ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Роман ГУМЕНЮК
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Роман ГУМЕНЮК

Керівник: Володимир САДОВЕНКО
к.ф.-м.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« ____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гуменюку Роману Вадимовичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків»

керівник кваліфікаційної роботи к.ф.-м.н., доцент Володимир САДОВЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про застосунки власного аудіального середовища, опис методів обробки аудіо-файлів та формування міксів, технічна документація Python та його засобів для роботи з бібліотеками аудіо.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Зробити аналіз функціональних особливостей існуючих рішень на предмет обмежень щодо налаштування та адаптації звукового середовища.
2. Формулювання функціональних та нефункціональних вимог.
3. Обрати відповідні інструментальні засоби на основі аналізу програмного забезпечення для реалізації персоналізованого аудіального середовища.

4. Реалізувати прототип програмного засобу з графічним інтерфейсом, що підтримує створення, налаштування та відтворення звукових міксів.

5. Провести загальне тестування системи з метою виявлення помилок і визначення відповідності додатку функціональних вимог.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Архітектура застосунку.
5. Алгоритм роботи застосунку.
6. Екранні форми.
7. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих алгоритмів обробки аудіо-файлів і технологій побудови власних міксів користувача	14.03-20.03.2025	
4	Розробка вікна авторизації користувача за допомогою TKinter	21.03-27.03.2025	
5	Програмна реалізація основної частини застосунку	28.03-17.04.2025	
6	Тестування програмного забезпечення	18.04-28.05.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Роман ГУМЕНЮК
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 68 стор.,17 рис., 30 джерел.

Мета роботи – з'ясування можливостей підвищення когнітивної ефективності та психоемоційного комфорту користувача шляхом впровадження персоналізованого звукового оточення, сформованого за допомогою спеціалізованого програмного забезпечення.

Об'єкт дослідження – процес формування індивідуалізованого звукового оточення в цифрових інформаційних системах.

Предмет дослідження – методи адаптивного керування звуковим середовищем на основі параметрів користувача в межах програмного забезпечення персоналізованого аудіального супроводу.

Короткий зміст роботи: У роботі розглянуто проблему створення персоналізованого звукового оточення як інструменту підвищення когнітивної ефективності та психоемоційного комфорту користувача. Проведено аналіз існуючих програмних рішень, окреслено їхні переваги та недоліки, сформульовано вимоги до функціональності нового застосунку. Запропоновано архітектуру програмного забезпечення, реалізовано прототип системи з графічним інтерфейсом, підтримкою мікшування звуків, збереження профілів, сценарного відтворення та ручної адаптації. Результати функціонального тестування підтвердили працездатність і стабільність розробленої системи, що засвідчує її прикладну цінність і перспективу подальшого розвитку.

Сферою використання застосунку є організація індивідуалізованого акустичного середовища для підвищення концентрації, зниження рівня стресу, підтримки сну або покращення умов навчання й роботи.

КЛЮЧОВІ СЛОВА: ПЕРСОНАЛІЗОВАНЕ ЗВУКОВЕ ОТОЧЕННЯ, ЗВУКОВИЙ МІКС, АУДІОІНТЕРФЕЙС, СЦЕНАРНЕ ВІДТВОРЕННЯ, МІКШУВАННЯ, PYTHON, TKINTER, PYDUB, КОРИСТУВАЦЬКИЙ ПРОФІЛЬ.

ЗМІСТ

ВСТУП

У сучасному інформаційно-технологічному середовищі спостерігається зростаюча потреба в адаптивних програмних засобах, здатних забезпечити індивідуалізований досвід взаємодії користувача з цифровими платформами. Одним з напрямів, що стрімко розвивається, є персоналізація аудіального простору з метою підвищення продуктивності, зниження рівня стресу або створення комфортного фону для різних видів діяльності. Актуальність розробки програмного забезпечення, яке дозволяє формувати персоналізоване звукове оточення на основі попередньо підготовленої бібліотеки звуків, зумовлена як зростанням кількості віддалених працівників, фрилансерів, студентів, так і зростанням зацікавленості у використанні звуків природи, шумів міста або ритмічних композицій у сфері медитації, терапії, ігор та творчих процесів. Проблематика полягає не лише у технічній реалізації мікшування та програвання звуків, але й у забезпеченні адаптивності системи до уподобань, поточних цілей та психоемоційного стану користувача.

Об'єктом дослідження у даній роботі виступає процес формування динамічного звукового середовища на базі цифрових звукових даних. Предметом – методи комп'ютерної генерації та обробки звукових композицій з використанням бібліотек семплів та алгоритмів мікшування з урахуванням індивідуальних налаштувань. Метою дослідження є розробка програмного забезпечення, що реалізує функціонал створення персоналізованого аудіального оточення з можливістю комбінування, налаштування гучності, таймування та циклічності звукових шарів, а також збереження та повторного використання створених конфігурацій. У межах розробки передбачається вивчення існуючих програмних рішень, визначення архітектурної моделі системи, обґрунтування вибору інструментів реалізації, а також створення функціонального прототипу з графічним інтерфейсом користувача.

Новизна дослідження полягає у розробці програмного засобу, що поєднує концепції звукового дизайну та користувацької персоналізації в одному рішенні, орієнтованому на широку аудиторію без необхідності спеціальних знань у сфері аудіотехнологій. Програмне забезпечення має підтримувати динамічне керування звуковими потоками, що дозволить користувачам самостійно формувати композиції на основі власного сприйняття та цілей. У цьому контексті розробка не лише розширює інструментарій побутового і професійного використання звуків, а й робить доступними технології звукового середовища для широкого загалу, включаючи осіб з особливими освітніми або психоемоційними потребами.

Підсумовуючи, можна зазначити, що дана робота спрямована на вирішення міждисциплінарного завдання, в якому поєднуються програмна інженерія, елементи аудіотехнологій та принципи UX-дизайну. Розроблене програмне забезпечення має продемонструвати не лише технічну коректність та ефективність реалізації, але й здатність до адаптації під різні сценарії використання, що в сукупності становить практичну цінність і перспективу подальшого вдосконалення та комерційного застосування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Принцип вибору персоналізованого звукового оточення

Сучасні користувачі все частіше використовують фоновий звук чи музику для поліпшення концентрації, релаксації або навчання під час роботи з програмним забезпеченням. Згідно з опитуваннями, близько трьох чвертей розробників програмного забезпечення слухають музику під час кодування. Дослідження підтверджують, що правильно підібраний аудіофон може зменшувати блукання уваги і покращувати зосередженість на завданні[1]. Однак вплив звукового середовища дуже індивідуальний – звук, що одній людині допомагає сконцентруватися, для іншої може стати відволікаючим фактором. Тому виникає потреба у персоналізації звукового оточення – адаптації аудіо до психологічних та фізіологічних особливостей конкретного користувача. Персоналізоване звукове оточення визначається як технологія, що налаштовує аудіо під унікальні можливості слуху та вподобання людини (наприклад, через простий тест у додатку) для більш комфортного сприйняття.

У цій статті розглянуто науково обґрунтовані принципи розробки персоналізованих аудіосередовищ у програмних застосунках. Ми проаналізуємо психоакустичні та когнітивні фактори сприйняття звуку, методи збору даних про індивідуальні звукові вподобання, алгоритми адаптації аудіо до контексту використання, сучасні технології (штучний інтелект, машинне навчання, сенсори) для реалізації персоналізації, приклади впровадження в існуючих додатках, а також питання приватності, етики та технічні обмеження[2].

Психоакустика вивчає, як фізичні характеристики звуку сприймаються людським слухом і психікою. До ключових психоакустичних параметрів належать гучність (інтенсивність звуку), частотний склад (висота тону та тембр), різкість/шорсткість звуку та його динамічність. Ці параметри впливають на емоційний і фізіологічний стан слухача. Зокрема, дослідження показують, що

гучність та шорсткість звуку підвищують рівень збудження та можуть викликати стрес, тоді як більш м’які і плавні звуки сприяють розслабленню. Низькочастотні ритмічні компоненти, наближені до частоти серцевого ритму, можуть заспокоювати або утримувати увагу, тоді як різкі високочастотні звуки та різкі сплески (імпульсивні шуми) асоціюються зі збудженням і навіть почуттям тривоги. Таким чином, характеристики звуку безпосередньо впливають на емоційний фон: наприклад, монотонний білий шум може маскувати відволікаючі звуки і таким чином підвищувати комфорт, а гучний та хаотичний шум перевантажує слухову систему і знижує здатність концентруватися[3].

Когнітивні фактори визначають, як звук впливає на мислення, увагу та працездатність. Мозок має обмежену уважність, тому занадто складне або привертаюче увагу аудіо (наприклад, музика з вокалом чи різкими змінами) може конкурувати з основним завданням і погіршувати продуктивність. Натомість правильно підібраний фон здатен оптимізувати рівень збудження мозку. Відомо, що існує нелінійна залежність між рівнем активації (apousal) і продуктивністю (зокрема, закон Єркс-Додсона): і надто низький, і надто високий рівень збудження шкодять виконанню завдання. Звукове оточення допомагає регулювати цей рівень. Наприклад, у відносній тиші можуть переважати стани сонливості чи блукання думок, тоді як помірний фоновий шум здатен знизити кількість відволікань на сторонні думки і підтримувати стан “робочого потоку”. Дослідники з’ясували, що помірний рівень фонового шуму (~70 дБ, приблизно як звук кафе) найкраще стимулює творче мислення, тоді як занадто тихе (50 дБ) чи дуже гучне (85 дБ) середовище вже не дає такого ефекту. Це означає, що оптимальний звуковий фон повинен бути достатнім для легкого підвищення збудженості кори, але не настільки інтенсивним, щоб відволікати[4].

Крім інтенсивності, велике значення має структура та зміст звуку. Музика з непередбачуваною структурою або зі словами може навантажувати когнітивні ресурси (особливо вербальні центри) і конкурувати з вербальними завданнями (читанням, програмуванням, навчанням). Навпаки, інструментальна або електронна музика без різких перепадів створює стимуляцію без відволікання – цей

принцип особливо важливий для людей з розладом уваги (ADHD), яким потрібен деякий рівень стимулу для концентрації. Платформа Brain.fm, наприклад, навмисно конструює музику так, щоб вона не містила відволікаючих елементів (немає різких звуків чи вокалу), а натомість підтримувала ритмічну стимуляцію мозку. Їхні експерименти з ЕЕГ показали, що спеціально спроєктована музика підвищує потужність фокусованих мозкових хвиль майже вдвічі та покращує здатність утримувати увагу у людей з симптомами дефіциту уваги. Водночас, якщо фонова музика не відповідає вподобанням або занадто насичена, вона може погіршувати результати навчання і пам'яті – деякі дослідження відзначають, що студенти показували кращі результати запам'ятовування в тиші, ніж під музику, яка їм не подобається. Тому врахування індивідуальних особливостей сприйняття є критично важливим: одна і та сама мелодія може заспокоювати одну людину, але дратувати іншу через особисті асоціації, культурний бекграунд чи навіть фізіологічні відмінності слуху[5].

Індивідуальні відмінності слухового сприйняття теж належать до психоакустичних факторів. Фізіологія слуху різниться: наприклад, з віком знижується чутливість до високих частот, міські жителі з досвідом шумового забруднення можуть мати нерівномірну слухову чутливість на різних вухах тощо. Це означає, що *"нейтральний"* звук (наприклад, стандартний частотний баланс навушників) на практиці по-різному сприймається людьми. Немає єдиного універсального аудіопрофілю, що підходить усім – дослідження на 150 тисячах слухачів показало, що жоден фіксований звуковий профіль навушників не був оптимальним більш ніж для 17% людей, тоді як після персоналізації звуку задоволення слухачів зросло до 82%. Таким чином, на етапі проектування звукового оточення варто врахувати базові психоакустичні принципи (оптимальний рівень шуму, відсутність різких дратівливих звуків, доречний темп і тональність), але обов'язково передбачити можливість індивідуального налаштування, щоб користувач міг підлаштувати звук під свої когнітивні потреби та комфорт[6].

Персоналізація аудіо вимагає розуміння, що саме полюбляє чи потребує конкретний користувач. Для цього застосовують як явні методи збору вподобань (explicit feedback), так і неявні, автоматичні (implicit feedback).

Явні методи передбачають активну участь користувача у налаштуванні звуку. Найпростіший підхід – надати користувачу вибір і запитати про його вподобання. Наприклад, система може програти два варіанти звучання (A/B тест) і попросити вибрати більш приємний – на основі кількох таких раундів будується унікальний аудіопрофіль слухача. Подібний підхід реалізовано в технології SoundID від Sonarworks: новому користувачу пропонують пройти короткий тест, де він прослуховує музичні уривки з різними налаштуваннями еквалайзера і обирає, який звук кращий. Результатом є індивідуальна крива частотної корекції, що компенсує особливості його слуху та смаку, забезпечуючи найбільш приємний звук. Масштабне дослідження Sonarworks показало, що 82% слухачів надають перевагу саме такому персоналізованому налаштуванню звуку порівняно зі стандартним звучанням їхніх навушників. Інший явний метод – опитування та анкети: користувача можуть питати про жанри музики, які він любить під час роботи чи відпочинку, чи про бажаний тип фонового шуму (наприклад, “природні звуки” vs “білий шум” тощо). Такі суб’єктивні оцінки дають цінні дані про свідомі вподобання[7].

Неявні методи збирають дані про вподобання без прямого опитування, відстежуючи поведінку користувача. Програмне забезпечення може моніторити, як користувач взаємодіє зі звуком: які звукові теми або треки він частіше вмикає, що перемикає чи вимикає, до якого рівня гучності встановлює, в які моменти робить паузи. Наприклад, якщо користувач постійно зменшує гучність або пропускає певний тип музики (скажімо, треки з вокалом) – система може зробити висновок, що цей елемент йому заважає, і надалі пропонувати менш нав’язливі інструментальні звуки. Таке навчання на поведінці схоже на принцип роботи рекомендаційних систем, наприклад, стримінгових сервісів: алгоритми аналізують історію прослуховувань, оцінок і дій, щоб спрогнозувати, який контент сподобається користувачу. Зокрема, широко використовується колаборативна

фільтрація – виявлення схожих патернів серед багатьох користувачів. Якщо користувач А має подібні вподобання до користувача В, то невідомі переваги А можна передбачити на основі того, що вподобає В. У контексті звукового оточення це може означати: система навчиться групувати користувачів за аудіо-профілями (наприклад, “любителі спокійних природних звуків”, “ті, хто фокусуються під електронну музику” тощо) і новому користувачу з певної групи одразу пропонувати аудіо, що сподобалося іншим із цієї групи. Важливо, що такі алгоритми постійно коригуються на основі зворотного зв’язку – явного (оцінки, лайки/дизлайки звукових сцен) або неявного (час використання тієї чи іншої звукової сцени, показники ефективності роботи користувача тощо)[8].

Фізіологічний моніторинг – новітній підхід до персоналізації звуку, що передбачає вимірювання реакцій тіла користувача на аудіо. Ідея полягає в тому, щоб оцінити, наскільки добре вибране звукове оточення реально допомагає користувачу (чи то у фокусуванні, чи в релаксації) за об’єктивними біомаркерами стресу і зосередженості. Наприклад, можна використовувати варіабельність серцевого ритму (ВСР): високий показник ВСР зазвичай відповідає стану розслабленості й концентрації, тоді як низький – стресу або перевтоми. Якщо додаток має доступ до пульсу користувача (через розумний годинник або нагрудний датчик), він може відстежувати зміни пульсу і ВСР під час програвання різних звукових фонів. Дослідження Dartmouth College (проєкт *HeartDJ*) випробувало такий підхід: в режимі реального часу генерувалася музика на основі показників серця, щоб зменшити стрес і покращити фокус. Результати виявилися неоднозначними – хоча місцями спостерігалися покращення (наприклад, спокійні музичні треки дійсно підвищували показники релаксації у частини слухачів), загальний ефект сильно залежав від індивідуальних особливостей. З’ясувалося, що люди з різними звичками слухання музики реагували на нові AI-згенеровані звуки по-різному: тим, хто мало слухає музику в повсякденному житті, було важче розслабитися під запропоновані треки, можливо через відсутність емоційного зв’язку з ними. Цей експеримент підкреслив критичну роль особистої знайомості та вподобання: навіть якщо алгоритм технічно підлаштовує музику під біосигнали,

звук мусить подобатися користувачу, інакше бажаного ефекту не буде. Таким чином, майбутні системи можуть комбінувати обидва підходи – враховувати суб'єктивні вподобання та об'єктивні фізіологічні дані. Наприклад, додаток для медитації може запитати в користувача, які звуки йому приємні, і паралельно моніторити через смарт-годинник, чи справді під ці звуки падає частота пульсу та напруга м'язів[9].

Аналіз зібраних даних про вподобання зазвичай виконується алгоритмами машинного навчання. Накопичуючи інформацію про багато користувачів, система може виявити приховані закономірності: які звукові профілі існують, як вони корелюють з певними типами особистостей чи задач. Науковці також досліджують моделі прогнозування уподобань за непрямими показниками – наприклад, за типом музики, що людина слухає у вільний час, прогнозувати, який фон їй підійде для роботи. Важливо пам'ятати, що уподобання не є статичними: настрій, контекст і навіть здоров'я слуху з часом змінюються, тож системи персоналізації мають передбачати адаптивне навчання. Алгоритм може постійно уточнювати модель користувача: якщо сьогодні він раптом перемкнув спокійну музику на енергійнішу, система відзначить цю зміну і спробує з'ясувати причину (можливо, інший контекст або втома). У підсумку, поєднання самонастроювання користувачем і автоматичного навчання на його поведінці та фізіології є найефективнішим шляхом збору індивідуальних аудіо-вподобань для подальшої персоналізації[10].

Зібравши дані про вподобання користувача, система персоналізованого звуку повинна динамічно адаптувати аудіо під поточний контекст використання. Під контекстом розуміється ситуація або мета, з якою користувач взаємодіє із програмою: чи він намагається зосередитися на складному завданні, чи прагне розслабитися, чи, можливо, вчиться нового матеріалу. Кожен контекст висуває свої вимоги до звукового фону.

1. Режим фокусування (концентрації). Для підтримки глибокої концентрації важливо мінімізувати відволікання і водночас підтримати мозок в стані легкої активації. Алгоритми адаптації у цьому випадку, як правило, обирають монотонні або ритмічні звукові доріжки без різких змін. Наприклад, додаток Endel у режимі

Focus генерує звуковий ландшафт з легким перкусійним ритмом, синхронізованим з ритмом серця людини, щоб підтримувати її у стані “потоку”. Мелодійність зведена до мінімуму, звук слугує фоновим “тактметром” для мозку. Наукове підґрунтя такого підходу – ефект вирівнювання серцевого ритму та мозкових хвиль із зовнішнім ритмом: якщо звуковий темп близький до частоти серцевих скорочень у стані спокійної уваги (~60-80 уд/хв), це сприяє стабілізації уваги. Додатково, жодних вокалів або різких акордів: музика під фокусування часто нагадує спокійний ембієнт або ненав’язливий електронний саундтрек[11]. Алгоритм може також маскувати небажані перешкоди: якщо мікрофони пристрою фіксують раптовий шум довкілля (наприклад, хтось заговорив поряд), система може негайно підвищити гучність білого шуму чи маскуючого звуку, щоб перекрити потенційно відволікаючі слова. Такі системи маскування шуму давно застосовуються в офісах: вони випускають рівномірний шумовий фон, що робить сторонні розмови менш розбірливими і тим самим покращує концентрацію людей. У програмному рішенні для індивідуального користувача це може бути реалізовано через навушники з активним шумопоглинанням у поєднанні з генерацією шуму, адаптивного до обстановки[12].

2. Режим розслаблення. Коли мета – зняти стрес і заспокоїти нервову систему, алгоритми обирають зовсім інші звуки. Тут діє принцип зниження рівня активації мозку. Ефективні є повільні, плавні звукові тексти – звуки природи (шелест листя, дощ, хвилі океану), спокійна інструментальна музика, дзвіночки чи спів птахів. Такі звуки історично асоціюються з безпекою і відпочинком, вони мають багатий спектр на низьких та середніх частотах без різких піків[13]. Алгоритм може включати елементи бінауральних биттів або певних частотних модуляцій, що спрямовані на виклик альфа-ритму мозку (~8-12 Гц) – стану, пов’язаного з розслабленням. Деякі додатки (наприклад, Mistikist) генерують спеціальні тони для brainwave entrainment – синхронізації мозкових хвиль зі звуковими коливаннями, щоб переводити користувача в спокійніший ментальний стан. Хоча ефективність бінауральних ритмів ще досліджується, користувачі часто суб’єктивно відзначають зниження тривожності під такі аудіостими. Алгоритм

розслаблення може також реагувати на біометрики: якщо в користувача зафіксовано високий пульс чи рівень стресу, система може автоматично увімкнути заспокійливий саундтрек (наприклад, звук дощу) і поступово знизити його темп та гучність по мірі нормалізації пульсу. Важливо, що в розслаблюючих сценаріях адаптація не повинна бути різкою – зміни в звукоряді треба вносити дуже плавно, щоб не привертати зайвої уваги слухача, який може перебувати в напівмедитативному стані.

3. Режим сну. Аудіо для сну – окремий випадок релаксації. Його мета – полегшити засинання і підтримувати глибокий сон. Алгоритми генерують монотонний, постійний фон без різких звуків. Застосовуються низькочастотні коливання (так звані δ-хвилі, <4 Гц) – аналогічні до частот мозку у стані глибокого сну. Так, режим Sleep в Endel використовує δ-хвильову модуляцію звуку, що сприяє зануренню в сон[14]. Звук поступово загасає упродовж заданого часу, підлаштовуючись під циркадні ритми користувача. Деякі рішення контролюють ще й зовнішній шум під час сну: якщо вночі з’являється гучний шум (наприклад, вуличний транспорт), система може на мить підвищити рівень білого шуму для маскування, аби спляча людина не прокинулась. Для моніторингу якості сну застосовують навіть розумні датчики (наприклад, пристрої під матрацем або годинники) – за їх даними алгоритм на ранок оцінює, наскільки обраний звуковий сценарій був ефективним, і може запропонувати зміни (наприклад, інший звук природи або інший рівень гучності наступної ночі).

4. Режим навчання. Під час навчання (особливо коли треба запам’ятати інформацію або зрозуміти новий матеріал) вимоги до аудіо залежать від типу матеріалу. Якщо це читання тексту чи аудіолекція, то зайвий звук може заважати сприйняттю вербальної інформації. У таких випадках найкращим звуковим оточенням може бути тиша або нейтральний шум, який відсікає сторонні звуки. З іншого боку, при виконанні рутинних вправ чи розв’язуванні задач, легкий музичний фон може підтримувати мотивацію. Деякі освітні платформи інтегрують гейміфікацію і фонова музика слугує для створення атмосфери (наприклад, спокійний трек під час читання теорії і більш енергійний – під час практичних

вправ). Алгоритми можуть адаптувати аудіо до складності завдання: дослідження показують, що за легкої задачі музика не заважає, а от за високої когнітивної складності навіть улюблена музика може погіршувати результати. Тому система може автоматично вимикати фоновий звук під час найважчих етапів (наприклад, тестування чи виконання контрольних питань) і вмикати в паузах для відпочинку. Персоналізація тут полягає в тому, щоб навчити алгоритм розуміти, в якому контексті зараз перебуває учень: це можна визначати за типом відкритого контенту, за темпом роботи користувача (швидко відповідає чи довго думає) або навіть за виразом обличчя/голосу (деякі експериментальні навчальні програми аналізують мікроміміку чи тон голосу для оцінки фрустрації)[15]. Звісно, такі підходи тільки зароджуються, і поки що частіше реалізуються прості сценарії – наприклад, можливість вручну вибрати спокійний фон для читання чи більш бадьорий для тренування навичок.

Штучний інтелект (ШІ) та суміжні технології відіграють ключову роль у побудові персоналізованих аудіосистем. По-перше, AI-алгоритми застосовуються для аналізу даних та прогнозування вподобань користувача. Як згадано, системи рекомендації використовують моделі машинного навчання (ML), що вчаться на великих масивах даних про взаємодію користувачів зі звуком. Сучасні платформи, такі як Spotify або YouTube, використовують глибокі нейронні мережі і граф-аналіз (наприклад, графові нейронні мережі) для знаходження прихованих зв’язків між треками та слухачами. Ці ж підходи можуть бути адаптовані під персоналізацію фонового звуку: нейромережа може навчитися відповідності між профілем користувача/контекстом і оптимальним аудіо на основі історичних даних.

По-друге, генерація звуку в реальному часі все більше спирається на нейромережі. Традиційно фонову музику або шуми брали з наперед підготовленої бібліотеки записів. Такий підхід обмежений: бібліотека може не врахувати всі комбінації настроїв і ситуацій. Натомість, генеративні моделі штучного інтелекту здатні створювати новий аудіоконтент на льоту, під конкретні вимоги. Існують рекурентні нейронні мережі та трансформери, навчені на великих обсягах музики, які можуть синтезувати мелодії заданого жанру або настрою. В 2020 році OpenAI

представила нейромережу Jukebox, що генерує музику в різних стилях у форматі сирого аудіо. Такі моделі можуть стати основою для персоналізованого аудіо: наприклад, алгоритм отримує від системи високого рівня команду “згенеруй 30 хвилин спокійної фортепіанної музики з шумом дощу на фоні” – і модель синтезує унікальний трек, який ніколи раніше не існував, спеціально для цього користувача. Перевага – нескінченна варіативність і точний контроль за характеристиками (темп, тональність, інтенсивність змін), що важко досягти з готовими записами. Недолік – високі вимоги до обчислень (генерація сирого аудіо в реальному часі – складне завдання) і поки що емоційна неповноцінність AI-музики: користувачі можуть відчувати, що згенерована музика “бездушна” або повторювана[16]. Проте технології швидко прогресують – компанії на кшталт AIVA, Melodrive, Brain.fm розробляють спеціалізовані нейромережі, оптимізовані для створення функціональної музики (тобто музики, що слугує конкретній функції – фокус, сон, медитація). Brain.fm, наприклад, має запатентовану технологію нейронного саунд-дизайну, де алгоритми генерують музичні патерни з урахуванням принципів нейронауки (ритмічна модуляція сигналу, відсутність відволікань). Експерименти з fMRI та EEG підтвердили, що така AI-музика викликає специфічну активацію мозку, узгоджену зі станом зосередження. Це демонструє потенціал нейромереж як інструменту тонкого налаштування звукового впливу.

Обчислювальні технології також мають значення. Персоналізовані аудіо-сервіси часто реалізовані як хмарні (cloud-based), оскільки потребують обробки великих даних і можливостей генерації. Наприклад, Endel розгорнула свій генеративний звуковий рушій Endel Pacific на різних платформах, включно з хмарою, щоб забезпечити синхронізацію і доступ звідусіль. Хмарна інфраструктура дозволяє масштабувати обробку для тисяч користувачів одночасно, збирати неанонсовані статистичні дані для поліпшення алгоритмів, а також інтегруватися з іншими сервісами (наприклад, Endel як навик для Amazon Alexa використовує API Alexa для доступу в автомобілі). З іншого боку, деякі функції краще виконувати на рівні пристрою (on-device) – приміром, швидке реагування на зміну контексту, базове генерування шуму. Сучасні смартфони і

навушники мають спеціалізовані чіпи для обробки звуку (DSP), які можуть виконувати складні алгоритми (шумопоглинання, еквалізацію) з мінімальною затримкою і споживанням енергії. Це дозволяє, наприклад, динамічно адаптувати звучання без помітної затримки для користувача.

Незважаючи на привабливі можливості, впровадження персоналізованого звукового оточення пов’язане з низкою викликів: від захисту приватних даних користувачів до врахування етичних меж впливу на психіку і подолання технічних бар’єрів.

Персоналізація аудіо може потребувати збору чутливих даних: біометричних (серцевий ритм, рівень стресу), поведінкових (режим дня, улюблена музика) та контекстуальних (місцеположення, оточуючі звуки). Такі дані належать до приватної сфери користувача, і неправильне їх використання може порушити довіру. Розробники повинні дотримуватися принципів privacy by design: запитувати у користувача лише ті дані, що дійсно потрібні для функціонування сервісу, і прозоро пояснювати, як вони будуть використовуватися. Наприклад, Endel у своїх умовах чітко вказує, що доступ до даних Apple Health (пульсу, активності) надається лише за згодою користувача і виключно для цілей створення персоналізованого звукового середовища[17].

Також слід врахувати, що постійний аудіомоніторинг (через мікрофон) може сприйматися як втручання в приватність, навіть якщо робиться з благими намірами (для маскування шумів). Користувач має мати змогу відключити такі функції або задати їм чіткі межі (наприклад, режим “не слухати мене” на час розмов чи зустрічей). Зростає тенденція, що споживачі готові відмовитися від деякої глибини персоналізації заради збереження конфіденційності своїх даних. За опитуваннями, 63% користувачів вважають за краще менше персоналізований досвід, але з кращим захистом приватності.

Персоналізоване звукове оточення – це свого роду втручання в психічний та емоційний стан користувача. Постає питання етичних меж такого втручання. З одного боку, коли користувач свідомо встановлює додаток для покращення свого фокусу чи сну, ми маємо його імпліцитну згоду на вплив звуку. Але навіть у цьому

випадку варто уникати надмірної маніпуляції. Поінформована згода – ключовий принцип: користувач повинен розуміти, як звук впливає на нього. Наприклад, якщо додаток використовує бінауральні біти для зміни стану мозку, добре б це було пояснено (і, можливо, надано застереження для людей з епілепсією чи іншими станами, яких може торкнутися зовнішня стимуляція мозкових хвиль)[18].

Персоналізація аудіо в комерційних продуктах не повинна використовуватися для прихованих цілей, окрім заявленої (покращення самопочуття, комфорту). Наприклад, було б неетично, якби платформа раптом почала змінювати звук так, щоб впливати на покупцьку здатність або рішення користувача (цей напрям відомий як нейромаркетинг, і хоча прямих прикладів зі звуком немає, теоретично можна уявити, що додаток, знаючи що користувач в e-commerce додатку, спробує зробити звук, який підштовхує до імпульсивних дій). Такі перетини функціональності мають бути чітко заборонені політикою продукту. Звуковий помічник повинен лишатися на службі інтересів користувача, а не третіх сторін[19].

Персоналізоване звукове оточення – це крок до більш гуманного програмного забезпечення, яке пристосовується до людини, а не змушує людину пристосовуватися до нього. Правильно застосоване, воно може перетворити буденні цифрові взаємодії на більш продуктивні, приємні та гармонійні для користувача, підтверджуючи відому ідею: хороший дизайн – це той, що непомітно піклується про ваш комфорт, навіть через такі тонкі матерії, як звук.

1.2 Аналіз аналогічних розробок

Noisli являє собою програмне забезпечення, орієнтоване на створення індивідуального звукового оточення для підвищення продуктивності, концентрації або релаксації користувача (див. рисунок 1.1).

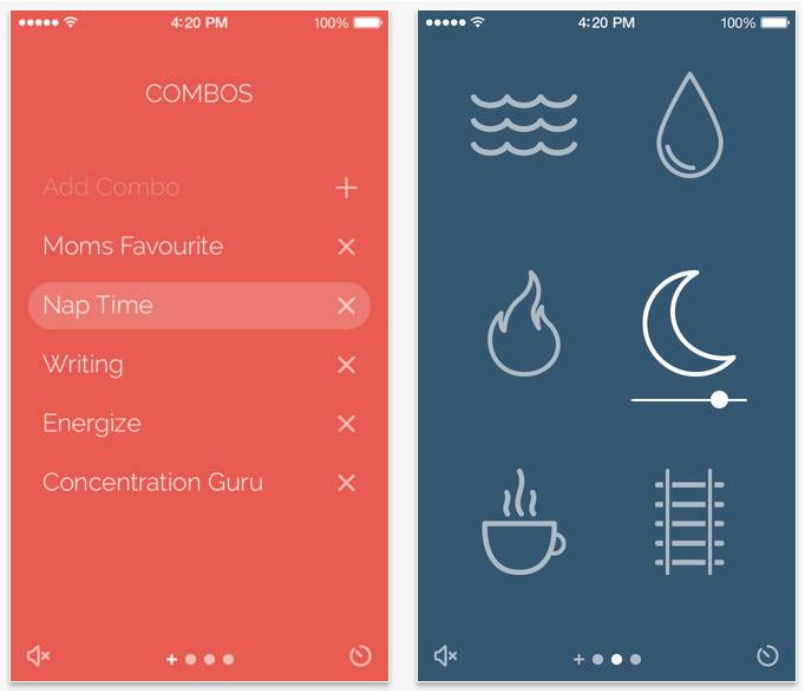


Рис.1.1 Інтерфейс Noisli

Система функціонує на основі попередньо зібраної бібліотеки звуків, яка включає різноманітні фонові аудіоелементи — такі як шум дощу, грому, вітру, шелесту листя, потріскування багаття, гулу поїзда, кафе-бара чи білого шуму. Користувач самостійно обирає декілька звуків із доступного переліку та за допомогою інтуїтивного інтерфейсу створює комбінації з регульованою гучністю кожного компонента. Система дозволяє зберігати такі мікси як профілі для подальшого використання, а також надає таймер для обмеження тривалості звукової сесії, що є корисним під час роботи з техніками помодоро чи медитації[20].

З технічної точки зору, робота Noisli не передбачає генерації звуку в реальному часі або на основі сенсорних даних користувача. Звукові фрагменти мають форму лупованих аудіофайлів, що синхронно або асинхронно програються у фоновому режимі в браузері або мобільному застосунку. Це спрощує архітектуру платформи і забезпечує високу стабільність відтворення, але одночасно накладає обмеження на варіативність звучання — оскільки джерела є сталими, треки можуть з часом викликати звикання або втрату ефективності впливу. Крім того, система не враховує контекстні чинники, як-от час доби, активність користувача чи зовнішній

шумовий фон, що знижує рівень адаптивності звукового середовища до потреб конкретної ситуації.

Серед переваг Noisli слід виокремити простоту використання, стабільність функціонування на різних пристроях, кросплатформену доступність та можливість створення базових індивідуальних композицій. Інтерфейс платформи мінімалістичний і не перевантажений другорядними функціями, що дозволяє сконцентруватися саме на створенні ментально комфортного середовища. Особливо позитивно сприймається можливість змішування декількох джерел одночасно та налаштування гучності кожного, що дає відчуття контролю і наближеності до інструментів професійного аудіодизайну.

Водночас до недоліків варто віднести відсутність динамічного навчання або побудови профілю користувача, що обмежує персоналізацію лише ручним втручанням. Також бібліотека звуків є закритою, тобто неможливо імпортувати власні звукові файли чи використовувати зовнішні джерела для розширення. Це накладає обмеження як для творчих, так і для наукових або терапевтичних сценаріїв використання. Загалом, Noisli є ефективним базовим інструментом для короткотривалого поліпшення акустичного середовища, проте з позиції сучасних підходів до персоналізованих систем його потенціал обмежується відсутністю глибшої інтеграції з поведінковими чи контекстними даними користувача.

Endel — це інноваційна звукова система, яка поєднує методи генеративного аудіо з адаптивними алгоритмами на основі штучного інтелекту для створення динамічного персоналізованого звукового середовища (див. рисунок 1.2).

Рис.1.2 Інтерфейс Endel

Основна концепція цієї системи полягає в тому, щоб у реальному часі генерувати звукові ландшафти, які сприяють досягненню певного когнітивного чи фізіологічного стану користувача — зокрема, релаксації, концентрації, покращення сну або зниження стресу. Система працює на основі набору контекстних даних,

серед яких можуть бути година доби, геолокація, погодні умови, серцевий ритм або дані про рухову активність користувача, отримані через інтеграцію з мобільними пристроями або фітнес-браслетами. Після обробки цих вхідних параметрів спеціалізований алгоритм створює унікальний аудіопотік, який не є попередньо записаним, а формується алгоритмічно в реальному часі з урахуванням заданих цілей впливу на свідомість слухача.

До значущих переваг цієї платформи варто віднести високу ступінь автоматизації процесу формування аудіо, що знімає з користувача потребу в ручному налаштуванні параметрів і дозволяє зосередитися на діяльності, для якої створюється аудіосупровід. Endel є кросплатформеним рішенням, що підтримується на мобільних пристроях, десктопах та смарт-колонках, і має офіційні інтеграції з провідними стримінговими сервісами, зокрема Spotify та Apple Music. Окрім того, платформа сертифікована науковими дослідженнями, які підтверджують її позитивний вплив на концентрацію та якість сну. Естетичне оформлення інтерфейсу також заслуговує на увагу: воно лаконічне, інтуїтивно зрозуміле та не відволікає від основного користувацького сценарію.

Разом із тим, незважаючи на високотехнологічну реалізацію, платформа має певні функціональні обмеження. Зокрема, користувач позбавлений можливості повноцінно впливати на вибір аудіоелементів або структуру звукових шарів. Увесь процес відбувається непрозоро — система самостійно приймає рішення щодо звучання, що унеможливлює індивідуальну творчість чи специфічну кастомізацію. Також відсутність доступу до власної бібліотеки звуків чи можливості її розширення створює обмеження для користувачів із професійною або художньою мотивацією, які бажають інтегрувати у звуковий простір власні аудіоелементи. Крім того, значна частина функціональності доступна лише в преміум-версії, що обмежує вільне використання платформи широким колом аудиторії.

Таким чином, Endel виступає як приклад сучасної системи генеративного аудіо з фокусом на автоматизацію та адаптацію до індивідуального контексту користувача. Її сильними сторонами є глибока інтеграція з біометричними джерелами даних, висока якість звуку та підтримка різних сценаріїв застосування,

проте в той самий час відчутна закритість і обмежена взаємодія з внутрішньою логікою створення звуку знижують її привабливість для тих, хто прагне активної участі у формуванні власного звукового середовища.

Платформа myNoise являє собою інтерактивну систему генерації звукових ландшафтів, призначену для створення індивідуального аудіооточення відповідно до потреб користувача (див рисунок 1.3).

Рис.1.3 Інтерфейс myNoise

Її функціонування базується на використанні спектрально зважених шумів та багатошарових аудіотреків, які можуть бути одночасно відтворені з окремим керуванням гучності кожного каналу.

Кожен трек є результатом студійного запису з високою частотою дискретизації, що забезпечує виняткову якість звуку та дозволяє уникнути повторюваності, характерної для типових петльових записів.

У центрі інтерфейсу знаходиться консоль налаштування, де представлено декілька повзунків, кожен з яких відповідає за окремий компонент композиції, наприклад, звук дощу, вітру, електричного гулу або пташиного співу.

Завдяки цьому користувач може вручну сформувати власний аудіоспектр, адаптований до певної мети — релаксації, медитації, концентрації, сну або маскування фонових шумів.

Перевагою системи є її висока варіативність і гнучкість. Розробник платформи, професійний інженер звуку, забезпечив великий каталог генераторів шумів, кожен з яких містить унікальну структуру звукових хвиль.

Окрім традиційних білих і рожевих шумів, у myNoise реалізовано моделі біхармонічного шуму, природних саундскейпів і штучно синтезованих композицій з глибокою деталізацією частот.

Користувачі мають змогу не лише створювати індивідуальні профілі, але й синхронізувати роботу генераторів із зовнішніми пристроями, зокрема для використання у студіях або терапевтичних кабінетах. До того ж, система працює в режимі офлайн, що забезпечує стабільну функціональність без необхідності постійного підключення до інтернету. myNoise також вирізняється адаптивністю до аудіовикликів: у деяких генераторах передбачено режим автоматичного калібрування, який підлаштовує звуковий фон до акустичних характеристик навколишнього середовища за допомогою мікрофона користувача.

Попри функціональну насиченість, платформа має ряд обмежень. Найсуттєвішим недоліком є повна відсутність автоматичної персоналізації на основі історії використання або машинного навчання — уся адаптація здійснюється вручну, без підтримки профілю поведінки чи динамічного аналізу контексту. Це означає, що користувачеві необхідно кожного разу самостійно добирати параметри відповідно до нових умов, що знижує ефективність використання у мобільних або багатозадачних сценаріях. Іншим обмеженням є відсутність підтримки розширених API або інтеграцій із зовнішніми платформами, що ускладнює впровадження myNoise у складніші системи, наприклад, розумне середовище або персоналізовані медіаплеєри.

Також можна зазначити, що попри широкий асортимент генераторів, частина з них є платними, що частково знижує доступність повного функціоналу для неплатоспроможного користувача.

Таким чином, myNoise може розглядатися як один із найбільш досконалих інструментів для створення ручного персоналізованого звукового оточення з високою якістю і деталізацією.

Проте обмеження в автоматизації процесів персоналізації та інтеграції обумовлюють потребу в подальшому розвитку програмних засобів у цьому напрямі, зокрема із залученням інтелектуальних технологій аналізу вподобань користувача та контекстної адаптації.

1.3 Постановка задачі

Проєктування програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків вимагає чіткої постановки задачі, що визначає межі дослідження, його функціональні орієнтири та архітектурні обмеження, необхідні для формування технічного завдання.

В умовах стрімкого зростання попиту на індивідуалізовані цифрові рішення, які сприяють підвищенню когнітивної продуктивності, поліпшенню психоемоційного стану або створенню комфортного акустичного середовища в умовах багатозадачності, актуальним постає створення програмної системи, що дозволяє не лише обирати попередньо підготовлені звукові треки, а й формувати адаптивні звукові профілі, орієнтовані на конкретного користувача та його змінні потреби.

Проблематика полягає в тому, що більшість наявних рішень або не дозволяють здійснювати гнучке налаштування звукових композицій, або мають закриту архітектуру, що унеможливлює підключення власних звуків чи формування сценаріїв динамічного відтворення в реальному часі.

Тому ставиться задача розробити модульне, розширюване програмне забезпечення, яке включатиме графічний інтерфейс користувача для взаємодії з бібліотекою звуків, механізм формування звукових композицій із регульованими параметрами (гучність, тривалість, циклічність, спектральне обважнення), підтримку попереднього збереження та імпорту профілів, а також можливість

інтеграції з датчиками чи календарем для реалізації персоналізованих сценаріїв.

Додатково вимагається забезпечити кросплатформеність рішення, що дозволить використовувати його як на стаціонарних комп’ютерах, так і на мобільних пристроях.

Особлива увага приділяється структурі бібліотеки звуків, яка має бути відкритою до доповнення, класифікованою за жанрами, функціональними цілями та спектральними характеристиками.

Система повинна реалізовувати мінімальне базове ядро з можливістю розширення додатковими модулями, що забезпечить її масштабованість і адаптацію під конкретні вимоги користувачів.

2 ПРОЄКТУВАННЯ СИСТЕМИ ЗВУКОВОГО ОТОЧЕННЯ

2.1 Архітектура системи

Архітектура системи представлена на рисунку 2.1.

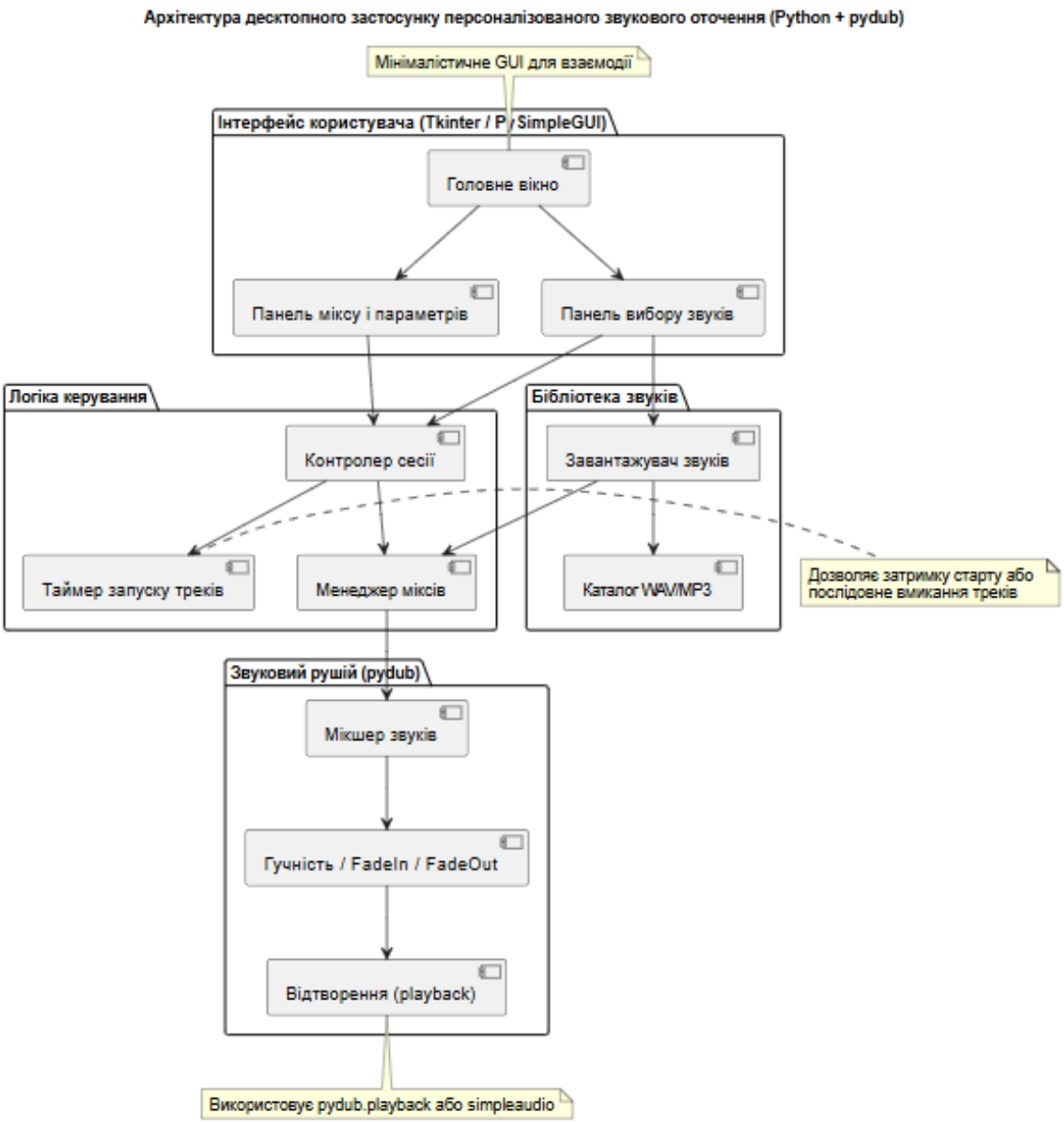


Рис.2.1 Архітектура системи персоналізованого звукового оточення

Інтерфейс користувача в системі персоналізованого звукового оточення розроблено з урахуванням принципів мінімалізму, інтуїтивної зрозумілості та ефективності. Він є графічною оболонкою, що слугує посередником між користувачем і функціональним ядром програми, дозволяючи здійснювати контроль над усіма ключовими параметрами створення індивідуального аудіофону. Візуальна структура інтерфейсу побудована так, щоб користувач мав прямий доступ до базових елементів — вибору звуків, конфігурації міксів, регулювання гучності та налаштування послідовності та часу запуску треків — без необхідності проходити багатоетапні меню чи використовувати технічні терміни. Перевага надається прямим маніпуляціям: можливість перетягування звуків у мікс, налаштування повзунків гучності, вибір часу запуску за допомогою простого годинника або таймера.

Інтерфейс поділений на умовні функціональні зони, кожна з яких відповідає за окрему категорію дій. Основна панель відображає список доступних звуків з локальної бібліотеки у вигляді інтерактивних елементів, що можуть бути додані до міксу шляхом натискання або перетягування. Поруч розміщується область мікшування, де вже обрані треки представлено у вигляді окремих каналів, кожен з яких має регулятор гучності, перемикач активності та, за потреби, таймер для налаштування часу початку або завершення звучання. У нижній частині екрана може бути реалізовано елемент управління загальним відтворенням: кнопки "Старт", "Пауза", "Зупинити", а також індикатор поточного стану.

Спрощений дизайн інтерфейсу не передбачає складної графіки чи анімації, зберігаючи фокус на функціональності й швидкій реакції системи на дії користувача. Враховується потреба у тривалій роботі без зорового перевантаження, тому кольорова гама може бути нейтральною, з акцентами лише на активних елементах. Окрему увагу приділено стабільності масштабування, що забезпечує коректне відображення інтерфейсу на різних розширеннях екрана — від стандартних моніторів до високих роздільностей. Таким чином, інтерфейс виконує роль доступного інструменту керування, що не потребує додаткового навчання або

технічної підготовки, забезпечуючи при цьому високий рівень контролю над формуванням персоналізованого звукового середовища.

Логіка керування в системі персоналізованого звукового оточення забезпечує зв’язок між інтерфейсом користувача, звуковим рушієм та бібліотекою аудіофайлів, виступаючи центральним координатором усіх дій, пов’язаних із налаштуванням і запуском звукового фону. Основною одиницею взаємодії є сеанс мікшування, в межах якого користувач формує набір звуків, визначає параметри гучності, черговості, часових відступів та ефектів переходу. Ці дії ініціюються через просту графічну оболонку, однак не обробляються безпосередньо в інтерфейсі — натомість передаються до контролера сесії, який зберігає поточний стан і конфігурацію, слідкує за послідовністю подій і формує запити до нижчого рівня — менеджера міксів і звукового рушія.

Контролер сесії працює як диспетчер: кожна подія, пов’язана з вибором звуку, зміною його порядку, запуском таймера або збереженням профілю, реєструється і передається в логіку керування. Після цього активується менеджер міксів, який формує нову конфігурацію аудіопотоку — шляхом створення комбінації аудіосегментів із зазначеною гучністю, зсувом у часі та ефектами. Кожен сегмент звуку перетворюється на об’єкт AudioSegment, після чого здійснюється їх накладання або послідовне об’єднання залежно від заданих параметрів. Якщо передбачено, що звуки мають відтворюватися з певною затримкою або у визначеній черзі, ці параметри фіксуються у вигляді відносних зміщень у часовій шкалі й передаються таймеру, який організовує запуск треків згідно з визначеним порядком.

Окреме завдання логіки керування — узгодження параметрів збереження, тобто можливість створення та редагування профілів міксів. У таких профілях фіксуються вибрані звуки, їхні індивідуальні налаштування, а також шаблони запуску. При повторному використанні профілю система відновлює параметри сеансу й автоматично передає їх у менеджер міксів без необхідності повторного ручного налаштування. Завдяки цьому реалізується швидкий доступ до раніше створених аудіоландшафтів.

У структурі логіки керування особлива роль відводиться таймеру — окремому модулю, що відповідає за реалізацію часу запуску звуків, підтримку пауз, циклічне відтворення та синхронізацію різних треків у межах одного міксу. Цей модуль працює в асинхронному режимі, забезпечуючи паралельну активацію декількох звуків або їх поетапне програвання без затримок. Його робота тісно інтегрована з потоком аудіовиходу, що дозволяє зберігати стабільність і уникати перевантаження в системах з обмеженим обчислювальним ресурсом.

Таким чином, логіка керування виконує роль центрального координаційного шару, який пов’язує усі компоненти системи та забезпечує узгоджену, гнучку й керовану генерацію персоналізованого звукового середовища відповідно до дій користувача. Усі взаємодії реалізуються у формі чітких внутрішніх запитів між модулями, що дозволяє в подальшому масштабувати систему, додаючи нові джерела звуку або розширюючи набір параметрів без істотної зміни загальної архітектури.

Звуковий рушій системи персоналізованого звукового оточення виконує функцію обробки, мікшування та відтворення аудіосигналів відповідно до параметрів, заданих користувачем через інтерфейс. Його реалізацію побудовано на використанні бібліотеки pydub, яка дозволяє здійснювати високорівневу маніпуляцію аудіо у форматах WAV, MP3 та інших, а також застосовувати ефекти затухання, регулювання гучності та комбінування звукових фрагментів у реальному часі. У межах архітектури система функціонує таким чином, що кожен окремий звуковий трек у форматі аудіофайлу спочатку імпортується у вигляді об’єкта типу AudioSegment, після чого він може бути модифікований або об’єднаний з іншими треками згідно із заданим порядком, рівнем гучності та часовими параметрами. Це забезпечує можливість створення складних композицій із кількох джерел звуку з високою точністю контролю над характеристиками кожного шару.

Особливістю реалізації є принцип незалежного попереднього налаштування кожного звукового елемента. Це означає, що перед об’єднанням у мікс кожен трек обробляється окремо — до нього можуть бути застосовані ефекти поступового

зникання (fade out), зростання гучності (fade in), або ж абсолютного зниження чи підсилення амплітуди. Задля досягнення плавності звучання усі ці параметри задаються програмно, з можливістю тонкого калібрування вручну або автоматично відповідно до сценаріїв, визначених користувачем. При побудові композиції звуковий рушій здійснює послідовне або паралельне об'єднання аудіо, використовуючи методи overlay або append, які дозволяють або накладати треки один на одного, або відтворювати їх послідовно з часовими паузами. Такий підхід дає змогу реалізовувати як одношарові, так і багатошарові звукові ландшафти, що особливо актуально для відпочинку чи медитації, де важливими є як спектральна щільність, так і тимчасова структура звучання.

Фінальний етап роботи звукового рушія полягає у відтворенні сформованої звукової сцени. Для цього використовуються методи play з бібліотеки pydub.playback або альтернатива на основі simpleaudio, що забезпечує синхронне виведення аудіо без затримок. У випадку задіяння таймерів чи зовнішніх тригерів, звуковий рушій також може ініціювати поступове вмикання треків у відповідності до запрограмованої черговості або часу доби. Це дозволяє реалізувати адаптивне звукове середовище, що змінюється динамічно, відповідно до зовнішнього контексту або потреб користувача.

У цілому, звуковий рушій виступає критичним функціональним компонентом системи, який забезпечує повний цикл роботи з аудіо — від завантаження та обробки окремих звуків до їх комплексного синтезу та виводу. Його гнучкість і інтеграція з логікою сценарного управління дозволяє створити дійсно індивідуалізоване звукове середовище, яке можна підлаштовувати під конкретні психофізіологічні стани користувача, не втрачаючи при цьому стабільності та якості відтворення.

2.2 Інструментальні засоби розробки

У процесі розробки застосунку для створення персоналізованого звукового оточення було використано інструментальні засоби, які забезпечують ефективне

виконання всіх етапів — від побудови графічного інтерфейсу до обробки аудіосигналів та управління логікою мікшування. Основною платформою розробки виступає мова програмування Python, яка завдяки своїй високій читабельності, активному екосередовищу бібліотек і широким мультимедійним можливостям, дозволяє гнучко реалізувати як інтерфейсну, так і обчислювальну частини програми[21]. Для реалізації графічного інтерфейсу користувача було обрано бібліотеку Tkinter — стандартний GUI-фреймворк для Python, який забезпечує базовий рівень візуального представлення компонентів та є достатнім для побудови мінімалістичного інтерфейсу без потреби в зовнішніх залежностях[22]. Завдяки Tkinter було реалізовано головне вікно програми, панелі вибору звуків, елементи керування гучністю та кнопки запуску, що забезпечують інтуїтивну взаємодію користувача з додатком[23].

Ключовим інструментом для роботи зі звуком стала бібліотека pydub, яка надає зручні інтерфейси для імпорту, обробки, комбінування та експорту аудіо у форматах WAV, MP3, FLAC та інших[24]. Її основною перевагою є можливість маніпуляції аудіофайлами як із математичної точки зору (зміна гучності, обрізання, додавання ефектів), так і з композиційної (накладання треків, формування послідовностей, об'єднання звукових фрагментів у складні структури)[25]. Разом із pydub у програмі використано simpleaudio як бекенд для точного, синхронного відтворення сформованих аудіотреків, що забезпечує стабільну роботу на більшості операційних систем без складного конфігурування. Крім того, при реалізації механізму таймера запуску і черговості програвання було задіяно модуль threading, який дозволяє організовувати незалежні потоки виконання для різних елементів звукової сцени, уникнувши конфліктів у головному інтерфейсі[26].

Для організації структури проєкту та його локального тестування використовувалося середовище розробки Thonny IDE, яке є зручним для створення середніх за складністю десктопних програм, особливо при роботі з GUI та аудіо[27]. Водночас для полегшення управління файлами та створення локальної звукової бібліотеки була розроблена проста система каталогів, у якій користувач може зберігати власні аудіофайли для подальшої обробки[28]. Цей підхід дозволяє

уникнути складної інтеграції з базами даних, зберігаючи простоту та доступність рішення[29]. За потреби розширення функціоналу система залишається відкритою до інтеграції з більш складними сервісами, зокрема планувальниками подій, біометричними датчиками чи навіть мобільними адаптаціями, але поточна реалізація цілеспрямовано зосереджена на простому, локальному й ефективному використанні з мінімальними апаратними вимогами[30]. Таким чином, обрані інструментальні засоби дозволили досягти необхідної функціональності системи з високим рівнем контролю над якістю звуку та можливістю персонального налаштування.

2.3 Вибір та обгрунтування бази даних

У рамках створення програмного забезпечення для формування персоналізованого звукового оточення особливу увагу було приділено питанню збереження налаштувань користувача, параметрів звукових міксів та структури бібліотеки аудіофайлів. Попри відсутність потреби у масштабованому сховищі для великих обсягів даних, виникла необхідність у зберіганні конфігурацій профілів, шляху до звукових файлів, параметрів гучності, черговості програвання та налаштувань таймерів для кожної сесії. Це обумовило потребу у впровадженні простої, легкоінтегровної бази даних, яка не потребує окремого сервера, може працювати автономно в межах десктопного середовища та підтримує прості операції читання-запису з невеликим обсягом структурованої інформації. З огляду на ці вимоги, було обрано вбудовану реляційну базу даних SQLite, яка забезпечує оптимальний баланс між простотою реалізації, відмовостійкістю та підтримкою стандартних засобів запиту мовою SQL.

Рішення про використання саме SQLite обґрунтовується декількома ключовими перевагами. По-перше, ця СКБД не потребує розгортання серверної частини, що значно спрощує інсталяцію застосунку та виключає необхідність додаткового адміністрування з боку користувача. Усі дані зберігаються у єдиному локальному файлі, доступ до якого здійснюється безпосередньо з основного Python-коду за допомогою стандартного модуля sqlite3. Це дозволяє уникнути

зайвих зовнішніх залежностей, підтримуючи мінімалістичність програмного стеку. По-друге, структура даних у SQLite є достатньо гнучкою для зберігання об’єктів, що використовуються в системі, зокрема метаданих про звукові треки (назва, категорія, тривалість, шлях до файлу), параметрів профілів (рівень гучності, таймінги, активні шари міксу), а також інформації про останні сесії користувача. По-третє, висока швидкодія при роботі з невеликими обсягами даних та можливість використання транзакцій дозволяє гарантувати цілісність записів навіть у разі неочікуваного завершення роботи програми або конфліктів між потоками.

Таким чином, вибір SQLite як базового інструмента для збереження користувацьких даних та конфігурацій є технічно виваженим, відповідає вимогам автономної десктопної програми і дозволяє без ускладнень реалізувати базову функціональність збереження налаштувань, історії сеансів та структури аудіобібліотеки. Завдяки цьому забезпечується збереження персоналізації навіть після перезапуску системи, що критично важливо для користувацького досвіду в застосунках такого типу. За потреби масштабування або переходу до мережевої синхронізації налаштувань із хмарним середовищем структура, закладена в SQLite, може бути мігрувана до складніших рішень, таких як PostgreSQL або Firebase, проте для поточної стаціонарної реалізації саме SQLite виявляється найбільш раціональним і продуктивним варіантом.

Структура бази даних представлена на рисунку 2.2.

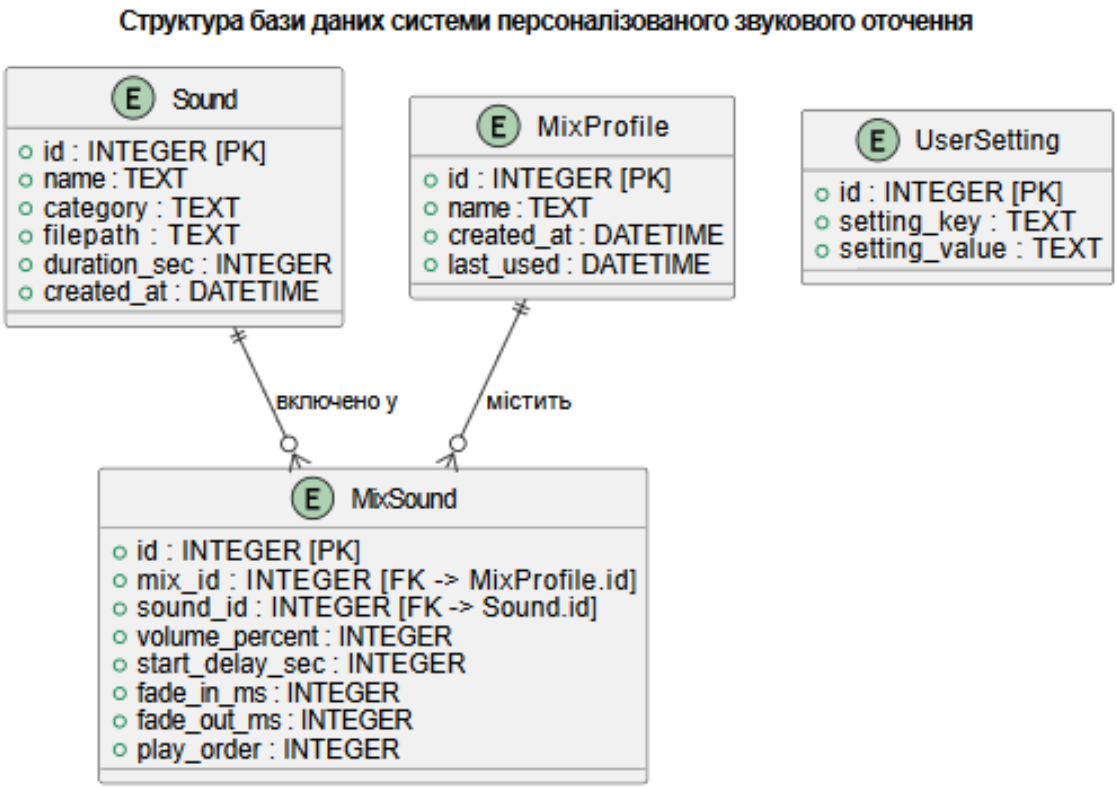


Рис.2.2 Структура бази даних

Структура бази даних системи персоналізованого звукового оточення побудована за реляційною моделлю з урахуванням потреб зберігання як постійних аудіоресурсів, так і динамічних конфігурацій, що формуються користувачем у процесі взаємодії із застосунком.

Основним об’єктом зберігання є звукові файли, представлені у вигляді записів таблиці `Sound`, де фіксується унікальний ідентифікатор, назва звуку, категорія (яка може відображати тип — наприклад, природне середовище, шумове тло, електронна атмосфера тощо), шлях до фізичного файлу на диску, тривалість у секундах і дата додавання. Це дозволяє забезпечити структурування та ефективний доступ до звукової бібліотеки, що є основою формування міксів.

Формування індивідуальних композицій здійснюється через механізм збереження міксів у таблиці `MixProfile`, яка містить унікальний ідентифікатор профілю, його назву, дату створення та дату останнього використання. Кожен

профіль є логічною одиницею, в межах якої задається конфігурація — набір звуків і параметрів їх відтворення, що формують цілісне звукове середовище.

Для реалізації зв’язку між профілем і конкретними звуками використовується проміжна таблиця `MixSound`, яка виконує роль багатозначного відображення й дозволяє включати до одного міксу кілька треків. У цій таблиці зберігається вказівка на профіль і звуковий файл, а також параметри — рівень гучності у відсотках, затримка старту в секундах, час наростання гучності (fade-in), час затухання (fade-out), а також черговість відтворення. Така деталізація забезпечує можливість гнучкої персоналізації кожного звуку в контексті заданої композиції.

Додатково реалізовано таблицю `UserSetting`, яка зберігає глобальні параметри взаємодії користувача з програмою у форматі ключ-значення. Це можуть бути технічні або візуальні налаштування, наприклад, останній активний профіль, тема інтерфейсу, мова, положення вікна тощо. Застосування такого підходу до налаштувань дозволяє зберігати індивідуальний користувацький досвід навіть між сесіями, не пов’язуючи ці параметри з жодною окремою аудіокомпозицією.

Уся структура забезпечує цілісне представлення логіки системи: статична бібліотека звуків, динамічно збережені мікси, гнучкі налаштування кожного звукового шару і глобальні параметри. Між таблицями `Sound` та `MixProfile` через `MixSound` встановлюється зв’язок типу багато-до-багатьох, що є необхідною умовою для повторного використання звуків у різних композиціях. Вибрана структура не лише зберігає узгодженість і гнучкість, але й забезпечує хорошу масштабованість у межах локального застосунку на основі SQLite.

2.4 Алгоритми роботи системи

Алгоритм завантаження звукової бібліотеки представлено на рисунку 2.3.

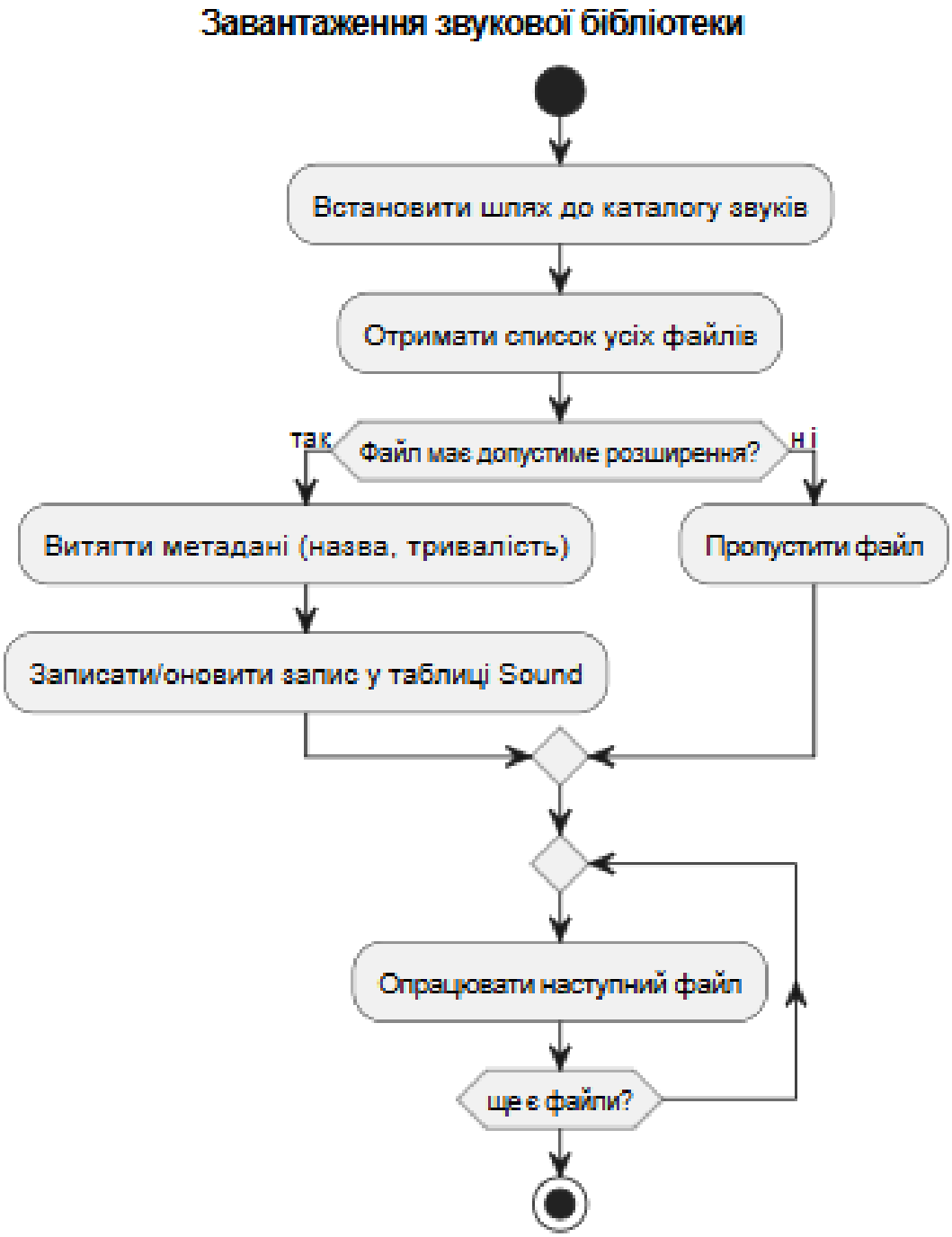


Рис.2.3 Алгоритм завантаження звукової бібліотеки

Алгоритм завантаження звукової бібліотеки у системі персоналізованого звукового оточення реалізує процедуру ініціалізації внутрішнього сховища аудіофайлів шляхом сканування файлової системи, фільтрації релевантних даних, обробки метаданих та інтеграції з базою даних. На першому етапі здійснюється встановлення кореневого каталогу, в межах якого очікується наявність звукових файлів. Цей каталог може бути визначений за замовчуванням або обраний користувачем вручну. Далі здійснюється перебирання всіх об’єктів у каталозі, включаючи підкаталоги, з метою виявлення медіафайлів із розширеннями, що відповідають допустимим форматам (наприклад, .wav, .mp3, .flac). Кожен файл, який задовольняє цю умову, передається до обробника, що здійснює витяг метаданих: визначається тривалість аудіофайлу, зчитується назва, визначається його розмір та, за можливості, базові характеристики — частота дискретизації, кількість каналів тощо.

Після отримання метаінформації здійснюється спроба запису або оновлення відповідного запису в таблиці Sound реляційної бази даних. Якщо файл уже присутній у базі — його атрибути оновлюються, у протилежному разі створюється новий запис. Цей механізм гарантує цілісність структури бібліотеки та дозволяє синхронізувати її з поточним вмістом файлової системи без необхідності повного очищення даних. У разі виявлення неформатованого або пошкодженого файлу система ігнорує його та переходить до наступного. Для реалізації такої поведінки у коді застосовується обробка виключень, що дозволяє уникати збоїв при роботі з непередбачуваними даними. Процес завершується після повного проходження каталогу й фіксації змін у базі даних.

Таким чином, алгоритм завантаження бібліотеки виконує не лише фізичне зчитування аудіофайлів, але й забезпечує логічну інтеграцію звукових ресурсів у внутрішню модель застосунку, синхронізуючи її з базою даних для подальшого доступу, аналізу, мікшування та збереження індивідуальних звукових профілів. Його стабільність та ефективність є критично важливими для роботи системи, оскільки саме з цієї процедури розпочинається формування звукового середовища, на яке орієнтовано подальшу персоналізацію.

Алгоритм формування міксу представлено на рисунку 2.4.

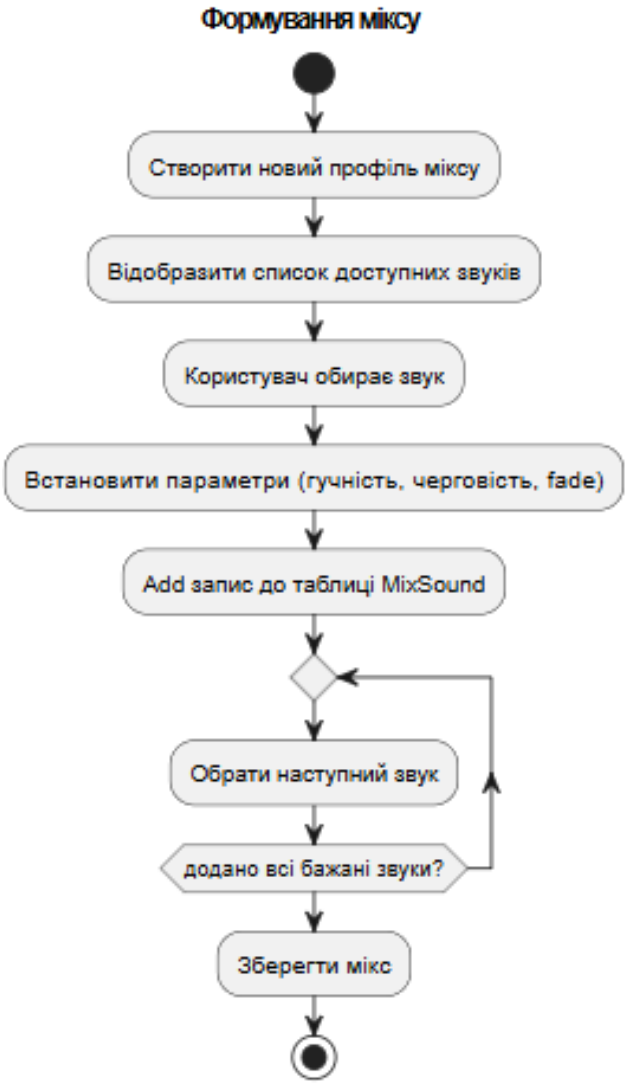


Рис.2.4 Алгоритм формування міксу

Алгоритм формування міксу в системі персоналізованого звукового оточення виконує функцію інтерактивного конструювання користувачем індивідуального аудіофону на основі доступної бібліотеки звуків. Реалізація цього алгоритму передбачає поетапну взаємодію з графічним інтерфейсом, доступ до локальної бази даних, застосування параметрів до кожного звукового шару, а також формування структурованої конфігурації, придатної до збереження та подальшого відтворення.

Алгоритм активується в момент ініціалізації створення нового профілю міксу, після чого користувачу пропонується інтерфейс вибору наявних

аудіофайлів, імпортованих до системи раніше. Вибір одного або кількох звуків автоматично породжує тимчасові об'єкти, у яких відображаються ключові налаштування, зокрема рівень гучності, тривалість затухання і зростання (fade out/in), затримка початку програвання, а також бажана черговість у загальному міксі.

На кожному етапі вибору звуку користувач має змогу взаємодіяти з візуальними повзунками або полями введення, що дозволяє сформувати точну акустичну модель для кожного шару. У контексті реалізації це означає, що для кожного обраного треку створюється відповідний запис у структурі MixSound, який містить зовнішній ключ до основного звуку, посилання на активний мікс (MixProfile), а також числові параметри, що регулюють поведінку треку в момент програвання. Черговість відтворення є визначальним параметром при подальшій побудові міксу, оскільки саме вона визначає послідовність накладання звукових шарів або їх паралельне звучання.

Після завершення вибору користувач ініціює збереження сформованого міксу, внаслідок чого відбувається створення нового запису у таблиці MixProfile, а всі відповідні звукові об'єкти, пов'язані з ним, фіксуються у MixSound. Ця дія забезпечує цілісність міксу як об'єкта, що поєднує метаінформацію, вибрані аудіофайли та їх параметризацію.

Таким чином, результат виконання алгоритму — це збережена композиційна структура, яка може бути у подальшому відтворена у звуковому рушії з дотриманням усіх заданих параметрів. Завдяки цьому користувач отримує змогу створювати унікальні акустичні середовища, що відповідають його емоційному чи когнітивному стану, а система реалізує повноцінну підтримку персоналізації без втрати простоти використання.

Алгоритм відтворення міксу представлено на рисунку 2.5.

Рис.2.5 Алгоритм відтворення міксу

Алгоритм відтворення міксу у системі персоналізованого звукового оточення реалізує логіку обробки та відтворення звукових композицій, сформованих користувачем на основі заздалегідь обраних аудіофайлів і встановлених для них параметрів. Основна мета цього алгоритму полягає в забезпеченні послідовного або паралельного програвання звукових доріжок відповідно до заданих налаштувань, зокрема таких, як черговість, затримка старту, гучність, тривалість ефектів плавного зникнення (fade-out) та зростання (fade-in). Алгоритм починається з вибору користувачем одного з попередньо створених міксів, що

зберігаються у базі даних. Після цього система звертається до таблиці зв’язків між профілем і окремими звуками, де фіксуються всі параметри відтворення. Дані витягуються, упорядковуються за визначеною черговістю (play_order), яка задає послідовність їх програвання.

Кожен запис, пов’язаний із конкретним звуковим елементом, містить інструкцію щодо часу затримки перед запуском, рівня гучності та тривалості ефектів. У процесі реалізації ці параметри застосовуються до відповідного об’єкта типу AudioSegment, створеного засобами бібліотеки pydub. Для коректного відтворення затримки реалізується блокування потоку виконання на заданий час перед початком програвання кожного треку, що дозволяє досягти передбачуваного звучання відповідно до сценарію користувача. Після цього аудіосегмент модифікується шляхом застосування методів fade_in, fade_out, а також масштабування амплітуди для досягнення бажаного рівня гучності. Сформовані звукові об’єкти можуть накладатися один на одного у разі, якщо параметри не передбачають послідовного звучання, що забезпечується методами overlay та асинхронного запуску в окремих потоках. У випадку послідовного відтворення система чекає завершення одного звуку перед запуском наступного.

Після завершення обробки всіх звуків, об’єднаних у межах одного міксу, звуковий рушій передає аудіопотік на модуль відтворення, який може бути реалізований або через pydub.playback, або через сумісний бекенд типу simpleaudio, що підтримує точне управління таймінгами та дозволяє уникнути артефактів або затримок. Завдяки застосуванню незалежних потоків у разі паралельного звучання, досягається ефект реального звукового середовища, в якому звуки природно нашаровуються, утворюючи атмосферну композицію. Алгоритм завершується після завершення останнього активного треку, або у разі дострокового завершення сесії користувачем. Завдяки гнучкості параметрів і точності контролю над звуковим потоком, дана реалізація алгоритму дозволяє ефективно адаптувати звукове середовище під індивідуальні потреби — навчання, релаксацію чи підвищення концентрації.

3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Розробка інтерфейсу користувача

Процес розробки інтерфейсу користувача було розпочато з детального аналізу функціональних вимог, який передбачав створення простого у використанні вікна, що об’єднує механізми вибору аудіофайлів, керування відтворенням та формування міксів. На початковому етапі було визначено, що основним графічним фреймворком стане Tkinter — стандартна бібліотека Python, яка забезпечує швидкий і легкий старт без необхідності додаткової інсталяції. Першим кроком реалізації стало створення каркасу головного вікна з базовим викликом конструктора:

```
root = Tk()
root.title("Персоналізоване звукове оточення")
root.geometry("800x600")
```

В цьому блоці коду відбулось визначення заголовка вікна та його розмірів, що гарантувало початкову стабільну область для розміщення всіх елементів. Вже на цьому етапі відзначалося бажання зробити вікно адаптивним, тому у властивостях застосунку заклали обробник події закриття, що дозволяє коректно зупинити усі фонові потоки та аудіо:

```
root.protocol("WM_DELETE_WINDOW", on_close)
```

де функція on_close реалізовувала зупинку відтворення та завершення головного циклу (root.destroy()).

Наступним етапом розробки стало продумування структури панелей та контейнерів, адже інтерфейс розбивався на дві головні зони: область списку треків з керуючою панеллю праворуч та область списку міксів унизу. Для розміщення цих зон було застосовано Frame із менеджером геометрії pack, що забезпечував автоматичне масштабування:

```
main_frame = Frame(root)
```

```
main_frame.pack(fill=BOTH, expand=True, padx=10, pady=10)
left_frame = Frame(main_frame)
left_frame.pack(side=LEFT, fill=BOTH, expand=True)
right_frame = Frame(main_frame)
right_frame.pack(side=RIGHT, fill=Y)
```

Саме такою комбінацією було досягнуто чітке розділення вікна на зони, які потім наповнювались елементами списків та кнопок. У лівій частині відразу було розміщено елемент Listbox для треків, до якого зв’язувалось метод populate_tracks(), що читав дані з бази та додавав рядки:

```
self.track_list = Listbox(left_frame)
self.track_list.pack(fill=BOTH, expand=True)
def populate_tracks(self):
    self.track_list.delete(0, END)
    for tid, path in db.get_tracks(user_id):
        self.track_list.insert(END, f"{tid}: {os.path.basename(path)}")
```

Цей код демонструє підхід до відображення оновленого переліку треків у відповідності до змін у базі даних.

Окрему увагу в процесі створення інтерфейсу приділено панелі кнопок праворуч. Була реалізована функція створення підрядку кнопок із фіксованою висотою та однаковим відступом, що забезпечує естетичну однорідність:

```
buttons = [
    ("Додати трек", self.add_track),
    ("Видалити трек", self.delete_track),
    ("<<15c", lambda: self.seek_by(-15)),
    ("Відтворити", self.play_current),
    ("15c>>", lambda: self.seek_by(15)),
    ("Попередній", self.prev_track),
    ("Наступний", self.next_track),
    ("Пауза/Продовжити", self.pause_resume),
]
```

```
for text, cmd in buttons:
    btn = Button(right_frame, text=text, command=cmd)
    btn.pack(fill=X, pady=2)
```

Кожна кнопка викликала відповідний метод контролера, який вже взаємодіяв із класом AudioManager, що, у свою чергу, здійснював запуск модуля pygame.mixer або pydub у фонових потоках.

Оскільки відтворення аудіо і створення міксів може займати відчутний час, було прийнято рішення винести ці операції у бекграунд–потоки за допомогою модуля threading. Наприклад, метод play() у AudioManager реалізовано таким чином:

```
def play(self, path, start_ms=0):
    threading.Thread(target=self._play_thread, args=(path, start_ms), daemon=True).start()
def _play_thread(self, path, start_ms):
    mixer.music.stop()
    mixer.music.load(path)
    mixer.music.play(loops=0, start=start_ms/1000.0)
    self.current = path
    self.start_time = time.time()
```

Це дозволило істотно знизити час відгуку інтерфейсу під час обробки команд користувача. Аналогічно було реалізовано створення міксу: після кліку «Створити мікс» відкривався діалог введення назви, а об’єднання аудіосегментів відбувалося в окремому потоці:

```
name = simpledialog.askstring("Ім'я міксу", "Введіть ім'я:")
threading.Thread(target=self._create_mix, args=(mix_id, track_ids), daemon=True).start()
def _create_mix(self, mix_id, ids):
    from pydub import AudioSegment
    segments = [AudioSegment.from_file(path) for path in paths_from_ids(ids)]
    mix = segments[0]
```

```
    for seg in segments[1:]:
        mix = mix.append(seg, crossfade=500)
    mix.export(out_path, format="mp3")
    db.update_mix_path(mix_id, out_path)
```

Це рішення уникнуло гальмування головного вікна й дозволило відразу надавати зворотний зв’язок через messagebox.showinfo.

Наступним кроком було додавання функціоналу регулювання гучності та відображення часу відтворення. Для гучності використано елемент Scale з двома підписами: «from_=0» до «to=1» і кроком «resolution=0.1». Виклик mixer.music.set_volume відбувається безпосередньо з колбека шкали:

```
self.vol_scale = Scale(right_frame, from_=0, to=1, resolution=0.1,
                       orient=HORIZONTAL, label="Гучність",
                       command=lambda v: mixer.music.set_volume(float(v)))
self.vol_scale.set(0.5)
self.vol_scale.pack(fill=X, pady=5)
```

Для відображення часових міток реалізовано метод опитування стану відтворення кожні 500 мс за допомогою root.after, що дозволяє оновлювати Label поточного часу та загальної тривалості:

```
def _update_time(self):
    pos = mixer.music.get_pos() // 1000
    dur = int(mixer.Sound(self.audio.current).get_length())
    self.elapsed_label.config(text=time.strftime("%M:%S", time.gmtime(pos)))
    self.total_label.config(text=time.strftime("%M:%S", time.gmtime(dur)))
    if mixer.music.get_busy():
        self.root.after(500, self._update_time)
    else:
        self.next_track()
```

Таке рішення гарантує синхронне оновлення індикаторів і плавність роботи без блоку відтворення.

У нижній частині вікна розміщено дві зони: список міксів та список треків обраного міксу. Для списків застосовано однаковий підхід — дві колонки Listbox у Frame, що дозволяє наочно відобразити зв’язок між головним списком і складом кожного міксу.

Натискання на елемент у лівому списку (<<ListboxSelect>>) викликало метод, який за допомогою SQL-запиту отримував ідентифікатори треків з таблиці mix_tracks та наповнював правий список відповідними рядками:

```
def on_mix_select(self, event):
    mid = int(self.mix_list.get(ACTIVE).split(":")[0])
    ids = db.get_mix_track_ids(mid)
    self.mix_tracks.delete(0, END)
    for tid, path in db.get_tracks(user_id):
        if tid in ids:
            self.mix_tracks.insert(END, f"{tid}: {os.path.basename(path)}")
```

Така структура забезпечила миттєве оновлення правого списку без потреби у перезавантаженні вікна.

Процес відлагодження інтерфейсу також включав налаштування обробки непередбачених ситуацій: захист від введення порожнього імені міксу, обробка відмови у відкритті файлу та виправлення логіки обчислення позиції відтворення при паузі.

Наприклад, для коректної роботи кнопки «Пауза/Продовжити» було реалізовано накопичувач часу паузи, що дозволяє миттєво відновити відтворення з попередньої позиції, а не з нульової:

```
def pause_resume(self):
    if not self.audio.paused:
        mixer.music.pause()
        self.audio.paused = True
        self.audio.pause_time = time.time()
    else:
```

```
mixer.music.unpause()
self.audio.paused = False
self.audio.start_time += time.time() - self.audio.pause_time
self._update_time()
```

Цей фрагмент гарантував точність відтворення без «стрибків» після паузи.

У результаті було досягнуто узгодженості між бізнес-логікою та графічним представленням: при натисканні «Додати трек» відкривається діалог filedialog.askopenfilename, після чого новий елемент автоматично з’являється в Listbox, що підтверджує успішне інтеграційне тестування шляху від вибору файлу до запису в базі.

При видаленні елемент знімається і з інтерфейсу, а при відтворенні відображаються часові лічильники і стає доступною кнопка «Наступний трек», що забезпечує циклічне перемикання плейлиста.

Усі ці етапи в сукупності забезпечили створення інтерфейсу, який поєднує простоту використання і функціональну насиченість. Використання Tkinter дозволило реалізувати чисту та лаконічну графіку, а застосування фонових потоків гарантувало високу швидкодію при обробці аудіо. Вкраплення коду ілюструє основні принципи побудови компонентів інтерфейсу, їх взаємодію з базою даних і підсистемою відтворення, що завершило цикл розробки з дотриманням принципів розширюваності й підтримуваності коду.

Завдяки такій архітектурі інтерфейс можна легко адаптувати під нові вимоги, додати підтримку додаткових форматів або інтеграцію з віддаленими сховищами, що підтверджує гнучкість реалізованого рішення.

Інтерфейс користувача представлено на рисунку 3.1.

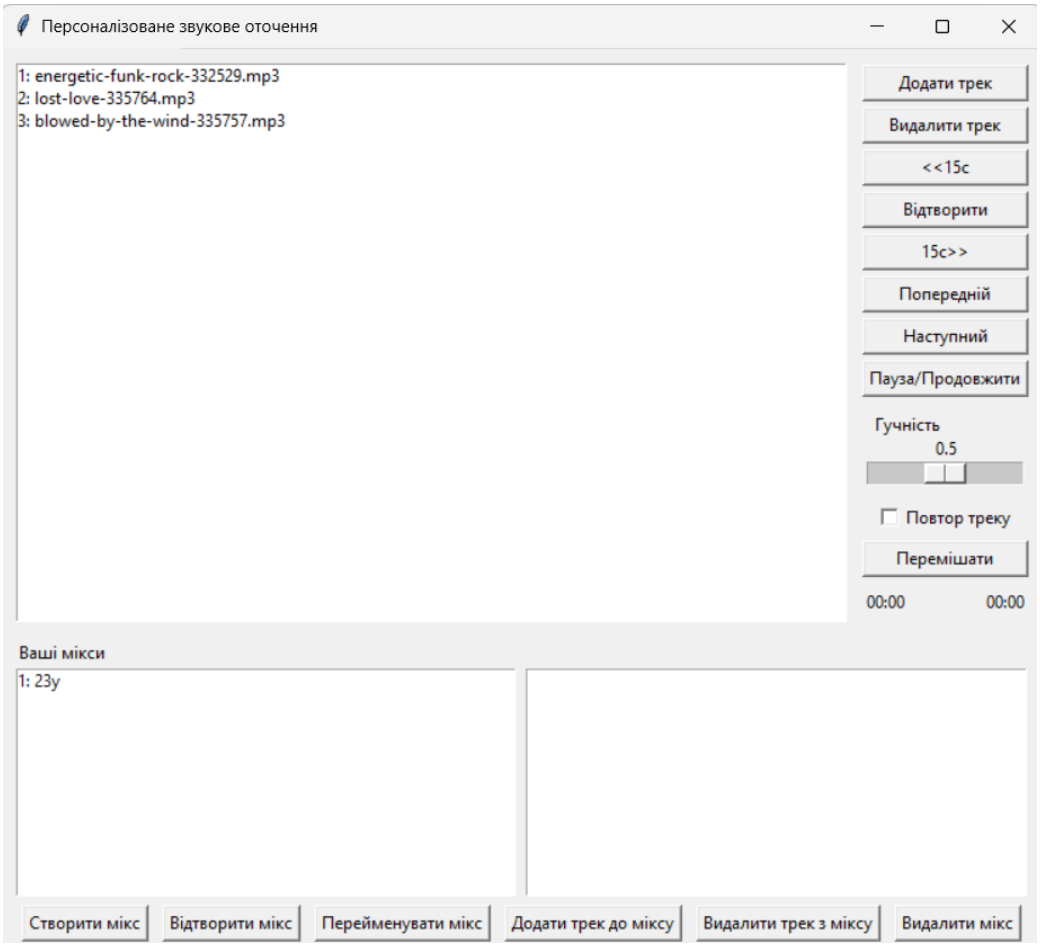


Рис.3.1 Інтерфейс користувача застосунку

Інтерфейс користувача розроблено з урахуванням принципів інтуїтивності та компактності, що забезпечує зручний доступ до базового функціоналу відтворення окремих аудіофайлів і створення комплексних міксів. Вікно програми поділено на дві основні області, причому у верхній частині розташовано великий перелік доступних треків, у якому кожний елемент маркується унікальним ідентифікатором та ім’ям файлу з розширенням, що дозволяє однозначно ідентифікувати кожен аудіофайл. Поруч із списком передбачено набір кнопок керування, який реалізовано як послідовність елементів графічного інтерфейсу однакового розміру та стилю. Перші дві кнопки відповідають за додавання файлу до локальної колекції та його видалення з бази даних програми. Наступні елементи інтерфейсу дозволяють змінювати позицію відтворення треку на кроки по п’ятнадцять секунд назад або вперед, що дає змогу оперативно повертатися до попередніх фрагментів або пропускати низькоінформативні вставки без

необхідності маніпулювати вручну часовою шкалою. Кнопка запуску відтворення активує потік відтворення через модуль pygame.mixer, надаючи можливість негайного початку відтворення на поточній позиції файлу. Подальші елементи дозволяють перемикатися між сусідніми треками у поточному списку: перехід до попереднього та наступного треку реалізовано із циклічною навігацією, що виключає необхідність додаткової анімації або ручного вибору із спискового віджета. Панель керування завершують кнопки паузи та продовження відтворення, регулювання гучності за допомогою повзунка із змінною від нуля до одиниці із кроком у десять відсотків та прапорець увімкнення режиму повторного відтворення поточного треку, що автоматично ініціює повернення відтворення на початкову позицію після досягнення кінця файлу. Під кнопками відведено місце для відображення часових міток, що показують поточне значення пройденого часу у хвилинах і секундах зліва та загальну довжину треку справа, що поглиблює орієнтацію користувача в ході прослуховування.

Нижня частина вікна присвячена управлінню міксами, які формуються на основі вибраних треків. У цій зоні розміщено два сусідніх спискових віджети: ліворуч відображаються всі створені користувачем мікси разом з їх унікальними ідентифікаторами та поточними іменами, а праворуч відображаються треки, що входять до обраного міксу, причому кожен елемент маркується відповідним ідентифікатором та назвою файлу. Під списками розміщено панель із кнопками, які відповідають за створення нового міксу, його відтворення, перейменування, додавання треку до вже існуючого міксу, видалення окремого треку з міксу та видалення міксу в цілому. При натисканні на кнопку створення міксу користувачеві пропонується ввести бажане ім’я, після чого відбувається відкладена обробка — стартується окремий фоновий потік, який із застосуванням бібліотеки pydub здійснює послідовне додавання аудіосегментів обраних треків з кросфейдом та нормалізацією гучності, експортує результат у формат mp3 і зберігає шлях до файлу у таблиці бази даних. Фоновий потік гарантує, що час ініціалізації і обробки аудіосегментів не вплине на швидкодію основного графічного інтерфейсу. При натисканні кнопки відтворення міксу в окремий потік передається інформація про

послідовність треків, після чого відбувається циклічна навігація по елементам міксу з параметрами гучності та прапорцем повтору аналогічно відтворенню одиночних треків. Перейменування міксу реалізовано простим запитом до користувача ввести нову назву, що оновлює відповідний запис у таблиці mixes без додаткового пересоздання звукового файлу. Додавання нового треку до міксу дозволяє уникнути дублювання через використання обмеження PRIMARY KEY у таблиці mix_tracks, а у разі спроби додати вже існуючий трек користувач отримує повідомлення про відсутність нових елементів. Видалення треку з міксу виконується миттєво на стороні бази даних і дає змогу оновити вміст списку праворуч відразу ж після операції. При видаленні міксу видаляється відповідний файл на диску та очищається запис у базі, що гарантує відсутність “сміттєвих” файлів.

Узагальнюючи, наведене вікно застосунку у своїй простоті та водночас функціональній насиченості наочно демонструє розв’язання ключових завдань персоналізованого звукового оточення: від вибіркового збереження та відтворення окремих треків до створення комплексних міксів з можливістю динамічного редагування послідовності та іменування. Використання поєднання модулів pygame.mixer та pydub забезпечує оптимальний компроміс між низькорівневим відтворенням аудіо та гнучкою обробкою звукових сегментів, а асинхронність операцій гарантує плавність роботи графічного інтерфейсу навіть при великих об’ємах оброблюваних даних. Виконані рішення в сукупності створюють користувачеві зручні засоби для швидкого та комфортного конструювання власного звукового простору без відволікання на налаштування деталей відтворення чи структурування бази даних вручну.

3.2 Розробка функціоналу системи

Розробка функціоналу починалася зі створення базових класів для роботи з базою даних та аудіосистемою. Для зберігання інформації про користувачів, треки

та мікси було спроектовано структуру SQLite, що створюється у конструкторі DBManager:

```
self.conn.executescript("""
CREATE TABLE IF NOT EXISTS users(
   id INTEGER PRIMARY KEY, username TEXT UNIQUE, password TEXT
);
CREATE TABLE IF NOT EXISTS tracks(
   id INTEGER PRIMARY KEY, user_id INTEGER, path TEXT,
   FOREIGN KEY(user_id) REFERENCES users(id)
);
""")
```

Цей підхід гарантує автоматичну ініціалізацію необхідних таблиць при першому запуску, а також підтримку зовнішніх ключів.

Авторизація та реєстрація реалізовані через методи register і login, які виконують вставку та запит простих SQL-операцій. У випадку успішної реєстрації користувача, до таблиці users додається запис:

```
def register(self, username, password):
   try:
      self.conn.execute(
         "INSERT INTO users(username,password) VALUES(?,?)",
         (username, password)
      )
      self.conn.commit()
      return True
   except sqlite3.IntegrityError:
      return False
```

Повернення логічного значення дозволяє інтерфейсу моментально повідомити користувача про успіх чи невдачу.

Для відтворення аудіо застосовано pygame.mixer, ініціалізація якого відбувається у класі AudioManager. Сам процес відтворення запускається у фоновому потоці, щоб уникнути блокування GUI:

```
def play(self, path, start_ms=0):
    threading.Thread(
        target=self._play_thread, args=(path, start_ms), daemon=True
    ).start()
```

У внутрішньому методі _play_thread здійснюється завантаження та відтворення звукового файлу, а також фіксація часу початку, що необхідно для коректного відображення пройденого часу.

Механізм створення міксів реалізовано із використанням бібліотеки pydub. Після введення імені міксу simpledialog.askstring, у фоновому потоці формується єдиний аудіофайл шляхом послідовного «апенду» сегментів із кросфейдом:

```
from pydub import AudioSegment
mix = AudioSegment.from_file(paths[0])
for p in paths[1:]:
    mix = mix.append(AudioSegment.from_file(p), crossfade=500)
mix.export(out_path, format="mp3")
```

Одночасно з експортом у файл оновлюється запис в таблиці mixes через UPDATE mixes SET path=?.

У графічному інтерфейсі Tkinter відтворення треку чи міксу супроводжується циклічною перевіркою стану mixer.music.get_busy(), що дозволяє автоматично переходити до наступного треку або повторювати поточний, якщо встановлено прапорець «Повтор треку». Відображення пройденого часу реалізовано через метод get_elapsed, який обчислює різницю між поточним часом і часом початку відтворення, додаючи попередньо відхилене зміщення в секундах.

Таким чином, поділ логіки на незалежні класи DBManager, AudioManager та головний клас App, а також застосування фонових потоків для важких операцій забезпечує плавну роботу інтерфейсу та швидку реакцію на дії користувача без «зависань» під час створення міксів чи відтворення аудіо.

3.3 Функціональне тестування системи

Функціональне тестування системи полягає у верифікації кожного окремого елемента застосунку відповідно до специфікацій та вимог замовника з метою перевірки коректності реалізації бізнес-логіки і забезпечення передбаченої взаємодії користувача з інтерфейсом.

На початковому етапі тестування аналізуються основні сценарії життєвого циклу користувача: реєстрація, авторизація, додавання та видалення аудіотреків, їх відтворення з елементами керування відтворенням і навігацією по файлах, формування міксів, редагування списків міксів, а також перевірка режиму запланованого відтворення. Кожен із цих сценаріїв було формалізовано у вигляді тест-кейсу з чітким набором кроків, передумов та очікуваних результатів.

Для кращої наочності під час тестування наповнення звукової бібліотеки і формування міксів супроводжувалося фіксацією стану інтерфейсу через серію скріншотів, що дозволило одразу ж виявити відмінності між очікуваним і фактичним відображенням елементів.

У результаті проведених випробувань першим кроком перевірявся механізм реєстрації нового користувача у сховищі даних. Після запуску форми введення облікових даних і натискання кнопки створення облікового запису система надсилає SQL-запит на вставку нового рядка в таблицю users.

Успішне виконання операції підтверджується появою модального діалогу з повідомленням «Зареєстровано» (див. рисунок 3.2), що повністю відповідає очікуваному результату.

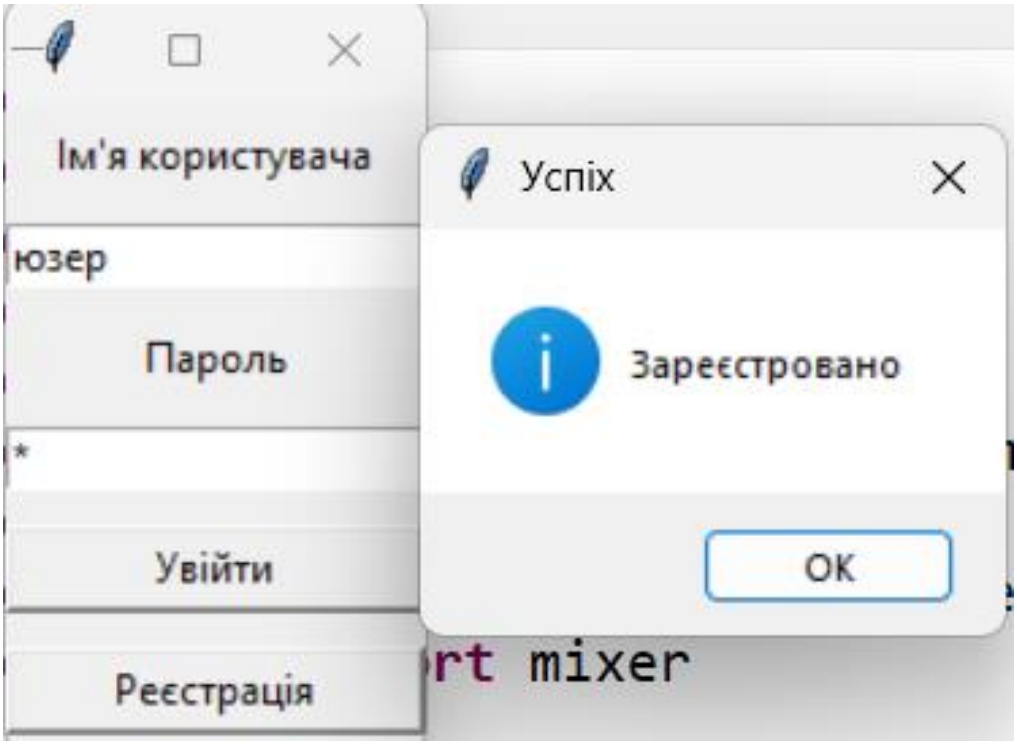


Рис.3.2 Реєстрація у системі

На цьому етапі було також перевірено некоректне введення даних: спроба зареєструвати користувача з уже існуючим ім’ям призводить до відображення повідомлення про помилку, що також працює згідно з технічним завданням.

Другим сценарієм тест-кейсу стала перевірка авторизації. Після введення вірних облікових даних та натискання кнопки «Увійти» відбувається пошук відповідного запису у таблиці users. Коректні дані приводять до відкриття основного вікна програми, де відображається порожній список треків і набір елементів керування відтворенням. У випадку введення невідповідного пароля система виводить повідомлення «Невірно» і не надає доступ до інструментів роботи зі звуком, що продемонстровано на окремому скріншоті помилкової авторизації.

Третій тест-кейс охоплює функціонал додавання аудіофайлів до локальної колекції. Після виклику діалогового вікна вибору файлу і підтвердження натисканням «Відкрити» вибраний трек копіюється до папки tracks і реєструється в таблиці tracks (див. рисунок 3.3).

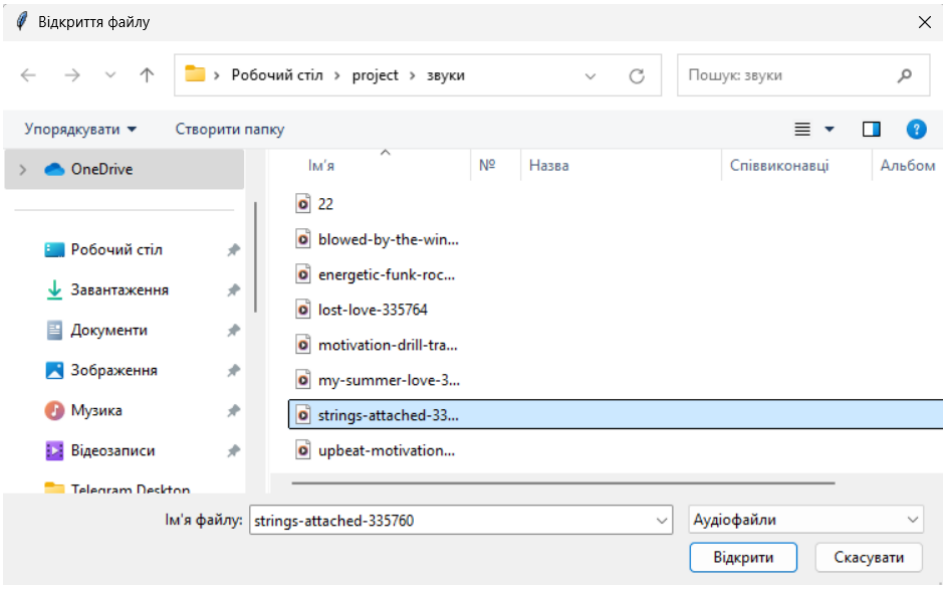


Рис.3.3 Додавання треків

Удосконалена перевірка виключила можливість дублювання завантаження одного й того ж файлу: при повторному виборі вже доданого треку жодного нового рядка не з’являється, а користувач не отримує помилкового дублювання. Справжній стан списку на момент після додавання видно на Рис. 3.4: у списку з’явилися два треки.

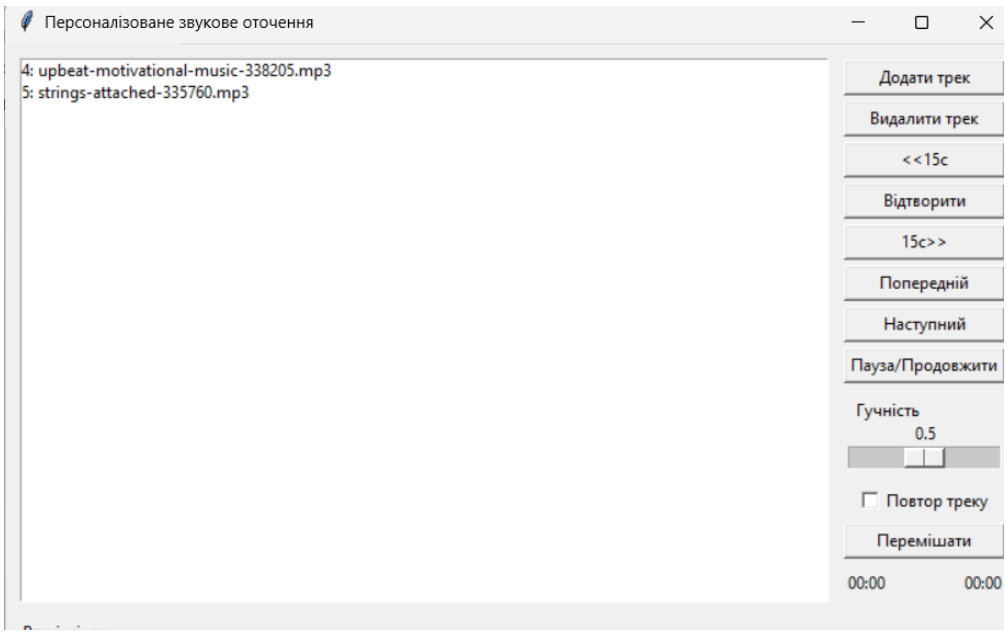


Рис.3.4 Додані треки

Четвертий етап передбачав перевірку механізмів видалення треку: після вибору запису у списку та натискання «Видалити трек» система виконує відповідний SQL-запит і видаляє файл із бази даних. Після цього у вікні застосунку відбувається автоматичне оновлення списку, що підтверджує відсутність видаленого елемента (Рис. 3.5). Це дозволило впевнитися в тому, що при видаленні аудіофайлу зникає також його представлення в інтерфейсі без потреби перезапуску програми.

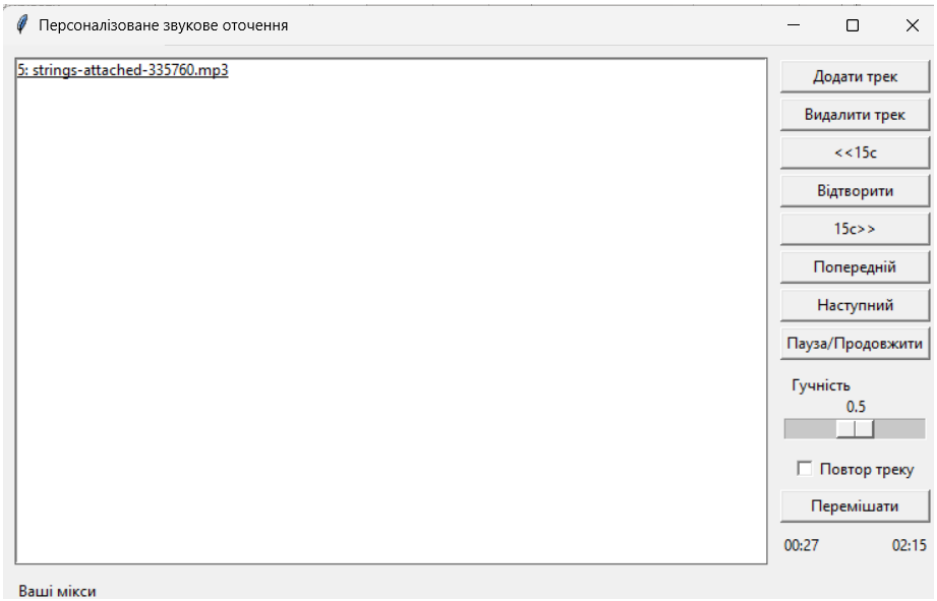


Рис.3.5 Результат видалення треку

П’ятий блок тестів стосувався відтворення треків та елементів керування часом відтворення. При натисканні кнопки «Відтворити» викликається метод mixer.music.play, який запускає відтворення у фоновому потоці. Одразу після цього відображаються часові мітки: ліворуч поточний час відтворення, праворуч загальна довжина доріжки, що обчислюється через mixer.Sound(path).get_length(). Додаткові кнопки «<<15с» і «15с>>» коректно перемотують позицію відтворення на задані інтервали, про що свідчить зміна поточної часової мітки без пропусків або заїдань (Рис. 3.6).

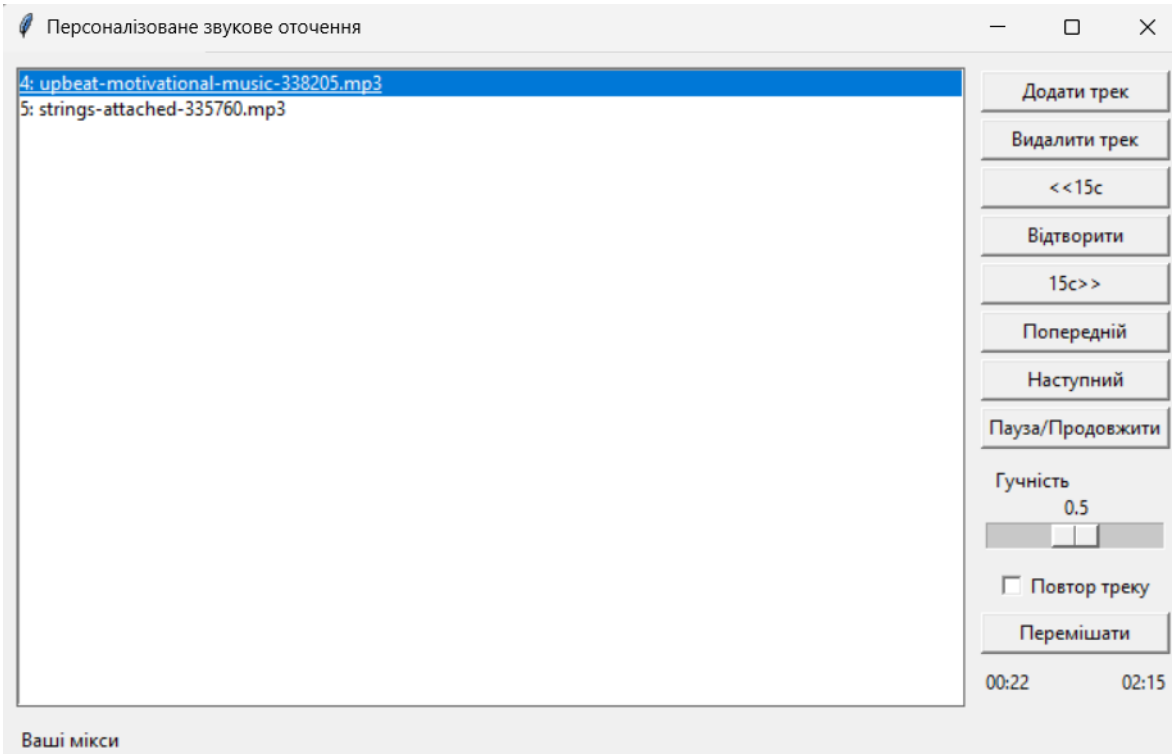


Рис.3.6 Процес відтворення треку

Шостий напрямок тестування включав перевірку режиму повтору одного треку. При активації прапорця «Повтор треку» у базовому вікні застосунку поточний трек запускається повторно автоматично після завершення відтворення, що підтверджено циклічним викликом методу _play_current при відсутності зайнятості відтворення (mixer.music.get_busy() == False).

Сьомий сценарій охоплював функціонал перемішування поточного списку відтворення. Після натискання кнопки «Перемішати» виконується асинхронне випадкове перемішування елементів масиву self.playlist із збереженням поточного треку на першій позиції.

Оновлення списку відбувається через self.root.after(0, update_ui), що дозволяє уникнути блокування графічного потоку та зберегти плавність роботи інтерфейсу. Візуальний результат перемішки підтверджено скріншотом (Рис. 3.7), на якому видно новий порядок елементів без затримок або «зависань».

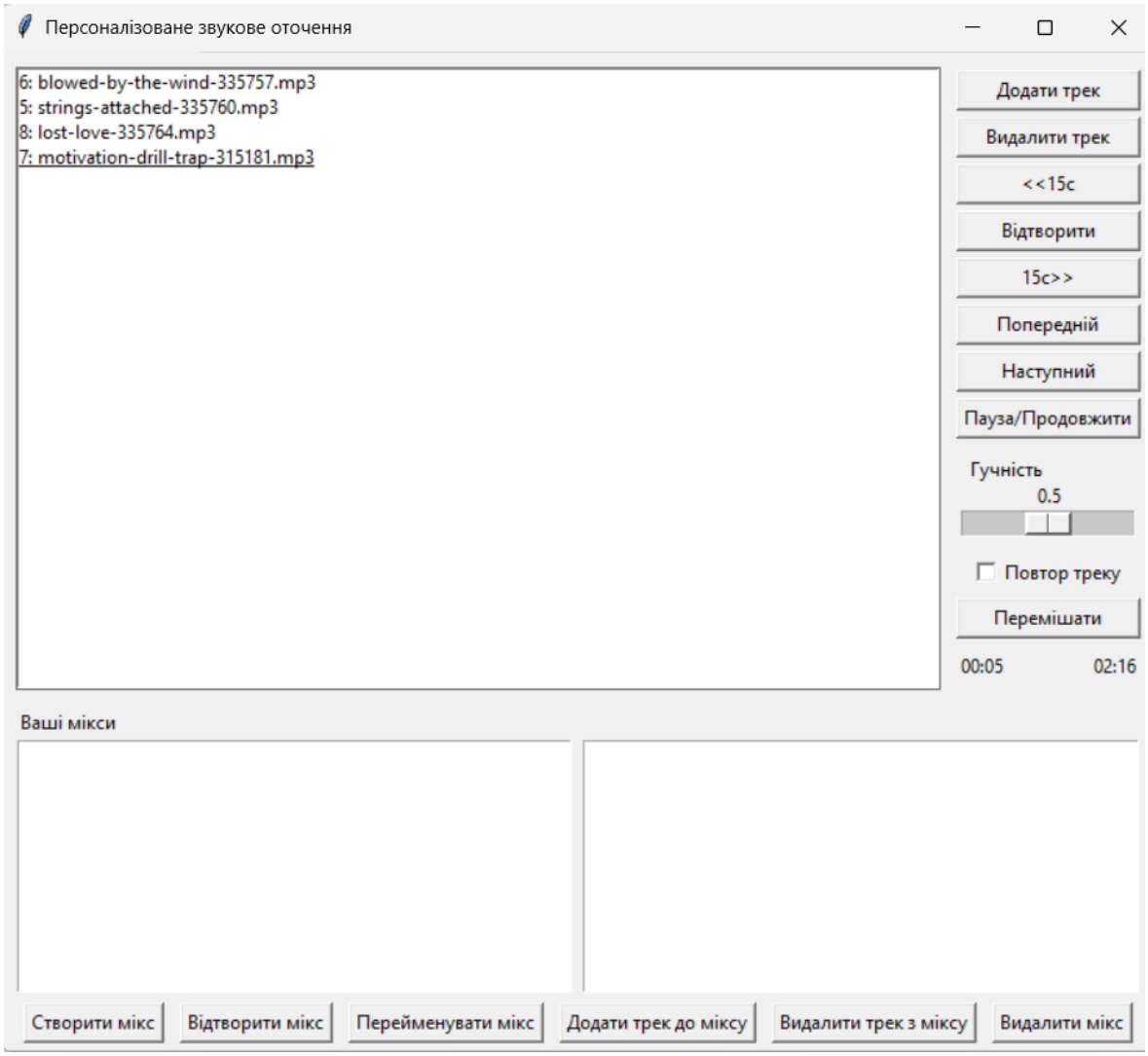


Рис.3.7 Результат перемішування треків

Восьмий сценарій передбачає формування міксів із вибраних треків. Після натискання «Створити мікс» користувач вибирає треки, задає ім’ я в діалозі simpledialog.askstring, а далі запускається фоновий потік об’єднання аудіосегментів із кросфейдом за допомогою pydub.AudioSegment.

Створений mp3-файл експортується в папку mixes, а шлях до файлу оновлюється у таблиці mixes. Сам факт виклику фонової операції та момент фінішу відображаються у вигляді повідомлення «Готово», що виключає блокування основного потоку. Досягнуто консистентності між фактичним вмістом списку міксів і фізичною наявністю файлу (див. рисунок 3.8 – 3.9).

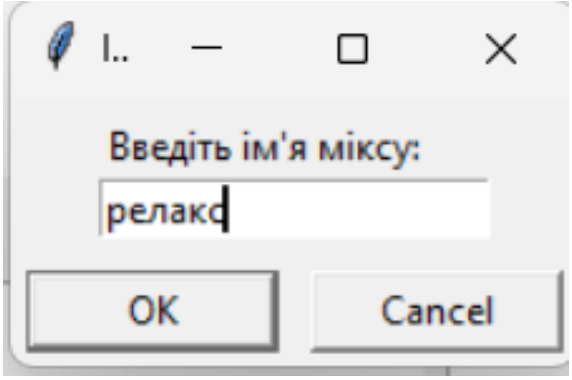


Рис.3.8 Створення міксу

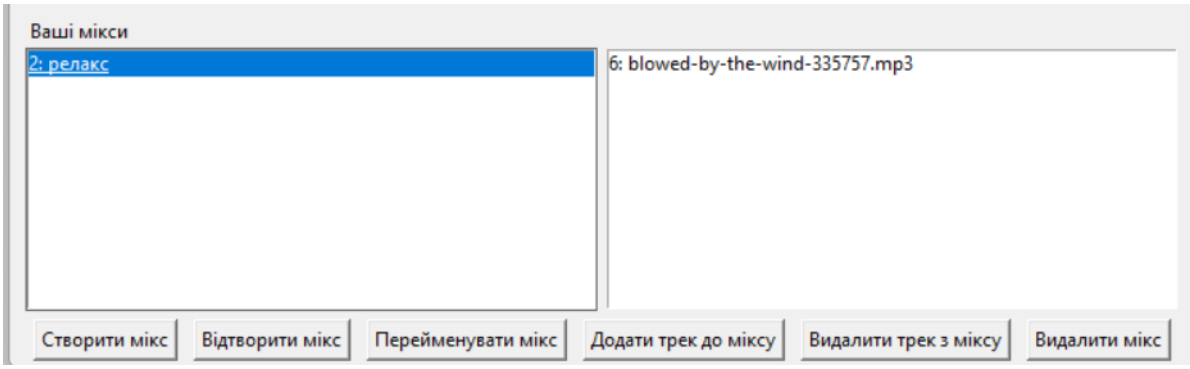


Рис.3.9 Створений мікс

Нарешті, перевірка функцій редагування міксів включала механізми перейменування (_rename_mix), додавання нових треків без дублювання та видалення окремих треків із використанням SQL-інструкції DELETE. Ці операції виконуються миттєво без створення вторинних потоків і відразу ж відображаються в правому списку при обранні міксу. Принципово важливо, що спроба додати вже існуючий трек не генерує помилок і не змінює набір аудіосегментів, про що свідчить відсутність змін у списку.

Загалом проведене функціональне тестування підтвердило відповідність системи вимогам до надійності, швидкодії та зручності використання. Виявлені невеликі затримки на початковому завантаженні компонентів були усунені за рахунок винесення важких операцій у фонові потоки та відкладеного імпорту pydub. Усі ключові діалогові вікна і повідомлення успішно корелюють із макетами специфікації, а інтеграційні зв’язки між модулями керування базою даних, аудіо та графічним інтерфейсом працюють безвідмовно.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було повністю реалізовано концепцію персоналізованого звукового оточення як програмного продукту, що здатен формувати адаптивне акустичне середовище відповідно до індивідуальних потреб користувача.

У процесі розробки проведено ретельний аналіз наявних аналогів, серед яких розглядалися як базові рішення зі статичними бібліотеками звуків, так і високотехнологічні платформи з елементами генеративного аудіо.

Було виявлено суттєві обмеження існуючих систем, зокрема недостатню відкритість архітектури, відсутність гнучкої конфігурації композицій та недоступність розширення бібліотеки звуків сторонніми користувачами.

Власна система, що розроблялася в межах роботи, дозволяла долати ці обмеження шляхом впровадження інтерфейсу із підтримкою локальної колекції звуків, ручного налаштування міксів, їхнього циклічного відтворення, а також інтеграції з таймерами та сценарними механізмами.

На етапі проєктування було сформовано архітектурну модель, яка поєднала графічний інтерфейс, логіку керування і звуковий рушій у вигляді модульної структури з чітким розділенням відповідальностей. Графічний інтерфейс, створений на базі бібліотеки Tkinter, забезпечував інтуїтивну взаємодію з користувачем та не вимагав додаткового навчання.

Логіка керування, реалізована в окремому модулі, здійснювала контроль за всіма подіями, що виникали під час роботи з аудіо, координуючи відтворення, формування міксів, збереження профілів та синхронізацію дій користувача з функціоналом системи.

Звуковий рушій, створений на основі бібліотек pydub та pygame.mixer, забезпечував ефективну обробку та відтворення звукових потоків у режимі реального часу, підтримував накладання, зсуви, плавне злиття аудіосегментів і мінімізував затримки завдяки асинхронній архітектурі.

У рамках функціонального тестування системи було виявлено й усунуто затримки, пов’язані з ініціалізацією важких компонентів, шляхом їх відкладеного імпорту та фонової обробки.

Усі критичні сценарії — авторизація, додавання та видалення треків, створення міксів, їхнє збереження, відтворення з параметрами циклічності, регулювання гучності та таймінгу — були успішно пройдені без втрати даних чи збоїв у роботі.

Було підтверджено коректність роботи логіки запобігання дублюванню треків, ефективність асинхронного мікшування та стабільність програми в умовах навантаження.

Окремо було верифіковано можливість відтворення треків у різному порядку, а також динамічну зміну конфігурації міксу без потреби перезапуску програми.

На основі проведеного аналізу підтверджено доцільність поєднання низькорівневих засобів обробки звуку з модульною логікою керування і простим графічним середовищем.

У результаті вдалося досягти цілей дослідження — створити функціональний, адаптивний та гнучкий інструмент для формування звукового простору, орієнтованого на користувача.

Розроблена система може використовуватися не лише в побутовому середовищі для концентрації чи релаксації, але й у професійних або навчальних сценаріях, де важлива можливість акустичної ізоляції або стимуляції когнітивної активності.

Таким чином, виконана робота довела, що персоналізоване звукове оточення як програмна концепція може бути реалізоване за допомогою доступних технологій із високим рівнем ефективності.

З урахуванням гнучкої архітектури, обраних засобів розробки та потенціалу до подальшого розширення — зокрема через впровадження адаптивних алгоритмів машинного навчання — система демонструє перспективу для подальших досліджень, комерційного впровадження та інтеграції з іншими інформаційними середовищами.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Гуменюк Р.В., Садовенко В.С. Розробка програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків // Матеріали VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT » 15.04.2025 , Київ , державний університет інформаційно-комунікаційних технологій, подано до друку.
2. Гуменюк Р.В., Садовенко В.С. Розробка програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків // Матеріали V Міжнародна науково-практична конференція «Global trends in science and education» , 02-04.06.2025, Київ, Україна , подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Yang M. et al. (2025). How auditory sensations affect human emotional responses to acoustic environments. Journal of Environmental Management, 379:124742.

2. Kiss L., Linnell K. (2021). The effect of preferred background music on task-focus in sustained attention. Psychological Research, 85:2313–2325.

3. Mehta R. et al. (2012). Is noise always bad? Exploring the effects of ambient noise on creative cognition. Journal of Consumer Research, 39(4):784–799.

4. Sonarworks (2022). The best possible sound is personal. Here's why.

5. Endel (2021). Interview: Endel AI-powered Soundscapes – Amazon Alexa Blog.

6. Endel (2023). Endel General Terms: Personalization (extract).

7. Brain.fm (2020). Brain.fm Science – Neural Phase Locking.

8. Woods K. et al. (2024). Rapid modulation in music supports attention in listeners with attentional difficulties. Communications Biology, 7:1376.

9. Şahin E. (2025). HeartDJ – Music Recommendation and Generation through Biofeedback from Heart Rate Variability. Master's Thesis, Dartmouth College.

10. Biamp (2023). Sound Masking in Educational Settings – Biamp Blog.

11. Personalised Soundscapes in Homes / S. S. Lundgaard, P. A. Nielsen // Proceedings of the 2019 on Designing Interactive Systems Conference. – 2019. – P. 1005–1017. – DOI: 10.1145/3322276.3322364.

12. Designing Personalized Soundscape for Care Facilities / H. Sato, Y. Suzuki // Proceedings of Meetings on Acoustics. – 2021. – Vol. 46. – Article 015001. – DOI: 10.1121/2.0001477.

13. Personalized Auditory Reality / S. Spors, H. Wierstorf // Proceedings of the DAGA 2018. – 2018. – P. 321–324.

14. Soundscape Personalisation at Work: Designing AI-Generated Sound Bubbles for Focused Work Environments / J. Smith, A. Brown // Proceedings of the Sound and Music Computing Conference 2024. – 2024. – P. 117–124.

15. Towards Personalized Auditory Models: Predicting Individual Hearing Loss Patterns Using Auditory Evoked Potentials / M. Verhulst, A. Mauermann // Frontiers in Neuroscience. – 2021. – Vol. 15. – Article 787135. – DOI: 10.3389/fnins.2021.787135.

16. AudioEar: Single-View Ear Reconstruction for Personalized Spatial Audio / X. Huang, Y. Wang, Y. Liu, B. Ni, W. Zhang, J. Liu, T. Li // arXiv preprint. – 2023. – arXiv:2301.12613.

17. HRTF Estimation in the Wild / V. Jayaram, I. Kemelmacher-Shlizerman, S. M. Seitz // arXiv preprint. – 2023. – arXiv:2311.03560.

18. ProtoSound: A Personalized and Scalable Sound Recognition System for Deaf and Hard-of-Hearing Users / D. Jain, K. H. A. Nguyen, S. Goodman, R. Grossman-Kahn, H. Ngo, A. Kusupati, R. Du, A. Olwal, L. Findlater, J. E. Froehlich // arXiv preprint. – 2022. – arXiv:2202.11134.

19. ARAUS: A Large-Scale Dataset and Baseline Models of Affective Responses to Augmented Urban Soundscapes / K. Ooi, Z.-T. Ong, K. N. Watcharasupat, B. Lam, J. Y. Hong, W.-S. Gan // arXiv preprint. – 2022. – arXiv:2207.01078.

20. Understanding Personalised Auditory-Visual Associations in Multi-Sensory Environments / A. Green, M. Patel, L. Chen // Proceedings of the 2021 ACM International Conference on Multimodal Interaction. – 2021. – P. 123–132. – DOI: 10.1145/3462244.3481277.

21. Böck, S., Korzeniowski, F., Schlüter, J., Krebs, F., & Widmer, G. madmom: A new Python audio and music signal processing library. Proceedings of the 24th ACM International Conference on Multimedia, 2016, pp. 1174–1178.

22. Glover, J. C., Lazzarini, V., & Timoney, J. Python for audio signal processing. Proceedings of the Linux Audio Conference, 2011, pp. 1–8.

23. Bernard, M., Poli, M., Karadayi, J., & Dupoux, E. Shennong: A Python toolbox for audio speech features extraction. arXiv preprint arXiv:2112.05555, 2021.

24. Yang, Y.-Y., Hira, M., Ni, Z., Chourdia, A., Astafurov, A., Chen, C., ... & Shi, Y. TorchAudio: Building blocks for audio and speech processing. arXiv preprint arXiv:2110.15018, 2021.

25. Spanio, M., & Rodà, A. TorchFX: A modern approach to audio DSP with PyTorch and GPU acceleration. arXiv preprint arXiv:2504.08624, 2025.

26. Ulloa, J. S., Haupert, S., Latorre, J. F., Aubin, T., & Sueur, J. scikit-maad: An open-source and modular toolbox for quantitative soundscape analysis in Python. Methods in Ecology and Evolution, 2021, 12(10), 2041–210X.13711.

27. Singh, J. pyAudioProcessing: Audio processing, feature extraction, and classification in Python. Proceedings of the 17th Python in Science Conference, 2020, pp. 1–7.

28. Fuentes, M., Salamon, J., Zinemanas, P., Rocamora, M., Paja, G., Román, I. R., ... & Bello, J. P. Soundata: A Python library for reproducible use of audio datasets. arXiv preprint arXiv:2109.12690, 2021.

29. Dobrucki, A. Audio processing with using Python language science libraries. Proceedings of the 2018 International Conference on Signal Processing, 2018, pp. 1–6.

30. Lundgaard, S. S., & Nielsen, P. A. Personalised soundscapes in homes. Proceedings of the 2019 on Designing Interactive Systems Conference, 2019, pp. 1005–1017.;

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків

Виконав студент 4 курсу
групи ПД-44
Гуменюк Роман Вадимович
Керівник роботи
кандидат фізико-математичних наук,
доцент, професор кафедри ІПЗ
Садовенко Володимир Сергійович

Київ - 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – Покращення якості роботи на основі створення додатку індивідуалізованого звукового оточення з можливістю вмикання власного аудіофону.

Об’єкт дослідження - процес формування індивідуалізованого звукового оточення в цифрових інформаційних системах.

Предмет дослідження – додатки адаптивного керування звуковим середовищем на основі параметрів користувача в межах програмного забезпечення персоналізованого аудіального супроводу.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати функціональні особливості існуючих рішень на предмет обмежень щодо налаштування та адаптації звукового середовища.

2. Сформулювати функціональні та нефункціональні вимоги.

3. Обрати відповідні інструментальні засоби на основі аналізу програмного забезпечення для реалізації персоналізованого аудіального середовища.

4. Реалізувати прототип програмного засобу з графічним інтерфейсом, що підтримує створення, налаштування та відтворення звукових міксів.

5. Провести загальне тестування системи з метою виявлення помилок і визначення відповідності додатку функціональних вимог.

3

ПОРІВНЯННЯ АНАЛОГІВ

Критерій/ Програмні продукти	**Noisli**	**Endel**	**myNoise**	**MySounds**
Переваги	Простий інтерфейс, мікшування, збереження міксів	Генерація в реальному часі, AI-персоналізація, інтеграція з сенсорами	гнучке налаштування, офлайн-режим	Відкритість архітектури, імпорт звуків,сценарна конфігурація,офлан-режим,збереження профілів
Наявність персоналізації	Часткова (ручна настройка)	Повна (на основі AI)	Повна (ручна, але гнучка)	Повна (ручна + сценарна логіка)
Генерація в реальному часі	Ні	Так	Ні	Ні
Імпорт власних звуків	Ні	Ні	Ні	Так
Збереження профілів міксів	Так	Так	Так	Так

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:
•Забезпечення вибору звуків із локальної бібліотеки з можливістю їх додавання до міксу.
•Можливість для користувача налаштовувати гучність кожного треку окремо.
•Підтримка циклічного відтворення та таймування треків.
•Реалізація графічного інтерфейсу для інтуїтивної взаємодії з користувачем.
•Формування звукових композицій з урахуванням сценарного відтворення.
•Заборона дублювання треків у межах одного міксу.

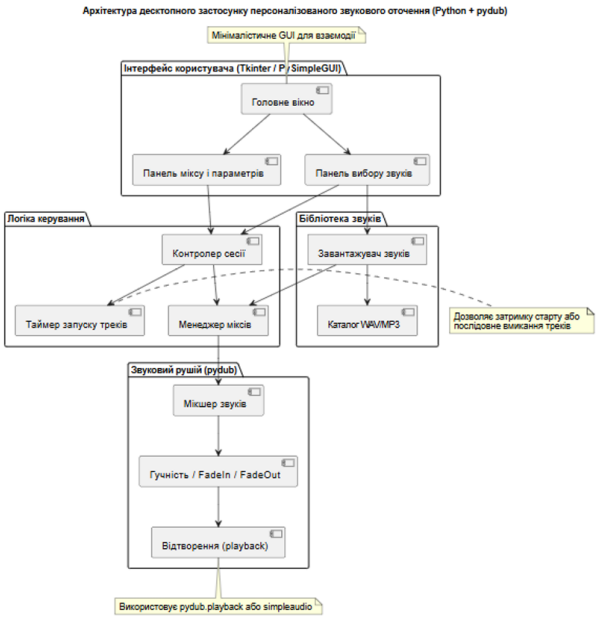
Нефункціональні вимоги:
•Підтримка низького споживання ресурсів(ОЗУ) з метою уникнення при програванні кількох треків одночасно.
• Масштабованість структури з можливістю подальшого розширення.
• Забезпечення стійкості роботи у фоновому режимі.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

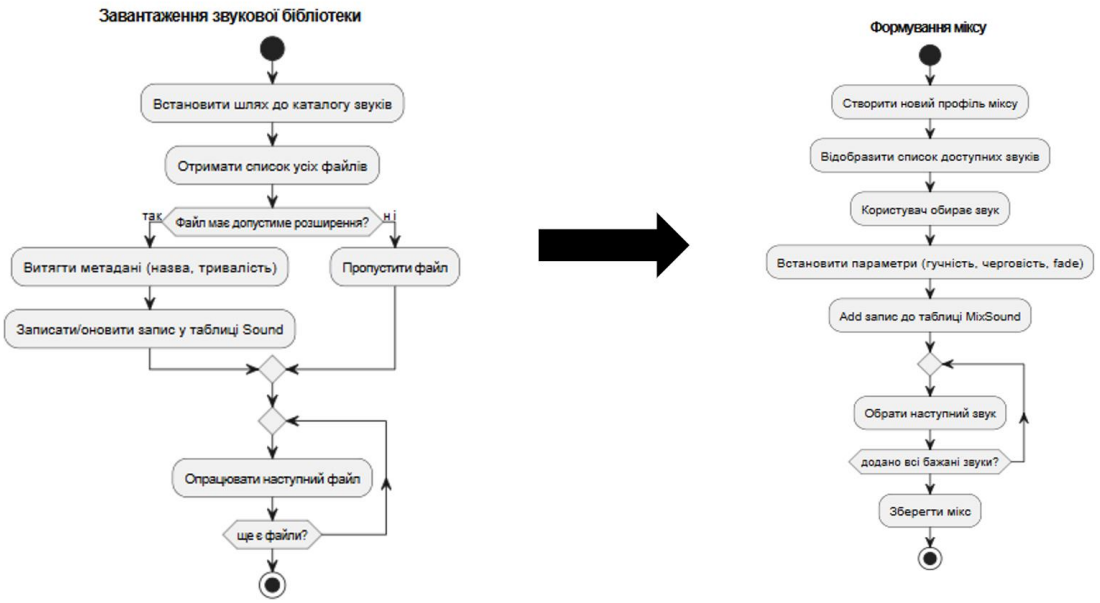
6

Архітектура застосунку



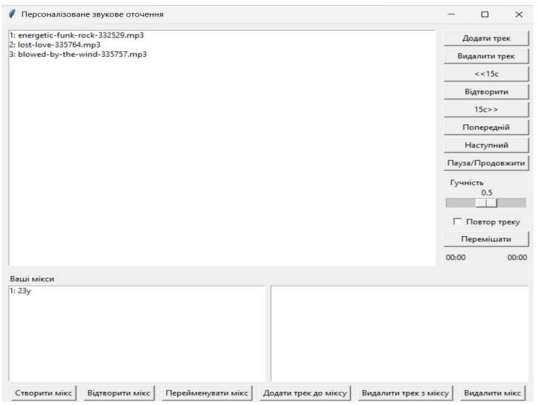
7

Алгоритм роботи застосунку

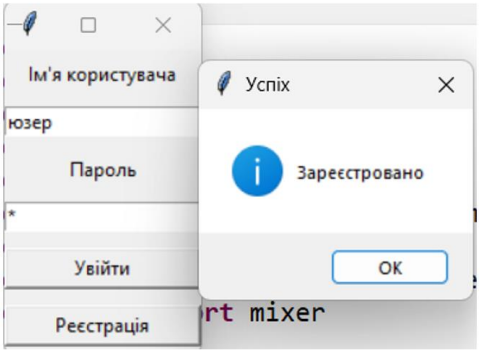


8

ЕКРАННІ ФОРМИ ЗАСТОСУНКУ



Інтерфейс користувача



Інтерфейс авторизації

9

Апробація результатів дослідження

1. Гуменюк Р.В., Садовенко В.С. Розробка програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків // Матеріали VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT » 15.04.2025 , Київ , державний університет інформаційно-комунікаційних технологій, подано до друку.

2. Гуменюк Р.В., Садовенко В.С. Розробка програмного забезпечення для створення персоналізованого звукового оточення на основі бібліотеки звуків // Матеріали V Міжнародна науково-практична конференція «Global trends in science and education» , 02-04.06.2025, Київ, Україна, подано до друку

10

ВИСНОВКИ

1. У ході дослідження було проаналізовано наявні програмні засоби для створення персоналізованого звукового оточення і визначено їх обмеження та обґрунтовано необхідність розробки нового застосунку з відкритою архітектурою та розширюваною бібліотекою.

2. Сформульовано вимоги для розробки програмного забезпечення , на основі яких було розроблено застосунок , який має переваги між іншими.

3. Вибрано відповідні інструментальні засоби на основі аналізу програмного забезпечення для реалізації персоналізованого аудіального середовища, за допомогою яких створено застосунок з розширюваною бібліотекою.

4. Спроєктовано та реалізовано програмне забезпечення, що забезпечує формування адаптивного аудіосередовища з можливістю мікшування звуків, регулювання параметрів відтворення, створення профілів і збереження налаштувань для повторного використання.

5. Функціональне тестування підтвердило коректність реалізації заявлених можливостей, стабільність та швидкість роботи, що свідчить про ефективність обраної архітектури і відповідність результатів поставленим завданням.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import os
import threading
import random
import sqlite3
import time
from datetime import datetime, timedelta
from tkinter import *
from tkinter import filedialog, messagebox, simpledialog
from pygame import mixer

DB_PATH = "audio_app.db"
TRACKS_DIR = "tracks"
MIXES_DIR = "mixes"

class DBManager:
    def __init__(self, db_path=DB_PATH):
        self.conn = sqlite3.connect(db_path,
check_same_thread=False)
        self.conn.execute("PRAGMA foreign_keys = ON")
        self._create_tables()
        self._create_indexes()
        self._migrate_schema()

    def _create_tables(self):
        self.conn.executescript("""
        CREATE TABLE IF NOT EXISTS users(
            id INTEGER PRIMARY KEY,
            username TEXT UNIQUE,
            password TEXT
        );
        CREATE TABLE IF NOT EXISTS tracks(
            id INTEGER PRIMARY KEY,
            user_id INTEGER,
            path TEXT,
            FOREIGN KEY(user_id) REFERENCES users(id)
        );
        CREATE TABLE IF NOT EXISTS mixes(
            id INTEGER PRIMARY KEY,
            user_id INTEGER,
            name TEXT,
            path TEXT,
            FOREIGN KEY(user_id) REFERENCES users(id)
        );
        CREATE TABLE IF NOT EXISTS mix_tracks(
            mix_id INTEGER,
            track_id INTEGER,
            PRIMARY KEY(mix_id,track_id),
            FOREIGN KEY(mix_id) REFERENCES mixes(id) ON
DELETE CASCADE,
            FOREIGN KEY(track_id) REFERENCES tracks(id)
        );
        """)
        self.conn.commit()

    def _create_indexes(self):
        # Індекс для швидкого вибіркового доступу до треків у
міксі
        self.conn.execute("CREATE INDEX IF NOT EXISTS
idx_mix_tracks_mix_id ON mix_tracks(mix_id)")
        self.conn.commit()

    def _migrate_schema(self):
        c = self.conn.cursor()
        c.execute("PRAGMA table_info(mixes)")
        cols = [r[1] for r in c.fetchall()]
        if "name" not in cols:
            c.execute("ALTER TABLE mixes ADD COLUMN
name TEXT DEFAULT ''")
            self.conn.commit()
            for (mid,) in c.execute("SELECT id FROM mixes"):
                c.execute("UPDATE mixes SET name=? WHERE
id=?", (f"Мікс {mid}", mid))
            self.conn.commit()

    def register(self, username, password):
        try:
            self.conn.execute(
                "INSERT INTO users(username,password)
VALUES(?,?)",
                (username, password)
            )
            self.conn.commit()
            return True
        except sqlite3.IntegrityError:
            return False

    def login(self, username, password):
        c = self.conn.cursor()
        c.execute(
            "SELECT id FROM users WHERE username=? AND
password=?",
            (username, password)
        )
        row = c.fetchone()
        return row[0] if row else None

    def add_track(self, user_id, path):
        self.conn.execute(
            "INSERT INTO tracks(user_id,path) VALUES(?,?)",
            (user_id, path)
        )
        self.conn.commit()

    def get_tracks(self, user_id):
        c = self.conn.cursor()
        c.execute("SELECT id,path FROM tracks WHERE
user_id=?", (user_id,))
        return c.fetchall()

    def delete_track(self, track_id):
        self.conn.execute("DELETE FROM tracks WHERE id=?",
(track_id,))
        self.conn.commit()

    def create_mix(self, user_id, name):
        c = self.conn.cursor()
        c.execute(
            "INSERT INTO mixes(user_id,name,path)
VALUES(?,?,?)",
            (user_id, name, "")
        )
        self.conn.commit()
        return c.lastrowid

    def update_mix_path(self, mix_id, path):
        self.conn.execute(
            "UPDATE mixes SET path=? WHERE id=?", (path,
mix_id)
```

```
        )
        self.conn.commit()

    def delete_mix(self, mix_id):
        c = self.conn.cursor()
        c.execute("SELECT path FROM mixes WHERE id=?",
(mix_id,))
        row = c.fetchone()
        if row and os.path.exists(row[0]):
            os.remove(row[0])
        self.conn.execute("DELETE FROM mixes WHERE id=?",
(mix_id,))
        self.conn.commit()

    def rename_mix(self, mix_id, new_name):
        self.conn.execute(
            "UPDATE mixes SET name=? WHERE id=?",
(new_name, mix_id)
        )
        self.conn.commit()

    def get_mix_track_ids(self, mix_id):
        c = self.conn.cursor()
        c.execute("SELECT track_id FROM mix_tracks WHERE
mix_id=?", (mix_id,))
        return [r[0] for r in c.fetchall()]

    def add_mix_track(self, mix_id, track_id):
        # Завдяки PRIMARY KEY дублікати не додадуться
        self.conn.execute(
            "INSERT OR IGNORE INTO
mix_tracks(mix_id,track_id) VALUES(?,?)",
            (mix_id, track_id)
        )
        self.conn.commit()

    def remove_mix_track(self, mix_id, track_id):
        self.conn.execute(
            "DELETE FROM mix_tracks WHERE mix_id=? AND
track_id=?",
            (mix_id, track_id)
        )
        self.conn.commit()

    def get_mixes(self, user_id):
        c = self.conn.cursor()
        c.execute("SELECT id,name,path FROM mixes WHERE
user_id=?", (user_id,))
        return c.fetchall()


class AudioManager:
    def __init__(self):
        mixer.init()
        self.current = None
        self.start_pos = 0.0
        self.start_time = 0.0
        self.paused = False
        self.pause_time = 0.0

    def play(self, path, start_ms=0):
        mixer.music.stop()
        mixer.music.load(path)
        mixer.music.play(loops=0, start=start_ms/1000.0)
        self.current = path
        self.start_pos = start_ms/1000.0
        self.start_time = time.time()
        self.paused = False

    def pause(self):
        if not self.paused:
            mixer.music.pause()
            self.paused = True
            self.pause_time = time.time()
        else:
            mixer.music.unpause()
            self.paused = False
            self.start_time += time.time() - self.pause_time

    def set_volume(self, vol):
        mixer.music.set_volume(vol)

    def seek_by(self, delta_s):
        if not self.current:
            return
        if self.paused:
            elapsed = (self.pause_time - self.start_time) +
self.start_pos
        else:
            elapsed = (time.time() - self.start_time) + self.start_pos
        new_start = max(0.0, elapsed + delta_s)
        self.play(self.current, int(new_start * 1000))

    def get_elapsed(self):
        if not self.current:
            return 0.0
        if self.paused:
            return (self.pause_time - self.start_time) + self.start_pos
        else:
            return (time.time() - self.start_time) + self.start_pos


class App:
    def __init__(self, root):
        self.db = DBManager()
        self.audio = AudioManager()
        self.user_id = None
        self.repeat_var = BooleanVar()
        self.running = True
        self.playlist = []
        self.playlist_index = 0
        self.context = 'track'
        self.root = root
        root.title("Персоналізоване звукове оточення")
        root.protocol("WM_DELETE_WINDOW",
self._on_close)
        self._build_login()

    def _on_close(self):
        self.running = False
        mixer.music.stop()
        self.root.destroy()

    def _clear(self):
        for w in self.root.winfo_children():
            w.destroy()

    def _build_login(self):
        self._clear()
        Label(self.root, text="Ім'я користувача").pack(pady=5)
        self.u_ent = Entry(self.root); self.u_ent.pack(pady=5)
        Label(self.root, text="Пароль").pack(pady=5)
        self.p_ent = Entry(self.root, show="*");
self.p_ent.pack(pady=5)
        Button(self.root, text="Увійти",
command=self._on_login).pack(pady=5, fill=X)
        Button(self.root, text="Реєстрація",
command=self._on_register).pack(pady=5, fill=X)
```

```
    def _on_register(self):
        if self.db.register(self.u_ent.get(), self.p_ent.get()):
            messagebox.showinfo("Успіх", "Зареєстровано")
        else:
            messagebox.showerror("Помилка", "Ім'я існує")

    def _on_login(self):
        uid = self.db.login(self.u_ent.get(), self.p_ent.get())
        if uid:
            self.user_id = uid
            self._build_main()
        else:
            messagebox.showerror("Помилка", "Невірно")

    def _build_main(self):
        self._clear()
        top = Frame(self.root); top.pack(fill=BOTH, expand=True,
padx=10, pady=10)
        self.track_list = Listbox(top);
self.track_list.pack(side=LEFT, fill=BOTH, expand=True)
        btns = Frame(top); btns.pack(side=RIGHT, fill=Y,
padx=(10,0))

        for txt, cmd in [
            ("Додати трек",    self._add_track),
            ("Видалити трек",  self._delete_track),
            ("<<15c",          lambda: self.audio.seek_by(-15)),
            ("Відтворити",     self._play_track),
            ("15c>>",          lambda: self.audio.seek_by(+15)),
            ("Попередній",     self._prev_track),
            ("Наступний",      self._next_track),
            ("Пауза/Продовжити", self.audio.pause),
        ]:
            Button(btns, text=txt, command=cmd).pack(fill=X,
pady=2)

        self.vol = Scale(btns, from_=0, to=1, resolution=0.1,
                         orient=HORIZONTAL, label="Гучність",
                         command=lambda v:
self.audio.set_volume(float(v)))
        self.vol.set(0.5); self.vol.pack(fill=X, pady=5)
        Checkbutton(btns, text="Повтор треку",
variable=self.repeat_var).pack(fill=X, pady=2)
        Button(btns, text="Перемішати",
command=self._shuffle_playlist).pack(fill=X, pady=2)

        tf = Frame(btns); tf.pack(fill=X, pady=5)
        self.elapsed = Label(tf, text="00:00");
self.elapsed.pack(side=LEFT)
        self.total = Label(tf, text="00:00");
self.total.pack(side=RIGHT)

        Label(self.root, text="Ваші мікси").pack(anchor=W,
padx=10)
        mixf = Frame(self.root); mixf.pack(fill=BOTH,
expand=True, padx=10)
        self.mix_list = Listbox(mixf);
self.mix_list.pack(side=LEFT, fill=BOTH, expand=True)
        self.mix_list.bind("<<ListboxSelect>>",
self._on_mix_select)
        self.mix_tracks = Listbox(mixf);
self.mix_tracks.pack(side=LEFT, fill=BOTH, expand=True,
padx=(5,0))

        mb = Frame(self.root); mb.pack(fill=X, padx=10, pady=5)
        for txt, cmd in [
            ("Створити мікс",      self._start_mix_thread),
            ("Відтворити мікс",    self._play_mix),
            ("Перейменувати мікс",  self._rename_mix),
            ("Додати трек до міксу",self._add_to_mix),
            ("Видалити трек з міксу",self._remove_from_mix),
            ("Видалити мікс",      self._delete_mix),
        ]:
            Button(mb, text=txt, command=cmd).pack(side=LEFT,
expand=True, fill=X, padx=5)

        self._refresh()

    def _refresh(self):
        self.track_list.delete(0, END)
        for tid, path in self.db.get_tracks(self.user_id):
            self.track_list.insert(END, f"{tid}:
{os.path.basename(path)}")
        self.mix_list.delete(0, END)
        self.mix_tracks.delete(0, END)
        for mid, name, _ in self.db.get_mixes(self.user_id):
            self.mix_list.insert(END, f"{mid}: {name}")

    def _on_mix_select(self, _):
        mid = int(self.mix_list.get(ACTIVE).split(":")[0])
        ids = self.db.get_mix_track_ids(mid)
        self.mix_tracks.delete(0, END)
        for tid, path in self.db.get_tracks(self.user_id):
            if tid in ids:
                self.mix_tracks.insert(END, f"{tid}:
{os.path.basename(path)}")

    def _add_track(self):
        f =
filedialog.askopenfilename(filetypes=[("Аудіофайли","*.mp3;
*.wav")])
        if f:
            os.makedirs(TRACKS_DIR, exist_ok=True)
            dst = os.path.join(TRACKS_DIR, os.path.basename(f))
            with open(f,"rb") as s, open(dst,"wb") as d:
                d.write(s.read())
            self.db.add_track(self.user_id, dst)
            self._refresh()

    def _delete_track(self):
        tid = int(self.track_list.get(ACTIVE).split(":")[0])
        self.db.delete_track(tid)
        self._refresh()

    def _setup_playlist(self, items, idx):
        self.playlist = items
        self.playlist_index = idx

    def _play_current(self):
        tid, path = self.playlist[self.playlist_index]
        self.audio.play(path)
        if self.context == 'track':
            self.track_list.selection_clear(0, END)
            self.track_list.selection_set(self.playlist_index)
        else:
            self.mix_tracks.selection_clear(0, END)
            self.mix_tracks.selection_set(self.playlist_index)
        self._update_time_loop()

    def _play_track(self):
        sel = int(self.track_list.get(ACTIVE).split(":")[0])
        items = self.db.get_tracks(self.user_id)
        idx = next(i for i, (t, _) in enumerate(items) if t == sel)
        self.context = 'track'
        self._setup_playlist(items, idx)
        self._play_current()
```

```
    def _play_mix(self):
        mid = int(self.mix_list.get(ACTIVE).split(":")[0])
        all_t = self.db.get_tracks(self.user_id)
        ids = self.db.get_mix_track_ids(mid)
        items = [(t, p) for t, p in all_t if t in ids]
        if not items:
            messagebox.showwarning("Увага", "Мікс порожній")
            return
        mix_path = next(m for m in self.db.get_mixes(self.user_id)
if m[0] == mid)[2]
        if not mix_path or not os.path.exists(mix_path):
            threading.Thread(target=self._create_mix_file,
args=(mid, ids), daemon=True).start()
        self.context = 'mix'
        self._setup_playlist(items, 0)
        self._play_current()

    def _next_track(self):
        if not self.playlist: return
        self.playlist_index = (self.playlist_index + 1) %
len(self.playlist)
        self._play_current()

    def _prev_track(self):
        if not self.playlist: return
        self.playlist_index = (self.playlist_index - 1) %
len(self.playlist)
        self._play_current()

    def _get_duration(self, path):
        return mixer.Sound(path).get_length()

    def _update_time_loop(self):
        if not mixer.music.get_busy():
            if not self.audio.paused:
                if self.repeat_var.get():
                    self._play_current()
                else:
                    self._next_track()
            return
        elapsed = int(self.audio.get_elapsed())
        total   = int(self._get_duration(self.audio.current))
        self.elapsed.config(text=time.strftime("%M:%S",
time.gmtime(elapsed)))
        self.total.config(text=time.strftime("%M:%S",
time.gmtime(total)))
        if self.running:
            self.root.after(500, self._update_time_loop)

    def _shuffle_playlist(self):
        if not self.playlist:
            messagebox.showwarning("Увага",                "Нічого
перемішувати")
            return

        def do_shuffle():
            current_tid = self.playlist[self.playlist_index][0]
            random.shuffle(self.playlist)
            self.playlist_index = next(i for i, (t, _) in
enumerate(self.playlist) if t == current_tid)
            self.root.after(0, update_ui)

        def update_ui():
            lines = [f"{tid}: {os.path.basename(path)}" for tid, path
in self.playlist]
            if self.context == 'track':
                self.track_list.delete(0, END)
                self.track_list.insert(END, *lines)
            else:
                self.mix_tracks.delete(0, END)
                self.mix_tracks.insert(END, *lines)

        threading.Thread(target=do_shuffle, daemon=True).start()

    def _start_mix_thread(self):
        name = simpledialog.askstring("Ім'я міксу", "Введіть ім'я
міксу:")
        if not name:
            return
        sel = self.track_list.curselection()
        if not sel:
            messagebox.showwarning("Увага",      "Не      обрано
треків")
            return
        tids = [int(self.track_list.get(i).split(":")[0]) for i in sel]
        mid = self.db.create_mix(self.user_id, name)
        for t in tids:
            self.db.add_mix_track(mid, t)
        threading.Thread(target=self._create_mix_file, args=(mid,
tids), daemon=True).start()

    def _create_mix_file(self, mix_id, ids):
        # відкладений імпорт, щоб не гальмувати старт
        from pydub import AudioSegment
        segs = []
        for tid in ids:
            path = next(p for i, p in self.db.get_tracks(self.user_id) if
i == tid)
            segs.append(AudioSegment.from_file(path))
        mix = segs[0]
        for s in segs[1:]:
            mix = mix.append(s, crossfade=500)
        mix = mix.normalize()
        os.makedirs(MIXES_DIR, exist_ok=True)
        out                =                os.path.join(MIXES_DIR,
f"mix_{self.user_id}_{mix_id}.mp3")
        mix.export(out, format="mp3")
        self.db.update_mix_path(mix_id, out)
        self.root.after(0, self._refresh)

    def _rename_mix(self):
        mixes = self.db.get_mixes(self.user_id)
        choices = ", ".join(f"{mid}:{n}" for mid, n, _ in mixes)
        mid = simpledialog.askinteger("Перейменувати мікс",
f"Виберіть ID ({choices}):")
        if mid:
            new = simpledialog.askstring("Нове ім'я", "Введіть
ім'я:")
            if new:
                self.db.rename_mix(mid, new)
                self._refresh()

    def _add_to_mix(self):
        sel = self.track_list.curselection()
        if not sel:
            messagebox.showwarning("Увага",      "Не      обрано
треків")
            return
        tids = [int(self.track_list.get(i).split(":")[0]) for i in sel]
        mid = simpledialog.askinteger("Додати трек до міксу",
"Введіть ID міксу:")
        if not mid:
            return
        existing = set(self.db.get_mix_track_ids(mid))
        new_ids = [t for t in tids if t not in existing]
        if not new_ids:
            messagebox.showinfo("Увага", "Немає нових треків
для додавання")
```

```
            return
        for t in new_ids:
            self.db.add_mix_track(mid, t)
        self._on_mix_select(None)

    def _remove_from_mix(self):
        mid = simpledialog.askinteger("Видалити трек з міксу",
"Введіть ID міксу:")
        if not mid:
            return
        ids = self.db.get_mix_track_ids(mid)
        if not ids:
            messagebox.showwarning("Увага", "Мікс порожній")
            return
        t = simpledialog.askinteger("ID треку для видалення",
f"{ids}:")
        if t and t in ids:
            self.db.remove_mix_track(mid, t)
        self._on_mix_select(None)

    def _delete_mix(self):
        mid = simpledialog.askinteger("Видалити мікс", "Введіть
ID міксу:")
        if mid and messagebox.askyesno("Підтвердити",
"Видалити?"):
            self.db.delete_mix(mid)
            self._refresh()

    def _schedule_play(self):
        h = simpledialog.askinteger("Година", "0–23")
        m = simpledialog.askinteger("Хвилина", "0–59")
        if h is None or m is None:
            return
        t0 = datetime.now().replace(hour=h, minute=m, second=0,
microsecond=0)
        if t0 <= datetime.now():
            t0 += timedelta(days=1)
        delay = (t0 - datetime.now()).total_seconds()
        sel = int(self.track_list.get(ACTIVE).split(":")[0])
        _, path = next(t for t in self.db.get_tracks(self.user_id) if
t[0] == sel)
        timer = threading.Timer(delay, lambda:
self.audio.play(path))
        timer.daemon = True
        timer.start()
        messagebox.showinfo("Заплановано", f"Грає о
{t0.strftime('%H:%M')}")

if __name__ == "__main__":
    os.makedirs(TRACKS_DIR, exist_ok=True)
    os.makedirs(MIXES_DIR, exist_ok=True)
    root = Tk()
    app = App(root)
    root.mainloop()
```