Write a React Component Like a Pro

Selcuk Ozdemir · Following
Published in JavaScript in Plain English · 3 min read · May 1, 2024

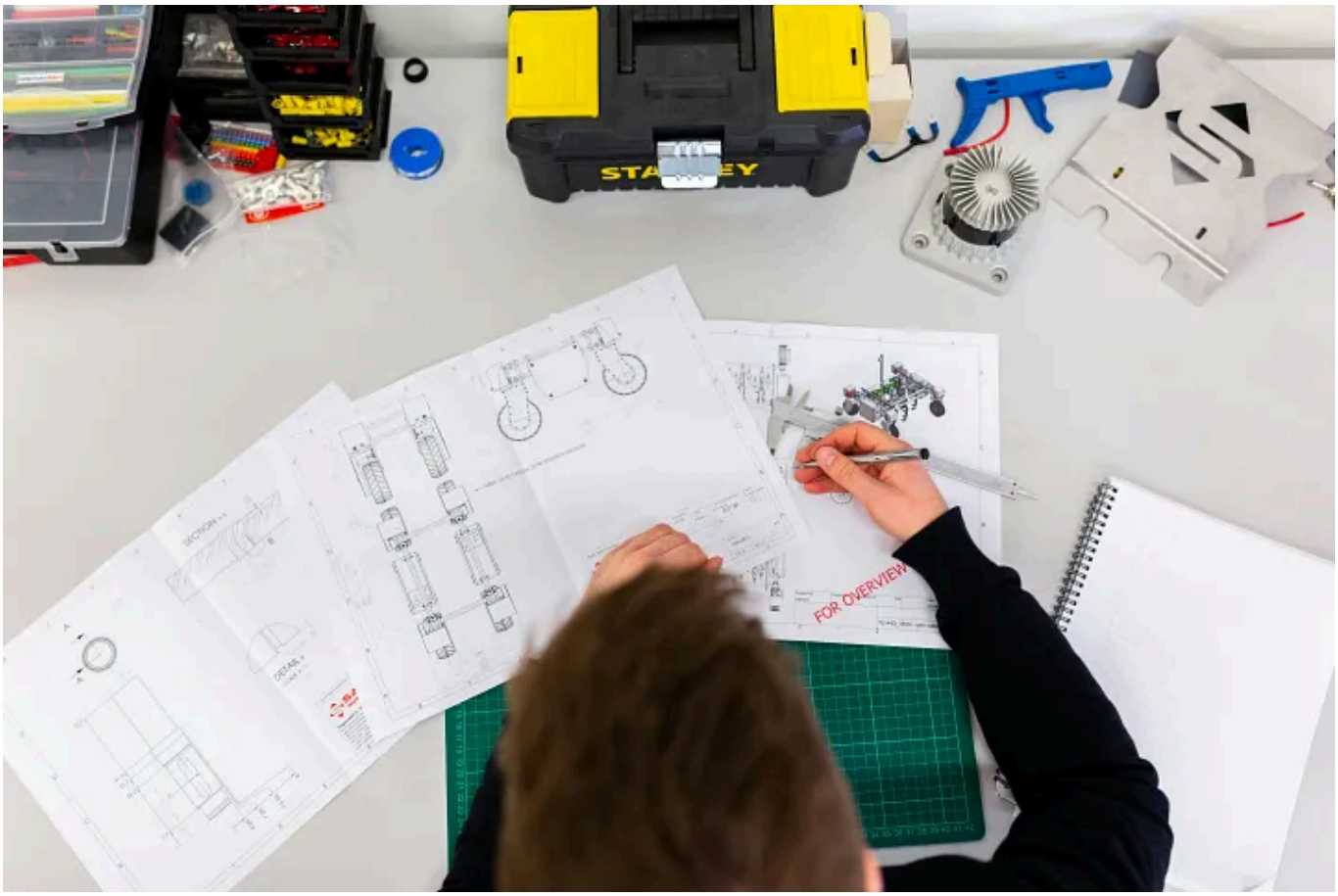


Photo by ThisisEngineering on Unsplash

In the world of React, writing components is an art. It's not just about making them work — it's about making them work well. Today, we're going to look at how to craft your components like a pro, focusing on readability, reusability, and efficiency.

Create a List Component

Let's start with a basic List component:

```
// src/components/List.js
import React from 'react';

const List = ({ data }) => {
  return (
    <ul>
      {data.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
};

export default List;
```

This component takes an array of `data` and renders it as a list.

Enhancing Components with HOCs

Higher-Order Components (HOCs) are a powerful pattern for reusing component logic. They essentially wrap a component to extend its functionality without altering its structure.

For example, a `withLoading` HOC can be used to display a loading state:

```
// src/hocs/withLoading.js
import React, { useState } from 'react';

function withLoading(Component) {
  return function WithLoading({ isLoading, ...props }) {
    if (isLoading) {
      return <div>Loading...</div>;
    }
    return <Component {...props} />;
  };
}

export default withLoading;
```

This HOC checks the `isLoading` prop. If it's true, it renders a "Loading..." message. Otherwise, it renders the wrapped component, allowing for a seamless user experience during data fetching.

Similarly, `withErrorHandling` is another HOC that can manage error states:

```
// src/hocs/withErrorHandling.js
import React from 'react';

function withErrorHandling(Component) {
  return function WithErrorHandling({ error, ...props }) {
    if (error) {
      return <div>Error: {error.message}</div>;
    }
    return <Component {...props} />;
  };
}

export default withErrorHandling;
```

When an error occurs, `withErrorHandling` displays an error message. Otherwise, it renders the component as usual. This HOC is particularly useful for handling fetch errors or issues within the component lifecycle.

By combining `withLoading` and `withErrorHandling`, we can create a robust component that handles both loading and error states elegantly. This approach promotes code reuse and separation of concerns, making our components more maintainable and easier to understand.

Fetching Data with Hooks

React hooks allow us to use state and other React features without writing a class. **useFetch** is a custom hook that fetches data from an API:

```
// src/hooks/useFetch.js
import { useState, useEffect } from 'react';

const useFetch = (url) => {
  const [data, setData] = useState([]);
  const [isLoading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      setLoading(true);
      try {
        const response = await fetch(url);
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        const json = await response.json();
        setData(json);
      } catch (error) {
        setError(error);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
```

```
    // Cleanup function
    return () => {
      // Cleanup logic if needed
    };
  }, [url]);

  return { data, isLoading, error };
};

export default useFetch;
```

It handles the fetching state, data storage, and errors, making it easy to fetch and display data in our components.

Assembling the App

Finally, we bring everything together in the `App` component:

```
// src/App.js
import React from 'react';
import withLoading from './hocs/withLoading';
import withErrorHandling from './hocs/withErrorHandling'; // Yeni HOC eklendi
import useFetch from './hooks/useFetch';
import List from './components/List';

const ListWithLoading = withLoading(List);
const ListWithErrorHandling = withErrorHandling(ListWithLoading); // ListWithLoa

const App = () => {
  const { data, isLoading, error } = useFetch('https://api.example.com/data');

  return (
    <div>
      <h1>List Component</h1>
      <ListWithErrorHandling data={data} isLoading={isLoading} error={error} />
    </div>
```

```
export default App;
```

We use our `useFetch` hook to load data and pass it to our `List` component, which is enhanced with loading and error handling capabilities through our HOCs.

Conclusion

Writing components like a pro means thinking about the bigger picture. It's about creating components that are easy to read, maintain, and reuse. By using patterns like HOCs and hooks, we can create a clean and efficient codebase that stands the test of time.

Happy coding!

In Plain English 🚀

*Thank you for being a part of the **In Plain English** community! Before you go:*

- Be sure to **clap** and **follow** the writer 👏
- Follow us: **X** | **LinkedIn** | **YouTube** | **Discord** | **Newsletter**
- Visit our other platforms: **Stackademic** | **CoFeed** | **Venture** | **Cubed**
- More content at **PlainEnglish.io**

JavaScript React React Native Web Development Software Engineering

Written by Selcuk Ozdemir

Following

356 Followers · Writer for JavaScript in Plain English

Software Engineer at Jotform https://www.linkedin.com/in/scozdev/
https://github.com/scozdev

More from Selcuk Ozdemir and JavaScript in Plain English

Selcuk Ozdemir in JavaScript in Plain English

5 Cool Chrome DevTools Features Most Developers Don’t Know About

Chrome DevTools is an essential and powerful tool for web developers. You can use it to vie...

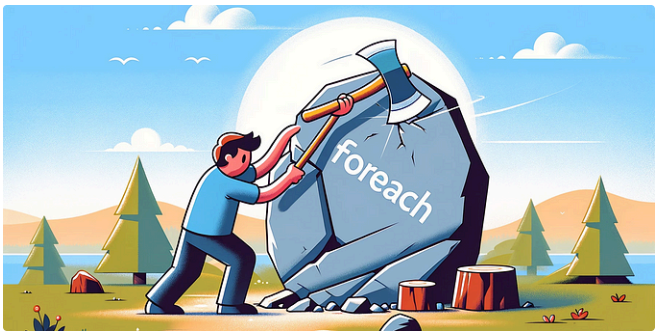
May 14 447 5

Afan Khan in JavaScript in Plain English

The HTMX Team Warned Us

An introduction to HTMX, intercooler.js history, how to use HTMX and the BETH...

Jun 14 201 5

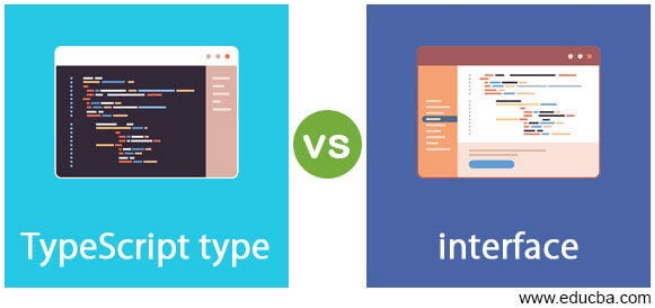


Jayanth babu S in JavaScript in Plain English

JavaScript Interview: Can You Stop or Break a forEach Loop? 🛑

Introduction

Dec 28, 2023 1.8K 26



Vikas Kumar in JavaScript in Plain English

Types Vs Interfaces in TypeScript

TypeScript is full of surprises and some behaviors are only native to TypeScript.

May 12 156 1

See all from Selcuk Ozdemir

See all from JavaScript in Plain English

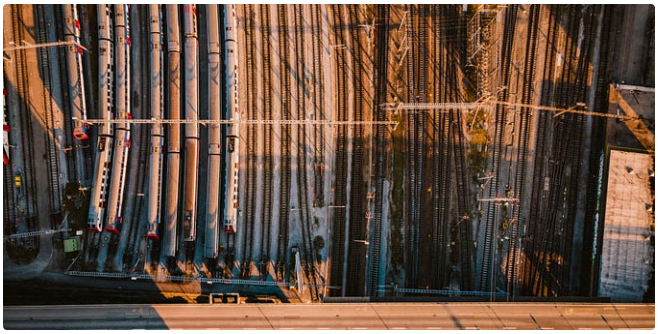
Recommended from Medium

Tari Ibaba in Coding Beauty

5 amazing new JavaScript features in ES15 (2024)

5 juicy ES15 features with new functionality for cleaner and shorter JavaScript code in 2024.

Jun 2 1.4K 11



Andrew Zuo

Async Await Is The Worst Thing To Happen To Programming

I recently saw this meme about async and await.

Jun 21 1.6K 110

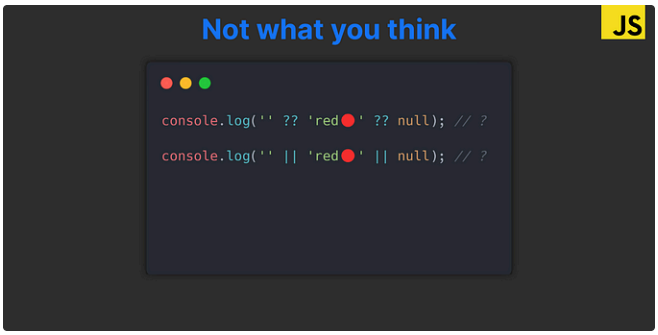
Lists

General Coding Knowledge
20 stories · 1375 saves

Stories to Help You Grow as a Software Developer
19 stories · 1190 saves

Leadership
53 stories · 378 saves

Coding & Development
11 stories · 696 saves

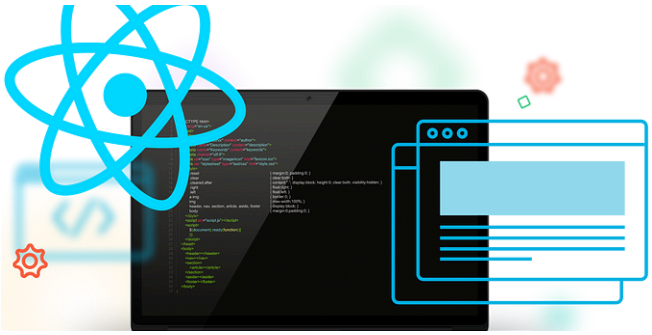


Tari Ibaba in Coding Beauty

?? vs || in JavaScript: The little-known difference

Learn it once and for all and avoid painful bugs down the line.

Jun 8 464 3

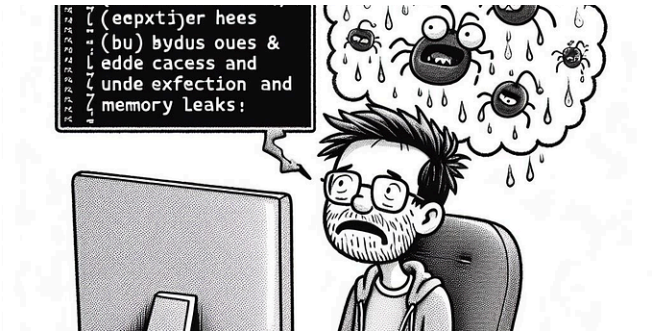


Bryan Aguilar

React Design Patterns

Learn how to apply design patterns in your React applications.

Feb 28 279 2



Daniel Craciun in Level Up Coding

The Dark Side of useEffect in React

One Mistake away from Disaster

Mar 3 491 8

Deepak Chaudhari

Advanced JavaScript Concepts: 2024

Description: Uncover the intricacies of advanced JavaScript concepts, from nested...

Jan 18 1.3K 6

See more recommendations