Demystifying Node & React

Roger Theriault

Ahmed El Azzabi

Objectives & audience:

+ This workshop is:
    - **For everyone**. No programming experience necessary!
    - **Hands-on** (optionally). You can do the exercises right on your Mac.
    - **A starting point**. You will be demystified, but we'll take questions too.

+ After this workshop, you should:
    - Have a basic understanding of **Node.js, React, Redis**, and how they can work with WordPress on VIP Go.
    - **Understand how data travels** between the browser, and Node and WordPress in a decoupled architecture.
    - Be able to get **a complete stack running** using Docker Compose on your desktop.
    - **Be comfortable having conversations with clients** about Node.js and React.

What is Node.js and React.js?

Some history:

In order to have a web page, you need 3 files: HTML, CSS, and JavaScript.

HTML is for markup: titles, paragraphs, lists... CSS is for styling: changing colors, spacing... Both make a static web page.

JavaScript is used to make pages dynamic: animations, HTTP requests... JavaScript is a programming language, HTML and CSS are not.

Some history:

Problem: JavaScript interpreters (programs transforming JavaScript files to machine code, equivalent to a compiler) were tied to the browsers. Which made JavaScript only work on browsers.

In 2008, Google open sourced Chrome V8, the JavaScript interpreter in Chrome as a standalone program.

Node.js was born the same year to use Chrome V8 in the server.

What are these things: Node and React?

Node.js

A JavaScript runtime for the server

https://nodejs.org/en/about/

React.js

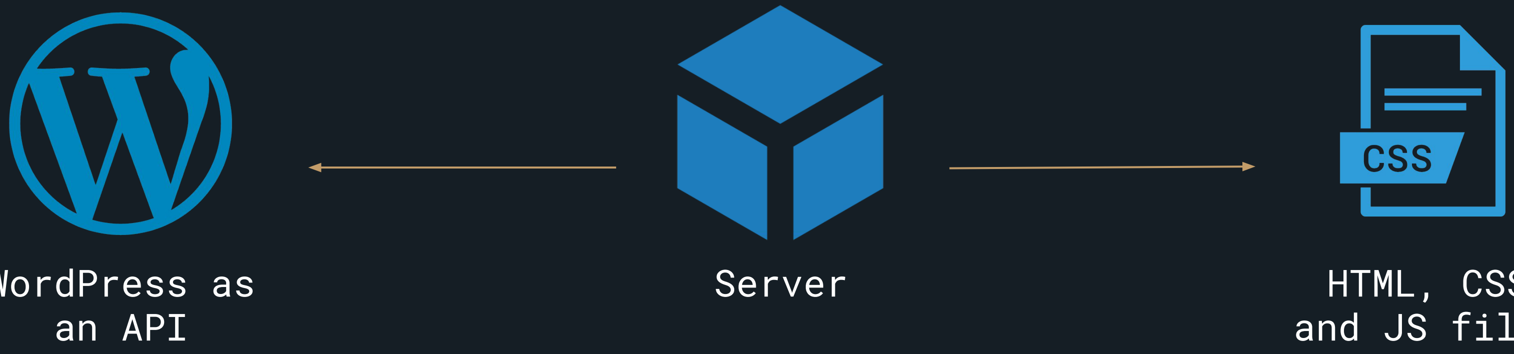
A JavaScript library for building user interfaces

https://reactjs.org/

> React does not run only on Node.js, it runs in any JavaScript runtime including a browser. It does use Node, in development mode, to assist with development, and uses a node package manager to manage and install dependencies.

Understanding the decoupled architecture

A simple decoupled architecture:



Things to remember about React.js:

- **You can serve React from WordPress**
  React is not tied to Node.js. You can construct your pages and serve them directly with WordPress
- **You're not obliged to use React everywhere on your site**
  React can be your whole site, some of it, or just a module somewhere in a page
- **React uses Node Package Manager**
  To install and manage dependencies. Node Package Manager (NPM) is used
- **React runs on a browser (usually)**
  Different JavaScript frameworks used to build rich frontend applications
- **You have other choices**
  Even if we talk here about React. There are other choices: Vue, Angular, etc. Or you can use JavaScript without a framework if you don't need a complexe user interface

Things to remember about Node.js:

+ **A standalone service**
  Node.js runs as a standalone server and isn't always tied with a WordPress backend

+ **Can be connected with other services**
  Node.js can be connected with other services using HTTP(s) and also supports libraries for databases, memcache, etc.

+ **Offered on VIP Go**
  It is commonly used to decouple WordPress, but can also be used as a separate service

+ **Node.js uses NPM too**
  Like React, Node.js uses NPM to manage dependencies too

+ **Runs on a server (usually)**
  Node.js usually run on a server

What we support at VIP

What we currently support:

Node.js Application

1

Can be used as a microservice, a frontend app consuming a backend (can be WP or anything else), etc.

Node.js & Redis

2

Applications needing a caching layer (Redis). Can be used by APIs to cache responses, etc.

Node.js & MySQL

3

Applications needing to store data (MySQL). Can be used by apps performing data manipulation, log audits...

Node.js & Redis & MySQL

4

Workshop

If you haven’t already, you can get exercise files from GitHub:

```
$ git clone https://github.com/Automattic/vip-gm2019-workshop-node.git
```

Setting up Node.js:

1 Install Node Version Manager

```
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
```

2 Install the latest Node.js version

```
$ nvm install 10
```

&

```
$ nvm install 10
```

3 (Optional) Install Yarn

```
$ curl -o- -L https://yarnpkg.com/install.sh | bash
```

&

```
$ yarn --version
```

Creating a Node.js server: Part One

1 Create a new folder and use npm init to create a package.json

```
$ mkdir server && cd server && npm init -y
```

2 Install Express.js

```
$ npm install express
```

We are using Express.js to create our HTTP servers

3 Create a server.js file

```
$ touch server.js
```

4 Inside server.js, require express and create an Express app:

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

5 Define a route responding to GET requests:

```
app.get( '/ping', ( req, res ) => {
    return res.send( 'pong' );
} );
```

Creating a Node.js server: Part One

6 Listen to traffic on a port:

```
app.listen( 4000, () => {
    console.log( ‘listening on PORT 4000’ )
} );
```

7 Execute your file:

```
$ node server.js
```

8 Test it using curl or your browser:

```
$ curl http://localhost:4000/ping
```

1 Install morgan:

```
$ npm install morgan
```

2 Require morgan in server.js:

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

3 Use morgan with your app:

```
app.use( morgan( 'dev' ) );
```

Fetching data: Build it

1 **Install axios and cors:**

```
$ npm install axios cors
```

2 **Require them in server.js:**

```
const axios = require( 'axios' );
const cors = require( 'cors' );
```

3 **Use cors with your app:**

```
app.use( cors() );
```

Fetching data: Build it

4 Add a /users route:

```
app.get( '/users', async ( req, res ) => {
    const count = req.query.count || 10;
    const response = await axios.get( ‘https://randomuser.me/api?results=’ + count );
    res.json( { data: response.data.results } );
})
```

5 Restart your server:

```
$ node server.js
```

6 Test it using curl or your browser:

```
$ curl localhost:4000/users
$ curl localhost:4000/users?count=1
```

1 Install create-react-app globally:

```
$ yarn global add create-react-app
// or
$ npm install -g create-react-app
```

2 Create a new app:

```
$ create-react-app myApp
```

3 Start the app using yarn or npm:

```
$ yarn start
$ npm start
```

1 Clone exercise files (if not already done):

```
$ git clone https://github.com/Automattic/vip-gm2019-workshop-node.git
```

2 Replace your App.js file with this file:

```
vip-gm2019-workshop-node/exercises/ex4-react-node/App.js
```

1 Build your React application to make it ready for production:

```
$ yarn build
//or
$ npm run build
```

2 Back to our server.js file, let’s define static files directory:

```
// Import the path library
const path = require( ‘path’ );

// Serve client built files
app.use( express.static( path.join( __dirname, '../path/build/directory/' ) ) )
```

3 Let’s respond to requests from / with our index file:

```
// map / to serve index.html
app.get( '/', ( req, res ) => {
    res.sendFile( 'index.html', {
        root: path.join( __dirname, '../path/build/directory' ) }
    );
} );
```

4 Restart your server:

```
$ node server.js
```

Welcome Redis

Docs at https://redis.io/

redis is a bit different from Memcached:

- Different data types including lists & sets
- Operations on data
- Lua scripting
- Persistence on disk
- A powerful redis-cli

Adding Redis

1

Install redis (requires Homebrew):

```
$ brew install redis
```

2

Two ways of starting redis:

```
// redis as a backend service (opened automatically after reboot…)
$ brew services start redis

// redis as a simple service
$ redis-server /usr/local/etc/redis.conf
```

Some redis-cli commands

1 Start the cli:

```
$ redis-cli
```

2 Play with it:

```
> get foo
(nil)
> set foo bar
OK
> get foo
“bar”
> exit
```

1 Install redis with npm or yarn:

```
$ npm install redis
OR
$ yarn add redis
```

2 Require redis and create a client:

```
const redis = require( 'redis' );
const client = redis.createClient( 6379 );
```

3 In your server.js file, replace /users route code with:

```
app.get( '/users', async ( req, res ) => {
    const count = req.query.count || 10;
    const cacheKey = 'users-' + count;
    return client.get( cacheKey, async ( err, results ) => {
        if ( results ) {
            return res.json({ source: 'cache', data: JSON.parse(results) })
        }

        const response = await axios.get( `https://randomuser.me/api?results=${count}` )

        client.setex( cacheKey, 3600, JSON.stringify(response.data.results) )

        res.json( { source: 'api', data: response.data.results } )
   } )
} )
```

4 Restart your server, and try it:

```
$ curl localhost:4000/users?count=1
$ curl localhost:4000/users?count=1
$ curl localhost:4000/users?count=2
$ curl localhost:4000/users?count=2
```

5 The logs should display something like:

```
GET /users?count=1 200 315.831 ms - 1097
GET /users?count=1 200 0.964 ms - 1099
GET /users?count=2 200 135.653 ms - 2148
GET /users?count=2 200 0.618 ms - 2150
```

Caching with redis key learnings

+ **We cache each endpoint**
  Each endpoint have a separate key and cached for 3600 seconds

+ **Cache updated when data is expired**
  When data is expired, we hit the API to get new data

+ **Cached responses take less time**
  Given we don’t hit the API everytime, cached responses take less time

+ **The app behaves differently**
  Given we are caching the response, users aren’t random anymore

Desktop Exercises Summary

+ **Node server**
    - Handling http requests on port 4000
    - Serving /ping and /users
    - Fetching and caching random(ish) users
    - Serving the React build and other static files and assets

+ **React client project**
    - Served by Node.js in production
    - Displaying random (or cached) users

+ **Redis server**
    - Storing our cached data

Combining everything in Docker

Go to this exercise directory:

1

```
$ cd vip-gm2019-workshop-node/full-docker/
```

2

Execute our jumpstart script:

```
$ ./bin/jumpstart.sh
```

3

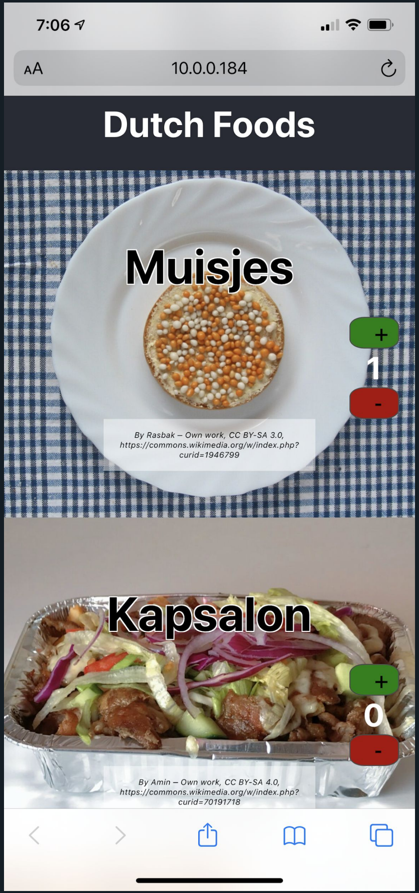
Start docker container:

```
$ docker-compose up
```

You now have a full decoupled environment

+ **WordPress + MariaDB**
  - A few articles in the food category
  - A custom hook to refresh node

+ **Node + Redis + React**
  - Fetches food articles via REST API and caches in Redis
  - Handles food voting and stores in Redis
  - Client polls for updates and displays food votes


7:06
10.0.0.184
Dutch Foods
Muisjes
+
1
-
By Rasbak – Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1946799
Kapsalon
+
0
-
By Amin – Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=70191718

How it works

- **Vote count**
  Simple vote count functionality with a Node endpoint /vote

- **User sends an action**
  User's action is used to increment or decrement a redis counter in a hash

- **A polling system**
  An ajax polling request continuously fetches the current list of votes and updates the state of the items

- **Listens to WordPress changes**
  When you add a new food or change something in WordPress it will be automatically updated on the clients

Summary

Node.js is used in a server

React is a client framework that can be served from Node or WordPress

WordPress is awesome (and uses MariaDB/MySQL and memcached)

Redis is a data store that's popular with Node.js

Yarn and NPM is used to manage dependencies for React and Node.js and to build projects

Create-react-app is a bootstrap that includes all the pieces to develop and deliver React client apps

Gutenberg uses React

Docker allows you to run a server in a container and is good for closely replicating production on your Mac

Docker-compose runs interdependent microservices in multiple containers (our VIPd doesn't really use those)

Summary

Resources

+ This workshop is on GitHub:
  https://github.com/Automattic/vip-gm2019-workshop-node
+ Redis commands cheat sheet:
  https://www.cheatography.com/tasjaevan/cheat-sheets/redis/
+ Docker Compose exercise:
  https://docs.docker.com/compose/gettingstarted/