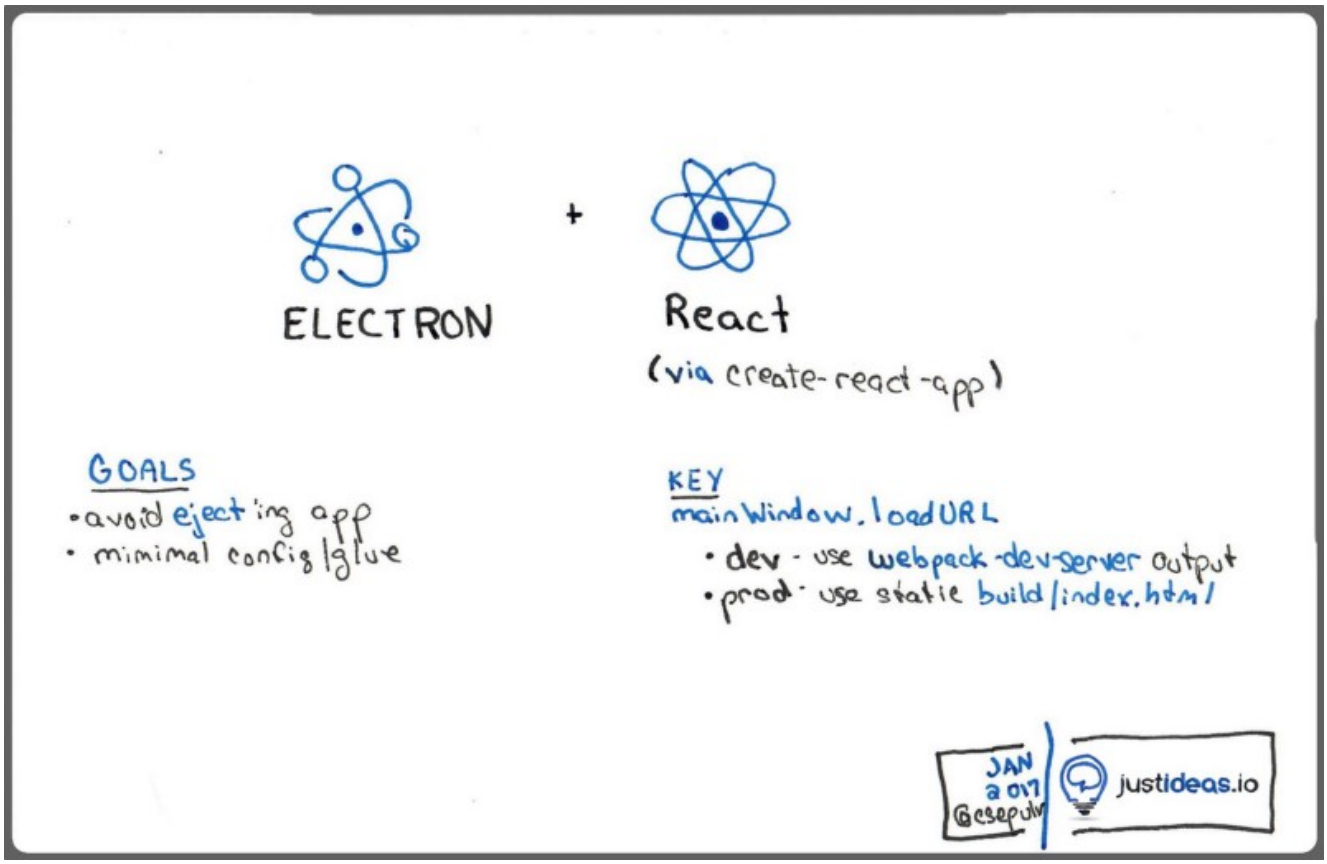
11 JANUARY 2017 / #JAVASCRIPT

Building an Electron application with create-react-app

Christian Sepulveda
VP Engineering Bastille Networks, coder (at times), espresso fanatic (that one should perhaps come first...)



No webpack configuration or “ejecting” necessary.

I recently built an Electron app using create-react-app. I didn’t need to muck about with Webpack, or “eject” my app, either. I’ll walk you through how I accomplished this.

the webpack configuration details. But my search for existing guides for using Electron and create-react-app together didn't bear any fruit, so I just dove in and figured it out myself.

If you're feeling impatient, you can dive right in and look at my code. Here's the GitHub repo for my app.

Before we get started, let me tell you about Electron and React, and why create-react-app is such a great tool.

Electron and React

React is Facebook's JavaScript view framework.

***A JavaScript library for building user interfaces* - React**
A JavaScript library for building user interfacesfacebook.github.io

And Electron is GitHub's framework for building cross-platform desktop apps in JavaScript.

Electron
Build cross platform desktop apps with JavaScript, HTML, and CSS.electron.atom.io

Most use webpack for the configuration necessary for React development. webpack is a configuration and build tool that most of the React community has adopted over alternatives like Gulp and Grunt.

The configuration overhead varies (more on this later), and there are many boilerplate and application generators available, but in July 2016 Facebook Incubator released a tool, create-react-app. It hides most of the configuration and lets the developer use simple

their apps.

What is ejecting, and why do you want to avoid it?

create-react-app makes certain assumptions about a typical React setup. If these assumptions aren't for you, there is an option to **eject** an application (`npm run eject`). Ejecting an application copies all the encapsulated configuration of create-react-app to the your project, providing a boilerplate configuration that you can change as you wish.

But this is a *one way* trip. You can't undo ejecting and go back. There have been 49 releases (as of this post) of create-react-app, each making improvements. But for an ejected application, you would have to either forgo these improvements or figure out how to apply them.

An ejected configuration is over 550 lines spanning 7 files (as of this post). I don't understand it all (well, most of it, actually) and I don't want to.

Goals

My goals are simple:

- avoid ejecting the React app
- minimize glue to get React and Electron working together
- preserve the defaults, assumptions and conventions made by Electron and create-react-app/React. (This can make it easier to use other tools that assume/require such conventions.)

1. run `create-react-app` to generate a basic React application
2. run `npm install --save-dev electron`
3. add `main.js` from `electron-quick-start` (we'll rename it to `electron-starter.js`, for clarity)
4. modify call to `mainWindow.loadURL` (in `electron-starter.js`) to use `localhost:3000` (webpack-dev-server)
5. add a main entry to `package.json` for `electron-starter.js`
6. add a run target to start Electron to `package.json`
7. `npm start` followed by `npm run electron`

Steps 1 and 2 are pretty straightforward. Here's the code for steps 3 and 4:

```
const electron = require('electron');
// Module to control application life.
const app = electron.app;
// Module to create native browser window.
const BrowserWindow = electron.BrowserWindow;

const path = require('path');
const url = require('url');

// Keep a global reference of the window object, if you don't, the w
// be closed automatically when the JavaScript object is garbage col
let mainWindow;

function createWindow() {
    // Create the browser window.
    mainWindow = new BrowserWindow({width: 800, height: 600});

    // and load the index.html of the app.
    mainWindow.loadURL('http://localhost:3000');

    // Open the DevTools.
    mainWindow.webContents.openDevTools();

    // Emitted when the window is closed.
```

```
        // in an array if your app supports multi windows, this is t
        // when you should delete the corresponding element.
        mainWindow = null
    })
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.on('ready', createWindow);

// Quit when all windows are closed.
app.on('window-all-closed', function () {
    // On OS X it is common for applications and their menu bar
    // to stay active until the user quits explicitly with Cmd + Q
    if (process.platform !== 'darwin') {
        app.quit()
    }
});

app.on('activate', function () {
    // On OS X it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (mainWindow === null) {
        createWindow()
    }
});

// In this file you can include the rest of your app's specific main
// code. You can also put them in separate files and require them he
```

(Gist)

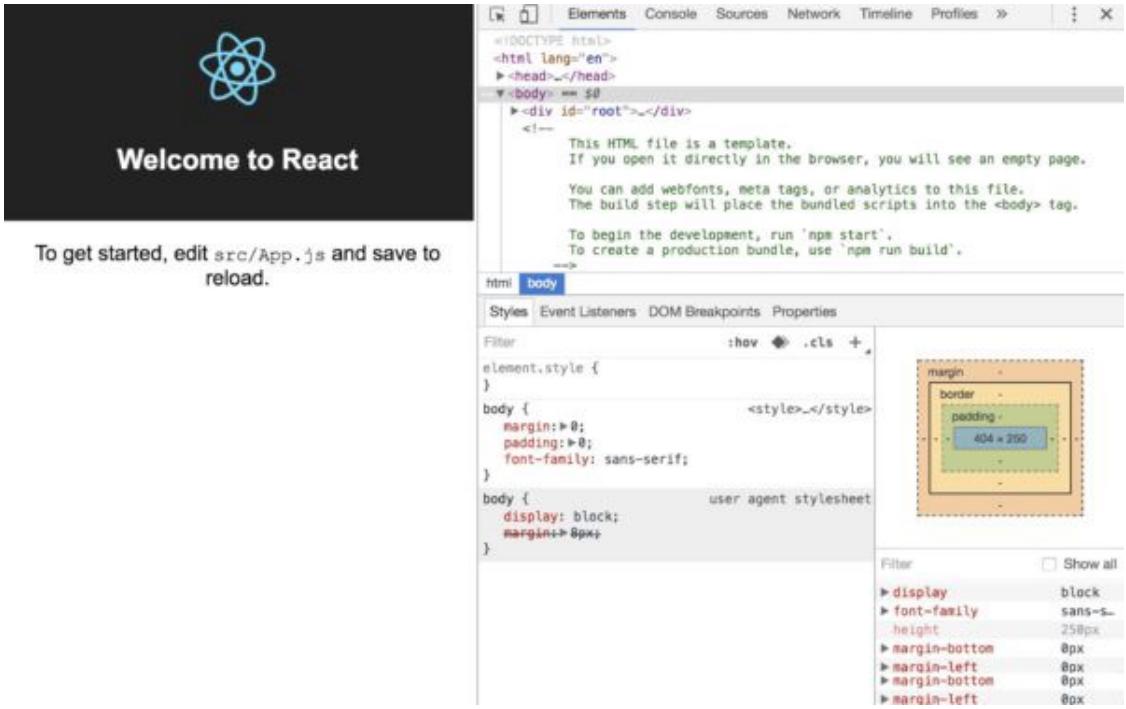
And for steps 5 and 6:

```
{
  "name": "electron-with-create-react-app",
  "version": "0.1.0",
  "private": true,
  "devDependencies": {
    "electron": "^1.4.14",
```

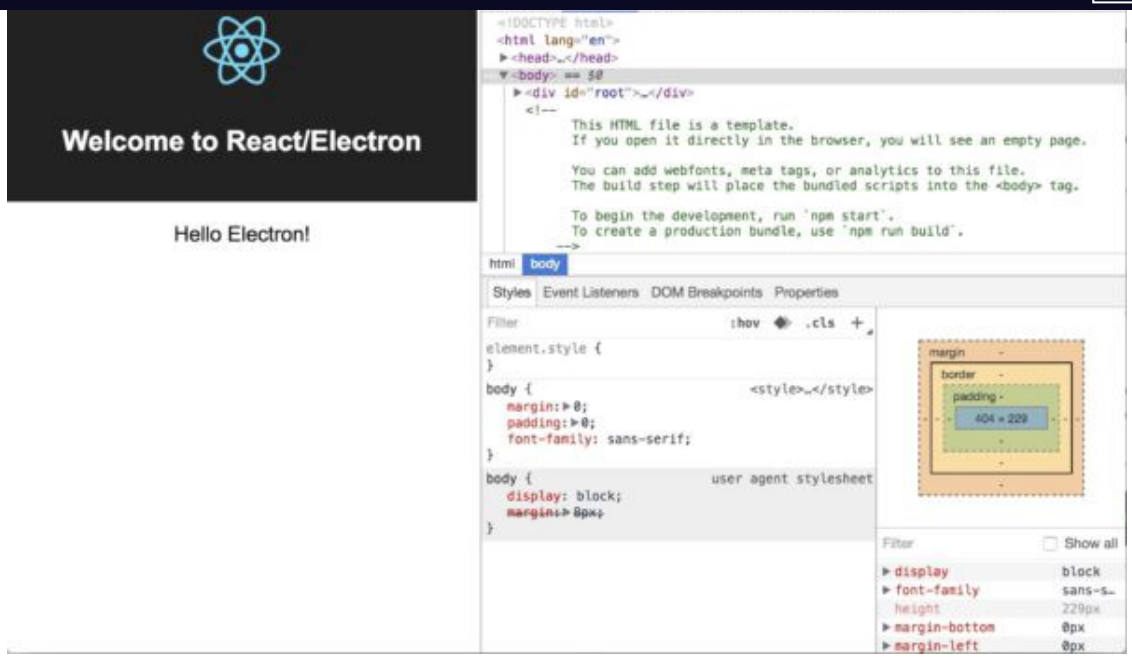
```
  "dependencies": {
    "react": "^15.4.2",
    "react-dom": "^15.4.2"
  },
  "main": "src/electron-starter.js",
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject",
    "electron": "electron ."
  }
}
```

(Gist)

When you run the npm commands in step 7, you should see this:



You can make live changes to the React code and you should see them reflected in the running Electron app.



This works okay for development, but has two shortcomings:

- production won't use `webpack-dev-server`. It needs to use the static file from building the React project
- (small) nuisance to run both npm commands

Specifying the loadURL in Production and Dev

In development, an environment variable can specify the url for `mainWindow.loadURL` (in `electron-starter.js`). If the env var exists, we'll use it; else we'll use the production static HTML file.

We'll add a npm run target (to `package.json`) as follows:

```
"electron-dev": "ELECTRON_START_URL=http://localhost:3000 electr
```

@bfarmilo)

```
"electron-dev": "set ELECTRON_START_URL=http://localhost:3000 &&
```

In `electron-starter.js`, we'll modify the `mainWindow.loadURL` call as follows:

```
const startUrl = process.env.ELECTRON_START_URL || url.format({
            pathname: path.join(__dirname, '/../build/index.html'),
            protocol: 'file:',
            slashes: true
        });
    mainWindow.loadURL(startUrl);
```

(Gist)

There is a problem with this: `create-react-app` (by default) builds an `index.html` that uses absolute paths. This will fail when loading it in Electron. Thankfully, there is a config option to change this: set a `homepage` property in `package.json`. (Facebook documentation on the property is here.)

So we can set this property to the current directory and `npm run build` will use it as a relative path.

```
"homepage": "./",
```

Using Foreman to Manage React and

1. launch/manage both React dev server and Electron processes (I'd rather deal with one)
2. wait for the React dev server to start and then launch Electron

Foremen is a good process management tool. We can add it,

```
npm install --save-dev foreman
```

and add the following `Procfile`

```
react: npm startelectron: npm run electron
```

(Gist)

That deals with (1). For (2), we can add a simple node script (`electron-wait-react.js`) that waits for the React dev server to start, then starts Electron.

```
const net = require('net');
const port = process.env.PORT ? (process.env.PORT - 100) : 3000;

process.env.ELECTRON_START_URL = `http://localhost:${port}`;

const client = new net.Socket();

let startedElectron = false;
const tryConnection = () => client.connect({port: port}, () => {
        client.end();
```

```
            startedElectron = true;
            const exec = require('child_process').exec;
            exec('npm run electron');
        }
    }
);

tryConnection();

client.on('error', (error) => {
    setTimeout(tryConnection, 1000);
});
```

(Gist)

> NOTE: Foreman will offset the port number by 100 for processes of different types. (See here.) So, `electron-wait-react.js` subtracts 100 to set the port number of the React dev server correctly.

Now modify the `Procfile`

```
react: npm startelectron: node src/electron-wait-react
```

(Gist)

Finally, we'll change the run targets in `package.json` to replace `electron-dev` with:

```
"dev" : "nf start"
```

```
npm run dev
```

> UPDATE (1/25/17) : I've added the following section in response to some user comments (here and here). They need access to Electron from within the react app and a simple require or import throws an error. I note one solution below.

Accessing Electron from the React App

An Electron app has two main processes: the Electron host/wrapper and your app. In some cases, you'd like access to Electron from within your application. For example, you might want to access the local file system or use Electron's `ipcRenderer`. But if you do the following, you'll get an error

```
const electron = require('electron')
//or
import electron from 'electron';
```

There is some discussion about this error in various GitHub and Stack Overflow issues, such as this one. Most solutions propose webpack config changes, but this would require ejecting the application.

However, there is a simple workaround/hack.

```
const electron = window.require('electron');
```

```
const electron = window.require('electron');
const fs = electron.remote.require('fs');
const ipcRenderer  = electron.ipcRenderer;
```

Wrapping Up

For convenience, here is a GitHub repo that has all the changes above, with tags for each step. But, there it isn't much work to bootstrap an Electron application that uses create-react-app. (This post is much longer than the code and changes you would need to integrate the two.)

And if you are using create-react-app, you might want to check out my post, Debugging tests in WebStorm and create-react-app.

Thanks for reading. You can check out more of my posts at justideas.io

> UPDATE (2/2/17). A reader, Carl Vitullo, suggested to use `npm start` instead of `npm run dev` and submitted a pull request with the changes, on GitHub. These tweaks are available in this branch.

If this article was helpful, tweet it or share it.

Donate $5 and buy the world 250 hours of learning.

Countinue reading about

Don't just lint your code - fix it with Prettier

3 easy ways to boost your web application performance

10 More Utility Functions Made with Reduce

See all 1508 posts →

#AWS

Amazon S3 — Cloud File Storage for Performance and Cost Savings

3 YEARS AGO

#WRITING

How to get published in the freeCodeCamp Medium publication

QUINCY LARSON 3 YEARS AGO

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

Our Nonprofit

About
Alumni Network
Open Source
Shop
Support
Sponsors
Academic Honesty

Trending Guides

2019 Web Developer Roadmap
Python Tutorial
CSS Flexbox Guide
JavaScript Tutorial
Python Example
HTML Tutorial
Linux Command Line Guide

React Tutorial
Java Tutorial
Linux Tutorial
CSS Tutorial
jQuery Example
SQL Tutorial
CSS Example
React Example
Angular Tutorial
Bootstrap Example
How to Set Up SSH Keys
WordPress Tutorial
PHP Example