Node.js

For PHP developers

What are we going to do?

- **Short presentation of Node.js**
  - What?
  - Why?
  - When?
  - How?
- **Follow along coding examples**
  - Good practices (Import, Asynchronous programming, Promisified functions)
  - Essentials™
    - Our first http server
    - Using Express
    - DB, authentication, sessions, cookies
  - What to do Next? (pun intended)
- **Open session**

Code snippets available here:

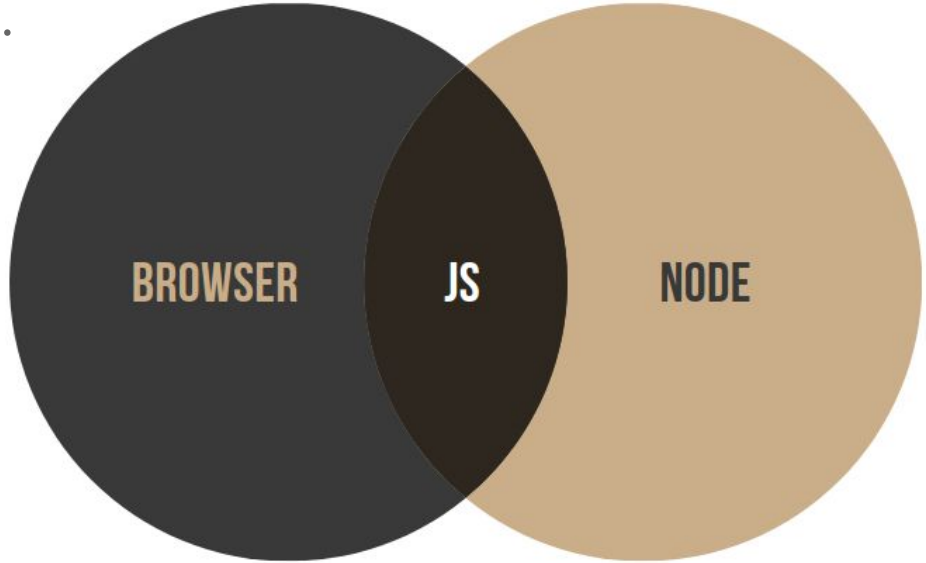
https://github.com/BjrInt/Node-for-PHP-developers

Make sure to check out the new Node.js introduction track if you want to experiment after this workshop!

What is Node.js?

A concise definition...

Node.js is an open-source, cross-platform, **JavaScript runtime environment** that runs on the **V8 engine** and executes JavaScript code **outside a web browser**.



What is part of the Javascript language?

- Expression & operators
- Variables & constants
- Statements (conditions, loops, switches)
- Arrays, “objects”, Functions, Typed Arrays, Set, Map
- Built in objects, such as…
  - Date
  - Math
  - Regex

Some specific features of each environments

Browser

- DOM interactions
- WebRTC (Webcam, microphone, ...)
- Fetch*
- Canvas
- LocalStorage
- Mobile interaction (push notifications, battery info, network status, ...)

... and many more!

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

Node.js

- HTTP module
- Streams/Buffer
- OS & Machine related libs
- File System

... and many more!

https://nodejs.org/dist/latest-v16.x/docs/api/

MDN might help

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Deno	Node.js
`import`	✓ 61	✓ 16	✓ 60	✗ No	✓ 48	✓ 10.1	✓ 61	✓ 61	✓ 60	✓ 45	✓ 10.3	✓ 8.0	✓ 1.0	✓ 13.2.0 *
Dynamic import	✓ 63	✓ 79	✓ 67	✗ No	✓ 50	✓ 11.1	✓ 63	✓ 63	✓ 67	✓ 46	✓ 11.3	✓ 8.0	✓ 1.0	✓ 13.2.0 *
Available in workers	✓ 80	✓ 80	✗ No *	✗ No	✗ No	✓ 15	✓ 80	✓ 80	✗ No *	✗ No	✓ 15	✓ 13.0	✓ 1.0	✗ No

Why should I learn Node.js?

PROs & CONs

Node is “bare metal”, you get complete freedom over your backend...	Everything has to be done from the ground up
Biggest package repository	NPM is a mess, from blatant malwares, to obsolete stuff.
It’s just Javascript...	It’s still Javascript...
Can be extremely fast with a good knowledge of event-driven processes	... Or extremely slow. Design mistakes can easily sink projects

How does it surpasses PHP ?

- It is not strictly backend oriented. You can do a whole range of applications using Node (Discord & VSCode are made with Node).
- Creating an HTTP server is analogous to how you would manage it with other languages (Python, Go, ...), it is more generic than a LAMP stack, thus more reusable in a “teaching” context.
- Furthermore you don’t need an Apache or Nginx server to start from, you only need the Node.js interpreter.
- Creating Event-driven workflows or CRON jobs is as simple as setting an interval.
- There are a variety of frameworks (*more on that later*) that allow you to create a fullstack application using a single workflow (Next/Nuxt.js, ...)

When should I use Node? (instead of PHP)

Interesting use-cases (vs PHP)

- **Rest APIs**
- HTTP interfacing (ex: Microservices)
- Stream-based applications
- Real time communications (using sockets)
- (Web3?)

Apps that focus on “complex” user interaction and dynamic rendering.

Ex: Notion, Trello, FixMyStreet, Figma, Canva, Whatsapp web, Food delivery apps, Streaming services

Not that interesting

- Static (or mostly static website)
- Traditional server rendered applications.
- Computation heavy manipulation (anything that is I/O blocking)

Ex: E-commerce websites, blogs, landing pages

Let’s get started!

Get Node.js

- Use your OS package manager (apt, brew)
- Always check that you are using an **even** version (14, 16, 18, ...), odd versions are for experimental development (`node -v`)
- As a rule of thumb, when in doubt, always go for the **LTS** version.
- You can update your Node.js version using the “n” node package `sudo npm i -g n && sudo n lts` to get the LTS version
- ... you want multiple version on your machine?
  - NVM, Node Version Manager, to easily switch between versions
  - Use the Node.js docker image and leverage the power of containers

Have fun! (while it lasts)

- Open Node in REPL mode
- Write some Javascript code
- You can even execute files
- Congratulations! You’re a Node dev

Looking for Javascript syntax in 2022

Callbacks

Promises

Async/Await

require

import

CommonJS? UMD?

-- experimental-flag

var x = new Array()

Some life saving principles to survive the clusterfudge

A linter might help (vscode has one)

- **Never use var** (use let/const)
- Never use deprecated methods

Good practices.

- Prefer **import** over require
- Never use **sync** methods when dealing with incoming requests.
- Prefer **promisified functions** over callbacks

Let's learn good practices with an example

To learn more about good practice, we'll go over opening a file in js.

1. We'll convert old Node.js **require**s into shiny ES Modules **import**
2. We'll learn to **avoid** using **synchronous** method when dealing with I/O
3. We'll see how to reduce **callback hell** with promisified functions

Prefer import over require

1/3 Use import

JS has a long history with module and dependency management. In the old days (back when we were only talking about client-side js), there were **no way to split applications into module**, so when Node.js came to existence they needed to introduce a way to split logic into separate files and the “**require**” keyword came to life.

Fast forward to present time. **ES Modules** have become the standard way of importing and exporting stuff in JS.

const { readFile } = require('fs')

import { readFile } from 'fs'

1/3 Use import

The perks of using ES Modules :

- You use the same methods than on your browser
- This is a more standard way of importing dependencies anyways (ex: Python, Go, Java, ...)
- You choose how to handle the global namespace:
  - import ‘filename.mjs’ (similar to PHP include())
  - import * as myLibrary from ‘filename.mjs’ (similar to PHP use)
- You can do asynchronous loading of ES modules

Never use sync methods (unless you know what you're doing)

2/3 No sync method

https://nodejs.org/dist/latest-v16.x/docs/api/fs.html#synchronous-api

The synchronous APIs perform all operations synchronously, blocking the event loop until the operation completes or fails.

Using a synchronous method basically mean your server cannot handle incoming requests while processing the function!

2/3 No sync method

The naive dirty way...

```
import { readFileSync } from 'fs'

const fileContent = readFileSync('/path/to/file', 'utf-8')
```

The shiny A S Y N C method...

```
import { readFile } from 'fs'

readFile('/path/to/file', 'utf-8', (err, data) => {
  if(err)
    return console.log(err)

  console.log(data)
})
```

2/3 No sync method

Some other examples of why using async process (beside unblocking the main thread) might be useful :

- It's easier to manage errors
- When making a CLI tool you can display the progress of the task
- You can abort the operation if it takes too long to resolve (using the AbortController method)
- You can batch operations with concurrent tasks

If a sync function is taking too much time, there certainly exists an async equivalent you should use! (db transaction, file manipulation, http requests)

Prefer promisified functions over callbacks

```
// Callback Hell


a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});

```

3/3 Use promisified methods

```
import { readFile } from "fs/promises"

const displayFileLength = async () => {
  try {
    const content = await readFile()
    return content.length
  } catch (err) {
    return err
  }
}
```

3/3 Use promisified methods

Most Node.js modules dealing with I/O (file manipulation, databases), whether they are part of the standard library or downloaded from NPM generally have a promisified API.

Even if they don't, there is utils.promisify to automagically convert a callback-based method into its promise-based counterpart.

Backend
ESSENTIALS

The HTTP module

The express framework

Cookies, sessions & authentication