

# How to Create a React App with a Node Backend: The Complete Guide



Reed Barger

A React frontend connected to a Node backend is a rock-solid combination for any application you want to build.

This guide is designed to help you create full-stack projects with React as easily as possible.

Let's see how to set up an entire project using React and Node from scratch and deploy it to the web.

## Tools You Will Need

Make sure Node and NPM are installed on your computer. You can download both at [nodejs.org](https://nodejs.org) (NPM is included in your Node installation)

Use a code editor of your choice. I am using and would personally recommend using VSCode. You can download VSCode at [code.visualstudio.com](https://code.visualstudio.com).

Make sure you have Git installed on your computer. This is necessary for deploying our application with Heroku. You can get it at [git-scm.com](https://git-scm.com)

An account at [heroku.com](https://heroku.com).

# Step 1: Create your Node (Express) backend

First create a folder for your project, called `react-node-app` (for example).

Then, drag that folder into your code editor.

To create our Node project, run the following command in your terminal:

```
npm init -y
```

This will create a `package.json` file which will allow us to keep track of all our app scripts and manage any dependencies our Node app needs.

Our server code will live in a folder of the same name: `server`. Let's create that folder.

In it, we'll place a single file, out of which we'll run our server: `index.js`.

We'll use Express to create a simple web server for us which runs on port 3001 if no value is given for the environment variable `PORT` (Heroku will set this value when we deploy our app).

```
// server/index.js

const express = require("express");

const PORT = process.env.PORT || 3001;

const app = express();

app.listen(PORT, () => {
  console.log(`Server listening on ${PORT}`);
});
```

Then in our terminal, we will install Express as a dependency to use it:

```
npm i express
```

After that, we will create a script in package.json that will start our web server when we run it with `npm start` :

```
// server/package.json

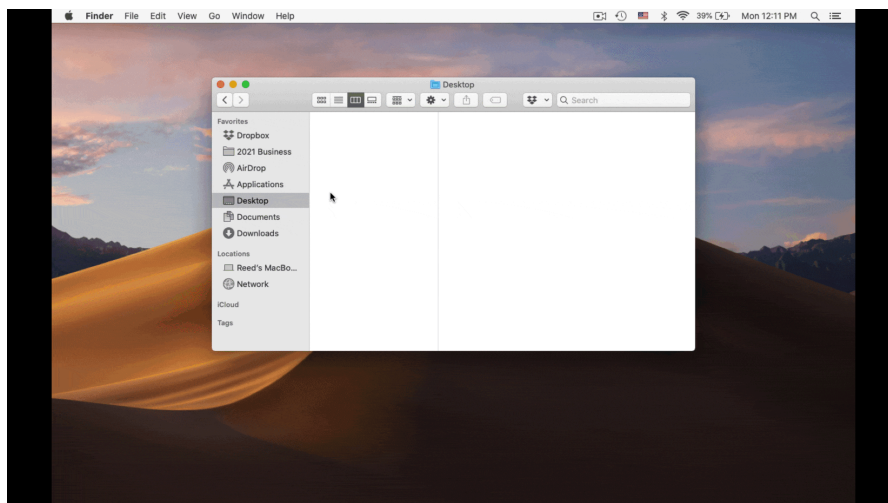
...
"scripts": {
  "start": "node server/index.js"
},
...
```

Finally, we can run our app using this script by running `npm start` in our terminal and we should see that it is running on port 3001:

```
npm start
```

```
> node server/index.js
```

```
Server listening on 3001
```



## Step 2: Create an API Endpoint

We want to use our Node and Express server as an API, so that it can give our React app data, change that data, or do some other operation only a server can do.

In our case, we will simply send our React app a message that says "Hello from server!" in a JSON object.

The code below creates an endpoint for the route `/api`.

If our React app makes a GET request to that route, we respond (using `res`, which stands for response) with our JSON data:

```
// server/index.js
...

app.get("/api", (req, res) => {
  res.json({ message: "Hello from server!" });
});

app.listen(PORT, () => {
  console.log(`Server listening on ${PORT}`);
});
```

*Note: Make sure to place this above the `app.listen` function.*

Since we've made changes to our Node code, we need to restart our server.

To do that, end your start script in the terminal by pressing Command/Ctrl + C. Then restart it by running `npm start` again.

And to test this, we can simply visit `http://localhost:3001/api` in our browser and see our message:

```
1  const express = require("express");
2
3  const PORT = process.env.PORT ||
3001;
4
5  const app = express();
6
7  app.listen(PORT, () => {
8    console.log(`Server listening on ${
{PORT}}`);
9  });
10
```

# Step 3: Create your React frontend

After creating our backend, let's move to the frontend.

Open another terminal tab and use create-react-app to create a new React project with the name `client`:

```
npx create-react-app client
```

After that, we will have a React app with all of its dependencies installed.

The only change we have to make is to add a property called `proxy` to our `package.json` file.

This will allow us to make requests to our Node server without having to provide the origin it is running on (`http://localhost:3001`) every time we make a network request to it:

```
// client/package.json
```

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

Then we can start up our React app by running its start script, which is the same as our Node server. First make sure to `cd` into the newly-created client folder.

After that, will start up on `localhost:3000`:

```
cd client  
npm start
```

Compiled successfully!

You can now view client [in](#) the browser.

Local: <http://localhost:3000>

A screenshot of a code editor with two tabs: 'index.js' and 'package.json'. The 'index.js' tab is active, showing a JavaScript file with the following code:

```
1 const express = require("express");
2
3 const PORT = process.env.PORT || 3001;
4
5 const app = express();
6
7 app.get("/api", (req, res) => {
8   res.json({ message: "Hello from server!" });
9 })
```

The terminal window at the bottom shows the command `react-node-app@1.0.0 start /Users/reedbarger/Desktop/react-node-app` and the output `> node server/index.js` followed by `Server listening on 3001`. The status bar at the bottom indicates 'Ln 13, Col 4', 'Spaces: 2', 'UTF-8', 'LF', 'JavaScript', 'ESLint', and 'Prettier: ✓'.

## Step 4: Make HTTP Requests from React to Node

Now that we have a working React app, we want to use it to interact with our API.

Let's see how to fetch data from the `/api` endpoint that we created earlier.

To do so, we can head to the `App.js` component in our `src` folder and make an HTTP request using `useEffect`.

We will make a simple GET request using the Fetch API to our backend and then have our data returned as JSON.

Once we have the data returned to us, we will get the `message` property (to grab our greeting that we sent from the server) and then put it in a state variable called `data`.

This will allow us to display that message in our page if we have it. We are using a conditional in our JSX to say that if our data is not there yet, show the text "Loading...".

```
// client/src/App.js

import React from "react";
import logo from "../logo.svg";
import "../App.css";

function App() {
  const [data, setData] = React.useState(null);

  React.useEffect(() => {
```

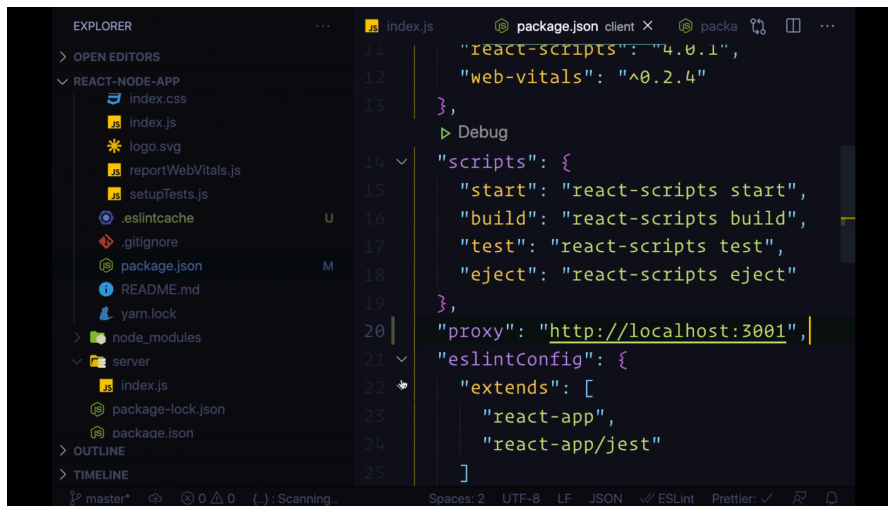
```

    fetch("/api")
      .then((res) => res.json())
      .then((data) => setData(data.message));
  }, []);

return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="Logo" />
      <p>{data ? "Loading..." : data}</p>
    </header>
  </div>
);
}

export default App;

```



## Step 5: Deploy your app to the web with Heroku

Finally, let's deploy our application to the web.

First, within our client folder, make sure to remove the Git repo that is automatically initialized by create-react-app.

This is essential to deploy our app, because we are going to set up a Git repo in the root folder of our project ( react-node-app ), not in client :

```
cd client
```

When we deploy, both our Node backend and React frontend are going to be served on the same domain (i.e. mycoolapp.herokuapp.com).

We see how our requests are being handled by our Node API, so we need to write some code that will display our React app when it is requested by our user (for example, when we go to the home page of our app).

We can do this back in `server/index.js` by adding the following code:

```
// server/index.js
const path = require('path');
const express = require('express');

...

// Have Node serve the files for our built React app
app.use(express.static(path.resolve(__dirname, '../client/build')));

// Handle GET requests to /api route
app.get("/api", (req, res) => {
  res.json({ message: "Hello from server!" });
});

// All other GET requests not handled before will return our React app
app.get('*', (req, res) => {
  res.sendFile(path.resolve(__dirname, '../client/build', 'index.html'));
});
```

This code will first allow Node to access our built React project using the `express.static` function for static files.

And if a GET request comes in that is not handled by our `/api` route, our server will respond with our React app.

**This code allows our React and Node app to be deployed together on the same domain.**

Then we can tell our Node App how to do that by adding a `build` script to our server `package.json` file that builds our React app for production:



```
// server/package.json
```

```
...  
"scripts": {  
  "start": "node server/index.js",  
  "build": "cd client && npm install && npm run build"  
},  
...
```

I would also recommend providing a field called "engines", where you want to specify the Node version you are using to build your project. This will be used for deployment.

You can get your Node version by running `node -v` and you can put the result in "engines" (i.e. 14.15.4):

```
// server/package.json  
  
"engines": {  
  "node": "your-node-version"  
}
```

After this, we're ready to deploy using Heroku, so make sure you have an account at [Heroku.com](https://heroku.com).

Once you are signed in and are looking at your dashboard, you'll select New > Create New App and provide a unique app name.

After that, you'll want to install the Heroku CLI on your computer so you can deploy your app whenever you make any changes using Git. We can install the CLI by running:

```
sudo npm i -g heroku
```

Once that's installed, you will log in to Heroku through the CLI using the `heroku login` command:

```
heroku login
```

Press any key to login to Heroku

Once you have logged in, just need to follow the deployment instructions for our created app in the "Deploy" tab.

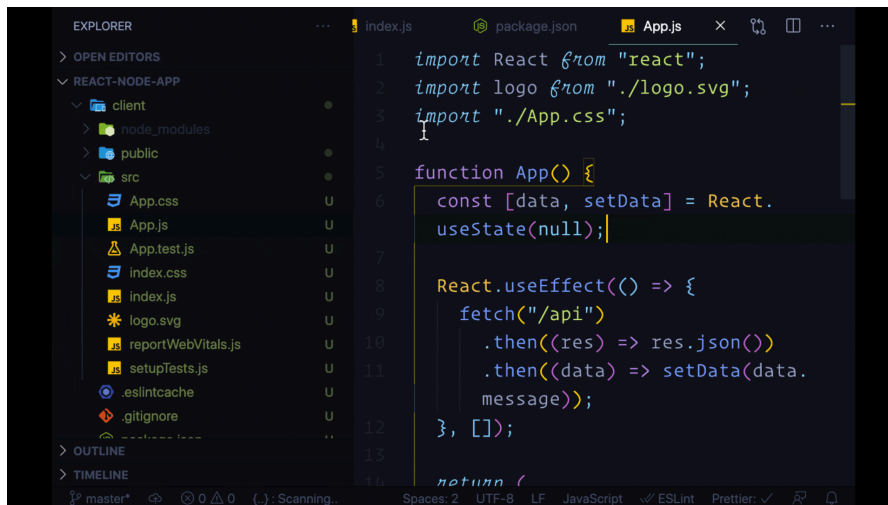
The following four commands will initialize a new Git repo for our project, add our files to it, commit them, and add a Git remote for Heroku.

```
git init
heroku git:remote -a insert-your-app-name-here
git add .
git commit -am "Deploy app to Heroku"
```

Then the very last step is to publish our app by pushing the Heroku Git remote we just added using:

```
git push heroku master
```

Congratulations! Our full-stack React and Node app is live! 🎉



When you want to make changes to your app going forward (and deploy them), you just have to use Git to add your files, commit them and then push to our Heroku remote:

```
git add .
git commit -m "my commit message"
git push heroku master
```

# Become a Professional React Developer


React is hard. You shouldn't have to figure it out yourself.

I've put everything I know about React into a single course, to help you reach your goals in record time:

## Introducing: The React Bootcamp

It's the one course I wish I had when I started learning React.

Click below to try the React Bootcamp for yourself:

A promotional banner for the React Bootcamp. It features a dark blue background. On the left, the text "React Bootcamp" is written in a large, bold, blue font. Below it, in a smaller white font, is "Go from zero to React pro in one course." followed by "Become a React developer in record time." in a green font. At the bottom left is a green button with the text "START LEARNING NOW" in white. On the right side of the banner is a stylized illustration of a microchip with a React logo (a blue atom-like symbol) in the center.

**React Bootcamp**

Go from zero to React pro in one course.  
Become a React developer in record time.

**START LEARNING NOW**

*Click to get started*