How to Serve a React Single-Page App with Django

#django #react #webpack #babel

Zach Taylor 21 set 2020 · 7 min read

TL;DR

You can download the finished code from my GitHub repository. Leave a star if you found it helpful!

Intro

This is a guide on setting up Django to serve a React single-page application. Going through this process really helped me understand Webpack and Babel better, so if Django + React isn't your stack, you might still learn something!

All the commands and file paths you'll see are relative to the project root unless otherwise specified. If you don't have a project already, you can create one with

```
$ pip install Django
$ django-admin startproject django_react_starter
$ python manage.py migrate
```

Let's get to it.

Step 1 - Create a Front End App

The first thing you'll want to do is create a Django app for your front end. I called mine `frontend`.

```
$ python manage.py startapp frontend
```

Add your app to `INSTALLED_APPS` in your project's `settings.py` file.

```
INSTALLED_APPS = [
  'frontend',
  ...
```

Step 2 - Create the View

Now that your `frontend` app is created, you need to create the Django view that will serve the React app.

In your `frontend` folder, create a folder called `templates`, and inside that, create a folder called `frontend`. In `frontend/templates/frontend/` create an `index.html` file and put the following inside it.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Site</title>
</head>
<body>
  <div id="app"></div>
</body>
</html>
```

Pretty simple. This HTML file is the single page in your single-page application. The `<div id="app"></div>` is where you will render your React app.

Next, you need to wire up a view to your index page. In `frontend/views.py` add the following.

```
from django.shortcuts import render

def index(request):
  return render(request, 'frontend/index.html')
```

All this function does is render the `index.html` page you just created.

Now you need to tell Django the url at which it will find your `index.html` page. In your **project level** `urls.py`, add the following to the bottom of your `urlpatterns`.

```
from django.urls import include, path

urlpatterns = [
  ...,
  path('', include('frontend.urls'))
]
```

In your `frontend` folder, create a `urls.py` file and put the following in it.

```
from django.urls import path
```

```
urlpatterns = [
  path('', views.index)
]
```

These two `urls.py` files tell Django to call your `index` view when someone visits the url `/`. Try running the server with

```
$ python manage.py runserver
```

Go to `localhost:8000` in your browser and you should see a blank page with `My Site` on the tab.

Great! Now let's add React to your HTML page.

Step 3 - Set up React, Babel, and Webpack

From the root of your project, run `npm init -y` to create a `package.json` file. You'll need several packages for this setup. The first two are React itself and ReactDom.

```
$ npm install react react-dom
```

Once you have React and ReactDom installed, you'll need to get Babel and Webpack set up.

Babel

Let's start with Babel. To install Babel, run

```
$ npm install --save-dev @babel/core
```

If you don't already know, Babel is a JavaScript transpiler, which essentially means it lets you use things in your JavaScript code (like JSX) that the browser wouldn't understand natively.

By default, Babel does nothing. If you want Babel to transpile a specific thing in your JavaScript code, you need to install a plugin for it. Your project might need several plugins, so Babel also has this concept of *presets*, which are just collections of plugins. You will only need two presets for this setup: `@babel/preset-env` and `@babel/preset-react`.

```
$ npm install --save-dev @babel/preset-env @babel/preset-react
```

`@babel/preset-env` is a collection of plugins that allows you to use the latest JavaScript features even if your browser doesn't support them yet. `@babel/preset-react` is a

Once you install the presets, you need to tell Babel to use them. Create a `.babelrc` file in the root of your project with the following content.

```
{
  "presets": ["@babel/preset-env", "@babel/preset-react"]
}
```

Webpack

Webpack is a tool that will take your codebase and all its dependencies and transform them into one or more *bundles*, or files, that can be executed in a browser. The way it works is pretty simple, in concept. You give Webpack a JavaScript file (the entry point), and it will recursively gather all the dependencies of that file (indicated with `import` or `require` statements) and combine them into one, larger, file.

If you're not used to JavaScript, it might not make sense why Webpack is needed. Historically, there was no way to `import` or `require` resources in JavaScript running in the browser. You either had to put all your JavaScript into one file or put it in several files along with a `<script>` tag for each in your HTML. That's fine if your web site doesn't have much JavaScript, but it quickly becomes messy and hard to maintain as the amount of JavaScript you have grows. Webpack allows you to separate your JavaScript code into reusable files and `import` or `require` what you need.

And Webpack isn't just for JavaScript. It also allows you to import JSON by default as well, and it can be configured to allow imports from `.css`, `.sass`, `.hbs` and more with *loaders*.

For this Webpack setup, you'll need several packages.

```
webpack-bundle-tracker@0.4.3 babel-loader css-loader style-loader clean-webpack-plugin
```

That's quite a few! Let's break it down:

- `webpack` is... well, Webpack
- `webpack-cli` allows you to run Webpack commands from the command line
- `webpack-bundle-tracker` is a plugin that writes some stats about the bundle(s) to a JSON file.
- `babel-loader` is a loader that tells Webpack to run Babel on the file before adding it to the bundle.
- `css-loader` and `style-loader` are loaders that allow you to import `.css` files into

- `clean-webpack-plugin` is a plugin that deletes old bundles from Webpack's output directory every time a new bundle is created.

Now create a file called `webpack.config.js` in the root of your project. This is where you'll configure Webpack to use the plugins and loaders we just installed.

```
const path = require('path')
const BundleTracker = require('webpack-bundle-tracker')
const { CleanWebpackPlugin } = require('clean-webpack-plugin')

module.exports = {
  entry: {
    frontend: './frontend/src/index.js',
  },
  output: {
    path: path.resolve('./frontend/static/frontend/'),
    filename: '[name]-[hash].js',
  },
  plugins: [
    new CleanWebpackPlugin(),
    new BundleTracker({
      path: __dirname,
      filename: './webpack-stats.json',
    }),
  ],
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: ['babel-loader']
        },
      },
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader'],
      },
    ],
  },
}
```

Let's break it down:

- `entry` tells Webpack where to start gathering your code
- `output` is where Webpack will put the finished bundle.
- `plugins` tells Webpack which plugins to use

- `module` is where you configure your loaders. Each rule tells Webpack that whenever it comes across a file that matches the `test` regex, it should use the specified loaders to process it.

Now that Webpack is set up, you'll want to add a couple scripts to your `package.json` to run Webpack.

```
{
  ...,
  "scripts": {
    ...,
    "dev": "webpack --config webpack.config.js --watch --mode development",
    "build": "webpack --config webpack.config.js --mode production"
  }
}
```

These scripts allow you to create a development bundle with `npm run dev` and a production bundle with `npm run build`.

Step 4 - Add the Bundle to your HTML

Now that you have a process to create a JavaScript bundle, you need to include the bundle in your HTML page. To do that, you'll need to install one more package.

```
$ pip install django-webpack-loader
```

This package allows Django to use the stats produced by `webpack-bundle-tracker` to load the correct bundle in your HTML page. In your `settings.py` file, add the following configuration.

```
import os

...

INSTALLED_APPS = [
  'webpack_loader',
  ...
]

...

WEBPACK_LOADER = {
  'DEFAULT': {
    'BUNDLE_DIR_NAME': 'frontend/',
    'STATS_FILE': os.path.join(BASE_DIR, 'webpack-stats.json')
```

Then in your `frontend/templates/frontend/index.html` file, add a template tag to load the bundle into your page.

```
<!DOCTYPE html>
+ {% load render_bundle from webpack_loader %}
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Site</title>
</head>
<body>
  <div id="app"></div>
+ {% render_bundle 'frontend' %}
</body>
</html>
```

Step 5 - Create Your React App

We now have all the pieces in place for you to begin writing your React application! In your `frontend` folder, create a folder called `src`, and inside that, create a file called `App.js` with the following content.

```
import React from 'react'

const App = () => {
  return (
    <div>Hello, World!</div>
  )
}

export default App
```

In your `frontend/src` folder, create another file called `index.js` with the following.

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'

ReactDOM.render(
  <App />,
  document.getElementById('app')
)
```

In the terminal navigate to your project and run

In another terminal window or tab, navigate to your project and run

```
$ python manage.py runserver
```

The order you run these two commands is important. Make sure you do `npm run dev` first.

Navigate to `localhost:8000` in your browser and you should see `Hello, World!` printed on the screen. Awesome! You've successfully set up Django to serve a React single-page application. You can view or download the finished code on my GitHub repository.

Going through the process of setting this up was so helpful to me in understanding Webpack and Babel. I hope you found it enlightening as well!

Discussion

Subscribe

DEV

Add to the discussion

Zach Taylor

I'm passionate about a lot of things. Code is one of my favorites.

Follow

WORK
Web Developer at Chicago Venture Partners

LOCATION
Chicago, IL

JOINED
17 ago 2020

More from Zach Taylor

Model Inheritance in Django