ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка застосунку мовою C# для управління лізингом автотранспорту»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим ДОКУЧАЄВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Максим ДОКУЧАЄВ

Керівник: Віталій ЗАЛИВА

доктор філософії (PhD)

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Докучаєву Максиму Олеговичу

1. Тема кваліфікаційної роботи: «Розробка застосунку мовою C# для управління лізингом автотранспорту»
керівник кваліфікаційної роботи доктор філософії (PhD), старший викладач кафедри ІПЗ Віталій ЗАЛИВА
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: Аналіз основних аспектів предметної області, теоретичні відомості про лізинг автомобілей, опис алгоритму основних задач, опис роботи програми.

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

1. Огляд та аналіз існуючих методів та технологій управління лізингом автотранспорт.

2. Проектування застосунку для управління лізингом автотранспорту.

3. Програмна реалізація та опис функціонування застосунку для застосунку управління лізингом автотранспорту.

4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Блок схеми основних алгоритмів.
6. Діаграма класів.
7. Схема бази даних
8. Екранні форми.
9. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих рішень з лізингу автотранспорту.Дослідження їх переваг і недоліків.	14.03-26.03.2024	
4	Проектування застосунку для управління лізингом автотранспорту.	27.03-10.04.2024	
5	Програмна реалізація застосунку	11.04-17.04.2024	
6	Тестування застосунку	18.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Максим ДОКУЧАЄВ
(підпис)

Керівник кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 2 табл., 14 рис., 22 джерел.

Мета роботи – удосконалення управління процесами автомобільного лізингу та прокату автомобілів.

Об'єкт дослідження – процес управління автомобільним лізингом.

Предмет дослідження застосунок для управління автомобільним лізингом.

1. *Короткий зміст роботи*: У результаті виконання дипломної роботи було розроблено програмне забезпечення для управління автомобільним лізингом. Проаналізовано існуючі рішення для управління лізингом. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: можливість виведення даних про клієнтів, машини, технічні огляди, бронювання і прокат машин, можливість додавати в базу даних дані про клієнтів, машини, технічні огляди, бронювання і прокат машин, можливість редагувати дані, можливість видаляти дані з системи Проведено тестування додатку. В роботі використано бібліотеку C# для створення програмного застосунку, WPF для створення користувацького інтерфейсу, Microsoft SQL в якості бази даних.

Сферою використання застосунку є управління лізингом автотранспорту приватних компаній.

КЛЮЧОВІ СЛОВА: C#, WPF, Лізинг, SQL.

ЗМІСТ

ВСТУП

Управління автомобільним лізингом стає все більш актуальною задачею в умовах розвитку автомобільного ринку та зростання популярності лізингових послуг серед підприємств та приватних осіб. Ефективне управління автопарком та угодами лізингу вимагає високотехнологічних рішень, що робить розробку спеціалізованого програмного забезпечення необхідною.

Метою дипломної роботи є удосконалення управління процесами автомобільного лізингу та прокату автомобілів. Програма має надавати функціонал для зберігання, видалення, редагування та перегляду цих даних з метою оптимізації процесів управління автомобільним лізингом.

Об'єктом дослідження є процес управління автомобільним лізингом, включаючи зберігання та обробку даних про клієнтів, автомобілі, угоди лізингу, бронювання та техогляди машин.

Предметом дослідження є застосунок для управління автомобільним лізингом.

Для досягнення поставленої мети слід виконати наступні задачі:

1. Провести дослідження та аналіз існуючих систем управління лізингом автомобілів для визначення їх переваг та недоліків.
2. Дослідити доступні технічні засоби й обрати набір інструментів для створення застосунку
3. Створити систему управління лізингом автомобілів, враховуючи функціональні вимоги та структуру бази даних.
4. Розробити та реалізувати застосунок для управління лізингом автомобілів, використовуючи технології C#, WPF та MS SQL Server.
5. Провести тестування розробленого застосунку для перевірки його працездатності та відповідності вимогам.

Розроблене програмне забезпечення має наукову новизну у використанні сучасних технологій та підходів до управління автомобільним лізингом. Воно поєднує у собі зручний інтерфейс користувача з високою функціональністю та ефективністю у роботі з даними.

Розроблене програмне забезпечення стане ефективним інструментом для управління автомобільним лізингом, що дозволить підприємствам та приватним особам оптимізувати процеси управління автопарком та угодами лізингу.

Апробація результатів дослідження:

1. Докучаєв М.О. Залива В.В. Створення додатку для управління лізингом автотранспорту з використанням мови програмування c#. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 р., ДУІКТ, м.Київ. с.212
2. Докучаєв М.О. Залива В.В. Актуальність розробки додатків для управління лізингом автотранспорту. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 р., ДУІКТ, м.Київ. с. 119

1 АНАЛІЗ ОСНОВНИХ АСПЕКТІВ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Лізинг автомобілів

Лізинг транспортних засобів — це надання в оренду (або користування) транспортного засобу на певний строк за погоджену грошову суму в оренду. Його зазвичай пропонують дилери як альтернативу купівлі транспортних засобів, але широко використовують підприємства як спосіб придбання (або використання) транспортних засобів для бізнесу без зазвичай необхідних грошових витрат. Основна відмінність лізингу полягає в тому, що після закінчення основного терміну (зазвичай 2, 3 або 4 роки) автомобіль потрібно або повернути лізинговій компанії, або викупити за залишковою вартістю.

Лізинг автомобіля пропонує переваги як покупцям, так і продавцям. Для покупця лізингові платежі зазвичай будуть нижчими, ніж платежі по автокредиту. У більшості штатів будь-який податок із продажу сплачується лише з кожного місячного платежу, а не одразу з усієї ціни покупки, як у випадку продажу на виплат або позики. Деякі споживачі можуть віддати перевагу лізингу, оскільки він дає їм змогу просто повернути автомобіль і вибрати нову модель після закінчення терміну лізингу, дозволяючи споживачеві керувати новим транспортним засобом кожні кілька років, не турбуючись про негативний капітал після здачі автомобіля, на відміну від тих випадків, коли придбаний транспортний засіб можна обміняти лише через 2-3 роки володіння. Теоретично транспортний засіб, що перебуває в лізингу, завжди знаходиться на заводській гарантії, тому лізингоодержувачу не потрібно платити за ремонт. Лізингоодержувач не повинен хвилюватися про майбутню вартість транспортного засобу, а власник автомобіля. Майже всі договори оренди

включають фіксовану ціну покупки наприкінці оренди, тому, якщо вартість транспортного засобу перевищує прогнозовану вартість, орендар може купити його, але якщо він коштує менше, орендар може повернути його. Орендодавець має податкові переваги. Лізингоодержувач-споживач також сплачує менше податку з продажу протягом терміну дії лізингу, ніж придбання транспортного засобу.

Для продавця лізинг приносить прибуток від транспортного засобу, яким продавець (або виробнича корпорація чи її фінансова дочірня компанія) все ще володіє, і його можна буде повторно орендувати або продати через ремаркетинг транспортного засобу після закінчення початкового (або основного) договору оренди. Оскільки споживачі, як правило, користуються орендованим транспортним засобом протягом коротшого періоду часу, ніж той, який вони купують безпосередньо, лізинг може швидше залучати постійних клієнтів, що може вписуватися в різні аспекти бізнес-моделі дилера. Крім того, лізингоодержувачі більш лояльні до одного і того ж виробника автомобіля, ніж покупці.

У лютому 2014 року середній рівень проникнення нових легкових транспортних засобів на роздрібний ринок лізингу в Сполучених Штатах досяг історичного рекорду в 26,5%. Це означає відновлення після серйозного падіння під час фінансової кризи 2007–2008 років. Станом на 2016 рік лізинг становив близько 25 відсотків від загального обсягу продажів автомобілів і 31 відсоток роздрібних продажів у Сполучених Штатах.

Поширеність лізингу в Сполучених Штатах для GM, Ford і Chrysler піднялася близько до галузевої норми після досягнення низьких однозначних цифр у 2009 році, але все ще нижча, ніж у BMW і Mercedes-Benz.

Договори оренди зазвичай передбачають плату за дострокове розірвання та обмежують кількість миль, які орендар може проїхати (для легкових автомобілів загальною цифрою є 10 000 миль на рік, хоча сума може бути обумовлена

клієнтом і може складати від 5 000 до 25 000 миль на рік). . У разі перевищення дозволеного пробігу може стягуватися плата. Дилери, як правило, дозволяють орендареві домовитися про більший пробіг для більшої орендної плати. У договорах оренди зазвичай вказується допустимий рівень зносу транспортного засобу, і орендар може стягнути плату, якщо цей рівень зносу перевищено.[4] Договір оренди з обслуговуванням (широко відомий у Великій Британії як Contract Hire) може включати всі витрати на експлуатацію автомобіля, за винятком палива та страхування.

Фактичні лізингові платежі розраховуються дуже подібно до платежів за кредитом, але замість річної процентної ставки компанія використовує так званий грошовий фактор.

Після закінчення терміну лізингу лізингоодержувач повинен або повернути транспортний засіб власнику, або викупити його у власника. Кінцева ціна оренди зазвичай узгоджується під час підписання договору оренди.

Зазвичай лізингова компанія має мінімальну тривалість оренди, наприклад від 24 місяців до 60 місяців. Нещодавно новий погляд на лізинг полягає в тому, що ринок короткострокової оренди під назвою «flexi-lease» зріс. Флексі-лізинг – це коли особа може орендувати новий транспортний засіб на 3 місяці, а потім вибрати повернути автомобіль/фургон або продовжити оренду на інший період. Це майже те ж саме, що оренда фургона, але зазвичай фінансова або лізингова компанія обслуговує та несе остаточну відповідальність за автомобіль.

Деякі компанії, такі як AutoTT, пропонують так звану короткострокову оренду автомобіля в Європі. Цей продукт має на меті надати лізинг автомобіля без ПДВ для нерезидентів Європи, автомобіль зареєстрований на ім’я клієнта. Програмі виповнилося 50 років, і вона почалася з французьких виробників автомобілів Renault, Peugeot і Citroën. Пропозиція включає нову машину, повне страхування від ризику та цілодобову допомогу та багато в чому схожа на оренду

автомобіля. Після закінчення перебування «власник» автомобіля повертає машину, і більше не має чим платити.

1.2 Розробка програмного забезпечення

Розробка програмного забезпечення є складним та багатофазним процесом, що включає різні етапи, методології, інструменти та практики. Від початкового аналізу вимог до остаточного тестування і впровадження, кожен етап вимагає ретельного планування та виконання. Давайте розглянемо основні аспекти розробки програмного забезпечення.

Перший етап у розробці програмного забезпечення полягає у зборі та аналізі вимог. Це критичний крок, оскільки правильне розуміння вимог забезпечує основу для подальших етапів. Аналітики співпрацюють із замовниками та кінцевими користувачами для визначення функціональних і нефункціональних вимог до системи. Документація вимог включає специфікації, діаграми варіантів використання (use case), та інші артефакти, які допомагають зрозуміти, що саме потрібно від програмного забезпечення.

Проектування є наступним етапом після аналізу вимог і включає створення високорівневих і детальних технічних специфікацій. У цьому етапі використовуються різні моделі та діаграми, такі як UML діаграми (діаграми класів, послідовності, активності тощо), для візуалізації структури системи та її поведінки. Проектування включає архітектурне проектування, що визначає основну архітектуру системи, її компоненти та їх взаємодію. Проектування бази даних включає створення моделі даних, що включає схему бази даних, таблиці, зв'язки та ключі. Інтерфейс користувача (UI/UX) проектування розробляє інтерфейс користувача з урахуванням зручності використання та естетики.

Реалізація або програмування включає безпосереднє написання коду на основі проектних специфікацій. Програмісти використовують різні мови

програмування, фреймворки та інструменти для створення функціональних компонентів системи. Важливі аспекти реалізації включають кодування, контроль версій та документування коду. Кодування передбачає написання програмного коду відповідно до стандартів і найкращих практик. Контроль версій включає використання систем контролю версій (наприклад, Git) для відстеження змін у коді та управління командною роботою. Документування коду включає написання коментарів і документації, що пояснюють функціональність коду.

Тестування є критичним етапом, що забезпечує якість програмного забезпечення. Метою тестування є виявлення дефектів та забезпечення відповідності системи вимогам. Основні типи тестування включають юніт-тестування, інтеграційне тестування, системне тестування та приймальне тестування. Юніт-тестування перевіряє окремі модулі або компоненти на правильність функціонування. Інтеграційне тестування перевіряє взаємодію між компонентами системи. Системне тестування є комплексним тестуванням всієї системи на відповідність вимогам. Приймальне тестування здійснюється кінцевими користувачами або замовниками для підтвердження готовності до впровадження.

Впровадження включає доставку програмного забезпечення кінцевим користувачам та його інсталяцію у виробничому середовищі. Це може включати налаштування системи, що передбачає конфігурацію програмного забезпечення відповідно до вимог виробничого середовища, перенесення даних зі старих систем до нової системи, а також підготовку користувачів, включаючи проведення навчання для кінцевих користувачів та надання супровідної документації.

Після впровадження програмне забезпечення потребує постійного супроводу та підтримки. Це включає виправлення помилок, що передбачає розв'язання проблем та дефектів, що виникають у процесі використання,

оновлення та вдосконалення програмного забезпечення шляхом додавання нових функцій, поліпшення продуктивності та безпеки, а також підтримку користувачів, що включає надання технічної підтримки та консультацій кінцевим користувачам.

Документування є важливою частиною кожного етапу розробки програмного забезпечення. Це включає технічну документацію, що описує архітектуру системи, деталі реалізації та інструкції з використання. Хороша документація полегшує подальшу підтримку і розвиток системи, а також забезпечує прозорість та зрозумілість для всіх учасників проекту.

Розробка програмного забезпечення є багатофазним процесом, що включає аналіз вимог, проектування, реалізацію, тестування, впровадження та супровід. Кожен етап має свої особливості і вимоги, і від якості виконання кожного з них залежить успішність проекту в цілому. Використання сучасних методологій, таких як Agile або Waterfall, допомагає структурувати процес розробки і забезпечити високу якість кінцевого продукту.

1.3 Огляд існуючих рішень

LeasePlan:

Короткий опис: LeasePlan - одна з компаній управління автопарком і лізингу автомобілів у світі, пропонує широкий спектр послуг з управління лізингом автомобілів для корпоративних клієнтів. Заснована у 2019 році, компанія налічує понад 100 автомобілів у своєму автопарку і має офіси в декількох районах міста Київ. LeasePlan відомий своєю експертністю у впровадженні інноваційних технологій, що полегшують управління автопарком і підвищують ефективність користувачів. Застосунок був розроблений за вимогами компанії LeasePlan.

Переваги застосунку: Наявність ключових функцій додатку(Додавання/Редагування автомобілей в автопарку, можливість додавання інформації про технічні огляди авто, також бронювання автомобілей для клієнтів, оновлення данних клієнтів, а також оновлення данних автомобілей в парку)

Недоліки застосунку: Простий дизайн, відсутність звязку між різними офісами, відсутність привязки Бази Даних додатку до сайту компанії.

CarLeasePro:

Короткий опис: CarLeasePro - провідна міжнародна компанія з управління автопарком і лізингу автомобілів, що пропонує індивідуальні рішення для корпоративних та приватних клієнтів. Компанія має більше ніж 50 працівників та обслуговує понад 350 автомобілів у місті Київ. CarLeasePro відомий своєю широкою географічною присутністю та гнучкими умовами лізингу, що дозволяє клієнтам отримувати індивідуальні рішення, що відповідають їхнім потребам.

Переваги застосунку: Можливості додавання/редагування автомобілів, зв'язок Бази даних з сайтом компанії.

Недоліки застосунку: Відсутність інформації про технічний огляд авто, бронювання, оновлення даних клієнтів, автомобілів компанії.

Winner Leasing:

Короткий опис: Winner Leasing - компанія з управління автопарком і лізингу автомобілів, що надає послуги для корпоративних клієнтів у багатьох містах України. Входить до складу BNP Paribas Group, єдиної фінансової групи, яка займається фінансовим лізингом та управлінням автопарком. Winner Leasing

відомий своєю широкою мережею партнерів у більш ніж 3 містах України та індивідуальним підходом до кожного клієнта.

Переваги: Додавання/редагування інформації автомобілів, інформації бронювання автомобілів, оновлення даних клієнтів, автомобілів в автопарку,

Недоліки: Відсутність звітності про техогляд автомобілів, відсутність зв’язку Бази Даних з сайтом компанії.

1.4 Постановка задачі

Робота передбачає створення програми для управління даними про клієнтів, автомобілі, лізинги, бронювання та техогляди машин. Програма повинна надавати функціонал для зберігання, видалення, редагування та перегляду усієї цієї інформації.

Функціонал програми повинен включати в себе:

1. Модуль зберігання даних:
    - Збереження даних про клієнтів, включаючи особисті дані та контактну інформацію.
    - Збереження даних про автомобілі, такі як марка, модель, рік випуску, номер кузова та інші характеристики.
    - Збереження даних про умови лізингу, включаючи строк, вартість та умови угоди.
    - Збереження даних про бронювання автомобілів для клієнтів.
    - Збереження даних про техогляди автомобілів та їх результати.
2. Модуль видалення даних:
    - Видалення записів про клієнтів, автомобілі, лізинги, бронювання та техогляди за потреби.
3. Модуль редагування даних:
    - Редагування інформації про клієнтів, автомобілі, лізинги, бронювання та техогляди для оновлення даних.
4. Модуль перегляду даних:
    - Перегляд інформації про клієнтів, автомобілі, лізинги, бронювання та техогляди для контролю та аналізу.

Цілі програми:

- Забезпечити зручний та ефективний спосіб управління даними про клієнтів та автомобілі.
- Забезпечити можливість оперативного внесення та зміни інформації.
- Забезпечити безпеку та цілісність збережених даних.
- Забезпечити можливість виведення звітів та аналізу даних для прийняття управлінських рішень.

Ця програма допоможе ефективно керувати усіма аспектами лізингу автомобілів і забезпечить зручний інтерфейс для користувачів для виконання рутинних операцій управління даними.

2 ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ РОЗРОБКИ СИСТЕМИ КОНТРОЛЮ ЛІЗИНГУ АВТОМОБІЛІВ

2.1 Огляд мови програмування

C# і C++ — це обидві високорівневі мови програмування, що активно використовуються в різних галузях розробки програмного забезпечення. Хоча обидві мови мають деякі спільні риси, вони суттєво відрізняються в багатьох аспектах, включаючи синтаксис, парадигми програмування, управління пам'яттю та підтримку різних платформ. У цьому розділі буде проведено порівняння мов C# та C++ за кількома ключовими параметрами.

C# і C++ мають схожий синтаксис завдяки своєму походженню від мови C. Проте C# була розроблена з акцентом на простоту використання та зменшення складності програмування. Вона включає багато вдосконалень, що полегшують життя розробникам, таких як автоматичне управління пам'яттю за допомогою збирача сміття (Garbage Collector), підтримка властивостей (properties) і подій (events), а також інтегрована підтримка асинхронного програмування.

C++ більш складна в освоєнні, оскільки пропонує розробнику широкий спектр можливостей, включаючи детальний контроль за пам'яттю, використання вказівників, множинне успадкування та шаблони (templates). Це дозволяє створювати високопродуктивні програми, але водночас вимагає більш глибоких знань і уваги до деталей.

Однією з найбільших відмінностей між C# і C++ є управління пам'яттю. У C++ розробник несе відповідальність за виділення та звільнення пам'яті. Це дає великий контроль, але також підвищує ризик виникнення помилок, таких як витоки пам'яті та використання видаленої пам'яті.

C#, навпаки, використовує автоматичне управління пам'яттю через збирача сміття, який автоматично видаляє об'єкти, що більше не використовуються. Це значно зменшує ризик помилок, пов'язаних з пам'яттю, і спрощує процес розробки, але може вплинути на продуктивність у певних сценаріях.

C# була розроблена як мова для платформ .NET і підтримує сучасні парадигми програмування, такі як об'єктно-орієнтоване програмування (ООП), функціональне програмування та асинхронне програмування. Мова має вбудовану підтримку для LINQ (Language Integrated Query), що дозволяє писати запити до колекцій даних у зрозумілій і зручній формі.

C++ також підтримує ООП та інші парадигми, але вимагає більшого обсягу коду для реалізації схожих функціональностей. Використання шаблонів у C++ дозволяє реалізовувати узагальнене програмування (generic programming), що забезпечує високу гнучкість і продуктивність, але додає складності у написанні та підтримці коду.

C# тісно інтегрована з екосистемою .NET, що забезпечує відмінну підтримку різних платформ, включаючи Windows, macOS, Linux, а також мобільні платформи через Xamarin. .NET також пропонує велику кількість бібліотек і інструментів, що значно спрощують розробку, тестування та розгортання програм.

C++ є більш універсальною мовою і використовується для розробки програмного забезпечення на різних платформах, включаючи вбудовані системи, операційні системи, ігрові рушії та наукові додатки. Проте відсутність єдиної стандартної бібліотеки та необхідність у використанні сторонніх бібліотек можуть ускладнювати процес розробки.

C++ вважається однією з найшвидших мов програмування завдяки низькорівневому доступу до апаратного забезпечення та ефективному управлінню пам'яттю. Це робить її ідеальною для розробки високопродуктивних додатків, таких як ігри, графічні програми та системне програмне забезпечення.

C# менш продуктивна порівняно з C++ через використання віртуальної машини .NET та збирача сміття. Однак, завдяки постійним оптимізаціям у .NET, продуктивність C# значно покращилася за останні роки, і вона може бути достатньою для більшості бізнес-додатків та веб-розробки.

Обидві мови програмування мають свої переваги і недоліки, і вибір між ними залежить від конкретних вимог проекту та досвіду команди розробників. C++ надає більше можливостей для оптимізації та високої продуктивності, але вимагає глибших знань і більшої уваги до деталей.

C#, з іншого боку, пропонує простоту використання, безпеку управління пам'яттю та багату екосистему .NET, що робить її привабливим вибором для широкого спектру додатків.

2.2 Огляд середовища розробки

Visual Studio 2019 (VS 2019) є однією з найпопулярніших інтегрованих середовищ розробки (IDE) від компанії Microsoft. Вона призначена для розробки програмного забезпечення на різних мовах програмування та платформах. Завдяки широкому спектру інструментів та можливостей, VS 2019 стала вибором багатьох професійних розробників для створення складних додатків і сервісів. Visual Studio 2019 була випущена 2 квітня 2019 року і з того часу отримала декілька оновлень, що значно розширили її функціональні можливості. Вона підтримує розробку на мовах C#, C++, Python, JavaScript, TypeScript та багатьох інших. IDE надає інтеграцію з різними системами контролю версій, такими як Git, та підтримку розробки для платформ Windows, macOS, Linux, Android, iOS та веб-платформ.

Інтерфейс Visual Studio 2019 було значно вдосконалено порівняно з попередніми версіями. Він став більш інтуїтивним та швидким. Нова стартова сторінка дозволяє швидко приступити до роботи з існуючими проектами,

створити новий проект або клонувати репозиторій з Git. Крім того, теми та налаштування кольорових схем роблять роботу більш комфортною для очей розробника.

Visual Studio 2019 надає потужну інтеграцію з Git та Azure DevOps. Розробники можуть легко керувати гілками, зливати зміни та відстежувати історію комітів безпосередньо з інтерфейсу IDE. Це значно спрощує процес командної розробки та дозволяє зберігати контроль над версіями коду.

VS 2019 пропонує вдосконалені інструменти для налагодження, що включають покращену підтримку діагностики помилок, можливості відстеження виконання коду у реальному часі, та інструменти для аналізу продуктивності додатків. Інструменти тестування, такі як Live Unit Testing, дозволяють автоматично запускати тести під час написання коду, що сприяє швидкому виявленню та виправленню помилок.

Visual Studio 2019 підтримує широкий спектр мов програмування. Завдяки цьому, розробники можуть працювати з єдиною IDE для проектів, написаних на різних мовах. Підтримка C#, C++, Python, JavaScript, TypeScript, та інших мов робить VS 2019 універсальним інструментом для розробників з різними потребами.

Visual Studio 2019 надає глибоку інтеграцію з хмарними сервісами Microsoft Azure. Розробники можуть легко створювати, розгортати та керувати хмарними додатками безпосередньо з IDE. Це включає підтримку для розгортання контейнерів Docker, використання функцій Azure Functions та інтеграцію з базами даних Azure.

VS 2019 оптимізовано для забезпечення високої продуктивності. Вона швидко завантажується, забезпечує швидку компіляцію та зручний інтерфейс користувача, що дозволяє розробникам зосередитися на написанні коду, а не на очікуванні завершення операцій.

Visual Studio 2019 підтримує широкий спектр розширень, що дозволяє користувачам налаштовувати IDE відповідно до своїх потреб. Магазин розширень Visual Studio пропонує тисячі безкоштовних та платних розширень, що додають нові функціональні можливості, підтримку мов програмування, інструменти для підвищення продуктивності та багато іншого.

Інструменти для командної роботи в Visual Studio 2019 включають інтеграцію з Git, Azure DevOps та іншими сервісами. Це дозволяє розробникам ефективно співпрацювати, відстежувати зміни в коді, організовувати спринти та керувати проектами безпосередньо з IDE.

Visual Studio 2019 є потужною IDE, вона також вимагає значних ресурсів. Для комфортної роботи потрібен комп'ютер з сучасним процесором, великою кількістю оперативної пам'яті та швидким SSD. Це може бути обмеженням для розробників, які працюють на старіших або менш потужних пристроях.

Хоча існує безкоштовна версія Visual Studio Community, професійні та корпоративні версії вимагають оплати. Вартість ліцензії може бути високою для малих компаній або індивідуальних розробників, що може стати бар'єром для використання всіх можливостей IDE.

Visual Studio 2019 є потужним та універсальним середовищем розробки, що підходить для широкого спектра завдань і мов програмування. Вона пропонує сучасний інтерфейс, глибоку інтеграцію з хмарними сервісами, потужні інструменти для налагодження та тестування, а також підтримку командної роботи. Хоча вона має певні недоліки, такі як високі системні вимоги та вартість ліцензії, її переваги роблять Visual Studio 2019 вибором багатьох професійних розробників та компаній. Завдяки своїм можливостям та постійним оновленням, VS 2019 залишається однією з провідних IDE на ринку.

Завдяки вище описаним критеріям була вибрана середа розробки VS Studio 2019. Microsoft SQL Server представляє собою реляційну базу даних, створену компанією Microsoft, що служить для зберігання та відновлення інформації за

запитом інших програмних аплікацій. Це рішення ефективно працює як на локальних, так і на віддалених серверах, включно з підключенням через Інтернет. Microsoft пропонує багато різних версій SQL Server, щоб задовольнити потреби різної аудиторії, від малих додатків до великих інтернет-проектів із значним числом одночасних користувачів.

2.3 Вибір СКБД

Хронологія розвитку Microsoft SQL Server розпочалася з версії SQL Server 1.0 у 1989 році для OS/2, який був спільним проектом компаній Sybase, Ashton-Tate та Microsoft, і продовжується дотепер. Розвиток продукту пройшов кілька важливих етапів:

Перехід на Windows NT з версією SQL Server 4.2 у 1993 році.

Вихід з партнерства з Sybase і початок самостійної розробки версії SQL Server 6.0 у 1995 році.

Переписування коду з C на C++ у SQL Server 7.0, що вийшов у 1998 році.

Завершення переходу від коду Sybase до власного коду Microsoft у SQL Server 2005.

Додавання стовпчастого сховища в пам'яті, відомого як xVelocity, у SQL Server 2012, що відкрило нові можливості для аналітики в реальному часі.

Підтримка Linux у SQL Server 2017, що розширило доступність продукту.

SQL Server Express, безкоштовна версія, пропонує базовий механізм бази даних без обмежень на кількість баз даних або користувачів, але з обмеженнями щодо використання ресурсів (один процесор, 1 ГБ пам'яті, до 10 ГБ обсягу бази даних). SQL Server Express з інструментами включає SQL Server Management Studio Basic, а версія з розширеними послугами надає додаткові функції, такі як повнотекстовий пошук та служби звітування.

База даних є сукупністю структурованих таблиць, що містять рядки та стовпці для організації інформації. Microsoft SQL Server підтримує широкий спектр типів даних, включаючи базові типи, такі як цілі числа (Integer), числа з плаваючою точкою (Float), числа з фіксованою точністю (Decimal), символьні рядки (Char та Varchar для змінної довжини), бінарні дані для BLOB-об’єктів, та текст для зберігання великих обсягів тексту. SQL Server використовує методи округлення, такі як симетричне арифметичне округлення, для перетворення плаваючих значень у цілі числа, наприклад, SELECT Round(2.5, 0) поверне 3.

Крім стандартних типів даних, SQL Server дозволяє створювати власні складені типи даних (UDT) та надає доступ до статистики сервера через віртуальні таблиці та подання, відомі як Dynamic Management Views (DMV). Бази даних можуть включати не лише таблиці, а й інші об'єкти, такі як подання, збережені процедури, індекси, обмеження та журнали транзакцій, із максимальною кількістю 2^31 об'єктів. Бази даних можуть розподілятися на декілька файлів різних файлових систем із максимальним розміром файла у 260 байт (1 екзабайт), з первинними файлами даних (.mdf), вторинними файлами даних (.ndf) для розподілу даних бази по кільком файлам, та файлами журналів (.ldf) для журналів транзакцій.

Дані у SQL Server організовані у сторінки розміром 8 КБ, які є базовими одиницями для читання та запису даних. Кожна сторінка містить 96-байтний заголовок з метаданими і може належати до різних типів, включно з даними таблиці, індексами, та іншими спеціалізованими типами сторінок. Дані керуються на рівні екстентів, кожен з яких складається з 8 сторінок, і можуть бути використані як окремо (однорідні екстенти), так і спільно з іншими об'єктами (змішані екстенти). Розмір рядка обмежений 8 КБ, але для даних, що перевищують цей обсяг, наприклад у стовпцях varchar або varbinary, використовуються додаткові сторінки, що підтримуються вказівниками, для зберігання великих об'ємів даних.

Для організації фізичного зберігання таблиць в Microsoft SQL Server, рядки розподіляються між декількома розділами, які мають індивідуальні номери від 1 до n, де розмір кожного розділу може бути налаштованим користувачем. За умовчанням, усі рядки знаходяться у єдиному розділі, але для розподілу даних між кластерами комп'ютерів таблицю можна розділити на кілька розділів. Залежно від того, чи є у таблиці кластерний індекс, рядки можуть бути організовані в структурі B-дерева або в купі. Таблиці з кластерним індексом зберігають рядки у відсортованому порядку згідно з ключами індексу в B-дереві, де дані розміщені в листкових вузлах, а інші вузли містять вказівники на ці дані. У таблиць без кластерного індексу дані зберігаються у невпорядкованій купі, але можуть бути некластеризовані індекси для поліпшення швидкості доступу до даних. І купи, і B-дерева можуть займати кілька одиниць розподілу для зберігання даних.

Доступ до даних у SQL Server здійснюється через запити, формульовані на мові T-SQL, що є специфічним діалектом SQL для Microsoft, що ділить спільні риси з Sybase SQL Server завдяки спільній історії. Запити задають, які дані потрібно вибрати, в декларативній формі. Процесор запитів аналізує запит та розробляє план його виконання, який визначає найефективніший спосіб отримання запитуваних даних. Можливо кілька варіантів виконання для одного запиту, залежно від використання операторів join і select, тому SQL Server автоматично вибирає оптимальний план для мінімізації часу виконання. Цей процес, відомий як оптимізація запитів, є ключовим для підтримки високої продуктивності в роботі з базами даних.

SQL Server містить оптимізатор запитів, який оцінює витрати на виконання запитів, з метою мінімізації ресурсів, необхідних для їх обробки. Враховуючи певний запит, оптимізатор аналізує структуру бази даних, статистику та поточне навантаження на систему, щоб визначити найефективніший спосіб доступу до даних. Він обирає оптимальну послідовність доступу до таблиць, метод

виконання операцій та спосіб доступу до даних, наприклад, вирішуючи, використовувати індекс або ні, залежно від його ефективності для даного запиту. Оптимізатор також визначає, чи слід виконувати запит паралельно, що може збільшити швидкість обробки за рахунок більшого використання ресурсів. Сформований план запиту тимчасово зберігається в кеші, щоб використовувати його для подальших однакових запитів, поки він не буде видалений через неактивність.

SQL Server дозволяє створювати збережені процедури, які є наборами параметризованих запитів T-SQL, збереженими на сервері. Вони приймають вхідні дані від користувачів та можуть повертати результати як вихідні параметри, дозволяючи викликати вбудовані функції та інші збережені процедури, включаючи можливість рекурсивного самовиклику. Збережені процедури забезпечують безпеку доступу, зменшення мережевого трафіку та підвищення продуктивності завдяки зменшенню кількості переданих даних і кешуванню планів виконання.

T-SQL, або Transact-SQL, є розширенням SQL, запатентованим Microsoft для SQL Server, що додає процедурні можливості, включаючи інструкції для управління транзакціями, обробки помилок, розширення стандартних SQL команд для маніпулювання даними (DML) та визначення даних (DDL). Це розширення сприяє більш гнучкому управлінню даними та налаштуванню конфігурацій SQL Server.

SQL Server пропонує оптимізатор запитів, орієнтований на мінімізацію витрат, який оцінює ресурси, необхідні для обробки запитів. Він аналізує структуру бази, статистику та актуальне навантаження, вибираючи найкращий шлях доступу до даних і методи виконання операцій. Вибір між використанням індексів залежить від їх ефективності, як наприклад, коли індекси на стовпцях із низькою вибірковістю можуть бути невигідні. Оптимізатор також вирішує, чи виконувати запит паралельно для прискорення обробки, незважаючи на

збільшений загальний час процесора. Створені плани запитів зберігаються у кеші для використання у майбутніх запитах, оптимізуючи продуктивність.

SQL Server уможливлює створення збережених процедур, параметризованих запитів, що виконуються на сервері, знижуючи мережевий трафік і покращуючи продуктивність. Ці процедури можуть використовувати вхідні та вихідні параметри, викликати інші процедури або навіть себе, та мають прив'язку до імен, що спрощує їх виклик. Плани виконання для збережених процедур також кешуються, подальше підвищуючи ефективність.

T-SQL, розширення SQL розроблене Microsoft для SQL Server, включає додаткові процедурні можливості для роботи з даними та управління сервером. Це мова включає ключові слова для створення та модифікації схем баз даних, маніпулювання даними, моніторингу та адміністрування сервера. Зокрема, можливість створення пов'язаних серверів через T-SQL дозволяє обробляти запити через кілька серверів одночасно.

3 ПРОЕКТУВАННЯ І РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ЛІЗИНГУ АВТОМОБІЛІВ

3.1 Опис алгоритму основних задач

Основні задачі системи можна представити чотирма окремими категоріями, для кожної з якої побудувати загальну блок-схему:

1. Виведення даних в таблицю
2. Додавання даних в базу даних
3. Видалення даних з бази даних
4. Оновлення бази даних

На рисунку 3.1 представлено блок-схему виведення даних в таблицю.

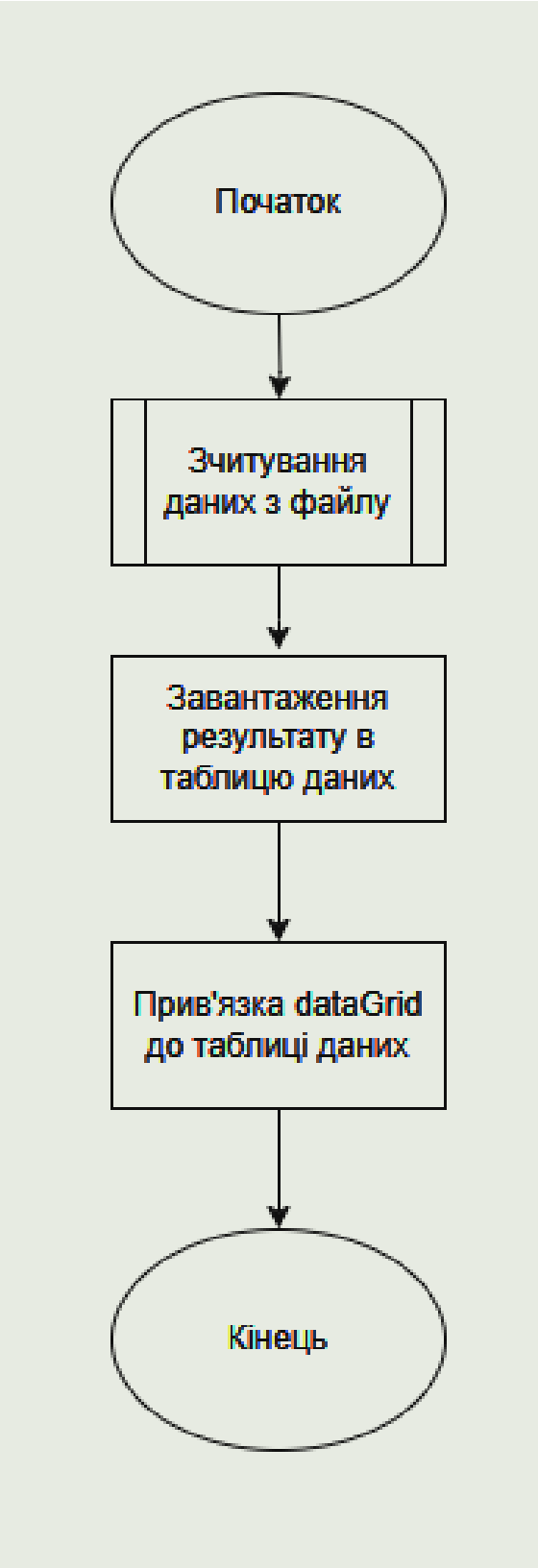


Рис. 3.1 Блок-схема виведення даних в таблицю

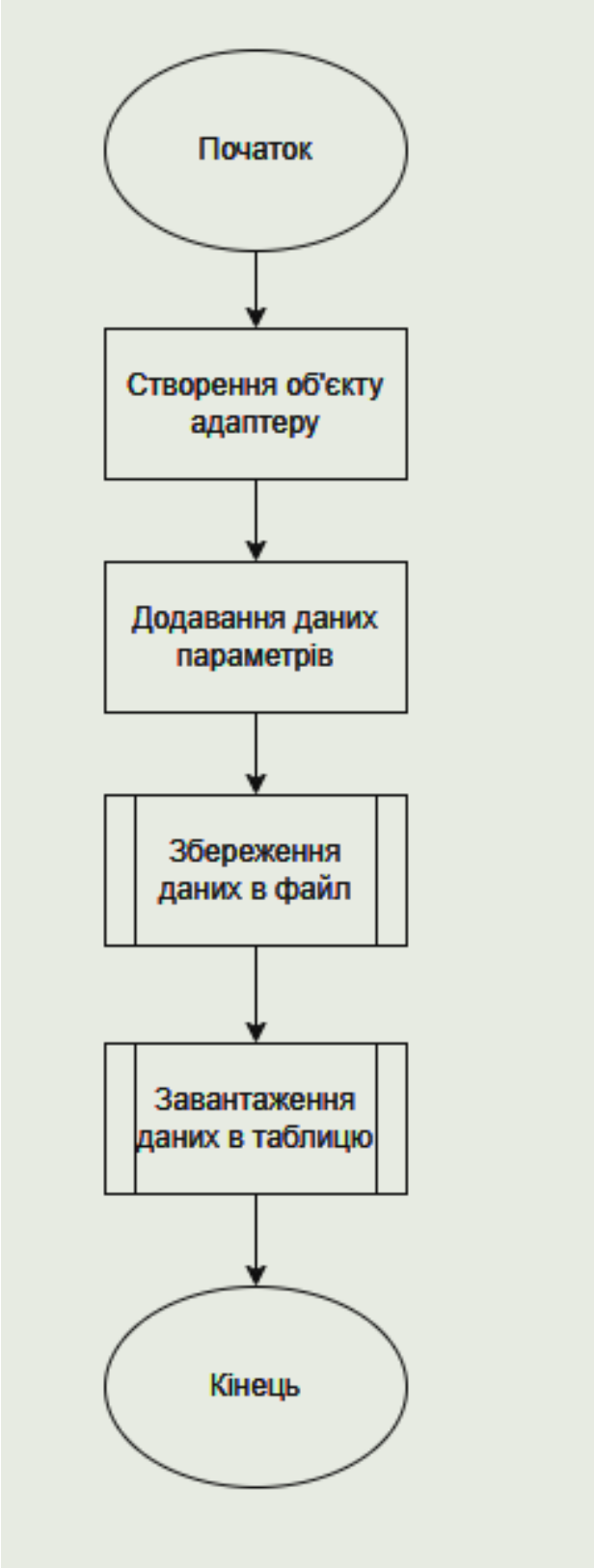


Рис. 3.2 Блок-схема додавання даних в БД

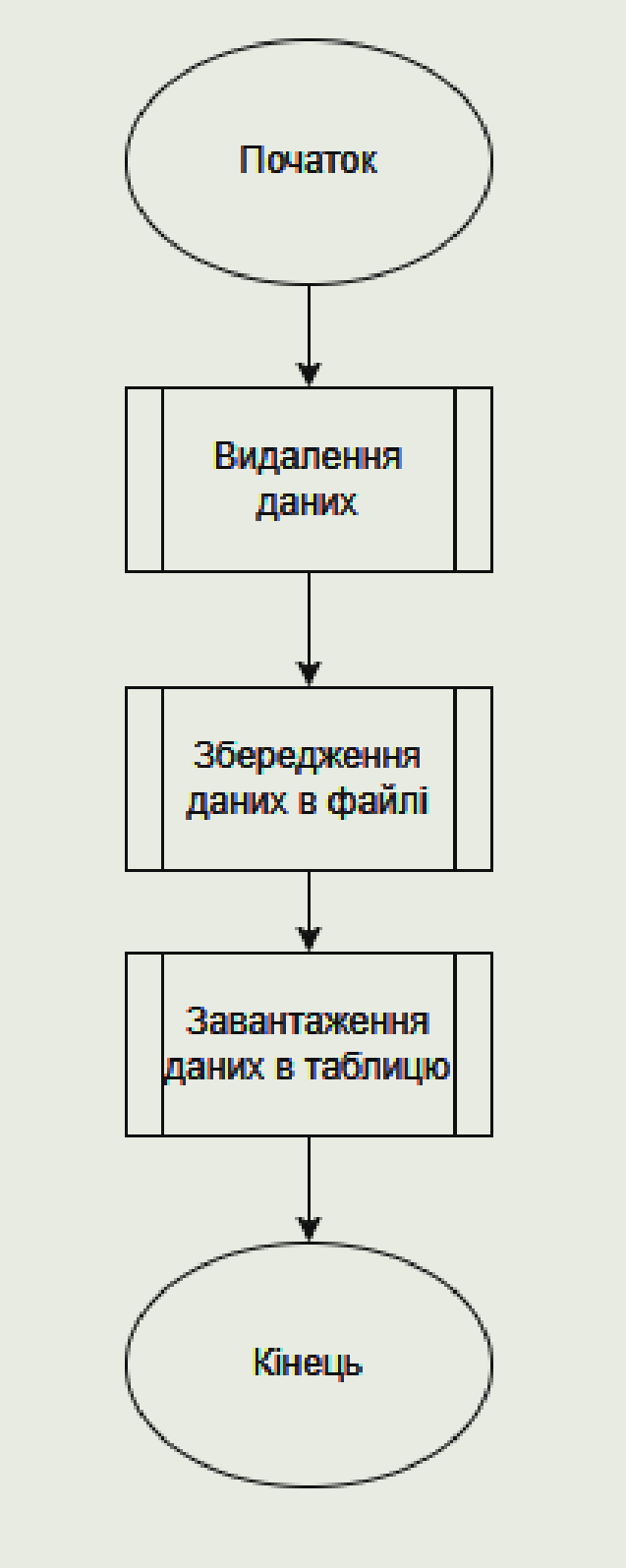


Рис. 3.3 Блок-схема видалення даних

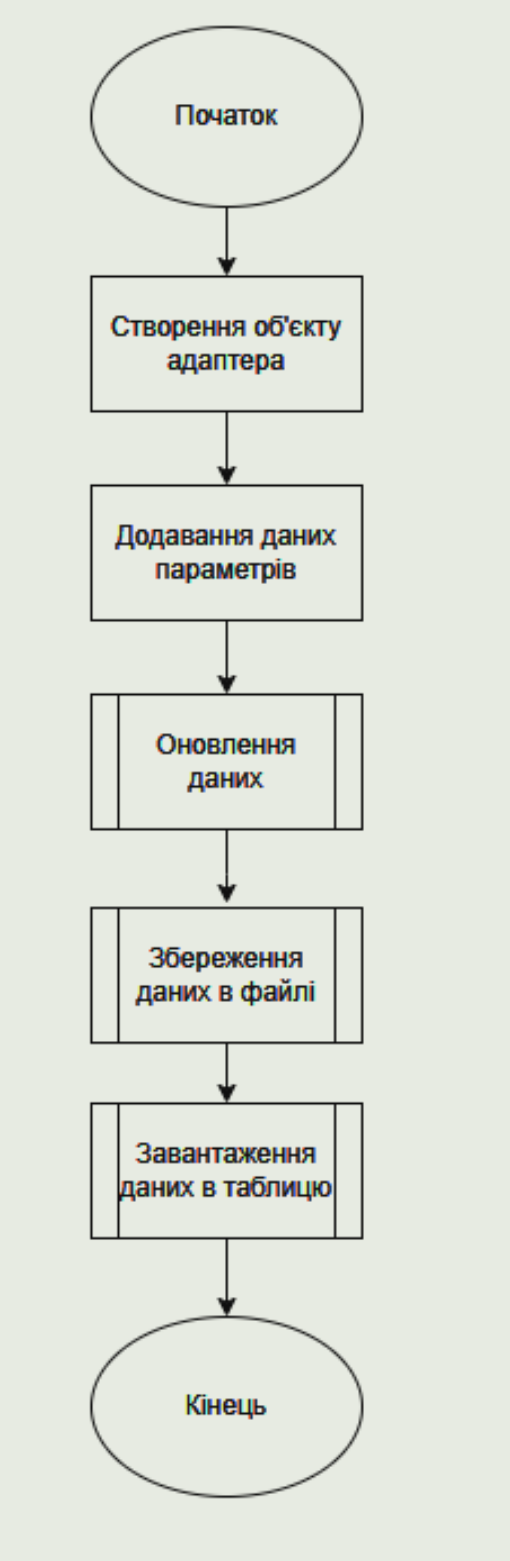


Рис. 3.4 Блок-схема оновлення даних

3.2 Проектування функціоналу

При проектуванні функціонального навантаження системи необхідно ретельно дослідити варіанти використання, що забезпечить глибоке розуміння призначення та можливої функціональної діяльності інформаційної системи в майбутньому.

Найпростішою формою візуалізації взаємодії між користувачем та системою є діаграма використання, яка дозволяє ілюструвати взаємодію користувача з різними сценаріями використання системи. Ці діаграми можуть ідентифікувати різні типи користувачів системи та різні моделі використання. Вони зазвичай поєднуються з іншими видами діаграм для отримання повної картини.

Хоча кожну можливість можна детально розглядати, використання діаграм допомагає надати загальний огляд системи на вищому рівні. Ці діаграми спрощують комунікацію зі зацікавленими сторонами і допомагають їм краще зрозуміти, як система буде розроблятися. Якщо говорити відомими словами, "план використання - це суть вашої системи."

Завдяки їх простоті, плани використання можуть стати ефективним інструментом комунікації для зацікавлених сторін. Ці зображення намагаються відтворити реальний світ та допомагають зацікавленим сторонам зрозуміти, як система буде розроблятися. Дослідження показало, що використання діаграм передає наміри системи зацікавленим сторонам більш зрозуміло, ніж діаграми класів.

Основною метою використання діаграм є відображення динамічних аспектів системи. Для більш повного функціонального і технічного опису системи можуть використовуватися інші схеми та документи. Вони надають спрощене графічне представлення того, як система має функціонувати.

• Межа системи представляє собою прямокутник з ім'ям та еліпсом (прецедент), але може опускатися в випадках відсутності корисної інформації.

• Актор - стилізована роль особи, яка відображає набір користувачів, які взаємодіють з системою або іншими сутностями. Актори не пов'язані один з одним.

• Прецедент - еліпс з підписом, що вказує на системну операцію, яка виконується інтерфейсом користувача та призводить до певних результатів. Назва може бути описом того, "що" в системі виконується, а не "як". Прецеденти представляють різні сценарії використання системи.

На рисунку 3.5 представлено діаграму прецедентів, яка ілюструє поведінку розроблюваної системи.

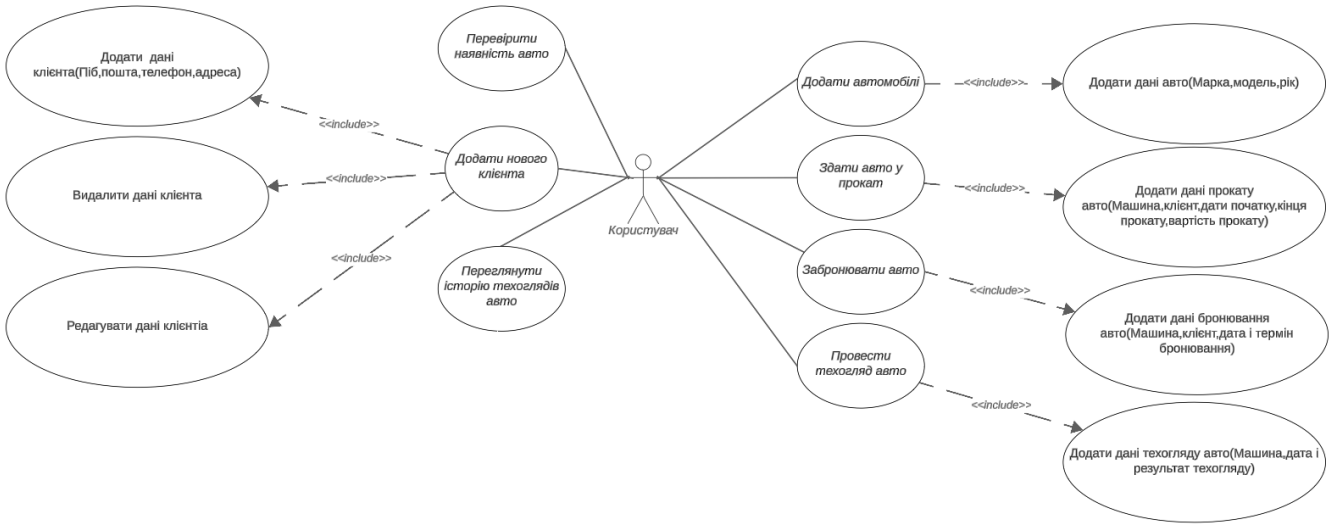


Рис. 3.5 Діаграма варіантів використання

3.3 Проектування бази даних

Проектування бази даних є одним з найважливіших етапів проектування програмного забезпечення.

На рисунку 3.6 представлено схему бази даних.

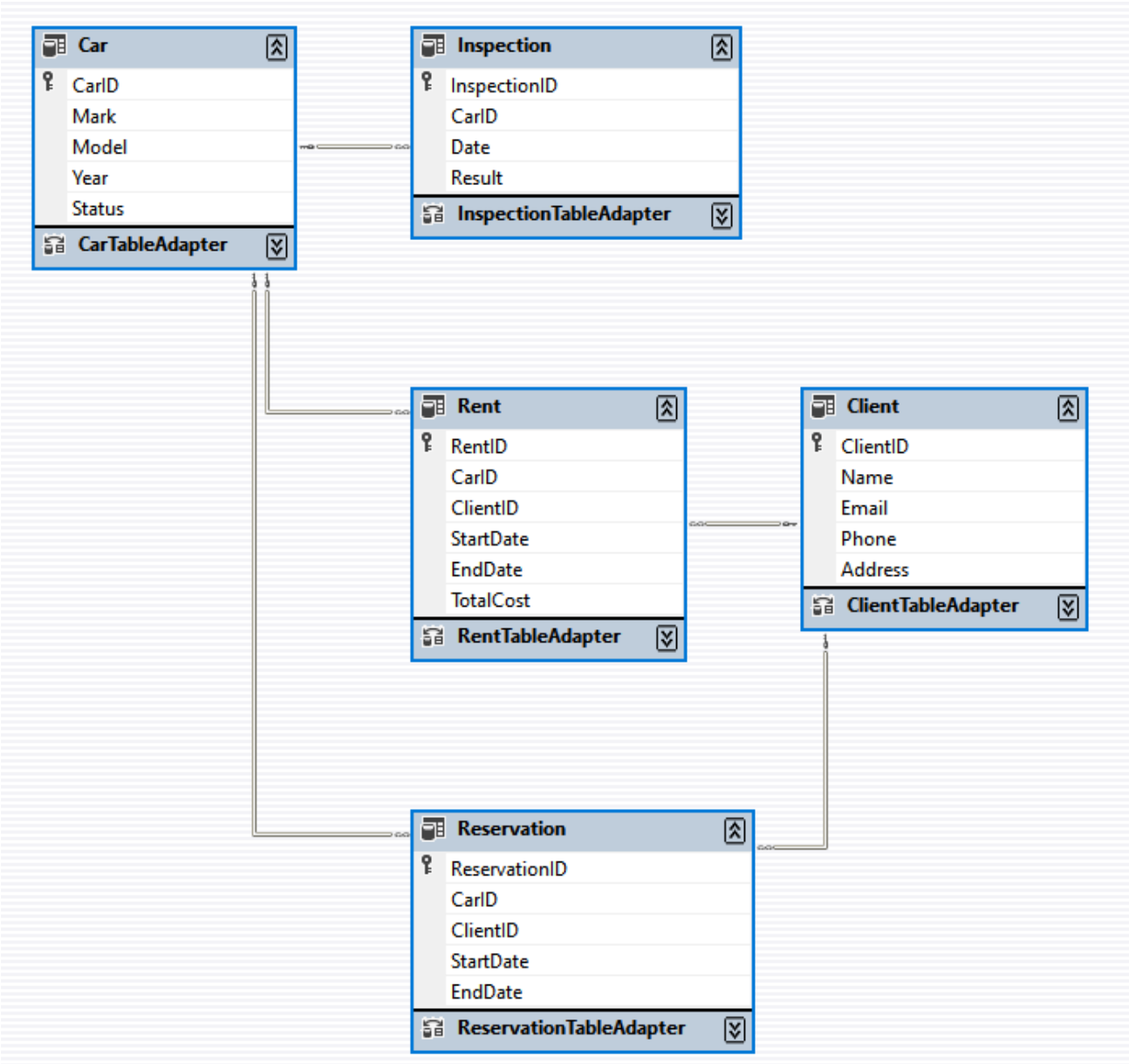


Рис. 3.6 Схема бази даних

База даних включає наступні таблиці:

1. Таблиця "Car":
    - CarID: ідентифікатор автомобіля.
    - Mark: марка автомобіля.
    - Model: модель автомобіля.
    - Year: рік випуску автомобіля.
    - Status: статус автомобіля (наприклад, доступний, на обслуговуванні тощо).
2. Таблиця "Inspection":
    - InspectionID: ідентифікатор огляду.
    - CarID: ідентифікатор автомобіля, який проходить огляд.
    - Date: дата огляду.
    - Result: результат огляду.
3. Таблиця "Rent":
    - RentID: ідентифікатор прокату.
    - CarID: ідентифікатор автомобіля, який беруть в прокат.
    - ClientID: ідентифікатор клієнта, який бере автомобіль в прокат.
    - StartDate: дата початку прокату.
    - EndDate: дата завершення прокату.
    - TotalCost: загальна вартість прокату.
4. Таблиця "Client":
    - ClientID: ідентифікатор клієнта.
    - Name: ім'я клієнта.
    - Email: електронна адреса клієнта.
    - Phone: телефонний номер клієнта.
    - Address: адреса клієнта.
5. Таблиця "Reservation":
    - ReservationID: ідентифікатор бронювання.

- CarID: ідентифікатор автомобіля, який бронюють.
- ClientID: ідентифікатор клієнта, який робить бронювання.
- StartDate: дата початку бронювання.
- EndDate: дата завершення бронювання.

Ці таблиці пов'язані між собою зв'язками, що відображають взаємозв'язки між автомобілями, клієнтами, оглядами, прокатами та бронюваннями.

3.4 Проектування внутрішньої будови

Проектування внутрішньої будови системи є однією з ключових стадій розробки програмного забезпечення. Цей процес включає створення різноманітних діаграм і моделей, які допомагають зрозуміти, як саме функціонує система, як взаємодіють її компоненти, і як вона буде реалізована на технічному рівні. Основними інструментами для цього є діаграми класів, UML (Unified Modeling Language) діаграми, а також діаграми модулів.

Діаграма класів є основним компонентом об'єктно-орієнтованого проектування, що відображає структуру системи шляхом моделювання класів, їх атрибутів, методів та зв'язків між ними. Класи представляють основні будівельні блоки системи, а діаграма класів допомагає зрозуміти їх ролі та взаємодії. Класи визначають типи об'єктів, що використовуються у системі. Кожен клас має набір атрибутів (змінних) та методів (функцій), які визначають його поведінку. Атрибути є властивостями або характеристиками класу. Наприклад, для класу "Автомобіль" атрибутами можуть бути "колір", "модель", "рік випуску". Методи є функціями, що визначають поведінку класу. Для класу "Автомобіль" методами можуть бути "запустити двигун()", "зупинити двигун()". Зв'язки відображають відношення між класами. Вони можуть бути асоціаціями, агрегаціями, композиціями або успадкуванням. Наприклад, клас "Водій" може бути пов'язаний з класом "Автомобіль" асоціацією "керує".

UML (Unified Modeling Language) є стандартним підходом до моделювання систем з використанням об'єктно-орієнтованого підходу. UML включає декілька типів діаграм, кожна з яких має своє призначення та специфіку. Діаграма варіантів використання (Use Case Diagram) показує взаємодію між користувачами (акторами) і системою через різні сценарії використання. Допомагає визначити функціональні вимоги до системи. Діаграма послідовності (Sequence Diagram) відображає взаємодію між об'єктами у вигляді послідовності повідомлень, що передаються між ними. Використовується для детального опису сценаріїв використання. Діаграма активності (Activity Diagram) моделює потоки роботи або бізнес-процеси. Відображає послідовність дій і можливі розгалуження процесу. Діаграма станів (State Diagram) показує стани об'єкта і переходи між цими станами. Використовується для опису поведінки об'єктів, що мають кінцеву кількість станів.

Діаграми модулів відображають логічну і фізичну структуру системи у вигляді модулів або компонентів, що взаємодіють між собою. Кожен модуль виконує окрему функцію в системі і може бути розроблений незалежно. Модулі є відносно незалежними блоками системи, які можуть бути розроблені, протестовані і замінені окремо. Модулі можуть взаємодіяти через інтерфейси або шини даних. Інтерфейси визначають правила взаємодії між модулями. Це можуть бути функціональні інтерфейси (API), протоколи обміну даними або інші механізми зв'язку. Залежності відображають взаємозалежність модулів. Наприклад, модуль користувача може залежати від модуля аутентифікації для перевірки прав доступу.

Процес проектування внутрішньої будови системи починається з аналізу вимог і створення високорівневих моделей. Ці моделі допомагають визначити основні компоненти системи та їх взаємодію. Аналіз вимог включає визначення функціональних та нефункціональних вимог до системи. Створення діаграми

варіантів використання допомагає визначити сценарії взаємодії користувачів із системою. Деталізація діаграм класів включає визначення класів, їх атрибутів, методів і зв'язків між ними. Розробка діаграм послідовності описує сценарії використання на рівні взаємодії об'єктів. Моделювання процесів за допомогою діаграм активності описує бізнес-процеси і робочі потоки. Проектування діаграм станів описує поведінку об'єктів у різних станах. Створення діаграм модулів включає визначення логічної і фізичної структури системи, розподіл функцій між модулями.

Проектування внутрішньої будови системи є складним і багатогранним процесом, що включає використання різних моделей і діаграм для детального опису структури і функціонування системи. Використання діаграм класів, UML діаграм і діаграм модулів дозволяє створити чітке уявлення про систему, забезпечуючи ефективну розробку, тестування і подальше супроводження програмного забезпечення.

На рисунку 3.7 представлено діаграму класів системи.

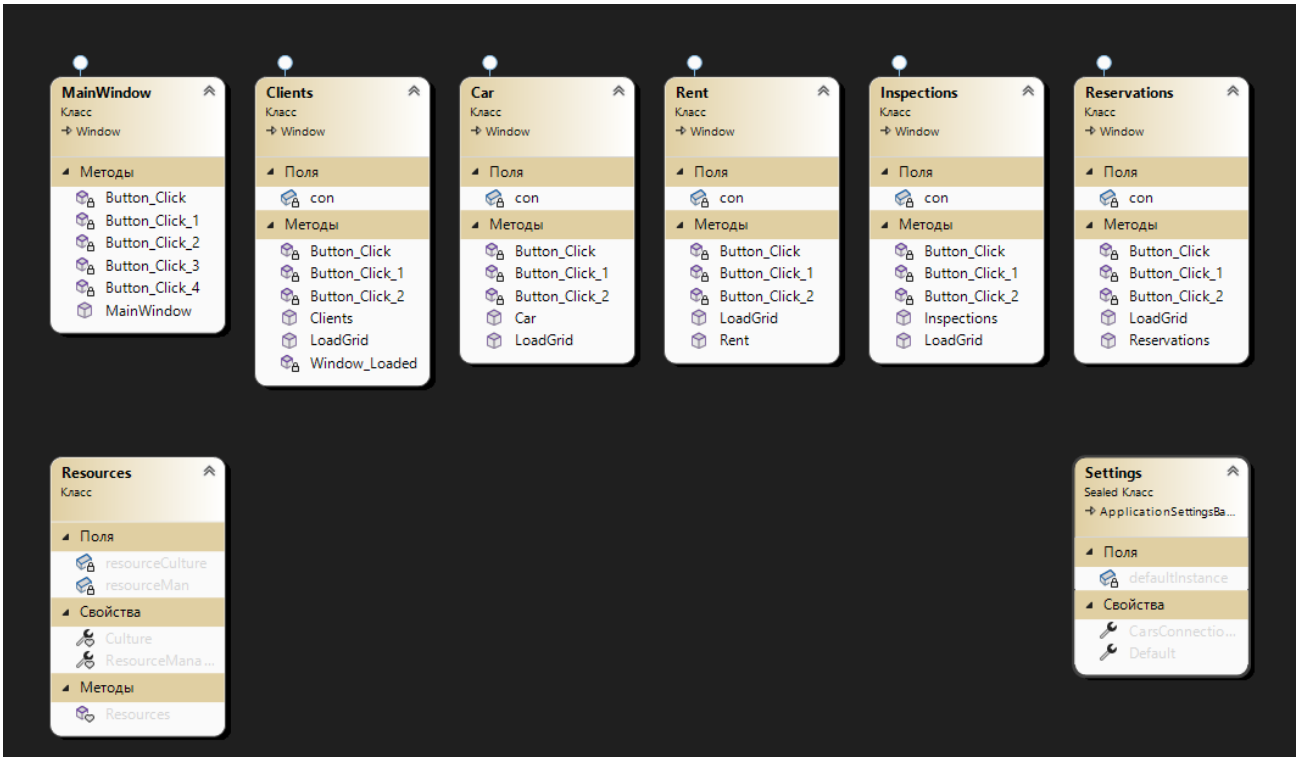


Рис.3.7 Діаграма класів системи

3.5 Структура проекту

Застосунок створений за допомогою WPF з використанням мови програмування C#.

На рисунку 3.8 представлено оглядач рішень програмного застосунку. В таблиці 3.1 представлено опис модулів проекту.

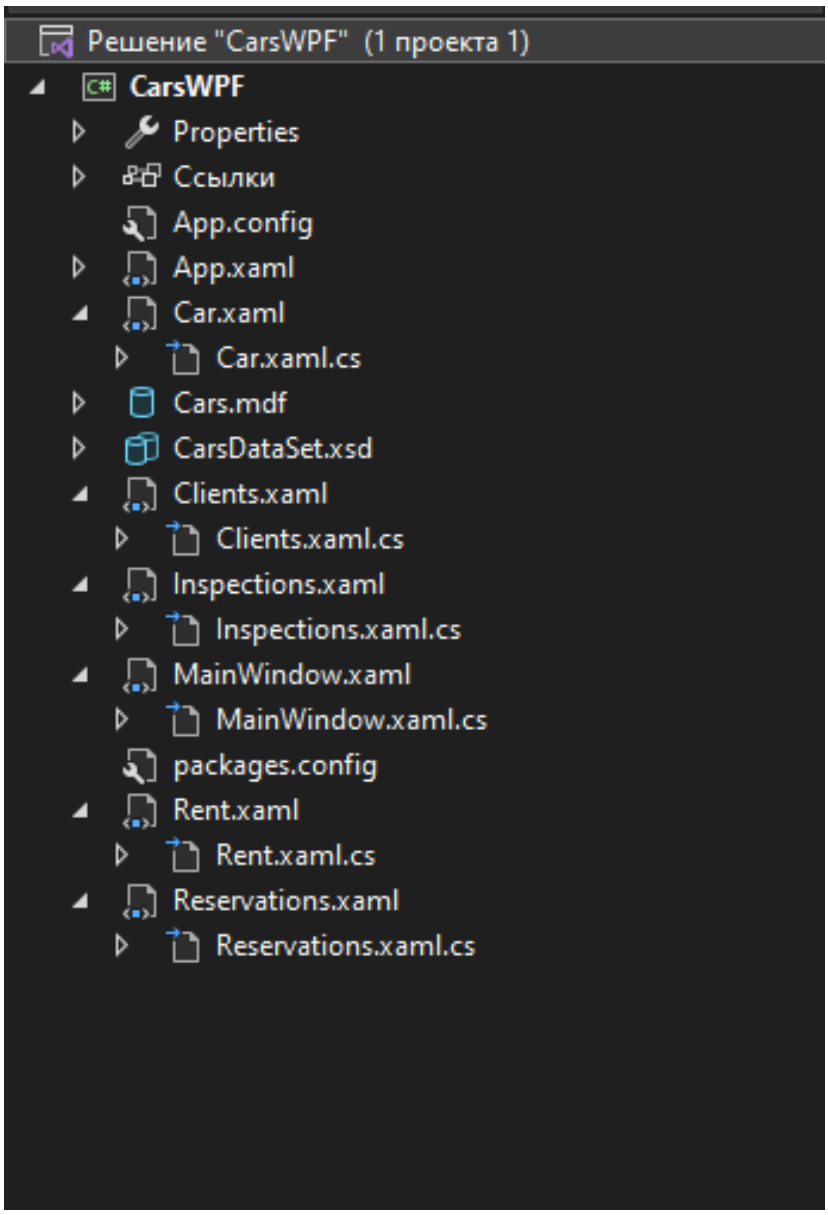


Рис.3.8 Оглядач рішень

Таблиця 3.1

Призначення модулів програми

Назва модуля	Призначення модуля
Car.xaml.cs	Вікно для роботи з сутністю, що зберігає дані про машини.
Clients.xaml.cs	Вікно для роботи з сутністю, що зберігає дані про клієнтів.
Inspections.xaml.cs	Вікно для роботи з сутністю, що зберігає дані про тех.оглядів.
MainWindow.xaml.cs	Головне вікно програми, яке містить меню системи.
Rent.xaml.cs	Вікно для роботи з сутністю, що зберігає дані про лізинги.
Reservations.xaml.cs	Вікно для роботи з сутністю, що зберігає дані про бронювання.

3.6 Опис роботи програми

Після запуску програми відкривається головне меню програми, яке містить 5 кнопок, які відкривають вікна сутностей БД.

Вікно клієнтів містить поля для введення ПІБ, пошти, телефону та адреси, таблицю для введення даних і кнопки для додавання, видалення і оновлення даних.

Вікно машин містить поля для введення марки, моделі, року випуску та поточного статусу для введення даних і кнопки для додавання, видалення і оновлення даних.

Вікно тех.оглядів містить поля для введення ключа машини, дати огляду та результату огляду, таблицю для введення даних і кнопки для додавання, видалення і оновлення даних.

Вікно прокатів містить поля для введення ключа машини, ключа клієнта, дати початку і кінця прокату, вартість прокату, таблицю для введення даних і кнопки для додавання, видалення і оновлення даних.

Вікно бронювань містить поля для введення ключа машини, ключа клієнта, дати початку і кінця прокату, таблицю для введення даних і кнопки для додавання, видалення і оновлення даних.

При відкритті будь-якого вікна з головного меню — меню приховується. При закритті периферичного вікна — відкривається головне меню.

Детальний опис й пояснення процесів окремих функцій розробляємої системи утворюють функціональну схему програми(рис. 3.9).

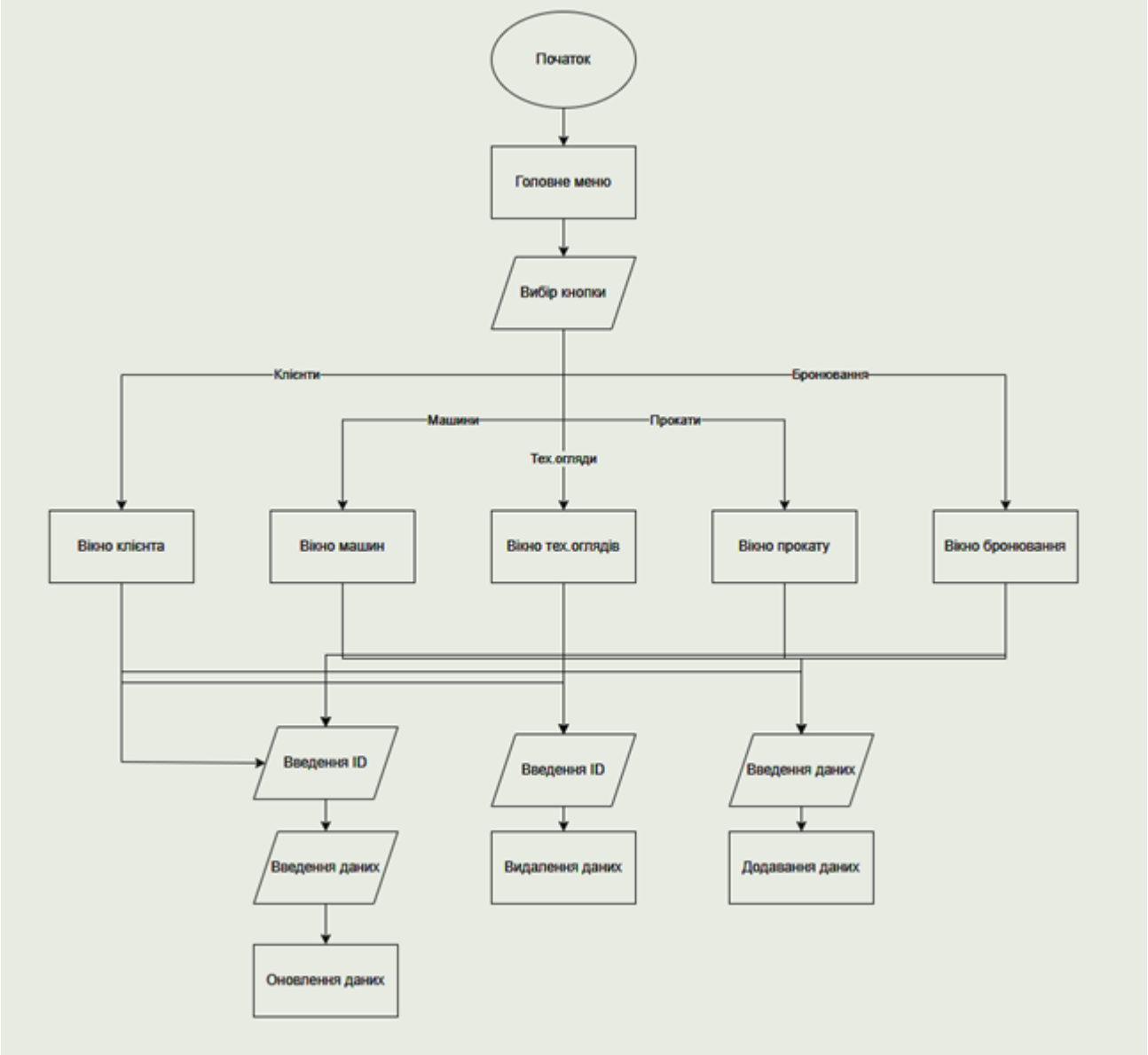


Рис. 3.9 Функціональна схема програмного коду

3.7 Створення інтерфейсу системи

Розробка графічного інтерфейсу користувача (GUI) є важливим етапом у створенні будь-якої програмної системи, оскільки він є головним засобом взаємодії користувача з додатком. Для створення інтерфейсу нашої системи було обрано мову програмування C# у поєднанні з технологією Windows Presentation Foundation (WPF), яка забезпечує потужні можливості для розробки сучасних та функціональних інтерфейсів. Вибір C# і WPF був обґрунтований їхньою інтеграцією з середовищем розробки Visual Studio, широким спектром інструментів для дизайну інтерфейсу, а також підтримкою сучасних стандартів UI/UX.

Розробка інтерфейсу почалася з проектування його структури. Основні принципи дизайну включали простоту, консистентність і доступність, що дозволяє створити інтуїтивно зрозумілий інтерфейс. На етапі проектування було створено кілька ескізів, які демонстрували основні компоненти інтерфейсу, такі як головне вікно, форми введення даних і екрани звітів. Після узгодження дизайну з потенційними користувачами та отримання їхніх відгуків, був розроблений інтерактивний прототип у програмі Figma, що дозволило виявити та виправити недоліки на ранньому етапі.Основний процес розробки інтерфейсу включав створення XAML-файлів для опису структури інтерфейсу.

Після завершення розробки інтерфейсу було проведено функціональне та юзабіліті-тестування. Функціональне тестування включало перевірку коректності роботи всіх елементів інтерфейсу та їх взаємодії з бізнес-логікою додатку. Юзабіліті-тестування проводилося з залученням реальних користувачів, що дозволило отримати цінні відгуки та внести покращення в інтерфейс. Відгуки користувачів допомогли виявити проблемні місця у використанні інтерфейсу, а також покращити загальний досвід взаємодії з додатком.

Таким чином, процес створення графічного інтерфейсу користувача включав вибір інструментів, проектування, реалізацію та тестування. Використання C# та WPF дозволило створити сучасний, функціональний та зручний у використанні інтерфейс, який відповідає потребам кінцевих користувачів та забезпечує ефективну взаємодію з системою.

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

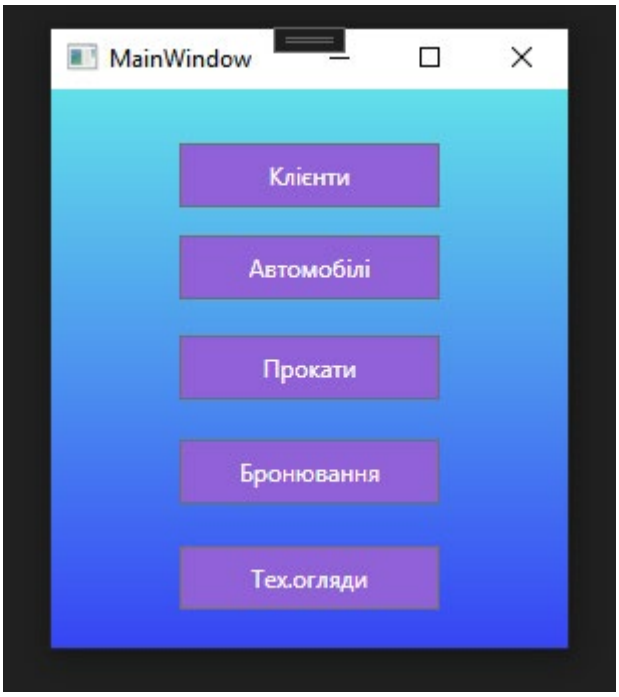


Рис. 3.10 Головне вікно програми

Вікно головного меню (рис. 3.10) містить 5 кнопок:

1. Кнопка «Клієнти». При натисканні кнопки відкривається вікно клієнтів, яке відповідає за роботу з відповідною сутністю бази даних.
2. Кнопка «Автомобілі». При натисканні кнопки відкривається вікно машин, яке відповідає за роботу з відповідною сутністю бази даних.
3. Кнопка «Прокати». При натисканні кнопки відкривається вікно прокатів, яке відповідає за роботу з відповідною сутністю бази даних.

4. Кнопка «Бронювання». При натисканні кнопки відкривається вікно бронювань, яке відповідає за роботу з відповідною сутністю бази даних.
5. Кнопка «Тех.огляди». При натисканні кнопки відкривається вікно тех.оглядів, яке відповідає за роботу з відповідною сутністю бази даних.

–

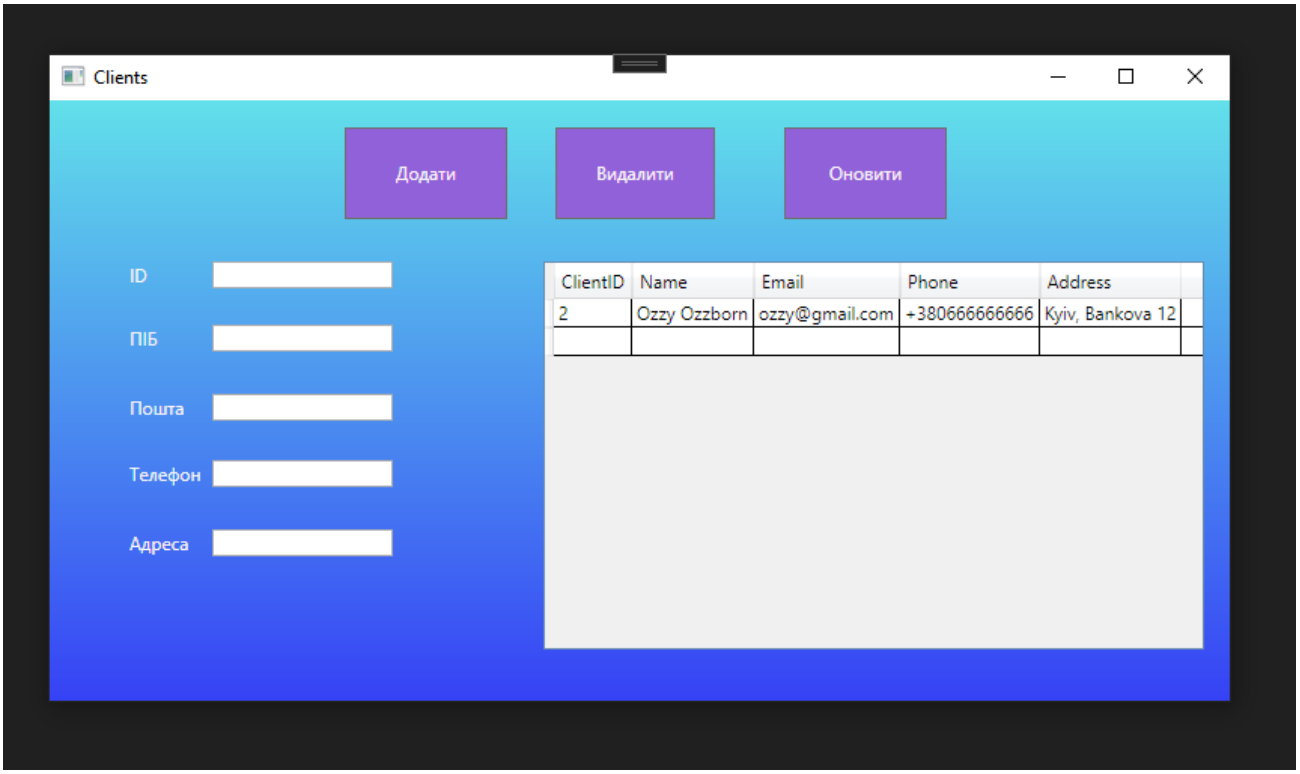


Рис. 3.11 Вікно клієнтів

Вікно клієнтів містить 3 кнопки:

1. Кнопка додавання нового запису
2. Кнопка видалення запису
3. Кнопка оновлення запису

Окрім цього, форма містить 5 текстових полів для введення даних і таблицю для виведення даних.

Рис. 3.12 Вікно автомобілів

Вікно клієнтів містить 3 кнопки:

4. Кнопка додавання нового запису
5. Кнопка видалення запису
6. Кнопка оновлення запису

Окрім цього, форма містить 5 текстових полів для введення даних і таблицю для виведення даних.

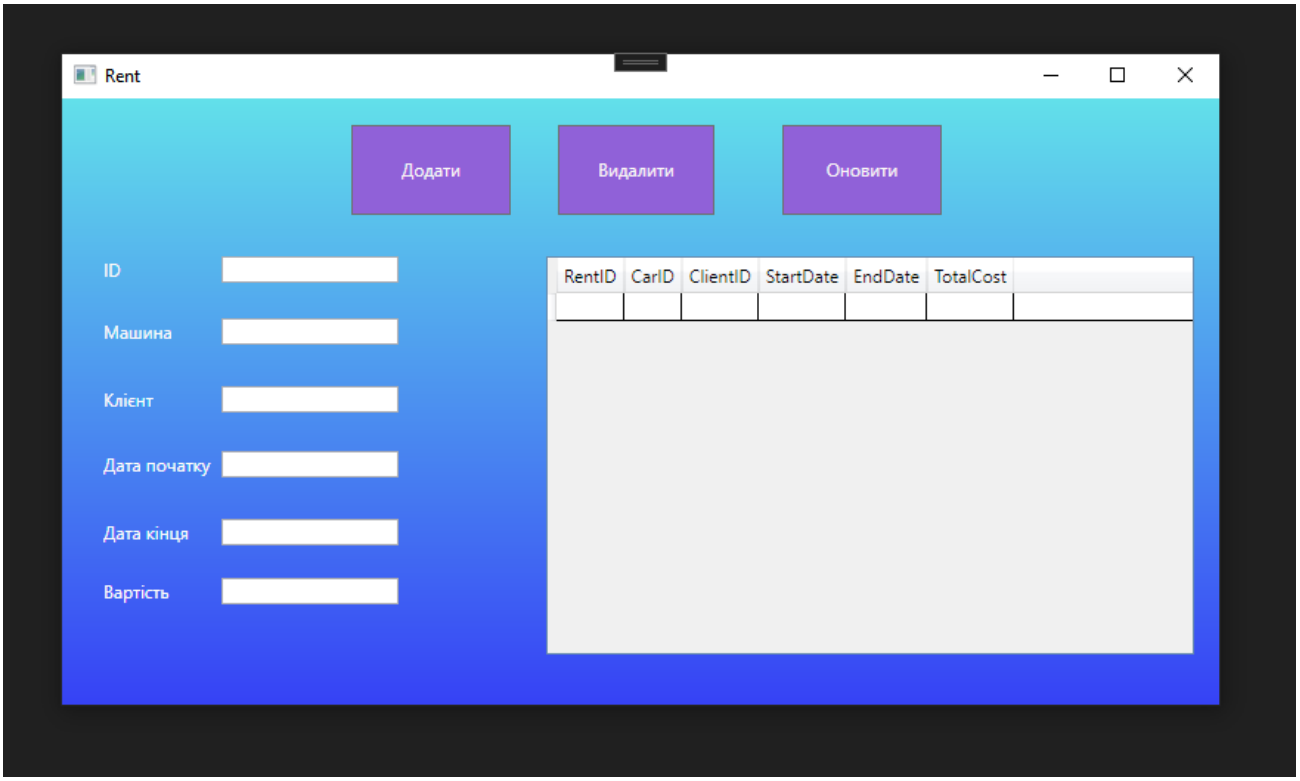


Рис. 3.13 Вікно покатів

Вікно клієнтів містить 3 кнопки:

7. Кнопка додавання нового запису
8. Кнопка видалення запису
9. Кнопка оновлення запису

Окрім цього, форма містить 6 текстових полів для введення даних і таблицю для виведення даних.

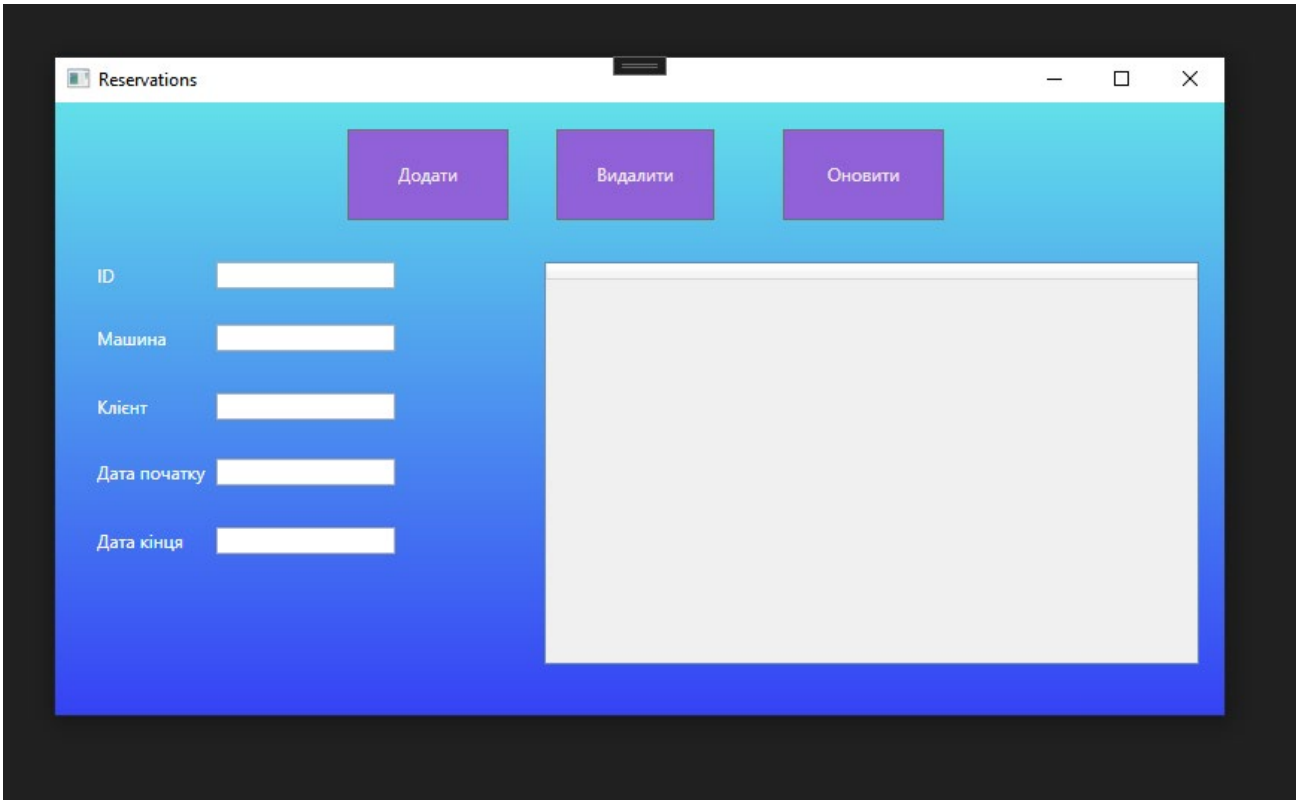


Рис. 3.14 Вікно бронювань

Вікно клієнтів містить 3 кнопки:

10. Кнопка додавання нового запису
11. Кнопка видалення запису
12. Кнопка оновлення запису

Окрім цього, форма містить 4 текстових поля для введення даних і таблицю для виведення даних.

3.8 Розробка основних алгоритмів

Основними для даного програмного продукту є алгоритми роботи за даними, а саме зчитування, додавання, видалення та редагування.

В першу чергу слід розглянути сам механізм роботи з БД.

Код складається з двох частин: класу DataManager і інтерфейсу IEntity.

```
public interface IEntity
{
    int Id { get; }
}
```

Цей інтерфейс визначає властивість Id, яка повинна бути реалізована в класах, що імплементують цей інтерфейс. Це дозволяє використовувати об'єкти з інтерфейсом IEntity в DataManager та використовувати їхні ідентифікатори для управління ними.

```
public class DataManager
{
    public void Create<T>(T entity, List<T> entities, string filePath) where T : IEntity
    {
        entities.Add(entity);
        FileStorageHelper.SaveToFile(entities, filePath);
    }

    public void Delete<T>(int id, List<T> entities, string filePath) where T : IEntity
    {
        var entityToRemove = entities.FirstOrDefault(e => e.Id == id);
        if (entityToRemove != null)
        {
            entities.Remove(entityToRemove);
            FileStorageHelper.SaveToFile(entities, filePath);
        }
    }

    public void Update<T>(int id, T newEntity, List<T> entities, string filePath) where T :
IEntity
    {
        var index = entities.FindIndex(e => e.Id == id);
        if (index != -1)
        {
            entities[index] = newEntity;
            FileStorageHelper.SaveToFile(entities, filePath);
        }
    }
}
```

Метод Create додає новий об'єкт до списку entities і зберігає цей список у файлі за допомогою класу FileStorageHelper.

Метод Delete видаляє об'єкт зі списку entities за його ідентифікатором і також зберігає оновлений список у файлі.

Метод Update замінює існуючий об'єкт у списку entities на новий об'єкт за його ідентифікатором і зберігає оновлений список у файлі.

Цей код демонструє простий підхід до управління об'єктами в пам'яті та їх збереження в файлі.

Наступним кроком є адаптер даних.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CarsWPF
{
   public class CarAdapter : IEntity
   {
      public int Id { get; set; }
      public string Mark { get; set; }
      public string Model { get; set; }
      public string Year { get; set; }
      public string Status { get; set; }
   }

}
```

Цей код визначає клас CarAdapter, який реалізує інтерфейс IEntity. Клас CarAdapter представляє об'єкт автомобіля з такими властивостями: Id (ідентифікатор), Mark (марка), Model (модель), Year (рік випуску), Status (стан).

Інтерфейс IEntity має властивість Id, яка визначає унікальний ідентифікатор об'єкта. Цей інтерфейс використовується для реалізації об'єктів, що можуть бути ідентифіковані у додатку.

І останнім кроком є реалізація взаємодії між користувачем і менеджером даних.

```
   public Car()
   {
      InitializeComponent();
      LoadGrid();
   }
```

Конструктор Car():

Ініціалізує компоненти (інтерфейс) форми.

Викликає метод LoadGrid(), який завантажує дані про автомобілі в DataGrid.

```
public void LoadGrid()
{
    cars = FileStorageHelper.LoadFromFile<CarAdapter>(FileStorageHelper.CarFilePath);
    dataGrid.ItemsSource = cars;
}
```

Метод LoadGrid():

Завантажує дані про автомобілі з файлу за допомогою методу LoadFromFile класу FileStorageHelper.

Встановлює ItemsSource для DataGrid, щоб відобразити завантажені дані.

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var newCar = new CarAdapter
    {
        Id = GetNewCarId(),
        Mark = MarkTxt.Text,
        Model = ModelTxt.Text,
        Year = YearTxt.Text,
        Status = StatusTxt.Text
    };

    dataManager.Create(newCar, cars, FileStorageHelper.CarFilePath);
    LoadGrid();
}
```

Метод Button_Click_1 (для додавання автомобіля):

Створює новий об'єкт CarAdapter з даними з текстових полів у інтерфейсі.

Викликає метод Create об'єкта dataManager, щоб додати новий автомобіль у список і зберегти його у файл.

Після цього викликає метод LoadGrid() для оновлення списку автомобілів у DataGrid.

```
private void Button_Click(object sender, RoutedEventArgs e)
```

```
{
    if (int.TryParse(SearchTxt.Text, out int carId))
    {
        dataManager.Delete<CarAdapter>(carId, cars, FileStorageHelper.CarFilePath);
        LoadGrid();
    }
}
```

Метод Button_Click (для видалення автомобіля):

Перевіряє, чи можна конвертувати текст у полі пошуку в число (ідентифікатор автомобіля).

Викликає метод Delete об'єкта dataManager, щоб видалити автомобіль за його ідентифікатором.

Після цього викликає метод LoadGrid() для оновлення списку автомобілів у DataGrid.

```
private void Button_Click_2(object sender, RoutedEventArgs e)
{
    if (int.TryParse(SearchTxt.Text, out int carId))
    {
        var updatedCar = new CarAdapter
        {
            Id = carId,
            Mark = MarkTxt.Text,
            Model = ModelTxt.Text,
            Year = YearTxt.Text,
            Status = StatusTxt.Text
        };

        dataManager.Update(carId, updatedCar, cars, FileStorageHelper.CarFilePath);
        LoadGrid();
    }
}
```

Метод Button_Click_2 (для оновлення даних про автомобіль):

Перевіряє, чи можна конвертувати текст у полі пошуку в число (ідентифікатор автомобіля).

Створює новий об'єкт CarAdapter з новими даними з текстових полів у інтерфейсі.

Викликає метод Update об'єкта dataManager, щоб оновити дані про автомобіль за його ідентифікатором.

Після цього викликає метод LoadGrid() для оновлення списку автомобілів у DataGrid.

```
    private int GetNewCarId()
    {
        return cars.Count > 0 ? cars.Max(c => c.Id) + 1 : 1;
    }
}
```

Метод GetNewCarId():

Повертає новий ідентифікатор для нового автомобіля, який є на 1 більшим за найбільший ідентифікатор у списку автомобілів, або 1, якщо список порожній.

Взаємодія користувача з іншими вікнами влаштована аналогічним чином.

3.9 Тестування системи

Тестування програмного забезпечення (ПЗ) є важливим етапом у розробці, який забезпечує відповідність програми очікуваним вимогам і допомагає виявити помилки перед випуском. Основні типи тестування включають функціональне тестування, яке перевіряє, чи виконує ПЗ свої функції правильно, та нефункціональне тестування, яке оцінює продуктивність, безпеку та інші якості. Використання тест кейсів дозволяє систематично перевірити різні аспекти ПЗ, забезпечуючи високу якість кінцевого продукту.

Тест кейси представлені в таблиці 3.2.

Таблиця 3.2

Тест кейси

№	Назва	Очікуваний результат	Отриманий результат
1	Додавання нового авто	Авто успішно додано до бази даних	Авто успішно додано до бази даних
2	Видалення авто	Авто успішно видалено з бази даних	Авто успішно видалено з бази даних
3	Оновлення інформації про авто	Інформація успішно оновлена	Інформація успішно оновлена
4	Додавання нового клієнта	Клієнта успішно додано до бази даних	Клієнта успішно додано до бази даних
5	Видалення клієнта	Клієнта успішно видалено з бази даних	Клієнта успішно видалено з бази даних
6	Оновлення інформації про клієнта	Інформація успішно оновлена	Інформація успішно оновлена
7	Додавання нової оренди	Оренду успішно додано до бази даних	Оренду успішно додано до бази даних
8	Видалення оренди	Оренду успішно видалено з бази даних	Оренду успішно видалено з бази даних
9	Оновлення інформації про оренду	Інформація успішно оновлена	Інформація успішно оновлена
10	Додавання нового резервування	Резервування успішно додано до бази даних	Резервування успішно додано до бази даних
11	Видалення резервування	Резервування успішно видалено з бази даних	Резервування успішно видалено з бази даних
12	Оновлення інформації про резервування	Інформація успішно оновлена	Інформація успішно оновлена

Усі тест кейси були успішно виконані, що свідчить про високу якість і правильну роботу програмного забезпечення.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено програмне забезпечення для управління автомобільним лізингом, яке включає в себе модулі зберігання, видалення, редагування та перегляду даних про клієнтів, автомобілі, угоди лізингу, бронювання та техогляди машин.

Досліджено основні аспекти управління автомобільним лізингом та проведено порівняльний аналіз існуючих рішень у цій галузі. Досліджено інструменти та технології для розробки застосунку. Розроблене програмне забезпечення відповідає сучасним вимогам управління автопарком та угодами лізингу, а його впровадження може значно удосконалити процес управління автомобільним лізингом для підприємств та приватних осіб.

Реалізовано програмне забезпечення що дозволить підприємствам та приватним особам ефективніше управляти автомобільним лізингом, зменшуючи витрати та підвищуючи загальну продуктивність управління автопарком.

Апробація результатів дослідження:

1.Докучаєв М.О. Залива В.В. Створення додатку для управління лізингом автотранспорту з використанням мови програмування c#. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 р., ДУІКТ, м.Київ. с.212

2. Докучаєв М.О. Залива В.В. Актуальність розробки додатків для управління лізингом автотранспорту. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 р., ДУІКТ, м.Київ. с. 119

ПЕРЕЛІК ПОСИЛАНЬ

1. "Selecting a development approach" (PDF). Centers for Medicare & Medicaid Services (CMS) Office of Information Service. United States Department of Health and Human Services (HHS). March 27, 2008 [Original Issuance: February 17, 2005]. Archived from the original (PDF) on June 20, 2012. Retrieved October 27, 2008.
2. Geoffrey Elliott (2004). Global Business Information Technology: an integrated systems approach. Pearson Education. p. 87.
3. Suryanarayana, Girish (2015). "Software Process versus Design Quality: Tug of War?". IEEE Software. 32 (4): 7–11. doi:10.1109/MS.2015.87.
4. "software development process". August 19, 2020. Archived from the original on September 27, 2020.
5. "Continuous Integration".
6. Booch, Grady (1991). Object Oriented Design: With Applications. Benjamin Cummings. p. 209. ISBN 9780805300918. Retrieved August 18, 2014.
7. Whitten, Jeffrey L.; Lonnie D. Bentley, Kevin C. Dittman. (2003). Systems Analysis and Design Methods. 6th edition. ISBN 0-256-19906-X.
8. Markus Rerych. "Wasserfallmodell > Entstehungskontext". Institut für Gestaltungs- und Wirkungsforschung, TU-Wien (in German). Retrieved November 28, 2007.
9. Conrad Weisert. "Waterfall methodology: there's no such thing!". Archived from the original on August 2, 2022.
10. Barry Boehm (August 1986). "A Spiral Model of Software Development and Enhancement". ACM SIGSOFT Software Engineering Notes. Association for Computing Machinery. 11 (4): 14–24. doi:10.1145/12944.12948. S2CID 1781829.

11. Richard H. Thayer; Barry W. Boehm (1986). Tutorial: software engineering project management. Computer Society Press of the IEEE. p. 130.
12. Barry W. Boehm (2000). Software cost estimation with Cocomo II: Volume 1.
13. "Foreword by Jason Fried | Shape Up". basecamp.com. Retrieved September 11, 2022.
14. "Is Shape Up just a nice theory?". Curious Lab. Retrieved September 12, 2022.
15. "Principles of Shaping | Shape Up". basecamp.com. Retrieved September 11, 2022.
16. "Bets, Not Backlogs | Shape Up". basecamp.com. Retrieved September 11, 2022.
17. "Hand Over Responsibility | Shape Up". basecamp.com. Retrieved September 11, 2022.
18. "Show Progress | Shape Up". basecamp.com. Retrieved September 12, 2022.
19. "Atlassian Marketplace". marketplace.atlassian.com. Retrieved September 12, 2022.
20. Lübke, Daniel; van Lessen, Tammo (2016). "Modeling Test Cases in BPMN for Behavior-Driven Development". IEEE Software. 33 (5): 15–21. doi:10.1109/MS.2016.117. S2CID 14539297.
21. Докучаєв М.О. Залива В.В. Створення додатку для управління лізингом автотранспорту з використанням мови програмування c#. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 року, ДУІКТ, м.Київ. с.212
22. Докучаєв М.О. Залива В.В. Актуальність розробки додатків для управління лізингом автотранспорту. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 року, ДУІКТ, м.Київ. с. 119

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка застосунку мовою C# для управління лізингом автотранспорту

Виконав студент 4 курсу
групи ПД-42
Докучаєв Максим Олегович
Керівник роботи
Професор філософії, К.т.н, старший викладач кафедри ІПЗ Залива
Віталій Вікторович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- Мета: удосконалення управління процесами автомобільного лізингу та прокату автомобілів.
- Об'єкт дослідження: процес управління автомобільним лізингом.
- Предмет дослідження: застосунок для управління автомобільним лізингом.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести дослідження та аналіз існуючих систем управління лізингом автомобілів для визначення їх переваг та недоліків.
2. Дослідити доступні технічні засоби й обрати набір інструментів для створення застосунку
3. Створити систему управління лізингом автомобілів, враховуючи функціональні вимоги та структуру бази даних.
4. Розробити та реалізувати застосунок для управління лізингом автомобілів, використовуючи технології C#, WPF та MS SQL Server.
5. Провести тестування розробленого застосунку для перевірки його працездатності та відповідності вимогам.

3

АНАЛІЗ АНАЛОГІВ

Властивості	Winner Leasing	CarLeasePro	LeasePlan
Додавання/редагування автомобілів	+	+	+
Технічний огляд авто	-	-	+
Бронювання авто	+	-	+
Оновлення данних клієнта	+	-	+
Оновлення данних автомобілей в парку	+	-	+
Зв'язок БД додатку з сайтом компанії.	-	+	-

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Можливість виведення даних про клієнтів, машини, технічні огляди, бронювання і прокат машин
2. Можливість додавати в базу даних дані про клієнтів, машини, технічні огляди, бронювання і прокат машин
3. Можливість редагувати дані
4. Можливість видаляти дані з системи

Нефункціональні вимоги:

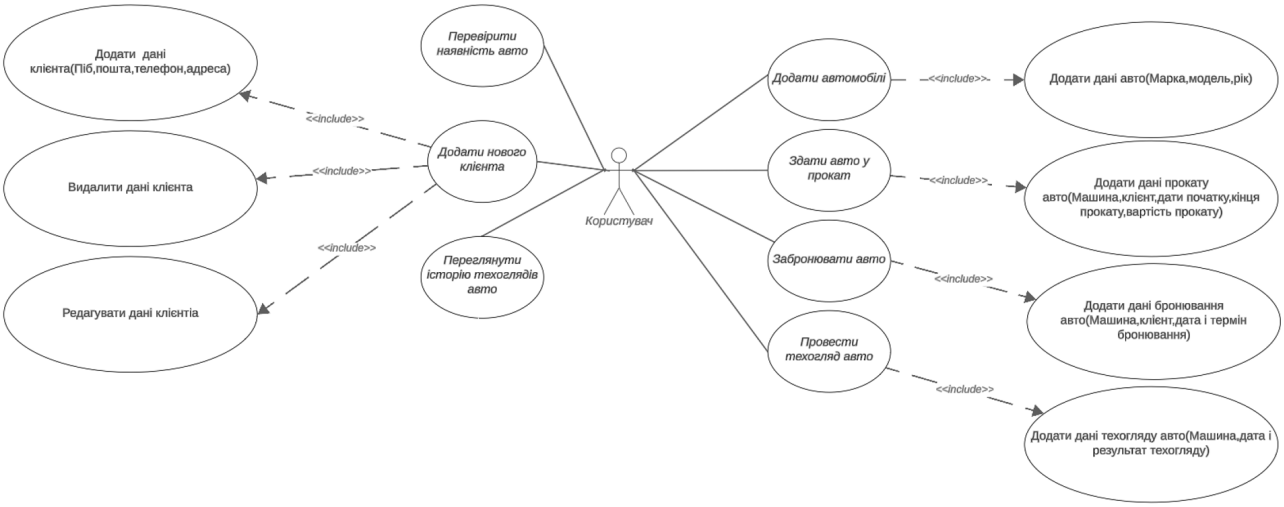
1. Зовнішній інтерфейс користувача має бути реалізованим за допомогою створення вікон в середовищі WPF додатку.
2. Додаток має бути багатовіконним
3. Усі поля введення повинні бути захищені від некоректного введення
4. Реалізація окремого вікна для кожної сутності бази даних

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

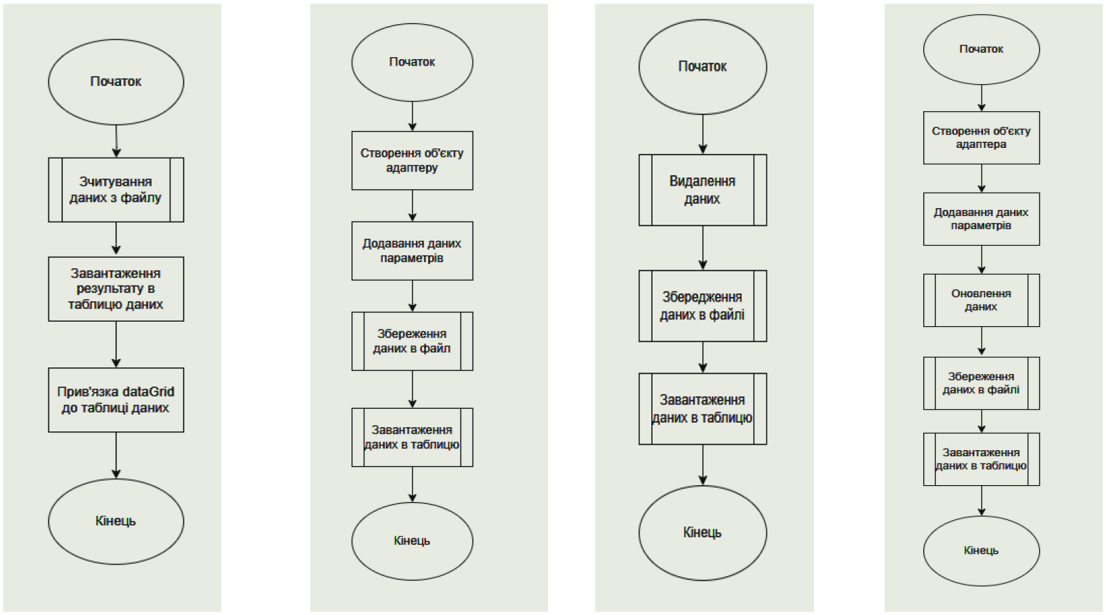
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



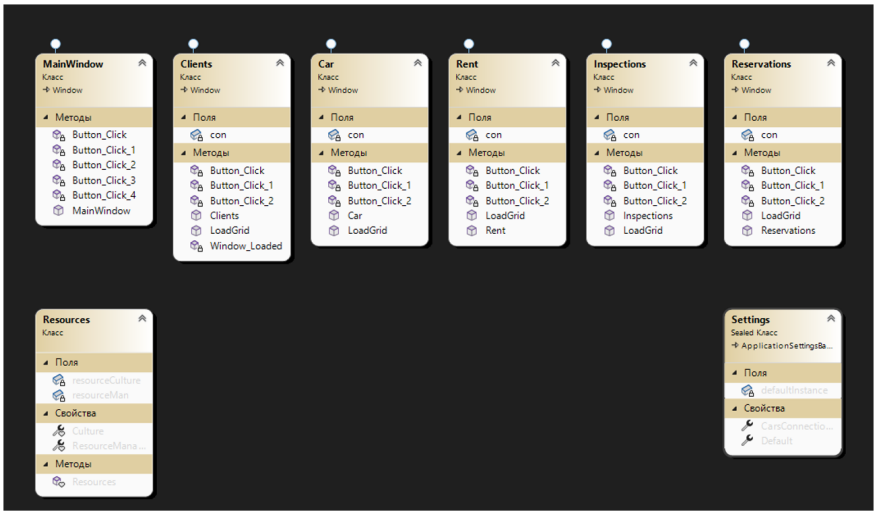
7

БЛОК-СХЕМИ ОСНОВНИХ АЛГОРИМТІВ



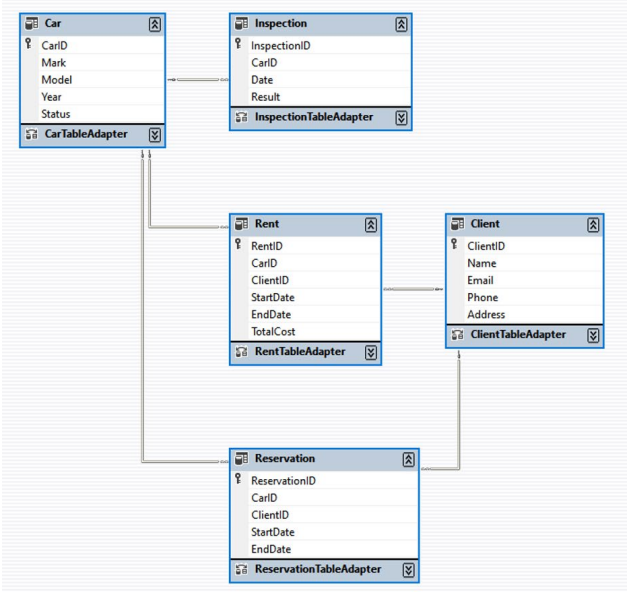
8

ДІАГРАМА КЛАСІВ



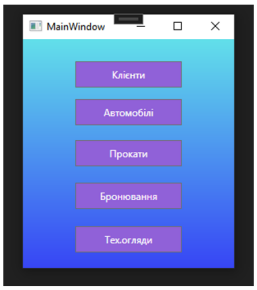
9

СХЕМА БАЗИ ДАНИХ



10

ЕКРАННІ ФОРМИ

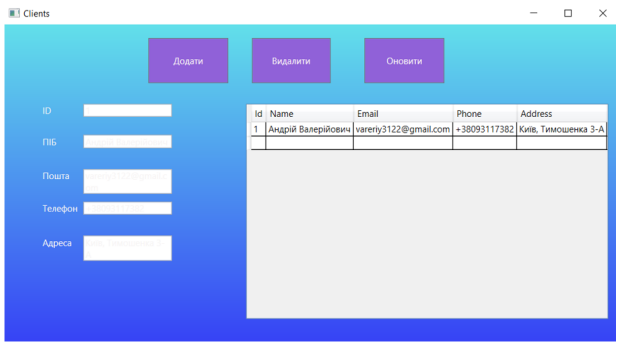


Головне вікно

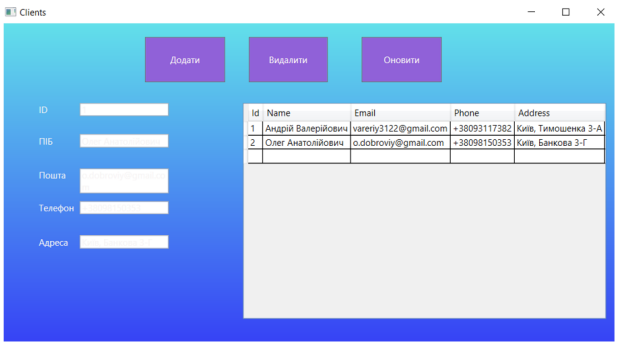
Вікно автомобілів

Вікно автомобілів з доданим авто Audi A7

ЕКРАННІ ФОРМИ



Вікно клієнтів



Додавання клієнта

ЕКРАННІ ФОРМИ

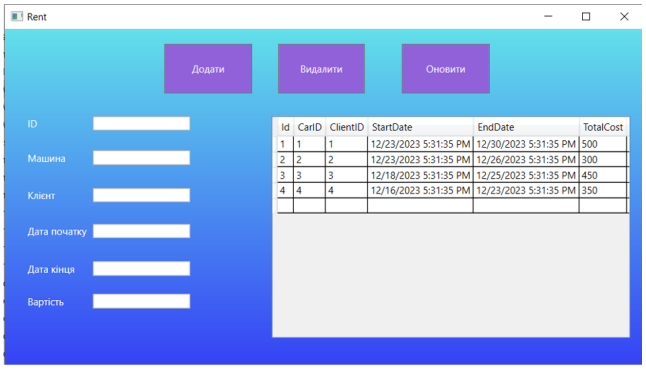
Вікно бронювання

Додали бронювання

13

Екранні форми

Вікно техогляду



Вікно прокатів

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1.Докучаєв М.О. Залива В.В. Створення додатку для управління лізингом автотранспорту з використанням мови програмування c#. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 р., ДУІКТ, м.Київ. с.212

2.Докучаєв М.О. Залива В.В. Актуальність розробки додатків для управління лізингом автотранспорту. Чертверта Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці і освіті” Збірник тез 15 травня 2024 р., ДУІКТ, м.Київ. с. 119

16

ВИСНОВКИ

1. Проаналізовано існуючі рішення для управління лізингом.
2. Встановлено переваги та недоліки застосунків.
3. Розроблено функціональні і нефункціональні вимоги.
4. Досліджено інструменти та технології для розробки застосунку.
5. Реалізовано застосунок для управління лізингом автомобілів.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНОГО КОДУ ТА XAML РОЗМІТКА

```
        <Window x:Class="CarsWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"

xmlns:d="http://schemas.microsoft.com/expression/ble
nd/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup
-compatibility/2006"
                xmlns:local="clr-namespace:CarsWPF"
                mc:Ignorable="d"
                Title="MainWindow"        Height="301"
Width="274">
            <Grid Margin="0,0,0,-6">
                <Button                 Content="Клієнти"
HorizontalAlignment="Center"        Margin="0,27,0,0"
VerticalAlignment="Top"   Height="32"   Width="130"
Click="Button_Click"/>
                <Button              Content="Автомобілі"
HorizontalAlignment="Center"        Margin="0,73,0,0"
VerticalAlignment="Top"   Height="32"   Width="130"
Click="Button_Click_1"/>
                <Button                 Content="Прокати"
HorizontalAlignment="Center"      Margin="0,123,0,0"
VerticalAlignment="Top"   Height="32"   Width="130"
Click="Button_Click_2"/>
                <Button           Content="Бронювання"
HorizontalAlignment="Center"      Margin="0,175,0,0"
VerticalAlignment="Top"   Height="32"   Width="130"
Click="Button_Click_3"/>
                <Button             Content="Тех.огляди"
HorizontalAlignment="Center"      Margin="0,228,0,0"
VerticalAlignment="Top"   Height="32"   Width="130"
Click="Button_Click_4"/>
            </Grid>
        </Window>


        <Window x:Class="CarsWPF.Car"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"

xmlns:d="http://schemas.microsoft.com/expression/ble
nd/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup
-compatibility/2006"
                xmlns:local="clr-namespace:CarsWPF"
                mc:Ignorable="d"
                Title="Car" Height="450" Width="800">
            <Grid Margin="400,282,0,0">
                <DataGrid              x:Name="dataGrid"
Margin="-72,-175,17,34"/>
                <Label                  Content="Марка"
HorizontalAlignment="Left"    Margin="-352,-137,0,0"
VerticalAlignment="Top"/>
                <Label                  Content="Модель"
HorizontalAlignment="Left"    Margin="-352,-91,0,0"
VerticalAlignment="Top"/>
                <Label                      Content="Рік"
HorizontalAlignment="Left"    Margin="-352,-47,0,0"
VerticalAlignment="Top"/>
                <Label                  Content="Статус"
HorizontalAlignment="Left"    Margin="-352,-1,0,0"
VerticalAlignment="Top"/>
                <TextBox             x:Name="MarkTxt"
HorizontalAlignment="Left"    Margin="-292,-133,0,0"
TextWrapping="Wrap"       VerticalAlignment="Top"
Width="120"/>
                <TextBox             x:Name="ModelTxt"
HorizontalAlignment="Left"    Margin="-292,-87,0,0"
TextWrapping="Wrap"       VerticalAlignment="Top"
Width="120"/>
                <TextBox             x:Name="YearTxt"
HorizontalAlignment="Left"    Margin="-292,-43,0,0"
TextWrapping="Wrap"       VerticalAlignment="Top"
Width="120"/>
                <TextBox             x:Name="StatusTxt"
HorizontalAlignment="Left"    Margin="-292,3,0,0"
TextWrapping="Wrap"       VerticalAlignment="Top"
Width="120"/>
                <Button                 Content="Додати"
HorizontalAlignment="Left"    Margin="-204,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_1"/>
                <Button              Content="Видалити"
HorizontalAlignment="Left"    Margin="-64,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="106"
Click="Button_Click"/>
                <Button              Content="Оновити"
HorizontalAlignment="Left"    Margin="88,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_2"/>
                <Label                      Content="ID"
HorizontalAlignment="Left"    Margin="-352,-179,0,0"
VerticalAlignment="Top"/>
                <TextBox             x:Name="SearchTxt"
HorizontalAlignment="Left"    Margin="-292,-175,0,0"
```

```
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>

            </Grid>
        </Window>

        <Window x:Class="CarsWPF.Clients"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"

xmlns:d="http://schemas.microsoft.com/expression/ble
nd/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup
-compatibility/2006"
                xmlns:local="clr-namespace:CarsWPF"
                mc:Ignorable="d"
                Title="Clients"                Height="450"
Width="800" Loaded="Window_Loaded">
            <Grid Margin="400,282,0,0">
                <DataGrid              x:Name="dataGrid"
Margin="-72,-175,17,34"/>
                <Label                      Content="ПІБ"
HorizontalAlignment="Left"    Margin="-352,-137,0,0"
VerticalAlignment="Top"/>
                <Label                  Content="Пошта"
HorizontalAlignment="Left"     Margin="-352,-91,0,0"
VerticalAlignment="Top"/>
                <Label                  Content="Телефон"
HorizontalAlignment="Left"     Margin="-352,-47,0,0"
VerticalAlignment="Top"/>
                <Label                  Content="Адреса"
HorizontalAlignment="Left"      Margin="-352,-1,0,0"
VerticalAlignment="Top"/>
                <TextBox            x:Name="NameTxt"
HorizontalAlignment="Left"    Margin="-292,-133,0,0"
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>
                <TextBox            x:Name="EmailTxt"
HorizontalAlignment="Left"     Margin="-292,-87,0,0"
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>
                <TextBox            x:Name="PhoneTxt"
HorizontalAlignment="Left"     Margin="-292,-43,0,0"
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>
                <TextBox            x:Name="AddressTxt"
HorizontalAlignment="Left"       Margin="-292,3,0,0"
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>
                <Button                 Content="Додати"
HorizontalAlignment="Left"    Margin="-204,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_1"/>
                <Button                 Content="Видалити"
HorizontalAlignment="Left"     Margin="-64,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="106"
Click="Button_Click"/>
                <Button                 Content="Оновити"
HorizontalAlignment="Left"      Margin="88,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_2"/>
                <Label                      Content="ID"
HorizontalAlignment="Left"    Margin="-352,-179,0,0"
VerticalAlignment="Top"/>
                <TextBox            x:Name="SearchTxt"
HorizontalAlignment="Left"    Margin="-292,-175,0,0"
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>


            </Grid>
        </Window>

        <Window x:Class="CarsWPF.Inspections"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"

xmlns:d="http://schemas.microsoft.com/expression/ble
nd/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup
-compatibility/2006"
                xmlns:local="clr-namespace:CarsWPF"
                mc:Ignorable="d"
                Title="Inspections"            Height="450"
Width="800">
            <Grid Margin="400,282,0,0">
                <DataGrid              x:Name="dataGrid"
Margin="-72,-175,17,34"/>
                <Label                  Content="Машина"
HorizontalAlignment="Left"    Margin="-377,-137,0,0"
VerticalAlignment="Top"/>
                <Label                      Content="Дата"
HorizontalAlignment="Left"     Margin="-377,-95,0,0"
VerticalAlignment="Top"/>
                <TextBox            x:Name="CarTxt"
HorizontalAlignment="Left"    Margin="-292,-133,0,0"
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>
                <TextBox            x:Name="DateTxt"
HorizontalAlignment="Left"     Margin="-292,-91,0,0"
TextWrapping="Wrap"        VerticalAlignment="Top"
Width="120"/>
                <Button                 Content="Додати"
HorizontalAlignment="Left"    Margin="-204,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_1"/>
```

```
            <Button Content="Видалити"
HorizontalAlignment="Left" Margin="-64,-264,0,0"
VerticalAlignment="Top" Height="61" Width="106"
Click="Button_Click"/>
            <Button Content="Оновити"
HorizontalAlignment="Left" Margin="88,-264,0,0"
VerticalAlignment="Top" Height="61" Width="108"
Click="Button_Click_2"/>
            <Label Content="ID"
HorizontalAlignment="Left" Margin="-377,-179,0,0"
VerticalAlignment="Top"/>
            <TextBox x:Name="SearchTxt"
HorizontalAlignment="Left" Margin="-292,-175,0,0"
TextWrapping="Wrap" VerticalAlignment="Top"
Width="120"/>
            <Label Content="Результат"
HorizontalAlignment="Left" Margin="-377,-50,0,0"
VerticalAlignment="Top"/>
            <TextBox x:Name="ResultTxt"
HorizontalAlignment="Left" Margin="-292,-46,0,0"
TextWrapping="Wrap" VerticalAlignment="Top"
Width="120"/>


        </Grid>
    </Window>

    <Window x:Class="CarsWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"

xmlns:d="http://schemas.microsoft.com/expression/ble
nd/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup
-compatibility/2006"
            xmlns:local="clr-namespace:CarsWPF"
            mc:Ignorable="d"
            Title="MainWindow" Height="301"
Width="274">
        <Grid Margin="0,0,0,-6">
            <Button Content="Клієнти"
HorizontalAlignment="Center" Margin="0,27,0,0"
VerticalAlignment="Top" Height="32" Width="130"
Click="Button_Click"/>
            <Button Content="Автомобілі"
HorizontalAlignment="Center" Margin="0,73,0,0"
VerticalAlignment="Top" Height="32" Width="130"
Click="Button_Click_1"/>
            <Button Content="Прокати"
HorizontalAlignment="Center" Margin="0,123,0,0"
VerticalAlignment="Top" Height="32" Width="130"
Click="Button_Click_2"/>
            <Button Content="Бронювання"
HorizontalAlignment="Center" Margin="0,175,0,0"
VerticalAlignment="Top" Height="32" Width="130"
Click="Button_Click_3"/>
            <Button Content="Тех.огляди"
HorizontalAlignment="Center" Margin="0,228,0,0"
VerticalAlignment="Top" Height="32" Width="130"
Click="Button_Click_4"/>
        </Grid>
    </Window>


    <Window x:Class="CarsWPF.Rent"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"

xmlns:d="http://schemas.microsoft.com/expression/ble
nd/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup
-compatibility/2006"
            xmlns:local="clr-namespace:CarsWPF"
            mc:Ignorable="d"
            Title="Rent" Height="450"
Width="800">
        <Grid Margin="400,282,0,0">
            <DataGrid x:Name="dataGrid"
Margin="-72,-175,17,34"/>
            <Label Content="Машина"
HorizontalAlignment="Left" Margin="-377,-137,0,0"
VerticalAlignment="Top"/>
            <Label Content="Клієнт"
HorizontalAlignment="Left" Margin="-377,-91,0,0"
VerticalAlignment="Top"/>
            <Label Content="Дата початку"
HorizontalAlignment="Left" Margin="-377,-47,0,0"
VerticalAlignment="Top"/>
            <Label Content="Дата кінця"
HorizontalAlignment="Left" Margin="-377,-1,0,0"
VerticalAlignment="Top"/>
            <TextBox x:Name="CarTxt"
HorizontalAlignment="Left" Margin="-292,-133,0,0"
TextWrapping="Wrap" VerticalAlignment="Top"
Width="120"/>
            <TextBox x:Name="ClientTxt"
HorizontalAlignment="Left" Margin="-292,-87,0,0"
TextWrapping="Wrap" VerticalAlignment="Top"
Width="120"/>
            <TextBox x:Name="StartTxt"
HorizontalAlignment="Left" Margin="-292,-43,0,0"
TextWrapping="Wrap" VerticalAlignment="Top"
Width="120"/>
            <TextBox x:Name="EndTxt"
HorizontalAlignment="Left" Margin="-292,3,0,0"
```

```
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>
            <Button  Content="Додати"
HorizontalAlignment="Left"  Margin="-204,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_1"/>
            <Button  Content="Видалити"
HorizontalAlignment="Left"  Margin="-64,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="106"
Click="Button_Click"/>
            <Button  Content="Оновити"
HorizontalAlignment="Left"  Margin="88,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_2"/>
            <Label  Content="ID"
HorizontalAlignment="Left"  Margin="-377,-179,0,0"
VerticalAlignment="Top"/>
            <TextBox  x:Name="SearchTxt"
HorizontalAlignment="Left"  Margin="-292,-175,0,0"
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>
            <Label  Content="Вартість"
HorizontalAlignment="Left"  Margin="-377,39,0,0"
VerticalAlignment="Top"/>
            <TextBox  x:Name="CostTxt"
HorizontalAlignment="Left"  Margin="-292,43,0,0"
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>


        </Grid>
    </Window>

    <Window x:Class="CarsWPF.Reservations"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"

xmlns:d="http://schemas.microsoft.com/expression/ble
nd/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup
-compatibility/2006"
            xmlns:local="clr-namespace:CarsWPF"
            mc:Ignorable="d"
            Title="Reservations"  Height="450"
Width="800">
        <Grid Margin="400,282,0,0">
            <DataGrid  x:Name="dataGrid"
Margin="-72,-175,17,34"/>
            <Label  Content="Машина"
HorizontalAlignment="Left"  Margin="-377,-137,0,0"
VerticalAlignment="Top"/>
            <Label  Content="Клієнт"
HorizontalAlignment="Left"  Margin="-377,-91,0,0"
VerticalAlignment="Top"/>
            <Label  Content="Дата початку"
HorizontalAlignment="Left"  Margin="-377,-47,0,0"
VerticalAlignment="Top"/>
            <Label  Content="Дата кінця"
HorizontalAlignment="Left"  Margin="-377,-1,0,0"
VerticalAlignment="Top"/>
            <TextBox  x:Name="CarTxt"
HorizontalAlignment="Left"  Margin="-292,-133,0,0"
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>
            <TextBox  x:Name="ClientTxt"
HorizontalAlignment="Left"  Margin="-292,-87,0,0"
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>
            <TextBox  x:Name="StartTxt"
HorizontalAlignment="Left"  Margin="-292,-43,0,0"
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>
            <TextBox  x:Name="EndTxt"
HorizontalAlignment="Left"  Margin="-292,3,0,0"
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>
            <Button  Content="Додати"
HorizontalAlignment="Left"  Margin="-204,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_1"/>
            <Button  Content="Видалити"
HorizontalAlignment="Left"  Margin="-64,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="106"
Click="Button_Click"/>
            <Button  Content="Оновити"
HorizontalAlignment="Left"  Margin="88,-264,0,0"
VerticalAlignment="Top"  Height="61"  Width="108"
Click="Button_Click_2"/>
            <Label  Content="ID"
HorizontalAlignment="Left"  Margin="-377,-179,0,0"
VerticalAlignment="Top"/>
            <TextBox  x:Name="SearchTxt"
HorizontalAlignment="Left"  Margin="-292,-175,0,0"
TextWrapping="Wrap"  VerticalAlignment="Top"
Width="120"/>


        </Grid>
    </Window>



using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
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 CarsWPF
{
    /// <summary>
    /// Логика взаимодействия для Car.xaml
    /// </summary>
    public partial class Car : Window
    {
        SqlConnection con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\source\repos\CarsWPF\CarsWPF\Cars.mdf;Integrated Security=True");

        public Car()
        {
            InitializeComponent();
            LoadGrid();
        }

        public void LoadGrid()
        {
            SqlCommand cmd = new SqlCommand("select * from Car", con);
            DataTable dt = new DataTable();
            con.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            dt.Load(reader);
            con.Close();
            dataGrid.ItemsSource = dt.DefaultView;



        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            SqlCommand cmd = new SqlCommand("INSERT INTO Car VALUES (@Mark, @Model, @Year, @Status)", con);
            cmd.CommandType = CommandType.Text;

cmd.Parameters.AddWithValue("@Mark", MarkTxt.Text);

cmd.Parameters.AddWithValue("@Model", ModelTxt.Text); ;

cmd.Parameters.AddWithValue("@Year", YearTxt.Text);

cmd.Parameters.AddWithValue("@Status", StatusTxt.Text);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            LoadGrid();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            SqlCommand cmd = new SqlCommand("DELETE FROM Car WHERE CarID = " + SearchTxt.Text + " ", con);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            LoadGrid();
        }

        private void Button_Click_2(object sender, RoutedEventArgs e)
        {
            SqlCommand cmd = new SqlCommand(string.Format(
                "update Car set Mark = '{0}', Model = '{1}', Year = '{2}', Status = '{3}' where CarID = {4}",
                MarkTxt.Text, ModelTxt.Text, YearTxt.Text, StatusTxt.Text, SearchTxt.Text), con);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            LoadGrid();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
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 CarsWPF
{
  /// <summary>
  /// Логика взаимодействия для Clients.xaml
  /// </summary>
  public partial class Clients : Window
  {
    SqlConnection con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\source\repos\CarsWPF\CarsWPF\Cars.mdf;Integrated Security=True");
    public Clients()
    {
      InitializeComponent();
      LoadGrid();
    }

    private void Window_Loaded(object sender, RoutedEventArgs e)
    {

    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
      SqlCommand cmd = new SqlCommand("DELETE FROM Client WHERE ClientID = " + SearchTxt.Text + " ", con);
      con.Open();
      cmd.ExecuteNonQuery();
      con.Close();
      LoadGrid();
    }

    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
      SqlCommand cmd = new SqlCommand("INSERT INTO Client VALUES (@Name, @Email, @Phone, @Address)", con);
      cmd.CommandType = CommandType.Text;

cmd.Parameters.AddWithValue("@Name", NameTxt.Text);

cmd.Parameters.AddWithValue("@Email", EmailTxt.Text);

cmd.Parameters.AddWithValue("@Phone", PhoneTxt.Text);

cmd.Parameters.AddWithValue("@Address", AddressTxt.Text);
      con.Open();
      cmd.ExecuteNonQuery();
      con.Close();
      LoadGrid();

    }

    public void LoadGrid()
    {
      SqlCommand cmd = new SqlCommand("select * from Client", con);
      DataTable dt = new DataTable();
      con.Open();
      SqlDataReader reader = cmd.ExecuteReader();
      dt.Load(reader);
      con.Close();
      dataGrid.ItemsSource = dt.DefaultView;



    }

    private void Button_Click_2(object sender, RoutedEventArgs e)
    {
      SqlCommand cmd = new SqlCommand(string.Format(
        "update Client set Name = '{0}', Email = '{1}', Phone = '{2}', Address = '{3}' where ClientID = {4}",
        NameTxt.Text, EmailTxt.Text, PhoneTxt.Text, AddressTxt.Text, SearchTxt), con);
      con.Open();
      cmd.ExecuteNonQuery();
      con.Close();
      LoadGrid();
    }
  }
}

using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
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;
using System.Runtime.Remoting.Contexts;

namespace CarsWPF
{
    /// <summary>
    /// Логика взаимодействия для Inspections.xaml
    /// </summary>
    public partial class Inspections : Window
    {

        SqlConnection con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\source\repos\CarsWPF\CarsWPF\Cars.mdf;Integrated Security=True");

        public Inspections()
        {
            InitializeComponent();
            LoadGrid();
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            SqlCommand cmd = new SqlCommand(
                "INSERT INTO Inspection VALUES (@CarID, @Date, @Result)", con);
            cmd.CommandType = CommandType.Text;

cmd.Parameters.AddWithValue("@CarID", CarTxt.Text);

cmd.Parameters.AddWithValue("@Date", DateTxt.Text);

cmd.Parameters.AddWithValue("@Result", ResultTxt.Text);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            LoadGrid();
        }

        public void LoadGrid()
        {
            SqlCommand cmd = new SqlCommand("select * from Inspection", con);
            DataTable dt = new DataTable();
            con.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            dt.Load(reader);
            con.Close();
            dataGrid.ItemsSource = dt.DefaultView;
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            SqlCommand cmd = new SqlCommand("DELETE FROM Inspection WHERE InspectionID = " + SearchTxt.Text + " ", con);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            LoadGrid();
        }

        private void Button_Click_2(object sender, RoutedEventArgs e)
        {
            SqlCommand cmd = new SqlCommand(string.Format(
                "update Inspection set CarID = {0}, Date = '{1}', Result = '{2}' where InspectionID = {3}",
                CarTxt.Text, CarTxt.Text, DateTxt.Text, ResultTxt.Text, SearchTxt.Text), con);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            LoadGrid();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
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.Navigation;
using System.Windows.Shapes;

namespace CarsWPF
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
```

```
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            Clients clients = new Clients();
            this.Hide();
            clients.ShowDialog();
            this.Show();
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            Car cars = new Car();
            this.Hide();
            cars.ShowDialog();
            this.Show();
        }

        private void Button_Click_2(object sender, RoutedEventArgs e)
        {
            Rent rent = new Rent();
            this.Hide();
            rent.ShowDialog();
            this.Show();
        }

        private void Button_Click_3(object sender, RoutedEventArgs e)
        {
            Reservations reservations = new Reservations();
            this.Hide();
            reservations.ShowDialog();
            this.Show();

        }

        private void Button_Click_4(object sender, RoutedEventArgs e)
        {
            Inspections inspections = new Inspections();
            this.Hide();
            inspections.ShowDialog();
            this.Show();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
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 CarsWPF
{
    /// <summary>
    /// Логика взаимодействия для Rent.xaml
    /// </summary>
    public partial class Rent : Window
    {
        SqlConnection con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\source\repos\CarsWPF\CarsWPF\Cars.mdf;Integrated Security=True");

        public Rent()
        {
            InitializeComponent();
            LoadGrid();
        }

        public void LoadGrid()
        {
            SqlCommand cmd = new SqlCommand("select * from Rent", con);
            DataTable dt = new DataTable();
            con.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            dt.Load(reader);
            con.Close();
            dataGrid.ItemsSource = dt.DefaultView;
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            SqlCommand cmd = new SqlCommand("INSERT INTO Rent VALUES (@CarID, @ClientID, @StartDate, @EndDate, @TotalCost)", con);
            cmd.CommandType = CommandType.Text;

cmd.Parameters.AddWithValue("@CarID", CarTxt.Text);
```

```
cmd.Parameters.AddWithValue("@ClientID",
ClientTxt.Text);

cmd.Parameters.AddWithValue("@StartDate",
StartTxt.Text);

cmd.Parameters.AddWithValue("@EndDate",
EndTxt.Text);

cmd.Parameters.AddWithValue("@TotalCost",
CostTxt.Text);
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
                LoadGrid();
            }

            private void Button_Click(object sender,
RoutedEventArgs e)
            {
                SqlCommand cmd = new
SqlCommand("DELETE FROM Rent WHERE RentID
= " + SearchTxt.Text + " ", con);
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
                LoadGrid();
            }

            private void Button_Click_2(object
sender, RoutedEventArgs e)
            {
                SqlCommand cmd = new
SqlCommand(string.Format(
                    "update Rent set CarID = {0},
ClientID = {1}, StartDate = '{2}', EndDate = '{3}',
TotalCost = {4} where RentID = {5}",
                    CarTxt.Text, ClientTxt.Text,
StartTxt.Text, EndTxt.Text,
CostTxt.Text,SearchTxt.Text), con);
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
                LoadGrid();
            }
        }
    }

    using System;
    using System.Collections.Generic;
    using System.Data.SqlClient;
    using System.Data;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;
    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;
    using System.Runtime.Remoting.Contexts;

    namespace CarsWPF
    {
        /// <summary>
        /// Логика взаимодействия для
Reservations.xaml
        /// </summary>
        public partial class Reservations : Window
        {
            SqlConnection con = new
SqlConnection(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilenam
e=D:\source\repos\CarsWPF\CarsWPF\Cars.mdf;Integr
ated Security=True");

            public Reservations()
            {
                InitializeComponent();
            }

            public void LoadGrid()
            {
                SqlCommand cmd = new
SqlCommand("select * from Reservation", con);
                DataTable dt = new DataTable();
                con.Open();
                SqlDataReader reader =
cmd.ExecuteReader();
                dt.Load(reader);
                con.Close();
                dataGrid.ItemsSource =
dt.DefaultView;
            }

            private void Button_Click_1(object
sender, RoutedEventArgs e)
            {
                SqlCommand cmd = new
SqlCommand("INSERT INTO Reservation VALUES
(@CarID, @ClientID, @StartDate, @EndDate)", con);
                cmd.CommandType =
CommandType.Text;

cmd.Parameters.AddWithValue("@CarID",
CarTxt.Text);

cmd.Parameters.AddWithValue("@ClientID",
ClientTxt.Text);

cmd.Parameters.AddWithValue("@StartDate",
StartTxt.Text);
```

```
cmd.Parameters.AddWithValue("@EndDate",
EndTxt.Text);
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
                LoadGrid();
            }

            private void Button_Click(object sender,
RoutedEventArgs e)
            {
                SqlCommand cmd = new
SqlCommand("DELETE FROM Reservation WHERE
ReservationID = " + SearchTxt.Text + " ", con);
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
                LoadGrid();
            }

            private void Button_Click_2(object
sender, RoutedEventArgs e)
            {
                SqlCommand cmd = new
SqlCommand(string.Format(
                    "update Rent set Reservation = {0},
ClientID = {1}, StartDate = '{2}', EndDate = '{3}' where
ReservationID = {4}",
                    CarTxt.Text, ClientTxt.Text,
StartTxt.Text, EndTxt.Text, SearchTxt.Text), con);
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
                LoadGrid();
            }
        }
    }
```