



 Search

 Write

Sign
up

Sign
in



Build a React Embeddable widget



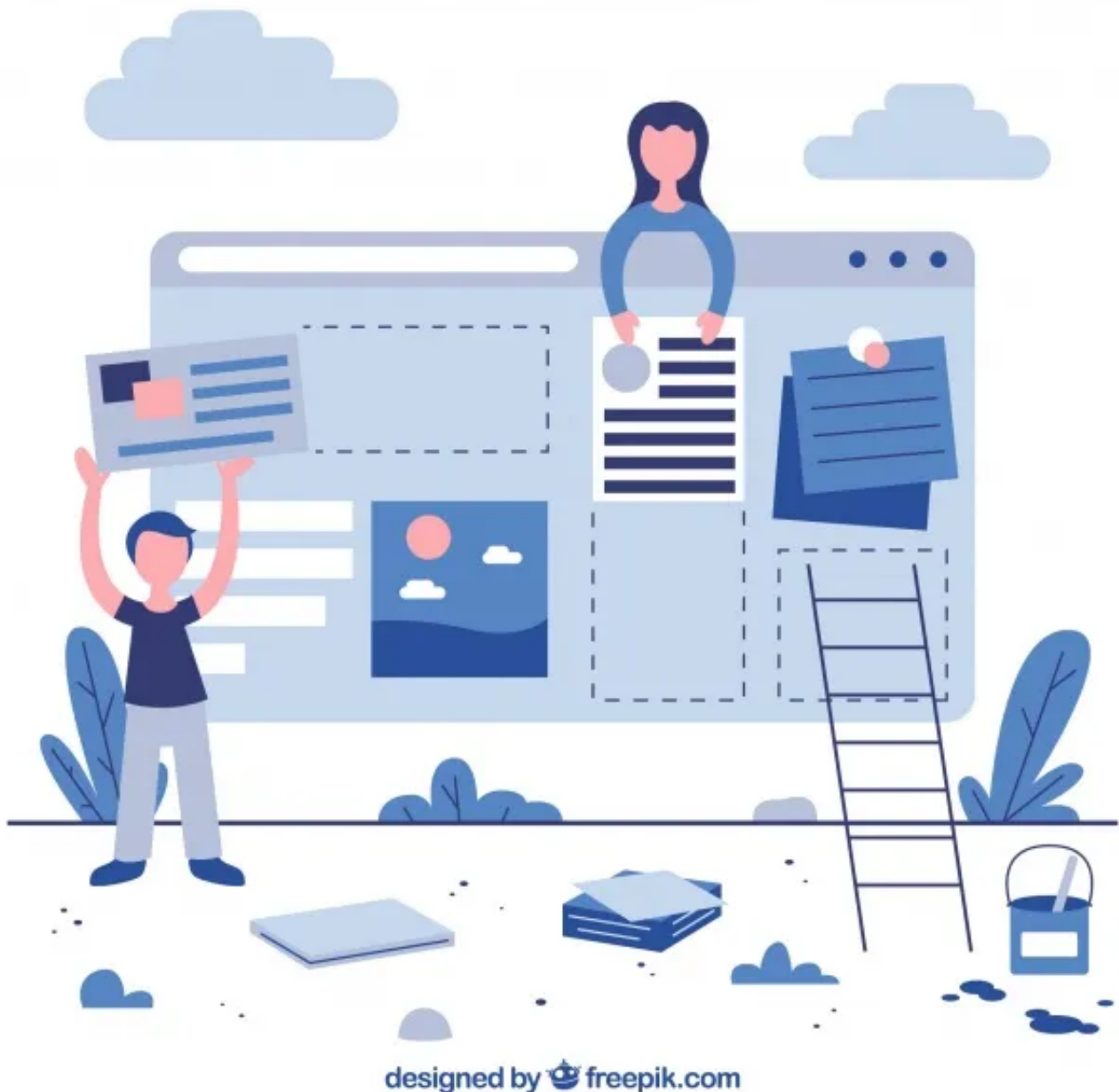
Nicolas Raynaud · [Follow](#)

6 min read · Feb 3, 2021

 476

 7





Chances are, if you're doing web applications, that you will at least once, come across a need where you want to embed some component in a website that you did not develop, or one that is in a different technology than your preferred stack. After all, web development is all about integration, and if you can't create a component with a single responsibility principle and inject it anywhere, well you should !

Like me, you like well built apps and websites, so the perspective of creating a pure javascript widget to inject html in the DOM by yourself is pure evil ! You can't sleep at night thinking that you'll have to use jQuery or reinvent the wheel by coding again some utils functions that npm packages have had in store for a decade...

Also, since you know **React** (or not, don't leave just for that), you know that there might be a way to build a proper mini app that can instantiate in any website because, after all, that's what a React app does. You just can't put your finger on the right setup to do so !

Well, you're in luck, there is a way ! Hopefully it will help you streamline your widget development, and hey, you might even enjoy doing it after that !

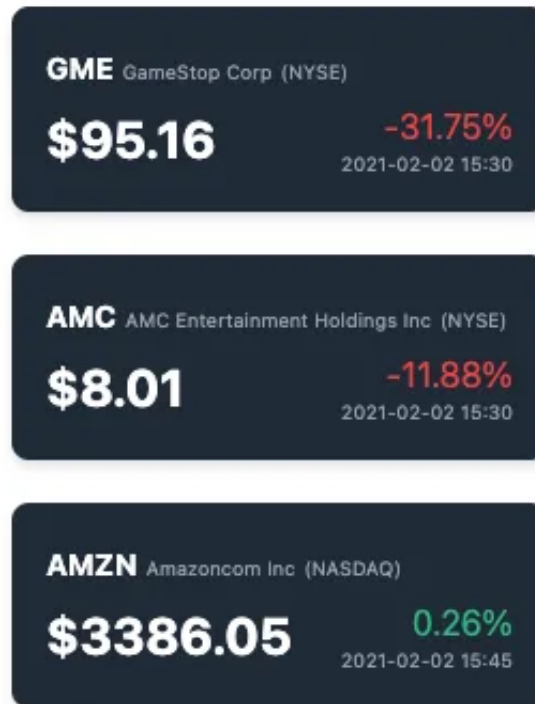
What we'll build

I have chosen to create a widget that needs to consume an API to display some information. Nothing extraordinary, but just enough so that you can start to feel the interest of using React rather than coding it using jQuery...

We'll create a widget displaying a stock price, because it seems like the trendy thing to do right now !

The widget will take the stock symbol as parameter and we need to be able to inject it several times in the same HTML page, either with the same, or other stock symbols.

This is what we're looking to create :



3 stock widgets in a page

Create the app widget

As usual, a React app development starts with a create-react-app, no surprise there :

```
npx create-react-app my-widget  
cd my-widget  
npm install axios
```

We'll need a couple libs, because there is no limit to what one can do with

a React app, and your widget could also be something bigger than just a quote !

At this point, you have a very working and very disturbingly empty and common React app.

Change its initialization and injection

Now, the fun begins. By default a React app instantiate itself automatically in a div named “root” as you can see in the default “index.js”

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5  ReactDOM.render(
6    <React.StrictMode>
7      <App />
8    </React.StrictMode>,
9    document.getElementById('root')
10 );
```

index.js hosted with ♥ by GitHub

[view raw](#)

But that does not help us much because :

- A **div id="root"** is too common, you don't know where your widget will be injected and, chances are, some other developer will have named something “root”.
- We don't have any control on input parameters or the fact that we want to render the app multiple times : here, it is impossible because

it only identifies one div, and inject the app in the first it finds.

So, we'll need to help our widget find its home by changing it a little bit :

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5
6  // Find all widget divs
7  const widgetDivs = document.querySelectorAll('.nicoraynaud-finance-widget');
8
9  // Inject our React App into each class
10 widgetDivs.forEach(div => {
11     ReactDOM.render(
12         <React.StrictMode>
13         <App symbol={div.dataset.symbol}/>
14         </React.StrictMode>,
15         div
16     );
17 });
```

index is hosted with a by GitHub

view raw

What we did here, is look for all divs identified by a class, and for each of them, inject a **new instance** of our widget. Also, look at **div.dataset.symbol**, this is an automatic shortcut to reference an element attribute identified by the prefix “data-”. This is what we'll use to specify our stock symbol parameter. You can learn more about it [here](#).

In dev, npm start will inject the React app in the **public/index.html**, so that is where we want to change things to see our app working again. We'll remove :

```
<div id="root"></div>
```

And replace it with :

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8" />
5      <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6      <meta name="viewport" content="width=device-width, initial-scale=1" />
7      <meta name="theme-color" content="#000000" />
8      <meta
9        name="description"
10       content="Web site created using create-react-app"
11     />
12     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13     <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
14     <title>React App</title>
15   </head>
16   <body>
17     <noscript>You need to enable JavaScript to run this app.</noscript>
18
19     <div id="gamestop" data-symbol="GME" class="nicoraynaud-finance-widget" style=
20     <div id="amc" data-symbol="AMC" class="nicoraynaud-finance-widget" style="wid
21     <div id="amzn" data-symbol="AMZN" class="nicoraynaud-finance-widget" style="w
22
23   </body>
24 </html>
```

Here, i have already created three widgets, each with a different parameter so that we'll have several stock quotes displaying.

Implement the feature

That is not really the point of this blog post but this is still fun !

I'll just need axios, tailwindcss and a REST API.

I chose MarketStack for the finance API : they offer a free plan, with CORS compatible setup (so I can query the prices directly from the browser, not fighting with cross origin issues).

*Disclaimer : This, though, requires me to create an account with them and use an access key for each HTTP request. Bear in mind that this is **not secure**, as your access key will be available to whoever looks at network traffic from your widget. I chose this API for the sake of simplicity and because Yahoo Finance API does not allow cross domain requests.*

Now let's create a **StockQuote** component that will fetch and display the necessary data :

```
1  import React, {useState, useEffect} from 'react';
2  import moment from 'moment';
3  import axiosInstance from '../index';
4
5  const MARKET_STACK_QUOTE_URL = `${process.env.REACT_APP_MARKETSTACK_BASE_URL}/int:
6  const MARKET_STACK_TICKER_URL = `${process.env.REACT_APP_MARKETSTACK_BASE_URL}/tic
7
8  function StockQuote(props) {
9      const [quote, setQuote] = useState({
10         price: '---',
11         var: '---',
12         time: '---'
13     });
14     const [stock, setStock] = useState({
15         stockExchange: 'N/A',
16         name: 'N/A',
```



```
17     });
18
19     useEffect(() => {
20       axiosInstance.get(MARKET_STACK_QUOTE_URL, {
21         params: {
22           access_key: process.env.REACT_APP_MARKETSTACK_ACCESS_KEY,
23           symbols: props.symbol,
24           interval: '15min',
25           date_from: moment().subtract(1, 'day').format('YYYY-MM-DD'),
26           date_to: moment().format('YYYY-MM-DD'),
27           limit: '1',
28         }
29       }).then((result) => {
30         if (!result.data.data || result.data.data.length <= 0) {
31           return;
32         }
33         const lastQuote = result.data.data[0];
34         setQuote({
35           price: lastQuote.last,
36           var: Math.trunc(-(1 - (lastQuote.last/lastQuote.open)) * 10000) /
37           time: moment(lastQuote.date).format('YYYY-MM-DD HH:mm'),
38         })
39       });
40     });
41
42     useEffect(() => {
43       axiosInstance.get(`${MARKET_STACK_TICKER_URL}/${props.symbol}`, {
44         params: {
45           access_key: process.env.REACT_APP_MARKETSTACK_ACCESS_KEY,
46         }
47       }).then((result) => {
48         if (!result.data) {
49           return;
50         }
51         setStock({
52           stockExchange: result.data.stock_exchange.acronym,
53           name: result.data.name,
54         })
55       });
56     });
57
58     const varColor = quote.var < 0 ? 'text-red-500' : 'text-green-500';
```

```

58     <div className={quoteSymbol} text-sm text-white font-bold>{props.symbol}</div>
59
60     return (
61       <div className={'quote rounded-lg shadow-md p-4 bg-gray-800'}>
62         <span className={'quoteSymbol text-sm text-white font-bold'}>{props.symbol}</span>
63         <span className={'quoteSymbol text-2xs text-gray-400 ml-1'}>{stock.name}</span>
64         <span className={'quoteSymbol text-2xs text-gray-400 ml-1'}>({stock.symbol}</span>
65         <div className={'quote flex flex-row justify-between mt-1'}>
66           <div className={'quotePrice self-center text-2xl font-bold items-center'}>
67             <div className={'flex flex-col text-right'}>
68               <div className={'quoteVar ' + varColor}>{quote.var}%</div>
69               <div className={'quoteTime text-right text-2xs text-gray-400'}>{quote.time}</div>
70             </div>
71           </div>
72         </div>
73       </div>
74     );
  }

```

Now we can put this widget in the App, passing it the symbol as a prop (remember, the symbol comes from the data attributes) :

```

1  import StockQuote from "../components/StockQuote";
2
3  function App(props) {
4
5    return (
6      <div>
7        <StockQuote symbol={props.symbol}/>
8      </div>
9    );
10  }
11
12  export default App;

```

App.js hosted with ♥ by GitHub

[view raw](#)

Build as a widget

Now, our widget works well, but it is still a React App, and if we build our app using the standard command, we'll just create a whole bunch of static files that do not make it an easily embeddable widget :

```
→ my-widget git:(master) ✗ npm run build

> my-widget@0.1.0 build
> craco build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

 65.04 KB  build/static/js/2.0476f53c.chunk.js
 1.58 KB   build/static/css/main.5592c7a8.chunk.css
 1.22 KB   build/static/js/main.7e14f79e.chunk.js
 776 B     build/static/js/runtime-main.73042487.js

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  yarn global add serve
  serve -s build

Find out more about deployment here:

https://cra.link/deployment
```

See all these static files ? That means, anyone wanting to embed the widget will need these 4 files. This is too much, I am sure we can do better.

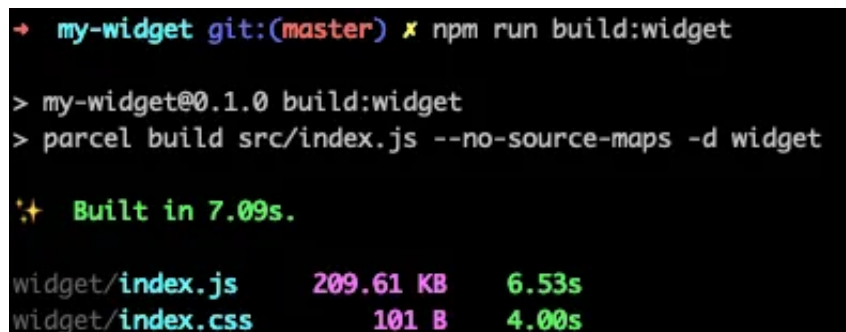
We'll need a bundler tool like (parcel) to help us squash all this. (Note that many other bundler exist, i'm only using parcel because it is a one liner solution in my case) :

```
npm install -D parcel-bundler
```

Then we'll create a new build script in package.json :

```
...  
"scripts": {  
  "eject": "react-scripts eject",  
  "start": "craco start",  
  "build": "craco build",  
  "test": "craco test",  
  "build:widget": "parcel build src/index.js --no-source-maps -d  
widget && cp build/static/css/*.css widget/index.css"  
},  
...
```

This new “**build:widget**” command will let parcel parse our app, and output it as a single *js* and a single *css* file in the “*widget*” directory. Here is the command being ran :



```
→ my-widget git:(master) ✕ npm run build:widget  
  
> my-widget@0.1.0 build:widget  
> parcel build src/index.js --no-source-maps -d widget  
  
✨ Built in 7.09s.  
  
widget/index.js    209.61 KB    6.53s  
widget/index.css   101 B       4.00s
```

Output of parcel script is now just 2 files

The only drawback from parcel, is that it does not automatically process my tailwind file. As a temporary fix, I rely on the processed css file from build/static/css and overwrite the one generated by parcel. Hence the

subsequent copy command.

And that's it ! You can now import these 2 files anywhere and the widget will get instantiated automatically !

Host it somewhere

Now that you have your binary, should you want to embed it somewhere, you'll need to host it so the html page in which you'd like to display it can fetch the *js* and *css* files easily.

Embed your widget

Now comes the rewarding moment, injecting your widget in your html !! Here is a sample code that will just do that :

```
1  ...
2  <div id="something-in-your-website">
3    <div class="nicoraynaud-finance-widget"
4      data-symbol="GME">
5    </div>
6  </div>
7  ...
8  ...
9  <div id="something-else-in-your-website">
10   <div class="nicoraynaud-finance-widget"
11     data-symbol="AMZN">
12   </div>
13 </div>
14 <!-- /\ Only add these two tags once per page -->
15 <link href="https://tekinico.com/demo/react-widget/index.css" rel="stylesheet"/>
16 <script src="https://tekinico.com/demo/react-widget/index.js"></script>
17 ...
```

embedded.html hosted with a by GitHub

view raw

Notice how we only need to import the js and css resources **once**. This is because when the script is loaded, it will automatically loop through all html elements with class “nicoraynaud-finance-widget” and inject an instance of the widget inside. So, be careful not to import it several times.

I also recommend importing these scripts at the end of the page, to avoid blocking your content with these resources. Once they are downloaded and the js script ran, widget will automatically appear.

Conclusion

So, I know, this wasn't a big deal, but it took me a couple hours to set this up together and streamline a react app into a widget that easily, so I figured : why not sharing it with you !

The whole project is available to you on my github @ <https://github.com/nicoraynaud/react-widget>

Enjoy :)

[React](#)[Widget](#)[Development](#)[Tailwind Css](#)[Parcel](#)



Written by Nicolas Raynaud

18 Followers

Follow

I am a curious developer working hard to follow Software Craftsmanship principles.

More from Nicolas Raynaud



Nicolas Raynaud

Inject your SpringBoot app data in Google BigQuery

We all know data is king ! But to be analyzed, data needs to be injected and centralized in data warehouses first. This tutorial covers a...

7 min read · Oct 15, 2020

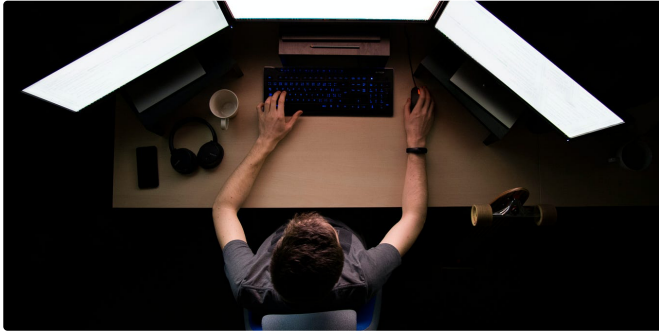


8



See all from Nicolas Raynaud

Recommended from Medium



Nicolas Zamarreno in sleepers

Create a Web Component from a React Component

The wonderful world that Web Component encapsulation offers you!

8 min read · Oct 6, 2023



80



2



M Mahdi Ramadhan, M. Si

Best Practices for Storing Data in Web Applications with ReactJS

Introduction

6 min read · Nov 6, 2023



4



Lists



Growth Marketing

11 stories · 91 saves



Stories to Help You Grow as a Software Developer

19 stories · 948 saves



General Coding Knowledge

20 stories · 1075 saves



Modern Marketing

103 stories · 531 saves



 George Leshchyshyn


Building a Chrome Extension with React!

In today's digital age, browser extensions have become essential tools for enhancin...

7 min read · Oct 3, 2023

 64  3



 sharath.v

How to install Webpack and configure in React JS?

Webpack is a popular JavaScript module bundler that is commonly used in React...

3 min read · Oct 5, 2023

 65  1



 Bhairab Patra

.env file in React js

In a React.js project, the .env file is typically used to store environment variables....

2 min read · Oct 28, 2023

 117  2



 Rish... in JavaScript Journal: Unlocking Project...

Building a TypeScript-React Project from Scratch with...

Webpack is the go-to bundler for many JavaScript projects, offering robust and...

3 min read · Oct 13, 2023

 25  3



See more recommendations

[Help](#) [Status](#) [About](#) [Careers](#) [Blog](#) [Privacy](#) [Terms](#) [Text to speech](#) [Teams](#)