# Зміст

# 

[**Зміст**](#_8kx3j7bznq1g) **1**

[**Вступ**](#_fclejtut2lk) **2**

[**Розділ 1. Постановка задачі**](#_iupybs4jjy46) **4**

[1.1 Опис інформаційної сутності задачі](#_gmjmov76l8qa) 4

[1.2 Специфікація вимог до програми](#_1zzt10ky9wwb) 5

[**Розділ 2. Аналіз предметної області**](#_2wflwt7gneut) **8**

[2.2 Реферативний опис театральної каси](#_7ldplp7ze2sb) 8

[2.3 Діаграма прецедентів системи](#_izd1s0os2mm3) 9

[2.3.1 Опис сценаріїв використання](#_tpg6c0vth2z9) 9

[2.3.2 Побудова діаграми прецедентів](#_q9n9j48kr75u) 10

[2.4 Сценарій роботи програми](#_twq9zagtry2q) 11

[2.4.1 Запуск програми](#_5zpkh1jdgx2) 11

[2.4.3 Деталі вистави](#_e5j7h2aovlcd) 12

[2.4.4 Управління квитками](#_t1usl8is1f2m) 13

[2.4.5 Управління системою](#_tvs5kdxtotqb) 13

[2.4.6 Вихід з програми](#_7cjie2c2oksq) 14

[**Розділ 3. Проектування архітектури програмної системи**](#_iztn349r19do) **15**

[3.1 Опис взаємодії між класами](#_s2eqj1km65bc) 15

[3.2 Візуалізація UML діаграми](#_kpv14itc2qrp) 16

[**Розділ 4. Опис програмного забезпечення**](#_d84rypbtkjnx) **18**

[4.1 Таблиця опису класів та їх методів](#_lh29di29pq3m) 18

[4.1.1 Клас Performance](#_lgxmuqs9ggff) 18

[4.1.2 Клас PerformanceManager](#_e5m3dz7gf35j) 19

[**Розділ 5. Тестування програмної системи**](#_oyowvw1rfy9d) **21**

[5.1 Методика тестування](#_lu5jjvjdu1u8) 21

[5.2 План тестування](#_cm0kv7g7gs75) 21

[5.3 Результати тестування](#_fcxmxib7m4rz) 24

[**Розділ 6. Інструкція користувача**](#_pynbpebkwsja) **25**

[6.1 Призначення програми](#_qoxymuptncl3) 25

[6.2 Вимоги до системи](#_stgifvdr8ocv) 25

[6.3 Склад програмного забезпечення](#_gkgmpxb51ut0) 25

[6.4 Опис інтерфейсу](#_armrvki8selv) 26

[6.5 Послідовність дій для виконання функцій програми](#_g6db6b43big) 27

[**Висновок**](#_3pos6rkm8zga) **28**

[**Додаток A**](#_g7qwt6fxcl9o) **29**

# 

# 

# Вступ

У сучасному світі програмування відіграє ключову роль у розвитку технологій і автоматизації процесів у різноманітних галузях діяльності. Курс "Основи програмування" призначений для засвоєння студентами фундаментальних принципів створення програмного забезпечення і реалізації об'єктно-орієнтованих рішень у практичних задачах. Головна мета даної курсової роботи — не тільки навчити основам мови програмування, але й демонструвати, як ці знання можуть бути застосовані для розробки реальних проектів, які вирішують специфічні задачі.

Курсова робота зосереджується на створенні системи продажу квитків для театру, що дозволяє користувачам ефективно управляти розкладом вистав, бронюванням, продажем та відміною квитків. Система повинна надавати інтуїтивно зрозумілий інтерфейс і бути адаптована до потреб кінцевих користувачів, що вимагає глибокого розуміння як технічних аспектів розробки, так і бізнес-вимог клієнтів.

Ця курсова робота також ставить за мету вивчити можливості мови програмування C++, зокрема, її об'єктно-орієнтовані особливості, які є найбільш корисними при створенні модульних і масштабованих аплікацій. Студенти навчаться проектувати класи та об'єкти, які ефективно моделюють сутність реального світу театральної каси, вирішуючи такі задачі, як керування запасами квитків, обробка транзакцій та забезпечення даних для звітів.

Крім технічних аспектів, важливою частиною роботи є аналіз взаємодії системи з кінцевими користувачами. Буде проведений огляд зручності інтерфейсів, оцінка можливостей системи забезпечити швидке та точне обслуговування користувачів, а також розроблені рекомендації щодо покращення користувацького досвіду. Це включає в себе дослідження щодо оптимізації процесів бронювання та купівлі квитків, щоб забезпечити максимальну ефективність та задоволення потреб клієнтів.

# Розділ 1. Постановка задачі

## 1.1 Опис інформаційної сутності задачі

Ця частина курсової роботи присвячена детальному аналізу інформаційної структури системи продажу квитків у театр, яка була розроблена на мові програмування C++ з використанням принципів об'єктно-орієнтованого програмування (ООП). Головна мета системи — автоматизувати процеси бронювання, продажу, та управління квитками для різноманітних вистав, що значно спрощує адміністрування театральної каси та покращує досвід користувачів.

Основні сутності системи:

1. Вистава (Performance): Центральний об'єкт системи, який представляє театральну виставу. Для кожної вистави зберігається наступна інформація:

Назва вистави

Автор

Жанр

Дата і час проведення

Доступні квитки, розподілені за цінами

Об'єкти класу Performance відповідають за зберігання інформації про вистави та управління статусами квитків (доступні, зарезервовані, продані).

2. Керування виставами (PerformanceManager): Клас, який використовується для управління списком вистав. Цей клас надає функції для додавання, видалення, пошуку та управління виставами. Він також відповідає за взаємодію з користувачем через консольне меню, де можна вибрати різні операції пов'язані з виставами.

3. Квитки (Tickets): Хоча конкретний клас для квитків не був створений окремо, управління квитками інтегровано у клас Performance. Кожен квиток має ціну та статус (доступний, заброньований, проданий), що дозволяє системі трекати доступність та вести продажі ефективно.

Основні операції системи:

Пошук вистав: користувачі можуть шукати вистави за автором, назвою, жанром, та датою.

Управління квитками: включає продаж квитків, резервування квитків, відміну резервації та підтвердження резервації.

Аналіз вхідних і вихідних даних:

Вхідні дані: команди від користувача через консольне меню (наприклад, вибір вистави для пошуку, введення даних для резервації квитка).

Вихідні дані: результати пошуку вистав, статус операцій з квитками, інформація про доступні/зарезервовані/продані квитки.

Ця структура та функціональність системи була розроблена з метою максимально автоматизувати процес управління театральною касою, забезпечити зручність та ефективність обслуговування відвідувачів, а також підвищити точність обліку квитків.

## 1.2 Специфікація вимог до програми

Специфікація вимог є ключовим компонентом у процесі розробки програмного забезпечення, який визначає очікувані функції та обмеження системи. Для системи продажу квитків у театр, ці вимоги складають основу для розробки, тестування та подальшого впровадження. Нижче представлені детальні вимоги, розділені на функціональні та нефункціональні компоненти.

Функціональні вимоги

1. Управління виставами:

Система повинна дозволяти адміністратору додавати нові вистави, включаючи інформацію про назву, автора, жанр, дату та час проведення, а також первинні дані про квитки (ціни та кількість).

Можливість редагувати інформацію про існуючі вистави, включаючи всі аспекти, від назви до деталей квитків.

Функція видалення вистав з системи, яка повинна також враховувати видалення всіх пов'язаних з ними квитків.

Забезпечення пошуку вистав за різними критеріями, такими як назва, автор, жанр, дата.

2. Управління квитками:

Можливість продажу квитків, включаючи вибір конкретного місця та оновлення бази даних про доступність квитків.

Система має дозволяти резервування квитків з подальшою можливістю їх підтвердження або скасування.

Отримання звітів про статус квитків (продані, доступні, зарезервовані).

3. Звітність та аналітика:

Автоматичне генерування звітів про продажі та резервації за обраними періодами.

Функції для виведення статистики за різними параметрами, що допомагає у плануванні маркетингових заходів і просуванні вистав.

Нефункціональні вимоги

1. Продуктивність:

Система має обробляти запити користувача максимум за 2 секунди навіть під високим навантаженням.

Оптимізація бази даних для швидкого доступу та зміни інформації.

2. Надійність:

Система повинна забезпечувати високий рівень точності в обробці транзакцій, мінімізуючи помилки в продажах та резерваціях.

Резервне копіювання даних має виконуватись автоматично щодня.

3. Безпека:

Авторизація та аутентифікація користувачів для різних рівнів доступу: адміністратори, касири та менеджери.

Захист від несанкціонованого доступу до даних та їх зміни.

4. Інтерфейс та взаємодія з користувачем:

Інтуїтивно зрозумілий графічний інтерфейс для адміністраторів і касирів, що мінімізує потребу в навчанні користувачів.

Система повинна підтримувати базові функції для взаємодії з користувачами через консоль та/або графічний інтерфейс.

# Розділ 2. Аналіз предметної області

## 2.2 Реферативний опис театральної каси

Театральна каса є ключовим елементом у системі управління театром, відповідальним за продаж і бронювання квитків на різноманітні культурні заходи. Вона виступає як місток між виставами театру та аудиторією, забезпечуючи доступність і зручність придбання квитків для широкого кола глядачів. З розвитком технологій театральні каси стали інтегрувати різноманітні цифрові платформи для покращення сервісу, включаючи онлайн-продажі, автоматизовані кіоски у фойє театрів, а також мобільні додатки для замовлення квитків.

Сучасні театральні каси мають вирішувати кілька критичних завдань, які є фундаментальними для ефективного функціонування театру:

1. Продаж квитків: Продаж квитків є основною діяльністю каси. Система повинна забезпечувати швидкий і надійний продаж квитків як в офлайн-так і в онлайн-режимах. Це включає прийом платежів різними способами, такими як готівка, банківські картки та цифрові платежі.

2. Бронювання квитків: Бронювання квитків дозволяє користувачам зарезервувати місця заздалегідь. Система має враховувати статус місць (заброньовані, продані, вільні) і дозволяти касирам та користувачам онлайн переглядати актуальну інформацію щодо доступності місць.

3. Обробка повернень та обмінів: Важливою функцією каси є можливість обробки повернень і обмінів квитків. Це вимагає гнучкої системи управління, яка могла б швидко вирішувати такі питання без значних збитків для театру.

4. Звітність та аналітика: Для планування маркетингових заходів та фінансового менеджменту необхідна комплексна система звітності. Театральна каса має вести облік усіх транзакцій, включаючи продажі, бронювання, повернення, що дозволяє аналізувати продажі, попит на вистави та відвідуваність.

5. Клієнтський сервіс: Надання високоякісного сервісу клієнтам є важливим для забезпечення хорошого враження від відвідування театру. Це включає зручність бронювання квитків, чіткість інформації про вистави, доступність служби підтримки та зручність використання онлайн-сервісів.

Основною технологічною вимогою до системи театральної каси є її здатність інтегруватися з іншими системами управління театром, такими як системи обліку та CRM-системи (Customer Relationship Management), що сприяє збору даних про клієнтів та їхні переваги для подальшого маркетингового аналізу та персоналізації пропозицій.

## 2.3 Діаграма прецедентів системи

### 2.3.1 Опис сценаріїв використання

Діаграма прецедентів (use case diagram) є ключовим елементом у визначенні взаємодій користувачів із системою та основних функціональностей, які система повинна підтримувати. В контексті системи продажу квитків для театру, основними користувачами є адміністратори, касири та клієнти. Нижче наведено сценарії використання для кожної з цих ролей.

1. Клієнти:

* Пошук вистав: Клієнти можуть шукати вистави за різними критеріями (автор, назва, жанр, дата).
* Бронювання квитків: Клієнти можуть бронювати квитки на вибрані вистави.
* Купівля квитків: Клієнти можуть купувати квитки на вибрані вистави.

2. Касири:

* Продаж квитків: Касири виконують продаж квитків клієнтам.
* Обробка повернень квитків: Касири обробляють запити на повернення квитків.
* Реєстрація резервацій: Касири реєструють бронювання квитків, внесені клієнтами.

3. Адміністратори:

* Управління виставами: Адміністратори додають, редагують або видаляють інформацію про вистави.
* Управління користувачами: Адміністратори відповідають за додавання та управління ролями користувачів системи.
* Звітність: Адміністратори генерують звіти по продажах, бронюваннях і відвідуваннях.

### 2.3.2 Побудова діаграми прецедентів

* Актор "Клієнт" зв'язаний із прецедентами "Пошук вистав", "Бронювання квитків", "Купівля квитків".
* Актор "Касир" зв'язаний із прецедентами "Продаж квитків", "Обробка повернень квитків", "Реєстрація резервацій".
* Актор "Адміністратор" зв'язаний із прецедентами "Управління виставами", "Управління користувачами", "Звітність".

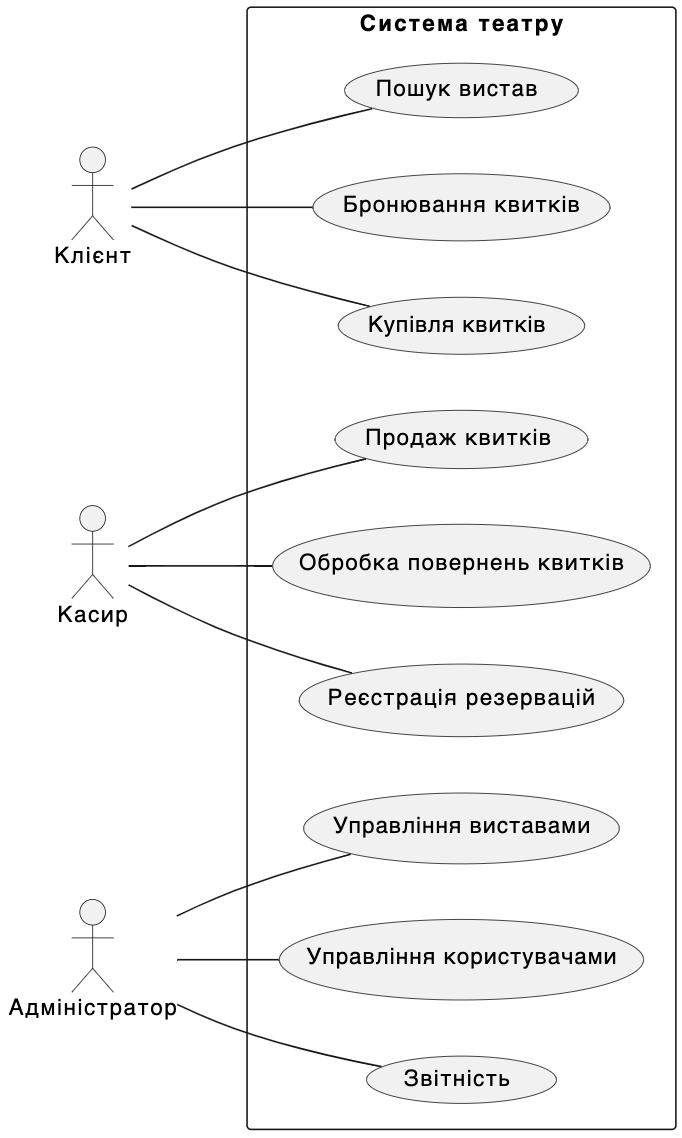


Рисунок 2.1 Діаграма прецедентів

## 2.4 Сценарій роботи програми

### 2.4.1 Запуск програми

Коли програма запускається, користувачу спочатку відображається головне меню, з якого можна обрати різноманітні опції управління виставами, квитками, та пошуку інформації. Головне меню має наступні пункти:

1. Пошук вистав

2. Деталі вистави

3. Управління квитками

4. Управління системою

5. Вихід

### 2.4.2 Пошук вистав

За обрання пункту "Пошук вистав" користувача перенаправляють до меню пошуку, де можна шукати вистави за автором або за назвою. Меню пошуку включає:

1. Пошук за автором

2. Пошук за назвою

3. Повернення до головного меню

Користувач вводить ім'я автора або назву вистави, і програма відображає всі доступні результати, або повідомлення про те, що вистави не знайдено.

### 2.4.3 Деталі вистави

При виборі "Деталі вистави" користувачу пропонується ввести назву вистави, інформацію про яку хоче переглянути. Програма знаходить виставу та відображає деталі, такі як назва, автор, жанр, дата проведення та інформацію про доступні, заброньовані та продані квитки. Якщо виставу не знайдено, користувачу відображається відповідне повідомлення.

### 2.4.4 Управління квитками

У розділі "Управління квитками" користувач може переглянути та управляти статусом квитків для конкретної вистави. Спочатку користувач вводить назву вистави, для якої хоче управляти квитками. Програма відображає деталі квитків і надає наступні можливості:

1. Продаж квитка

2. Бронювання квитка

3. Скасування бронювання

4. Підтвердження бронювання

5. Повернення до попереднього меню

Кожна дія вимагає введення ціни квитка і підтвердження дії. Програма відображає статус операції (успішно/неуспішно).

### 2.4.5 Управління системою

Цей розділ дозволяє користувачу вносити зміни до вистав, які вже є в системі, а також додавати нові вистави. Опції меню:

1. Додати нову виставу

2. Редагувати існуючу виставу

3. Видалити виставу

4. Оновити ціни на квитки

5. Повернення до головного меню

Для кожної опції користувач вводить відповідні дані, наприклад, назву, автора, жанр, дату і ціни квитків для нової вистави. Програма підтверджує виконання дії або видає помилку, якщо операція не може бути виконана.

### 2.4.6 Вихід з програми

При виборі "Вихід" програма завершує роботу, і користувач виходить з системи. Перед виходом програма може запитати підтвердження, щоб уникнути випадкового закриття.

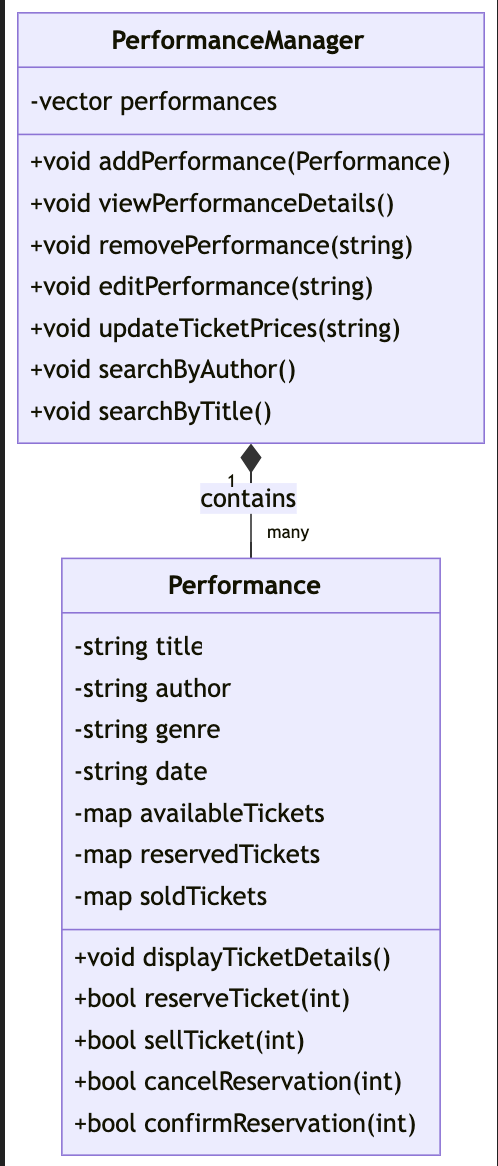
# Розділ 3. Проектування архітектури програмної системи

Діаграма класів UML (Unified Modeling Language) використовується для візуалізації структури програмної системи, зокрема для зображення класів, їхніх атрибутів, методів, а також взаємин між ними.

## 3.1 Опис взаємодії між класами

1. Клас Performance є основним класом, який зберігає всю інформацію про конкретну виставу, включаючи квитки та їх статуси. Він також включає методи для маніпулювання статусами квитків (продаж, резервація, скасування, підтвердження).
2. Клас PerformanceManager управляє колекцією performances, яка зберігає всі вистави. Цей клас відповідає за додавання, видалення, редагування вистав, а також за пошук вистав за різними критеріями.

## 3.2 Візуалізація UML діаграми



Діаграма має відображати класи з їх атрибутами та методами, а також зв'язки між ними, такі як асоціації та агрегації. Зокрема, клас PerformanceManager агрегує кілька екземплярів класу Performance, відображаючи відносини "один до багатьох".

Кожен клас представляється як прямокутник, поділений на три розділи: назва класу, атрибути класу, та методи класу. Зв'язки між класами позначаються лініями, зі стрілками або без, в залежності від типу відносин.

# Розділ 4. Опис програмного забезпечення

## 4.1 Таблиця опису класів та їх методів

### 4.1.1 Клас Performance

На основі зображення, яке ви надали, ви хочете, щоб опис класів був організований у вигляді таблиці з певними колонками. Ось як може виглядати заповнена таблиця для класу Performance з вашої програми:

| № п/п | Назва класу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Performance | getTitle | Повертає назву вистави | Немає | string: назва вистави | Performance.hpp |
| 2 | Performance | getAuthor | Повертає автора вистави | Немає | string: автор вистави | Performance.hpp |
| 3 | Performance | getGenre | Повертає жанр вистави | Немає | string: жанр вистави | Performance.hpp |
| 4 | Performance | getDate | Повертає дату вистави | Немає | string: дата вистави | Performance.hpp |
| 5 | Performance | displayTicketDetails | Відображає деталі квитків | Немає | void | Performance.hpp |
| 6 | Performance | reserveTicket | Резервує квиток | int: ціна квитка | bool: успішність операції | Performance.hpp |
| 7 | Performance | sellTicket | Продає квиток | int: ціна квитка | bool: успішність операції | Performance.hpp |
| 8 | Performance | cancelReservation | Скасовує резервацію квитка | int: ціна квитка | bool: успішність операції | Performance.hpp |
| 9 | Performance | confirmReservation | Підтверджує резервацію квитка | int: ціна квитка | bool: успішність операції | Performance.hpp |

Ця таблиця забезпечує чіткий та структурований опис кожного методу класу Performance, включаючи інформацію про призначення, параметри, і повернення. Такий формат документації може бути використаний для кожного класу у вашій системі, що забезпечує ефективне посилання і розуміння для розробників та аналітиків системи.

### 4.1.2 Клас PerformanceManager

Для класу PerformanceManager з вашої програми можна створити аналогічну таблицю, яка описує його методи та атрибути:

| **№** | **Назва класу** | **Назва методу** | **Призначення методу** | **Опис вхідних параметрів** | **Опис вихідних параметрів** | **Заголовний файл** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | PerformanceManager | addPerformance | Додає нову виставу до списку | Performance: вистава для додавання | void | PerformanceManager.hpp |
| 2 | PerformanceManager | viewPerformanceDetails | Відображає деталі вистави | засновані на назві | Немає (діє через ввод користувача) | PerformanceManager.hpp |
| 3 | PerformanceManager | removePerformance | Видаляє виставу зі списку за назвою | string: назва вистави | void | PerformanceManager.hpp |
| 4 | PerformanceManager | editPerformance | Редагує деталі існуючої вистави | string: назва вистави | void | PerformanceManager.hpp |
| 5 | PerformanceManager | updateTicketPrices | Оновлює ціни квитків для вистави | string: назва вистави | void | PerformanceManager.hpp |
| 6 | PerformanceManager | searchByAuthor | Відображає всі вистави за автором | Немає (діє через ввод користувача) | void | PerformanceManager.hpp |
| 7 | PerformanceManager | searchByTitle | Відображає вистави за назвою | Немає (діє через ввод користувача) | void | PerformanceManager.hpp |
| 8 | PerformanceManager | viewAndManageTickets | Перегляд та управління квитками для вистави | Немає (діє через ввод користувача) | void | PerformanceManager.hpp |

Ця таблиця відображає ключові методи класу PerformanceManager, які забезпечують різні аспекти управління виставами та квитками в системі. Кожен метод детально описаний з вказівкою його призначення та параметрів, що допомагає зрозуміти його роль і функціональність в контексті програми.

# Розділ 5. Тестування програмної системи

## 5.1 Методика тестування

Тестування програмної системи для управління театральними виставами є ключовим компонентом забезпечення її надійності та ефективності. Методика тестування включає як ручне, так і автоматизоване тестування для перевірки функціональності, продуктивності, безпеки та користувацького інтерфейсу.

Ручне тестування фокусується на перевірці користувацького досвіду та відповідності функціональних вимог через безпосередній взаємодію користувачів з програмою.

Автоматизоване тестування використовує спеціалізоване програмне забезпечення для виконання повторюваних завдань та перевірки коректності обробки даних. Воно дозволяє покрити більш широкий спектр сценаріїв та гарантувати стабільність програми під час рефакторингу та розширення функціоналу.

## 5.2 План тестування

Об'єкти тестування:

1. Інтерфейс користувача.

2. Функціональність управління виставами.

3. Функціональність управління квитками.

4. Пошук вистав за автором та назвою.

5. Збереження та зчитування даних.

Тестові набори та очікувані результати:

1. Тестування функціональності вистав:

* Додавання нової вистави.

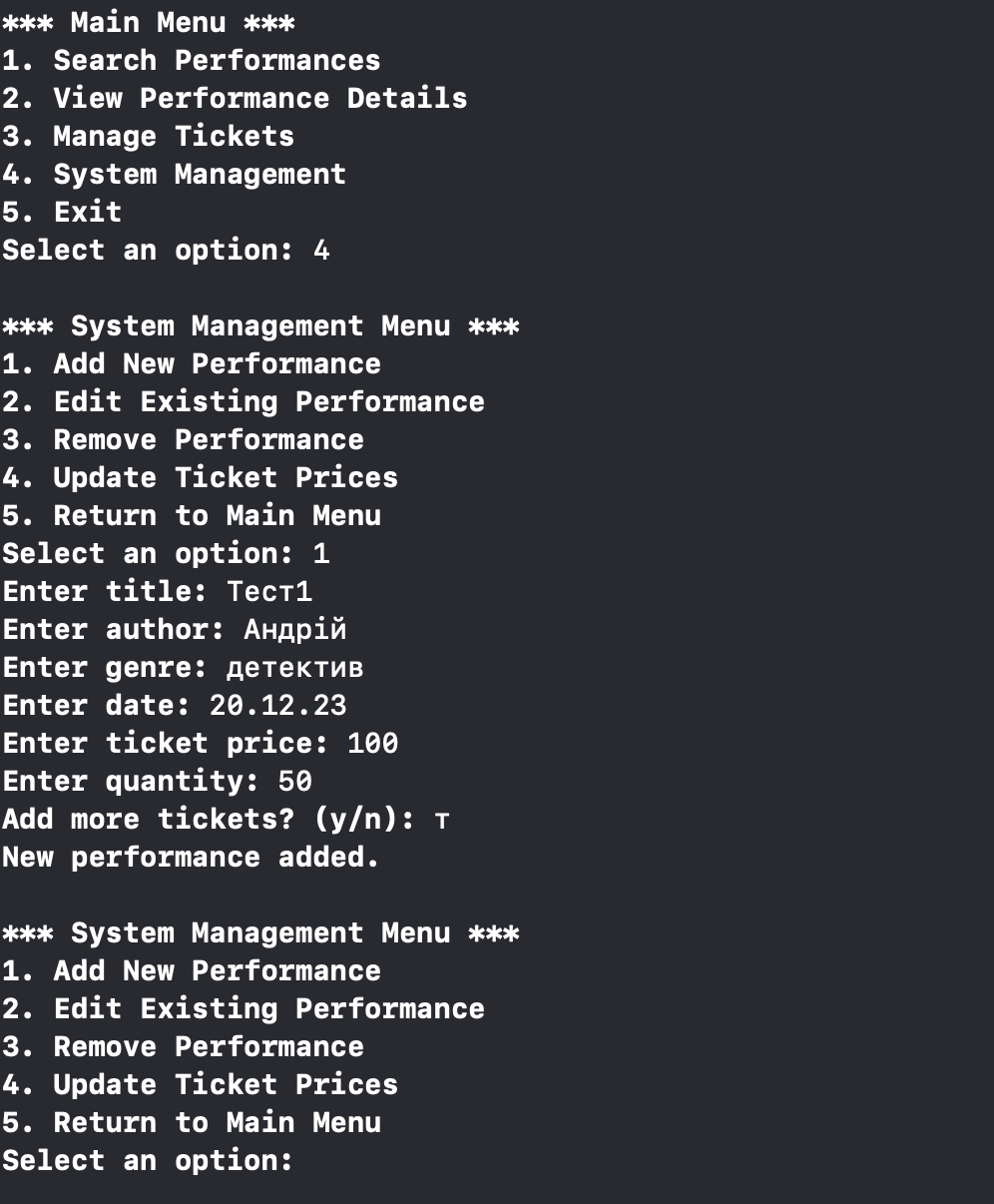


Рисунок 5.1 Додавання нової вистави

* Видалення вистави.

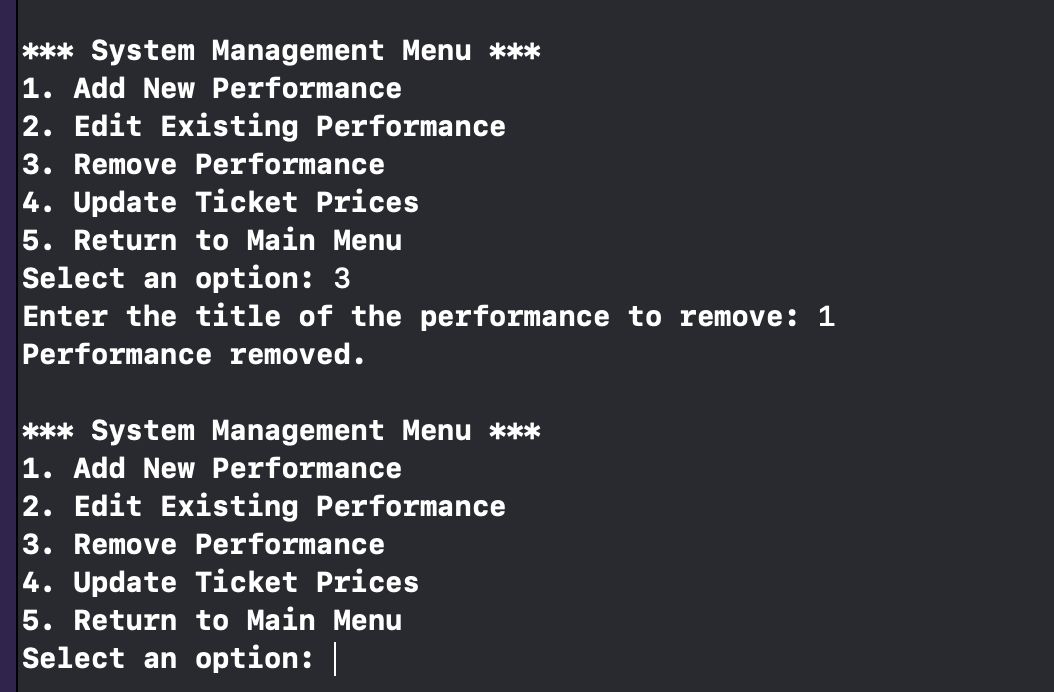


Рисунок 5.2 Видалення нової вистави

* Очікувані результати: дії виконуються без помилок, дані зберігаються коректно.

2. Тестування квитків:

* Продаж, бронювання, скасування та підтвердження квитків.

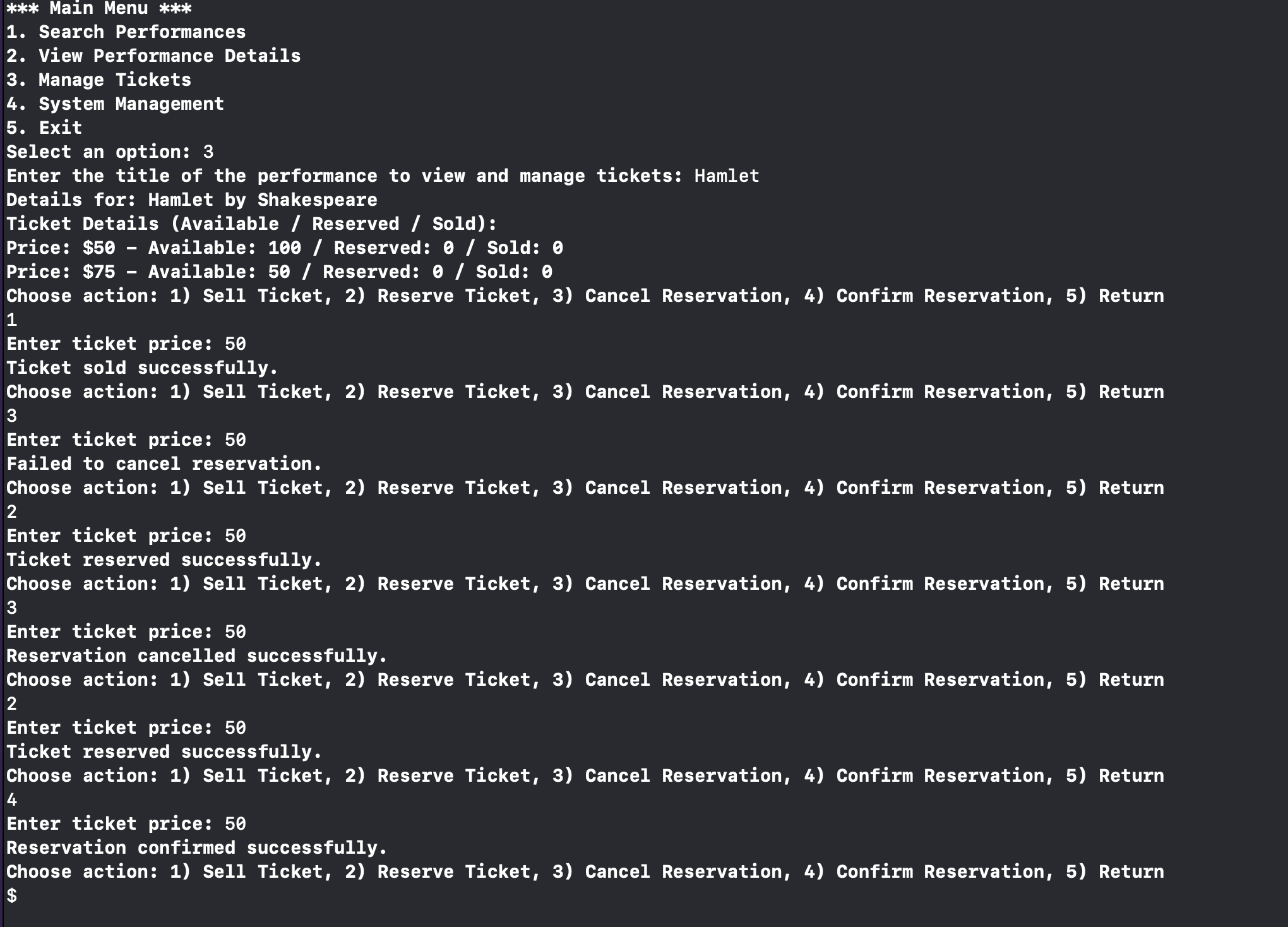


Рисунок 5.3 Продаж, бронювання, скасування та підтвердження квитків.

* Очікувані результати: зміни відображаються в системі відповідно до очікуваних.

3. Тестування пошуку:

* Пошук вистав за автором і назвою.

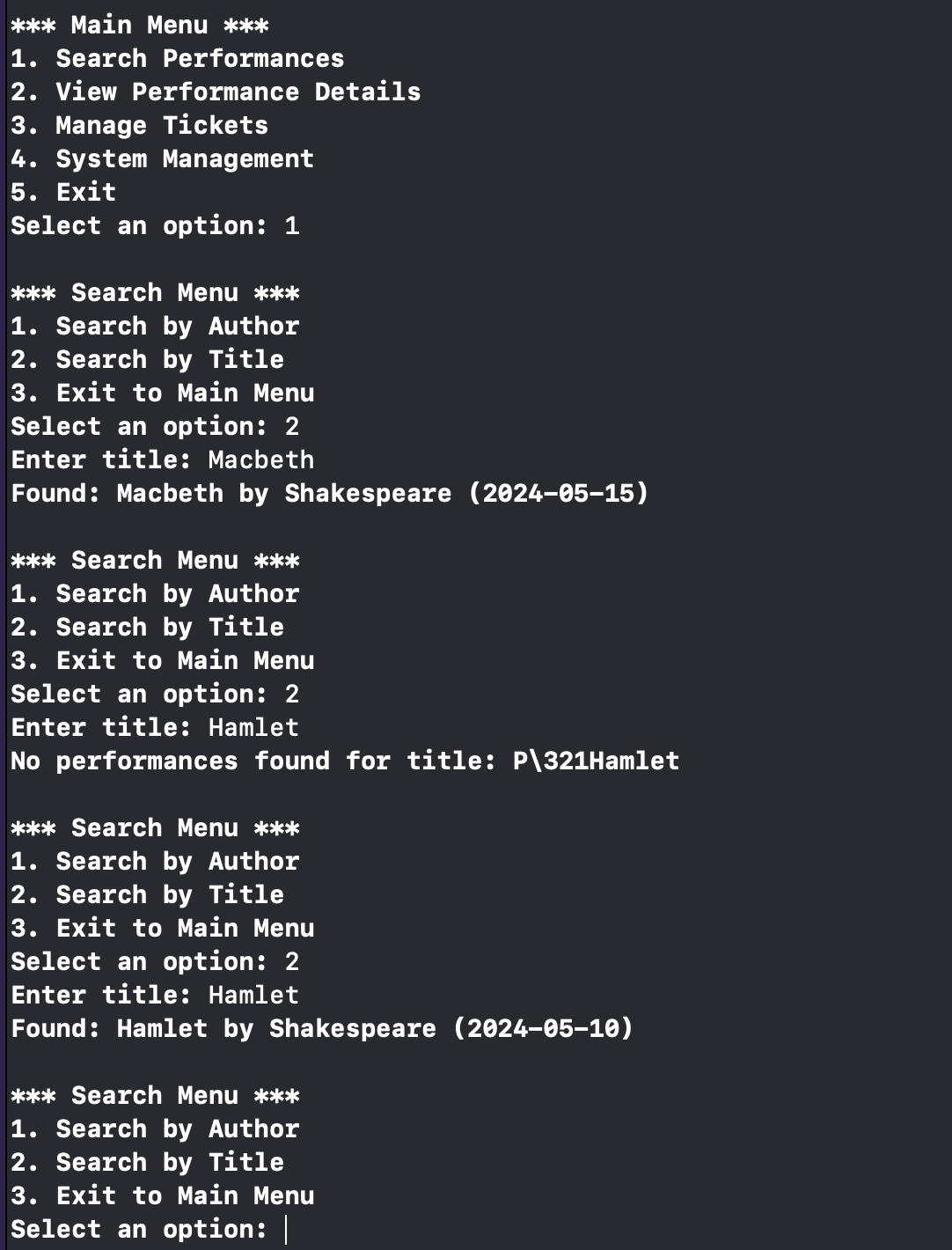


Рисунок 5.4 Пошук вистав за автором і назвою.

* Очікувані результати: список відповідних вистав відображається відповідно до запиту.

## 5.3 Результати тестування

* Тестування функціональності управління виставами: Всі операції з додавання, редагування та видалення були успішно протестовані.
* Тестування управління квитками: Операції з квитками виконувалися згідно з очікуваннями, без помилок.

# Розділ 6. Інструкція користувача

## 6.1 Призначення програми

Програма для управління театральними виставами створена для зручності керування процесом продажу квитків, резервації, ведення репертуару театру та контролю за доступністю місць. Вона дозволяє театральним касам і адміністраторам організувати інформацію про вистави, їхні дати та авторів, а також керувати продажем і бронюванням квитків.

## 6.2 Вимоги до системи

* Для стабільної роботи програми необхідно:
* Операційна система: Windows 10 або новіше, macOS 10.13 або новіше.
* Процесор: мінімум 2.0 GHz двоядерний.
* RAM: мінімум 4 GB.
* Місце на жорсткому диску: мінімум 500 MB вільного простору.
* Наявність Java Runtime Environment версії 8 або новіше.

## 6.3 Склад програмного забезпечення

Програма складається з наступних основних файлів:

* main.cpp (250 MB): Виконавчий файл програми.

## 6.4 Опис інтерфейсу

Програма має графічний користувацький інтерфейс, який дозволяє легко керувати всіма функціями. На зображенні, наданому користувачем, видно основні елементи інтерфейсу:

1. Головне меню:

* Search Performances: Пошук вистав за автором або назвою.
* View Performance Details: Перегляд деталей про конкретну виставу.
* Manage Tickets: Управління квитками на вистави.
* System Management: Керування системними налаштуваннями програми.
* Exit: Вихід з програми.

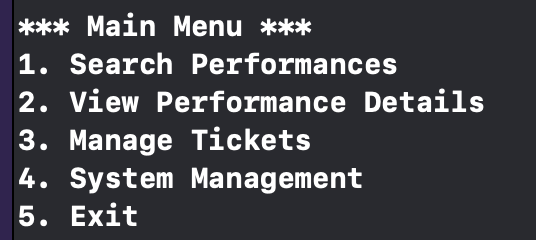


Рисунок 6.1 Головне меню.

2. Search Menu:

* Search by Author: Пошук вистав за автором.
* Search by Title: Пошук вистав за назвою.
* Exit to Main Menu: Повернення до головного меню.

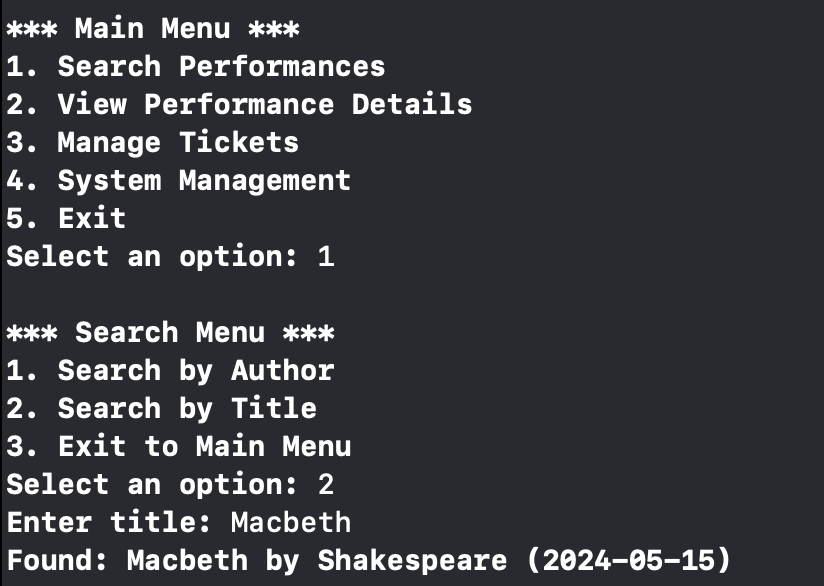


Рисунок 6.2 Меню пошуку вистави.

## 6.5 Послідовність дій для виконання функцій програми

Пошук вистави за назвою:

1. З головного меню вибрати "Search Performances".

2. У відкритому меню пошуку вибрати "Search by Title".

3. Ввести назву вистави в поле вводу та натиснути Enter.

4. Програмавідобразить результати пошуку. Якщо вистава знайдена, вона буде показана разом із датою проведення.

5. Якщо виставу не знайдено, з'явиться відповідне повідомлення.

Кожна взаємодія в програмі супроводжується чіткими інструкціями та повідомленнями про стан виконання операцій, що робить її зручною та доступною для користувачів різного рівня.

# Висновок

Курсова робота, присвячена розробці та аналізу програмної системи для управління театральними виставами, продемонструвала важливість інтеграції сучасних інформаційних технологій у сфері культури та розваг. Розроблена система дозволяє ефективно управляти різними аспектами театрального репертуару, включаючи продаж квитків, бронювання, а також організацію вистав.

Програма виконує важливу роль у підвищенні ефективності роботи театральних кас, спрощуючи багато процесів, які традиційно вимагали значно більше часу та ресурсів. Це стосується не тільки внутрішніх операцій, але й покращення сервісу для відвідувачів. Завдяки інтуїтивно зрозумілому інтерфейсу та автоматизації рутинних завдань, система забезпечує зручність і швидкість обслуговування клієнтів.

Протягом курсової роботи було виконано наступні завдання:

1. Аналіз вимог та потреб користувачів.

2. Проектування архітектури програмної системи.

3. Розробка програмного забезпечення з використанням сучасних технологій програмування.

4. Тестування системи для забезпечення її надійності та ефективності.

5. Розробка інструкції для користувачів для забезпечення легкості використання системи.

# Додаток A

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <map>

class Performance {

private:

std::string title;

std::string author;

std::string genre;

std::string date;

std::map<int, int> availableTickets;

std::map<int, int> reservedTickets;

std::map<int, int> soldTickets;

std::map<int, int> tickets;

public:

Performance(const std::string& title, const std::string& author, const std::string& genre, const std::string& date, const std::map<int, int>& tickets)

: title(title), author(author), genre(genre), date(date), availableTickets(tickets) {}

const std::string& getTitle() const { return title; }

const std::string& getAuthor() const { return author; }

const std::string& getGenre() const { return genre; }

const std::string& getDate() const { return date; }

const std::map<int, int>& getTickets() const { return tickets; }

void displayTicketDetails() const {

std::cout << "Ticket Details (Available / Reserved / Sold):" << std::endl;

for (const auto& pair : availableTickets) {

int reserved = reservedTickets.count(pair.first) ? reservedTickets.at(pair.first) : 0;

int sold = soldTickets.count(pair.first) ? soldTickets.at(pair.first) : 0;

std::cout << "Price: $" << pair.first << " - Available: " << pair.second << " / Reserved: " << reserved << " / Sold: " << sold << std::endl;

}

}

void updateTickets(const std::map<int, int>& newTickets) {

availableTickets = newTickets;

}

bool reserveTicket(int price) {

if (availableTickets[price] > 0) {

availableTickets[price]--;

reservedTickets[price]++;

return true;

}

return false;

}

bool sellTicket(int price) {

if (availableTickets[price] > 0) {

availableTickets[price]--;

soldTickets[price]++;

return true;

}

return false;

}

bool cancelReservation(int price) {

if (reservedTickets.find(price) != reservedTickets.end() && reservedTickets[price] > 0) {

reservedTickets[price]--;

availableTickets[price]++;

return true;

}

return false;

}

bool confirmReservation(int price) {

if (reservedTickets.find(price) != reservedTickets.end() && reservedTickets[price] > 0) {

reservedTickets[price]--;

soldTickets[price]++;

return true;

}

return false;

}

};

class PerformanceManager {

private:

std::vector<Performance> performances;

public:

void addPerformance(const Performance& performance) {

performances.push\_back(performance);

}

void viewPerformanceDetails() {

std::string title;

std::cout << "Enter the title of the performance to view details: ";

std::getline(std::cin, title);

bool found = false;

for (const auto& perf : performances) {

if (perf.getTitle() == title) {

std::cout << "Performance: " << perf.getTitle() << " by " << perf.getAuthor() << " on " << perf.getDate() << std::endl;

perf.displayTicketDetails();

found = true;

break;

}

}

if (!found) {

std::cout << "No performance found with the title: " << title << std::endl;

}

}

void removePerformance(const std::string& title) {

auto it = std::remove\_if(performances.begin(), performances.end(), [&title](const Performance& perf) {

return perf.getTitle() == title;

});

performances.erase(it, performances.end());

std::cout << "Performance removed.\n";

}

void editPerformance(const std::string& title) {

for (auto& perf : performances) {

if (perf.getTitle() == title) {

std::string newTitle, newAuthor, newGenre, newDate;

std::cout << "Enter new title (current: " << perf.getTitle() << "): ";

getline(std::cin, newTitle);

std::cout << "Enter new author (current: " << perf.getAuthor() << "): ";

getline(std::cin, newAuthor);

std::cout << "Enter new genre (current: " << perf.getGenre() << "): ";

getline(std::cin, newGenre);

std::cout << "Enter new date (current: " << perf.getDate() << "): ";

getline(std::cin, newDate);

// Updating the details of the performance

perf = Performance(newTitle.empty() ? perf.getTitle() : newTitle,

newAuthor.empty() ? perf.getAuthor() : newAuthor,

newGenre.empty() ? perf.getGenre() : newGenre,

newDate.empty() ? perf.getDate() : newDate,

perf.getTickets()); // Keeps current tickets as is

std::cout << "Performance updated.\n";

return;

}

}

std::cout << "No performance found with the title: " << title << std::endl;

}

void updateTicketPrices(const std::string& title) {

for (auto& perf : performances) {

if (perf.getTitle() == title) {

std::map<int, int> newPrices;

int price, quantity;

char more = 'y';

while (more == 'y') {

std::cout << "Enter new price: ";

std::cin >> price;

std::cout << "Enter quantity for price $" << price << ": ";

std::cin >> quantity;

newPrices[price] = quantity;

std::cout << "Add more prices? (y/n): ";

std::cin >> more;

}

perf.updateTickets(newPrices);

std::cout << "Ticket prices updated.\n";

return;

}

}

std::cout << "No performance found with the title: " << title << std::endl;

}

void searchByAuthor() {

std::string author;

std::cout << "Enter author's name: ";

std::getline(std::cin, author);

bool found = false;

for (const auto& perf : performances) {

if (perf.getAuthor() == author) {

std::cout << "Found: " << perf.getTitle() << " by " << perf.getAuthor() << " (" << perf.getDate() << ")" << std::endl;

found = true;

}

}

if (!found) {

std::cout << "No performances found for author: " << author << std::endl;

}

}

void searchByTitle() {

std::string title;

std::cout << "Enter title: ";

std::getline(std::cin, title);

bool found = false;

for (const auto& perf : performances) {

if (perf.getTitle() == title) {

std::cout << "Found: " << perf.getTitle() << " by " << perf.getAuthor() << " (" << perf.getDate() << ")" << std::endl;

found = true;

}

}

if (!found) {

std::cout << "No performances found for title: " << title << std::endl;

}

}

void viewAndManageTickets() {

std::string title;

std::cout << "Enter the title of the performance to view and manage tickets: ";

std::getline(std::cin, title);

for (auto& perf : performances) {

if (perf.getTitle() == title) {

std::cout << "Details for: " << perf.getTitle() << " by " << perf.getAuthor() << std::endl;

perf.displayTicketDetails();

manageIndividualTickets(perf);

return;

}

}

std::cout << "No performance found with the title: " << title << std::endl;

}

void manageIndividualTickets(Performance& perf) {

int action;

do {

std::cout << "Choose action: 1) Sell Ticket, 2) Reserve Ticket, 3) Cancel Reservation, 4) Confirm Reservation, 5) Return\n";

std::cin >> action;

std::cin.ignore();

int price;

if (action >= 1 && action <= 4) {

std::cout << "Enter ticket price: ";

std::cin >> price;

std::cin.ignore();

}

switch (action) {

case 1:

if (perf.sellTicket(price)) std::cout << "Ticket sold successfully.\n";

else std::cout << "Failed to sell ticket.\n";

break;

case 2:

if (perf.reserveTicket(price)) std::cout << "Ticket reserved successfully.\n";

else std::cout << "Failed to reserve ticket.\n";

break;

case 3:

if (perf.cancelReservation(price)) std::cout << "Reservation cancelled successfully.\n";

else std::cout << "Failed to cancel reservation.\n";

break;

case 4:

if (perf.confirmReservation(price)) std::cout << "Reservation confirmed successfully.\n";

else std::cout << "Failed to confirm reservation.\n";

break;

case 5:

std::cout << "Returning to previous menu...\n";

break;

default:

std::cout << "Invalid option, please try again.\n";

}

} while (action != 5);

}

};

void showMainMenu() {

std::cout << "\n\*\*\* Main Menu \*\*\*\n";

std::cout << "1. Search Performances\n";

std::cout << "2. View Performance Details\n";

std::cout << "3. Manage Tickets\n";

std::cout << "4. System Management\n";

std::cout << "5. Exit\n";

std::cout << "Select an option: ";

}

void showSearchMenu() {

std::cout << "\n\*\*\* Search Menu \*\*\*\n";

std::cout << "1. Search by Author\n";

std::cout << "2. Search by Title\n";

std::cout << "3. Exit to Main Menu\n";

std::cout << "Select an option: ";

}

void showSystemManagementMenu() {

std::cout << "\n\*\*\* System Management Menu \*\*\*\n";

std::cout << "1. Add New Performance\n";

std::cout << "2. Edit Existing Performance\n";

std::cout << "3. Remove Performance\n";

std::cout << "4. Update Ticket Prices\n";

std::cout << "5. Return to Main Menu\n";

std::cout << "Select an option: ";

}

void handleSystemManagement(PerformanceManager& manager) {

while (true) {

showSystemManagementMenu();

int choice;

std::cin >> choice;

std::cin.ignore();

switch (choice) {

case 1: {

std::string title, author, genre, date;

std::map<int, int> tickets;

std::cout << "Enter title: ";

getline(std::cin, title);

std::cout << "Enter author: ";

getline(std::cin, author);

std::cout << "Enter genre: ";

getline(std::cin, genre);

std::cout << "Enter date: ";

getline(std::cin, date);

int price, quantity;

char cont = 'y';

while (cont == 'y') {

std::cout << "Enter ticket price: ";

std::cin >> price;

std::cout << "Enter quantity: ";

std::cin >> quantity;

tickets[price] = quantity;

std::cout << "Add more tickets? (y/n): ";

std::cin >> cont;

std::cin.ignore();

}

manager.addPerformance(Performance(title, author, genre, date, tickets));

std::cout << "New performance added.\n";

break;

}

case 2: {

std::string title;

std::cout << "Enter the title of the performance to edit: ";

getline(std::cin, title);

manager.editPerformance(title);

break;

}

case 3: {

std::string title;

std::cout << "Enter the title of the performance to remove: ";

getline(std::cin, title);

manager.removePerformance(title);

break;

}

case 4: {

std::string title;

std::cout << "Enter the title of the performance to update ticket prices: ";

getline(std::cin, title);

manager.updateTicketPrices(title);

break;

}

case 5:

return;

default:

std::cout << "Invalid option, please try again.\n";

}

}

}

int main() {

PerformanceManager manager;

manager.addPerformance(Performance("Hamlet", "Shakespeare", "Drama", "2024-05-10", {{50, 100}, {75, 50}}));

manager.addPerformance(Performance("Macbeth", "Shakespeare", "Tragedy", "2024-05-15", {{40, 150}, {65, 100}}));

while (true) {

showMainMenu();

int mainChoice;

std::cin >> mainChoice;

std::cin.ignore();

switch (mainChoice) {

case 1:

while (true) {

showSearchMenu();

int searchChoice;

std::cin >> searchChoice;

std::cin.ignore();

if (searchChoice == 1) {

manager.searchByAuthor();

} else if (searchChoice == 2) {

manager.searchByTitle();

} else if (searchChoice == 3) {

break;

} else {

std::cout << "Invalid option, please try again.\n";

}

}

break;

case 2:

manager.viewPerformanceDetails();

break;

case 3:

manager.viewAndManageTickets();

break;

case 4:

handleSystemManagement(manager);

break;

case 5:

std::cout << "Exiting program.\n";

return 0;

default:

std::cout << "Invalid option, please try again.\n";

}

}

return 0;

}