МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ БРОНЮВАННЯ КВИТКІВ У ТЕАТРИ»

Рівень «бакалавр»
Галузь знань 12 «Інформаційні технології»,
спеціальність 122 «Комп’ютерні науки»
Виконала здобувачка 4 курсу
Сейрік Юлія Юріївна

Керівник к.пед.н., доцент
Лобода Юлія Геннадіївна

Рецензент к.т.н., доцент
Чикунов Павло Олександрович

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань: 12 Інформаційні технології
Спеціальність: 122 Комп'ютерні науки
Назва освітньої програми: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Н.І. Логінова
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачці Сейрік Юлії Юріївні

1. Тема роботи: «Інформаційна система для бронювання квитків у театри»
   Керівник роботи: к.пед.н., доцент Лобода Юлія Геннадіївна
   затверджені наказом НУ «ОЮА» від «02» квітня 2024 року № 809-06
2. Строк подання здобувачем роботи «20» травня 2024 року
3. Дата видачі завдання «15» вересня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Формування функціональних вимог	02.10.23-10.11.23	
2	Аналіз та вибір засобів розробки	12.11.23-20.12.23	
3	Розробка архітектури інформаційної системи	23.12.23-25.01.24	
4	Проектування бази даних	27.01.24-16.02.24	
5	Розробка серверної частини	18.02.24-21.03.24	
6	Розробка клієнтської частини засобами Python	22.03.24-27.04.24	
7	Оформлення звітної частини	29.04.24-17.05.24	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота студентки 4-го курсу Сейрік Юлії Юріївни на тему «Інформаційна система для бронювання квитків у театри» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 122 «Комп'ютерні науки», освітньою програмою «Комп'ютерні науки» містить 3 розділи, 65 рисунків, 9 таблиць, 3 додатки, 31 літературних джерел за переліком посилань. Робота виконана на 87 сторінках загального тексту і 72 сторінках основного тексту.

Метою роботи є створення інформаційної системи з функціоналом для здійснення броні квитків на вистави у театри, інформація про які знаходиться у базі даних.

Об’єктом дослідження є інформаційна система для бронювання квитків у театри.

Предметом дослідження є процес розробки інформаційної системи з графічним інтерфейсом користувача та підключеної до неї бази даних.

У кваліфікаційній роботі розроблено інформаційну систему для бронювання квитків у театри на вистави. Система має можливість реєстрації та авторизації користувача, перегляду персональних даних, вибору локації та театру, де буде проходити вистава, перегляду отриманих квитків, здійснення та скасування броні. Досліджено специфіку розробки інформаційної системи за допомогою засобів Python та бази даних MySQL.

Ключові слова: система, застосунок, бронювання, віджет, сайзер, клас, функція, MySQL, Python, wxPython.

ABSTRACT

The qualification work of the 4th year student Yulia Yuriivna Seirik on the topic ‘Information system for booking tickets to theatres’ for the first (bachelor's) level of higher education in the specialty 122 ‘Computer Science’, educational programmer ‘Computer Science’ contains 3 chapters, 65 figures, 9 tables, 3 appendices, 31 references. The work is executed on 87 pages of the general text and 72 pages of the main text.

The purpose of the work is to create an information system with functionality for booking tickets to theatres, information about which is in the database.

The object of research is an information system for booking tickets to theatres.

The subject of research is the process of developing an information system with a graphical user interface and a database connected to it.

In the qualification work, an information system for booking tickets to theatres for performances was developed. The system has the ability to register and authorize a user, view personal data, select the location and theatre where the performance will take place, view received tickets, make and cancel a reservation. The specifics of developing an information system using Python and the MySQL database are investigated.

Keywords: system, application, booking, widget, sizer, class, function, MySQL, Python, wxPython.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 6
ВСТУП 7
1 АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ 9
1.1 Аналіз предметної області 9
1.2 Аналіз наявних аналогів 13
1.3 Визначення функціональних вимог 17
1.4 Вибір засобів розробки 19
1.5 Висновки до розділу 1 23
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ 24
2.1 Архітектура програмного застосунку 24
2.2 Проектування бази даних 26
2.3 Моделювання програмної системи 31
2.4 Висновки до розділу 2 41
3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 42
3.1 Розробка серверної частини 42
3.2 Розробка клієнтської частини 49
3.3 Тестування застосунку 59
3.4 Висновки до розділу 3 67
ВИСНОВКИ 68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70
ДОДАТКИ 73
Додаток А. Результати SQL – запитів 73
Додаток Б. Лістинг програмного коду 79
Додаток В. Копії документів про апробацію результатів роботи 84

ПЕРЕЛІК СКОРОЧЕНЬ

MySQL – My Structured Query Language

ІС – Інформаційна система

АСК ТП – Автоматизована система керування технологічним процесом

АСКВ – Автоматизована система керування виробництвом

САП – Система автоматизованого проектування

АСНД – Автоматизована система наукових досліджень

АІПС – Автоматизована інформаційно-пошукова система

СППР – Система підготовки прийняття рішення

лат. – латинська

фр. – французька

ЕОМ – Електронна обчислювальна машина

БД – База даних

СУБД – Система управління базами даних

SQL – Structured Query Language

API – Application Programming Interface

X DevAPI – Cross-Platform Development Application Programming Interface

TCP – Transmission Control Protocol

IP – Internet Protocol

GTK2, GTK3 – GIMP Toolkit 2, GIMP Toolkit 3

ER - Entity-Relationship

DFD - Data Flow Diagram

UML - Unified Modeling Language

GUI - Graphical User Interface

ВСТУП

Актуальність теми дослідження обумовлена тим, що люди завжди хочуть бути впевнені у тому що річ або послуга, яку вони збираються придбати обов'язково буде в наявності, тому у сучасному технологічно розвиненому світі була створена неймовірно велика кількість різних додатків для бронювання будь-чого. Зараз кожен, не виходячи з дому, здатен, за допомогою телефону чи комп'ютера, забезпечити собі можливість спокійно переглянути всі можливі варіанти та вибрати той що до вподоби після чого, перевіривши його доступність, здійснити бронювання, тобто відмітити, і таким чином, інша людина не зможе придбати його раніше. Саме тому інформаційна система для бронювання квитків на різні вистави у театри країни, розроблена мовою програмування Python з використанням бази даних MySQL, може бути корисною для тих хто бажає поринути у світ мистецтва.

Об'єктом дослідження є інформаційна система для бронювання квитків у театри.

Предметом дослідження є процес розробки інформаційної системи з графічним інтерфейсом користувача та підключеної до неї бази даних.

Метою роботи є створення інформаційної системи з функціоналом для здійснення броні квитків на вистави у театри, інформація про які знаходиться у базі даних.

Для досягнення поставленої мети потрібно вирішити наступні завдання:

– розробка зрозумілого та легкого для використання інтерфейсу;

– створення бази даних, яка містить інформацію про театри країни, вистави які відбуваються у них, та користувачів які зареєстровані;

– реалізація функціоналу необхідного для реєстрації та авторизації користувача;

– реалізація функціоналу для здійснення броні, її видалення, запису про стан квитків у базу даних;

– реалізація функцій для перегляду особистих даних користувача, які були введені при реєстрації, та квитків які вже були заброньовані ним.

Методи дослідження роботи:

– теоретичний метод (пошук, аналіз та синтез інформації з різних джерел);

– емпіричний метод (експеримент, який спостерігається через дослідження програмного опрацювання даних, використовуючи сучасні комп’ютерні та інформаційні технології).

Практичне значення отриманих результатів дослідження: можливість користуватися зручним, зрозумілим інтерфейсом для того щоб легко та швидко бронювати квитки на вистави у театрах країни.

Результати кваліфікаційного дослідження апробовані на IX Всеукраїнської науково-практичній конференції «Інформаційне суспільство: проблеми та перспективи» (м. Одеса, 24 травня 2024 р.) [1] (Додаток В).

1 АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Аналіз предметної області

У нашій країні існує значна кількість театрів з різноманітними направленнями (жанрами) і звичайно з різними виставами, які вони проводять. Для створення інформаційної системи спрямованої на бронюванні квитків з інтерфейсом для користувачів потрібно враховувати численні фактори, наприклад для визначення ціни важливим є театр в якому відбувається виступ, розташування місця в залі (його номер, ряд, категорія), дата та час проведення тощо. Саме тому перш ніж почати проектування програмного продукту важливо розглянути основні аспекти цієї теми, включаючи визначення інформаційної системи та її видів.

Інформаційна система – це сукупність інформації, апаратно-програмних і технологічних засобів, засоби телекомунікації, баз і банків даних, методів процедур обробки даних, персоналу управління, які реалізують функції збирання, передавання, обробки та накопичування інформації для підготовки і прийняття ефективних управлінських рішень [2].

Інформаційні системи можуть значно різнитися за типами об'єктів управління в організаційно-економічних системах, характером та обсягом задач, які вони розв'язують, та низкою інших ознак [2].

Розглянемо класифікації інформаційних систем за основними ознаками:

– за рівнем автоматизації процесів управління: ручні, механізовані, автоматизовані, автоматичні;

– за призначенням: системи моделювання (системи штучного інтелекту), інформаційно-пошукові системи, інформаційно-довідкові системи, інформаційно-керуючі системи, навчальні та екзаменуючи системи, експертні системи [2];

– за сферою призначення: економічні ІС (призначені для виконання функцій управління на підприємстві); медичні ІС (призначені для використання в лікувальному або лікувально-профілактичному закладі); географічні ІС

(забезпечують збір, збереження, обробку, доступ, відображення і розповсюдження даних); адміністративні; виробничі; навчальні; екологічні; криміналістичні; військові, розважальні, культурні, та інші;

– за функціональним призначенням: системи керування (АСК ТП, АСКВ); проектувальні (САП); наукового пошуку (АСНД, АІПС, експертні системи); діагностичні, моделювальні; система підготовки прийняття рішення (СППР) [3].

За описаними вище ознаками, інформаційна система, що досліджується у цій роботі належить:

– за рівнем автоматизації - до автоматизованої ІС, оскільки частина функцій здійснюється через графічний інтерфейс та програмне забезпечення, а інша - через базу даних та програмні алгоритми;

– за призначенням - до інформаційно-керуючої системи, тому що вона забезпечує управління процесом бронювання квитків у театри;

– за сферою призначення - до розважальних і культурних ІС;

– за функціональним призначенням - до системи підготовки прийняття рішень (СППР), оскільки надає користувачам інформацію про доступні вистави та квитки, для прийняття рішень щодо бронювання.

Далі розглянемо основні аспекти театральної сфери.

Театральне мистецтво охоплює широкий спектр творчих видів діяльності, які об'єднуються для створення живої вистави. Ця спільна форма мистецтва об'єднує такі елементи, як акторська майстерність, режисура, сценографія, дизайн костюмів, освітлення, звук тощо, щоб передати історію або викликати емоції [4].

Театр (з грецької – місце для видовищ) – 1) рід мистецтва, який відображає дійсність у художніх сценічних образах; 2) видовище, спектакль; 3) приміщення, де відбуваються вистави [5].

Направлення або жанр театру – це спеціальна категорія, яка визначає тип і характер вистав, які відіграють у театрі. Він описує загальну тематику, стиль або тип вистав, які можуть включати в себе різноманітні жанри та естетичні підходи.

Як було зазначено вище театри мають багато різних направлень, і вони включають наступні найвідоміші жанри: драма, комедія, трагедія, трагікомедія,

музична комедія, опера, балет, пантоміма. Крім вже перерахованих, також є театри: експериментальний, театр сучасної драматургії, театр ляльок, фольклорний театр, музично-драматичний, театр для дітей та юнацтва, духовний театр, релігійний, театр тіней, дитячий театр тощо.

Вистава, спектакль (з лат. – видовище) – театральна постановка, в основі якої – будь-який драматичний твір або інсценізація епічного чи ліричного твору. Результат колективної праці акторів, режисерів, художників, композитора та інших представників театрального мистецтва. Складається з одного або декількох актів, дій, частин, що перериваються або не перериваються антрактом [5].

Афіша (з фр.) – оголошення про щось; театральна афіша - оголошення, переважно друковане, про виставу/спектакль. Вид реклами [5].

На початку аналізу предметної області було визначено що одним з факторів який вливає на встановлення ціни на квиток є дата та час проведення вистави. Тому важливим буде розглянути яким саме чином вони пов’язані. Залежність ціни квитка на виставу від дати та часу проведення може бути визначена кількома факторами, і нижче наведено основні з них:

– популярність дати та часу - вистави, які проводяться у вихідні дні та ввечері коштують дорожче за інші;

– спеціальні події та свята - на деякі вистави може бути збільшений попит тому і ціна на них також зростає;

– сезонність - у певні періоди року ціна на вистави може збільшуватися чи зменшуватися, в залежності від того яка пора року;

– продаж квитків у попередньому порядку - квитки на виступи які продаються заздалегідь можуть бути зі зниженою ціною.

Ще одним фактором, який не був згаданий вище, але так само впливає на ціну квитка, є актори, які приймають участь у виставі. Виступ відомих та престижних акторів, тобто популярних або тих хто має великий досвід гри на сцені або отримав нагороди за свою роботу, може призвести до підвищення вартості квитка, оскільки це збільшує інтерес глядачів.

Для розуміння принципів формування цін на квитки, також важливим буде розглянути відомості про план глядацької зали, яка містить дані про розподіл категорій місць та їх нумерації.

У театрах схема глядацького залу дуже часто відрізняється один від одного, що впливає на систему категоризації місць. Деякі театри можуть мати лише базові категорії, такі як партер, ложі та балкони, тоді як інші можуть мати більш складні системи, які включають різні типи лож, спеціально обладнані місця для інвалідів, VIP-ложі та інші. Ось основні категорії місць, які зустрічаються частіше всього: партер, амфітеатр, ложі, бельетаж, балкон.

Партер (з фр., буквально – по землі) – частина театральної зали, призначена для глядачів, розташована нижче рівня сцени [5].

Амфітеатр (з грецької – з обох сторін; місця для глядачів) – місця для глядачів у сучасних театральних залах, розташовані відразу за партером на невеликому підвищенні, іноді – на другому поверсі [5].

Бельетаж (з фр. – гарний поверх) – перший ярус балкона глядацької зали над партером та амфітеатром [5].

Балкон (з фр., від німецької – балка) – у театрі – місця для глядачів, розташовані в різних ярусах зали навпроти сцени [5].

Ложа (з фр.) – група місць у глядацькій залі навколо партеру і на ярусах, відокремлена перегородками або бар’єрами [5].

У категорії ложі, як було зазначено вище, є декілька під категорій таких як: ложа бенуар, ложа бельетаж та ложа балкону.

Бенуар (з фр., буквально – ванна) – нижній ярус лож, розташований з обох боків партеру на одному рівні зі сценою [5].

На рис. 1.1 наведено приклад глядацького залу українського музично-драматичного театру ім. М.В. Гоголя, де зображені основні категорії, бенуар та декілька під категорій лож.

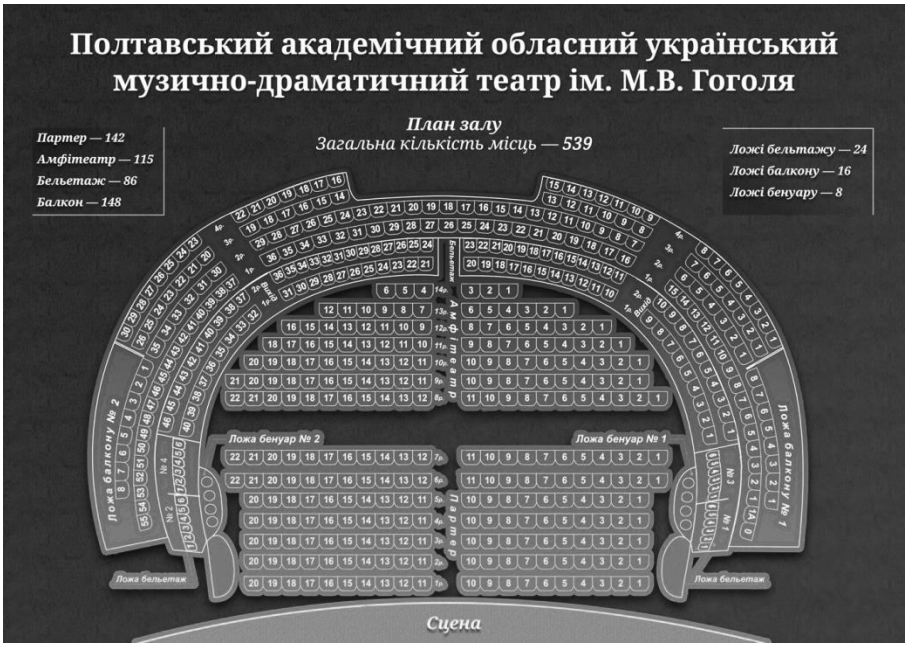


Рисунок 1.1 – Приклад плану глядацького залу

1.2 Аналіз наявних аналогів

На сьогоднішній день у світі існує безліч різних сервісів для бронювання квитків на будь-які розважальні події і театральні у тому числі. Для того щоб розробити власний програмний продукт важливим кроком буде зробити пошук та аналіз вже наявних аналогів для подальшого формування функціональних вимог. Тому розглянемо переваги та недоліки деяких вибраних з них, а саме: concert.ua, karabas.com, todaytix.com.

Сервіс Concert.ua - це популярний український сервіс для бронювання квитків на різноманітні події, включаючи театральні вистави, концерти, фестивалі та інші культурні заходи [7].

Він має в наявності веб-сайт та мобільний додаток. Далі описані основні переваги та недоліки цього сервісу.

Переваги: широкий вибір подій, зручний інтерфейс, доступність у різних містах України, після реєстрації з'являється можливість оплачувати онлайн, переглядати власні трансляції, квитки, сертифікати, та створити список бажань користувача.

Недоліки: можливість обмежень у виборі подій у деяких регіонах, можливість технічних проблем, дещо складний процес реєстрації у мобільному додатку порівняно з веб-сайтом цього ж сервісу, обмеженість однією мовою.

На рис. 1.2 зображено інтерфейс цього сервісу, вкладки афіша і квитки театрів.

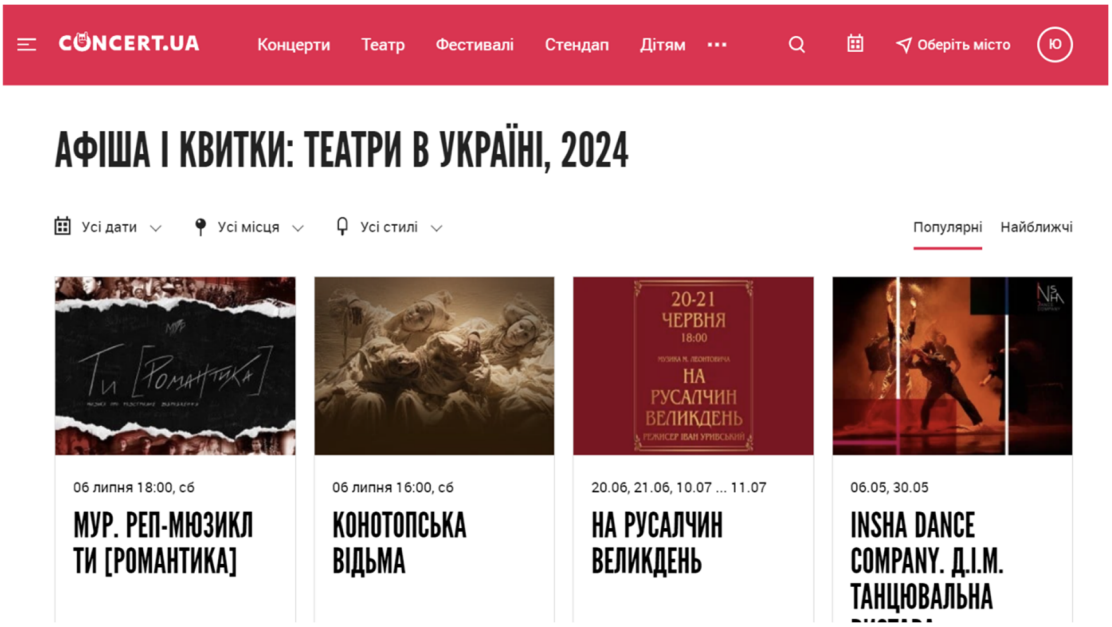


Рисунок 1.2 – Зображення інтерфейсу веб-сайту concert.ua

Сервіс Karabas.com - це український сервіс для бронювання квитків на культурні та розважальні події, включаючи театральні вистави, концерти, фестивалі, спортивні заходи та інші [8].

Має в наявності веб-сайт та мобільний додаток. Далі перераховані переваги та недоліки сервісу.

Основні переваги: широкий вибір подій, зручний пошук та фільтрація, доступність у великій кількості різних міст України, можливість вибрати мову веб-сайту та мобільного додатку, після реєстрації – здатність здійснювати оплату онлайн.

Недоліки: можливість перебоїв у роботі, конкуренція з іншими сервісами, складний процес реєстрації як на веб-сайті так і у додатку.

На рис. 1.3 зображено інтерфейс цього сервісу, а саме вкладки афіші подій.

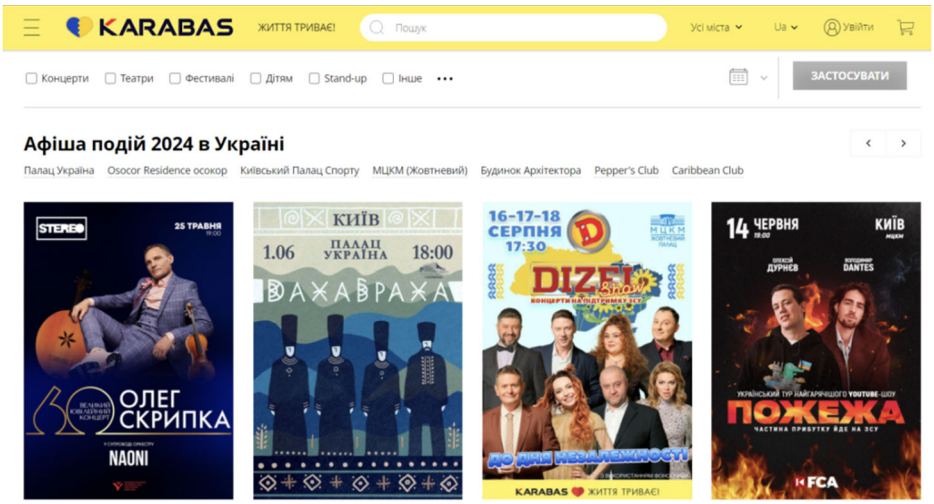


Рисунок 1.3 – Зображення інтерфейсу веб-сайту karabas.com

Сервіс todaytix - це міжнародний сервіс , спеціалізований на театральних виставах. Він пропонує користувачам вибір театральних подій у різних містах світу [9].

Цей сервіс має веб-сайт та мобільний додаток. Далі описані його основні переваги та недоліки.

Переваги: спеціалізація на театральних виставах, ексклюзивні пропозиції та знижки, зручний інтерфейс додатку, дуже легкий процес реєстрації.

Недоліки: обмежений вибір подій поза театральними виставами, доступність обмежена деякими регіонами.

На рис. 1.4 зображено інтерфейс цього сервісу.

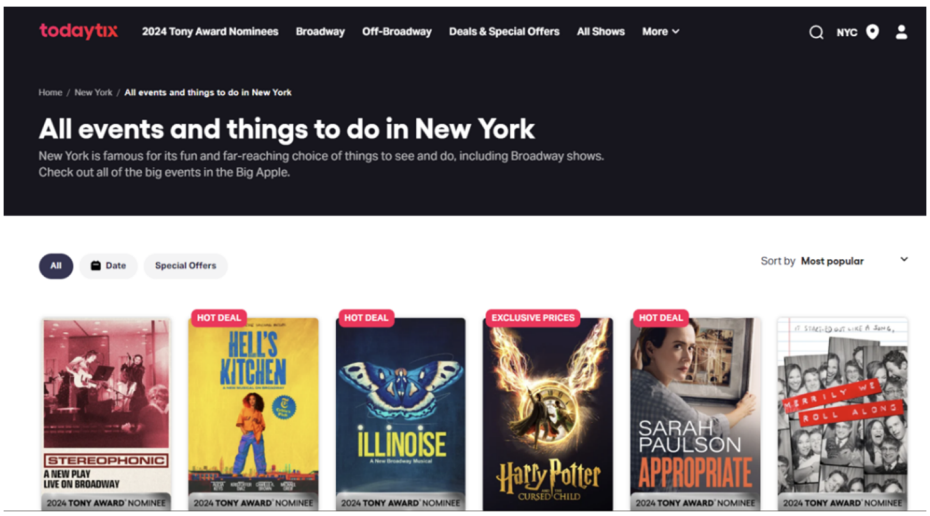


Рисунок 1.4 – Зображення інтерфейсу веб-сайту todaytix

Наступним кроком зробимо SWOT-аналіз кожного з розглянутих сервісів.

SWOT-аналіз – це аналітичний метод, який використовується для оцінювання сильних і слабких сторін, а також можливостей і загроз, пов’язаних із конкуренцією, що впливають на людину або бізнес. Абревіатура SWOT складається зі скорочень слів і означає наступне: S (Strengths) – сильні сторони, W (Weaknesses) – слабкі сторони, O (Opportunities) – можливості, T (Threats) – загрози [10].

Далі наведений SWOT-аналіз сервісів аналогів у табл. 1.1.

Таблиця 1.1 - SWOT-аналіз сервісів аналогів

Критерії аналізу	**Concert.ua**	**Karabas.com**	**todaytix**
Сильні сторони	Широкий вибір подій, зручний інтерфейс, наявність мобільного додатку	Широкий вибір подій, зручний пошук та фільтрація, наявність мобільного додатку, можливість вибрати мову веб-сайту та мобільного додатку, зручний інтерфейс користувача	Спеціалізація на театральних виставах, ексклюзивні пропозиції, наявність мобільного додатку
Слабкі сторони	Можливість технічних проблем, обмеження вибору подій у деяких регіонах, складний процес реєстрації у мобільному додатку, обмеженість однією мовою	Можливість перебоїв у роботі сайту чи додатку, конкуренція з іншими сервісами, складний процес реєстрації на веб-сайті та у мобільному додатку	Обмежений вибір подій поза театральними виставами, доступність обмежена деякими регіонами, обмеженість однією мовою
Можливості	Розширення асортименту подій, покращення технічних можливостей	Розширення партнерських взаємозв'язків з театрами, покращення сервісу підтримки користувачів, покращення процесу реєстрації	Розширення асортименту подій, розширення географії обслуговування
Загрози	Конкуренція з іншими сервісами, зміна вимог законодавства	Зміна вимог законодавства, негативні відгуки користувачів	Конкуренція з іншими сервісами, зміна вимог законодавства

Отримавши результати SWOT-аналізу можливо побачити що кожен з аналогів має спільні а також відмінні один від одного особливості кожного з категорій. Сервіси Concert.ua та Karabas.com розповсюджені лише на території

України, todaytix в свою чергу має глобальну доступність. Concert.ua та Karabas.com мають складнощі з реєстрацією користувача у мобільному додатку, другий сервіс має їх і на веб-сайті. У сервісі todaytix така проблема відсутня. Concert.ua та Karabas.com зосереджені на бронювання квитків на різноманітні події, у той час як todaytix лише на театральних виставах. Проте кожен з цих сервісів має досить зручний інтерфейс та широкий вибір вистав.

1.3 Визначення функціональних вимог

Після виконання дослідження предметної області інформаційної системи для бронювання квитків у театри, та пошуку та аналізу аналогів, їх функцій, наступним кроком є визначення функціональних вимог програмного продукту цієї кваліфікаційної роботи. Інформаційна система бронювання квитків у театри повинна мати функціонал який в першу чергу відповідає темі та задовольняє її вимогам, крім того потрібен зручний та легкий інтерфейс для її користувачів. Основний функціонал програмного продукту:

1. Початкове вікно:

– Можливість переходу користувачем до вікна реєстрації або авторизації.

2. Вікно реєстрації:

– Надання користувачу форми реєстрації, в якій може ввести свої дані (логін, email та пароль).

– Забезпечення перевірки на введення обов’язкових даних у всі поля.

– Виконання перевірки унікальності та правильності формату даних перед їх записом у базу.

– Створення облікового запису користувача на основі введених даних.

– Можливість навігації до початкового вікна та до особистого кабінету.

3. Вікно авторизації

– Надання користувачу форми авторизації, в якій може ввести свої дані (логін та пароль).

– Забезпечення перевірки на введення обов’язкових даних у всі поля.

– Перевірка наявності введених даних у базі.

– Можливість навігації до особистого кабінету користувача та до початкового вікна.

4. Особистий кабінет користувача:

– Навігація між вікнами за допомогою інтерфейсу.

– Допомога, яка містить пояснення функціоналу інтерфейсу.

– Відображення логіну користувача.

5. Персональні дані користувача:

– Перегляд даних введених при реєстрації та створенні облікового запису.

– Можливість навігації до особистого кабінету.

6. Вибір локації (міста):

– Виведення списку міст, в яких наявна можливість забронювати квиток на театральну подію.

– Перегляд та вибір міста в якому користувач бажає забронювати квиток.

– Перевірка на наявність вибраної локації.

– Можливість навігації до особистого кабінету та до вікна з афішею театрів.

– Допомога, яка містить пояснення функціоналу інтерфейсу.

7. Афіша театрів:

– Виведення театрів наявних у вибраному місті.

– Перегляд та вибір театру в якому користувач бажає забронювати квиток.

– Відображення афіші вибраного театру.

– Перегляд та вибір наявних вистав, які проводяться у вибраному театрі.

– Перевірка на наявність вибраного театру та вистави для бронювання квитка.

– Можливість навігації до вікон з вибором локації та з детальною інформацією про виставу.

– Допомога, яка містить пояснення функціоналу інтерфейсу.

8. Вікно з інформацією про виставу:

– Відображення детальної інформації про вибрану виставу.

– Можливість навігації до вікон з афішею театрів та бронювання квитка.

– Допомога, яка містить пояснення функціоналу інтерфейсу.

9. Бронювання квитка на виставу:

– Відображення назви вибраної вистави.

– Вибір критеріїв квитка (дати, часу, номеру місця, ряду та категорії).

– Виведення ціни на квиток відповідно до вибраних критеріїв.

– Перевірка наявності та бронювання квитка.

– Можливість навігації до вікон з детальною інформацією про виставу та забронюваними квитками користувача.

– Допомога, яка містить пояснення функціоналу інтерфейсу.

10. Вікно з забронюваними квитка користувача:

– Відображення даних про квитки заброньовані користувачем.

– Видалення бронювання на квиток.

– Можливість навігації до особистого кабінету.

– Допомога, яка містить пояснення функціоналу інтерфейсу.

Визначені функціональні вимоги програмного продукту допомагають створити зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

1.4 Вибір засобів розробки

Мова програмування − формалізована мова, призначена для описування алгоритмів розв’язування задач на ЕОМ. Мова програмування потрібна, щоб записувати алгоритм у термінах, зрозумілих компіляторові, який, своєю чергою, перекладе його на зрозумілу процесорові машинну мову [11].

Для розробки програмного продукту в цій кваліфікаційній роботі використовується мова програмування високого рівня - Python.

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних

компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах [12].

Python підтримує різноманітні підходи до програмування, включаючи об'єктно-орієнтований, функціональний, аспектно-орієнтований та процедурний.

Мова Python була обрана як засіб розробки через те що вона має: простий і зрозумілий синтаксис, велику кількість бібліотек та фреймворків, широкий спектр застосувань, і досить високу швидкість розробки. Крім того Python є найпопулярнішою мовою програмування за рейтингом TIOBE індексу на 2024 рік.

Індекс TIOBE Programming Community - це показник популярності мов програмування. Індекс оновлюється раз на місяць. Рейтинг базується на кількості кваліфікованих інженерів у всьому світі, курсів та сторонніх постачальників [13].

Цей рейтинг наведено на рис 1.5.

Apr 2024	Apr 2023	Change	Programming Language	Ratings	Change
1	1		Python	16.41%	+1.90%
2	2		C	10.21%	-4.20%
3	4	↑	C++	9.76%	-3.20%
4	3	↓	Java	8.94%	-4.29%
5	5		C#	6.77%	-1.44%
6	7	↑	JavaScript	2.89%	+0.79%
7	10	↑	Go	1.85%	+0.57%
8	6	↓	Visual Basic	1.70%	-2.70%
9	8	↓	SQL	1.61%	-0.06%

Рисунок 1.5 – Рейтинг мов програмування на 2024 рік

База даних (БД) - це організована колекція даних, до якої можна легко отримати доступ. Для управління базами даних використовують системи управління базами даних (СУБД). Є два поширені типи баз даних: нереляційні та реляційні [14].

Для створення бази даних і роботи з нею були вибрані мова SQL, та СУБД - MySQL.

Мова структурованих запитів (скор. “SQL” від англ. “Structured Query Language”) - це стандартна мова запитів, яка використовується для роботи з реляційними базами даних. SQL використовується для: створення баз даних, створення таблиць в базі даних, читання даних з таблиць, вставки даних в таблиці, оновлення даних в таблиці, видалення даних з таблиці, видалення таблиць бази даних, видалення баз даних, та інших операцій [14].

MySQL є реляційною СУБД і зберігає дані у табличному форматі.

SQL була обрана засобом розробки БД, тому що: є стандартизованою мовою для управління реляційними БД, є ефективною, простою у використанні, а також має широкі можливості. Крім того вона є однією з найпопулярніших мов за рейтингом TIOBE індексу на 2024 рік – займає 9 місце.

СУБД MySQL була обрана так як вона: є відкритою та безкоштовною, має високу продуктивність, має широкі можливості, а також велику кількість документації. Крім того вона є однією з найпопулярніших СУБД за рейтингом DB-Engines на 2024 рік, так як займає 2 місце. Цей рейтинг наведено на рис. 1.6.

Rank			DBMS	Database Model	Score		
May 2024	Apr 2024	May 2023			May 2024	Apr 2024	May 2023
1.	1.	1.	Oracle	Relational, Multi-model	1236.29	+2.02	+3.66
2.	2.	2.	MySQL	Relational, Multi-model	1083.74	-3.99	-88.72
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	824.29	-5.50	-95.80
4.	4.	4.	PostgreSQL	Relational, Multi-model	645.54	+0.49	+27.64
5.	5.	5.	MongoDB	Document, Multi-model	421.65	-2.31	-14.96
6.	6.	6.	Redis	Key-value, Multi-model	157.80	+1.36	-10.33
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	135.35	+0.57	-6.28
8.	8.	↓ 7.	IBM Db2	Relational, Multi-model	128.46	+0.97	-14.56
9.	9.	↑ 11.	Snowflake	Relational	121.33	-1.87	+9.61
10.	10.	↓ 9.	SQLite	Relational	114.32	-1.69	-19.54

Рисунок 1.6 – Рейтинг СУБД

Для створення програмного продукту були обрані 2 бібліотеки Python: wxPython та MySQL Connector/Python.

wxPython - це крос-платформний інструментарій графічного інтерфейсу для мови програмування Python. Він дозволяє програмістам Python легко і просто створювати програми з надійним, функціональним графічним інтерфейсом користувача. Він реалізований у вигляді набору модулів розширення Python, які обгортають компоненти графічного інтерфейсу популярної крос-платформної бібліотеки wxWidgets, написаної на C++ [16].

Наразі підтримуваними платформами є Microsoft Windows, Mac OS X та macOS, а також Linux або інші unix-подібні системи з бібліотеками GTK2 або GTK3 [16].

Бібліотека wxPython необхідна для розробки графічного інтерфейсу користувача для інформаційної системи бронювання квитків у театри. Вона була обрана, так як в першу чергу не залежить від платформи, як зазначалося вище вона може взаємодіяти з різними операційними системами. Крім того бібліотека має широкий набір елементів інтерфейсу, різних стилів і тем оформлення. Також вона є простою у використанні і має документацію.

MySQL Connector/Python дозволяє програмам на Python отримувати доступ до баз даних MySQL за допомогою API, що відповідає специфікації Python Database API Specification v2.0 [17].

MySQL Connector/Python включає підтримку:

– Майже всіх функцій, що надаються сервером MySQL версії 8.0 і вище.

– X DevAPI.

– Перетворення значень параметрів між типами даних Python і MySQL, наприклад, Python datetime і MySQL DATETIME.

– Всіх розширень MySQL до стандартного синтаксису SQL.

– Стиснення протоколу, що дозволяє стискати потік даних між клієнтом і сервером.

– Підключення за допомогою TCP/IP-сокетів, а в Unix - за допомогою Unix-сокетів.

– Безпечних TCP/IP-з'єднань за допомогою SSL.

– Самостійний драйвер. Connector/Python не потребує клієнтської бібліотеки MySQL або будь-яких модулів Python за межами стандартної бібліотеки [17].

MySQL Connector/Python є необхідним для підключення до програмного продукту бази даних і передачі необхідної інформації від неї до створеного графічного інтерфейсу користувача.

1.5 Висновки до розділу 1

Під час дослідження предметної області кваліфікаційної роботи було розглянуто поняття інформаційної системи, її різновиди та виявлено до якого типу належить розроблювана. Також була досліджена театральна сфера, та детально розглянуті фактори, які впливають на встановлення ціни на квитки до вистав.

У процесі пошуку наявних аналогів було знайдено численні сервіси для бронювання квитків на театральні події. Після проведення SWOT-аналізу була здійснена оцінка їх переваг та недоліків, а також їх можливостей для розвитку та загрози для них, що є важливим для визначення функціоналу створюваного продукту, адже це зробить його більш конкурентоспроможним і корисним для користувачів.

У функціональних вимогах було визначено основні функції, які має містити програмний продукт, для відповідності темі роботи та її основним вимогам.

Під час вибору засобів розробки були досліджені та проаналізовані можливі інструменти, та їх функціонал. В результаті чого обрані ті, які зможуть найбільш ефективно розробити необхідний функціонал програмного продукту.

За допомогою цих дослідження стає можливим створити інформаційну систему, що задовольнить користувачів.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Архітектура програмного застосунку

Після проведення аналізу та вибору засобів розробки інформаційної системи бронювання квитків у театри, наступним етапом є створення чіткої та логічної структури програмного продукту. Для досягнення цього необхідно розглянути компоненти архітектури розроблюваного застосунку.

Система включає наступні складові:

1. Клієнтська частина, яка відправляє запити на сервер для обробки даних. До клієнтської частини належить графічний інтерфейс користувача (GUI), який розроблюється за допомогою класів та бібліотеки мови програмування Python – wxPython. Цей інтерфейс включає:

– Вікна – є контейнерами для інших елементів інтерфейсу, а також складаються з фреймів (Frame) та вікон повідомлень (MessageBox).

– Текстові елементи (StaticText) – відображають статичний текст, що знаходиться на вікнах (фреймах).

– Елементи управління – дозволяють користувачу взаємодіяти з програмою. В додатку наявні наступні елементи: кнопки(Button), поля введення (TextCtrl), списки, що випадають (Choice).

– Меню – містять команди або опції, які користувач може вибрати, для виконання певних дій. За допомогою опції меню “Допомога” у програмному застосунку є можливість відкрити вікна, що містять пояснення щодо функціоналу. У додатку є 5 меню з такими вікнами.

– Списки – елементи, які відображають дані у вигляді списків. В застосунку є наступні списки: з пунктами (ListBox), та у вигляді таблиці (ListCtrl).

– Зображення – елементи, які відображають статичні зображення, і знаходяться на вікнах (фреймах)

2. Серверна частина (сервер баз даних). Вона відповідає за зв’язок між користувачем, через GUI, та базою даних. Сервер приймає запит з клієнтської

частини, обробляє та відправляє результат. Вона розроблюється за допомогою функцій та MySQL Connector/Python, і включає наступні компоненти:

– Підключення до бази даних.

– Виконання запитів – SQL – запити з серверної частини до бази даних. У додатку наявні запити на вибірку та оновлення даних.

– Обробка результатів запитів.

– Обробка винятків і помилок.

3. База даних MySQL до якої відбувається підключення за допомогою серверної частини.

Для кращого розуміння структури програмного застосунку також необхідно виділити в клієнтській частині компоненти вікон. У додатку наявні наступні елементи:

– Початкове вікно (MyFrame1) – містить назву додатку та 2 опції на вибір: реєстрація або авторизація користувача.

– Вікно реєстрації (MyFrame2) – містить форму для створення облікового запису користувачу, що надає можливість входу до особистого кабінету.

– Вікно авторизації (MyFrame3) – має форму для входу до вже створеного облікового запису.

– Особистий кабінет (MyFrame4) – вікно, яке містить логін користувача, а також декілька опцій на вибір, які включають: здійснення броні квитка, перегляд вже отриманих, перевірка особистих даних, вихід з особистого кабінету.

– Вікно персональних даних (MyFrame5) – містить дані введені користувачем при реєстрації.

– Вікно локації (MyFrame7) – має список міст, серед яких користувач може обрати те, де він бажає бронювати квиток.

– Вікно афіші театрів (MyFrame8) – відображає 2 списки, перший з яких містить доступні театри в обраному місті, а після вибору – вистави у ньому.

– Вікно вистави (MyFrame9) – містить детальну інформацію про виставу

– Вікно бронювання (MyFrame10) – має форму для здійснення броні квитка на обрану виставу.

– Вікно заброньованих квитків (MyFrame11) – містить список з детальною інформацією про всі квитки користувача, на які він здійснив бронювання.

– Вікна допомоги (MyFrame12, MyFrame13, MyFrame14, MyFrame15, MyFrame16, MyFrame17) – містять текстові пояснення щодо функціоналу вікон в яких вони знаходяться.

– Вікна повідомлень (MessageBox) – відображення коротких повідомлень та попереджень користувачу про певні події у системі або про помилки.

Після ретельного огляду описаних вище компонентів додатку було виявлено, що створена архітектура відноситься до клієнт-серверної. Ця архітектура має два види: дворівневу та багаторівневу (зазвичай використовується термін трирівнева архітектура). Розроблювана інформаційна система використовує дворівневу клієнт-серверну архітектуру, оскільки обробка даних відбувається на одному сервері.

Дворівнева клієнт-серверна архітектура наведена на схемі на рис. 2.1.

Рисунок 2.1 – Схема дворівневої клієнт-серверної архітектури

2.2 Проектування бази даних

Визначивши компоненти створюваного програмного додатку та розглянувши більш детально клієнтську та серверну частини, наступним

важливим кроком буде огляд бази даних для бронювання квитків у театри. Для цього важливо виділити її сутності, які є необхідними для створення інформаційної системи.

Сутність (Entity) – реальний або уявний об'єкт, що має суттєве значення для розглянутої предметної області [20].

База даних “Театральні квитки” (theater_tickets) має 8 сутностей (таблиць), вони наступні: “Театр” (theaters), “Направлення театру” (genres), “Зв’язок театру та направлення” (c2c_theaters_genres), “Вистава” (performances), “Актор” (actors), “Виступ” (actions), “Користувач” (ticket_users), “Квиток” (tickets_programs).

Кожна з зазначених сутностей має власні атрибути, які необхідно ретельно дослідити. Саме тому наступною дією буде виявлення цих властивостей (атрибутів), їх опис (дані які вони зберігають, наявність ключа), а також типів даних, які зберігаються в них.

Атрибут сутності – це іменована характеристика, що є деякою властивістю сутності [20].

Атрибути, які належать сутності “Театр” (theaters) наведено у табл. 2.1.

Таблиця 2.1 – Опис сутності “Театр”

Атрибут	**Тип даних**	**Опис**
id_theater	int	Зберігає індекс театру. Первинний ключ
name_theater	varchar	Зберігає назву театру
name_town	varchar	Зберігає назву міста в якому знаходиться театр
name_address	varchar	Зберігає назву вулиці на якій знаходиться театр

Опис сутності “Направлення театру” (genres) зображено у табл. 2.2.

Таблиця 2.2 – Опис сутності “Направлення театру”

Атрибут	**Тип даних**	**Опис**
id_genre	int	Зберігає індекс направлення. Первинний ключ
name_genre	varchar	Зберігає назву направлення

Наступна сутність - “Зв’язок театру та направлення” (c2c_theaters_genres), пов’язує дві інші описані вище: “Театр” (theaters) та “Направлення театру” (genres). Дослідження її атрибутів наведено у табл. 2.3.

Таблиця 2.3 – Опис сутності "Зв'язок театру та направлення"

Атрибут	Тип даних	Опис
id_theater	int	Зберігає індекс театру. Зовнішній ключ
id_genre	int	Зберігає індекс направлення. Зовнішній ключ

Сутність "Вистава" (performances) складається з атрибутів розглянутих у табл. 2.4.

Таблиця 2.4 – Опис сутності "Вистава"

Атрибут	Тип даних	Опис
id_performance	int	Зберігає індекс вистави. Первинний ключ
name_performance	varchar	Зберігає назву вистави
start_date	date	Зберігає дату прем'єри вистав (дата з якої починається показ у театрах)

Дослідження атрибутів сутності "Актор" (actors) розглянуто у табл. 2.5.

Таблиця 2.5 – Опис сутності "Актор"

Атрибут	Тип даних	Опис
id_actor	int	Зберігає індекс актора. Первинний ключ
name_actor	varchar	Зберігає ім'я та прізвище актора

Наступна таблиця (сутність), "Виступ" (actions), так само як "Зв'язок театру та направлення" пов'язує дві інші сутності, а саме "Вистава" (performances) та "Актор" (actors). Опис цієї сутності наведено у табл. 2.6.

Таблиця 2.6 – Опис сутності "Виступ"

Стовбець(атрибут)	Тип даних	Опис
id_action	int	Зберігає індекс виступу. Первинний ключ
id_performance	int	Зберігає індекс постанови яка проводиться. Зовнішній ключ
id_actor	int	Зберігає індекс актора який виступає. Зовнішній ключ
data_time_action	datetime	Зберігається дата та час виступу
ticket_price	int	Зберігається ціна квитка на виступ

Далі відбувається дослідження сутності, в якій знаходиться інформація про зареєстрованих користувачів застосунку - "Користувач" (ticket_users). Її атрибути розглядаються у табл. 2.7.

Таблиця 2.7 – Опис сутності “Користувач”

Атрибут	Тип даних	Опис
id_user	int	Зберігає індекс користувача. Первинний ключ
name_user	varchar	Зберігає логін користувача
email_user	varchar	Зберігає email користувача
password_user	varchar	Зберігає пароль користувача

Останньою восьмою сутністю є “Квиток” (tickets_programs), вона містить інформацію про всі квитки на вистави, а також пов’язує такі сутності як: “Театр” (theaters), “Виступ” (actions) та “Користувач” (ticket_users). Її атрибути наведено у табл. 2.8.

Таблиця 2.8 – Опис сутності “Квиток”

Атрибут	Тип даних	Опис
id_ticket	int	Зберігає індекс квитка. Первинний ключ
id_theater	int	Зберігає індекс театру. Зовнішній ключ
id_action	int	Зберігає індекс вистави. Зовнішній ключ
id_user	int	Зберігає індекс користувача додатку. Зовнішній ключ
seats_number	int	Зберігає номер місця
rows_number	int	Зберігає номер ряду
seats_categories	varchar	Зберігає категорію місця
ticket_state	enum	Зберігає стан квитка, має два значення: ‘Броньований’, ‘Неброньований’

В описаній вище базі даних між усіма її сутностями (таблицями) наявний один тип зв’язку – один до багатьох.

Зв'язок один до багатьох означає, що кожному екземпляру одного об'єкта (A) може відповідати кілька екземплярів іншого об'єкта (B), а кожному екземпляру другого об'єкта (B) може відповідати тільки один екземпляр першого об'єкта (A) [21].

Цей зв'язок встановлюється за допомогою ключового поля у першій таблиці (сутності), та не ключового поля у другій. Зазвичай першу таблицю називають батьківською, а другу - дочірньою.

Прикладом зв'язку "один до багатьох" є відношення між двома сутностями: “Виступ” (actions) та “Актор” (actors), яке зображено на рис. 2.2.

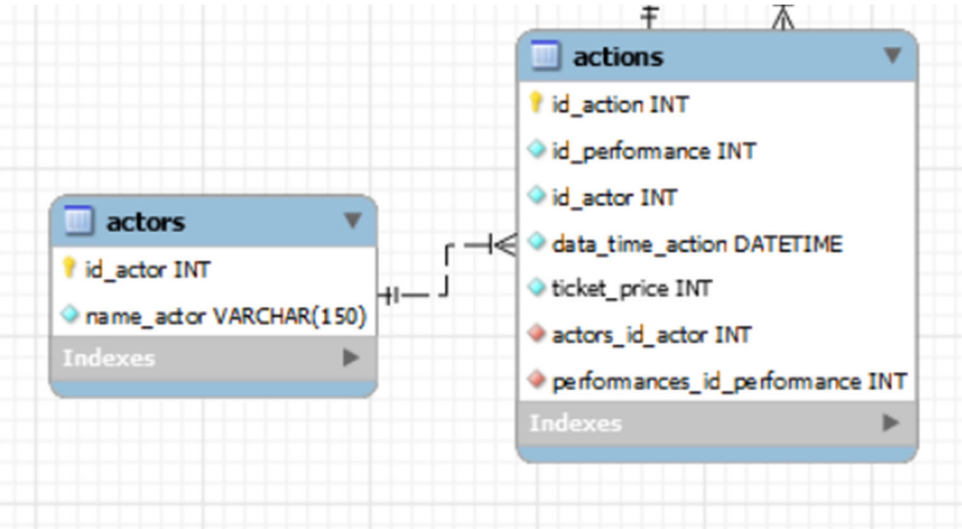


Рисунок 2.2 – Зв'язок між сутностями “Виступ” та “Актор”

Схема зв'язку "один до багатьох" між тими самими сутностями є наведеною на рис. 2.3.

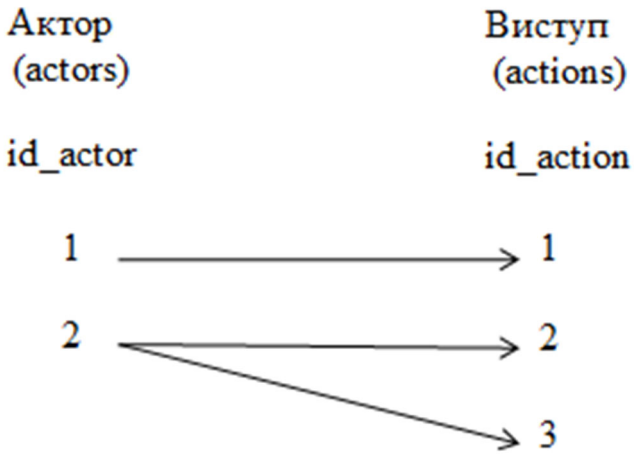


Рисунок 2.3 – Схема зв'язку між сутностями “Виступ” та “Актор”(1:N)

Зв'язок між цими двома сутностями належить до цього типу відношень так як один актор може приймати участь у багатьох виступах. І розглянутий тип зв'язку розповсюджується на усі сутності за цією ж схемою.

Так як вище були досліджені всі вісім сутностей, їх атрибути та зв'язки між ними, була розроблена ER модель, яка наведена на рис. 2.4.

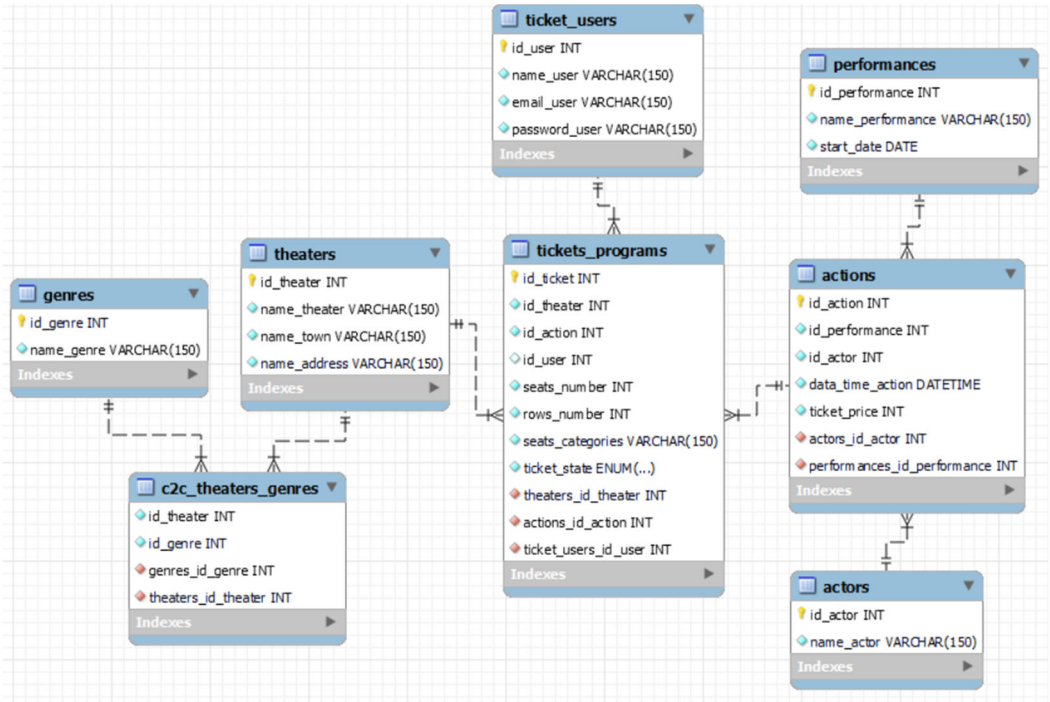


Рисунок 2.4 – ER модель БД "Театральні квитки

2.3 Моделювання програмної системи

Моделювання даних є важливою частиною проектування програмного продукту в першу чергу тому що це сприяє підвищенню якості та стабільності програми, а також допомагає в оптимізації продуктивності.

Модель даних – це формальний опис даних та їх відносин, який використовується для аналізу, проектування та розробки програмного забезпечення. Модель даних описує структуру даних, їх типи та зв'язки між ними. Модель даних може бути представлена у вигляді схеми бази даних або діаграми, яка показує зв'язки між різними таблицями [22].

У підрозділі 2.2 - проектування бази даних, вже була створена одна з моделей даних, а саме концептуальна – ER модель. Але її недостатньо для повноцінного проектування програмного додатку, тому наступним кроком важливо описати дані за допомогою інших моделей даних та UML-діаграм.

Діаграма DFD (Data Flow Diagram) - це діаграми, на яких відображаються потоки даних, процеси перетворення вхідних потоків на вихідні, сховища інформації, джерела і споживачі інформації, зовнішні щодо системи [20].

Перша створювана діаграма DFD є інструментом моделювання системи зовнішньою сутністю якої є користувач. Елементи цієї діаграми описані нижче.

Зовнішні сутності:

1. Користувач – це сутність, яка є джерелом та отримувачем інформації, а також тим хто виконує процеси.

2. Система – це проміжна сутність, яка отримує запити з процесів та надсилає їх до сховища даних, після чого відправляє відповідь сутності.

Процеси:

1. Реєстрація користувача:

– вхідні дані: логін, email, пароль;

– вихідні дані: повідомлення про успішне або невдале реєстрування.

2. Авторизація користувача:

– вхідні дані: логін, пароль;

– вихідні дані: повідомлення про успішну або невдалу авторизацію.

3. Перегляд афіші театрів:

– вхідні дані: назва міста;

– вихідні дані: список театрів та вистав які вони представляють.

4. Бронювання квитків:

– вхідні дані: ID театру, ID вистави, дата та час виступу, номер місця, номер ряду, категорія місця.

– вихідні дані: повідомлення про успішне або невдале бронювання квитка.

5. Видалення броні на квиток:

– вхідні дані: ID квитка;

– вихідні дані: повідомлення про видалення броні.

Сховище даних:

База даних "Театральні квитки" – зберігає інформацію про усі театри, їх направлення, вистави, виступи, акторів, користувачі додатку, та квитки.

Потоки даних:

– Користувач – Реєстрація користувача: зовнішня сутність передає дані необхідні для створення облікового запису.

– Користувач – Авторизація користувача: зовнішня сутність передає дані для входу до створеного облікового запису.

– Користувач – Перегляд афіші театрів: користувач вибирає назву міста, за якою виводяться список театрів та вистав, які проводять в них.

– Користувач – Бронювання квитків: користувач вибирає дату та час виступу, а також номер місця, його ряду, а також його категорії.

– Користувач – Видалення броні на квиток: користувач вибирає ID квитка для скасування броні.

– Процеси – Система: передача запитів від кожного процесу проміжній сутності для подальшої обробки та відправки в базу даних.

– Система – База даних “Театральні квитки”(від процесу Реєстрація користувача): запит на перевірку наявності даних. У випадку відсутності – збереження введеної інформації у базу даних.

– Система - База даних “Театральні квитки”(від процесу Авторизація користувача): запит на перевірку наявності введеної інформації.

– Система – База даних “Театральні квитки”(від процесу Перегляд афіші театрів): запит на передачу до БД назви міста для пошуку відповідних даних про театри та вистави, після чого відправка їх системі та користувачу.

– Система – База даних “Театральні квитки”(від процесу Бронювання квитків): запит на передачу до БД вхідних даних, для пошуку ID квитка та оновлення його стану на заброньований.

– Система – База даних “Театральні квитки”(від процесу Видалення броні на квиток): запит на передачу ID квитка до БД для оновлення його стану на небронь ований.

– Сховище даних – Система: повідомлення про успішне або невдале не виконання операцій для подальшої передачі його користувачу.

– Система – Користувач – повідомлення про результати виконання операцій.

Діаграма DFD для зовнішньої сутності – користувача, наведена на рис. 2.5.

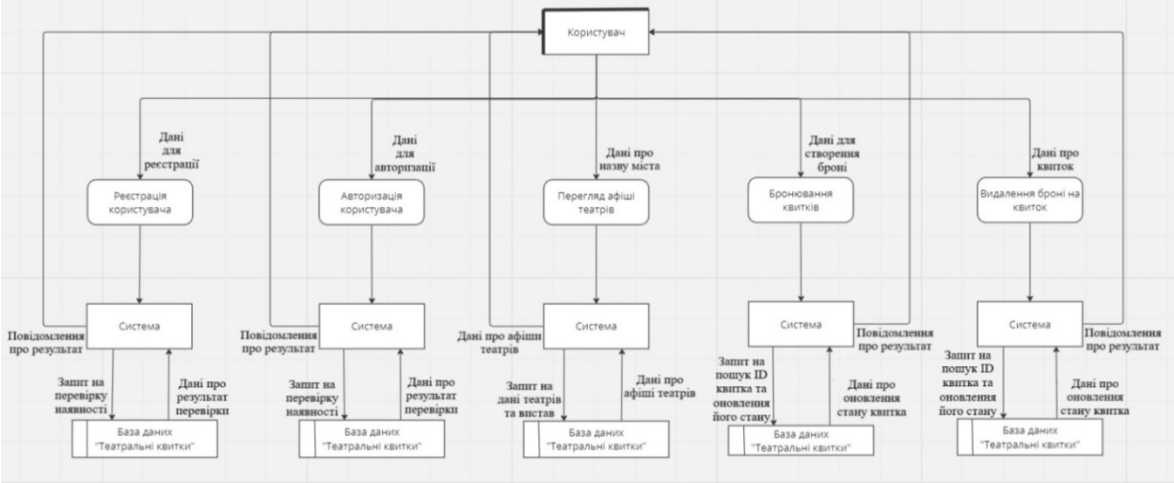


Рисунок 2.5 – Діаграма DFD для зовнішньої сутності – користувача

Друга створювана діаграма DFD має зовнішню сутність – адміністратор бази даних. Крім того ця модель даних також має те ж саме сховище даних. Елементи цієї діаграми описані нижче.

Зовнішні сутності:

1. Адміністратор баз даних – це сутність, яка виконує усі описані нижче процеси.

2. Система – це проміжна сутність, яка отримує запити з процесів та надсилає їх до сховища даних, після чого відправляє відповідь сутності.

Процеси:

1. Додавання квитків:

– вхідні дані: ID квитка, ID театру, ID виступу, номер місця, номер ряду, категорія місця;

– вихідні дані: повідомлення про успішне або невдале додавання.

2. Видалення квитків:

– вхідні дані: ID квитка;

– вихідні дані: повідомлення про успішне або невдале видалення.

3. Оновлення даних квитків:

– вхідні дані: ID квитка, а також нові дані про квиток;

– вихідні дані: повідомлення про успішне або невдале оновлення.

Сховище даних:

База даних "Театральні квитки" – зберігає інформацію про усі театри, їх направлення, вистави, виступи, акторів, користувачі додатку, та квитки.

Потоки даних:

– Адміністратор баз даних – Додавання квитків: зовнішня сутність передає дані про новий квиток.

– Адміністратор баз даних – Видалення квитків: адміністратор передає ID квитка для видалення

– Адміністратор баз даних – Оновлення даних квитків: адміністратор передає ID квитка та його нові дані для оновлення його інформації.

– Процеси – Система: передача запитів від кожного процесу проміжній сутності для подальшої обробки та відправки в базу даних .

– Система – База даних “Театральні квитки”: запити на збереження, видалення та оновлення інформації про квитки.

– Система - Адміністратор баз даних: повідомлення про успішне або невдале виконання операцій.

Діаграма DFD для зовнішньої сутності – адміністратора баз даних, наведена на рис. 2.6.

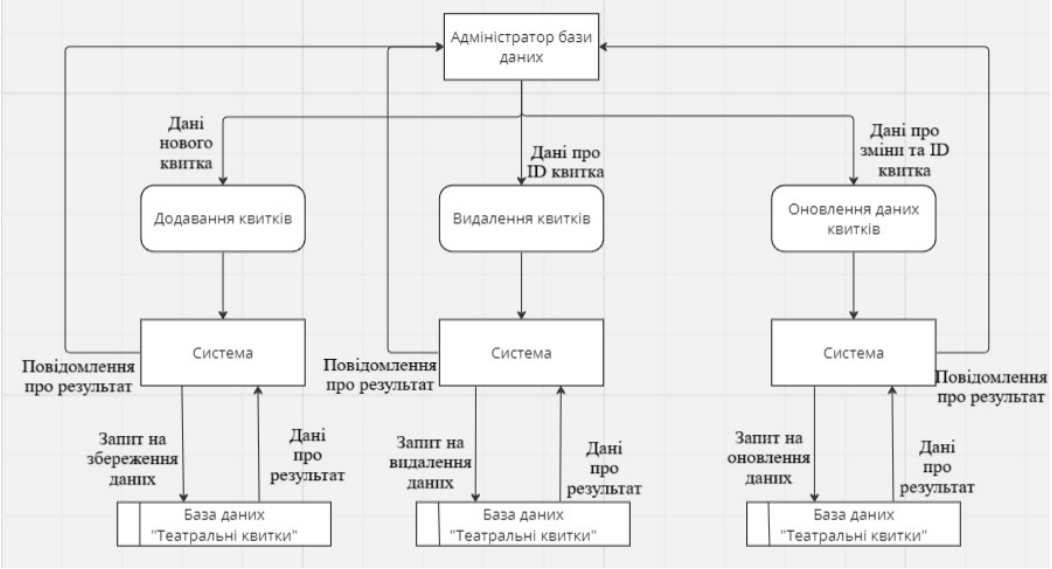


Рисунок 2.6 – Діаграма DFD для адміністратора баз даних

Наступною у цьому підрозділі створюється і розглядається UML - діаграма варіантів використання.

Діаграма варіантів використання (або діаграма прецедентів, Use case diagram) є найбільш загальним поданням функціональних вимог до системи. Сценарій детально документує процес взаємодії дійової особи з системою, що реалізується, в рамках варіанта використання [20].

Елементами створюваної діаграми є варіанти використання та актори, які з'єднують між собою зв'язки: асоціації, розширення та включення. Складові діаграми використання описані нижче.

Актори:

1. Користувач – це людина, яка використовує систему для бронювання квитків у театри.
2. Адміністратор баз даних – це людина яка займається обслуговування бази даних "Театральні квитки".
3. Система – це система для бронювання квитків у театри.

Варіанти використання "Користувач":

1. Реєстрація у системі:
– Заповнення форми реєстрації
– Валідація введених даних (системою)
– Створення облікових записів
– Додавання персональних даних до БД (системою)
– Перевірка наявності введених даних у БД (системою)
2. Авторизація у системі:
– Заповнення форми авторизації
– Валідація введених даних (системою)
– Створення облікових записів
– Перевірка наявності введених даних у БД (системою)
3. Перегляд персональних даних користувача
4. Перегляд локацій театрів:
– Вибір локації театру

5. Перегляд списку театрів:

– Вибір локації театру

– Вибір театру

6. Перегляд афіші театрів:

– Вибір театру

– Вибір вистави

7. Перегляд деталей вистави:

– Вибір вистави

8. Бронювання квитків:

– Заповнення форми бронювання

– Вибір локації театру

– Вибір театру

– Вибір вистави

– Зміна стану квитка у базі даних (системою)

– Визначення ціни квитка за вказаними користувачем даними (системою)

– Перевірка на можливість здійснення броні

9. Перегляд заброньованих квитків:

10. Скасування броні:

– Перегляд заброньованих квитків (системою)

– Перевірка на можливість скасування броні (системою)

– Зміна стану квитка у базі даних (системою)

Варіанти використання “Адміністратор бази даних”:

– Додавання даних до БД

– Видалення даних з БД

– Оновлення даних у БД

– Управління даними користувачів:

– Створення облікових записів.

– Видалення облікових записів.

– Перегляд персональних даних користувача.

Варіанти використання “Система” (не перераховані вище):

1. Перевірка правильності заповнення форм
2. З’єднання з базою даних:

– Додавання даних до БД

– Видалення даних з БД

– Оновлення даних у БД

Діаграма варіантів використання наведена на рис. 2.7.

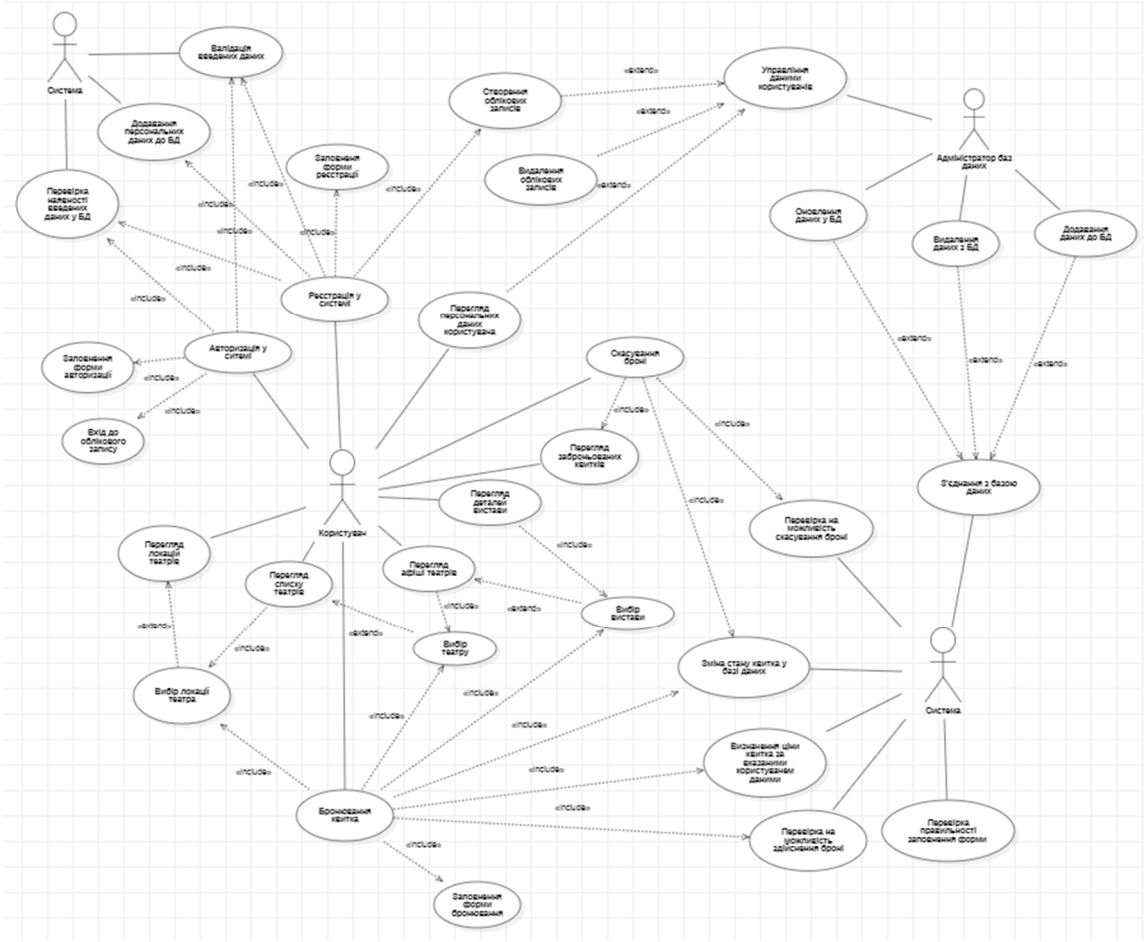


Рисунок 2.7 – UML - діаграма - Use Case

Далі, останньою у цьому підрозділі створюється та описується діаграма класів.

Діаграма класів (class diagram) використовуються для моделювання статичного подання системи з точки зору проектування в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати різні

взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їх внутрішню структуру і типи відносин [20].

Класи містять ім’я та атрибути, які описані нижче:

1. Theater – клас який описує театри:
– id_theater – містить унікальний ідентифікатор театру;
– name_theater – містить назву театру;
– name_town – містить назву міста в якому знаходиться театр;
– name_address – містить адресу театру.
2. Genre – клас який описує направлення(жанри) театрів
– id_genre – містить унікальний ідентифікатор направлення;
– name_genre – містить назву направлення.
3. Connection – клас, який описує зв’язок між театром та направленням:
– id_theater – містить унікальний ідентифікатор театру;
– id_genre – містить унікальний ідентифікатор направлення.
4. Performance – клас, який описує виставу:
– id_performance – містить унікальний ідентифікатор вистави;
– name_performance – містить назву вистави;
– start_date – містить дату початку показу вистави у театрах.
5. Actor – клас який описує актора:
– id_actor – містить унікальний ідентифікатор актору;
– name_actor – містить ім’я та прізвище актора.
6. Action – клас який описує виступ вистави у театрі:
– id_action – містить унікальний ідентифікатор виступу;
– id_performance – містить унікальний ідентифікатор вистави;
– id_actor – містить унікальний ідентифікатор актора який виступає;
– data_time_action – містить дату та час виступу (показу вистави);
– ticket_price – містить ціну квитка на виступ.
7. User – клас який описує користувача:
– id_user – містить унікальний ідентифікатор користувача;
– name_user – містить логін користувача;

– email_user – містить email користувача;

– password_user – містить пароль користувача.

8. Ticket – клас який описує квиток на виступ у театр:

– id_ticket – містить унікальний ідентифікатор квитка;

– id_theater – містить унікальний ідентифікатор театру;

– id_action – містить унікальний ідентифікатор виступу;

– id_user – містить унікальний ідентифікатор користувача;

– seats_number – містить номер місця;

– rows_number – містить номер ряду;

– seats_categories – містить категорію місця;

– ticket_state – містить стан квитка (броньований, неброньований).

Класи описані вище мають наступні методи:

– Theater: getIdTheater, getNameTheater, getNameTown.

– Genre: getIdGenre, getNameGenre.

– Connection: немає методів.

– Performance: getIdPerformance, getNamePerformance, getStartDate.

– Actor: getIdActor, getNameActor.

– Action: getIdAction, getIdPerformance, getIdActor, getDataTimeAction, getTicketPrice.

– User: addUser, getIdUser, getNameUser, getUserInfo.

– Ticket: getIdTicket, getIdTheater, getIdAction, addUser, getSeatsNumber, getRowsNumber, getSeatsCategories, getTicketState, updateTicketState.

Більшість перерахованих вище методів спрямовані на отримання інформації про класи. Але також є методи для: створення нового користувача - addUser, оновлення інформації про стан квитку - updateTicketState, та отримання всієї інформації для одного з класів – getUserInfo.

Крім того діаграма має одне й те саме відношення між усіма класами – агрегація, яка передає зв’язок один до багатьох. Також агрегація вказує на те, що один клас є частиною іншого класу.

На рис. 2.8 наведено створену діаграму класів.

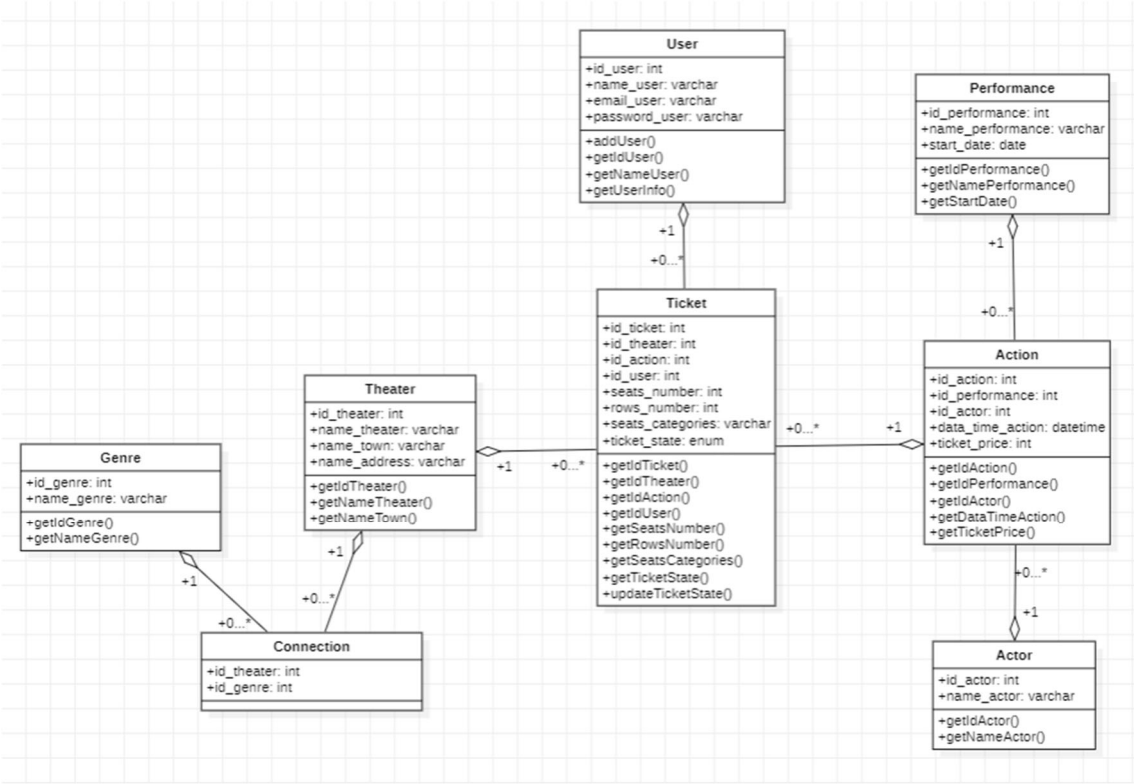


Рисунок 2.8 – UML - діаграма - Class

2.4 Висновки до розділу 2

Під час проектування програмного продукту була створена його архітектура. Оскільки для розроблюваного застосунку обрана дворівнева клієнт-серверна архітектура, було проведено дослідження її компонентів. Для цього були більш детально розглянуті бібліотека wxPython та MySQL Connector/Python.

Під час проектування бази даних, яка також є компонентом архітектури, було виділено сутності, їх атрибути та зв’язки між ним. Також була побудована ER модель яка ілюструє описане дослідження.

Для моделювання були створені: модель даних DFD та UML - діаграми варіантів використання та класів. Розробка цих моделей була зосереджена на визначенні функціональних вимог, структури, а також потоків даних, що сприяє підвищенню якості та оптимізації продуктивності створюваної системи.

За допомогою цих досліджень та моделей був спроектований програмний продукт, який у подальшому можна реалізувати.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка серверної частини

В попередніх розділах було проведено дослідження, вибрані необхідні інструменти розробки, спроектовано структуру та здійснено моделювання програмного продукту, а також визначено його функціонал. Це надає можливість розробити всі компоненти створюваного застосунку. Першим етапом буде створення серверної частини.

Як було зазначено у розділі 2, сервер приймає запит з клієнтської частини, обробляє та відправляє результат до бази даних, після чого, отримавши відповідь надсилає її до інтерфейсу користувача. Саме тому є необхідним в першу чергу здійснити підключення до бази даних.

Для того щоб створити з’єднання з сервером MySQL використовується конструктор connect(), який належить MySQL Connector/Python. У ньому вказується: назва хосту, ім’я користувача, пароль для входу до MySQL, та назва бази даних до якої потрібно підключитися. Щоб мати можливість оброблювати та перехоплювати помилки які можуть виникнути при з’єднанні застосовується конструкція try … except (errors). Код підключення до бази даних цього додатку наведено на рис. 3.1.

```
import wx
import mysql.connector



#Підключення до бази даних
24 usages
def connect_to_mysql():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="11052002baza*+",
            database="theater_tickets"
        )
        return conn
    except mysql.connector.Error as e:
        print("Помилка підключення до MySQL:", e)
        return None
```

Рис. 3.1 – Підключення до бази даних “Театральні квитки”

Наступним кроком після з’єднання є розробка запитів до цієї бази даних, які мають реалізувати функціонал для інтерфейсу користувача. Всього у додатку наявні 25 функцій, які за допомогою MySQL Connector/Python опрацьовують ці запити. Необхідно зробити їх огляд та додатково виконати операції безпосередньо у MySQL для перевірки успішності результатів.

Кожна з функцій має структуру описану далі. Спочатку здійснюється перевірка з’єднання з базою даних, після чого відбувається створення курсору для виконання запитів до неї.

Клас MySQLCursor створює об'єкти, які можуть виконувати такі операції, як оператори SQL. Об'єкти курсору взаємодіють з сервером MySQL за допомогою об'єкта MySQLConnection. Щоб створити курсор, потрібно використати метод cursor() [23].

Після цього створюється запит до бази даних, який присвоюється змінній для зручності, та виконується за допомогою методу execute(), який належить класу курсора.

Вказуючи змінні, потрібно використовувати стиль параметрів %s або %(name)s (тобто, використовуючи стиль format або pyformat). execute() повертає ітератор, якщо multi дорівнює True [24].

Наступними діями після відправки запиту є отримання результату, та закриття курсору. Для цього можна скористатися методами fetchone()/fetchall() та close() відповідно.

Метод fetchone() отримує наступний рядок набору результатів запиту і повертає єдину послідовність або None, якщо більше немає рядків [25].

Метод fetchall() витягує всі (або всі, що залишилися) рядки з набору результатів запиту і повертає список кортежів. Якщо рядків більше немає, повертається порожній список [26].

Метод close() закриває курсор, обнуляє всі результати і гарантує, що об'єкт курсору не матиме посилань на початковий об'єкт з'єднання [27].

Останніми діями є закриття з’єднання з базою даних та повернення результату який отримали через метод fetchone(). На рис. 3.2 наведено описану вище структуру на прикладі функції.

Наступним етапом є огляд кожної функції окремо для визначення їх призначення, який здійснено нижче.

– Функція check_duplicate_user є потрібною для перевірки наявності користувача за вказаною ним інформацією при реєстрації - логін, email та пароль. Вона є необхідною щоб уникнути ситуації коли наявні декілька облікових записів з однаковими даними. Крім того відсутність цієї функції може створити ще одну помилку, так як у базі даних, в таблиці “Користувач”, дані можуть бути лише унікальними, тобто неповторюваними. Код описаної функції перевірки зображений на рис. 3.2.

```
def check_duplicate_user(conn, name_user, email_user, password_user):
    if not conn:
        print("Немає з'єднання з базою даних")
        return False

    cursor = conn.cursor() # Створення курсору для виконання запитів до бази даних

    query = "SELECT * FROM ticket_users WHERE name_user = %s OR email_user = %s OR password_user = %s"
    cursor.execute(query, (name_user, email_user, password_user))
    duplicate_user = cursor.fetchone() # Отримання результату запиту

    cursor.close() # Закриття курсору

    if duplicate_user: # Перевірка наявності знайденого користувача
        return True
    else:
        return False
```

Рис. 3.2 – Код функції check_duplicate_user вікна “Реєстрація”

Результат запиту функції check_duplicate_user до бази даних у самій у MySQL наведено на рис. 3.3.

```
1 •  SELECT * FROM ticket_users WHERE name_user = 'Alina97'
2    OR email_user = 'alina19997@gmail.com' OR password_user = 'Sunday1234*'
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id_user	name_user	email_user	password_user
1	Alina97	alina19997@gmail.com	Sunday1234*
NULL	NULL	NULL	NULL

Рис. 3.3 – Запит функції check_duplicate_user

– Функція add_user додає користувача в базу даних. Ця подія відбувається після заповнення форми реєстрації та після перевірки наявності вказаної інформації у базі, і лише у випадку її відсутності. Якщо минула описана функція визначає що користувач з такими даними вже існує то і його додавання неможливе. Окрім команд зазначених вище в загальному описі структури, наявний метод commit(), який необхідний для збереження будь-яких змін безпосередньо у базі даних, крім того відсутні методи витягування інформації так як тут це не потрібно. Код описаної функції зображено на рис. 3.4.

```
def add_user(conn, name_user, email_user, password_user):
    if not conn:
        print("Немає з'єднання з базою даних")
        return False

    cursor = conn.cursor()

    # Додавання користувача в базу даних
    query = "INSERT INTO ticket_users (name_user, email_user, password_user) VALUES (%s, %s, %s)"
    cursor.execute(query, (name_user, email_user, password_user))

    conn.commit() # Фіксація, тобто збереження зміни в базі даних
    cursor.close()

    return True
```

Рис. 3.4 – Код функції add_user вікна “Реєстрація”

– Функція check_credentials здійснює перевірку наявності користувача з вказаними логіном та паролем для входу до особистого кабінету. Функція перевірки даних реєстрації та авторизації є майже однаковими, і відрізняються лише запитами, так як у цій немає необхідності перевіряти email.

– get_user_info – це функція, яка необхідна для вікна “Персональні дані”, вона отримує інформацію про користувача, яку він надав при реєстрації у додатку (email та пароль) за логіном. Крім того у цій функції так само як і у минулих наявний метод fetchone(), для отримання наступного рядку з результатів запиту і повернення єдиної послідовності, так як це найбільш доречно в цьому випадку.

– Функція get_unique_towns отримує з бази даних список неповторюваних назв міст. Використовується у вікні “Локація”, для того щоб додати ці дані до списку звідки користувач може вибрати те де він бажає відвідати виставу в театрі. У цій функції застосовується метод fetchall(), який повертає список кортежів.

– Функція get_theaters_in_town отримує з бази даних список назв театрів за містом вибраним у вікні “Локація”. Так само як і минула функція використовує метод fetchall() для отримання даних.

– get_theater_address – це функція необхідна для отримання адреси театру в якому проводиться вибрана вистава. Вона використовується так само як і попередня у фреймі “Афіша театрів”, для встановлення адреси обраного театру, але сам текст буде відображатися на наступному вікні, у деталях вистави на яку можна забронювати квиток. Так як дані які потрібно витягнути складають лише один запис, то застосовується метод fetchone().

– Функція get_performance_actors створена для отримання з бази даних імен та прізвищ акторів які приймають участь у виставі за її назвою. Так само як і функція get_theater_address застосовується у вікні “Афіша театрів”, але самі отримані значення знаходяться на фреймі “Вистава”. У цій функції застосовується метод fetchall(), який повертає список кортежів.

– Функція get_performance_start_date призначена для отримання даних про дату початку проведення вистави за її назвою. За допомогою методу fetchone() витягує з бази даних інформацію про неї у вікні афіші, та зазначається у деталях обраної вистави.

– get_theaters_genres – це функція, яка необхідна для отримання направлень (жанрів) театру в якому проводиться вистава, за його назвою. Так як один театр може мати багато жанрів, то в цій функції для одержання списку їх назв використовується метод курсору fetchall().

– Функція refresh_performances_list знаходиться безпосередньо у вікні “Афіша театрів”, і направлена на оновлення списку вистав за назвою вибраного

театру у тому ж самому фреймі. Так як на відміну від інших знаходиться у класі, то вона має як серверну частину так і клієнтську. Ця функція оновлює список, тобто видаляє минулі значення та додає нові після кожного вибору назви театру. Для витягування даних у клієнтській частині застосовується метод fetchall().

– get_id_performance – це функція, що створена для отримання ID вистави обраної для бронювання квитків за її назвою. Є необхідною для визначення доступних дат і часу, ID виступів, а також встановлені ціни квитка. Щоб отримати дані про ID вистави був обраний метод fetchone(), так як не потрібно повертати список, лише одне значення.

– Функція get_user_id містить команди та запити призначені для отримання ID користувача за логіном з яким був проведений вхід до особистого кабінету. Використовується такий самий метод для витягування інформації з бази даних.

– get_theater_id – це функція, яка необхідна для отримання ID театру за допомогою його назви, міста в якому він знаходиться а також адреси визначених раніше. Так як містить лише одне значення то одержується за допомогою методу курсору fetchone().

– get_performance_dates – це функція, яка здійснює заповнення списку вибору датою та часом проведення вистави. Вона застосовується у вікні для бронювання квитків, і показує користувачу всі доступні вистави за вказаним ID вистави, який визначається за допомогою функції get_id_performance. Так як у результаті потрібно отримати список, використовується fetchall().

– Функція get_action_id потрібна для отримання значення ID виступу, за допомогою ID вистави, та дати та часу виступу котрий був обраний серед списку доступних варіантів для бронювання. Оскільки має одне значення, то застосовується fetchone() для витягування його.

– Функції get_seats_numbers, get_rows_numbers, get_seats_categories необхідні для заповнення списків вибору номера місця, ряду та виду його категорії на виставу відповідно. Кожна з цих функцій має однакову структуру, і відрізняється лише запитами. Ці дані можна отримати за допомогою визначених

ID виступу та його дати та часу. Крім того для того щоб їх витягнути з бази даних , використовується метод fetchall() в кожній функції.

– get_ticket_price – це функція створена для отримання ціни квитка на виставу за вибраними параметрами ID виступу, його дати та часу, номерів місця, ряду та виду його категорії. Перший параметр був отриманий за допомогою функції, інші були обрані користувачем у вікні бронювання в списках доступних варіантів. Значення ціни квитка отримується з бази даних через використання fetchone().

– get_ticket_id – це функція необхідна для отримання ID квитка за допомогою ID театру, ID виступу, номерів місця, ряду та виду його категорії. Вона є потрібною для зміни статусу квитка при його бронюванні. Для витягування даних застосовується такий самий метод, як і в попередній функції.

– get_ticket_state – це функція для отримання стану квитка за визначеним у get_ticket_id - ID квитка. Для витягування цієї інформації можна використати той же самий метод, який застосовувався в попередніх двох функціях.

– update_ticket_state – це функція для оновлення стану квитка та встановлення значення ID користувача. Тобто мається на увазі, що вона змінює значення ticket_state з “Небронюваний” на “Бронюваний”, та замість значення NULL для id_user, вона встановлює ID того хто бажає забронювати квиток. Так як в цій функції відбуваються зміни у базі даних то застосовується метод commit() для їх збереження.

– Функція get_user_tickets використовується для отримання інформації про заброньовані квитки користувача за його ID. Вона використовується у вікні “Квитки користувача” та відображає наступні дані: ID квитка, назва вистави, театру, міста та адреси де вона проводиться, дата та час виступу, його ціна, а також номера місця, його ряду і категорії. Вся ця інформація отримується за допомогою методу fetchall().

– Функція cancel_reservation необхідна для скасування бронювання квитка і видалення прив'язки до користувача. Іншими словами вона виконує дії

протилежні update_ticket_state: змінює значення ticket_state з “Броньований” на “Неброньований”, та замість значення ID користувача, встановлюється NULL. І так як в ній також відбуваються зміни у базі даних то застосовується метод commit().

Для ретельного огляду функцій, які реалізували серверну частину були проведені запити операції безпосередньо у MySQL для перевірки успішності результатів, які представлені на рисунках 3.5 -3.28, рисунках А.1- А.24.

У зв’язку з обмеженим обсягом роботи, наведемо деякі результати SQL - запитів. Додаток А.

3.2 Розробка клієнтської частини

У розділі 2 було розглянуто компоненти клієнтської частини, яка розроблюється за допомогою бібліотеки wxPython. Саме тому в першу чергу створюються вікна, які є контейнерами для інших елементів інтерфейсу. У додатку наявні 10 головних вікон, які відповідають за основний функціонал та 6 додаткових, які відкриваються з меню та містять текстові пояснення до нього. Крім того, наявні вікна повідомлення, які з’являються для того щоб повідомити користувача про успішний або невдалий результат проведення ним певних дій.

У програмному додатку реалізуються 2 класи:

– App – це загальний клас додатку, який представляє весь застосунок в цілому, і за допомогою якого запускається повний цикл обробки подій.

– Frame – клас, який створює окреме стандартне вікно.

Для простішого та більш зручного створення та модифікації зазначених вікон розробляються власні класи, що успадковуються від Frame. У цих класах, реалізований конструктор __init__,за допомогою якого і створюються форми. В ньому наявні наступні базові параметри: посилання на батьківський клас (parent), ID фрейму (id), назва (title), позиція (pos), розмір вікна (size), його стилізація (style). Параметр стиль визначає те що дозволено вікну: наявність назви, кнопки згортання, розгортання та закриття, системного меню, можливість зміни розміру

тощо. У створених фреймах встановлено стиль за замовчуванням, він містить всі перераховані вище.

Приклад класу, що успадковується від Frame наведено на рис. 3.29.

У самому класі також доступне використання різних методів бібліотеки wxPython для встановлення стилізація вікна, тобто визначення кольору фону та тексту, шрифту тощо. У всіх створених формах були застосовані методи: SetBackgroundColour, який встановлює колір фону за допомогою RGB, та SetSizeHints, який встановлює мінімальний та максимальний розмір вікна (був позначений за замовчуванням, так як розмір вказувався у параметрах).

Перш ніж почати заповнювати інтерфейс іншими елементами (віджетами), на фреймі спочатку необхідно встановити макет (Layout), а саме сайзери (Sizer).

wx.Sizer - це абстрактний базовий клас, який використовується для розміщення дочірніх вікон у вікні [28].

Цей клас неможна використовувати безпосередньо, тому застосовуються його підкласи: wx.BoxSizer, wx.StaticBoxSizer, wx.GridSizer, wx.FlexGridSizer, wx.WrapSizer та wx.GridBagSizer [28].

В додатку застосовуються лише 3 підкласи сайзерів: BoxSizer, GridSizer, FlexGridSizer.

wx.BoxSizer – це сайзер, який розташовує елементи інтерфейсу у ряд, тобто горизонтально (wx.HORIZONTAL) або в стовпчик, тобто вертикально (wx.VERTICAL) [29].

wx.GridSizer – це сайзер, який розташовує елементи у вигляді двомірної таблиці, де всі комірки мають однаковий розмір [30].

wx.FlexGridSizer – це сайзер, який розташовує елементи у вигляді двомірної таблиці, де всі поля в одному рядку мають однакову висоту, а всі поля в одному стовпчику - однакову ширину [31].

Після огляду відомостей, які є необхідними для розробки кожного фрейму, наступним етапом буде формування опису класів.

1. Клас MyFrame1 (назва фрейму – “BT-TICKET”) – містить початкове вікно програмного додатку.

Ця форма складається з вертикального сайзеру - BoxSizer, яких містить:

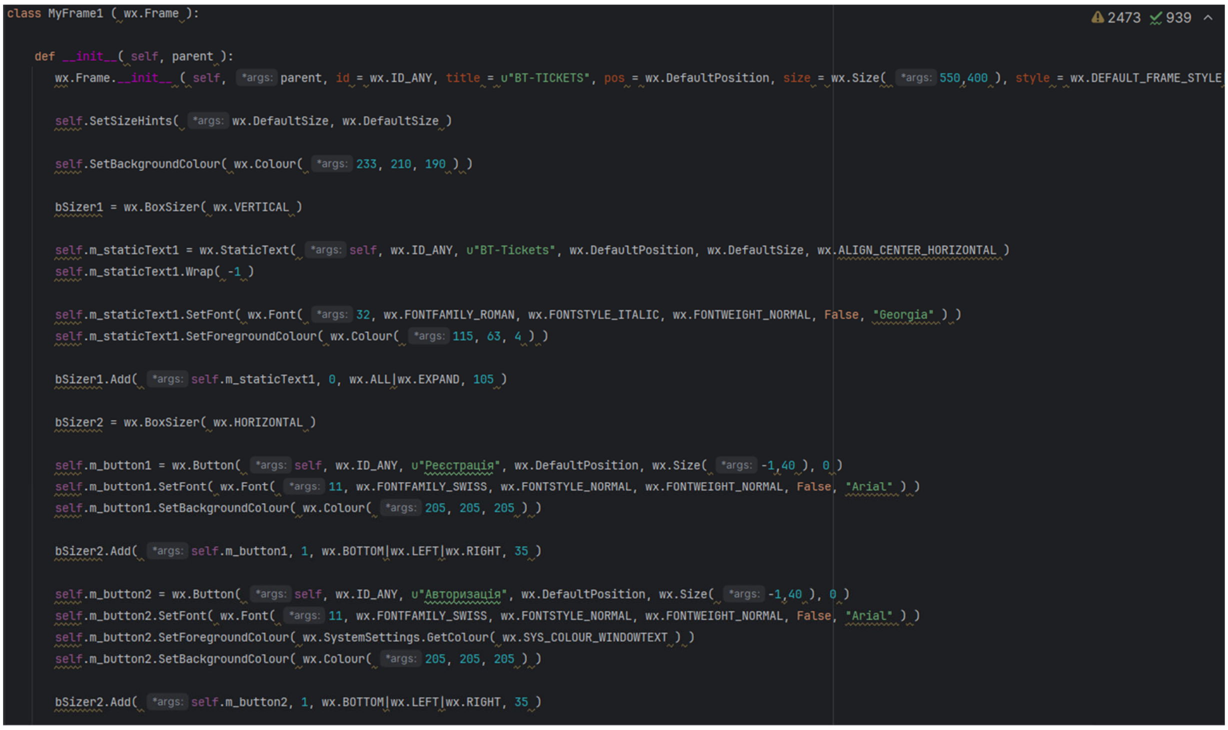
– один текстовий елемент типу StaticText;

– горизонтальний BoxSizer, в якому 2 елементи типу Button.

Для створення текстового елементу та кнопок можна скористатися класами wx.StaticText та wx.Button відповідно.

Для додавання цих елементів інтерфейсу до сайзеру використовується метод Add, який має такі основні параметри: посилання на елемент, параметр пропорції (proportion), аспекти поведінки(flag), границі відступу (border).

Створення і додавання елементів цього фрейму зображено на рис. 3.29.



```
class MyFrame1 ( wx.Frame ):

    def __init__( self, parent ):
        wx.Frame.__init__ ( self, *args: parent, id = wx.ID_ANY, title = u"BT-TICKETS", pos = wx.DefaultPosition, size = wx.Size( *args: 550,400 ), style = wx.DEFAULT_FRAME_STYLE

        self.SetSizeHints( *args: wx.DefaultSize, wx.DefaultSize )

        self.SetBackgroundColour( wx.Colour( *args: 233, 210, 190 ) )

        bSizer1 = wx.BoxSizer( wx.VERTICAL )

        self.m_staticText1 = wx.StaticText( *args: self, wx.ID_ANY, u"BT-Tickets", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTER_HORIZONTAL )
        self.m_staticText1.Wrap( -1 )

        self.m_staticText1.SetFont( wx.Font( *args: 32, wx.FONTFAMILY_ROMAN, wx.FONTSTYLE_ITALIC, wx.FONTWEIGHT_NORMAL, False, "Georgia" ) )
        self.m_staticText1.SetForegroundColour( wx.Colour( *args: 115, 63, 4 ) )

        bSizer1.Add( *args: self.m_staticText1, 0, wx.ALL|wx.EXPAND, 105 )

        bSizer2 = wx.BoxSizer( wx.HORIZONTAL )

        self.m_button1 = wx.Button( *args: self, wx.ID_ANY, u"Реєстрація", wx.DefaultPosition, wx.Size( *args: -1,40 ), 0 )
        self.m_button1.SetFont( wx.Font( *args: 11, wx.FONTFAMILY_SWISS, wx.FONTSTYLE_NORMAL, wx.FONTWEIGHT_NORMAL, False, "Arial" ) )
        self.m_button1.SetBackgroundColour( wx.Colour( *args: 205, 205, 205 ) )

        bSizer2.Add( *args: self.m_button1, 1, wx.BOTTOM|wx.LEFT|wx.RIGHT, 35 )

        self.m_button2 = wx.Button( *args: self, wx.ID_ANY, u"Авторизація", wx.DefaultPosition, wx.Size( *args: -1,40 ), 0 )
        self.m_button2.SetFont( wx.Font( *args: 11, wx.FONTFAMILY_SWISS, wx.FONTSTYLE_NORMAL, wx.FONTWEIGHT_NORMAL, False, "Arial" ) )
        self.m_button2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOWTEXT ) )
        self.m_button2.SetBackgroundColour( wx.Colour( *args: 205, 205, 205 ) )

        bSizer2.Add( *args: self.m_button2, 1, wx.BOTTOM|wx.LEFT|wx.RIGHT, 35 )
```

Рис. 3.29 – Програмний код класу MyFrame1

Щоб зробити створені кнопки функціональними вони пов’язуються з подіями за допомогою метода Bind(), та обробника wx.EVT_BUTTON, який викликається після натискання на них.

При натисканні на кнопку m_button1 та m_button2 відбувається знищення цієї форми за допомогою методу Destroy(), та перехід до вікна “Реєстрації” або “Авторизації” відповідно.

Код який ілюструє використання методу Bind для прив'язки подій до кнопок наведено на рис.3.30.

```
self.m_button1.Bind(wx.EVT_BUTTON, self.m_button1_clk)
self.m_button2.Bind(wx.EVT_BUTTON, self.m_button2_clk)
```

Рис. 3.30 – Код методу Bind() класу MyFrame1

Код функцій-обробників для описаних вище кнопок зображено на рис. 3.31.

```
1 usage
def m_button1_clk(self, event): # Перехід до реєстрації користувача
    self.Destroy()
    frame1 = MyFrame2(None)
    frame1.Show()
    event.Skip()

1 usage
def m_button2_clk(self, event): # Перехід до авторизації користувача
    self.Destroy()
    frame2 = MyFrame3(None)
    frame2.Show()
    event.Skip()
```

Рис. 3.31 – Код функцій-обробників класу MyFrame1

2. Клас MyFrame2 (назва фрейму – “Реєстрація”) – містить форму для створення облікового запису користувача.

Форма має вертикальний BoxSizer, який містить:

– сайзер FlexGridSizer: 3 елемента типу StaticText та 3 типу TextCtrl.

– горизонтальний BoxSizer: 2 елементи типу Button.

Для створення текстового поля можна скористатися класом wx.TextCtrl. Ці елементи необхідні, так як в них користувач вводить персональні дані для реєстрації і створенні особистого кабінету. Після чого при натисканні на кнопку “Вхід” відбувається ряд подій які включають запити до серверної частини та бази

даних для порівняння введених у ці поля даних з записами у базі даних, і у випадку відсутності їх додавання. Крім того після отримання відповіді від серверу, що супроводжується появою вікон повідомлень (wx.MessageBox), відбувається знищення цього вікна та перехід до наступного – “Особистий кабінет”. Код функцій які здійснюють ці події наведена на рис.3.32.

```
def on_registration_click(self, event):
    name_user = self.m_textCtrl1.GetValue()
    email_user = self.m_textCtrl2.GetValue()
    password_user = self.m_textCtrl3.GetValue()

    # Перевірка на заповненість всіх полів
    if not name_user or not email_user or not password_user:
        wx.MessageBox( message: "Будь ласка, заповніть всі поля!", caption: "Помилка", wx.OK | wx.ICON_ERROR)
        return  # Завершення виконання методу, якщо є пусті поля

    conn = connect_to_mysql()

    # Перевірка наявності користувача з вказаними логіном, email та паролем
    if check_duplicate_user(conn, name_user, email_user, password_user):
        wx.MessageBox( message: "Користувач з такими даними вже існує!", caption: "Помилка", wx.OK | wx.ICON_ERROR)
    else:
        # Додавання користувача в базу даних
        add_user(conn, name_user, email_user, password_user)
        wx.MessageBox( message: "Реєстрація проведена успішно!", caption: "Успіх", wx.OK | wx.ICON_INFORMATION)
        self.open_personal_cabinet(name_user)

    if conn:
        conn.close()

# Функція для відкриття особистого кабінету з передачею імені користувача
1 usage
def open_personal_cabinet(self, name_user):
    self.Destroy()
    frame3 = MyFrame4( parent: None, name_user)
    frame3.Show()
```

Рис. 3.32 – Код функцій обробників кнопки “Вхід” класу MyFrame2

При клацанні на кнопку “Назад” відбувається подія руйнування фрейму повернення до минулого “BT-TICKET”. Код цієї функції обробника має такі самі команди, як і ті що прив'язані до кнопок на минулій формі.

3. Клас MyFrame3 (назва фрейму – “Авторизація”) – містить форму для входу до вже створеного облікового запису користувача. Має схожий інтерфейс с фреймом реєстрації, але відсутнє поле “Email”. Саме тому, форма має вертикальний BoxSizer, який містить:

– сайзер FlexGridSizer: 2 елемента типу StaticText та 2 типу TextCtrl.

– горизонтальний BoxSizer: 2 елементи типу Button.

При натисканні кнопки “Вхід” так само відбувається подія перевірки наявності користувача з вказаними даними, але в цьому випадку при авторизації це є обов’язковим етапом для входу до кабінету. Якщо запис у базі даних знайдено то за допомогою вікна повідомлення (wx.MessageBox), приходить сповіщення про це, після що відбувається знищення форми і вхід до особистого кабінету. Функція яка виконує описані події зображена на рис. 3.33.

```
def on_login_click(self, event):
    # Отримуємо введені користувачем дані
    name_user = self.m_textCtrl4.GetValue()
    password_user = self.m_textCtrl5.GetValue()

    # Перевірка заповнення полів
    if not name_user or not password_user:
        wx.MessageBox( message: "Будь ласка, заповніть всі поля!", caption: "Помилка", wx.OK | wx.ICON_ERROR)
        return  # Завершення виконання методу якщо є пусті

    conn = connect_to_mysql()

    # Перевірка даних авторизації в базі даних
    if check_credentials(conn, name_user, password_user):
        wx.MessageBox( message: "Авторизація проведена успішно!", caption: "Успіх", wx.OK | wx.ICON_INFORMATION)
        self.open_personal_cabinet_2(name_user)
    else:
        wx.MessageBox( message: "Користувача з такими даними не існує!", caption: "Помилка", wx.OK | wx.ICON_ERROR)

    if conn:
        conn.close()

# Функція для відкриття особистого кабінету з передачею імені користувача
1 usage
def open_personal_cabinet_2(self, name_user):
    self.Destroy()
    frame4 = MyFrame4( parent: None, name_user)
    frame4.Show()
```

Рис. 3.33 – Код функцій обробників кнопки “Вхід” класу MyFrame3

4. Клас MyFrame4 (назва фрейму – “Особистий кабінет”) – містить вікно, де знаходиться логін користувача, а також функціонал, який включає: здійснення броні квитка, перегляд вже отриманих, перевірка особистих даних, вихід з особистого кабінету.

Ця форма містить меню, з якого можна відкрити вікно “Допомога”, та вертикальний BoxSizer, який містить наступні сайзери та віджети:

– горизонтальний BoxSizer: 1 елемент типу Button;

– один текстовий елемент типу StaticText;

– BoxSizer (wx.HORIZONTAL): 2 елементи типу Button;

– BoxSizer (wx.HORIZONTAL): 1 елемент типу Button.

Текстовий елемент типу StaticText містить логін користувача, який передається з вікна “Реєстрації” або “Авторизації”. Що ж до кнопок, то всі вони містять подію стирання цієї форми та переходу до іншого вікна. Перша з позначкою трьох крапок відкриває фрейм “Персональні дані”, які були введені користувачем при створені облікового запису. При натисканні на кнопку “Вистави” відбувається подія переходу до вікна “Локація” – перше вікно для бронювання квитка у театр. Наступна кнопка - “Квитки користувача”, відкриває форму з такою ж назвою, яка містить інформацію про заброньовані квитки користувача. І остання з назвою “Вихід” містить подію переходу до першого фрейму - “BT-TICKET”.

5. Клас MyFrame5 (назва фрейму - “Персональні дані”) – містить вікно, в якому знаходяться дані користувача введені ним у формі “Реєстрація” для створення облікового запису, а саме: логін, email та пароль.

Ця форма містить меню, з якого можна відкрити вікно “Допомога”, та вертикальний BoxSizer, який містить:

– сайзер FlexGridSizer: 6 елементів типу StaticText;

– горизонтальний BoxSizer: 1 елемент типу Button.

У сайзері FlexGridSize наявні два стовпчики: в першому вказані назви облікових даних, у другому – сама інформація про них, яка отримується з серверної частини за допомогою функції get_user_info, описаної вище. При натисканні кнопки “Назад” відбувається подія знищення цього вікна та відкриття особистого кабінету.

6. Клас MyFrame7 (назва фрейму – “Локація”) – містить вікно, в якому знаходиться список назв міст, з яких користувач може обрати те де він бажає відвідати виставу в театрі.

Форма, так само як і дві минулі, має меню, з якого можна відкрити вікно “Допомога”, та вертикальний BoxSizer, який містить:

– Один елемент типу ListBox

– BoxSizer (wx.HORIZONTAL): 2 елементи типу Button.

Для створення списку використовується клас wx.ListBox. Щоб зберегти вибране значення та використати його пізніше для отримання інших даних для бронювання квитка, можна застосувати метод wx.ListBox – GetStringSelection.

При натисканні кнопки з назвою “Назад” відбувається повернення до особистого кабінету. Якщо клацнути на ту, що має назву “Вибрати”, то здійснюється подія для зберігання вибраного міста описана вище, та перехід до вікна “Афіша театрів”.

7. Клас MyFrame8 (назва фрейму – “Афіша театру”) – містить вікно, в якому знаходяться два списки в першому з яких доступні театри вибраного на минулому фреймі міста, а в другому назви вистав.

Форма містить меню, з якого можна відкрити вікно “Допомога”, та вертикальний BoxSizer, який має:

– сайзер GridSizer: 2 елементи типу ListCtrl;

– горизонтальний BoxSizer: 2 елементи типу Button.

Віджет типу ListCtrl можна створити за допомогою команди wx.ListCtrl, крім того цей елемент є списком табличного виду. До першого списку додана подія оновлення списку відповідно до назви міста вибраного користувачем у минулій формі. У другому віджеті ListCtrl було встановлено можливість заповнення його новими значеннями відповідно до вибраного театру у іншому списку.

Після натискання кнопки з назвою “Назад” відбувається повернення до минулого вікна. При клацанні на “Вибрати” - здійснюється подія для зберігання вибраного театру та вистави, а також переходу до форми - “Вистава”.

8. Клас MyFrame9 (назва фрейму – “Вистава”) – складається з вікна, яке містить детальну інформацію про виставу вибрану в афіші театрів.

Форма має меню, з якого можна відкрити вікно “Допомога”, та вертикальний BoxSizer, який містить:

– горизонтальний BoxSizer: 2 елементи: типу StaticBitmap та ListCtrl;

– BoxSizer (wx.HORIZONTAL): 2 елементи типу Button.

Віджет типу StaticBitmap – це статичне зображення, яке створюється за допомогою команди wx.StaticBitmap. У ListCtrl знаходиться інформація про виставу (її назва, театр в якому вона проводиться, за якою адресою, та жанром, актори які приймають участь у виступі та дата початку).

При натисканні кнопки з назвою “Назад” відбувається повернення до минулого вікна. При клацанні на “Вибрати” - здійснюється подія переходу до форми - “Бронювання”.

9. Клас MyFrame10 (назва фрейму - “Бронювання”) – містить форму для здійснення броні квитка на виставу вибрану у минулих вікнах.

Цей фрейм складається з меню, з якого можна відкрити вікно “Допомога”, та вертикального BoxSizer, в якому наявні наступні сайзери та віджети:

– горизонтальний BoxSizer: 1 елемент типу StaticBitmap та сайзер FlexGridSizer (6 об’єктів StaticText, 2 TextCtrl, 4 Choice);

– горизонтальний BoxSizer: 2 елементи типу Button.

Віджет Choice – це елемент, який має можливість вибору з випадаючого списку, та створюється через клас wx.Choice. У них знаходяться дані про доступні дати та час виступів, номера місць, рядів, та його категорій.

У елементах TextCtrl знаходяться назва вистави та ціна квитка на неї, яка вираховується за допомогою серверної частини.

Після клацання на кнопку “Назад” відбувається перехід до минулого фрейму з деталями вистави. При натисканні на “Бронювати”, здійснюються події зміни стану квитка на заброньований, встановлення ID користувача для нього, а також відкриття вікна заброньованих квитків.

10. Клас MyFrame11 (назва фрейму – “Квитки користувача”) – містить вікно, в якому знаходиться детальна інформація про заброньовані користувачем квитки, а також містить функціонал для їх видалення. Цей фрейм можливо відкрити з “Особистий кабінет” та “Бронювання”.

Форма складається з меню, з якого можна відкрити вікно “Допомога”, та вертикального BoxSizer, в якому наявні:

– один текстовий елемент типу StaticText;

– один віджет ListCtrl;

– горизонтальний BoxSizer: 2 елементи типу Button.

У списку табличного типу знаходиться вся інформація про заброньовані користувачем квитки, яка отримується з серверної частини за допомогою функції get_user_tickets.

У горизонтальному BoxSizer знаходяться кнопки після клацання на які відбуваються наступні події: “Особистий кабінет” – перехід до особистого кабінету користувача, “Видалити бронь” – скасування броні вибраного квитка у списку ListCtrl.

11. Класи MyFrame12 - MyFrame17 (“Допомога”) – це вікна які мають текстові пояснення щодо функціоналу фреймів звідки були викликані через пункт меню. На відміну від минулих форм при їх відкриті не знищують основне вікно. Вони складаються лише з горизонтального BoxSizer, у якому знаходиться текстовий віджет StaticText.

Для того щоб запустити додаток потрібно виконати описані далі команди. В першу чергу необхідно зробити ініціалізацію об’єкта wx.App, який був розглянутий вище, далі потрібно створити екземпляр класу MyFrame1 – початкового вікна, звідки відбуваються переходи до інших, і відобразити його на екрані через команду Show(). Останньою дією є запуск головного циклу обробки подій додатку - MainLoop(). Описаний вище код для виконання застосунку зображений на рис. 3.34.

```
app = wx.App()
frame = MyFrame1(None)
frame.Show()
app.MainLoop()
```

Рис. 3.34 – Код для запуску додатку

3.3 Тестування застосунку

В цьому підрозділі здійснюється тестування застосунку з метою перевірки відповідності функціональним вимогам, встановленим під час аналізу предметної області, а також його працездатності та надійності. Основні його етапи наведені нижче на рис. 4.1 - 4.17.

1. Тест вибору процедури входу

Після відкриття додатку наявні два способи входу до особистого кабінету користувача: реєстрація або авторизація. Ці два шляхи наявні у початковому вікні застосунку наведеному на рис. 4.1.

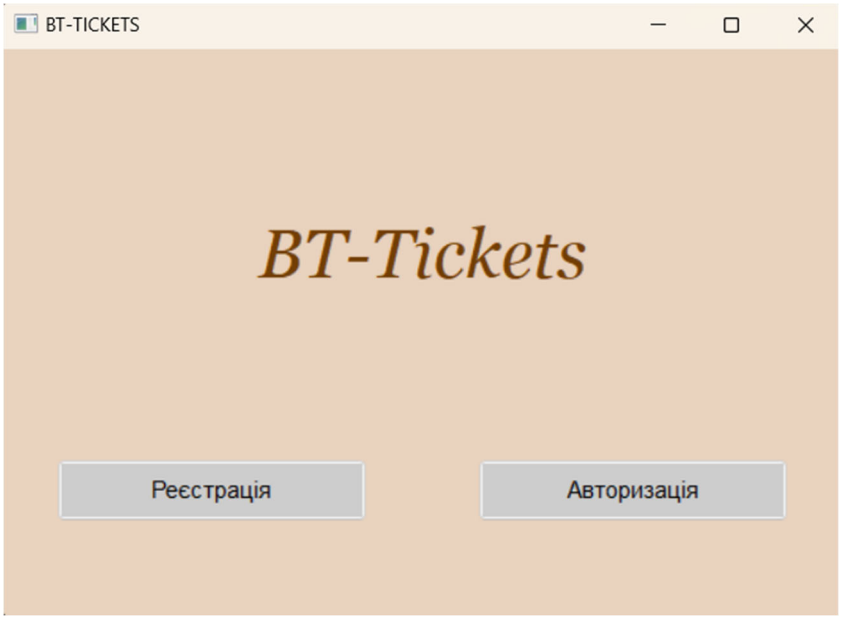


Рис.4.1 – Способи входу до особистого кабінету користувача

2. Тест реєстрації користувача.

При реєстрації можливі декілька варіантів подій: заповнення не всіх полів форми, введення даних користувача який вже існує та успішне створення облікового запису. В перших двох випадках з’являється вікно повідомлення про помилку, у третьому – про успішну операцію. Форма реєстрації та описані результати зображені на рис. 4.2 – 4.4 відповідно.

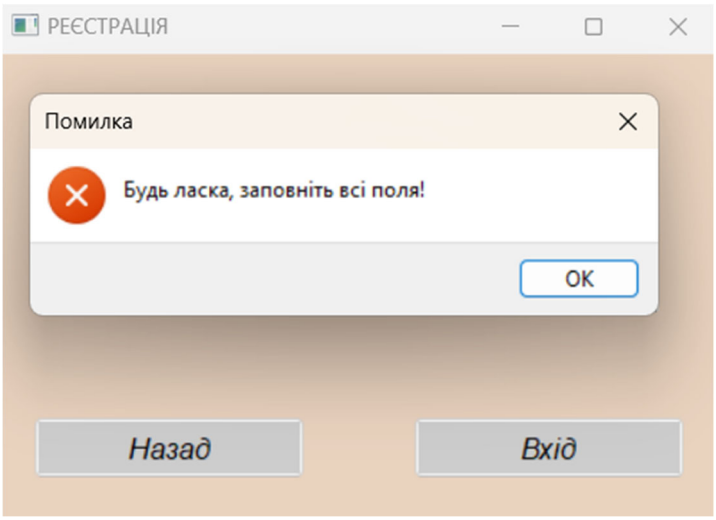


Рис. 4.2 – Повідомлення про наявність порожніх полів форми

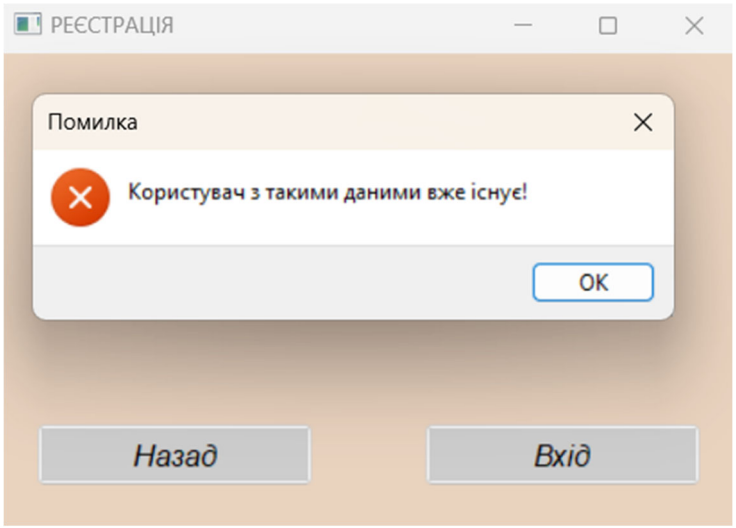


Рис. 4.3 – Повідомлення про наявність користувача з цими даними

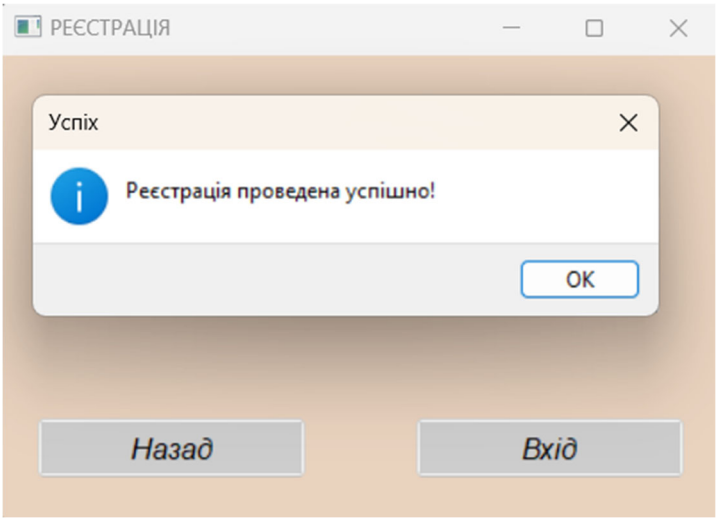


Рис.4.4 – Повідомлення про успішну реєстрацію користувача

3. Тест авторизації користувача.

При авторизації можуть статися наступні варіанти подій: заповнення не всіх полів форми, введення неправильних даних та успішне вхід до облікового запису. В перших двох випадках з’являється вікно повідомлення про помилку, у третьому – про успішну операцію. Описані результати останніх двох і зображені на рис. 4.5 – 4.6 відповідно.

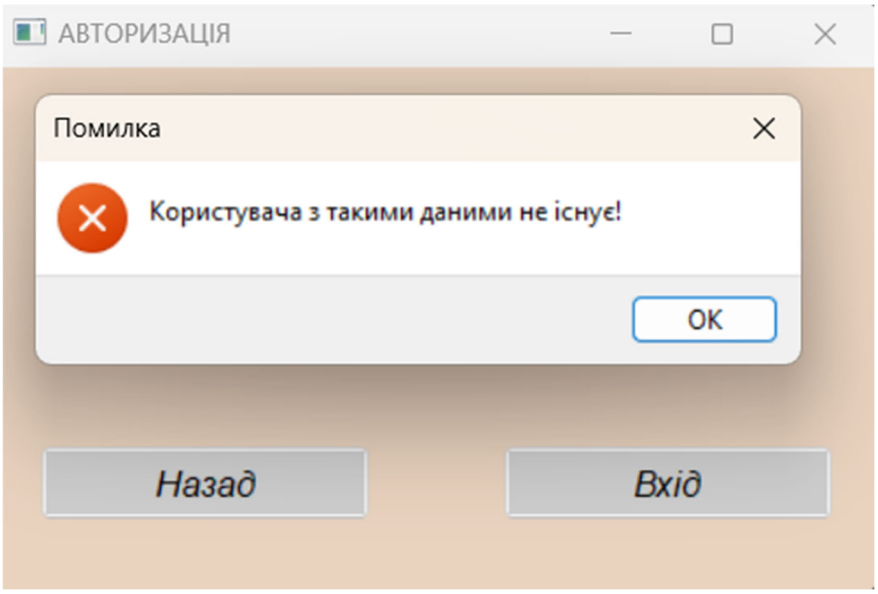


Рис. 4.5 – Повідомлення про помилку при введені даних

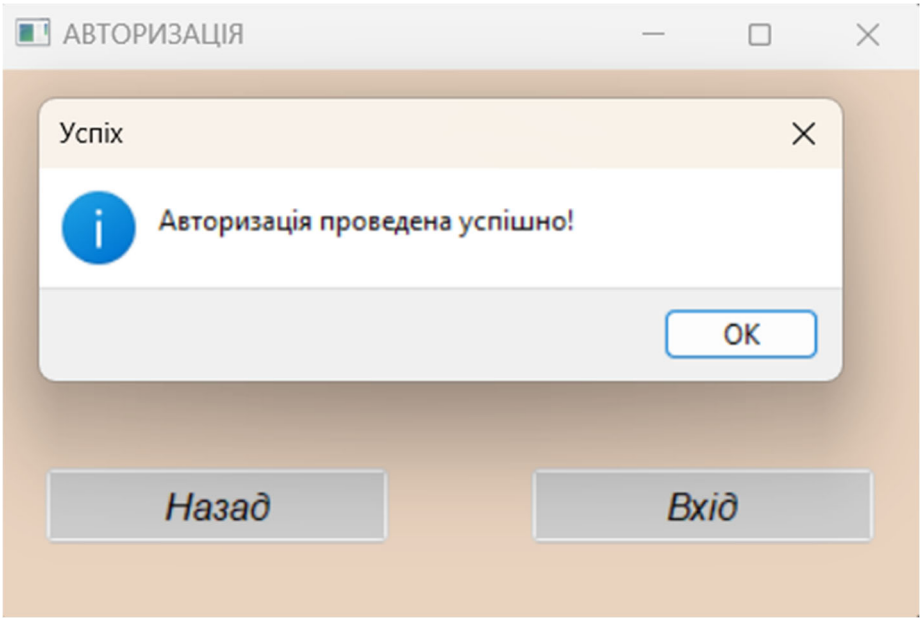


Рис. 4.6 – Повідомлення про успішну авторизацію

4. Тест здійснення перегляду персональних даних

Для того щоб переглянути інформацію користувача потрібно в особистому кабінеті натиснути кнопку с трьома крапками, яку наведено на рис. 4.8, після чого з’явиться вікно з усіма даними введеними при створені облікового запису. Вікно з цими даними наведено на рис. 4.7.

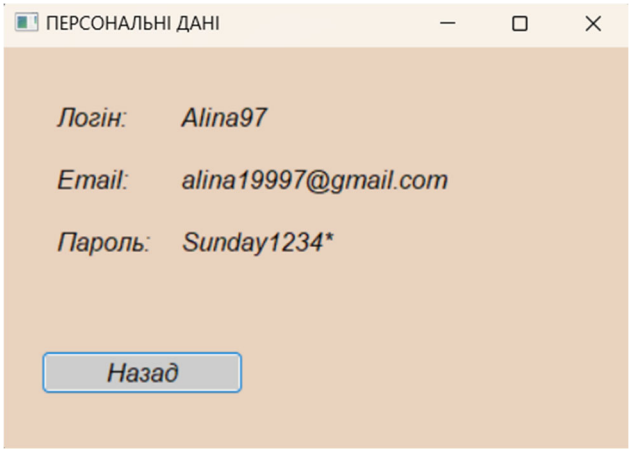


Рис. 4.7 – Перегляд персональних даних користувача

5. Тест здійснення броні квитка у театр.

Для того щоб здійснити бронь потрібно вибрати: локацію театру, його назву та бажану виставу. Після чого у вікні бронювання можна вибрати необхідні параметри для квитка: дата та час виступу, номер місця, ряду, та його категорію. Етапи отримання квитка наведені на рис. 4.8 – 4.12 відповідно.

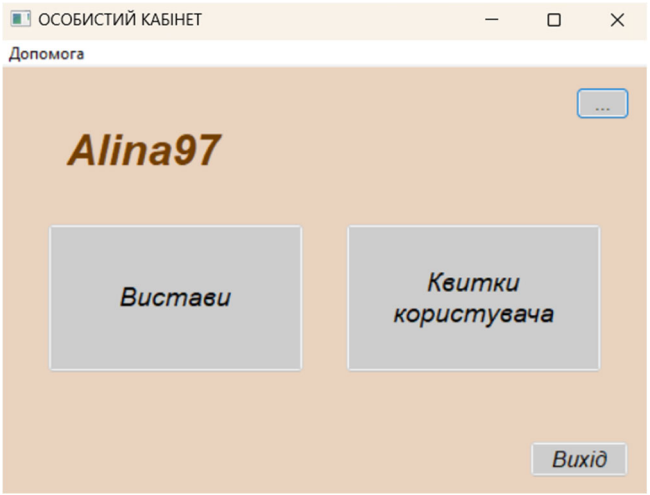


Рис. 4.8 – Вибір кнопки “Вистави” для початку бронювання

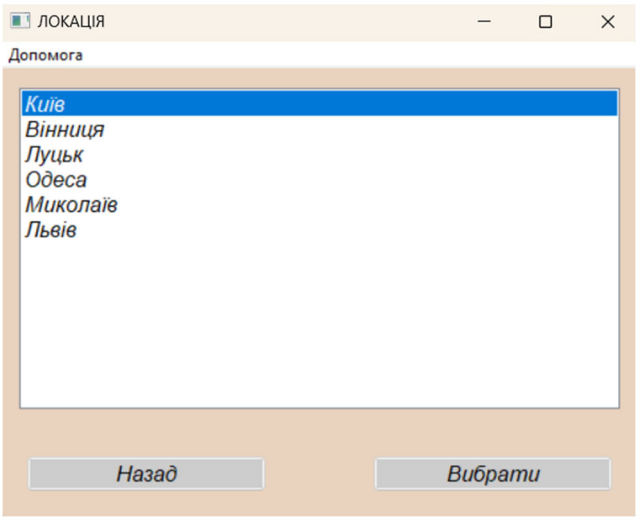


Рис. 4.9 – Вибір міста в якому потрібно забронювати квиток в театр

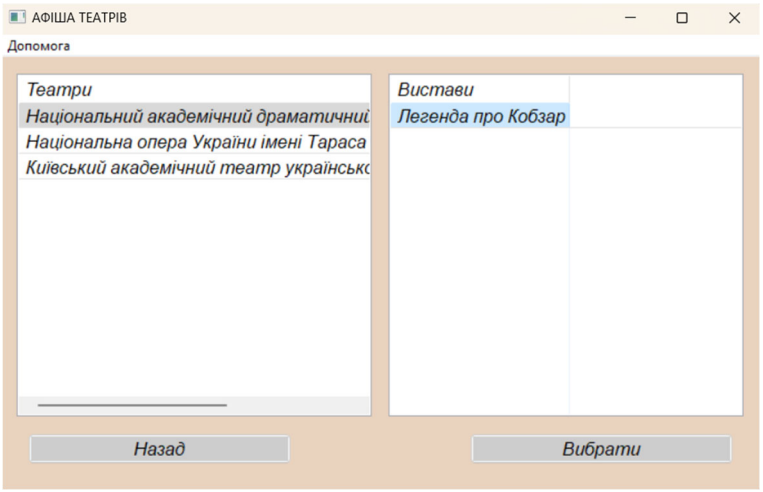


Рис.4.10 – Вибір театру та вистави на яку потрібен квиток

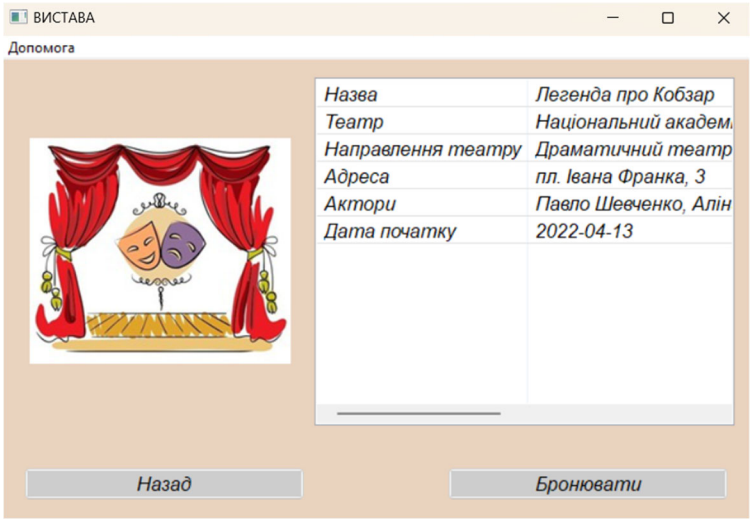


Рис. 4.11 – Перегляд деталей вистави

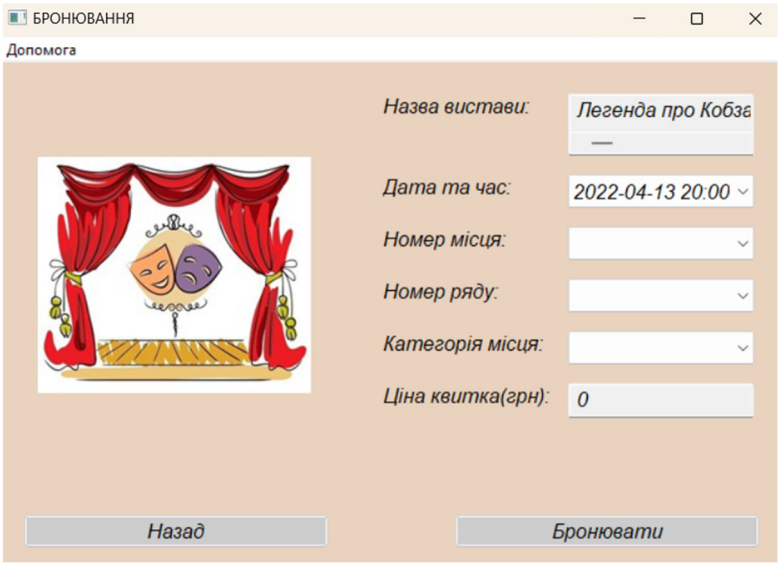


Рис. 4.12 – Вікно бронювання квитків на виставу

При здійсненні броні можливі декілька варіантів подій: вказані параметри неправильні (у випадку незаповненості деяких полів), квиток вже є заброньованим, та бронювання здійснено успішно. Описані результати зображені на рис. 4.13 – 4.15 відповідно.

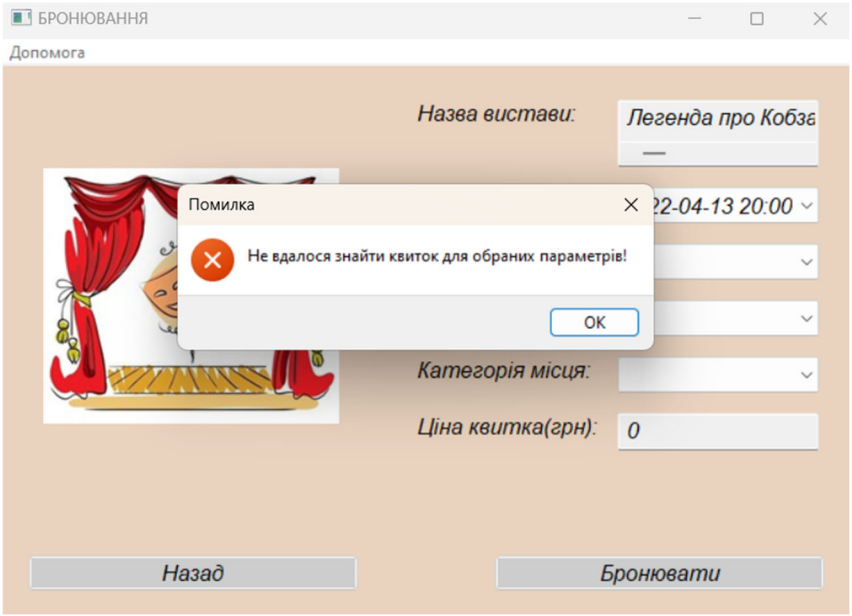


Рис. 4.13 – Повідомлення про деяких незаповненість параметрів

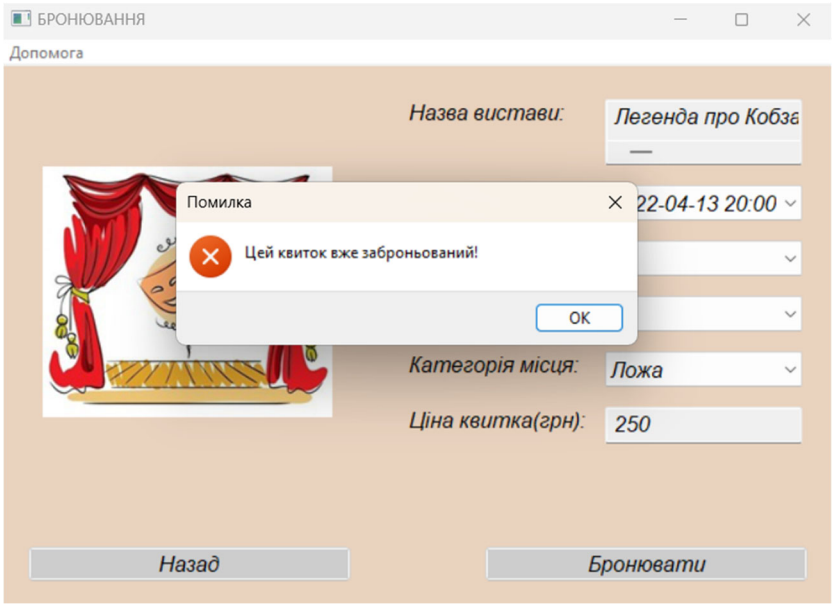


Рис. 4.14 – Повідомлення про наявність броні на вибраний квиток

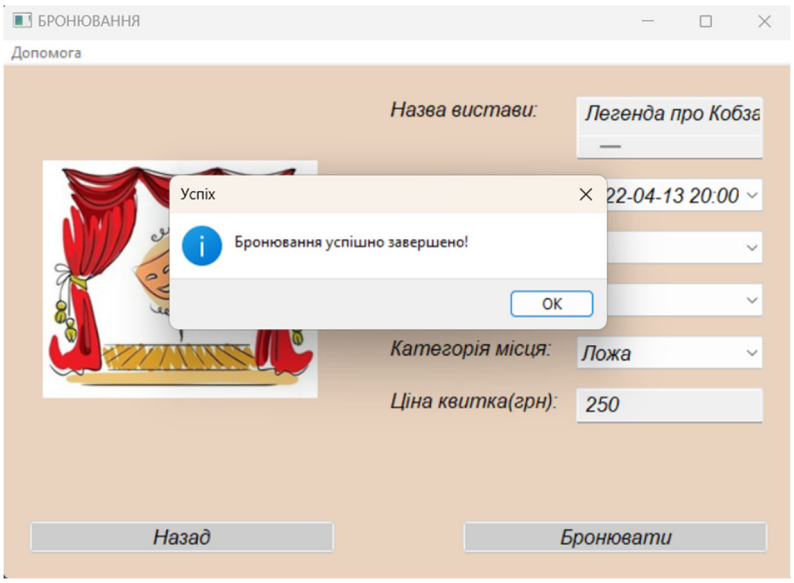


Рис. 4.15 – Повідомлення про успішне бронювання квитка

6. Тест скасування броні на квиток.

Для того щоб видалити бронювання потрібно перейти з особистого кабінету до списку квитків користувача та вибрати той який необхідно скасувати, після чого отримати повідомлення про успішний результат. Описаний процес зображений на рис. 4.16 - 4.17.

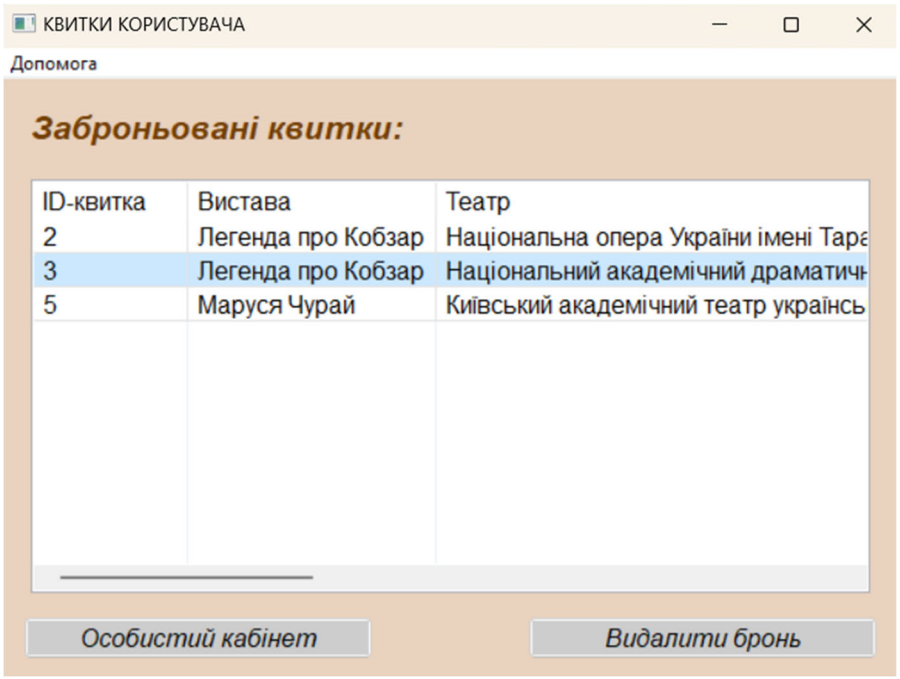


Рис. 4.16 – Вибір квитка для видалення зі списку заброньованих

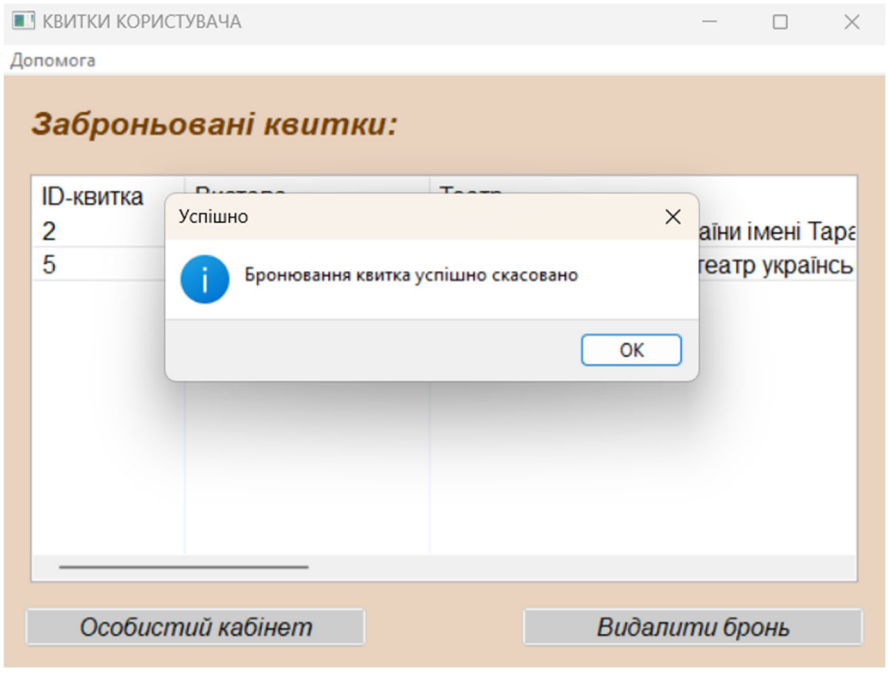


Рис. 4.17 – Повідомлення про успішне скасування броні на квиток

Проаналізувавши результати тестів можна зробити висновок що розроблений додаток відповідає визначеним функціональним вимогам, а також є працездатним, надійним, і має зручний інтерфейс користувача.

3.4 Висновки до розділу 3

Під час розробки програмного продукту був створений код що поділяється на декілька компонентів: серверна та клієнтська частина, які були реалізовані за допомогою MySQL Connector/Python та бібліотеки wxPython відповідно.

Для розробки серверної частини в першу чергу був проведений додатковий огляд зазначеного вище драйверу що з’єднує програми написані мовою Python з сервером MySQL. На основі цього були створені 25 функцій, що містять запити до бази даних: для витягування, додавання та оновлення інформації. Вони є необхідними для реалізації функціоналу додатку.

Наступним етапом було створення клієнтської частини, в якій відбувалося розроблення 16 класів, з використанням бібліотеки wxPython. За допомогою них було створено вікна, які є компонентами інтерфейсу користувача, а також контейнерами для інших елементів.

Після проведення додаткового вивчення засобів розробки та написання коду для реалізації додатку бронювання квитків у театри, було здійснено його тестування. Проаналізувавши отримані результати було визначено що застосунок є повністю працездатним, містить затребуваний функціонал, і має інтуїтивно зрозумілий, простий у використанні та зручний інтерфейс.

ВИСНОВКИ

Кваліфікаційна робота є зорієнтованою на розробці інформаційної системи для бронювання квитків у театри з використанням засобів мови програмування Python.

На початку роботи були поставлені основні задачі, які мають бути розв’язані наприкінці її виконання. Саме тому в першу чергу були проведені дослідження всієї предметної області, а також пошук та аналіз наявних аналогів, з метою визначення функціональних вимог, що повинні відповідати цим завданням. Після цього була здійснена додаткова пошукова робота для вибору засобів розробки програмного продукту, які здатні найкраще реалізувати визначені функції. Серед обраних інструментів наявні: бібліотека wxPython, необхідна для написання програмного коду інтерфейсу користувача, мова SQL та СУБД MySQL для створення та модифікації бази даних, а також драйвер MySQL Connector/Python який потрібен для розробки серверної частини, що відповідає за з’єднання GUI з MySQL.

Після закінчення першого етапу, наступним кроком було здійснення проектування програмного продукту на основі отриманих результатів досліджень. Цей розділ складається з: визначення основних компонентів застосунку, створення бази даних “Театральні квитки”, а також моделювання програмної системи. У результаті виконаної роботи було отримано: компоненти додатку та їх складові, вид архітектури розроблюваного програмного продукту (дворівнева клієнт-серверна), працездатна реляційна база даних, моделі (ERM, DFD) і UML – діаграми (Use Case, Class), які відображають функціонал, структури та потоки даних, що сприяє підвищенню продуктивності застосунку.

Останнім етапом для розв’язання завдань є розробка програмного коду, який складається з серверної та клієнтської частин. Перша повинна доповнювати інтерфейс користувача, за допомогою запитів до бази даних, які мають бути відповідними функціоналу. Тобто вона мати можливість перевірки наявності інформації, додавання облікових дані споживача, оновлення даних квитка,

витягування записів для їх відображення на екрані, тощо. Клієнтська частина містить вікна, які реалізують у собі функціонал, що відповідає завданням кваліфікаційної роботи. До цих вікон належать: початкове вікно входу, реєстрації, авторизації, особистого кабінету, персональних даних користувача, локації, афіші театрів, деталей вистави, бронювання та квитків споживача. Крім того були також розроблені вікна, що містять пояснення функціоналу.

У результаті проведеної роботи використовуючи теоретичний та емпіричний методи дослідження, була розроблена інформаційна система, яка містить зрозумілий та легкий у використанні інтерфейс користувача. Крім того була розроблена база даних “Театральні квитки”, в якій наявна інформація про театри, вистави, користувачів застосунку, та яка має з’єднання з GUI. Також інформаційна система містить функціонал для реєстрації, авторизації, здійснення та скасування броні на квиток, запису про стан квитка до бази даних, а також перегляду персональних даних.

Після отримання та аналізів результатів роботи можна зробити висновок що всі поставлені завдання успішно виконані, і мета досягнута. Розроблена інформаційна система є працездатною, надійною, містить необхідний функціонал та є легкою у використанні. При правильній модифікації може стати конкурентоспроможною серед відомих аналогів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сейрік Ю.Ю. Створення кросплатформених GUI - застосунків з використанням wxPython та MySQL. *Інформаційне суспільство: проблеми та перспективи*: матер. IX міжнар. наук.-практ. конф. Одеса, 24 травня 2024 (депановано).
2. Бутенко Т.А., Сирий В.М. Інформаційні системи та технології [Електронний ресурс]: навч. посібник Харків, 2020. С. 34-53 URL: http://surl.li/qofmv
3. Інформаційна система. Вікіпедія. URL: http://surl.li/krqi
4. Introduction Into Theater Arts: Definition, History, and More. Rockstar academy. URL: http://surl.li/tsvla
5. Словник театрознавчих термінів і понять [Електронний ресурс]: / укладачі Савченко І., Ліпницька І. Київ, 2021. 144 с. URL: https://enpuir.npu.edu.ua/bitstream/handle/123456789/37502/Slovnyk_Savchenko_Lipnytska.pdf?sequence=1
6. План глядацької зали. Полтавський академічний обласний український музично-драматичний театр імені М. Гоголя. URL: https://teatr-gogolya.pl.ua/pro-teatr/plan-glyadatskoji-zali
7. concert.ua. URL: https://concert.ua/uk
8. karabas. URL: https://karabas.com/ua/
9. todaytix. URL: https://www.todaytix.com/
10. SWOT-аналіз: що це таке та приклади використання. Wedex. URL: https://wedex.com.ua/blog/swot-analiz-shho-tse-take-ta-prikladi-vikoristannya/
11. Трофименко О.Г., Прокоп Ю.В., Логінова Н.І., Задерейко О.В. С++. Алгоритмізація та програмування технології [Електронний ресурс]: підручник. 2-ге вид. перероб. і доповн. Одеса: Фенікс, 2019. 477 с. URL: https://dspace.onua.edu.ua/server/api/core/bitstreams/45cb4ce7-e915-41b6-834c-8731dc8dd602/content
12. Front Matter. Python Reference Manual. URL: http://surl.li/tsynm
13. TIOBE Index for April 2024. TIOBE. URL: https://www.tiobe.com/tiobe-index/

14. Що таке SQL та Бази даних? Acode. URL: https://acode.com.ua/sql-intro/
15. DB-Engines Ranking. DB-Engines. URL: https://db-engines.com/en/ranking
16. Overview of wxPython. wxPython. URL: https://wxpython.org/pages/overview/
17. Chapter 1 Introduction to MySQL Connector/Python. MySQL. URL: https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html
18. Кліент-серверна архітектура. QATestLab training center. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/
19. Кліент-серверна архітектура. JavaRush Лекції. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture00
20. Моделювання програмного забезпечення [Електронний ресурс]: навч.-метод. посібник / Укладачі: Манаков С. Ю., Трофименко О. Г., Лобода Ю. Г., Дика А. І. Одеса: Фенікс, 2023. 145 с. URL: https://dspace.onua.edu.ua/server/api/core/bitstreams/daa317b6-7412-4d8c-b9fe-5dd3a7c6f523/content
21. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Інформаційні системи і технології в управлінні" для студентів напряму підготовки 6.030502 "Економічна кібернетика" всіх форм навчання: [Електронне ресурс] / уклад. Р.М. Яценко, К.С. Коваленко. – Харків : ХНЕУ ім. С. Кузнеця, 2015. – 137 с. URL: http://surl.li/tsawr
22. Поняття моделі даних, основні моделі даних. UA5.ORG. URL: https://ua5.org/database/1795-ponyattya-modeli-danyh-osnovni-modeli-danyh.html
23. cursor.MySQLCursor Class. MySQL Connectors Documentation. URL: https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html
24. MySQLCursor.execute() Method. MySQL Connectors Documentation. URL: https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-execute.html

25. MySQLCursor.fetchone() Method. MySQL Connectors Documentation. URL: https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-fetchone.html
26. MySQLCursor.fetchall() Method. MySQL Connectors Documentation. URL: https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-fetchall.html
27. MySQLCursor.close() Method. MySQL Connectors Documentation. URL: https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-close.html
28. wx.Sizer. wxPython Cross-Platform GUI library Documentation. URL: https://docs.wxpython.org/wx.Sizer.html#wx-sizer
29. wx.BoxSizer. wxPython Cross-Platform GUI library Documentation. URL: https://docs.wxpython.org/wx.BoxSizer.html#wx-boxsizer
30. wx.GridSizer. wxPython Cross-Platform GUI library Documentation. URL: https://docs.wxpython.org/wx.GridSizer.html#wx-gridsizer
31. wx.FlexGridSizer. wxPython Cross-Platform GUI library Documentation. URL: https://docs.wxpython.org/wx.FlexGridSizer.html#wx-flexgridsizer

ДОДАТКИ

Додаток А. Результати SQL – запитів

Рисунок А.1 – Запит функції add_user

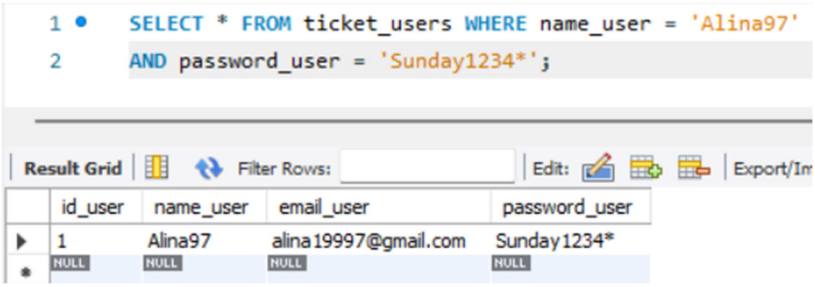


Рисунок А.2 – Запит функції check_credentials

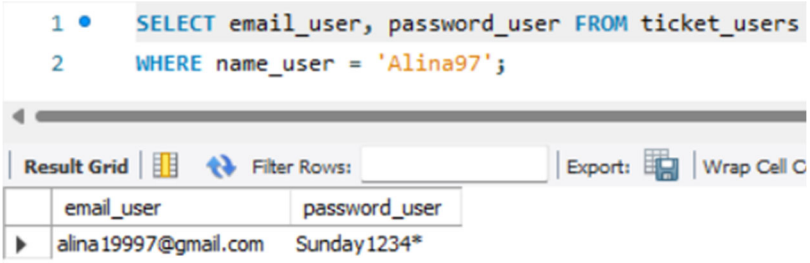


Рисунок А.3 – Запит функції get_user_info

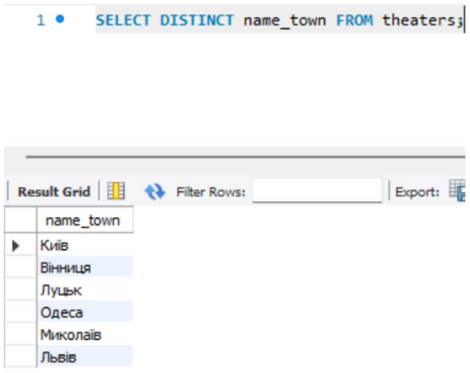


Рисунок А.4 – Запит функції get_unique_towns

```
1 •  SELECT name_theater FROM theaters
2    WHERE name_town = 'Київ';
```

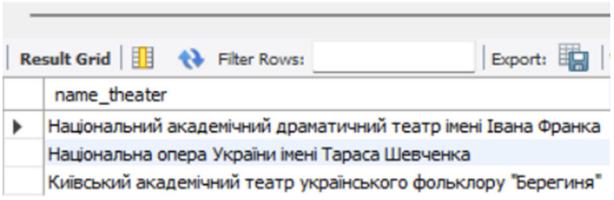


Рисунок А.5 – Запит функції get_theaters_in_town

```
1 •  SELECT name_address FROM theaters
2    WHERE name_theater = 'Національний академічний драматичний театр імені Івана Франка';
```

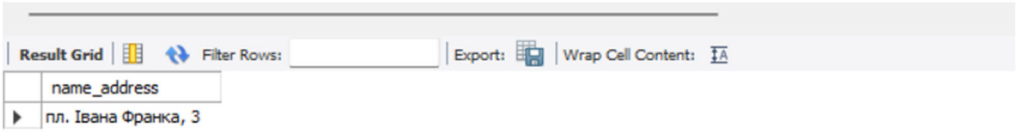


Рисунок А.6 – Запит функції get_theater_address

```
1 •  SELECT actors.name_actor FROM actions
2    INNER JOIN actors ON actions.id_actor = actors.id_actor
3    INNER JOIN performances ON actions.id_performance = performances.id_performance
4    WHERE performances.name_performance = 'Легенда про Кобзар';
```

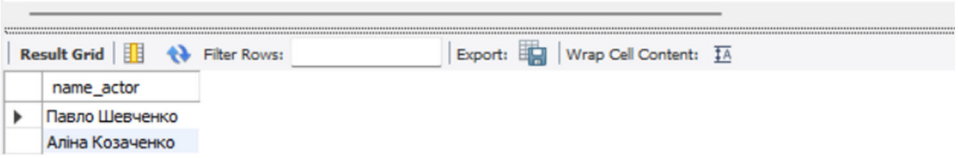


Рисунок А.7 – Запит функції get_performance_actors

```
1 •  SELECT start_date FROM performances
2    WHERE name_performance = 'Легенда про Кобзар';
```

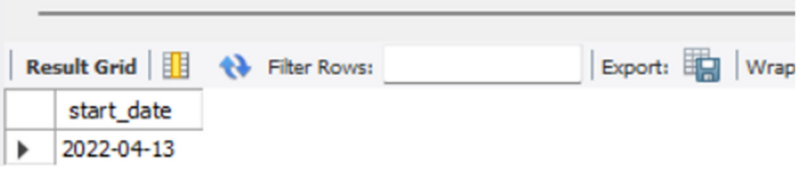


Рисунок А.8 – Запит функції get_performance_start_date

```
SELECT genres.name_genre FROM genres
JOIN c2c_theaters_genres ON genres.id_genre = c2c_theaters_genres.id_genre
JOIN theaters ON c2c_theaters_genres.id_theater = theaters.id_theater
WHERE theaters.name_theater = 'Національний академічний драматичний театр імені Івана Франка';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name_genre
Драматичний театр
Експериментальний театр
Театр сучасної драматургії

Рисунок А.9 – Запит функції get_theaters_genres

```
SELECT DISTINCT performances.name_performance FROM performances
INNER JOIN actions ON performances.id_performance = actions.id_performance
INNER JOIN tickets_programs ON actions.id_action = tickets_programs.id_action
INNER JOIN theaters ON tickets_programs.id_theater = theaters.id_theater
WHERE theaters.name_theater = 'Національний академічний драматичний театр імені Івана Франка';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name_performance
Легенда про Кобзар

Рисунок А.10 – Запит функції refresh_performances_list

```
SELECT id_performance FROM performances
WHERE name_performance = 'Легенда про Кобзар';
```

Result Grid | Filter Rows: | Edit:

id_performance
1

Рисунок А.11 – Запит функції get_id_performance

```
SELECT id_user FROM ticket_users
WHERE name_user = 'Alina97';
```

Result Grid | Filter Rows:

id_user
1

Рисунок А.12 – Запит функції get_user_id

```
SELECT id_theater FROM theaters
WHERE name_theater = 'Національний академічний драматичний театр імені Івана Франка'
AND name_town = 'Київ' AND name_address = 'пл. Івана Франка, 3';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id_theater
1

Рисунок А.13 – Запит функції get_theater_id

```
SELECT DISTINCT data_time_action FROM actions
WHERE id_performance = 1;
```

Result Grid | Filter Rows: | Export: | Wra

data_time_action
2022-04-13 20:00:00
2022-04-13 18:00:00

Рисунок А.14 – Запит функції get_performance_dates

```
SELECT id_action FROM actions
WHERE id_performance = 1
AND data_time_action = '2022-04-13 20:00:00';
```

Result Grid | Filter Rows: | Edit:

id_action
1

Рисунок А.15 – Запит функції get_action_id

```
SELECT DISTINCT tickets_programs.seats_number FROM tickets_programs
JOIN actions ON tickets_programs.id_action = actions.id_action
WHERE actions.id_action = 1
AND actions.data_time_action = '2022-04-13 20:00:00';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

seats_number
15
6

Рисунок А.16 – Запит функції get_seats_numbers

```
SELECT DISTINCT tickets_programs.rows_number FROM tickets_programs
JOIN actions ON tickets_programs.id_action = actions.id_action
WHERE actions.id_action = 1
AND actions.data_time_action = '2022-04-13 20:00:00';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

rows_number
3
4

Рисунок А.17 – Запит функції get_rows_numbers

```
SELECT DISTINCT tickets_programs.seats_categories FROM tickets_programs
JOIN actions ON tickets_programs.id_action = actions.id_action
WHERE actions.id_action = 1
AND actions.data_time_action = '2022-04-13 20:00:00';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

seats_categories
Партер
Ложа

Рисунок А.18 – Запит функції get_seats_categories

```
SELECT ticket_price FROM tickets_programs
JOIN actions ON tickets_programs.id_action = actions.id_action
WHERE tickets_programs.id_action = 1
AND data_time_action = '2022-04-13 20:00:00' AND seats_number = 6
AND rows_number = 4 AND seats_categories = 'Ложа';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ticket_price
250

Рисунок А.19 – Запит функції get_ticket_price

```
SELECT id_ticket FROM tickets_programs
WHERE id_theater = 1 AND id_action = 1 AND seats_number = 6
AND rows_number = 4 AND seats_categories = 'Ложа';
```

Result Grid | Filter Rows: | Edit: | Export/Import:

id_ticket
3

Рисунок А.20 – Запит функції get_ticket_id

```
1 •  SELECT ticket_state FROM tickets_programs
2    WHERE id_ticket = 3;
```

Result Grid | Filter Rows: | Export:

ticket_state
Неброньований

Рисунок А.21 – Запит функції get_ticket_state

```
1 •  UPDATE tickets_programs
2    SET ticket_state = 'Броньований', id_user = 1
3    WHERE id_ticket = 3;
```

Output

Action Output

#	Time	Action
1	16:33:30	UPDATE tickets_programs SET ticket_state = 'Броньований', id_user = 1 WHERE id_ticket = 3

Рисунок А.22 – Запит функції update_ticket_state

```
2    a.data_time_action, a.ticket_price, tp.seats_number, tp.rows_number, tp.seats_categories
3    FROM tickets_programs tp
4    JOIN actions a ON tp.id_action = a.id_action
5    JOIN performances p ON a.id_performance = p.id_performance
6    JOIN theaters t ON tp.id_theater = t.id_theater
7    WHERE tp.id_user = 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id_ticket	name_performance	name_theater	name_town	name_address	data_time_action
16	Танець під повітряними кульками	Одеський національний академічний театр опери та балету	Одеса	пров. Чайковського, 1	2021-06-09 20:00:00
19	Таємниця старого маяка	Одеський театр юного глядача імені Юрія Олеші	Одеса	вул. Грецька, 48-а	2022-05-13 20:00:00
24	Доля вогненних сердець	Миколаївський академічний художній драматичний театр	Миколаїв	вул. Адміральська, 25	2023-09-01 16:30:00

Рисунок А.23 – Запит функції get_user_tickets

```
1 •  UPDATE tickets_programs
2    SET ticket_state = 'Неброньований', id_user = NULL
3    WHERE id_ticket = 3;
```

Output

Action Output

#	Time	Action
1	16:35:17	UPDATE tickets_programs SET ticket_state = 'Неброньований', id_user = NULL WHERE id_ticket = 3

Рисунок А.24 – Запит функції cancel_reservation

Додаток Б. Лістинг програмного коду

```
import wx
import mysql.connector


#Підключення до бази даних
def connect_to_mysql():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="11052002baza*+",
            database="theater_tickets"
        )
        return conn
    except mysql.connector.Error as e:
        print("Помилка підключення до MySQL:", e)
        return None


# Class MyFrame1

class MyFrame1 ( wx.Frame ):

    def __init__( self, parent ):
        wx.Frame.__init__ ( self, parent, id = wx.ID_ANY, title =
u"BT-TICKETS", pos = wx.DefaultPosition, size = wx.Size( 550,400 ),
style = wx.DEFAULT_FRAME_STYLE|wx.TAB_TRAVERSAL )

        self.SetSizeHints( wx.DefaultSize, wx.DefaultSize )
        self.SetBackgroundColour( wx.Colour( 233, 210, 190 ) )

        bSizer1 = wx.BoxSizer( wx.VERTICAL )

        self.m_staticText1 = wx.StaticText( self, wx.ID_ANY, u"BT-
Tickets", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTER_HORIZONTAL )
        self.m_staticText1.Wrap( -1 )
        self.m_staticText1.SetFont( wx.Font( 32, wx.FONTFAMILY_ROMAN,
wx.FONTSTYLE_ITALIC, wx.FONTWEIGHT_NORMAL, False, "Georgia" ) )
        self.m_staticText1.SetForegroundColour( wx.Colour( 115, 63, 4
) )

        bSizer1.Add( self.m_staticText1, 0, wx.ALL|wx.EXPAND, 105 )
\        bSizer2 = wx.BoxSizer( wx.HORIZONTAL )

        self.m_button1 = wx.Button( self, wx.ID_ANY, u"Реєстрація",
wx.DefaultPosition, wx.Size( -1,40 ), 0 )
        self.m_button1.SetFont( wx.Font( 11, wx.FONTFAMILY_SWISS,
wx.FONTSTYLE_NORMAL, wx.FONTWEIGHT_NORMAL, False, "Arial" ) )
```

```
        self.m_button1.SetBackgroundColour( wx.Colour( 205, 205, 205
) )

        bSizer2.Add( self.m_button1, 1, wx.BOTTOM|wx.LEFT|wx.RIGHT,
35 )

        self.m_button2 = wx.Button( self, wx.ID_ANY, u"Авторизація",
wx.DefaultPosition, wx.Size( -1,40 ), 0 )
        self.m_button2.SetFont( wx.Font( 11, wx.FONTFAMILY_SWISS,
wx.FONTSTYLE_NORMAL, wx.FONTWEIGHT_NORMAL, False, "Arial" ) )
        self.m_button2.SetForegroundColour(
wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOWTEXT ) )
        self.m_button2.SetBackgroundColour( wx.Colour( 205, 205, 205
) )

        bSizer2.Add( self.m_button2, 1, wx.BOTTOM|wx.LEFT|wx.RIGHT,
35 )

        bSizer1.Add( bSizer2, 1, wx.EXPAND, 5 )
        self.SetSizer( bSizer1 )
        self.Layout()
        self.Centre( wx.BOTH )

        self.m_button1.Bind(wx.EVT_BUTTON, self.m_button1_clk)
        self.m_button2.Bind(wx.EVT_BUTTON, self.m_button2_clk)

    def m_button1_clk(self, event): # Перехід до реєстрації
користувача
        self.Destroy()
        frame1 = MyFrame2(None)
        frame1.Show()
        event.Skip()

    def m_button2_clk(self, event): # Перехід до авторизації
користувача
        self.Destroy()
        frame2 = MyFrame3(None)
        frame2.Show()
        event.Skip()


# Class MyFrame2

class MyFrame2 ( wx.Frame ):

    def __init__( self, parent ):
        wx.Frame.__init__ ( self, parent, id = wx.ID_ANY, title =
u"РЕЄСТРАЦІЯ", pos = wx.DefaultPosition, size = wx.Size( 420,300 ),
style = wx.DEFAULT_FRAME_STYLE|wx.TAB_TRAVERSAL )

        self.SetSizeHints( wx.DefaultSize, wx.DefaultSize )
        self.SetBackgroundColour( wx.Colour( 233, 210, 190 ) )
```

```
        bSizer3 = wx.BoxSizer( wx.VERTICAL )
        fgSizer1 = wx.FlexGridSizer( 3, 2, 10, 10 )
        fgSizer1.SetFlexibleDirection( wx.BOTH )
        fgSizer1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )

        self.m_staticText2 = wx.StaticText( self, wx.ID_ANY,
u"Логін:", wx.DefaultPosition, wx.DefaultSize, 0 )
        self.m_staticText2.Wrap( -1 )
        self.m_staticText2.SetFont( wx.Font( 12, wx.FONTFAMILY_SWISS,
wx.FONTSTYLE_ITALIC, wx.FONTWEIGHT_NORMAL, False, "Arial" ) )

        fgSizer1.Add( self.m_staticText2, 0, wx.ALL, 5 )

        self.m_textCtrl1 = wx.TextCtrl( self, wx.ID_ANY,
wx.EmptyString, wx.DefaultPosition, wx.DefaultSize, 0 )
        fgSizer1.Add( self.m_textCtrl1, 0, wx.ALL|wx.EXPAND, 5 )

        self.m_staticText3 = wx.StaticText( self, wx.ID_ANY,
u"Email:", wx.DefaultPosition, wx.DefaultSize, 0 )
        self.m_staticText3.Wrap( -1 )
        self.m_staticText3.SetFont( wx.Font( 12, wx.FONTFAMILY_SWISS,
wx.FONTSTYLE_ITALIC, wx.FONTWEIGHT_NORMAL, False, "Arial" ) )

        fgSizer1.Add( self.m_staticText3, 0, wx.ALL, 5 )

        self.m_textCtrl2 = wx.TextCtrl( self, wx.ID_ANY,
wx.EmptyString, wx.DefaultPosition, wx.DefaultSize, 0 )
        fgSizer1.Add( self.m_textCtrl2, 0, wx.ALL|wx.EXPAND, 5 )

        self.m_staticText4 = wx.StaticText( self, wx.ID_ANY,
u"Пароль:", wx.DefaultPosition, wx.DefaultSize, 0 )
        self.m_staticText4.Wrap( -1 )
        self.m_staticText4.SetFont( wx.Font( 12, wx.FONTFAMILY_SWISS,
wx.FONTSTYLE_ITALIC, wx.FONTWEIGHT_NORMAL, False, "Arial" ) )

        fgSizer1.Add( self.m_staticText4, 0, wx.ALL, 5 )

        self.m_textCtrl3 = wx.TextCtrl( self, wx.ID_ANY,
wx.EmptyString, wx.DefaultPosition, wx.DefaultSize, wx.TE_PASSWORD )
        fgSizer1.Add( self.m_textCtrl3, 0, wx.ALL|wx.EXPAND, 5 )

        bSizer3.Add( fgSizer1, 1, wx.ALL|wx.EXPAND, 30 )
        fgSizer1.AddGrowableCol(1,1)
        bSizer4 = wx.BoxSizer(wx.HORIZONTAL)

        self.m_button23 = wx.Button(self, wx.ID_ANY, u"Назад",
wx.DefaultPosition, wx.Size(120, 35), 0)
        self.m_button23.SetFont(wx.Font(12, wx.FONTFAMILY_SWISS,
wx.FONTSTYLE_ITALIC, wx.FONTWEIGHT_NORMAL, False, "Arial"))
        self.m_button23.SetBackgroundColour(wx.Colour(205, 205, 205))

        bSizer4.Add(self.m_button23, 1, wx.ALIGN_CENTER | wx.RIGHT,
30)
```

```
        self.m_button3 = wx.Button(self, wx.ID_ANY, u"Вхід",
wx.DefaultPosition, wx.Size(120, 35), 0)
        self.m_button3.SetFont(wx.Font(12, wx.FONTFAMILY_SWISS,
wx.FONTSTYLE_ITALIC, wx.FONTWEIGHT_NORMAL, False, "Arial"))
        self.m_button3.SetBackgroundColour(wx.Colour(205, 205, 205))

        bSizer4.Add(self.m_button3, 1, wx.ALIGN_CENTER | wx.LEFT, 30)
        bSizer3.Add(bSizer4, 1, wx.ALL | wx.EXPAND, 20)

        self.SetSizer( bSizer3 )
        self.Layout()
        self.Centre( wx.BOTH )

        self.m_button3.Bind(wx.EVT_BUTTON,
self.on_registration_click)
        self.m_button23.Bind(wx.EVT_BUTTON, self.m_button23_clk)

    def on_registration_click(self, event):
        name_user = self.m_textCtrl1.GetValue()
        email_user = self.m_textCtrl2.GetValue()
        password_user = self.m_textCtrl3.GetValue()

        # Перевірка заповності всіх полів
        if not name_user or not email_user or not password_user:
            wx.MessageBox("Будь ласка, заповніть всі поля!",
"Помилка", wx.OK | wx.ICON_ERROR)
            return  # Завершення виконання методу, якщо є пусті поля

        conn = connect_to_mysql()

        # Перевірка наявності користувача з вказаними логіном, email
та паролем
        if check_duplicate_user(conn, name_user, email_user,
password_user):
            wx.MessageBox("Користувач з такими даними вже існує!",
"Помилка", wx.OK | wx.ICON_ERROR)
        else:
            # Додавання користувача в базу даних
            add_user(conn, name_user, email_user, password_user)
            wx.MessageBox("Реєстрація проведена успішно!", "Успіх",
wx.OK | wx.ICON_INFORMATION)
            self.open_personal_cabinet(name_user)

        if conn:
            conn.close()

    # Функція для відкриття особистого кабінету з передачею імені
користувача
    def open_personal_cabinet(self, name_user):
        self.Destroy()
        frame3 = MyFrame4(None, name_user)
        frame3.Show()
```

```
    def m_button23_clk(self, event):  # Перехід до початкового
фрейму
        self.Destroy()
        frame27 = MyFrame1(None)
        frame27.Show()
        event.Skip()


# Функція для перевірки наявності користувача з вказаними
логіном,email та паролем
def check_duplicate_user(conn, name_user, email_user,
password_user):
    if not conn:
        print("Немає з'єднання з базою даних")
        return False

    cursor = conn.cursor() # Створення курсору для виконання запитів
до бази даних
    query = "SELECT * FROM ticket_users WHERE name_user = %s OR
email_user = %s OR password_user = %s"
    cursor.execute(query, (name_user, email_user, password_user))
    duplicate_user = cursor.fetchone() # Отримання результату запиту
    cursor.close() # Закриття курсору

    if duplicate_user: # Перевірка наявності знайденого користувача
        return True
    else:
        return False


# Функція для додавання користувача в базу даних
def add_user(conn, name_user, email_user, password_user):
    if not conn:
        print("Немає з'єднання з базою даних")
        return False

    cursor = conn.cursor()

    # Додавання користувача в базу даних
    query = "INSERT INTO ticket_users (name_user, email_user,
password_user) VALUES (%s, %s, %s)"
    cursor.execute(query, (name_user, email_user, password_user))
    conn.commit() # Фіксація, тобто збереження зміни в базі даних
    cursor.close()

    return True

app = wx.App()
frame = MyFrame1(None)
frame.Show()
app.MainLoop()
```

Додаток В. Копії документів про апробацію результатів роботи

СТВОРЕННЯ КРОСПЛАТФОРМЕНИХ GUI - ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ WXPYTHON ТА MYSQL

Сейрік Юлії Юріївни

студентка 4-го курсу факультету кібербезпеки та інформаційних технологій Національного університету «Одеська юридична академія»

У сучасному світі з досить високим рівнем розвитку інформаційних технологій широким попитом користуються застосунки, які можуть працювати на багатьох різних платформах та операційних системах. Їхня актуальність пояснюється тим, що такі додатки охоплюють більшу аудиторію клієнтів порівняно з тими що є зосередженими лише на одній платформі. Це надає можливість більшій кількості людей почати взаємодіяти з ними.

Крім того, є й інші причини популярності кросплатформних програмних продуктів. Однією з них є те, що розробка та підтримка таких застосунків відбувається значно простіше, оскільки відсутня необхідність створювати окремий код для кожної платформи. Це також дозволяє знизити витрати часу та ресурсів що є без сумнівів великою перевагою.

Кросплатформені GUI - застосунки – це програмні продукти, які мають графічний інтерфейс користувача, і можуть працювати на різних платформах без потреби у створенні окремого коду для кожної з них. Одним з інструментів, який можна використовувати для їх розробки – бібліотека wxPython, яка належить мові програмування Python.

wxPython - це кросплатформний інструментарій графічного інтерфейсу для мови програмування Python. Він дозволяє програмістам Python легко і просто створювати програми з надійним, функціональним графічним інтерфейсом користувача. Він реалізований у вигляді набору модулів розширення Python, які обгортають компоненти графічного інтерфейсу популярної кросплатформної бібліотеки wxWidgets, написаної на C++ [1].

Наразі підтримуваними платформами є Microsoft Windows, Mac OS X та macOS, а також Linux або інші unix - подібні системи з бібліотеками GTK2 або GTK3 [1].

Ця бібліотека має перевагу у виборі не лише завдяки можливості використання програм, що розроблені з її допомогою, на різних операційних системах, а й через широкий вибір різних інструментів, для створення будь-якого дизайну інтерфейсу. Крім того, вона не складна у використанні і має досить зрозумілу і докладну документацію.

Більшість GUI - застосунків використовують клієнт-серверну модель архітектури, яка складається з клієнтської частини (інтерфейсу користувача) та серверної частини, що забезпечує взаємодію між клієнтом та базою даних через запити до неї.

Однією з баз даних, яка може бути вигідним вибором для використання у розробці кросплатформених GUI – застосунків, є реляційна. В першу чергу реляційна база даних має підтримку СУБД на різних платформах. Окрім того вона може працювати за об'єктно-орієнтованою моделлю, що особливо актуально при застосуванні мови програмування Python. Також вона використовує запити SQL, для роботи з даними, що робить її незалежною від платформи, а також дозволяє використовувати однин і той самий запит багато разів у коді.

Мова структурованих запитів (скор. “SQL” від англ. “Structured Query Language”) - це стандартна мова запитів, яка використовується для роботи з реляційними базами даних. SQL використовується для: створення баз даних, створення таблиць в базі даних, читання даних з таблиць, вставки даних в таблиці, оновлення даних в таблиці, видалення даних з таблиці, видалення таблиць бази даних, видалення баз даних, та інших операцій [2].

MySQL – це система управління реляційними базами даних. Окрім багатоплатформеності, серед її переваг також можна визначити те що вона є простою у використанні, має високу продуктивність, а також є стабільною та надійною.

Для створення серверної частини, яка займається з'єднанням інтерфейсу користувача та бази даних можна використати драйвер MySQL Connector/Python.

MySQL Connector/Python дозволяє програмам на Python отримувати доступ до баз даних MySQL за допомогою API, що відповідає специфікації Python Database API Specification v2.0 [3].

MySQL Connector/Python включає підтримку:

– Майже всіх функцій, що надаються сервером MySQL версії 8.0 і вище.

– X DevAPI.

– Перетворення значень параметрів між типами даних Python і MySQL, наприклад, Python datetime і MySQL DATETIME.

– Всіх розширень MySQL до стандартного синтаксису SQL.

– Стиснення протоколу, що дозволяє стискати потік даних між клієнтом і сервером.

– Підключення за допомогою TCP/IP-сокетів, а в Unix - за допомогою Unix-сокетів.

– Безпечних TCP/IP-з'єднань за допомогою SSL.

– Самостійний драйвер. Connector/Python не потребує клієнтської бібліотеки MySQL або будь-яких модулів Python за межами стандартної бібліотеки [3].

Інструменти для створення багатоплатформених додатків, такі як база даних MySQL та драйвер MySQL Connector/Python мають досить велику та докладну документацію, яка роз'яснює всі аспекти цих засобів у доступній формі.

Кросплатформені GUI – застосунки розроблені з використанням бібліотеки wxPython, бази даних MySQL та драйверу MySQL Connector/Python мають численні вагомі переваги завдяки яким користувачі можуть легко застосовувати подібні програмні продукти.

Список використаних джерел

1. Overview of wxPython. wxPython. URL: http://surl.li/ttdwj

2. Що таке SQL та Бази даних? Acode. URL: https://acode.com.ua/sql-intro/

3. Chapter 1 Introduction to MySQL Connector/Python. MySQL Connectors Documentation. URL: http://surl.li/ttdvb

Ключові слова: платформа, застосунок, GUI, wxPython, MySQL, драйвер.

Keywords: platform, application, GUI, wxPython, MySQL, driver.

Науковий керівник: доцент Лобода Ю.Г.