Interactive Web Programming

1st semester of 2021

Murilo Camargos
(**murilo.filho@fgv.br**)

Heavily based on **Victoria Kirst** slides

Schedule

Today:

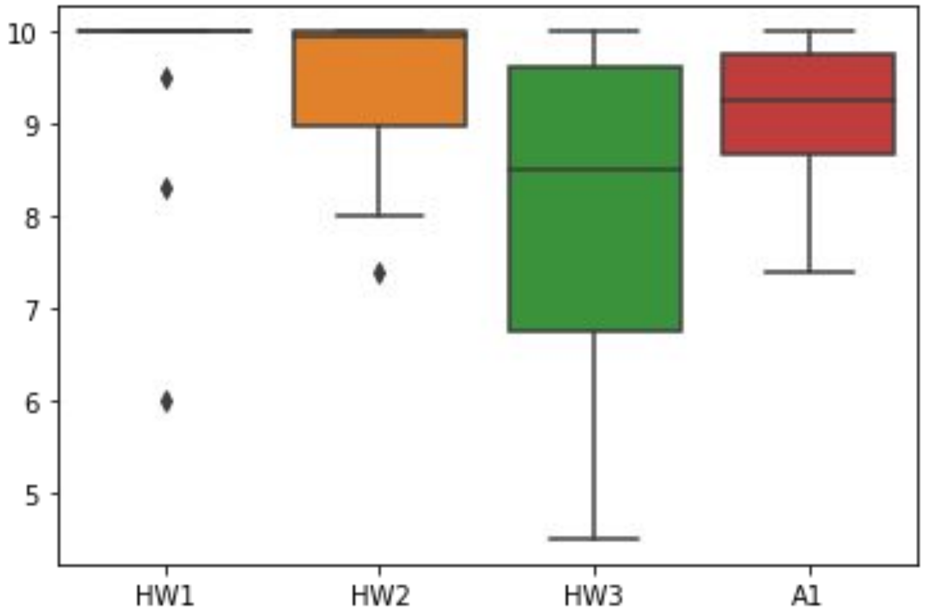
- Fetch
    - JSON
    - Fetch in an class
- Promises
- Querying REST APIs
    - Form submission

- **HW4 out!**

A quick note on HW3 and A1

General stuff:

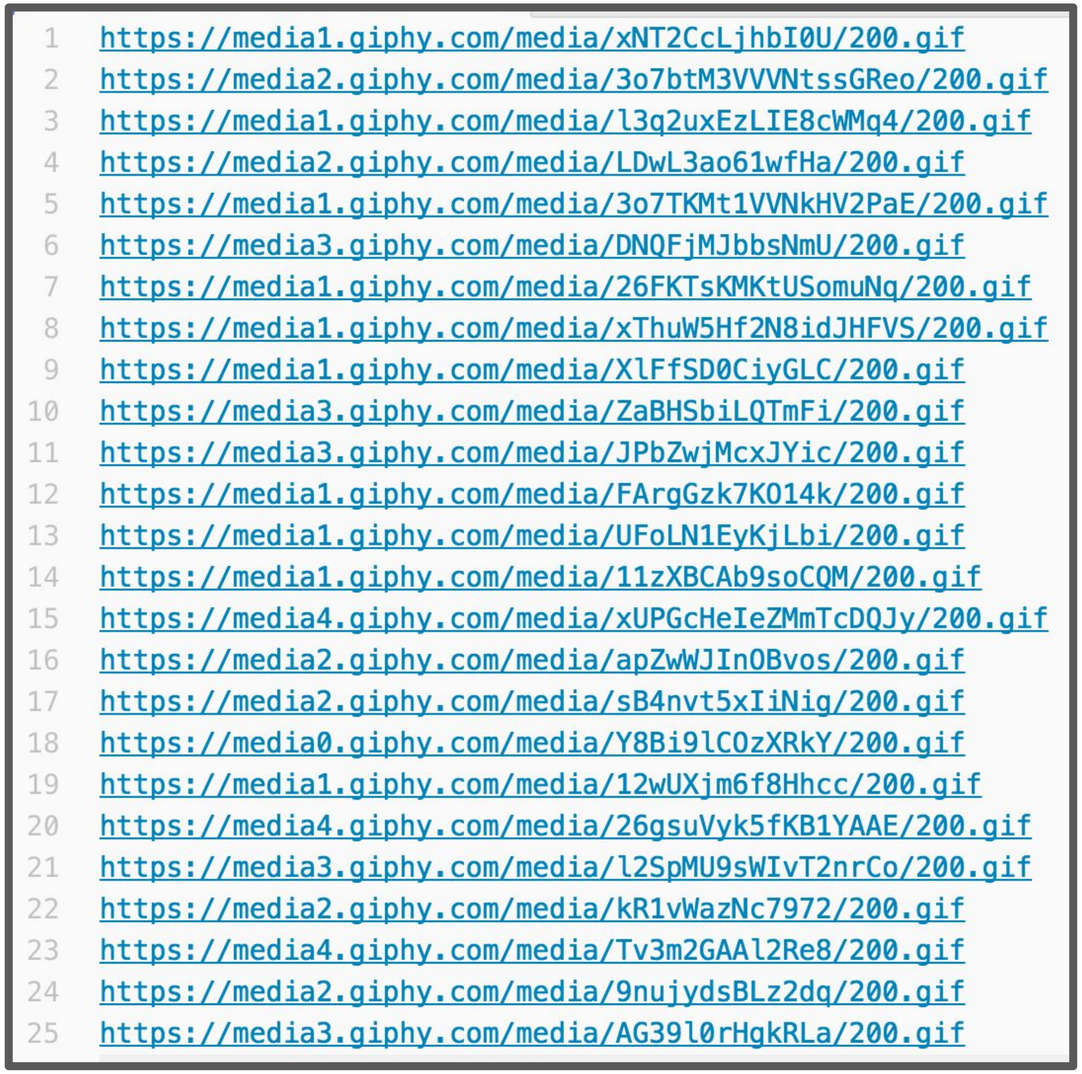
- I sent a feedback for each one of you on **sunday** at **19h**. Probably went to SPAM! :(
- If you **didn't receive anything** from me, please, send me an email **ASAP** (**murilo.filho@fgv.br**).



Loading data from files

Loading data from a file

What if you had a list of URLs in a text file that you wanted to load as images 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
```

Fetch API

Fetch API

`fetch()`: Function to load resources in JavaScript

```
fetch(pathToResource)
    .then(onResponse)
    .then(onResourceReady);
```

onResponse:

- Return `response.text()` from this function to get the resource as a string in ***onResourceReady***

onResourceReady:

- Gets the resource as a parameter when it's ready

Fetch API

```
function onTextReady(text) {
  // do something with text
}

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

fetch('images.txt')
    .then(onResponse)
    .then(onTextReady);
```

Completed example

```
function onTextReady(text) {
  const urls = text.split('\n');
  for (const url of urls) {
    const image = document.createElement('img');
    image.src = url;
    document.body.append(image);
  }
}

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

fetch('images.txt')
    .then(onResponse)
    .then(onTextReady);
```

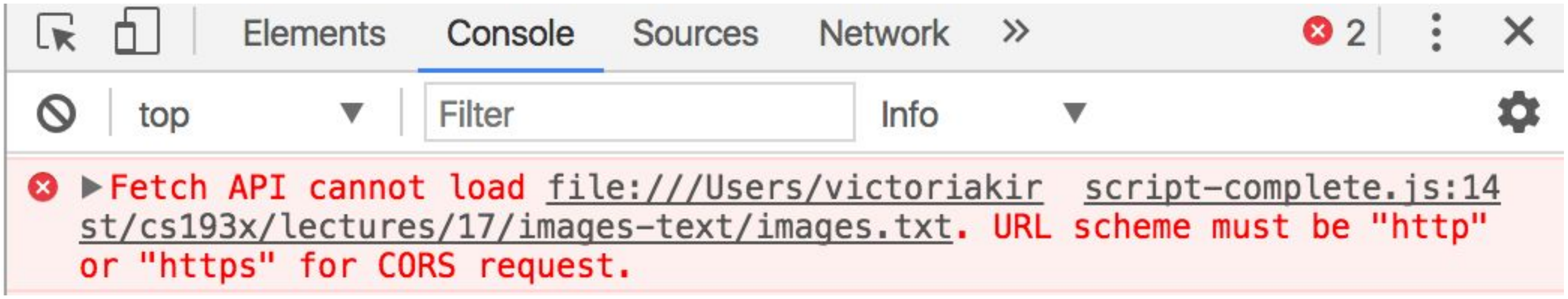
Completed example

```
function onTextReady(text) {
  const urls = text.split('\n');
  for (const url of urls) {
    const image = new Image();
    image.src = url;
    document.body.append(image);
  }
}

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

fetch('images.txt')
    .then(onResponse)
    .then(onTextReady);
```

Live example

fetch() limitations

- You cannot fetch a resource that is hosted on file://
  - You must serve your resource over HTTP / HTTPS



```
Fetch API cannot load file:///Users/victoriakir  script-complete.js:14
st/cs193x/lectures/17/images-text/images.txt. URL scheme must be "http"
or "https" for CORS request.
```

Serve over HTTP

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

```
$ python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
```

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/`

Note: Fetch Polyfill

Fetch is supported on all major browsers, though Safari added support only within the last couple of months

- If you need to support older browsers, add a Fetch Polyfill the way we did with Pointer Events
- (We've done this for you in HW4 starter code)

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","danci
ng"]}';

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
https://i.scdn.co/image/82f13700dfa78fa877a8cdaad725ad552c0c0653
```

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.

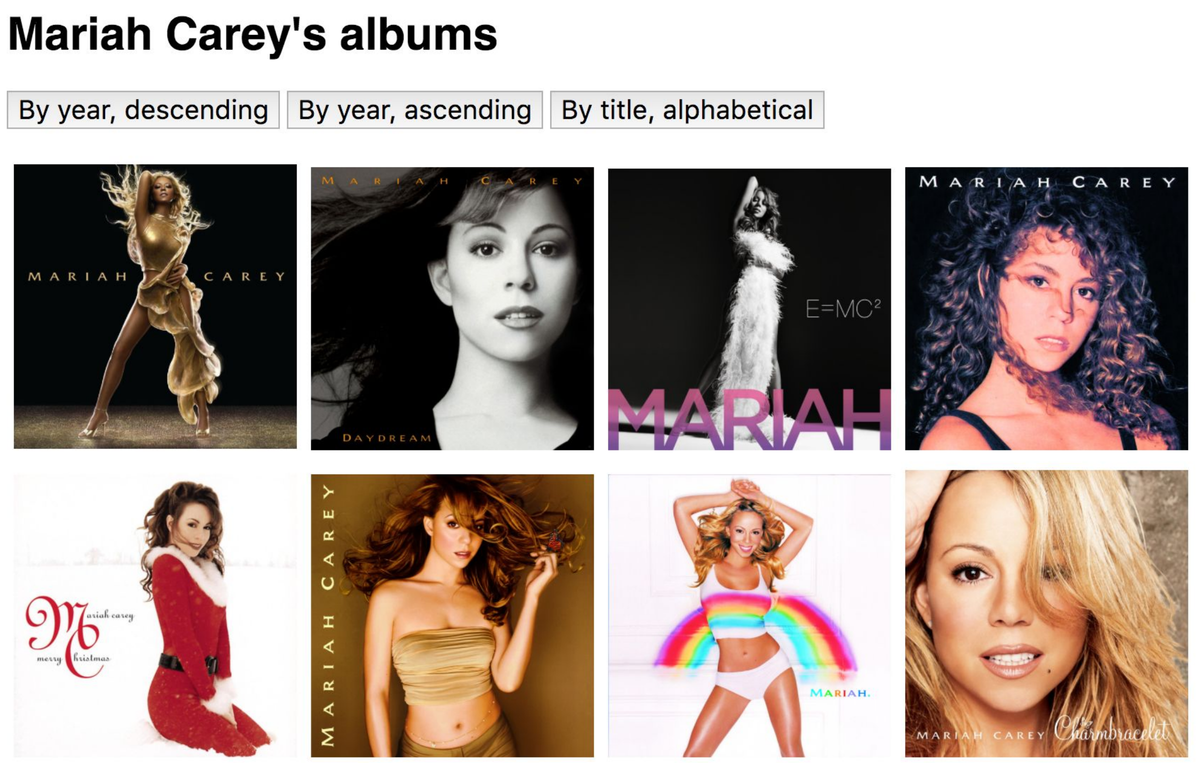
```
function onJsonReady(json) {
  const albums = json.albums;
  ...
}
```

Live example / GitHub

Fetch in a class

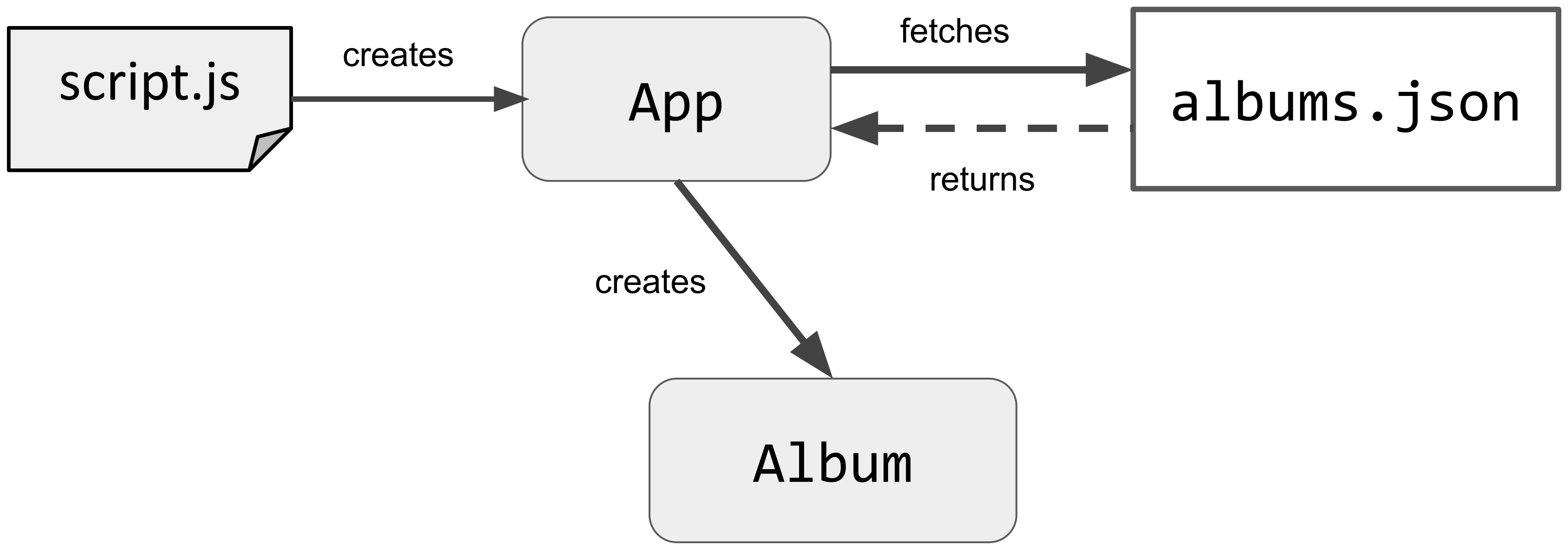
Discography page

Let's write a web page that lists the Mariah Carey albums stored in albums.json and lets us sort the albums: (demo)

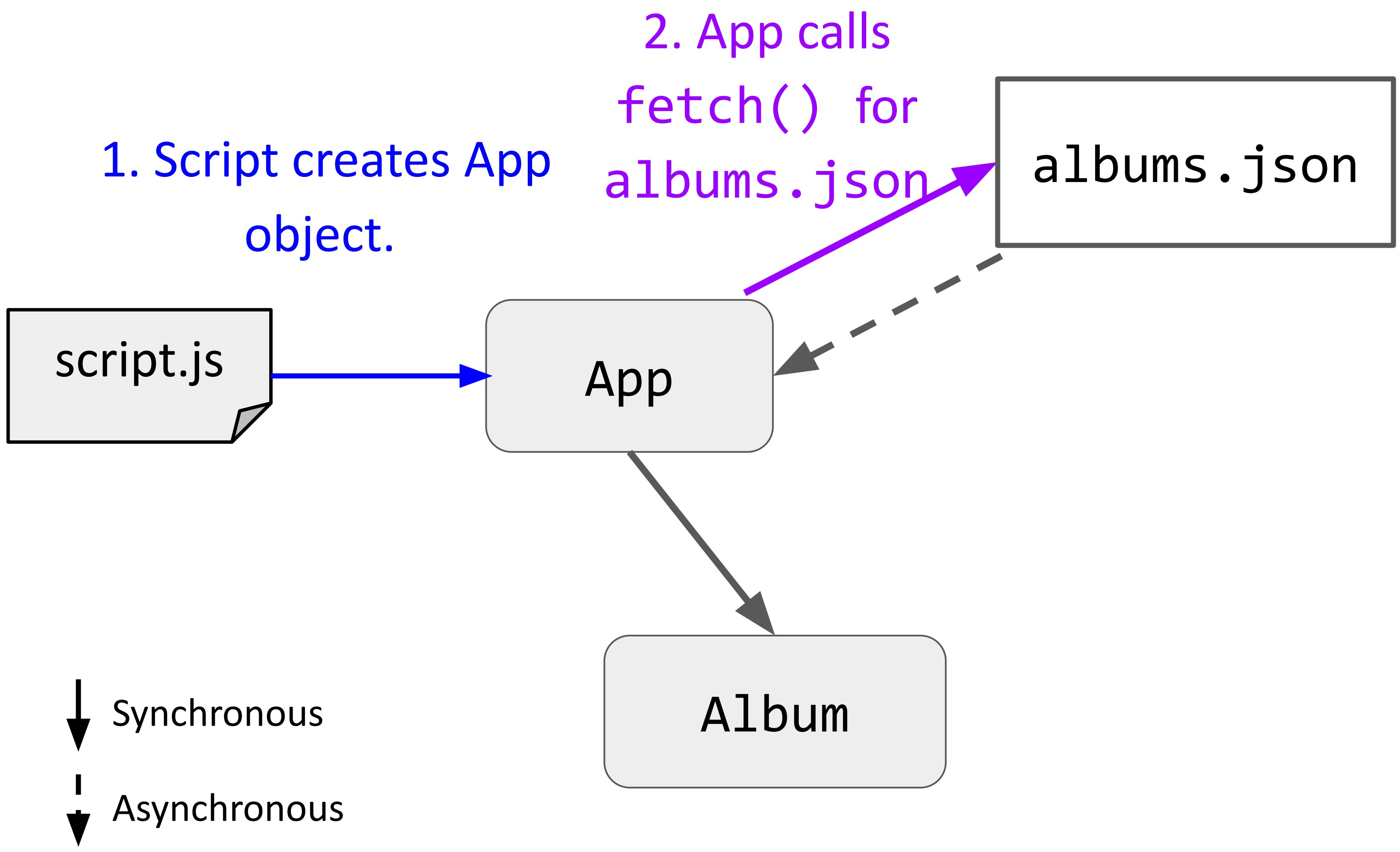

Mariah Carey's albums

Class diagram

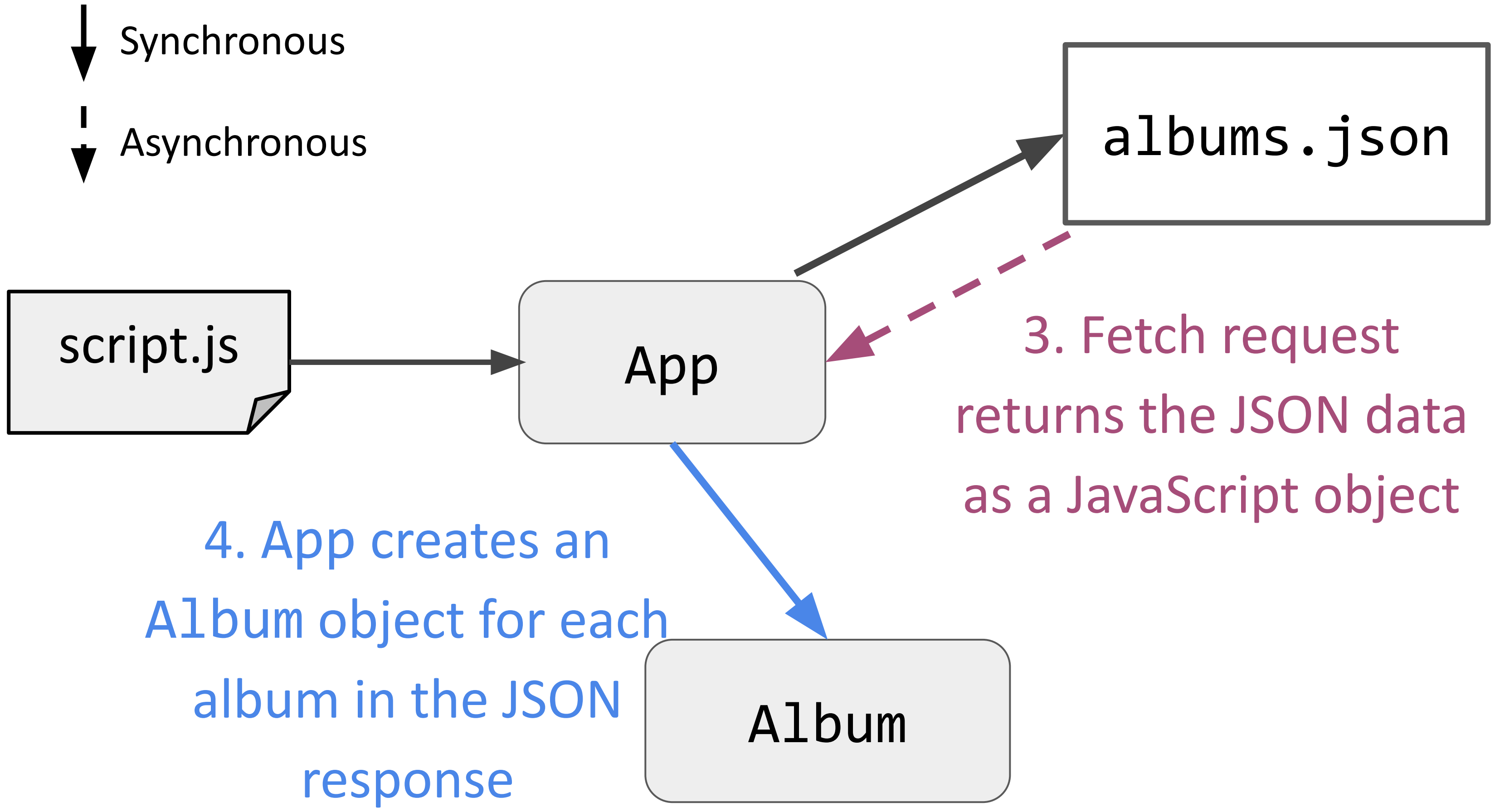
The class diagram is going to look something like this:



Album fetch()

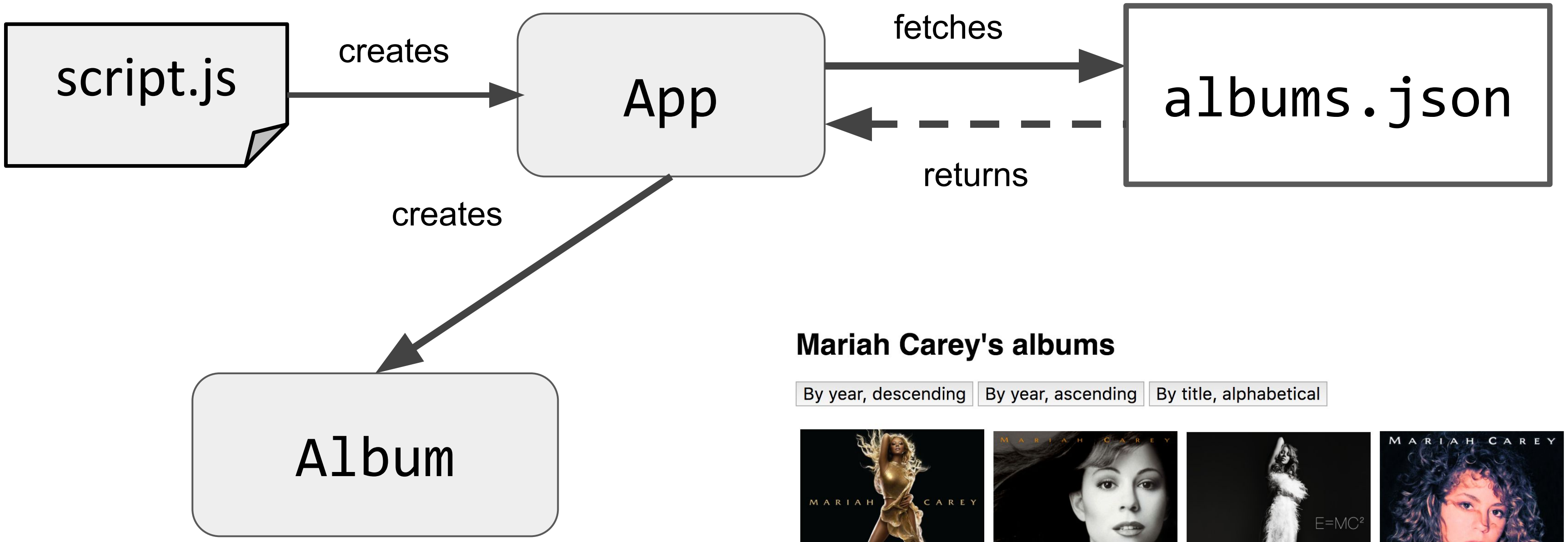

2. App calls
fetch() for
albums.json
1. Script creates App
object.
script.js
App
albums.json
Album
Synchronous
Asynchronous

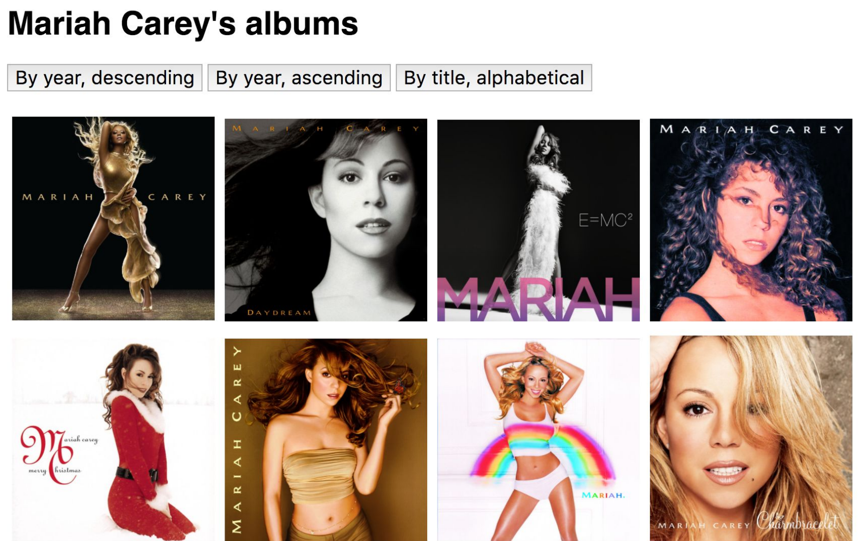
Album fetch()


Synchronous
Asynchronous
albums.json
script.js
App
3. Fetch request returns the JSON data as a JavaScript object
4. App creates an Album object for each album in the JSON response
Album

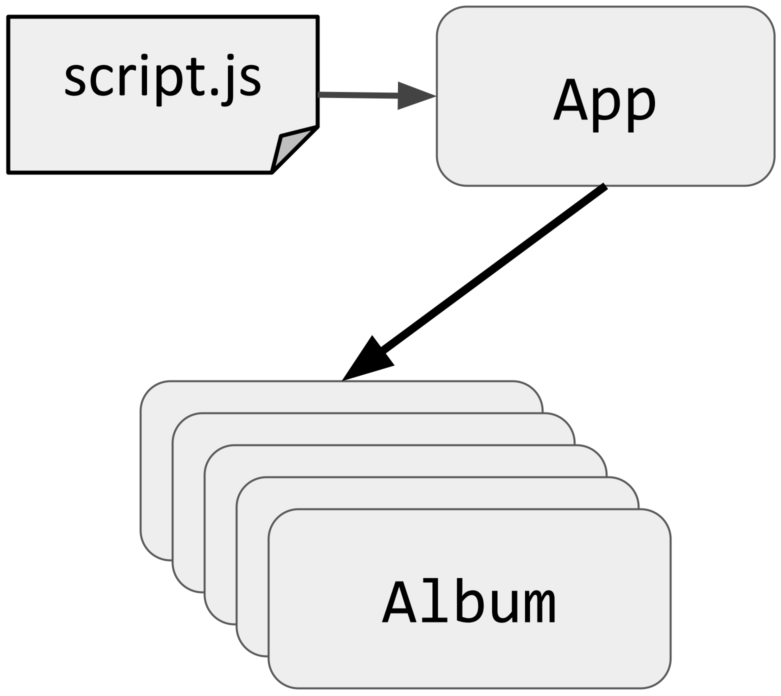
Discography page

Q: How do we begin to implement this??




Mariah Carey's albums

Getting started



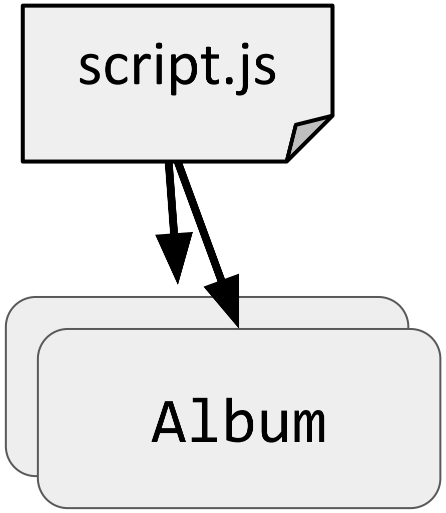
Suggestion:

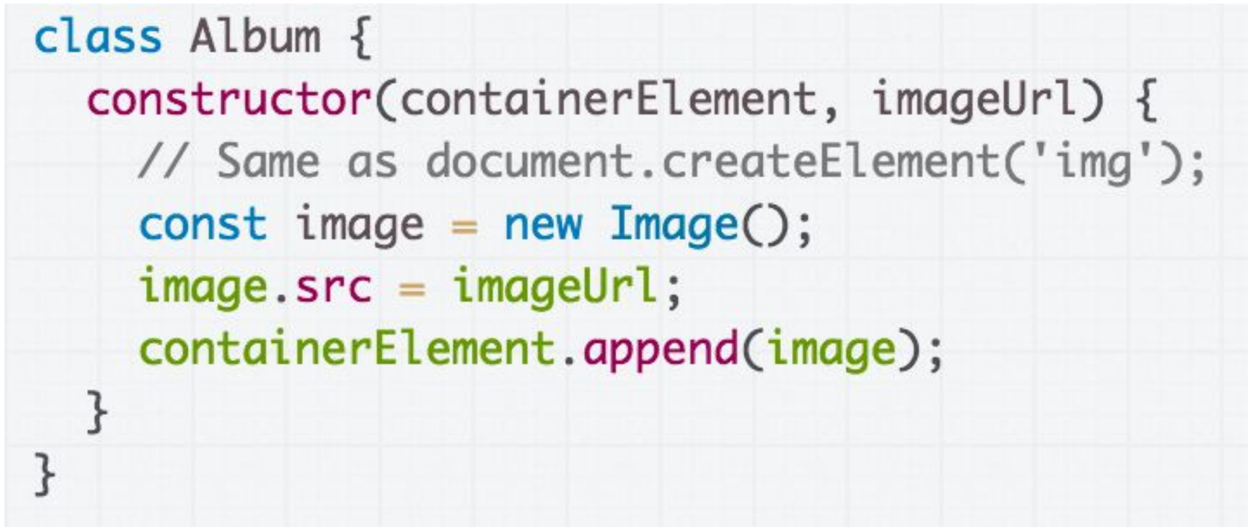
Implement the `Album` class first!

- The App class will have to use the `Album` class, meaning it is dependent on the `Album` class.
- The `Album` class doesn't have any dependencies, so let's create that first.

Starter

Milestone 1: Album



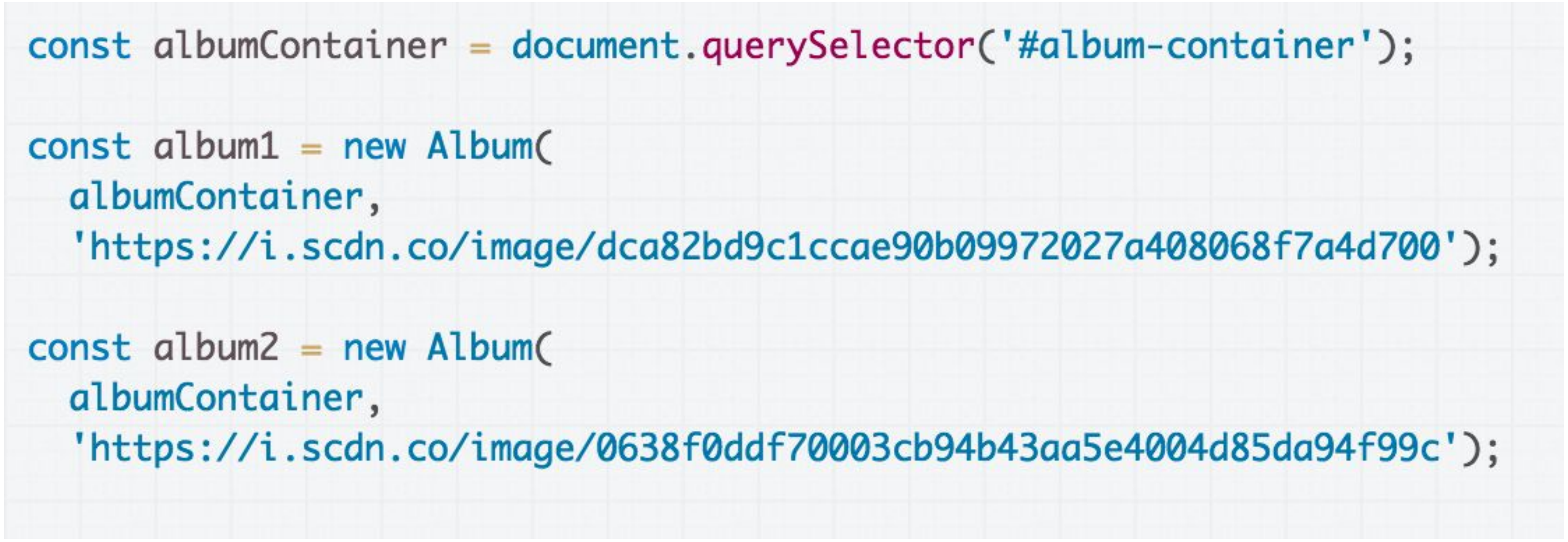


```
class Album {
  constructor(containerElement, imageUrl) {
    // Same as document.createElement('img');
    const image = new Image();
    image.src = imageUrl;
    containerElement.append(image);
  }
}
```

For your first step, just implement the `Album` class: ignore App/`fetch()`/etc for now.

Milestone 1: Album

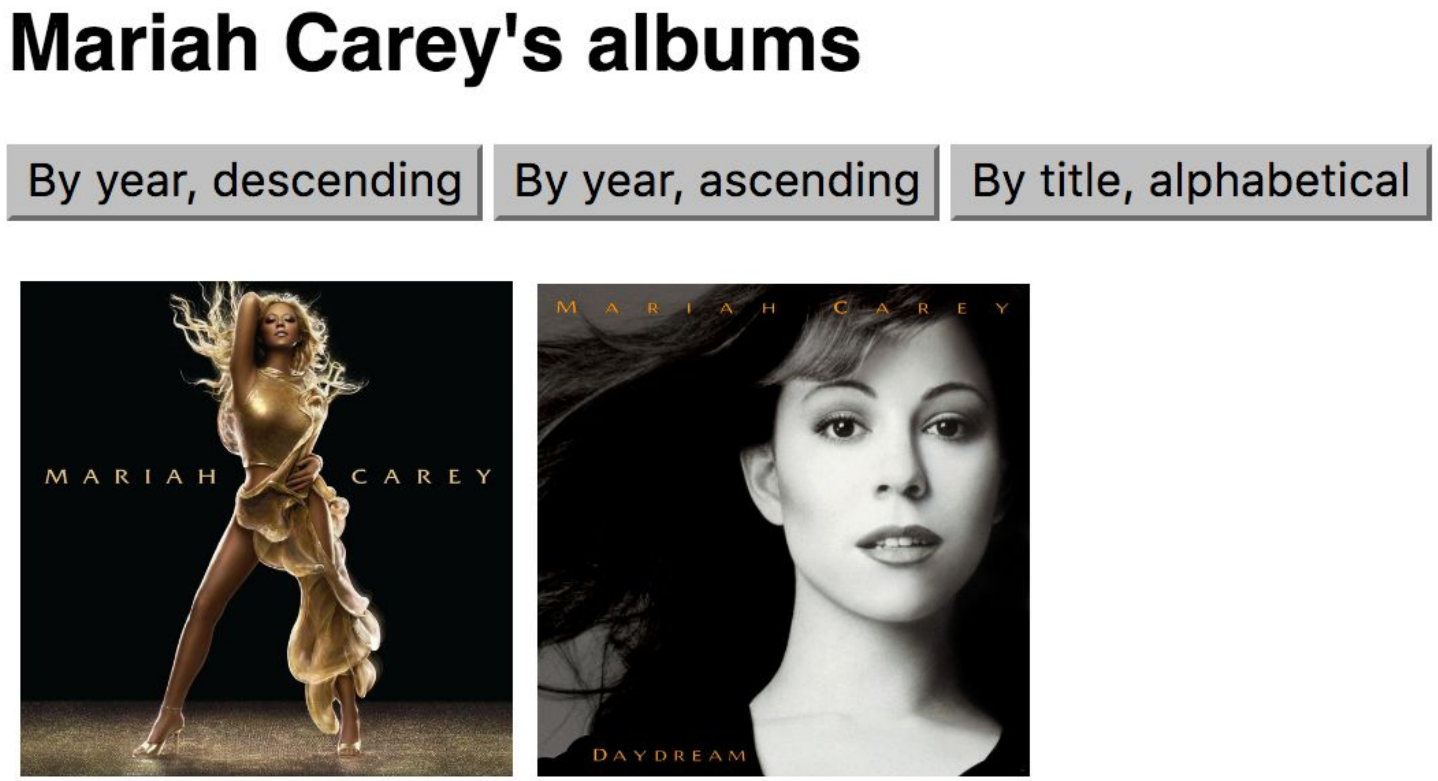
Modify `script.js` to create two `Album`s.



```
const albumContainer = document.querySelector('#album-container');

const album1 = new Album(
  albumContainer,
  'https://i.scdn.co/image/dca82bd9c1ccae90b09972027a408068f7a4d700');

const album2 = new Album(
  albumContainer,
  'https://i.scdn.co/image/0638f0ddf70003cb94b43aa5e4004d85da94f99c');
```

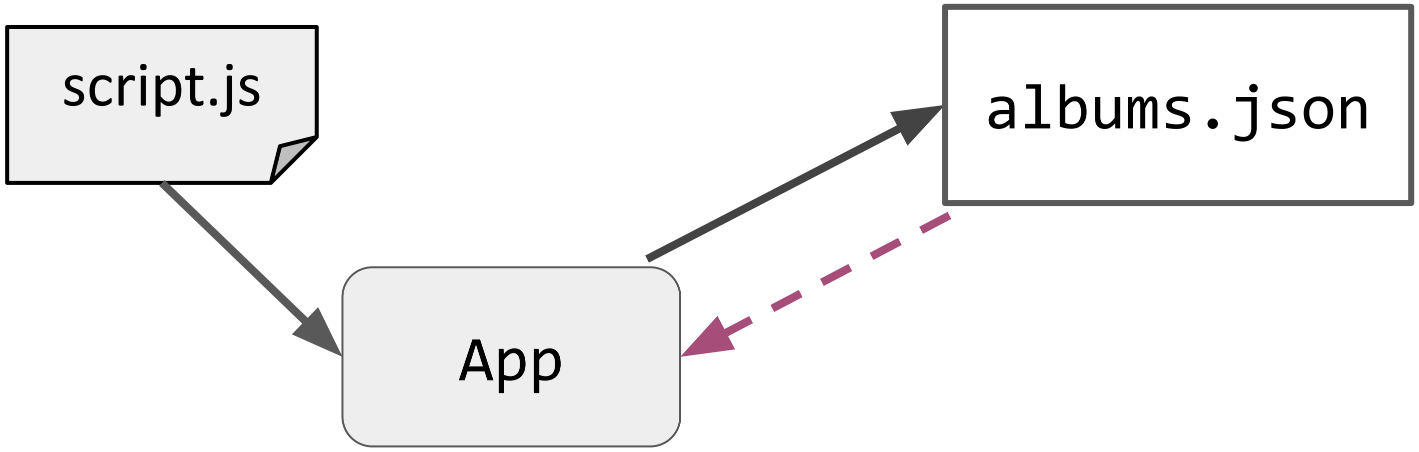
Milestone 1: Album

Milestone 1: CodePen / page



Mariah Carey's albums

Milestone 2: Print album info

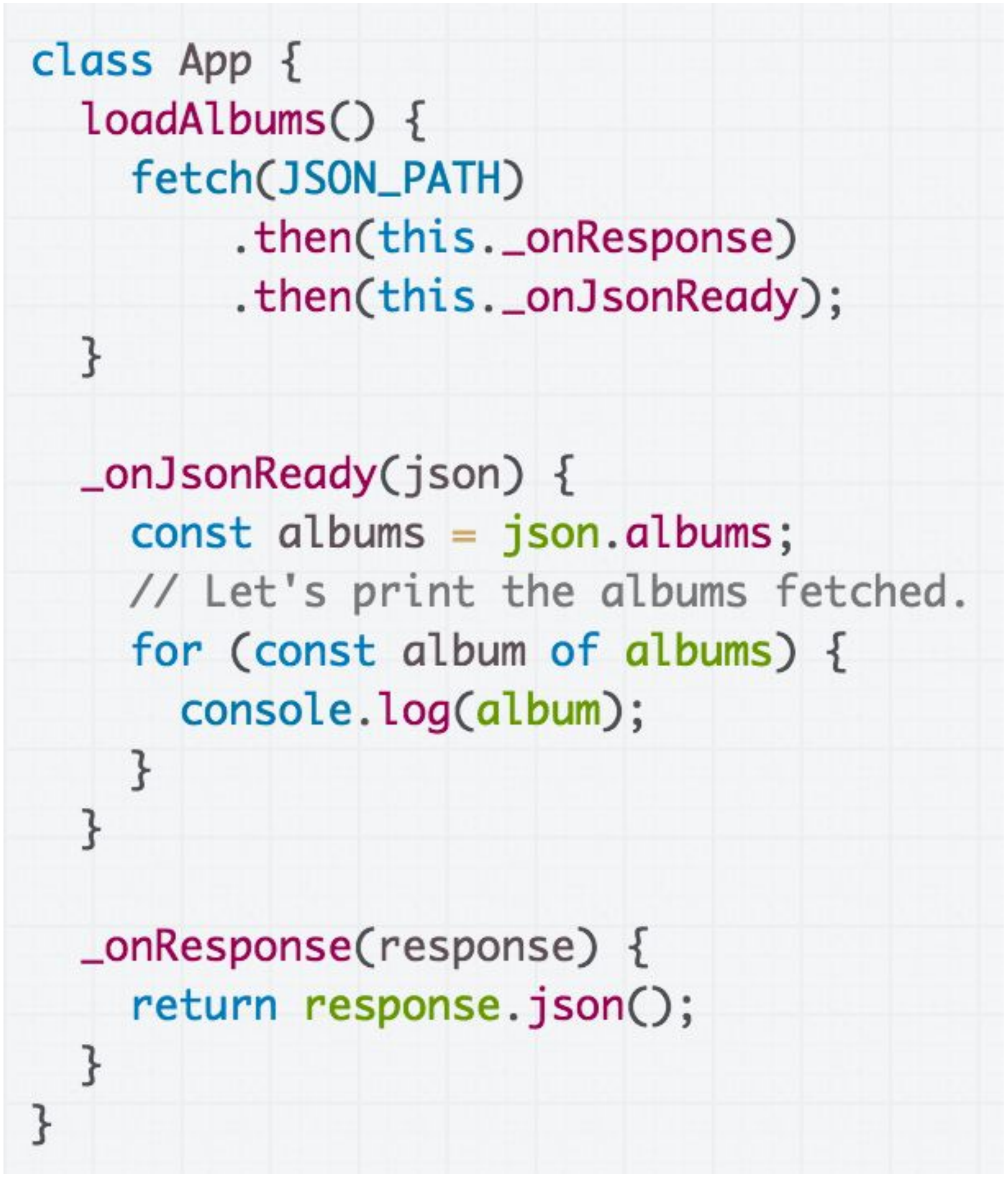


Suggestion: Implement the `fetch()` next!

- The App class is going to fetch data from albums.json, then it will create `Albums` based on that data.
- Let's implement `fetch()` first and make sure it works by printing out the results to the console.

Create a method `loadAlbums()` that calls `fetch()` like we did in the previous examples.

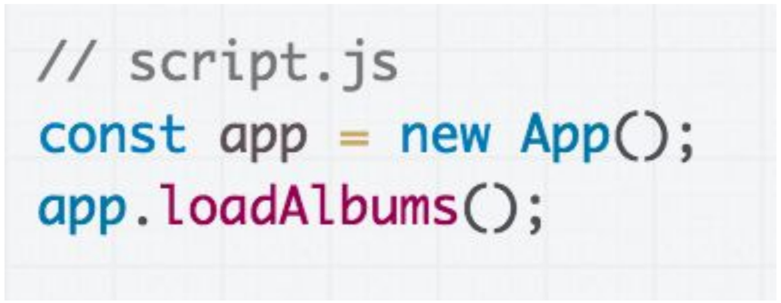
(Note: We don't have to define a constructor if we don't want to do in the constructor.)



```
class App {
  loadAlbums() {
    fetch(JSON_PATH)
        .then(this._onResponse)
        .then(this._onJsonReady);
  }

  _onJsonReady(json) {
    const albums = json.albums;
    // Let's print the albums fetched.
    for (const album of albums) {
      console.log(album);
    }
  }

  _onResponse(response) {
    return response.json();
  }
}
```

Milestone 2: Print album info

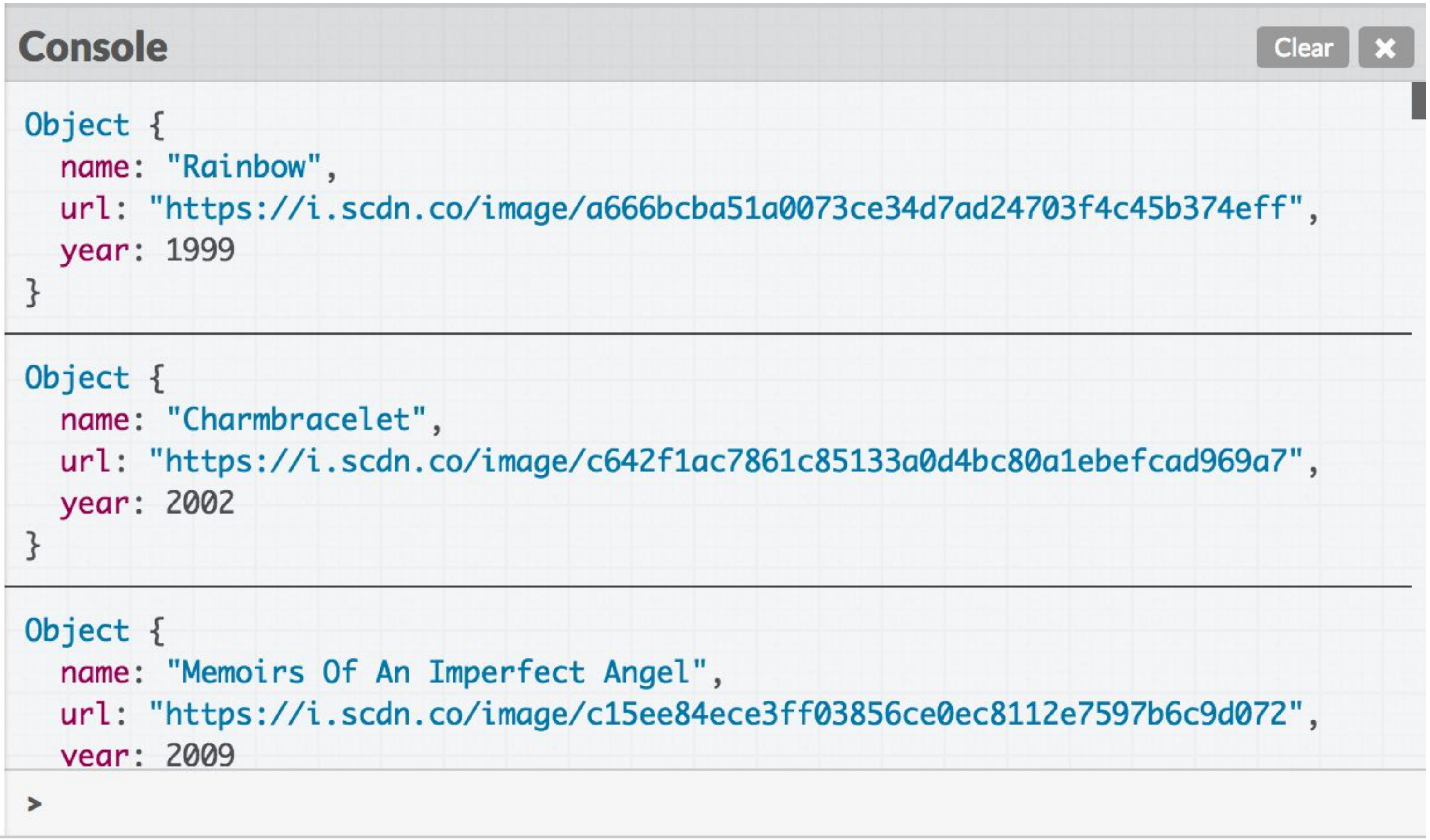
Modify `script.js` to create an App and call its `loadAlbums()` method.



```
// script.js
const app = new App();
app.loadAlbums();
```

Milestone 2: Print album info

Milestone 2: CodePen / page



Console

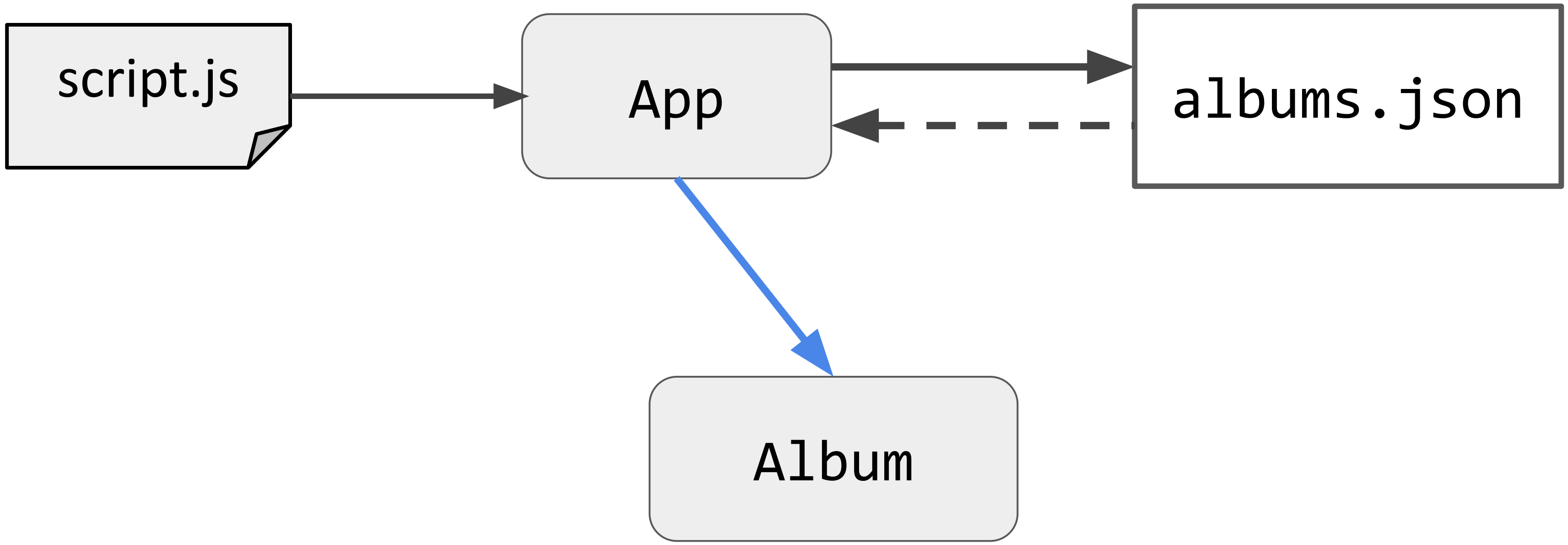
Clear ✖

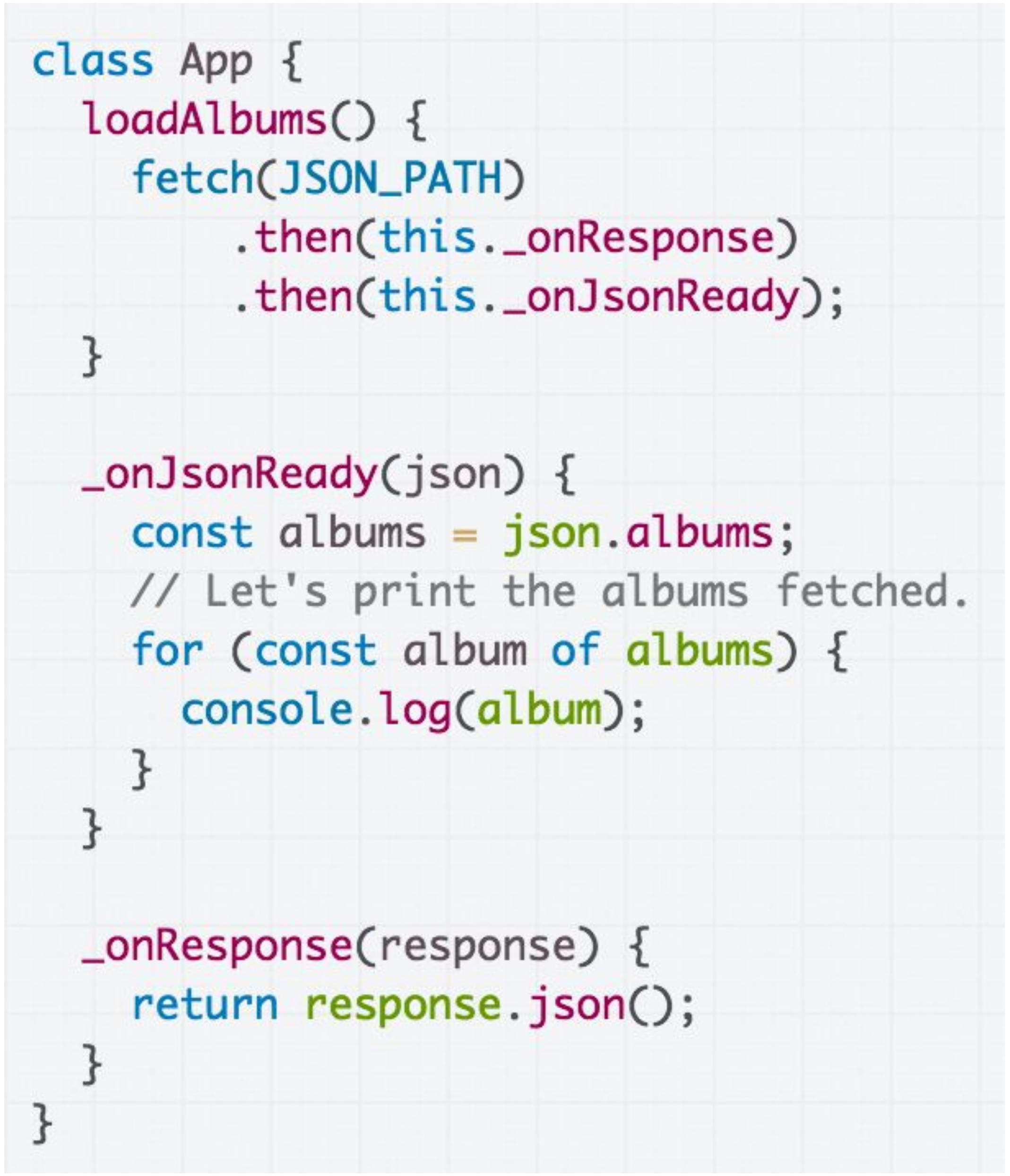
```
Object {
   name: "Rainbow",
   url: "https://i.scdn.co/image/a666bcba51a0073ce34d7ad24703f4c45b374eff",
   year: 1999
}
Object {
   name: "Charmbracelet",
   url: "https://i.scdn.co/image/c642f1ac7861c85133a0d4bc80a1ebefcad969a7",
   year: 2002
}
Object {
   name: "Memoirs Of An Imperfect Angel",
   url: "https://i.scdn.co/image/c15ee84ece3ff03856ce0ec8112e7597b6c9d072",
   year: 2009
>
```

Milestone 3: Create Albums

Now let's connect App and Album:

- The App class is supposed to create `Albums` based on the data fetched from the JSON file.
- Since `Album` and `fetch()` are working separately, now let's try making them work together.



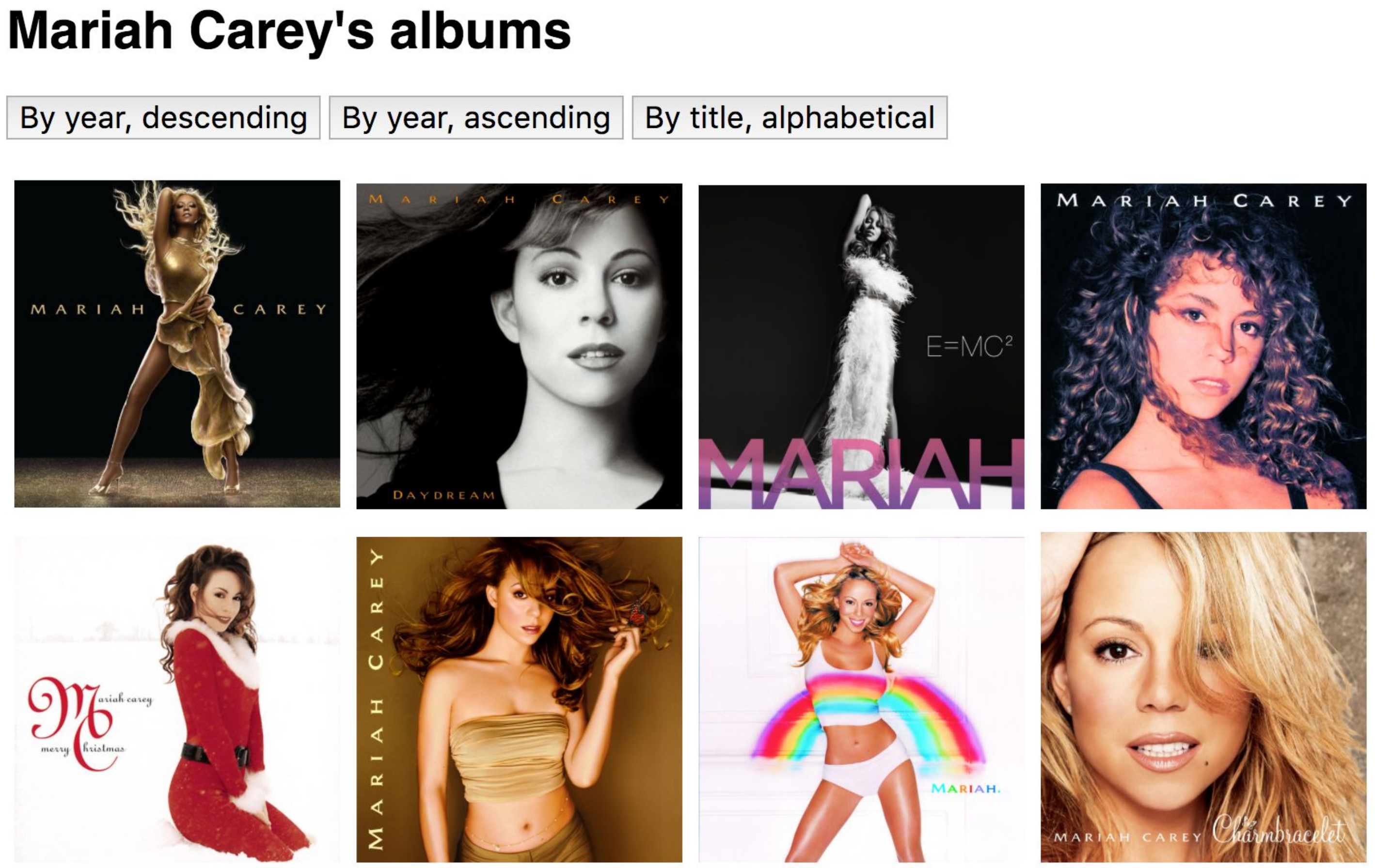


```
class App {
  loadAlbums() {
    fetch(JSON_PATH)
        .then(this._onResponse)
        .then(this._onJsonReady);
  }

  _onJsonReady(json) {
    const albums = json.albums;
    // Let's print the albums fetched.
    for (const album of albums) {
      console.log(album);
    }
  }

  _onResponse(response) {
    return response.json();
  }
}
```

```
class App {
  loadAlbums() {
    fetch(JSON_PATH)
        .then(this._onResponse)
        .then(this._onJsonReady);
  }

  _onJsonReady(json) {
    const albums = json.albums;
    const albumContainer = document.querySelector('#album-container');
    for (const info of albums) {
      const album = new Album(albumContainer, info.url);
    }
  }

  _onResponse(response) {
    return response.json();
  }
}
```

Milestone 3: Create albums

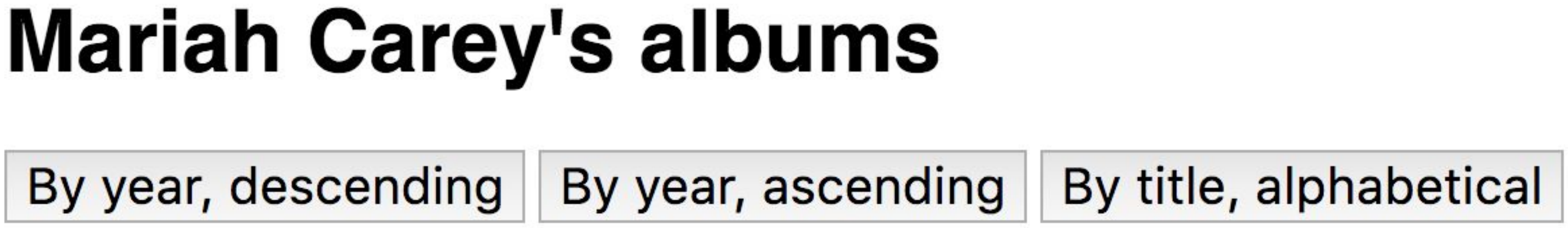
Milestone 3: CodePen / page


Mariah Carey's albums

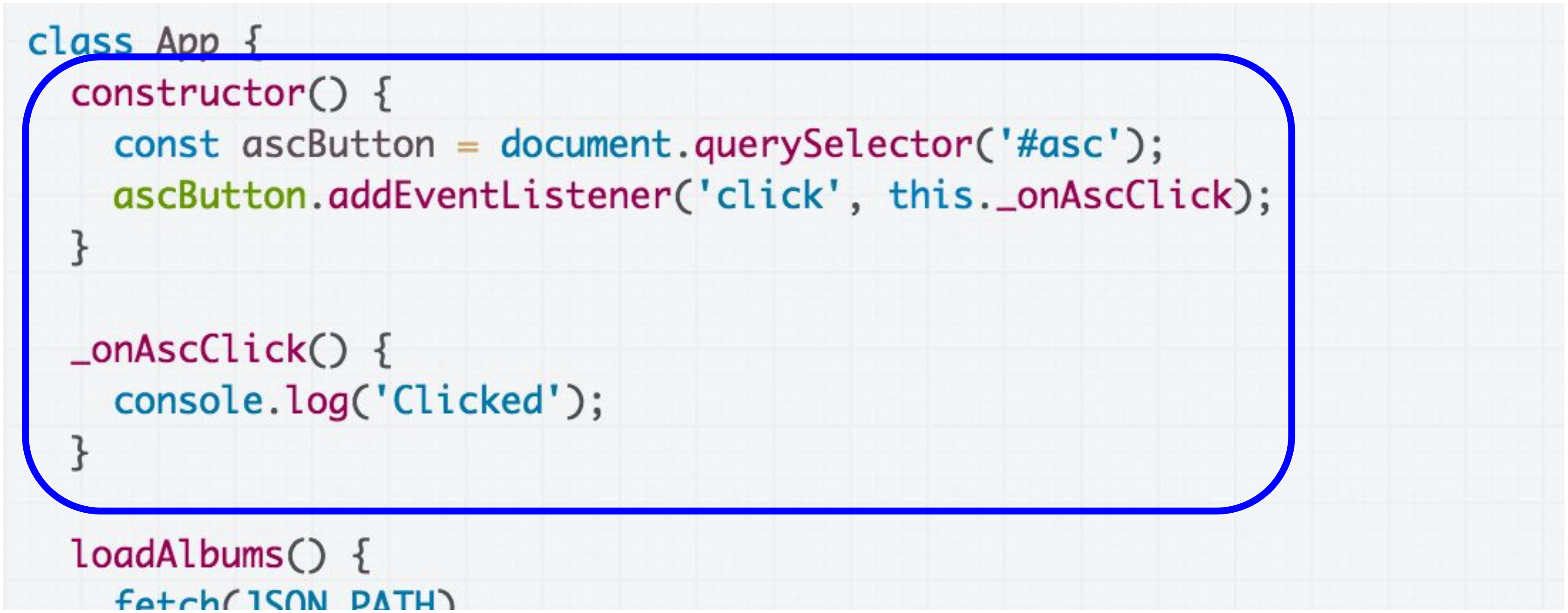
Milestone 4: Sort by year, asc

Let's now implement the Sort by Year, Ascending:

- On button click:
  - Print to console
  - Unrender albums
  - Sort albums data
  - Rereunder albums



Milestone 4: Sort by year, asc

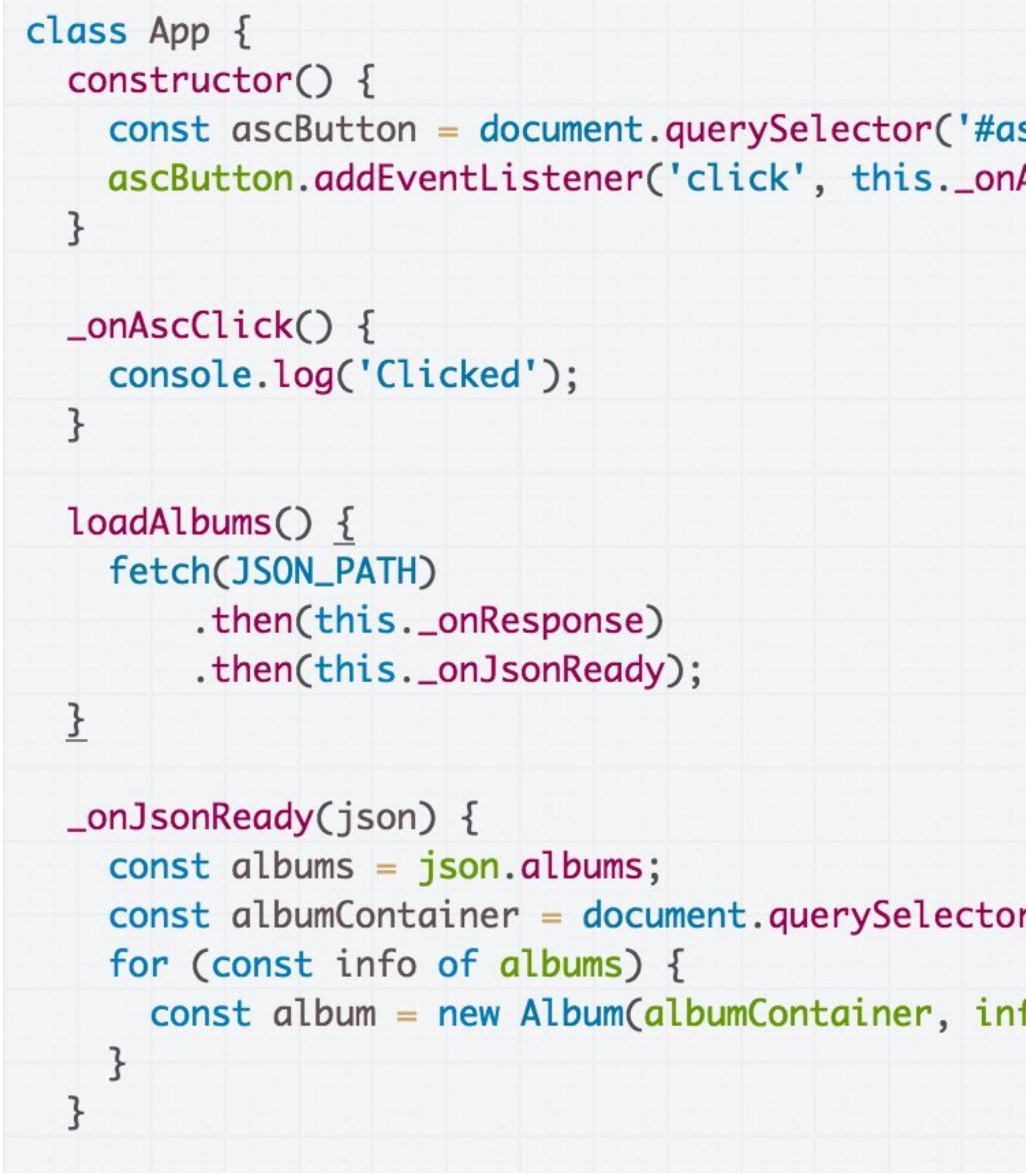


```
class App {
  constructor() {
    const ascButton = document.querySelector('#asc');
    ascButton.addEventListener('click', this._onAscClick);
  }

  _onAscClick() {
    console.log('Clicked');
  }

  loadAlbums() {
    fetch(JSON_PATH)
```

Start with adding an event handler and log to make sure it works: [CodePen]()

Now we want to:

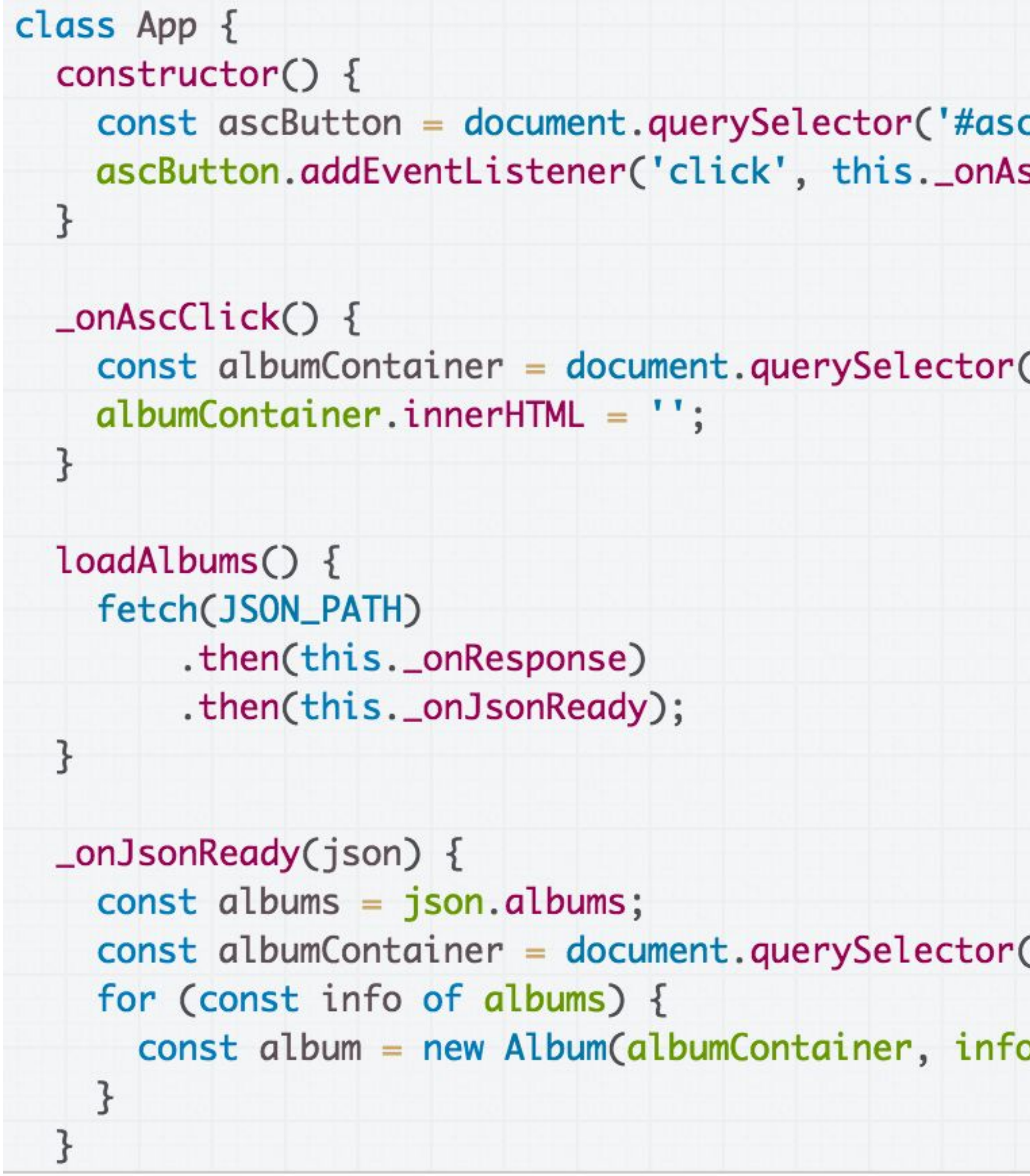
- Unrender the albums



```
class App {
  constructor() {
    const ascButton = document.querySelector('#as
    ascButton.addEventListener('click', this._onA
  }

  _onAscClick() {
    console.log('Clicked');
  }

  loadAlbums() {
    fetch(JSON_PATH)
        .then(this._onResponse)
        .then(this._onJsonReady);
  }

  _onJsonReady(json) {
    const albums = json.albums;
    const albumContainer = document.querySelector
    for (const info of albums) {
      const album = new Album(albumContainer, inf
    }
  }
```

Now we want to:

- Unrender the albums (CodePen)

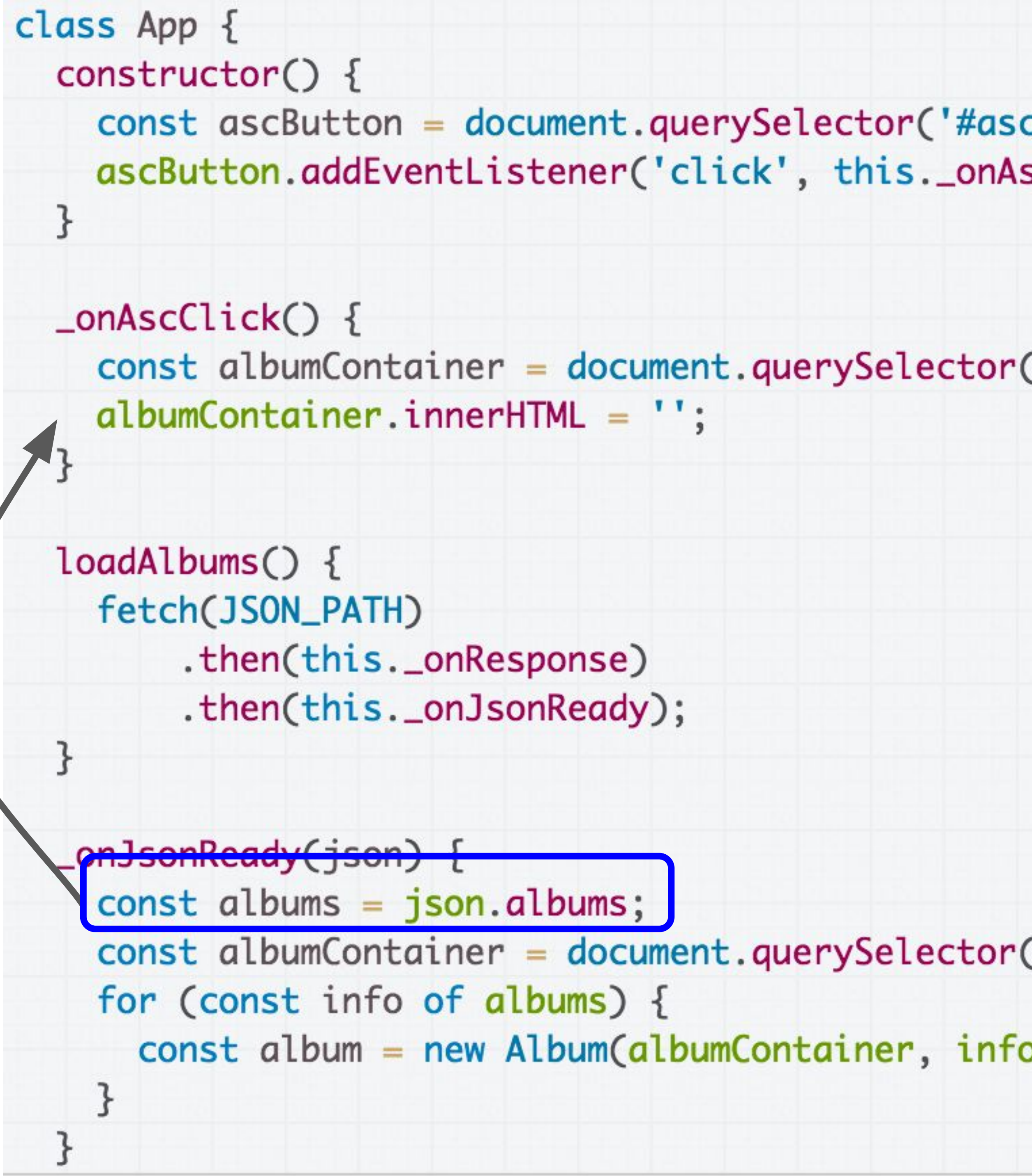


```
class App {
  constructor() {
    const ascButton = document.querySelector('#asc
    ascButton.addEventListener('click', this._onAs
  }

  _onAscClick() {
    const albumContainer = document.querySelector(
    albumContainer.innerHTML = '';
  }

  loadAlbums() {
    fetch(JSON_PATH)
        .then(this._onResponse)
        .then(this._onJsonReady);
  }

  _onJsonReady(json) {
    const albums = json.albums;
    const albumContainer = document.querySelector(
    for (const info of albums) {
      const album = new Album(albumContainer, info
    }
  }
```

Now we want to:

- Sort the albums data

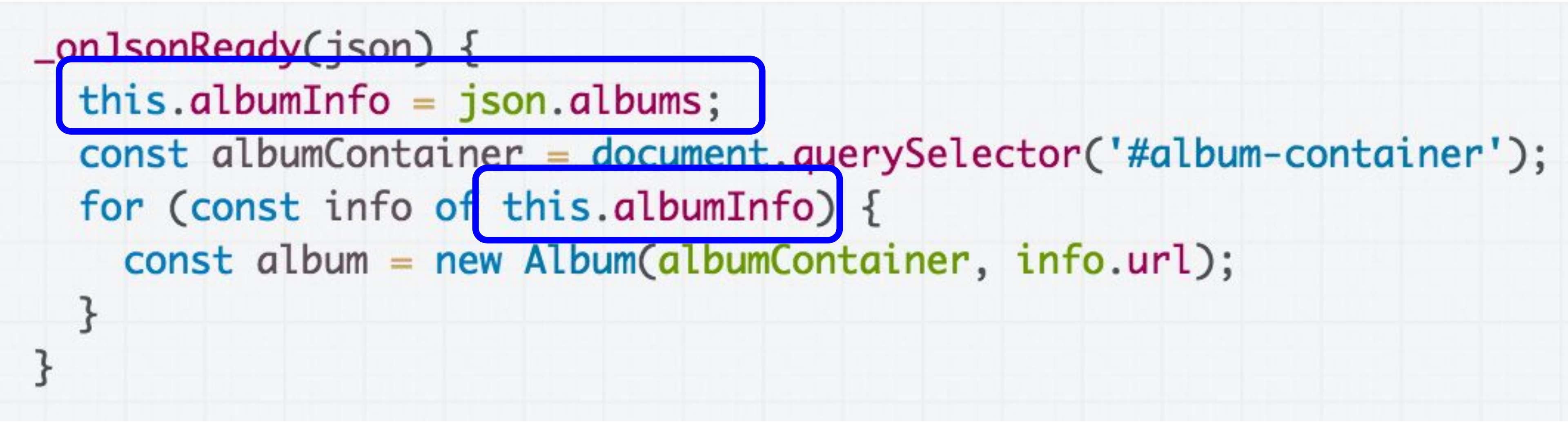
Meaning we need the json.albums from the fetch request available in the onClick



```
class App {
  constructor() {
    const ascButton = document.querySelector('#asc
    ascButton.addEventListener('click', this._onAs
  }

  _onAscClick() {
    const albumContainer = document.querySelector(
    albumContainer.innerHTML = '';
  }

  loadAlbums() {
    fetch(JSON_PATH)
        .then(this._onResponse)
        .then(this._onJsonReady);
  }

  _onJsonReady(json) {
    const albums = json.albums;
    const albumContainer = document.querySelector(
    for (const info of albums) {
      const album = new Album(albumContainer, info
    }
  }
```

Saving data from `fetch()`

We can save the data from the `fetch()` command in a field of the App class (CodePen):

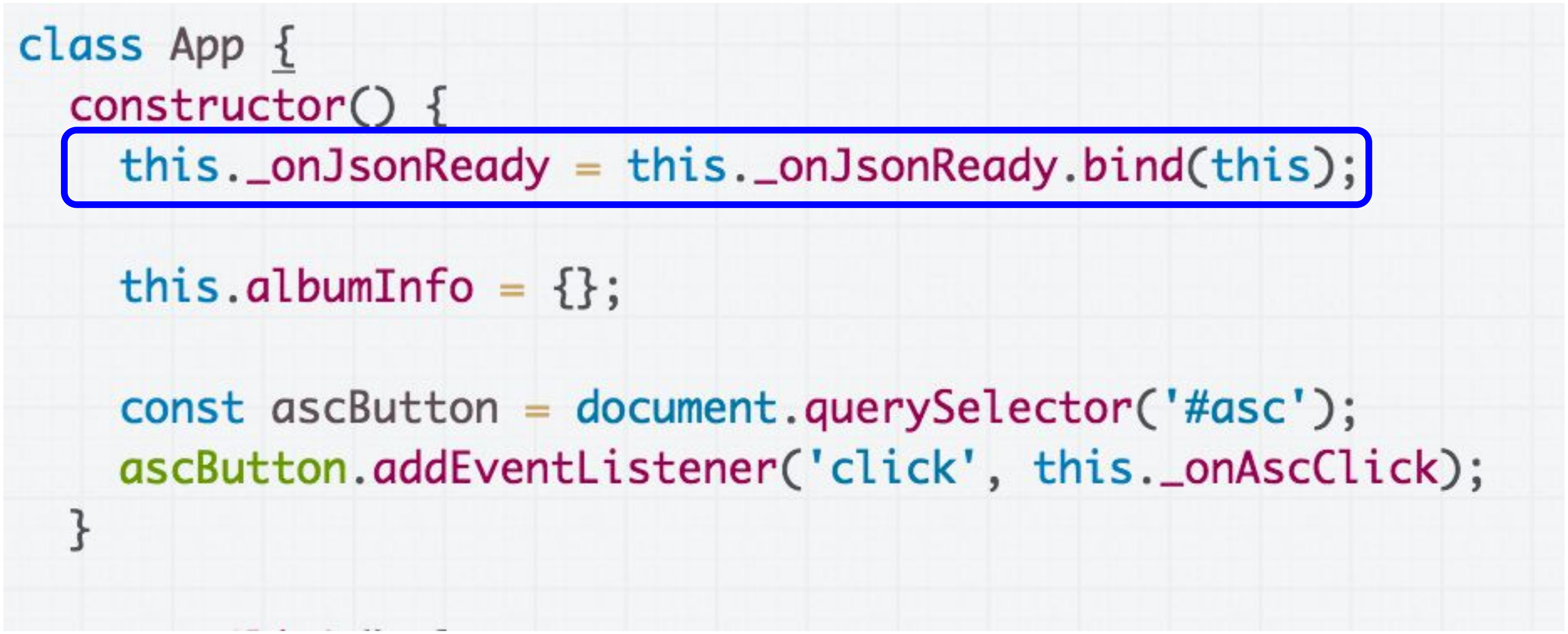


```
_onJsonReady(json) {
  this.albumInfo = json.albums;
  const albumContainer = document.querySelector('#album-container');
  for (const info of this.albumInfo) {
    const album = new Album(albumContainer, info.url);
  }
}
```

But now we are using `this` in a callback… so…
What do we need to do?

Saving data from `fetch()`

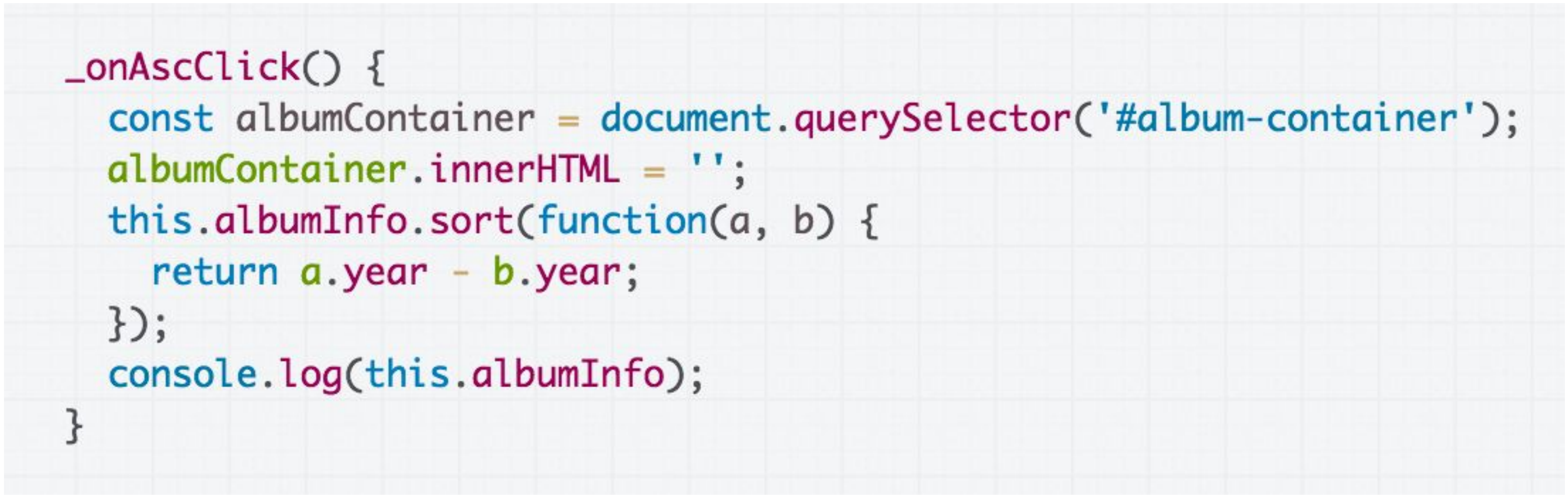
We need to bind `_onJsonReady` in the constructor:



```
class App {
  constructor() {
    this._onJsonReady = this._onJsonReady.bind(this);

    this.albumInfo = {};

    const ascButton = document.querySelector('#asc');
    ascButton.addEventListener('click', this._onAscClick);
  }
```

Saving data from `fetch()`

We are now going to sort the album info on click (CodePen):

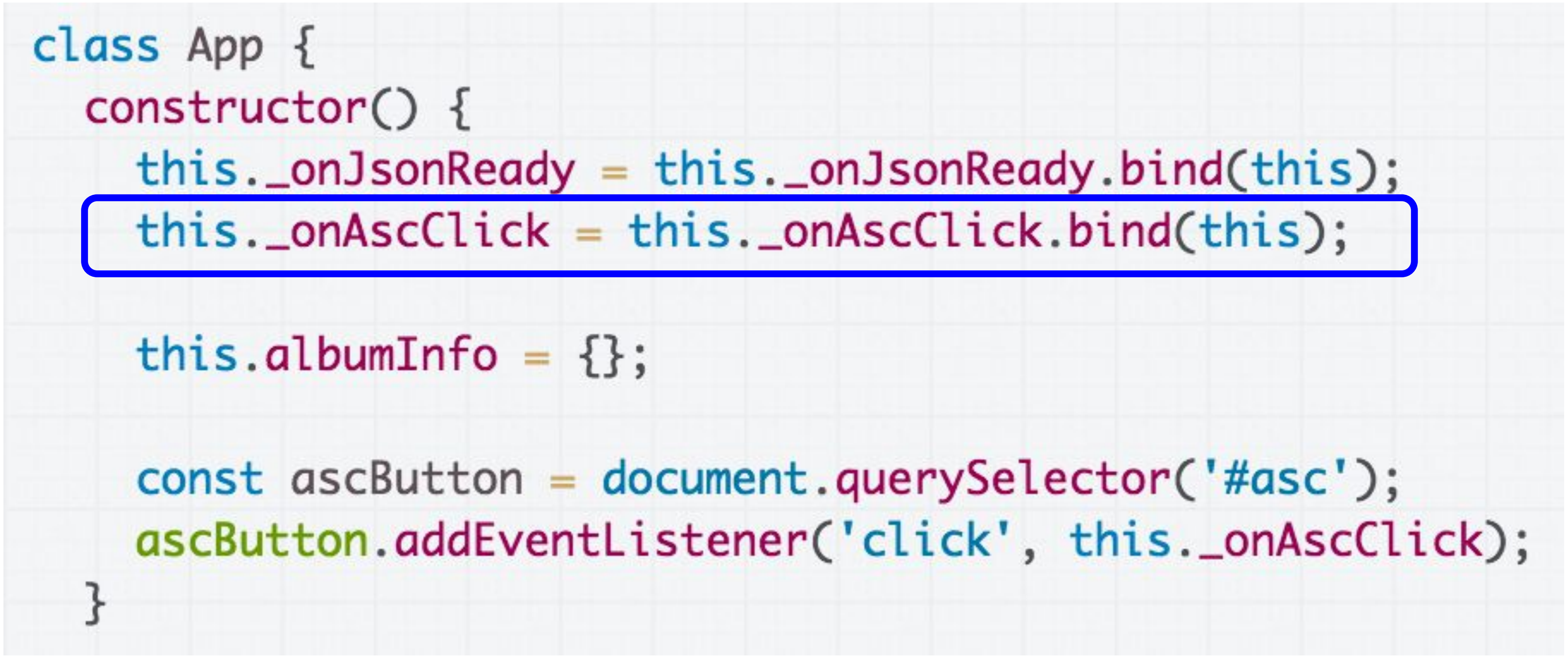


```
_onAscClick() {
  const albumContainer = document.querySelector('#album-container');
  albumContainer.innerHTML = '';
  this.albumInfo.sort(function(a, b) {
    return a.year - b.year;
  });
  console.log(this.albumInfo);
}
```

But now we are using `this` in an event handler… so…
What do we need to do?

Saving data from `fetch()`

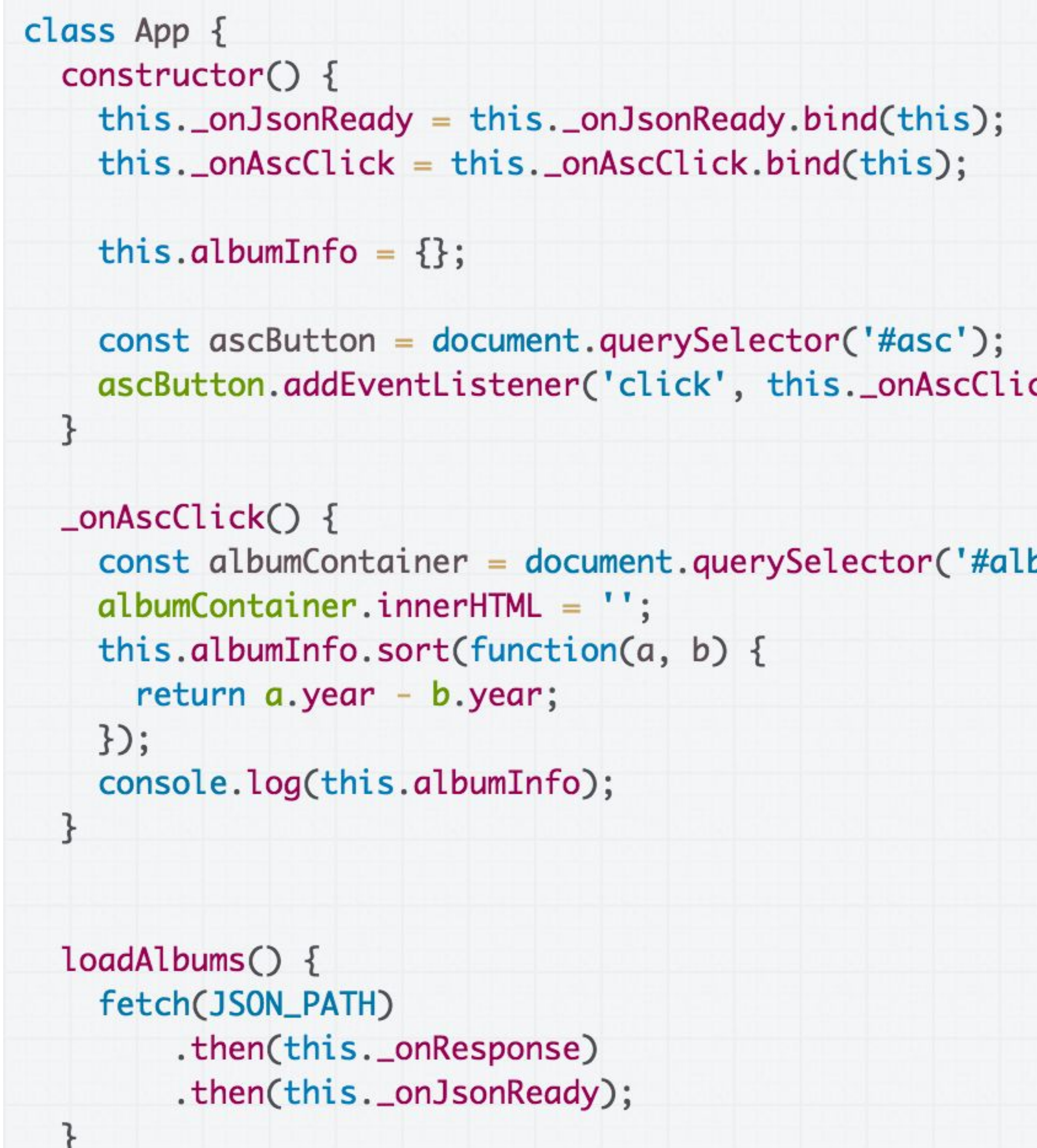
We need to bind `_onAscClick` in the constructor:



```
class App {
  constructor() {
    this._onJsonReady = this._onJsonReady.bind(this);
    this._onAscClick = this._onAscClick.bind(this);

    this.albumInfo = {};

    const ascButton = document.querySelector('#asc');
    ascButton.addEventListener('click', this._onAscClick);
  }
```

Last, we want to:

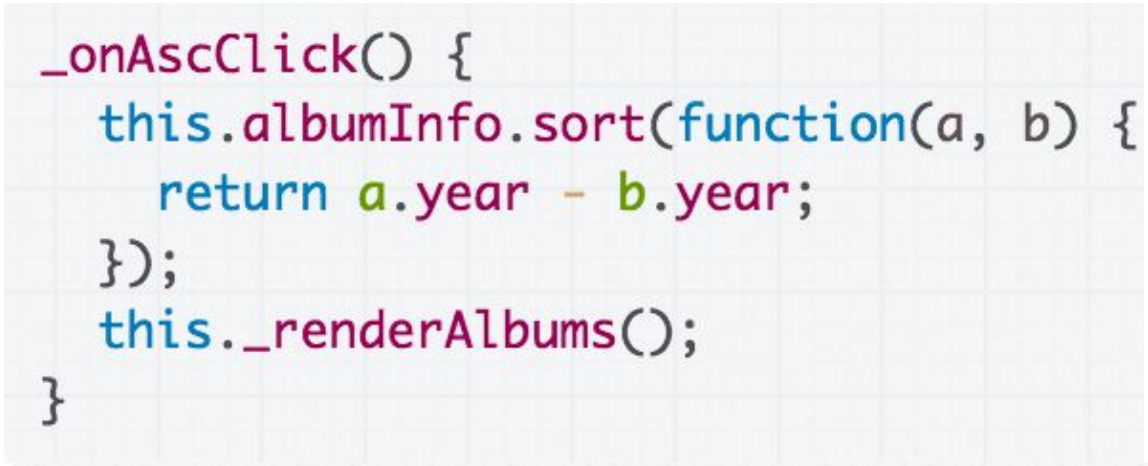
- Rerender the albums data



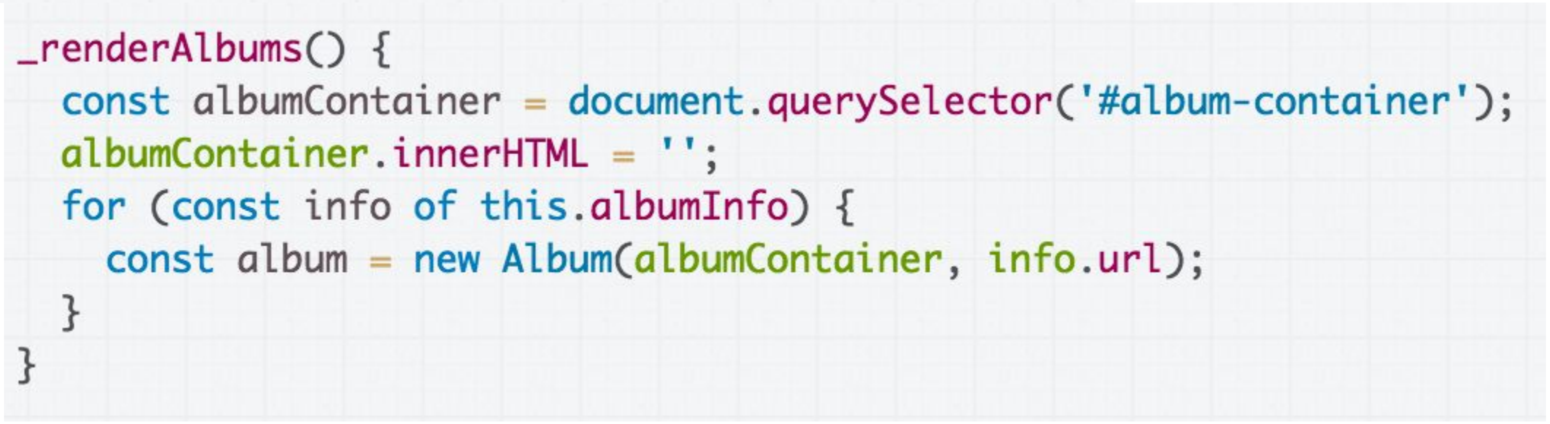
```
class App {
  constructor() {
    this._onJsonReady = this._onJsonReady.bind(this);
    this._onAscClick = this._onAscClick.bind(this);

    this.albumInfo = {};

    const ascButton = document.querySelector('#asc');
    ascButton.addEventListener('click', this._onAscClic
  }

  _onAscClick() {
    const albumContainer = document.querySelector('#alb
    albumContainer.innerHTML = '';
    this.albumInfo.sort(function(a, b) {
      return a.year - b.year;
    });
    console.log(this.albumInfo);
  }


  loadAlbums() {
    fetch(JSON_PATH)
        .then(this._onResponse)
        .then(this._onJsonReady);
  }
```

Rerender albums data

We can put the render code in a helper method and call it: (CodePen)



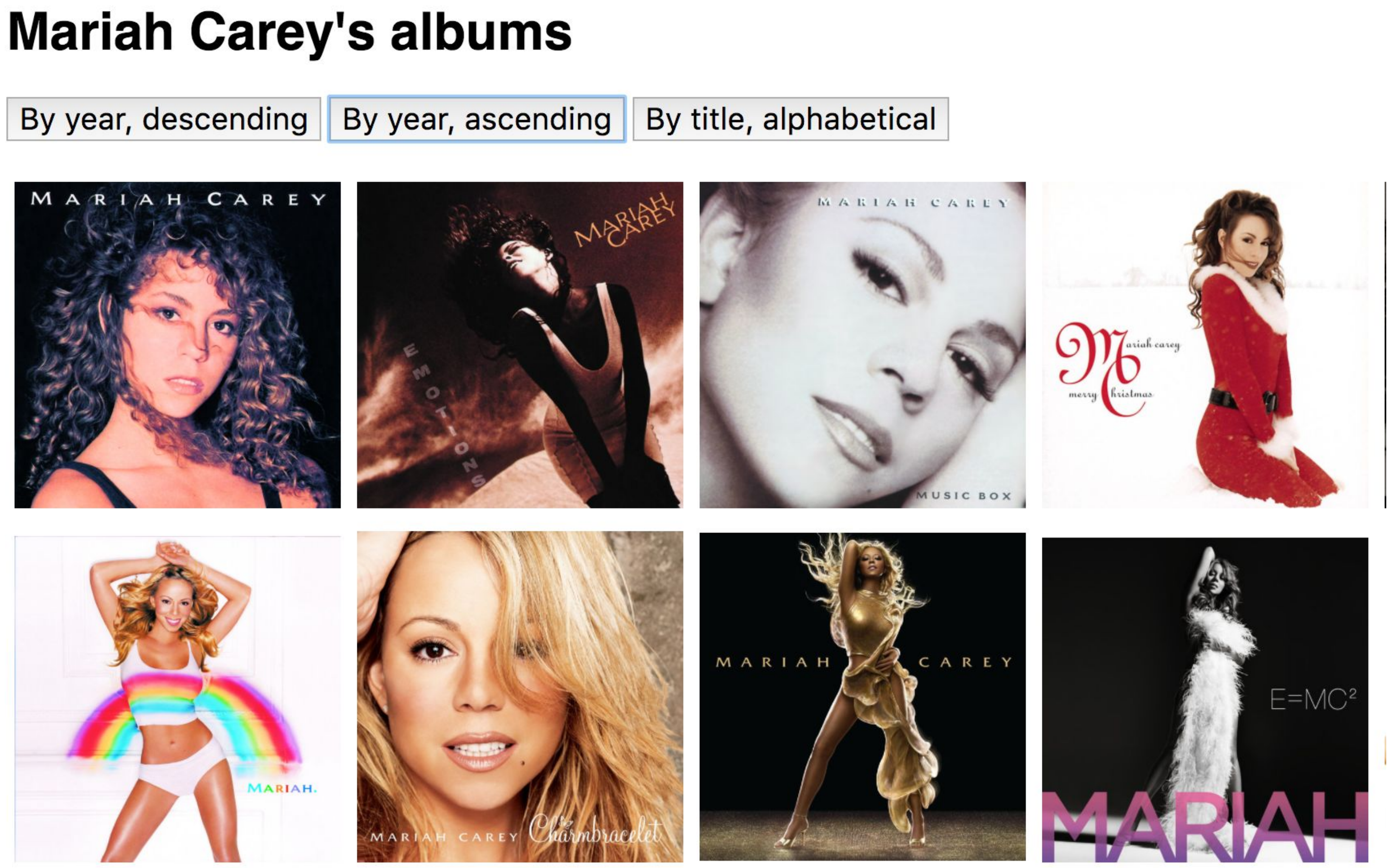
```
_onAscClick() {
  this.albumInfo.sort(function(a, b) {
    return a.year - b.year;
  });
  this._renderAlbums();
}
```



```
_renderAlbums() {
  const albumContainer = document.querySelector('#album-container');
  albumContainer.innerHTML = '';
  for (const info of this.albumInfo) {
    const album = new Album(albumContainer, info.url);
  }
}
```

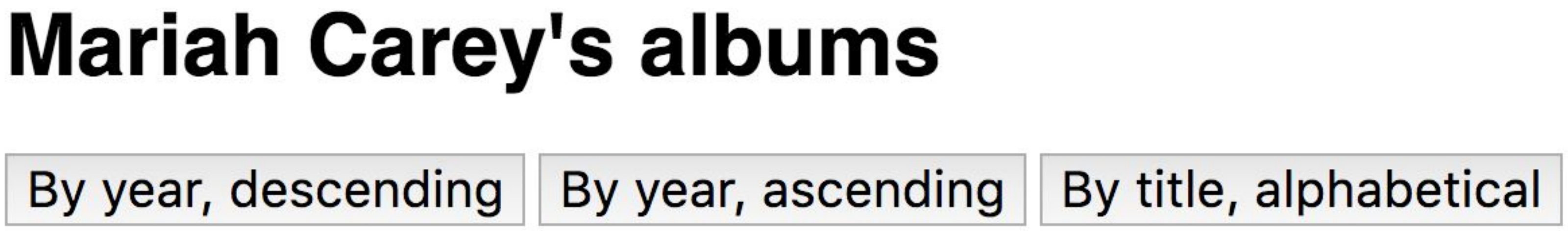
Milestone 4: Sort by year, asc

Milestone 4: [CodePen](#) / [page](#)


Mariah Carey's albums

Milestone 5: Other buttons

Finally, let's implement the other two buttons:

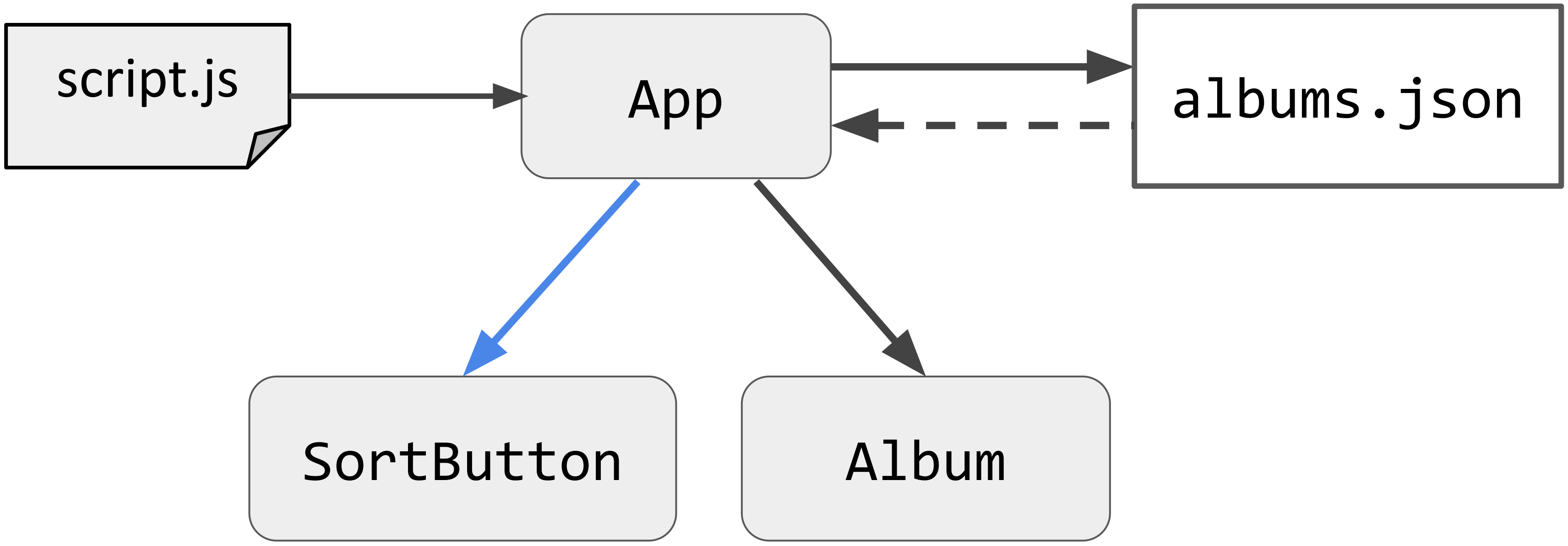

Mariah Carey's albums

Actually, the behavior is almost identical for each button, except the sort function...

Add SortButton class

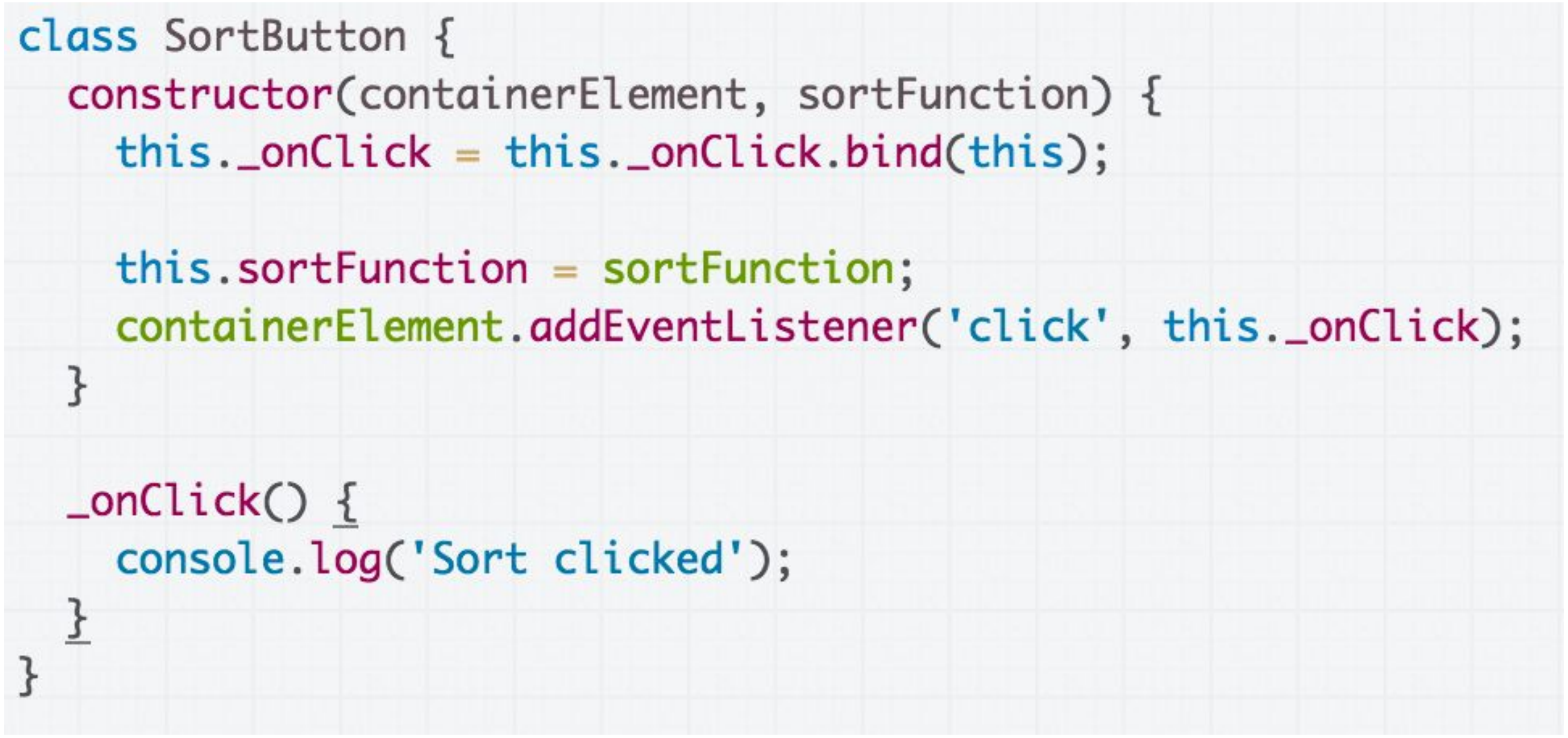
Let's add a SortButton class

- The App class will create 3 `SortButtons`
- Each `SortButton` will take a sorting function as a parameter.



Add SortButton class

We'll add and test the `SortButton` first… CodePen

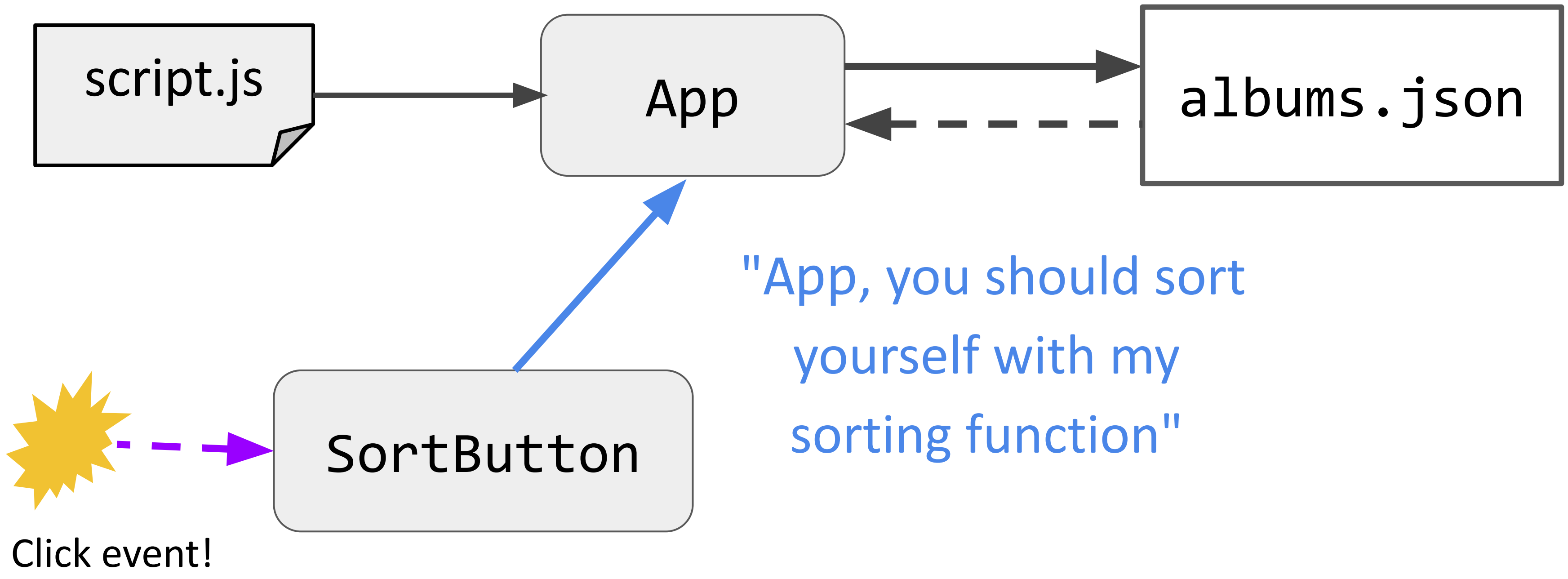


```
class SortButton {
  constructor(containerElement, sortFunction) {
    this._onClick = this._onClick.bind(this);

    this.sortFunction = sortFunction;
    containerElement.addEventListener('click', this._onClick);
  }

  _onClick() {
    console.log('Sort clicked');
  }
}
```

Sorting the albums

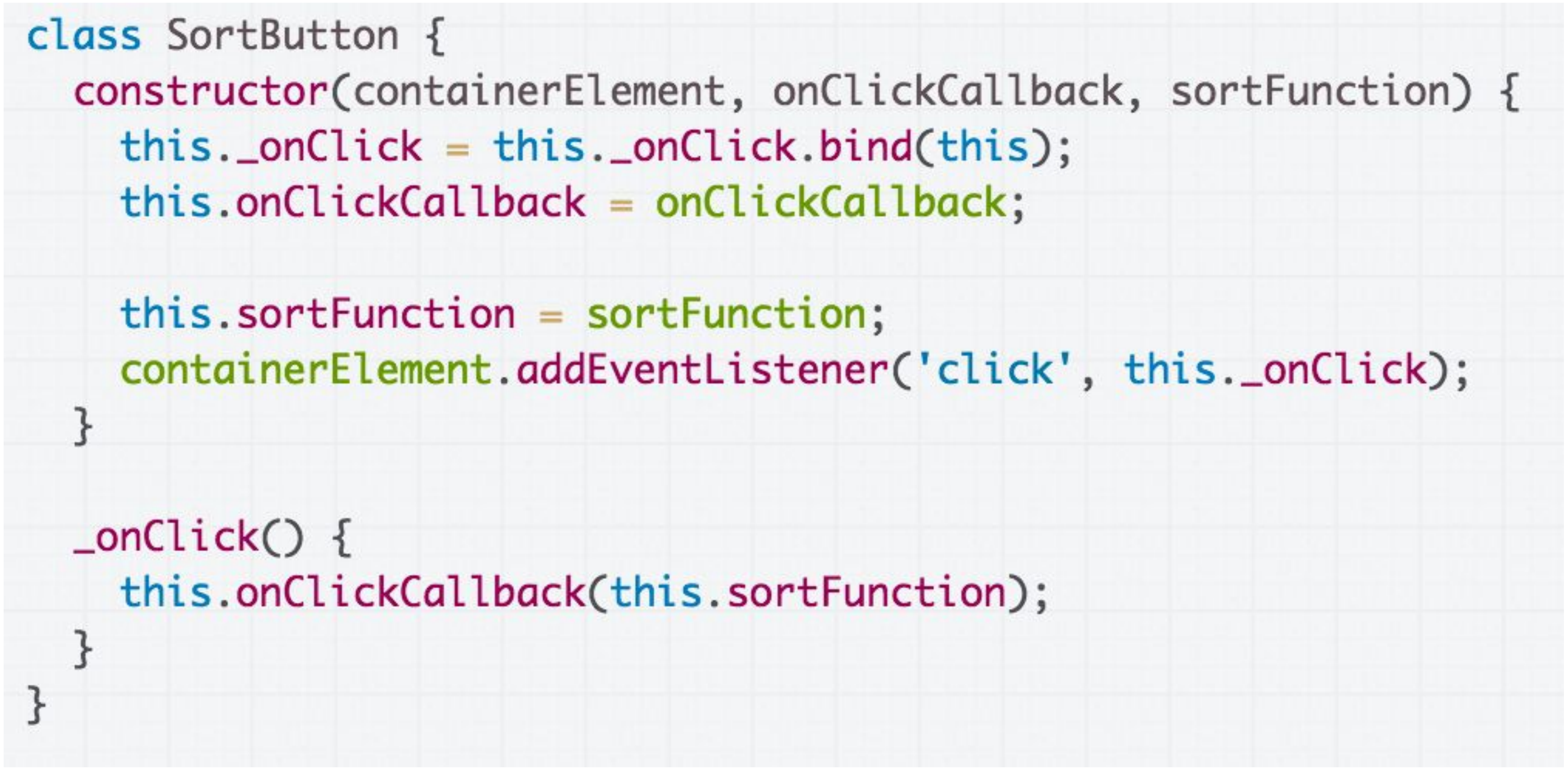
But then when we click a sort button, we want the `Albums` to be sorted… and the `Albums` are in the App class.

- **Q: How do we communicate between `SortButton` and App?**



Sorting the albums

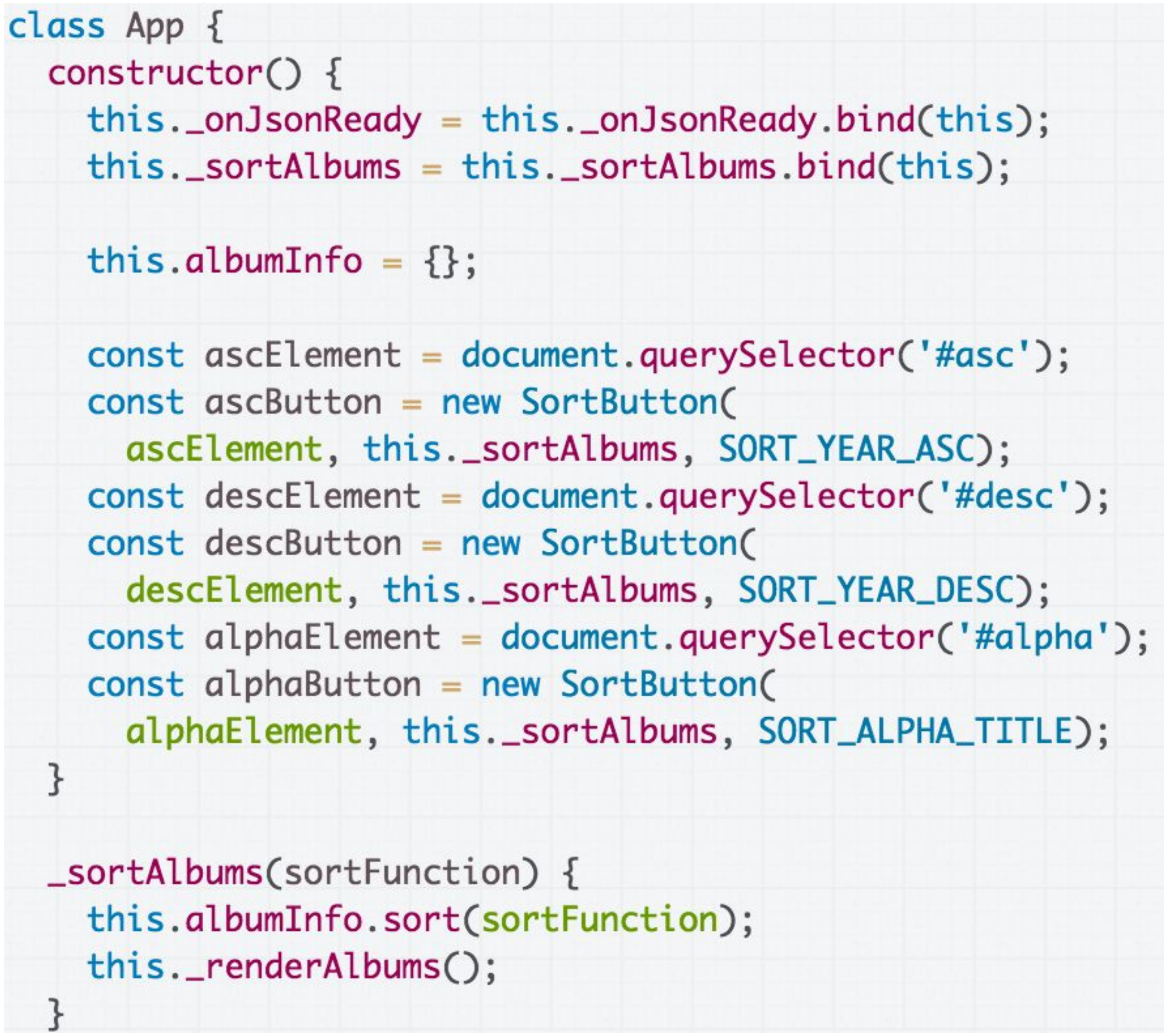
We can add an `onClickCallback` in the `SortButton` constructor (or fire a `CustomEvent`):



```
class SortButton {
  constructor(containerElement, onClickCallback, sortFunction) {
    this._onClick = this._onClick.bind(this);
    this.onClickCallback = onClickCallback;

    this.sortFunction = sortFunction;
    containerElement.addEventListener('click', this._onClick);
  }

  _onClick() {
    this.onClickCallback(this.sortFunction);
  }
}
```

Sorting the albums

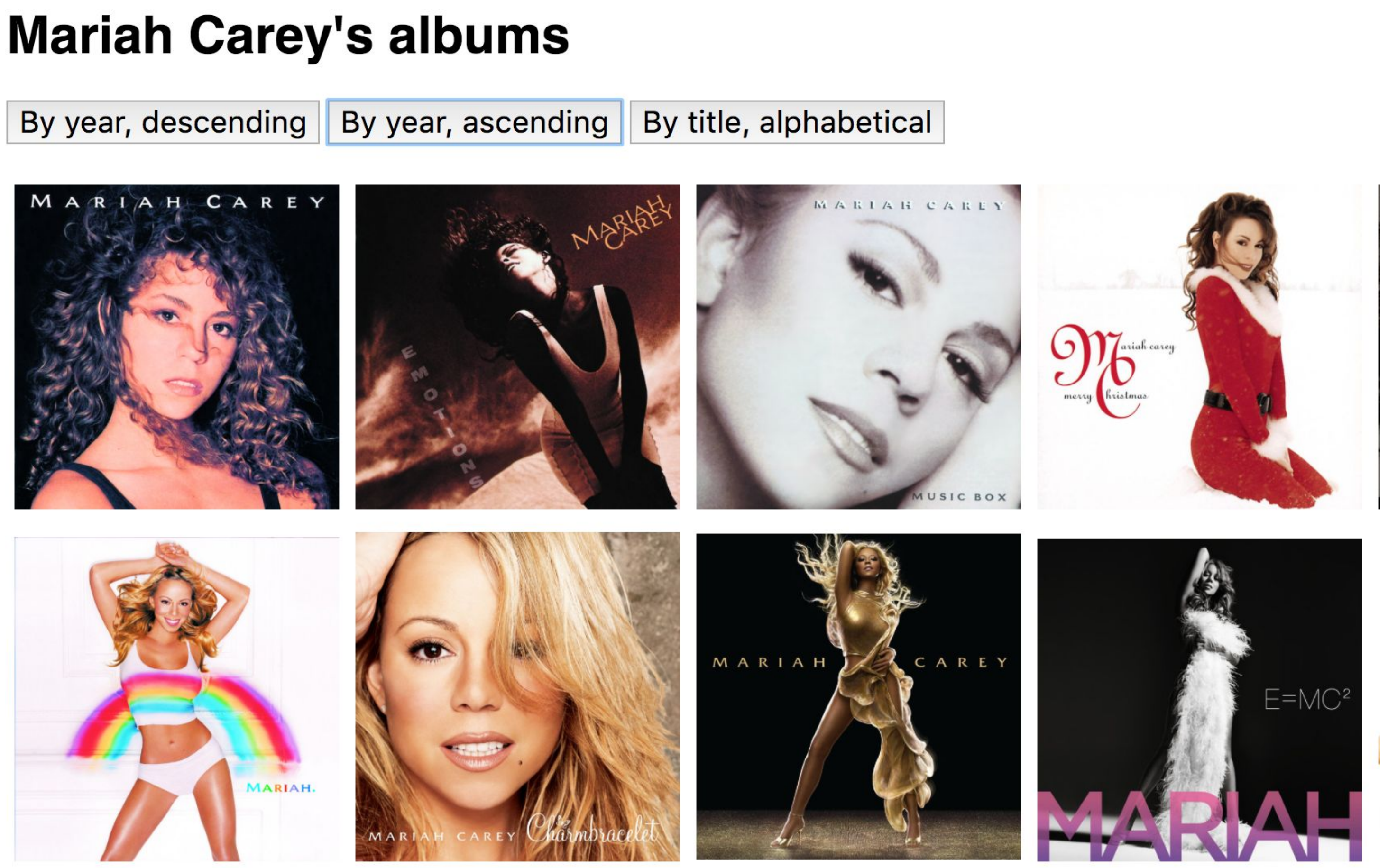
When constructing `SortButton`, pass it the `_sortAlbums` function.



```
class App {
  constructor() {
    this._onJsonReady = this._onJsonReady.bind(this);
    this._sortAlbums = this._sortAlbums.bind(this);

    this.albumInfo = {};

    const ascElement = document.querySelector('#asc');
    const ascButton = new SortButton(
      ascElement, this._sortAlbums, SORT_YEAR_ASC);
    const descElement = document.querySelector('#desc');
    const descButton = new SortButton(
      descElement, this._sortAlbums, SORT_YEAR_DESC);
    const alphaElement = document.querySelector('#alpha');
    const alphaButton = new SortButton(
      alphaElement, this._sortAlbums, SORT_ALPHA_TITLE);
  }

  _sortAlbums(sortFunction) {
    this.albumInfo.sort(sortFunction);
    this._renderAlbums();
  }
```

Milestone 5: Completed!

Milestone 5: [CodePen](#) / [page](#) / [GitHub](#)

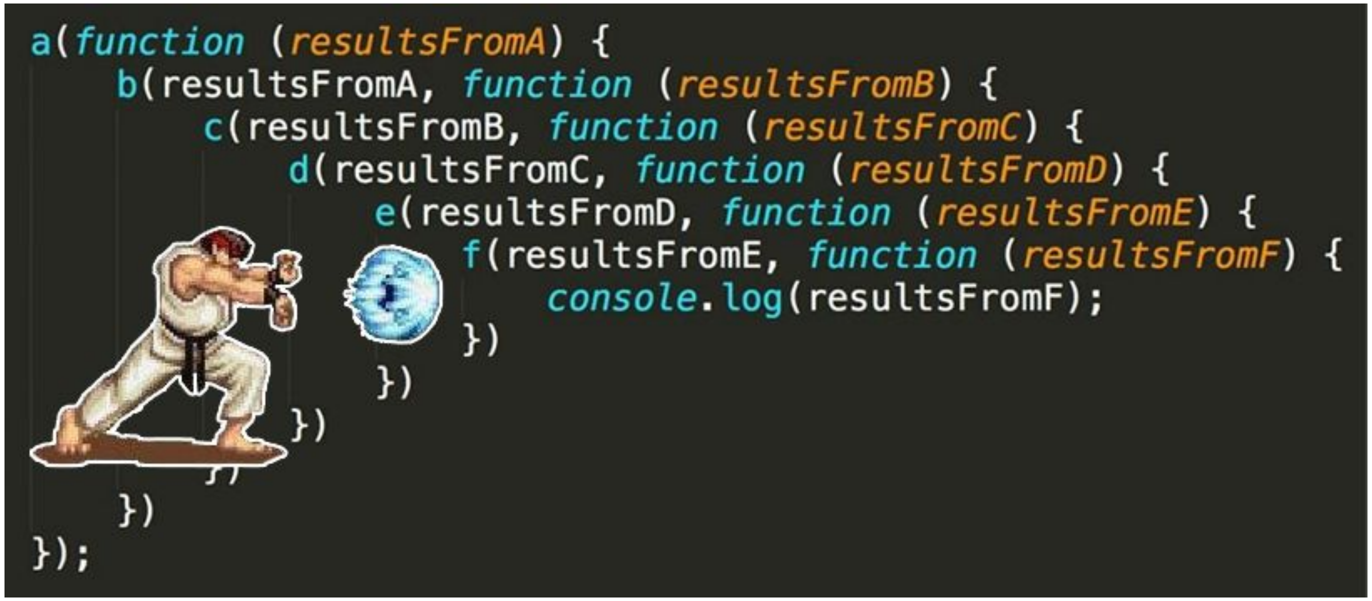

Mariah Carey's albums

Another look on Promises

Promises vs Callbacks

Callbacks:

- The simplest solution to asynchronous tasks
- We can lose control over callbacks that tie together other callbacks (callback hell)



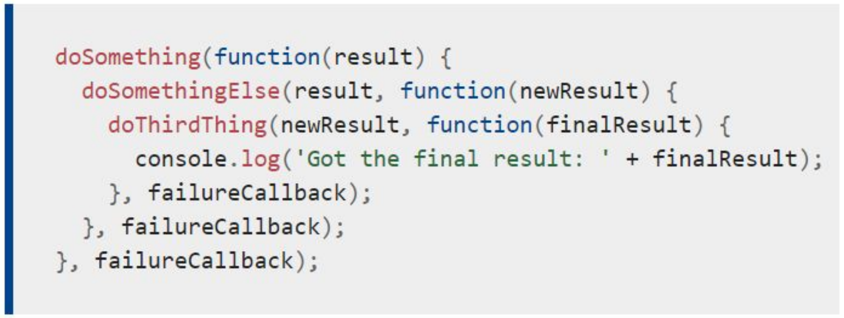
```
a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});
```

Promises vs Callbacks

Promises:

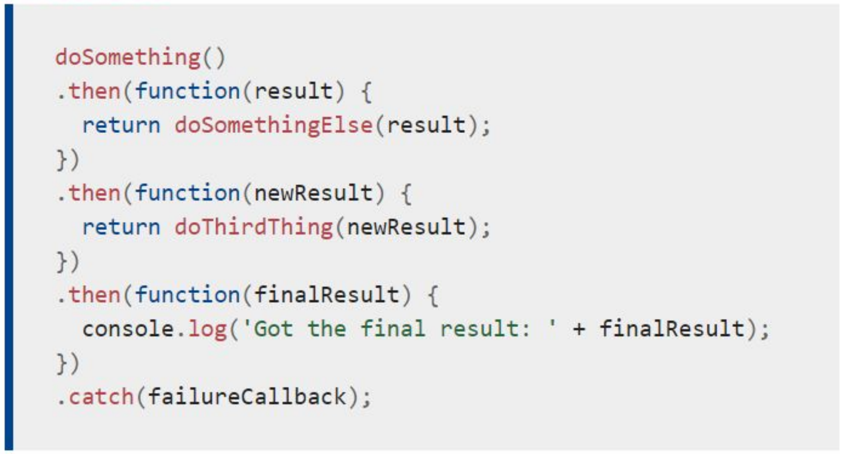
- A Promise is an object representing the **eventual completion or failure** of an asynchronous operation.
- It has three well defined states:
  - Pending
  - Fulfilled
  - Rejected
- They can be easily chained avoiding the callback hell.

Callbacks



```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Promises



```
doSomething()
.then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

Promises API

- How to create new Promise objects

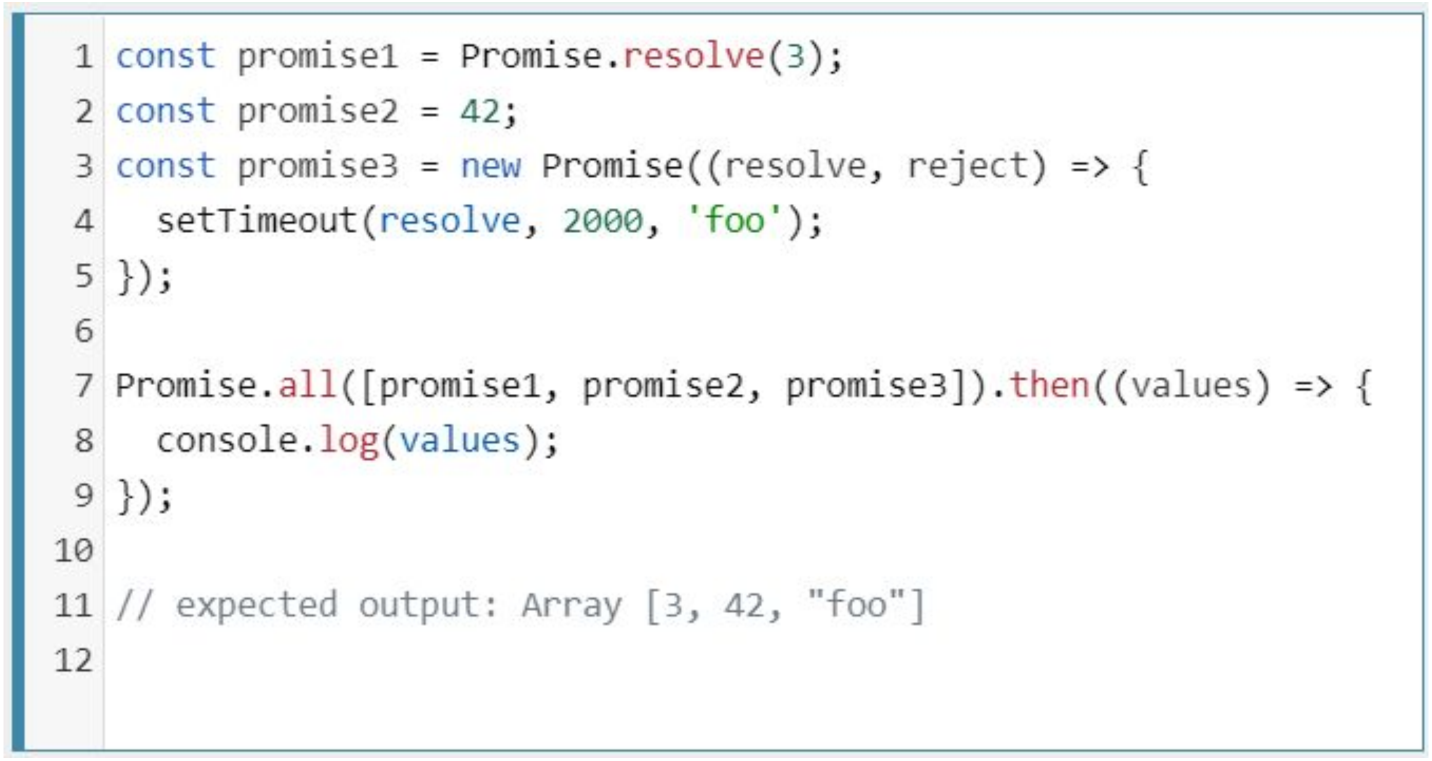
```
// New Promises start in “Pending” state
const myPromise = new Promise(function (resolve, reject) {
   // Transition to “Rejected” state
   reject(new Error(‘A meaningful error’));

   // Transition to “Fulfilled” state
   resolve({ my: ‘data’ });
});
```

Promises API: example

```
const sendMail = function(toEmail, message) {
    return new Promise(function (resolve, reject) {
        if (toEmail.exists()) {
            try {
                SendEmailAPI(toEmail, message); // Takes time!
                // Transition to “Fulfilled” state
                resolve();
            } catch (err) {
                reject(new Error(err));
            }
        } else {
            // Transition to “Rejected” state
            reject(new Error(‘E-mail not found!’));
        }
    });
}
```

Promises API: Composition

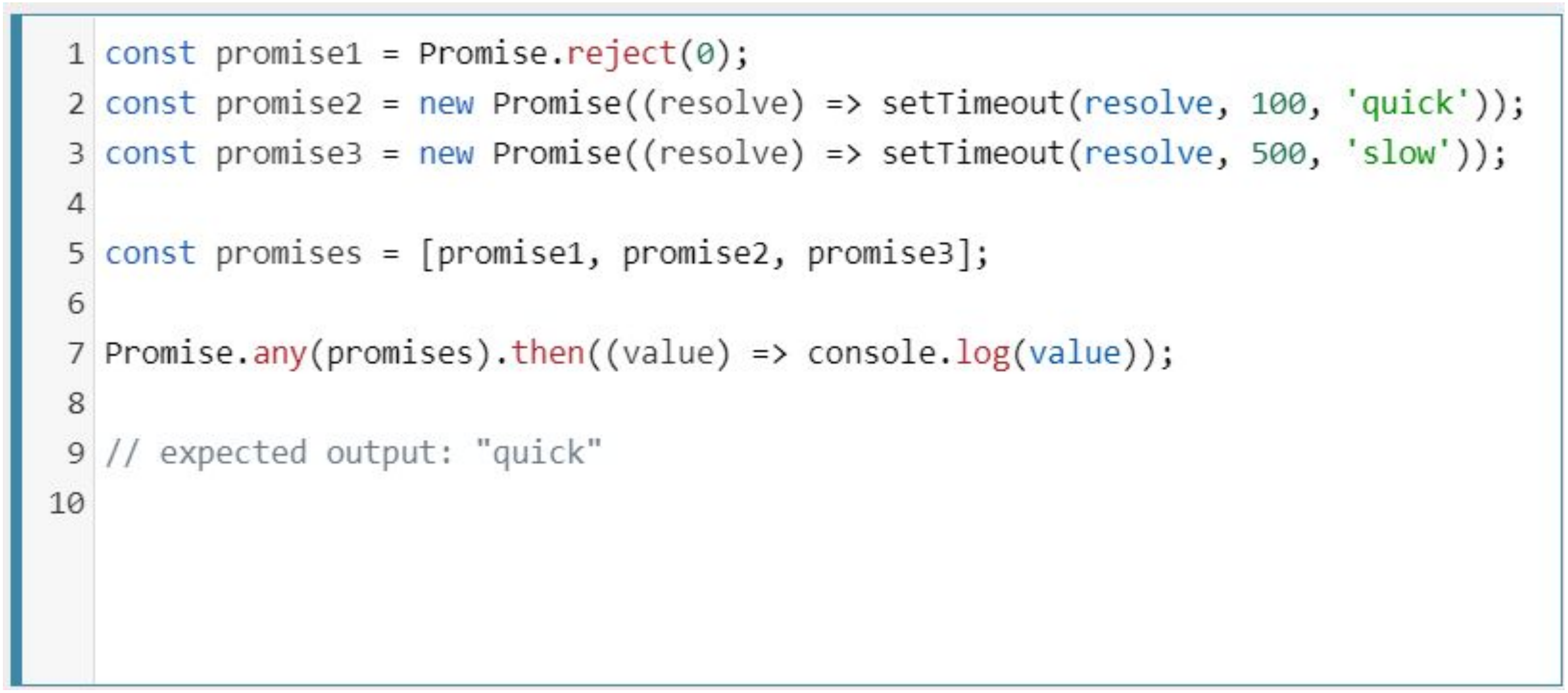
- You can wait for multiple Promises to fulfill using Promise.all

- 

```javascript
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 2000, 'foo');
});

Promise.all([promise1, promise2, promise3]).then((values) => {
  console.log(values);
});

// expected output: Array [3, 42, "foo"]

```

Promises API: Composition

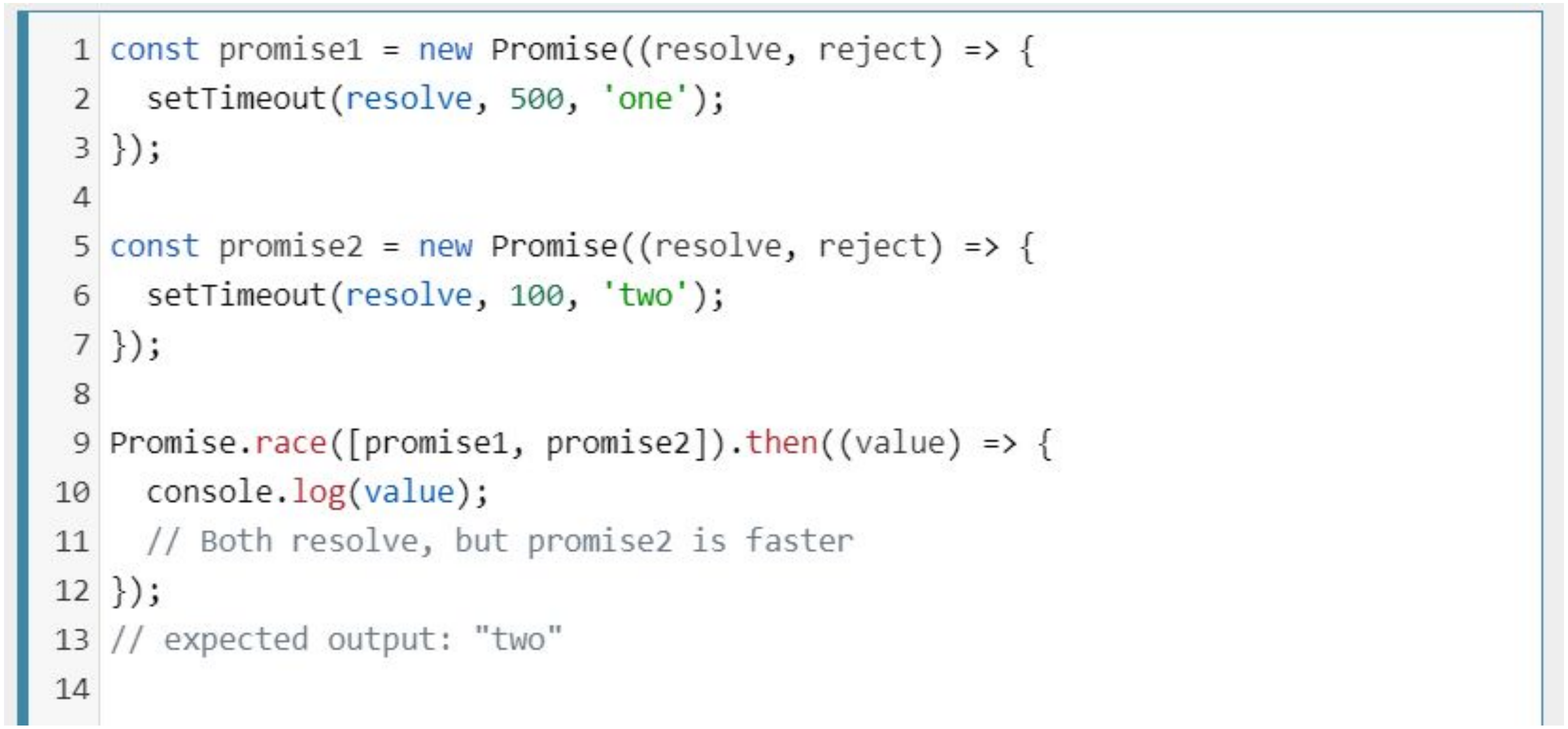
- You can wait for at least one of the Promises to fulfill using Promise.any.



```
const promise1 = Promise.reject(0);
const promise2 = new Promise((resolve) => setTimeout(resolve, 100, 'quick'));
const promise3 = new Promise((resolve) => setTimeout(resolve, 500, 'slow'));

const promises = [promise1, promise2, promise3];

Promise.any(promises).then((value) => console.log(value));

// expected output: "quick"

```

Promises API: Composition

- You can wait for at least one of the Promises to fulfill **or reject** using Promise.race.



```javascript
const promise1 = new Promise((resolve, reject) => {
  setTimeout(resolve, 500, 'one');
});

const promise2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, 'two');
});

Promise.race([promise1, promise2]).then((value) => {
  console.log(value);
  // Both resolve, but promise2 is faster
});
// expected output: "two"

```

Promises API

- **Some references:**
    - Checkout the MDN page about Promises
    - Also these great slides from a Derek Stavis's talk

-

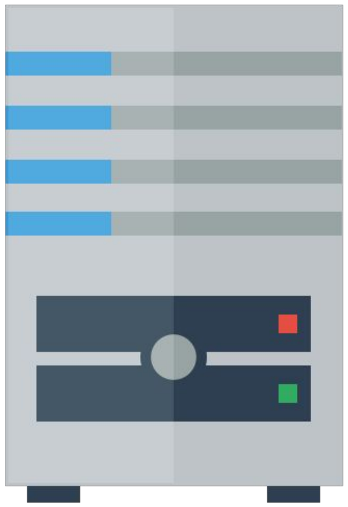
Querying REST APIs

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



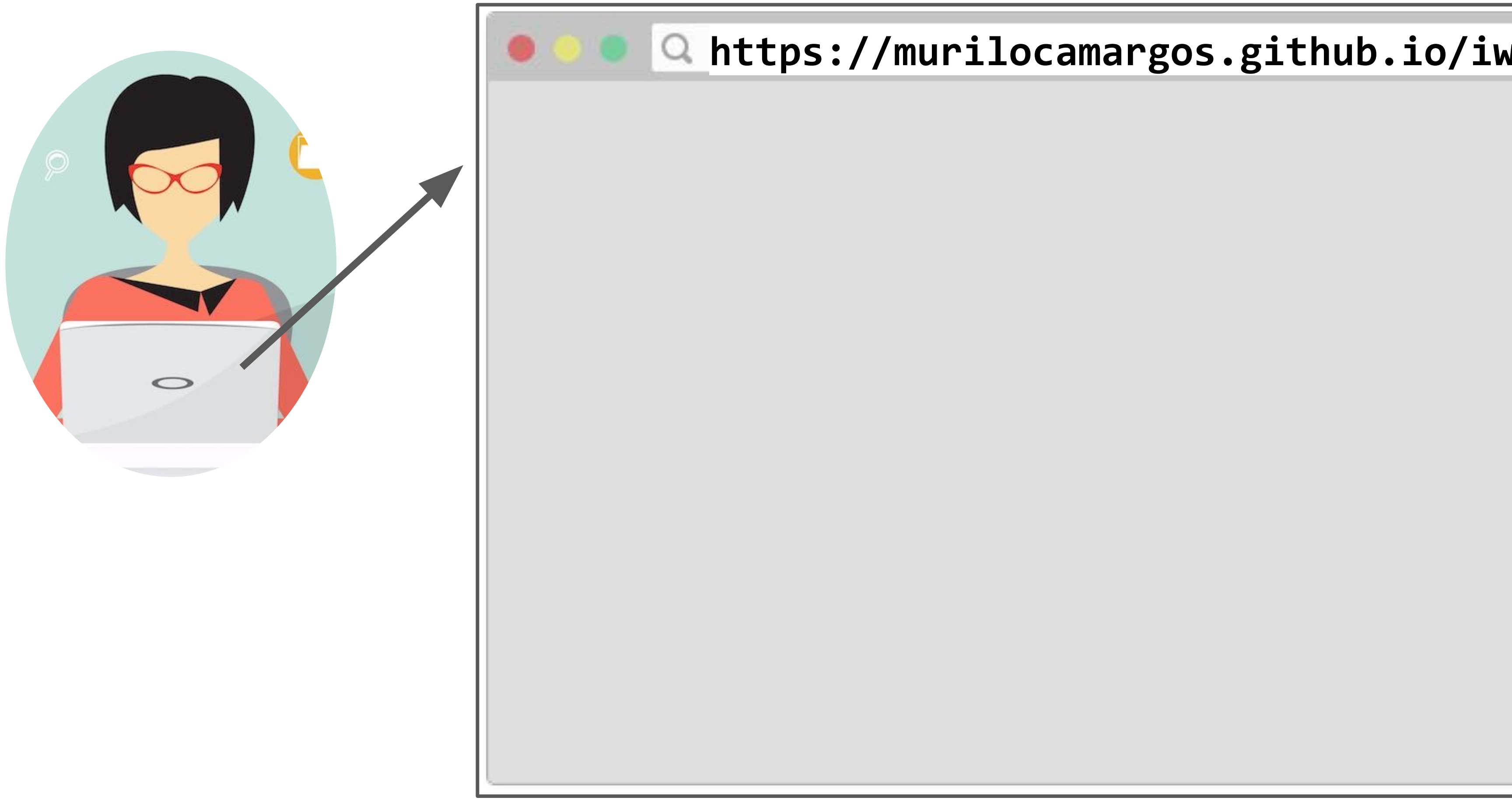
HTTP: Hypertext Transfer Protocol, the protocol for sending files and messages through the web

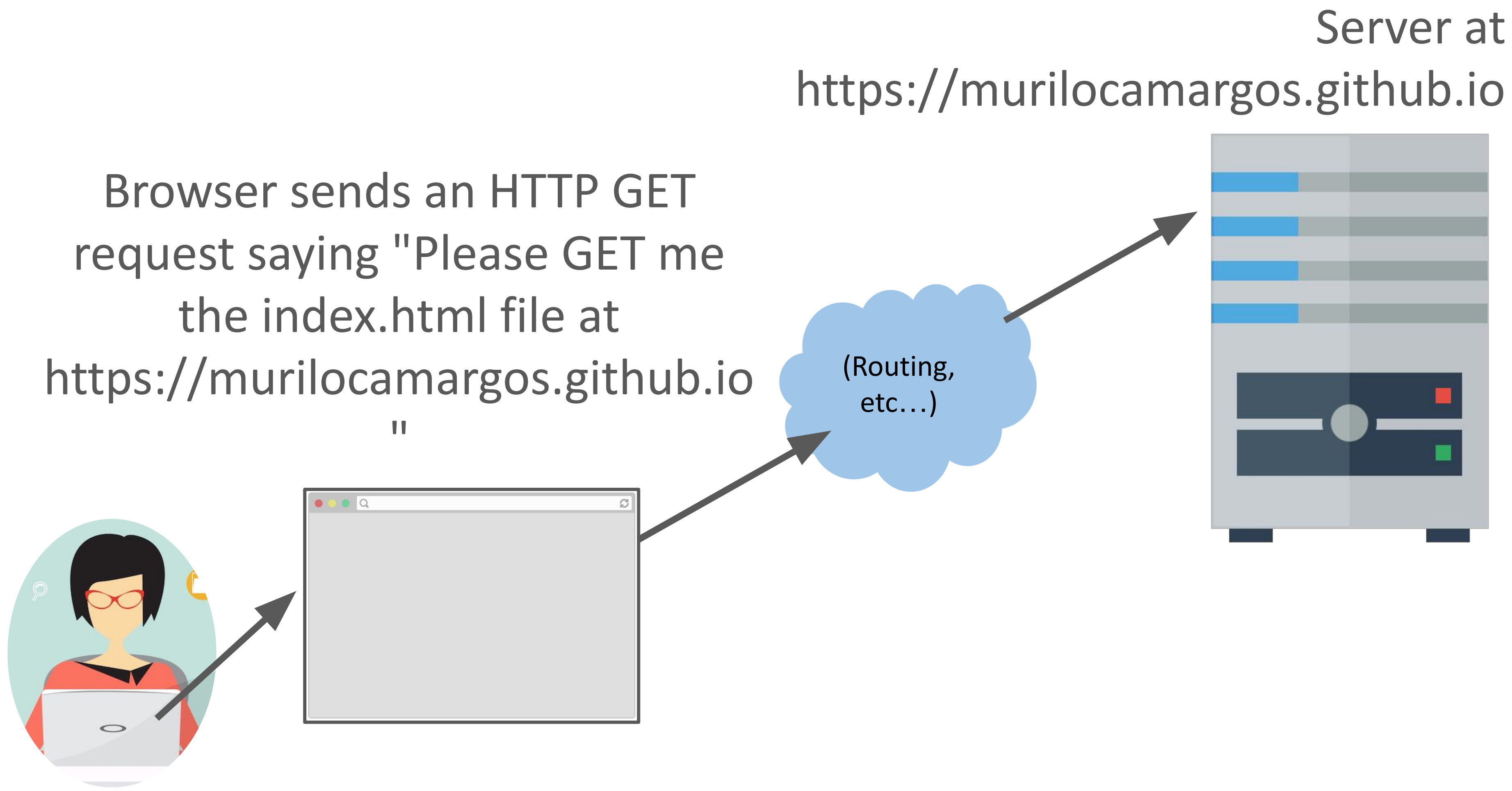
HTTP methods

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"

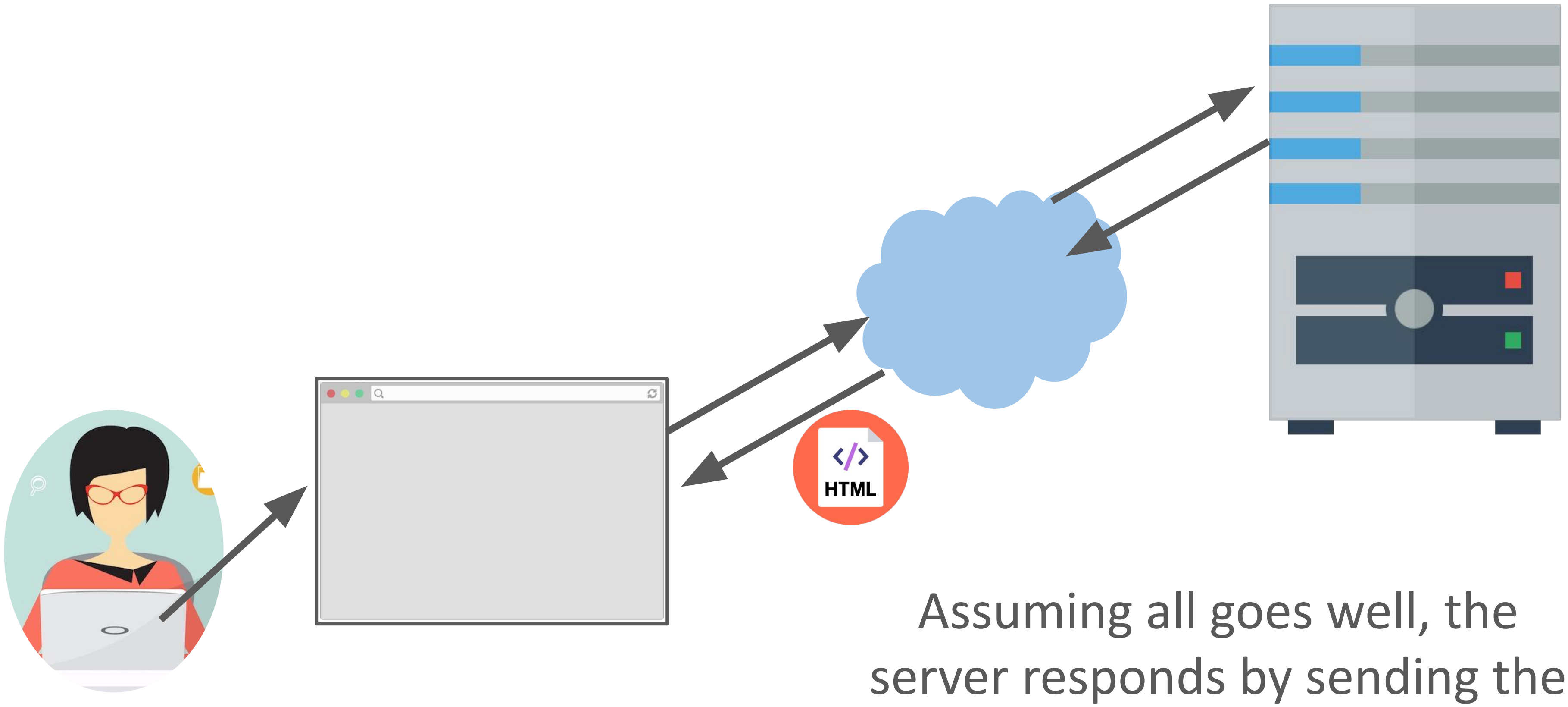



Server at https://murilocamargos.github.io
Browser sends an HTTP GET request saying "Please GET me the index.html file at https://murilocamargos.github.io"
(Routing, etc…)

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

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

Server at
https://murilocamargos.github.io

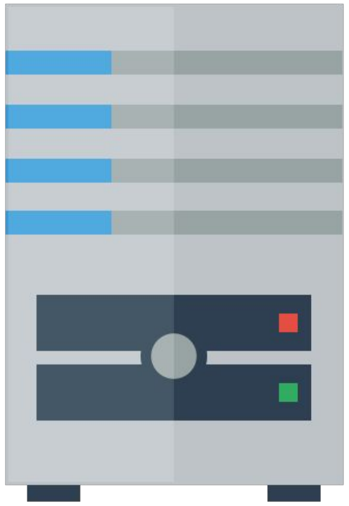


Assuming all goes well, the server responds by sending the HTML file through the internet back to the browser to display.

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



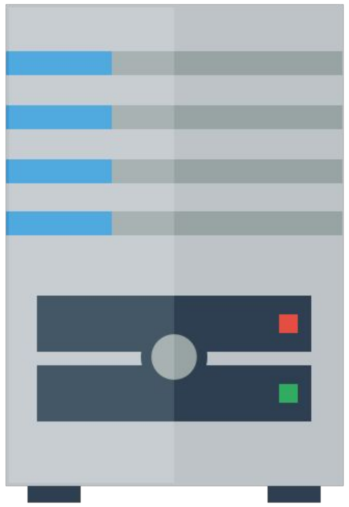
But that's not always the case.

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.



API endpoint example

Look at the URL for this Google slide deck:

```
https://docs.google.com/presentation/d/1WmNb6bbFKP
CqCNwsw4brDg3R-X1hl44wM0lx6O2x7FQ
```

API endpoint example

Look at the URL for this Google slide deck:

`https://docs.google.com/presentation/d/1WmNb6bbFKPCqCNwsw4brDg3R-X1hl44wM0lx6O2x7FQ`

- **`presentation`**: Tells the server that we are requesting a doc of type "presentation"
- **`d/1WmNb6bbFKPCqCNwsw4brDg3R-X1hl44wM0lx6O2x7FQ`**: Tells the server to request a doc ("d") with the document id of "1WmNb6bbFKPCqCNwsw4brDg3R-X1hl44wM0lx6O2x7FQ"

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.

RESTful API

Almost every website on the internet uses RESTful URLs / RESTful APIs to handle requests to its servers.

Notable alternatives to REST:

- GraphQL,
    - Used by Facebook since 2012
    - Open-sourced by Facebook since 2015
    - Still early but some big clients: GitHub, Pinterest
- Falcor?
    - Netflix's REST alternative, introduced ~2015
    - Probably cool but never hear of anyone using it
    - Doesn't even have a Wikipedia page

Using REST APIs

3rd-Party APIs

Many websites expose REST APIs to outside developers.
These are often called "**3rd-party API**s" or "**Developer APIs**"

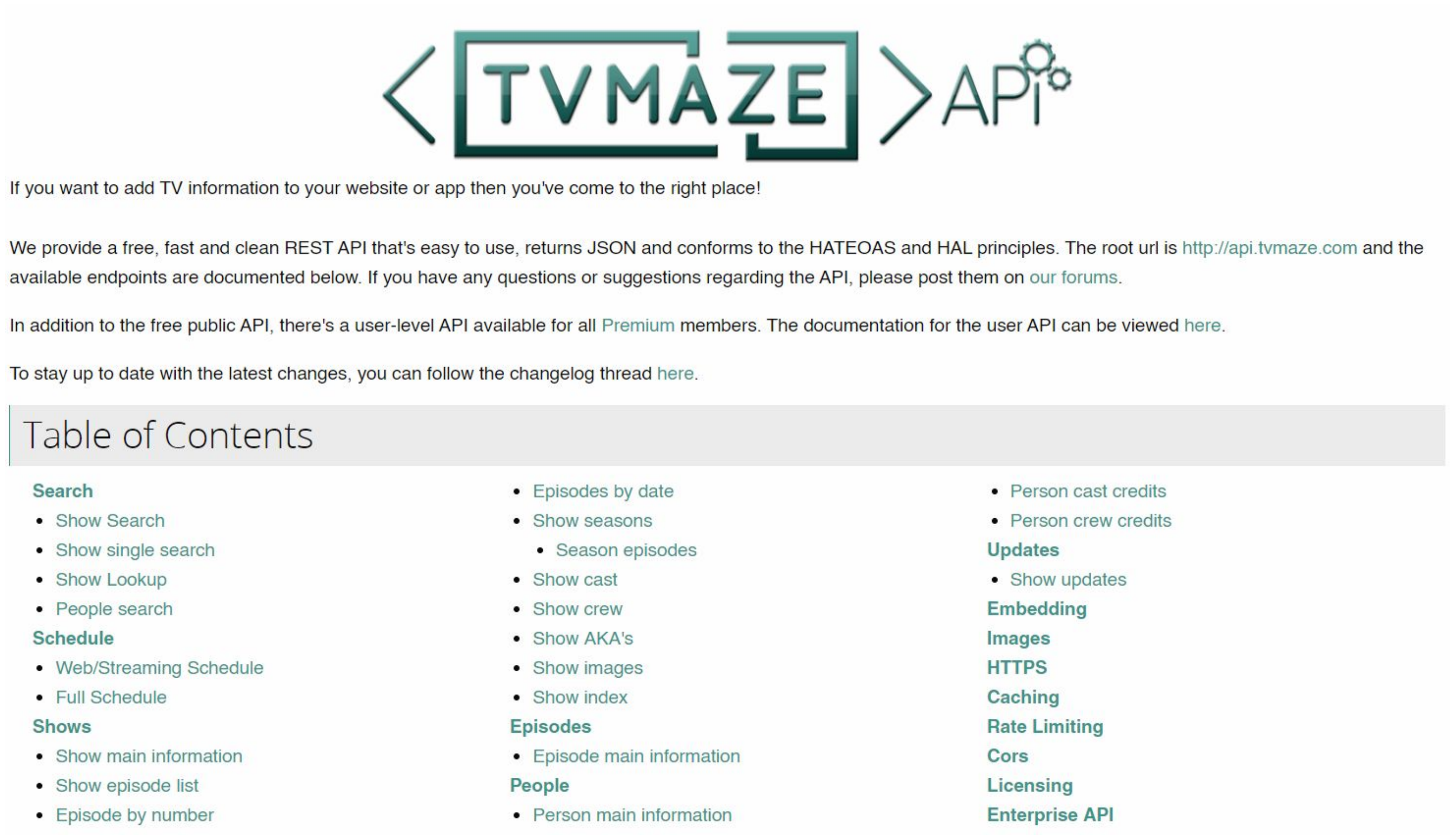
Examples:

- Spotify
- Giphy
- GitHub
- Hoards of Google APIs
- Facebook
- Instagram
- Twitter
- etc...

Try Googling "<product name> API" to see if one exists for a given company!

Example: TVMaze

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



If you want to add TV information to your website or app then you've come to the right place!

We provide a free, fast and clean REST API that's easy to use, returns JSON and conforms to the HATEOAS and HAL principles. The root url is http://api.tvmaze.com and the available endpoints are documented below. If you have any questions or suggestions regarding the API, please post them on our forums.

In addition to the free public API, there's a user-level API available for all Premium members. The documentation for the user API can be viewed here.

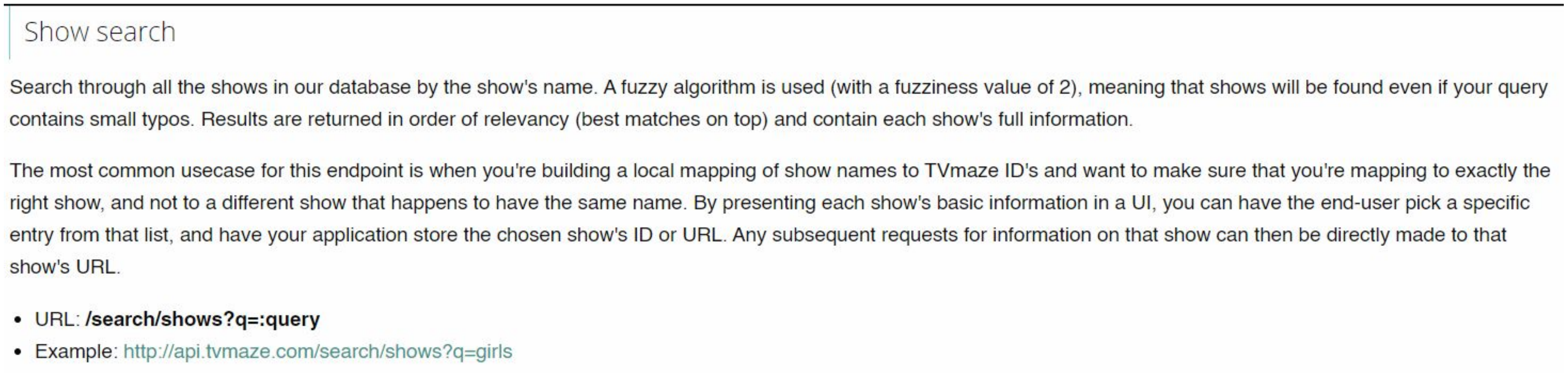
To stay up to date with the latest changes, you can follow the changelog thread here.

Table of Contents

Example: TVMaze

REST API structure (details):

- The **Base URL** is `https://api.tvmaze.com`
- The **HTTP method** is GET
- The **API endpoint** to query is:
  `https://api.tvmaze.com/search/shows?q=`***:query***
- It returns **JSON data** about the album that's requested



Show search

Search through all the shows in our database by the show's name. A fuzzy algorithm is used (with a fuzziness value of 2), meaning that shows will be found even if your query contains small typos. Results are returned in order of relevancy (best matches on top) and contain each show's full information.

The most common usecase for this endpoint is when you're building a local mapping of show names to TVmaze ID's and want to make sure that you're mapping to exactly the right show, and not to a different show that happens to have the same name. By presenting each show's basic information in a UI, you can have the end-user pick a specific entry from that list, and have your application store the chosen show's ID or URL. Any subsequent requests for information on that show can then be directly made to that show's URL.

- URL: **/search/shows?q=:query**
- Example: http://api.tvmaze.com/search/shows?q=girls

Example: TVMaze

If we had a TV Show name “The Witcher”, how would we make a GET request for the album information?

REST API structure (details):

- The **Base URL** is `https://api.tvmaze.com`
- The **HTTP method** is GET
- The **API endpoint** to query is:
  `https://api.tvmaze.com/search/shows?q=`***:query***
- It returns **JSON data** about the album that's requested

GET request: Browse to URL

Loading a URL in a browser issues an HTTP GET request for that resource.

So if we just piece together this URL:

- **API Endpoint:** `https://api.tvmaze.com/search/shows?q=`***`The Witcher`***
- **Query:** `The Witcher`
- **Request URL:** [`https://api.tvmaze.com/search/shows?q=`***`The Witcher`***](https://api.tvmaze.com/search/shows?q=The%20Witcher)

If you click on the link, you see it returns a JSON object.

GET request: `fetch()`

Actually, the `fetch()` API also issues an HTTP GET request by default.

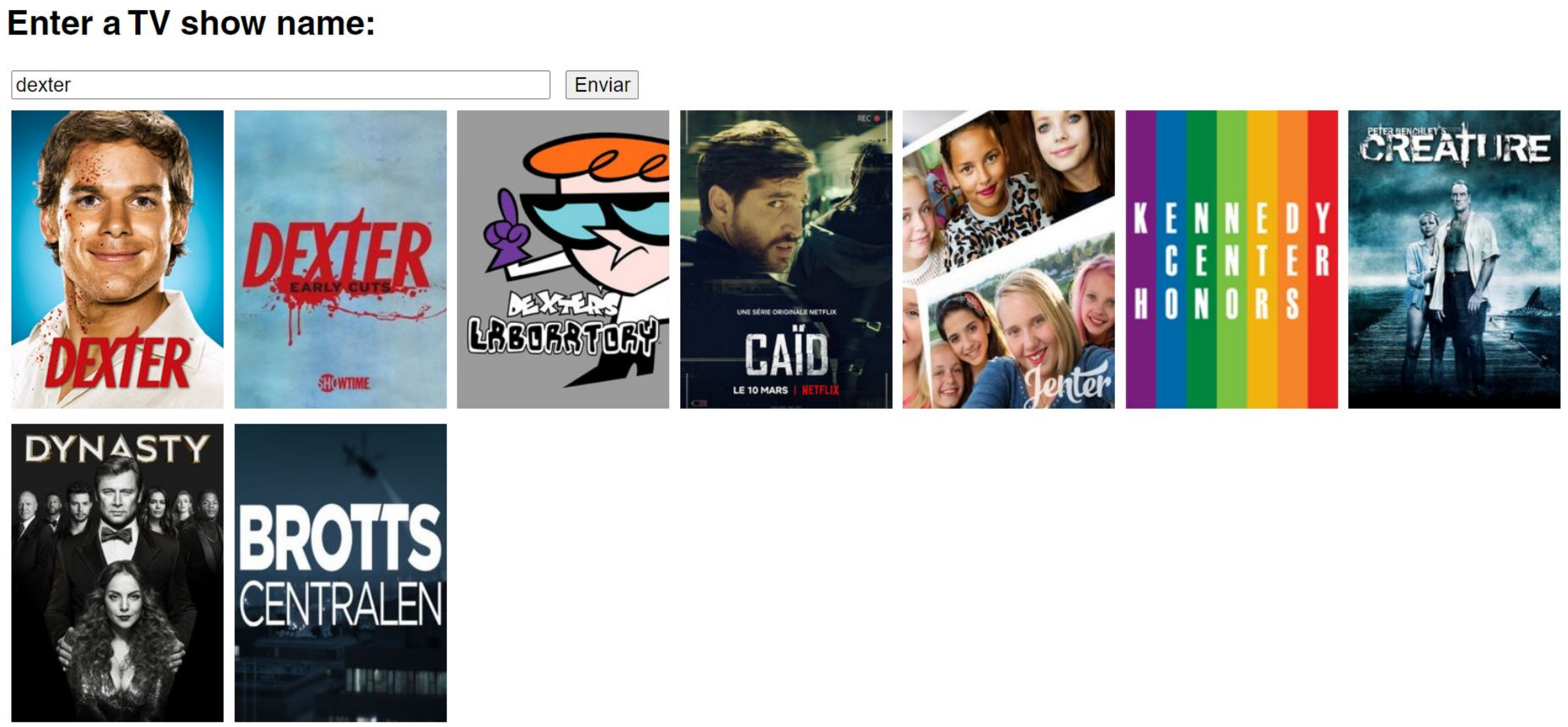
So if we do:

```
fetch('https://api.tvmaze.com/search/shows?q=The Witcher')
    .then(onResponse)
    .then(onTextReady);
```

...we can load the JSON data as a JavaScript object, as we did with our .json files!
(CodePen)

Shows example

Let's write a web page that asks the user to enter a search query, then displays the found tv shows


Enter a TV show name:

dexter

Enviar

TVMaze search API

TVMaze Search URL:

`https://api.tvmaze.com/search/shows?q=`***`query`***

E.g.

`https://api.tvmaze.com/search/shows?q=`***`The Witcher`***

Q: Hey, what's that at the end of the URL?

- `?q=The Witcher`

Query parameters

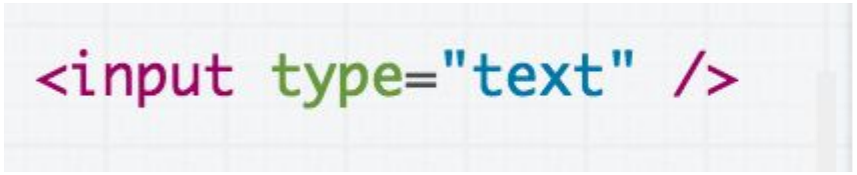
You can pass parameters to HTTP GET requests by adding **query parameters** to the URL:

```
?q=The Witcher&param2=...
```

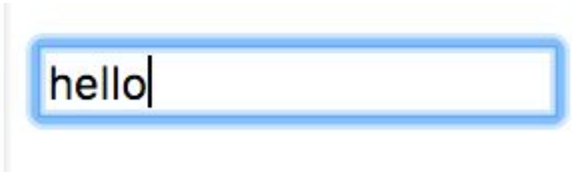
- Defined as key-value pairs
  - ***param=value***
- The first query parameter starts with a **?**
- Subsequent query parameters start with **&**

Reminder: HTML elements

Single-line text input:



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



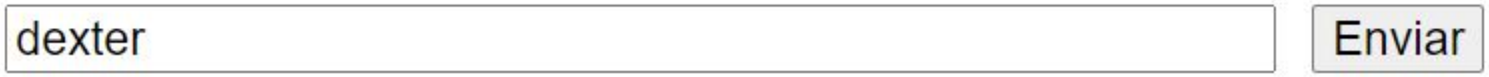
hello

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

Some other input types:

- Select
- Textarea
- Checkbox

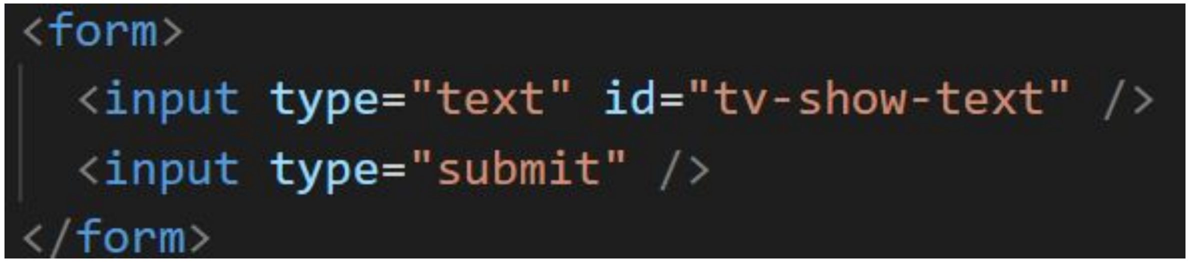
Form submit



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

Form submit

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

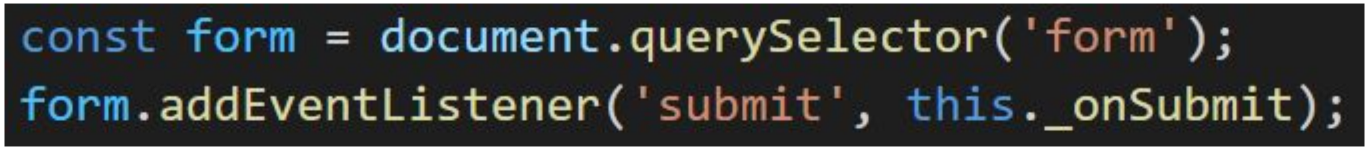


```html
<form>
  <input type="text" id="tv-show-text" />
  <input type="submit" />
</form>
```

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:

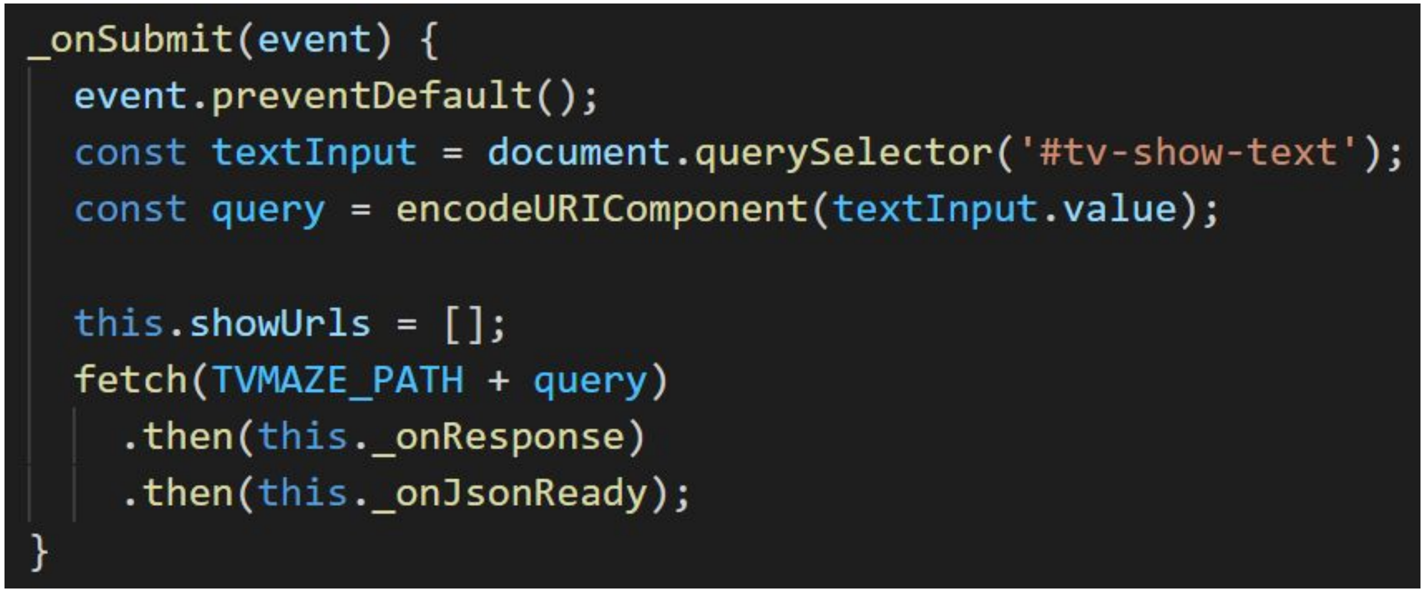


```
const form = document.querySelector('form');
form.addEventListener('submit', this._onSubmit);
```

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

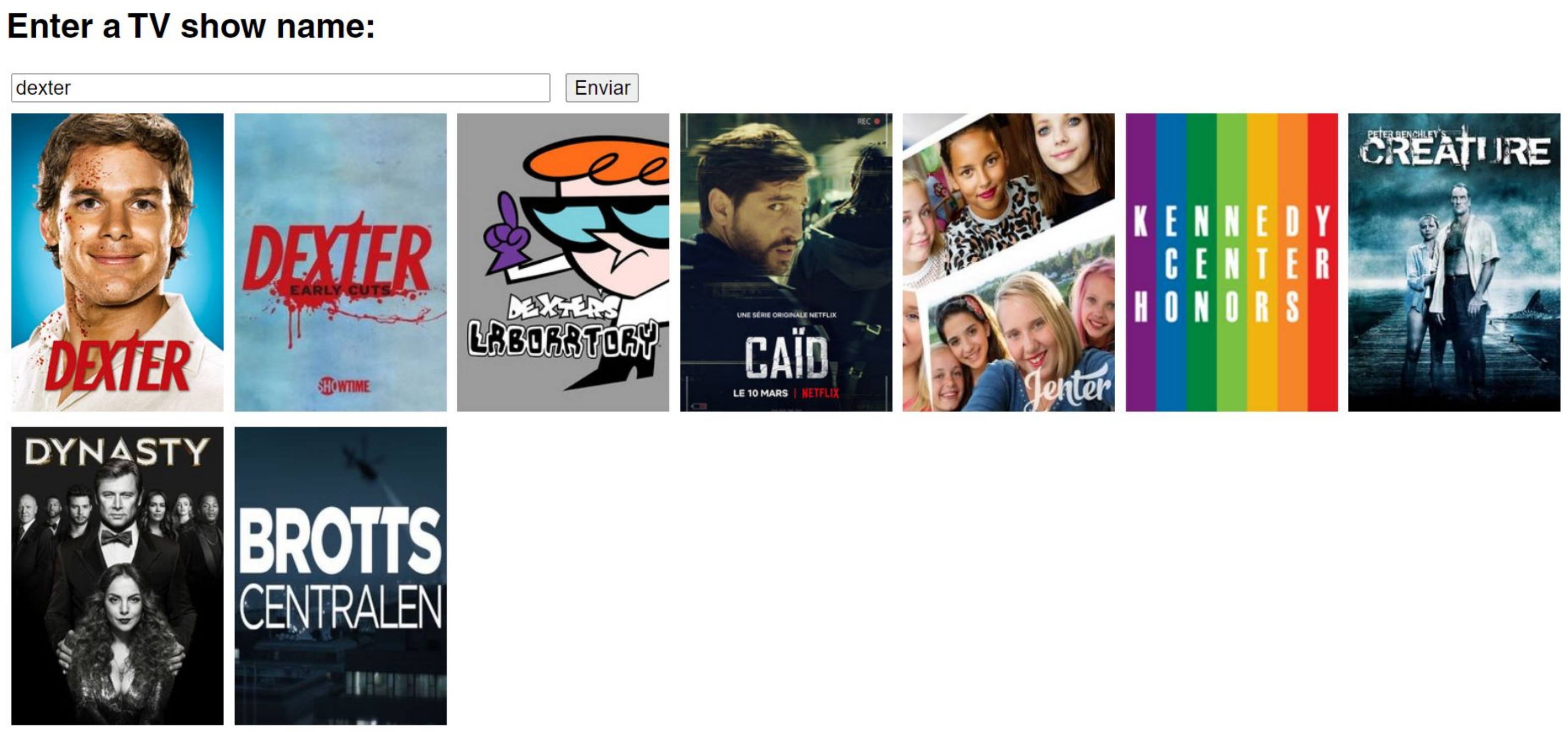
3. Prevent the default action before handling the event through `event.preventDefault()`:


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

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

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

Show search example

Solution: GitHub / Demo



Enter a TV show name:

dexter

Enviar