A dynamic React component which requires data to be fetched will have a brief period when there is no data to show.

This is a very common scenario and it is important to handle it properly. I will share the different ways you can handle asynchronous requests in dynamic React component including handling all stages of the request.

Lets consider the choices a developer can make.

Display nothing until the request is complete

This is the easiest thing to do. Show nothing unless the data exists.

Consider a component that has a state object containing an array property like this:

```
this.state = {
    items: []
};
```

The items must be requested from an API resource, so they are initially empty.

The **componentDidMount** function contains the call to a separate asynchronous function.

```
componentDidMount(){
    fetchData().then(items => {
        this.setState({
            items
        });
    });
}
```

```
render(){
    const { items } = this.state; // items are initially empty
    <div>
        { items.map(/* ... */) }
    </div>
}
```

This method will show an empty space for a moment until the request has completed and the state is updated with the new results.

This delay of the render can cause confusion for the user as the user is not aware that the component is loading.

It not very graceful.

Display a loading message

To improve upon the first method, the next best thing to do is to display a loading message until the request has completed.

For this we can add another property to the state object called 'isLoading'.

```
this.state = {
    items: [],
    isLoading: true
};
```

This value is initially 'true' until the request has completed.

```
    fetchData().then(items => {
        this.setState({
            items,
            isLoading: false
        });
    });
}
```

Our render function now has a conditional statement which will render a loading message.

```
render(){
    const { items, isLoading } = this.state;
    <div>
        { isLoading ? <p>Loading</p> : items.map(/* ... */) }
    </div>
}
```

Now the user is aware that the component is loading. No more confusion... That is unless the API request has failed.

If the request fails, then the user cannot tell if there has been an error with the request or if there are simply no results to show.

Handle all stages of the request

To combat this problem, we need to identify the 4 stages of a dynamic component:

- Idle
- Loading
- Success
- Fail

Once the request is made to fetch the data, it is changed to the 'loading' state. Then depending on the response, it will change to the 'success' or 'fail' state.

To implement these stages, we need to make a change to the state object:

```
this.state = {
    items: [],
    loadState: 'idle'
};
```

Now we are using the 'loadState' property to identify the stage of the request process. We must change the componentDidMount function to reflect the 'loadState' values.

```
componentDidMount(){
    this.setState({
        loadState: 'loading'
    });
    fetchData().then(items => {
        this.setState({
            items,
            loadState: 'success'
        });
    }).catch(e => {
        this.setState({
            loadState: 'fail'
        });
    });
}
```

Our render function can be changed to this:

```
    switch(loadState) {
        case 'loading':
            return <p>Loading...</p>;
        case 'fail':
            return <p>There has been a problem!</p>;
        case 'success':
            return items.map(/* ... */);
        case idle:
        default:
            return null;
    }
};

<div>
    { renderContent() }
</div>
}
```

The render function now handles all stages of the dynamic request gracefully and will even show an error message if the request has failed.

Ian Jackson

I help bring our designs to life on the front end while collaborating closely with the UX team. Always ready to tackle any project head on no matter how daunting, I like to keep on top of all things new and exciting in the world of web development.

VIEW ALL ARTICLES BY THIS AUTHOR

Email *

Comment *

ClearPeople is committed to protecting and respecting your privacy, and we'll only use your personal information to administer your account and to provide the products and services you requested from us. From time to time, we would like to contact you about our products and services, as well as other content that may be of interest to you. If you consent to us contacting you for this purpose, please tick below to say how you would like us to contact you:

I agree to receive other communications from ClearPeople.

In order to provide you the content requested, we need to store and process your personal data. If you consent to us storing your personal data for this purpose, please tick the checkbox below.

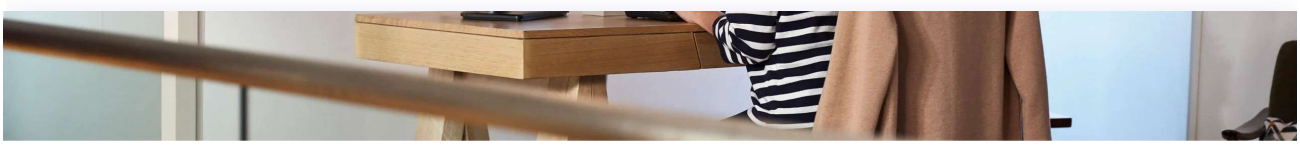
I agree to allow ClearPeople to store and process my personal data. *

You can unsubscribe from these communications at any time. For more information on how to unsubscribe, our privacy practices, and how we are committed to protecting and respecting your privacy, please review our Privacy Policy.

protected by reCAPTCHA
Privacy - Terms

SUBMIT COMMENT

Related blogs

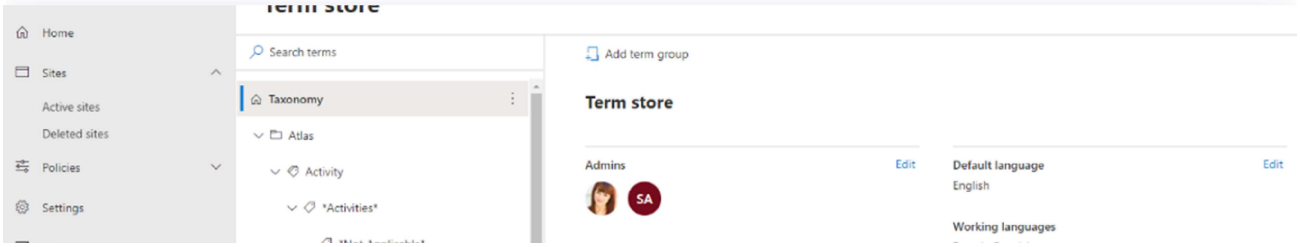


September 24, 2021

How to make hybrid working successful

September 22, 2021

Intranet vs digital workplace solutions



August 6, 2021

SharePoint Term Store management and restoring deleted metadata

Contact form

Careers

Privacy & Disclaimer

Accessibility

Standard Terms

Atlas

Book a demo

Atlas Digital Workspace

Atlas & Microsoft Viva

Atlas Knowledge Management

Atlas Communications Intranet

Atlas Extranet

Atlas Matter Management

Insights

News

Resources

Blog

Events

Tips for Teams

Expertise

Microsoft Office 365 Knowledge Management

Microsoft Office 365 Adoption

Microsoft Office 365 Intranet

SharePoint Knowledge Management

SharePoint Document Management

SharePoint Intranet

Microsoft Teams Consultancy