

API avec NodeJS

Mounir BENDAHMANE



Qui suis-je ?

A qui s'adresse ce cours ?

Les prérequis

- Des notions de bases en développement Web
- Connaissances en Javascript moderne
- De la curiosité

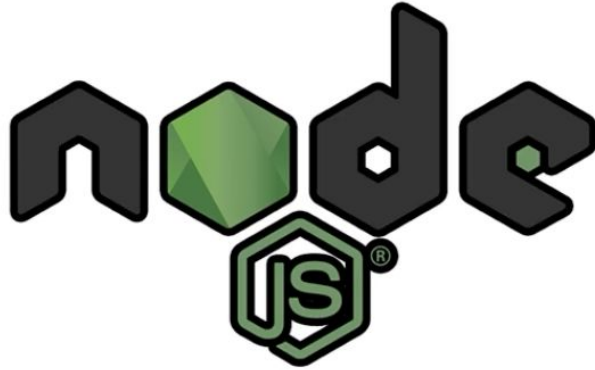
Objectifs:

A la fin de ce cours vous saurez :

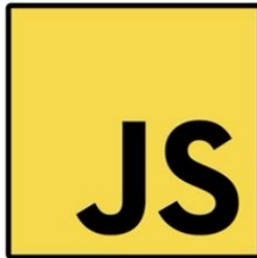
- Créer des APIs REST avec Node.js et le framework Express*
- Documenter et structurer correctement votre code
- Travailler en équipe sur un projet Node.js*
- Déployer une application Node.js
- Persister les données manipulées avec une base de données

Qu'est-ce que Node.js ?


Qu'est-ce que Node.js ?



A JavaScript Runtime

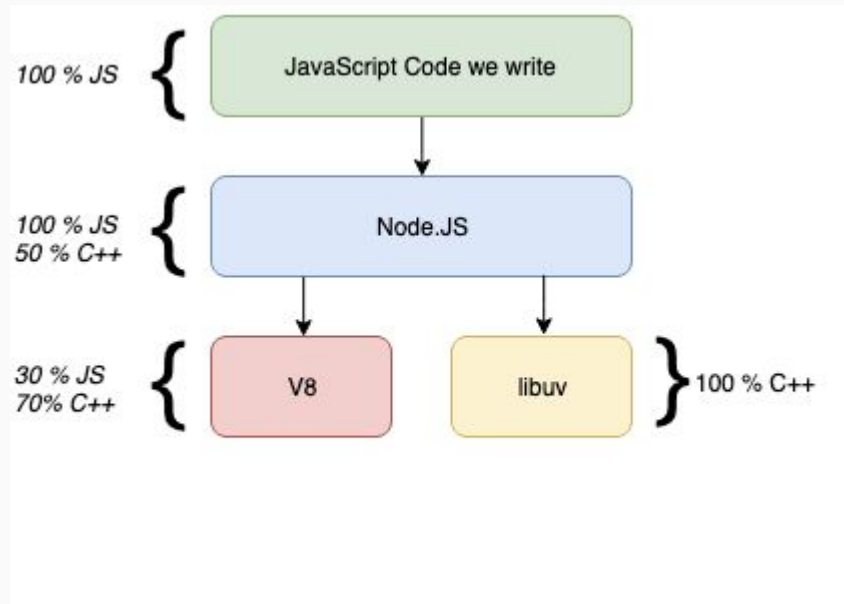


Qu'est-ce que Node.js ?

[Node](#)  (ou plus formellement *Node.js*) est un environnement d'exécution open-source, multi-plateforme, qui permet aux développeuses et développeurs de créer toutes sortes d'applications et d'outils côté serveur en [JavaScript](#). Cet environnement est destiné à être utilisé en dehors du navigateur (il s'exécute directement sur son ordinateur ou dans le système d'exploitation du serveur). Aussi, Node ne permet pas d'utiliser les API JavaScript liées au navigateur mais des API plus traditionnellement utilisées sur un serveur dont notamment celles pour HTTP ou la manipulation de systèmes de fichier.

Source: https://developer.mozilla.org/fr/docs/Learn/Server-side/Express_Nodejs/Introduction

Qu'est-ce que Node.js ?



Démo

Dans cette partie...

- Comment le Web fonctionne ?
- Création d'un serveur Node.js
- Utilisation des modules intégrés à Node
- Travailler avec des requêtes et des réponses (HTTP)
- Ecrire du code asynchrone (Event loop)

Les “core modules”

Http → lancer un serveur web, envoyer des requêtes

Https → lancer un serveur SSL

Fs

Path

Os

Les “core modules”

Http → lancer un serveur web, envoyer des requêtes *

Https → lancer un serveur SSL

Fs

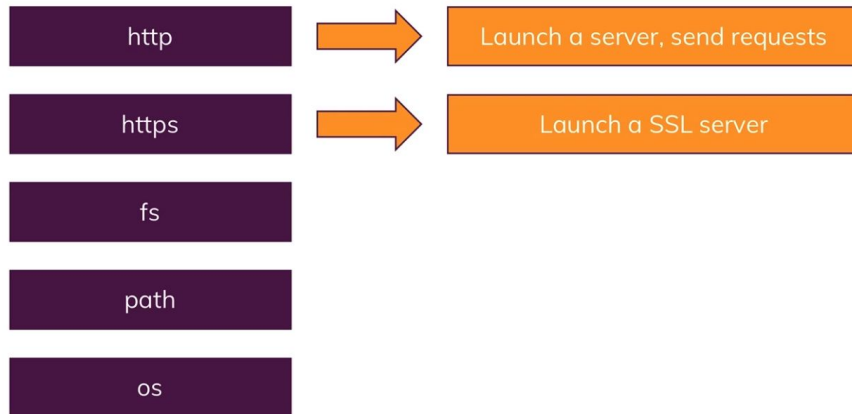
Path

Os

Créer un serveur web

Dans cette partie nous allons créer un simple serveur web avec NodeJS

Code démo



Le cycle de vie d'un programme Node

Création d'un eventListener

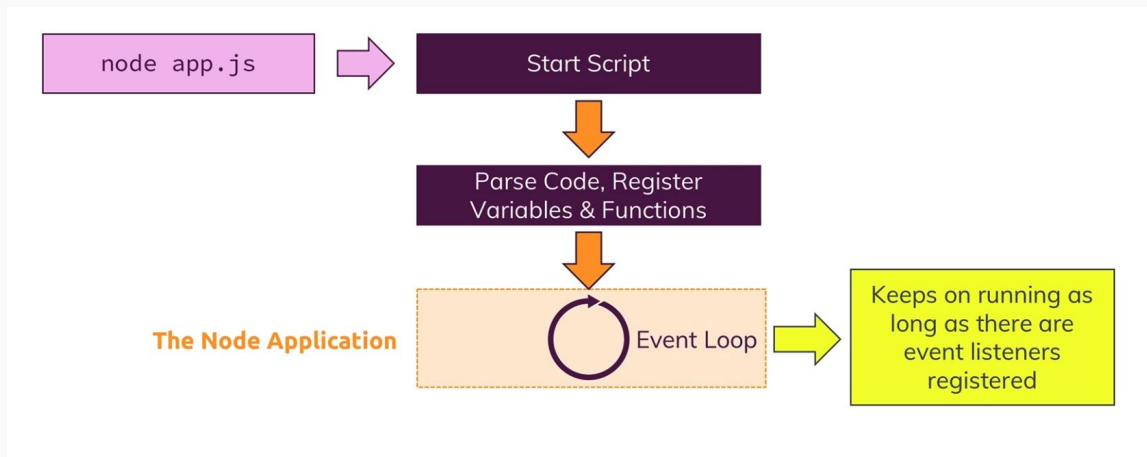
Code démo

```
JS app.js x
1  const http = require('http');
2
3  const server = http.createServer((req, res) => {
4    console.log(req);
5  });
6
7  server.listen(3000);
```

Le cycle de vie d'un programme Node

Création d'un eventListener

La "Node.js Event Loop"



Le cycle de vie d'un programme Node

Création d'un eventListener

La "Node.js Event Loop"

Process module

```
app.js  x
1  const http = require('http');
2
3  const server = http.createServer((req, res) => {
4    console.log(req);
5    process.exit();
6  });
7
8  server.listen(3000);
```

L'objet Request

Headers : métadonnées ajoutées à une requête http

```
httpVersion: '1.1',
complete: false,
headers:
  { host: 'localhost:3000',
    connection: 'keep-alive',
    'cache-control': 'max-age=0',
    'upgrade-insecure-requests': '1',
    'user-agent':
      'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440
06 Safari/537.36',
    accept:
      'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
    'accept-encoding': 'gzip, deflate, br',
    'accept-language': 'en-US,en;q=0.9,de-DE;q=0.8,de;q=0.7',
    cookie: '_ga=GA1.1.922359435.1535034935' },
rawHeaders:
  [ 'Host',
    'localhost:3000',
    'Connection',
    'keep-alive',
    'Cache-Control',
    'max-age=0',
    'Upgrade-Insecure-Requests',
    '1',
    'User-Agent',
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.
06 Safari/537.36',
    'Accept',
```

L'objet Request

Headers : métadonnées ajoutées à une requête http

Url :

Method:

Body:

L'objet Response

```
const server = http.createServer((req, res) => {  
  console.log(req.url, req.method, req.headers);  
  // process.exit();  
  res.setHeader('Content-Type', 'text/html');  
  res.write('<html>');  
  res.write('<head><title>My First Page</title><head>');  
  res.write('<body><h1>Hello from my Node.js Server!</h1></body>');  
  res.write('</html>');  
  res.end();  
});  
  
server.listen(3000);
```

L'objet Response

Pour aller plus loin :

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

Router la requête http

Récupérer l'information d'une requête sur une url spécifique

Stocker cette information sur dans un fichier

Comprendre comment Node manipule les requêtes Http

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

const server = http.createServer((req, res) => {
  const url = req.url;
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>My First Page</title><head>');
  res.write('<body><h1>Hello from my Node.js Server!</h1></body>');
  res.write('</html>');
  res.end();
});

server.listen(3000);
```



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

const server = http.createServer((req, res) => {
  const url = req.url;
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>My First Page</title><head>');
  res.write('<body><h1>Hello from my Node.js Server!</h1></body>');
  res.write('</html>');
  res.end();
});

server.listen(3000);
```

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

```
const server = http.createServer((req, res) => {
```

```
  const url = req.url;
```

```
  if (url === '/') {
```

```
    res.write('<html>');
```

```
    res.write('<head><title>Enter Message</title><head>');
```

```
    res.write('<body><form action="/message" method="POST"><input type="text" name="message"><');  
    res.write('</html>');
```

```
    return res.end();
```

```
  }
```

```
  res.setHeader('Content-Type', 'text/html');
```

```
  res.write('<html>');
```

```
  res.write('<head><title>My First Page</title><head>');
```

```
  res.write('<body><h1>Hello from my Node.js Server!</h1></body>');
```

```
  res.write('</html>');
```

```
  res.end();
```

```
});
```

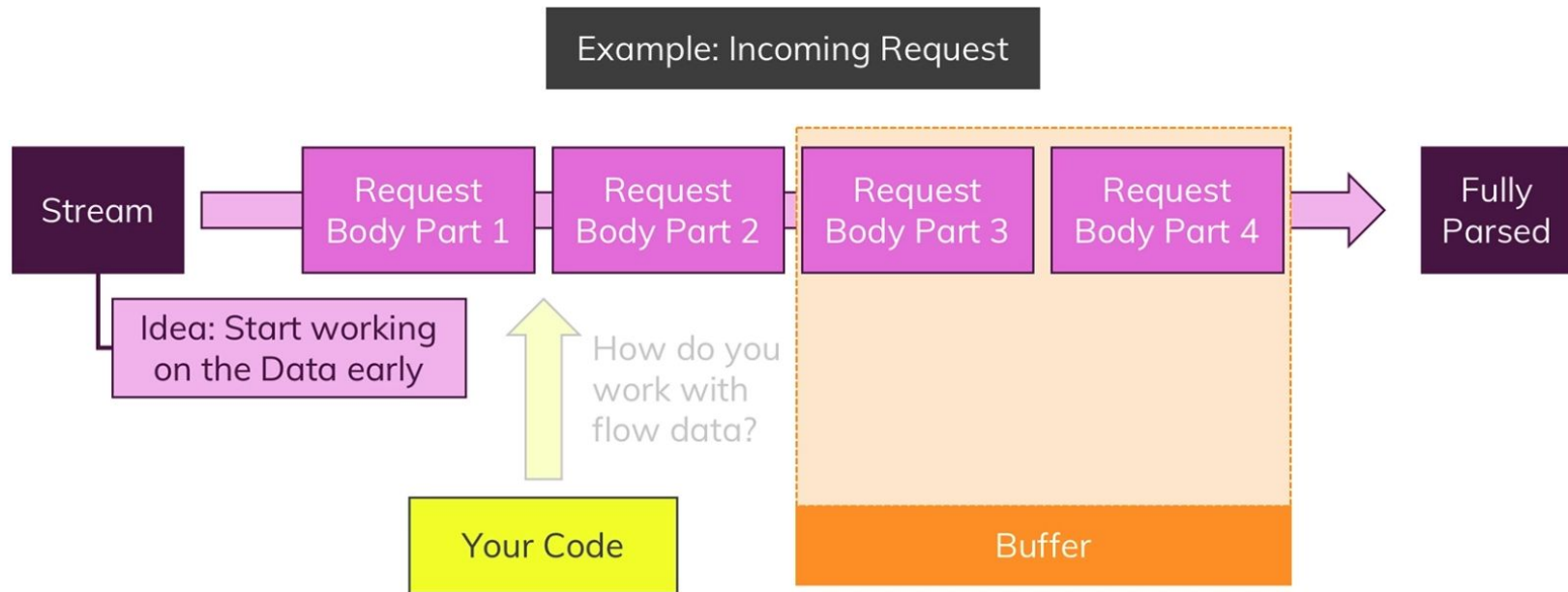
I

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

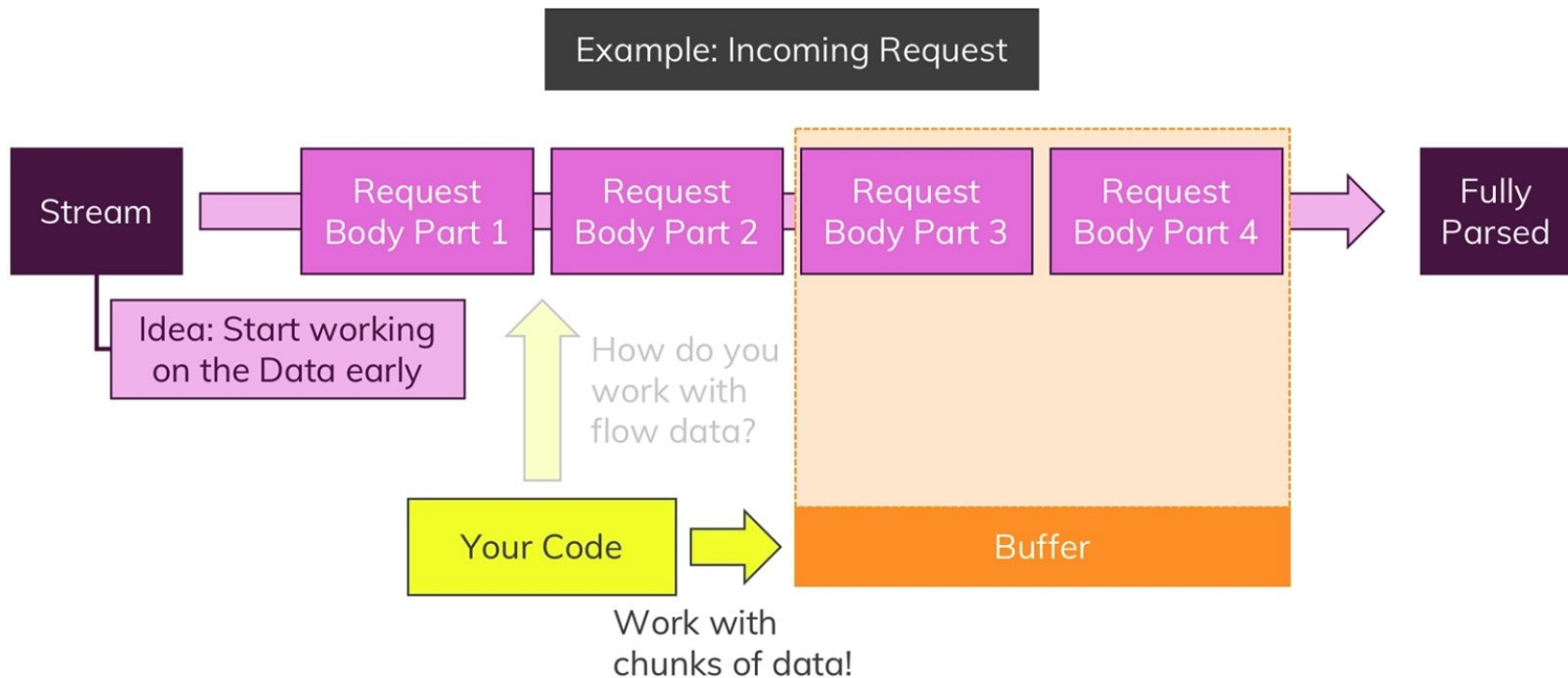
Router la http

Démo de code

Comment récupérer les données des requêtes entrantes ?



Comment récupérer les données des requêtes entrantes ?



Exécution de code pilotée par des évènements

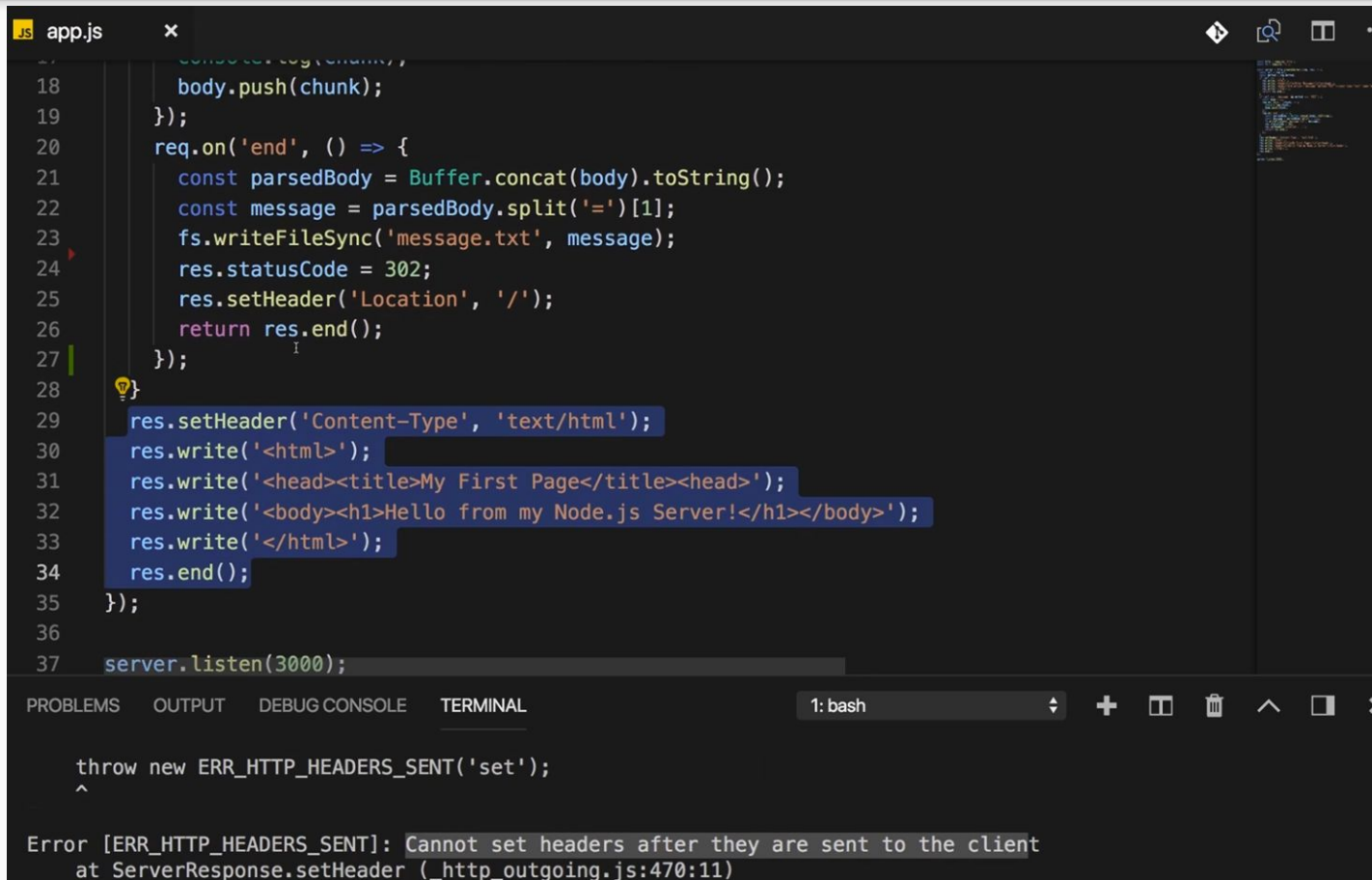
Node JS possède un registre global interne d'Events et d'EventListeners

Le code est exécuté de façon asynchrone

Attention à la portée des fonctions passées en argument d'autres fonctions

Notion de callback

Exécution de code pilotée par des évènements : erreur type



```
18     body.push(chunk);
19   });
20   req.on('end', () => {
21     const parsedBody = Buffer.concat(body).toString();
22     const message = parsedBody.split('=')[1];
23     fs.writeFileSync('message.txt', message);
24     res.statusCode = 302;
25     res.setHeader('Location', '/');
26     return res.end();
27   });
28
29   res.setHeader('Content-Type', 'text/html');
30   res.write('<html>');
31   res.write('<head><title>My First Page</title><head>');
32   res.write('<body><h1>Hello from my Node.js Server!</h1></body>');
33   res.write('</html>');
34   res.end();
35 });
36
37 server.listen(3000);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: bash

```
throw new ERR_HTTP_HEADERS_SENT('set');
^
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
at ServerResponse.setHeader (_http_outgoing.js:470:11)
```

Code synchrone et Code Asynchrone

```
if (url === '/message' && method === 'POST') {  
  const body = [];  
  req.on('data', (chunk) => {  
    console.log(chunk);  
    body.push(chunk);  
  });  
  return req.on('end', () => {  
    const parsedBody = Buffer.concat(body).toString();  
    const message = parsedBody.split('=')[1];  
    fs.writeFileSync('message.txt', message);  
    res.statusCode = 302;  
    res.setHeader('Location', '/');  
    return res.end();  
  });  
}
```


Code synchrone et Code Asynchrone

```
if (url === '/message' && method === 'POST') {  
  const body = [];  
  req.on('data', (chunk) => {  
    console.log(chunk);  
    body.push(chunk);  
  });  
  return req.on('end', () => {  
    const parsedBody = Buffer.concat(body).toString();  
    const message = parsedBody.split('=')[1];  
    fs.writeFileSync('message.txt', message);  
    res.statusCode = 302;  
    res.setHeader('Location', '/');  
    return res.end();  
  });  
}
```

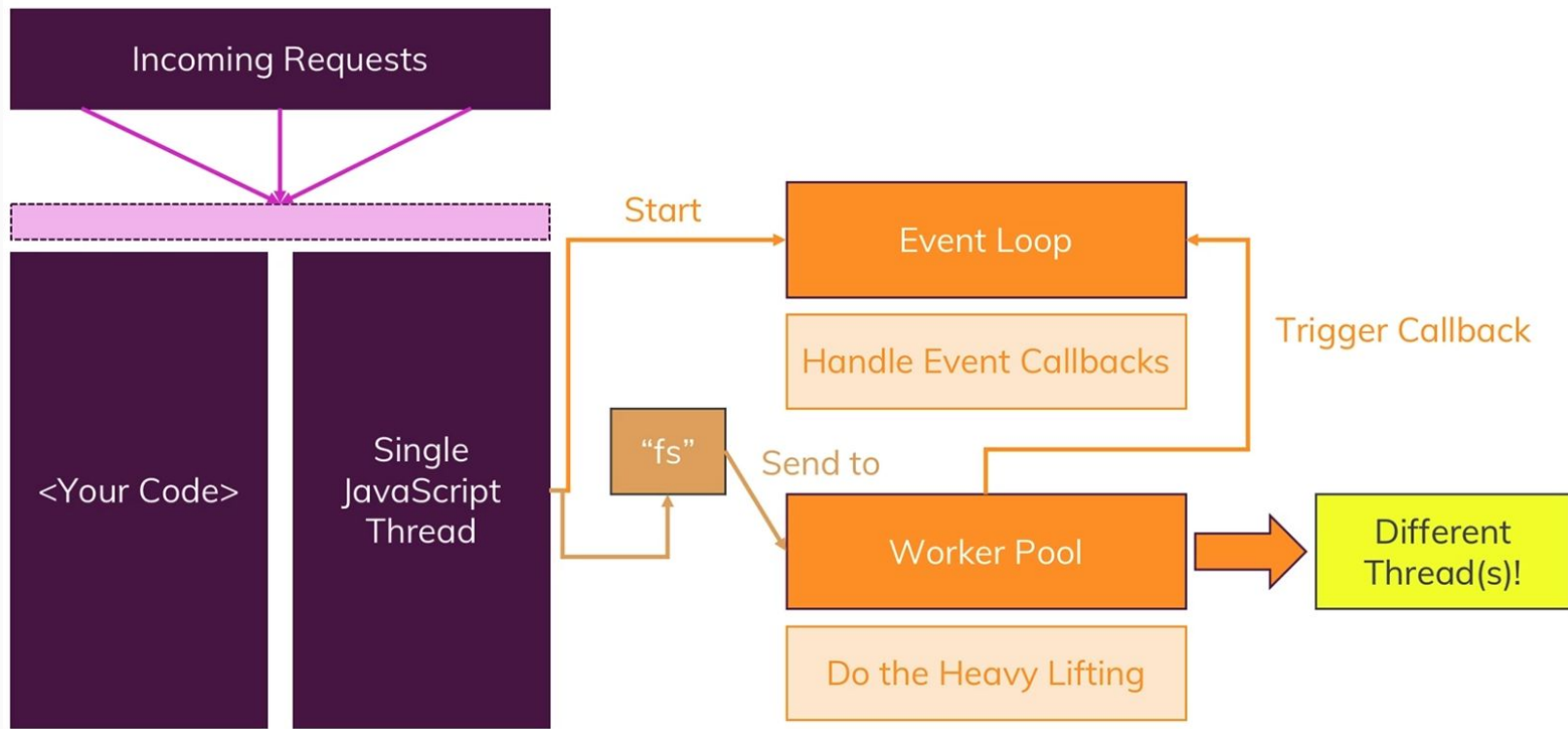
Code synchrone et Code Asynchrone

```
    body.push(chunk);
  });
  return req.on('end', () => {
    const parsedBody = Buffer.concat(body).toString();
    const message = parsedBody.split('=')[1];
    fs.writeFile('message.txt', message, err => {
      res.statusCode = 302;
      res.setHeader('Location', '/');
      return res.end();
    });
  });
}
res.setHeader('Content-Type', 'text/html');
res.write('<html>');
res.write('<head><title>My First Page</title></head>');
```

On parle d' "Event driven architecture"

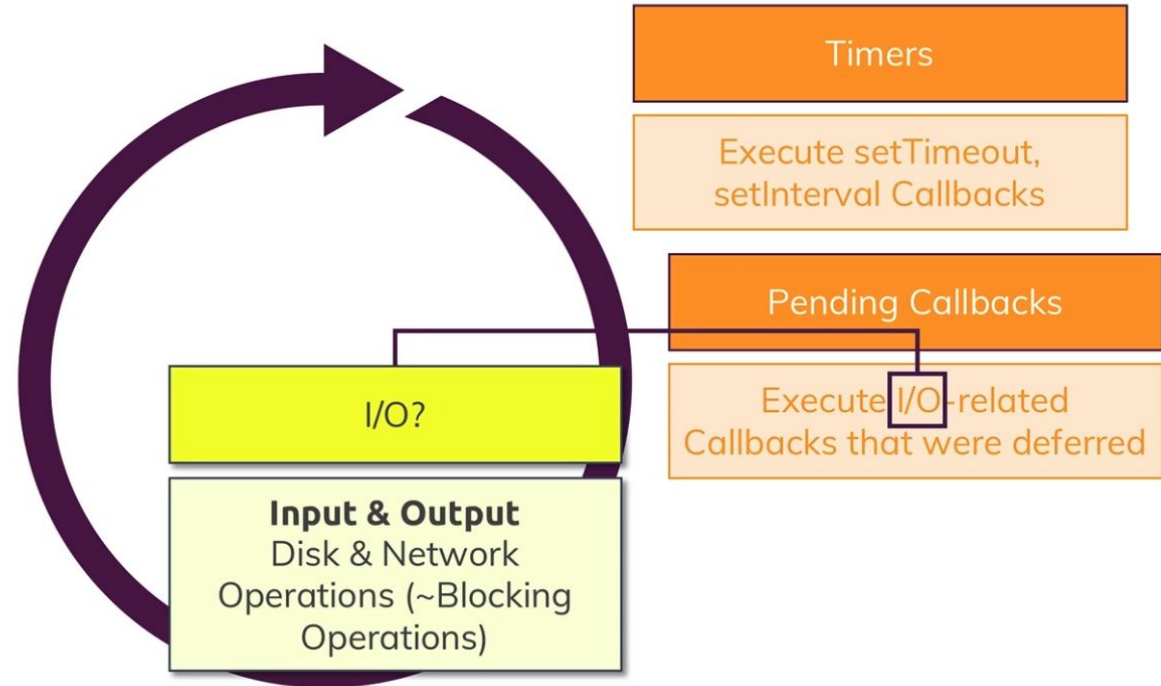
Code synchrone et Code Asynchrone

Thread, Event Loop et Code bloquant



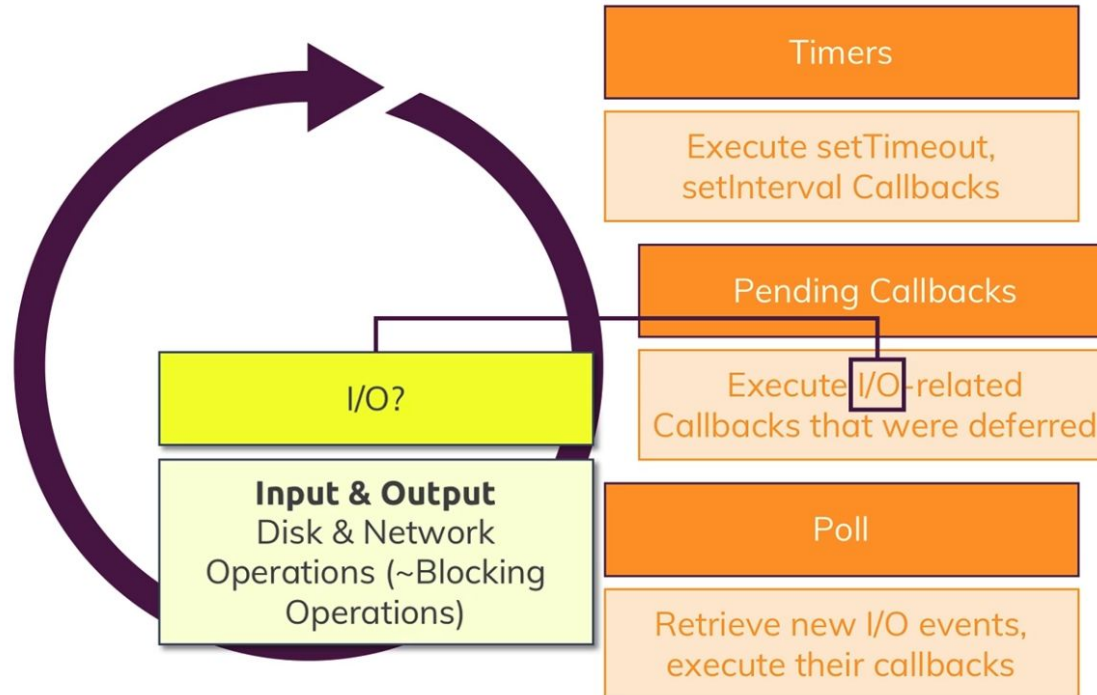
Code synchrone et Code Asynchrone

Event Loop



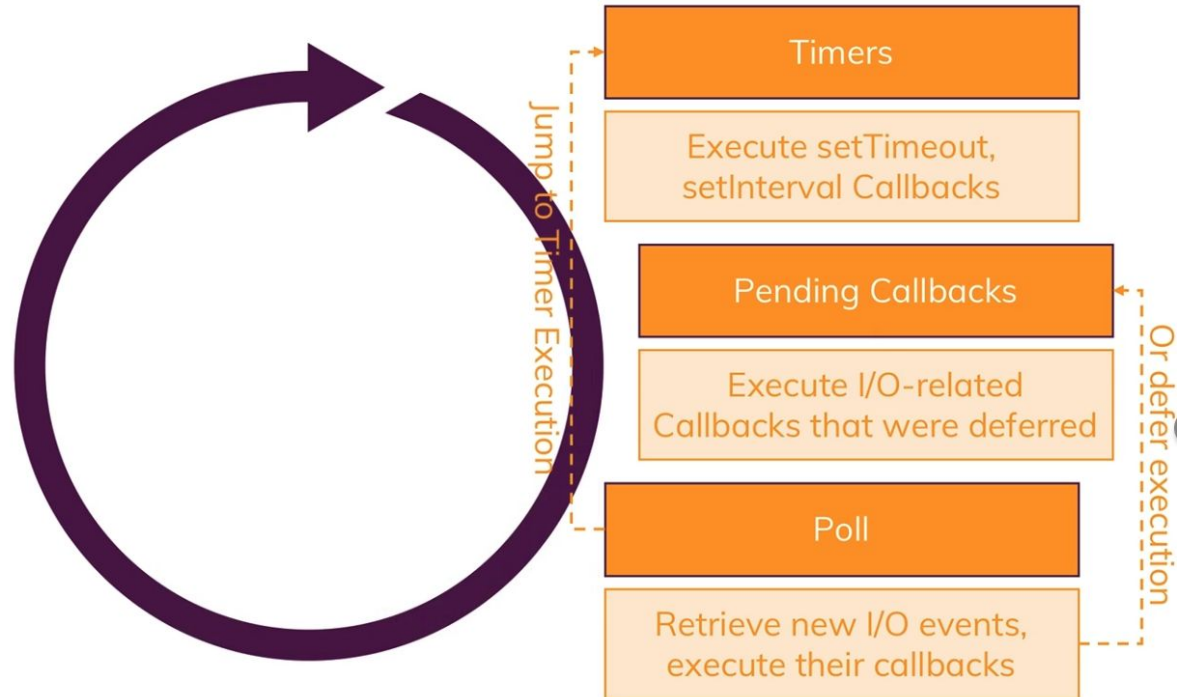
Code synchrone et Code Asynchrone

Event Loop



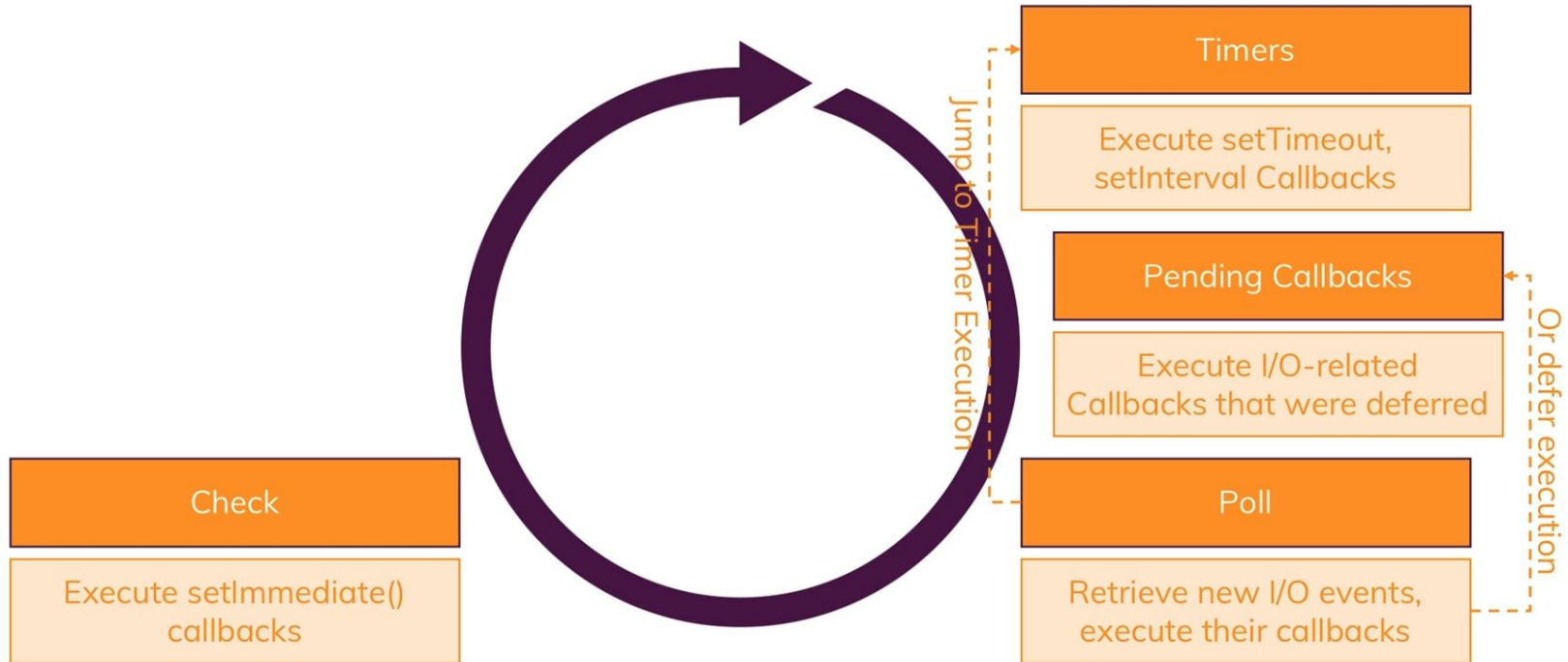
Code synchrone et Code Asynchrone

Event Loop



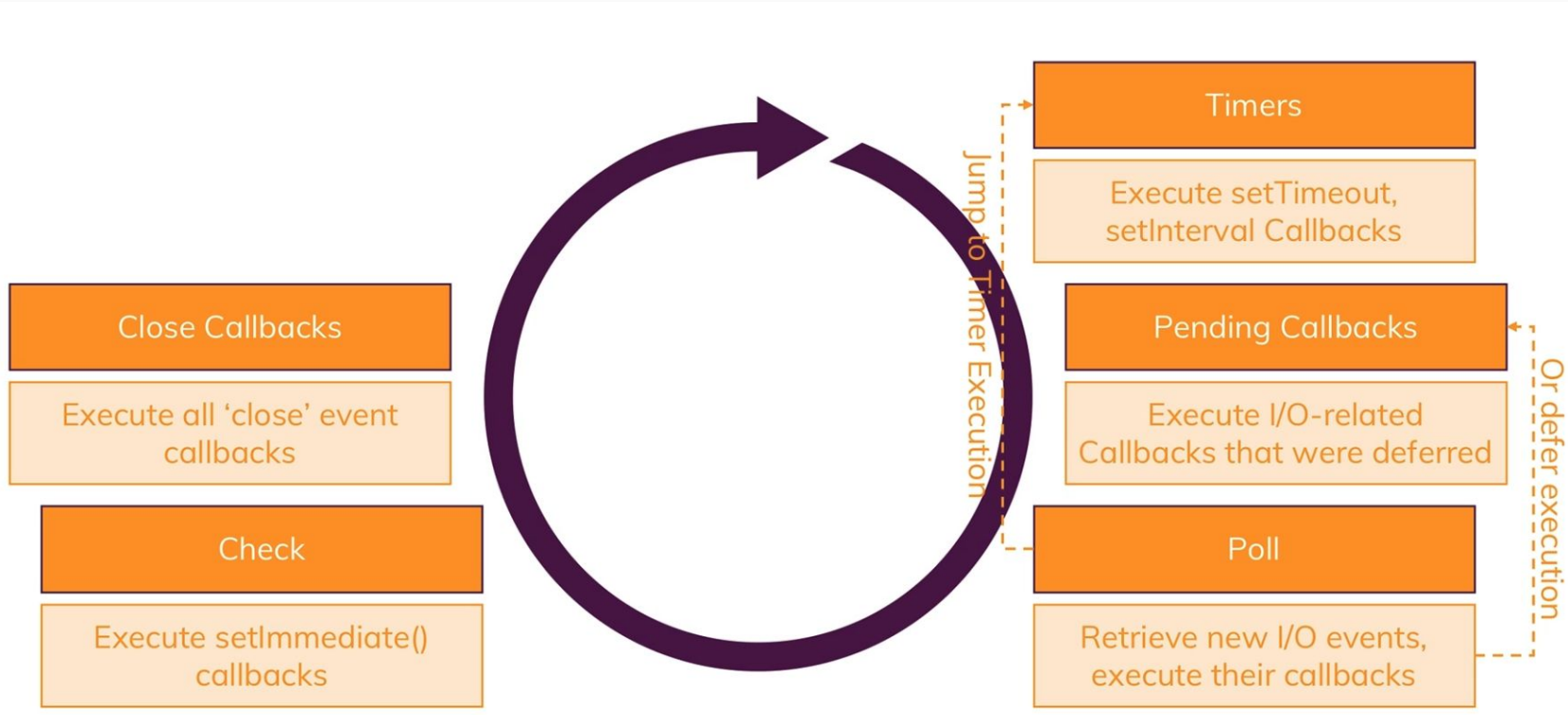
Code synchrone et Code Asynchrone

Event Loop



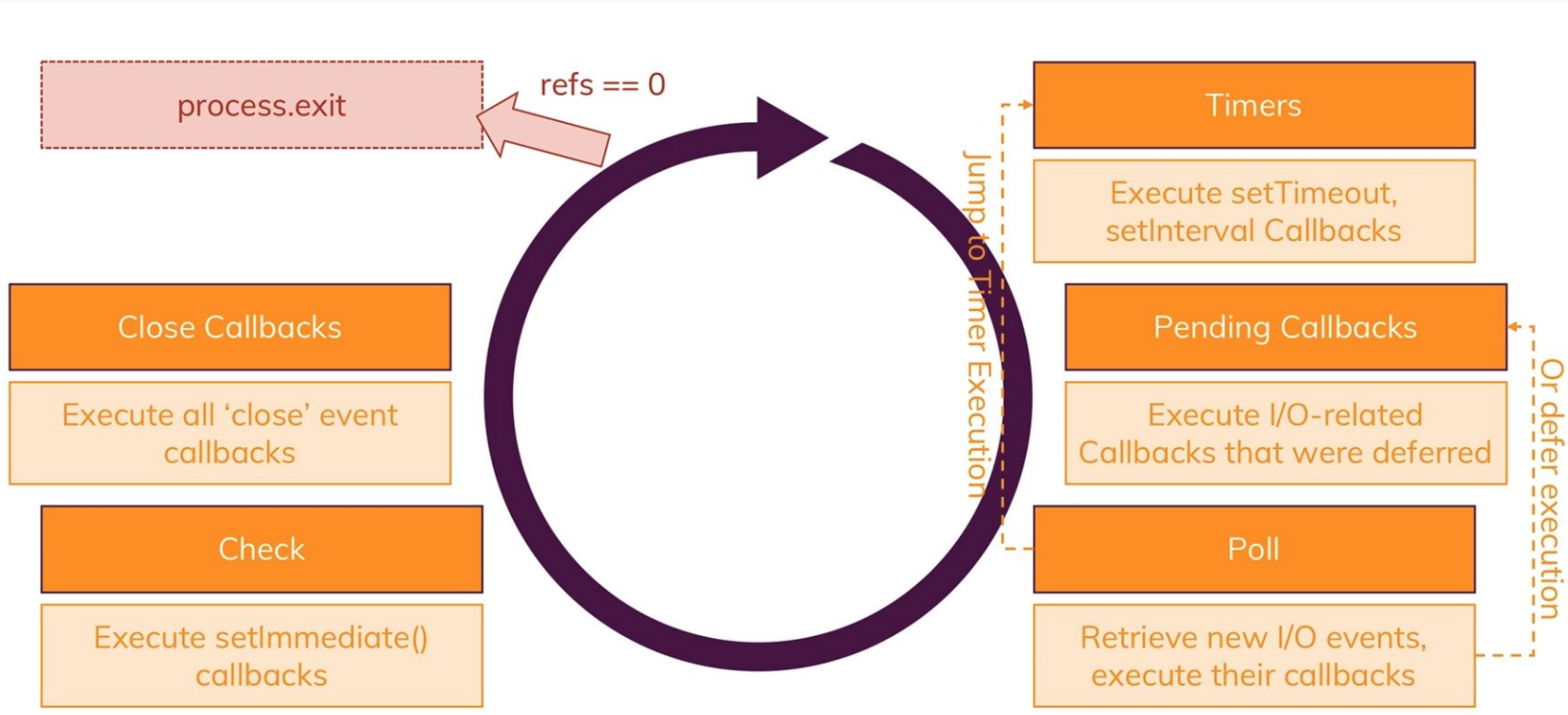
Code synchrone et Code Asynchrone

Event Loop

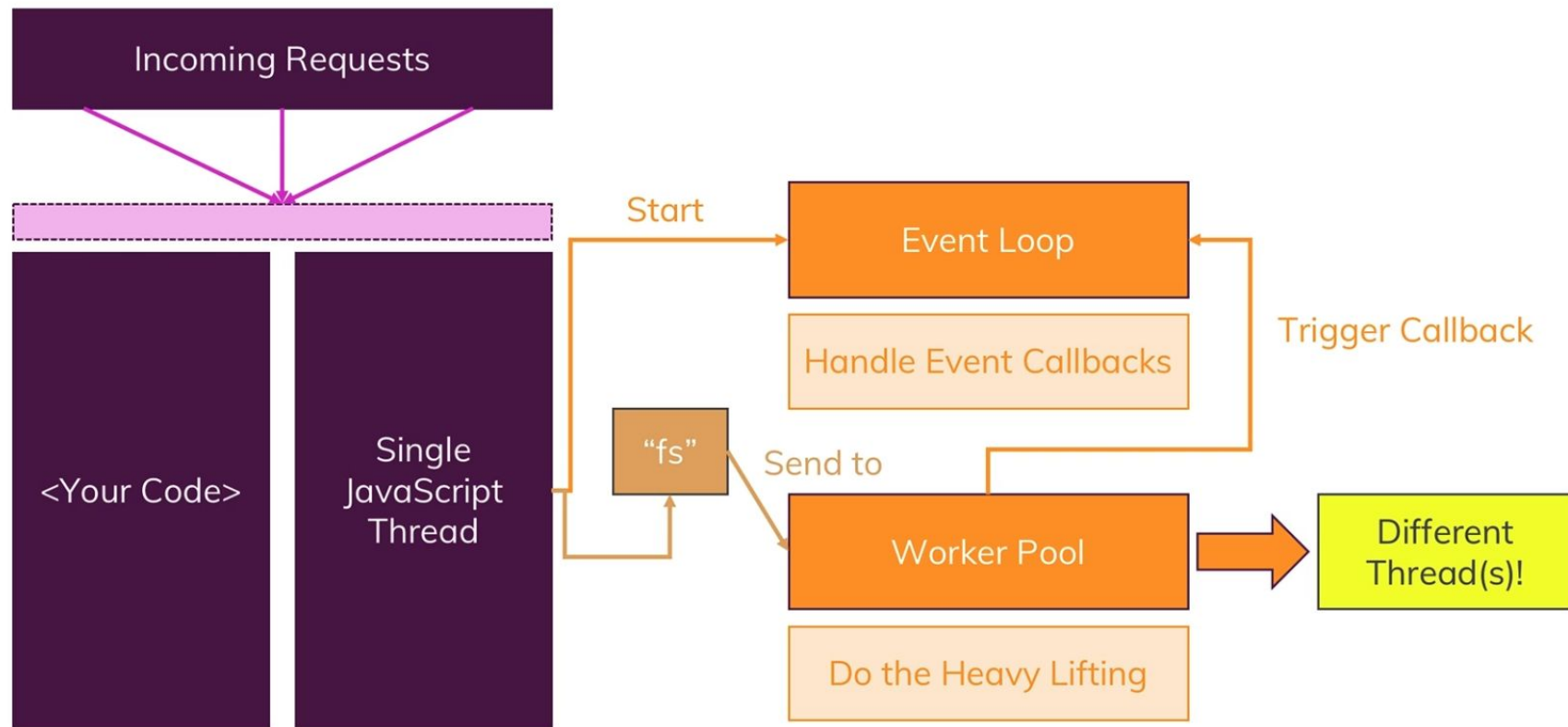


Code synchrone et Code Asynchrone

Event Loop



A retenir



Améliorons notre code

Méthode de travail pour ce module

Exercise

Exercice

Créer un serveur Nodejs

Créer au moins trois routes

Gérer une liste d'utilisateurs : l'afficher, ajouter des utilisateurs

Rediriger l'utilisateur à la liste après une création

Dans cette partie...

- Installation d'Express
- Qu'est-ce qu'une API REST ?
- Architecture de base d'une API avec Node.js

Express JS

Express JS

Gérer la logique du serveur peut très vite s'avérer compliqué

Nous voulons nous intéresser à notre Business Logic uniquement

Utilisons un framework pour faire tout ce travail à notre place

Il existe d'autres alternatives à Express...

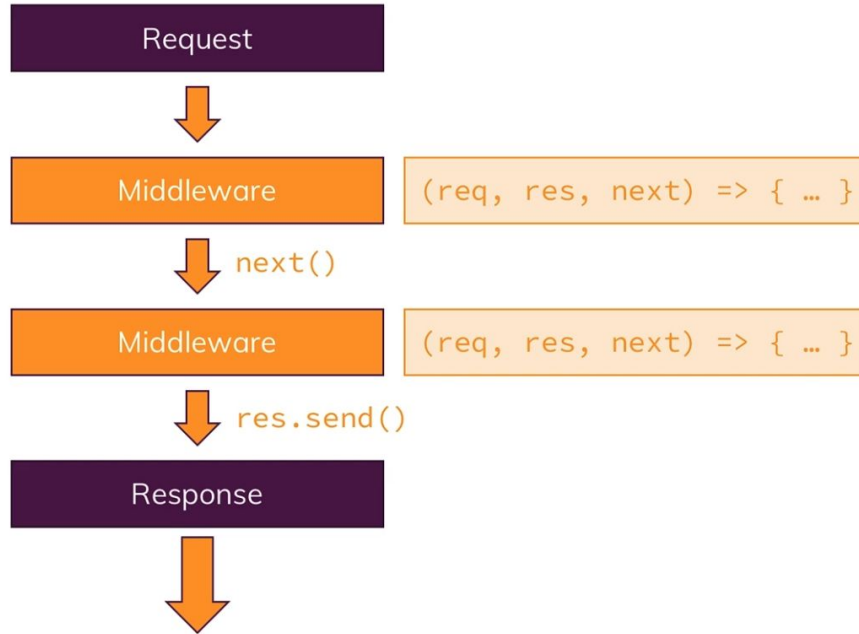
Alternatives à Express JS

- Rester sur du Node.js (Vanilla)
- Koa
- Sails
- Nest
- Et une multitude d'autres...

Installation d'Express

Démo

Notion de middleware



Express JS

<https://github.com/expressjs/express>

Express JS : res.send()

```
88     if (link) link += ', ';
89     return this.set('Link', link + Object.keys(links).map(function(rel){
90         return '<' + links[rel] + '>; rel="' + rel + '"';
91     }).join(', '));
92 };
93
94 /**
95  * Send a response.
96  *
97  * Examples:
98  *
99  *   res.send(Buffer.from('wahoo'));
100  *   res.send({ some: 'json' });
101  *   res.send('<p>some html</p>');
102  *
103  * @param {string|number|boolean|object|Buffer} body
104  * @public
105  */
106
107 res.send = function send(body) {
108     var chunk = body;
109     var encoding;
110     var req = this.req;
111     var type;
112
113     // settings
114     var app = this.app;
115
116     // allow status / body
117     if (arguments.length === 2) {
118         // res.send(body, status) backwards compat
119         if (typeof arguments[0] !== 'number' && typeof arguments[1] === 'number') {
```

Express JS: app.listen()

```
594
595 /**
596  * Listen for connections.
597  *
598  * A node 'http.Server' is returned, with this
599  * application (which is a 'Function') as its
600  * callback. If you wish to create both an HTTP
601  * and HTTPS server you may do so with the "http"
602  * and "https" modules as shown here:
603  *
604  *   var http = require('http')
605  *       , https = require('https')
606  *       , express = require('express')
607  *       , app = express();
608  *
609  *   http.createServer(app).listen(80);
610  *   https.createServer({ ... }, app).listen(443);
611  *
612  * @return {http.Server}
613  * @public
614  */
615
616 app.listen = function listen() {
617   var server = http.createServer(this);
618   return server.listen.apply(server, arguments);
619 };
620
621 /**
622  * Log error using console.error.
623  *
624  * @param {Error} err
625  */
```

Express JS: routage basique

`app.use([path,] callback [, callback...])`

Mounts the specified [middleware](#) function or functions at the specified path; the middleware function is executed when the base of the requested path matches `path`.

Arguments

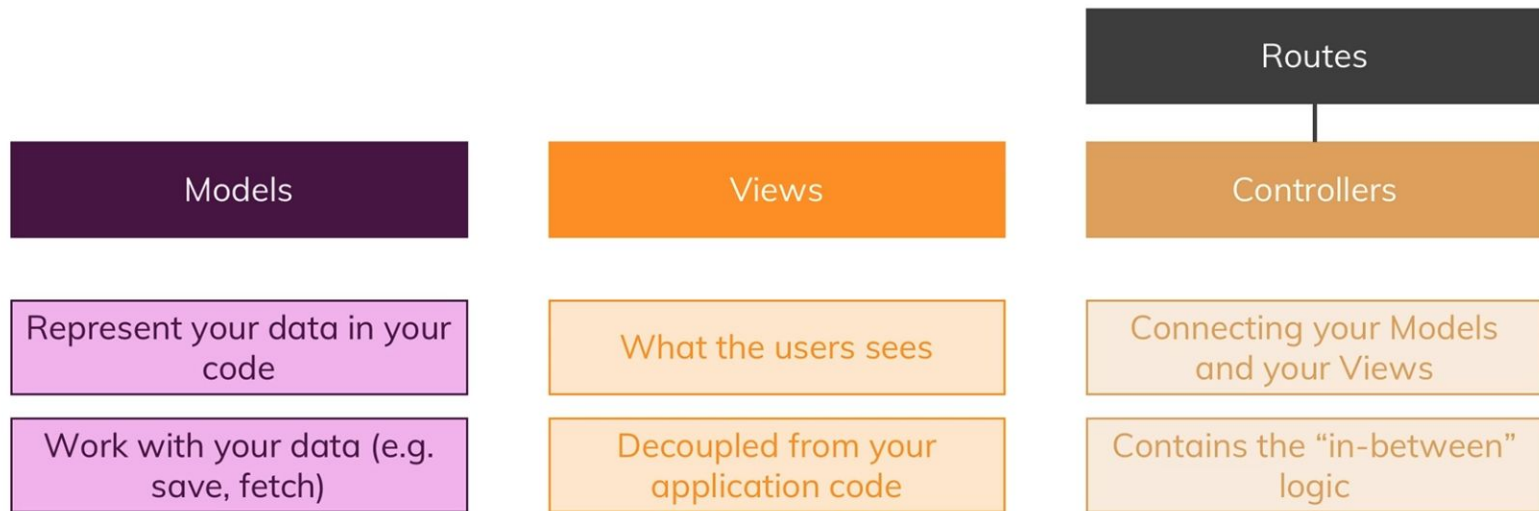
Argument	Description	Default
<code>path</code>	<p>The path for which the middleware function is invoked; can be any of:</p> <ul style="list-style-type: none">• A string representing a path.• A path pattern.• A regular expression pattern to match paths.• An array of combinations of any of the above. <p>For examples, see Path examples.</p>	<code>/</code> (root path)
<code>callback</code>	<p>Callback functions; can be:</p> <ul style="list-style-type: none">• A middleware function.• A series of middleware functions (separated by commas).• An array of middleware functions.• A combination of all of the above. <p>You can provide multiple callback functions that behave just like middleware, except that these callbacks can invoke <code>next('route')</code> to bypass the remaining route callback(s). You can use this mechanism to impose pre-conditions on a route, then pass control to subsequent routes if there is no reason to proceed with the current route.</p> <p>Since router and app implement the middleware interface, you can use them as you would any other middleware function.</p> <p>For examples, see Middleware callback function examples.</p>	None

Description

A route will match any path that follows its path immediately with a `"/`. For example: `app.use('/apple', ...)` will match `/apple`, `/apple/images`, `/apple/images/news`, and so on.

Express JS: design Pattern MVC

Separation of Concerns



Les APIs REST

Découpler Frontend et Backend

Toutes les applications (UI) ne nécessitent pas forcément un rendu de code HTML côté Serveur:

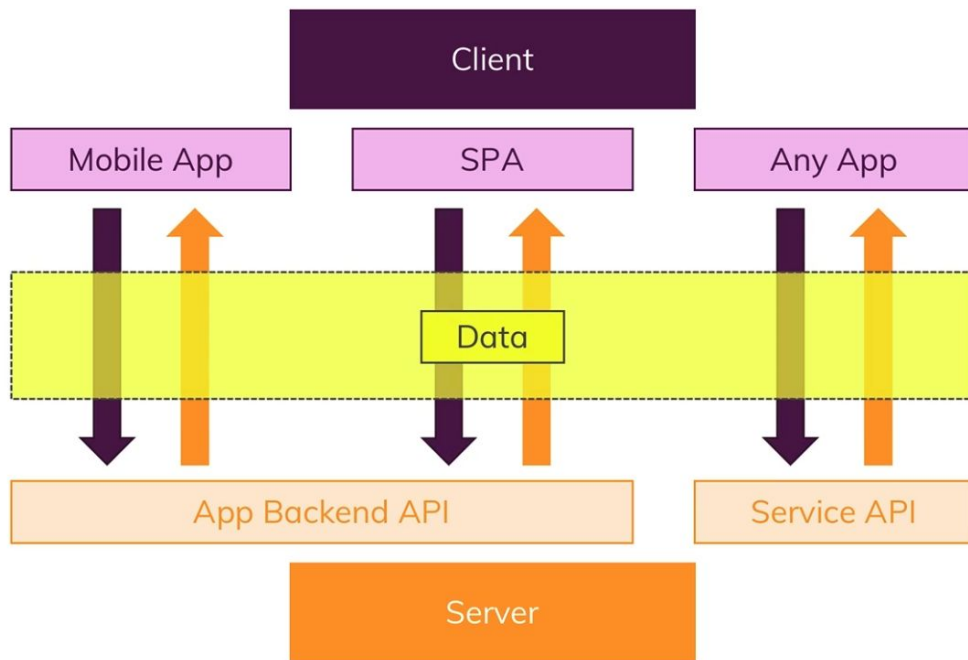
- Les applications Mobiles
- Les SPAs
- Les APIs de service (Google Maps API, Poké API)

Les APIs REST

Representational State Transfer

Transfer Data instead of User Interfaces

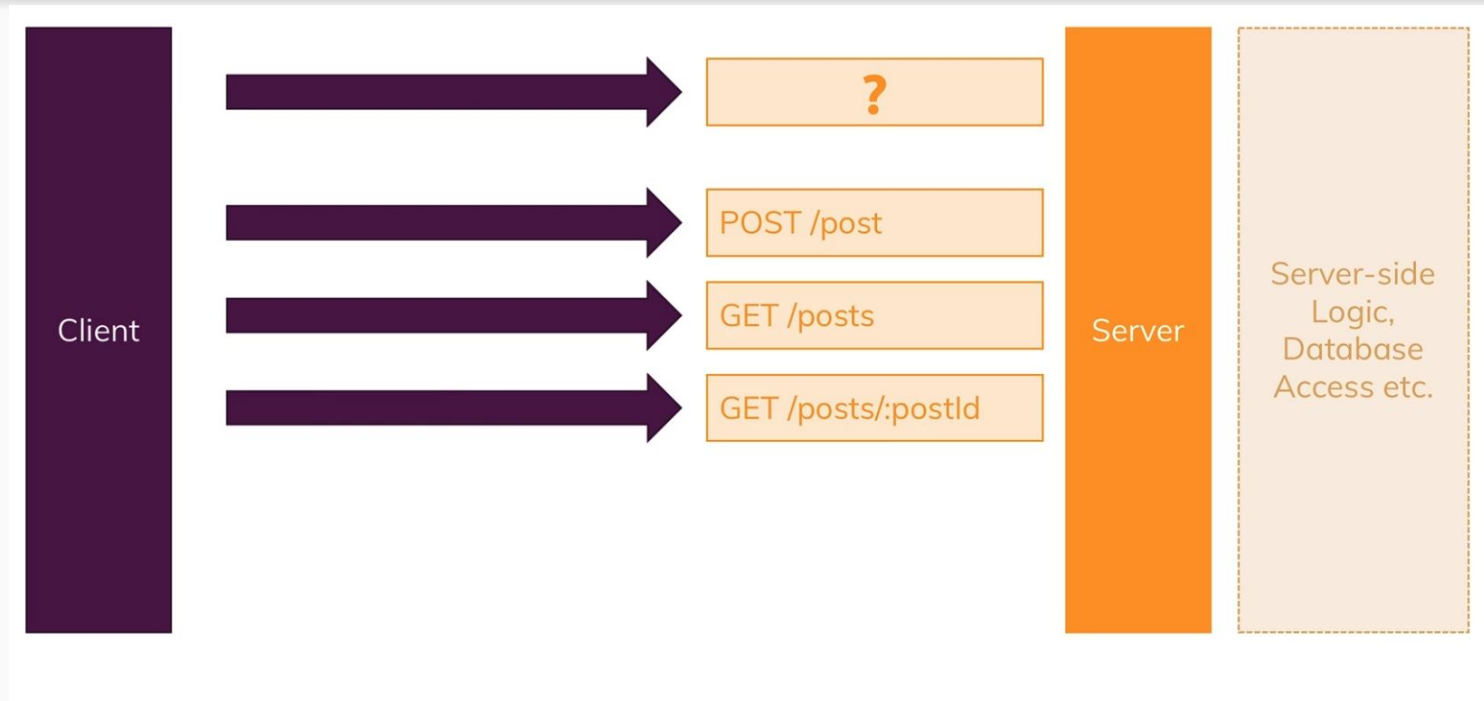
Les APIs REST



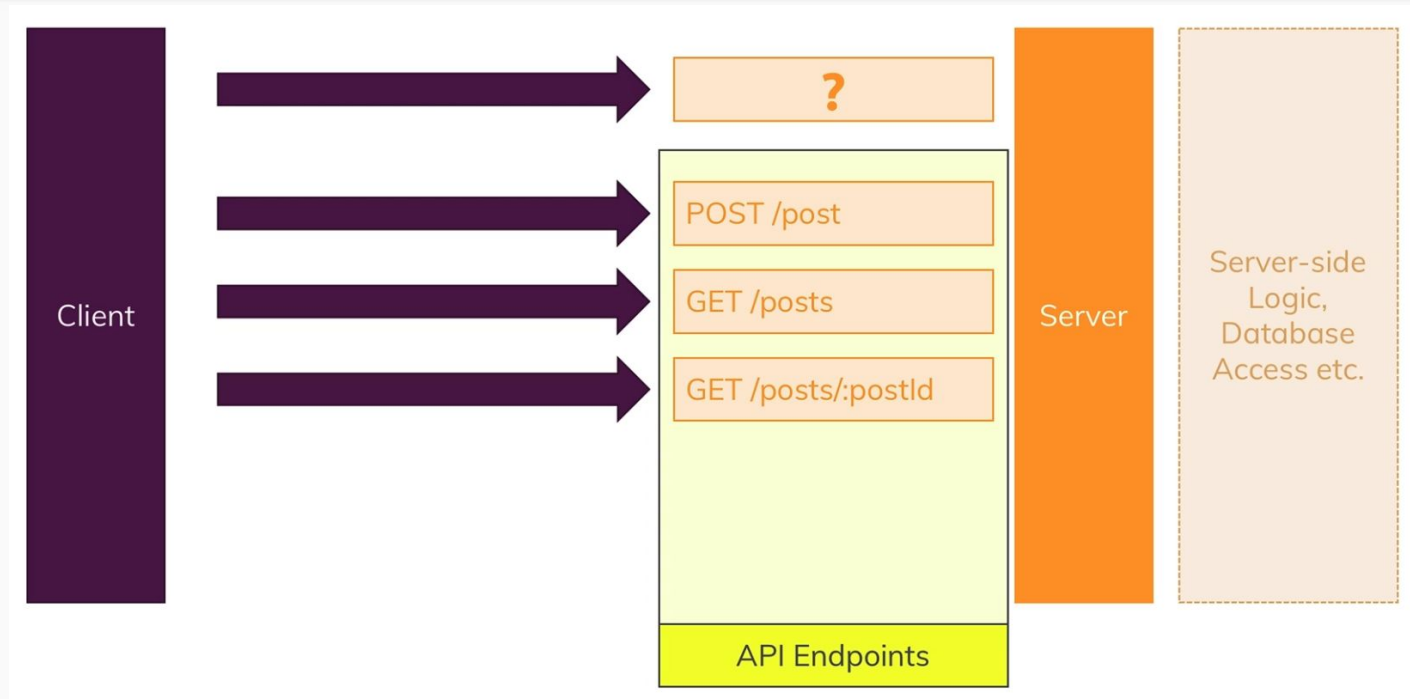
Les APIs REST : les formats de données

HTML	Plain Text	XML	JSON
<code><p>Node.js</p></code>	<code>Node.js</code>	<code><name>Node.js</name></code>	<code>{"title": "Node.js"}</code>
Data + Structure	Data	Data	Data
Contains User Interface	No UI Assumptions	No UI Assumptions	No UI Assumptions
Unnecessarily difficult to parse if you just need the data	Unnecessarily difficult to parse, no clear data structure	Machine-readable but relatively verbose; XML-parser needed	Machine-readable and concise; Can easily be converted to JavaScript

Les APIs REST : Routing



Les APIs REST : Routing et endpoints



Les APIs REST : Méthodes Http

GET

Get a Resource from the Server

POST

Post a Resource to the Server (i.e. create or append Resource)

PUT

Put a Resource onto the Server (i.e. create or overwrite a Resource)

PATCH

Update parts of an existing Resource on the Server

DELETE

Delete a Resource on the Server

OPTIONS

Determine whether follow-up Request is allowed (sent automatically)

Les APIs REST : Fondamentaux

Uniform Interface

Clearly defined API endpoints
with clearly defined request +
response data structure

Notre API doit être :

- Prédicible
- Bien documentée
- Bien structurée

Les APIs REST : Fondamentaux

Stateless Interactions

Server and client don't store any connection history, every request is handled separately

Chaque requête doit être traitée sans "historique", pas de session. Le serveur doit traiter chaque requête indépendamment. Le serveur ne se soucie pas du client. On parle de découplage fort entre Serveur et client.

Les APIs REST : Fondamentaux

Cacheable

Servers may set
caching headers to
allow the client to
cache responses

Une API REST doit pouvoir informer de la durée de validité d'une réponse http, afin que le client puisse la mettre en cache.

Les APIs REST : Fondamentaux

Client-Server

Server and client are separated, client is not concerned with persistent data storage

Séparation Client/Server :
Le client n'a pas à se soucier de la persistance des données

Les APIs REST : Fondamentaux

Layered System

Server may forward
requests to other
APIs

Le serveur doit être capable de “forwarder” la requête à d’autres services sans que le client ne soit au courant.

Création d'une API REST basique

Démo

Création d'une API REST basique

```
HTML
1 <button id="get">Get
  Posts</button>
2 <button
  id="post">Create a
  Post</button>

CSS

JS
1 const getButton = document.getElementById('get');
2 const postButton = document.getElementById('post');
3
4 getButton.addEventListener('click', () => {
5   fetch('http://localhost:8080/feed/posts')
6     .then(res => res.json())
7     .then(resData => console.log(resData))
8     .catch(err => console.log(err));
9 });
```

Get Posts Create a Post

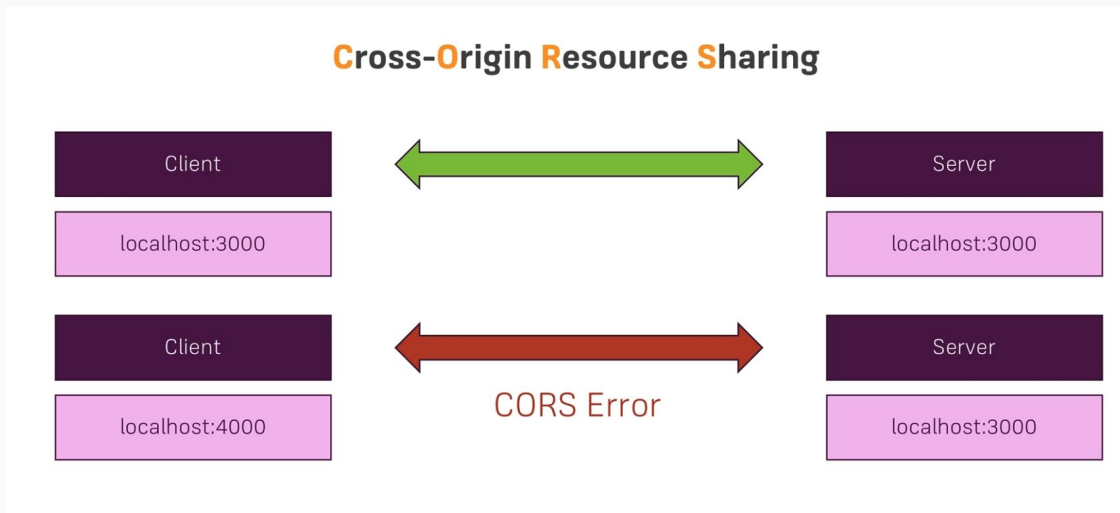
Failed to load <http://localhost:8080/feed/posts> index.html:1
ts: No 'Access-Control-Allow-Origin' header is present on
the requested resource. Origin '<https://s.codepen.io>' is
therefore not allowed access. If an opaque response serves
your needs, set the request's mode to 'no-cors' to fetch
the resource with [CORS](#) disabled.

TypeError: Failed [console_runner-ce303...a462c826d54a73.js:1](#)
to fetch

▶ Cross-Origin Read Blocking (CORB) blocked cross- [pen.js:5](#)
origin response <http://localhost:8080/feed/posts> with MIME
type application/json. See [https://www.chromestatus.com/fea](https://www.chromestatus.com/feature/5629709824032768)
[ture/5629709824032768](https://www.chromestatus.com/feature/5629709824032768) for more details.

Les APIs REST : CORS

Par défaut les CORS ne sont pas autorisées



Les APIs REST : CORS

Par défaut les CORS ne sont pas autorisées

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3
4  const feedRoutes = require('./routes/feed');
5
6  const app = express();
7
8  // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
9  app.use(bodyParser.json()); // application/json
10
11 app.use((req, res, next) => {
12   |   res.setHeader('Access-Control-Allow-Origin', '*');
13   | });
14
15 app.use('/feed', feedRoutes);
16
17 app.listen(8080);
```

Les APIs REST : CORS

Par défaut les CORS ne sont pas autorisées

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3
4  const feedRoutes = require('./routes/feed');
5
6  const app = express();
7
8  // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
9  app.use(bodyParser.json()); // application/json
10
11 app.use((req, res, next) => {
12   res.setHeader('Access-Control-Allow-Origin', '*');
13   res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
14   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
15 });
16
17 app.use('/feed', feedRoutes);
18
19 app.listen(8080);
```

Les APIs REST : CORS

Par défaut les CORS ne sont pas autorisées

```
2  const bodyParser = require('body-parser');
3
4  const feedRoutes = require('./routes/feed');
5
6  const app = express();
7
8  // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
9  app.use(bodyParser.json()); // application/json
10
11 app.use((req, res, next) => {
12   res.setHeader('Access-Control-Allow-Origin', '*');
13   res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
14   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
15   next();
16 });
17
18 app.use('/feed', feedRoutes);
19
20 app.listen(3000);
```

Résumé

REST Concepts & Ideas

- REST APIs are all about data, no UI logic is exchanged
- REST APIs are normal Node servers which expose different endpoints (Http method + path) for clients to send requests to
- JSON is the common data format that is used both for requests and responses
- REST APIs are decoupled from the clients that use them

Requests & Responses

- Attach data in JSON format and let the other end know by setting the “Content-Type” header
- CORS errors occur when using an API that does not set CORS headers

API Rest : Projet