Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ТА ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ СИНХРОНІЗОВАНОГО ПРОСЛУХОВУВАННЯ АУДІОПОТОКУ З ІНТЕРАКТИВНИМ ГОЛОСУВАННЯМ
DEVELOPMENT AND RESEARCH OF A WEB-BASED SYNCHRONIZED AUDIO STREAMING APPLICATION WITH INTERACTIVE VOTING FEATURES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-112
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ФЕДИШЕН С.В.
(ПРІЗВИЩЕ та ініціали)
Керівник КОЛПАКОВА Т.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОЦУР М.І.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
“____” _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ФЕДИШЕНА Станіслава Віталійовича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Розроблення та дослідження програмного забезпечення вебзастосунку для синхронізованого прослуховування аудіопотоку з інтерактивним голосуванням. Development and Research of a Web-Based Synchronized Audio Streaming Application with Interactive Voting Features
керівник проєкту (роботи) к.т.н., доцент, КОЛПАКОВА Тетяна Олексіївна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КОЛПАКОВА Т.О., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	**Примітка**
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Розробка архітектури програми.	2 тиждень	Розділ 2
4	Розробка програми.	3-4 тижні	Розділ 3
5	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
6	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
7	Нормоконтроль та рецензування.	7 тиждень	
8	Захист роботи.	8 тиждень	

Студент(ка) _____ Станіслав ФЕДИШЕН
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Тетяна КОЛПАКОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 85 с., 11 табл., 25 рис., 3 дод., 15 джерел.

ВЕБЗАСТОСУНОК, СИНХРОНІЗАЦІЯ АУДІОПОТОКУ, WEBSOCKET, РЕАЛЬНИЙ ЧАС, ГОЛОСУВАННЯ, LARAVEL, КОМПЕНСАЦІЯ ЗАТРИМКИ, LIVEWIRE, REDIS.

Об'єкт дослідження – процеси програмної інженерії, створення, експлуатація та супровід вебзастосунків реального часу.

Предмет дослідження – методи та технології синхронізації аудіопотоку, інтерактивного голосування, масштабування WebSocket-з'єднань та забезпечення якості програмного забезпечення.

Мета роботи – зменшення затримки синхронізації відтворення аудіоконтенту між учасниками спільної сесії прослуховування шляхом розроблення вебзастосунку з механізмом компенсації мережевої затримки на стороні клієнта та інтерактивним голосуванням за наступний трек.

Матеріали, методи та технічні засоби. Використано методи об'єктно-орієнтованого проєктування та патерни MVC, Observer і Command. Серверну частину реалізовано на фреймворку Laravel 12 із використанням PHP 8.3, Eloquent ORM та бази даних MySQL 8.0. WebSocket-з'єднання забезпечує Laravel Reverb, черги завдань – Laravel Horizon на базі Redis. Клієнтську частину реалізовано з використанням Livewire 3, Alpine.js та Tailwind CSS. Для відтворення аудіоконтенту використано SoundCloud Widget API. Середовище розгортання організоване через Docker Compose.

Результати. Розроблено вебзастосунок Aurasync, що забезпечує синхронізоване прослуховування аудіоконтенту для кількох учасників одночасно. Реалізовано алгоритм компенсації мережевої затримки на стороні клієнта, систему голосування за наступний трек з автоматичним визначенням

переможця через механізм черг, чат реального часу з модерацією та захистом від спаму, а також систему ролей з трьома рівнями доступу. Проведено функціональне тестування з 18 тест-кейсами, всі пройдено успішно.

Висновки. Мету роботи досягнуто: реалізований механізм компенсації мережевої затримки забезпечує прийнятну точність синхронізації відтворення між учасниками. За результатами експериментального дослідження середнє відхилення позиції відтворення при приєднанні до активної сесії становить ±0.28 с, затримка при команді play – 1.06 с, при перемотуванні – 0.80 с. Отримані результати підтверджують практичну застосовність підходу синхронізації команд відтворення через WebSocket для задач спільного прослуховування аудіоконтенту.

Галузь використання – розважальні та соціальні вебплатформи, сервіси спільного прослуховування музики, подкастів та аудіокниг.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 85 pages, 11 tables, 25 figures, 3 appendixes, 15 sources.

WEB APPLICATION, AUDIO STREAM SYNCHRONIZATION, WEBSOCKET, REAL-TIME, VOTING, LARAVEL, LATENCY COMPENSATION, LIVEWIRE, REDIS.

Research object – software engineering processes, development, operation, maintenance, and quality assurance of real-time web applications.

Research subject – methods and technologies for audio stream synchronization, interactive voting, WebSocket connection scaling, and software quality assurance.

Purpose of the work – reducing the synchronization latency of audio content playback between participants of a shared listening session by developing a web application with a client-side network latency compensation mechanism and interactive voting for the next track.

Materials, methods, and technical means. Object-oriented design methods and MVC, Observer, and Command patterns were used. The server side was implemented using the Laravel 12 framework with PHP 8.3, Eloquent ORM, and MySQL 8.0 database. WebSocket connections are provided by Laravel Reverb, and task queues are managed by Laravel Horizon based on Redis. The client side was implemented using Livewire 3, Alpine.js, and Tailwind CSS. The SoundCloud Widget API was used for audio content playback. The deployment environment is organized via Docker Compose.

Results. The Aurasync web application was developed to provide synchronized audio content listening for multiple participants simultaneously. A client-side network latency compensation algorithm was implemented, along with a voting system for the next track with automatic winner determination via a queue

mechanism, a real-time chat with moderation and anti-spam protection, and a role-based access system with three access levels. Functional testing with 18 test cases was conducted, all passed successfully.

Conclusions. The purpose of the work has been achieved: the implemented latency compensation mechanism provides acceptable synchronization accuracy between participants. According to the experimental study, the average deviation of playback position when joining an active session is ±0.28 s, the delay for a play command is 1.06 s, and for seeking – 0.80 s on average. The results confirm the practical applicability of the playback command synchronization approach via WebSocket for shared audio content listening.

Field of application – entertainment and social web platforms, shared music, podcast, and audiobook listening services.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AJAX – Asynchronous JavaScript and XML;

API – Application Programming Interface;

CRUD – Create, Read, Update, Delete;

CSRF – Cross-Site Request Forgery;

CSS – Cascading Style Sheets;

DOM – Document Object Model;

HTML – HyperText Markup Language;

HTTP – HyperText Transfer Protocol;

JSON – JavaScript Object Notation;

MVC – Model-View-Controller;

ORM – Object-Relational Mapping;

PHP – PHP: Hypertext Preprocessor;

RBAC – Role-Based Access Control;

Redis – Remote Dictionary Server;

RTT – Round-Trip Time;

SPA – Single Page Application;

SQL – Structured Query Language;

URL – Uniform Resource Locator;

WebSocket – двонаправлений протокол зв'язку між клієнтом і сервером у реальному часі;

СУБД – система управління базами даних.

ВСТУП

Останніми роками застосунки реального часу стали невід'ємною частиною повсякденного життя. Сервіси для спільного споживання медіаконтенту набувають дедалі більшої популярності, однак більшість з них зосереджені виключно на синхронізації відтворення і не надають користувачам інструментів для колективного вибору контенту.

Тема роботи є актуальною, оскільки на сьогодні практично відсутні доступні вебзастосунки, які б поєднували синхронізоване прослуховування аудіоконтенту з можливістю голосування за наступний трек та спілкування учасників у реальному часі. Існуючі платформи, наприклад, Spotify Sessions, або вимагають платної підписки, або не підтримують інтерактивного вибору контенту аудиторією.

Метою роботи є зменшення затримки синхронізації відтворення аудіоконтенту між учасниками спільної сесії прослуховування шляхом розроблення вебзастосунку з механізмом компенсації мережевої затримки на стороні клієнта та інтерактивним голосуванням за наступний трек.

Для досягнення мети необхідно вирішити наступні завдання:

– проаналізувати існуючі рішення для спільного прослуховування медіаконтенту та визначити їх переваги й недоліки;

– обґрунтувати вибір технологічного стеку та архітектурних рішень;

– спроєктувати архітектуру системи, схему бази даних та механізми взаємодії компонентів;

– реалізувати ключові модулі застосунку: синхронізацію аудіопотоку через WebSocket, систему голосування та чат реального часу;

– провести тестування та дослідження характеристик затримки синхронізації.

Об'єктом дослідження є процеси програмної інженерії – створення, експлуатація та супровід вебзастосунків реального часу.

Предметом дослідження є методи та технології синхронізації аудіопотоку, інтерактивного голосування, масштабування WebSocket-з'єднань та забезпечення якості програмного забезпечення.

Методи дослідження: об'єктно-орієнтоване проєктування, патерни проєктування MVC та Observer, експериментальне вимірювання затримок синхронізації, функціональне та навантажувальне тестування.

Практичне значення роботи полягає у розробленні вебзастосунку Aurasync, який можна використовувати для організації спільних сесій прослуховування музики, подкастів та аудіокниг з можливістю голосування за наступний трек.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області та актуальність задачі

Спільне прослуховування аудіоконтенту – це усталена соціальна практика, яка з розвитком стрімінгових сервісів поступово перейшла у цифровий простір. Технічна реалізація такого підходу вимагає вирішення нетривіальної задачі синхронізації відтворення між географічно розподіленими учасниками з урахуванням мережевих затримок.

Основна проблема полягає в тому, що учасники підключаються через мережу Інтернет, де затримка передачі пакетів може варіюватись від кількох до сотень мілісекунд залежно від географічного розташування та стану мережі. Пряма трансляція аудіопотоку в реальному часі вимагає значної пропускної здатності і складної інфраструктури. Натомість підхід із синхронізацією команд відтворення (play, pause, seek) дозволяє досягти ефекту спільного прослуховування при значно менших вимогах до інфраструктури.

Крім синхронізації, важливою є інтерактивність – можливість учасників впливати на перебіг сесії через голосування за наступний трек. Це перетворює пасивне прослуховування на спільний досвід і підвищує залученість користувачів.

1.2 Огляд існуючих рішень

На ринку існує кілька рішень для спільного прослуховування медіаконтенту, кожне з яких має певні обмеження.

1.2.1 Spotify Group Session

Spotify Group Session – функція преміум-підписки платформи Spotify, що дозволяє до п'яти учасників слухати музику синхронно [1]. Серед переваг – глибока інтеграція з каталогом Spotify, висока якість синхронізації та

підтримка мобільних платформ. Основні недоліки: вимагає преміум-підписки від усіх учасників, не підтримує голосування за наступний трек, є закритою платформою без API для розробників та обмежує кількість учасників до п'яти.

1.2.2 SyncWave (syncwave.xyz)

SyncWave – вебзастосунок для спільного прослуховування музики через YouTube з автоматичною синхронізацією відтворення [2]. Переваги: безкоштовний доступ без реєстрації, підтримка чату, можливість формування плейлиста. Недоліки: відсутній механізм голосування за наступний трек; підтримується лише Youtube як єдине джерело контенту; обмежені можливості модерації; відсутня система ролей; виявлено недостатню обробку помилок на стороні клієнта – зокрема, у разі введення некоректних даних інтерфейс відображає необроблені відповіді сервера замість зрозумілих повідомлень для користувача.

1.2.3 Watch2Gether

Watch2Gether – вебзастосунок для спільного перегляду відео та прослуховування музики з підтримкою YouTube, SoundCloud та інших джерел [3]. Серед переваг – підтримка кількох джерел контенту, безкоштовний базовий доступ та наявність чату. Недоліки: відсутній механізм голосування за наступний трек, обмежені можливості модерації, відсутня система ролей, частина функціоналу доступна лише за платною підпискою.

1.2.4 Порівняльний аналіз

Аналіз існуючих рішень показав, що жодне з них не поєднує синхронізоване прослуховування з механізмом голосування за наступний трек

та гнучкою системою ролей. Саме це і стало головним обґрунтуванням для розроблення власного застосунку.

Порівняльну характеристику розглянутих рішень наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Критерій	Spotify Group Session	SyncWave	Watch2Gether	Aurasync (розроб.)
Синхронізація відтворення	+	+	+	+
Голосування за трек	–	–	–	+
Чат у реальному часі	–	+	+	+
Безкоштовний доступ	–	+	+	+
Без реєстрації (гість)	–	+	+	+
Система ролей	–	–	–	+
Модерація чату	–	–	частково	+
Кілька джерел контенту	–	–	+	+
Надійність інтерфейсу	+	низька	+	+

1.3 Формулювання вимог до системи

1.3.1 Функціональні вимоги

На основі аналізу предметної області та виявлених недоліків існуючих рішень визначено наступні функціональні вимоги до системи Aurasync:

а) управління сесіями прослуховування:

1) організатор повинен мати можливість створювати сесії з унікальним кодом для запрошення учасників;

2) учасники повинні мати можливість приєднуватись до сесії за кодом без обов'язкової реєстрації (у режимі гостя);

3) система повинна підтримувати три ролі: організатор, авторизований учасник та гість;

б) управління відтворенням:

1) організатор повинен мати можливість керувати відтворенням через вбудований плеєр SoundCloud;

2) синхронізація позиції відтворення між усіма учасниками повинна здійснюватись із затримкою менше 2 секунд;

3) при приєднанні нового учасника під час активного відтворення система повинна автоматично встановлювати актуальну позицію;

в) система голосування:

1) організатор повинен мати можливість формувати пул треків через посилання SoundCloud;

2) організатор повинен мати можливість запускати раунд голосування, обираючи від 2 до 5 варіантів треків з пулу;

3) авторизовані учасники повинні мати можливість голосувати за один трек протягом встановленого часу;

4) результати голосування повинні оновлюватись у реальному часі для всіх учасників;

5) після завершення голосування переможний трек повинен автоматично розпочинати відтворення;

г) чат:

1) авторизовані учасники повинні мати можливість надсилати текстові повідомлення у реальному часі;

2) організатор повинен мати можливість видаляти повідомлення та блокувати учасників чату;

3) система повинна обмежувати частоту надсилання повідомлень для запобігання спаму.

1.3.2 Нефункціональні вимоги

До нефункціональних вимог системи належать:

– продуктивність: затримка синхронізації між учасниками не повинна перевищувати 2 секунди за нормальних мережевих умов;

– масштабованість: архітектура системи повинна забезпечувати можливість горизонтального масштабування WebSocket-з'єднань через Redis pub/sub;

– надійність: система повинна автоматично відновлювати синхронізацію після розриву з'єднання учасника;

– безпека: доступ до функцій модерації та управління відтворенням повинен бути обмежений відповідними ролями;

– зручність використання: інтерфейс повинен бути адаптивним та коректно відображатись на мобільних пристроях.

1.4 Постановка задачі дослідження

На основі проведеного аналізу визначено задачу дослідження: розробити вебзастосунок Aurasync для синхронізованого прослуховування аудіоконтенту з інтерактивним голосуванням, що відповідає визначеним

функціональним та нефункціональним вимогам, та дослідити характеристики затримки синхронізації між учасниками сесії.

Застосунок повинен реалізовувати модель синхронізації на основі команд відтворення через WebSocket-з'єднання з компенсацією мережевої затримки на стороні клієнта. Ключовим аспектом дослідження є залежність точності синхронізації від мережевих умов та кількості одночасних учасників.

Очікується, що результати роботи підтвердять практичну застосовність обраного підходу до синхронізації та доведуть, що сучасні вебтехнології на базі WebSocket забезпечують достатню точність для задоволення вимог спільного прослуховування аудіоконтенту.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку для розроблення застосунку Aurasync базувався на кількох ключових критеріях: підтримка WebSocket-з'єднань реального часу, наявність зрілої екосистеми для швидкої розробки, можливість горизонтального масштабування та доступність документації. Нижче наведено обґрунтування вибору кожного компонента стеку.

2.1.1 Laravel 12 як серверний фреймворк

Laravel – це PHP-фреймворк з відкритим вихідним кодом, який реалізує архітектурний патерн MVC. Для даного проєкту Laravel було обрано з таких причин:

– Laravel надає вбудовану підтримку черг завдань, широкомовлення подій та WebSocket через пакет Laravel Reverb – все це є критично важливим для реалізації синхронізації в реальному часі;

– фреймворк включає потужний ORM Eloquent, що суттєво спрощує роботу з базою даних та дозволяє описувати зв'язки між сутностями на рівні моделей;

– Laravel має розвинену екосистему офіційних пакетів: Breeze для автентифікації, Horizon для моніторингу черг, що скорочує час розробки типових компонентів.

Laravel 12, випущений у лютому 2025 року, є актуальною стабільною версією фреймворку з повною підтримкою PHP 8.3 та всіх використаних у проєкті пакетів [4].

2.1.2 Laravel Reverb як WebSocket-сервер

Laravel Reverb – офіційний WebSocket-сервер від команди Laravel, представлений у 2024 році. Він є самостійним процесом, який запускається поруч із основним застосунком і забезпечує двосторонній зв'язок між сервером та клієнтами в реальному часі [5].

Reverb було обрано замість альтернатив з кількох причин:

– як офіційний пакет Laravel, він безшовно інтегрується з існуючою інфраструктурою фреймворку та не вимагає зовнішніх сервісів;

– Reverb підтримує горизонтальне масштабування через Redis pub/sub, що відповідає нефункціональним вимогам системи;

– будучи самостійним рішенням, він не передбачає залежності від хмарних сервісів і може бути розгорнутий на власній інфраструктурі.

2.1.3 Livewire 3 та Alpine.js для фронтенду

Livewire – це фреймворк для створення реактивних інтерфейсів у Laravel без написання окремого SPA на JavaScript. Компоненти Livewire виконуються на сервері і автоматично синхронізують стан з браузером через AJAX-запити [6].

У проєкті Aurasync Livewire використовується для реалізації компонентів із серверною логікою: панелі голосування, чату та панелі організатора. Така архітектура дозволяє зберігати бізнес-логіку на сервері та уникати дублювання валідації на клієнті.

Alpine.js – легкий JavaScript-фреймворк для додавання інтерактивності безпосередньо в HTML-розмітці [7]. Livewire 3 включає Alpine.js як залежність і надає механізм двостороннього зв'язку між ними через директиву @entangle. У проєкті Alpine.js відповідає за клієнтську логіку плеєра: ініціалізацію SoundCloud Widget API, обробку подій відтворення та синхронізацію стану між компонентами.

Розподіл відповідальності між Livewire і Alpine.js є принциповим архітектурним рішенням: Livewire керує серверним станом (голосування, чат, пул треків), а Alpine.js – клієнтським (плеєр, WebSocket-слухачі). Це дозволяє уникнути конфліктів між серверними оновленнями DOM та клієнтськими маніпуляціями з плеєром.

2.1.4 Redis як брокер повідомлень та сховище кешу

Redis – це сховище даних у оперативній пам'яті, яке підтримує різноманітні структури даних та механізм pub/sub [8]. У проєкті Redis виконує кілька функцій:

– є драйвером для черг завдань Laravel Queue. Зокрема, після запуску голосування у чергу додається завдання FinalizeVotingRound з відкладеним виконанням – воно спрацьовує після закінчення таймера голосування і визначає переможця;

– використовується як pub/sub-брокер для Laravel Reverb, що забезпечує можливість горизонтального масштабування WebSocket-з'єднань між кількома серверними процесами;

– слугує сховищем для кешу та сесій застосунку, що знижує навантаження на базу даних.

2.1.5 Laravel Horizon для моніторингу черг

Laravel Horizon – офіційна панель моніторингу для черг Laravel, що базується на Redis. Horizon надає вебінтерфейс для перегляду стану черг, статистики виконання завдань та управління воркерами [9].

У контексті проєкту Horizon є важливим інструментом для відстеження виконання завдань FinalizeVotingRound – можна в реальному часі бачити чи завдання виконалось успішно чи потрапило до черги невдалих завдань.

Horizon також автоматично перезапускає воркери у разі збоїв, що підвищує надійність системи голосування.

2.1.6 SoundCloud Widget API як джерело аудіоконтенту

SoundCloud Widget API – це JavaScript-бібліотека, яка надає програмний доступ до вбудованого плеєра SoundCloud через iframe. API дозволяє керувати відтворенням, отримувати метадані треку та підписуватись на події плеєра [10].

SoundCloud було обрано як джерело аудіоконтенту з таких міркувань:

– платформа надає відкритий публічний Widget API без необхідності реєстрації застосунку;

– підтримує великий каталог музики включно з авторськими треками незалежних виконавців;

– дозволяє завантажувати метадані треку через oEmbed API за посиланням. Це спрощує процес додавання треків до пулу: організатор вводить посилання на трек, застосунок автоматично отримує назву через oEmbed і зберігає її разом з посиланням.

Архітектура застосунку спроєктована таким чином, що SoundCloud є змінним компонентом – інтерфейс взаємодії з плеєром винесено в окремий модуль на Alpine.js, що теоретично дозволяє замінити джерело контенту без зміни серверної логіки.

2.1.7 MySQL як реляційна база даних

MySQL 8.0 обрано як систему управління базою даних через широку підтримку з боку Laravel Eloquent ORM, стабільність та зрілість рішення [11]. MySQL добре підходить для даного проєкту, оскільки структура даних є реляційною з чітко визначеними зв'язками між сутностями.

Перелік використаних технологій з відповідними версіями наведено у таблиці 2.1.

Таблиця 2.1 – Технологічний стек застосунку Aurasync

Технологія / інструмент	Версія та призначення
PHP	8.3 – серверна мова програмування
Laravel	12 – серверний MVC-фреймворк
Laravel Reverb	2.x – WebSocket-сервер
Laravel Horizon	5.x – моніторинг черг
Livewire	3.x – реактивні серверні компоненти
Alpine.js	3.x – клієнтська інтерактивність
Redis	7.2 – черги, кеш, pub/sub
MySQL	8.0 – реляційна база даних
SoundCloud Widget API	— – джерело аудіоконтенту
Docker Engine	24.x – контейнеризація середовища
Nginx	1.25 – вебсервер
Vite	7.x – збірка фронтенд-ресурсів

2.2 Архітектура системи

2.2.1 Загальна архітектура

Застосунок Aurasync побудований за багаторівневою архітектурою, яка включає рівень клієнта (браузер), рівень вебсервера (Nginx + PHP-FPM), рівень застосунку (Laravel), рівень черг (Horizon + Redis) та рівень бази даних (MySQL). Окремим процесом виступає WebSocket-сервер Reverb, який працює паралельно з основним застосунком.

Загальну схему архітектури системи наведено на рисунку 2.1.

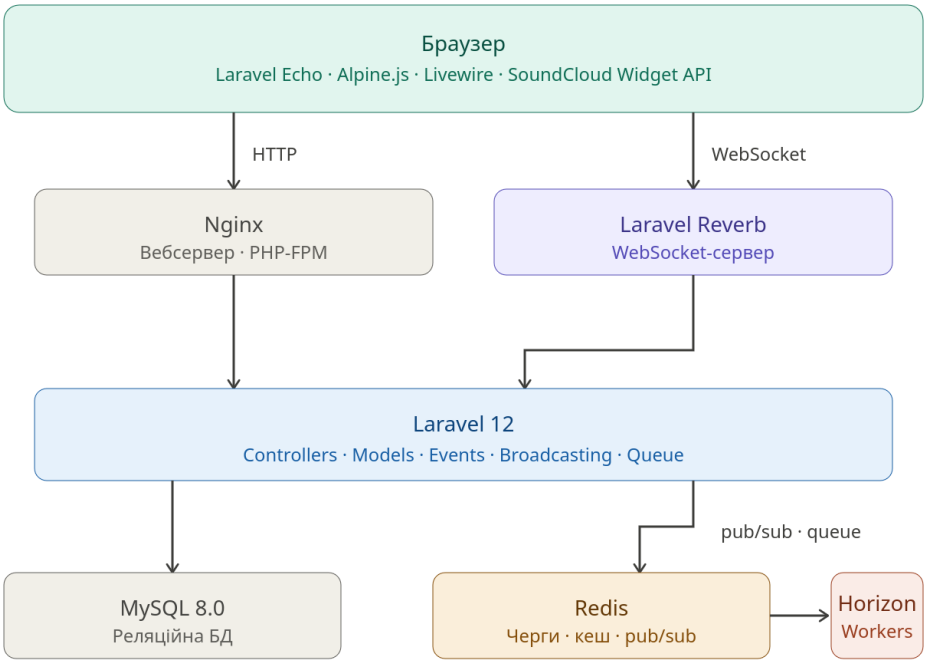


Рисунок 2.1 – Загальна архітектура системи Aurasync

Взаємодія між компонентами відбувається за двома каналами. Перший – стандартний HTTP-канал: браузер надсилає HTTP-запити до Nginx, який передає їх до PHP-FPM для обробки Laravel. Другий – WebSocket-канал: браузер встановлює постійне з'єднання з Reverb через Laravel Echo, і саме цим каналом надходять події синхронізації, голосування та чату в реальному часі.

Livewire-компоненти на сторінці використовують власний механізм оновлення через HTTP, тоді як Alpine.js-компоненти слухають WebSocket-події напряму через Echo. Такий розподіл є принциповим: Livewire відповідає за дані з сервера, Alpine.js – за реакцію на події реального часу.

2.2.2 Патерни проєктування

У процесі розроблення застосунку використано кілька архітектурних патернів.

MVC – базовий патерн Laravel. Моделі інкапсулюють логіку роботи з даними та зв'язки між сутностями. Контролери обробляють HTTP-запити [12]. Представлення реалізовані через Blade-шаблони та Livewire-компоненти.

Observer – реалізований через механізм подій Laravel. Коли організатор виконує дію (play, pause, запуск голосування), застосунок генерує відповідну подію, яка транслюється через Reverb усім підключеним клієнтам. Клієнти підписані на ці події через Laravel Echo і реагують на них відповідно до своєї ролі.

Command – реалізований через систему черг Laravel. Завдання FinalizeVotingRound є інкапсульованою командою із відкладеним виконанням, яка підраховує результати голосування після завершення таймера.

2.2.3 Контейнеризація

Середовище розробки та розгортання застосунку організоване за допомогою Docker та Docker Compose [13]. Система складається з шести контейнерів: app, nginx, mysql, redis, reverb та horizon.

Контейнери reverb і horizon побудовані на основі того самого образу, що й app – це забезпечує ідентичне середовище виконання для всіх процесів Laravel. Всі контейнери об'єднані в спільну Docker-мережу, що дозволяє їм звертатись один до одного за іменем контейнера.

2.3 Проєктування бази даних

2.3.1 Концептуальна модель даних

База даних застосунку містить дев'ять таблиць, які можна поділити на чотири групи: управління користувачами, управління сесіями, система голосування та чат.

Центральною сутністю системи є таблиця listening_sessions, яка зберігає стан сесії прослуховування. Вона пов'язана з таблицею users через зовнішній ключ host_user_id та з таблицею tracks через current_track_id.

ER-діаграму бази даних наведено на рисунку 2.2.

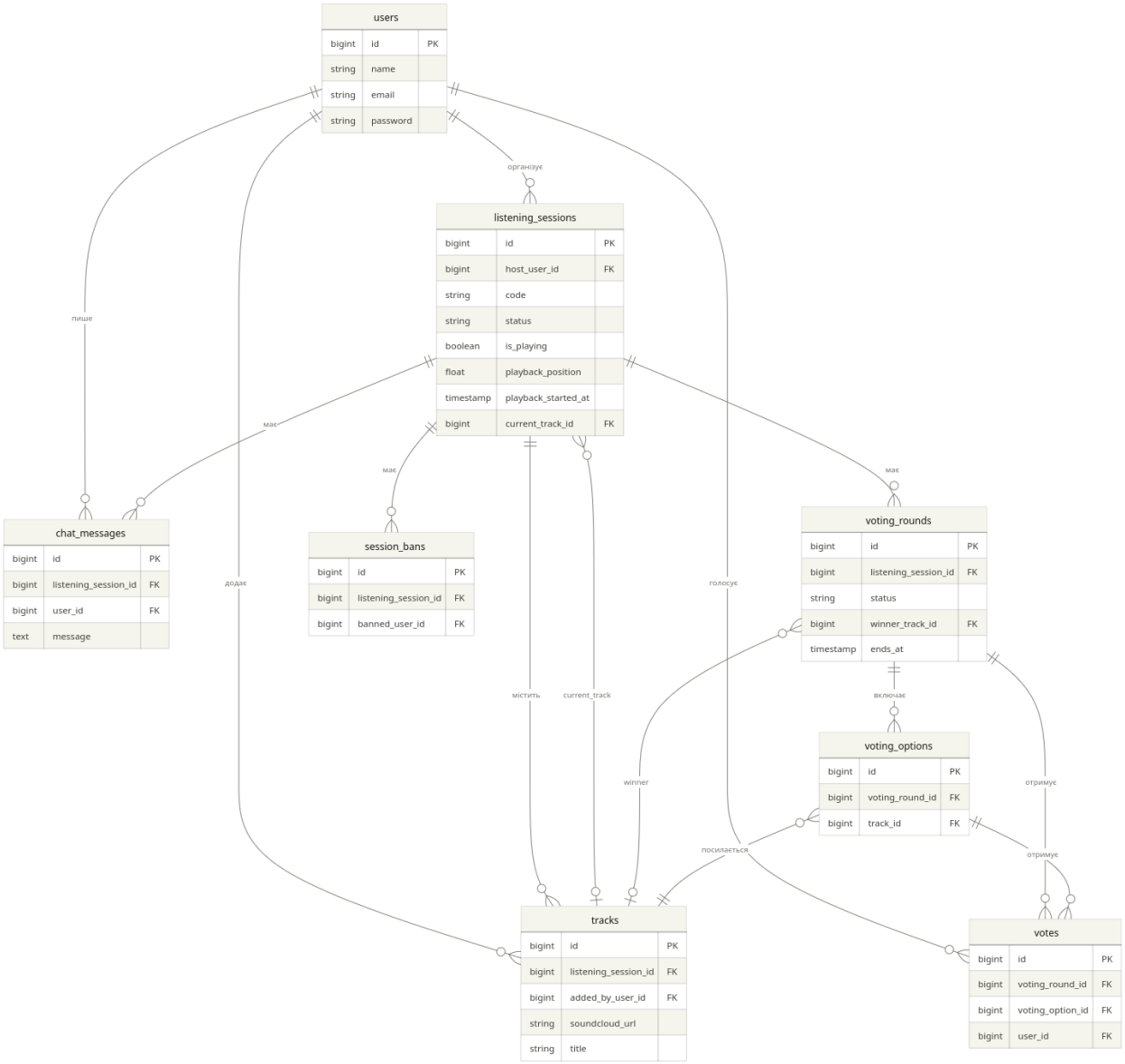


Рисунок 2.2 – ER-діаграма бази даних

2.3.2 Опис таблиць бази даних

Таблиця users зберігає дані зареєстрованих користувачів: ім'я, email та хешований пароль. Стандартна таблиця Laravel Breeze без змін.

Таблиця listening_sessions є центральною сутністю системи. Окрім зовнішніх ключів, вона містить поля для відстеження стану відтворення: is_playing, playback_position та playback_started_at. Ці три поля разом дозволяють розрахувати актуальну позицію відтворення в будь-який момент без постійного оновлення бази даних – достатньо знати позицію на момент останнього натискання play та час, що минув відтоді.

Таблиця tracks зберігає пул треків сесії. Кожен трек містить посилання на SoundCloud та назву, отриману через oEmbed API. Треки не утворюють чергу відтворення – наступний трек завжди визначається через механізм голосування.

Таблиці voting_rounds та voting_options реалізують систему голосування. Один раунд може містити від 2 до 5 варіантів, кожен з яких посилається на конкретний трек з пулу. Після завершення раунду поле winner_track_id заповнюється ідентифікатором переможного треку.

Таблиця votes реалізує голоси учасників. Унікальний складений індекс по полях voting_round_id та user_id гарантує, що кожен користувач може проголосувати лише один раз у межах одного раунду – це обмеження забезпечується як на рівні бази даних, так і на рівні застосунку.

Таблиця chat_messages зберігає повідомлення чату з прив'язкою до сесії та користувача.

Таблиця session_bans реалізує блокування учасників чату. Унікальний складений індекс по listening_session_id та banned_user_id гарантує відсутність дублікатів.

Структуру основних таблиць бази даних наведено у таблиці 2.2.

Таблиця 2.2 – Структура основних таблиць бази даних

Таблиця	Ключові поля
1	2
users	id, name, email, password

Продовження таблиці 2.2

1	2
listening_sessions	id, host_user_id, code, status, is_playing, playback_position, playback_started_at, current_track_id
tracks	id, listening_session_id, soundcloud_url, title
voting_rounds	id, listening_session_id, status, winner_track_id, ends_at
voting_options	id, voting_round_id, track_id
votes	id, voting_round_id, voting_option_id, user_id
chat_messages	id, listening_session_id, user_id, message
session_bans	id, listening_session_id, banned_user_id

2.4 Механізм синхронізації відтворення

2.4.1 Підхід до синхронізації

Існує два основних підходи до синхронізації відтворення між учасниками: трансляція аудіопотоку в реальному часі та синхронізація команд відтворення.

Перший підхід передбачає передачу аудіоданих від сервера до всіх учасників одночасно. Його перевага – висока точність синхронізації, оскільки всі учасники отримують одні й ті самі дані в один момент часу. Недоліком є значні вимоги до пропускної здатності каналу та складність інфраструктури – потрібен медіасервер, кодеки, буферизація.

Другий підхід, обраний у даному проєкті, базується на передачі команд відтворення та поточної позиції через WebSocket [14]. Кожен клієнт відтворює аудіо самостійно зі свого джерела, а сервер лише координує позицію та стан відтворення. Цей підхід є значно простішим в реалізації та не вимагає

спеціальної медіаінфраструктури, однак потребує механізму компенсації мережевої затримки.

2.4.2 Алгоритм компенсації мережевої затримки

Основна проблема синхронізації команд полягає в тому, що між моментом, коли організатор натискає play, і моментом, коли клієнт отримує відповідну подію через WebSocket, минає певний час. Якщо просто встановити позицію відтворення без урахування цієї затримки, клієнт буде відставати від організатора.

Для вирішення цієї проблеми реалізовано алгоритм компенсації затримки на стороні клієнта. Алгоритм працює наступним чином.

При отриманні події PlaybackUpdated клієнт отримує три значення: playback_position (позиція відтворення у момент генерації події на сервері), server_time (серверний часовий штамп у момент генерації події) та is_playing (стан відтворення). Клієнт розраховує зміщення між серверним та локальним часом: offset = (Date.now() − server_time) / 1000. Після цього розраховується скоригована позиція: adjusted_position = playback_position + offset. Клієнт виконує seekTo(adjusted_position) та play() – тим самим починає відтворення з позиції, яка відповідає поточній позиції організатора.

Схему алгоритму компенсації затримки наведено на рисунку 2.3.

Точність алгоритму залежить від кількох факторів: величини мережевої затримки між клієнтом і сервером, часу буферизації аудіоконтенту на стороні плеєра та затримок при позиціонуванні треку.

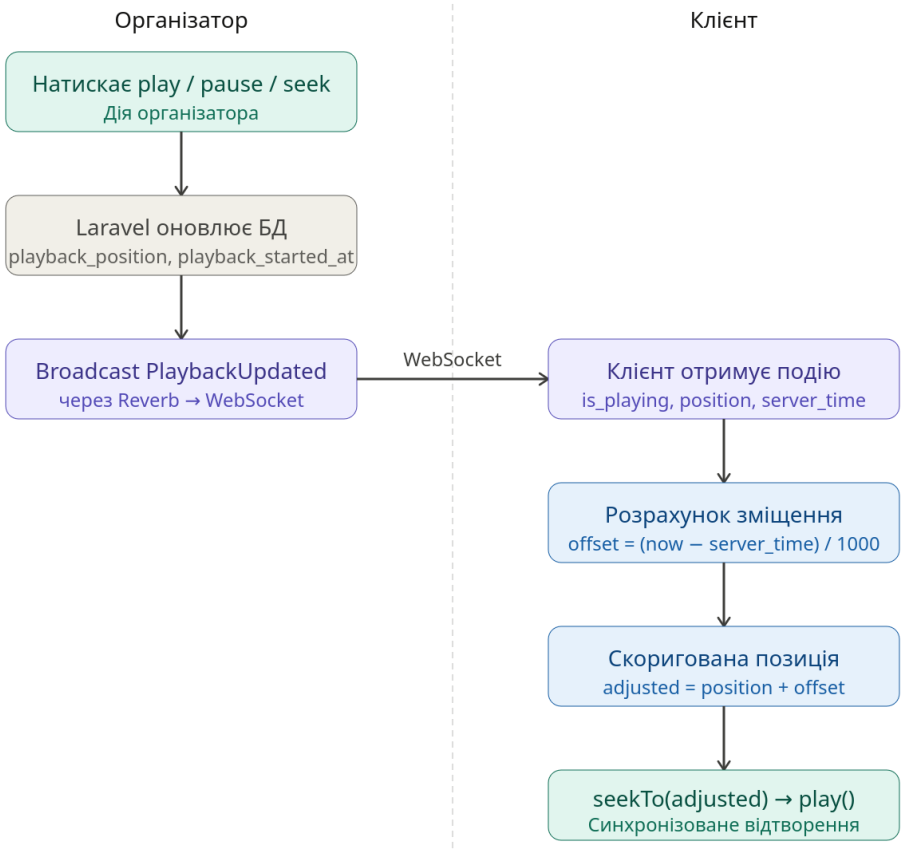


Рисунок 2.3 – Схема компенсації мережевої затримки

2.4.3 Синхронізація при підключенні нового учасника

Окремим випадком є приєднання нового учасника до вже активної сесії. У цьому разі клієнт не отримує жодної WebSocket-події – він просто підключається до вже запущеної сесії.

Для вирішення цієї проблеми реалізовано HTTP-ендпоінт GET /sessions/{code}/state, який повертає актуальний стан сесії у форматі JSON. Ключовою особливістю є те, що сервер не просто повертає збережену позицію з бази даних, а розраховує актуальну позицію динамічно: якщо сесія активна, то поточна позиція = playback_position + (now − playback_started_at). Разом з відповіддю повертається server_time, що дозволяє клієнту застосувати той самий алгоритм компенсації затримки.

Після отримання стану клієнт чекає готовності плеєра і лише тоді виконує seekTo та play. Це необхідно, оскільки SoundCloud Widget API потребує повного завантаження iframe перед тим, як приймати команди.

2.4.4 Трансляція подій через WebSocket

Система подій застосунку реалізована через механізм Broadcasting Laravel у поєднанні з Reverb [15]. Всі події транслюються у публічний канал session.{code}, де {code} – унікальний код сесії. Клієнти підписуються на цей канал через Laravel Echo після приєднання до сесії.

У системі визначено шість типів подій реального часу, перелік яких наведено у таблиці 2.3.

Таблиця 2.3 – Події реального часу системи Aurasync

Подія	Призначення
PlaybackUpdated	Оновлення стану відтворення
TrackChanged	Зміна поточного треку
VotingStarted	Початок нового раунду голосування
VotingEnded	Завершення раунду голосування
Votecast	Новий голос учасника
MessageSent	Нове повідомлення в чаті
SessionEnded	Завершення сесії організатором

2.5 Механізм голосування

2.5.1 Логіка проведення раунду голосування

Система голосування в Aurasync функціонує за принципом обов'язкового раунду перед кожним наступним треком. Організатор вибирає від 2 до 5 треків з пулу, встановлює тривалість голосування і запускає раунд.

При запуску раунду відбувається кілька операцій: у базі даних створюється запис VotingRound зі статусом active та полем ends_at, в базі створюються записи VotingOption для кожного обраного треку, через Reverb транслюється подія VotingStarted з варіантами для голосування, а в чергу Redis з відкладеним виконанням додається завдання FinalizeVotingRound – воно спрацює рівно у момент ends_at.

Схему процесу голосування наведено на рисунку 2.4.

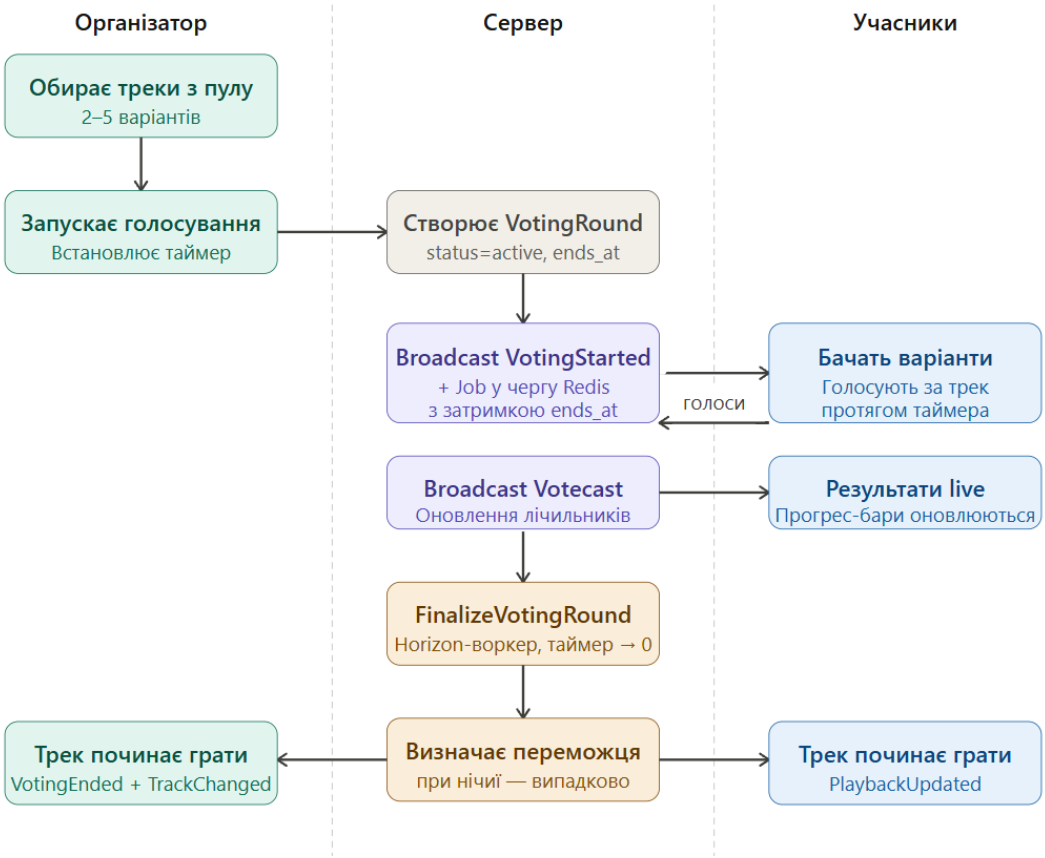


Рисунок 2.4 – Схема процесу голосування

2.5.2 Визначення переможця

Завдання FinalizeVotingRound виконується Horizon-воркером у момент закінчення таймера. Алгоритм визначення переможця такий: підраховується кількість голосів для кожного варіанту, визначається максимальна кількість

голосів, якщо кілька варіантів мають однакову максимальну кількість – переможець обирається випадковим чином серед лідерів.

Після визначення переможця завдання: оновлює статус раунду на ended та записує winner_track_id, оновлює сесію – встановлює current_track_id, is_playing = true, playback_position = 0 та playback_started_at = now(), транслює події VotingEnded, TrackChanged та PlaybackUpdated через Reverb.

Використання черги для завершення голосування є принциповим архітектурним рішенням: якщо б завершення відбувалось через HTTP-запит від клієнта, існував би ризик маніпуляцій з таймером або затримки через мережеві проблеми. Черга гарантує виконання у точно визначений момент на стороні сервера.

2.5.3 Захист від подвійного голосування

Захист від подвійного голосування реалізований на двох рівнях. На рівні бази даних таблиця votes має унікальний складений індекс по полях voting_round_id та user_id – будь-яка спроба вставити дублікат призведе до помилки на рівні СУБД. На рівні застосунку перед створенням голосу перевіряється наявність існуючого голосу від того самого користувача у поточному раунді.

2.6 Методи забезпечення якості та безпеки

2.6.1 Автентифікація та авторизація

Система автентифікації реалізована на базі Laravel Breeze – офіційного пакету для швидкого додавання реєстрації та входу. Паролі зберігаються у вигляді bcrypt-хешів.

Авторизація базується на системі ролей, яка не є формальною RBAC-системою, а реалізована через перевірки в контролерах та компонентах. Роль визначається контекстно: якщо user_id збігається з host_user_id сесії –

користувач є організатором; якщо користувач автентифікований – він є авторизованим учасником; в іншому разі – гостем. Такий підхід є достатнім для даного застосунку і не потребує окремої таблиці ролей.

2.6.2 Захист від спаму в чаті

Для запобігання надмірній активності у чаті реалізовано rate limiting на основі Laravel RateLimiter. Застосовується два обмеження одночасно: не більше 5 повідомлень за 10 секунд від одного користувача в межах однієї сесії та мінімальний інтервал між повідомленнями в 1 секунду. При перевищенні ліміту користувач отримує повідомлення про помилку без блокування акаунту.

2.6.3 Валідація вхідних даних

Всі вхідні дані валідуються на стороні сервера засобами Laravel. Для HTTP-запитів використовуються правила валідації в контролерах. Для Livewire-компонентів валідація виконується безпосередньо в методах компонента перед збереженням даних. Це гарантує цілісність даних незалежно від того, чи обходить клієнт клієнтську валідацію.

3 ОПИС ПРОГРАМИ

3.1 Структура проєкту

3.1.1 Організація файлової системи

Проєкт Aurasync побудований на основі стандартної структури Laravel-застосунку з незначними розширеннями. Серверна частина розміщена у директорії app, яка містить підкаталоги для моделей, контролерів, подій та завдань черги. Фронтендні ресурси знаходяться у директорії resources, де окремо виділено шаблони Blade, Livewire-компоненти та JavaScript-файли.

Основні директорії проєкту та їх призначення наведено у таблиці 3.1.

Таблиця 3.1 – Структура директорій проєкту

Директорія	Призначення
app/Models	Моделі: User, ListeningSession, Track, VotingRound, VotingOption, Vote, ChatMessage, SessionBan
app/Http/Controllers	Контролери: ListeningSessionController, TrackController, VotingRoundController, ChatMessageController
app/Livewire	Livewire-компоненти: HostPanel, VotingPanel, ChatPanel, SessionPlayer, SessionInfo
app/Events	Події: PlaybackUpdated, TrackChanged, VotingStarted, VotingEnded, Votecast, MessageSent, MessageDeleted, UserBanUpdated, SessionEnded
app/Jobs	Завдання черги: FinalizeVotingRound
database/migrations	Міграції бази даних
resources/views	Blade-шаблони сторінок та Livewire-компонентів
resources/js	JavaScript: bootstrap.js з налаштуванням Echo, app.js з Alpine-компонентом sessionPlayer
docker	Конфігурація Docker: PHP Dockerfile, конфіг Nginx

3.1.2 Контейнерна інфраструктура

Середовище розробки та розгортання організоване через Docker Compose [13]. Конфігурація визначає шість сервісів: app, nginx, mysql, redis, reverb та horizon. Всі PHP-сервіси побудовані на одному Dockerfile на базі образу php:8.3-fpm з доданими розширеннями pdo_mysql, mbstring, bcmath, sockets та Redis.

Сервіси reverb та horizon запускаються на основі того самого образу, що й основний застосунок, але з різними командами запуску: php artisan reverb:start для WebSocket-сервера та php artisan horizon для менеджера черг.

3.2 Реалізація серверної частини

3.2.1 Моделі та зв'язки між сутностями

Центральною моделлю системи є ListeningSession, яка містить методи для роботи зі станом сесії. Метод activeVotingRound повертає поточний активний раунд голосування, а метод isBanned перевіряє чи заблокований конкретний користувач у даній сесії. Модель визначає зв'язки hasMany з Track, VotingRound, ChatMessage та SessionBan, а також belongsTo з User через host_user_id та зворотній belongsTo з Track через current_track_id. Реалізацію моделі ListeningSession наведено у Додатку Б.5.

Модель VotingRound пов'язана з VotingOption через hasMany та з Track через belongsTo для поля winner_track_id. Модель Vote має унікальний складений індекс на рівні бази даних по полях voting_round_id та user_id, що гарантує неможливість подвійного голосування навіть при паралельних запитах.

Усі моделі використовують Eloquent ORM з явно визначеними полями у масиві fillable для захисту від масового присвоєння. Поля з датами та булеві поля описані у методі casts для автоматичного перетворення типів.

3.2.2 Маршрутизація та контролери

Маршрутизація застосунку визначена у файлі routes/web.php та розділена на дві групи. Публічні маршрути дозволяють гостям переглядати сторінку сесії та приєднуватись за кодом. Захищені маршрути, обгорнуті у middleware auth, вимагають автентифікації для створення сесій, додавання треків та участі у голосуванні.

ListeningSessionController обробляє основний CRUD для сесій та містить метод state, який повертає актуальний стан відтворення у форматі JSON. Цей метод є публічним і використовується клієнтом при приєднанні до активної сесії для отримання поточної позиції відтворення. Ключова особливість методу – динамічний розрахунок актуальної позиції: якщо сесія активна, позиція обчислюється як збережена позиція плюс час, що минув з моменту останнього запуску відтворення. Реалізацію методу state наведено у Додатку Б.3.

ListeningSessionController також містить метод updatePlayback, який обробляє команди play та pause від організатора. При команді play оновлюються поля is_playing, playback_position та playback_started_at, після чого генерується подія PlaybackUpdated. При команді pause зберігається поточна позиція і скидається is_playing.

3.2.3 Система подій та широкомовлення

У застосунку визначено дев'ять класів подій, кожен з яких реалізує інтерфейс ShouldBroadcast. Всі події транслюються у публічний канал session.{code}, де code – унікальний восьмисимвольний код сесії. Метод broadcastWith кожної події повертає масив даних, що передається клієнту. Реалізацію класу події PlaybackUpdated наведено у Додатку Б.4.

Подія PlaybackUpdated передає чотири поля: is_playing, playback_position, playback_started_at та server_time. Поле server_time є

серверною міткою часу у момент генерації події і використовується клієнтом для розрахунку компенсації мережевої затримки.

Подія VotingStarted передає ідентифікатор раунду, час завершення ends_at та масив варіантів голосування з назвами треків. Це дозволяє клієнту відобразити панель голосування без додаткових запитів до сервера.

Подія SessionEnded не передає жодних даних – її отримання є сигналом для клієнта перейти на головну сторінку.

3.2.4 Завдання черги FinalizeVotingRound

FinalizeVotingRound є єдиним завданням черги у застосунку. Воно додається до черги Redis з відкладеним виконанням у момент запуску голосування: затримка дорівнює різниці між ends_at та поточним часом. Horizon-воркер виконує завдання у точно визначений момент. Повну реалізацію завдання наведено у Додатку Б.2.

Логіка завдання складається з кількох кроків. Спочатку перевіряється, чи раунд все ще активний – це захист від повторного виконання у разі збою. Потім для кожного варіанту підраховується кількість голосів. Визначається максимальна кількість, і якщо кілька варіантів мають однакову максимальну кількість, переможець обирається випадковим чином з-поміж них через колекцію Laravel та метод random. Після визначення переможця оновлюється запис VotingRound та запис ListeningSession. Сесія отримує новий current_track_id, а також скидаються поля відтворення: is_playing встановлюється у true, playback_position у нуль, playback_started_at у поточний час. Завершується завдання трьома broadcast-подіями: VotingEnded, TrackChanged та PlaybackUpdated.

3.3 Реалізація клієнтської частини

3.3.1 Livewire-компонент HostPanel

HostPanel є найбільшим Livewire-компонентом застосунку і відображається виключно для організатора сесії. Компонент реалізує три вкладки: пул треків, голосування та заблоковані учасники.

Вкладка пулу треків містить форму додавання треку через SoundCloud URL. Після введення URL клієнтський код на Alpine.js виконує запит до публічного SoundCloud oEmbed API для отримання назви треку [10]. Отримана назва передається у Livewire через виклик методу resolveTrack, після чого відображається картка підтвердження з кнопкою додавання. При підтвердженні трек зберігається у таблицю tracks з прив'язкою до поточної сесії.

Вкладка голосування відображає список треків з пулу у вигляді чекбоксів. Організатор обирає від 2 до 5 треків та встановлює тривалість голосування від 10 до 120 секунд. Реалізацію методу startVoting наведено у Додатку Б.6.

Вкладка заблокованих учасників відображає список користувачів, заблокованих організатором у чаті поточної сесії, з можливістю розблокування.

3.3.2 Livewire-компонент VotingPanel

VotingPanel відображається для всіх користувачів і показує поточний стан голосування. Якщо активний раунд існує, компонент відображає варіанти треків з прогрес-барами та таймером зворотного відліку. Таймер реалізований на Alpine.js і оновлюється кожну секунду без звернення до сервера.

Авторизовані учасники можуть проголосувати за один варіант. Після голосування кнопка змінює вигляд на позначку підтвердження. Метод vote

перевіряє чи користувач вже голосував у цьому раунді, після чого зберігає голос та транслює подію Votecast.

Компонент підписаний на Livewire-події votingStarted, votingEnded та votecast через масив listeners. При отриманні будь-якої з цих подій компонент оновлюється викликом refresh, що забезпечує актуальне відображення стану голосування у всіх учасників.

3.3.3 Livewire-компонент ChatPanel

ChatPanel реалізує чат реального часу з модерацією. При завантаженні компонент отримує останні 50 повідомлень сесії з бази даних у зворотному хронологічному порядку, після чого перевертає їх для відображення від найстарішого до найновішого.

Відправка повідомлення проходить через rate limiting на двох рівнях: не більше 5 повідомлень за 10 секунд та мінімальний інтервал у 1 секунду між повідомленнями. Laravel RateLimiter використовує Redis як сховище для лічильників, що забезпечує точність підрахунку навіть при паралельних запитах [4].

Організатор бачить додаткові кнопки при наведенні на повідомлення: видалення конкретного повідомлення та блокування автора. При видаленні повідомлення транслюється подія MessageDeleted, яка через Echo доходить до всіх клієнтів і видаляє повідомлення з їхніх списків без перезавантаження. При блокуванні транслюється подія UserBanUpdated, яка оновлює стан isBanned у компоненті заблокованого користувача. Реалізацію методів sendMessage, deleteMessage та banUser наведено у Додатку Б.7.

3.3.4 Alpine.js-компонент sessionPlayer

sessionPlayer є найбільш технічно складним компонентом клієнтської частини. Він реалізований як глобальна функція на Alpine.js і відповідає за всю

логіку відтворення та синхронізації. Компонент виноситься за межі будь-якого Livewire-компонента, щоб уникнути перемальовування DOM при оновленні Livewire.

При ініціалізації компонент отримує початковий стан з атрибутів HTML: код сесії, роль користувача, URL поточного треку, стан відтворення та позицію. Якщо користувач є організатором, одразу ініціалізується плеєр та підписка на WebSocket-канал. Для інших користувачів спочатку відображається кнопка приєднання. Реалізацію методів join та hostPlay наведено у Додатку Б.8.

При натисканні кнопки приєднання виконується HTTP-запит до ендпоінту /sessions/{code}/state для отримання актуального стану. Якщо відтворення активне, компонент чекає готовності плеєра та виконує синхронізацію. Очікування реалізовано через setInterval з інтервалом 200 мс, який зупиняється після встановлення прапорця playerReady.

Для уникнення конфлікту між Alpine.js та SoundCloud Widget API об'єкт плеєра не зберігається у реактивному стані Alpine.js, а отримується щоразу заново через виклик SC.Widget з посиланням на iframe за ідентифікатором. Це запобігає пошкодженню внутрішнього стану Widget при реактивних оновленнях Alpine.js.

3.4 Реалізація синхронізації відтворення

3.4.1 Керування відтворенням організатором

Організатор керує відтворенням через кнопки play та pause, розташовані під плеєром. При натисканні play виконується послідовність дій: спочатку локально встановлюється позиція відтворення та запускається відтворення на SoundCloud Widget, після чого надсилається HTTP-запит до ендпоінту updatePlayback з дією play та поточною позицією. Сервер оновлює БД та транслює PlaybackUpdated. Після отримання відповіді від сервера клієнт

оновлює локальне значення playbackStartedAt для коректного розрахунку позиції при наступній паузі.

При паузі спочатку виконується HTTP-запит до ендпоінту state для отримання актуальної позиції з сервера – це необхідно у випадку, якщо організатор перезавантажив сторінку і локальний стан не відповідає серверному. Після отримання відповіді SoundCloud Widget зупиняється і надсилається запит updatePlayback з дією pause.

Організатор також може перемотувати трек безпосередньо у плеєрі SoundCloud. Для відстеження перемотування використовується подія SEEK з Widget API. При кожній перемотці надсилається запит updatePlayback з новою позицією, що транслює оновлення до всіх учасників.

3.4.2 Синхронізація на стороні клієнта

Клієнти отримують стан відтворення через WebSocket-подію PlaybackUpdated. Обробник події оновлює локальні змінні isPlaying, playbackPosition та playbackStartedAt. Для учасників додатково викликається функція syncPlayback, яка застосовує алгоритм компенсації затримки. Реалізацію функції syncPlayback наведено у Додатку Б.1.

Функція syncPlayback розраховує зміщення між серверним та локальним часом і додає його до отриманої позиції. Після цього виконується подвійний seekTo з затримкою 300 мс між викликами – це необхідно через особливість SoundCloud Widget API, який може ігнорувати перший виклик seekTo якщо трек ще не достатньо буферизований. Після другого seekTo виконується play.

При зміні треку клієнт отримує подію TrackChanged з новим URL. Alpine.js оновлює значення currentTrackUrl, що призводить до перерендеру iframe з новим src. Після завантаження iframe встановлюється прапорець playerReady та виконується запит до ендпоінту state для синхронізації з поточною позицією нового треку.

3.5 Інтерфейс користувача

3.5.1 Загальний опис інтерфейсу

Інтерфейс застосунку реалізований у темній кольоровій схемі з фіолетовим акцентним кольором. Для стилізації використовується Tailwind CSS, який підключено через Vite. Шрифт – Figtree від Google Fonts.

Навігаційна панель містить логотип застосунку, посилання на Dashboard для авторизованих користувачів та елементи автентифікації. Для неавторизованих відображаються кнопки входу та реєстрації.

3.5.2 Головна сторінка

Головна сторінка містить hero-секцію з коротким описом застосунку та двома кнопками – для реєстрації та входу. Нижче розміщена форма приєднання до сесії за кодом: поле вводу з автоматичним переведенням у верхній регістр та кнопка переходу. Ще нижче – секція з трьома карточками, що описують ключові можливості: синхронізацію, голосування та чат.

Скріншот головної сторінки наведено на рисунку 3.1.

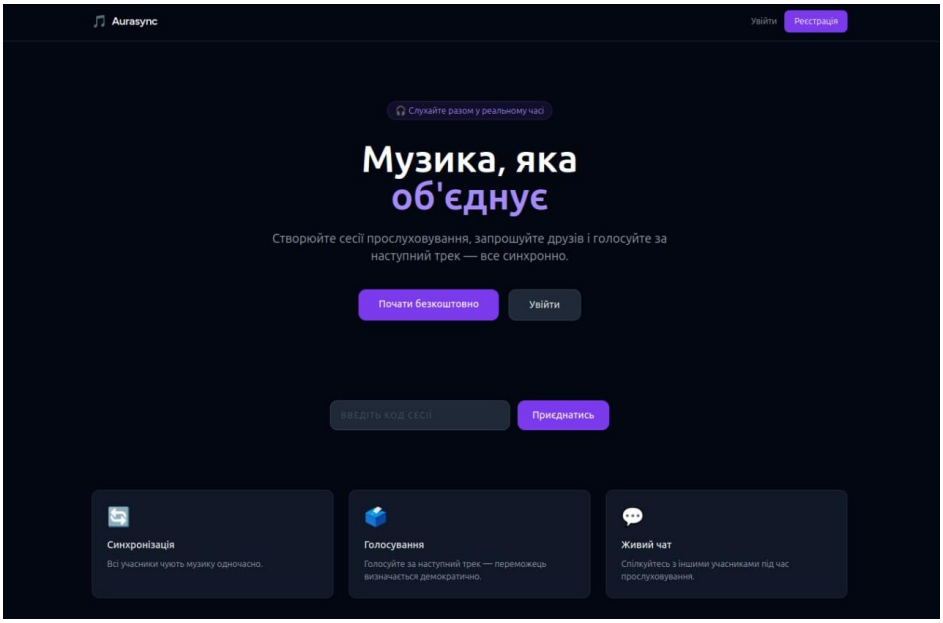


Рисунок 3.1 – Головна сторінка застосунку

3.5.3 Сторінка сесії

Сторінка сесії є основним робочим екраном застосунку. Вона побудована за двоколонковим макетом на великих екранах: ліва колонка містить плеєр, панель голосування та панель організатора, права – чат. На мобільних пристроях колонки розташовуються вертикально.

У верхній частині лівої колонки розміщено плеєр SoundCloud з регулятором гучності під ним. Для учасників плеєр перекритий прозорим шаром, що блокує безпосереднє керування. Під плеєром – кнопки відтворення та паузи для організатора. Якщо користувач ще не приєднався до сесії, замість плеєра відображається кнопка приєднання.

Скріншот сторінки сесії з точки зору організатора наведено на рисунку 3.2.

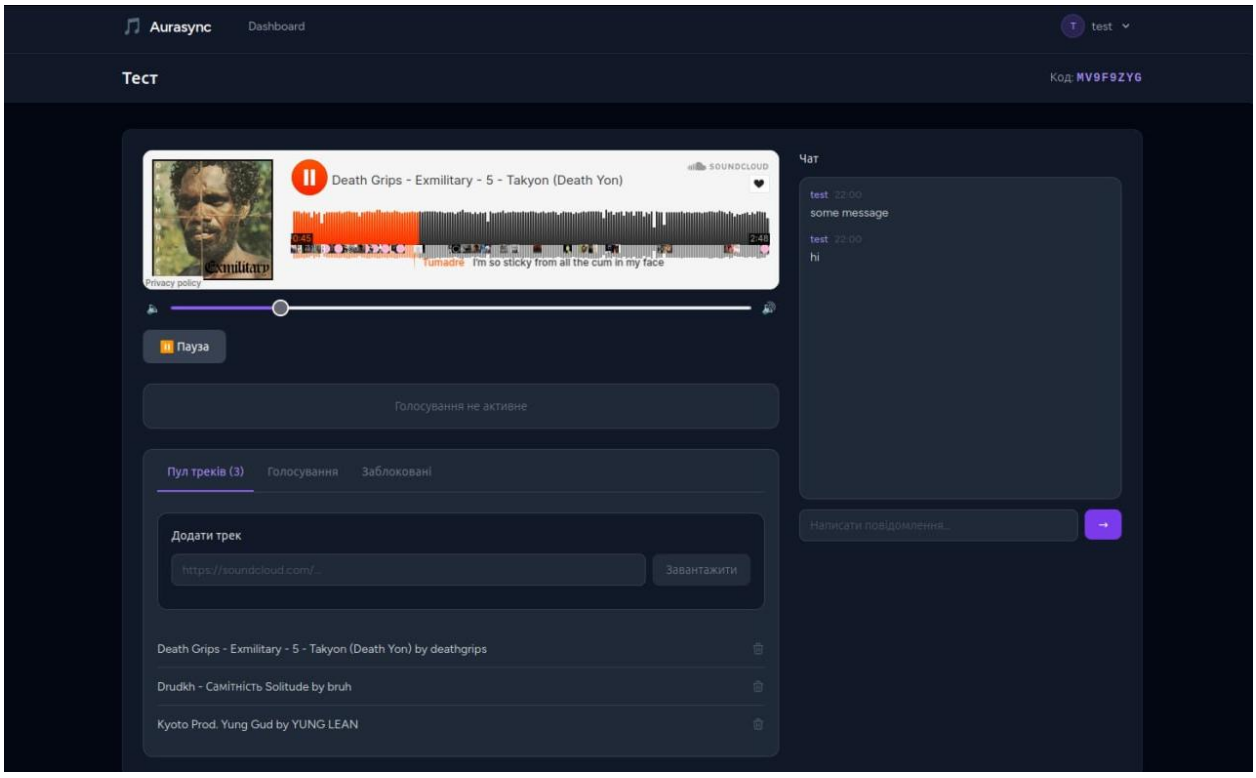


Рисунок 3.2 – Сторінка сесії з боку організатора

Скріншот сторінки сесії з точки зору учасника наведено на рисунку 3.3.

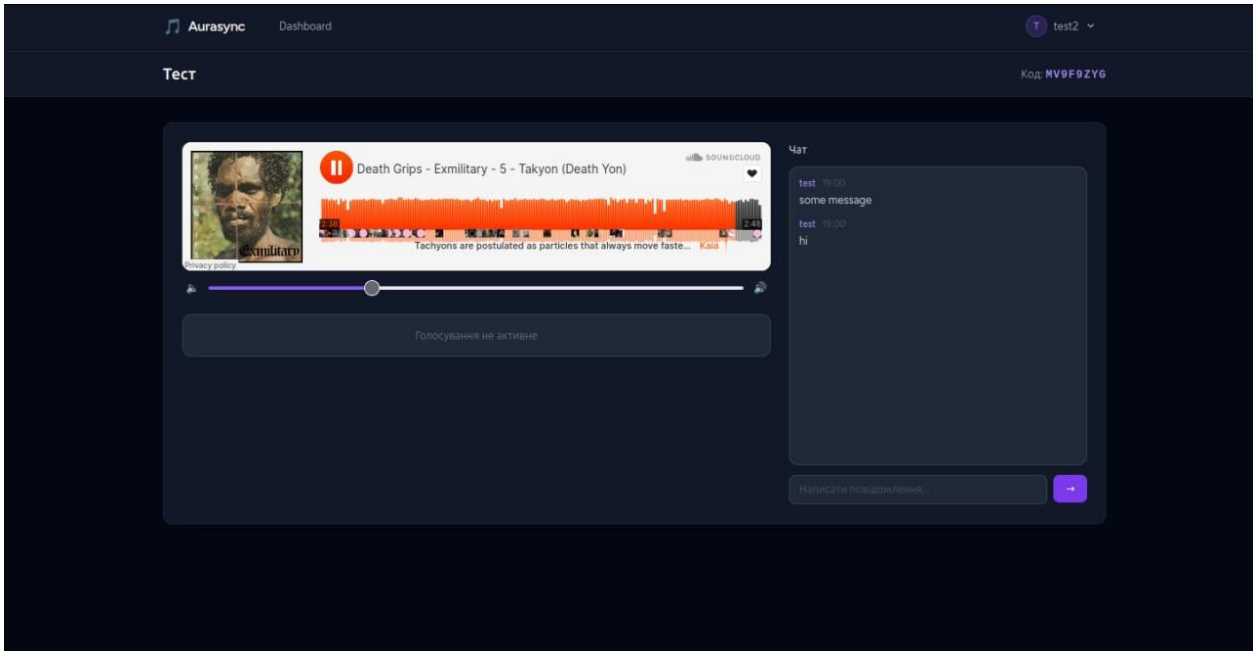


Рисунок 3.3 – Сторінка сесії з боку учасника

3.5.4 Панель голосування

Панель голосування відображається для всіх користувачів під плеєром. Коли голосування неактивне, панель показує повідомлення про очікування. Під час активного раунду відображаються варіанти треків у вигляді карточок з кнопкою голосування, лічильником голосів та прогрес-баром. У правому верхньому куті панелі показується таймер зворотного відліку.

Після голосування кнопка замінюється галочкою підтвердження. Прогрес-бари оновлюються в реальному часі при надходженні нових голосів від інших учасників.

Скріншот панелі голосування під час активного раунду наведено на рисунку 3.4.

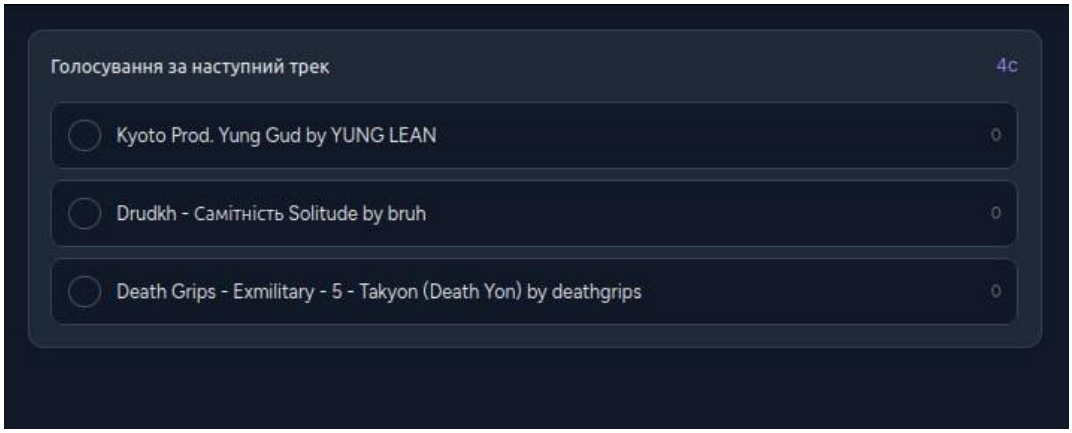


Рисунок 3.4 – Панель голосування під час активного раунду

3.5.5 Панель організатора

Панель організатора розташована під панеллю голосування і містить три вкладки. Вкладка пулу треків містить форму додавання треку через SoundCloud URL та список доданих треків з кнопками видалення. Вкладка голосування дозволяє обрати треки для нового раунду через чекбокси та встановити тривалість. Вкладка заблокованих відображає список заблокованих учасників з кнопками розблокування.

Скріншот панелі організатора, вкладка пулу треків, наведено на рисунку 3.5.

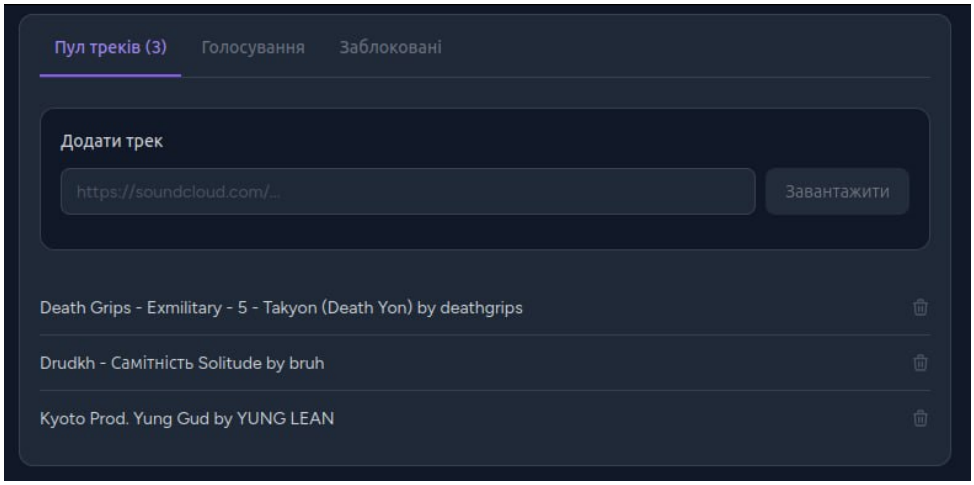


Рисунок 3.5 – Вкладка пулу треків на панелі організатора

3.5.6 Чат

Чат розміщено у правій колонці сторінки сесії. Повідомлення відображаються у хронологічному порядку – найновіші внизу. Кожне повідомлення містить ім'я автора, час відправки та текст.

При наведенні на повідомлення організатор бачить додаткові кнопки: видалення повідомлення та блокування автора. Для заблокованих користувачів поле вводу замінюється повідомленням про блокування. Неавторизованим користувачам пропонується посилання для входу.

Скріншот чату наведено на рисунку 3.6.

Рисунок 3.6 – Чат застосунку

3.5.7 Dashboard та управління сесіями

Dashboard відображає список сесій поточного користувача у вигляді карточок. Кожна карточка містить назву сесії, унікальний код та час

створення. Організатор може видалити сесію через кнопку в правому куті карточки з підтвердженням. При видаленні активної сесії всі підключені учасники отримують подію SessionEnded та перенаправляються на головну сторінку.

Скріншот Dashboard наведено на рисунку 3.7.

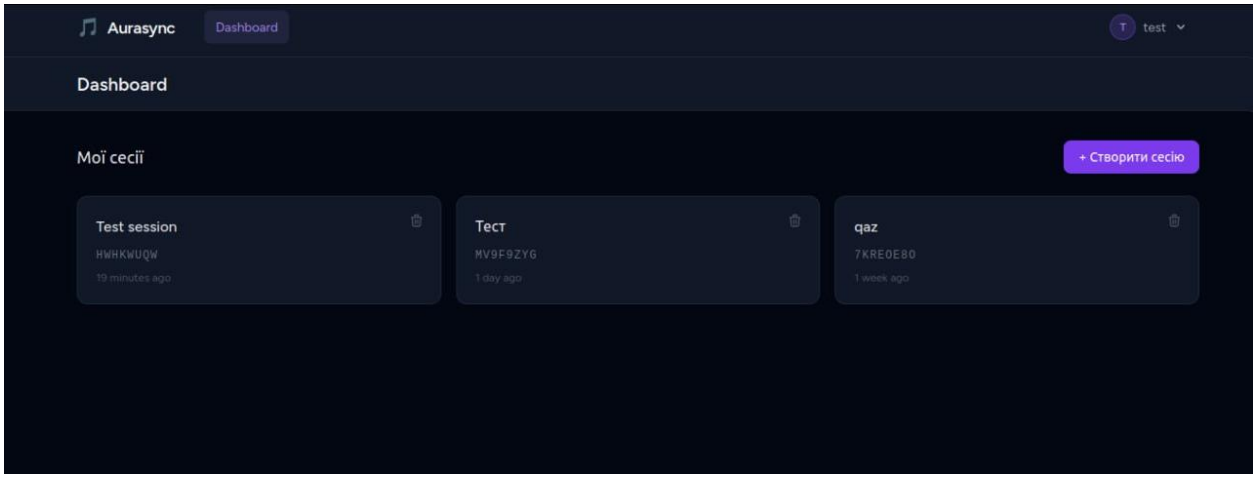


Рисунок 3.7 – Сторінка Dashboard

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Вимоги до технічних та програмних засобів

4.1.1 Вимоги для розробки

Для розгортання середовища розробки необхідне таке програмне та апаратне забезпечення. Мінімальні вимоги до апаратного забезпечення: процесор з тактовою частотою від 2 ГГц та підтримкою віртуалізації, оперативна пам'ять обсягом не менше 8 ГБ, вільний простір на диску не менше 10 ГБ.

Необхідне програмне забезпечення наведено у таблиці 4.1.

Таблиця 4.1 – Вимоги до програмного забезпечення для розробки

Програмне забезпечення	Версія	Призначення
Docker Engine	24.0 або новіше	Контейнеризація сервісів
Docker Compose	2.20 або новіше	Оркестрація контейнерів
Браузер	Chrome / Firefox актуальна	Тестування інтерфейсу
Git	2.x	Керування версіями

4.1.2 Вимоги для кінцевого користувача

Для роботи з застосунком Aurasync кінцевому користувачу не потрібно встановлювати жодного додаткового програмного забезпечення. Достатньо мати сучасний веббраузер з підтримкою WebSocket та JavaScript. Рекомендовані браузери: Google Chrome версії 90 або новіше, Mozilla Firefox версії 88 або новіше, Safari версії 14 або новіше.

Для коректної роботи аудіоплеєра необхідне стабільне підключення до мережі Інтернет зі швидкістю не менше 1 Мбіт/с. Мінімальна роздільна

здатність екрану – 360×640 пікселів для мобільних пристроїв та 1024×768 для настільних комп'ютерів.

4.2 Розгортання та запуск застосунку

4.2.1 Клонування репозиторію та налаштування середовища

Для розгортання застосунку необхідно клонувати репозиторій та перейти у директорію проєкту. Після цього створити файл .env на основі .env.example та заповнити змінні середовища: параметри підключення до бази даних, конфігурацію Redis та налаштування Laravel Reverb.

Ключові змінні середовища, які необхідно налаштувати перед запуском, наведено у таблиці 4.2.

Таблиця 4.2 – Ключові змінні середовища

Змінна	Значення за замовчуванням	Призначення
DB_DATABASE	syncwave	Назва бази даних
DB_USERNAME	syncwave	Користувач бази даних
DB_PASSWORD	—	Пароль бази даних
REDIS_HOST	redis	Хост Redis
BROADCAST_CONNECTION	reverb	Драйвер broadcasting
REVERB_APP_KEY	—	Ключ Reverb-застосунку
REVERB_HOST	localhost	Хост Reverb для клієнта
REVERB_PORT	8080	Порт WebSocket-сервера

4.2.2 Запуск через Docker Compose

Після налаштування змінних середовища запуск застосунку виконується у кілька кроків. Спочатку збираються Docker-образи командою make build. Потім запускаються контейнери командою make up. Після успішного запуску всіх контейнерів виконуються міграції бази даних командою make migrate. Застосунок стає доступним за адресою http://localhost:8000, WebSocket-сервер – за адресою ws://localhost:8080, панель моніторингу Horizon – за адресою http://localhost:8000/horizon.

Для перевірки стану контейнерів використовується команда docker compose ps [13]. Всі шість контейнерів повинні мати статус Up: app, nginx, mysql, redis, reverb та horizon.

4.3 Інструкція з використання програми

4.3.1 Реєстрація та вхід

Для повноцінного використання застосунку необхідно зареєструватись. Реєстрація доступна за посиланням /register та вимагає введення імені, електронної адреси та пароля. Після реєстрації користувач автоматично авторизується та перенаправляється на Dashboard.

Незареєстровані користувачі можуть приєднуватись до сесій як гості через форму на головній сторінці або за прямим посиланням. Гості мають доступ до прослуховування, але не можуть голосувати та писати у чат.

4.3.2 Створення сесії та управління пулом треків

Для створення сесії організатор переходить на сторінку /sessions/create, вводить назву сесії та натискає кнопку створення. Після створення відкривається сторінка сесії з унікальним кодом у правому верхньому куті заголовка.

Для наповнення пулу треків організатор переходить на вкладку "Пул треків" в панелі організатора, вводить посилання на трек SoundCloud та натискає "Завантажити". Застосунок автоматично отримує назву треку і пропонує підтвердити додавання. Після підтвердження трек з'являється у списку пулу. Мінімальна кількість треків у пулі для запуску голосування – два.

4.3.3 Запуск голосування та відтворення

Для запуску голосування організатор переходить на вкладку "Голосування", обирає від 2 до 5 треків з пулу через чекбокси, встановлює тривалість голосування та натискає "Запустити голосування". Всі учасники сесії одразу бачать панель голосування з варіантами та таймером зворотного відліку.

Після завершення таймера система автоматично визначає переможця та розпочинає відтворення. Організатор може поставити трек на паузу в будь-який момент за допомогою відповідної кнопки під плеєром. Для запуску наступного раунду голосування організатор знову переходить на вкладку "Голосування" та повторює процедуру.

4.3.4 Використання чату та модерація

Авторизовані учасники можуть надсилати повідомлення у чат, розташований у правій частині сторінки сесії. Повідомлення надсилаються натисканням кнопки або клавіші Enter. Система обмежує частоту надсилання до 5 повідомлень за 10 секунд.

Організатор має розширені права у чаті: при наведенні на будь-яке повідомлення з'являються кнопки видалення повідомлення та блокування автора. Заблокований користувач бачить відповідне повідомлення замість поля вводу. Список заблокованих учасників з можливістю розблокування доступний у вкладці "Заблоковані" панелі організатора.

4.4 Тестування програми

4.4.1 Функціональне тестування

Функціональне тестування проводилось методом чорної скриньки – перевірялась відповідність поведінки застосунку визначеним функціональним вимогам. Для кожного тест-кейсу фіксувались вхідні дані, очікуваний та фактичний результат.

Результати функціонального тестування наведено у таблиці 4.3.

Таблиця 4.3 – Результати функціонального тестування

Тест-кейс	Очікуваний результат	Фактичний результат	Статус
1	2	3	4
Реєстрація з коректними даними	Акаунт створено, перенаправлення на Dashboard	Відповідає очікуваному	Пройдено
Реєстрація з існуючим email	Повідомлення про помилку	Відповідає очікуваному	Пройдено
Створення сесії	Сесія створена, відкрита сторінка з кодом	Відповідає очікуваному	Пройдено
Приєднання за коректним кодом	Перенаправлення на сторінку сесії	Відповідає очікуваному	Пройдено
Приєднання за некоректним кодом	Повідомлення про помилку	Відповідає очікуваному	Пройдено
Додавання треку через SoundCloud URL	Трек з'являється у пулі	Відповідає очікуваному	Пройдено
Додавання треку з некоректним URL	Повідомлення про помилку	Відповідає очікуваному	Пройдено

Продовження таблиці 4.3

1	2	3	4
Запуск голосування з менше 2 треків	Кнопка недоступна	Відповідає очікуваному	Пройдено
Запуск голосування з 3 треками	Голосування розпочато у всіх учасників	Відповідає очікуваному	Пройдено
Повторне голосування в одному раунді	Другий голос не приймається	Відповідає очікуваному	Пройдено
Завершення таймера голосування	Визначено переможця, розпочато відтворення	Відповідає очікуваному	Пройдено
Відтворення треку організатором	Трек грає у всіх учасників	Відповідає очікуваному	Пройдено
Пауза організатором	Трек зупиняється у всіх учасників	Відповідає очікуваному	Пройдено
Надсилання повідомлення у чат	Повідомлення з'являється у всіх учасників	Відповідає очікуваному	Пройдено
Спам у чат (6 повідомлень за 10с)	Повідомлення про перевищення ліміту	Відповідає очікуваному	Пройдено
Блокування учасника організатором	Заблокований не може писати у чат	Відповідає очікуваному	Пройдено
Гість намагається голосувати	Кнопка голосування не відображається	Відповідає очікуваному	Пройдено
Гість намагається писати в чат	Відображається посилання для входу	Відповідає очікуваному	Пройдено

Усі 18 тест-кейсів пройдено успішно. Застосунок коректно обробляє як штатні сценарії використання, так і граничні випадки – некоректні вхідні дані, спроби несанкціонованого доступу та перевищення лімітів.

4.4.2 Тестування безпеки

В рамках тестування безпеки перевірялись основні вектори атак на вебзастосунки.

Перевірка CSRF-захисту: всі форми та AJAX-запити містять CSRF-токен, який перевіряється на сервері. Спроба надіслати запит без токена повертає відповідь з кодом 419.

Перевірка авторизації: спроба виконати дії організатора від імені звичайного учасника повертає відповідь з кодом 403. Зокрема, учасник не може запустити голосування, видалити трек або заблокувати іншого учасника.

Перевірка валідації: всі вхідні дані валідуються на сервері незалежно від клієнтської перевірки. Введення SQL-ін'єкцій у поля форм не призводить до помилок бази даних завдяки використанню параметризованих запитів Eloquent ORM.

Перевірка захисту від подвійного голосування: спроба проголосувати двічі в одному раунді блокується як на рівні застосунку, так і на рівні унікального індексу бази даних.

4.5 Експериментальне дослідження затримки синхронізації

4.5.1 Методика проведення експерименту

Для оцінки ефективності реалізованого алгоритму компенсації мережевої затримки проведено серію вимірювань у контрольованому середовищі. Експеримент виконувався на локальній машині з двома браузерними вкладками – одна в ролі організатора, інша в ролі учасника.

Мережеве з'єднання між ними відповідало умовам localhost, тобто мережева затримка між процесами була мінімальною і не перевищувала 1 мс.

Досліджувались три сценарії синхронізації: приєднання учасника до активної сесії, команда play від організатора та перемотування треку організатором. Для кожного сценарію проведено 5 вимірювань. Результати фіксувались шляхом візуального порівняння позицій відтворення на обох вкладках.

4.5.2 Результати вимірювань

Результати вимірювань для сценарію приєднання до активної сесії наведено у таблиці 4.4. Додатне значення відхилення означає випередження організатора учасником, від'ємне – відставання.

Таблиця 4.4 – Відхилення позиції відтворення при приєднанні до активної сесії

Вимірювання	1	2	3	4	5	Середнє
Відхилення, с	+0.3	−0.3	+0.1	+0.5	−0.2	±0.28

Результати вимірювань для сценарію команди play наведено у таблиці 4.5.

Таблиця 4.5 – Затримка між командою play організатора та початком відтворення у учасника

Вимірювання	1	2	3	4	5	Середнє
Затримка, с	0.9	1.1	1.3	0.8	1.2	1.06

Результати вимірювань для сценарію перемотування наведено у таблиці 4.6.

Таблиця 4.6 – Затримка між перемотуванням організатора та отриманням нової позиції учасником

Вимірювання	1	2	3	4	5	Середнє
Затримка, с	0.2	1.8	0.4	1.5	0.1	0.8

4.5.3 Аналіз результатів

Результати експерименту показують, що реалізований алгоритм компенсації мережевої затримки забезпечує прийнятну точність синхронізації в умовах локального середовища.

При приєднанні до активної сесії середнє відхилення позиції відтворення становить ±0.28 с. Це пояснюється двома факторами: по-перше, алгоритм компенсації використовує різницю між серверним та клієнтським часом, яка навіть на localhost не є абсолютно нульовою; по-друге, SoundCloud Widget API має власну буферизацію і виконує seekTo з певною затримкою відносно моменту виклику. Відхилення у межах ±0.5 с є суб'єктивно непомітним при прослуховуванні більшості жанрів музики.

Затримка при команді play становить в середньому 1.06 с. Ця величина складається з кількох компонентів: час обробки HTTP-запиту на сервері, час трансляції події через Reverb, час обробки події клієнтом та два виклики seekTo з затримками по 300 мс між ними. Останній компонент є домінуючим і є свідомим архітектурним рішенням для забезпечення надійного позиціонування у SoundCloud Widget. В умовах реальної мережі до цього значення додасться RTT між клієнтом та сервером.

Найбільший розкид спостерігається при перемотуванні – від 0.1 до 1.8 с при середньому значенні 0.8 с. Така варіативність пояснюється станом буферизації треку на стороні SoundCloud: якщо цільова позиція вже знаходиться у буфері плеєра, seekTo виконується майже миттєво; якщо ні – плеєр очікує на завантаження відповідного фрагменту перед відтворенням.

Цей фактор є зовнішнім по відношенню до застосунку і не може бути усунений засобами Widget API.

Загалом результати підтверджують, що підхід синхронізації команд відтворення з компенсацією мережевої затримки є практично застосовним для задач спільного прослуховування аудіоконтенту. Всі виміряні значення затримки є прийнятними і не створюють відчутного дискомфорту при сприйнятті.

4.5.4 Оцінка масштабованості

Окремим аспектом дослідження є оцінка масштабованості архітектури при збільшенні кількості учасників. Оскільки механізм синхронізації базується на broadcast-моделі – сервер надсилає одне повідомлення у канал, а всі підписники отримують його одночасно – кількість учасників сесії не впливає на затримку синхронізації для кожного окремого клієнта.

Горизонтальне масштабування забезпечується через Redis pub/sub: кілька екземплярів Reverb можуть обслуговувати різні WebSocket-з'єднання та обмінюватись повідомленнями через спільний Redis-канал. Це дозволяє масштабувати систему для обслуговування великої кількості одночасних сесій без деградації продуктивності.

Єдиним вузьким місцем при зростанні навантаження є база даних MySQL, яка отримує запити при кожній команді відтворення та голосуванні. Однак завдяки використанню Redis для кешування сесійних даних та черг, кількість прямих звернень до бази даних є мінімальною.

ВИСНОВКИ

У роботі розроблено та досліджено вебзастосунок Aurasync для синхронізованого прослуховування аудіоконтенту з інтерактивним голосуванням. В процесі роботи виконано всі поставлені завдання.

За результатами аналізу існуючих рішень – Spotify Group Session, SyncWave та Watch2Gether – встановлено, що жодне з них не поєднує синхронізоване відтворення з механізмом колективного голосування за наступний трек та гнучкою системою ролей. Це підтвердило доцільність розроблення спеціалізованого застосунку.

Обґрунтовано вибір технологічного стеку: Laravel 12 як серверний фреймворк, Laravel Reverb як WebSocket-сервер, Livewire 3 та Alpine.js для реактивного інтерфейсу, Redis як брокер повідомлень та сховище черг, SoundCloud Widget API як джерело аудіоконтенту. Показано, що поєднання цих технологій забезпечує необхідну функціональність при мінімальній складності інфраструктури.

Спроєктовано архітектуру системи, що включає дев'ять таблиць реляційної бази даних, шість типів WebSocket-подій та чотири Livewire-компоненти. Центральним архітектурним рішенням є розподіл відповідальності між Livewire і Alpine.js: Livewire керує серверним станом, Alpine.js – клієнтською логікою плеєра та обробкою WebSocket-подій.

Реалізовано ключові модулі застосунку: механізм синхронізації відтворення на основі компенсації мережевої затримки, систему голосування з відкладеним завданням черги для визначення переможця, чат реального часу з модерацією та захистом від спаму, а також систему ролей з трьома рівнями доступу.

Проведено функціональне тестування з 18 тест-кейсами, всі пройдено успішно. Перевірено захист від CSRF-атак, несанкціонованого доступу, некоректних вхідних даних та подвійного голосування.

Експериментальне дослідження затримки синхронізації підтвердило практичну застосовність реалізованого підходу. При приєднанні до активної сесії середнє відхилення позиції відтворення становить ±0.28 с, затримка при команді play – 1.06 с, при перемотуванні – 0.80 с у середньому. Отримані результати свідчать про те, що мету роботи досягнуто: реалізований механізм компенсації мережевої затримки на стороні клієнта забезпечує прийнятну точність синхронізації відтворення між учасниками спільної сесії прослуховування.

Серед напрямків подальшого розвитку застосунку можна виділити: підтримку додаткових джерел аудіоконтенту, реалізацію push-сповіщень для запрошення учасників, розширення можливостей модерації, а також проведення вимірювань затримки в умовах реальної мережі з різними значеннями RTT для більш повної оцінки характеристик системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Spotify. Group Session. URL: https://support.spotify.com/us/article/group-session (date of access: 10.05.2025).

2. SyncWave. Synchronized Music Listening Service. URL: https://syncwave.xyz (date of access: 10.05.2025).

3. Watch2Gether. Watch Videos Together. URL: https://w2g.tv (date of access: 10.05.2025).

4. Laravel Documentation. Release Notes. Laravel 12.x. URL: https://laravel.com/docs/12.x/releases (date of access: 10.05.2025).

5. Laravel Reverb. Official Documentation. Laravel 12.x. URL: https://laravel.com/docs/12.x/reverb (date of access: 10.05.2025).

6. Livewire. Official Documentation. Version 3.x. URL: https://livewire.laravel.com/docs (date of access: 10.05.2025).

7. Alpine.js. Official Documentation. Version 3.x. URL: https://alpinejs.dev/start-here (date of access: 10.05.2025).

8. Redis Documentation. Introduction to Redis. URL: https://redis.io/docs/about (date of access: 10.05.2025).

9. Laravel Horizon. Official Documentation. Laravel 12.x. URL: https://laravel.com/docs/12.x/horizon (date of access: 10.05.2025).

10. SoundCloud Widget API. Developer Documentation. URL: https://developers.soundcloud.com/docs/api/html5-widget (date of access: 10.05.2025).

11. MySQL 8.0 Reference Manual. – Oracle Corporation, 2024. URL: https://dev.mysql.com/doc/refman/8.0/en (date of access: 10.05.2025).

12. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 с.

13. Docker Documentation. Docker Compose Overview. URL: https://docs.docker.com/compose (date of access: 10.05.2025).

14. Fette I., Melnikov A. The WebSocket Protocol. RFC 6455. – IETF, 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (date of access: 10.05.2025).

15. Laravel Broadcasting. Official Documentation. Laravel 12.x. URL: https://laravel.com/docs/12.x/broadcasting (date of access: 10.05.2025).

ДОДАТОК А

Технічне завдання

Вступ

Вебзастосунок Aurasync призначений для організації спільних сесій прослуховування аудіоконтенту в реальному часі з можливістю інтерактивного голосування за наступний трек. Система розроблена для використання організаторами сесій та їхніми учасниками, які бажають спільно слухати музику, подкасти або аудіокниги з синхронізованим відтворенням. Область застосування охоплює розважальні та соціальні вебплатформи. Застосунок реалізований у вигляді адаптивного вебінтерфейсу, розрахованого на використання як на десктопних, так і на мобільних пристроях.

А.1 Підстави до розробки

Підставою для розробки є завдання на дипломну роботу бакалавра. Найменування теми розробки – «Розроблення та дослідження програмного забезпечення вебзастосунку для синхронізованого прослуховування аудіопотоку з інтерактивним голосуванням».

А.2 Призначення розробки

Застосунок розроблений для забезпечення синхронізованого спільного прослуховування аудіоконтенту кількома користувачами одночасно. Система реалізує механізм компенсації мережевої затримки на стороні клієнта, що забезпечує мінімальне відхилення позиції відтворення між учасниками. Додатково реалізовано механізм колективного голосування за наступний трек, чат реального часу та систему ролей з трьома рівнями доступу. Експлуатаційне призначення продукту полягає у наданні зручного інструменту для спільного прослуховування аудіоконтенту з можливістю демократичного вибору наступного треку аудиторією.

А.3 Вимоги до програми чи програмного виробу

А.3.1 Вимоги до функціональних характеристик

Вебзастосунок Aurasync розроблений для вирішення задач синхронізованого спільного прослуховування аудіоконтенту. Система забезпечує:

– реєстрацію та автентифікацію користувачів з розмежуванням доступу за трьома ролями: організатор, авторизований учасник та гість;

– створення сесій прослуховування з унікальним восьмисимвольним кодом для запрошення учасників;

– приєднання до сесії за кодом без обов'язкової реєстрації у режимі гостя;

– керування відтворенням організатором через вбудований плеєр SoundCloud;

– синхронізацію позиції відтворення між усіма учасниками з компенсацією мережевої затримки на стороні клієнта;

– автоматичне встановлення актуальної позиції відтворення при приєднанні нового учасника до активної сесії;

– формування пулу треків організатором через посилання SoundCloud з автоматичним отриманням метаданих;

– запуск раундів голосування з вибором варіантів треків та встановленням таймера;

– голосування авторизованих учасників за один трек протягом раунду з оновленням результатів у реальному часі;

– автоматичне визначення переможця після завершення таймера та запуск відтворення переможного треку;

– чат реального часу для авторизованих учасників з обмеженням частоти надсилання повідомлень;

– модерацію чату організатором: видалення повідомлень та блокування учасників.

Вхідними даними є текстові запити користувачів, посилання на треки SoundCloud та команди керування відтворенням. Вихідними результатами є синхронізований аудіопотік, результати голосування та повідомлення чату в реальному часі.

А.3.2 Вимоги до надійності

Застосунок Aurasync є клієнт-серверним вебзастосунком, тому для його роботи необхідне стабільне підключення до мережі Інтернет. Надійність системи забезпечується такими механізмами: автоматичне відновлення синхронізації після розриву WebSocket-з'єднання та повторного підключення учасника; захист від подвійного голосування на рівні унікального індексу бази даних та на рівні застосунку; захист від CSRF-атак через токени у всіх формах та AJAX-запитах; валідація всіх вхідних даних на стороні сервера незалежно від клієнтської перевірки; обробка помилок при взаємодії із зовнішнім SoundCloud API з відображенням інформативних повідомлень користувачу.

А.3.3 Умови експлуатації

Застосунок є веборієнтованим, тому ключовою вимогою є наявність сучасного браузера з підтримкою WebSocket та JavaScript. Рекомендовані браузери: Google Chrome версії 90 або новіше, Mozilla Firefox версії 88 або новіше, Safari версії 14 або новіше. Необхідне підключення до мережі Інтернет зі швидкістю не менше 1 Мбіт/с для коректної роботи аудіоплеєра та WebSocket-з'єднання. Рівень підготовки користувача передбачає базові навички роботи з вебінтерфейсами. Система розроблена з адаптивним інтерфейсом, що забезпечує коректне відображення на екранах з роздільною здатністю від 360×640 пікселів.

А.3.4 Вимоги до складу і параметрів технічних засобів

Для розгортання середовища розробки пристрій повинен задовольняти наступні мінімальні технічні вимоги:

– операційна система: Ubuntu 22.04 або новіше, macOS 12 або новіше, Windows 10/11 з підтримкою Docker;

– процесор з тактовою частотою від 2 ГГц та підтримкою апаратної віртуалізації;

– оперативна пам'ять: не менше 8 ГБ для стабільної роботи Docker-контейнерів;

– вільний простір на диску: не менше 10 ГБ для зберігання Docker-образів та даних;

– мережеве з'єднання: обов'язкове підключення до мережі Інтернет.

Для кінцевого користувача спеціальних вимог до апаратного забезпечення не висувається – достатньо будь-якого пристрою з сучасним браузером та доступом до Інтернету.

А.3.5 Вимоги до інформаційної та програмної сумісності

Обмін даними між клієнтом і сервером здійснюється у форматах JSON та через бінарний WebSocket-протокол. Програмна реалізація базується на такому стеку технологій:

– серверна частина: PHP 8.3, Laravel 12;

– WebSocket-сервер: Laravel Reverb;

– менеджер черг: Laravel Horizon;

– клієнтська частина: Livewire 3, Alpine.js, Tailwind CSS;

– база даних: MySQL 8.0;

– брокер повідомлень та черг: Redis 7.2;

– джерело аудіоконтенту: SoundCloud Widget API;

– середовище розгортання: Docker, Docker Compose, Nginx.

Застосунок сумісний з усіма сучасними браузерами, що підтримують стандарти WebSocket та ES6+. Розгортання здійснюється через Docker Compose без залежності від конкретної операційної системи хост-машини.

А.3.6 Вимоги до маркування та пакування

Дистрибуція застосунку Aurasync здійснюється через мережу Інтернет за допомогою унікальної вебадреси, що забезпечує миттєвий доступ без необхідності локального встановлення. Вихідний код розповсюджується через систему контролю версій Git. Розгортання здійснюється через Docker Compose, що забезпечує відтворюване середовище виконання на будь-якій платформі.

Оскільки застосунок є програмним продуктом, будь-які вимоги до фізичного пакування, матеріальних носіїв або логістичного супроводу не висуваються. Оновлення системи виконуються через оновлення вихідного коду та перезапуск Docker-контейнерів.

А.3.7 Вимоги до транспортування та зберігання

Враховуючи цифрову природу застосунку, будь-які вимоги щодо фізичного транспортування або спеціальних умов зберігання матеріальних компонентів є неактуальними. Вихідний код зберігається у репозиторії системи контролю версій Git. Розгортання нових версій виконується через оновлення репозиторію та перезбірку Docker-образів командою docker compose build. Резервне копіювання даних забезпечується засобами MySQL та Docker volumes.

А.4 Вимоги до програмної документації

Комплект документації до застосунку Aurasync включає це технічне завдання та пояснювальну записку. Пояснювальна записка детально описує архітектуру системи, схему бази даних, механізм синхронізації відтворення та алгоритм компенсації мережевої затримки, реалізацію ключових модулів, результати функціонального тестування та експериментального дослідження затримки синхронізації. Окрема увага приділена опису взаємодії між Livewire-компонентами та Alpine.js у контексті роботи з WebSocket-подіями.

А.5 Техніко-економічні показники

Розроблений застосунок має практичне значення для організації спільного дозвілля та соціальної взаємодії через музику. Система реалізована виключно на основі безкоштовного програмного забезпечення з відкритим вихідним кодом, що виключає ліцензійні витрати. Розгортання через Docker Compose дозволяє запустити повноцінне середовище на власному сервері без залежності від хмарних сервісів.

Реалізований механізм голосування за наступний трек підвищує залученість учасників порівняно з пасивним прослуховуванням, перетворюючи індивідуальне споживання контенту на колективний досвід. Архітектура системи на базі Redis забезпечує можливість горизонтального масштабування без суттєвих змін у кодовій базі.

А.6 Стадії та етапи розробки

Створення застосунку розподілено на наступні стадії:

– аналіз предметної області та огляд існуючих рішень, визначення функціональних і нефункціональних вимог (1 тиждень);

– проєктування архітектури системи, схеми бази даних та механізмів взаємодії компонентів (2 тижні);

– налаштування середовища розробки через Docker Compose та розгортання базової інфраструктури (3 дні);

– реалізація серверної частини: моделі, контролери, система подій та завдання черги (2 тижні);

– реалізація клієнтської частини: Livewire-компоненти, Alpine.js-логіка плеєра та синхронізація (1 тиждень);

– функціональне тестування та дослідження характеристик затримки синхронізації (3 дні);

– написання пояснювальної записки та підготовка презентації 1 тиждень).

А.7 Порядок контролю та приймання

Підтвердження функціональності застосунку Aurasync передбачає багаторівневий цикл випробувань. Функціональне тестування методом чорної скриньки охоплює перевірку всіх визначених функціональних вимог: реєстрацію та автентифікацію, створення та приєднання до сесій, керування відтворенням, систему голосування та чат. Для кожного тест-кейсу фіксується очікуваний та фактичний результат.

Тестування безпеки включає перевірку CSRF-захисту, коректності авторизації за ролями, валідації вхідних даних та захисту від подвійного голосування. Експериментальне дослідження затримки синхронізації проводиться шляхом вимірювання відхилення позиції відтворення між учасниками для трьох сценаріїв: приєднання до активної сесії, команда play та перемотування. Після завершення всіх етапів перевірки та підтвердження відповідності визначеним вимогам система вважається готовою до введення в експлуатацію.

ДОДАТОК Б

Текст програми

Б.1 Функція syncPlayback – алгоритм компенсації мережевої затримки

```
syncPlayback(data) {
    if (!this.playerReady) return;
    const player = this.getPlayer();
    if (!player) return;
    const serverTime = new Date(data.server_time).getTime();
    const offset = (Date.now() - serverTime) / 1000;
    const position = data.playback_position + offset;
    if (data.is_playing) {
        player.seekTo(position * 1000);
        setTimeout(() => {
            player.seekTo(position * 1000);
            setTimeout(() => {
                player.play();
            }, 300);
        }, 300);
    } else {
        player.seekTo(data.playback_position * 1000);
        setTimeout(() => {
            player.seekTo(data.playback_position * 1000);
            player.pause();
        }, 300);
    }
},
```

Б.2 Клас FinalizeVotingRound – завдання черги для визначення переможця голосування

```
<?php

namespace App\Jobs;

use App\Events\PlaybackUpdated;
use App\Events\TrackChanged;
use App\Events\VotingEnded;
use App\Models\VotingRound;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Queue\Queueable;

class FinalizeVotingRound implements ShouldQueue
{
    use Queueable;

    public function __construct(
        public readonly int $votingRoundId
    ) {}

    public function handle(): void
    {
        $round = VotingRound::with([
            'options.votes',
            'options.track',
```

```
            'listeningSession'
        ])->findOrFail($this->votingRoundId);

        if ($round->status !== 'active') return;

        $options = $round->options->map(fn($option) => [
            'option' => $option,
            'votes'  => $option->votes->count(),
        ])->sortByDesc('votes');

        $maxVotes = $options->first()['votes'];

        $winners = $options
            ->filter(fn($o) => $o['votes'] === $maxVotes)
            ->values();

        $winner     = $winners->random();
        $winnerTrack = $winner['option']->track;

        $round->update([
            'status'          => 'ended',
            'winner_track_id' => $winnerTrack->id,
        ]);

        $session = $round->listeningSession;
        $session->update([
            'current_track_id'    => $winnerTrack->id,
            'is_playing'          => true,
            'playback_position'   => 0,
            'playback_started_at' => now(),
        ]);

        $session->refresh();

        broadcast(new VotingEnded($round));
        broadcast(new TrackChanged($session));
        broadcast(new PlaybackUpdated($session));
    }
}
```

Б.3 Метод state контролера – динамічний розрахунок позиції відтворення

```
public function state(string $code)
{
    $session = ListeningSession::where('code', $code)
        ->firstOrFail();

    $position = $session->playback_position;

    if ($session->is_playing && $session->playback_started_at) {
        $elapsed = now()->diffInSeconds(
            $session->playback_started_at,
            false
```

```
        );
        if ($elapsed > 0) {
            $position = $session->playback_position + $elapsed;
        }
    }

    return response()->json([
        'is_playing'          => $session->is_playing,
        'playback_position'   => $position,
        'playback_started_at' => $session->playback_started_at
                                    ?->toISOString(),
        'server_time'         => now()->toISOString(),
        'current_track'       => $session->currentTrack ? [
            'id'            => $session->currentTrack->id,
            'title'         => $session->currentTrack->title,
            'soundcloud_url'=> $session->currentTrack->soundcloud_url,
        ] : null,
    ]);
}
```

Б.4 Клас події PlaybackUpdated

```
<?php

namespace App\Events;

use App\Models\ListeningSession;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class PlaybackUpdated implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public function __construct(
        public readonly ListeningSession $session
    ) {}

    public function broadcastOn(): Channel
    {
        return new Channel('session.' . $this->session->code);
    }

    public function broadcastAs(): string
    {
        return 'playback.updated';
    }

    public function broadcastWith(): array
    {
        return [
            'is_playing'          => $this->session->is_playing,
            'playback_position'   => $this->session->playback_position,
            'playback_started_at' => $this->session
```

```
                                        ->playback_started_at
                                        ?->toISOString(),
            'server_time'         => now()->toISOString(),
        ];
    }
}
```

Б.5 Модель ListeningSession

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

class ListeningSession extends Model
{
    use HasFactory;

    protected $fillable = [
        'host_user_id',
        'name',
        'code',
        'status',
        'is_playing',
        'playback_position',
        'playback_started_at',
        'current_track_id',
    ];

    protected function casts(): array
    {
        return [
            'is_playing'          => 'boolean',
            'playback_position'   => 'float',
            'playback_started_at' => 'datetime',
        ];
    }

    public function host(): BelongsTo
    {
        return $this->belongsTo(User::class, 'host_user_id');
    }

    public function currentTrack(): BelongsTo
    {
        return $this->belongsTo(Track::class, 'current_track_id');
    }

    public function tracks(): HasMany
    {
        return $this->hasMany(Track::class);
    }
```

```
    public function votingRounds(): HasMany
    {
        return $this->hasMany(VotingRound::class);
    }

    public function chatMessages(): HasMany
    {
        return $this->hasMany(ChatMessage::class);
    }

    public function bans(): HasMany
    {
        return $this->hasMany(SessionBan::class);
    }

    public function activeVotingRound()
    {
        return $this->votingRounds()
            ->where('status', 'active')
            ->latest()
            ->first();
    }

    public function isBanned(int $userId): bool
    {
        return $this->bans()
            ->where('banned_user_id', $userId)
            ->exists();
    }
}
```

Б.6 Метод startVoting компонента HostPanel

```
public function startVoting(): void
{
    $this->validate([
        'selectedTrackIds' => ['required', 'array', 'min:2', 'max:5'],
        'votingDuration'       =>  ['required',  'integer',  'min:10',
'max:120'],
    ]);

    abort_if($this->session->activeVotingRound() !== null, 403);

    $tracks = $this->session->tracks()
        ->whereIn('id', $this->selectedTrackIds)
        ->get();

    abort_if(
        $tracks->count() !== count($this->selectedTrackIds),
        422
    );

    $round = $this->session->votingRounds()->create([
        'status'  => 'active',
        'ends_at' => now()->addSeconds($this->votingDuration),
```

```
        ]);

        foreach ($tracks as $track) {
            $round->options()->create(['track_id' => $track->id]);
        }

        broadcast(new VotingStarted($round->load('options.track')));

        FinalizeVotingRound::dispatch($round->id)
            ->delay($round->ends_at);

        $this->reset(['selectedTrackIds', 'votingDuration']);
        $this->session->refresh();
    }
```

Б.7 Методи sendMessage, deleteMessage та banUser компонента ChatPanel

```
    public function sendMessage(): void
    {
        if (!auth()->check()) return;
        if ($this->isBanned) return;

        $key         = 'chat-message:' . auth()->id() . ':' . $this->session-
    >id;
        $cooldownKey  =  'chat-cooldown:'  .  auth()->id()  .  ':'  .  $this-
    >session->id;

        if (RateLimiter::tooManyAttempts($cooldownKey, 1)) {
            $this->addError(
                'message',
                'Зачекайте секунду перед наступним повідомленням.'
            );
            return;
        }

        if (RateLimiter::tooManyAttempts($key, 5)) {
            $this->addError(
                'message',
                'Ви надсилаєте повідомлення занадто швидко.'
            );
            return;
        }

        $this->validate([
            'message' => ['required', 'string', 'max:300'],
        ]);

        RateLimiter::hit($cooldownKey, 1);
        RateLimiter::hit($key, 10);

        $chatMessage = $this->session->chatMessages()->create([
            'user_id' => auth()->id(),
            'message' => $this->message,
        ]);
```

```
        $chatMessage->load('user');
        broadcast(new MessageSent($chatMessage));
        $this->reset('message');
        $this->dispatch('scroll-chat');
    }

    public function deleteMessage(int $messageId): void
    {
        if (!$this->isHost) return;

        ChatMessage::where('id', $messageId)
            ->where('listening_session_id', $this->session->id)
            ->delete();

        $this->messages = array_values(array_filter(
            $this->messages,
            fn($m) => $m['id'] !== $messageId
        ));

        broadcast(new MessageDeleted($messageId, $this->session->id));
    }

    public function banUser(int $userId): void
    {
        if (!$this->isHost) return;
        if ($userId === auth()->id()) return;

        SessionBan::firstOrCreate([
            'listening_session_id' => $this->session->id,
            'banned_user_id'       => $userId,
        ]);

        broadcast(new UserBanUpdated($this->session, $userId, true));
    }
```

Б.8 Методи join та hostPlay компонента sessionPlayer

```
async join() {
    this.joined = true;
    this.watchIframe();

    const response = await fetch(
        `/sessions/${this.sessionCode}/state`
    );
    const data = await response.json();

    this.isPlaying          = data.is_playing;
    this.playbackPosition   = data.playback_position;
    this.playbackStartedAt  = data.playback_started_at;

    if (data.is_playing) {
        const waitForPlayer = setInterval(() => {
            if (this.playerReady) {
                clearInterval(waitForPlayer);
                this.syncPlayback(data);
```

```
            }
        }, 200);
    }
},

hostPlay() {
    if (!this.playerReady) return;
    const player = this.getPlayer();
    if (!player) return;

    player.seekTo(this.playbackPosition * 1000);
    setTimeout(() => {
        player.seekTo(this.playbackPosition * 1000);
        setTimeout(() => {
            player.play();
            this.isPlaying         = true;
            this.playbackStartedAt = new Date().toISOString();

            fetch(`/sessions/${this.sessionCode}/playback`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'X-CSRF-TOKEN': document.querySelector(
                        'meta[name="csrf-token"]'
                    ).content,
                },
                body: JSON.stringify({
                    action:   'play',
                    position: this.playbackPosition,
                }),
            }).then(r => r.json()).then(data => {
                this.playbackStartedAt = data.playback_started_at;
            });
        }, 300);
    }, 300);
},
```

ДОДАТОК В

Слайди презентації

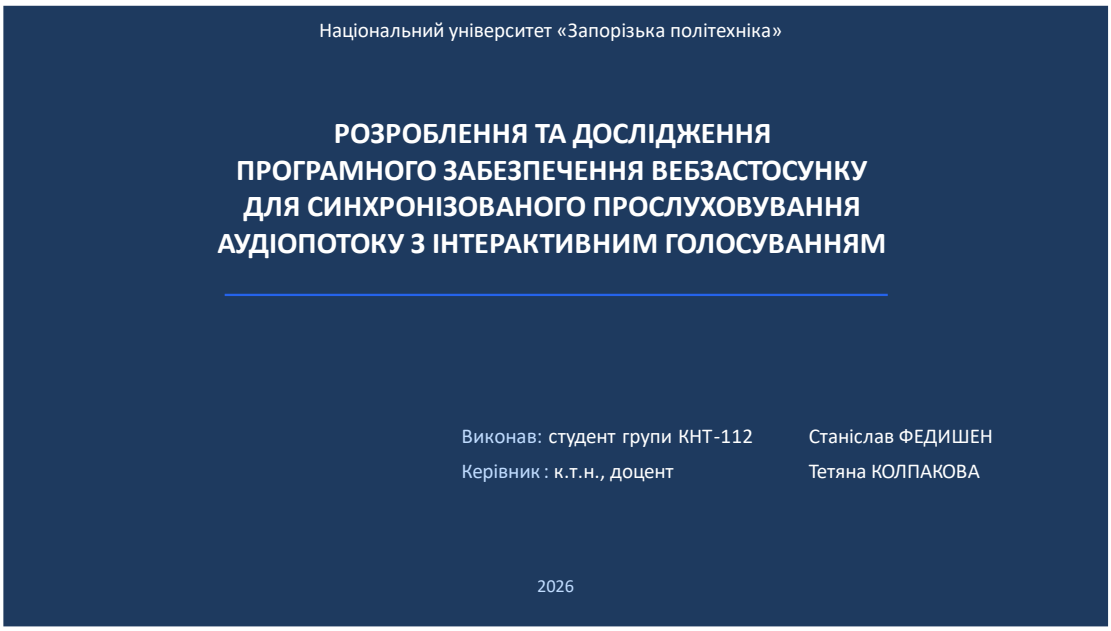


Рисунок В.1 – Слайд №1

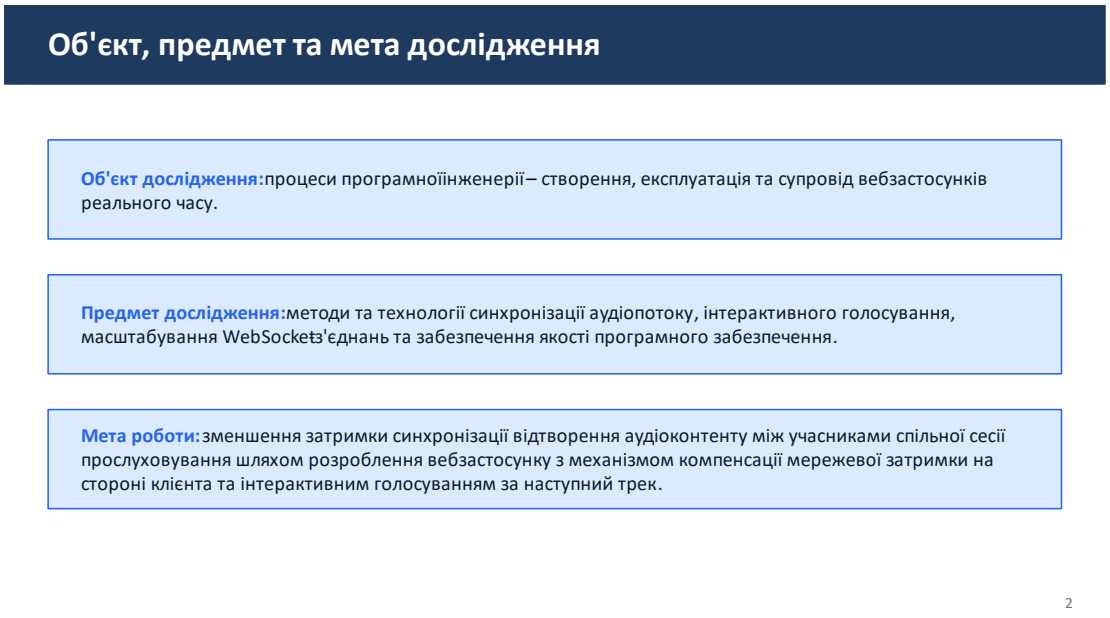


Рисунок В.2 – Слайд №2

Завдання дослідження

1. Проаналізувати існуючі рішення для спільного прослуховування медіаконтенту та визначити їх переваги й недоліки
2. Обґрунтувати вибір технологічного стеку та архітектурних рішень для реалізації застосунку
3. Спроєктувати архітектуру системи, схему бази даних та механізми взаємодії компонентів
4. Реалізувати ключові модулі застосунку: синхронізацію аудіопотоку черезWebSocket, систему голосування та чат реального часу
5. Провести тестування та дослідження характеристик затримки синхронізації

3

Рисунок В.3 – Слайд №3

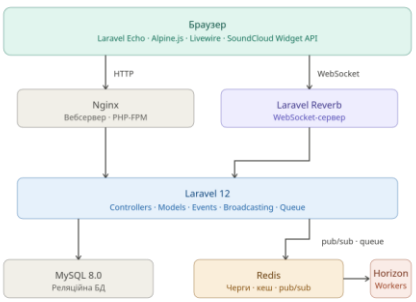
Аналіз існуючих рішень

Критерій	Spotify Group Session	SyncWave	Watch2Gether	Aurasync
Синхронізація відтворення	+	+	+	+
Голосування за трек	–	–	–	+
Чат у реальному часі	–	+	+	+
Безкоштовний доступ	–	+	+	+
Без реєстрації (гість)	–	+	+	+
Система ролей	–	–	–	+
Модерація чату	–	–	частково	+
Надійність інтерфейсу	+	низька	+	+

4

Рисунок В.4 – Слайд №4

Архітектура системи



5

Рисунок В.5 – Слайд №5

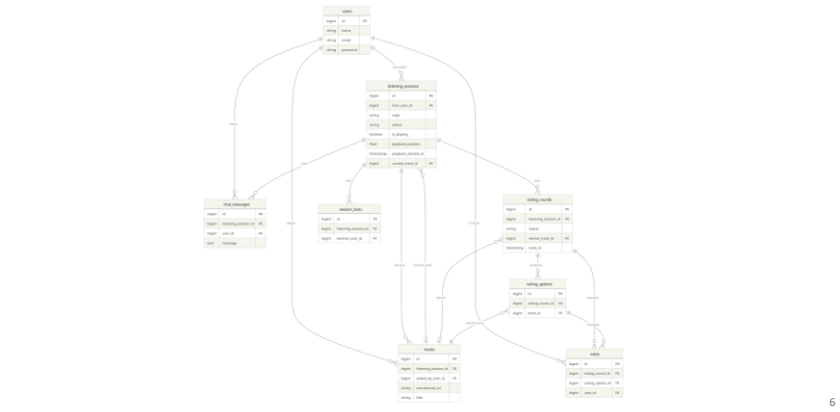
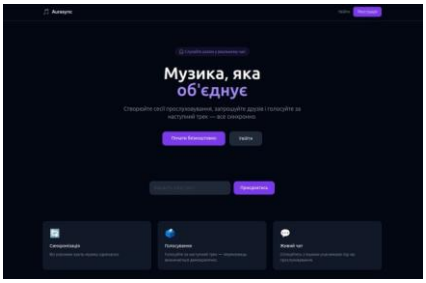


Рисунок В.6 – Слайд №6

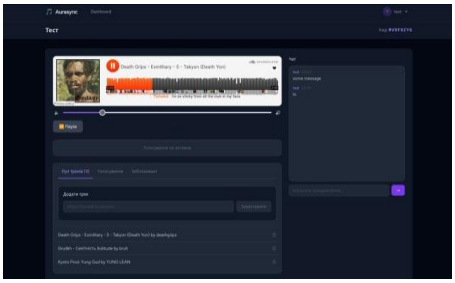
Рисунок В.7 – Слайд №7

Рисунок В.8 – Слайд №8

Інтерфейс застосунку (1/2)



Головна сторінка

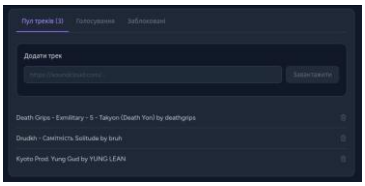


Сторінка сесії

9

Рисунок В.9 – Слайд №9

Інтерфейс застосунку (2/2)



Панель організатора

Панель голосування

10

Рисунок В.10 – Слайд №10

Результати функціонального тестування

18	18	0
тест-кейсів виконано	успішно пройдено	виявлених помилок

Ключові тест-кейси	Результат
Створення сесії та приєднання учасника за кодом	Пройдено
Синхронізація відтворення між учасниками	Пройдено
Запуск голосування та автоматичне визначення переможця	Пройдено
Захист від подвійного голосування	Пройдено
Надсилання повідомлень у чат та rate limiting	Пройдено
Блокування учасника та видалення повідомлень організатором	Пройдено
Захист від CSRF та несанкціонованого доступу	Пройдено
Автоматичний запуск відтворення після завершення голосування	Пройдено

11

Рисунок В.11 – Слайд №11

Результати дослідження затримки синхронізації

Умови: localhost, 5 вимірювань на кожен сценарій

Таблиця 1 – Відхилення позиції при приєднанні до активної сесії

Вимірювання	1	2	3	4	5	Середнє
Відхилення, с	+0.3	−0.3	+0.1	+0.5	−0.2	±0.28

Таблиця 2 – Затримка команди play

Вимірювання	1	2	3	4	5	Середнє
Затримка, с	0.9	1.1	1.3	0.8	1.2	1.06

Таблиця 3 – Затримка перемотування

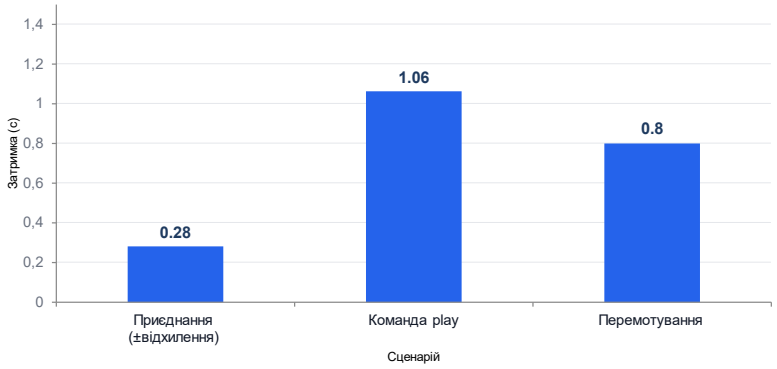
Вимірювання	1	2	3	4	5	Середнє
Затримка, с	0.2	1.8	0.4	1.5	0.1	0.8

Всі виміряні значення є прийнятними для задач спільного прослуховування аудіоконтенту

12

Рисунок В.12 – Слайд №12

Графік затримки синхронізації по сценаріях



13

Рисунок В.13 – Слайд №13

Висновки

1. За результатами аналізу аналогів встановлено, що жодне з існуючих рішень не поєднує синхронізоване прослуховування з механізмом колективного голосування за наступний трек – що обґрунтувало доцільність розроблення застосунку.
2. Розроблено та реалізовано алгоритм компенсації мережевої затримки на стороні клієнта: при отриманні WebSocket події клієнт розраховує зміщення між серверним та локальним часом і коригує позицію відтворення, що забезпечує синхронізацію між учасниками.
3. Проведено функціональне тестування: усі 18 тест-кейсів пройдено успішно.
4. Мету роботи досягнуто: реалізований механізм забезпечує прийнятну точність синхронізації – середнє відхилення при приєднанні ±0.28 с, затримка при команді play – 1.06 с, при перемотуванні – 0.8 с.

14

Рисунок В.14 – Слайд №14