**Лекція 17. Застосування розширення JSX в React**

1. Особливості синтаксису React.

2. Основи мови JSX.

3. Змінні та атрибути тегів в JSX.

4. Робота з CSS, умовами, циклами та методами класу в JSX.

1. **Особливості синтаксису React**

При написанні коду React прийнято використовувати **const** замість **var**. Нагадую, що **const** - це константи, тобто змінні, тип та значення яких не можна змінити. Останнє не відноситься до *масивів та об'єктів* - у цьому випадку не можна буде змінити тип змінної, а змінювати/додавати/видаляти елементи масиву - можна буде.

Константи якраз зручні при роботі з масивами та об'єктами, їх використання не дає змінити тип змінної - наприклад, не дає затерти масив і записати туди щось інше, наприклад, рядок.

Будьте уважні - при використанні **const** на вас можуть чекати несподівані проблеми.

Наприклад, проблеми можуть виникнути при роботі з **if** , тому що змінні з **const**, як і **let** , видно тільки всередині фігурних дужок (не так, як var) і не видно зовні.

Дивіться приклад:

if (тут умова) {

const name **=** 'Коля';

} else {

const name **=** 'Вася';

}

alert(name); //буде undefined

Якщо спробувати визначити константу над ІФ - це також призведе до помилки, так як при визначенні константи обов'язково має бути вказано її значення:

const name; //поверне помилку

if (тут умова) {

name **=** 'Коля';

} else {

name **=** 'Вася';

}

alert(name);

Для того, щоб виправити проблему замість **const** напишемо **let** , причому **let** обов'язково над іфом, інакше через область видимості змінні не будуть видно зовні:

let name; //визначаємо змінну let ззовні If

if (тут условие) {

name **=** 'Коля';

} else {

name **=** 'Вася';

}

alert(name); //все працює!

**Компонентний підхід**

Нехай ми маємо сайт. На цьому сайті ми можемо виділити деякі блоки: хедер, контент, сайдбар, футер тощо. Кожен блок можна розділити на дрібніші підблоки. Наприклад у хедері зазвичай можна назвати логотип, меню, блок контактів тощо.

У React кожен такий блок називається **компонентом** . Кожен компонент може містити в собі дрібніші компоненти, ті в свою чергу ще дрібніші і таке інше.

Кожному компоненту React відповідає ООП клас:

class Ім’яКласу {

}

Щоб бути компонентом, цей клас обов'язково має успадковувати від класу **React.Component** :

class Ім’яКласу extends React.Component {

}

Клас **React.Component** вбудований у React і доступний після підключення фреймворку. Просто успадкуєте від нього - і все, решта не повинна хвилювати вас.

**Підключення компонентів**

Нехай у нас є, наприклад, компонент під назвою **Test** :

class Test extends React.Component {

}

Кожному компоненту відповідає власний тег. За допомогою цього тега можна вставити сторінку **результат роботи компонента** .

Тег має таку ж назву, як ім'я класу компонента. Тобто: якщо, наприклад, ми компонент **Test** , йому відповідає тег **<Test />** .

Цей тег пишеться з великої літери і відразу ж закривається за допомогою зворотного сліша (закривати обов'язково!).

Нехай у нас є тег **header** :

<header></header>

Давайте в цей **header** вставимо результат роботи нашого компонента **Test** :

<header><Test /></header>

**Результат роботи компонента**

Отже, ми з'ясували, що там, де напишемо тег **<Test />** , виведеться результат роботи нашого компонента, заданого класом **Test** .

Нагадую, що сам компонент є класом. Всередині цього класу ви можете написати будь-які методи. Але є один метод, який має особливе значення: його назва **render()** . Те, що поверне метод **render()** і буде результатом роботи будь-якого компонента:

class Test extends React.Component {

render() {

return тут результат роботи компонента;

}

}

**2. Основи мови JSX**

У React вбудовано спеціальну мову під назвою JSX. Суть цієї мови така: *HTML теги є коректним JavaScript-кодом* і ми можемо просто писати їх прямо в скрипті, не беручи в лапки.

Наприклад, запишемо в змінну **text** шматочок HTML:

const text **=** **<**div**>**текст**</**div**>**;

Ось він прямо так і записується – без лапок. Докладніше про це ми говоритимемо у наступному уроці.

**JSX у компонентах**

Давайте зробимо так, щоб результатом роботи компонента **Test** було **<div>текст</div>** :

class Test extends React.Component {

render() {

return **<**div**>**текст**</**div**>**;

}

}

У результаті замість цього коду:

<header><Test /></header>

Отримаємо таке:

<header><div>текст</div></header>

**Основний компонент**

Ми вже з'ясували, що на сторінці є компоненти, у яких вкладені компоненти тощо. Проте є найголовніший компонент верхнього рівня - у ньому лежать решта компонентів.

Зазвичай цей компонент називають **App** (але не обов'язково):

class App extends React.Component {

}

Щоб вставити цей компонент на сторінку мало написати тег **<App />** . В цьому випадку необхідно вказати елемент DOM, всередину якого вставиться результат роботи компонента App.

Нехай у нас є блок **#content** :

<div id="content"></div>

В цьому випадку наступний код змусить вставити результат роботи компонента **App** в блок **#content** :

ReactDOM.render(

**<**App **/>**,

document.getElementById('content')

);

Давайте зберемо все разом:

<div id="content"></div>

class App extends React.Component {

render() {

return **<**div**>**текст**</**div**>**;

}

}

ReactDOM.render(

**<**App **/>**,

document.getElementById('content')

);

Результатом цього коду буде наступне:

<div id="content">

<div>текст</div>

</div>

**Особливості методу render**

Давайте тепер пограємось із JSX. Модифікуємо результат роботи методу **render**. Можна розбити наш div на кілька рядків:

class App extends React.Component {

render() {

return **<**div**>**

текст

**</**div**>**;

}

}

Проте, просто знести **<div>** вниз не можна – це видасть помилку:

class App extends React.Component {

render() {

//цей код викличе помилку:

return

**<**div**>**

текст

**</**div**>**;

}

}

Для вирішення цієї проблеми можна поставити круглі дужки:

class App extends React.Component {

render() {

//Так працювати буде:

return (

**<**div**>**

текст

**</**div**>**

);

}

}

**Декілька тегів у return**

Результатом рендеренгу може бути кілька тегів:

class App extends React.Component {

render() {

// цей код викличе помилку:

return (

**<**p**>**текст 1**</**p**>**

**<**p**>**текст 2**</**p**>**

);

}

}

Декілька тегів обов'язково потрібно обернути в якийсь загальний тег. Давайте обернемо наші абзаци в загальний тег **div** :

class App extends React.Component {

render() {

// Так працювати буде:

return (

**<**div**>**

**<**p**>**текст 1**</**p**>**

**<**p**>**текст 2**</**p**>**

**</**div**>**

);

}

}

1. **Змінні та атрибути тегів в JSX**

Як згадувалося раніше, шматочки HTML коду можна записувати в змінні. Давайте зробимо це всередині методу **render**: запишемо шматочок тексту в змінну **text**, а потім напишемо цю змінну після **return**:

class App extends React.Component {

render() {

const text **=** **<**div**>**текст**</**div**>**;

return text;

}

}

Результатом роботи цього коду буде наступне:

<div>

текст

</div>

**Вставка значень змінних**

Нехай у нас є змінна з текстом, наприклад, змінна з ім'ям **str**. Ми можемо вставити значення цієї змінної в будь-який шматочок HTML коду, написавши її у фігурних дужках ось так: **<div>{str}</div>**. У цьому випадку при рендерінгу замість змінної у фігурних дужках вставиться її значення:

class App extends React.Component {

render() {

const str **=** 'текст';

return (

**<**div**>**

{str}

**</**div**>**

);

}

}

Результатом роботи цього коду буде наступне:

<div>

текст

</div>

У змінній замість тексту може зберігатися шматочок HTML коду - це не має значення, все буде працювати:

class App extends React.Component {

render() {

const str **=** **<**p**>**текст**</**p**>**;

return (

**<**div**>**

{str}

**</**div**>**

);

}

}

Результатом роботи цього коду буде наступне:

<div>

<p>текст</p>

</div>

Зверніть увагу, що кілька тегів, що зберігаються в змінній, обов'язково потрібно обернути в якийсь загальний тег. Тобто ось так не працюватиме:

class App extends React.Component {

render() {

const list **=** **<**p**>**1**</**p**><**p**>**2**</**p**><**p**>**3**</**p**>**;

return (

**<**div**>**

{list}

**</**div**>**

);

}

}

А ось так буде, тому що абзаци обгорнуті в один **div** тег:

class App extends React.Component {

render() {

const list **=** **<**div**><**p**>**1**</**p**><**p**>**2**</**p**><**p**>**3**</**p**></**div**>**;

return (

**<**div**>**

{list}

**</**div**>**

);

}

}

**Виконання JavaScript**

Отже, всередині шматочків HTML коду можна робити вставки змінних. Нехай у нас є змінна **num**. Давайте вставимо в певне місце вміст цієї змінної:

class App extends React.Component {

render() {

const num **=** 3;

const text **=** **<**div**>**текст {num}**</**div**>**;

return text;

}

}

Результатом роботи цього коду буде наступне:

<div id="content">

<div>текст 3</div>

</div>

Усередині фігурних дужок можна не тільки вставляти змінні, але й довільний JavaScript код. Давайте, наприклад, у момент вставки додамо до змінної **num** одиницю:

class App extends React.Component {

render() {

const num **=** 3;

const text **=** **<**div**>**текст {num **+** 1}**</**div**>**;

return text;

}

}

Результатом роботи цього коду буде наступне:

<div id="content">

<div>текст 4</div>

</div>

Нехай у нас тепер є дві змінні: **num1** та **num2** . Давайте виведемо на екран суму цих змінних:

class App extends React.Component {

render() {

const num1 **=** 1;

const num2 **=** 2;

const text **=** **<**div**>**текст {num1 **+** num2}**</**div**>**;

return text;

}

}

Результатом роботи цього коду буде наступне:

<div id="content">

<div>текст 3</div>

</div>

**Робота з атрибутами тегів**

Вставку змінних можна робити не тільки в тексти тегів, але й атрибути. При цьому *лапки від атрибутів не ставляться* :

class App extends React.Component {

render() {

const str **=** 'elem';

const text **=** **<**div id**=**{str}**>**текст**</**div**>**;

return text;

}

}

Результатом роботи цього коду буде наступне:

<div id="content">

<div id="elem">текст</div>

</div>

Таке працює для всіх атрибутів, але є **виняток**. Замість атрибуту **class** слід писати атрибут **className**:

class App extends React.Component {

render() {

const str **=** 'elem';

const text **=** **<**div className**=**{str}**>**текст**</**div**>**;

return text;

}

}

Результатом роботи цього коду буде наступне:

<div id="content">

<div class="elem">текст</div>

</div>

Врахуйте, що навіть якщо ви пишите безпосередньо ім'я класу в атрибуті **class** , все одно слід писати **className** :

class App extends React.Component {

render() {

const text **=** **<**div className**=**"content"**>**текст**</**div**>**;

return text;

}

}

1. **Робота з CSS, умовами, циклами та методами класу в JSX**

За допомогою атрибутів можна також додавати стилі CSS елементам.

Для цього в атрибут **style** передається об'єкт із назвами CSS властивостей та їх значеннями.

Зробимо, для прикладу, наш div червоного кольору і розміром шрифту **30px** :

class App extends React.Component {

render() {

const css **=** {color: 'red', fontSize: '30px'};

const text **=** **<**div style**=**{css}**>**текст**</**div**>**;

return text;

}

}

Можна об'єкт написати прямо в атрибуті – у цьому випадку нам потрібні дві фігурні дужки: перша від JSX вставки, а друга – від об'єкта:

class App extends React.Component {

render() {

const text **=** **<**div style**=**{ {color: 'red', fontSize: '30px'} }**>**

текст

**</**div**>**;

return text;

}

}

У цьому об'єкті також можна використовувати змінні. Нехай, наприклад, у змінній **value** зберігається колір. Давайте властивість **color** встановимо значення з цієї змінної:

class App extends React.Component {

render() {

const value **=** 'red';

const text **=** **<**div style**=**{ {color: value} }**>**

текст

**</**div**>**;

return text;

}

}

**Умови**

При роботі зі шматочками HTML можна застосовувати умови **if** . Давайте зробимо так, щоб в залежності від вмісту змінної **showText** на екран виводиться один або інший текст:

class App extends React.Component {

render() {

const text;

const showText **=** true;

if (showText) {

text **=** **<**div**>**текст1**</**div**>**;

} else {

text **=** **<**div**>**текст2**</**div**>**;

}

return text;

}

}

Зверніть увагу на те, що константу **text** потрібно визначити над if, інакше вона не буде видно зовні цієї if.

Вийде, якщо змінна **showText** має значення **true** , то буде видно один текст, і якщо **false** - то інший.

**Цикли**

Шматочки HTML коду можна робити у циклі. Зазвичай для цього використовується цикл [map](http://old.code.mu/javascript/array/map.html). Приклад: нехай у нас є масив **arr**, давайте кожен елемент цього масиву обернемо в тег **li** і збережемо цей набір **li** в змінну **list** :

const arr **=** [1, 2, 3, 4, 5];

const list **=** arr.map(function(item, index) {

return **<**li**>**{item}**</**li**>**;

});

У **list** запишеться текст <li>1</li><li>2</li><li>3</li><li>4</li><li>5</li> - і це буде працювати незважаючи на те, що ці елементи не обгорнуті в загальний тег. Тобто в циклі таке можна робити, а безпосередньо так одразу написати у змінну – ні.

Давайте додамо метод **render** :

class App extends React.Component {

render() {

const arr **=** [1, 2, 3, 4, 5];

const list **=** arr.map(function(item, index) {

return **<**li**>**{item}**</**li**>**;

});

return **<**ul**>**

{list}

**</**ul**>**

}

}

Результатом роботи цього коду буде наступне:

<ul>

<li>1</li>

<li>2</li>

<li>3</li>

<li>4</li>

<li>5</li>

</ul>

**Проблема із ключами**

У попередньому прикладі ми формували наші елементи в циклі, ось так:

const list **=** arr.map(function(item, index) {

return **<**li**>**{item}**</**li**>**;

});

Це буде працювати, однак, якщо зазирнути в консоль - ми побачимо помилку: *Warning: Each child in array or iterator should have unique "key" prop* . В даному випадку React вимагає, щоб кожному елементу ми дали унікальний номер, щоб реакту було простіше з цими елементами працювати надалі.

Цей номер додається за допомогою атрибута **key** . Найчастіше (але не обов'язково) як номер використовується номер елемента в масиві. У нашому випадку цей номер зберігається в змінній **index** і виправлений рядок буде виглядати ось так: **<li key={index}>{item}</li>** .

Давайте запустимо – помилка з консолі зникне:

class App extends React.Component {

render() {

const arr **=** [1, 2, 3, 4, 5];

const list **=** arr.map(function(item, index) {

return **<**li key**=**{index}**>**{item}**</**li**>**;

});

return **<**ul**>**

{list}

**</**ul**>**

}

}

Ще раз: цей атрибут **key** має службове значення для реакції. Поки що просто знайте: якщо ви бачите таку помилку - додайте атрибут **key** з унікальним для кожного тега значенням і проблема зникне.

*Ключ****key****повинен бути унікальним лише всередині цього циклу, в іншому циклі значення****key****можуть співпадати зі значеннями іншого циклу.*

**Методи класу**

Зараз у нас вся робота відбувається у класі **App** . Це звичайний [клас JavaScript](http://old.code.mu/books/javascript/oop/) , відповідно в ньому ми можемо робити якісь свої методи. Давайте, наприклад, зробимо метод **getText()** , який повертатиме якийсь текст:

class App extends React.Component {

getText() {

return 'текст';

}

}

Врахуйте, що метод render є обов'язковим у класі, інакше React викине помилку. У простих прикладах я цей метод опускатиму, як це зроблено тут, щоб він не відволікав вашу увагу.

Викличемо наш метод **getText** у методі **render** і запишемо результат роботи нашого методу в змінну **text** . Нагадую, щоб викликати метод, до нього слід звернутися через **this**, так: **this.getText()**. Дивіться приклад:

class App extends React.Component {

getText() {

return 'текст';

}

render() {

const text **=** this.getText();

return **<**div**>**{text}**</**div**>**;

}

}

Можна не використовувати проміжну змінну **text**, а зробити **return** безпосередньо до результату роботи нашого методу **this.getText()**:

class App extends React.Component {

getText() {

return 'текст';

}

render() {

return **<**div**>**{ this.getText() }**</**div**>**;

}

}

Наш метод, звичайно ж, може повертати не просто текст, а шматочок HTML коду:

class App extends React.Component {

getText() {

return **<**div**>**текст**</**div**>**;

}

render() {

const text **=** this.getText();

return text;

}

}