

My blog is now available at www.christianalfoni.com

” Full stack web
application *enthusiast*,
running the web
standards marathon ”

Tweet

41



15 Aug 2014

React JS and a browserify workflow

I have been working with **React JS** for a few days and I must say I am impressed. You can not really compare it to complete frameworks like **Angular JS** or **Ember JS**, but at the same time it is worth mentioning in the same context. A full framework increases your development speed and is a dream when it comes to prototyping, but building a high performance web application is easier when you control each part. More on that in a later post, lets dive into how I found an ideal way to work with React JS.

Update: I updated the boilerplate with better testing and better handling of dependencies. Wrote about it here, [React JS and a browserify workflow, PART 2](#).

The problem

I really do not appreciate the simplicity of examples shown on probably all library/framework sites. Nobody builds an application in the global scope, nobody builds an application in a single file and everybody wants to make their code production ready. The **React JS** site is no different and it took me quite some time creating a good workflow. This is what I needed:

- Write JSX (The React JS javascript format) and transform it to regular javascript on the fly
- Write my code in a module pattern
- Use the React JS chrome dev tool
- Bundle my javascript and use source maps for debugging
- Run other tasks on file change, like CSS concat etc.
- It has to be extremely fast

This post will go through each step I did to find a solution. If you just want the solution, [jump ahead](#).

First try

So my first challenge was to solve the JSX transformation on the fly. Looking at the [React JS guide](#) I quickly figured out that the only thing I needed was to include an extra script tag in my HTML. It solves my problem, but it is not what I am looking for. The extra script tag is not something you want to put in production, so I would need to pre-transform my files containing JSX.

Second try

As the **React JS guide** states, you can install the **react-tools** globally and use a command line tool to watch files and convert them. It solves my problem, but it transforms each individual file and puts it in a different folder. This will create a messy project structure. I needed to change my strategy! I decided to look at some module tools instead, they would have to support the transformation of JSX anyways.

Third try

Having quite a bit of experience with **requirejs** I thought that would be a good bet, but it became more complex than I initially thought. A great job has been done on this plugin: [jsx-requirejs-plugin](#), but you get extra dependencies like the "text" plugin for requirejs and you have to use a modified version of the "JSXTransformer", that does not feel good. There was also an issue with bundling the whole project on each file change, it was too slow.

Fourth try

Personally I have never tried **browserify** before, but I was running out of options. It turns out that browserify is quite awesome, it makes it possible to write node syntax in your javascript files. The result of this is that you get modules and the possibility to re-use your JSX files in Node. That is a very good thing because **React JS** allows for server side compiling of HTML and send the string out to the client for further handling by the client side version of the library.

What is also good about **browserify** is that you have a plugin called **watchify** that will cache your project and watch it for changes, only doing the necessary re-bundling on updates. It is blazingly fast! Since I am used to **Grunt** I tried using the **grunt-browserify** and **grunt-watchify** tasks. This worked quite well, though I could not get the sourcemapping to work. But an even bigger issue was that I did not only need to watch the javascript files for changes, I also needed to concatenate my css files.

After some research I found a **grunt-concurrent** task that would allow me to run two parallel watch tasks. This worked, but it was slow. And I still could not get my sourcemapping to work.

Fifth try

So the way I used **Grunt** was too slow, what about **Gulp**? Now **Gulp** is a build tool like **Grunt**, only it streams the process so that you can manipulate the build process in memory instead of creating temporary files that are picked up by the next step in the build process.

Searching the web for a solution for **Gulp** I hit [this](#) post on stackoverflow. It shows multiple solutions to the same problem and way down I found an example that also handled transforming JSX with **reactify**, but it had some steps I did not understand. Why all this require stuff? And why require specific react.js file? It feels a bit hacky. It also gave me an error when I ran it in the browser.

This led me to using **gulp-browserify** which worked beautifully. But what about the **watchify** part? Searching for **gulp-watchify** it states: "experimental". That does not feel right, so how could I get this stuff to work? Searching the web again I found the following statement: "gulp-browserify has been blacklisted". DOH!, what now?

Sixth try

Sometimes you just have to do it from scratch. Using what I had experienced so far I built my own build process using the **browserify**, **watchify** and **reactify** npm modules. I was glad to see that **Gulp** also handled watching both my CSS for one task and that **watchify** task in parallel without any issues.

The solution

So this is how my **gulpfile.js** file looks like:

```
var gulp = require('gulp');
var source = require('vinyl-source-stream'); // Used to stream bundle for further handling
var browserify = require('browserify');
var watchify = require('watchify');
var reactify = require('reactify');
var concat = require('gulp-concat');

gulp.task('browserify', function() {
  var bundler = browserify({
```

```
    entries: ['./app/main.js'], // Only need initial file, browserify finds the deps
    transform: [reactify], // We want to convert JSX to normal javascript
    debug: true, // Gives us sourcemapping
    cache: {}, packageCache: {}, fullPaths: true // Requirement of watchify
  });
  var watcher = watchify(bundler);

  return watcher
    .on('update', function () { // When any files update
      var updateStart = Date.now();
      console.log('Updating!');
      watcher.bundle() // Create new bundle that uses the cache for high performance
        .pipe(source('main.js'))
      // This is where you add uglifying etc.
        .pipe(gulp.dest('./build/'));
      console.log('Updated!', (Date.now() - updateStart) + 'ms');
    })
    .bundle() // Create the initial bundle when starting the task
    .pipe(source('main.js'))
    .pipe(gulp.dest('./build/'));
  });

  // I added this so that you see how to run two watch tasks
  gulp.task('css', function () {
    gulp.watch('styles/**/*.css', function () {
      return gulp.src('styles/**/*.css')
        .pipe(concat('main.css'))
        .pipe(gulp.dest('build/'));
    });
  });
});
```

```
// Just running the two tasks  
gulp.task('default', ['browserify', 'css']);
```

So to summarize:

- Write JSX and transform it to regular javascript on the fly (**SOLVED**)
- Write my code in a module pattern (**SOLVED**)
- Use the React JS chrome dev tool
- Bundle my javascript and use source maps for debugging (**SOLVED**)
- Run other tasks on file change, like CSS concat etc. (**SOLVED**)
- It has to be extremely fast (**SOLVED**)

What about the REACT DEV-TOOLS?

The **React DEV-TOOLS** needs an instance of the React JS lib on the global scope to trigger. To fix this you simply have to add it to your **main.js** file.

```
/** @jsx React.DOM */  
  
var React = require('react');  
// Here we put our React instance to the global scope. Make sure you do not put it  
// into production and make sure that you close and open your console if the  
// DEV-TOOLS does not display  
window.React = React;  
  
var App = require('./App.jsx');
```

```
React.renderComponent(<App/>, document.body);
```

So that was my journey. I truly hope that the people at Facebook will put an example of this on the React JS site. React JS is truly awesome and its very sad if people leave it out because it is such a pain setting up a good workflow. Until then feel free to use an application boilerplate I have set up which also handles testing: [react-app-boilerplate](#).

Thanks for listening to my story and have fun with [React JS](#)!

[Tweet](#)

41

**29 Comments****christianalfoni****1 Login** ▾**Recommend** 4**Share****Sort by Best** ▾

Join the discussion...

**dianale** • 4 days ago

Thank you so much for starting this workflow! Trying to get ReactJS started in my IDE was an absolute nightmare, and this definitely simplified things. With the environment set up, I am now trying to figure out how to create a tree menu with React (<http://mandarinconlabarba.gith...> But since the creator's set up contains .jsx files, I'm a bit confused as to how to get the UI to look right. I'm completely new to web development, so any help would be appreciated. Is it okay to just copy over the files in the react-tree-menu's example module over to the "app" module in your boilerplate but with .js files?

^ | ▾ • Reply • Share >

**Christian Alfoni Jørgensen** Mod → dianale • 4 days ago

Lets talk on the email thread instad :-)

^ | ▾ • Reply • Share >

**gmilbank** • 14 days ago

Thanks, this helped me a lot.

^ | v • Reply • Share >

**chiedojohn** • 3 months ago

Thanks for putting this together! It's awesome! Any idea how I can modify this to also do babel es6 to es5 transforms?

^ | v • Reply • Share >

**chiedojohn** ➔ chiedojohn • 3 months ago

Actually, figured it out. using babelify.

At the top of the gulpfile:

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

On the line before ".bundle()"

```
.transform(babelify)
```

1 ^ | v • Reply • Share >

**Christian Alfoni Jørgensen** Mod ➔ chiedojohn • 2 months ago

Great! :-) Glad you figured it out!

^ | v • Reply • Share >

**Ram** • 3 months ago

Thanks for this write-up. I was looking for how to speed up by gulp + browserify + react builds and this solution looks to be the way.

^ | v • Reply • Share >

**Christian Alfoni Jørgensen** Mod ➔ Ram • 3 months ago

Glad it was helpful and thanks for the feedback :-)

^ | v • Reply • Share >



Ram → Christian Alfoni Jørgensen • 3 months ago

I refactored your solution slightly -> <https://gist.github.com/hidden...>

This also safely catches Browserify parsing errors without breaking the watchify loop.

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → Ram • 2 months ago

Ah, yes, I believe the boilerplate is updated with catching bundle errors, though not the article. But yes, important part of it indeed!

^ | v • Reply • Share >



Alias • 4 months ago

I'm currently having an issue where compile time keeps increasing on each save :(

<http://stackoverflow.com/quest...>

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → Alias • 3 months ago

I saw you figured it out, great!

^ | v • Reply • Share >



airandfingers • 5 months ago

Thanks so much! I followed the "Fast browserify builds with watchify" gulp recipe, and I couldn't figure out why my sourcemaps weren't working. Getting rid of gulp-sourcemaps and adding debug: true to my browserify was what I needed. Thanks again!

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → airandfingers • 3 months ago

Sweet, glad it helped! :-)

^ | v • Reply • Share >



kevbook • 6 months ago



Question, so i went to the github repo, ran the build - main.js is 240kb, i feel this is way too big considering it was a hello world app, and reactjs itself is only 30kb. We need a way that you can do all the above but not have such a big file.

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → kevbook • 6 months ago

Hi @kevbook,

Hm, React JS is 130kb minified. So some overhead to that should be our target. I agree it is quite large now. What I have forgotten to add is that you should run the DEPLOY in production environment. Then we are down to 200kb. That leaves us with 70kb overhead. Removing react-addons does not affect that build at all, so that is not where the overhead is coming from. But removing the fullpaths at the initial app bundle options when deploying, and we are down to 136 kb :-)

I will update the repo now with the new information. Thanks for bringing it to our attention!

^ | v • Reply • Share >



kevbook → Christian Alfoni Jørgensen • 6 months ago

Thanks. Would love to see how thats done.

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → kevbook • 6 months ago

Ah, sorry, if you go to the repo: [https://github.com/christianalfoni....](https://github.com/christianalfoni/reactjs-workflow) You will see the complete gulp file and guide for using it :-)

But please let me know if you have any more question!

^ | v • Reply • Share >



Alexander Baron • 7 months ago

I had gone down a similar path and built something very similar to this. What really helped me from your example is the console.logs. Extra points for code comments and great readability!

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → Alexander Baron • 3 months ago

Thanks for the feedback Alexander, much appreciated and very glad it helped you out! :-)

^ | v • Reply • Share >



Yudong Li • 7 months ago

@**Christian Alfoni Jørgensen** thanks for your awesome post, which is really helpful. The only thing I found is the links seem are all broken (quoted as part of your blog github url)...

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → Yudong Li • 7 months ago

Hi @**Yudong Li**!

Thanks so much for your feedback. Yup, the links was broken, but are fixed now. Was using some quotes there, removed them now :-)

Thanks again!

^ | v • Reply • Share >



frostymarvelous • 8 months ago

It's taken me two days, and I still can't get started cos the work flow is not defined anywhere! And everyone assumes you're using node in production so... Smfh.

Tomorrow, I'm gonna look into your solution. It's should work for me and allow me to finally start building awesome stuff. 🤖

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → frostymarvelous • 8 months ago

Hi @**frostymarvelous**! Yeah, I spent a bit of time figuring out this workflow, there is not a lot of help out there yet. It really is too bad there is no default workflow released with React JS. I got an approved pull request for the React JS site, pointing to the wiki where there are a few boilerplates and other tools to get going. The pull request is not live yet, but I hope it will be soon so people do not have to go on this wild "workflow-hunt" :-)

This is the link to the wiki: <https://github.com/facebook/re...>

1 ^ | v • Reply • Share >



Rob van der Linde • 8 months ago

Thanks for this guide, it's been very helpful. I also ran the final thing through gulp-uglify to compress the bundle.js file it generates (which in turn required vinyl-buffer to work). One of the things I noticed when using fullPaths = true as an argument to browserify, was the compressed bundle.js file that uglify generates is a bit bigger (238kb vs 167kb without fullPaths). A bit of a shame as I do want the smaller file size more, so I can no longer use watchify I guess?

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → Rob van der Linde • 8 months ago

Hi @Rob van der Linde, glad this was of help to you! I forgot to update that in the boilerplate. If you check now it uses the "options.development" flag to set fullpaths. So in normal workflow, using watchify, it has fullpaths. But when you deploy your code it will not use that and your file will be smaller :-)

^ | v • Reply • Share >



Noah Grant • 9 months ago

Hey Christian, thanks so much for this. Can you tell me what versions of these plugins you're using? I'm on browserify 6.2.0, watchify 2.1.1, reactify 0.15.2, and gulp 3.8.9. I don't think I'm getting an 'update' event triggered, because it doesn't respond to subsequent saves, and I think it might be due to a difference in library versions. Thanks!!

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → Noah Grant • 9 months ago

Hi Noah! Very glad it helped you out! :-) You know what, I was building a workflow with Angular JS and noticed that I had misunderstood the gulp watch API. It needs to notify when it is done doing "its thing" before it lets the watch callback run again. This is now fixed in the boilerplate, v.2.1.0. Please let me know if you still have issues!

^ | v • Reply • Share >



Christian Alfoni Jørgensen Mod → Christian Alfoni Jørgensen • 9 months ago

You know what... that wasn't it :p I updated the dependency version in package.json now and tested

them and it works. These are the version numbers:

```
"browserify": "^6.2.0"
"watchify": "^2.1.1",
"gulp": "^3.8.9",
"reactify": "^0.15.2",
```


I can not see why it would break. Are you running on MAC-OS? Is it the files in your app/ folder that does not trigger a rebundle?

^ | v • Reply • Share >

ALSO ON CHRISTIANALFONI

Handling complex state with Baobab


38 comments • 4 months ago

 **oskbor** — great article. Looking forward to a react-boabab-hot-loading-es6-es7-boilerplate repo ;)

WHAT'S THIS?

The great Angular component experiment

1 comment • 4 months ago

 **Vittorio C** — I've been using Angular 1.x for a couple of years now and I'm seriously hitting it's limits: - no easy and consistent way to lazy load a module - dirty checking

Christian Alfoni

Living in
Norway

Follow @christianalfoni