

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Платформа для бронювання та придбання квитків»

Виконав студент групи ПЗ-20-2
ВАКУЛОВИЧ Нікіта Вікторович

Керівник роботи:
ЛЕВЧЕНКО Антон Юрійович

Консультант:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Завідувач кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВАКУЛОВИЧ Нікіта Вікторович**

Керівник роботи: **ЛЕВЧЕНКО Антон Юрійович**

Тема роботи: **«Платформа для бронювання та придбання квитків»**,
затверджена Наказом закладу вищої освіти від «26» лютого 2024 р., №74с

Вихідні дані для роботи: **процес публікації та купівлі місць на заходи з аутентифікацією квитка.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Левченко А.Ю., Єфремов Ю.М.	27.02.2023р.	27.02.2023р.
2	Левченко А.Ю., Єфремов Ю.М.	27.02.2023р.	27.02.2023р.
3	Левченко А.Ю., Єфремов Ю.М.	27.02.2023р.	27.02.2023р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра включає в себе створення веб-платформа для бронювання та придбання квитків та пояснювальної записки. Пояснювальна записка до роботи налічує 69 сторінок, містить 39 ілюстрації, 35 таблиць, 33 джерела та 1 додаток.

Основною метою роботи є розробка платформи для продажу квитків, що використовує сучасні технології та шифрування для аутентифікації придбаних квитків.

Архітектура платформи для купівлі квитків базується на сучасних вебтехнологіях. Серверна частина реалізована за допомогою Node.js, що забезпечує високопродуктивний та масштабований бекенд. Фронтенд частина створена за допомогою React, що дозволяє створювати динамічні та інтерактивні інтерфейси користувача.

Платформа надає користувачам можливість легко знаходити та купувати квитки на бажані заходи, одночасно гарантуючи захист їхніх особистих даних та безпеку транзакцій.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, ПЛАТФОРМА, NODE.JS, REACT, КВИТКИ, АУТЕНТИФІКАЦІЯ, ШИФРУВАННЯ.

					ІПЗ.КР.Б-121-24-ІПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Платформа для бронювання та придбання квитків	Літ.	Арк.	Аркушів
Розроб.		Вакулович Н.В.						
Керівник		Левченко А.Ю.					3	81
Рецензент		Морозов А.В.				Житомирська політехніка, група ІПЗ-20-2		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's final qualification project includes the development of a web-based platform for booking and purchasing tickets and an explanatory note. The explanatory note contains 69 pages, 39 illustrations, 35 tables 33 sources and 1 appendix.

The primary goal of the project is to create a ticket sales platform that utilizes modern technologies and encryption for the authentication of purchased tickets.

The architecture of the ticket purchasing platform is based on modern web technologies. The server-side is implemented using Node.js, providing a high-performance and scalable backend. The frontend is created using React, enabling the creation of dynamic and interactive user interfaces.

The platform allows users to easily find and purchase tickets for desired events while ensuring the protection of their personal data and the security of transactions.

KEYWORDS: WEB APPLICATION, PLATFORM, NODE.JS, REACT, TICKETS, AUTHENTICATION, ENCRYPTION.

					ИП3.КР.Б-121-24-П3	
						4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ БРОНЮВАННЯ ТА ПРИДБАННЯ КВИТКІВ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	17
Висновки до першого розділу.	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ ПЛАТФОРМИ ДЛЯ БРОНЮВАННЯ ТА ПРИДБАННЯ КВИТКІВ	21
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	21
2.2 Розробка бази даних системи.....	34
2.3 Проектування та реалізація алгоритмів роботи системи	40
2.4 Реалізація функціоналу платформи для бронювання та придбання квитків	43
Висновки до другого розділу	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ.....	50
3.1 Порядок встановлення та налаштування платформи.....	50
3.2 Структура інтерфейсу веб-платформи	51
3.3 Тестування роботи веб-платформи	62
Висновки до третього розділу	64
ВИСНОВКИ	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	67
ДОДАТКИ.....	70
Додаток А.....	71

					ІПЗ.КР.Б-121-24-ПЗ	
						5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

MVC – схема розділення даних додатків і керуючої логіки на три окремі компоненти Model-View-Controller

CRUD – основні операції, що використовуються для управління даними (Create , Read , Update , Delete)

ORM – Об'єктно-реляційне відображення

					ІПЗ.КР.Б-121-24-ПЗ	
						6

ВСТУП

Актуальність теми полягає у тому, що сучасні цифрові технології істотно змінили способи організації та участі у заходах. Зростання популярності онлайн-покупок, потреба у безпеці та захисті даних, а також необхідність автоматизації процесів роблять створення платформи для бронювання та придбання квитків надзвичайно важливою. Така платформа забезпечує зручність і доступність для користувачів, дозволяє автоматизувати обробку замовлень та інтеграцію з платіжними системами, що зменшує навантаження на організаторів заходів. Окрім цього, збільшення кількості заходів та підвищення вимог до якості обслуговування створюють додатковий попит на ефективні рішення для продажу квитків. Таким чином, розробка цієї платформи відповідає сучасним вимогам ринку та сприяє розвитку індустрії розваг та культури.

Метою випускної кваліфікаційної роботи бакалавра є розробка сучасної платформи для продажу квитків на заходи, яка забезпечує максимальну зручність користувачам та високий рівень безпеки. Платформа має використовувати технології для аутентифікації придбаних квитків, інтегруватися з платіжними системами та мати інтуїтивно зрозумілий інтерфейс. Це дасть змогу користувачам швидко знаходити та купувати квитки, а організаторам заходів – автоматизувати процеси продажу і управління квитками, одночасно забезпечуючи захист особистих даних та безпеку фінансових транзакцій.

Завданням випускної кваліфікаційної роботи є комплексна розробка сучасної платформи для продажу квитків на заходи. Спочатку необхідно провести аналіз аналогічних продуктів та визначити вимоги користувачів. Далі слід проектувати архітектуру платформи, створити інтуїтивний інтерфейс і визначити технологічний стек. Важливим етапом є розробка модулів аутентифікації, пошуку, купівлі квитків та інтеграції з платіжними системами, з особливою увагою до безпеки даних. Платформу необхідно протестувати на продуктивність і стійкість до навантажень.

Об’єктом дослідження є розробка ефективних та безпечних механізмів продажу квитків через інтернет, використанні технологій для захисту особистих даних та операційних процесів, а також забезпеченні зрозумілого інтерфейсу для користувачів.

Предметом дослідження предметом дослідження є розробка та реалізація платформи для бронювання та придбання квитків на заходи, зокрема концерти, театральні вистави, спортивні події та конференції. Розробка механізмів продажу квитків через інтернет, використанні передових технологій для захисту особистих даних та операційних процесів.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ БРОНЮВАННЯ ТА ПРИДБАННЯ КВИТКІВ

1.1 Постановка задачі

Метою кваліфікаційної роботи бакалавра є розробка платформи для бронювання та придбання квитків на заходи, використовуючи сучасні технології, яка значно полегшить процес продажу квитків для організаторів та закладів, а користувачам надасть можливість зручно та безпечно купувати квитки на цікаві їм заходи.

Онлайн-продаж квитків дозволяє користувачам купувати квитки на події, такі як концерти, спортивні змагання, театральні вистави та інші заходи, безпосередньо через Інтернет. Це дуже зручно, оскільки користувачі можуть здійснити покупку в будь-який зручний для них момент, використовуючи комп'ютер або мобільний пристрій.

Онлайн-платформи зазвичай пропонують широкий вибір подій різного типу та різноманітні способи оплати, такі як кредитні картки, електронні платіжні системи та інші. Після завершення оплати користувач отримує квиток у електронному вигляді або може завантажити його на свій мобільний пристрій, що робить процес покупки максимально зручним та ефективним.

Крім того, онлайн-продаж квитків дозволяє користувачам швидко порівнювати ціни та вибирати найкращий варіант для себе, а також отримувати додаткову інформацію про подію, таку як розклад, місце проведення та інше, що допомагає приймати правильні рішення

платформ для бронювання та придбання квитків має представляти собою декілька програмних модулів, що забезпечують різноманітні функції та можливості для заходів та учасників:

- Модуль пошуку та перегляду. Модуль надає можливість користувачам переглядати та шукати заходи, використовуючи пошукові фільтри та сортування. Також отримувати детальну інформацію про захід та заклад, в якому він відбудуватиметься.

- Модуль транзакцій. Модуль надає авторизованим користувачам засоби для придбання квитків на заходи. Він забезпечує безпеку фінансових транзакцій та захист від шахрайства.
- Модуль повідомлень. Модуль надає можливість користувачам отримувати вчасно важливо інформацію про події з їх особистим акаунтом та оновлення стану оплати квитка
- Модуль адміністрування. Модуль забезпечує адміністрування заходів та закладів. Надає можливість редагувати, додавати, видаляти та підтверджувати публікацію заходів.
- Модуль організатора. Модуль надає можливість організаторам надавати заклади до публікації, переглядати їх статус та заповнювати інформацію про них.

В результаті повинно бути створено вебсайт на якому будь-який бажаючий може придбати квиток на захід. Також важливою складовою є можливість опублікувати свій захід, для організатора.

Основними етапами розробки платформи є:

1. Розробка дизайну вебсайту:
 - розробка загальної концепції дизайну застосунку;
 - створення компонентів інтерфейсу користувача;
 - проектування макетів сторінок;
2. Побудова архітектури клієнтської частини додатку:
 - Встановлення основної структури клієнтського коду;
 - Організація файлів, бібліотек і фреймворків на клієнтській стороні;
 - Організація спілкування серверної частини з клієнтською;
3. Побудова архітектури серверної частини додатку:
 - Налаштування сервера та робочого середовища;
 - Розробка серверної логіки та обробки запитів;
4. Розробка системи авторизації та ролей користувачів.
 - Створення механізму аутентифікації користувачів.

- Налаштування системи управління правами доступу до функціоналу.
5. Розробка основного функціоналу:
- Налаштування системи управління правами доступу до функціоналу.
 - імплементація бекенду на клієнтській стороні;
 - Створення та підтримка бази даних для зберігання інформації.
6. Тестування додатку:
- Проведення функціонального тестування окремих частин додатку.
 - тестування графічного інтерфейсу.

Результатом виконання плану дій є створення платформи, за допомогою якої організатори можуть створювати відповідні заходи, які після підтвердження адміністраторами будуть відображатись на сайті, що дозволить будь-якому авторизованому користувачу придбати квиток на даний захід.

1.2 Аналіз аналогів програмного продукту

Створення платформи для продажу квитків потребує уважного вивчення конкурентного середовища. Ми проведемо аналіз популярних платформ цього типу, оцінюючи їх функціонал, зручність використання, рівень безпеки та застосовані технології. Це дозволить нам зрозуміти потреби ринку та визначити найкращі практики для успішного впровадження нашої платформи. Одним з таких додатків є платформа Ticketmaster (рис. 1.1).

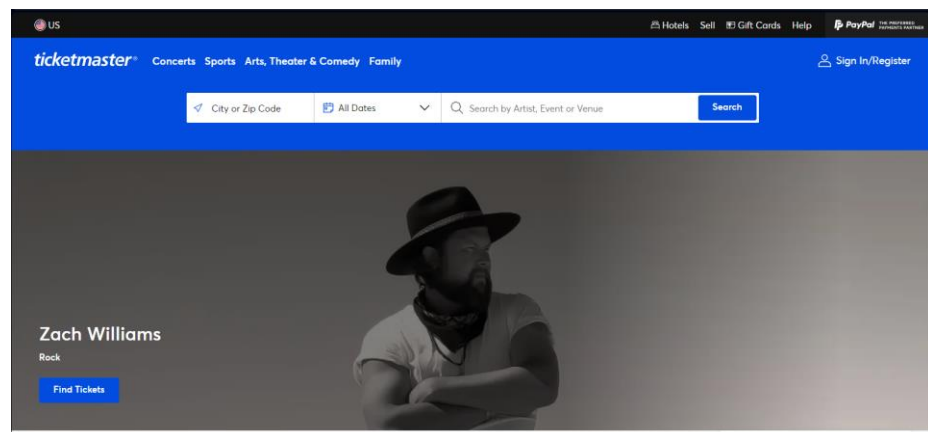


Рис. 1.1. Платформа продажу квитків Ticketmaster^[32]

					ІПЗ.КР.Б-121-24-ПЗ	
						11

Одна з найпопулярніших платформ для придбання квитків на різноманітні події, від концертів до спортивних заходів. Вона відома своєю широкою базою подій та надійністю обслуговування..

Основними перевагами Ticketmaster є:

- Ергономічний дизайн;
- Інтуїтивність дизайну;
- наявність мобільних додатків IOS та Android;
- існує декілька регіонів, зі своїми заходами;
- Висока надійність

Недоліками є:

- Завантаження сторінок ;
- Недоліки мобільної версії.
- Система перепродажу

Eventbrite - популярна платформа, яка надає послуги для організації та продажу квитків на події. Вона відома своєю простотою використання та гнучкістю для організаторів.

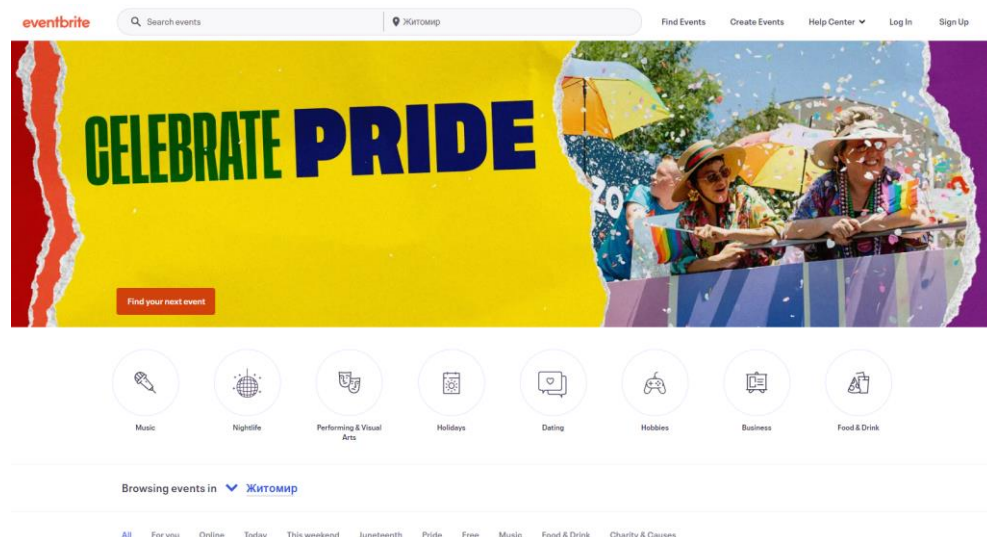


Рис. 1.2. Платформа продажу квитків Eventbrite[31]

Основними перевагами Eventbrite є:

- Мінімалістичний дизайн
- Гнучкість для організаторів

Недоліками є:

- Відсутність гарантії автентичності квитків
- Специфічний мобільний інтерфейс
- Ризик порушення безпеки даних

StubHub - платформа, що спеціалізується на продажі квитків на спортивні та розважальні події. Вона відома своєю великою базою подій та можливістю встановлення власних цін продавцями.

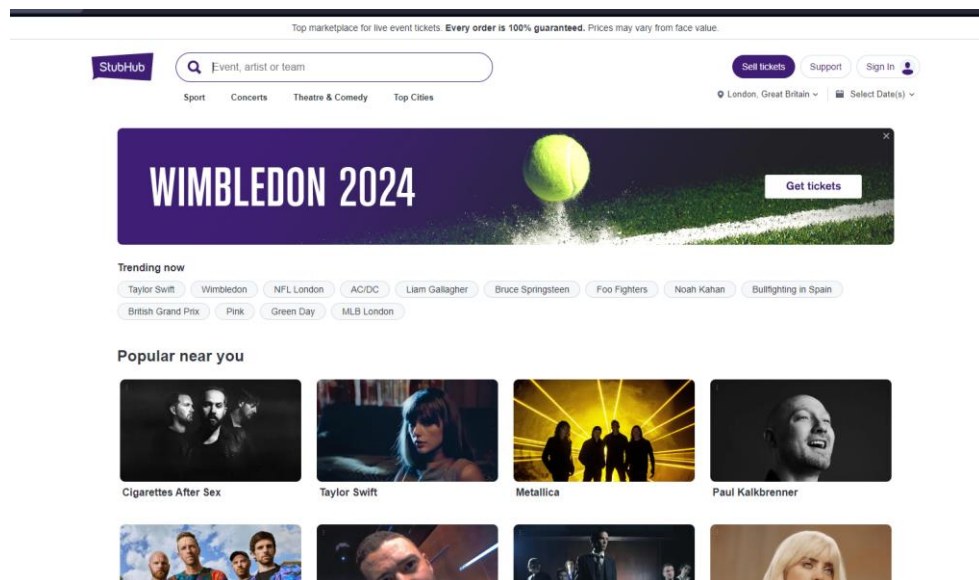


Рис. 1.3. Платформа продажу квитків StubHub^[33]

Основні переваги StubHub :

- Сучасний дизайн
- Швидкий інтерфейс

Основні недоліки StubHub

- перенавантажений інтерфейс
- Проблеми в системі оплати квитків

Karabas.com – це найбільша база заходів і анонсів подій в Україні. На сайті реалізовані сервіси e-commerce, є можливість продажу додаткових товарів. Сайт підключений до найбільших платіжних систем. Всі заходи на сайті супроводжуються докладними описами, фото - і відеоматеріалами, планами, описами залів та їх адресами.

Відвідувач сайту може обирати заходи по жанрам, містам і датам проведення, і заробляти бонуси.

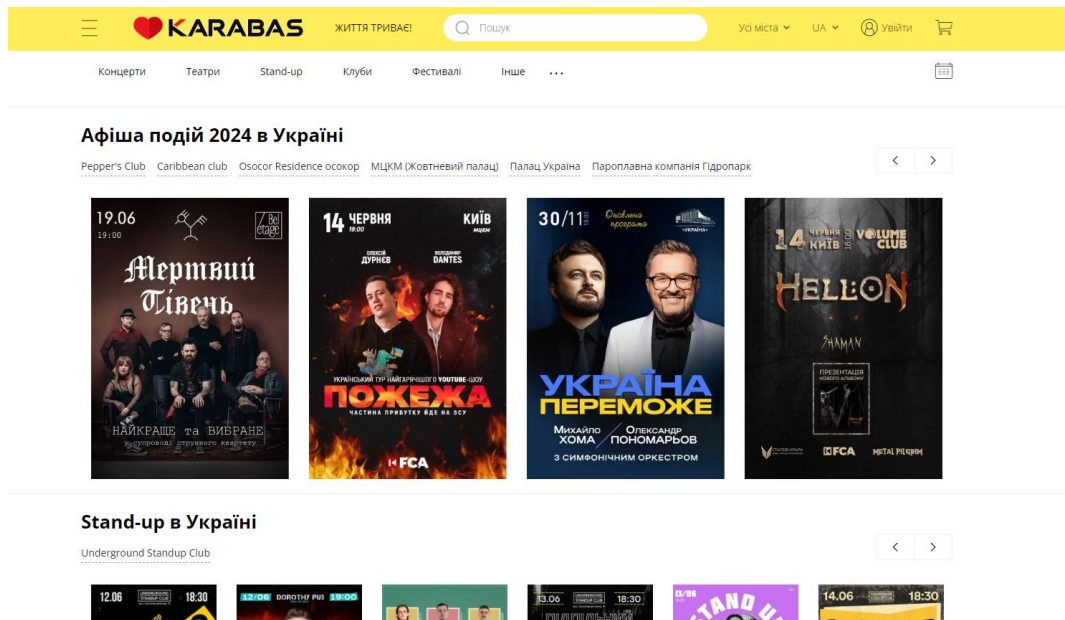


Рис. 1.4. Платформа продажу квитків karabas^[1]

Основні переваги karabas:

- швидка робота сайту;
- наявність застосунків для мобільних платформ;
- зручний пошук з фільтрацією;

Основні недоліки karabas:

- погана система повернення коштів;
- зміна інформації про захід за декілька хвилин до його початку;
- продаж вже проданих місць;
- застарілий дизайн;
- складна авторизація.

Kontramarka - провідний квитковий оператор, який використовує в основі своєї роботи унікальну систему Maestro Ticket System. Компанія працює, використовуючи найсучасніші технології, надаючи своїм клієнтам усі переваги квиткових сервісів і усуваючи всі недоліки в цій сфері. Найнадійніша система продажу квитків онлайн завдяки тому, що компанія не є розповсюджувачем або посередником, а пропонує квитки безпосередньо.

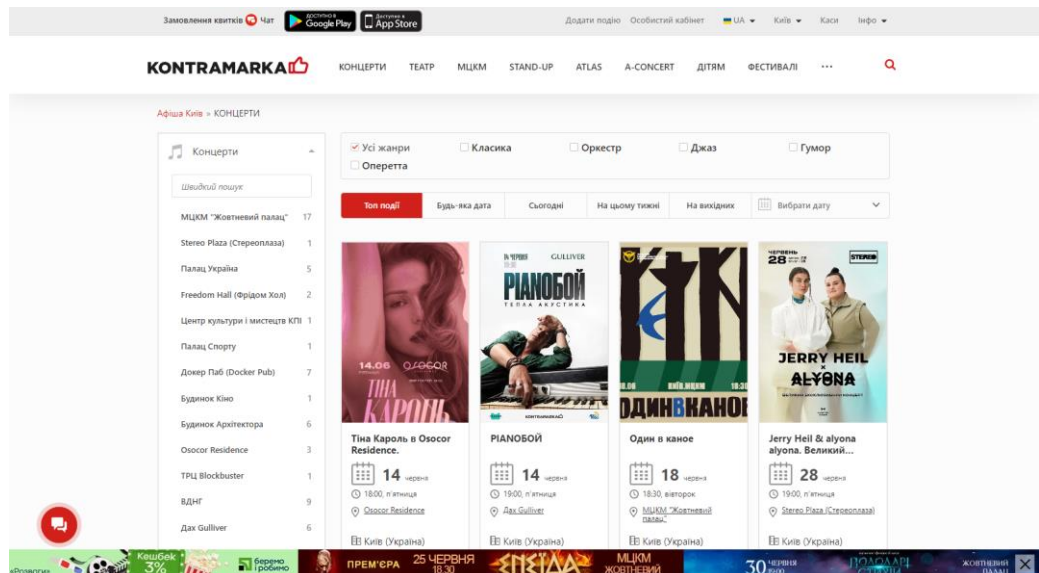


Рис. 1.5. Платформа продажу квитків Kontramarka

Основними перевагами Kontramarka є:

- інтуїтивно зрозуміла система у користуванні;
- вебдодаткок підтримує різні мови в тому числі і українську;
- можна оформити підписку на автора і підтримувати його на постійній основі;
- проста авторизація;
- наявність мобільних додатків IOS та Android;
- недосконала адаптивність сайту;

Недоліками є:

- погана система повернення коштів;
- перенавантажений інтефрейс сайту

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця порівняння платформ та продажу квитків

Характеристика	Ticketmaster	Eventbrite	StubHub	Karabas	Kontramarka
Адаптивність	+	+	-	+	-
Вчасне повернення коштів	+	+	-	-	+
Перенавантажений інтерфейс	-	-	+	+	+
складна авторизація	-	-	-	+	+
Проблема перепродажу квитків	+	+	+	+	+

Після аналізу різних платформ для продажу квитків було виявлено декілька важливих проблем, які часто виникають під час їхнього використання. По-перше, проблема перепродажу квитків становить серйозний виклик для організаторів заходів, оскільки важко гарантувати, що квиток придбаний на платформі не буде перепроданий з метою заробітку. По-друге, алгоритми повернення коштів у багатьох сервісах працюють лише за індивідуальним запитом користувача, а не автоматично після скасування заходу, що може створити труднощі для користувачів при скасуванні великих подій.

Ще однією проблемою є недостатній контроль за використанням квитків. Відсутність ефективної системи перевірки ідентифікації покупця на вході може дозволити багаторазове використання одного квитка або його перепродаж після використання. У відповідь на ці виклики було вирішено створити платформу, оснащену вбудованими інструментами та засобами для легкого впровадження на місці організаторами, які допоможуть ідентифікувати власника квитка за

допомогою унікального коду, а також забезпечать ефективний механізм повернення коштів.

1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вебдодаток буде побудований на клієнт-серверній архітектурі. Серверна частина буде представлена у вигляді API та буде написана з використанням сучасної платформи Node.js. Для клієнтської частини буде використано бібліотеку React.

Node.js[\[3\]](#) - це середовище виконання JavaScript, побудоване на движку V8 (той самий, що використовується в браузері Google Chrome), яке дозволяє виконувати JavaScript код на сервері. Він працює на однопотоковій архітектурі з подіями (event-driven), що робить його особливо ефективним для обробки великого обсягу одночасних з'єднань з мережею.

Переваги Node.js:

- Non-blocking I/O (введення-виведення): Node.js працює на основі асинхронного програмування, що означає, що він може продовжувати виконання коду під час чекання на завершення операцій введення-виведення. Це забезпечує високу ефективність та швидкодію, особливо у випадку мережевих операцій.
- Висока масштабованість: Node.js добре підходить для роботи з великою кількістю одночасних підключень, так як він здатен обробляти асинхронні запити без затримок. Це робить його ідеальним вибором для розробки високонавантажених додатків, таких як чати, стрімінгові сервіси тощо.
- Спільне використання коду між клієнтом та сервером: Node.js дозволяє розробникам використовувати JavaScript як на клієнтській, так і на серверній стороні. Це спрощує розробку, оскільки однаковий код може бути використаний як на фронтенді, так і на бекенді, сприяючи уніфікації та зменшенню навантаження на розробників.

- Спрощений веб-сервер: Node.js має вбудовану бібліотеку http, яка дозволяє швидко створювати власні веб-сервери без необхідності використання сторонніх серверів, таких як Apache чи Nginx. Це дозволяє розробникам більш гнучко керувати конфігурацією та логікою сервера.

Отже, переваги Node.js охоплюють ефективну роботу з ресурсами завдяки асинхронному програмуванню, значну масштабованість для додатків з високим навантаженням, можливість використання однієї мови програмування для розробки на клієнтській та серверній частинах, а також легкість у створенні веб-серверів.

React[\[2\]](#) - це відкрита JavaScript бібліотека для створення користувацьких інтерфейсів, розроблена компанією Facebook і підтримувана великою спільнотою розробників. React використовується для створення веб-додатків з високою динамічністю та великою кількістю інтерактивних елементів.

Переваги React:

- Ефективність: React використовує віртуальний DOM для оптимізації рендерингу, що забезпечує високу продуктивність додатків навіть з великою кількістю компонентів.
- Компонентний підхід: React дозволяє будувати додатки з малими, самодостатніми компонентами, що полегшує розробку, тестування та підтримку коду.
- Односторонній потік даних: Використання одностороннього потоку даних (у напрямку від батьківських до дочірніх компонентів) спрощує управління станом додатку та робить його більш передбачуваним.
- Розширюваність: Завдяки широкому спектру сторонніх бібліотек та інструментів, які доповнюють функціональність React, розробники можуть швидко розширювати можливості своїх додатків.

Підсумовуючи, React — це потужна JavaScript бібліотека для створення інтерактивних веб-інтерфейсів. Вона забезпечує ефективне управління віртуальним DOM, використовує компонентний підхід для спрощення розробки

та підтримує односторонній потік даних, що полегшує управління станом додатків. Завдяки великій спільноті розробників та широким можливостям для розширення функціоналу, React користується великою популярністю серед інструментів для розробки веб-додатків.

PostgreSQL[\[10\]](#) - це потужна вільна та відкрита реляційна СУБД, яка зарекомендувала себе як одна з найбільш надійних, масштабованих та функціональних.

Ось деякі її переваги:

- PostgreSQL забезпечує високу надійність через вбудовану підтримку транзакцій та механізми відновлення після збоїв.
- Можливість встановлення обмежень цілісності даних (через перевірки, зовнішні ключі тощо) гарантує правильність даних у вашій базі.
- PostgreSQL може ефективно обробляти великі обсяги даних та велику кількість одночасних з'єднань, завдяки своїй оптимізації та архітектурі.
- Можливість розподілення даних (sharding) та реплікації дозволяє масштабувати базу даних зростанням обсягів даних та навантаження.
- PostgreSQL має велику кількість розширень (extensions) та модулів, які дозволяють розширити його функціонал за потреби.

Враховуючи все вищесказане, PostgreSQL - це не лише потужна реляційна база даних, але й гнучке та розширюване рішення, яке може відповісти на різні потреби проектів будь-якого розміру та складності. Його надійність, продуктивність та розширюваність роблять його одним з перевагованих виборів для багатьох розробників.

Висновки до першого розділу.

Детальний аналіз предметної області був проведений для визначення основних вимог до нової платформи, яка буде використовуватися для організації заходів та продажу квитків. В процесі аналізу були досліджені різні існуючі платформи, котрі, хоча й працюють за різними принципами, мають спільний функціонал у сфері організації та продажу квитків. За результатами дослідження

					ІПЗ.КР.Б-121-24-ПЗ	
						19

були визначені ключові вимоги для розробки власної платформи. На основі цих вимог було вибрано комплект основних технологій та інструментів для створення веб-додатку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПЛАТФОРМИ ДЛЯ БРОНЮВАННЯ ТА ПРИДБАННЯ КВИТКІВ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Для забезпечення безперервної роботи і відповідності встановленим стандартам і специфікаціям були розроблені високорівневі вимоги, які повинна виконувати платформа для продажу квитків. Ці вимоги мають забезпечити стабільне функціонування системи та її відповідність необхідним нормам і стандартам.

Бізнес-вимоги включають:

- Для користувачів: забезпечення зручної та безпечної платформи, яка дозволяє легко знаходити та купувати квитки на різноманітні події онлайн.
- Для організаторів: надання інструменту для створення, управління та просування заходів. Платформа має забезпечувати високу надійність, безпеку обробки платежів і бути масштабованою для підтримки зростаючої кількості користувачів і подій.

Оптимізація бізнес-процесів:

- Платформа повинна автоматизувати процеси створення та продажу квитків, забезпечуючи зручність і швидкість обслуговування. Вбудовані функції перевірки автентичності квитків забезпечують підвищений рівень безпеки.
- Платформа має дозволяти користувачам ділитися враженнями від відвіданих заходів, залишати відгуки та оцінки, що допомагає іншим робити обґрунтований вибір і сприяє створенню довіри до системи..

Вимоги користувачів

- Гість
 1. Пошук заходів за запитом
 2. Перегляд інформації про заходи
 3. Авторизація

4. Перегляд інформації про заклади
 5. Реєстрація
 6. Перегляд коментарів
- Авторизований *користувач*
 1. Перегляд повідомлень
 2. Купівля та перегляд придбаних квитків
 3. Написання коментарів до закладів
 4. Прив'язка платіжної картки
 5. Додавання та редагування особистої інформації
 6. Запит на зміну ролі, до організатора
 - Організатор
 1. Створення та відправка заходів на модерацию
 2. Отримання інформації про створені заклади
 - Адміністратор
 1. Створення заходів
 2. Створення закладів
 3. Модерація заходів
 4. Зміна ролі користувачів
 5. Отримання особистої інформації користувачів

Функціональні вимоги

- *Реєстрація та аутентифікація користувачів:*
 1. Автоматизувати процес реєстрації з використанням електронної пошти.
 2. Авторизація за логіном та паролем.
- *Управління профілем користувача:*
 1. Забезпечити користувачам можливість самостійно змінювати свої дані через зручний інтерфейс.
 2. Забезпечити користувачам можливість прив'язувати картку.
- *Пошук та перегляд заходів:*
 1. Використовувати швидкі та ефективні алгоритми пошуку з можливістю фільтрації за різними критеріями, такими як дата, ціна та адреса.

- Купівля квитків:
 1. Забезпечити простий і зрозумілий процес купівлі квитків на заходи.
 2. Інтегрувати платіжні системи для зручності користувачів.
- Управління заходами для організаторів:
 1. Надати організаторам інструменти для легкого створення та редагування заходів.
- Безпека та надійність:
 1. Гарантувати захист чутливих даних користувачів та безпечну обробку платежів.
 2. Забезпечити високу надійність системи та можливість швидкого відновлення даних у разі збою.

Нефункціональні вимоги

- *Продуктивність*
 1. Швидкість відповіді: Система має реагувати на запити користувачів протягом 2 секунд.
 2. Одночасне обслуговування: Підтримка до 1000 активних користувачів одночасно без втрати продуктивності.
- *Безпека*
 1. Захист від кібератак: Захист від SQL-ін'єкцій, та інших кібератак.
- *Масштабованість*
 1. Надійність
 2. Час доступності: Мінімальний час доступності системи - 99% на рік, включаючи перерви для технічного обслуговування.
- *Інтерфейс користувача*
 1. Простота використання: Інтуїтивний і зрозумілий інтерфейс для користувачів різних технічних рівнів.
 2. Адаптивний дизайн: Оптимізація інтерфейсу для роботи на різних пристроях (комп'ютери, планшети, мобільні телефони).
 3. Підтримка користувачів: Ефективна технічна підтримка через електронну пошту.

1. Вимоги до середовища виконання

- Серверна та клієнтська частина :

1. Node.js: версія 14
2. npm або yarn для управління пакетами

- База даних :

1. PostgreSQL: версія 12 або новіша

2. Вимоги до серверу

- Операційна система:

1. Linux (рекомендовано, наприклад, Ubuntu 20.04 або новіша)

- Апаратне забезпечення:

1. CPU: Мінімум 2 ядра
2. RAM: Мінімум 4 ГБ (рекомендовано 8 ГБ або більше)
3. Дисковий простір: Мінімум 50 ГБ (рекомендовано SSD для кращої продуктивності)

- Програмне забезпечення:

1. Node.js: версія 14
2. npm або yarn для управління пакетами
3. PostgreSQL: версія 12 або новіша

- Безпека:

1. Встановлення та налаштування брандмауера (наприклад, UFW на Ubuntu)
2. Захист бази даних паролем та шифруванням

- Мережа:

1. Мінімум 1 Гбіт/с мережеве підключення
2. Статична IP-адреса для сервера

Після глибокого аналізу користувацьких вимог була виявлена необхідність у розробці конкретних сценаріїв використання для платформи продажу квитків. Цей процес є критичним у фазі проектування, адже детально розроблені сценарії допомагають визначити реальні моделі взаємодії користувачів з додатком. Для наглядного представлення цих сценаріїв була створена діаграма використання

програмної системи (рисунок 2.1), що ілюструє взаємодії між різними користувачами та функціональними можливостями програми.

Ця діаграма деталізує кожен сценарій, показуючи, які дії виконує програмна система відповідно до дій користувачів. Також, чітко визначені сценарії спрощують процес тестування програми, оскільки вони виділяють критичні точки взаємодії, які потрібно перевірити для забезпечення бездоганної роботи системи у реальних умовах..

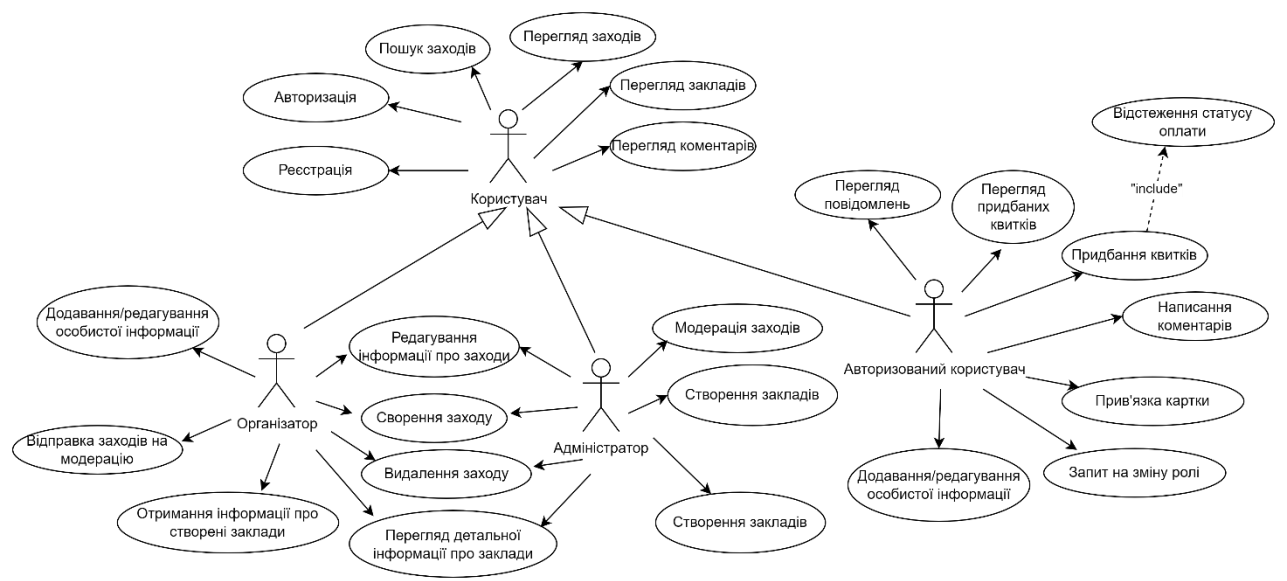


Рис. 2.1. Діаграма варіантів використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.15.

Таблиця 2.1

Опис прецеденту «Реєстрація»

Назва варіанту використання	Реєстрація
Основні актори	Гість
Короткий опис	Гість має можливість зареєструватись та створити особистий акаунт в системі. Отримуючи цим доступ до функціоналу платформи

Таблиця 2.2

Опис прецеденту «Авторизація»

Назва варіанту використання	Авторизація
Основні актори	Гість
Короткий опис	Гість може авторизуватись, підтвердивши цим свою особистість.

Таблиця 2.3

Опис прецеденту «Перегляд заходів»

Назва варіанту використання	Перегляд заходів
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть переглядати дані про захід, отримувати загальну інформацію про нього.

Таблиця 2.4

Опис прецеденту «Перегляд закладів»

Назва варіанту використання	Внесення фінансового внеску
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть переглядати дані про заклад кожного заходу, отримувати загальну інформацію про нього.

Таблиця 2.5

Опис прецеденту «Перегляд коментарів»

Назва варіанту використання	Перегляд коментарів
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть переглядати коментарі, отримувати загальну інформацію про репутацію закладу.

Таблиця 2.6

Опис прецеденту «Пошук заходів»

Назва варіанту використання	Пошук заходів
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть здійснювати пошук заходів за певними фільтрами та використовувати сортування результатів

Таблиця 2.7

Опис прецеденту «Перегляд повідомлень»

Назва варіанту використання	Перегляд повідомлень
Основні актори	Авторизований користувач
Короткий опис	Авторизований користувач може переглядати свої повідомлення, також вони дублюються на вказану електронну пошту

Таблиця 2.8

Опис прецеденту «Придбання квитків»

Назва варіанту використання	Придбання квитків
Основні актори	Авторизований користувач
Короткий опис	Авторизований користувач може придбати квиток на будь-який захід, за умови, що квитки на захід все ще продаються та користувач має прив'язану картку

Таблиця 2.9

Опис прецеденту «Написання коментарів»

Назва варіанту використання	Написання коментарів
Основні актори	Авторизований користувач
Короткий опис	Авторизований користувач може написати 2 коментарі до закладу та поставити оцінку від 0 до 5 зірок

Таблиця 2.10

Опис прецеденту «Написання коментарів»

Назва варіанту використання	Написання коментарів
Основні актори	Авторизований користувач
Короткий опис	Авторизований користувач може написати 2 коментарі до закладу та поставити оцінку від 0 до 5 зірок

Таблиця 2.11

Опис прецеденту «Відправка заходу на модерацию»

Назва варіанту використання	Відправка заходу на модерацию
Основні актори	Організатор
Короткий опис	Актор з роллю організатор має можливість відправити на модерацию до адміністратора створений ним захід, заповнивши інформацію про нього перед цим.

Таблиця 2.12

Опис прецеденту «Відправка заходу на модерацию»

Назва варіанту використання	Відправка заходу на модерацию
Основні актори	Організатор
Короткий опис	Актор з роллю організатор має можливість відправити на модерацию до адміністратора створений ним захід, заповнивши інформацію про нього перед цим.

Таблиця 2.13

Опис прецеденту «Отримання інформації про створені заклади»

Назва варіанту використання	Отримання інформації про створені заклади
Основні актори	Організатор
Короткий опис	Користувач організатор має можливість отримати список всіх його заходів, котрі зараз знаходяться на модерации.

Таблиця 2.14

Опис прецеденту «Модерація заходу»

Назва використання	варіанту	Модерація заходу
Основні актори		Адміністратор
Короткий опис		Адміністратор має може отримати список всіх заходів, котрі зараз знаходять на модерації

Таблиця 2.15

Опис прецеденту «Зміна ролі»

Назва використання	варіанту	Зміна ролі
Основні актори		Адміністратор, користувач
Короткий опис		Адміністратор має може отримати прохання від користувача про зміну ролі та за потреби реалізувати його. Користувач в свою чергу може відправити запит на зміну ролі.

Після детального аналізу функціональних вимог було визначено ключові сутності, необхідні для розробки додатку: User, Event, Notification, Order, Transaction, Review, PaymentCard, та Venue. Для наглядного представлення цих сутностей було створено діаграму класів (рис. 2.2). Далі були розглянуті взаємозв'язки між сутностями, їх атрибути та методи, що дозволило визначити необхідні модулі для реалізації функціоналу, зокрема модулі для організації заходів, пошуку та модерації заходів, а також управління ролями користувачів.

На основі цього аналізу було розроблено проєкт бази даних. Враховуючи поставлені завдання та вимоги, було вирішено, що оптимальним рішенням буде використання реляційної бази даних. Такий підхід забезпечить ефективне

управління інформацією та підтримку всіх необхідних функцій додатку, включно з обробкою транзакцій та збереженням даних користувачів, а також керуванням заходами.

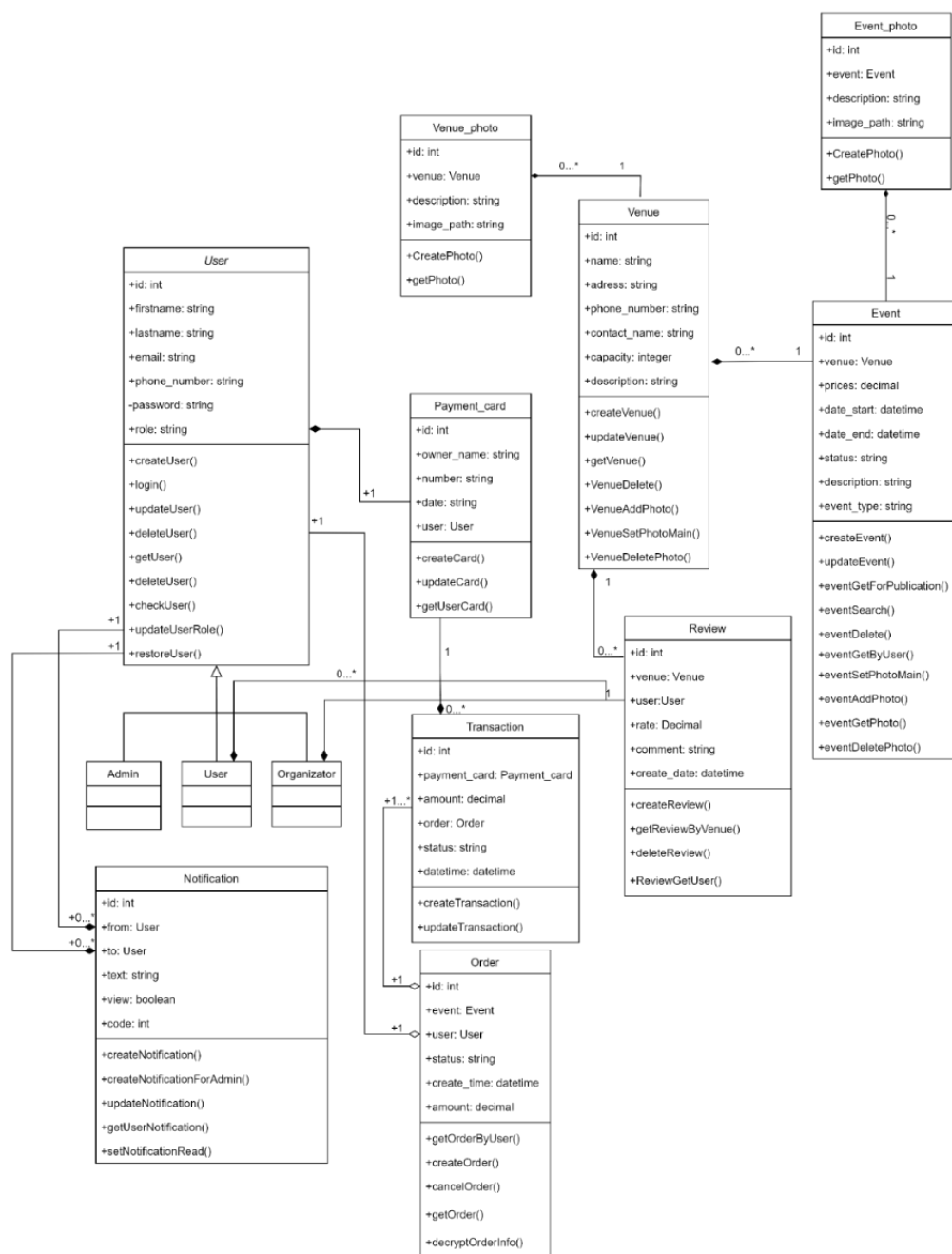


Рис. 2.2. Діаграма класів

User – клас який відповідає за користувача:

1. `createUser()` - Цей метод створює нового користувача в системі.
2. `login()` - Метод для аутентифікації користувача в системі .
3. `updateUser()` - Метод, який оновлює існуючий запис користувача з новими даними.

4. deleteUser() - Видаляє користувача з системи, точніше позначає його як видаленого.
5. getUser() - Повертає інформацію про користувача на основі заданого ідентифікатора.
6. checkUser() - Перевіряє токен користувача.
7. updateUserRole() - Змінює роль або права доступу користувача в системі.
8. restoreUser() - Відновлює раніше видаленого користувача в системі.

Event – клас котрий відповідає за заходи:

1. createEvent() - метод створює нову подію в системі з наданими даними.
2. updateEvent() - Він оновлює існуючу подію.
3. eventGetForPublication() - Цей метод повертає список подій, які очікують на перевірку адміністратором.
4. eventSearch() - Він здійснює пошук подій за певними критеріями.
5. eventDelete() - Цей метод видаляє існуючу подію з системи.
6. eventGetByUser() - Він повертає список подій, які створені певним користувачем.
7. eventSetPhotoMain() - Цей метод встановлює певну фотографію як головну для події.
8. eventAddPhoto() - Він додає фотографію до події.
9. eventGetPhoto() - Цей метод повертає список фотографій, які пов'язані з подією.
10. eventDeletePhoto() - Він видаляє фотографію, яка пов'язана з подією.

Notification – клас котрий відповідає за повідомлення:

1. createNotification() - Створює нове повідомлення.
2. createNotificationForAdmin() - Створює нове повідомлення для випадкового адміністратора системи.
3. getNotification() - Отримує конкретне повідомлення за його ідентифікатором.
4. getUserNotification() - Повертає список повідомлень, призначених для

конкретного користувача.

5. `getNotificationNumberByUser()` - Повертає кількість непрочитаних повідомлень для конкретного користувача.

Order – клас котрий відповідає за замовлення:

1. `getOrderByUser()` - Повертає список замовлень, зроблених певним користувачем.
2. `createOrder()` - Створює нове замовлення.
3. `cancelOrder()` - Скасовує конкретне замовлення.
4. `getOrder()` - Повертає деталі конкретного замовлення за його ідентифікатором.
5. `decryptOrderInfo()` – Повертає інформацію про замовлення.

photo_venue – клас котрий відповідає за фото закладів:

1. `CreatePhoto()`: Створює нову фотографію закладу.
2. `getPhoto()`: Повертає інформацію про конкретну фотографію за її ідентифікатором.

Payment_card – клас котрий відповідає за платіжну картку:

1. `createCard()` - Створює нову карту для користувача в системі з наданими даними.
2. `updateCard()` - Оновлює існуючу карту користувача з новими даними.
3. `getUserCard()` - Повертає інформацію про карту конкретного користувача в системі.

Review – клас котрий відповідає за огляди:

1. `ReviewCreate()` - Створює новий відгук.
2. `ReviewGetUser()` - Повертає список відгуків, що належать конкретному користувачеві.
3. `ReviewGet()` - Повертає конкретний відгук.
4. `ReviewGetVenue()` - Повертає список відгуків, що стосуються певного місця (залу, закладу тощо).
5. `ReviewDelete()` - Видаляє конкретний відгук.

Venue – клас котрий відповідає за заклади:

1. createVenue() - Створює новий заклад в системі.
2. updateVenue() - Оновлює інформацію про існуючий заклад за новими даними.
3. getVenue() - Повертає інформацію про конкретний заклад за його ідентифікатором.
4. VenueDelete() - Видаляє конкретний заклад.
5. VenueAddPhoto() - Додає фотографію до певного закладу.
6. VenueSetPhotoMain() - Встановлює фотографію як головну для певного закладу.
7. VenueDeletePhoto() - Видаляє певну фотографію, пов'язану з закладом.

photo_event – клас котрий відповідає за фото заходів:

1. CreatePhoto(): Створює нову фотографію заходу.
2. getPhoto(): Повертає інформацію про конкретну фотографію за її ідентифікатором.

2.2 Розробка бази даних системи

Для збереження даних у додатку використовується СУБД PostgreSQL, вона є представником бази даних реляційного типу.

База даних системи складається з п'яти таблиць:

1. User – таблиця для зберігання даних користувачів;
2. Event – таблиця для зберігання інформації про заходи;
3. Notification – таблиця для зберігання інформації про повідомлення;
4. Order – таблиця для зберігання інформації про замовлення;
5. PaymentCard – таблиця для зберігання інформації про платіжні картки користувачів;
6. Photo_event – таблиця для зберігання інформації про фото закладів;
7. Photo_venue – таблиця для зберігання інформації про фото заходів;
8. Review - таблиця для зберігання інформації про відгуки;
9. Transaction- таблиця для зберігання інформації про транзакції користувачів;

Venue- таблиця для зберігання інформації про заклади;

Опис кожної з таблиць бази даних наведений у таблицях 2.16 – 2.24.

Таблиця 2.16

Опис таблиці User

User		
Поле	Тип	Призначення
id	int	Ідентифікатор
firstName	varchar	Ім'я
lastName	varchar	Прізвище
email	varchar	Електронна пошта
phone_number	varchar	Телефоний номер
password	varchar	Пароль
role	enum	Роль

Таблиця 2.17

Опис таблиці Event

Event		
Поле	Тип	Призначення
id	int	Ідентифікатор
name	varchar	Назва
price	int	Ціна квитка
date_start	Date	Дата та час початку заходу
date_end	Date	Дата та час кінця заходу
status	Enum	Стан заходу
available_places	Int	Кількість вільних місць
description	varchar	Опис

Продовження таблиці 2.17

event_type	varchar	Тип заходу
created_by	Int	Ідентифікатор користувача
venue	Int	Ідентифікатор заходу

Таблиця 2.18

Опис таблиці Notification

Notification		
Поле	тип	Призначення
id	int	Ідентифікатор
code	enum	Код повідомлення
createdDate	date	Айді збору в блокчейні
view	bool	Адреса гаманця користувача
message	varchar	Текст повідомлення
from	int	Ідентифікаторо користувача
to	int	Ідентифікаторо користувача

Таблиця 2.19

Опис таблиці Order

Order		
Поле	Тип	Призначення
id	int	Ідентифікатор
status	enum	Статус замовлення
createdDate	date	Дата та час створення
Amount	int	Ціна
place_number	int	Кількість квитків
Transaction	int	Ідентифікатор транзакції
user	int	Ідентифікатор користувача
event	int	Ідентифікатор заходу

Таблиця 2.20

Опис таблиці paymentCard

paymentCard		
Поле	Тип	Призначення
id	int	Ідентифікатор
owner_name	varchar	Ім'я та прізвище власника
card_number	varchar	Номер картки
date	varchar	Дата закінчення актуальності картки
user	int	Ідентифікатор користувача

Таблиця 2.21

Опис таблиці photo_event

photo_event		
Поле	Тип	Призначення
id	int	Ідентифікатор
description	varchar	Опис
image_key	varchar	Унікальний ключ фотографії
event	Int	Ідентифікатор заходу
isMain	Bool	Вказує, чи є фото головним зображенням заходу

Таблиця 2.22

Опис таблиці photo_venue

photo_venue		
Поле	Тип	Призначення
id	int	Ідентифікатор
description	varchar	Опис
image_key	varchar	Унікальний ключ фотографії
venue	Int	Ідентифікатор закладу
isMain	Bool	Вказує, чи є фото головним зображенням закладу

Таблиця 2.23

Опис таблиці Review

Review		
Поле	Тип	Призначення
id	int	Ідентифікатор
rate	int	Оцінка
comment	varchar	Коментар
createdDate	date	Дата та час створення
venue	int	Ідентифікатор закладу
user	int	Ідентифікатор користувача

Таблиця 2.24

Опис таблиці Transaction

Transaction		
Поле	Тип	Призначення
id	int	Ідентифікатор
status	Enum	Статус замовлення
createdDate	date	Дата та час створення
payment_card	int	Ідентифікатор платіжної картки

Опис таблиці Venue

Venue		
Поле	Тип	Призначення
id	int	Ідентифікатор
name	varchar	Назва
address	varchar	Адреса
phone_number	Date	Телефонний номер
capacity	Enum	Вмістимість
contact_name	Int	Контактний номер
rate	varchar	Оцінка

Ці таблиці підтримують зв'язки між собою, забезпечують збереження, керування та обробку всіх необхідних даних для ефективної та прогнозованої роботи додатку. Оскільки в мові SQL немає можливості створювати масиви було створено декілька проміжних таблиць photo_event та photo_venue.

2.3 Проектування та реалізація алгоритмів роботи системи

Node.js має кілька ключових переваг для використання на серверній стороні. По-перше, його висока швидкість і ефективність досягаються завдяки V8 JavaScript engine від Google, який компілює JavaScript безпосередньо в машинний код, що забезпечує високу продуктивність і швидкість виконання. Node.js використовує подійну модель та неблокуючий ввід/вивід, що дозволяє обробляти велику кількість одночасних запитів.

Використання JavaScript як на серверній, так і на клієнтській стороні дозволяє працювати з однією мовою по всьому стеку, що спрощує процес розробки та дозволяє легко ділитися кодом між фронтендом і бекендом. Node.js також забезпечує масштабованість, дозволяючи легко масштабувати програми як у горизонтальному, так і у вертикальному напрямках. Велика екосистема

модулів та пакетів (наприклад, npm і yarn) дозволяє швидко додавати нову функціональність і скорочувати час розробки.

Для реалізації зв'язку між Node.js і React використовується архітектура клієнт-сервер. React відповідає за клієнтську частину, забезпечуючи динамічний і інтерактивний інтерфейс користувача, тоді як Node.js працює як сервер, обробляючи запити клієнтів і взаємодіючи з базою даних.

На клієнтській стороні за допомогою React створюються компоненти для відображення даних і взаємодії з користувачем. Компоненти React відправляють HTTP-запити до серверу Node.js за допомогою бібліотеки Axios.

На серверній стороні Node.js використовує Express.js для обробки цих HTTP-запитів. Наприклад, коли клієнт надсилає запит для придбання квитка, сервер перевіряє автентифікацію користувача за допомогою JWT, перевіряє доступність квитків, зберігає інформацію про придбання в базі даних та оновлює кількість доступних квитків для заходу.

Таким чином, основна взаємодія між React і Node.js полягає у відправці запитів з клієнтської частини на сервер, де обробляється бізнес-логіка та взаємодія з базою даних, а результати передаються назад клієнту для відображення. Це дозволяє створювати динамічні та інтерактивні веб-додатки з використанням сучасних технологій.

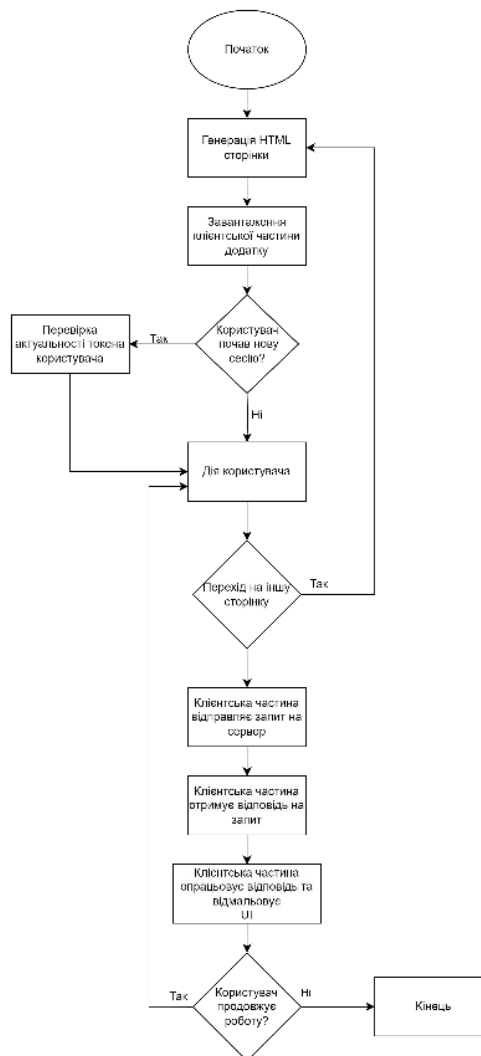


Рисунок 2.3 – Загальний алгоритм роботи додатку

Після розробки загального алгоритму додатку важливо працювати над алгоритмами для специфічних сценаріїв. Для цього часто застосовуються діаграми діяльності, які допомагають візуалізувати та детальніше роз'яснити окремі процеси в додатку. Наприклад, було створено діаграму для сценарію «Публікація заходу» (рисунок 2.4).

У цьому сценарії, після авторизації, користувач отримує доступ до форми створення заходу. Після заповнення форми користувач відправляє запит на сервер для створення заходу. Якщо дані введені коректно, захід реєструється в базі даних зі статусом «В очікуванні», що означає, що він доданий до черги для перевірки адміністратором. Адміністратор має можливість відхилити або схвалити публікацію заходу. Ці кроки представлені на діаграмі діяльності, яка деталізує весь процес від початку до кінця, що допомагає зрозуміти взаємодію

користувача з системою та адміністративні процеси, пов'язані з публікацією заходу.

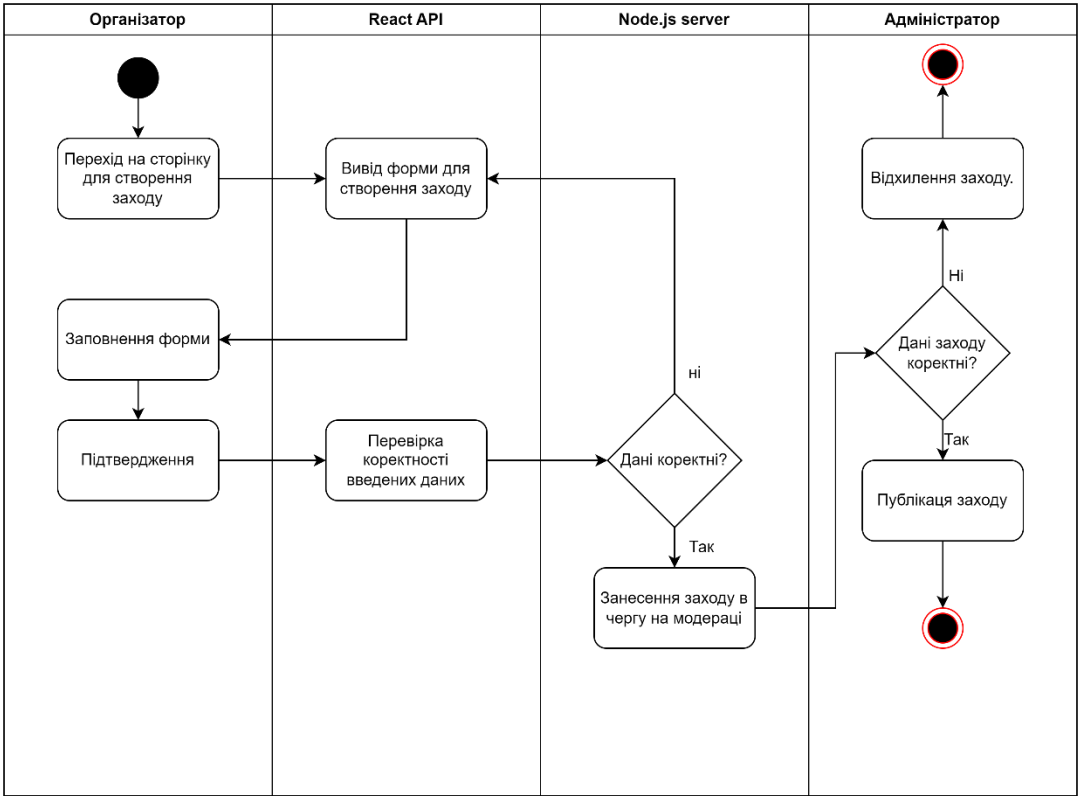


Рисунок 2.4 – Діаграма діяльності «Публікація заходу»

2.4 Реалізація функціоналу платформи для бронювання та придбання квитків

Розглянемо основні стадії алгоритму створення заходу на платформі. Спершу користувач повинен зареєструвати новий обліковий запис. Після успішної реєстрації він отримує можливість створити новий захід. Процес створення заходу відображено нижче, повна функція знаходиться в додатках.

Функція створення заходу: Функція призначена для створення нової події. Вона отримує дані про подію з запиту користувача, валідує їх за схемою, знаходить відповідний заклад та користувача в базі даних, і створює новий запис події. Після збереження події у базі даних, вона повертає ідентифікатор створеної події у відповідь на запит користувача. У випадку помилки, функція правильно обробляє та відображає відповідну помилку користувачеві.

```
export const eventCreate = asyncHandler(
```

```

async (req: UserRequest, res: Response) => {
  let {
    name,
    price,
    date_start,
    date_end,
    description,
    event_type,
    available_places,
    venue_id,
  } = req.body;

  // Create a new event
  let new_event: Event = await eventRepository.create({
    name,
    price,
    date_start: new Date(date_start),
    date_end: new Date(date_end),
    description,
    event_type,
    venue: Venue,
    available_places,
    created_by: user,
  });
  new_event = await eventRepository.save(new_event);
  res.json({ event_id: new_event.id });
} catch (err) {
  if (res.statusCode === 200) {
    res.status(500);
  }
  throw new Error(err.message);
}
}
);

```

Далі Адміністратору потрібно отримати список всіх заходів, котрі очікують на перевірку, для цього реалізовано окрему функцію, лістинг її наведено нижче:

Функція для пошуку заходів, очікуючих перевірки: Функція призначена для адміністративного отримання подій, які готуються до публікації. Вона перевіряє права доступу користувача, валідує вхідні дані, шукає та повертає відсортований список подій.

```

export const eventGetForPublication = asyncHandler(
  async (req: UserRequest, res: Response) => {
    let { sortorder } = req.query;
    console.log(sortorder);
    if (req.user.role !== UserRole.ADMIN) {

```

					ІПЗ.КР.Б-121-24-ПЗ	
						44

```

        res.status(403);
        throw new Error("Користувач не має права отримати дану інформацію");
    }
    try {
        await eventForPulication.validate({
            sortOrder: sortorder,
        });
    } catch (err) {
        res.status(400);
        throw new Error(err.message);
    }
    let events: Event[] = await eventRepository.find({
        where: { status: eventStatus.PENDING },
        order: { date_start: sortorder },
        relations: { photos: true, created_by: true },
    });
    res.status(200).json(events);
}
);

```

Після успішного створення заходу має відбутись перевірка та підтвердження його адміністратором, для цього використовується функцію для оновлення даних заходу, лістинг частково наведено нижче:

Функція для оновлення заходів: Функція призначена для оновлення інформації про подію. Вона отримує дані з тіла запиту, валідує їх, знаходить існуючу подію за ідентифікатором, оновлює її атрибути, зберігає зміни в базі даних і повертає підтвердження успішного оновлення. У разі оновлення даних користувачем з роллю «організатор», захід автоматично направляється на повторну перевірку адміністратором.

```

export const eventUpdate = asyncHandler(
    async (req: UserRequest, res: Response) => {
        const {
            name,
            price,
            date_start,
            date_end,
            description,
            status,
            event_type,
            venue_id,
        } = req.body;
        const event_id: number = parseInt(req.params.id);
        let venue: Venue;
        try {

```

					ІПЗ.КР.Б-121-24-ПЗ	
						45

```

const event = await eventRepository.findOne({
  where: { id: event_id },
  relations: { venue: true },
});
const user = await userRepository.findOneBy({ id: req.user.id });

// If a new venue ID is provided, find and update the venue
if (venue_id) {
  venue = await venueRepository.findOneBy({ id: venue_id });
  if (!venue) {
    res.status(404);
    throw new Error("заклад не знайдено");
  }
  event.venue = venue;
}
// Update event attributes based on the provided data
event.name = name || event.name;
event.price = price || event.price;
event.venue = venue;
event.date_start = date_start ? new Date(date_start) : event.date_start;
event.date_end = date_end ? new Date(date_end) : event.date_end;
event.description = description || event.description;
event.event_type = event_type || event.event_type;
event.status = status || event.status;

if (req.user.role === UserRole.ORGANIZER) {
  event.status = eventStatus.PENDING;
}

await eventRepository.save(event);
res.status(200).json({ status: "інформацію оновлено" });
} catch (err) {
  if (res.statusCode === 200) {
    res.status(500);
  }
  throw new Error(err);
}
}
);

```

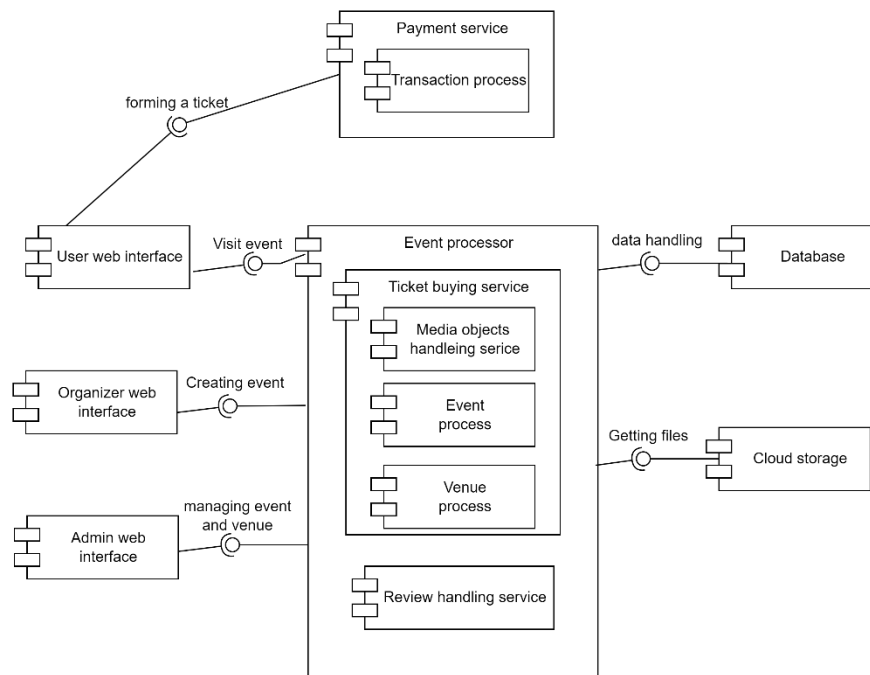


Рис. 2.16. Діаграма компонентів

Детальний опис окремих компонентів діаграми:

1. **User Web Interface (Інтерфейс користувача):** Компонент, який відповідає за взаємодію з кінцевим користувачем системи.
2. **Organizer Web Interface (Інтерфейс організатора):** Компонент, призначений для організаторів подій, щоб керувати своїми подіями через веб-інтерфейс.
3. **Admin Web Interface (Інтерфейс адміністратора):** Компонент, що забезпечує адміністраторам доступ до системи для виконання адміністративних завдань.
4. **Event Processor (Обробник подій):** Центральний компонент системи, який складається з:
 - а. **Ticket Buying Service (Сервіс купівлі квитків):** Обробляє процеси купівлі квитків
 1. **Media Objects Handling Service (Сервіс обробки медіа-об'єктів):** Обробляє медіа-об'єкти, такі як зображення або відео, пов'язані з подіями.

2. **Event Process (Процес подій):** Відповідає за обробку та управління подіями.

3. **Venue Process (Процес місця проведення):** Керує інформацією про місця проведення подій.

b. **Review Handling Service (Сервіс обробки відгуків):** Відповідає за обробку та управління відгуками користувачів.

5. **Payment Service (Сервіс платежів):** Обробляє фінансові транзакції, пов'язані з купівлею квитків.

a. **Transaction Process (Процес транзакцій):** Конкретний процес обробки платежів.

6. **Database (База даних):** Компонент для зберігання всіх даних системи.

7. **Cloud Storage (Хмарне зберігання):** Використовується для зберігання великих медіа-об'єктів та інших даних, які потребують масштабованого зберігання.

Висновки до другого розділу

У процесі розробки платформи для продажу квитків було проведено глибокий аналіз усіх потенційних сценаріїв використання програмного забезпечення згідно з встановленими вимогами. На базі цього аналізу була розроблена структура платформи, що включала моделювання її архітектури та визначення взаємодії між окремими компонентами системи.

Для забезпечення функціональності було створено базу даних, яка відповідає за зберігання інформації на серверній стороні додатку, включаючи дані про користувачів, події та інше важливе для роботи платформи. Також було ретельно розроблено алгоритми взаємодії користувачів з додатком та описано послідовні кроки для виконання різних сценаріїв використання.

Основний функціонал платформи, зокрема створення, підтвердження адміністрацією та публікація подій, був чітко описаний і систематизований, що дозволяє користувачам ефективно організовувати та керувати подіями через платформу. Це дало змогу забезпечити логічний і послідовний процес взаємодії

					ІПЗ.КР.Б-121-24-ПЗ	
						48

всіх учасників з системою, оптимізуючи її використання і покращуючи загальний досвід користувачів..

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ

3.1 Порядок встановлення та налаштування платформи

Встановлення та налаштування платформи вимагає розгортання всіх його компонентів та налаштування як кожного елементу окремо, так і загальної системи в цілому.

Діаграма розгортання відображає архітектуру веб-платформи та взаємодію між його різними компонентами. На діаграмі представлено кілька основних блоків.

Пристрій користувача містить браузер, через який здійснюється взаємодія з платформою через веб-інтерфейс.

Веб-сервер є центральним компонентом системи і включає кілька підсистем. Сервіс оплати містить процесор транзакцій, який обробляє платіжні транзакції. Процеси обробки подій включають кілька компонентів для обробки подій додатку, інформації про заклади та відгуків користувачів. Інтерфейс бази даних забезпечує взаємодію з базою даних, а процесор управління файлами управляє файлами, збереженими в системі. Інтерфейс хмари забезпечує взаємодію з хмарним сховищем.

Постачальник платіжних послуг інтегрується з веб-сервером через інтерфейс постачальника платіжних послуг, що дозволяє обробляти транзакції.

База даних використовується для зберігання даних додатку. В даному випадку це база даних Postgres.

Файлове сховище використовує хмарне сховище для зберігання файлів. Взаємодія між компонентами відбувається наступним чином: браузер на пристрої користувача взаємодіє з веб-сервером через веб-інтерфейс. Веб-сервер, в свою чергу, взаємодіє з постачальником платіжних послуг для обробки транзакцій, з базою даних для зберігання та отримання даних, а також з хмарним сховищем для збереження файлів. Ця структура дозволяє розподілити

функціональність між різними компонентами і забезпечує їх взаємодію для виконання різних задач додатку.

Для опису архітектури платформи було створено діаграму розгортання (рис. 3.1).

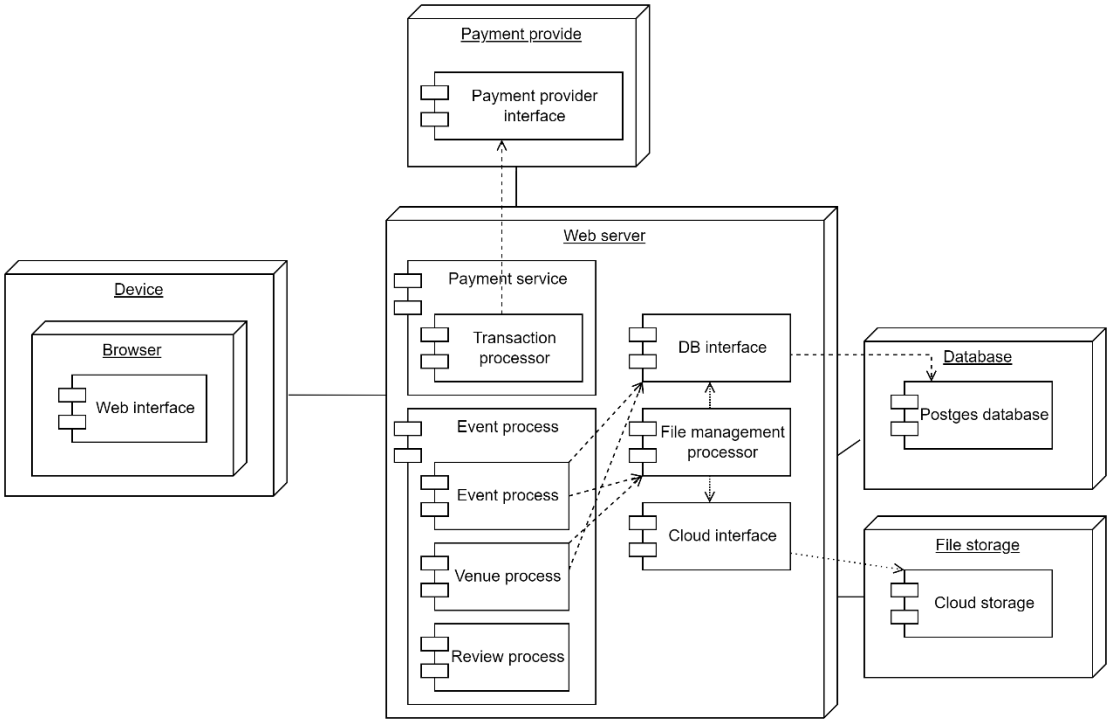


Рис. 3.1. Діаграма розгортання платформи для продажу квитків блокчейном.

3.2 Структура інтерфейсу веб-платформи

На головній сторінці неавторизований користувач може побачити: панель швидкого доступу з пошуком та кнопками для авторизації та реєстрації, декілька підбірок заходів.

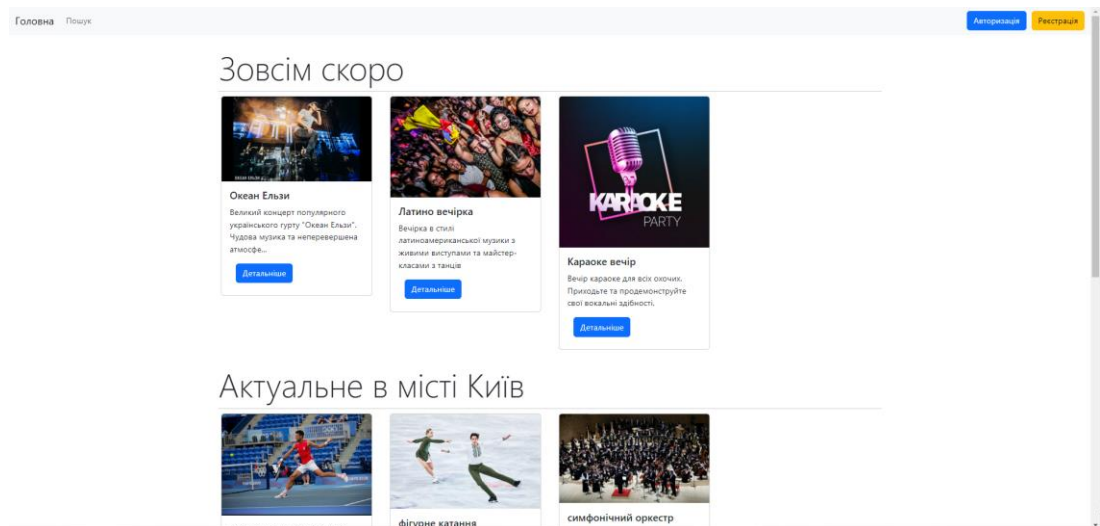


Рис. 3.3. Головна сторінка додатку

Перейшовши на сторінку певного заходу неавторизований користувач матиме такий самий набір інформації, що і авторизований з роллю “user” але не матиме можливість прибрати квитки.

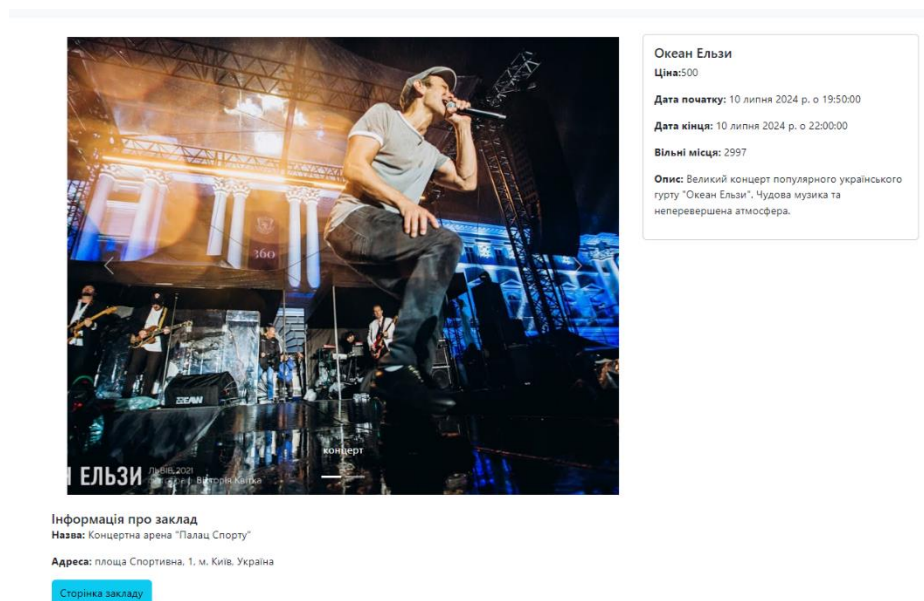


Рис. 3.5 сторінка заходу

Так само він має можливість перейти до сторінки закладу, в якому проходить захід, та переглянути основну інформацію, лише буде обмежено право написання коментарів але не їх перегляд.

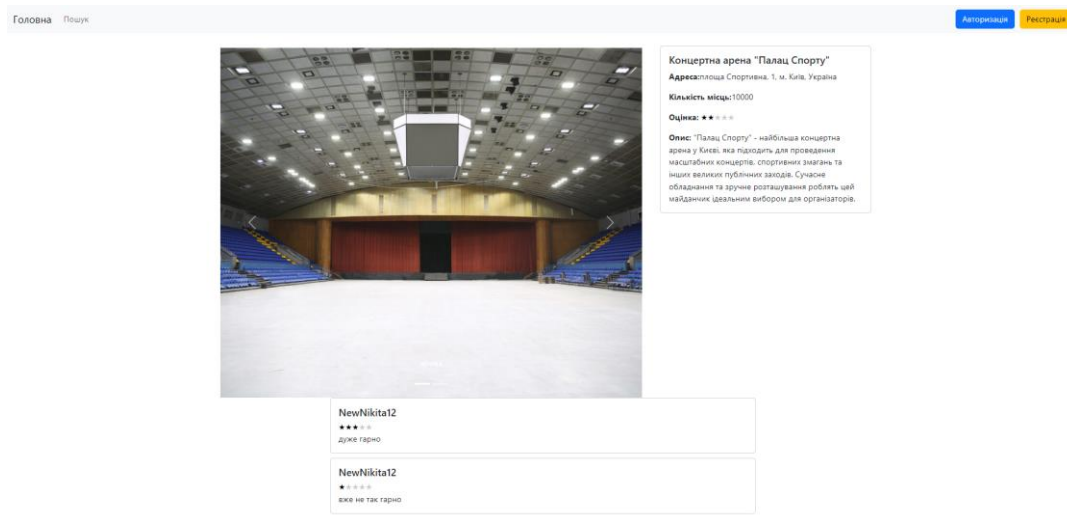


Рис. 3.6 сторінка заходу

Скориставшись пошуком користувач отримає можливість знайти потрібний йому захід чи заходи, використовуючи різноманітні пошукові фільтри та сортування

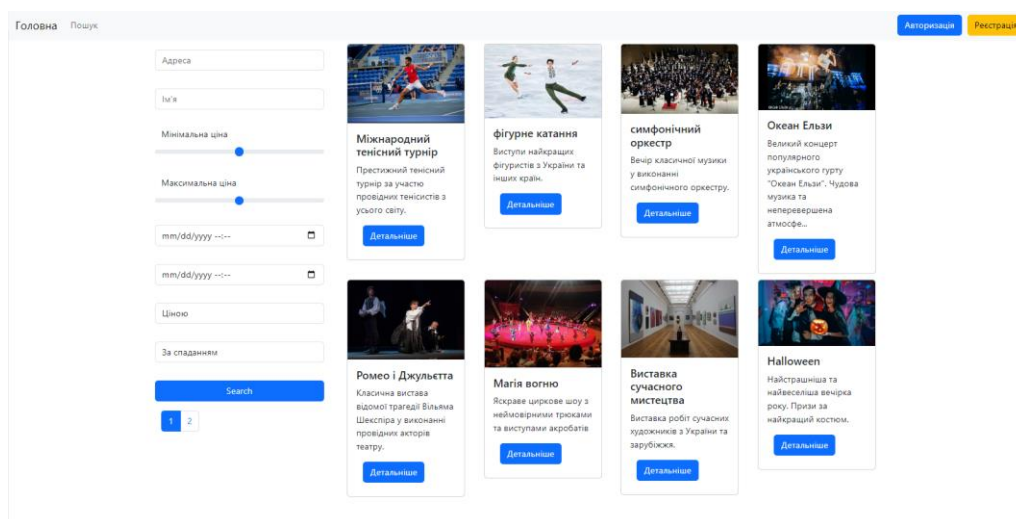


Рис. 3.6 сторінка пошуку

Ознайомившись з інтерфейсом користувач може перейти до сторінки реєстрації натиснувши відповідну кнопку на панелі швидкого доступу

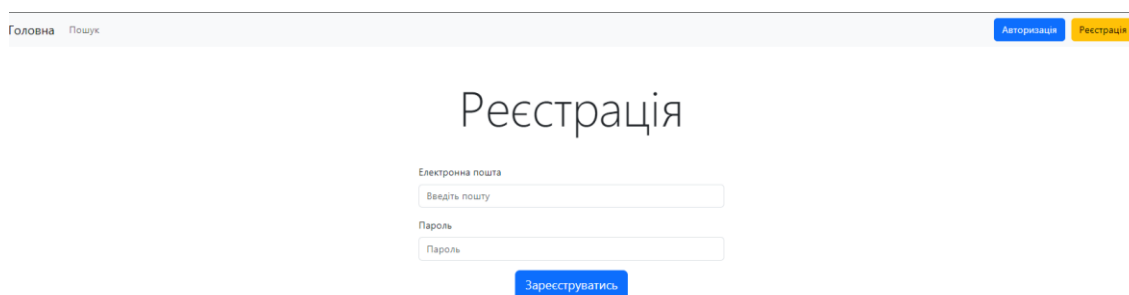
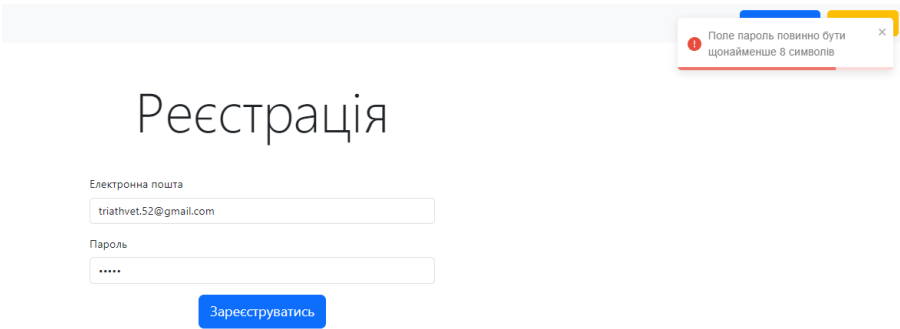


Рис. 3.7 сторінка реєстрації

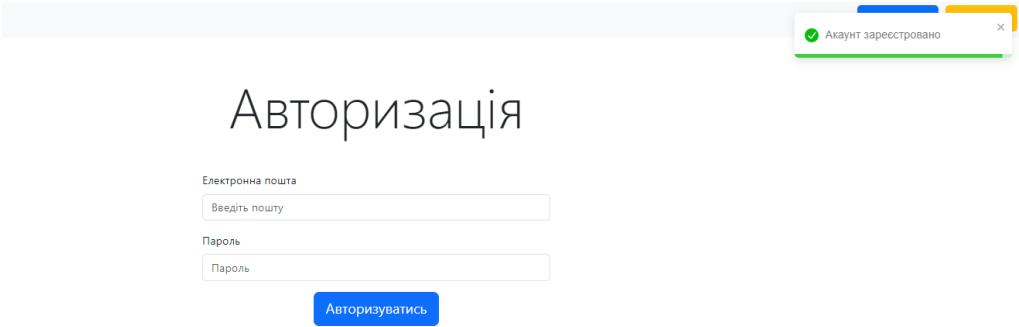
У разі помилково ведених даних користувач отримає повідомлення з описом помилки



The registration form is titled "Реєстрація". It contains two input fields: "Електронна пошта" (Email) with the value "triathlet.52@gmail.com" and "Пароль" (Password) with masked characters "*****". A blue button labeled "Зареєструватись" is at the bottom. A red error message box in the top right corner states: "Поле пароль повинно бути щонайменше 8 символів" (The password field must be at least 8 characters).

Рис. 3.8 помилка реєстрації

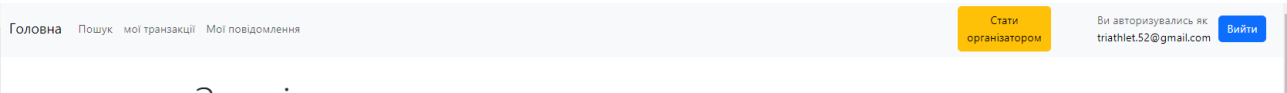
У разі вдалої реєстрації користувача буде автоматично перенаправлено на сторінку для авторизації та виведено повідомлення про успішну реєстрацію



The authorization form is titled "Авторизація". It contains two input fields: "Електронна пошта" (Email) with the placeholder "Введіть пошту" and "Пароль" (Password) with the placeholder "Пароль". A blue button labeled "Авторизуватись" is at the bottom. A green success message box in the top right corner states: "Акаунт зареєстровано" (Account registered).

Рис. 3.9 вдала реєстрація

Після вдалої авторизації в особистий акаунт користувача ми можемо помітити нові кнопки на панелі швидкого доступу, вони відповідають за перегляд статусу транзакції та повідомлень, кнопка “стати організатором”, та відомості про авторизованого користувача натичнувши на яку ми перейдемо до кабінету користувача



The navigation bar includes links: "Головна", "Пошук", "мої транзакції", and "Мої повідомлення". On the right, there is a yellow button "Стати організатором", a user profile section showing "Ви авторизувались як triathlet.52@gmail.com" with a "Вийти" (Logout) button.

Рис. 3.10 навігаціна панель авторизованого користувача

Сторінка ‘Мої транзакції’ служить для демонстрації інформації про транзакції, користувач може використовувати зручний фільтр статусу для полегшення пошуку потрібної інформації

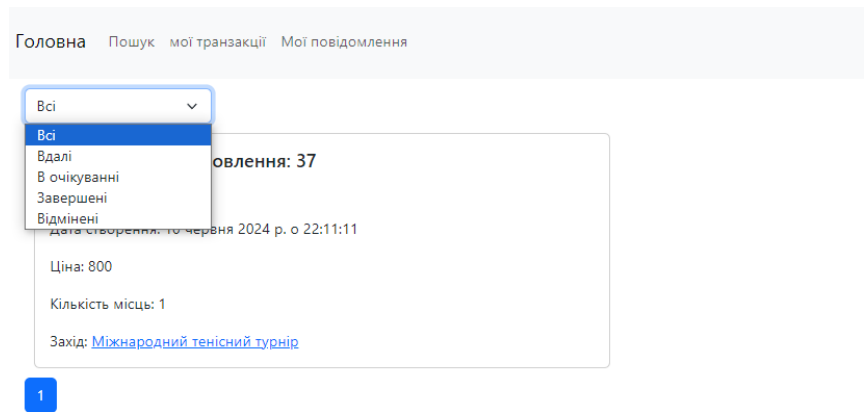


Рис. 3.11 сторінка “Мої транзакції”

На сторінці “Мої повідомлення” користувач може переглянути адресовані йому повідомлення, використовувати зручний фільтр прочитаних повідомлень та сортування за датою створення повідомлення.

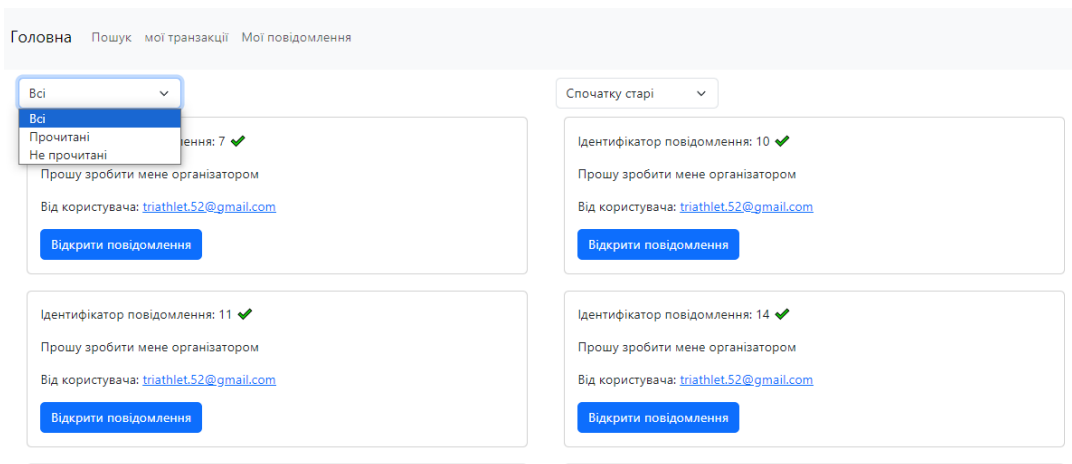


Рис. 3.12 сторінка “Мої повідомлення”

Натиснувши на кнопку “Відкрити повідомлення” ми можемо переглянути детальну інформацію про адресата та адресанта

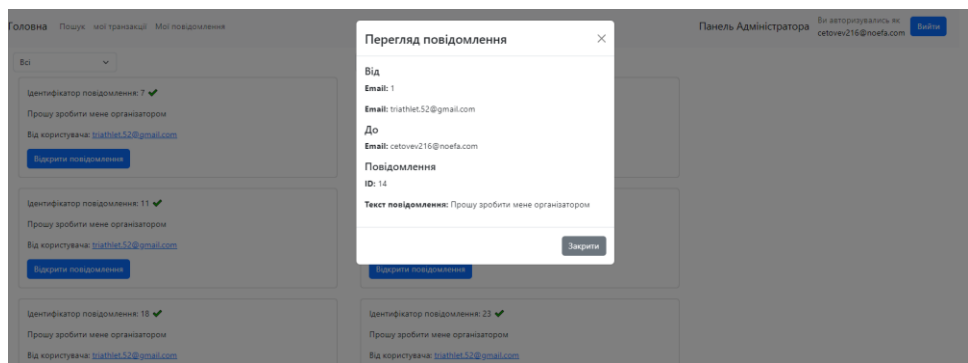


Рис. 3.13 детальний перегляд повідомлення

Для редагування особистих даних користувача ми можемо перейти до кабінету користувача та обрати потрібний пункт меню



Рис. 3.14 Кабінет користувача

Обравши пункт “Особисті дані” користувач отримає форму для редагування основних даних користувача, також виводяться дані котрі не можна змінити самостійно: унікальний ідентифікатор та роль.

Особисті дані

унікальний ідентифікатор: 1

Роль: користувач

Ім'я

NewNikita12

Прізвище

Mailjetov

Номер телефону

380931355223

Email

triathlet.52@gmail.com

Зберегти

Рис. 3.15 Особисті дані

Щоб видалити або додати платіжну картку, потрібно обрати пункт “Платіжна картка”, після додавання картки користувач отримує можливість купувати квитки на заходи.

Схоже у вас наразі немає картки

Введіть дані картки

Номер картки:

2341342111234532

Власник:

VAKULOVYCH NIKITA

Дійсна до:

12/24

CVV

000

Додати картку

Рис. 3.16 Платіжна картка

Після вдалого додавання платіжна картка виводиться в тій же вкладці, у користувача є можливість видалити та заново додати картку.

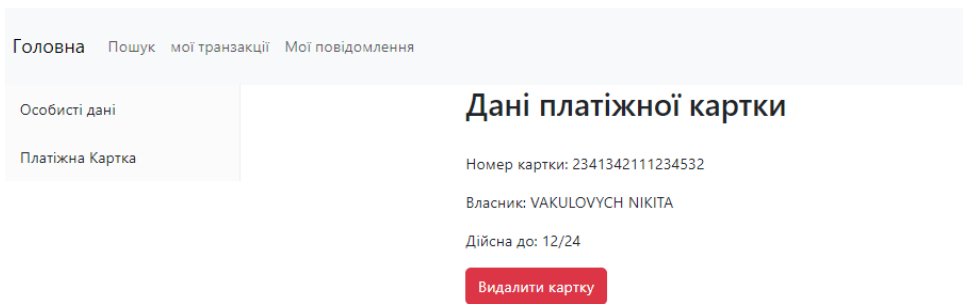


Рис. 3.17 Додана платіжна картка

Задля зміни ролі користувач може використати кнопку “Стати організатором”, в результаті отримаємо повідомлення про успішний запит

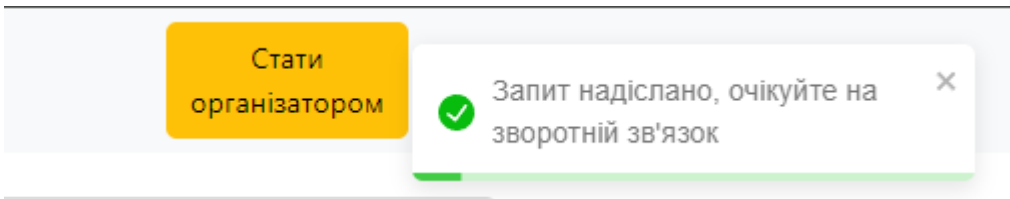


Рис. 3.18 Успішний запит на зміну ролі

В результаті успішного запиту випадковий адміністратор отримає повідомлення, про запит

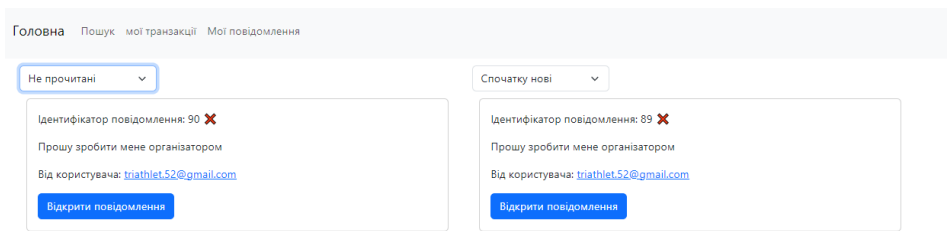


Рис. 3.19 Повідомлення на стороні адміністратора

Після отримання повідомлення адміністратор може перейти на сторінку користувача, переглянути його контактні дані та змінити йому роль на «Адміністратор» або «Організатор»

Дані користувача

Ім'я: NewNikita12

Роль: користувач

Прізвище: Mailjetov

Номер телефону: 380931355223

Email: triathlet.52@gmail.com

Зробити організатором

Зробити адміністратором

Рис. 3.20 Сторінка користувача

Також адміністратор може повернути йому роль користувача

Дані користувача

Ім'я: NewNikita12

Роль: організатор

Прізвище: Mailjetov

Номер телефону: 380931355223

Email: triathlet.52@gmail.com

Зробити користувачем

Рис. 3.21 Сторінка користувача, зі зміненою роллю

Після отримання ролі «Організатор», користувач отримує можливість створювати власні заходи, в кабінеті користувача, для цього передбачено «Інструменти організатора», в кабінеті користувача

Головна

Пошук

Міні транзакції

Міні повідомлення

Особисті дані

Платіжна Картка

Інструменти організатора

Міні заходи

Створити захід

Особисті дані

унікальний ідентифікатор: 1

Роль: організатор

Ім'я

NewNikita11

Прізвище

Mailjetov

Номер телефону

380931355223

Email

triathlet.52@gmail.com

Зберегти

Ви авторизувались як triathlet.52@gmail.com

Вийти

Рис. 3.22 Оновлений кабінет користувача

Задля створення заходу організатору необхідно перейти на сторінку «Створити захід»,

Створити захід

Назва

Ціна

Доступні місця

Дата початку

mm/dd/yyyy --:--

Дата закінчення

mm/dd/yyyy --:--

Статус

Очікується

Вибір закладу

Пошук закладу ✖

Введіть назву закладу

Пошук

Опис

Тип події

Створити

Рис. 3.23 Створення заходу

Після вдалого створення заходу користувача перенаправляє на сторінку редагування, на ній організатор може редагувати дані та додавати фото

Змагання

Вибір фото для завантаження:

Choose File 1670470450_1-kartinkin-net-p-triatlon-kartinki-krasivo-1.jpg

Опис

Завантажити

фото 1

ГОЛОВНЕ ✓

фото 2

ГОЛОВНЕ ✓

Оновити

Видалити

Рис. 3.24 редагування заходу

Після цього адміністратор може переглянути захід та опублікувати його або ж відправити на доопрацювання організатором, обравши вкладку «Заходи на перевірку» в панелі адміністратора

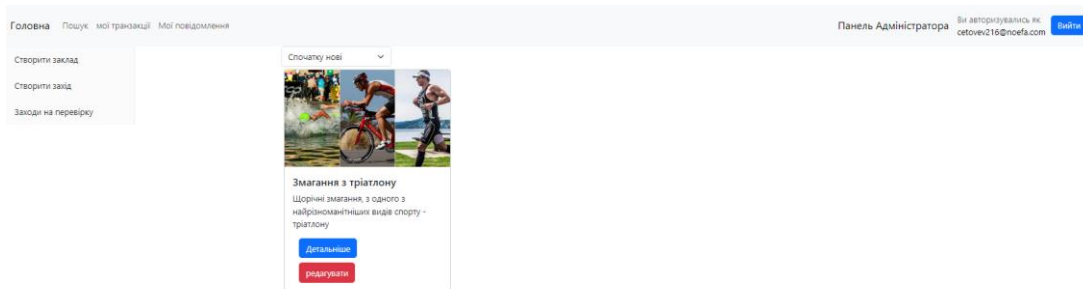


Рис. 3.25 «Заходи на перевірку»

Організатор, у свою чергу, може переглядати статус свого заходу у вкладці «Мої заходи», в «Інструменти організатора».

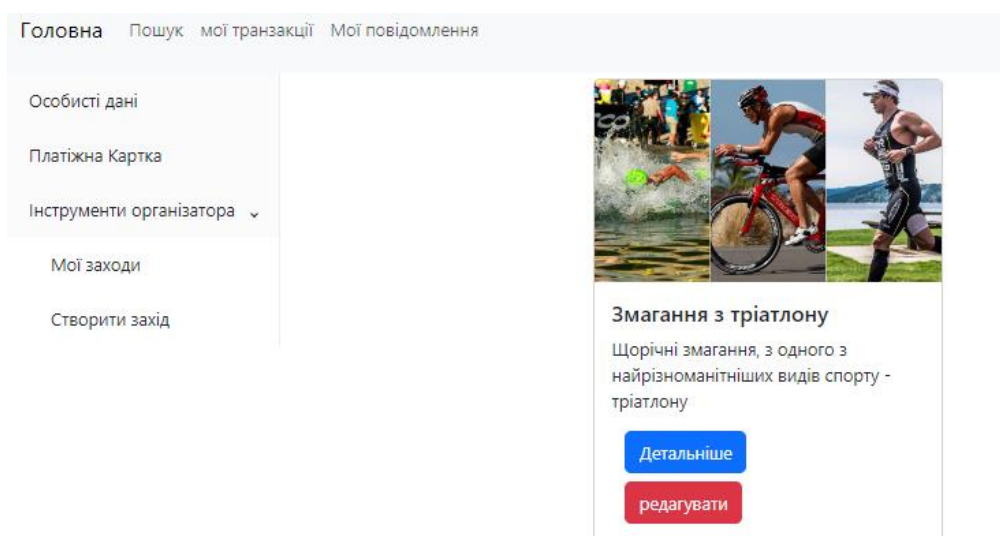


Рис. 3.26 «Мої заходи»

Після підтвердження заходу, він буде доступний до пошуку та користувачі зможуть придбати квитки на нього, розглянемо процес придбання квитку користувачем. На сторінці заходу у користувача присутня кнопка «Придбати квиток», при натиску на яку з'являється модальне вікно для придбання квитку

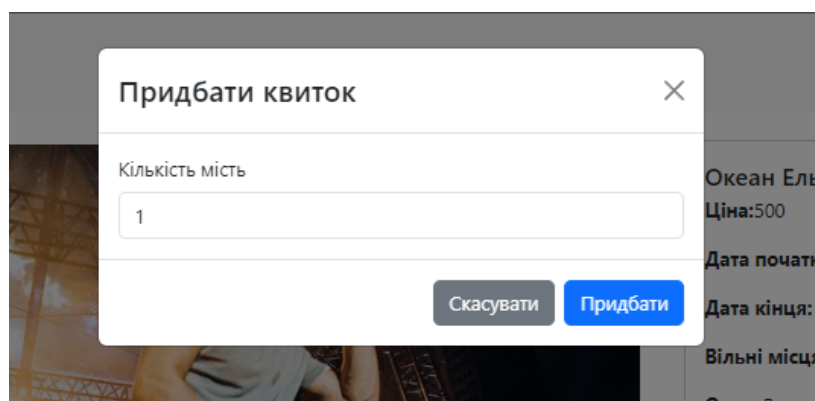


Рис. 3.27 Вікно для придбання квитку

Обравши потрібно кількість користувач може натиснути кнопку Придбати, після чого, у разі наявності платіжної картки, його буде перенаправлено на сторінку для оплати замовлення, дані про картку та сума вже вказані

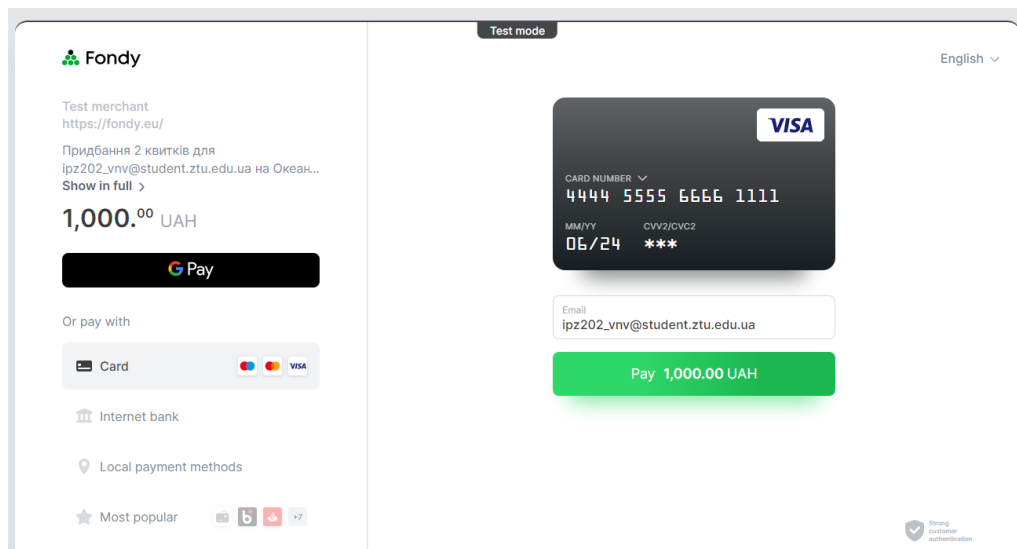


Рис. 3.28 Сторінка для оплати

Після вдалої оплати користувач отримує на пошту лист з унікальним qr-кодом, в якому зберігається посилання, перейшовши за яким організатор зможе перевірити валідність квитку та підтвердити особу замовника.



Рис. 3.29 Лист



Рис. 3.30 Результат сканування QR-коду

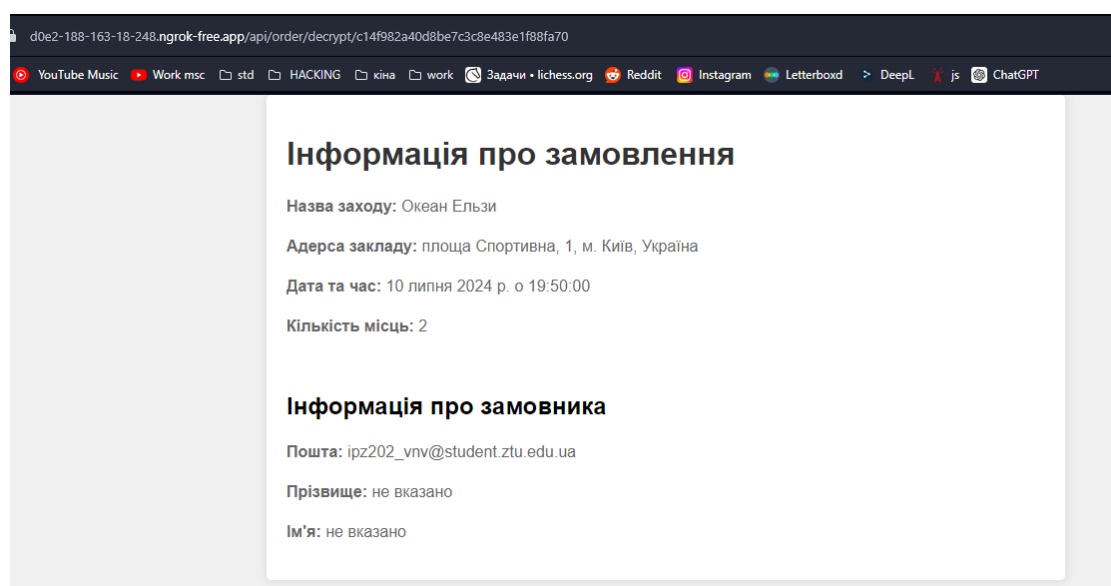


Рис. 3.31 Сторінка, на яку веде QR-код

3.3 Тестування роботи веб-платформи

Мануальне тестування є критично важливим етапом забезпечення якості програмного забезпечення. У даному документі представлено сценарії тестування для ключових функцій веб-додатку, зокрема, реєстрації користувача та додавання закладу. Ці функції є основними для взаємодії користувачів з системою та забезпечення її основної функціональності.

Мета тестування полягає у виявленні можливих помилок, недоліків та невідповідностей у роботі додатку, щоб забезпечити його надійність, зручність

					ІПЗ.КР.Б-121-24-ПЗ	
						62

та безпеку для кінцевих користувачів. Докладні сценарії тестування допоможуть виявити потенційні проблеми та покращити загальну якість продукту перед його запуском у виробництво. Сценарії тестування веб-додатку:

Реєстрація

Сценарій 1: Успішна реєстрація

- 1. Відкрити сторінку реєстрації.
- 2. Заповнити поле «Електронна пошта» валідною адресою електронної пошти (наприклад, «olena@example.com»).
- 3. Заповнити поле «Пароль» валідним паролем (наприклад, "Password123").
- 4. Натиснути кнопку «Зареєструватися».
- 5. Очікуваний результат: Користувач перенаправляється на сторінку сторінку входу з повідомленням про успішну реєстрацію.

Сценарій 2: Реєстрація з невалідною електронною поштою

- 1. Відкрити сторінку реєстрації.
- 2. Заповнити поле «Електронна пошта» невалідною адресою електронної пошти (наприклад, «olenaexample.com»).
- 3. Заповнити поле «Пароль» валідним паролем (наприклад, «Password123»).
- 4. Натиснути кнопку «Зареєструватися».
- 5. Очікуваний результат: Відображається повідомлення про помилку «Невалідна адреса електронної пошти».

Додавання закладу

Сценарій 1: Успішне додавання закладу

- 1. Увійти в систему за допомогою валідних даних.
- 2. Перейти до сторінки додавання закладу.
- 3. Заповнити поля валідними даними
- 4. Завантажити фото закладу.
- 5. Натиснути кнопку «Додати заклад».

6.Очікуваний результат: Заклад успішно додано, користувач перенаправляється на сторінку перегляду закладу з повідомленням про успішне додавання.

Сценарій 2: Додавання закладу з пустими обов'язковими полями

1. Увійти в систему за допомогою валідних даних.
2. Перейти до сторінки додавання закладу.
3. Заповнити лише частину обов'язкових полів або залишити їх пустими.
4. Натиснути кнопку «Додати заклад».

5. Очікуваний результат: Відображається повідомлення про помилку з зазначенням обов'язкових для заповнення полів.

Сценарій 3: Додавання закладу з невалідною адресою

1. Увійти в систему за допомогою валідних даних.
2. Перейти до сторінки додавання закладу.
3. Заповнити всі поля, окрім «Адреса» .
4. Заповнити поле «Адреса» невалідними даними (наприклад, «123»).
5. Натиснути кнопку «Додати заклад».

6. Очікуваний результат: Відображається повідомлення про помилку «Невалідна адреса».

Висновки до третього розділу

У цьому розділі була ретельно описана схема розгортання платформи, включаючи докладний аналіз кожного структурного елемента системи і його взаємодію з іншими компонентами платформи. Цей підхід дозволив отримати глибоке розуміння архітектури системи та взаємозв'язків між її складовими частинами.

Далі було надано опис інтерфейсу розробленого програмного продукту, що включав розгляд всіх функціональних можливостей додатку, адаптованих до ролі кожного типу користувача. Кожен користувач отримав доступ до унікальних функцій та можливостей в системі, що сприяє ефективному використанню додатку з урахуванням індивідуальних потреб.

Також було продемонстровано, як різні інтерфейси, залежно від ролі користувача, впливають на доступність статистичної інформації для адміністраторів та звичайних користувачів.

ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи було розроблено та реалізовано платформу для онлайн-продажу квитків на різноманітні заходи. Опрацьовуючи предметну область, було проведено глибокий аналіз існуючих платформ конкурентів, які також спеціалізуються на продажу квитків. На основі цього аналізу визначені основні вимоги до нової платформи.

Проектування включало розробку архітектури системи, моделювання взаємодії між її компонентами та проектування бази даних для ефективного зберігання інформації. Описано алгоритм роботи з додатком та визначено послідовні кроки його використання. Основний функціонал включає можливість створення, підтвердження та публікацію заходів, а також інтерактивну взаємодію користувачів з платформою.

Детально розглянута схема розгортання платформи включає взаємодію всіх структурних елементів системи, що забезпечує її стабільну та ефективну роботу. Описано інтерфейс розробленого програмного продукту, який забезпечує інтуїтивно зрозуміле використання для різних типів користувачів. Користувачі можуть легко створювати заходи, взаємодіяти з ними, залишати коментарі та придбати необхідні квитки.

Цей проект демонструє не лише технічну компетентність, а й високий рівень уваги до потреб користувачів у сфері онлайн-продажу квитків на події різного масштабу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Javascript.info [Електронний ресурс] – Режим доступу до ресурсу:
<https://uk.javascript.info>
2. React [Електронний ресурс] – Режим доступу до ресурсу:
<https://uk.legacy.reactjs.org>
3. Node.js [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodejs.org/docs/latest/api/>
4. Amazone S3 [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.aws.amazon.com/s3/>
5. Express [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com>
6. Node-cron [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodecron.com/docs/>
7. Mailjet [Електронний ресурс] – Режим доступу до ресурсу:
<https://dev.mailjet.com/email/reference/overview/>
8. TypeScript [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>
9. JWT [Електронний ресурс] – Режим доступу до ресурсу:
<https://jwt.io/introduction>
10. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.postgresql.org/docs/>
11. Qrcode [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/qrcode>
12. Typeorm [Електронний ресурс] – Режим доступу до ресурсу: <https://typeorm.io>
13. Yup [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/yup/v/1.0.0-alpha.3>
14. React starter kit [Електронний ресурс] – Режим доступу до ресурсу:
<https://blog.glitch.com/post/react-starter-kit/>
15. Learn React [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.codecademy.com/learn/react-101>

- 16.Node http [Электронный ресурс] – Режим доступа до ресурсу:
<https://nodejs.org/docs/latest/api/http.html>
- 17.Express Router [Электронный ресурс] – Режим доступа до ресурсу:
<https://expressjs.com/en/4x/api.html#router>
- 18.QR Code Scanner [Электронный ресурс] – Режим доступа до ресурсу:
<https://qrcoderaptor.com>
- 19.React Bootstrap [Электронный ресурс] – Режим доступа до ресурсу: <https://react-bootstrap.netlify.app>
- 20.Typeorm Working with DataSource [Электронный ресурс] – Режим доступа до ресурсу: <https://typeorm.io/data-source>
- 21.Karabas [Электронный ресурс] – Режим доступа до ресурсу:
<https://karabas.com/ua/>
- 22.The Beginner's Guide to React [Электронный ресурс] – Режим доступа до ресурсу:
<https://egghead.io/courses/the-beginner-s-guide-to-react>
- 23.HTML [Электронный ресурс] – Режим доступа до ресурсу:
<https://developer.mozilla.org/en-US/docs/Web/HTML>
- 24.Express Response [Электронный ресурс] – Режим доступа до ресурсу:
<https://expressjs.com/en/4x/api.html#res>
- 25.Testing Documentation [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.javatpoint.com/testing-documentation>
- 26.Class Diagrams [Электронный ресурс] – Режим доступа до ресурсу:
<https://centria.github.io/design-and-documentation/part2/>
- 27.React Bootstrap Components [Электронный ресурс] – Режим доступа до ресурсу:
<https://react-bootstrap.netlify.app/docs/components/accordion>
- 28.Express Hello world [Электронный ресурс] – Режим доступа до ресурсу:
<https://expressjs.com/en/starter/hello-world.html>
- 29.Typeorm Using Validation [Электронный ресурс] – Режим доступа до ресурсу:
<https://typeorm.io/validation>
- 30.draw.io [Электронный ресурс] – Режим доступа до ресурсу: draw.io

- 31.kontramarka [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.kontramarka.com/ua/>
- 32.ticketmaster [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.ticketmaster.com/>
- 33.StubHub [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.stubhub.ie/>

ДОДАТКИ

					ІІЗ.КР.Б-121-24-ІІЗ	
						70

```

import { Response } from "express";
import { UserRequest } from "../interfaces/UserRequest";
import asyncHandler from "express-async-handler";
import {
  eventCreateSchema,
  eventForPublication,
  eventSearchSchema,
  eventUpdateSchema,
} from "../validators/eventValidator";
import { AppDataSource } from "../data-source";
import { Event, eventStatus } from "../entity/event";
import { Venue } from "../entity/venue";
import { User, UserRole } from "../entity/user";
import { deleteEventCascade } from "../scripts/entityDelete";
import { AddPhotoSchema } from "../validators/venueValidator";
import fs from "fs";
import { photo_event } from "../entity/photo_event";
import { deleteFileAWS, uploadFile } from "../scripts/aws_s3";
import { token_info } from "../scripts/scripts";
import { UserTokenInfo } from "../interfaces/UserTokenInfo";

// Get repositories from the data source
const eventRepository = AppDataSource.getRepository(Event);
const venueRepository = AppDataSource.getRepository(Venue);
const userRepository = AppDataSource.getRepository(User);
const eventPhotoRepository = AppDataSource.getRepository(photo_event);

// Handler for creating an event
export const eventCreate = asyncHandler(
  async (req: UserRequest, res: Response) => {
    let {
      name,
      price,
      date_start,
      date_end,
      description,
      event_type,
      available_places,
      venue_id,
    } = req.body;

    try {
      await eventCreateSchema.validate({
        role: req.user.role,
        venue_id,
        name,
        price,
        date_start,
        date_end,
        description,

```

```

        available_places,
        event_type,
    });
} catch (err) {
    res.status(400);
    throw new Error(err.message);
}

try {
    const Venue = await venueRepository.findOneBy({ id: venue_id });
    const user = await userRepository.findOneBy({ id: req.user.id });
    if (!Venue) {
        res.status(404);
        throw new Error("Заклад не знайдено");
    }

    // Create a new event
    let new_event: Event = await eventRepository.create({
        name,
        price,
        date_start: new Date(date_start),
        date_end: new Date(date_end),
        description,
        event_type,
        venue: Venue,
        available_places,
        created_by: user,
    });
    new_event = await eventRepository.save(new_event);
    res.json({ event_id: new_event.id });
} catch (err) {
    if (res.statusCode === 200) {
        res.status(500);
    }
    throw new Error(err.message);
}
}
);

// Handler for getting an event by ID
export const eventGet = asyncHandler(
    async (req: UserRequest, res: Response) => {
        // Extract event ID from the request parameters
        const event_id: number = parseInt(req.params.id);

        if (!event_id) {
            res.status(400);
            throw new Error("Ідентифікатор заходу помилковий");
        }

        // Get user information from the token

```

```

let user: UserTokenInfo = await token_info(req);
let Event: Event = await eventRepository.findOne({
  where: { id: event_id },
  relations: { created_by: true, venue: true },
  select: {
    created_by: { id: true },
    venue: { id: true, name: true, address: true },
  },
});
if (!Event) {
  res.status(404);
  throw new Error("захід не знайдено");
}

// If the user is not an admin, remove the created_by field from the response
if (user?.role !== UserRole.ADMIN) {
  delete Event["created_by"];
}

res.status(200).json(Event);
}
);

export const eventGetForPublication = asyncHandler(
  async (req: UserRequest, res: Response) => {
    let { sortorder } = req.query;
    console.log(sortorder);
    if (req.user.role !== UserRole.ADMIN) {
      res.status(403);
      throw new Error("Користувач не має права отримати дану інформацію");
    }
    try {
      await eventForPulication.validate({
        sortOrder: sortorder,
      });
    } catch (err) {
      res.status(400);
      throw new Error(err.message);
    }
    let events: Event[] = await eventRepository.find({
      where: { status: eventStatus.PENDING },
      order: { date_start: sortorder },
      relations: { photos: true, created_by: true },
    });

    // If the user is not an admin, remove the created_by field from the response

    res.status(200).json(events);
  }
);

export const eventSetPhotoMain = asyncHandler(

```

```

async (req: UserRequest, res: Response) => {
  // Extract event ID from the request parameters
  const mainPhoto_id: number = parseInt(req.params.id);

  // If the user is not an admin, remove the created_by field from the response

  let PhotoEvent: Event = await eventRepository.findOne({
    where: { photos: { id: mainPhoto_id } },
    relations: { photos: true },
    select: { id: true, photos: { id: true } },
  });
  let Event: Event = await eventRepository.findOne({
    where: { id: PhotoEvent.id },
    relations: { photos: true, created_by: true },
    select: { created_by: { id: true }, photos: true },
  });

  if (!Event) {
    res.status(404);
    throw new Error("Захід не знайдено");
  }
  if (
    req.user?.role !== UserRole.ADMIN &&
    req.user.id !== Event.created_by.id
  ) {
    res.status(403);
    throw new Error("Користувач не має права редагувати дані");
  }
  Event.photos = Event.photos.map((photo) => {
    photo.isMain = photo.id === mainPhoto_id;
    return photo;
  });

  eventPhotoRepository.save(Event.photos);
  res.status(200).json({ message: "Головне фото оновлено" });
}
);
// Handler for updating an event
export const eventUpdate = asyncHandler(
  async (req: UserRequest, res: Response) => {
    const {
      name,
      price,
      date_start,
      date_end,
      description,
      status,
      event_type,
      venue_id,
    } = req.body;
    const event_id: number = parseInt(req.params.id);

```

```

let venue: Venue;

try {
  // Validate the update request data
  await eventUpdateSchema.validate({
    role: req.user.role,
    venue_id,
    event_id,
    name,
    price,
    date_start,
    date_end,
    description,
    event_type,
    status,
  });
} catch (err) {
  res.status(400);
  throw new Error(err.message);
}
try {
  const event = await eventRepository.findOne({
    where: { id: event_id },
    relations: { venue: true },
  });

  if (!event) {
    res.status(404);
    throw new Error("захід не знайдено");
  }
  const user = await userRepository.findOneBy({ id: req.user.id });

  if (!user) {
    res.status(404);
    throw new Error("користувача не знайдено");
  }

  // If a new venue ID is provided, find and update the venue
  if (venue_id) {
    venue = await venueRepository.findOneBy({ id: venue_id });
    if (!venue) {
      res.status(404);
      throw new Error("заклад не знайдено");
    }
    event.venue = venue;
  }
  // Update event attributes based on the provided data
  event.name = name || event.name;
  event.price = price || event.price;
  event.venue = venue;
  event.date_start = date_start ? new Date(date_start) : event.date_start;

```

```

    event.date_end = date_end ? new Date(date_end) : event.date_end;
    event.description = description || event.description;
    event.event_type = event_type || event.event_type;
    event.status = status || event.status;

    if (req.user.role === UserRole.ORGANIZER) {
        event.status = eventStatus.PENDING;
    }

    await eventRepository.save(event);
    res.status(200).json({ status: "інформацію оновлено" });
} catch (err) {
    if (res.statusCode === 200) {
        res.status(500);
    }
    throw new Error(err);
}
}
);

// Handler for deleting an event
export const eventDelete = asyncHandler(
    async (req: UserRequest, res: Response) => {
        const event_id: number = parseInt(req.params.id);

        // Find the event by ID with creator information
        let Event: Event = await eventRepository.findOne({
            where: { id: event_id },
            relations: { created_by: true },
            select: { id: true, created_by: { id: true } },
        });

        // Check if the current user has permission to delete the event
        if (
            req.user.id !== Event.created_by.id &&
            req.user.role !== UserRole.ADMIN
        ) {
            res.status(403);
            throw new Error("Видаляти захід може лише організатор та адміністратор");
        }
        if (!Event) {
            res.status(404);
            throw new Error("заклад не знайдено");
        }

        try {
            // Perform cascade deletion of the event
            await deleteEventCascade(event_id);
            res.status(200).json({ message: "Захід видалено" });
        } catch (err) {
            res.status(500);
        }
    }
);

```

```

        throw new Error(err);
    }
}
);

// Handler for getting events created by a user
export const eventGetByUser = asyncHandler(
    async (req: UserRequest, res: Response) => {
        const user_id: number = parseInt(req.params.id);

        // Check if the current user has permission to view events created by the
        // specified user
        if (req.user.role !== UserRole.ADMIN && req.user.id !== user_id) {
            res.status(403);
            throw new Error("Ця інформація доступна тільки адміністратору та автору");
        }
        const organizer: User = await userRepository.findOneBy({
            id: user_id,
        });
        if (!organizer) {
            res.status(404);
            throw new Error("Користувача не знайдено");
        }

        let Event: Event[] = await eventRepository.find({
            where: { created_by: organizer },
            relations: { created_by: true, photos: true },
            select: { created_by: { id: true } },
        });
        res.status(200).json(Event);
    }
);

// Handler for adding a photo to an event
export const eventAddPhoto = asyncHandler(
    async (req: UserRequest, res: Response) => {
        const photo = req.file;
        const { description } = req.body;
        const event_id: number = parseInt(req.params.id);

        const Event = await eventRepository.findOne({
            where: { id: event_id },
            relations: { photos: true },
        });
        try {
            // Check if the event exists and has not reached the maximum photo count
            if (!Event) {
                res.status(404);
                throw new Error("заклад не знайдено");
            } else if (Event.photos.length > 10) {
                res.status(409);
                throw new Error("досягнуто максимальної кількості фото");
            }
        }
    }
);

```

```

    }
    // Validate the request data
    await AddPhotoSchema.validate({
      photo,
      role: req.user.role,
      id: event_id,
      description,
    });
  } catch (err) {
    res.status(400);
    fs.unlinkSync(photo.path);
    throw new Error(err.message);
  }
  // Upload the photo to AWS S3
  const result = await uploadFile(photo);
  fs.unlinkSync(photo.path);

  // Create a new photo event entity
  let photo_event = await eventPhotoRepository.create({
    description,
    image_key: photo.filename,
    event: Event,
    isMain: Event.photos.length === 0,
  });
  const photo_result = await eventPhotoRepository.save(photo_event);
  res.json(photo_result);
}
);

// Handler for getting photos of an event
export const eventGetPhoto = asyncHandler(
  async (req: UserRequest, res: Response) => {
    const event_id: number = parseInt(req.params.id);
    if (!event_id) {
      res.status(400);
      throw new Error("id invalid");
    }
    const event = await eventRepository.findOne({
      where: { id: event_id },
    });
    if (!event) {
      res.status(404);
      throw new Error("заклад не найдено");
    }
    // Get the list of photos for the event
    const photo_list = await eventPhotoRepository.find({
      where: { event },
    });
    res.status(200).json({ photo_list });
  }
);

```

```

// Handler for deleting a photo from an event
export const eventDeletePhoto = asyncHandler(
  async (req: UserRequest, res: Response) => {
    const eventPhoto_id: number = parseInt(req.params.id);
    if (!eventPhoto_id) {
      res.status(400);
      throw new Error("ідентифікатор помилковий");
    }

    // Find the photo event by ID with related information
    const Photo = await eventPhotoRepository.findOne({
      where: { id: eventPhoto_id },
      relations: ["event", "event.created_by"],
    });
    if (Photo.isMain) {
      res.status(400);
      throw new Error(
        "Головне фото не видаляється, оберіть інше головне фото спочатку"
      );
    }
    // Check if the user has the rights to delete the photo
    if (
      req.user.role !== UserRole.ADMIN &&
      req.user.id === Photo.event.created_by.id
    ) {
      res.status(403);
      throw new Error("користувач не має прав");
    }
    if (!Photo) {
      res.status(404);
      throw new Error("фото відсутні");
    }
    // Delete the photo from AWS S3
    deleteFileAWS(Photo.image_key);
    await eventPhotoRepository.remove(Photo);
    res.status(200).json({ status: "фото видалено" });
  }
);

// Handler for searching events based on specified criteria
export const eventSearch = asyncHandler(
  async (req: UserRequest, res: Response) => {
    let {
      address,
      name,
      price_min,
      date_start,
      price_max,
      date_end,
      event_type,
    }

```

```

    sortBy, // New parameter for sorting
    sortOrder, // New parameter for sort order
    page_number,
  } = req.query;
  // Extract pagination parameters
  const take: number = parseInt(req.query.take) || 8;
  const skip: number = parseInt(page_number) * take || 0;

  try {
    // Validate search criteria
    await eventSearchSchema.validate({
      skip,
      take,
      address,
      name,
      price_min,
      date_start,
      price_max,
      date_end,
      event_type,
      sortBy,
      sortOrder,
      page_number,
    });

    // Create a query builder for events
    const EventsQuery = eventRepository.createQueryBuilder("event");
    EventsQuery.innerJoinAndSelect(
      "event.photos",
      "photo",
      "photo.isMain = true"
    );
    // Apply search criteria to the query
    EventsQuery.andWhere("event.status = :status", {
      status: eventStatus.CONFIRMED,
    });

    if (name) {
      EventsQuery.andWhere("event.name ILIKE :q", { q: `%${name}%` });
    }
    if (address) {
      EventsQuery.leftJoinAndSelect("event.venue", "venue").andWhere(
        "venue.address ILIKE :q",
        { q: `%${address}%` }
      );
    }
    if (price_min) {
      EventsQuery.andWhere("price >= :price_min", { price_min });
    }

    if (price_max) {

```

```

        EventsQuery.andWhere("price <= :price_max", { price_max });
    }
    if (date_start) {
        EventsQuery.andWhere("date_start >= :date_start", { date_start });
    }
    if (date_end) {
        EventsQuery.andWhere("date_start <= :date_end", { date_end });
    }
    // Apply sorting criteria
    if (sortBy) {
        const sortOrderValue = sortOrder === "desc" ? "DESC" : "ASC";
        EventsQuery.orderBy(`event.${sortBy}`, sortOrderValue);
    }
    // Apply pagination parameters
    EventsQuery.skip(skip).take(take);
    // Execute the query and get the results
    const result = await EventsQuery.getMany();
    const totalCount = await EventsQuery.getCount();

    res.json({ result, totalCount });
} catch (err) {
    if (err.name === "ValidationError") {
        res.status(400);
    } else {
        res.status(500);
    }
    throw new Error(err.message);
}
}
);

```