https://alligator.io/ Script CSS React Vue.js Angular Gatsby

Fri Nov 22 2019 16:41:21 GMT+0700 (Indochina Time)

Front-End Web Development, Chewed Up

Angular 2+, Vue.js, React, Svelte
JavaScript, CSS...

Top Posts

- Using TypeScript with React
- Building a Modal Component with Vue.js
- Using Axios with React
- Exploring Async/Await Functions in JavaScript
- Cropping Images in CSS With object-fit
- Pure-CSS Parallax Scrolling Effect

Subscribe & stay in the loop:

your email

Subscribe

React

- Countdown Timer with React Hooks
- Manage User Login with React Context
- Creating Complex Animations in React Using react-spring
- react-notifications-component, a Powerful React Notifications Library
- Intro to Animations in React Using React Spring
- A Guide for Refs in React

+

JavaScript

- Exploring the JavaScript forEach Method for Looping Over Arrays
- Building a Countdown Timer in JavaScript
- Using toLocaleString with Numbers, Arrays or Dates in JavaScript
- Finally Understand the JavaScript Reduce Method
- Creating a Custom webpack Plugin
- The V8 Engine and JavaScript Optimization Tips

+

CSS

- Comparing CSS Pseudo-Classes: nth-child vs nth-of-type
- CSS Hex Code Colors with Alpha Values
- Understanding Specificity in CSS
- Change Text Case with CSS's text-transform Property
- CSS will-change Property: When and When Not to Use It
- Using Google Fonts in Your Web Pages

+

Angular 2+

- Building Maps in Angular using Leaflet, Part 4: The Shape
- Building Maps in Angular using Leaflet, Part 3: The Popup
- Building Maps in Angular using Leaflet, Part 2: The Marker

Service

Service

Service

Building Maps in Angular Using Leaflet, Part 1: Generating Maps

Using Angular CLI Schematics

Infinite Scroll in Angular Using the CDK

+

Vue.js

Renderless Behavioral Slots in Vue.js

Using TypeScript with Vue Single File Components

How to Create an Image Slider with Vue.js

What's New in vue-styleguidist 3.0

Testing Vue with Jest

How to use Vue.js with Electron and Vuex

+

Workflow

Introduction to Docker Containers

Introduction to Docker: Getting Started

Command-line Basics: How to Loop Through Files in a Directory

Command-line Basics: Dressing up Output with cowsay

Command-line Basics: Finding and Replacing Text in Files with sed

Command-line Basics: Resizing Images with ImageMagick

+

Gatsby

Custom Fonts in Gatsby

Automating a Gatsby Blog with GitHub and Buddy

sponsored

Using Plugins in Gatsby

Intro to MDX in Gatsby

Gatsby, Contentful, and Netlify Walk Into a Bar

Gatsby CLI Quick Reference

+

Node.js

Using Server-Sent Events in Node.js to Build a Realtime App

npm Commands and Features You Should Know

Avoid Invalid Requests to Your Express.js Server Using celebrate

Adding and Removing Packages Using npm or Yarn

Getting Started with Compression in Node.js

Understanding Express Template Engines

+

Catch up with the latest in front-end

#REACT

Using TypeScript with React

Paul Halliday

> TypeScript is awesome. So is React. Let's use them both together! Using TypeScript allows us to get the benefits of IntelliSense, as well as the ability to further reason about our code. As well as this, adopting TypeScript is easy as files can be incrementally upgraded without causing issues throughout the rest of your project.

Let's start off by creating a new React project and integrate TypeScript. Run the following commands to initiate the project:

```
# Make a new directory
$ mkdir react-typescript

# Change to this directory within the terminal
$ cd react-typescript

# Initialise a new npm project with defaults
$ npm init -y

# Install React dependencies
$ npm install react react-dom

# Make index.html and App.tsx in src folder
$ mkdir src
$ cd src
$ touch index.html
$ touch App.tsx

# Open the directory in your favorite editor
$ code .
```


more react

Countdown Timer with React Hooks

Manage User Login with React Context

Creating Complex Animations in React Using react-spring

react-notifications-component, a Powerful React Notifications Library

all react posts

follow us @alligatorio

We can then make an `index.html` file with the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <title>React + TypeScript</title>
  <meta name="viewport" content="width=device-width, initial-scal
</head>
<body>
  <div id="main"></div>
  <script src="./App.tsx"></script>
</body>
</html>
```

We'll be using Parcel as our bundler, but you can elect to use webpack or another bundler if you wish. Or, check this section if you prefer using Create React App. Let's add Parcel to our project:

```
# Install Parcel to our DevDependencies
$ npm i parcel-bundler -D

# Install TypeScript
$ npm i typescript -D

# Install types for React and ReactDOM
$ npm i -D @types/react @types/react-dom
```

We can update our `package.json` with a new task that will start our development server:

```
{
  "name": "react-typescript",
  "version": "1.0.0",
  "description": "An example of how to use React and TypeScript w
  "scripts": {
    "dev": "parcel src/index.html"
  },
  "keywords": [],
  "author": "Paul Halliday",
  "license": "MIT"
}
```

We can now populate a `Counter.tsx` file with a simple counter:

```
import * as React from 'react';

export default class Counter extends React.Component {
  state = {
    count: 0
  };

  increment = () => {
    this.setState({
      count: (this.state.count + 1)
    });
  };

  decrement = () => {
    this.setState({
      count: (this.state.count - 1)
    });
  };

  render () {
    return (
      <div>
        <h1>{this.state.count}</h1>
        <button onClick={this.increment}>Increment</button>
        <button onClick={this.decrement}>Decrement</button>
      </div>
    );
  }
}
```

Then, inside of `App.tsx`, we can load the Counter:

```
import * as React from 'react';
import { render } from 'react-dom';
```

```
import Counter from './Counter';

render(<Counter />, document.getElementById('main'));
```

Our project can be ran with `$ npm run dev` and accessed at `http://localhost:1234`.

Create React App and TypeScript

If you'd rather use Create React App to initiate your project, you'll be pleased to know that CRA now supports TypeScript out of the box.

Just use the `--typescript` flag when invoking the `create-react-app` command:

```
$ create-react-app my-new-app --typescript
```

Functional Components

Stateless or functional components can be defined in TypeScript like so:

```
import * as React from 'react';

const Count: React.FunctionComponent<{
  count: number;
}> = (props) => {
  return <h1>{props.count}</h1>;
};

export default Count;
```

We're using `React.FunctionComponent` and defining the object structure of our expected props. In this scenario we're expecting to be passed in a single prop named `count` and we're defining it in-line. We can also define this in other ways, by creating an interface such as `Props`:

```
interface Props {
  count: number;
}

const Count: React.FunctionComponent<Props> = (props) => {
  return <h1>{props.count}</h1>;
};
```

Class Components

Class components can similarly be defined in TypeScript as such:

```
import * as React from 'react';

import Count from './Count';

interface Props {}

interface State {
  count: number;
};

export default class Counter extends React.Component<Props, State
  state: State = {
    count: 0
  };

  increment = () => {
    this.setState({
      count: (this.state.count + 1)
    });
  };

  decrement = () => {
    this.setState({
      count: (this.state.count - 1)
    });
  };

  render () {
    return (
      <div>
        <Count count={this.state.count} />
        <button onClick={this.increment}>Increment</button>
        <button onClick={this.decrement}>Decrement</button>
```

```
      </div>
    );
  }
}
```

Default Props

We can also define `defaultProps` in scenarios where we may want to set default props. We can update our Count example to show this:

```
import * as React from 'react';

interface Props {
  count?: number;
}

export default class Count extends React.Component<Props> {
  static defaultProps: Props = {
    count: 10
  };

  render () {
    return <h1>{this.props.count}</h1>;
  }
}
```

We'll need to stop passing `this.state.count` in to the `Counter` component too, as this will overwrite our default prop:

```
render () {
  return (
    <div>
      <Count />
      <button onClick={this.increment}>Increment</button>
      <button onClick={this.decrement}>Decrement</button>
    </div>
  )
}
```

You should now have a project that's set up to use TypeScript and React, as well as the tools to create your own functional and class-based components!

Published: February 11, 2019