SYNERGY
IT INSTITUTE

React

Занятие 1. Установка и запуск

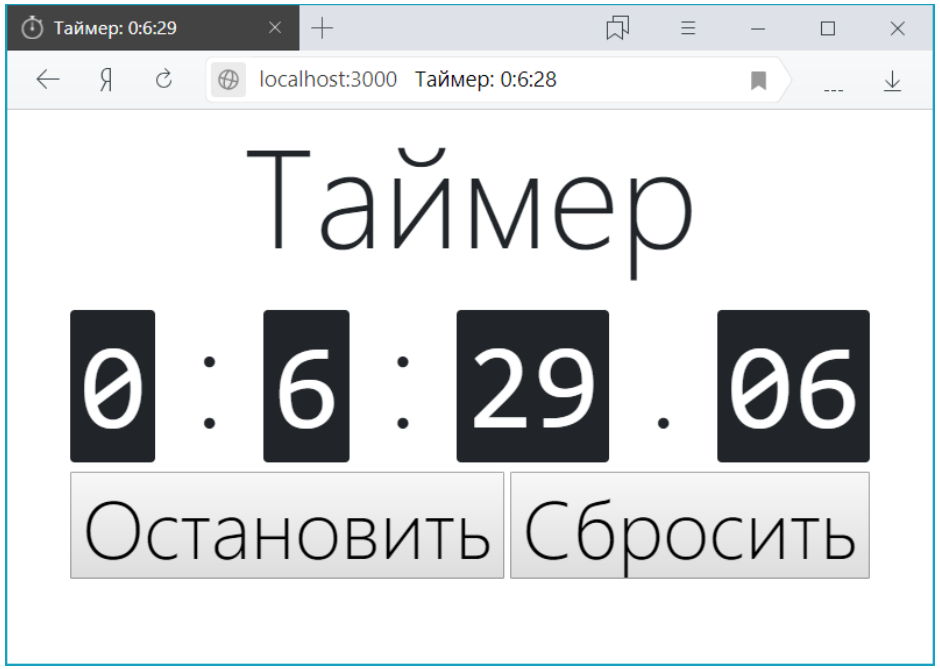
React

- **React** – это библиотека JavaScript, которая используется для создания frontend-приложений.

- Для краткого знакомства с возможностями React можете сделать таймер по статье:

«Начать с React и Bootstrap за 2 дня»

https://habr.com/ru/post/431826/



Установка Node.JS и npm

- Установить Node.JS

https://nodejs.org/en/download/

В него уже встроен npm

- Проверить версию Node.JS

node –v

npm –v

Если вывелись цифры, обозначающие номер версии, то всё ok.

- Если на macOS возникла ошибка, есть решение:

https://support.apple.com/ru-ru/HT202491

Создание проекта

- Перейти в папку, в которой будут проекты:

cd «путь к папке»

- Запустить создание проекта

create-react-app app-name

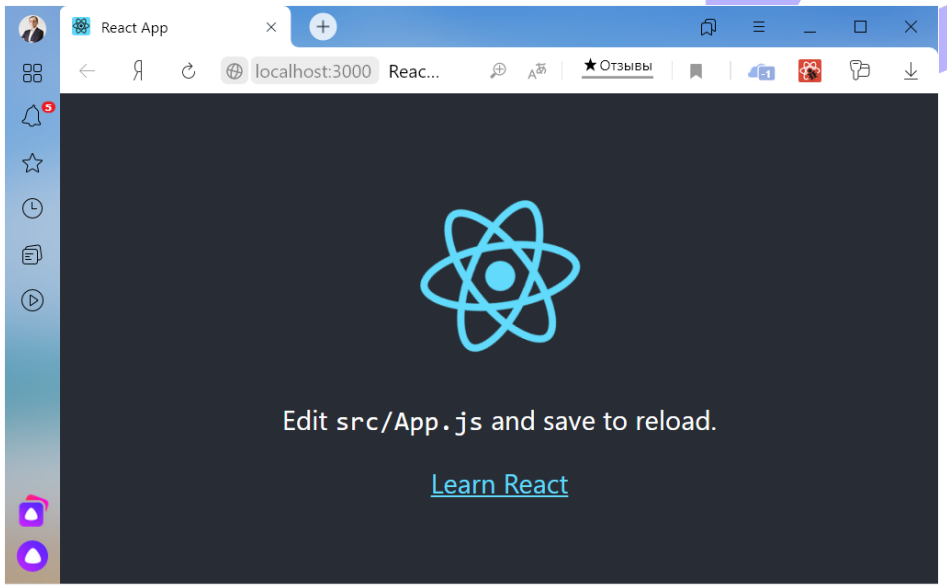
Где **app-name** – название проекта

Запуск проекта

- Перейти в папку созданного проекта

cd app-name

Где app-name – название проекта



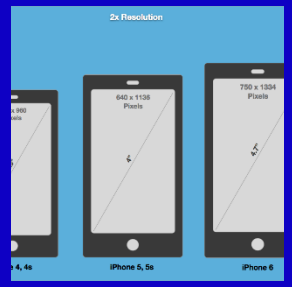
- Запустить проект

npm start

- В итоге откроется браузер со страницей:

http://localhost:3000/

Занятие 2. JSX и компоненты



App

Component

JSX

Hello, World!

- В проекте открыть файл **index.js**

Этот файл находится в папке **src**

- Заменить текст

ReactDOM.render(<App />, document.getElementById('root’));

- На следующий текст

ReactDOM.render(<h1>Hello, world!</h1>, document.getElementById('root'));

- В браузере будет выведено:

Hello, world!

По адресу **http://localhost:3000/**

JSX

- **JSX** – это язык описания интерфейсов в React, который совмещает в себе HTML и JavaScript

- Помимо стандартных тегов HTML, написанных строчными (маленькими) буквами – можно использовать теги своих компонентов, написанные с заглавной(большой) буквы.

Например: **
<UserList/>
**

- Можно подставлять выражения из JavaScript в фигурных скобках

Например: **<img src={user.avatarUrl}></img>**

Ограничения JSX

- Должен быть только 1 корневой тег

Некорректный JSX: **<h1>Привет, Мир!</h1> <Content> **
Корректный JSX: ** <h1>Привет, Мир!</h1> <Content> **

- Не ставьте кавычки в атрибуте при вставке JavaScript

Можно: **<img src="https://test.com/image.jpg"></img>**
Можно: **<img src={user.avatarUrl}></img>**
Некорректно: **<img src="{user.avatarUrl}"></img>**

- JSX предотвращает атаки инъекцией

Например, безопасно:

```
const title = response.potentiallyMaliciousInput;
const element = <h1>{title}</h1>;
```

SYNERGY IT INSTITUTE

Возможности JSX

- JSX это выражение и JavaScript-объект

Его можно вставить внутрь другого JSX

const user = Иван; const userList = {user};

- Можно встраивать функции и любые выражения

Например: **<h1>Hello, {formatName(user)}!</h1>**

- JSX можно заменить чистым JavaScript

Например, следующий JSX и JavaScript равнозначны

```
const element =
<h1 className="greeting">Hello, world!</h1>;
```

```
const element = React.createElement("h1",
{className: "greeting"}, "Hello, world!");
```

DOM vs React DOM

- **DOM** – это древовидная **модель HTML-документа.** Состоит из тегов, являющихся **элементами HTML**

- **React DOM – виртуальная** древовидная **модель**. Состоит из **неизменяемых React-элементов**, являющихся JS-объектами, написанная на JSX. React сам переводит React-элементы в DOM

Отрисовка React-элементов

- В index.html есть строка

** **

Этот элемент называется корневым (root) узлом

Файл index.html находится в проекте в папке public

- Внутрь этого тега будет вставляться React DOM

ReactDOM.render(<App />,document.getElementById('root'));

Для этого нужно вызвать функцию render(), передать в неё JSX и корневой узел

Этот код находится в файле index.js, который лежит в папке src

SYNERGY IT INSTITUTE

Компонент

- **Компонент** - это React-элемент, написанный разработчиком. Обычно часть UI, который содержит свою структуру и функциональность

- В JSX компонент выглядит как обычный тег

Но в отличие от строчных HTML-тегов пишется с заглавной буквы

- Есть 2 способа объявить компонент:
- Через JavaScript-функцию, которая возвращает JSX
- Через ES6-класс, в котором есть функция render(), возвращающая JSX

Примеры объявления компонента

- Вызов компонента в JSX

```
<Welcome name="React"/>
```

- Объявление JavaScript-функции

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

- Объявление ES6-класса

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

Передача параметров компоненту

- Для передачи параметров используется **props**

- В JSX параметры описываются как атрибуты тега

<Welcome name="React"/>

- К параметру можно обратиться через **props**:
- **props.name** – в функциональном компоненте
- **this.props.name** - в компоненте-классе

Чистые функции и read-only props

- **Чистая функция** - это функция, в которой не меняются значения переданных в неё параметров, и для одинакового набора параметров всегда возвращает один и тот же результат

Пример не чистой функции (в ней меняется значение переданного параметра):

```
function withdraw(account, amount) {
    account.total = amount;
}
```

- Все React-компоненты должны работать как чистые функции в отношении своих свойств props

То есть **props** являются read only и не может меняться внутри компонента

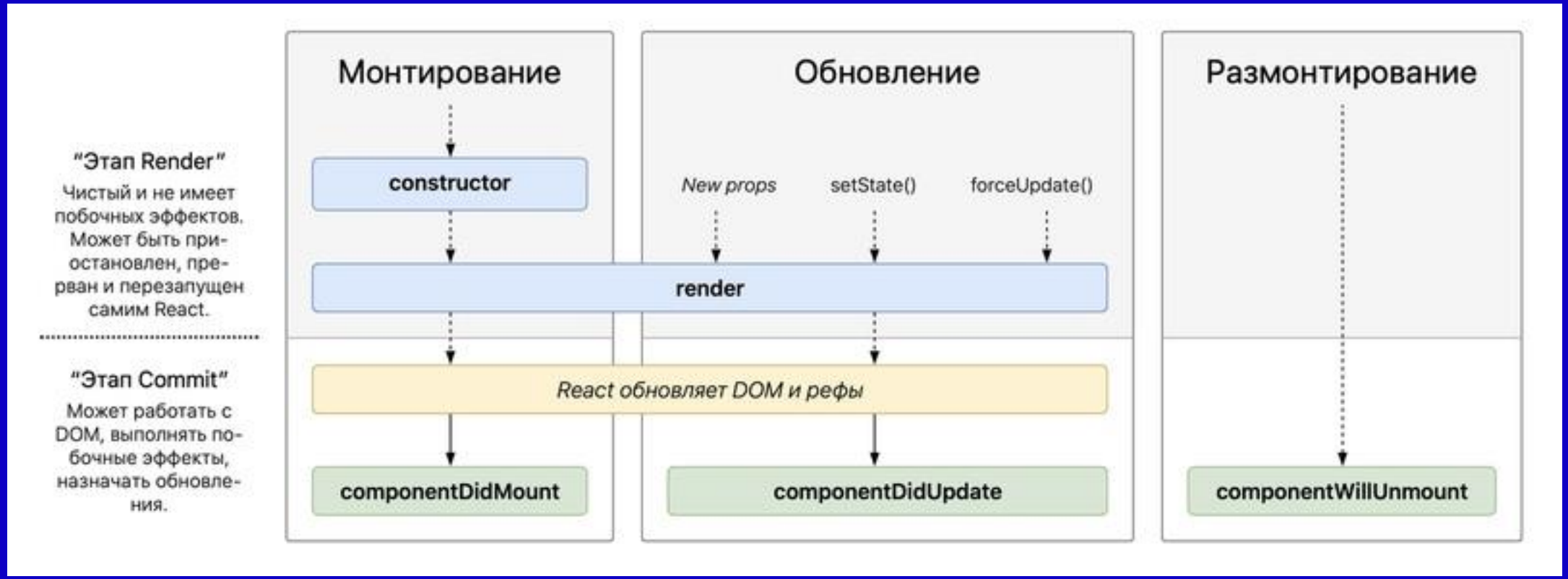
Композиция компонентов

- Одни компоненты могут входить в другие

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}
function App() {
    return (<div>
        <Welcome name="Ivan" />
        <Welcome name=“Petr" />
        <Welcome name=“Alex" />
    </div>);
}
ReactDOM.render(<App />, document.getElementById('root’));
```

Это позволяет выделять компоненты до любого уровня детализации

Занятие 3. События и жизненный цикл



SYNERGY IT INSTITUTE

Вызов обработчика события

- В HTML события называются строчными буквами и указывается функция со скобками

Например:

<button **onclick="deleteAllUsers()"**>Удалить всех пользователей</button>

- В React события называются в верблюжьей нотации и указывается лишь название функции

Например:

<button **onClick={deleteAllUsers}**>Удалить всех пользователей</button>

Обработка события функции

- Достаточно функции

```
function Component(user) {
    function userName() { console.log(user); }
    return (<button onClick={userName}>Пользователь</button>);
}
```

- Если требуется передать ещё параметр, то
- Можно использовать биндинг

```
<button onClick={this.userName.bind(this, id)}>Пользователь</button>
```

- Или можно использовать стрелочную функцию

```
<button onClick={(e) => this.userName(id, e)}>Пользователь</button>
```

Обработка события класса

- Для работы с this в конструкторе класса нужен биндинг

```
class Component extends React.Component {
    constructor(props){ super(props); this.userName = this.userName.bind(this); }
    function userName() { console.log(this.props.user); }
    render() {return (<button onClick={this.userName}>Пользователь</button>);}
}
```

- Либо можно использовать стрелочные функции

```
class Component extends React.Component {
    userName = () =>  { console.log(this.props.user); }
    render() {return (<button onClick={this.userName}>Пользователь</button>);}
}
```

Отключение в обработчике поведения по умолчанию

- В обработчике вызвать функцию e.preventDefault()

Например, чтобы ссылка работала как кнопка без перехода на страницу

```
function DeleteUserLink() {
    function onClick(e) {
        e.preventDefault();
        console.log('Пользователь был удален.');
    }
    return (
        <a href="#" onClick={onClick}>Удалить пользователя</a>
    );
}
```

SYNERGY IT INSTITUTE

Методы жизненного цикла

- **До монтирования** вызывается конструктор **constructor()**
- Когда компонент будет впервые отрисован в DOM – это называется **монтированием/монтажом** компонента. Перед монтированием вызывается метод **componentWillMount()**
- Когда компонент смонтирован, то можно его показывать и производить **обновление/рендеринг/отрисовка** компонента. При этом вызывается метод **render()**
- Сразу **после монтирования** вызывается **componentDidMount()**
- Обратная процедура, при которой DOM, созданный компонентом удаляется, называется **демонтированием/демонтажом**. При демонтаже вызывается метод **componentWillUnmount()**

Эти методы называются «**lifecycle hooks**» или **методами жизненного цикла**

Занятие 4. Состояния

В разработке

Занятие 5. Формы

Описание формы

Форма описывается в render() в JSX, например так:

```
render() {
        return (
                <form onSubmit={this.onSubmit}>
                        <input type=“text” name=“login”/>
                        <input type=“submit” value=“Войти”/>
                </form>
        );
}
```

Хранение данных формы

Данные формы лучше всего хранить в state

```
constructor(props){
    super(props);
    this.state = {login: ''};
}
```