UNIVERSITATEA DE STAT DIN MOLDOVA
Specialitatea: «Informatica aplicată»
Disciplina: «React»

Лабораторная работа №1

A verificat:
Denis Negura

A elaborat:
Ianciuc Nadejda

Chișinău, 2025

План

Цели работы

1. Научиться создавать базовые компоненты,
2. Использовать хук состояния useState,
3. Использовать хук эффектов useEffect,
4. Написать простой CRUD (create / read / update / edit) над сущностью данных.

Используемое API

PokéAPI[1] - по тематике “Покемоны”.

API выбрано благодаря:

- простоте работы,
- наличию всех необходимых данных,
- отсутствию авторизации,
- хорошей структуре ответов.

[1] URL: https://pokeapi.co/api/v2/pokemon

Реализация требований

1. Загрузка данных и отображение списка сущностей

Загрузка выполняется в PokemonContext.jsx внутри useEffect, чтобы загрузить данные один раз при монтировании приложения. Используется fetch() и Promise.all() для получения детальной информации по каждому покемону.

Функционал включает:

- глобальное состояние списка покемонов;
- индикатор загрузки;
- генерацию уникального ключа для каждого элемента;
- обработку ошибок.

2. Пагинация

В компоненте HomePage.jsx реализована пагинация:

- 20 элементов на страницу,
- вычисление диапазона для текущей страницы через useMemo.

Пагинация оптимизирована, пересчитывается только при изменении зависимостей.

Файл: src/pages/HomePage.jsx

```
const ITEMS_PER_PAGE = 20;
const [currentPage, setCurrentPage] = useState(1);

const paginatedPokemons = useMemo(() => {
    const startIndex = (currentPage - 1) * ITEMS_PER_PAGE;
    const endIndex = startIndex + ITEMS_PER_PAGE;
    return filteredPokemons.slice(startIndex, endIndex);
}, [filteredPokemons, currentPage, ITEMS_PER_PAGE]);
```

3. Просмотр сущности (READ)

Просмотр покемона реализован на странице PokemonDetailPage.jsx:

- получение параметра id через useParams;
- поиск покемона в глобальном состоянии через getById;
- отображение изображения, имени, роста, веса, типов и характеристик.

Модальное окно просмотра реализовано в компоненте PokemonModal.jsx

- отображает данные поверх контента,

- содержит кнопки “Редактировать” и “Удалить”,
- закрывается по клику на кнопку.

Файл: src/components/PokemonModal.jsx

```
const PokemonModal = ({ pokemon, isOpen, onClose, onUpdate, onDelete }) => {
    if (!isOpen) return null;

    return (
        <div className="fixed inset-0 bg-black bg-opacity-50">
            <div className="bg-white rounded-2xl">
                <div className="flex justify-between">
                    <h2>{pokemon?.name}</h2>
                    <button onClick={onClose}>✕</button>
                </div>

                <img src={pokemon.sprites?.front_default} />

                                                    <button onClick={() => onUpdate(pokemon)}>Редактировать</button>
                                                    <button onClick={() => onDelete(pokemon.id)}>Удалить</button>
            </div>
        </div>
    );
};
```

4. Создание сущности (CREATE)

Страница CreatePokemonPage.jsx включает:

- локальное состояние формы,
- валидацию обязательных полей,
- выбор типов покемона,
- добавление сущности через addEntity,
- перенаправление после создания.

Реализация в контексте (addEntity)

- генерируется новый ID,
- создаётся объект покемона нужной структуры,

- добавляется в глобальный массив через setPokemons,
- обновление UI происходит мгновенно без перезагрузки страницы.

Файл: src/pages/CreatePokemonPage.jsx

```
const CreatePokemonPage = () => {
    const { addEntity } = usePokemon();
    const navigate = useNavigate();

    const [formData, setFormData] = useState({
        name: '',
        height: '',
        weight: '',
        image: '',
        types: [],
        hp: '',
        attack: '',
        defense: '',
    });

    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData(prev => ({
            ...prev,
            [name]: value
        }));
    };

    const handleTypeToggle = (type) => {
        setFormData(prev => ({
            ...prev,
            types: prev.types.includes(type)
                ? prev.types.filter(t => t !== type)
                : [...prev.types, type]
        }));
    };

    const handleSubmit = (e) => {
        e.preventDefault();

        if (!formData.name.trim()) {
```

```
            alert('Введите имя покемона!');
            return;
        }

        if (formData.types.length === 0) {
            alert('Выберите хотя бы один тип!');
            return;
        }

        addEntity(formData);
        navigate('/');
    };

    return (
        <form onSubmit={handleSubmit}>
                    <input  type="text"  name="name"  value={formData.name}
onChange={handleChange} />
                <input type="number" name="height" value={formData.height}
onChange={handleChange} />

            {pokemonTypes.map(type => (
                <button
                    type="button"
                    onClick={() => handleTypeToggle(type)}
                     className={formData.types.includes(type) ? 'selected'
: ''}
                >
                    {type}
                </button>
            ))}

            <button type="submit">Создать покемона</button>
        </form>
    );
};
```

5. Редактирование сущности (UPDATE)

На странице EditPokemonPage.jsx:

- данные формы предзаполняются значениями выбранного покемона,

- возможна корректировка имени, роста, веса, изображения, типов и характеристик,
- после подтверждения вызывается метод updateEntity,
- выполняется навигация назад на страницу деталей.

Реализация в контексте (updateEntity)

- происходит замена объекта в массиве,
- используются map() и spread-оператор,
- обновляются только изменённые поля, остальные сохраняются.

Файл: src/pages/EditPokemonPage.jsx

```
const EditPokemonPage = () => {
    const { id } = useParams();
    const { getById, updateEntity } = usePokemon();
    const navigate = useNavigate();

    const pokemon = getById(id);

    const getInitialFormData = () => {
        if (pokemon) {
            return {
                id: pokemon.id,
                name: pokemon.name,
                height: pokemon.height,
                weight: pokemon.weight,
                image: pokemon.sprites?.front_default || '',
                types: pokemon.types?.map(t => t.type.name) || [],
                          hp: pokemon.stats?.find(s => s.stat.name ===
'hp')?.base_stat || '',
                       attack: pokemon.stats?.find(s => s.stat.name ===
'attack')?.base_stat || '',
                      defense: pokemon.stats?.find(s => s.stat.name ===
'defense')?.base_stat || '',
            };
        }
        return {};
    };

    const [formData, setFormData] = useState(getInitialFormData);
```

```
    const handleSubmit = (e) => {
        e.preventDefault();
        updateEntity(formData);
        navigate(`/pokemon/${id}`);
    };

    return (
        <form onSubmit={handleSubmit}>
                    <input  type="text"  name="name"  value={formData.name}
onChange={handleChange} />
            <button type="submit">Сохранить изменения</button>
        </form>
    );
};
```

6. Удаление сущности (DELETE)

На странице PokemonDetailPage.jsx:

- реализована кнопка “Удалить”,
- используется модальное окно подтверждения (ConfirmModal),
- после подтверждения вызывается deleteEntity,
- пользователь перенаправляется на главную страницу.

Реализация в контексте (deleteEntity)

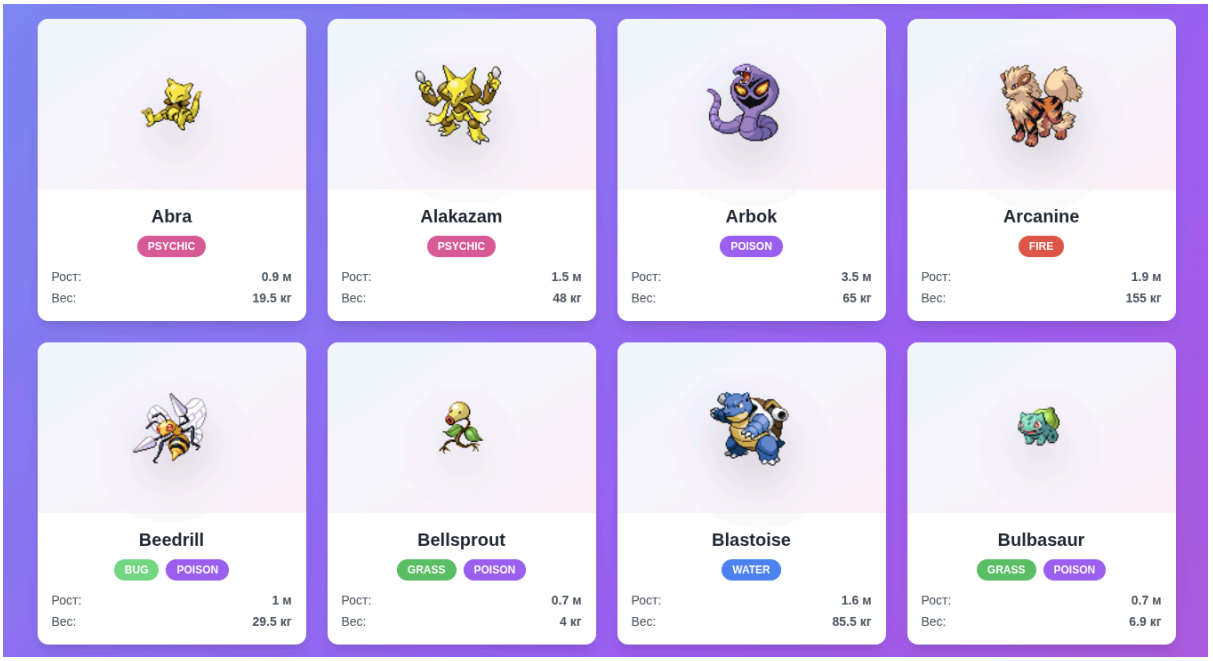
- используется filter() для исключения элемента из массива,
- обновление состояния приводит к перерисовке UI.

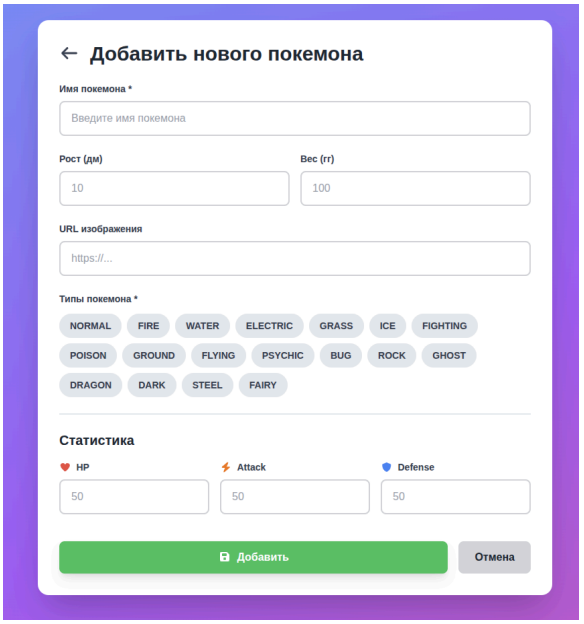
Файл: PokemonDetailPage.jsx

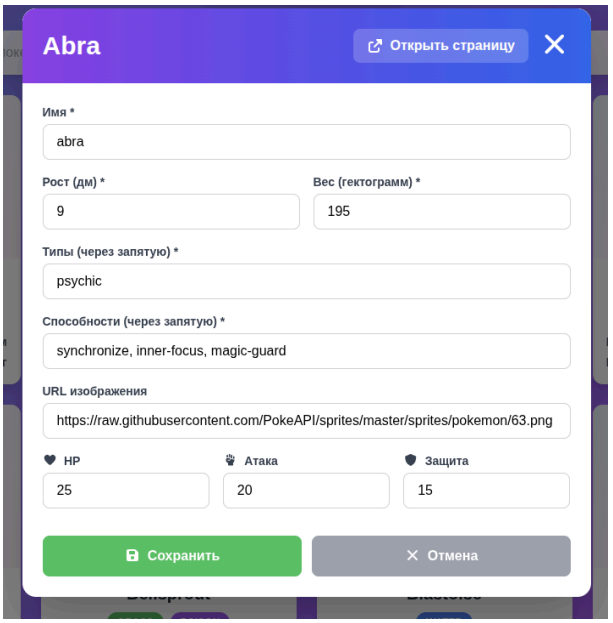
```
const PokemonDetailPage = () => {
    const { id } = useParams();
    const { deleteEntity } = usePokemon();
    const navigate = useNavigate();
    const [showConfirm, setShowConfirm] = useState(false);

    const handleDelete = () => {
        setShowConfirm(true);
    };
```

```
    const handleConfirmDelete = () => {
        deleteEntity(id);
        navigate('/');
    };

    return (
        <div>
            <button onClick={handleDelete}>Удалить</button>

            <ConfirmModal
                isOpen={showConfirm}
                message="Вы уверены, что хотите удалить этого покемона?"
                onConfirm={handleConfirmDelete}
                onCancel={() => setShowConfirm(false)}
            />
        </div>
    );
};
```

Результаты (скриншоты)







Вывод

В ходе выполнения лабораторной работы были достигнуты поставленные цели: создано React-приложение с реализацией всех основных CRUD-операций, обеспечено корректное взаимодействие компонентов, и настроено получение данных из внешнего API. Реализация позволила закрепить навыки работы с хуками useState и useEffect, а также углубить понимание принципов построения компонентной архитектуры.

Работа с внешним API (PokéAPI) дала возможность на практике отработать обработку асинхронных запросов, управление состоянием загрузки и работу с полученными данными. Была создана контекстная структура для хранения состояния приложения, что упростило доступ к данным из разных компонентов.

В процессе разработки были реализованы такие важные элементы интерфейса, как пагинация, модальные окна и страницы редактирования/создания сущностей. Тестирование функционала показало, что интерфейс реагирует на действия пользователя корректно, изменения отображаются без перезагрузки страницы, а навигация между компонентами происходит плавно и логично.

Таким образом, лабораторная работа позволила на практике освоить основные подходы разработки одностраничных приложений на React, научиться работать с API, а также лучше понять принципы организации проекта, разделения логики по компонентам и применения современных инструментов фронтенд-разработки. Полученные знания и навыки могут служить хорошей основой для дальнейшего изучения фреймворков, оптимизации приложения и расширения функциональности.