НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
Олександр ПАВЛОВ
(підпис) (ініціали, прізвище)
“ ” 2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення комп’ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему *«Програмне забезпечення підтримки концертної діяльності музичних колективів»*

Виконав: студент IV курсу, групи *ІП-71 Кувічка Максим Євгенович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ас. Черненький Артем Юрійович* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук Катерина Ігорівна* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:
ст. вик. Шемсединов Тимур Гафарович
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – *121 Інженерія програмного забезпечення*
Освітньо-професійна програма – *Програмне забезпечення комп’ютеризованих систем*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Кувічці Максиму Євгеновичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Програмне забезпечення підтримки концертної діяльності музичних колективів»*

керівник проєкту *Черненький Артем Юрійович ас.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “11” травня 2021 р. №1139-с

2. Термін подання студентом проєкту «08» *червня 2021 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог, постановка задачі

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення

3) Аналіз якості та тестування програмного забезпечення

4) Розгортання та впровадження програмного забезпечення

5. Перелік графічного матеріалу

1) Креслення вигляду екранних форм

2) Схема структурна варіантів використань

3) Схема бізнес-процесу перевірки дійсності квитка

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «14» *березня 2021 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*01.02.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*10.02.2021*	
3.	*Постановка та формалізація задачі*	*14.02.2021*	
4.	*Аналіз вимог до програмного забезпечення*	*17.02.2021*	
5.	*Моделювання програмного забезпечення*	*26.02.2021*	
6.	*Обґрунтування використовуваних технічних засобів*	*03.03.2021*	
7.	*Розробка архітектури програмного забезпечення*	*10.03.2021*	
8.	*Розробка програмного забезпечення*	*25.03.2021*	
9.	*Налагодження програми*	*15.04.2021*	
10.	*Виконання графічних документів*	*28.04.2021*	
11.	*Оформлення пояснювальної записки*	*01.05.2021*	
12.	*Подання ДП на попередній захист*	*17.05.2021*	
13.	*Подання ДП рецензенту*	*02.06.2021*	
14.	*Подання ДП на основний захист*	*08.06.2021*	

Студент _____ Максим КУВІЧКА
(підпис)

Керівник _____ Артем ЧЕРНЕНЬКИЙ
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-7112.045490.01.90	Відомість дипломного проєкту	1	
3	А4	КПІ.ІП-7112.045490.02.81	Пояснювальна записка	69	
4	А4	КПІ.ІП-7112.045490.03.34	Керівництво користувача	10	
5	А4	КПІ.ІП-7112.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-7112.045490.05.91	Технічне завдання	11	
7	А4	КПІ.ІП-7112.045490.06.13	Опис програми	29	
8	А3	КПІ.ІП-7112.045490.07.99	Графічні матеріали	3	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-7112.045490.01.90		
Розробн.		Кувічка М.Є.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Черненький А.Ю.					
Консульт.						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-71	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту містить 69 сторінок, 9 рисунків, 27 таблиць, 10 джерел.

Цей дипломний проєкт націлений на полегшення процесу підтримки концертної діяльності не популярним €музичним колективам, які не мають менеджерів та будь-якого іншого персоналу, окрім безпосередньо музикантів, тобто займаються організаційною діяльністю концертів особисто. Дані процеси можуть бути спрощені шляхом створення частково автоматизованої централізованої системи для організації та узгодження підготовки події концерту.

Головною метою розробки є економія часу для організаторів, економія фінансів організаторів, надання можливості пошуку подій поблизу відвідувача та зменшення часу на пошук та купівлю квитка.

У розділі аналізу вимог до програмного забезпечення був проведений аналіз предметної області та розглянуті основні сценарії використання розроблюваного програмного продукту. На базі цих даних були виведені відповідні вимоги та рішення поставленої задачі.

У розділі моделювання та конструювання програмного забезпечення був проведений аналіз оптимальної архітектури розробленого програмного забезпечення та на базі цього був вибраний технологічний стек. Проведено аналіз безпеки даних та системи в цілому.

У розділі аналіз якості та тестування програмного забезпечення було описане і проведене детальне тестування, що дозволило знайти проблеми в роботі та виправити їх.

КЛЮЧОВІ СЛОВА: МУЗИЧНІ КОЛЕКТИВИ, КОНЦЕРТ, КВИТОК, АВТОМАТИЗАЦІЯ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project contains 69 pages, 9 figures, 27 tables, 10 sources.

This diploma project aims to facilitate the process of supporting concert activities for unpopular music groups that do not have managers and any other staff, except directly musicians, ie are engaged in the organizational activities of concerts in person. These processes can be simplified by creating a partially automated centralized system for organizing and coordinating the preparation of a concert event.

The main purpose of the development is to save time for the organizers, save the finances of the organizers, provide the ability to search for events near the visitor and reduce the time to search and buy a ticket.

In the section of analysis of software requirements the analysis of the subject area was carried out and the basic scenarios of use of the developed software product were considered. Based on these data, the relevant requirements and solutions to the problem were derived.

In the section of software modeling and design, the analysis of the optimal architecture of the developed software was performed and on the basis of this the technological stack was selected. The analysis of data security and the system as a whole is carried out.

The section of quality analysis and software testing described and conducted detailed testing, which allowed to find problems in the work and fix them.

KEY WORDS: MUSIC BANDS, CONCERT, TICKET, AUTOMATION.

Пояснювальна записка
до дипломного проєкту

на тему: «*Програмне забезпечення підтримки концертної діяльності музичних колективів*»

Київ – 2021 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ *11*

ВСТУП *12*

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ *15*

1.1 Загальні положення **15**

1.2 Змістовний опис і аналіз предметної області **15**

1.3 Аналіз успішних ІТ-проєктів **17**

1.4 Аналіз вимог до програмного забезпечення **20**

1.4.1 Розроблення функціональних вимог 34

1.4.2 Розроблення нефункціональних вимог 35

1.4.3 Постановка комплексу завдань 36

1.5 Висновки до розділу **37**

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ *38*

2.1 Моделювання та аналіз програмного забезпечення **38**

2.2 Архітектура програмного забезпечення **39**

2.2.1 Опис використаних рішень 39

2.2.2 Опис сховища даних 41

2.2.3 Опис модулів клієнтської частини системи 46

2.2.4 Опис модулів серверної частини системи 50

2.3 Аналіз безпеки даних **55**

2.4 Висновки до розділу **56**

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ *57*

3.1 Аналіз якості програмного забезпечення 57

3.2 Опис процесу тестування 57

3.3 Опис контрольного прикладу 58

3.4 Висновки до розділу 63

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 64

4.1 Розгортання програмного забезпечення 64

4.2 Робота з програмним забезпеченням 65

4.3 Висновки до розділу 65

ВИСНОВКИ 66

ПЕРЕЛІК ПОСИЛАНЬ 67

ДОДАТОК А ЗВІТ ПРО ПЕРЕВІРКУ ПОДІБНОСТІ 68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Концерт – прилюдне виконання музичних творів, балетних, естрадних і т. ін. номерів за певною програмою.

JWT (JSON Web Token) – запропонований стандарт для створення даних із необов'язковим підписом та/або необов'язковим шифруванням, який базується на JSON даних.

API (Application Proggramming Interface) – визначені інтерфейси, за допомогою яких відбувається взаємодія між сервером та клієнтами.

Хешування – перетворення вхідних даних будь-якого розміру в дані фіксованого розміру.

KDF-функція – функція, що формує один або кілька секретних ключів на основі секретного значення за допомогою псевдовипадкової функції.

Фреймворк – програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту.

Express.js – веб-фреймворк Node.js для створення ефективних, надійних та масштабованих серверних додатків.

Docker – набір PaaS продуктів, які використовують віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, званих контейнерами.

ВСТУП

У наш час сценічна діяльність є відкритою для всіх охочих, тож потрапити на велику сцену може дійсно кожен. Ми маємо купу талант-шоу, конкурсів, програм, вебінарів та лекцій, де шукають виконавців будь-якого віку з будь-якими музичними вподобаннями, на яких потім майбутніх артистів навчають не тільки розвиватись у творчому плані, а й вести власну підприємницьку діяльність як колектива.

На жаль, далеко не всі талановиті особистості в результаті збирають мільйонну аудиторію - більшість із них так і залишається відомою тільки для невеликої кількості поціновувачів такої творчості. Проте виконавці все одно прагнуть організовувати певні зустрічі з фанатами, концерти та інші публічні заходи.

Хоча в наші дні придбати квитки на виступ улюбленого виконавця не є проблемою, адже кожен знає популярні сервіси в мережі Інтернет, які займаються рекламою, продажем квитків та інформуванням користувачів щодо аспектів концерту, розмістити своє оголошення на таких сайтах досить проблематично. На таких веб-ресурсах зовсім відсутня автоматизація подачі заяв на публікацію користувацької події, а єдиним можливим шляхом є прямий контакт з адміністраторами і вирішення цього питання в особистому листуванні. З цього випливає ще одна, навіть більша проблема - недостатня швидкість обробки таких листів і відсутність гарантії отримання відповіді в цілому, адже листування поштою не дає ніякого відгуку: чи потрапило повідомлення до отримувача, чи прочитав він надісланий запит тощо.

Все це вимальовується в ситуацію, коли розмістити свою афішу на одному з таких онлайн-сервісів дійсно може або виконавець, у якого є спеціальний маркетинговий відділ, що витрачає купу часу на подібні організаційні моменти, або виконавець, особисто знайомий з одним із адміністраторів. Проте обидва ці варіанти виключають доступність для

місцевих груп з невеликою аудиторією і концертами не на стадіонах, а в кафе, будинках культури, просто неба тощо.

Також досить популярними є доволі застарілі методи на кшталт розклеювання оголошень на спеціально відведених для цього дошках і реклама в соціальних мережах. Проблемою в першому випадку є відчутні фінансові затрати на дизайн плаката та витрата великої кількості часу на розповсюдження цих постерів, а у другому – спрямованість тільки на невелику частину своєї аудиторії, яка слідкує за виконавцем саме в цих соціальних мережах. Але найбільшими їх спільними недоліками є вартість друку квитка і складність його доставлення до покупця.

Наприклад, уявімо ситуацію, де музикант Максим домовився з власником місцевого кафе про оренду зали для проведення свого виступу й розмістив оголошення на своїй сторінці в соціальній мережі про цей захід. Там йому написала повідомлення фанатка Діана з бажанням придбати декілька квитків. Тоді Максим повинен звернутись до друкарні, щоб йому виготовили необхідну кількість квитків, які він у вільний час повинен поїхати й віддати покупцю. Якщо таких охочих десятки або сотні, це вже складає чималі проблеми для однієї людини. Не треба забувати про фінансові витрати на дизайн та друк квитків, на дорогу для їх доставлення покупцю, на необхідність купівлі спеціального пристрою зчитування інформації з квитка для перевірки його дійсності та виявлення підробки.

Для розв’язання попередньо описаних недоліків оптимальним рішенням є створення програмного забезпечення, яке буде покривати повний перелік дій, необхідних для організації концерту: для артистів - створення оголошення з описом події, перевірка квитків при допуску відвідувача на вхід, а для прихильників - можливість переглянути цікаві події поблизу та придбати квитки через створену систему. Метою цієї дипломної роботи є створення саме такого програмного продукту.

Такий застосунок значно полегшить виконавцям організацію концертів, збереже їм купу часу та звільнить від чималих фінансових витрат, а відвідувачам таких подій допоможе швидко знайти цікаві заходи поблизу та зручно придбати квитки.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Концерт, за визначенням, - прилюдне виконання музичних творів, балетних, естрадних і т. ін. номерів за певною програмою. Це захід, який орієнтований на виступ однієї групи людей перед певною іншою. З таких міркувань випливають декілька доволі важливих вимог до програмного забезпечення для автоматизації таких процесів: оголошення дати та факту наявності деякого концерту, розповсюдження інформації про це серед якомога більшої кількості людей, створення унікального квитка для кожного користувача, який не можна використати для іншого концерту і більше, ніж 1 раз, та перевірку квитка на дійсність при вході на концерт.

Предметною областю цієї дипломної роботи є саме не популярні музичні колективи, які не мають менеджерів та будь-якого іншого персоналу, окрім безпосередньо музикантів, тобто займаються організаційною діяльністю концертів особисто.

1.2 Змістовний опис і аналіз предметної області

Основні необхідні бізнес-процеси організації та проведення концертів, можна виділити такі:

– створення події концерту та розміщення його оголошення на публічних прощадках або в соціальних мережах;

– створення дизайну квитка, друк кожного окремого екземпляру;

– особистий продаж квитків усім бажаючим їх придбати;

– перевірка дійсності квитка при вході клієнта на подію.

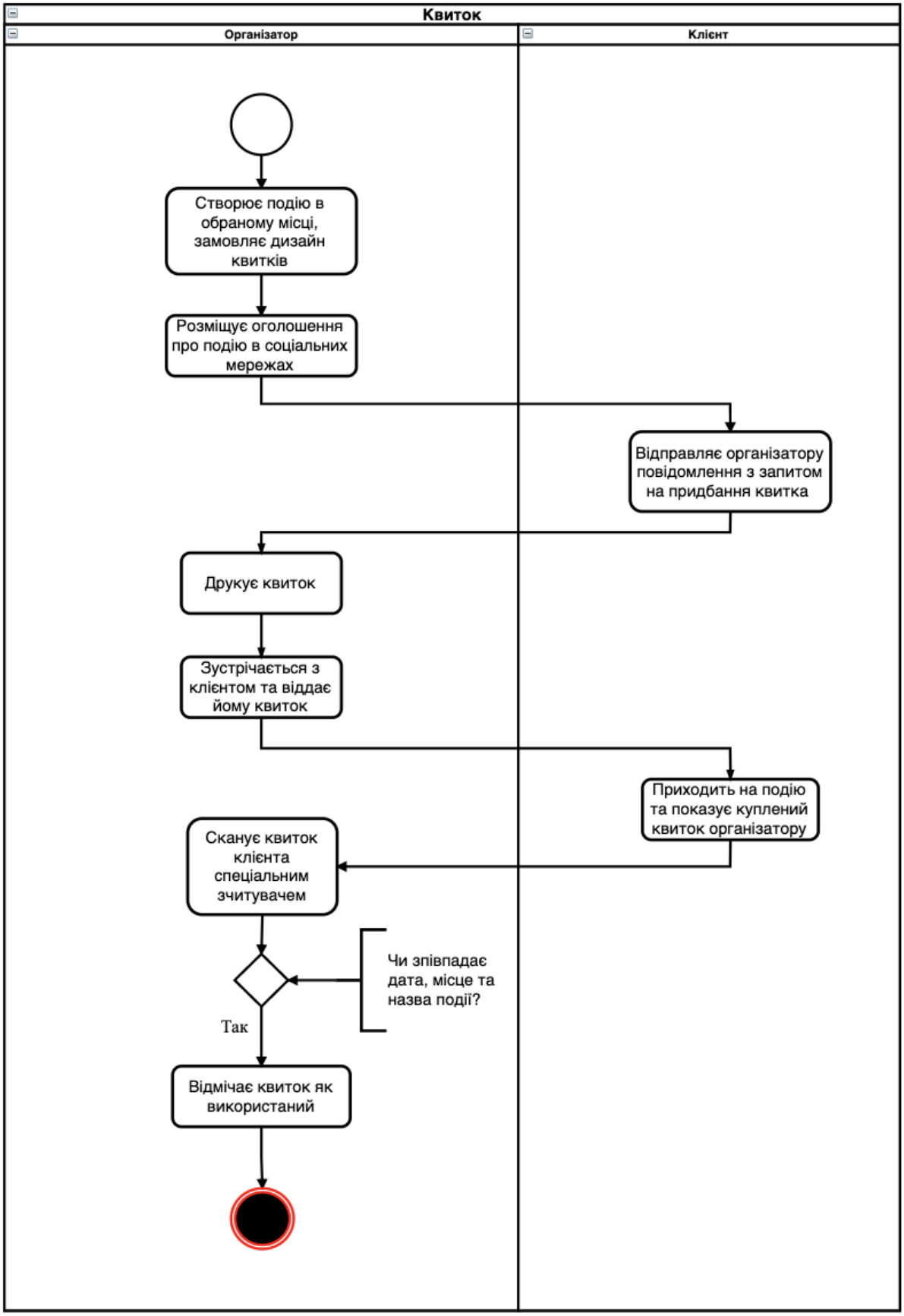


Рисунок 1.1 - Діаграма бізнес-процесів організації події концерту as-is

1.3 Аналіз успішних ІТ-проєктів

Для аналізу існуючих аналогів використовувався список необхідних критеріїв, що сформовано базуючись на вищевказаному списку необхідних можливостей системи:

– можливість для всіх бажаючих створити оголошення концерту;

– можливість пошуку та фільтрації оголошень;

– можливість користувачу створити власний профіль;

– можливість автоматичного і безпечного створення квитка при покупці;

– можливість онлайн оплати квитка;

– можливість перевірки дійсності квитка;

– можливість переглянути концерти поблизу користувача.

Concert.ua

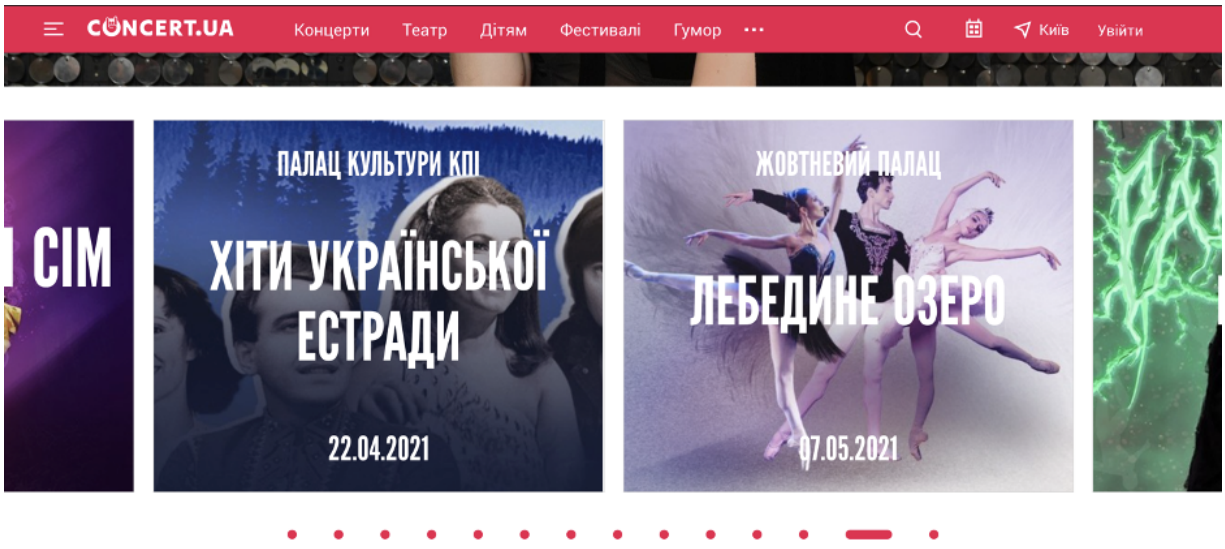


Рисунок 1.2 - Інтерфейс Concert.ua

Concert.ua – це одна з найпопулярніших в Україні платформ для пошуку квитків на різні заходи, в тому числі концерти. Вона налічує сотні різноманітних заходів та тисячі клієнтів.

Переваги:

– можливість пошуку та фільтрації оголошень;

– можливість користувачу створити власний профіль;

– можливість автоматичного і безпечного створення квитка при покупці;

– можливість онлайн оплати квитка.

Недоліки:

– можливість для всіх бажаючих створити оголошення концерту;

– можливість перевірки дійсності квитка;

– можливість переглянути концерти поблизу користувача.

Karabas

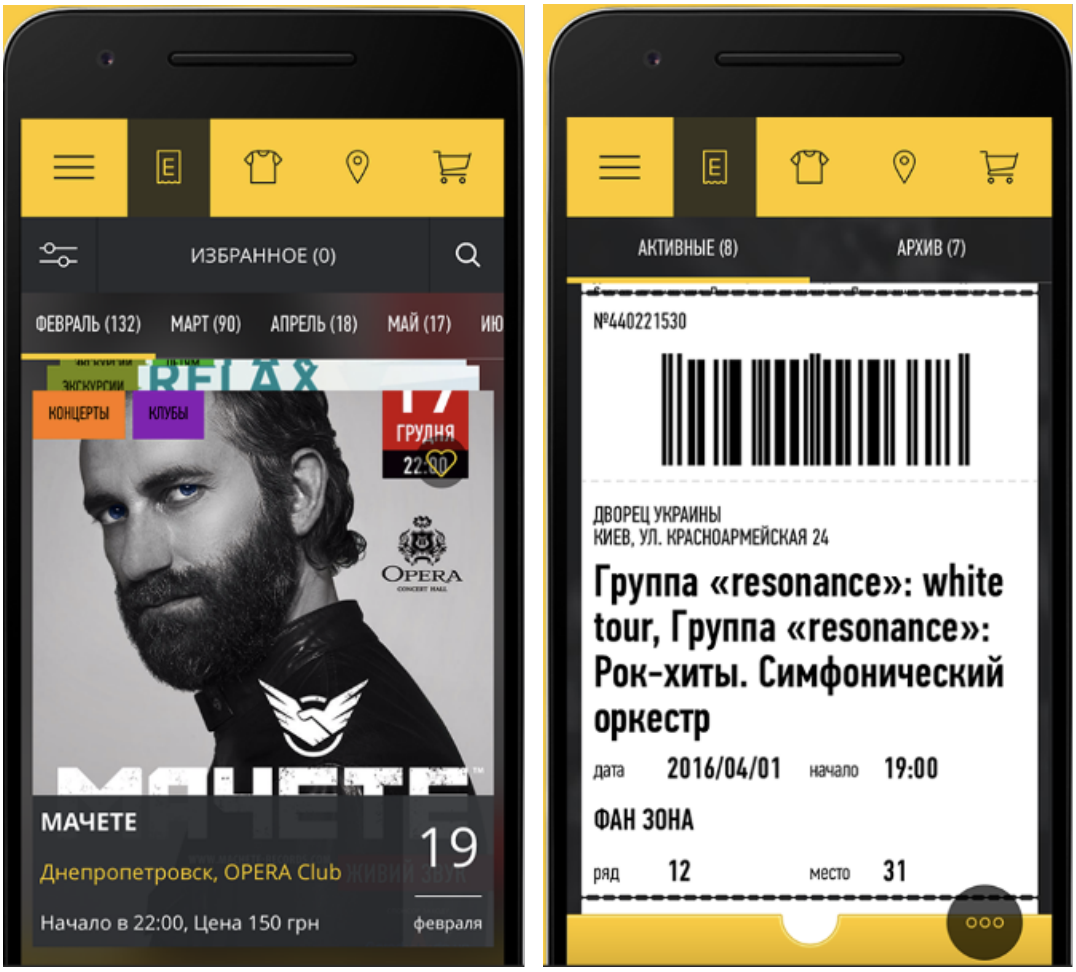


Рисунок 1.3 - Інтерфейс Karabas

Karabas – система продажу квитків на різні заходи: концерти, фестивалі, бізнес конференції та семінари, театральні постановки, дитячі та спортивні заходи. Платформа працює вже з 2010 року та заробила собі за цей час доволі гарну репутацію.

Переваги:

– можливість пошуку та фільтрації оголошень;

– можливість користувачу створити власний профіль;

– можливість автоматичного і безпечного створення квитка при покупці;

– можливість онлайн оплати квитка.

Недоліки:

– можливість для всіх бажаючих створити оголошення концерту;

– можливість перевірки дійсності квитка;

– можливість переглянути концерти поблизу користувача.

Internet-bilet

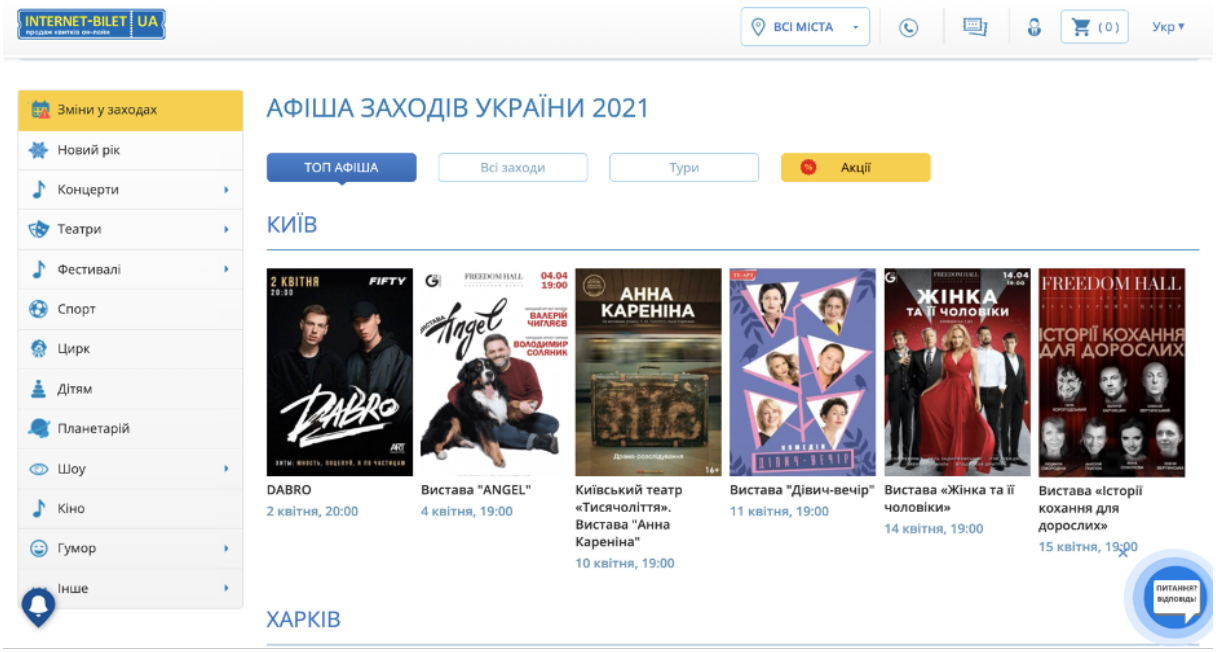


Рисунок 1.4 - Інтерфейс Internet-bilet

Internet-bilet – це онлайн сервіс для пошуку та придбання квитків на різні заходи, що існує вже більше 7 років. Також наявність окремого сайту для більшості міст України доволі вигідно виділяє цю платформу на фоні конкурентів.

Переваги:

– можливість пошуку та фільтрації оголошень;

– можливість користувачу створити власний профіль;

– можливість автоматичного і безпечного створення квитка при покупці;

– можливість онлайн оплати квитка.

Недоліки:

– можливість для всіх бажаючих створити оголошення концерту;

– можливість перевірки дійсності квитка;

– можливість переглянути концерти поблизу користувача.

1.4 Аналіз вимог до програмного забезпечення

Можливості взаємодії організатора з системою зображені на діаграмі варіантів використання програмного забезпечення. Він може виконати вхід у систему (UC-O1), що включає в себе реєстрацію та авторизацію для ідентифікації в системі. Основною функцією яку може здійснити організатор є створення події (UC-O2). Також, використовуючи систему користувач-організатор має можливість переглянути список створених подій (UC-O3) та може відредагувати кількість доступних квитків для певної події (UC-O4). На додачу, організатор може виконати перевірку квитка на дійсність (UC-O5). Діаграма прецедентів для користувача «Організатор» представлена у документі КПІ.ІП-7112.045490.05.99.СС Графічні матеріали.

Всі визначені варіанти дій користувача “Організатор” та їх опис наведені у таблицях 1.1 – 1.5.

Таблиця 1.1 – Варіант використання UC-O1

Атрибут	Значення
Use Case ID	UC-O1
Use Case Name	Увійти у систему
Use Case Description	Цей варіант використання описує механізм входу організатора у систему
Primary Actors	Організатор
Secondary Actors	-
Preconditions	Організатор має електронну пошту
Postconditions	Організатор авторизований
Trigger	Організатор заходить на сторінку входу в систему
Main flow	– Організатор вводить свою поштову адресу в поле «Email» та пароль в поле «Password». – Система успішно авторизує організатора. – Варіант використання завершує свою роботу.
Alternative flows	– Організатор вводить свою поштову адресу в поле «Email» та пароль в поле «Password». – Організатор отримує лист, в якому є посилання з токеном підтвердження. – Організатор переходить за цим посиланням. – Система успішно реєструє організатора та підтверджує його поштову адресу. – Варіант використання завершує свою роботу.

Продовження таблиці 1.1

Exception flows	Введений токен підтвердження неправильний. – Система виводить на екран повідомлення, про те, що введені дані неправильні. – Керування переходить на крок 1 основного потоку.
Result	Організатор авторизований

Таблиця 1.2 – Варіант використання UC-O2

Атрибут	Значення
Use Case ID	UC-O2
Use Case Name	Створити подію
Use Case Description	Цей варіант використання описує механізм створення організатором події концерту
Primary Actors	Організатор
Secondary Actors	-
Preconditions	Організатор авторизований у системі
Postconditions	Організатором створена нова подія концерту
Trigger	Організатор натискає на кнопку «Створити нову подію»
Main flow	– Організатор заповнює форму з необхідними для створення події даними. – Організатор натискає на кнопку «Створити». – Варіант використання завершує свою роботу.

Продовження таблиці 1.2

Alternative flows	-
Exception flows	Організатор вводить некоректні дані в формі створення події. – Система виводить на екран повідомлення, про те, що введені дані неправильні. – Керування переходить на крок 1 основного потоку.
Result	Організатор створив подію концерту в системі

Таблиця 1.3 – Варіант використання UC-O3

Атрибут	Значення
Use Case ID	UC-O3
Use Case Name	Переглянути список створених подій
Use Case Description	Цей варіант використання описує механізм перегляду організатором списку створених ним подій
Primary Actors	Організатор
Secondary Actors	-
Preconditions	– Організатор авторизований у системі. – Цим організатором створена хоча б одна подія у системі.
Postconditions	Організатор переглянув список подій
Trigger	Організатор натискає на кнопку «Переглянути список створених подій»

Продовження таблиці 1.3

Main flow	– Система виводить на екран список всіх створених цим організатором подій. – Варіант використання завершує свою роботу.
Alternative flows	– Система виводить на екран список всіх створених цим організатором подій. – Організатор натискає на одну із подій в списку. – Система виводить сторінку зі статистикою переглядів цієї події у системі. – Варіант використання завершує свою роботу.
Exception flows	-
Result	Організатор отримав інформацію про створені ним події

Таблиця 1.4 – Варіант використання UC-O4

Атрибут	Значення
Use Case ID	UC-O4
Use Case Name	Змінити кількість доступних квитків
Use Case Description	Цей варіант використання описує механізм редагування- організатором кількості доступних квитків для певної події
Primary Actors	Організатор
Secondary Actors	-

Продовження таблиці 1.4

Preconditions	– Організатор авторизований у системі. – Цим організатором створена хоча б одна подія у системі.
Postconditions	Організатор змінив кількість доступних квитків для події
Trigger	Організатор натискає на кнопку «Редагувати квитки» біля однієї з подій в списку
Main flow	– Організатор заповнює форму з новою кількістю квитків. – Організатор натискає на кнопку «Зберегти зміни». – Варіант використання завершує свою роботу.
Alternative flows	-
Exception flows	Організатор при редагуванні квитків вводить кількість квитків меншу, ніж вже придбано в системі. – Система виводить на екран повідомлення, про те, що введені дані неправильні. – Керування переходить на крок 1 основного потоку.
Result	Організатор отримав інформацію про створені ним події

Таблиця 1.5 – Варіант використання UC-O5

Атрибут	Значення
Use Case ID	UC-O5
Use Case Name	Перевірити дійсність квитка
Use Case Description	Цей варіант використання описує механізм перевірки організатором дійсності квитка
Primary Actors	Організатор
Secondary Actors	-
Preconditions	– Організатор авторизований у системі. – Цим організатором створена подія у системі. – Клієнтом придбано квиток на цю подію.
Postconditions	Квиток відмічений в системі як використаний
Trigger	Організатор натискає на кнопку «Перевірити дійсність квитка»
Main flow	– Організатор заходить на сторінку зі списком подій. – Організатор натискає на кнопку «Перевірити дійсність квитка» біля події у списку. – Організатор сканує QR-код на квитку клієнта. – Варіант використання завершує свою роботу.
Alternative flows	-

Продовження таблиці 1.5

Exception flows	Квиток не є дійсним. – Система виводить на екран повідомлення, про те, що введені дані неправильні. – Керування переходить на крок 3 основного потоку.
Result	Квиток відмічений як використаний

Можливості взаємодії клієнта з системою зображені на діаграмі варіантів використання програмного забезпечення. Він може виконати вхід у систему (UC-C1), що включає в себе реєстрацію та авторизацію для ідентифікації в системі. Також до можливостей клієнта входять: перегляд списку подій (UC-C2), перегляд подій поблизу себе на мапі (UC-C3), придбання квитка на подію (UC-C4), а також можливість завантаження квитка на свій пристрій (UC-C5). Діаграма прецедентів для користувача «Клієнт» представлена у документі КПІ.ІП-7112.045490.05.99.СС Графічні матеріали.

Всі визначені варіанти дій користувача “Клієнт” та їх опис наведені у таблицях 1.6 – 1.10.

Таблиця 1.6 – Варіант використання UC-C1

Атрибут	Значення
Use Case ID	UC-C1
Use Case Name	Увійти у систему
Use Case Description	Цей варіант використання описує механізм входу клієнта у систему
Primary Actors	Клієнт

Продовження таблиці 1.6

Secondary Actors	-
Preconditions	Клієнт має електронну пошту
Postconditions	Клієнт авторизований
Trigger	Клієнт заходить на сторінку входу в систему
Main flow	– Клієнт вводить свою поштову адресу в поле «Email» та пароль в поле «Password». – Система успішно авторизує клієнта. – Варіант використання завершує свою роботу.
Alternative flows	– Клієнт вводить свою поштову адресу в поле «Email» та пароль в поле «Password». – Клієнт отримує лист, в якому є посилання з токеном підтвердження. – Клієнт переходить за цим посиланням. – Система успішно реєструє клієнта та підтверджує його поштову адресу. – Варіант використання завершує свою роботу.
Exception flows	Введений токен підтвердження неправильний. – Система виводить на екран повідомлення, про те, що введені дані неправильні. – Керування переходить на крок 1 основного потоку.
Result	Клієнт авторизований

Таблиця 1.7 – Варіант використання UC-C2

Атрибут	Значення
Use Case ID	UC-C2
Use Case Name	Переглянути список подій
Use Case Description	Цей варіант використання описує механізм перегляду клієнтом актуальних подій у системі
Primary Actors	Клієнт
Secondary Actors	-
Preconditions	– Клієнт авторизований у системі. – Створена хоча б одна подія у системі. – До початку створеної у системі події залишилось більше 1 години.
Postconditions	Клієнт отримав список подій у системі
Trigger	Клієнт заходить на сторінку зі списком подій
Main flow	– Клієнт отримує список актуальних подій з наявними квитками. – Варіант використання завершує свою роботу.
Alternative flows	-
Exception flows	-
Result	Клієнт переглянув список актуальних подій у системі

Таблиця 1.8 – Варіант використання UC-C3

Атрибут	Значення
Use Case ID	UC-C3
Use Case Name	Переглянути події поблизу себе на мапі
Use Case Description	Цей варіант використання описує механізм перегляду клієнтом на мапі актуальних подій у системі поблизу себе
Primary Actors	Клієнт
Secondary Actors	-
Preconditions	– Клієнт авторизований у системі. – Створена хоча б одна подія у системі з заданими координатами.
Postconditions	Клієнт переглянув список подій поблизу себе на мапі
Trigger	Клієнт заходить на сторінку з мапою
Main flow	– Клієнт переходить на сторінку з мапою. – Клієнт отримує список актуальних подій з координатами місць їх проведення. – Клієнт бачить своє місцезнаходження на мапі та відмітки подій поблизу себе. – Варіант використання завершує свою роботу.
Alternative flows	-
Exception flows	-
Result	Клієнт переглянути події поблизу себе на мапі

Таблиця 1.9 – Варіант використання UC-C4

Атрибут	Значення
Use Case ID	UC-C4
Use Case Name	Придбати квиток на подію
Use Case Description	Цей варіант використання описує механізм придбання клієнтом квитка на подію
Primary Actors	Клієнт
Secondary Actors	-
Preconditions	– Клієнт авторизований у системі. – Створена хоча б одна подія у системі. – Хоча б одна подія має доступні квитки в системі.
Postconditions	Клієнт придбав квиток на подію
Trigger	Клієнт натискає на кнопку “Buy Ticket” на сторінці з деталями події
Main flow	– Клієнт переходить на сторінку з деталями обраної події. – Клієнт натискає на кнопку “Buy Ticket” на сторінці з деталями події. – Клієнт отримує квиток на цю подію. – Варіант використання завершує свою роботу.
Alternative flows	-
Exception flows	-
Result	Клієнт придбав квиток на подію

Таблиця 1.10 – Варіант використання UC-C5

Атрибут	Значення
Use Case ID	UC-C5
Use Case Name	Повернути придбаний квиток на подію
Use Case Description	Цей варіант використання описує механізм повернення придбаного користувачем квитка на подію
Primary Actors	Клієнт
Secondary Actors	-
Preconditions	– Клієнт авторизований у системі. – Клієнт має хоча б один придбаний квиток на подію. – До початку події залишається не менше 24 годин.
Postconditions	Клієнт повернув кошти за придбаний квиток на подію
Trigger	– Клієнт натискає на кнопку опцій квитка в списку придбаних ним квитків. – Клієнт натискає на кнопку “Return Ticket”.
Main flow	– Клієнт переходить на сторінку зі списком придбаних ним квитків. – Клієнт натискає на кнопку опцій одного з квитків у списку. – Клієнт натискає на кнопку “Return Ticket”. – Клієнт повертає квиток у систему. – Варіант використання завершує свою роботу.

Продовження таблиці 1.10

Alternative flows	-
Exception flows	Клієнт намагається повернути квиток н подію занадто пізно. – Клієнт переходить на сторінку зі списком придбаних ним квитків. – Клієнт натискає на кнопку опцій одного з квитків у списку. – Клієнт натискає на кнопку “Return Ticket”. – Квиток не проходить перевірку на умови повернення квитка. – Квиток залишається придбаний клієнтом. – Варіант використання завершує свою роботу.
Result	Клієнт повернув квиток у систему

Таблиця 1.11 – Варіант використання UC-C6

Атрибут	Значення
Use Case ID	UC-C6
Use Case Name	Завантажити квиток у локальне сховище даних
Use Case Description	Цей варіант використання описує механізм завантаження клієнтом файла з квитком на свій смартфон
Primary Actors	Клієнт

Продовження таблиці 1.11

Secondary Actors	-
Preconditions	– Клієнт авторизований у системі. – Клієнт має хоча б один придбаний квиток на подію.
Postconditions	Клієнт завантажив квиток на обрану подію на смартфон
Trigger	– Клієнт натискає на кнопку опцій квитка в списку придбаних ним квитків. – Клієнт натискає на кнопку “Download Ticket”.
Main flow	– Клієнт переходить на сторінку зі списком придбаних ним квитків. – Клієнт натискає один із квитків. – Клієнт завантажує файл з обраним квитком. – Варіант використання завершує свою роботу.
Alternative flows	-
Exception flows	-
Result	Клієнт завантажив файл з квитком на свій пристрій

1.4.1 Розроблення функціональних вимог

Відповідно до описаних вище варіантів використання розроблюваної системи можна сформулювати функціональні вимоги, наведені нижче.

Для організатора:

– реєстрація за адресою електронної пошти;

– авторизація;

– створення події;

– перегляд списку створених подій;
– можливість зміни кількості доступних квитків для події;
– можливість перевірки дійсності квитка.

Для клієнта:

– реєстрація за адресою електронної пошти;
– авторизація;
– перегляд повного списку подій;
– наявність пошуку та фільтрів для списку подій;
– перегляд списку подій поблизу себе на мапі;
– можливість придбати квиток;
– можливість завантажити квиток.

Відповідність функціональних вимог до визначених варіантів використання системи представлене через матрицю трасування, що представлена на рисунку (Рисунок 1.5).

	UC-O1	UC-O2	UC-O3	UC-O4	UC-O5	UC-C1	UC-C2	UC-C3	UC-C4	UC-C5
R 1.1.1	■									
R 1.1.2	■									
R 1.1.3		■								
R 1.1.4			■							
R 1.1.5				■						
R 1.1.6					■					
R 1.2.1						■				
R 1.2.2						■				
R 1.2.3							■			
R 1.2.4							■			
R 1.2.5								■		
R 1.2.6									■	
R 1.2.7										■

Рисунок 1.5 – Матриця трасування

1.4.2 Розроблення нефункціональних вимог

До програмного забезпечення були висунуті наступні нефункціональні вимоги:

– мова додатку - англійська;

– наявність доступу до Інтернет;

– для мобільного застосунку Android версії 6.0 і вище;

– для розгортання backend-сервісу одна з операційних систем Windows (7, 8 та 10) або Linux (Ubuntu 20.04, Fedora 32);

– для бази даних розгорнутий PostgreSQL сервер версії 9 і вище;

– паролі користувачів у БД повинні зберігатися в захешованому вигляді;

– чутливі особисті дані користувача (такі як поштова скринька або ім’я) у БД повинні зберігатися в зашифрованому вигляді;

– використана платіжна система має сертифікацію PCI DSS, що підтверджує повну відповідність системи обробки даних платіжних карт банку стандартам безпеки міжнародних платіжних систем Mastercard і Visa [6];

– взаємодія між backend-сервісом та мобільним застосунком повинна відбуватися за допомогою HTTP запитів.

1.4.3 Постановка комплексу завдань

Метою роботи є створення програмного забезпечення, яке буде покривати повний перелік дій, необхідних для організації концерту, економити час та фінансові ресурси користувачів, надавати клієнтам актуальну інформацію про доступні поблизу активності та значно спрощувати процедуру організації події.

Для досягнення поставленої мети були сформульовані такі задачі:

– реєстрація користувачів;

– авторизація користувачів;

– пошук та фільтрація подій за різними критеріями;

– доступ до створення події усім зареєстрованим користувачам;

– автоматична генерація електронної версії квитка при оплаті проходу на подію;

– завантаження квитка на пристрій для доступу до нього за відсутності інтернет-з’єднання;

– перегляд подій поблизу користувача на мапі.

1.5 Висновки до розділу

У цьому розділі були сформовані основні сценарії використання системи для двох ролей: “Організатор” та “Клієнт”. З описаних сценаріїв були сформовані основні функціональні та нефункціональні вимоги до програмного забезпечення, поставлені задачі для розробки.

Також завданням цього розділу був аналіз програмних продуктів конкурентів та існуючих рішень на ринку. Виходячи з отриманих в результаті даних, видно, що систем зі схожим функціоналом існує доволі небагато як серед веб-застосунків, так і серед мобільних додатків та різниця між ними досить не суттєва. Загалом, вони всі мають однакові недоліки та не покривають всі функціональні вимоги. Тому система, розроблена в ході цієї дипломної роботи, вигідно відрізняється від конкурентів та буде затребувана серед вже існуючих аналогів на ринку.

Отже, аналіз предметної області та існуючих рішень показав актуальність розробки системи керування та підтримки концертної діяльності музичних колективів.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для подальшої розробки вже безпосередньо програмного забезпечення коректно необхідно його змоделювати та проаналізувати. На цьому етапі треба визначити бізнес-процеси, які будуть задовольняти визначені раніше вимоги. Для відображення бізнес-процесів було використано діаграми BPMN. Модель та позначення бізнес-процесів (BPMN), тобто «модель та рейтинг бізнес-процесів», є методом моделювання бізнес-процесів для опису ланцюгів доданої вартості та ділової діяльності організації у формі графічного зображення [2].

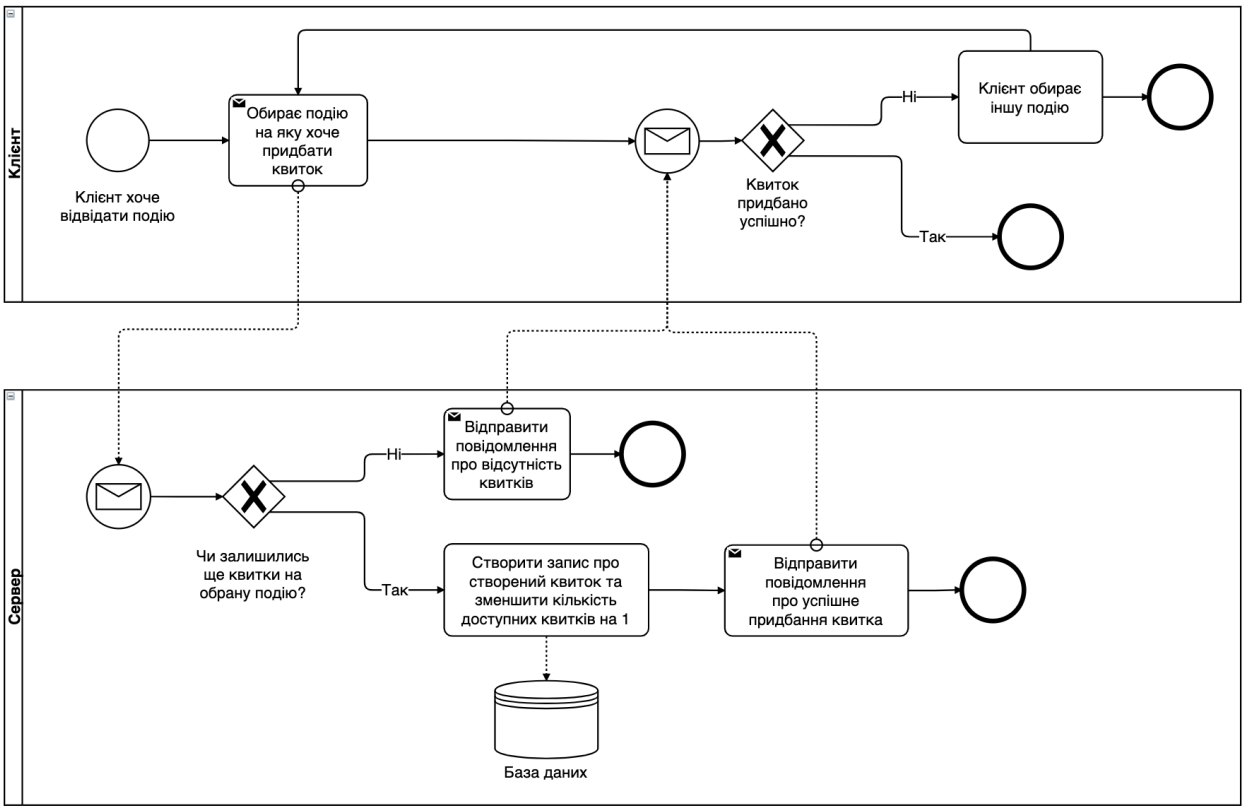


Рисунок 2.1 – BPMN-діаграма процесу купівлі квитка

Опис процесу купівлі квитка:

– клієнт обирає подію, на яку він бажає придбати квиток;

– клієнт надсилає запит про придбання обраного квитка на сервер;

– сервер проводить перевірку на наявність квитків на цю подію;

– якщо квитки ще залишились, сервер створює в базі даних запис про купівлю квитка та зменшує кількість доступних квитків на цю подію;

– у випадку якщо квитків не залишилось, клієнту повертається відповідне повідомлення після чого він може обрати квиток на іншу подію;

– у разі успішної перевірки та купівлі квитка, клієнт може переглянути придбаний ним квиток.

BPMN-діаграма процесу перевірки дійсності квитка представлена у документі КПІ.ІП-7112.045490.05.99.СС Графічні матеріали.

Опис процесу перевірки дійсності квитка:

– клієнт показує організатору фотографію з QR-кодом квитка;

– організатор робить фотографію QR-коду квитка клієнта;

– організатор відправляє фотографію з QR-кодом на сервер;

– сервер зчитує дані з QR-коду (такі як дата та ідентифікаційний номер події та ідентифікаційний номер клієнта);

– сервер виконує перевірку квитка на дійсність;

– у випадку невідповідності організатора події, клієнта або самої події, організатору відобразиться помилка зчитування квитка;

– якщо квиток є дійсним, організатору відобразиться відповідне повідомлення організатор пропускає клієнта на подію.

2.2 Архітектура програмного забезпечення

2.2.1 Опис використаних рішень

Для розробки було обрано середовища Android Studio IDE, VisualStudio Code IDE та DataGrip IDE, що полегшило розробку модулю. Так як для написання серверної частини була обрана мова JavaScript, середовище VisualStudio Code IDE проводило аналіз коду та автоматичне виправлення стилістичних помилок під час розробки, що пришвидшило розробку та

покращило автодоповнення. Android Studio IDE дозволяє зручно та гнучко налаштувати емулятор Android-пристрою для подальшого запуску клієнтської частини мобільного додатку на ньому. Також DataGrip IDE - найкращий вибір програмного забезпечення для налагодження та запуску SQL-коду та візуалізації структури бази даних.

Для реалізації серверної частини додатку була вибрана програмна платформа Node.js, заснована на рушію V8, яка дозволяє писати на мові JavaScript високонавантажені мережеві застосунки. Для написання сервера розглядались декілька популярних фреймворків – Express, Koa.js та Nest.js. В кінцевому результаті був обраний Express, так як ця бібліотека надає досить зручний функціонал та можливість розробляти високонавантажені масштабовані застосунки без накладних витрат, а також займає найменше пам’яті, що значно зменшує вимоги до апаратного забезпечення, на якому застосунок працюватиме та робить цей варіант найкращим в швидкості роботи.

Express є найпопулярнішим веб-середовищем Node.js і є базовим фреймворком для ряду інших популярних веб-платформ Node.js. Він забезпечує механізми для таких дій:

– написання обробників запитів для різних методів HTTP-запитів за різними шляхами URL (маршрутами);

– інтеграція з рушіями візуалізації, щоб генерувати відповіді, вставляючи дані в шаблони;

– встановлення загальних налаштувань веб-додатків, такі як порт для підключення та розташування шаблонів, які використовуються для відправлення відповіді;

– створення додаткової обробки запитів - "middleware" в будь-який етап обробки запиту [4].

Для розробки мобільного застосунку був обраний фреймворк Flutter, що базується на мові програмування Dart. Flutter - набір засобів розробки і

фреймворк з відкритим вихідним кодом для створення кросс-платформенних мобільних застосунків для Android та iOS, розроблений і створюваний корпорацією Google. Цікава ця платформа своєю простотою і швидкістю роботи на рівні з нативними додатками. Висока продуктивність програми і швидкість розробки досягається за рахунок далі зазначених технік:

– на відміну від багатьох відомих на сьогоднішній день кросс-платформенних мобільних засобів розробки, Flutter не використовує JavaScript ні в якому вигляді. В якості мови програмування для Flutter вибрали Dart, який компілюється в бінарний код, за рахунок чого досягається швидкість виконання операцій порівнянна з Objective-C, Swift, Java, або Kotlin;

– Flutter зовсім не використовує нативні компоненти, так що не доводиться писати ніяких проміжних шарів для роботи з ними. Замість цього, подібно до ігрових рушіїв, він промальовує весь інтерфейс самотужки.

– для побудови UI Flutter використовує декларативний підхід, як у веб-бібліотеці ReactJS, на основі віджетів. Для ще більшого приросту в швидкості роботи інтерфейсу віджети перемальовуються лише коли в них щось змінилося;

– на додаток до всього, в фреймворк вбудований зручна утиліта для розробки - Hot-reload, яка все ще зустрічається в нативних платформах досить рідко [9].

У якості сховища даних була використана реляційна база даних PostgreSQL. Для спілкування з цією базою даних був використаний конструктор SQL-запитів knex.js, який надає зручний функціонал для базової роботи із драйвером бази даних і також забезпечує міграції [7].

2.2.2 Опис сховища даних

Сховищем даних в цьому програмному продукті виступає база даних. Під час проєктування модулю була розроблена її схема (Рисунок 2.2).

Рисунок 2.2 – Схема бази даних

Користувачі системи були представлені в таблиці users. Вона містить повну інформацію про користувача та системні дані для його ідентифікації.

Таблиця 2.1 – Опис таблиці users

Назва поля	Тип	Первинний ключ	Обов’язкове поле	Опис
id	integer	Так	Так	Ідентифікаційний номер користувача
email	varchar	Ні	Так	Поштова адреса користувача
password	varchar	Ні	Так	Пароль користувача для доступу

Продовження таблиці 2.1

full_name	varchar	Ні	Ні	Повне ім’я користувача
photo_url	varchar	Ні	Ні	Посилання на фотографію користувача
date_of_birth	timestamp	Ні	Ні	Дата народження користувача
token	varchar	Ні	Ні	Токен доступу до функціоналу, що доступний лише авторизованим користувачам
email_verification_token	varchar	Ні	Ні	Токен для підтвердження користувачем доступу до вказаної ним поштової адреси
reset_password_token	varchar	Ні	Ні	Токен для оновлення паролю користувача
created_at	timestamp	Ні	Так	Дата створення запису про користувача
deleted_at	timestamp	Ні	Ні	Дата видалення запису про користувача

Для зберігання інформації про наявні події використовується таблиця events. З неї можна дізнатись повну інформацію про дату і місце проведення події, хто її створив та скільки квитків в наявності.

Таблиця 2.2 – Опис таблиці events

Назва поля	Тип	Первинний ключ	Обов’язкове поле	Опис
id	integer	Так	Так	Ідентифікаційний номер події
title	varchar	Ні	Так	Назва події
location	varchar	Ні	Ні	Місце проведення події
price	integer	Ні	Так	Ціна квитка для події
event_runner	varchar	Ні	Так	Ім’я організатора події
start_date	timestamp	Ні	Так	Дата та час початку події
end_date	timestamp	Ні	Так	Дата та час кінця події
max_visitors	integer	Ні	Ні	Максимальна кількість відвідувачів події
description	varchar	Ні	Ні	Опис події
created_at	timestamp	Ні	Так	Дата створення запису про користувача
deleted_at	timestamp	Ні	Ні	Дата видалення запису про користувача

Продовження таблиці 2.2

longitude	double precision	Ні	Ні	Градус довготи місця проведення події
latitude	double precision	Ні	Ні	Градус широти місця проведення події

Дані про картинки для подій збережені в таблиці event_images. Кожна подія може мати декілька картинок, тому ця таблиця зв’язана з events як багато-до-одного.

Таблиця 2.3 – Опис таблиці event_images

Назва поля	Тип	Первинний ключ	Обов’язкове поле	Опис
id	integer	Так	Так	Ідентифікаційний номер запису про картинку для події
event_id	integer	Ні	Так	Ідентифікаційний номер події, зовнішній ключ
image_url	varchar	Ні	Так	Посилання на картинку

Інформація про придбані квитки знаходиться в таблиці tickets.

Таблиця 2.4 – Опис таблиці tickets.

Назва поля	Тип	Первинний ключ	Обов’язкове поле	Опис
id	integer	Так	Так	Ідентифікаційний номер запису про картинку для події

Продовження таблиці 2.4

event_id	integer	Ні	Так	Ідентифікаційний номер події, зовнішній ключ
user_id	integer	Ні	Так	Ідентифікаційний номер користувача, що придбав квиток, зовнішній ключ
created_at	timestamp	Ні	Так	Дата створення запису про користувача
deleted_at	timestamp	Ні	Ні	Дата видалення запису про користувача

2.2.3 Опис модулів клієнтської частини системи

Основні елементи в архітектурі застосунків, написаних на Flutter - модулі та компоненти. Модулі частіше за все створюються з урахуванням ієрархічного положення їх компонентів в UI застосунку та абстрактністю їх інтерфейсів. Всередині модулів знаходяться компоненти – утиліти, віджети, компоненти графічного інтерфейсу, сутності тощо, які тісно пов’язані між собою сферою та способом використання. Структурна схема компонентів створеного програмного забезпечення представлена на рисунку (Рисунок 2.3).

Рисунок 2.3 - Діаграма компонентів клієнтської частини застосунку

Наявні в системі модулі та їх опис представлені в таблиці (Таблиця 2.5).

Таблиця 2.5 – Опис модулів клієнтської частини системи

Назва модулю	Опис модулю
utils	Функціональний модуль, який тримає в собі компоненти для роботи з мережею, стилем UI, валідацією, набір допоміжних функцій, які використовуються в багатьох частинах системи

Продовження таблиці 2.5

entities	Функціональний модуль, який інкапсулює логіку роботи із базовими сутностями систем та моделями таблиць бази даних
screens	Функціональний модуль, який тримає в собі набір основних віджетів системи та всю специфічну для них логіку: від стану віджета та специфічних для нього функцій до ієрархії графічного дерева відображення
components	Функціональний модуль, який відповідає за набір абстрактних допоміжних віджетів та UI компонентів, що розташовуються та використовуються в різних частинах системи

Більш детальний опис компонентів клієнтської частини застосунку наведений у таблиці (Таблиця 2.6).

Таблиця 2.6 – Опис компонентів клієнтської частини системи

Назва компонента	Опис компонента
AuthScreen	Компонент для обробки логіки та відображення UI на сторінці авторизації користувача
EventDetailsScreen	Компонент для обробки логіки та відображення користувачу повної інформації про певну подію. Також зберігає логіку придбання квитка
EventListScreen	Компонент для зберігання логіки спілкування з сервером та відображення UI на сторінці з повним списком доступних подій

Продовження таблиці 2.6

FirstScreen	Компонент для відображення першого екрану користувачеві
LoginScreen	Компонент для обробки логіки та відображення UI на сторінці авторизації користувача
MapScreen	Компонент для взаємодії з зовнішнім Google Maps API та логікою його коректного відображення
MyTicketsScreen	Компонент для управління списком придбаних користувачем квитків
ValidateTicketScreen	Компонент, що використовується організатором подій для перевірки дійсності квитка користувача та взаємодії з апаратним забезпеченням пристрою, а саме з камерою та файловою системою
EventPreview	Компонент, який тримає в собі інкапсульовану логіку відображення даних про певну подію
Header	Компонент, який відповідає за відображення основної панелі управління застосунком, що знаходиться вгорі додатку
InputField	Компонент, що інкапсулює логіку відображення та обробки вводу даних користувачем, їх валідацію та подальше використання
SideBar	Компонент, який відповідає за основну маршрутизацію по сторінках застосунку

Продовження таблиці 2.6

Spinner	Компонент, що зберігає в собі графічний елемент інформування користувача про обробку певних даних або запит мережею

2.2.4 Опис модулів серверної частини системи

Серверні застосунки, що розроблені за допомогою мінімалістичного фреймворку Express, зазвичай поділяються за їх зоною відповідальності: обробка запитів від клієнта, перевірка наявності доступу запиту до ресурсу, перевірка дійсності запиту та надісланих в ньому даних, зв’язок з базою даних, конструювання запитів до бази даних, обробка помилок тощо. Базуючись на цьому принципі було виділено чотири основні модулі застосунку: routes, controllers, services та utils. Більш детальний опис та діаграма компонентів наведені нижче.

Рисунок 2.4 – Діаграма компонентів серверної частини застосунку

Таблиця 2.7 – Опис модулів серверної частини системи

Назва модуля	Опис модуля
routes	Функціональний модуль для обробки точок входу в систему, первинної перевірки дійсності запиту та наявності доступу до необхідного клієнту ресурсу
controllers	Функціональний модуль для обробки даних та файлів, що були передані в запиті, обробка помилок їх недійсності або невідповідності логіці запитаного ресурса, надсилання відповіді клієнту

Продовження таблиці 2.7

services	Функціональний модуль для обробки запитів до бази даних, конструювання SQL-запитів, підтримка з'єднання з базою даних
utils	Функціональний модуль для збереження та надання іншим модулям системи різних допоміжних функцій для обробки спілкування з зовнішніми сервісами та значень для збереження компонентів серверного застосунку в цілісному стані

Більш детальний опис компонентів серверної частини застосунку наведений у таблиці (Таблиця 2.8).

Таблиця 2.8 – Опис компонентів серверної частини системи

Назва компонента	Опис компонента
Event	Компонент, що обробляє та перенаправляє в необхідні компоненти запити, що стосуються подій в системі
Ticket	Компонент, що обробляє та перенаправляє в необхідні компоненти запити, що стосуються квитків в системі
User	Компонент, що обробляє та перенаправляє в необхідні компоненти запити, що стосуються користувачів в системі
BuyTicket	Компонент, який інкапсулює логіку, яка пов’язана з купівлею та оплатою квитка
DownloadTicket	Компонент, який відповідає за генерацію файлу квитка та надсилання його клієнту

Продовження таблиці 2.8

GetEventList	Компонент, що займається наданням клієнту даних про наявні в системі події
GetUserTickets	Компонент, який дозволяє авторизованим користувачам отримати список придбаних ним квитків
ReturnTicket	Компонент, який займається обробкою запитів щодо повернення квитків та коштів за них клієнту, а також перевіркою виконання умов можливості повернення квитка
ValidateTicket	Компонент, що відповідає за логіку, пов’язану з перевіркою дійсності квитка на подію, створену користувачем, що відправив відповідний запит
LoginUser	Компонент, який відповідає за авторизацію користувача в системі та надання йому доступу до недоступним іншим даним
LogoutUser	Компонент, який зберігає в собі функціонал виходу користувача з системи
RegisterUser	Компонент, який інкапсулює логіку створення нового користувача в системі
EventService	Компонент, що займається спілкуванням інших компонентів системи з сутністю події в базі даних та даних, які з нею пов’язані
TicketService	Компонент, який відповідає за збереження та отримання даних з таблиці квитків в базі даних

Продовження таблиці 2.8

UserService	Компонент, що дозволяє іншим модулям працювати з таблицею користувачів в сховищі даних
errors	Компонент, що інкапсулює можливі помилки в системі та надає доступ до них для інших компонентів
constants	Компонент, який зберігає дані, що необхідні для коректної роботи програмного забезпечення та дані, що використовуються неодноразово в різних компонентах чи модулях додатку
knex	Компонент, що відповідає за налаштування та коректне використання конструктора SQL-запитів та його зв'язок з базою даних
passport	Компонент, який займається перевіркою можливості доступу запиту користувача до певного ресурсу та управлінням наданням такого доступу
validation	Компонент, який відповідає за логіку перевірки коректності та надійності надісланих клієнтом в запиті даних

В результаті серверна частина надає RestAPI із всіма необхідними для функціонування точками входу (Таблиця 2.9).

Таблиця 2.9 – Опис точок входу системи

Шлях точки входу	HTTP метод	Призначення
/	GET	Отримати статус роботи сервера

Продовження таблиці 2.9

/login	POST	Авторизуватись у системі, отримати JWT токен
/logout	GET	Вийти з системи
/register	POST	Зареєструвати нового користувача в системі
/list	GET	Отримати список актуальний подій, наявних в системі
/buyTicket	POST	Придбати квиток на обрану подію
/downloadTicket	GET	Завантажити обраний квиток серед придбаних
/list	GET	Отримати список придбаних користувачем квитків
/validate	POST	Перевірити дійсність завантаженого квитка

2.3 Аналіз безпеки даних

Для забезпечення цілісності даних та захисту вразливих даних від передбачених системою змін та зчитування необхідно використовувати шифрування та захищені з’єднання. Тому мобільний застосунок та серверна частина обмінюються повідомленнями, використовуючи безпечне HTTPS з’єднання. Для авторизації і подальшого допуску користувача до захищених точок доступу сервера використовується стратегія Bearer Token, який

генерується при логіні у вигляді хешу від даних, необхідних для подальшої унікальної ідентифікації кожного користувача, він дійсний всього 4 години, після чого його необхідно поновлювати, що зменшує час для несанкціонованого втручання у разі його викрадення.

Для безпечного зберігання чутливих даних користувачів, таких як пароль, використовується KDF-функція Bcrypt, яка є одним з найкращих виборів для цього на даний момент.

2.4 Висновки до розділу

Даний розділ містить в собі основні етапи моделювання, конструювання та проєктування розробленої системи автоматизованої підтримки концертної діяльності музичних колективів. Для повного розуміння бізнес-процесів були створені BPMN-діаграми. На базі вимог та необхідного набору функціоналу були вибраний стек середовищ, технологій, платформ та інструментів розробки, які дають можливість створювати програмний продукт максимально ефективно та досягнути поставлених задач.

Також були описані модулі та компоненти програмного забезпечення, описана структура та представлена графічна схема бази даних із необхідними таблицями, їх використанням, наповненням тощо.

Не менш важливим пунктом у даному розділі був аналіз безпеки даних, який показав достатній рівень захищеності створеної системи. Були описані процеси авторизації користувача і процес збереження чутливих даних в безпечному стані. Були пояснені вибрані підходи та їх переваги.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості програмного забезпечення

Процес тестування програмного забезпечення є важливим та невід'ємним етапом життєвого циклу розробки будь-якого проєкту. Цей процес дозволяє не лише перевірити якість готового продукту, а й перевірити виконання поставлених вимог та функцій, які мають бути реалізовані. За допомогою тестування можна охопити різні елементи застосунку, опираючись на наявні часові та людські ресурси. Тестування надає змогу виявити критичні проблеми до здійснення розгортання та впровадження програмного забезпечення.

3.2 Опис процесу тестування

Для тестування створеного програмного забезпечення було обрано такі методи тестування:

– функціональне тестування [10], за допомогою якого можна перевірити правильність виконання поставлених задач та відповідність усім вимогам до програмного продукту;

– інтеграційне тестування [8], за допомогою якого перевіряється коректність роботи системи в цілому;

– тестування макету [5], що допомагає перевірити чи відповідає відображення розробленої клієнтської частини дизайнам.

Таким чином контроль якості проходять і модулі системи незалежно один від одного, і коректність їх взаємодії між собою, і UI/UX додатку.

Так як тестування макету досить складно автоматизувати, а кінцевий програмний продукт допускає деякі зміни в розташуванні або стилізації окремих компонентів клієнтської частини додатку, воно проводилось вручну перед створенням нової версії мобільного додатку. На відміну від такого

підходу, інтеграційне та функціональне тестування можна автоматизувати та запускати кожен раз при розгортанні нової тестової версії застосунку. В такому випадку, після закінчення процесу тестування, розробнику виводиться не тільки повідомлення про статус завершення процесу тестування, а й помилки, які були знайдені, і які дані та в якому місці в коді викликали цю помилку.

3.3 Опис контрольного прикладу

В процесі тестування була перевірена уся функціональність розробленого програмного забезпечення. Послідовно були перевірені усі варіанти використання, результати представлені у відповідних таблицях:

- створення користувача (Таблиця 3.1);
- створення події (Таблиця 3.2);
- завантаження квитка (Таблиця 3.3);
- перегляд повного списку доступних подій (Таблиця 3.4);
- перегляд списку квитків певного користувача (Таблиця 3.5);
- перевірка дійсності квитка (Таблиця 3.6);
- повернення придбаного квитка (Таблиця 3.7).

Таблиця 3.1 – Створення користувача

Мета тесту	Перевірка створення користувача
Початковий стан	Відкрито сторінку реєстрації, користувач з такою поштовою адресою не зареєстрований
Вхідні дані	Електронна пошта та пароль користувача
Схема проведення тесту	Увести реєстраційні дані користувача до відповідних полів форми, натиснути кнопку “Sign Up”

Продовження таблиці 3.1

Очікуваний результат	Користувач є зареєстрованим з наданими даними
Стан програмного продукту після проведення випробувань	Автоматично відкрита сторінка з повним списком актуальних подій, дані користувача збережено у базі даних, користувач автоматично авторизований у системі

Таблиця 3.2 – Створення події

Мета тесту	Перевірка створення нової події в системі
Початковий стан	Відкрито сторінку створення події
Вхідні дані	Назва події, дата та час початку та кінця події, місце її проведення, ціна квитка, максимальна кількість відвідувачів
Схема проведення тесту	Увести дані про подію до відповідних полів форми, натиснути кнопку “Create Event”
Очікуваний результат	Подія є створеною та відображається у клієнтів системи в списку
Стан програмного продукту після проведення випробувань	Автоматично відкрита сторінка з повним списком актуальних подій, дані про подію збережено у базі даних

Таблиця 3.3 – Завантаження квитка

Мета тесту	Перевірка завантаження квитка
Початковий стан	Відкрито сторінку зі списком придбаних користувачем квитків, користувач авторизований в системі, користувач має хоча б 1 придбаний квиток
Вхідні дані	-
Схема проведення тесту	Натиснути на обраний квиток
Очікуваний результат	Квиток завантажився в файлову систему смартфона. Змістом файлу є QR-код
Стан програмного продукту після проведення випробувань	Користувач залишився на цій же сторінці

Таблиця 3.4 – Перегляд повного списку доступних подій

Мета тесту	Перевірка перегляду повного списку доступних подій
Початковий стан	Користувач авторизований в системі
Вхідні дані	-
Схема проведення тесту	Користувач переходить на сторінку зі списком актуальних подій
Очікуваний результат	Користувач бачить список подій з часом початку пізніше поточного моменту та з наявними квитками

Продовження таблиці 3.4

Стан програмного продукту після проведення випробувань	Користувач залишився на цій же сторінці

Таблиця 3.5 – Перегляд списку квитків певного користувача

Мета тесту	Перевірка перегляду списку квитків певного користувача
Початковий стан	Користувач авторизований в системі
Вхідні дані	-
Схема проведення тесту	Користувач переходить на сторінку зі списком придбаних ним квитків
Очікуваний результат	Користувач бачить список придбаних ним квитків
Стан програмного продукту після проведення випробувань	Користувач залишився на цій же сторінці

Таблиця 3.6 – Перевірка дійсності квитка

Мета тесту	Пересвідчитися у можливості перевіряти дійсність квитка

Продовження таблиці 3.6

Початковий стан	Користувач А авторизований в системі та має створену подію, користувач В має придбаний квиток на цю подію
Вхідні дані	Ідентифікатор події, ідентифікатор користувача
Схема проведення тесту	Користувач А переходить на сторінку перевірки квитків, користувач В показує користувачу А QR-код квитка, після чого користувач А зчитує цей QR-код
Очікуваний результат	Користувач А бачить повідомлення, що квиток на цю подію для користувача В є дійсним, цей квиток відмічений як недоступний для подальшого використання
Стан програмного продукту після проведення випробувань	Користувач А залишився на цій же сторінці

Таблиця 3.7 - Повернення придбаного квитка

Мета тесту	Перевірити можливість повернути квиток
Початковий стан	Відкрито сторінку зі списком придбаних користувачем квитків, користувач авторизований в системі, користувач має хоча б 1 придбаний квиток
Вхідні дані	Ідентифікатор події, ідентифікатор користувача
Схема проведення тесту	Користувач натискає на обраний квиток і обирає опцію «Return Ticket»

Продовження таблиці 3.7

Очікуваний результат	Користувач бачить повідомлення, що квиток на цю подію було повернуто, цей квиток зникає зі списку доступних для цього користувача
Стан програмного продукту після проведення випробувань	Користувач залишився на цій же сторінці

3.4 Висновки до розділу

У цьому розділі був описаний процес тестування програмного забезпечення, проведено аналіз якості розроблюваного програмного продукту шляхом проведення функціонального, інтеграційного тестування та тестування макету.

В ході роботи були обрані загальні критерії тестування та описані варіанти тестування згідно з варіантами використання системи.

У результаті тестування було розроблено план тестування і виявлено, що даний модуль задовольняє всі вимоги, та готовий до розгортання.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розгортання серверної частини додатку відбувалося за допомогою CI/CD системи Bitbucket Pipelines - проста в налаштуванні інтегрована система CI/CD для Bitbucket Cloud автоматизує код на етапах від тестування до робочого розгортання [1]. В конфігурації повний процес розгортання розбивається на кроки і описується послідовність команд, які будуть виконуватись під час кожного кроку. Так в системі наявні такі кроки як збірка, тестування, розгортання в хмарному середовищі.

Також важливу роль в цьому грає утиліта контейнеризації Docker. Docker абстрагує програмне забезпечення від середовища, на якому воно виконується, тому даний крок не вимагає ніяких попередніх установок. Для розгортання в такому випадку необхідно скачати код модулю та запустити всі необхідні контейнери за допомогою команди: «docker-compose up -d» [3].

Так для розгортання системи потрібно:

– встановити на сервер середовище виконання Node.js (^14.0), базу даних PostgreSQL (^9.0);

– за допомогою docker завантажити docker image зі сховища образів контейнерів Docker Hub;

– створити базу даних;

– налаштувати підключення до бази даних;

– задати необхідні змінні оточення;

– запустити проєкт.

4.2 Робота з програмним забезпеченням

Після запуску модуль буде продовжувати працювати безперебійно в автономному режимі. Подальша робота описана у документі КПІ.ІП-7112.045490.02.34 Керівництво користувача.

4.3 Висновки до розділу

У цьому розділі були описані способи розгортання програмного продукту підтримки концертної діяльності музичних колективів.

ВИСНОВКИ

В рамках роботи над дипломним проєктом була створена система підтримки концертної діяльності музичних колективів.

Спочатку був проведений детальний аналіз предметної області та розглянуті основні сценарії використання розроблюваного програмного продукту. На базі цих даних були виведені відповідні вимоги та рішення поставленої задачі.

Також був проведений аналіз оптимальної архітектури розробленого програмного забезпечення та на базі цього був вибраний технологічний стек. Проведено аналіз безпеки даних та системи в цілому.

Для підвищення якості розроблюваного програмного продукту було описане і проведене детальне тестування, що дозволило знайти проблеми в роботі та виправити їх.

Останнім кроком було створення документації для полегшення роботи з програмним продуктом, розглянуто впровадження та розгортання системи для її подальшого використання.

ПЕРЕЛІК ПОСИЛАНЬ

1) Bitbucket Pipelines [Електронний ресурс] – Режим доступу до ресурсу: https://bitbucket.org/product/ru/features/pipelines.

2) Object Management Group. Business Process Model and Notation [Електронний ресурс] – Режим доступу до ресурсу: https://www.bpmn.org.

3) Docker Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com.

4) Express.js Docs [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com.

5) Layout testing [Електронний ресурс] – Режим доступу до ресурсу: https://chromium.googlesource.com/chromium/src.git/+/65.0.3283.0/docs/testing/layout_tests.md.

6) LiqPay [Електронний ресурс] – Режим доступу до ресурсу: https://www.liqpay.ua/uk.

7) PostgreSQL Docs [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs.

8) Інтеграційне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/integration-testing.html.

9) Про Flutter, кратко: Основы [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/430918.

10) Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/functional-testing.html.

ДОДАТОК А ЗВІТ ПРО ПЕРЕВІРКУ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008308296

Дата перевірки:
16.06.2021 06:27:53 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.06.2021 06:34:49 EEST

ID користувача:
77149

Назва документа: **Kuvichka_bachelor_ip71_3**

Кількість сторінок: **56** Кількість слів: **7968** Кількість символів: **63560** Розмір файлу: **214.55 KB** ID файлу: **1008376226**

15.4%
Схожість

Найбільша схожість: **3.33%** з джерелом з Бібліотеки (ID файлу: **1008366809**)

6.49% Джерела з Інтернету	231	Сторінка 58
14.2% Джерела з Бібліотеки	602	Сторінка 61

0% Цитат

Не знайдено жодних цитат

Вилучення списку бібліографічних посилань вимкнене

84.3%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

Немає вилучених Інтернет-джерел

84.3% Вилученого тексту з Бібліотеки	2	Сторінка 61

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ КОНЦЕРТНОЇ ДІЯЛЬНОСТІ МУЗИЧНИХ КОЛЕКТИВІВ

Керівництво користувача

КПІ.ІП-7112.045490.03.34

“ПОГОДЖЕНО”

Керівник проєкту:

А. Ю. Черненький

Нормоконтроль:

К.І. Ліщук

Виконавець:

М. Є. Кувічка

Київ – 2021 року

Точкою входу в систему для всіх користувачів є екран авторизації або реєстрації в системі.

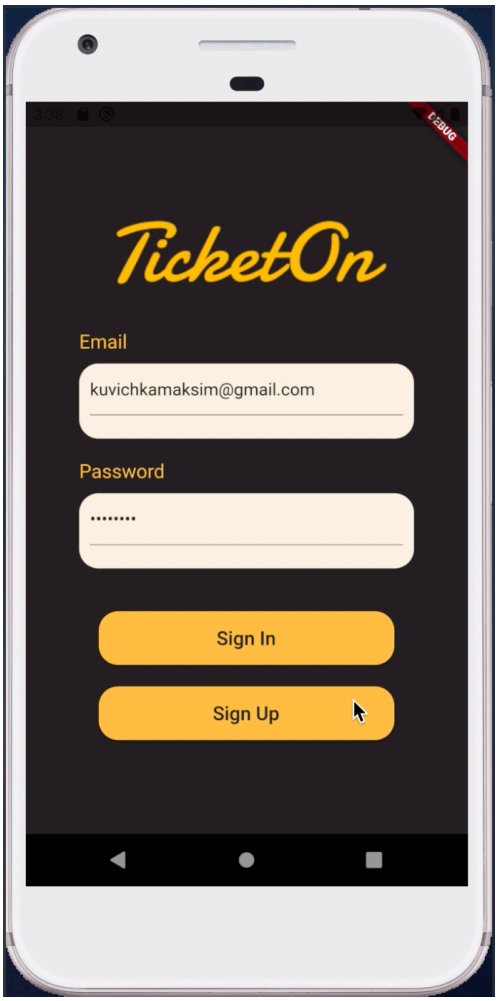


Рисунок 1 – Екран входу в систему

Треба заповнити поля email та password та натиснути на “Sign in” для авторизації або на “Sign up” для реєстрації.

Далі користувач потрапляє на головний екран додатку і бачить список всіх доступних подій в системі.

Рисунок 2 – Сторінка списку подій у системі

Для навігації між сторінками додатку необхідно відкрити бічну панель. Для цього необхідно натиснути на кнопку зліва вгорі екрану.

Ці кнопки ведуть на такі екрани:

– Create event – на екран створення події;
– My tickets – на екран зі списком придбаних подій;
– Validate Tickets – на екран перевірки дійсності квитка;
– Privacy Policy – на екран політики конфіденційності;
– Terms of Service – на екран умов користування системою;
– Settings – на екран налаштувань профілю;
– Log out – вийти з системи і перейти на екран авторизації.

Рисунок 3 – Бічна панель навігації

Далі можна створити подію. Для цього треба перейти на екран створення події, заповнити необхідні поля: назва, адреса, ім’я організатора, дата події, час початку та кінця події, максимальна кількість відвідувачів та натиснути на кнопку “Create event” для збереження. Після цього всі користувачі зможуть знайти цю подію в списку.

Рисунок 4 – Екран створення події

Інший користувач може придбати квиток на щойно створену подію. Для цього йому треба знайти цю подію в списку подій і натиснути для відображення розгорнутої інформації про подію. Після цього треба натиснути на кнопку “Buy Ticket”, перейти на сторінку оплати, заповнити дані картки і здійснити оплату. Відтепер квиток відображається в списку подій цього користувача.

Рисунок 5 – Екран з детальною інформацією про подію

Рисунок 6 – Екран оплати

Щоб переглянути список придбаних квитків необхідно перейти на екран придбаних квитків.

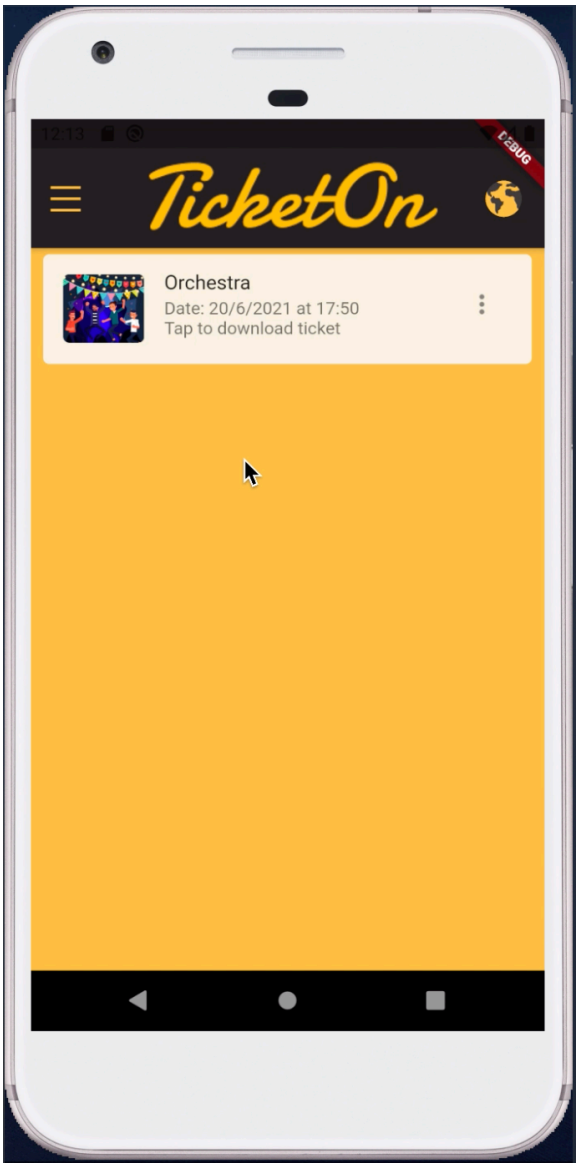


Рисунок 7 – Список придбаних квитків

Окрім перегляду, на цьому екрані можна завантажити квиток і повернути його. Для цього треба натиснути на три крапки в лівій частині елементу квитка і натиснути кнопку “Download ticket” або “Return ticket” відповідно.

Рисунок 8 – Вікно можливих дій з придбаним квитком

Окрім пошуку подій в списку, можна ще переглянути події поблизу себе на мапі. Для цього необхідно натиснути на кнопку з глобусом в правій верхній частині екрану зі списком подій. Користувач побачить своє місцезнаходження і відмітки подій поблизу з підписами їх назв.

Рисунок 9 – Екран пошуку подій поблизу користувача на мапі

При вході на концерт, необхідно показати квиток організатору, а йому, в свою чергу, треба перевірити дійсність цього квитка і його власника. Для цього організатор переходить на екран перевірки дійсності квитка, відвідувач показує йому QR-код квитка, організатор його фотографує і отримує повідомлення чи дійсний цей квиток.

Рисунок 10 – Екран перевірки дійсності квитка

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ КОНЦЕРНОЇ ДІЯЛЬНОСТІ МУЗИЧНИХ КОЛЕКТИВІВ

Програма та методика тестування

КПІ.ІП-7112.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ А.Ю. Черненький

Нормоконтроль:

_____ К.І. Ліщук

Виконавці:

_____М.Є. Кувічка

Київ – 2021 року

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАНЬ 3

2 МЕТА ТЕСТУВАННЯ 4

3 МЕТОДИ ТЕСТУВАННЯ 4

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ 5

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Система для автоматизації підтримки концертної діяльності музичних колективів представлена мобільним додатком та сервером. Окрім взаємодії між собою, система ще взаємодіє з такими зовнішніми сервісами як Google Maps API та LiqPay API. Для зв’язку компоненти використовують REST API та побудовані на основі бібліотеки Express.js.

2 МЕТА ТЕСТУВАННЯ

Метою проведення тестування є перевірка якості системи для гарантування коректності її роботи. У процесі тестування повинні бути перевірені такі пункти:

- можливість створення користувача;
- можливість створення події;
- можливість завантаження квитка;
- можливість перегляду повного списку доступних подій;
- можливість перегляду списку квитків певного користувача;
- можливість перевірки дійсності квитка;
- можливість повернення придбаного квитка.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування створеного програмного забезпечення було обрано такі методи тестування:

– функціональне тестування, за допомогою якого можна перевірити правильність виконання поставлених задач та відповідність усім вимогам до програмного продукту;

– інтеграційне тестування, за допомогою якого перевіряється коректність роботи системи в цілому;

– тестування макету, що допомагає перевірити чи відповідає відображення розробленої клієнтської частини дизайнам.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування проводиться за допомогою бібліотеки Jest.js використовуючи командну оболонку bash.

Коректність роботи програмного продукту перевіряється таким чином:

– ручне тестування шляхом надсилання запитів до серверної частини через додаток Postman, які містять некоректні або граничні дані;

– ручне тестування відображення компонентів мобільного додатку на пристроях з різним розміром екрану;

– автоматичне тестування працездатності основних логічних компонентів програми під час розгортання програмного продукту;

– тестування навантаженням при користуванні системою великою кількістю користувачів одночасно.

Коректність інтеграції з зовнішними сервісами перевіряється за допомогою відправки запитів через технологію webhook з цих сервісів.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ КОНЦЕРТНОЇ ДІЯЛЬНОСТІ МУЗИЧНИХ КОЛЕКТИВІВ

Технічне завдання

КПІ.ІП-7112.045490.05.91

“ПОГОДЖЕНО”

Керівник проекту:

А.Ю. Черненький

Нормоконтроль:

К.І. Ліщук

Виконавець:

М.Є. Кувічка

Київ – 2021 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ.....2

2 ПІДСТАВА ДЛЯ РОЗРОБКИ.....4

3 ПРИЗНАЧЕННЯ РОЗРОБКИ 5

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6

4.1 Вимоги до функціональних характеристик.....6

4.2 Вимоги до надійності.....6

4.3 Умови експлуатації.....7

4.4 Вимоги до складу і параметрів технічних засобів.....7

4.5 Вимоги до інформаційної та програмної сумісності 7

4.6 Вимоги до маркування та пакування.....7

4.7 Вимоги до транспортування та зберігання 8

4.8 Спеціальні вимоги 8

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ 9

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ 10

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ 11

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення підтримки концертної діяльності музичних колективів.

Галузь застосування:

Наведене технічне завдання поширюється на розробку «Програмне забезпечення підтримки концертної діяльності музичних колективів» котра використовується для організації концертів і спрощення пошуку варіантів проведення дозвілля та призначена для створення оголошення з описом події, перевірки квитків при допуску відвідувача на вхід, перегляду цікавих подій поблизу та придбання квитків через створену систему.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «Програмне забезпечення підтримки концертної діяльності музичних колективів» є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут» ім. Ігоря Сікорського (КПІ ім. Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спрощення проведення концерту як для організатора, так і для користувача.

Метою розробки є:

– економія часу та фінансів для організаторів;

– надання можливості пошуку подій поблизу клієнта;

– зменшення часу на пошук та купівлю квитка.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для організатора:

- реєстрація за номером телефону;
- авторизація;
- створення події;
- перегляд списку створених подій;
- перегляд статистики по створених подіях;
- можливість зміни кількості доступних квитків для події;
- можливість перевірки дійсності квитка.

4.1.1.2 Для клієнта:

- реєстрація за номером телефону;
- авторизація;
- перегляд повного списку подій;
- наявність пошуку та фільтрів для списка подій;
- перегляд списку подій поблизу себе на мапі;
- можливість придбати квиток;
- можливість завантажити квиток.

4.1.2 Розробку виконати на мові програмування Dart у вигляді мобільного застосунку.

4.1.3 Додаткові вимоги відсутні.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно з СанПін 2.2.2.542 – 96.

4.3.2 Обслуговування - вимоги не висуваються.

4.3.3 Обслуговуючий персонал - вимоги не висуваються.

4.4 Вимоги до склада і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на смартфонах під управлінням операційної системи Android.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору – Qualcomm Snapdragon 636.

4.4.2.2 Об’єм ОЗП - 2048 Мб.

4.4.2.3 Підключення до мережі Інтернет зі швидкістю від 100 мегабіт/сек.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Android версії 4.4.2 та вище.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: HTTP RESTful запитів із тілом у форматі JSON або x-www-form-url-encoded.

4.5.3 Результати повинні бути представлені в наступному форматі:

– 4.5.3.1 RESTful HTTP відповіді від сервера;

– 4.5.3.2 Зображення в форматі png.

4.5.4 Програмне забезпечення повинно бути розроблено на мовах Dart та JavaScript з використання бібліотек express для серверної частини та бібліотеки Flutter для клієнтської частини. База даних – PostgreSQL версії 13.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 40 аркушах формату А4 (без додатків 5.3.2 - 5.3.5).

5.3.2 Керівництво користувача.

5.3.3 Програма та методика тестування.

5.3.4 Технічне завдання.

5.3.5 Опис програми.

5.4 Графічна частина повинна бути виконана на аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Креслення вигляду екранних форм.

5.4.2 Схема структурна варіантів використань.

5.4.3 Схема бізнес-процесу перевірки дійсності квитка.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Таблиця 6.1 – Стадії та етапи розробки

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	09.02.2021	
2.	Розробка технічного завдання	04.04.2021	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.02.2021	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	03.03.2021	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	10.04.2021	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	15.04.2021	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	02.05.2021	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	11.05.2021	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	16.05.2021	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до документу “Програма та методика тестування”.

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
“___” _____ 2021 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ КОНЦЕРТНОЇ ДІЯЛЬНОСТІ МУЗИЧНИХ КОЛЕКТИВІВ

Опис програми

КПІ.ІП-7112.045490.06.13

“ПОГОДЖЕНО”
Керівник проєкту:
А. Ю. Черненький

Нормоконтроль:
К.І. Ліщук_____

Виконавець:
М. Є. Кувічка

Київ – 2021 року

Тексти програмного коду

Програмне забезпечення підтримки концертної діяльності музичних колективів

(Найменування програми (документа))

DVD-R

(Вид носія даних)

29 арк, 33.1 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2021 року

EventPreview.dart

```
import 'package:flutter/material.dart';

import '../entities/Event.dart';
import '../utils/helpers.dart';

class EventPreview extends StatelessWidget {
  final Event currentEvent;
  final Function onEventTap;

  EventPreview({ this.currentEvent, this.onEventTap });

  onCardTap() {
    onEventTap(currentEvent);
  }

  wrapInContainer(context, child) {
    return Container(
      padding: EdgeInsets.all(3),
      decoration: BoxDecoration(
        borderRadius: BorderRadius.all(Radius.circular(10)),
        color: Theme.of(context).backgroundColor,
      ),
      child: child,
    );
  }

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onCardTap,
      child: Card(
        shape: BeveledRectangleBorder(
          borderRadius: BorderRadius.circular(5),
        ),
        child: Container(
          height: 175,
          decoration: BoxDecoration(
            image: DecorationImage(
              image: NetworkImage(currentEvent.imageUrl),
              fit: BoxFit.cover,
            ),
            borderRadius: BorderRadius.all(Radius.circular(5))
```

```
        ),
        child: Stack(
          children: <Widget>[
            Positioned(
              top: 5,
              left: 5,
              child: wrapInContainer(context, Text(currentEvent.title)),
            ),
            Positioned(
              top: 30,
              left: 5,
              child: wrapInContainer(context, Text(currentEvent.locationAddress)),
            ),
            Positioned(
              bottom: 5,
              left: 5,
              child: wrapInContainer(context,
Text(formatDate(currentEvent.eventDate))),
            ),
            Positioned(
              bottom: 5,
              right: 5,
              child: wrapInContainer(context, Text(
                '${currentEvent.price.toString()}₴',
                style: new TextStyle(
                  fontSize: 20.0,
                  fontWeight: FontWeight.w700,
                ),
              )),
            ),
          ]
        ),
      ),
    ),
  );
 }
}
```

InputField.dart

```
import 'package:flutter/material.dart';

class _InputFieldState extends State<InputField> {
  void onFieldSubmit (value) {
```

```
  if (widget.nextField != null) {
    widget.nextField.requestFocus();
  } else {
    widget.nextField.unfocus();
  }
}

bool checkIfNeedValidation () {
  if (widget.type == 'email' || widget.type == 'password') {
    return true;
  }
  return false;
}

@override
Widget build(BuildContext context) {
  return Column(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      Text(
        widget.title,
        style: TextStyle(
          fontSize: 18,
          color: Theme.of(context).primaryColor,
        ),
      ),
      SizedBox(height: 10),
      Material(
        borderRadius: BorderRadius.circular(18),
        color: Theme.of(context).backgroundColor,
        child: TextFormField(
          obscureText: widget.type == 'password',
          autovalidate: this.checkIfNeedValidation(),
          onChanged: widget.onChange,
          onFieldSubmitted: this.onFieldSubmit,
          textInputAction: TextInputAction.next,
          keyboardType: TextInputType.emailAddress,
          validator: widget.validator,
        ),
      )
    ],
  );
}
```

```
}

class InputField extends StatefulWidget {
  InputField({
    Key key,
    this.title,
    this.onChange,
    this.validator,
    this.type,
    this.nextField
  }) : super(key: key);

  final String title;
  final void Function(String) onChange;
  final String Function(String) validator;
  final String type; // email, password, primary-text, number, etc.
  final FocusNode nextField;

  @override
  _InputFieldState createState() => _InputFieldState();
}
```

Event.dart

```
class Event {
  final int eventId;
  final String title;
  final int price;
  final String locationAddress;
  final double longitude;
  final double latitude;
  final DateTime eventDate;
  final String imageUrl;
  final int maxVisitors;
  final String description;
  final int ticketsBought;

  Event({this.eventId, this.title, this.price, this.locationAddress, this.longitude,
this.latitude, this.eventDate, this.imageUrl, this.maxVisitors, this.ticketsBought,
this.description});

  factory Event.fromJson(dynamic json) {
    return Event(
      eventId: json['id'],
```

```
      title: json['title'],
      price: json['price'],
      locationAddress: json['location'],
      longitude: json['longitude'],
      latitude: json['latitude'],
      eventDate: DateTime.parse(json['start_date']),
      imageUrl: json['image_url'],
      maxVisitors: json['max_visitors'],
      ticketsBought: int.parse(json['ticketsBought']),
      description: json['description'],
    );
  }
}
```

AuthScreen.dart

```
import 'package:flutter/material.dart';

class _AuthScreenState extends State<AuthScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      resizeToAvoidBottomInset: true,
      backgroundColor: Theme.of(context).accentColor,
      body: Center(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Container(
                padding: EdgeInsets.symmetric(horizontal: 60),
                child: Image.asset("assets/logo.png"),
              ),
              Container(
                margin: EdgeInsets.symmetric(horizontal: 50),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  crossAxisAlignment: CrossAxisAlignment.stretch,
                  children: <Widget>[
                    SizedBox(
                      height: 20,
                    ),
                    widget.form,
                  ],
```

```
              ),
            ),
          ],
        ),
      ),
    ),
  );
 }
}

class AuthScreen extends StatefulWidget {
 AuthScreen({Key key, this.form}) : super(key: key);

 final StatefulWidget form;

 @override
 _AuthScreenState createState() => _AuthScreenState();
}
```

MapScreen.dart

```
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:flutter/material.dart';
import 'dart:async';
import 'dart:convert';

import '../entities/Event.dart';
import '../utils/api.dart';

class MapScreenState extends State<MapScreen> {
 Completer<GoogleMapController> _controller = Completer();
 final Map<String, Marker> _markers = {};

 @override
 void initState() {
  super.initState();
 }

 List<Event> parseResponse(response) {
  var eventsJSON = jsonDecode(response.body)["eventList"] as List;

  List<Event> parsedEvents = eventsJSON.map((eventJSON) =>
Event.fromJson(eventJSON)).toList();
  return parsedEvents;
```

```
}

Future getEventList() async {
  final response = await getRequest('event/list');
  final List<Event> parsedEvents = parseResponse(response);

  return parsedEvents;
}

void onEventTap(event) {
  Navigator.pushNamed(
    context,
    '/eventDetails',
    arguments: event,
  );
}

Future<void> _onMapCreated(GoogleMapController controller) async {
  final List<Event> events = await getEventList();
  setState(() {
    _markers.clear();
    for (final event in events) {
      print('${event.title}: ${event.latitude}, ${event.longitude}');

      final marker = Marker(
        markerId: MarkerId(event.title),
        position: LatLng(event.latitude, event.longitude),
        infoWindow: InfoWindow(
          title: event.title,
          snippet: event.locationAddress,
          onTap: () => onEventTap(event),
        ),
      );
      _markers[event.title] = marker;
    }
  });
}

@override
Widget build(BuildContext context) {
  return new Scaffold(
    body: GoogleMap(
      mapType: MapType.normal,
      initialCameraPosition: CameraPosition(
```

```
          target: LatLng(50.4502531, 30.4514705),
          zoom: 16,
        ),
        onMapCreated: _onMapCreated,
        markers: _markers.values.toSet(),
      ),
    );
  }
}

class MapScreen extends StatefulWidget {
  @override
  State<MapScreen> createState() => MapScreenState();
}
```

ValidateTicketScreen.dart

```
import 'package:path_provider/path_provider.dart';
import 'package:path/path.dart' show join;
import 'package:http/http.dart' as http;
import 'package:flutter/material.dart';
import 'package:camera/camera.dart';
import 'dart:async';
import 'dart:io';

import '../components/Header.dart';
import '../components/SideBar.dart';
import '../components/Spinner.dart';
import '../utils/api.dart';

class ValidateTicketScreen extends StatefulWidget {
  final CameraDescription camera;

  const ValidateTicketScreen({
    Key key,
    @required this.camera,
  }) : super(key: key);

  @override
  ValidateTicketScreenState createState() => ValidateTicketScreenState();
}

class ValidateTicketScreenState extends State<ValidateTicketScreen> {
  CameraController camera;
```

```
  Future<void> _initializeControllerFuture;

  @override
  void initState() {
    super.initState();
    camera = CameraController(
      widget.camera,
      ResolutionPreset.medium,
    );

    _initializeControllerFuture = camera.initialize();
  }

  @override
  void dispose() {
    camera.dispose();
    super.dispose();
  }

  _asyncFileUpload(filePath) async{
    var request = http.MultipartRequest("POST",
Uri.parse(getApi('ticket/validate')));

    var image = await http.MultipartFile.fromPath("file_field", filePath);

    request.files.add(image);
    var response = await request.send();

    //Get the response from the server
    var responseData = await response.stream.toBytes();
    var responseString = String.fromCharCodes(responseData);

    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Your
ticket is valid!')));
    print(responseString);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: Header(),
      drawer: SideBar(),
      body: FutureBuilder<void>(
        future: _initializeControllerFuture,
```

```
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.done) {
            return CameraPreview(camera);
          } else {
            return Center(child: Spinner());
          }
        },
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.camera_alt),
        onPressed: () async {
          try {
            await _initializeControllerFuture;
            final path = join(
              (await getExternalStorageDirectory()).path,
              '${DateTime.now()}.png',
            );
            print(path);

            await camera.takePicture(path);

            _asyncFileUpload(path);

            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => DisplayPictureScreen(imagePath: path),
              ),
            );
          } catch (e) {
            print(e);
          }
        },
      ),
    );
  }
}

class DisplayPictureScreen extends StatelessWidget {
  final String imagePath;

  const DisplayPictureScreen({Key key, this.imagePath}) : super(key: key);

  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: Header(),
      drawer: SideBar(),
      // The image is stored as a file on the device. Use the `Image.file`
      // constructor with the given path to display the image.
      body: Image.file(File(imagePath)),
    );
  }
}
```

api.dart

```
import 'package:downloads_path_provider/downloads_path_provider.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:http/http.dart' as http;
import 'package:dio/dio.dart';
import 'dart:io';

import 'exceptions.dart';
import 'dart:convert';

const backend_host = 'https://44d9fdb1e7a3.ngrok.io'; //localhost address

String getApi(route) => '$backend_host/$route';

String token = '';

respond(response) {
  if (response.statusCode == 401) throw UnauthorizedException();
  if (response.statusCode == 403) throw ForbiddenException();
  if (response.statusCode == 400) throw BadRequestException();
  if (response.statusCode == 500) throw UnhandledException();
  return response;
}

getRequest(route) async {
  // final token = await getFieldFromStorage('token');
  return http.get(
      getApi(route),
      headers: {
      HttpHeaders.contentTypeHeader: "application/json",
      HttpHeaders.authorizationHeader: 'Bearer $token'
```

```
      }
    ).then((response) => respond(response));
}

postRequest(route, params) async {
  // final token = await getFieldFromStorage('token');
  return http.post(
      getApi(route),
      headers: {
        HttpHeaders.contentTypeHeader: "application/json",
        HttpHeaders.authorizationHeader: 'Bearer $token'
      },
      body: json.encode(params)
    ).then((response) => respond(response));
}

downloadTicket(eventId, eventTitle) async {
  Dio dio = new Dio();
  final downloadsDirectory = await DownloadsPathProvider.downloadsDirectory;

  var response;
  if (await Permission.storage.request().isGranted) {
    response = await dio.download(
      getApi('event/downloadTicket?event_id=$eventId'),
      '${downloadsDirectory.path}/$eventTitle-ticket.png',
      options: Options(
        headers: {
          HttpHeaders.contentTypeHeader: "application/json",
          HttpHeaders.authorizationHeader: 'Bearer $token'
        },
      ),
    );
  }
  print (response.data);
}

saveToken(newToken) => token = newToken;

getValueFromStorage(fieldName) async {
  Future<SharedPreferences> _prefs = SharedPreferences.getInstance();
  final SharedPreferences prefs = await _prefs;
  prefs.getString(fieldName);
}
```

```
setValueToStorage(fieldName, value) async {
  Future<SharedPreferences> _prefs = SharedPreferences.getInstance();
  final SharedPreferences prefs = await _prefs;

  if (value is String) prefs.setString(fieldName, value);
  if (value is int) prefs.setInt(fieldName, value);
}
```

theme.dart

```
import 'package:flutter/material.dart';

const blackColor = Color(0xFF242121);
const yellowColor = Color(0xFFF8C25C);
const redColor = Color(0xFFF52D2D);
const whiteColor = Color(0xFFF8F3E3);

class Themes {
  static final ThemeData main = ThemeData(
    primaryColor: yellowColor,
    accentColor: blackColor,
    highlightColor: redColor,
    backgroundColor: whiteColor,
  );
}
```

validation.dart

```
final RegExp emailRegexp = RegExp(r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-
/=?^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+");
final RegExp passwordRegexp = RegExp(r"^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[0-9a-
zA-Z]{6,}$");
```

main.dart

```
import 'package:flutter/material.dart';
import 'package:camera/camera.dart';

import './utils/theme.dart';
import './screens/FirstScreen.dart';
import './screens/LoginScreen.dart';
import './screens/EventListScreen.dart';
import './screens/EventDetailsScreen.dart';
import './screens/MyTicketsScreen.dart';
```

```
import './screens/ValidateTicketScreen.dart';
import './screens/MapScreen.dart';
import './screens/CreateEventScreen.dart';
import './screens/PaymentScreen.dart';

void main() async {
  // Ensure that plugin services are initialized so that `availableCameras()`
  // can be called before `runApp()`
  WidgetsFlutterBinding.ensureInitialized();

  // Obtain a list of the available cameras on the device.
  final cameras = await availableCameras();

  // Get a specific camera from the list of available cameras.
  final firstCamera = cameras.first;

  runApp(MyApp(camera: firstCamera));
}

class MyApp extends StatelessWidget {
  final CameraDescription camera;

  MyApp({ this.camera });

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: Themes.main,
      initialRoute: '/',
      routes: {
        '/': (context) => FirstScreen(),
        '/email': (context) => LoginScreen(),
        '/register': (context) => LoginScreen(),
        '/eventList': (context) => EventListScreen(),
        '/eventDetails': (context) => EventDetailsScreen(),
        '/myTickets': (context) => MyTicketsScreen(),
        '/validate': (context) => ValidateTicketScreen(camera: camera),
        '/map': (context) => MapScreen(),
        '/createEvent': (context) => CreateEventScreen(),
        '/checkout': (context) => PaymentScreen(),
      },
    );
  }
}
```

app.js

```
const express = require('express');
const path = require('path');
const cookieParser = require('cookie-parser');
const bodyParser = require('body-parser');
const loggerMorgan = require('morgan');

const indexRouter = require('./app/routes');
const userRouter = require('./app/routes/user');
const eventRouter = require('./app/routes/event');
const ticketRouter = require('./app/routes/ticket');

const { MainErrorHandler } = require('./app/utils/errors');
const { usePassport } = require('./app/utils/passport');

const app = express();

// view engine setup
app.set('views', path.join(__dirname, 'app/views'));
app.set('view engine', 'ejs');// may cause error: change to 'pug'

app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE,
HEAD, OPTIONS');
  res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-
Type, Accept, Authorization');
  next();
});

app.use(loggerMorgan('[:date[iso]] :method :url :status :response-time ms ---
:res[content-length]'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'app/public')));

app.use('/', indexRouter);
app.use('/user', userRouter);
app.use('/event', eventRouter);
app.use('/ticket', ticketRouter);
```

```
// initiate passport
usePassport();

// catch 404 and forward to error handler
app.use((req, res, next) => {
  const err = new Error('Not Found');
  err.status = 404;
  next(err);
});

app.use(MainErrorHandler);

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

create_DB.sql

```
CREATE TABLE "users" (
  "id" serial PRIMARY KEY,
  "email" varchar UNIQUE,
  "password" varchar,
  "full_name" varchar,
  "photo_url" varchar,
  "date_of_birth" date,
  "token" varchar,
  "email_verification_token" varchar,
  "reset_password_token" varchar,
  "created_at" timestamp DEFAULT (now()),
  "deleted_at" timestamp
);

CREATE TABLE "events" (
  "id" serial PRIMARY KEY,
  "title" varchar NOT NULL,
```

```
  "location" varchar NOT NULL,
  "price" float NOT NULL,
  "event_runner" varchar,
  "start_date" timestamp NOT NULL,
  "end_date" timestamp,
  "max_visitors" int NOT NULL,
  "description" varchar,
  "created_at" timestamp DEFAULT (now()),
  "deleted_at" timestamp
);

CREATE TABLE "event_images" (
  "id" serial PRIMARY KEY,
  "event_id" int,
  "image_url" varchar
);

CREATE TABLE "tickets" (
  "id" serial PRIMARY KEY,
  "event_id" int,
  "user_id" int,
  "created_at" timestamp DEFAULT (now()),
  "deleted_at" timestamp
);

CREATE UNIQUE INDEX email_unique_index ON users(email) WHERE
deleted_at IS NULL;

ALTER TABLE "event_images" ADD FOREIGN KEY ("event_id")
REFERENCES "events" ("id");

ALTER TABLE "tickets" ADD FOREIGN KEY ("event_id") REFERENCES
"events" ("id");

ALTER TABLE "tickets" ADD FOREIGN KEY ("user_id") REFERENCES
"users" ("id");
```

downloadTicketController.js

```
const QRCode = require('qrcode');
const path = require('path');

const {
  findEventById,
```

```
} = require('../../../services/event');
const { NotFoundError, BadRequestError } = require('../../../utils/errors');

const downloadTicketController = async (req, res) => {

  const { id: user_id } = req.user;
  const { event_id } = req.query;

  const eventFromDB = await findEventById(event_id);
  if (!eventFromDB) throw new NotFoundError('Event was deleted or already
run');

  try {
    const encryptedString = JSON.stringify({ event_id, user_id });
    await QRCode.toFile(__dirname + '/../../../../images/qrcode.png',
encryptedString, { type: 'png' });

    return res.sendFile(path.resolve('images/qrcode.png'));
    // return res.sendStatus(200);
  } catch (err) {
    console.log('Error while running transaction');
    if (err.errors) {
      throw new BadRequestError('Something went wrong');
    }
    throw err;
  };
};

module.exports = downloadTicketController;
```

getEventListController.js

```
const {
  getEventsData,
} = require('../../../services/event');

const getEventListController = async (req, res) => {
  try {
    const eventList = await getEventsData();

    return res.status(200).json({ eventList });
  } catch (err) {
    console.log('Error while getting event list');
    if (err.errors) {
```

```
      throw new BadRequestError('Something went wrong');
    }
    throw err;
  };
};

module.exports = getEventListController;
```

returnTicketController.js

```
const Joi = require('joi');

const {
  ticketSchema,
} = require('../../../utils/validation');
const {
  returnTicket,
} = require('../../../services/event');

const returnTicketController = async (req, res) => {
  const validation = Joi.validate(req.body, ticketSchema);

  // return error if smth wrong with fields
  if (validation.error) {
    throw new BadRequestError(validation.error.details[0].message);
  }

  const { id: userId } = req.user;
  const { event_id } = req.body;

  try {
    await returnTicket(event_id, userId);

    return res.sendStatus(200);
  } catch (err) {
    console.log('Error while returning ticket');
    if (err.errors) {
      throw new BadRequestError('Something went wrong');
    }
    throw err;
  };
};

module.exports = returnTicketController;
```

vlidateTicketController.js

```
const QrCode = require('qrcode-reader');
const path = require('path');
const Jimp = require("jimp");

const {
  findEventById,
  activateTicket,
} = require('../../../services/event');
const { NotFoundError, BadRequestError } = require('../../../utils/errors');

const validateTicketController = async (req, res) => {
  const img = req.files[0];

  const image = await Jimp.read(img.buffer);
  const qr = new QrCode();

  const value = await new Promise((resolve, reject) => {
    qr.callback = (err, v) => err != null ? reject(err) : resolve(v);
    qr.decode(image.bitmap);
  });

  const result = JSON.parse(value.result);

  const { event_id, user_id } = result;
  console.log(event_id, user_id);

  const eventFromDB = await findEventById(event_id);
  if (!eventFromDB) throw new NotFoundError('Event was deleted or already
run');

  try {
    await activateTicket(event_id, user_id);
    return res.status(200).json({ isValid: true });
  } catch (err) {
    console.log('Error while validation process');
    if (err.errors) {
      throw new BadRequestError('Something went wrong');
    }
    throw err;
  };
};
```

```
module.exports = validateTicketController;
```

loginController.js

```
const Joi = require('joi');

const {
  authSchema,
} = require('../../../utils/validation');
const {
  findUserByEmail,
  checkIfPasswordIsCorrect,
  loginUser,
} = require('../../../services/user');
const { NotFoundError, BadRequestError } = require('../../../utils/errors');

const loginController = async (req, res) => {
  // validate text fields
  const validation = Joi.validate(req.body, authSchema);

  // return error if smth wrong with fields
  if (validation.error) {
    throw new BadRequestError(validation.error.details[0].message);
  }

  const { email, password } = req.body;

  const lowerCaseEmail = email.toLowerCase();

  // find user by email
  const userFromDB = await findUserByEmail(lowerCaseEmail);

  if (!userFromDB[0]) {
    throw new NotFoundError('User with such email and password combination not
found');
  }

  // if password is correct -> login user
  // if password is not correct -> return error
  if (userFromDB[0] && checkIfPasswordIsCorrect(password,
userFromDB[0].password)) {
    const loggedInUser = await loginUser(userFromDB[0].email);
    return res.status(201).json({ user: loggedInUser[0] });
```

```
  }
  throw new BadRequestError('Password or login is not correct');
};

module.exports = loginController;
```

registerController.js

```
const Joi = require('joi');

const {
  authSchema,
} = require('../../../utils/validation');
const {
  findUserByEmail,
  checkIfPasswordIsCorrect,
  loginUser,
  createUser,
} = require('../../../services/user');
const { NotFoundError, BadRequestError } = require('../../../utils/errors');

const loginController = async (req, res) => {
  // validate text fields
  const validation = Joi.validate(req.body, authSchema);

  // return error if smth wrong with fields
  if (validation.error) {
    throw new BadRequestError(validation.error.details[0].message);
  }

  const { email, password } = req.body;

  const lowerCaseEmail = email.toLowerCase();

  // find user by email
  const userFromDB = await findUserByEmail(lowerCaseEmail);

  if (userFromDB[0]) {
    throw new NotFoundError('User with such email is already registered');
  }

  try {
    const userModel = await createUser({ email: lowerCaseEmail, password });
    console.log(`User with email ${userModel.email} created successfully`);
```

```
    return res.status(201).send({ user: userModel });
  } catch (err) {
    console.log(`Error while creating user with email ${email}`);
    if (err.errors && err.errors[0].validatorKey === 'not_unique') {
      throw new BadRequestError(`User with email ${email} already registered`);
    }
    throw err;
  }
};

module.exports = loginController;
```

event.js

```
const knex = require('../utils/knex');

const eventBasicModel = ['id', 'title', 'location', 'longitude', 'latitude', 'price',
'start_date', 'max_visitors', 'description'];

const findEventById = (event_id) => knex('events')
  .where({ id: event_id })
  .where('start_date', '>=', knex.raw('NOW()'))
  .select(eventBasicModel);

const getEventsData = async (where) => {
  const fields = eventBasicModel.map(field =>
`events.${field}`).concat(['event_images.image_url']);

  const events = await knex('events')
    .where({ 'events.deleted_at': null, ...where })
    .where('events.start_date', '>=', knex.raw('NOW()'))
    .join('event_images', 'events.id', '=', 'event_images.event_id')
    .leftJoin('tickets', 'events.id', '=', 'tickets.event_id')
    .count('tickets.id AS ticketsBought')
    .groupBy(fields)
    .orderBy('id')
    .select(fields);

  return events;
};

const activateTicket = (event_id, user_id) => knex('tickets')
  .where({ event_id, user_id })
```

```
  .whereNull('deleted_at')
  .update({ deleted_at: knex.raw('NOW()') });

const buyTicket = (event_id, user_id) => knex('tickets').insert({ event_id, user_id
});

const returnTicket = (event_id, user_id) => knex('tickets')
  .where({ 'event_id': event_id, 'user_id': user_id })
  .update({ 'deleted_at': new Date() });

module.exports = {
  findEventById,
  getEventsData,
  buyTicket,
  activateTicket,
  returnTicket,
};
```

user.js

```
const knex = require('../utils/knex');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const {
  SECRET_JWT,
  FOUR_HOURS,
  SALT_ROUNDS,
  MILLI_SECOND,
} = require('../utils/constants');

const userBasicModel = ['email', 'full_name', 'photo_url', 'date_of_birth'];
const userFullModel = userBasicModel.concat(['token']);

const findUserByEmail = (email) => knex('users').where({ email, deleted_at: null
});

const updateUser = (what, where) => knex('users').where(where).update(what);

const createUser = (user) => {
  // generate token for 4 hours
  const token = jwt.sign({ exp: Math.floor(Date.now() / MILLI_SECOND) +
(FOUR_HOURS), email: user.email }, SECRET_JWT);
  const password = createPassword(user.password);
```

```
  return knex('users').returning(userFullModel).insert({ email: user.email,
password, token });
};

// returns only needed for user fields
const findUserByEmailForLoginRes = where => knex('users').where({ ...where,
deleted_at: null }).select(userFullModel);

// compares passwords
const checkIfPasswordIsCorrect = (reqPass, dbPass) =>
bcrypt.compareSync(reqPass, dbPass);

// create password and hash it
const createPassword = password => bcrypt.hashSync(password,
bcrypt.genSaltSync(SALT_ROUNDS));

// logins user
const loginUser = async (email) => {
  const where = { email };
  // generate token for 4 hours
  const token = jwt.sign({ exp: Math.floor(Date.now() / MILLI_SECOND) +
(FOUR_HOURS), email }, SECRET_JWT);

  // update db with new token
  await updateUser({ token }, where);
  // return updated user with needed fields
  return findUserByEmailForLoginRes(where);
};

module.exports = {
  findUserByEmail,
  updateUser,
  checkIfPasswordIsCorrect,
  createPassword,
  createUser,
  loginUser,
}
```

constants.js

```
const SECRET_JWT = 'tocketon';

const SALT_ROUNDS = 10;
const FOUR_HOURS = 4 * 60 * 60;
```

```
const MILLI_SECOND = 1000;

module.exports = {
  SECRET_JWT,
  FOUR_HOURS,
  SALT_ROUNDS,
  MILLI_SECOND,
}
```

passport.js

```
const knex = require('../utils/knex');
const passport = require('passport');
const BearerStrategy = require('passport-http-bearer');
const jwt = require('jsonwebtoken');
const { SECRET_JWT } = require('./constants');

function handleUnauthenticatedError(res) {
  const output = {
    error: {
      message: 'Not authenticated',
    },
  };
  const statusCode = 401;
  res.status(statusCode).json(output);
}

const authenticated = (req, res, next) => passport.authenticate('bearer', { session:
false }, (err, user) => {
  // throw 401 error in case token validation error happened or no token was
provided at all
  if (err || !user) {
    return handleUnauthenticatedError(res);
  }

  req.user = user;

  return next();
})(req, res, next);

const usePassport = () => passport.use(new BearerStrategy(async (token, done) =>
{
  // check if token expired
  let tokenExpired = false;
```

```
  await jwt.verify(token, SECRET_JWT, (err) => {
    if (err) {
      tokenExpired = true;
      return done(true, false);
    }
  });
  if (!tokenExpired) {
    await knex('users').where({ token, deleted_at: null }).then((user) => {
      if (!user[0]) {
        return done(true);
      } else {
        return done(null, user[0]);
      }
    });
  }
}));
```

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
“___” _____ 2021 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ КОНЦЕРТНОЇ ДІЯЛЬНОСТІ МУЗИЧНИХ КОЛЕКТИВІВ

Графічні матеріали

КПІ.ІП-7112.045490.07.99

“ПОГОДЖЕНО”
Керівник проєкту:
А. Ю. Черненький

Нормоконтроль:
К.І. Ліщук

Виконавець:
М. Є. Кувічка

Київ – 2021 року

Зм.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Кузічка М. С.			
Перев.		Черенький А. Ю.			
Т. Кон.					
Н. Кон.		Лищук К.І.			
Зате.		Черенький А. Ю.			

КПІ.ІП-7112.045490.07.99.КЕ

Кресленя вулядку екранних форм

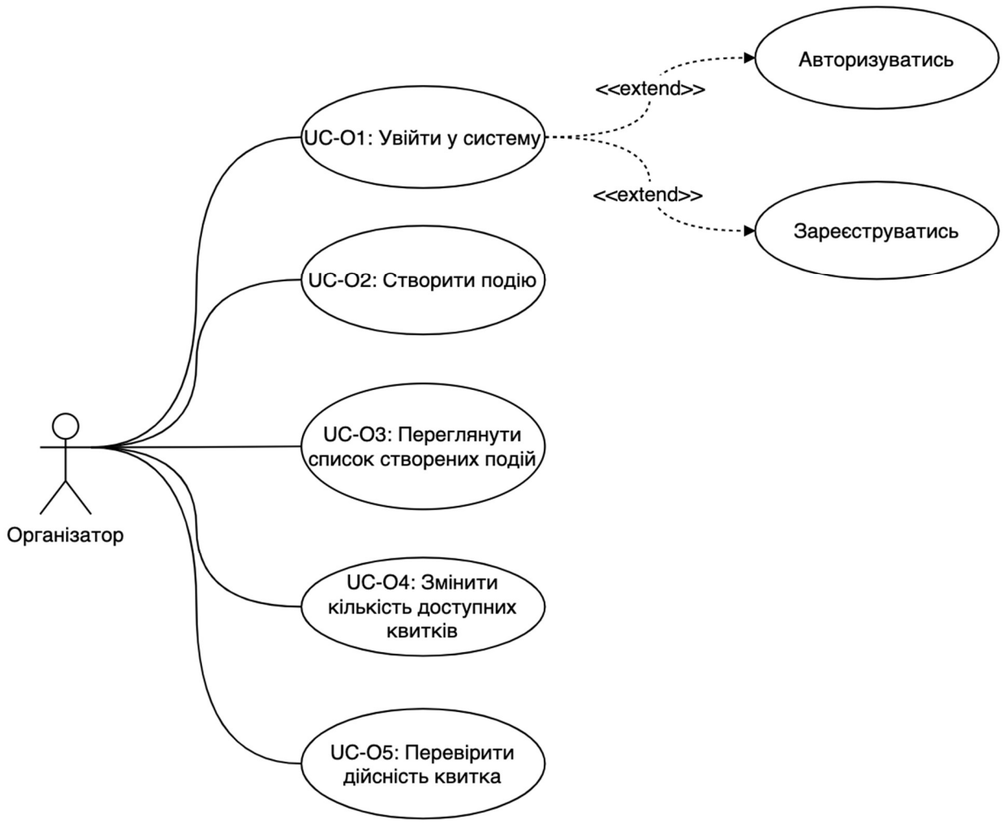
Літера	Маса	Масштаб

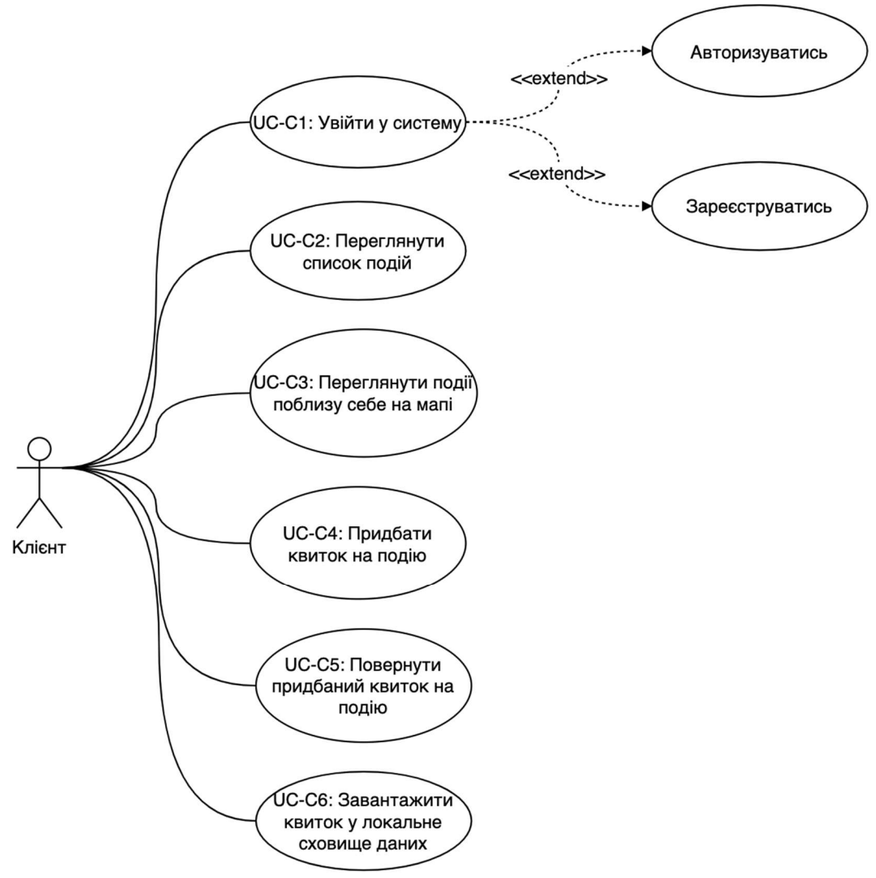
Аркуш 1 | Аркушів 3

Програмне забезпечення підтримки діяльності музичних колективів

КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-71

КПІ.ІП-7112.045490.07.99.СС





Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*
Розроб.		*Кувічка М. Є.*		
Перев.		*Черненький А. Ю.*		
Т. Кон.				
Н. Кон.		*Ліщук К.І.*		
Затв.		*Черненький А. Ю.*		

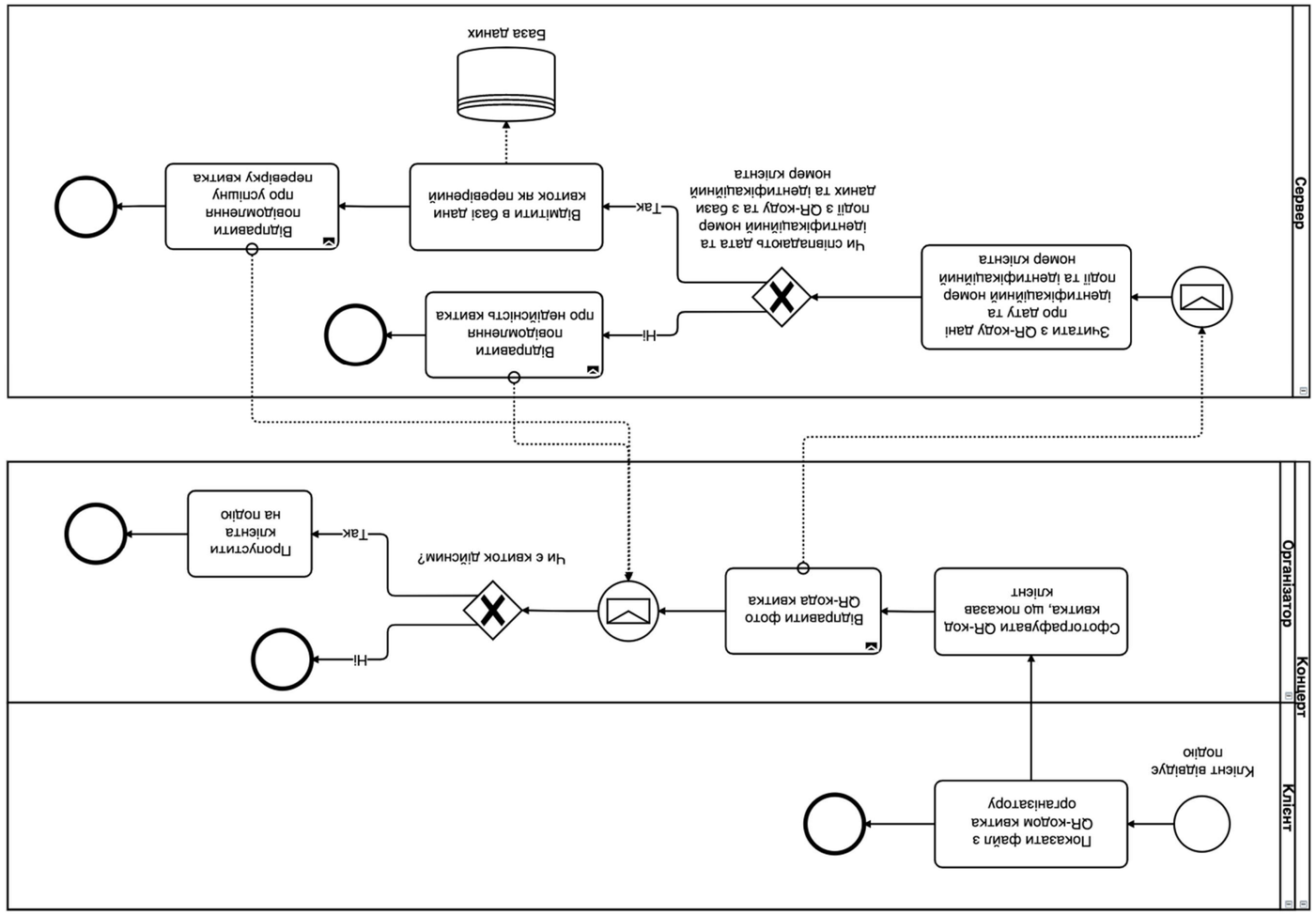
КПІ.ІП-7112.045490.07.99.СС

Схема структурна варіантів використань

Літера	*Маса*	*Масштаб*
Аркуш 2	*Аркушів 3*	

Програмне забезпечення підтримки діяльності музичних колективів

КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-71



Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розроб.		*Кузічка М. С.*			
Перев.		*Черенький А. Ю.*			
Т. Кон.					
Н. Кон.		*Лищук К.І.*			
Затв.		*Черенький А. Ю.*			

КПІ.ІП-7112.045490.07.99.СБП

Схема бізнес-процесу перевірки дійсності квитка

Літера	*Маса*	*Масштаб*

Аркуш 3 *Аркушів 3*

Програмне забезпечення підтримки діяльності музичних колективів

КПІ ім. Ігоря Сікорського кафедра АСОІУ
гр. ІП-71