Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

РОЗРОБКА ВЕБПЛАТФОРМИ ТАНЦЮВАЛЬНИХ ПОСЛУГ ДЛЯ КОРИСТУВАЧІВ З РІЗНИМИ МОЖЛИВОСТЯМИ З УРАХУВАННЯМ БАЗОВИХ КРИТЕРІЇВ ДОСТУПНОСТІ ЗА СТАНДАРТОМ WCAG 2.2

Текстова частина до дипломної роботи

за спеціальністю «Інженерія програмного забезпечення» 121

Керівник дипломної роботи

Ст. викладач Вознюк О.М.

_____

(*підпис*)

«__» _____ 2025 р.

Виконала студентка БП ІПЗ-4

Мацевко Є.Д.

«__» _____ 2025 р.

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

доцент., к.ф-м.н. Гороховський С.С.

_____

(*підпис*)

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студенту 4-го курсу, факультету інформатики

Мацевко Євгенії Дмитрівній

ТЕМА: Розробка вебплатформи танцювальних послуг для користувачів з різними можливостями з урахуванням базових критеріїв доступності за стандартом WCAG 2.2.

Зміст ТЧ до дипломної роботи:

Зміст

Анотація

Вступ

Аналіз предметної області та потреб користувачів

Проектування вебплатформи

Робота з базою даних

Огляд серверної частини вебплатформи

Огляд клієнтської частини вебплатформи

Висновки

Список використаних джерел

Дата видачі „___” _____ 2024 р. Керівник _____*(підпис)*

Завдання отримав _____*(підпис)*

Тема: Розробка вебплатформи танцювальних послуг для користувачів з різними можливостями з урахуванням базових критеріїв доступності за стандартом WCAG 2.2

Календарний план виконання роботи:

№ п/п	Назва етапу	Термін виконання	Примітка
1.	Отримання теми дипломної роботи.	15.09.2024	
2.	Ознайомлення з предметною областю.	10.10.2024	
3.	Огляд і аналіз аналогів.	20.10.2024	
4.	Окреслення основних функціональних і нефункціональних вимог до застосунку.	11.11.2024	
5.	Пошук, аналіз та вибір технологій розробки.	02.12.2024	
6.	Початок створення практичної частини.	07.01.2025	
7.	Початок написання теоретичної частини.	10.04.2025	
8.	Створення презентації.	18.05.2025	
9.	Завершення практичної та теоретичної частин.	24.05.2025	
10.	Подання роботи на кафедру для перевірки на плагіат.	28.05.2025	

Студент Мацевко Є.Д.

Керівник Вознюк О.М.

“_____”_____

Зміст

АНОТАЦІЯ

Дипломна робота присвячена розробці вебплатформи танцювальних послуг доступної для різних категорій користувачів, зокрема взято до уваги потреби людей з інвалідністю, різними фінансовими можливостями, новачків, викладачів та танцювальних шкіл.

Основну увагу приділено створенню зручного та доступного інтерфейсу, реалізації функціональності пошуку занять та організації приватних уроків. Серверна частина застосунку реалізована на Java з використанням Spring Boot, клієнтська – з використанням React. Для збереження даних використано PostgreSQL, а для зберігання файлів – MinIO. Контейнеризація здійснена за допомогою Docker. Забезпечено доступність інтерфейсу згідно з рекомендаціями WCAG 2.2.

Результатом є сучасний, функціональний та інклюзивний застосунок, що забезпечує зручну взаємодію для широкого кола користувачів та відповідає сучасним стандартам. У роботі надано опис його інтерфейсу та функціональних можливостей.

Ключові слова: танці, вебплатформа, доступність, Spring Boot, React.

ВСТУП

Протягом десятиліть ведуться дискусії та боротьба за забезпечення рівних прав доступу до навчання, ринку праці, дозвілля тощо для різних груп населення. В той час як надання фізичної доступності вимагає великої кількості зусиль та часу, проблема цифрової доступності надання інформації із розвитком сучасних технологій стає все простішою для вирішення. Розробнику пропонується широкий діапазон інструментів як для впровадження інклюзивності, так і для її тестування. У зв'язку з цим достатній рівень доступності цифрового продукту став однією з важливих вимог до нього, що регулюється міжнародним стандартом WCAG [1].

Проте, в той час як широковикористовувані цифрові продукти відзначаються високим рівнем інклюзивності, більш вузькоспеціалізовані застосунки, на жаль, часто не забезпечують її на належному рівні.

У даній дипломній роботі буде досліджуватись сфера танцювальних послуг та потенціал покращення доступу до неї через впровадження нової вебплатформи, адаптованої для людей з різними можливостями. При цьому, під доступністю мається на увазі не лише врахування потреб людей з інвалідністю, але й забезпечення зручності для новачків, користувачів з різним рівнем цифрової грамотності та фінансових можливостей. Галузь танцювальних послуг вже опрацьовувалась у межах попередніх проєктів, що дозволяє спиратись на наявні напрацювання та результати досліджень для подальшого поглиблення й розширення підходів до створення спеціалізованого цифрового продукту для цієї сфери.

Актуальність та **прикладна цінність** даного проєкту є беззаперечними, адже, за статистикою Української федерації танцю, аж 20% українців займаються даним видом спорту [2]. Це свідчить про популярність сфери та важливість забезпечення широкофункціональних застосунків для її обслуговування. Крім того, з початком повномасштабного вторгнення в Україні

значно зросла кількість людей з інвалідністю, що спричинило активний розвиток інклюзивних ініціатив щодо покращення як фізичної, так і цифрової доступності. Зокрема, 83% українців сприймають безбар'єрність як нову суспільну цінність [3]. Враховуючи це, створення платформи танцювальних послуг, доступної для всіх, є важливим кроком до зміцнення соціальної інтеграції та підвищення якості життя населення.

Метою роботи є розробка вебплатформи танцювальних послуг для людей із різними можливостями, яка враховує базові критерії цифрової доступності згідно зі стандартом WCAG 2.2, забезпечуючи зручний інтерфейс, широкий функціонал і ефективні засоби пошуку для всіх груп користувачів.

Об'єктом дослідження є процес надання танцювальних послуг за допомогою вебплатформ для людей з різними можливостями.

Предметом дослідження є створення вебплатформи танцювальних послуг із функціоналом, який забезпечує доступність відповідно до базових вимог стандарту WCAG 2.2 для різних груп користувачів.

Наукова новизна роботи полягає у створенні інноваційної вебплатформи, яка поєднує інклюзивний дизайн із широким набором функцій для організації танцювальних заходів. Платформа враховує специфічні потреби користувачів із різними можливостями, забезпечуючи зручність пошуку, реєстрації, комунікації та адаптації контенту.

Робота складається з 5 основних розділів.

Перший розділ присвячений аналізу предметної області, потреб користувачів та існуючих рішень, визначено вимоги до кінцевого застосунку.

Другий розділ містить характеристику архітектури системи та обгрунтування вибору засобів розробки.

Третій розділ описує роботу з базою даних, зокрема її структуру та підходи до формування запитів до неї.

Четвертий розділ надає інформацію про особливості реалізації серверної частини вебплатформи.

У п’ятому розділі розглянуто особливості розробки застосунку на стороні клієнта та забезпечення його доступності, а також надано опис реалізованого користувацького інтерфейсу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОТРЕБ КОРИСТУВАЧІВ

1.1 Проблема доступності танцювального середовища

Зміцнення м’язів, покращення координації та витривалості, розвиток пам’яті і підвищення самооцінки – усе це лише частина позитивного впливу танців. Вони також сприяють соціалізації, дарують позитивні емоції та загалом покращують, попри численні їхні переваги, доступ до цієї сфери залишається психоемоційний стан. Проте обмеженим для частини населення і проблема, на жаль, часто полягає не у відсутності відповідних ініціатив, а в нестачі інформації про них.

Для ілюстрації цієї проблеми доцільно розглянути кілька типових життєвих ситуацій.

- Молодий студент із захопленням переглядає чергове відео з танцями в інстаграмі. Його вражає ритмічність музики та контроль виконавців над тілом. Будучи відвертим, він вже давно мріяв спробувати себе в даному виді спорту, проте перехід від захоплення до конкретних дій здається складним. Необхідність самостійно шукати школу та орієнтуватися серед безлічі напрямів викликає розгубленість. Попри сильне бажання спробувати, він не знає, з чого почати.
- Жінка, яка все життя танцювала, після ДТП опинилася в інвалідному візку. Її танцювальна школа, на жаль, не адаптована для людей з інвалідністю. Звичайно в інтернеті можна знайти безліч інших шкіл, але невже доведеться обдзвонити їх всі, аби отримати необхідну інформацію про доступність?
- Чоловік із порушенням зору дізнається про новий цікавий танцювальний проєкт. Йому теж хотілось би спробувати, але ще на етапі пошуку виникають складнощі – сайт не читається скрінрідером,

частина кнопок недоступна, немає описів до відео. Через це навіть дізнатися подробиці або записатись на заняття стає майже неможливо.

На перший погляд, ці ситуації видаються зовсім різними. Проте, насправді, всі вони мають спільну рису: бажання людей займатись танцями та одночасну відсутність доступу до необхідної інформації про танцювальні можливості.

Таким чином, виникає потреба у розробці доступної вебплатформи, яка б надавала увесь необхідний функціонал. При цьому, важливо врахувати різні групи населення та надати користувацький інтерфейс адаптований для людей з різними можливостями. Таким чином, користувачам, навіть із досить специфічним запитом або потребами, більше не потрібно буде відвідувати десятки інших сайтів та обдзвонювати танцювальні школи. Уся необхідна інформація знаходиться в одному місці із реалізованим функціоналом зручного пошуку, фільтрації та сортування.

1.2 Потреби користувачів

З метою глибшого розуміння потреб цільової аудиторії, доцільно сегментувати потенційних користувачів платформи та проаналізувати специфіку запитів кожної групи.

Для цього виокремимо основні категорії користувачів, які можуть бути зацікавлені в доступній танцювальній платформі, з огляду на їхній досвід, фізичні можливості, соціальний статус та інші фактори, та надамо коротку характеристику запиту кожного з них:

1. Професійні танцюристи та викладачі

Потреби:

- Створення профілю викладача та поширення танцювального контенту.
- Організація подій та курсів.
- Можливість надавати приватні уроки.

2. Танцювальні школи

Потреби:

- Створення профілю школи та поширення танцювального контенту.
- Організація подій та курсів, публікація розкладу занять.
- Додавання інформації про кілька відділень однієї школи.

3. Люди з інвалідністю

Потреби:

- Інформація про фізичну доступність шкіл, подій та курсів.
- Пошук викладачів із досвідом інклюзивної роботи.
- Наявність онлайн формату занять.
- Пошук шкіл, подій та курсів з необхідною опцією доступності
- Візуальна/аудіо/текстова адаптація інтерфейсу платформи.

4. Новачки

Потреби:

- Надання персоналізованих рекомендацій щодо вибору танцювального напряму.
- Фільтрація за рівнем складності.

5. Люди з обмеженими фінансовими можливостями та з віковими особливостями

Потреби:

- Пошук, фільтрація та сортування подій, шкіл, курсів та профілів викладачів за необхідним запитом

Крім того, для усіх груп є потреба в наявності на платформі системи відгуків і рейтингів для спрощення вибору танцювальних ініціатив, можливості реєстрації на події, курси та приватні уроки, та зручного подання інформації щодо власного розкладу у вигляді календаря.

Отже, було виділено основні потреби цільової аудиторії, які визначають ключові функціональні та сервісні вимоги до платформи, спрямовані на забезпечення доступності та комфортного користування для всіх зацікавлених груп.

1.3 Огляд існуючих рішень

Завдяки популярності танців та активності їхньої спільноти, дана галузь постійно розвивається, тому вже сьогодні можна знайти чимало як і вузькоспеціалізованих, так і досить шикорофункціональних вебсайтів, що обслуговують дану сферу. У межах дослідження, було відвідано та протестовано чималу кількість вебзастосунків і у підсумку визначено два із найширшим функціоналом послуг. Давайте їх детально розглянемо та проаналізуємо переваги та недоліки в контексті обслуговування сфери танцювальних послуг для людей з різними можливостями.

Розпочнемо із вебсайту «The Dance Platform» [4] (рис. 1.1).

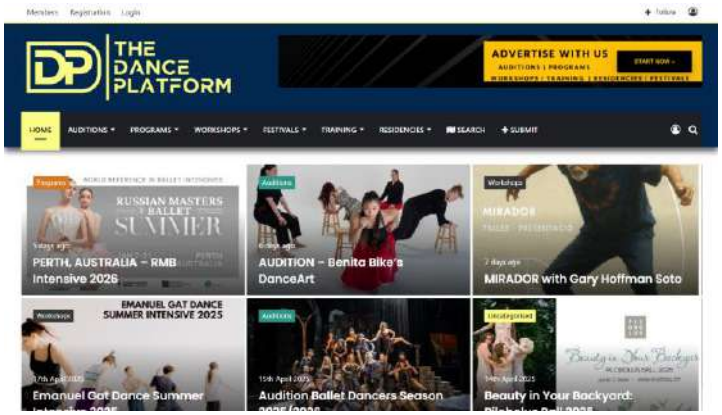


Рисунок 1.1 Головна сторінка вебсервісу «The Dance Platform»

Платформа спеціалізується у наданні інформації про різноманітні танцювальні можливості, що робить її зручною для пошуку та організації заходів у цій сфері. Серед основних переваг варто виділити зручне групування подій за типом та можливість пошуку за ключовими словами, що значно спрощує навігацію. Крім того, реалізований функціонал створення власних подій, що підвищує залученість користувачів. Платформа також пропонує актуальну інформацію про вакансії у танцювальній сфері та відображає події на карті з розширеними фільтрами для точного пошуку (рис 1.2).

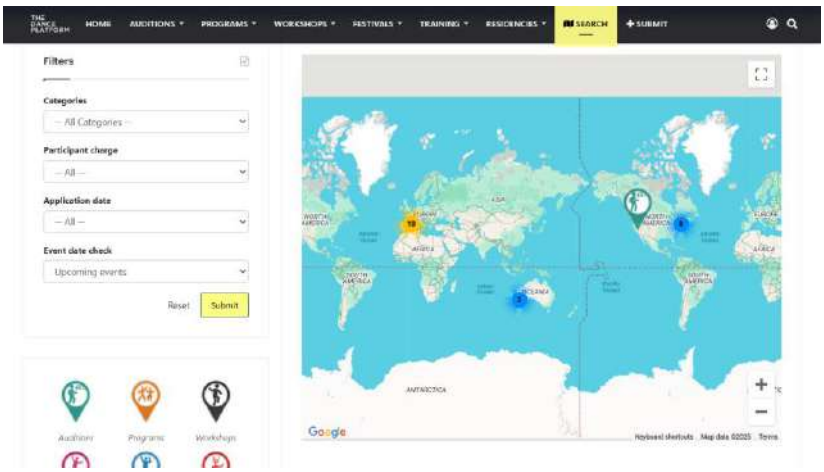


Рисунок 1.2 Пошук подій на карті

Водночас, платформа має і низку недоліків, які потребують уваги. Перш за все, дизайн ресурсу місцями виглядає перевантаженим контентом, що

суперечить принципам інклюзивного дизайну. Також відсутня інформація про розклад та курси танцювальних шкіл, а також немає функціоналу для організації приватних уроків, що обмежує можливості викладачів. Крім того, платформа не повністю адаптована для використання технологій екранного зчитування, що створює перешкоди для людей з вадами зору. Важливо також зазначити відсутність спеціалізованої інформації та пошукових фільтрів, орієнтованих на людей з інвалідністю. Можливість подавати події без обов'язкової реєстрації робить платформу доступнішою та простою у використанні. Водночас така відкритість може створювати ризики поширення спаму або недостовірної інформації. Тому для підтримки безпеки та якості контенту важливо впроваджувати ефективні механізми модерації та перевірки інформації.

Підсумовуючи, платформа забезпечує корисний та зручний функціонал для пошуку танцювальних подій, вакансій та створення власних заходів, проте потребує суттєвих удосконалень у сфері доступності, безпеки та розширення можливостей.

Наступним розглянемо вебсервіс «Dance Place» [5] (рис. 1.3).

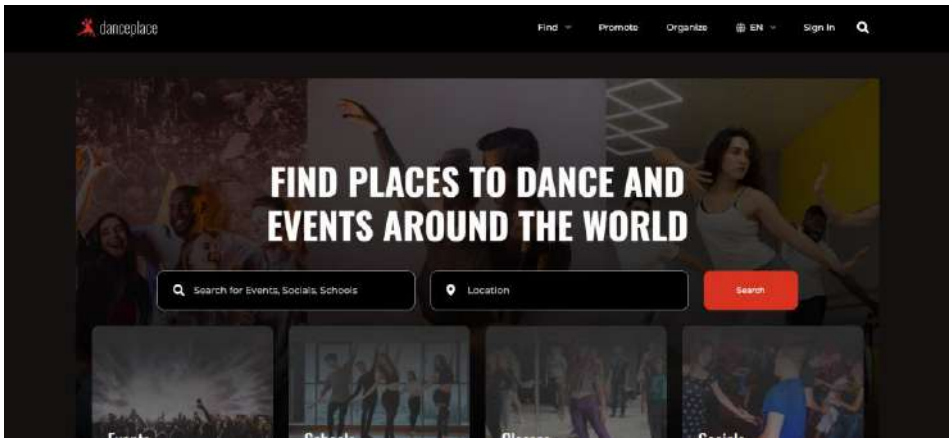


Рисунок 1.3 Головна сторінка вебсайту «Dance Place»

Одразу привертає увагу привабливий та сучасний інтерфейс сайту. Головна сторінка добре організована, не перевантажена непотрібними елементами, що забезпечує зручну навігацію. Окремі поля для пошуку за

ключовими словами та за місцем розташування сприяють швидкому старту роботи з платформою. Проте, набір доступних фільтрів є, що ускладнює пошук при великій кількості заходів.

Сторінка з подіями реалізована логічно та інформативно – кожен захід супроводжується детальним описом. Особливо корисною є можливість одразу зареєструватися на подію або придбати квитки.

У вебзастосунку також закладено концепцію танцювальної школи, що є позитивним моментом у порівнянні з аналогічними сервісами. Водночас функціонал, пов'язаний з цим елементом, або ще не реалізовано, або він працює з помилками. Наприклад, пошукові фільтри мають нестилізований вигляд, а при виборі «Canada», де згідно з інформацією на сайті зареєстровано понад 400 шкіл, жодна з них не відображається (рис. 1.4).

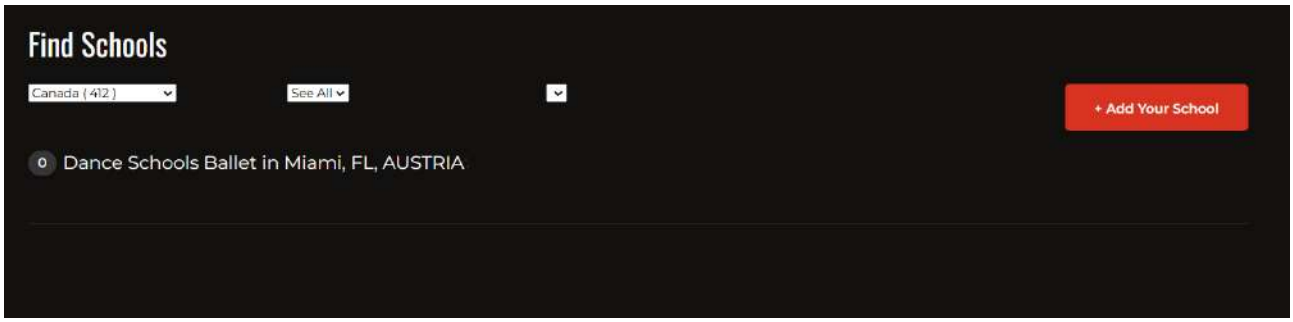


Рисунок 1.4 Сторінка пошуку шкіл

Також варто зазначити, що наразі не реалізовано окрему сутність викладача та можливість запису на приватні уроки, що могло б значно розширити функціонал платформи та зробити її привабливішою для користувачів.

Окремо було протестовано доступність сайту за допомогою скрінрідера NVDA. Результати виявили часткову доступність: базова навігація можлива, однак деякі елементи інтерфейсу не мають належної семантики або не озвучуються. Це ускладнює взаємодію для користувачів із вадами зору.

Отож проведене дослідження підтверджує, що танцювальна сфера активно представлена у цифровому середовищі, й існує низка вебсервісів, які вже сьогодні дозволяють користувачам шукати події, взаємодіяти з танцювальною спільнотою та публікувати власний контент. Зокрема, сильними сторонами проаналізованих вебсервісів є: інформативність, доступ до подій, зручний інтерфейс та розширений функціонал для організаторів.

Водночас вони мають й низку суттєвих недоліків, таких як обмежені фільтри для пошуку, відсутність функцій приватних уроків та взаємодії з викладачами та недостатній рівень адаптованості для людей з інвалідністю.

Отож з вище наведеного аналізу випливає потреба у створенні нової, сучасної та доступної платформи, яка поєднає переваги існуючих сервісів і усуне їхні недоліки. Такий вебзастосунок має орієнтуватися на широке коло користувачів, забезпечувати зручні інструменти пошуку, реєстрації, організації занять, а також підтримувати якісний і доступний інтерфейс.

1.4. Аналіз технічного завдання

Наступним кроком після визначення потреб користувачів та вивчення існуючих рішень є формування та аналіз технічного завдання для створення сучасного вебзастосунку, орієнтованого на зручність, доступність і функціональність для різних категорій користувачів. Нижче наведено основні вимоги до системи, яка підтримує чотири типи користувачів: незареєстрованого користувача, зареєстрованого звичайного користувача, викладача та танцювальну школу.

Функціонал, доступний всім користувачам системи:

- Перегляд інформації про події, курси, танцювальні школи та викладачів з можливістю пошуку, фільтрування та сортування (зокрема також за опціями доступності).

- Перегляд відгуків зареєстрованих користувачів про танцювальні школи та їхнього рейтингу.
- Доступ до профілів інших користувачів (у тому числі викладачів та шкіл), перегляд організованих ними подій та курсів у вигляді списку та календаря з формою для фільтрації.
- Перегляд портфоліо та інформації про відділення танцювальної школи.
- Доступ по портфоліо викладачів та інформація про їхні вільні години.
- Проходження тесту на визначення танцювального стилю, який тобі підходить.
- Вхід у систему.

Додатковий функціонал, що доступний лише незареєстрованому користувачу:

- Реєстрація як звичайний користувач, викладач або танцювальна школа.

Функціонал, доступний лише зареєстрованим користувачам:

- Редагування особистої інформації
- Видалення акаунту і пов’язаної інформації
- Організація подій та керування ними: редагування та видалення.
- Перегляд власного календаря активностей.
- Вихід з акаунту.

Функціонал, доступний зареєстрованим звичайним користувачам та викладачам:

- Реєстрація на події та приватні курси.
- Додавання подій та курсів у вибрані.
- Написання відгуків про танцювальні школи та видалення їх.
- Подача запитів на приватні уроки та скасування запитів, перегляд інформації про них та відображення підтверджених у календарі.
- Перегляд списку реєстрацій на події та приватні курси.

Функціонал, доступний тільки танцювальним школам та викладачам:

- Додавання портфоліо з відео контентом та керування ним.

Додатковий функціонал, доступний тільки викладачам:

- Організація приватних курсів

- Додавання вільних таймслотів у календар та їх редагування/видалення.
- Керування запитами на приватні уроки.

Додатковий функціонал, доступний тільки танцювальним школам:

- Створення відділень та керування ними: редагування, видалення.
- Організація щотижневих курсів та керування ними.

Нефункціональними вимогами до застосунку є:

- Забезпечення безпеки користувацьких даних.
- Надання сучасного, зручного та доступного інтерфейсу.
- Сумісність з технологіями екранного зчитування.
- Висока швидкодія роботи застосунку, зокрема під час виконання складних операцій пошуку та фільтрації.

1.5. Висновки до розділу 1

У розділі 1 було досліджено та доведено актуальність проблеми доступності у сфері танцювальних послуг , зокрема для людей з інвалідністю, початківців, викладачів та представників різних груп за фінансовою забезпеченістю. Після сегментації цільової аудиторії було визначено конкретні потреби кожної групи, серед яких – зручний пошук заходів, можливість організації приватних уроків та наявність доступного інтерфейсу.

Аналіз існуючих рішень підтвердив факт того, що незважаючи на стрімкий технологічний прогрес, у сфері танців ще є прогалини, які треба заповнити, зокрема в контексті доступності інформації та забезпечення широкого і зручного функціоналу, який би задовільнив різні категорії користувачів.

На основі проведеного аналізу було сформульовано функціональні та нефункціональні вимоги до вебплатформи, які стануть основою для подальшого проектування та розробки системи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБПЛАТФОРМИ

2.1. Архітектура системи

Вебплатформа має контейнеризовану клієнт-серверну архітектуру, в якій усі основні компоненти працюють в окремих Docker-контейнерах. Такий підхід забезпечує гнучкість, ізоляцію сервісів і полегшує подальше масштабування.

Основні компоненти архітектури (рис. 2.1):

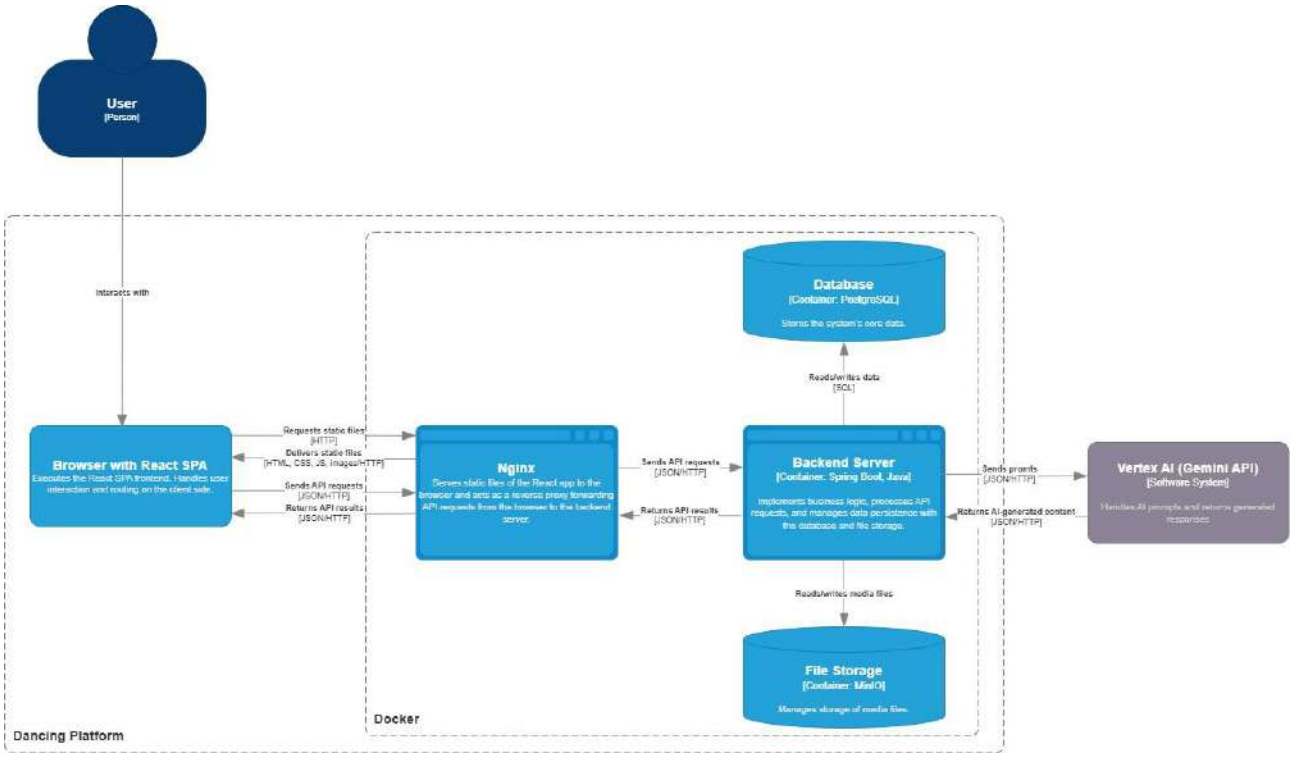


Рисунок 2.1 Діаграма c4 рівня 2 – схема компонентів системи

- **Сервер (Spring Boot)**: реалізовує бізнес логіку, обробляє запити користувачів і відповідає за роботу з базою даних та файловим сховищем.
- **Клієнт (React)**: забезпечує інтерфейс користувача та реалізує взаємодію з сервером через API.

- **Проксі-сервер (Nginx)**: виконує роль зворотного проксі-сервера, який обслуговує статичні ресурси (HTML, CSS, JS, медіафайли) React-додатку та маршрутизує API-запити на серверну частину.
- **База даних (PostgreSQL)**: реляційна система керування базами даних, яка використовується для надійного зберігання даних.
- **Сховище файлів (MinIO)**: об'єктне сховище, що використовується для зберігання великих медіафайлів (зображень та відео) та надає API для подальшої роботи з ними.
- **Платформа управління контейнерами (Docker)**: забезпечує єдине середовище для розгортання всієї платформи та ізоляцію компонентів.
- **Зовнішня система (Vertex AI Gemini API)**: сервіс генеративного ШІ від Google, який сервер використовує для обробки текстових запитів та отримання відповідей через HTTP у форматі JSON.

Комунікація між клієнтською та серверною частинами здійснюється за допомогою REST API через протокол HTTP у форматі JSON. Статичні файли запитуються браузером у Nginx і передаються у вигляді стандартних відповідей HTTP з MIME-типами, відповідно до вмісту (наприклад, text/html, application/javascript, text/css тощо).

Було обрано використовувати **клієнт-серверну архітектуру**, так як вона забезпечує чіткий поділ відповідальностей між частинами системи: клієнт відповідає за взаємодію з користувачем, а сервер – за логіку обробки запитів, роботу з базою даних та іншими сервісами [6]. Такий підхід дозволяє легко оновлювати й масштабувати кожен компонент окремо.

Ще однією перевагою є те, що над клієнтом і сервером можуть працювати різні команди одночасно, що прискорює розробку та оновлення системи.

Крім того, така архітектура полегшує процес тестування. Оскільки клієнт і сервер є незалежними компонентами, їх можна тестувати окремо. Це допомагає виявляти помилки на ранніх стадіях розробки, підтримуючи високу якість коду.

З урахуванням усіх вищезазначених переваг, клієнт-серверна архітектура є оптимальним рішенням для побудови нашого програмного сервісу.

Щодо комунікації між клієнтом і сервером, було прийнято рішення використовувати **REST API** [7], оскільки це простий і стандартизований спосіб обміну даними через HTTP. Він добре підтримується сучасними фреймворками, легко тестується, масштабований і дозволяє передавати дані у зручному форматі JSON. Це робить його чудовим вибором для вебплатформи з клієнт-серверною архітектурою.

Для забезпечення стабільності, гнучкості та простоти розгортання усі основні компоненти платформи були контейнеризовані за допомогою **Docker**.

2.2. Обгрунтування вибору технологій та інструментів розробки

Для серверної частини вебзастосунку використовується мова програмування Java з фреймворком Spring Boot.

Java було обрано через її стабільність, багатий набір бібліотек, можливість масштабування та потужну екосистему, яка спрощує інтеграцію з різними базами даних та зовнішніми сервісами. Велика кількість документації та онлайн-ресурсів допомагає швидко знаходити відповіді на складні питання, а у разі нестачі інформації, активна спільнота розробників забезпечує швидку підтримку та обмін знаннями [8].

Spring Boot, в свою чергу, є популярним рішенням для розробки серверної частини вебзастосунків. Він забезпечує зручне управління життєвим циклом додатку, автоматичне налаштування конфігурації та підтримку вбудованих серверів, що значно спрощує процес розробки і розгортання. Однією з головних

переваг Spring Boot є його інтеграція зі Spring Data JPA, що забезпечує зручну і ефективну роботу з базами даних. Крім того, Spring Boot активно використовує анотації для конфігурації компонентів і впровадження залежностей, що дозволяє легко керувати зв’язками між класами, підвищує модульність і тестованість застосунку [9].

Як систему збірки було використано **Maven**. Він автоматизує процес компіляції, управління залежностями та створення проєкту [10].

PostgreSQL вибрано через його розширені можливості роботи з транзакціями та кращу відповідність стандартам SQL у порівнянні з MySQL, яка використовувалась раніше. На відміну від MySQL, PostgreSQL забезпечує більш гнучкі механізми індексації та кращу продуктивність при складних запитах [11].

Як систему зберігання файлів було використано **MinIO**. Це високопродуктивне об'єктне сховище з відкритим кодом, сумісне з Amazon S3, яке забезпечує надійне та масштабоване збереження даних зі зручним веб-інтерфейсом [12].

Клієнтську частину вебзастосунку реалізовано на мові **TypeScript**, яка є надбудовою над JavaScript. Вона додає статичну типізацію, що допомагає виявляти помилки на етапі розробки, підвищує якість коду та спрощує його підтримку [13].

React було обрано через його компонентну архітектуру, яка дозволяє створювати повторно використовувані, ізольовані частини інтерфейсу. Ще однією ключовою перевагою React є використання віртуального DOM, який працює як проміжний шар між логікою застосунку і реальним DOM. Замість безпосереднього оновлення елементів на сторінці, React спочатку вносить зміни у віртуальну копію DOM, а потім ефективно визначає, які саме частини потрібно змінити в реальному DOM. Такий підхід мінімізує кількість операцій у браузері, що робить інтерфейс швидким навіть при частих оновленнях [14].

Nginx використано як вебсервер і зворотний проксі-сервер через його високу продуктивність та низьке споживання ресурсів. Крім того, він добре підходить для розгортання React-додатків у контейнерах Docker: дозволяє просто налаштувати маршрути та швидко подавати зібраний клієнтський [15].

Docker було використано для спрощення процесу розгортання і забезпечення стабільної роботи застосунку в різних середовищах. Використання контейнерів дозволяє ізолювати сервіси один від одного та уникнути проблем із залежностями. Такий підхід значно спрощує масштабування системи, дозволяє швидко впроваджувати оновлення та підтримувати безперебійну роботу сервісів, що є важливим для реалізованого програмного продукту.

2.3. Висновки до розділу 2

У розділі 2 було детально описано архітектуру вебплафторми та надано діаграму компонентів системи. Було обрано клієнт-серверний підхід як найбільш придатний для забезпечення гнучкої, масштабованої та незалежної взаємодії між фронтендом і бекендом. Для обміну даними між клієнтом і сервером передбачено використання REST API, що дозволяє стандартизувати комунікацію, спростити інтеграцію та підтримувати зрозумілу структуру запитів. З метою полегшення розгортання системи та її масштабування передбачено використання Docker-контейнеризації.

Також обґрунтовано вибір технологій та інструментів розробки. Зокрема, використання Java із Spring Boot для розробки серверної логіки, MinIO - для зберігання файлів та PostgreSQL - як основна реляційна база даних. Для клієнтської частини обрано використовувати React, TypeScript та Nginx.

РОЗДІЛ 3. РОБОТА З БАЗОЮ ДАНИХ

3.1. Структура бази даних

У проєкті використовується реляційна база даних із системою керування базами даних PostgreSQL. Структура бази даних була автоматично створена за допомогою Spring Data JPA та Hibernate на основі Entity-класів, визначених у коді (рис. 3.1).

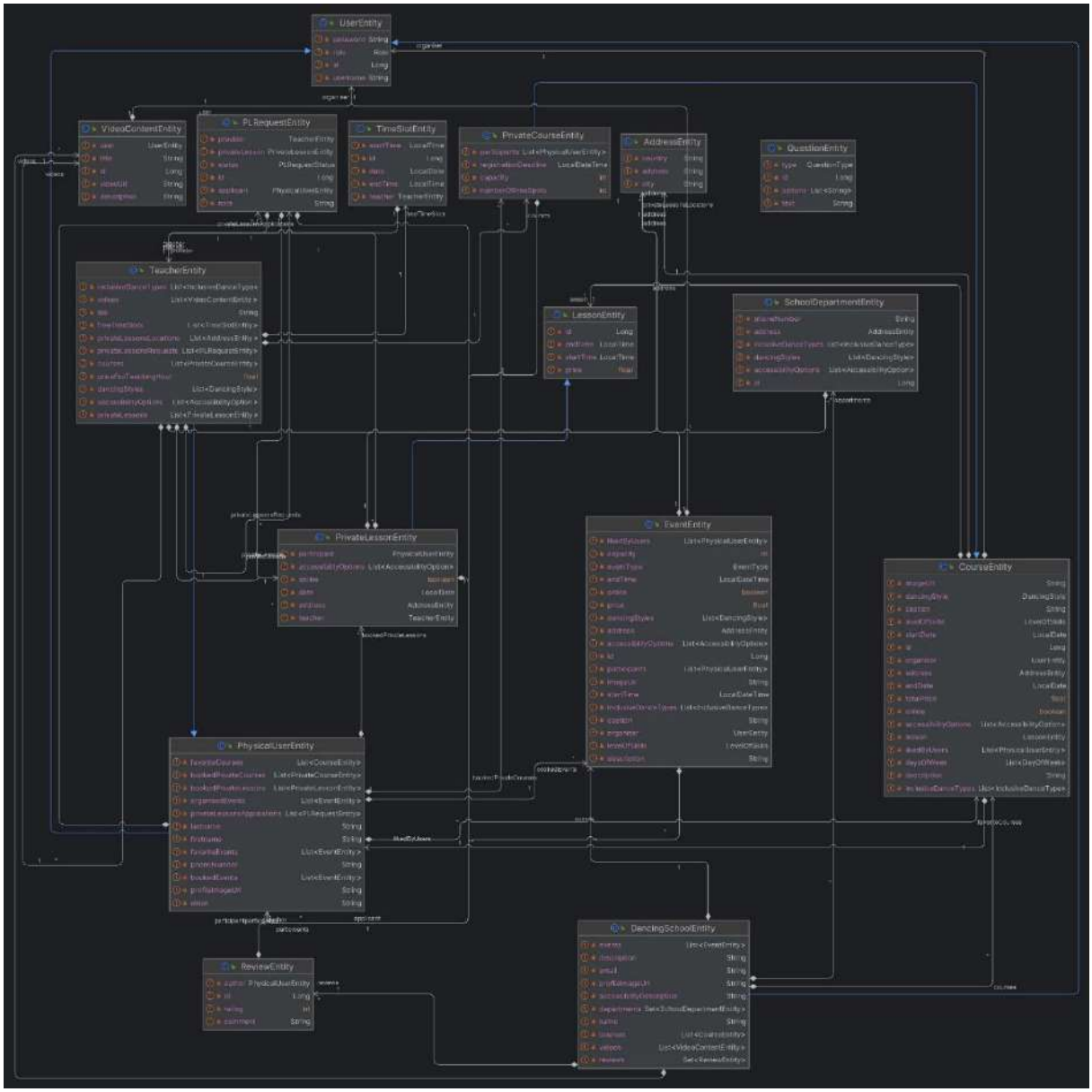


Рисунок 3.1 ER діаграма згенерована IntelliJ IDEA

Проаналізуємо детальніше структуру бази даних.

Директорія з сутностями бази даних називається **model** (рис. 3.2).

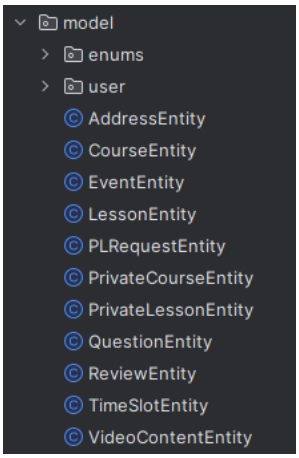


Рисунок 3.2 Структура директорії model

У цій директорії міститься підпапка **enums**, де зберігаються енумерації, що використовуються у різних сутностях. Зокрема, це такі енумерації:

- DancingStyle
- DayOfWeek
- EventType
- LevelOfSkills
- PLRequestStatus – може набувати значень OPEN, CANCELLED, ACCEPTED та DECLINED.
- QuestionType
- Role з можливими значеннями REGISTERED_USER, TEACHER і DANCING_SCHOOL
- та енумерації для забезпечення доступності: AccessibilityOption та InclusiveDanceType.

Директорія **user** включає сутності, що представляють різні типи користувачів, а саме класи:

- UserEntity – використовується як батьківський клас для всіх наступних типів користувачів. Містить лише базові поля: ім'я користувача, пароль і роль.
- PhysicalUserEntity – розширює UserEntity та додає інформацію, характерну для фізичної особи – електронну пошту, номер телефону, ім'я та прізвище. Крім того, у ньому зберігаються зв’язки з подіями, курсами та приватними уроками, які користувач організовує, бронює або позначає як улюблені.
- TeacherEntity – розширює PhysicalUserEntity та представляє викладача. Містить додаткові поля: ціна за годину навчання, біографія, доступні стилі танців, типи інклюзивного танцю та опції доступності. Також зберігає локації, де викладач може проводити приватні заняття, його вільні часові слоти, приватні уроки, курси та відеоконтент.
- DancingSchoolEntity – розширює UserEntity та являє собою танцювальну школу. Містить поля з інформацією про назву, електронну пошту, опис школи, зображення профілю, опис доступності та відеоконтент. Має зв’язки з курсами та подіями, організованими школою, відгуками користувачів, а також відділеннями, представленими через SchoolDepartmentEntity.

У головній директорії **model** міститься клас CourseEntity, що описує загальну структуру танцювального курсу та виступає батьківською сутністю для PrivateCourseEntity, який представляє приватний курс, що може бути створений лише викладачем. На відміну від звичайного, приватний курс підтримує реєстрацію учасників – для цього визначено зв’язок з сутністю PhysicalUserEntity.

Крім того визначено класи-сутності LessonEntity, PrivateLessonEntity та PLRequestEntity. LessonEntity є батьківським касом для PrivateLessonEntity, що представляє собою приватний урок та містить додатково зв’язок з сутністю

викладача та фізичного користувача як учасника. PLRequestEntity являє собою запит на приватний урок.

Також наявні класи: AddressEntity, TimeSlotEntity (містить зв’язок з викладачем), ReviewEntity, QuestionEntity та VideoContentEntity.

3.2. Стратегії формування запитів до бази даних

Spring Data JPA є частиною фреймворку Spring Boot, що значно спрощує роботу з базами даних. Вона надає високрівневий API для роботи з JPA (Java Persistance API), реалізований у вигляді абстрактного шару JPA репозиторіїв, що забезпечують розробників готовими CRUD операціями над сутностями бази даних [16]. Це чудово покриває необхідні операції для невеликих та нескладних застосунків, економить час та зменшує кількість стандартизованого коду.

Саме цей підхід було початково використано для роботи з базою даних. Проте, оскільки однією з ключових вимог до вебплатформи було забезпечення розширеного та гнучкого пошуку з параметрами фільтрації та сортування, базових можливостей виявилось недостатньо.

У зв’язку з цим було проаналізовано підходи до побудови складніших запитів у межах екосистеми Spring. Одним з ефективних рішень став механізм формування запитів на основі конвенцій імен у Spring Data JPA. Саме таким чином реалізовано метод findAllByTeacherIdAndParticipantId, який дозволяє отримувати приватні уроки за ідентифікаторами викладача та учасника (рис. 3.3).

```
@Repository
public interface IPrivateLessonRepository extends JpaRepository<PrivateLessonEntity, Long> {
    1 usage  new *
    List<PrivateLessonEntity> findAllByTeacherIdAndParticipantId(Long teacherId, Long participantId);
```

Рисунок 3.3 Метод findAllByTeacherIdAndParticipantId. Використовує конвенцію імен для генерації

У даному випадку Spring Data JPA генерує запит із назви методу. Однак такий підхід зручний лише при невеликій кількості умов. Коли ж виникає потреба у складнішій логіці, доцільно використовувати анотацію @Query, де SQL-запит вказується явно. Прикладом такої реалізації є метод findAllLikedCoursesByUser, який дозволяє отримати перелік курсів, вподобаних конкретним користувачем (рис. 3.4).

Рисунок 3.4 Метод findAllLikedCoursesByUser. Використовує @Query

Розширений пошук можна було б реалізувати за допомогою даного підходу, однак для зручнішої роботи з динамічними фільтрами та обробки великої кількості параметрів було віддано перевагу Criteria API. Цей інструмент дозволяє програмно будувати запити з будь-якою кількістю умов. Це не лише спрощує розширення функціоналу в майбутньому, а й робить код більш структурованим і підтримуваним. Спільні частини запитів можна виносити в окремі методи, що дозволяє уникнути дублювання та підвищує читабельність коду.

Саме так було реалізовано пошуковий функціонал для подій, курсів, шкіл, викладачів, приватних уроків і так далі. Для цього визначено окремий сервіс SearchService що реалізовує інтерфейс ISearchService (рис. 3.5).

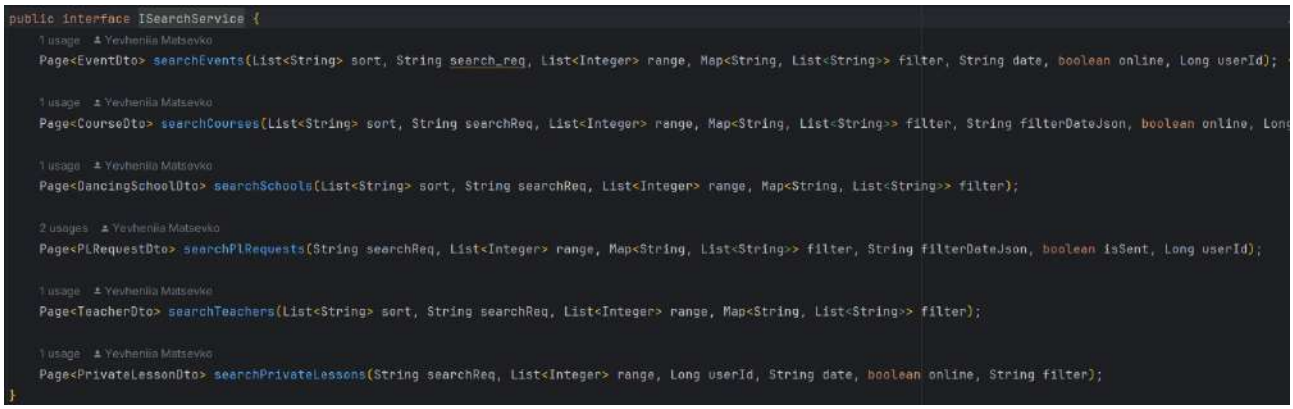


Рисунок 3.5 Інтерфейс ISearchService

Також при виконанні пошуку враховується пагінація результатів, щоб ефективно працювати з великими вибірками даних. Для цього було виокремлено узагальнений метод executePagedQuery (рис. 3.6).

```
public <T> Page<T> executePagedQuery(
        CriteriaQuery<T> query,
        List<Integer> range
) {
    int page = Math.max(range.get(0), 0);
    int size = range.get(1);

    TypedQuery<T> typedQuery = entityManager.createQuery(query);

    int totalRows = typedQuery.getResultList().size();

    typedQuery.setFirstResult(page * size);
    typedQuery.setMaxResults(size);

    List<T> results = typedQuery.getResultList();

    return new PageImpl<>(results, PageRequest.of(page, size), totalRows);
}
```

Рисунок 3.6 Реалізація методу executePagedQuery

Він на вхід приймає CriteriaQuery та параметри пагінації - номер сторінки й кількість елементів на сторінці, та повертає об’єкт типу Page<T>, що містить отримані дані поточної сторінки разом з інформацією про загальну кількість записів у вибірці. Цей метод використовується для формування результатів кожного пошукового запиту сервісу.

Таким чином, у проєкті використовуються усі чотири підходи формування запитів до бази даних: від стандартних CRUD-операцій та методів із генерацією

на основі імен, до ручного визначення логіки через @Query та побудови запитів за допомогою Criteria API. Кожен із них застосовується там, де це найдоцільніше: прості запити – через CRUD і конвенції імен, окремі специфічні – через @Query, а гнучкі та динамічні – через Criteria. Це дозволило досягти балансу між простотою реалізації та гнучкістю, зробивши роботу з даними максимально ефективною та зрозумілою.

3.3. Висновки до розділу 3

У цьому розділі представлено структуру бази даних, реалізовану за допомогою Spring Data JPA та Hibernate, що забезпечує зручне та ефективне управління даними. Завдяки використанню ORM-підходу розробка зосереджується на бізнес-логіці, мінімізуючи потребу в написанні низькорівневих SQL-запитів. Наведено ER-діаграму, а також коротко описано основні сутності та зв’язки між ними.

Для формування запитів до бази даних застосовано комбінований підхід: стандартні CRUD-операції Spring Data JPA, методи, згенеровані за конвенцією імен, SQL-запити через анотацію @Query, а також динамічні запити, побудовані за допомогою Criteria API. Така комбінація дозволяє ефективно працювати як із простими, так і з комплексними сценаріями пошуку й фільтрації, зберігаючи при цьому читабельність і масштабованість коду.

РОЗДІЛ 4. ОГЛЯД СЕРВЕРНОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ

4.1. Забезпечення безпеки

Для забезпечення безпеки у вебзастосунку використовується **Spring Security** у поєднанні з **JSON Web Token (JWT)**.

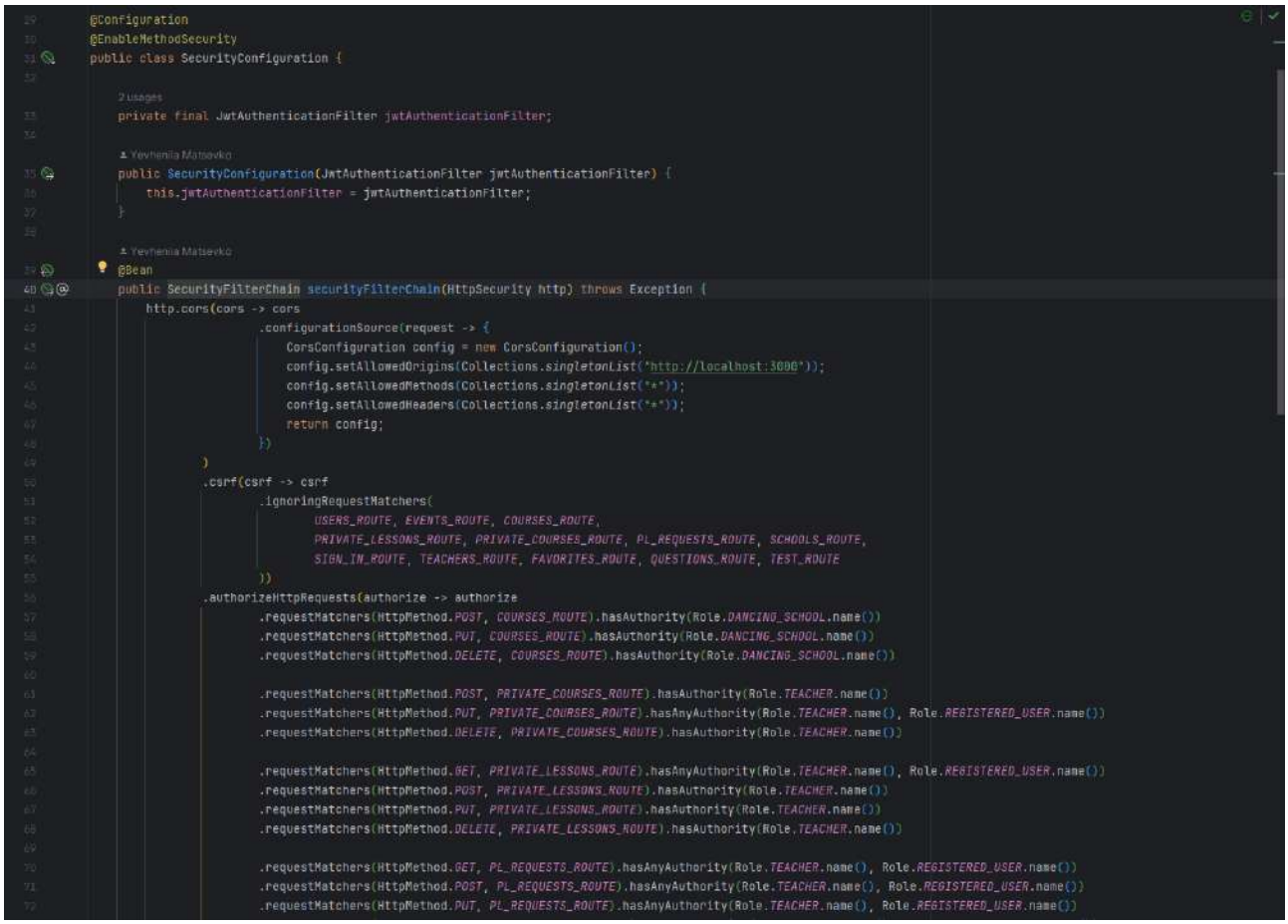
JWT складається з трьох частин, які кодуються окремо та розділені крапками [17]:

1. Заголовок (Header) – тут записані основні відомості про тип токена і метод шифрування, який використовується для підпису (наприклад, HS256 або RS256). Це допомагає системі зрозуміти, як працювати з токеном.
2. Тіло (Payload) – ця частина містить інформацію про користувача і додаткові дані. Тут можуть бути ID користувача, роль, дата створення токена, термін його дії тощо.
3. Підпис (Signature) – це цифровий підпис, який формується на основі заголовка і тіла за допомогою секретного ключа або пари ключів. Він гарантує, що токен не змінений і захищає від підробки.

Завдяки такій структурі JWT дозволяє безпечно передавати інформацію між клієнтом і сервером без потреби зберігання стану сесії на сервері.

Spring Security в свою чергу відповідає за перевірку токена, визначення прав доступу та захист ресурсів від несанкціонованого використання [18].

Клас SecurityConfiguration містить основні визначення забезпечення безпеки застосунку (рис. 4.1).



```java
@Configuration
@EnableMethodSecurity
public class SecurityConfiguration {

    2 usages
    private final JwtAuthenticationFilter jwtAuthenticationFilter;

    ▲ Yevhenia Matsevko
    public SecurityConfiguration(JwtAuthenticationFilter jwtAuthenticationFilter) {
        this.jwtAuthenticationFilter = jwtAuthenticationFilter;
    }

    ▲ Yevhenia Matsevko
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.cors(cors -> cors
                .configurationSource(request -> {
                    CorsConfiguration config = new CorsConfiguration();
                    config.setAllowedOrigins(Collections.singletonList("http://localhost:3000"));
                    config.setAllowedMethods(Collections.singletonList("*"));
                    config.setAllowedHeaders(Collections.singletonList("*"));
                    return config;
                })
        )
        .csrf(csrf -> csrf
                .ignoringRequestMatchers(
                        USERS_ROUTE, EVENTS_ROUTE, COURSES_ROUTE,
                        PRIVATE_LESSONS_ROUTE, PRIVATE_COURSES_ROUTE, PL_REQUESTS_ROUTE, SCHOOLS_ROUTE,
                        SIGN_IN_ROUTE, TEACHERS_ROUTE, FAVORITES_ROUTE, QUESTIONS_ROUTE, TEST_ROUTE
                ))
        .authorizeHttpRequests(authorize -> authorize
                .requestMatchers(HttpMethod.POST, COURSES_ROUTE).hasAuthority(Role.DANCING_SCHOOL.name())
                .requestMatchers(HttpMethod.PUT, COURSES_ROUTE).hasAuthority(Role.DANCING_SCHOOL.name())
                .requestMatchers(HttpMethod.DELETE, COURSES_ROUTE).hasAuthority(Role.DANCING_SCHOOL.name())

                .requestMatchers(HttpMethod.POST, PRIVATE_COURSES_ROUTE).hasAuthority(Role.TEACHER.name())
                .requestMatchers(HttpMethod.PUT, PRIVATE_COURSES_ROUTE).hasAnyAuthority(Role.TEACHER.name(), Role.REGISTERED_USER.name())
                .requestMatchers(HttpMethod.DELETE, PRIVATE_COURSES_ROUTE).hasAuthority(Role.TEACHER.name())

                .requestMatchers(HttpMethod.GET, PRIVATE_LESSONS_ROUTE).hasAnyAuthority(Role.TEACHER.name(), Role.REGISTERED_USER.name())
                .requestMatchers(HttpMethod.POST, PRIVATE_LESSONS_ROUTE).hasAuthority(Role.TEACHER.name())
                .requestMatchers(HttpMethod.PUT, PRIVATE_LESSONS_ROUTE).hasAuthority(Role.TEACHER.name())
                .requestMatchers(HttpMethod.DELETE, PRIVATE_LESSONS_ROUTE).hasAuthority(Role.TEACHER.name())

                .requestMatchers(HttpMethod.GET, PL_REQUESTS_ROUTE).hasAnyAuthority(Role.TEACHER.name(), Role.REGISTERED_USER.name())
                .requestMatchers(HttpMethod.POST, PL_REQUESTS_ROUTE).hasAnyAuthority(Role.TEACHER.name(), Role.REGISTERED_USER.name())
                .requestMatchers(HttpMethod.PUT, PL_REQUESTS_ROUTE).hasAnyAuthority(Role.TEACHER.name(), Role.REGISTERED_USER.name())
```

Рисунок 4.1 Клас SecurityConfiguration

У ньому конфігурується CORS, вимикається CSRF для певних маршрутів, визначаються дозволи на HTTP-запити відповідно до ролей користувачів (REGISTERED_USER, DANCING_SCHOOL, TEACHER), встановлюється політика сесії як STATELESS, додається фільтр JWT-автентифікації, а також налаштовується провайдер автентифікації, менеджер автентифікації та сервіси для обробки користувацьких даних.

4.2. Взаємодія з MinIO

Для зберігання користувацьких фото- та відеоматеріалів у даному вебзастосунку використовується система зберігання об’єктів MinIO. Це високопродуктивне, масштабоване та сумісне з API Amazon S3 рішення, яке

дозволяє ефективно керувати великими обсягами неструктурованих даних, таких як зображення, відео та резервні копії [12].

У MinIO дані організовані в «бакети» (buckets), які функціонують подібно до каталогів у файловій системі та кожен бакет може містити необмежену кількість об'єктів.

Також MinIO є повністю безкоштовним та з відкритим вихідним кодом (open source), що дозволяє вільно використовувати його без ліцензійних обмежень. Крім того, MinIO надає зручний веб-інтерфейс, використовуючи який адміністратори можуть легко керувати вмістом бакетів напряму через браузер.

У реалізованому вебзастосунку MinIO розгорнуто у середовищі Docker. Для цього до загального docker-compose файлу проєкту додано окремий сервіс minio, що запускається в окремому контейнері (рис. 4.2).

```
minio:
  image: minio/minio:latest
  container_name: minio
  restart: unless-stopped
  volumes:
    - "./minio/data:/data"
    - "./minio/minio:/minio"
    - "./minio/config:/root/.minio"
  environment:
    TZ: ${TZ}
    LANG: en_US.UTF-8
    MINIO_PROMETHEUS_AUTH_TYPE: "public"
    MINIO_ACCESS_KEY: ${MINIO_ACCESS_KEY}
    MINIO_SECRET_KEY: ${MINIO_SECRET_KEY}
  command: server /data --console-address ":9001"
  logging:
    driver: "json-file"
    options:
      max-size: "200m"
  ports:
    - "9000:9000"
    - "9001:9001"
```

Рисунок 4.2 Конфігурація контейнеру minio

Використовується офіційний образ minio/minio:latest. Контейнер має назву minio і автоматично перезапускається, якщо щось іде не так, завдяки параметру restart: unless-stopped.

У MinIO монтуються кілька локальних папок через volumes, зокрема:

- ./minio/data:/data – для самих файлів,
- ./minio/minio:/minio – додаткові ресурси,
- ./minio/config:/root/.minio – конфігурація.

Через змінні середовища передаються часовий пояс (TZ), мова, а також ключ доступу (MINIO_ACCESS_KEY) і секретний ключ (MINIO_SECRET_KEY).

Веб-інтерфейс MinIO запускається на порту **9001**, а основний сервер обробки запитів працює на порту **9000**, через який і здійснюється взаємодія з backend-застосунком.

Для взаємодії з MinIO на сервері визначено клас конфігурації, який створює Bean типу MinioClient із нашими локальними налаштуваннями (рис 4.3).

```
@Data
@Configuration
@ConfigurationProperties(prefix = "minio")
public class MinioConfig {
    private String endpoint;
    private String accessKey;
    private String secretKey;
    private String bucketName;
    Yevheniia Matsevko
    @Bean
    public MinioClient minioClient() {
        return MinioClient.builder()
                .endpoint(endpoint)
                .credentials(accessKey, secretKey)
                .build();
    }
}
```

Рисунок 4.3 Клас MinioConfig

Крім того, визначено допоміжний клас-утиліта MinioUtils з операціями роботи з бакетами та файлами. Однією з основних функцій є завантаження файлу на сервіс (метод uploadFile) (рис. 4.4), розглянемо її детальніше.

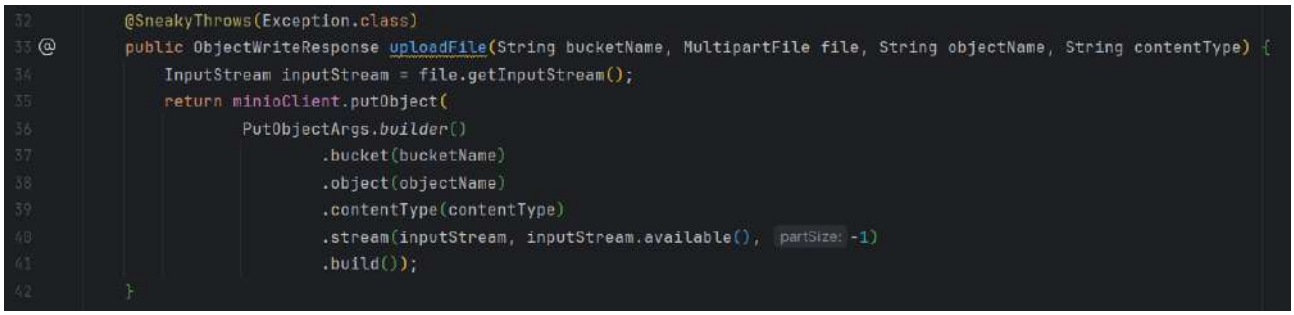


Рисунок 4.4 Метод uploadFile класу MinioUtils

Даний метод приймає такі параметри: назву бакету, сам файл у форматі MultipartFile, ім’я файлу та його формат.

За допомогою методу getInputStream() отримуємо потік даних з файлу. Далі викликаємо метод putObject клієнта MinIO, у який передаємо налаштування:

- куди саме завантажувати файл (bucketName),
- як назвати файл (objectName),
- тип контенту (contentType),
- сам потік даних із розміром файлу.

Крім того, важливо зазначати, що для забезпечення безпеки, бакет, що використовується застосунком має приватний рівень доступу. Це означає, що прямий доступ до файлів ззовні обмежений. Щоб фронтенд міг отримати доступ до фото чи відео, сервер надає йому спеціально згенероване посилання за допомогою функції getPresignedObjectUrl (рис. 4.5)

```
public String getPresignedObjectUrl(String bucketName, String objectName) {
    GetPresignedObjectUrlArgs args = GetPresignedObjectUrlArgs.builder()
            .bucket(bucketName)
            .object(objectName)
            .method(Method.GET).build();
    return minioClient.getPresignedObjectUrl(args);
}
```

Рисунок 4.5 Метод getPresignedObjectUrl класу MinioUtils

Вона створює тимчасове посилання (pre-signed URL) для конкретного об’єкта у бакеті. Посилання дає дозвіл на доступ до файлу без необхідності відкривати бакет повністю. Параметри вказують, з якого бакету і який файл потрібно отримати, а також що доступ буде здійснюватися через HTTP-метод GET [12].

Таким чином, фронтенд отримує безпечний тимчасовий URL для завантаження необхідних матеріалів, не порушуючи рівень приватності бакету.

4.3. Використання бібліотеки Lombok

Для зменшення кількості шаблонного коду у проєкті було використано бібліотеку Lombok. Вона містить 16 основних анотацій [19], з яких в проєкті було застосовано 9, а саме @ToString, @EqualsAndHashCode, @NoArgsConstructor, @AllArgsConstructor, @RequiredArgsContructor, @Data, @Getter, @Setter та @SneakyThrows.

Розглянемо для прикладу клас EventDto (рис. 4.6).

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class EventDto {
    private Long id;
    private String imageUrl;
    private String caption;
    private EventType eventType;
    private LocalDateTime startTime;
    private LocalDateTime endTime;
    private AddressDto address;
    private float price;
    private LevelOfSkills levelOfSkills;
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    private List<PhysicalUserDto> participants;
    private String description;
    private boolean online;
    private List<DancingStyle> dancingStyles;
    private int capacity;
    private UserDto organiser;
    private List<AccessibilityOption> accessibilityOptions;
    private List<InclusiveDanceType> inclusiveDanceTypes;
}
```

Рисунок 4.6 Клас EventDto

З використанням анотацій Lombok код класу займає 23 рядки.

Для експерименту визначимо усі автозгенеровані методи Lombok перепишемо цей клас без Lombok, вручну визначивши всі гетери, сетери, конструктори, методи toString, equals, hashCode (рис. 4.7, 4.8).

```java
    @Override
    public int hashCode() {
        return Objects.hash(id, imageUrl, caption, eventType, startTime, endTime, address, price,
                levelOfSkills, description, online, dancingStyles, capacity, organiser,
                accessibilityOptions, inclusiveDanceTypes);
    }

    new *
    @Override
    public String toString() {
        return "EventDto{" +
                "id=" + id +
                ", imageUrl='" + imageUrl + '\'' +
                ", caption='" + caption + '\'' +
                ", eventType=" + eventType +
                ", startTime=" + startTime +
                ", endTime=" + endTime +
                ", address=" + address +
                ", price=" + price +
                ", levelOfSkills=" + levelOfSkills +
                ", description='" + description + '\'' +
                ", online=" + online +
                ", dancingStyles=" + dancingStyles +
                ", capacity=" + capacity +
                ", organiser=" + organiser +
                ", accessibilityOptions=" + accessibilityOptions +
                ", inclusiveDanceTypes=" + inclusiveDanceTypes +
                '}';
    }
}
```

Рисунки 4.7, 4.8 Початок та кінець класу EventDto з вручну визначеними методами

Тепер реалізація класу EventDto займає аж 169 рядків коду.

Такий об’ємний код значно важче читати: основна логіка класу губиться серед великої кількості шаблонних методів. Його складніше швидко зрозуміти, довше писати вручну та суттєво важче змінювати – навіть незначна модифікація може потребувати редагування десятків рядків. Це підвищує ризик помилок і ускладнює підтримку. Таким чином, використання Lombok є виправданим і доцільним для підвищення ефективності розробки.

4.4. Контейнеризація застосунку в Docker

Docker – це інструмент, який дозволяє «запакувати» застосунок разом з усім необхідним для його роботи (бібліотеки, залежності, середовище) в окремий ізольований контейнер. Кожен контейнер працює незалежно і поводиться однаково на будь-якій машині, де встановлений Docker [20].

Основні складові Docker компонентів [20]:

- **Dockerfile** – файл з інструкціями, як саме зібрати образ контейнера;
- **Docker Image** – зібраний з Dockerfile образ, що містить усе необхідне для запуску;
- **Docker Container** – запущений екземпляр образу;
- **Docker Compose** – інструмент, який дозволяє одночасно запускати кілька пов’язаних між собою контейнерів.

У даному вебзастосунку наявні 4 docker-контейнери (рис. 4.9):

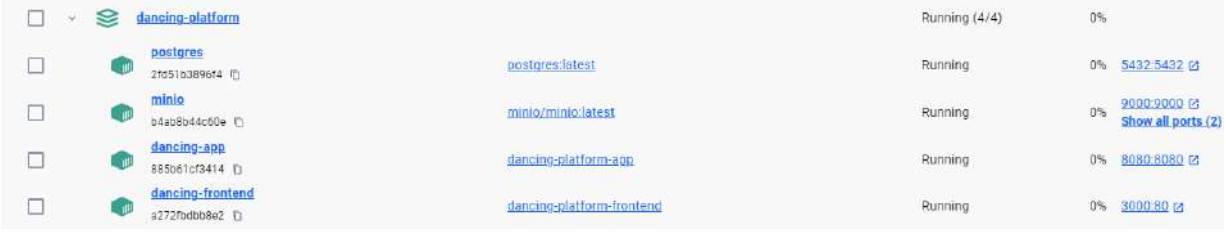


Рисунок 4.9 Docker-контейнери застосунку

- **minio**: контейнер сервісу MinIO;
- **postgresql:** контейнер бази даних PostgreSQL;
- **dancing-app**: серверна частина, реалізована на Spring Boot;
- **dancing-frontend**: контейнер з клієнтською частиною, який розгортається через Nginx.

Сервер та клієнт мають власні Dockerfile.

Приклад Dockerfile серверу ви можете побачити на рисунку (рис. 4.10).

Рисунок 4.10 Dockerfile серверу

Пояснення:

- FROM openjdk:24 – задає базовий образ з встановленим JDK версії 24, на якому буде будуватися контейнер.
- WORKDIR /app – встановлює робочу директорію всередині контейнера, куди будуть копіюватися файли.
- COPY target/DancingApp-0.0.1-SNAPSHOT.jar app.jar – копіює скомпільований JAR-файл з локальної директорії проєкту у внутрішню папку контейнера.
- ENTRYPOINT ["java", "-jar", "app.jar"] – команда, яка запускається під час старту контейнера, виконує запуск Java-додатку.

В кореневій директорії проєкту знаходиться файл docker-compose.yml, який відповідає за одночасну збірку та запуск усіх контейнерів системи. Він дозволяє зручно описати взаємодію між контейнерами [21].

Одна з важливих директив у docker-compose – це depends_on, яка вказує, що певний контейнер залежить від запуску інших. Наприклад у нашому випадку, серверна частина (бекенд) залежить від бази даних та сховища файлів, тому вона не буде запускатись, поки не будуть готові postgres та minio (рис. 4.11).

```
39  app:
40    build:
41      context: server/DancingApp
42      dockerfile: Dockerfile
43    container_name: dancing-app
44    depends_on:
45      - postgres
46      - minio
47    environment:
```

Рисунок 4.11 Конфігурація контейнеру app

Аналогічно, фронтенд залежить від серверної частини, адже без неї немає сенсу запускати інтерфейс.

4.5. Використання Vertex AI Gemini API

Для реалізації логіки рекомендацій танцювальних стилів на основі відповідей користувача в проєкті використовується **Vertex AI Gemini API** від Google. Це багатофункціональний інтерфейс, який дозволяє інтегрувати моделі нового покоління безпосередньо в бекенд застосунку. Модель використовується для того, щоб зрозуміти індивідуальні уподобання користувача і підібрати саме ті стилі, які йому найкраще підійдуть. Основна мета – допомогти новачкам зробити перший крок у світ танців, не перевантажуючи їх зайвою складністю.

Інтеграція з Gemini API побудована через Spring AI та клас ChatClient, який конфігурується через Builder, наданий бібліотекою. Усе інкапсульовано в сервісному класі AIService, який реалізує інтерфейс IAIService (рис. 4.12).

```java
@Service
@Slf4j
public class AIService implements IAIService {
    2 usages
    private final IQuestionRepository questionRepository;
    2 usages
    private final ChatClient chatClient;

    new *
    public AIService(ChatClient.Builder builder, IQuestionRepository questionRepository) {
        this.chatClient = builder.build();
        this.questionRepository = questionRepository;
    }

    1 usage  new *
    @Override
    public List<DancingStyle> getResult(Map<Long, String> answers) {
        List<QuestionEntity> questions = questionRepository.findAllById(answers.keySet());

        String prompt = buildPrompt(questions, answers);

        String result = chatClient
                .prompt(prompt) ChatClientRequestSpec
                .call() CallResponseSpec
                .content();

        assert result != null;

        return Stream.of(result.split( regex: ",")) Stream<String>
                .map(String::trim)
                .map(styleName -> {
                    try {
                        return DancingStyle.valueOf(styleName);
                    } catch (IllegalArgumentException e) {
                        log.warn("Unknown dancing style from AI result: {}", styleName);
                        return null;
                    }
                }) Stream<DancingStyle>
                .filter(Objects::nonNull)
                .toList();
    }
```

Рисунок 4.12 Клас AIService

Основний метод getResult(Map<Long, String> answers) приймає мапу відповідей користувача (ідентифікатор питання та відповідь), витягує відповідні сутності QuestionEntity з бази, і на основі цього будує текстовий промпт. Промпт складається з загального опису задачі, списку доступних танцювальних, а також – усіх запитань з відповідями користувача у впорядкованому вигляді.

Сформований запит передається до моделі через chatClient.prompt(prompt).call().content(). У відповідь модель повертає простий рядок із трьома назвами танцювальних стилів у верхньому регістрі, розділеними комами (наприклад, BACHATA, CONTEMPORARY, TANGO).

Таким чином, система поєднує класичний CRUD-інтерфейс з БД (для зберігання питань) із генеративним ШІ, створюючи адаптивну рекомендаційну систему, яка може гнучко реагувати на індивідуальні особливості користувача.

Інтеграція з Vertex AI дозволила уникнути громіздких алгоритмів відповідності або навчання власної ML-моделі. Усе, що потрібно – це якісно сформульований prompt і відповідна конфігурація ChatClient.

4.6. Висновки до розділу 4

У розділі було детально розглянуто серверну частину вебзастосунку, основні технології, що використовуються, та принципи роботи з ними.

Зокрема, безпека забезпечується інтеграцією Spring Security та JWT, що гарантує надійний захист доступу до ресурсів. Для зберігання медіаконтенту використовується MinIO – об’єктне сховище, сумісне з API Amazon S3, розгорнуте у Docker-контейнері, що забезпечує надійність і зручність управління файлами користувачів.

Окремо наведено приклад використання бібліотеки Lombok для оптимізації коду, а також описано роль Docker у розгортанні та взаємодії сервісів.

Для реалізації рекомендаційної системи танцювальних стилів було впроваджено Vertex AI Gemini API у поєднанні зі Spring AI та ChatClient. Цей підхід виключив необхідність створення та навчання власної моделі машинного навчання, одночасно підвищуючи якість рекомендацій завдяки використанню передових можливостей штучного інтелекту.

Загалом, серверна частина вебзастосунку побудована з урахуванням сучасних підходів до розробки, що забезпечує гнучкість, безпеку та масштабованість платформи.

РОЗДІЛ 5. ОГЛЯД КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ

5.1. Особливості реалізації

Використання React для клієнтської частини вебзастосунку дало можливість просто та зручно створювати незалежні компоненти інтерфейсу та багаторазово їх перевикористовувати.

Візьмемо для прикладу компонент Pagination (рис. 5.1).

```
interface PaginationProps {
    currentPage: number;
    totalPages: number;
    onPageChange: (pageNumber: number) => void;
}

8+ usages  Yevheniia Matsevko
const Pagination: React.FC<PaginationProps> = ({currentPage : number , totalPages : number , onPageChange}) => {
    3 usages  Yevheniia Matsevko
    const handlePageChange = (pageNumber: number) : void  => {
        onPageChange(pageNumber);
    };

    1 usage  Yevheniia Matsevko
    const getPageNumbers = () => {
        const halfMaxPages : number  = Math.floor( x: MAX_PAGE_NUMBERS_FOR_PAGINATION / 2);
        let startPage : number  = Math.max( values: 0, currentPage - halfMaxPages);
        let endPage : number  = Math.min( values: totalPages - 1, currentPage + halfMaxPages);
```

Рисунок 5.1 Компонент Pagination

Компонент повністю універсальний – його можна інтегрувати з будь-яким джерелом даних. За допомогою параметрів currentPage, totalPages та функції onPageChange ми можемо легко керувати його станом та синхронізувати з логікою інших частин інтерфейсу.

У реалізованому вебзастосунку даний компонент використано понад 8 разів у різних модулях: для навігації по списках подій, курсів, шкіл, викладачів, реєстрацій тощо.

Використання такого компонентного підходу дозволяє уникати дублювання коду, спрощує тестування системи та подальше масштабування.

Дотримуючись цього ж принципу, було реалізовано компонент CalendarView для відображення розкладу користувача (рис.5.2).

```
import FullCalendar from '@fullcalendar/react';
import dayGridPlugin from '@fullcalendar/daygrid'
import timeGridPlugin from '@fullcalendar/timegrid'
import interactionPlugin from '@fullcalendar/interaction'

1 usage  Yevhenia Matsevko
interface CalendarViewProps {
    customButtons?: any;
    headerToolBar: any;
    displayedCalendarEvents: EventInput[];
    eventClick: (info: any) => void;
}

5+ usages  Yevhenia Matsevko *
const CalendarView: React.FC<CalendarViewProps> = ({customButtons, headerToolBar, displayedCalendarEvents : EventInput[] , eventClick}) => {

    return (
        <FullCalendar
            plugins={[dayGridPlugin, timeGridPlugin, interactionPlugin]}
            initialView="dayGridMonth"
            customButtons={customButtons}
            headerToolbar={headerToolBar}
            events={displayedCalendarEvents}
            eventClick={eventClick}
            editable={true}
            selectable={true}
            eventTimeFormat={{
                hour: '2-digit',
                minute: '2-digit',
                meridiem: false,
                hour12: false
            }}
            slotLabelFormat={{
                hour: '2-digit',
                minute: '2-digit',
                meridiem: false,
                hour12: false
            }}
            displayEventTime={false}
        />
    );
};
```

Рисунок 5.2 Компонент CalendarView

За допомогою бібліотеки @fullcalendar вдалося швидко додати інтерактивний календар з можливістю перегляду подій у форматі місяця, тижня або дня. Компонент дозволяє користувачу бачити свої майбутні заняття, а також легко орієнтуватися у розкладі.

Завдяки модульності бібліотеки, використовувались лише необхідні плагіни, такі як dayGrid, timeGrid та interaction, що дозволило зменшити загальну вагу застосунку.

Компонент є багаторазовим і повністю адаптованим до змін в даних, що надходять з бекенду, завдяки чому його можна легко інтегрувати в будь-яку частину вебзастосунку, пов’язану з плануванням або переглядом розкладу.

В окремі TypeScript-файли були винесені основні користувацькі типи даних, що використовуються у застосунку. Такий підхід допомагає структурувати код, уникати помилок при роботі з даними та спрощує автодоповнення у редакторі коду, що загалом пришвидшує розробку.

5.2. Забезпечення доступності

Цифрова доступність – можливість використання цифрових сервісів якомога більшою кількістю людей, включно з людьми з інвалідністю [22].

Основним міжнародним стандартом, що визначає вимоги до доступності, є **WCAG** (*Web Content Accessibility Guidelines*). Актуальна версія – WCAG 2.2, включає принципи [23]:

- **Perceivable** (доступний для сприйняття): наприклад, додавання текстових альтернатив до зображень.
- **Operable** (керований): забезпечення керованості інтерфейсу за допомогою клавіатури.
- **Understandable** (зрозумілий): зрозуміла мова, інструкції та текст помилок.
- **Robust** (надійний): коректна робота з допоміжними технологіями.

У вебзастосунку було реалізовано базові принципи доступності.

По-перше, вся HTML-структура побудована з дотриманням правильної семантики: використовуються теги <header>, <main>, <footer>, <nav>. Це допомагає користувачам із читачами екрана краще орієнтуватися на сторінці. На рисунку надано приклад використання header і nav у проєкті (рис.5.3).

Рисунок 5.3 Компонент Header

Таким чином, контент логічно групується, а заголовки мають чітку ієрархію без пропущених рівнів, що дозволяє screen reader'ам легко пересуватись між розділами [23].

Для додаткової підтримки користувачів з порушеннями зору були застосовані ARIA-атрибути. Наприклад, aria-label надає опис кнопкам без тексту, aria-describedby вказує на допоміжні підказки, а aria-live="polite" інформує читача екрана про динамічні зміни вмісту, такі як повідомлення про помилки. Також було використано атрибути aria-invalid, aria-haspopup, role="alert" – усе для того, щоб користувачі отримували чіткі та зрозумілі сигнали про зміни чи помилки.

Застосунок було протестовано як вручну, за допомогою NVDA (NonVisual Desktop Access), так і автоматизовано через Lighthouse Accessibility Report. Під час тестування перевірялася навігація виключно з клавіатури (Tab, Enter, Esc), видимість фокусу, правильна передача повідомлень про помилки, контрастність елементів, масштабування інтерфейсу до 200% без втрат читабельності, а також наявність альтернативних текстів для зображень і кнопок. Було досягнуто відповідність вимогам доступності рівнів A та AA.

Окрім технічних аспектів, платформа має функціональну підтримку інклюзивності. Користувачі можуть швидко шукати заняття, адаптовані до конкретних потреб – наприклад, танцювальні програми для людей із

порушеннями слуху, моторики чи зору. Цей підхід робить платформу по-справжньому доступною не лише за стандартами, а й за відчуттям користувачів – коли вони можуть легко знайти щось, що підходить саме їм.

5.3. Огляд користувацького інтерфейсу

Перейдімо до огляду користувацького інтерфейсу.

На рисунку ви бачите початкову сторінку вебплатформи (рис. 5.4).

Рисунок 5.4 Початкова сторінка вебплатформи «DanceSpace»

У верхній частині сторінки знаходиться панель навігації. Вона пропонує:

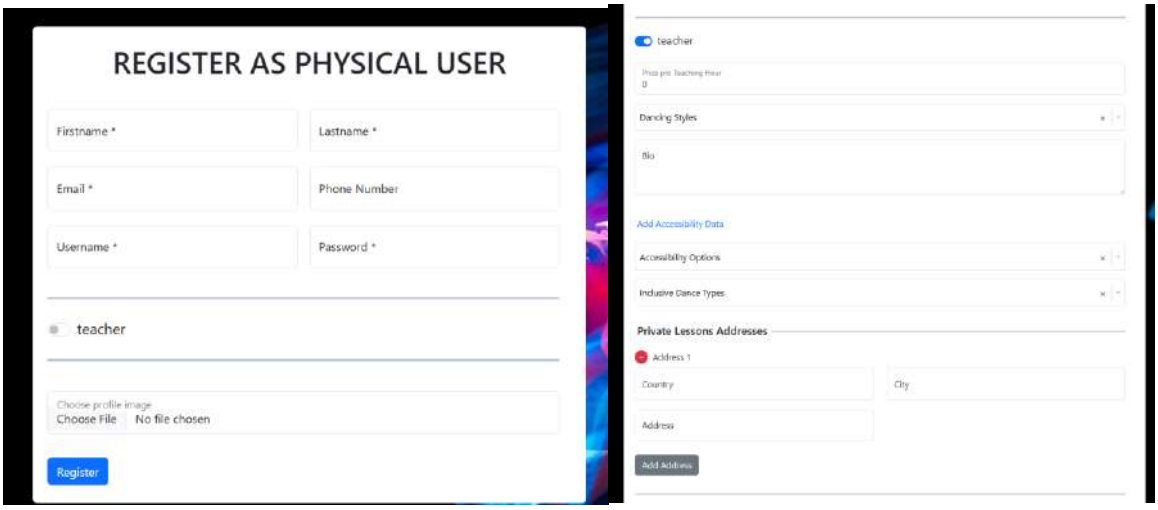
- Проходження тесту на визначення танцювального стилю.
- Меню пошуку подій, шкіл, курсів та викладачів.
- Можливість організовувати танцювальні ініціативи.
- Можливість авторизації/реєстрації користувача.

Також на початковій сторінці ми бачимо форму входу у систему.

Розглянемо детальніше процес реєстрації. При натисканні на кнопку Sign Up, користувачу пропонується вибір між реєстрацією як танцювальна школа та фізичний користувач (рис. 5.5).

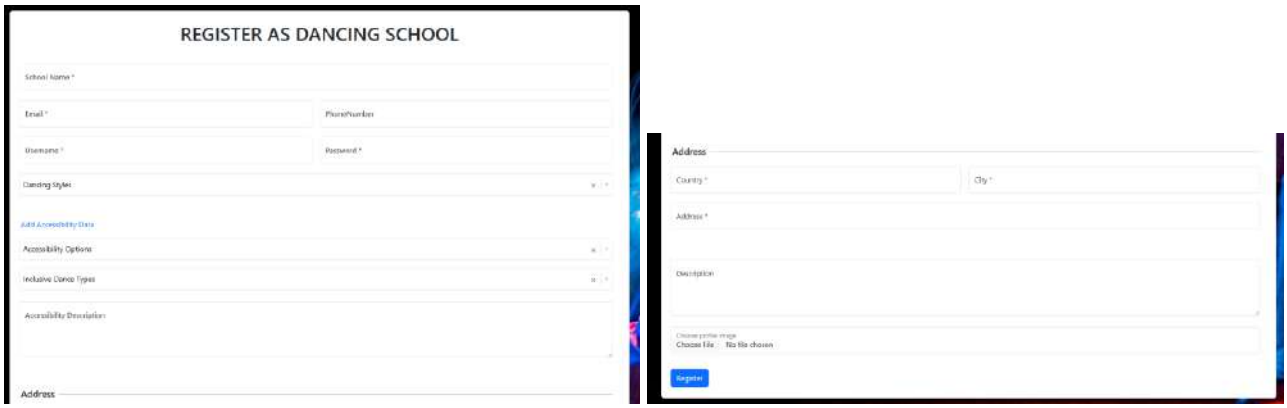
Рисунок 5.5 Вибір реєстрації як фізичний користувач чи танцювальна школа

Форма реєстрації фізичного користувача окрім звичайних полів вводу, включає також можливість реєстрації як викладач. Тоді форма розширюється та пропонує заповнити додаткові поля, такі як вартість години приватного заняття, танцювальні стилі, в яких спеціалізується користувач, короткий опис, також є можливість додавання адрес надання приватних уроків та інформації про доступність (рис. 5.6, 5.7).



Рисунки 5.6, 5.7 Форма реєстрації фізичного користувача із розширенням при реєстрації викладача

Форма реєстрації танцювальної школи містить поля для вводу загальної інформації про неї, даних про доступність, вибору танцювальних стилів та завантаження фото профілю (рис. 5.8, 5.9).



Рисунки 5.8, 5.9 Форма реєстрації танцювальної школи

Крім того, незареєстрованому користувачу доступне меню подій, курсів, шкіл та викладачів із можливістю розширеного пошуку. Особливу увагу варто звернути на фільтри щодо доступності (рис. 5.10).

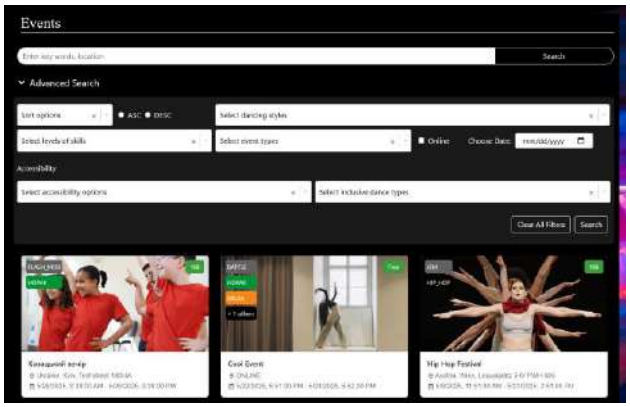


Рисунок 5.10 Сторінка пошуку події із панелю розширеного пошуку

Розглянемо сторінку окремої події (рис. 5.11).

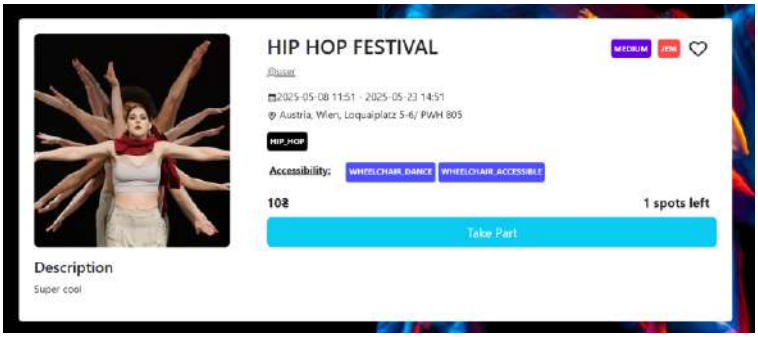


Рисунок 5.11 Сторінка події

У зареєстрованого користувача є можливість реєстрації на подію при наявності вільних місць та додавання події у обрані.

Такий ж функціонал поширюється на приватні курси, організовані викладачами. Курси танцювальних шкіл в свою чергу реєстрації не передбачають і використовуються лише для інформування.

Також для всіх типів користувачів доступна функція проходження тесту на визначення танцювального стилю, що їм найкраще підходить (рис. 5.12).

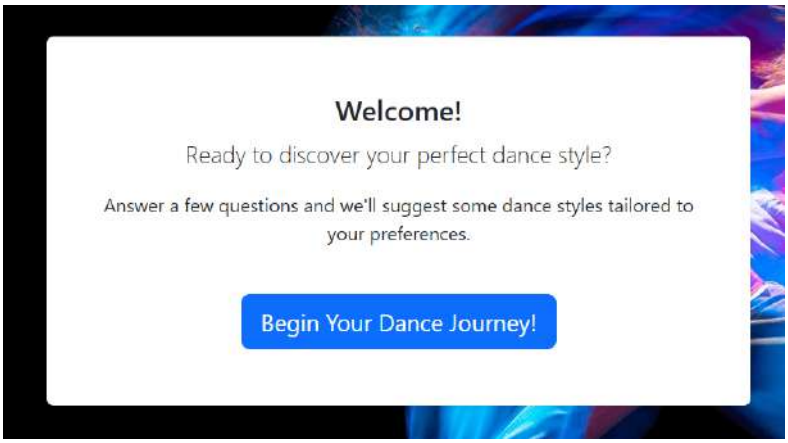


Рисунок 5.12 Початкова сторінка тесту на визначення танцювального стилю

Даний функціонал розрахований на початківців, які ще не визначились, чим саме хочуть займатись. Система пропонує 25 запитань з різними типами відповіді та вкінці рекомендує три танцювальні стилі (рис.5.13).

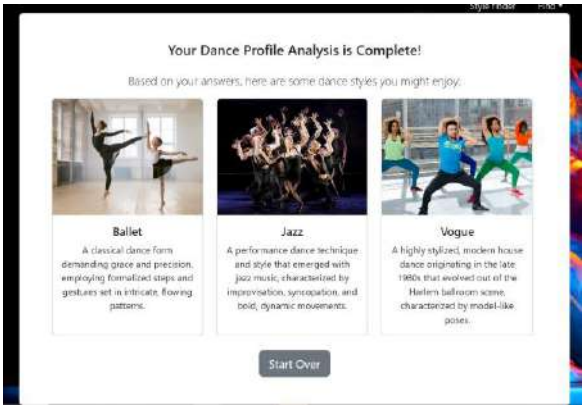


Рисунок 5.13 Сторінка з результатами тесту на визначення танцювального стилю

Перейдімо на сторінку танцювальної школи (рис. 5.14).

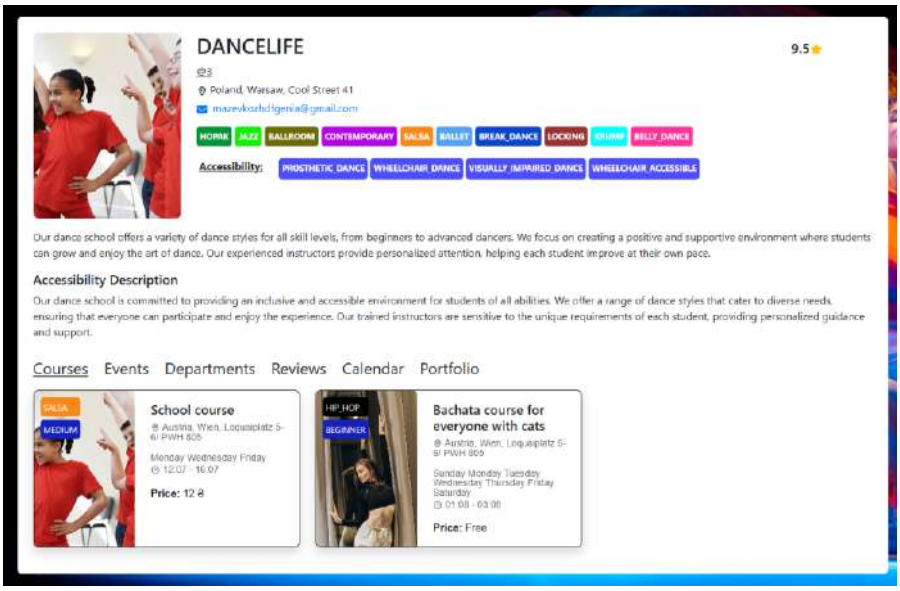


Рисунок 5.14 Сторінка танцювальної школи

Вона містить 6 підвкладок, а саме:

- Перелік організованих школою курсів
- Перелік організованих подій
- Список відділень школи
- Відгуки про школу з можливістю додавати новий для зареєстрованого користувача
- Календар (рис. 5.15)

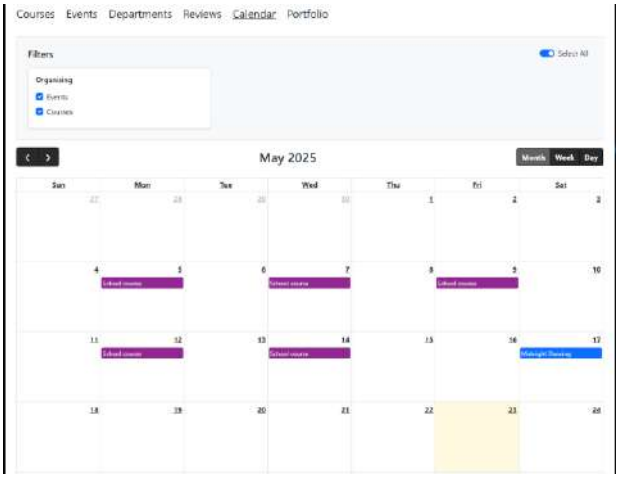


Рисунок 5.15 Календар танцювальної школи

Користувачу надається форма з фільтрами, які саме типи активностей показувати, та сам календар, що має три режими відображення даних: місяць, тиждень та день. Натискання на подію або курс відкриє їхню сторінку у новій вкладці.

- Портфоліо з відео контентом школи відображене з використанням каруселі (рис. 5.16).

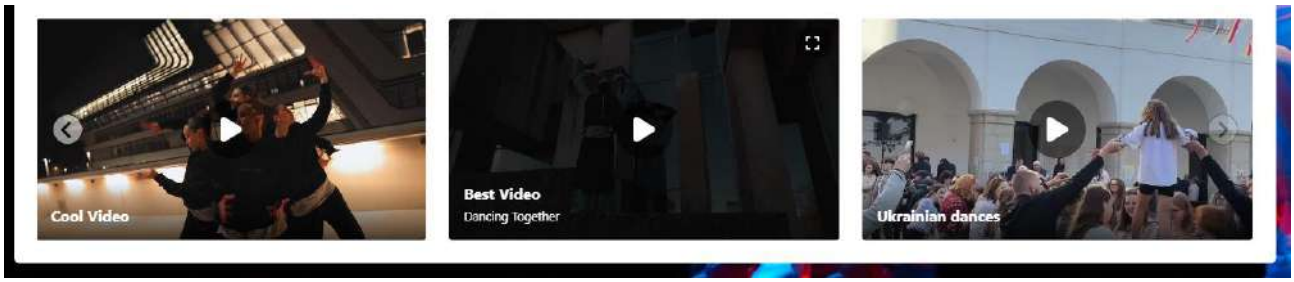


Рисунок 5.16 Портфоліо

Сторінка викладача містить теж список організованих подій та приватних курсів, портфоліо та календар. Однак у ньому, на відміну від школи ще відображені спільні приватні уроки для зареєстрованих користувачів та вільні таймслоти добавлені викладачем (рис. 5.17).

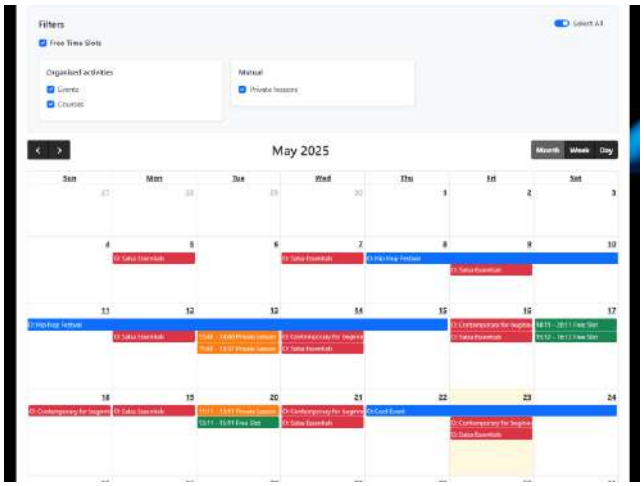


Рисунок 5.17 Календар викладача

На них зареєстровані користувачі можуть подавати заявки на приватні уроки.

Ці заявки зберігаються в розділі Приватні уроки (рис. 5.18).

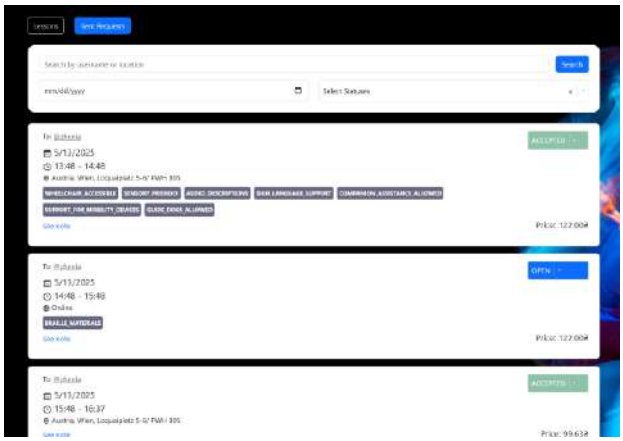


Рисунок 5.18 Сторінка приватних уроків та запитів на них

Також тут є зручна форма пошуку. Звичайним користувачам доступна можливість скасування запиту на приватний урок, а викладач може або приймати, або відхиляти запити.

Крім того є сторінка з власними реєстраціями та вподобаними активностями.

Зареєстровані користувачі можуть організовувати події, викладачі – події та приватні курси та танцювальні школи – події та звичайні курси.

При організації також вказується інформація щодо доступності (рис. 5.19)

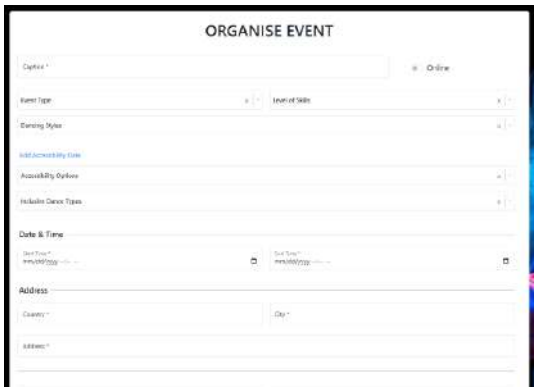


Рисунок 5.19 Форма реєстрації події

Для танцювальних шкіл доступний функціонал додавання нових відділень та прив’язки курсів до них.

Таким чином, було реалізовано зручний та сучасний користувацький інтерфейс, який поєднує естетичний вигляд із високим рівнем доступності та зручності, забезпечуючи позитивний користувацький досвід та легкість взаємодії з системою.

5.4. Висновки до розділу 5

У розділі 5 було розглянуто ключові аспекти реалізації клієнтської частини вебзастосунку з акцентом на зручність і модульність інтерфейсу. Використання React як основної бібліотеки дало змогу створювати багаторазові компоненти, які значно підвищують ефективність розробки та полегшують її підтримку, а використання бібліотеки @fullcalendar дозволило швидко інтегрувати інтерактивний та адаптивний календар, який забезпечує користувачам комфортний перегляд розкладу у різних форматах.

Імплементація базових принципів цифрової доступності відповідно до стандарту WCAG 2.2 забезпечила зручну та інтуїтивно зрозумілу взаємодію з вебплатформою для користувачів з різними можливостями. Результати проведеного тестування підтвердили ефективність застосованих рішень у забезпеченні доступності.

Взагальному, клієнтська частина реалізована з фокусом на зручність, гнучкість та підтримку масштабування, що гарантує позитивний користувацький досвід і відкриває можливості для подальшого розвитку функціоналу.

ВИСНОВКИ

У виконаній дипломній роботі було проведено комплексний аналіз танцювальної сфери, визначено основні проблеми та ключові потреби користувачів. Огляд існуючих рішень виявив їхні недоліки і підкреслив необхідність створення більш зручного та інклюзивного вебзастосунку.

Проектування вебплатформи грунтувалось на ретельному виборі архітектури системи та інструментів розробки, що забезпечило гнучкість, масштабованість і надійність.

У результаті було розроблено серверну частину, що надає високий рівень безпеки, ефективне зберігання медіаконтенту та функціонал розширеного пошуку. Для цього було використано Spring Security, MinIO, Docker, Vertex AI Gemini API та інші сучасні бібліотеки.

Клієнтська частина реалізована з урахуванням зручності та доступності. Впровадження принципів цифрової доступності відповідно до стандартів WCAG 2.2, а також застосування семантичної розмітки та ARIA-атрибутів гарантує інклюзивність і позитивний користувацький досвід.

Потенціал подальшого розвитку платформи полягає у розширенні функціоналу профілів користувачів із можливістю ведення блогів, додавання онлайн-трансляцій занять, а також впровадження нових функцій доступності, зокрема субтитрів для відеоуроків та інтеграції з голосовими помічниками, що значно спростить навігацію платформою та зробить її ще більш дружньою для користувачів.

Таким чином, створена вебплатформа є комплексним рішенням, що відповідає сучасним вимогам ринку та користувачів танцювальної сфери, забезпечуючи ефективну взаємодію, безпеку, доступність і відкриваючи широкі можливості для подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wikipedia. Web Content Accessibility Guidelines. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Web_Content_Accessibility_Guidelines
2. Національна бібліотека України для дітей, Міжнародний день танцю. [Електронний ресурс] – Режим доступу до ресурсу: https://chl.kiev.ua/Default.aspx?id=9595
3. Укрінформ, Безбар'єрність вважають новою цінністю суспільства – 83% українців. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ukrinform.ua/rubric-society/3876500-bezbarernist-vvazaut-novou-cinnistu-suspilstva-83-ukrainciv.html
4. The Dance Platform. [Електронний ресурс] – Режим доступу до ресурсу: https://thedanceplatform.com/
5. Dance Place. [Електронний ресурс] – Режим доступу до ресурсу: https://www.danceplace.com/
6. QATestLab. Клієнт-серверна архітектура. [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/
7. Mozilla Developer Network. REST. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/REST
8. Medium, J. Singh, 25 Reasons Why Java Is Still Around in 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/javarevisited/25-reasons-why-java-is-still-around-in-2024-452c582d55d0
9. Офіційна документація Spring Boot. [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-boot
10. Apache Maven, Introduction to the Build Lifecycle. [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

11. Amazon Web Services, The Difference Between MySQL vs. PostgreSQL. [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/
12. MinIO. MinIO: High Performance Object Storage. [Електронний ресурс] – Режим доступу до ресурсу: https://min.io/
13. SitePoint, An Introduction to TypeScript: Static Typing for the Web. [Електронний ресурс] – Режим доступу до ресурсу: https://www.sitepoint.com/introduction-to-typescript/
14. React Official Documentation, Introduction to the React Concepts. [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/docs/faq-internals.html
15. Nginx, Using Nginx as a Reverse Proxy. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/
16. Burak KOCAK, JPA, Hibernate And Spring Data JPA. [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@burakkocakeu/jpa-hibernate-and-spring-data-jpa-efa71feb82ac
17. Auth0, Introduction to JSON Web Tokens. [Електронний ресурс] – Режим доступу до ресурсу: https://auth0.com/learn/json-web-tokens/
18. Spring Security, Introduction to Spring Security. [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-security
19. Java Revisited, All the 16 Lombok Annotations Explained in a 4-Minute Article. [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/javarevisited/all-the-16-lombok-annotations-explained-in-a-4-minute-article-926f71934ec6
20. Docker Documentation, Overview of Docker Components. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/get-started/overview/

21. Docker Documentation, Use Compose to Run Multi-Container Applications. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/compose/gettingstarted/
22. UNDP Ukraine, Accessibility in Government: Guidelines and Recommendations for Inclusive Public Services. [Електронний ресурс] – Режим доступу до ресурсу: https://www.undp.org/sites/g/files/zskgke326/files/migration/ua/Accessibility_Goverment_UKR_final.pdf
23. W3C, Web Content Accessibility Guidelines (WCAG) 2.2. [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/WCAG22/
24. W3C, Accessible Rich Internet Applications (WAI-ARIA) 1.2. [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/wai-aria-1.2/