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-1-21

Практическое занятие 21: Менеджер эффектов и контролируемые поля

В рамках данного занятия будут рассмотрены возможности useEffect и некоторые элементы работы с LocalStorage (в заданиях). Подробную информацию о них можно найти в материалах лекций, либо ознакомиться со статьями:

useEffect: https://habr.com/ru/companies/rshb/articles/687364/

LocalStorage: https://blog.logrocket.com/using-localstorage-react-hooks/

Теоретическая часть

Пример 1. Базовое использование useEffect

Исходный код в файле `WindowSizeTracker.jsx`:

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

function WindowSizeTracker() {
  // Состояние для хранения размеров окна
  const [windowSize, setWindowSize] = useState({
    width: window.innerWidth,
    height: window.innerHeight
  });

  // useEffect для отслеживания изменения размера окна
  useEffect(() => {
    // Функция-обработчик изменения размера
    const handleResize = () => {
      setWindowSize({
        width: window.innerWidth,
        height: window.innerHeight
      });
```

```
    };

    // Добавляем слушатель события resize
    window.addEventListener('resize', handleResize);

    // Функция очистки - удаляем слушатель при размонтировании компонента
    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []); // Пустой массив зависимостей = эффект только при монтировании

  // Определяем тип экрана based on ширины
  const getScreenType = () => {
    if (windowSize.width < 768) return 'мобильный';
    if (windowSize.width < 1024) return 'планшет';
    return 'десктоп';
  };

  return (
    <div className="window-tracker">
      <h2>Отслеживание размера окна</h2>
      <div className="size-info">
        <p>Ширина: <strong>{windowSize.width}px</strong></p>
        <p>Высота: <strong>{windowSize.height}px</strong></p>
        <p>Тип экрана: <strong>{getScreenType()}</strong></p>
      </div>
    </div>
  );
}

export default WindowSizeTracker;
```

Подключаем компонент, добавив его в App.

Пример 2. Работа с API и состоянием загрузки

Исходный код в файле `UserProfile.jsx`:

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

function UserProfile() {
  // Три состояния: данные, загрузка, ошибка
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // useEffect для загрузки данных при монтировании компонента
  useEffect(() => {
    // Асинхронная функция для запроса данных
    const fetchUser = async () => {
      try {
```

```
        // Сбрасываем состояния перед началом загрузки
        setLoading(true);
        setError(null);

        // Имитация запроса к API (можно заменить на реальный endpoint)
        const response = await
fetch('https://jsonplaceholder.typicode.com/users/1');

        // Проверяем успешность ответа
        if (!response.ok) {
          throw new Error('Не удалось загрузить данные пользователя');
        }

        // Парсим JSON и обновляем состояние
        const userData = await response.json();
        setUser(userData);
      } catch (err) {
        // Обрабатываем ошибки
        setError(err.message);
      } finally {
        // Выключаем индикатор загрузки в любом случае
        setLoading(false);
      }
    };

    // Вызываем функцию загрузки
    fetchUser();
  }, []); // Пустой массив = загрузка только при первом рендере

  // Показываем индикатор загрузки
  if (loading) {
    return (
      <div className="user-profile loading">
        <p>Загрузка профиля...</p>
      </div>
    );
  }

  // Показываем сообщение об ошибке
  if (error) {
    return (
      <div className="user-profile error">
        <p>Ошибка: {error}</p>
        <button onClick={() => window.location.reload()}>
          Попробовать снова
        </button>
      </div>
    );
  }

  // Рендерим данные пользователя
  return (
    <div className="user-profile">
      <h2>Профиль пользователя</h2>
```

```
        <div className="user-info">
          <p><strong>Имя:</strong> {user.name}</p>
          <p><strong>Email:</strong> {user.email}</p>
          <p><strong>Телефон:</strong> {user.phone}</p>
          <p><strong>Website:</strong> {user.website}</p>
        </div>
      </div>
    );
  }

  export default UserProfile;
```

Подключаем компонент, добавив его в App.

Пример 3. Контролируемые поля с валидацией

Исходный код в файле `ContactForm.jsx`:

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

  function ContactForm() {
    // Состояние для данных формы
    const [formData, setFormData] = useState({
      name: '',
      email: '',
      phone: '',
      message: ''
    });

    // Состояние для ошибок валидации
    const [errors, setErrors] = useState({});
    // Состояние для общей валидности формы
    const [isFormValid, setIsFormValid] = useState(false);

    // useEffect для валидации при каждом изменении formData
    useEffect(() => {
      const validateForm = () => {
        const newErrors = {};

        // Валидация имени - обязательно, минимум 2 символа
        if (!formData.name.trim()) {
          newErrors.name = 'Имя обязательно для заполнения';
        } else if (formData.name.trim().length < 2) {
          newErrors.name = 'Имя должно содержать минимум 2 символа';
        }

        // Валидация email - проверка формата
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!formData.email) {
          newErrors.email = 'Email обязателен для заполнения';
        } else if (!emailRegex.test(formData.email)) {
```

```
        newErrors.email = 'Введите корректный email адрес';
      }

      // Валидация сообщения - обязательно, минимум 10 символов
      if (!formData.message.trim()) {
        newErrors.message = 'Сообщение обязательно для заполнения';
      } else if (formData.message.trim().length < 10) {
        newErrors.message = 'Сообщение должно содержать минимум 10 символов';
      }

      // Обновляем состояния ошибок и валидности
      setErrors(newErrors);
      setIsFormValid(Object.keys(newErrors).length === 0);
    };

    validateForm();
  }, [formData]); // Зависимость от formData - валидация при каждом изменении

  // Обработчик изменения полей ввода
  const handleChange = (e) => {
    const { name, value } = e.target;
    // Обновляем formData, сохраняя предыдущие значения
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  // Обработчик отправки формы
  const handleSubmit = (e) => {
    e.preventDefault();
    if (isFormValid) {
      // В реальном приложении здесь был бы API запрос
      console.log('Данные для отправки:', formData);
      alert('Форма успешно отправлена!');

      // Сброс формы после отправки
      setFormData({
        name: '',
        email: '',
        phone: '',
        message: ''
      });
    }
  };

  return (
    <form onSubmit={handleSubmit} className="contact-form">
      <h2>Форма обратной связи</h2>

      {/* Поле имени */}
      <div className="form-group">
        <label htmlFor="name">Имя *</label>
        <input
```

```
            id="name"
            type="text"
            name="name"
            value={formData.name}
            onChange={handleChange}
            className={errors.name ? 'error' : ''}
            placeholder="Введите ваше имя"
          />
          {errors.name && <span className="error-message">{errors.name}</span>}
        </div>

        {/* Поле email */}
        <div className="form-group">
          <label htmlFor="email">Email *</label>
          <input
            id="email"
            type="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
            className={errors.email ? 'error' : ''}
            placeholder="example@mail.com"
          />
          {errors.email && <span className="error-message">{errors.email}</span>}
        </div>

        {/* Поле телефона (необязательное) */}
        <div className="form-group">
          <label htmlFor="phone">Телефон</label>
          <input
            id="phone"
            type="tel"
            name="phone"
            value={formData.phone}
            onChange={handleChange}
            placeholder="+7 (999) 999-99-99"
          />
        </div>

        {/* Поле сообщения */}
        <div className="form-group">
          <label htmlFor="message">Сообщение *</label>
          <textarea
            id="message"
            name="message"
            value={formData.message}
            onChange={handleChange}
            rows="4"
            className={errors.message ? 'error' : ''}
            placeholder="Введите ваше сообщение..."
          />
          {errors.message && <span className="error-message">{errors.message}
</span>}
        </div>
```

```
      {/* Кнопка отправки */}
      <button
        type="submit"
        disabled={!isFormValid}
        className={!isFormValid ? 'disabled' : ''}
      >
        Отправить сообщение
      </button>
    </form>
  );
}

export default ContactForm;
```

Подключаем компонент, добавив его в App.

Практическая часть

Добавление заметок к технологиям с сохранением в localStorage

Работа с LocalStorage производится встроенными средствами React и его дополнениями. Материалы по использованию есть в лекций и статье по ссылке.

Шаг 1: Обновление состояния технологий Добавьте поле для заметок в каждую технологию в компоненте App:

```
const [technologies, setTechnologies] = useState([
  {
    id: 1,
    title: 'React Components',
    description: 'Изучение базовых компонентов',
    status: 'not-started',
    notes: '' // Новое поле для заметок пользователя
  },
  // ... остальные технологии
]);
```

Шаг 2: Автосохранение в localStorage Добавьте простой эффект для сохранения данных:

```
// Сохраняем технологии в localStorage при любом изменении
useEffect(() => {
  localStorage.setItem('techTrackerData', JSON.stringify(technologies));
  console.log('Данные сохранены в localStorage');
}, [technologies]);
```

Шаг 3: Загрузка из localStorage при запуске Добавьте эффект для загрузки сохраненных данных:

```
// Загружаем данные из localStorage при первом рендере
useEffect(() => {
  const saved = localStorage.getItem('techTrackerData');
  if (saved) {
    setTechnologies(JSON.parse(saved));
    console.log('Данные загружены из localStorage');
  }
}, []);
```

Шаг 4: Компонент для заметок Создайте простой компонент для редактирования заметок:

```
// TechnologyNotes.jsx
function TechnologyNotes({ notes, onNotesChange, techId }) {
  return (
    <div className="notes-section">
      <h4>Мои заметки:</h4>
      <textarea
        value={notes}
        onChange={(e) => onNotesChange(techId, e.target.value)}
        placeholder="Записывайте сюда важные моменты..."
        rows="3"
      />
      <div className="notes-hint">
        {notes.length > 0 ? `Заметка сохранена (${notes.length} символов)` :
'Добавьте заметку'}
      </div>
    </div>
  );
}
```

Шаг 5: Функция обновления заметок Добавьте в App.js функцию для изменения заметок:

```
const updateTechnologyNotes = (techId, newNotes) => {
  setTechnologies(prevTech =>
    prevTech.map(tech =>
      tech.id === techId ? { ...tech, notes: newNotes } : tech
    )
  );
};
```

Самостоятельная работа

Простое задание: Добавьте поле поиска по технологиям

1. Создайте состояние для поискового запроса
2. Добавьте input для ввода текста
3. Отфильтруйте технологии по названию и описанию

4. Показывайте количество найденных результатов

Пример реализации:

```
// В компоненте App
const [searchQuery, setSearchQuery] = useState('');

// Фильтрация технологий
const filteredTechnologies = technologies.filter(tech =>
  tech.title.toLowerCase().includes(searchQuery.toLowerCase()) ||
  tech.description.toLowerCase().includes(searchQuery.toLowerCase())
);

// В JSX добавьте:
<div className="search-box">
  <input
    type="text"
    placeholder="Поиск технологий..."
    value={searchQuery}
    onChange={(e) => setSearchQuery(e.target.value)}
  />
  <span>Найдено: {filteredTechnologies.length}</span>
</div>
```

Что проверить:

- Заметки сохраняются после перезагрузки страницы
- Поиск находит технологии по названию и описанию
- Статусы технологий сохраняются правильно
- Приложение не показывает ошибок в консоли