Chapter 12: How to setup a basic webpack, react and babel environment

Section 12.1: How to build a pipeline for a customized "Hello world" with images

Step 1: Install Node.js

The build pipeline you will be building is based in Node.js so you must ensure in the first instance that you have this installed. For instructions on how to install Node.js you can checkout the SO docs for that here

Step 2: Initialise your project as an node module

Open your project folder on the command line and use the following command:

```
npm init
```

For the purposes of this example you can feel free to take the defaults or if you'd like more info on what all this means you can check out this SO doc on setting up package configuration.

Step 3: Install necessary npm packages

Run the following command on the command line to install the packages necessary for this example:

```
npm install --save react react-dom
```

Then for the dev dependencies run this command:

```
npm install --save-dev babel-core babel-preset-react babel-preset-es2015 webpack babel-loader css-loader style-loader file-loader image-webpack-loader
```

Finally webpack and webpack-dev-server are things that are worth installing globally rather than as a dependency of your project, if you'd prefer to add it as a dependency then that will work to, I don't. Here is the command to run:

```
npm install --global webpack webpack-dev-server
```

Step 3: Add a .babelrc file to the root of your project

This will setup babel to use the presets you've just installed. Your .babelrc file should look like this:

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

Step 4: Setup project directory structure

Set yourself up a directory stucture that looks like the below in the root of your directory:

```
|- node_modules
|- src/
  |- components/
  |- images/
  |- styles/
  |- index.html
```

```
  |- index.jsx
|- .babelrc
|- package.json
```

NOTE: The `node_modules`, `.babelrc` and `package.json` should all have already been there from previous steps I just included them so you can see where they fit.

Step 5: Populate the project with the Hello World project files

This isn't really important to the process of building a pipeline so I'll just give you the code for these and you can copy paste them in:

src/components/HelloWorldComponent.jsx

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

class HelloWorldComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {name: 'Student'};
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    this.setState({name: e.target.value});
  }

  render() {
    return (
      <div>
        <div className="image-container">
          <img src="./images/myImage.gif" />
        </div>
        <div className="form">
          <input type="text" onChange={this.handleChange} />
          <div>
            My name is {this.state.name} and I'm a clever cloggs because I built a React build
pipeline
          </div>
        </div>
      </div>
    );
  }
}

export default HelloWorldComponent;
```

src/images/myImage.gif

Feel free to substitute this with any image you'd like it's simply there to prove the point that we can bundle up images as well. If you provide your own image and you name it something different then you'll have to update the `HelloWorldComponent.jsx` to reflect your changes. Equally if you choose an image with a different file extension then you need to modify the `test` property of the image loader in the webpack.config.js with appropriate regex to match your new file extension..

src/styles/styles.css

```
.form {
  margin: 25px;
```

```
  padding: 25px;
  border: 1px solid #ddd;
  background-color: #eaeaea;
  border-radius: 10px;
}

.form div {
  padding-top: 25px;
}

.image-container {
  display: flex;
  justify-content: center;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Learning to build a react pipeline</title>
</head>
<body>
  <div id="content"></div>
  <script src="app.js"></script>
</body>
</html>
```

index.jsx

```
import React from 'react';
import { render } from 'react-dom';
import HelloWorldComponent from './components/HelloWorldComponent.jsx';

require('./images/myImage.gif');
require('./styles/styles.css');
require('./index.html');

render(<HelloWorldComponent />, document.getElementById('content'));
```

Step 6: Create webpack configuration

Create a file called webpack.config.js in the root of your project and copy this code into it:

webpack.config.js

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

var config = {
  context: path.resolve(__dirname + '/src'),
  entry: './index.jsx',
  output: {
    filename: 'app.js',
    path: path.resolve(__dirname + '/dist'),
  },
  devServer: {
    contentBase: path.join(__dirname + '/dist'),
    port: 3000,
    open: true,
```

```
  },
  module: {
    loaders: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        loader: 'babel-loader'
      },
      {
        test: /\.css$/,
        loader: "style!css"
      },
      {
        test: /\.gif$/,
        loaders: [
          'file?name=[path][name].[ext]',
          'image-webpack',
        ]
      },
      { test: /\.(html)$/,
        loader: "file?name=[path][name].[ext]"
      }
    ],
  },
};

module.exports = config;
```

Step 7: Create npm tasks for your pipeline

To do this you will need to add two properties to the scripts key of the JSON defined in the package.json file in the root of your project. Make your scripts key look like this:

```
  "scripts": {
    "start": "webpack-dev-server",
    "build": "webpack",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

The `test` script will have already been there and you can choose whether to keep it or not, it's not important to this example.

Step 8: Use the pipeline

From the command line, if you are in the project root directory you should now be able to run the command:

```
npm run build
```

This will bundle up the little application you've built and place it in the `dist/` directory that it will create in the root of your project folder.

If you run the command:

```
npm start
```

Then the application you've built will be served up in your default web browser inside of a webpack dev server instance.