Write a React Component Like a Pro

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

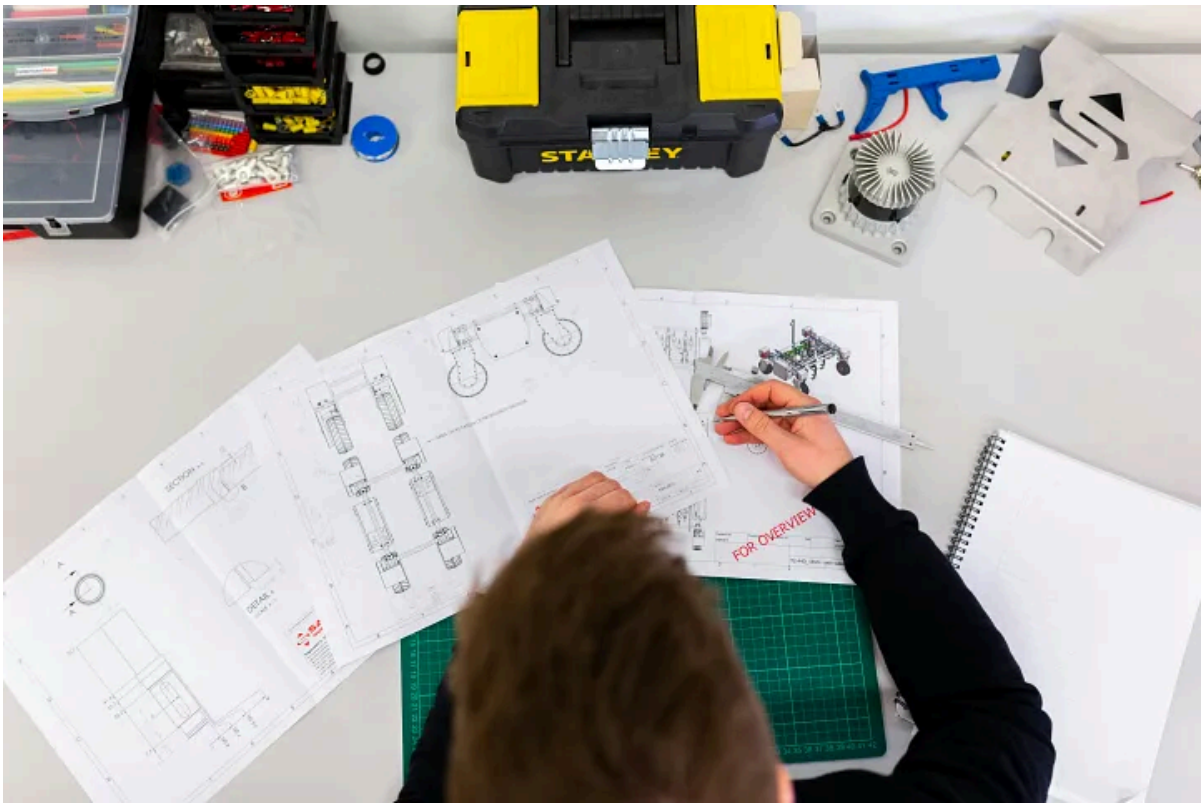


Photo by ThisisEngineering on Unsplash

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

Follow

270 Followers · Writer for JavaScript in Plain English

Software Engineer at Jotform

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...

4 min read · May 14, 2024

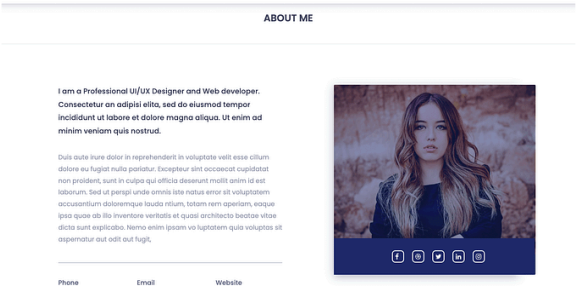
Rehan Pinjari in JavaScript in Plain English

15 Time-Saving Websites Every Developer Needs

Ever thought that there aren't enough hours in the day for all your never-ending...

5 min read · Apr 29, 2024

412 5 1.4K 17



Niemvuilaptrinh in JavaScript in Plain English

40 Portfolio Templates Free For Web Design

Today we will together learn about beautiful, free portfolio templates for website design...

5 min read · Dec 27, 2021

806 3

Rehan Pinjari in JavaScript in Plain English

The Absolute Best Free Tools for Web Developers in 2024 (Seriousl...

Whether you're an experienced developer or just starting, free tools can be a game...

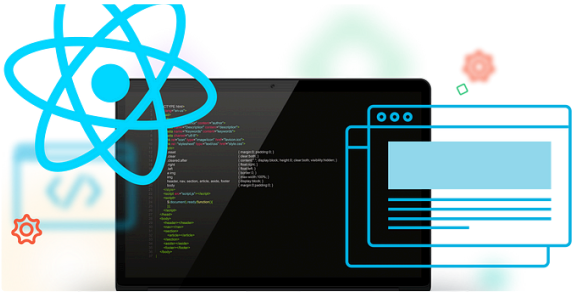
7 min read · Apr 10, 2024

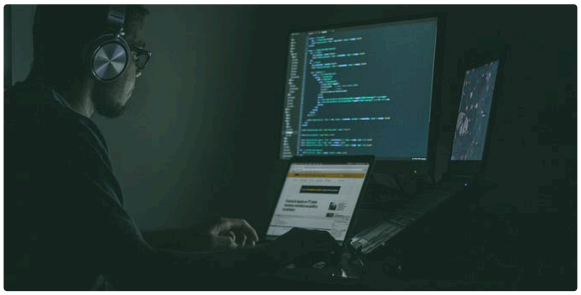
1.2K 7

See all from Selcuk Ozdemir

See all from JavaScript in Plain English

Recommended from Medium





Bryan Aguilar

React Design Patterns

Mate Marschalko

Learn how to apply design patterns in your React applications.

19 min read · Feb 28, 2024

164

28 JavaScript One-Liners every Senior Developer Needs to Know

Learn how to implement complex logic with beautifully short and efficient next-level...

· 9 min read · Mar 1, 2024

899 11

Lists

General Coding Knowledge
20 stories · 1285 saves

Stories to Help You Grow as a Software Developer
19 stories · 1119 saves

Leadership
50 stories · 346 saves

Coding & Development
11 stories · 645 saves

Enes Talay in CodeX

Stop Using find() Method in JavaScript

Forget the find() Method in JavaScript: Alternative Approaches for Cleaner Code

5 min read · Apr 1, 2024

2.2K 36

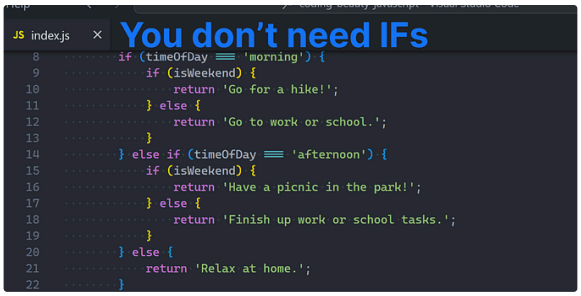
Brian Jenney

How You Can Start a 5 Figure Side Business as Software Engineer

I've started too many failed businesses to count.

6 min read · Apr 22, 2024

1.6K 20



Oliver in Stackademic

5 Custom React Hooks Every Developer Should Know

As a seasoned ReactJS developer, I've had my fair share of challenges and triumphs while...

4 min read · Apr 22, 2024

840 9

Tari Ibaba in Coding Beauty

You don't actually NEED if statements (ever)

Powerful 💪 IF upgrades to completely transform your JavaScript code

· 5 min read · Apr 22, 2024

1.1K 67

See more recommendations