PHASE 3
WEEK 1

DAY 1

ELBRUS/BOOTCAMP

Знакомство с React

План

1. CRA, ES modules (import/export)
2. React, компоненты, элементы
3. JSX
4. React hooks (useState, useEffect)
5. React Strict Mode
6. React Dev Tools

Create React App

Инструмент для создания React-приложений. Устанавливает пакеты для удобной разработки.

```
// создать папку my-app с dev окружением
npx create-react-app my-app --template typescript

// наполнить текущую папку dev окружением
npx create-react-app . --template typescript
```

ES modules

Ранее мы использовали CommonJS модули для разбивки приложения на файлы — с синтаксисом `require/exports`.

Для проектов на CRA будем использовать ES модули — с синтаксисом `import/export`.

```
// App.tsx
export default App;
```

```
// index.tsx
import App from './App';
```

React

JS-библиотека для отрисовки пользовательских интерфейсов (user interfaces, UI)

React

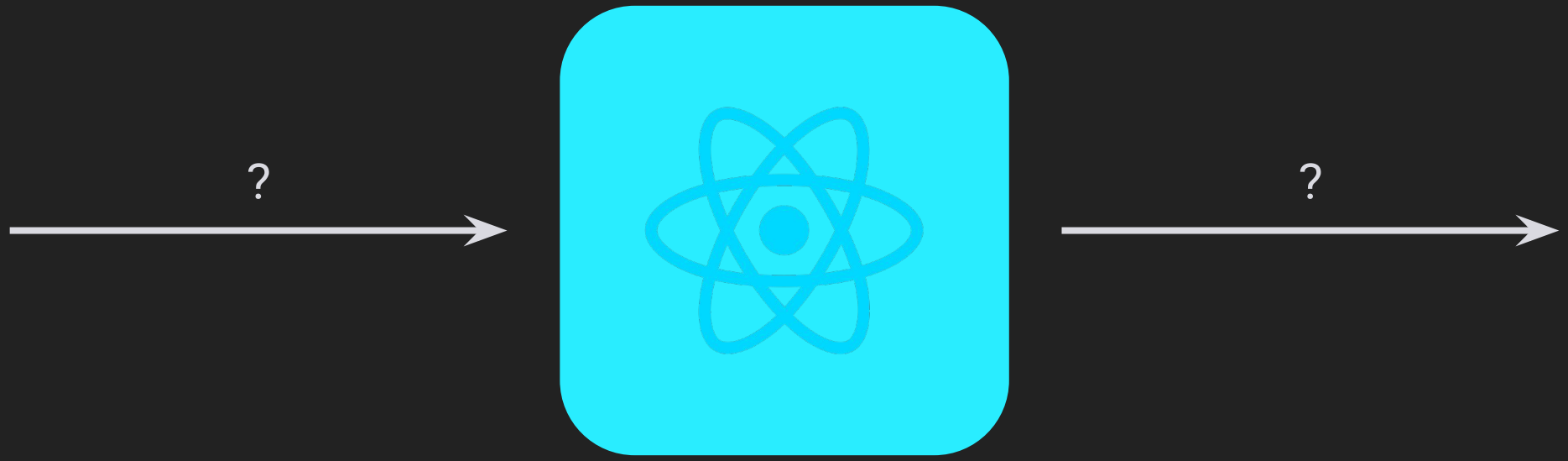
Позволяет разбивать UI на отдельные части — компоненты.

Оптимизирует работу с DOM-деревом.

Позволяет обновлять только те части, что были изменены.

React компонент

Компонент



Компонент

React-компонент — функция или класс для создания *React-элемента*.

React-элемент — объект с описанием того, что React должен отрисовать в DOM-дереве.

Классовый компонент

```
class Button extends React.Component {
  render(): JSX.Element {
    return (
      <button>Learn React!</button>
    )
  }
}

export default Button
```

Функциональный компонент

```
function MyButton(props: {}): JSX.Element {
  return (
    <button>Learn React!</button>
  )
}

export default MyButton
```

Принимает один параметр — объект `props` (свойства)

Создание React-элемента (native)

```
React.createElement(type, props, ...children);
```

`type` – строка с названием HTML-тега или ссылка на React-компонент.

JSX

JavaScript XML

Расширение синтаксиса JavaScript для описания шаблонов в React.

Создание React-элемента через JSX

```
{/* JSX */}
<button name="example">Нажми</button>
  └─┬─┘   └──────┬──────┘   └─┬─┘
   Type        Props      Children
```

JSX-шаблон компилируется в обычный JavaScript-код →

Создание React-элемента

```
// TS
React.createElement('button', { name: 'example' }, 'Нажми');
                    └──┬──┘     └──────┬──────┘     └──┬──┘
                       │               │               │
                     Type            Props          Children
```

Полученный элемент описывает HTML-узел.

Пример компонента с JSX

```
function MyHeading(props: {}): JSX.Element {
    return <h3>{props.title}</h3>
}
```

Компоненты должны называться с большой буквы.

Иначе React попытается отрисовать HTML-элемент с таким названием.

JS in JSX

```
// Можно писать TS-выражения в JSX внутри фигурных скобок
function Two(): JSX.Element {
    return <p>{ 1 + 1 }</p>
}
```

JS in JSX / Условный рендеринг

Условный рендеринг — отрисовка элементов по какому-то условию.

Важно: инструкция `if...else` внутри `return` невозможна.

Наиболее часто используются:

- логический оператор `&&`
- тернарное условие `?:`

JS in JSX / Условный рендеринг &&

```tsx
export function UserBalance({ user }: { user: User }): JSX.Element {
  return (
    <>
      {
        user.balance &&
        <p>{`Баланс ${user.balance} ₽`}</p>
      }
    </>
  )
}
```

JS in JSX / Условный рендеринг ? :

```
export function UserBalance({ user }: { user: User }): JSX.Element {
  return (
    <p>
      {
        user.balance
          ? `Баланс ${user.balance} ₽`
          : 'Пополните счёт'
      }
    </p>
  )
}
```

React.Fragment === <></>

```
import React from "react"

export function SomeComponent():
JSX.Element {
  return (
    <React.Fragment>
      <h1>React is simple!</h1>
      <h2>Example node</h2>
    </React.Fragment >
  )
}
```

Из компонента можно возвращать только один элемент.

Фрагмент используется для обёртки нескольких элементов в один служебный, который не появится в DOM.

Вместо записи:
<React.Fragment></React.Fragment>
можно использовать укороченный вариант <></>

Компонент

props

Сокращение от “properties” (свойства)

Объект со свойствами компонента, передаваемый из родительского компонента.

Свойства могут быть разного типа

```
export function ComponentPile(): JSX.Element {
  return (
    <div>
      <p className="text-dark">Да, тут CSS-класс</p>  {/* строка */}
      <TheAnswer value={42} />                        {/*  число */}
      <MyDog isGoodBoy={true} />                {/* булевое значение */}
      <SomeList items={[1, 2, 3, 4]} />                {/* массив */}
      <Text func={() => 'Text from function!'} />    {/* функция */}
    </div>
  )
}
```

props.children

Доступ к дочерним узлам внутри компонента:

```
export function MyHeading(
    { children }: { children: React.ReactNode }
): JSX.Element {
    return <h3>{children}</h3>;
}
```

`props.children` содержит либо один узел, либо массив узлов. Его тип чаще всего: `React.ReactNode`. Узлы бывают текстовые или в виде React-элемента.

props только для чтения

Свойства (props) предназначены только для чтения.

Их нельзя менять внутри компонента.

В работе с React есть одно строгое правило.

Компоненты должны вести себя **как чистые функции** по отношению к свойствам.

Чистая функция

Функция считается **чистой**, если:

- возвращает одинаковый результат при одинаковых входных параметрах
- не производит побочных эффектов (мутаций)

React элемент

React-элемент

Отрисовка элементов в DOM

```
// DOM-узел, куда отрисовывать результат
const rootElement = document.getElementById('root');

if (rootElement) {
  // Корневой React-элемент для отрисовки
  const root = ReactDOM.createRoot(rootElement);

  // Отрисовка (рендер)
  root.render(<App />);
}
```

Virtual DOM

Virtual DOM, VDOM

Virtual DOM — это концепция программирования, в которой идеальное или «виртуальное» представление пользовательского интерфейса хранится в памяти и синхронизируется с «реальным» DOM при помощи библиотеки, такой как ReactDOM.

React сам превращает корневой компонент и все его дочерние узлы в соответствующие в React-элементы. В результате получается дерево React-элементов — Virtual DOM.

Отрисовка элементов в DOM

Монтирование:

Первичная отрисовка компонента в реальный DOM.

Перед отрисовкой каждый компонент превращается в дерево элементов.

Обновление:

Повторная отрисовка изменённого дерева элементов в DOM.

Дерево элементов меняется при изменении props или состояния компонента.

Монтирование дерева элементов

```
const root = ReactDOM.createRoot(document.getElementById('root')!);
root.render(<App />);
```

При монтировании дерево элементов вставляется как потомок реального DOM-узла.

В идеале использовать условие if для проверки, что такой DOM-узел найден (см. код на нескольких слайдах тому назад).

Здесь вместо условия для “успокоения” TS используется восклицательный знак — !

Reconciliation, согласование обновления

При обновлении VDOM React:

- сравнивает виртуальный DOM с реальным
- применяет изменения точечно

Этот процесс называется согласование (reconciliation)

Diffing Algorithm

Алгоритм сравнения (Diffing Algorithm) предполагает, что разработчик укажет, какие дочерние элементы могут быть стабильными между отрисовками.

Это делается с помощью свойства key.

Key

Ключ нужно указывать для всех дочерних элементов, которые создаются из массива через map.

Ключ должен быть **уникален и стабилен**.

Вопрос: можно ли использовать индекс элемента в качестве ключа?

Используйте стабильные ключи

```
export function Test(): JSX.Element {
  const users = [{ id: 1, name: 'Max' }, { id: 2, name: 'Olga' }]
  // индексы и performance.now() уникальны, но нестабильны, используйте id
из БД
  return (
  <>
    <ul>
      {users.map(user => <li key={user.id}>{user.name}</li>)}
    </ul>
  </>
  )
}
```

React hooks

Виды компонентов

Компоненты могут иметь состояние так и не иметь его.

Синонимы умного и глупого компонента:

- stateful / stateless
- container / presentational
- smart / dumb

Компоненты

Компонент с состоянием отслеживает изменения в нём и отрисовывается (рендерится) заново.

Компонент без состояния — оболочка для статичного контента, данных или других компонентов.

Компоненты

```tsx
// Компонент без состояния
function User(): JSX.Element {
 return (
   <li>Пользователь #1</li>
 )
}

export default User
```

```tsx
// Компонент с props, но без состояния
function User(
  { name }: { name: string }
): JSX.Element {
 return (
   <li>
     {name}
   </li>
 )
}

export default User
```

useState

Хук состояния, позволяет писать “умные” компоненты с состоянием.

https://ru.reactjs.org/docs/hooks-state.html

useState

```
const [value, setValue] = useState(initialState)
```

Принимает стартовое значение (`initialState`) в качестве параметра .

Возвращает массив из переменной состояния (`value`) и функции для её изменения (`setValue`).

useState: newState & prevState

Функция для изменения переменной состояния принимает либо новое значение:

```
setValue(newState)
```

…либо callback-функцию с предыдущим значением в качестве параметра:

```
setValue((prevState) => {/* ... */})
```

useEffect

Хук эффекта, позволяет выполнять побочные эффекты в компоненте.

https://ru.reactjs.org/docs/hooks-effect.html

useEffect

Принимает callback-функцию и массив зависимостей в качестве параметров.

Callback-функция может возвращать другую функцию, которая будет выполнена сразу перед размонтированием (unmount) компонента.

useEffect, example code

```
useEffect(() => {
    /* функция-эффект */

    return () => {
      /* функция
      очистки
      эффекта */
    }
}, [/* массив-зависимостей */])
```

useEffect, массив зависимостей

Массив зависимостей:

- Если отсутствует — эффект запускается при каждом рендере. Чревато бесконечным циклом.
- Если пустой — эффект сработает только при первом рендере.
- Если содержит ссылки на переменные — эффект запустится при первом рендере и при изменении переменных, от которых он зависит.

useState, example code

```
import { useState } from 'react'
import './style.css'

export function Card(): JSX.Element {
 const [isActive, setIsActive] = useState(false)
 const changeStyle = () => setIsActive(!isActive)

 return (
   <div className={isActive ? 'card active' : 'card default'}
onClick={changeStyle}>Card #1</div>
 )
}
```

useEffect, unmount

Размонтирование компонента, unmount — момент, когда полученное из компонента дерево элементов удаляется из реального DOM-дерева.

useEffect, example code

```
import { useState, useEffect } from 'react'

export function App(): JSX.Element {
  const [date, setDate] = useState(new Date())
  useEffect(() => {
    const timer = setTimeout(() => setDate(new Date()), 1000)
    return () => clearTimeout(timer)
  }, [date])

  return (
    <div className="App">{date.toLocaleString()}</div>
  )
}
```

Альтернативы React

- Vue.js
- Angular
- Svelte
- Ember.js

React Strict Mode

React Strict Mode

`StrictMode` — инструмент для обнаружения потенциальных проблем в приложении. Также как и `Fragment`, `StrictMode` не рендерит видимого UI. Строгий режим активирует дополнительные проверки и предупреждения для своих потомков.

Важно!

Проверки строгого режима работают только в режиме разработки; они не оказывают никакого эффекта в продакшен-сборке.

React Strict Mode, example code

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './components/App/App';

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

React Strict Mode, применение

На данный момент StrictMode помогает в:

- Обнаружении небезопасных методов жизненного цикла
- Предупреждении об использовании устаревшего API строковых реф
- Предупреждении об использовании устаревшего метода findDOMNode
- Обнаружении неожиданных побочных эффектов
- Обнаружении устаревшего API контекста

Подробная информация: https://ru.reactjs.org/docs/strict-mode.html

React Dev Tools

React Dev Tools

Расширение для Chrome и Firefox, упрощает работу с React-приложениями.

https://github.com/facebook/react/tree/master/packages/react-devtools

VS Code Extensions

Reactjs code snippets

https://marketplace.visualstudio.com/items?itemName=xabikos.ReactSnippets

Основные алиасы сниппетов:

- `rsf` — создаёт функциональный компонент
- `rcfc` — создаёт классовый компонент