Web Technologies 1

Lecture 1. Introduction to back-end web development with Node.js

Maxime Guériau & Alexandre Pauchet

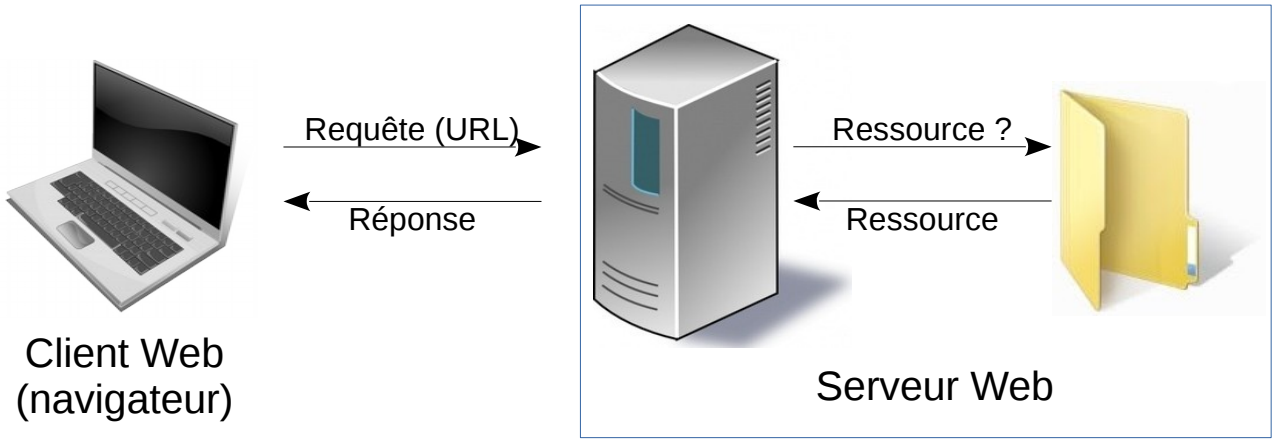
INSA Rouen - Département ASI

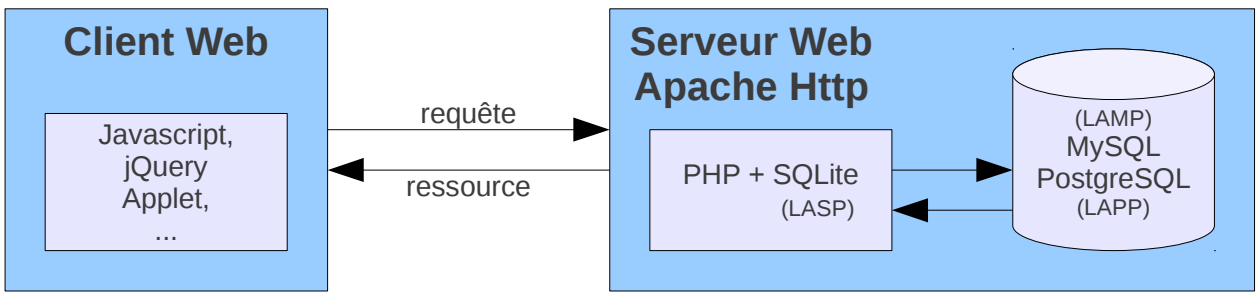
BO.B.RC.18, pauchet@insa-rouen.fr

Plan

Back end, front end, full stack?

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

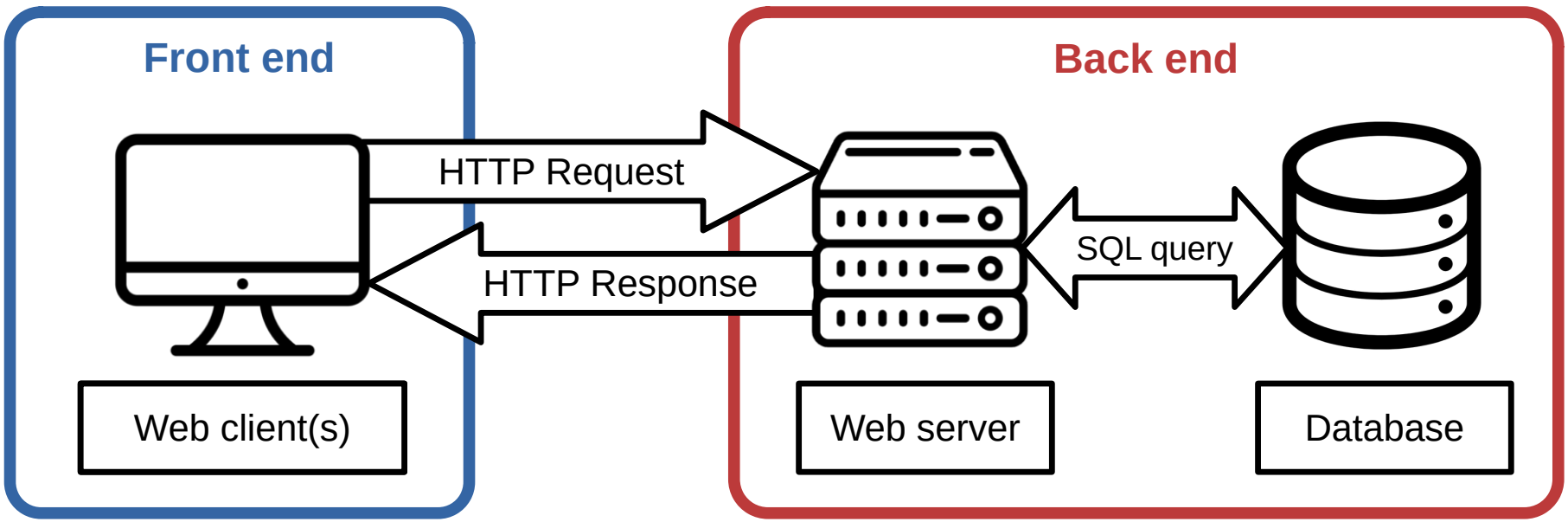




Source: [1]

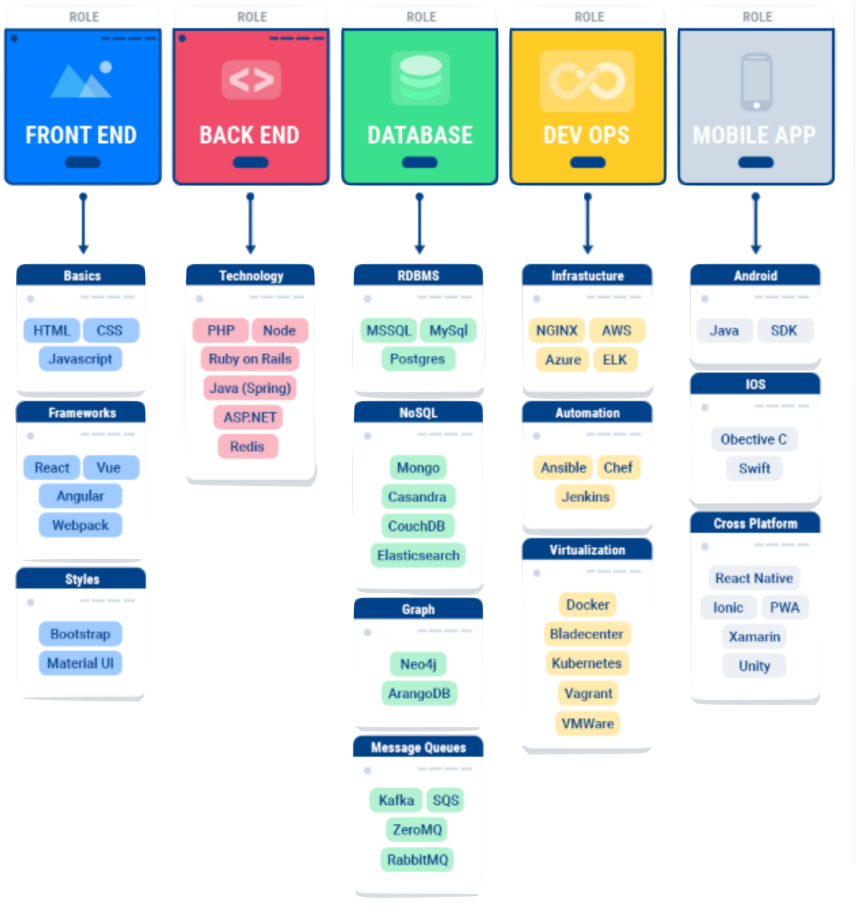
Back end, front end, full stack?

Front-end *vs.* back-end



Back end, front end, full stack?

Full stack web development



Source: [2]

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 [3].
- Comes with `npm` [4, 5], the world's largest **software registry**, a powerful **package manager** and **installer**.

Node.js

Why Node.js? [6]

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

Why Node.js? [6]

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? [6]

Node.js can:

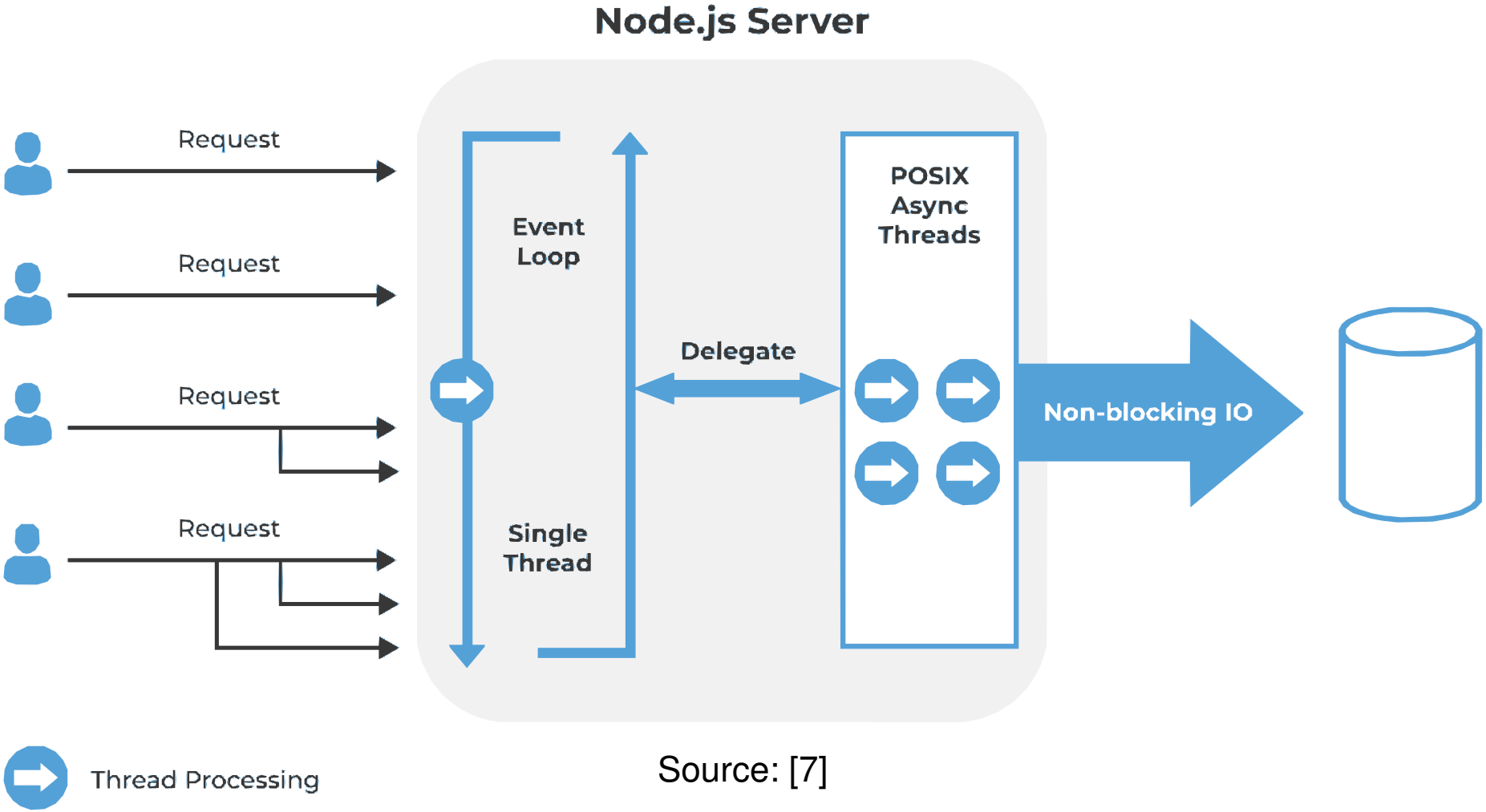
- ✓ generate dynamic page content;
- ✓ create, open, read, write, delete, and close files on the server;
- ✓ collect form data;
- ✓ add, delete, modify data in a database;

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: [7]

Node.js

Event loop [9]

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

Problem: Javascript is in fact single-threaded (and weird [8] 🎞).

Node.js

Event loop [9]

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

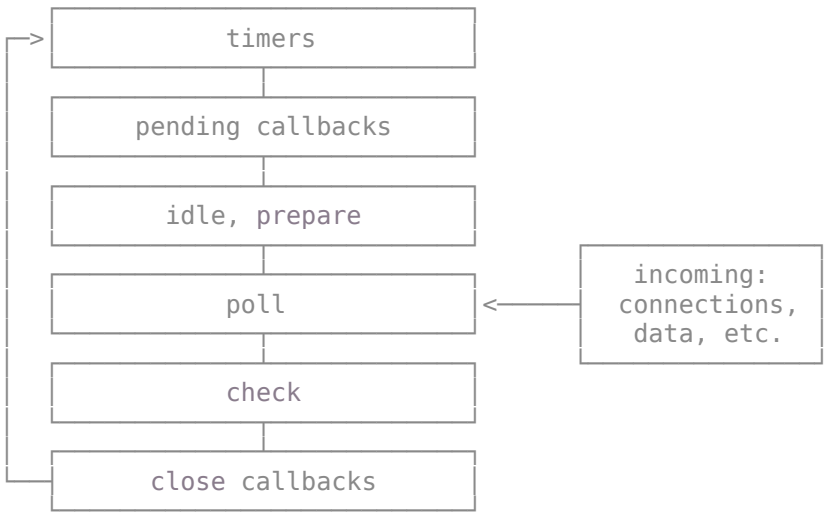
Problem: Javascript is in fact single-threaded (and weird [8] 🎞).

Solution: Offloading operations to the system kernel whenever possible:

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

Node.js

Phases of the Event loop [9]



Source: [9]

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 the event loop, 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 [10]

- 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 [11])

- 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 [12])

```
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:

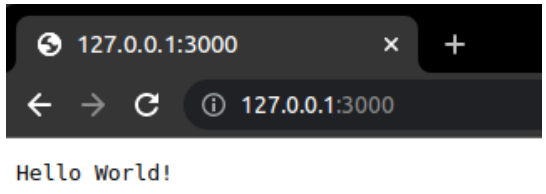

```
node server
```

or

```
node server.js
```

output:

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



- alternatively you can also install `nodemon` module [13]:

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

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.

Your first Node.js (back-end) server

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 });
```

...

Your first Node.js (back-end) server

Handling GET requests: NodeJSGET/server.js (adapted from [14])


```
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 });
```

Your first Express.js (back-end) server (app)

NodeJSMini/server.js (adapted from [12])

```
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}/`);
```

Your first Express.js (back-end) server (app)

NodeJSMini/server.js (adapted from [12])

```
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 });
```

ExpressJSMini/server.js (from [?])


```
1  // import express module and create your
       express app
2  const express = require('express');
3  const app = express();
4
5  // set the server host and port
6  const port = 3000;
7
8  // create your express server endpoint
9  app.get('/', function (req, res) {
10     // set the response of your endpoint
11     res.send('Hello World!');
12 });
13
14 // run the server
15 app.listen(port, () => {
16     // callback executed when the server is
        launched
17     console.log(`Express app listening on port
        ${port}`);
18 });
```

Your first Express.js (back-end) server (app)

Installation [?]

Express is not part of the NodeJS APIs.

If we try to use it, we'll get an error:

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

```
module.js:327
    throw err;
    ^

Error: Cannot find module 'express'
    at Function.Module._resolveFilename
```

We need to install Express via npm.

Your first Express.js (back-end) server (app)
Installation

- To install Express, simply run:

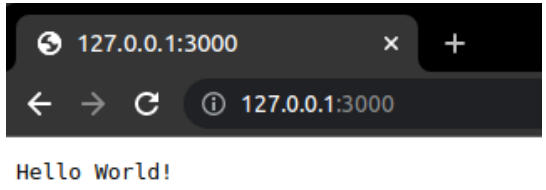
```
npm install express
```

- and then run your (express app) server:

```
node server.js
```

output:

```
Express app listening on port 3000
```



Your first Express.js (back-end) server (app)

Hello world example [?]

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

The `require()` lets us load the ExpressJS module.

The module actually contains a function that creates a new Express Application object.

Your first Express.js (back-end) server (app)

Hello world example [**?**]

```
app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
})
```

The ExpressJS `listen()` is identical to the NodeJS `listen()` function:

- This binds the server process to the given **port number**.
- Now messages sent to the OS's port 3000 will be routed to this server process.
- The function parameter is a callback that will execute when it starts listening for HTTP messages (when the process has been bound to port 3000)

Your first Express.js (back-end) server (app)

Serving static files [?]

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

app.use(express.static('public'));

app.get('/', function (req, res) {
  res.send('Main page!');
});
```

This line of code makes our server now start serving the files in the 'public' directory directly.

Your first Express.js (back-end) server (app)

Example: ExpressJSstatic


```
/ExpressJSstatic
├── /node_modules
├── package.json
├── /public
│   ├── index.html
│   └── resource.txt
└── server.js
```

ExpressJSstatic/server.js

```
...
5 // enable your express app to serve static files located in
      ”public” directory
6 app.use(express.static('public'));
...
```

Your first Express.js (back-end) server (app)

Managing dependencies [**?**]

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 Express.js (back-end) server (app)

Managing dependencies [**?**]

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 Express.js (back-end) server (app)

Managing dependencies [?]

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 Express.js (back-end) server (app)

Managing dependencies

Example of an auto-generated `package.json`:

```
{
    "name": "expressjsstatic",
    "version": "1.0.0",
    "description": "",
    "main": "server.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" &&
exit 1",
        "start": "node server.js"
    },
    "author": "Alexandre Pauchet",
    "license": "ISC",
    "dependencies": {
        "express": "^4.18.1"
    }
}
```

Your first Express.js (back-end) server (app)

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

// set the server host and port
const hostname = '127.0.0.1';
const port = 3000;
```

...

```
// run the first server
server.listen(port, hostname, () => {
        // callback executed when the server is launched
        console.log(`Server running at http://${hostname}:${port}/`);
});

// import express module and create your express app
const express = require('express');
const app = express();
// enable your express app to serve static files located in "public" directory
app.use(express.static('public'));
app.listen(3030);
console.log(`Express running at http://${hostname}:3030/`);
```

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

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

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

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

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.

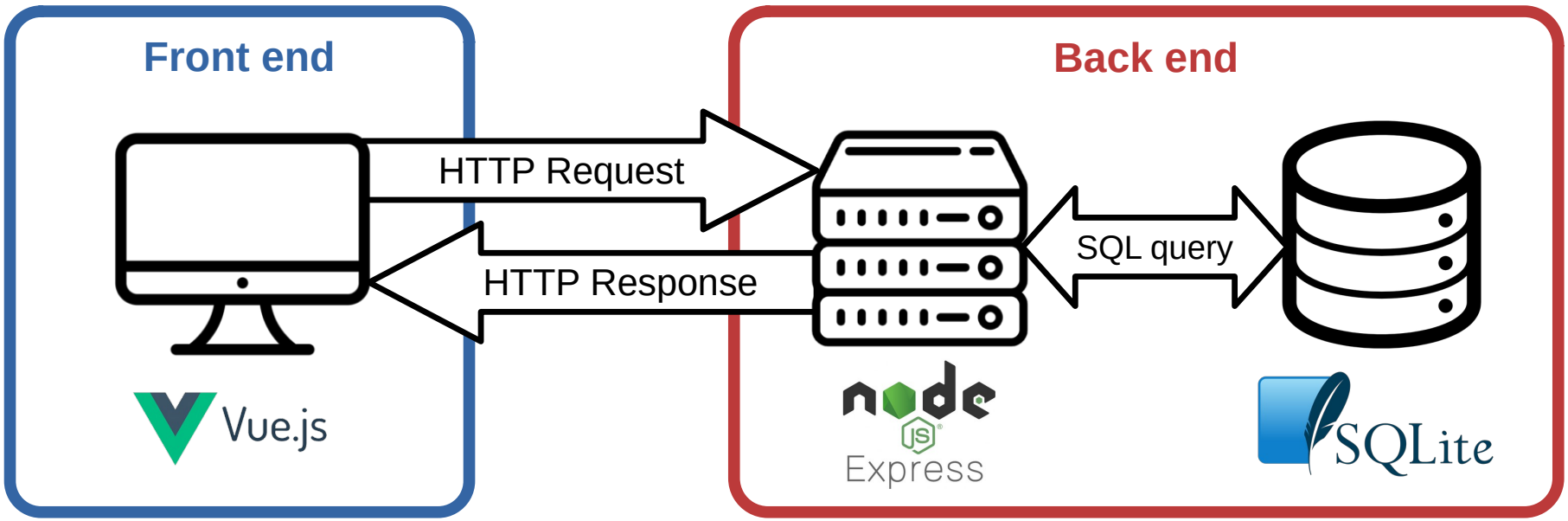
Conclusion

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:



Questions?

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] What is v8? URL `https://v8.dev/`.

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

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

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

References and further reading/watching II

[7] 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.

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

[9] 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.

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

References and further reading/watching III

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

[12] 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

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

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

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

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

References and further reading/watching IV

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

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

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

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

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

References and further reading/watching V

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

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

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

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

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

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

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