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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка застосунку для обліку колекції автомобілів мовою C#»

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

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

_____ Даниїл ПОПЕРЕШНЯК
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Даниїл ПОПЕРЕШНЯК

Керівник: Ігор ГАМАНЮК
старший викладач кафедри ІПЗ

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Поперешняку Даниїлу Ігоровичу

1. Тема кваліфікаційної роботи: «Розробка застосунку для обліку колекції автомобілів мовою C#»
керівник кваліфікаційної роботи старший викладач кафедри ІПЗ Ігор ГАМАНЮК, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

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

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про облік колекції автомобілів, опис даних необхідних для обліку.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд та аналіз існуючих методів та технологій обліку колекції автомобілів.
2. Проектування застосунку для обліку колекції автомобілів.
3. Програмна реалізація та опис функціонування застосунку обліку колекції

автомобілів.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма діяльності.
6. Діаграма класів.
7. Екранні форми, ч.1.
8. Екранні форми, ч.2.
9. Екранні форми, ч.3.
10. Відео-демонстрація роботи застосунку.
11. Апробація результатів дослідження.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих методів та технологій обліку автомобілів	14.03-17.03.2024	
4	Проектування застосунку для обліку колекції автомобілів	18.03-26.03.2024	
5	Програмна реалізація та опис функціональної частини застосунку для обліку колекції автомобілів	27.03-15.04.2024	
6	Тестування застосунку	16.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Даниїл ПОПЕРЕШНЯК
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ГАМАНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 40 стор., 8 табл., 32 рис., 12 джерел.

Мета роботи – спростити процес обліку колекції автомобілів шляхом впровадження застосунку для обліку колекції автомобілів.

Об'єкт дослідження – процес обліку колекції автомобілів.

Предмет дослідження – застосунок для обліку колекції автомобілів.

Короткий зміст роботи: В роботі проаналізовано потреба в застосунку для обліку колекції автомобілів. Проаналізовано застосунки аналоги, такі як: Collector Notepad, iCollect Everything.

Розроблено алгоритм роботи застосунку та програмно реалізовано ключові функціональні можливості, такі як: створення колекцій, опис колекції, додавання зображення до колекції, сортування колекції, додавання автомобілів, сортування автомобілів, редагування вже доданих автомобілів, перегляд детальної інформації про автомобіль.

Проведено тестування застосунку. В роботі використано середовище розробки Visual Studio, мову програмування C#, графічну підсистему WPF, базу даних SQLite, застосунок для створення прототипів Figma, та хмарний інструмент для зберігання, редагування та поширення коду GitHub.

Сферою використання застосунку є облік власної колекції автомобілів.

КЛЮЧОВІ СЛОВА: ОБЛІК КОЛЕКЦІЇ АВТОМОБІЛІВ, C#, WPF.

ЗМІСТ

ВСТУП

У сучасному світі де майже кожен мріє про якесь авто, або декілька автомобілів постає питання про необхідність впорядкування власних вже наявних або майбутніх авто. Проте наявні застосунки для обліку колекцій, або не мають відповідної категорії, або мають дуже обмежений функціонал. Крім того технології не стоять на місці, постійно відкриваючі нові можливості для покращення застосунків чи процесів.

Розроблюваний застосунок має спростити процес обліку колекції автомобілів, надаючи користувачам можливості до створення власних колекцій з власними зображеннями, додавати автомобілі та інформацію про них, прикладаючи власне зображення.

Мета роботи – спростити процес обліку колекції автомобілів шляхом впровадження застосунку для обліку колекції автомобілів.

Об’єкт дослідження – процес обліку колекції автомобілів.

Предмет дослідження – застосунок для обліку колекції автомобілів.

Завдання роботи:

1) Проаналізувати існуючі засоби для обліку колекції автомобілів.

2) Скласти вимоги застосунку для обліку колекції автомобілів

3) Вибрати інструменти розробки.

4) Спроектувати застосунок.

5) Розробити застосунок для обліку колекції автомобілів.

6) Провести тестування.

Результатами даної роботи можуть користуватись власники автомобілів, чи просто люди які зацікавлені у впорядкуванні автомобілів які їм подобаються.

1 АНАЛІЗ РІШЕНЬ ДЛЯ ОБЛІКУ КОЛЕКЦІЇ АВТОМОБІЛІВ

1.1 Постановка задачі щодо обліку колекції автомобілів

Облік колекції – це зручний спосіб впорядкувати власну колекцію. Види обліку бувають різні: письмово та за допомогою застосунків. Письмовий спосіб – це введення обліку за допомогою письмових засобів. Облік колекції будь-чого, дуже розповсюджене у людей, адже зручніше мати впорядковану систему власної колекції, ніж тримати все у голові або у записниках, де важко шукати щось конкретне.

Вимоги до вирішення задач:

– Легкий доступ додавання.
– Можливість категоризації.
– Можливість переглянути більш детальну інформацію про авто.

Цільовою аудиторію – є власники автомобілів, автомобільні ентузіасти, та можливо люди які планують та візуалізують власні бажання. Для цільової аудиторії задача може бути вирішена різними методами, такими як: фізичний нотатник, журнал, блокнот, тощо – можна використати для обліку, але є можливість втрати, випадкового пошкодження. Але для деяких людей це може бути більш зручний спосіб, адже є різні вікові категорії та навички користування гаджетами. Та використання профільних застосунків – найбільш зручний спосіб, адже необхідно просто заповнити відповідні поля і до вашої колекції буде додано заповнений елемент.

Розробка даного застосунку для обліку колекції автомобілів відбувалась у декілька етапів, таких як – створення прототипу, проектування застосунку, розробка застосунку.

1.2 Огляд інструментальних засобів для обліку колекції автомобілів

Серед існуючих інструментальних засобів для обліку колекції автомобілів, є такі як: iCollect Everything та Collector Notepad. Детальніше про кожен з них описано у підрозділах нижче.

1.2.1. iCollect Everything

iCollect Everything – застосунок для каталогізації будь-яких предметів колекціонування, які ви можете придумати. Застосунок наявний на всіх ОС, але має обмежений функціонал, і в застосунку для персонального комп’ютера лише платна версія, навіть для перегляду вже існуючої колекції.

Переваги:

- Великий базовий функціонал;
- Велика кількість базових категорій;
- Можливість додати фото;

Недоліки:

- Відсутній базова категорія для автомобілів;
- Для персонального комп’ютера тільки платна версія;

Для створення колекції треба вибрати необхідну категорію, додавати до неї елементи заповнюючи необхідні розділи. Приклад вигляду головного меню застосунку рис 1.1, та налаштування колекції та вигляду клітинку рис 1.2.

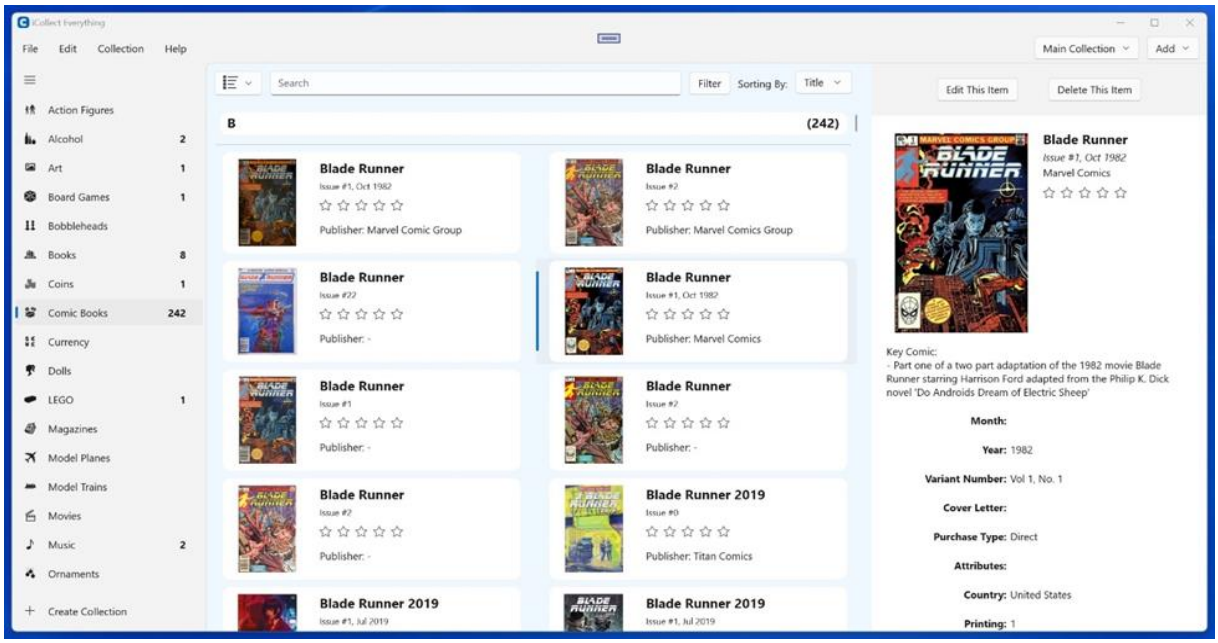


Рис. 1.1 Головне меню ICollect Everything

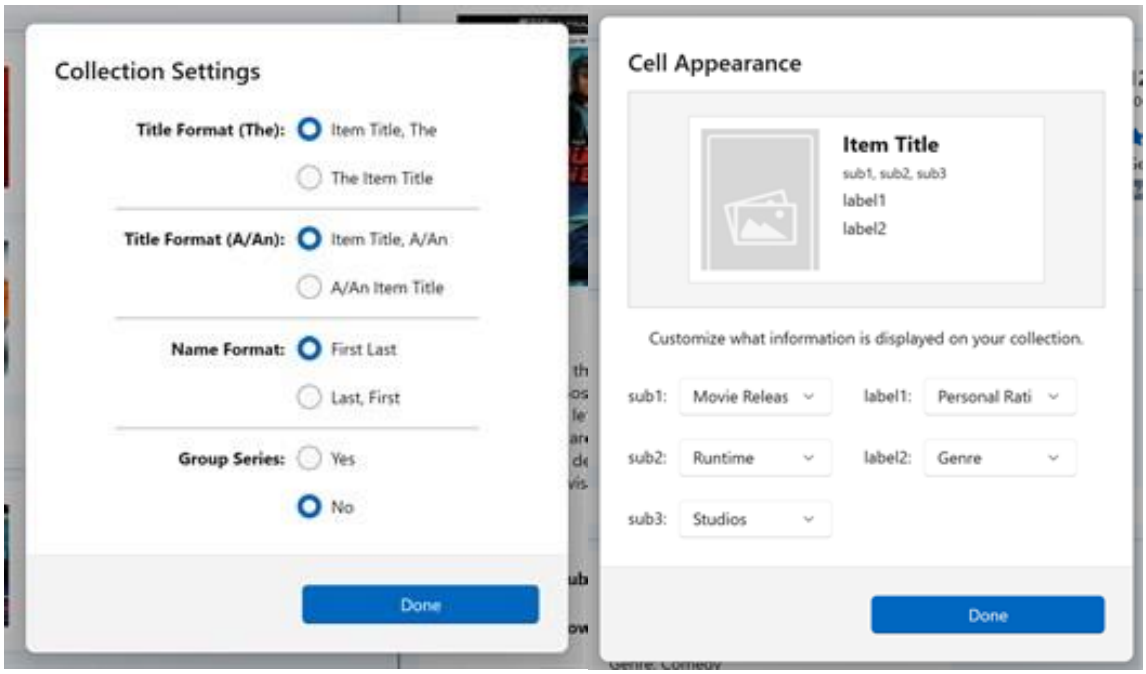


Рис. 1.2 Створення колекції та налаштування вигляду клітинки

1.2.3. Collector Notepad

Collector Notepad - безкоштовна програма для каталогізації вашої колекції. Дозволяє впорядковувати та катологізувати вашу колекцію за допомогою базових інструментів.

Застосунок безкоштовний та наявний тільки для персональних комп'ютерів на операційній системі Windows.

Переваги:

- Можна додати детальний опис у поле "Comment".
- Показує як буде виглядати зображення до збереження.

Недоліки:

- Відсутні будь які поля окрім «Title».
- Застарілий застосунок.
- Нема підказок для заповнення полів.
- Відсутність локалізації.
- Відсутність додавання власного зображення.

Для початку необхідно створити файл та надати йому ім'я, це і буде назва колекції. Потім необхідно натиснути на додавання елементу, ввести назву, дату додавання, а також є можливість додати 3 фото, та детальний опис чи коментар.

Приклад перегляду колекції на рис 1.3, додавання або редагування елементу колекцію рис 1.4

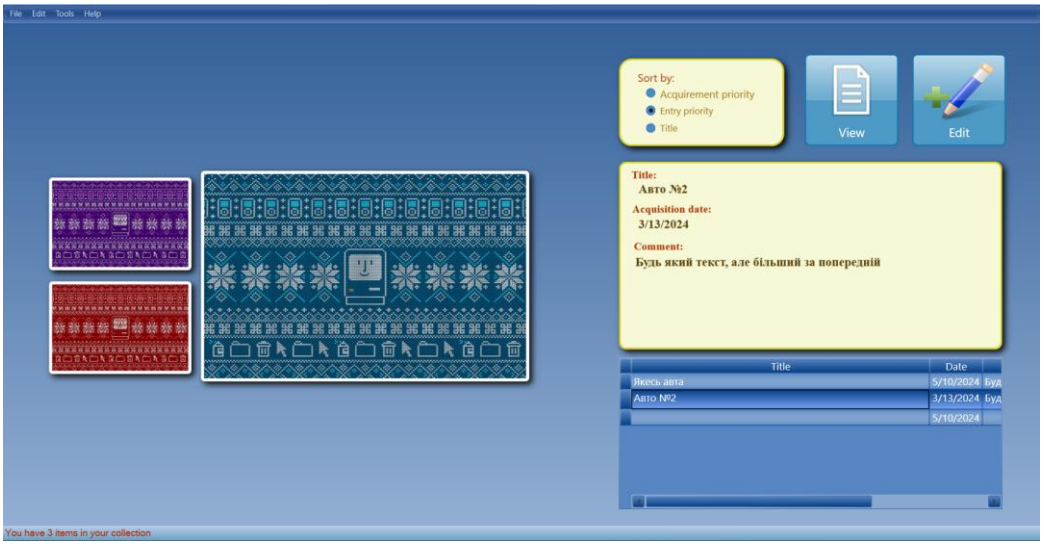


Рис 1.3 Приклад перегляду колекції у застосунку Collector Notepad

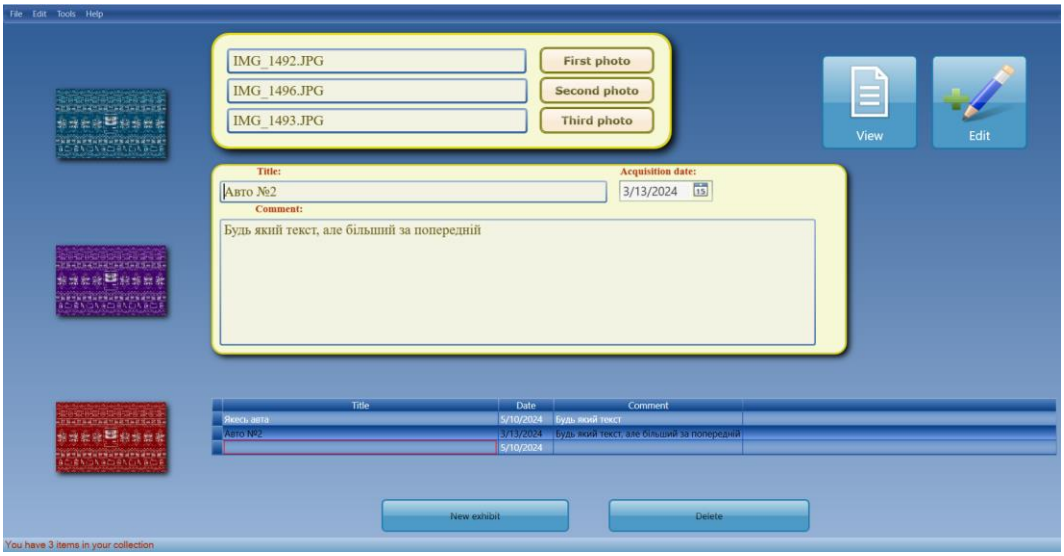


Рис. 1.4 Приклад додавання елементу у застосунку Collector Notepad

1.2.4. Порівняння

Нижче наведено порівняльну таблицю, у яку додано переваги та недоліки кожного застосунку, а саме iCollect Everything, Collector Notepad та мого застосунку.

З результатів таблиці ми можемо помітити переваги нашого застосунку, а саме створення власних колекцій з власним фото, перегляд детальної інформації.

Таблиця 1.1

Таблиця порівняння аналогів

Характеристики	Collector Notepad	iCollect Everything	My App
Створення власних колекцій	+	-	+
Детальна інформація	-	+	+
Сортування автомобілів	+	-	+

Продовження таблиці 1.1

Таблиця порівняння аналогів

Характеристики	Collector Notepad	iCollect Everything	My App
Додавання фото	-	+	+
Сортування колекцій	+	-	+
Редагування вже доданого автомобіля	-	+	+

1.3 Інструментальні засоби розробки застосунку для обліку колекції автомобілів

Для реалізації додатку для обліку колекції автомобілів в якості мови програмування обрано C#. C# - це проста, сучасна, об'єктно-орієнтована мова програмування з безпекою типів, яка поєднує в собі високу продуктивність мов швидкої розробки додатків з необробленою потужністю C і C++.[1] Основною перевагою цією мови є гарна інтеграція з ОС Windows, дозволяючи використовувати великий спектр інструментів, бібліотек та засобів. Також вона є однією з найбільш популярних мов у світі, тому має велику спільноту, документацію, що в свою чергу полегшує розробку, адже завжди можна скористатись допомогою форумів, чи групами спільноти.

Для написання, буде використано інтегроване середовище розробки Visual Studio - найповніше середовище розробки для розробників .NET та C++ під Windows. Повністю укомплектована чудовим набором інструментів і функцій для покращення та вдосконалення кожного етапу розробки програмного забезпечення.[2]. Він має великий базовий функціонал, гарну інтеграцію з іншими

службами Microsoft, такими як Windows Presentation Foundation (далі WPF). WPF - це середовище розробки, яке використовується для створення десктопних додатків. WPF має незалежний від роздільної здатності векторний рушій рендерингу, який допомагає працювати з сучасним графічним обладнанням. Остання версія WPF - 4.6. У цьому фреймворку інтерфейс програми розроблений на мові XAML, а логіка програми написана на мові програмування C#.[3] Основні переваги – це легко стилізація, базовані інструменти дозволяють стилізувати та кастомізувати вигляд елементів керування, також там є наявні шаблони. Також можна створювати анімації, ефекти та інші візуальні засоби покращення взаємодії з додатком, що збільшує перевагу над Windows Form.

Для створення бази даних, обрано SQLite – це невелика, швидка, вбудована база даних. Її популярність пояснюється поєднанням механізму та інтерфейсу бази даних в одній бібліотеці, а також можливістю зберігати всі дані в одному файлі. За функціональністю вона знаходиться між MySQL та PostgreSQL, проте є швидшою за обидві бази даних., адже вона має високу доступність.[4] Система автоматично зберігає резервну копію, для запобігання повної втрати даних, оптимізовані обчислювальні ресурси надають високу продуктивність та швидкодію.

Figma – інструмент для створення макету/прототипу майбутнього застосунку, дозволяє взаємодіяти разом з командою в одному проекті, що полегшує роботу в команді, адже можна зобразити майбутнє бачення. Чому саме Figma: співпраця в режимі реального часу, Figma дозволяє дизайнерам та іншим членам команди працювати одночасно в режимі реального часу, що виводить спільний робочий процес на абсолютно новий рівень. Ця потужна функція виділяє Figma серед інших інструментів, оскільки вона покращує не лише роботу над дизайном, але й сам процес командної співпраці. «Співпраця - це важко. Ми робимо її легшою» - це один з основоположних принципів Figma. Рішення «все в одному»: Figma вдалося об'єднати цілий набір інструментів для проектування, щоб створити універсальне рішення. Figma охоплює практично все, що потрібно для створення складного інтерфейсу, від мозкового штурму та фреймворку до

створення прототипів та обміну ресурсами. [5, с 5]

Для зберігання, та віддаленої роботи з кодовою частиною було обрано GitHub – хмарний кросс-платформенний інструмент, для зберігення, спільного редагування та поширення коду.[6] Основні переваги:

– Система керування версіями – надає можливість безпечного оновлення коду застосунку, адже завжди можна буде повернутись на безпечну та стабільну версію.

– Клонування гілок – можна розгалужити основну гілку для введення нової функції, а потім об’єднати їх і опублікувати зміни. Розгалужена гілка дозволяє спробувати нові функції, не змінюючи основного коду та не піддаючи ризику застосунок.

– Спільна робота та опис проблем – можна відмітити елемент у коді, або задачу окремо, для її вирішення чи просто обговорення.

2 ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОБЛІКУ КОЛЕКЦІЇ АВТОМОБІЛІВ

2.1 Проектування архітектури застосунку для обліку колекції автомобілів

Для даного застосунку було обрано архітектуру MVVM (Model-View-ViewModel) – дана архітектурна модель складається з трьох складових, таких як:

– Model – використовується для опису даних всередині моделі.

– View – візуальний інтерфейс з яким взаємодіє користувач.

– ViewModel – пов’язує модель (Model) з інтерфейсом (View).

Основні переваги використання даної моделі це:

– Інтерфейс програми може бути перероблений, не торкаючись моделі представлення та коду моделі, за умови, що представлення реалізовано повністю на XAML або C#. Таким чином, нова версія представлення повинні працювати з існуючою моделлю представлення.

– Дизайнери і розробники можуть працювати незалежно і паралельно над своїми компонентами під час розробки. Дизайнери можуть зосередитися на представленні, в той час як розробники можуть працювати над моделлю представлення модель і компоненти моделі.

Ключ до ефективного використання MVVM полягає в розумінні того, як розбити код програми на правильні класи і те, як класи взаємодіють між собою.[7, с 10]

У файлі проекту йде розподілення коду на два види файлів .xaml та .xaml.cs (рис 2.1). Файл .xaml відповідає за інтерфейс користувача, у ньому прописані користувацькі елементи, такі як: кнопки, комбіновані списки та інше. Він відображає дані які передаються через ViewModel.

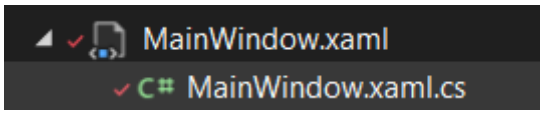


Рис. 2.1 Файлова структура головного екрану

У файлі .xaml.cs прописана логіка взаємодії з користувачем та основна модель Car(рис 2.2).

```
public class Car : INotifyPropertyChanged
{
    private string _brand;
    private string _model;
    private string _year;
    private string _color;
    private string _fuelType;
    private string _engineVolume;
    private string _vinCode;
    private string _licensePlate;

    private BitmapImage _image;
```

Рис. 2.2 Модель Car відображена у застосунку

Але взаємодія у програмі не обмежується головним вікном, адже є ще додавання автомобілів, створення колекцій, редагування, видалення та детальний перегляд інформації про автомобіль. Це все краще зобразити діаграмою класів на рис 2.3. На даному рисунку зображено взаємодію вікон між собою та взаємодію вікон з класами, такими як CategoryMenuWindow взаємодіє з класом Category, у якому описано властивості моделі даних: String Name, String Description, BitmapeImage Image, які відповідають за назву категорію(Name) , опис категорії(Description) та додане зображення до категорії(Image). Інші вікна, такі як MainWindow, AddCarWindow та EditCarWindow взаємодіють з класом Car, для додавання автомобіля, зміни його характеристик та відображення його у головного меню. Між собою вікно взаємодіють за допомогою функцій які прописані всередині кожного файла відповідного вікна, та передають зміни на файл який відповідає за інтерфейс користувача відображаючи зміни які були зроблено.

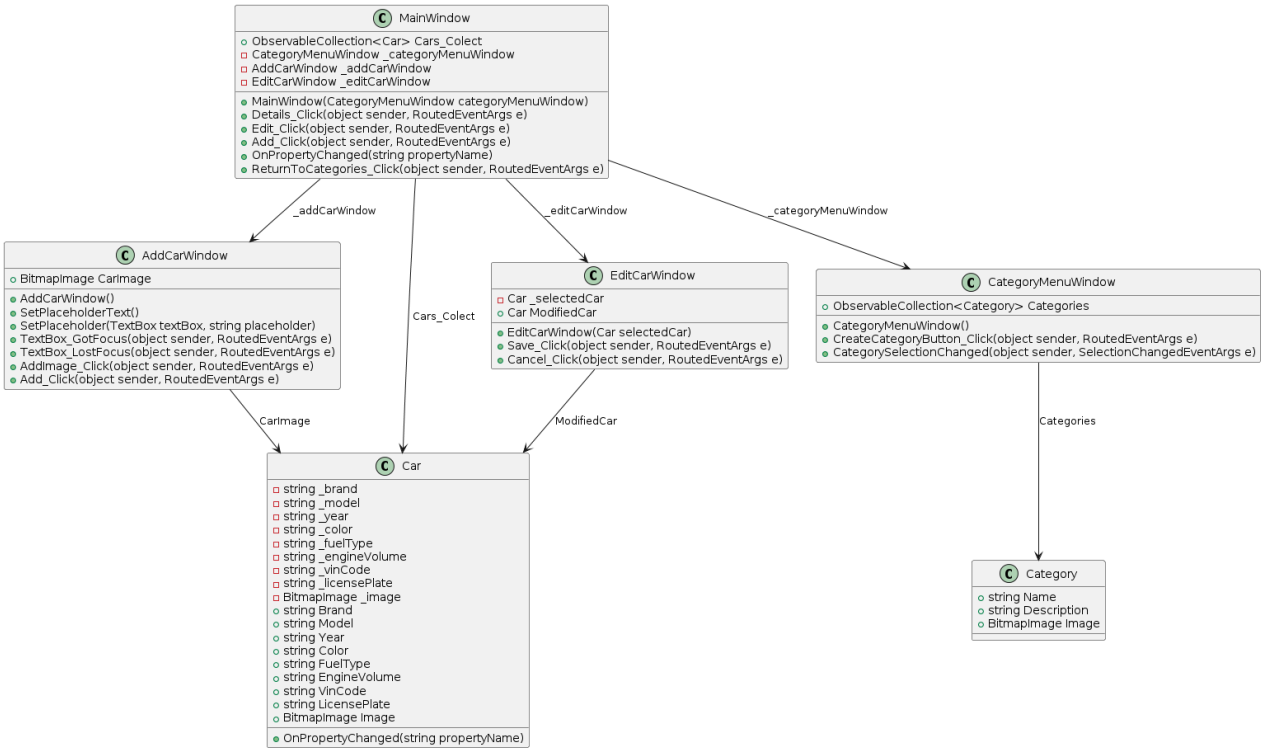


Рис. 2.3 Діаграма класів застосунку для обліку колекції автомобілів.

2.2 Моделювання вимог до застосунку для обліку колекції автомобілів

Аналізуючи аналоги було виявлено необхідні вимоги до застосунку. Всі вимоги поділяються на два типи, а саме – Функціональні, та Нефункціональні.

Функціональні вимоги визначають можливості та функції системи. Іншими словами, вони описують, що саме повинен робити програмний продукт за нормальних умов, щоб задовольнити потреби користувача. З точки зору розробника, функціональні вимоги - це можливості, які необхідно реалізувати, щоб система працювала належним чином.[8] Функціональні вимоги грають ключову функцію у розробці, адже за допомогою них надають рекомендації для розробників, дозволяють усвідомити чітке бачення кінцевого продукту.

Нефункціональні вимоги визначають якості програмної системи, які забезпечують її ефективність при одночасному впровадженні будь-яких обмежень та обмежень на проектування.[9, с 60] Нефункціональні вимоги також грають

ключову роль у розробці, за допомогою них можна визначити критерії роботи застосунку, а не його функціональної частини.

Функціональні вимоги:

1. Створення колекції;
2. Додавання автомобілів до колекції;
3. Перегляд детальної інформації;
4. Редагування інформації про автомобіль;
5. Сортування списку автомобілів.
6. Сортування списку категорій.

Нефункціональні вимоги:

1. Створення XAML інтерфейсу;
2. Відображення прикладу заповнення полей.
3. Формат зображень .jpeg, .png.

На рис 2.3, зображено діаграму використання, вона відображає варіанти використання застосунку.

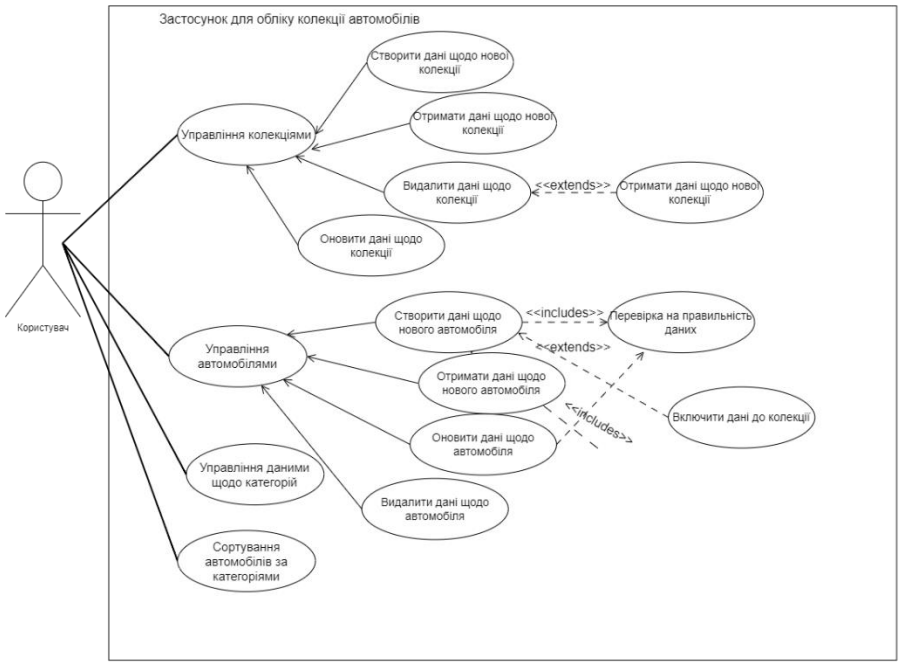


Рис. 2.4 діаграма варіантів використання застосунку для обліку колекції автомобілів

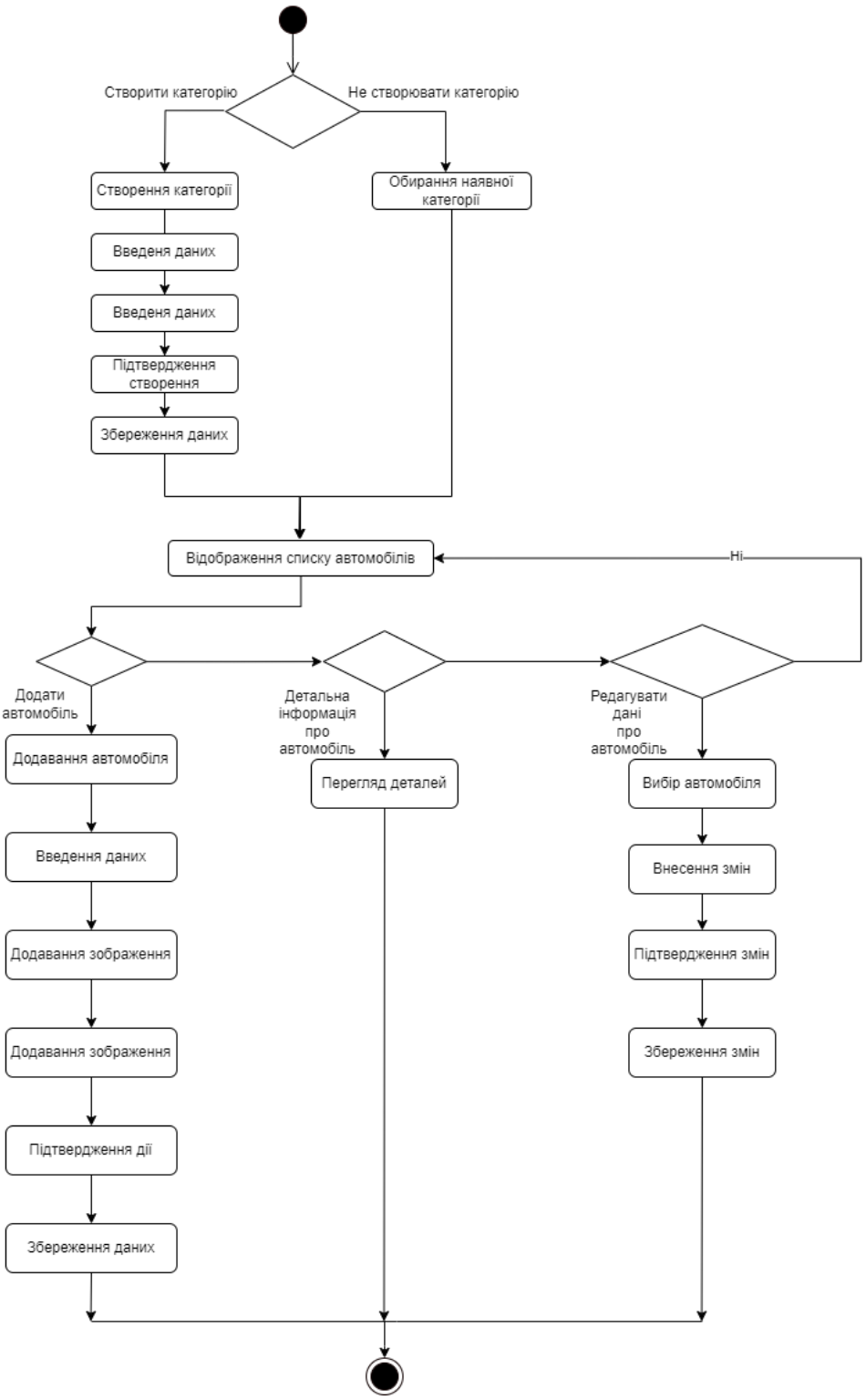


Рис. 2.5 Діаграма діяльності застосунку для обліку колекції автомобілів

На рисунку 2.4 зображено діаграма діяльності застосунку для обліку колекції автомобілів, яка відображає основний напрям дії застосунку. Для більше детальних прикладів нижче додано таблиці сценаріїв 2.1 та 2.2.

В таблиці умовні позначення це К-користувач, З-застосунок.

Таблиця 2.1

Сценарій використання - створення категорії

Створення категорії	Крок	Опис
	1	К: Натискає кнопку «створити»
	2	З: Відкриває вікно створення категорії
	3	К: Заповнює поля
	4	З: Показує вікно категорій з вже доданою категорією

Таблиця 2.2

Сценарій використання – додавання та редагування авто

Додавання та редагування авто	Крок	Опис
	1	К: Натискає додати авто
	2	З: Відкриває вікно додавання авто
	3	К: Заповнює необхідні поля
	4	З: Перевіряє заповнення, все вірно. Додає авто до колекції
	5	К: Натискає редагувати авто
	6	З: Відкриває вікно редагування авто
	7	К: Змінює поля
	8	З: Перевіряє заповнення, все вірно. Змінює характеристики авто
Примітки	4.1	В разі помилки повертає пункт 2
	8.1	В разі помилки повертає пункт 6

2.3 Проектування інтерфейсу користувача застосунку для обліку колекції автомобілів

Для проектування інтерфейсу користувача було використано застосунок Figma, адже він надає поширений базовий функціонал та велику кількість плагінів для полегшення. Основні плагіни які були використанні для розробки це:

– Design Lint – плагін який дозволяє витримати єдиний стиль усіх наявних форм у вашому проекті. Перевіряє заливки, форми, колір та сам змінює їх до одного вигляду.

– Font Preview – плагін який відображає введений текст одразу у всіх шрифтах, що полегшує вибір шрифту для застосунку.

– Iconofy – плагін має величезну колекцію іконок та різних піктограм, які можна легко додати одразу до проекту.

Для застосунку було підібрано основні кольори, а саме:

– Gray – для кнопки повернення до категорій, вікно MainWindow.

– Green – для кнопок які підтверджують дію.

– White – основний колір фону для усіх вікон.

– IndianRed – для кнопки скасування дії.

Основні інструменти:

– Плагіни які були перераховані вище.

– Текстові поля.

– Стандартні геометричні фігури.

– Базова функція «рамка», яка відіграє роль вікна застосунку.

В результаті вийшло 6 екраних форм, з яких дві основні для відображення списку автомобілів та списку категорій, а також 4 додаткові – додавання та редагування авто, створення колекції та перегляд детальної інформації про автомобіль.

Основні форми складаються з кнопок створення, та списку елементів який розташований у вигляді плиток. Колірна палітра складається з зрозумілих кольорі, адже додавання та підтвердження дії виражається зеленим кольором, а відміна дії

червоним.

Форми додавання, редагування складаються з текстових полів які необхідно заповнити, кожне поле підписано, та у формі додавання автомобіля є приклад заповнення який зникає при виділенні полі. У формі додавання, при виборі зображення воно відобразиться у зменшеному форматі, що дозволяє побачити як буде виглядати у застосунку.

На рис 2.6, зображено три категорії, одна зі стандартним зображенням та заповнення, друга та третя була заповнене так як забажає користувач, чи то текстом, чи то цифрами, та доданим зображенням. У правому верхньому куті, зелена кнопка створення категорії яка відкриває вікно зображене на рис 2.7, на якому є два поля – назва категорії та опис категорії, також є вікно передогляду зображення, та дві кнопки – ліва відповідає за додавання категорії до списку, а права надає можливість додати зображення.

Рис. 2.6 Основне вікно категорій

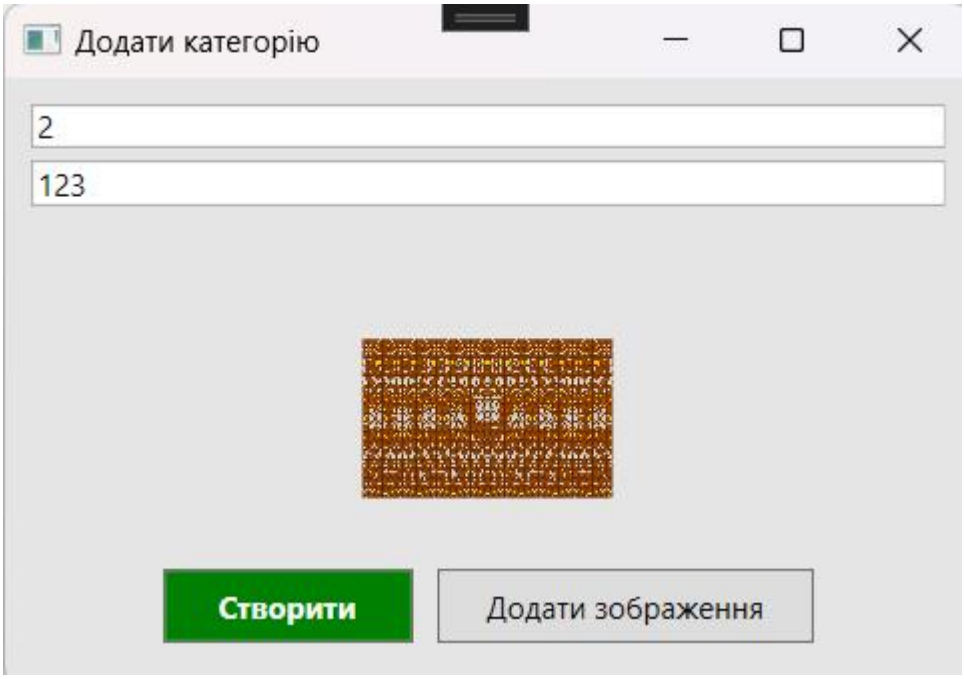


Рис. 2.7 Вікно створення категорії

На рис 2.8 зображено екран доданих автомобілів до колекції. Ми бачимо два додані авто у вигляді плиток, ліве авто було додано з даними по замовчуванню, а праве було змінено та додане фото. Тут також наявні дві кнопки, повернутись до категорій сірого кольору, та додати зеленого кольору яка трохи збільшується при наведені на неї, а також поруч із нею розташовано випадний список з способами сортування, а саме – За брендом, та за датою додавання. При натисненні на кнопку додати, відкривається вікно зображене на рис 2.9., на ньому ми бачимо поля необхідні для заповнення, випадний список з назвою «Вид палива» у якого є три пункти – Бензин, Дизель, Електро. Також є вікно для передогляду зображення яке планується для додавання.

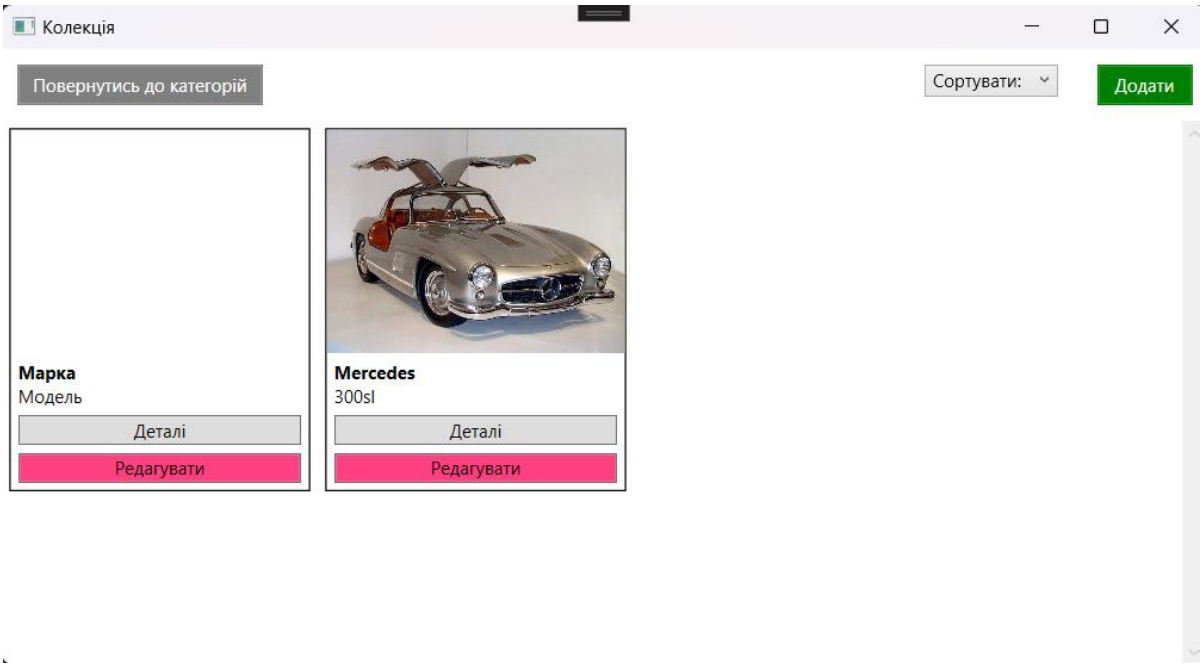


Рис. 2.8 Вікно автомобілів

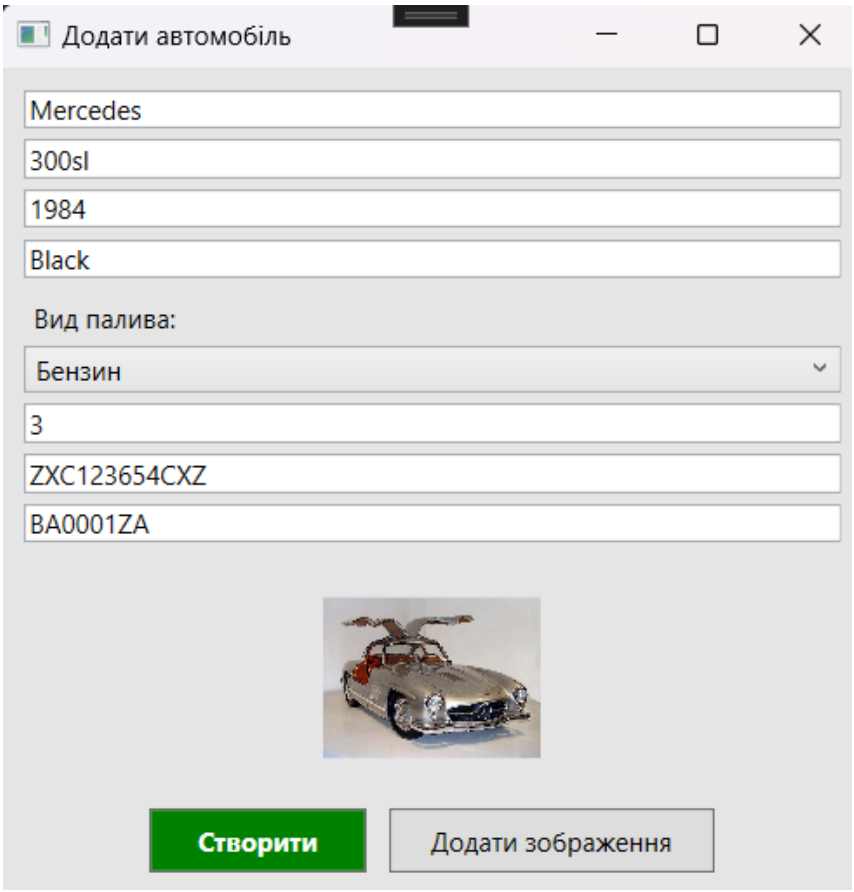


Рис. 2.9 Вікно додавання авто

Нижче зображене рис 2.10 та 2.11, на яких ми бачимо що було змінено рік автомобіля та його реєстраційний номер. На рис 2.10, ми так само як і на рис 2.9 маємо поля які можна редагувати, але тут з’явилась кнопка скасування дії, яка повертає зміни до попереднього стану.

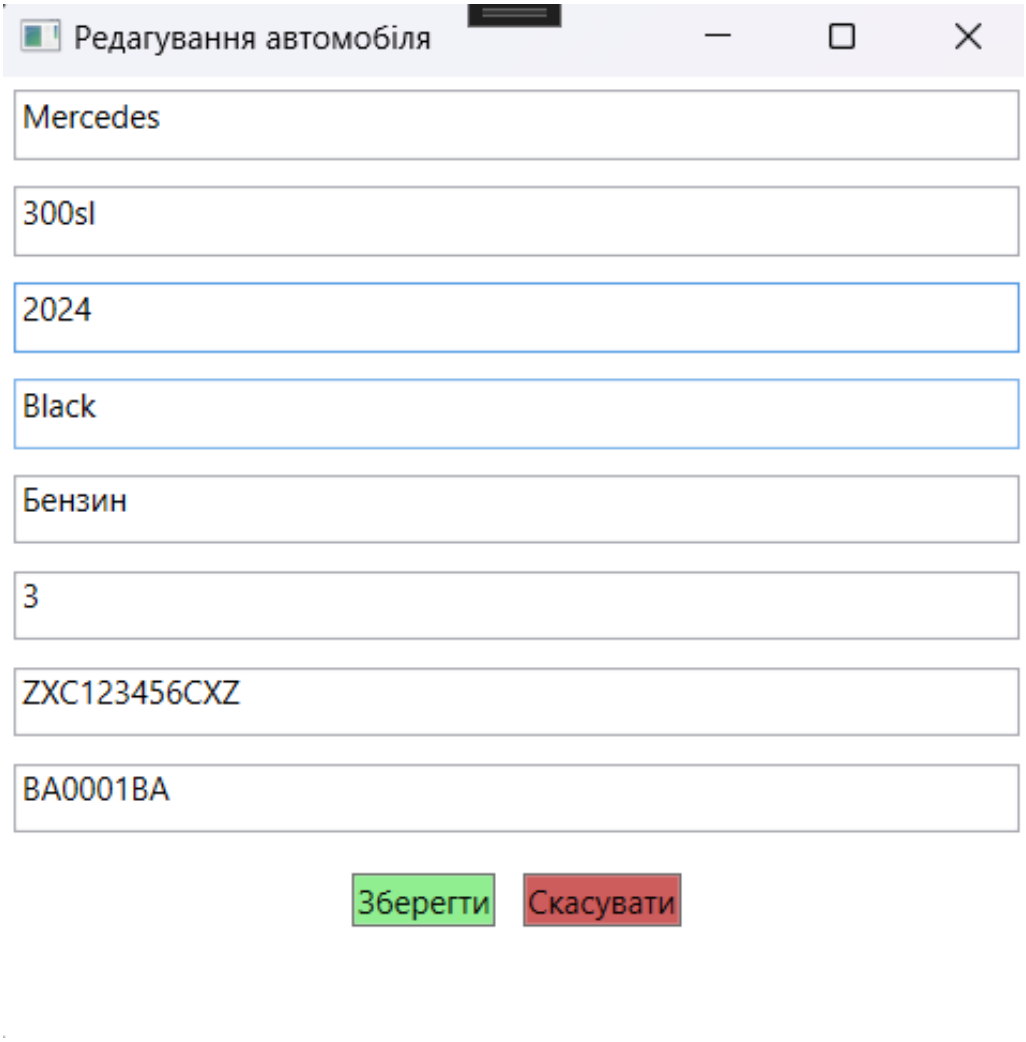


Рис. 2.10 Вікно редагування авто

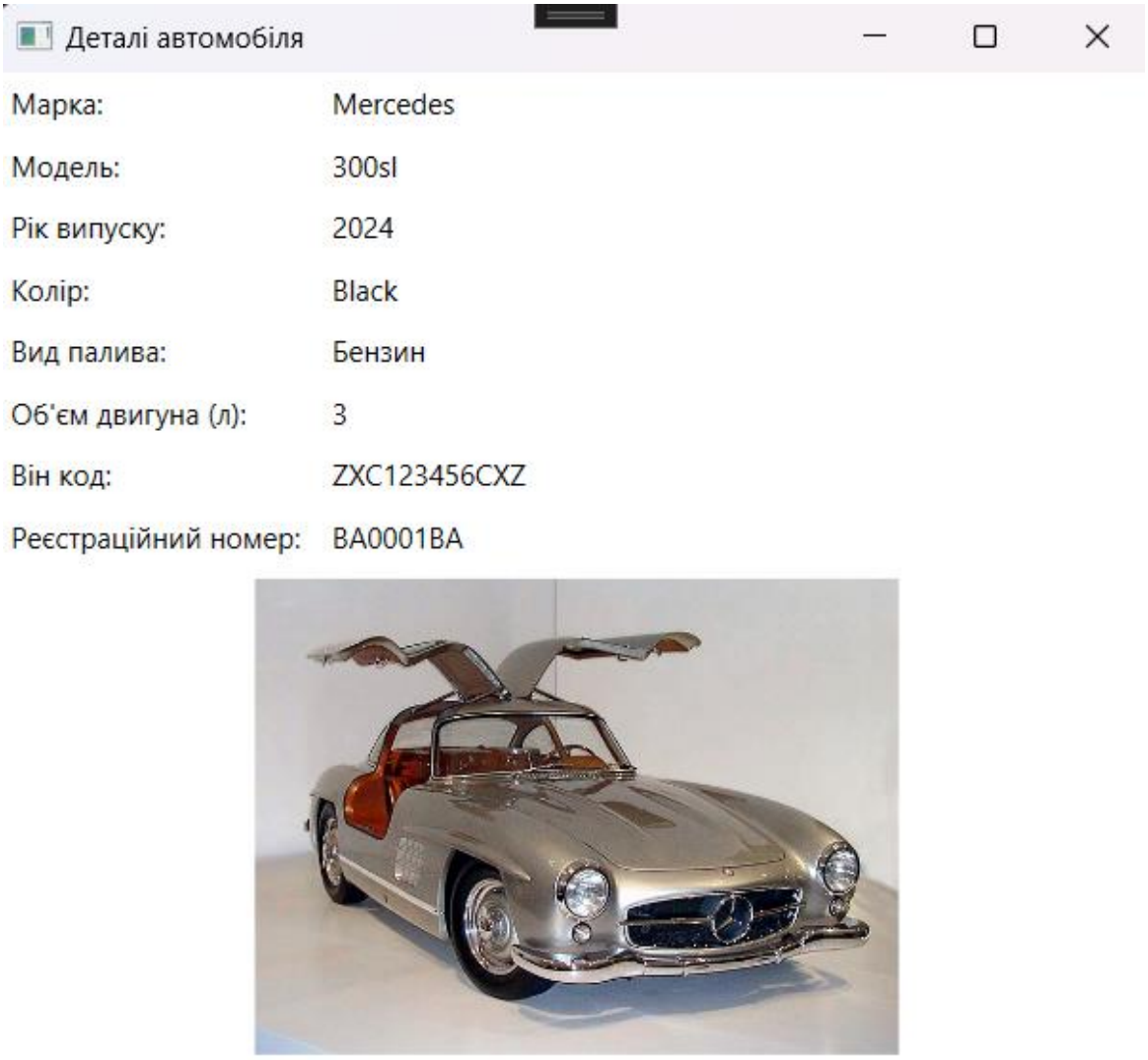


Рис. 2.11 Перегляд детальної інформації про авто

3 РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ОБЛІКУ КОЛЕКЦІЇ АВТОМОБІЛІВ

3.1 Розробка застосунку для обліку колекції автомобілів

Розробка застосунку відбувається у Visual Studio – середовище розробки, розроблене компанією Microsoft. Для створення проекту треба відкрити середовище, та обрати необхідний проект зі списку наявних варіантів зображених на рис 3.1, обираємо WPF Application.

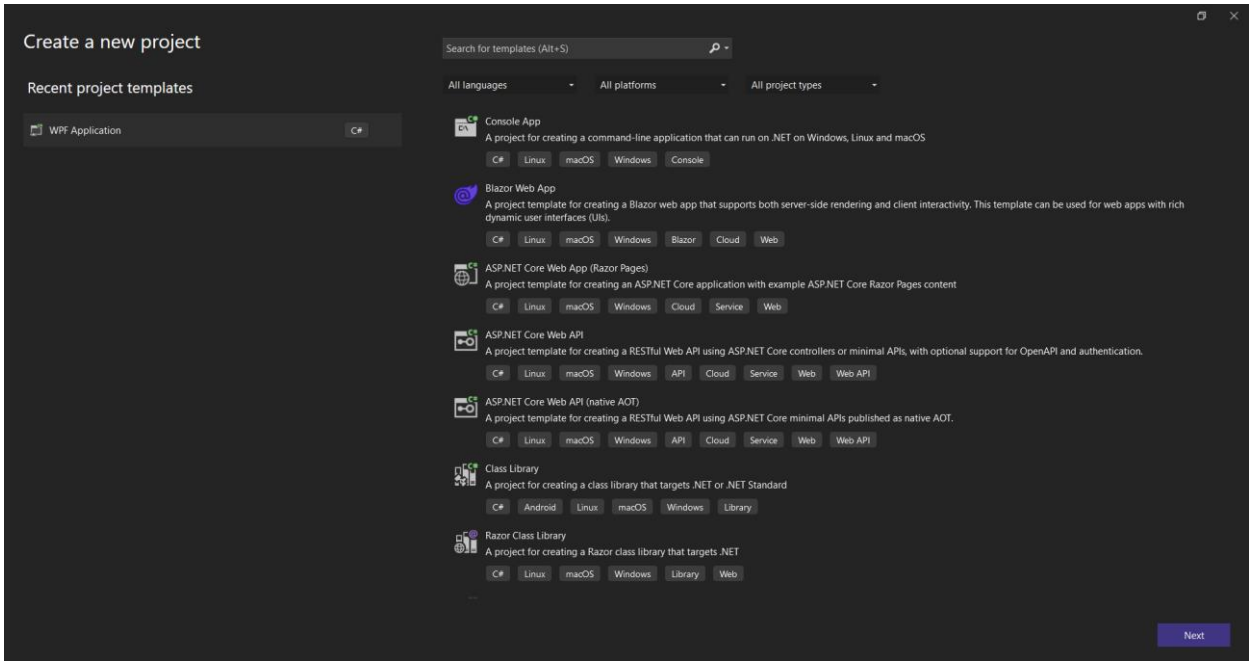


Рис. 3.1 Вибір шаблону проекту

Наступним кроком буде вибір розташування проекту, та вибір назви для застосунку, рис 3.2, та згодом обираємо Framework .NET 8.0, рис 3.3 , він має переваги над версією 6.0, а саме – покращення лаконічності коду, підтримує новітні функції та кращу ефективність використання пам’яті.

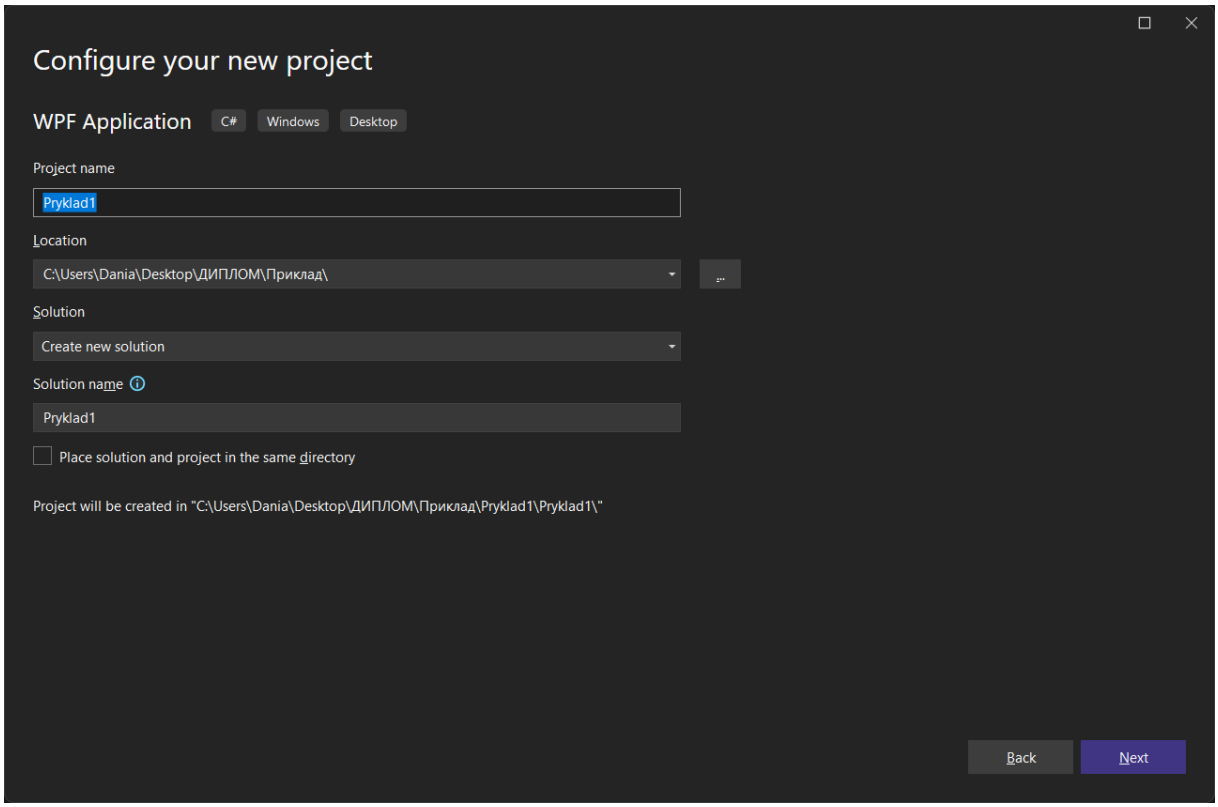


Рис. 3.2 Створення проекту

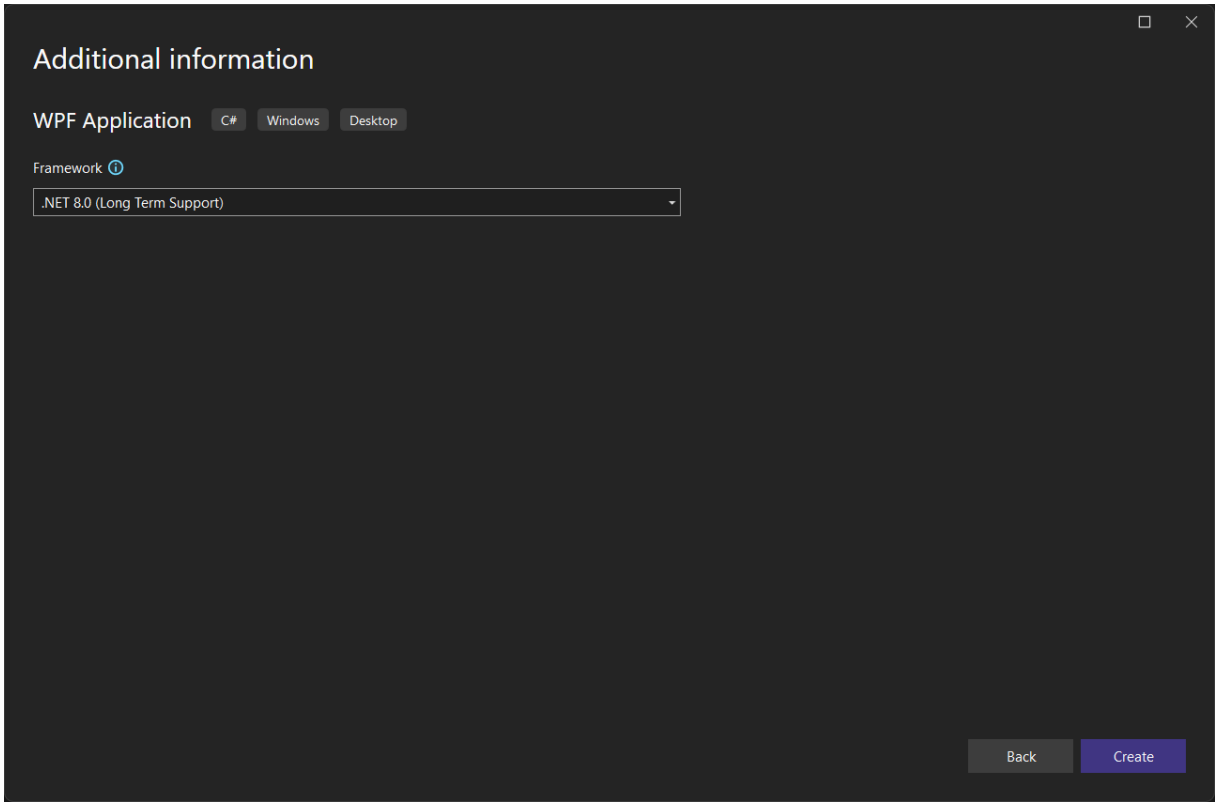


Рис. 3.3 Вибір Framework

Після підтвердження створення відкриється проект з початковою порожньою формою екрану, на яку додаються необхідні елементи інтерфейсу користувача. Для елементів інтерфейсу користувача використовується файл з розширенням .xaml, при його відкритті застосунок буде виглядати як на рис 3.4. Тут можна помітити головне вікно по центру, в якому будуть відображатись зміни які були написані у вікні нижче, або додані через ToolBox який розташовано ліворуч на рис 3.4. У колонці з правого боку екрана розташовано файли вашого застосунку, саме тут додаються нові вікна, вони ж форми.

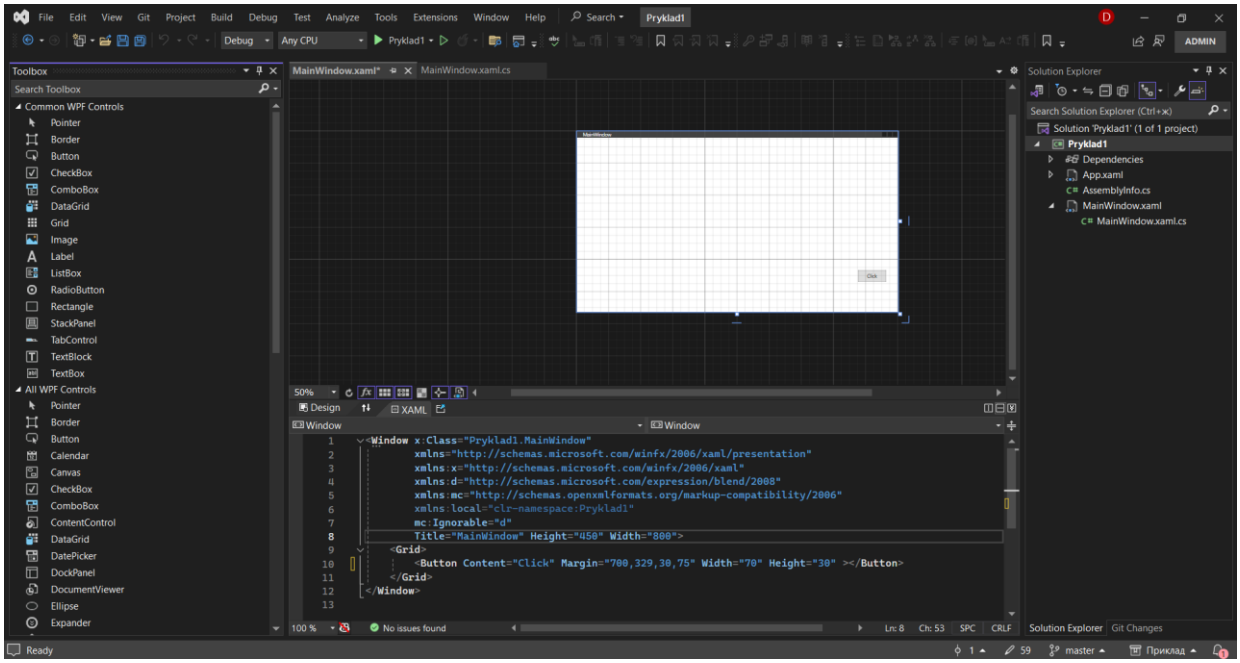


Рис. 3.4 Вигляд Visual Studio для створення інтерфейсу користувача

3.2 Реалізацію інтерфейсу користувача застосунку для обліку колекції автомобілів

Для додавання елементів на екранну форму можна використовувати ToolBox, чи власноруч прописувати кожен елемент. Так само можна вручну поставити будь-який розмір об’єкту, кут нахилу, текст всередині та розташування на екрані.

Для додавання дій для кнопок можна прописувати вручну атрибут

всередині кнопки, чи подвійне натискання на кнопку присвоює дію з назвою «Click», та відкривається файл з тою ж назвою тільки розширенням .xaml.cs, саме у цьому файлі прописується логіка застосунку, та його взаємодія з інтерфейсом користувача чи іншими вікнами.

Розробка інтерфейсу користувача виконується мовою XAML, яка використовується для написання інтерфейсу при використанні WPF. Вона дозволяє проектувати дизайн незалежно від логіки програми, та змінювати дизайн залишаючи минулу логіку. Дизайн мовою XAML схожий на розмітку, шо значно полегшує написання.

Для опису кольору елементу не потрібно створювати окремий файл, необхідно просто прописати атрибут «Background» та присвоїти йому колір. Ці всі базові атрибути зображено на рис 3.5

```
<Grid>
    <Button Content="Click" Margin="700,329,30,75" Width="70" Height="30" Click="Button_Click" Background="Aqua" ></Button>
</Grid>
```

Рис. 3.5 Код кнопки

3.3 Написання логіки застосунку для обліку колекції автомобілів

Для опису логіки використовується мова C#. Основна логіка застосунку прописана у файлі MainWindow.xaml.cs.

У цьому файлі основними компонентами є:

– Клас MainWindow.
– Клас CategoryMenuWindow.
– Клас CarDetailsWindow.
– Клас EditCarWindow.
– Клас AddCarWindow.
– Модель Car.

Для кожного класу є власна подія, яку викликає натискання на кнопку.

Натискання на кнопку додавання автомобіля викликає нове вікно AddCarWindow, в якому користувач заповнює необхідні поля, та якщо заповнене

вірно то повертає назад результат true та додає авто до колекції рис 3.6.

```
private void Add_Click(object sender, RoutedEventArgs e)
{
    AddCarWindow addCarWindow = new AddCarWindow();
    if (addCarWindow.ShowDialog() == true)
    {
        Cars_Colect.Add(new Car
        {
            Brand = addCarWindow.txtBrand.Text,
            Model = addCarWindow.txtModel.Text,
            Year = addCarWindow.txtYear.Text,
            Color = addCarWindow.txtColor.Text,
            FuelType = addCarWindow.cmbFuelType.Text,
            EngineVolume = addCarWindow.txtEngineVolume.Text,
            VinCode = addCarWindow.txtVinCode.Text,
            LicensePlate = addCarWindow.txtLicensePlate.Text,
            Image = addCarWindow.CarImage
        });
    }
}
```

Рис. 3.6 Клас AddCarWindow

Натискання на кнопку редагування викликає нове вікно EditCarWindow передаючи вибраний автомобіль та дані про нього, в якому користувач змінює поля, та після закриття змінює дані про авто у колекції рис 3.7.

```
private void Edit_Click(object sender, RoutedEventArgs e)
{
    Car selectedCar = (sender as Button).Tag as Car;

    EditCarWindow editCarWindow = new EditCarWindow(selectedCar);

    editCarWindow.ShowDialog();

    if (editCarWindow.DialogResult == true)
    {
        int index = Cars_Colect.IndexOf(selectedCar);
        if (index != -1)
        {
            Cars_Colect[index] = editCarWindow.ModifiedCar;
        }
    }
}
```

Рис. 3.7 Редагування деталей про автомобіль

Клас MainWindow відповідає за приймання параметру CategoryMenuWindow, який являється існуючим вікном категорій, що дозволяє повернутись назад без створення нового вікна. Конструктор ініціалізує дані та колекцію, рис 3.8. _categoryMenuWindow – зберігає посилання на вікно категорій.

```
private CategoryMenuWindow _categoryMenuWindow;

1 reference
public MainWindow(CategoryMenuWindow categoryMenuWindow)
{
    InitializeComponent();
    DataContext = this;
    Cars_Colect = new ObservableCollection<Car>();
    _categoryMenuWindow = categoryMenuWindow;
```

Рис. 3.8 Клас MainWindow

На рис 3.9 зображено інтерфейс INotifyPropertyChanged, який відповідає за зміни властивостей класу Car.

```
public event PropertyChangedEventHandler PropertyChanged;

0 references
protected virtual void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Рис. 3.9 Інтерфейс INotifyPropertyChanged

Також у даному файлі є клас Car з наступними властивостями, які мають метод set (рис 3.10 зображено два, інші виглядають аналогічно), який відповідає за оновлення інтерфейсу за допомогою OnPropetryChanged, рис 3.11

Властивості класу Car рис 3.12:

– Brand.
– Model.
– Year.

– Color.
– FuelType.
– EngineVolume.
– VinCode.
– LicensePlate.
– Image.

```
public string Brand
{
    get { return _brand; }
    set
    {
        _brand = value;
        OnPropertyChanged("Brand");
    }
}

2 references
public string Model
{
    get { return _model; }
    set
    {
        _model = value;
        OnPropertyChanged("Model");
    }
}
```

Рис. 3.10 Властивості класу Car з методом set

```
public event PropertyChangedEventHandler PropertyChanged;

9 references
protected virtual void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Рис. 3.11 Оновлення за допомою OnPropertyChanged

```
public class Car : INotifyPropertyChanged
{
    private string _brand;
    private string _model;
    private string _year;
    private string _color;
    private string _fuelType;
    private string _engineVolume;
    private string _vinCode;
    private string _licensePlate;

    private BitmapImage _image;
```

Рис. 3.12 Властивості класу Car

4 ТЕСТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОБЛІКУ КОЛЕКЦІЇ АВТОМОБІЛІВ

4.1 Види тестування застосунку

Тестування застосунку є ключовим етапом перед випуском його у світ, чи перед публікацією оновленого застосунку для перевірки працездатності. Під час розробки програмного забезпечення, може бути важко виявити помилки або визначити, як численні компоненти програми працюватимуть разом під час її розгортання. Тестування програмного забезпечення може перевірити, чи всі компоненти бездоганно працюють разом.[10]

Основні переваги тестування програмного забезпечення:

– Виявлення помилок – ранішнє виявлення помилок, забезпечує полегшення виправлення помилок, та швидкість розробки.

– Краща якість програмного забезпечення – після проведення тестування, та вирішення проблем продукт виявиться більш якісний для кінцевого користувача, адже не матиме базових помилок та основний функціонал швидше за все працюватиме справно.

– Чим раніше виявляться помилки тим дешевше для компанії їх виправити, адже виправлення помилок на етапі розробки значно легше ніж у вже випущеного продукту.

Всього існує дуже багато видів тестування, такі як:

– Димне тестування.

– Функціональне тестування.

– Нефункціональне тестування.

– Модульне тестування.

– Стрес-тестування.

– Мануальне тестування.

– Та інші.

Для мого продукту було обрано димне тестування, та сценарне тестування. Димне тестування було обрано для перевірки основного функціоналу додатку, та виконувалось вручну. Тестування сценаріями, в свою чергу, використовувалось для перевірки більш детальних сценаріїв користувача. Воно необхідне для більш розгалуженої перевірки, наприклад користувач вирішить записати в поле потужності двигуни, якесь слово, як відреагує система та інші варіанти, які будуть описані нижче.

Димне тестування – димове тестування, smoke test – це метод тестування програмного забезпечення, який використовується для оцінки стабільності програмного забезпечення та його готовності до наступних етапів тестування. Цей метод оцінює функціональність найважливіших функцій програми, не заглиблюючись у складні деталі. [11] Димне тестування є початковою перевіркою розроблюваного програмного забезпечення, якщо застосунок не проходить димне тестування то немає сенсу проводити більше детальне тестування, адже застосунок не справляється з початковою задачею. Існує декілька видів такого тестування, а саме – ручне, автоматичне та гібридне. Для перевірки застосунку для обліку колекції автомобілів було обрано ручне димне тестування. Основні поради для димного тестування:

1. Визначення основних функцій – димне тестування дозволяє перевірити тільки критичні функції.
2. Залучення більшої кількості зацікавлених людей.

Сценарне тестування – Scenario based testing – це техніка тестування програмного забезпечення, яка використовує сценарії, щоб допомогти тестувальнику впоратися зі складною проблемою або тестовою системою. Ідеальний сценарний тест – це достовірна, складна, переконлива або мотивуюча історія, результат якої легко оцінити.[12] Особливості сценарного тестування:

– Сценарій має бути заснований на реальних ситуаціях використання програми.
– Перевірка застосунку з точки зору користувача, а саме виявлення проблема з інтерфейсом та простотою використання програми.

Основними етапами сценарного тестування є – визначення основних сценаріїв, виконання тестів та аналіз результату.

4.2 Димне тестування застосунку для обліку колекції автомобілів

Таблиця 4.1

Сценарій –створення колекції

Опис	Крок	Дія	Очікуваний результат	Реальний результат	Статус
Створення колекції	1	Відкриття застосунку	У вікні категорій з’явиться нова категорія	У вікні категорій з’явилась нова категорія	Виконано
	2	Натиснути на кнопку «Створити колекцію»			
	3	Заповнити поля у отриманому вікні			
	4	Натиснути кнопку підтвердження			

У таблиця 4.1 описано сценарій створення нової колекції, як результат ми бачимо що колекція створилась отже статус даного тесту «Виконано».

Таблиця 4.2

Сценарій – додавання авто до колекції

Опис	Крок	Дія	Очікуваний результат	Реальний результат	Статус
Додавання авто до колекції	1	Відкриття застосунку	У вікні автомобілів має з’явитись плитка з доданим авто	У вікні автомобілів має з’явитись плитка з доданим авто	Виконано
	2	Відкрити колекцію			
	3	Натиснути кнопку додати			
	4	Заповнити необхідні поля			
	5	Додати фото			
	6	Натиснути кнопку підтвердження			

Результат виконання таблиці 4.2 зображено на рис 4.1

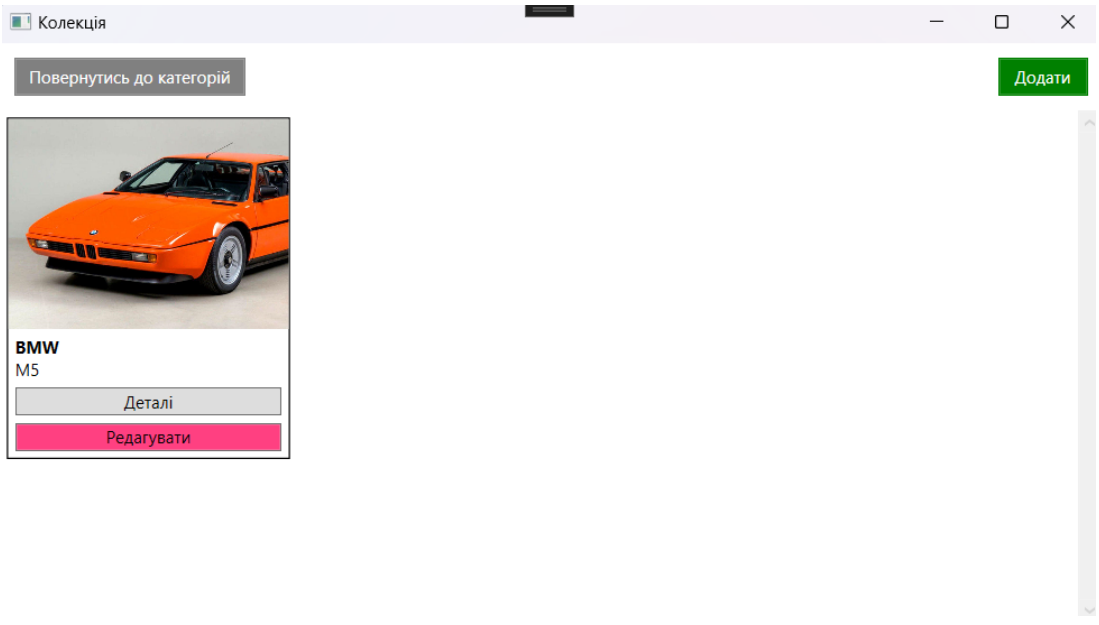


Рис. 4.1 Результат виконання тесту

У таблиці 4.3 описано тестування редагування інформації про авто та перегляд детальної інформації.

В результаті Димного тестування, було перевірено усі основні функції, а саме створення колекції, додавання авто до колекції, редагування та перегляд інформації про авто. Усі основні функції працюють так як і очікувалось, отже можна переходити до наступного тестування, а саме сценарного тестування.

Таблиця 4.3

Сценарій – редагування інформації та перегляд детальної інформації

Опис	Крок	Дія	Очікуваний результат	Реальний результат	Статус
Редагування інформації та перегляд детальної інформації	1	Натиснути на кнопку Редагувати під необхідним авто	З’явиться вікно редагування авто, після підтвердження дії зміни збережуться та при відкриті детальної інформації буде оновлена інформація	З’явиться вікно редагування авто, після підтвердження дії зміни збережуться та при відкриті детальної інформації буде оновлена інформація	Виконано
	2	Змінити необхідні поля у новому вікні			
	3	Підтвердити дію			
	4	Натиснути на кнопку Деталі під необхідним авто			
	5	Переглянути нове вікно			

4.3 Сценарне тестування застосунку для обліку колекції автомобілів

На таблиці 4.4 наведено більш детальні сценарії тестування застосунку, такі як додавання автомобіля з фото чи без, створення колекції з фото чи без, редагування інформації про авто, та перевірка коректності заповнення. Проаналізувавши дану таблицю було виявлено проблеми та відповідно їх вирішено. Сценарне тестування виявило проблеми які неможливо було виявити при димному тестуванні, а саме перевірка коректності заповнення.

Таблиця 4.4

Сценарне тестування застосунку

Сценарій	Попередні умови	Кроки	Очікуваний результат	Реальний результат
Додавання нового автомобіля з фото	Немає	1. Перейти до списку автомобілів. 2. Натиснути кнопку додати. 3. Ввести марку, модель, рік випуску, номерний знак. 4. Додати фото автомобіля. 5. Натиснути кнопку "Зберегти".	Новий автомобіль успішно додано до колекції з фото.	Новий автомобіль успішно додано до колекції з фото. Рис 4.2
Додавання нового автомобіля без фото	Немає	1. Перейти до списку автомобілів. 2. Натиснути кнопку додати. 3. Ввести марку, модель, рік випуску, номерний знак. 4. Натиснути кнопку "Зберегти".	Новий автомобіль успішно додано до колекції без фото.	Новий автомобіль успішно додано до колекції без фото. Рис 4.2
Створення колекції з фото	Немає	1. Відкрити вікна колекцій. 2. Натиснути кнопку «Створити колекцію» 3. Ввести назву та опис колекції. 4. Додати фото для колекції. 5. Натиснути кнопку "Зберегти".	Колекцію успішно створено з фото.	Колекцію успішно створено з фото. Рис 4.3

Продовження таблиці 4.4

Сценарне тестування Застосунку

Створення колекції без фото	Немає	1. Відкрити вікна колекцій. 2. Натиснути кнопку «Створити колекцію» 3. Ввести назву та опис колекції. 4. Натиснути кнопку "Зберегти".	Колекцію успішно створено без фото.	Колекцію успішно створено без фото. Рис 4.3
Редагування інформації про автомобіль	Автомобіль додано до колекції	1. Відкрити колекцію. 2. Натиснути кнопку «Редагувати» під необхідним авто 3. Змінити необхідну інформацію 4. Натиснути кнопку "Зберегти".	Інформацію про автомобіль успішно відредаговано.	Інформацію про автомобіль успішно відредаговано.
Перегляд детальної інформації про автомобіль	Автомобіль додано до колекції	1. Відкрити колекцію. 2. Натиснути кнопку «Деталі» під необхідним авто	Відображається детальна інформація про автомобіль.	Відображається детальна інформація про автомобіль. Рис 4.4
Перевірка коректності заповнення полів	Немає	1. Перейти до списку автомобілів. 2. Натиснути кнопку додати. 3. Ввести некоректні дані у поле «Об’єм двигуна» 4. Натиснути кнопку "Зберегти".	Застосунок відображає повідомлення про помилку та підказує, яке поле заповнене неправильно.	Застосунок відображає повідомлення про помилку та підказує, яке поле заповнене неправильно. Рис 4.5

Нижче наведено рисунки 4.2-4.5, як результати проведення сценарного тестування. На рис 4.2, відображено два доданих авто, ліворуч було додано авто без додавання фото, а праворуч було додане власне фото. Як можна помітити обидва авто відображається однаково, помилок не виникає, єдина різниця то відсутність будь-якого зображення.

На рис 4.3, додано три категорії, категорія без змін, категорія з фото та категорія зі зміненою назвою та описом, але без власного фото. Застосунок сам додає стандартне фото, яке додано по-замовчуванню.

На рис 4.4, відображається детальна інформація про авто, та трошки більше зображення, для того що б було легше його розглянути.

На риск 4.5, можна помітити що підсвічує червоним контуром неправильне заповнене поле, та при спробі зберегти з таким заповненням з’являється вікно яке вказує що помилково заповнене поле «Об’єм двигуна», та при натисканні на кнопку «ОК», повертає форму заповнення автомобіля в якій треба виправити виділене поле.

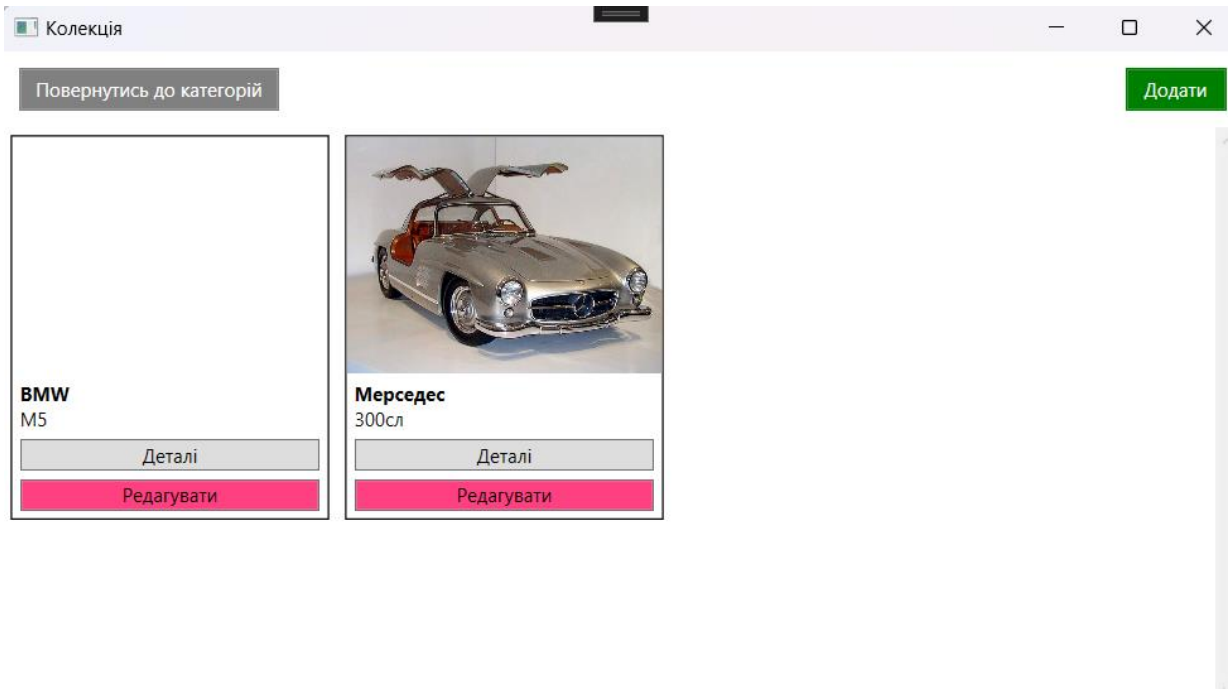


Рис. 4.2 Додавання авто з фото та без

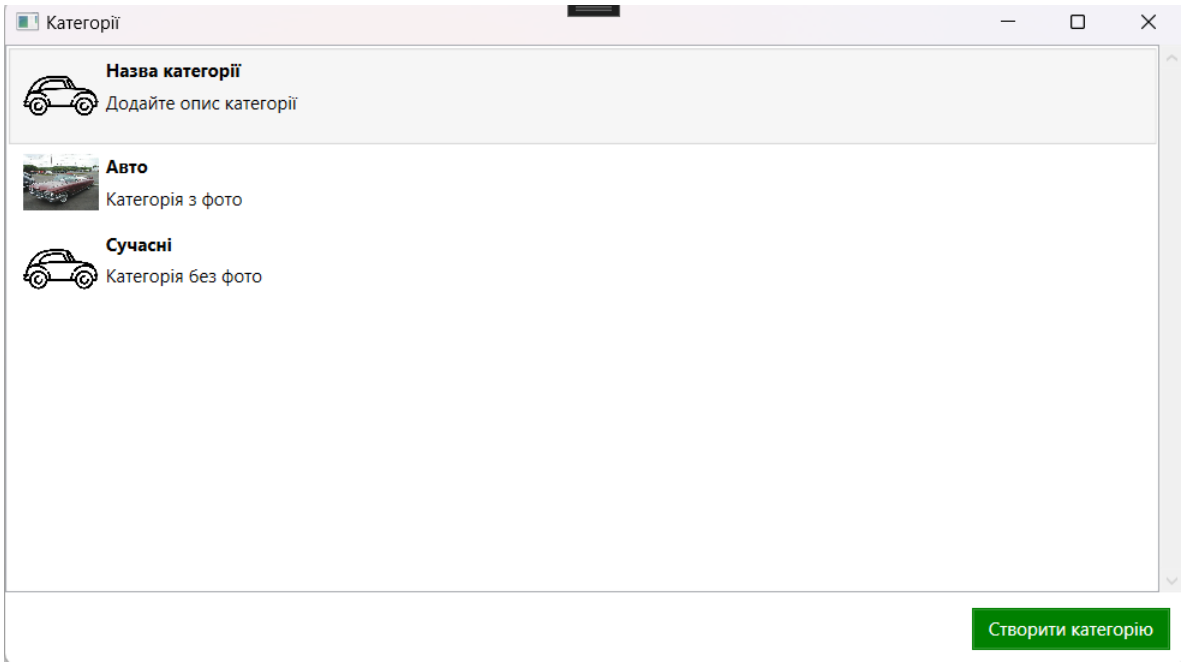


Рис. 4.3 Додавання категорій з фото та без

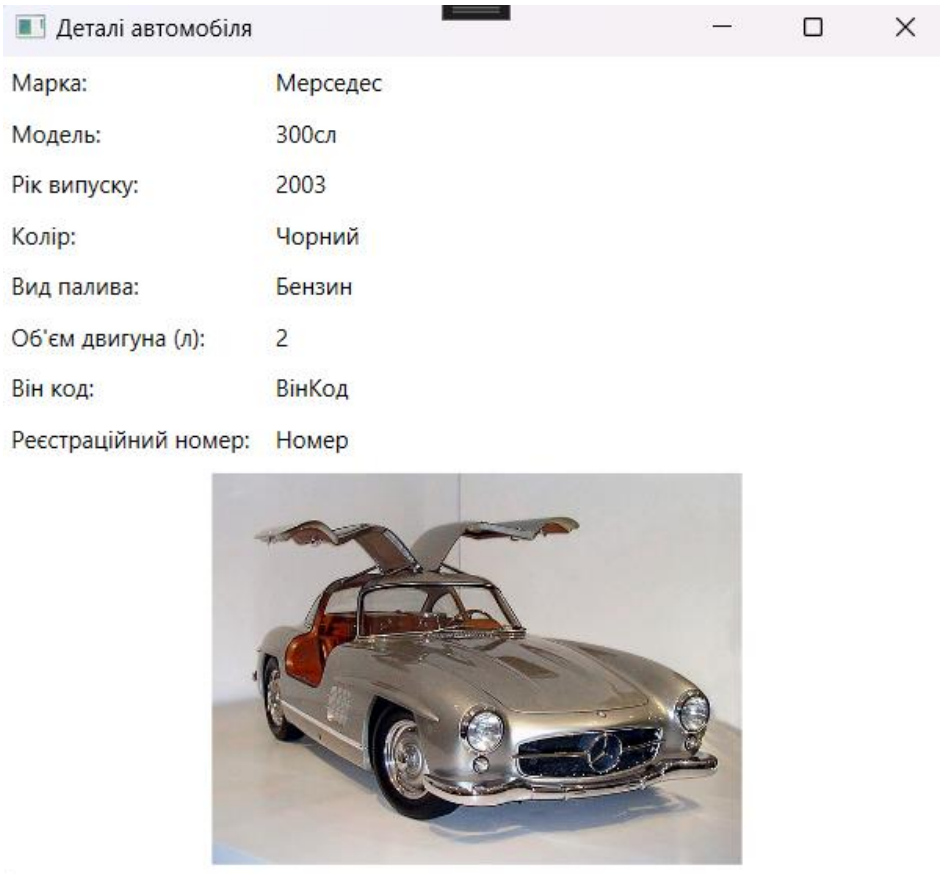


Рис. 4.4 Перегляд детальної інформації про авто

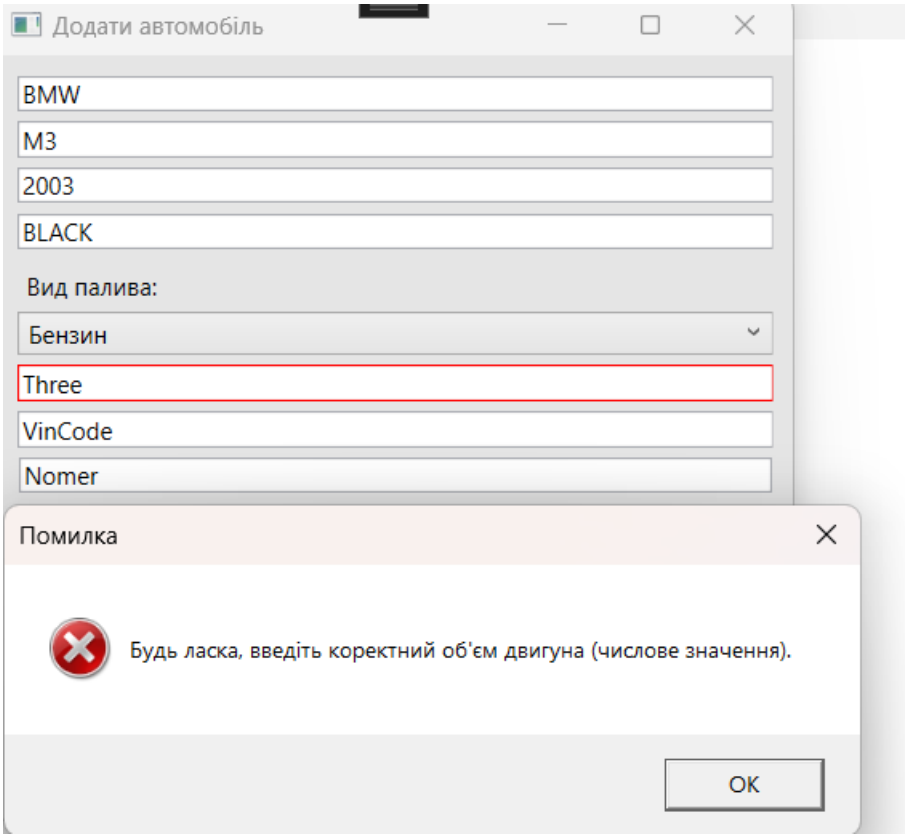


Рис. 4.5 Неправильно заповнене поле «Об’єм двигуна»

ВИСНОВКИ

Проаналізовано існуючі засоби для обліку колекції автомобілів, а саме:

– iCollect Everything.

– Collector Notepad.

Було складено функціональні та нефункціональні вимоги, які описані вище.

Було обрано інструменти розробки, такі як інтегроване середовище розробки Visual Studio, мову програмування C#, інструмент для розробки прототипу застосунок Figma, платформа для керуванням версіями коду GitHub, мова розмітки XAML, WPF для створення настільних застосунків, вбудована база даних SQLite.

На основі визначених вимог було спроектовано та розроблено застосунок, беручи до уваги переваги обраних інструментів розробки. У ході проектування було розроблено діаграми класів, варіантів використання та діяльності, для полегшення розуміння ключових аспектів розробки.

Проведено тестування застосунку за допомогою Димного та Сценарних методів. В результаті тестування було визначено та вирішено проблеми застосунку, а також зведено таблицю тестування. Тестування застосунку відбувалось у інтегрованому середовищі з двох різних ПК.

Дипломна робота пройшла апробацію на двох секціях однієї конференції:

1. Поперешняк Д.І., Гаманюк І.М. Розробка програмного забезпечення щодо застосунку для обліку колекції автомобілів з використанням UML діаграми прецедентів. Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 18 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій Збірник тез К: ДУІКТ 2024 С.158-159.

2. Поперешняк Д.І., Гаманюк І.М. Використання «GitHub» для розробки програмного забезпечення щодо застосунку для обліку колекції автомобілів. Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 18 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій Збірник тез К: ДУІКТ 2024 С.388-389.

ПЕРЕЛІК ПОСИЛАНЬ

1. C# Language Specification | Guide books. *Guide books*. URL: https://dl.acm.org/doi/abs/10.5555/861332.

2. Visual Studio: IDE and Code Editor for Software Developers and Teams. *Visual Studio*. URL: https://visualstudio.microsoft.com/.

3. What is WPF? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/what-is-wpf/.

4. SQLite (Developer's Library): | Guide books | ACM Digital Library. *Guide books*. URL: https://dl.acm.org/doi/book/10.5555/1201593.

5. Staiano F. Designing and prototyping interfaces with Figma : learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop. 2nd ed. Birmingham : Packt Publishing, 2023. 464 p.

6. About GitHub and Git - GitHub Docs. *GitHub Docs*. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

7. Stonis M. Enterprice Application Patterns using .NET MAUI. Redmond, Washington : Microsoft Developer Division, .NET, and Visual Studio product teams, 2022. 108 p.

8. Functional Requirements in Software Development: Types and B. *AltexSoft*. URL: https://www.altexsoft.com/blog/functional-requirements/.

9. Rahy S., M. Bass J. IET Software. The Institution of Engineering and Technology., 2022.

10. What is Software Testing? Definition and Types. *GitHub Resources*. URL: https://resources.github.com/software-development/what-is-software-testing/.

11. Sabirov R. A complete guide to smoke testing. *Qase Blog*. URL: https://qase.io/blog/smoke-testing/.

12. Scenario Testing - Software Testing - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/software-testing-scenario-testing/.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ОБЛІКУ КОЛЕКЦІЇ АВТОМОБІЛІВ МОВОЮ C#

Виконав студент 4 курсу
групи ПД-43
Поперешняк Даниїл Ігорович
Керівник роботи
старший викладач кафедри ІПЗ Гаманюк Ігор Михайлович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спростити процес обліку колекції автомобілів шляхом впровадження застосунку для обліку колекції автомобілів.

Об’єкт дослідження – процес обліку колекції автомобілів.

Предмет дослідження – застосунок для обліку колекції автомобілів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати та визначити недоліки існуючих засобів для обліку колекції;
2. Скласти вимоги до розроблюваного застосунку для обліку колекції автомобілів.
3. Вибрати інструменти розробки застосунку для обліку колекції автомобілів.
4. Спроектувати застосунок для обліку колекції автомобілів.
5. Розробити застосунок для обліку колекції автомобілів.
6. Провести тестування розробленого застосунку для обліку колекції автомобілів.

3

АНАЛІЗ АНАЛОГІВ

Порівняльна таблиця			
Характеристики	Collector Notepad	iCollect Everything	My App
Створення власних колекцій	-	-	+
Детальна інформація	-	+	+
Сортування автомобілів	+	-	+
Редагування вже доданого автомобіля	-	+	+
Додавання фото	-	+	+
Сортування колекцій	+	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Створення колекції;
2. Додавання автомобілів до колекції;
3. Перегляд детальної інформації;
4. Редагування інформації про автомобіль;
5. Сортування списку автомобілів.
6. Сортування списку категорій.

Нефункціональні вимоги:

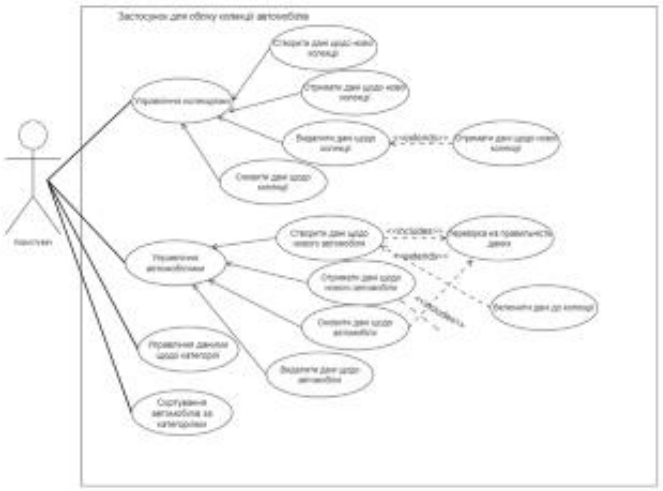
1. Створення XAML інтерфейсу;
2. Відображення прикладу заповнення полей у вікні для додавання автомобілів.
3. Формат зображень .jpeg, .png.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

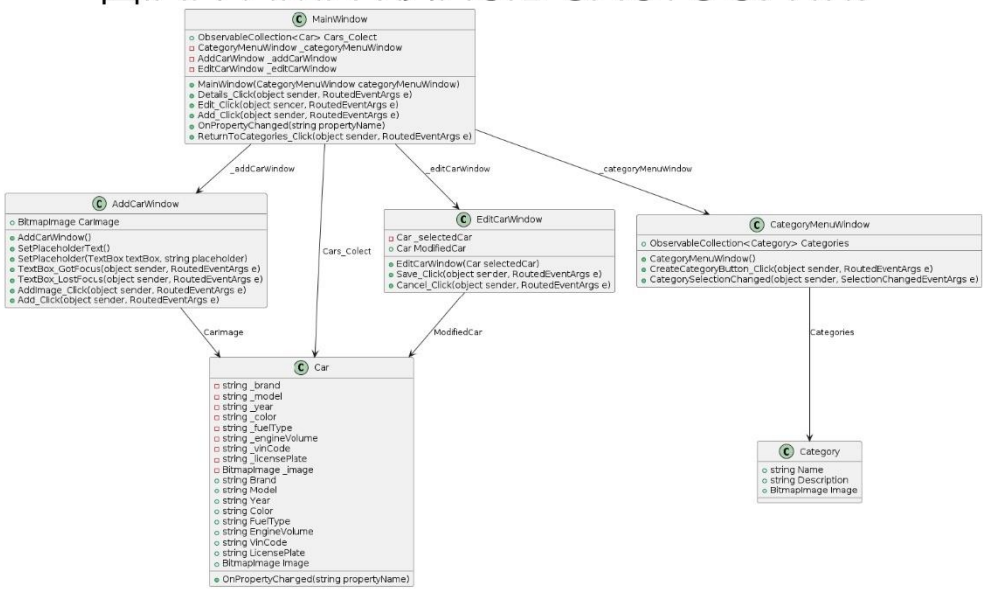


7

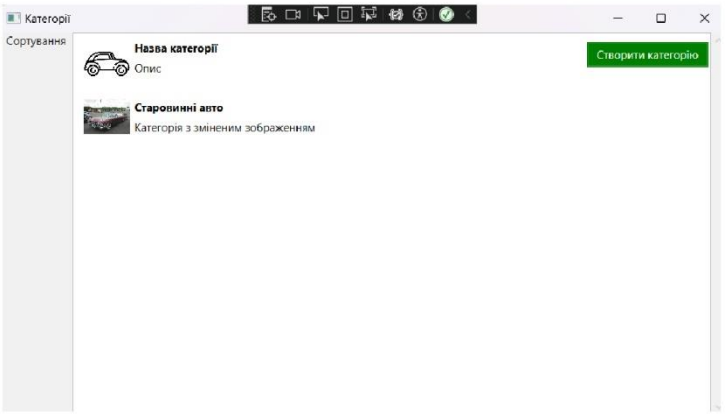
ДІАГРАМА ДІЯЛЬНОСТІ

8

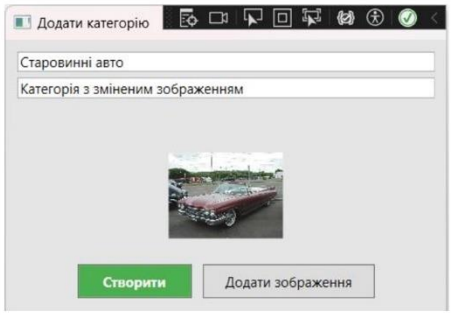
ДІАГРАМА КЛАСІВ ЗАСТОСУНКУ



ЕКРАННІ ФОРМИ

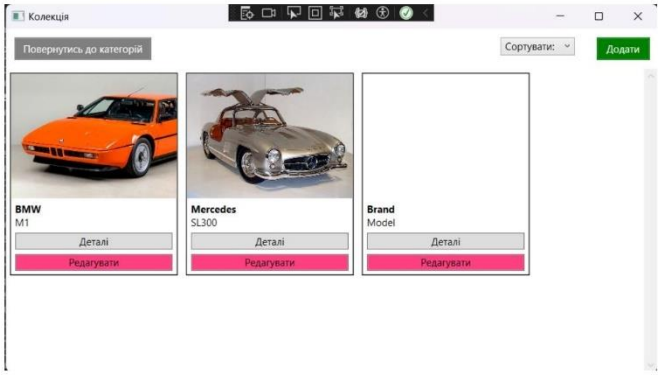


Меню категорій

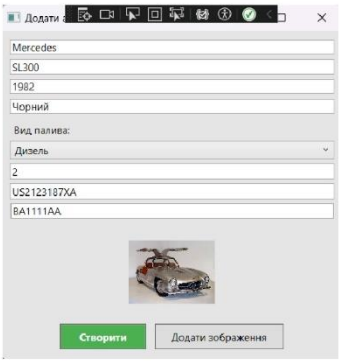


Додавання категорій

ЕКРАННІ ФОРМИ



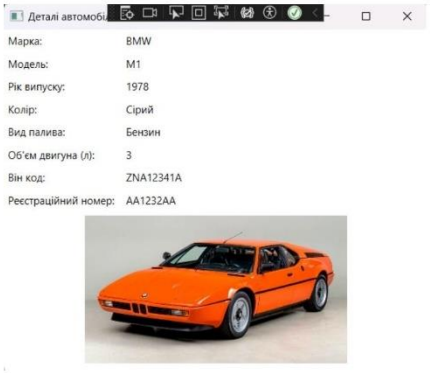
Список автомобілів



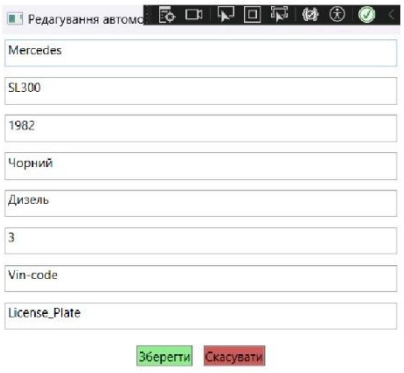
Додавання автомобілю

11

ЕКРАННІ ФОРМИ



Деталі про автомобіль



Зміни характеристик вже створеного автомобілю

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Поперешняк Д.І. Розробка програмного забезпечення щодо застосунку для обліку колекції автомобілів з використанням UML діаграми прецедентів. Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 158-159.
2. Поперешняк Д.І. Використання «GitHub» для розробки програмного забезпечення щодо застосунку для обліку колекції автомобілів. Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 388-389.

13

ВИСНОВКИ

1. Проаналізовано та порівняно існуючі рішення обліку колекції автомобілів.
2. Складено вимоги до застосунку для обліку колекції автомобілів.
3. Обрано інструменти розробки застосунку для обліку колекції автомобілів.
4. Спроектовано та розроблено застосунок для обліку колекції автомобілів, використовуючи мову C# та мову розмітки XAML.
5. Проведено тестування розробленого застосунку для обліку колекції автомобілів.

14

ДОДАТОК ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Код логіки вікна списку автомоболів

```
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media.Imaging;

namespace Cars_Colect
{
  public partial class MainWindow : Window, INotifyPropertyChanged
  {
    public ObservableCollection<Car> Cars_Colect { get; set; }

    private CategoryMenuWindow _categoryMenuWindow;

    public MainWindow(CategoryMenuWindow categoryMenuWindow)
    {
      InitializeComponent();
      DataContext = this;
      Cars_Colect = new ObservableCollection<Car>();
      _categoryMenuWindow = categoryMenuWindow;
    }

    private void Details_Click(object sender, RoutedEventArgs e)
    {

      Car selectedCar = (sender as Button).Tag as Car;


      CarDetailsWindow detailsWindow = new CarDetailsWindow(selectedCar);


      detailsWindow.ShowDialog();
    }

    private void Edit_Click(object sender, RoutedEventArgs e)
    {

      Car selectedCar = (sender as Button).Tag as Car;


      EditCarWindow editCarWindow = new EditCarWindow(selectedCar);

      // Показуємо вікно редагування
      editCarWindow.ShowDialog();

      if (editCarWindow.DialogResult == true)
      {
        // Оновлюємо відповідний автомобіль у колекції
        int index = Cars_Colect.IndexOf(selectedCar);
        if (index != -1)
        {
          Cars_Colect[index] = editCarWindow.ModifiedCar;
        }
      }
    }

    private void Add_Click(object sender, RoutedEventArgs e)
    {
      // Створюємо та відкриваємо нове вікно для додавання автомобіля
      AddCarWindow addCarWindow = new AddCarWindow();
      if (addCarWindow.ShowDialog() == true)
      {
        Cars_Colect.Add(new Car
        {
          Brand = addCarWindow.txtBrand.Text,
          Model = addCarWindow.txtModel.Text,
          Year = addCarWindow.txtYear.Text,
          Color = addCarWindow.txtColor.Text,
          FuelType = addCarWindow.cmbFuelType.Text,
          EngineVolume = addCarWindow.txtEngineVolume.Text,
          VinCode = addCarWindow.txtVinCode.Text,
          LicensePlate = addCarWindow.txtLicensePlate.Text,
          Image = addCarWindow.CarImage
        });
      }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
      PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
```

```
        }

        private void ReturnToCategories_Click(object sender, RoutedEventArgs e)
        {
            _categoryMenuWindow.Show();
            this.Hide();
        }
    }


    public class Car : INotifyPropertyChanged
    {
        private string _brand;
        private string _model;
        private string _year;
        private string _color;
        private string _fuelType;
        private string _engineVolume;
        private string _vinCode;
        private string _licensePlate;

        private BitmapImage _image;

        public string Brand
        {
            get { return _brand; }
            set
            {
                _brand = value;
                OnPropertyChanged("Brand");
            }
        }

        public string Model
        {
            get { return _model; }
            set
            {
                _model = value;
                OnPropertyChanged("Model");
            }
        }

        public string Year
        {
            get { return _year; }
            set
            {
                _year = value;
                OnPropertyChanged("Year");
            }
        }

        public string Color
        {
            get { return _color; }
            set
            {
                _color = value;
                OnPropertyChanged("Color");
            }
        }

        public string FuelType
        {
            get { return _fuelType; }
            set
            {
                _fuelType = value;
                OnPropertyChanged("FuelType");
            }
        }

        public string EngineVolume
        {
            get { return _engineVolume; }
            set
            {
                _engineVolume = value;
                OnPropertyChanged("EngineVolume");
            }
        }

        public string VinCode
        {
            get { return _vinCode; }
            set
            {
                _vinCode = value;
                OnPropertyChanged("VinCode");
            }
        }
        public string LicensePlate
        {
            get { return _licensePlate; }
            set
            {
                _licensePlate = value;
                OnPropertyChanged("LicensePlate");
            }
        }
        public BitmapImage Image
        {
            get { return _image; }
            set
            {
                _image = value;
                OnPropertyChanged("Image");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
```

```
}
```

Код інтерфейсу екрану списку автомобілів

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

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

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Колекція" Height="450" Width="800">
    <Window.Resources>

        <Storyboard x:Key="buttonAnimation">
          <DoubleAnimation Storyboard.TargetProperty="Width" From="64" To="88" Duration="0:0:0.2" AutoReverse="True"/>
        </Storyboard>
    </Window.Resources>
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>



        <Button Content="Повернутись до категорій" Click="ReturnToCategories_Click" HorizontalAlignment="Left" VerticalAlignment="Top" Padding="10,5"
        Background="Gray" Foreground="White" BorderBrush="Gray"
        Margin="10">
        </Button>

        <!-- Контейнер -->
        <ScrollViewer Grid.Row="1" Grid.ColumnSpan="2">
          <StackPanel>
            <ItemsControl ItemsSource="{Binding Cars_Colect}">
              <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                  <WrapPanel />
                </ItemsPanelTemplate>
              </ItemsControl.ItemsPanel>
              <ItemsControl.ItemTemplate>
                <DataTemplate>
                  <Border BorderBrush="Black" BorderThickness="1" Margin="5">
                    <Border.Triggers>
                      <!-- Анімація появи елемента -->
                      <EventTrigger RoutedEvent="Loaded">
                        <BeginStoryboard>
                          <Storyboard>

<DoubleAnimation Storyboard.TargetProperty="Opacity" To="1" Duration="0:0:1"/>
                          </Storyboard>
                        </BeginStoryboard>
                      </EventTrigger>
                    </Border.Triggers>
                    <Grid Width="200">
                      <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                      </Grid.RowDefinitions>
                      <Image Source="{Binding Image}" Width="200" Height="150" Stretch="UniformToFill" />
                      <StackPanel Grid.Row="1" Margin="5">
                        <TextBlock Text="{Binding Brand}" FontWeight="Bold" />
                        <TextBlock Text="{Binding Model}" />
                        <Button Content="Деталі" Tag="{Binding}" Margin="0,5,0,0" Click="Details_Click"/>
                        <Button Content="Редагувати" Tag="{Binding}" Margin="0,5,0,0" Click="Edit_Click" Background="#FF4081"/>
                      </StackPanel>
                    </Grid>
                  </Border>
                </DataTemplate>
              </ItemsControl.ItemTemplate>
            </ItemsControl>
          </StackPanel>
        </ScrollViewer>



        <Button Content="Додати" Click="Add_Click" HorizontalAlignment="Right" VerticalAlignment="Top" Padding="10,5"
                Background="Green" Foreground="White" BorderBrush="Green"
                Grid.Row="0" Grid.Column="1" Margin="10">
          <Button.Style>
            <Style TargetType="Button">
              <Style.Triggers>
                <!-- Підсвічування при наведенні миші -->
                <Trigger Property="IsMouseOver" Value="True">
                  <Setter Property="Background" Value="LightGreen"/>
                </Trigger>
              </Style.Triggers>
            </Style>
          </Button.Style>

          <Button.Triggers>
            <EventTrigger RoutedEvent="MouseEnter">
              <BeginStoryboard
```

```
Storyboard="{StaticResource buttonAnimation}" />
                    </EventTrigger>
                </Button.Triggers>
            </Button>
        </Grid>
    </Window>
```

Код логіки вікна додавання автомобіля

```
using Microsoft.Win32;
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace Cars_Colect
{
    public partial class AddCarWindow : Window
    {
        public BitmapImage CarImage { get; set; }

        public AddCarWindow()
        {
            InitializeComponent();
            SetPlaceholderText();
            txtEngineVolume.TextChanged += txtEngineVolume_TextChanged;
        }

        private void SetPlaceholderText()
        {
            SetPlaceholder(txtBrand, "Марка");
            SetPlaceholder(txtModel, "Модель");
            SetPlaceholder(txtYear, "Рік випуску");
            SetPlaceholder(txtColor, "Колір");
            SetPlaceholder(txtEngineVolume, "Об'єм двигуна (л)");
            SetPlaceholder(txtVinCode, "Він код");
            SetPlaceholder(txtLicensePlate, "Реєстраційний номер");
        }

        private void SetPlaceholder(TextBox textBox, string placeholder)
        {
            textBox.Text = placeholder;
            textBox.Tag = placeholder;
            textBox.Foreground = Brushes.Gray;
        }

        private void TextBox_GotFocus(object sender, RoutedEventArgs e)
        {
            TextBox textBox = sender as TextBox;
            if (textBox != null && textBox.Text.Equals(textBox.Tag))
            {
                textBox.Text = "";
                textBox.Foreground = Brushes.Black;
            }
        }

        private void TextBox_LostFocus(object sender, RoutedEventArgs e)
        {
            TextBox textBox = sender as TextBox;
            if (textBox != null && string.IsNullOrWhiteSpace(textBox.Text))
            {
                textBox.Text = (string)textBox.Tag;
                textBox.Foreground = Brushes.Gray;
            }
        }

        private void txtEngineVolume_TextChanged(object sender, TextChangedEventArgs e)
        {
            if (!double.TryParse(txtEngineVolume.Text, out _))
            {
                txtEngineVolume.BorderBrush = Brushes.Red;
            }
            else
            {
                txtEngineVolume.BorderBrush = Brushes.Gray;
            }
        }

        private void AddImage_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Image files (*.jpg, *.jpeg, *.png) | *.jpg; *.jpeg; *.png";
            if (openFileDialog.ShowDialog() == true)
            {
                string imagePath = openFileDialog.FileName;

                CarImage = new BitmapImage(new Uri(imagePath));

                imgSample.Source = CarImage;
            }
        }

        private void Add_Click(object sender, RoutedEventArgs e)
        {
            if (!double.TryParse(txtEngineVolume.Text, out double engineVolume))
            {
                MessageBox.Show("Будь ласка, введіть коректний об'єм двигуна (числове значення).", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
```

```
                return;
            }

            string brand = txtBrand.Text;
            string model = txtModel.Text;
            string year = txtYear.Text;
            string color = txtColor.Text;
            string vinCode = txtVinCode.Text;
            string licensePlate = txtLicensePlate.Text;

            DialogResult = true;
        }
    }
}
```

Код інтерфейсу вікна додавання авто

```
<Window x:Class="Cars_Colect.AddCarWindow"

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

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Додати автомобіль" Height="420" Width="400">
    <Grid Background="#FFE5E5E5">

        <Grid Margin="10">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>

            <TextBox x:Name="txtBrand" Grid.Row="0" Margin="0 0 0 5" VerticalAlignment="Center" GotFocus="TextBox_GotFocus" LostFocus="TextBox_LostFocus"/>

            <TextBox x:Name="txtModel" Grid.Row="1" Margin="0 0 0 5" VerticalAlignment="Center" GotFocus="TextBox_GotFocus" LostFocus="TextBox_LostFocus"/>

            <TextBox x:Name="txtYear" Grid.Row="2" Margin="0 0 0 5" VerticalAlignment="Center" GotFocus="TextBox_GotFocus" LostFocus="TextBox_LostFocus"/>

            <TextBox x:Name="txtColor" Grid.Row="3" Margin="0 0 0 5" VerticalAlignment="Center" GotFocus="TextBox_GotFocus" LostFocus="TextBox_LostFocus"/>

            <Label Content="Вид палива:" Grid.Row="4"/>
            <ComboBox x:Name="cmbFuelType" Grid.Row="5" Margin="0 0 0 5" VerticalAlignment="Center">

<ComboBoxItem>Бензин</ComboBoxItem>

<ComboBoxItem>Дизель</ComboBoxItem>

<ComboBoxItem>Електро</ComboBoxItem>
            </ComboBox>

            <TextBox x:Name="txtEngineVolume" Grid.Row="6" Margin="0 0 0 5" VerticalAlignment="Center" GotFocus="TextBox_GotFocus" LostFocus="TextBox_LostFocus"/>

            <TextBox x:Name="txtVinCode" Grid.Row="7" Margin="0 0 0 5" VerticalAlignment="Center" GotFocus="TextBox_GotFocus" LostFocus="TextBox_LostFocus"/>

            <TextBox x:Name="txtLicensePlate" Grid.Row="8" GotFocus="TextBox_GotFocus" LostFocus="TextBox_LostFocus" HorizontalAlignment="Center" VerticalAlignment="Top" Width="376" Margin="0,0,0,5"/>

            <Image x:Name="imgSample" Grid.Row="8" Margin="0 30 0 5" Stretch="Uniform" Width="100" Height="100" Source="images/sampleimage.jpg"/>

            <StackPanel Grid.Row="9" HorizontalAlignment="Center" Orientation="Horizontal" Margin="0 0 0 5">
                <Button Content="Створити" Click="Add_Click" Margin="5" Width="100" Height="30" Background="#FF4CAF50" Foreground="White" FontWeight="Bold"/>
                <Button Content="Додати зображення" Click="AddImage_Click" Margin="5" Width="150" Height="30"/>
            </StackPanel>
        </Grid>
    </Grid>
</Window>
```