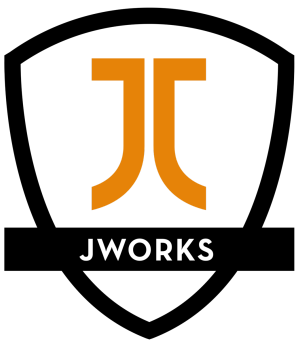
INTRODUCTION TO WEBPACK

MODULE BUNDLER



POWERED BY ORDINA

HI, MY NAME IS JAN.

Frontend Developer
Ordina Belgium
@Mr_Jean
https://github.com/MrJean

TOPICS

Introduction

Configuration

Integration with AngularJS & React

INTRODUCTION

WEBWHAT?

- Like Gulp or Grunt, but different
- Does more with less
- Fairly easy configuration
- Bundles files
- Uses JavaScript modules
- Reduces the amount of (XHR) requests

BASICS

```
$ npm install webpack -g
$ webpack app.js bundle.js
$ webpack app.js bundle.js --watch
```

REQUIRE A FILE

```
require('./filename') // CommonJS syntax
```

```
import {filename} from './filename' // ES6 syntax
```

Relative referral to filename.js

EXAMPLE 1

world.js

```
module.exports = 'world!';
```

hello.js

```
var world = require('./world');
document.write('Hello' + world);
```

Run

```
$ webpack hello.js bundle.js
```

EXAMPLE 2

world.js

```
module.exports = {
    sayWorld: function() {
        return 'world!';
    }
}
```

hello.js

```
var world = require('./world');
document.write('Hello ' + world.sayWorld());
```

Run

```
$ webpack hello.js bundle.js
```

WEBPACK-DEV-SERVER

- Use when you need HTTP
- Sets up webserver
- Serves files virtually (not from disk)
- Reloads browser when files change

WEBPACK-DEV-SERVER

Install

```
$ npm install webpack-dev-server –g
```

Run

```
$ webpack-dev-server
```

Needs **webpack.config.js** file

WEBPACK-DEV-SERVER

Default location gives status window with automatic reload

```
http://localhost:8080/webpack-dev-server/
```

Just the app and no reload

```
http://localhost:8080
```

Enable reload on **`http://localhost:8080`**

```
$ webpack-dev-server --inline
```

CONFIGURATION

WEBPACK.CONFIG.JS

- Plain JavaScript
- Use require to include modules or plugins
- Write own custom logic

WEBPACK.CONFIG.JS

```
module.exports = {
    entry: ["./hello"],
    output: {
        filename: "bundle.js"
    }
}
```

FLAT STRUCTURE

```
hello.js
world.js
index.html
bundle.js
webpack.config.js
```

MORE STRUCTURE

```
/js
    hello.js
    world.js
/public
    index.html
/build
    /js
        bundle.js
webpack.config.js
```

ENTRY

```
var path = require('path');

module.exports = {
    entry: ["./utils", "./hello"]
}
```

The entry files webpack will use.

OUTPUT

```
var path = require('path');

module.exports = {
    output: {
        path: path.resolve('build/js/'),
        publicPath: '/public/assets/js/',
        filename: "bundle.js"
    }
}
```

`path`: location where webpack will build the files.

`publicPath`: location where webpack webserver serves files from, each request will be mapped to look up in **`build/js/`**.

CONTEXT

```
var path = require('path');

module.exports = {
    context: path.resolve('js')
}
```

The base directory for resolving the entry option

DEVSERVER

```
var path = require('path');

module.exports = {
    devServer: {
        contentBase: 'public'
    }
}
```

`contentBase`: Defines the root of the server.

RESOLVE

```
var path = require('path');

module.exports = {
    resolve: {
        extensions: ['', '.js']
    }
}
```

Options affecting the resolving of modules.

WEBPACK.CONFIG.JS

```
var path = require('path');

module.exports = {
    context: path.resolve('js'),
    entry: ["./utils", "./hello"],
    output: {
        path: path.resolve('build/js/'),
        publicPath: '/public/assets/js/',
        filename: "bundle.js"
    },
    devServer: {
        contentBase: 'public'
    },
    resolve: {
        extensions: ['', '.js']
    }
}
```

LOADERS

- Loaders are pieces that transform files
- Used to teach webpack some new tricks
- Much like tasks in other build tools
- Preprocess files as you `require()` or "load" them

BABEL LOADER (ES6 - ES5)

```
$ npm install babel-core babel-loader babel-preset-es2015 --save-dev
```

- **`babel-preset-es2015`** is required
- **`.babelrc`** file required and must contain **`{ "presets": ["es2015”] }`**
- filetype is **`.es6`** when using ES6 (ES2015)
- run in strict mode **`'use strict'`**

INTEGRATING THE BABEL LOADER

```
module: {
    loaders: [
        {
            test: /\.es6$/,
            exclude: /node_modules/,
            loader: "babel-loader"
        }
    ]
},
resolve: {
    extensions: ['', '.js', '.es6']
}
```

PRELOADERS

- Just like loaders but ...
- Before executing your loaders
- Before building your bundles

JSHINT

```
$ npm install jshint jshint-loader --save-dev
```

- `.jshintrc` is required and contains `{}` by default

INTEGRATING JSHINT

```
module: {
    preLoaders: [
        {
            test: /\.js$/,
            exclude: /node_modules/,
            loader: "jshint-loader"
        }
    ]
}
```

PLUGINS

- Inject custom build steps (like a grunt task)
- Do things you can't do with loaders
- Work on the entire bundle

EXAMPLE: PROVIDE JQUERY

Install

```
$ npm install jquery --save
```

Make the jQuery object visible in every module, without requiring jQuery.

```
var webpack = require('webpack');
plugins: [
     new webpack.providePlugin({
          $: “jquery",
          jQuery: “jquery",
          “window.jQuery”: “jquery"
     })
]
```

EXAMPLE: ADD BANNER (TIMESTAMP, …)

Install

```
$ npm install timestamp-webpack-plugin --save-dev
```

```
var webpack = require('webpack');
var TimestampWebpackPlugin = require('timestamp-webpack-plugin');
plugins: [
    new TimestampWebpackPlugin({
        path: __dirname,
        filename: 'timestamp.json'
    }),

    new webpack.BannerPlugin("************\nGenerated by web pack\n***********\n")
]
```

START SCRIPTS

No need to run **`webpack jibber jabber`** all the time

Use **`package.json`** to define a start script

```
"scripts": {
        "start": "webpack-dev-server --inline"
    }
```

Run

```
$ npm start
```

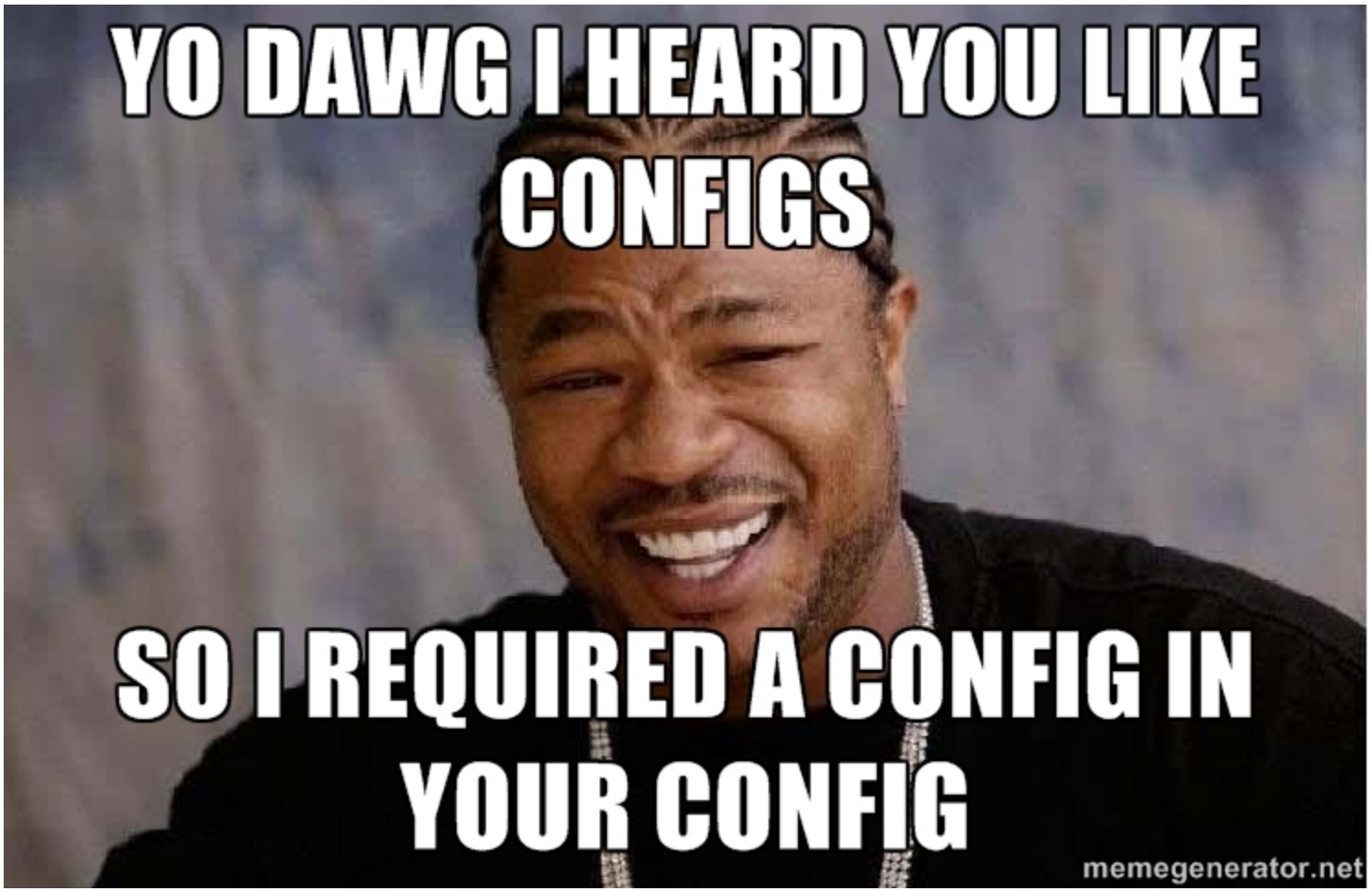
PRODUCTION BUILDS?

Separate config files

Not easy to maintain

Guaranteed headache

RIGHT?


YO DAWG I HEARD YOU LIKE CONFIGS
SO I REQUIRED A CONFIG IN YOUR CONFIG
memegenerator.net

STRIP THINGS

Strip things from your files before going to production

```
$ npm install strip-loader --save-dev
```

MINIMISE + UGLIFY

Built into webpack

Use the **-p** flag

```
$ webpack -p
```

WEBPACK-PRODUCTION.CONFIG.JS

```
var WebpackStrip = require('strip-loader');
var devConfig = require('./webpack.config.js');
var stripLoader = {
    test: [/\.js$/, /\.es6$/],
    exclude: /node_modules/,
    loader: WebpackStrip.loader('console.log', 'perfLog')
}
devConfig.module.loaders.push(stripLoader);
module.exports = devConfig;
```

RUN IT

```
$ webpack --config webpack-production.config.js -p
```

SCRIPT IT

Add in **`package.json`**

```
"scripts": {
    "prod": "webpack --config webpack-production.config.js -p"
}
```

Run

```
$ npm run prod
```

DEBUGGING

Source maps

```
$ webpack -d
```

Debugger statement

```
debugger;
```

MULTIPLE BUNDLES

- Multiple HTML pages
- Lazy loading
- Resources per page

MULTIPLE BUNDLES

```
module.exports = {
    context: path.resolve('js'),
    entry: {
        about: './about_page.js',
        home: './home_page.js',
        contact: './contact_page.js'
    },
    output: {
        filename: "[name].js"
    }
}
```

SHARED CODE

Using the **`CommonsChunkPlugin`** which generates an extra chunk, which contains common modules shared between entry points.

```
var webpack = require('webpack');
var commonsPlugin = new webpack.optimize.CommonsChunkPlugin('shared.js');

module.exports = {
    context: path.resolve('js'),
    entry: {
        about: './about_page.js',
        home: './home_page.js',
        contact: './contact_page.js'
    },
    output: {
        filename: "[name].js"
    },
    plugins: [
        commonsPlugin
    ]
}
```

INTEGRATING STYLES

- `css-loader`
- `style-loader`
- Require/import css files
- CSS integrated in bundle
- Less network requests
- `css-loader` adds inline style tags in the head of the page

IMPLEMENT

Install node packages

```
npm install css-loader style-loader --save-dev
```

Add loader

```
{
    test: /\.css$/,
    exclude: /node_modules/,
    loader: "style-loader!css-loader
}
```

Require

```
import {} from '../css/default.css'
```

USING SASS OR SCSS

Install node packages

```
npm install sass-loader --save-dev
```

Add loader

```
{
    test: /\.css$/,
    exclude: /node_modules/,
    loader: "style-loader!css-loader!sass-loader
}
```

Use by requiring .scss file

USING LESS

Install

```
npm install less-loader --save-dev
```

Add loader

```
{
    test: /\.css$/,
    exclude: /node_modules/,
    loader: "style-loader!css-loader!less-loader
}
```

Use by requiring .less file

SEPARATE CSS BUNDLES

Use separate files instead of inlined styles in the head.

```
npm install extract-text-webpack-plugin --save-dev
```

Include plugin and adapt **`output`** params

```
var ExtractTextPlugin = require('extract-text-webpack-plugin');

output: {
    path: path.resolve('build/'),
    publicPath: '/public/assets/'
}
```

Add plugin

```
plugins: [
    new ExtractTextPlugin('styles.css')
]
```

SEPARATE CSS BUNDLES

Adapt loaders

```
{
    test: /\.css$/,
    exclude: /node_modules/,
    loader: ExtractTextPlugin.extract("style-loader", "css-loader")
},
{
    test: /\.scss$/,
    exclude: /node_modules/,
    loader: ExtractTextPlugin.extract("style-loader", "css-loader!sass-loader")
}
```

AUTOPREFIXER

Install

```
npm install autoprefixer-loader --save-dev
```

Adapt loader

```
{
    test: /\.css$/,
    exclude: /node_modules/,
    loader: ExtractTextPlugin.extract("style-loader", "css-loader!autoprefixer-loader
},
{
    test: /\.scss$/,
    exclude: /node_modules/,
    loader: ExtractTextPlugin.extract("style-loader", "css-loader!autoprefixer-loader
}
```

IMAGES

Webpack can add images to your build

Based on the limit, webpack will:

- Base64 encode inline the image
- Reference the image

IMAGES

Install

```
npm install url-loader --save-dev
```

Add the loader

```
{
    test: /\.(png|jpg)$/,
    exclude: /node_modules/,
    loader: 'url-loader?limit=100000'
}
```

ADDING FONTS

Also uses the **`url-loader`** loader.

Add extensions to the loader

```
{
    test: /\.(png|jpg|ttf|eot)$/,
    exclude: /node_modules/,
    loader: 'url-loader?limit=100000'
}
```

CUSTOM LOADERS

Install

```
$ npm install json-loader strip-json-comments --save-dev
```

Create file **`strip.js`** in **`loaders`** folder.

```
var stripComments = require('strip-json-comments');

module.exports = function(source) {
     this.cacheable();
     console.log('source', source);
     console.log('strippedSource', stripComments(source));
     return stripComments(source);
}
```

CUSTOM LOADERS

Implement loader

```
loaders: [
    {
        test: /\.json$/,
        exclude: node_modules,
        loader: 'json_loader!' + path.resolve('loaders/strip')
    }
]
```

INTEGRATION WITH ANGULARJS & REACT

ANGULARJS AND WEBPACK

AMD, CommonJS, ES6 are real module systems

The Angular 1 way is not really module based

GET STARTED

```
$ npm install angular webpack babel-loader --save-dev
```

WON'T WORK

Webpack cannot use global module var like so

```
var app = angular.module('app', []);
app.controller();
```

WILL WORK

```
var angular = require('angular');
var app = angular.module('app', []);
angular.module('app').controller();
```

CREATE A DIRECTIVE, FACTORY, …

```
module.exports = function(app) {
     app.directive('helloWorld', function() {
          return {
               template: '<h1>Hello {{world}}</h1>'',
               restrict: 'E',
               controller: function($scope) {
                    $scope.world = 'world!';
               }
          }
     });
}
```

INCLUDE DIRECTIVE

```
var angular = require('angular');
var app = angular.module('app', []);
require('./hello-world')(app);
```

ORGANISING AN ANGULARJS APP

`index.js` in subfolder

```
module.exports = function(app) {
     require('./hello-world')(app);
     require('./hello-jworks')(app);
}
```

In `app.js`

```
require('./bands')(app)
```

ADDING TEMPLATES

Install

```
$ npm install raw-loader --save-dev
```

Add loader

```
{
      test: /\.html$/,
      exclude: /node_modules/
      loader: 'raw-loader'
}
```

ADDING TEMPLATES

```
{
    template: require('./hello-world.html')
}
```

This way you can bundle the templates and prevent XHR requests.

REACT AND WEBPACK

Install

```
$ npm install react --save
$ npm install babel-preset-react --save-dev
```

Add **.babelrc** file

```
{ "presets": ["es2015", "react"] }
```

Ready

THANKS FOR WATCHING!

Now kick some ass!


JWORKS
POWERED BY ORDINA