

<WA1/>  
<AW1/>  
2025

# Fetch API

Enabling the link to the server side

Fulvio Corno

Luigi De Russis

Enrico Masala



# Goal

- Loading data asynchronously
- Sending asynchronous HTTP requests
- Handling multiple requests
- Using alternative libraries



JavaScript: The Definitive Guide, 7th Edition  
Chapter 11. Asynchronous JavaScript

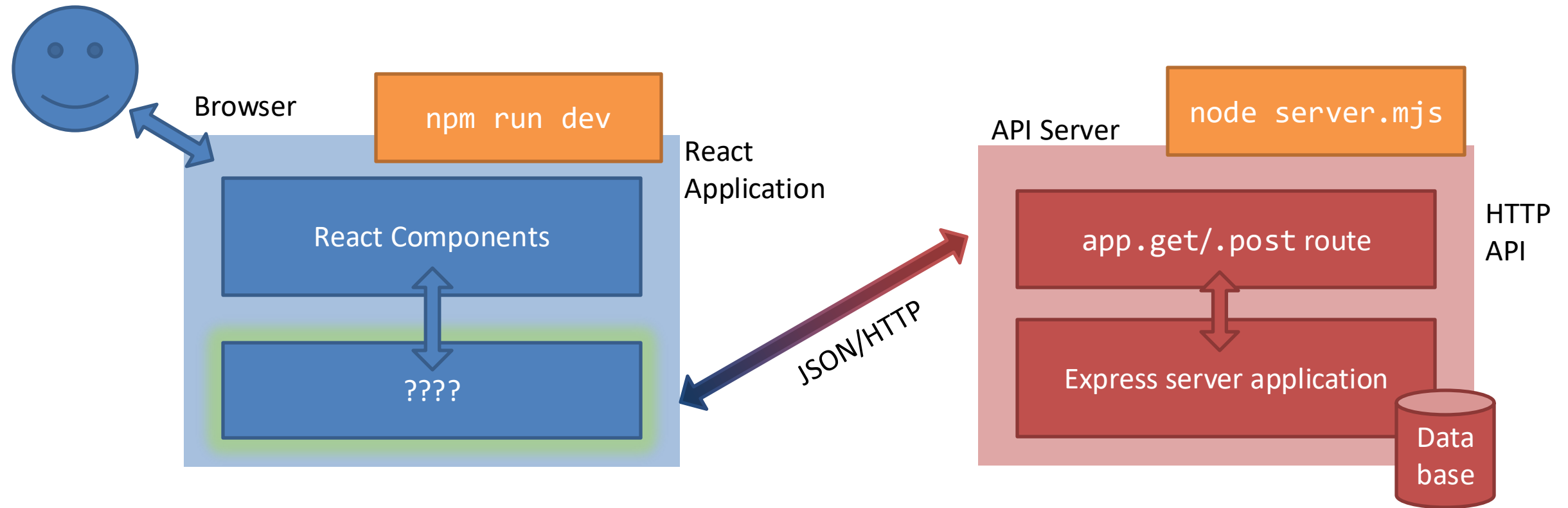
Mozilla Developer Network:  
Web technology for developers —  
Web API — Fetch API

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)

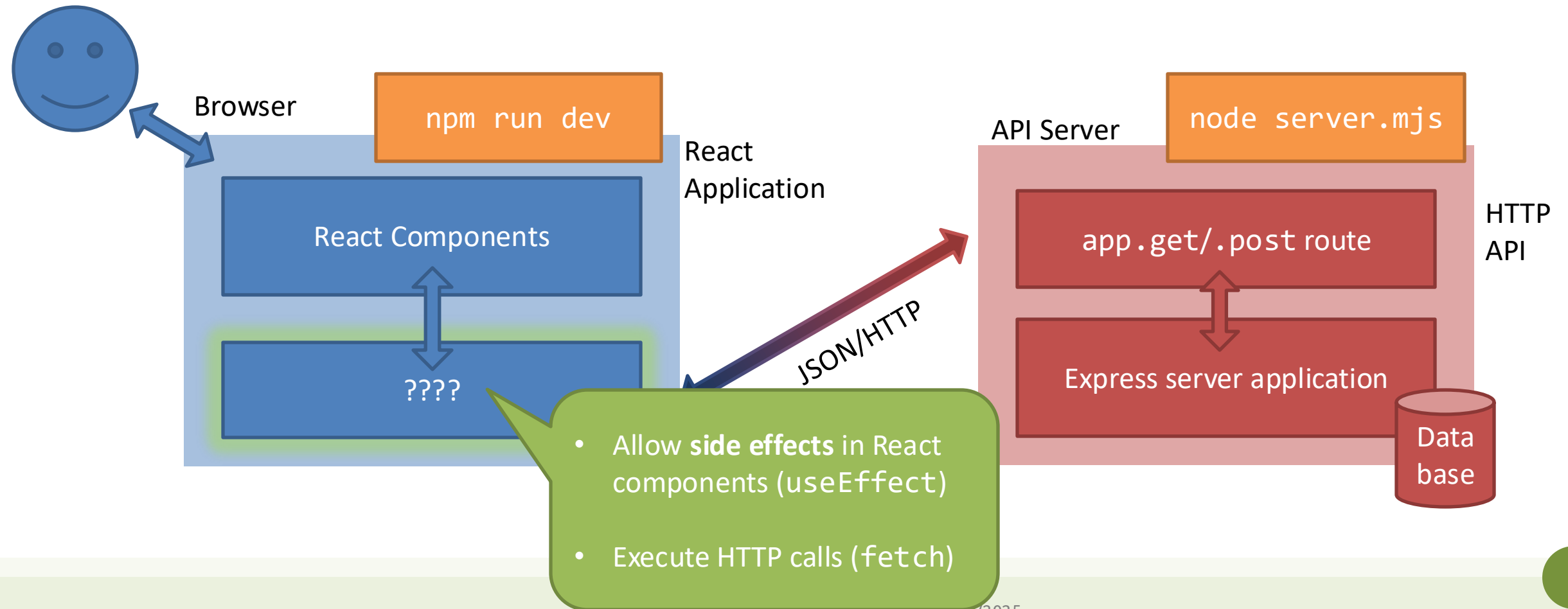
Fetch API

# ASYNCHRONOUS JS REQUESTS

# Asynchronous API Data Transfers



# Asynchronous API Data Transfers



# How to Exchange Data Asynchronously

- Make asynchronous HTTP requests using browser-provided Web API
- Use the Fetch API, i.e., **fetch()** method
  - Parameters: **URL** of the resource, **object** with request parameters (optional)
  - Default request type: **GET**
- Available in almost any context (e.g., from `window` object)
- Returns a **Promise** that will resolve once the load operation finishes
  - Resolves to the **Response** object, that allows to access the details of the HTTP transaction and the content
  - The promise is rejected only in case of network errors

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)

# Example

- Just handle the promise (`.then` or `await`)

```
fetch('http://example.com/exams.json')  
  .then((response) => {  
    return response.json();  
  })  
  .then((data) => {  
    console.log(data);  
  })
```

```
const response = await  
  fetch('http://example.com/exams.json');  
  
const data = await response.json();  
  
console.log(data);
```

# Response Object

- The fulfilled Promise returns a **Response** object
- Main properties
  - `Response.ok` (boolean): HTTP successful (code 200-299)
  - `Response.status`, `Response.statusText`
  - `Response.headers`: collection of HTTP headers of the response
  - `Response.url`: final URL (potentially after HTTP redirects)
  - `Response.body`: a readable stream of the body content

<https://developer.mozilla.org/en-US/docs/Web/API/Response>



# Accessing Response Headers

```
fetch('http://localhost/data.json')
  .then(response => {
    console.log(response.headers.get('Content-Type'));
    console.log(response.headers.get('Date'));

    console.log(response.status);
    console.log(response.statusText);
    console.log(response.type);
    console.log(response.url);
  })
```

```
application/html; charset=utf-8
Sat, 11 Apr 2020 13:41:04 GMT
```

```
404
Not Found
undefined
http://localhost/data.json
```

<https://developers.google.com/web/updates/2015/03/introduction-to-fetch>

# Error Handling

- Promise is only rejected for non-HTTP errors (e.g., **network** connection error, only)
  - Any HTTP status value (200 OK, 404: Not found, 500: Internal server error, ...) returns a **fulfilled** Promise
- *Suggested* error handling approach:
  - Check **response.ok**: true for HTTP status **200-299**
  - Check content type header (depends on the application needs)
  - Provide a `catch()` for other types of errors

# Example: Error Handling

```
fetch(url)
  .then(response => {
    if (!response.ok) { throw Error(response.statusText) }
    let type = response.headers.get('Content-Type');
    if (type !== 'application/json') {
      //then() returns a rejected promise if something is thrown
      throw new TypeError(`Expected JSON, got ${type}`)
    }
    return response;
  })
  .then(response => {
    //...
  })
  .catch(err => console.log(err)) // either the throw value or other errors
```

# Fetch Options

- `const fetchResponsePromise = fetch(resource [, init])`
- Main properties of (optional) ***init*** object
  - **method**
  - **headers** (an object with a property per each header)
  - **body**
  - **mode** (cors, no-cors, same-origin)
  - **credentials** (omit, same-origin, include), to send cookies with the request
  - **signal**: an AbortSignal object instance to communicate with the fetch request

<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch>

# Example: POST with JSON content

```
let objectToSend = { 'title': 'Do homework' , 'urgent': true, 'private': false,
  'sharedWithIds': [3, 24, 58] };

fetch(url, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(objectToSend), // Conversion in JSON format
})
.catch(function (error) {
  console.log('Failed to store data on server: ', error);
});
```

# Reading The Response Body

- Can use (**only once**) one of the following methods
  - ...then the body is “**consumed**”
- These methods **also return a Promise**, which returns the response body...
  - `response.text()`: as plain text (string)
  - `response.json()`: as a JS object, by parsing the body as JSON
  - `response.formData()`: as a FormData object
  - `response.blob()`: as Blob (binary data with type)
  - `response.arrayBuffer()`: as ArrayBuffer (low-level representation of binary data)
- `response.body` is a ReadableStreaming object to read it chunk-by-chunk

<https://javascript.info/fetch>

# Sequential Fetches

- Suggestion: use `async`, avoid nesting `fetch` in `.then()`

```
const getFirstUserData = async () => {  
  const response = await fetch('/users.json'); // get users list  
  const users = await response.json(); // parse JSON  
  
  const user = users[0]; // pick first user  
  
  const userResponse = await fetch(`/users/${user.name}`); // get user data  
  const userData = await userResponse.json(); // parse JSON  
  
  return userData;  
}
```

# Parallel Fetches

- Multiple fetches in parallel: use `Promise.all()`

```
// array of URLs
const urls = [url1, url2];

// Convert to an array of Promises
const promises = urls.map(url => fetch(url) );
// Wait only for the fetch Promise

// Run all promises in parallel, wait for all
Promise.all(promises)
  .then(results => { // process according to the order needed by the app
    for (const res of results) res.text().then( t => console.log(t) );
  })
  .catch(e => console.error(e))
```



# Basic Fetch vs. Other Libraries

- Most common alternative library: **Axios**
  - Does polyfill for older browsers
  - Has an easier way to cancel a request
  - Has a way to set a response **timeout** (not supported by fetch, which needs a `setTimeout()` to call the `AbortController.abort()` method)
  - Easier support for progress bar via Axios Progress Bar module (fetch requires quite some code around a `ReadableStream` object)
  - Performs automatic JSON conversion
  - Provides an easier way to separate responses of parallel requests
  - Works well also in Node.js (fetch is not included by default)

<https://flaviocopes.com/axios/>

<https://blog.logrocket.com/axios-or-fetch-api/>

# License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for [commercial purposes](#).
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
  - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

