adeva

LARAVEL

# Creating a Modern Application: Using Laravel With React.js

by **Farhan Hasin Chowdhury**

13 min read ·

I started my web development career as a Python programmer, and my go-to framework in those days was Django. I came across Laravel only 1.5 years into my career and started building stuff using the framework. Eventually, I got better at Laravel and reached a point where I would choose between the frameworks depending on requirements.

For example, if I saw that a project has pages that require something to happen in real-time, I would immediately choose Laravel over Django. Back in the day, Laravel used to come with Vue pre-installed and pre-configured with every new project.

Laravel has come a long way, and the options for creating modern, reactive web applications using the framework are many. There's Inertia.js, Alpine.js, Livewire, and more. But the framework hasn't said goodbye to the old ways of doing things. It's still possible and very straightforward to set up Vue or React with a Laravel project example.

as well as React.js developers out there, can proceed with the project safely.

I'll use Laravel 8 throughout the article (one of the reasons being the excellent Laravel security features of the version), but you should be able to follow along with future versions of the framework as well. The article assumes that you have experience working with both Laravel and ReactJs.

## Bootstrapping a New Project and Installing the Dependencies

Let's start our Laravel and React js process by creating a new Laravel project anywhere on your local computer. You can use whatever method of bootstrapping you may like. I prefer using the Laravel Installer in this case.

```
laravel new guide-to-laravel-and-react
```

Once the project is ready, open it in your favorite IDE or text editor. If you look inside the package.json file present in every new Laravel project, you'll see the following list of packages under the devDependencies section:

- axios - promise-based HTTP client
- laravel-mix - an elegant wrapper over webpack
- lodash - popular functional programming library
- postcss - utility tool for automating routine CSS operations

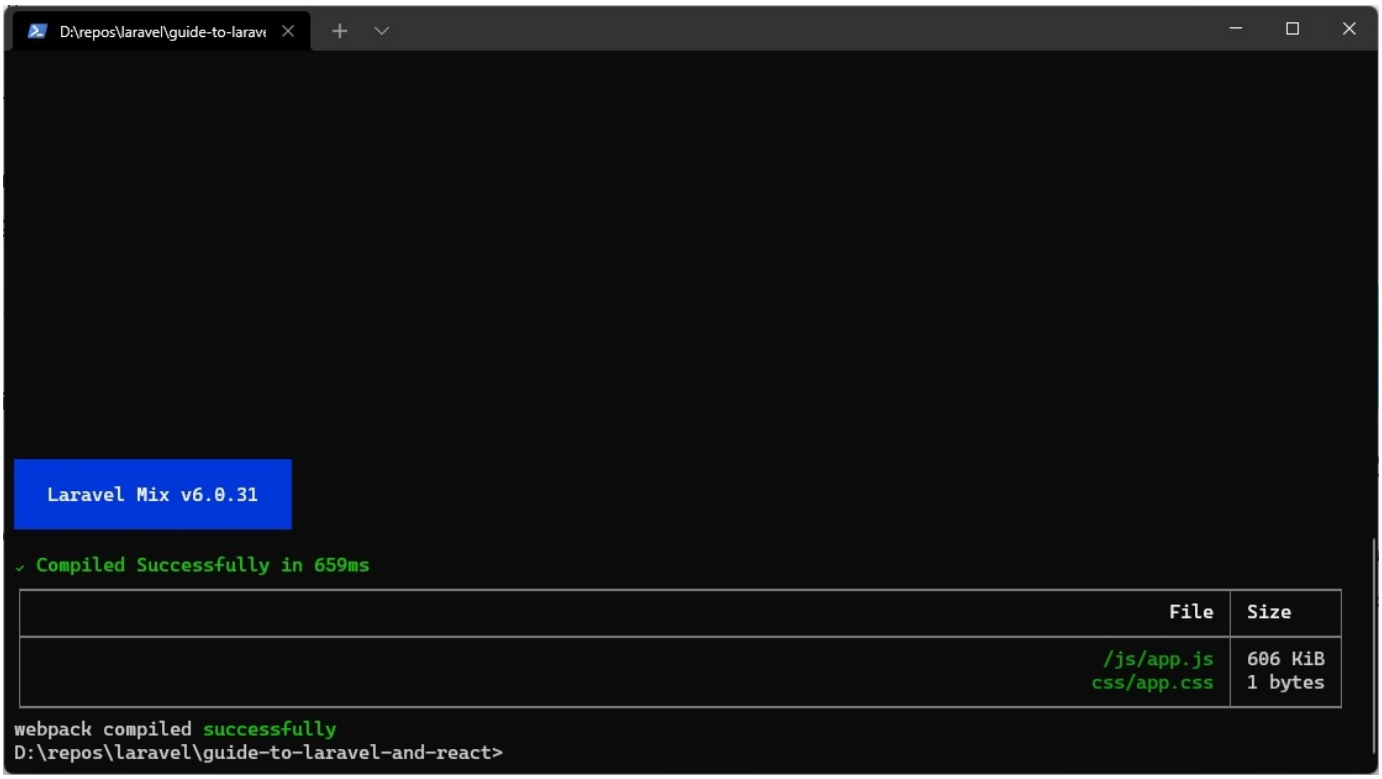
In this article, you'll be working with all of these libraries except for Lodash.

Now that you know about all these dependencies, let's go ahead and install them:

```
npm install
```

```
npm run dev
```

If everything works fine, you should see something as follows on your terminal:




Let me explain what just happened.

When you executed the npm run dev command, Laravel Mix compiled the **resources/js/app.js** file and the **resources/css/app.css** file into **public/js/app.js** and **public/css/app.css** files.

# Installing React and Writing Your First Component

Let's go ahead and start bringing React into the scene to see the whole process of using Laravel with React. First, you'll have to install two new JavaScript libraries. They are the **react** and **react-dom** libraries. To do so, execute the following command:

It'll install the libraries as development dependencies. The reason behind installing them as development dependencies is that you'll compile all your JavaScript code into a single file and attach that with your Blade templates.

Create a new file **resources/js/components/HelloReact.js** and put the following content in there:

```
// resources/js/components/HelloReact.js

import React from 'react';
import ReactDOM from 'react-dom';

export default function HelloReact() {
    return (
        <h1>Hello React!</h1>
    );
}

if (document.getElementById('hello-react')) {
    ReactDOM.render(<HelloReact />, document.getElementById('hello-re
}
```

It's a functional React component that mounts on any element with the id **"hello-react"** and prints **"Hello React!"** on the page. To make this usable throughout your entire application, you'll have to include this in your JavaScript code bundle. To do so, open the **resoureces/js/app.js** file and update its code as follows:

```
// resoureces/js/app.js

require('./bootstrap');

// React Components
require('./components/HelloReact')
```

Before you can use this updated code in your blade templates, you'll have to recompile it.

Previously you were writing only vanilla JavaScript in your code, but now you're writing React-related code. You'll have to change the Laravel Mix configuration so that it can compile the React component properly. To do so, open the **webpack.mix.js** file from the project root and update its code as follows:

```
// webpack.mix.js

const mix = require('laravel-mix');

mix.js('resources/js/app.js', 'public/js')
    .react()
    .postCss('resources/css/app.css', 'public/css', [
        //
    ]);
```

By default, the configuration file doesn't call the **react()** function. Calling this function will tell Laravel Mix to install the **babel-preset-react** plugin required for compiling React components.

After updating the Laravel Mix configuration, execute the **npm run dev** command once again. Keep in mind that the compilation will not happen this time. Instead, Laravel Mix will realize some libraries are missing and install them. So run the **npm run dev** command once again, and this time the execution should succeed.

Now that the component is ready to use, you'll have to prepare a blade template. Open the **resources/views/welcome.blade.php** file and update its content as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Laravel React</title>

    <script src="{{ asset('js/app.js') }}" defer></script>
</head>
<body>
    <div id="hello-react"></div>
</body>
</html>
```

Another essential aspect in the React with Laravel process is to make sure that you defer the script execution or put it just before the body tag ends. Also, regarding Laravel React, make sure to match the element ID with the ID you

Now start the development server by executing **php artisan serve** and visit http://localhost:8000 using your favorite browser. You should see something as follows:


If you see nothing on the screen, check the browser console for any errors. If you find something, resolve them and try again. Let's move on to the next Laravel Reactjs part, creating a counter component using React hooks.

## Creating a Counter Component Using React Hooks

Let's write a counter component using the state hook. Create a file **resources/js/Counter.js** and put the following content in it:

```
// resources/js/components/Counter.js
```

```
export default function Counter() {
  // Set the initial count state to zero, 0
  const [count, setCount] = useState(0);

  // Create handleIncrement event handler
  const handleIncrement = () => {
    setCount(prevCount => prevCount + 1);
  };

  // Create handleDecrement event handler
  const handleDecrement = () => {
    setCount(prevCount => prevCount - 1);
  };
  return (
    <div>
        <button onClick={handleDecrement}>-</button>
        <span> {count} </span>
        <button onClick={handleIncrement}>+</button>
    </div>
  );
}

if (document.getElementById('counter')) {
    ReactDOM.render(<Counter />, document.getElementById('counter'));
}

if (document.getElementById('counter')) {
    ReactDOM.render(<Counter />, document.getElementById('counter'));
}
```

Open the **resources/js/app.js** file and require it just like the previous one. Instead of executing the **npm run dev** command, again and again, you can execute the **npm run watch** command, which keeps an eye on your files and compiles them automatically on change.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Laravel React</title>

    <script src="{{ asset('js/app.js') }}" defer></script>
</head>
<body>
    <div id="hello-react"></div>
    <div id="counter"></div>
</body>
</html>
```

Refresh the page if it's still open in your browser, and you should see something as follows:


Try hitting the increment and decrement button, and you will see the number change. Anything you can do in a regular React component, you can do here. Even if you want to write class components, go ahead.

# Communicating With the Back-End

So far in this Laravel React turtorial, you've only worked in the front-end. In this section, I'll show you how you may communicate with the Laravel backend tutorial from your React components.

```
<?php

// routes/web.php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Route::post('count', function (Request $request) {
    return response()->json([
        'message' => $request->message,
    ]);
});
```

I won't overcomplicate things by introducing databases and stuff. This simple route receives a **string** named **message** from the front-end and sends it back.

Open the **resources/js/Counter.js** file and update its content as follows:

```
// resources/js/Counter.js

import axios from "axios";
import React, { useState } from "react";
import ReactDOM from 'react-dom';

export default function Counter() {
  // Set the initial count state to zero, 0
  const [count, setCount] = useState(0);

  // Create handleIncrement event handler
  const handleIncrement = () => {
    setCount(prevCount => prevCount + 1);
    notifyServer();
```

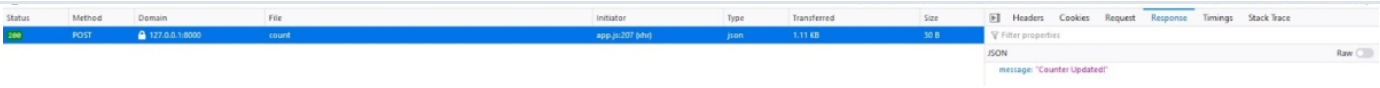
```
  // Create handleDecrement event handler
  const handleDecrement = () => {
    setCount(prevCount => prevCount - 1);
    notifyServer();
  };

  // Notifies the server about the change
  const notifyServer = () => {
      axios.post('/count', {
          message: 'Counter Updated!',
      })
  }

  return (
    <div>
        <button onClick={handleDecrement}>-</button>
        <span> {count} </span>
        <button onClick={handleIncrement}>+</button>
    </div>
  );
}

if (document.getElementById('counter')) {
    ReactDOM.render(<Counter />, document.getElementById('counter'));
}
```

The notifyServer() function sends a POST request to the /count endpoint with a message. Make sure to call it inside both the handleDecrement() and handleIncrement() functions. Recompile the assets and refresh the page. Now, before pressing the increment or decrement buttons, open the network tab in your browser. Now press the increment or the decrement button, and a POST request will show up as follows:




As you can see, the Laravel React js server has received the request and responded with the message. You can send any request to the server this way. The Axios library also handles the CSRF token by sending the XSRF-TOKEN cookie with every request, so you should be able to connect to any of the web routes.

You can also use API routes here, but then, you'll have to implement a stateless authentication system. In my opinion, if you're using React components within blade templates, work with web routes. It'll give you much less hassle and allow you to do almost anything that the API routes allow you to do.

# Vendor Extraction

The last topic that I would like to discuss in this article is vendor extraction. One downside of bundling up your application-centric JavaScript code and the React library together is that it makes long-term caching difficult. A single change in your code will force the browser to re-download the vendor library i.e. React even though they haven't changed.

One way to solve this issue is by extracting the vendor library code in a separate file. To do so, open the **webpack.mix.js** file and update its code as follows:

```
// webpack.mix.js

const mix = require('laravel-mix');

mix.js('resources/js/app.js', 'public/js')
    .react()
    .extract(['react'])
    .postCss('resources/css/app.css', 'public/css', [
        //
    ]);
```

adeva

- public/js/manifest.js: The Webpack manifest runtime
- public/js/vendor.js: Your vendor libraries
- public/js/app.js: Your application code

Open the **resources/views/welcome.blade.php** file and update its content as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Laravel React</title>
</head>
<body>
    <div id="hello-react"></div>
    <div id="counter"></div>

    <script src="{{ asset('js/manifest.js') }}"></script>
    <script src="{{ asset('js/vendor.js') }}"></script>
    <script src="{{ asset('js/app.js') }}"></script>
</body>
</html>
```

Instead of deferring code execution, I've moved the scripts to the bottom of the file. Also, make sure to load the files in this exact order. If there are multiple application-centric JavaScript files, they should come after the vendor script. The manifest will always be the first one. Reload your browser, and you should see no change in results whatsoever.

# Conclusion

stuff from this React js Laravel article. If you're confused about any part of this article, feel free to browse the reference repository. I would also suggest you read through the official Laravel docs on Compiling Assets (Mix) to learn about concepts such as cache busting, environment variables, and more. As you can see, using Laravel and React can be highly beneficial. And don't forget, it's not about Laravel vs React, but much more about how to use them together.

If you have any questions, feel free to reach out to me. I'm available on Twitter and LinkedIn and always happy to help. Till the next one, stay safe and keep on learning about React js and Laravel.

## FAQs

### Q: Can I use React JS with Laravel?

Laravel doesn't require you to use any specific Javascript framework. As long as you've configured Laravel Mix properly, any framework should work.

### Q: What is React and Laravel?

### Q: How does Reactjs integrate with Laravel?

**We're hiring!**

If you are passionate about Laravel, come and join us at Adeva! We are always looking for new talent to join our network.

**Farhan Hasin Chowdhury**

adeva

open-source mindset and loves sharing his knowledge with the community.

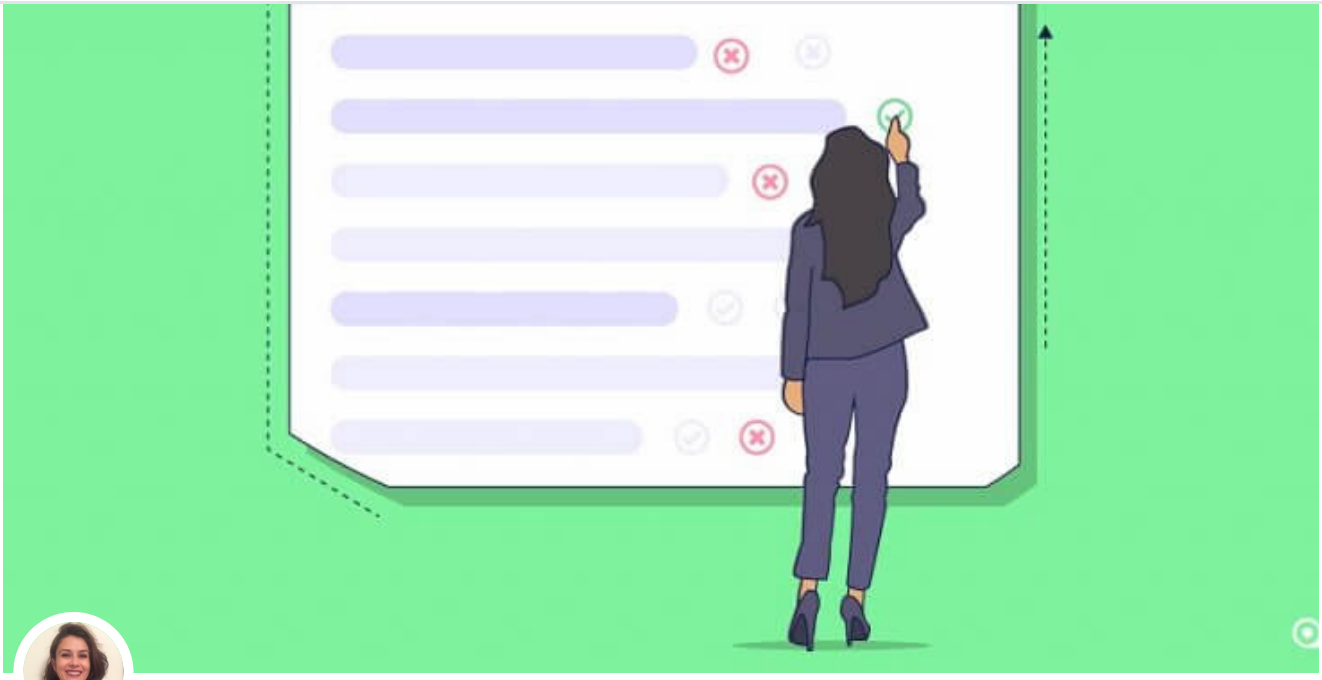
**EXPERTISE**

LARAVEL MYSQL VUE.JS NODE.JS AWS +16



## Featured Laravel Articles

adeva





WORKPLACE

## Leaving the Comfort Zone: Salary Negotiation Tips for Women in Tech

by **Sandra Petrova**

In January 2018, the world turned its ears to the controversy over the pay gap between Mark Wahlberg and Michelle Williams fo...

8 min read

adeva

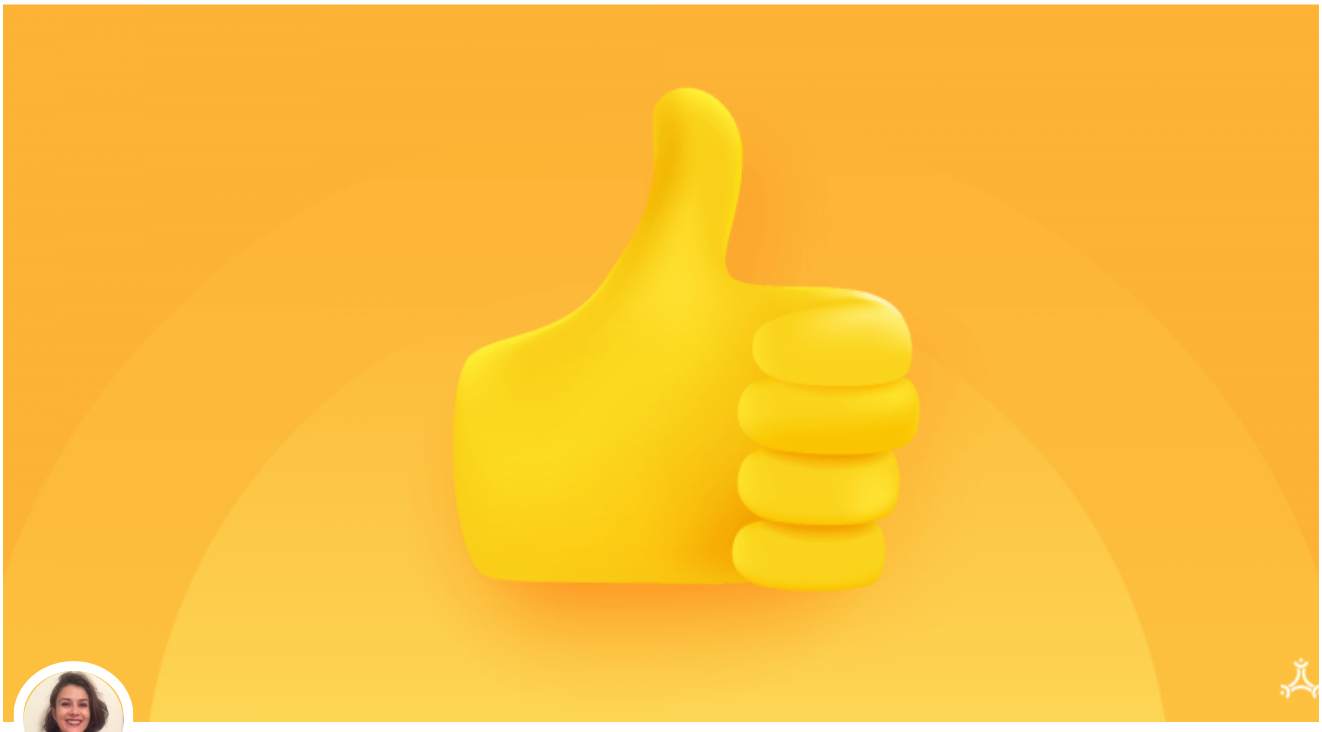
WORKPLACE

## 5 Tools That Will Help You Assess Your Engineers [2023]

by **Sandra Petrova**

If you want to build a successful technology startup, you'll need more than just a great idea. You'll also need a great engin...

9 min read



adeva

## Startup

by **Sandra Petrova**

Motivation is a key factor for successful remote teams. Learn the best practices for how to motivate remote employees and bui...

7 min read

adeva

**Front End** >

AdonisJS Developers

AngularJS Developers

JavaScript Developers

ReactJS Developers

Vue.js Developers

**Mobile** >

Android Developers

Flutter Developers

Huawei Developers

Ionic Developers

iOS Developers

React Native Developers

**Platform**

DevOps Engineers

Integration Developers

Magento Developers

Prismic.io Developers

Salesforce Developers

Shopify Developers

Wordpress Developers

adeva

Functional Testing

Manual Testing

Regression Testing

Usability Testing

> **work without boundaries**

**Solutions**

For Startups

For Enterprises

**Community**

About

Careers

Contact

FAQs

Technical Writers Program

Media Kit

**Insights**

adeva

Future of Work

Startups

**Resources**

Guides

Interview Questions



Privacy Policy

Website Terms