I created the exact same app in React and Vue. Here are the differences.

Sunil Sandhu Follow
Jul 24 · 12 min read

Having used Vue at work, I had a fairly solid understanding of it. I was, however, curious to know what the grass was like on the other side of the fence—the grass in this scenario being React.

I'd read the React docs and watched a few tutorial videos and, while they were great and all, what I really wanted to know was how different React was from Vue. By "different", I didn't mean things such as whether they both had virtual DOMS or how they went about rendering pages. I wanted someone to take the time to explain the code and to tell me what was going on! I wanted to find an article that took the time to explain these differences so that someone new to either Vue or React (or Web Development as a whole) could gain a better understanding of the differences between the two.

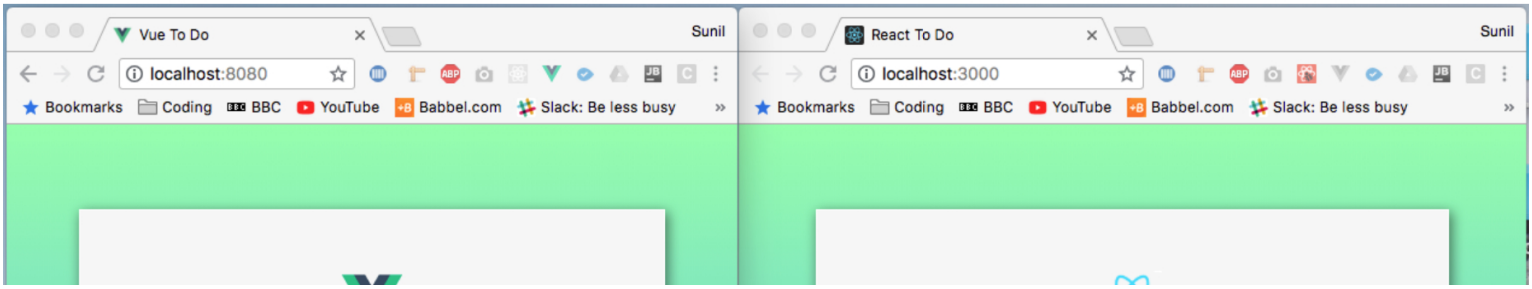
But I couldn't find anything that tackled this. So I came to the realisation that I'd have to go ahead and build this myself in order to see the similarities and differences. In doing so, I thought I'd document the whole process so that an article on this will finally exist.

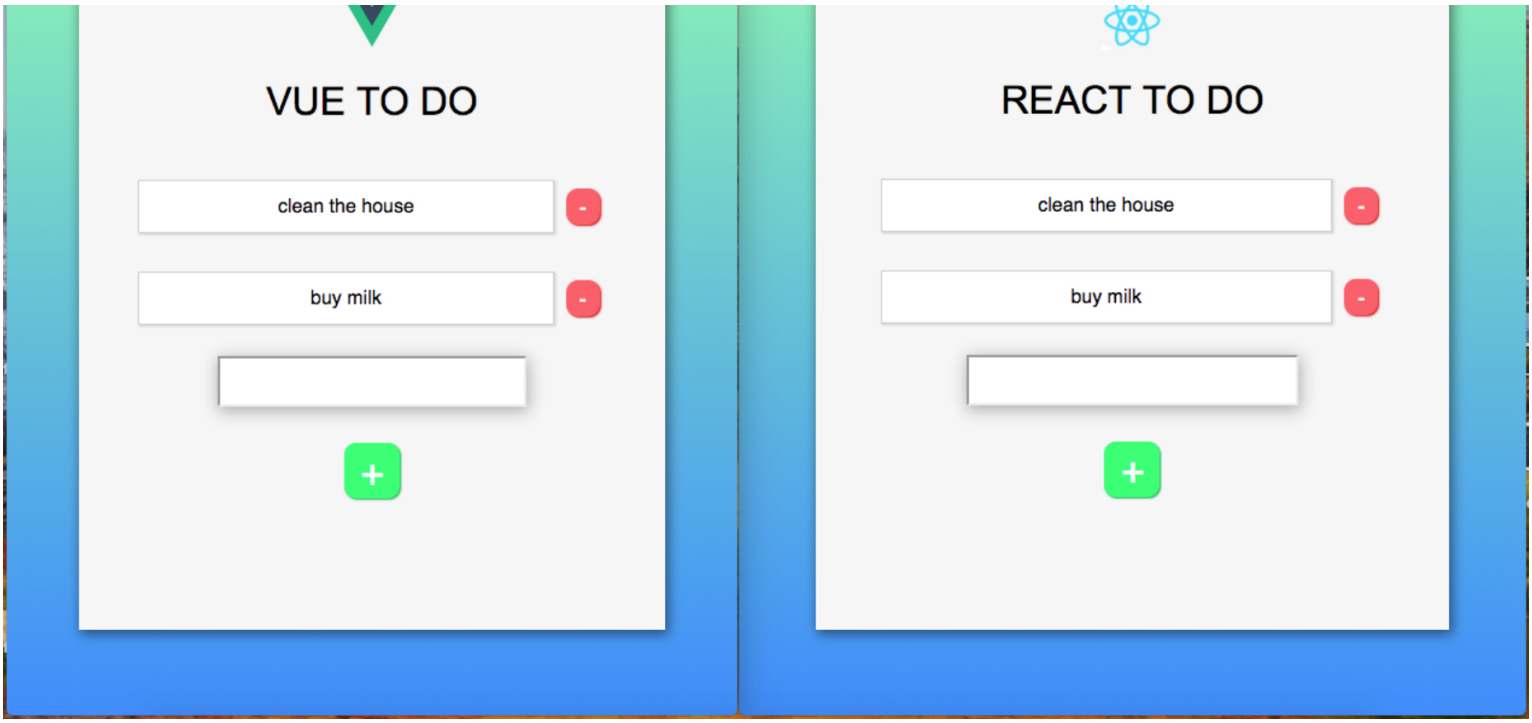
Who wore it better?

I decided to try and build a fairly standard To Do App that allows a user to add and delete items from the list. Both apps were built using the default CLIs (create-react-app for React, and vue-cli for Vue). CLI stands for Command Line Interface by the way. 🤓

Top highlight

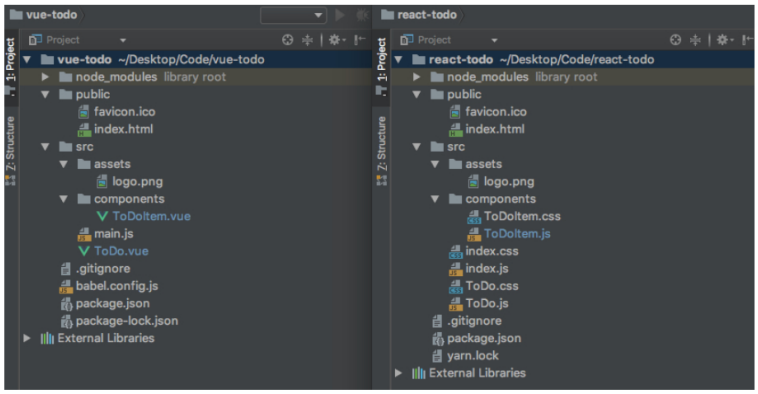
Anyway, this intro is already longer than I'd anticipated. So let's start by having a quick look at how the two apps look:





Vue vs React: The Irresistible Force meets The Immovable Object

The CSS code for both apps are exactly the same, but there are differences in where these are located. With that in mind, let's next have a look at the file structure of both apps:

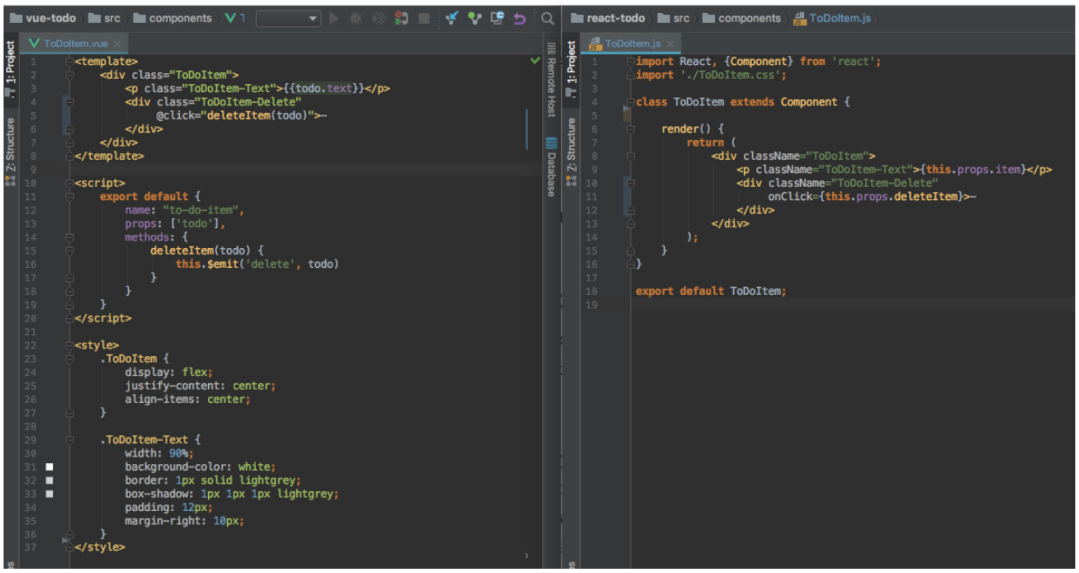


Who wore it better?

You'll see that their structures are almost identical as well. The only difference here is that the React app has three CSS files, whereas the Vue app doesn't have any. The reason for this is because, in create-react-app, a React component will have an accompanying file to hold its styles, whereas Vue CLI adopts an all encompassing approach, where the styles are declared inside the actual component file.

Ultimately, they both achieve the same thing, and there is nothing to say that you can't go ahead and structure your CSS differently in React or Vue. It really comes down to personal preference - you'll hear plenty of discussion from the dev community over how CSS should be structured. For now, we'll just follow the structure laid out in both CLIs.

But before we go any further, let's take a quick look at what a typical Vue and React component look like:



Vue on the left. React on the right

Now that's out of the way, let's get into the nitty gritty detail!

How do we mutate data?

But first, what do we even mean by "mutate data"? Sounds a bit technical doesn't it? It basically just means changing the data that we have stored. So if we wanted to change the value of a person's name from John to Mark, we would be 'mutating the data'. So this is where a key difference between React and Vue lies. While Vue essentially creates a data object, where data can freely be updated, React creates a state object, where a little more legwork is required to carry out updates. Now React implements the extra legwork with good reason, and we'll get into that in a little bit. But first, let's take a look at the **data** object from Vue and the **state** object from React:

Vue data object on the left. React state object on the right.

So you can see that we have passed the same data into both, but they're simply labelled differently. So passing initial data into our components is very, very similar. But as we've mentioned, how we go about changing this data differs between both frameworks.

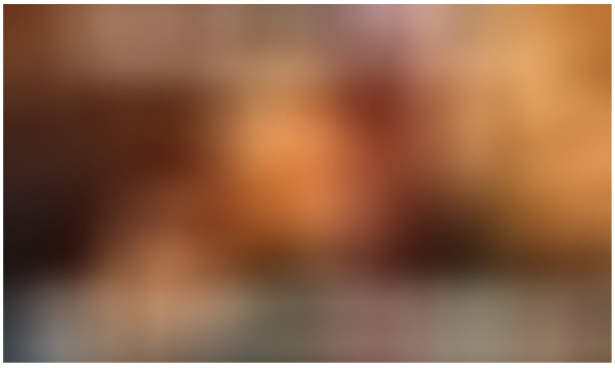
Let's say that we have a data element called **name: 'Sunil'**.

In Vue, we reference this by calling **this.name**. We can also go about updating this by calling **this.name = 'John'**. This would change my name to John. I'm not sure how I feel about being called John, but hey ho, things happen! 😅

In React, we would reference the same piece of data by calling **this.state.name**. Now the key difference here is that we cannot simply write **this.state.name** = 'John', because React has restrictions in place to prevent this kind of easy, care-free mutation-making. So in React, we would write something along the lines of **this.setState({name: 'John'})**.

While this essentially does the same thing as we achieved in Vue, the extra bit of writing is there because Vue essentially combines its own version of setState by default whenever a piece of data gets updated. So in short, React requires setState and then the updated data inside of it, whereas Vue makes an assumption that you'd want to do this if you were updating values inside the data object. So Why does React even bother with this, and why is setState even needed? Let's hand this over to Revanth Kumar for an explanation:

> *"This is because React wants to re-run certain life cycle hooks, [such as] componentWillReceiveProps, shouldComponentUpdate, componentWillUpdate, render, componentDidUpdate, whenever state changes. It would know that the state has changed when you call the setState function. If you directly mutated state, React would have to do a lot more work to keep track of changes and what lifecycle hooks to run etc. So to make it simple React uses setState."*



Bean knew best

Now that we have mutations out of the way, let's get into the nitty, gritty by looking at how we would go about adding new items to both of our To Do Apps.

How do we create new To Do Items?

React:

```
createNewToDoItem = () => {
    this.setState( ({ list, todo }) => ({
      list: [
          ...list,
        {
          todo
        }
      ],
      todo: ''
    })
  );
};
```

How did React do that?

In React, our input field has an attribute on it called **value.** This value gets

automatically updated through the use of a couple of functions that tie together to create something that closely resembles **two-way binding** (if you've never heard of this before, there's a more detailed explanation in the *How did Vue do that* section after this). We create this form of two-way binding by having an additional **onChange event listener** attached to the **input** field. Let's quickly take a look at the **input** field so that you can see what is going on:

```
<input type="text"
       value={this.state.todo}
       onChange={this.handleInput}/>
```

The handleInput function is run whenever the value of the input field changes. It updates the **todo** that sits inside the state object by setting it to whatever is in the input field. This function looks as such:

```
handleInput = e => {
  this.setState({
    todo: e.target.value
  });
};
```

Now, whenever a user presses the + button on the page to add a new item, the **createNewToDoItem** function essentially runs this.setState and passes it a function. This function takes two parameters, the first being the entire **list** array from the state object, the second being the **todo** (which gets updated by the **handleInput** function). The function then returns a new object, which contains the entire **list** from before and then adds **todo** at the end of it. The entire list is added through the use of a spread operator (Google this if you've not seen this before—it's ES6 syntax).

Finally, we set **todo** to an empty string, which automatically updates the **value** inside the **input** field.

Vue:

```
createNewToDoItem() {
    this.list.push(
        {
            'todo': this.todo
        }
    );
    this.todo = '';
}
```

How did Vue do that?

In Vue, our **input** field has a handle on it called **v-model**. This allows us to do something known as **two-way binding**. Let's just quickly look at our input field, then we'll explain what is going on:

```
<input type="text" v-model="todo"/>
```

V-Model ties the input of this field to a key we have in our data object called toDoItem. When the page loads, we have toDoItem set to an empty string, as such: **todo: ''**. If this had some data already in there, such as **todo: 'add some text here'**, our input field would load with *add some text here* already inside

text here, our input field would load with that *some text here* already inside the input field. Anyway, going back to having it as an empty string, whatever text we type inside the input field gets bound to the value for **todo**. This is effectively two-way binding (the input field can update the data object and the data object can update the input field).

So looking back at the **createNewToDoItem()** code block from earlier, we see that we push the contents of **todo** into the **list** array and then update **todo** to an empty string.

How do we delete from the list?

React:

```
deleteItem = indexToDelete => {
    this.setState(({ list }) => ({
      list: list.filter((toDo, index) => index !== indexToDelete)
    }));
};
```

How did React do that?

So whilst the deleteItem function is located inside **ToDo.js**, I was very easily able to make reference to it inside **ToDoItem.js** by firstly, passing the **deleteItem()** function as a prop on **<ToDoItem/>** as such:

```
<ToDoItem deleteItem={this.deleteItem.bind(this, key)}/>
```

This firstly passes the function down to make it accessible to the child. You'll see here that we're also binding **this** as well as passing the key parameter, as key is what the function is going to use to be able to differentiate between which **ToDoItem** is attempting to delete when clicked. Then, inside the **ToDoItem** component, we do the following:

```
<div className="ToDoItem-Delete" onClick={this.props.deleteItem}>-
</div>
```

All I had to do to reference a function that sat inside the parent component was to reference **this.props.deleteItem**.

Vue:

```
onDeleteItem(todo){
  this.list = this.list.filter(item => item !== todo);
}
```

How did Vue do that?

A slightly different approach is required in Vue. We essentially have to do three things here:

Firstly, on the element we want to call the function:

```
<div class="ToDoItem-Delete" @click="deleteItem(todo)">-</div>
```

Then we have to create an emit function as a method inside the child component (in this case, **ToDoItem.vue**), which looks like this:

```
deleteItem(todo) {
    this.$emit('delete', todo)
}
```

Along with this, you'll notice that we actually reference a **function** when we add **ToDoItem.vue** inside of **ToDo.vue**:

```
<ToDoItem v-for="todo in list"
          :todo="todo"
          @delete="onDeleteItem" // <-- this :)
          :key="todo.id" />
```

This is what is known as a custom event-listener. It listens out for any occasion where an emit is triggered with the string of 'delete'. If it hears this, it triggers a function called **onDeleteItem**. This function sits inside of **ToDo.vue,** rather than **ToDoItem.vue**. This function, as listed earlier, simply filters the **todo array** inside the **data object** to remove the item that was clicked on.

It's also worth noting here that in the Vue example, I could have simply written the **$emit** part inside of the **@click** listener, as such:

```
<div class="ToDoItem-Delete" @click="$emit('delete', todo)">-</div>
```

This would have reduced the number of steps down from 3 to 2, and this is simply down to personal preference.

In short, child components in React will have access to parent functions via **this.props** (providing you are passing props down, which is fairly standard practice and you'll come across this loads of times in other React examples), whilst in Vue, you have to emit events from the child that will usually be collected inside the parent component.

How do we pass event listeners?

React:

Event listeners for simple things such as click events are straight forward. Here is an example of how we created a click event for a button that creates a new ToDo item:

```
<div className="ToDo-Add" onClick={this.createNewToDoItem}>+</div>.
```

Super easy here and pretty much looks like how we would handle an in-line onClick with vanilla JS. As mentioned in the Vue section, it took a little bit longer to set up an event listener to handle whenever the enter button was pressed. This essentially required an onKeyPress event to be handled by the input tag, as such:

```
<input type=”text” onKeyPress={this.handleKeyPress}/>.
```

This function essentially triggered the **createNewToDoItem** function whenever it recognised that the ‘enter’ key had been pressed, as such:

```
handleKeyPress = (e) => {

if (e.key === ‘Enter’) {

this.createNewToDoItem();

}

};
```

Vue:

In Vue it is super straight-forward. We simply use the @ symbol, and then the type of event-listener we want to do. So for example, to add a click event listener, we could write the following:

```
<div class=”ToDo-Add” @click=”createNewToDoItem()”>+</div>
```

Note: **@click** is actually shorthand for writing **v-on:click**. The cool thing with Vue event listeners is that there are also a bunch of things that you can chain on to them, such as .once which prevents the event listener from being triggered more than once. There are also a bunch of shortcuts when it comes to writing specific event listeners for handling key strokes. I found that it took quite a bit longer to create an event listener in React to create new ToDo items whenever the enter button was pressed. In Vue, I was able to simply write:

```
<input type=”text” v-on:keyup.enter=”createNewToDoItem”/>
```

How do we pass data through to a child component?

React:

In react, we pass props onto the child component at the point where it is created. Such as:

```
<ToDoItem key={key} item={todo} />
```

Here we see two props passed to the **ToDoItem** component. From this point on, we can now reference them in the child component via this.props. So to access the **item.todo** prop, we simply call **this.props.item**.

Vue:

In Vue, we pass props onto the child component at the point where it is created. Such as:

```
<ToDoItem v-for="todo in list"
           :todo="todo"
           :key="todo.id"
           @delete="onDeleteItem" />
```

Once this is done, we then pass them into the props array in the child component, as such: **props: ['todo']**. These can then be referenced in the child by their name—so in our case, '**todo**'.

How do we emit data back to a parent component?

React:

We firstly pass the function down to the child component by referencing it as a prop in the place where we call the child component. We then add the call to function on the child by whatever means, such as an **onClick**, by referencing **this.props.whateverTheFunctionIsCalled**. This will then trigger the function that sits in the parent component. We can see an example of this entire process in the section *'How do we delete from the list'*.

Vue:

In our child component, we simply write a function that emits a value back to the parent function. In our parent component, we write a function that listens for when that value is emitted, which can then trigger a function call. We can see an example of this entire process in the section *'How do we delete from the list'*.

And there we have it! 🎉

We've looked at how we add, remove and change data, pass data in the form of props from parent to child, and send data from the child to the parent in the form of event listeners. There are, of course, lots of other little differences and quirks between React and Vue, but hopefully the contents of this article has helped to serve as a bit of a foundation for understanding how both frameworks handle stuff 🤓

If you found this useful, be sure to give lots and lots of claps 👋 . Hint, you can leave up to 50! 😉

Translations

Chinese

Japanese

Korean

Polish

Portuguese

Russian

Taiwanese

If you are interested in making this article more accessible by translating into another language, please feel free to do so—let me know if you do so that I can add a link to it on this page. 😄

But what about Angular?

I'm glad you asked! Because the awesome Sam Borick has written Part 2 of this article!

And what about [insert framework/library]?

HyperApp

AppRun

LitElement

If you're interested in forking the styles used in this article and want to make your own equivalent piece, please feel free to do so! And be sure to let me know so that I can add a link to your article 👍

Github links to both apps:

Vue ToDo: https://github.com/sunil-sandhu/vue-todo

React ToDo: https://github.com/sunil-sandhu/react-todo

. . .

I thought I'd also take a moment to let you know that we are currently looking for writers to join our team at Javascript In Plain English. If you're passionate about Javascript and have a story to tell, feel free to contact us at submissions@javascriptinplainenglish.com to find out more. 🤓

JavaScript Vuejs React Vue Front End Development

47K claps 129

Sunil Sandhu
Developer, Editor of JavaScript In Plain English (JSIPE)

Follow

JavaScript In Plain English
Taking the sting out of learning the web's most important programming language.

Follow

More from JavaScript In Plain English
Declaring Event Handlers
Packt_Pub
4 min read
93

More from JavaScript In Plain English
Real-life use cases for computed properties in Vue
Sunil Sandhu
5 min read
1.3K

More from JavaScript In Plain English
How to Implement a GraphQL CRUD BFF
Packt_Pub
3 min read
99

Responses

Write a response...

Conversation between Dmitry Scheglov and Sunil Sandhu.

Dmitry Scheglov
Jul 30 · 1 min read

Well, the key differences between `React` and `Vue` are:

1. Vue implements state observation but React doesn't. But are you sure that it is good for you to observe recursively all component state (how much unnecessary work Vue does in order to observe todo list? At the end of the

observation Vue in any case calls its own…

Read more…

1.4K 15 responses

Sunil Sandhu
Jul 30 · 1 min read

Hi Dmitry, some very good points you have raised here that may be of interest to those who already have a solid understanding of Javascript and web development as a whole :)

26

Applause from Sunil Sandhu (author)

Lucas Everett
Jul 26 · 1 min read

Hi Sunil,

Just a couple notes on React. When adding an item to the list, because React is asynchronous, it's best to not reference this.state.list to do your "mutation." Referencing state directly is the cause of many issues people encounter when learning React. It's best to pass setState a function rather than object to do…

Read more…

584

Applause from Sunil Sandhu (author)

ZEESHAN ARSHAD
Jul 31 · 1 min read

Vue looks like winner in keep it simple stupid way.

473

Conversation between Revanth Kumar and Sunil Sandhu.

Revanth Kumar
Jul 30 · 1 min read

> There are actually reasons here for why React makes mutations differently to Vue, but we won't get into them here.

For those who wanna know why here is a short answer:

This is because react wants to re run certain life cycle hooks (componentWillReceiveProps, shouldComponentUpdate, componentWillUpdate, render, componentDidUpdate) when state changes and it would know state has changed or will be changed when you call the setState function…

Read more…

123 3 responses

Sunil Sandhu
Jul 30 · 1 min read

This is a great, succinct answer—would you mind if I added this and credited yourself?

40 1 response

Revanth Kumar
Jul 30 · 1 min read

Definitely.

20 1 response

Sunil Sandhu
Jul 30 · 1 min read

Added and credited! :)

15

Sunil Sandhu
Jul 28 · 1 min read

Hi all, this article has now been updated to include some key changes to both set ups. Special thanks to Lucas Everett and Dan Charousek for the suggestions 🤓

81 1 response

Applause from Sunil Sandhu (author)

Rob Muhlestein
Jul 31 · 1 min read

> whereas Vue CLI adopts an all encompassing approach, where the styles are declared inside the actual component file.

Just so much cleaner.

145 3 responses

Conversation between Chintan Dhandhusariya and Sunil Sandhu.

Chintan Dhandhusariya
Jul 31 · 1 min read

> React handles two-way binding by having an additional onChange function attached to the input field.

Actually, React doesn't *handle* it directly, **the developer** has to write the onChange functions for inputs. This could mislead someone who's new and doesn't know React. (If I didn't know React & I would just be going through this article, I would've had that confusion)

162 1 response

Sunil Sandhu
Jul 31 · 1 min read

Hi Chintan, this is a valid point that you raise—I've amended the text to better explain that we physically create two-way binding rather than React simply handling this itself. Thank you for your input :)

20 1 response

Conversation between Toni Almeida and Sunil Sandhu.

Toni Almeida
Aug 1 · 1 min read

What about the final app sizes? Which one did the same with less KB's?

Thanks!

76 1 response

Sunil Sandhu
Aug 2 · 1 min read

That's a fine question Toni. I didn't try to build production versions of each - as I was more interested in the 'how do you make things work' in each framework, rather than the compressed size. Both repositories are up on Github though, so I welcome you to try both and let me know which one wins!

My initial prediction would be that Vue would come in smaller though.
Thanks

9 1 response

Toni Almeida
Aug 2 · 1 min read

Sunil Sandhu, just did it! Here are the results (as the difference in css is irrelevante, these are the values for JS only)

VUE: 77,84 Kb

React: 127,24 Kb

50Kb difference! amazing win for VUE.

396 4 responses

Sunil Sandhu
Aug 2 · 1 min read

Awesome! Thanks a bunch for taking the time to do this!

31

Conversation between Sunil Sandhu and Lucas Everett.

Sunil Sandhu
Jul 26 · 1 min read

Hi Lucas Everett. Thank you for taking the time to read and to provide feedback. I'm still very new to React so I'm sure there are going to be a few hiccups along the way for me. I'll be sure to update the article with the changes you have recommended. Also noticed I haven't linked the GitHub repositories for the two apps on here, so I'll get the code updated and post those links too. 😊

42 1 response

Lucas Everett
Jul 27 · 1 min read

You're welcome, Sunil Sandhu.

Another thing that can be changed for React is to manage the todo input in the state rather than use document.querySelector('input').

import React, { Component } from "react";

import "./ToDo.css";

Read more...

22

Applause from Sunil Sandhu (author)

Norrapat Nimmanee
Jul 30 · 1 min read

You can use splice instead of filter

17 2 responses

Conversation between Sam Benskin and Sunil Sandhu.

Sam Benskin
Jul 30 · 1 min read

Great read, you've explained the differences very clearly which has helped me better understand what they are. Any chance of a follow-up where you analyse the differences with approach to styling, routing, etc?

25 1 response

Sunil Sandhu
Jul 30 · 1 min read

Hi Sam, thank you for taking the time to read and I'm really pleased to hear that you found it useful. A fantastic suggestion made and would serve as a natural follow up to this piece—may just have to write that over the coming weeks!

20

Conversation between Chintan Dhandhusariya and Sunil Sandhu.

Chintan Dhandhusariya
Jul 31 · 1 min read

> The entire list is added through the use of a spread operator (Google this if you've not seen this before — it's ES6 syntax).

The reason the spread operator is used is to maintain immutability.

34 1 response

Sunil Sandhu
Jul 31 · 1 min read

Indeed, and it's a very clever way of doing it too! :)

3

Conversation with Sunil Sandhu.

Nicolas Grilly
Jul 30 · 1 min read

> Then we have to create an emit function as a method inside the child component (in this case, ToDoItem.vue), which looks like this:

Instead of using this.$parent.$emit, why don't you simply pass the delete function as a prop in Vue as well?

22 3 responses

Sunil Sandhu
Jul 30 · 1 min read

Hi Nicolas, thank you for raising this. I didn't do it initially because I'm an idiot who is still fairly new to Vue (only been using for a few months). I've since learned this though from fantastic members of the Vue community who have kindly taken the time to make updates to the git repository. I now know better! 🤓

17 1 response

Applause from Sunil Sandhu (author)