Tooling for building React applications

What are we going to cover

Create-React-App

Babel

NPM

Webpack

TypeScript

Gulp

ESLint

Create-React-App

Create React apps with **no build configuration**

The **react-scripts** package does most of the work
  - Uses Babel, Webpack and ESLint under the hood.

Getting started

```
// Install create-react-app

npm install -g create-react-app


// To create a new application

create-react-app my-new-app

cd my-new-app

npm start

Open browser at http://localhost:3000
```

Babel

Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Babel

Babel is an **ECMAScript compiler**

It takes flavors of JavaScript and transpiles it down to **standard ECMAScript 5**
- Supported by all modern browsers

The React team depends on Babel to **transpile JSX** code to JavaScript
- Sebastian McKenzie the author is now employed by Facebook

Babel has many **presets** for common uses cases
- New in Babel 6

Transpiling React code requires the **react preset**
- The es2015 and state-0 presets are frequently added

Cofiguring Babel

Use a .babelrc file to configure babel

```
{
    presets: [
        [
            "env",
            {
                targets: {
                    ie: 9
                }
            }
        ],
        "react"
    ]
}
```

Babel converts JSX to JavaScript

Input

```
import React, { Component} from 'react';


export class Greeter extends Component {

    render() {

        return (

            <div>Hello {this.props.name}</div>

        );

    }

}
```

Babel converts JSX to JavaScript

Output

```
var Greeter = exports.Greeter = function (_Component) {

  _inherits(Greeter, _Component);

  function Greeter() {

    _classCallCheck(this, Greeter);

    return _possibleConstructorReturn(this,

        (Greeter.__proto__ || Object.getPrototypeOf(Greeter)).apply(this, arguments));

  }

  _createClass(Greeter, [{

    key: 'render',

    value: function render() {

      return _react2.default.createElement(

        'div', null, 'Hello ', this.props.name);

    }

  }]);


  return Greeter;

}(_react.Component);
```

NPM

NPM

The **Node Package Manager**
- ◦ Originally intended for Node.js development
- ◦ The default package manager for most JavaScript these days

React is distributed using NPM
- ◦ As is Babel and all other packages in this module

NPM is normally installed with **Node.js**
- ◦ But developed and versioned separately

The *NPM* command is used for all actions
- ◦ Some editors have tooling that hide the NPM command

NPM Basics

Each module or application has a **package.json**
- Create a new one using ***NPM INIT***

Each module can list other modules it **depends on**
- NPM INSTALL <package> --SAVE
- These modules are installed from the NPM repository

There is a difference between **runtime** and **development dependencies**
- development dependencies are only needed during development
    - Packages like Babel, Browserify, ESLint etc
- Use –SAVE-DEV

Use **NPM INSTALL** to install all packages listed in the package.json

NPM Commands

The **package.json** file defines what a number of NPM commands do
- ◦ Useful to help developers new to a module

NPM TEST is the standard command to run the modules unit tests
- ◦ CI servers like [Travis](#) will download the code, run ***npm install*** and ***npm test*** to test your code

NPM START is the standard command to start the application
- ◦ Can start any executable

NPM and React

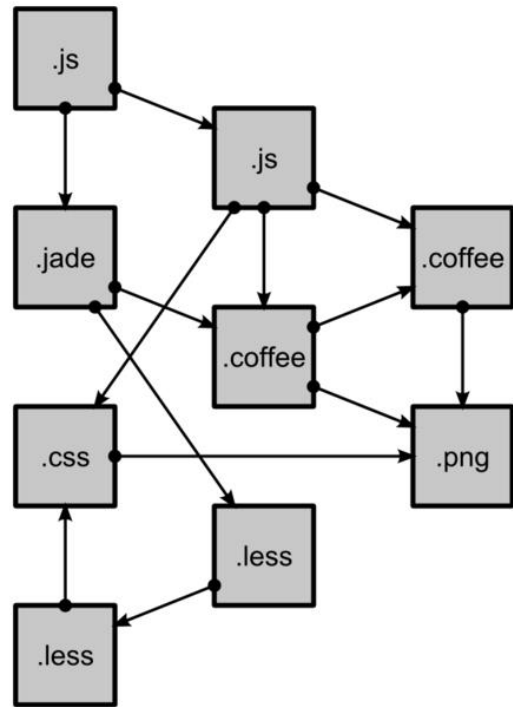NPM is the normal way to include **React**
- ◦ Then use tools like Babel and Webpack to bundle everything together
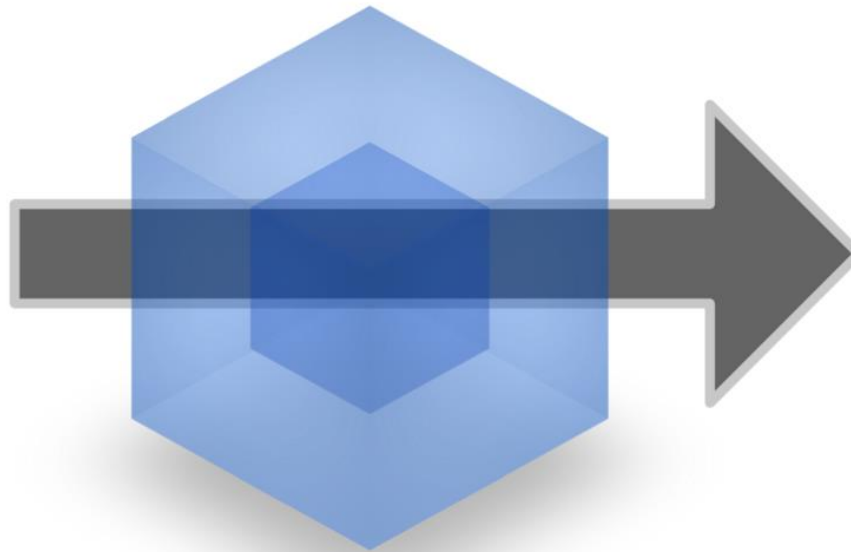
You can skip NPM if you really want to
- ◦ Download the React starter kit and reference the prebuilt copies of React and React DOM
- ◦ Not the recommended approach!
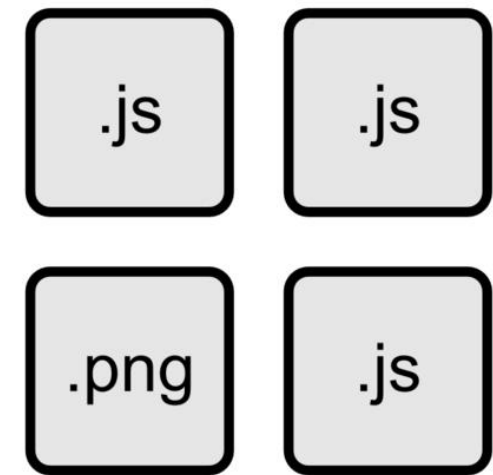
npm install --save react react-dom

Webpack

modules
with dependencies

webpack
MODULE BUNDLER

static
assets

Webpack

Webpack is an extremely flexible **module bundler**
- Much more advanced then Browserify

Some of the **advantages** of Webpack
- Fast incremental builds
- Hot reloading of changes
- Use different module styles as needed

Can **bundle different artifacts** together
- JavaScript
- CSS
- Images
- Webpack is very popular in the React community

Webpack

Loaders let you preprocess files
- Babel-loader will transpile code using Babel
- TS-loader will compile code using TypeScript
- LESS-loader or SASS-loader and CSS-loader preprocess and bundle CSS

Plugins can be used to control Webpack
- Bundle output into multiple bundles
- Hot module replacement
- Run Uglify over the code
- Add a banner to each module included
- etc

Webpack configuration

```
var webpackConfig = {

  entry: "./app.js",

  output: {

    filename: "./dist/app-bundle.js"

  },

  module: {

    rules: [{

        test: /\.jsx?$/,

        exclude: [/node_modules/],

        loaders: ["babel-loader"]

    }, {

        test: /\.less$/,

        loaders: ["css", "less"]

    }

    ]

  }

};
```

Webpack Analyse Tool

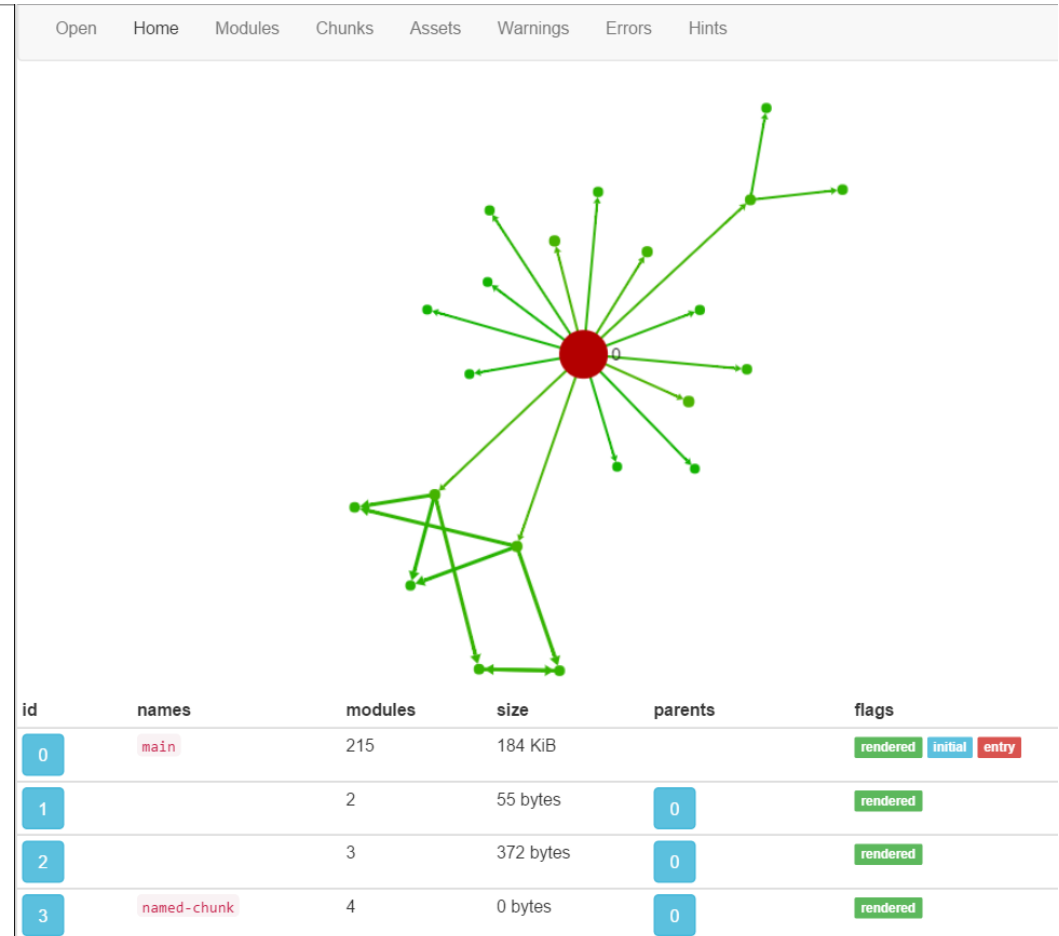A useful tool to check the Webpack results
- https://webpack.github.io/analyse
- Run Webpack with --profile --json as input

Shows everything Webpack bundled
- Chunks are about grouping modules
- Modules being included

Diagnostics information
- Warnings and errors to fix
- Hint that can make Webpack better

TypeScript

TypeScript

React is often written using JSX and JavaScript
- ◦ Frequently combined with ECMAScript 2015 features

JSX and JavaScript are **not compiled**
- ◦ This can lead to errors you don't notice until later

The code is not checked until you run it
- ◦ Either in the browser or using unit tests

TypeScript will let you check your code with a **compiler**
- ◦ Prevents mistakes due to typing errors or incorrect type

TypeScript

Requires **TypeScript 1.6** or later
- Use TSX as the file extension

Get type **definition files** from [DefinitelyTyped](#)
- Start with react.d.ts and react-dom.d.ts

No additional libraries needed at runtime
- It is still all just standard JavaScript

TypeScript example

```typescript
interface GreeterProps {

  name: string;

}

interface GreeterState {

  clickCount: number;

}

class Greeter extends Component<GreeterProps, GreeterState> {

  constructor(props: GreeterProps) {

    super(props);

    this.state = { clickCount: 0 };

  }

  private clicked() {

    this.setState({clickCount: this.state.clickCount + 1});

  }

  render() {

    return (<div>

        <h1>Hello {this.props.name}</h1>

        <button onClick={this.clicked}>Click me</button>

      </div>);

  }

}
```

Gulp

Gulp

Used to automate the **whole workflow**
- ◦ NPM commands are often used to start Gulp tasks

Write **tasks** that need to be done
- ◦ Running Gulp will start the *default* task
- ◦ Each task can depend on other tasks being finished

Some tasks can be done using either Gulp or Webpack
- ◦ For example ESLint
- ◦ Choose whatever feels best for you

ESLint

ESLint

The pluggable linting utility for JavaScript and JSX

Get Started »

ESLint

It is easy to make mistakes in JavaScript
- Most errors don't show up until you execute the code
  - And some are even then really subtle
- There is no compiler to catch common error

Static analysis of the code can find quite a few issues
- But not all of them

ESLint is a popular way of checking JavaScript code
- More configurable than JSLint or JSHint

There are many popular rule **configurations** available
- The AirBNB configuration is a popular starting point

Configuring ESLint

Create a **.eslintrc** file to configure ESLint

- ◦ ESLint can also be configured using command line parameters
- ◦ Or with a parameter object when using Gulp

Configuring ESLint

```
{
  "extends": "airbnb",
  "env": {
    "browser": true,
    "es6": true
  },
  "parserOptions": {
    "ecmaVersion": 6
  },
  "rules": {
    "no-console": "off",
    "strict": ["error", "global"],
    "curly": "warn"
  }
}
```

ESLint

ESLint can be used with Gulp
- Use **gulp-eslint**

ESLint can also be use from Webpack
- Use eslint-loader as a module preLoaders

ESLint can be run as an NPM command
- Or using a global NPM install
- Useful for running ESLint with --fix

ESLint can also be extended with custom **rules**
- Add specific rules for React using eslint-plugin-react
- Contains many useful rules that can be configured

Consider using TSLint to check your TypeScript
- Even with a compiler using a linter is still useful

Custom rule configuration

```
{
    parser: "babel-eslint",
    extends: "airbnb",
    plugins: ["react"],
    rules: {
        // warn if indentation is not 4 spaces
        indent: [1, 4],
        // error if missing props validation
        "react/prop-types": 2
    }
}
```

Conclusion

Babel is the standard way to transpile React JSX into JavaScript
  ◦ TypeScript is a great alternative if you like type safety and compile time checking

NPM is used to install a JavaScript modules needed during development
  ◦ React and ReactDOM are distributed using NPM

Webpack is the most common way to bundle your code with all NPM modules

Gulp is a great general purpose task runner that is often used

ESLint is a great way of statically checking JavaScript and JSX for common mistakes.
  ◦ TSLint is a great alternative if you are using TypeScript instead of JavaScript