# Use Effect

One of the main benefits of lifecycle methods in class components is that they allow for us to program many side effects. UseEffect simply lets us apply any side effect that we'd like to our functional components. A typical usage of side effects could be making network requests to another service, logging analytics, or updating DOM elements manually.

When using the useEffect hook, we're telling react that we want to do something immediately after rendering, so once we do render (or re-render) a component, react will call the function(s) within our useEffect and invoke it!

## Implementing useEffect:

If we take our network calls from the `API calls` module and convert them to using hooks, we get:

```
import React, { useState, useEffect } from "react";

export const Reddit = props => {
  const [allPosts, setAllPosts] = useState(startingArrays);
  const [searchTerm, setSearchTerm] = useState("");
  const [posts, setPosts] = useState(startingArrays);


  // instead of using promises and `.then()s`, we can use
  // asynchronous programming with async await.
  const callWithFetch = async () => {
    const response = await fetch(`r/${props.subreddit}.json`);
    const json = await response.json();
    const allPosts = json.data.children.map(obj => obj.data);

    setAllPosts(allPosts);
    setPosts(allPosts);
  };

  // Use effect allows for us to program side effects with our
  // functional componennts. This way we can kick off a side
  // effect like a network call. What should be noted is that
  // this should be treated as if it were an equivalent to both
  // component did mount, and component did update. Basically,
  // it's a question of whether or not we called render.
  useEffect(() => {
```

```
27      if (allPosts.length === 0) callWithFetch();
28    });
29
30    const handleSearch = event => {
31      setSearchTerm(event.target.value);
32      setPosts(allPosts.filter(post =>
  post.title.includes(event.target.value)));
33    };
34
35    return (
36      <div>
37        <h1>r/{props.subreddit}</h1>
38        <div> FILTER THE CUTENESS </div>
39        <input type="text" value={searchTerm} onChange={handleSearch} />
40        <ul>
41          {posts.map(post => (
42            <li key={post.id}>
43              <br />
44              <img src={post.thumbnail} />
45              {post.title}
46            </li>
47          ))}
48        </ul>
49        ;
50      </div>
51    );
52  };
53
```

Take note of the useEffect on line 26. Because useEffect gets kicked off after every render, we want to write some logic to make sure that we don't make the network calls, update our state, re-render, call useEffect, make another network call, update our state, re-render, call useEffect, make another network call, and so on infinitely. With a quick check of our state variable, we can quickly make sure we only call our network calls once.

## Separation of Concerns

While you could do everything in one single useEffect, to keep your code clean, you can use as many useEffects as you like! Suppose that you'd want to set the title of the page based upon the number of elements that were filtered by the handleSearch function. We could use the same useEffect, but making a network call has almost nothing to do with setting our page's title, so we should use a different useEffect:

```
1  import React, { useState, useEffect } from "react";
```

```

export const Reddit = props => {
  const [allPosts, setAllPosts] = useState(startingArrays);
  const [searchTerm, setSearchTerm] = useState("");
  const [posts, setPosts] = useState(startingArrays);


  // instead of using promises and `.then()s`, we can use
  // asynchronous programming with async await.
  const callWithFetch = async () => {
    const response = await fetch(`r/${props.subreddit}.json`);
    const json = await response.json();
    const allPosts = json.data.children.map(obj => obj.data);

    setAllPosts(allPosts);
    setPosts(allPosts);
  };

  // Use effect allows for us to program side effects with our
  // functional componennts. This way we can kick off a side
  // effect like a network call. What should be noted is that
  // this should be treated as if it were an equivalent to both
  // component did mount, and component did update. Basically,
  // it's a question of whether or not we called render.
  useEffect(() => {
    if (allPosts.length === 0) callWithFetch();
  });

  // UPDATE THE TITLE!
  // This code will get run after every update and update
  // the title of the page/tab
  useEffect(() => {
    document.title = `Showing ${posts.length} posts`
  })

  const handleSearch = event => {
    setSearchTerm(event.target.value);
    setPosts(allPosts.filter(post =>
post.title.includes(event.target.value)));
  };

  return (
    <div>
      <h1>r/{props.subreddit}</h1>
      <div> FILTER THE CUTENESS </div>
      <input type="text" value={searchTerm} onChange={handleSearch} />
      <ul>
        {posts.map(post => (
          <li key={post.id}>
            <br />
            <img src={post.thumbnail} />
            {post.title}
          </li>
        ))}
      </ul>
      ;
    </div>
  );
};
```

Separation of concerns is not necessary, but it does make finding and updating code significantly easier.