodhound within, an
nose, both for sniffing out people's desires and for finding
when you're confused and tilt your head when things are f
your favorite characters are the ones who have complex st
winding subplots. However, you like short sentences. You
start a garden. Unfortunately, plants are not as easy to und
the last time you took care of a flower, it didn't turn out so
very good care of your friends, and they are lucky to have you.

You got: A positive force in the world

You are optimistic and are confident in your ability to change something for the better, whether that is a group, a process, or yourself. It is difficult to label you as an introvert or an extrovert. You have many books that you have been meaning to read, and next week you will start one. When you wake up in the morning, It is hard for you to remember what you dreamed about. You like the sound of rain but do not enjoy getting wet. You are more fearless than the average person, especially in terms of sharing who you are. At the same time, you tend to be very aware of how your actions affect others. You fear that you are not doing enough, but to everyone else, you are!

Open-minded and big-hearted

Harry Potter Pup. The pup chosen to right the wrongs in the world and
y back to Middle Earth. You enjoy free-range chicken, reading
graphies and retaking the Myers Briggs Personality test to reaffirm that
lity type is as rare as a snowflake landing on a jaguar's backside. You
owers, but I would highly recommend not putting that on your resume,
le might not agree and (2) you are a dog and seeking traditional
employment might already be difficult.

Back to our client applications

BUT PRACTICALLY,
WHERE IS THE DATA FROM?

Back to our client applications

BUT PRACTICALLY,
WHERE IS THE DATA FROM?

Database, [3rd party] API

Back to our client applications

BUT PRACTICALLY,
WHERE IS THE DATA FROM?

Database*, [3rd party] API+

DYNAMIC

* Database = Files + Database Management System (DBMS)
+ Facebook API, Google API, Github API…

Loading data from files

Loading data from a file

What if you had a list of images in a text file that you wanted to load in your web page?

```
https://media1.giphy.com/media/xNT2CcLjhbI0U/200.gif
https://media2.giphy.com/media/3o7btM3VVVNtssGReo/200.gif
https://media1.giphy.com/media/l3q2uxEzLIE8cWMq4/200.gif
https://media2.giphy.com/media/LDwL3ao61wfHa/200.gif
https://media1.giphy.com/media/3o7TKMt1VVNkHV2PaE/200.gif
https://media3.giphy.com/media/DNQFjMJbbsNmU/200.gif
https://media1.giphy.com/media/26FKTsKMKtUSomuNq/200.gif
https://media1.giphy.com/media/xThuW5Hf2N8idJHFVS/200.gif
https://media1.giphy.com/media/XlFfSD0CiyGLC/200.gif
https://media3.giphy.com/media/ZaBHSbiLQTmFi/200.gif
https://media3.giphy.com/media/JPbZwjMcxJYic/200.gif
https://media1.giphy.com/media/FArgGzk7KO14k/200.gif
https://media1.giphy.com/media/UFoLN1EyKjLbi/200.gif
https://media1.giphy.com/media/11zXBCAb9soCQM/200.gif
https://media4.giphy.com/media/xUPGcHeIeZMmTcDQJy/200.gif
https://media2.giphy.com/media/apZwWJInOBvos/200.gif
https://media2.giphy.com/media/sB4nvt5xIiNig/200.gif
https://media0.giphy.com/media/Y8Bi9lCOzXRkY/200.gif
https://media1.giphy.com/media/12wUXjm6f8Hhcc/200.gif
https://media4.giphy.com/media/26gsuVyk5fKB1YAAE/200.gif
https://media3.giphy.com/media/l2SpMU9sWIvT2nrCo/200.gif
https://media2.giphy.com/media/kR1vWazNc7972/200.gif
https://media4.giphy.com/media/Tv3m2GAAl2Re8/200.gif
https://media2.giphy.com/media/9nujydsBLz2dq/200.gif
https://media3.giphy.com/media/AG39l0rHgkRLa/200.gif
```

Intuition: `loadFromFile`

If we wanted to have an API to load **external** files in JavaScript, it might look something like this:

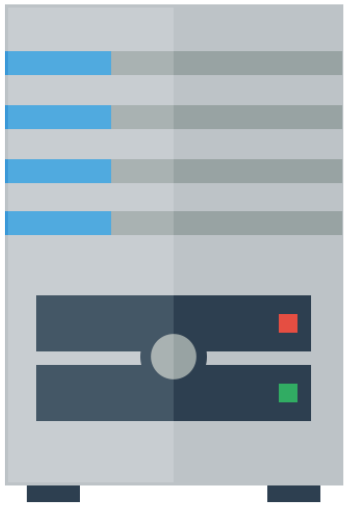
```
// FAKE HYPOTHETICAL API.
// This is not real a JavaScript function!
const contents = loadFromFile('images.txt');
```

First: Servers again

Servers

Sometimes when you type a URL in your browser, the URL is a **path to a file** on the internet:

- Your browser connects to the host address and requests the given file over **HTTP**
- The web server software (e.g. Apache) grabs that file from the server's local file system, and sends back its contents to you



HTTP: Hypertext Transfer Protocol, the protocol for sending files and messages through the web

HTTP methods

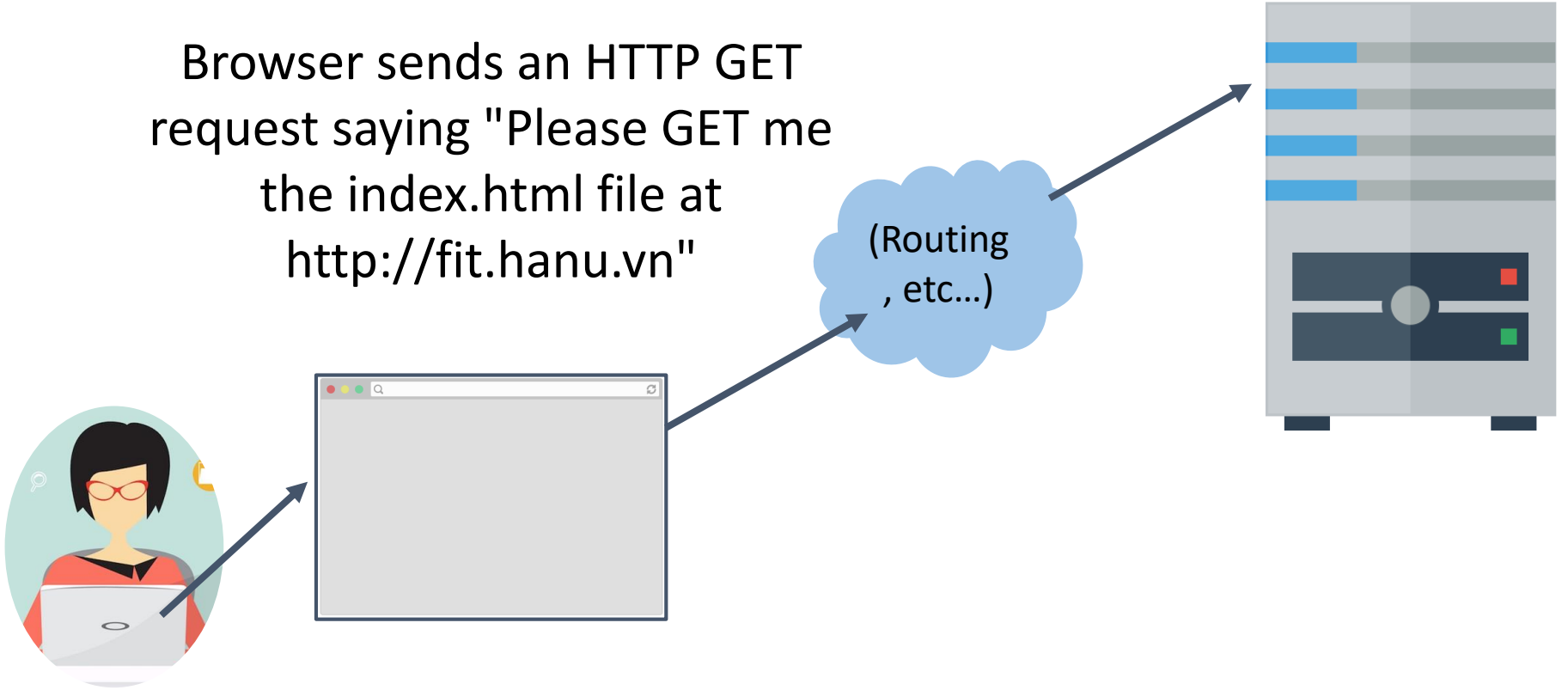
HTTP Methods: the set of commands understood by a web server and sent from a browser

- **GET**: request/retrieve data
  This is request sent by the browser automatically whenever you navigate to a URL!
- **POST**: send/submit data
- **PUT**: upload file
- **PATCH**: updates data
- **DELETE**: delete data
- More HTTP methods

You type a URL in
the address bar and
hit "enter"

Server at
http://fit.hanu.vn

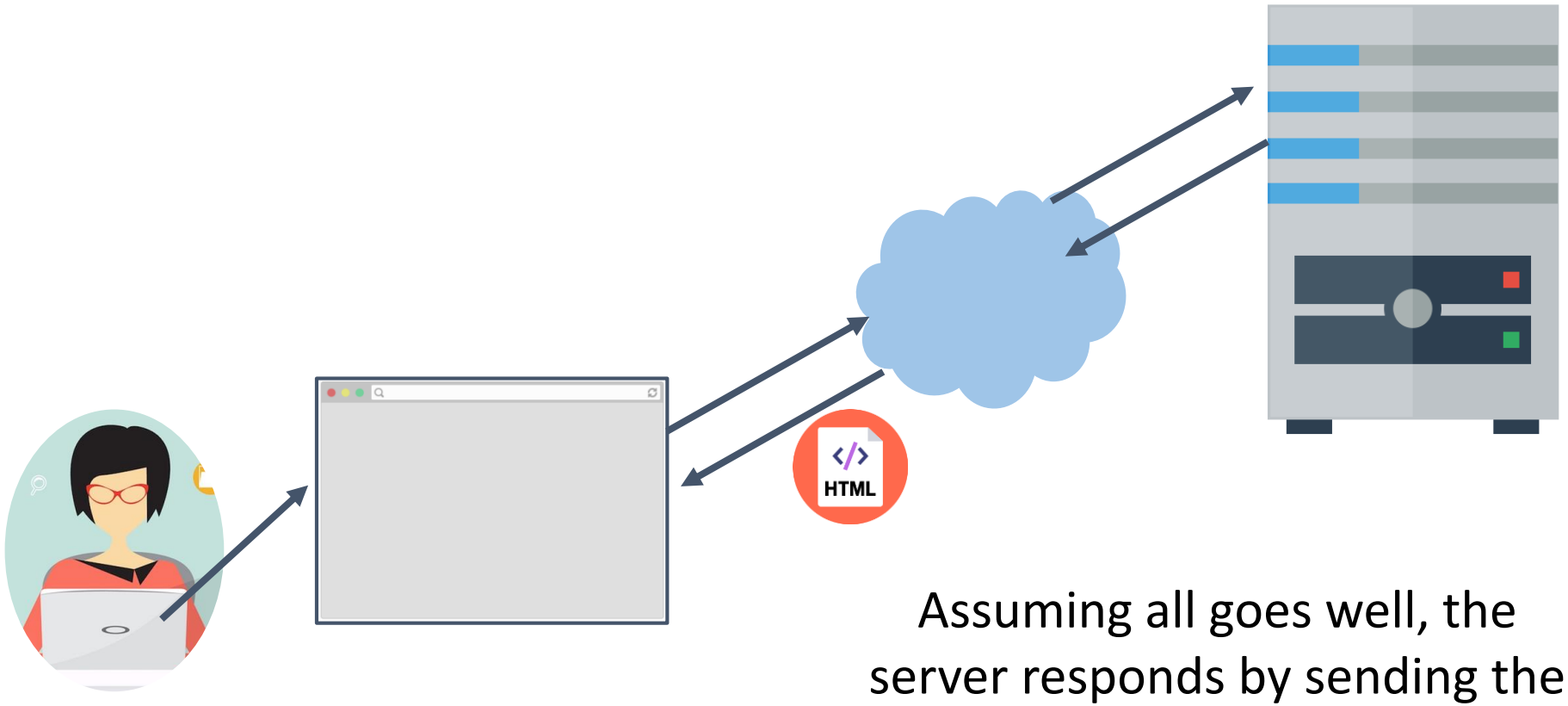
Browser sends an HTTP GET request saying "Please GET me the index.html file at http://fit.hanu.vn"



(**Warning:** Somewhat inaccurate, massive hand-waving begins now.

See this Quora answer for slightly more detailed/accurate handwaving)

Server at
http://fit.hanu.vn

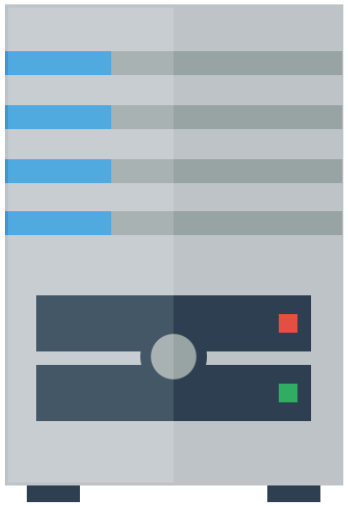


Assuming all goes well, the server responds by sending the HTML file through the internet back to the browser to display.

Servers

Sometimes when you type a URL in your browser, the URL is a **path to a file** on the internet:

- Your browser connects to the host address and requests the given file over **HTTP**
- The web server software (e.g. Apache) grabs that file from the server's local file system, and sends back its contents to you



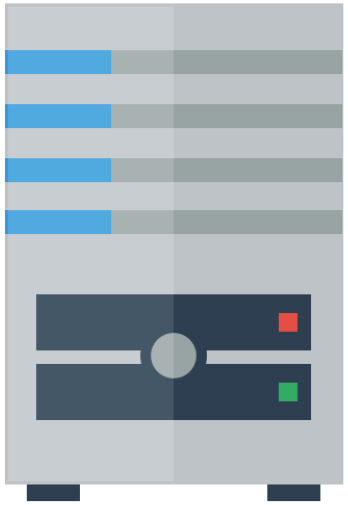
But that's not always the case.

Web Services

Other times when you type a URL into your browser, the URL represents **an API endpoint**, and not a path to a file.

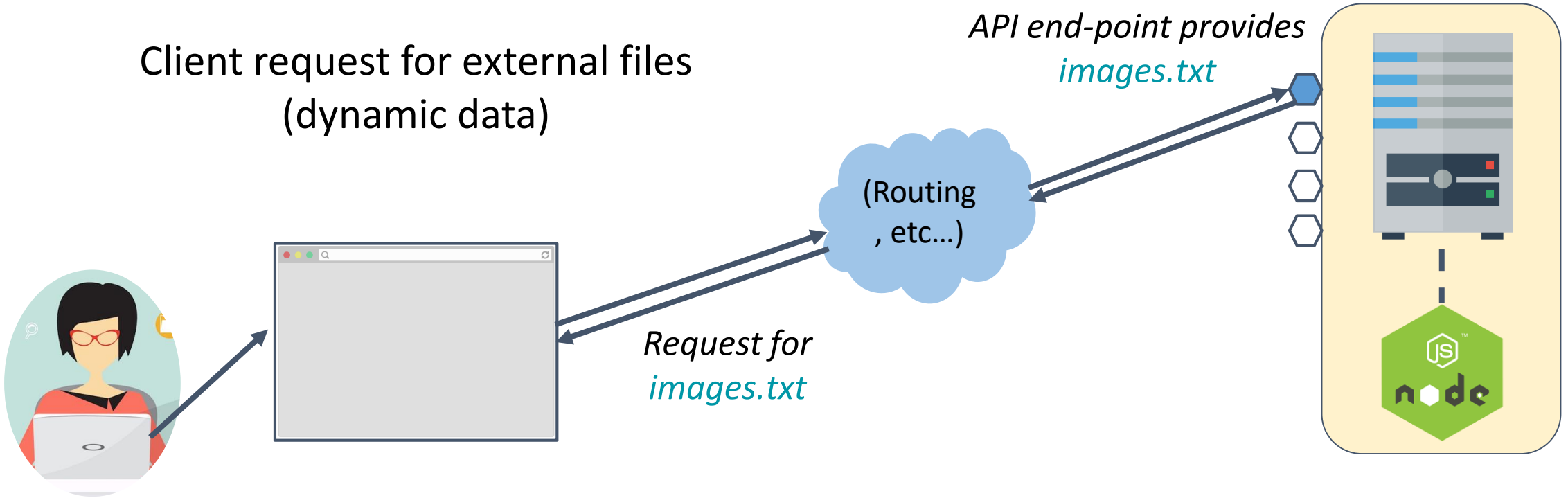
That is:

- The web server does **not** grab a file from the local file system, and the URL is **not** specifying where a file is located.
- Rather, the URL represents **a parameterized request**, and the web server dynamically generates a response to that request.



Design artifacts

Client request for external files
(dynamic data)



Server provides **API end-points** to serve “data”

API endpoint example

Look at the URL for this Google slide deck:

`https://docs.google.com/presentation/d/1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8`

API endpoint example

Look at the URL for this Google slide deck:

`https://docs.google.com/presentation/d/1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8`

- **`presentation`**: Tells the server that we are requesting a doc of type "presentation"
- **`d/1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8`**: Tells the server to request a doc ("d") with the document id of "1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8"

Example: Spotify

Spotify has a REST API that external developers (i.e. people who aren't Spotify employees) can query:

Our Web API endpoints give external applications access to Spotify catalog and user data.

Web API Base URL: `https://api.spotify.com` User Guide | Tutorial | Code Examples

Search:

METHOD	ENDPOINT	USAGE	RETURNS
GET	/v1/albums/{id}	Get an album	album
GET	/v1/albums?ids={ids}	Get several albums	albums
GET	/v1/albums/{id}/tracks	Get an album's tracks	tracks*
GET	/v1/artists/{id}	Get an artist	artist
GET	/v1/artists?ids={ids}	Get several artists	artists
GET	/v1/artists/{id}/albums	Get an artist's albums	albums*

RESTful API

RESTful API: URL-based API that has these properties:

- Requests are sent as an **HTTP request**:
  - HTTP Methods: GET, PUT, POST, DELETE, etc
- Requests are sent to **base URL**, also known as an "**API Endpoint**"
- Requests are sent with a specified **MIME/content type**, such as HTML, CSS, JavaScript, plaintext, JSON, etc.

Node for servers

server.js (GitHub):

```
const http = require('http');

const server = http.createServer();

server.on('request', function(req, res) {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.on('listening', function() {
  console.log('Server running!');
});

server.listen(3000);
```

Node for servers

Include the HTTP NodeJS library

```
const http = require('http');

const server = http.createServer();
```

When the server gets a request, send back "Hello World" in plain text

```
server.on('request', function(req, res) {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});
```

When the server is started, print a log message

```
server.on('listening', function() {
  console.log('Server running!');
});
```

Start listening for messages!

```
server.listen(3000);
```

Intuition: `loadFromFile`

```
// FAKE HYPOTHETICAL API.
// This is not real a JavaScript function!
const contents = loadFromFile('images.txt');
```

A few problems with this hypothetical fake API:

- We want to load the file **asynchronously**: the JavaScript should not block while we're loading the file
- There's no way to check the status of the request. What if the resource didn't exist? What if we're not allowed to access the resource?

Intuition: `loadFromFile`

An asynchronous version of this API might look like this:

```
// FAKE HYPOTHETICAL API.
// This is not real a JavaScript function!
function onSuccess(response) {
  const body = response.text;
  ...
}
loadFromFile('images.txt', onSuccess, onFail);
```

Where *`onSuccess`* and *`onFail`* are callback functions that should fire if the request succeeded or failed, respectively.

Fetch API

Fetch API: `fetch()`

The Fetch API is the API to use to load external resources (text, JSON, etc) in the browser.

The Fetch API is made up of one function, and its syntax is concise and easy to use:

```
fetch('images.txt');
```

Note: `XMLHttpRequest` ("XHR") is the old API for loading resources from the browser. XHR still works, but is clunky and harder to use.

Fetch API: `fetch()`

The Fetch API is the API to use to load external resources (text, JSON, etc) in the browser.

The Fetch API is made up of one function, and its syntax is concise and easy to use:

```
fetch('images.txt');
```

- The `fetch()` method takes the string path to the resource you want to fetch as a parameter
- It returns a `Promise`

Fetch API: `fetch()`

The Fetch API is the API to use to load external resources (text, JSON, etc) in the browser.

The Fetch API is made up of one function, and its syntax is concise and easy to use:

```
fetch('images.txt');
```

- The `fetch()` method takes the string path to the resource you want to fetch as a parameter
- It returns a `Promise`
  - **What the heck is a `Promise`?**

`Promise`s:
Another conceptual odyssey

`Promises` and `.then()`

A `Promise`:

- An object used to manage asynchronous results
- Has a *`then()`* method that lets you attach functions to execute *onSuccess* or *onError*
- Allows you to build **chains** of asynchronous results.

`Promises` are easier to **use** than to **define**...

Simple example: getUserMedia

There is an API called `getUserMedia` that allows you get the media stream from your webcam.

There are two versions of getUserMedia:

- `navigator.getUserMedia` **(deprecated)**
  - Uses callbacks

- `navigator.mediaDevices.getUserMedia`
  - Returns a `Promise`

getUserMedia with callbacks

```
const video = document.querySelector('video');

function onCameraOpen(stream) {
  video.srcObject = stream;
}

function onError(error) {
  console.log(error);
}

navigator.getUserMedia({ video: true },
  onCameraOpen, onError);
```

CodePen

getUserMedia with Promise

```
const video = document.querySelector('video');

function onCameraOpen(stream) {
  video.srcObject = stream;
}

function onError(error) {
  console.log(error);
}

navigator.mediaDevices.getUserMedia({ video: true })
  .then(onCameraOpen, onError);
```

CodePen

Hypothetical Fetch API

```
// FAKE HYPOTHETICAL API.
// This is not how fetch is called!
function onSuccess(response) {
  ...
}

function onFail(response) {
  ...
}

fetch('images.txt', onSuccess, onFail);
```

Real Fetch API

```
function onSuccess(response) {
  ...
}

function onFail(response) {
  ...
}

fetch('images.txt').then(onSuccess, onFail);
```

Promise syntax

Q: How does this syntax work?

```
fetch('images.txt').then(onSuccess, onFail);
```

Promise syntax

Q: How does this syntax work?

```
fetch('images.txt').then(onSuccess, onFail);
```

The syntax above is the same as:

```
const promise = fetch('images.txt');
promise.then(onSuccess, onFail);
```

Promise syntax

```
const promise = fetch('images.txt');
promise.then(onSuccess, onFail);
```

The object `fetch` returns is of type `Promise`.

A promise is in one of three states:

- **pending**: initial state, not fulfilled or rejected.
- **fulfilled**: the operation completed successfully.
- **rejected**: the operation failed.

You attach handlers to the promise via `.then()`

Promise syntax

```
const promise = fetch('images.txt');
promise.then(onSuccess, onFail);
```

The objec

A promis
- **pendi**
- **fulfill**
- **reject**

You attac

(Right now we will just use Promises.)

Using Fetch

```
function onSuccess(response) {
  console.log(response.status);
}

fetch('images.txt').then(onSuccess);
```

The success function for Fetch gets a response parameter:

- `response.status`: Contains the status code for the request, e.g. 200 for HTTP success
  - HTTP status codes

Fetch attempt

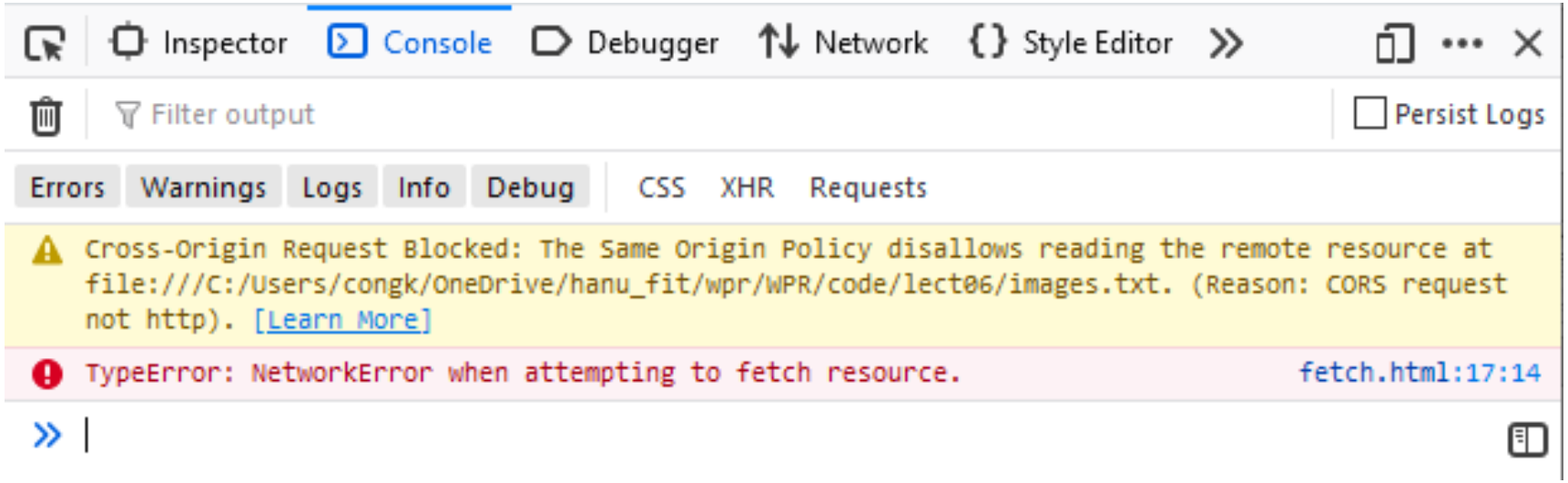
```
function onSuccess(response) {
  console.log(response.status);
}

function onError(error) {
  console.log('Error: ' + error);
}

fetch('images.txt')
    .then(onSuccess, onError);
```

Fetch error

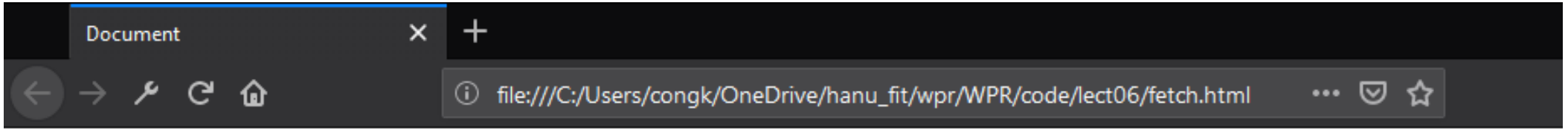
If we try to load this in the browser, we get the following JavaScript error:



Notice that our onError function was also called.

Local files

When we load a web page in the browser that is saved on our computer, it is served via `file://` protocol:



We are **not allowed** to load files in JavaScript from the `file://` protocol, which is why we got the error.

Serve over HTTP

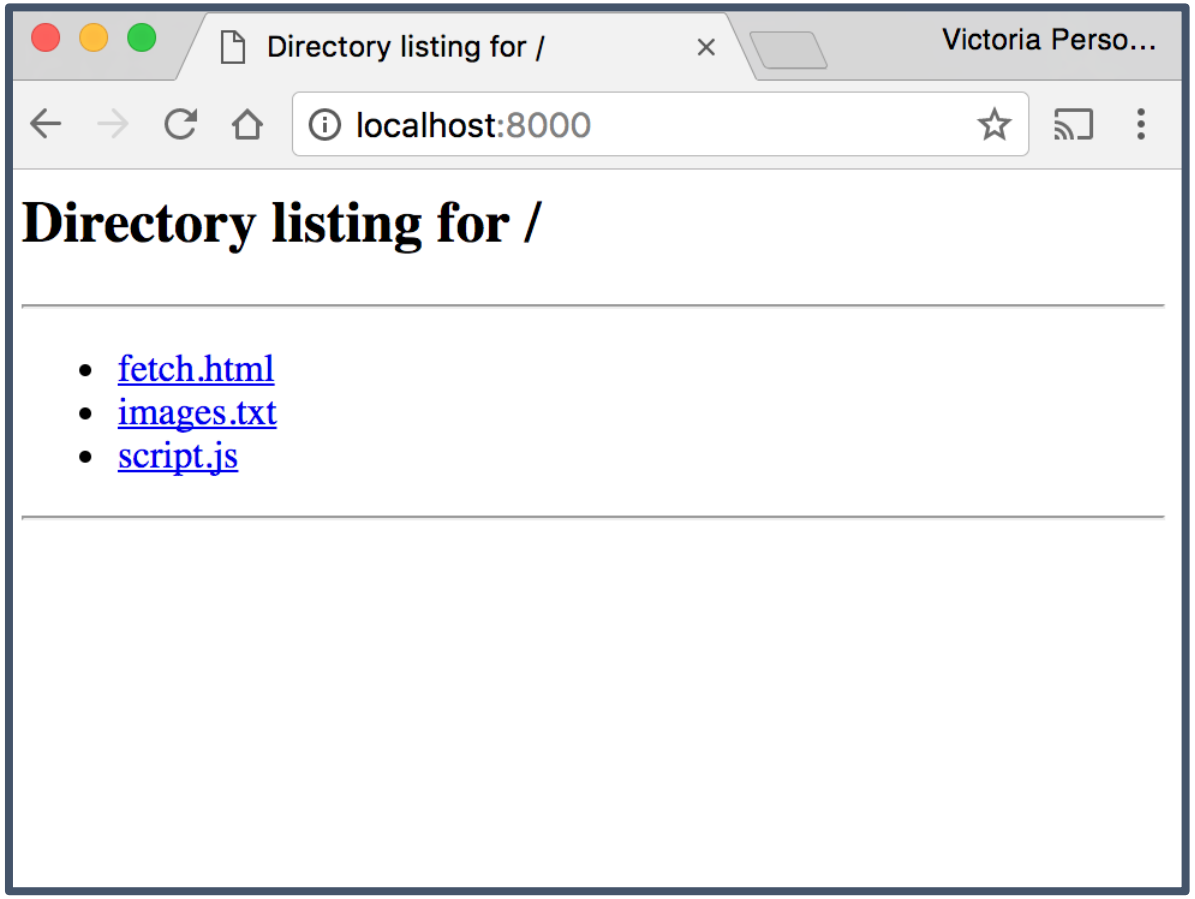
We can run a program to serve our local files over HTTP:
XAMPP

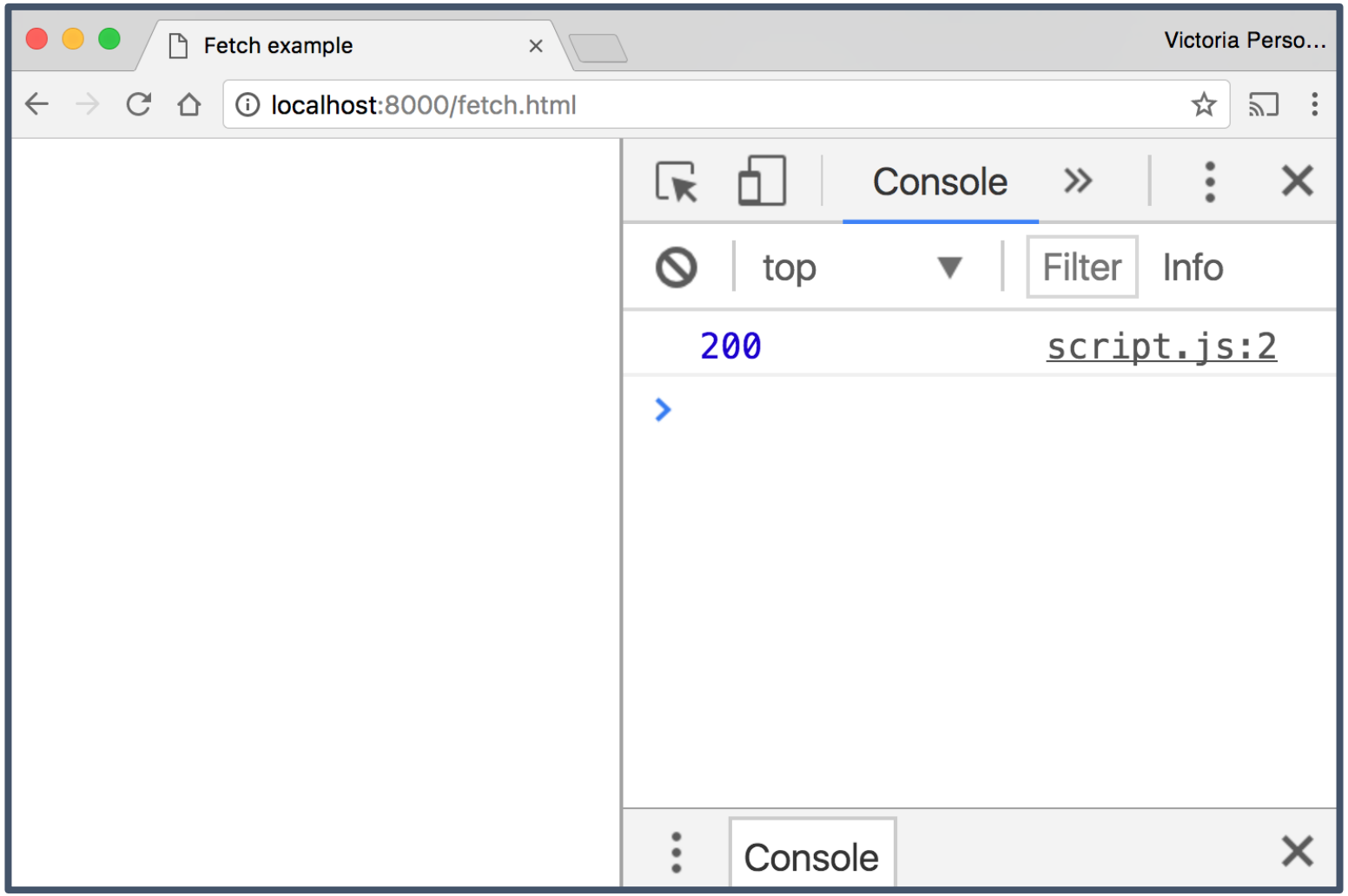
Put your html, txt files in folder */htdocs/fetch-demo/* of XAMPP
Run XAMPP & start Apache server

This now starts up a **server** that can load the files in the current directory over HTTP.

- We can access this server by navigating to: `http://localhost:8000/fetch-demo/`

Run xampp Apache service at port 8000

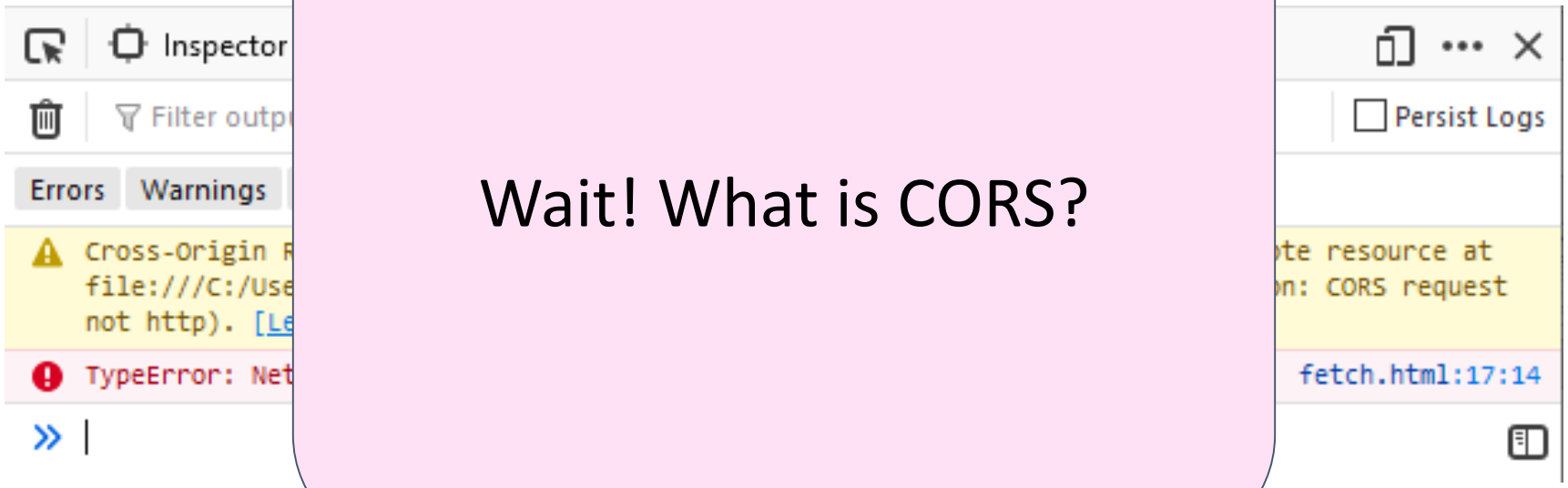




We got HTTP response 200, which is success! (codes)

Fetch error

If we try to load this in the browser, we get the following JavaScript error:



Wait! What is CORS?

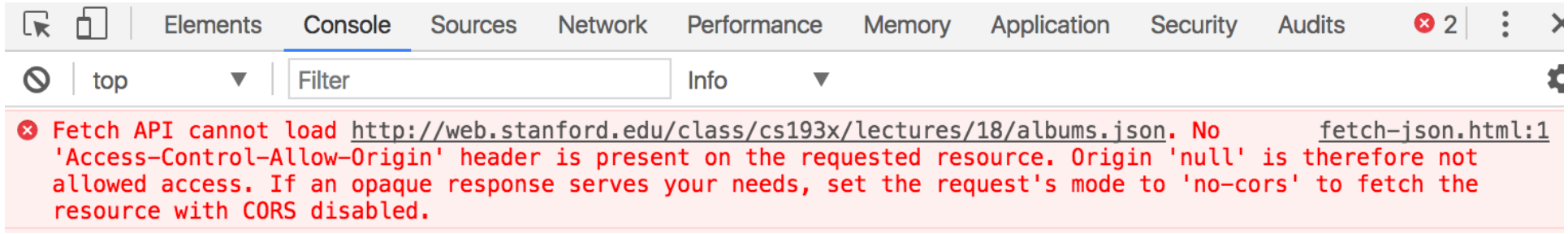
Notice that our onError function was also called.

Fetch gotchas

CORS error

If you try to `fetch()` this JSON file:
http://web.stanford.edu/class/cs193x/lectures/18/albums.json

You get this error:



Q: Why do we get this error, when the JSON file is served over HTTP?

CORS

CORS: **C**ross-**O**rigin **R**esource **S**haring (wiki)

- Browser policies for what resources a web page can load
- **Cross-origin**: between two different domains
  - If abc.com/users requests something from abc.com/search, it's still a **same-origin** request (not cross-origin) because it's the same domain
  - But if abc.com/foo requests something from xyz.com/foo, it's a **cross-origin** request.

CORS summarized

- You can make **same-origin** requests by default for any request type

- You can make **cross-origin** requests by default for:
  - Images loaded via <img>
  - CSS files loaded via <link>
  - JavaScript files loaded via <script>
  - Etc

- You **cannot** make cross-origin requests by default for:
  - Resources loaded via fetch() or XHR

CORS configuration

However, a web server can be configured to override these default rules:

- If you want to allow other domains to make `fetch()` requests to your servers, you can configure your server to allow them (e.g. on apache)
  - All 3rd party APIs do this, otherwise you couldn't access them

- If you don't want other domains to certain resources such as images, you can disallow them

In this class

In WPR, we will either be:

- Making same-origin requests
- Making requests on APIs that have allowed cross-origin access

So you don't need to do anything with CORS for WPR.

Still, CORS is good to know about:

- Helps you understand error messages
- You may have to deal with this in the future (common scenario: file:// trying to access an HTTP resource: HTTP resource must allow CORS for this to be allowed)

How do we get the data from `fetch()`?

Using Fetch

```
function onSuccess(response) {
  ..
}
fetch('images.txt').then(onSuccess);
```

- `response.status`: Status code for the request
- `response.text()`:
  - **Asynchronously** reads the response stream
  - **Returns a `Promise`** that resolves with the string containing the response stream data.

text() Promise

Q: How do we change the following code to print out the response body?

```
function onSuccess(response) {
  console.log(response.status);
}

function onError(error) {
  console.log('Error: ' + error);
}

fetch('images.txt')
    .then(onSuccess, onError);
```

```
function onStreamProcessed(text) {
  console.log(text);
}

function onResponse(response) {
  console.log(response.status);
  response.text().then(onStreamProcessed);
}

function onError(error) {
  console.log('Error: ' + error);
}

fetch('images.txt').then(onResponse, onError);
```

Chaining Promises

We want the following asynchronous actions to be completed in this order:

1. When the `fetch` completes, run `onResponse`
2. When `response.text()` completes, run `onStreamProcessed`

```
function onStreamProcessed(text) { … }
function onResponse(response) {
  response.text().then(onStreamProcessed);
}
fetch('images.txt').then(onResponse, onError);
```

We can rewrite this:

```
function onStreamProcessed(text) {
  console.log(text);
}

function onResponse(response) {
  response.text().then(onStreamProcessed);
}

function onError(error) {
  console.log('Error: ' + error);
}

fetch('images.txt').then(onResponse, onError);
```

We can rewrite this:

```
function onStreamProcessed(text) {
  console.log(text);
}

function onResponse(response) {
  return response.text();
}

function onError(error) {
  console.log('Error: ' + error);
}

fetch('images.txt')
    .then(onResponse, onError)
    .then(onStreamProcessed);
```

Chaining Promises

```
function onStreamProcessed(text) {
  console.log(text);
}

function onResponse(response) {
  return response.text();
}

fetch('images.txt')
    .then(onResponse, onError)
    .then(onStreamProcessed);
```

Chaining Promises

```
function onStreamProcessed(text) {
  console.log(text);
}

function onResponse(response) {
  return response.text();
}

const responsePromise = fetch('images.txt')
    .then(onResponse, onError)
responsePromise.then(onStreamProcessed);
```

The Promise returned by `onResponse` is effectively* the Promise returned by `fetch`. (*Not actually what's happening, but that's how we'll think about it for right now.)

Chaining Promises

```
function onStreamProcessed(text) {
  console.log(text);
}

function onResponse(response) {
  return response.text();
}

fetch('images.txt')
    .then(onResponse, onError)
    .then(onStreamProcessed);
```

If we don't think about it too hard, the syntax is fairly intuitive.

We'll think about this more deeply later!

Completed example

```
function onStreamProcessed(text) {
  const urls = text.split('\n');
  for (const url of urls) {
    const image = document.createElement('img');
    image.src = url;
    document.body.append(image);
  }
}
function onSuccess(response) {
  response.text().then(onStreamProcessed)
}
function onError(error) {
  console.log('Error: ' + error);
}

fetch('images.txt').then(onSuccess, onError);
```

JSON

JavaScript Object Notation

JSON: Stands for **J**ava**S**cript **O**bject **N**otation

- Created by Douglas Crockford
- Defines a way of **serializing** JavaScript objects
  - **to serialize**: to turn an object into a string that can be deserialized
  - **to deserialize**: to turn a serialized string into an object

JSON.stringify()

We can use the `JSON.stringify()` function to seralize a JavaScript object:

```
const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing']
};

const serializedBear =
JSON.stringify(bear);
console.log(serializedBear);
```

CodePen

JSON.parse()

We can use the `JSON.parse()` function to deseralize a JavaScript object:

```
const bearString = '{"name":"Ice Bear","hobbies":["knitting","cooking","dancing"]}';

const bear = JSON.parse(bearString);
console.log(bear);
```

CodePen

Fetch API and JSON

The Fetch API also has built-in support for JSON:

```
function onJsonReady(json) {
  console.log(json);
}

function onResponse(response) {
  return response.json();
}

fetch('images.json')
    .then(onResponse)
    .then(onJsonReady);
```

Return `response.json()` instead of `response.text()` and Fetch will essentially call `JSON.parse()` on the response string.

Why JSON?

Let's say we had a file that contained a list of albums.

Each album has:

- Title
- Year
- URL to album image

We want to display each album in chronological order.

Text file?

We could create a text file formatted consistently in some format that we make up ourselves, e.g.:

```
The Emancipation Of Mimi
2005
https://i.scdn.co/image/dca82bd9c1ccae90b09972027a408068f7a4d700

Daydream
1995
https://i.scdn.co/image/0638f0ddf70003cb94b43aa5e4004d85da94f99c

E=MC²
2008
https://i.scdn.co/image/bca35d49f6033324d2518656531c9a89135c0ea3

Mariah Carey
1990
```

Text file processing

We would have to write all this custom file processing code:

- Must convert numbers from strings
- If you ever add another attribute to the album, we'd have to change our array indices

```
function onTextReady(text) {
  const lines = text.split('\n\n');
  const albums = [];
  for (let i = 0; i < lines.length; i++) {
    const infoText = lines[i];
    const infoStrings = infoText.split('\n');
    const name = infoStrings[0];
    const year = infoStrings[1];
    const url = infoStrings[2];
    albums.push({
      name: name,
      year: parseInt(year),
      url: url
    });
  }
  ...
}
```

Live example / GitHub

JSON file

It'd be much more convenient to store the file in JSON format:

```
{
  "albums": [
    {
      "name": "The Emancipation Of Mimi",
      "year": 2005,
      "url":
"https://i.scdn.co/image/dca82bd9c1ccae90b09972027a408068f7a4d700
"
    },
    {
      "name": "Daydream",
      "year": 1995,
      "url":
"https://i.scdn.co/image/0638f0ddf70003cb94b43aa5e4004d85da94f99c
"
    },
    {
```

JSON processing

Since we're using JSON, we don't have to manually convert the response strings to a JavaScript object:

- JavaScript has built-in support to convert a JSON string into a JavaScript object.

```
function onJsonReady(json) {
  const albums = json.albums;
  ...
}
```

Live example / GitHub

JavaScript Object Notation

JSON: Stands for **J**ava**S**cript **O**bject **N**otation

- Created by Douglas Crockford
- Defines a way of **serializing** JavaScript objects
  - **to serialize**: to turn an object into a string that can be deserialized
  - **to deserialize**: to turn a serialized string into an object
- `JSON.stringify(`***object***`)` returns a string representing ***object*** serialized in JSON format
- `JSON.parse(`***jsonString***`)` returns a JS object from the ***jsonString*** serialized in JSON format

JSON.stringify()

We can use the `JSON.stringify()` function to seralize a JavaScript object:

```
const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing']
};

const serializedBear = JSON.stringify(bear);
console.log(serializedBear);
```

CodePen

JSON.parse()

We can use the `JSON.parse()` function to deseralize a JavaScript object:

```
const bearString = '{"name":"Ice
Bear","hobbies":["knitting","cooking","dancing"]}';

const bear = JSON.parse(bearString);
console.log(bear);
```

CodePen

Why JSON?

JSON is a useful format for storing data that we can load into a JavaScript API via fetch().

Let's say we had a list of Songs and Titles.

- If we stored it as a text file, we would have to know how we are separating song name vs title, etc
- If we stored it as a JSON file, we can just deserialize the object.

JSON

songs.json

```json
{
  "cranes": {
    "fileName": "solange-cranes-kaytranada.mp3",
    "artist": "Solange",
    "title": "Cranes in the Sky [KAYTRANADA Remix]"
  },
  "timeless": {
    "fileName": "james-blake-timeless.mp3",
    "artist": "James Blake",
    "title": "Timeless"
  },
  "knock": {
    "fileName": "knockknock.mp4",
    "artist": "Twice",
    "title": "Knock Knock"
  },
  "deep": {
    "fileName": "janet-jackson-go-deep.mp3",
    "artist": "Janet Jackson",
    "title": "Go Deep [Alesia Remix]"
  },
  "discretion": {
    "fileName": "mitis-innocent-discretion.mp3",
    "artist": "MitiS",
    "title": "Innocent Discretion"
  },
  "spear": {
    "fileName": "toby-fox-spear-of-justice.mp3",
    "artist": "Toby Fox",
    "title": "Spear of Justice"
  }
}
```

Fetch API and JSON

The Fetch API also has built-in support for json:

```
function onStreamProcessed(json) {
  console.log(json);
}

function onResponse(response) {
  return response.json();
}

fetch('songs.json')
    .then(onResponse, onError)
    .then(onStreamProcessed);
```

Query parameters

You can pass parameters to HTTP GET requests by adding **query parameters** to the URL:

```
?type=album&q=beyonce
```

- Defined as key-value pairs
  - ***param=value***
- The first query parameter starts with a **?**
- Subsequent query parameters start with **&**

Reminder: HTML elements

Single-line text input:

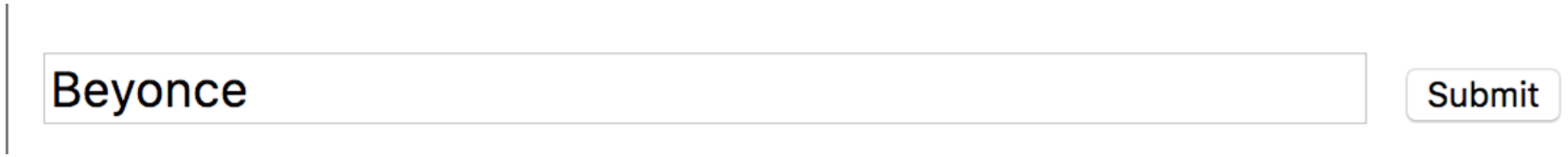
```
<input type="text" />
```

In JavaScript, you can read and set the input text via `inputElement.value`

Some other input types:

- Select
- Textarea
- Checkbox

Form submit



Q: What if you want the form to submit after you click "enter"?

Form submit

1. Wrap your input elements in a `<form>`

```
<form>
  <input type="text" id="artist-text" />
  <input type="submit" />
</form>
```

You should also use `<input type="submit">` instead of `<button>` for the reason on the next slide...

Form submit

2. Listen for the `'submit'` event on the form element:

```
const form = document.querySelector('form');
form.addEventListener('submit', this._onSubmit);
```

This is why you want to use `<input type="submit">` instead of `<button>` -- the 'submit' event will fire on click for but not `<button>`.

Form submit

3. Prevent the default action before handling the event through `event.preventDefault()`:

```
_onSubmit(event) {
  event.preventDefault();
  const textInput = document.querySelector('#artist-text');
  const query = encodeURIComponent(textInput.value);

  this.albumUrls = [];
  fetch(SPOTIFY_PATH + query)
      .then(this._onResponse)
      .then(this._onJsonReady);
}
```

The page will refresh on submit unless you explicitly prevent it.