Web Technologies 1

Introduction to back-end web development with Node.js

Maxime Guériau & Alexandre Pauchet

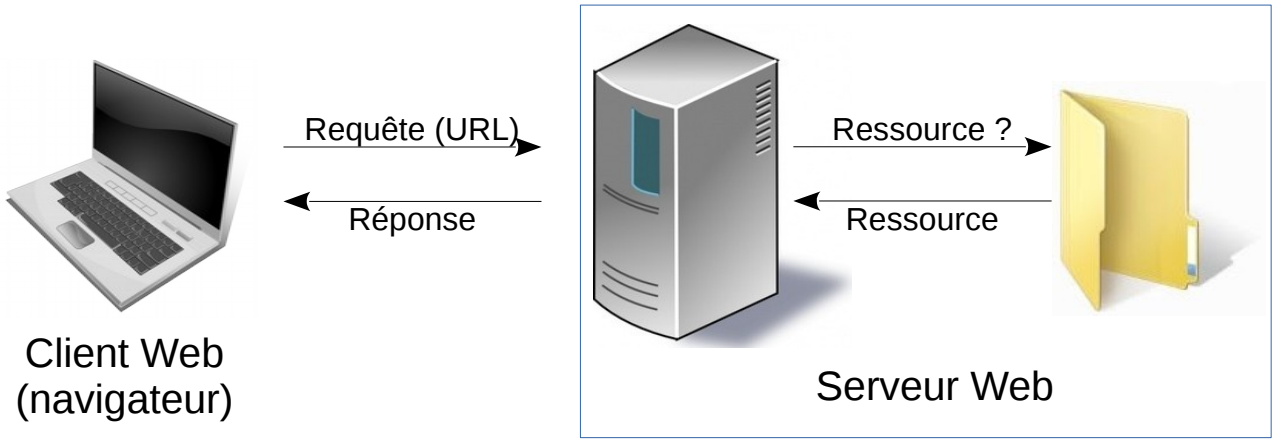
INSA Rouen Normandie - Département ISIA

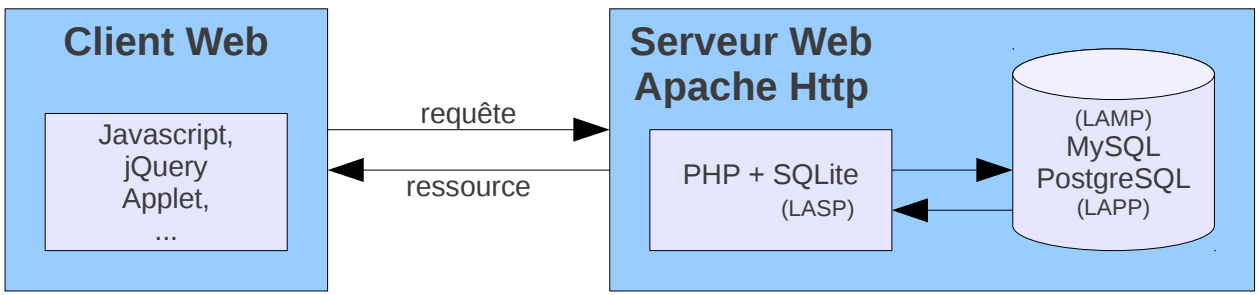
BO.B.RC.18, pauchet@insa-rouen.fr

Plan

Back end, front end, full stack?

Traditional client-server architecture (back to old-fashioned TW1)

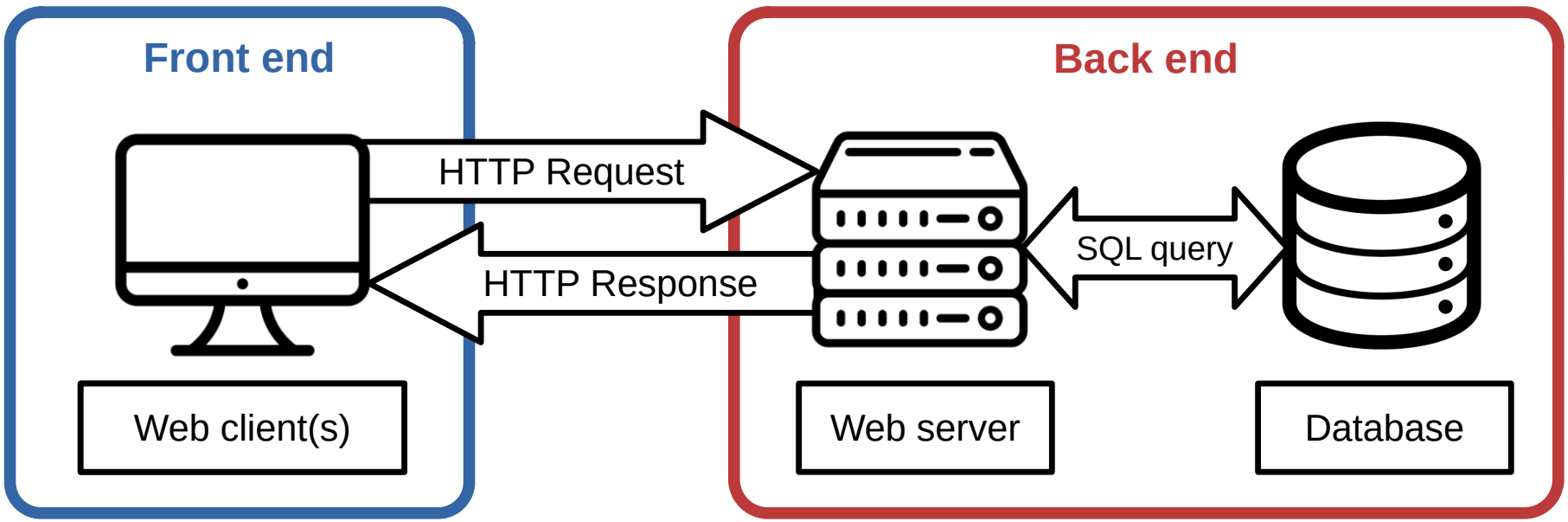




Source: [1]

Back end, front end, full stack?

Front-end *vs.* back-end



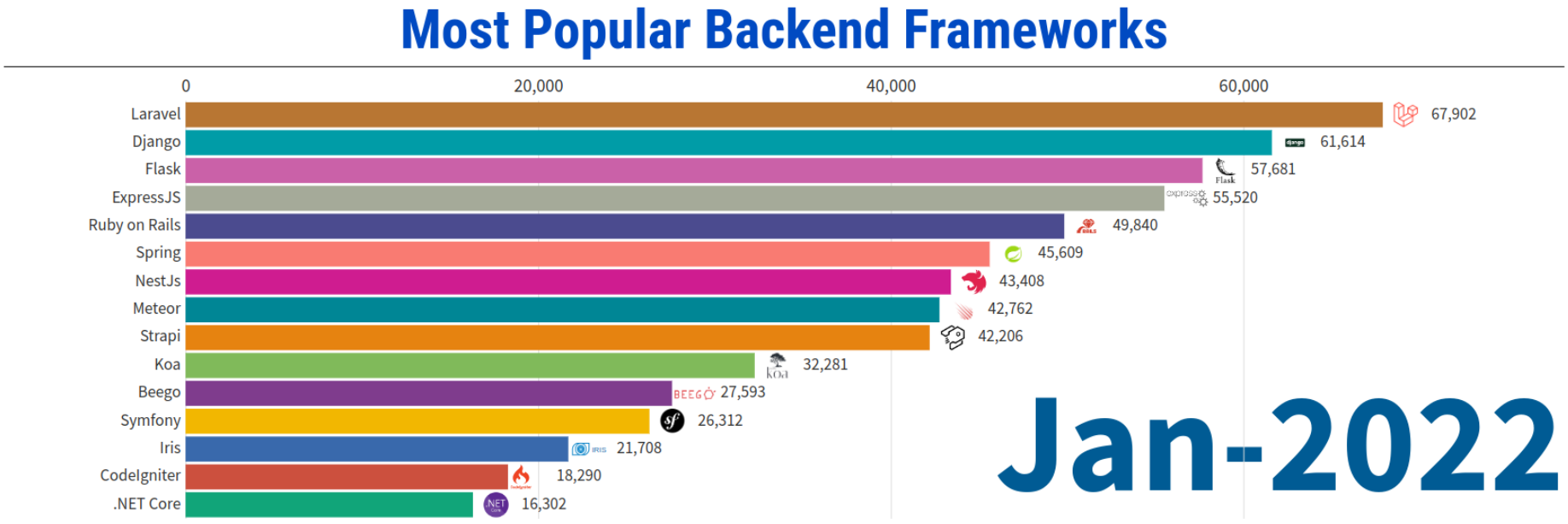
Back end, front end, full stack?

Full stack web development

Source: [2]

Review of current web technologies

Frameworks



Source: [3]

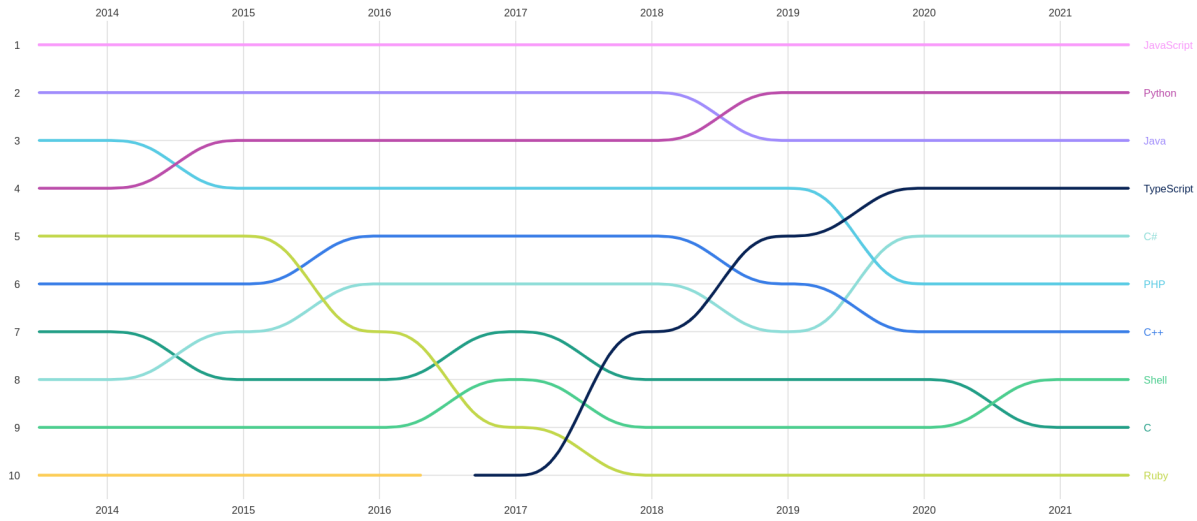
Review of current web technologies

Programming languages

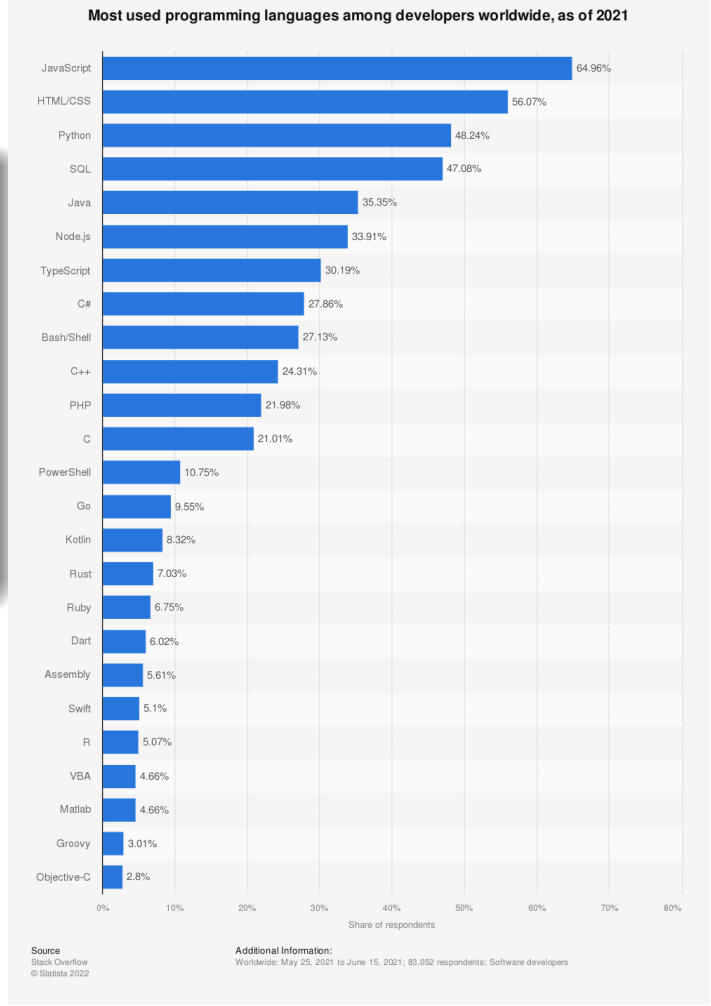
Atwood's Law [4]

"Any application that can be written in JavaScript, will eventually be written in JavaScript."

– Jeff Atwood, Author, Entrepreneur, Cofounder of StackOverflow



Source: [5]



Source: [6]

Node.js
What is Node.js?

Node.js

- Node.js is an **asynchronous event-driven JavaScript runtime**;
- Designed to build **scalable** network applications;
- Particularly suited for **server-side scripting**.
- Provides an **open-source** and **cross-platform environment** based on V8 JavaScript engine [7].
- Comes with `npm` [8, 9], the world's largest **software registry**, a powerful **package manager** and **installer**.

Node.js

Why Node.js? [10]

How a file request is handled by:

PHP, **CGI** or **ASP**

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Node.js

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

✓ **Node.js** eliminates the waiting, and simply continues with the next request, and **runs** (memory efficient) single-threaded, non-blocking, **asynchronous programming**.

Node.js

What can Node.js do? [10]

Node.js can:

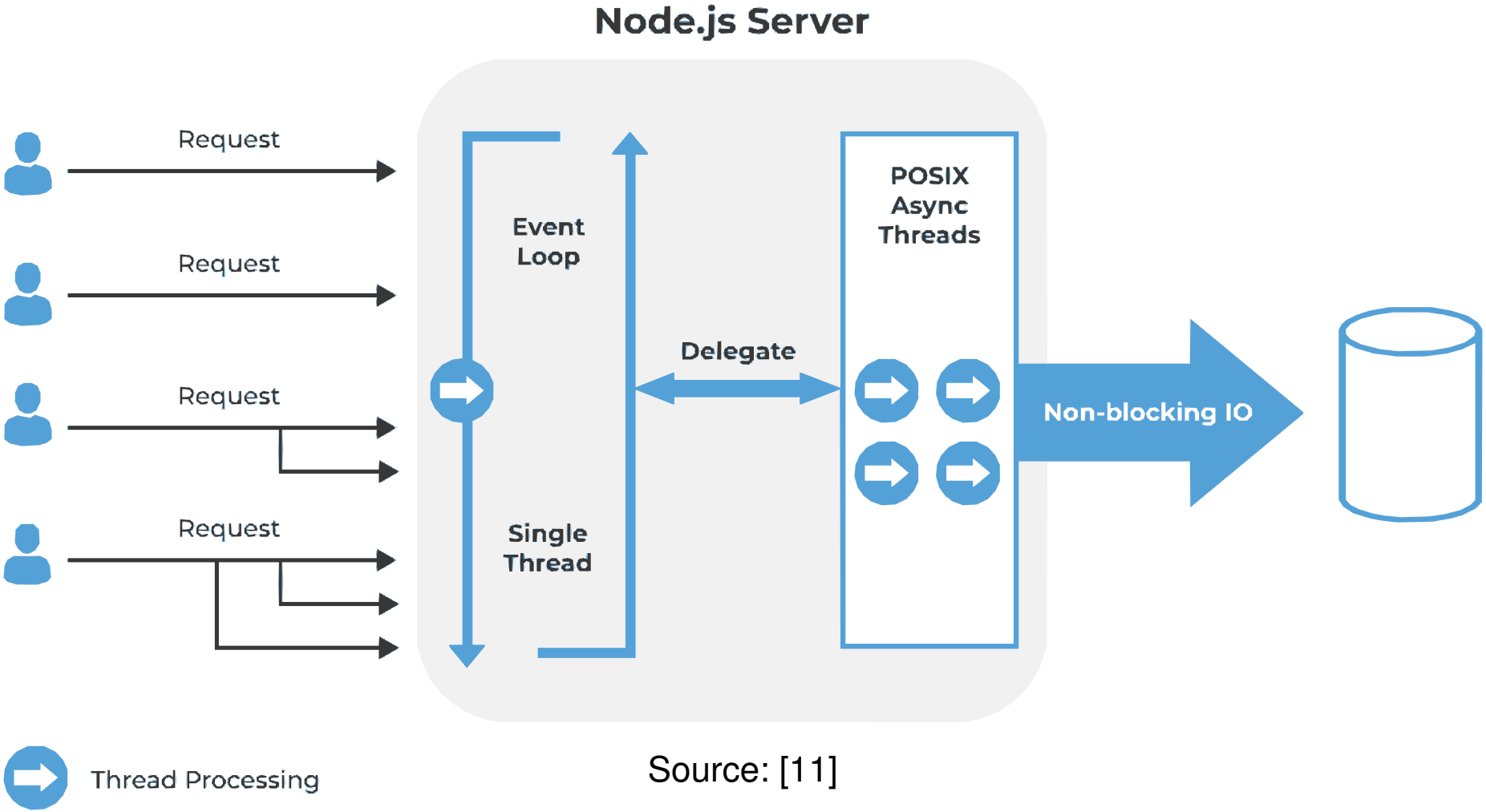
- ✓ generate dynamic page content;
- ✓ create, open, read, write, delete, and close files on the server;
- ✓ collect form data;
- ✓ add, delete, modify data in a database;

through the use of **Node.js files** that:

- have extension `.js`;
- contain tasks that will be executed on certain events (typically someone trying to access a port on the server);
- must be initiated on the server before having any effect;

Node.js

How it works



Source: [11]

Node.js

Event loop [13]

The **Event loop** is what allows Node.js to perform **non-blocking I/O operations**.

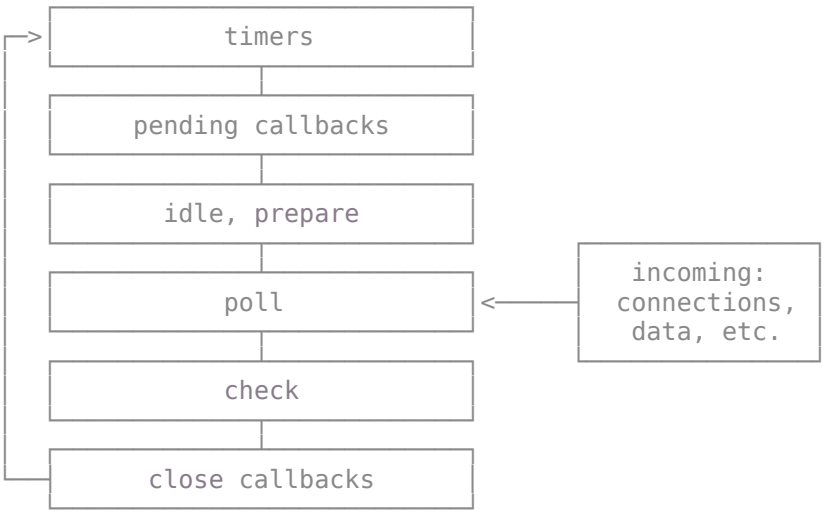
Problem: Javascript is in fact single-threaded (and weird [12] ).

Solution: Offloading operations to the system kernel when possible:

- most modern kernels are multi-threaded;
- callbacks can be executed asynchronously.

Node.js

Phases of the Event loop [13]



Source: [13]

1. `timers`: executes callbacks scheduled by `setTimeout()` and `setInterval()`.
2. `pending callbacks`: executes I/O callbacks deferred to the next loop iteration.
3. `idle, prepare`: only used internally.
4. `poll`: retrieve new I/O events; execute I/O related callbacks (except immediate, close callbacks, and timers).
5. `check`: invoke `setImmediate()` callbacks.
6. `close callbacks`: *e.g.* `socket.on('close')`.

Between each run of œ, Node.js checks if it is waiting for any asynchronous I/O or timers and shuts down cleanly if there are not any.

Node.js

Wait, what's a callback again?

Quick reminder:

Javascript callbacks [14]

- A callback is a function passed as an argument to another function.
- This technique allows a function to call another function.
- A callback function can run after another function has finished.

Asynchronous JavaScript [15])

- Functions running in parallel with other functions are called asynchronous.
- In the real world, callbacks are most often used with asynchronous functions.
- A typical example is JavaScript `setTimeout()`.

Your first Node.js (back-end) server

Installation (on your own setup)

- Install Node.js:

```
sudo apt install nodejs
```

- Check installed version:

```
node -v
```

or

```
node --version
```

- Install `npm`:

```
sudo apt install npm
```

- Check installed version:

```
npm -v
```

or

```
npm --version
```

Your first Node.js (back-end) server

Get started

- Create a directory:

```
mkdir backend
cd backend
```

- Initialize a new Node.js project in this folder:

```
npm init
```

Choose the name of your **main entry point** file (example: `server.js`)

- (optional) Initialize a git repository

```
git init
```

Important: add the `node_modules` directory to your `.gitignore` file!

- Finally, create your main entry point file:

```
touch server.js
```

Your first Node.js (back-end) server

Server example: NodeJSMini/server.js (adapted from [16])

```
1  // import http module; documentation: https://nodejs.org/api/http.html
2  const http = require('http');
3
4  // set the server host and port
5  const hostname = '127.0.0.1';
6  const port = 3000;
7
8  // create a http server endpoint
9  const server = http.createServer((req, res) => {
10       // set the response of your endpoint
11       res.end('Hello World!');
12 });
13
14 // run the server
15 server.listen(port, hostname, () => {
16       // callback executed when the server is launched
17       console.log(`Server running at http://${hostname}:${port}/`);
18 });
```

Your first Node.js (back-end) server

Start your server!

- To start your server, simply run:

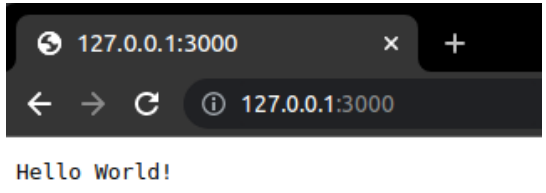

```
node server
```

or

```
node server.js
```

output:

```
Server running at http
    ://127.0.0.1:3000/
```



- alternatively you can also install `nodemon` module [17]:

globally:

```
npm install -g
    nodemon
```

or

as a development dependency:

```
npm install --save-dev
    nodemon
```

to restart your server automatically.

Your first Node.js (back-end) server

Managing dependencies [18]

When you upload NodeJS code to a GitHub repository (or any code repository), you should **not** upload the `node_modules` directory:

- You shouldn't be modifying code in the `node_modules` directory, so there's no reason to have it under version control
- This will also increase your repo size significantly

Q: But if you don't upload the node_modules directory to your code repository, how will anyone know what libraries they need to install?

Your first Node.js (back-end) server

Managing dependencies [18]

If we don't include the `node_modules` directory in our repository, we need to somehow tell other people what `npm` modules they need to install.

`npm` provides a mechanism for this: `package.json`

Your first Node.js (back-end) server

Managing dependencies [18]

You can put a file named `package.json` in the root directory of your NodeJS project to specify metadata about your project.

Create a `package.json` file using the following command:

```
$ npm init
```

This will ask you a series of questions then generate a `package.json` file based on your answers.

Your first Node.js (back-end) server

Managing dependencies [18]

Now when you install packages, you should pass in the --save parameter:

```
$ npm install --save express
$ npm install --save body-parser
```

This will also add an entry for this library in package.json.

```
"dependencies": {
  "body-parser": "^1.17.1",
  "express": "^4.15.2"
},
```

Your first Node.js (back-end) server

Managing dependencies [18]

If you remove the node_modules directory:

```
$ rm -rf node_modules
```

You can install your project dependencies again via:

```
$ npm install
```

- This also allows people who have downloaded your code from GitHub to install all your dependencies with one command instead of having to install all dependencies individually.

Your first Node.js (back-end) server

Managing dependencies [18]

Your package.json file also defines scripts:

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node server.js"
},
```

You can run these scripts using `$ npm` ***scriptName***

E.g. the following command runs "node server.js"

```
$ npm start
```

Your first Node.js (back-end) server

Managing dependencies: fixing `nodemon`

An other example of dependency is `nodemon` module, that you can install as a local development dependency:

```
npm install --save-dev nodemon
```

Doing so will add a few specific lines in your `package.json`:

```
...
"devDependencies": {
    "nodemon": "^2.0.20"
},
...
```

You can not use `nodemon` directly as a command line but you can update your `package.json` so `npm` can now use `nodemon`:

```
...
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon server.js"
},
...
```

Your first Node.js (back-end) server

Managing dependencies

Example of `package.json` (auto-generated + manually updated):

```
{
    "name": "nodejsmini",
    "version": "1.0.0",
    "description": "",
    "main": "server.js",
    "scripts": {
        "test": "echo \"Error: no test specified\",
        "start": "nodemon server.js"
    },
    "author": "Alexandre Pauchet",
    "license": "ISC",
    "dependencies": { },
    "devDependencies": {
        "nodemon": "^3.1.14"
    }
}
```

Going further with Node.js
Node.js Globals

Node.js Globals

Node.js uses objects that are available everywhere; they are called the **Globals**:

- `__dirname`: stores the path of the current folder.
- `__filename`: returns the name of the file being executed.
- `require()`: function that allows to load modules.
- `module`: returns info on the current module.
- `process`: returns info about the current environment.

Going further with Node.js

HTML server example: NodeJSHTML/server.js

...

```
8  // create a http server endpoint
9  const server = http.createServer((req, res) => {
10         // response status code (200 = OK)
11         res.statusCode = 200;
12         // response header
13         res.setHeader('Content-Type', 'text/html');
14         // set the response of your endpoint
15         res.end('<!DOCTYPE html>
16                  <html>
17                      <body>
18                          <h1>Hello World! (but it's in HTML)</h1>
19                      </body>
20                  </html>');
21 });
```

...

Going further with Node.js

Handling GET requests: NodeJSGET/server.js (adapted from [19])


```
3  // import url module; documentation: https://nodejs.org/api/url.html
4  const url = require('url');

   ...

11 const server = http.createServer((req, res) => {
12
13      // read and parse the URL
14      const queryObject = url.parse(req.url, true).query;

   ...

21      // set the response of your endpoint
22      if ('name' in queryObject) {
23          res.end(`Hey ${queryObject.name}!`);
24      } else {
25          res.end("Hey you!");
26      }
27 });
```

Going further with Node.js

Routes and static files: NodeJSGET/server.js (adapted from [19])

```
3 const Static = require('node-static');
...
3 const fileServer = new Static.Server('./public');
...
3 const server = http.createServer((req, res) => {
4   if (req.method === 'GET') {
5     res.statusCode = 200;
6     res.setHeader('Content-Type', 'text/plain');
7     const queryObject = url.parse(req.url, true).query;
8     if (req.url === '/marco')
9       { res.end("polo"); }
10    if (req.url.startsWith('/hello')) {
11      if ('name' in queryObject) {
12        res.end(`Hello ${queryObject.name}!`);
13      } else {
14        res.end("Hello you!");
15      }
16    }
17    return;
18  }
19  fileServer.serve(req, res);
20 });
```

Going further with Node.js

Reading POST data: NodeJSPOST/backend/server.js (adapted from [25])

```
3  // import querystring module
4  const querystring = require('querystring');
5  // import node-static module; documentation: https://www.npmjs.com/package/node-static
6  const static = require('node-static');

   ...

14 const server = http.createServer((req, res) => {
15         // response header
16         res.setHeader('Content-Type', 'text/plain');
17         // check if we received POST data
18         if(req.method === 'POST') {
19                 return post(req,res);
20         } else {
21                 let file = new(static.Server)(__dirname+ "/../frontend/index.html");
22                 file.serve(req, res);
23         }
24 }).listen(port, hostname, () => {
25         // callback executed when the server is launched
26         console.log(`Server running at http://${hostname}:${port}/`);
27 });
```

Going further with Node.js

Reading POST data: NodeJSPOST/backend/server.js (adapted from [25])

```
29 function post(req, res){
30     // aggregate chunks of data
31     let input = "";
32     req.on("data", (chunk) => {
33         input += chunk.toString();
34     });
35     // when the last chunk is received
36     req.on("end", () => {
37         // parse the data
38         let queryObject = querystring.parse(input);
39         // set the response of your endpoint
40         if ('name' in queryObject && queryObject.name!="") {
41             res.statusCode = 200;
42             res.end(`Hey ${queryObject.name}!`);
43         } else {
44             res.statusCode = 400;
45             res.end("POST parameters error!");
46         }
47     });
48 };
```

Going further with Node.js

Reading/writting files: Compteurs/index.js (adapted from [25])

```
3  const http = require('http');
4  const fs = require('fs');
5
6  const hostname = '127.0.0.1';
7  const port = 3000;
8
9  function lire_hits() {
10   try {
11     return(parseInt(fs.readFileSync('hits.txt', 'utf8')));
12   } catch (err) {
13     console.error(err);
14   }
15 }
16
17 function ecrire_hits(hits) {
18   fs.writeFile('hits.txt', hits.toString(), function (err) {
19     if (err)
20       console.log('Problème écriture de fichier');
21   });
22 }
```

Going further with Node.js

Reading/writting files: Compteurs/index.js (adapted from [25])


```
29
30 const server = http.createServer((req, res) => {
31     res.setHeader('Content-Type', 'text/plain; charset=utf-8');
32     res.statusCode = 200;
33     let hits = lire_hits();
34     hits++;
35     ecrire_hits(hits);
36     res.end(`Nombre de connexions: ${hits}`);
37   });
38
39 server.listen(port, hostname, () => {
40   console.log(`Server listening on port ${port}`);
41 });
```

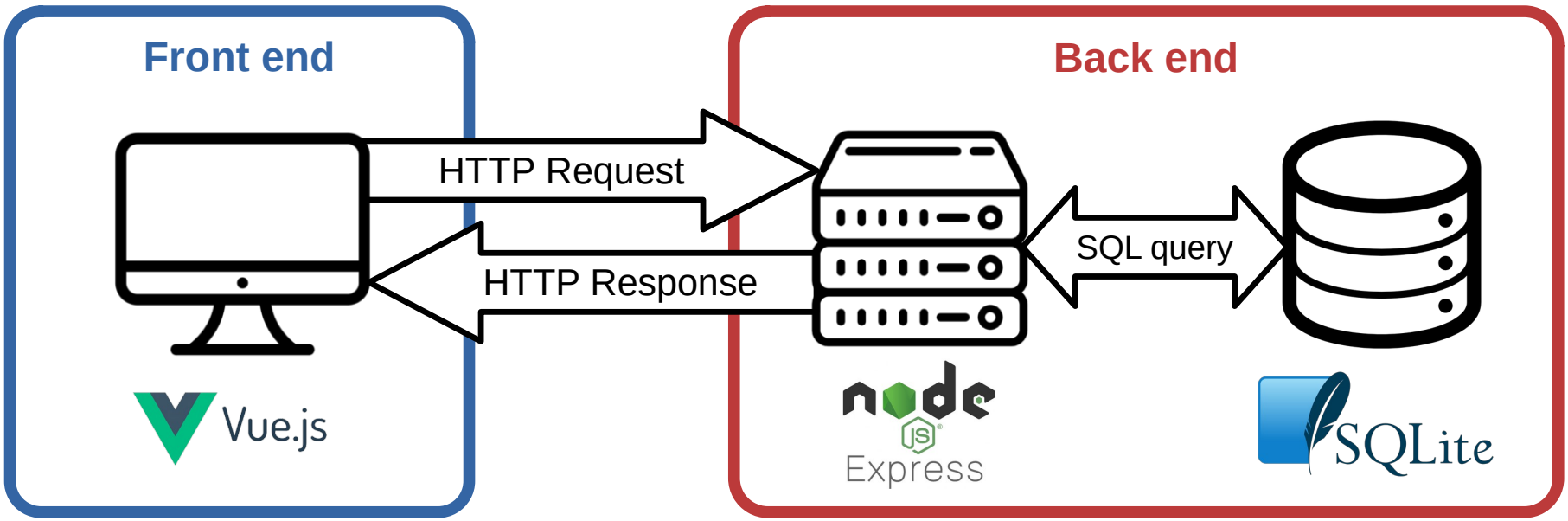
Conclusion

Key takeaways:

- A full-stack web developer is a person who can develop both client and server software.
- This course introduces modern ways to program and interconnect: (1) a browser, (2) a server, and (3) a database.
- The back end is the data access layer and the software infrastructure hosted on the web server.
- Node.js provides an asynchronous runtime able to answer to multiple requests
- ... while performing non-blocking I/O operations.
- Node.js takes advantages of JavaScript callbacks and asynchronous functions.
- Running a simple Node.js server is a piece of cake ...
- ... but the use of frameworks is highly recommended for large-scale applications/services.

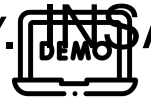
Conclusion

This is the plan for TW2:



References and further reading/watching I

[1] Alexandre Pauchet. *Techologies Web 2 – AJAX/JQuery*. INSA Rouen - Département ITI, 2021.



[2] What hiring managers look for in a full stack developer. URL `https://www.cybercoders.com/insights/what-hiring-managers-look-for-in-a-full-stack-dev`

[3] Most popular backend frameworks 2012-2022. URL `https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2022/`.

[4] Rethinking atwood's law. URL `https://jayaprabhakar.medium.com/rethinking-atwoods-law-64a894b54aa4`.

[5] Top languages over the years (github). URL `https://octoverse.github.com/#top-languages-over-the-years`.

References and further reading/watching II

[6] Most used programming languages among developers worldwide as of 2021. URL https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/.

[7] What is v8? URL https://v8.dev/.

[8] What is npm? (w3c), . URL https://www.w3schools.com/whatis/whatis_npm.asp.

[9] What is npm? (node.js), . URL https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/.

[10] Node.js introduction, . URL https://www.w3schools.com/nodejs/nodejs_intro.asp.

References and further reading/watching III

[11] How javascript is quickly becoming the market leader. URL
https://www.mobilelive.ca/blog/javascript-leader#:~:text=According%20to%20the%20latest%20survey,integrated%20with%20other%20frameworks%2Flanguages.

[12] Javascript is weird (extreme edition). URL
https://youtu.be/sRWE5tnaxlI.

[13] The node.js event loop, timers, and process.nexttick(). URL
https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/#:~:text=The%20event%20loop%20is%20what,operations%20executing%20in%20the%20background.

[14] Javascript callbacks, . URL
https://www.w3schools.com/js/js_callback.asp.

References and further reading/watching IV

[15] Asynchronous javascript, . URL https://www.w3schools.com/js/js_asynchronous.asp

[16] Go full-stack with node.js, express, and mongodb. URL https://openclassrooms.com/fr/courses/5614116-go-full-stack-with-node-js-express-and-mo

[17] nodemon, . URL https://www.npmjs.com/package/nodemon.

[18] Victoria Kirst. *CS193X Web Programming Fundamentals*, 2017. URL https://web.stanford.edu/class/archive/cs/cs193x/cs193x.1176/.

[19] How to access query string parameters, . URL https://nodejs.org/en/knowledge/HTTP/clients/how-to-access-query-string-parameters/.

[20] Node.js official documentation, . URL https://nodejs.org/en/docs/.

References and further reading/watching V

[21] Introduction to node.js, . URL https://nodejs.dev/learn.

[22] June 2022 web server survey. URL https://news.netcraft.com/archives/2022/06/30/june-2022-web-server-survey.html.

[23] Front end vs back end. URL https://xyzcoding.com/course/the-internet/how-the-internet-works/front-end-vs-back-end/.

[24] Philip roberts: What the heck is the event loop anyway?). URL https://youtu.be/8aGhZQkoFbQ.

[25] Getting post parameters in node.js. URL https://usefulangle.com/post/93/nodejs-post-parameters.

[26] Routing in node.js. URL https://www.geeksforgeeks.org/routing-in-node-js/.

[27] The node.js fs module. URL https://nodejs.dev/learn/the-nodejs-fs-module.

References and further reading/watching VI

[28] Reading files with node.js, . URL `https://nodejs.dev/learn/reading-files-with-nodejs`.

[29] Writing files with node.js, . URL `https://nodejs.dev/learn/writing-files-with-nodejs`.