Demystifying Node and React

WordPress + Node.js Workshop*

And more.. mind-blowingly more! After this workshop you will know all-the-things!

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.

Workshop Optional Prerequisites

We'd encourage everyone to bring their MacBook to the workshop and follow along.
If you have time before the workshop, please install these apps:

Install XCode or Command Line Tools `$ xcode-select --install`

Install Homebrew https://brew.sh/

Install Visual Studio Code https://code.visualstudio.com/download

Install Docker Desktop (and follow the simple tutorial to create your first project in Docker Hub) https://www.docker.com/products/docker-desktop

Part One	*What is Node.js and React.js?*
Part Two	*Understanding the decoupled architecture*
Part Three	*What we support at VIP*
Part Four	*Workshop*

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

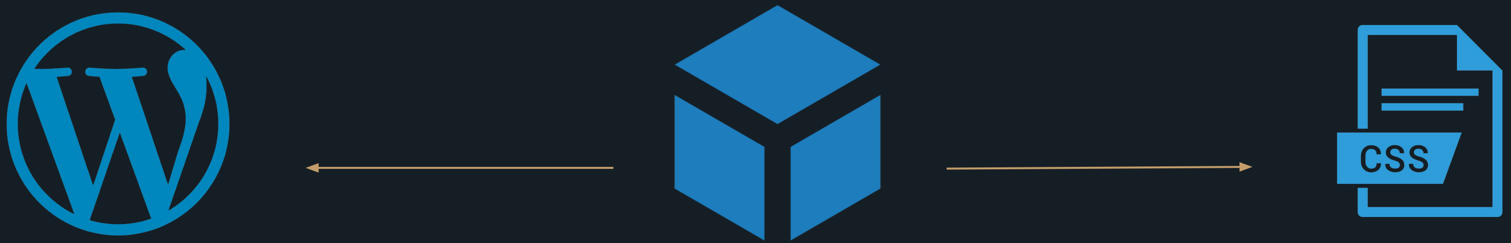
A JavaScript library for building user interfaces

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

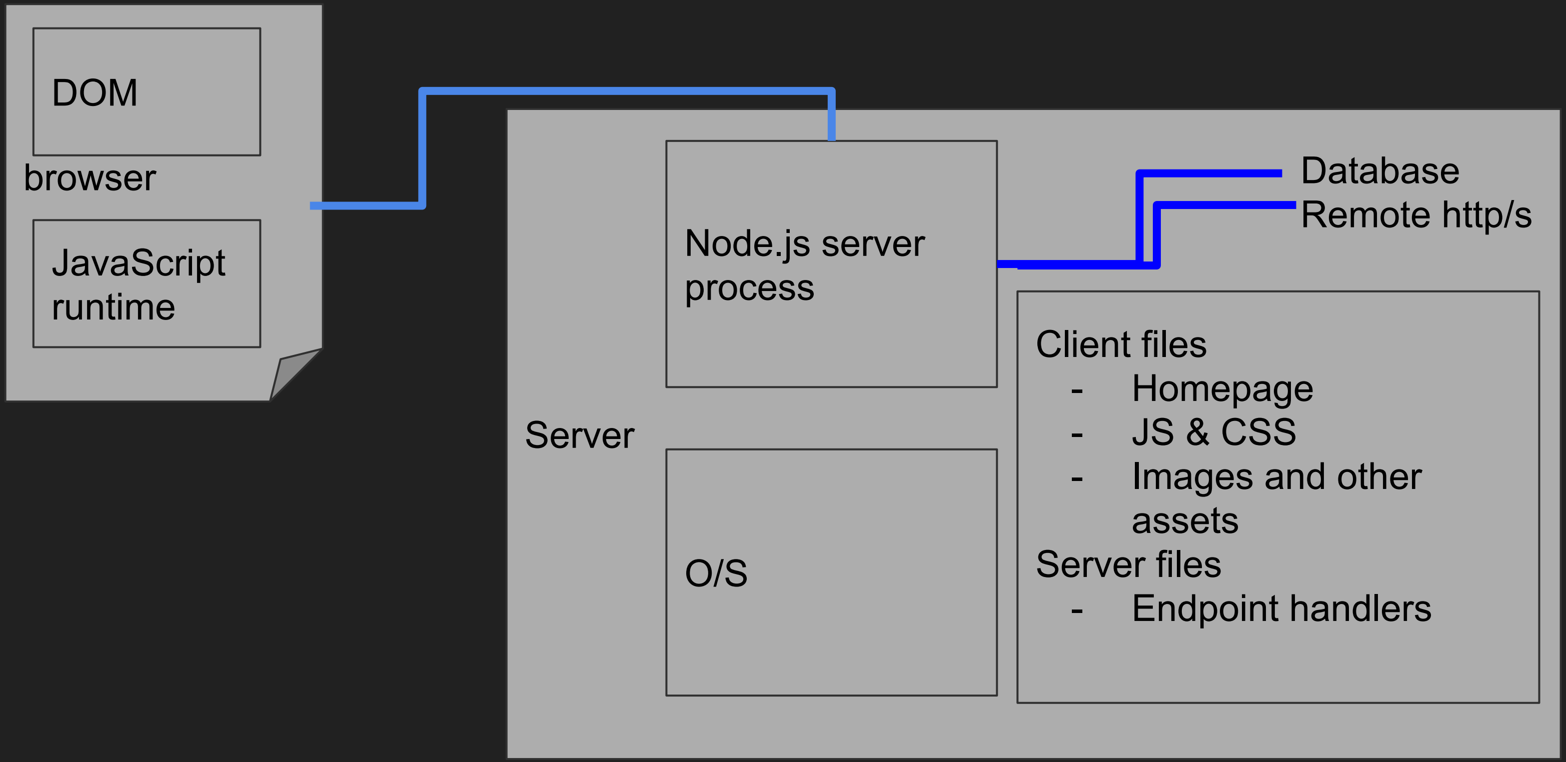
A simple decoupled architecture:



WordPress as an API

Server

HTML, CSS and JS files



Node.js and React Typical Combination

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

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 i express
```

We are using Express.js to create our HTTP servers

3 Create a server.js file

```
$ touch server.js
```

Creating a Node.js server: Part One

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
```

Creating a Node.js server: Part One

8 Test it using curl or your browser:

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

Creating a Node.js server: Part Two

1 Install morgan:

```
$ npm i morgan
```

2 Require morgan in server.js:

```
const morgan = require( ‘morgan’ );
```

3 Use morgan with your app:

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

EX1 - Create a Node.js Server

Create folder and server directory

Create (mkdir) or clone

```
$ cd ~
$ git clone https://github.com/Automattic/vip-gm2019-workshop-node.git
$ cd vip-gm2019-workshop-node
$ mkdir server; cd server
```

Initialize & add packages

```
$ yarn init -y
$ touch index.js
$ yarn add morgan express cors axios
$ yarn add --dev nodemon
```

Add code & test

```
$ node index.js
$ curl http://localhost:4000/ping
```

```
// v1 - basic ping/pong demo
const express = require('express')
const morgan = require('morgan')
const cors = require('cors')
const axios = require('axios')

const port = 4000

// Express
const app = express()

app.use(express.json())
app.use(morgan('dev'))
app.use(cors())

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

app.listen(port, () => {
    console.log(`listening on PORT ${port}`)
})
```

Fetching data: Build it

1. Install axios and cors:

```
$ npm i 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
```

Fetching data: Test it

6 Test it using curl or your browser:

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

What did we just do?

Yarn installed packages into `node_modules`

Express is a minimal web application framework. `app` instantiates that.

Morgan is logging the requests.

The app is set up with a request handler for /ping which simply returns “pong”

`app.listen` listens on port 4000 and then matching handlers may take action

Express: https://expressjs.com/

Node package module reference:

https://www.npmjs.com/package/morgan

https://www.npmjs.com/package/cors

https://www.npmjs.com/package/axios

EX2 - Fetching data

v2 adds a listener on `/users` that returns a list of users fetched from *randomuser.me*

```
app.get('/users', async (req, res) => {
    const count = req.query.count || 10
    const response = await
axios.get(`https://randomuser.me/api?results=${coun
t}`)
    res.json({data: response.data.results})
})
```

```
$ node index-2.js

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

EX3 - Create a React application

Install create-react-app

```
$ cd ~/vip-gm2019-workshop-node
$ yarn global add create-react-app
```

Create and run client app

```
$ create-react-app client
$ cd client
$ yarn start
```

A browser should load localhost:3000

Update App.js (hotloaded) to load the JSON

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  const [count, setCount] = React.useState(0)
  const [people, setPeople] = React.useState([])

  async function getPeople() {
    const res = await
fetch(`http://localhost:4000/users?count=${count}`)
    const resData = await res.json()
    setPeople(resData.data)
  }

  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo"
alt="logo" />
        <div>
          <h1>React and Node People Fetcher</h1>
          <input
            style={{
              fontSize: '2rem'
            }}
```

EX4 - Build a production client

Build the client

```
$ yarn build
```

Note the new build directory

Update the server and start node

```
$ node index-4.js
```

Open localhost:4000 in a browser

```
const port = 4000

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

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

Adding Redis

Install redis (requires Homebrew)

```
$ brew install redis
To have launchd start redis now and restart at
login:
  brew services start redis
Or, if you don't want/need a background service you
can just run:
  redis-server /usr/local/etc/redis.conf
$ brew services start redis

$ redis-cli
> get foo
(nil)
> set foo bar
OK
> get foo
“bar”
> exit
```

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

redis-cli is powerful

```
$ redis-cli monitor
$ redis-cli --scan
```

EX5 - Caching with Redis

Add Redis to the server package

```
$ yarn add redis

$ node index-5.js
listening on PORT 4000

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

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

$ redis-cli
> get users-1
{JSON STRING}
> del users-1
OK
```

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

client.on('error', (err) => {
    console.log("Redis Error " + err)
});

// in get(‘/users’):
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 })
})
```

EX5 - Caching with Redis

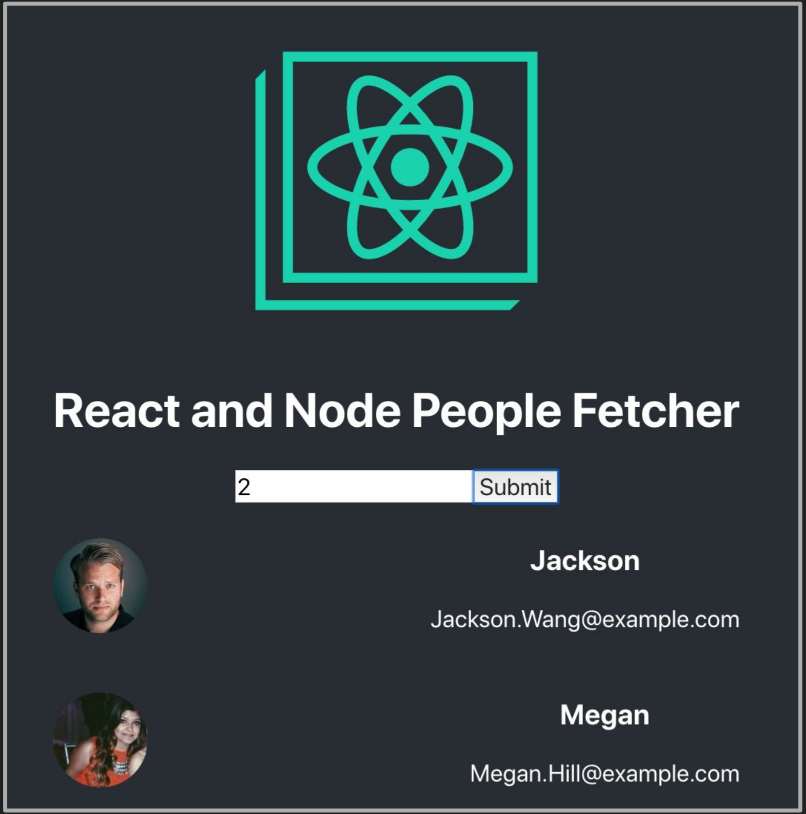
What did we add?

- Cache each endpoint with a separate key, with 3600s expiry
- When it expires, the api is hit again
- Cached responses take much less time

The app behaves differently: users are no longer random!

This doesn't work for all use cases

```
$redis-cli --scan
```



React and Node People Fetcher

Jackson

Jackson.Wang@example.com

Megan

Megan.Hill@example.com

Dockerize

Now we'll set up a local VIP Go development environment

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

We've supplied a jumpstart script:

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

```
$ docker-compose up -d --build
$ docker-compose ps
$ docker-compose down
```

https://docs.docker.com/compose/

WordPress user: welcome/welcome

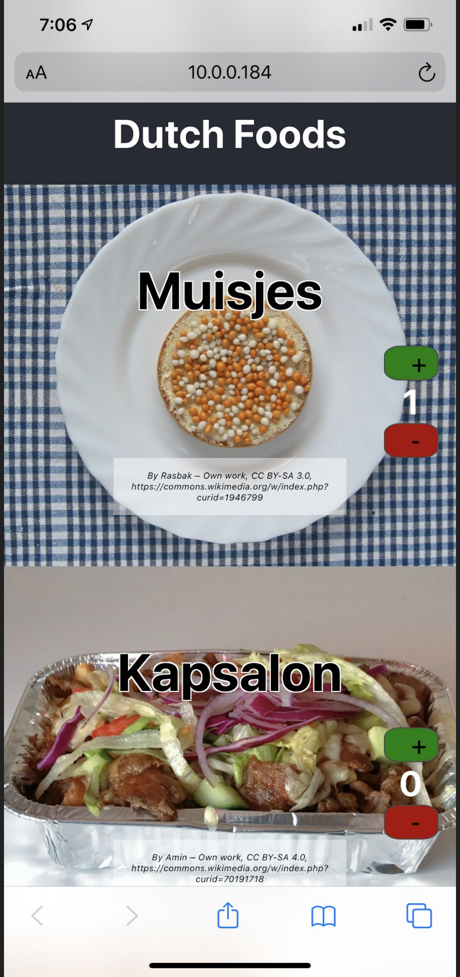
You now have a full decoupled env!

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



Dutch Foods

Muisjes

By Rasbak – Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1946799

Kapsalon

By Amin – Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=70191718

How it works

Simple vote count functionality with a Node endpoint /vote

User’s action is used to increment a redis counter in a hash

An ajax polling request fetches the current list of votes and updates the state of the items

When you add a new food or change something in WordPress it will be updated on the clients

Summary

Node.js is 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 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)

What else?

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/