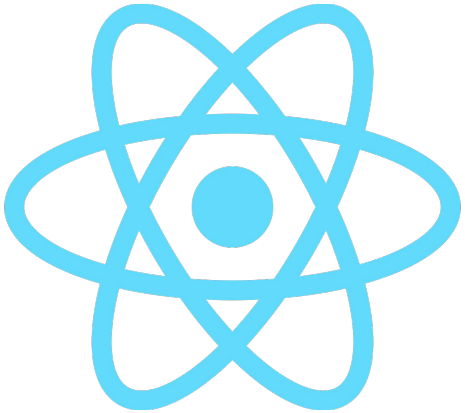

Тема занятия:

React:

styles,
controlled and
uncontrolled components,
map

Styling Components

1 подход

Обычный CSS

Написание стилей происходит в отдельных CSS файлах и затем они импортируются в ваши компоненты

```css
/* styles.css */
.myComponent {
  color: blue;
  font-size: 16px;
}
```

css файл

```jsx
// MyComponent.js
import React from 'react';
import './styles.css'; // Импорт стилей

const MyComponent = () => {
  return <div className="myComponent">Пример компонента</div>;
};

export default MyComponent;
```

react компонент

2 подход

Inline Styles:

Стили указываются напрямую внутри JSX элемента.

```jsx
const MyComponent = () => {
  return <div style={{ color: 'blue', fontSize: '16px' }}>Пример компонента</div>;
};
```

С помощью создания отдельной переменной

```jsx
const MyComponent = () => {
  const style = {
    color: 'blue',
    fontSize: '16px',
  };

  return <div style={style}>Пример компонента</div>;
};
```

3 подход

CSS-in-JS библиотеки, такой как **emotion**

Она предоставляет множество возможностей для создания стилей, включая локальную область видимости и использование динамических стилей.

Установка

Для начала, установите **emotion** в вашем проекте с помощью npm

```
npm install --save @emotion/react
```

```
npm install --save @emotion/styled
```

3 подход - emotion

Основная концепция

Emotion позволяет создавать стилизованные компоненты с использованием синтаксиса тегов.

```
import styled from '@emotion/styled';

export const StyledButton = styled.button`
  color: white;
  background-color: #3498db;
  border: none;
  border-radius: 5px;
  cursor: pointer;
`;
```

styles.js (файл с стилями):

```
import { StyledButton} from './styles';

const MyComponent = () => {
  return (
    <div>
      <StyledButton onClick={()=>{}}>Click
me</StyledButton>
    </div>
  );
};

export default MyComponent;
```

MyComponent.js (компонент, использующий стили):

Возможности emotion

1. Вы можете передавать пропсы и использовать их для определения стилей.

```
import styled from
'@emotion/styled';

const StyledButton = styled.button`
  color: ${props => (props.primary ?
'white' : 'black')};
  background-color: ${props =>
(props.primary ? 'blue' :
'lightgray')};
  padding: 10px;
  }
`;
```

styles.js (файл с стилями):

```
const MyComponent = () => {
  return (
    <div>

      <StyledButton onClick={() =>{}}>Standart button
</StyledButton>

      <StyledButton primary onClick={() => {}}>Primary
button</StyledButton>

    </div>
  );
};
```

MyComponent.js (компонент, использующий стили):

Возможности emotion

2. Расширение стилей

Вы можете расширять стили.

```
export const BaseComponent = styled.div`
  background: red;
  font-size: 50px;
  margin: 20px;
`;

export const ExtendedComponent =
styled(BaseComponent)`
  color: blue;
`;
```

Возможности emotion

3. Использование css функции.

Вы можете задавать общие стили и затем переиспользовать их с помощью функции **css**

```
import { css } from '@emotion/react';
import styled from '@emotion/styled';


const commonStyles = css`
  padding: 10px;
  margin: 5px;
`;


const StyledButton = styled.button`
  ${commonStyles};
  color: white;
  background-color: #3498db;
  border: none;
`;
```

Возможности emotion

4. Глобальные стили

В emotion глобальные стили могут быть заданы с использованием компонента **Global**.

1 шаг. Создаём стили в файле **GlobalStyles.js**

```
import { css } from '@emotion/react';

export const globalStyles = css`
  body, html {
    margin: 0;
    padding: 0;
  }
`;
```

Возможности emotion

4. Глобальные стили

В emotion глобальные стили могут быть заданы с использованием компонента **Global**.

2 шаг. Используем глобальные стилей в нашем приложении, добавляя их с помощью компонента **Global** в **App.js**

```
import { Global } from '@emotion/react';
import { globalStyles } from './GlobalStyles';

const App = () => {
  return (
    <>
      <Global styles={globalStyles} />
       …
    </>
  );
};

export default App;
```

Controlled and uncontrolled components

В React компоненты могут быть разделены на две основные категории:

- контролируемые (controlled)
- неконтролируемые (uncontrolled).

Эти термины относятся к тому, как компонент управляет своим состоянием и данными.

Контролируемые компоненты

Контролируемый компонент - это компонент, который управляет своим состоянием с помощью React.

Любые изменения ввода пользователя или другие события приводят к обновлению состояния компонента через setState.

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

const ControlledComponent = () => {
  const [inputValue, setInputValue] = useState('');

  const handleChange = (e) => {
    setInputValue(e.target.value);
  };

  return (
    <input
      type="text"
      value={inputValue}
      onChange={handleChange}
    />
  );
};
```

Неконтролируемые компоненты

Неконтролируемый компонент - это компонент, в котором состояние не контролируется React.

Вместо этого, данные хранятся в DOM, и доступ к этим данным осуществляется напрямую через ссылки на DOM-элементы.

```jsx
import React, { useRef } from 'react';

const UncontrolledComponent = () => {
  const inputRef = useRef();

  const handleClick = () => {
    alert(`Input value: ${inputRef.current.value}`);
  };

  return (
    <>
      <input type="text" ref={inputRef} />
      <button onClick={handleClick}>Get Value</button>
    </>
  );
};
```

В этом примере **inputRef** представляет собой ссылку на DOM-элемент **<input>**. Значение не хранится в состоянии компонента; вместо этого, при необходимости вы можете получить доступ к значению напрямую через inputRef.current.value.

Когда выбирать между контролируемыми и неконтролируемыми компонентами:

- **Контролируемые компоненты**: Полезны, когда React должен полностью контролировать состояние компонента, особенно при работе с формами. Позволяют React легко управлять вводом и обновлять UI в ответ на изменения.

- **Неконтролируемые компоненты**: Могут быть удобными, когда вам нужно интегрироваться с кодом или библиотеками, которые управляют DOM напрямую. Они также могут уменьшить необходимость в использовании состояния и setState.