

ПЛАН ЗАНЯТИЯ

1. VirtualDom
2. JSX синтаксис
3. React компонент
4. Import/Export

Основные преимущества React JS:

1. Компонентный подход:

React.js основан на концепции создания множества маленьких, переиспользуемых компонентов. Эти компоненты затем объединяются в более крупные приложения.

2. Виртуальный DOM:

Это одна из главных особенностей React.js. Это означает, что React не обновляет все элементы на странице, а только те, которые действительно изменились, что делает приложение меньше и быстрее.

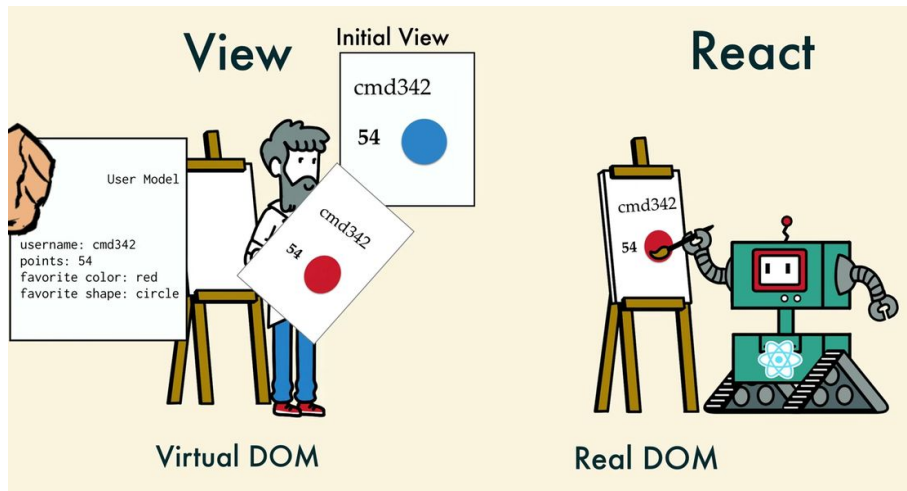
3. Однонаправленный поток данных:

Философия React.js заключается в том, что данные должны передаваться только в одном направлении: от родительского компонента к дочернему.

4. Использование синтаксиса JSX

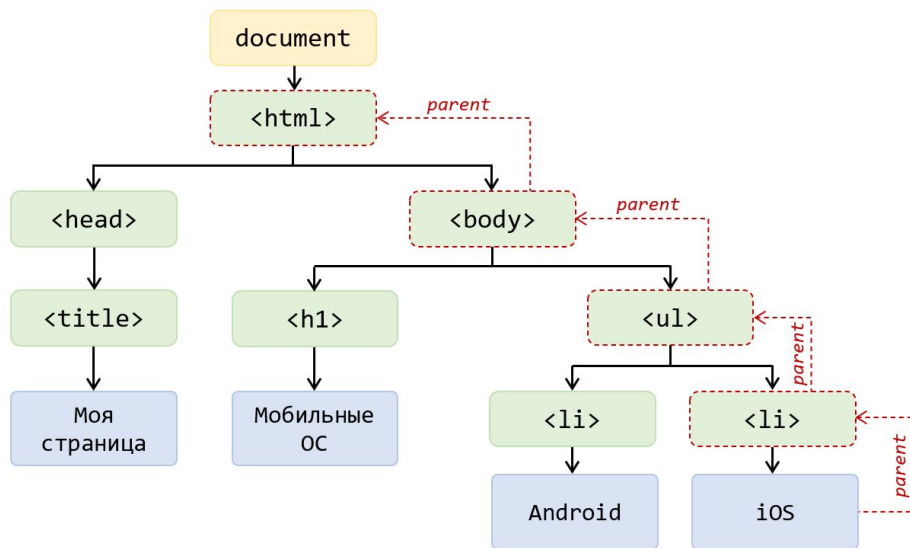


Virtual Dom

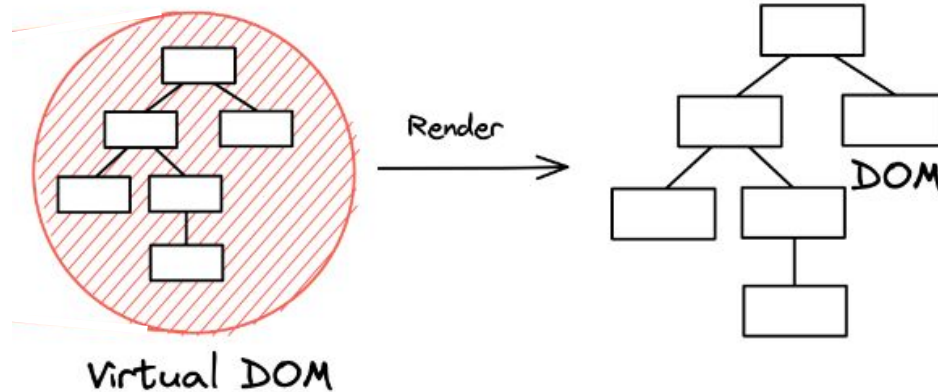


Что делает манипуляции с DOM медленными?

Когда происходят изменения в DOM, например, при добавлении, удалении или изменении элемента, происходят обновления DOM. Браузер перестраивает DOM-дерево и обновляет отображение веб-страницы в соответствии с этими изменениями.



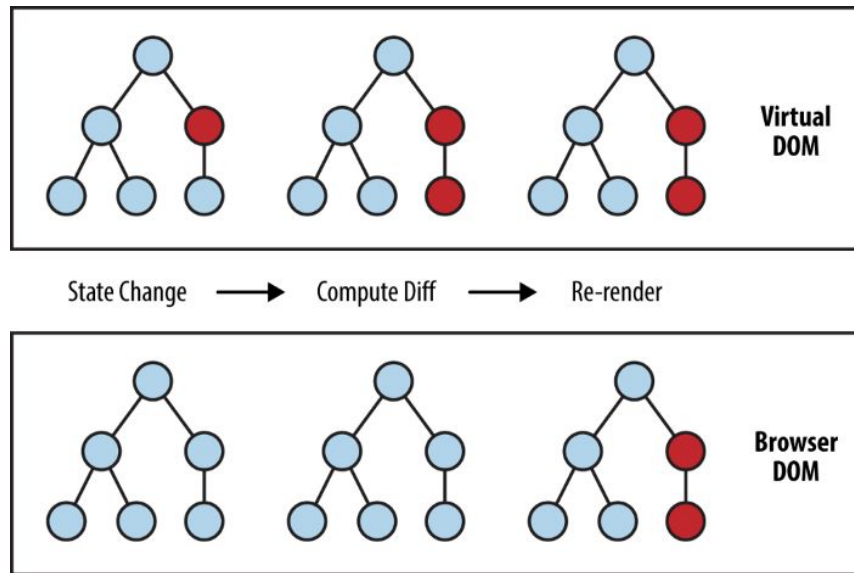
Virtual DOM (виртуальный DOM) - это концепция программирования, в которой «виртуальное» представление пользовательского интерфейса хранится в памяти и синхронизируется с «настоящим» DOM



Принцип работы React и VirtualDOM

Когда в UI добавляются новые элементы, создается Virtual DOM в виде дерева. Каждый элемент является узлом этого дерева. При изменении состояния любого элемента, создается новое дерево. Затем это новое дерево сравнивается (diffed) со старым.

После этого вычисляется наиболее эффективный метод внесения изменений в DOM. Цель данных вычислений состоит в минимизации количества операций, совершаемых с DOM. Тем самым, уменьшаются накладные расходы, связанные с обновлением DOM.



Принцип работы React

Этап 1

Написание кода на React



Принцип работы React

Этап 2

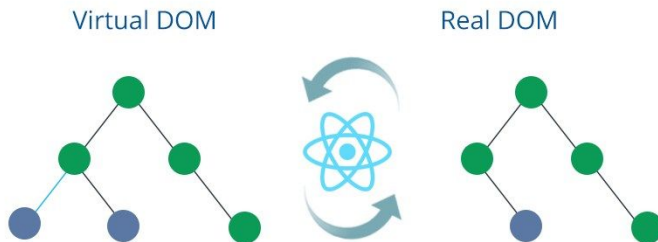
Транспиляция с помощью Babel: Код на React, написанный с использованием современных синтаксических возможностей JavaScript (например, JSX), может быть не полностью совместим с браузерами. Babel преобразует ваш код React в стандартный JavaScript, который браузеры могут понимать.



Принцип работы React

Этап 3

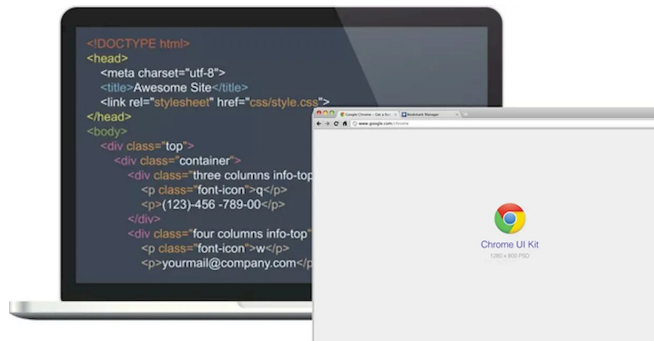
Создание виртуального DOM: React использует виртуальный DOM (Document Object Model), который представляет собой внутреннюю копию реального DOM. Это делается для оптимизации производительности при обновлении интерфейса.



Принцип работы React

Этап 4

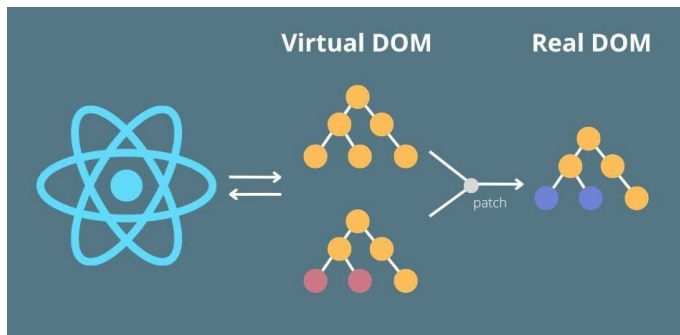
Отправка кода в браузер: Транспиливанный код React и другие зависимости отправляются в браузер, где он выполняется.



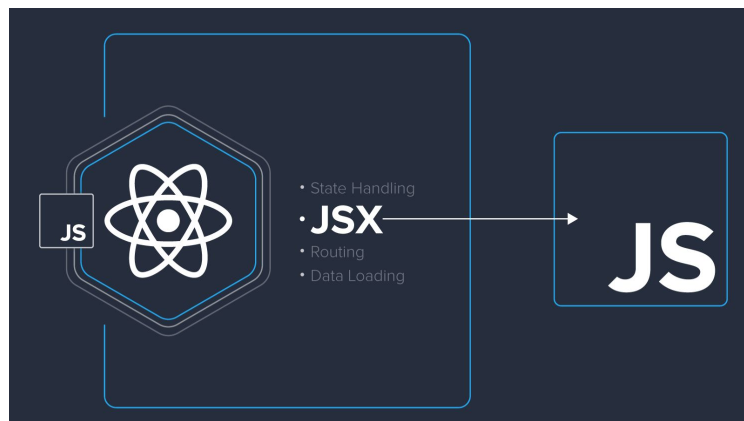
Принцип работы React

Этап 5

Обновление реального DOM: После внесения изменений в виртуальный DOM, React сравнивает его с реальным DOM и обновляет только те части, которые изменились.



JSX



Рассмотрим объявление переменной:

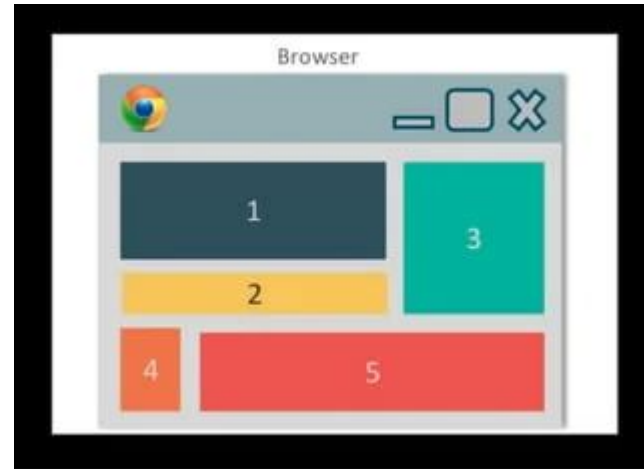
```
const element = <h1>Привет, мир!</h1>;
```

Этот странный тег — ни строка, ни фрагмент HTML.



Это JSX — расширение языка JavaScript. Его рекомендуем использовать его, когда требуется объяснить React, как должен выглядеть UI. JSX напоминает язык шаблонов, наделённый силой JavaScript.

JSX создаёт «элементы» React



Встраивание переменных в JSX

В JSX мы можем не просто создавать элементы, но и делать их гибкими, с помощью встраивания переменных.

В приведённом ниже примере мы объявляем переменную с именем `name`, а затем используем ее внутри JSX, обернув ее в фигурные скобки:

```
const name = 'Josh Perez';  
const element = <h1>Привет, {name}</h1>;
```

Установка атрибутов с помощью JSX

Вы можете использовать кавычки для указания строковых литералов в качестве атрибутов:

```
const element = <div tabIndex="0"></div>;
```

Вы также можете использовать фигурные скобки для вставки JavaScript-выражения в атрибут:

```
const element = <img src={user.avatarUrl}></img>;
```


Встраивание выражений в JSX

JSX допускает использование любых корректных JavaScript-выражений внутри фигурных скобок. Например, `2 + 2`, `user.firstName` и `formatName(user)` являются допустимыми выражениями.

```
function formatName(user) {  
  return user.firstName + ' ' + user.lastName;  
}  
  
const user = {  
  firstName: 'Марья',  
  lastName: 'Моревна'  
};  
  
const element = (  
  <h1>  
    Здравствуй, {formatName(user)}!  
  </h1>  
);
```

JSX это тоже выражение

JSX можно использовать внутри инструкций if и циклов for, присваивать переменным, передавать функции в качестве аргумента и возвращать из функции.

```
function getGreeting(user) {  
  if (user) {  
    return <h1>Здравствуй, {formatName(user)}!</h1>;  
  }  
  return <h1>Здравствуй, незнакомец.</h1>;  
}
```

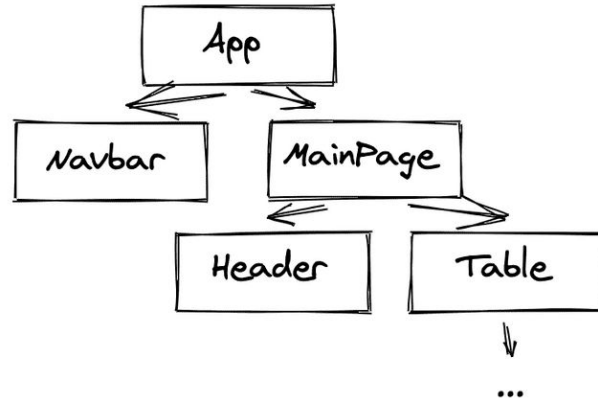
Встроенный оператор if-else с тернарным оператором

Другой метод встроенной условной отрисовки элементов — использование условного оператора в JavaScript

`условие ? true : false.`

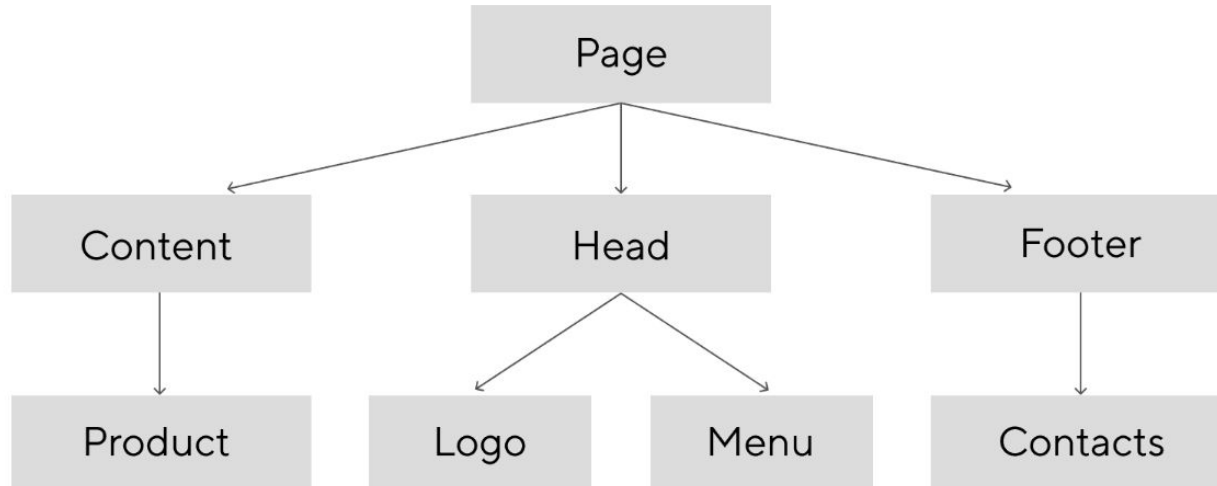
```
return (  
  <div>  
    Пользователь <b>{isLoggedIn ? 'в данный момент' : 'не'}</b> авторизован.  
  </div>  
);  
}
```

React-компонент



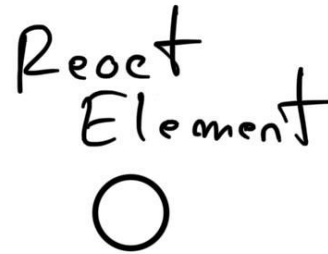
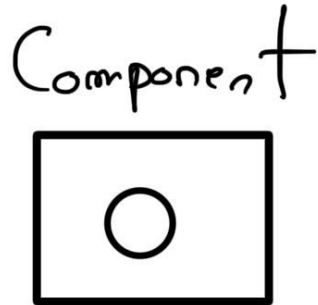
Приложения React состоят из компонентов.

Компонент — это часть пользовательского интерфейса, которая имеет свою собственную логику и внешний вид. Компонент может быть маленьким, как кнопка, или большим, как целая страница.



Раньше мы сталкивались только с элементами React, представляющие DOM-теги:

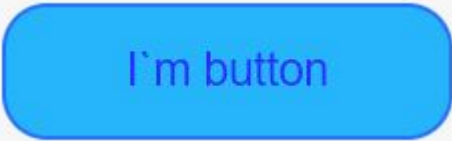
Однако элементы также могут быть пользовательскими компонентами



Компоненты React — это функции JavaScript, которые возвращают разметку:

```
1 function MyButton() {  
2   return <button>I'm a button</button>;  
3 }
```

Обратите внимание, что MyButton начинается с заглавной буквы. Так вы узнаете, что это компонент React.



I'm button

Компоненты, которые мы создаём могут быть вложены в другие теги или компоненты

```
1 export default function MyApp() {  
2   return (  
3     <div>  
4       <h1>Welcome to my app</h1>  
5       <MyButton />  
6     </div>  
7   );  
8 }
```

Welcome to my app

I'm button

Импорт и экспорт



Для использования функционала и инструментов, предоставляемые определенными библиотеками, мы должны импортировать их к себе в файл

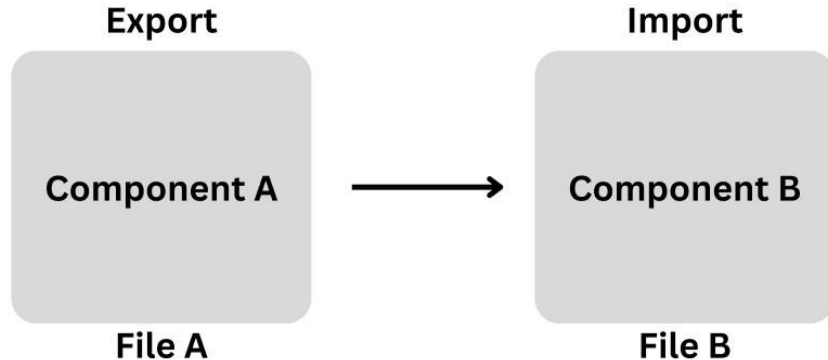
```
import React from 'react';
import moment from 'moment';

const MyComponent = () => {
  const currentDate = moment().format('MMMM Do YYYY, h:mm:ss a');

  return (
    <div>
      <h1>Hello! I am a React component.</h1>
      <p>Current date and time: {currentDate}</p>
    </div>
  );
};

export default MyComponent;
```

Магия компонентов заключается в возможности их повторного использования, часто имеет смысл начать разделять их на разные файлы. Это позволяет легко сканировать файлы и повторно использовать компоненты в большем количестве мест.



Экспорт по умолчанию

Экспорт компонента по умолчанию. В таком случае его можно будет использовать с любым именем при импорте.

```
// User.js
import React from 'react';

const UserProfile = () => {
  return <h1>User Profile</h1>;
};

export default UserProfile;
```

EXPORT

Импорт компонента экспортированного по умолчанию

Мы добавляем компонент UserProfile из предыдущего примера, в файл Dashboard, но под другим именем

```
// Dashboard.js
import React from 'react';
import CustomUserProfile from './User'; // Импортируем с другим именем

const Dashboard = () => {
  return (
    <div>
      <h2>Dashboard</h2>
      <CustomUserProfile />
    </div>
  );
};

export default Dashboard;
```

IMPORT

Именованный экспорт

Экспортируем компоненты по имени для возможности импорта нескольких компонентов из одного файла.

```
// UserComponents.js
import React from 'react';

export const UserProfile = () => {
  return <p>User Profile</p>;
};

export const UserPosts = () => {
  return <p>Recent Posts</p>;
};
```



Именованный импорт

Импортируем именованные компоненты из файла UserComponents в UserDashboard.

```
// UserDashboard.js
import React from 'react';
import { UserProfile, UserPosts } from './UserComponents';

const UserDashboard = () => {
  return (
    <div>
      <UserProfile />
      <UserPosts />
    </div>
  );
};

export default UserDashboard;
```

