Форма № Н-6.01у

Кременчуцький національний університет імені Михайла Остроградського
(повне найменування вищого навчального закладу)
Кафедра автоматизації та інформаційних систем
(повна назва кафедри, циклової комісії)

КУРСОВИЙ ПРОЄКТ (РОБОТА)

з дисципліни «Сучасні мови об'єктно-орієнтованого програмування»
(назва дисципліни)

на тему Довідник туриста.

Студента 2 курсу КН–23–1 групи
Ступінь вищої освіти «Бакалавр»
(бакалавр, магістр)
Спеціальність 122 – «Комп'ютерні науки»
Освітньо-професійна програма
«Комп'ютерні науки»
Гладкий І.Г
(прізвище та ініціали)
Керівник старший викладач кафедри АІС
Бельська В. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____

Кількість балів: _____. Оцінка: ЄКТС ____

Члени комісії _____. В. Ю. Бельська
(підпис) (ініціали та прізвище)
_____ Істоміна Н.М.
(підпис) (ініціали та прізвище)
_____ Кліменко О.В.
(підпис) (ініціали та прізвище)

м. Кременчук 2024 рік

Форма № Н-9.01у

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра автоматизації та інформаційних систем

Дисципліна «Сучасні мови об'єктно-орієнтованого програмувння»

Освітній ступінь «Бакалавр»

Спеціальність 122 – «Комп'ютерні науки»

Освітня програма «Комп'ютерні науки»

Курс 2 група КН-23-1 семестр 3

ЗАВДАННЯ
НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Гладкий Іван Геннадійович
(прізвище, ім'я, по-батькові)

1. Тема роботи: « Довідник туриста »
2. Термін здачі студентом роботи 7 грудня 2023 р
3. Вихідні дані до роботи: .xml формату, що містять колекції даних від яких залежить функціонал курсової роботи
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): Об'єктно-орієнтованого програмування: інкапсуляція, спадкування, поліморфізм
5. Перелік графічного матеріалу:
6. Дата видачі завдання: 9.09. 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ пор.	Назва етапів курсового проекту	Терміни виконання етапів проекту	Вказівки та зауваження викладача (з зазначенням дати консультації)	Оцінювання етапів проекту		
				за національною шкалою	за шкалою ЄКТС	кількість балів
1	Етап 1 Аналіз предметної області	05.10.24- 25.10.24				
2	Етап 2 Створення моделі даних	16.10.24- 28.10.24				
3	Етап 3 Розробка віконного інтерфейсу та створення основного програмного коду додатку	29.10.24- 24.11.24				
4	Етап 4 Тестування програмного коду	16.10.24- 29.11.24				
5	Етап 5 Оформлення пояснювальної записки	25.11.24- 31.11.24				
6	Етап 9 Захист	05.12.24				
	Разом	8 тижнів				

Студент _____
(підпис)

Керівник _____ В. Ю. Бельська
(підпис) (ініціали та прізвище)

«5» грудня 2024 р.

РЕФЕРАТ

Курсова робота містить 6 сторінки, 2 розділи, 16 рисунків, 1 таблицю, 3 використаних джерела.

Об'єкт розробки – WPF додаток «Wander».

Мета: створення багатосторінкового WPF-додатку, що реалізує застосунок «Довідник туриста».

Під час виконання курсової роботи було проведено аналіз вимог до додатку. Визначено основні функціональні та нефункціональні характеристики, які забезпечують комфортне користування та управління інформацією про туристичні послуги. Було створено модель даних, що включає:

- Реєстрацію та авторизацію користувачів.
- Інформацію про послуги: країна, місто або маршрут круїзу, умови проживання та транспорту, екскурсійне обслуговування, сервіс приймаючої сторони, вартість путівки.
- Модулі для відображення даних та графічного інтерфейсу, що дозволяють легко переглядати, шукати та фільтрувати туристичні пропозиції.

В якості мови програмування для створення Wpf додатку використовувалася мова c# та середовище програмування Visual Studio 22

Результатом виконання всіх етапів є застосунок, який може реєструвати користувачів, придбання квитків, та пошук за фільром.

ЗМІСТ

ВСТУП

В сучасному світі комп’ютерні технології використовуються майже у всіх сферах нашого життя. Однією з важливих галузей їх застосування є туристична індустрія. Завдяки сучасним технологіям подорожі стали доступнішими, а планування маршруту перетворилося на зручний і швидкий процес. У 2024 році туристичні додатки та сервіси стали невід’ємною частиною життя багатьох людей, допомагаючи їм відкривати нові країни, міста та культури. Туристичні платформи дозволяють знаходити пропозиції від агентств, які включають широкий спектр послуг: від бронювання проживання та транспорту до організації екскурсій та розваг. Користувачі можуть взаємодіяти з цими платформами через зручний інтерфейс, що підтримує пошук, сортування та фільтрацію за різними параметрами, такими як ціна, тривалість поїздки, умови проживання та інші. На сьогоднішній день туристична індустрія розвивається стрімкими темпами. Кількість подорожуючих зростає щороку, а разом з тим і попит на цифрові сервіси для планування подорожей. Інвестори все більше вкладаються у розвиток туристичних платформ, які стають інноваційними та інтерактивними. Туристичні сервіси пропонують користувачам не тільки можливість планування, а й соціальну взаємодію, наприклад, через рейтинги, відгуки чи рекомендації. Водночас, інноваційні рішення, такі як віртуальні тури, дозволяють заздалегідь ознайомитися з місцем подорожі, що підвищує комфорт і впевненість користувачів. Отже, туристичні сервіси є важливим інструментом сучасної людини, що дозволяє економити час і ресурси, відкриваючи світ і даруючи новий досвід. Ніщо не ідеальне, і туристичні сервіси також мають свої обмеження, однак їх переваги значно перевищують недоліки.

Метою даного курсового проекту є розробка багатовіконного WPF-додатку «Довідник туриста», що надає користувачам зручний доступ до інформації про туристичні агентства та пропоновані послуги.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Exception. Призначення блоків try, catch, finaly. call stack.

Виняток (Exception) — це подія, що виникає під час виконання програми, яка порушує її нормальний потік виконання. Винятки використовуються для повідомлення про помилки або інші непередбачувані ситуації, які програма не може обробити звичайними засобами. Винятки можуть бути викликані як системою (наприклад, поділ на нуль, спроба доступу до неіснуючого елемента масиву), так і самим програмістом у випадках, коли необхідно явним чином згенерувати помилку

TRY - Цей блок використовується для коду, виконання якого може призвести до виникнення винятків. У ньому розташовуються інструкції, які потрібно виконати. Якщо під час виконання коду у блоці try виникає виняток, він перериває нормальний потік виконання і передає управління відповідному блоку catch.

CATCH - Цей блок обробляє винятки, які виникли у блоці try. Для кожного типу винятків можна створити окремий блок catch, забезпечуючи таким чином більш детальну обробку різних помилок.

FINNALY - Цей блок виконується в будь-якому випадку — незалежно від того, чи стався виняток у блоці try, чи ні. Він зазвичай використовується для звільнення ресурсів, закриття файлів, з'єднань із базою даних тощо.

CALL STACK - це структура даних, яка використовується для відстеження активних методів (функцій) під час виконання програми. Кожен раз, коли метод викликається, у стек додається новий фрейм (контекст виклику), що містить інформацію про:

- Адресу повернення після завершення виклику.
- Локальні змінні та параметри методу.
- Іншу контекстну інформацію.

Механізм обробки винятків є важливим інструментом для забезпечення стабільної роботи програмного забезпечення. Розуміння принципів роботи блоків try, catch, finally та стеку викликів допомагає програмістам створювати ефективний, чистий і захищений від помилок код.

1.2 Аналіз технічного завдання на роботу

Мета курсового проєкту – Довідник туриста.

Під час розробки моделей даних та графічного інтерфейсу були висунуті наступні вимоги:

1.2.1 Функціональні вимоги

1. **Реєстрація та авторизація користувачів:**
   - Реєстрація та авторизація користувачів.
   - Створення нового облікового запису з перевіркою даних.
   - Вхід у систему через логін і пароль
2. **Збереження даних користувачів:**
   - Збереження інформації про облікові записи (ім'я, контактні дані, історія покупок).
   - Безпечне зберігання паролів із використанням хешування.
3. **Покупка турів:**
   - Вибір туру із запропонованих варіантів.
   - Можливість оформлення покупки з реєстрацією платежу.
4. **Фільтрація даних:**
   - Пошук турів за параметрами: країна, місто, датою.
5. **Особистий кабінет користувача:**
   - Перегляд історії покупок.

1.2.2 Нефункціональні вимоги

1. Зовнішній інтерфейс користувача має бути реалізованим за допомогою створення вікон в середовищі WPF додатку.

2. Додаток має бути багатосторінковий.

3. Усі поля введення повинні бути захищені від некоректного введення.

4. Використання патерну MVVM та прив’язки даних.

5. Усі дані повинні зберігатися у XML форматі.

6. Тестування

1.3 Опис алгоритму основних задач/підзадач у роботі

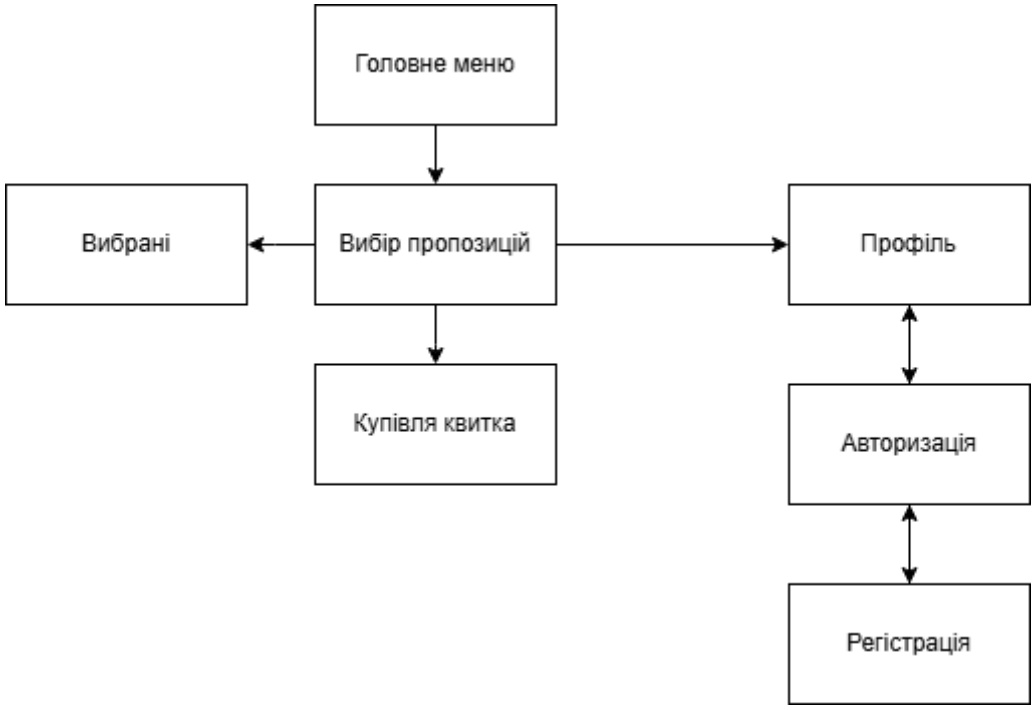

Рисунок 1.1 – Діаграма загальньного представлення алгоритму основних задач

Програма має складатися з «Головного меню» «Профіль» «Купівля» «Авторизація/Регістрація» та «Купівля».

У головному меню користувач повинен мати змогу переходити до будь-якої частини.

Профіль має надавати перегляну данні користивуча, вийти з самого аккаунта та переглянути списку куплених квитків.

Авторизація надає користовучу зайти в свій аккаунт.

Регістрація надає новим користувачам зробити новий аккаунт за допомогою своїх данних.

Купівля надає користовучу купити вибраний їм тур за домогою персональних данних.

Висновки до розділу

У першому розділі було проведено глибокий аналіз предметної області та сформульовано основні принципи побудови програмного забезпечення для туристичного агентства. Розглянуто ключові аспекти об'єктно-орієнтованого програмування, такі як інкапсуляція, спадкування та поліморфізм, які є фундаментальними для створення масштабованого та безпечного додатку.

Сформульовані функціональні та нефункціональні вимоги до системи, що забезпечують її стабільну роботу та зручність у використанні. Виділено основні функції, зокрема:

- реєстрація та авторизація користувачів,
- збереження їхніх даних,
- фільтрація турів,
- купівля квитків,
- перегляд особистих даних та історії покупок.

Також визначено, що програма має бути багатовіконною з використанням патерну MVVM, а зберігання даних реалізовано у форматі XML.

Розроблено загальний алгоритм роботи програми, який дозволяє забезпечити логічну структуру взаємодії між її компонентами. У підсумку, всі ці елементи формують основу для реалізації зручного, ефективного та безпечного додатку, що задовольняє потреби туристичного агентства.

2 ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Структура програмного забезпечення

Структура програмного забезпечення є поєднанням усіх застосованих модулів програми, що утворюють працюючу систему. Кожний модуль повинен реалізовувати функції програми, що вирішують основні завдання проєкту. Так як це груповий проект, буде висвітлена тільки частина, виконана мною, а саме розробка моделей даних (Model), моделей вигляду (ViewModel), що забезпечують основну логіку роботи додатку, обробку даних та взаємодію між компонентами системи..

Застосунок створений за допомою модулю Windows Presentation Foundation (WPF) на базі мови C#.

На рис. 2.1 зображено оглядач рішень програмного застосунку. В табл.2.1 представлено опис модулів(класів) проекту курсової роботи

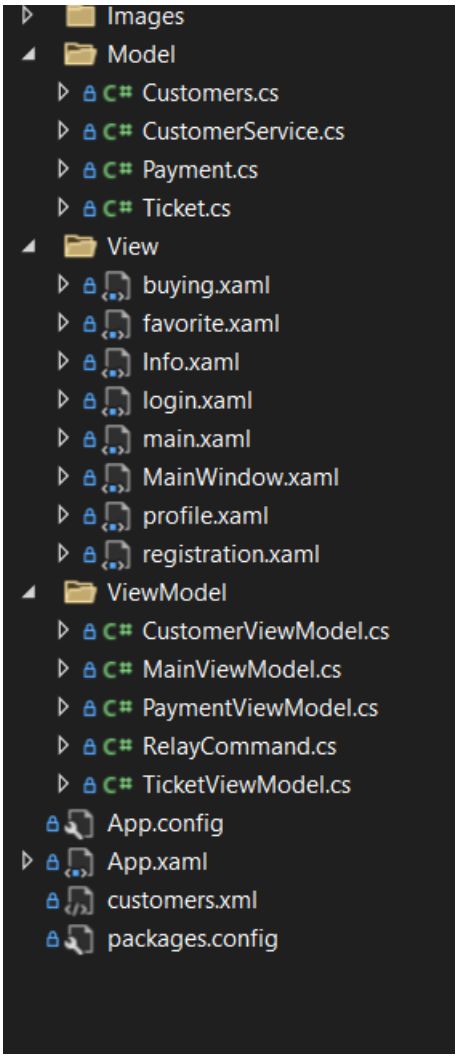

Рисунок 2.1 – Оглядач рішень

Таблиця 2.1 – Призначення модулів програми

Назва модуля	**Призначення модуля**
Customer.cs	Базовий клас для представлення даних користувача (ім’я, фамілія, номер телефону, нікнейм, пароль)
CustomerService.cs	**Синглтон**-клас для управління даними користувачів та обновленням їх інформації. Клас дозволяє працювати з файлом customers.xml .
Payment.cs	Базовий клас для представлення даних про оплату (номер картки, cvv, термін придатності, ім’я власника, адреса)
Ticket.cs	Базовий клас для представлення даних про квиток (пункт назначення, прибуття, дата, назва та картинка отелю, період, наявність квитка, опис, ціна)
RellayCommand.cs	Реалізує інтерфейс ICommand
MainViewModel.cs	Клас для зв’язування ViewModel класів
TicketViewModel.cs	Відповідає за роботу з даними класа Ticket
PaymentViewModel.cs	Відповідає за роботу з даними класа Payment
CustomerViewModel.cs .	Відповідає за роботу з даними класа Customer

2.2 Опис роботи програми

Після запуску програми відкривається вікно меню, з якого можна перейти у основне меню програми. Напоч покупка атку будуть вислітлюватисяпропозиції турів і кнопки «Профіль» «Очистити» «Придбати» та «Шукати».Усі кнопки окрім «Пошуку» просять нового користувача авторизуватися. Якщо такої можливості нема користувач повинен зарегіструватися.

Після авторизації кнопки вже будуть перенаправляти на нові вікна:

- Сторінка «Профіль» містить данні користувача та кнопку «Мої квитки». Також можна вийти з акаунта для заходу в інший;
- Сторінка «Вибрані» надає користовачу вибрати улюблені їм квитки і при необходимості видалити або придбати.
- Сторінка «Придбати» показує користовачу поля в які він повинен ввести особисті данні для купівлі квитка.

2.3 Функціональна схема програми

Детальний опис й пояснення процесів окремих функцій розробляємої системи утворюють функціональну схему програми(рис. 2.5).

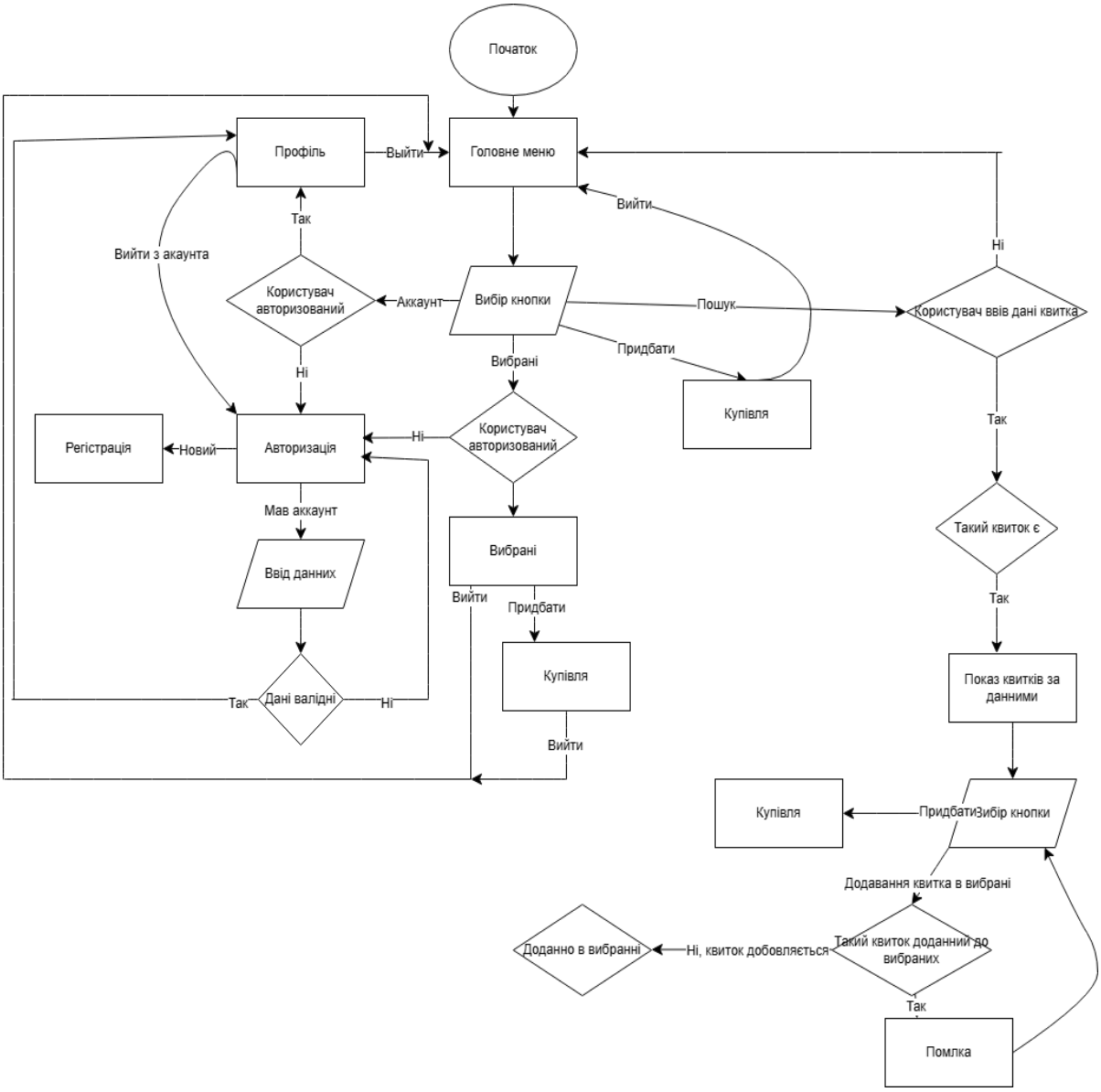

Рисунок 2.2 – Функціональна схема програмного коду

2.4 Опис інтерфейсу програми

При розробці програми створено зручний інтерфейс користувача. Опишемо основні вікна/сторінки програми:

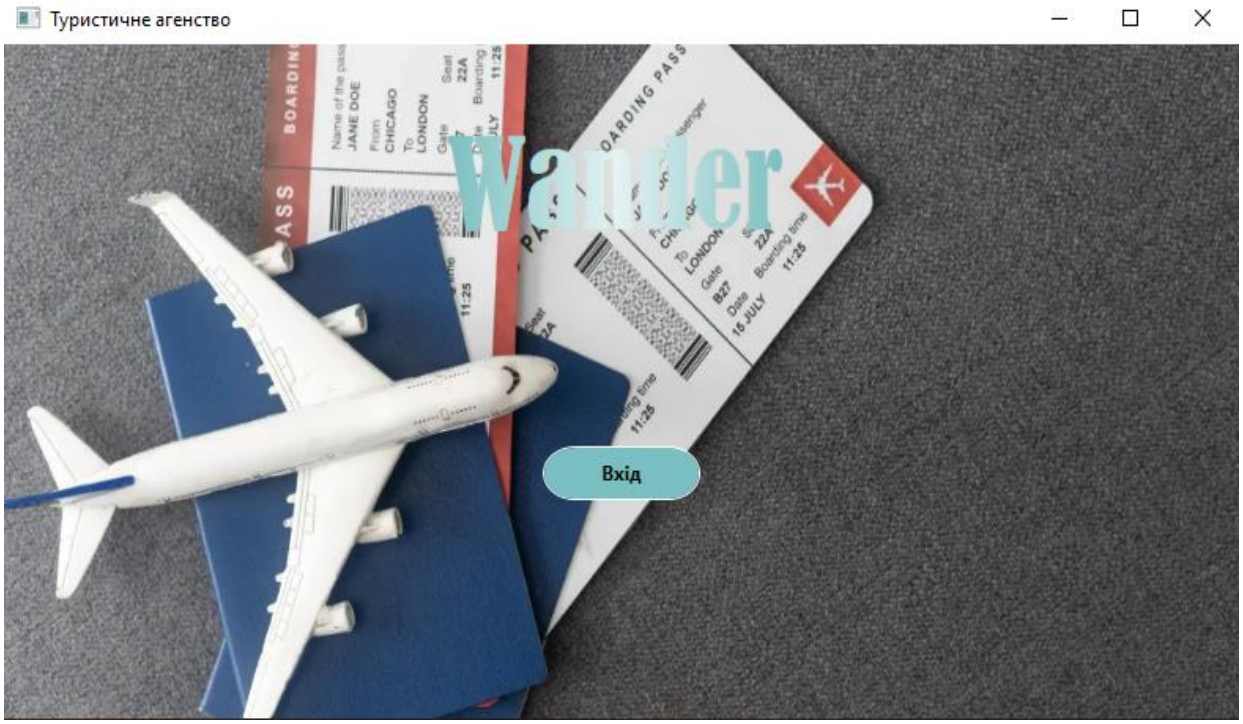

Рисунок 2.3 – Інтерфейс вікна MainWindow

Вікно меню (рисунок 2.5) містить 1 кнопок 1 текст:

1) При натисканні кнопки «Вхід», користувача, перекидує.у головне меню застосунку

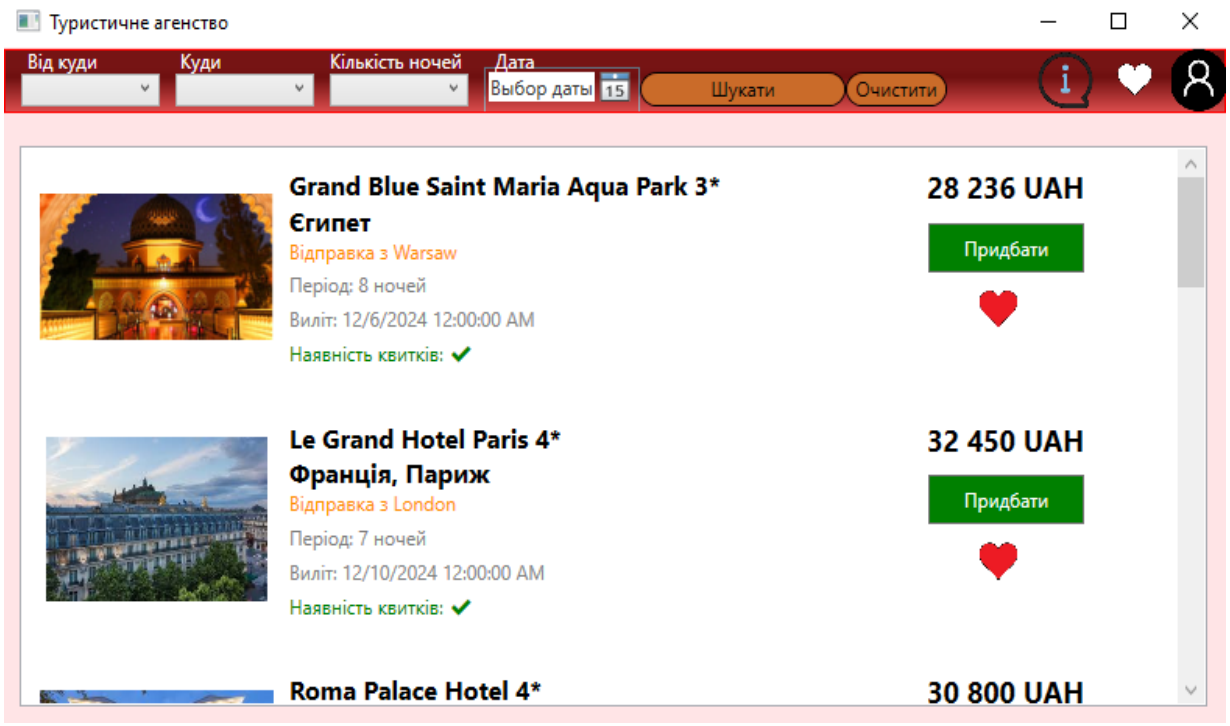

Рисунок 2.4 – Сторінка головного меню

Сторінка головного меню (рисунок 2.6) містить 2 кнопки, 3 комбінованих списків тексту 1 видір дати , 3 вибору для фільтрації також 5 кнопок(1 Datapicker, 3 ComboBox, 5 Button, 4 Lable):

1) Кнопка «Шукати» відправляє введені дані на перевірку валідності, при правильній вводі даних при натискані кнопки пройде пошук за задам фільтр і видасть результат

2) Кнопка «Очистити» витирає фільтрацію з полів

3) В виборі дати користувач має право вибрати дату виліту

4) Кнопка «Вибрані» переводить користувача на свої любимі квитки які він зможе куптім, якщо користувач не авторизован його перекине на сторінку авторизації.

5) Кнопка «Профіль» переводить користувача на сторінки авторизації.

6) Кнопка «О нас» переводить на розробників застосунки і контакних данних.

7) Біля кожного туру є кнопка «Придбати» яка переводить на сторінку купівлі квитка, якщо користувач не авторизований він повинен це зробити.

8) Біля кожного туру є ще одна кнопка «Вибраний» додає тур на сторінку Вибрані, якщо користувач не авторизований він повинен це зробити.

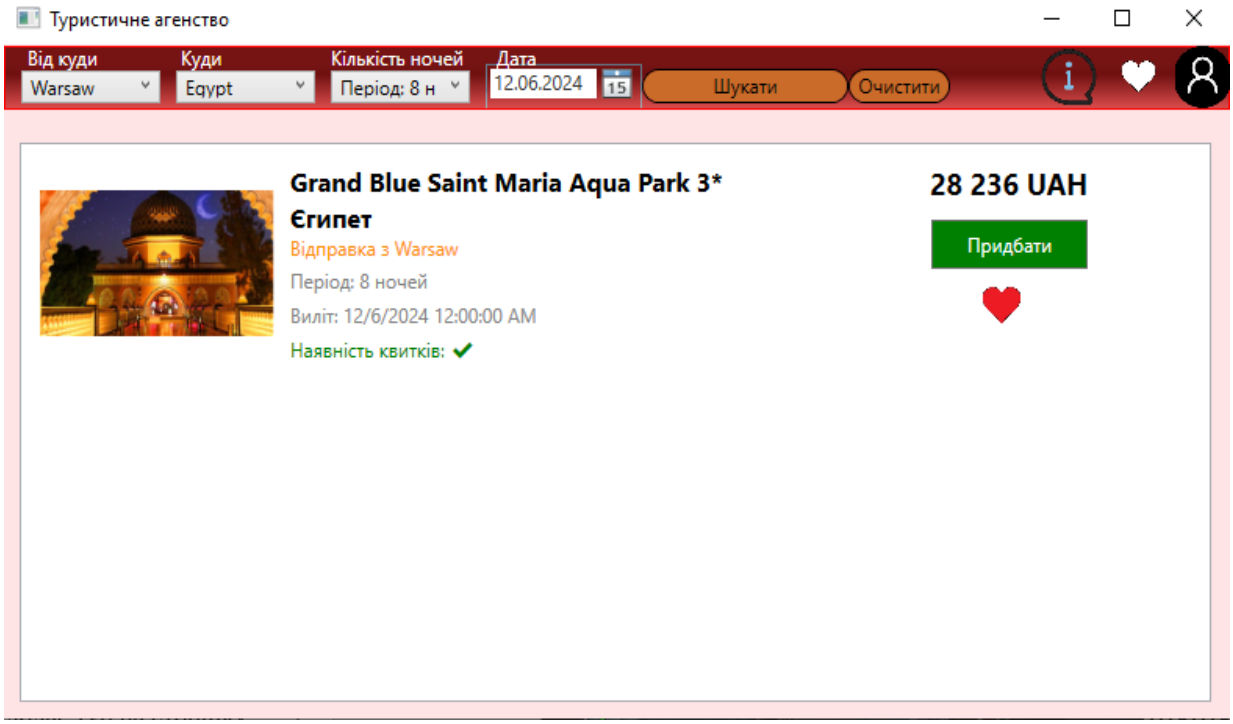

Рисунок 2.5 – Сторіннка з виконаним пошуком

Сторінка авторизації(Рисунок 2.8) містить 2 кнопоки і 2 поле вводу данних.

1) Кнопка «Вхід» переводить користувача до профіля якщо дані введені правильні, якщо дані не правильні буде помилка(Рисунок2.9)
2) Кнопка «Регістрація» переводить користувача до сторінки регістрації нового аккаунта.

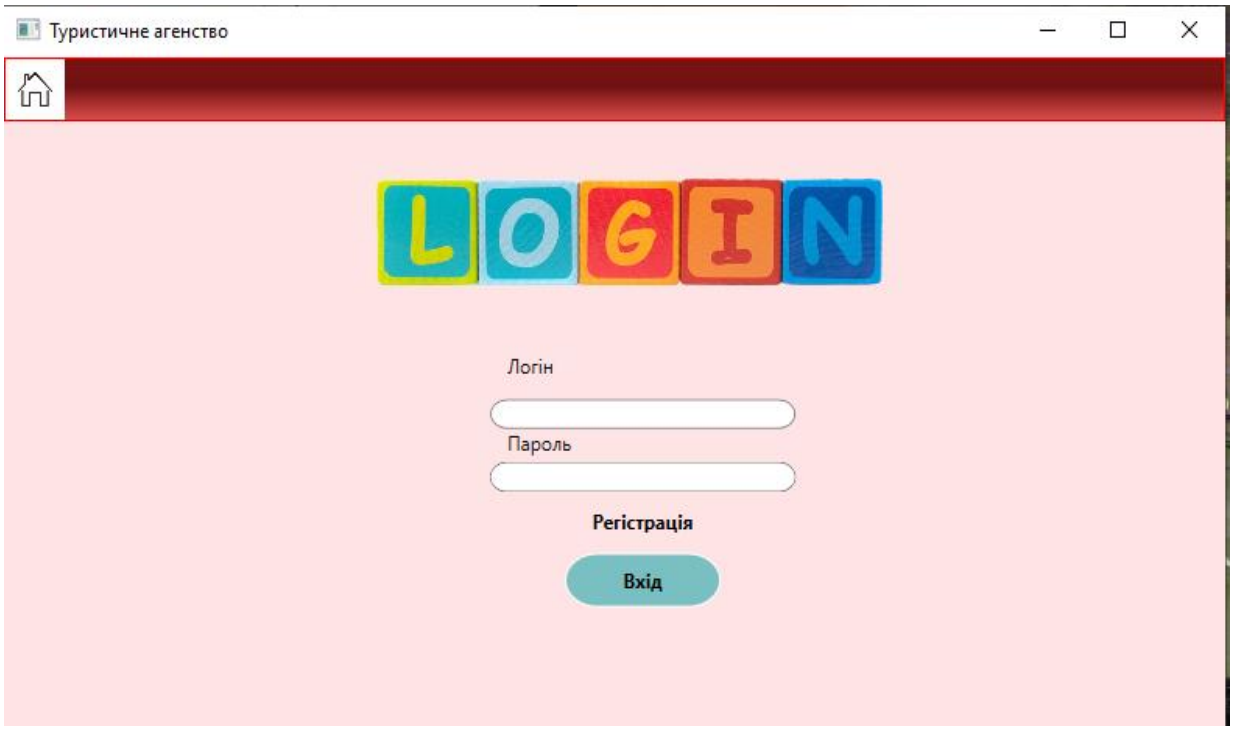

Рисунок 2.6 – Сторінка авторизації

Рисунок 2.7 – Помилка в авторизації

Вікно регістрації (Рисунок 2.10) містить 4 кнопоки і 5 поле вводу данних:

1) Кнопка «Реєстрації» створює аккаунт за допомогою данних які ввів користувач.

   Якщо користувач ввів не правильно номер телефона або логін вже використан застосунок видасть помилку(Рисунок 2.11).

2) Кнопки «Рєстрації за допомогою соцмережеє» реєструє(візуально, кнопки не виконують ніякого функціонала, вони використані тільки як дізайн) аккаунт в застосунку

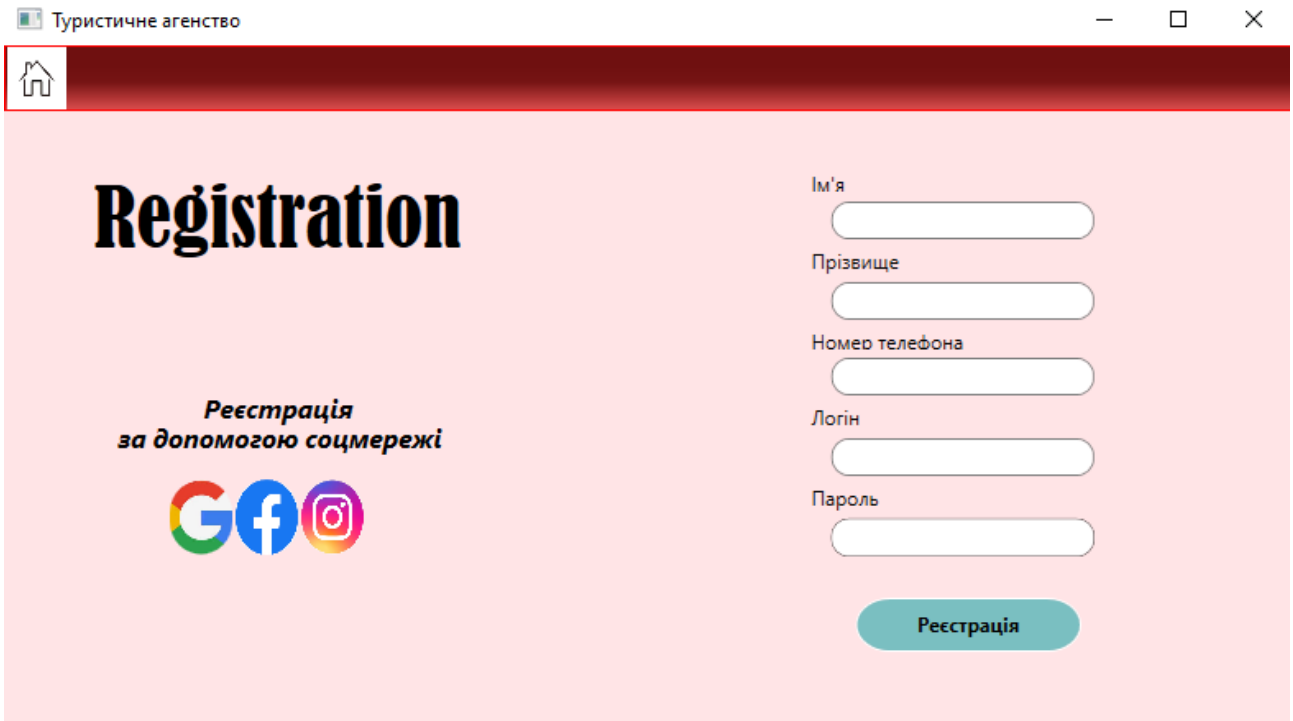

Рисунок 2.8 – Сторінка реєстрації

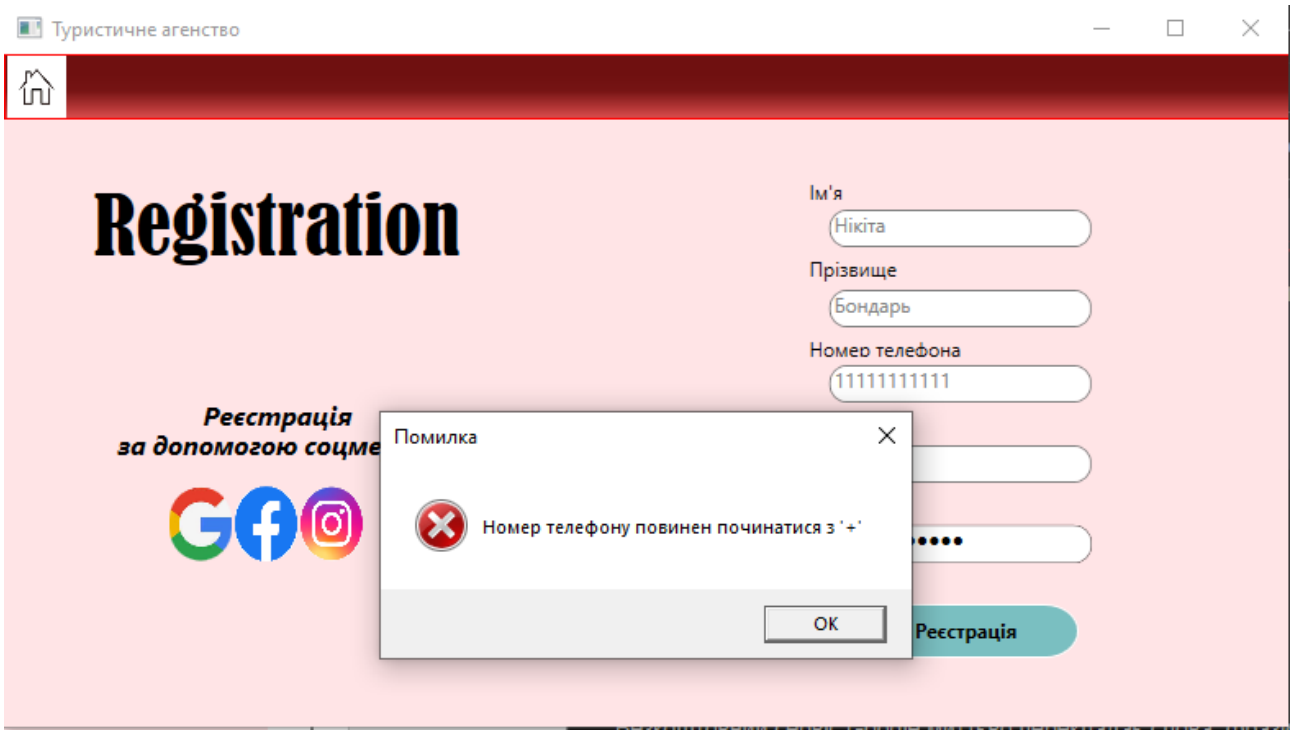

Рисунок 2.9 – MessageBox із вказанням невірних даних

Меню профілю (рисунок 2.12) містить 2 кнопки:

1) Кнопка «Вийти» повертає користувача до авторизації та виходить із акаунту (знову необхідно входити у акаунт для користуванням застосунком).

2) Кнопка «Додому» повертає до головного меню.

3) Кнопка «Змінити фото» змінює фото профілю локально.

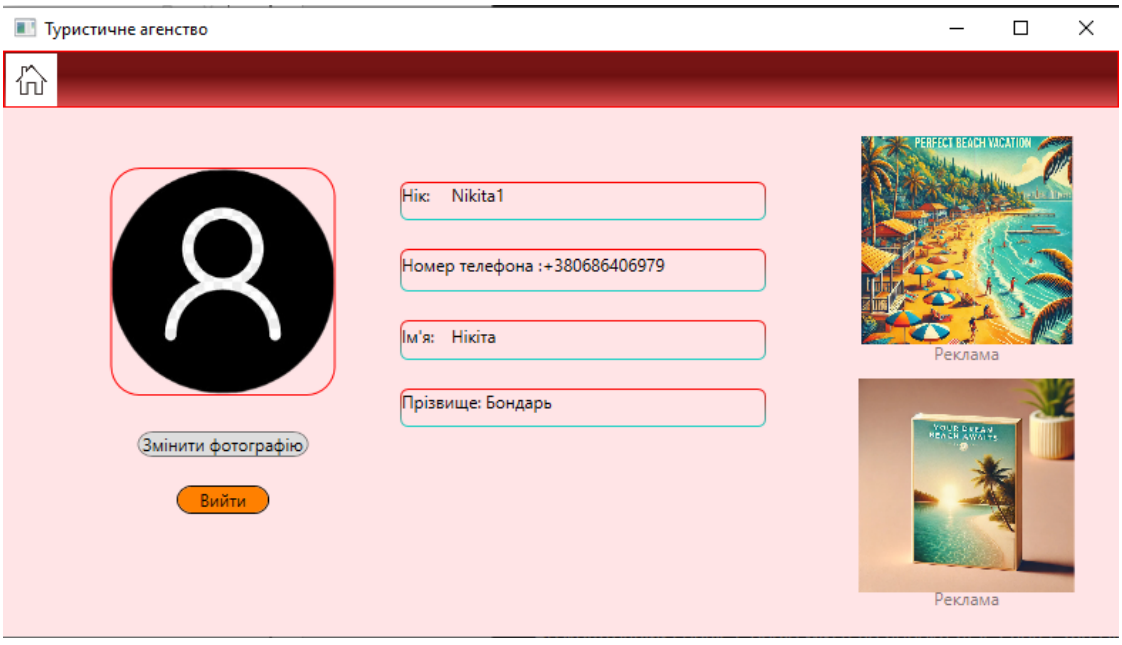

Рисунок 2.10 – Сторінка профілю

Сторінка «Вибрані» (рисунок 2.13) містить декілька кнопок – «Додому», «Придбати» та «Видалити»

1) Кнопка «Додому» повертає до головного меню.
2) Кнопка «Придбати» яка переводить на сторінку купівлі квитка.
3) Кнопка «Видалити» видаляє квитка с сторінки Вибрані.

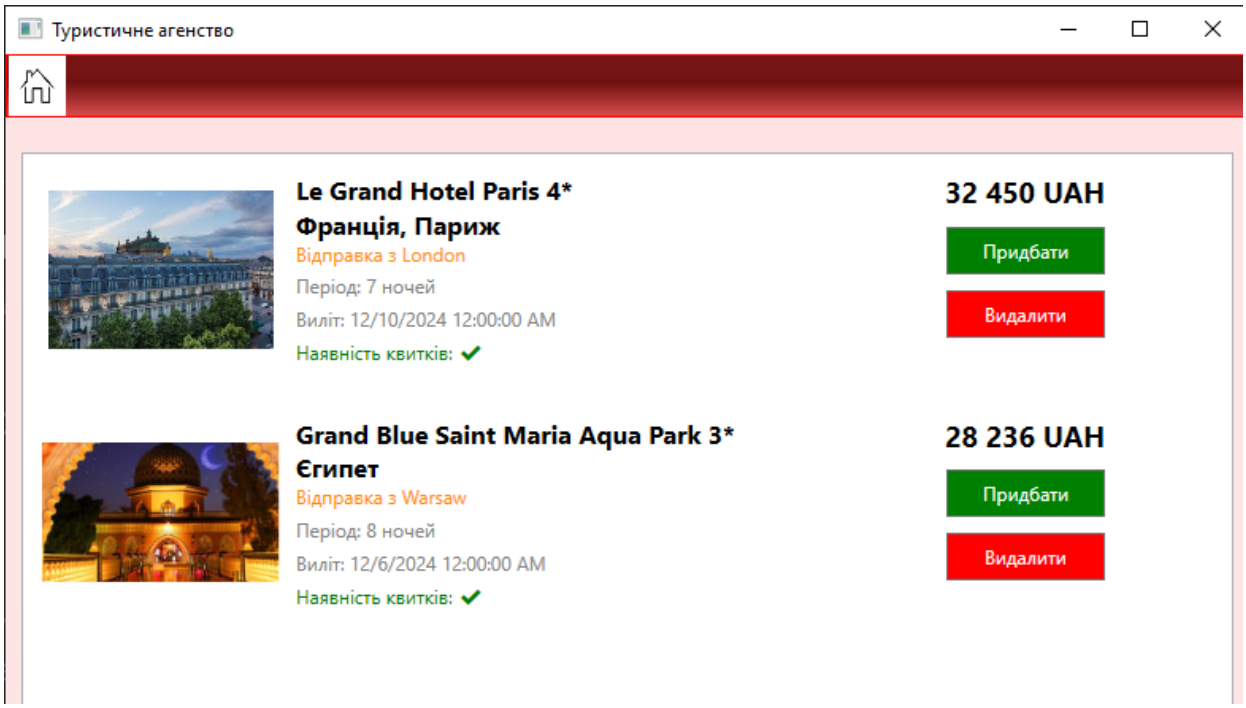

Рисунок 2.11-Сторінка Вибрані

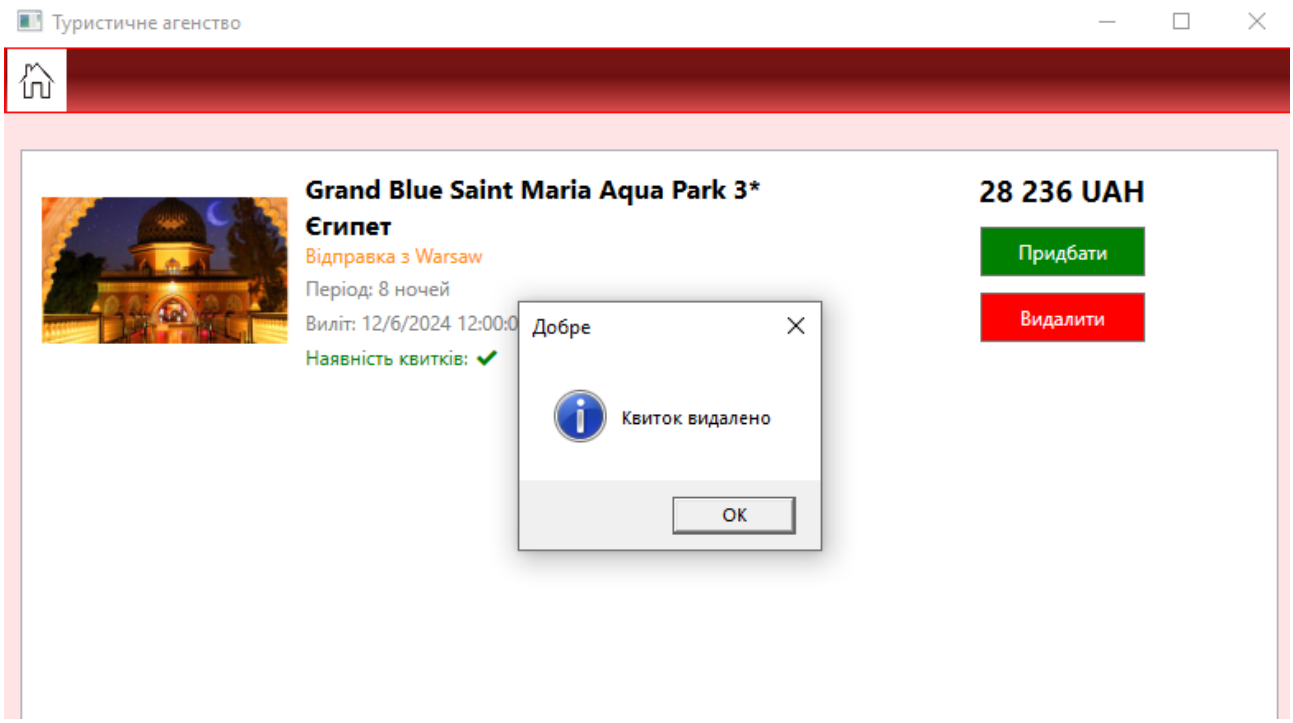

Рисунок 2.12- MessageBox із вказанням виконання кнопки

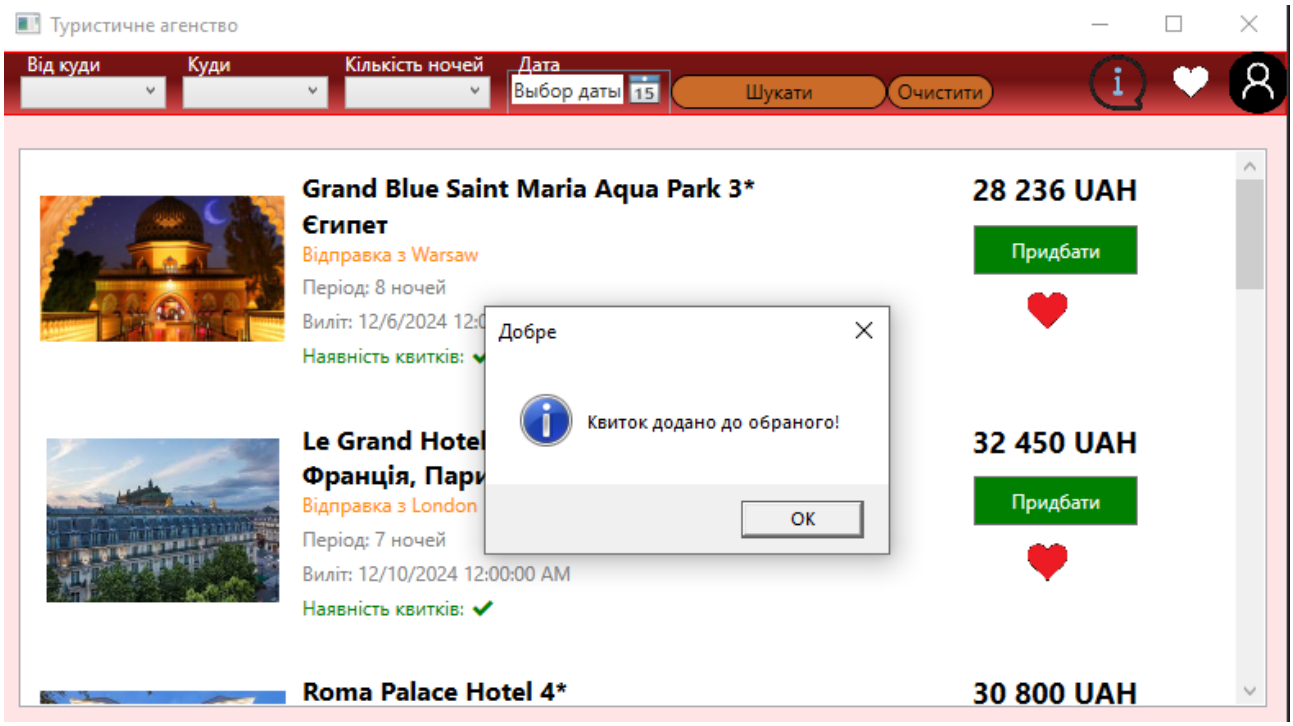

Рисунок 2.13- Додає квиток в Вибрані

Сторінка «Придбати» (Рисунок 2.16) містить 1 кнопку і 5 полів вводу даних.

1) Кнопка «Покупка» покупає(симулює) квиток.(Рисунок 2.17)

2) Якщо дані введені не правильно або не дописно застосунок повідомить (Рисунок 2.18)
3) Кнопка «Додому» повертає до головного меню.

Рисунок 2.14 – Сторінка «Придбати»

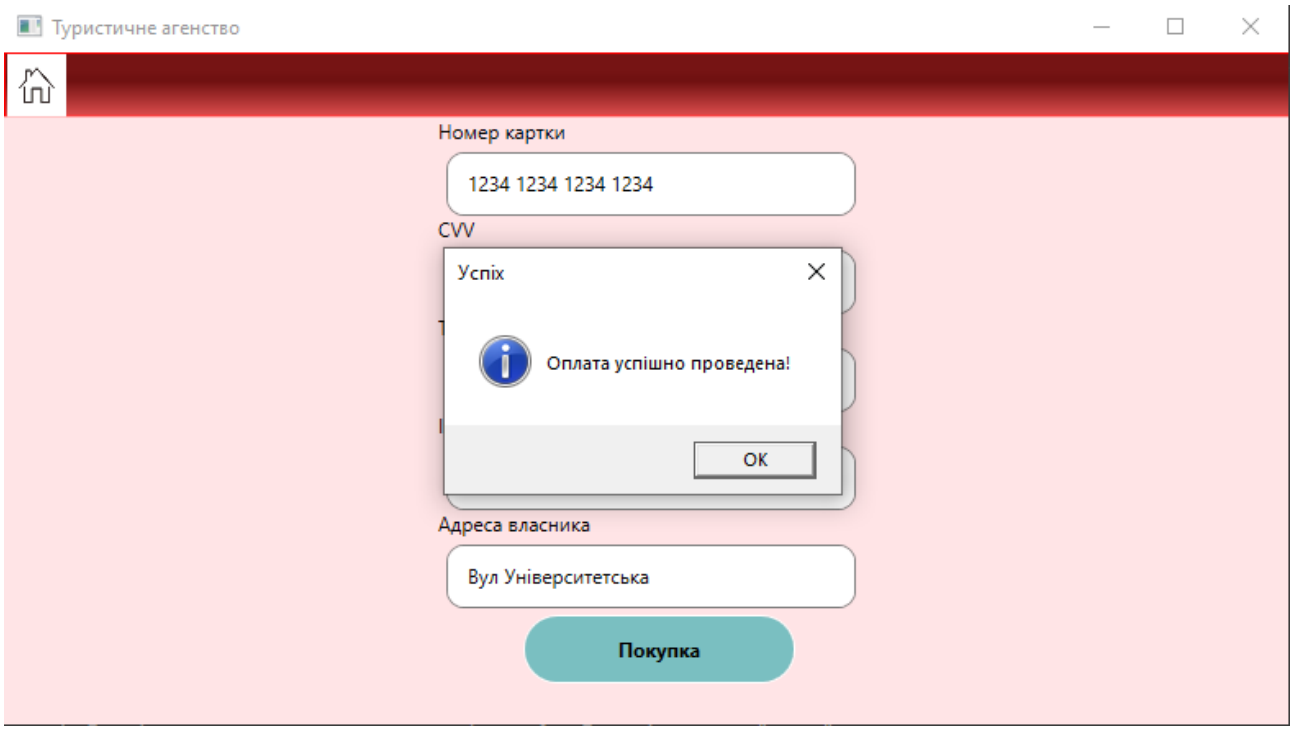

Рисунок 2.15 Виконання покупки

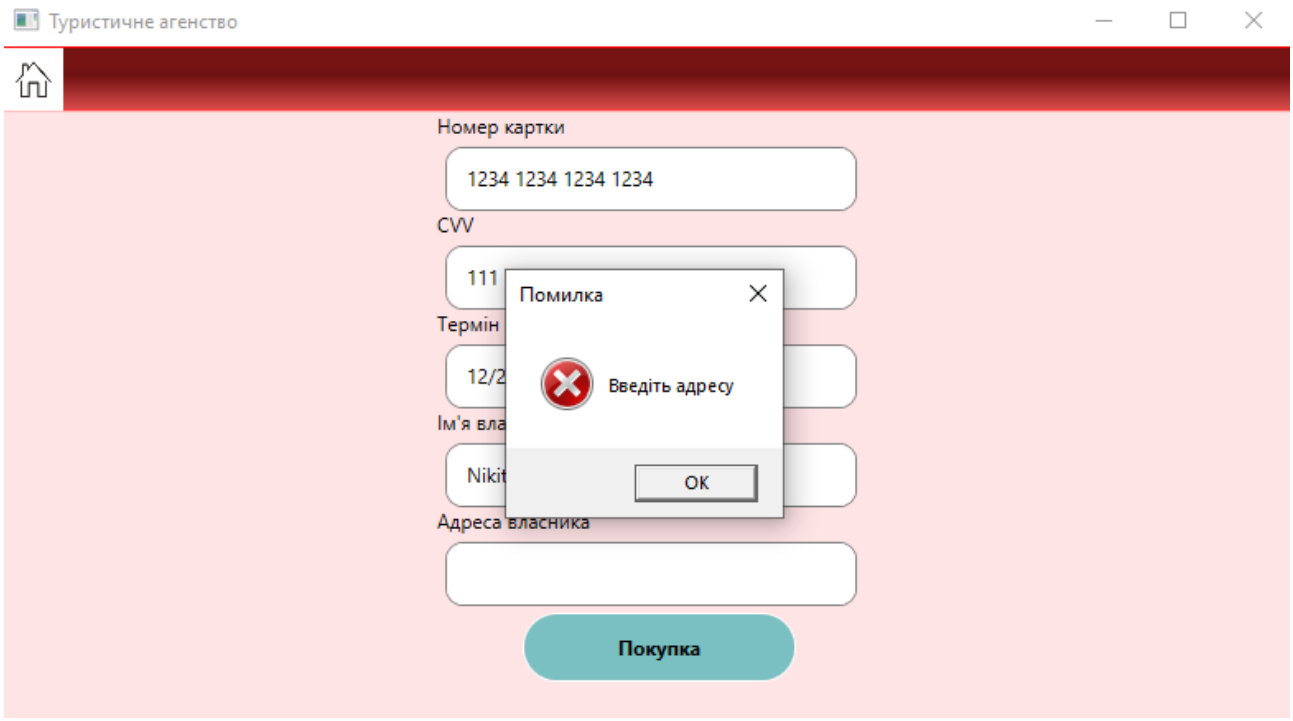

Рисунок 2.16 MessageBox із вказанням невірних даних

Висновки до розділу

У другому розділі детально розглянуто архітектурну організацію проекту, включаючи структуру класів, механізми взаємодії між компонентами та особливості ініціалізації даних. Представлено функціональну схему, яка відображає логіку роботи системи та взаємозв'язки між її складовими. Описано розроблений користувацький інтерфейс та принципи взаємодії користувача з системою через різні функціональні вікна програми.

ВИСНОВКИ

Курсовий проєкт спрямований на розробку WPF-додатку для туристичної агенції. У проєкті розглянуто базові принципи об'єктно-орієнтованого програмування, проаналізовано предметну область та розроблено технічне завдання. Реалізовано функціонал для реєстрації та авторизації користувачів, створення та перегляду замовлень на туристичні послуги, а також можливість оплати вибраних турів.

Основною мовою програмування обрано C#, яка чудово підходить для побудови програм на основі ООП. Особлива увага приділялася плануванню і створенню зручного інтерфейсу користувача. Основними завданнями були забезпечення комфортної роботи з інтерфейсом, швидкого обміну даними, простого управління та реалізації всіх необхідних функцій. Інтерфейс виконано з використанням приємних кольорових схем, багатовіконної структури та передбачено механізми захисту від некоректного введення даних.

Проєкт демонструє основні методи роботи з WPF, його інструментами та можливостями, що дозволяє ефективно розв’язувати завдання у сфері розробки програмного забезпечення для туристичних послуг.

СПИСОК ЛІТЕРАТУРИ

1. Офіційна документація Microsoftдля WPF
https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-8.0

2. Паттерн MVVM .Інформаційний ресурс.
https://metanit.com/sharp/wpf/22.1.php

3. Офіційна документація Microsoft для C#
https://learn.microsoft.com/uk-ua/dotnet/csharp/

Додаток Б

Класи MODEL

Клас Customer.cs

```
namespace kursach.Model
{
   [Serializable]
   public class Customer
   {
      public string FirstName { get; set; }
      public string SecondName { get; set; }
      public string NickName { get; set; }
      public string Password { get; set; }
      public string PhoneNumber { get; set; }
      public string AvatarPath { get; set; }
      [XmlArray("LikedTickets")]
      [XmlArrayItem("Ticket")]
      public List<Ticket> LikedTickets { get; set; }
      public Customer()
      {
         AvatarPath = "pack://application:,,,/Images/Без_названия-removebg-
preview.png";
         LikedTickets = new List<Ticket>();
      }
      public Customer(string firstName, string lastName, string phoneNumber, string
nickName, string password)
      { FirstName = firstName; SecondName = lastName; PhoneNumber =
phoneNumber; NickName = nickName; Password = password; AvatarPath =
"pack://application:,,,/Images/Без_названия-removebg-preview.png"; LikedTickets
= new List<Ticket>(); }
      public void AddLikedTicket(Ticket ticket)
      {
         if (!LikedTickets.Any(t => t.HotelName == ticket.HotelName && t.Date ==
ticket.Date))
         {
            LikedTickets.Add(ticket);
         }
      }
      public void RemoveLikedTicket(Ticket ticket)
      {
         var ticketToRemove = LikedTickets.FirstOrDefault(t =>
            t.HotelName == ticket.HotelName &&
            t.Date == ticket.Date);

         if (ticketToRemove != null)
         {
            LikedTickets.Remove(ticketToRemove);
         }
      }
```

```
    }
}
```

Клас CustomerService.cs

```
namespace kursach.Model
{
    public class CustomerService
    {
        private static CustomerService _instance;
        public static CustomerService Instance
        {
            get
            {
                if (_instance == null)
                {
                    _instance = new CustomerService();
                }
                return _instance;
            }
        }
        private static readonly string FilePath =
Path.Combine(Directory.GetParent(AppDomain.CurrentDomain.BaseDirectory).Pare
nt.Parent.FullName,"customers.xml");
        private List<Customer> _customers;
        public static bool IsUserLoggedIn { get; set; } = false;
        public static Customer CurrentCustomer { get; private set; }
        public CustomerService()
        {
            LoadCustomers();
        }
        private void LoadCustomers()
        {
            if (File.Exists(FilePath))
            {
                var serializer = new XmlSerializer(typeof(List<Customer>));
                using (var stream = File.OpenRead(FilePath))
                {
                    _customers = (List<Customer>)serializer.Deserialize(stream) ?? new
List<Customer>();
                }
            }
            else
            {
                _customers = new List<Customer>();
                SaveChanges();
```

```
        }
    }
    public Customer GetByNickName(string nickName) =>
        _customers.FirstOrDefault(c => c.NickName.Equals(nickName,
StringComparison.OrdinalIgnoreCase));
    public bool ValidateUserCredentials(string nickName, string password)
    {
        var customer = GetByNickName(nickName);
        if (customer != null && customer.Password == password)
        {
            CurrentCustomer = customer;
            IsUserLoggedIn = true;
            return true;
        }
        IsUserLoggedIn = false;
        CurrentCustomer = null;
        return false;
    }
    public void SaveCustomer(Customer customer)
    {
        var existingCustomer = GetByNickName(customer.NickName);
        if (existingCustomer == null)
        {
            _customers.Add(customer);
            SaveChanges();
        }
    }
    public void UpdateCustomerAvatar(string nickName, string avatarPath)
    {
        var customer = GetByNickName(nickName);
        if (customer != null)
        {
            customer.AvatarPath = avatarPath;
            SaveChanges();
        }
    }
    public void UpdateCurrentCustomerLikedTickets(Ticket ticket, bool isAdding)
    {
        if (CurrentCustomer == null) return;

        if (isAdding)
        {
            CurrentCustomer.AddLikedTicket(ticket);
        }
        else
        {
```

```
            CurrentCustomer.RemoveLikedTicket(ticket);
        }
        var customerToUpdate = _customers.FirstOrDefault(c => c.NickName ==
CurrentCustomer.NickName);
        if (customerToUpdate != null)
        {
            customerToUpdate.LikedTickets = CurrentCustomer.LikedTickets;
        }

        SaveChanges();
    }
    public void SaveChanges()
    {
        var serializer = new XmlSerializer(typeof(List<Customer>));
        using (var stream = File.Create(FilePath))
        {
            serializer.Serialize(stream, _customers);
        }
    }
  }
}
```

Клас Payment.cs

```
namespace kursach.Model
{
   internal class Payment
   {
      public string Number { get; set; }
      public int? Code { get; set; }
      public string Period { get; set; }
      public string OwnerName { get; set; }
      public string Adress { get; set; }
   }
}
```

Клас Ticket.cs

```
namespace kursach.Model
{
```

```
    public class Ticket
    {
        public string Destination { get; set; }
        public DateTime Date { get; set; }
        public string HotelName { get; set; }
        public string HotelImage { get; set; }
        public string Period { get; set; }
        public string TicketAvailability { get; set; }
        public string Description { get; set; }
        public string Price { get; set; }
        public string Departure { get; set; }
    }
}
```

Додаток В

Класи ViewModel

Клас CustomerViewModel.cs

```
 namespace kursach.ViewModel
{
   public class CustomerViewModel : INotifyPropertyChanged
   {
      private readonly CustomerService _repository = new CustomerService();
      private string _firstName;
      private string _secondName;
      private string _nickName;
      private string _phoneNumber;
      private string _password;
      private string _avatarPath;
      public string NickName
      {
         get => _nickName;
         set { _nickName = value; OnPropertyChanged(); }
      }
      public string Password
{
         get => _password;
         set { _password = value; OnPropertyChanged(); }
      }
      public string FirstName
      {
         get => _firstName;
         set { _firstName = value; OnPropertyChanged(); }
```

```
}
public string SecondName
{
    get => _secondName;
    set { _secondName = value; OnPropertyChanged(); }
}
public string PhoneNumber
{
    get => _phoneNumber;
    set { _phoneNumber = value; OnPropertyChanged(); }
}
public string PhoneNumberProfile
{
    get
    {
        if (CustomerService.CurrentCustomer != null)
        {
            return CustomerService.CurrentCustomer.PhoneNumber;
        }
        else { return ""; }
    }
    set
    {
        if (_phoneNumber != value)
        {
            _phoneNumber = value;
```

```
            OnPropertyChanged();
        }
    }
}
public string NickNameProfile
{
    get
    {
        if (CustomerService.CurrentCustomer != null)
        {
            return CustomerService.CurrentCustomer.NickName;
        }
        else { return ""; }
    }
    set
    {
        if (_phoneNumber != value)
        {
            _phoneNumber = value;
            OnPropertyChanged();
        }
    }
}
public string FirstNameProfile
{
    get
```

```
        {
            if (CustomerService.CurrentCustomer != null)
            {
                return CustomerService.CurrentCustomer.FirstName;
            }
            else { return ""; }
        }
        set
        {
            if (_phoneNumber != value)
            {
                _phoneNumber = value;
                OnPropertyChanged();
            }
        }
    }
    public string SecondNameProfile
    {
        get
        {
            if (CustomerService.CurrentCustomer != null)
            {
                return CustomerService.CurrentCustomer.SecondName;
            }
            else { return ""; }
        }
```

```
        set
        {
            if (_phoneNumber != value)
            {
                _phoneNumber = value;
                OnPropertyChanged();
            }
        }
    }
    public string AvatarPath
    {
        get => _avatarPath;
        set
        {
            _avatarPath = value;
            OnPropertyChanged();
        }
    }
    public ICommand LoginCommand { get; }
    public ICommand NavigateToLoginOrProfileCommand { get; }
    public ICommand NavigateToRegisterCommand {  get; }
    public ICommand NavigateToMainCommand { get; }
    public ICommand RegisterCommand {  get; }
    public ICommand LogOutCommand {  get; }
    public ICommand NavigateToFavoriteCommand { get; }
    public ICommand ChangeAvatarCommand { get; }
```

```
        public ICommand NavigateToInfoCommand { get; }

        public CustomerViewModel()

        {

            LoginCommand = new RelayCommand(Login);

            NavigateToLoginOrProfileCommand = new
RelayCommand(NavigateToLoginOrProfile);

            NavigateToRegisterCommand = new RelayCommand(NavigateToRegister);

            RegisterCommand = new RelayCommand(Register);

            NavigateToMainCommand = new RelayCommand(NavigateToMain);

            LogOutCommand = new RelayCommand(LogOutAndLogin);

            NavigateToFavoriteCommand = new RelayCommand(NavigateToFavorite);

            ChangeAvatarCommand = new RelayCommand(ChangeAvatar);

            NavigateToInfoCommand = new RelayCommand(NavigateToInfo);

            LoadUserAvatar();

        }

        public void NavigateToInfo(object parametr)

        {

            if (Application.Current.MainWindow is MainWindow mainWindow)

            {

                mainWindow.MyFrame.Navigate(new Uri("/View/Info.xaml",
UriKind.Relative));

            }

        }

        private void NavigateToFavorite(object parametr)

        {

            if (CustomerService.IsUserLoggedIn == true)

            {
```

```
            if (Application.Current.MainWindow is MainWindow mainWindow)
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/favorite.xaml",
UriKind.Relative));
            }
        }
        else
        {
            if (Application.Current.MainWindow is MainWindow mainWindow)
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/login.xaml",
UriKind.Relative));
            }
        }
    }
    private void LogOutAndLogin(object parametr)
    {
        CustomerService.IsUserLoggedIn = false;
        if (Application.Current.MainWindow is MainWindow mainWindow)
        {
            mainWindow.MyFrame.Navigate(new Uri("/View/login.xaml",
UriKind.Relative));
        }
    }
    private void NavigateToLoginOrProfile(object parameter)
    {
        if (CustomerService.IsUserLoggedIn == true)
```

```
        {
            if (Application.Current.MainWindow is MainWindow mainWindow)
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/profile.xaml",
UriKind.Relative));
            }
        }
        else
        {
            if (Application.Current.MainWindow is MainWindow mainWindow)
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/login.xaml",
UriKind.Relative));
            }
        }
    }
    private void NavigateToRegister(object parameter)
    {
        if (Application.Current.MainWindow is MainWindow mainWindow)
        {
            mainWindow.MyFrame.Navigate(new Uri("/View/registration.xaml",
UriKind.Relative));
        }
    }
    public void NavigateToMain(object parameter)
    {
        if (Application.Current.MainWindow is MainWindow mainWindow)
```

```
        {
            mainWindow.MyFrame.Navigate(new Uri("/View/main.xaml",
UriKind.Relative));
        }
    }
    private void Login(object parameter)
    {
        if (_repository.ValidateUserCredentials(NickName, Password))
        {
            CustomerService.IsUserLoggedIn = true;
            MessageBox.Show("Вітаємо!", "Info", MessageBoxButton.OK,
MessageBoxImage.Information);
            if (Application.Current.MainWindow is MainWindow mainWindow)
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/profile.xaml",
UriKind.Relative));
            }
        }
        else
        {
            MessageBox.Show("Логін або пароль невірний", "Error",
MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    private void Register (object parameter)
    {
        if (!ValidateFirstName(FirstName) || !ValidateLastName(SecondName) ||
!ValidatePhoneNumber(PhoneNumber) || !ValidateNickName(NickName) ||
!ValidatePassword(Password))
```

```
        {
            return;
        }
        var existingCustomer = _repository.GetByNickName(NickName);
        if (existingCustomer != null)
        {
            MessageBox.Show("Логін зайнятий", "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }
        var customer = new Customer(FirstName, SecondName, PhoneNumber,
NickName, Password);
        _repository.SaveCustomer(customer);
      MessageBox.Show("Регістрація успішна!", "Info", MessageBoxButton.OK,
MessageBoxImage.Information);
       CustomerService.IsUserLoggedIn = false;
       if (Application.Current.MainWindow is MainWindow mainWindow)
       {
          mainWindow.MyFrame.Navigate(new Uri("/View/login.xaml",
UriKind.Relative));
       }
   }
   private bool ValidateFirstName(string firstName)
   {
       if (string.IsNullOrWhiteSpace(firstName))
       {
          MessageBox.Show("Ім'я не може бути порожнім", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
          return false;
       }

       if (firstName.Length > 25)
       {
          MessageBox.Show("Ім'я не повинно перевищувати 25 символів",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
          return false;
       }
```

```
        if (!System.Text.RegularExpressions.Regex.IsMatch(firstName, @"^[a-zA-Za-яА-ЯіІїЇєЄ]+$"))
        {
            MessageBox.Show("Ім'я повинно містити лише літери", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        return true;
    }
    private bool ValidateLastName(string lastName)
    {
        if (string.IsNullOrWhiteSpace(lastName))
        {
            MessageBox.Show("Прізвище не може бути порожнім", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (lastName.Length > 25)
        {
            MessageBox.Show("Прізвище не повинно перевищувати 25 символів", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (!System.Text.RegularExpressions.Regex.IsMatch(lastName, @"^[a-zA-Za-яА-ЯіІїЇєЄ]+$"))
        {
            MessageBox.Show("Прізвище повинно містити лише літери", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        return true;
    }
    private bool ValidatePhoneNumber(string phoneNumber)
    {
        if (string.IsNullOrWhiteSpace(phoneNumber))
        {
            MessageBox.Show("Номер телефону не може бути порожнім", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (!phoneNumber.StartsWith("+"))
        {
```

```
            MessageBox.Show("Номер телефону повинен починатися з '+'",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        string digitsOnly = new string(phoneNumber.Where(char.IsDigit).ToArray());
        if (digitsOnly.Length > 20)
        {
            MessageBox.Show("Номер телефону не повинен перевищувати 20 цифр",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        return true;
    }
    private bool ValidateNickName(string nickName)
    {
        if (string.IsNullOrWhiteSpace(nickName))
        {
            MessageBox.Show("Нік не може бути порожнім", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (nickName.Length > 20)
        {
            MessageBox.Show("Нік не повинен перевищувати 20 символів",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (nickName.Length < 6)
        {
            MessageBox.Show("Нік повинен бути не менше 6 символів", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (!System.Text.RegularExpressions.Regex.IsMatch(nickName, @"^[a-zA-Z0-
9]+$"))
        {
            MessageBox.Show("Нік повинен містити лише англійські літери та
цифри", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }
        if (_repository.GetByNickName(nickName) != null)
        {
```

```
            MessageBox.Show("Такий нік вже існує", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }
        return true;
    }
    private bool ValidatePassword(string password)
    {
        if (string.IsNullOrWhiteSpace(password))
        {
            MessageBox.Show("Пароль не може бути порожнім", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (password.Length > 20)
        {
            MessageBox.Show("Пароль не повинен перевищувати 20 символів",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (!System.Text.RegularExpressions.Regex.IsMatch(password, @"^[a-zA-Z0-
9]+$"))
        {
            MessageBox.Show("Пароль повинен містити лише англійські літери та
цифри", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }

        if (password.Length < 6)
        {
            MessageBox.Show("Пароль повинен бути не менше 6 символів",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }
        return true;
    }
    private void LoadUserAvatar()
    {
        if (CustomerService.CurrentCustomer == null)
        {
            AvatarPath = "pack://application:,,,/Images/Без_названия-removebg-
preview.png";
        }
        else if (string.IsNullOrEmpty(CustomerService.CurrentCustomer.AvatarPath))
```

```
        {
            AvatarPath = "pack://application:,,,/Images/Без_названия-removebg-
preview.png";
        }
        else
        {
            AvatarPath = CustomerService.CurrentCustomer.AvatarPath;
        }
    }
    private void ChangeAvatar(object parameter)
    {
        var openFileDialog = new OpenFileDialog
        {
            Filter = "Image files (*.png;*.jpeg;*.jpg)|*.png;*.jpeg;*.jpg|All files
(*.*)|*.*",
            Title = "Виберіть картинку"
        };

        if (openFileDialog.ShowDialog() == true)
        {
            string selectedPath = openFileDialog.FileName;
            string avatarsDirectory =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Аватари",
CustomerService.CurrentCustomer.NickName);
            if (!System.IO.Directory.Exists(avatarsDirectory))
            {
                System.IO.Directory.CreateDirectory(avatarsDirectory);
            }
            string fileName = $"аватар{System.IO.Path.GetExtension(selectedPath)}";
            string destinationPath = System.IO.Path.Combine(avatarsDirectory,
fileName);
            try
            {
                System.IO.File.Copy(selectedPath, destinationPath, true);
                AvatarPath = destinationPath;
                if (CustomerService.CurrentCustomer != null)
                {
                    CustomerService.CurrentCustomer.AvatarPath = destinationPath;

_repository.UpdateCustomerAvatar(CustomerService.CurrentCustomer.NickName,
destinationPath);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Не вдалося завантажити аватар: {ex.Message}",
```

```
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string propertyName =
null)
    {
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }

}

}
```

Клас TicketViewModel.cs

```
namespace kursach.ViewModel

 {

public class TicketViewModel : INotifyPropertyChanged

{

private DateTime? _startDateFilter;

 private string _destinationFilter;

 private string _periodFilter;

 private string _departureFilter;

private ObservableCollection _tickets;

private ObservableCollection _filteredTickets;

 private static ObservableCollection _likedTickets;

 private static Customer _lastCustomer;

 public ObservableCollection PossibleDestinations

{ get; private set; }

 public ObservableCollection PossiblePeriod
```

```
{ get; private set; }

public ObservableCollection PossibleDepartures

{ get; private set; }

 public ObservableCollection Tickets

 { get => _tickets; set { _tickets = value; OnPropertyChanged(); } }

public ObservableCollection FiltertedTickets

{ get => _filteredTickets; set { _filteredTickets = value; OnPropertyChanged(); } }

public DateTime? StartDateFilter { get => _startDateFilter; set { _startDateFilter =
value; OnPropertyChanged(); } }

 public string DestinationFilter { get => _destinationFilter; set { _destinationFilter =
value; OnPropertyChanged(); } } public string PeriodFilter { get => _periodFilter; set
{ _periodFilter = value; OnPropertyChanged(); } }

 public string DepartureFilter { get => _departureFilter; set { _departureFilter =
value; OnPropertyChanged(); } }

public ObservableCollection LikedTickets

{ get

{ if

(_lastCustomer != CustomerService.CurrentCustomer)

 { _likedTickets = CustomerService.IsUserLoggedIn &&
CustomerService.CurrentCustomer != null ? new
ObservableCollection(CustomerService.CurrentCustomer.LikedTickets) : new
ObservableCollection();

 _lastCustomer = CustomerService.CurrentCustomer; } return _likedTickets; } set {
_likedTickets = value; OnPropertyChanged();

 }

 }

public ICommand ApplyFiltersCommand { get; }

public ICommand BuyCommand { get; }

public ICommand ResetFiltersCommand { get; }

public ICommand LikeCommand { get; private set; }
```

public ICommand RemoveFromLikedCommand { get; private set; } public TicketViewModel() { Tickets = new ObservableCollection { new Ticket { Destination = "Egypt", Date = new DateTime(2024, 12, 6), Description = "Grand Blue Saint Maria Aqua Park 3\n*Єгипет", HotelImage = "/Images/egipt.jpg", Period = "Період: 8 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "28 236 UAH", Departure = "Warsaw", HotelName = "Grand Blue Saint Maria Aqua Park 3*" }, new Ticket { Destination = "Paris", Date = new DateTime(2024, 12, 10), Description = "Le Grand Hotel Paris 4\nФранція, Париж", HotelImage = "/Images/Le Grand Hotel Paris 4.jpg", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "32 450 UAH", Departure = "London", HotelName = "Le Grand Hotel Paris" }, new Ticket { Destination = "Rome", Date = new DateTime(2025, 1, 15), Description = "Roma Palace Hotel 4\nІталія, Рим", HotelImage = "/Images/Roma Palace Hotel 4.jpg", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "30 800 UAH", Departure = "Berlin", HotelName = "Roma Palace Hotel" }, new Ticket { Destination = "Bali", Date = new DateTime(2024, 12, 20), Description = "Bali Paradise Resort 5\nІндонезія, Балі", HotelImage = "/Images/Bali Paradise Resort 5.jpg", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "45 000 UAH", Departure = "Warsaw", HotelName = "Bali Paradise Resort 5" }, new Ticket { Destination = "Barcelona", Date = new DateTime(2024, 12, 12), Description = "Catalonia Plaza Hotel 4\nІспанія, Барселона", HotelImage = "/Images/Catalonia Plaza Hotel 4.jpg", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "35 200 UAH", Departure = "London", HotelName = "Catalonia Plaza Hotel 4" }, new Ticket { Destination = "New York", Date = new DateTime(2025, 1, 5), Description = "Manhattan Grand Hotel 5\nСША, Нью-Йорк", HotelImage = "/Images/Manhattan Grand Hotel 5.jpg", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "50 000 UAH", Departure = "Paris", HotelName = "Manhattan Grand Hotel 5" }, new Ticket { Destination = "Venice", Date = new DateTime(2025, 2, 14), Description = "Venetian Palace Hotel 4\nІталія, Венеція", HotelImage = "/Images/Venetian Palace Hotel 4.jpg", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "31 500 UAH", Departure = "Berlin", HotelName = "Venetian Palace Hotel" }, new Ticket { Destination = "Amsterdam", Date = new DateTime(2024, 12, 22), Description = "Amsterdam Art Hotel 4\nНідерланди, Амстердам", HotelImage = "/Images/Amsterdam Art Hotel 4.jpg", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "29 900 UAH", Departure = "Paris", HotelName = "Amsterdam Art Hotel" }, new Ticket { Destination = "Antalya", Date = new DateTime(2024, 12, 25), Description = "Antalya Beach Resort 5\nТурція, Анталія", HotelImage = "/Images/Antalya Beach Resort 5.png", Period = "Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "27 800 UAH", Departure = "Warsaw", HotelName = "Antalya Beach Resort 5" }, new Ticket { Destination = "Split", Date = new DateTime(2025, 3, 1), Description = "Split Sunset Hotel 4*\nХорватія, Спліт", HotelImage = "/Images/Split Sunset Hotel 4.png", Period =

"Період: 7 ночей", TicketAvailability = "Наявність квитків: ✓", Price = "28 600 UAH", Departure = "London", HotelName = "Split Sunset Hotel" } };

FiltertedTickets = new ObservableCollection(Tickets); PossibleDestinations = new ObservableCollection(Tickets.Select(t => t.Destination).Distinct());

PossibleDepartures = new ObservableCollection(Tickets.Select(t => t.Departure).Distinct());

PossiblePeriod = new ObservableCollection(Tickets.Select(t => t.Period).Distinct());

ApplyFiltersCommand = new RelayCommand(ApplyFilters); BuyCommand = new RelayCommand(Buy); LikeCommand = new RelayCommand(LikeTicket); RemoveFromLikedCommand = new RelayCommand(RemoveFromLiked); ResetFiltersCommand = new RelayCommand(ResetFilters); }

public void ApplyFilters()

{

var filtered = Tickets.Where(v => (!StartDateFilter.HasValue || v.Date >= StartDateFilter.Value) && (string.IsNullOrEmpty(PeriodFilter) || v.Period.Equals(PeriodFilter, StringComparison.OrdinalIgnoreCase)) && (string.IsNullOrEmpty(DestinationFilter) || v.Destination.Equals(DestinationFilter, StringComparison.OrdinalIgnoreCase)) && (string.IsNullOrEmpty(DepartureFilter) || v.Departure.Equals(DepartureFilter, StringComparison.OrdinalIgnoreCase))).ToList();

```
    FiltertedTickets.Clear();
    foreach (var ticket in filtered)
    {
        FiltertedTickets.Add(ticket);
    }
}

public void ResetFilters()
{
    StartDateFilter = null;
    DestinationFilter = null;
    DepartureFilter = null;
    PeriodFilter = null;
    FiltertedTickets = new ObservableCollection<Ticket>(Tickets);
}
private void Buy(object parameter)
{
    if (CustomerService.IsUserLoggedIn == true)
    {
        if (Application.Current.MainWindow is MainWindow mainWindow)
```

```
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/buying.xaml",
UriKind.Relative));
            }
        }
        else
        {
            if (Application.Current.MainWindow is MainWindow mainWindow)
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/login.xaml",
UriKind.Relative));
            }
        }
    }
    private void LikeTicket(object parameter)
    {
        if (CustomerService.IsUserLoggedIn)
        {
            if (parameter is Ticket ticket && CustomerService.CurrentCustomer != null)
            {
                if (!LikedTickets.Any(t => t.HotelName == ticket.HotelName && t.Date
== ticket.Date))
                {
                    CustomerService.Instance.UpdateCurrentCustomerLikedTickets(ticket,
true);
                    LikedTickets.Add(ticket);
                    MessageBox.Show("Квиток додано до обраного!", "Добре",
MessageBoxButton.OK, MessageBoxImage.Information);
                }
                else
                {
                    MessageBox.Show("Цей квиток вже є в обраному!", "Інформація",
MessageBoxButton.OK, MessageBoxImage.Error);
                }
            }
        }
        else
        {
            if (Application.Current.MainWindow is MainWindow mainWindow)
            {
                mainWindow.MyFrame.Navigate(new Uri("/View/login.xaml",
UriKind.Relative));
            }
        }
    }
    private void RemoveFromLiked(object parameter)
```

```
    {
        if (CustomerService.IsUserLoggedIn && CustomerService.CurrentCustomer !=
null)
        {
            if (parameter is Ticket ticket)
            {
                CustomerService.Instance.UpdateCurrentCustomerLikedTickets(ticket,
false);
                LikedTickets.Remove(ticket);
                MessageBox.Show("Квиток видалено", "Добре", MessageBoxButton.OK,
MessageBoxImage.Information);
            }
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string propertyName =
null)
    {
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}


}
```

Додаток Г

Тестування додатку

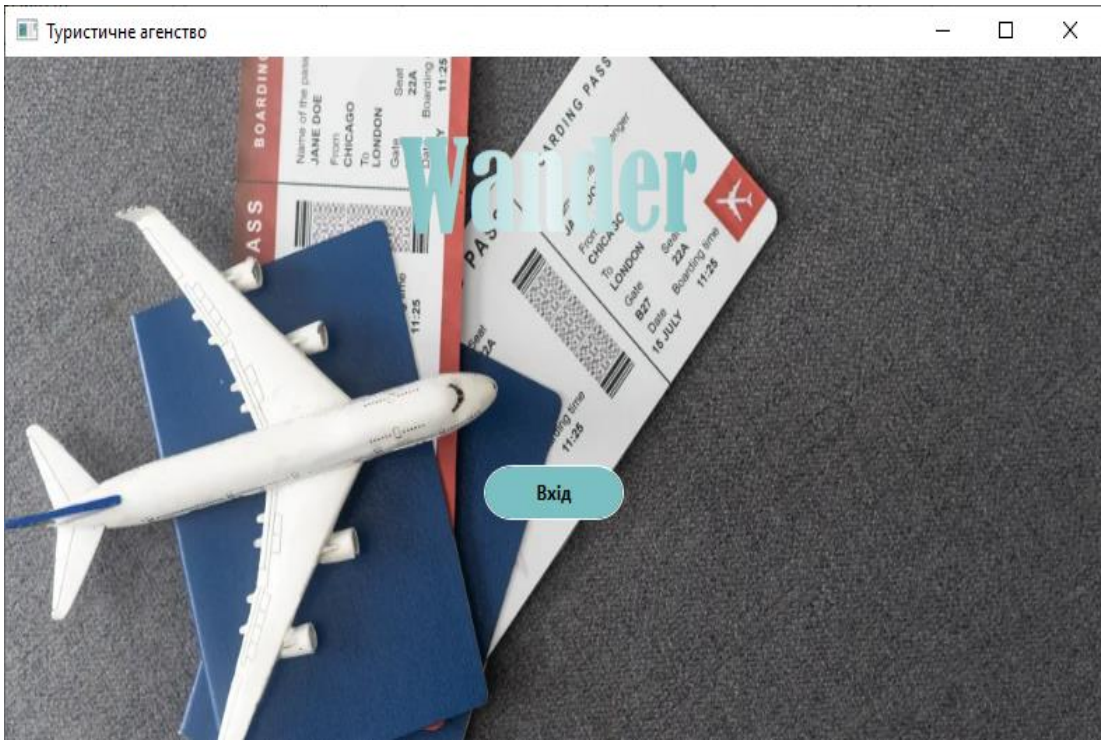

Рисунок 1- Перехідне меню MainWindow

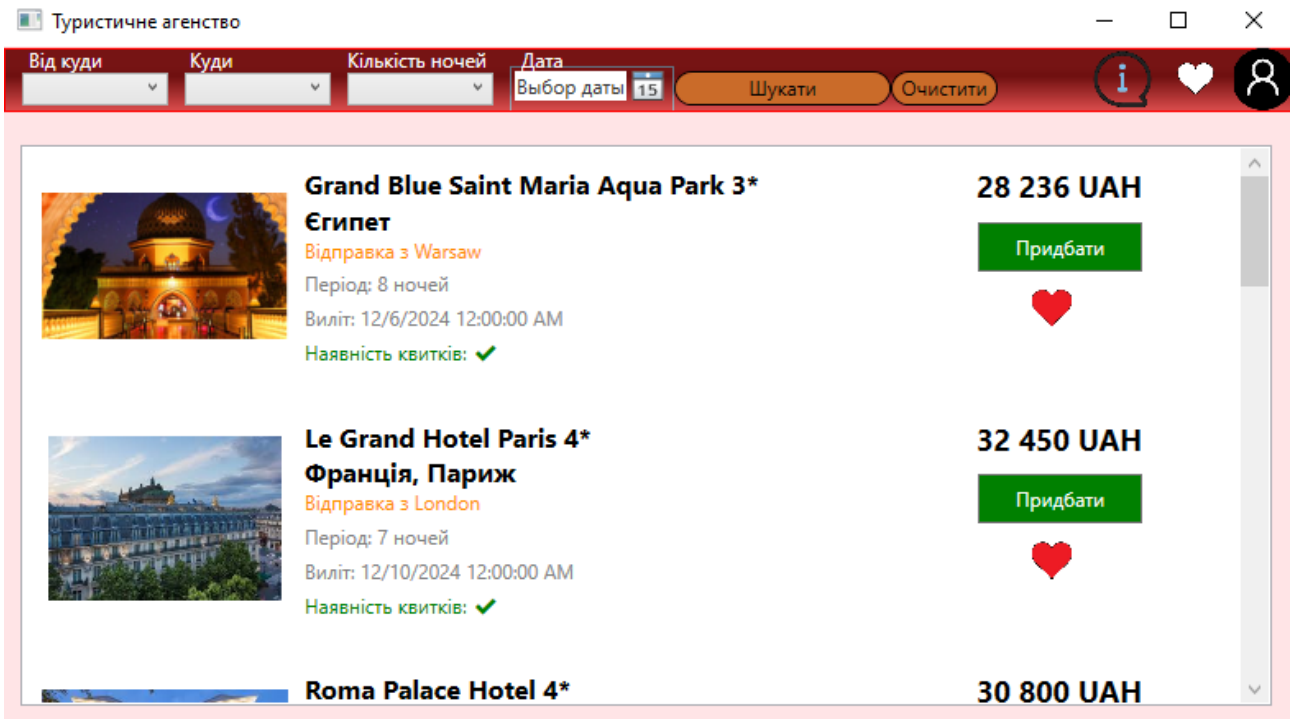

Рисунок 2 – Головне Меню

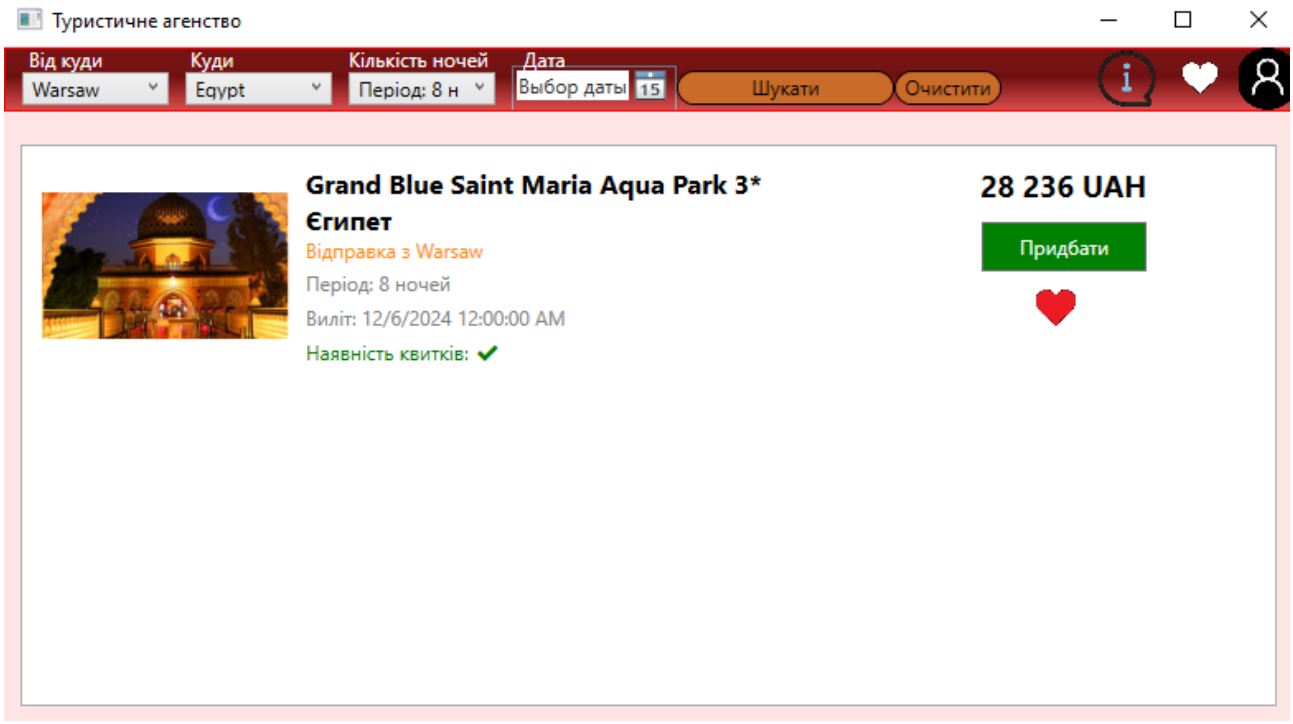

Рисунок 3 – Використання фільтра

-

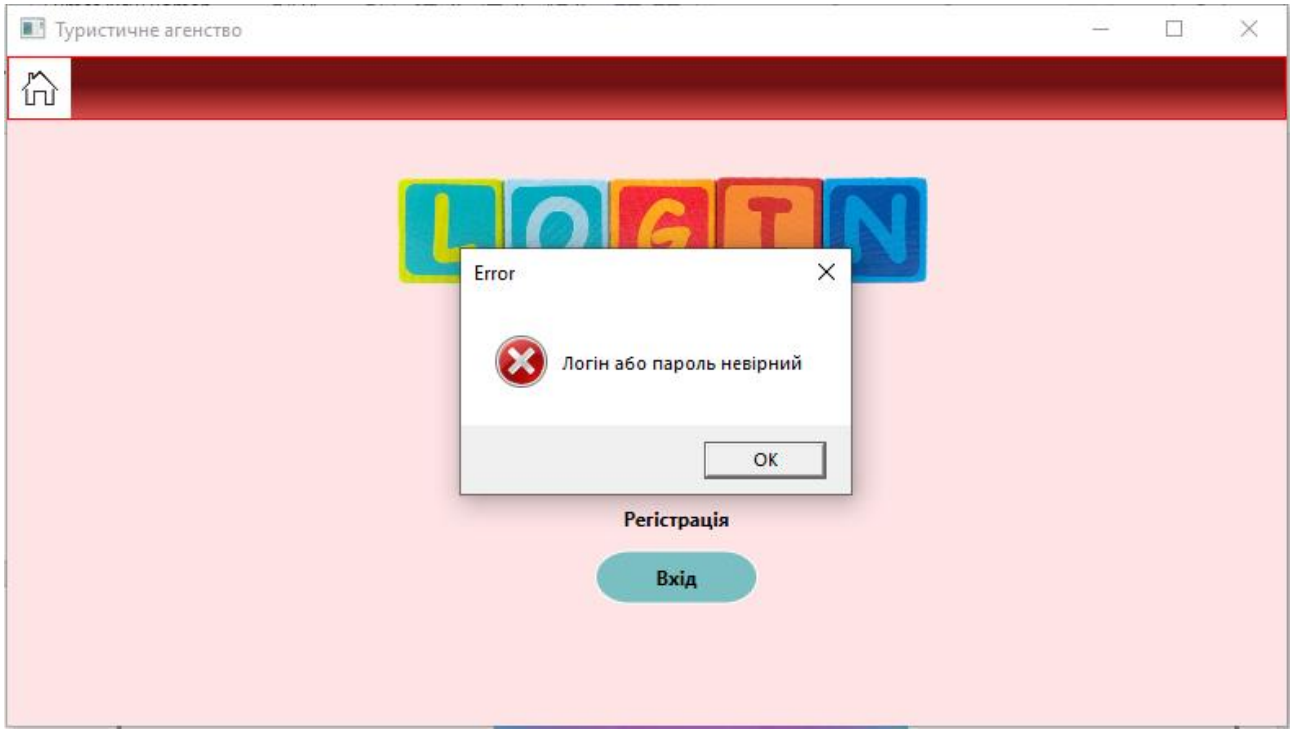

Рисунок 4 – Введення неправильних даних

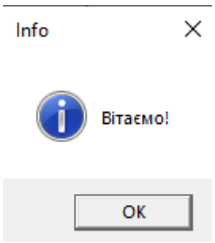

Рисунок 5 – Сповіщення правильної авторизації

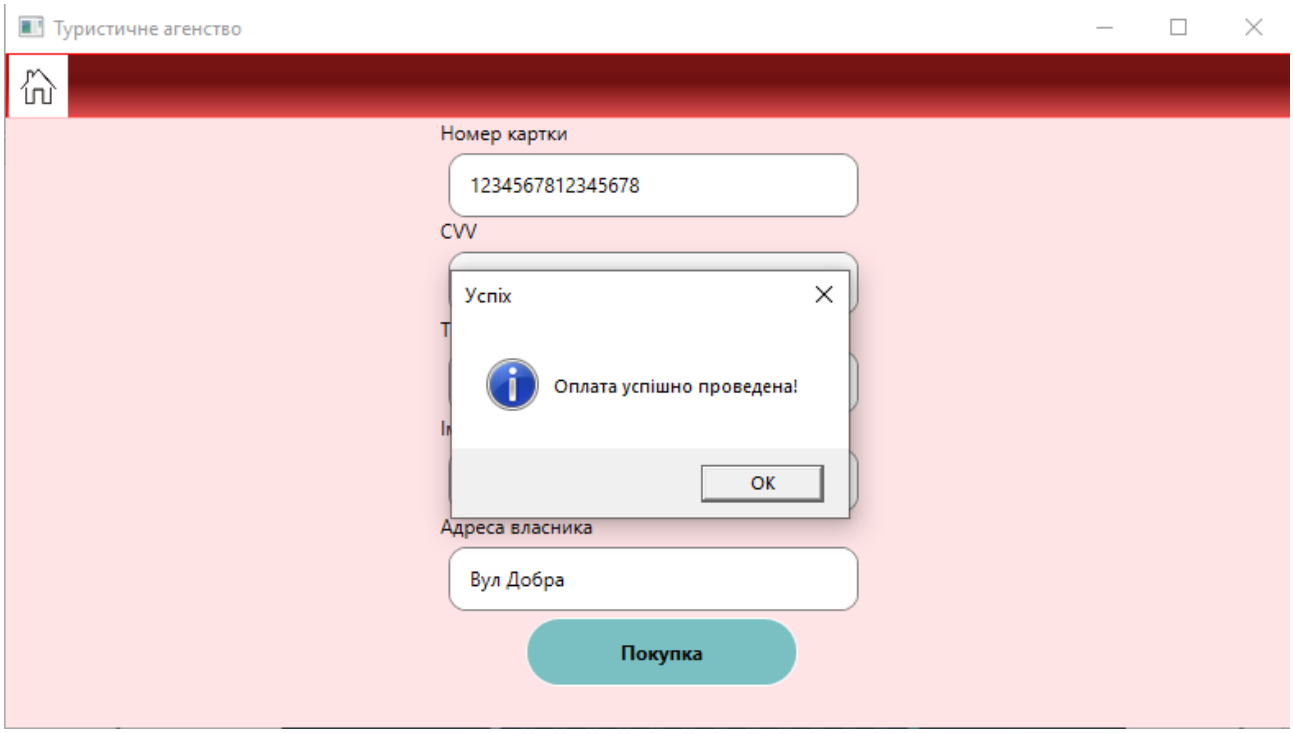

Рисунок 6 – Покупка