


Тема занятия:

React:

lifecycle,
useEffect,
re-rendering of components


useEffect



```
useEffect(() => {  
  ...  
}, [])
```

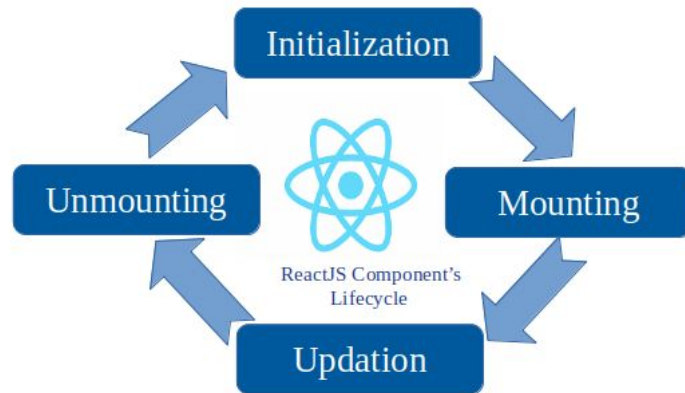


```
componentDidMount() {}  
componentDidUpdate() {}  
componentWillUnmount() {}
```



3 этапа жизни компонента:

- визуализация компонента - **Mounting**(монтирование);
- обновление компонента - **Updating**(обновление);
- удаление компонента из DOM - **Unmounting**(размонтирование).



Что такое `useEffect`?

`useEffect` — это хук, который можно использовать для замены в определенные моменты жизненного цикла компонента.

`useEffect` принимает два параметра.

Первый аргумент — это функция обратного вызова

Второй аргумент – массив зависимостей. Второй аргумент является необязательным.

```
useEffect(setup, dependencies?)
```

Использование **useEffect** сразу после создания элемента (Mounting)

Если мы передаем второй аргумент в виде пустого массива, побочный эффект в функции обратного вызова сработает только один раз при первой визуализации компонента.

```
function MyComponent() {  
  useEffect(() => {  
    // This side effect will only run once, after the first render  
  }, [])  
}
```

Вариантом использования для этого может быть получение данных из API

Использование `useEffect` для при обновлении (Updating)

- При каждом рендере компонента

```
function MyComponent() {  
  useEffect(() => {  
    // The side effect will run after every render  
  })  
}
```

- при изменении значений зависимостей, переданны в массиве

```
import { useEffect, useState } from 'react'  
function MyComponent({ prop }) {  
  const [state, setState] = useState('')  
  useEffect(() => {  
    // the side effect will only run when the props or state changed  
  }, [prop, state])  
}
```

Примером использования для этого может быть функция поиска.

Использование **useEffect** для при размонтировании (**Unmounting**)

Это функция очистки, которая позволяет нам остановить побочные эффекты непосредственно перед размонтированием компонента.

```
function MyComponent() {  
  useEffect(() => {  
    // this side effect will run after every render  
    return () => {  
      // this side effect will run before the component is unmounted  
    }  
  })  
}
```

Пример использования функции очистки

```
import { useEffect } from "react"

const Modal = ({ modalContent, closeModal }) => {
  useEffect(() => {
    let timeout = setTimeout(() => closeModal(), 3000)

    return () => clearTimeout(timeout)
  })
  return (
    <div className="modal">
      <p>{modalContent}</p>
    </div>
  )
}

export default Modal
```




re-rendering of
components

Повторный рендеринг компонентов в React — это процесс обновления визуального представления компонента в результате изменения его состояния или пропсов. React воссоздает виртуальное дерево компонентов, сравнивает его с предыдущим состоянием и определяет минимальное количество изменений, необходимых для обновления пользовательского интерфейса.



Случаи повторного рендеринга

1. Изменение состояния (useState)

Когда вызывается функция изменения состояния, компонент перерендеривается с новым состоянием

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

function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1); // Изменение состояния
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

Случаи повторного рендеринга

2.Изменение пропсов

Когда компонент получает новые пропсы, он перерендеривается с новыми значениями пропсов

```
import React from 'react';

function Greeting({ name }) {
  return <p>Hello, {name}!</p>;
}

// Повторный рендеринг при изменении пропсов
<Greeting name="Alice" />
<Greeting name="Bob" />
```