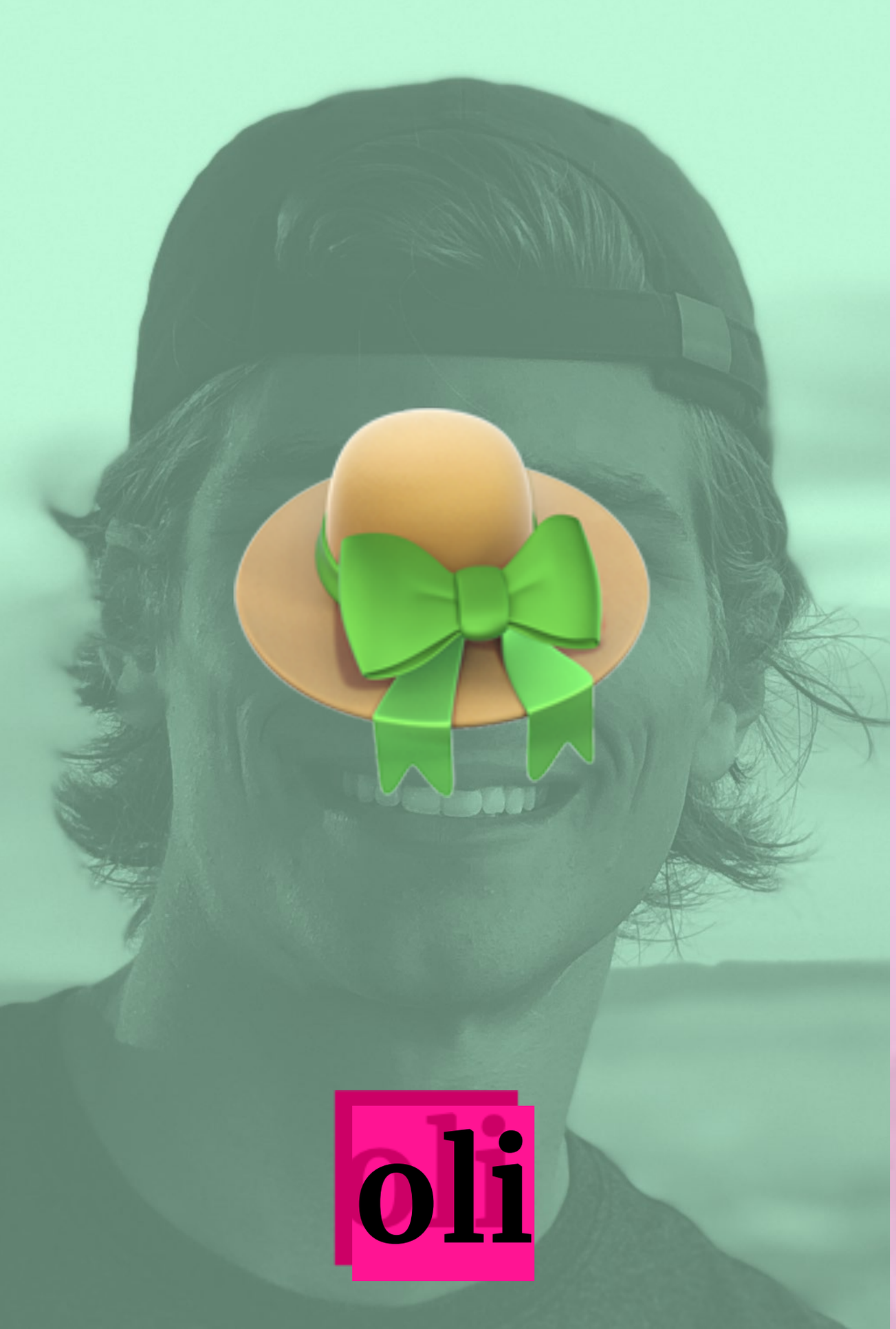
web programming

fetching resources


oli


rec

agenda

1. Fetch API + promises & `async/await`
2. JSON
3. CORS

1. Fetch API

what is Fetch API?

- interface for fetching resources
- provides `Request` and `Response` objects, as well as a global `fetch()` method
- better alternative to `XMLHttpRequest`
- allows us to do AJAX

what is AJAX?

- **A**synchronous **J**avaScript **A**nd **X**ml
- not a technology in itself, but a **combination of technologies**
- allows a web app to make updates to the UI without reloading the browser page
- makes the app faster and more responsive to user interaction

`fetch()`

- easy way to fetch resources asynchronously across the network
- uses Promises

Promises

- objects used to manage asynchronous results
- allow to add callbacks via the `then()` method
- are **chainable**

`fetch()` example

```
fetch('https://foo.example/bar')
  .then(response => response.json())
  .then(json => {
    console.log(JSON.stringify(json));
  });
```

request options

- `fetch()` optionally takes a second parameter `init`
- the `init` object allows to specify request options, e.g. request method, headers, body, etc.

request options example

```
fetch('https://foo.example/bar', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: '{ "foo": 42 }',
})
  .then(response => {
    console.log(response.status);
  });
```

synchronous `fetch()`?

```
// THIS CODE DOES NOT WORK!
const response = fetch('https://foo.example/bar');
const json = response.json();
console.log(JSON.stringify(json));
// YET IT LOOKS MUCH CLEANER!
```

`async/await`

- allows to write asynchronous code that looks like synchronous code
- works with Promises
- two keywords: `async` & `await`
  - `async` is used in the function definition and specifies that the function always returns a Promise
  - `await` is used inside an async function and makes JavaScript wait until the Promise is settled

async/await fetch()

```
async function loadFoo() {
  const response = await fetch('https://foo.example/bar');
  const json = await response.json();
  console.log(JSON.stringify(json));
}

loadFoo();
```

2. JSON

what is JSON?

- **J**ava**S**cript **O**bject **N**otation
- syntax for **serializing** JavaScript values
- **to serialize** is to convert a value to a format that can be stored or transmitted, and later deserialized
- **to deserialize** is to reconstruct the original value from the serialized source

JSON.stringify()

- serializes a JavaScript value into a string:

```
const foo = {
    bar: 42,
};

const serializedFoo = JSON.stringify(foo);

console.log(serializedFoo);
```

JSON.parse()

- deserializes a string into a JavaScript value:

```
const serializedFoo = '{"bar":42}';

const foo = JSON.parse(serializedFoo);

console.log(foo);
```

why JSON?

- useful to transmit data from server to web app
- lightweight*
- built-in support in Fetch API, i.e. `response.json();`

(*): compared to alternatives such as XML

3. CORS

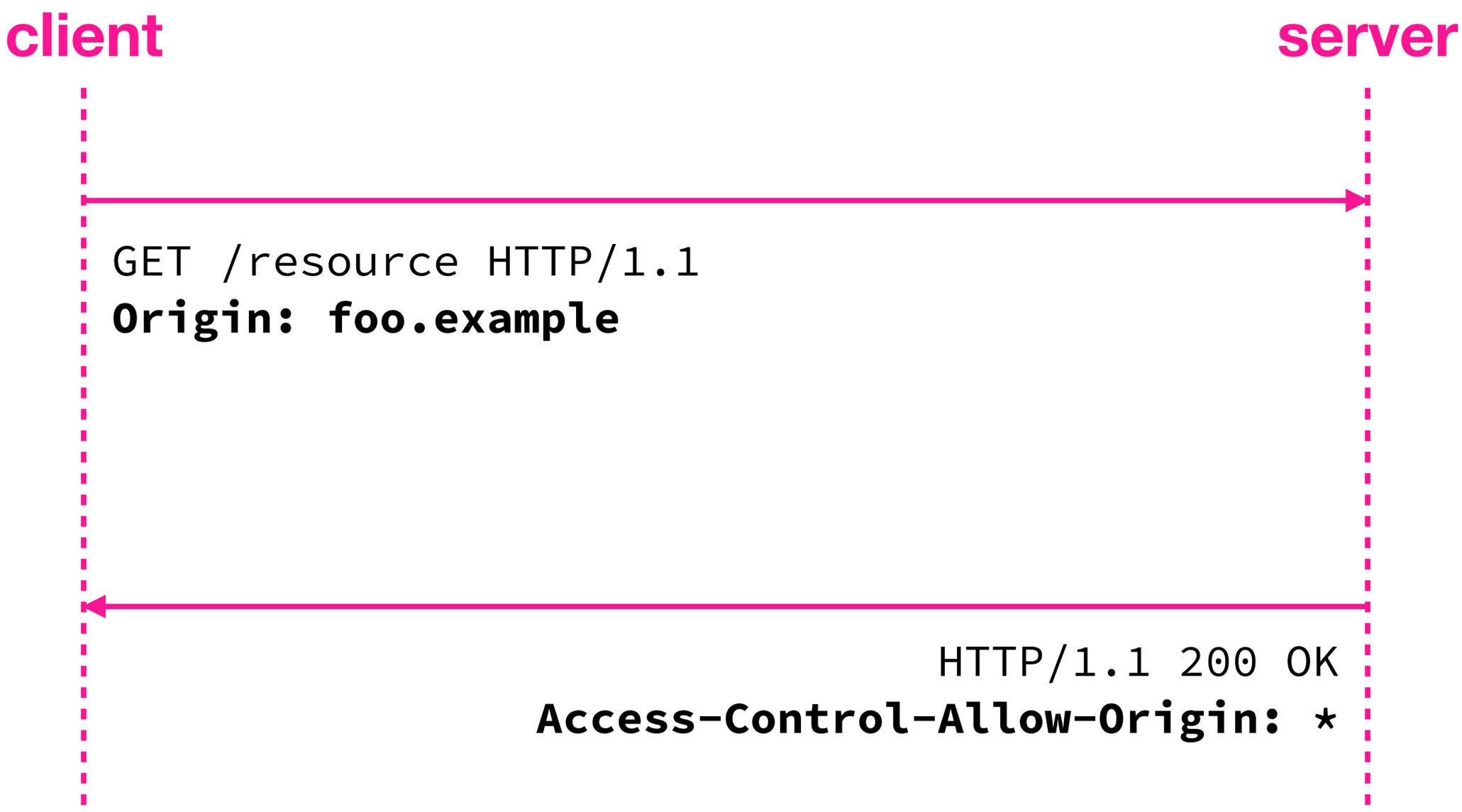
same-origin policy

- critical security mechanism
- implemented by browsers
- restricts a web app to **only access resources from the same origin the application was loaded from**
- origin = protocol + host + port

what is CORS?

- **C**ross-**O**rigin **R**esource **S**haring
- allows AJAX requests to skip the same-origin policy and **access resources from different origins**
- uses **HTTP headers**

example


client
server
GET /resource HTTP/1.1
Origin: foo.example
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *

enabling CORS in Express

```
const cors = require('cors');
const express = require('express');

const app = express();

app.use(cors());

// ...
```

<EXERCISE>

make an API and fetch its data from your web app

<EXERCISE>

How?

[api] create a new Express server with just one endpoint that responds with some hardcoded JSON.

[web] use `fetch()` to get the data and present it to the user.

<EXERCISE>

e.g.

[api] `/products` endpoint with an array of products.

[web] modify the DOM to show a list of, `{{name}}: {{price}}`

thanks!