



Gaurav Singhal

How to Handle Ajax with React

Gaurav Singhal

Feb 5, 2020 • 14 Min read • 19,025 Views

Feb 5, 2020 • 14 Min read • 19,025 Views

Web Development

React

Introduction



- [Introduction](#)
- [When to Make Ajax Requests in React](#)
- [Making Ajax Calls in React](#)
- [Using Axios to Make HTTP Calls in React](#)
- [Using Fetch to Make HTTP Calls in React](#)

Introduction

React is a JavaScript-based library used to create a user interface with component-based architecture. While using it, we also need to access data from external remote endpoints.

Most web-based or mobile-based apps need to access data from a remote server to render data for user interactivity, for example, static data for payments, for item details, and so on.

- [Conclusion](#)
- [Top ^](#)

Since the HTTP call is the primary thing while working in a web app, we have to choose a suitable library based on our functional requirements.

We will learn about how to use the Ajax call into a React app in this guide.

When to Make Ajax Requests in React

When working with a web app, we should have data to be rendered, and the data should be consumed either from local mock data or from a remote endpoint.

React does not have an in-built package for calling the API because React is a library, as opposed to Angular, which is a complete model view controller framework.

We need to call APIs from a remote endpoint to access external resources, which we can do using Ajax to configure the request and respond to the resources.

But the limitation is that we cannot access the remote endpoint, so we need third-party libraries such as Axios or an in-built one like Fetch to consume the data coming from the server.

Making Ajax Calls in React

As we know, React is a JavaScript-based library that does not have the ability to make HTTP requests; thus, we need to use third-party libraries to achieve this.

There are plenty of libraries available to make HTTP calls into React apps. A few of them are listed below.

- Axios
- Fetch
- Superagent
- React-axios
- Use-http
- React-request

Many more third-party libraries are supported for making HTTP requests; We can choose any..

Using Axios to Make HTTP Calls in React

Axios is a promise-based HTTP client that supports the browser and the node.js as well.

Below are the few features of the Axios library.

- It is used to make **XMLHttpRequests** from the browser .
- It makes HTTP requests from the node.js platform .
- It intercepts the request and the response .
- It automatically transforms JSON-specific data .
- You can cancel requests any time .
- It supports the complete promise-based API .
- It transforms the request and response data .

As we can see, many features are supported; that's why the Axios is one of the most widely used libraries on the market.

Let's install the Axios library into the existing application using the below command.

```
1      npm install axios
```

powershell

After executing the above command, we will be able to use the Axios library into the existing React application.

But if we don't want to use **npm package**, then we can also import the script into the existing application like this.

```
1      <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

javascript

The basic syntax of Axios will look like this.

```
jsx
axios
  .get("API_URL")
    .then(response => {
      1           // manipulate the response here
      2     })
      3   .catch(function(error) {
      4           // manipulate the error response here
      5   });
      6
      7
      8
```

As you can see in the this syntax, we should pass the API URL accordingly so that it can access the remote endpoint to fetch the data from the server.

After getting the appropriate response from the server, we should manipulate the response from the component so that we can make use of the response data.

Let's jump directly into an example that shows how to use Axios in our React app by using a dummy endpoint URL to get dummy data to render.

jsx

```
async getTodos() {
  // Simple one
  let todos = await axios.get(
    "https://jsonplaceholder.typicode.com/todos?_page=1&_limit=10"
  );
  console.log(todos.data);
}
```

In this example, we have used the Axios package and the [GET HTTP](#) call, with the URL provided as an argument.

But what if we get the response from the server? We don't have a section in the above example that shows how to use the response from the server, so the solution is given below along with the response section.

jsx

```
axios
  .get("API_URL")
  .then(response => {
    this.setState({ todos: response.data });
  })
  .catch(function(error) {
    console.log(error);
});
```

The `.then` the section manages the response if we get the expected response from the server. If not, then we have an additional section called `.catch` that is used to manage the error while accessing the remote endpoint.

Now let's access the different properties of the response object, like this.

```
jsx
1  async getTodos() {
2    // With all properties
3    axios
4      .get("https://jsonplaceholder.typicode.com/todos?_page=1&_limit=1")
5      .then(response => {
6        console.log(response.data);
7        console.log(response.status);
8        console.log(response.statusText);
9        console.log(response.headers);
10       console.log(response.config);
11     })
12     .catch(function(error) {
13       console.log(error);
14     });
15 }
```

We have accessed different properties of the response object, including data, status, headers, config, and status text. We can make

use of such properties to do things like show the response status, like 200, 404, 502, and so on.

Let's post the data using the Axios **POST** request with this example.

```
jsx
1  async getTodos() {
2      // With all properties
3      let body = {
4          userId: 1111,
5          title: "This is POST request with body",
6          completed: true
7      };
8      axios
9          .post("https://jsonplaceholder.typicode.com/todos", body)
10         .then(function(response) {
11             console.log(response.data);
12         })
13         .catch(function(error) {
14             console.log(error);
15         });
16     }
```

Here in this example, we have used the **POST** request along with the Axios client that passes the additional request data to post the record to the server.

The complete example will look like this.

```
1  import React, { Component } from "react";
2  import axios from "axios";
3
4  class UsingAxios extends Component {
5      constructor() {
6          super();
7          this.state = {
8              name: "React"
9          };
10         this.getTodos = this.getTodos.bind(this);
11     }
12
13     componentDidMount() {
14         this.getTodos();
15     }
16
17     async getTodos() {
18         // With all properties
19         let body = {
20             userId: 1111,
21             title: "This is POST request with body",
22             completed: true
23         };
24         axios
25             .post("https://jsonplaceholder.typicode.com/todos", body)
26             .then(function(response) {
27                 console.log(response.data);
28             })
29             .catch(function(error) {
30                 console.log(error);
31             });
32     }
33
34     render() {
```

```
35         const { todos } = this.state;
36         return (
37             <div>
38                 <h3>Using Axios in React for API call</h3>
39                 <hr />
40             </div>
41         );
42     }
43 }
44
45 export default UsingAxios;
```

This example shows that we will add one todo item using the Axios [POST](#) request, bypassing the additional object, which is the request object. If the request is successful, then the data can be used for further manipulation.

That is why Axios is a good promise-based HTTP client to make HTTP-based calls to access data from the server or update existing data.

Now we will use a different approach for making an HTTP request called [fetch\(\)](#).

Using Fetch to Make HTTP Calls in React

Fetch is browser-based API which is widely used to make HTTP calls from apps, but it's not limited to React—it can also be used from other frameworks such as Angular or Vuejs.

Fetch is a modern method used by developers around the globe because it supports all the modern browsers, but we have to keep in mind that a few browsers still don't support Fetch, which is a drawback.

Let's look at the basic syntax of the Fetch API, which looks like this.

jsx

```
1  let response = fetch(  
2      API_URL,  
3      [additioanl options]  
4  );
```

Here in the syntax, we pass the `API URL`, which is mandatory to fetch the data from the remote server. If we have other request types, like `POST` or `PUT`, then we can also pass the additional arguments along with the URL.

We can simply use the URL with the Fetch API like this.

jsx

```
1  let response = fetch("https://jsonplaceholder.typicode.com/users");
```

The Fetch API always returns the promise so that it can be resolved into the response object as soon as we get the appropriate response from the remote server.

Let's look at the additional options we can use along with the Fetch API with a simple example.

```
jsx
1  async getUsers() {
2      // With additional headers
3      const response = await fetch("https://jsonplaceholder.typicode.com/us
4          method: "GET", // *Type of request GET, POST, PUT, DELETE
5          mode: "cors", // Type of mode of the request
6          cache: "no-cache", // options like default, no-cache, reload, force
7          credentials: "same-origin", // options like include, *same-origin,
8          headers: {
9              "Content-Type": "application/json" // request content type
10         },
11         redirect: "follow", // manual, *follow, error
12         referrerPolicy: "no-referrer", // no-referrer, *client
13         // body: JSON.stringify(data) // Attach body with the request
14     });
15     this.setState({ users: await response.json() });
16 }
```

Here in this example, we have a simple API URL, but along with the URL, we have used various arguments.

- **Method:** Which type of HTTP calls
- **Mode:** The type of request mode, i.e. `cors`, `no-cors`, and so on

- **Headers:** The additional request header to specify the authorization type, content type of the request, and other options
- **Body:** If we have additional request data, we can use the body option, which passes the data to the server as requested data

Other options are also supported, but you don't need to use every option along with every API call. You can just make use of the required one in a specific HTTP call.

We can also get the response by using the `.then` clause along with the `.catch` block to manage the response and the error while making the HTTP call, like this.

```
jsx
1  async getUsers() {
2      // With .then and .catch section
3      let response = await fetch("https://jsonplaceholder.typicode.com/user");
4          .then(response => {
5              return response.json();
6          })
7          .catch(error => {
8              console.log(error);
9          });
10     }
```

As you can see, we have two additional sections called `.then` and `.catch`. Both of these sections are used to get the response from

the server, and if something goes wrong while making an HTTP call, then the appropriate error will be highlighted.

Now let's try the `POST` request to post the data to the server using the `POST` as the request type with Fetch.

jsx

```
1  async getUsers() {
2      let body = {
3          userId: 1111,
4          title: "This is POST request with body",
5          completed: true
6      };
7      fetch("https://jsonplaceholder.typicode.com/todos", {
8          method: "POST",
9          body: JSON.stringify(body)
10     })
11     .then(response => {
12         let json = response.json();
13         console.log(json);
14         if (!response.ok) {
15             throw new Error("Network response was not ok");
16         }
17         return response.blob();
18     })
19     .catch(error => {
20         console.error(
21             "There has been a problem with your fetch operation:",
22             error
23         );
24     });
25 }
```

In this method, we have used the additional options with the URL to specify the type of request and a body attached to the request.

jsx

```
1   fetch("https://jsonplaceholder.typicode.com/todos", {  
2       method: "POST",  
3       body: JSON.stringify(body)  
4   })
```

The property `method` specifies that the request is the `POST` and we have to pass the requested data as the body of the API call. Here is the complete code snippet of the file.

jsx

```
1   import React, { Component } from "react";  
2  
3   class UsingFetch extends Component {  
4       constructor() {  
5           super();  
6           this.getUsers = this.getUsers.bind(this);  
7       }  
8  
9       componentDidMount() {  
10           this.getUsers();  
11       }  
12  
13       async getUsers() {  
14           // With error handling  
15           let body = {  
16               userId: 1111,  
17               title: "This is POST request with body",
```

```
18         completed: true
19     };
20     fetch("https://jsonplaceholder.typicode.com/todos", {
21         method: "POST",
22         body: JSON.stringify(body)
23     })
24     .then(response => {
25         let json = response.json();
26         console.log(json);
27         this.setState({ users: json })
28         if (!response.ok) {
29             throw new Error("Network response was not ok");
30         }
31         return response.blob();
32     })
33     .catch(error => {
34         console.error(
35             "There has been a problem with your fetch operation:",
36             error
37         );
38     });
39 }
40
41 render() {
42     const { users } = this.state;
43     return (
44         <div>
45             <h3>Using Fetch in React for API call</h3>
46             <hr />
47             {users &&
48                 users.map((user, index) => {
49                     return <p key={user.id}>{user.name}</p>;
50                 })}
51             <div />
52         </div>
```

```
53      );
54    }
55  }
56
57 export default UsingFetch;
```

We have called the method `getUsers()`, which contains our API call, using the Fetch API. As soon as we get the response, we can show the list of users coming from the response.

This is how to use the Fetch API to make HTTP calls and manipulate the response based on the functional requirement.

Conclusion

In this guide, we have learned how to implement Ajax in a React app and covered two different approaches to make HTTP calls, Axios and Fetch API.

You can also choose any other libraries from the [npm registry](#). I hope this guide helps you to choose the appropriate way to make Ajax calls. Stay tuned for more upcoming guides.



