НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»
спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування автомобілів

Виконав студент IV курсу, групи ІТ-82
(шифр групи)
Богаченко Сергій Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент каф.ІПІ, к.т.н., Новінський В.П _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент каф.ІПІ, к.т.н., Ліщук К.І _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент каф ІСТ, к.т.н. Попенко В.Д 
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2022

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії
Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 121 Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютеризованих систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Богаченку Сергію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування автомобілів

керівник проєкту доцент, к.т.н., доц., Новінський В.П
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»квітня 2022 р. №_____

2. Термін подання студентом проєкту « 19 » червня 2022 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до праграмного забезпечення: змістовий опис, загальні положення і аналіз предметної області, опис процесу діяльності, огляд існючих технічних рішень та аналіз вимог до програмного забезпечення.

2) Моделювання та конструювання програмного забезпечення: архітектура програми, моделювання та аналіз програмного забезпечення.

3) Аналіз якості та тестування програмного забезпечення: опис процесів тестування, аналіз якості ПЗ та опис контрольного прикладу.

4) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна діяльності роботи програмного забезпечення
2) Схема бізнес-процесів роботи програмного забезпечення
3) Схема бази даних

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2022 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2022	
2	Аналіз існуючих методів розв'язання задачі	10.03.2022	
3	Постановка та формалізація задачі	19.03.2022	
4	Розробка інформаційного забезпечення	27.03.2022	
5	Алгоритмізація задачі	8.04.2022	
6	Обґрунтування вибору використаних технічних засобів	18.04.2022	
7	Розробка програмного забезпечення	24.04.2022	
8	Налагодження програми	15.05.2022	
9	Виконання графічних документів	30.05.2022	
10	Оформлення пояснювальної записки	30.05.2022	
11	Подання ДП на попередній захист	06.06.2022	
12	Подання ДП рецензенту	12.06.2022	
13	Подання ДП на основний захист	19.06.2022	

Студент _____ Сергій БОГАЧЕНКО
(підпис) (ініціали, прізвище)

Керівник _____ Валерій НОВІНСЬКИЙ
(підпис) (ініціали, прізвище)

Анотація

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 21 таблиць, 4 рисунків та 10 джерел – загалом 56 сторінок.

Дипломний проєкт присвячений розробці програмного забезпечення для управління процесами замовлень клієнтів на станції технічного обслуговування.

Метою цієї роботи є підвищення та покращення роботи користувача при роботі на станції технічного обслуговуванні під час роботи з замовленнями клієнтів.

У першому розділі наведені загальні положення, опис та аналіз предметної області, головна мета, цілі, функціональні і нефункціональні вимоги та призначення.

У другому розділі описане моделювання, розроблена архітектура та компоненти. Також наведений опис модулів, функцій та класів.

У третьому розділі описано процес тестування, наведені приклади тестів та проаналізована якість програмного забезпечення.

У четвертому розділі неведений опис та розгортання програми.

Результати роботи над проєктом є програмне забезпечення, що допомагає керувати процесами виконання замовлень клієнтів на станції технічного обслуговування.

КЕРУВАННЯ ПРОЦЕСАМИ, УПРАВЛІННЯ, ЗАМОВЛЕННЯ.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 21 tables, 4 figures and 10 sources - a total of 56 pages.

The diploma project is devoted to the development of software for managing customer order processes at the service station.

The purpose of this work is to increase and improve the user's work when working at the service station while working with customer orders.

The first section provides general provisions, description and analysis of the subject area, the main purpose, objectives, functional and non-functional requirements and purposes.

The second section describes modeling, developed architecture and components. There is also a description of modules, functions and classes.

The third section describes the testing process, gives examples of tests and analyzes the quality of the software.

The fourth section does not describe and deploy the program.

The results of the project are software that helps manage the process of fulfilling customer orders at the service station.

PROCESS MANAGEMENT, MANAGEMENT, ORDERS.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-8206.045490.02.81	Пояснювальна записка	56	
3	А4	КПІ.ІТ-8206.045490.03.91	Технічне завдання	10	
4	А4	КПІ.ІТ-8206.045490.04.51	Програма та методика тестування	6	
5	А4	КПІ.ІТ-8206.045490.05.34	Керівництво користувача	9	
6	А4	КПІ.ІТ-8206.045490.06.13	Опис програми	27	
7	А4	КПІ.ІТ-8206.045490.07.99ССД	Схема структурна діяльності	1	
8	А4	КПІ.ІТ-8206.045490.07.99СБП	Схема бізнес-процесів	1	
9	А3	КПІ.ІТ-8206.045490.07.99СБД	Схема бази даних	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-8206.045490.00.90		
Розроб.		Богаченко С.В.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Новінський В.П.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-82	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЦЕСАМИ ВИКОНАННЯ ЗАМОВЛЕНЬ КЛІЄНТІВ НА СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Технічне завдання

КПІ.ІТ-8206.045490.03.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Валерій НОВІНСЬКИЙ

Нормоконтроль:
_____ Валерій НОВІНСЬКИЙ

Виконавець:
_____ Сергій БОГАЧЕНКО

Київ – 2022

Зміст

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування автомобілів.

Галузь застосування: управління процесами виконання замовлення клієнтів на СТО на комп'ютері.

Наведене технічне завдання поширюється на розробку програмного забезпечення додатку для управління процесами виконання замовлень клієнтів КПІ.ІТ-8206.045490.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення управління процесами виконання замовлень клієнтів на станції технічного обслуговування є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії і управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для управління процесами виконання замовлень клієнтів на СТО.

Метою розробки є підвищення ефективності роботи користувача при управлінні виконання замовлень клієнтів на СТО, за допомогою сучасних технологій взаємодії з базою даних та зручного та простого графічного інтерфейсу користувача.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

– авторизація користувача;
– додавання нового замовлення;
– додавання, змінення, видалення клієнтів;
– додавання, змінення, видалення автомобілів;
– додавання, змінення, видалення послуг;
– змінення існуючого замовлення;
– видалення замовлення;
– перегляд всіх замовлень.

4.1.1.2 Для адміністратора системи:

– авторизація адміністратора;
– додавання, змінення, видалення користувачів;
– додавання, змінення, видалення запчастин;
– додавання, змінення, видалення послуг;
– перегляд всіх замовлень.

Розробку виконати на операційній системі Windows 8 і вище.

4.2 Вимоги до надійності:

- передбачити перевірку на коректність введених даних;
- передбачити перевірку на пусті поля вводу;
- передбачити зрозумілість та доступність інтерфейсу для користувача.

4.3 Умови експлуатації

Не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Наявність вільної пам’ять на пристрої для встановлення та функціювання програми.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Операційна система на якій буде запускатися програма повинна бути Windows 7 і новіше.

4.5.2 Для функціювання додатку на пристрої повинен бути встановлений локальний SQL Server 2013 року і новіше.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему.

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 50 аркушах формату А4 (без додатків 5.3.2 - 5.3.6):

- технічне завдання;
- керівництво користувача;
- програма та методика тестування.

5.4 Графічна частина повинна бути виконана на аркуші формату А3 та А4, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структура діяльності процесу роботи програмного забезпечення.

5.4.2 Схема бізнес-процесу роботи програмного забезпечення.

5.4.3 Схема бази даних.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	22.02.2022	
2.	Аналіз інуючих методів розв'язання задачі	22.02.2022	
3.	Постановка та формалізація задачі	01.03.2022	Специфікації програмного забезпечення
4.	Аналіз вимог до програмного забезпечення	05.03.2022	Схема матриці трасування
5.	Алгоритмізація задачі	15.03.2022	Схема структурна діяльності роботи програмного забезпечення
6.	Моделювання програмного забезпечення	20.03.2022	Схема бізнес-процесу роботи програмного забезпечення
7.	Обґрунтування використовуваних технічних засобів	25.03.2022	
8.	Розробка архітектури програмного забезпечення	05.04.2022	Схема архітектурних компонентів
9.	Розробка програмного забезпечення	10.04.2022	Технічна документація
10.	Налагодження програми	15.05.2022	Програма та методика тестування
11.	Виконання графічних документів	20.05.2022	Графічний матеріал проєкту
12.	Оформлення пояснювальної записки	20.05.2022	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Пояснювальна записка
до дипломного проєкту

на тему: програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування автомобілів

КПІ.ІТ-8206.045490.02.81

Київ – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ

WPF (Windows Presentation Foundation) – є частиною платформи .NET і є розробкою власне графічного інтерфейсу. Характерна риса технології - використання розмітки XAML. Тож можна створювати насичений графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу або написаний код на мовах C#, VB.NET і F#, або поєднувати їх між собою. Також можна зазначити, що ця технологія походить від старшого варіанта реалізації інтерфейса – Windows Form.

EF (Entity Framework) – це інструмент, який є рішенням задач у роботі з базами даних в усіх проєктах, де використовуються мови сімейства .NET. Завдяки йому є можливість взаємодії з СУБД за допомогою використання сутностей (entity), а не таблиць. Також можна зазначити, що написання коду таким методом відбувається набагато швидше у порівнянні з класичним методом. Ще однією перевагою є те, що розробнику не потрібно думати про підключення до мережі, підготовку SQL і параметрів, відправку запитів і транзакцій. EF робить всі ці задачі автоматично, а розробник працює з сутностями та лише вказує, коли треба зберегти всі зміни.

.NET – це технологія, яка підтримує створення та виконання веб-служб та програм Windows. Перевагою даної технології є те, що можна поєднувати між собою проєкти які написані на різних мовах та використовувати їх як одне ціле.

БД- база даних, ПЗ – програмне забезпечення.

ВСТУП

На сьогодні у світі широко використовуються комп’ютерні технології для покращення, спрощення та пришвидшення управлінь певними виробничими процесами, які використовуються у всіх сферах побуту, адже кожен користувач прагне досягти максимальної ефективності та прибутку від власної справи. У цьому проєкті ми розглянемо випадок ефективного використання комп’ютерних технологій на прикладі станції технічного обслуговування. Виходячи з цього, тема даного дипломного проєкту є актуальною.

У кожному місті чи невеликому селищі є автомобілі, котрі є необхідністю для задоволення особистих потреб. На жаль, їх потрібно не лише заправляти пальним, але й технічно підтримувати. Як тільки з’являється питання щодо надання ТО для авто, водій починає думати, де це можна зробити – очевидно, що на станції технічного обслуговування. СТО можуть бути як і великими підприємствами, так і створеними приватними особами. Звичайно, якщо мова йде про успішні станції, то вони можуть з легкістю оформлювати клієнтів, їх автомобілі, використовуючи ліцензовані програмні забезпечення. Проте, якщо ми будемо розглядати невелику та маловідому станцію технічного обслуговування, то тут постає питання де знайти програмне забезпечення, котре зможе оформлювати та пришвидшувати виконання замовлення клієнтів.

Сучасні варіанти рішень цієї задачі є доволі ефективними та продуктивними, але мають певні недоліки. Одним із таких, у першу чергу, є ціна, тому що далеко не всі малі бізнеси можуть собі дозволити придбати програмне забезпечення. Другим недоліком є недоступність останнього.

Метою цієї роботи є надання доступного та безкоштовного програмного забезпечення для управління процесами виконання замовлення клієнтів на станції технічного обслуговування.

Практичним застосуванням даної програми можуть скористатися малі бізнеси, які тільки почали працювати і не мають можливості придбати ліцензійне програмне забезпечення.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

На даному етапі розвитку стає більш актуальним використання програмного забезпечення для управління процесами на станціях технічного обслуговування. Найпростішим способом реалізації такого ПЗ є локальний. Більш складною реалізацією буде використання онлайн серверів задля регулювання процесів.

Проте, обидва варіанти здатні вирішити поставлену задачу – управління процесами виконання замовлень клієнтів. Перший варіант реалізації є прикладом використання ПЗ для невеликих станцій технічного обслуговування, адже зв’язок із сервером є непотрібним, тому що всі дані зберігаються локально. Другий варіант реалізації є прикладом використання мережевих станцій технічного обслуговування. Тут уже потрібен зв’язок із сервером для регулювання інтенсивності потоку клієнтів, і всі дані повинні синхронізуватися на сервері.

1.2 Змістовий опис і аналіз предметної області

Технологія WPF (Windows Presentation Foundation) є частиною платформи .NET і є розробкою власне графічного інтерфейсу. Характерна риса технології - використання розмітки XAML. Тож можна створювати насичений графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу або написаний код на мовах C#, VB.NET і F#, або поєднувати їх між собою. Також можна зазначити, що ця технологія походить від старшого варіанта реалізації інтерфейса – Windows Form.

Чому вибір зупинився саме на цьому варіанті реалізації графічного інтерфейсу? По-перше, він дає можливість використання мови C#, як основної мови для написання логіки всієї програми та використання мови розмітки

XAML. По-друге, це простий та зрозумілий інтерфейс конструювання для побудови графічного представлення програми.

Entity Framework (EF) – це інструмент, який є рішенням задач у роботі з базами даних в усіх проєктах, де використовуються мови сімейства .NET. Завдяки йому є можливість взаємодії з СУБД за допомогою використання сутностей (entity), а не таблиць. Також можна зазначити, що написання коду таким методом відбувається набагато швидше у порівнянні з класичним методом. Ще однією перевагою є те, що розробнику не потрібно думати про підключення до мережі, підготовку SQL і параметрів, відправку запитів і транзакцій. EF робить всі ці задачі автоматично, а розробник працює з сутностями та лише вказує, коли треба зберегти всі зміни.

Існує 3 різні підходи моделювання в EF:

- Code – First.
- Model – First.
- Database – First.

Code – First – як видно з назви, ця реалізація передбачає створення бази даних на основі вже готового коду (додатку). Цю модель, як правило, описують за допомогою декількох класів, і кодом, який зв’язує ці класи між собою. Зазвичай цей варіант використовують, коли маленький додаток розширяється і зберігати невелику кількість даних стає неможливим. Розробнику не потрібно думати як база даних буде створена на основі його класів, адже EF робить все автоматично.

Model – First – цей підхід використовують розробники, коли не хочуть використовувати інструменти СУБД для створення та управління базами даних і також не мають бажання самотужки налаштовувати Entity Data Model (EDM). Цей спосіб моделювання є найбільш простим для EF. Робочий процес починається з того, що розробник моделює базу даних за допомогою елементарних знань SQL.

Database – First – підхід, який дозволяє написати додаток вже для існуючої бази даних. Бази даних в реальних додатках стають з часом доволі складними, а також значно збільшується час для розробки їх моделі, яку могли б зрозуміти розробники. Цей підхід є протилежним до підходу Model – First, оскільки база даних вже існує: розробник повинен знати, де знаходиться база даних та її ім’я. Проте, розробник не повинен розуміти внутрішню роботу бази даних, тому що EF все ще виконує це все автоматично. Тож перед початком роботи цього підходу треба створити базу даних. Після цього згенерувати відповідні класи. Власне, далі робота схожа на два попередні підходи.
Порівняння необхідних умов для використання підходів наведені в таблиці 1.1

Таблиця 1.1 – Порівняння необхідних умов підходів

	Code - First	Model - First	Database - First
Потрібна готова база даних?	Ні	Ні	Так
Потрібно моделювати в графічному інтерфейсі?	Ні	Так	Ні
Потрібні класи (моделі) перед підключенням Entity Framework	Так	Так	Ні

Отже, в результаті порівняння можна сказати що всі 3 варіанта є досить ефективними. Проте, область їх використання є різною залежно від поставленого завдання.

1.3 Опис процесу діяльності

Першим етапом процесу є авторизація користувача. Юзером може бути або механіком або адміністратором.

Після успішного входу в систему, згідно з, типом авторизованого користувача - надається відповідний функціонал.

Функціонал механіка:

- додати нового клієнта;
- додати машину клієнта;
- створити нове замовлення;
- зареєструвати замовлення;
- подивитися деталі замовлення;
- редагувати інформацію замовлення, клієнтів, їх автомобілів.

Функціонал адміністратора:

- весь функціонал механіка;
- додати нового користувача;
- додати нові запчастини;
- додати нові види послуг;
- редагувати всю інформацію.

Для того, щоб зобразити діяльність процесу механіка та адміністратора будуть наведені схеми які зображені на рисунках 1.2 та 1.3

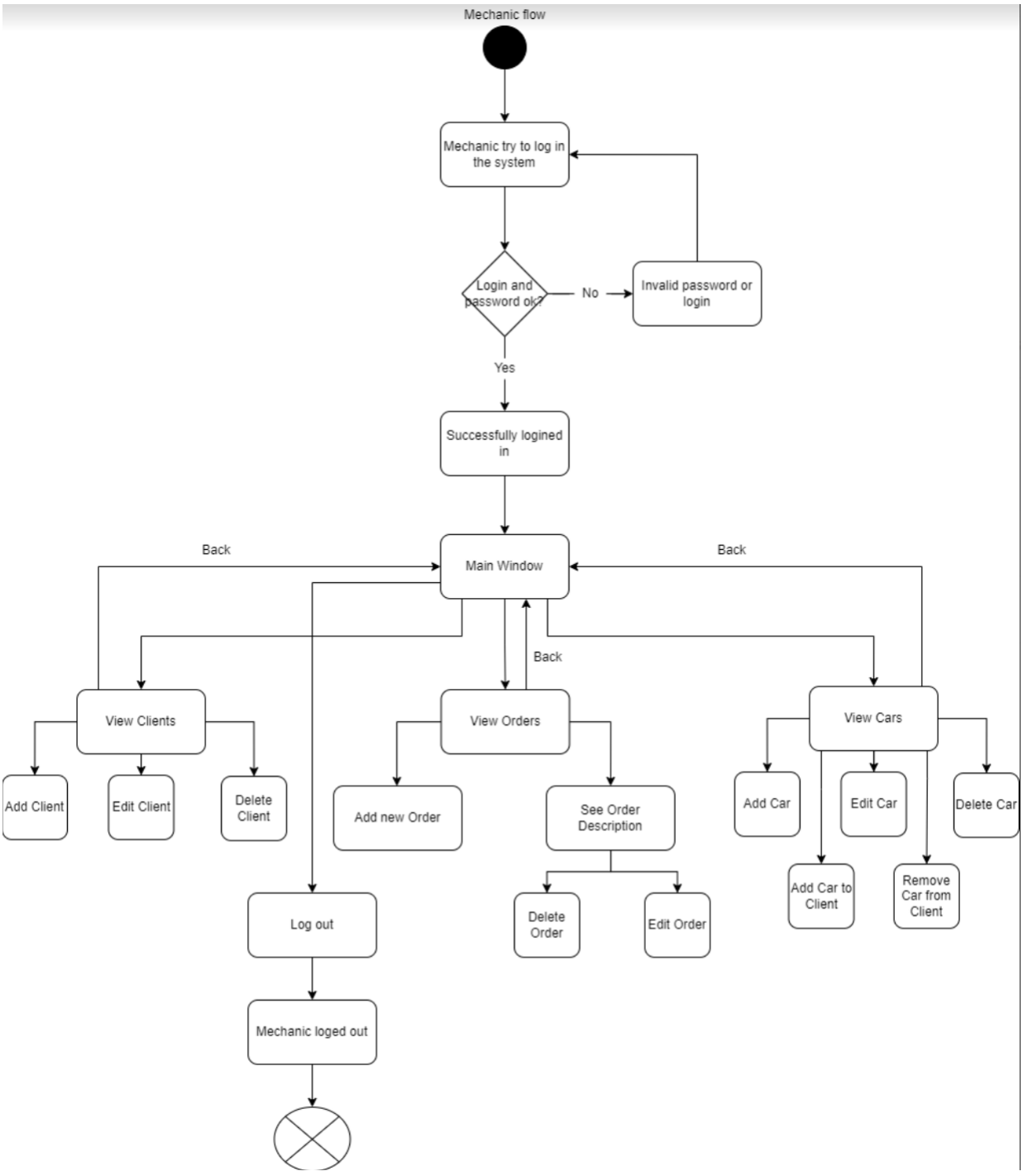


Рисунок 1.2 - Схема структурна діяльності процесу механіка

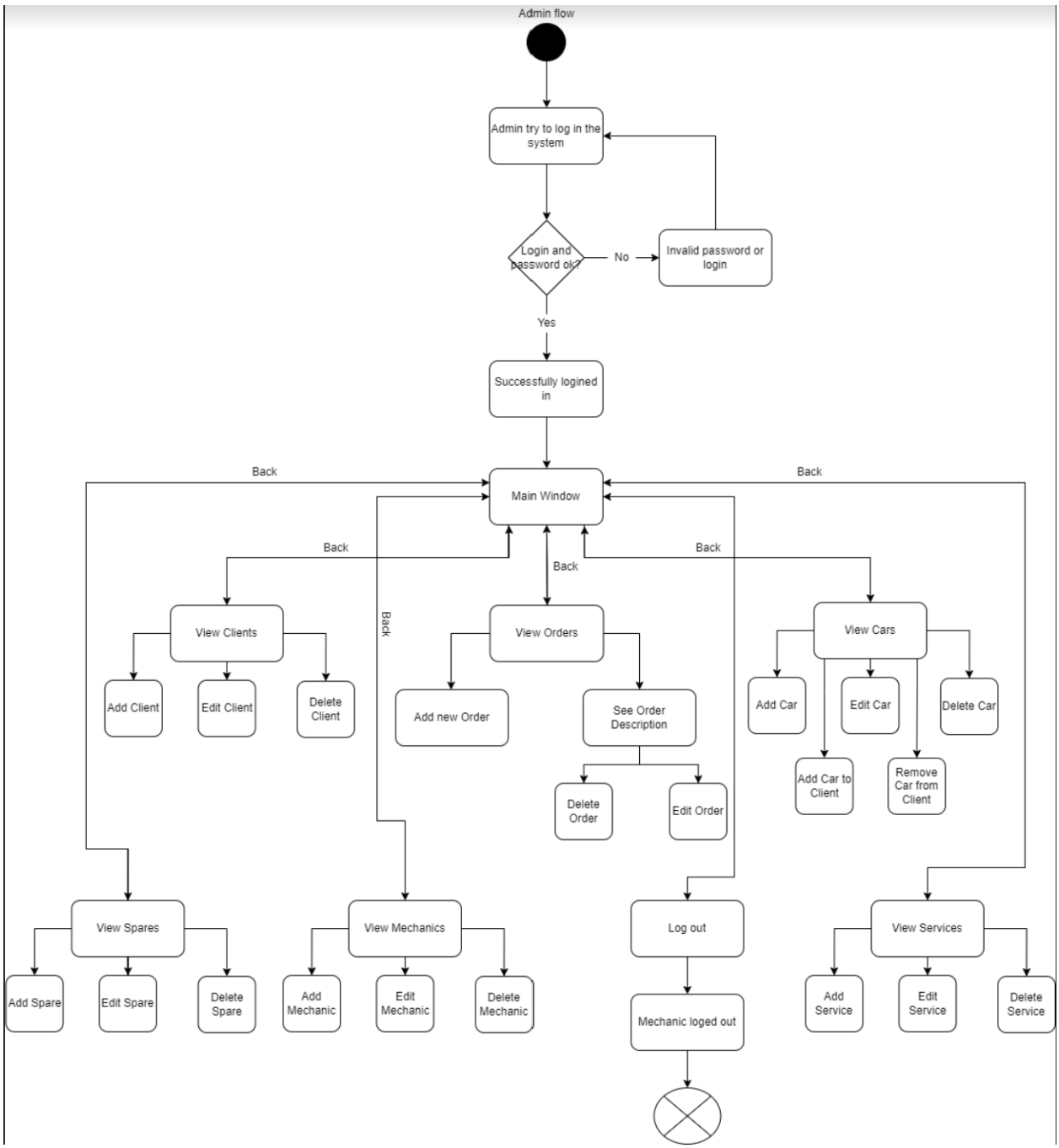


Рисунок 1.3 - Схема структурна діяльності процесу адміністратора

З даних схем видно, що адміністратор має весь функціонал механіка, але має додаткові можливості, що є логічним, адже робітник повинен мати доступ лише до тих речей які йому потрібні і які він може контролювати.

1.4 Аналіз аналогічних рішень ПЗ

Технологічне рішення, які б надавали можливість управління процесами за допомогою графічного інтерфейсу та працюючи з базами даних не існує. Проте є технології за допомогою яких можна вирішити ці поставлені задачі:

- Entity Framework – бібліотека яка була створена для вирішення задачі роботи з базами даних в проєктах на платформі .NET. Використовувати цю технологію можна в будь якому проєкті, де використовується мова C#.

- Windows Form – інтерфейс програмування додатків, який відповідає за графічний інтерфейс користувача. Також входить до .NET. Цей інтерфейс надає доступ до елементів Microsoft Windows. Є простим рішенням для нескладних програм.

- *WPF* – є частиною платформи .NET і є розробкою власне графічного інтерфейсу. Характерна риса технології - використання розмітки XAML. Тож можна створювати насичений графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу або написаний код на мовах C#, VB.NET і F#, або поєднувати їх між собою. Також можна зазначити, що ця технологія походить від старшого варіанта реалізації інтерфейса – Windows Form. Завдяки WPF можна створювати двовимірні та тривимірні інтерфейси.

1.5 Аналіз відомих програмних продуктів

Звичайно є велика кількість готових продуктів, які надають можливості управління процесами замовлень на СТО. Прикладами таких програм є:

- Мегаплан – універсальна програма, яка підходить для різного бізнесу, включаючи автосалони. У системі є зручна клієнтська база, контроль співробітників, історія взаємодій, вирва продажів, інтеграція зі сторонніми сервісами. Універсальні рішення зазвичай підкуповують кількістю функцій, але, якщо частина їх у результаті не використовується, доводиться переплачувати.

- Автодилер - програма для автосервісу та СТО де весь її функціонал заточений під потреби саме цієї галузі. На вигляд додаток більше нагадує таблиці в Excel, але це не скасовує її функціональності.

- STOCRM – ще одна галузевий продукт для СТО. По функціоналу схожа на ПЗ «Автодилер», але з більш сучасним та приємним дизайном.

- Splus - сучасна галузева програма. Існує все необхідне для СТО наряд-замовлення, інтеграція з постачальниками, підключення онлайн-каси. Має перевагу над конкурентами тим, що ціна залежить від кількості працівників.

Тож проаналізувавши вище вказаних готових продуктів можна зазначити переваги та недоліки існуючих рішень. Деякі з них є універсальними рішеннями для управління певними процесам, інші виключно розроблені для використання на СТО. Майже всі наведені вище програми є платними. Ті рішення які безкоштовні – є універсальними додатками, тому зазвичай функціоналу можу не вистачати. Тож можна зробити висновок, що створення безкоштовного продукту, який міг би вирішити задачу управління процесами замовлень клієнтів на СТО є досить актуальним.

1.6 Аналіз вимог до програмного забезпечення

1.6.1 Розроблення функціональних вимог

Діаграма варіантів використання представлена на рисунку 1.4. В таблиці 1.5 наведений опис функціональних вимог.

ë

Рис 1.4 – Схема варіантів використання

Таблиця 1.5 – вимоги функціоналу та їх пріоритети

Варіант Використання	Вимога в функціоналі	Пріоритет	Виконавець
Phone to describe problems and possible booking.	Мобільний телефон	Низький	Клієнт, Механік
Phone and book TO	Мобільний телефон	Середній	Клієнт, Механік
Arrive to STO without booking	-	Низький	Клієнт
View Clients	1. Програма має надавати можливість перегляду інформації про клієнтів: редагування, видалення та додавання. 1.1 Додаток має надавати можливість прив'язку та відв'язку машин від клієнта.	Високий	Механік, Адміністратор
View Orders	2. Програма має надавати можливість перегляду всіх замовлень. 2.1 Додаток має надавати можливість додавати, змінювати та видаляти замовлення.	Високий	Механік, Адміністратор

Продовження таблиці 1.5

View Cars	3. Програма має надавати можливість перегляду всіх зареєстрованих автомобілів. 3.1 Додаток має надавати можливість додавати, змінювати автомобілі. 3.2 Програма повинна надавати можливість прив’язувати та відв’язувати автомобіль від клієнта.	Високий	Механік, Адміністратор
View Mechanics	4. Програма має надавати можливість перегляду всіх Механіків. 4.1 Додаток має надавати можливість додавати, змінювати та видаляти механіків.	Середній	Адміністратор
View Spares	5. Програма має надавати можливість перегляду всіх запчастин. 5.1 Додаток має надавати можливість додавати, змінювати та видаляти запчастини.	Середній	Адміністратор

Продовження таблиці 1.5

View Services	6. Програма має надавати можливість перегляду всіх видів роботи. 6.1 Додаток має надавати можливість додавати, змінювати та видаляти послуги.	Середній	Адміністратор
Register new Client	7. Програма має надавати можливість реєстрування нового клієнта та його автомобіля.	Високий	Механік, Адміністратор

1.6.2 Розроблення нефункціональних вимог

Далі нижче будуть наведені наступні нефункціональні вимоги:

- операційна система на якій може функціонувати продукт є Windows 7 та новіше.
- має бути встановлений локальний сервер SQL для роботи бази даних;
- інтерфейс має бути приємним, зрозумілим та достатньо інформативним для повноцінного використання;
- захист користувача від некоректних дій.

1.6.1 Постановка комплексу завдань модулю

Призначенням розробки є програмне забезпечення, яке надає можливість управління процесами виконання замовлень клієнтів на СТО(створювати, змінювати та видаляти замовлення).

Метою розробки є підвищення ефективності роботи працівника та спрощення процесів управління виконання замовлень за допомогою простого та зрозумілого інтерфейсу користувача.

Для досягнення поставлених цілей треба вирішити такі задачі:

- визначити елементи інтерфейсу що погіршують вигляд та розуміння програми;
- розробити оптимальний спосіб збереження та діставання інформації з бази даних;
- визначити оптимальний метод входу користувачів до програмного забезпечення для прискорення цього процесу;
- розробити модуль інтерфейсу користувача для взаємодії з користувачем;
- створення функцію додавання, змінення та видалення замовлення;
- розробити модуль взаємодії з базою даних.

Висновки по розділу

В результаті проведення аналізу вимог до розробки ПЗ було проведено дослідження предметного середовища – Entity Framework. У даний час ця технологія широко використовується в додатках, де потрібно працювати з базами даних і де використовуються проєкти на базі .NET.

Було досліджені підходи до EF. Визначені особливості та необхідні умови для використання кожного з них.

Крім того були наведені схеми за допомогою яких було показано діяльність процесів механіка та адміністратора. Вони складаються з таких етапів:

- авторизація;
- робота с клієнтами;
- робота з замовленнями;
- робота з машинами;
- робота з послугами;
- робота з запчастинами;
- вихід з системи.

Також була наведена схема варіантів та використання для клієнта, механіка та адміністратора з якої можна виділити основні варіанти використання:

- перегляд, додавання, редагування та видалення замовлення;
- перегляд, додавання, редагування та видалення клієнта;
- перегляд, додавання, редагування та видалення автомобіля;
- перегляд, додавання, редагування та видалення послуг;
- перегляд, додавання, редагування та видалення запчастин;
- реєстрування нового замовлення;
- реєстрування нового клієнта з його автомобілем.

Проведено дослідження готових аналогічних програмних забезпечень (Мегаплан, Автодилер, STOCRM, Splus). Було виявлено, що майже всі

продукти є платними (Мегаплан, STOCRM, Splus). Також деякі з них не надають достатній функціонал(Мегаплан, Splus). Виходячи з цього було прийняте рішення створити безкоштовний та достатньо функціональний додаток котрий міг би допомогти в управлінні процесами виконання замовлень на СТО.

Сформовані головні завдання та задачі розробки програмного забезпечення: авторизуватися, виконувати потрібні дії яких потребує поставлена задача на момент використання програми, вихід з системи.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Основні процеси, які проходить користувач, включають в себе процес авторизації, виконання дій яких потребує ситуація, вихід з системи.

Послідовний опис процесу авторизації:

- користувач заповняє текстові поля логіну і пароля;
- натискає кнопку увійти. У разі успішної авторизації працівник переходить до головної сторінки додатку.

Послідовний опис процесу вибору дій:

- відповідно до типу користувача, працівник обирає той функціонал який йому потрібен залежно від ситуації;
- обирає одне з вікон де він продовжить виконувати потрібне йому завдання (наприклад: додавання нового клієнта з його автомобілем);
- продовжує працювати(переходити в потрібні йому вікна).

Послідовний опис процесу виходу з системи:

- коли користувач завершив роботу він має можливість вийти з додатка закривши головне вікно.

Схема бізнес-процесу роботи даного програмного забезпечення представлена у графічному матеріалі дипломного проекту.

2.2 Архітектура програмного забезпечення

Додаток складається з 2 основних модулів, кожні з яких виконують свою задачі:

- модуль роботи з базою даних;
- модуль інтерфейсу.

Для розробки програмного забезпечення була обрана мова C# для написання функціональних модулів. Проектування інтерфейсу було вирішено розробити використовуючи тип проєкту з сімейства .NET – Windows Presentation Foundation. Для роботи з базою даних було обрано використовувати технологію Entity Framework. Для створення бази даних було обрано середовище Management Studio 2017.

Проєктування БД є дуже важливим процесом, адже, якщо схема буде спроектовано некоректно то з великою вірогідністю подальша розробка програмного забезпечення буде невірною, що може спричинити затримки в часі розробки додатку. Тож для логічної та правильної побудови бази даних треба дотримуватись правил проєктування та правил побудови таблиць. Нижче буде наведена схема діаграми класів на рисунку 2.1 та опис структури даних в таблиці 2.2.

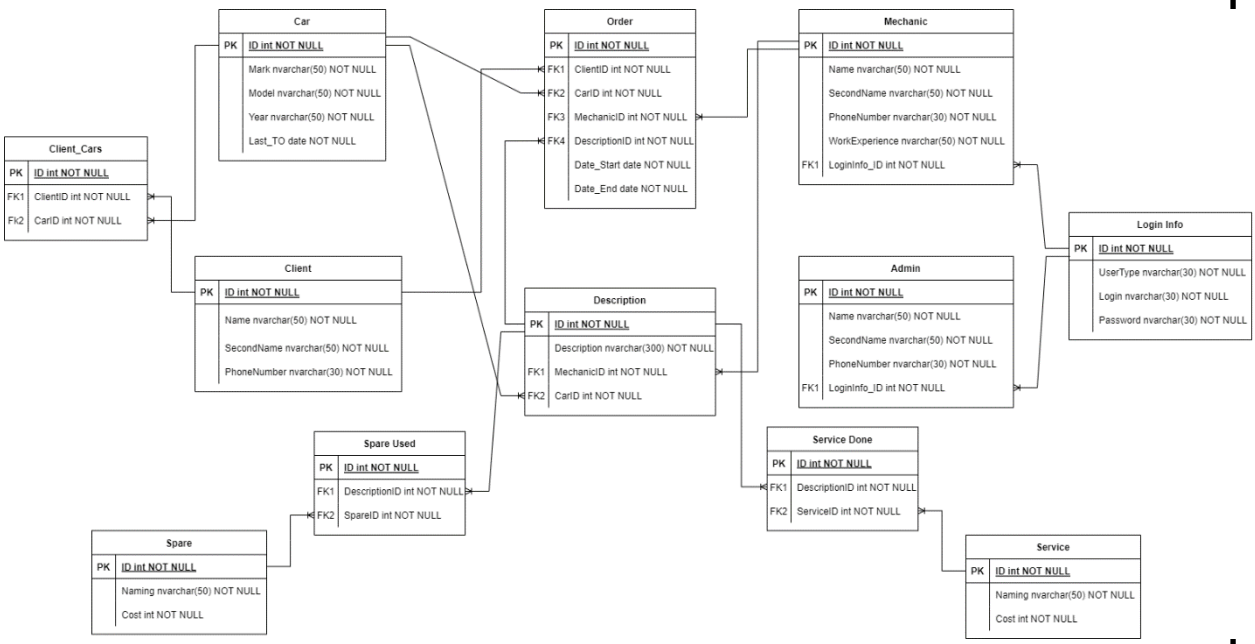


Рис 2.1 – Схема діаграми класів

Таблиця 2.2

Назва таблиці	Назва поля	Тип даних	Допустиме нульове значення	Спеціальні позначки
Order	ID	int	Ні	Primary Key

Продовження таблиці 2.2

	ClientID	int	Hi	Foreign Key
	CarID	int	Hi	Foreign Key
	MechanicID	int	Hi	Foreign Key
	DescriptionID	int	Hi	Foreign Key
	Date_Start	date	Hi	-
	Date_End	date	Hi	-
Car	ID	int	Hi	Primary Key
	Mark	nvarchar(50)	Hi	-
	Model	nvarchar(50)	Hi	-
	Year	nvarchar(50)	Hi	-
	Last_TO	date	Hi	-
Client	ID	int	Hi	Primary Key
	Name	nvarchar(50)	Hi	-
	SecondName	nvarchar(50)	Hi	-
	PhoneNumber	nvarchar(30)	Hi	-
Client_Cars	ID	int	Hi	Primary Key
	ClientID	int	Hi	Foreign Key
	CarID	int	Hi	Foreign Key
Mechanic	ID	int	Hi	Primary Key
	Name	nvarchar(50)	Hi	-
	SecondName	nvarchar(50)	Hi	-
	PhoneNumber	nvarchar(30)	Hi	-
	WorkExperience	nvarchar(50)	Hi	-
	LoginInfo_ID	int	Hi	Foreign Key
Admin	ID	int	Hi	Primary Key
	Name	nvarchar(50)	Hi	-
	SecondName	nvarchar(50)	Hi	-

Продовження таблиці 2.2

	PhoneNumber	nvarchar(30)	Hi	-
	LoginInfo_ID	int	Hi	Foreign Key
Login Info	ID	int	Hi	Primary Key
	UserType	nvarchar(30)	Hi	-
	Login	nvarchar(30)	Hi	-
	Password	nvarchar(30)	Hi	-
Description	ID	int	Hi	Primary Key
	Description	nvarchar(300)	Hi	-
	MechanicID	int	Hi	Foreign Key
	CarID	int	Hi	Foreign Key
Spare	ID	int	Hi	Primary Key
	Naming	nvarchar(50)	Hi	-
	Cost	int	Hi	-
Spare Used	ID	int	Hi	Primary Key
	DescriptionID	int	Hi	Foreign Key
	SpareID	int	Hi	Foreign Key
Service	ID	int	Hi	Primary Key
	Naming	nvarchar(50)	Hi	-
	Cost	int	Hi	-
Service Used	ID	int	Hi	Primary Key
	DescriptionID	int	Hi	Foreign Key
	ServiceID	int	Hi	Foreign Key

Схема створеної бази даних буде представлене у графічному матеріалі дипломного проєкту.

2.2.1 Опис архітектури програмного забезпечення

Програма включає в себе такі модулі: Database Manager Module, User Interface module. Кожен модуль відповідає за певні задачі та частини функціоналу. Нижче в таблиці 2.2 буде описаний кожен модуль та класи, які в нього входять.

Таблиця 2.2 – опис основних модулів програми

Назва Модуля	Назва Класу	Призначення та опис класу
Database Manager Module	DbWorker	Клас відповідає за взаємодію з базою даних. Додавання, редагування та видалення конкретних даних з БД.
User Interface Module	MainWindow	Відображає вікно для авторизації. Виконує функцію входу в додаток.
	CarWindow	Відображає вікно для роботи з автомобілями.
	CarWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.
	ClientWindow	Відображає вікно для роботи з клієнтами.
	ClientWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.

Продовження таблиці 2.2

	MechanicWindow	Відображає вікно для роботи з механіком.
	MechanicWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.
	SpareWindow	Відображає вікно для роботи з запчастин.
	SpareWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.
	ServiceWindow	Відображає вікно для роботи з послугам.
	ServiceWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.
	MainAppWindow	Відображає вікно для роботи з головних вікном.
	MainAppWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.
	NewOrderWindow	Відображає вікно для роботи з новим замовлення.
	NewOrderWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.
	OrderDescriptionWindow	Відображає вікно для роботи з подробицями замовленнями.

	OrderDescriptionWindowModel	Містить данні які потрібні для роботи з інтерфейсом та методи для взаємодії з базою даних.

2.2.2 Архітектурні компоненти програмного забезпечення

Модуль роботи з базою даних.

Для того, щоб відображати, змінювати, додавати інформацію треба звертатися до БД. Цю поставлену задачу виконує клас DbWorker. Оскільки цей клас існує тільки для звернення до бази даних то статичні поля є непотрібними. Внизу буде наведена таблиця 2.3 в якій будуть наведені всі методи класа DbWorker.

Таблиця 2.3 – Опис класів та методів модуля Database Manager.

Назва класа	Метод	Параметри які приймає і який тип повертає	Опис
DbWorker	CheckLogin()	string login, string password/ bool	Перевіряє введені дані користувача для авторизації.
	GetUserType()	string login, string password/ string	Отримує тип користувача.
	AddCarToDb()	CarModel carModel/ void	Додає автомобіль до БД.
	EditCar()	CarModel carModel/ void	Корегую інформацію автомобіля.
	DeleteCarFromDb()	CarModel carModel/ void	Видалити автомобіль з БД.

Продовження таблиці 2.3

	GetCars()	-/ List<CarModel>	Отримує список всіх автомобілів.
	AddClientToDb()	ClientModel clientModel / void	Додає автомобіль до БД.
	EditClient()	ClientModel clientModel/ void	Корегую інформацію клієнта.
	DeleteClientFromDb()	ClientModel clientModel/ void	Видаляє клієнта з БД.
	GetClients()	-/ List<ClientModel>	Отримує список всіх клієнтів.
	GetAllAvailableCars()	ClientModel clientmodel / List<CarModel>	Отримує список всіх автомобілів у яких нема власника.
	GetClientCars()	ClientModel clientModel/ List<CarModel>	Отримує список всіх автомобілів клієнта.
	RemoveCarFromClient()	ClientModel clientModel, CarModel carModel/ void	Видаляє автомобіль від клієнта з БД.

Продовження таблиці 2.3

	AddCarToClient()	ClientModel clientModel, CarModel carModel/ void	Додає автомобіль до клієнта в БД.
	AddLoginInfoToDb()	LogininfoModel loginInfoModel/ void	Додає дані для входу в систему у БД.
	AddMechanictoDb()	MechanicModel mechanicModel/ void	Додає механіка до БД.
	EditMechanic()	mechanicModel mechanicModel/ void	Корегую інформацію механіка.
	DeleteMechanicFromDb()	mechanicModel mechanicModel/ void	Видаляє механіка з БД.
	GetMechanics()	-/ List<MechanicModel>	Отримує список всіх механіків.
	AddServiceToDb()	Servicemodel serviceModel/ void	Додає послугу до БД.
	EditService()	Servicemodel serviceModel/ void	Корегую інформацію послуги.
	DeleteServiceFromDb()	Servicemodel serviceModel/ void	Видаляє послугу з БД.

Продовження таблиці 2.3

	GetServices()	-/ List<ServiceModel>	Отримує список всіх послуг.
	AddSpareToDb()	SpareModel spareModel/ void	Додає запчастину до БД.
	EditSpare()	SpareModel spareModel/ void	Корегую інформацію запчастини.
	DeleteSpareFromDb()	SpareModel spareModel/ void	Видаляє запчастину з БД.
	GetSpares()	-/ List<SpareModel>	Отримує список всіх запчастин.
	AddOrderToDb()	OrderModel orderModel/ void	Додає замовлення до БД.
	EditOrder()	OrderModel orderModel/ void	Корегую інформацію замовлення.
	DeleteorderFromDb()	OrderModel orderModel/ void	Видаляє замовлення з БД.
	GetOrders()	-/ List<OrderModel>	Отримує список всіх замовлень.

Продовження таблиці 2.3

	GetClientCars()	Int clientId/ List<CarModel>	Отримує список всіх автомобілів клієнта.

Даний модуль займає достатньо велику частину проєкту, адже тут реалізовані всі запити до бази даних, яких є багато. Виконує всі запити один клас DbWorker. До нього звертаються з метою отримання, редагування, видаленню та додавання інформації. Для роботи з БД в методах реалізовано звернення за допомогою технології Entity Framework, яка дозволяє репрезентувати всі сутності як класи(моделі) для подальшого маніпулювання, яке потрібно виконати над інформацією.

Модуль інтерфейс користувача

Інтерфейс розроблений за допомогою Windows Presentation Foundation. Складається він з декількох вікон : MainWindow, MainAppWindow, CarWindow, ClientWindow, MechanicWindow, ServiceWindow, SpareWindow, OrderDescriptionWindow та NewOrderWindow. Відповідно для кожного вікна WPF існує клас за розширенням .cs де прописана вся логіка взаємодії з елементами які розташовані у вікні в Grid. Також для кожного вікна був створений відповідний клас моделі, завданнями якого, є зберігання даних на момент роботи додатку та звернення до класу DbWorker для подальшою відображення нових даних. У інтерфейсів даного додатку (вікон WPF) використовуються наступні графічні елементи: TextBox, ComboBox, Button, Label та DataGrid.

В таблиці 2.4 будуть наведені методи які використовують класи даного модуля.

Таблиця 2.4 – Опис методів і класів модулю інтерфейсу

Назва класу	Метод	Параметри які приймає і який тип повертає	Опис
MainWindow	Login_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Увійти» для авторизації користувача.
CarWindow	CarWindow_OnLoaded	object sender, RoutedEventArgs e/ void	Реагує на момент завантаження вікна. Завантажує список всіх автомобілів.
	AddCar_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Додати автомобіль» для додавання автомобіля.
	EditCar_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Редагувати автомобіль» для редагування автомобіля.
	DeleteCar_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Видалити автомобіль» для видалення автомобіля.
	Car_DataStorage_OnSelectionChanged	object sender, SelectionChangedEventArgs e/ void	Реагує на зміну обраного елемента в Data Grid. Передає моделі дані обраного об’єкта.

Продовження таблиці 2.4

ClientWindow	ClientWindow_OnLoaded	object sender, RoutedEventArgs e/ void	Реагує на момент завантаження вікна. Завантажує список всіх клієнтів.
	AddClient_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Додати клієнта» для додавання клієнта.
	EditClient_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Редагувати клієнта» для редагування клієнта.
	DeleteClient_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Видалити клієнта» для видалення клієнта.
	Client_DataStorage_OnSelectionChanged	object sender, SelectionChangedEventArgs e/ void	Реагує на зміну обраного елемента в Data Grid. Передає моделі дані обраного об’єкта.
MechanicWindow	MechanicWindow_OnLoaded	object sender, RoutedEventArgs e/ void	Реагує на момент завантаження вікна. Завантажує список всіх механіків.
	AddMechanic_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Додати механіка» для додавання механіка.
	EditMechanic_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Редагувати механіка» для редагування механіка.

Продовження таблиці 2.4

	DeleteMechanic_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Видалити механіка» для видалення механіка.
	Mechanic_DataStorage_OnSelectionChanged	object sender, SelectionChangedEventArgs e/ void	Реагує на зміну обраного елемента в Data Grid. Передає моделі дані обраного об’єкта.
ServiceWindow	ServiceWindow_OnLoaded	object sender, RoutedEventArgs e/ void	Реагує на момент завантаження вікна. Завантажує список всіх послуг.
	AddService_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Додати послугу» для додавання механіка.
	EditService_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Редагувати послугу» для редагування послугу.
	DeleteService_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Видалити послугу» для видалення послугу.
	Service_DataStorage_OnSelectionChanged	object sender, SelectionChangedEventArgs e/ void	Реагує на зміну обраного елемента в Data Grid. Передає моделі дані обраного об’єкта.

Продовження таблиці 2.4

SpareWindow	SpareWindow_OnLoaded	object sender, RoutedEventArgs e/ void	Реагує на момент завантаження вікна. Завантажує список всіх запчастин.
	AddSpare_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Додати запчастину» для додавання запчастини.
	EditSpare_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Редагувати запчастину» для редагування запчастини.
	DeleteSpare_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Видалити запчастину» для видалення запчастини.
	Spare_DataStorage_OnSelectionChanged	object sender, SelectionChangedEventArgs e/ void	Реагує на зміну обраного елемента в Data Grid. Передає моделі дані обраного об’єкта.
MainAppWindow	MainAppWindow_onLoaded	object sender, RoutedEventArgs e/ void	Реагує на момент завантаження вікна. Завантажує список всіх замовлень.
	PresentCarWindow_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Перейти до вікна роботи з автомобілями» для роботи з автомобілями.

Продовження таблиці 2.4

	PresentClient Window_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Перейти до вікна роботи з клієнтами» для роботи з клієнтами.
	PresentSpareWindow_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Перейти до вікна роботи з запчастинами» для роботи з запчастинами.
	PresentServiceWindow_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Перейти до вікна роботи з послугами» для роботи з послугами.
	PresentMechanicWindow_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Перейти до вікна роботи з механіками» для роботи з механіками.
	AddOrder_Button_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Перейти до вікна роботи з новим замовленням» для роботи з новим замовленням.
	OrderDataStorage_OnSelectionChanged	object sender, SelectionChangedEventArgs e/ void	Реагує на зміну обраного елемента в Data Grid. Передає моделі дані обраного об’єкта.

Продовження таблиці 2.4

	SeeDetailsButton_Click	object sender, RoutedEventArgs e/ void	Реагує на натиск кнопки «Перейти до вікна роботи з деталями замовлення» для роботи з деталями замовлення.

Основні використані елементи інтерфейсу:

- Window – є головним елементом інтерфейсу завдяки якому можна відображати всі інші графічні елементи;
- Button – елемент інтерфейсу, який при натисканні на нього виконує функціонал який був написаний в відповідному методі натискання кнопки;
- Label – елемент інтерфейсу, завдяки якому можна відображати статичну інформацію яку не можна змінювати під час робити програми;
- TextBox – елемент інтерфейсу, який представляє собою текстове поле куди користувач може вводити текст;
- ComboBox – елемент інтерфейсу, який представляє собою випадаючим списком об’єктів. Надає можливість вибору елементу зі списку;
- DataGrid – елемент інтерфейсу, який має можливість відображати список об’єктів та їх поля для зручного відображення. Також надає можливість вибору об’єкта зі списку.

Висновки по розділу

На даному етапі розробки було визначено 2 основні модулі програмного забезпечення: модуль роботи з базою даних та модуль інтерфейсу користувача.

Модуль роботи з базою даних надає функціонал, який дозволяє працювати з БД для занесення, змінення, видалення та отримання інформації, яку потребує програма на момент використання.

Модуль інтерфейсу користувача надає можливість керування програним забезпеченням та його функціоналом за допомогою графічного представлення елементів інтерфейсу.

Програмне забезпечення надає можливість керувати процесами виконання замовлень на станції технічного обслуговування за допомогою інтерфейсу Windows Foundation Presentation та фреймворку Entity Framework.

Додаток надає користувачу зручний та зрозумілий для використання інтерфейс.

2 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Одним із основних аспектів розробки програмного забезпечення є тестування додатка. Завдяки проведення покрокових тестів можна побачити та перевірити функціонал, зручність використання та коректність роботи функціоналу. Далі буде наведені переваги проведення тестування:

- виявлення та усунення наявних помилок, безпосередньо перед тим як додаток буде випущений та доступний для використання користувачів;
- зменшення до мінімуму витрат в часі, що випливає в прискорення розробки програмного забезпечення завдяки використання тестів на всіх етапах створення додатка;
- своєчасна оптимізація коду.

Розробка програмного забезпечення для комп’ютера має свої плюси. Одним з таких є необмеженість в потужності пристрою, що дає можливість обробляти велику кількість даних та використання динамічних графічних інтерфейсів. Ще одним плюсом розробки на комп’ютері є можливість розробки додатків буд будь - які пристрої. Але незважаючи на зазначені вище переваги, тестування додатку є не такою легкою та очевидною роботою.

Спочатку треба проаналізувати ти виконати тестування на всіх етапах розробки ПЗ. Потім враховуючи результат проведених тестувань та всі фактори розробки, можна буде виявити як швидко йде розробка додатка згідно графіка або навіть з прискоренням.

Тож після виконаних успішно тестувань користувач має можливість використовувати продукт без перешкод.

3.2 Опис процесів тестування

Як правило тестування виконується на декількох етапах роботи програми, коли розробляється ПЗ. Для тестів можна виділити такі етапи:

- виявлення доступного та недоступного для тестування функціоналу програми;
- написання Test – case;
- формування вимог виконання тестів;
- тестування та оцінка результатів;
- перегляд інформації результатів.

Системні вимоги:

- операційна система - Window 7 і новіше;
- наявність локального SQL сервера.

Апаратні вимоги:

- доступність вільної апаратної пам’яті для встановлення додатку;
- процесор з тактовою частотою 1 GHz і більше;
- достатньо оперативної пам’яті для активного використання програми.

Тестування було проведено на ноутбуці Mac book Pro з процесором Intel Core i9 – 9980H CPU, тактовою частотою 2.3 GHz, операційною системою Windows 10 та при наявності SQL Server 2017.

3.3 Опис контрольного прикладу

Під час проведення тестів даного додатку було виконано перевірку всього функціоналу та всі варіанти використання. Також було враховано роботу інтерфейсу і перевірені запобіжні функції, які викликаються при некоректних введених даних. Далі будуть наведений перелік таблиць які відображають кожне тестування:

- авторизація в систему (таблиця 3.1);
- перевірки авторизації з некоректними даними для входу (таблиця 3.2);
- робота з клієнтом (додавання, змінення, видалення) (таблиця 3.3);
- перевірка роботи з клієнтом з незаповненим(-и) полем(-ями).) (таблиця 3.4);
- робота з автомобілем (додавання, змінення, видалення) (таблиця 3.5);
- перевірка роботи з автомобілем з незаповненим(-и) полем(-ями). (таблиця 3.6);
- робота з механіком (додавання, змінення, видалення) (таблиця 3.7);
- перевірка роботи з механіком з незаповненим(-и) полем(-ями). (таблиця 3.8);
- робота з сервісом (додавання, змінення, видалення) (таблиця 3.9);
- перевірка роботи з сервісом з незаповненим(-и) полем(-ями). (таблиця 3.10);
- робота з запчастиною (додавання, змінення, видалення) (таблиця 3.11);
- перевірка роботи з запчастиною з незаповненим(-и) полем(-ями). (таблиця 3.12);
- додавання нового замовлення (таблиця 3.13);
- перевірка додавання замовлення з незаповненим(-и) полем(-ями). (таблиця 3.14);
- змінення обраного замовлення (таблиця 3.15);
- перевірка змінення обраного замовлення з незаповненим(-и) полем(-ями). (таблиця 3.16).

Таблиця 3.1 – Авторизація в систему

Мета Тесту	Перевірити функцію авторизації в систему користувача.
Початковий стан	Запущений додаток, відображений інтерфейс користувача.
Вхідні дані	String login, string password
Проведення тесту	Заповненні поля логіну та пароля. Далі натиснути кнопку увійти.
Очікуваний результат	Успішна авторизація та подальший перехід до головного вікна додатку
Фактичний результат	Успішна авторизація з подальшим переходом до головного вікна.

Таблиця 3.2 - Перевірки авторизації з некоректними даними для входу

Мета Тесту	Перевірити функцію авторизації в систему користувача з некоректними даними для входу.
Початковий стан	Запущений додаток, відображений інтерфейс користувача.
Вхідні дані	String login, string password
Проведення тесту	Заповнити поля логіну та пароля некоректними даними. Далі натиснути кнопку увійти.
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Некоректні дані для входу в систему».
Фактичний результат	Вивелось вікно оповіщення з повідомленням: «Некоректні дані для входу в систему».

Таблиця 3.3 – Робота з клієнтом

Мета Тесту	Перевірити функціонал роботи з клієнтом.
Початковий стан	Запущений додаток, відображене вікно для роботи з клієнтами.
Вхідні дані	Обраний клієнт зі списку всіх клієнтів
Проведення тесту	Заповнити поля клієнта. Далі натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Обраний клієнт доданий, змінений або видалений, поля вводу очищені для отримання нової інформації.
Фактичний результат	Клієнт доданий, змінений або видалений з бази даних.

Таблиця 3.4 - Перевірка роботи з клієнтом з незаповненим(-и) полем(-ями)

Мета Тесту	Перевірити функціонал роботи з клієнтом при незаповнених полів.
Початковий стан	Запущений додаток, відображене вікно для роботи з клієнтами.
Вхідні дані	Обраний клієнт зі списку всіх клієнтів
Проведення тесту	Натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Одне чи декілька полів є пустими».

Продовження таблиці 3.4

Фактичний результат	Вивелось вікно оповіщення з повідомленням Одне чи декілька полів є пустими».

Таблиця 3.5 - Робота з автомобілем

Мета Тесту	Перевірити функціонал роботи з автомобілем.
Початковий стан	Запущений додаток, відображене вікно для роботи з автомобілями.
Вхідні дані	Обрана машина зі списку всіх машин.
Проведення тесту	Заповнити поля автомобіля. Далі натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Обрана машина додана, змінена або видалена, поля вводу очищені для отримання нової інформації.
Фактичний результат	Автомобіль доданий, змінений або видалений з бази даних.

Таблиця 3.6 - Перевірка роботи з авто з незаповненим(-и) полем(-ями)

Мета Тесту	Перевірити функціонал роботи з автомобілем при незаповнених полів.
Початковий стан	Запущений додаток, відображене вікно для роботи з автомобілями.
Вхідні дані	Обрана машина зі списку всіх машин.

Продовження таблиці 3.6

Проведення тесту	Натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Одне чи декілька полів є пустими».
Фактичний результат	Вивелось вікно оповіщення з повідомленням Одне чи декілька полів є пустими».

Таблиця 3.7 - Робота з механіком

Мета Тесту	Перевірити функціонал роботи з механіком.
Початковий стан	Запущений додаток, відображене вікно для роботи з механіками.
Вхідні дані	Обраний механік зі списку всіх механіків.
Проведення тесту	Заповнити поля механіка. Далі натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Обраний механік доданий, змінений або видалений, поля вводу очищені для отримання нової інформації.
Фактичний результат	Механік доданий, змінений або видалений з бази даних.

Таблиця 3.8 - Перевірка роботи з клієнтом з незаповненими полями

Мета Тесту	Перевірити функціонал роботи з механіком при незаповнених полів.
Початковий стан	Запущений додаток, відображене вікно для роботи з механіком.
Вхідні дані	Обраний механік зі списку всіх механіків.
Проведення тесту	Натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Одне чи декілька полів є пустими».
Фактичний результат	Вивелось вікно оповіщення з повідомленням Одне чи декілька полів є пустими».

Таблиця 3.9 - Робота з послугою

Мета Тесту	Перевірити функціонал роботи з послугою.
Початковий стан	Запущений додаток, відображене вікно для роботи з послугами.
Вхідні дані	Обрана послуга зі списку всіх послуг.
Проведення тесту	Заповнити поля послуги. Далі натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Обрана послуга додана, змінена або видалена, поля вводу очищені для отримання нової інформації.

Продовження таблиці 3.9

Фактичний результат	послуга доданий, змінений або видалений з бази даних.

Таблиця 3.10 - Перевірка роботи з авто з незаповненим(-и) полем(-ями)

Мета Тесту	Перевірити функціонал роботи з послугою при незаповнених полів.
Початковий стан	Запущений додаток, відображене вікно для роботи з послугами.
Вхідні дані	Обрана послуга з списку всіх машин.
Проведення тесту	Натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Одне чи декілька полів є пустими».
Фактичний результат	Вивелось вікно оповіщення з повідомленням Одне чи декілька полів є пустими».

Таблиця 3.11 - Робота з запчастиною

Мета Тесту	Перевірити функціонал роботи з запчастиною.
Початковий стан	Запущений додаток, відображене вікно для роботи з запчастинами.
Вхідні дані	Обрана запчастина зі списку всіх запчастин.
Проведення тесту	Заповнити поля запчастини. Далі натиснути кнопку Додати/Змінити/Видалити.

Продовження таблиці 3.11

Очікуваний результат	Обрана запчастина додана, змінена або видалена, поля вводу очищені для отримання нової інформації.
Фактичний результат	Запчастина додана, змінена або видалена з бази даних.

Таблиця 3.12 - Перевірка роботи з авто з незаповненим(-и) полем(-ями)

Мета Тесту	Перевірити функціонал роботи з запчастиною при незаповнених полів.
Початковий стан	Запущений додаток, відображене вікно для роботи з запчастинами.
Вхідні дані	Обрана запчастина зі списку всіх запчастин.
Проведення тесту	Натиснути кнопку Додати/Змінити/Видалити.
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Одне чи декілька полів є пустими».
Фактичний результат	Вивелось вікно оповіщення з повідомленням Одне чи декілька полів є пустими».

Таблиця 3.13 - Додавання нового замовлення

Мета Тесту	Перевірити функціонал додавання замовлення.
Початковий стан	Запущений додаток, відображене вікно для додавання нового замовлення.
Вхідні дані	Поля замовлення.
Проведення тесту	Заповнити поля замовлення. Далі натиснути кнопку Додати.
Очікуваний результат	Замовлення додане до бази даних, поля вводу очищені для отримання нової інформації.
Фактичний результат	Замовлення успішно додане до бази даних

Таблиця 3.14 - Перевірка додавання замовлення з незаповненим(-и) полем(-ями)

Мета Тесту	Перевірити функціонал додавання замовлення при незаповнених полів.
Початковий стан	Запущений додаток, відображене вікно для додавання нового замовлення.
Вхідні дані	Поля замовлення.
Проведення тесту	Натиснути кнопку Додати.
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Одне чи декілька полів є пустими».

Продовження таблиці 3.14

Фактичний результат	Вивелось вікно оповіщення з повідомленням Одне чи декілька полів є пустими».

Таблиця 3.15 – Змінення обраного замовлення

Мета Тесту	Перевірити функціонал змінення замовлення.
Початковий стан	Запущений додаток, відображене вікно для змінення обраного замовлення.
Вхідні дані	Поля замовлення.
Проведення тесту	Заповнити поля замовлення. Далі натиснути кнопку Змінити.
Очікуваний результат	Замовлення змінено в базі даних, поля вводу очищені для отримання нової інформації.
Фактичний результат	Замовлення успішно змінено в базі даних.

Таблиця 3.16 - Перевірка змінення обраного замовлення з незаповненим(-и) полем(-ями)

Мета Тесту	Перевірити функціонал змінення замовлення при незаповнених полях.
Початковий стан	Запущений додаток, відображене вікно для змінення обраного замовлення.
Вхідні дані	Поля замовлення.
Проведення тесту	Натиснути кнопку Змінити
Очікуваний результат	Вивід вікна оповіщення з повідомленням: «Одне чи декілька полів є пустими».
Фактичний результат	Вивелось вікно оповіщення з повідомленням Одне чи декілька полів є пустими».

Висновки по розділу

Була досліджена якість програмного забезпечення. Під час дослідження було виявлено, що для даної програми потрібно проводити тестування функціональності та роботи інтерфейсу користувача.

В доповнення до цього був описаний процес тестування, апаратні та системні вимоги які є обов’язковою вимогою розробки додатку. Крім цього були наведені програмні та технічні вимоги для програмного забезпечення.

Також були детально описані Test – Case у вигляді таблиць. Були проведені тестування усіх функціональних вимог ПЗ.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання програмного забезпечення потрібно мати ноутбук чи комп’ютер, на якому встановлена операційна система Windows 7 або вище. Також обов’язковим елементом для запуску програми є встановлений локальний SQL Server 2013 і новіше. Для запуску додатка потрібно запустити файл StoService.exe. Перед використанням додатку треба спочатку ознайомитись з інструкцією користувача, щоб дізнатись про весь доступний функціонал програми. Після ознайомлення з інструкцією можна переходити до використання програмного забезпечення.

4.2 Робота з програмним забезпеченням

Для того щоб почати працювати з додатком потрібно спочатку пройти авторизацію. Потрібно зазначити, що ,згідно з, типом користувача буде наданий відповідні функціональні можливості використання програми. Після успішної авторизації користувачу надається можливість працювати з додатком та виконувати потрібні для управління процесами виконання замовлення клієнтів на станції технічного обслуговування. Детальний опис разом із зображеннями наведені в додатку «Інтерфейс роботи користувача».

Висновки по розділу

В даному розділі було детально описане розгортання програми на пристрої користувача та представлені вимоги системи, які є необхідними для виконання функціональних вимог додатку.

Також були наведені покрокове використання ПЗ.

Висновки

Даний дипломний проєкт присвячений розробці додатку для управління процесами виконання замовлень клієнтів на станції технічного обслуговування.

Тема проєкта є досить актуальною, оскільки СТО є необхідною частиною кожного великого міста або містечка, проте далеко не кожний стартап може дозволити собі програмне забезпечення, яке коштує досить дорого.

В пункті «Аналіз вимог до ПЗ» були наведені опис предметного середовища та загальні положення. Також було зазначено, що таке Entity Framework та які підходити можна застосувати для його використання.

Крім того в даному пункті був описаний огляд на існуючі аналогічні додатки та варіанти готових рішень. Стало зрозуміло, що майже всі додатки є платними або недостатньо функціональними. Тож створення додатку який міг би надати можливість керувати замовленнями клієнтів є досить актуальним.

Окрім цього були сформовані варіанти використання, призначення, головна мета, функціональні вимоги, нефункціональні вимоги та основні цілі розробки даного продукту.

Поетапного описання моделі роботи користувача з додатком описане в розділі «Моделювання та конструювання програмного забезпечення». Також було розроблено та описано архітектуру ПЗ. В результаті було створено 2 модулі: модуль який відповідає за роботу з базою даних та модуль який відповідає за інтерфейс користувача.

Далі в підрозділі «Опис архітектури програмного забезпечення» були описані всі модулі та наведені клас, які в них входять. В наступному підрозділі була наведена детальна інформація про функціонал додатка. «Модуль роботи з базою даних» містить в собі методи за допомогою яких здійснюється робота з БД. Крім того наведений детальний опис класів та

функціоналу даного модуля. «Модуль інтерфейсу користувача» містить інформацію про всі графічні об’єкти з якими може працювати користувач. Окрім цього описує детально роботу елементів інтерфейсу які мають можливість активно змінюватись.

В розділі «Аналіз якості та тестування програмного забезпечення» було проведено аналіз якості додатки та виявлені основні моменти які є необхідними при тестуванні ПЗ.

Крім того були проведені Test – Cases та детально описані у вигляді таблиць.

В останньому розділі «Впровадження та супровід програмного забезпечення» було наведено детальний опис та реалізація розгортання додатку на пристрою користувача. Також було створено інструкцію по використанню програми для користувача.

В перспективі для покращення даного ПЗ можна перевести зберігання даних на онлайн сервіс та реалізувати розгортання додатка через інтернет браузер.

ПЕРЕЛІК ПОСИЛАНЬ

1. .net [Електронний ресурс]// https://docs.microsoft.com/ru-ru/dotnet/framework/get-started/overview
2. .net [Електронний ресурс]// https://skillbox.ru/media/code/kak_rabotaet_net_i_zachem_on_nuzhen/
3. C# 10 and .NET 6 – Modern Cross-Platform development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code, 6th Edition.
4. WPF [Електронний ресурс]// https://docs.microsoft.com/ru-ru/visualstudio/designers/getting-started-with-wpf?view=vs-2022
5. WPF [Електронний ресурс]// https://metanit.com/sharp/wpf/
6. Chris Sells & Ian. Griffiths Programming WPF: Building Windows UI with Windows Presentation Foundation.
7. Entity Framework [Електронний ресурс]// https://metanit.com/sharp/entityframework/1.1.php
8. Entity Framework [Електронний ресурс]// https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/ef/overview
9. Jon P Smith Entity Framework Core in Action 1st Edition.
10. Julia Lerman Programming entity Framework 1st Edition.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЦЕСАМИ ВИКОНАННЯ ЗАМОВЛЕНЬ КЛІЄНТІВ НА СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Опис програми

КПІ.ІТ-8206.045490.06.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Валерій НОВІНСЬКИЙ

Нормоконтроль:
_____ Валерій НОВІНСЬКИЙ

Виконавець:
_____ Сергій БОГАЧЕНКО

Київ – 2022

ТЕКСТ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування автомобілів

(Найменування програми (документа))

CD-R

(Вид носія даних)

26 арк, 50,6 МБ

(Обсяг програми, арк., Кб)

Київ – 2022

```
namespace Diplom
{
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
         Login_TextBox.Text = "apostol@gmail.com";
         Password_TextBox.Text = "1234567";
      }

      private void Login_Button_Click(object sender, RoutedEventArgs e)
      {
         var worker = new DbWorker();
         if (CheckInputs())
         {
            if (worker.CheckLogin(Login_TextBox.Text, Password_TextBox.Text))
            {
               if (worker.GetUserType(Login_TextBox.Text, Password_TextBox.Text) == "admin")
               {
                  var form = new MainAppWindow() { Owner = this };
                  form.UserType = "admin";
                  form.UserName = worker.GetAdmin(Login_TextBox.Text, Password_TextBox.Text).Name;
                  form.UserSecondName = worker.GetAdmin(Login_TextBox.Text, Password_TextBox.Text).SecondName;
                  CleanInputs();
                  form.ShowDialog();
               }
               else
               {
                  var form = new MainAppWindow() { Owner = this };
                  form.UserType = "mechanic";
                  form.UserName = worker.GetMechanic(Login_TextBox.Text, Password_TextBox.Text).Name;
                  form.UserSecondName = worker.GetMechanic(Login_TextBox.Text, Password_TextBox.Text).SecondName;
                  CleanInputs();
                  form.ShowDialog();
               }
            }
            else
            {
               MessageBox.Show("Invalid login or password please try again");
               CleanInputs();
            }
         }
         else
         {
            MessageBox.Show("One or more Fields are empty");
            CleanInputs();
         }

      }

      public bool CheckInputs()
      {
         bool result;
         if (Login_TextBox.Text != "" && Password_TextBox.Text != "")
         {
            result = true;
         }
         else
         {
            result = false;
         }
         return result;
      }

      public void CleanInputs()
```

```
        {
            Login_TextBox.Text = "";
            Password_TextBox.Text = "";
        }
    }
}

namespace Diplom
{
    public partial class SpareWindow : Window
    {
        private readonly  SpareWindowModel _spareWindowModel = new SpareWindowModel();
        public SpareWindow()
        {
            InitializeComponent();
        }

        private void SpareWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            LoadData();
        }

        public void LoadData()
        {
            Spare_DataStorage.ItemsSource = null;
            Spare_DataStorage.ItemsSource = _spareWindowModel.GetSpares();
        }

        private void AddSpare_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
                {
                    if (CheckForNumber(SpareCost_TextBox.Text))
                    {
                        var spare = new SpareModel()
                        {
                            Naming = SpareNaming_TextBox.Text,
                            Cost = Convert.ToInt32(SpareCost_TextBox.Text)
                        };
                        _spareWindowModel.AddSpareToDb(spare);
                        CleanInputs();
                        UpdateData();
                    }
                    else
                    {
                        CleanInputs();
                        MessageBox.Show("type correct number without coma or dot");
                    }

                }
                else
                {
                    CleanInputs();
                    MessageBox.Show("One or more texBox fields are too long");
                }
            }
            else
            {
                MessageBox.Show("One or more textBox fields are empty");
            }
        }

        private void EditSpare_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
```

```
            if (CheckForSize())
            {
                if (CheckForNumber(SpareCost_TextBox.Text))
                {
                    _spareWindowModel.GetDataForModel(SpareNaming_TextBox.Text,
Convert.ToInt32(SpareCost_TextBox.Text));
                    _spareWindowModel.EditSpare();
                    CleanInputs();
                    UpdateData();
                }
                else
                {
                    CleanInputs();
                    MessageBox.Show("type correct number without coma or dot");
                }
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }

    }

    private void DeleteSpare_Button_Click(object sender, RoutedEventArgs e)
    {
        if (CheckInputs())
        {
            if (CheckForSize())
            {
                _spareWindowModel.DeleteSpare();
                CleanInputs();
                UpdateData();
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }
    }

    private void Spare_DataStorage_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        if (Spare_DataStorage.SelectedItem == null)
        {
            return;
        }
        _spareWindowModel.SpareModel = Spare_DataStorage.SelectedItem as SpareModel;
        SpareNaming_TextBox.Text = _spareWindowModel.SpareModel.Naming;
        SpareCost_TextBox.Text = _spareWindowModel.SpareModel.Cost.ToString();
    }

    public void UpdateData()
    {
        Spare_DataStorage.ItemsSource = _spareWindowModel.GetSpares();
    }

    public bool CheckForSize()
    {
```

```
            bool result;
            if (SpareNaming_TextBox.Text.Length < 50 && SpareCost_TextBox.Text.Length < 10)
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public bool CheckInputs()
        {
            bool result;
            if (SpareNaming_TextBox.Text != "" && SpareCost_TextBox.Text != "")
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public bool CheckForNumber(string str)
        {
            int numericValue = 0;
            bool isNumber = int.TryParse(str, out numericValue);
            return isNumber;
        }


        private void CleanInputs()
        {
            SpareNaming_TextBox.Text = "";
            SpareCost_TextBox.Text = "";
        }


    }
            }


namespace Diplom.SpareFlow
{
    public class SpareWindowModel
    {
        public SpareModel SpareModel = new SpareModel();
        private readonly DbWorker _worker = new DbWorker();

        public List<SpareModel> GetSpares()
        {
            return _worker.GetSpares();
        }

        public void AddSpareToDb(SpareModel spareModel)
        {
            _worker.AddSpareToDb(spareModel);
        }

        public void EditSpare()
        {
            _worker.EditSpare(SpareModel);
        }

        public void DeleteSpare()
```

```
        {
            _worker.DeleteSpareFromDb(SpareModel);
        }

        public void GetDataForModel(string naming, int cost)
        {
            SpareModel.Naming = naming;
            SpareModel.Cost = cost;
        }
    }
}

namespace Diplom
{
    public partial class ServiceWindow : Window
    {
        private readonly ServiceWindowModel _serviceWindowModel = new ServiceWindowModel();
        public ServiceWindow()
        {
            InitializeComponent();
        }

        private void ServiceWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            LoadData();
        }

        public void LoadData()
        {
            Service_DataStorage.ItemsSource = null;
            Service_DataStorage.ItemsSource = _serviceWindowModel.GetServices();
        }

        private void AddService_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
                {
                    if (CheckForNumber(ServiceCost_TextBox.Text))
                    {
                        var service = new ServiceModel()
                        {
Naming = ServiceNaming_TextBox.Text,
                            Cost = Convert.ToInt32(ServiceCost_TextBox.Text)
                        };
                        _serviceWindowModel.AddServiceToDb(service);
                        CleanInputs();
                        UpdateData();
                    }
                    else
                    {
                        CleanInputs();
                        MessageBox.Show("type correct number without coma or dot");
                    }

                }
                else
                {
                    CleanInputs();
                    MessageBox.Show("One or more texBox fields are too long");
                }
            }
            else
            {
                MessageBox.Show("One or more textBox fields are empty");
            }
        }
```

```
        private void EditService_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
                {
                    if (CheckForNumber(ServiceCost_TextBox.Text))
                    {
                        _serviceWindowModel.GetDataForModel(ServiceNaming_TextBox.Text,
Convert.ToInt32(ServiceCost_TextBox.Text));
                        _serviceWindowModel.EditService();
                        CleanInputs();
                        UpdateData();
                    }
                    else
                    {
                        CleanInputs();
                        MessageBox.Show("type correct number without coma or dot");
                    }
                }
                else
                {
                    CleanInputs();
                    MessageBox.Show("One or more texBox fields are too long");
                }
            }
            else
            {
                MessageBox.Show("One or more textBox fields are empty");
            }

        }

        private void DeleteService_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
                {
                    _serviceWindowModel.DeleteService();
                    CleanInputs();
                    UpdateData();
                }
                else
                {
                    CleanInputs();
                    MessageBox.Show("One or more texBox fields are too long");
                }
            }
            else
            {
                MessageBox.Show("One or more textBox fields are empty");
            }

        }

        private void Service_DataStorage_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (Service_DataStorage.SelectedItem == null)
            {
                return;
            }
            _serviceWindowModel.ServiceModel = Service_DataStorage.SelectedItem as ServiceModel;
            ServiceNaming_TextBox.Text = _serviceWindowModel.ServiceModel.Naming;
            ServiceCost_TextBox.Text = _serviceWindowModel.ServiceModel.Cost.ToString();
        }

        public void UpdateData()
```

```
        {
            Service_DataStorage.ItemsSource = _serviceWindowModel.GetServices();
        }
        public bool CheckForSize()
        {
            bool result;
            if (ServiceNaming_TextBox.Text.Length < 50 && ServiceCost_TextBox.Text.Length < 10)
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public bool CheckInputs()
        {
            bool result;
            if (ServiceNaming_TextBox.Text != "" && ServiceCost_TextBox.Text != "")
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public bool CheckForNumber(string str)
        {
            int numericValue = 0;
            bool isNumber = int.TryParse(str, out numericValue);
            return isNumber;
        }


        private void CleanInputs()
        {
            ServiceNaming_TextBox.Text = "";
            ServiceCost_TextBox.Text = "";
        }
    }
}

namespace Diplom.ServiceFlow
{
    public class ServiceWindowModel
    {
        public ServiceModel ServiceModel = new ServiceModel();
        private readonly DbWorker _worker = new DbWorker();

        public List<ServiceModel> GetServices()
        {
            return _worker.GetServices();
        }

        public void AddServiceToDb(ServiceModel serviceModel)
        {
            _worker.AddServiceToDb(serviceModel);
        }

        public void EditService()
        {
            _worker.EditService(ServiceModel);
        }
```

```
        public void DeleteService()
        {
            _worker.DeleteServiceFromDb(ServiceModel);
        }

        public void GetDataForModel(string naming, int cost)
        {
            ServiceModel.Naming = naming;
            ServiceModel.Cost = cost;
        }

    }
}



namespace Diplom
{
    public partial class MainAppWindow : Window
    {
        public string UserName;
        public string UserSecondName;
        public string UserType;

        private readonly MainAppWindowModel _mainAppWindowModel = new MainAppWindowModel();


        public MainAppWindow()
        {
            InitializeComponent();
            LoadData();
        }

        public void LoadData()
        {
            OrderDataStorage.ItemsSource = null;
           OrderDataStorage.ItemsSource = _mainAppWindowModel.GetAllOrders();
        }
        private void MainAppWindow_OnClosed(object sender, EventArgs e)
        {
            var form = this.Owner as MainWindow;
        }

        private void MainAppWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            UserInfo_Label.Content = UserName + " " + UserSecondName;
            GetAvailableOptions();
        }

        public void GetAvailableOptions()
        {
            if (UserType == "admin")
            {
                AddOrderButton.IsEnabled = false;
                AddOrderButton.Visibility = Visibility.Hidden;
                SeeDetailsButton.IsEnabled = false;
                SeeDetailsButton.Visibility = Visibility.Hidden;
                DeleteOrderButton.IsEnabled = false;
                DeleteOrderButton.Visibility = Visibility.Hidden;
            }
            else
            {
                PresentMechanicWindow.IsEnabled = false;
                PresentMechanicWindow.Visibility = Visibility.Hidden;
                PresentSpareWindow_Button.IsEnabled = false;
                PresentSpareWindow_Button.Visibility = Visibility.Hidden;
            }
        }
```

```
    private void PresentCarWindow_Button_Click(object sender, RoutedEventArgs e)
    {
        var form = new CarWindow() { Owner = this };
        form.ShowDialog();
    }

    private void PresentClientWindow_Button_Click(object sender, RoutedEventArgs e)
    {
        var form = new ClientWindow() { Owner = this };
        form.ShowDialog();
    }

    private void PresentSpareWindow_Button_Click(object sender, RoutedEventArgs e)
    {
        var form = new SpareWindow() { Owner = this };
        form.ShowDialog();
    }

    private void PresentServiceWindow_Button_Click(object sender, RoutedEventArgs e)
    {
        var form = new ServiceWindow() { Owner = this };
        form.ShowDialog();
    }

    private void PresentMechanicWindow_Click(object sender, RoutedEventArgs e)
    {
        var form = new MechanicWindow() { Owner = this };
        form.ShowDialog();
    }

    private void AddOrderButton_Click(object sender, RoutedEventArgs e)
    {
        var form = new NewOrderWindow() { Owner = this };
        form.UserName = UserName;
        form.UserSecondName = UserSecondName;
        form.ShowDialog();

    }

    private void OrderDataStorage_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        if (OrderDataStorage.SelectedItem == null)
        {
            return;
        }
        _mainAppWindowModel.OrderModel = OrderDataStorage.SelectedItem as OrderModel;
    }

    private void SeeDetailsButton_Click(object sender, RoutedEventArgs e)
    {
        var form = new OrderDescriptionWindow() { Owner = this };
        form.OrderData = OrderDataStorage.SelectedItem as OrderModel;
        form.UserName = UserName;
        form.UserSecondName = UserSecondName;
        form.ShowDialog();
        OrderDataStorage.SelectedItem = null;
    }

    private void DeleteOrderButton_Click(object sender, RoutedEventArgs e)
    {
        if (OrderDataStorage.SelectedItem == null)
        {
            return;
        }
        _mainAppWindowModel.DeleteOrder();
        LoadData();
    }
}
```

```
        }
namespace Diplom.Flows.AppFlow
{
    public class MainAppWindowModel
    {
        public OrderModel OrderModel = new OrderModel();
        private readonly DbWorker _worker = new DbWorker();

        public List<OrderModel> GetAllOrders()
        {
            return _worker.GetOrders();
        }

        public void DeleteOrder()
        {
            _worker.DeleteOrderServices(OrderModel.Description.Id);
            _worker.DeleteOrderSpares(OrderModel.Description.Id);
            _worker.DeleteOrderFromDb(OrderModel);
            _worker.DeleteOrderDescription(OrderModel.Description.Id);
        }


    }
        }
namespace Diplom.Flows.AppFlow.OrderFlow.Order_description
{
    public partial class OrderDescriptionWindow : Window
    {
        private readonly OrderDescriptionWindowModel _orderDescriptionWindowModel = new
OrderDescriptionWindowModel();
        public OrderModel OrderData = new OrderModel();
        public string UserName;
        public string UserSecondName;
        public OrderDescriptionWindow()
        {
            InitializeComponent();
        }

        private void OrderDescriptionWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            _orderDescriptionWindowModel.GetMechanic(UserName, UserSecondName);
            UserNameInfo.Content = UserName + " " + UserSecondName;
            _orderDescriptionWindowModel.OrderModel = OrderData;
            LoadData();

        }

        private void OrderDescriptionWindow_OnClosed(object sender, EventArgs e)
        {
            var form = this.Owner as MainAppWindow;
            form.LoadData();
        }

        public void LoadData()
        {
            ClientCarsComboBox.ItemsSource = null;
            SpareDataStorage.ItemsSource = null;
            ServiceDataStorage.ItemsSource = null;
            SpareInOrderDataStorage.ItemsSource = null;
            ServiceInOrderDataStorage.ItemsSource = null;

            ClientsComboBox.ItemsSource = _orderDescriptionWindowModel.GetClients();
            SpareDataStorage.ItemsSource = _orderDescriptionWindowModel.GetSpares();
            ServiceDataStorage.ItemsSource = _orderDescriptionWindowModel.GetServices();
            ServiceInOrderDataStorage.ItemsSource = _orderDescriptionWindowModel.GetServicesInOrder();
            SpareInOrderDataStorage.ItemsSource = _orderDescriptionWindowModel.GetSparesInOrder();
            DescriptionTextBox.Text = _orderDescriptionWindowModel.OrderModel.Description.Description;
            DateStartPicker.SelectedDate = Convert.ToDateTime(_orderDescriptionWindowModel.OrderModel.DateStart);
```

```
            DateEndPicker.SelectedDate = Convert.ToDateTime(_orderDescriptionWindowModel.OrderModel.DateEnd);
            UpdatePrice();
        }


        private void EditOrderButton_Click(object sender, RoutedEventArgs e)
        {
            if (ClientsComboBox.SelectedItem != null && ClientCarsComboBox.SelectedItem != null &&
                DateStartPicker.Text == "" && DateEndPicker.Text == "" && DescriptionTextBox.Text == "" &&
ServiceInOrderDataStorage.Items.Count == 0)
            {
                return;
            }
            _orderDescriptionWindowModel.OrderModel.Description.Description = DescriptionTextBox.Text;
            _orderDescriptionWindowModel.OrderModel.DateStart = DateStartPicker.SelectedDate.ToString();
            _orderDescriptionWindowModel.OrderModel.DateEnd = DateEndPicker.SelectedDate.ToString();
            _orderDescriptionWindowModel.EditOrder();
            //ResetData();
        }

        private void ClientsComboBox_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (ClientsComboBox.SelectedItem == null)
            {
                return;
            }
            _orderDescriptionWindowModel.OrderModel.Client = ClientsComboBox.SelectedItem as ClientModel;
            ClientCarsComboBox.ItemsSource = null;
            ClientCarsComboBox.ItemsSource = _orderDescriptionWindowModel.GetClientCars();
        }

        private void ClientCarsComboBox_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (ClientCarsComboBox.SelectedItem == null)
            {
                return;
            }
            _orderDescriptionWindowModel.OrderModel.Car = ClientCarsComboBox.SelectedItem as CarModel;
        }

        private void AddServiceToOrderButton_OnClick(object sender, RoutedEventArgs e)
        {
            if (ServiceDataStorage.SelectedItem == null)
            {
                return;
            }
            _orderDescriptionWindowModel.ServicesOrderList.Add(ServiceDataStorage.SelectedItem as ServiceModel);
            ServiceInOrderDataStorage.ItemsSource = null;
            ServiceInOrderDataStorage.ItemsSource = _orderDescriptionWindowModel.ServicesOrderList;
            UpdatePrice();
        }

        private void AddSpareToOrderButton_OnClick(object sender, RoutedEventArgs e)
        {
            if (SpareDataStorage.SelectedItem == null)
            {
                return;
            }
            _orderDescriptionWindowModel.SparesOrderList.Add(SpareDataStorage.SelectedItem as SpareModel);
            SpareInOrderDataStorage.ItemsSource = null;
            SpareInOrderDataStorage.ItemsSource = _orderDescriptionWindowModel.SparesOrderList;
            UpdatePrice();
        }

        private void RemoveServiceFromOrderButton_OnClick(object sender, RoutedEventArgs e)
        {
            if (ServiceInOrderDataStorage.SelectedItem == null)
            {
                return;
```

```
            }
            _orderDescriptionWindowModel.RemoveServiceFromOrder(ServiceInOrderDataStorage.SelectedIndex);
            ServiceInOrderDataStorage.ItemsSource = null;
            ServiceInOrderDataStorage.ItemsSource = _orderDescriptionWindowModel.ServicesOrderList;
            UpdatePrice();
        }

        private void RemoveSpareFromOrderButton_OnClick(object sender, RoutedEventArgs e)
        {
            if (SpareInOrderDataStorage.SelectedItem == null)
            {
                return;
            }
            _orderDescriptionWindowModel.RemoveSpareFromOrder(SpareInOrderDataStorage.SelectedIndex);
            SpareInOrderDataStorage.ItemsSource = null;
            SpareInOrderDataStorage.ItemsSource = _orderDescriptionWindowModel.SparesOrderList;
            UpdatePrice();
        }


        public void UpdatePrice()
        {
            TotalCostLabel.Content = _orderDescriptionWindowModel.CountTotalPrice();
        }

        public void ResetData()
        {
            ClientsComboBox.SelectedItem = null;
            ClientCarsComboBox.SelectedItem = null;
            ClientCarsComboBox.ItemsSource = null;

            DateStartPicker.SelectedDate = null;
            DateEndPicker.SelectedDate = null;
            DescriptionTextBox.Text = "";

            SpareInOrderDataStorage.ItemsSource = null;
            ServiceInOrderDataStorage.ItemsSource = null;
            TotalCostLabel.Content = "";
        }
    }
}
namespace Diplom.User_Interface.AppFlow.OrderFlow.Order_description
{
    public class OrderDescriptionWindowModel
    {
        public OrderModel OrderModel = new OrderModel();
        private readonly DbWorker _worker = new DbWorker();
        public List<SpareModel> SparesOrderList = new List<SpareModel>();
        public List<ServiceModel> ServicesOrderList = new List<ServiceModel>();

        public List<ClientModel> GetClients()
        {
            return _worker.GetClients();
        }

        public List<SpareModel> GetSpares()
        {
            return _worker.GetSpares();
        }

        public List<ServiceModel> GetServices()
        {
            return _worker.GetServices();
        }

        public List<CarModel> GetClientCars()
        {
            return _worker.GetClientCars(OrderModel.Client);
        }
```

```
        public List<SpareModel> GetSparesInOrder()
        {
            SparesOrderList = _worker.GetSpareDoneForOrder(OrderModel.Description.Id);
            return _worker.GetSpareDoneForOrder(OrderModel.Description.Id);
        }

        public List<ServiceModel> GetServicesInOrder()
        {
            ServicesOrderList = _worker.GetServiceDoneForOrder(OrderModel.Description.Id);
            return _worker.GetServiceDoneForOrder(OrderModel.Description.Id);
        }

        public void RemoveServiceFromOrder(int index)
        {
            ServicesOrderList.RemoveAt(index);
        }

        public void RemoveSpareFromOrder(int index)
        {
            SparesOrderList.RemoveAt(index);
        }

        public void EditOrder()
        {
            // GetDescriptionId();
            EditDescription();
            AddServicesAndSparesToDb();
            _worker.EditOrder(OrderModel);
        }

        private void AddServicesAndSparesToDb()
        {
            _worker.DeleteSparesFromOrder(OrderModel.Description.Id);
            _worker.DeleteServicesFromOrder(OrderModel.Description.Id);
            _worker.AddSparesToOrder(SparesOrderList, OrderModel.Description.Id);
            _worker.AddServicesToOrder(ServicesOrderList, OrderModel.Description.Id);
        }

        public void GetMechanic(string name, string secondName)
        {
            OrderModel.Mechanic = _worker.GetMechanicForName(name, secondName);
        }

        private void EditDescription()
        {
            OrderModel.Description.CarId = OrderModel.Car.Id;
            OrderModel.Description.MechanicId = OrderModel.Mechanic.Id;
            _worker.EditDescription(OrderModel.Description);
        }

        public int CountTotalPrice()
        {
            return ServicesOrderList.Sum(service => service.Cost) + SparesOrderList.Sum(selector: spare => spare.Cost);
        }
    }
}

namespace Diplom.Flows.AppFlow.NewOrder
{
    public partial class NewOrderWindow : Window
    {
        private readonly NewOrderWindowModel _newOrderWindowModel = new NewOrderWindowModel();
        public string UserName;
        public string UserSecondName;
        public NewOrderWindow()
        {
            InitializeComponent();
```

```
            LoadData();

        }

        private void NewOrderWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            _newOrderWindowModel.GetMechanic(UserName, UserSecondName);
            UserNameInfo.Content = UserName + " " + UserSecondName;
        }

        public void LoadData()
        {
            ClientCarsComboBox.ItemsSource = null;
            SpareDataStorage.ItemsSource = null;
            ServiceDataStorage.ItemsSource = null;
            ClientsComboBox.ItemsSource = _newOrderWindowModel.GetClients();
            SpareDataStorage.ItemsSource = _newOrderWindowModel.GetSpares();
            ServiceDataStorage.ItemsSource = _newOrderWindowModel.GetServices();
        }

        private void AddOrderButton_Click(object sender, RoutedEventArgs e)
        {
            if (ClientsComboBox.SelectedItem != null && ClientCarsComboBox.SelectedItem != null &&
                DateStartPicker.Text == "" && DateEndPicker.Text == "" && DescriptionTextBox.Text == "" &&
ServiceInOrderDataStorage.Items.Count == 0)
            {
                return;
            }

            _newOrderWindowModel.DescriptionModel.Description = DescriptionTextBox.Text;
            _newOrderWindowModel.OrderModel.DateStart = DateStartPicker.SelectedDate.ToString();
            _newOrderWindowModel.OrderModel.DateEnd = DateEndPicker.SelectedDate.ToString();
            _newOrderWindowModel.AddOrderToDataBase();
            ResetData();
        }

        private void AddServiceToOrder_Click(object sender, RoutedEventArgs e)
        {
            if (ServiceDataStorage.SelectedItem == null)
            {
                return;
            }
            _newOrderWindowModel.ServicesOrderList.Add(ServiceDataStorage.SelectedItem as ServiceModel);
            ServiceInOrderDataStorage.ItemsSource = null;
            ServiceInOrderDataStorage.ItemsSource = _newOrderWindowModel.ServicesOrderList;
            UpdatePrice();
        }

        private void AddSpareToOrderButton_Click(object sender, RoutedEventArgs e)
        {
            if (SpareDataStorage.SelectedItem == null)
            {
                return;
            }
            _newOrderWindowModel.SparesOrderList.Add(SpareDataStorage.SelectedItem as SpareModel);
            SpareInOrderDataStorage.ItemsSource = null;
            SpareInOrderDataStorage.ItemsSource = _newOrderWindowModel.SparesOrderList;
            UpdatePrice();
        }

        private void RemoveServiceFromOrderButton_Click(object sender, RoutedEventArgs e)
        {
            if (ServiceInOrderDataStorage.SelectedItem == null)
            {
                return;
            }
            _newOrderWindowModel.RemoveServiceFromOrder(ServiceInOrderDataStorage.SelectedIndex);
            ServiceInOrderDataStorage.ItemsSource = null;
            ServiceInOrderDataStorage.ItemsSource = _newOrderWindowModel.ServicesOrderList;
```

```
            UpdatePrice();
        }

        private void RemoveSpareFromOrderButton_Click(object sender, RoutedEventArgs e)
        {
            if (SpareInOrderDataStorage.SelectedItem == null)
            {
                return;
            }
            _newOrderWindowModel.RemoveSpareFromOrder(SpareInOrderDataStorage.SelectedIndex);
            SpareInOrderDataStorage.ItemsSource = null;
            SpareInOrderDataStorage.ItemsSource = _newOrderWindowModel.SparesOrderList;
            UpdatePrice();
        }

        private void ClientsComboBox_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (ClientsComboBox.SelectedItem == null)
            {
                return;
            }
            _newOrderWindowModel.ClientModel = ClientsComboBox.SelectedItem as ClientModel;
            ClientCarsComboBox.ItemsSource = null;
            ClientCarsComboBox.ItemsSource = _newOrderWindowModel.GetClientCars();
        }

        private void ClientCarsComboBox_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (ClientCarsComboBox.SelectedItem == null)
            {
                return;
            }
            _newOrderWindowModel.CarModel = ClientCarsComboBox.SelectedItem as CarModel;
        }

        public void UpdatePrice()
        {
            TotalCostLabel.Content = _newOrderWindowModel.CountTotalPrice();
        }

        public void ResetData()
        {
            ClientsComboBox.SelectedItem = null;
            ClientCarsComboBox.SelectedItem = null;
            ClientCarsComboBox.ItemsSource = null;

            DateStartPicker.SelectedDate = null;
            DateEndPicker.SelectedDate = null;
            DescriptionTextBox.Text = "";

            SpareDataStorage.SelectedItem = null;
            ServiceDataStorage.SelectedItem = null;

            SpareInOrderDataStorage.ItemsSource = null;
            ServiceInOrderDataStorage.ItemsSource = null;
            TotalCostLabel.Content = "";
        }

        private void NewOrderWindow_OnClosed(object sender, EventArgs e)
        {
            var form = this.Owner as MainAppWindow;
            form.LoadData();
        }
    }
}
namespace Diplom.Flows.AppFlow.OrderFlow.NewOrder
{
    public class NewOrderWindowModel
    {
```

```
public OrderModel OrderModel = new OrderModel();
public ClientModel ClientModel = new ClientModel();
public CarModel CarModel = new CarModel();
public MechanicModel MechanicModel = new MechanicModel();
public DescriptionModel DescriptionModel = new DescriptionModel();
private readonly DbWorker _worker = new DbWorker();
public List<SpareModel> SparesOrderList = new List<SpareModel>();
public List<ServiceModel> ServicesOrderList = new List<ServiceModel>();

public List<ClientModel> GetClients()
{
  return _worker.GetClients();
}

public List<SpareModel> GetSpares()
{
   return _worker.GetSpares();
}

public List<ServiceModel> GetServices()
{
   return _worker.GetServices();
}

public List<CarModel> GetClientCars()
{
   return _worker.GetClientCars(ClientModel);
}

public void RemoveServiceFromOrder(int index)
{
   ServicesOrderList.RemoveAt(index);
}

public void RemoveSpareFromOrder(int index)
{
   SparesOrderList.RemoveAt(index);
}

public void AddOrderToDataBase()
{
   CreateDescription();
   GetDescriptionId();
   AddServicesAndSparesToDb();
   _worker.AddOrderToDb(OrderModel);
}

public void GetDescriptionId()
{
   OrderModel.Description = _worker.GetOrderDescription(DescriptionModel.Description);
   OrderModel.Client = ClientModel;
   OrderModel.Car = CarModel;
   OrderModel.Mechanic = MechanicModel;
}

public void AddServicesAndSparesToDb()
{
   _worker.AddSparesToOrder(SparesOrderList, _worker.GetOrderDescription(DescriptionModel.Description).Id);
   _worker.AddServicesToOrder(ServicesOrderList, _worker.GetOrderDescription(DescriptionModel.Description).Id);
}

public void GetMechanic(string name, string secondName)
{
   MechanicModel = _worker.GetMechanicForName(name, secondName);
}

public void CreateDescription()
{
   DescriptionModel.CarId = CarModel.Id;
```

```
            DescriptionModel.MechanicId = MechanicModel.Id;
            _worker.AddDescriptionToDataBase(DescriptionModel);
        }

        public int CountTotalPrice()
        {
            return ServicesOrderList.Sum(service => service.Cost) + SparesOrderList.Sum(selector:spare => spare.Cost);
        }
    }
            }


namespace Diplom
{
    public partial class MechanicWindow : Window
    {
        readonly MechanicWindowModel _mechanicWindowModel = new MechanicWindowModel();
        public MechanicWindow()
        {
            InitializeComponent();
        }

        private void MechanicWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            Mechanik_DataStorage.ItemsSource = _mechanicWindowModel.GetMechanics();
        }

        private void Mechanik_DataStorage_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (Mechanik_DataStorage.SelectedItem == null)
            {
                return;
            }
            _mechanicWindowModel.MechanicModel = Mechanik_DataStorage.SelectedItem as MechanicModel;
            MechanikName_TextBox.Text = _mechanicWindowModel.MechanicModel.Name;
            MechanikSecondName_TextBox.Text = _mechanicWindowModel.MechanicModel.SecondName;
            MechanikPhone_TextBox.Text = _mechanicWindowModel.MechanicModel.PhoneNumber;
            MechanikExperience_TextBox.Text = _mechanicWindowModel.MechanicModel.WorkExperience;
            MechanicLogin_TextBox.Text = _mechanicWindowModel.MechanicModel.LoginInfo.Login;
            MechanicPassword_TextBox.Text = _mechanicWindowModel.MechanicModel.LoginInfo.Password;
        }

        private void DeleteMechanik_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
                {
                    _mechanicWindowModel.DeleteMechanicFromDb();
                    CleanInputs();
                    UpdateData();
                }
                else
                {
                    CleanInputs();
                    MessageBox.Show("One or more texBox fields are too long");
                }
            }
            else
            {
                MessageBox.Show("One or more textBox fields are empty");
            }
        }

        private void EditMechanik_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
```

```
            {
                _mechanicWindowModel.MechanicModel.Name = MechanikName_TextBox.Text;
                _mechanicWindowModel.MechanicModel.SecondName = MechanikSecondName_TextBox.Text;
                _mechanicWindowModel.MechanicModel.PhoneNumber = MechanikPhone_TextBox.Text;
                _mechanicWindowModel.MechanicModel.WorkExperience = MechanikExperience_TextBox.Text;
                _mechanicWindowModel.MechanicModel.LoginInfo.Login = MechanicLogin_TextBox.Text;
                _mechanicWindowModel.MechanicModel.LoginInfo.Password = MechanicPassword_TextBox.Text;
                _mechanicWindowModel.EditMechanic();
                CleanInputs();
                UpdateData();
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }
    }

    private void AddMechanik_Button_Click(object sender, RoutedEventArgs e)
    {
        if (CheckInputs())
        {
            if (CheckForSize())
            {
                var mechanic = new MechanicModel()
                {
                    Name = MechanikName_TextBox.Text,
                    SecondName = MechanikSecondName_TextBox.Text,
                    PhoneNumber = MechanikPhone_TextBox.Text,
                    WorkExperience = MechanikExperience_TextBox.Text,
                    LoginInfo = new LoginInfoModel()
                    {
                        Login = MechanicLogin_TextBox.Text,
                        Password = MechanicPassword_TextBox.Text,
                        UserType = "mechanik"
                    }
                };
                _mechanicWindowModel.AddMechanicToDb(mechanic);
                CleanInputs();
                UpdateData();
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }
    }

    public bool CheckInputs()
    {
        bool result;
        if (MechanikName_TextBox.Text != "" && MechanikSecondName_TextBox.Text != "" &&
MechanikPhone_TextBox.Text != "" &&
            MechanikExperience_TextBox.Text != "" && MechanicLogin_TextBox.Text != "" &&
MechanicPassword_TextBox.Text != "")
        {
            result = true;
        }
        else
```

```
            {
                result = false;
            }
            return result;
        }

        public bool CheckForSize()
        {
            bool result;
            if (MechanikName_TextBox.Text.Length < 50 && MechanikSecondName_TextBox.Text.Length < 50 &&
MechanikPhone_TextBox.Text.Length < 15
                && MechanikExperience_TextBox.Text.Length < 30 && MechanicLogin_TextBox.Text.Length < 50 &&
MechanicPassword_TextBox.Text.Length < 50)
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public void UpdateData()
        {
            Mechanik_DataStorage.ItemsSource = _mechanicWindowModel.GetMechanics();
        }

        public void CleanInputs()
        {
            MechanikName_TextBox.Text = "";
            MechanikSecondName_TextBox.Text = "";
            MechanikPhone_TextBox.Text = "";
            MechanikExperience_TextBox.Text = "";
            MechanicLogin_TextBox.Text = "";
            MechanicPassword_TextBox.Text = "";
        }


    }
}

namespace Diplom.MechanicFlow
{
    public class MechanicWindowModel
    {
        public MechanicModel MechanicModel = new MechanicModel();
        private readonly DbWorker _worker = new DbWorker();

        public List<MechanicModel> GetMechanics()
        {
            var result = _worker.GetMechanics();
            return result;
        }

        public void AddMechanicToDb(MechanicModel mechanicModel)
        {
            _worker.AddMechanicToDb(mechanicModel);
        }

        public void DeleteMechanicFromDb()
        {
            _worker.DeleteMechanicFromDb(MechanicModel);
        }

        public void EditMechanic()
        {
            _worker.EditMechanic(MechanicModel);
        }
```

```
    }
}

namespace Diplom
{
    public partial class ClientWindow : Window
    {
        readonly ClientWindowModel _clientWindowModel = new ClientWindowModel();
        public ClientWindow()
        {
            InitializeComponent();
            LoadData();
        }
        private void ClientWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            LoadData();
        }

        public void LoadData()
        {
            var worker = new DbWorker();
            Client_DataStorage.ItemsSource = null;
            Client_DataStorage.ItemsSource = _clientWindowModel.GetClients();
        }


        private void AddClient_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
                {
                    var client = new ClientModel()
                    {
                        Name = ClientName_TextBox.Text,
                        SecondName = ClientSecondName_TextBox.Text,
                        PhoneNumber = ClientPhone_TextBox.Text
                    };
                    _clientWindowModel.AddClientToDb(client);
                    CleanInputs();
                    UpdateData();
                }
                else
                {
                    CleanInputs();
                    MessageBox.Show("One or more texBox fields are too long");
                }
            }
            else
            {
                MessageBox.Show("One or more textBox fields are empty");
            }

        }

        private void EditClient_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
                if (CheckForSize())
                {

_clientWindowModel.GetDataForModel(ClientName_TextBox.Text,ClientSecondName_TextBox.Text,ClientPhone_TextBox.T
ext);
                    _clientWindowModel.EditClient();
                    UpdateData();
                    CleanInputs();
```

```
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }
    }

    private void DeleteClient_Button_Click(object sender, RoutedEventArgs e)
    {
        if (CheckInputs())
        {
            if (CheckForSize())
            {
                _clientWindowModel.DeleteClient();
                UpdateData();
                CleanInputs();
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }
    }

    private void Client_DataStorage_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        if (Client_DataStorage.SelectedItem == null)
        {
            return;
        }
        _clientWindowModel.ClientModel = Client_DataStorage.SelectedItem as ClientModel;
        ClientName_TextBox.Text = _clientWindowModel.ClientModel.Name;
        ClientSecondName_TextBox.Text = _clientWindowModel.ClientModel.SecondName;
        ClientPhone_TextBox.Text = _clientWindowModel.ClientModel.PhoneNumber;
        ClientCars_ComboBox.ItemsSource = null;
        Cars_ComboBox.ItemsSource = null;
        ClientCars_ComboBox.ItemsSource = _clientWindowModel.GetClientCars();
        Cars_ComboBox.ItemsSource = _clientWindowModel.GetAllAvailableCars();
    }

    public void UpdateData()
    {
        Client_DataStorage.ItemsSource = _clientWindowModel.GetClients();
        ClientCars_ComboBox.SelectedItem = null;
        ClientCars_ComboBox.ItemsSource = null;
        Cars_ComboBox.SelectedItem = null;
        Cars_ComboBox.ItemsSource = null;
    }

    public void CleanInputs()
    {
        ClientName_TextBox.Text = "";
        ClientSecondName_TextBox.Text = "";
        ClientPhone_TextBox.Text = "";
    }

    public bool CheckInputs()
    {
```

```
            bool result;
            if (ClientName_TextBox.Text != "" && ClientSecondName_TextBox.Text != "" && ClientPhone_TextBox.Text != "")
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public bool CheckForSize()
        {
            bool result;
            if (ClientName_TextBox.Text.Length < 50 && ClientSecondName_TextBox.Text.Length < 50 &&
                ClientPhone_TextBox.Text.Length < 15)
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        private void ClientCars_ComboBox_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            CarModel carModel = ClientCars_ComboBox.SelectedItem as CarModel;
        }

        private void Delink_Button_Click(object sender, RoutedEventArgs e)
        {
            if (ClientCars_ComboBox.SelectedItem == null)
            {
                return;
            }
            var car = ClientCars_ComboBox.SelectedItem as CarModel;
            _clientWindowModel.RemoveCarFromClient(car);
            UpdateData();
            CleanInputs();
        }

        private void Link_Button_Click(object sender, RoutedEventArgs e)
        {
            if (Cars_ComboBox.SelectedItem == null)
            {
                return;
            }
            var car = Cars_ComboBox.SelectedItem as CarModel;
            _clientWindowModel.AddCarToClient(car);
            UpdateData();
            CleanInputs();
        }


    }
}
namespace Diplom.ClientFlow
{
    public class ClientWindowModel
    {
        public ClientModel ClientModel = new ClientModel();
        private readonly DbWorker _worker = new DbWorker();

        public void GetDataForModel(string name,string secondName,string phone)
        {
            ClientModel.Name = name;
```

```
            ClientModel.SecondName = secondName;
            ClientModel.PhoneNumber = phone;
        }

        public List<ClientModel> GetClients()
        {
            return _worker.GetClients();
        }

        public void AddClientToDb(ClientModel clientModel)
        {
            _worker.AddClientToDb(clientModel);
        }

        public void EditClient()
        {
            _worker.EditClient(ClientModel);
        }

        public void DeleteClient()
        {
            _worker.DeleteClientFromDb(ClientModel);
        }

        public List<CarModel> GetClientCars()
        {
            return _worker.GetClientCars(ClientModel);
        }

        public List<CarModel> GetAllAvailableCars()
        {
            return _worker.GetAllAvailableCars(ClientModel);
        }

        public void RemoveCarFromClient(CarModel carModel)
        {
            _worker.RemoveCarFromClient(ClientModel,carModel);
        }

        public void AddCarToClient(CarModel carModel)
        {
            _worker.AddCarToClient(ClientModel,carModel);
        }
    }
}
namespace Diplom
{
    public partial class CarWindow : Window
    {
        private readonly CarWindowModel _carWindowModel = new CarWindowModel();
        public CarWindow()
        {
            InitializeComponent();
        }
        private void CarWindow_OnLoaded(object sender, RoutedEventArgs e)
        {
            LoadData();
        }

        public void LoadData()
        {
            Car_DataStorage.ItemsSource = null;
            Car_DataStorage.ItemsSource = _carWindowModel.GetCars();
        }

        private void AddCar_Button_Click(object sender, RoutedEventArgs e)
        {
            if (CheckInputs())
            {
```

```
            if (CheckForSize())
            {
                var car = new CarModel()
                {
                    Mark = CarMark_TextBox.Text,
                    Model = CarModel_TextBox.Text,
                    Year = CarYear_TextBox.Text,
                    LastTo = CarLastTO_TextBox.Text
                };
                _carWindowModel.AddCarToDb(car);
                CleanInputs();
                UpdateData();
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }
    }

    private void EditCar_Button_Click(object sender, RoutedEventArgs e)
    {
        if (CheckInputs())
        {
            if (CheckForSize())
            {

_carWindowModel.GetDataForModel(CarMark_TextBox.Text,CarModel_TextBox.Text,CarYear_TextBox.Text,CarLastTO_Te
xtBox.Text);
                _carWindowModel.EditCar();
                CleanInputs();
                UpdateData();
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
        }
    }

    private void DeleteCar_Button_Click(object sender, RoutedEventArgs e)
    {
        if (CheckInputs())
        {
            if (CheckForSize())
            {
                _carWindowModel.DeleteCar();
                CleanInputs();
                UpdateData();
            }
            else
            {
                CleanInputs();
                MessageBox.Show("One or more texBox fields are too long");
            }
        }
        else
        {
            MessageBox.Show("One or more textBox fields are empty");
```

```
            }

        }

        private void Car_DataStorage_OnSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (Car_DataStorage.SelectedItem == null)
            {
                return;
            }
            _carWindowModel.CarModel = Car_DataStorage.SelectedItem as CarModel;
            CarMark_TextBox.Text = _carWindowModel.CarModel.Mark;
            CarModel_TextBox.Text = _carWindowModel.CarModel.Model;
            CarYear_TextBox.Text = _carWindowModel.CarModel.Year;
            CarLastTO_TextBox.Text = _carWindowModel.CarModel.LastTo;
        }
        public void UpdateData()
        {
            Car_DataStorage.ItemsSource = _carWindowModel.GetCars();
        }
        public bool CheckForSize()
        {
            bool result;
            if (CarMark_TextBox.Text.Length < 30 && CarModel_TextBox.Text.Length < 30 &&
                CarYear_TextBox.Text.Length < 15 && CarLastTO_TextBox.Text.Length < 30)
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public bool CheckInputs()
        {
            bool result;
            if (CarMark_TextBox.Text != "" && CarModel_TextBox.Text != "" && CarYear_TextBox.Text != "" &&
                CarLastTO_TextBox.Text != "")
            {
                result = true;
            }
            else
            {
                result = false;
            }
            return result;
        }

        public void CleanInputs()
        {
            CarMark_TextBox.Text = "";
            CarModel_TextBox.Text = "";
            CarYear_TextBox.Text = "";
            CarLastTO_TextBox.Text = "";
        }

    }
}
namespace Diplom.CarFlow
{
    public class CarWindowModel
    {
        public CarModel CarModel = new CarModel();
        private readonly DbWorker _worker = new DbWorker();

        public List<CarModel> GetCars()
        {
```

```
            return _worker.GetCars();
        }

        public void AddCarToDb(CarModel carModel)
        {
            _worker.AddCarToDb(carModel);
        }
        public void EditCar()
        {
            _worker.EditCar(CarModel);
        }

        public void DeleteCar()
        {
            _worker.DeleteCarFromDb(CarModel);
        }

        public void GetDataForModel(string mark, string model, string year, string lastTo)
        {
            CarModel.Mark = mark;
            CarModel.Model = model;
            CarModel.Year = year;
            CarModel.LastTo = lastTo;
        }

    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЦЕСАМИ ВИКОНАННЯ ЗАМОВЛЕНЬ КЛІЄНТІВ НА СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Програма та методика тестування

КПІ.ІТ-8206.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Валерій НОВІНСЬКИЙ

Нормоконтроль:
_____ Валерій НОВІНСЬКИЙ

Виконавець:
_____ Сергій БОГАЧЕНКО

Київ – 2022

Зміст

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є програмне забезпечення, що керує процесами виконання замовлень клієнта на станції технічного обслуговування. Додаток створений на платформі .NET з використанням технологій WPF та Entity Framework.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення у відповідно до функціональних вимог;

– перевірка на правильність введених даних користувачем;

– запобігання некоректних дій користувача;

– знаходження проблем та недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється уся документація, яка аналізується на предмет дотримання стандартів програмування;

– динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на безлічі тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми в роботі програми;

– тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;

– інтеграційне тестування;

– тестування продуктивності програмного забезпечення;

– тестування інтерфейсу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується вручну, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності в користуванні. Для того, щоб перевірити працездатність та відмовостійкість додатку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на мобільних пристроях з різною роздільною здатністю екрану;

– тестування на мобільних пристроях з різною версією операційної системи;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування зміни орієнтації екрану;

– тестування працездатності програми у випадку відсутності з’єднання до мережі;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЦЕСАМИ ВИКОНАННЯ ЗАМОВЛЕНЬ КЛІЄНТІВ НА СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Керівництво користувача

КПІ.ІТ-8206.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Валерій НОВІНСЬКИЙ

Нормоконтроль:
_____ Валерій НОВІНСЬКИЙ

Виконавець:
_____ Сергій БОГАЧЕНКО

Київ – 2022

Зміст

1 ЗАГАЛЬНІ ВІДОМОСТІ

Система управління замовлень використовується для управління замовленнями клієнтів на СТО. Додаток має функціональний та простий для розуміння інтерфейс.

Функціонал програми є:

- додавання нового замовлення;
- змінення існуючого замовлення;
- видалення обраного замовлення;
- отримання даних про всі замовлення;
- додавання, змінення, видалення клієнта;
- додавання, змінення, видалення запчастини;
- додавання, змінення, видалення сервісів;
- додавання, змінення, видалення механіків;
- додавання, змінення, видалення автомобілів.

2 ПІДГОТОВКА ДО РОБОТИ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- наявність комп’ютера або ноутбука;
- операційна система Windows 7 і новіше;
- наявність локального SQL Server;
- достатньо вільної пам’яті на пристрої для встановлення додатка;
- процесор з тактовою частотою 1 GHz і вище;
- оперативна пам’ять 2 Gb і більше.

2.2 Завантаження застосунку

Щоб почати роботу з програмним забезпеченням, потрібно запустити відповідний .exe файл(StoService.exe). Якщо після цього з’явиться вікно для логіну в систему то програма встановлена успішно.

3 РОБОТА ІЗ ЗАСТОСУНКОМ

Після успішного встановлення програми на екран виведеться вікно для авторизації користувача, що зображене на рисунку 3.1.

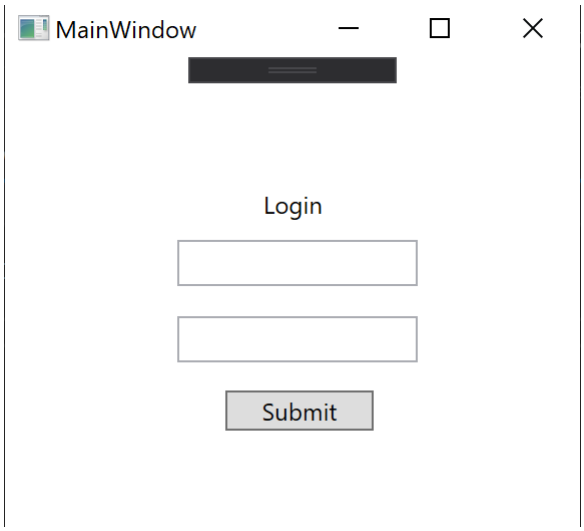


Рисунок 3.1 – Початкова сторінка додатку

Користувач повинен ввести свій логін та пароль щоб увійти до системи. Якщо авторизація була неуспішною то виведеться повідомлення, що зображене на рисунку 3.2

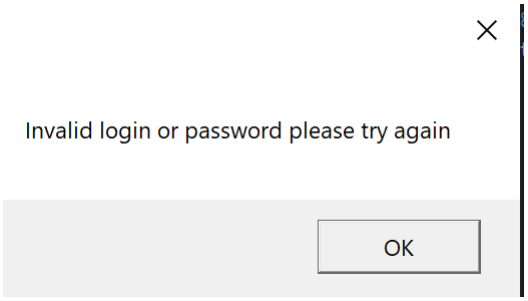


Рисунок 3.2 – Вікно з повідомлення про невдачу

Після закриття вікна з повідомленням всі поля вводу будуть очищені для повторної спроби авторизуватись.

Далі як авторизація була успішна, буде представлене головне вікно програми, яке зображене на рисунку 3.3.

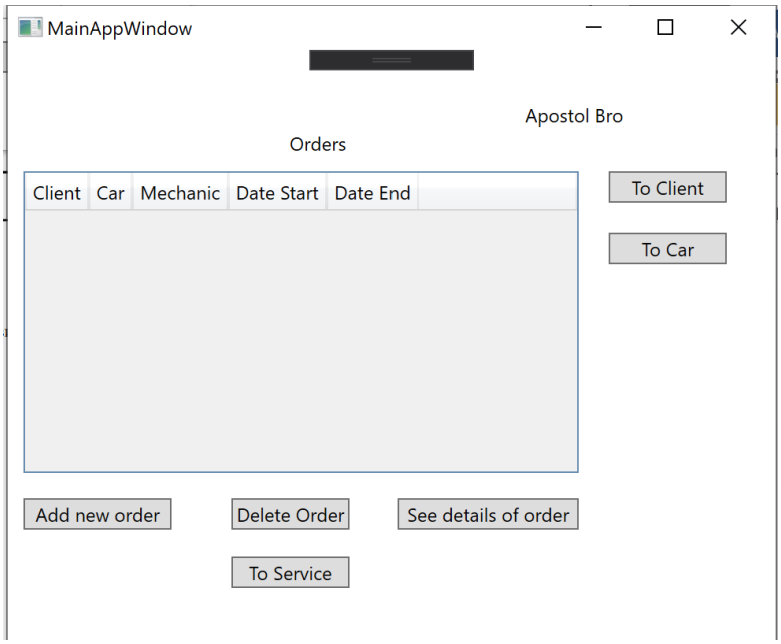


Рисунок 3.3 – Головна сторінка додатку

Залежно від типу користувача буде представлений відповідний функціонал.

Далі будуть наведені 2 рисунка для відображення двох можливих варіантів інтерфейсу відповідно до типу користувача.

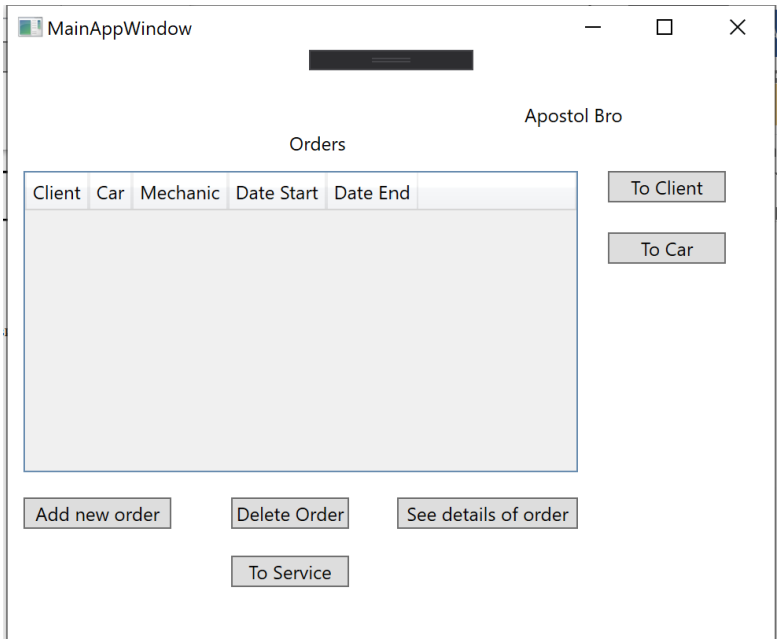


Рисунок 3.4 – Головна сторінка додатку для Механіка

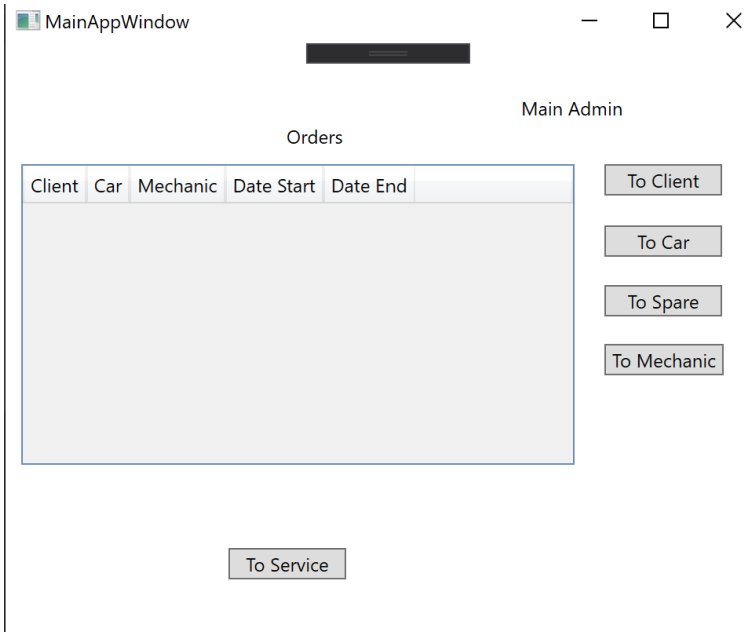


Рисунок 3.5 – Головна сторінка додатку для Адміністратора

Далі розглянемо приклад використання додатку механіком.
Для того щоб додати клієнта треба натиснути кнопку «Додати клієнта», щоб перейти до відповідного вікна, яке зображене на рисунку 3.6.

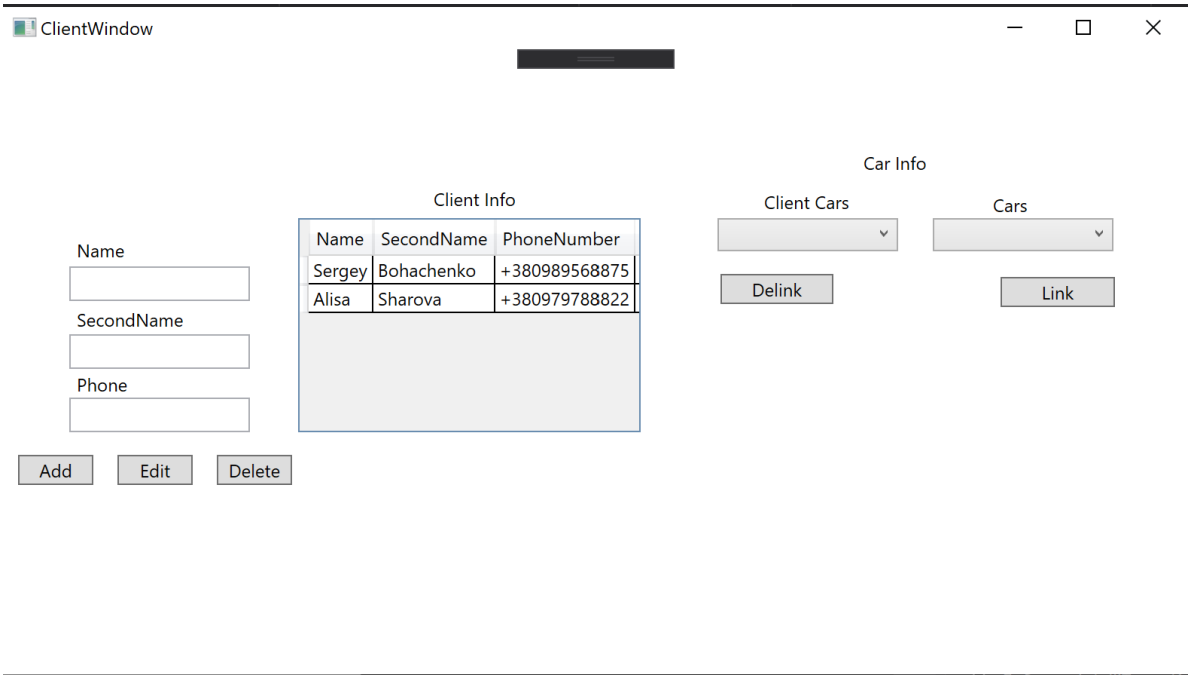


Рисунок 3.6 – Вікно для роботи з клієнтом

Далі, щоб додати нового клієнта потрібно заповнити всі поля ти вибрати всі елементи. Якщо хоч одне поле буде не заповнене то виведеться повідомлення з зауваженням, яке представлене на рисунку 3.7.

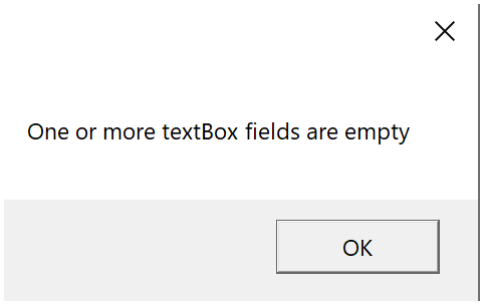
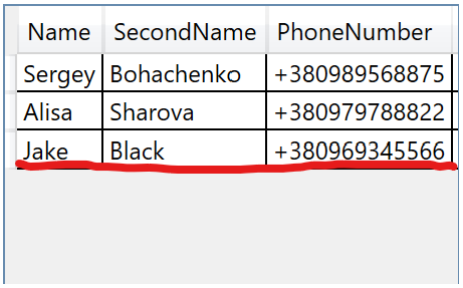


Рисунок 3.7 – Вікно з повідомленням про невдачу

Якщо всі поля були заповнені то після натискання кнопки «Додати», клієнт буде доданий до бази даних. Після додавання всі поля будуть очищені

для повторного використання та в таблиці всіх клієнтів буде доданий новий клієнт, що і продемонстровано на рисунку 3.8.



Name	SecondName	PhoneNumber
Sergey	Bohachenko	+380989568875
Alisa	Sharova	+380979788822
Jake	Black	+380969345566

Рисунок 3.8 – Результат додавання нового клієнта

Наведений вище приклад є лише одним із багатьох можливих варіантів використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЦЕСАМИ ВИКОНАННЯ ЗАМОВЛЕНЬ КЛІЄНТІВ НА СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Графічні матеріали

КПІ.ІТ-8206.045490.07.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Валерій НОВІНСЬКИЙ

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Сергій БОГАЧЕНКО

Київ – 2022

КПІ.ІТ-8206.045490.07.99ССД

Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-8206.045490.07.99ССД		
Розроб.		Богаченко С.В			Схема струтурна діяльності	Лит.	Арк.	Аркушів
Перев.		Новінський В.П					1	
Т. Кон.						Аркуш	Аркушів	
Н. Кон.		Ліщук К.І.			Програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування	КПІ ім.Ігоря Сікорського Кафедра ФІОТ гр. ІТ-82		
Затв.		Новінський В.П						

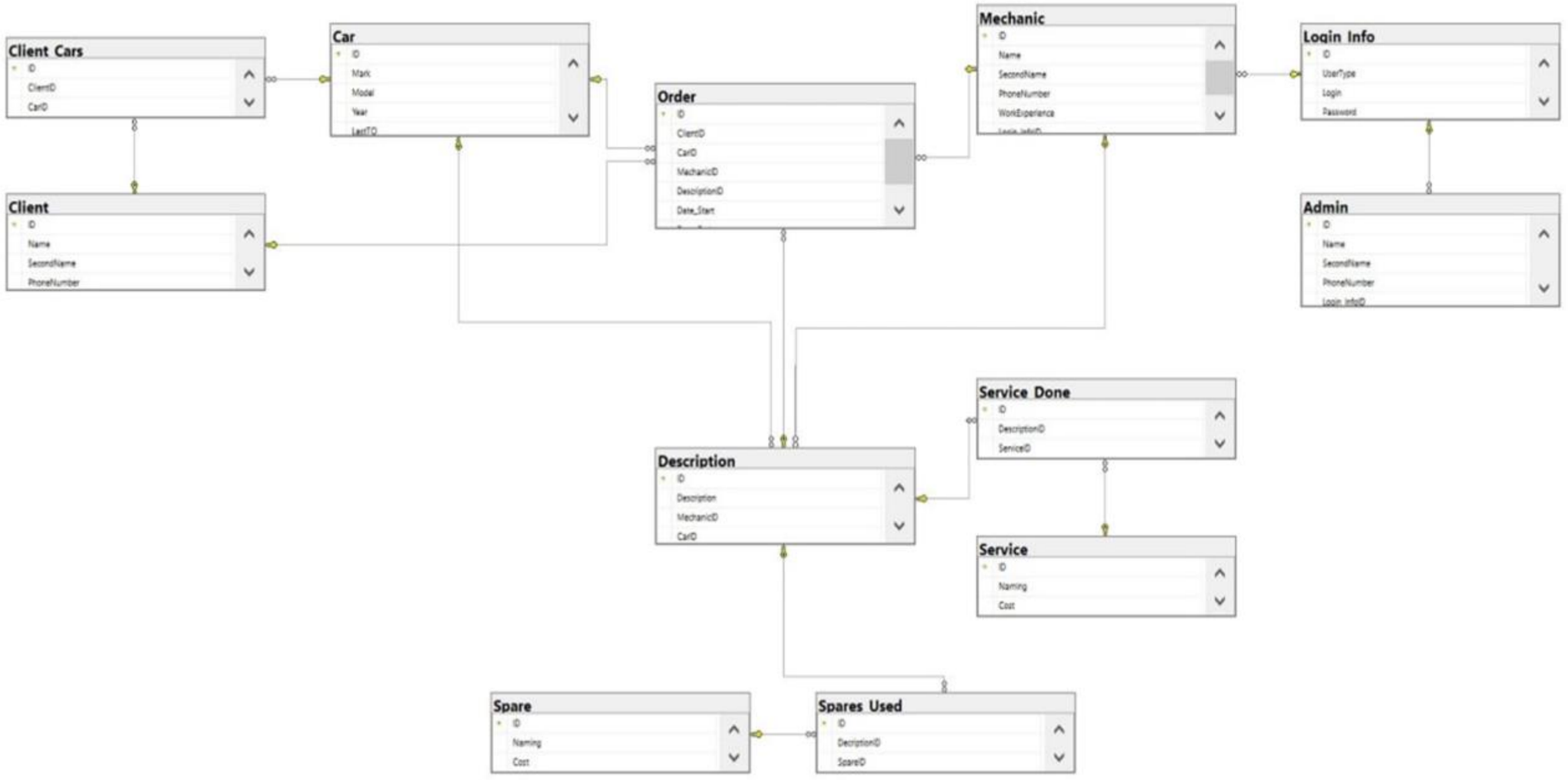
Інв. № підл. | Підп. дата | Інв. № взам. підл. | Інв. № дубл. | Підп. дата

КПІ.ІТ-8206.045490.07.99СБП

Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-8206.045490.07.99СБП		
Розроб.		Богаченко С.В			Схема бізнес-процесів	Лит.	Арк. / Аркушів: 1
Перев.		Новінський В.П				Аркуш	Аркушів
Т. Кон.					Програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування	КПІ ім.Ігоря Сікорського Кафедра ФІОТ гр. ІТ-82	
Н. Кон.		Ліщук К.І.					
Затв.		Новінський В.П					

Інв. № підл. | Підп. дата | Інв. № взам. підл. | Інв. № дубл. | Підп. дата

КПІ.ІТ-8206.045490.07.99СБД



					КПІ.ІТ-8206.045490.07.99СБД		
						Літера	Маса Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема бази даних		
Розробив		Богаченко С.В				Аркуш	Аркушів
Перевірив		Новінський В.П					
Т. кон.							
Н. кон.		Ліщук К.І.			Програмне забезпечення для управління процесами виконання замовлень клієнтів на станції технічного обслуговування	КПІ ім.Ігоря Сікорського Кафедра ФІОТ гр. ІТ-82	
Затвердив		Новінський В.П					