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-1-24

Практическое занятие 24: Извлечение данных из API и их обработка

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

https://purpleschool.ru/knowledge-base/article/react-js-api

https://habr.com/ru/articles/706802/

Теоретическая часть

Пример 1. Базовый запрос к API с использованием fetch

Проблема: Нужно получить данные с внешнего API и отобразить их в компоненте, обрабатывая состояния загрузки и ошибок.

Подход к решению: Используем fetch для выполнения HTTP-запроса, useState для хранения данных и useEffect для выполнения запроса при монтировании компонента.

Исходный код в файле `UserList.jsx`:

```
import { useState, useEffect } from 'react';
import 'UserList.css'

function UserList() {
  // Состояния для данных, загрузки и ошибок
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // Функция для загрузки пользователей
  const fetchUsers = async () => {
    try {
```

```
      setLoading(true);
      setError(null);

      // Выполняем GET-запрос к API
      const response = await fetch('https://jsonplaceholder.typicode.com/users');

      // Проверяем успешность ответа
      if (!response.ok) {
        throw new Error(`Ошибка HTTP: ${response.status}`);
      }

      // Парсим JSON-ответ
      const userData = await response.json();
      setUsers(userData);

    } catch (err) {
      // Обрабатываем ошибки
      setError(err.message);
      console.error('Ошибка при загрузке пользователей:', err);
    } finally {
      // Выключаем индикатор загрузки в любом случае
      setLoading(false);
    }
  };

  // Выполняем запрос при монтировании компонента
  useEffect(() => {
    fetchUsers();
  }, []);

  // Функция для повторной загрузки
  const handleRetry = () => {
    fetchUsers();
  };

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

  // Показываем сообщение об ошибке
  if (error) {
    return (
      <div className="user-list error">
        <h2>Произошла ошибка</h2>
        <p>{error}</p>
        <button onClick={handleRetry} className="retry-button">
          Попробовать снова
        </button>
```

```
      </div>
    );
  }

  // Отображаем список пользователей
  return (
    <div className="user-list">
      <h2>Список пользователей ({users.length})</h2>

      <div className="users-grid">
        {users.map(user => (
          <div key={user.id} className="user-card">
            <h3>{user.name}</h3>
            <p><strong>Email:</strong> {user.email}</p>
            <p><strong>Телефон:</strong> {user.phone}</p>
            <p><strong>Город:</strong> {user.address.city}</p>
            <p><strong>Компания:</strong> {user.company.name}</p>
          </div>
        ))}
      </div>
    </div>
  );
}

export default UserList;
```

Стили для компонента в файле `UserList.css`:

```
.user-list {
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

.loading {
  text-align: center;
  padding: 40px;
}

.spinner {
  border: 4px solid #f3f3f3;
  border-top: 4px solid #3498db;
  border-radius: 50%;
  width: 40px;
  height: 40px;
  animation: spin 2s linear infinite;
  margin: 0 auto 20px;
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
```

```
}

.error {
  text-align: center;
  padding: 40px;
  color: #e74c3c;
}

.retry-button {
  background-color: #3498db;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 5px;
  cursor: pointer;
  margin-top: 10px;
}

.retry-button:hover {
  background-color: #2980b9;
}

.users-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  gap: 20px;
  margin-top: 20px;
}

.user-card {
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 20px;
  background-color: white;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.user-card h3 {
  margin-top: 0;
  color: #2c3e50;
}

.user-card p {
  margin: 8px 0;
  font-size: 14px;
}
```

Добавим компонент в App.js и посмотрим его работу.

Пример 2. Поиск с debounce и обработка отмены запросов

Проблема: Нужно реализовать поиск по API с задержкой (debounce) и обработать отмену предыдущих запросов при новом поиске.

Подход к решению: Используем setTimeout для debounce и AbortController для отмены предыдущих запросов.

Исходный код в файле `ProductSearch.jsx`:

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

function ProductSearch() {
  const [products, setProducts] = useState([]);
  const [searchTerm, setSearchTerm] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  // Используем useRef для хранения таймера и AbortController
  const searchTimeoutRef = useRef(null);
  const abortControllerRef = useRef(null);

  // Функция для поиска продуктов
  const searchProducts = async (query) => {
    // Отменяем предыдущий запрос, если он существует
    if (abortControllerRef.current) {
      abortControllerRef.current.abort();
    }

    // Создаем новый AbortController для текущего запроса
    abortControllerRef.current = new AbortController();

    try {
      setLoading(true);
      setError(null);

      // Если поисковый запрос пустой, очищаем результаты
      if (!query.trim()) {
        setProducts([]);
        setLoading(false);
        return;
      }

      const response = await fetch(
        `https://dummyjson.com/products/search?q=${encodeURIComponent(query)}`,
        { signal: abortControllerRef.current.signal }
      );

      if (!response.ok) {
        throw new Error(`Ошибка HTTP: ${response.status}`);
      }

      const data = await response.json();
      setProducts(data.products || []);
```

```
    } catch (err) {
      // Игнорируем ошибки отмены запроса
      if (err.name !== 'AbortError') {
        setError(err.message);
        console.error('Ошибка при поиске продуктов:', err);
      }
    } finally {
      setLoading(false);
    }
  };

  // Обработчик изменения поискового запроса
  const handleSearchChange = (e) => {
    const value = e.target.value;
    setSearchTerm(value);

    // Очищаем предыдущий таймер
    if (searchTimeoutRef.current) {
      clearTimeout(searchTimeoutRef.current);
    }

    // Устанавливаем новый таймер для debounce (500ms)
    searchTimeoutRef.current = setTimeout(() => {
      searchProducts(value);
    }, 500);
  };

  // Очистка при размонтировании компонента
  useEffect(() => {
    return () => {
      if (searchTimeoutRef.current) {
        clearTimeout(searchTimeoutRef.current);
      }
      if (abortControllerRef.current) {
        abortControllerRef.current.abort();
      }
    };
  }, []);

  return (
    <div className="product-search">
      <h2>Поиск продуктов</h2>

      <div className="search-box">
        <input
          type="text"
          placeholder="Введите название продукта..."
          value={searchTerm}
          onChange={handleSearchChange}
          className="search-input"
        />
        {loading && <span className="search-loading">⏳</span>}
      </div>
```

```
        {error && (
          <div className="error-message">
            Ошибка: {error}
          </div>
        )}

        <div className="search-results">
          {products.length > 0 ? (
            <>
              <h3>Найдено продуктов: {products.length}</h3>
              <div className="products-grid">
                {products.map(product => (
                  <div key={product.id} className="product-card">
                    <img
                      src={product.thumbnail}
                      alt={product.title}
                      className="product-image"
                    />
                    <div className="product-info">
                      <h4>{product.title}</h4>
                      <p className="product-price">${product.price}</p>
                      <p className="product-category">{product.category}</p>
                      <p className="product-description">{product.description}</p>
                    </div>
                  </div>
                ))}
              </div>
            </>
          ) : (
            searchTerm.trim() && !loading && (
              <p className="no-results">Продукты не найдены</p>
            )
          )}
        </div>
      </div>
    );
}

export default ProductSearch;
```

Добавим в App.js и просмотрим работу. Товары из API примера на странице `https://dummyjson.com/products`, например: `Essence Mascara Lash Princess`

Пример 3. Кастомный хук для работы с API

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

Подход к решению: Создаем кастомный хук useApi, который инкапсулирует логику запросов, состояний загрузки и ошибок.

Исходный код в файле `useApi.jsx`:

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

// Кастомный хук для работы с API
function useApi(url, options = {}) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // Функция для выполнения запроса
  const fetchData = useCallback(async (abortController) => {
    try {
      setLoading(true);
      setError(null);

      const response = await fetch(url, {
        ...options,
        signal: abortController?.signal
      });

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      const result = await response.json();
      setData(result);

    } catch (err) {
      // Игнорируем ошибки отмены запроса
      if (err.name !== 'AbortError') {
        setError(err.message);
      }
    } finally {
      setLoading(false);
    }
  }, [url]); // Только url как зависимость

  // Выполняем запрос при изменении URL
  useEffect(() => {
    const abortController = new AbortController();

    // Выполняем запрос только если URL существует
    if (url) {
      fetchData(abortController);
    }

    // Функция очистки - отменяем запрос при размонтировании
    return () => {
      abortController.abort();
    };
  }, [url, fetchData]); // fetchData стабильна благодаря useCallback
```

```
  // Функция для повторного выполнения запроса
  const refetch = useCallback(() => {
    const abortController = new AbortController();
    fetchData(abortController);
    return () => abortController.abort();
  }, [fetchData]);

  return { data, loading, error, refetch };
}

export default useApi;
```

Использование кастомного хука в компоненте `PostList.jsx`:

```
// PostList.js - компонент для отображения списка постов
import useApi from './useApi';

function PostList() {
  // Используем наш кастомный хук
  const { data: posts, loading, error, refetch } = useApi(
    'https://jsonplaceholder.typicode.com/posts'
  );

  if (loading) {
    return (
      <div className="post-list loading">
        <p>Загрузка постов...</p>
      </div>
    );
  }

  if (error) {
    return (
      <div className="post-list error">
        <h2>Ошибка при загрузке постов</h2>
        <p>{error}</p>
        <button onClick={refetch}>Попробовать снова</button>
      </div>
    );
  }

  return (
    <div className="post-list">
      <div className="post-list-header">
        <h2>Список постов ({posts?.length || 0})</h2>
        <button onClick={refetch} className="refresh-button">
          Обновить
        </button>
      </div>

      <div className="posts-container">
        {posts?.map(post => (
```

```
            <article key={post.id} className="post-card">
              <h3>{post.title}</h3>
              <p>{post.body}</p>
              <div className="post-meta">
                <span>ID: {post.id}</span>
                <span>User: {post.userId}</span>
              </div>
            </article>
          ))}
        </div>
      </div>
    );
}

export default PostList;
```

Практическая часть

Интеграция API в трекер технологий

ВАЖНО! Шаг 0: Найдите для себя API для своего стека технологий и/или для конкретного набора для загрузки карточек и/или описаний технологий на внешнем ресурсе, так как `roadmap.sh` не содержит публичных API

В качестве примеров: https://github.com/public-api-lists/public-api-lists

Если найти удачный для практической работы пример не удаётся, то можно применить API к другой проблеме в вашем проекте (творческое задание), а загрузка-выгрузка карточек технологий будет происходить исключительно механизмами экспорта-импорта.

Шаг 1: Создайте кастомный хук для работы с технологиями

```
// hooks/useTechnologiesApi.js
import { useState, useEffect } from 'react';

function useTechnologiesApi() {
  const [technologies, setTechnologies] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // Загрузка технологий из API
  const fetchTechnologies = async () => {
    try {
      setLoading(true);
      setError(null);

      // В реальном приложении здесь будет запрос к вашему API
      // Сейчас имитируем загрузку с задержкой
      await new Promise(resolve => setTimeout(resolve, 1000));

      // Mock данные - в реальном приложении замените на реальный API
```

```
      const mockTechnologies = [
        {
          id: 1,
          title: 'React',
          description: 'Библиотека для создания пользовательских интерфейсов',
          category: 'frontend',
          difficulty: 'beginner',
          resources: ['https://react.dev', 'https://ru.reactjs.org']
        },
        {
          id: 2,
          title: 'Node.js',
          description: 'Среда выполнения JavaScript на сервере',
          category: 'backend',
          difficulty: 'intermediate',
          resources: ['https://nodejs.org', 'https://nodejs.org/ru/docs/']
        },
        {
          id: 3,
          title: 'TypeScript',
          description: 'Типизированное надмножество JavaScript',
          category: 'language',
          difficulty: 'intermediate',
          resources: ['https://www.typescriptlang.org']
        }
      ];

      setTechnologies(mockTechnologies);

    } catch (err) {
      setError('Не удалось загрузить технологии');
      console.error('Ошибка загрузки:', err);
    } finally {
      setLoading(false);
    }
  };

  // Добавление новой технологии
  const addTechnology = async (techData) => {
    try {
      // Имитация API запроса
      await new Promise(resolve => setTimeout(resolve, 500));

      const newTech = {
        id: Date.now(), // В реальном приложении ID генерируется на сервере
        ...techData,
        createdAt: new Date().toISOString()
      };

      setTechnologies(prev => [...prev, newTech]);
      return newTech;

    } catch (err) {
      throw new Error('Не удалось добавить технологию');
```

```
    }
  };

  // Загружаем технологии при монтировании
  useEffect(() => {
    fetchTechnologies();
  }, []);

  return {
    technologies,
    loading,
    error,
    refetch: fetchTechnologies,
    addTechnology
  };
}

export default useTechnologiesApi;
```

Шаг 2: Создайте компонент для загрузки дорожных карт из API

```
// components/RoadmapImporter.js
import { useState } from 'react';
import useTechnologiesApi from '../hooks/useTechnologiesApi';

function RoadmapImporter() {
  const { technologies, loading, error, addTechnology } = useTechnologiesApi();
  const [importing, setImporting] = useState(false);

  const handleImportRoadmap = async (roadmapUrl) => {
    try {
      setImporting(true);

      // Имитация загрузки дорожной карты из API
      const response = await fetch(roadmapUrl);
      if (!response.ok) throw new Error('Не удалось загрузить дорожную карту');

      const roadmapData = await response.json();

      // Добавляем каждую технологию из дорожной карты
      for (const tech of roadmapData.technologies) {
        await addTechnology(tech);
      }

      alert(`Успешно импортировано ${roadmapData.technologies.length}
технологий`);

    } catch (err) {
      alert(`Ошибка импорта: ${err.message}`);
    } finally {
      setImporting(false);
    }
```

```
  };

  const handleExampleImport = () => {
    // Пример импорта из фиктивного API
    handleImportRoadmap('https://api.example.com/roadmaps/frontend');
  };

  return (
    <div className="roadmap-importer">
      <h3>Импорт дорожной карты</h3>

      <div className="import-actions">
        <button
          onClick={handleExampleImport}
          disabled={importing}
          className="import-button"
        >
          {importing ? 'Импорт...' : 'Импорт пример дорожной карты'}
        </button>
      </div>

      {error && (
        <div className="error-message">
          {error}
        </div>
      )}
    </div>
  );
}

export default RoadmapImporter;
```

Шаг 3: Обновите главный компонент для использования API

```
// App.js
import useTechnologiesApi from './hooks/useTechnologiesApi';
import RoadmapImporter from './components/RoadmapImporter';
import TechnologyList from './components/TechnologyList';

function App() {
  const { technologies, loading, error, refetch } = useTechnologiesApi();

  if (loading) {
    return (
      <div className="app-loading">
        <div className="spinner"></div>
        <p>Загрузка технологий...</p>
      </div>
    );
  }

  return (
```

```
      <div className="app">
        <header className="app-header">
          <h1>🚀 Трекер изучения технологий</h1>
          <button onClick={refetch} className="refresh-btn">
            Обновить
          </button>
        </header>

        {error && (
          <div className="app-error">
            <p>{error}</p>
            <button onClick={refetch}>Попробовать снова</button>
          </div>
        )}

        <main className="app-main">
          <RoadmapImporter />
          <TechnologyList technologies={technologies} />
        </main>
      </div>
    );
  }

  export default App;
```

Самостоятельная работа

Задание 1: Создайте компонент для поиска технологий с использованием debounce

Задание 2: Добавьте возможность загрузки дополнительных ресурсов для каждой технологии из API

Что проверить перед завершением:

- Данные загружаются из API при запуске приложения
- Состояния загрузки и ошибок обрабатываются корректно
- Поиск работает с задержкой и отменой предыдущих запросов
- Импорт дорожных карт добавляет новые технологии