**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ**
**І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

| | |
|---|---|
| **ПОГОДЖЕНО** | **ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** |
| **Декан факультету** | **Завідувач кафедри комп'ютерних наук** |
| _____ **Ігор БОЛБОТ** | _____ **Белла ГОЛУБ** |
| (підпис) | (підпис) |
| «___» _____ 2026 р. | «___» _____ 2026 р. |

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему:** Програмне забезпечення для веборієнтованої системи продажу музичного контенту

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

**Гарант освітньої програми**
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

**Керівник бакалаврської кваліфікаційної роботи**
к.ф.-м.н., доцент _____ **Віктор КИРИЧЕНКО**
(підпис)

**Виконав** _____ **Антон ШУЛЯК**
(підпис)

**Київ-2026**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ**
**І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри комп'ютерних наук**

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

**ЗАВДАННЯ**
**ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ**
**ЗДОБУВАЧУ**
Шуляку Антону Олександровичу

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення для веборієнтованої системи продажу музичного контенту»

затверджена наказом від «19» грудня 2025 р. №3196 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Під час розробки використати клієнт-серверну архітектуру, сучасні вебтехнології React, REST API, PostgreSQL, JWT-автентифікацію та адаптивний користувацький інтерфейс. Передбачити ролі гостя, зареєстрованого користувача та адміністратора.

Перелік питань, що підлягають дослідженню:
1. Дослідження особливостей онлайн-продажу цифрової музики
2. Побудова UML-діаграм прецедентів, послідовностей та діяльності
3. Поєктування структури бази даних для зберігання треків, користувачів, кошиків та покупок
4. Реалізація модулів пошуку, прослуховування, кошика, покупки та адміністрування музичного контенту

Перелік графічних документів (за необхідності).
Дата видачі завдання «19» грудня 2025 р.

**Керівник бакалаврської кваліфікаційної роботи** _____ (підпис) **Віктор КИРИЧЕНКО**

**Завдання взяв до виконання** _____ (підпис) **Антон ШУЛЯК**

# ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 5

ВСТУП 7

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10

1.1 10

1.2 13

1.3 22

1.4 28

2. 32

2.1. 33

2.2 Вибір системи управління базами даних 36

2.3 Розробка структури інформаційної бази 39

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 45

3.1. 46

3.2. 49

3.3. 50

3.4 Компонентна структура системи 50

3.5 Вибір інструментарію 52

3.6 Алгоритмізація та програмування модулів 54

4. 58

4.1. 58

4.2. 72

4.3 Склад інсталяційного пакета 73

ВИСНОВКИ 80

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 82

ДОДАТКИ 85

Додаток А 85

Додаток Б 86

Додаток В 87

Додаток Д 88

Додаток Е 94

Додаток Ж 95

Додаток І 101

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface - програмний інтерфейс взаємодії між клієнтською та серверною частинами системи

CORS – Cross-Origin Resource Sharing - механізм, що дозволяє або забороняє запити до серверної частини з інших доменів

CRUD – Create, Read, Update, Delete - базові операції роботи з даними в базі

CSS – Cascading Style Sheets - мова таблиць стилів для оформлення інтерфейсу

ER – Entity-Relationship - тип діаграм для відображення структури бази даних

HTML – HyperText Markup Language - мова розмітки для структурування вебсторінок

HTTP – HyperText Transfer Protocol - протокол передачі даних між клієнтом і сервером

JS – JavaScript - мова програмування для розробки клієнтської частини

JSX – JavaScript XML - синтаксичне розширення JavaScript, що використовується у React для опису структури інтерфейсу

JSON – JavaScript Object Notation - формат обміну даними між клієнтською та серверною частинами системи

JWT – JSON Web Token - стандарт токену авторизації, що використовується для автентифікації користувачів

ORM – Object-Relational Mapping - технологія відображення об'єктів Python на таблиці бази даних, реалізована через SQLAlchemy

REST – Representational State Transfer - архітектурний стиль для побудови API взаємодії між клієнтом і сервером

SPA – Single Page Application - тип вебзастосунку, в якому навігація між розділами відбувається без перезавантаження сторінки; реалізований за допомогою React

SQL – Structured Query Language - мова запитів до реляційних баз даних

UI – User Interface - користувацький інтерфейс системи

UML – Unified Modeling Language - уніфікована мова моделювання для побудови діаграм предметної області

UX – User Experience - досвід взаємодії користувача з системою

VPS – Virtual Private Server - віртуальний виділений сервер для розгортання системи

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних (у проєкті використовується PostgreSQL)

# ВСТУП

*Актуальність:* Розвиток електронної комерції та зростання попиту на персоналізовані продукти зумовлюють необхідність створення нових інструментів, які забезпечують зручний та ефективний процес онлайн-покупок. У музичній індустрії, зокрема у сфері продажу музичного контенту, важливого значення набуває можливість швидкого пошуку, перегляду, придбання та отримання музичних файлів без використання фізичних носіїв. Музичний контент є не лише цифровим продуктом, а й важливим елементом сучасної медіа-індустрії, від якого залежить зручність доступу до музики, можливість її легального придбання, а також ефективність взаємодії між продавцем і кінцевими користувачами. Персоналізація музичного контенту дає змогу користувачам знаходити потрібні композиції за жанром, назвою, настроєм або іншими характеристиками, а також формувати власну добірку придбаних треків відповідно до музичних уподобань і потреб.

Із розвитком веб-технологій з’являється можливість створення інтерактивних онлайн-сервісів, які забезпечують користувачеві зручний перегляд каталогу, швидкий пошук потрібних композицій, додавання цифрового продукту до кошика та оформлення покупки. Одним із найефективніших рішень є веб-застосунок, який дає змогу виконувати швидкий пошук і купівлю в режимі реального часу безпосередньо у браузері. Такий підхід дозволяє покупцеві краще орієнтуватися в доступному музичному асортименті, швидко знаходити потрібні треки та створює позитивний користувацький досвід завдяки персоналізованому вибору, а також зручному адаптивному інтерфейсу з можливістю купівлі цифрових продуктів.

Веборієнтоване програмне забезпечення для продажу музичного контенту виступає комплексним інструментом, який забезпечує не лише процес онлайн-придбання цифрового продукту, а й можливість його попереднього

прослуховування та ознайомлення з основними характеристиками треку. Крім того, застосунок оптимізований для мобільних пристроїв, що робить його зручним для широкого кола користувачів. Система також містить адміністративну панель, яка надає можливість додавати, редагувати та керувати музичними композиціями в каталозі.

*Об'єктом дослідження:* процес онлайн-продажу цифрового музичного контенту, зокрема музичних треків, з можливістю попереднього прослуховування, пошуку, вибору та придбання через веборієнтовану систему.

*Предметом дослідження*: програмне забезпечення веборієнтованої системи продажу музичного контенту.

*Метою роботи* є спрощення процесу пошуку, попереднього прослуховування та придбання цифрового контенту через веборієнтовану систему.

Для досягнення поставленої мети спочатку було проведено аналіз наявних рішень на ринку онлайн-продажу цифрового музичного контенту з метою визначення вимог до майбутнього програмного продукту. Наступним етапом став вибір технологічного стеку: React для клієнтської частини, REST API, JSON та Axios для забезпечення взаємодії між клієнтською і серверною частинами системи, а також Figma для прототипування інтерфейсу. Під час розробки було перевірено взаємодію компонентів, адаптивність користувацького інтерфейсу, роботу авторизації та реєстрації користувачів, поведінку каталогу треків, пошуку, кошика, попереднього прослуховування музичного контенту, адміністративної панелі для додавання та редагування треків, а також імітацію оплати через сторінку Dashboard.

*Наукова новизна роботи:* полягає в реалізації веб-застосунку для продажу цифрового музичного контенту з функціями попереднього прослуховування, пошуку треків, перегляду характеристик композицій, формування кошика, оформлення покупки та керування музичними треками через адміністративну панель. Запропоноване рішення поєднує інструменти електронної комерції,

клієнт-серверну архітектуру, адаптивний користувацький інтерфейс і механізми взаємодії користувача з цифровим продуктом в одному застосунку.

*Практична значущість:* полягає у створенні інструменту, який може бути використаний в електронній комерції для продажу цифрового музичного контенту, автоматизації процесів пошуку, вибору, попереднього прослуховування та купівлі, а також для покращення користувацького досвіду. Реалізований застосунок також може бути адаптований до інших видів цифрового контенту, які потребують попереднього перегляду або прослуховування та подальшого придбання.

# 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Веборієнтована система продажу музичного контенту

Програмне забезпечення веб-системи продажу музичного контенту - це онлайн-платформа, що дозволяє користувачам фільтрувати, прослуховувати та купувати цифрові музичні продукти відповідно до своїх потреб за жанром, настроєм, темпом, тональністю та іншими параметрами. Система відповідає потребам сучасних споживачів, які прагнуть швидко, зручно та легально отримати доступ до високоякісного музичного контенту, і надає функції попереднього прослуховування треків та вибору композицій за власним смаком.

Основна мета системи — автоматизувати процеси вибору, управління та продажу музичного контенту, а також підвищити рівень задоволеності користувачів, надаючи їм можливість через зручний інтерфейс швидко здійснювати пошук, фільтрувати за ключовими характеристиками, попередньо прослуховувати треки, додавати обрані композиції до кошика та купувати цифрові продукти. Крім того, система передбачає адміністративну панель для додавання, редагування та управління музичними треками в каталозі.

Існуючі способи придбання музичного контенту через онлайн-магазини або окремі платформи часто мають обмежені можливості пошуку, фільтрації та попереднього прослуховування. Користувачі не завжди мають зручний спосіб швидко обрати композиції за жанром, настроєм, темпом, тональністю або іншими характеристиками перед покупкою. Професійні музиканти, відеоконтент-мейкери, подкастери та інші користувачі потребують точного підбору музичного матеріалу для творчих або комерційних цілей, тоді як звичайні слухачі хочуть легко знаходити, прослуховувати та купувати треки відповідно до власних музичних уподобань.

Аналіз сфери електронної комерції свідчить про те, що зручність пошуку, простота покупки та можливість попереднього прослуховування цифрових продуктів суттєво впливають на рівень задоволеності та довіри користувачів до

онлайн-платформ. У сфері продажу музичного контенту це також актуально: користувачі повинні мати змогу не лише бачити назву треку, жанр і ціну, а й попередньо прослухати композицію, оцінити її звучання та прийняти обґрунтоване рішення про покупку.

У сфері цифрової музики важливою є можливість швидкого доступу до потрібного контенту. Користувачі можуть здійснювати пошук треків за жанром, настроєм, темпом, тональністю або назвою, що дозволяє обирати музичні продукти відповідно до власних уподобань або конкретних вимог. Такий підхід підвищує зручність використання системи та робить процес покупки більш усвідомленим.

Основними елементами веб-застосунку для продажу музичного контенту є такі:

- каталог музичних композицій, який відображає назву, обкладинку, ціну та основні характеристики;
- система пошуку та фільтрації для швидкого знаходження потрібних композицій;
- аудіо-плеєр для прослуховування бажаних треків;
- кошик для додавання вибраних музичних продуктів;
- система авторизації та реєстрації користувачів;
- панель керування для додавання, редагування та управління музичним контентом;
- адаптивний інтерфейс, оптимізований для використання на різних пристроях;
- особистий кабінет користувача з доступом до придбаних треків; здійснення покупки через сторінку панелі керування в тестовому режимі.

Переваги:

- можливість попереднього прослуховування треку в максимальній якості;
- швидкий пошук музичних композицій за характеристиками продукту;

- зручність використання завдяки інтуїтивно зрозумілому інтерфейсу, доступному з будь-якого пристрою;
- автоматизація процесу додавання треків до кошика та здійснення покупки;
- наявність списку бажань для збереження композицій, які цікавлять користувача;
- панель керування, яка дозволяє продавцю легко налаштовувати каталог без спеціальних технічних навичок;
- наявність широких можливостей для розширення функціональності системи;
- доступ користувача до придбаних файлів через особистий кабінет.

Недоліки:

- необхідність стабільного інтернет-з'єднання;
- необхідність захисту музичних даних від несанкціонованого доступу;
- необхідність надійної організації зберігання аудіофайлів;
- необхідність сумісності з різними пристроями та браузерами;
- необхідність подальшої інтеграції справжньої платіжної системи.

Аналіз показує, що вебсистема для продажу музичного контенту є сучасним рішенням, адаптованим до ринку цифрових продуктів. Додаток дозволяє автоматизувати пошук, вибір, ознайомлення, купівлю та керування музичними треками. Впровадження такаї системи сприяє покращенню взаємодії користувачів із цифровим контентом та закладає основу для подальшого розвитку платформи з продажу музичної продукції.

У результаті аналізу стає очевидним, що веб-орієнтована система для продажу музичного контенту є сучасним рішенням, яке відповідає ринку цифрових продуктів. Застосунок дозволяє автоматизувати процес пошуку, вибору, повного ознайомлення, придбання та адміністрування музичних треків. Реалізація такої системи сприяє покращення взаємодії користувачем з цифровим

контентов та створює основу для подальшого розвиту платформи продажу музичних продуктів.

## 1.2Моделювання предметної області

Для створення ефективного програмного забезпечення необхідно не лише розуміти логіку функціонування предметної області, а й уміти вибудовувати структуру інформації у вигляді моделей. Моделювання предметної області дозволяє описати ключові об'єкти, процеси, зв'язки та сценарії використання. Це забезпечує глибший аналіз вимог користувачів, дозволяє визначити вимоги до системи, а також спрощує подальше проектування та реалізацію.

У випадку веб-системи для продажу музичного контенту моделювання дозволяє визначити ключові елементи взаємодії між користувачами, відвідувачами, адміністраторами та самою системою. Надзвичайно важливо візуалізувати послідовність дій, що виникають під час використання платформи: перегляд головної сторінки, пошук та фільтрація треків, перегляд сторінки конкретної композиції, прослуховування демоверсії, додавання треків до кошика або списку бажань, оформлення замовлення, оплата та отримання доступу до придбаних музичних файлів.Управління музичним контентом займає окреме місце у цій предметній області.

Адміністратору доступні функції керування треками (додавання нових композицій, редагування інформації про треки, завантаження файлів, видалення застарілих записів), управління користувачами та перегляд статистики продажів. Це дозволяє підтримувати актуальність каталогу, а також ефективно керувати системою та забезпечувати її підтримку.

Для опису цієї області доцільно використовувати **UML (уніфіковану мову моделювання)** — стандартизовану мову моделювання, яка дозволяє наочно представити функціональні та логічні аспекти системи у вигляді об'єктно-орієнтованих моделей. У цьому розділі розглядаються основні типи діаграм UML, такі як: діаграма прецедентів (Use Case), діаграма послідовності (Sequence) та діаграма діяльності (Activity). Ці UML-діаграми дозволяють

візуально відобразити взаємодію користувачів у веб-системі продажу музичного контенту.

Діаграма прецедентів (Use Case diagram) є однією з ключових діаграм у цій області. Вона створена для відображення основних учасників взаємодії із системою, а також функцій, доступних кожному з акторів. Діаграма прецедентів допомагає визначити межі майбутнього програмного забезпечення, встановити ролі користувачів та описати сценарії використання.

У контексті веб-системи для продажу музичного контенту діаграма прецедентів візуалізує взаємодію між користувачами та основні процеси, пов'язані з переглядом каталогу треків, пошуком музичного контенту, прослуховуванням демоверсій, додаванням композицій до кошика, здійсненням покупки, переглядом придбаних треків та адмініструванням системи. Доступні функції моделюються для незареєстрованого відвідувача, зареєстрованого користувача та адміністратора. Це важливо, оскільки для подальшого проектування структури ПЗ різні категорії користувачів мають різні права доступу та виконують у системі різні дії.

Основними елементами діаграми прецедентів є актори (actors) та прецеденти (use cases). Актори — це зовнішні сутності, які взаємодіють із системою. Прецеденти — це дії або сценарії, які актори можуть виконувати для досягнення конкретного результату. Тобто в межах цієї доменної області акторами є гість, користувач та адміністратор, а прецеденти включають такі дії/сценарії, як перегляд каталогу, пошук треків, покупка або управління композиціями.

На *рис. 1* зображено діаграму прецедентів програмного забезпечення для веб-системи продажу музичного контенту. Вона демонструє основні ролі користувачів системи, доступні функції та зв'язки між окремими прецедентами.


*Рис. 1 Діаграма прецедентів ПЗ для веб-орієнтованої системи продажу музичного контенту*

**Опис акторів**

Визначено 3 основні типи зацікавлених сторін, які взаємодіють із системою:

**Гість** – це незареєстрований у системі відвідувач, який має доступ до базових функцій перегляду. Гість може переглядати головну сторінку та каталог треків, використовувати пошук і фільтри, прослуховувати демо-версії цифрових продуктів, а також реєструватися або проходити автентифікацію для отримання доступу до розширених функцій.

**Користувач** – це зареєстрований у системі учасник, який має доступ до всіх функцій, пов'язаних із купівлею музичного контенту. Користувач може переглядати каталог, сторінки окремих треків, прослуховувати демо-версії,

додавати треки до кошика, здійснювати покупки та оплату, переглядати особистий кабінет і завантажувати придбані треки.

**Адміністратор** – це користувач із розширеними правами доступу, який відповідає за керування каталогом системи. Адміністратор може входити до панелі керування (адмін-панелі), додавати та видаляти треки, редагувати інформацію, завантажувати файли, керувати доступом користувачів і переглядати статистику.

**Опис прецедентів**

***Гість:***

Перегляд головної сторінки – ознайомлення з головною сторінкою вебсистеми, виконавцями, загальною метою та доступними розділами.

Перегляд каталогу – відображення списку музичних композицій, зареєстрованих у системі.

Фільтрація – вибір музичних композицій за такими параметрами, як жанр, темп, настрій тощо.

Пошук музичних композицій – пошук музичного контенту за назвою.

Прослуховування демо-версії – відтворення демо-версії для повноцінного ознайомлення з треком.

Перегляд сторінки треку – відображення детальної інформації про конкретну музичну композицію.

Автентифікація – вхід до системи з використанням облікових даних для доступу до розширених функцій.

Реєстрація – створення облікового запису.

***Користувач:***

Додавання в кошик – додавання обраного продукту для подальшого оформлення покупки.

Оплата – процес отримання обраного музичного контенту, саме цей процес є складовою процесу оформлення покупки.

Особистий кабінет – перегляд, редагування персональної інформації та доступ до придбаного треку.

Перегляд придбаних треків – історія замовлень куплених треків.

Завантаження придбаного треку – отримання повної музичної версії файлу після успішного придбання.

***Адміністратор:***

Вхід до адмінпанелі – авторизація для отримання адміністративних прав.

Керування треками – основний адміністративний сценарій завдяки якому можна виконувати дії над треками, користувачами та перегляду статистики.

Додавання треку – створення нового запису цифрового продукту в системі.

Редагування треку – можливість змінити інформацію про раніше доданий трек у системі.

Видалення треку – можливість вилучити раніше доданий трек із системи.

Завантаження треку – можливість додавання аудіо-файлів, обкладинок та інших матеріалів пов’язаних з треком.

Керування користувачами – зміна стану облікового запису користувача (видалення, блокування, відновлення прав).

Перегляд статистики продажів – аналіз інформації покупки, популярності.

Діаграма прецедентів демонструє загальну структуру взаємодії різних типів акторів із системою. Вона дозволяє визначити основні функціональні можливості кожного з акторів та розмежувати їх права. Така модель є важливим етапом аналізом та реалізації предметної області тому що, дає змогу узгодити функціональні вимоги до системи перед реалізацією.

Діаграма послідовності – це тип UML-діаграми, яка ілюструє динамічну поведінку програмного забезпечення та відображає порядок взаємодії між акторами, об'єктами та компонентами в системі. Використовуючи цю діаграму, можна зрозуміти, які дії виконують учасники процесу, у якому порядку ці дії відбуваються та які повідомлення обмінюються між елементами системи.

Діаграми послідовності є ключовими в аналізі логіки програмного забезпечення, оскільки вони ілюструють як загальну функціональність системи, так і порядок, у якому вони виконуються. Ця діаграма дозволяє побачити, як

користувачі взаємодіють з інтерфейсами, як система обробляє запити, як вона отримує доступ до баз даних та як формуються результати певних дій.

Основні елементи діаграми послідовності включають актори, об'єкти, повідомлення, лінії життєвого циклу та тригерні панелі. Актори – це учасники, які виконують певні дії. Об'єкти або компоненти системи обробляють запити та беруть участь у виконанні сценаріїв. Лінії життєвого циклу показують часове існування акторів, а повідомлення показують взаємодію між акторами. Панелі активації вказують на тривалість, протягом якої компоненти виконують дії або обробляють запити.

***Додаток А*** містить діаграми послідовності, що ілюструють основні сценарії розробленої системи. Діаграми, що включають «користувачів», ілюструють взаємодію із системою, таку як автентифікація, перегляд каталогу, вибір пісень, додавання до кошика, оплата та доступ до придбаних продуктів. Діаграми, що включають «адміністраторів», ілюструють сценарій, у якому після успішної автентифікації користувач може додавати пісні до системного каталогу.

**Опис акторів та об’єктів діаграм**

Користувач – учасник, зареєстрований у системі, який взаємодіє з інтерфейсом для пошуку, перегляду, прослуховування та купівлі музики. Він ініціює вхід, переходить до бібліотеки, вибирає пісню, додає її до кошика, оплачує та здійснює покупку на свій розсуд.

Адміністратор – користувач з розширеними правами, який керує музикою. У наведеному прикладі конфігурації адміністратор має повноваження та додає нову пісню з відповідними властивостями до конфігурації.

Інтерфейс – це клієнтська частина системи, через яку користувач або адміністратор взаємодіє з функціями системи. Інтерфейс отримує дані, надсилає запити на сервер та відображає результати.

Сервер – основна частина, яка обробляє запити, виконує тести, керує логікою транзакцій та забезпечує зв'язок між інтерфейсом, базою даних та частиною рішення.

База даних – частина програмного забезпечення, яка зберігає інформацію, використовується для перевірки статистики, отримання номерів маршрутів, обслуговування велосипеда, здійснення покупок, оновлення інформації та її статусу.

Рішення – частина рішення, яка бере участь у процесі рішення. Вона отримує запит, обробляє рішення та повертає результат.

**Опис сценарію користувача**

Цей сценарій демонструє послідовність дій користувача під час придбання треку.

Користувач вводить дані для авторизації. Інтерфейс передає ці дані на сервер через запит login(userName, password). Сервер звертається до БД із запитом findUser(userName, password) для перевірки облікових даних. У випадку якщо дані збігаються, база даних повертає підтвердження, після чого сервер повідомляє інтерфейсу про успіх авторизації. Інтерфейс перенаправляє користувача до його аккаунта.

Після авторизації користувач переходить до каталогу треків. Інтерфейс надсилає серверу запит на отримання списку доступних треків. Сервер звертається до бази даних, яка повертає наявні треки, інтерфейс відображає користувачу каталог музичного контенту. На цьому етапі користувач може налаштовувати фільтри, а система відображає відповідні результати.

Далі користувач обирає трек. Інтерфейс надсилає запит серверу getTrackById(id), сервер звертається до бази даних за допомогою запиту getTrack(id). База даних повертає інформацію про вибраний трек, а сервер передає ці дані інтерфейсу. У результаті цього, користувачу відображається сторінка треку з детальною інформацією про нього.

Після відкриття сторінки треку користувач може прослухати демоверсію. У цьому випадку інтерфейс запускає аудіоплеєр із демонстраційним фрагментом треку. Це дає змогу ознайомитися з музичним контентом перед покупкою, не отримуючи доступу до повної версії файлу.

Якщо користувач вирішує придбати трек, він натискає кнопку додавання до кошика. Інтерфейс надсилає серверу запит addToCart(userID, trackID). Сервер передає відповідний запит до бази даних, де інформація про вибраний трек додається до кошика користувача. Після успішного виконання, база даних повертає підтвердження, сервер передає його інтерфейсу, а користувач отримує повідомлення про те, що трек додано до кошика.

Коли користувач переходить до кошика. Інтерфейс відображає вміст кошика, зокрема обрані треки та інформацію, необхідну для оформлення покупки. Користувач натискає кнопку оплати, після чого інтерфейс надсилає серверу запит initiatePayment(). Сервер створює запис про покупку зі статусом pending, тобто очікування оплати. Далі формується запит до платіжного компонента requestPayment(amount, orderId), у якому передається сума замовлення та ідентифікатор покупки.

Після створення платіжного запиту система надає користувачу посилання на платіжну систему та перенаправляє його до відповідного інтерфейсу оплати. Користувач переходить до платіжної системи та виконує оплату. У разі успішної операції платіжна частина повертає підтвердження про успішну оплату. Після цього сервер оновлює статус покупки в базі даних за допомогою запиту updateStatus("success").

Завершальний етап, система повідомляє інтерфейс про зміну статусу покупки. Інтерфейс відображає користувачу придбаний трек у розділі придбаного контенту. Таким чином, після успішної оплати користувач отримує доступ до повної версії треку.

**Опис сценарію адміністратора**

Ця діаграма послідовності демонструє роботу адміністратора з додавання нового треку до системи.

Спочатку адміністратор вводить дані для авторизації. Інтерфейс передає ці дані на сервер за допомогою запиту login(adminName, password). Сервер звертається до бази даних через запит findAdmin(adminName, password) для перевірки облікових даних адміністратора. Якщо дані збігаються, база даних

повертає підтвердження, а сервер повідомляє інтерфейс про успішну авторизацію. Після цього адміністратор перенаправляється до адміністративної панелі.

Після успішної авторизації адміністратор обирає функцію додавання нового треку. Він вводить да додає необхідні дані та файли. Інтерфейс передає ці дані на сервер через запит addTrack(image, title, genre, mood, bpm, key, price, demoFile).

Сервер обробляє отриману інформацію та передає її до бази даних за допомогою запиту saveTrack(...). У базі даних створюється новий запис про трек, де зберігаються його основні характеристики. Після успішного збереження база даних повертає підтвердження про те, що зміни збережено.

Після цього сервер повідомляє інтерфейс про успішне додавання треку, а адміністратор отримує повідомлення, що новий трек завантажено. У результаті новий музичний контент стає доступним у каталозі системи для перегляду користувачами.

У діаграмі також передбачено альтернативний сценарій невдалої авторизації. Якщо введені адміністратором облікові дані не збігаються з даними в базі, база даних повертає повідомлення про помилку. Сервер передає інтерфейсу інформацію про безуспішну авторизацію, після чого адміністратор отримує повідомлення про неправильно введені дані та доступ до облікового запису заборонений.

Під час аналізу було ствреноо модель представлення основних процесів в системі, вона показує послідовність дій та логіку виконання.

Саме діаграма активності відображає ці основні процеси, вона використовується для моделювання динаміки роботи системи та показує як саме відбувається перехід від однієї дії до іншої.

У межах розроблюваної веб-орієнтованої системи продажу музичного контенту було побудовано дві діаграми активності які відображені в ***Додатку Б***: для користувача та адміністратора. Вони демонструють основні етапи взаємодії з веб-застосунком залежно від ролі учасника системи.

*Діаграма активності користувача* починається з відкриття ПЗ, після чого користувач переходить до каталогу, далі він може скористуватися фільтрами та обрати бажаний трек. Після вибору треку користувач має можливість прослухати демонстраційну версію. Після цього настає етап прийняття рішення щодо купівлі. Якщо користувач не бажає купувати трек, процес завершується, якщо користувач бажає купити, то він додає його до кошика, переходить до кошику та оплачує, отримує підтвердження про успішну покупку та після цього має можливість завантажити придбаний трек.

*Діаграма активності адміністратора* починається з входу в адмін-панель та проходження авторизації. Після цього адміністратор переходить до розділу керування треками, де може виконати одну з наступних операцій над треками: додати, видалити, редагувати. Після виконання обраної дії зміни зберігаються та процес завершується.

Діаграма активності мають основні елементи UML такі як: початковий вузол, потоки керування, логічне розгалуження та кінцевий вузол. Дії які подані у вигляді намальованого прямокутника із заокругленими кутами, стрілки показують вектор виконання процесу, а ромбовидний елемент та діаграмі користувача відображає умову прийняття рішень щодо купівлі.

Тобто, діаграма активності дозволяє представити логіку виконання основних процесів у веб-орієнтованій системі продажу музичного контенту. Вони допомагають визначити послідовність дій користувача та адміністратор та виявити ключові етапи роботи, збудувати основу для подальшої реалізації програми.

## 1.3Огляд існуючих функціональних аналогів

У сучасних умовах виконавці, стрімери, блогери та інші користувачі цифрового контенту не завжди мають навички для самостійного створення музичного матеріалу. Тому вони активно використовують музику для власних потреб. Однак, використання музики без відповідного дозволу не завжди є

легальним, оскільки більша частина аудіоконтенту захищена авторським правом і не може вільно використовуватись у відео, трансляції, рекламі чи інших комерційних матеріалах. У сучасних умовах цифрова індустрія активно використовує вебзастосунки для пошуку та купівлі треків, оскільки це швидко, зручно та доступно. Серед найбільш відомих можна виділити BeatStars, Airbit та Bandcamp. В межах огляду існуючих функціональних аналогів ми розглянемо саме їх, а також визначимо їх переваги й недоліки.

**BeatStars**

BeatStars – це одна з найвідоміших онлайн-платформ для купівлі музичного цифрового контенту. Сервіс орієнтований насамперед на музичних продюсерів, виконавців і авторів, які шукають інструментали для власних композицій. Платформа дозволяє продавати біти, пісні, вокальні партії, sound kits. BeatStars також має можливості пошуку за жанром, BPM, настроєм та іншими характеристиками, що є важливим для швидкого підбору потрібного музичного матеріалу.

Для розроблюваної системи BeatStars є важливим аналогом, оскільки він демонструє ефективну модель продажу цифрової музики через каталог, фільтрацію та попереднє прослуховування. Водночас така платформа є глобальним маркетплейсом, а не окремим авторським магазином, орієнтованим на конкретну аудиторію, але саме така платформа є базою для розширення до маркетплейсу.

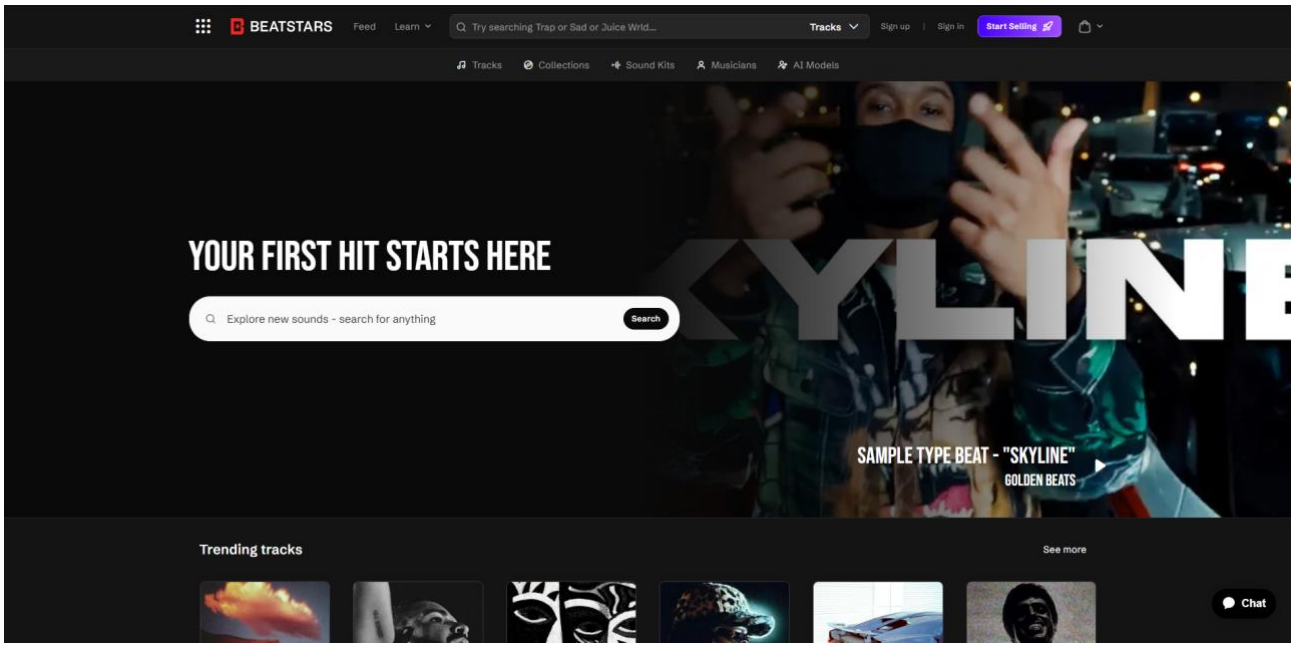



*Рис. 2 Інтерфейс системи BeatStars*

**Переваги:**

- Велика база музичного контенту;
- Можливість купівлі, продажу та ліцензування бітів;
- Пошук за жанром, BPM, настроєм та іншими параметрами;
- Підтримка завантаження MP3, WAV та інших матеріалів;
- Можливість створення власної сторінки продавця;

**Недоліки:**

- Значна конкуренція між авторами контенту;
- Частина функцій доступна лише в платних тарифах;
- Інтерфейс і функціонал надмірний для користувача;
- відсутній акцент на українську аудиторію та локальний ринок продажу авторських треків;
- платформа більше орієнтована на міжнародний маркетплейс, ніж на окремий авторський магазин.

**Airbit**

Airbit – це онлайн-маркетплейс для продажу бітів та інструменталів, який надає продюсерам інструменти для завантаження, продажу й монетизації музичного контенту. Платформа підтримує продаж бітів, приймання оплат через

PayPal і банківські картки. За даними самої платформи, Airbit має понад 800 тисяч користувачів і понад 2 млн проданих бітів.

Airbit – це можна розглядати як аналог системи продажу музичного контенту, оскільки він також поєднує каталог аудіоматеріалів, попереднє прослуховування та оформлення покупки.

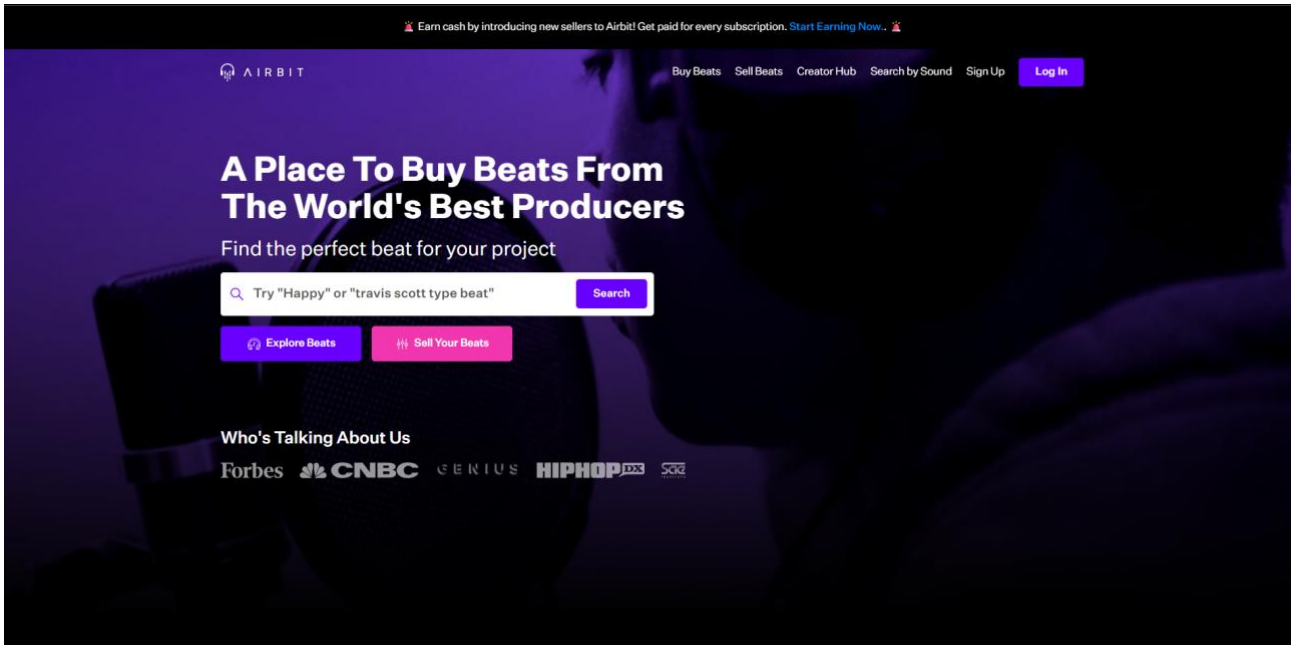



*Рис. 3 Інтерфейс системи Airbit*

**Переваги:**

- спеціалізація на продажу бітів та інструменталів;
- можливість створення окремого магазину для продавця;
- підтримка платежів через PayPal і банківські картки;
- можливість завантаження великої кількості музичного контенту, безкоштовно.

**Недоліки:**

- орієнтація переважно на ринок бітів, а не на універсальний продаж музичного контенту;
- частина можливостей залежить від тарифного плану;
- для початківців система може бути складною через велику кількість налаштувань;

- менша впізнаваність серед звичайних слухачів порівняно з більшими музичними платформами;
- відсутність локальної адаптації під український ринок і потреби україномовної аудиторії.

**Bandcamp**

Bandcamp – це онлайн-платформа для продажу цифрової музики, фізичних носіїв і мерчу, яка орієнтована на підтримку незалежних артистів. Сервіс позиціонується як онлайн-магазин і музична спільнота, де слухачі можуть відкривати нову музику, підтримувати виконавців і купувати цифрові альбоми або окремі треки. Після покупки користувач може отримати доступ до стримінгу та завантаження музики у високій якості.

Bandcamp є близьким аналогом для розроблюваної системи, оскільки також підтримує продаж авторської музики. Однак платформа більше зосереджена на альбомах та музичній спільноті, тоді як у розроблюваному застосунку акцент робиться на зручному каталозі окремих авторських треків, їх фільтрації, попередньому прослуховуванні та швидкому придбанні.

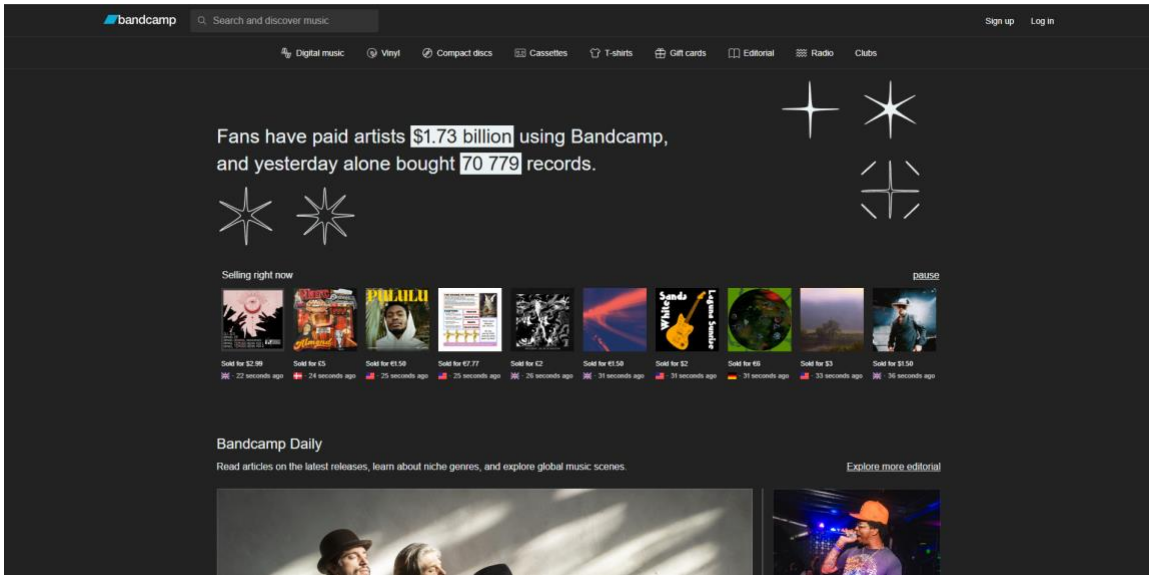



*Рис. 4 Інтерфейс системи Bandcamp*

**Переваги:**

- орієнтація на незалежних артистів;
- можливість продажу цифрової музики, фізичних носіїв і мерчу;
- підтримка завантаження придбаної музики;
- простий механізм підтримки виконавців;
- наявність музичної спільноти та сторінок артистів.

**Переваги:**

- менше можливостей для детальної фільтрації за технічними характеристиками треку, такими як BPM або тональність;
- платформа більше орієнтована на альбоми та артистів, ніж на швидкий підбір окремих треків за параметрами;
- обмежені можливості персоналізації інтерфейсу окремого магазину;
- не всі функції підходять для реалізації вузькоспеціалізованого магазину музичних треків.
- відсутній окремий акцент на українську аудиторію, локальну валюту та специфіку продажу авторських треків у межах українського ринку.

Отже, розглянуті вище системи мають розвинений функціонал для продажу, поширення та ліцензування музичного контенту. Вони не повністю відповідають потребам невеликого веборієнтованого застосунку на українську аудиторію. BeatStars і Airbit більшою мірою зосереджені на глобальному маркетплейсі, тоді як Bandcamp більше орієнтований на незалежних артистів, альбоми та музичну спільноту. Саме тому є доцільним створення власної веборієнтованої системи, яка поєднує каталог треків, фільтрацію, попереднє прослуховування, кошик, список бажаного, коментарі, особистий кабінет та адміністративну панель для керування контентом та користувачами. До переваг розроблюваної системи можна віднести наявність базового функціоналу для подальшого масштабування та розширення до рівня повноцінного маркетплейсу.

## 1.4 Постановка завдання

Метою проєкту є розроблення програмного забезпечення для веборієнтованої системи продажу музичного контенту, яка дозволяє користувачам швидко знаходити авторські треки, прослуховувати демонстраційні версії, додавати композиції до кошика або списку бажаного, оформлювати покупку та отримувати доступ до придбаних музичних файлів.

Система орієнтована на українську аудиторію та може бути використана як платформа для продажу цифрового музичного контенту з можливістю подальшого масштабування до рівня маркетплейсу.

**Вхідні дані системи**

Розроблена системаповинна обробляти такі вхідні дані:

- Дані користувача: ім’я, електронна пошта, пароль.
- Дані музичного контенту: назва, жанр, настрій, темп, тональність, тривалість, ціна, обкладинка, демонстраційний файл та повний аудіофайл.
- Дані кошика: користувач, обрані треки, кількість позицій та загальна сума покупки.
- Дані користувача і збережені треки;
- Дані покупки: користувач, придбані треки, дата й час придбання, статус оплати;
- Дані адміністратора: обліковий запис, роль, права доступу.

**Вихідні дані системи:**

Система повинна формувати такі результати:

- Каталог музичних треків із можливістю пошуку та фільтрації
- Сторінку окремого треку з характеристиками, обкладинкою та можливістю попереднього прослуховування;
- Вміст кошика користувача;
- Доступ до завантаження придбаного музичного файлу;
- Адміністративний інтерфейс для керування треками, користувачами, перегляду статистики продажів.

**Умовно-постійна інформація**

До умовно-постійної інформації системи належать:

- перелік жанрів музичного контенту;

- перелік настроїв треків;
- допустимі значення темпу та тональності треків;
- ролі користувачів: гість, користувач, адміністратор;
- статуси покупки та оплати;
- налаштування доступу до demo-файлів і повних аудіофайлів;
- дані адміністративного доступу.

**Функціональні вимоги до системи**

Розроблена система має забезпечувати реєстрацію та авторизацію користувачів. Для захисту персональних даних необхідно реалізувати механізм входу в обліковий запис. Після авторизації користувач отримує доступ до кошика, покупок та особистого кабінету.

**Для гостя система має забезпечувати:**

Перегляд головної сторінки, каталогу музичних треків, сторінки окремого треку, пошук, фільтрацію та попереднє прослуховування demo-версій. Гість також повинен мати можливість перейти до реєстрації або авторизації.

**Для зареєстрованого користувача система повинна забезпечувати:**

- пошук і фільтрацію треків за назвою, жанром, настроєм, темпом і тональністю;
- попереднє прослуховування demo-версій;
- додавання треків до кошика;
- видалення треків із кошика;
- оформлення покупки;
- оплату через платіжний сервіс у тестовому режимі;
- перегляд придбаних треків в особистому кабінеті;
- завантаження придбаних музичних файлів;
- захист від повторного придбання вже купленого треку.

**Для адміністратора система повинна забезпечувати:**

- вхід до адміністративної панелі;
- додавання нових музичних треків;
- редагування інформації про треки;
- видалення треків із каталогу;
- завантаження demo-файлу, повного аудіофайлу та обкладинки;
- блокування або видалення облікових записів користувачів за потреби;
- перегляд статистики продажів;
- контроль актуальності музичного каталогу.

**Нефункціональні вимоги до системи:**

*Безпека:* Система повинна забезпечувати захист облікових записів користувачів, розмежування прав доступу між користувачем та адміністратором, а також захист персональних даних і придбаного музичного контенту від несанкціонованого доступу.

*Зручність використання:* Інтерфейс системи має бути зрозумілим, логічним і зручним для користувачів різного рівня підготовки. Основні функції, такі як пошук, прослуховування, додавання до кошика та оформлення покупки, повинні бути доступні без зайвих дій.

*Адаптивність:* Вебзастосунок повинен коректно відображатися на різних пристроях, зокрема на комп’ютерах, планшетах і смартфонах.

*Швидкодія:* Система повинна забезпечувати швидке завантаження сторінок, стабільну роботу каталогу, пошуку, фільтрів та аудіоплеєра.

*Масштабованість:* Архітектура системи повинна передбачати можливість подальшого розширення функціоналу, зокрема додавання нових авторів, треків, категорій, платіжних можливостей і перехід до формату повноцінного маркетплейсу.

*Сумісність:* Система повинна працювати в сучасних браузерах і бути доступною через мережу Інтернет після розміщення на хостингу.

# 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1. Логічна модель даних у вигляді ER-діаграми

Логічна модель даних виконує важливу роль розробці ПЗ, оскільки вона дозволяє формалізувати структуру зберігання, визначити основні сутності, їхні атрибути та взаємодію. Для подання структури БД використовується ER-діаграма, тобто діаграма “сутність-зв’язок”, саме вона відобразить логіку організаційних даних ще до етапу реалізації фізичної бази.

У розробленій веборієнтованій системі продажу музичного контенту було визначено такі ключові сутності: користувачі, музичні треки, кошики, елементи кошика, покупки, елементи покупки. Кожна сутність має набір атрибутів, які описують її властивості та забезпечують збереження необхідної інформації для роботи системи. Для зв’язку між сутностями використовуються зовнішні ключі, що дозволяє забезпечити цілісність даних і правильну взаємодію між користувачами, треками, кошиком, покупками та іншими функціональними модулями.

***ER-діаграма розробленої логічної моделі розміщена в Додатку В.***

**Опис сутностей та атрибутів**

Сутність *User* використовується для зберігання інформації про зареєстрованих користувачів. Вона містить дані, необхідні для авторизації, ідентифікації користувача та розмежування прав доступу.

- id — унікальний ідентифікатор користувача;
- name — ім’я користувача;
- email — логін;
- password_hash — хешований пароль;
- role — роль користувача в системі;
- is_blocked — ознака блокування облікового запису;
- created_at — дата створення облікового запису.

Сутність *Track* є центральною сутністю системи, саме трек виступає основним цифровим продуктом, який користувач може переглядати, зберігати у списку бажаного, прослуховувати, додавати до кошика та купувати.

- id — унікальний ідентифікатор треку;
- title — назва треку;
- genre — жанр композиції;
- mood — настрій треку;
- bpm — темп композиції;
- key — тональність;
- price — ціна треку;
- duration — тривалість композиції;
- image_url — посилання на обкладинку;
- demo_file_url — посилання на demo-файл для попереднього прослуховування;
- full_file_url — посилання на повний аудіофайл;
- created_at — дата додавання треку.

Сутність *Cart* використовується для збереження інформації кошика. Кошик пов’язаний з користувачем і містить перелік треків, які користувач бажає придбати.

- id — унікальний ідентифікатор кошика;
- user_id — зовнішній ключ, що посилається на користувача.

Сутність *CartItem* є проміжною сутністю між треком та кошиком. Вона дозволяє зберігати інформацію про трек, а які треки були додані користувачем до кошика.

- id — унікальний ідентифікатор елемента кошика;
- cart_id — зовнішній ключ, що посилається на кошик;
- track_id — зовнішній ключ, що посилається на трек.

Сутність *Purchase* використовується для збереження інформації про придбання контенту. Вона пов’язана з користувачем, містить дату створення покупки.

- id — унікальний ідентифікатор покупки;
- user_id — зовнішній ключ, що посилається на користувача;
- created_at — дата та час створення покупки.

Сутність *PurchaseItem* використовується для збереження треків, які входять до покупки. Вона є проміжною сутністю між покупкою та треком.

- id — унікальний ідентифікатор елемента покупки;
- purchase_id — зовнішній ключ, що посилається на покупку;
- track_id — зовнішній ключ, що посилається на музичний трек.

**Опис зв’язків**

- Один користувач може мати лише один кошик. Реалізація зв’язку через зовнішній ключ *user_id* у таблиці *Cart*, що посилається на первинний ключ id у таблиці User.
- Один кошик може містити багато елементів кошика. Зв’язок реалізовано через зовнішній ключ *cart_id* у таблиці *CartItem*, що посилається на первинний ключ id у таблиці *Cart*. Кожен елемент кошика належить лише одному кошику.
- Один музичний трек може бути доданий до багатьох кошиків, а один кошик може містити багато треків. Для реалізації такого зв’язку використовується проміжна таблиця *CartItem*, яка містить зовнішні ключі *cart_id* і *track_id*. Завдяки цьому реалізується зв’язок “багато до багатьох”.
- Один користувач може мати багато покупок. Реалізація зв’язку через зовнішній ключ *user_id* у таблиці *Purchase*, що посилається на

первинний ключ id у таблиці *User*. Кожна покупка пов’язана з одним користувачем.

- Одна покупка може містити багато придбаних треків. Зв’язок реалізовано через зовнішній ключ *purchase_id* у таблиці *PurchaseItem*, що посилається на первинний ключ id у таблиці *Purchase*.
- Один музичний трек може входити до багатьох покупок, а одна покупка може містити багато треків. Для реалізації зв’язку використовується проміжна таблиця *PurchaseItem, яка містить,* зовнішні ключі purchase_id і track_id.

**Перевірка відповідності нормальним формам**

Для того щоб перевірити ефективність зберігання, обробки та уникання зайвих даних у БД, важливо привести модель, що відповідає принципам нормалізації, тобто організувати атрибути і відношення у БД таким чином, щоб зменшити дублювання даних та забезпечити цілісність структури.

У процесі розробки структури бази даних було дотримано основні нормальні форми до третьої включно.

**Перша нормальна форма (1НФ)**

Модель відповідає першій нормальній формі, тому що всі атрибути сутностей містять неподільні значення. Наприклад, у таблиці *Track* окремо зберігаються назва треку, жанр, настрій, темп, тональність, ціна та посилання на файли. У таблиці *User* окремо зберігаються ім’я, хеш пароля, роль, статус блокування.

**Друга нормальна форма (2НФ)**

Модель відповідає другій нормальній формі, оскільки всі неключові атрибути повністю залежать від первинного ключа таблиці. Наприклад, у таблиці *Track* усі характеристики залежать від його ідентифікатора id, а в таблиці *Purchase* дата покупки та користувач пов’язані з конкретним записом покупки. Проміжні таблиці *CartItem, PurchaseItem* використовуються для подання зв’язків між сутностями без дублювання даних.

**Третя нормальна форма (3НФ)**

Модель також відповідає і третій нормальній формі, так як неключові атрибути не залежать один від одного, а залежать лише від первинного ключа. Наприклад, інформація про користувача не дублюється в таблицях покупок, а зберігається окремо в таблиці *User*.

Таким чином, логічна модель даних веборієнтованої системи продажу музичного контенту відповідає вимогам до третьої нормальної форми, саме це дозволяє забезпечити цілісність даних, зменшити дублювання інформації та створити надійну основу для фізичної структури БД.

## 2.2 Вибір системи управління базами даних

Для вибору СУБД було розглянуто два основні типи рішень: файл-серверні та клієнт-серверні системи управління БД.

Файл-серверні СУБД можна використовувати для невеликих проєктів, де обсяг даних та кількість користувачів є незначними. До таких систем можна віднести SQLite або Microsoft Access. Їхні переваги полягають у простоті встановлення, мінімальних вимогах до налаштування та зручності використання для локальних систем.

Однак для веборієнтованої системи продажу музичного контенту файл-серверні СУБД мають такі обмеження:

• неефективність при одночасній роботі багатьох користувачів;
• значні обмеження для масштабування;
• менші можливості керування доступом і безпекою;
• складність використання у повноцінному клієнт-серверному веборієнтованому застосунку;
• обмежена придатність для системи, яка має працювати на хостингу та обробляти покупки, музичний каталог, кошик і дані користувачів.

Клієнт-серверні СУБД краще підходять для вебзастосунків, оскільки вони забезпечують підтримку багатокористувацького доступу, централізоване зберігання даних, транзакції, резервне копіювання, керування правами доступу

та мають високу здатність до масштабування. До таких систем належать PostgreSQL, MySQL, MariaDB, Microsoft SQL Server.

Перевагами клієнт-серверних СУБД є:

• можливість одночасної роботи багатьох користувачів;

• забезпечення цілісності даних;

• ефективна робота зі складними зв’язками між таблицями;

• можливість інтеграції з серверною частиною вебзастосунку;

• підтримка резервного копіювання та відновлення даних;

• можливість подальшого масштабування системи.

З огляду на необхідність централізованого зберігання інформації, підтримки зв’язків між користувачами, треками, кошиком і покупками, а також інтеграції з серверною частиною на FastAPI, для реалізації системи було вирішено обрати PostgreSQL. Разом із SQLAlchemy ця СУБД дозволяє описувати структуру таблиць у вигляді моделей та взаємодіяти з БД із серверної частини застосунку. У файлі підключення до бази даних використовується PostgreSQL-адреса, а для роботи із сесіями застосовується sessionmaker.

**Обґрунтування вибору PostgreSQL**

Вибір PostgreSQL як основної СУБД обґрунтований її надійністю при роботі зі складними реляційними зв'язками. Оскільки архітектура платформи передбачає динамічне керування кошиком, транзакційність покупок та розмежування ролей (гість, клієнт, адмін), використання зв'язків ForeignKey на рівні бази даних гарантує цілісність інформації та захищає від аномалій при очищенні кошика після оплати.

**Переваги:**

**Інтеграція з серверною частиною**

PostgreSQL добре інтегрується з Python та FastAPI через SQLAlchemy. Це дозволяє описувати сутності системи у вигляді моделей, працювати з таблицями через ORM-підхід, а також зручно виконувати такі операції, як створення, читання, оновлення та видалення даних.

**Підтримка складних зв’язків між таблицями**

Для веборієнтованої системи продажу важливо коректно зберігати зв’язки між користувачами, продуктами, кошиком і покупками. PostgreSQL дозволяє реалізувати такі зв’язки за допомогою первинних і зовнішніх ключів. Наприклад, кошик пов’язаний з користувачем, елемент кошика — з кошиком і треком, а покупка — з користувачем і придбаними треками. Такі зв’язки відображені у моделях БД через ForeignKey і relationship.

**Надійність і цілісність даних**

PostgreSQL підтримує транзакції, що є важливим для операцій придбання музичного контенту. Під час оформлення покупки система повинна коректно створити запис про покупку, додати придбані треки та очистити кошик користувача. Використання транзакцій дозволяє уникнути проблем із некоректним збереженням даних або неповним завершенням процесу покупки.

**Масштабованість**

У майбутньому структура системи може бути розширена додаванням сутностей авторів, ліцензій, рейтингів та інших модулів без необхідності повної перебудови БД.

**Безпека**

PostgreSQL підтримує механізми керування користувачами, правами доступу та захистом даних. Це є важливою складовою для збереження облікових записів користувачів, інформації про покупки та доступу до придбаних музичних файлів. На рівні програмного забезпечення додатково використовується хешування паролів і JWT-авторизація.

**Продуктивність**

PostgreSQL ефективно працює з великими обсягами структурованих даних, підтримує індексацію та оптимізацію запитів. Це важливо для швидкого пошуку треків, фільтрації за характеристиками, перегляду каталогу, отримання даних кошика, покупок і статистики продажів.

Для обґрунтування вибору PostgreSQL проведено порівняльний аналіз у таблиці 1.

| **Критерій** | **PostgreSQL** | **MySQL** | **SQLite** |
|---|---|---|---|
| Тип СУБД | клієнт-серверна реляційна | клієнт-серверна реляційна | файлова реляційна |
| Підтримка багатокористувацької роботи | висока | висока | обмежена |
| Підтримка складних зв’язків | висока | висока | базова |
| Масштабованість | висока | висока | низька |
| Придатність для вебзастосунку | висока | висока | переважно для локальних або малих проєктів |
| Інтеграція з FastAPI/SQLAlchemy | зручна | зручна | зручна для прототипів |
| Надійність для комерційної системи | висока | висока | обмежена |

*Таблиця 1. Порівняльна таблиця СУБД*

Отже із порівнянь СУБД ми побачили що PostgreSQL та MySQL є чудовим вибором для веборієнтованої системи продажу музичного контенту, оскільки я маю більше досвіду в PostgreSQL саме, тому вона отримує вищій пріорітет вибору. PostgreSQL забезпечує надійне зберігання даних, підтримує складні зв’язки між сутностями, добре інтегрується з FastAPI та SQLAlchemy, має достатній рівень продуктивності, безпеки й масштабованості.

### 2.3 Розробка структури інформаційної бази

У процесі створення веборієнтованої системи продажу музичного контенту особливу увагу приділено розробці структури інформаційної бази, оскільки саме вона забезпечує збереження, обробку та взаємозв’язок основних даних системи. До інформаційної бази даних належать відомості про користувачів, треки, кошик, покупки, придбані файли та службові дані.

Взаємодія серверної частини з базою даних здійснюється за допомогою SQLAlchemy ORM. Такий підхід спрощує процес створення структури бази даних, забезпечує зручну роботу з об’єктами та дозволяє уникнути написання великої кількості SQL-запитів.

Замість підходу, коли таблиці створюються безпосередньо за допомогою SQL-команд, у цьому проєкті структура БД формується на основі ORM-моделей. Кожна модель відповідає окремій таблиці бази даних, а поля визначають атрибути таблиці, типи даних, первинні та зовнішні ключі.

Підключення до БД реалізовано в окремому модулі, в якому створюється об’єкт підключення до PostgreSQL, налаштовується сесія з БД та визначається базовий клас для моделей. Для цього використано функції create_engine, sessionmaker та declarative_base.

Підключення до PostgreSQL відбувається через змінну DATABASE_URL, після цього створюється об’єкт engine, сесія SessionLocal та базовий клас Base. Також визначено функцію get_db(), яка використовується у FastAPI для отримання сесії бази даних під час обробки HTTP-запитів. Фрагмент підключення до бази даних відображено на *Рис. 5*.

```
DATABASE_URL = "postgresql://postgres:password@localhost:5432/music_store"

engine = create_engine(DATABASE_URL)

SessionLocal = sessionmaker(
    autocommit=False,
    autoflush=False,
    bind=engine
)

Base = declarative_base()
```

*Рис. 5 Створення бази даних “Music_store”*

Функція get_db() виконує відкриття сесії, передачу її у відповідний маршрут FastAPI та закриває після завершення роботи із запитом. Такий підхід зменшує ризик помилок при роботі з даними.

Створення таблиць реалізовано методом опису ORM-моделей. Кожна модель — це Python-клас, який успадковується від базового класу Base. Назва таблиці задається через атрибут __tablename__, а поля — за допомогою типів Column, String, Numeric, Integer, DateTime, Boolean та ForeignKey.

Для центральної сутності “музичний трек” зберігаються такі дані: назва, жанр, настрій, темп, тональність, ціна, тривалість, посилання на обкладинку, demo-файл і повний аудіофайл. Фрагмент відображено на *Рис. 6*.

```python
class Track(Base):
    __tablename__ = "tracks"

    id = Column(Integer, primary_key=True, index=True)

    title = Column(String(255), nullable=False)

    genre = Column(String(100), nullable=True)
    mood = Column(String(100), nullable=True)

    bpm = Column(Integer, nullable=True)
    key = Column(String(20), nullable=True)

    price = Column(Numeric(10, 2), nullable=False)

    duration = Column(Integer, nullable=True)

    image_url = Column(String(500), nullable=True)
    demo_file_url = Column(String(500), nullable=True)
    full_file_url = Column(String(500), nullable=True)

    created_at = Column(DateTime(timezone=True), server_default=func.now())

    __table_args__ = (
        CheckConstraint("length(title) >= 2", name="check_track_title_min_length"),

        CheckConstraint(
            "bpm IS NULL OR (bpm >= 40 AND bpm <= 250)",
            name="check_track_bpm_range"
        ),

        CheckConstraint(
            "price >= 0",
            name="check_track_price_positive"
        ),

        CheckConstraint(
            "duration IS NULL OR duration > 0",
            name="check_track_duration_positive"
        ),

        CheckConstraint(
            "duration IS NULL OR duration <= 3600",
            name="check_track_duration_max"
        ),
    )
```

*Рис. 6 Створення опису моделі музичного треку “Tracks”*

Таким чином, ORM-моделі дають дозвіл для опису структури таблиці без SQL-запитів, забезпечивши зручну взаємодію серверної логіки з БД. Повний код створення всіх таблиць із зв’язками подано в ***Додатку Д***.

Щоб забезпечити цілісність даних у базі, було використано зовнішні ключі та зв’язки між таблицями. У SQLAlchemy це реалізовується через ForeignKey та relationship.

Кошик відображено таблицею Cart, яка містить зовнішній ключ user_id, що посилається на таблицю користувачів. Фрагмент відображено на *Рис. 8*. Один

користувач може мати один кошик, а кошик може містити багато елементів. Для збереження окремих треків у кошику використовується таблиця CartItem, яка містить посилання на кошик і трек. Фрагмент відображено *на Рис. 9*.

```python
class Cart(Base):
    __tablename__ = "carts"

    id = Column(Integer, primary_key=True, index=True)

    user_id = Column(
        Integer,
        ForeignKey("users.id", ondelete="CASCADE"),
        unique=True,
        nullable=False
    )

    user = relationship("User")

    items = relationship(
        "CartItem",
        back_populates="cart",
        cascade="all, delete-orphan"
    )

    __table_args__ = (
        CheckConstraint("user_id > 0", name="check_cart_user_id_positive"),
    )
```

*Рис. 7 Створення опису моделі кошику "Carts"*

```python
class CartItem(Base):
    __tablename__ = "cart_items"

    id = Column(Integer, primary_key=True, index=True)

    cart_id = Column(
        Integer,
        ForeignKey("carts.id", ondelete="CASCADE"),
        nullable=False
    )

    track_id = Column(
        Integer,
        ForeignKey("tracks.id", ondelete="CASCADE"),
        nullable=False
    )

    cart = relationship("Cart", back_populates="items")
    track = relationship("Track")

    __table_args__ = (
        UniqueConstraint(
            "cart_id",
            "track_id",
            name="unique_track_in_cart"
        ),

        CheckConstraint("cart_id > 0", name="check_cart_item_cart_id_positive"),
        CheckConstraint("track_id > 0", name="check_cart_item_track_id_positive"),
    )
```

*Рис. 8 Створення опису моделі кошику "Cart-items"*

Для збереження інформації про придбання контенту використовується таблиця Purchase. Вона пов'язана з користувачем через зовнішній ключ user_id. Окремі треки, що входять до покупки, зберігаються в таблиці PurchaseItem, яка посилається на покупку та трек. Фрагмент реалізації покупки показано на *Рис. 9* (Purchase) та *Рис. 10* (PurchaseItem).

```python
class Purchase(Base):
    __tablename__ = "purchases"

    id = Column(Integer, primary_key=True, index=True)

    user_id = Column(
        Integer,
        ForeignKey("users.id", ondelete="CASCADE"),
        nullable=False
    )

    created_at = Column(DateTime(timezone=True), server_default=func.now())

    user = relationship("User")

    items = relationship(
        "PurchaseItem",
        back_populates="purchase",
        cascade="all, delete-orphan"
    )

    __table_args__ = (
        CheckConstraint("user_id > 0", name="check_purchase_user_id_positive"),
    )
```

*Рис. 9 Створення опису моделі для збереження дії придбання контенту.*

```python
class PurchaseItem(Base):
    __tablename__ = "purchase_items"

    id = Column(Integer, primary_key=True, index=True)

    purchase_id = Column(
        Integer,
        ForeignKey("purchases.id", ondelete="CASCADE"),
        nullable=False
    )

    track_id = Column(
        Integer,
        ForeignKey("tracks.id", ondelete="RESTRICT"),
        nullable=False
    )

    purchase = relationship("Purchase", back_populates="items")
    track = relationship("Track")

    __table_args__ = (
        UniqueConstraint(
            "purchase_id",
            "track_id",
            name="unique_track_in_purchase"
        ),

        CheckConstraint(
            "purchase_id > 0",
            name="check_purchase_item_purchase_id_positive"
        ),

        CheckConstraint(
            "track_id > 0",
            name="check_purchase_item_track_id_positive"
        ),
    )
```

*Рис. 10 Створення опису моделі для збереження дії придбання контенту.*

Такі зв’язкі забезпечують правильне збереження інформації про користувачів, їхні кошики, покупки та придбані музичні треки.

Після опису моделей вони створюються в БД PostgreSQL за допомогою методу Base.metadata.create_all(bind=engine). Завдання цього методу полягає в аналізі моделей, які успадкувалися від Base, та створенні відповідних таблиць у БД, якщо вони ще не існують.

Такий підхід до створення таблиць зручний на етапі розробки, тому що дозволяє швидко побудувати структуру БД відповідно до описаних ORM-моделей.

Фізична структура бази даних розробленої системи складається з набору таблиць, що відповідають основним сутностями розробленого програмного забезпечення.

Основні таблиці:

- users — зберігає дані користувачів;
- tracks — містить інформацію про музичні треки;
- carts — зберігає кошики користувачів;
- cart_items — містить треки, додані до кошика;
- purchases — зберігає інформацію про покупки;
- purchase_items — містить треки, що входять до складу покупки;

Основні типи даних для опису полів:

- Integer — для унікальних ідентифікаторів і числових значень;
- String — для текстових даних, таких як ім’я, назва треку, жанр;
- Numeric — для збереження ціни музичного треку;
- Boolean — для логічних значень, наприклад статусу блокування користувача;
- DateTime — для збереження дати створення записів;
- ForeignKey — для реалізації зв’язків між таблицями.

Фізична структура бази даних відповідає логічній моделі, яка була описана у попередньому пункті. Вона забезпечує збереження всіх необхідних даних для повноцінної роботи системи: користувачів, треків, кошика, покупок та службових даних. Відображення фізичної структури подано в ***Додатку Е.***

# 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення є одним із ключових етапів створення веборієнтованої системи для продажу музичного контенту. Саме на цьому етапі здійснюється перехід від аналізу, створення моделей і проєктування БД до реалізації функціонування системи. Основна увага приділяється створенню клієнтської та серверної частин, організації взаємодії між ними, реалізації бізнес-логіки, авторизації користувачів, роботі з музичним каталогом, кошиком, покупками та адміністративною панеллю.

У межах цього розділу буде розглянуто структуру програмного забезпечення, основні абстракції, компоненти системи та їхню взаємодію. Також буде описано вибір інструментів для створення веборієнтованої системи, зокрема React для клієнтської частини, FastAPI та Python для серверної логіки, PostgreSQL для збереження даних, SQLAlchemy для роботи з БД, Axios і REST API для обміну інформацією.

Розроблена система для продажу музичного контенту має клієнт-серверну архітектуру. Клієнтська частина відображає інтерфейс для користувачів різного рівня, забезпечує навігацію між сторінками, пошук і фільтрацію треків, попереднє прослуховування, роботу з кошиком та особистим кабінетом.

Серверна частина відповідає за обробку запитів, авторизацію, перевірку прав доступу, взаємодію з базою даних, керування треками, покупками та користувачами. Також було приділено увагу розмежуванню ролей користувачів.

## 3.1. Абстракції предметної області

Абстракція — це важливий етап створення програмного забезпечення, який представляє узагальнений вигляд об’єкта або процесу та відображає лише найважливіші властивості й функції, необхідні для роботи системи.

Використання абстракцій дає змогу спростити складну структуру та виділити основні сутності з визначеними ролями у системі.

Для веборієнтованої системи продажу музичного контенту основними абстракціями є користувач, адміністратор, музичний трек, каталог треків, кошик, покупка, платіж, особистий кабінет та аудіоплеєр. Кожна з описаних абстракцій бере участь у реалізації основного сценарію роботи застосунку.

Абстракція “музичний трек” є основною для обраної предметної області, оскільки саме трек є продуктом, навколо якого відбувається логіка роботи системи. Трек має властивості, завдяки яким користувач може знайти композицію, ознайомитися з її вмістом, переглянути додаткову інформацію та пов’язати з ним покупку.

Абстракція користувача описує учасника системи, який взаємодіє з музичним контентом. Користувач може використовувати більшість функцій програмного забезпечення, зокрема пошук, фільтрацію, оформлення покупки та інші можливості. Для роботи з платформою користувач має створити обліковий запис, який надає йому відповідну роль і право на використання певних функцій.

До окремої абстракції належить адміністратор. Він має розширені права доступу. Його завданням є керування цифровим продуктом та обліковими записами користувачів.

Каталог треків виступає абстракцією, яка відповідає за організацію музичного контенту. Він містить доступні композиції, фільтрацію та пошук. Завдання цієї абстракції полягає в ознайомленні користувача з контентом і допомозі в пошуку потрібного треку.

Аудіоплеєр є функціональною абстракцією системи. Його функція полягає в забезпеченні попереднього прослуховування демонстраційної версії або повної версії треку залежно від рівня доступу.

Абстракція кошика виконує роль тимчасового списку, в якому знаходяться обрані треки для подальшої покупки. Покупка, у свою чергу, фіксує факт придбання музичного контенту, зберігає інформацію про придбані треки, дату та

статус оплати. Після успішного придбання користувач отримує доступ до файлу купленого контенту.

Абстракція платежу відповідає за оплату покупки. Вона містить інформацію, пов’язану з ціною, статусом та результатом виконання платіжної операції.

Особистий кабінет є абстракцією, що забезпечує доступ до персональних даних користувача і придбаних ним цифрових продуктів.

Усі перелічені абстракції дозволяють створити структуру програмного забезпечення, визначити об’єкти та функції системи. Фрагмент відображення всіх перелічених абстракцій подано на *Рис. 11*.

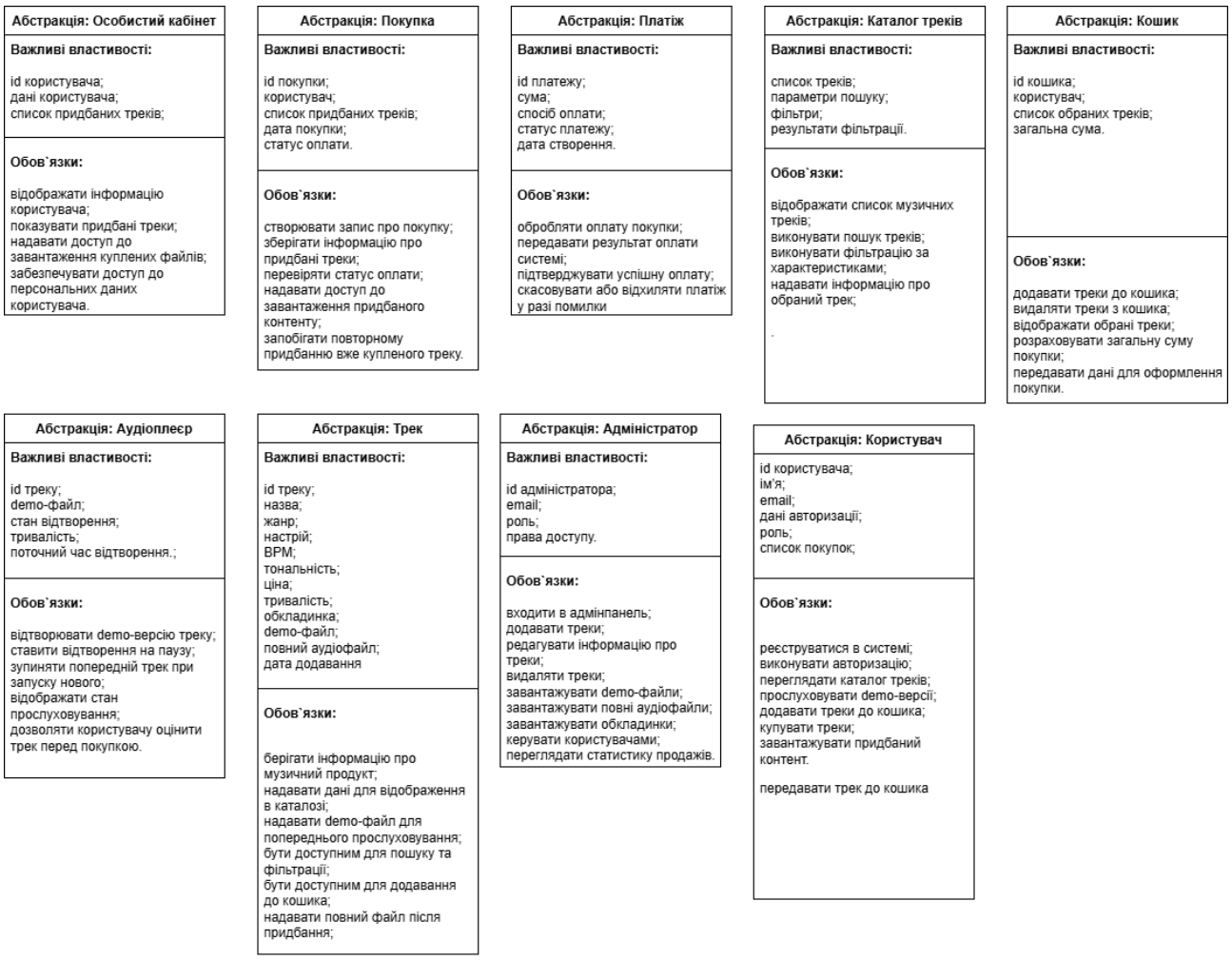



*Рис.11 Абстракції веборієнтованої системи для продажу музичного контенту.*

## 3.2. Моделювання структури та взаємодії об’єктів

Моделювання структури та взаємодії об’єктів дозволяє зрозуміти, як окремі частини програмного забезпечення пов’язані між собою, які дані передаються та як реалізуються процеси роботи вебзастосунку.

Клієнтська частина реалізована за допомогою React, який відповідає за відображення інтерфейсу в розробленому програмному забезпеченні. Серверна частина реалізована за допомогою FastAPI. Завдяки цьому фреймворку в системі обробляються запити від клієнта, виконується бізнес-логіка, перевіряється авторизація, здійснюється взаємодія з PostgreSQL та повертаються результати у форматі JSON.

Взаємодія між клієнтською та серверною частиною виконується через REST API. Наприклад, під час відкриття користувачем каталогу frontend надсилає запит до backend, сервер отримує дані у вигляді списку треків із БД і повертає їх у клієнтську частину. Клієнтська частина, за яку відповідає React, за допомогою компонентів відображає треки у вигляді карток із назвою, обкладинкою, ціною, характеристиками та можливістю попереднього прослуховування демонстраційного треку.

Основні об’єкти системи взаємодіють один із одним відповідно до робочих сценаріїв користувача. Користувач вибирає трек з каталогу, прослуховує демо-версію, додає його до кошика або списку бажаного, а потім може перейти до покупки. Після прийняття рішення про покупку система перевіряє дані кошика, створює запис про покупку, обробляє оплату та надає доступ до повного аудіофайлу в обліковому записі користувача.

Взаємодія адміністратора із системою реалізована окремо. В адміністративній панелі адміністратор може додавати нові треки, редагувати інформацію про них, завантажувати демо-файли, повні аудіофайли, переглядати статистику продажів та керувати користувачами. Усі ці дії виконуються через запити до серверної частини, яка вносить відповідні зміни до бази даних.

Таким чином, моделювання структури та взаємодії об’єктів показує, що система складається з трьох основних частин: клієнтського інтерфейсу, серверної логіки та бази даних. Такий поділ забезпечує зручну організацію програмного забезпечення, спрощує підтримку системи та дозволяє здійснювати подальше розширення її функціональності.

## 3.3. Організаційна структура програмного забезпечення

Сутність організаційної структури полягає у способі поділу системи на частини, кожна з яких виконує власні функції та взаємодіє з іншими частинами. Розроблена система має логічне розділення інтерфейсу, бізнес-логіки сервера, бази даних та зовнішніх сервісів.

За реалізацію клієнтської частини відповідає React. Він відображає інтерфейс та забезпечує взаємодію користувача з основним функціоналом платформи. До клієнтської частини входять головна сторінка, каталог треків, сторінка окремого треку, аудіоплеєр, кошик, особистий кабінет, сторінки авторизації та реєстрації, а також адміністративна панель. Клієнтська частина системи надсилає запити до серверної частини, отримує відповіді у форматі JSON та виводить відповідні дані користувачу.

Серверна частина реалізована мовою програмування Python з використанням FastAPI. Саме ця частина обробляє запити від клієнтської частини, забезпечує авторизацію, розподіл ролей, роботу з кошиком, покупками, користувачами та адміністративними функціями. Також серверна частина виконує взаємодію з базою даних.

Дані, які використовуються у веборієнтованій системі, керуються за допомогою PostgreSQL. Серверна частина взаємодіє з базою даних через SQLAlchemy ORM. Цей інструмент дозволяє працювати з таблицями БД через моделі Python. В обраній предметній області база даних містить відомості про користувачів, музичні треки, кошик, покупки, придбані файли та службові дані.

REST API — це інструмент для обміну даними між клієнтською та серверною частинами. Для виконання HTTP-запитів використовується бібліотека Axios. За допомогою цього підходу можна розподілити відповідальність між frontend і backend та спростити підтримку системи.

Зовнішній сервіс Stripe виконує організаційну функцію в системі, оскільки забезпечує реалізацію процесу оплати музичного контенту в тестовому режимі. Після того як користувач оформлює покупку, він обирає спосіб оплати, а система перенаправляє його на зовнішній сервіс, до якого передаються дані про ціну та об’єкт покупки. Після успішного завершення процесу система отримує результат оплати та надає доступ до купленого продукту.

Загальна структура:

React frontend — інтерфейс користувача та адміністратора;

FastAPI backend — серверна логіка та обробка запитів;

PostgreSQL — збереження даних системи;

SQLAlchemy ORM — взаємодія backend-частини з базою даних;

REST API + JSON — обмін даними між клієнтом і сервером;

Axios — виконання HTTP-запитів на frontend-частині;

Stripe — зовнішній сервіс для оформлення оплати в тестовому режимі.

### 3.4 Компонентна структура системи

Програмне забезпечення веборієнтованої системи продажу музичного контенту використовує React як клієнтську частину та FastAPI як серверну. Щоб зробити масштабований, зрозумілий та швидкий вебзастосунок, необхідно використовувати компоненти.

React-компонент є окремою функціональною частиною, на якій будується принцип роботи цього фреймворку. Він має власну структуру, стилі та логіку. Підхід використання компонентів дозволяє не тільки розділити сторінки системи на менші частини, а й повторно використовувати окремі елементи та спростити підтримку коду. Приклади використання React-компонентів відображено в ***Додатку Ж***.

Серверна частина складається з файлів: main.py, models.py, schemas.py, auth.py та database.py. Файл database.py відповідає за підключення до PostgreSQL, створення сесій і базового класу для ORM-моделей.

Опис компонентів серверної частини:

- Основний файл у якому створюється FastAPI додаток, підключаються статичні файли, налаштовується CORS та описуються API-маршрути називається main.py. Через перечислені маршрути, реалізовується такі функції як створення покупка, перегляд придбаних композицій, керування користувачами, додавання треків до кошика, інтеграція зі Stripe, реєстрація й авторизація, завантаження файлів та перегляд треків.
- Файл auth.py реалізує механізм авторизація а саме: перевірку пароля, отримання поточного користувача, хешування паролів, створення JWT-токен, покупок та адміністративної панелі.
- Файл schemas.py виконує функцію опису структури даних, які надходять від клієнта і повертаються сервером. Тобто в ньому зберігаються схеми для реєстрації, авторизації, кошика, покупок і треків та придбаних композицій. За допомогою файлу schemas.py бекенд конролює корекність даних під час обміну між клієнтською частиною і сервером.
- Файл models.py містить опис БД: користувачів, треків, кошика, елементів кошика, покупок і придбаних треків. В самих моделях задано зв’яки між сутностями та обмеження для перевірки коректних даних.

В підсумку ми маємо, React-компонент забезпечує відображення інтерфейсу та взаємодію користувача із веборієнтованою системою, а backend-файли відповідають за збереження даних. Такий підхід робить структуру доодатка зручною та зрозумілою.

Приклад використання файлів в ***Додатку И***.

### 3.5 Вибір інструментарію

Для розробки веборієнтованої системи продажу музичного контенту важливо обрати інструменти, які забезпечать зручну розробку програмного забезпечення, а саме клієнтської та серверної частин, бази даних, обмін даними між компонентами системи, інтеграцію з платіжним сервісом, розробку дизайну та написання коду.

**Draw.io**

Перед тим як перейти до написання коду, необхідно описати діаграми, які дозволять повноцінно відобразити структуру та логіку роботи програмного забезпечення. Для виконання цього завдання було обрано Draw.io, оскільки цей інструмент є безкоштовним, має готові шаблони та необхідні позначення для створення діаграм у сфері інженерії програмного забезпечення.

**Figma**

Після визначення функціональних вимог програмного забезпечення необхідно створити макет клієнтської частини. Це відіграє важливу роль у розробці зручного, сучасного та адаптивного програмного продукту. Для цього було обрано Figma, оскільки цей інструмент широко використовується дизайнерами, має зручний інтерфейс, підтримує створення макетів, прототипів та анімацій. Також основні функції Figma доступні безкоштовно, що робить її зручним вибором для проєктування інтерфейсу вебзастосунку.

**Visual Studio Code**

Для написання коду необхідне середовище розробки, яке є зручним у використанні та підтримує потрібні мови програмування і технології. Visual Studio Code має простий інтерфейс, вбудований термінал, автодоповнення коду та підсвічування синтаксису, що допомагає краще орієнтуватися в коді. Також він має багато розширень, які спрощують процес розробки. У розробленій системі VS Code використовується для написання як серверної, так і клієнтської частини, що дозволяє працювати з проєктом в одному середовищі розробки.

**React**

Для написання клієнтської частини необхідно обрати бібліотеку для створення динамічного веб-інтерфейсу. Однією з найпопулярніших бібліотек, які використовуються в сучасних веб-застосунках, є React. Він дозволяє будувати сторінки з окремих компонентів і повторно використовувати їх у різних частинах коду. Це спрощує розробку та подальшу підтримку проєкту. Для написання інтерфейсу використовується JSX, який поєднує можливості JavaScript та HTML, завдяки чому код стає більш зрозумілим і зручним для підтримки.

**FastAPI**

Для створення серверної частини потрібен інструмент, який дає змогу обробляти запити від React. Саме тому було обрано FastAPI. Це сучасний фреймворк для швидкої розробки серверної частини мовою програмування Python. Він дозволяє створювати REST API для обробки запитів від клієнтської частини, має високу продуктивність, зручну структуру коду та автоматично формує документацію API, що спрощує тестування і підтримку backend-частини.

**PostgreSQL**

Система керування базами даних є важливою складовою для надійного зберігання даних. Для розробленої системи було обрано PostgreSQL, оскільки вона добре підходить для зберігання інформації про музичний контент, користувачів, кошик і покупки. PostgreSQL підтримує зв’язки між таблицями, обмеження цілісності даних і складні запити, що робить її стабільною основою для подальшого розширення системи.

**SQLAlchemy**

Для спрощення роботи з PostgreSQL було обрано SQLAlchemy. Це інструмент для роботи з базою даних у Python, який дозволяє описувати таблиці у вигляді моделей. Завдяки SQLAlchemy легше створювати, змінювати й

підтримувати структуру БД, а також виконувати запити до бази даних із серверної частини застосунку.

**REST API**

Для забезпечення зручної взаємодії між клієнтською та серверною частинами необхідний підхід, який дозволяє надсилати запити до сервера та отримувати відповіді у структурованому вигляді. Для цього REST API було обрано як архітектурний підхід для обміну даними між frontend і backend.

**Stripe**

Для розроблення програмного забезпечення веборієнтованої системи продажу музичного контенту необхідно реалізувати функцію оплати, оскільки вона є однією з основних функцій системи. У межах проєкту було використано зовнішній сервіс Stripe, який дозволяє організувати процес оформлення покупки, створювати платежі та тестувати оплату в безпечному режимі. Це дає змогу в майбутньому підключити систему до реальної платіжної системи.

**Axios**

Для реалізації HTTP-запитів між React і FastAPI було обрано бібліотеку Axios. Вона дозволяє зручно надсилати запити до REST API, отримувати дані, а також передавати токен для авторизації користувача. Такий підхід допомагає зробити взаємодію між frontend і backend простішою, зручнішою для підтримки та зрозумілішою. У підсумку обрано набір інструментів, які доповнюють один одного та забезпечують простоту, швидкість, зрозумілість і сучасний підхід до розробки програмного забезпечення.

### 3.6 Алгоритмізація та програмування модулів

Процес створення послідовності дій для розв’язання задачі називається алгоритмізацією. У розробленій системі реалізовано модулі авторизації, маршрутизації, каталогу, перегляду окремого музичного контенту, кошика, оплати та адміністративної панелі. Описані модулі дають змогу зрозуміти алгоритм дій основних функцій застосунку. Для реалізації клієнтської частини

використовується Axios, який налаштований у вигляді окремого API-модуля. Це означає, що до запитів автоматично додається токен авторизації, що дозволяє звертатися до захищених маршрутів серверної частини.

Модуль авторизації забезпечує функцію входу користувача до системи. Користувач вводить email і пароль, після чого дані передаються на сервер через метод AuthService.login. Якщо авторизація успішна, токен, статус авторизації та дані користувача зберігаються в localStorage. Після цього система перевіряє роль і перенаправляє користувача до головного меню.

Маршрутизація поділена на публічні та приватні маршрути. До приватних належать кошик, придбані треки, сторінка успішної оплати та адміністративна панель. Алгоритм роботи каталогу полягає в отриманні списку треків із сервера та їх відображенні у вигляді карток. Також реалізовано пошук і фільтрацію. Для відтворення демоверсій використовується стан playingTrackId, який дозволяє відтворювати лише один трек одночасно. Після завершення відтворення активний трек скидається.

Модуль перегляду треку відповідає за ознайомлення користувача з розширеною інформацією. Ідентифікатор треку береться з параметрів маршруту, після чого надсилається запит до сервера. Після отримання відповіді відображаються розширені дані, можливість прослуховування треку та додавання музичного контенту в кошик.

Модуль кошика відображає вміст кошика з можливістю додавання та видалення треків. CartService надсилає запити до серверної частини. Після додавання треку користувач отримує повідомлення про успішну операцію.

Оплата реалізовується через створення сесії. Клієнтська частина надсилає запит на backend, передаючи токен авторизації в заголовку. Після успішної оплати треки стають доступними користувачу на сторінці придбаних треків. Робота з придбаними треками виконується через PurchaseService.

Адміністративний модуль призначений для керування контентом. Після успішного входу адміністратор може додавати, редагувати та видаляти треки, а також завантажувати треки разом із файлами. Для цього використовується

TrackService, який містить методи створення, оновлення, видалення та завантаження.

# 4. РЕКОМЕНДАЦІЇ ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

## 4.1. Тестування системи

Тестування розробленого програмного забезпечення є важливим етапом для розробки, саме цей етап дозволяє перевірити правильність роботи, виявити помилки та розуміння що програма функціонує згідно поставленим вимогам.

Для перевірки розробленої веборієнтованої системи продажу музичного контенту було обрано метод функціонального тестування, цей метод передбачає перевірку окремих функцій без детального аналізу реалізації коду. Основна увага при тестуванні приділяється на те щоб функції користувача працювали коректно та були зручні для використання.

**Перевірені основні функції можливості системи:**

- реєстрація нового користувача;
- вхід до системи;
- відкриття каталогу музичних треків;
- пошук треків;
- фільтрація та сортування композицій;
- відтворення демоверсій треків;
- перегляд сторінки окремого треку;
- додавання треку до кошика;
- оформлення покупки через платіжний сервіс;
- відображення придбаних треків;
- робота адміністративної панелі;
- додавання, редагування та видалення треків адміністратором;
- завантаження файлів треків;

- адаптивність інтерфейсу на різних розмірах екрана.

**Перевірка реєстрацій та входу:**

Перевірка полягає у введені коректних і некоректних даних користувача (*Рис. 12, 13, 14, 15*), якщо користувач вводить пароль та email, то система зберегти токен доступу та перенаправляла користувча відповідно до його ролі. У випадку якщо дані були відсутні результатів програмного забезпечення, виводлась повідомлення про помилку (*Рис.16*).

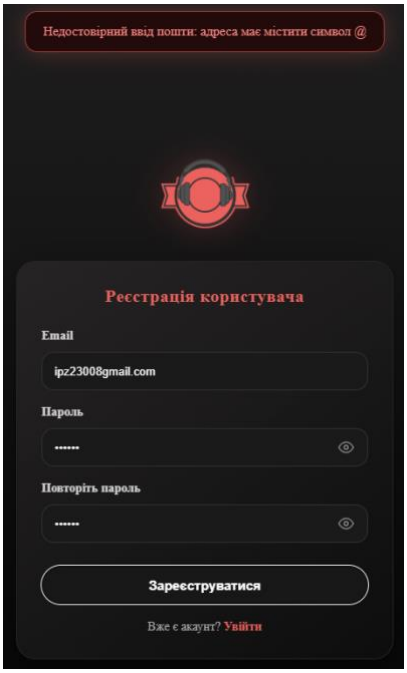



*Рис.11 Помилка входу, користувач не ввів пошту коректно.*

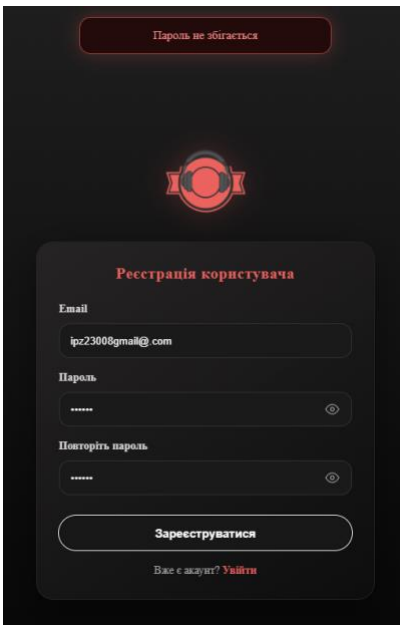


*Рис.12 Помилка вводу повторного паролю реєстрації.*

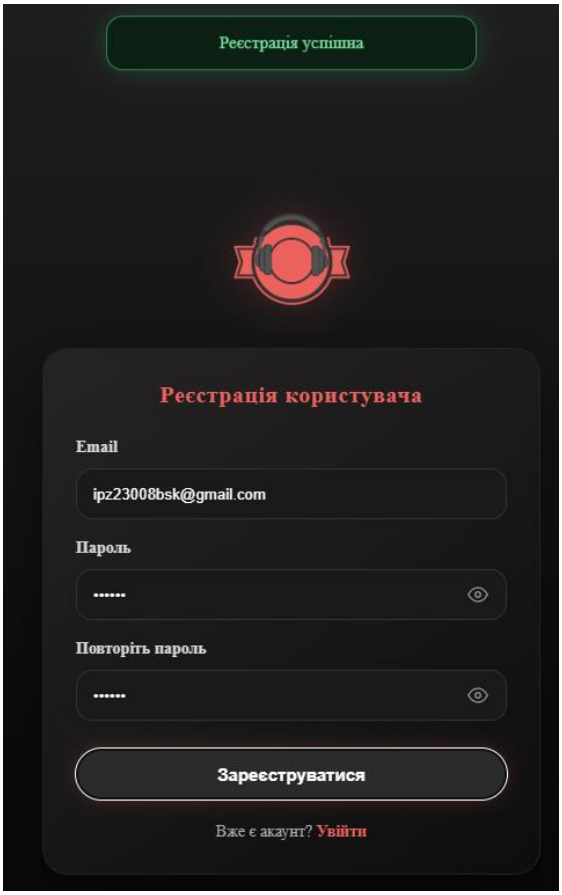



*Рис.13 Успішна реєстрація.*

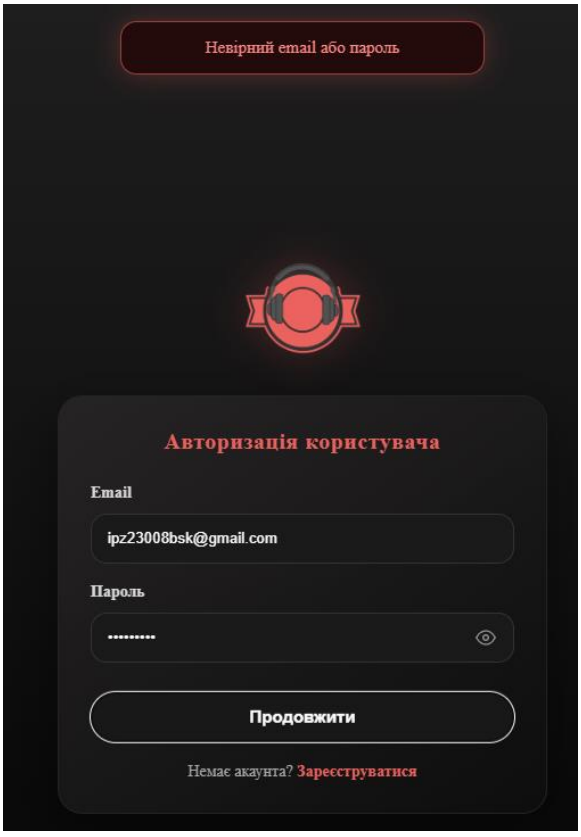


*Рис.14 Успішна реєстрація*

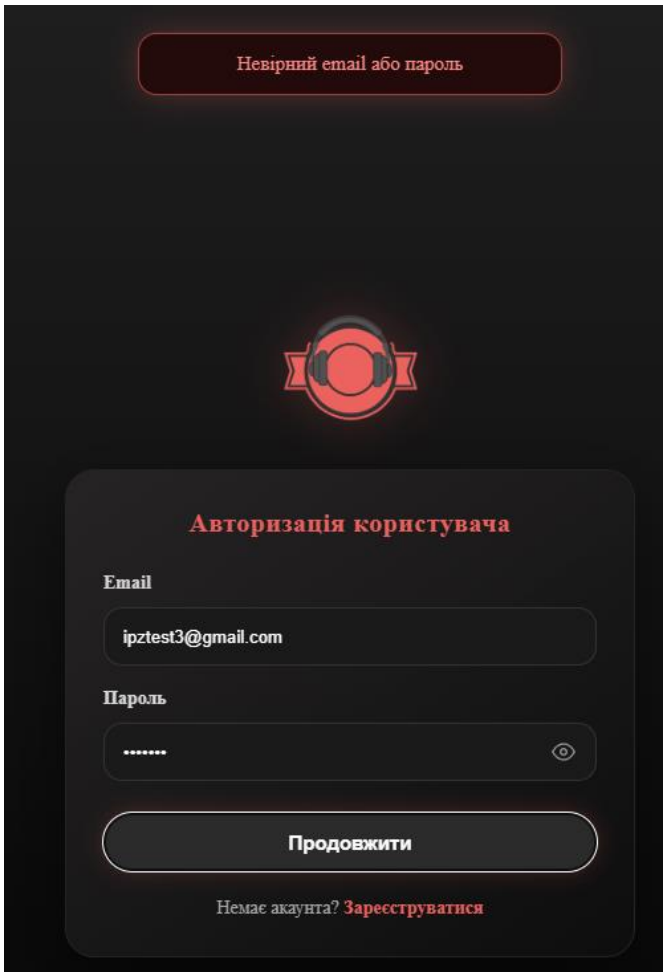



*Рис.15 Помилка перевірки даних користувача.*

**Тестування каталогу:**

Перевірка передбачає завантаження списку композицій, відображення карток, роботи рядка та фільтрів (*Рис. 16).* Було перевірено,при введенні пошукового запиту список треків оновлюється відповідно до введених даних (*Рис. 17*), а при відсутності результатів система показує повідомлення про те, що треки не знайдені (*Рис. 18*).


*Рис.16 Відображення роботи фільрів.*

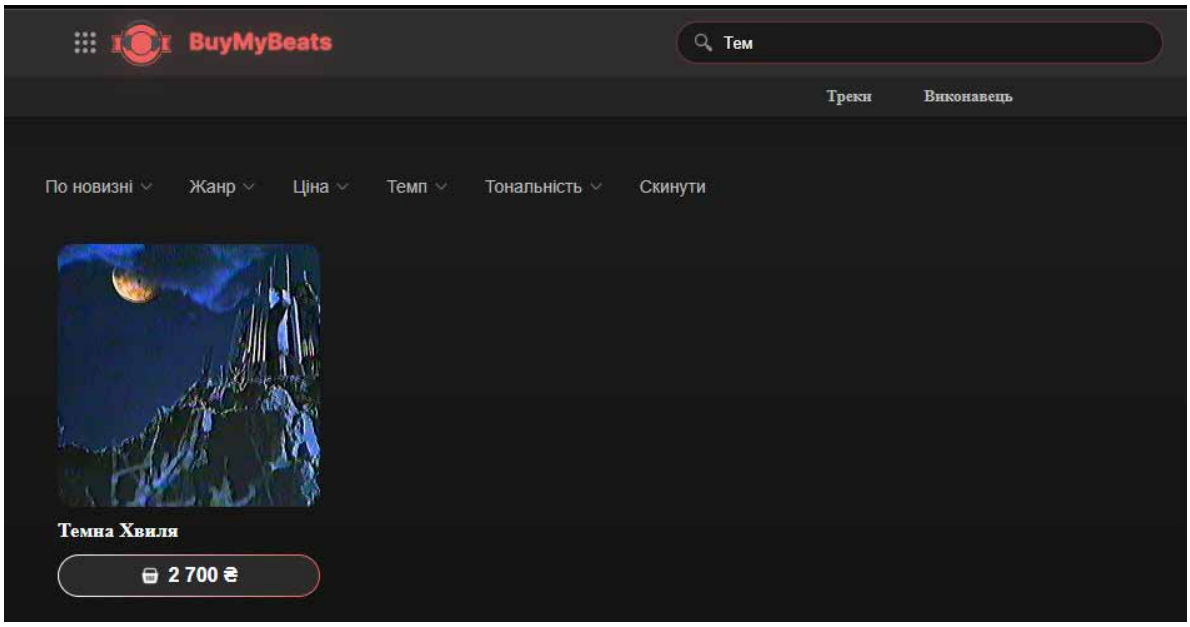



*Рис.17 Відображення введених даних в рядок.*

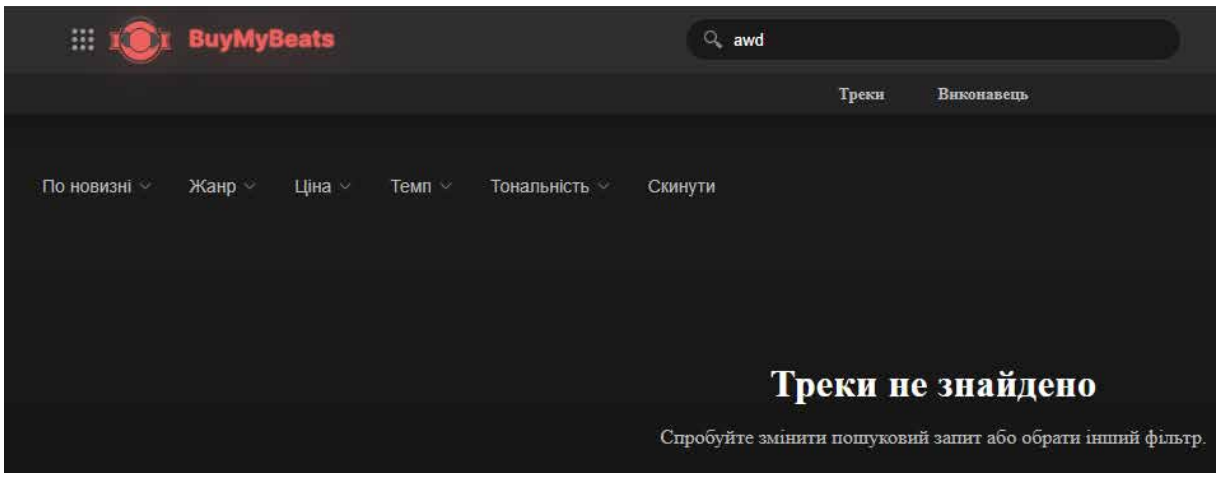



*Рис.18 Відсутні результати пошуку.*

**Тестування кошика:**

Для цього тесту включається перевірка додавання треку (*Рис. 19*), видалення треку (*Рис. 20*) та правильного відображення загальної суми замовлення (*Рис. 21*). Після додавання композиції до кошика перевірялося, чи зберігається вона для поточного користувача та чи не виникають помилки під час повторних дій (*Рис. 22*).

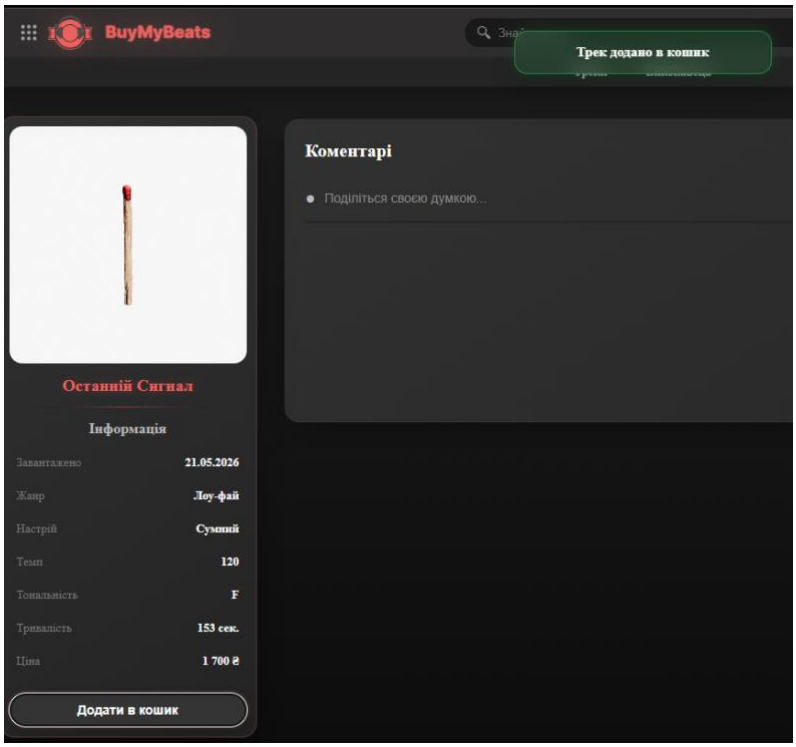



*Рис.19 Успішний доданий трек.*

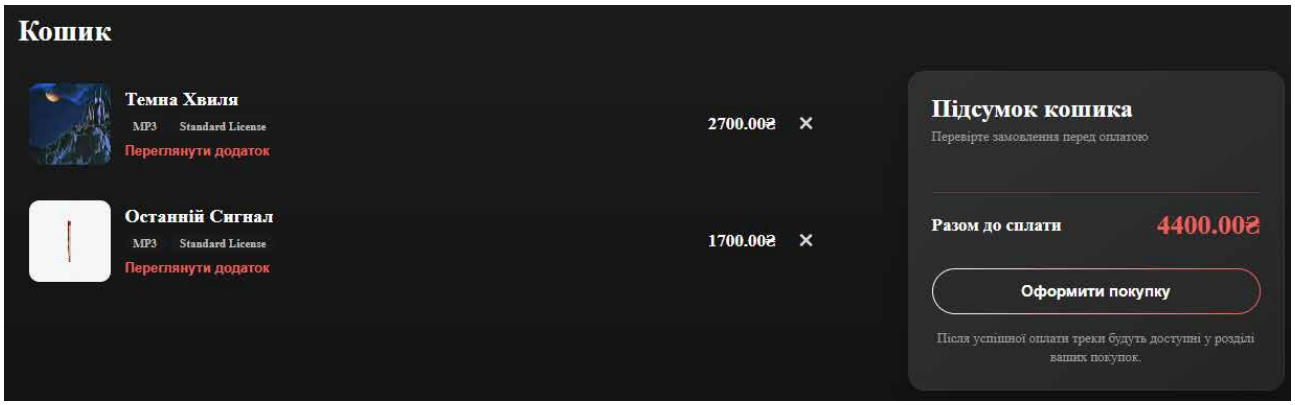



*Рис.20 Стан до натискання кнопки видалення.*

При натискання на хрестик картки товару, картка зникає, змінюючи сумму.

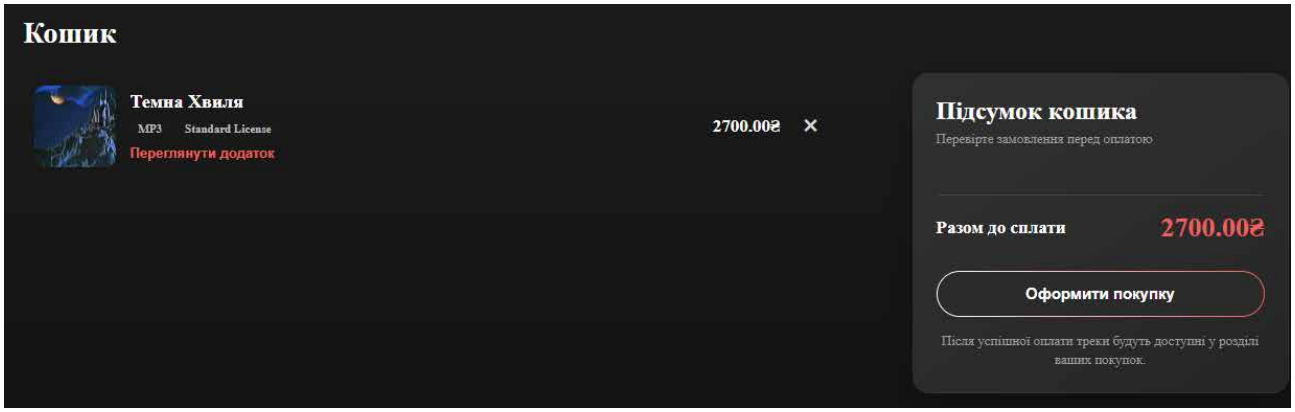



*Рис.21 Видалений трек та зміна загальної суми.*

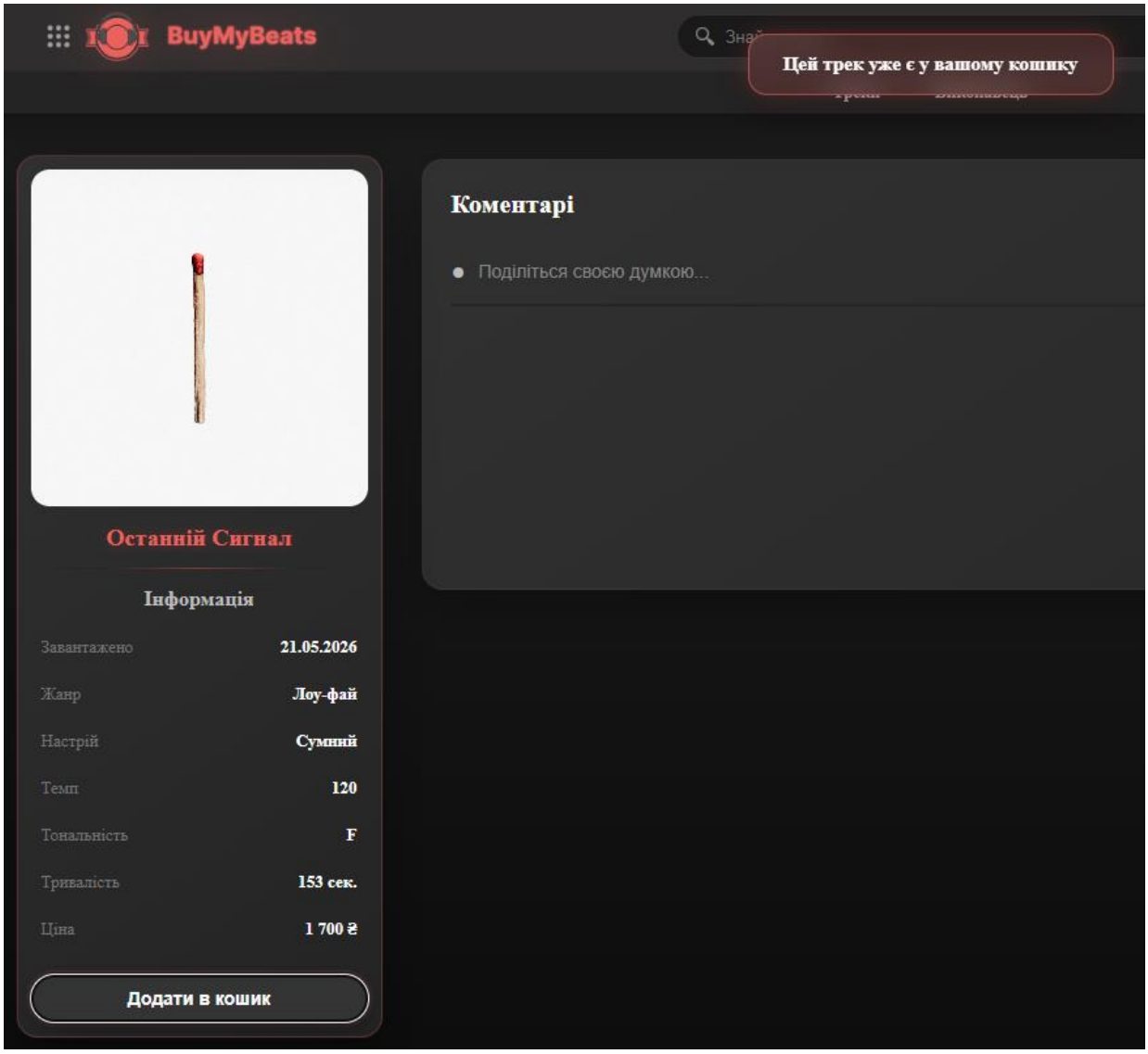



*Рис.20 Відображення помилки при спроби додати вже доданий трек.*

**Модуль покупки:**

Перевірявся через тестовий режим платіжного сервісу Stripe. Було протестовано створення checkout-сесії (*Рис.20*), перехід користувача на сторінку оплати (*Рис.21*), успішне завершення платежу та подальше відображення придбаного треку у відповідному розділі системи (*Рис.22*).

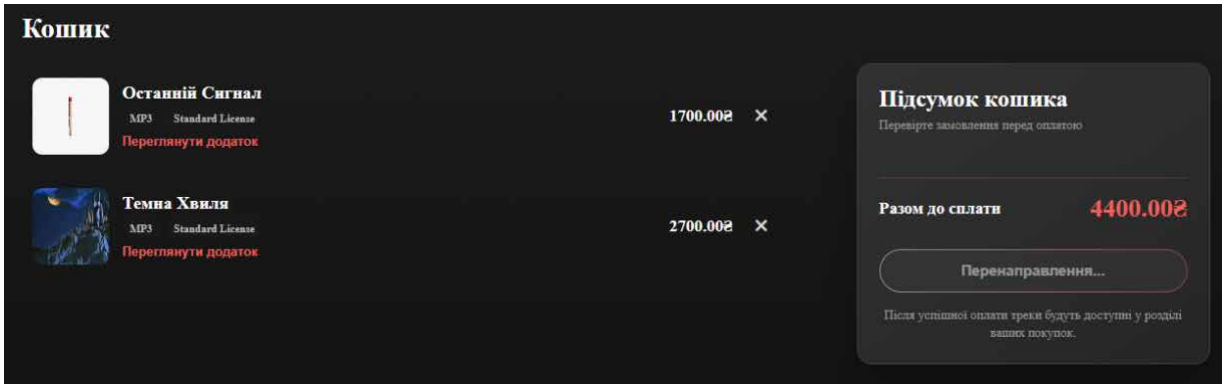



*Рис.20 Створення CheclOut-сесія.*

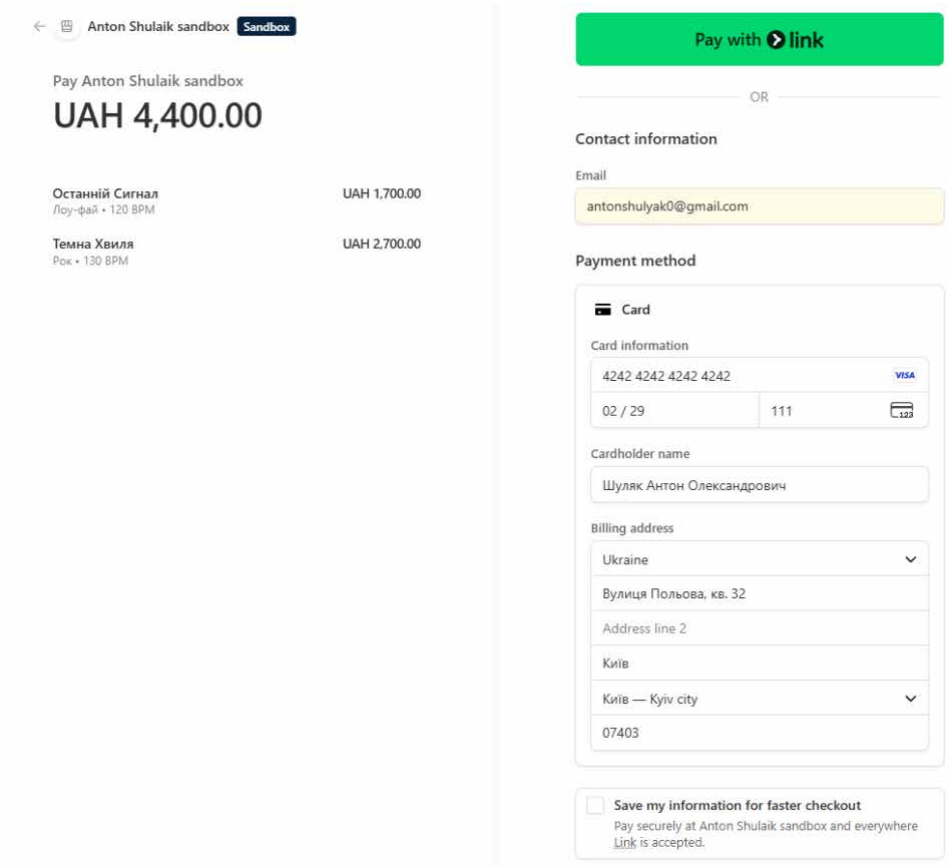



*Рис.21 Перехід користувача на сторінку оплати.*

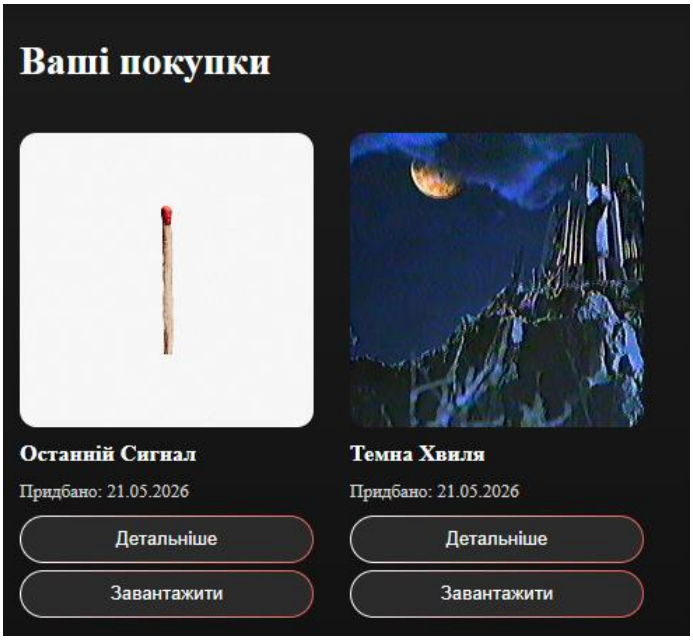



*Рис.22 Відображення треку в розділі "Мої покупки".*

**Адміністративна панель:**

Перевірено доступ до неї, випадок коли доступ має лише користувач із роллю адміністратора (*Рис.23*). Також перевірялись операції додавання нового треку та завантаження файлів (*Рис.24*), редагування існуючої композиції (*Рис.25*), видалення треку (*Рис.26*). Після кожної операції перевірялося оновлення списку треків у системі.

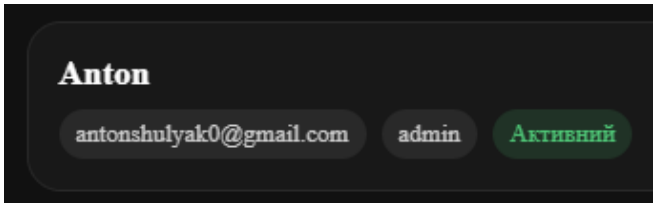



*Рис.23 Відображення ролі в адміністративній панелі.*

Коли користувач без прав користування адмін-панелі вводить адресу сторінкі, нічого не відбувається, перехід не здійснюється, він залишається на тій сторінці, на якій намагався перейти до адмін-панелі.

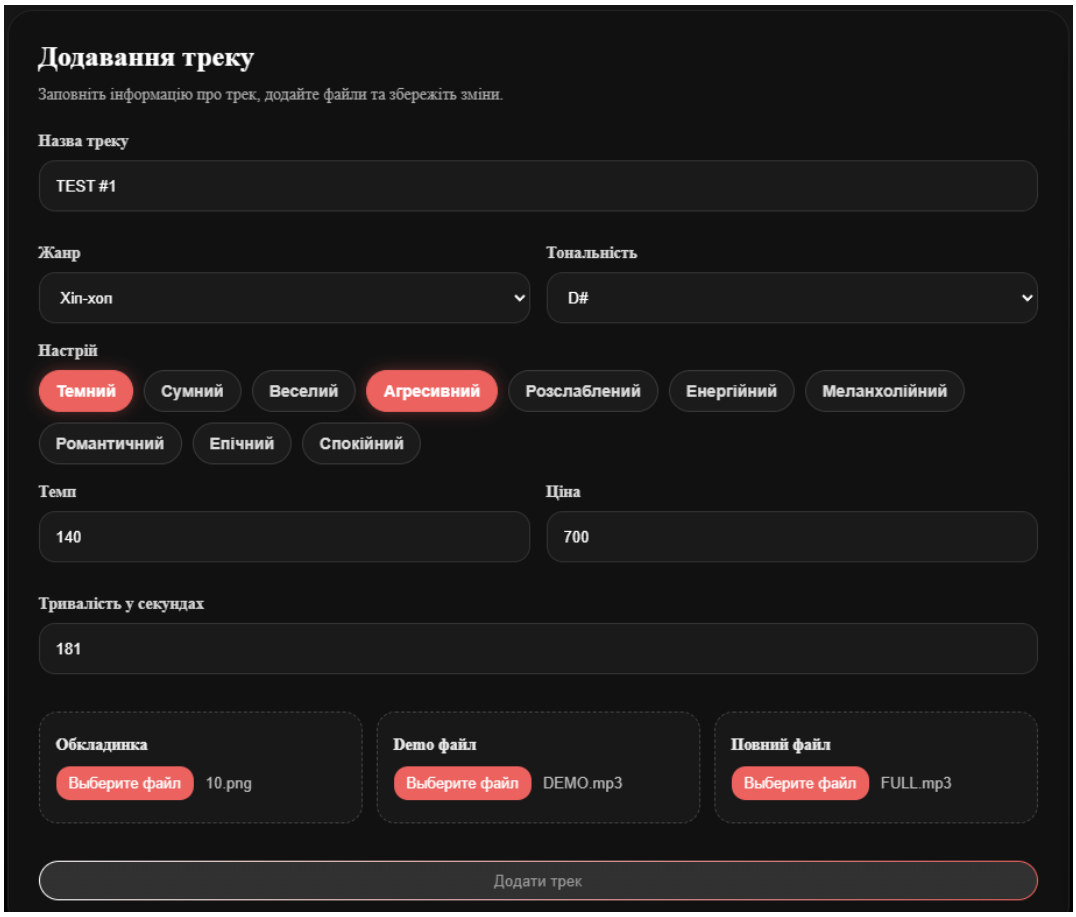



*Рис.24 Відображення коректного вводу даних.*

Адміністратор має поради щодо введення даних, коли він успішно вводить ці данні, з’являється новий трек в адмін панелі та каталозі (*Рис.24.1*).

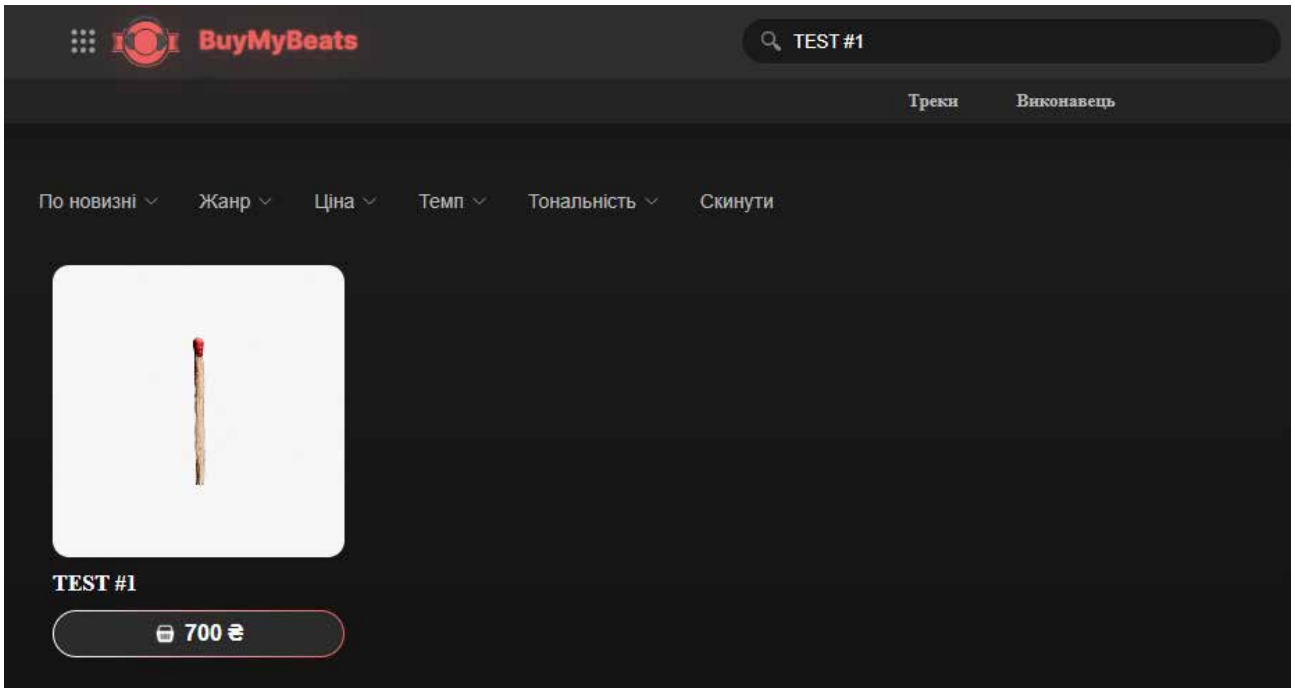



*Рис.24.1 Поява трека в каталозі.*

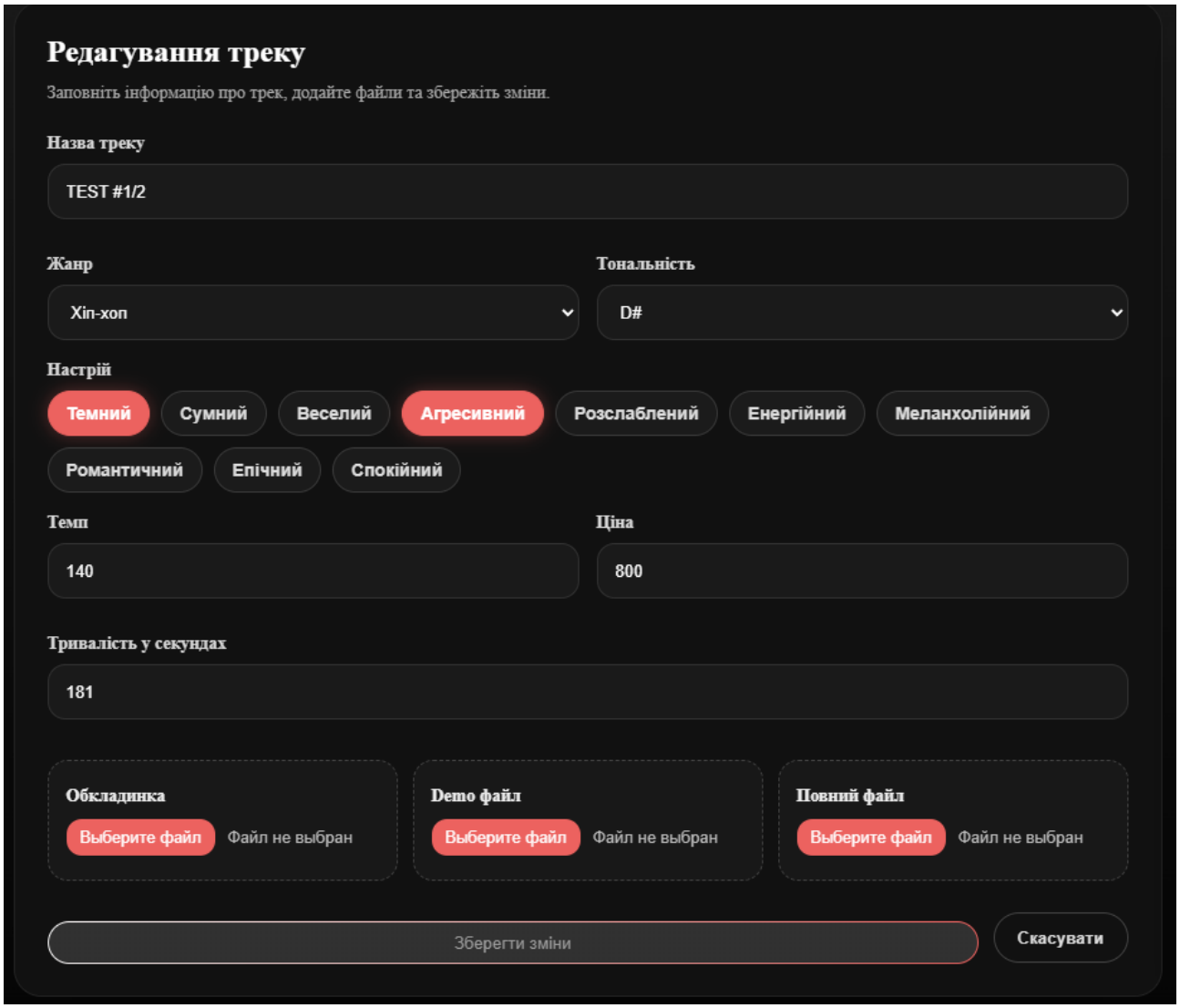



*Рис.25 Процес зміни інформації.*

При натисканні на кнопку редагувати, яка знаходиться на картці треку в адмін-панелі, заповнює данні які можна змінити та зберегти зміни (*Рис.25.1*).


*Рис.25.1 Результат зміни інформації.*

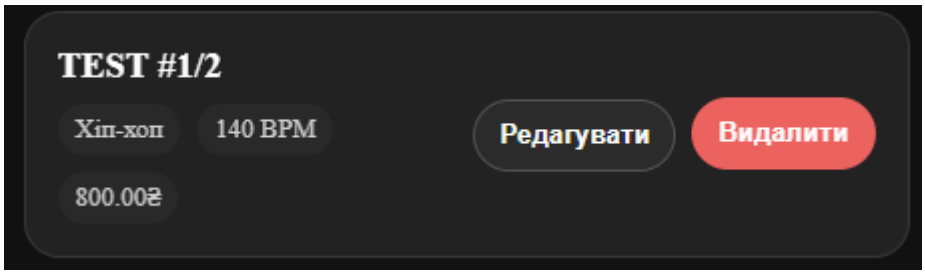



*Рис.26 Результат зміни інформації.*

При натискання кнопки видалення на картці,  трек видаляється і стає недоступний (*Рис. 26.1*).

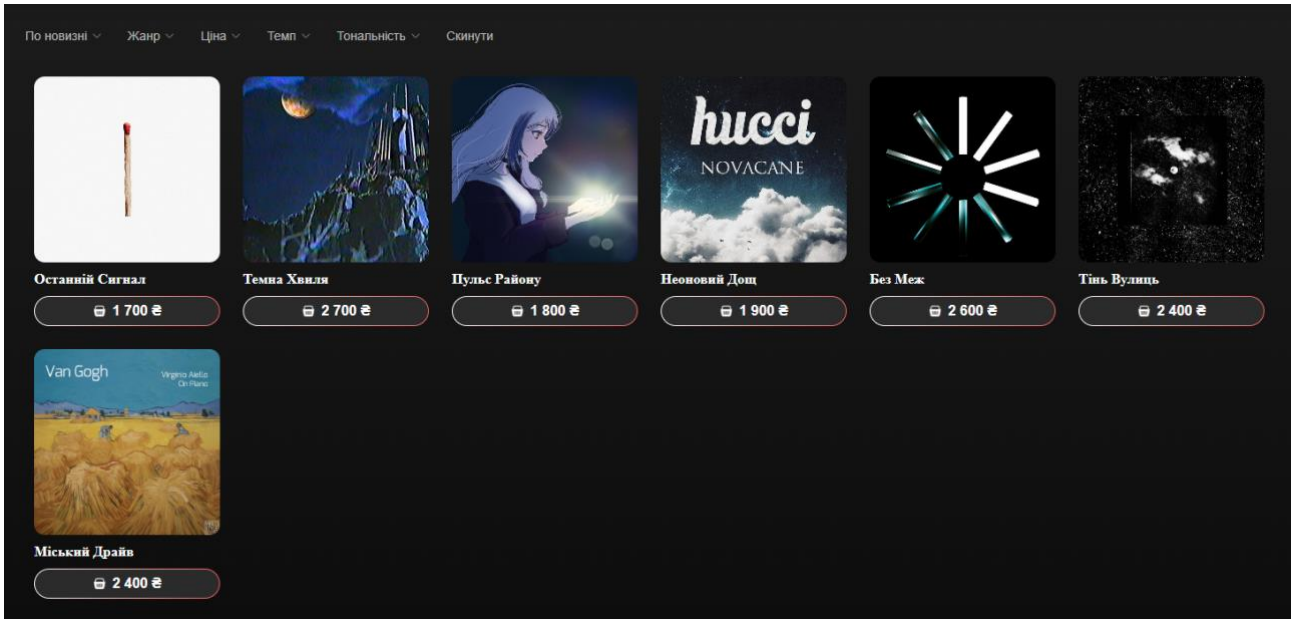



*Рис.26.1 Результат видалення трека.*

**Адаптивності інтерфейсу:**

Сторінки системи переглядалися на різних розмірах екрана, зокрема на мобільному форматі. Основна увага приділялась правильному розташуванню елементів, зручності навігації, відображенню карток треків (*Рис. 27*), кошика (*Рис. 28*), сторінки входу (*Рис. 28*) та адміністративної панелі (*Рис. 29*).

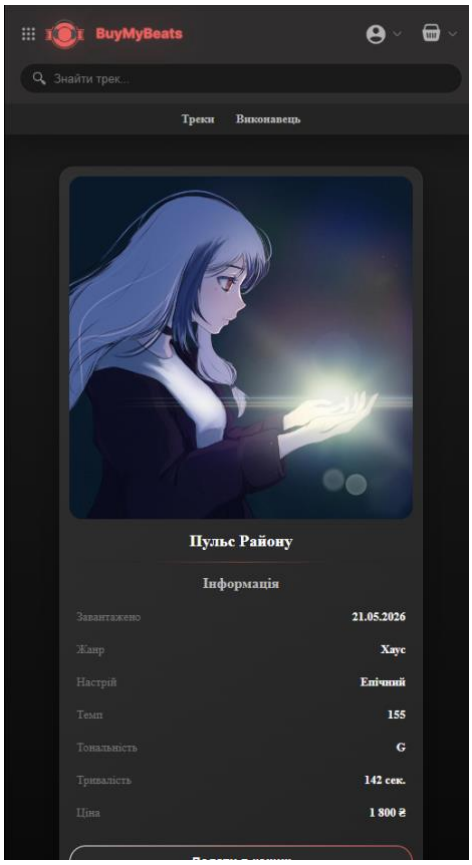



*Рис.27 Адаптивний інтерфейс катки конкретного трека.*

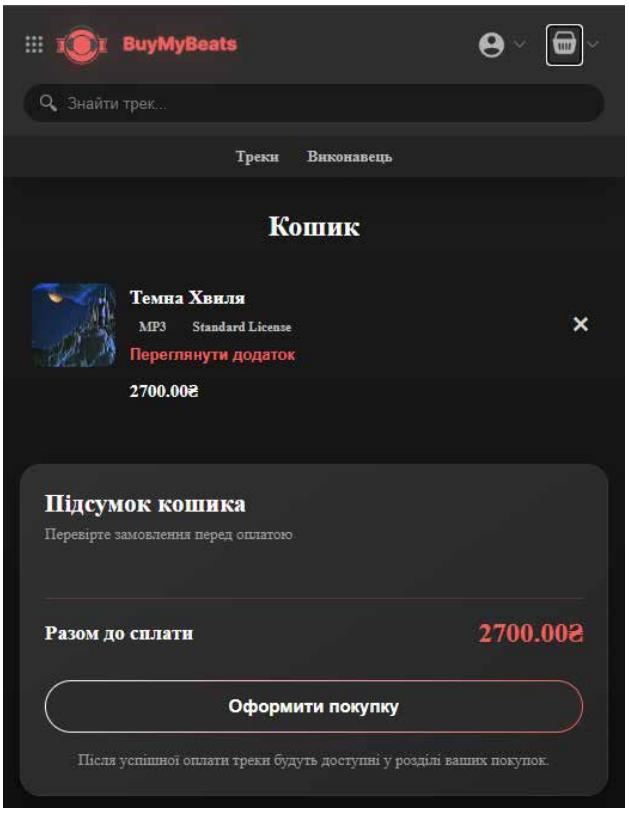



*Рис.28 Адаптивний інтерфейс кошика.*

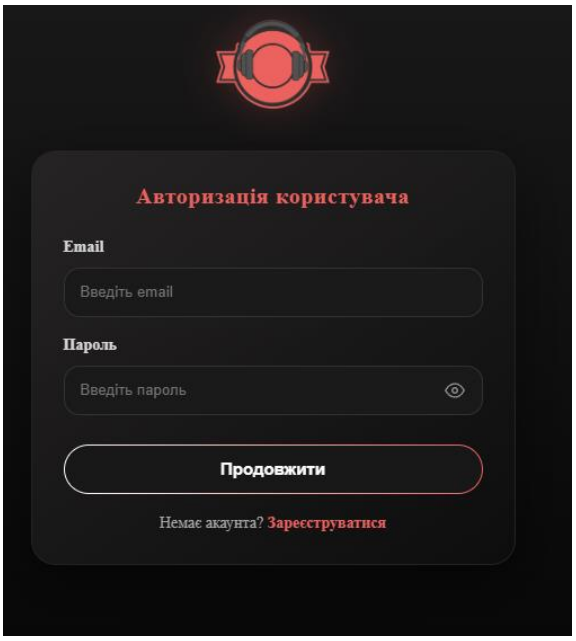



*Рис.28 Адаптивний інтерфейс сторінки входу.*

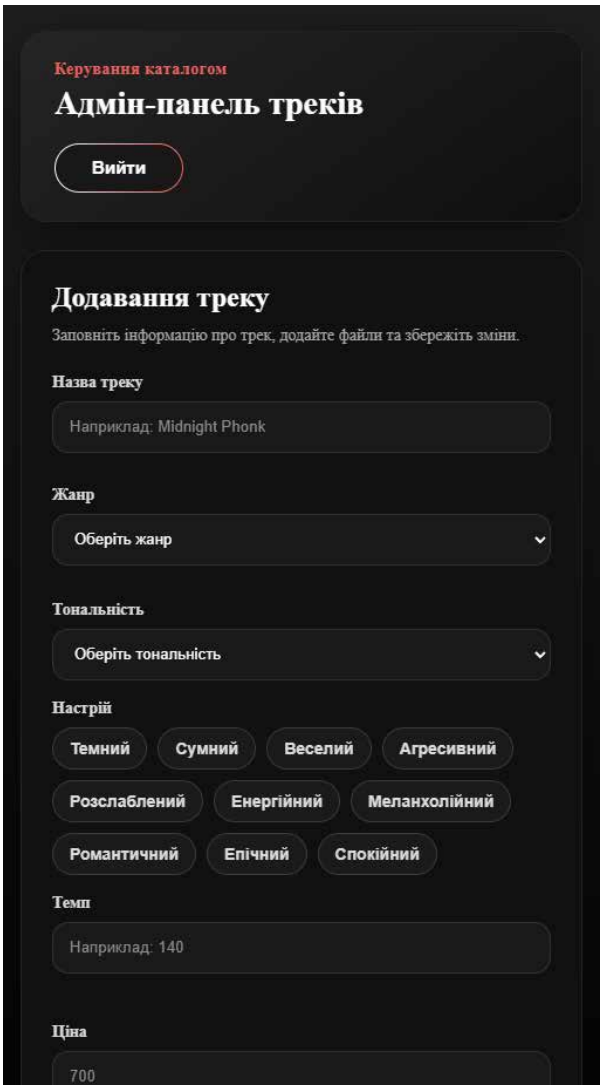



*Рис.28 Адаптивний інтерфейс адміністративної панелі.*

Виходячи із тестування було перевірено, що основні модулі системи працюють коректно, взаємодія між frontend і backend виконується належним чином, а користувач може пройти повний сценарій роботи: від реєстрації та перегляду каталогу до покупки музичного контенту. Адміністратор, має можливість керувати треками через окрему адміністративну панель. Проведене тестування підтвердило працездатність і відповідність системи основним функціональним вимогам.

## 4.2. Вимоги до апаратного та програмного забезпечення

Для розгортання та коректної роботи веборієнтованої системи продажу музичного контенту необхідно враховувати вимоги як до апаратного, так і до програмного забезпечення серверної та клієнтської частин. Оскільки система побудована на клієнт-серверній архітектурі, вимоги розділяються на дві категорії: вимоги до серверного середовища та вимоги до пристрою кінцевого користувача.

### Вимоги до апаратного забезпечення серверної частини

Серверна частина системи розроблена з використанням FastAPI та Python і потребує відповідних апаратних ресурсів для стабільної обробки запитів, роботи з базою даних PostgreSQL та зберігання аудіо-файлів.

Мінімальні вимоги до апаратного забезпечення сервера:

- процесор із тактовою частотою не нижче 1.8 ГГц (рекомендується 2-ядерний або вище);
- оперативна пам'ять - не менше 2 ГБ (рекомендується 4 ГБ і більше);
- дисковий простір - не менше 10 ГБ для системи, бази даних та зберігання аудіо-контенту (рекомендується виділений розділ для

медіафайлів обсягом від 50 ГБ, залежно від кількості треків у каталозі);

- стабільне мережеве підключення зі швидкістю не нижче 10 Мбіт/с.

Для розгортання в хмарному середовищі допускається використання VPS або хмарних платформ (наприклад, AWS, DigitalOcean, Heroku), що задовольняють наведені вище вимоги.

**Вимоги до програмного забезпечення серверної частини**

Для розгортання серверної частини системи необхідне таке програмне забезпечення:

- операційна система - Linux (рекомендується Ubuntu 20.04 LTS або новіша версія), також підтримується Windows 10/11 та macOS для локального розгортання в цілях розробки;
- Python версії 3.10 і вище - необхідний для запуску серверної логіки на FastAPI;
- pip - менеджер пакетів Python для встановлення залежностей;
- PostgreSQL версії 14 і вище - система управління базами даних, яка використовується для зберігання всіх даних системи;
- Git - система контролю версій для отримання вихідного коду з репозиторію.

Для коректної роботи серверної частини необхідно встановити Python-бібліотеки, перелічені у файлі requirements.txt інсталяційного пакета, зокрема: FastAPI, Uvicorn, SQLAlchemy, Psycopg2-binary, Pydantic, Python-Jose, Passlib, Stripe, Boto3, Pillow та інші залежності.

**Вимоги до апаратного та програмного забезпечення клієнтської частини**

Клієнтська частина системи реалізована у вигляді SPA на React і запускається безпосередньо у браузері користувача, що суттєво знижує вимоги до апаратного забезпечення кінцевого пристрою.

Мінімальні вимоги до апаратного забезпечення клієнта:

- процесор із тактовою частотою не нижче 1.0 ГГц;
- оперативна пам'ять - не менше 1 ГБ;
- будь-який пристрій з підтримкою сучасного браузера та відтворення аудіо (комп'ютер, ноутбук, планшет, смартфон);
- стабільне підключення до мережі Інтернет зі швидкістю не нижче 5 Мбіт/с для комфортного попереднього прослуховування демоверсій треків.

Вимоги до програмного забезпечення клієнта зводяться до наявності одного з підтримуваних браузерів:

- Google Chrome версії 90 і вище;
- Mozilla Firefox версії 88 і вище;
- Microsoft Edge версії 90 і вище;
- Safari версії 14 і вище.

Використання застарілих браузерів або браузерів без підтримки Web Audio API може призвести до некоректної роботи аудіоплеєра. Встановлення додаткового програмного забезпечення на стороні клієнта не вимагається.

Таким чином, система є доступною для широкого кола користувачів без необхідності встановлення додаткового ПЗ на клієнтський пристрій, що є однією з ключових переваг веборієнтованого підходу до розробки.

### 4.3 Склад інсталяційного пакета

Інсталяційний пакет веборієнтованої системи продажу музичного контенту містить усі необхідні компоненти для розгортання та запуску системи в робочому середовищі. Пакет складається з вихідного коду клієнтської та серверної частин, файлів конфігурації, залежностей та супровідної документації. Структура інсталяційного пакета програмного забезпечення подано на ***Рис. 29***

**Структура інсталяційного пакета програмного забезпечення**

```
├── MUSIC_BACKEND/
│   ├── main.py
│   ├── models.py
│   ├── schemas.py
│   ├── auth.py
│   ├── database.py
│   ├── requirements.txt
│   └── uploads/
│       ├── covers/
│       ├── demo/
│       └── full/
├── music_frontend/
│   ├── public/
│   ├── src/
│   │   ├── API/
│   │   │   ├── api.js
│   │   │   ├── AuthService.js
│   │   │   ├── TrackService.js
│   │   │   ├── CartService.js
│   │   │   ├── PurchaseService.js
│   │   │   ├── PaymentService.js
│   │   │   ├── AdminUserService.js
│   │   │   └── AdminStatsService.js
│   │   ├── Components/
│   │   │   ├── UI/
│   │   │   │   ├── button/
│   │   │   │   ├── input/
│   │   │   │   ├── loader/
│   │   │   │   ├── modal/
│   │   │   │   ├── navbar/
│   │   │   │   └── pagination/
│   │   │   ├── filter/
│   │   │   ├── AppRouter.jsx
│   │   │   ├── PostForm.jsx
│   │   │   ├── PostItem.jsx
│   │   │   └── PostList.jsx
│   │   ├── pages/
│   │   │   ├── About.jsx
│   │   │   ├── AdminTracks.jsx
│   │   │   ├── Cart.jsx
│   │   │   ├── Error.jsx
│   │   │   ├── Login.jsx
│   │   │   ├── MyTracks.jsx
│   │   │   ├── PaymentSuccess.jsx
│   │   │   ├── PostIdPage.jsx
│   │   │   ├── Posts.jsx
│   │   │   ├── PostsAdmin.jsx
│   │   │   └── Register.jsx
│   │   ├── hooks/
│   │   │   ├── useFetching.js
│   │   │   ├── useObserver.js
│   │   │   └── usePosts.js
│   │   ├── context/
│   │   │   └── index.js
│   │   ├── router/
│   │   │   └── index.js
│   │   ├── utils/
│   │   │   ├── pages.js
│   │   │   └── trackTranslations.js
│   │   ├── image/
│   │   ├── styles/
│   │   │   ├── AdminTracks.css
│   │   │   ├── App.css
│   │   │   ├── Cart.css
│   │   │   ├── Login.css
│   │   │   ├── MyTracks.css
│   │   │   ├── PostList.css
│   │   │   ├── Posts.css
│   │   │   └── TrackPage.css
│   │   ├── App.js
│   │   └── index.js
│   ├── package.json
│   └── package-lock.json
└── README.md
```

*Рис. 29 Структура інсталяційного пакета програмного забезпечення*

**Опис:**

Каталог MUSIC_BACKEND/ містить серверну частину системи, реалізовану мовою Python з використанням FastAPI. До нього входять:

main.py – основний файл серверного застосунку, в якому описано API-маршрути, налаштовано CORS, підключено статичні файли, реалізовано обробку запитів, роботу з треками, кошиком, покупками, адміністративними функціями та інтеграцію з платіжним сервісом Stripe;

models.py – файл ORM-моделей SQLAlchemy, який описує структуру таблиць бази даних. У ньому визначено сутності користувачів, музичних треків, кошика, елементів кошика, покупок та придбаних треків;

schemas.py - файл Pydantic-схем, які використовуються для валідації вхідних даних і формування відповідей API. Схеми визначають формат даних для реєстрації, авторизації, треків, кошика, покупок та інших об’єктів системи;

auth.py – модуль авторизації, який відповідає за хешування паролів, перевірку облікових даних користувача, генерацію та перевірку JWT-токенів, а також отримання поточного користувача із захищених запитів;

database.py – файл налаштування підключення до бази даних PostgreSQL. У ньому створюються engine, SessionLocal та базовий клас Base, який використовується для опису ORM-моделей;

uploads/ – каталог для зберігання файлів, які завантажуються через адміністративну частину системи. У ньому зберігаються обкладинки треків, demo-аудіофайли та повні версії музичних файлів;

venv/ – віртуальне середовище Python, у якому зберігаються встановлені залежності серверної частини;

everything.txt – допоміжний файл проєкту, який може використовуватися для збереження проміжної інформації, нотаток або резервного текстового вмісту під час розробки.

Клієнтська частина системи розміщена у каталозі src/ та реалізована з використанням бібліотеки React. Вона відповідає за відображення інтерфейсу користувача, маршрутизацію між сторінками, взаємодію з backend-частиною та обробку дій користувача.

**До структури клієнтської частини входять:**

src/API/ – каталог сервісів для взаємодії з серверною частиною через Axios. У ньому розміщені файли api.js, AuthService.js, TrackService.js, CartService.js, PurchaseService.js, PaymentService.js, AdminUserService.js та AdminStatsService.js. Вони відповідають за надсилання HTTP-запитів до API, авторизацію, роботу з треками, кошиком, покупками, оплатою та адміністративною статистикою;

src/Components/ – каталог багаторазових React-компонентів, які використовуються на різних сторінках застосунку. До нього входять компоненти списку треків, картки треку, форми додавання даних, маршрутизатора, а також окремі UI-компоненти;

src/Components/UI/ – каталог базових елементів інтерфейсу користувача. У ньому розміщено компоненти кнопок, полів введення, завантажувача, модальних вікон, навігаційної панелі та пагінації;

src/Components/filter/ – каталог компонентів фільтрації, які використовуються для пошуку та сортування музичного контенту в каталозі;

src/pages/ – каталог сторінок застосунку. У ньому розміщено сторінки каталогу треків, сторінки окремого треку, кошика, придбаних треків, авторизації,

реєстрації, адміністративної панелі, сторінки успішної оплати, інформаційної сторінки та сторінки помилки;

src/hooks/ – каталог користувацьких React-хуків. Вони використовуються для спрощення роботи із завантаженням даних, фільтрацією, пагінацією та відстеженням елементів на сторінці;

src/context/ – каталог для зберігання глобального стану застосунку. У ньому визначено контексти, які використовуються для авторизації користувача та передачі спільних даних між компонентами;

src/router/ – каталог налаштування маршрутів клієнтської частини. У ньому описано доступні маршрути для звичайного користувача, адміністратора та неавторизованого відвідувача;

src/utils/ – каталог допоміжних файлів. У ньому розміщено функції для розрахунку сторінок пагінації та файл trackTranslations.js, який використовується для перекладу характеристик треків українською мовою;

src/styles/ – каталог CSS-файлів, які відповідають за оформлення сторінок застосунку. У ньому зберігаються стилі для каталогу, сторінки треку, кошика, сторінки придбаних треків, авторизації, адміністративної панелі та загального вигляду застосунку;

App.js – головний компонент клієнтської частини, який об’єднує основні елементи застосунку, підключає маршрутизацію та глобальний контекст;

index.js – вхідний файл React-застосунку, який відповідає за рендеринг головного компонента App у DOM.

Окремо у frontend-частині використовуються конфігураційні файли package.json та package-lock.json. Файл package.json містить перелік залежностей, необхідних для роботи React-застосунку, а також npm-скрипти для запуску, розробки та збирання клієнтської частини. Файл package-lock.json фіксує точні

версії встановлених пакетів, що забезпечує стабільність роботи проєкту на різних пристроях.

Для коректної роботи системи необхідно налаштувати підключення до бази даних PostgreSQL, секретний ключ для JWT-токенів, параметри CORS, а також ключі платіжного сервісу Stripe. Ці параметри можуть зберігатися у конфігураційних файлах або задаватися безпосередньо в середовищі розробки. Основними налаштуваннями є DATABASE_URL для підключення до бази даних, SECRET_KEY для генерації токенів, STRIPE_SECRET_KEY для роботи з оплатою та FRONTEND_URL для коректного перенаправлення користувача після виконання платежу.

**Порядок встановлення системи**:

Розгортання програмного забезпечення виконується у кілька послідовних етапів. Спочатку необхідно встановити серверну частину системи, яка розміщена у каталозі MUSIC_BACKEND/. Для цього потрібно перейти до відповідного каталогу проєкту, створити та активувати віртуальне середовище Python, після чого встановити необхідні залежності. Залежності серверної частини включають FastAPI, Uvicorn, SQLAlchemy, Pydantic, бібліотеки для роботи з JWT-токенами, хешуванням паролів, PostgreSQL та платіжним сервісом Stripe.

Після встановлення залежностей необхідно налаштувати підключення до бази даних PostgreSQL. Для цього створюється база даних, після чого в конфігураційних параметрах серверної частини задається рядок підключення до неї. Також необхідно вказати секретний ключ для генерації JWT-токенів, параметри CORS для доступу з клієнтської частини та ключ Stripe для роботи з тестовою оплатою.

Запуск серверної частини виконується командою:

```
uvicorn main:app --reload
```

Після запуску FastAPI-сервер стає доступним за локальною адресою, наприклад http://127.0.0.1:8001. Під час першого запуску відбувається

ініціалізація структури бази даних за допомогою методу Base.metadata.create_all(), який створює необхідні таблиці відповідно до ORM-моделей, описаних у файлі models.py. До таких таблиць належать користувачі, треки, кошики, елементи кошика, покупки та придбані треки.

Наступним етапом є встановлення клієнтської частини системи, яка реалізована засобами React. Для цього необхідно перейти до каталогу music_frontend/, після чого встановити npm-залежності командою:

***npm install***

Після встановлення залежностей потрібно перевірити налаштування адреси серверної частини в API-конфігурації. У файлі api.js вказується базова адреса backend-сервера, наприклад:

***http://127.0.0.1:8001***

Ця адреса використовується для виконання HTTP-запитів із клієнтської частини до серверної через Axios. Також у цьому файлі реалізовано автоматичне додавання JWT-токена до заголовків запитів, якщо користувач авторизований у системі.

Запуск клієнтської частини виконується командою:

***npm start***

Після запуску React-застосунок відкривається у браузері за адресою http://localhost:3000. Користувач отримує доступ до каталогу музичного контенту, сторінок авторизації та реєстрації, кошика, сторінки придбаних треків і адміністративної панелі.

Для розгортання системи у продуктивному середовищі клієнтська частина збирається командою:

***npm run build***

У результаті створюється оптимізована версія React-застосунку, яку можна розмістити на веб-сервері або хостингу. Серверна частина при цьому запускається окремо як FastAPI-застосунок, а база даних PostgreSQL має бути доступною для підключення. Для коректної роботи системи також необхідно

забезпечити доступ до каталогів uploads/covers/, uploads/demo/ та uploads/full/, оскільки вони використовуються для зберігання обкладинок, demo-файлів і повних аудіофайлів музичних треків.

# ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи на тему «Програмне забезпечення для веборієнтованої системи продажу музичного контенту» було здійснено повний цикл розробки інформаційної системи, що охоплює етапи аналізу предметної області, проєктування бази даних, розробки клієнтської та серверної частин вебзастосунку, тестування функціоналу та підготовки рекомендацій щодо впровадження і експлуатації.

Проведений аналіз предметної області електронної комерції у сфері цифрового музичного контенту дозволив визначити ключові функціональні напрями системи: каталог музичних треків, пошук і фільтрацію за характеристиками, попереднє прослуховування, управління кошиком, авторизацію та реєстрацію користувачів, особистий кабінет, адміністративну панель, а також інші супутні процеси обслуговування. На основі виявлених потреб було сформульовано технічне завдання та розроблено концепцію програмного забезпечення.

Для опису предметної області було побудовано UML-діаграми: діаграму прецедентів, діаграму послідовності та діаграму активності, що дозволило визначити ролі користувачів, сценарії взаємодії з системою та логіку виконання основних процесів. Спроєктовано структуру реляційної бази даних у середовищі PostgreSQL із нормалізованими таблицями для зберігання даних про користувачів, треки, кошики, покупки та придбані файли. Ініціалізація структури бази даних здійснюється автоматично засобами ORM SQLAlchemy під час першого запуску серверної частини.

Серверна частина програмного забезпечення реалізована мовою Python з використанням фреймворку FastAPI. Такий вибір забезпечив швидку та ефективну розробку REST API з автоматичною документацією. Для взаємодії з базою даних використано ORM-бібліотеку SQLAlchemy, а для валідації даних -

Pydantic. Авторизація та автентифікація користувачів реалізовані на основі JWT-токенів, що забезпечує безпечний доступ до захищених ресурсів системи.

Клієнтська частина програмного забезпечення реалізована з використанням бібліотеки React у форматі Single Page Application. Такий підхід забезпечив розробку сучасного, зручного та адаптивного інтерфейсу для користувачів усіх пристроїв. Взаємодія між клієнтською і серверною частинами здійснюється через REST API з використанням бібліотеки Axios та формату JSON. У систему інтегровано аудіоплеєр для попереднього прослуховування треків, що є ключовим елементом платформи продажу музичного контенту.

Окремою складовою розробленої системи є інтеграція тестового платіжного функціоналу на основі Stripe. Реалізована dashboard-сторінка дозволяє імітувати процес придбання музичного контенту, що є основою для подальшого підключення реальної платіжної системи.

Проведено тестування основних функцій системи, що підтвердило коректну роботу авторизації та реєстрації, каталогу треків, пошуку та фільтрації, кошика, аудіоплеєра, особистого кабінету та адміністративної панелі. Розроблено інструкцію з розгортання системи, яка охоплює встановлення серверної та клієнтської частин, налаштування бази даних PostgreSQL, а також запуск у локальному та продуктивному середовищі.

У результаті виконаної роботи всі поставлені цілі та завдання були повністю досягнуті. Розроблена система успішно автоматизує ключові процеси продажу цифрового музичного контенту, поєднуючи зручність використання, стабільність роботи, ефективність обробки даних та адаптивність інтерфейсу. Реалізоване рішення є практичним, функціональним та готовим до подальшого розвитку і застосування в реальних умовах комерційної діяльності.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Томашевський В. М. Моделювання систем. – К.: Видавнича група BHV, 2005. – 352 с.

2. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd Edition. – Prentice Hall, 2004. – 736 p.

3. Що таке діаграма варіантів використання UML – [Електронний ресурс] – режим доступу: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/ (дата звернення: 24.04.2026)

4. Як будувати UML-діаграми – [Електронний ресурс] – режим доступу: https://dou.ua/forums/topic/40575/ (дата звернення: 25.04.2026)

5. Structural Diagrams – UML – GeeksforGeeks – [Електронний ресурс] – режим доступу: https://www.geeksforgeeks.org/structural-diagrams-unified-modeling-languageuml/ (дата звернення: 26.04.2026)

6. React Documentation – [Електронний ресурс] – режим доступу: https://react.dev/ (дата звернення: 27.04.2026)

7. React Quick Start – [Електронний ресурс] – режим доступу: https://react.dev/learn (дата звернення: 28.04.2026)

8. React Router Documentation – [Електронний ресурс] – режим доступу: https://reactrouter.com/ (дата звернення: 29.04.2026)

9. FastAPI Documentation – [Електронний ресурс] – режим доступу: https://fastapi.tiangolo.com/ (дата звернення: 30.04.2026)

10. Pydantic Documentation – [Електронний ресурс] – режим доступу: https://pydantic.dev/docs/ (дата звернення: 02.05.2026)

11. SQLAlchemy Documentation – [Електронний ресурс] – режим доступу: https://docs.sqlalchemy.org/ (дата звернення: 03.05.2026)

12. SQLAlchemy ORM Documentation – [Електронний ресурс] – режим доступу: https://docs.sqlalchemy.org/orm/ (дата звернення: 04.05.2026)

13. PostgreSQL Documentation – [Електронний ресурс] – режим доступу: https://www.postgresql.org/docs/ (дата звернення: 05.05.2026)

14. PostgreSQL Downloads – [Електронний ресурс] – режим доступу: https://www.postgresql.org/download/ (дата звернення: 06.05.2026)

15. What is a Logical Data Model? – [Електронний ресурс] – режим доступу: https://www.tibco.com/glossary/what-is-a-logical-data-model (дата звернення: 07.05.2026)

16. Лекція: Нормальні форми бази даних – [Електронний ресурс] – режим доступу: https://javarush.com/ua/quests/lectures/ua.questhibernate.level17.lecture02 (дата звернення: 08.05.2026)

17. Нормалізація СУБД – [Електронний ресурс] – режим доступу: https://www.guru99.com/uk/database-normalization.html (дата звернення: 09.05.2026)

18. REST API Tutorial – [Електронний ресурс] – режим доступу: https://restfulapi.net/ (дата звернення: 10.05.2026)

19. Axios Documentation – [Електронний ресурс] – режим доступу: https://axios-http.com/docs/intro (дата звернення: 11.05.2026)

20. Axios GitHub Repository – [Електронний ресурс] – режим доступу: https://github.com/axios/axios (дата звернення: 12.05.2026)

21. Stripe Documentation – [Електронний ресурс] – режим доступу: https://docs.stripe.com/ (дата звернення: 13.05.2026)

22. Stripe API Reference – [Електронний ресурс] – режим доступу: https://docs.stripe.com/api (дата звернення: 14.05.2026)

23. Visual Studio Code Documentation – [Електронний ресурс] – режим доступу: https://code.visualstudio.com/docs (дата звернення: 15.05.2026)

24. Figma Documentation – [Електронний ресурс] – режим доступу: https://help.figma.com/ (дата звернення: 16.05.2026)

25. Draw.io Documentation – [Електронний ресурс] – режим доступу: https://www.drawio.com/doc/ (дата звернення: 17.05.2026)

26. Тестування ПЗ: види тестування – [Електронний ресурс] – режим доступу: https://drukarnia.com.ua/articles/testuvannya-pz-vidi-testuvannya-JInS1 (дата звернення: 18.05.2026)

27. Hardware and software requirements: Overview, definition, and example – [Електронний ресурс] – режим доступу: https://www.cobrief.app/resources/legal-glossary/hardware-and-software-requirements-overview-definition-and-example / (дата звернення: 19.05.2026)

28. BeatStars – [Електронний ресурс] – режим доступу: https://www.beatstars.com/ (дата звернення: 20.05.2026)

29. Airbit – [Електронний ресурс] – режим доступу: https://airbit.com/ (дата звернення: 21.05.2026)

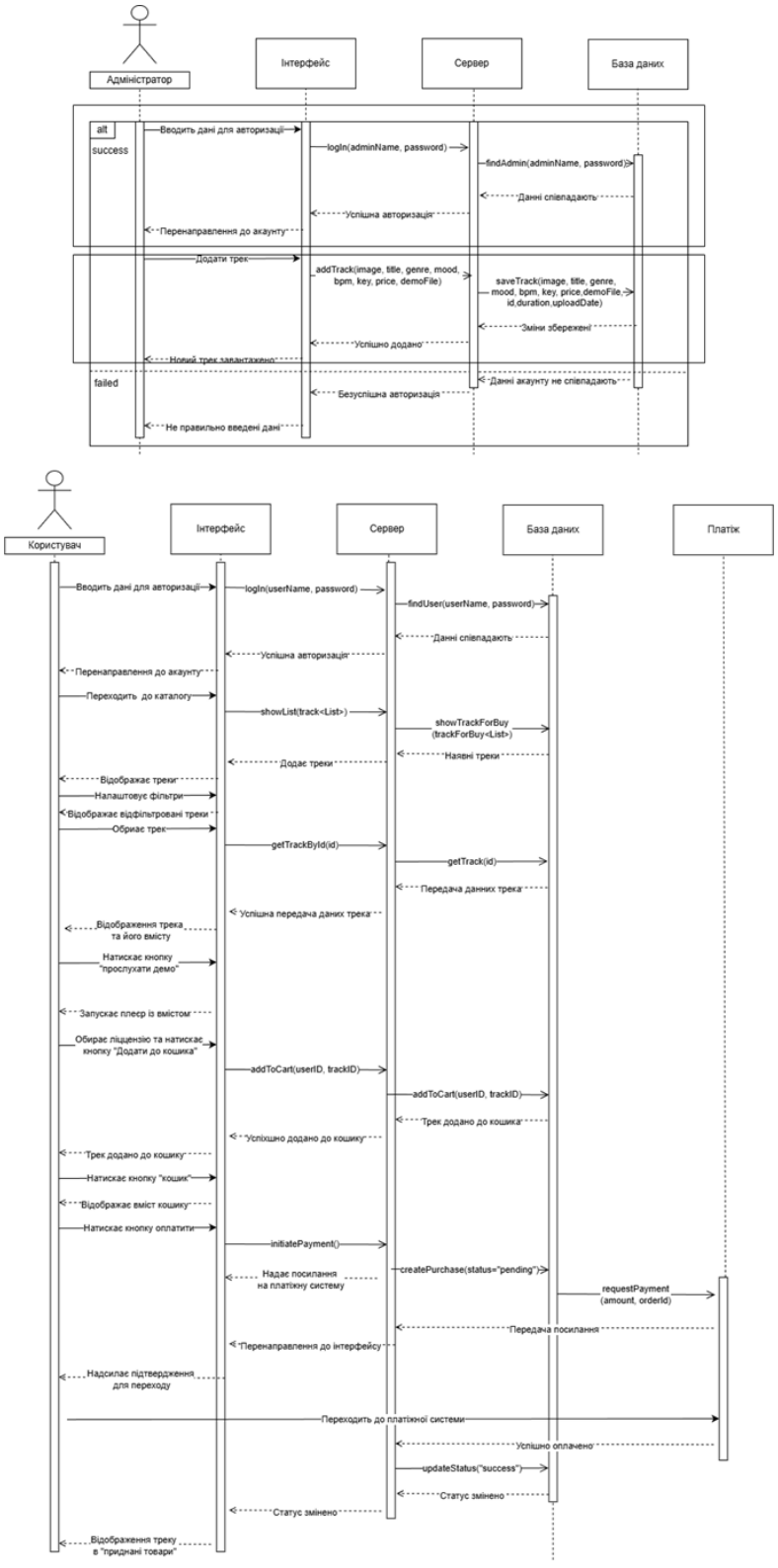
30. Bandcamp – [Електронний ресурс] – режим доступу: https://bandcamp.com/ (дата звернення: 21.05.2026)

31. OpenAI ChatGPT – [Електронний ресурс] – режим доступу: https://chatgpt.com/ (дата звернення: 22.05.2026)

# ДОДАТКИ

## Додаток А

## Діаграма послідовності



**Додаток Б**

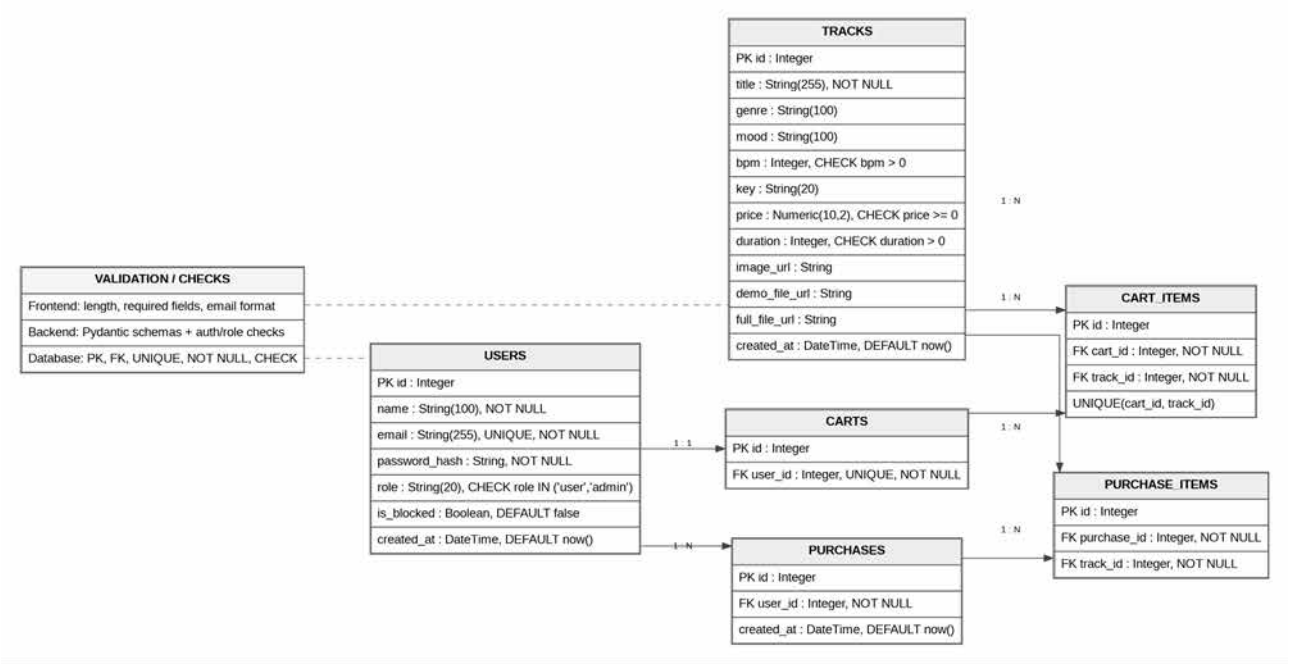
# Діаграма активності

**(Користувач)**

**(Адміністратор)**

**Додаток В**

# ER-діаграма



**Додаток Д**

**Створення всіх таблиць із зв’язками**

```
from sqlalchemy import (
        Column,
        Integer,
        String,
        DateTime,
        Numeric,
        ForeignKey,
        Boolean,
        CheckConstraint,
        UniqueConstraint
)
from sqlalchemy.sql import func
from sqlalchemy.orm import relationship
from database import Base


class User(Base):
        __tablename__ = "users"

        id = Column(Integer, primary_key=True, index=True)

        name = Column(String(100), nullable=False)
        email = Column(String(255), unique=True, index=True, nullable=False)
        password_hash = Column(String, nullable=False)

        role = Column(String(20), nullable=False, default="user")
        is_blocked = Column(Boolean, nullable=False, default=False)
```

```
        created_at = Column(DateTime(timezone=True), server_default=func.now())

        __table_args__ = (
        CheckConstraint("length(name) >= 2", name="check_user_name_length"),
        CheckConstraint("position('@' in email) > 1",
name="check_user_email_format"),
        CheckConstraint(
        "role IN ('user', 'admin')",
        name="check_user_role"
        ),
        )


class Track(Base):
        __tablename__ = "tracks"

        id = Column(Integer, primary_key=True, index=True)

        title = Column(String(255), nullable=False)
        genre = Column(String(100), nullable=True)
        mood = Column(String(100), nullable=True)

        bpm = Column(Integer, nullable=True)
        key = Column(String(20), nullable=True)

        price = Column(Numeric(10, 2), nullable=False)
        duration = Column(Integer, nullable=True)

        image_url = Column(String, nullable=True)
        demo_file_url = Column(String, nullable=True)
```

```
        full_file_url = Column(String, nullable=True)

        created_at = Column(DateTime(timezone=True), server_default=func.now())

        __table_args__ = (
        CheckConstraint("length(title) >= 2", name="check_track_title_length"),
        CheckConstraint("bpm IS NULL OR bpm > 0",
name="check_track_bpm_positive"),
        CheckConstraint("price >= 0", name="check_track_price_positive"),
        CheckConstraint(
        "duration IS NULL OR duration > 0",
          name="check_track_duration_positive"
        ),
        )


class Cart(Base):
        __tablename__ = "carts"

        id = Column(Integer, primary_key=True, index=True)
        user_id = Column(
        Integer,
        ForeignKey("users.id"),
        unique=True,
        nullable=False
        )

        user = relationship("User")
        items = relationship(
        "CartItem",
```

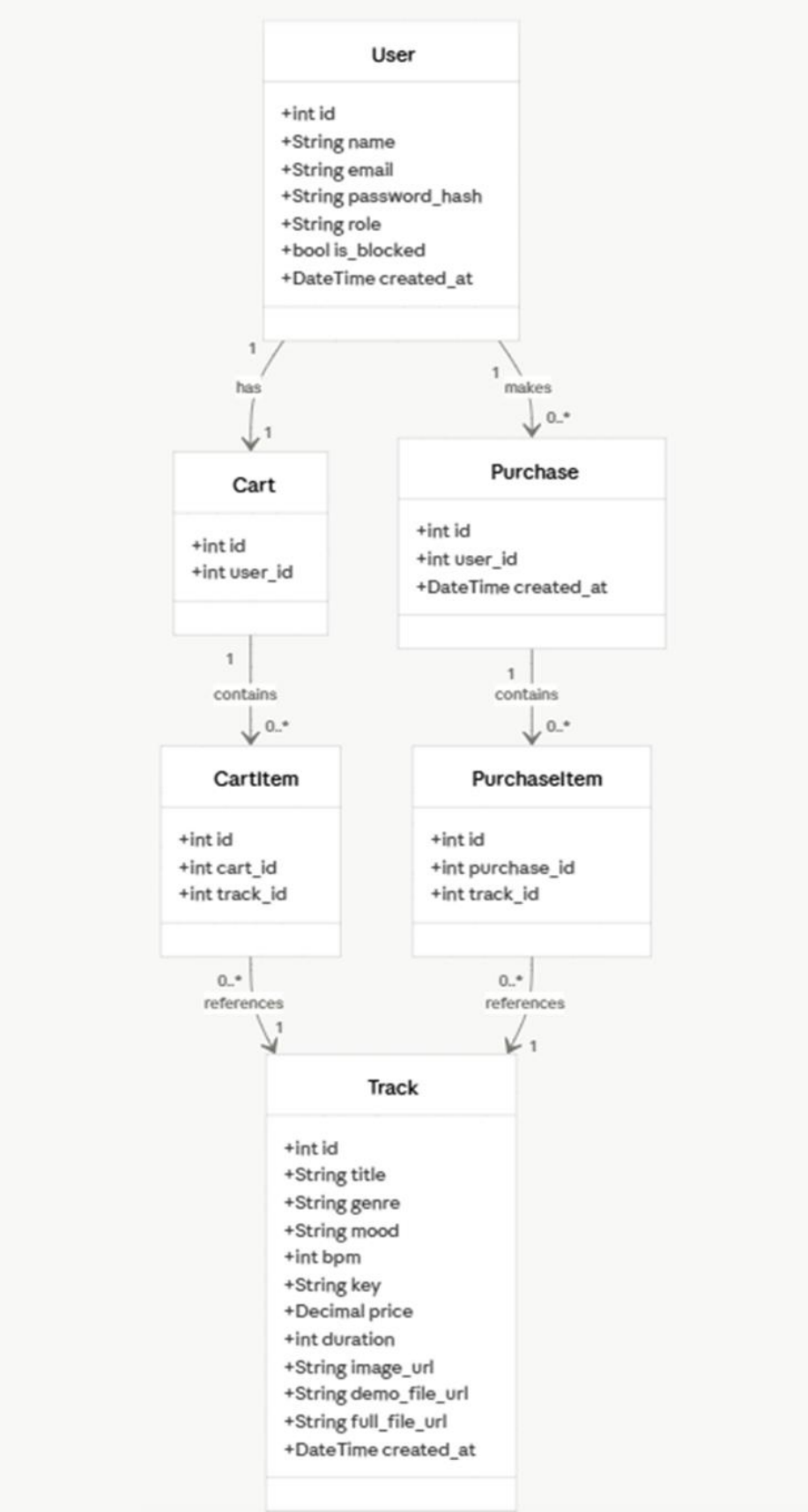
```
        back_populates="cart",
        cascade="all, delete"
        )

        __table_args__ = (
        CheckConstraint("user_id > 0", name="check_cart_user_id_positive"),
        )


class CartItem(Base):
        __tablename__ = "cart_items"

        id = Column(Integer, primary_key=True, index=True)
        cart_id = Column(Integer, ForeignKey("carts.id"), nullable=False)
        track_id = Column(Integer, ForeignKey("tracks.id"), nullable=False)

        cart = relationship("Cart", back_populates="items")
        track = relationship("Track")

        __table_args__ = (
        UniqueConstraint(
        "cart_id",
        "track_id",
         name="unique_track_in_cart"
        ),
        CheckConstraint("cart_id > 0", name="check_cart_item_cart_id_positive"),
        CheckConstraint("track_id > 0", name="check_cart_item_track_id_positive"),
        )
```

```
class Purchase(Base):
    __tablename__ = "purchases"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now())

    user = relationship("User")
    items = relationship(
    "PurchaseItem",
    back_populates="purchase",
    cascade="all, delete"
    )

    __table_args__ = (
    CheckConstraint("user_id > 0", name="check_purchase_user_id_positive"),
    )


class PurchaseItem(Base):
    __tablename__ = "purchase_items"

    id = Column(Integer, primary_key=True, index=True)
    purchase_id = Column(Integer, ForeignKey("purchases.id"), nullable=False)
    track_id = Column(Integer, ForeignKey("tracks.id"), nullable=False)

    purchase = relationship("Purchase", back_populates="items")
    track = relationship("Track")

    __table_args__ = (
```

```
UniqueConstraint(
"purchase_id",
"track_id",
 name="unique_track_in_purchase"
),
CheckConstraint(
"purchase_id > 0",
 name="check_purchase_item_purchase_id_positive"
),
CheckConstraint(
"track_id > 0",
 name="check_purchase_item_track_id_positive"
),
)
```

**Додаток Е**

# Фізична структура бази даних



**Додаток Ж**

# Використання React-компонентів

## Ж.1. Компонент кнопки MyButton.

Код компонента:

```
import React from "react";
import classes from "./MyButton.module.css";

const MyButton = ({ children, ...props }) => {
  return (
    <button {...props} className={classes.myBtn}>
      {children}
    </button>
  );
};

export default MyButton;
```

Приклад використання компонента:

```
import React from "react";
import MyButton from "../Components/UI/button/MyButton";

export default function ButtonExample() {
  function handleClick() {
    alert("Дію виконано");
  }

  return (
```

```
    <div>
      <MyButton onClick={handleClick}>
        Додати трек
      </MyButton>
        </div>
  );
}
```

**Ж.2. Компонент поля введення MyInput.**

Код компонента:

```
import React from "react";
import classes from "./MyInput.module.css";

const MyInput = React.forwardRef((props, ref) => {
  return (
    <input
      ref={ref}
      className={classes.myInput}
      {...props}
        />
  );
});

export default MyInput;
```

Приклад використання компонента:

```
import React, { useState } from "react";
import MyInput from "../Components/UI/input/MyInput";
```

```
import MyButton from "../Components/UI/button/MyButton";

export default function TrackFormExample() {
  const [title, setTitle] = useState("");
  const [price, setPrice] = useState("");

  function saveTrack(event) {
    event.preventDefault();

    console.log({
      title: title,
      price: price
        });
  }

  return (
    <form onSubmit={saveTrack}>
      <MyInput
        type="text"
        placeholder="Назва треку"
        value={title}
        onChange={(event) => setTitle(event.target.value)}
      />

      <MyInput
        type="number"
        placeholder="Ціна треку"
        value={price}
        onChange={(event) => setPrice(event.target.value)}
      />
```

```
      <MyButton>
        Зберегти
      </MyButton>
    </form>
  );
}
```

**Ж.3. Компонент модального вікна MyModal.**

Код компонента:

```
import React from "react";
import cl from "./MyModal.module.css";

const MyModal = ({ children, visible, setVisible }) => {
  const rootClasses = [cl.myModal];

  if (visible) {
    rootClasses.push(cl.active);
  }

  return (
    <div
      className={rootClasses.join(" ")}
      onClick={() => setVisible(false)}
        >
      <div
        className={cl.myModalContent}
```

```
            onClick={(event) => event.stopPropagation()}
          >
            {children}
          </div>
        </div>
      );
    };

    export default MyModal;
```

Приклад використання компонента:

```
    import React, { useState } from "react";
    import MyModal from "../Components/UI/modal/MyModal";
    import MyInput from "../Components/UI/input/MyInput";
    import MyButton from "../Components/UI/button/MyButton";

    export default function ModalExample() {
      const [modal, setModal] = useState(false);
      const [comment, setComment] = useState("");

      return (
        <div>
          <MyButton onClick={() => setModal(true)}>
            Відкрити вікно
          </MyButton>

          <MyModal visible={modal} setVisible={setModal}>
            <h2>Додати коментар</h2>

            <MyInput
```

```
          placeholder="Введіть коментар"
          value={comment}
          onChange={(event) => setComment(event.target.value)}
        />

        <MyButton onClick={() => setModal(false)}>
          Зберегти
        </MyButton>
        </MyModal>
        </div>
    );
}
```

**Додаток І**

## Використання файлів серверної частини

Файл database.py:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

DATABASE_URL = "postgresql://postgres:password@localhost:5432/music_store"

engine = create_engine(DATABASE_URL)

SessionLocal = sessionmaker(
   autocommit=False,
   autoflush=False,
   bind=engine
)

Base = declarative_base()



def get_db():
      db = SessionLocal()
      try:
      yield db
   finally:
      db.close()
```

Приклад використання:

```
from database import Base, engine, get_db

Base.metadata.create_all(bind=engine)



@app.get("/tracks")
def get_tracks(db: Session = Depends(get_db)):
    tracks = db.query(Track).all()
    return tracks
```

Файл models.py:

```
from sqlalchemy import Column, Integer, String, DateTime, Numeric, Boolean
from sqlalchemy.sql import func
from database import Base



class User(Base):
  __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    email = Column(String(255), unique=True, index=True, nullable=False)
  password_hash = Column(String, nullable=False)
    role = Column(String(20), nullable=False, default="user")
  is_blocked = Column(Boolean, nullable=False, default=False)
  created_at = Column(DateTime(timezone=True),
server_default=func.now())
```

```
class Track(Base):
    __tablename__ = "tracks"

        id = Column(Integer, primary_key=True, index=True)
        title = Column(String(255), nullable=False)
        genre = Column(String(100), nullable=True)
        mood = Column(String(100), nullable=True)
        bpm = Column(Integer, nullable=True)
        key = Column(String(20), nullable=True)
        price = Column(Numeric(10, 2), nullable=False)
    duration = Column(Integer, nullable=True)
    image_url = Column(String, nullable=True)
    demo_file_url = Column(String, nullable=True)
    full_file_url = Column(String, nullable=True)
    created_at = Column(DateTime(timezone=True),
server_default=func.now())
```

Приклад використання:

```
from models import User, Track


@app.get("/tracks")
def get_tracks(db: Session = Depends(get_db)):
    tracks = db.query(Track).all()
    return tracks
```

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

## Декларація про академічну доброчесність та використання ШІ

## ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Шуляк Антон Олександрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення для веборієнтованої системи продажу музичного контенту» є результатом моєї особистої праці.

1. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
2. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [✓] інструменти ШІ ChatGPT, Gemini, були використані для:
        - [ ✓] перекладу іншомовних джерел;
        - [✓ ] стилістичного редагування та перевірки граматики;
        - [✓ ] допомоги у написанні/відлагодженні програмного коду;
        - [✓ ] інше: структурування окремих фрагментів тексту.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р _____ **Шуляк Антон**

(підпис)

**БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА**

**ШУЛЯКА АНТОНА ОЛЕКСАНДРОВИЧА**

Наказ НУБіП України №3196 «С» від 19.12.2025