**What is Gulp?**

Gulp is a [task runner](http://jpsierens.com/task-runners-a-comparison-between-grunt-gulp-broccoli-and-mimosa/). It automates tasks that you have to do repeatedly when developing. A concrete example is having to compile your react component’s JSX to plain javascript every time you change something.

**How do I get it?**

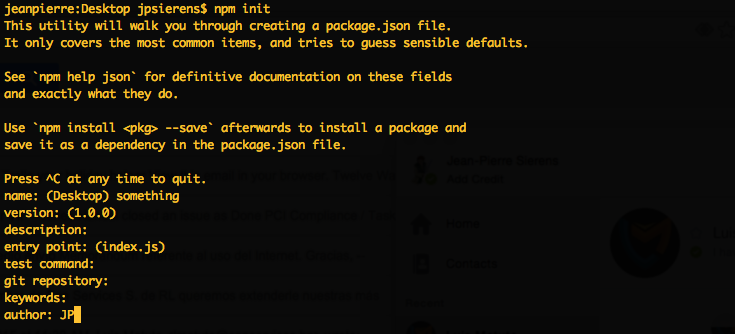
Gulp is very easy to get. First of, install nodejs and npm. If you have them, type this in your terminal:

npm install -g gulp

The -g is for global, so that it is available everywhere.

**package.json**

Before starting any project that will use gulp you need a package.json file in your project. Usually it is located in the root of the project. You don’t write this file, but rather **generate** it. Go to your terminal and in the root of your project type npm init this should take you to several steps to complete the package.json. You can just press enter on all steps to generate something quick.

[](http://jpsierens.com/wp-content/uploads/2015/12/Screen-Shot-2015-12-21-at-8.43.44-AM.png)how npm init should look like

**gulpfile.js**

The gulpfile is the file where you tell gulp what to do. It is usually located in the root of your project. Anything related to gulp goes there. It’s the only file you need to have for using gulp.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | /\*  \* Task Automation to make my life easier.  \* Author: Jean-Pierre Sierens  \* ===========================================================================  \*/    // declarations, dependencies  // ----------------------------------------------------------------------------  var gulp = require('gulp');  var browserify = require('browserify');  var source = require('vinyl-source-stream');  var gutil = require('gulp-util');  var babelify = require('babelify');    // External dependencies you do not want to rebundle while developing,  // but include in your application deployment  var dependencies = [  'react',     'react-dom'  ];  // keep a count of the times a task refires  var scriptsCount = 0;    // Gulp tasks  // ----------------------------------------------------------------------------  gulp.task('scripts', function () {      bundleApp(false);  });    gulp.task('deploy', function (){  bundleApp(true);  });    gulp.task('watch', function () {  gulp.watch(['./app/\*.js'], ['scripts']);  });    // When running 'gulp' on the terminal this task will fire.  // It will start watching for changes in every .js file.  // If there's a change, the task 'scripts' defined above will fire.  gulp.task('default', ['scripts','watch']);    // Private Functions  // ----------------------------------------------------------------------------  function bundleApp(isProduction) {  scriptsCount++;  // Browserify will bundle all our js files together in to one and will let  // us use modules in the front end.  var appBundler = browserify({       entries: './app/app.js',       debug: true     })    // If it's not for production, a separate vendors.js file will be created  // the first time gulp is run so that we don't have to rebundle things like  // react everytime there's a change in the js file     if (!isProduction && scriptsCount === 1){     // create vendors.js for dev environment.     browserify({  require: dependencies,  debug: true  })  .bundle()  .on('error', gutil.log)  .pipe(source('vendors.js'))  .pipe(gulp.dest('./web/js/'));     }     if (!isProduction){     // make the dependencies external so they dont get bundled by the  // app bundler. Dependencies are already bundled in vendor.js for  // development environments.     dependencies.forEach(function(dep){     appBundler.external(dep);     })     }       appBundler     // transform ES6 and JSX to ES5 with babelify     .transform("babelify", {presets: ["es2015", "react"]})      .bundle()      .on('error',gutil.log)      .pipe(source('bundle.js'))      .pipe(gulp.dest('./web/js/'));  } |

Let’s go over the code. First we declare the plugins we are going to use.

* **Gulp**: Well, you know what this one is for: The task runner itself.
* **Browserify**: Bundles your javascript files together and let’s you use modules that can be exported and imported in your javascript code.
* **vinyl-source-stream**: Plugin used for working with stream outputs. Need this to work with Browserify easily.
* **gulp-util**: Utility functions for gulp plugins, like nice logging.
* **babelify:** This is our transpiler. It converts ES6 and JSX to plain old javascript.v6.0+ of babelify must include presets in order to work. Basically they did the same as react and are embracing the unix philosophy of how to build tools even more. So that means having more plugins that do one thing and do that one thing well.

*Note: If you are using old gulpfiles babelify is going to cause pain for you, because of this change of separating the presets (like es6 and react) from the main plugin. You need to download the presets as well, more on this later.*

We then define an array of **dependencies** (react and react-dom) which we will use later to tell browserify not to bundle them together with the app files, so that browserify doesn’t have to process them on each save. Think about it, they don’t need to get bundled each time you save your js files, they are libraries, they won’t change.

After that we initialise a **counter** that goes up each time gulp detects a change in the js files. More on this later.

**Three gulp tasks** are defined afterwards. “**scripts**” is the task fired when there’s a change in the js file. “**deploy**” is when we want to deploy to production. “**watch**” is the task that actually watches for changes in the js files. Use “**gulp scripts”**in the terminal for example if you want to fire the scripts task.

Later on we define a fourth task, the **default** one. Here we tell gulp to first fire the scripts task and then to start watching. We fire this task with **“gulp”** on the terminal. This is normally the only one you should use unless you want to deploy.

Now, inside both the “scripts” and “deploy” task we just see a call to the same function “**bundleApp**“. This function is where the bundling and conversion from JSX and ES6 to plain javascript happens.

Inside the function we first increase the counter. We then initialise **browserify** with some configurations passed in. We keep a reference under the variable “appBundler”.

After, we have a conditional block that will only execute if it’s the first time you run the “scripts” task. Note that it won’t run under the “deploy” task. What we do inside is to create another browserify object and bundle all our libraries together and save them as “**vendors.js**“. As I explained before, we do this so that when developing, you don’t have to rebundle all the libraries again (unnecessarily) as they don’t change. When deploying, we don’t want to have a separate “vendors.js” file, we want to have everything together under “bundle.js”, and preferably uglified to maximize performance.

We then have another conditional block that only lets you in if you’re in development mode (“scripts” task fired). It will tell the “appBundler” object not to bundle the **dependencies** we defined earlier. They are already bundled in “vendors.js”.

Finally, we fire the “**appBundle**” function so that it first **transforms the JSX and ES6 (i.e. ES2015)**and later **bundle** everything together in “**bundle.js**“. If you deploy with **gulp deploy** in your terminal, ALL your js files will be bundled to “bundle.js” including the libraries. This way, in your production environment you only have 1 script to rule them all!

*Note: You can uglify the javascript files right after they are bundled, this will make your server happy. Search for the gulp-uglify plugin for more.*

**Installing Dependencies**

Now we are going to download the development dependencies we listed earlier. This can be done as easy as putting the following code in your terminal:

npm install --save-dev gulp browserify vinyl-source-stream gulp-util babelify babel-preset-es2015 babel-preset-react

You also need the dependencies for the project, whic1h is just React for now. Since React 0.14 is out, you now need to install two packages instead of one: react and react-dom.

npm install --save react react-dom

**React Components**

In the root of your project, create a folder called “app”. Then, inside it, create a file called “SearchableTable.js”.

Let’s use the same react component from one of the official react tutorials. It’s a filterable table but I transformed it to Javascript 6 and made it compatible with React v0.14. Put the following code in “SearchableTable.js”:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83 | /\*  \* Searchable Table  \* Author: Jean-Pierre Sierens  \* ===========================================================================  \*/    import React from 'react';    export default class SearchableTable extends React.Component {  constructor() {  super();  // Initial state of the component          this.state = {filterText: ''}      }      handleUserInput(filterText) {       // When there's a change in the state, the component and all its       // sub-components get updated.          this.setState({filterText: filterText});      }  render(){  return (  <div>  <SearchBar  filterText={this.state.filterText}                      onUserInput={this.handleUserInput.bind(this)}                  />  <Table  data={this.props.data}  filterText={this.state.filterText}  />  </div>  );  }  }    class SearchBar extends React.Component {  handleChange() {  // passing filter data up by using a callback          this.props.onUserInput(           // ref is like the id              this.refs.filterTextInput.value          );      }  render(){  return (              <form>                  <input                   type="text"                   placeholder="Search for one keyword..."                   ref="filterTextInput"                   value= {this.props.filterText}                   onChange= {this.handleChange.bind(this)}                  />              </form>          );  }  }    class Table extends React.Component {  render(){  let sections = [];  let data = this.props.data;  data.forEach(function(product){  if (product.name.indexOf(this.props.filterText) === -1) {  return;  }  sections.push(<Section key={product.name} data={product} />);  }.bind(this))  return(  <div>{sections}</div>  );  }  }    class Section extends React.Component {  render(){  return(  <div>  <p>{this.props.data.name} = {this.props.data.price} </p>  </div>  );  }  } |

Things to note on React v0.14 for this component:

* **ReactDOM**: ReactDOM is now the object that handles DOM operations for you component. This was separated from the react object to separate pure component definition with implementation (DOM or mobile) such as to pave the way for easir react + react-native integration.
* **DOM node refs**: To get the value of the input in React v0.13 and under you would have used “this.refs.filterTextInput.getDOMNode().value”. The getDOMNode() was removed, as now the filterTextInput is the actual node itself. so in React v0.14 you use “this.refs.filterTextInput.value”.

Let’s also define the data this app will use. Put the following code in a file called “data.js”, also under the “app” folder.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | export const data = [    {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},    {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},    {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},    {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},    {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},    {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}  ]; |

Now we need a file where we import the react component and pass it the data it’s going to use. Create a file called “app.js” also under the “app” folder and put the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\*========================================================================  \*/    import React from 'react';  import ReactDOM from 'react-dom';  import SearchableTable from './SearchableTable';  import {data} from './data';    // Filterable CheatSheet Component  ReactDOM.render( <SearchableTable data={data}/>, document.getElementById('searchableTable') ); |

This will be the file where you do other stuff for your app in the future. You see we are using ES6 in these files. For example we use “import” to import modules and other files.

**HTML**

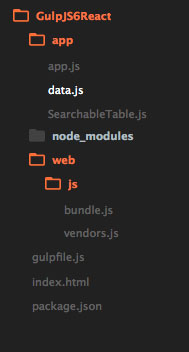
Our app needs some html to work. Let’s keep it minimal. Create a file called “index.html” on the root of your folder.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>React and ES6</title>  </head>  <body>  <!-- React element goes here -->  <div id="searchableTable"></div>    <!-- one script to rule them all (in production) -->  <script src="web/js/vendors.js"></script>  <script src="web/js/bundle.js"></script>  </body>  </html> |

Simple, we put up a container with id “searchableTable” which is where the component is going to mount on. We also load the two javascript files: vendors.js and bundle.js.

**Closing**

Your project structure should look like this:

[](http://jpsierens.com/wp-content/uploads/2016/01/projecttree.jpg)

In your terminal, type “gulp” in the root of the project to fire gulp and let it bundle everything when there’s a change in the js files.

Open the folder and open **index.html** in browser

[](http://jpsierens.com/wp-content/uploads/2016/01/projectexample.jpg)

> npm install -g gulp

>npm install --save-dev gulp browserify vinyl-source-stream gulp-util babelify babel-preset-es2015 babel-preset-react

>npm install --save react react-dom

>gulp

Open the folder and open **index.html** in browser