Școala
informală
de IT

React Lifecycle & Hooks

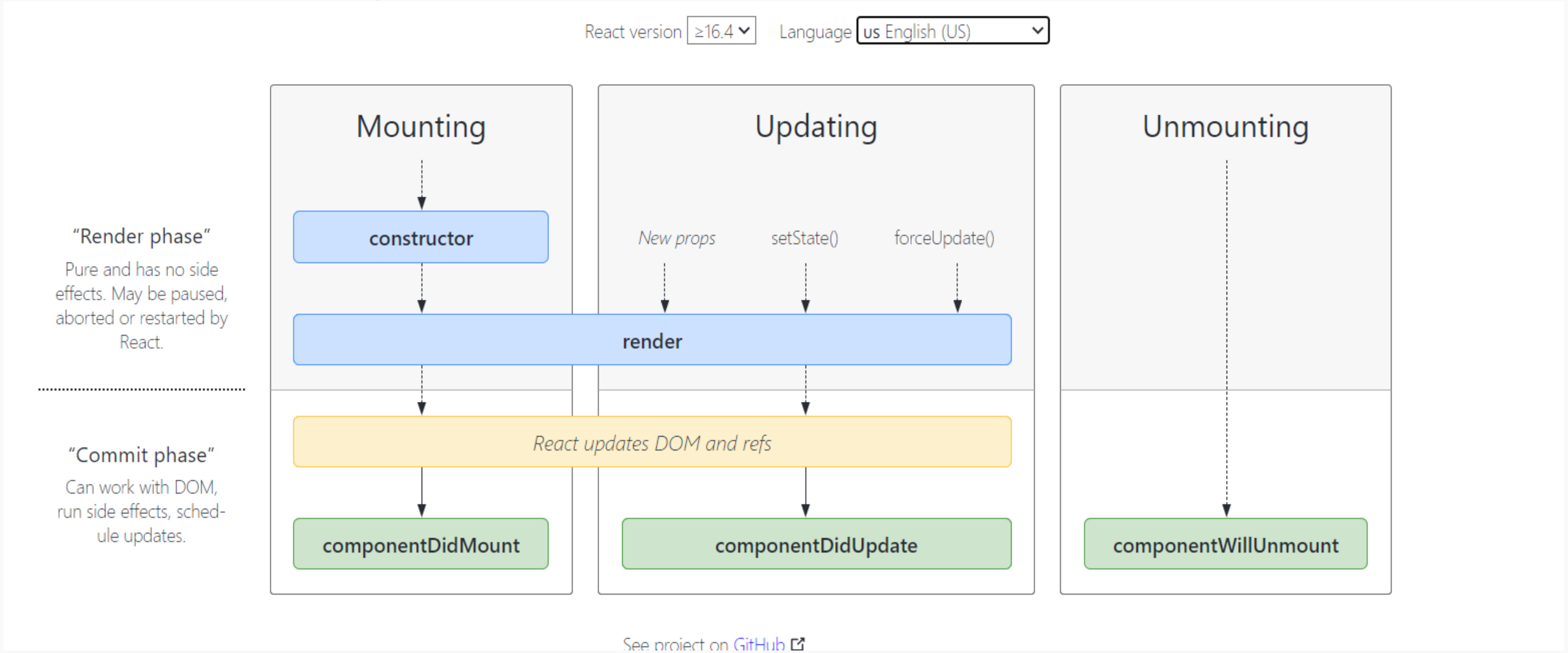
Școala
informală
de IT

React Lifecycle

- Mounting (adding nodes to the DOM)
- Updating (altering existing nodes in the DOM)
- Unmounting (removing nodes from the DOM)
- Error handling (verifying that your code works and is bug-free)

- component’s birth, growth, and death
- Error handling is like an annual physical.

React Lifecycle

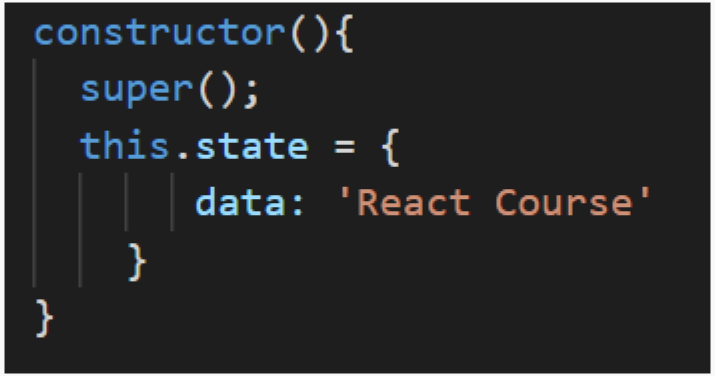


Școala informală de IT

Mounting phase

Contructor

- the very first method called as the component is “brought to life.”
- method used to initialize an object's state in a class
- automatically called during the creation of an object in a class
- in a React component is called before the component is mounted
- call **super(props) or super()** method before any other statement, this will call the constructor from React.Component parent
- Initialize the state in the constructor



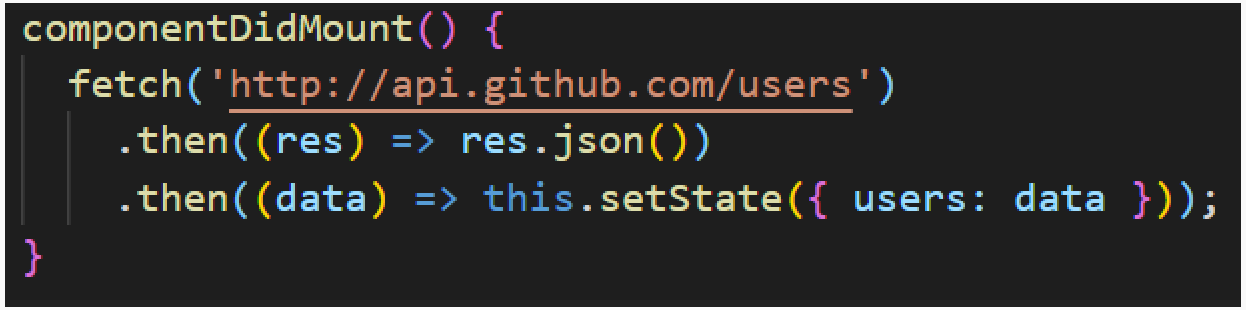
```
constructor(){
  super();
  this.state = {
        data: 'React Course'
    }
}
```

Render method

- the only required method from the class component
- responsible for describing the view to be rendered to the browser window
- render() is not callable by the user
- called by React at various app stages:
  - *when the React component is first instantiated*
  - *when there is a new update to the component state*
- Reconciliation Algorithm – use key prop for creating unique parts in the component
- To intelligently determine what needs to be rendered on every call, React compares the current state of the virtual DOM and the real one and only makes changes to the physical DOM where it recognizes that the UI has been updated.

ComponentDidMount

- Is called after all the elements of the page is rendered correctly
- **componentDidMount()** method is the perfect place, where we can call the **setState()** method to change the state of our application
- fetch any data from an API then API call should be placed in this lifecycle method
- calling the **setState()** method will automatically invoke the **render()** method and the component will be updated based on the new state



```
componentDidMount() {
  fetch('http://api.github.com/users')
    .then((res) => res.json())
    .then((data) => this.setState({ users: data }));
}
```

Școala
informală
de IT

Updating phase

Updating phase – ComponentDidUpdate

- Whenever a change is made to state or props the component is rerendered so the **render()** method is invoked
- is invoked immediately after updating occurs. This method is not called for the initial render.
- good place to do API requests as long as you compare the current props to previous props

```
componentDidUpdate(prevProps, prevState, snapshot)
```

React has a **rarely used** lifecycle method `getSnapshotBeforeUpdate` where you get the opportunity to look at your current DOM right before it changes.
The return value of this method is the third parameter of `componentDidUpdate`.

```
getSnapshotBeforeUpdate(prevProps, prevState) {
    return "value from snapshot";
}
```

Școala
informală
de IT

Unmounting phase

Unmounting phase – ComponentWillUnmount

- when a component is removed from the DOM – (e.g. user interaction – click event)
- Only one buil-in method that gets called on this phase
- **componentWillUnmount()** method is called when the component is about to be deleted from the DOM

Școala informală de IT

React Hooks

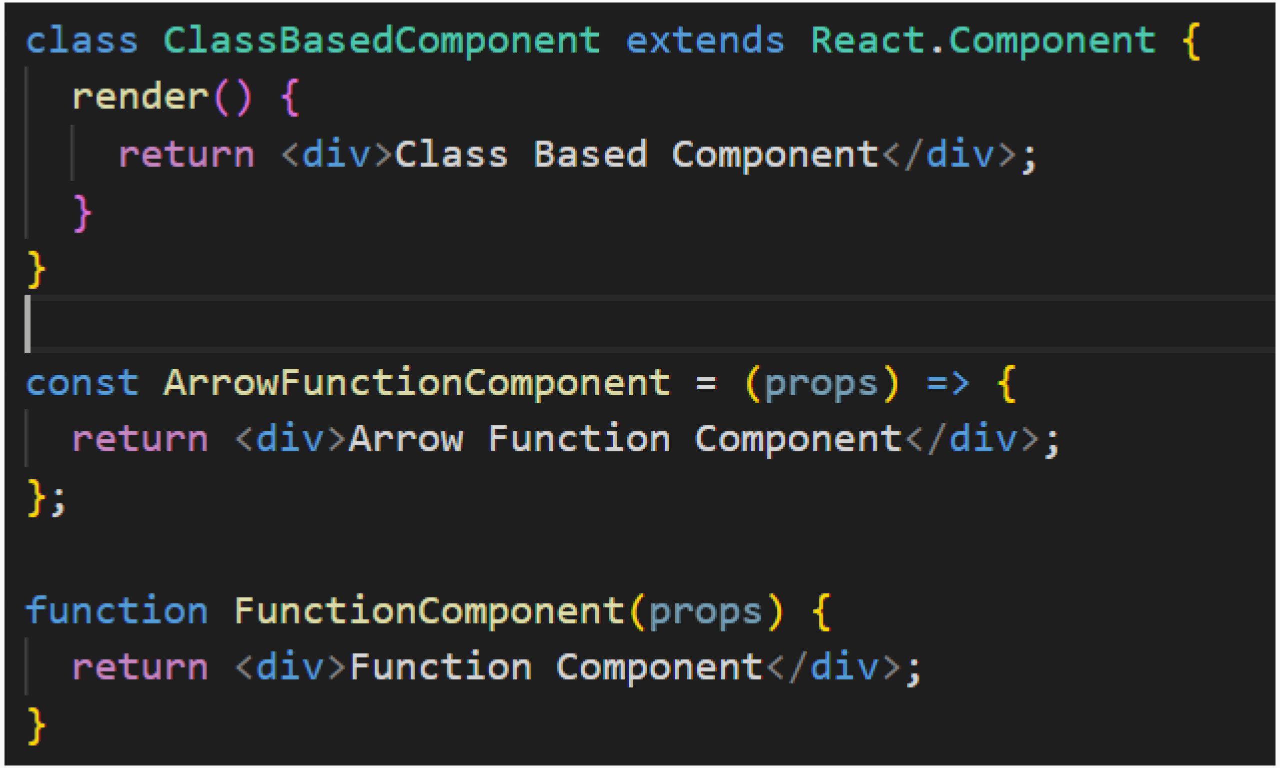
Hooks

- Introduced in feb 2019, in React 16.8
- Changes the way that we are writing components
- The community is divided(class based vs function component)
- Class based component – the only way to use internal state ?!?
- Hooks are introduces based on the community feedback
  - Not so easy to use lifecycle methods
  - New way introduced – arrow function components and Hooks

Hooks –> new way to use the functionalities from class based components in functional components

We can use Hooks only in react 16.8 or higher AND ONLY IN FUNCTIONAL COMPONENTS

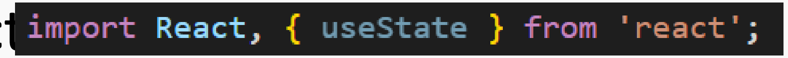
Class Component vs Functional Component



```
class ClassBasedComponent extends React.Component {
  render() {
    return <div>Class Based Component</div>;
  }
}

const ArrowFunctionComponent = (props) => {
  return <div>Arrow Function Component</div>;
};

function FunctionComponent(props) {
  return <div>Function Component</div>;
}
```

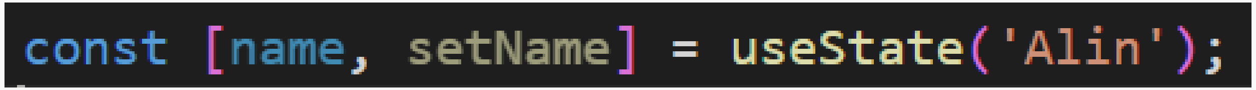
useState hook

- Import **useState()** hook from react



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

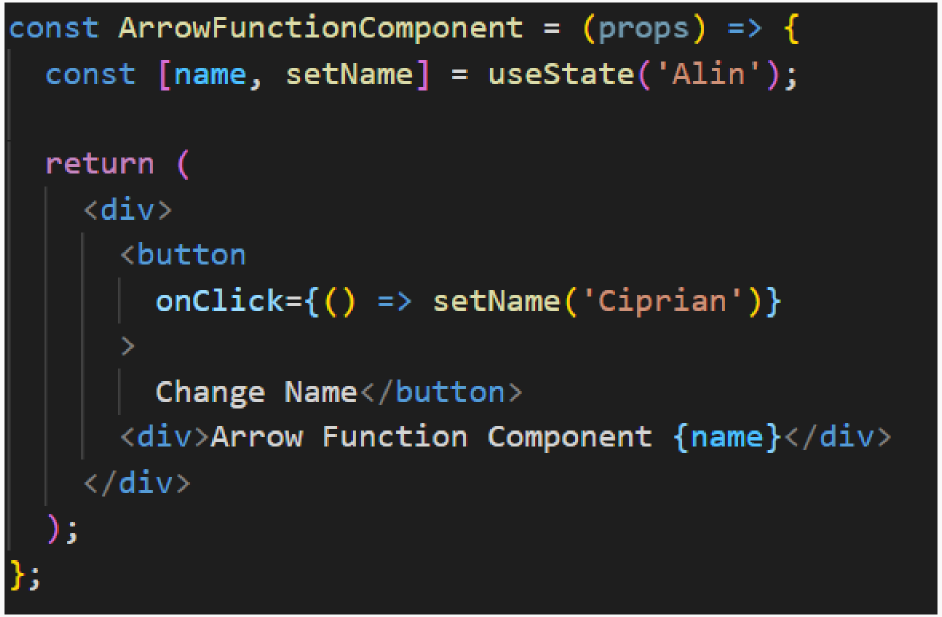
- Allows functional components to have access to internal state
- useState() – give an array with 2 values
- Uses array destructuring
- https://www.freecodecamp.org/news/array-and-object-destructuring-in-javascript/



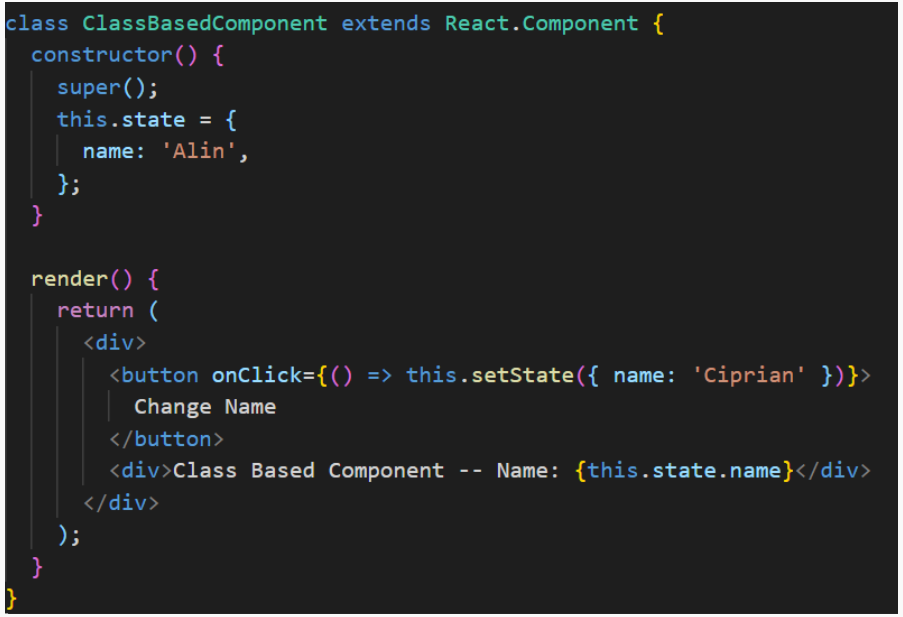
```
const [name, setName] = useState('Alin');
```

- **name** – a property from state
- **setName** – a function that allow us to set the **name** property
- pass to **useState()** the **initial** value

useState hook

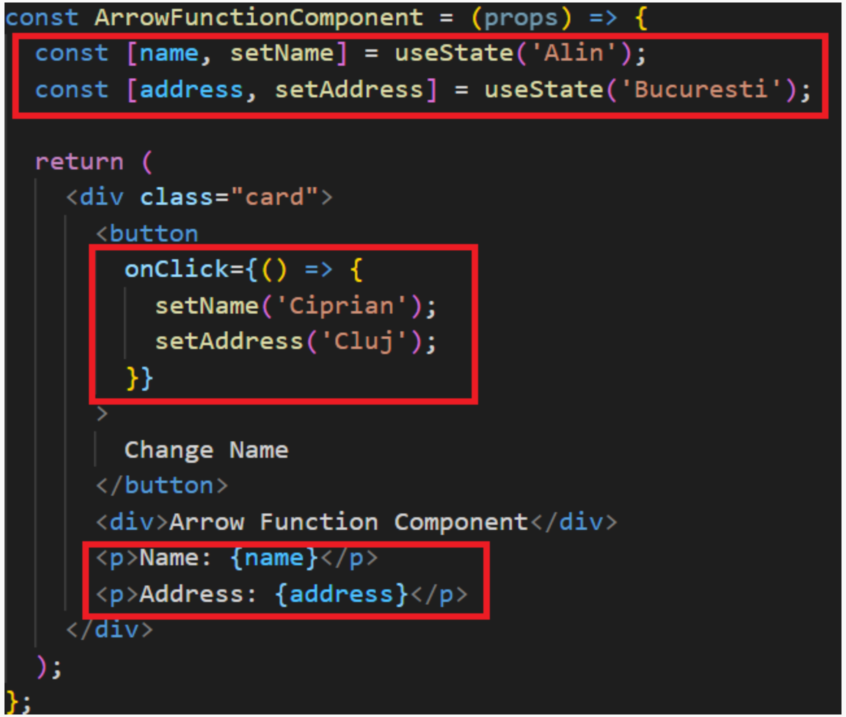


```
const ArrowFunctionComponent = (props) => {
  const [name, setName] = useState('Alin');

  return (
    <div>
      <button
        onClick={() => setName('Ciprian')}
      >
        Change Name</button>
      <div>Arrow Function Component {name}</div>
    </div>
  );
};
```

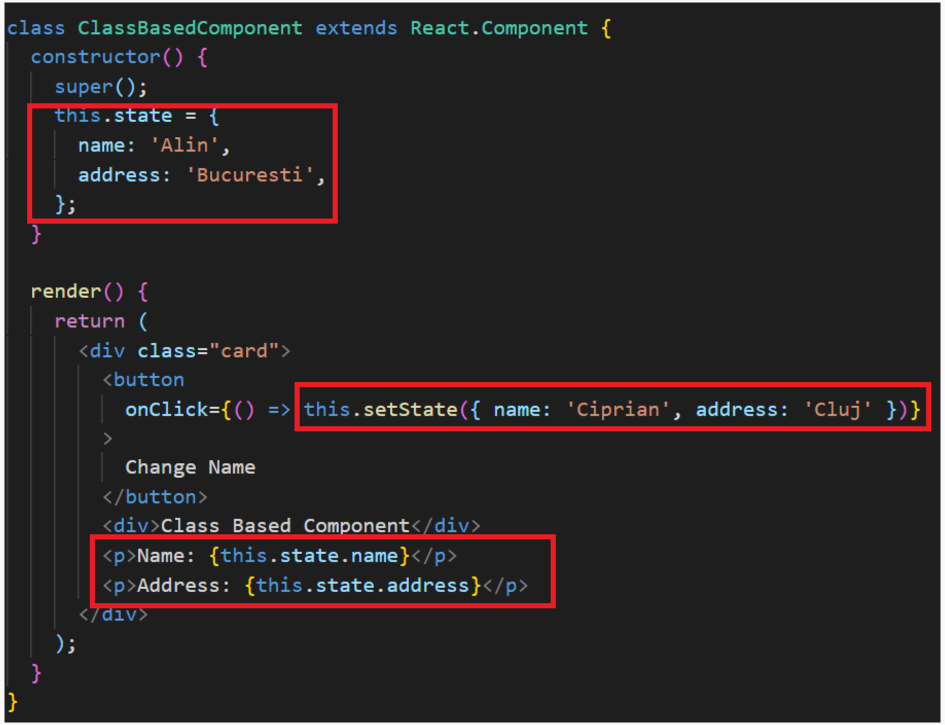


```
class ClassBasedComponent extends React.Component {
  constructor() {
    super();
    this.state = {
      name: 'Alin',
    };
  }

  render() {
    return (
      <div>
        <button onClick={() => this.setState({ name: 'Ciprian' })}>
          Change Name
        </button>
        <div>Class Based Component -- Name: {this.state.name}</div>
      </div>
    );
  }
}
```

Adding new properties to state



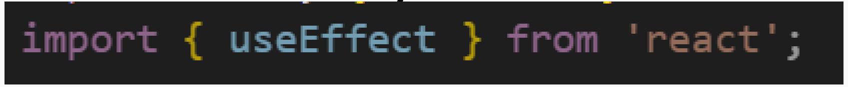
```
const ArrowFunctionComponent = (props) => {
  const [name, setName] = useState('Alin');
  const [address, setAddress] = useState('Bucuresti');

  return (
    <div class="card">
      <button
        onClick={() => {
          setName('Ciprian');
          setAddress('Cluj');
        }}
      >
        Change Name
      </button>
      <div>Arrow Function Component</div>
      <p>Name: {name}</p>
      <p>Address: {address}</p>
    </div>
  );
};
```



```
class ClassBasedComponent extends React.Component {
  constructor() {
    super();
    this.state = {
      name: 'Alin',
      address: 'Bucuresti',
    };
  }

  render() {
    return (
      <div class="card">
        <button
          onClick={() => this.setState({ name: 'Ciprian', address: 'Cluj' })}
        >
          Change Name
        </button>
        <div>Class Based Component</div>
        <p>Name: {this.state.name}</p>
        <p>Address: {this.state.address}</p>
      </div>
    );
  }
}
```

useEffect hook

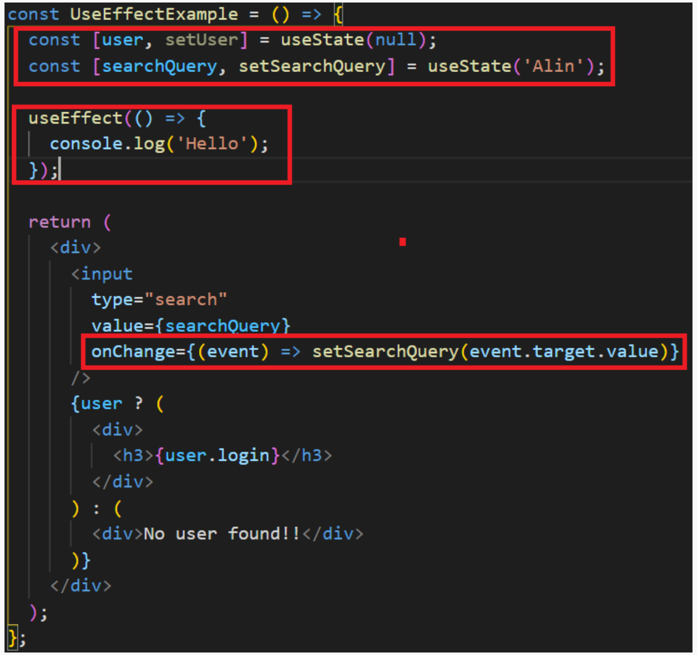
- Gives the ability to fire side effects inside functional components



```
import { useEffect } from 'react';
```

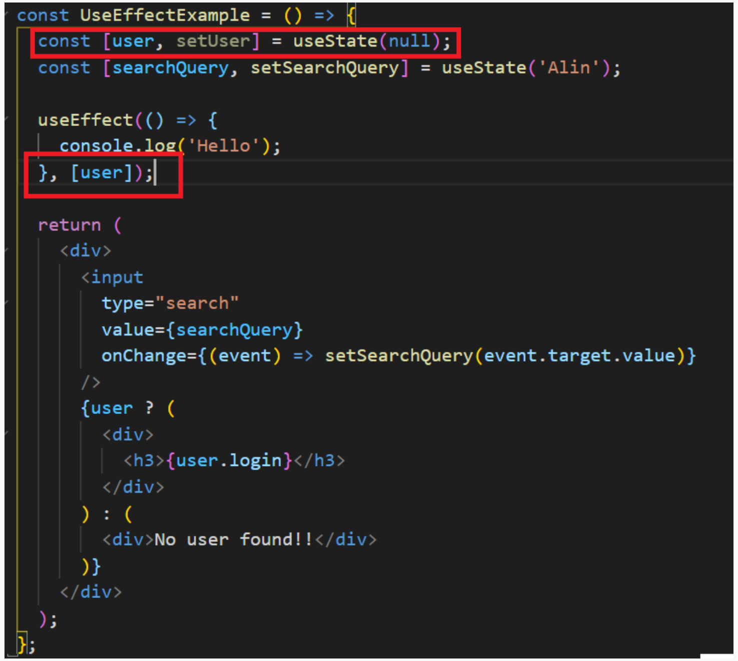
- Is a function that is called every time a component changes
  - Component is mounted
  - The component is rendered(rerendered)
  - Any time a state is changed using **useState()** hook
- **useEffect()** could be configured to mimic only componentDidMount() we can pass to it an array of properties and when this properties are changed the **useEffect()** method is called(by default is call for any change in the component)

useEffect hook – array of properties param



```
const UseEffectExample = () => {
  const [user, setUser] = useState(null);
  const [searchQuery, setSearchQuery] = useState('Alin');

  useEffect(() => {
    console.log('Hello');
  });

  return (
    <div>
      <input
        type="search"
        value={searchQuery}
        onChange={(event) => setSearchQuery(event.target.value)}
      />
      {user ? (
        <div>
          <h3>{user.login}</h3>
        </div>
      ) : (
        <div>No user found!!</div>
      )}
    </div>
  );
};
```

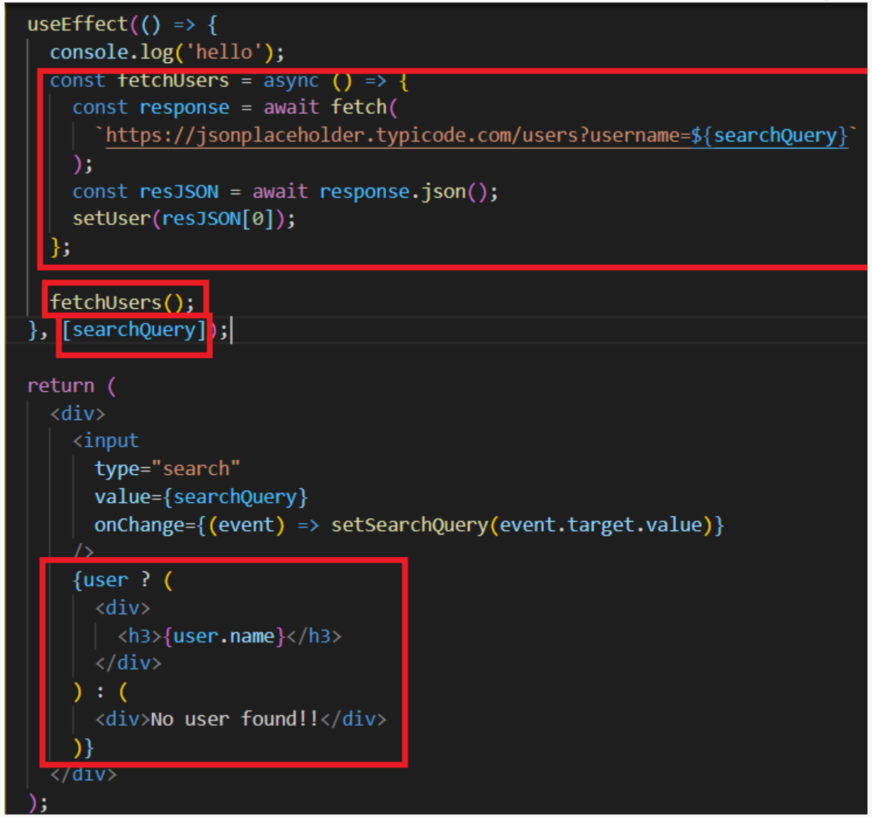
- **useEffect()** will be called when any state is changed



```
const UseEffectExample = () => {
  const [user, setUser] = useState(null);
  const [searchQuery, setSearchQuery] = useState('Alin');

  useEffect(() => {
    console.log('Hello');
  }, [user]);

  return (
    <div>
      <input
        type="search"
        value={searchQuery}
        onChange={(event) => setSearchQuery(event.target.value)}
      />
      {user ? (
        <div>
          <h3>{user.login}</h3>
        </div>
      ) : (
        <div>No user found!!</div>
      )}
    </div>
  );
};
```

- **useEffect()** will be called **only** when the user state is changed

useEffect hook – use async await



```jsx
useEffect(() => {
  console.log('hello');
  const fetchUsers = async () => {
    const response = await fetch(
      `https://jsonplaceholder.typicode.com/users?username=${searchQuery}`
    );
    const resJSON = await response.json();
    setUser(resJSON[0]);
  };

  fetchUsers();
}, [searchQuery]);

return (
  <div>
    <input
      type="search"
      value={searchQuery}
      onChange={(event) => setSearchQuery(event.target.value)}
    />
    {user ? (
      <div>
        <h3>{user.name}</h3>
      </div>
    ) : (
      <div>No user found!!</div>
    )}
  </div>
);
```

Resources

- https://blog.logrocket.com/react-lifecycle-methods-tutorial-examples/
- https://reactjs.org/docs/hooks-intro.html#motivation