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-1-22

Практическое занятие 22: Использование локального хранилища и переиспользование компонентов

В рамках данного занятия будут рассмотрены возможности переиспользования компонентов. Подробную информацию об этом можно найти в материалах лекций, а также в материалах:

https://habr.com/ru/companies/docsvision/articles/694774/
https://habr.com/ru/companies/yandex/articles/560194/

Теоретическая часть

Пример 1. Создание кастомного хука для localStorage

Проблема: Нужно создать переиспользуемую логику для работы с localStorage, чтобы избежать повторения кода в разных компонентах.

Подход к решению: Создаем кастомный хук useLocalStorage, который инкапсулирует логику чтения/записи в localStorage.

Исходный код в файле `useLocalStorage.js`:

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

// Кастомный хук для работы с localStorage
function useLocalStorage(key, initialValue) {
  // Инициализируем состояние, пытаясь получить значение из localStorage
  const [storedValue, setStoredValue] = useState(() => {
    try {
      // Пытаемся получить значение по ключу из localStorage
      const item = window.localStorage.getItem(key);
      // Если значение найдено, парсим его из JSON, иначе используем initialValue
      return item ? JSON.parse(item) : initialValue;
```

```
    } catch (error) {
      // В случае ошибки (например, недоступный localStorage) используем
initialValue
      console.error(`Ошибка чтения из localStorage ключа "${key}":`, error);
      return initialValue;
    }
  });

  // Функция для обновления значения в состоянии и localStorage
  const setValue = (value) => {
    try {
      // Разрешаем value быть функцией, как в useState
      const valueToStore = value instanceof Function ? value(storedValue) : value;

      // Сохраняем в состояние
      setStoredValue(valueToStore);

      // Сохраняем в localStorage
      window.localStorage.setItem(key, JSON.stringify(valueToStore));
    } catch (error) {
      console.error(`Ошибка записи в localStorage ключа "${key}":`, error);
    }
  };

  return [storedValue, setValue];
}

export default useLocalStorage;
```

Пример использования кастомного хука:

```
// UserSettings.jsx - компонент настроек пользователя
import useLocalStorage from './useLocalStorage';

function UserSettings() {
  // Используем наш кастомный хук для разных настроек
  const [username, setUsername] = useLocalStorage('username', 'Гость');
  const [theme, setTheme] = useLocalStorage('theme', 'light');
  const [notifications, setNotifications] = useLocalStorage('notifications',
true);

  return (
    <div className="user-settings">
      <h2>Настройки пользователя</h2>

      <div className="setting-group">
        <label>Имя пользователя:</label>
        <input
          type="text"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          placeholder="Введите ваше имя"
```

```
          />
        </div>

        <div className="setting-group">
          <label>Тема оформления:</label>
          <select value={theme} onChange={(e) => setTheme(e.target.value)}>
            <option value="light">Светлая</option>
            <option value="dark">Темная</option>
            <option value="auto">Авто</option>
          </select>
        </div>

        <div className="setting-group">
          <label>
            <input
              type="checkbox"
              checked={notifications}
              onChange={(e) => setNotifications(e.target.checked)}
            />
            Включить уведомления
          </label>
        </div>

        <div className="current-settings">
          <h3>Текущие настройки:</h3>
          <p>Имя: {username}</p>
          <p>Тема: {theme}</p>
          <p>Уведомления: {notifications ? 'Включены' : 'Выключены'}</p>
        </div>
      </div>
    );
}

export default UserSettings;
```

После добавления в компонент App проверяем через средства разработчика LocalStorage и проверяем корректность работы.

Пример 2. Создание переиспользуемого компонента модального окна

Проблема: Нужно создать универсальный компонент модального окна, который можно использовать в разных частях приложения с разным содержимым.

Подход к решению: Создаем компонент Modal, который принимает содержимое через children и props для управления видимостью.

Исходный код в файле `Modal.jsx`:

```
import './Modal.css';

// Простой переиспользуемый компонент модального окна
```

```jsx
function Modal({ isOpen, onClose, title, children }) {
  // Если модалка закрыта - не показываем ничего
  if (!isOpen) {
    return null;
  }

  // Функция для закрытия модалки при клике на фон
  const handleBackgroundClick = (event) => {
    if (event.target === event.currentTarget) {
      onClose();
    }
  };

  return (
    <div className="modal-background" onClick={handleBackgroundClick}>
      <div className="modal-window">
        {/* Шапка модалки с заголовком и кнопкой закрытия */}
        <div className="modal-header">
          <h2>{title}</h2>
          <button className="close-button" onClick={onClose}>
            ×
          </button>
        </div>

        {/* Основное содержимое модалки */}
        <div className="modal-content">
          {children}
        </div>
      </div>
    </div>
  );
}

export default Modal;
```

Стили для модального окна (Modal.css):

```css
.modal-background {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 1000;
}

.modal-window {
  background: white;
```

```
  border-radius: 8px;
  padding: 0;
  width: 90%;
  max-width: 500px;
  max-height: 80vh;
  overflow-y: auto;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

.modal-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 16px 20px;
  border-bottom: 1px solid #e0e0e0;
}

.modal-header h2 {
  margin: 0;
  font-size: 1.3rem;
  color: #333;
}

.close-button {
  background: none;
  border: none;
  font-size: 24px;
  cursor: pointer;
  color: #666;
  padding: 5px 10px;
}

.close-button:hover {
  color: #333;
}

.modal-content {
  padding: 20px;
}
```

Пример использования модального окна:

```
// SimpleModalExample.jsx
import { useState } from 'react';
import Modal from './Modal';

function SimpleModalExample() {
  const [isModalOpen, setIsModalOpen] = useState(false);

  return (
    <div>
      <button onClick={() => setIsModalOpen(true)}>
```

```
        Открыть модальное окно
      </button>

      <Modal
        isOpen={isModalOpen}
        onClose={() => setIsModalOpen(false)}
        title="Пример модального окна"
      >
        <div>
          <p>Это содержимое модального окна.</p>
          <p>Здесь может быть любой React-компонент.</p>
          <button onClick={() => setIsModalOpen(false)}>
            Закрыть
          </button>
        </div>
      </Modal>
    </div>
  );
}

export default SimpleModalExample;
```

Пример 3. Компонент прогресс-бара с разными вариантами использования

Проблема: Нужно создать универсальный компонент прогресс-бара, который можно использовать для отображения прогресса разного типа с разными стилями.

Подход к решению: Создаем компонент ProgressBar, который принимает различные props для кастомизации внешнего вида и поведения.

Исходный код в файле `ProgressBar.jsx`:

```
// Универсальный компонент прогресс-бара
function ProgressBar({
  progress,           // Текущее значение прогресса (от 0 до 100)
  label = '',         // Подпись к прогресс-бару
  color = '#4CAF50',  // Цвет заполнения
  height = 20,        // Высота прогресс-бара
  showPercentage = true, // Показывать ли процент
  animated = false    // Анимировать ли заполнение
}) {
  // Обеспечиваем, чтобы прогресс был в пределах 0-100
  const normalizedProgress = Math.min(100, Math.max(0, progress));

  return (
    <div className="progress-bar-container">
      {/* Заголовок с лейблом и процентом */}
      {(label || showPercentage) && (
        <div className="progress-bar-header">
          {label && <span className="progress-label">{label}</span>}
          {showPercentage && (
```

```
            <span className="progress-percentage">{normalizedProgress}%</span>
          )}
        </div>
      )}

      {/* Внешняя оболочка прогресс-бара */}
      <div
        className="progress-bar-outer"
        style={{
          height: `${height}px`,
          backgroundColor: '#f0f0f0',
          borderRadius: '10px',
          overflow: 'hidden'
        }}
      >
        {/* Заполняемая часть прогресс-бара */}
        <div
          className={`progress-bar-inner ${animated ? 'animated' : ''}`}
          style={{
            width: `${normalizedProgress}%`,
            backgroundColor: color,
            height: '100%',
            transition: animated ? 'width 0.5s ease-in-out' : 'none',
            borderRadius: '10px'
          }}
        />
      </div>
    </div>
  );
}

export default ProgressBar;
```

Примеры использования прогресс-бара в разных сценариях:

```
// ProgressDashboard.jsx - демонстрация разных вариантов прогресс-бара
import ProgressBar from './ProgressBar';
import './ProgressDashboard.css'; // Добавим файл со стилями

function ProgressDashboard() {
  const overallProgress = 65;
  const frontendProgress = 80;
  const backendProgress = 50;
  const databaseProgress = 30;

  // Прогресс по неделям
  const weeklyProgress = [90, 75, 60, 45, 30, 15];

  return (
    <div className="progress-dashboard">
      <h2>Мой прогресс в изучении</h2>
```

```
        {/* Основной прогресс-бар */}
        <ProgressBar
          progress={overallProgress}
          label="Общий прогресс"
          color="#2196F3"
          height={25}
        />

        {/* Прогресс-бар для фронтенда */}
        <ProgressBar
          progress={frontendProgress}
          label="Фронтенд разработка"
          color="#4CAF50"
          showPercentage={true}
        />

        {/* Прогресс-бар для бэкенда */}
        <ProgressBar
          progress={backendProgress}
          label="Бэкенд разработка"
          color="#FF9800"
          showPercentage={true}
        />

        {/* Прогресс-бар для баз данных */}
        <ProgressBar
          progress={databaseProgress}
          label="Базы данных"
          color="#F44336"
          height={15}
          showPercentage={false}
        />

        {/* Прогресс по неделям - исправленная версия */}
        <div className="weekly-progress">
          <h3>Прогресс по неделям:</h3>
          <div className="weekly-bars">
            {weeklyProgress.map((progress, index) => (
              <div key={index} className="week-item">
                <span className="week-label">Неделя {index + 1}</span>
                <ProgressBar
                  progress={progress}
                  height={12}
                  showPercentage={false}
                  color="#9C27B0"
                />
                <span className="week-percentage">{progress}%</span>
              </div>
            ))}
          </div>
        </div>
      </div>
    );
  }
```

```
export default ProgressDashboard;
```

Файл `ProgressDashboard.css`:

```
.progress-dashboard {
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
}

.progress-dashboard h2 {
  text-align: center;
  color: #333;
  margin-bottom: 30px;
}

.weekly-progress {
  margin-top: 30px;
  padding: 20px;
  background-color: #f8f9fa;
  border-radius: 8px;
}

.weekly-progress h3 {
  margin-top: 0;
  color: #555;
}

.weekly-bars {
  display: flex;
  flex-direction: column;
  gap: 15px;
}

.week-item {
  display: flex;
  align-items: center;
  gap: 15px;
}

.week-label {
  min-width: 80px;
  font-size: 14px;
  color: #666;
}

.week-percentage {
  min-width: 40px;
  font-size: 14px;
  color: #333;
```

```
  font-weight: bold;
}
```

Практическая часть

Интеграция переиспользуемых компонентов в трекер технологий

Шаг 1: Создайте кастомный хук для работы с технологиями

Создайте файл `useTechnologies.js`:

```
import useLocalStorage from './useLocalStorage';

// Начальные данные для технологий
const initialTechnologies = [
  {
    id: 1,
    title: 'React Components',
    description: 'Изучение базовых компонентов',
    status: 'not-started',
    notes: '',
    category: 'frontend'
  },
  {
    id: 2,
    title: 'Node.js Basics',
    description: 'Основы серверного JavaScript',
    status: 'not-started',
    notes: '',
    category: 'backend'
  },
  // ... добавьте еще технологии
];

function useTechnologies() {
  const [technologies, setTechnologies] = useLocalStorage('technologies',
initialTechnologies);

  // Функция для обновления статуса технологии
  const updateStatus = (techId, newStatus) => {
    setTechnologies(prev =>
      prev.map(tech =>
        tech.id === techId ? { ...tech, status: newStatus } : tech
      )
    );
  };

  // Функция для обновления заметок
  const updateNotes = (techId, newNotes) => {
    setTechnologies(prev =>
      prev.map(tech =>
```

```
          tech.id === techId ? { ...tech, notes: newNotes } : tech
        )
      );
    };

    // Функция для расчета общего прогресса
    const calculateProgress = () => {
      if (technologies.length === 0) return 0;
      const completed = technologies.filter(tech => tech.status ===
'completed').length;
      return Math.round((completed / technologies.length) * 100);
    };

    return {
      technologies,
      updateStatus,
      updateNotes,
      progress: calculateProgress()
    };
}

export default useTechnologies;
```

Шаг 2: Обновите главный компонент App.js

```
import useTechnologies from './useTechnologies';
import ProgressBar from './ProgressBar';
import TechnologyCard from './TechnologyCard';
import TechnologyModal from './TechnologyModal';

function App() {
  const { technologies, updateStatus, updateNotes, progress } = useTechnologies();

  return (
    <div className="app">
      <header className="app-header">
        <h1>Трекер изучения технологий</h1>
        <ProgressBar
          progress={progress}
          label="Общий прогресс"
          color="#4CAF50"
          animated={true}
          height={20}
        />
      </header>

      <main className="app-main">
        <div className="technologies-grid">
          {technologies.map(tech => (
            <TechnologyCard
              key={tech.id}
              technology={tech}
```

```
              onStatusChange={updateStatus}
              onNotesChange={updateNotes}
            />
          ))}
        </div>
      </main>
    </div>
  );
}
```

Самостоятельная работа

Задание: Создайте компонент "Быстрые действия" (QuickActions)

1. Создайте компонент, который показывает кнопки для массовых действий
2. Добавьте кнопки:
   - "Отметить все как выполненные"
   - "Сбросить все статусы"
   - "Экспорт данных"
3. Используйте переиспользуемые компоненты там, где это возможно

Пример реализации:

```
// QuickActions.jsx
import { useState } from 'react';
import Modal from './Modal';

function QuickActions({ onMarkAllCompleted, onResetAll, technologies }) {
  const [showExportModal, setShowExportModal] = useState(false);

  const handleExport = () => {
    const data = {
      exportedAt: new Date().toISOString(),
      technologies: technologies
    };
    const dataStr = JSON.stringify(data, null, 2);
    // Здесь можно добавить логику для скачивания файла
    console.log('Данные для экспорта:', dataStr);
    setShowExportModal(true);
  };

  return (
    <div className="quick-actions">
      <h3>Быстрые действия</h3>
      <div className="action-buttons">
        <button onClick={onMarkAllCompleted} className="btn btn-success">
          ☑ Отметить все как выполненные
        </button>
        <button onClick={onResetAll} className="btn btn-warning">
          🔄 Сбросить все статусы
        </button>
```

```
          <button onClick={handleExport} className="btn btn-info">
            📥 Экспорт данных
          </button>
        </div>

        <Modal
          isOpen={showExportModal}
          onClose={() => setShowExportModal(false)}
          title="Экспорт данных"
        >
          <p>Данные успешно подготовлены для экспорта!</p>
          <p>Проверьте консоль разработчика для просмотра данных.</p>
          <button onClick={() => setShowExportModal(false)}>
            Закрыть
          </button>
        </Modal>
      </div>
    );
}

export default QuickActions;
```

Что проверить перед завершением:

- Кастомный хук useTechnologies корректно сохраняет данные в localStorage
- Прогресс-бар отображает актуальный прогресс
- Модальные окна открываются и закрываются корректно
- Быстрые действия работают и влияют на все технологии
- Приложение остается отзывчивым и без ошибок