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

Лабораторная работа №3

A verificat:
Denis Negura

A elaborat:
Ianciuc Nadejda

Chișinău, 2025

План

Цели работы

1. Научится использовать ContextAPI для хранения сущностей и передачи сущностей
2. Реализовать маршрутизацию с помощью react-router-dom
3. Научиться передавать данные между страницами без перезагрузки и без props drilling.

Реализация требований

1. Создание и настройка Context API

Основное:

- createContext() - создаёт объект контекста
- PokemonProvider - оборачивает приложение, предоставляет state и методы
- usePokemon() - кастомный хук для доступа к данным

State:

- pokemons - массив сущностей\
- loading - индикатор загрузки
- ITEMS_PER_PAGE - количество покемонов на странице

Методы:

1. fetchEntities() - загружает данные с API, параллельно получает детали каждого покемона
2. getById(id) - возвращает покемона по идентификатору
3. addEntity(entity) - добавляет нового покемона, создаёт уникальный id
4. updateEntity(entity) - обновляет выбранного покемона по id
5. deleteEntity(id) - удаляет покемона по id

Преимущества:

- Нет необходимости передавать props через все компоненты
- Централизованное управление CRUD
- Доступ к данным из любого компонента

Файл: src/context/PokemonContext.jsx

```
import React, { createContext, useState, useContext, useEffect } from 'react';

// Создаём контекст
const PokemonContext = createContext();

// Хук для использования контекста
export const usePokemon = () => {
    const context = useContext(PokemonContext);
    if (!context) {
            throw new Error('usePokemon должен использоваться внутри PokemonProvider');
    }
    return context;
};

// Провайдер контекста
export const PokemonProvider = ({ children }) => {
    const [pokemons, setPokemons] = useState([]);
    const [loading, setLoading] = useState(true);

    const ITEMS_PER_PAGE = 20;
    const TOTAL_ITEMS = 100;

    // ============================
    // A) fetchEntities() - загрузка данных с API
    // ============================
    const fetchEntities = async () => {
        try {
            setLoading(true);
            const response = await fetch(

`https://pokeapi.co/api/v2/pokemon?limit=${TOTAL_ITEMS}&offset=0`
            );
            const data = await response.json();

            // Загружаем детальную информацию параллельно
            const detailedPokemons = await Promise.all(
```

```
                data.results.map(async (pokemon) => {
                    const res = await fetch(pokemon.url);
                    const pokemonData = await res.json();
                    return {
                        ...pokemonData,
                                                                uniqueKey:
`${pokemonData.id}-${Date.now()}-${Math.random()}`
                    };
                })
            );

            setPokemons(detailedPokemons);
            setLoading(false);
        } catch (error) {
            console.error('Ошибка загрузки покемонов:', error);
            setLoading(false);
        }
    };

    // Загружаем данные один раз при монтировании
    useEffect(() => {
        fetchEntities();
    }, []);

    // ==============================
    // B) getById(id) - получить сущность по идентификатору
    // ==============================
    const getById = (id) => {
        return pokemons.find(pokemon => pokemon.id === parseInt(id));
    };

    // ==============================
    // C) addEntity(entity) - добавить новую сущность
    // ==============================
    const addEntity = (newPokemon) => {
        const pokemon = {
            id: Math.max(...pokemons.map(p => p.id), 0) + 1,
            name: newPokemon.name.toLowerCase(),
            height: parseInt(newPokemon.height) || 0,
            weight: parseInt(newPokemon.weight) || 0,
```

```
            sprites: {
                                        front_default:  newPokemon.image  ||
'https://via.placeholder.com/150'
            },
            types: Array.isArray(newPokemon.types)
                ? newPokemon.types.map(type => ({
                    type: { name: type }
                }))
                : [],
            stats: [
                                    { stat: { name: 'hp' }, base_stat:
parseInt(newPokemon.hp) || 0 },
                                { stat: { name: 'attack' }, base_stat:
parseInt(newPokemon.attack) || 0 },
                                { stat: { name: 'defense' }, base_stat:
parseInt(newPokemon.defense) || 0 },
            ],
            uniqueKey: `${Date.now()}-${Math.random()}`
        };

        setPokemons(prevPokemons => {
            const newList = [pokemon, ...prevPokemons];
            return newList;
        });

        return pokemon;
    };

    // ============================
    // D) updateEntity(entity) - обновить данные существующей сущности
    // ============================
    const updateEntity = (updatedPokemon) => {
        setPokemons(prevPokemons =>
            prevPokemons.map(pokemon =>
                pokemon.id === updatedPokemon.id
                    ? {
                        ...pokemon,
                        name: updatedPokemon.name.toLowerCase(),
                        height: parseInt(updatedPokemon.height) || 0,
                        weight: parseInt(updatedPokemon.weight) || 0,
```

```
                        sprites: {
                            ...pokemon.sprites,
                                front_default: updatedPokemon.image ||
pokemon.sprites.front_default
                        },
                        types: updatedPokemon.types.map(type => ({
                            type: { name: type }
                        })),
                        stats: [
                                    { stat: { name: 'hp' }, base_stat:
parseInt(updatedPokemon.hp) || 0 },
                                  { stat: { name: 'attack' }, base_stat:
parseInt(updatedPokemon.attack) || 0 },
                                 { stat: { name: 'defense' }, base_stat:
parseInt(updatedPokemon.defense) || 0 },
                        ]
                    }
                    : pokemon
            )
        );
    };

    // ============================
    // E) deleteEntity(id) - удалить из списка
    // ============================
    const deleteEntity = (id) => {
        setPokemons(prevPokemons =>
            prevPokemons.filter(pokemon => pokemon.id !== parseInt(id))
        );
    };

    // Значение контекста (что будет доступно компонентам)
    const value = {
        pokemons,
        loading,
        ITEMS_PER_PAGE,
        fetchEntities,
        getById,
        addEntity,
        updateEntity,
```

```
        deleteEntity,
    };

    return (
        <PokemonContext.Provider value={value}>
            {children}
        </PokemonContext.Provider>
    );
};
```

2. Маршрутизация

Основное:

- BrowserRouter - оборачивает приложение для работы маршрутов
- PokemonProvider - даёт доступ к контексту на всех страницах
- Routes - описывает пути и компоненты

Список маршрутов:

Путь	Компонент	Назначение
/	HomePage	Главная страница со списком
/pokemon/:id	PokemonDetailPage	Детальная информация
/create	CreatePokemonPage	Создание нового покемона
/edit/:id	EditPokemonPage	Редактирование покемона

Файл: src/App.jsx

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { PokemonProvider } from './context/PokemonContext';
import HomePage from './pages/HomePage';
import PokemonDetailPage from './pages/PokemonDetailPage';
import CreatePokemonPage from './pages/CreatePokemonPage';
import EditPokemonPage from './pages/EditPokemonPage';
import './App.css';

function App() {
```

```
    return (
                    <BrowserRouter  future={{  v7_startTransition:  true,
v7_relativeSplatPath: true }}>
            <PokemonProvider>
                <Routes>
                    <Route path="/" element={<HomePage />} />
                                                <Route  path="/pokemon/:id"
element={<PokemonDetailPage />} />
                     <Route path="/create" element={<CreatePokemonPage />}
/>
                     <Route path="/edit/:id" element={<EditPokemonPage />}
/>
                </Routes>
            </PokemonProvider>
        </BrowserRouter>
    );
}

export default App;
```

Описание страниц:

- Главная страница

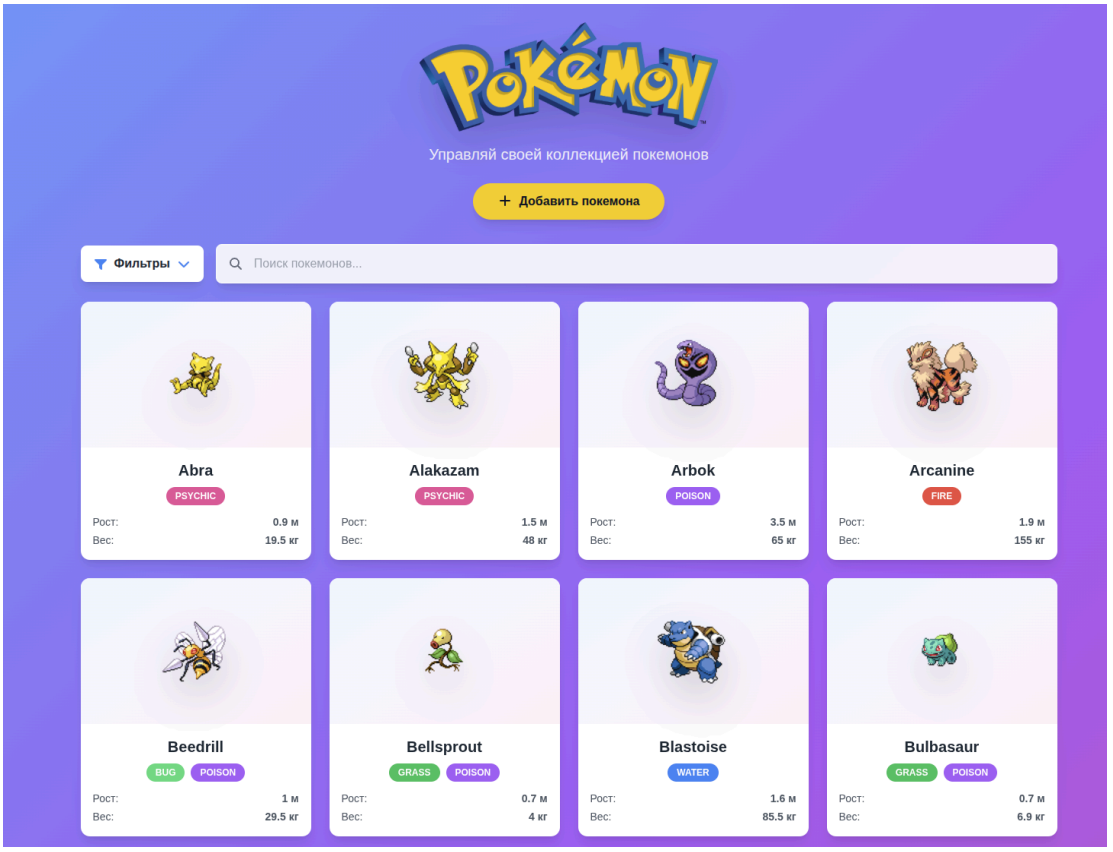
  Отображение карточек с изображением, типами и базовой информацией

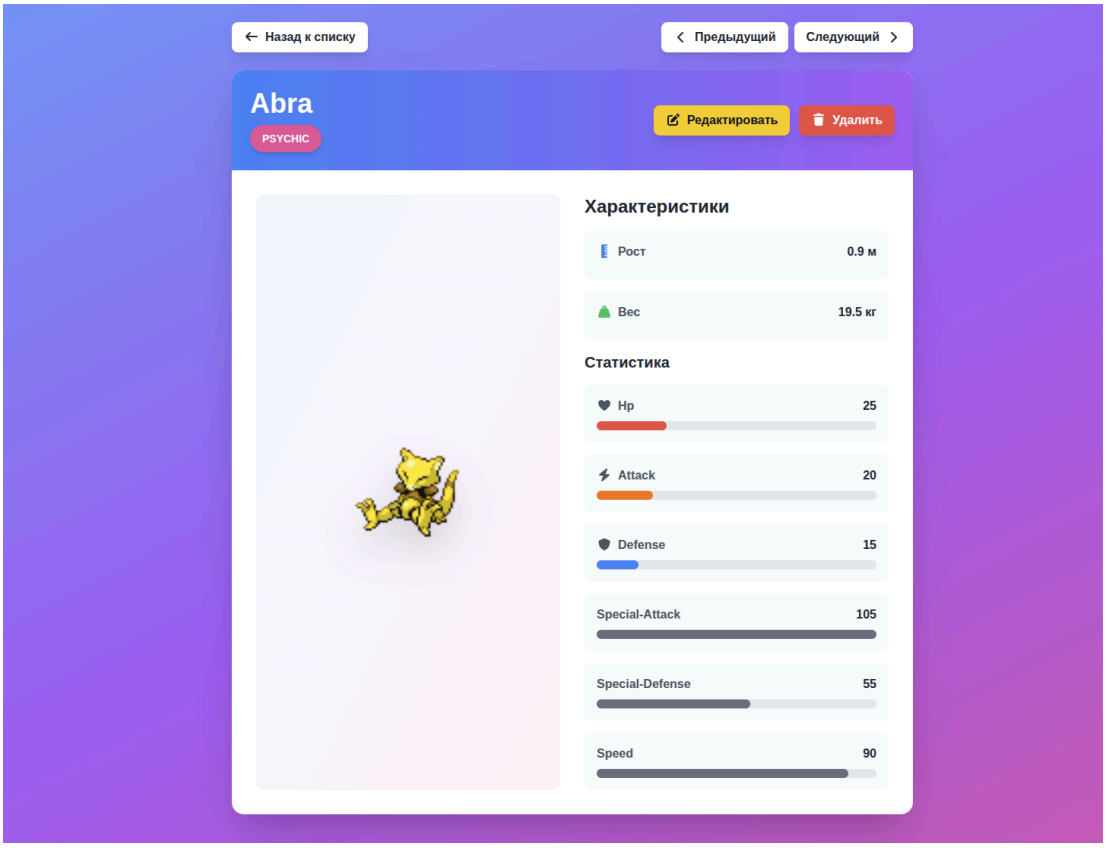
  Клик на карточку открывает модальное окно с деталями и действиями: редактировать, удалить, перейти на страницу деталей
- Страница деталей

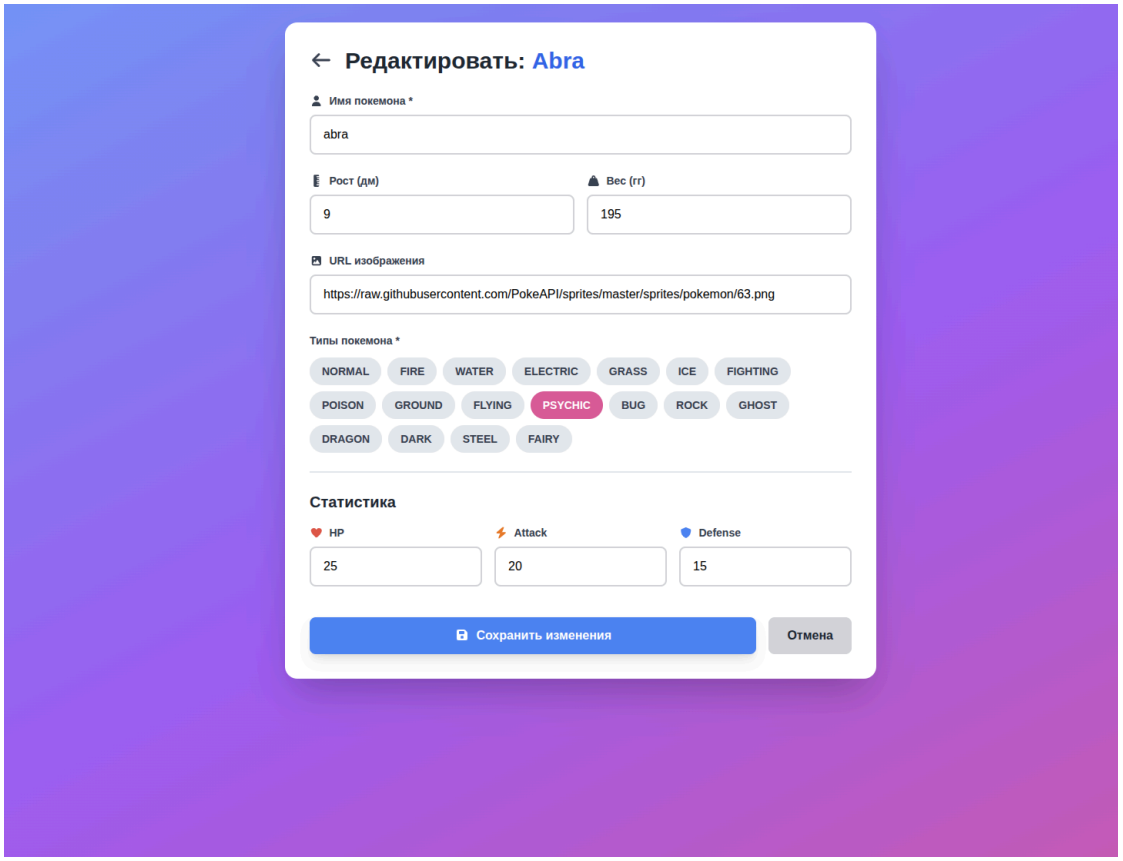
  Переход к предыдущему/следующему покемону; Просмотр характеристик, изображений, типов; Кнопки редактирования и удаления.
- Страницы создания и редактирования

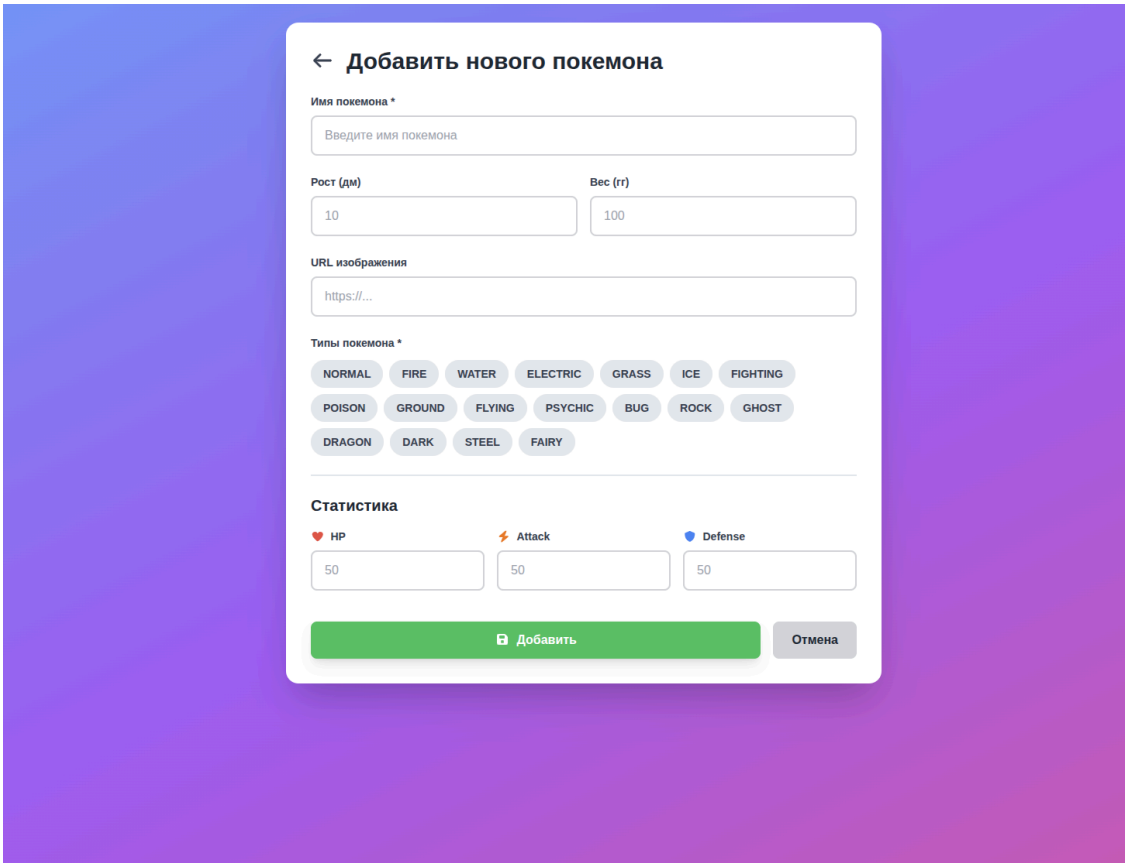
  Изменение данных существующего покемона. Сохранение через updateEntity()

Результаты (скриншоты)






Редактировать: Abra
Имя покемона *
abra
Рост (дм)
9
Вес (гг)
195
URL изображения
https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/63.png
Типы покемона *
NORMAL FIRE WATER ELECTRIC GRASS ICE FIGHTING
POISON GROUND FLYING PSYCHIC BUG ROCK GHOST
DRAGON DARK STEEL FAIRY
Статистика
HP
25
Attack
20
Defense
15
Сохранить изменения
Отмена


Добавить нового покемона
Имя покемона *
Введите имя покемона
Рост (дм)
10
Вес (гг)
100
URL изображения
https://...
Типы покемона *
NORMAL FIRE WATER ELECTRIC GRASS ICE FIGHTING
POISON GROUND FLYING PSYCHIC BUG ROCK GHOST
DRAGON DARK STEEL FAIRY
Статистика
HP
50
Attack
50
Defense
50
Добавить
Отмена

Вывод

В результате выполнения данной лабораторной работы были изучены и применены ключевые инструменты, необходимые для разработки современных одностраничных веб-приложений. В ходе работы была реализована структура SPA, организована маршрутизация между страницами с использованием библиотеки react-router-dom, а также создан глобальный контекст приложения для хранения и передачи данных без необходимости prop drilling.

Использование Context API позволило понять принципы управления состоянием на уровне всего приложения и упростило взаимодействие между компонентами. Была продемонстрирована возможность передавать выбранные сущности между страницами без перезагрузки страницы и без обращения к серверу. Такая архитектура особенно актуальна в проектах, где требуется централизованное хранение данных и синхронизация состояния.

Кроме того, выполнение лабораторной работы помогло закрепить навыки структурирования компонентов, разделения логики по уровням и грамотного использования маршрутов. Были отработаны базовые подходы к созданию динамических интерфейсов, работе с параметрами маршрутов, отображению списка элементов и детальной информации о каждом из них.

В итоге была создана полноценная mini-SPA, в которой корректно работает навигация, контекст, передача данных и отображение сущностей. Полученные навыки являются важным этапом в изучении React и послужат основой для разработки более сложных и функциональных веб-приложений в дальнейшем.