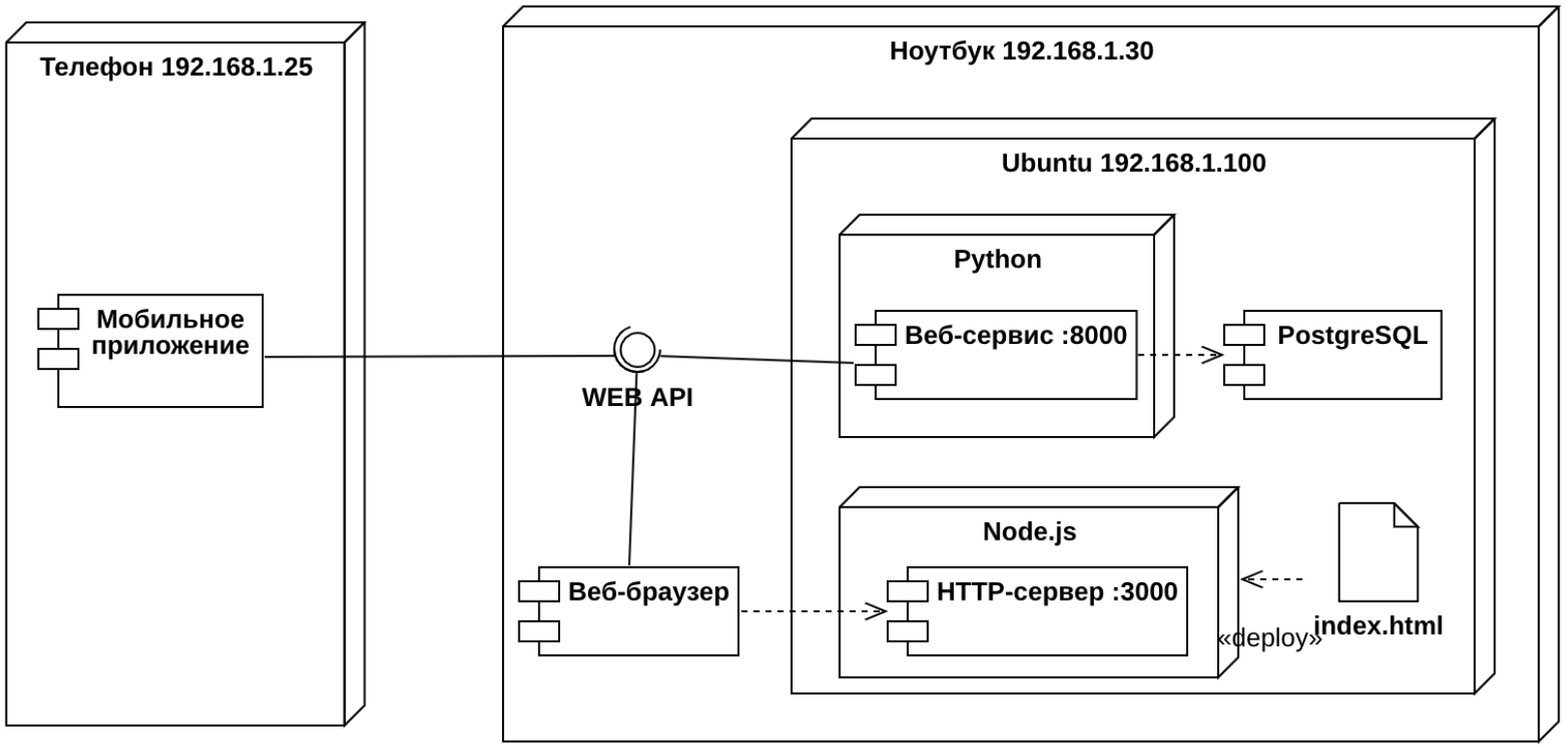
Лекция 8
Введение в React

Проектирование систем и продуктовая веб-разработка

Канев Антон Игоревич

Трехзвенная архитектура. AJAX

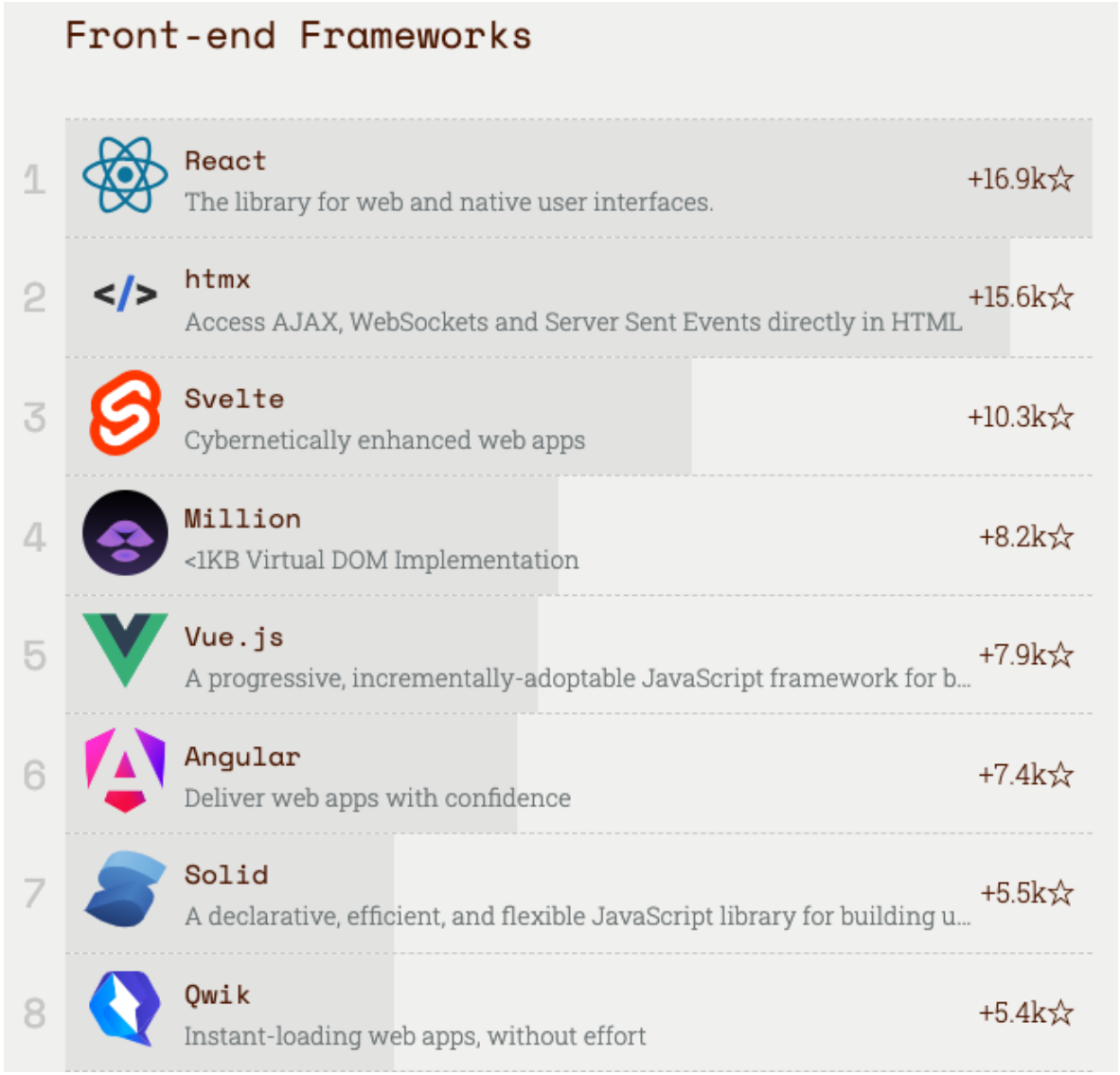
- На **диаграмме развертывания** мы указали наши бэкенд и фронтенд
- Указали IP и номера портов, которые используются этими приложениями
- Указать **реверс – прокси!**



Web-фреймворки. Фронтенд

- Есть большое множество фреймворков для разработки фронтенд приложений на JS/TS

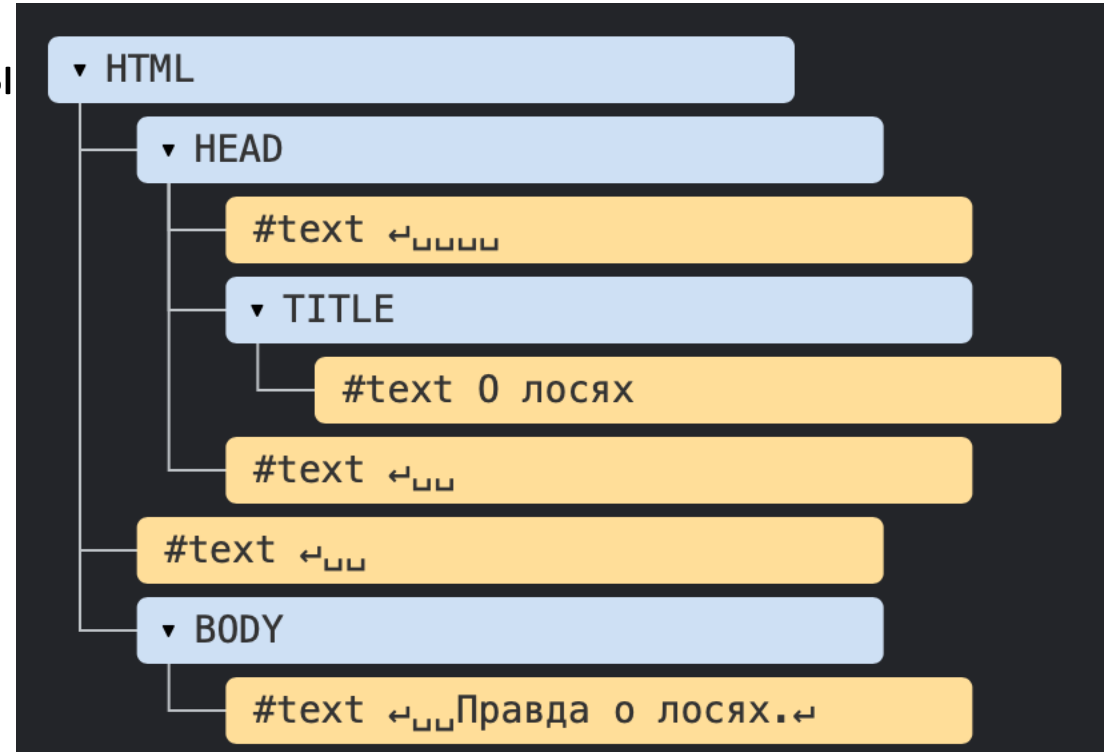
- Самым распространенным в России и в мире является React
- Поддерживается компанией из BigTech.



DOM

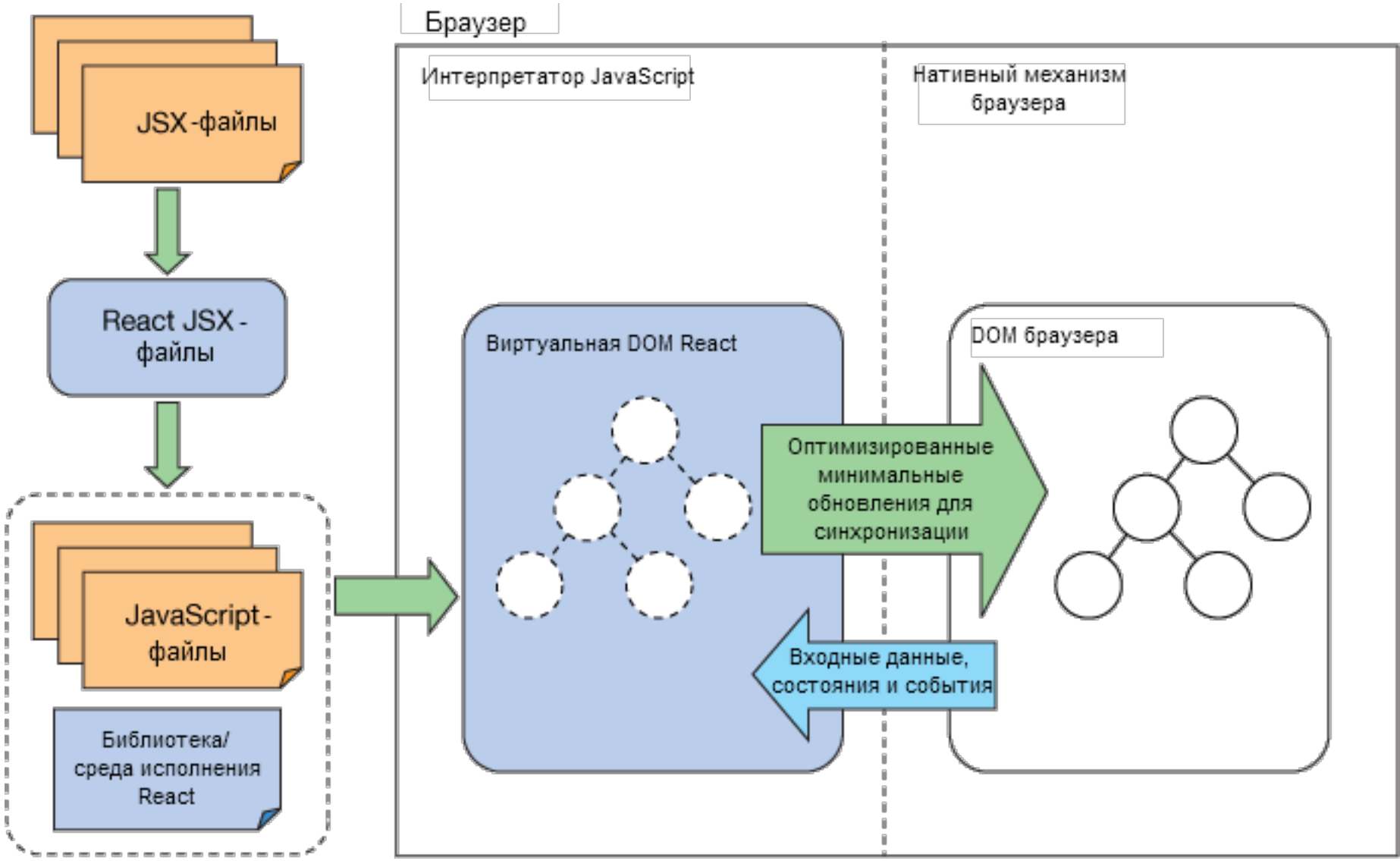
- Основой HTML-документа являются теги.
- В соответствии с объектной моделью документа («Document Object Model», коротко DOM), каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.
- Все эти объекты доступны при помощи JS
- Можем использовать их для изменения страницы

```
<!DOCTYPE HTML>
<html>
<head>
  <title>О лосях</title>
</head>
<body>
  Правда о лосях.
</body>
</html>
```



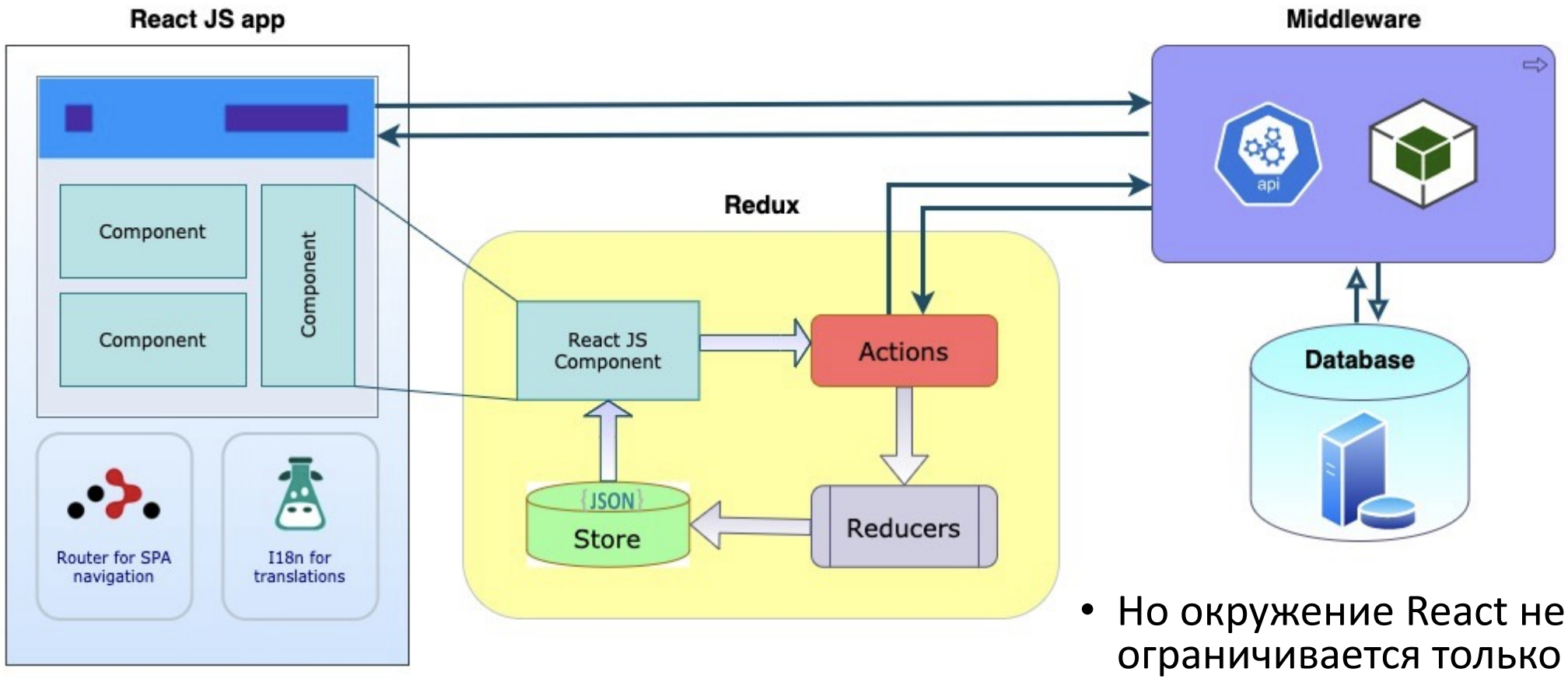
React

- Библиотека для работы с виртуальным DOM
- Документация



https://reactjs.org/docs/getting-started.html

React



- React в первую очередь отвечает за реализацию и работу View (компоненты)
- Это наш новый «механизм шаблонизации»

- Но окружение React не ограничивается только виртуальным DOM
- Это запросы к API, библиотеки компонентов, хранение данных и тд

JSX

- JSX — расширение синтаксиса JavaScript. TSX – то же, но для TS
- Этот синтаксис выглядит как язык шаблонов, но наделён всеми языковыми возможностями JavaScript. Родился из XHP для PHP
- В результате компиляции JSX возникают простые объекты — «React-элементы».

- React DOM использует стиль именования camelCase для свойств вместо обычных имён HTML-атрибутов.
- Например, в JSX атрибут tabindex станет tabIndex.
- В то время как атрибут class записывается как className, поскольку слово class уже зарезервировано в JavaScript

Компоненты

- React-компоненты — это повторно используемые части кода, которые возвращают React-элементы для отображения на странице.
- Функциональные и классовые компоненты

```
function Welcome(props) {
  return <h1>Привет, {props.name}</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Привет, {this.props.name}</h1>;
  }
}
```

Props

- props (пропсы) — это входные данные React-компонентов, передаваемые от родительского компонента дочернему компоненту.
- В любом компоненте доступны props.children. Это контент между открывающим и закрывающим тегом компонента.
- Для классовых компонентов используйте this.props.children

```
export interface BrowserRouterProps {
    basename?: string | undefined;
    children?: React.ReactNode;
    getUserConfirmation?: ((message: string, callback: (ok:
    forceRefresh?: boolean | undefined;
    keyLength?: number | undefined;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Привет, {this.props.name}</h1>;
  }
}
```

```
<BrowserRouter basename="/">
          <Switch>
            <Route exact path="/">
              <h1>Это наша стартовая страница</h1>
            </Route>
            <Route path="/new">
              <h1>Это наша страница с чем-то новеньким</h1>
            </Route>
          </Switch>
</BrowserRouter>
```

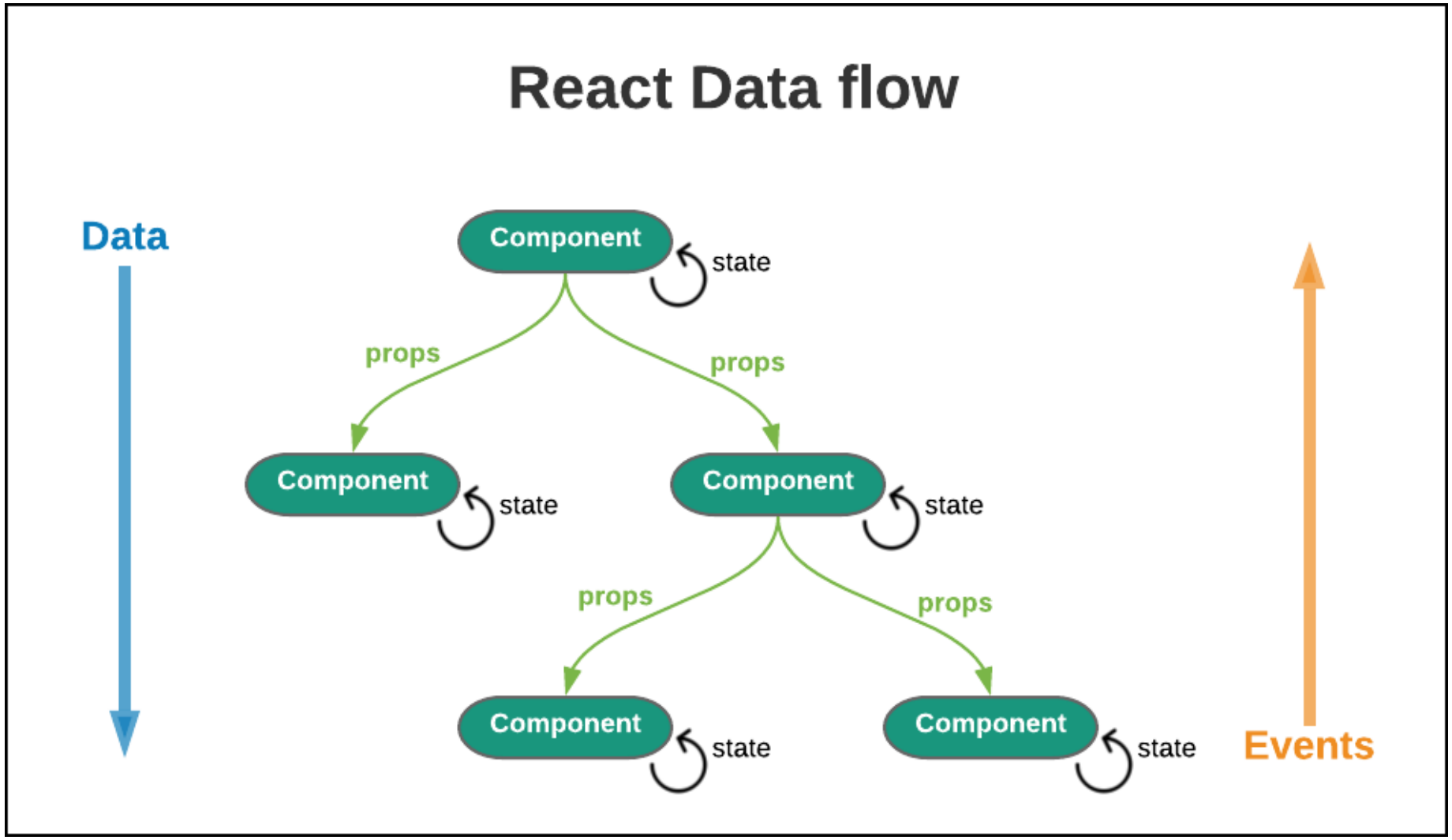
Состояние

- Компонент нуждается в state, когда данные в нём со временем изменяются.
- Например, компоненту Checkbox может понадобиться состояние isChecked.

- Разница между пропсами и состоянием заключается в основном в том, что состояние нужно для управления компонентом, а пропсы для получения информации.

Поток данных и сообщений

- **Слева** иконка корзины
- **Справа** массив из карточек наших услуг
- В корзине меняется **количество**, она становится **доступной** или нет
- Массив услуг **тоже меняется**, например при поиске
- Кнопка поиска **порождает событие** и передает **наверх** на страницу



Жизненный цикл приложения

- **1: Монтирование**

компонент запускает getDerivedStateFromProps(), потом запускается render(), возвращающий JSX. React «монтируется» в DOM

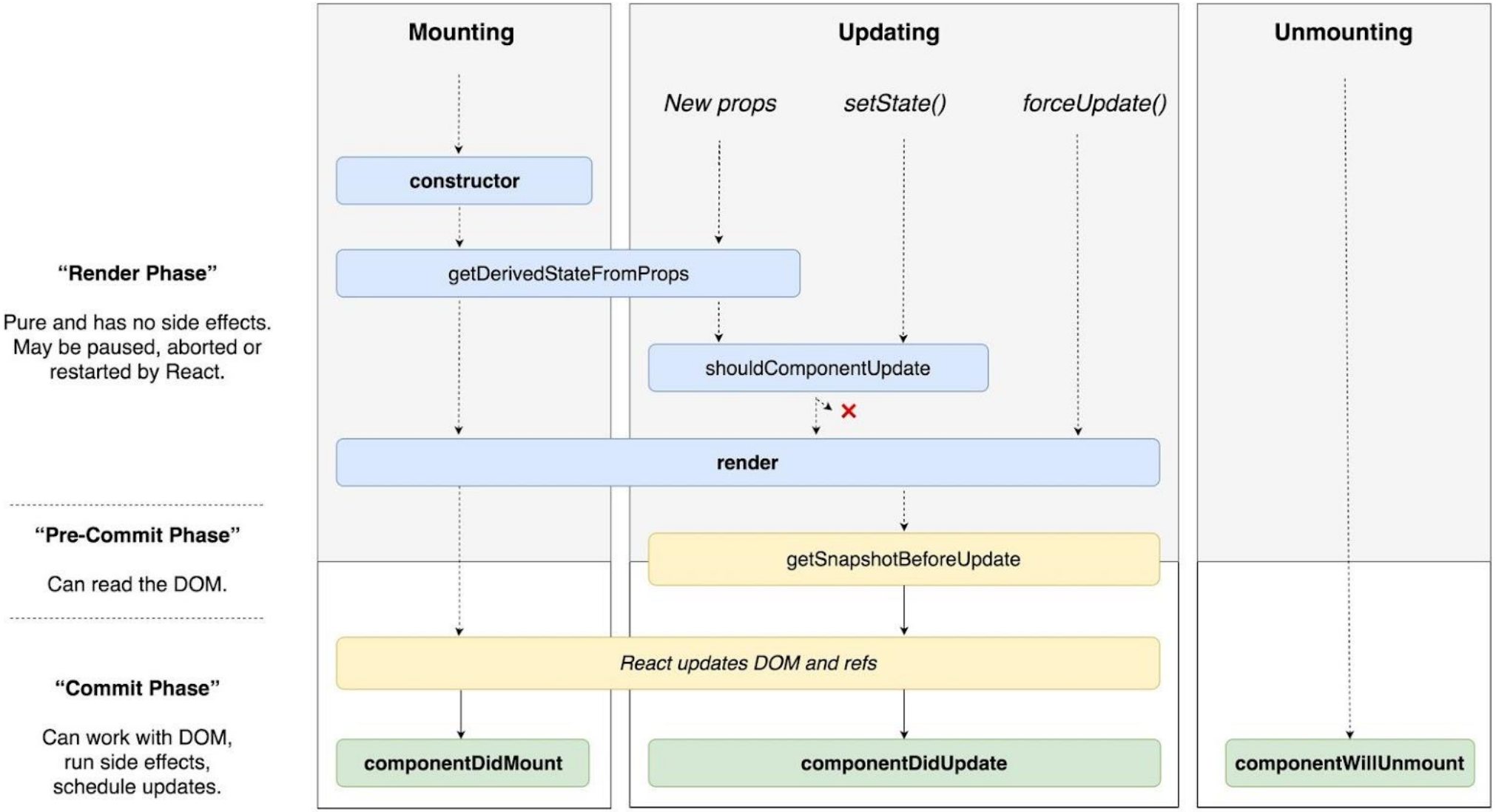
- **2: Обновление**

Данный этап запускается во время каждого изменения состояния либо свойств

- **3: Размонтирование**

React выполняет запуск componentWillUnmount() непосредственно перед удалением из DOM

Методы жизненного цикла компонента



useEffect

- componentDidMount()
- componentDidUpdate()
- componentWillUnmount()

- Не путать useEffect и жизненный цикл – **ЭТО РАЗНЫЕ ВЕЩИ**
- Жизненный цикл это гораздо более широкое понятие: работа с props, render, отслеживание состояний и тд
- useEffect это только 3 события: зеленые блоки из предыдущей схемы

```
useEffect( effect: ()=>{
    console.log('Этот код выполняется только на первом рендере компонента')
    // В данном примере можно наблюдать Spread syntax (Троеточие перед массивом)
    setNames( value: names=>[...names, 'Бедный студент'])

    return () => {
        console.log('Этот код выполняется, когда компонент будет размонтирован')
    }
}, deps: [])

useEffect( effect: ()=>{
    console.log('Этот код выполняется каждый раз, когда изменится состояние showNames ')
    setRandomName(names[Math.floor( x: Math.random()*names.length)])
}, deps: [showNames])
```

Функциональные компоненты

- Описание компонентов с помощью чистых функций создает меньше кода, а значит его легче поддерживать.
- Чистые функции намного проще тестировать. Вы просто передаете props на вход и ожидаете какую то разметку.
- В будущем чистые функции будут выигрывать по скорости работы в сравнении с классами из-за отсутствия методов жизненного цикла

- Все это стало возможным благодаря хукам https://react.dev/reference/react

Хуки

```jsx
import React, { useEffect, useState } from 'react'
import Axios from 'axios'

export default function Hello() {

    const [Name, setName] = useState("")

    useEffect(() => {
        Axios.get('/api/user/name')
        .then(response => {
            setName(response.data,name)
        })
    }, [])

    return (
        <div>
            My name is {Name}
        </div>
    )
}
```

```jsx
import React, { Component } from 'react'
import Axios from 'axios'

export default class Hello extends Component {

    constructor(props) {
        super(props);
        this.state = { name: ""};
    }

    componentDidMount() {
        Axios.get('/api/user/name')
        .then(response => {
            this.setState({ name: response.data.name })
        })
    }

    render() {
        return (
            <div>
                My name is {this.state.name}
            </div>
        )
    }
}
```

Хуки. useState

- Хуки позволяют работать с состоянием компонентов, с методами их жизненного цикла, с другими механизмами React без использования классов.

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
```

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

function Example() {
  // Объявление новой переменной состояния «count»
  const [count, setCount] = useState(0);
        return <div onClick={()=>setCount(count=>count++)}>{count}</div>
}
```

ES6

- ECMAScript 2015

```
function foo(x, y, z) {
    console.log(x, y, z);
}

let arr = [1, 2, 3];
foo(...arr); // 1 2 3
```

```
var a = 2;
{
    let a = 3;
    console.log(a); // 3
}
console.log(a); // 2
```

```
class Task {
    constructor() {
        console.log("Создан экземпляр task!");
    }

    showId() {
        console.log(23);
    }

    static loadAll() {
        console.log("Загружаем все tasks...");
    }
}
```

```
function foo(...args) {
    console.log(args);
}
foo(1, 2, 3, 4, 5); // [1, 2, 3, 4, 5]
```

```
// Классическое функциональное выражение
let addition = function(a, b) {
    return a + b;
};

// Стрелочная функция
let addition = (a, b) => a + b;
```

Babel

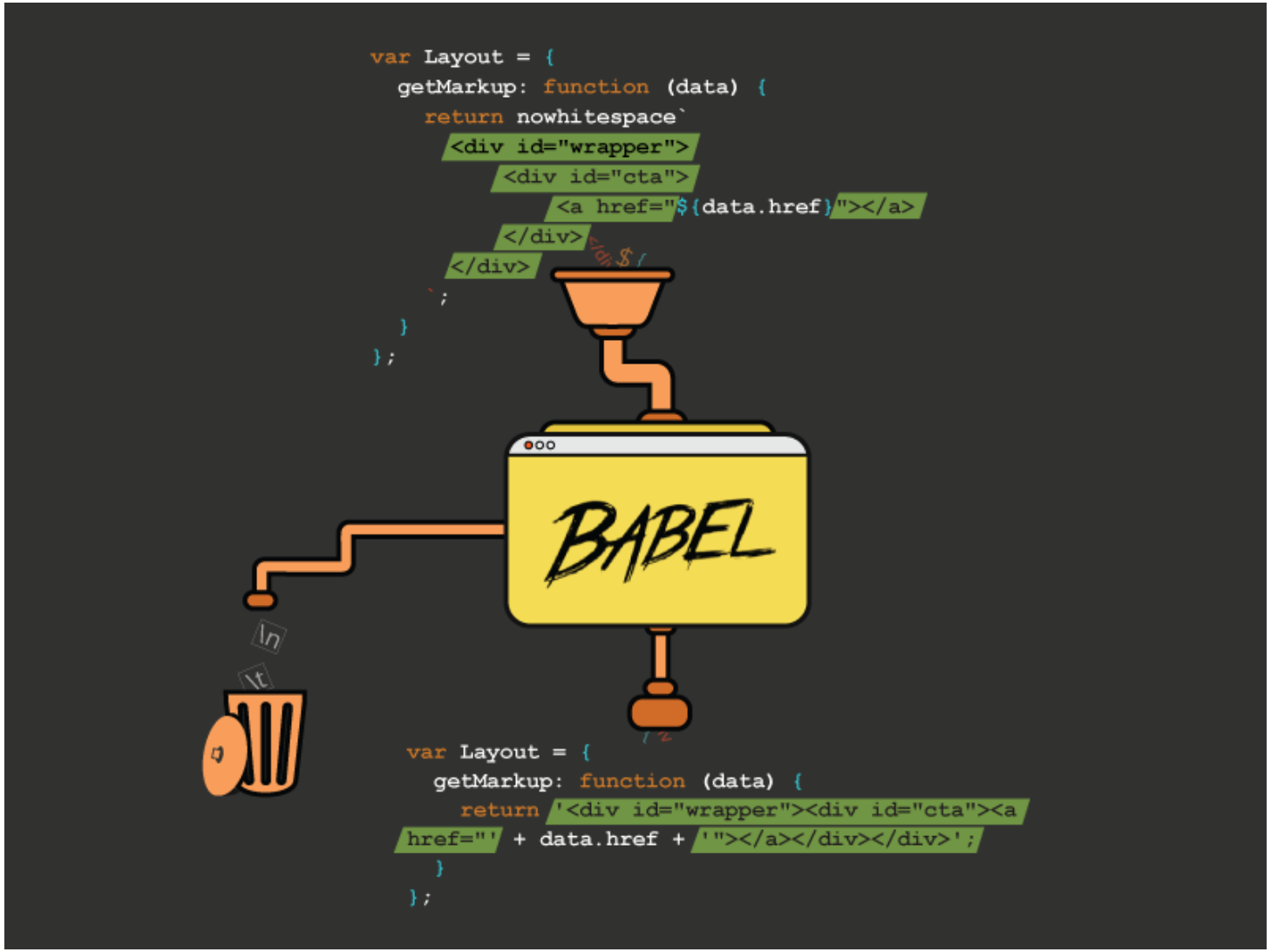
Babel — многофункциональный транспайлер, позволяет транспиллировать например код из ES6 (ES2015) в старый ES5

- It turns ES2015:

```
const adding = (a, b) => a + b
```

- into old JavaScript:

```
'use strict';
var adding = function adding(a, b) {
    return a + b;
};
```



Преимущества TypeScript

TypeScript — язык программирования, представленный Microsoft в 2012 году. TypeScript является **обратно совместимым** с JavaScript и компилируется в последний. TypeScript отличается от JavaScript возможностью явного статического назначения типов, а также поддержкой подключения модулей

- Аннотации типов и проверка их согласования на этапе компиляции
- Интерфейсы, кортежи, декораторы свойств и методов, расширенные возможности ООП
- TypeScript — **надмножество JavaScript**, поэтому любой код на JavaScript будет выполнен и в TypeScript
- Широкая поддержка IDE и адекватный автокомплит
- Поддержка ES6-модулей из коробки

TypeScript (vs JS)

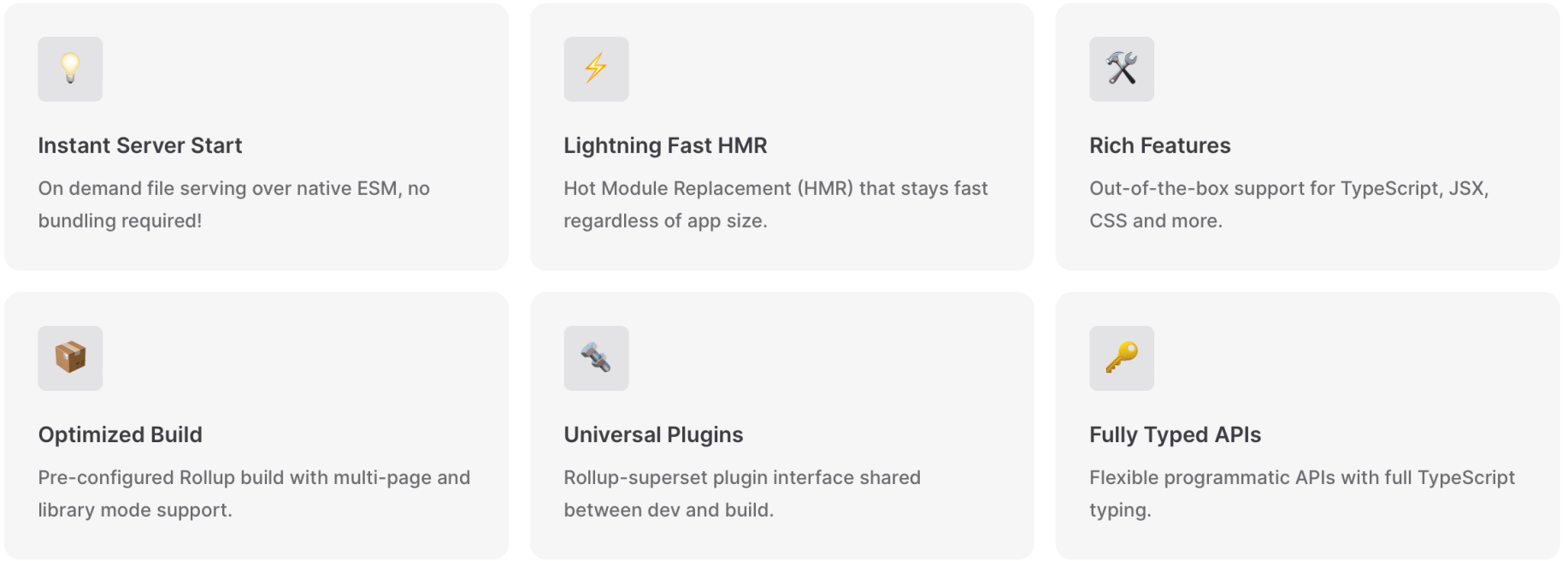
```
1 function sum(a: number, b: number): number {
2     retun a + b
3 }
```

```
1 function sum(a, b) {
2     retun a + b
3 }
```

```
1 interface Person {
2     name: string;
3     age: number;
4 }
5 function meet(person: Person) {
6     return `Привет, я ${person.name}, мне ${parson.age}`;
7 }
8 const user = { name: "Jane", age: 21 };
9 console.log(meet(user));
```

Vite

- Окружение для разработки и сборщик
- Мы используем Vite вместо CRA
- Поддерживает React и Vue.js
- Поддерживает ряд возможностей

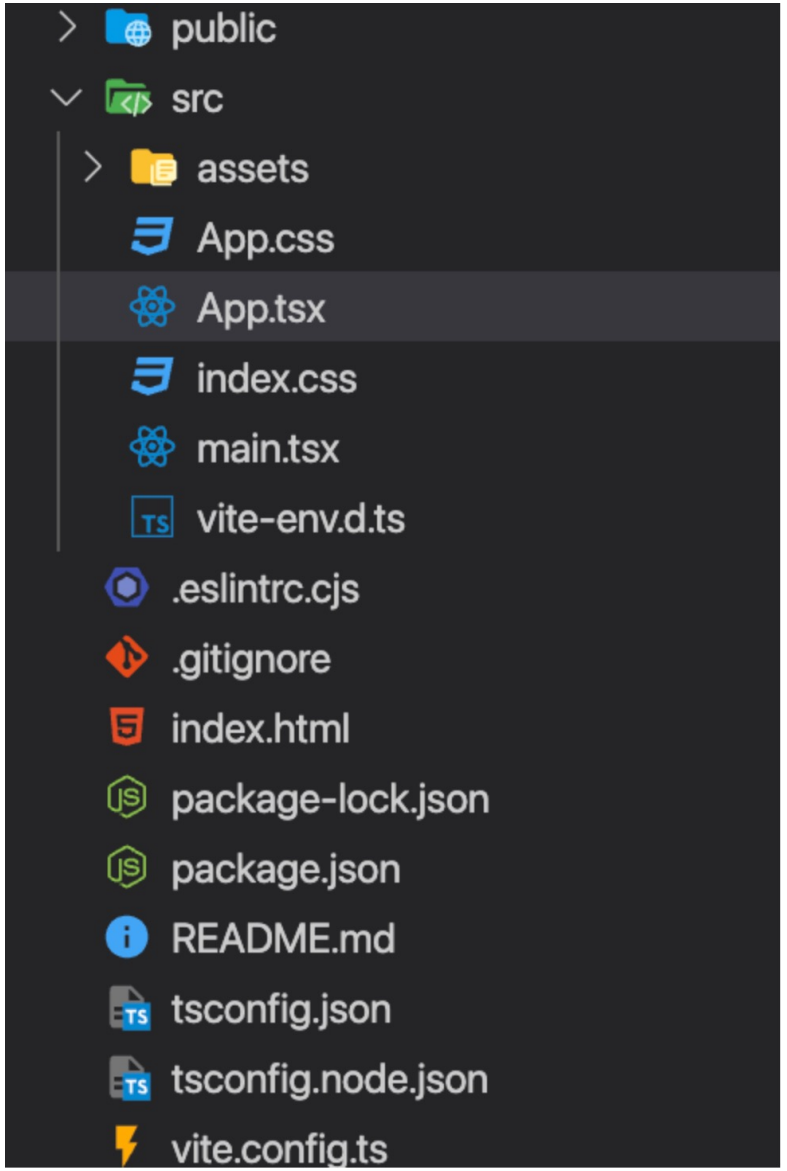



```
npm create vite@latest my-app -- --template react-ts
cd my-app
npm install
```

Структура проекта Vite

В папке проекта у нас будут следующие файлы:

- package.json - основной файл с информацией о проекте
- package-lock.json - лок файл со списком зависимостей
- vite.config.ts - конфигурационный файл сборщика Vite
- tsconfig.json - конфигурационный файл TypeScript
- tsconfig.node.json - конфигурационный файл TypeScript при запуске на Node
- .eslintrc.cjs - конфигурационный файл Eslint
- index.html - основной файл нашего приложения. Он будет первым загружаться, когда пользователь заходит на страницу
- src/main.tsx - основной TS файл нашего приложения. Тут мы запускаем отрисовку приложения
- src/App.tsx - верстка приложения. Логотип Vite и React



Vite + React. Основные файлы

- Vite – альтернатива Create React App. Сборщик модулей JS

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React + TS</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

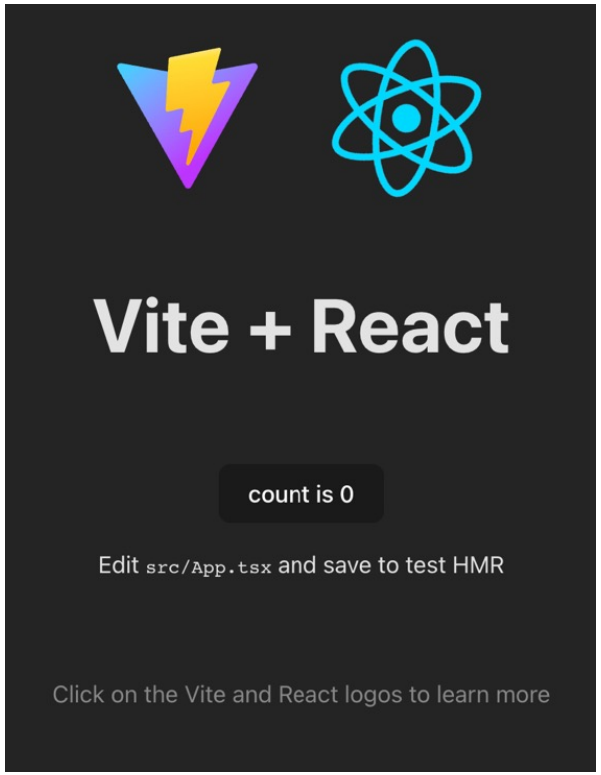
- index.html
- Это наш html файл, но все содержимое мы снова описываем на JS(TS) как в прошлом году

- main.js
- Файл с которого начинается описание структуры наших компонентов
- Мы должны все описать или поместить в **компонент App**

```tsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.tsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Vite + React. App.tsx

- Первый и **главный компонент** нашего приложения
- В шаблоне уже есть **кнопка**, **хук состояния**
- Наша задача будет разделить эту структуру на **отдельные компоненты**




```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

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

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App
```

Роутинг

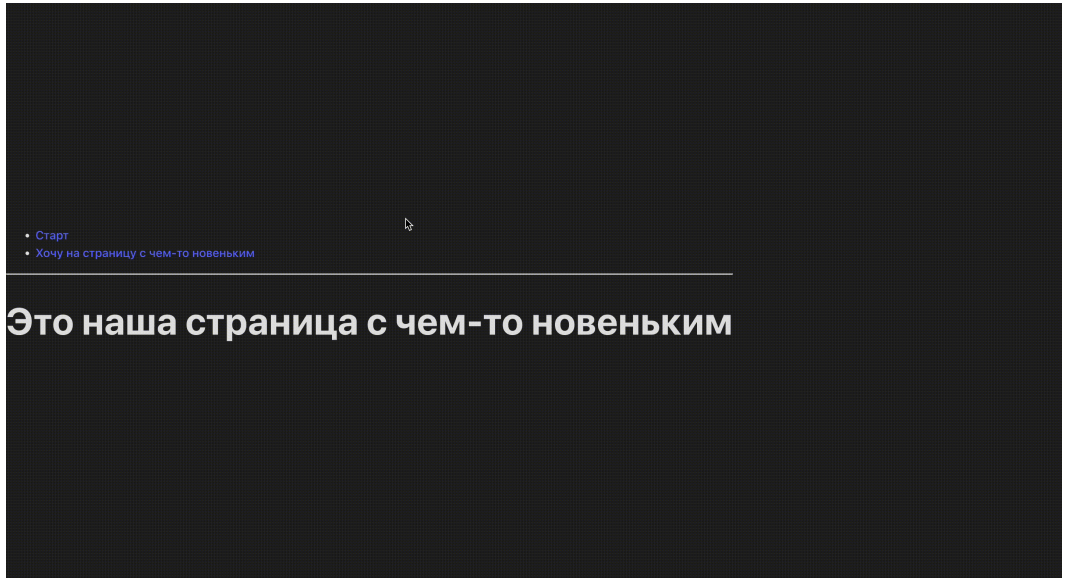
- Добавляем роутер в main.js для обработки переходов по url нашего приложения
- Пока заменили наш компонент App.tsx на простые элементы
- Получить мы их можем если введем нужный url

```
npm i react-router-dom
npm i @types/react-router-dom -D
```

```
import React from 'react'
import ReactDOM from 'react-dom/client'

import { createBrowserRouter, RouterProvider } from 'react-router-dom'
import './index.css'

const router = createBrowserRouter([
  {
    path: '/',
    element: <h1>Это наша стартовая страница</h1>
  },
  {
    path: '/new',
    element: <h1>Это наша страница с чем-то новеньким</h1>
  }
])

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>,
)
```

Router

- Добавляем новый общий элемент с ссылками на страницы нашего приложения
- Нам теперь не нужно вводить вручную url, можем просто нажать ссылку



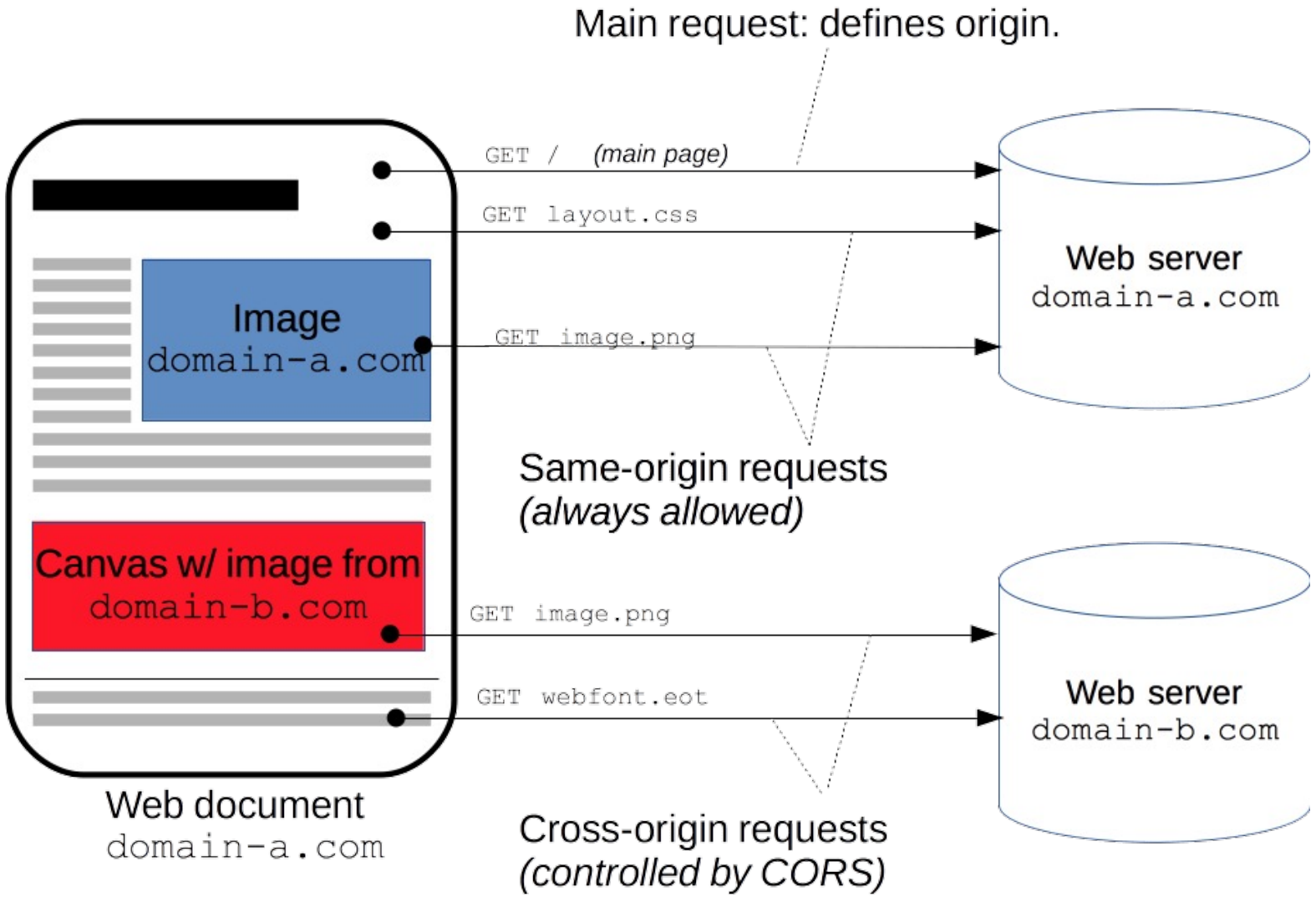

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { createBrowserRouter, RouterProvider, Link } from 'react-router-dom'
import './index.css'

const router = createBrowserRouter([
  {
    path: '/',
    element: <h1>Это наша стартовая страница</h1>
  },
  {
    path: '/new',
    element: <h1>Это наша страница с чем-то новеньким</h1>
  }
])

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <ul>
      <li>
        <a href="/">Старт</a>
      </li>
      <li>
        <a href="/new">Хочу на страницу с чем-то новеньким</a>
      </li>
    </ul>
    <hr />
    <RouterProvider router={router} />
  </React.StrictMode>,
)
```

Cors

- CORS - мы получили страницу с одного домена, а запросы отправляем на другой

Как решить?

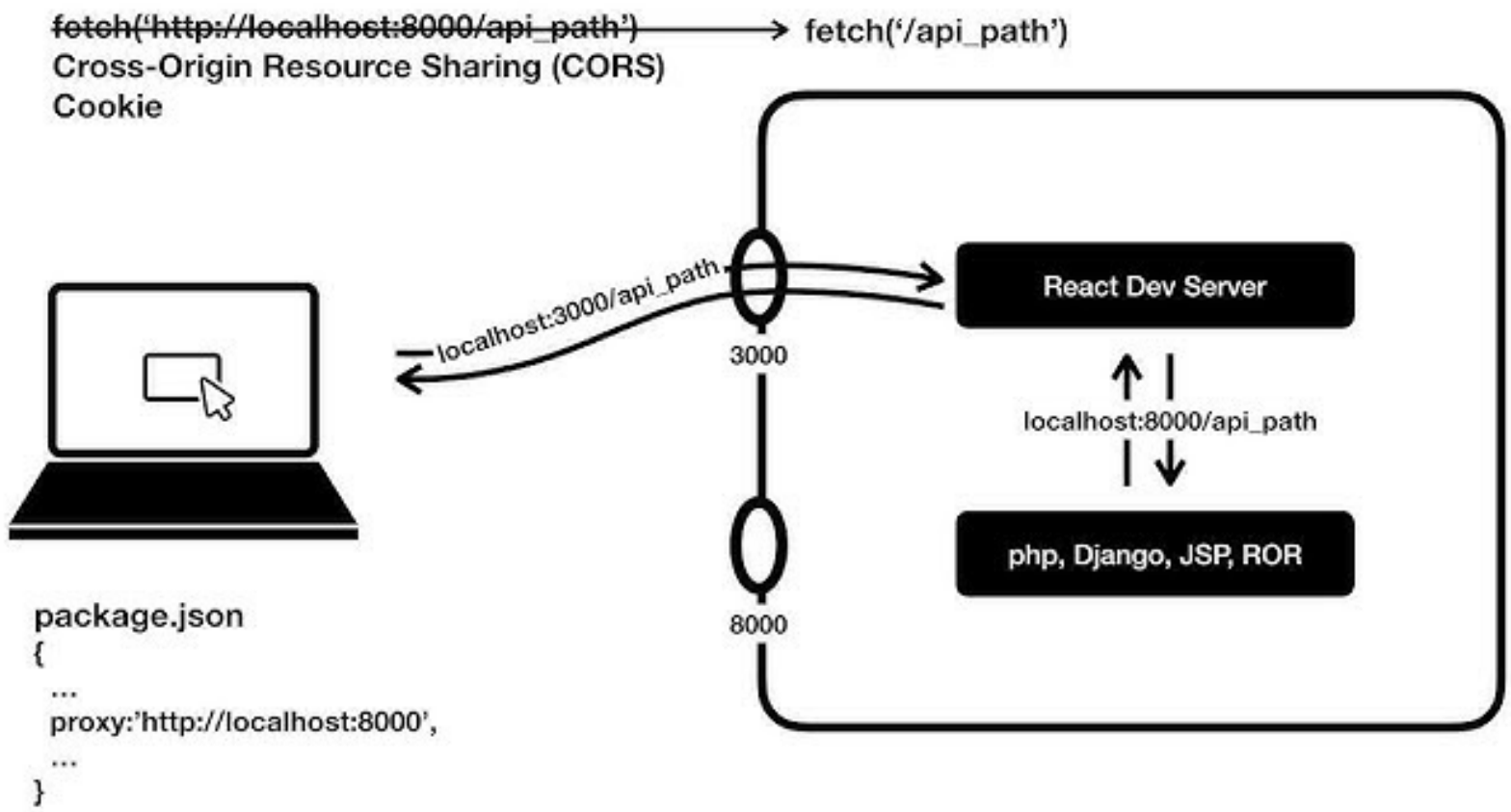
- CORS – заголовки на бекенде
- Проксирование через сервер фронтенда



Обратный прокси-сервер для CORS

- Одно из решений - отправляем запросы на напрямую в веб-сервис, а проксируем через наш сервер фронтенда

- Похоже на prod решение при проксировании через Nginx

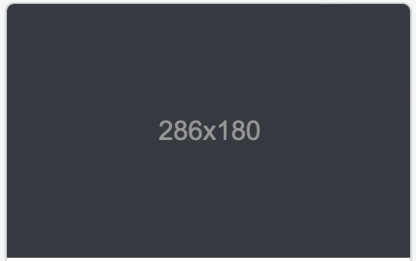


React-Bootstrap vs MUI

Primary Secondary Success Warning Danger Info Light Dark Link

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

This is a info alert—check it out!



Card Title

Some quick example text to build on the card title and make up the bulk of the card's content.

Go somewhere

Small Medium Large

ADD TO CART

Button

Standard Filled Outlined

Check out this alert!

Alert

Outlined Standard Filled

Username

Ultraviolet

Text Field

CLICK TO OPEN

Menu

Dessert	Calories
Frozen yoghurt	109
Cupcake	305

Table

Want to see more?

Check out the docs for details of the complete library.

Learn more ›

Figma MUI

- Набор компонентов MUI доступен в Figma
- Можно создать дизайн, используя готовые компоненты, иконки