Code.Hub

The first Hub for Developers

Ztoupis Konstantinos

Babel/Webpack

Code.Learn Program:
React

Javascript

- a language is being updated faster than ever before
- browsers and other JavaScript engines (node.js) haven't quite caught up with these changes

How can I use these features today?

What's new in ES2015?

- many of the new things available can already be accomplished in ES5
- a way to write cleaner more readable code
- actually quite a large list of changes and features
- cutting down on the size of your applications

CoffeeScript

- a little language that compiles into JavaScript
- an attempt to expose the good parts of JavaScript in a simple way
- the golden rule is: *“It’s just JavaScript.”*

CoffeeScript

CoffeeScript

```coffeescript
# Conditions:
number = -42 if opposite

# Functions:
square = (x) -> x * x

# Arrays:
list = [1, 2, 3, 4, 5]

# Objects:
math =
  root:   Math.sqrt
  square: square
  cube:   (x) -> x * square x

# Splats:
race = (winner, runners...) ->
  print winner, runners

# Existence:
alert "I knew it!" if elvis?

# Array comprehensions:
cubes = (math.cube num for num in list)
```

```javascript
number = 42;

opposite = true;

if (opposite) {
  // Conditions:
  number = -42;
}

// Functions:
square = function(x) {
  return x * x;
};

// Arrays:
list = [1, 2, 3, 4, 5];

// Objects:
math = {
  root: Math.sqrt,
  square: square,
  cube: function(x) {
    return x * square(x);
  }
};

// Splats:
race = function(winner, ...runners) {
  return print(winner, runners);
};

if (typeof elvis !== "undefined" && elvis
!== null) {
  // Existence:
  alert("I knew it!");
}

// Array comprehensions:
cubes = (function() {
  var i, len, results;
  results = [];
  for (i = 0, len = list.length; i < len;
i++) {
    num = list[i];
    results.push(math.cube(num));
  }
  return results;
})();
```

TypeScript

- a typed superset of JavaScript that compiles to plain JavaScript
- runs on any browser, in Node.js, or in any JavaScript engine that supports ECMAScript 3 (or newer)

TypeScript

TypeScript

```
export class Entity {
  private _id: number;
  private _title: string;
  private _creationDate: Date;

  constructor(id: number, title: string) {
    this._id = id;
    this._title = title;
    this._creationDate = new Date();
  }

  get id(): number {
    return this._id;
  }

  get title(): string {
    return this._title;
  }

  set title(title: string) {
    this._title = title;
  }

  get creationDate(): Date {
    return this._creationDate;
  }
}
```

CoffeeScript - TypeScript

JS

```
// Good 'ol JS
function printSecret ( secret ) {
    console.log(`${secret}. But don't tell anyone.`);
}

printSecret("I don't like CoffeeScript");
```

```
# CoffeeScript
printSecret (secret) =>
    console.log '#{secret}. But don't tell anyone.'

printSecret "I don't like JavaScript."
```

```
// TypeScript -- JavaScript, with types and stuff
function printSecret ( secret : string ) {
    console.log("${secret}. But don't tell anyone.");
}

printSecret("I don't like CoffeeScript.");
```

Anything you can write in JavaScript, you can write in CoffeeScript or TypeScript

TypeScript

Code.Hub

The problem

JavaScript environments only understand . . .
JavaScript

```
var getMyHTML = function () {
  return <p>Hello <bold>World</bold></p>;
}
```


```
var getMyHTML = function getMyHTML() {
  return React.createElement(
    "p",
    null,
    "Hello ",
    React.createElement(
      "bold",
      null,
      "World"
    )
  );
};
```

Transpilers

tools that read source code written in one programming language, and produce the equivalent code in same language

In Defense of Transpilers

In the case of languages that target JavaScript, it's largely a matter of preference or background

JavaScript Today

browser compatibility issues: its not as simple as writing JavaScript that runs everywhere

different JavaScript engine

V8

SpiderMonkey

Chakra

Code.Hub

Transpilers

- allow us to write compile-to-JavaScript languages, like CoffeeScript, TypeScript
- let us use new and potential JavaScript features, reliably
- contribute to the development of the ECMAScript specification

BABEL

- a free and open-source JavaScript compiler and configurable transpiler used in web development.
- popular tool for using the newest features of the JavaScript programming language
- is downloaded 5 million times a month

BABEL

- pronounced "babble"
- a community-driven project used by many companies and projects
- is maintained by a group of volunteers
- a tool that helps you write code in the latest version of JavaScript
- your supported environments don't support certain features natively -> Babel will help you compile those features down to a supported version

BABEL

IN

```
// ES2015 arrow function
[1, 2, 3].map((n) => n + 1);
```

OUT

```
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

BABEL

IN

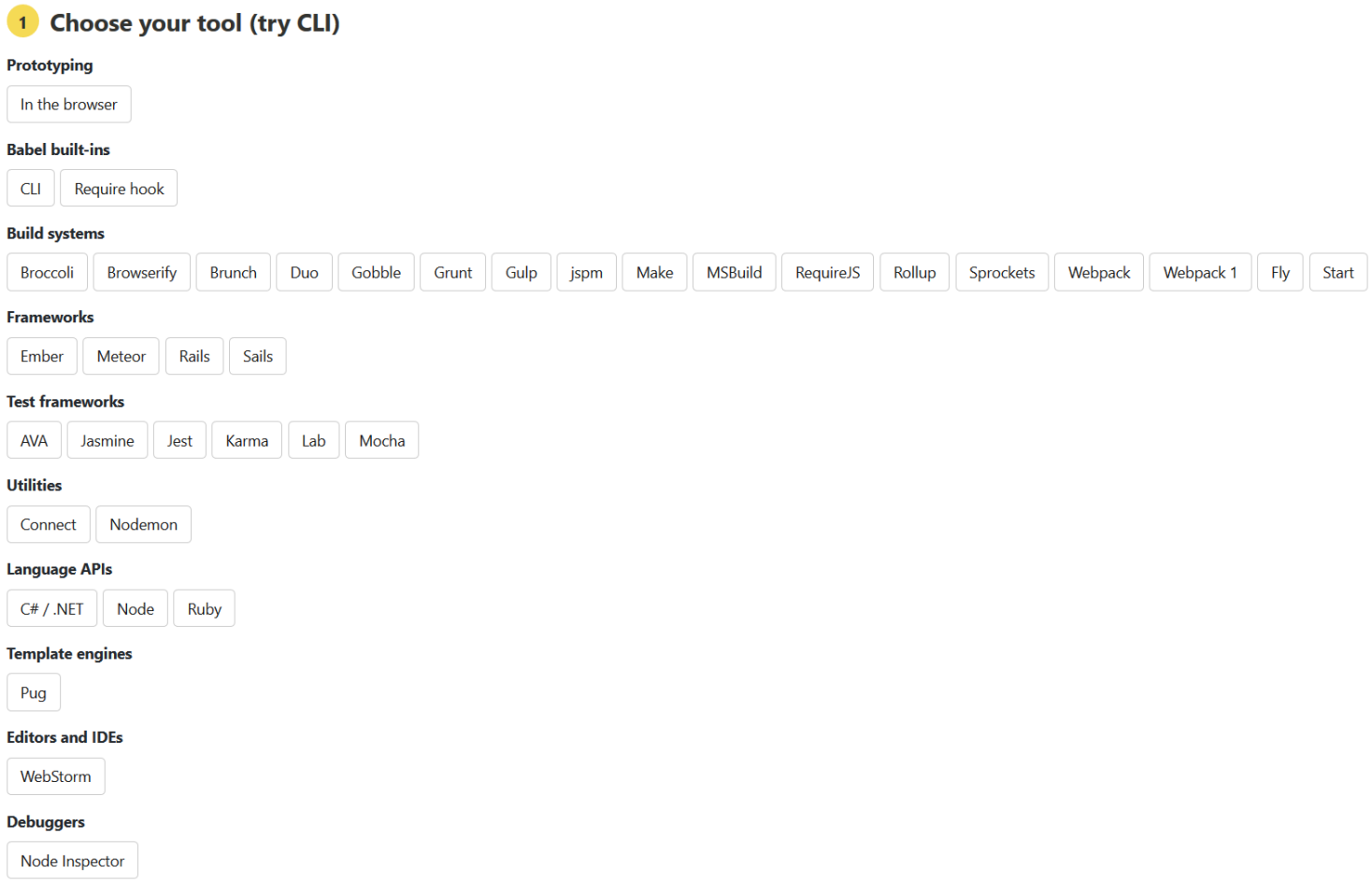
```
class Planet {

  constructor (mass, moons) {
    this.mass  = mass;
    this.moons = moons || 0;
  }

  reportMoons () {
    console.log(`I have ${this.moons} moons.`)
  }

}

// Yeah, Jupiter really does have (at least) 67 moons.
const jupiter = new Planet('Pretty Big', 67);
jupiter.reportMoons();
```

OUT

Code.Hub

Setup Babel

instructions for pretty much every scenario that you can imagine from raw CLI to node.js to Meteor



Code.Hub

Setup Babel

- npm i --save-dev @babel/core @babel/preset-env @babel/preset-react
- configuration file called ”.babelrc”

```
{
    presets: ['es2015']
}
```

What's new in ES2015?

- large list of changes and features
- writing cleaner more readable code

What's new in ES2015?

- Block Scope
- Arrow Functions
- Default Parameters
- Rest Parameters
- Spread operator
- Destructuring
- Template String Literals
- Classes
- Maps and Sets
- Promises
- Symbols
- Iterables and Iterators
- Generators

Block Scope

Using the ***let*** keyword allows you to scope a variable to the block that it's in, rather than just the function

```
var links = [];
for(var i=0; i<5; i++) {
    links.push({onclick: function() {
console.log('link: ', i); }});
}

links[0].onclick(); // link: 5
links[1].onclick(); // link: 5
```

```
var links = [];
for(let i=0;i<5;i++) {
    links.push({onclick: function() {
console.log('link: ', i); }});
}

links[0].onclick(); // link: 0
links[1].onclick(); // link: 1
```

Block Scope

```
function test() {
  let foo = 1;
  // will execute ?
  if (foo === 1) {
    let foo = 22;
    console.log(foo); ?
  }
```

```
  // will execute ?
  if (foo === 22) {
    let foo = 33;
    console.log(foo);
  }

  console.log(foo);  ?
}
test();
```

Block Scope

```
const msg = 'hello world';

msg = 123; // will silently fail, msg not changed

var msg = 123; // Syntax error: msg already defined
```

Another block scoped keyword is **const**. Constants are block scoped, and also they can only be defined once (within their scope).

Block Scope

Function and Class declarations are also block scoped

```
function makeAnimalClass(legs) {
  let AnimalClass;

  if (legs > 2) {
    class Animal {
      constructor(name) {
        this.name = name;
        this.legType = 'multiped';
      }
    }

    AnimalClass = Animal;
  } else {
    class Animal {
      constructor(name) {
        this.name = name;
        this.legType = 'biped';
      }
    }

    AnimalClass = Animal;
  }

  return AnimalClass;
}

var Animal = makeAnimalClass(4);
var dog = new Animal('dog');
console.log(dog); // {"name":"dog","legType":"multiped"}
```

Arrow Functions

Arrow functions are a new shorthand way of declaring functions that also share scope with their parent.

```
var addOne = a => a + 1;
addOne(1); // 2

var add = (a, b) => a + b;
add(1,2); // 3

var addLogged = (a, b) => {
    let c = a + b;
    console.log(a, '+', b, '=', c);
    return c;
};
addLogged(1, 2); // 1 + 2 = 3
```

Arrow Functions

```
function NumberX(number) {
  this.multiplier = number;
}

NumberX.prototype.number = function(numbers) {
  let result = Math.floor(numbers.reduce((i, j) => i + j * this.multiplier,
0));
  console.log(result);
  return result;
};

var total = new NumberX(2);
```

The scope of the arrow function is shared with the parent of said function

Type Annotations (Flow)

Babel can strip out type annotations!

```
npm install --save-dev @babel/preset-flow
```

```
// @flow
function square(n: number): number {
  return n * n;
}
```

Type Annotations (TypeScript)

```
npm install --save-dev @babel/preset-typescript
```



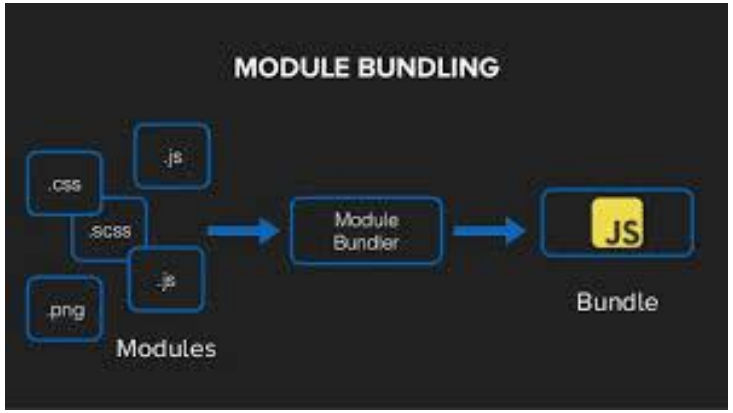
```
function Greeter(greeting: string)
{
  this.greeting = greeting;
}
```

Editors

- Atom
- Sublime Text 3
- Vim
- Visual Studio Code
- WebStorm

Popular editors support ES2015+ syntax highlighting out of the box, while some require installing additional extensions

Bundlers



allow us to package, compile, and organize the many assets and libraries needed for a modern web project

Bundlers

- Webpack
- Gulp
- Browserify
- NPM scripts
- Grunt

Webpack

- powerful open-source bundler and preprocessor that can handle a huge variety of different tasks
- is used to compile JavaScript modules

Webpack

- **static module bundler** for modern JavaScript applications
- builds a **dependency graph** which maps every module your project needs and generates one or more bundles

Webpack

Core Concepts:

- Entry
- Output
- Loaders
- Plugins

Pros of Webpack

- Great for working with singe-page apps
- Accepts both require() and import module syntaxes
- Allows for very advanced code splitting
- Hot Reload for quicker development with React, Vue.js and similar frameworks
- Most popular build tool according to the 2016 JavaScript survey

Cons of Webpack

- Not suitable for beginners in web development
- Working with CSS files, images, and other non-JS resources is confusing at first
- Documentation could be better
- Changes a lot, even most 2016 tutorials are already outdated

Installation

- npm or Yarn or another → npm install webpack --save-dev
- Node.js

```
"scripts": {
  "start": "webpack-dev-server --config ./webpack.config.js --mode development",
  "build": "webpack --mode production",
},
```

Entry

- an **entry point** indicates which module webpack should use to begin building out its internal **dependency graph**.
- webpack will figure out which other modules and libraries that entry point depends on directly and indirectly.

```
module.exports = {
  entry: './app/js/index.js‘
};
```

Output

The **output** property tells webpack where to emit the bundles it creates and how to name these files

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

module.exports = {
  entry: './app/js/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

Output

The only script file that we will link in our HTML:
<script src="./dist/bundle.js"></script>

Loaders

Loaders allow webpack to process other types of files and convert them into valid modules that can be consumed by your application and added to the dependency graph.

Loaders

```
module.exports = {
  entry: './app/js/index.js',
  output: {..//..},
  module: {
    rules: [{
      test: /\.js$/,
      exclude: /node_modules/,
      use: 'jshint-loader'
    }]  }
};
```

JSHint loader: catch all kinds of bad practices and errors

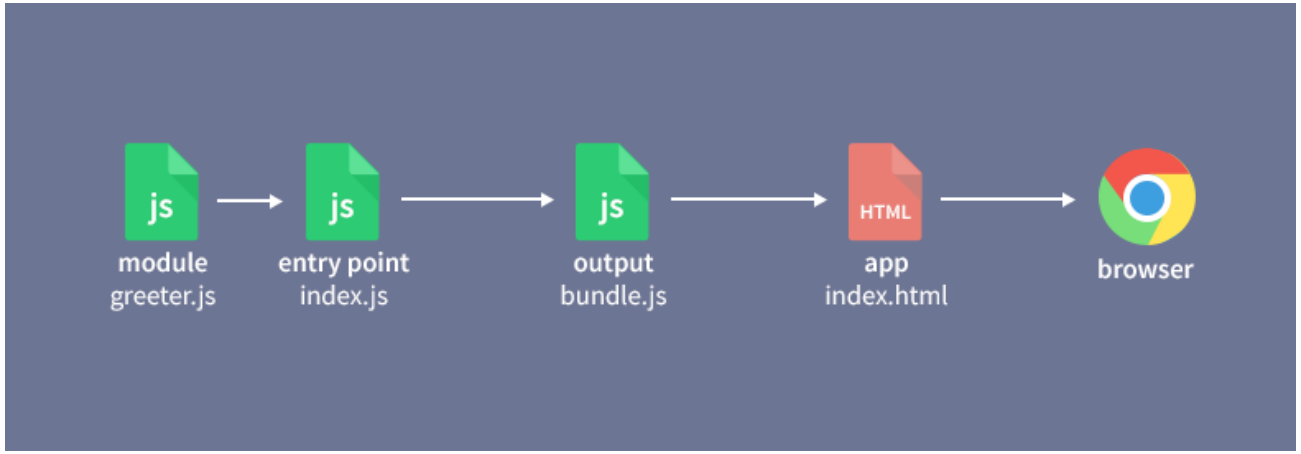
Modules

```
//greeter.js
function greet() {
  console.log('Have a great day!');
};
export default greet;
```

Webpack provides multiple ways to work with modules

```
//index.js
import greet from './greeter.js';
console.log("I'm the entry point");
greet();
```

Modules


js
module
greeter.js
js
entry point
index.js
js
output
bundle.js
HTML
app
index.html
browser

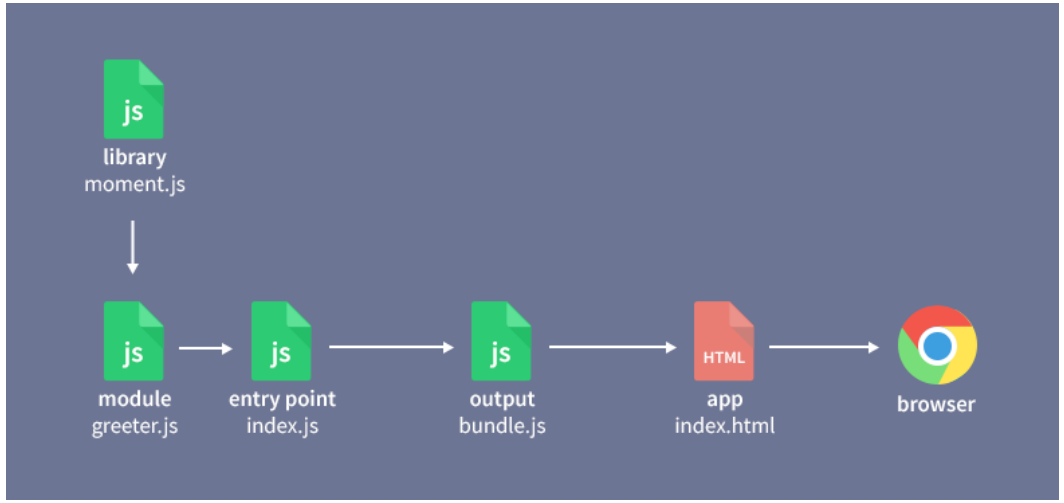
Requiring Libraries

```
//greeter.js
import moment from 'moment';
function greet() {
    var day = moment().format('dddd');
    console.log('Have a great ' + day + '!');
};
export default greet;
```

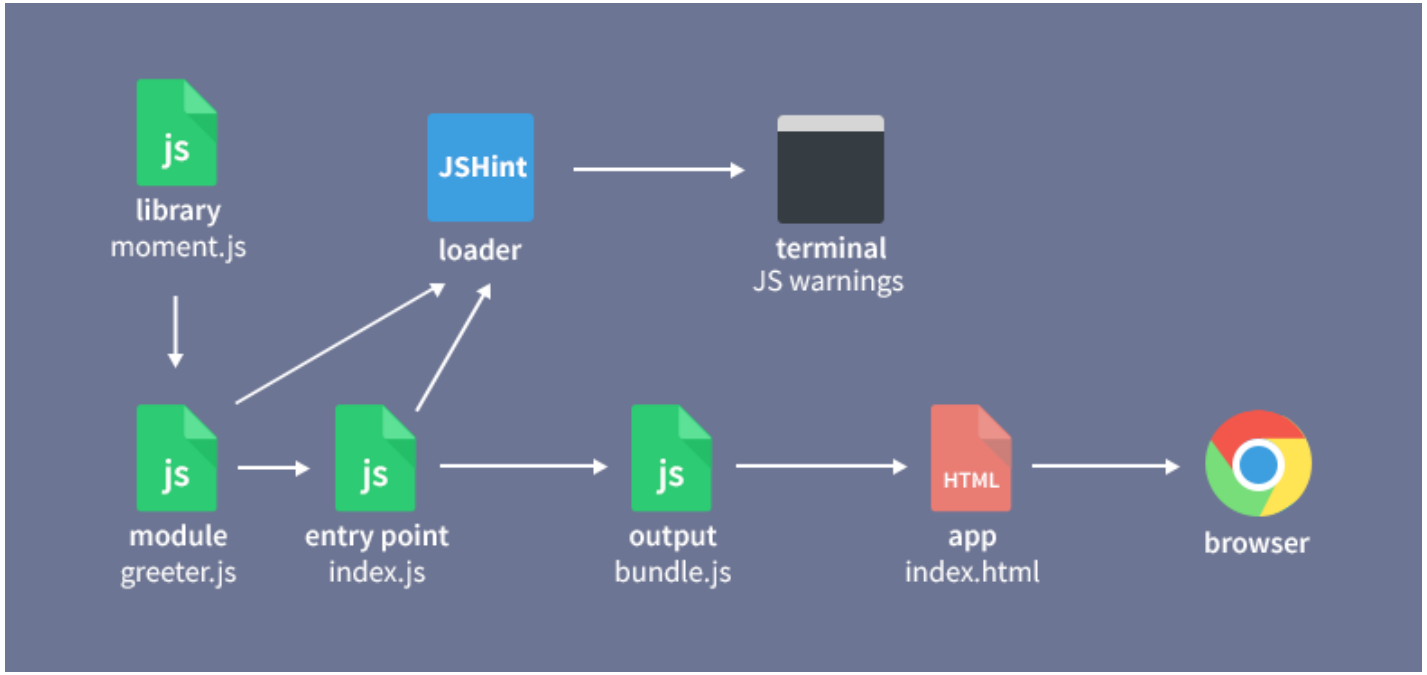
Adding moment library

```
index.js
import greet from './greeter.js';
console.log("I'm the entry
point");
```

Requiring Libraries



Loaders


js
library
moment.js
JSHint
loader
terminal
JS warnings
js
module
greeter.js
js
entry point
index.js
js
output
bundle.js
HTML
app
index.html
browser

Plugins

While loaders are used to transform certain types of modules, plugins can be leveraged to perform a wider range of tasks like bundle optimization, asset management and injection of environment variables.

Plugins

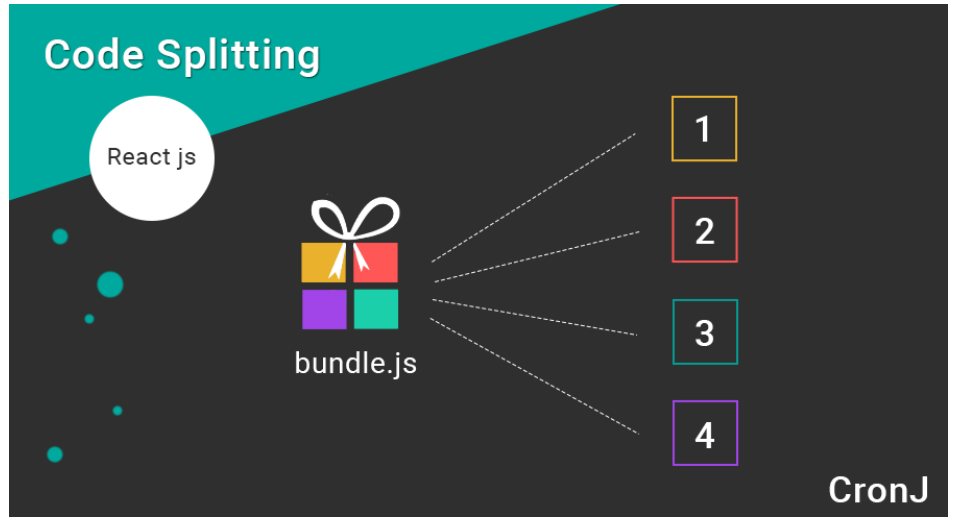
```
const HtmlWebpackPlugin = require(html-webpack-
plugin');
module.exports = {
  entry: './app/js/index.js',
  output: {..//..},
  module: {..//..},
  plugins: [new HtmlWebpackPlugin({
      template: './src/index.html'
    })]
```

Browser Compatibility

webpack supports all browsers that are ES5-compliant (IE8 and below are not supported)

support older browsers: load a polyfill

Code Splitting



Instead of having your application in one big bundle, you can have multiple bundles each loading asynchronously or in parallel

production and development mode

- a configuration file for development, for defining webpack dev server and other stuff
- a configuration file for production, for defining UglifyJSPlugin, sourcemaps and so on

production and development mode

```
"scripts": {
        "dev": "webpack --mode development",
        "build": "webpack --mode production"
}
```

npm run dev

A **not** minified bundle

npm run build

A minified bundle

Code.Hub

Babel and Webpack

3 Babel dependencies in order to use it with Webpack:

- *babel-loader*: interface between Babel and Webpack
- *babel-core*: understands how to read & parse code, and generate corresponding output
- *babel-preset-es2015*: rules for Babel on how to process ES2015 code and convert it into ES5

Babel and Webpack

```
module: {
  loaders: [{
    test: /\.js$/,
    loader: 'babel-loader',
    exclude: /node_modules/,
    query: {presets: ['es2015']}
  }]
}
```

CSS & Styling

Loaders to process our CSS:

- *css-loader*: knows how to process CSS imports - takes the imported CSS and loads the file contents
- *style-loader*: takes CSS data(from imports) and adds them to the HTML document

CSS & Styling

```
module: {
  loaders: [{..//..},
    {
      test: /\.css$/,
      loader : ['style-loader', 'css-loader']
    }
  ]
}
```

Pictures

Loaders to process our pictures:

- *image-webpack-loader*: will try to automatically compress large images
- *url-loader*: will inline the results from image-webpack-loader if the results are small, and include the image in the output directory if they are large

Pictures

```
module: {
  loaders: [{..//..}, {..//..}, {
        test: /\.png$/,
        loaders: [
          'url-loader?limit=5000',
          'image-webpack-loader'
        ]}
    ]
}
```