УДК 004.42

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ В ВЕБ-РАЗРАБОТКЕ НА ПРИМЕРЕ «REACT»

[1]Кудласевич Д. Р., [2]Кондратёнок Е. В.
[1]Беларусский национальный технический университет,
Минск, Беларусь, dilapsor12@gmail.com,
[21]Беларусский национальный технический университет,
Минск, Беларусь, elena_kondr@tut.by

Аннотация. Рассмотрены особенности библиотека React на примере проекта интернет-магазина. Описан компонентный подход к созданию приложений Virtual DOM

Ключевые слова: библиотека React, веб-технологии, веб-разработка, JSX, JavaScript.

Abstract. The features of the React library are considered using the example of an online store project. Describes a component-based approach to creating Virtual DOM applications.

Key words: React library, web technologies, web development, JSX, JavaScript.

В настоящее время веб-технологии находятся на пике своего развития. Веб-приложения уже начинают полностью поглощать десктопные программы. Последние необходимо устанавливать на компьютер пользователя, где они запускаются локально и выполняют свой код. Веб-приложения же, как правило, не требуют инсталляции дополнительного программного обеспечения на рабочую станцию. Существует два основных способа реализации веб-приложений: классический и с использованием различных инструментов, таких как Angular, React, Vue и другие.

Классический способ написания веб-приложений появился еще во времена создания интернета. Под ним понимают написание CSS и JavaScript файлов с их последующим объединением в HTML документе. Несмотря на то, что этот подход достаточно старый, его все еще используют из-за низкого порога вхождения, а также для поддержки старых проектов. С помощью данного способа разработки трудно реализовывать интерактивные веб-приложения, он больше подходит для написания статических страниц [1].

Для облегчения разработки и появления динамических страниц стали создаваться различные библиотеки JavaScript, в частности – React.

Структура проекта.

Рассмотрим особенности библиотеки на примере проекта интернет-магазина продажи компьютерных комплектующих (рис. 1).

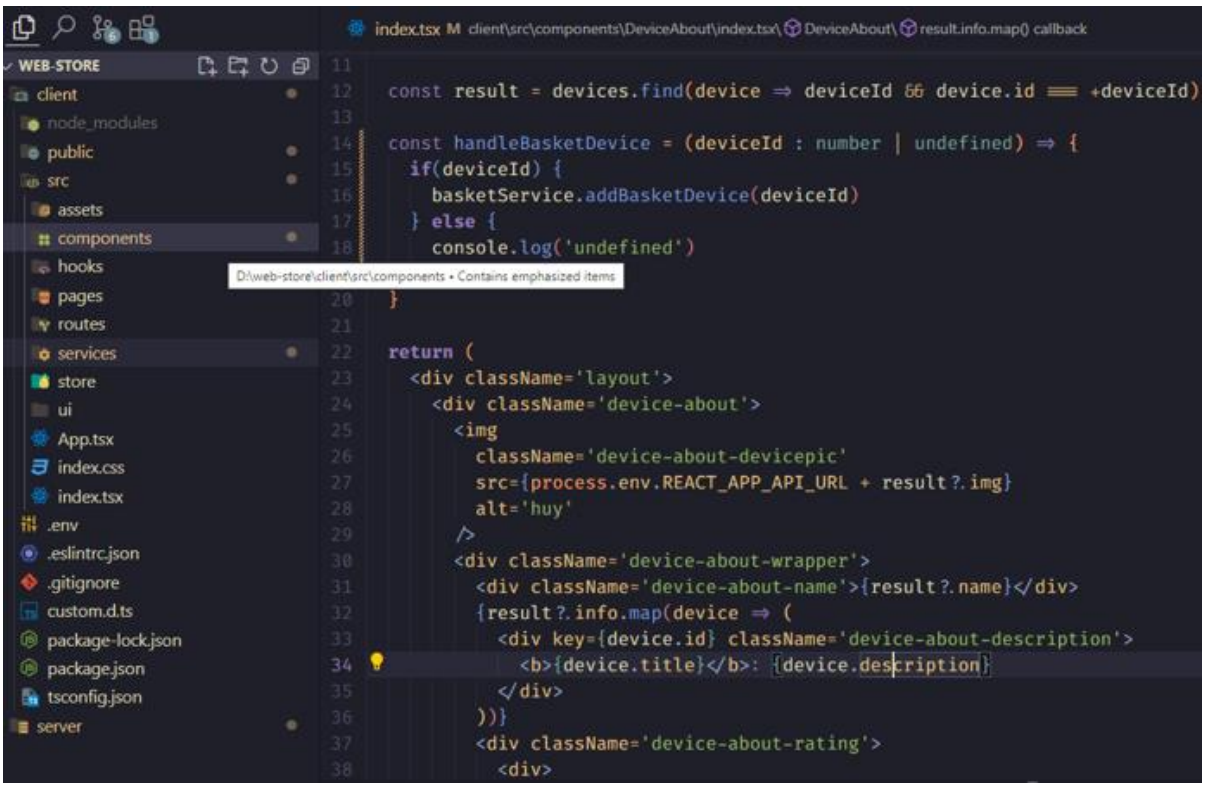

Рисунок 1 – Структура проекта

Одной из главных особенностей библиотеки является отсутствие свода правил ведения единого архитектурного стиля и общепринятой структуры проекта. В каждой компании, разрабатывающей на React может быть принята своя структура, которой следуют разработчики. Это может расцениваться как плюс и как минус, в зависимости от задачи, которая стоит перед командой и от окончательного продукта. Однако несмотря на то, что сами разработчики данной технологии не диктуют правила структурирования проекта, в сообществе есть общепринятые негласные правила архитектуры папок и композиции в приложении. При инициализации клиентской части приложения автоматически создаются папки public, src и несколько файлов конфигурации. Папка public содержит в себе главный HTML проект и несколько файлов для SEO-оптимизации. Все основные файлы приложения располагаются в папке src [2].

Компонентный подход.

Как было сказано выше, раньше при разработке приходилось подключать отдельные JavaScript файлы к HTML, что имело существенный недостаток в виде невозможности вносить изменения «на лету» и изменять лишь какую-то часть страницы, а не всю целиком. С приходом React на рынок веб-разработки все изменилось, так как был представлен новый подход к созданию приложений – компонентный.

Для того, чтобы понять смысл компонентного подхода рассмотрим, что такое сам компонент.

Компонент – независимый модуль приложения или независимый кусок кода, который можно переиспользовать в любом месте приложения неограниченное количество раз. Компонентный подход в свою очередь заключается в создании приложения, основанном на таких компонентах.

```
export default function BasketDevice({ basketDevice }: BasketDeviceProps) {
  const deleteBasketDevice = (data : number) => {
    basketService.deleteBasketDevice(data)
  }

  console.log(basketDevice)

  return (
    <div className='basket-device'>
      <div className='basket-device-wrapper'>
        <img
          className='basket-device-img'
          src={process.env.REACT_APP_API_URL + basketDevice.img}
          alt='basketpic'
        />
        <div className='basket-device-name'>{basketDevice.name}</div>
      </div>
      <form className='basket-device-form'>
        <div className='basket-device-price'>
          {new Intl.NumberFormat('ru-RU', {
            style: 'currency',
            currency: 'USD',
          }).format(basketDevice.price)}
        </div>
        <button className='basket-device-button' onClick={() =>
        deleteBasketDevice(basketDevice.idInBasket)}>-</button>
      </form>
    </div>
  )
```

Рисунок 2 – Функциональный компонент

В React используется два типа компонентов:

1. Функциональный;
2. Классовый.

Второй способ на данный момент считается устаревшим и практически не используется, поэтому рассмотрим функциональный вариант создания компонента. На рис. 2 продемонстрирован пример функционального компонента. Каждый новый компонент начинает свое существование одинаково. Сначала определяется шаблон React.js для создания элементов из компонентов. Указывается, где используется шаблон. Например, внутри вызова функции рендеринга другого компонента или с помощью ReactDOM.render React создает экземпляр элемента и передает ему набор свойств (props). Затем React монтирует компонент, взаимодействует с браузером через DOM API, и React отображает компонент.

Хуки (Hooks).

Когда функциональные компоненты лишь начинали вытеснять классовые, они не имели всех важных составляющих требуемых для написания качественных компонентов. Они не имели ни состояния, ни методы жизненного цикла. Такое положение подтолкнуло разработчиков React придумать реше-

ние данной проблемы в виде хуков, которые в последствии стали основой разработки на долгие годы вперед. Но для начала опишем что такое жизненный цикл компонента и его состояние.

Компонент может находиться лишь в одном состоянии одновременно и имеет несколько этапов перехода из одного состояния в другое:

1. Монтирование.
2. Обновление.
3. Размонтирование.

Все вместе эти этапы и составляют жизненный цикл компонента. Однако, как известно, функция не имеет состояний, в отличие от объекта класса, именно поэтому для придания «жизни» функциональным компонентам были придуманы хуки. Они позволяют имитировать жизненный цикл у функционального компонента. В React существует несколько основных хуков:

– useEffect();
– useState();
– useCallback();
– useMemo();
– useRef().

Кроме того, мы можем создавать пользовательские хуки, которые могут объединять в себя функционал основных хуков для создания продвинутой логики. Рассмотрим два самых часто используемых хука на реальном примере – useState() и useEffect() [3].

Мы уже выяснили, что функциональный компонент априори не имеет состояний, поэтому мы не можем изменять вложенные свойства и переменные напрямую. Для этого можно воспользоваться хуком useState(), который добавляет состояние в функциональный компонент и позволяет изменять его при каждом ререндере (рис. 3).

```
export default function Shop() {
  const { devices } = useAppSelector(state ⇒ state.device)
  const { role } = useAppSelector(state ⇒ state.user)
  const [active, setActive] = useState<boolean>(false);

  const handleModalWindow = () ⇒ {
    setActive(true)
  }

  const closeModalWindow = () ⇒ {
    setActive(false)
  }
```

Рисунок 3 – Хук useState()

Синтаксис состоит в следующем – в квадратных скобках указаны два аргумента, первый – переменная-состояние, которую нам требуется изменять при отрисовке страницы, второй – функция изменения этого состояния, так называемый сеттер(setter). В скобках самого useState() мы указываем начальное состояние при монтировании компонента. Затем в нужном месте разметки

мы вызываем нашу функцию изменения состояния, например при нажатии кнопки и таким образом изменяем значение находящееся в нашей переменной.

Хук useEffect() позволяет имитировать жизненный цикл функционального компонента и выполнять различные действия на каждом этапе (рис. 4).

Рисунок 4 – Хук useEffect()

Хук принимает в себя стрелочную функцию, которая содержит в себе логику которую необходимо выполнить, и массив зависимостей, при изменении которых и выполняется код внутри стрелочной функции. В данном примере массив зависимостей – пустой массив, а значит хук будет имитировать работу этапа монтирования или же componentDidMount и вся логика выполнится только один раз при рендере компонента вместе со страницей. Если же мы хотим сымитировать работу этапа обновления, нам нужно передать в массив зависимостей переменную-состояние или же функцию, при изменении или вызове которых код внутри стрелочной функции будет вновь выполняться [4].

Манипуляции с DOM и Virtual DOM.

В React объектная модель документа (DOM) и виртуальная DOM (VDOM) являются ключевыми концепциями, направленными на оптимизацию производительности рендеринга веб-приложений.

DOM служит интерфейсом программирования, который представляет структуру документа в виде дерева объектов, обычно в контексте HTML или XML-документов. Он позволяет скриптам динамически получать доступ и изменять содержимое, структуру и стиль документа.

С другой стороны, виртуальный DOM – это облегченное, хранимое в памяти представление реальных элементов DOM. React использует виртуальный DOM для повышения производительности за счет минимизации прямых манипуляций с реальным DOM. Когда происходит изменение состояния компонента React, генерируется новое дерево Virtual DOM. Затем это новое дерево сравнивается с предыдущим, и вычисляются только различия (этот процесс

известен как diffing). Затем путем сверки определяется минимальный набор изменений, необходимых для обновления реального DOM, и эти изменения применяются, снижая общие вычислительные затраты.

Использование виртуального DOM в React дает несколько преимуществ, включая оптимизацию производительности за счет сокращения прямых манипуляций с реальным DOM, пакетные обновления для более эффективного рендеринга, а также декларативный синтаксис, который упрощает выражение состояния пользовательского интерфейса, позволяя библиотеке управлять обновлениями без проблем. Это сочетание способствует более плавному и производительному взаимодействию с пользователем. (рис. 5).

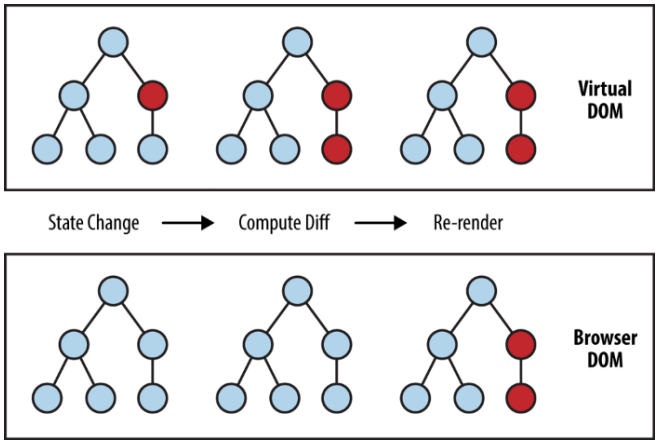

Рисунок 5 – Процесс согласования VDOM с DOM

JSX.

Как можно заметить на рис. 2, функциональный компонент возвращает HTML разметку, что может сбивать с толку, однако это не так. Данная особенность React называется JSX и является по своей сути синтаксическим сахаром обертки над функцией React.createElement() для упрощения написания кода:

JSX код:

```
<CustomButton variant=’outline’ type=’submit’>
  Click Me
</MyButton>
```

компилируется в:

```
React.createElement(CustomButton,
  {variant: ‘outline’, type: ‘submit’},
  'I am custom button')
```

Начальная часть JSX тега определяет тип элемента React.

Типы, которые определяются со строчной буквы являются обычными HTML тегами, если же вы указываете тег который начинаются с заглавной буквы, то на моменте компиляции он будет воспринят как React-компонент. Из особенностей JSX можно выделить несколько составляющих:

1. Так как JSX представляет собой JavaScript, то атрибут класса тега нужно указывать как className, а не class, соответствующий обычному HTML.

2. React-компонент позволяет возвращать лишь один корневой элемент, поэтому вся разметка написанная в компоненте должна быть обернута в единственный общий элемент, например .

3. Встраивание в разметку JS выражений происходит посредством фигурных скобок {…}:

{basketDevice.name} .

Заключение.

React предоставляет разработчикам гибкий подход к созданию веб-приложений, оставляя свободу в организации структуры проекта. Несмотря на отсутствие строгих правил, в сообществе существуют распространенные практики. Компонентный подход, введенный данной библиотекой сфокусирован на создании независимых и переиспользуемых элементов. Функциональные компоненты, особенно с использованием хуков, стали предпочтительным выбором разработчиков. Хуки useState() и useEffect() играют ключевую роль в управлении состоянием и имитации жизненного цикла компонентов. Использование Virtual DOM в React улучшает производительность, обеспечивая более эффективное управление изменениями в реальном DOM. Таким образом, React предоставляет разработчикам мощные инструменты для создания гибких и производительных веб-приложений, а компоненты и хуки облегчают процесс разработки, делая его более эффективным и удобным.

Список использованных источников:

1. Сучков, А. А. Использование ReactJS в современной web-разработке / А. А. Сучков, Д. К. Гэк, А. П. Багаева //Актуальные проблемы авиации и космонавтики. – 2019. – Т. 2.

2. React [Электронный ресурс]. – Режим доступа: https://ru.legacy.reactjs.org/docs/faq-structure.html. – Дата доступа: 05.11.2023.

3. React Hooks простыми словами [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/simbirsoft/articles/652321/. – Дата доступа: 05.11.2023.

4. Краткий обзор хуков [Электронный ресурс]. – Режим доступа: https://ru.legacy.reactjs.org/docs/hooks-overview.html. – Дата доступа: 05.11.23.

5. Немного о том, как работает виртуальный дом в React [Электронный ресурс] – Режим доступа: https://habr.com/ru/companies/macloud/articles/558/. – Дата доступа: 05.11.2023.

6. Основы использования JSX в React [Электронный ресурс]. – Режим доступа: https://itchief.ru/react/jsx. – Дата доступа: 05.11.2023.