Hooks

Hooks are a new addition in React 16.8. **They let you use state and other React features without using a class.** Because of this, class components are generally no longer needed. Although Hooks generally replace class components, there are no plans to remove classes from React.

If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing function component

Hook Rules

There are 3 rules for hooks:

- Hooks can only be called **inside React function components**.
- Hooks can only be called at the top level of a component.
- Hooks cannot be conditional

Note: Hooks will not work in React class components.

`useState` is a Hook that lets you add React **useState Hook**

state to function components.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  var [count, setCount] = useState(0);
  var [age,setAge] = useState(18);
}
```

What does calling `useState` do?

It declares a “state variable”. Our variable is called `count` but we could call it anything else, like `banana`. This is a way to “preserve” some values between the function calls — `useState` is a new way to use the exact same capabilities that `this.state` provides in a class. Normally, variables “disappear” when the function exits but state variables are preserved by React.

What do we pass to `useState` as an argument? The only argument to the `useState()` Hook is the initial state. Unlike with classes, the state doesn’t have to be an object. We can keep a number or a string and we can stored object. In our example, we just want a number for how many times the user clicked, so pass `0` as initial state for our variable. (If we wanted to store two different values in state, we would call `useState()` twice.)

What does `useState` return? It returns a pair of values: the current state and a function that updates it. This is why we write `var [count, setCount] = useState()`. This is similar to `this.state.count` and `this.setState` in a class, except you get them in a pair. Let us see example of hook.

```
        import React, { useState } from 'react';
         function Example() {
          //declration of state variable using hook
           var [count, setCount] = useState(0);
           return (
             <div>
               <p>You have clicked this button {count} times</p>
               <button onClick={() => setCount(count + 1)}>Click me</button>
             </div>
           );
         }
```

Here, `useState` is a *Hook* We call it inside a function component to add some local state to it. React will preserve this state between re-renders. `useState` returns a pair: the *current* state value and a function that lets you update it. You can call this function from an event handler or somewhere else. It's similar to `this.setState` in a class, except it doesn't merge the old and new state together.

The only argument to `useState` is the initial state. In the example above, it is `0` because our counter starts from zero. Note that unlike `this.state`, the state here doesn't have to be an object — although it can be if you want. The initial state argument is only used during the first render.

Effect Hook

You've likely performed data fetching, subscriptions, or manually changing the DOM from React components before. We call these operations "side effects" (or "effects" for short) because they can affect other components and can't be done during rendering.

The Effect Hook, `useEffect`, adds the ability to perform side effects from a function component. It serves the same purpose as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` in React classes, but unified into a single API.

```
import React, { useState, useEffect } from 'react';
function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
useEffect(() => {
// Update the document title using the browser API
document.title = `You clicked ${count} times`;
});
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

When you call `useEffect`, you're telling React to run your "effect" function after flushing changes to the DOM. Effects are declared inside the component so they have access to its props and state. By default, React runs the effects after every render — *including* the first render.

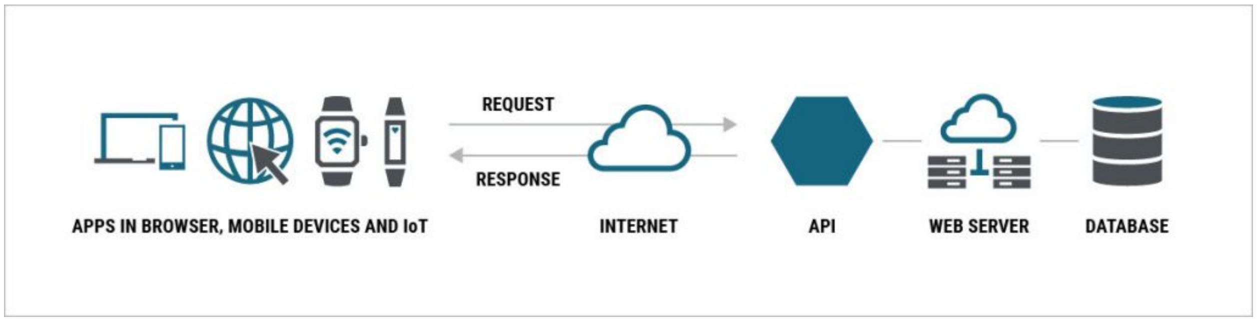
How to send/receive(fetch) data to and from remote location using React js?

In React JS we can send data to remote location as well as we can fetch data from remote location. Here remote location can be database, cache, server etc. After data is fetched then we do some processing and then render the data on the UI. All the data can be fetched through APIs and most of the time the format of fetched data is JSON. So here we need to understand two things.

- API (full form is application programming interface)
- JSON (full form is JavaScript Object Notation)

What is an API?

An application programming interface (API) is code that enables two software programs to communicate. An API defines how a developer should request services from other application, and fetch data and use if for different purpose. API allows two unrelated applications to “talk” to each other.

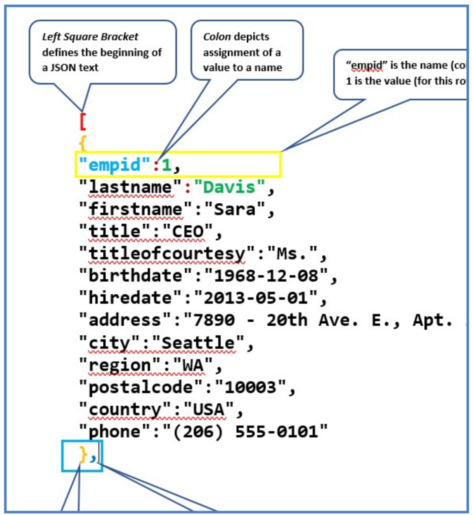


When you buy ticket for movie online from some app or website, it is API which makes it possible for you, when you send payment from paytm, gpay etc, it is API which makes payment between different banks. Using Web APIs requires the use of HTTP request methods. there are different HTTP request methods like GET, POST, PUT, DELETE, etc. Let’s have a brief description of the mentioned HTTP request.

- **GET:** Used to request data from an endpoint
- **POST:** Sends data to an endpoint
- **DELETE:** Remove data from an endpoint.
- **PUT**: Update a record or data value at an endpoint.

What Is JSON?

JSON stands for **JavaScript Object Notation** and is a way of representing data that looks like **JavaScript objects**.


Left Square Bracket defines the beginning of a JSON text
Colon depicts assignment of a value to a name
“empid” is the name (co
1 is the value (for this ro

```
[
{
"empid":1,
"lastname":"Davis",
"firstname":"Sara",
"title":"CEO",
"titleofcourtesy":"Ms.",
"birthdate":"1968-12-08",
"hiredate":"2013-05-01",
"address":"7890 - 20th Ave. E., Apt.
"city":"Seattle",
"region":"WA",
"postalcode":"10003",
"country":"USA",
"phone":"(206) 555-0101"
},
```

This is fairly easy to understand as data is stored as key/value pairs. here we can see the key on the left and the value on the right. Key means variable and value means value of the variable. JSON is everywhere in the modern web, mobile, and IoT applications. It’s easy to understand, lightweight, and works super well with applications written in any language. Data can be generated as well used in JSON in any programming languages. This makes APIs developer-friendly, highly scalable, and platform-independent.

Different objects would have a same key but different value. For example each object has empid, lastname and firstname key, but value will be different for each object.

How to call API's in React (how to get data from api)

You can use APIs in a React application in a variety of ways, two of the most popular approaches: axios (a promise-based HTTP client) and Fetch API(function) (a browser in-built web API)

How to Consume/call/execute APIs Using the Fetch function(A.P.I.)?

The Fetch API is a JavaScript built-in method for retrieving resources (mostly JSON) from a server or an API endpoint. It's built-in, so you don't need to install any dependencies or packages.

The fetch() method requires a compulsory argument, which is the path or URL (API) of the resource you want to fetch. Then it returns a Promise so you can handle success or failure using the **then() and catch() methods.**

A basic fetch request is very simple to write and looks like the below code. We are simply fetching data from a URL that returns data as JSON and then display it on screen:

```
import ReactDOM from "react-dom/client";
import React, { useState, useEffect } from 'react';

function App(){
  var [posts, setPosts] = useState([]);  //posts is state array variable
  useEffect(() => {
     fetch('https://jsonplaceholder.typicode.com/posts?_limit=10')
     .then((response) => response.json()) //convert string response into json
     .then((data) => {
           console.log(data);
           setPosts(data);
        })
        .catch((error) => {
           console.log(error.message);
        });
  }, []);
  const deletePost = async (id) => {
   await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`, {
      method: 'DELETE',
   }).then((response) => {
      if (response.status === 200) {
         setPosts(
            posts.filter((post) => {
               return post.id !== id;
            })
         );
      } else {
         return;
      }
   });
```

```
      };
        return (
          <div className="container">
            <div className="row">
             {posts.map((item) => {
                return (
                   <div className="col-12" key={item.id}>
                      <h2>{item.title}</h2>
                      <p>{item.body}</p>
                      <a className="btn btn-danger" onClick={() =>
deletePost(item.id)}>Delete</a>
                      </div>
                );
             })}
             </div>
          </div>
          );
      }
      const root = ReactDOM.createRoot(document.getElementById('root'));
      root.render(<App />);
```

How to Perform a POST Request in React with Fetch API

You can use the HTTP POST method to send data to an endpoint. It works similarly to the GET request, the main difference being that you need to add the method and two additional parameters to the optional object:

The major parameters are the body and header.

The body holds the data we want to pass into the API, which we must first stringify because we are sending data to a web server. The header tells us the type of data, which is always the same when consuming REST API's. We also set the state to hold the new data and distribute the remaining data into the array.

```
import React, { useState, useEffect } from 'react';
import ReactDOM from "react-dom/client";
const App = () => {
const [title, setTitle] = useState('');
const [body, setBody] = useState('');
const [posts, setPosts] = useState([]);
const addPosts = async (title, body) => {
  await fetch('https://jsonplaceholder.typicode.com/posts', {
    method: 'POST',
    body: JSON.stringify({
      title: title,
      body: body,
      userId: Math.random().toString(36).slice(2),
    }),
    headers: {
      'Content-type': 'application/json; charset=UTF-8',
      },
   })
      .then((response) => response.json())
      .then((data) => {
         setPosts((posts) => [data, ...posts]);
         setTitle('');
         setBody('');
         console.log(data);
      })
      .catch((err) => {
         console.log(err.message);
      });
};

const handleSubmit = (e) => {
   e.preventDefault();
   addPosts(title, body);
};

return (
   <div className="container">
      <div className="row mt-3">
      <div className="col-12">
         <div className='card shadow'>
            <div className='card-header'>
               <h3>Add Post</h3>
            </div>
            <div className='card-body'>
               <form onSubmit={handleSubmit}>
                  <div className='mb-3'>
                  <b>Title</b>
                     <input placeholder='title' type="text" className="form-control" value={title} onChange={(e) =>
setTitle(e.target.value)} />
                  </div>
                  <div className='mb-3'>
                     <b>Message</b>
                     <textarea name="" className="form-control" id="" cols="10" rows="8"
                     value={body} onChange={(e) => setBody(e.target.value)}></textarea>
                  </div>
                  <button type="submit" className='btn btn-primary'>Add Post</button>
               </form>
            </div>
         </div>
      </div>
      </div>
   </div>
);
};
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```