ДИСЦИПЛИНА	Фронтенд и бэкенд разработка
ИНСТИТУТ	ИПТИП
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Методические указания к практическим занятиям
ПРЕПОДАВАТЕЛЬ	Астафьев Рустам Уралович
СЕМЕСТР	1 семестр, 2025/2026 уч. год

Ссылка на материал:
https://github.com/astafiev-rustam/frontend-and-backend-development/tree/practice-1-19

Практическое занятие 19: Базовые элементы работы с React.js и JSX

В рамках данного занятия рассмотим процесс установки, использования и запуска приложений на React, а также основы JSX.

Подробная теоретическая информация содержится в лекции, а также можно ознакомиться с официальной документацией по React:

https://react.dev/learn

Запуск проектов на React

Перед началом практической работы важно понять базовые принципы React. React — это библиотека для создания пользовательских интерфейсов, которая использует компонентный подход. Каждый компонент представляет собой независимую часть интерфейса со своей логикой и внешним видом.

Основной единицей в React является компонент, который может быть как функцией, так и классом. Мы будем использовать функциональные компоненты, так как это современный и рекомендуемый подход. JSX — это синтаксическое расширение для JavaScript, которое позволяет писать HTML-подобный код внутри JavaScript. Важно помнить, что JSX не является ни строкой, ни HTML, а компилируется в вызовы React.createElement.

Для работы с React потребуется настроенная среда разработки. Мы используем Create React App — официальный инструмент для быстрого создания React-приложений с предварительно настроенной средой разработки.

Для его использования необходимо скачать Node.js со встроенным пакетным менеджером npm, который мы будем использовать далее.

Ссылка для Node.js:

https://nodejs.org/en/download

После установки выполним следующие команды.

1. Проверим версию npm и корректную установку Node.js:

   ```
   npm --v
   ```

   В случае возникновения ошибок необходимо скопировать вывод консоли в поисковик и/или нейросеть для анализа и устранения ошибок.

2. Выполните команду создания проекта:

   ```
   npx create-react-app test-app
   ```

   Помимо проекта эта команда создаст папку test-app, в которую мы переходим и выполняем запуск приложения.

3. Перейдите в папку проекта и выполните запуск приложения:

   ```
   cd test-app
   npm start
   ```

   Просмотрите результат. Именно по данному результату мы с вами и будем выполнять сегодняшние примеры.

Теоретическая часть

Пример 1. Создание простого компонента с JSX

Проблема, которую мы решаем: необходимо создать компонент, который отображает приветствие для пользователя. В обычном HTML мы бы просто написали разметку, но в React нам нужно создать компонент, который можно переиспользовать и который содержит логику.

Подход к решению заключается в создании функционального компонента, который возвращает JSX. JSX позволяет нам комбинировать JavaScript и HTML-подобный синтаксис для описания того, что мы хотим видеть на экране.

Исходный код решения в файле `Greeting.jsx`:

```
function Greeting() {
  const userName = "Айнура";
  const currentTime = new Date().getHours();
  let timeOfDay;

  if (currentTime < 12) {
    timeOfDay = "Доброе утро";
```

```
  } else if (currentTime < 18) {
    timeOfDay = "Добрый день";
  } else {
    timeOfDay = "Добрый вечер";
  }

  return (
    <div className="greeting-container">
      <h1>{timeOfDay}, {userName}!</h1>
      <p>Рады видеть вас в нашем приложении.</p>
    </div>
  );
}

export default Greeting;
```

Пояснения к решению: мы создаем функциональный компонент Greeting, который содержит логику определения времени суток и возвращает JSX с приветствием. Ключевой особенностью является то, что мы можем встраивать JavaScript-выражения в JSX с помощью фигурных скобок. Для запуска этого кода необходимо импортировать и использовать компонент в основном файле приложения.

Содержимое файла `App.js`:

```
import logo from './logo.svg';
import './App.css';
import Greeting from './Greeting';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
      <Greeting />
    </div>
  );
}

export default App;
```

Пример 2. Компонент с свойствами (props)

Проблема: нам нужен универсальный компонент карточки пользователя, который может отображать информацию о разных пользователях без изменения своего кода.

Подход к решению: используем механизм props (свойств) в React для передачи данных в компонент. Это позволяет сделать компонент переиспользуемым и независимым от конкретных данных.

Исходный код решения в файле `UseCard.jsx`:

```
function UserCard({ name, role, avatarUrl, isOnline }) {
  return (
    <div className="user-card">
      <div className="avatar-section">
        <img src={avatarUrl} alt={`Аватар ${name}`} />
        <p>Статус: {isOnline? 'online':'offline'}</p>
      </div>
      <div className="user-info">
        <h3>{name}</h3>
        <p>{role}</p>
      </div>
    </div>
  );
}

export default UserCard;
```

Пояснения к решению: компонент UserCard принимает свойства name, role, avatarUrl и isOnline через объект props. Мы используем деструктуризацию для извлечения этих свойств. Компонент отображает переданные данные в соответствующей разметке. Для использования этого компонента нужно передать ему свойства при вызове.

Добавление компонента UseCard в файле `App.js`:

```
import logo from './logo.svg';
import './App.css';
import Greeting from './Greeting';
import UserCard from './UseCard';
import TaskList from './TaskList';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
```

```
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
        <Greeting />
        <UserCard
        name="Иван Иванов"
        role="Администратор"
        avatarUrl="https://encrypted-tbn0.gstatic.com/images?
q=tbn:ANd9GcRfVMhpKmVy_-iwfRLAiNiaDslMa-2oEz7KTw&s"
        isOnline={true}
        />
      </div>
    );
}

export default App;
```

Пример 3. Работа со списками и ключами

Проблема: необходимо отобразить список элементов, где каждый элемент имеет собственные данные и должен быть уникально идентифицирован.

Подход к решению: используем метод map для преобразования массива данных в массив JSX-элементов. Каждому элементу списка назначаем уникальный ключ (key), что помогает React эффективно обновлять интерфейс при изменениях.

Исходный код решения в файле `TaskList.jsx`:

```
function TaskList() {
  const tasks = [
    { id: 1, title: 'Изучить JSX', completed: true },
    { id: 2, title: 'Разобраться с компонентами', completed: false },
    { id: 3, title: 'Освоить работу с props', completed: false },
    { id: 4, title: 'Отклеить этикетки от бананов', completed: false }
  ];

  return (
    <div className="task-list">
      <h2>Список задач</h2>
      <ul>
        {tasks.map(task => (
          <li key={task.id} className={task.completed ? 'completed' : 'pending'}>
            <span>{task.title}</span>
            {task.completed ? '☑' : '⌛'}
          </li>
        ))}
```

```
      </ul>
    </div>
  );
}

export default TaskList;
```

Пояснения к решению: мы создаем массив задач и с помощью метода map преобразуем его в массив JSX-элементов. Ключевой аспект — атрибут key, который должен быть уникальным для каждого элемента списка. Это помогает React определить, какие элементы изменились, добавились или удалились. Условный рендеринг используется для отображения разных иконок в зависимости от статуса задачи.

Встраивание в `App.js`:

```
import logo from './logo.svg';
import './App.css';
import Greeting from './Greeting';
import UserCard from './UseCard';
import TaskList from './TaskList';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
      <Greeting />
      <UserCard
      name="Иван Иванов"
      role="Администратор"
      avatarUrl="https://encrypted-tbn0.gstatic.com/images?
q=tbn:ANd9GcRfVMhpKmVy_-iwfRLAiNiaDslMa-2oEz7KTw&s"
      isOnline={true}
      />
      <TaskList />
    </div>
  );
}
```

```
export default App;
```

Практическая часть

Перед выполнением практической части стоит ознакомиться с содержимым файла `task.pdf` в этом репозитории.

Теперь приступим к созданию нашего основного проекта - трекера изучения технологий. На этом занятии мы заложим фундамент приложения, создав базовые компоненты для дальнейшей разработки.

Начало работы

Шаг 1: Создание проекта Откройте терминал и выполните команду:

```
npx create-react-app technology-tracker
cd technology-tracker
npm start
```

Это создаст новый React-проект и запустит сервер разработки. Проверьте, что приложение открывается в браузере по адресу http://localhost:3000

Шаг 2: Организация структуры проекта Создайте в папке `src` следующую структуру:

```
src/
├── components/
│   ├── TechnologyCard.js
│   └── TechnologyCard.css
├── App.js
├── App.css
└── index.js
```

Такая организация поможет поддерживать порядок в коде по мере роста приложения.

Создание компонента карточки технологии

Шаг 3: Разработка компонента TechnologyCard В файле `TechnologyCard.jsx` создайте компонент, который будет отображать отдельный пункт дорожной карты. Компонент должен принимать следующие свойства:

- `title` - название технологии
- `description` - описание для изучения
- `status` - статус изучения

Используйте подход из второго теоретического примера, адаптировав его под нашу задачу. Не забудьте добавить условное отображение для разных статусов.

Шаг 4: Стилизация карточки В файле `TechnologyCard.css` создайте базовые стили. Важно предусмотреть визуальное различие для различных статусов изучения:

- Используйте разные цвета границ или фона
- Добавьте иконки или индикаторы прогресса
- Обеспечьте четкое визуальное разделение карточек

Сборка основного приложения

Шаг 5: Создание тестовых данных В компоненте `App.js` создайте массив с тестовыми данными для проверки работы. Пример структуры данных:

```
const technologies = [
  { id: 1, title: 'React Components', description: 'Изучение базовых компонентов',
status: 'completed' },
  { id: 2, title: 'JSX Syntax', description: 'Освоение синтаксиса JSX', status:
'in-progress' },
  { id: 3, title: 'State Management', description: 'Работа с состоянием
компонентов', status: 'not-started' }
];
```

Шаг 6: Отображение списка технологий Используя подход из третьего теоретического примера, отобразите массив технологий с помощью компонента `TechnologyCard`. Не забудьте передавать все необходимые свойства и назначить уникальные ключи для каждого элемента списка.

Доработка интерфейса

Шаг 7: Базовая стилизация приложения В файле `App.css` добавьте стили для основного контейнера приложения:

- Задайте максимальную ширину контейнера
- Добавьте отступы для лучшего восприятия
- Определите основные шрифты и цвета

Шаг 8: Проверка работы Убедитесь, что приложение корректно отображает все созданные карточки с разными статусами. Проверьте:

- Отображаются ли все переданные данные
- Работает ли условное отображение статусов
- Корректно ли применяются стили

Самостоятельная работа

Задание для самостоятельного выполнения: Создайте компонент `ProgressHeader`, который будет отображать общую статистику по дорожной карте. Компонент должен показывать:

- Общее количество технологий
- Количество изученных технологий
- Процент выполнения в виде прогресс-бара

Рекомендации по выполнению:

- Используйте полученные знания о работе с props
- Примените подход условного рендеринга для разных состояний прогресса
- Рассчитайте процент выполнения на основе количества технологий со статусом 'completed'
- Добавьте визуальный прогресс-бар с помощью CSS

Что проверить перед завершением:

- Все компоненты импортируются и экспортируются корректно
- Приложение запускается без ошибок
- Карточки отображаются с правильными данными и статусами
- Стили применяются ко всем элементам

По завершении этой работы у вас будет основа для трекера изучения технологий, который мы будем развивать на следующих занятиях. Не стремитесь к идеальному дизайну - сейчас важнее работоспособность и правильная архитектура компонентов.