Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Технології розроблення програмного забезпечення

Лабораторна робота №9

«РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE»

Виконала:
Студентка групи ІА-22
Микитенко Ірина

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: «РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE»

Мета: ознайомитися з різними видами взаємодії додатків: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE та набути практичних навичок їх застосування. Реалізувати частину функціоналу програми за допомогою одного з розглянутих видів взаємодії.

Хід роботи

..1 Музичний програвач (iterator, command, memento, facade, visitor, client-server)

Музичний програвач становить собою програму для програвання музичних файлів або відтворення потокової музики з можливістю створення, запам'ятовування і редагування списків програвання, перемішування/повторення (shuffle/repeat), розпізнавання різних аудіо-форматів, еквалайзер.

1. Короткі теоритичні відомості

Клієнт-серверні додатки

Клієнт-серверні додатки — це тип розподілених систем, які складаються з двох компонентів: клієнта (для взаємодії з користувачем) і сервера (для обробки та зберігання даних).

Типи клієнтів:

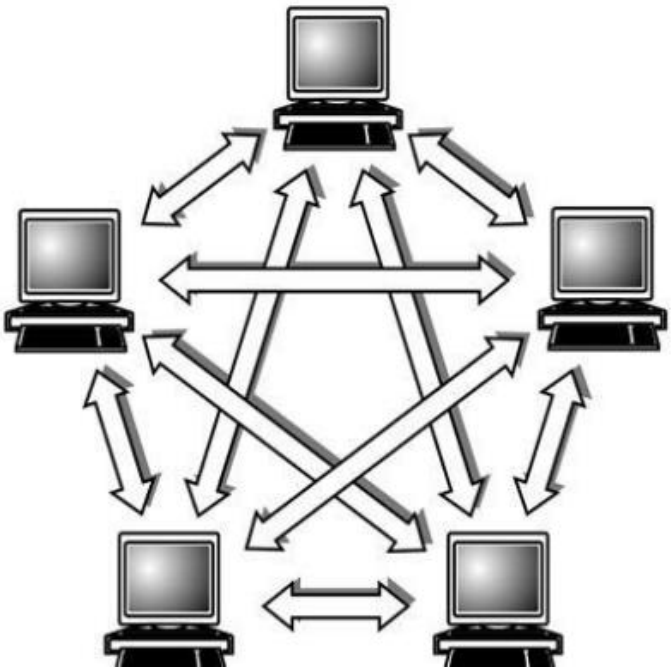
1. **Тонкий клієнт** — передає обробку логіки додатка на сервер і лише відображає результати. Недоліки: високе навантаження на сервер, переваги — захист даних.
2. **Товстий клієнт** — виконує більшість обчислень на клієнтській стороні, зменшуючи навантаження на сервер.

Модель "підписки/видачі": Клієнти підписуються на події, а сервер повідомляє про їх настання. Реалізується через шаблон "спостерігач", що спрощує оновлення даних, але збільшує навантаження на сервер.

3-рівнева структура:

1. **Клієнтська частина** — відповідає за інтерфейс і взаємодію з користувачем.
2. **Загальна частина (middleware)** — містить спільні класи та логіку.
3. **Серверна частина** — реалізує бізнес-логіку, управління даними та їх збереження.

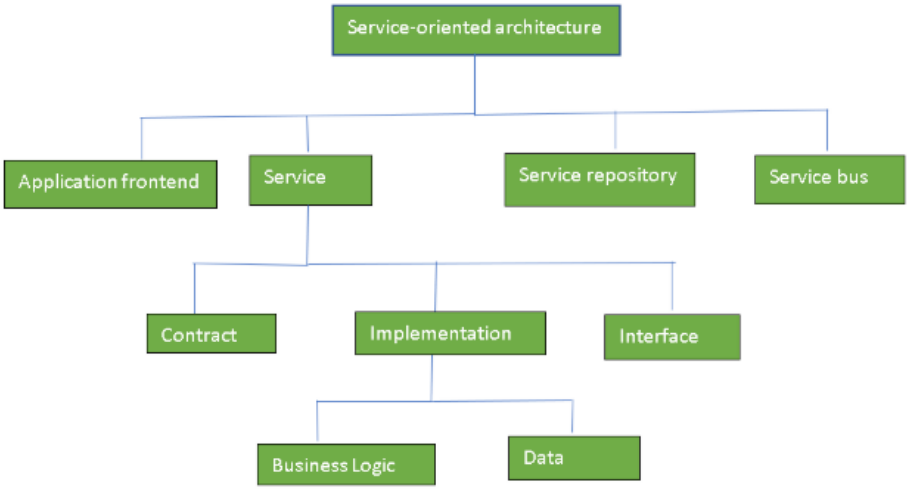
Додатки типу «peer-to-peer»



Peer-to-peer (P2P) додатки організовують рівноправну взаємодію між клієнтами без використання серверів. Всі клієнти обмінюються даними напряму для досягнення спільних цілей. Основними викликами є синхронізація даних та пошук інших клієнтів.

Для пошуку використовується або загальна адреса з переліком підключених клієнтів, або адреси зберігаються у самих клієнтів (структуровані однорангові мережі). Синхронізація даних може здійснюватися через алгоритми порівняння (наприклад, hash) чи узгодження набору для синхронізації. P2P-додатки базуються на спеціальних форматах і протоколах обміну для забезпечення ефективної взаємодії між клієнтами.

Сервіс-орієнтована архітектура (SOA)



Сервіс-орієнтована архітектура (SOA) — це модульний підхід до розробки програмного забезпечення, що базується на використанні розподілених і слабо пов'язаних компонентів зі стандартизованими інтерфейсами.

Системи SOA реалізуються через набір веб-служб, які взаємодіють за протоколами SOAP, REST або іншими. Компоненти SOA ізолюють деталі реалізації, що забезпечує масштабованість, незалежність від платформ і повторне використання.

Модель SaaS є прикладом SOA, де додатки надаються користувачам через Інтернет як послуга. Ключові переваги SaaS: відсутність витрат на обслуговування, оновлення програм, гнучка оплата, і прозорість модернізації.

Мікро-сервісна архітектура

Мікро-сервісна архітектура — це підхід до створення серверних додатків як набору незалежних малих служб, кожна з яких виконується у власному процесі та взаємодіє з іншими через протоколи, такі як HTTP, WebSockets, або AMQP.

Кожна мікрослужба реалізує конкретну бізнес-логіку в обмеженому контексті, розробляється автономно й розгортається незалежно.

Переваги мікро-сервісної архітектури:

- **Гнучкість і масштабованість**: легко адаптується до змін і підтримує високі навантаження.
- **Простота супроводу**: зручне обслуговування навіть у великих комплексних системах.
- **Автономність компонентів**: кожна служба має свій життєвий цикл і не залежить від інших.

2. Реалізація частини функціоналу робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.

У моїй програмі організована **клієнт-серверна архітектура** із впровадженням кількох шаблонів проектування, таких як **Iterator**, **Facade**, **Visitor**, а також використовується структуроване розділення на шари.

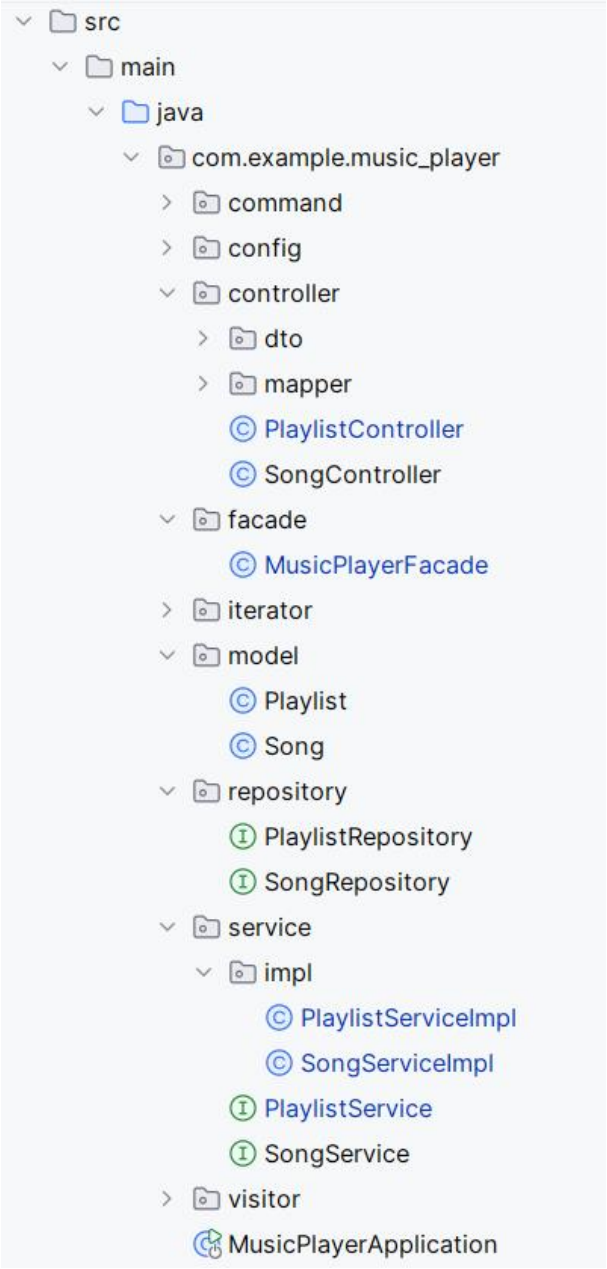


Рисунок 1 - структура сервера

На основі наявної структури програми, функціональні можливості досягаються завдяки взаємодії між компонентами:

Основні компоненти:

1. **Model**:

- Класи Playlist та Song представляють основні дані додатка. Вони містять інформацію про пісні та плейлисти.

2. **Repository**:
   - Класи PlaylistRepository і SongRepository відповідають за збереження, отримання та взаємодію з базою даних.
3. **Service**:
   - Інтерфейси PlaylistService і SongService забезпечують логіку роботи з плейлистами та піснями.
   - Класи PlaylistServiceImpl і SongServiceImpl реалізують цю логіку.
4. **Controller**:
   - PlaylistController і SongController керують запитами від клієнтів (наприклад, отримання плейлиста або додавання пісні).
5. **Iterator**:
   - У папці iterator знаходиться реалізація ітератора для проходження списку пісень у плейлисті.
6. **Facade**:
   - Клас MusicPlayerFacade об’єднує кілька підсистем для спрощення використання функціоналу.
7. **Visitor**:
   - Реалізується для виконання операцій без зміни структури об'єктів.

3. Реалізація взаємодії програми в одній з архітектур відповідно до обраної теми.

Взаємодія компонентів

Програма реалізує взаємодію між різними компонентами через **клієнт-серверну архітектуру**:

- **Запит від клієнта**: Користувач надсилає запит через REST API (наприклад, отримати всі пісні з певного плейлиста).
  Це обробляється в контролерах (PlaylistController або SongController).
- **Бізнес-логіка**: Контролери викликають сервіси (PlaylistService або SongService), які обробляють запити та взаємодіють з репозиторіями для доступу до даних.
- **Дані**: Репозиторії отримують або змінюють інформацію в базі даних.

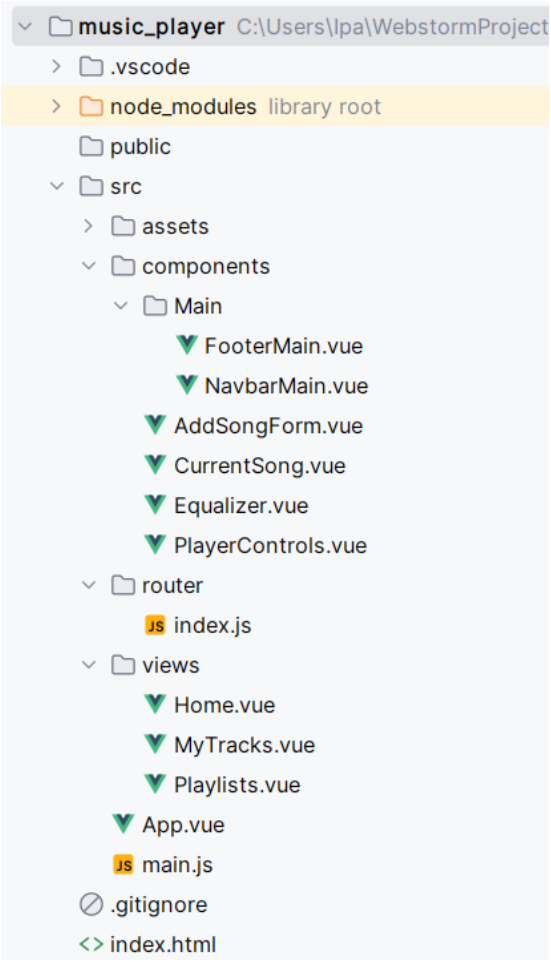


Рисунок 2 - структура клієнта

Взаємодія між бекендом і фронтендом у подібній архітектурі відбувається через API, яке надає бекенд. Ось як це працює на прикладі описаного коду:

1. Запити до бекенду

Фронтенд виконує HTTP-запити до API, яке реалізовано на бекенді. У даному випадку використовується бібліотека axios, щоб надсилати запити. Наприклад:

Отримання списку пісень (fetchSongs):

```
async fetchSongs() {
  try {
    const response = await axios.get( url: 'http://localhost:3000/songs');
    this.songList = response.data;
  } catch (error) {
    console.error("Error fetching songs:", error);
  }
},
```

Рисунок 3 – метод для отримання списку пісень на фронтенді

```
@GetMapping  imykytenko
public List<SongDto> findAll() {
    return musicPlayerFacade.getAllSongs().stream().map(songMapper::toDto).toList();
}
```

Рисунок 3 – отримання списку пісень на бекенді

Фронтенд звертається до бекенду за списком пісень, який повертається у форматі JSON. Ці дані потім використовуються для відображення пісень у списку.

2. Бекенд

- **API для обробки запитів**: Бекенд реалізує RESTful API, яке обробляє CRUD-операції:
    - **GET** для отримання списків пісень, плейлистів тощо.
    - **POST** для створення нових плейлистів чи додавання пісень.
    - **PUT** для редагування існуючих об'єктів.
    - **DELETE** для видалення пісень або плейлистів.

Приклад контроллера:

```
@RestController  imykytenko *
@RequestMapping("/playlists")
@RequiredArgsConstructor
@CrossOrigin(origins = {"http://localhost:5173"})
public class PlaylistController {

    @Autowired
    private MusicPlayerFacade musicPlayerFacade;

    @Autowired
    private PlaylistMapper playlistMapper;

    @PostMapping("/add")  imykytenko
    public ResponseEntity<PlaylistDto> createPlaylist(@RequestBody PlaylistCreationDto playlistDto) {
        var playlist = musicPlayerFacade.createPlaylist(playlistMapper.toEntity(playlistDto));
        return new ResponseEntity<>(playlistMapper.toDto(playlist), HttpStatus.CREATED);
    }

    @GetMapping("/{id}")  imykytenko
    public ResponseEntity<PlaylistDto> getPlaylistById(@PathVariable Long id) {
        Playlist playlist = musicPlayerFacade.getPlaylistById(id);
        return playlist != null ? ResponseEntity.ok(playlistMapper.toDto(playlist)) : ResponseEntity.notFound().build();
    }

    @GetMapping  imykytenko
    public List<PlaylistDto> getAllPlaylists() {
        return musicPlayerFacade.getAllPlaylists().stream().map(playlistMapper::toDto).toList();
    }
```

Рисунок 4 – PlaylistController

- **Використання бази даних**: Дані зберігаються в базі даних. Наприклад:
  - Таблиця songs для зберігання інформації про пісні.
  - Таблиця playlists для плейлистів.
  - Таблиця playlist_songs для зв’язку між піснями та плейлистами.
- **Сервісний рівень**: У бекенді є сервіси, які виконують бізнес-логіку, наприклад, обробляють запити на відтворення наступної пісні або попередньої, тощо.

Приклад Сервісу:

```
@Service  imykytenko *                                                    Review Mode
public class PlaylistServiceImpl implements PlaylistService{
    @Autowired
    private final PlaylistRepository playlistRepository;
    @Autowired
    private final SongRepository songRepository;
    private final Map<Long, Iterator<Song>> iterators = new HashMap<>();  4 usages
    private final PlaylistCommandInvoker commandInvoker = new PlaylistCommandInvoker();  2 usages

    @Autowired  imykytenko
    public PlaylistServiceImpl(PlaylistRepository playlistRepository, SongRepository songRepository) {
        this.playlistRepository = playlistRepository;
        this.songRepository = songRepository;
    }

    @Override  1 usage  imykytenko
    public Playlist createPlaylist(Playlist playlist) { return playlistRepository.save(playlist); }

    @Override  3 usages  imykytenko
    public Playlist getPlaylistById(Long id) { return playlistRepository.findById(id).orElse( other: null); }

    @Override  1 usage  imykytenko
    public List<Playlist> getAllPlaylists() { return playlistRepository.findAll(); }

    @Override  1 usage  imykytenko
    public Playlist updatePlaylist(Long id, Playlist playlist) {
        if (playlistRepository.existsById(id)) {
            playlist.setId(id);
            return playlistRepository.save(playlist);
        }
        return null;
    }
```

Рисунок 5 – PlaylistService

3. Відповіді від бекенду

Кожен запит повертає відповідь:

- **Успішна відповідь**: Дані у форматі JSON, які фронтенд використовує для оновлення інтерфейсу.

- **Помилка**: Наприклад, якщо пісня не знайдена, бекенд повертає код помилки (404), і фронтенд може показати відповідне повідомлення користувачеві.

4. Інтерактивність фронтенду

- На фронтенді Vue.js відповідає за реактивність. Зміна даних (наприклад, список пісень) автоматично оновлює DOM.
- Кнопки, такі як "Створити плейлист" чи "Редагувати плейлист", викликають відповідні методи, які надсилають запити до бекенду.

Приклад кнопок які відповідно виконують дії:

```
<template>  Show component usages
  <div class="controls">
    <button @click="previousSong" :disabled="currentIndex <= 0" class="btn">Назад</button>
    <button @click="togglePlayPause" class="btn">
      {{ isPlaying ? 'Pause' : 'Play' }}
    </button>
    <button @click="nextSong" :disabled="currentIndex >= songList.length - 1" class="btn">Далі</button>
    <button @click="shuffleSongs" class="btn">Перемішати</button>
    <button @click="repeatSong" class="btn">Повторити пісню</button>
  </div>
</template>
```

Рисунок 6 – Кнопки на фронтенді

5. Архітектурна модель

Для такої взаємодії часто використовується **архітектура "клієнт-сервер"**:

- **Фронтенд (Vue.js)**: Відповідає за відображення інтерфейсу, реактивність та інтерактивність.
- **Бекенд (Spring)**: Обробляє запити, виконує бізнес-логіку та надає дані через REST API.
- **База даних**: Зберігає інформацію про пісні, плейлисти, та їхній зв'язок

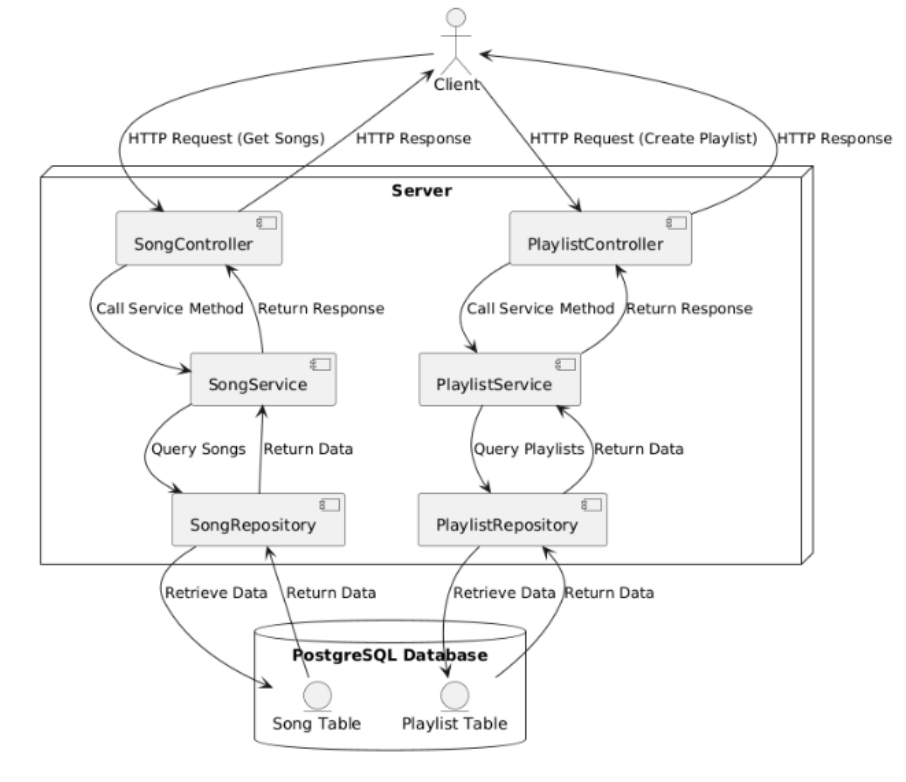


Рисунок 7 – діаграма клієнт-серверної взаємодії

4. Готова реалізація проекту

На рисунку зображено графічний інтерфейс системи. На початку нас зустрічає головна сторінка.

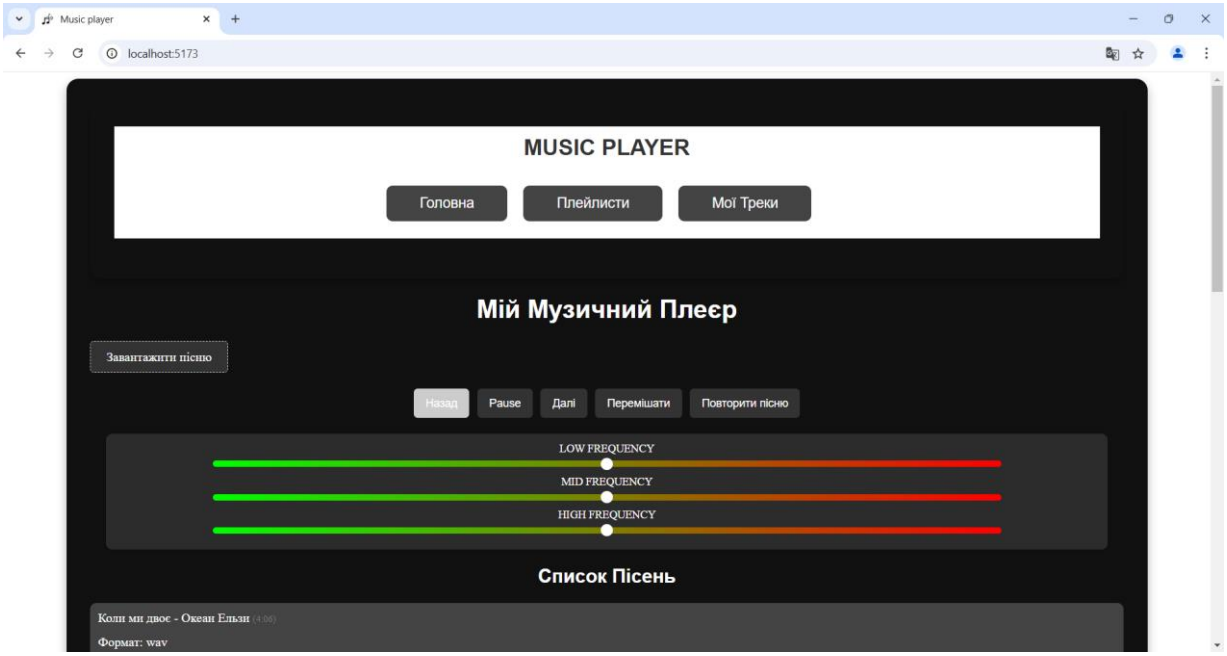


Рисунок 8– Головна сторінка

Перше що ми можемо зробити це завантажити пісню до нашого плеєру. Для цього натискаємо кнопку «Завантажити пісню».

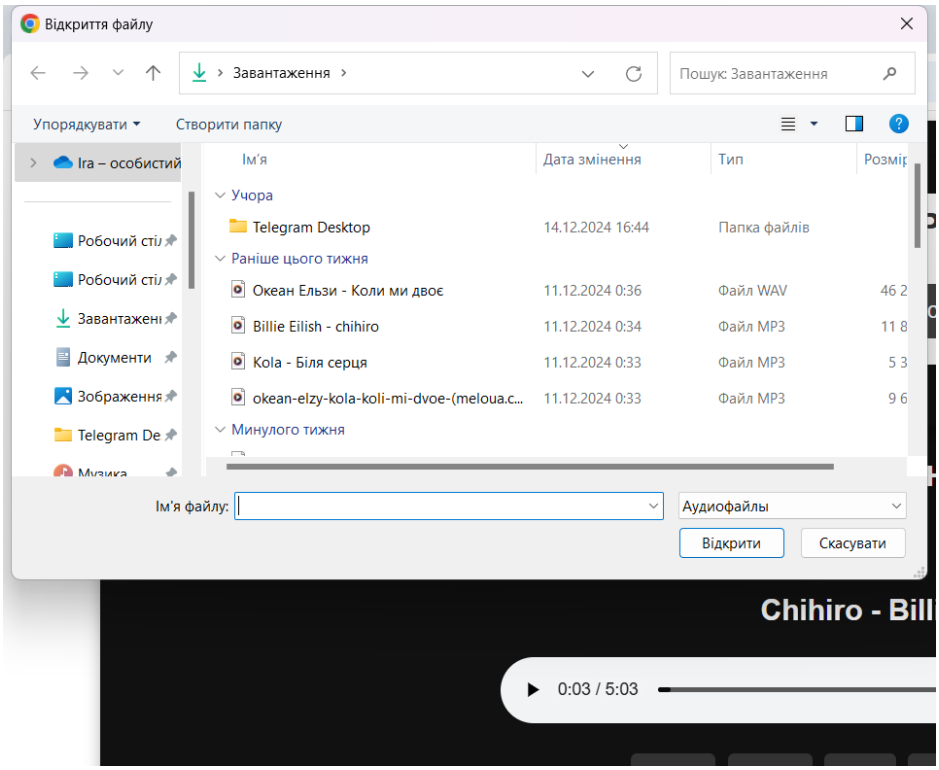


Рисунок 9– Вибір файлу для завантаження

Після натискання на кнопку і вибору пісні для завантаження бачимо форму яка заповнюється автоматично відповідно до того файлу який ми завантажили. Перевіряємо чи дані записані правильно

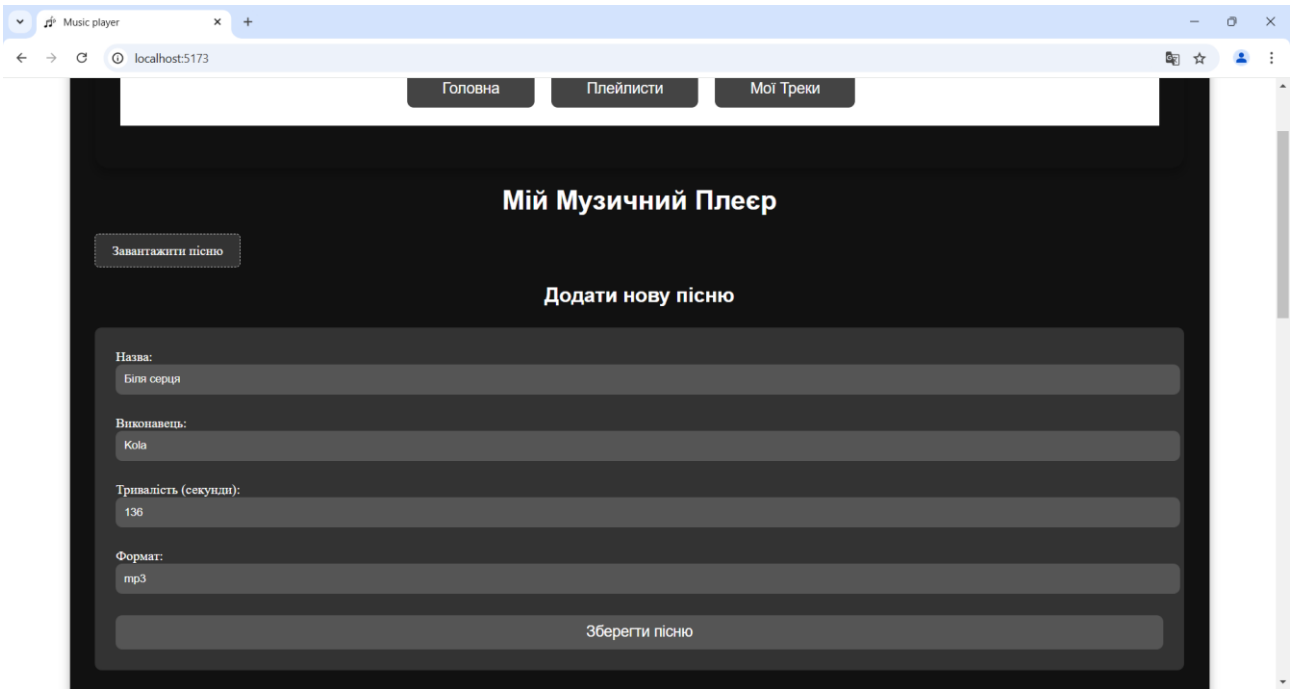


Рисунок 10 – Форма для збереження пісні.

Якщо дані записані правильно натискаємо кнопку «зберегти пісню».

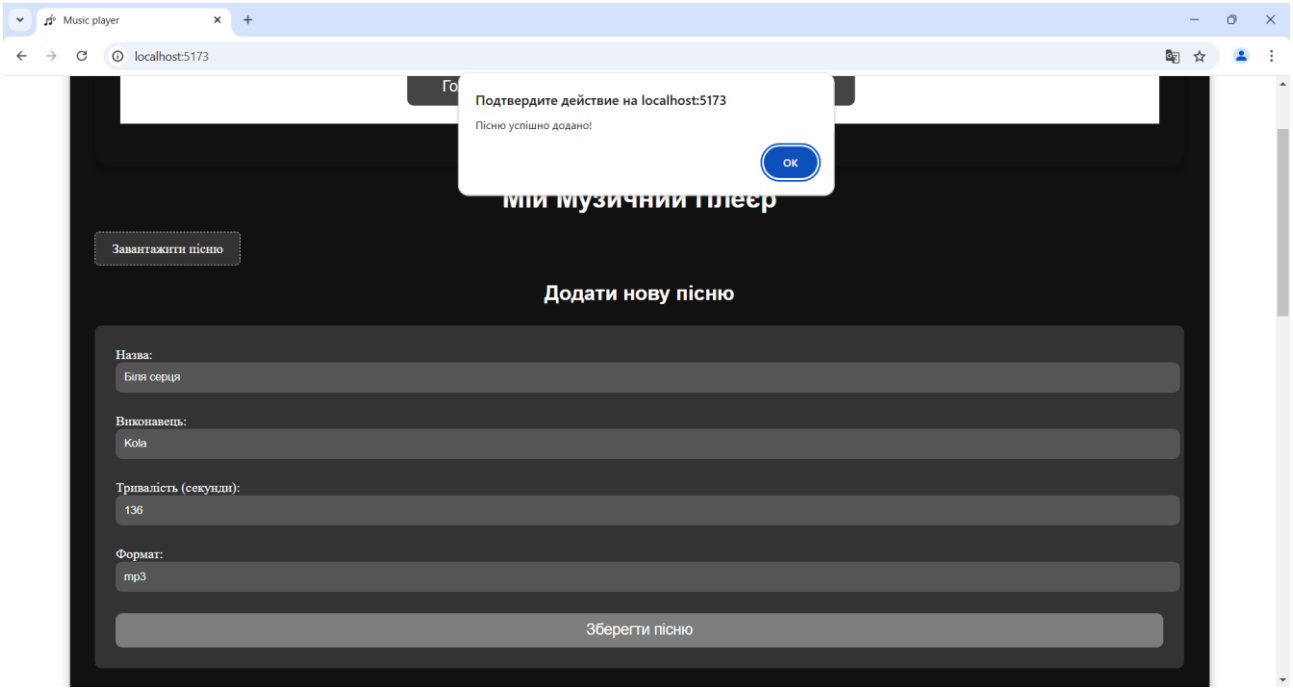


Рисунок 1 – отримання повідомлення про успішне збереження

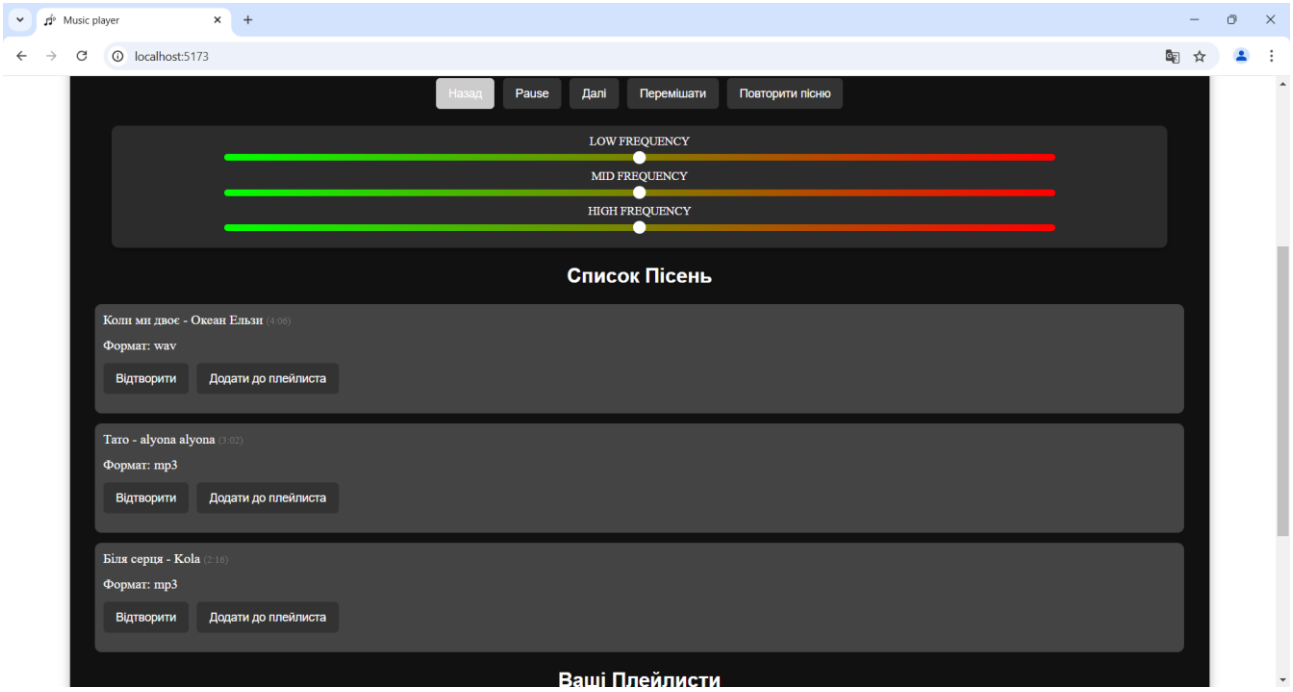


Рисунок 12 – пісня додалась до списку пісень

Також ми можемо створити плейлист. Для цього нам потрібно натиснути кнопку «Створити новий плейлист» та вписати імя плейлиста та після цього натиснути кнопку створити плейлист.

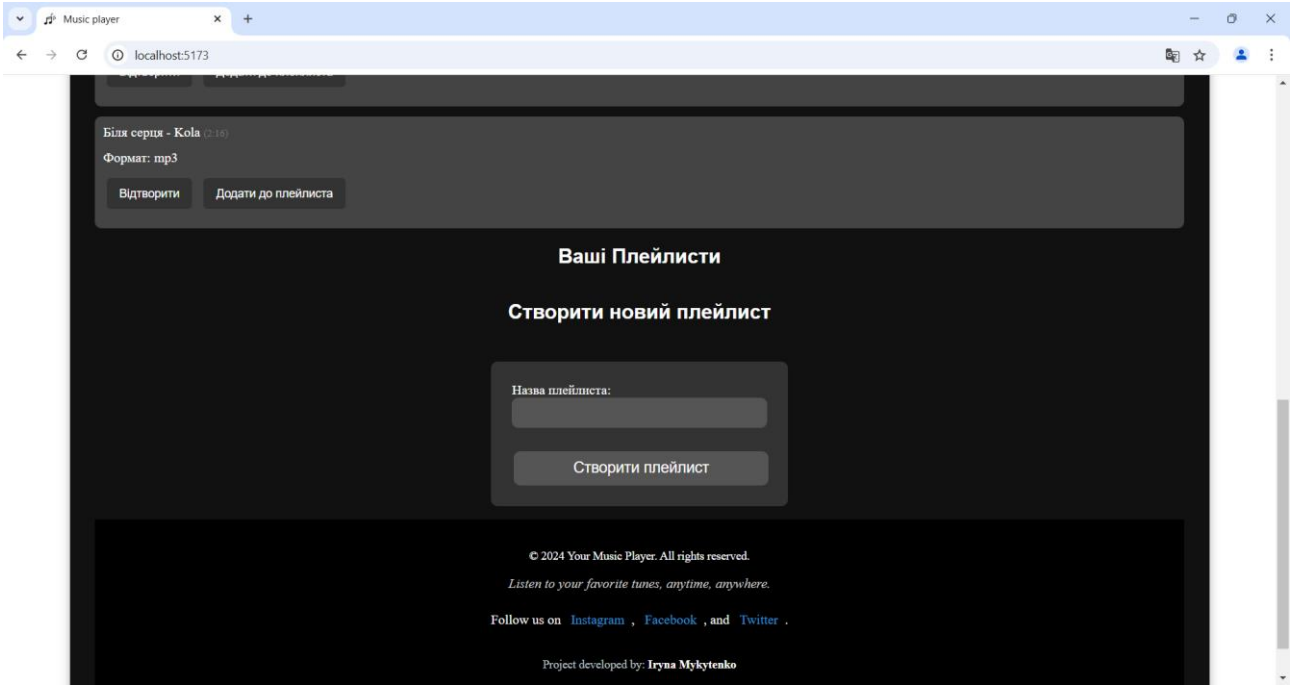


Рисунок 13 – створення плейлиста

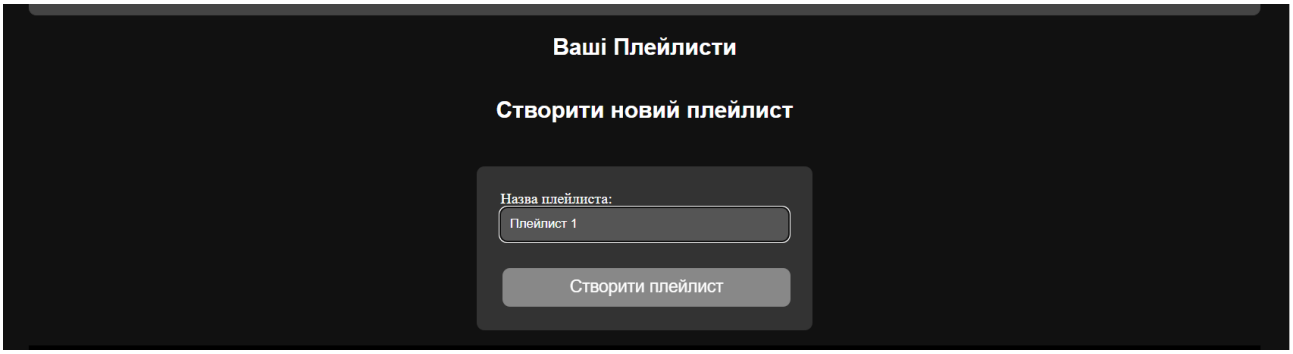


Рисунок 14 – заповнення поля

Отримуємо:

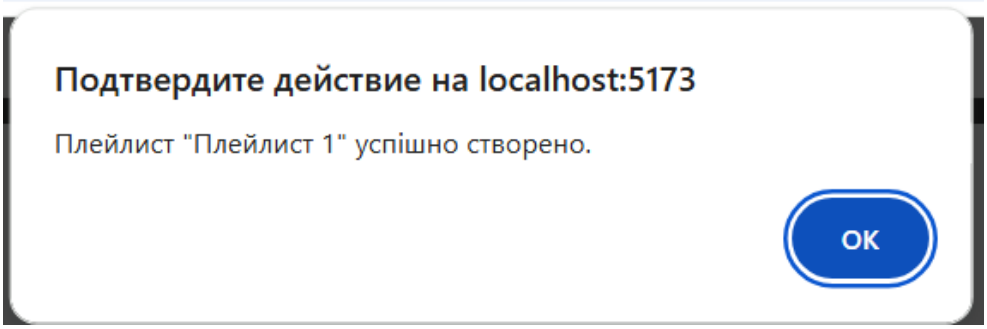


Рисунок 15 – повідомлення про успішне створення

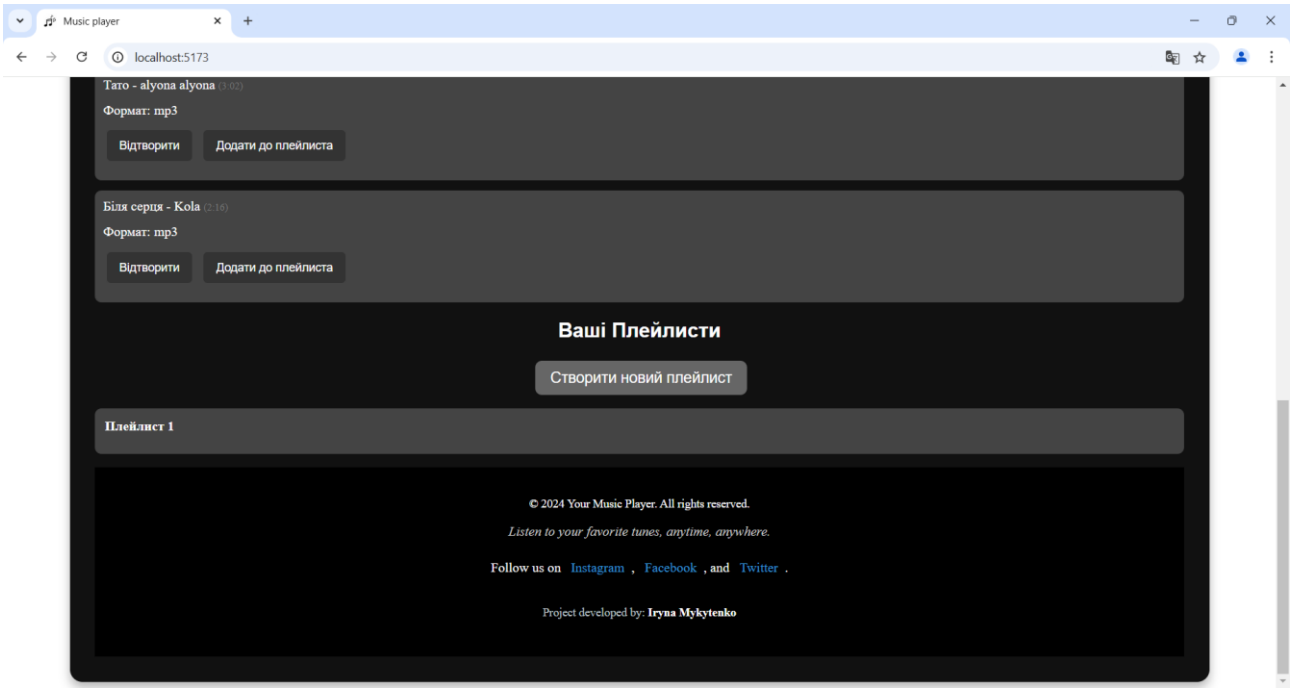


Рисунок 16 – Відображення доданого плейлиста

Натискаємо кнопку відтворити біля пісні:

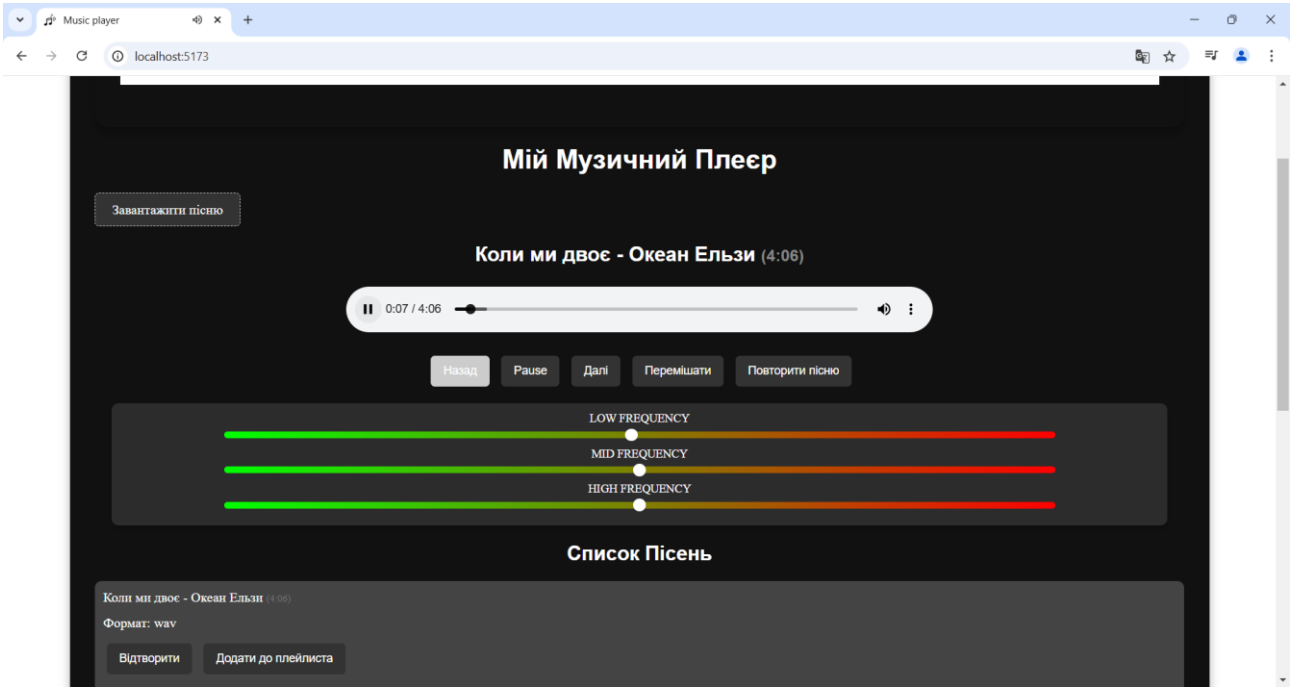


Рисунок 17 – Результат натискання на кнопку-відтворення пісні

Натискаємо на кнопку далі для того аби відтворити наступну пісню:

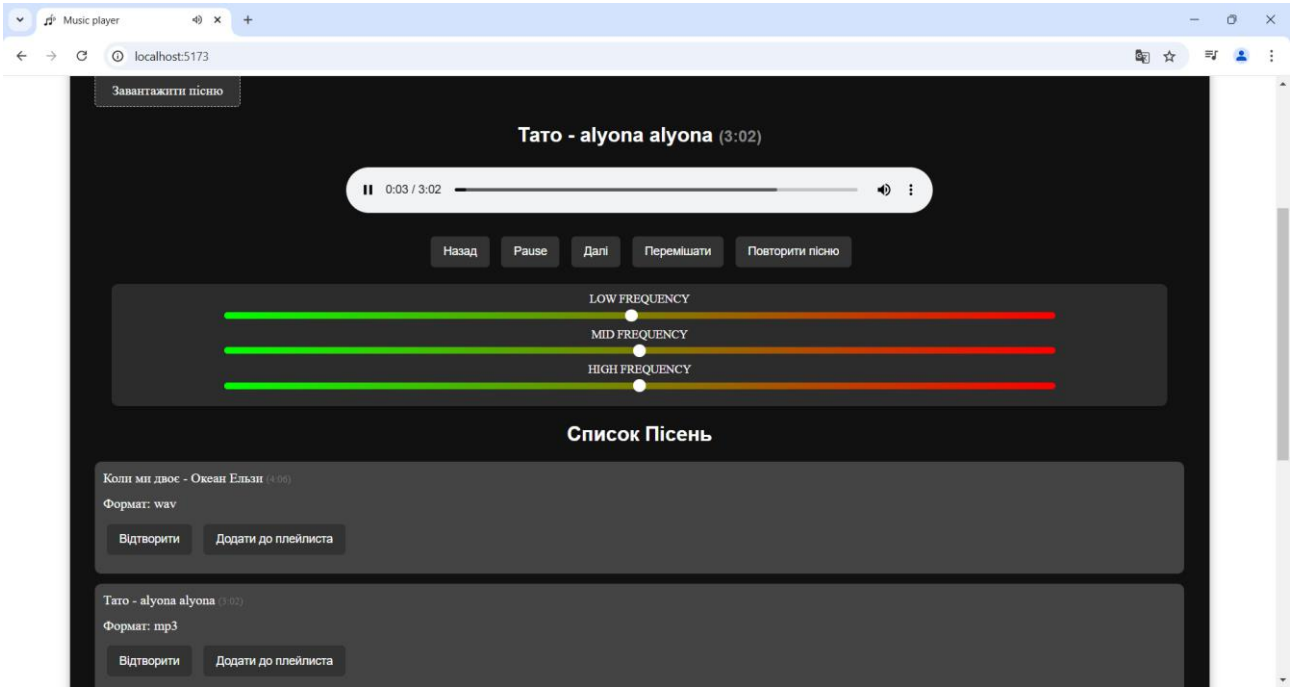


Рисунок 18 – Результат натискання на кнопку далі

Настинемо кнопку «перемішати» для того аби у випадковому порядку відвторити пісні.

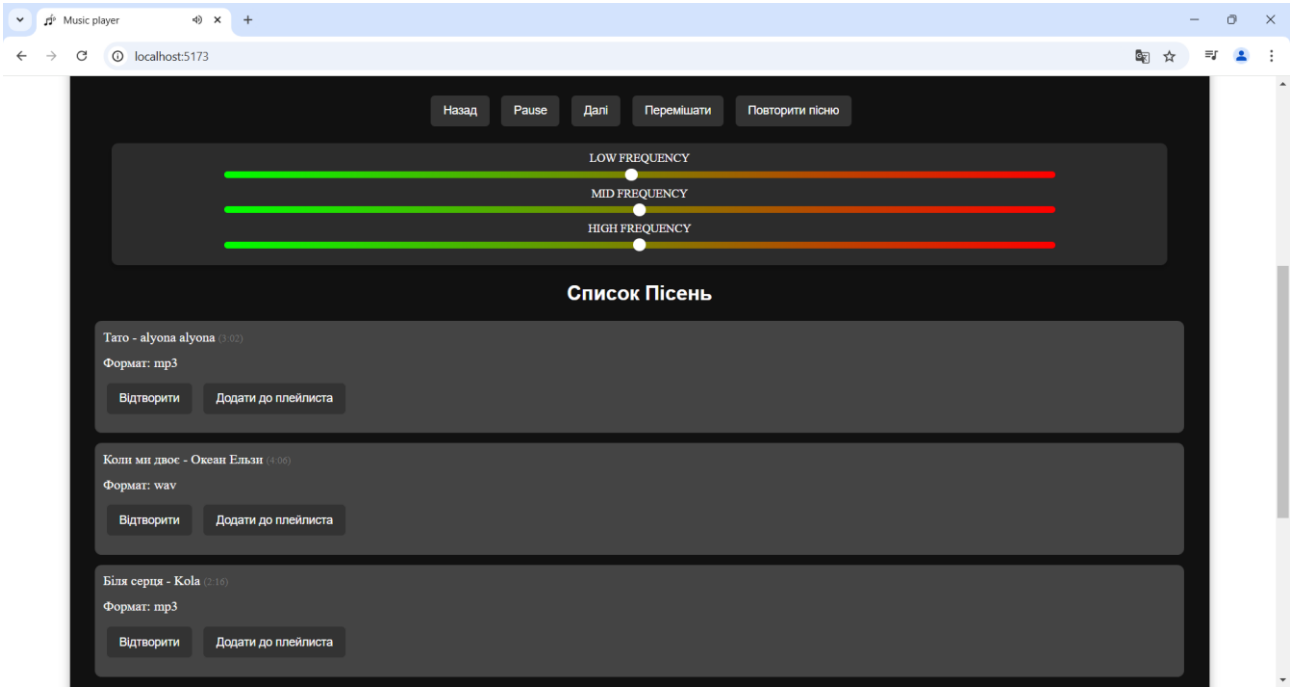


Рисунок 19 – Результат натискання на кнопку перемішати

Пісні в списку перемішались. Відповідно якщо натиснути кнопку далі то відтвориться наступна після яка стоїть після цієї яка відтворюється наступною в списку перемішаних пісень.

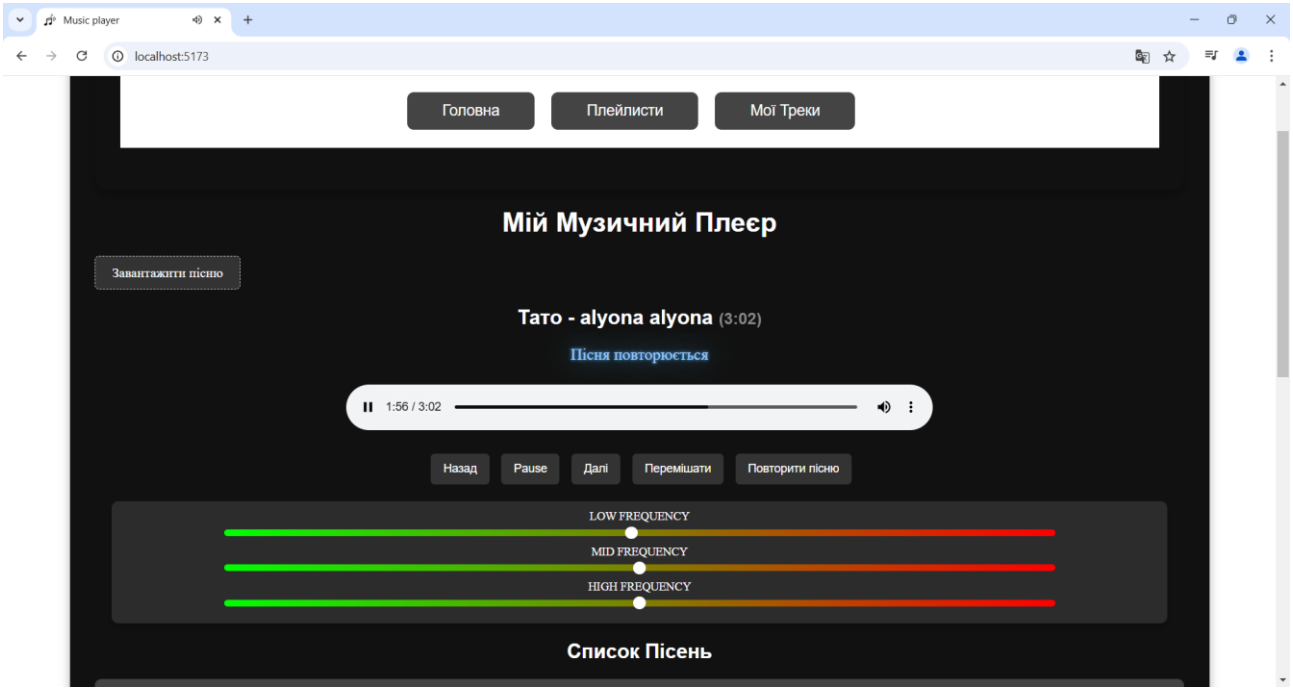


Рисунок 20 – Результат натискання на кнопку повторити пісню

Пісня доходить до кінця і відтворюється повторно. Окрім цього ми бачимо надпис який сповіщає користувачу про повторення пісні. При натисненні кнопки знову, повтор пісні юуде вимкнено.

Редагуємо налаштування еквалайзера(перетягуємо повзунки для різних частот):

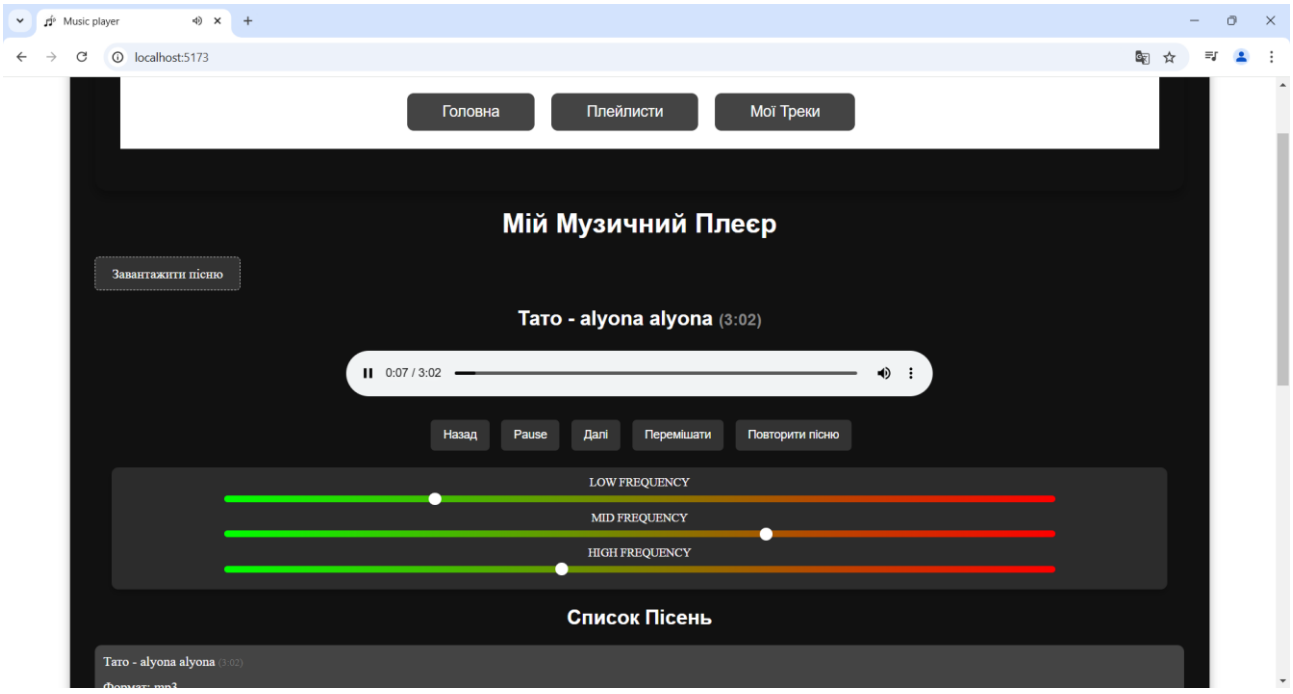


Рисунок 21 – Результат налаштування еквалайзера

Переходимо на вкладку плейлисти

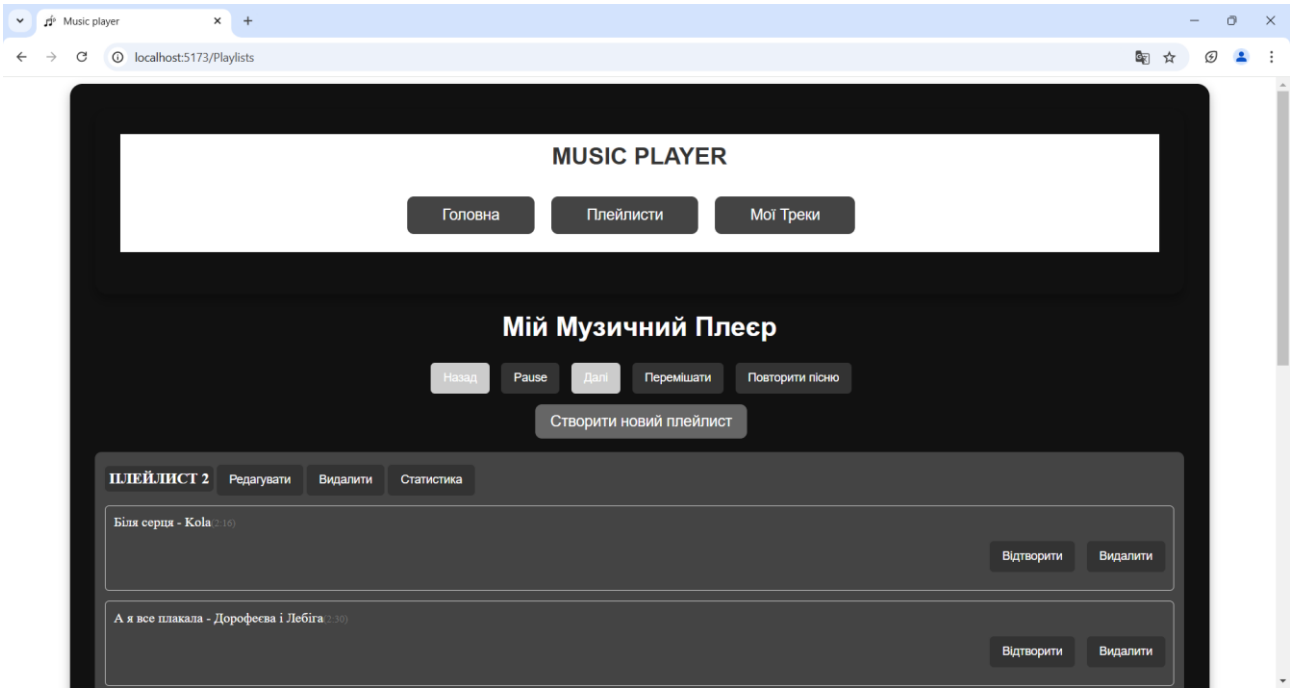


Рисунок 22 – Плейлисти

Ми можемо стоворити за таким же ж сценарієм як раніше новий плейлист, або редагувати існуючий.

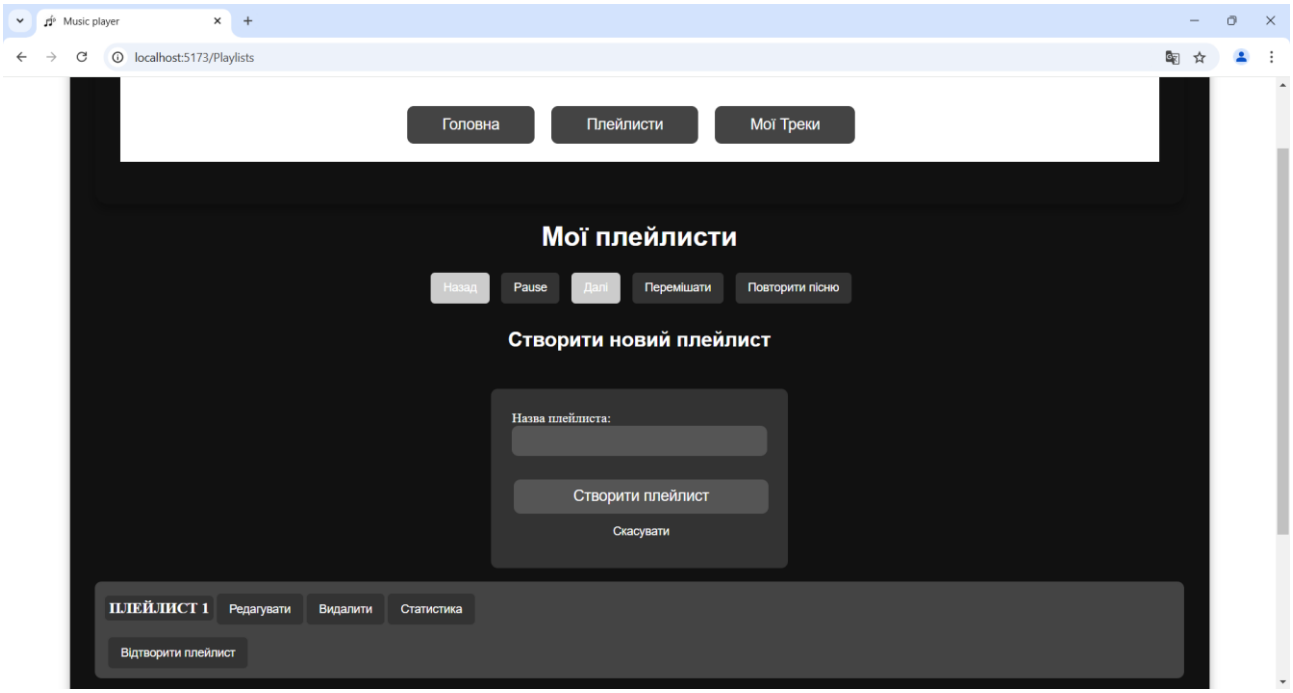


Рисунок 23 –Створення плейлиста

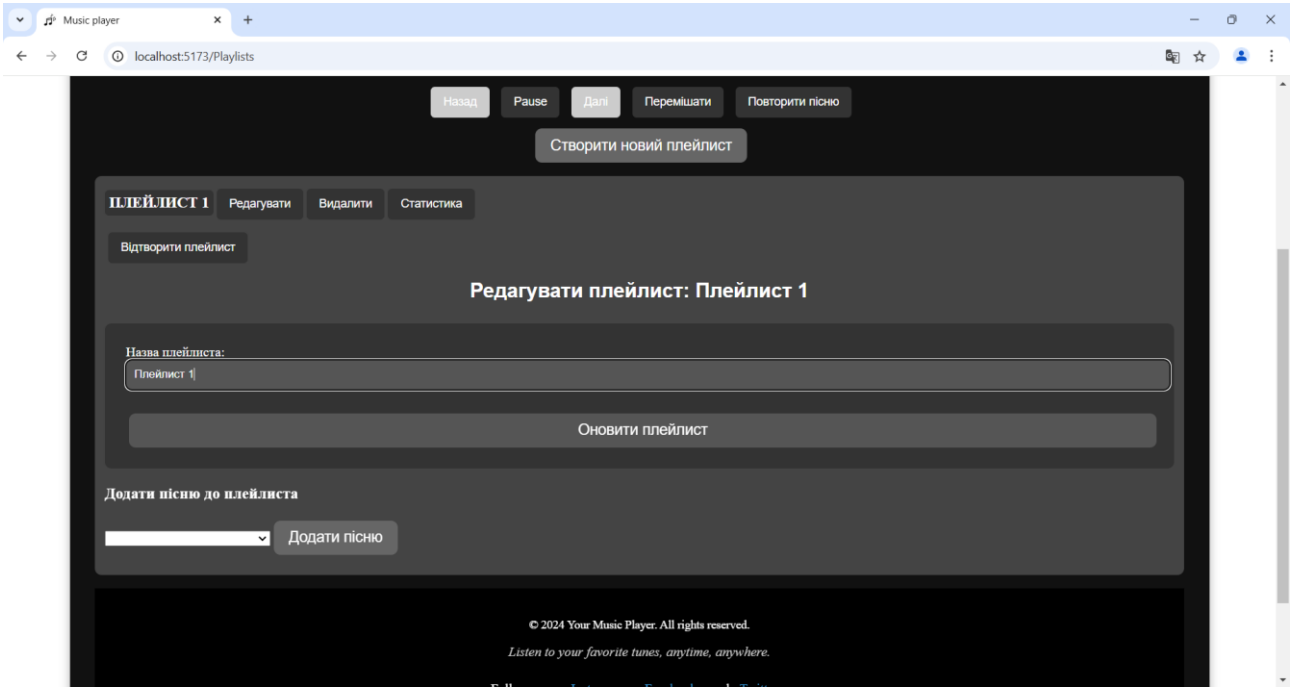


Рисунок 24 –Редагування плейлиста

Відповідно при редагуванні ми можемо додати пісню до плейлиста і змінити назву плейлиста. Змінимо назву плейлиста на «Улюблений плейлист» і додамо туди пісню Тато-альона альона.

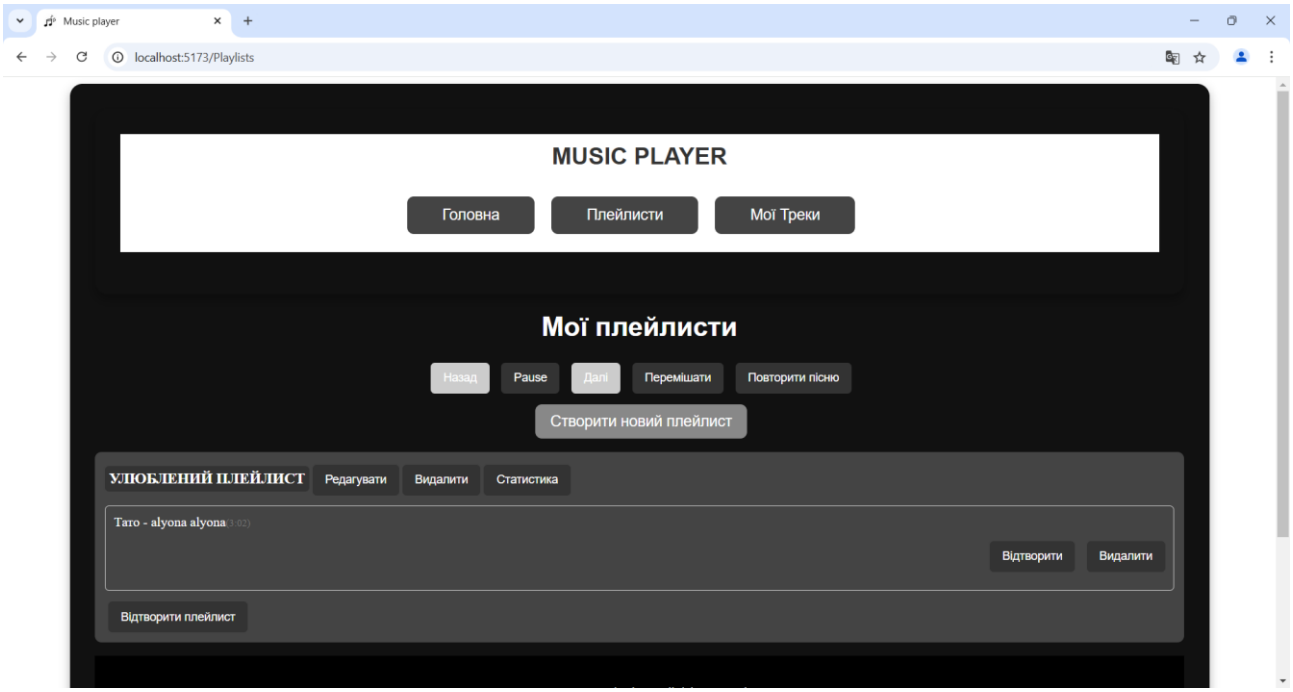


Рисунок 25 –Результат редагування плейлиста

Отримання статистики про плейлист:

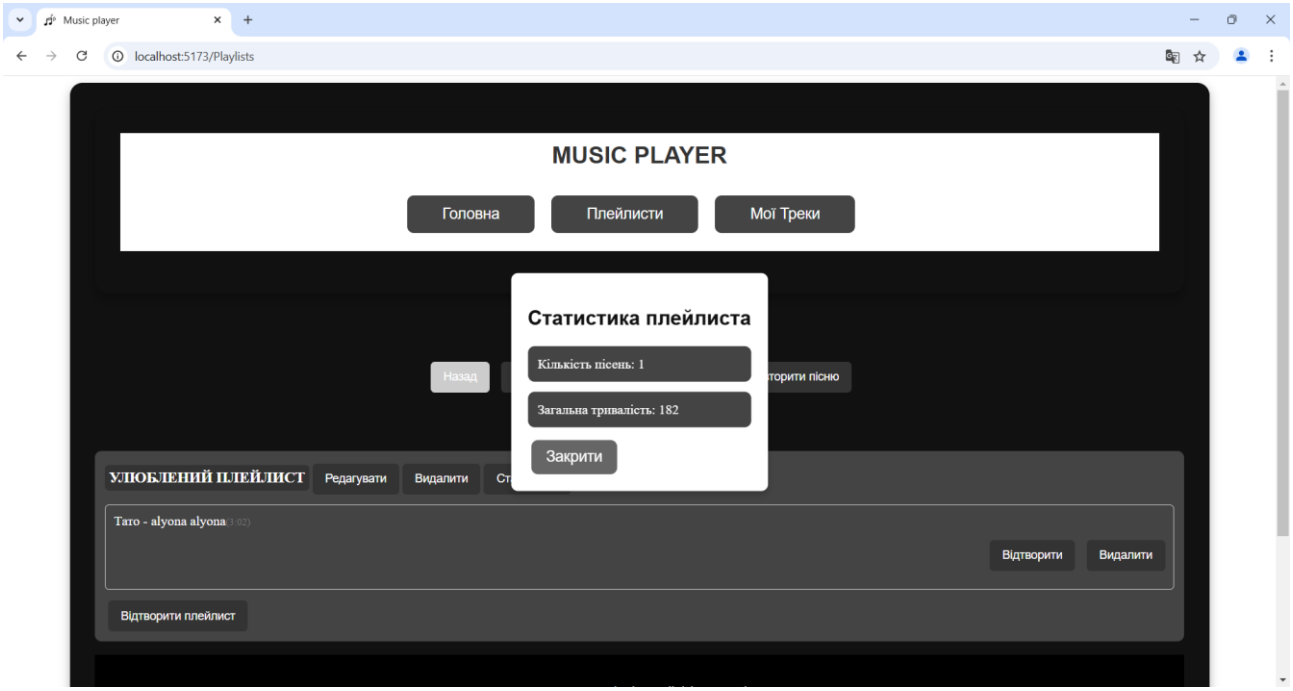


Рисунок 26 –Отримання статистики про плейлист

Натискаємо кнопку видалити і видаляємо плйлист:

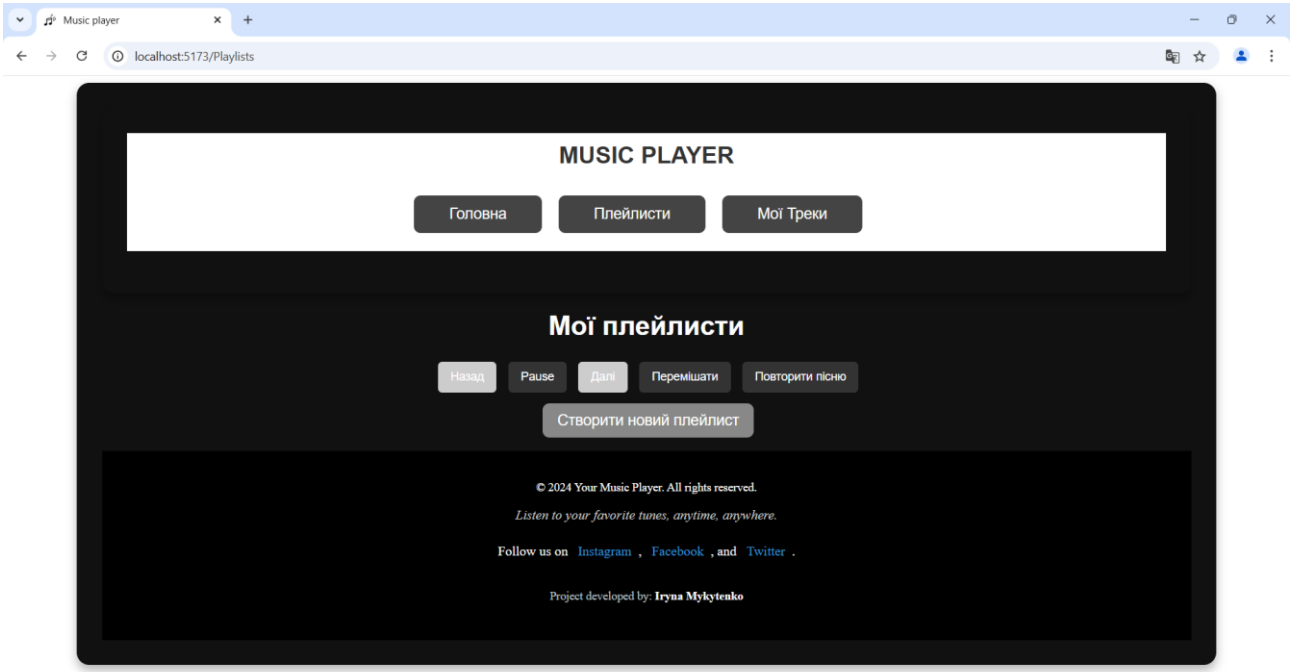


Рисунок 27 –Плейлист видалено

Окрім цього для плейлиста доступний функціонал такий самий як і для кожної пісні. Ми можемо перемішати пісні в плейлисті, відтворити плейлист по черзі(після відтворення останньої пісні почне відтворюватись перша яка є в

списку плейлиста), видалити пісню з плейлиста, натискати кнопки далі\назад для переміщення по плейлисту, та повторювати повторно пісні.

Вигляд сторінки «Мої треки» має вигяд:

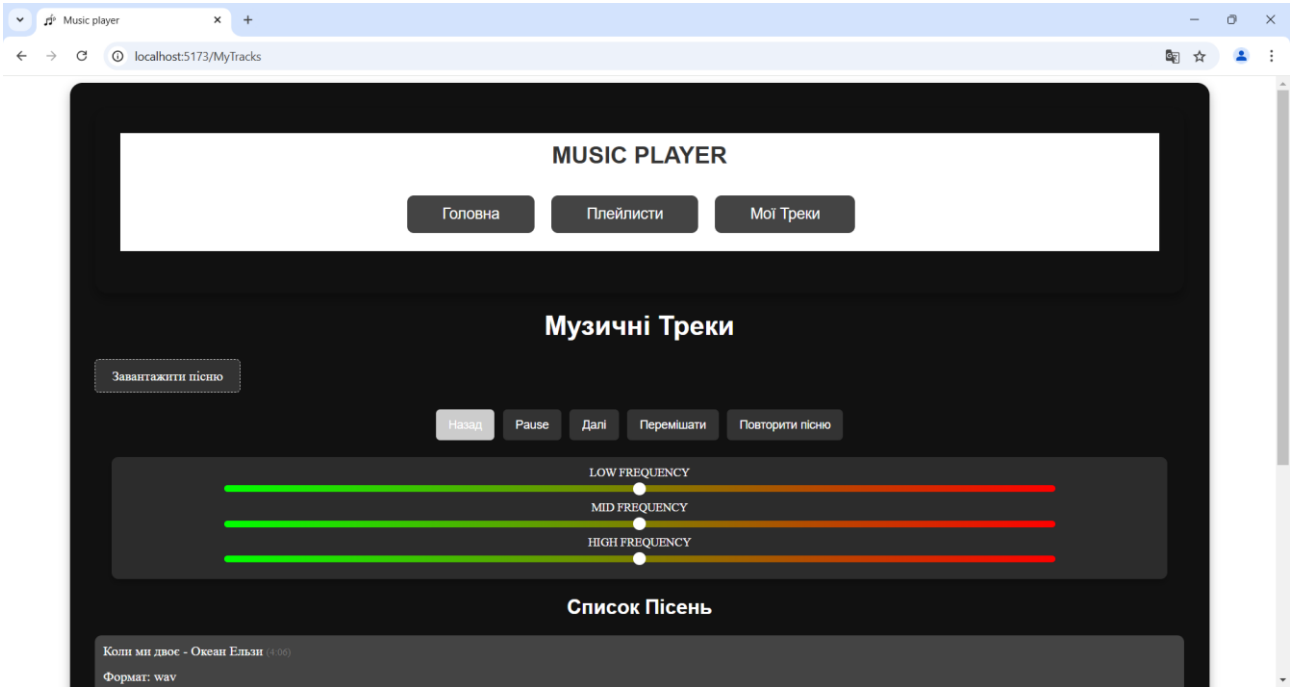


Рисунок 28 –Сторінка «Мої треки»

Має такий функціонал для відтворення пісень як і попередні сторінки, містить кнопку завантаження пісні(після завантаження така ж форма для заповнення як і на головній сторінці), також налаштування еквалайзера, та видалення пісень.

Повернемось на головну сторінку і додамо через неї пісню до плейлиста який створимо попередньо з назвою «Плейлист 1».

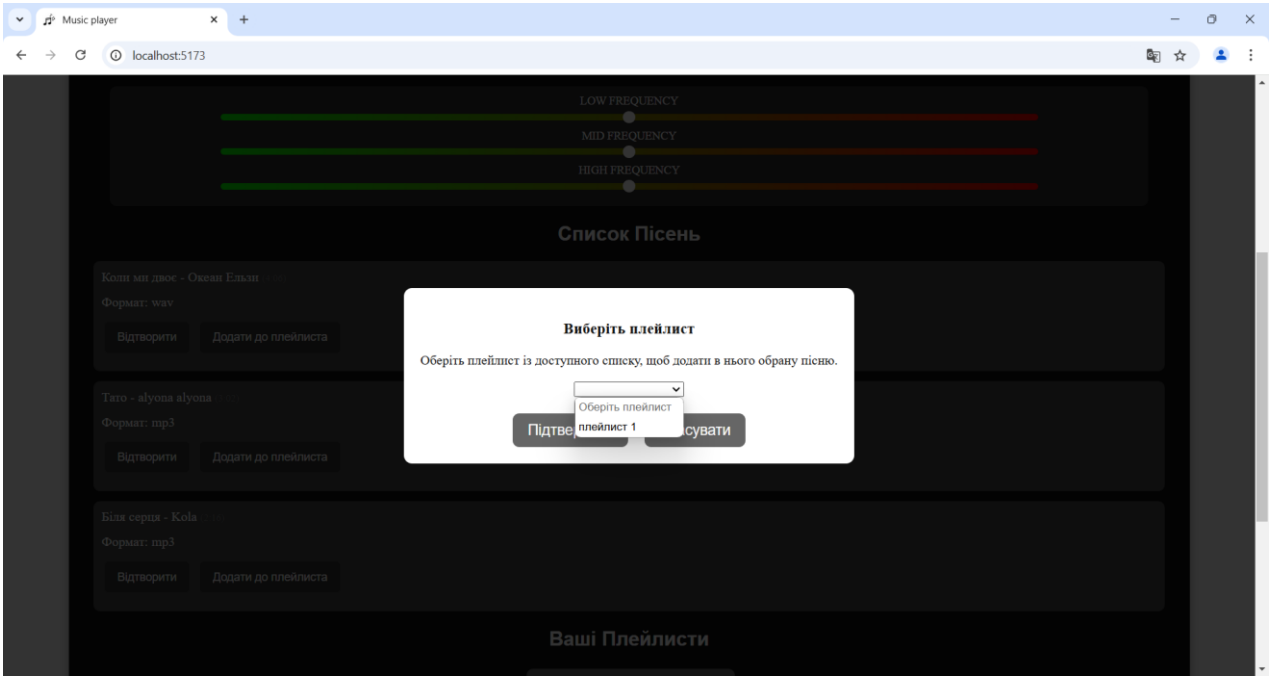


Рисунок 29 – додавання пісні до плейлиста через головну сторінку

При спробі додати пісню яка вже є в плейлисті нам виведе повідомлення про помилку.

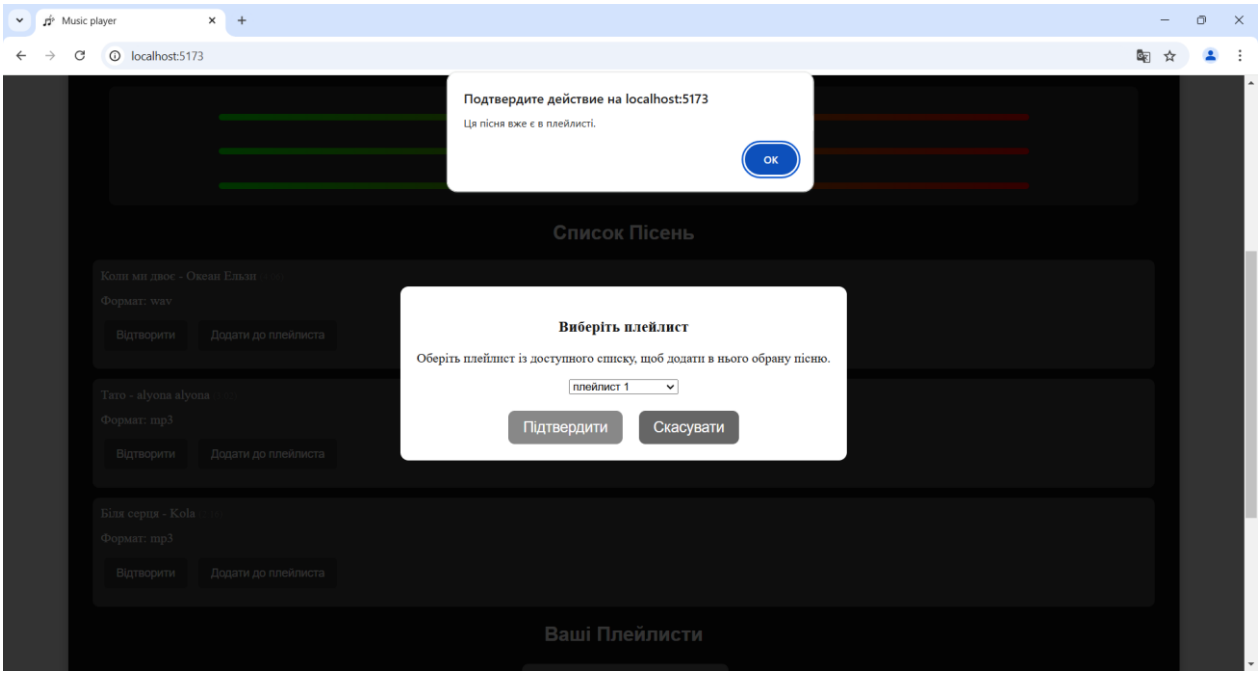


Рисунок 30– помилка при додаванні пісні

В даному випадку на вкладці плейлистів у нас дві пісні в плейлисті

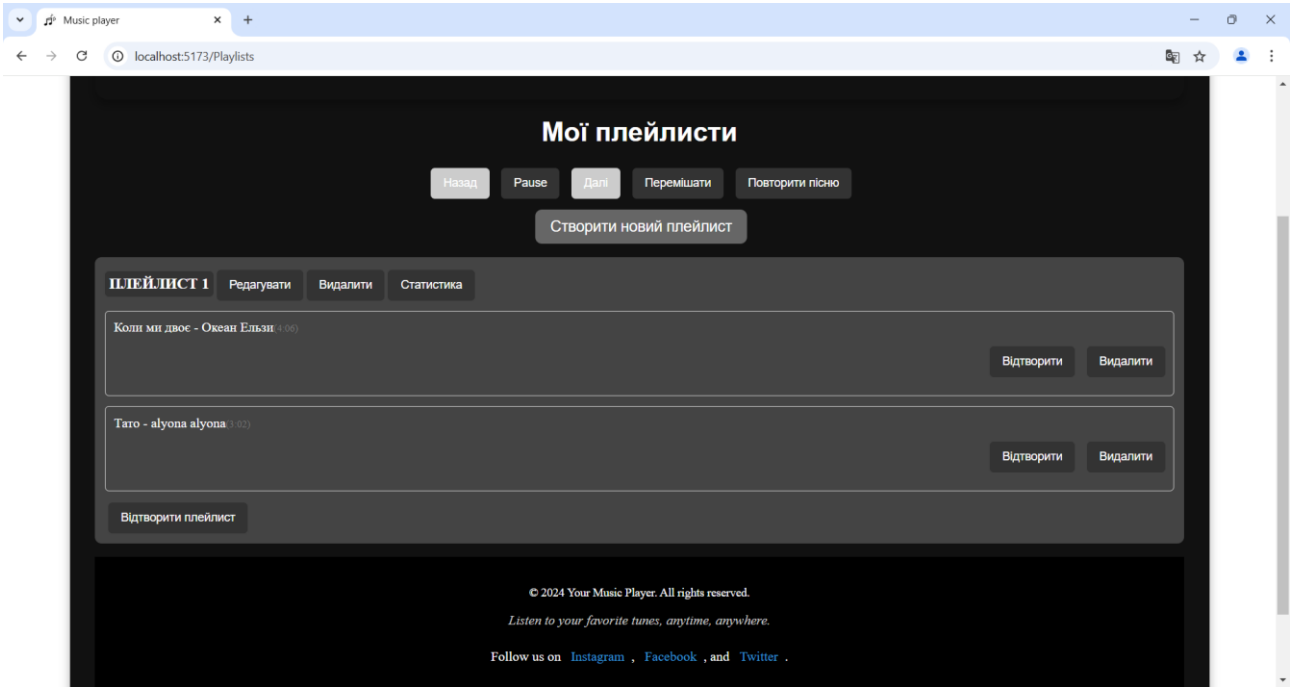


Рисунок 31 – плейлист з піснями

Якщо ми перейдемо на вкладку мої треки і звідти видалимо пісню, то відповідно вона видалиться і з плейлиста теж. Окремо ми можемо видаляти пісні тільки з плейлиста, але не з нашого плеєра.

Результат видалення пісні Тато:

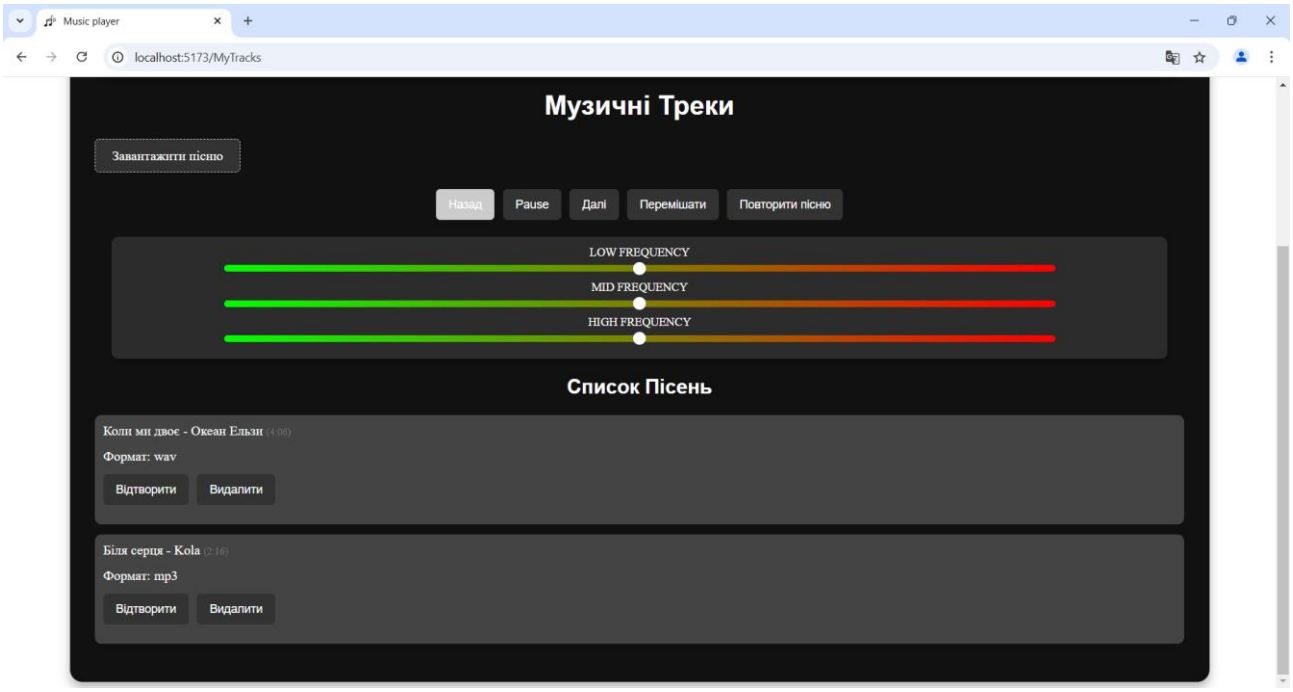


Рисунок 32 – пісня видалена з вкладки «мої треки»

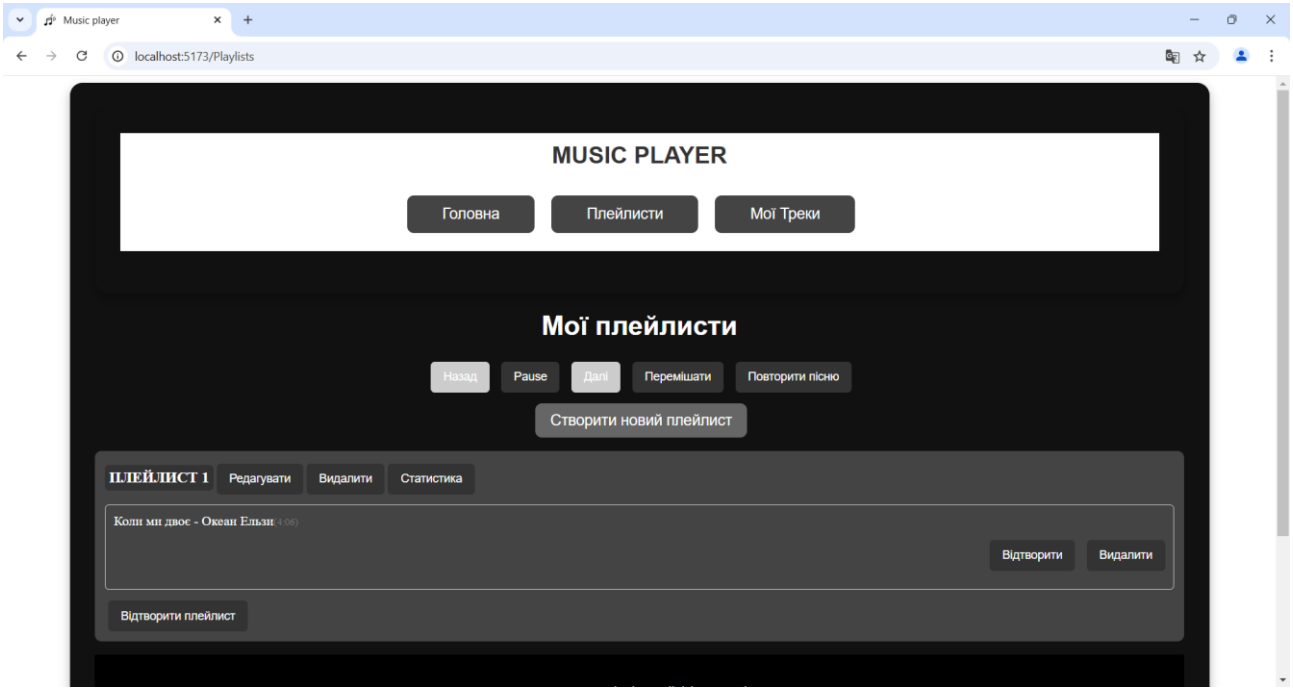


Рисунок 33 – пісня видалена з вкладки «плейлисти»

Висновок: У даній лабораторній роботі я реалізувала взаємодію компонентів програми в клієнт-серверній архітектурі. Взаємодія між фронтендом і бекендом відбувається через REST API, яке дозволяє здійснювати запити та отримувати відповіді у форматі JSON. На бекенді використовуються контролери для обробки запитів, сервіси для виконання бізнес-логіки та репозиторії для роботи з базою даних. Зокрема, за допомогою RESTful API виконуються CRUD-операції для пісень та плейлистів. Фронтенд, реалізований за допомогою Vue.js, здійснює запити до бекенду через бібліотеку axios та оновлює інтерфейс на основі отриманих даних. Користувач може взаємодіяти з програмою, створюючи, редагуючи та видаляючи плейлисти та пісні, а також отримувати список пісень через запити до серверної частини програми.