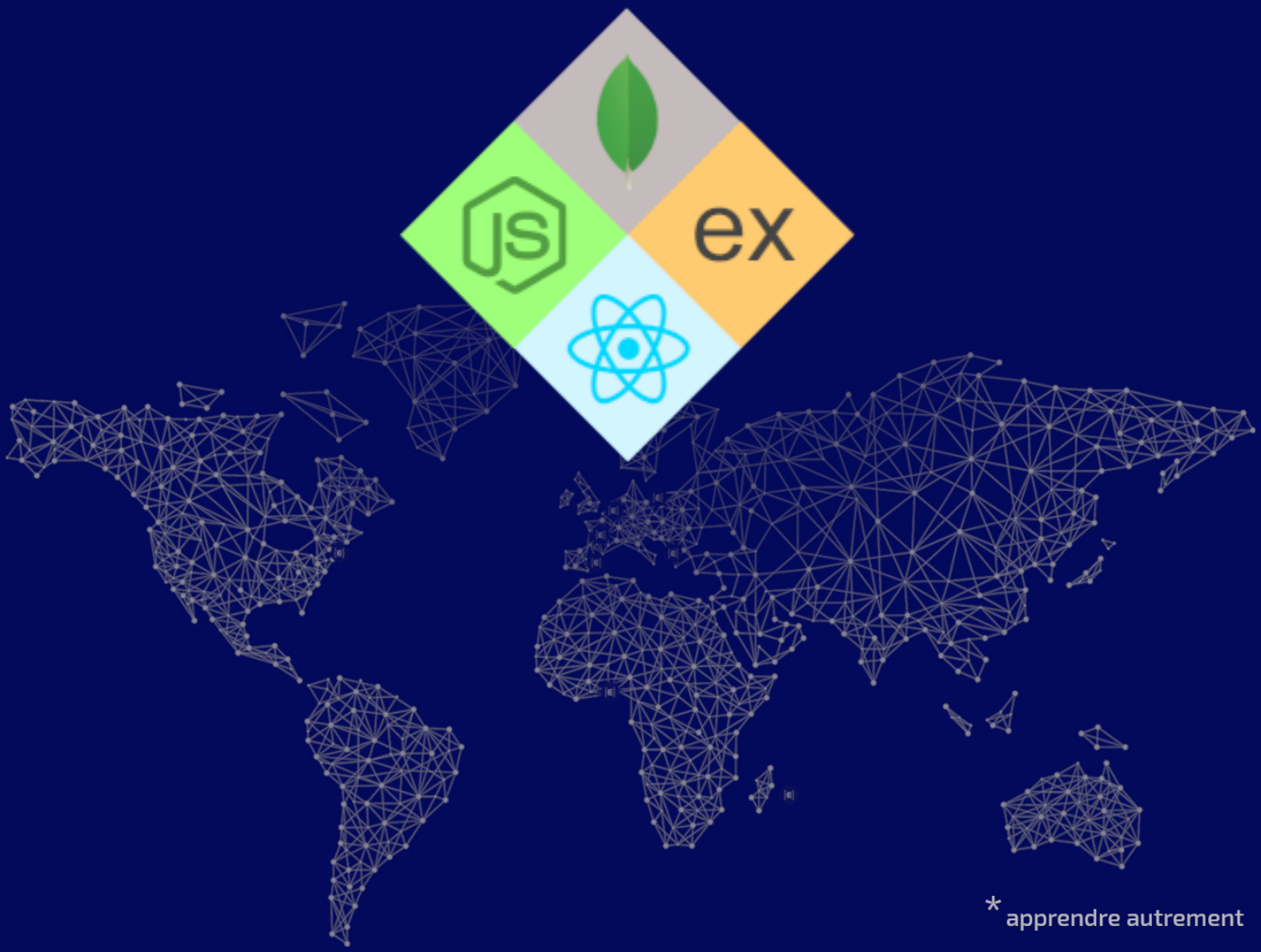
{EPITECH}
LEARN DIFFERENT*

# IRC

## BOOTSTRAP - NODE JS & EXPRESS JS




* apprendre autrement

# IRC

## Step 0

First of all, check out the Node.js documentation.

As you will see, there is a lot to read and to learn. It can be hard at times.

Getting a good understanding of Node will be a good thing for your career as a developer. Node is a very powerful and widely used tool.

For this project, synchronous functions are forbidden.
Node's strength lies in asynchronous functions. Let's do things in the Node way!

In Node, callbacks can quickly make your code difficult to read and to review.

It is relevant to organize your code into modules in order to encapsulate functionalities.

## Step 1

Using the **HTTP module** (asynchronous), create a **server module** that starts a server on a given port (passed as parameter to a *start* method).

Each time a client connects, the server must write `Client connected` on the standard output.
It must also write `Client disconnected` upon disconnection.

Since you are polite, your server should greet each new client by sending them:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF8"/>
    <title>Welcome</title>
  </head>
  <body>
    <p>Greetings $NAME!</p>
  </body>
</html>
```

This code must be found in the `index.html` file and retrieved using Node's **File System module**.

The `$NAME` parameter is obtained from the URL (GET).

For example, display `Greetings Martin!` to the user visiting the page `localhost:port?name=Martin`.

If no `name` parameter is found in the URL, the message should be `Greetings whoever you are!`.

`URL` module.

Think about how you are going to replace the `$NAME` variable in the HTML code read from `index.html`. It is just a matter of text manipulation.

## Step 2

There is an ugly way to add more pages: using `if`s and `else`s. But not for you!
Other developers will thank you for your cleanliness; you will even thank yourself.

Provide a way for your server to differentiate between the different URLs it receives requests for, and generate custom responses for each of these specific URLs.

To achieve this, you are going to implement a router.

Your router must check if the URL is found in the keys of an associative array (map), and if the corresponding value is a function:

- ✓ if so, it will be called with a request object and a response object ;
- ✓ otherwise, your router will respond with the error status code 404, and a plain text body simply stating `404 error: Page not found.`.

To implement the custom responses, create a `pages` module that contains methods to display specific pages by receiving as arguments a `request` object and a `response` object.

Of course, in each case the headers sent by your server must have a status code of `200` and a `Content-Type` indicated as `text/html`.

Modify the `start` method of your server so that it takes as parameters a port, your `route` method and an associative array handle. When a new request is received, your server will call `route` with the right arguments.

The associative array must contain paths as keys and your pages' methods as values. Obviously this array must be defined only once, in one place.

## Step 3

Just a tiny bit more advanced: handling POST requests.

First create a small form:

- ✓ in `form.html` ;
- ✓ submitting to `/index.html`;
- ✓ with an `input` field named `name` ;
- ✓ and a `Submit` button.

Modify whatever should be to display your form on `/form.html`.

If you hit the submit button, you are redirected to the index page. This is because the content of your form is not handled yet...

Modify the `index` method of your pages module so that when it receives a `POST` request with a field `name`, your index page uses it instead of `GET` in order to greet the visitor.

Keep in mind that your index method must be able to handle both behaviors.

## Step 4

Create a **client module** containing a method called `connect`, taking a host and a port as parameters.

As soon as the `connect` method is called, your client must:

- ✓ connect to the given address and port ;
- ✓ send a message containing `ping` before disconnecting.

Any answer from the server will be displayed on standard output followed by a new line.

## Step 5

Before going further, familiarize yourself with Express and its API.

As you may have seen with our small Node project, configuring a basic server with vanilla Node.js requires a fair amount of code and is not easy to use and maintain.

Moreover, it's quite easy to do it the wrong way.

Express should make things a lot easier.

## Step 6

First, use Express to implement a `start` method in an **app module**.

This method takes a port as parameter and launches an Express application listening on that port.

Use Express to create the following routes:

- ✓ `/` and `/index`: returns **index.html**, displaying `Greetings Traveler!` in plain text.
- ✓ `/image`: returns **image.html**.
- ✓ `/form`: returns **form.html**.
- ✓ `/student/X`, where `X` is any number: returns **student.ejs**

**student.ejs** must generate an HTML page, according to the following example.

For instance, a call to `localhost:port/student/5?name=Martin` should generate this code:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Student</title>
  </head>
  <body>
    <p>Greetings, Martin, student number 5!</p>
  </body>
</html>
```

Do you know what **ejs** is?

If a user tries to access a page that does not exist, return:

- ✓ a 404 error ;
- ✓ with an appropriate message in plain text.

No need to hack something together for non-existing files.
Express has something to do exactly that.

## Step 7

Using **cookie-parser**, add a new path to your application complying with the following behavior:

- ✓ path `/memory` renders a page displaying last access data to page `/student/X`.
- ✓ an access to `/student/999?name=rico` and then `/memory` displays in plain text `rico, student number 999 was here.`
- ✓ an access to `/student/999` then to `/memory` displays `student number 999 was here.`
- ✓ a direct access to `/memory` displays nothing (you don't have to remove the cookie).

In order to save those parameters even if the user closes their browser, save them in 2 cookies named `name` and `number`.

v 1.3.1




* apprendre autrement