How to create a React TypeScript application from scratch

REACT TYPESCRIPT

November 30th, 2020

Create a new project directory and navigate there:

```
mkdir my-app-name
cd my-app-name
```

Init this directory as an `npm` -package. It will allow you to install third-party packages such as React and TypeScript later:

```
npm init -y
#or
yarn init
```

Install React, React-DOM and TypeScript:

```
npm install --save typescript react react-dom
# or
yarn add typescript react react-dom
```

Install type definitions for React and React-DOM. They will allow TypeScript compiler to work with packages and figure out the types that can and cannot be used with

```
npm install --save @types/react @types/react-dom
# or
yarn add @types/react @types/react-dom
```

If you use some other packages you will need to install types for those packages too. The `DefinitelyTyped` repository contains the biggest amount of type declarations for npm packages. You can find the types for your package there and install them.

TypeScript Config

Generate a `tsconfig.json` file using `npx` package runner:

```
npx tsc --init
```

This file is used as settings for TypeScript compiler and tells it how to interpret the code.

After it's done, open this file and add the following option:

```
"jsx": "react"
```

This will allow a compiler to work with JSX components as React components.

Setting up Webpack

Webpack is a static module bundler for modern JavaScript applications. It takes care of compiling, bundling and minifying the code for you.

Install `webpack`, its command line interface, and `html-webpack-plugin`:

```
npm install webpack webpack-cli html-webpack-plugin ts-loader --save-dev
# or
yarn add webpack webpack-cli html-webpack-plugin ts-loader --dev
```

The latter is used to deal with the `html` -entrypoint so that we will be able to run our application in a browser.

The `ts-loader` package is a loader for webpack that knows how to work with TypeScript files. Webpack itself doesn't know that.

Then, create a `webpack.config.js` file.

```
const path = require("path");
const HtmlWebPackPlugin = require("html-webpack-plugin");

// Instantiate the plugin.
// The `template` property defines the source
```

```
// of a template file that this plugin
will use.
// We will create it later.
const htmlPlugin = new HtmlWebPackPlugin({
  template: "./src/index.html",
});

module.exports = {
  // Our application entry point.
  entry: "./src/index.tsx",

  // These rules define how to deal
  // with files with given extensions.
  // For example, .tsx files
  // will be compiled with ts-loader,
  // a spcific loader for webpack
  // that knows how to work with
TypeScript files.
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        use: "ts-loader",
        exclude: /node_modules/,
      },
    ],
  },

  // Telling webpack which extensions
  // we are interested in.
  resolve: {
    extensions: [".tsx", ".ts", ".js"],
  },

```

```
  // What file name should be used for the result file,
  // and where it should be palced.
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist"),
  },

  // Use the html plugin.
  plugins: [htmlPlugin],
};
```

Source code

Create an entry point file `src/index.tsx`:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./components/App";

ReactDOM.render(<App />,
document.getElementById("root"));
```

Create a first component `src/components/App/index.tsx`:

```
import React from "react";

const App: React.FC = () => <div>Hello worl
</div>;

export default App;
```

Add a template `src/index.html` file:

```
<!DOCTYPE html>
<html>
  <head>
    <title>App</title>
  </head>

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

Development Server

Install the webpack-dev-server package:

```
npm install webpack-dev-server --save-dev
# or
yarn add webpack-dev-server --dev
```

This package is used to run application while developing. Also, it will reload the application when source code is changed.

Set up development server in your `webpack.config.js` :

```
{

  // ...Previous config

  // Set up the directory
  // from which webpack will take the
static content.
  // The port field defines which port on
localhost
```

```
  // this application will take.
  devServer: {
    contentBase: path.join(__dirname,
'dist'),
    compress: true,
    port: 9000
  }
}
```

Runner Scripts

In your `package.json` file, update `scripts` section with these scripts:

```
{
  "scripts": {
    "dev": "webpack serve",
    "build": "webpack --mode production"
  }
}
```

The `build` script will build your application in a `production` mode which creates an optimized production build.

And the `dev` script will set up a development server on `localhost` for you to develop the application.

If you now run `npm run dev` and open `http://localhost:9000/` you will see a running app.

Sources

- Type Declarations

- `npx` package runner
- `tsconfig.json` file
- `DefinitelyTyped` repository
- `webpack`
- `webpack-dev-server`
- `ts-loader` for webpack

Alex Bespoyasov

MORE BY ALEX BESPOYASOV

How to Test Your First React Hook Using Testing Library

How to Write Your First Component Test in React + TypeScript App

How to Write Your First Unit Test in React + TypeScript App

Fullstack React with TypeScript is a step-by-step guide to patterns and best-practices with React with TypeScript

Download the first chapter

You Might Also Like

Beginner's Guide to Real-World React Coming Soon

May 9th, 2021

Deploying a Node.js and PostgreSQL Application to Heroku

May 5th, 2021

React Query Builder - The Ultimate Querying Interface

May 3rd, 2021

Learn

Fullstack React

ng-book

Fullstack D3

Fullstack Node.js

Fullstack React Native

Fullstack Rust

Requests

Request a Course

Request a Site Feature

Community

Student stories

Community Discord

Masterclasses

Tinyhouse: Fullstack React Masterclass

Fullstack Flask: Build a SaaS Masterclass

Shows

newline Podcast

Tutorials

React Tutorials

Node.js Tutorials

Rust Tutorials

Latest Book