Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему «Розроблення програмного забезпечення для автоматизації роботи аеропорту»

Виконав: студент групи ІПЗ19-1
Спеціальність 121 Інженерія програмного забезпечення
Комишний Кирило Васильович
(прізвище та ініціали)
Керівник к.е.н., доц. Яковенко Т. Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
В.о. завідувача кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доц. Прокопович-Ткаченко Д.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2023

АНОТАЦІЯ

Комишний К.В. Розроблення програмного забезпечення для автоматизації роботи аеропорту.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення». – Університет митної справи та фінансів, Дніпро, 2023.

Насамперед, в кваліфікаційній роботі необхідно було провести детальний аналіз та розуміння вимог авіаційної індустрії, аеропорту та його підсистем. Такий аналіз дозволив визначити як функціональні, так й нефункціональні вимоги до програмного забезпечення.

На наступному етапі було розроблено архітектуру та дизайн програмного забезпечення на основі проведеного аналізу вимог. Було визначено модулі, компоненти, інтерфейси та алгоритми, а також обрано відповідні технології та платформи розробки.

Фактична розробка програмного забезпечення включала написання коду, створення баз даних та налаштування необхідних інтеграцій. Після завершення розробки було проведено ретельне тестування для перевірки працездатності, надійності та відповідності до вимог. Тестування було виконано як на тестових даних, так і на реальних даних, щоб переконатися в ефективності програмного забезпечення.

Результатом цієї кваліфікаційної роботи є програмне забезпечення для автоматизації роботи аеропорту. Отримані результати мають практичне значення, оскільки сприяють покращенню якості автоматизації роботи аеропорту. Розроблене програмне забезпечення дозволяє ефективніше й надійно керувати процесами в авіаційній індустрії, а також підвищити безпеку та якість обслуговування пасажирів.

Ключові слова: авіаційна індустрія, аеропорт, підсистеми, компоненти, бази даних, інтеграції, тестування, робоча ефективність, надійність, Windows Presentation Foundation, Entity Framework, LINQ to SQL, DataGrid.

ABSTRACT

Komishniy K. V. Development of software for airport automation.

Qualification work for a bachelor's degree in specialty 121 «Software Engineering». – University of Customs and Finance, Dnipro, 2023.

First of all, the qualification work required a detailed analysis and understanding of the requirements of the aviation industry, the airport and its subsystems. This analysis allowed us to identify both functional and non-functional software requirements.

At the next stage, we developed the software architecture and design based on the requirements analysis. Modules, components, interfaces, and algorithms were defined, and appropriate technologies and development platforms were selected.

The actual software development included writing code, creating databases, and setting up the necessary integrations. After the development was completed, thorough testing was conducted to verify performance, reliability, and compliance with the requirements. Testing was performed on both test data and real data to ensure that the software was effective.

The result of this qualification work is the airport automation software. The results are of practical importance as they contribute to improving the quality of airport automation. The developed software allows for more efficient and reliable management of processes in the aviation industry, as well as improving the safety and quality of passenger service.

Keywords: aviation industry, airport, subsystems, components, databases, integrations, testing, operational efficiency, reliability, Windows Presentation Foundation, Entity Framework, LINQ to SQL, DataGrid.

ЗМІСТ

ВСТУП

У світі авіаційна індустрія відіграє важливу роль у забезпеченні глобальної зв'язності та пересування людей і товарів. Один з ключових елементів ефективної та безпечної роботи аеропорту – це його автоматизація. Розробка програмного забезпечення для автоматизації роботи аеропорту стає все більш актуальним завданням у контексті зростаючого обсягу пасажирських та вантажних перевезень, а також суворих вимог щодо безпеки та оперативності.

Аеропорт – це складна система, що включає безліч процесів й підсистем, таких як пасажирські термінали, вантажно-розвантажувальні комплекси, системи навігації та контролю повітряного руху, управління багажем, безпека та багато іншого. Для забезпечення безперебійної та ефективної роботи всіх цих компонентів необхідно розробляти спеціалізоване програмне забезпечення, яке керує та координує всі процеси.

Розробка програмного забезпечення автоматизації роботи аеропорту має кілька цілей.

По-перше, вона спрямована на покращення операційної ефективності аеропорту, скорочення часу на оформлення пасажирів, обробку багажу та реєстрацію рейсів.

По-друге, автоматизація допомагає підвищити рівень безпеки, контролюючи доступ до обмежених зон та виявляючи потенційні загрози.

По-третє, вона сприяє зниженню операційних витрат та поліпшенню планування ресурсів, таких як персонал, техніка та матеріали.

Актуальність роботи з розробки програмного забезпечення автоматизації роботи аеропорту обумовлена кількома чинниками.

По-перше, авіаційна індустрія продовжує зростати та розвиватися, що призводить до збільшення обсягів пасажирських та вантажних перевезень. Автоматизація роботи аеропорту дозволяє справлятися з навантаженням, що

збільшується, і забезпечувати пасажирам та вантажам більш швидку й ефективну обробку.

По-друге, безпека є одним з основних пріоритетів в авіації. Розробка програмного забезпечення для автоматизації аеропорту дозволяє покращити системи контролю та безпеки, виявляти потенційні загрози та запобігати інцидентам.

По-третє, операційні витрати та ефективне використання ресурсів стають все більш значущими в сучасній авіаційній промисловості. Автоматизація роботи аеропорту дозволяє скоротити витрати на персонал, оптимізувати процеси та покращити планування ресурсів.

Крім того, розвиток нових технологій, таких як машинне навчання, інтернет речей та хмарні обчислення, надає нові можливості для створення більш інтелектуальних та гнучких систем автоматизації аеропорту.

Всі ці фактори роблять розробку програмного забезпечення для автоматизації роботи аеропорту актуальним та затребуваним завданням, сприяючи підвищенню ефективності, безпеки та покращенню загального пасажирського досвіду.

Метою роботи є розроблення програмного забезпечення для автоматизації роботи аеропорту.

Для досягнення поставленої мети перед кваліфікаційною роботою ставляться наступні завдання:

1) аналіз та розуміння вимог. Необхідно провести детальний аналіз вимог авіаційної індустрії, аеропорту та його підсистем. Це включає вивчення процесів, протоколів та стандартів, а також взаємодію із заінтересованими сторонами, такими як оператори, пасажири та служби безпеки. Це дозволить визначити функціональні та нефункціональні вимоги до програмного забезпечення;

2) проектування системи. На основі аналізу вимог необхідно розробити архітектуру та дизайн програмного забезпечення. Це включає визначення

модулів, компонентів, інтерфейсів та алгоритмів, а також вибір відповідних технологій та платформ розробки;

3) розробка та тестування. На цьому етапі відбувається фактична розробка програмного забезпечення, включаючи написання коду, створення баз даних та налаштування необхідних інтеграцій. Після завершення розробки слід провести ретельне тестування для перевірки працездатності, надійності та відповідності вимогам. Важливо також проводити тестування на реальних даних та реальних умовах, щоб переконатися в ефективності програмного забезпечення;

4) підтримка та оновлення. Розробка програмного забезпечення для автоматизації аеропорту – це безперервний процес. Після впровадження необхідно забезпечити постійну підтримку системи, реагувати на зворотний зв'язок користувачів, виправляти помилки та випускати оновлення для покращення функціональності та безпеки.

Об'єктом дослідження є процеси роботи програмного забезпечення для автоматизації роботи аеропорту.

Предметом дослідження є апаратно-програмне забезпечення для розроблення програмного забезпечення для автоматизації роботи аеропорту.

Практичне значення одержаних результатів полягає у підвищенні якості автоматизації роботи аеропорту.

Результатами роботи є програмне забезпечення для автоматизації роботи аеропорту.

Кваліфікаційна робота складається зі вступу, 3-х розділів, висновків, використаних джерел з 17 найменувань, 3-х додатків.

Обсяг роботи складається з 51 сторінки основного тексту з 72 сторінок кваліфікаційної роботи та 23 рисунків.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз різноманітних аспектів в контексті процесів та підсистем роботи аеропорту

Аналіз процесів та підсистем аеропорту включає вивчення та опис всіх основних діяльностей, систем і функцій, пов'язаних з роботою аеропорту. Цей аналіз має на меті оптимізацію процесів та підвищення ефективності роботи аеропорту.

Основні аспекти аналізу процесів та підсистем аеропорту можуть бути представлені наступним чином:

1) графік польотів та операцій. Один з основних аспектів роботи аеропорту – це планування та координація графіка польотів та операцій. Аналіз включає вивчення завантаження повітряного простору, наземних операцій, посадкових і злітних смуг, а також обслуговуючих служб. Метою аналізу є визначення ефективності використання ресурсів та виявлення вузьких місць у графіку польотів;

2) обслуговування пасажирів. Аеропорт повинен забезпечувати комфортне та безпечне обслуговування пасажирів. Аналіз процесів та підсистем включає вивчення процедур реєстрації, проходження митного та паспортного контролю, багажного обслуговування, безпеки та очікування рейсів. Оптимізація цих процесів може допомогти скоротити час, що витрачається пасажирами на проходження формальностей та покращити загальне враження від відвідування аеропорту;

3) робота злітно-посадкових смуг. Аналіз підсистеми злітно-посадкових смуг включає вивчення використання смуг, розклад їх обслуговування, час керування, системи безпеки та координації між повітряним трафіком та наземними службами. Метою аналізу є оптимізація використання смуг та зниження затримок при зльоті та посадці;

4) вантажні операції. В аеропортах також проводяться вантажні операції, пов'язані з перевезенням вантажів та пошти. Аналіз підсистеми вантажних операцій включає вивчення процесів завантаження та розвантаження, складування, митних процедур та документообігу. Метою аналізу є підвищення ефективності та безпеки вантажних операцій;

5) технічне обслуговування. Аеропорт також забезпечує технічне обслуговування повітряних суден. Аналіз підсистеми технічного обслуговування включає вивчення процесів приймання, обслуговування та видачі повітряних суден, а також управління запасними частинами та матеріалами. Метою аналізу є оптимізація використання ресурсів, покращення процесів обслуговування та мінімізація часу простою повітряних суден.

Слід зазначити, що подібний аналіз може бути докладнішим й включати інші аспекти, специфічні для конкретного аеропорту та його операцій.

1.2 Види програмних рішень для автоматизації роботи аеропорту та покращення його функціонування

Існує ряд програмних рішень, призначених для автоматизації роботи аеропорту та покращення його операцій.

Ось кілька прикладів такого програмного забезпечення:

1) системи управління операціями аеропорту (AODB – Airport Operational Database). Це програмне забезпечення використовується для централізованого управління та координації різними операціями в аеропорту. Воно забезпечує функції планування ресурсів, розкладу польотів, управління повітряним рухом, обробки пасажирів та вантажу, а також відстеження статусу операцій. Схема AODB зображена на рисунку 1.1;

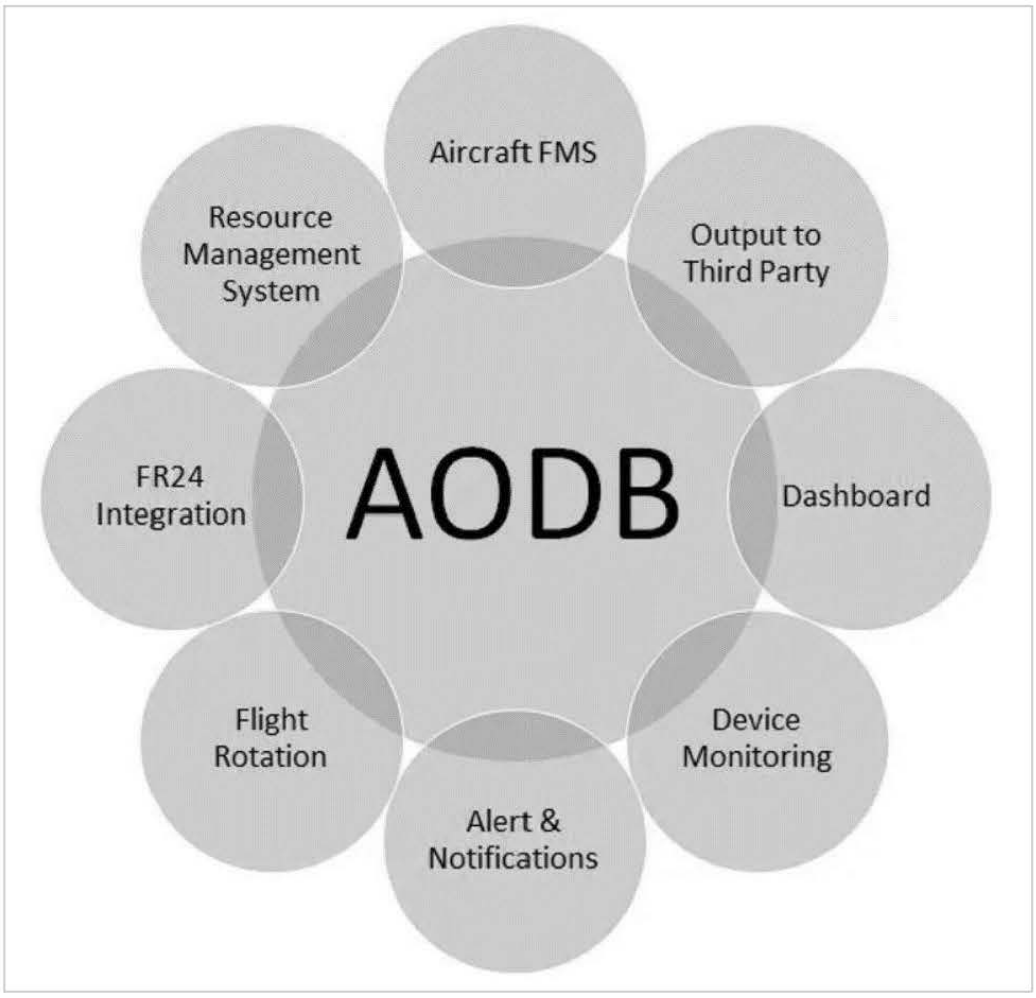

Рисунок 1.1 – Схема Airport Operational Database

2) системи керування багажем (BMS – Baggage Management System). Ці системи забезпечують автоматизовану обробку багажу пасажирів. Вони відстежують рух багажу від моменту реєстрації до моменту його видачі, використовуючи технології, такі як багажні конвеєри, зчитувачі штрих-кодів і RFID. Такі системи дозволяють скоротити час обробки багажу, мінімізувати помилки та підвищити його безпеку. Схема BMS зображена на рисунку 1.2;

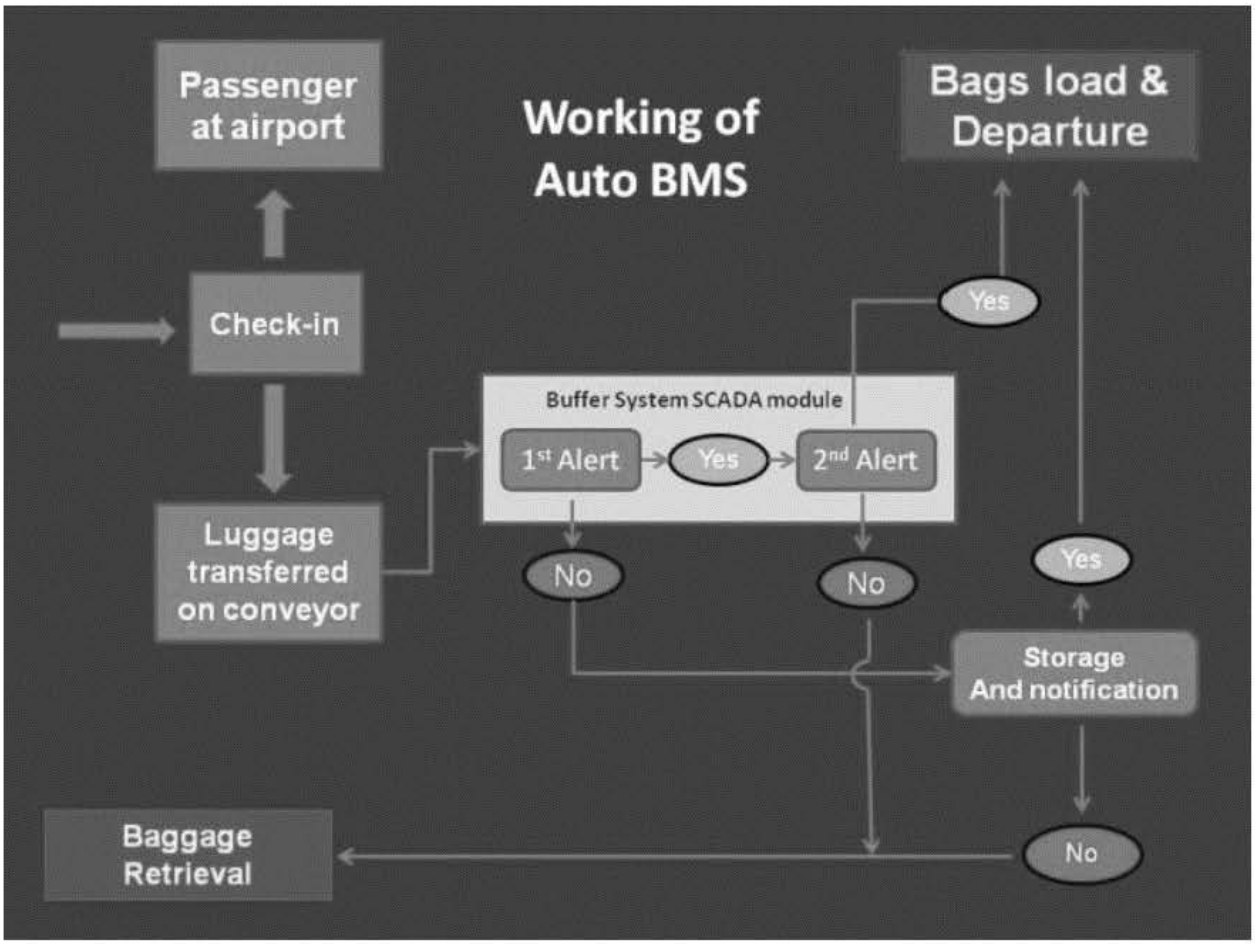

Рисунок 1.2 – Схема Baggage Management System

3) системи управління безпекою (SSMS – Security and Surveillance Management System). Ці системи надають засоби для моніторингу безпеки в аеропорту. Вони включають системи відеоспостереження, контролю доступу, виявлення вибухових речовин та інші технології, які допомагають забезпечити безпеку персоналу, пасажирів та аеропортових об'єктів;

4) системи управління повітряним рухом (ATM – Air Traffic Management). Це програмне забезпечення використовується для координації та управління повітряним рухом у межах аеропорту та повітряного простору. Воно включає системи навігації, радіозв'язку, планування польотів та диспетчерського управління, які допомагають забезпечити безпечний й ефективний рух літаків. Схема ATM зображена на рисунку 1.3;

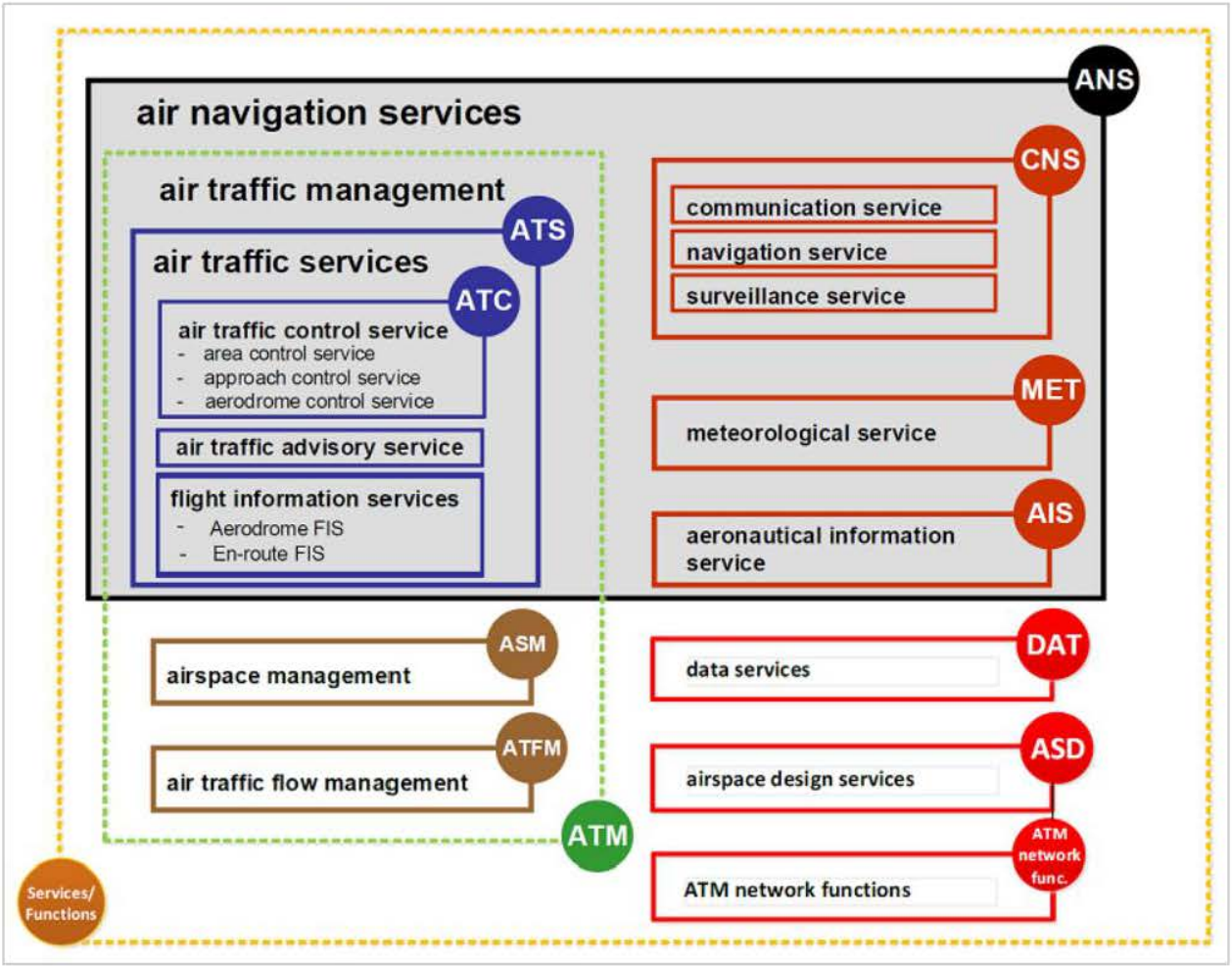

Рисунок 1.3 – Схема Air Traffic Management

5) пасажирські інформаційні системи (PIS – Passenger Information Systems). Ці системи надають інформацію та оновлення пасажирам про статус польотів, гейтів посадки, затримки, зміни розкладу та інших важливих даних. Вони можуть бути представлені у вигляді інформаційних табло, мобільних додатків чи інтерактивних інформаційних стендів, забезпечуючи пасажирам актуальну та надійну інформацію.

1.3 Огляд існуючого програмного забезпечення для автоматизації роботи аеропорту

Приклади реальних розробок програмного забезпечення, які широко використовуються аеропортами для автоматизації роботи представлені наступним чином:

– Amadeus Altéa. Комплексне програмне рішення для авіакомпаній, яке включає функції управління операціями на землі. Altéa надає інструменти для автоматизації бронювання та квитування пасажирів, управління ресурсами аеропорту, планування польотів та обробки багажу;

– SITA Airport Management. Це рішення пропонує набір інтегрованих модулів для керування операціями в аеропортах. Воно включає функції автоматизації обробки пасажирів, реєстрації, відстеження багажу, управління повітряним рухом, моніторингу безпеки та інші інструменти ефективної роботи аеропорту;

– ARINC AirPlan. Ця система управління операціями надає аеропортам централізований доступ до інформації про польоти, багаж, ресурси та операції на землі. AirPlan автоматизує процеси планування та розкладу польотів, управління ресурсами, відстеження багажу та обміну даними з іншими системами в аеропорту;

– Rockwell Collins ARINC AirDB. Це база даних, призначена для управління та відстеження операцій на землі в аеропорту. AirDB включає модулі для управління операціями пасажирів, багажу, вантажів, ресурсами і безпекою. Це дозволяє аеропортам ефективно планувати та координувати різні операції;

– RESA Airport Operations Management. Це рішення надає комплексний підхід до управління операціями в аеропорту. Воно включає модулі для планування та розкладу польотів, управління багажем, контролю безпеки, координації посадки та виходу літаків на злітно-посадкових смугах, а також моніторингу та звітності;

– Amadeus Airport Collaborative Decision Making (A-CDM). Ця система надає засоби для співпраці та прийняття узгоджених рішень між аеропортом, авіакомпаніями та контролюючими органами. A-CDM дозволяє учасникам оперативно обмінюватися інформацією про статус польотів, розклади, ресурси та інші дані, щоб знизити затримки, оптимізувати використання злітно-посадкових смуг та ресурсів аеропорту;

– SITA Airport Resource Management System (RMS). Ця система призначена для оптимізації використання ресурсів аеропорту, включаючи злітно-посадкові смуги, стоянки, обслуговуючі технічні служби та інші ресурси. RMS надає інструменти для планування, призначення та моніторингу ресурсів з урахуванням змін у розкладі польотів та потреб аеропорту;

– SITA AirportConnect Open. Це програмне забезпечення призначене для управління пасажирськими процесами та оптимізації роботи аеропортів. Воно забезпечує автоматизацію процесу реєстрації пасажирів, посадки на борт та видачі посадкових талонів, а також інтеграцію з іншими системами, такими як системи безпеки та системи обробки багажу;

– SITA Baggage Reconciliation System (BRS). Ця система призначена для автоматизації процесу відстеження та зіставлення багажу з пасажирами. BRS використовує технології зчитування штрих-кодів та RFID для забезпечення точності та ефективності обробки багажу, мінімізуючи помилки та час для звірки багажу з пасажирами.

Отже, вище були наведені реальні приклади програмного забезпечення, які використовуються для автоматизації роботи аеропортів.

1.4 Висновки до першого розділу. Постановка задач дослідження

Виходячи з першого розділу, можна зробити наступні висновки:

1) різноманітність аспектів автоматизації в аеропорту. Дослідження предметної області дозволяє зрозуміти, що автоматизація роботи аеропорту охоплює безліч різних аспектів, включаючи процеси та підсистеми. Це

включає обробку пасажирів, обробку багажу, керування повітряним рухом, забезпечення безпеки, керування ресурсами та інформаційне керування;

2) різноманітність програмних рішень. Аналіз різних видів програмних рішень для автоматизації роботи аеропорту показує, що є безліч варіантів і підходів. Вони можуть включати системи керування пасажирськими процесами, системи керування багажем, системи керування повітряним рухом, системи безпеки, системи керування ресурсами та інші;

3) існуюче програмне забезпечення. Огляд існуючого програмного забезпечення для автоматизації роботи аеропорту дозволяє оцінити вже доступні рішення на ринку. Це включає системи реєстрації та посадки пасажирів, системи відстеження та керування багажем, системи керування польотами, системи безпеки та інші;

4) постановка завдань дослідження. Висновки першого розділу допомагають визначити основні завдання та цілі подальшого дослідження. Це включає ідентифікацію конкретних потреб аеропорту, виявлення можливих поліпшень та оптимізації, а також розробку рекомендацій щодо вибору та впровадження програмного забезпечення для автоматизації роботи аеропорту.

Метою роботи є розроблення програмного забезпечення для автоматизації роботи аеропорту.

Для досягнення поставленої мети перед кваліфікаційною роботою ставляться наступні завдання:

1) аналіз та розуміння вимог. Необхідно провести детальний аналіз вимог авіаційної індустрії, аеропорту та його підсистем. Це включає вивчення процесів, протоколів та стандартів, а також взаємодію із заінтересованими сторонами, такими як оператори, пасажири та служби безпеки. Це дозволить визначити функціональні та нефункціональні вимоги до програмного забезпечення;

2) проектування системи. На основі аналізу вимог необхідно розробити архітектуру та дизайн програмного забезпечення. Це включає визначення

модулів, компонентів, інтерфейсів та алгоритмів, а також вибір відповідних технологій та платформ розробки;

3) розробка та тестування. На цьому етапі відбувається фактична розробка програмного забезпечення, включаючи написання коду, створення баз даних та налаштування необхідних інтеграцій. Після завершення розробки слід провести ретельне тестування для перевірки працездатності, надійності та відповідності вимогам. Важливо також проводити тестування на реальних даних та реальних умовах, щоб переконатися в ефективності програмного забезпечення;

4) підтримка та оновлення. Розробка програмного забезпечення для автоматизації аеропорту – це безперервний процес. Після впровадження необхідно забезпечити постійну підтримку системи, реагувати на зворотний зв'язок користувачів, виправляти помилки та випускати оновлення для покращення функціональності та безпеки.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ АЕРОПОРТУ

2.1 Вибір програмних засобів для розроблення програмного забезпечення для автоматизації роботи аеропорту

Для розробки програмного забезпечення для автоматизації роботи аеропорту з використанням WPF доцільно використати наступні технології та інструменти:

1) Windows Presentation Foundation (WPF). WPF є технологією розробки клієнтських додатків для Windows, яка надає розширені можливості для створення графічного інтерфейсу користувача. У даному випадку, WPF був використаний для створення графічного інтерфейсу користувача (GUI) для програмного забезпечення автоматизації аеропорту;

2) C#. Це об'єктно-орієнтована мова програмування, що використовується для розробки додатків під управлінням платформи .NET. В даному випадку, мова C# була використана для написання логіки програми, обробки подій, взаємодії з базою даних та інших функцій;

3) Entity Framework, що є об'єктно-орієнтованим маппером даних для .NET, який надає можливість роботи з базами даних з використанням об'єктно-орієнтованого підходу. У даному випадку, Entity Framework був використаний для створення та управління базою даних, збереження та отримання даних пасажирів;

4) XAML (Extensible Application Markup Language). XAML є декларативною мовою розмітки, яка використовується для визначення графічного інтерфейсу користувача в WPF. У даному випадку, XAML був використаний для опису графічного інтерфейсу користувача, включаючи розміщення елементів, стилізацію, пов'язані дані та події;

5) DataGrid. У WPF є вбудований елемент керування DataGrid, який надає можливість відображати табличні дані та взаємодіяти з ними. В даному

випадку, елемент DataGrid був використаний для відображення списку пасажирів та їх атрибутів у таблиці;

6) LINQ (Language-Integrated Query). LINQ є мовно-інтегрованим запитом для .NET, який надає можливість здійснювати запити до даних в об'єктно-орієнтованому стилі. В даному випадку, LINQ був використаний для отримання та фільтрації даних з бази даних;

7) Visual Studio, що є інтегрованим середовищем розробки (IDE) для розробки програмного забезпечення на платформі .NET. В даному випадку, середа розробки Visual Studio була використана для створення та редагування проекту, написання коду, відлагодження та збірки програми.

Перелічені технології та інструменти допомогли розробити програмне забезпечення для автоматизації роботи аеропорту з використанням WPF та бази даних.

2.2 Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF) – це технологія розробки клієнтських програм для операційної системи Windows. Вона надає розробникам широкі можливості для створення графічного інтерфейсу користувача (GUI) з використанням привабливого дизайну, багатої візуалізації та інтерактивності.

WPF заснована на мові розмітки XAML (Extensible Application Markup Language), яка дозволяє декларативно описувати елементи інтерфейсу та об'єктної моделі .NET Framework. Поєднання XAML і .NET Framework дозволяє створювати складні та гнучкі додатки з легкістю.

Основні переваги використання WPF для створення ПЗ полягають у наступних аспектах [1]—[4]:

– гнучкий та потужний дизайн інтерфейсу. WPF надає широкий набір графічних елементів керування, які можуть бути налаштовані та стилізовані для досягнення бажаного візуального ефекту. Можна створювати унікальні

інтерфейси користувача з привабливим дизайном, анімаціями, ефектами і переходами;

– векторна графіка. WPF використовує векторну графіку для відображення елементів інтерфейсу. Це дозволяє створювати програми з високою якістю графіки, яка масштабується на різних пристроях та дозволах без втрати якості. Векторна графіка також дозволяє реалізовувати складні ефекти та анімації;

– поділ логіки та подання. WPF використовує модель поділу логіки та подання, що полегшує супровід та тестування коду. Це дозволяє створювати окремі класи для логіки програми та відображення елементів інтерфейсу, що сприяє повторному використанню коду та спрощує супровід програми;

– підтримка стилів і шаблонів. WPF надає потужні засоби для стилізації елементів інтерфейсу та створення шаблонів користувача. Завдяки цьому можна визначати стилі елементів управління, які застосовуються автоматично під час їх використання. Це в значній мірі спрощує завдання створення узгодженого дизайну та підвищує ефективність розробки;

– багата підтримка взаємодії. WPF надає широкий набір подій й можливостей для обробки взаємодії користувача. Завдяки цьому можна реагувати на різні дії користувача, такі як натискання кнопки, рух миші або введення тексту. WPF також підтримує мультитач-взаємодія, що дозволяє створювати програми адаптовані для сенсорних екранів;

– інтеграція з .NET Framework. WPF повністю інтегрована з .NET Framework й успадковує його переваги, такі як потужна система типів, багатопоточність, обробка виключень тощо. Можна використовувати мови програмування .NET, такі як C# або Visual Basic, для написання логіки застосування;

– WPF надає переваги, інструменти та можливості для створення сучасних та професійних інтерфейсів для Windows-додатків. Завдяки гнучкості, потужності та багатим можливостям, WPF залишається однією з

популярних технологій для розробки клієнтських програм на платформі Windows.

Створення простого вікна класу Window зображено на рисунку 2.1.

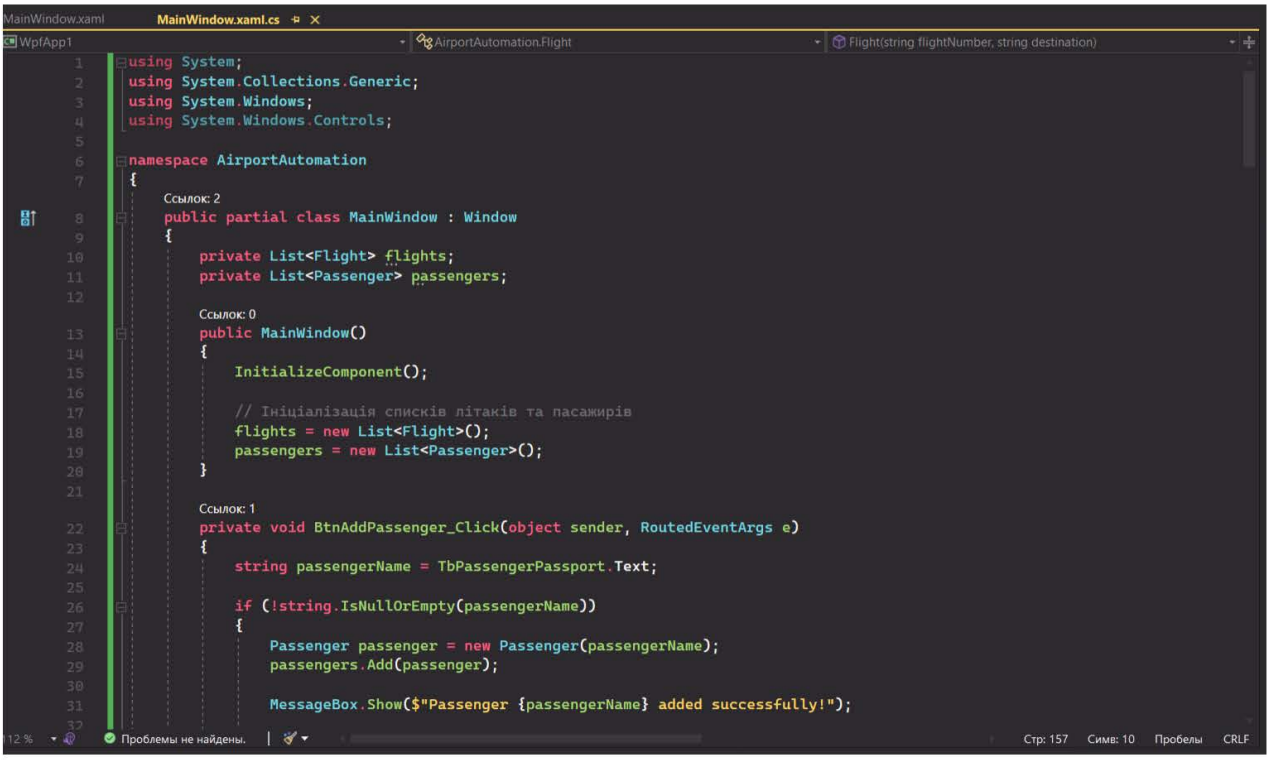

Рисунок 2.1 – Приклад створення вікна WPF

2.3 Мова програмування C# (C-Sharp)

C# (C-Sharp) – це сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. C# був випущений у 2000 році та є частиною платформи .NET. Він призначений для створення різних типів програм, включаючи програми Windows, веб-програми, мобільні програми, ігри та інші.

Основні риси та особливості C# [1]—[3], [5], [7]:

– об'єктно-орієнтоване програмування. C# повністю підтримує принципи об'єктно-орієнтованого програмування (ООП). Це дозволяє створювати класи, об'єкти, спадкування, поліморфізм та інкапсуляцію. ООП полегшує організацію коду, повторне використання та полегшення супроводу програм;

– сильна типізація. C# є мовою зі строгою типізацією, що означає, кожна змінна повинна мати певний тип даних, і всі операції над змінними перевіряються на відповідність типам. Сильна типізація підвищує безпеку та надійність програми, оскільки запобігає помилкам пов'язаним з типами даних;

– керований код. C# генерує керований код, який працює в середовищі виконання .NET, відомому як Common Language Runtime (CLR). CLR надає безліч можливостей, включаючи складання сміття, керування пам’яттю, обробку винятків, безпеку та інші. Керований код забезпечує високу безпеку та надійність додатків;

– багатопоточність. C# надає потужні можливості для роботи з багатопоточністю. Завдяки цьому можна створювати паралельні завдання, потоки виконання та використовувати синхронізацію для ефективної роботи з багатопоточністю. Це особливо корисно для розробки додатків, які вимагають обробки великих обсягів даних або асинхронних операцій;

– велика бібліотека класів. C# має велику бібліотеку класів, відому як .NET Framework. Ця бібліотека містить безліч готових класів та компонентів для виконання різних завдань, таких як робота з файлами, мережна взаємодія, бази даних, графіка, шифрування та багато іншого. Завдяки цьому, розробники можуть заощаджувати час та зусилля, використовуючи готові компоненти для своїх програм;

– широка платформна підтримка. C# і .NET Framework призначені для розробки програм для різних платформ, включаючи Windows, Linux та macOS. Існує також можливість використовувати C# для розробки програм для мобільних пристроїв за допомогою Xamarin, а також для веб-розробки за допомогою ASP.NET;

– інтеграція з Visual Studio. Visual Studio – це інтегроване середовище розробки (IDE), що надається компанією Microsoft, яке полегшує розробку C# додатків. Visual Studio надає редактор коду з підсвічуванням синтаксису, відладчик, інструменти для проектування інтерфейсу користувача, систему

складання та багато іншого. Це дозволяє збільшити продуктивність і ефективність розробки додатків на C#.

Отже, мова програмування C# – це потужна та гнучка мова програмування, яка широко використовується для створення різних типів додатків на платформі .NET. Вона має багатий набір функцій та інструментів, які роблять розробку програмного забезпечення більш ефективною та зручною для розробників.

2.4 Entity Framework (EF)

Entity Framework (EF) – це технологія об'єктно-реляційного відображення (ORM) у .NET Framework, яка надає програмістам зручний спосіб взаємодії з базами даних через об'єктно-орієнтований підхід. EF дозволяє розробникам працювати з даними, використовуючи об'єкти та запити замість явного написання SQL-запитів.

Основні концепції та можливості Entity Framework можна описати наступним чином [5]—[12]:

1) модель даних. Entity Framework ґрунтується на концепції моделі даних, яка описує структуру бази даних за допомогою класів та відносин між ними. Це дозволяє створювати класи, звані сутностями entities, які відповідають таблицям в базі даних, й визначати відносини між сутностями, такі як зв'язки один-до-одного, один-до-багатьом, а також багато-багатьом;

2) контекст даних. EF включає контекст даних (DbContext), який є посередником між об'єктами сутностей та базою даних. Контекст даних відстежує зміни в сутності та автоматично генерує SQL-запити для виконання операцій читання та запису даних до бази даних;

3) мова запитів LINQ. Entity Framework підтримує мову запитів LINQ (Language-Integrated Query), яка дозволяє розробникам писати виразні та типобезпечні запити до даних. LINQ інтегрований в C# й дозволяє

здійснювати запити до колекцій об’єктів, а також до даних у базі даних з використанням EF;

4) ліниве завантаження та попереднє завантаження. EF надає механізми для роботи з відкладеним завантаженням lazy loading та попереднім завантаженням eager loading пов’язаних даних. Ліниве завантаження дозволяє завантажувати дані тільки при доступі до них, тоді як попереднє завантаження дозволяє заздалегідь завантажити всі пов'язані дані для оптимізації запитів до бази даних;

5) міграції бази даних. Entity Framework дозволяє управляти змінами структури бази даних за допомогою механізму міграцій. Міграції дозволяють автоматично застосовувати зміни в моделі даних до бази даних, оновлювати схему бази даних без втрати даних та синхронізувати модель та базу даних;

6) підтримка різних баз даних. Entity Framework підтримує різноманітні особисті системи управління базами даних (СУБД), включаючи Microsoft SQL Server, MySQL, Oracle, PostgreSQL та інші. При цьому можна використовувати EF для роботи з різними типами баз даних без необхідності вивчення специфічних мов та інструментів кожної СУБД.

7) Code First підхід. Entity Framework підтримує підхід Code First, який дозволяє визначити модель даних з використанням класів і атрибутів C#. При цьому EF автоматично створить відповідну базу даних на основі певної моделі. Цей підхід спрощує розробку та забезпечує гнучкість при зміні моделі даних;

8) Database First підхід. Entity Framework також підтримує підхід «Database First», у якому модель даних створюється з урахуванням існуючої бази даних. При цьому можна використовувати інструменти EF для генерації класів сутностей та контексту даних, що відповідають структурі бази даних;

9) Model First підхід. EF підтримує ще один підхід, відомий як «Model First». У разі використання цього підходу можна створювати модель даних за допомогою графічного інструменту проектування, а потім EF автоматично створить відповідну базу даних та класи сутностей;

10) транзакції та управління змінами. Entity Framework надає можливості для роботи з транзакціями та управління змінами у базі даних. Це дозволяє виконувати різноманітні операції з даними у межах транзакцій, забезпечуючи цілісність даних. EF також надає механізми для відстеження змін, порівняння версій даних та реалізації песимістичного та оптимістичного блокування;

11) інтеграція з іншими технологіями. Entity Framework добре інтегрується з іншими технологіями та інструментами .NET, такими як ASP.NET, Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), WinForms та іншими. При цьому можна використовувати EF разом із цими технологіями для розробки повнофункціональних додатків.

12) Entity Framework надає потужні інструменти для роботи з даними та спрощує взаємодію з базами даних у .NET додатках. Завдяки його гнучкості, підходам Code First, Database First і Model First, а також підтримці різних СУБД, EF є одним з найбільш популярних і широко використовуваних ORM-фреймворків в екосистемі .NET.

Схема використання Entity Framework зображена на рисунку 2.2.

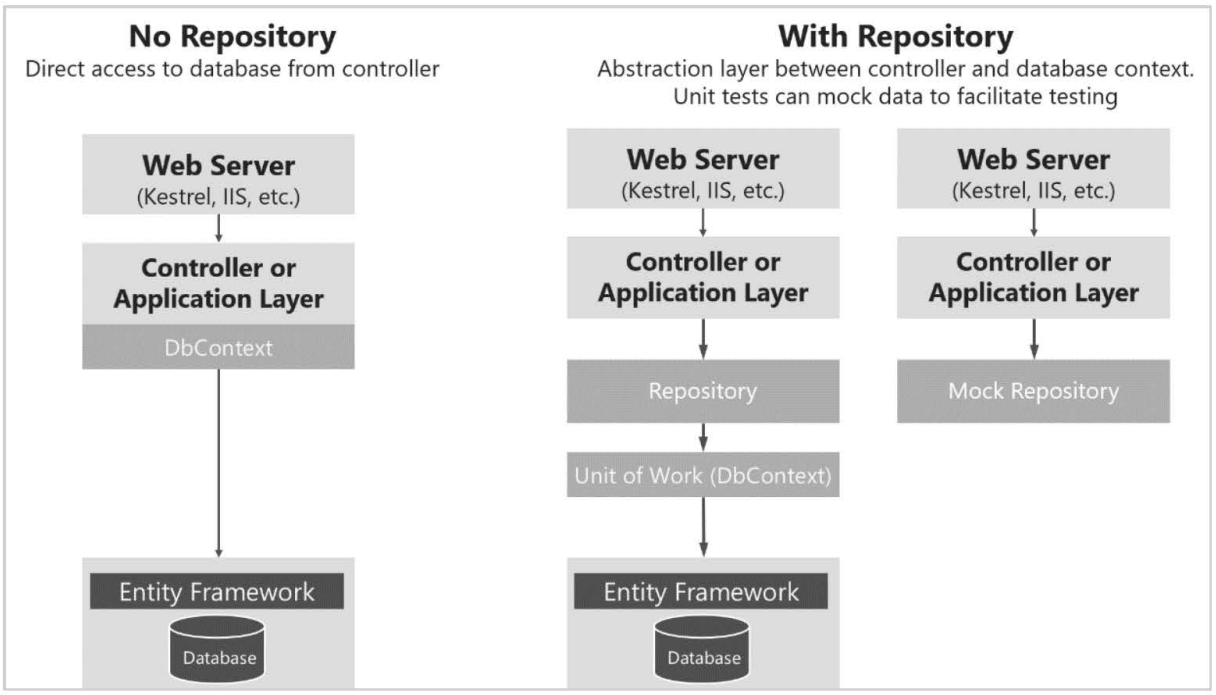

Рисунок 2.2 – Схема використання Entity Framework

2.5 XAML (Extensible Application Markup Language)

XAML (eXtensible Application Markup Language) - це мова розмітки, що використовується у платформі розробки програм Windows, таких як WPF (Windows Presentation Foundation), UWP (Universal Windows Platform) та Xamarin.Forms. XAML дозволяє розробникам описувати інтерфейс користувача і структуру програми за допомогою декларативного синтаксису, подібного до XML.

Основні концепції та можливості XAML полягають у наступному [13]—[15]:

1) декларативний синтаксис. XAML надає декларативний підхід до опису інтерфейсу користувача та структури програми. Розробники описують елементи інтерфейсу, їх взаємозв'язки та властивості за допомогою тегів та атрибутів, що робить код більш зрозумілим та легко підтримуваним;

2) поділ дизайну та логіки. XAML дозволяє розділяти дизайн інтерфейсу користувача від логіки програми. Завдяки цьому можна створювати окремі файли XAML для опису інтерфейсу та окремі файли коду мовою C# або іншими мовами для обробки подій та виконання логіки програми. Це сприяє чистоті коду та спрощує супровід та розробку;

3) ієрархічна структура елементів. XAML дозволяє створювати ієрархічну структуру елементів інтерфейсу. Можна вкладати елементи один в інший, створюючи комплексні макети та компоненти. Це дозволяє легко організовувати інтерфейс програми та керувати розташуванням та взаємодією елементів;

4) стилі та ресурси. XAML підтримує визначення стилів та ресурсів, які можуть бути повторно використані в різних частинах програми. При цьому можна визначити стилі елементів інтерфейсу, задаючи їх зовнішній вигляд, і використовувати їх застосування одноманітного оформлення у різних частинах докладання. Ресурси дозволяють визначити іменовані значення, такі

як кольори, шрифти або зображення, які можна використовувати у різних місцях програми;

5) прив'язка даних. XAML підтримує прив'язку даних, що дозволяє пов'язувати властивості елементів інтерфейсу з даними моделі або інших джерел. Це дозволяє встановлювати прив'язки даних XAML, визначаючи джерело даних, мета прив'язки і спосіб взаємодії. Це забезпечує динамічне оновлення інтерфейсу під час зміни даних;

6) анімація та трансформація. XAML підтримує опис анімації та трансформацій елементів інтерфейсу. Можна визначити анімацію для зміни властивостей елементів, таких як положення, розмір або прозорість. Це дозволяє створювати інтерактивні та привабливі інтерфейси користувача;

7) ресурси та поділ коду та розмітки. У XAML можна визначати різні ресурси, такі як стилі, шрифти, пензлі та інші, які можна використовувати у різних частинах розмітки. Це дозволяє повторно використовувати та централізовано керувати ресурсами, що спрощує підтримку та зміну зовнішнього вигляду програми;

8) вкладеність елементів та контролерів. XAML підтримує вкладеність елементів та контролерів, що дозволяє створювати складні ієрархічні структури інтерфейсу. Можна створювати контейнери, такі як панелі та групи елементів, для організації та маніпулювання з поделементами;

9) модель подія. XAML дозволяє прив'язувати події елементів інтерфейсу до обробників подій у коді програми. Можна визначити обробники подій у коді та зв'язати їх подіями елементів у XAML. Це дозволяє реагувати на дії користувача й виконувати відповідні операції;

10) респонсивний дизайн. XAML надає засоби для створення адаптивного та респонсивного дизайну інтерфейсу. Можна використовувати різні панелі панелі та властивості, такі як прив'язка розмірів і пропорційне масштабування, для створення інтерфейсів, які автоматично адаптуються до різних розмірів екранів і пристроїв;

11) робота з графікою та мультимедіа. XAML дозволяє вбудовувати графічні елементи, включаючи зображення, векторну графіку та 3D-моделі, в інтерфейс програми. Він також підтримує відтворення аудіо та відео файлів, що дозволяє створювати інтерактивні та мультимедійні програми;

12) розширюваність. XAML надає можливість розширення власних елементів користувача і контролів. Можна створювати власні класи, успадковані від базових елементів та контролів, та визначати їх у XAML. Це дозволяє створювати спеціалізовані та перевикористовуються елементи інтерфейсу;

13) XAML є потужним інструментом для створення інтерфейсів програм з використанням Windows Presentation Foundation та інших платформ Windows. Його декларативний синтаксис, поділ дизайну та логіки, прив'язка даних та інші можливості роблять його ефективним інструментом для розробки сучасних та інтуїтивних інтерфейсів користувача.

Приклад проектування інтерфейсу за допомогою XAML зображено на рисунку 2.3.

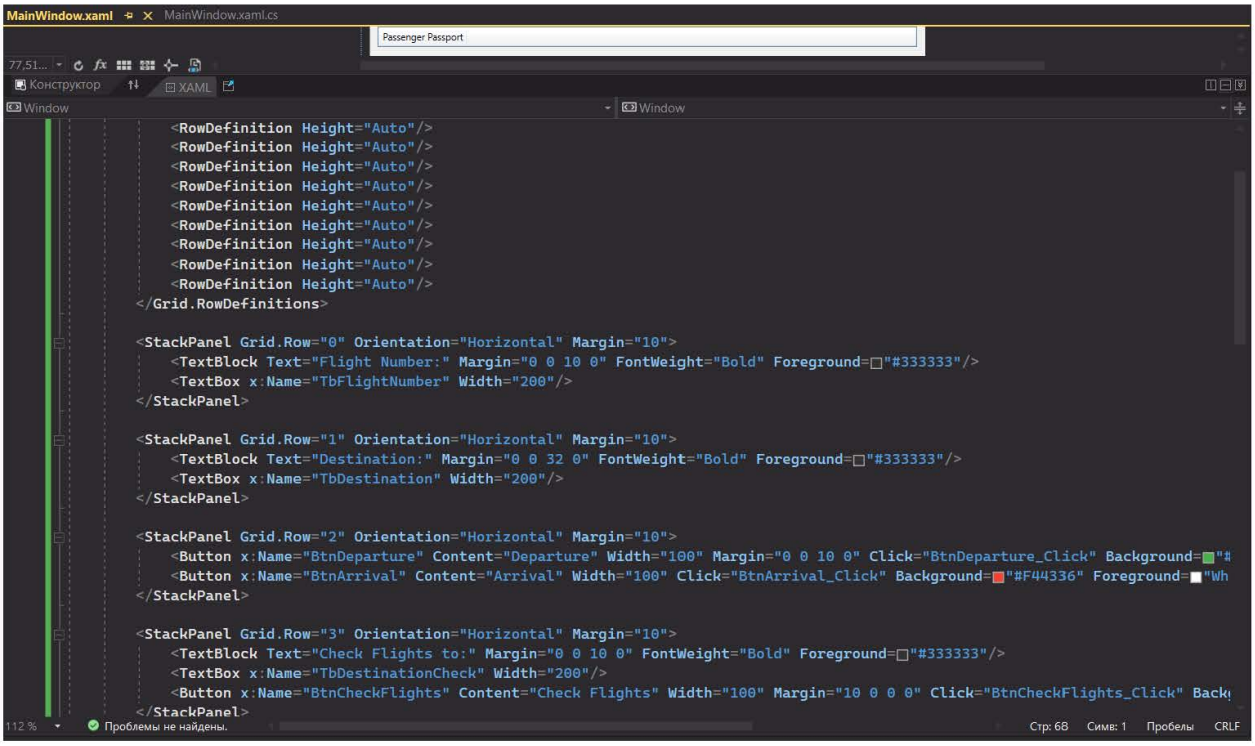

Рисунок 2.3 – Приклад проектування інтерфейсу за допомогою мови XAML

2.6 Елемент керування DataGrid

DataGrid – це елемент керування у Windows Presentation Foundation (WPF), який надає можливість відображати та редагувати дані у вигляді таблиці. Він є потужним інструментом для відображення та управління табличними даними, забезпечуючи широкий набір функцій для сортування, фільтрації, редагування, виділення та маніпулювання даними.

Основні особливості та можливості DataGrid можна виокремити наступним чином [1]—[4], [13]—[15]:

– відображення даних. DataGrid дозволяє відображати дані в табличній формі, де кожен рядок представляє окремий запис, а кожен стовпець представляє окреме поле даних. Це дозволяє зручно представляти та переглядати великі обсяги даних;

– редагування даних. DataGrid надає можливість редагування даних безпосередньо у осередках таблиці. Можна налаштувати режими редагування для різних стовпців, включаючи лише читання, одинарне редагування або множинне редагування;

– сортування та фільтрація. DataGrid дозволяє сортувати дані по стовпцям у зростаючому чи спадному порядку. Він також підтримує фільтрацію даних, що дозволяє користувачеві швидко знаходити потрібні записи великому обсязі даних;

– виділення даних. DataGrid дозволяє виділяти один або кілька рядків даних для виконання різних операцій, таких як видалення, копіювання або оновлення. При цьому можна налаштувати типи виділення, такі як одиночний вибір, множинний вибір або діапазон вибору;

– автоматичне створення стовпців. DataGrid може автоматично створювати стовпці з урахуванням структури даних, що дозволяє швидко відобразити дані без явного визначення кожного стовпця. Однак він також підтримує ручне визначення стовпців за допомогою XAML або коду;

– шаблони користувача. DataGrid дозволяє налаштовувати відображення осередків, стовпців і рядків за допомогою шаблонів користувача. Можна визначити власні шаблони для осередків, включаючи елементи управління чи стилізацію, що дозволяє створювати кастомні та індивідуальні уявлення даних;

– угруповання та зведені таблиці. DataGrid підтримує групування даних по одному або кільком стовпцям. Це дозволяє створювати ієрархічні уявлення даних. Крім того, він також підтримує агрегацію даних та створення зведених таблиць для аналізу та підсумовування даних;

– підтримка подій та команд. DataGrid генерує різні події, пов'язані зі зміною даних або користувальницькою взаємодією. Можна підписуватись на ці події для виконання необхідних операцій. Він також підтримує команди, які можуть бути прив'язані до дій користувачів;

– стилізація та зовнішній вигляд. DataGrid надає безліч налаштувань для стилізації та налаштування зовнішнього вигляду. При цьому можна змінювати кольори, шрифти, межі та інші аспекти оформлення таблиці. Вони також можуть настроювати зовнішній вигляд виділених рядків, заголовків стовпців та інших елементів;

– гнучкість у роботі з даними. DataGrid підтримує прив'язку даних, що дозволяє автоматично оновлювати таблицю при зміні даних та зберігати зміни назад у джерело даних. Він також підтримує події редагування, які можуть бути використані для валідації даних або виконання додаткових дій у разі зміни значень осередків;

– підтримка різних типів даних. DataGrid може працювати з різними типами даних, включаючи числа, рядки, дати, зображення та інші. Він надає можливість відображати та редагувати значення цих типів у відповідних осередках таблиці;

– віртуалізація. DataGrid підтримує віртуалізацію даних, що дозволяє ефективно працювати з великими обсягами даних. Він завантажує лише видимі рядки даних, що покращує продуктивність та заощаджує пам'ять;

– підтримка сортування та фільтрації. DataGrid надає вбудовані можливості для сортування даних за одним або декількома стовпцями. Він також дозволяє задавати фільтри для відображення тільки потрібних записів;

– посторінкова навігація. DataGrid підтримує посторінкову навігацію за даними. Можна налаштувати розмір сторінки та додати елементи керування для переходу до попередньої та наступної сторінок;

– інтеграція з іншими елементами керування DataGrid може бути інтегрований з іншими елементами керування WPF, такими як кнопки, текстові поля або фільтри. Це дозволяє створювати складні інтерфейси користувача, що включають взаємодію з даними в таблиці;

– DataGrid є одним з ключових елементів управління WPF, що забезпечує зручне відображення та управління табличними даними. Він надає багатий набір функцій та можливостей, що робить його незамінним інструментом для роботи з даними у програмах на платформі Windows.

Приклад використання компоненту DataGrid зображено на рисунку 2.4.

```
<DataGrid x:Name="PassengersDataGrid" Grid.Row="8" Margin="10" AutoGenerateColumns="False">
    <DataGrid.Columns>
        <DataGridTextColumn Header="Passenger Passport" Binding="{Binding Passport}"/>
    </DataGrid.Columns>
</DataGrid>
```

Рисунок 2.4 – Приклад використання компоненту DataGrid

2.7 LINQ (Language-Integrated Query)

Language-Integrated Query (LINQ) - це набір інтегрованих у мову C# і VB.NET засобів запиту, які дозволяють розробникам виконувати запити до різних джерел даних, таких як колекції об'єктів, бази даних, XML-документи та інші, з використанням синтаксису, близького до структурованої мови запитів (SQL).

LINQ надає розробникам зручніший і виразніший спосіб роботи з даними, дозволяючи писати запити безпосередньо в коді програми, а не використовувати традиційні методи та мови запитів.

Основні концепції та можливості LINQ [16]—[17]:

1) об'єкти запиту (Query Objects). LINQ дозволяє створювати об'єкти запиту, які представляють запит до джерела даних. Ці об'єкти можуть містити умови, фільтри, сортування та інші операції над даними;

2) мова запитів (Query Language). LINQ надає мову запитів, яка є частиною C# або VB.NET. Ця мова містить оператори та ключові слова, що дозволяють писати виразні та лаконічні запити до даних;

3) інтеграція у мову програмування. LINQ інтегрований у мову C# і VB.NET, що дозволяє легким чином писати запити у коді програми, використовуючи знайомий синтаксис мови. Це полегшує читання, розуміння та підтримку коду;

4) підтримка різних джерел даних. LINQ підтримує різноманітні джерела даних, включаючи колекції об'єктів, бази даних, XML-документи, веб-сервіси та інші. При цьому можна писати єдині запити до різних типів даних, що полегшує роботу з даними з різних джерел;

5) інтелектуальна перевірка типів (Type Checking). LINQ використовує інтелектуальну перевірку типів під час компіляції, що дозволяє виявляти помилки та невідповідності у запитах до даних на ранніх стадіях розробки. Це сприяє усуненню помилок та підвищенню надійності коду;

6) підтримка анонімних типів (Anonymous Types). LINQ дозволяє створювати анонімні типи даних на льоту, що дозволяє повернути із запиту набір даних з певними полями та значенням, не потрібно явного визначення класу;

7) розширюваність (Extensibility). LINQ має високий ступінь розширюваності, що дозволяє створювати провайдери даних користувача й розширювати можливості LINQ під свої специфічні потреби;

8) оператори запитів (Query Operators). LINQ надає безліч операторів запитів, які дозволяють виконати різні операції над даними, такі як фільтрація, сортування, групування, об'єднання, агрегація та інші. Оператори запитів дозволяють писати більш складні та потужні запити, обробляючи дані у зручній формі;

9) інтеграція з SQL. LINQ дозволяє створювати запити, які можуть бути виконані безпосередньо на сервері бази даних із використанням мови SQL. Це досягається за допомогою LINQ to SQL або Entity Framework, які надають механізми перетворення LINQ-запитів на SQL-запити;

10) підтримка паралельного виконання (Parallel Execution). LINQ надає можливість паралельного виконання запитів, що дозволяє ефективно використовувати багатопоточність для прискорення обробки даних. Це особливо корисно при роботі з великими обсягами даних або під час виконання обчислювально-інтенсивних операцій;

11) інтеграція з розширеннями NET. LINQ інтегрується з іншими розширеннями .NET, такими як PLINQ (Parallel LINQ), що надає розширені можливості паралельної обробки даних, та Entity Framework, який надає ORM (Object-Relational Mapping) для роботи з базами даних;

12) підтримка різних платформ. LINQ підтримується не тільки на .NET Framework, але і на інших платформах, таких як .NET Core і Xamarin. Це дозволяє використовувати LINQ для роботи з даними в різних типах програм, включаючи веб-додатки, мобільні та десктопні програми;

13) LINQ to XML. LINQ включає спеціальний провайдер – LINQ to XML, який дозволяє працювати з XML-документами з використанням LINQ-запитів. Це робить обробку та маніпулювання XML-даними більш простими та ефективними;

14) інтеграція з розширеннями сторонніх виробників. LINQ має підтримку від сторонніх виробників, які надають додаткові можливості та провайдери даних для роботи з різними джерелами даних, такими як бази даних NoSQL або веб-сервіси;

15) LINQ є сучасним та гнучким інструментом для роботи з даними у додатках на платформі. Він забезпечує зручний та виразний синтаксис для написання запитів, інтегрується з різними джерелами даних та розширеннями .NET, та дозволяє ефективно обробляти дані у різних сценаріях розробки додатків.

Схема використання LINQ зображена на рисунку 2.5.

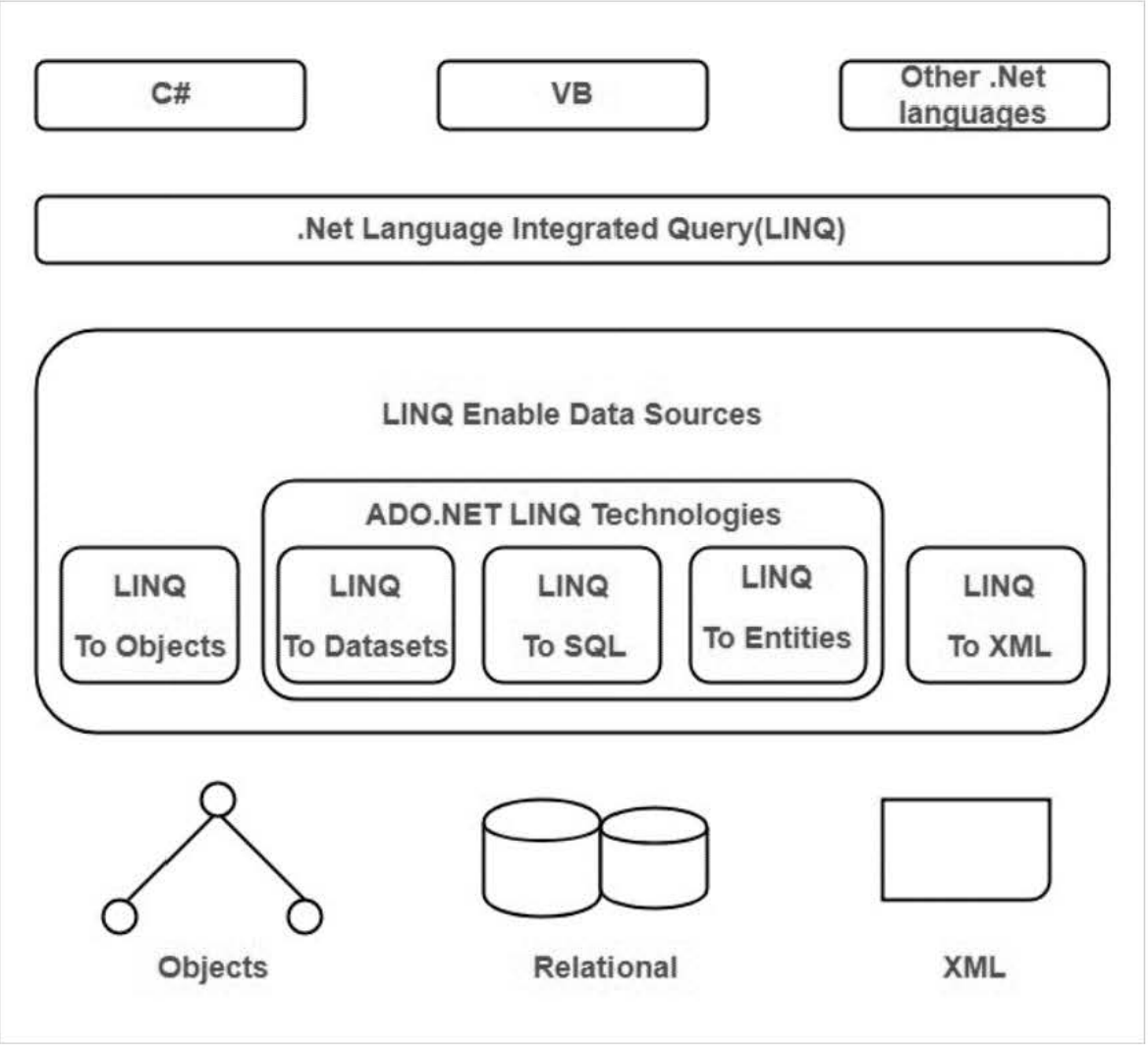

Рисунок 2.5 – Схема використання LINQ

Створити класи LINQ to SQL можна за допомогою додавання нового елемента (рис. 2.6).

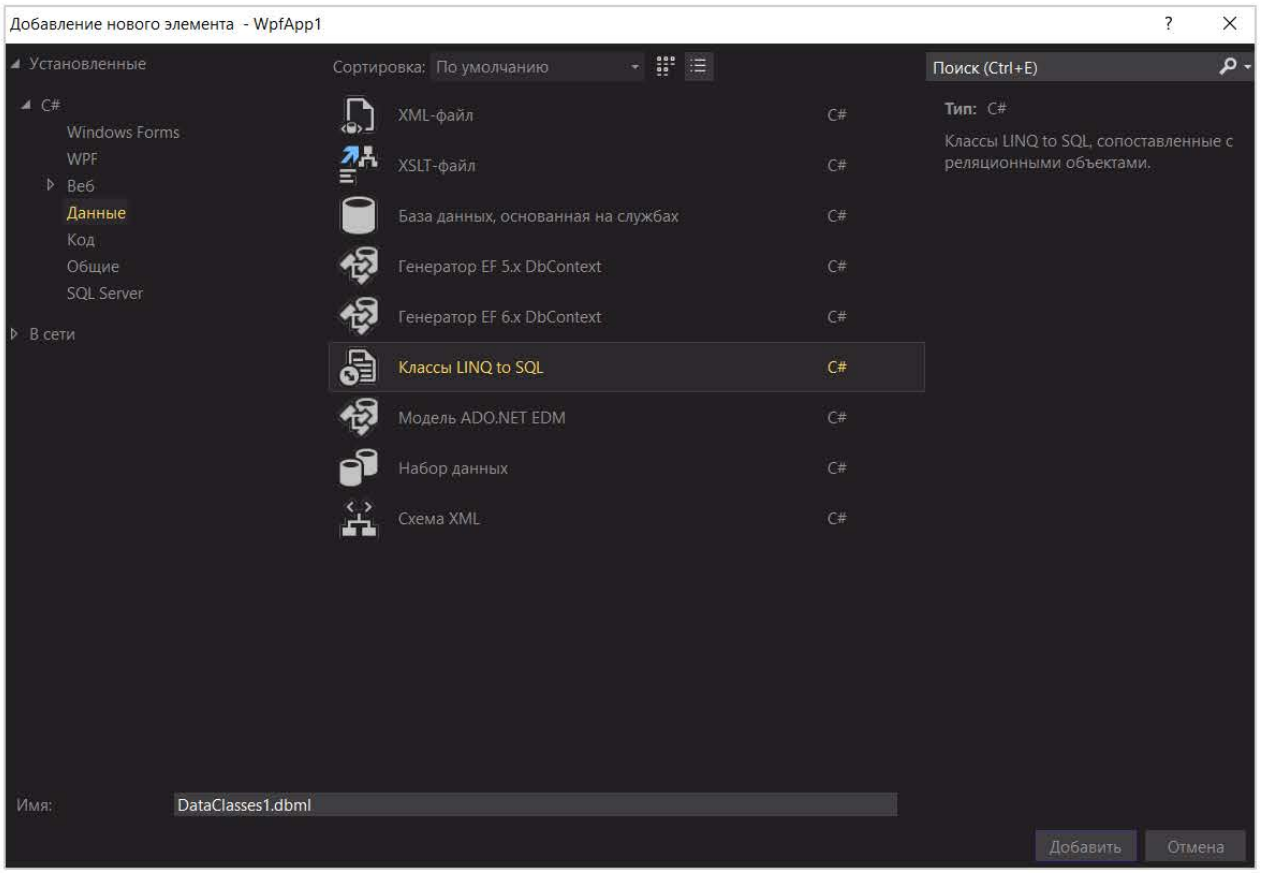

Рисунок 2.6 – Додавання нового елемента класів LINQ to SQL

2.8 Microsoft Visual Studio IDE

Microsoft Visual Studio – це інтегроване середовище розробки (IDE) від компанії Microsoft, призначене для розробки різних типів програм, включаючи десктопні програми, веб-програми, мобільні програми, ігри та інше. Visual Studio забезпечує розробникам всебічну підтримку, інструменти та ресурси для спрощення та прискорення процесу створення програмного забезпечення.

Основні можливості та компоненти Microsoft Visual Studio можна представити наступним чином:

– інтерфейс та оточення розробки. Visual Studio пропонує зручний та інтуїтивно зрозумілий інтерфейс, який полегшує навігацію та роботу з проектами. Він включає вікна та панелі інструментів для керування файлами, редагування коду, налагодження, версіонування та інших завдань розробки;

– редактор коду. Visual Studio має потужний редактор коду з підсвічуванням синтаксису, автодоповненням, функціями форматування, переходом до визначення, швидкими виправленнями помилок та іншими функціями, які допомагають розробникам писати якісний та чистий код;

– підтримка мов програмування. Visual Studio підтримує широкий спектр мов програмування, включаючи C#, Visual Basic, C++, F#, JavaScript, Python, TypeScript та інші. Це дозволяє обрати найбільш підходящу мову для своїх проектів та забезпечує гнучкість у розробці;

– налагодження та профілювання. Visual Studio надає потужні інструменти для налагодження додатків, що дозволяють шукати та виправляти помилки, аналізувати змінні, стежити за виконанням коду, використовувати точки зупинки та багато іншого. Крім того, Visual Studio пропонує інструменти профілювання для аналізу продуктивності програм та оптимізації їх роботи;

– управління проектами та складанням. Visual Studio забезпечує зручне управління проектами, включаючи створення, налаштування та складання додатків. Він підтримує різні типи проектів, такі як Windows Forms, WPF, ASP.NET, Xamarin, Unity тощо, та надає інструменти для керування залежностями, ресурсами, конфігурацією та іншими аспектами проекту;

– інтеграція із системами контролю версій. Visual Studio інтегрується з популярними системами контролю версій, такими як Git та Team Foundation Version Control (TFVC), що дозволяє ефективно керувати та відстежувати зміни у вихідному коді проекту;

– розширюваність та екосистема. Visual Studio пропонує велику екосистему розширень, які дозволяють розширити можливості середовища розробки. Це включає плагіни, шаблони проектів, бібліотеки компонентів та інші ресурси, які допомагають покращити процес розробки та підвищити продуктивність;

– тестування. Visual Studio надає інструменти для створення та виконання модульних тестів, інтеграційних тестів та інших видів тестування.

Він дозволяє автоматизувати тестування та забезпечувати якість своїх додатків;

– інтеграція з хмарними сервісами. Visual Studio інтегрується з різними хмарними сервісами, такими як Azure, для розгортання, керування та моніторингу програм у хмарному середовищі. Це дозволяє створювати та розробляти хмарні рішення безпосередньо у середовищі розробки Visual Studio.

Microsoft Visual Studio є одним з найбільш популярних інструментів розробки програмного забезпечення, що широко використовуються, надаючи потужні можливості, високу продуктивність та зручність використання для розробників на платформі .NET та інших технологіях.

Інтерфейс Microsoft Visual Studio IDE зображено на рисунку 2.7.

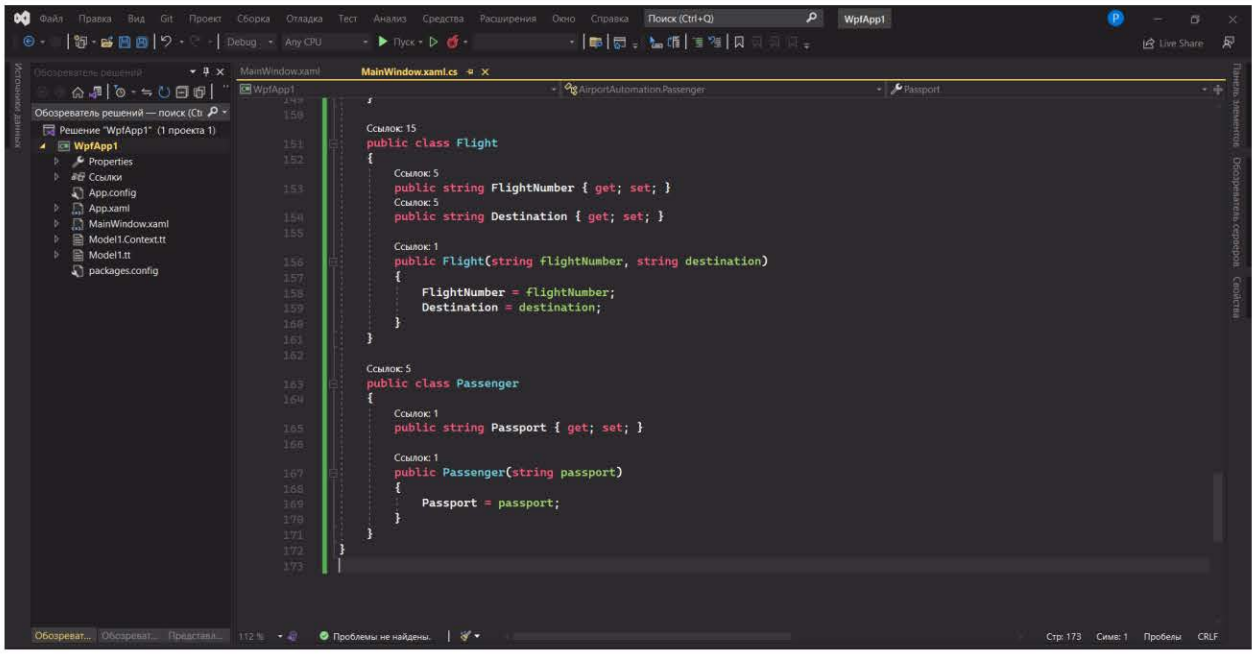

Рисунок 2.7 – Інтерфейс Microsoft Visual Studio IDE

2.9 Висновок до другого розділу

У другому розділі було розглянуто різні інструменти розробки програмного забезпечення автоматизації роботи аеропорту. Було проведено аналіз та вибір програмних засобів, які підходять для розробки WPF додатку.

Спочатку були розглянуті основні програмні засоби, що використовуються для розробки веб-додатків, з фокусом на автоматизацію роботи аеропорту. Потім було представлено опис Windows Presentation Foundation (WPF) – технології розробки інтерфейсів, що дозволяє створювати сучасні та інтерактивні програми з використанням графічних елементів.

Далі була розглянута мова програмування C# (C-Sharp) – потужна та гнучка мова програмування, яка є основною мовою розробки додатків на платформі .NET. Вона забезпечує широкі можливості для створення високопродуктивного та надійного коду.

Entity Framework (EF) був представлений як інструмент для роботи з даними в додатках, що надає зручний та виразний спосіб взаємодії з базами даних. EF спрощує процес доступу до даних та надає функції для роботи з об’єктами бази даних.

XAML (Extensible Application Markup Language) був представлений як мова розмітки, що використовується для створення інтерфейсів користувача в WPF додатках. XAML дозволяє описувати структуру та зовнішній вигляд елементів інтерфейсу у декларативній формі.

Далі було розглянуто елемент управління DataGrid, що надається WPF, який полегшує відображення та редагування табличних даних. DataGrid надає гнучку настройку, дозволяє прив’язувати дані та надає функціонал для сортування, фільтрації та групування даних.

LINQ (Language-Integrated Query) був представлений як інтегрована мова запитів, що дозволяє виконувати запити до різних джерел даних, включаючи бази даних, колекції об'єктів та інші. LINQ забезпечує виразний синтаксис для фільтрації, сортування та маніпулювання даними.

Нарешті, була представлена Microsoft Visual Studio IDE – інтегроване середовище розробки, яке надає всі необхідні інструменти та ресурси для створення, налагодження та управління проектами. Visual Studio забезпечує зручний інтерфейс, підтримку різних мов програмування, інтеграцію із системами контролю версій та розширюваність за допомогою плагінів та розширень.

Таким чином, другий розділ містить докладний огляд та опис обраних інструментів розробки, які є важливими компонентами для створення програмного забезпечення для автоматизації роботи аеропорту.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ АЕРОПОРТУ

3.1 Побудова проекту

Проект є простим додатком для автоматизації аеропорту. Він розроблений на платформі WPF (Windows Presentation Foundation) з використанням мови програмування C# і XAML.

3.1.1 Структура проекту

Структура проекту складається з:

1) файлу MainWindow.xaml. Він визначає графічний інтерфейс програми за допомогою мови розмітки XAML. Всередині елемента <Grid> розташовані кілька елементів <StackPanel>, в яких містяться різні елементи керування, такі як текстові блоки, текстові поля, кнопки та таблиця даних;

2) файлу MainWindow.xaml.cs. Містить код класу MainWindow, який є код-бехайндом для головного вікна програми. Тут визначено обробників подій для кнопок та інших елементів керування, а також логіку їхньої роботи. У цьому файлі також визначено два допоміжні класи Flight та Passenger, які подають дані про рейси та пасажирів відповідно.

3.1.2 Архітектура проекту

Загальна архітектура проекту може бути представлена наступним чином:

– Presentation Layer. Визначений у файлі MainWindow.xaml і відповідає за відображення графічного інтерфейсу та обробку дій користувача;

– Business Layer. Визначений у файлі MainWindow.xaml.cs та містить бізнес-логіку програми. Тут виконуються різні операції, такі як додавання пасажирів, реєстрація вильотів та прильотів рейсів, пошук рейсів тощо;

– Data Model. Представлений класами Flight та Passenger, які представляють моделі даних для рейсів та пасажирів.

Вікно програми складається з кількох рядків <Grid.RowDefinitions>, що визначають розміри рядків у сітці, та кількох елементів <StackPanel>, які розміщуються в кожному рядку для організації розміщення елементів керування.

В кожному обробнику подій кнопок визначено дії, які виконуються під час натискання кнопки. Наприклад, при натисканні кнопки «Departure» викликається метод BtnDeparture_Click, який отримує номер рейсу та пункт призначення з текстових полів та виконує відповідні дії, наприклад, оновлення бази даних та відображення повідомлення.

Також у коді визначено методи для роботи з пасажирами та рейсами, такі як додавання пасажирів, пошук рейсів, отримання доступних рейсів тощо.

У результаті, даний проект є програмним додатком для управління авіарейсами та пасажирами, що дозволяє додавати та відстежувати інформацію про рейси, реєструвати прильоти та вильоти, та виконувати інші операції пов'язані польотами.

3.1.3 Проектування та опис класів

Опис класів для проекту, що розробляється, представлений по тексту нижче.

Клас Flight:

1) властивості:

– FlightNumber (тип: string): номер рейсу;

– Destination (тип: string): пункт призначення.

2) Конструктор:

– Flight(string flightNumber, string destination): створює новий об’єкт рейсу із зазначеним номером та пунктом призначення;

Клас Passenger:

1) властивості:

– Passport (тип: string): номер паспорта пасажира.

2) конструктор:

– Passenger(string passport): створює нового пасажира із зазначеним номером паспорта.

Клас MainWindowViewModel:

1) властивості:

– Flights (тип: ObservableCollection<Flight>): колекція рейсів для відображення в GUI;

– Passengers (тип: ObservableCollection<Passenger>): колекція пасажирів для відображення в GUI.

2) методи:

– AddPassenger(string passport): додає нового пасажира із зазначеним номером паспорта до бази даних та оновлює колекцію пасажирів;

– GetAvailableFlights(string destination): повертає список доступних рейсів для зазначеного пункту призначення;

– AddFlight(string flightNumber, string destination): додає новий рейс із зазначеним номером та пунктом призначення до бази даних та оновлює колекцію рейсів;

– GetAllFlights(): повертає список усіх рейсів з бази даних;

– SearchFlightByNumber(string flightNumber): виконує пошук рейсу за вказаним номером у базі даних та повертає знайдений рейс або null, якщо рейс не знайдений.

Клас DatabaseContext (успадковується від DbContext):

1) властивості:

– Flights (тип: DbSet<Flight>): таблиця рейсів у базі даних;

– Passengers (тип: DbSet<Passenger>): таблиця пасажирів у базі даних.

У класі MainWindow може бути додане підключення та використання MainWindowViewModel для зв'язку GUI з бізнес-логікою та моделями даних. Також потрібне створення екземпляра DatabaseContext для взаємодії з базою даних.

3.2 Опис роботи програмного забезпечення для автоматизації роботи аеропорту

Опис роботи проекту, що розробляється, може бути представлений покроково наступним чином:

1) при запуску програми відкривається головне вікно MainWindow, яке містить інтерфейс користувача;

2) у вікні є кілька розділів, які призначені для виконання різних операцій;

3) у розділі «Add Passenger» користувач вводить номер паспорта пасажира у текстове поле та натискає кнопку «Add Passenger». Програма створює об'єкт Passenger із зазначеним номером паспорта та додає його до списку passengers. Потім відбувається оновлення таблиці із пасажирами, що відображає актуальні дані;

4) у розділі «Departure» користувач вводить номер рейсу та пункт призначення до відповідних текстових полів, потім натискає кнопку «Departure». Програма виконує необхідні дії для відправлення рейсу, наприклад, оновлює базу даних, відображає інформаційне повідомлення про виконану операцію;

5) у розділі Arrival користувач вводить номер рейсу в текстове поле і натискає кнопку Arrival. Програма виконує необхідні дії для прибуття рейсу, такі як оновлення бази даних, виведення інформаційного повідомлення;

6) у розділі «Check Flights» користувач вводить пункт призначення у текстове поле та натискає кнопку «Check Flights». Додаток перевіряє доступні рейси для зазначеного пункту призначення та виводить повідомлення з інформацією про доступні рейси;

7) у розділі «Add Flight» користувач вводить номер рейсу та пункт призначення у відповідні текстові поля та натискає кнопку «Add Flight». Програма створює новий об'єкт Flight із зазначеними даними та додає його до списку flights. Потім виводиться інформаційне повідомлення щодо додавання рейсу;

8) у розділі «View All Flights» користувач натискає кнопку «View All Flights». Програма виводить повідомлення зі списком усіх рейсів, доступних у базі даних;

9) у розділі «Search Flight» користувач вводить номер рейсу в текстове поле та натискає кнопку «Search Flight». Програма шукає рейс за вказаним номером та виводить інформацію про знайдений рейс або повідомлення про те, що рейс не знайдено;

10) додаток продовжує роботу, дозволяючи користувачеві виконувати операції додавання пасажирів, управління рейсами та отримання інформації про рейси та пасажирів.

3.3 Проектування користувацького інтерфейсу

Докладний опис процесу проектування інтерфейсу користувача на прикладі коду представлений по тексту нижче.

Визначення вікна та його параметрів:

– створюється об'єкт Window з класом AirportAutomation.MainWindow;

– задаються атрибути Title для заголовка вікна та Height та Width для його розмірів.

Створення контейнера Grid:

– створюється об’єкт Grid усередині вікна для розміщення елементів керування;

– встановлюється атрибут Background для завдання фону контейнера.

Визначення рядків Grid.RowDefinitions:

– визначаються рядки контейнера Grid, вказуючи їх висоту;

– використовується RowDefinition з атрибутом Height для кожного рядка.

Розміщення елементів керування:

– використовуючи StackPanel, елементи управління розміщуються у кожному рядку контейнера Grid;

– кожен StackPanel містить елементи управління, такі як TextBlock, TextBox та Button, а також їх атрибути та події;

– Grid.Row вказує, у якому рядку розміщується StackPanel.

Додавання DataGrid:

– створюється об’єкт DataGrid, призначений для відображення даних у вигляді таблиці;

– встановлюються атрибути x.Name для ідентифікації об’єкта, Grid.Row для вказівки рядка розміщення та AutoGenerateColumns=«False» для відключення автоматичної генерації стовпців;

– визначається DataGridTextColumn для відображення даних у стовпці таблиці.

На рисунку 3.1 зображено кінцевий вигляд інтерфейсу користувача.

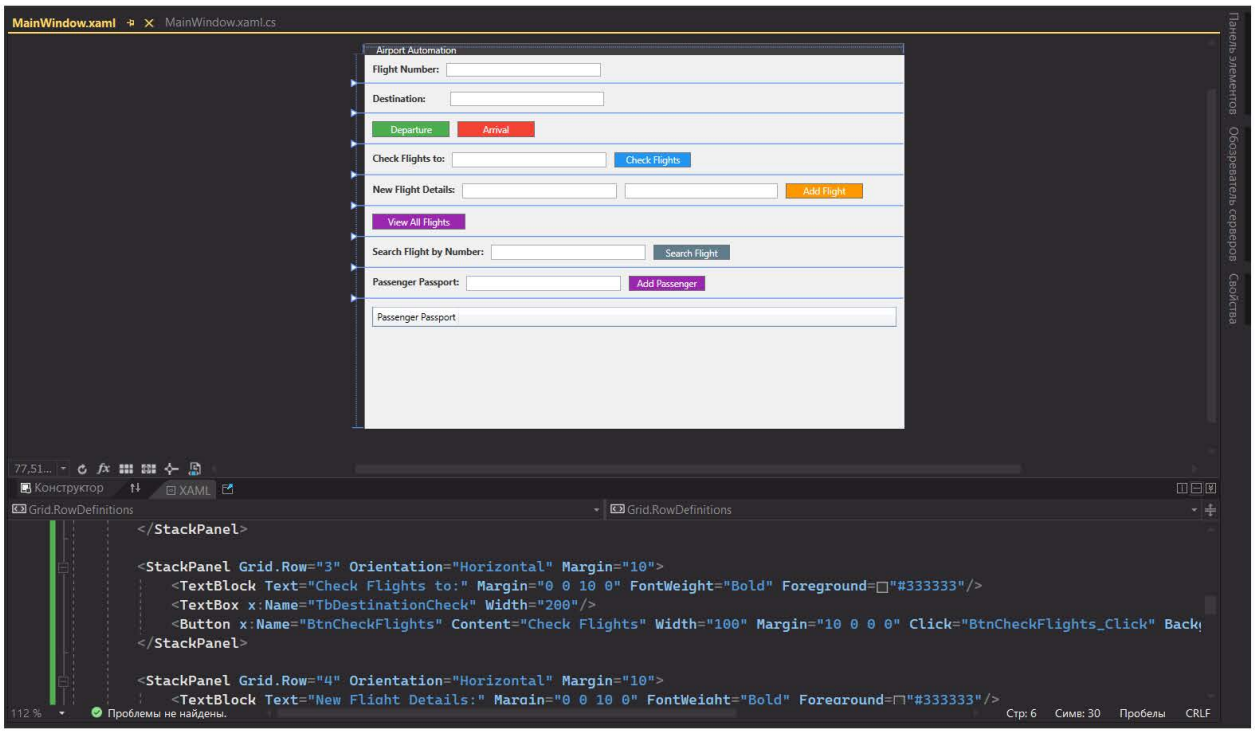

Рисунок 3.1 – Інтерфейс користувача

3.4 Проектування бази даних

Процес проектування бази даних у цьому проекті включає наступні кроки:

1) визначення сутностей. На початку процесу визначаються сутності, які будуть представлені у базі даних. У цьому проекті визначено дві сутності. Flight (рейс) та Passenger (пасажир). Кожна сутність представлена у вигляді класу з відповідними властивостями, атрибутом [Key] вказується первинний ключ;

2) створення контексту бази даних. Потім створюється клас контексту бази даних, який успадковується від DbContext (з простору імен System.Data.Entity). У даному коді клас AirportContext містить дві властивості DbSet<Flight> та DbSet<Passenger>, які представляють відповідні таблиці в базі даних;

3) ініціалізація контексту бази даних. У конструкторі класу MainWindow ініціалізується об’єкт AirportContext, який буде використовуватися для взаємодії з базою даних;

4) використання контексту бази даних. У методі BtnAddPassenger_Click відбувається створення нового об’єкта Passenger на основі введених даних, додавання його в контекст бази даних (context.Passengers.Add(passenger)) та збереження змін у базі даних (context.SaveChanges()).

Процес проектування бази даних вимагає також врахування інших аспектів, таких як визначення зв’язків між таблицями, налаштування індексів, оптимізація запитів тощо. Однак, у цьому прикладі вказані основні кроки, пов’язані зі створенням класів сутностей, контексту бази даних та використанням контексту для додавання даних до бази даних.

На рисунках 3.2-3.3 зображено процеси додавання до проекту бази даних Entity Framework.

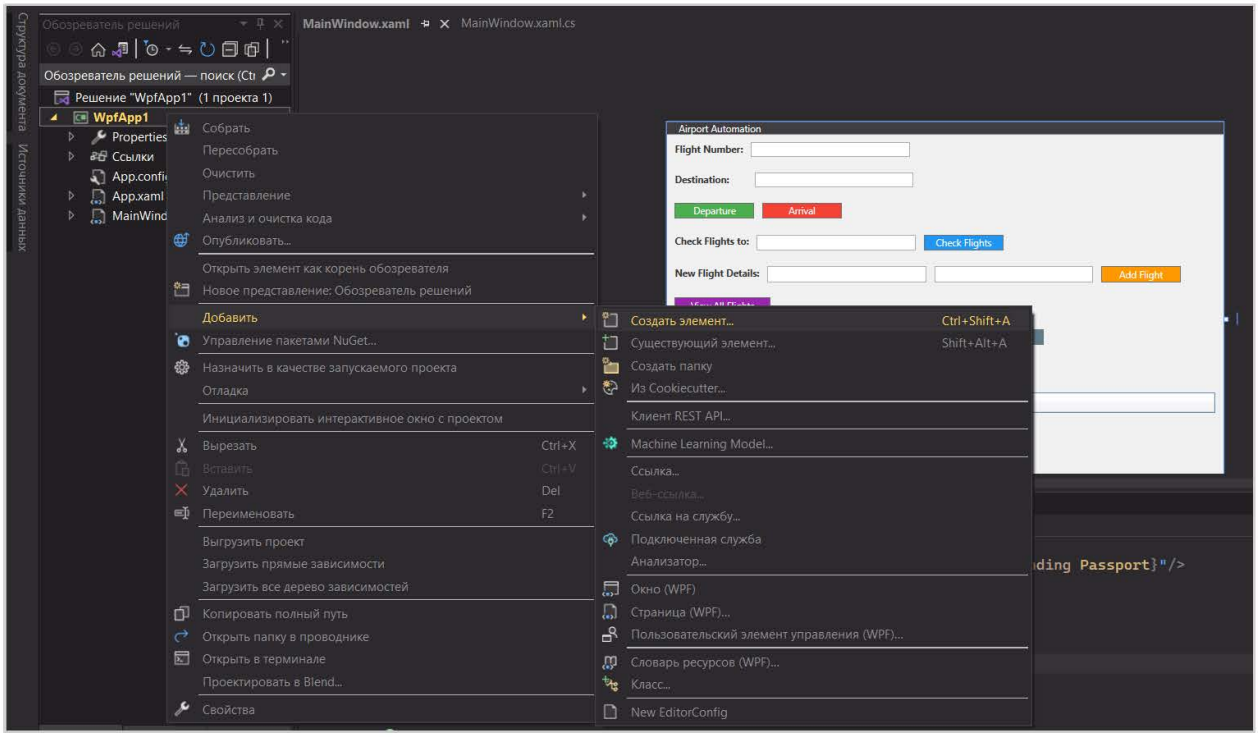

Рисунок 3.2 – Створення нового елемента

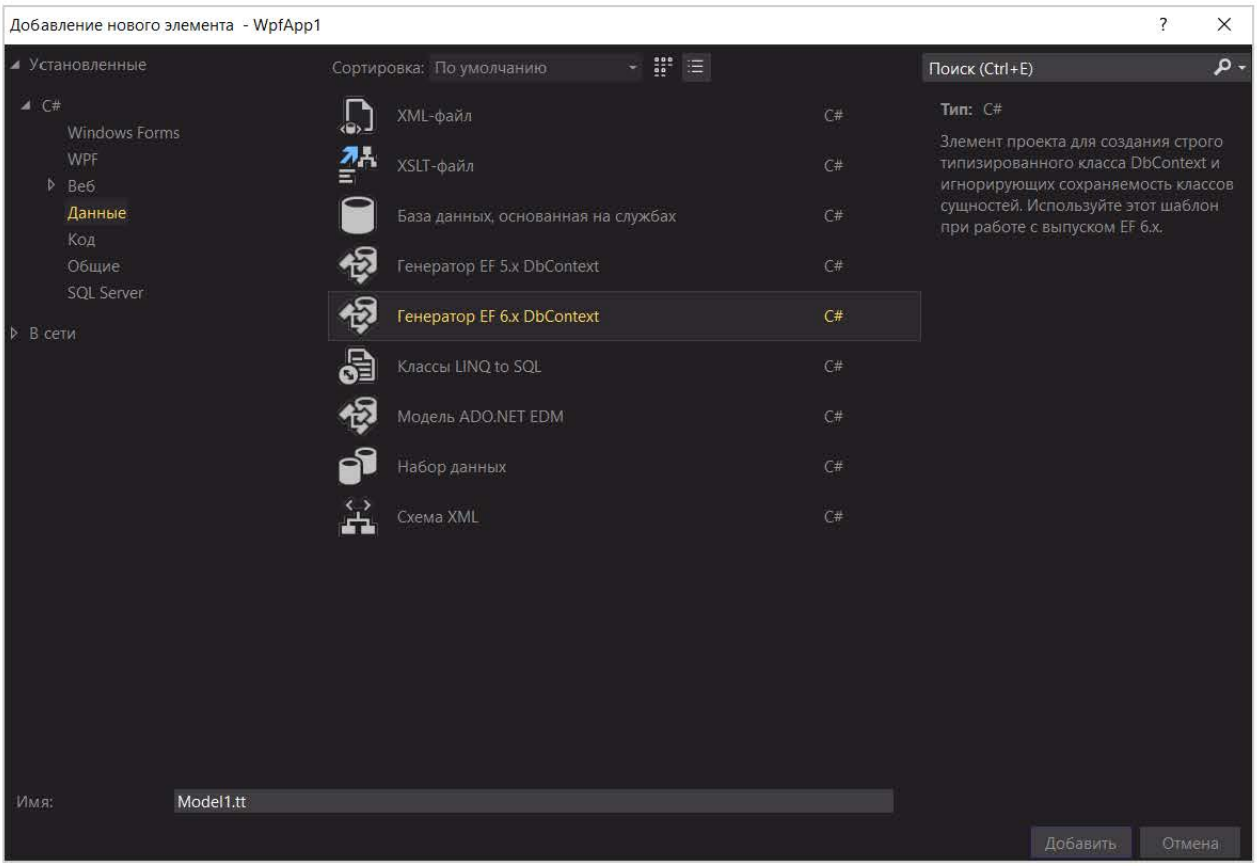

Рисунок 3.3 – Додавання генератора EF 6.x DbContext

3.5 Тестування програмного забезпечення для автоматизації роботи аеропорту

На рисунку 3.4 зображене головне вікно програмного застосунку для автоматизації роботи аеропорту.

На рисунку 3.5 зображено процес відльоту літака.

На рисунку 3.6 зображено процес приземлення літака.

На рисунку 3.7 можна переглянути наявні рейси, вказавши відповідний номер літака.

На рисунку 3.8 зображено процес додавання нового літака.

На рисунку 3.9 зображено функціонал перегляду усіх існуючих літаків у конкретний момент часу.

На рисунку 3.10 зображено функціонал перегляду інформації про конкретний літак за його номером.

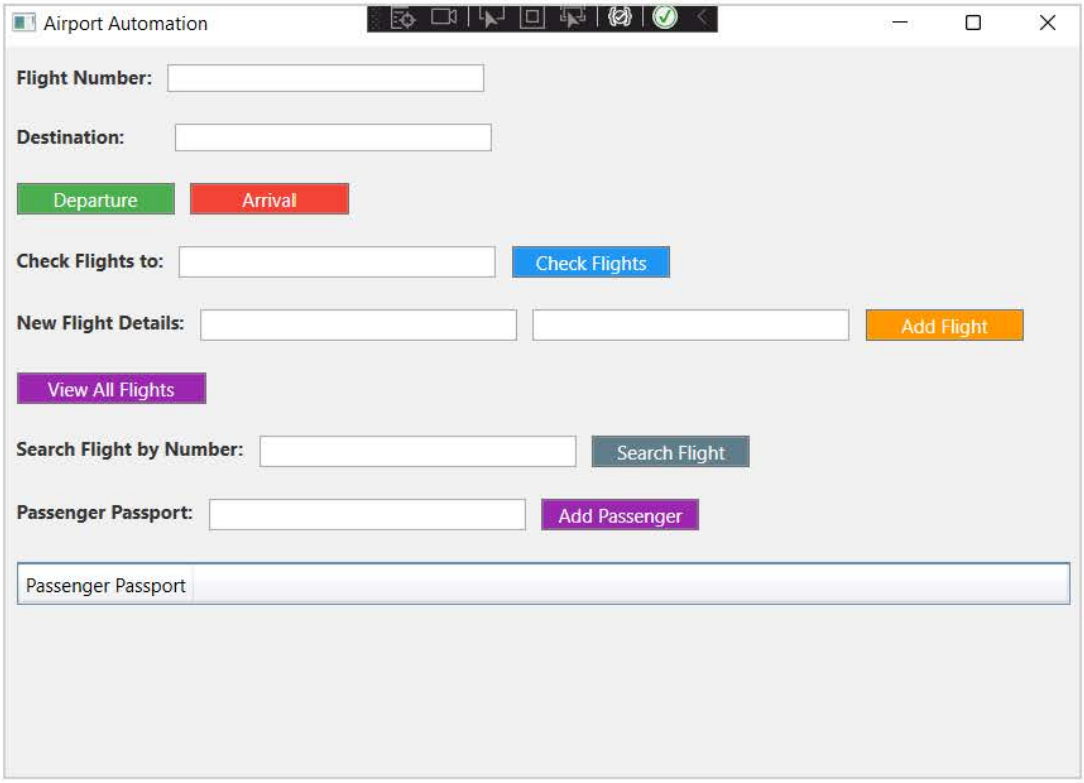

Рисунок 3.4 – Головне вікно програмного застосунку для автоматизації роботи аеропорту

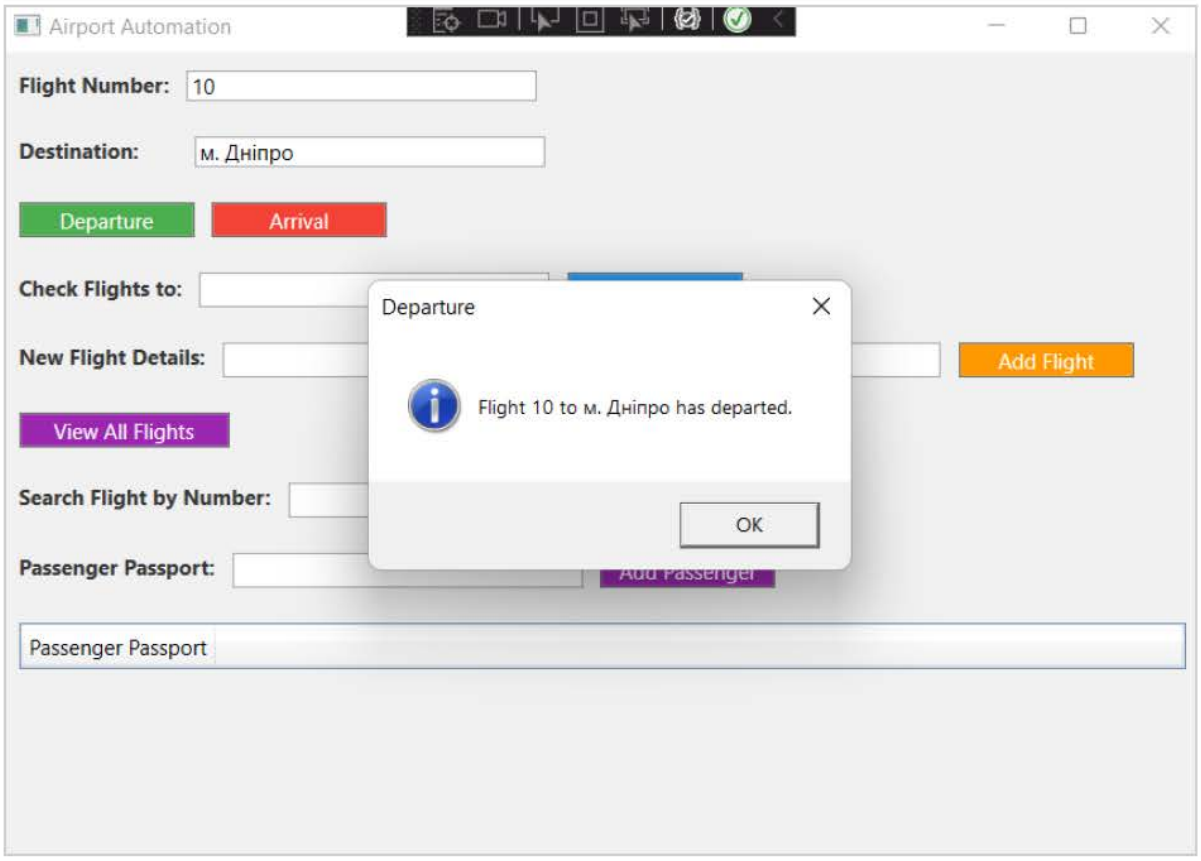

Рисунок 3.5 – Відліт літака

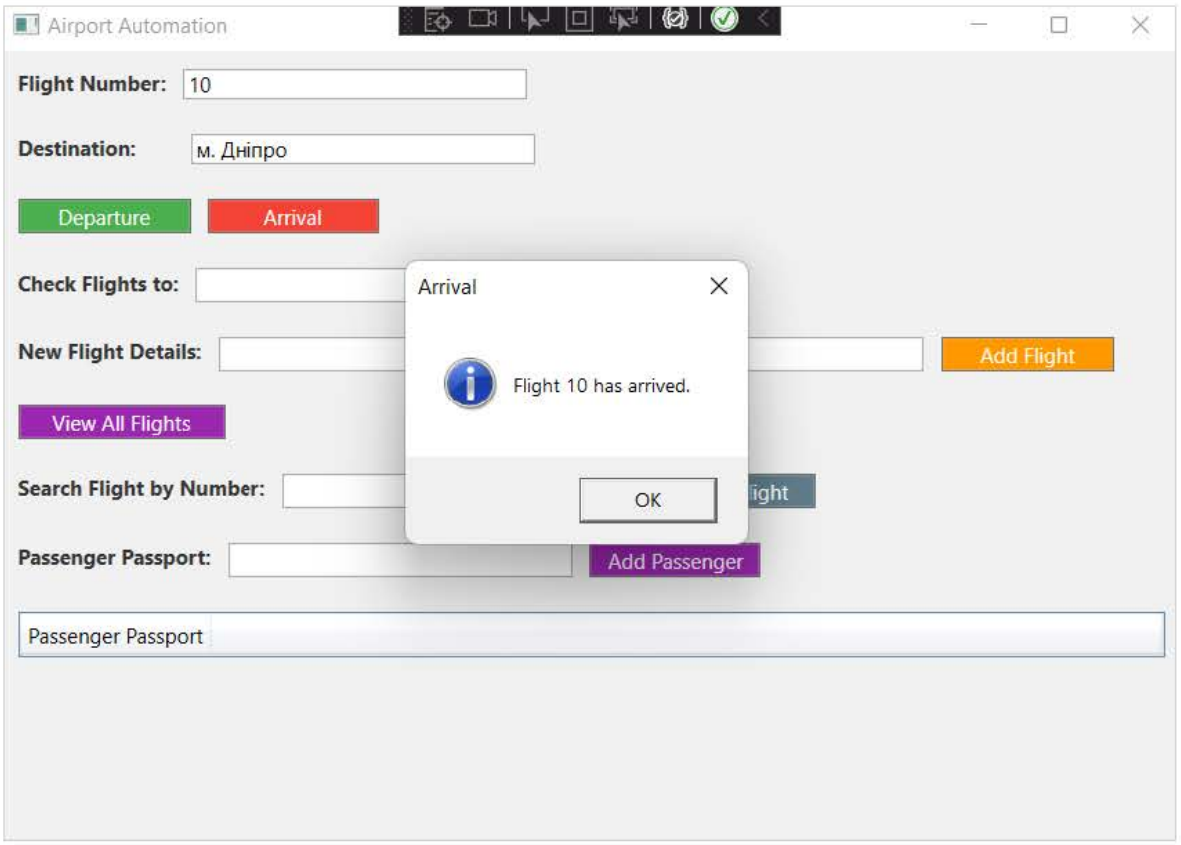

Рисунок 3.6 – Приліт літака

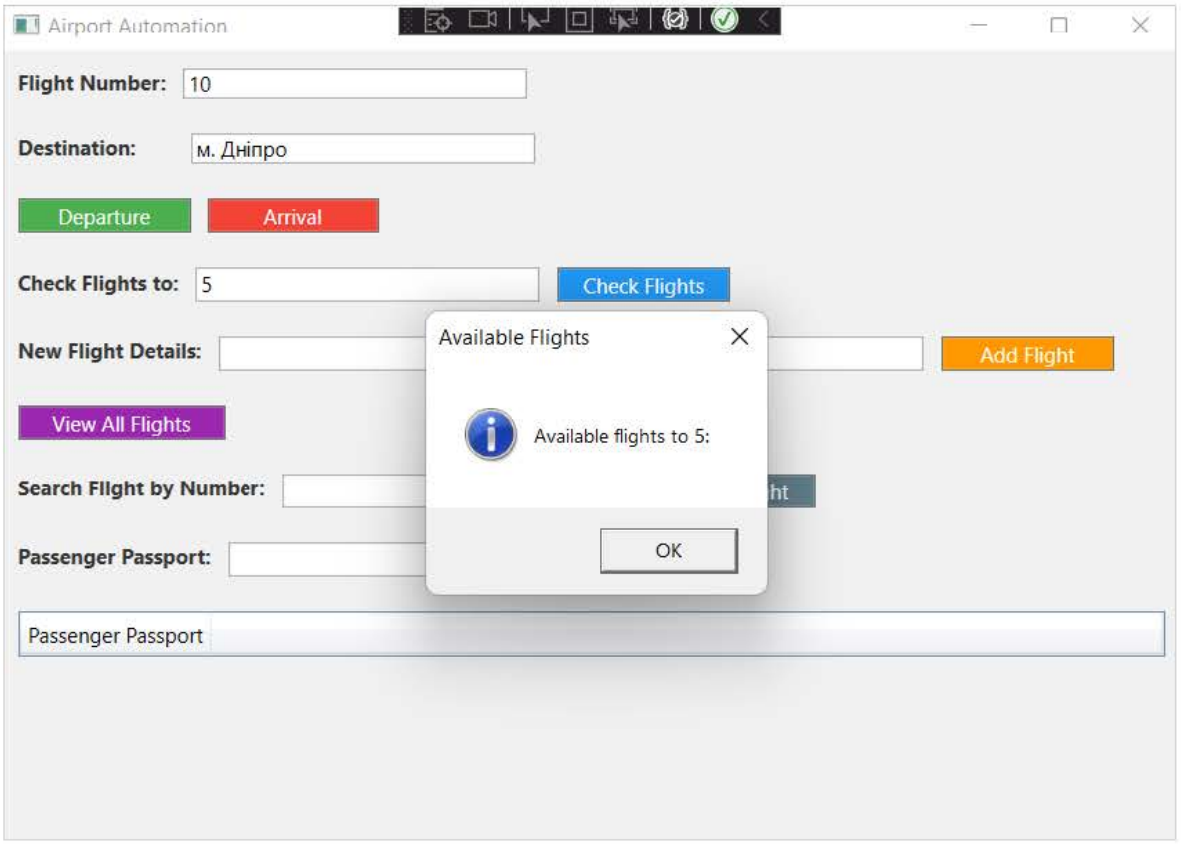

Рисунок 3.7 – Перевірка наявних рейсів для літака з відповідним номером

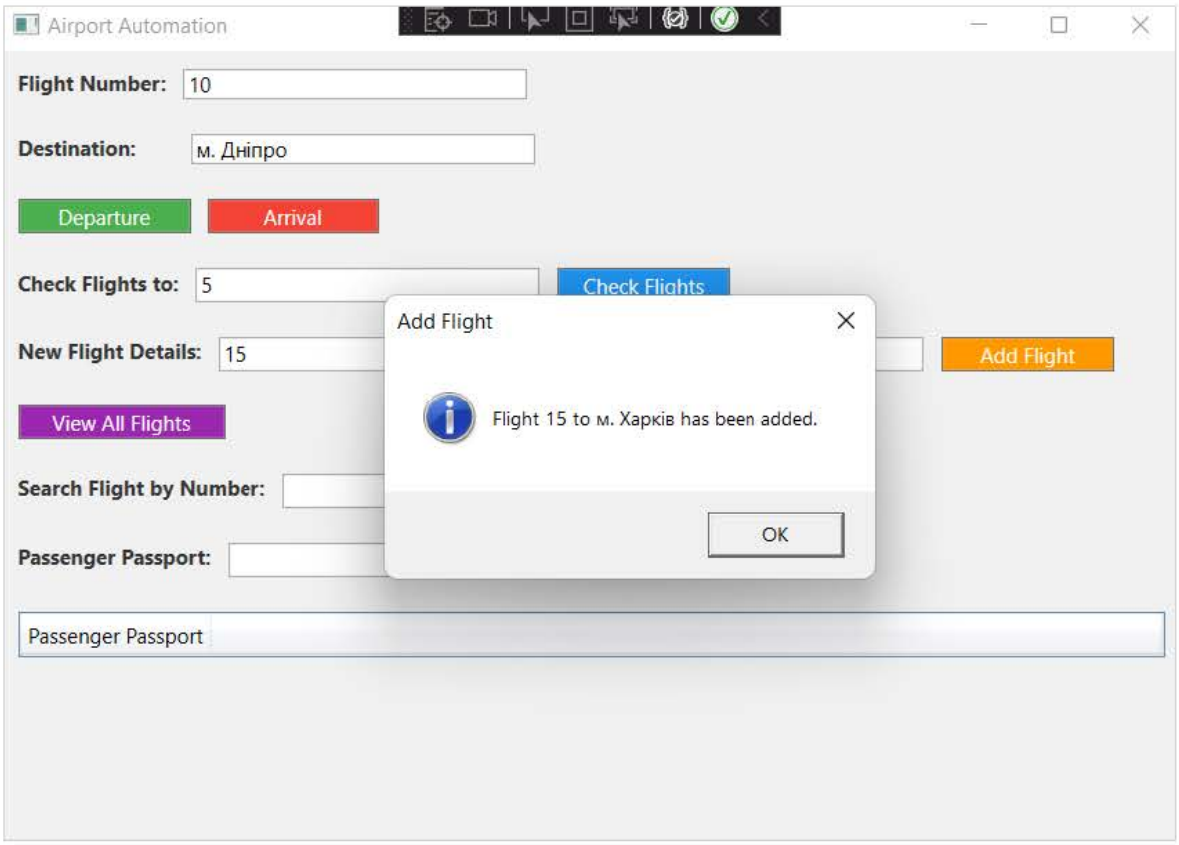

Рисунок 3.8 – Додавання нового літака

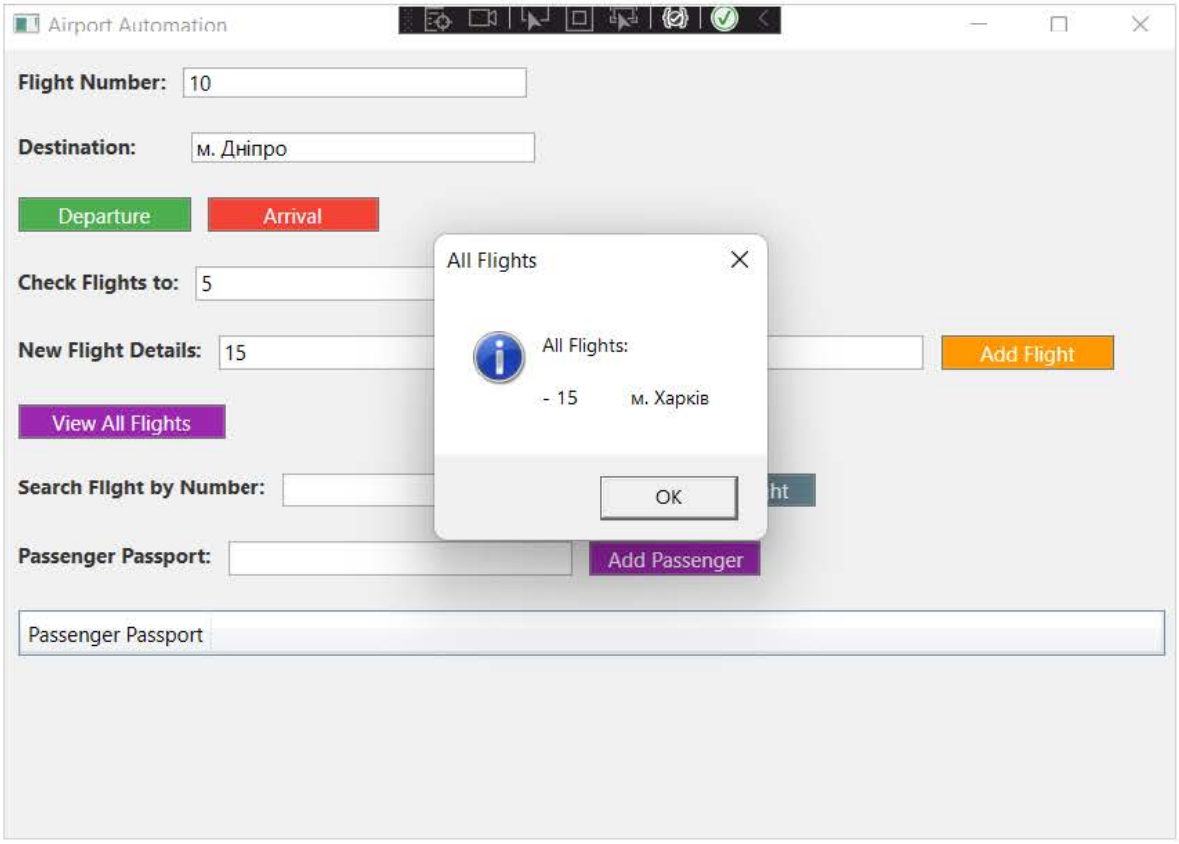

Рисунок 3.9 – Перегляд усіх існуючих літаків

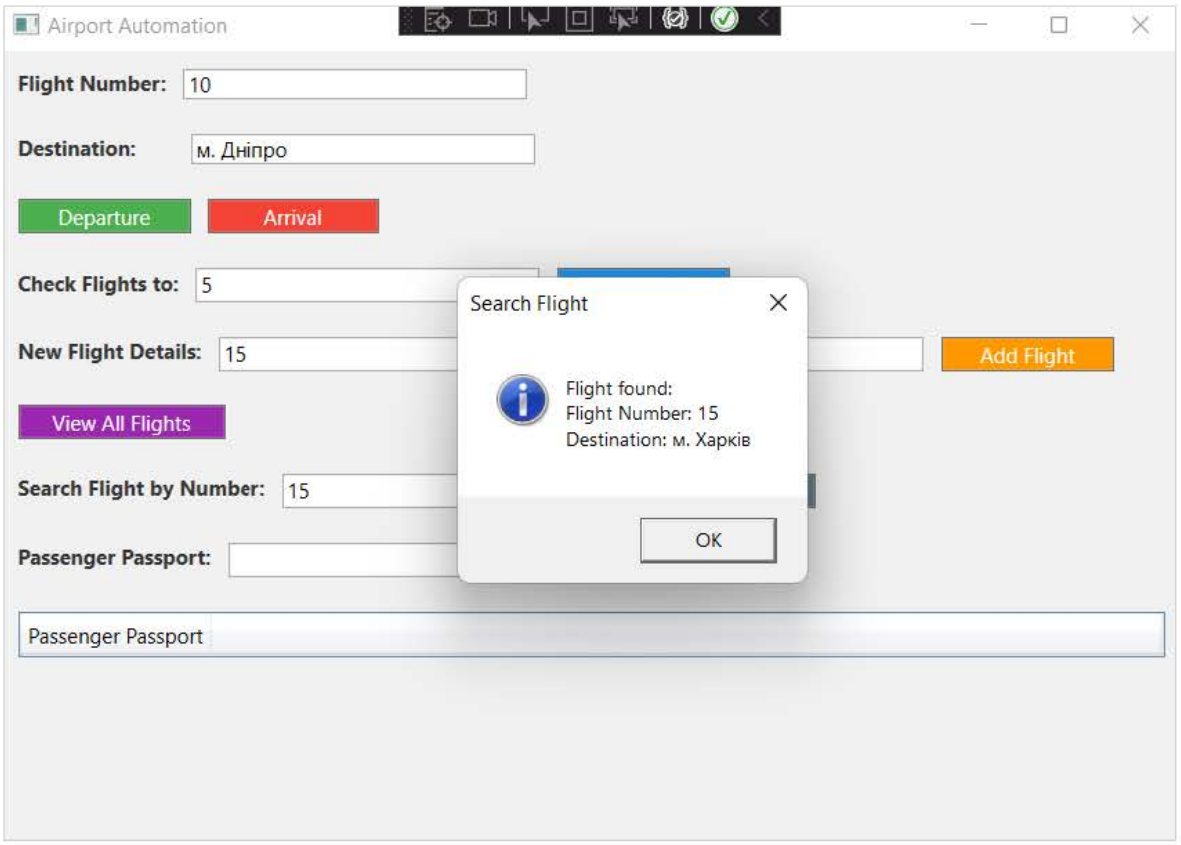

Рисунок 3.10 – Перегляд інформації про конкретний літак

На рисунку 3.11 зображено функціонал додавання нового пасажира за номером його паспорта.

На рисунку 3.12 зображено результат додавання нового пасажира за його номером паспорта.

Не вказавши номер паспорта пасажира, додати його неможливо (рис. 3.13).

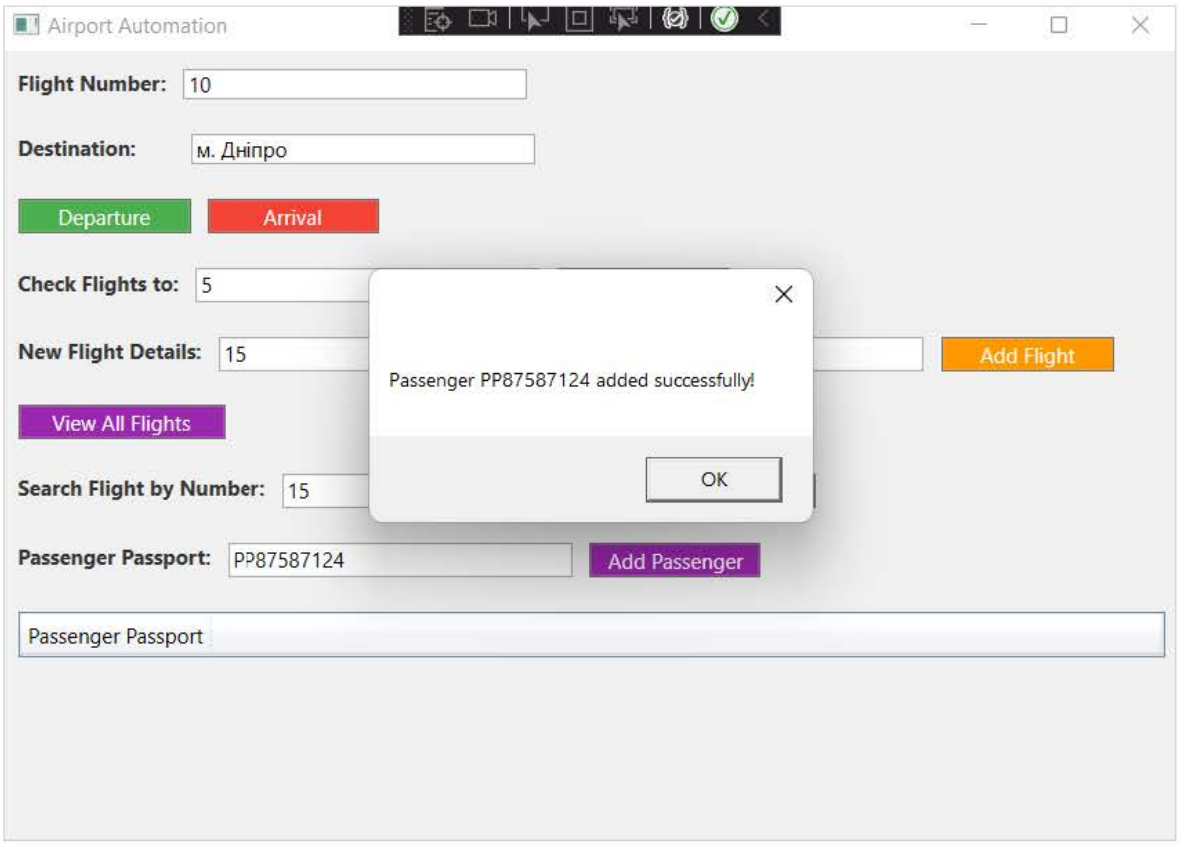

Рисунок 3.11 – Додавання нового пасажира за номером його паспорта

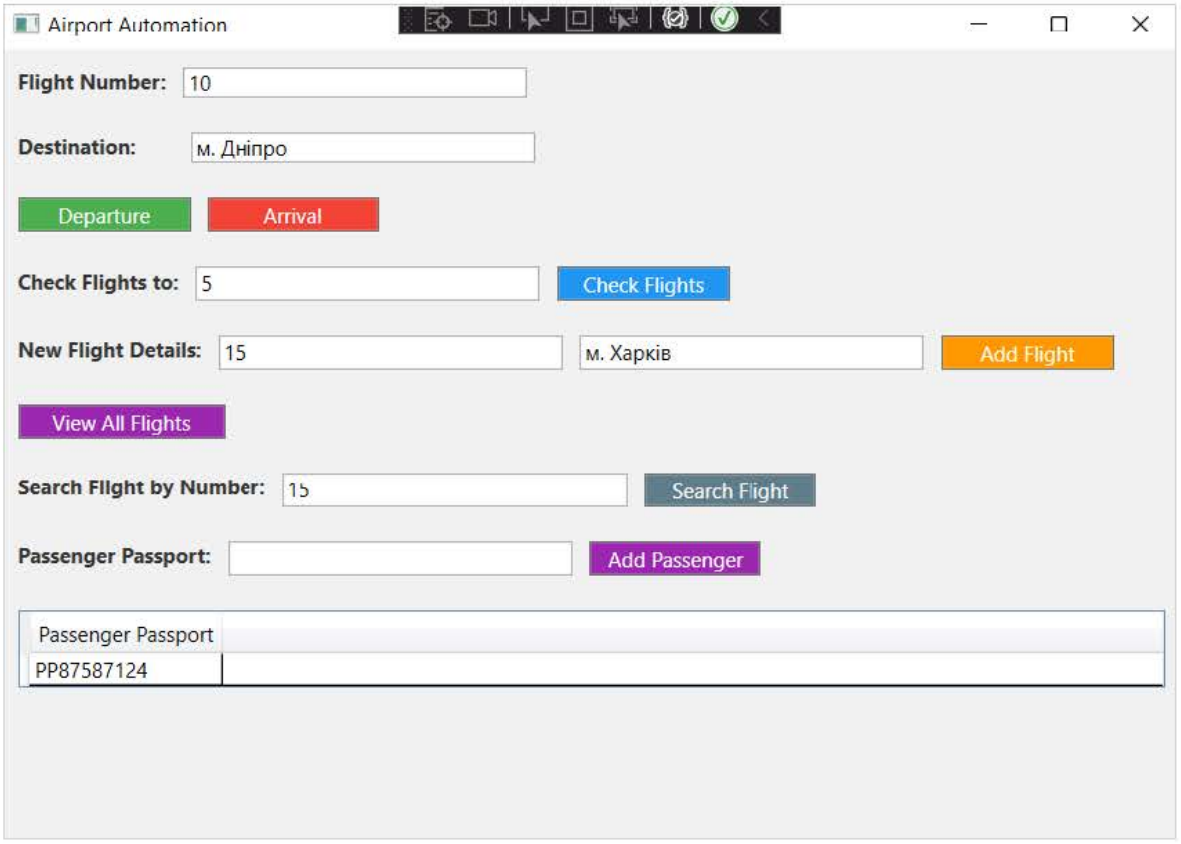

Рисунок 3.12 – Перегляд новододаного пасажира за номером його паспорта

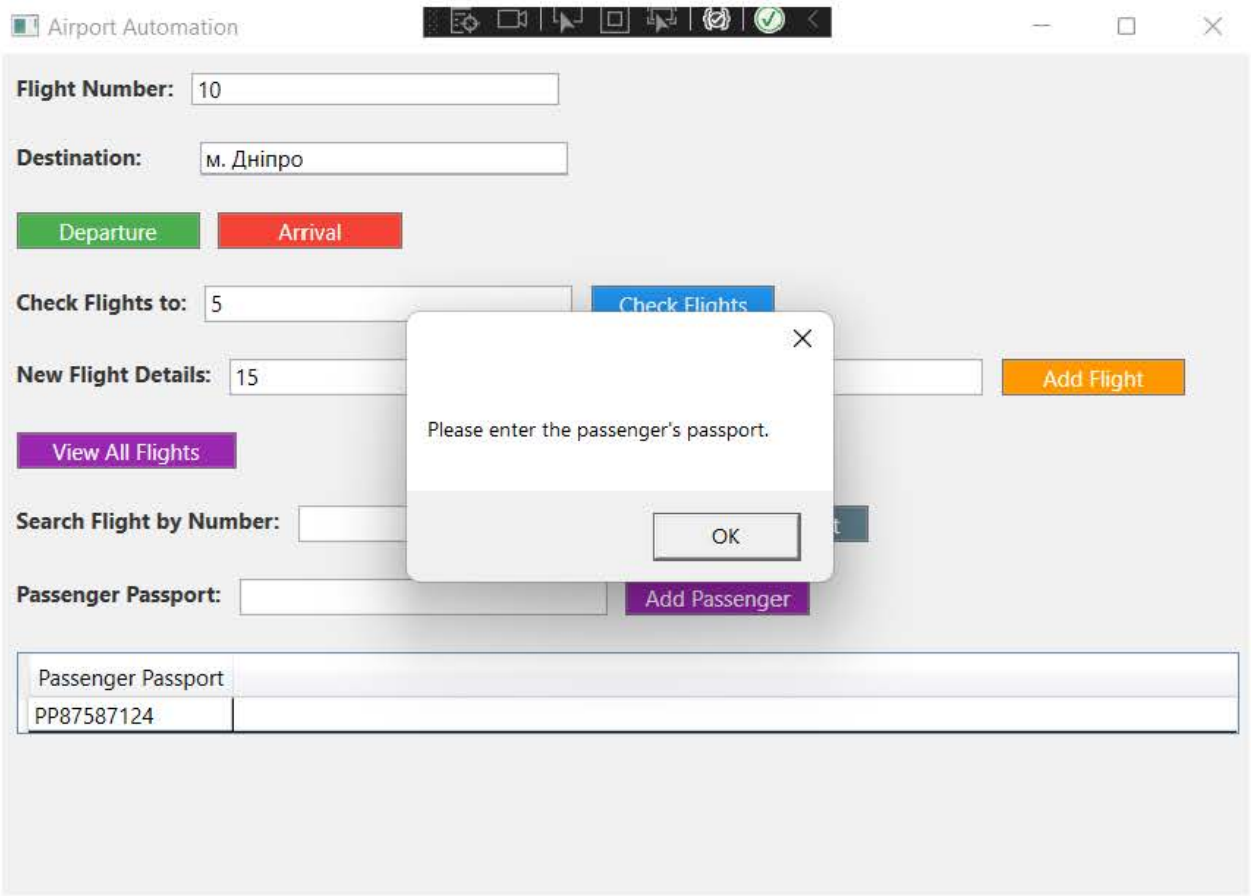

Рисунок 3.13 – Попередження про те, що номер паспорта пасажира не був вказаний

3.6 Висновки до третього розділу

У поточному розділі був проведений аналіз та опис процесу розробки програмного забезпечення для автоматизації роботи аеропорту. В результаті було отримано наступні висновки:

1) побудова проекту. У ході розробки було визначено структуру проекту, яка забезпечує організацію та керування всіма компонентами програмного рішення. Було розроблено архітектуру проекту, визначено основні модулі та їх взаємодію. Також було виконано проектування та опис класів, які становлять основу програмного рішення;

2) опис роботи програмного забезпечення для автоматизації роботи аеропорту. У цьому підрозділі було надано детальний опис функціональності розробленого програмного рішення. Було розглянуто основні завдання, які

виконує програмне забезпечення, такі як керування пасажирськими потоками, автоматизація багажних процесів, контроль безпеки та інші аспекти роботи аеропорту;

3) проектування інтерфейсу користувача. Був проведений аналіз та розробка інтерфейсу програмного рішення. Приділено увагу зручності використання, навігації та наданню необхідної інформації для операторів та користувачів системи автоматизації аеропорту;

4) проектування бази даних. Було розроблено структуру бази даних, яка зберігає та обробляє інформацію, необхідну роботи програмного рішення. Визначено сутності, зв'язки та атрибути, а також розроблено механізми забезпечення цілісності даних та безпеки;

5) тестування програмного забезпечення для автоматизації роботи аеропорту. Було проведено етап тестування розробленого програмного рішення. Були виконані функціональні та навантажувальні тести з метою перевірки працездатності, надійності та ефективності системи.

В результаті описаного розділу були отримані важливі етапи та результати розробки програмного забезпечення для автоматизації роботи аеропорту. Це дозволяє зробити висновок про те, що розроблене програмне рішення здатне оптимізувати процеси в аеропорту, забезпечити ефективне управління ресурсами та підвищити загальну безпеку та якість обслуговування.

ВИСНОВКИ

Метою роботи було розроблення програмного забезпечення для автоматизації роботи аеропорту.

Для досягнення поставленої мети перед кваліфікаційною роботою ставились наступні завдання:

1) аналіз та розуміння вимог. Необхідно було провести детальний аналіз вимог авіаційної індустрії, аеропорту та його підсистем. Це включало вивчення процесів, протоколів та стандартів, а також взаємодію із заінтересованими сторонами, такими як оператори, пасажири та служби безпеки. Це дозволило визначити функціональні та нефункціональні вимоги до програмного забезпечення;

2) проектування системи. На основі аналізу вимог необхідно було розробити архітектуру та дизайн програмного забезпечення. Це включало визначення модулів, компонентів, інтерфейсів та алгоритмів, а також вибір відповідних технологій та платформ розробки;

3) розробка та тестування. На цьому етапі відбувалась фактична розробка програмного забезпечення, включаючи написання коду, створення баз даних та налаштування необхідних інтеграцій. Після завершення розробки було проведене ретельне тестування для перевірки працездатності, надійності та відповідності вимогам. Також було проведене тестування на реальних даних та реальних умовах, щоб переконатися в ефективності програмного забезпечення;

4) підтримка та оновлення. Розробка програмного забезпечення для автоматизації роботи аеропорту – це безперервний процес. Після впровадження був передбачений варіант забезпечення постійної підтримки системи, реагування на зворотний зв'язок користувачів, виправлення помилок та випуск оновлень для покращення функціональності та безпеки.

Об'єктом дослідження були процеси роботи програмного забезпечення для автоматизації роботи аеропорту.

Предметом дослідження було апаратно-програмне забезпечення для розроблення програмного забезпечення для автоматизації роботи аеропорту.

Практичне значення одержаних результатів полягає у підвищенні якості автоматизації роботи аеропорту.

Результатами роботи є програмне забезпечення для автоматизації роботи аеропорту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yuen S. Mastering Windows Presentation Foundation: Build responsive UIs for desktop applications with WPF, 2nd Edition, 2020. 638 p.

2. MacDonald M. Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5, 2012. 1111 p.

3. Nathan A. WPF 4.5 Unleashed, 2013. 864 p.

4. Weil A. Learn WPF MVVM - XAML, C# and the MVVM pattern: Be ready for coding away next week using WPF and MVVM, 2016. 176 p.

5. Smith J. P. Entity Framework Core in Action, Second Edition, 2021. 1285 p.

6. Schwichtenberg H. Modern Data Access with Entity Framework Core: Database Programming Techniques for .NET, .NET Core, UWP, and Xamarin with C#, 2018. 817 p.

7. Gorman B. L. Practical Entity Framework Core 6: Database Access for Enterprise Applications, 2021. 828 p.

8. Lerman J. Programming Entity Framework: Building Data Centric Apps with the ADO.NET Entity Framework, 2010. 920 p.

9. Lerman J., Miller R. Programming Entity Framework: Code First: Creating and Configuring Data Models from Your Classes, 2011. 194 p.

10. Lerman J. Programming Entity Framework, 2009. 1286 p.

11. Lerman J., Miller R. Programming Entity Framework: DbContext: Querying, Changing, and Validating Your Data with Entity Framework, 2012. 258 p.

12. Gorman B. L. Practical Entity Framework: Database Access for Enterprise Applications, 2020. 836 p.

13. MacVittie L. A. XAML in a Nutshell: A Desktop Quick Reference (In a Nutshell (O'Reilly)), 2006. 306 p.

14. Dalal M., Ghoda A. XAML Developer Reference, 2011. 342 p.

15. Hermes D., Mazloumi N. Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals, 2019. 458 p.

16. Raúl R. A. LINQ to SQL: Manual para desarrolladores novatos, 2018. 98 p.

17. Albahari J., Albahari B. LINQ Pocket Reference: Learn and Implement LINQ for .NET Applications (Pocket Reference (O'Reilly)), 2008. 174 p.

ДОДАТОК А

ЛІСТИНГ СТВОРЕННЯ БАЗИ ДАНИХ

```
using System.ComponentModel.DataAnnotations;

namespace AirportAutomation.Models
{
    public class Flight
    {
        [Key]
        public int Id { get; set; }
        public string FlightNumber { get; set; }
        public string Destination { get; set; }
    }

    public class Passenger
    {
        [Key]
        public int Id { get; set; }
        public string Passport { get; set; }
    }
}

using System.Data.Entity;

namespace AirportAutomation.Models
{
    public class AirportContext : DbContext
    {
```

```
        public DbSet<Flight> Flights { get; set; }
        public DbSet<Passenger> Passengers { get; set; }
    }
}
```

ДОДАТОК Б

ЛІСТИНГ ПРОЕКТУВАННЯ ІНТЕРФЕЙСІВ

```
<Application x:Class="WpfApp1.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:WpfApp1"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>

<Window x:Class="AirportAutomation.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Airport Automation" Height="500" Width="700">
    <Grid Background="#F1F1F1">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
```

```
        <StackPanel Grid.Row="0" Orientation="Horizontal" Margin="10">
            <TextBlock Text="Flight Number:" Margin="0 0 10 0"
FontWeight="Bold" Foreground="#333333"/>
            <TextBox x:Name="TbFlightNumber" Width="200"/>
        </StackPanel>

        <StackPanel Grid.Row="1" Orientation="Horizontal" Margin="10">
            <TextBlock Text="Destination:" Margin="0 0 32 0" FontWeight="Bold"
Foreground="#333333"/>
            <TextBox x:Name="TbDestination" Width="200"/>
        </StackPanel>

        <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="10">
            <Button x:Name="BtnDeparture" Content="Departure" Width="100"
Margin="0 0 10 0" Click="BtnDeparture_Click" Background="#4CAF50"
Foreground="White"/>
            <Button x:Name="BtnArrival" Content="Arrival" Width="100"
Click="BtnArrival_Click" Background="#F44336" Foreground="White"/>
        </StackPanel>

        <StackPanel Grid.Row="3" Orientation="Horizontal" Margin="10">
            <TextBlock Text="Check Flights to:" Margin="0 0 10 0"
FontWeight="Bold" Foreground="#333333"/>
            <TextBox x:Name="TbDestinationCheck" Width="200"/>
            <Button x:Name="BtnCheckFlights" Content="Check Flights"
Width="100" Margin="10 0 0 0" Click="BtnCheckFlights_Click"
Background="#2196F3" Foreground="White"/>
        </StackPanel>
```

```
        <StackPanel Grid.Row="4" Orientation="Horizontal" Margin="10">
            <TextBlock Text="New Flight Details:" Margin="0 0 10 0"
FontWeight="Bold" Foreground="#333333"/>
            <TextBox x:Name="TbNewFlightNumber" Width="200"/>
            <TextBox x:Name="TbNewDestination" Width="200" Margin="10 0 0
0"/>
            <Button x:Name="BtnAddFlight" Content="Add Flight" Width="100"
Margin="10 0 0 0" Click="BtnAddFlight_Click" Background="#FF9800"
Foreground="White"/>
        </StackPanel>

        <StackPanel Grid.Row="5" Orientation="Horizontal" Margin="10">
            <Button x:Name="BtnViewAllFlights" Content="View All Flights"
Width="120" Click="BtnViewAllFlights_Click" Background="#9C27B0"
Foreground="White"/>
        </StackPanel>

        <StackPanel Grid.Row="6" Orientation="Horizontal" Margin="10">
            <TextBlock Text="Search Flight by Number:" Margin="0 0 10 0"
FontWeight="Bold" Foreground="#333333"/>
            <TextBox x:Name="TbSearchFlightNumber" Width="200"/>
            <Button x:Name="BtnSearchFlight" Content="Search Flight"
Width="100" Margin="10 0 0 0" Click="BtnSearchFlight_Click"
Background="#607D8B" Foreground="White"/>
        </StackPanel>

        <StackPanel Grid.Row="7" Orientation="Horizontal" Margin="10">
            <TextBlock Text="Passenger Passport:" Margin="0 0 10 0"
FontWeight="Bold" Foreground="#333333"/>
            <TextBox x:Name="TbPassengerPassport" Width="200"/>
```

```
        <Button x:Name="BtnAddPassenger" Content="Add Passenger"
Width="100" Margin="10 0 0 0" Click="BtnAddPassenger_Click"
Background="#9C27B0" Foreground="White"/>
    </StackPanel>
    <DataGrid x:Name="PassengersDataGrid" Grid.Row="8" Margin="10"
AutoGenerateColumns="False">
        <DataGrid.Columns>
            <DataGridTextColumn Header="Passenger Passport"
Binding="{Binding Passport}"/>
        </DataGrid.Columns>
    </DataGrid>
  </Grid>
</Window>
```

ДОДАТОК В

ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;

namespace AirportAutomation
{
   public partial class MainWindow : Window
   {
      private List<Flight> flights;
      private List<Passenger> passengers;
      private AirportContext context;


      public MainWindow()
      {
         InitializeComponent();

         // Ініціалізація списків літаків та пасажирів
         flights = new List<Flight>();
         passengers = new List<Passenger>();

               // Ініціалізація контексту бази даних
               context = new AirportContext();
      }
```

```
        private void BtnAddPassenger_Click(object sender, RoutedEventArgs e)
        {
            string passengerName = TbPassengerPassport.Text;

            if (!string.IsNullOrEmpty(passengerName))
            {
                Passenger passenger = new Passenger(passengerName);
                passengers.Add(passenger);
                context.Passengers.Add(passenger);
                context.SaveChanges();

                MessageBox.Show($"Passenger {passengerName} added
successfully!");

                // Очищення поля для введення імені пасажира
                TbPassengerPassport.Text = string.Empty;

                // Оновлення таблиці з пасажирами
                RefreshPassengersDataGrid();
            }
            else
            {
                MessageBox.Show("Please enter the passenger's passport.");
            }
        }

        private void RefreshPassengersDataGrid()
        {
            PassengersDataGrid.ItemsSource = null;
            PassengersDataGrid.ItemsSource = passengers;
```

```
    }

    private void BtnDeparture_Click(object sender, RoutedEventArgs e)
    {
        string flightNumber = TbFlightNumber.Text;
        string destination = TbDestination.Text;

        // Perform necessary actions for departure, e.g., update database, display
messages, etc.

        MessageBox.Show($"Flight {flightNumber} to {destination} has
departed.", "Departure", MessageBoxButton.OK, MessageBoxImage.Information);
    }

    private void BtnArrival_Click(object sender, RoutedEventArgs e)
    {
        string flightNumber = TbFlightNumber.Text;
        //string origin = TbOrigin.Text;

        // Perform necessary actions for arrival, e.g., update database, display
messages, etc.

        MessageBox.Show($"Flight {flightNumber} has arrived.", "Arrival",
MessageBoxButton.OK, MessageBoxImage.Information);
    }

    private void BtnCheckFlights_Click(object sender, RoutedEventArgs e)
    {
        string destination = TbDestinationCheck.Text;
```

```
        List<Flight> availableFlights = GetAvailableFlights(destination);

        string flightsMessage = "Available flights to " + destination + ":\n";
        foreach (Flight flight in availableFlights)
        {
            flightsMessage += "- " + flight.FlightNumber + "\t" + flight.Destination
+ "\n";
        }
        MessageBox.Show(flightsMessage, "Available Flights",
MessageBoxButton.OK, MessageBoxImage.Information);
    }

    private List<Flight> GetAvailableFlights(string destination)
    {
        List<Flight> availableFlights = new List<Flight>();

        foreach (Flight flight in flights)
        {
            if (flight.Destination == destination)
            {
                availableFlights.Add(flight);
            }
        }

        return availableFlights;
    }

    private void BtnAddFlight_Click(object sender, RoutedEventArgs e)
    {
        string flightNumber = TbNewFlightNumber.Text;
```

```
        string destination = TbNewDestination.Text;

        Flight newFlight = new Flight(flightNumber, destination);
        flights.Add(newFlight);

        MessageBox.Show($"Flight {flightNumber} to {destination} has been
added.", "Add Flight", MessageBoxButton.OK, MessageBoxImage.Information);
    }

    private void BtnViewAllFlights_Click(object sender, RoutedEventArgs e)
    {
        string allFlightsMessage = "All Flights:\n";
        foreach (Flight flight in flights)
        {
            allFlightsMessage += "- " + flight.FlightNumber + "\t" +
flight.Destination + "\n";
        }
        MessageBox.Show(allFlightsMessage, "All Flights",
MessageBoxButton.OK, MessageBoxImage.Information);
    }

    private void BtnSearchFlight_Click(object sender, RoutedEventArgs e)
    {
        string searchFlightNumber = TbSearchFlightNumber.Text;

        Flight foundFlight = SearchFlightByNumber(searchFlightNumber);

        if (foundFlight != null)
        {
```

```
            MessageBox.Show($"Flight found:\nFlight Number:
{foundFlight.FlightNumber}\nDestination: {foundFlight.Destination}", "Search
Flight", MessageBoxButton.OK, MessageBoxImage.Information);
        }
        else
        {
            MessageBox.Show("Flight not found.", "Search Flight",
MessageBoxButton.OK, MessageBoxImage.Warning);
        }
    }

    private Flight SearchFlightByNumber(string flightNumber)
    {
        foreach (Flight flight in flights)
        {
            if (flight.FlightNumber == flightNumber)
            {
                return flight;
            }
        }

        return null;
    }
}

public class Flight
{
    public string FlightNumber { get; set; }
    public string Destination { get; set; }
```

```
        public Flight(string flightNumber, string destination)
        {
            FlightNumber = flightNumber;
            Destination = destination;
        }
    }

    public class Passenger
    {
        public string Passport { get; set; }

        public Passenger(string passport)
        {
            Passport = passport;
        }
    }
}
```