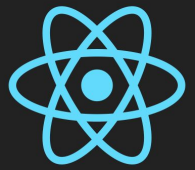


Demystifying Node and React

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

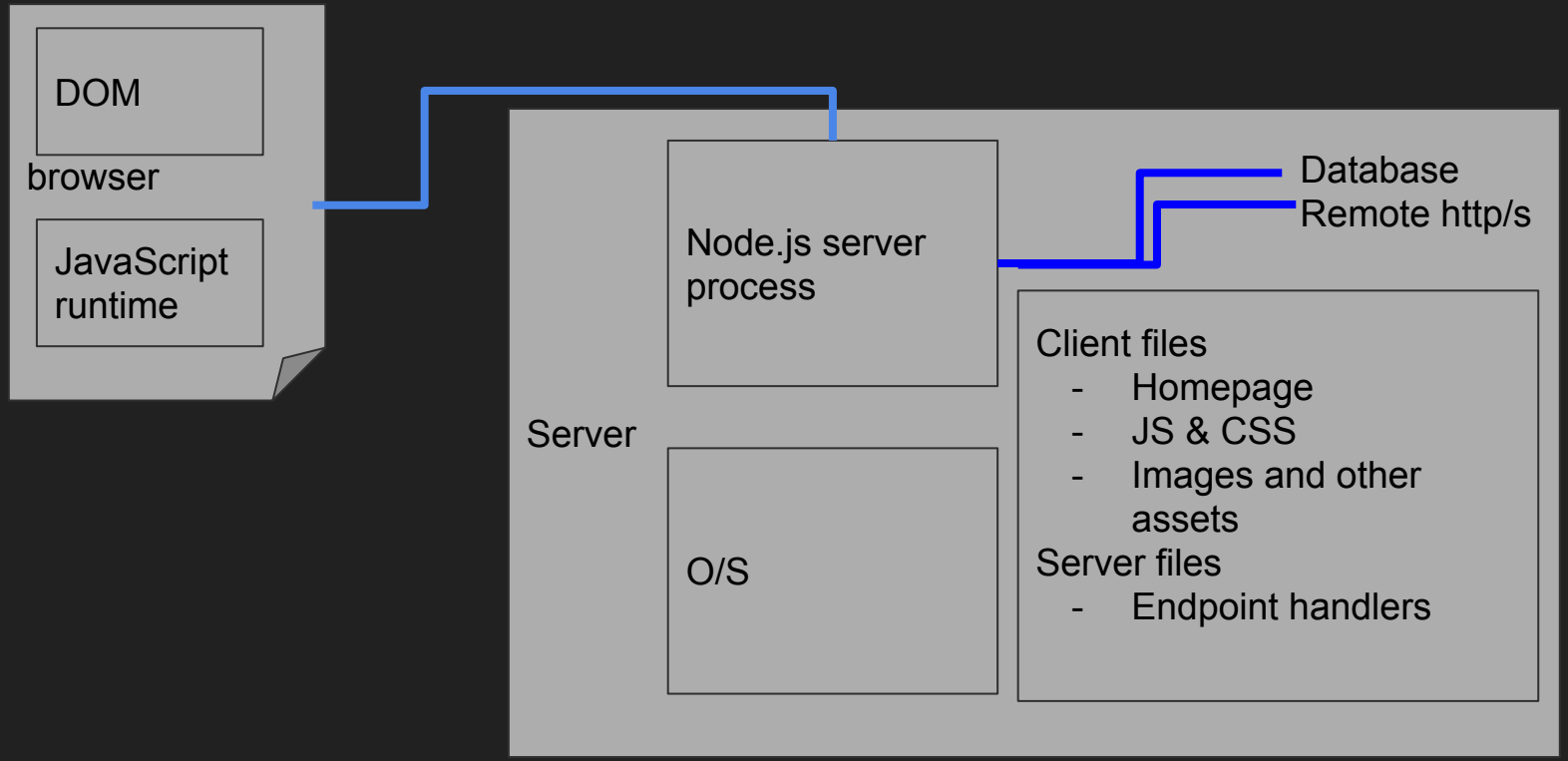
What are these things: Node and React?

Node.js: a JavaScript runtime https://nodejs.org/en/about/

React: A JavaScript library for user interfaces https://reactjs.org/

Did you know?

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.



Node.js and React Typical Combination

Things to remember

React

You can serve up React from WordPress, without Node.js

React can be all of your site, some of it, or just a module somewhere on a page

Developing a React app is very different from running in production

React uses node package manager

Runs on a browser (usually)

Node.js

Node.js typically operates as its own standalone service, without WordPress

Node.js can also connect to external resources via http(s) and also supports client libraries for databases, Redis, memcached, etc.

Commonly on VIP Go, Node.js is used to “decouple” WordPress from the browser, but can also be used as a separate service

Node uses node package manager

Runs on a server (usually)

Setting up Node.js

Install Node Version Manager https://github.com/nvm-sh/nvm

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

Install Node.js LTS

```
$ nvm install 10
```

```
$ nvm ls
```

Install Yarn, an alternative package manager (or use NPM if you prefer)

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

```
$ yarn --version
```

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}`)
})
```

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

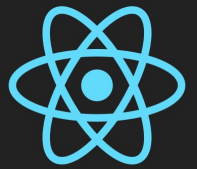
v2 adds a listener on `/users` that returns a list of users fetched from *randomuser.me*

```
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-3.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
```

Add caching layer to request/response

TODO

Major WIP

Remaining slides are placeholders for now

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/
```

Run compose up and a blank WordPress site should be running, along with a node site, redis, memcached, and mysql

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

Optionally, supply a db dump with 100 posts, and a theme repo that just returns those via REST API

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

You now have a WordPress local env!

Pull WordPress content into Node & React

Modify the server to fetch /posts from the WP API

Modify the client app to display top posts as cards with a count above them

Add some interaction

Add simple like count functionality with a new Node endpoint /like

User’s /post/:id/like/ is used to increment a redis counter for a post, and then update a tally of all counts

An ajax polling request fetches the current list of likes and updates the attributes of the posts

Demonstrate how React updates the UI automatically when state changes

Slides describing what we offer

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/