САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа № 2
“Взаимодействие с внешним API”

Выполнил:
Коротин А.М.

Группа:
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Вариант 5. Музыкальный плеер. Вам нужно привязать то, что Вы делали в ЛР1 к внешнему API средствами fetch/axios/xhr.

Ход работы

В качестве внешнего API с музыкой был выбран сервис https://rapidapi.com, а именно API сервиса Deezer. В последнее время у сервиса наблюдаются проблемы с работой из РФ, поэтому разработка производилась с использованием прокси-сервера.

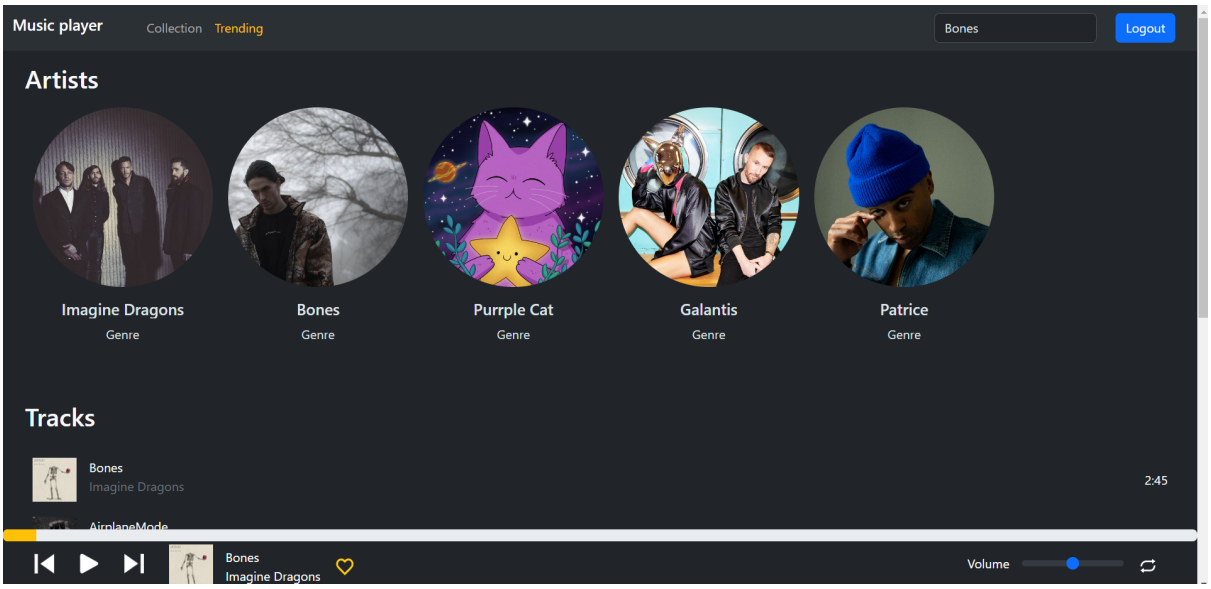

Рисунок 1 - Интерфейс поиска

Запрос к сервису Deezer, согласно документации, производится следующим образом:

```
import axios from 'https://cdn.jsdelivr.net/npm/axios@1.5.1/+esm';

const apiKeys = await fetch("/res/API_KEY.json")
                    .then((r) => r.json());

export const search = async (query) => {
    const options = {
        method: 'GET',
        url: 'https://deezerdevs-deezer.p.rapidapi.com/search',
        params: {q: query},
        headers: apiKeys
    };

    return await axios.request(options);
};
```

Рисунок 2 - Запрос к внешнему API

Для расположения элементов интерфейса использовалось подобие компонентного подхода: для каждой компоненты создавались функции с параметрами, которые возвращают готовый элемент:

```
const Playlist = (id, name, description) => {
    const markup = `
        <div type="button" class="card border-hover ms-1" playlistId=${id} data-bs-toggle="modal" data-bs-target="#playlist-modal">
            <img src="https://placekitten.com/300/200" class="card-img-top" alt="image">
            <div class="card-body">
                <h5 class="card-title text-truncate">${name}</h5>
                <h6 class="card-text text-truncate">${description}</h6>
            </div>
        </div>
    `

    let template = document.createElement('template');
    template.innerHTML = markup;
    return template.content;
}

export default Playlist;
```

Рисунок 3 - Функция для создания компоненты плейлиста

Также для имитации аутентификации/авторизации, хранения плейлистов из треков сервиса Deezer использовался json-server:

```
import axios from 'https://cdn.jsdelivr.net/npm/axios@1.5.1/+esm';
import { alert, checkAuth } from './auth.js';

const login = async (event) => {
    await event.preventDefault();

    const inputs = Array.from(event.target.querySelectorAll("input"));

    const loginData = {};

    for (const input of inputs) {
        loginData[input.name] = input.value;
    }

    axios.post("http://localhost:3000/login", loginData)
        .then((response) => {
            window.localStorage.user = JSON.stringify(response.data.user);
            window.localStorage.accessToken = response.data.accessToken;
            checkAuth();
        })
        .catch((reason) => {
            alert("Invalid email or password. Please, try again", "alert-placeholder");
        });
}

document.querySelector("form").addEventListener("submit", login);
document.addEventListener("DOMContentLoaded", checkAuth);
```

Рисунок 4 - Запрос аутентификации на json-server

```
export const getPlaylists = async (accessToken) => {
    const options = {
        method: "GET",
        headers: {Authorization: `Bearer ${accessToken}`},
        url: "http://localhost:3000/600/playlists?_embed=tracks"
    }

    return await axios.request(options);
}


export const getPlaylist = async (accessToken, playlistId) => {
    const options = {
        method: "GET",
        headers: {Authorization: `Bearer ${accessToken}`},
        url: `http://localhost:3000/600/playlists/${playlistId}?_embed=tracks`
    }

    return await axios.request(options);
}

export const createPlaylist = async (accessToken, playlist) => {
    const options = {
        method: "POST",
        headers: {Authorization: `Bearer ${accessToken}`},
        url: "http://localhost:3000/600/playlists",
        data: playlist
    }

    return await axios.request(options);
}
```

Рисунок 5 - Запросы для плейлистов к json-server

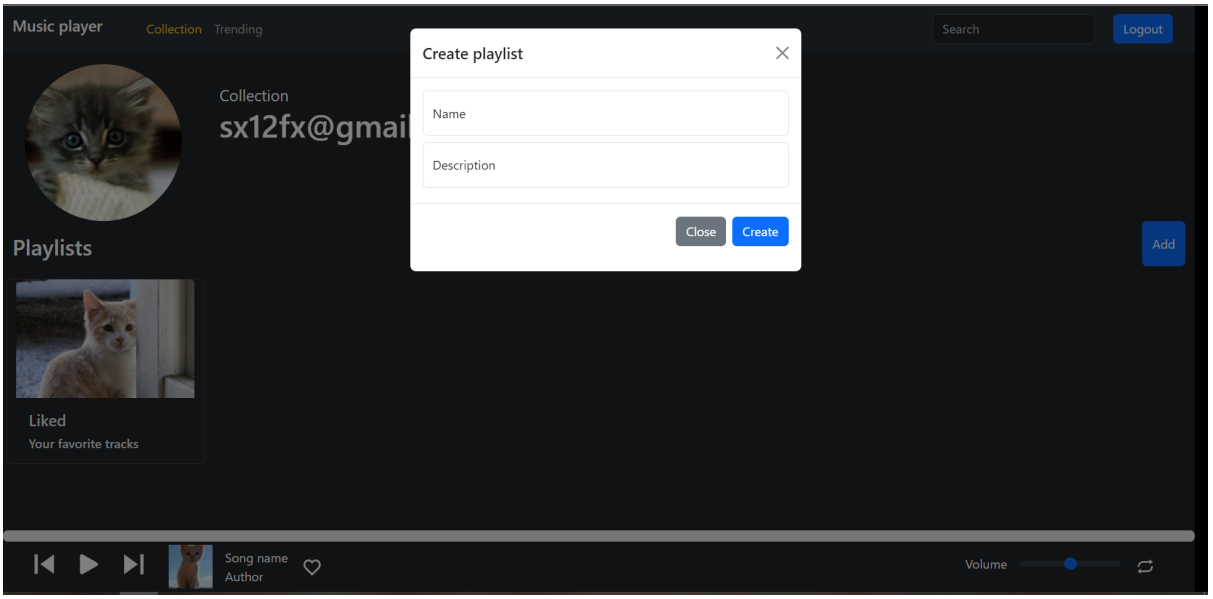

Рисунок 6 - Форма создания плейлиста с использованием модального окна

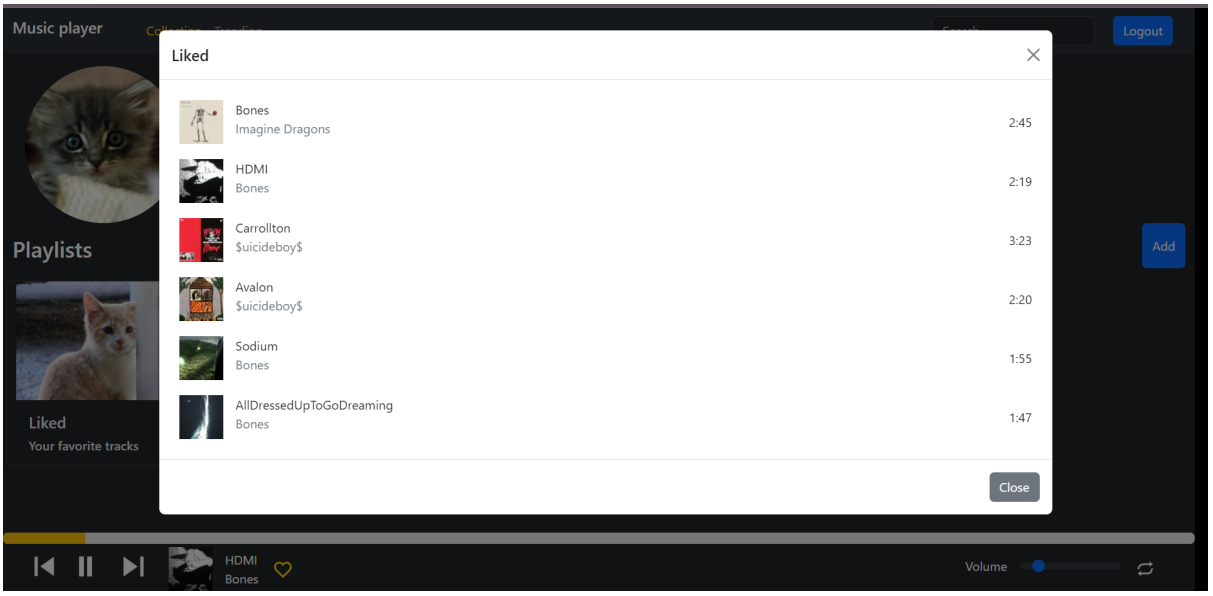

Рисунок 7 - Список треков плейлиста с использованием модального окна

```
const playlistOnClick = async (playlistId) => {
    let result = await getPlaylist(window.localStorage.accessToken, playlistId);
    if (result.status === 304) { //not changed => use cached
        result = JSON.parse(window.localStorage.playlists).find((p) => p.id == playlistId);
    } else {
        result = result.data;
    }
    const tracks = result.tracks.map((t) => {
        let minutes = Math.floor(t.duration / 60);
        let seconds = Math.floor(t.duration - minutes * 60);
        seconds = seconds < 10 ? `0${seconds}`: String(seconds);
        let track = Track(t.title, t.artist.name, `${minutes}:${seconds}`, t.album.cover_small);
        return track;
    });

    playlistTitle.innerText = result.name;

    modalTrackContainer.replaceChildren(...tracks);

    const topLevelChildren = document.querySelectorAll("#modal-track-container > div");

    topLevelChildren.forEach((node, index) => {
        node.addEventListener("click", (e) => {
            window.localStorage.currentTrackIndex = index;
            window.localStorage.lastSearched = JSON.stringify(result.tracks);
            startAudio();
        });
    });
}
```

Рисунок 8 - Функция заполнения модального окна плейлиста

Продемонстрируем функциональность добавления трека в список понравившихся, нажмем на иконку сердечка справа от названия трека:

Рисунок 9 - Трек для добавления

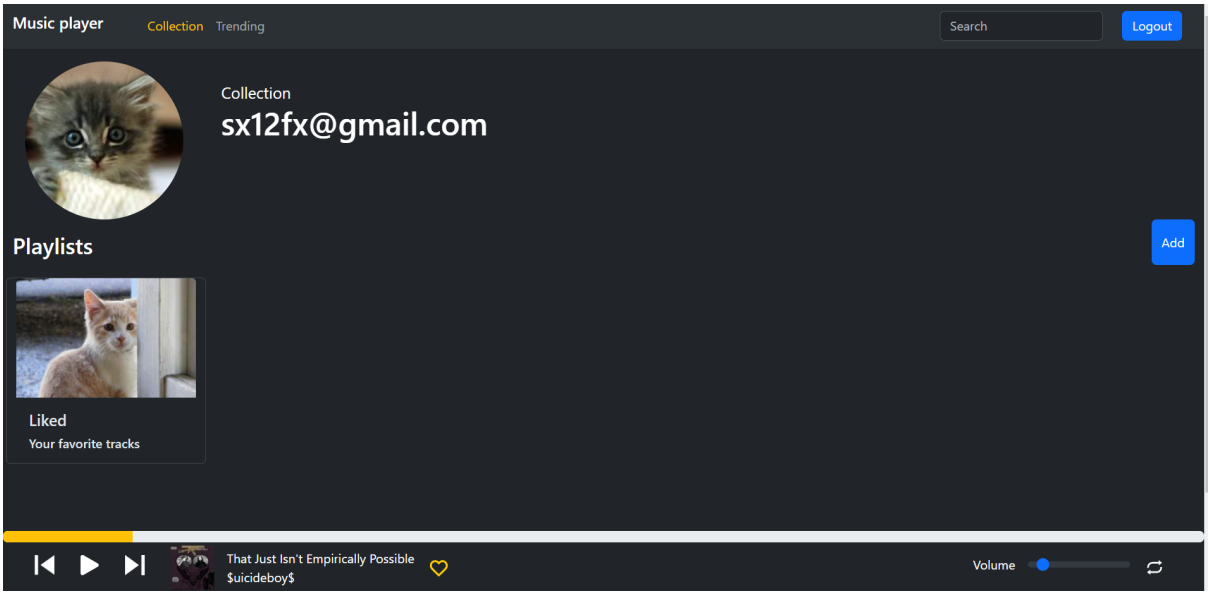

Рисунок 10 - Добавленный трек

Как можно заметить, иконка изменила свое состояние и трек появился в плейлисте понравившихся треков:

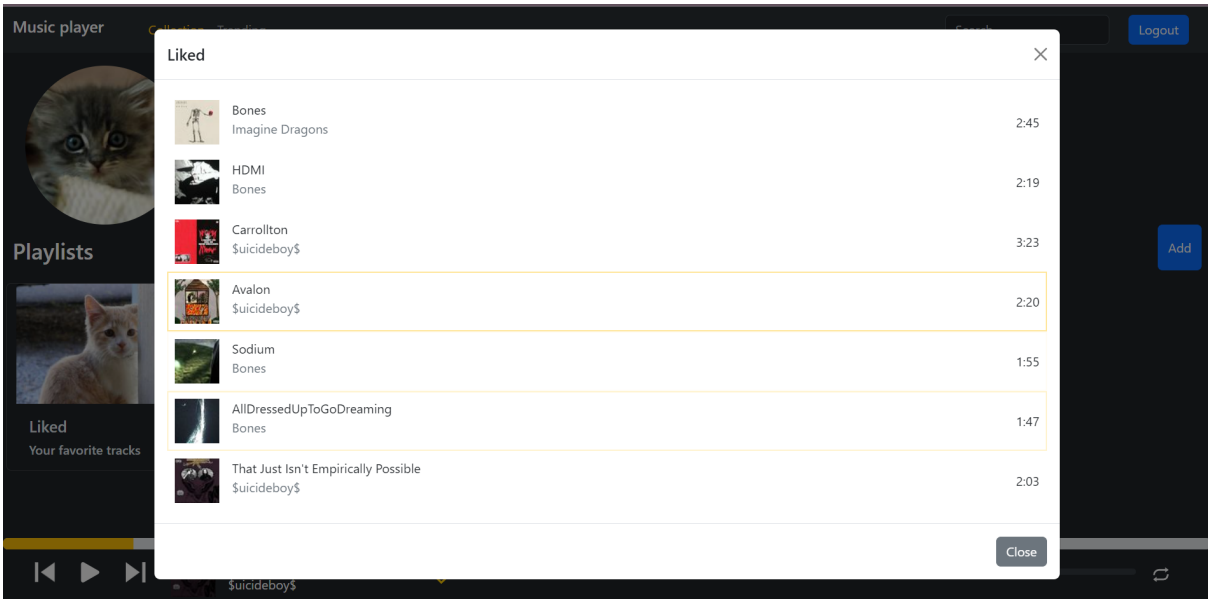

Рисунок 11 - Новый трек в плейлисте

Вывод:

В ходе выполнения лабораторной работы я научился пользоваться средствами axios для взаимодействия со внешними API из Javascript,