How to create a custom toast component with React

March 11, 2020 · 15 min read

Toast notifications are modal-like elements that display information to a user, often in the form of buttons or another call to action. The messages displayed tend to be brief and are sometimes removed via user action or set to auto-expire. Most importantly, toast notifications do not interfere with the user's interaction with your app, whether they are using a desktop or mobile device.

Developers commonly use toast notifications to display, among other things:

- A success message upon a successful form submission or API request
- An error message upon a failed API request
- Chat information

In this tutorial, I'll demonstrate how to create a custom toast component with React. We'll use React hooks such as `useState` and `useEffect`. After creating the toast component, we'll add some simple buttons to try out and display the toast on our page.

Here's what the toast notifications will look like after we create and call them:

You can reference the full source code for this tutorial in the GitHub repo.

After we create some buttons to trigger the toast notifications, the page should look like this:

Finally, we'll demonstrate how to auto-delete toast notifications.

Let's dive in and get started!

Getting started

To demonstrate how to create custom toast components, we must first create a React application. I'll assume that Node.js is already installed on your computer. Node.js comes with npm, and we'll use `create-react-app` with npx to build our React app.

Open a terminal, navigate to the directory where you want to add your project, and type the following.

```
npx create-react-app react-toast
```

You can name the project whatever you want. We will not install any other module inside the project; we'll simply use the modules added by the `create-react-app` tool.

The default folder structure is as follows.

The `src` folder is where we'll do most of our work. Inside `src`, create a new folder called `components`. We'll add our toast and button components to this folder.

In React, you can either use class components, which requires you to extend a `React.Component` and create a render function that returns a React element, or functional components, which are just plain JavaScript functions that accept props and return React elements. We'll use functional components throughout this tutorial. `create-react-app` uses functional components by default.

Inside the `App.js` component, you can remove the content of the header element and change the header to a div with `className="app-header"`. The parent element class should be changed to `app`. We'll also change the function

to an arrow function (that's just my personal preference; feel free to use the default function if you'd like).

```
import React from 'react';

import './App.css';

const App = () => {
    return (
        <div className="app">
            <div className="app-header">

            </div>
        </div>
    );
}
export default App;
```

Next, add the CSS style into the `App.css` file. Delete the contents of `App.css` and add the styles into the file. You can get the CSS styles from GitHub.

Some of the elements with styles in the CSS file have not been added. We'll add these elements as we progress. The styles consist of some simple CSS properties.

Delete the contents of `index.css` and add the following.

```
@import url('https://fonts.googleapis.com/css?family=Roboto&display=swap');
body {
  margin: 0;
  font-family: 'Roboto', 'sans-serif';
}
```

Creating a toast component

To create a toast component, create a folder called `toast` inside the `components` directory and add two files: `Toast.js` and `Toast.css`. We are using the .js extension for our JavaScript files as well as CSS — optionally, you can use JSX and SCSS files.

Inside the `Toast.js` file, create an arrow function called `Toast` and set the export function as `default`. Set the parent element to empty tags.

```
import React from 'react';
const Toast = () => {
    return (
        <>


        </>
    )
}
export default Toast;
```

The function will always return a React element. The first element to add is the notification container, which will wrap every toast notification element that will be displayed.

```
<div className="notification-container">
</div>
```

Later, we'll add a dynamic property to display the position of the notification container. We'll add other elements inside the container to display the button, image, title, and message.

```
<div className="notification toast">
    <button>
        X
    </button>
    <div className="notification-image">
        <img src="" alt="" />
    </div>
    <div>
        <p className="notification-title">Title</p>
        <p className="notification-message">Message</p>
    </div>
</div>
```

The button will be used to close a particular toast notification. An image icon will display depending on the type of toast. We will essentially end up with four types of toast:

1. Success
2. Danger
3. Info
4. Warning

Import the `Toast.css` file into the component and add the following CSS style for the `notification-container` to the `Toast.css` file.

```
.notification-container {
    font-size: 14px;
    box-sizing: border-box;
    position: fixed;
}
```

We'll have four different positions for the toast elements:

1. Top-right
2. Bottom-right
3. Top-left

4. Bottom-left

Below are the CSS styles for the position.

```
.top-right {
    top: 12px;
    right: 12px;
    transition: transform .6s ease-in-out;
    animation: toast-in-right .7s;
}

.bottom-right {
    bottom: 12px;
    right: 12px;
    transition: transform .6s ease-in-out;
    animation: toast-in-right .7s;
}

.top-left {
    top: 12px;
    left: 12px;
    transition: transform .6s ease-in;
    animation: toast-in-left .7s;
```

The positions will be added dynamically, depending on which `position props` the user adds to the toast component.

The next CSS styles are for styling the notification class, which contains the remove button, image, title, message, and animations to slide the toast either left or right of the page. Copy the styles from the GitHub repo and add them into the `Toast.css` file.

To see what the toast component looks like, let's apply some properties, such as `position` , to be passed as `props` inside the toast component.

`Props` , or `properties` , are used for passing data from one component to another in React.

The toast component takes in two props: `toastList` and `position`. `toastList` represents an array that will contain objects, and `position` determines the placement of the notification container on the page. Let's add a `props` parameter to the `Toast` function and then use the ES6 object `destructing` to get the `toastList` and `position` props.

```
const Toast = (props) => {
    const { toastList, position } = props;

    return (
        <>
            ...
        </>
    )
}
export default Toast;
```

To use the `position` prop, add it to the element with a `className` of `notification-container`. Remove the class from the `notification-container`, then add:

```
className={`notification-container ${position}`}
```

Next, remove the class from notification div and add the following.

```
className={`notification toast ${position}`}
```

Whatever position prop is passed into the toast component, it will be added as a class to those elements (recall that we already set the CSS position properties in the CSS file).

Since `toastList` is an array, we can loop through it directly in the HTML, but I am not going to do that. Instead, I will use the `useState` hook to create a new property. `useState` allows you to create a stateful variable and a function to update it.

First, import the `useState` and `useEffect` hooks from React where the `useState` will be used to create a variable and a function to update the variable. The `useEffect` hook will be called when there is a rerendering required.

```
import React, { useState, useEffect } from 'react';
```

Add this after the props destructuring:

```
const [list, setList] = useState(toastList);
```

The default value of the `useState` list is going to be whatever the default value of the `toastList` array is.

Add the `useEffect` method and use the `setList` to update the list property.

```
useEffect(() => {
    setList(toastList);
}, [toastList, list]);
```

The `useEffect` hook takes a function and an array of dependencies. The `setList` method is used to update the list array whenever a new object is added to the `toastList` array, which is passed as a prop. The array consist of dependencies that are watched whenever there is a change to their values. In other words, the `useEffect` method will always be called when there is an update to the values in the dependencies array.

Let's loop through the list array inside the HTML. We'll use the `map` method to loop through the array.

```
import React, { useState, useEffect } from 'react';

import './Toast.css';

const Toast = props => {
    const { toastList, position } = props;
    const [list, setList] = useState(toastList);

    useEffect(() => {
        setList(toastList);
    }, [toastList, list]);

    return (
        <>
            <div className={`notification-container ${position}`}>
                {
                    list.map((toast, i) =>
                        <div
                            key={i}
```

The structure of the objects that will be added to the `toastList` array looks like this:

```
{
    id: 1,
    title: 'Success',
    description: 'This is a success toast component',
    backgroundColor: '#5cb85c',
    icon: ''
}
```

We'll add background color of the toast dynamically. To achieve that, we need to add a style property to the notification element. On the element with class `notification toast`, add a style property that will use the `backgroundColor` from the list. Add it after the `className`.

```
style={{ backgroundColor: toast.backgroundColor }}
```

Let us now use this component inside the `App` component. Go into the `App` component and import the `toast` component.

```
import Toast from './components/toast/Toast';
```

After the div element with class name of `app-header` , add the `toast` component.

```
<Toast />
```

Now we need to pass the props into the `toast` component. Go to the GitHub repo and download the SVG files for the toast icons. Create a new directory called `assets` inside the src folder and add all the SVG files.

Add the imports to the `App` component.

```
import checkIcon from './assets/check.svg';
import errorIcon from './assets/error.svg';
import infoIcon from './assets/info.svg';
import warningIcon from './assets/warning.svg';
```

Each icon will be used for one of the following types of toast notification: `success` , `danger` , `info` , and `warning` .

To try out the toast component, add this array inside the `App` component (this is just for testing)

```
const testList = [
    {
      id: 1,
      title: 'Success',
      description: 'This is a success toast component',
      backgroundColor: '#5cb85c',
      icon: checkIcon
    },
    {
      id: 2,
      title: 'Danger',
      description: 'This is an error toast component',
      backgroundColor: '#d9534f',
      icon: errorIcon
    },
];
```

Pass this `testList` as a prop to the toast component and set the position to `bottom-right`.

```
<Toast
    toastList={testList}
    position="bottom-right"
/>
```

Recall that `toastList` and `position` are props that we destructured inside the toast component.

Open the project in a terminal and run `npm start` or `yarn start` to start the server. The server should run on port 3000. This is the result on the browser:

You can change the position to see the placement of the toasts on the page. If you hover on the toast elements, you'll see some effects. Add the following objects to the `testList` array.

```
{
    id: 3,
    title: 'Info',
    description: 'This is an info toast component',
    backgroundColor: '#5bc0de',
    icon: infoIcon
},
{
    id: 4,
    title: 'Warning',
    description: 'This is a warning toast component',
    backgroundColor: '#f0ad4e',
    icon: warningIcon
}
```

After adding the other objects to the array, the toast components should look like this:

Let's add some `prop-types` to the toast component. React provides type checking features to verify that components receive props of the correct type. `PropTypes` helps to make sure that components receive the right type of props.

Import `prop-types` from React. The toast component expects two props `toastList` and `position` .

```
import PropTypes from 'prop-types';
```

Add the following props check below the toast component arrow function.

```
Toast.defaultProps = {
    position: 'bottom-right'
}

Toast.propTypes = {
    toastList: PropTypes.array.isRequired,
    position: PropTypes.string
}
```

The `position` prop is not a required prop, but you can make it required if you want. If no position is set, the default position prop will be used.

Adding button components

Now that we've built a toast component, let's create a button component and use the buttons to trigger the toast notifications.

Create a new folder called `button` and add a file called `Button.js`. Paste the following code inside the file.

```
import React from 'react';
import PropTypes from 'prop-types';

const Button = props => {
    const { label, className, handleClick } = props;
    return (
        <>
            <button
                className={className}
                onClick={handleClick}
            >
                {label}
            </button>
        </>
    );
}

Button.propTypes = {
    label: PropTypes.string.isRequired,
```

The props required inside the button components are the label, `className` , and `handleClick` which is the `onClick` method on the button. All we need to do is pass the props into the button component.

Inside the `App` component, import the `Button` component and then add a `BUTTON_PROPS` array just before the `App` arrow function.

```
import Button from './components/button/Button';
```

```
const BUTTON_PROPS = [
  {
    id: 1,
    type: 'success',
    className: 'success',
    label: 'Success'
  },
  {
    id: 2,
    type: 'danger',
    className: 'danger',
    label: 'Danger'
  },
  {
    id: 3,
    type: 'info',
    className: 'info',
    label: 'Info'
  },
```

We're adding this array is so that we can pass the `Button` component inside a loop with the different properties.

Inside the div element with class `app-header` , add the following.

```
<p>React Toast Component</p>
<div className="toast-buttons">
    {
        BUTTON_PROPS.map(e =>
            <Button
                key={e.id}
                className={e.className}
                label={e.label}
                handleClick={() => showToast(e.type)}
            />
        )
    }
</div>
```

Instead of creating four different buttons, we used one button inside a loop. The loop displays the number of buttons according to the length of the `BUTTON_PROPS`.

Create a function called `showToast` and pass a parameter called `type`.

```
const showToast = (type) => {
}
```

Import the `useState` hook and then create a new property called list.

```
const [list, setList] = useState([]);
```

When any button is clicked, the app displays the corresponding toast, depending on the position selected by the user. If no position is selected, the default position is used.

Inside the `showToast` method, we'll use a JavaScript switch statement to pass the corresponding toast object into the `toastList` array.

Create a new variable.

```
let toastProperties = null;
```

We'll randomly generate the IDs of each toast notification because we'll use the IDs when the delete functionality is added.

Inside the `showToast` method, add the following.

```
const id = Math.floor((Math.random() * 100) + 1);
```

The IDs will be between 1 and 100. Each case inside the switch statement will correspond with a button.
The cases are `success`, `danger`, `info` and `warning`.

Add the following switch cases inside the `showToast` method.

```
switch(type) {
    case 'success':
        toastProperties = {
            id,
            title: 'Success',
            description: 'This is a success toast component',
            backgroundColor: '#5cb85c',
            icon: checkIcon
        }
        break;
    case 'danger':
        toastProperties = {
            id,
            title: 'Danger',
            description: 'This is an error toast component',
            backgroundColor: '#d9534f',
            icon: errorIcon
        }
        break;
```

The `toastProperties` object in each case is the same object we had inside the `testList`. You can delete the `testList` array. If the `success` button is clicked, the `toastProperties` object will be added to the list array.

```
setList([...list, toastProperties]);
```

The list array is first destructured using the spread operator and the `toastProperties` object is added. The `setList` method is used to update the list array.

Now the toast component inside the `App` component should look like this:

```
<Toast
    toastList={list}
    position="bottom-right"
/>
```

Here, we're using the `bottom-right` position to place the toast notifications. Let's create a select tag where the user can select a position from the dropdown.

Add these divs below `toast-buttons` div inside the `App` component.

```
<div className="select">
    <select
        name="position"
        value={position}
        onChange={selectPosition}
        className="position-select"
    >
        <option>Select Position</option>
        <option value="top-right">Top Right</option>
        <option value="top-left">Top Left</option>
        <option value="bottom-left">Bottom Left</option>
        <option value="bottom-right">Bottom Right</option>
    </select>
</div>
>
```

The `select` tag has the name, value, and `onChange` properties. Whenever an option is selected, the `position` property will be updated and set to the `value` property on the `select` tag. The position will be updated using a `useState` method inside the `selectPosition` method of the `onChange` .

Add a new `useState` method.

```
const [position, setPosition] = useState();
```

As you can see, the `useState` does not have a default value. That's because we set a default prop for the position inside the toast component. If you don't want to set the default props in the toast component, you can just add the default into the `useState` .

Create a function called `selectPosition` , which takes in a parameter called event. Inside this method, we'll pass the `event.target.value` into the `setPosition` to update the position based on the option selected by the user. We'll also pass an empty array into the `setList` method, which will always clear the list array whenever a new position from the tag is selected.

```
const selectPosition = (event) => {
    setPosition(event.target.value);
    setList([]);
}
```

After setting the position prop on the toast component to the `useState` position variable, the toast component inside the `App` component should look like this:

```
<Toast
    toastList={list}
    position={position}
/>
```

If the user does not select a position before clicking on a button, the default position set on the toast component `prop-types` will be used.

Top-right:

Top-left:

Bottom-left:

Bottom-right:

Deleting toast notifications

Up to this point, we've created a toast component and set notifications to display when buttons are clicked. Now it's time to add a method to delete a toast from the page as well as from the `toastList` array.

We'll use the JavaScript methods `findIndex` and `splice`. We'll also use the unique ID of the toast inside the `toastList` object array to find the index of the object and the use the `splice` method to remove the object from the array, thereby clearing the particular toast from the page.

Go into your toast component. On the button, add an `onClick` with a method called `deleteToast`, which takes a toast ID as a parameter.

```
onClick={() => deleteToast(toast.id)}
```

Create a `deleteToast` method.

```
const deleteToast = id => {
    const index = list.findIndex(e => e.id === id);
    list.splice(index, 1);
    setList([...list]);
}
```

After getting the index of the toast object inside the list array, the index is used inside the splice method, which removes the property at that index from the array. The number `1` lets the splice method know we want to remove only one value.

After removing the toast, use the spread operator to update the list array using the `setList` method. That's all you need to do to delete a toast notification.

See the full source code for this tutorial in the GitHub repo.

If you prefer to watch me as I code, you can check out this video tutorial on YouTube.

Auto-delete toast notifications

Toast notifications can be auto-deleted by adding a delete functionality inside the JavaScript `setInterval` method after a certain amount of time has passed.

The toast notification component will take two new `props` :

- `autoDelete` — a boolean that determines whether the notification needs to be deleted
- `autoDeleteTime` — a number in milliseconds

Add the new properties to the `props` object in the toast component.

```
const { ..., autoDelete, autoDeleteTime } = props;
```

You can add multiple React `useEffect` methods to a functional component as long as you preserve the order in which they are called.

Add another `useEffect` method.

```
useEffect(() => {
}, []);
```

Inside this `useEffect` , add the `setInterval()` method.

```
useEffect(() => {
    const interval = setInterval(() => {

    }, autoDeleteTime);
}, []);
```

The second parameter of the `setInterval` method is `autoDeleteTime`, which is a number in milliseconds that determines how the `setInterval` method is called. The `interval` variable is a number that needs to be cleared by calling the `clearInterval()` method. The `clearInterval()` method clears a timer set with the `setInterval()` method. The interval is cleared inside a `useEffect` cleanup method.

```
useEffect(() => {
    const interval = setInterval(() => {

    }, autoDeleteTime);
    return () => {
        clearInterval(interval);
    }
}, []);
```

The cleanup method is called after the `useEffect` method unmounts and starts a new rerender. If the `clearInterval` method is not called, the `interval` variable will always hold the last timer value, which will cause issues regarding how the `setInterval` method is called.

Let's update the `deleteToast` method by removing items from the `toastList` array.

```
const deleteToast = id => {
    ...
    ...
    const toastListItem = toastList.findIndex(e => e.id === id);
    toastList.splice(toastListItem, 1);
    ...
}
```

Whenever an item is removed from the `list` array, that same item is deleted from the `toastList` array. The `deleteToast` method will be called inside the `setInterval()` .

The `autoDelete` property is a boolean that determines whether the notifications are to be automatically removed after a certain time. If the property is `true` , the notifications are auto-deleted. Otherwise, they are not deleted automatically.

Inside the `setInterval()` , we need to check whether the `autoDelete` is true and whether the `list` and `toastList` arrays have values in them. Recall that the `deleteToast` method requires an `id` parameter to remove the toast from the arrays. We'll get the `id` of the first item in the `toastList` array and pass it into the `deleteToast` method.

```
useEffect(() => {
    const interval = setInterval(() => {
        if (autoDelete && toastList.length && list.length) {
            deleteToast(toastList[0].id);
        }
    }, autoDeleteTime);
    return () => {
        clearInterval(interval);
    }
}, []);
```

Each time the `setInterval` is called, the ID of the item at index 0 is passed into the delete method. The method is skipped if `toastList` and list arrays have no more values in them. We don't need the `defaultProps` inside the component, so it can be removed.

Add these to the props validation:

```
Toast.propTypes = {
    ...
    autoDelete: PropTypes.bool,
    autoDeleteTime: PropTypes.number
}
```

Now that we've updated the toast component with the new properties, let's add the values as props to the component from inside the `App` component. For demonstration purposes, I'll add a checkbox input and a text input so the user can dynamically set the values of the props.

Add two `useState` variables inside the `App` component.

```
let [checkValue, setCheckValue] = useState(false);
const [autoDeleteTime, setAutoDeleteTime] = useState(0);
```

Before the select dropdown tag, add these new elements:

```
<div className="select">
    <input
    id="auto"
    type="checkbox"
    name="checkbox"
    value={}
    onChange={}
    />
    <label htmlFor="auto">Auto Dismiss</label>
</div>

<div className="select">
    <input
    type="text"
    name="checkbox"
    placeholder="Dismiss time Ex: 3000"
    autoComplete="false"
    onChange={}
    />
```

CSS styles for the new elements:

```
input[type=checkbox] + label {
  display: block;
  cursor: pointer;
  margin-top: 1em;
}

input[type=checkbox] {
  display: none;
}

input[type=checkbox] + label:before {
  content: "\2714";
  border: 0.1em solid #fff;
  border-radius: 0.2em;
  display: inline-block;
  width: 1em;
  height: 1em;
  padding-top: 0.1em;
  padding-left: 0.2em;
```

Let's disable the buttons until a position is selected from the select dropdown and also disable the input if the auto-dismiss checkbox is unchecked. To achieve that, add the string `Select Position` to the `useState` method for `position`.

```
const [position, setPosition] = useState('Select Position');
```

Then, on the buttons `className`, add:

```
className={`${position === 'Select Position' ? `${e.className} btn-disable`
: `${e.className}`}`}
```

If the position is the string `Select Position`, add the class `btn-disable` and the class name for each button. If the position is not the string `Select Position`, then add only the button class name.

On the input text box for adding the auto delete time, add:

```
className={`${!checkValue ? 'disabled' : ''}`}
```

The input is disabled by default unless the checkbox is checked.

Disabled buttons and text inputs:

Enabled buttons and text inputs:

Update the toast component.

```
<Toast
    toastList={list}
    position={position}
    autoDelete={checkValue}
    autoDeleteTime={autoDeleteTime}
/>
```

Create a new method, `onCheckBoxChange`, inside the component and add it to an `onChange` method on the checkbox input.

```
<input
    id="auto"
    type="checkbox"
    name="checkbox"
    value={checkValue}
    onChange={onCheckBoxChange}
/>
```

```
const onCheckBoxChange = () => {
    checkValue = !checkValue;
    setCheckValue(checkValue);
    setList([]);
}
```

The default value for `useState` `checkValue` is `false`. If the checkbox is clicked, the value is changed to its opposite since it is a boolean and then updated with `setCheckValue` method and the `list` array is reset to empty.

Add an `onInputChange` method to the text input.

```
<input
    className={`${!checkValue ? 'disabled' : ''}`}
    type="text"
    name="checkbox"
    placeholder="Dismiss time Ex: 3000"
    autoComplete="false"
    onChange={onInputChange}
/>
```

```
const onInputChange = (e) => {
    const time = parseInt(e.target.value, 10);
    setAutoDeleteTime(time);
}
```

We cast the value from a string to a number and passed it into the `setAutoDeleteTime` method. Below is a demonstration of the result of auto-deleting toast notifications.

Auto-delete after two seconds:

Auto-delete after three seconds:

Conclusion

This tutorial should give you a solid understanding of how to create a simple toast component that is capable of displaying multiple notifications. You should now know how to use the popular React hooks `useState` and `useEffect`, display and dismiss a toast notification, and customize the component to your heart's desire.

Uzochukwu Eddie Odozi Follow

Web and mobile app developer. TypeScript and JavaScript enthusiast. Lover of Pro Evolution Soccer (PES).

#react

7 Replies to "How to create a custom toast component with React"

Stefan Says:
August 20, 2020 at 6:01 pm

Reply

Hi Eddie, thank you very much for your detailled explanation. I have modified your coding a little bit to suit my needs perfectly but I´m currently struggeling to get it working properly. Here is my coding so far:

My component is called Notification and is receiving a variable called message via props. This props is an object containing either nothing (empty object) or a few attributes. Based on the attribute “type” the notification is rendered. With my coding I don´t have any notification banner on my page. Can you help me through this?

```
import React, {useEffect} from “react”;
import successIcon from “../../../assets/images/check.svg”;
import warningIcon from “../../../assets/images/warning.svg”;
import errorIcon from “../../../assets/images/error.svg”;
import infoIcon from “../../../assets/images/check.svg”;
import “./Notification.module.css”

const Notification = props => {
let notificationProperties = null;
const { message } = props;

if (Object.keys(message).length > 0) {
const id = Math.floor((Math.random() * 262) + 1);
switch (props.message.type) {
case ‘success’:
notificationProperties = {
id,
title: ‘Success’,
text: props.message.text,
backgroundColor: ‘#5cb85c’,
icon: successIcon
};
break;
case ‘error’:
notificationProperties = {
id,
title: ‘Error’,
text: props.message.text,
backgroundColor: ‘#d9534f’,
icon: errorIcon
};
break;
case ‘warning’:
notificationProperties = {
id,
title: ‘Warning’,
text: props.message.text,
backgroundColor: ‘#f0ad4e’,
icon: warningIcon
```

```
};
break;
case 'info':
notificationProperties = {
id,
title: 'Info',
text: props.message.text,
backgroundColor: '#5bc0de',
icon: infoIcon
};
break;
default:
console.log('default clause...');
}

return (

X

{notificationProperties.title}

{notificationProperties.text}

)
} else {
return
}
}

export default Notification;
```

Uzochukwu Eddie Odozi Says:
August 21, 2020 at 8:24 am

Reply

where are you using the notification component?
Instead of using if/else, you can use a useState hook.

Do you have your code on github? If you do, share the link so i can see your complete code.

Stefan Says:
August 21, 2020 at 5:16 pm

Reply

I can´t share my repo link here in public. It´s almost working properly, I´m still having some css glitch, which I struggle most at the moment. Can I send you my repo link via mail or social media?

Stefan Says:
August 22, 2020 at 7:49 am

Reply

Can I send you the repo link privately? I don´t want to share it here in public.

Uzochukwu Eddie Odozi Says:
August 24, 2020 at 8:38 am

Reply

I sent you an email

Marlon Says:
January 22, 2021 at 8:38 am

Reply

Such a really nice piece of content!!! this is awesome!! Thanks for this awesome post!! My best regards!!

Bernard KANMADOZO Says:
March 31, 2021 at 12:23 pm

Reply

Hello.
You did a great tutorial there and I'm very happy because for several weeks I wanted to make alerts with React. Congratulations.
I modified the code a little to separate the types of toast and to programmatically create toasts whose message changes depending on the situation.
Everything seems to work fine but when I want to display the page, I get the following message:
"Error: Too many re-renders. React limits the number of renders to prevent an infinite loop."
I have been trying in vain to solve the problem for several days now, but I am certainly not succeeding because I am new to React that I learn by autodidact.
I have put all source code on github, here is the link:
https://github.com/kanmaber/ToastTest.git
Could you help me ?

Leave a Reply

Enter your comment here...