Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича
Інститут фізико-технічних та комп'ютерних наук
(повна назва інституту/факультету)
Кафедра інформаційних технологій та комп'ютерної фізики
(повна назва кафедри)

Продаж залізничних квитків

Дипломна робота
Рівень вищої освіти - перший (бакалаврський)

Виконав:
студент 4 курсу, групи 417ск
спеціальності
126 – інформаційні системи та технології
(шифр і назва спеціальності)
Сироватка І. Ю. _____
(прізвище та ініціали)
Керівник: доктор технічних наук, доцент
Баловсяк Сергій Васильович _____
(науковий ступінь, вчене звання, прізвище, ім'я, по-батькові)
Рецензент_____
_____
(науковий ступінь, вчене звання, прізвище, ім'я, по-батькові)

До захисту допущено:
Протокол засідання кафедри № 19
від „ 17 ” червня 2021 р.
зав. кафедри _____ доц. Борча М.Д.

Чернівці – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Інститут фізико-технічних та комп'ютерних наук
Кафедра інформаційних технологій та комп'ютерної фізики

ЗАТВЕРДЖУЮ
Завідувач кафедри
докт. фіз.-мат. наук, доц.
_____М. Д. Борча
" ___" _____ 2021 р.

ПРОДАЖ ЗАЛІЗНИЧНИХ КВИТКІВ

ЛИСТ ЗАТВЕРДЖЕННЯ

УЗГОДЖЕНО
Керівник роботи
докт. техн. наук, доцент
_____ С.В. Баловсяк
"___" _____ 2021 р.

Виконавець
студент 4-го курсу
_____ І. Ю. Сироватка
"___" _____ 2021 р.

2021

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Сироватці Івану Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Продаж залізничних квитків

керівник роботи *Баловсяк Сергій Васильович, докт. техн. наук, доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від "__"_____202_ року №_____

2. Строк подання студентом проекту (роботи) *27 травня 2021 р.*

3. Вихідні дані до проекту (роботи) *Мета роботи – створення програми для продажу залізничних квитків. Середовище розробки Visual Studio 2019, мова програмування – C#. 7.База даних Microsoft SQL Server 2019.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) описати існуючі системи продажу квитків

2) описати структуру та функції системи

3) створити програму для продажу залізничних квитків

4) дослідити ефективність розробленої програми

5. Перелік графічного матеріалу

1) Структура таблиць бази даних

2) Приклад роботи програми

3) Діаграма прецедентів програми

Студент _____ *Сироватка І.Ю.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Баловсяк С.В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота виконана студентом групи 417 ск Сироваткою Іваном Юрійовичем. Тема «Продаж залізничних квитків». Робота направлена на здобуття ступеня бакалавр за спеціальністю 126 «Інформаційні системи та технології».

Метою кваліфікаційної роботи є створення та розробка інформаційної системи “Продаж залізничних квитків”.

В результаті виконання кваліфікаційної роботи було створено програмний продукт продажі квитків на потяги, який би забезпечив потрібний нам функціонал, зручний інтерфейс та швидкий доступ

Кількість сторінок – 40, таблиць – 1, рисунків – 13, додатків – 1, джерел – 15.

Ключові слова: інформаційна система, бронювання, залізничні квитки, база даних, критерії якості системи.

ABSTRACT

Qualification work was performed by a student of the 417 group Syrovatka Ivan. Topic "Information system for evaluating the effectiveness of the computer network of the enterprise". The work is aimed at obtaining a bachelor's degree in specialty 126 "Information Systems and Technologies".

The purpose of the qualification work is to create and develop an information system "Sale of railway tickets".

As a result of the qualification work, a software product for ticket sales was created, which would provide the functionality we need, a user-friendly interface and quick access.

Number of pages - 40, tables - 1, figures - 13, appendices - 1, sources - 15.

Key words: information system, reservation, railway tickets, database, system quality criteria.

ЗМІСТ

ВСТУП

Залізничний транспорт як один із найважливіших видів транспорту відіграє важливе значення у перевезенні пасажирів, щороку кількість бажаючих скористатися потягом зростає. Маючи таку величезну клієнтську базу, проблема придбання залізничних квитків була дуже помітною.

Клієнтська частина системи - це додаток, який надає інтерфейс для придбання квитків. Доступ до різних функцій системи обмежений залежно від прав користувача.

Основне призначення даної розробки – створення сучасного додатку, який буде являти собою онлайн придбання залізничних квитків. Та забезпечувати достатнім функціоналом обслуговуючий персонал залізниці.

Одним з основних кроків у створенні програмного продукту є розробка архітектури рівня баз даних. Питання ефективного зберігання та використання великих даних та ефективного пошуку інформації стає дуже важливим у багатьох сферах нашого часу. Тому я вирішив використовувати базу даних Microsoft SQL Server насамперед через швидкість, тому що коли накопичуються дані, це відіграє важливу роль у продуктивнос

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Системний аналіз об’єкту дослідження та предметної області

1.1.1 Аналіз мети функціонування системи

Основне призначення даної розробки – створення сучасного додатку, який буде являти собою онлайн купівлю і бронювання залізничних квитків. Програмний модуль надає послуги, пов’язані із оглядом та замовленням квитків для користувачів, які бажають подорожувати залізницею, переглядати популярні маршрути та обирати той який до вподоби. Простота управління та інтуїтивно зрозумілий інтерфейс дозволяє без проблем користуватися додатком одразу після його відкриття.

Даний продукт буде корисним для осіб, які бажають придбати квитки на потяг онлайн. Також, ця програма надає можливість адміністратору системи зберігати та редагувати інформацію про залізничні маршрути, потяги, які їх обслуговують, проміжні станції, розклад руху потягів до місця призначення, типи квитків та їх вартість.

Областю застосування може бути національна залізниця, яка при впровадженні даної системи підвищить рентабельність, зменшить трудомісткість робочого процесу звичайного касира, та отримає ряд функцій з якими стане приємною, як для власника та працівників так і для звичайного покупця.

1.1.2 Забезпечення якості

Для надійності можна розрахувати величину економічного ризику, спричиненого можливістю помилок у програмі. По суті, цей ризик визначається розділами інструкцій, що встановлюють використання результатів програми в даному режимі роботи, та ймовірністю помилок, що впливають на кожен тип використання. Такий ризик може базуватися лише на прогнозуванні можливих помилок, що значно зменшує цінність такого

підходу. У той же час доцільно оцінити фактичні втрати за час експлуатації, спричинені помилками. Якщо ви визначите тенденцію зміни середніх реальних збитків за певний період (наприклад, на місяць чи рік), ви можете передбачити економічний ризик на майбутнє.

Часто оцінюється статистична надійність програми, яка вимірюється як додаткова ймовірність виявлення нової помилки, не врахована в попередніх виправленнях, у разі чергового виклику програми. Це відповідає підходу програми як чорного ящика, який іноді дає помилки. Недоліком цього підходу є неявно використана модель "повернення виборчих скриньок", яка не враховує коригування поїзда після виявлення кожної нової помилки. Слід також зазначити не цілком конструктивний, але логічно бездоганний підхід до визначення комбінаторної надійності програми, розрахованої як відношення кількості варіантів вихідних даних, на яких програма працює коректно, до загальної кількості варіантів вихідні дані. В умовах динамічного регулювання кортежу ця надійність постійно зростає. Однак його можна оцінити лише статистично.

Нестабільність програми, природно, вимірюється кількістю зареєстрованих помилок за певний період роботи, тобто кількістю коригувань, внесених до інструкцій за цей період. Оскільки технічний знос програми, інструкцій та режиму роботи відсутній, нестабільність зносу комп’ютера монотонно падає через коригування. У міру зношування комп’ютера (старіння технічної частини навколишнього середовища) проводяться спеціальні корективи, щоб уникнути поломок та поломок машини: нестабільність починає зростати. За нормального режиму роботи, своєчасна заміна обладнання та підвищена нестабільність. Можна також говорити про прогнозовану залишкову надійність кортежу з урахуванням вартості використання виправлень, але без витрат на їх розробку та прогнозу кількості нових помилок протягом решти терміну служби з вартістю їх аналіз. Ця характеристика буде називатися надійністю витрат.

Додаток «Продаж залізничних квитків» повинен містити в собі:

1. Інформація про маршрути перевезення (номер маршруту, пункт призначення, район, область, час відправлення, час прибуття).

2. Інформація про продані білети на даний рейс (місце, вартість, час придбання, прізвище пасажира, паспортні дані пасажира).

3. Інформація про диспетчера, що обслуговує клієнта (прізвище диспетчера, дата народження, адреса).

4. Інформація про потяг на якому здійснюється перевезення (номер потяга, номер, максимальна кількість посадкових місць, тип місць).

5. Номер перевезення.

1.2 Постановка задачі

1.2.1 Вимоги до системи

Розробити додаток який може зберігати та редагувати інформацію про залізничні маршрути, потяги, які їх обслуговують, проміжні станції, розклад руху потягів до місця призначення, типи квитків та їх вартість. Користувачі можуть виконувати пошук та перегляд інформації про маршрути та замовляти електронний квиток.

Програмний модуль повинен забезпечувати роботу трьом категоріям користувачів: адміністратор, касир, покупці.

1. Ідентифікація користувачів відповідних категорій повинна здійснюватися за допомогою логіна та паролю, які вводяться при покупці квитка.

2. Покупець повинен сформувати необхідний для нього перелік категорій для покупки квитка.

2.1. Користувач повинен мати можливість ознайомитися з усім переліком маршрутів, які обслуговують різні потяги.

2.2. Користувач повинен мати можливість замовляти електронний квиток, після проходження реєстрації.

3.1. Касир повинен мати можливість переглядати інформацію, введену адміністратором.

3.2. Касир повинен мати можливість здійснювати пошук потяга та потрібної станції.

3.3. Касир повинен мати можливість обирати маршрут до пункту призначення.

3.4. Здійснювати продаж квитків з наступним внесенням інформації до системи.

4. Програма повинна дозволяти зберігати та редагувати інформацію про залізничні маршрути.

5. Адміністратор повинен мати можливість керуванням загальною роботою системи.

5.1. Адміністратор повинен мати можливість зберігати та редагувати інформацію про залізничні маршрути.

5.2. Адміністратор повинен мати можливість зберігати та редагувати розклад руху потягів до місця призначення

5.3. Адміністратор повинен мати можливість зберігати та редагувати інформацію про потяги, які обслуговують залізничні маршрути.

5.4. Адміністратор повинен мати можливість зберігати та редагувати інформацію про проміжні станції.

5.5. Адміністратор повинен мати можливість зберігати та редагувати типи квитків та їх вартість.

1.2.2 Очікувані ефекти від впровадження

Є декілька сервісів для продажу квитків. Для порівняння бронювання квитків я обрав сервіс proizd.ua.

Цей сервіс являє собою онлайн бронювання залізничних квитків.

До переваг цього додатку можна віднести:

1. Простий інтерфейс.
2. Швидке бронювання квитка.
3. Зрозумілість.

Недоліками цього додатку є:

1. Відсутня інформація про маршрути слідування.
2. Немає розкладу потягів.
3. Труднощі з обміном квитків.
4. Повернення коштів здійснюється не у повному обсязі.
5. Відсутня панель адміністратора.

Головне вікно додатку (рисунок 1.1).

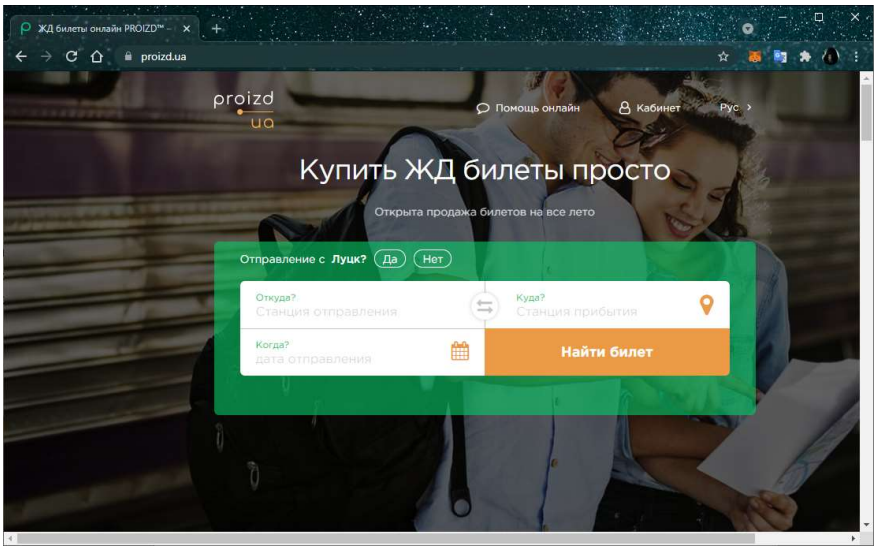

Рисунок 1.1 – Головне вікно додатку proizd.ua

На основі цього поріняння я вирішив спроектувати додаток, який буде в собі містити інтуїтивно-зрозумілий інтерфейс, швидке бронювання квитків, а також перекриє всі мінуси аналога.

Продаж залізничних квитків - це додаток, де можна замовити онлайн замовити квитки на потяг.

Цей додаток в основному створений для виконання відповідно до наступних вимог:

1. Додаток, який в режимі реального часу надаватиме інформацію про наявність квитків та їхню вартість.
2. Кожен зареєстрований користувач може переглянути свій ідентифікатор бронювання, який був зроблений на його ім'я.

3. Кожен зареєстрований користувач може змінити свій пароль у будь-який час.

4. Кожен користувач може шукати наявність поїздів, ціну квитка, прибуття та час відправлення, відстань між пунктом відправлення та пунктом призначення.

5. Кожен зареєстрований користувач має можливість роздрукувати свій квиток у будь-який час, коли він побажає.

6. В адміністративному режимі адміністратор може вносити зміни в деталі поїзда.

7. Він також може переглянути всі бронювання, зроблені користувачами.

Висновки до розділу

В даному розділі був виконаний аналіз мети функціонування системи та основних варіантів досягнення мети та наявних ресурсів. Виконана конкретизація вимог до системи. Розкриваються такі питання, як мета розроблення, призначення системи, місце застосування системи, постановка задачі. А також було виконано порівняння з аналогами.

РОЗДІЛ 2. ПОБУДОВА СИСТЕМИ

2.1 Моделювання системи

2.1.1 Вхідні дані

У цьому додатку є різноманітна інформація, яка допомагає щодо бронювання квитків.

Користувачі зможуть шукати наявність поїздів, час прибуття та відправлення поїзда, вони також можуть забронювати квиток, а після замовлення квитка, якщо користувач хоче його скасувати, він також може це легко зробити, не виходячи з дому.

Пасажири залізниці часто повинні знати про свій статус бронювання квитків, наявність у конкретному поїзді місця, деталі прибуття чи відправлення поїзда, спеціальні поїзди тощо.

Інформаційні центри клієнтів на залізничних станціях не можуть обслуговувати такі запити в пікові періоди.

У більшості систем бронювання є великі черги, тому обробку таких запитів потрібен тривалий час, щоб забронювати квиток.

Інтернет-система бронювання залізничних квитків має на меті розробити веб-додаток, який має на меті надання даних про потяги, а також можливість бронювання квитків через Інтернет для клієнтів.

Програму слід розділити на дві частини, а саме на частину користувача та адміністратора. І кожна з них має свої відповідні особливості.

Вхідними даними даної програми є запити користувача та введені ним дані у відведені під це форми для вводу.

2.1.2 Вихідні дані

Вихідними даними даної програми є відповіді на запити; залежно від типу запиту, можливі вихідні дані та зміни вікон від їхнього виконання (рисунок 2.1).

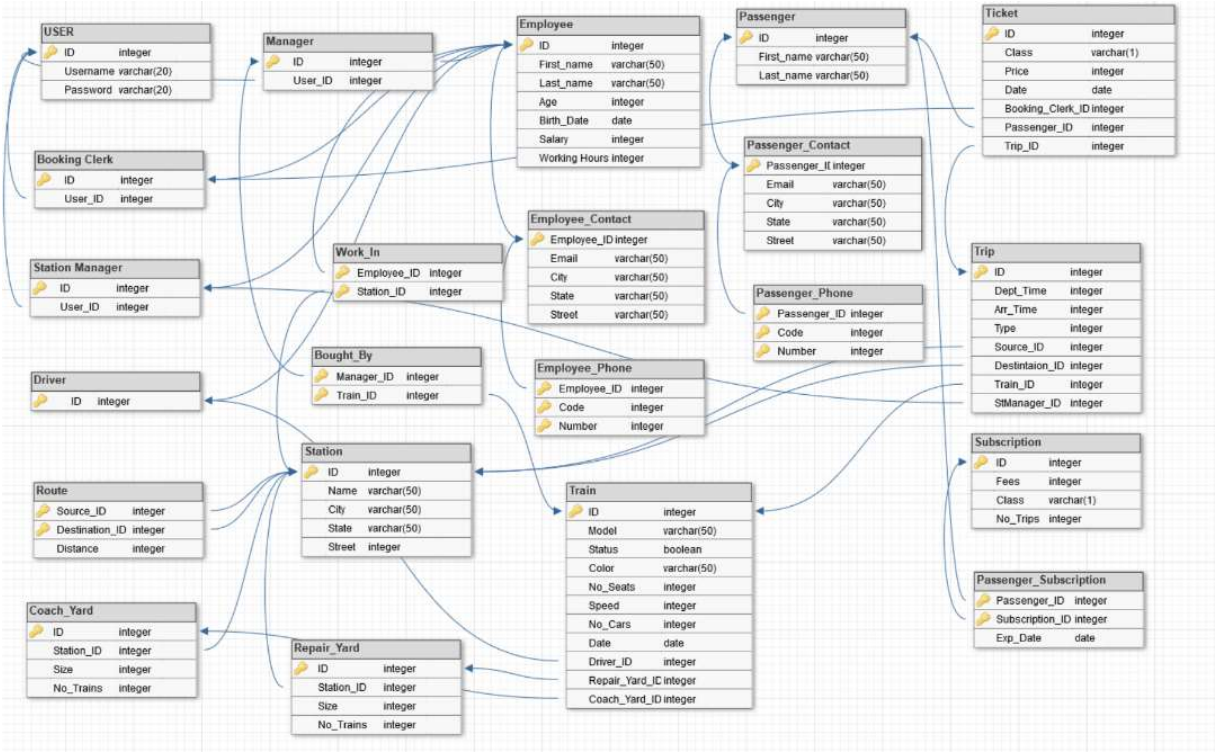

Рисунок 2.1 – Структура таблиць бази даних

Таблиця User призначена для збереження даних користувачів програми.

Таблиця Employee описує всіх людей, які працюють у системі (водії, службовці тощо.). А також дані, пов’язані з ними (імена, адреси тощо).

Таблиця Train описує поїзди та їх стан (модель, ремонт тощо).

Таблиця Route описує маршрути, якими рухатимуться поїзди, та станції, через які вони проїжджатимуть.

Таблиця Subscription містить в собі дані про передплату.

Таблиця Manager описує менеджера системи, який наймає та звільняє керівників станцій.

В таблиці Passenger описані пасажири поїзда.

Таблиця Trip описує пропоновані поїздки, включає станції відправлення та пункту призначення.

Таблиця Station описує залізничні станції та їх взаємозв’язок із сусідніми станціями та маршрутами, що проходять повз них.

В таблиці Ticket описуються квитки на кожну поїздку, а також номер місця.

В таблиці Contact описана вся інформація про робітників та пасажирів

В таблиці Repair yard описано депо, в яких пошкоджені поїзди ремонтуються і знову стають готовими до роботи, а також станція контролера та кількість поїздів, які наразі фіксуються.

Таблиця Coach yard описує депо, де відстоюються поїзди, коли їм не призначено поїздку, станцію контролера та максимальну кількість поїздів, яка вона може утримувати.

В таблиці Station manager описано керівника станції, який відповідає за найм та звільнення працівників та модернізацію станції, а також управління ремонтними майданчиками, якщо такі є під контролем станції.

Таблиця Booking clerk описує працівників, які відповідають за бронювання квитків для пасажирів.

2.1.3 Функції та структура системи

Пасажири можуть забронювати квитки на поїзд, в якому є вільні місця. Для цього пасажир повинен обрати бажаний номер поїзда та дату, на яку потрібно забронювати квиток.

Перед бронюванням квитка для пасажира перевіряється дійсність номера поїзда та дати бронювання. Після підтвердження номера поїзда та дати бронювання перевіряється наявність вільного місця. Якщо так, квиток оформляється із підтвердженням статусу та генерується відповідний ідентифікатор квитка, який зберігається разом з іншими даними пасажира.

Квиток після бронювання можна скасувати в будь-який час. Для цього пасажир повинен надати ідентифікатор квитка (унікальний ключ). Після цього шукається ідентифікатор квитка та видаляється відповідний запис. Завдяки цьому перший квиток із статусом очікування також отримує підтвердження.

Користувачі системи:

1. Керівник.
2. Керівник станції.
3. Пасажир.

4. Адміністратор.

Керівник має змогу виконувати такі функціональні можливості:

1. Найняти або звільнити керівника станції.
2. Оновлювати рухомий склад.
3. Відстежувати стан потягів.
4. Відкривати нові станції.
5. Змінювати зарплату керівників станцій.
6. Додавати або видаляти передплати квитків.
7. Оновлювати дані про станції.
8. Перегляд інформації про працівників.
9. Перегляд інформації про пасажирів.
10. Огляд поїздок.

Керівник станції може виконувати такі функціональні можливості:

1. Оновлювати дані про станції.
2. Перегляд даних про працівників.
3. Визначати ціну квитка.
4. Керувати поїздками.
5. Змінювати зарплати співробітників.

Пасажир може виконувати такі дії:

1. Перевіряти наявність квитків.
2. Переглядати інформацію про поїздки.
3. Забронювати місця.
4. Скасовувати бронювання квитка.
5. Редагувати дані квитка.

Адміністратор має функціональні можливості всіх користувачів, окрім можливості додавати або видаляти інших адміністраторів.

Схематично можливості, які забезпечує дана програма зображено діаграмою прецедентів (рисунок 2.2).

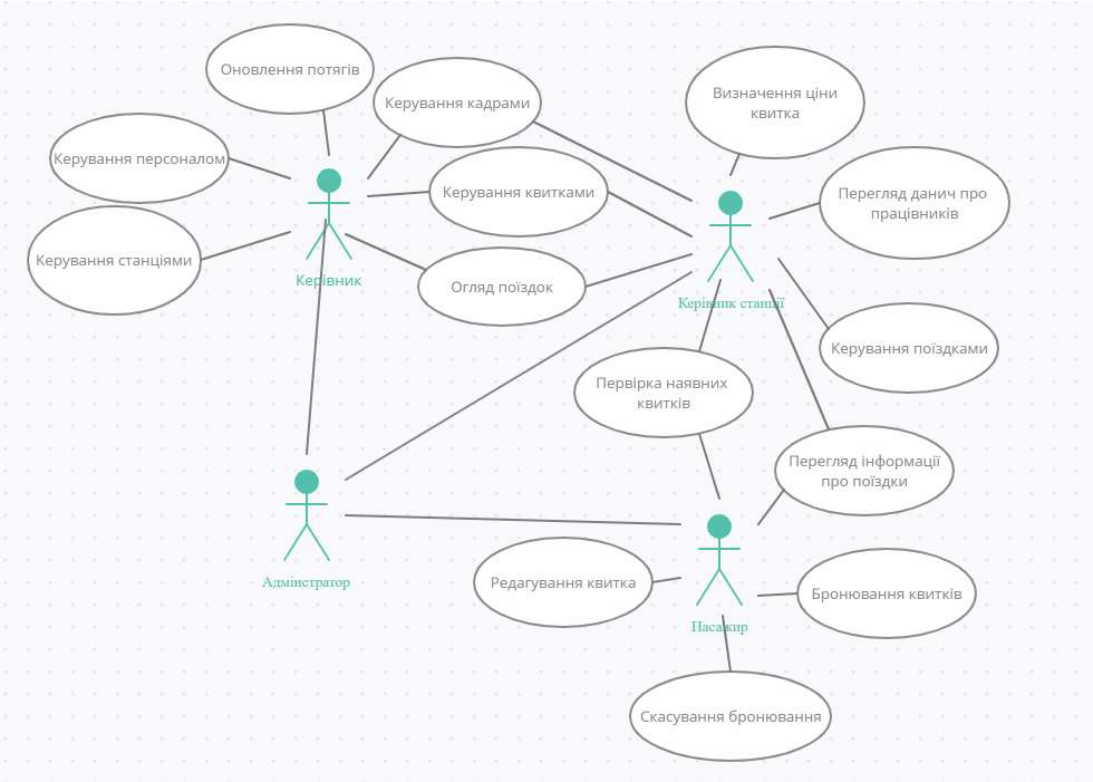

Рисунок 2.2 – Діаграма прецедентів програми

Одним з найважливіших завдань при розробці програмного забезпечення є вибір засобів програмування, які полегшили б роботу програміста, надавши всі необхідні інструменти для виконання завдання і дозволили б отримати результат, який повністю задовольняє користувача.

Засоби реалізації, які були використані при створенні програмного продукту (рисунок 2.3).

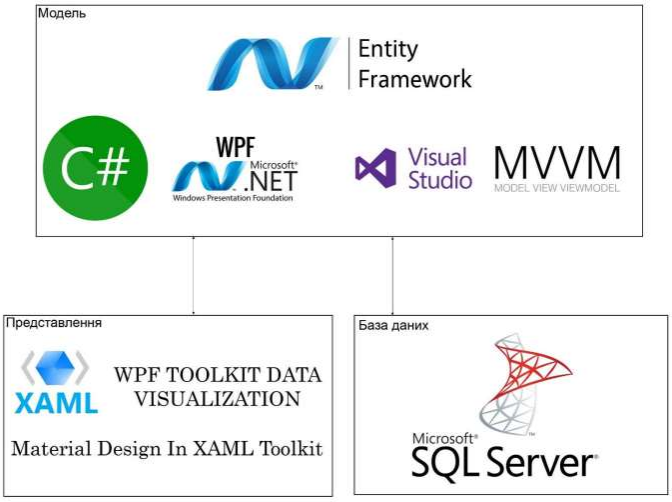

Рисунок 2.3 - Засоби реалізації програмного забезпечення

Для реалізації завдання продажу залізничних квитків було розроблено програмний додаток, який можна розгорнути на будь-якому комп'ютері під управлінням Windows.

При створенні програмного забезпечення використовувались такі інструменти реалізації:

1. Середовище розробки Visual Studio 2019.
2. Мова програмування C#.
3. Фреймворк .Net Framework.
4. Мова розмітки XAML.
5. Фреймворк ADO .Net Entity Framework.
6. Бібліотека WPF Material Design Data Visualization.
7. База даних Microsoft SQL Server 2019 та Microsoft SQL Server Management Studio 19.

Вибір технологій зумовлений тим, що ці інструменти забезпечують найбільшу функціональність серед конкурентів, саме тому були обрані технології - C #, XAML, бібліотеки Material Design в XAML Toolkit та WPF Toolkit Visualization Data. Для розробки бази даних було обрано Microsoft SQL Server 2019.

Розробка програмного забезпечення зумовлена тим, що більшість компаній, які займаються обліком та продажею залізничних квитків, використовують потужне обладнання та комп’ютери, а тому використання програм, спрямованих на високу продуктивність системи, є доцільним.

C # був обраний мовою програмування високого рівня, оскільки в ньому є безліч бібліотек, які допомагають спростити процес налаштування програми, а також стандартизувати архітектуру та стиль написання коду, що важливо для можливої розробки іншими розробниками.

У середовищі розробки Microsoft Visual Studio також використовується технологія IntelliSense, яка значно пришвидшує написання коду, покращує обробку коду та пропонує способи вирішення помилок розробки.

Базу даних було обрано Microsoft SQL Server 2019 насамперед через швидкість, оскільки коли накопичуються дані, вона відіграє важливу роль у продуктивності.

Висновки до розділу

У даному розділі подаються основні характеристики конкретних методів розв`язання задачі, способів подання знань та логічного виведення, програмні, системні та допоміжні засоби, які застосовуються в роботі для побудови додатку.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Засоби розробки

Вибір таких інструментів розробки зумовлений тим, що ці інструменти забезпечують найбільшу функціональність серед конкурентів, саме тому були обрані технології - C #, XAML, бібліотеки Material Design в XAML Toolkit та WPF Toolkit Visualization Data. Для розробки бази даних було обрано Microsoft SQL Server 2019.

Середовище розробки Microsoft Visual Studio - це лінійка програмних продуктів Microsoft, яка спеціалізується на розробці інтегрованого середовища розробки програмного забезпечення. Продукти компанії дозволяють розробляти консольні програми, графічні програми для користувачів, включаючи підтримку технології Windows Forms, веб-сайтів, веб-додатків та веб-сервісів для всіх платформ, що підтримуються компанією. Visual Studio включає редактор вихідного коду з вбудованою технологією IntelliSense, що значно спрощує процес написання коду та можливість швидкого рефакторингу коду, не витрачаючи часу. Вбудований налагоджувач працює і як налагоджувач рівня вихідного коду, і як налагоджувач рівня комп'ютерної машини. Інші інструменти включають редактор форм для поліпшення роботи програміста при розробці графічного інтерфейсу, веб-редактор для Інтернет-ресурсів, дизайнер класів та конструктор схем баз даних.

Середовище розробки Visual Studio надає можливість створювати та додавати сторонні програми до проектів, щоб покращити функціональність на всіх рівнях, включаючи можливість підтримувати контроль версій коду, додавати нові набори інструментів або інструменти для різних аспектів нового процесу проекту.

Середовище розробки Visual Studio допомагає писати код, незалежно від мови, від C #, VB та C ++ до JavaScript та Python, надаючи допомогу в режимі реального часу. IntelliSense описує API під час введення тексту, а автоматичне

заповнення збільшує швидкість і точність. Знайомство з новим API пришвидшується за рахунок звуження набору значень за категоріями. Підказка дозволяє перевірити визначення API. Проблемні ділянки виділені знаками тильди, які часто відображаються під час введення тексту.

Мова програмування C # - це об'єктно-орієнтована мова програмування із захищеною системою друку на платформі .NET. C # був розроблений Андерсом Галесбергом, Скоттом Вільтамутом та Пітером Голде під егідою Центру досліджень Microsoft.

Синтаксис C # може нагадувати синтаксис таких мов, як C ++ та Java. C # має суворе введення тексту, підтримує поліморфізм, перевантаження оператора, вказівники на функції члена класу, атрибути, події, властивості, винятки, коментарі у форматі XML. C # багато чому навчився у своїх попередників. У тому числі з мов програмування: C ++, Delphi, Module та Smalltalk. Мова C # виключає деякі переліки моделей, які, як було доведено, проблематично використовувати при розробці програмного забезпечення. Прикладом цього є відмова корпорації від багаторазового успадкування класів.

Мова C # була розроблена як мова програмування рівня додатків для CLR і тому значною мірою залежить від можливостей самого CLR. Наявність та відсутність деяких особливостей мови програмування зумовлена тим, чи можна обрану ознаку мови перекласти у відповідні конструкції CLR. CLR надає C #, як і всі інші мови, орієнтовані на .NET, багато функцій, яких не мають класичні мови програмування.

Для створення системної архітектури використовуються сучасні бібліотеки або фреймворки, які спрощують процес написання коду.

Програмне забезпечення - це набір готових до використання програмних рішень, що включає архітектуру проекту, логіку та основні функціональні можливості системи або підсистеми та навіть дизайн.

Структура включає набір бібліотек, які пропонують готовий набір рішень, а також можуть містити утиліти, сценарії та, як правило, все, що може

полегшити програмісту розробку та поєднання різних компонентів масового програмного забезпечення. Однією з головних переваг використання фреймворків є стандартизація структури додатків.

.Net Framework 4.6.1 - це міжплатформна технологія, запропонована корпорацією Майкрософт як платформа для створення загальних програм та веб-додатків. Багато в чому ця технологія є продовженням ідей та принципів, закладених під час розвитку технології Java.

Однією з основних ідей розвитку .NET є сумісність служб, написаних різними мовами програмування. Усі бібліотеки в .NET мають інформацію про версію. Це дозволяє позбутися можливих конфліктних ситуацій між різними версіями колекцій. Технологія .NET поділяється на дві основні частини - середовище виконання та засоби розробки.

Середовища розробки .NET включають використання таких програм, як Visual Studio .NET, SharpDevelop, Borland Developer Studio тощо. Програми також можна розробляти в текстовому редакторі та за допомогою консольного компілятора.

Однією з головних ідей Microsoft .NET є сумісність різних служб, написаних різними мовами.

Технологія ADO.NET Entity Framework - об’єктно-орієнтована технологія доступу до даних, є об’єктно-реляційним рішенням відображення для .NET Framework від Microsoft. Забезпечує можливість взаємодії з об’єктами як за допомогою LINQ як LINQ до сутностей, так і за допомогою сутності SQL. Для полегшення побудови веб-рішень використовуються як ADO.NET Data Services (Astoria), так і зв’язок з Windows Communication Foundation та Windows Presentation Foundation, що дозволяє створювати багаторівневі додатки, реалізуючи один із шаблонів дизайну MVC, MVP або MVVM.

Відмінною особливістю Entity Framework є використання запитів LINQ для отримання даних із бази даних. За допомогою інтегрованого мовного запиту ми можемо не лише отримувати певні рядки, що зберігають об’єкти з

бази даних, але й отримувати об’єкти, пов’язані з різними асоціативними посиланнями. Іншим ключовим поняттям є модель даних сутності. Ця модель порівнює суттєві класи з реальними таблицями в базі даних. Модель даних сутності складається з трьох рівнів: концептуального, рівня сховища та рівня відображення. На концептуальному рівні визначаються класи сутності, які будуть використовуватися в додатку. Рівень сховища визначає таблиці, стовпці, взаємозв'язки між таблицями та типи даних, з якими порівнюється використовувана база даних. Рівень відображення виступає посередником між попередніми двома рівнями, визначаючи відображення між властивостями класу сутності та стовпцями таблиць. Таким чином, можна взаємодіяти з таблицями з бази даних за допомогою класів, визначених у програмі.

Entity Framework пропонує три можливі способи взаємодії з базою даних:

1. Перший метод бази даних передбачає створення набору класів за допомогою Entity Framework, які представляють модель конкретної бази даних.

2. Спочатку модель Model передбачає створення моделі бази даних, після чого Entity Framework створить справжню базу даних на сервері.

3. Спочатку метод Code передбачає створення класів моделі даних, які будуть зберігатися в базі даних, а потім Entity Framework використовує цю модель для створення бази даних та її таблиць.

В архітектурі ADO.NET є посилання на фізичну структуру даних, тому під час написання коду для доступу до бази даних потрібно пам’ятати схеми таблиць та взаємозв’язків.

Основні класи можуть бути реструктуризовані відповідно до існуючих потреб, а середовище виконання Entity Framework відобразить ці унікальні імена до правильної схеми бази даних (рисунок 3.1).

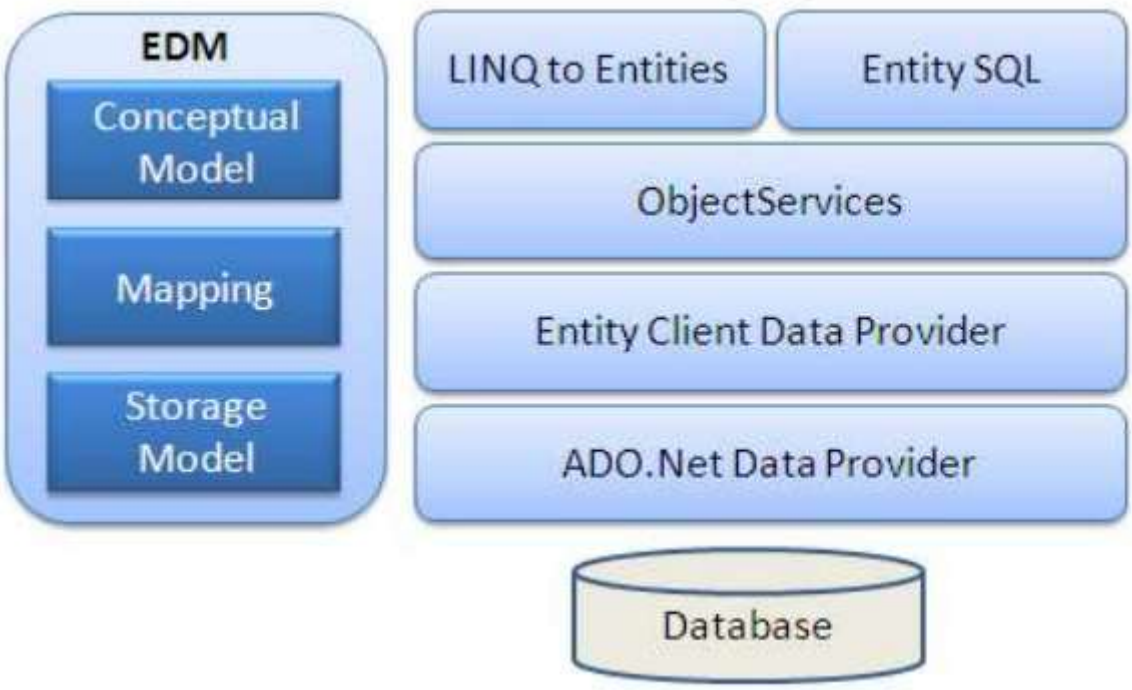

Рисунок 3.1 - Архітектура Entity Framework

Технологія ADO.NET призначена для того, щоб ізолювати програміста від вивчення структур баз даних різних виробників, представляючи постачальників баз даних, які інкапсулюють механізм роботи з певною базою даних. Це дозволяє створювати адаптери для будь-якої бази даних і повною мірою використовувати всі її можливості.

Windows Presentation Foundation - графічна підсистема Microsoft для розробки користувальницького інтерфейсу програмних додатків.

WPF базується на системі векторної візуалізації, яка не залежить від розширення вихідного пристрою. Він був створений з урахуванням можливостей сучасної графічної техніки. WPF надає інструменти для створення візуального інтерфейсу, включаючи XAML, елементи керування, прив'язку даних, макети, 2D та 3D графіку, анімацію, стилі, шаблони, документи, текст, мультимедіа та дизайн.

Графічна технологія DirectX - основна технологія WPF, на відміну від Windows Forms, яка використовує GDI / GDI +. Продуктивність WPF вища за

GDI + завдяки використанню апаратного графічного прискорення через DirectX.

Існує також усічена версія CLR під назвою WPF / E, також відома як Silverlight.

Для роботи з WPF потрібна будь-яка сумісна з .NET мова програмування. Цей список включає багато мов: C #, F #, VB.NET, C ++, Ruby, Python, Delphi, Lua та багато інших. Для повноцінної роботи можна використовувати Visual Studio та Expression Blend. Перший зосереджений на програмуванні, а другий - на дизайні і дозволяє робити багато речей без необхідності вручну редагувати XAML. Прикладами цього є анімація, стилізація, стан, створення елементів керування тощо.

WPF надає широкий спектр можливостей для створення інтерактивних настільних додатків. Прив'язка даних - це гнучкий механізм, який дозволяє через розширення розмітки XAML асоціювати різні дані від значень властивостей управління до загальнодоступних властивостей, що реалізують поля бази даних за допомогою Entity Framework. Прив'язка даних представлена класом Binding, який, у свою чергу, успадковується від MarkupExtension, що дозволяє використовувати прив'язки не тільки в коді, але і в розмітці.

На додаток до базового класу Binding, WPF реалізує ще кілька механізмів зв'язування:

1. Механізм MultiBinding дозволяє створювати кілька прив'язок, вказуючи кілька елементів.

2. Механізм TemplateBinding використовується в шаблонах для асоціювання властивостей елемента в шаблоні із властивістю елемента, до якого застосовано шаблон.

3. Механізм PriorityBinding класифікує список прив'язок і, відповідно до пріоритету, вибирає з них властивість, до якого буде застосовано прив'язку. Якщо прив'язка з найвищим пріоритетом успішно повертає значення, немає необхідності обробляти інші прив'язки у списку.

Основні переваги Windows Presentation Foundation:

1. Застосування WPF не залежить від розширення екрана. Під час створення програми у WPF ця програма не залежить від розширення екрана. Він автоматично використовує компоненти DirectX. Роздільна здатність WPF завжди однакова з будь-яким розширенням екрана.

2. Замість блокування елементів керування на місці за допомогою конкретних координат, WPF підтримує гнучкі потоки, розміщуючи елементи керування на основі їх вмісту. Результатом є користувальницький інтерфейс, який можна адаптувати для відображення дуже динамічного вмісту.

3. WPF підтримує відтворення будь-якого аудіо- чи відеофайлу, що підтримується Windows Media, що дозволяє одночасно відтворювати більше одного мультимедійного файлу. WPF надає інструменти для інтеграції відеоспостереження в іншу частину інтерфейсу користувача.

Технологія WPF розділена на два рівні: керований API та некерований API. Керований API містить код, який працює під час виконання .NET Common Language Runtime. Цей API описує основні функціональні можливості платформи WPF (рисунок 3.2).

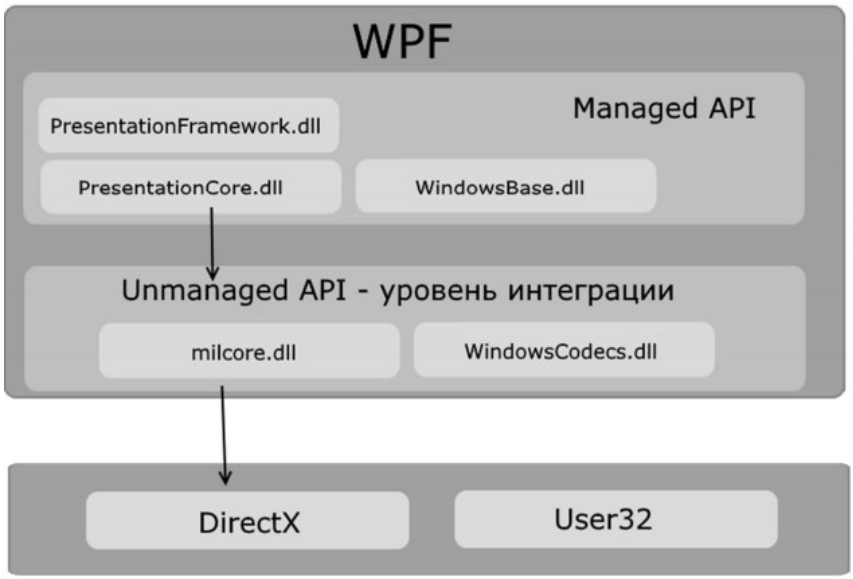

Рисунок 3.2 - Схематична архітектура WPF

Windows Presentation Foundation надає розробникам розширений користувальницький інтерфейс та графіку, недоступну у вікнах програми. Як і .NET Framework загалом, WPF - це технологія, орієнтована на Windows. Це означає, що програми WPF можна використовувати лише на комп’ютерах з операційною системою Windows.

XAML забезпечує робочий процес, який дозволяє декільком учасникам розробляти користувальницький інтерфейс та логіку програми, використовуючи потенційно різні інструменти.

Якщо вони представлені як код, файли XAML - це файли XML із розширенням .xaml. Файли можна зберігати в будь-якому кодуванні, яке підтримує XML, але зазвичай використовують кодування UTF-8.

Мова розмітки XAML широко використовується в .NET Framework 3.0, особливо в Windows Presentation Foundation (WPF), Windows Workflow Foundation (WWF) та Silverlight. У WPF XAML використовується як мова розмітки для інтерфейсу користувача для визначення елементів інтерфейсу користувача, прив'язки даних, підтримки подій та інших властивостей. У WWF ви можете використовувати XAML для визначення послідовностей дій.

Файли XAML можна створювати та редагувати за допомогою інструментів візуального дизайну, таких як: Microsoft Expression Blend, Microsoft Visual Studio, візуальний дизайнер WPF. Ви також можете створити їх за допомогою стандартного текстового редактора, редактора коду, такого як XAMLPad, або графічного редактора, такого як Vectropy.

Усе створене або реалізоване в XAML може відображатися за допомогою більш традиційних мов .NET, таких як C # або Visual Basic.NET. Однак ключовим аспектом технології є зменшення складності інструментів, що використовуються для обробки XAML, оскільки XAML базується на XML. Як результат, існує багато продуктів, які створюють додатки на основі XAML. Оскільки XAML базується на XML, розробники та дизайнери мають можливість одночасно працювати над вмістом без необхідності компіляції.

XAML відображає мовну функцію, тому клас може призначити лише одну зі своїх властивостей як властивість вмісту XAML. Дочірні елементи цього об'єкта використовуються для встановлення значення цієї властивості вмісту. Іншими словами, ви можете опустити елемент властивості для властивості вмісту, вказавши це властивість у розмітці XAML, і таким чином створити більш наочну метафору для батьківського або дочірнього елемента в розмітці.

SQL Server забезпечує можливість віддзеркалення та кластеризації баз даних. Кластер серверів SQL - це набір серверів, який було налаштовано таким же чином. Ця схема дозволяє розподілити навантаження між кількома серверами. Кожен сервер має одне віртуальне ім'я, і всі дані розподіляються на IP-адреси машин кластера протягом робочого циклу. У разі відмови або збою на одному із серверів кластера, навантаження буде автоматично перенесено на інший сервер.

Система SQL Server забезпечує можливість резервного копіювання даних у трьох можливих сценаріях розробки:

1. 1. Знімок робить "знімок" бази даних, який сервер потім надсилає користувачам.
2. 2. Історія змін призначена для постійної безпомилкової передачі всіх змін користувачам.
3. Синхронізація з іншими серверами означає, що бази даних кількох серверів будуть синхронізовані між собою. Зміни всіх баз даних відбуватимуться незалежно на кожному сервері. І вже в той момент, коли відбудеться синхронізація, система перевірить дані.

3.2 Опис реалізації системи

Для реалізації задачі продажу залізничних квитків був розроблений комп’ютерний додаток для операційної системи Windows (рисунок 3.3).

Програма розроблена з використанням мови програмування C # у .NET Framework відповідно до шаблону WPF.

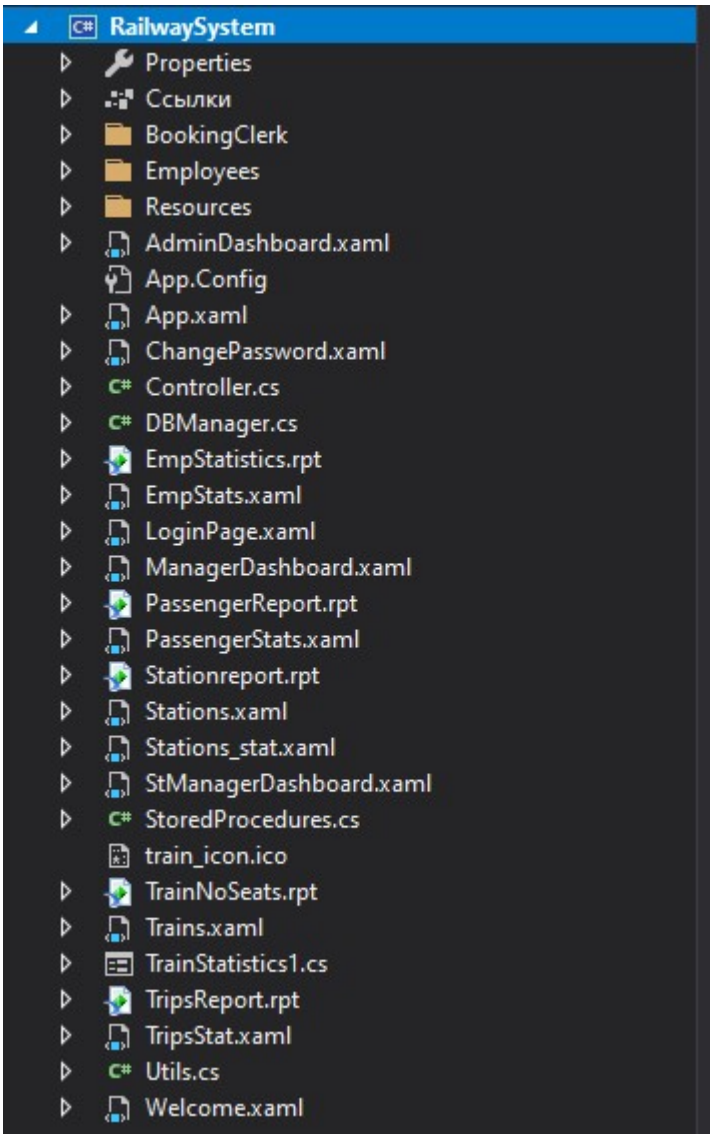

Рисунок 3.3 - Організація архітектури проекту

Модель проекту розділена на кілька класів, кожен з яких відповідає вікну, розробленому Windows Presentation Foundation, що є основним елементом технології.

Модель проекту розділена на кілька класів, кожен з яких відповідає вікну, розробленому Windows Presentation Foundation, яке є основним елементом технології.

3.3 Аналіз отриманих результатів

3.3.1 Контрольний приклад

Розглянемо клієнтську частину, для будь-якого користувача. Головна сторінка “Продаж залізничних квитків” (рисунок 3.4).

За допомогою даного вікна ми бачимо загальну структуру додатку.

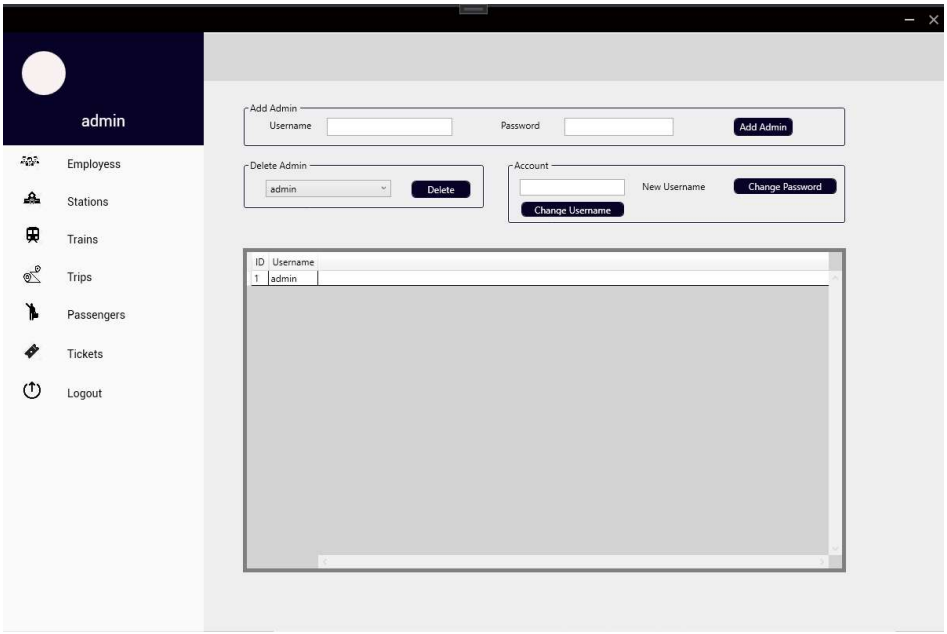

Рисунок 3.4 – Головне вікно додатку

Далі ми вікдриваємо вікно для перегляду працівників. В цьому вікні ми можемо додавати або видаляти працівників, змінювати їхню інформацію (рисунок 3.5).

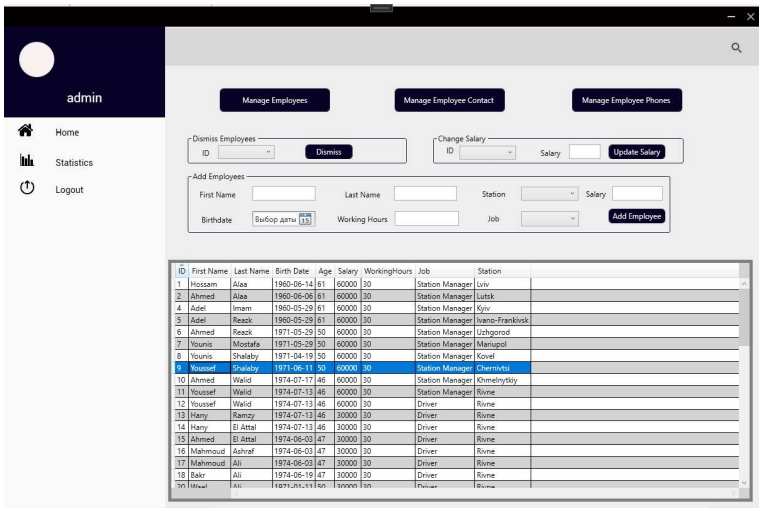

Рисунок 3.5 – Вікно працівники

Наступною вкладкою йдуть станції, тут ми можемо редагувати дані про станції, а також додавати або видаляти їх (рисунок 3.6).

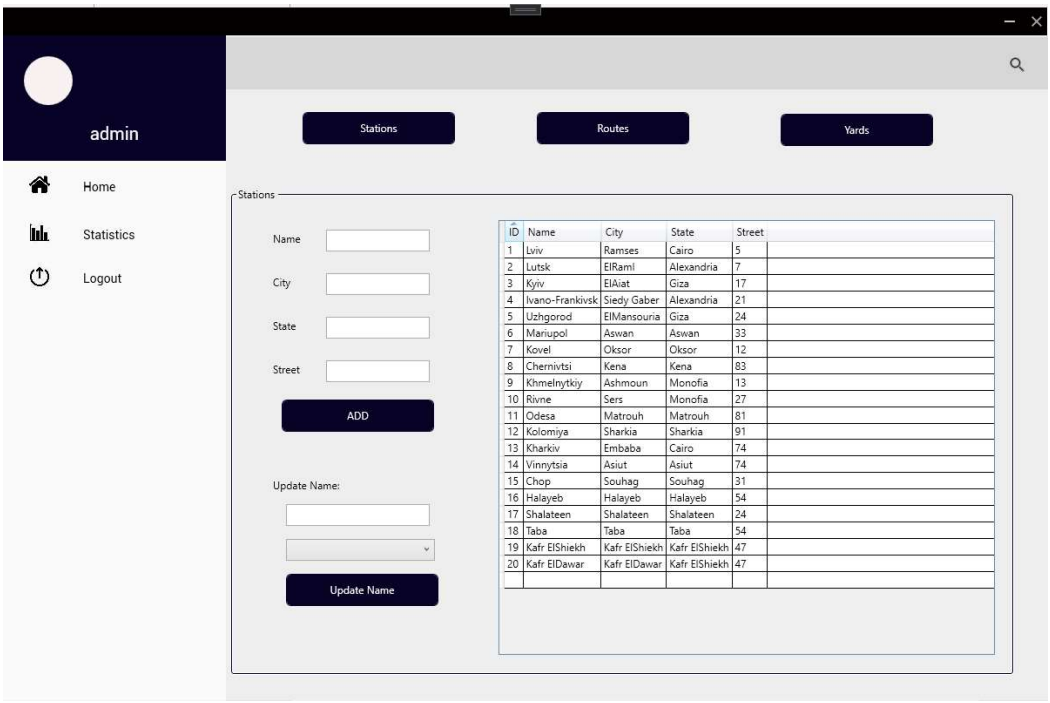

Рисунок 3.6 – Вкладка станцій

На вкладці потяги розміщена інформація про модель потяга, максимальну швидкість, колір, кількість місць. Тут адміністратор так само має змогу редагувати інформацію, додавати або видаляти потяги (рисунок 3.7).

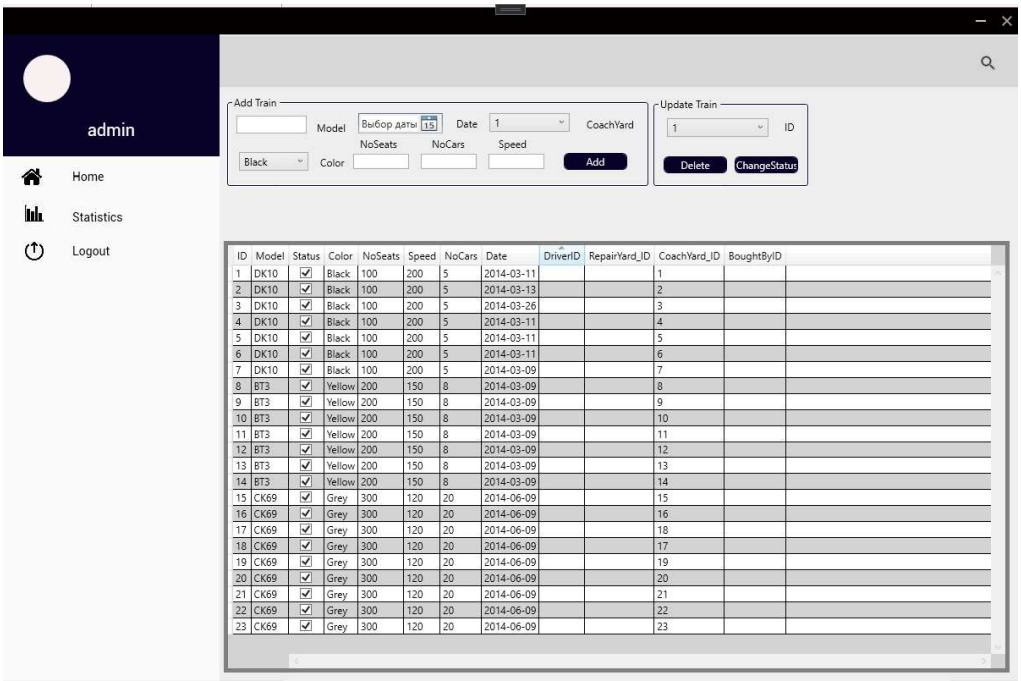

Рисунок 3.7 – Вікно потяги

Далі ми вікдриваємо вкладку маршрути, на який розміщені всі маршрути, які ми можемо редагувати (рисунок 3.8).

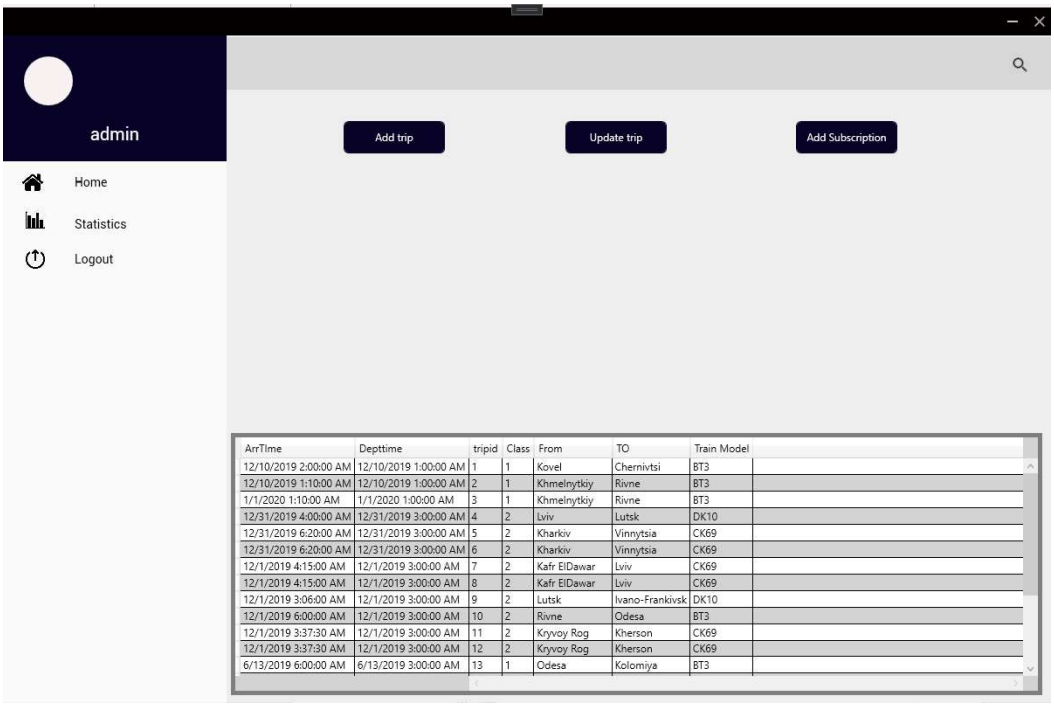

Рисунок 3.8 – Вкладка маршрути

Наступною йде вкладка пасажири. Тут ми можемо переглядати контактну інформацію про пасажирів, їхні квитки, а також редагувати їхні контактні дані (рисунок 3.9).

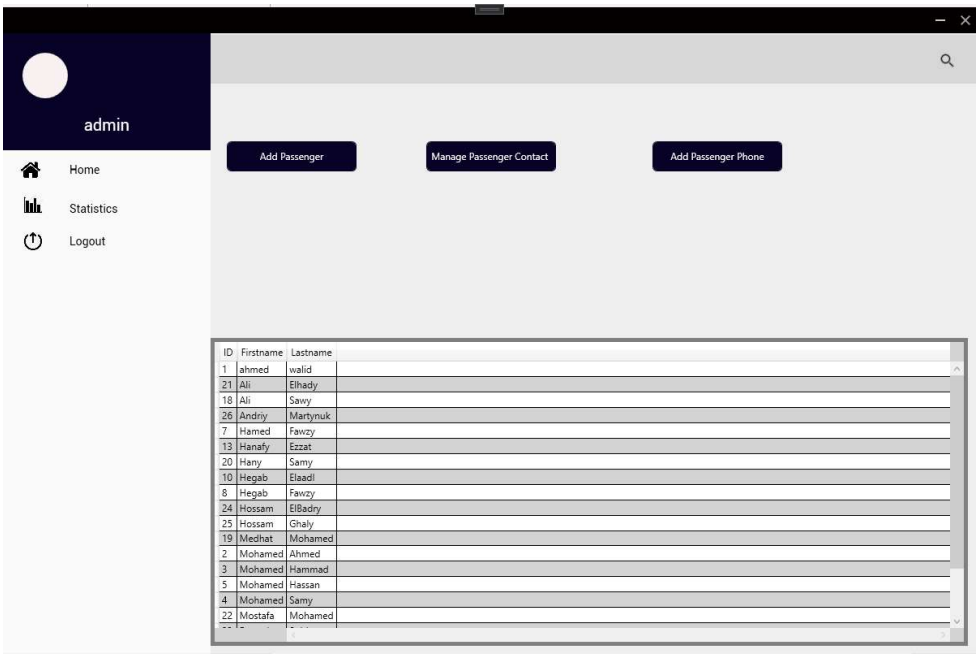

Рисунок 3.9 – Вікно пасажири

3.3.2 Аналіз критеріїв якості

Система вимагає встановлення програмного забезпечення на комп’ютері з ОС Windows.

Для коректної роботи з розробленою програмною системою користувачеві необхідні мінімальні потужності обладнання. Вимоги наведені (таблиця 3.1).

Таблиця 3.1 - Вимоги до апаратного забезпечення користувача

Пристрій	Характеристика
1	2
Процесор	Intel ® Core ™ 2 / 2 Duo / Pentium ® / Celeron ® / Xeon™ / i3 / i5 / i7 чи AMD 6 / Turion ™ / Athlon ™ / Duron ™ / Sempron ™ з тактовою частотою не нижче 1.5 GHz.
Оперативна пам’ять	Рекомендовано не менше 2Гб RAM
Апаратна архітектура	32-розрядна (x86) 64-розрядна (x64)

Висновки до розділу

У розділі «Засоби розробки» наводиться огляд, обґрунтування та вибір інструментарію для реалізації додатку. А також опис апаратних засобів, які необхідні для запуску проекту. Також є опис структури бази даних, структури, додатку.

Міститься контрольний приклад, який підтверджує працездатність розроблення та відповідає результатам функціонування системи

ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання даної бакалаврської дипломної роботи був розроблений програмний продукт для продажу залізничних квитків.

Створено програмний продукт, що має базу даних, в якій користувач може вносити зміни до існуючих налаштувань, автоматично обчислювати та зберігати їх.

Для реалізації програмного забезпечення було вирішено використовувати мову програмування C #, технологію Windows Presentation Foundation (WPF) для створення клієнтських додатків, версія .Net Framework. Інтерфейс програми був створений у форматі XAML. База даних була розроблена за допомогою Microsoft SQL Server та Microsoft SQL Server Management Studio.

Програмний модуль надає послуги, пов’язані із оглядом та замовленням квитків для осіб, які бажають придбати залізничні квитки.

Також, ця програма надає можливість адміністратору системи зберігати та редагувати інформацію про залізничні маршрути, потяги, які їх обслуговують, проміжні станції, розклад руху потягів до місця призначення, типи квитків та їх вартість.

Простота управління та інтуїтивно зрозумілий інтерфейс дозволяє без проблем користуватися додатком одразу після його відкриття.

Дана розробка певною мірою вирішить проблему придбання залізничних квитків. Та повинна автоматизувати роботу обслуговуючого персоналу залізниці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Адам Фримен - ASP.NET Core MVC с примерами на C# для профессионалов. 2017р.
2. Джеймс Чамберс - ASP.NET Core. Разработка приложений. 2018р.
3. Markus Egger - MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF.
4. Martin Fowler - GUI Architectures. Часть 1. 2009р.
5. Martin Fowler - GUI Architectures. Часть 2. 2009р.
6. Болье А. - Learning SQL. 2005р.

Рихтер Д. CLR via C# Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. – Киев: Питер, 2008. – 656 с.

7. Троелсен Э. Язык программирования C# и платформа .NET 4.5 / Эндрю Троелсен. – Киев: вильямс, 2014. – 1312 с.
8. Бодягін І. Model-View-Controller в .Net / Іван Бодягін. // RSDN Magazine. – 2016. – №2.
9. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э.Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Харьков: Питер, 2001. – 368 с.
10. Скіт Д. C# in Depth / Джон Скіт., 2013. – 582 с.
11. Дюбуа MySQL / Дюбуа, Поль. М.: Вильямс; Издание 2-е, 2012. 524 с.
12. Симдянов, И.В. MySQL 5 / И.В. Симдянов, М.В. Кузнецов. - М.: БХВ-П, 2014. 774 с.
13. Ульман MySQL / Ульман, Ларри. М.: ДМК Пресс, 2018. 352 с.
14. Буч Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон // Пер. с анл. – М.: ДМК Пресс, 2001. – 432 с.

ДОДАТКИ

Додаток А. Код програми

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace RailwaySystem
{
    /// <summary>
    /// Interaction logic for AdminDashboard.xaml
    /// </summary>
    ///
    public partial class AdminDashboard : Window
    {
        Controller ControllerObj;
        private int UserID;
        public AdminDashboard(int U)
        {
            InitializeComponent();
            UserID = U;

            ControllerObj = new Controller();

            AdminUsername.MaxLength = 20;
            AdminPassword.MaxLength = 8;
            NewUsernameTextbox.MaxLength = 20;

            BindAdmin();
        }

        private void BindAdmin()
        {
            DataTable dt = ControllerObj.GetAllAdmins();
            AdminsDataGrid.ItemsSource = dt.DefaultView;
            AdminsCombobox.ItemsSource = dt.DefaultView;
        }
        private void XClicked(object sender, RoutedEventArgs e)
        {
            Close();
        }

        private void MinimizeButton_Click(object sender, RoutedEventArgs e)
        {
            this.WindowState = WindowState.Minimized;
        }

        private void Window_Loaded_1(object sender, RoutedEventArgs e)
        {
            string username = ControllerObj.GetUsername(UserID);
            NameTextBox.Text = username;
        }
```

```
        private void Canvas_MouseDown_1(object sender, MouseButtonEventArgs
e)
        {
            this.DragMove();
        }
        private void LogoutTextButton_Click(object sender, RoutedEventArgs e)
        {
            this.Logout();
        }
        private void LogoutButton_Click(object sender, RoutedEventArgs e)
        {
            this.Logout();
        }
        private void Logout()
        {
            LoginPage LoginPage = new LoginPage();
            LoginPage.Show();
            this.Close();
        }

        private void GoTrains()
        {
            Trains TW = new Trains(UserID);
            TW.Show();
            this.Close();
        }

        private void EmployeesTextButton_Click(object sender, RoutedEventArgs
e)
        {
            this.GoEmployees();
        }
        private void StationsTextButton_Click(object sender, RoutedEventArgs
e)
        {
            this.GoStations();
        }

        private void TrainsTextButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoTrains();
        }

        private void TrainsButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoTrains();
        }

        private void StationsButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoStations();
        }

        private void GoStations()
        {
            Stations ST = new Stations(UserID);
            ST.Show();
            this.Close();
        }

        private void EmployeesButton_Click(object sender, RoutedEventArgs e)
        {
```

```
            this.GoEmployees();
        }

        private void GoEmployees()
        {
            Employees EM = new Employees(UserID);
            EM.Show();
            this.Close();
        }

        private void AddAdminButton_Click(object sender, RoutedEventArgs e)
        {
            this.AddAdmin();
        }

        private void AddAdmin()
        {
            if (AdminUsername.Text == "")
            {
                string msg = "Please Enter your username";
                string title = "Enter Username";
                MessageBox.Show(msg, title);
            }
            else if (AdminPassword.Password.ToString() == "")
            {
                string msg = "Please Enter your password";
                string title = "Enter Password";
                MessageBox.Show(msg, title);
            }
            else
            {
                int id = ControllerObj.AddAdmin(AdminUsername.Text,
AdminPassword.Password.ToString());
                if (id == 0)
                {
                    string msg = "This username Already Exists";
                    string title = "Enter Username";
                    MessageBox.Show(msg, title);
                }
                else
                {
                    BindAdmin();
                }
            }

        }

        private void AdminPassword_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Enter)
            {
                this.AddAdmin();
            }
        }

        private void AdminUsername_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Enter)
            {
                this.AddAdmin();
            }
        }
```

```
        private void DeleteAdminButton_Click(object sender, RoutedEventArgs
e)
        {
            if (AdminsCombobox.SelectedValue == null)
            {
                MessageBox.Show("Please Select Admin Name");
                return;
            }

            int id = Int32.Parse(AdminsCombobox.SelectedValue.ToString());
            var count = AdminsCombobox.Items.Count;
            if (count == 1)
            {
                MessageBox.Show("You Can't Delete All Admins", "Delete
Admin");
                return;
            }
            ControllerObj.DeleteUser(id);
            // Update
            BindAdmin();
            // Logout if your user is deleted
            if (id == UserID)
            {
                this.Logout();
            }
        }

        private void ChangePasswordButton_Click(object sender,
RoutedEventArgs e)
        {
            ChangePassword CP = new ChangePassword(UserID);
            CP.Show();
        }

        private void ChangeUsername()
        {
            if (NewUsernameTextbox.Text == "")
            {
                MessageBox.Show("Please Enter Username", "Enter Username");
                return;
            }
            string username = NewUsernameTextbox.Text;
            int ret = ControllerObj.ChangeUsername(UserID, username);
            if (ret == 0)
            {
                MessageBox.Show("This Username Already Exited", "Enter
Username");
            }
            else
            {
                NameTextBox.Text = username;
                BindAdmin();
            }
        }

        private void NewUsernameTextbox_KeyDown(object sender, KeyEventArgs
e)
        {
            if (e.Key == Key.Enter)
            {
                this.ChangeUsername();
            }
        }
```

```
        private void ChangeUsernameButton_Click(object sender,
RoutedEventArgs e)
        {
            this.ChangeUsername();
        }

        private void AdminsDataGrid_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {

        }

        private void TripButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoTrips();
        }

        private void TripTextButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoTrips();
        }

        private void GoTrips()
        {
            Trips trips = new Trips(UserID);
            trips.Show();
            this.Close();
        }

        private void PassengersTextButton_Click(object sender,
RoutedEventArgs e)
        {
            this.GoPassengers();
        }

        private void PassengerButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoPassengers();
        }

        private void GoPassengers()
        {
            Passenger P = new Passenger(UserID);
            P.Show();
            this.Close();
        }

        private void TicketTextButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoTicket();
        }

        private void TicketButton_Click(object sender, RoutedEventArgs e)
        {
            this.GoTicket();
        }

        private void GoTicket()
        {
            Ticket T = new Ticket(UserID);
            T.Show();
            this.Close()
```