



Передовые
инженерные
школы



МИНОБРНАУКИ
РОССИИ



УНИВЕРСИТЕТ
ИННОПОЛИС

Занятие 12

Введение в React



План занятия

1. Введение
2. Компонентный подход
3. Class Components
4. Functional Components
5. Pure Component
6. Stateless
7. Stateful

Введение

React - это интерфейсный JavaScript-фреймворк.

Его принципы проектирования учитывают множество распространенных проблем, с которыми сталкивались разработчики при создании пользовательских компонентов в своих веб-проектах.

React существует не для устранения проблем с сервером или базой данных, а скорее для устранения общих проблем проектирования и разработки интерфейса, которые существуют в более сложных веб-приложениях с отслеживанием состояния.

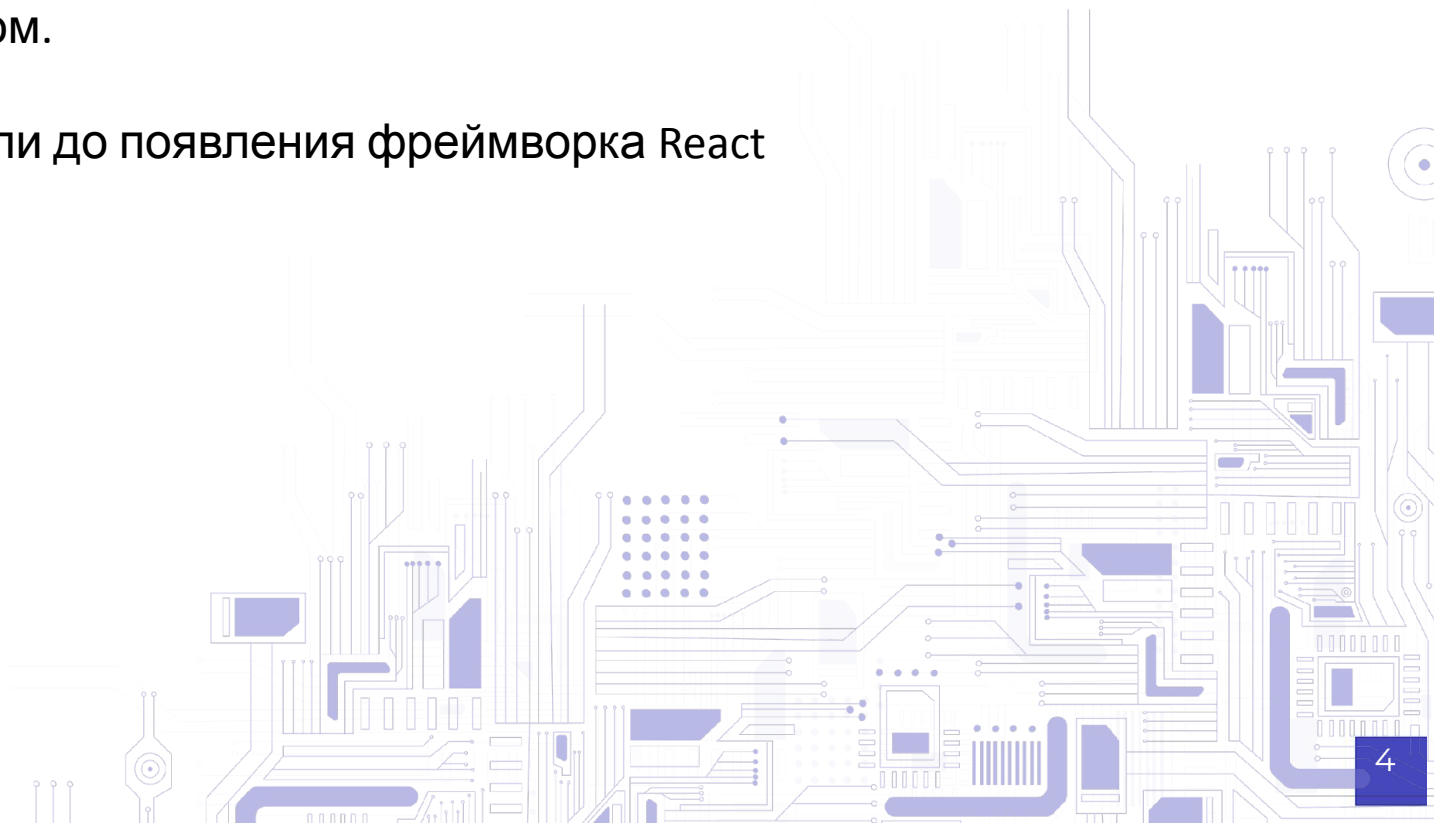
React - это фреймворк, который был создан во время безумия frontend JavaScript фреймворков.

React сам по себе - это не просто JavaScript, который вы можете использовать для создания элементов на странице.

Проблема добавления элементов объектной модели документа (DOM) на веб-страницу имела решения с первых дней существования Prototype и jQuery и др.

Вместо этого полезно думать о React как об устранителе разрыв между JavaScript и HTML или, более конкретно, кодом и браузером.

Рассмотрим проблемы, которые существовали до появления фреймворка React JavaScript.



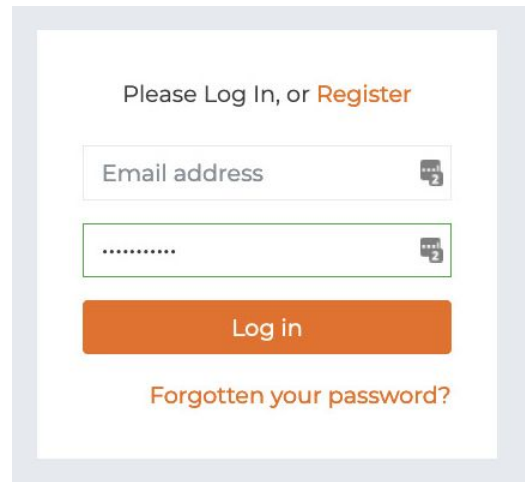
Проблемы до React

Веб-страница с небольшой формой для входа в учетную запись.


У вас есть по крайней мере два текстовых поля, обычно для имени пользователя и пароля, в которые вам нужно ввести данные.


Есть кнопка или ссылка, которая позволяет войти в систему после ввода данных, возможно, есть ссылка на случай напомнить пароль.

Есть логотип для этого сайта и какой-то фирменный стиль или другие привлекательные визуальные элементы



Please Log In, or [Register](#)

Email address 

..... 

[Log in](#)

[Forgotten your password?](#)

Проблемы до React

Веб-разработчик должен создать каждую из этих частей и соединить их таким образом, чтобы аудитория, посещающая этот сайт, могла взаимодействовать с каждым компонентом в отдельности.

Реальная проблема здесь в том, что все это может быть невероятно сложно собрать воедино.

Например, у вас может быть несколько входных данных формы, которые взаимодействуют друг с другом в зависимости от введенных и выбранных вами значений.

Каждая часть должна иметь возможность взаимодействовать со следующей, и существует множество различных типов сложных взаимодействий, которые должны происходить, выходя за рамки простого действия в результате нажатия кнопки.

Проблемы до React

Постепенно проверять поля, чтобы убедиться, что они соответствуют условиям валидации.

Далее...

Мы вводим новые и все более сложные уровни взаимодействия между различными элементами пользовательского интерфейса и общим состоянием каждого компонента.

Разберем состояние каждого компонента на:

- Визуальное состояние (как поле отображается пользователю)
- Состояние их данных (что введено в поле)
- Состояние по отношению к другим элементам пользовательского интерфейса (как форма входа в систему узнает о состоянии данных полей имени пользователя и пароля)
- Состояние их взаимодействия (можно ли нажать на кнопку, если поля имени пользователя или пароля пусты?)

Проблемы до React

Внезапно поймаем, что несколько строк HTML просто больше не сократят его.

Нужны более сложные блоки кода JavaScript с управлением состоянием как для визуальных состояний, так и для состояний данных для каждого компонента, а также способ связать их все вместе.

Нужно представить все это таким образом, чтобы не требовалось переписывать его каждый раз, когда кому-то нужно реализовать подобную форму, и таким образом, чтобы разработчик не стонал каждый раз, когда ему приходится прикасаться к коду.

Посмотрим, как React вписывается во все это и как он решает эту проблему.

Вместо того чтобы рассматривать каждый элемент, который браузер видит и отображает для пользователя, отдельно от HTML-кода и кода JavaScript (и, следовательно, отдельно от состояний, которые каждый из них отображает в любой данный момент времени), React позволяет рассматривать весь код как часть одной и той же структуры данных, все переплетены и работают вместе:

- Компонент
- Состояние
- Отображение (или рендеринг)



Вместо того чтобы рассматривать каждый элемент, который браузер видит и отображает для пользователя, отдельно от HTML-кода и кода JavaScript (и, следовательно, отдельно от состояний, которые каждый из них отображает в любой данный момент времени), React позволяет рассматривать весь код как часть одной и той же структуры данных, все переплетены и работают вместе:

- Компонент
- Состояние
- Отображение (или рендеринг)

Это значительно снижает сложность и накладные расходы при попытке объединить состояние компонента и отображение компонента с помощью смеси CSS и JavaScript

React представляет собой более элегантный способ, позволяющий разработчику целостно мыслить о том, что находится в браузере и с чем взаимодействует пользователь, а также о том, как это состояние меняется по ходу работы

Вместо того чтобы рассматривать каждый элемент, который браузер видит и отображает для пользователя, отдельно от HTML-кода и кода JavaScript (и, следовательно, отдельно от состояний, которые каждый из них отображает в любой данный момент времени), React позволяет рассматривать весь код как часть одной и той же структуры данных, все переплетены и работают вместе:

- Компонент
- Состояние
- Отображение (или рендеринг)

Это значительно снижает сложность и накладные расходы при попытке объединить состояние компонента и отображение компонента с помощью смеси CSS и JavaScript

React представляет собой более элегантный способ, позволяющий разработчику целостно мыслить о том, что находится в браузере и с чем взаимодействует пользователь, а также о том, как это состояние меняется по ходу работы

Введение в JSX

Чтобы писать компоненты в React, нужно понимать основной язык шаблонов, который React использует для создания компонента: **JSX**.

JSX - это своего рода гибрид HTML и JavaScript

```
index.jsx
1 import './main.css';
2
3 import React from 'react';
4 import App from './components/App.jsx';
5
6 main();
7
8 function main() {
9   const app = document.createElement('div');
10
11   document.body.appendChild(app);
12
13   React.render(<App />, app);
14 }
```



Введение в JSX

JSX представляет собой самый простой способ разрушить границы между HTML, который используется в качестве базовой разметки для своей веб-страницы, в основном пользовательского интерфейса приложения, и JavaScript, который нужен, чтобы сделать приложение интерактивным.

JSX за кулисами преобразуется в вызовы функций JavaScript, что позволяет писать свои компоненты удобным и привычным способом, как написание HTML для браузера.

Можно использовать большинство стандартных HTML-тегов и синтаксис, которые уже знаете (за несколькими заметными исключениями), но также можно включить JavaScript в свои шаблоны, чтобы лучше подходить к программному оформлению.

Идея заключается в том, что если умеете читать HTML и JavaScript, можно прочитать JSX.

Погружаемся глубже в JSX

```
<h1 className="greeting" style={ { "color": "red" } }>  
  Hello World!  
</h1>
```

далее

```
<div  
  className="greeting"  
  style={{ background: "black", color: "white" }}  
  onClick={() => alert('hello')}  
>
```

кроме `onClick` все так же остается множество методов по работе с элементами

Погружаемся глубже в JSX

Еще если не будет JSX то для создания простого кода h1

```
<h1> Hello World!</h1>
```

Потребуется писать так

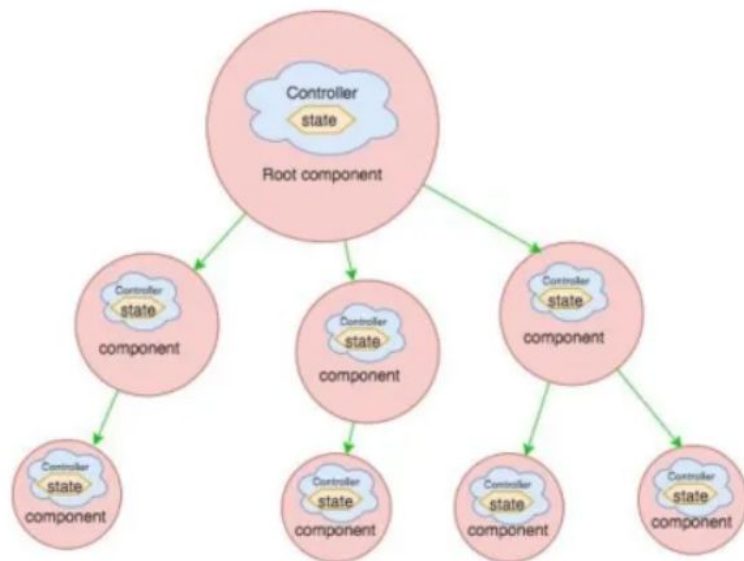
```
ReactDOM.render(  
  React.createElement('h1', null, 'Hello world!'),  
  document.getElementById('app')  
);
```


Компонентный подход

Компонент React — это JavaScript-функция или класс.

Элемент React — это то, что возвращается в результате компиляции языка JSX, с которым работает React, и вызова метода `React.createElement`, а после рендеринга внутри React становится в дереве DOM тем самым объектом, который пользователь увидит на экране.

То есть элемент создаётся в Virtual DOM, а при рендеринге он переносится в стандартный DOM веб-приложения и становится какой-нибудь кнопкой, на которую пользователь может нажать.



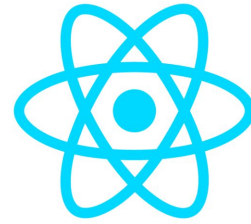
Компонентный подход

Компонент React:

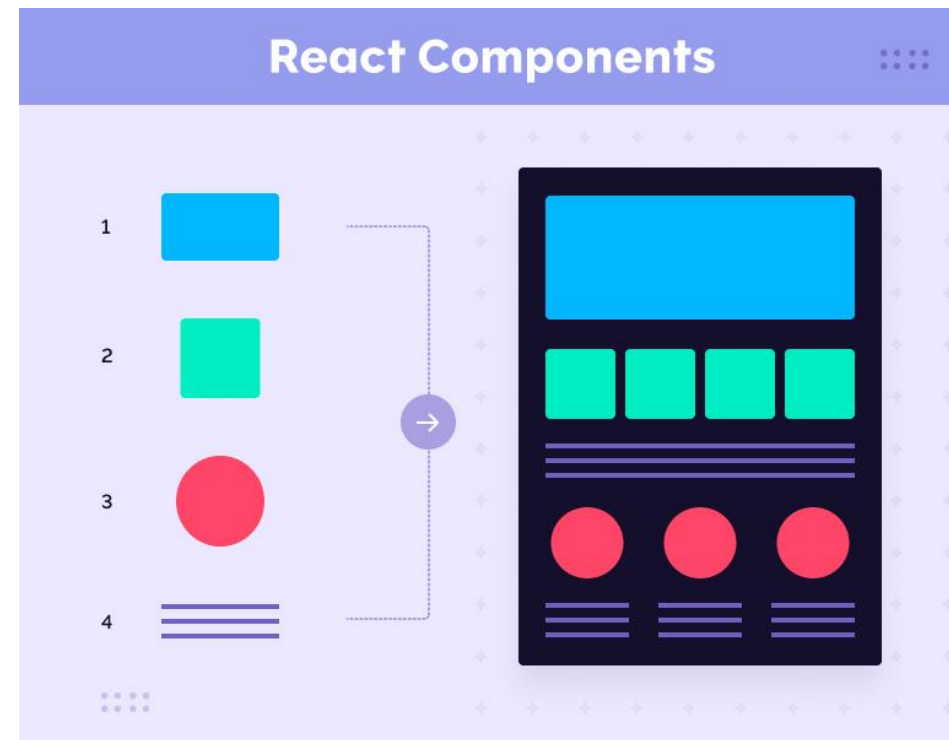
- может быть функцией или классом, который принимает в себя какие-то данные и возвращает элементы, другие компоненты, строки и числа;
- работает по жизненному циклу;
- содержит в себе разметку;
- может изменять своё состояние (мутабелен);
- работает с React hooks.

Элемент React:

- это то, что всегда возвращается компонентами;
- это представление объекта в узле DOM;
- создаётся и не изменяется (иммутабелен);
- не работает с React hooks, потому что не изменяется.



React Elements and Components



Классовый компонент

React позволяет определять компоненты на основе классов.

Производные от Component

Пользовательские компоненты построенные на основе классов обязаны расширять базовый обобщенный класс `Component<Props, State, Snapshot>` имеющего три необязательных параметра типа.

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

 импорт пространства имен *React*.

```
class Timer extends Component {  
  render() : ReactNode {  
    return null;  
  }  
}
```

```
export default Timer;
```

Классовый компонент

Props

Несмотря на то что *пропсы* делятся на обязательные и необязательные, все они по мере необходимости передаются в качестве аргументов конструктора при создании его экземпляра и доступны по ссылке **this.props**

Тем не менее за инициализацию необязательных пропсов ответственен сам классовой компонент для чего и предусмотрено статическое поле **defaultProps**

Аннотация `defaultProps` предполагает тип представляющий лишь ассоциированное с этим полем значение вынуждает разделить декларацию общих пропсов на два типа `DefaultProps` и `Props`. Ввиду того что тип `Props` представляет не только обязательные пропсы, но и необязательные, он должен расширять **extends** тип `DefaultProps`

```
interface DefaultProps {}  
interface Props extends DefaultProps {}  
class Timer extends Component {  
    public static readonly defaultProps = {};  
  
    constructor(props: Props) {  
        super(props);  
    }  
}
```

Классовый компонент

```
import React, { Component, ReactNode } from 'react';
// Имена интерфейсов получили префикс в виде названия компонента.
interface TimerDefaultProps {
  message: string;
}
interface TimerProps extends TimerDefaultProps {
  duration: number;
}
// Для конкретных типов преобразованных в типы только для чтения определен псевдоним.
type DefaultProps = Readonly<TimerDefaultProps>;
type Props = Readonly<TimerProps>;

class Timer extends Component<Props> {
  public static readonly defaultProps: DefaultProps = {
    message: `Done!`,
  };

  constructor(props: Props) {
    super(props);
  }
}
// Добавлен экспорт не только самого компонента, но и типа представляющего его основные
// пропсы.
export default Timer;
export { TimerProps }; // экспортируем типа *Props
```

Классовый компонент

Параметр children

```
class Label extends Component {  
  render() {  
    return (  
      <h1>{this.props.children}</h1>  
    );  
  }  
}
```

Классовый компонент

Состояние State

```
interface Props {}
interface State {
  yesCount: number;
  noCount: number;
}
class Counter extends Component<Props, State> {
  state = {
    yesCount: 0,
    noCount: 0,
  };

  buttonA_clickHandler = () => {
    // инкрементируем yesCount
    this.setState(
      (prevState): State => {
        return { yesCount: prevState.yesCount + 1 }; /**1 */
      }
    );
  };

  buttonB_clickHandler = () => {
    // инкрементируем noCount
    this.setState(
      (prevState): State => {
        return { noCount: prevState.noCount + 1 }; /**[2] */
      }
    );
  };
}
```

```
render() {
  return (
    <div>
      <p>Yes: {this.state.yesCount}</p>
      <p>No: {this.state.noCount}</p>
      <button onClick={this.buttonA_clickHandler}>
        yes++
      </button>
      <button onClick={this.buttonB_clickHandler}>
        no++
      </button>
    </div>
  );
}
```

Функциональный компонент

```
function ProfilePage(props) {  
  const showMessage = () => {  
    alert('Followed ' + props.user);  
  };  
  
  const handleClick = () => {  
    setTimeout(showMessage, 3000);  
  };  
  
  return (  
    <button onClick={handleClick}>Follow</button>  
  );  
}
```

Pure Компонент

PureComponent изменяет lifecycle-метод `shouldComponentUpdate`, автоматически проверяя, нужно ли заново отрисовывать компонент.

При этом PureComponent будет вызывать рендер только если обнаружит изменения в `state` или `props` компонента

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

class Greeting extends PureComponent {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```


Stateless



```
const BooksList = ({books}) => {  
  return (  
    <ul>  
      {books.map(book => {  
        return <li>book</li>  
      })}  
    </ul>  
  )  
}
```

Stateful

```
class Main extends Component {  
  constructor() {  
    super()  
    this.state = {  
      books: []  
    }  
  }  
  render() {  
    return <BooksList books={this.state.books} />;  
  }  
}
```



Передовые
инженерные
школы



МИНОБРНАУКИ
РОССИИ



УНИВЕРСИТЕТ
ИННОПОЛИС

Спасибо за внимание

