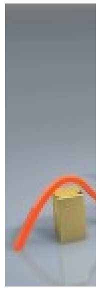
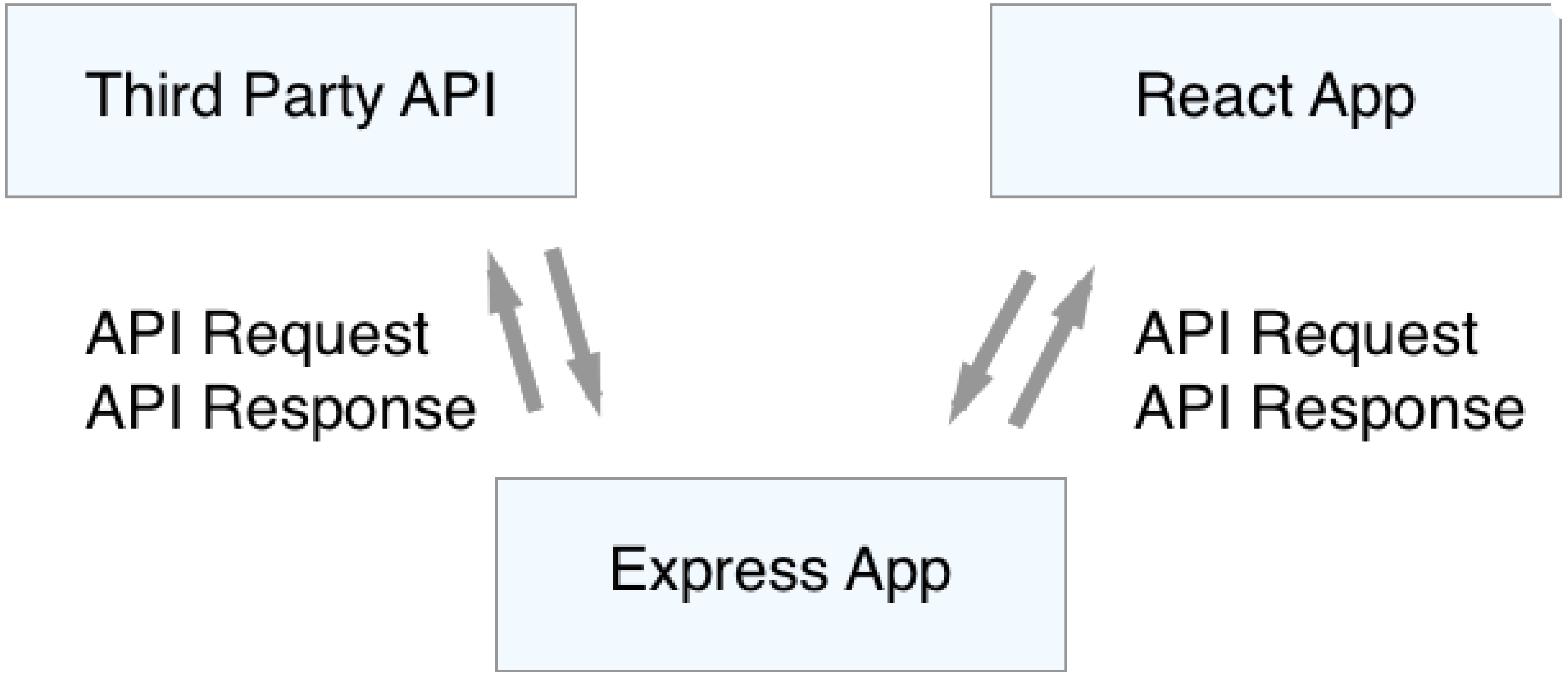
APIs mentione



Browse N



How to create a React app with Express

APIs mentione

By Steven // April 23, 2021

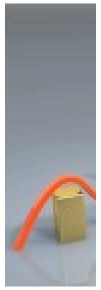


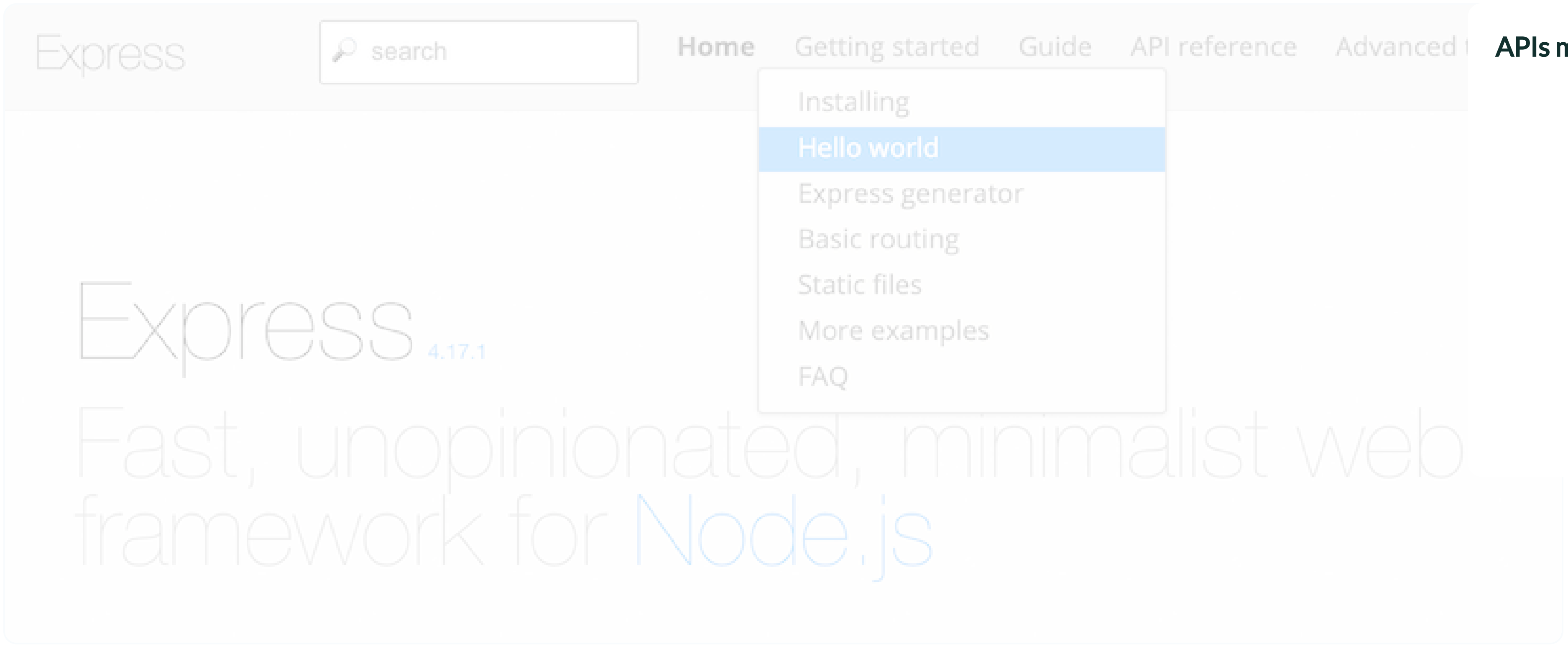
Table of Contents [show]

Creating web apps gives developers the power to reach a wide audience, for sharing content, selling produc and opening communication. In the past few years, the combination of **Express.js** and **React.js** has proven tc powerful tool in the software developer's tool belt. With these two frameworks, front-end engineers can qu..., create React apps on the front-end, quick communicate with a back-end through their own API.

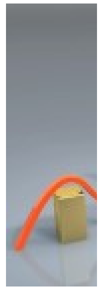
Browse N

View the Best Free APIs List

What is Express?



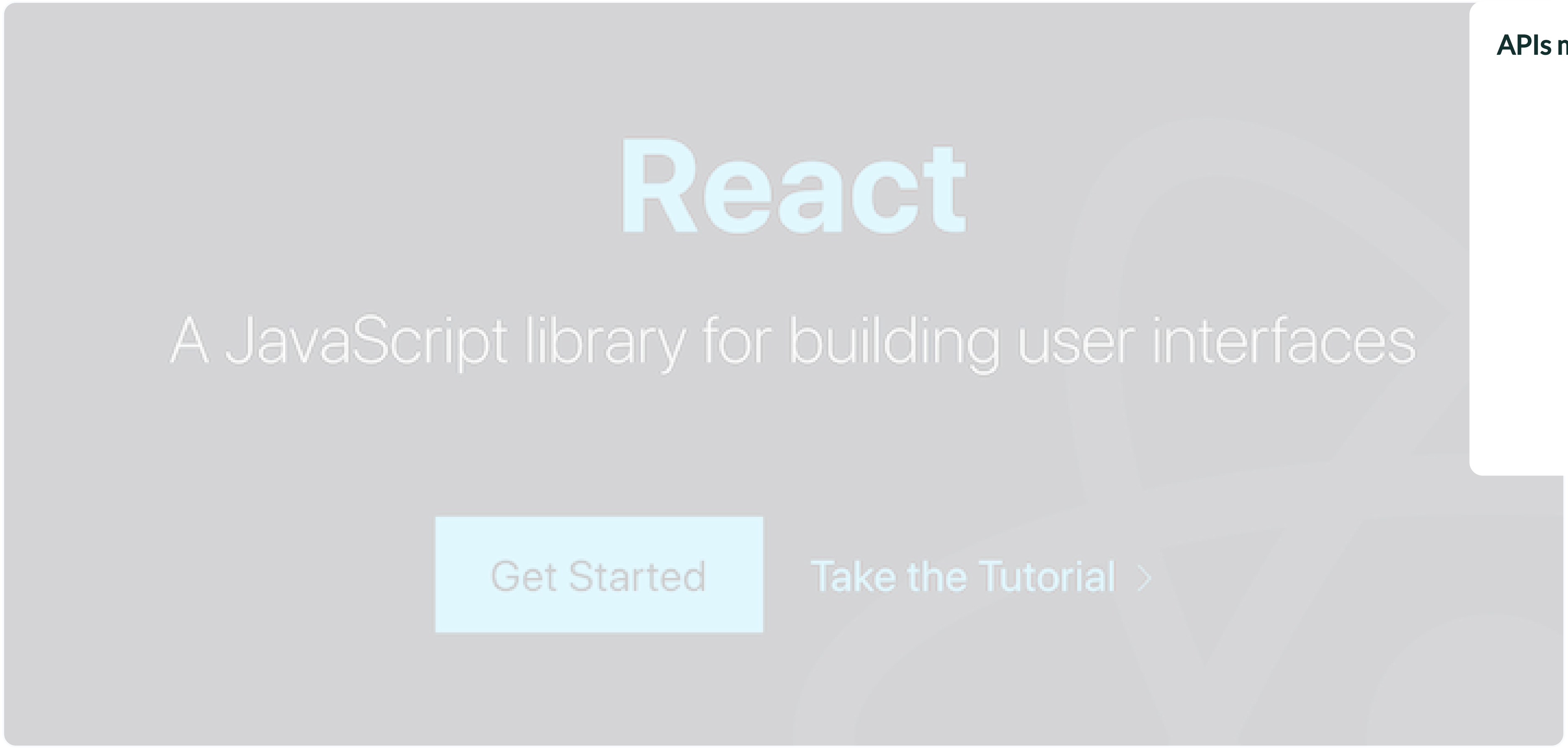
APIs mentione



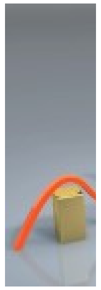
Browse N

Express is a minimalist, lightweight framework for making web apps in Node.js. Professional software developers have been using Express.js for a decade to rapidly create entire websites, back-ends, REST APIs, and more. One of the major benefits of Express is that developers can use their prior knowledge of JavaScript, without needing to learn a new language. This makes it even quicker to get started making new apps.

What is React?



APIs mentione



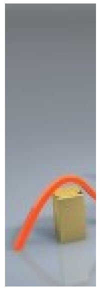
Browse N

Since it was introduced half a decade ago, React has been making waves in the web development world and beyond. And for good reason: React is incredibly powerful for allowing developers to express complex application logic with very little code. Because of this, React apps are usually very quick to write, very easy to add features to, and very small in their code size, which makes them easy to manage.

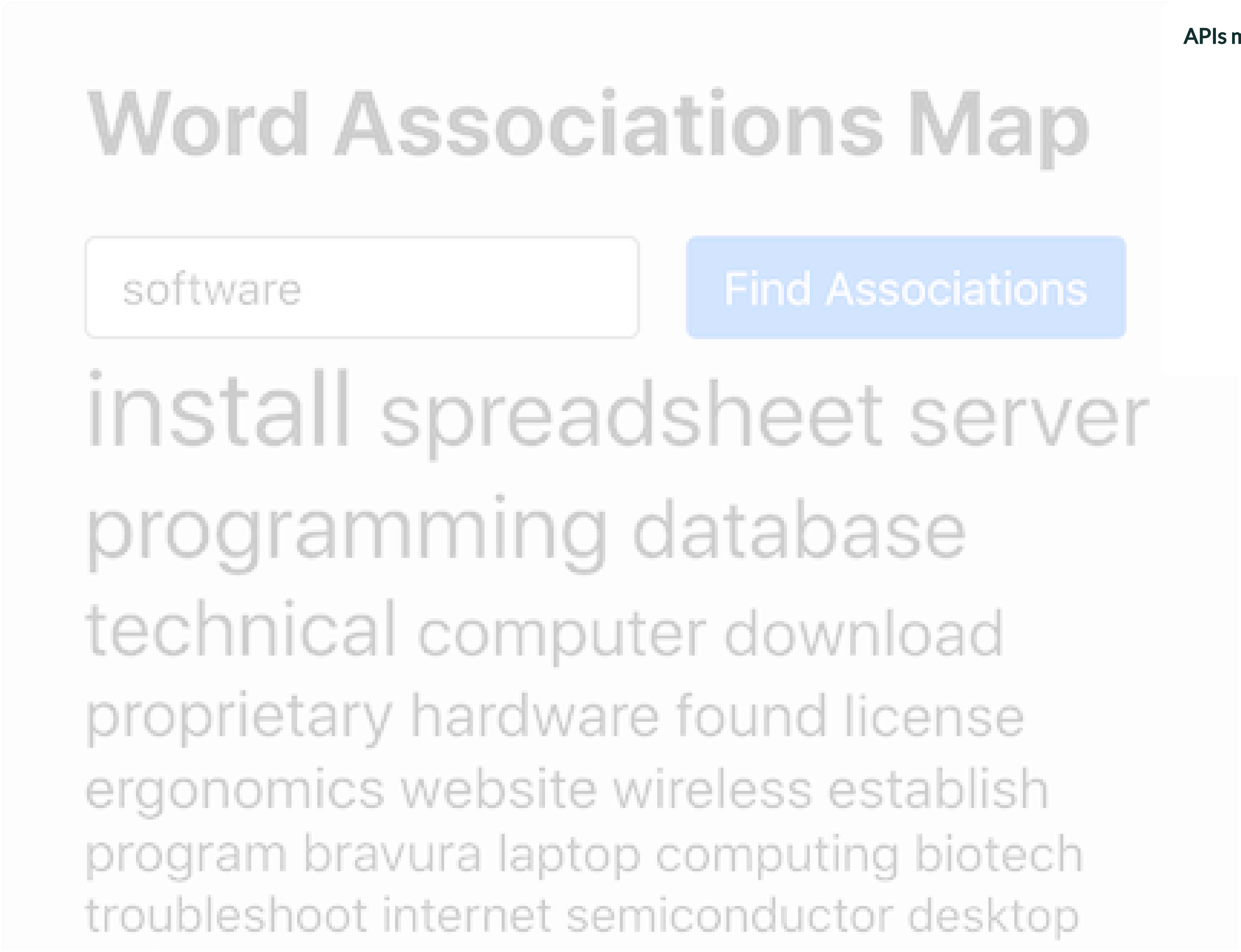
Example: Word Associations app in React and Express

APIs mentione

To show you how to use Express, React, and hook them up with each other, we're going to make a simple bu informative **Word Associations** app. The app will let you type a word, and show you a visual map of many associated words. The bigger the word is, the more associated with your word it will be, which will let us see relationships at a glance.



Browse N



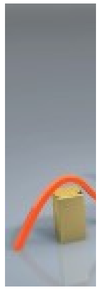
Word Associations Map

software

Find Associations

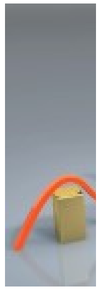
install spreadsheet server programming database technical computer download proprietary hardware found license ergonomics website wireless establish program bravura laptop computing biotech troubleshoot internet semiconductor desktop

APIs mentione



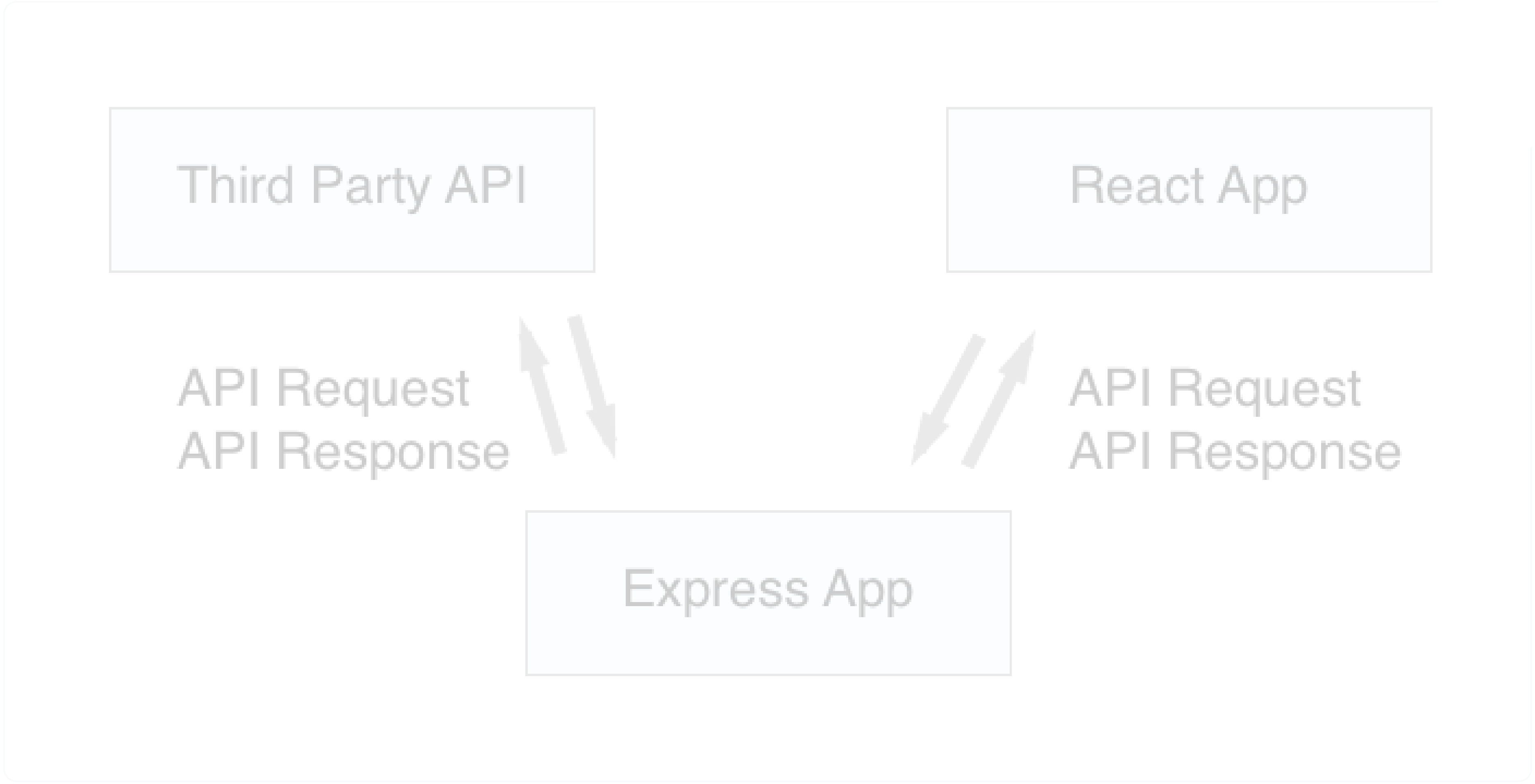
Browse N

APIs mentione



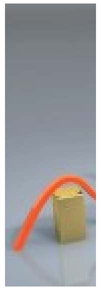
Browse N

From a software architecture standpoint, our app will work like this:



1. The user will **type in** a word and press the **submit button**
2. Our **React app** will send an API request to our **Express back-end**
3. The **Express back-end** will send another request to a **third party API**
4. When the **third party API** responds, our **Express back-end** gets a callback
5. The **Express back-end** will respond to our **React app** on the front-end
6. When the **React app** receives this data, it will **store it** in component state
7. Then, **React** will re-render our App component and **show Word Associations**!

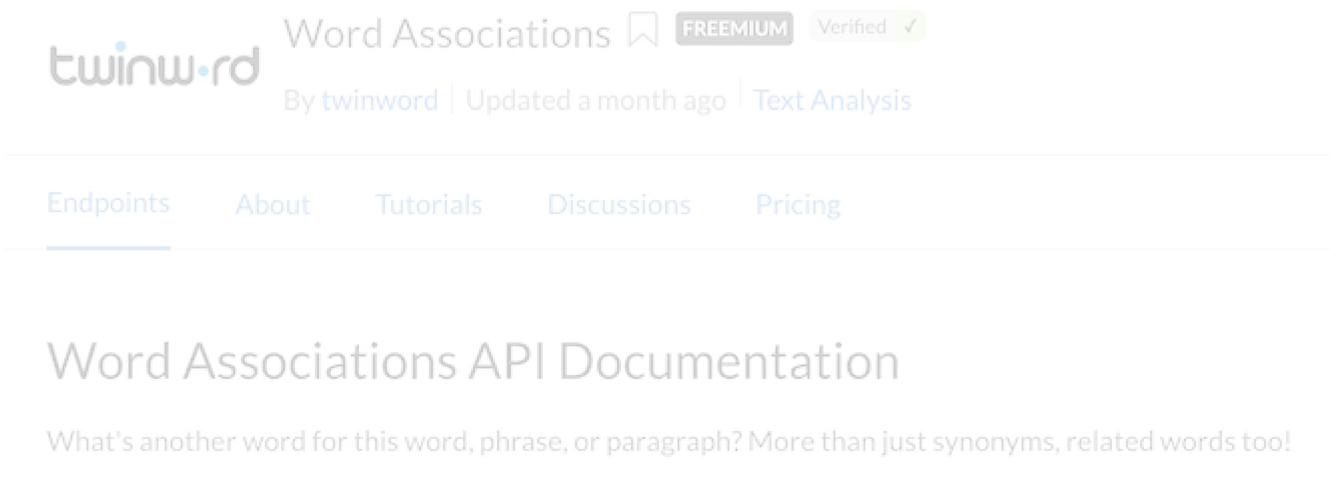
APIs mentione



Browse N

This architecture means we'll need a folder with two subfolders: one for the client, and one for the server. In other words, one for the front-end and one for the back-end.

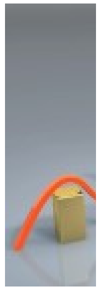
Step 1: Subscribe to the Word Associations API



Word Associations API Documentation

What's another word for this word, phrase, or paragraph? More than just synonyms, related words too!

APIs mentione



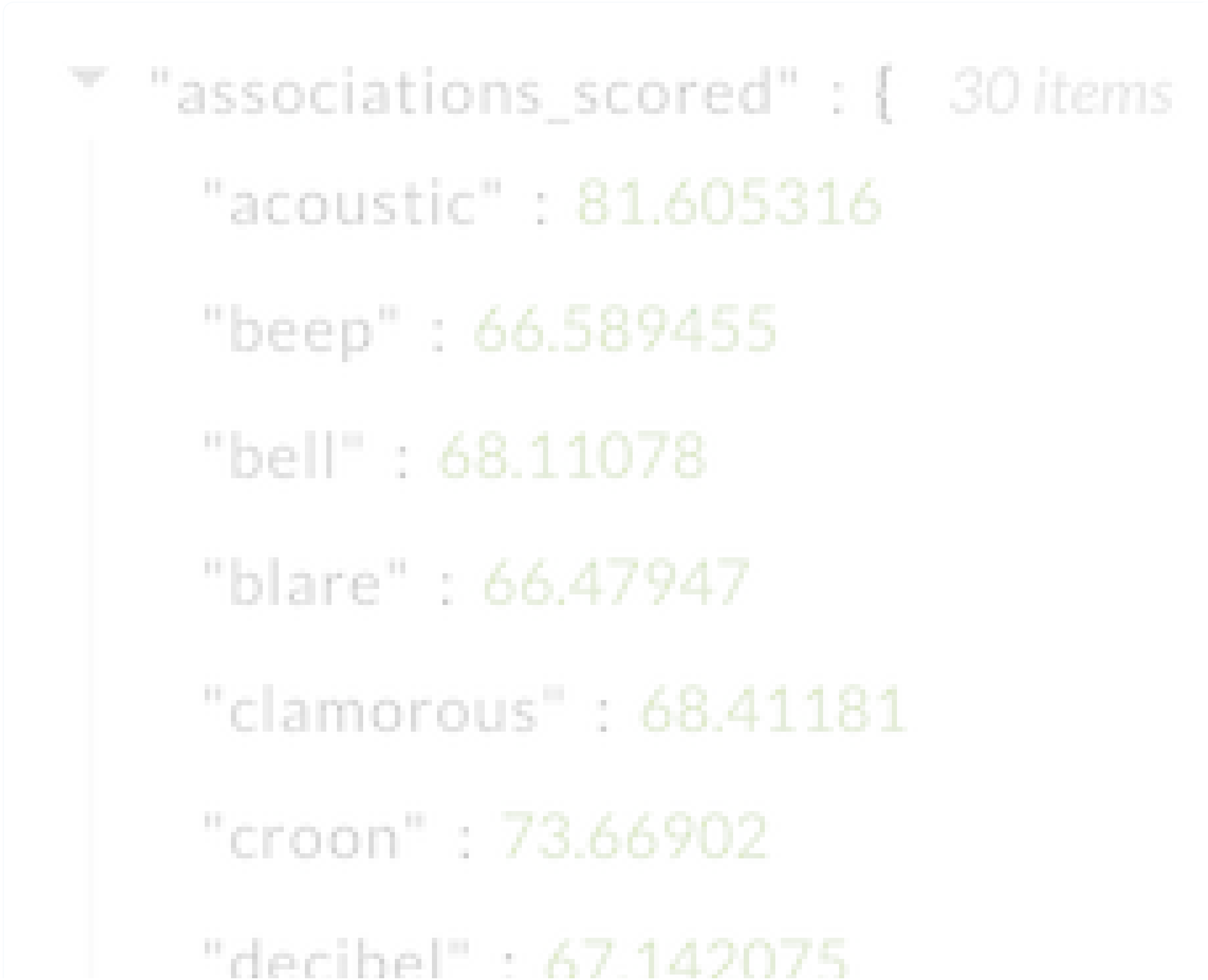
Browse N

First, visit the Word Associations page on RapidAPI. This API has an endpoint that will give us plenty of useful information, in a compact format that's easy to work with.

Tip: Not sure what an API Endpoint is exactly? Check out What is an API Endpoint?

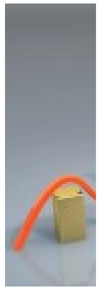
In particular, when you give this API's endpoint a word, it will give you a key-value mapping of words to scores. The words (in string form) represent associated words, and the score of each (which will be a floating point number) represents how closely the word relates to your original word. The higher the number, the closer the word is related.

For example, if we give this API the word "sound", we get back the following JSON object:



```
▼ "associations_scored" : {   30 items
    "acoustic" : 81.605316
    "beep" : 66.589455
    "bell" : 68.11078
    "blare" : 66.47947
    "clamorous" : 68.41181
    "croon" : 73.66902
    "decibel" : 67.142075
```

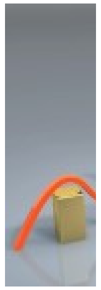
APIs mentione



Browse N

APIs mentione

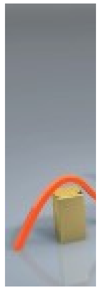
Because this API is so useful, it comes with a very small and very reasonable cost. Our tests will cost the sar
gum ball from the quarter machine at your local grocery store. So check out the Pricing page for this API to
subscribe to it.



Browse N

	Basic $0.00 Select Plan For individuals who just want the essentials to get started quickly
Objects	
requests	10000 / month quota + $0.003 each

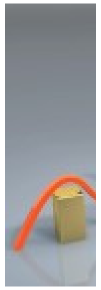
APIs mentione



Browse N

Note: if you're not already signed up on RapidAPI, you'll need a free account in order to do this step. And as bonus, a free account will give you access to many more APIs instantly.

APIs mentione



Browse N

Step 2: Prepare our Express back-end

First let's make our back-end:

```
$ mkdir server
$ cd server
$ npm init -y
$ npm install express
$ npm install unirest
$ npm install -g nodemon
```

We're not doing anything too fancy here:

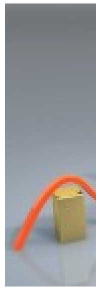
1. We create our "server" directory and go into it. This will live side-by-side with our "client" directory.
2. Then we set up NPM inside it. You'll need to have a somewhat-recent version of NPM installed.
3. We install Express for our web server, Unirest for API calls, and nodemon to make our lives easier.
4. Then we run nodemon, which will restart our web server every time our source code file changes.

Step 3: Create our back-end Express app

The Hello World of Express is extremely short, only about 10 lines:

APIs mentione

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```



Browse N

This just listens on the root path of the web server, and always responds with "Hello World!". Great starting point for our purposes.

Remember from our earlier steps, that our Express back-end is going to be the one calling our third party API. In this case, it will call the Word Associations API endpoint.

Tip: Need a refresher on making API Requests in Node.js? Check out How to use an API with Node.js.

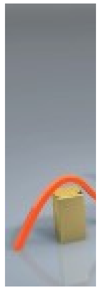
So let's find the Code Snippet for this API endpoint. It should look something like this:

Code Snippets

(Node.js) Unirest Install SDK Copy Code

```
var unirest = require("unirest");

var req = unirest("GET", "https://twinword-word-associations-v1.p.rapidapi.com/associations/

req.query({
  "entry": "sound"
});

req.headers({
  "x-rapidapi-host": "twinword-word-associations-v1.p.rapidapi.com",
  "x-rapidapi-key": "YOUR_RAPID_API_KEY_WILL_APPEAR_HERE",
  "useQueryString": true
});


req.end(function (res) {
  if (res.error) throw new Error(res.error);

  console.log(res.body);
});
```

APIs mentione



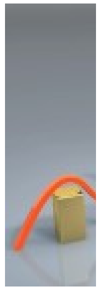
Browse N

Let's combine this with the Express Hello World we saw earlier. It's a bit trickier than normal, because both using the same variables, `req` and `res` to represent requests and responses. So we'll need to rename one of Since it's convention to use these short variable names in our Express routes, we'll leave those alone, and ch the variable names for our API call instead.

So paste the following code into your `server/index.js` file:

```
const express = require('express');
const app = express();
const port = 3001;
const unirest = require("unirest");

app.get('/api/associations/:word', (req, res) => {
  const request = unirest("GET", "https://twinword-word-associations-
v1.p.rapidapi.com/associations/");
  request.query({ "entry": req.params.word });
  request.headers({
    "x-rapidapi-host": "twinword-word-associations-v1.p.rapidapi.com",
    "x-rapidapi-key": "YOUR_RAPID_API_KEY_GOES_HERE",
    "useQueryString": true
  });

  request.end(function (response) {
    if (response.error) throw new Error(response.error);

    res.json(response.body.associations_scored || {});
  });

});
```

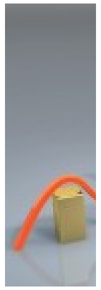
APIs mentione



Browse N

```
app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

APIs mentione



Browse N

We're doing a few things here:

1. We changed our route to `/api/associations/:word`, which uses a route-parameter. Check out more on rou parameters in the Express Routing page. But in short, this means that if we call `GET /api/associations/hell` then `req.params.word === "hello"`.
2. We make our call to Word Associations API using Unirest. We pass the word into the query parameter "entry" which this API endpoint takes as its only parameter.
   - *Note:* make sure you copy your RapidAPI key into this code from your Code Snippet!

3. Finally, when the Word Associations API gives us a response back, we respond back to the web client that made the request as JSON. (This will be our React client soon.) We look for the `associations_scored` key in the response body. If it doesn't exist, we just return an empty object.

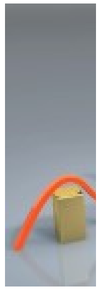
That's all we have to do! Now let's run our server:

```
$ nodemon
```

Now let's try it out! Since this is a GET route, we can use our browser.

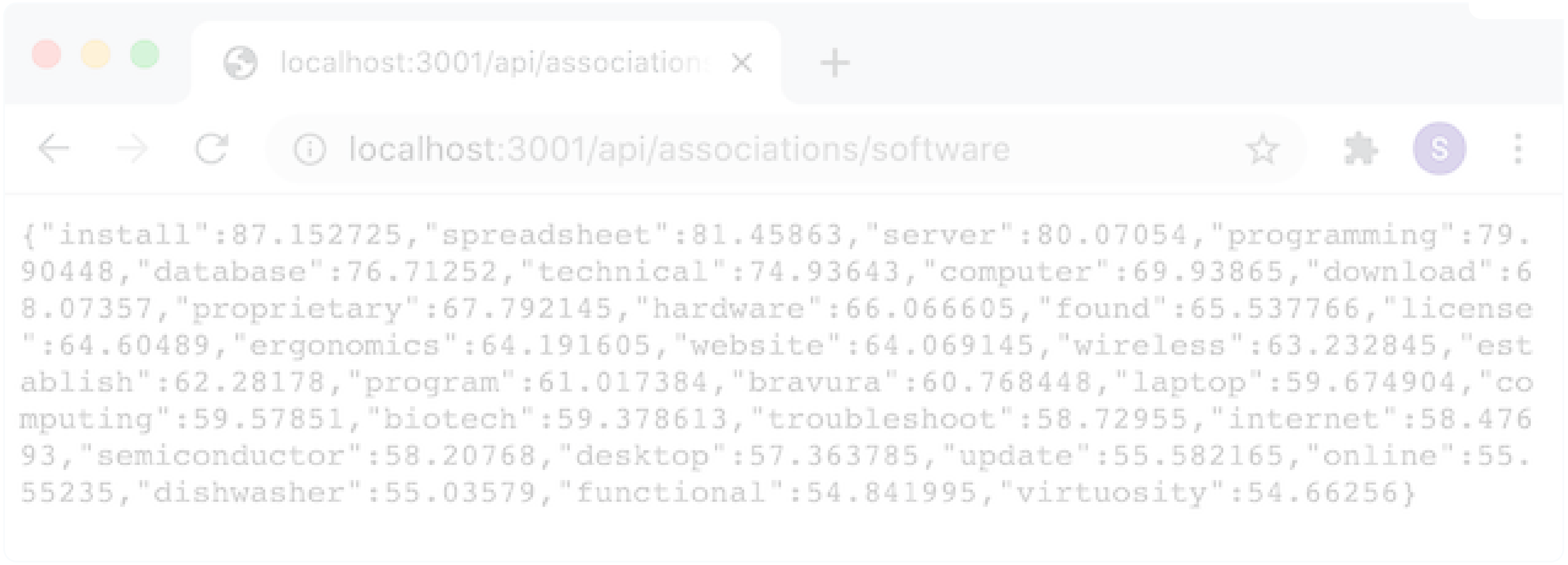
APIs mentione

Note: React apps generally use newer JavaScript features that require a modern browser. Older browsers v
require more polyfills and thus be slower, or might not be able to run some features at all. If you're using an
browser such as Internet Explorer, I recommend upgrading to the new Microsoft Chromium Edge browser,
although Google Chrome is also a popular choice for developers.



Navigate to http://localhost:3001/api/associations/software and you'll see something like this:

Browse N


localhost:3001/api/association
localhost:3001/api/associations/software
S

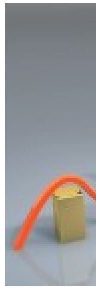
```
{"install":87.152725,"spreadsheet":81.45863,"server":80.07054,"programming":79.
90448,"database":76.71252,"technical":74.93643,"computer":69.93865,"download":6
8.07357,"proprietary":67.792145,"hardware":66.066605,"found":65.537766,"license
":64.60489,"ergonomics":64.191605,"website":64.069145,"wireless":63.232845,"est
ablish":62.28178,"program":61.017384,"bravura":60.768448,"laptop":59.674904,"co
mputing":59.57851,"biotech":59.378613,"troubleshoot":58.72955,"internet":58.476
93,"semiconductor":58.20768,"desktop":57.363785,"update":55.582165,"online":55.
55235,"dishwasher":55.03579,"functional":54.841995,"virtuosity":54.66256}
```

This is a great sign! It works beautifully, and we can already imagine how we'll integrate it into our front-end.

Step 4: Setup our front-end React app

APIs mentione

To help us set up our React app much easier, we'll use Create React App. This will setup a professional development environment that takes care of everything for us from compiling CSS, transforming JSX, and r a developer server with hot-loading enabled.



```
$ npm init react-app client
$ cd client
$ npm start
```

Browse N

Note: The `npm init` command will require a more recent version of NPM (version 6 or higher). If you have an earlier version installed, see Create React App's Getting Started page.

These commands will create a new React app environment in the "client" directory, open it up, and start the default React demo app.

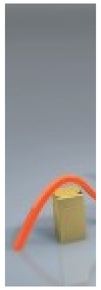
Next, open the newly-created "client" directory in your favorite IDE. If you don't have a favorite, I recommend VS Code, a free IDE with many powerful features built-in enabled by default.

Then, open up `src/App.js` and replace the `App` component with the following code:

```
function App() {
  const [word, setWord] = React.useState('software');
  const [associations, setAssociations] = React.useState(null);
  const getAssociations = () => {
    fetch('/api/associations/' + word)
      .then(result => result.json())
      .then(body => setAssociations(body));
  };

  return (
    <div className="app">
      <h1>Word Associations Map</h1>
      <input value={word} onChange={e => setWord(e.target.value)} />
      <button onClick={getAssociations}>Find Associations</button>
      {associations && (
        Object.keys(associations).length === 0
          ? <p>No results</p>
          : <div>
            {Object.entries(associations).map(([association, score]) => (
              <span s[illegible]le={{ fontSize: Math.pow(score, 2) / 200 }}>
                {association}
                {' '}
              </span>
            ))}
          </div>
      )}
    </div>
  );
}
```

The `getAssociations` function is how our App component interacts with our Express backend:

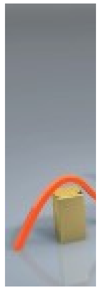
APIs mentione



Browse N

1. It uses the modern `fetch()` function to call our backend. Not all browsers support this, so if your target have older browsers, you may need a fetch-polyfill.
2. After calling the backend, it transforms the result into JSON, and sets as the new value of the `associati` array. This uses the new React Hooks feature.
3. That's it! That's all it takes to integrate our front-end React app with our back-end Express API!

APIs mentione



Browse N

The rest of this app is straightforward: set up some React state, show a form, and show the results if there a

We only do one fancy trick, and it's just a little bit of math to magnify the difference between scores, which we then use to set the font-size of each word.

Tip: For a more in-depth look at calling APIs from React, check out How To Use an API with ReactJS.

Step 5: Add some CSS styling to our React app

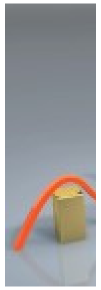
Our app works, but it isn't super pretty. With just a tiny bit of CSS, we can make it look respectable and professional.

So replace the contents of `src/App.css` with the following code:

```
body { background: #f7f7f7; }
.app { margin: 3em; width: 24em; }
```

```
input,
button {
  padding: 0.5em 0.75em;
  margin-right: 1em;
  border-radius: 4px;
  outline: none;
  border: 1px solid transparent;
  font: inherit;
}

input       { border-color: #888; }
input:focus { border-color: #17f; }

button        { background: #17f; color: #fff; }
button:active { background: #15d; }
```

APIs mentione



Browse N

Step 6: Make React proxy API calls to Express

Our app *looks* done, but it doesn’t quite work yet! Why? Because when we make our call to `/api/associations/software`, the API call is actually going to be handled the React development server that Create React App setup for us! Our Express app is never going to see it!

We have two solutions:

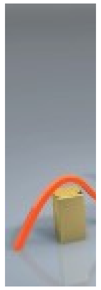
1. We could hardcode the app to make the API call based on an absolute URL, like `fetch("http://localhost:3001/api/associations/software")`. But this isn’t very flexible. We have to know the URL

ahead of time. And we have to remember to change it when we deploy our app.

2. We could use Create React App's proxy feature. Then the development server will proxy our request t

backend of our choosing.

APIs mentione



Browse N

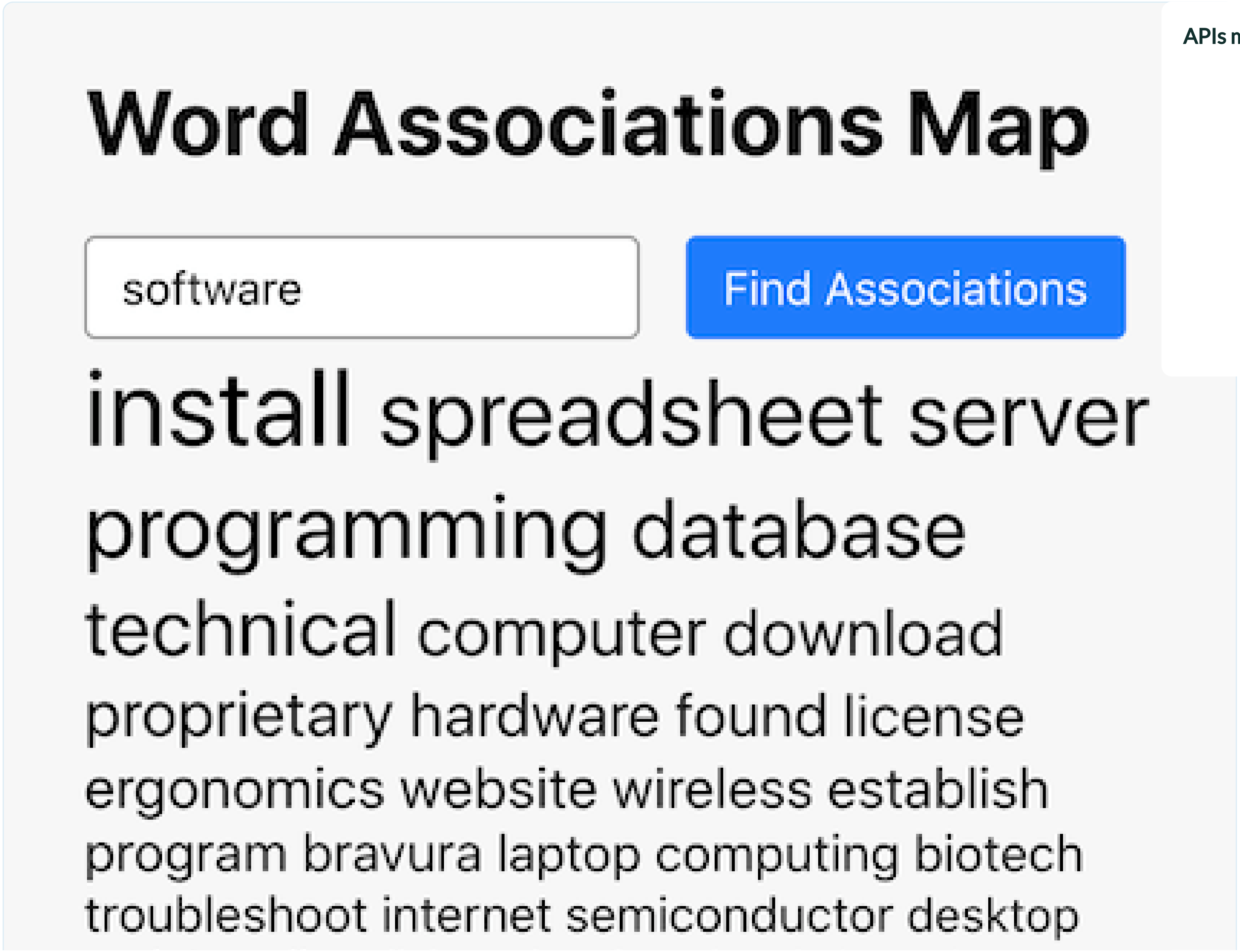
The second solution is by far better. So add the following to your `package.json` file in your **client** directory:

```
"proxy": "http://localhost:3001",
```

Note: Both directories have a `package.json` file! But we want to add this to the React app, which is in the client folder, *not* the server folder.

Step 7: Try it!

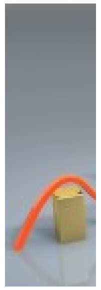
We finally have a working app:



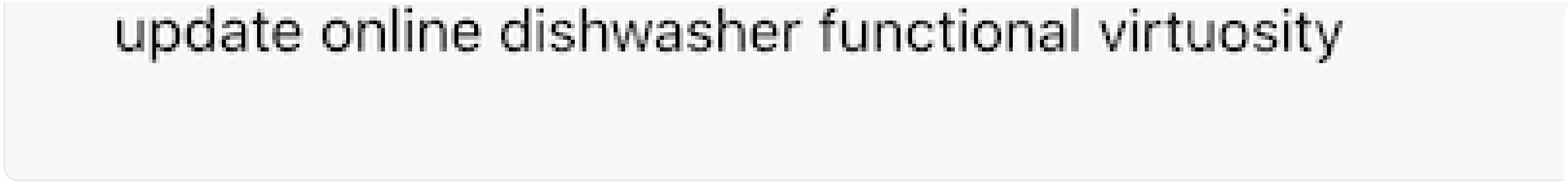
Word Associations Map

install spreadsheet server programming database technical computer download proprietary hardware found license ergonomics website wireless establish program bravura laptop computing biotech troubleshoot internet semiconductor desktop

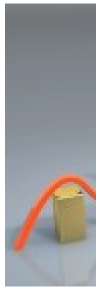
APIs mentione



Browse N


update online dishwasher functional virtuosity

APIs mentione



Browse N

Conclusion

With just a little bit of effort, we created a professional app that uses **React** for the front-end, **Express** for tl back-end, and integrates with third party APIs. We learned how to integrate React with Express by proxying requests. And we saw how to design our very own internal API, too.

FAQ

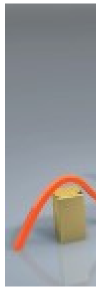
How does Express integrate with React?

There are many ways to integrate Express with React. But if you just remember that React isn't an application, but a set of files, then you'll have no problem integrating Express with React in any way you'd like. One simple way is to use Express's static file server to server files to the browser that contain your React app.

Do I need Express with React?

APIs mentione

There are many ways to host React apps. It's common to host React apps on static file servers such as Amazon S3 and CloudFront. This solution doesn't need Express at all. And with serverless architecture becoming more popular, React apps can call APIs instead of traditional back-ends.



Browse N

Is React JS frontend or backend?

React was developed for the front-end, but has proven very useful for the backend, which is often called Server-Side Rendering, or SSR. Using this technique is also useful for SEO, because your React pages can be served to search engines which can read and index them, whereas this isn't possible in just plain React.

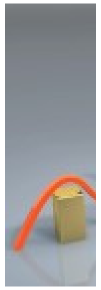
How do you serve a React app?

You can either use a developer environment like Create React App, or you can serve the files yourself using any web server software, such as Express. Just remember, the React app itself runs inside the browser.

View the Best Free APIs List

2.7/5 - (4

APIs mentione

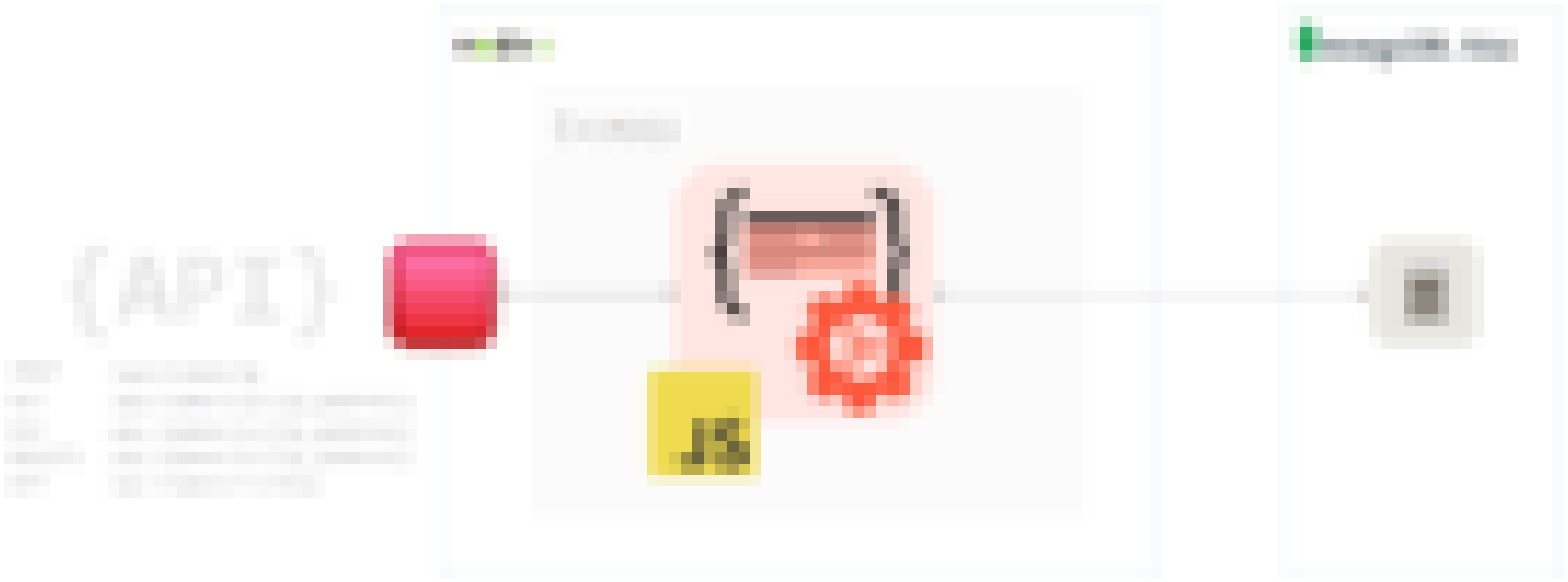


« How To Use The AliExpress API With Ruby

How to Make REST API Calls in React Native »

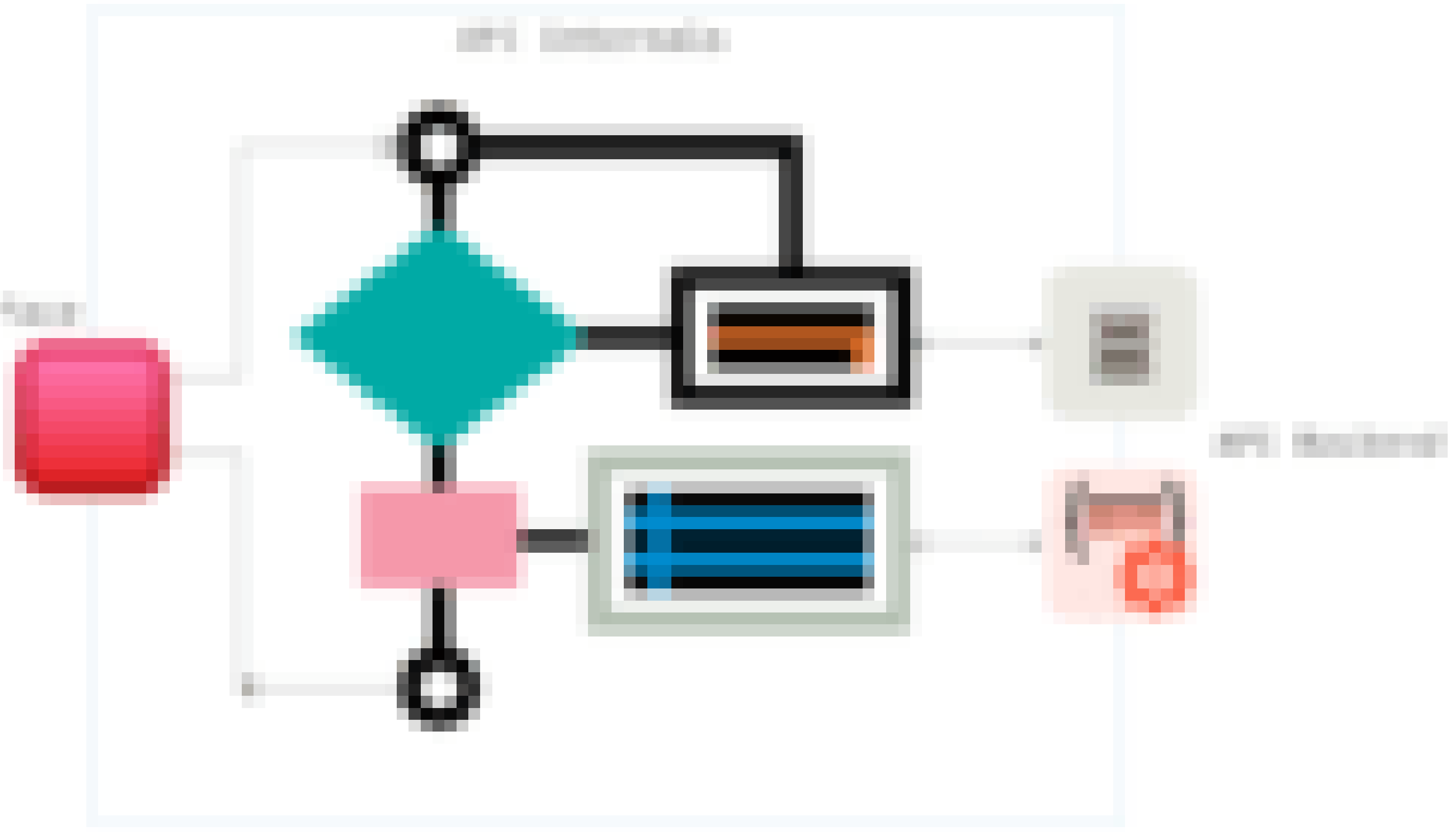
Browse N

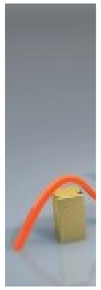
Related Blog Posts



How to build a REST API with Node.js (MongoDB & Express) – Part 2

APIs mentione



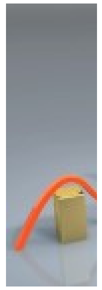


Browse N

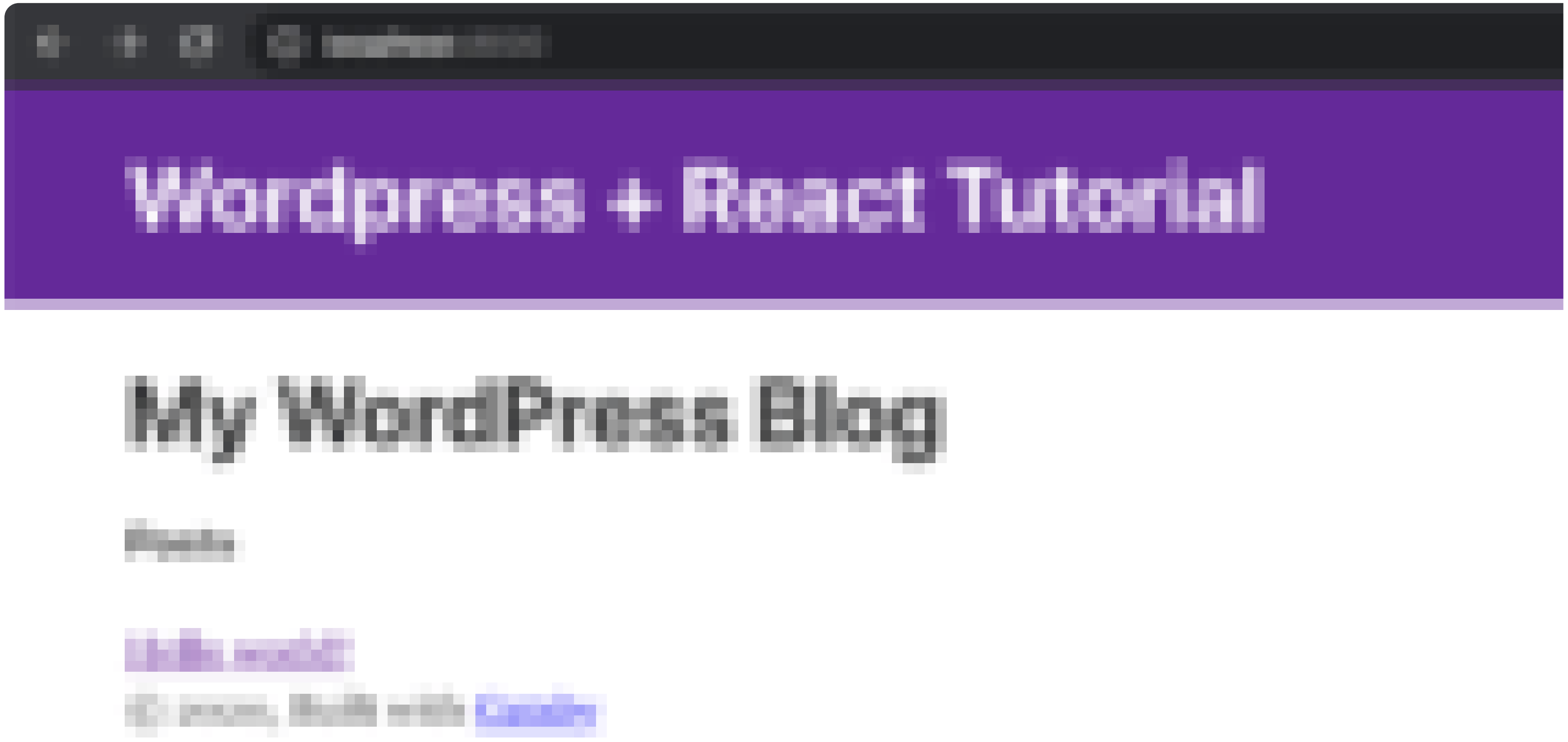
How to build a REST API with Node.js – Part 1

[React API Authentication & Authorization](https://rapidapi.com/blog/create-react-app-express/)

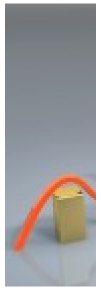
APIs mentione



Browse N



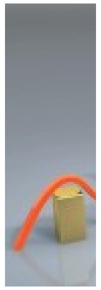
APIs mentione



Browse N

How to use WordPress with React (WordPress React API Tutorial)

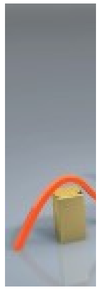
APIs mentione



Browse N

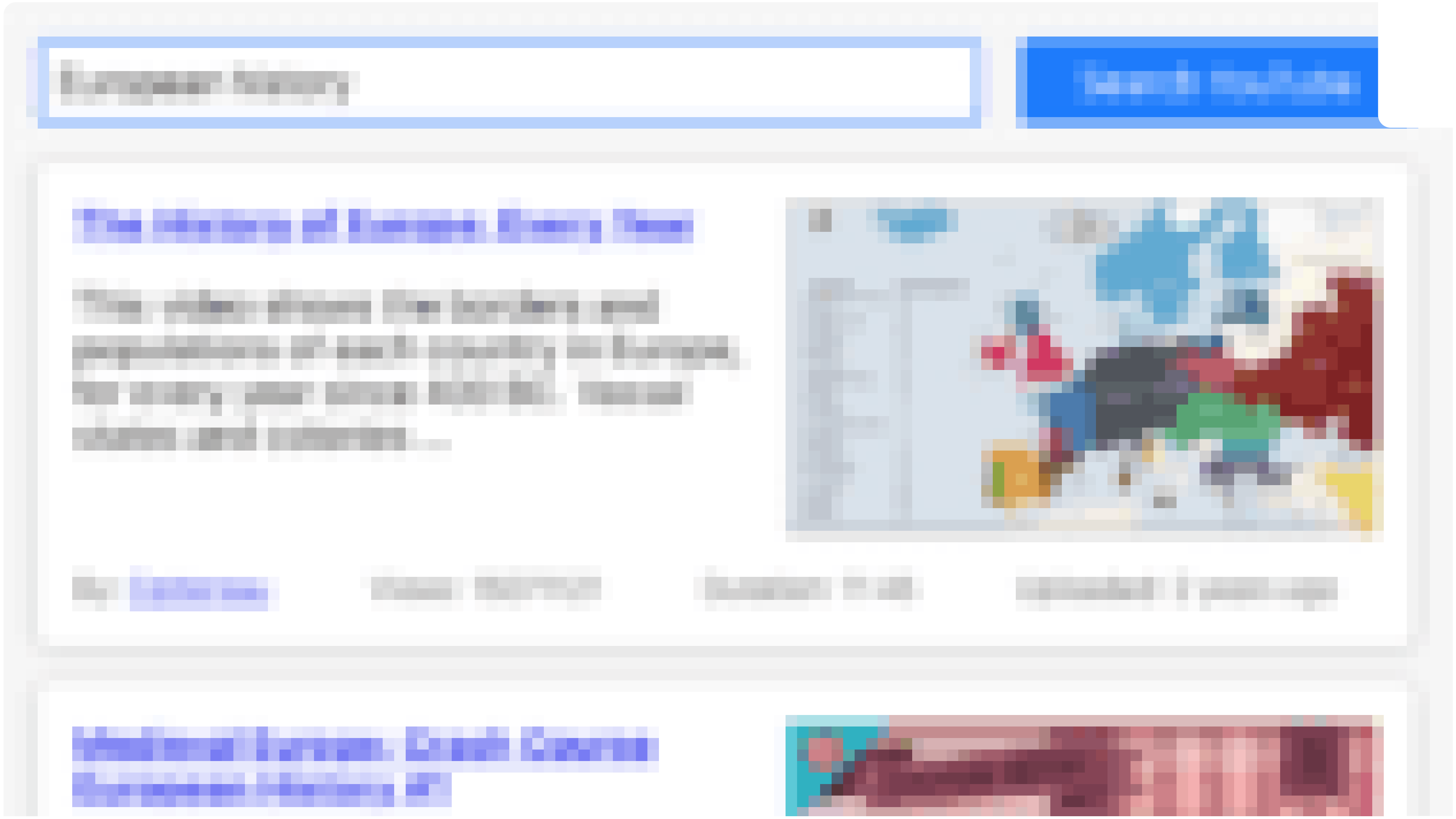
[illegible]

APIs mentione

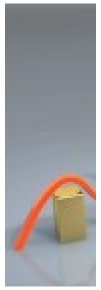


Browse N

How to Make REST API Calls in React Native

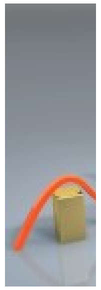


APIs mentione



Browse N

APIs mentione



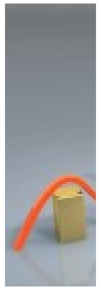
Browse N

How to Build a YouTube App with React (React Youtube API Tutorial)

Filed Under: Express.js API Tutorials , JavaScript API Tutorials , React API Tutorials , REST API Tutorials

Tagged With: #express , #express.js , #react , #react.js , #reactjs

APIs mentione



Browse M

Steven

With over a decade of professional front-end experience, Steven uses his accumulated knowledge and experience to write instructive technical articles about the latest technologies such as JavaScript, TypeScript, React.js, and many APIs.

Search this website

Search

Comments

Rajat says
MAY 31, 2021 AT 7:14 AM

On step 2 i ran into a problem:

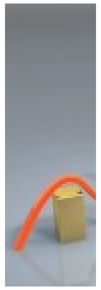
Could not find a declaration file for module 'unirest'.

i have uninstalled and re-installed this module, but the error still persists. If i proceed with the steps the website shows:

Cannot GET /

Reply

APIs mentione



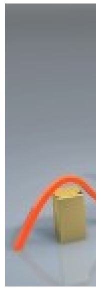
Browse N

Build anything with APIs, faster.

Discover, evaluate, and integrate with any API. RapidAPI is the world's largest API Hub with over 4 Million developers and 35,000 APIs.

Browse APIs »

APIs mentione



Browse N

RapidAPI Learn

API Guides

API Courses

API Glossary

API Testing

API Management

How to use an API

For API Providers

Most Popular APIs

For Developers

Free APIs List

API Examples

Learn REST API

Learn GraphQL

About

Team

Careers

Contact Us

Write for Us

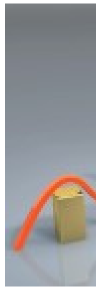
API Directory

Press Room

Privacy Policy

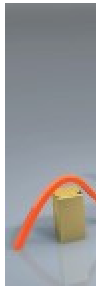
Terms of Use

APIs mentione



Browse N

APIs mentione



Browse N