

LAB # 10


Introductory to React

OBJECTIVE

Start off with React

THEORY

React vs Vue vs Angular

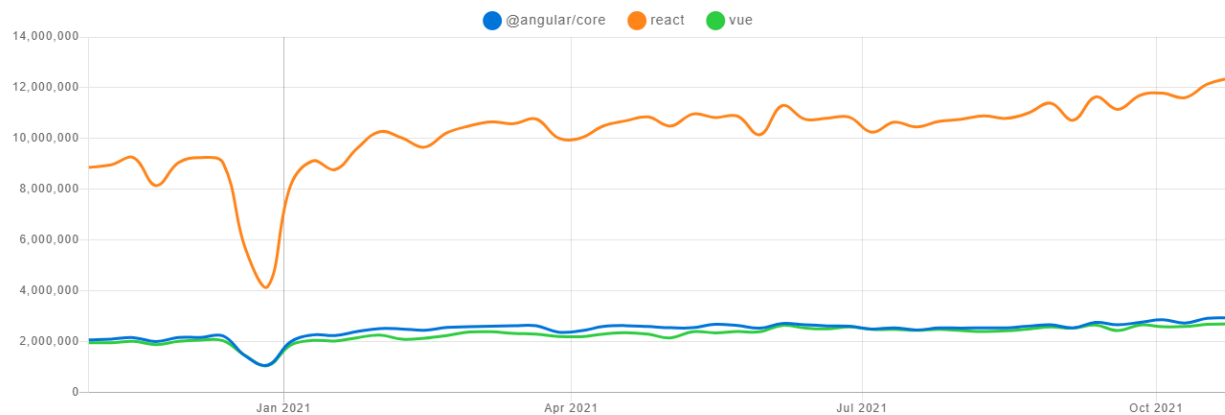
 npm trends

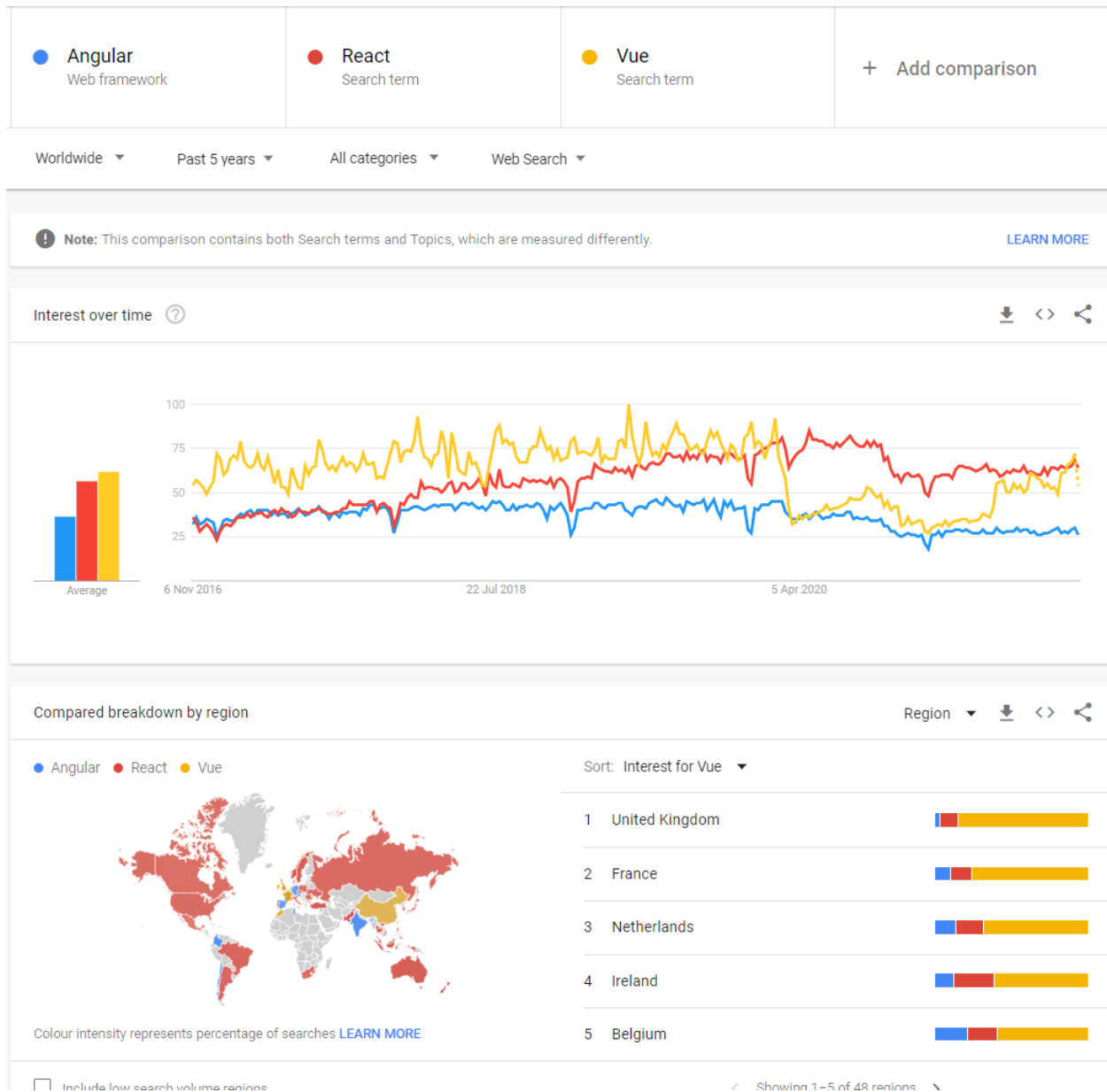
@angular/core vs react vs vue

Enter an npm package...

@angular/core x react x vue x + angular + ember-source

Downloads in past 1 Year v





<https://trends.google.com/trends/explore?date=today%205-y&q=%2F11c6w0ddw9,React,Vue>

Why React?

Let's start with React's official definition. It states that it is a JavaScript library for building user interfaces. It is important to understand the two different parts of this definition. React is a JavaScript library and not a framework. The words library and framework mean different things in different contexts, and this could be a point for or against React.

- What's important to remember here is that;
- React is small, and it's not a complete solution.

- You will need to use other libraries with React to form solutions.
- React does not assume anything about the other parts in any full solution.
- It focuses on just one thing, which is the second part of the definition, building user interfaces.

A user interface is anything we put in front of users to have them interact with a machine. User interfaces are everywhere, from the simple buttons on a microwave to the dashboard of a space shuttle. If the device we're trying to interface can understand JavaScript, we can use React to describe a user interface for it. Since web browsers understand JavaScript, we can use React to describe web user interfaces.

React is a small library that focuses on just one thing, enabling developers to declaratively describe their user interfaces and model the state of these interfaces. React basically gives developers the ability to work with a virtual browser that is friendlier than the real browser.

React's Basic Concepts

React has three simple and fundamental concepts that you need to understand.

The first concept is its components. With React, we describe user interfaces using components. In any programming language, we invoke functions with some input, and they give us some output in return. We can reuse functions as needed and compose bigger functions from smaller ones. React components are exactly the same. They receive certain input objects, and they output a description of a user interface. We can reuse a single component in multiple user interfaces, and components can contain other components. However, you don't really invoke a React component. You just use it in your HTML as if it was just a regular HTML element.

When the state of a React component, the input, changes the user interface it represents, the output, changes as well. This change in the description of the user interface has to be reflected in the device we are working with. In a browser, we need to regenerate the HTML views in the DOM tree. With React, we don't need to worry about how to reflect these changes or even manage when to take changes to the browser. React will simply react to the changes in a component's state and automatically update the parts of the DOM that need to be updated.

The third concept about React is its virtual representation of views in memory. Okay, this might sound a bit weird, but to build HTML web applications with React, we don't write HTML at all. We use JavaScript to generate HTML. When your web application receives just the data from the server in the background with AJAX, you need something more than HTML to work with that data, and you have two options. You can use an enhanced HTML template that has loops and conditionals, or you can rely on the power of JavaScript itself to generate the HTML from the data.

A React component can be one of two types. It can be either;

- a function component
- a class component

Both types can be stateful and have side effects, or they can be purely presentational. You should learn them both, but prefer to use function components over class components because they're really much simpler. Class components, however, are a bit more powerful. Both types can be compared to regular functions when it comes to their contract. They use a set of props and state as their input, and they output what looks like HTML, but is really a special JavaScript syntax called JSX.

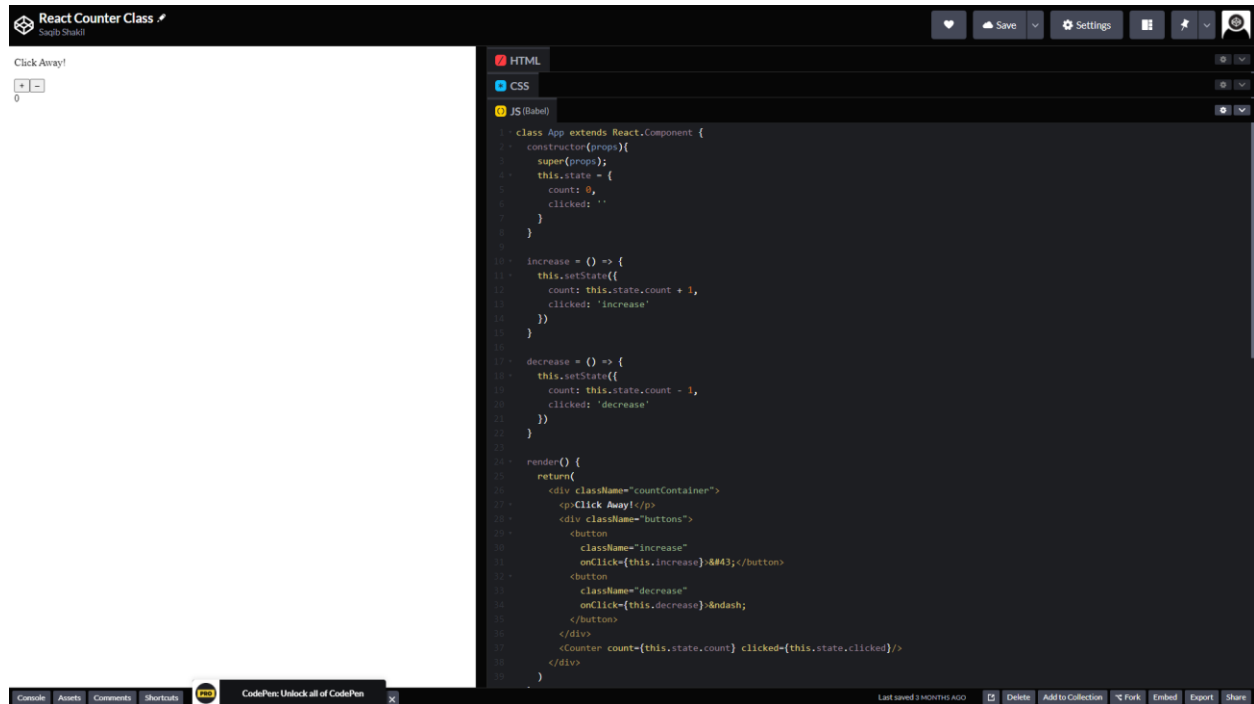
Create a Class component called Car

```
class Car extends React.Component {  
  render() {  
    return <h2>Hi, I am a Car!</h2>;  
  }  
}
```

Create a Function component called Car

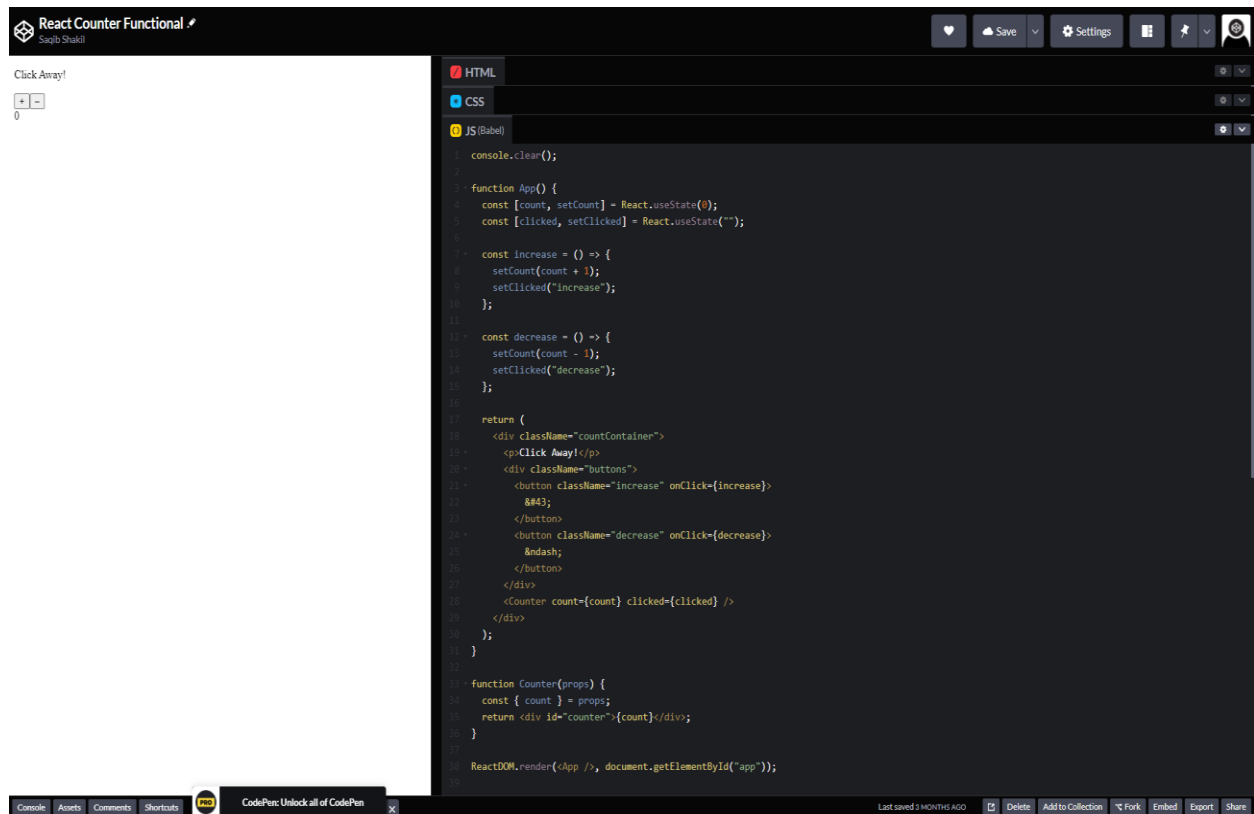
```
function Car() {  
  return <h2>Hi, I am a Car!</h2>;  
}
```

Sample Code



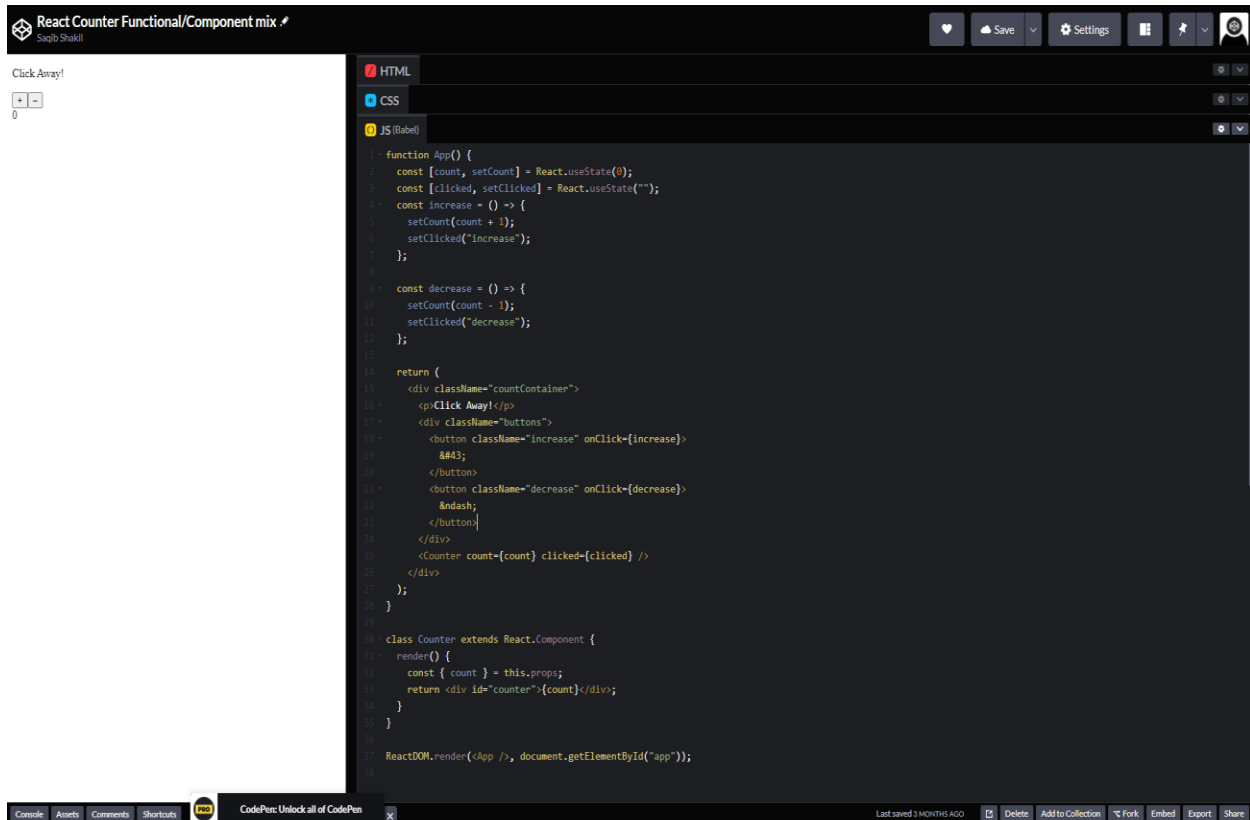
The screenshot shows a CodePen editor for a React Counter Class. The left sidebar displays the component's UI: a text input with "Click Away!", a counter showing "0", and two buttons labeled "+" and "-". The main editor area shows the JavaScript code for the class, which extends `React.Component`. It includes a constructor, `increase` and `decrease` methods, and a `render` method that returns the UI structure. The right sidebar shows the HTML, CSS, and JS (Babel) tabs, with the JS tab selected.

```
1 class App extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = {
5       count: 0,
6       clicked: ''
7     }
8   }
9
10  increase = () => {
11    this.setState({
12      count: this.state.count + 1,
13      clicked: 'increase'
14    })
15  }
16
17  decrease = () => {
18    this.setState({
19      count: this.state.count - 1,
20      clicked: 'decrease'
21    })
22  }
23
24  render() {
25    return (
26      <div className="countContainer">
27        <p>Click Away!</p>
28        <div className="buttons">
29          <button
30            className="increase"
31            onClick={this.increase}&#43;</button>
32          <button
33            className="decrease"
34            onClick={this.decrease}&#45;</button>
35        </div>
36        <Counter count={this.state.count} clicked={this.state.clicked}/>
37      </div>
38    )
39  }
```



The screenshot shows a CodePen editor for a React Counter Functional component. The left sidebar displays the component's UI: a text input with "Click Away!", a counter showing "0", and two buttons labeled "+" and "-". The main editor area shows the JavaScript code for the functional component, which uses `React.useState` for state management. It includes `increase` and `decrease` functions, and a `render` function that returns the UI structure. The right sidebar shows the HTML, CSS, and JS (Babel) tabs, with the JS tab selected.

```
1 console.clear();
2
3 function App() {
4   const [count, setCount] = React.useState(0);
5   const [clicked, setClicked] = React.useState("");
6
7   const increase = () => {
8     setCount(count + 1);
9     setClicked("increase");
10  };
11
12  const decrease = () => {
13    setCount(count - 1);
14    setClicked("decrease");
15  };
16
17  return (
18    <div className="countContainer">
19      <p>Click Away!</p>
20      <div className="buttons">
21        <button className="increase" onClick={increase}>
22          &#43;</button>
23        <button className="decrease" onClick={decrease}>
24          &#45;</button>
25      </div>
26      <Counter count={count} clicked={clicked} />
27    </div>
28  );
29 }
30
31 function Counter(props) {
32   const { count } = props;
33   return <div id="counter">{count}</div>;
34 }
35
36 ReactDOM.render(<App />, document.getElementById("app"));
```



```
1 function App() {
2   const [count, setCount] = React.useState(0);
3   const [clicked, setClicked] = React.useState("");
4   const increase = () => {
5     setCount(count + 1);
6     setClicked("increase");
7   };
8
9   const decrease = () => {
10    setCount(count - 1);
11    setClicked("decrease");
12  };
13
14  return (
15    <div className="countContainer">
16      <p>Click Away!</p>
17      <div className="buttons">
18        <button className="increase" onClick={increase}>
19          ++
20        </button>
21        <button className="decrease" onClick={decrease}>
22          --
23        </button>
24      </div>
25      <Counter count={count} clicked={clicked} />
26    </div>
27  );
28 }
29
30 class Counter extends React.Component {
31   render() {
32     const { count } = this.props;
33     return <div id="counter">{count}</div>;
34   }
35 }
36
37 ReactDOM.render(<App />, document.getElementById("app"));
```

Forkable Code at

<https://codepen.io/collection/ZMYKNa?cursor=ZD0wJm89MCZwPTEmdj00>

Lab Task

- 1 Create a Shell Component
- 2 Create a Student Listing Component
 - a. Load the Students Listing Component from the Shell
 - b. Fetch list of student on Students Listing Mount
 - c. Show the list of students in a Table

Home Task

- 1 Continue to develop student Detail page and open it on half of the screen
 - a. On Click of the table row send the id of the student to the student profile Component
 - b. On Change of the id in student profile load the student from the mock api
 - c. Show details in the side panel