МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ПРОСЯННІКОВ АНДРІЙ ВАДИМОВИЧ

Допускається до захисту:
Завідувач кафедри інформаційних технологій,
к. т. н. доцент
Зелінська О. В.
«__» _____ 2024р.

Розробка програми для індивідуальної конфігурації автомобіля, що купується через мережу автосалонів з використанням сучасних архітектурних шаблонів

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Антонов Ю.С., доцент кафедри
інформаційних технологій,
кандидат фіз.-мат. н., доцент

Оцінка ___/___/_____
(бали за шкалою ЄКТС/ за національною шкалою)
Голова ЕК:_____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Просянніков А.В. Розробка програми для індивідуальної конфігурації автомобіля, що купується через мережу автосалонів з використанням сучасних архітектурних шаблонів. Спеціальність 122 «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Кваліфікаційна (бакалаврська) робота присвячена розробці програми для індивідуальної конфігурації автомобіля, що купується через мережу автосалонів з використанням сучасних архітектурних шаблонів. Так, як існує велика кількість автомобільних компаній з власними видами індивідуальної конфігурації, які відрізняються один від одного.

Ключові слова: автомобіль, архітектурний шаблон, .NET, Windows Forms, MVP.

78 с., 27 рис., 58 джерел.

ABSTRACT

Prosiannikov A.V. Development of an Application for Individual Configuration of a Car Purchased Through a Network of Car Dealerships Using Modern Architectural Templates. Specialty 122 "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2024.

The qualification (bachelor's) work is devoted to development of an application for individual configuration of a car purchased through a network of car dealerships using modern architectural templates. Since there is a large number of car companies with their own types of individual configuration, which differ from each other.

Keywords: car, architectural pattern, .NET, Windows Forms, MVP.

78 p., 27 pict., 58 source.

ЗМІСТ

ВСТУП

Актуальність роботи:

В сучасному світі існує велика кількість виробників авто. У кожного з них є власні інструменти для індивідуальної конфігурації авто для клієнта. Вони відрізняються інтерфейсом і дизайном, функціональністю, доступними опціями, наявністю або відсутністю інтерактивності та технологічними можливостями. Тому не завжди зручно і інтуїтивно зрозуміло як користуватись кожним з них. Для цього потрібно ввести стандарти або розробити унітарну програму для індивідуальної конфігурації.

Мета дослідження: Розробка програми для індивідуальної конфігурації автомобіля, що купується через мережу автосалонів з використанням сучасних архітектурних шаблонів.

Завдання дослідження:

- Огляд існуючих аналогів
- Ознайомлення з існуючими архітектурними шаблонами
- Постановка задачі
- Проектування програми
- Реалізація програми
- Тестування та верифікація програми

Об’єкт дослідження: існуючі засоби для індивідуальної конфігурації автомобіля, їх дослідження та порівняння

Предмет дослідження: розробка засобу індивідуальної конфігурації авто

Апробація результатів дослідження: результати досліджень апробовані на V Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених (м.Вінниця 23.05.2024 р.)

Практичне значення одержаних результатів: створення незалежного додатку для покращення зручності конфігурації авто різних брендів

Структура кваліфікаційної роботи: Бакалаврська робота складається із вступу, чотирьох розділів, висновків до них та списку використаних джерел

РОЗДІЛ 1. Програми для конфігурації та використання архітектурних шаблонів в них

1.1 Значення індивідуалізації в автомобільній галузі

Автомобільна галузь займає 11% від загального ВВП світу [1]. Станом на 2022 рік у світі є 82 мільйон автомобілів [2]. Отже, попит на автомобілі є дуже великий. Чим більше попит на автомобілі, тим більше автомобільні компанії їх виготовляють, що в свою чергу забезпечує збільшення кількості робочих місць та збільшення економіки країн. З можливістю індивідуалізації авто, попит на них зростає ще більше. Тому це дуже важливий аспект як для галузі так і для економіки світу в цілому.

В цьому світі кожен є особистістю, яка відрізняється від інших. Також в автомобільній галузі авто різних виробників відрізняються між собою. Тому індивідуалізація має велике значення як для виробників, так і для споживачів.
В першу чергу автомобільні компанії повинні задовільнити потреби своїх цільових клієнтів. Сучасні споживачі часто шукають можливі рішення для підкреслення особистої індивідуальності та відмінності від інших. Щоб вони відповідали їх унікальним потребам, смакам і стилю життя. Індивідуалізація дає змогу кожному клієнту налаштовувати свій власний автомобіль з урахуванням особистих уподобань. Це може бути вибір кольору, обладнання, набору технологій, форми та розміру крісла, діаметру та стилю дисків, різні обвіси, такі як спойлер, пороги, решітка радіатора, ширина колісних арок та багато інших.

Можливість індивідуалізації підвищує конкурентоспроможність виробників. Автовиробники, які можуть надавати широкі можливості для індивідуалізації своїх автомобілів, мають перевагу на ринку. Це дозволяє їм привертати більше клієнтів, які шукають унікальність і персоналізацію.

Підвищення лояльності клієнтів збільшується за наявності більшої кількості опцій для індивідуалізації власного авто. Клієнти, які мають можливість індивідуалізувати свій автомобіль, зазвичай розвивають більш тісні зв'язки з

маркою. Це може призвести до підвищення лояльності до бренду і повторних купівель авто їх марки.

Автовиробники збільшують свій заробіток за допомогою додавання різних опцій за додаткову плату. За рахунок продажу різного можливого обладнання та послуг, які включаються в індивідуальні пакети, автомобільні бренди заробляють більше, що прияє їх розвитку та поширенню у світі.

Автомобільний тюнінг [3] бере свої початки у 1920-х роках в США, що сильно вплинуло на їх культуру та відобразилось на кіно і музиці. Перші ідеї модифікувати свої автомобілі з’явилися одночасно з появою перших автомобілів на початку 20-го століття. Проте саме в середині 1920-х років почали з’являтись перші спеціалізовані майстерні, які займались тюнінгом автомобілів [4]. У той час люди спробували покращити характеристики своїх автомобілів, використовуючи різні методи, такі як заміна компонентів двигуна та зміна зовнішнього вигляду. З тих пір тюнінг став невід'ємною частиною автомобільної культури, а індустрія тюнінгу значно розширилася та стала глобальною. У той час були популярні заїзди на максимальну швидкість та почали з’являтися перші перегони. Заїзди на швидкість проводили на соляних озерах у штаті Флорида, таких як Бонневілльське солоне озеро та солоне озеро Уейнеман [5]. Ці перегони виникли як результат пошуку великої площі землі з рівною поверхнею, ідеальної для швидкісних випробувань автомобілів і мотоциклів. Солоні озера Флориди, з їхнім великим, плоским і, головне, сухим ліжком, стали ідеальним місцем для таких заходів. Перші перегони на солоних озерах були вкрай популярними серед автомобілістів та швидкісних ентузіастів, які змагалися за рекордні швидкості на великих відкритих просторах. Ці події відіграли важливу роль у розвитку автомобільних технологій та популяризації мотоспорту в США і по всьому світу. Один з найвідоміших заходів цього типу - "Speed Week" [6] на Бонневілльському солоному озері в штаті Юта, який проводиться щорічно з початку 20-х років минулого століття і є одним з найважливіших заходів у світі швидкісного автомобільного спорту. Ентузіасти модифікували свої автомобілі щоб ставити нові рекорди. Потім це переросло в Драг-рейсинг – спринтерські заїзди на одну

четверту милі. Перші кільцеві гонки проводили на берегах моря або по місту, що було дуже небезпечно. Так виникли автогонки Індіанаполіс 500 або просто Інді 500. Перші Індіанаполіс 500 відбулися у 1911 році на трасі Indianapolis Motor Speedway в Індіанаполісі, штат Індіана, США [7]. Гонка була організована клубом Indianapolis Motor Speedway Company. Протягом наступних десятиліть Інді 500 стала визнаним символом автомобільного спорту. Гонка розвивалася, збільшуючи швидкості та вдосконалюючи безпеку. Вона також стала однією з найбільш популярних гонок в світі, привертаючи сотні тисяч глядачів. У Інді 500 виступали багато знаменитих гонщиків, таких як A.J. Foyt, Mario Andretti, Al Unser та Rick Mears. Також багато відомих автомобільних виробників приймали участь у гонці, змагаючись за перемогу та рекламу своїх машин. Інді 500 була платформою для впровадження нових технологій та інновацій у світі автомобільного спорту. Багато нововведень, які з'явилися на гонках, потім знаходили свій шлях до виробництва автомобілів загалом. У наші часи Інді 500 продовжує бути однією з найбільш очікуваних і спостережуваних видів автомобільних перегонів у світі. Кожного року тисячі фанатів збираються на Indianapolis Motor Speedway, щоб підтримати своїх улюблених гонщиків і насолодитися захопливою битвою на трасі. З появою комп'ютерних технологій в 1980-х роках, тюнінг ательє стали використовувати програмні засоби для зміни параметрів двигуна автомобіля. Це дозволяє встановлювати оптимальні настройки двигуна та інших систем автомобіля, що дозволяє забезпечувати вищу продуктивність та ефективність автомобіля [4]. Розвиток технологій в галузі автоперегонів стимулював попит на модифікування звичайних авто. Це в свою чергу стимулювало розвиток індивідуалізації автомобілів. Індивідуалізація може стимулювати розвиток нових технологій в автомобільній галузі, таких як розширена реальність для віртуального огляду автомобілів або розумна система конфігурації, яка пропонує рекомендації клієнтам. Що в свою чергу стимулює розвиток різних технологій у сфері IT та виробництві додаткових деталей для авто.

Є багато відомих тюнінг ательє [8], які займаються починаючи з детейлінгу закінчуючи побудовою майже нової машини на базі попередньої. Одне з них з'явилось у 1967 році – це всесвітньо відоме ательє AMG [9]. Воно спеціалізується на машинах марки Mercedes-Benz. Засновниками є два інженера Ганс Вернер Ауфрехт і Ерхард Мельхер. Спочатку вони просто тюнінгували серійні автомобілі, але згодом стали виробляти власні моделі AMG, такі як відомі моделі E63 AMG або C63 AMG. Друге це AC Schnitzer [10]. Німецьке тюнінг ательє, яке спеціалізується на модифікаціях BMW. Яке було засновано Івом Арнстом в 1987 році. Вони відомі своїми високоякісними комплектами змін, які включають у себе тюнінг двигуна, підвіску, косметичні зміни та інші аспекти автомобіля. Наступне по рахунку, але не по якості, це Brabus [11]. Це ще одне німецьке тюнінг ательє, яке спеціалізується на модифікаціях Mercedes-Benz, Smart і Tesla. Brabus було засновано в 1977 році, і відоме своїми вражаючими втручанням в двигун, екстер'єр та інтер'єр автомобілів. Дуже популярне ательє, яке займається розробкою та встановленням екстремальних кіт-комплектів стилю "wide-body" – це Liberty Walk [12]. Засновник, Кіто Кіріджі, відомий своєю сміливістю в дизайні та відмінною роботою з карбоновими волокнами. RUF Automobile [13] – легендарне тюнінг ательє, яке спеціалізується на автомобілях марки Porsche. Засноване Алоїсом Руфом у 1939 році, RUF відоме своїми моделями, які базуються на шасі Porsche, але мають власні двигуни та стильні зміни в дизайні. Піком популярності тюнінгу автомобілів вважаються 90-х роки у Японії. Тоді зародилась культура "JDM" [14], що означає японський внутрішній ринок та відноситься до автомобілів призначених для продажу в Японії. Це пов'язано з угодою між Японськими автовиробниками, щоб обмежити потужність двигунів, за для безпеки на дорогах. В свою чергу вони почали закладати неабияку міцність в будову двигунів, що дозволяло сильно підняти їх потужність з невеликим доопрацюванням. Це зародило цілу культуру нелегальних гонок по місту на великих швидкостях. Почали з'являтись цілі клани гонщиків, в деякі з них не брали якщо твоє авто не може їхати мінімум 200

кілометрів на годину. З того часу тюнінг авто, особливо японських, став набирати свою популярність та залишається популярним і на сьогодні.

В свою чергу є багато різних видів тюнінгу та індивідуалізації авто [15]. А саме:

- Зовнішній вигляд:
  - Зміна кузова: обробка або заміна кузовних деталей, спойлерів, обвісів тощо.
  - Фарбування та графічний дизайн: використання спеціальних фарб, вінілу, автомобільних плівок для створення унікального зовнішнього вигляду.
  - Модифікація колес: встановлення великих або спеціально виготовлених дисків, встановлення низькопрофільних шин.
- Внутрішній дизайн:
  - Відновлення салону: заміна тканин, шкіри або додавання додаткових функцій, таких як підігрів сидінь, підсвітка салону тощо.
  - Звукове обладнання: встановлення музичних систем вищого класу, додаткових динаміків, сабвуферів.
- Технічні покращення:
  - Підвіска: встановлення спортивної або адаптивної підвіски для кращої керованості і комфорту.
  - Двигун: покращення потужності, турбонаддув, чіп-тюнінг, встановлення вихлопної системи тощо.
- Електроніка:
  - Встановлення додаткових електронних пристроїв: навігаційні системи, системи безпеки, додаткові екрани тощо.
- Індивідуальні додатки:
  - Персоналізація: додавання деталей або елементів, які виражають індивідуальний стиль власника автомобіля.

1.2 Переваги використання та види спеціалізованих сервісів та програм

Переваги використання спеціалізованих програм та сервісів для індивідуальної конфігурації авто є як для клієнтів, так і для автовиробників.

Перевагами для клієнтів є:

- Персоналізація: Клієнти мають можливість створити автомобіль, який відповідає їхнім унікальним потребам, вподобанням і стилю.
- Більший вибір: Спеціалізовані сервіси дозволяють клієнтам вибирати з більш широкого асортименту опцій, обладнання та аксесуарів, що робить конфігурацію більш гнучкою і зручною.
- Реалізація ідеального автомобіля: Клієнти можуть створити автомобіль, який ідеально відповідає їхнім потребам і бажанням, забезпечуючи максимальний комфорт і задоволення від власництва.
- Додаткові можливості: Деякі спеціалізовані сервіси можуть пропонувати додаткові послуги, такі як фінансування, страхування, а також сервісні пакети, які полегшують процес придбання та утримання автомобіля.

Перевагами для виробників є:

- Підвищення продажів: За рахунок надання можливості індивідуальної конфігурації, виробники можуть залучити більше клієнтів і збільшити обсяги продажів.
- Збільшення лояльності клієнтів: Персоналізація автомобіля може сприяти розвитку тісних зв'язків між клієнтом і брендом, що призводить до збільшення лояльності та повторних покупок.
- Додаткові доходи: Виробники можуть заробляти додаткові кошти на продажу додаткового обладнання, аксесуарів та послуг, які клієнти додають до індивідуальної конфігурації.

Є різні сервіси та програми для конфігурації свого авто. Це можуть бути:

- Онлайн-конфігуратори на веб-сайтах таких виробників [16–18], як Mercedes-Benz [19,20], BMW [21,22] та Porsche [23,24]. Ці інтерактивні

інструменти дозволяють клієнтам налаштовувати свій автомобіль безпосередньо на веб-сайті виробника.

- Мобільні додатки для діагностики автомобіля [25]. Додатки, які можна завантажити на смартфони, надають зручний спосіб проводити діагностику автомобіля в будь-якому місці.
- Спеціалізовані консультаційні сервіси [26,27]. Деякі виробники надають послуги персональних консультантів, які допомагають клієнтам вибрати оптимальну конфігурацію автомобіля.
- Платформи з огляду автомобілів в розширеній реальності (AR) [28]. Вони дозволяють клієнтам віртуально оглядати автомобілі та експериментувати з різними опціями та конфігураціями.

Аналіз вже існуючих конфігураторів автомобілів, визначення їх відмінностей, плюсів та мінусів. Для аналізу було обрано конфігуратори від таких автовиробників, як Mercedes-Benz [18], BMW [17] та Porsche [16]. Аналіз проводиться за такими показниками:

1. Інтуїтивність інтерфейсу
2. Зручність використання
3. Актуальність наявних авто
4. Способи оплати
5. Терміни та тип доставки

Конфігуратор від компанії Mercedes-Benz:

1. Інтуїтивність інтерфейсу:
    1) Всі елементи інтерфейсу зрозумілі
    2) Дизайн приємний
    3) Шрифт добре видимий та легкий до прочитання
2. Зручність використання:
    1) Присутня можливість вибору модельного ряду авто
    2) Присутній список комплектацій для кожної з моделей авто
    3) Представлені параметри автомобілів та їх комплектацій
    4) Присутня можливість відмінити свій вибір

5) Присутнє фото яке відображає усі зміни користувача при вибору модифікацій

3. Актуальність наявних авто:
    1) Всі представлені авто є актуальними
    2) Не всі авто можна замовити
    3) Не всі авто можна модифікувати
4. Способи оплати:
    1) Оплата можлива при отримані готівкою або безготівковим способом
5. Терміни доставки:
    1) Доставка можлива лише на підрозділ дилера авто
    2) Терміни залежать від наявності вибраної конфігурації авто вони можуть займати від одного тижня до трьох місяців

Конфігуратор від компанії BMW:

1. Інтуїтивність інтерфейсу:
    1) Всі елементи інтерфейсу зрозумілі
    2) Дизайн приємний
    3) Шрифт добре видимий та легкий до прочитання
2. Зручність використання:
    1) Присутня можливість вибору модельного ряду авто
    2) Присутній список комплектацій для кожної з моделей авто
    3) Представлені параметри автомобілів та їх комплектацій
    4) Присутня можливість відмінити свій вибір
    5) Присутнє фото яке відображає усі зміни користувача при вибору модифікацій
3. Актуальність наявних авто:
    1) Всі представлені авто є актуальними
    2) Всі авто можна замовити
    3) Всі авто можна модифікувати
4. Способи оплати:

1) Оплата можлива при отримані готівкою або безготівковим способом

5. Терміни доставки:
    1) Доставка можлива лише на підрозділ дилера авто
    2) Терміни залежать від наявності вибраної конфігурації авто вони можуть займати від одного тижня до одного року

Конфігуратор від компанії Porsche:

1. Інтуїтивність інтерфейсу:
    1) Всі елементи інтерфейсу зрозумілі
    2) Дизайн приємний
    3) Шрифт добре видимий та легкий до прочитання
2. Зручність використання:
    1) Присутня можливість вибору модельного ряду авто
    2) Присутній список комплектацій для кожної з моделей авто
    3) Представлені параметри автомобілів та їх комплектацій
    4) Присутня можливість відмінити свій вибір
    5) Присутнє фото яке відображає усі зміни користувача при вибору модифікацій
3. Актуальність наявних авто:
    1) Всі представлені авто є актуальними
    2) Всі авто можна замовити
    3) Всі авто можна модифікувати
4. Способи оплати:
    1) Оплата можлива при отримані готівкою або безготівковим способом
5. Терміни доставки:
    1) Доставка можлива лише на підрозділ дилера авто
    2) Терміни залежать від наявності вибраної конфігурації авто вони можуть займати від одного тижня до одного року

Порівняння відмінностей між представленими конфігураторами авто:

1. Інтуїтивність інтерфейсу у кожному представленому варіанті є однаковою для користувача. Але у кожного різний дизайн.
2. Зручність використання кожного представленому варіанту є однаково зручною для користувача.
3. Актуальність наявних авто в компаній BMW та Porsche однаково на високому рівні, в компанії Mercedes-Benz можна замовити не всі представлені авто.
4. Способи оплати однакові у кожному з представлених варіантів.
5. Терміни доставки в BMW та Porsche однакові від одного тижня до одного року, в компанії Mercedes-Benz вони менші від одного тижня до трьох місяців.

Визначення плюсів та мінусів кожного з конфігураторів:

1. Mercedes-Benz:

   Плюси – Інтерфейс, Зручність використання, Способи оплати, Термін доставки

   Мінуси – Не всі представлені моделі можна заказати
2. BMW:

   Плюси – Інтерфейс, Зручність використання, Всі представлені моделі можна заказати, Способи оплати

   Мінуси – Доставка авто може тривати рік
3. Porsche:

   Плюси – Інтерфейс, Зручність використання, Всі представлені моделі можна заказати, Способи оплати

   Мінуси – Доставка авто може тривати рік

Отже, у кожного варіанту є свої плюси та мінуси. Кожен з розглянутих варіантів виконаний на високому рівні. Явних фаворитів виявити неможливо, тому що їх плюси та мінуси нівелюють один одного. BMW та Porsche завдяки актуальності наявних авто. Mercedes-Benz завдяки менших термінів доставки.

1.3 Значення архітектурних шаблонів у сучасному програмуванні

Архітектурні шаблони - це загальні рішення для типових проблем в архітектурі програмного забезпечення [29]. Їх використовують для створення структурованих, ефективних і легко розширюваних систем.

У 1979 році було вперше введено поняття “проектування на основі шаблонів” [30]. Було розглянуто програмування як сукупність шаблонів, які можна було б використовувати для розробки нових систем. Справжній прорив в цій області відбувся у 1994 році, коли опублікували книгу "Design Patterns: Elements of Reusable Object-Oriented Software" [31], авторами якої були Еріх Гамма, Річард Хелм, Ральф Джонсон та Джон Вліссідес, відомі як "Банду чотирьох" (Gang of Four). Книга містила опис 23 різних шаблонів проектування, таких як Singleton, Factory Method, Observer, та інші. Після цього архітектурні шаблони програмування стали загальноприйнятим підходом у розробці ПЗ.

Архітектурні шаблони мають велике значення в сучасному програмуванні. Вони допомагають підвищити якість програмного забезпечення, тобто створювати системи, які є більш надійними, масштабованими та безпечнішими, завдяки добре протестованих і перевірених рішень для певних проблем. Підвищити продуктивність розробки, щоб розробники уникали винаходження велосипедів, тобто вони використовують вже існуючі та перевірені рішення та адаптують їх для розв’язування своїх задач. Що значно зменшує час, який необхідний для розробки програмного забезпечення. Також їх використовують для стандартизації та виключення непорозуміння між розробниками. Дозволяє уникати “merge conflicts” [32], при використані різних систем контролю версій, таких як Git [33,34], GitLab [35,36], Bitbucket [37], Jira [38], Replit [39] та інші. Дозволяє розробникам спільно розуміти структуру та організацію системи, що полегшує комунікацію між членами команди та сприяє використанню спільних термінів та понять. Завдяки використанню шаблонів можна легко підтримувати рефакторинг та розширення коду. На їх основі створюється гнучка база для подальшого розвитку системи, легкого розширення функціоналу та внесення змін в архітектуру без значних витрат часу та зусиль. Подальші вивчення

архітектурних шаблонів сприяють навчанню та освоєнню нових технологій для розробників. Вони можуть їх адаптувати під певне програмне забезпечення, що сприяє їхньому професійному зростанню та розвитку як розробників, що підвищує рівень експертизи.

Отже, архітектурні шаблони важливі для сучасного програмування. Вони допомагають побудувати структуру проекту для його розробки та полегшити її.

1.4 Огляд популярних архітектурних шаблонів

Є велике різноманіття архітектурних шаблонів, кожен з них використовується у своїй галузі розробки програмного забезпечення.

Перший та найбільш популярний шаблон це MVC – Model-View-Controller [40]. Цей шаблон зазвичай використовується для розробки користувацьких інтерфейсів, він поділяє зв’язану логіку програми на три взаємопов’язаних елементи. А саме модель, вигляд та контролер. Модель є центральним компонентом даного шаблону та відповідає за поведінку програми, незалежну від інтерфейсу користувача. Модель прямо керує даними, логікою та правилами застосунку. Є активна та пасивна модель. В активній моделі вигляд відслідковує зміни в моделі та реагує на них. В пасивній моделі вигляд оновлюється через контролер. Вигляд являє собою будь-яке представлення інформації, яка отримується на виході, незалежно від інтерфейсу самого користувача. Зазвичай одночасно співіснують відразу декілька виглядів однакової інформації, наприклад діаграма для керівника компанії та таблиці для бухгалтерів. Контролер в свою чергу лише отримує вхідні дані та перетворює їх на зрозумілі команди для вигляду чи моделі. Зазвичай цей шаблон використовується для веб-додатків, в яких контролер обробляє запити від користувачів, модель взаємодіє з базою даних, а вигляд лише відображає дані користувачу. Графічно представлена діаграма роботи архітектурного шаблону Model-View-Controller на рис. 1.1.

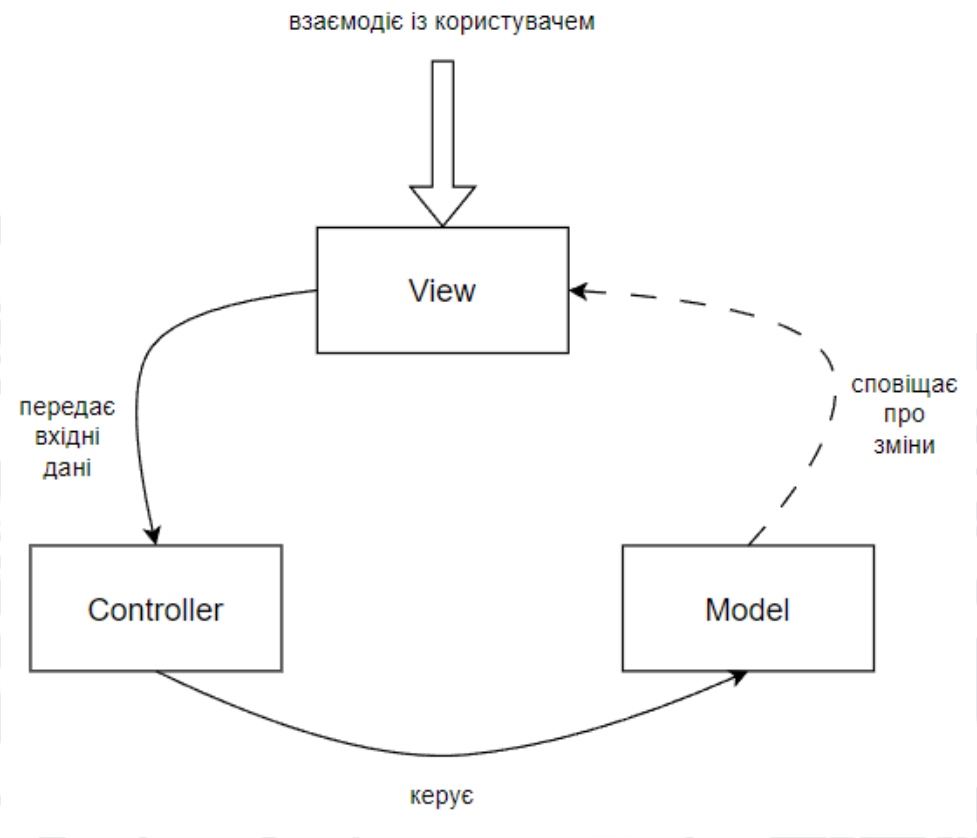

Рисунок 1.1 Діаграма взаємодії між компонентами шаблону MVC

Другий шаблон це MVP – Model-View-Presenter [41]. Цей шаблон поділяється на ті ж компоненти, що і MVC, але замінює контролер на пред'явник (Presenter). Цей шаблон відокремлює логіку вигляду він бізнес-логіки моделі та проміжного коду пред'явника. Пред'явник відповідає за обробку запитів користувача та оновлення даних в моделі. Він спрощує тестування та дає можливість розширення коду, так як логіка не прив’язана напряму до вигляду, тобто інтерфейсу який бачить користувач. Даний шаблон використовується для розробки додатків на основі Windows Forms, особливо тих, які мають в собі складку бізнес-логіку та вимагають детального тестування. Графічно представлена діаграма роботи архітектурного шаблону Model-View-Presenter на рис. 1.2.

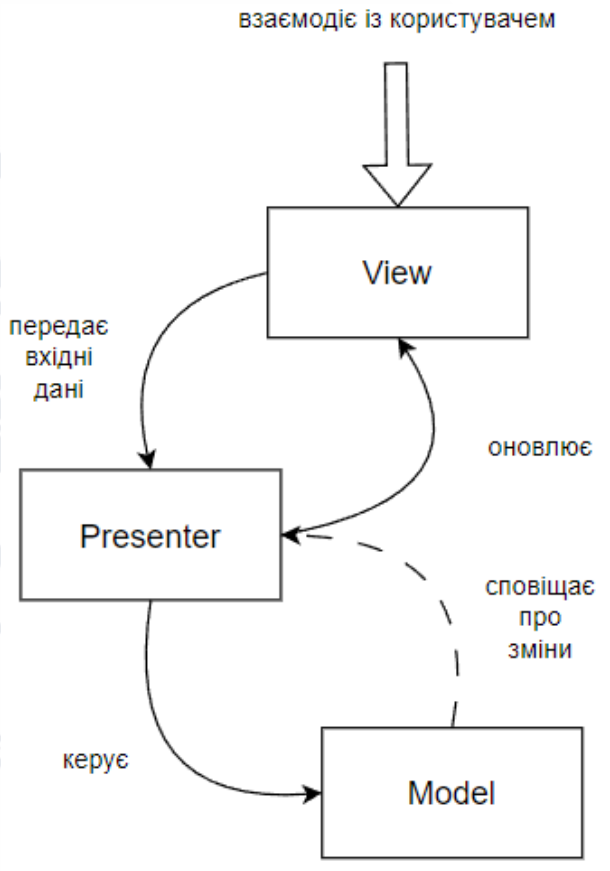

Рисунок 1.2 Діаграма взаємодії між компонентами шаблону MVP

Третій шаблон це MVVM – Model-View-ViewModel [42]. Цей шаблон використовується для проектування структури десктоп додатків. Найчастіше його застосовують при розробці програм на сучасних платформах розробки, таких як Windows Presentation Foundation (WPF) та Silverlight від компанії Microsoft.Шаблон MVVM дуже схожий на MVC та MVP, він використовується для розподілення функцій модуля контролер та заміняє його на модуль ViewModel. ViewModel відповідає за представлення даних та стану для вигляду. ViewModel відображає дані та стан моделі, що дозволяє використовувати механізми зв'язування даних .NET для автоматичного оновлення інтерфейсу користувача при зміні даних. Він розподіляє логіку вигляду управління даними. Надає можливість змінювати їх незалежно один від одного. Розподіляє роботу розробників, щоб дизайнер міг працювати з користувацьким інтерфейсом, а розробник, в свою чергу, працює над логікою оброки даних та роботи з даними.

Графічно представлена діаграма роботи архітектурного шаблону Model-View-ViewModel на рис. 1.3.

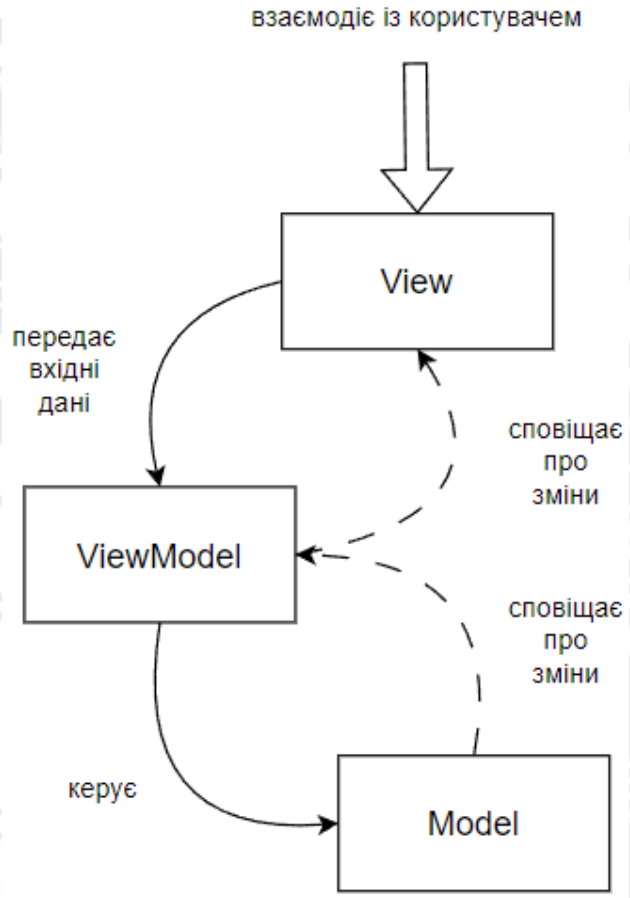

Рисунок 1.3 Діаграма взаємодії між компонентами шаблону MVVM

Дані шаблони, їх структура, принципи роботи та інше, були опубліковані у статі Мартіна Фаулера – відомого британського інженер-програміста.

1.5 Обґрунтування вибору архітектурного шаблону та мови програмування для розробки

Для реалізації програми для індивідуальної конфігурації авто, була обрана мова C# [43] , платформа .NET Framework [44**Ошибка! Источник ссылки не найден.**], Windows Forms [45] та архітектурний шаблон MVP [41].

C# це одна з найпопулярніших мов програмування, а .NET Framework – платформа для розробки різного програмного забезпечення, що

використовується для створення різних додатків, від веб-сайтів до десктопних програм та мобільних застосунків.

Мова програмування C# була розроблена у компанії Microsoft [46]. Це було пов’язано з їх стратегією розвитку програмного забезпечення. Наприкінці 90-х років компанія Microsoft розробляла .Net Framework, коли Java [47] вже була найпопулярнішою мовою програмування для розробки веб-додатків та додатків в різних інших областях. Для цього було прийнято рішення створити свою мову програмування, яка могла б конкурувати з мовою Java. Історія C# розпочалась з проекту, який мав кодову назву “Cool”, що розшифровується як C-like Object Oriented Language (С-подібна об'єктно-орієнтована мова). Цей проект бере свій початок у 1999 році, керівником даного проекту був Андерс Гейлсберг, який був відомий своєю роботою над Turbo Pascal [48] і Delphi [49] в Borland. Мова C# була представлена як частина .NET Framework під час конференції для розробників Microsoft Professional Developers Conference, яка відбулась у липні 2000 року. Перша версія мала назву C# 1.0 та була випущена на початку 2002 року, разом з .NET Framework 1.0. Перша версія C# включала базові конструкції мови, такі як класи, інтерфейси, методи, властивості та події. Інтегрувалася з .NET Framework для розробки програм під платформу Windows. Містила ключові концепції об'єктно-орієнтованого програмування, такі як спадкування, поліморфізм, інкапсуляція.. З часом C# пройшла через кілька версій, кожна з яких вводила нові функції і покращення. У C# 2.0, яка була випущена у 2005 році, було додано підтримку generics, яка дозволяла розробникам створювати загальні (generic) класи та методи. Введено partial types, які дозволяли розбити оголошення класу на декілька частин у різних файлах. Nullable types дозволили змінним значенням змінюватися на null, що полегшило роботу зі значеннями, які можуть бути відсутніми. У C# 3.0, яка була випущена у 2007 році, додано лямбда-вирази, які дозволяють створювати анонімні функції без визначення окремих методів. Введено Language Integrated Query (LINQ), що надає можливість виконання запитів до джерел даних безпосередньо в мові програмування. Додано автоматичне властивості (automatic properties), які спрощують визначення

простих властивостей класів. У C# 4.0, яка була випущена у 2010 році, було введено dynamic types, що дозволяють обходити перевірки типів компілятором під час компіляції. Додано named and optional arguments, які дозволяють передавати аргументи до методів за іменами та з заданими значеннями за замовчуванням. Розширено підтримку коваріантності і контраваріантності для узагальнених типів. В свою чергу у C# 5.0, яку випустили у 2012 році, було менше інновацій, але вони були не менш важливі за попередні. Додали ключові слова async та await для асинхронного програмування, що дозволяє створювати асинхронні методи та очікувати їхнього завершення без блокування потоку. У C# 6.0, 2015 рік випуску, вже було більше інновацій. Додали ініціалізатори властивостей (property initializers) для спрощення створення об'єктів з властивостями. Введено null-conditional operators для спрощення перевірки null-значень. Додано строкові інтерполяції (string interpolation), які дозволяють вбудовувати значення змінних в рядки. У версії C# 7.0 та пізніші продовжують вводити нові функції та покращення. Вони включають в себе такі можливості, як pattern matching, local functions, ref returns і ref locals, record types, switch expressions та інші.

Синтаксис мови C# базується на C та C++, але на відміну від них він більш простий та має вищий рівень абстракції, що робить його більш доступним для новачків та зручним для професійних розробників. Мова має велику кількість вбудованих бібліотек та різних функцій, які спрощують розробку. Вона підтримує такі концепції, як делегати, атрибути, винятки та багато інших, що робить код чистим і ефективним. Мова C# являється об’єктно-орієнтованою, отже вона підтримує даний підхід до програмування, що в свою чергу сприяє створенню модульних програм, які можна повторно використовувати та полегшує їх заміну. Також вона інтегрується з великою кількістю інструментів для розробки програмного забезпечення, наприклад Visual Studio [50]. Visual Studio має в собі великий набір засобів для редагування коду, налагодження, відлагодження та управління проектами. Дане інтегроване середовище розробки (IDE) являється одною з найпопулярніших у світі. Компанія Microsoft почала її

розробку в кінці 80-х на початку 90-х років минулого століття, як проект з кодовим ім'ям "Project Marvel". Перша версія Visual Studio була призначена для розробки на Visual Basic та Visual C++, за допомогою базових інструментів розробки для цих мов програмування. Visual Studio 97 – це перша версія, яка отримала свою подальшу назву Visual Studio. Як можна зрозуміти з назви, вона була випущена у 1997 році. Вона в собі мала інтегровані середовища для розробки на Visual Basic, Visual C++, Visual FoxPro та інших мов програмування. Дана IDE постійно покращувалась, з часом оновлення включали в себе все більше нових інструментів та функцій для підтримки та розробки для нових технологій. З'явилась Visual Studio .NET, яка почала підтримувати розробку за допомогою платформи .NET Framework. Курс розвитку став направлений для можливості розробки на все більшій кількості платформ, Visual Studio стала універсальною IDE для розробки на різних мовах та платформах, включаючи Windows, Linux, Android, iOS, веб-розробку та інші. З появою і розвитком хмарних технологій відбулась інтеграція з ними, такими як Azure. Які дозволяють розробникам розробляти, тестувати та розгортати свої додатки за допомогою вбудованого інтерфейсу від Visual Studio. На сьогодні це один з найпотужніших інструментів для розробки програмного забезпечення з будь-яким рівнем складності. Завдяки підтримки дуже широкого спектру мов програмування, вона має в собі функціонал для розширення та має зручні інструменти для продуктивної розробки, що полегшує роботу розробникам.

Azure [51] – це хмарна платформа та частина інфраструктури Microsoft. Свій початок вона бере з 2008 року, в той час Microsoft розпочали роботу над своїм проектом, на той час, відомим як "Red Dog". Головна мета даного проекту була створення новітньої хмарної платформи, що надасть змогу своїм клієнтам розгортати, керувати та масштабувати свої додатки та послуги віддалено за допомогою інтернету. Azure була офіційно представлена у 2010 році, під час конференції Professional Developers Conference. Всього за декілька перших років платформа розвилась великими темпами, додалися нові функції та сервіси, такі як віртуальні машини, віддалені бази даних, аналітика та інші. Протягом

наступних років ця платформа ставала все популярнішою і популярнішою, особливо серед підприємств та розробників. Це все завдяки своїй гнучкості, можливості масштабованості та неабиякій надійності. Компанія Microsoft інвестувала немалі зусилля за для розвитку даної платформи, розширюючи функціонал, можливості та географічну доступність. На сьогодні платформа Azure – це одна з провідних хмарних платформ у цілому світі. Яка надає достатньо широкий спектр послуг, які включають обчислення, зберігання даних, штучний інтелект, інтернет речей та багато іншого. Тисячі компаній по всьому світу використовують її за для підтримки свої додатків та послуг у хмарі.

Для збереження інформації була обрана система управління базами даних Microsoft SQL Server [52]. Ця СУБД являється розробкою компанії Microsoft. Вона використовується для зберігання та керування даними. MS SQL Server використовує клієнт-серверну архітектуру, в якій клієнтами є програми, які взаємодіють з сервером, тобто базою даних, для зберігання даних та виконання запитів. Для цієї взаємодії використовується мова структурованих запитів SQL. Microsoft SQL Server в собі має великий набір функцій, підтримує роботу з реляційними базами даних, транзакції, забезпечує безпеку даних, наявна індексація, що підвищує швидкість пошуку даних, підтримує різні типи даних, наприклад текст, числа, графічні дані, має інструменти для аналітики та формування звітів. SQL Server має інтеграцію для роботи з іншими продуктами Microsoft, такими як Azure, SharePoint, Excel, Visual Studio та інші. Дана програма підтримує роботу з даними у реальному часі, це потрібно для роботи з потоковими даними. Безпека доволі на високому рівні, механізми для захисту включають в себе автентифікація та авторизація користувачів, ролева базова модель доступу, шифрування даних, аудит дій користувачів та відстеження змін. Також підтримує роботу з великим обсягом даних, що дозволяє використовувати її для великих компаній, забезпечуючи ефективність, доступність та продуктивність. У критичних ситуаціях, за для не втрати даних, є вбудоване резервне копіювання даних та можливість відновити дані. Для розширеного індивідуального функціоналу MS SQL Server підтримує додавання сторонніх

модулів. Підтримує всі популярні на сьогодні мови програмування, що полегшує роботу з базою та побудову запитів.

Мовою для програмування запитів в реляційних базах даних є SQL [53] (Structured Query Language). Вона включає в себе 4 основні складові: DDL (Data Definition Language), DML (Data Manipulation Language), DCL (Data Control Language) та TCL (Transaction Control Language). DDL використовується для визначення структури бази даних. Виконує такі операції, такі як створення таблиць, видалення таблиць, визначення ключів доступу, індексів. DML використовується для роботи з даними в базі. Це включає в себе такі операції, як вибірка даних (запити SELECT), вставка нових даних (операція INSERT), оновлення існуючих даних (операція UPDATE) та видалення даних (операція DELETE). DCL використовується для управління правами доступу до бази даних. Це включає операції, такі як GRANT (надання прав доступу) і REVOKE (відкликання прав доступу). TCL використовується для керування транзакціями в базі даних. Основні команди TCL включають COMMIT (збереження змін), ROLLBACK (відкат змін) і SAVEPOINT (встановлення точки збереження). Мова SQL має простий і зрозумілий синтаксис, що зробила її популярною серед розробників та аналітиків даних.

Платформа .NET Framework поставляється з великою бібліотекою класів, яка містить готовий функціонал для роботи з мережами, базами даних, графікою, криптографією та іншими завданнями. Вона підтримує багато мов програмування, включаючи C#, Visual Basic .NET, F#, Managed C++, тому розробник можете використовувати ту, яка йому найбільше подобається або яка найкраще підходить для його проекту. Платформа .NET Framework працює на різних платформах, включаючи Windows, Linux та macOS, що дозволяє створювати кросплатформенні додатки. Наприкінці 90-х років минулого століття, компанія Microsoft прагнула зробити розробку програмного забезпечення більш ефективною та простішою. У цей період керівництво компанії визнала потребу в новій платформі, яка б могла підтримувати різні мови програмування та надавала б зручні інструменти для розробки програмного

забезпечення. У 2000 році було оголошено про план створення .NET Framework, який спочатку мав назву Next Generation Windows Services (NGWS). Його основана мета полягала у спрощені розробки програм, забезпечивши середовище, що могло б підтримувати декілька мов програмування та яке включає в себе багатий набір бібліотек. Перша версія була випущена у 2002 році разом з Visual Studio .NET. Це був великий крок у напрямку створення єдиної платформи яка могла б підтримувати розробку різних типів програмного забезпечення. У наступні роки Microsoft продовжувала розвивати та розширювати функціонал .NET Framework, додаючи нові функції, підтримку нових мов програмування та різних інструментів для розробників. З 2010 до 2020 року Microsoft оголосила про .NET Core - відкриту та кросплатформену версію .NET Framework. Сам .NET Core був заснований на новій архітектурі, що надавала підтримку для роботи на різних операційних системах, включаючи Windows, Linux і macOS. Подальші версії .NET стали об’єднанням платформ .NET Framework і .NET Core в одну спільну. Це стало кроком у напрямку єдиної, кросплатформенної та швидкої платформи для розробки програмного забезпечення.

Windows Forms – це фреймворк для створення графічних інтерфейсів користувача в програмах, що працюють під управлінням операційної системи Microsoft Windows. Фреймворк став доступний після випуску .NET Framework 1.0 у 2002 році. Він надав можливість створення класичних десктоп додатків за допомогою мов програмування C# або Visual Basic.NET. Windows Forms базується на подійно-орієнтованій моделі програмування. Щоб розробники мали змогу створити відгук на подію, наприклад натискання кнопок або зміни значень у текстових полях. Для розробки програм за допомогою Windows Forms використовується Visual Studio, яка надає зручні інструменти та інтерфейс. Популярність фреймворку швидко зросла у перші роки його існування, так як він надавав простіший і швидший спосіб створення класичних додатків для Windows, ніж WinAPI. Фреймворк Windows Forms продовжував розвиватись та покращуватись з додаванням нових функцій та інструментів. Але його

популярність трішки зменшилась з виходом більш новітніх WPF та UWP. Незважаючи на це він використовується і сьогодні, особливо в областях, де потрібно швидко та просто створювати десктоп додатки.

Для роботи за базою даних у додатках використовують об'єктно-орієнтовані мапінг об'єкти. Для розробки представленої програми буде використовуватись Entity Framework [54]. Це фреймворк для роботи з базами даних в додатках .NET. Завдяки йому робота з базою даних не потребує прямого написання SQL запитів, він робить роботу з базою більш об’єктно-орієнтованою. В Entity Framework представлені основні 6 концепцій, такі як DbContext, Entity, DbSet, LINQ to Entities, Code First, Database First, Model First, та міграції. DbContext це основний клас, який використовується для роботи з базою даних. Він являє контекст бази даних, в якому є набір об’єктів, також він дозволяє виконувати операції над ними, такі як додавання, оновлення та видалення. Entity це таблиці баз даних інтерпретовані у класи. Кожен об’єкт даного класу являється одним рядком в таблиці. DbSet це властивість, яка покаує Entity Framework, які класи відображаються в таблиці даних. Кожен DbSet являє собою колекцію об'єктів, які можна запитувати, додавати, оновлювати та видаляти. LINQ to Entities фреймворк підтримує мову LINQ (Language Integrated Query), за допомогою якої можна писати SQL запити безпосередньо в коді на мові C#. Code First, Database First, Model First це три різні підходи до розробки бази даних. Code First це Entity Framework створює базу даних на основі розроблених класів. Database First створює модель базі у коді на основі вже готової бази даних. Model First надає можливість за допомогою візуального редактора створити моделі. Міграції – Entity Framework дозволяє керувати структурою бази даних за допомогою міграцій. Присутня можливість створювати міграції для змін в структурі бази даних і застосовувати їх до бази даних, зберігаючи даних, які вже є в ній.

LINQ (Language Integrated Query) [55] - це набір функцій, що входять в .NET Framework, який дозволяє виконувати структуровані запити до даних безпосередньо в мовах програмування, таких як C# або Visual Basic. LINQ надає

можливість писати запити до різноманітних джерел даних, таких як колекції об'єктів, бази даних, XML документи тощо, використовуючи однорівневий синтаксис. У використані LINQ є три основні переваги:

1. Декларативний підхід: LINQ дозволяє писати запити в декларативному стилі, описуючи, що потрібно зробити, а не як саме це робити.
2. Статична перевірка типів: Код LINQ перевіряється на стадії компіляції, що допомагає уникнути багатьох помилок під час виконання програми.
3. Універсальність: LINQ можна використовувати з різними джерелами даних, що робить його універсальним і потужним інструментом для обробки даних.

В LINQ є чотири основні компоненти – це LINQ to Objects, що дозволяє виконувати запити до колекцій об’єктів, LINQ to SQL, що надає можливість робити запити до баз даних на основі SQL, LINQ to XML, дозволяє робити запити до XML документів, LINQ to Entities, що призначений для роботи з ORM (Object-Relational Mapping) Entity Framework, він надає доступ до даних баз даних через об'єктно-орієнтований інтерфейс.

Архітектурний шаблон MVP чудово підходить для реалізації поставленої задачі. У Windows Forms модель може бути набором класів, що відповідають за роботу з даними, базами даних та файлами. Вигляд відображає дані користувачу і реагує на дії користувача. У випадку Windows Forms це може бути форма. Пред'явник буде містити бізнес-логіку, яка керує взаємодією між моделлю та виглядом. Він використовує модель для отримання або збереження даних та оновлення вигляду відповідно до дій користувача. Пред'явник також буде включати в себе логіку валідації даних та обробки подій, що стосуються вигляду.

Висновок до розділу 1

Було розглянуто значення індивідуалізації у автомобільній галузі. Наведені приклади сервісів та програм для конфігурації власного автомобіля. Зазначені переваги використання цих сервісів та програм. Були наведені види спеціалізованих сервісів та прогам для конфігурації власного автомобіля.

Проведений аналіз існуючих конфігураторів. Були описані популярні архітектурні шаблони, обрана мова та шаблон для розробки програми з обґрунтуванням вибору.

РОЗДІЛ 2. Предметна область та вимоги

У даному розділі описується предметна область роботи та основні вимоги до реалізації програми.

2.1 Застосування архітектурного шаблону в контексті предметної області

Предметна область являє собою розробку засобу індивідуальної конфігурації авто. Вона включає в себе програмне забезпечення для конфігурації автомобіля, інтеграцію з базою даних автомобілів, де зберігається інформація про доступні моделі автомобілів, їх характеристики, вартість і доступні опції, використання обраного шаблону MVP, інтерфейс користувача, безпеку даних та транзакцій.

Застосувати шаблон MVP при розробці можна за допомогою його основних принципів: модель, вигляд і пред’явник. Модель в цьому випадку відповідає за представлення даних про автомобілі, їх характеристики, вартість, доступні опції. Вона може включати також дані користувачів, їхні замовлення та іншу інформацію, що використовується в програмі. Модель взаємодіє з базою даних і забезпечує доступ до потрібних даних. Вигляд відповідають за відображення інтерфейсу користувача. У випадку цієї програми, вигляд показуватиме користувачеві доступні марки, моделі та опції конфігурації автомобіля, його вигляд, вартість. Вигляди не містять логіки додатку, вони просто відображають інформацію та передає користувачеві взаємодію з програмою. Пред’явник відповідає за обробку дій користувача та взаємодію між моделлю та виглядом. Він отримує дані з моделі, форматує їх і передає вигляду для відображення. Також він слідкує за діями користувача і відправляє відповідні запити до моделі для оновлення даних.

2.2 Процес конфігурації автомобіля в додатку

Процес конфігурації автомобіля, перед його купівлею, є дуже кропітким та має багато кроків.

Перший крок – це вибір марки автомобіля, який бажає купити клієнт.

Другий крок – це вибір бажаної моделі автомобіля з доступного асортименту.

Третій крок – це вибір основної комплектації, на цьому кроці клієнт може обрати базову комплектацію, яка включає в себе основні характеристики та опції для даної моделі.

Четвертий крок – це вибір двигуна та трансмісії. Клієнт обирає тип двигуна, наприклад бензиновий, дизельний, гібридний тощо, та тип трансмісії, наприклад механічна, автоматична, роботизована.

П’ятий крок – це вибір кольору кузова. Клієнт може обрати бажаний для нього колір кузова автомобіля з доступного спектру кольорів.

Шостий крок – це вибір опцій та пакетів обладнання. На цьому кроці клієнт може обрати додаткові опції та пакети обладнання, які підвищують функціональність автомобіля або забезпечують певний рівень комфорту і безпеки. Це можуть бути такі опції, як система навігації, панорамний дах, підігрів сидінь, системи безпеки тощо.

Сьомий крок – це розрахунок ціни. Після вибору всіх необхідних опцій програма може розрахувати загальну ціну автомобіля, враховуючи базову вартість, вибрані опції, податки та інші витрати.

Восьмий крок – це перевірка термінів доставки. Клієнт може перевірити термін доставки обраного автомобіля.

Дев’ятий крок – це підтвердження замовлення. Після остаточного вибору конфігурації клієнт може підтвердити своє замовлення, ввести необхідні дані для оформлення угоди та оплати.

Десятий крок – це статус замовлення. Після підтвердження замовлення клієнт може відстежувати статус свого замовлення через програму, отримувати повідомлення про оновлення та очікувану дату отримання автомобіля.

2.3 Функціональні та нефункціональні вимоги до програми

Функціональні вимоги визначають, як саме програма має поводитися та які функції вона повинна виконувати. Наприклад, це може бути можливість

реєстрації користувачів, вхід у систему, відображення списку товарів, обробка замовлень тощо.

Нефункціональні вимоги, натомість, визначають якості, що повинні бути властивими програмі. Це може бути продуктивність програми, її ефективність, безпека, доступність, масштабованість, зручність інтерфейсу користувача та інше.

Приклади вимог:

1. Функціональна вимога: Система повинна дозволяти користувачам реєструватися та авторизуватися.
2. Нефункціональна вимога: Система повинна забезпечувати швидкий час відгуку на запити користувачів, відображати сторінки без затримок більше ніж 3 секунди.

Ці два типи вимог спільно визначають поведінку та якість програми, що допомагає забезпечити задоволення потреб користувачів та досягнення бізнес-цілей.

Функціональні вимоги до програми для індивідуальної конфігурації автомобіля, що купується через мережу автосалонів з використанням сучасних архітектурних шаблонів, є такими:

1. На першій сторінці повинні бути показані всі доступні марки автомобілів, які можна заказати.
2. Після вибору марки має бути представлений список моделей автомобілів даної марки. Користувач повинен мати можливість переглянути всі доступні моделі автомобілів від виробника. Кожна модель має мати свою власну сторінку з докладною інформацією про технічні характеристики та фотографії.
3. Після вибору моделі йде вибір комплектації. Користувач повинен мати змогу обрати базову комплектацію, яка включає в себе стандартні функції та опції. Доступні комплектації будуть представлені у списку та будуть відрізнятися за рівнем обладнання, технічними характеристиками і ціною.

4. Після вибору комплектації користувач має обрати тип двигуна та трансмісію, яка може бути в даній комплектації. Користувач може обрати тип двигуна, наприклад бензиновий, дизельний, електричний або гібридний та тип трансмісії, наприклад автоматичний, механічний, CVT тощо.
5. Після вибору двигуна та трансмісії користувач повинен обрати колір кузов з представленої палітри. В програмі має бути доступний вибір кольорів кузова для кожної моделі. Користувач може переглянути кожен колір на зразках, щоб побачити, як він виглядатиме на автомобілі.
6. Вибір опцій та пакетів обладнання. Після вибору кольору користувач повинен мати змогу обрати опції та пакети обладнання для автомобіля. Користувач може додатково налаштувати свій автомобіль, вибравши різноманітні додаткові опції та пакети обладнання, які відповідають його потребам і бюджету. Це може включати в себе такі опції, як система безпеки, комфорт, розваги, дизайн кузова або дисків тощо.
7. Коли користувач остаточно визначився з вибором всіх розширень він може натиснути кнопку підтвердити, якщо ж він хоче щось змінити, він повинен мати змогу повернутись на декілька кроків назад. Якщо деякий компонентів під час вибору немає в наявності вони будуть недоступні до вибору.
8. Після підтвердження вибору йде розрахунок ціни. Програма автоматично розраховує ціну автомобіля, враховуючи базову вартість, вибрані опції, податки та інші збори. Користувач може переглянути розгорнутий розрахунок ціни, який включає в себе всі витрати.
9. Далі йде перевірка термінів доставки. Програма повинна вказати терміни доставки та можливі варіанти доставки.
10. Підтвердження замовлення. Після остаточного вибору конфігурації користувач може підтвердити своє замовлення. Програма може

запросити від користувача необхідні дані для оформлення угоди та оплати.

11. Статус замовлення. Після підтвердження замовлення користувач може відстежувати статус свого замовлення за допомогою повідомлень на електрону пошту. Він може отримувати повідомлення про будь-які оновлення щодо статусу замовлення та очікувану дату отримання автомобіля.

Нефункціональні вимоги до програми для індивідуальної конфігурації автомобіля, що купується через мережу автосалонів з використанням сучасних архітектурних шаблонів, є такими:

1. Швидкодія та продуктивність програми. Програма повинна надавати швидкий відгук на запити користувача. Сторінки з великою кількістю, наприклад список доступних моделей автомобілів, повинні завантажуватись швидко, принаймні за 3 секунди.
2. Надійність та безпека. База даних повинна бути надійна та стабільна, щоб уникнути втрати даних у процесі оформлення замовлення.
3. Безпека. Забезпечення конфіденційності та цілісності даних користувачів, а також захист від несанкціонованого доступу до особистої інформації та фінансових даних при оплаті.
4. Доступність та сумісність. Програма повинна бути доступною на різних пристроях із різними роздільними здібностями.
5. Зручність користування. Інтерфейс користувача повинен бути інтуїтивно зрозумілим та легким у використанні. Важливо, щоб користувачі могли легко знаходити необхідну інформацію та виконувати потрібні дії без зайвих зусиль.
6. Масштабованість. Програма повинна бути готовою до масштабування, щоб впоратися зі зростанням обсягу користувачів та додаткових функцій в майбутньому.

7. Доступність даних про наявність. Програма повинна постійно оновлювати дані про доступність моделей автомобілів та комплектацій, щоб користувачі могли бачити тільки актуальну інформацію.
8. Доступність підтримки та обслуговування. Програма повинна мати підтримку для користувачів, які мають запитання чи проблеми під час процесу вибору та оформлення замовлення.
9. Резервне копіювання та відновлення. База даних повинна мати механізми резервного копіювання та відновлення даних, щоб уникнути втрати інформації у випадку технічних або людських помилок.
10. Локалізація. Програма повинна бути готовою до роботи у різних регіонах світу та підтримувати різні мови та культурні особливості користувачів.

Висновки до розділу 2

Була описана предметна область роботи. Були визначені функціональні та нефункціональні вимоги до програми.

РОЗДІЛ 3. Проектування та реалізація програми

У даному розділі описується основна архітектура та структура програми. Реалізація основних модулів та інтерфейс користувача.

3.1 Архітектура програми

Побудова архітектури програми починається з аналізу вимог до програми та визначення її функціональних та нефункціональних характеристик. Ці характеристики описані у другому розділі. Далі йде вибір основних технологій для кращої реалізації програми. Ці технології повністю описані та були обрані у першому розділі.

Розробка бази даних для програми. База даних має десять таблиць, які описують основні компоненти автомобіля. Перша таблиця це CarBrand(бренд автомобіля), в ній є два поля IdCarBrand та Name – це унікальний ідентифікатор, який є первинним ключем таблиці, та назва бренду автомобіля відповідно. Таблиця CarBrand представлена на рис. 3.1.

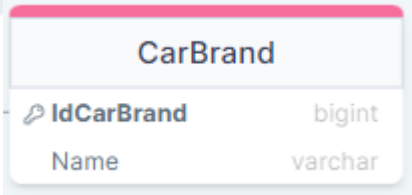

Рисунок 3.1 Таблиця CarBrand

Наступна таблиця CarModel(модель автомобіля), в ній знаходяться 4 поля, а саме IdCarModel, IdCarBrand, Name, Price. IdCarModel – це унікальний ідентифікатор для таблиці, який є первинним ключем таблиці. IdCarBrand – це зовнішній ключ, який зв’язує таблиці CarModel та CarBrand, який визначає бренд автомобіля, якому належить відповідна модель авто, тип зв’язку один до багатьох. Name – це поля яке зберігає в собі назву моделі автомобіля. Price – це поле яке зберігає базову ціну автомобіля даної моделі. Таблиця CarModel

представлена на рис. 3.2. Зв’язок між таблицями CarModel та CarBrand представлені на рис. 3.3.

Наступна таблиця EngineType(тип двигуна), в ній знаходяться два поля, унікальний ідентифікатор та первинний ключ IdEngineType та тип двигуна Type, який описує до якого типу може належати двигун, наприклад бензиновий, дизельний, електричний та гібридний. Таблиця EngineType представлена на рис. 3.4.

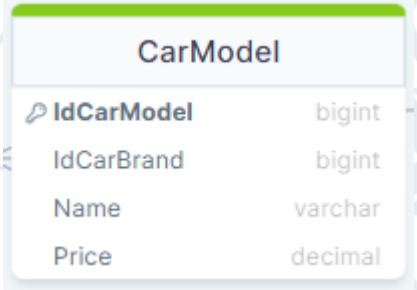

Рисунок 3.2 Таблиця CarModel

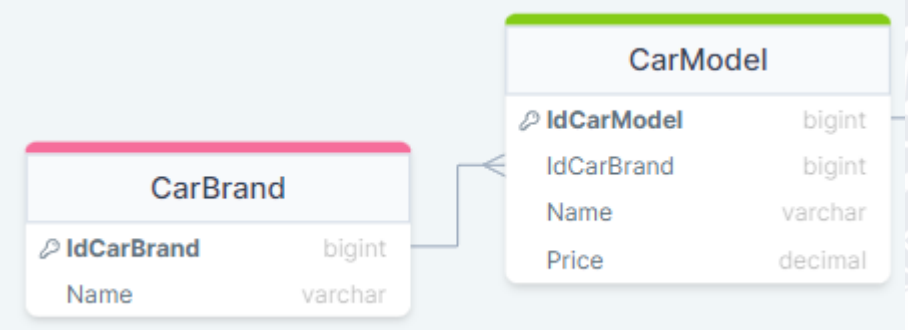

Рисунок 3.3 Зв’язок між таблицями CarBrand та CarModel

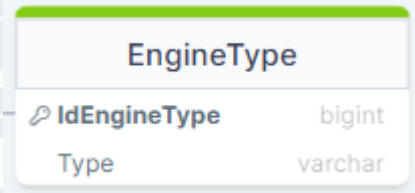

Рисунок 3.4 Таблиця EngineType

Таблиця Engine(Двигун), в ній знаходяться шість полів. IdEngine – унікальний ідентифікатор та первинний ключ. IdEngineType – зовнішній ключ, який зв’язаний з таблицею EngineType та вказує до якого типу належить двигун,

тип зв’язку один до багатьох. Name – поле для збереження назви двигуна. Volume – поле для збереження об’єму двигуна. Power – поле для збереження потужності двигуна. Price – поле для збереження ціни двигуна. Таблиця Engine представлена на рис.3.5. Зв’язок між таблицями EngineType та Engine представлений на рис. 3.6.

Таблиця TransmissionType (тип трансмісії), в ній знаходяться два поля. IdTransmissionType – унікальний ідентифікатор та первинний ключ таблиця. Type – тип трансмісії, який зберігає можливі типи трансмісії, наприклад механічна, автоматична, варіатор, подвійне щеплення та інші. Таблиця TransmissionType представлена на рис. 3.7.

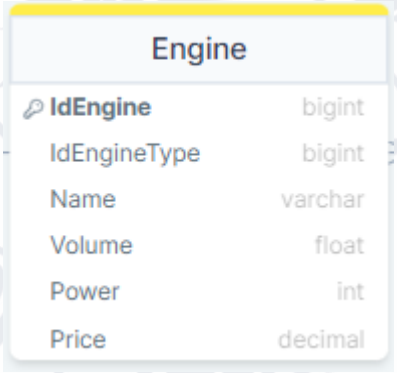

Рисунок 3.5 Таблиця Engine

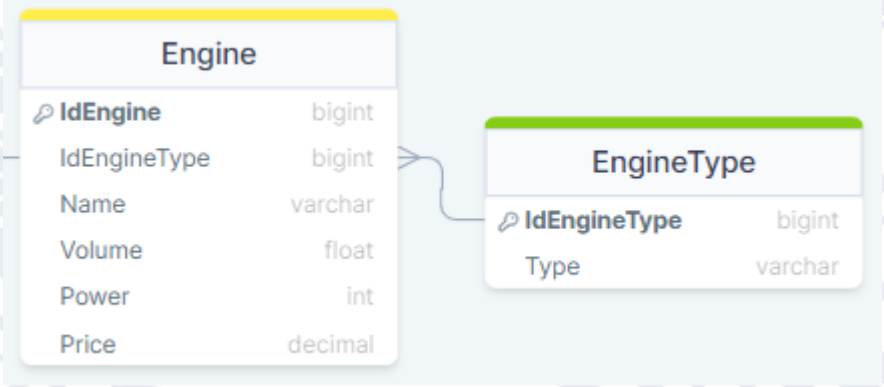

Рисунок 3.6 Зв’язок між таблицями Engine та EngineType

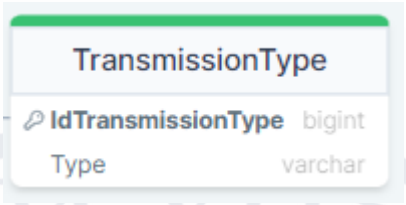

Рисунок 3.7 Таблиця TransmissionType

Таблиця Transmission(трансмісія), в ній знаходяться п’ять полів. IdTransmission – унікальний ідентифікатор та первинний ключ. IdTransmissionType – зовнішній ключ, який вказує до якого типу належить дана трансмісія, тип зв’язку один до багатьох. Name – назва трансмісії. Gears – кількість передач. Price – ціна даної трансмісії. Таблиця Transmission представлена на рис. 3.8. Зв’язок між таблицями Transmission та TransmissionType представлений на рис. 3.9.

Таблиця Colour (колір), в ній є три поля. IdColour – унікальний ідентифікатор та первинний ключ. Name – поле для збереження назва можливого кольору авто, наприклад білий, чорний, синій та інші. Price – ціна кольору. Таблиця Colour представлена на рис. 3.10.

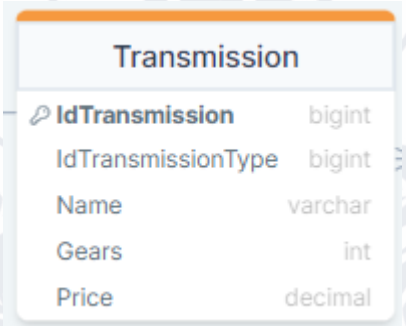

Рисунок 3.8 Таблиця Transmission

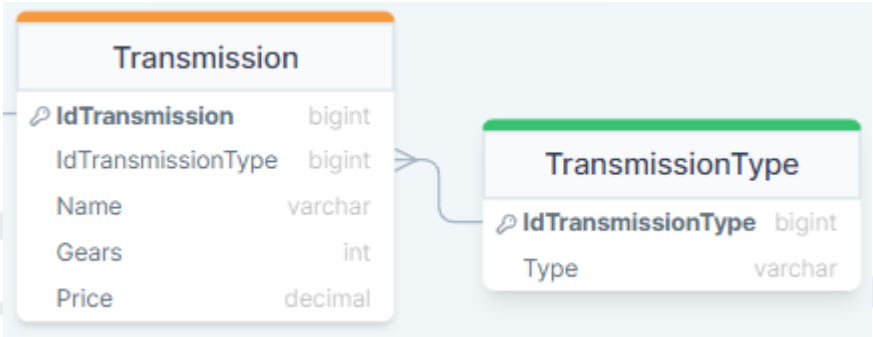

Рисунок 3.9 Зв’язок між таблицями Transmission та TransmissionType

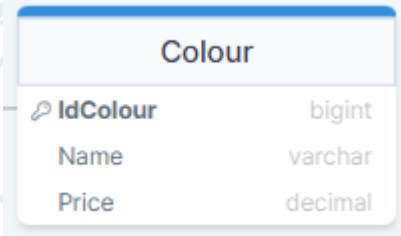

Рисунок 3.10 Таблиця Colour

Таблиця Wheel(колесо), в ній є три поля. IdWheel – унікальний ідентифікатор та первинний ключ. Diameter – поле для збереження діаметру колісного диску, наприклад 19, 20, 21 та інші. Price – ціна даного колеса. Таблиця Wheel представлена на рис. 3.11.

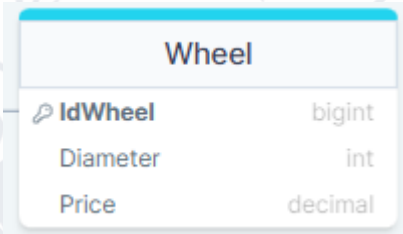

Рисунок 3.11 Таблиця Wheel

Таблиця BodyKit(кузовний обвіс), в ній є три поля. IdBodyKit – унікальний ідентифікатор та первинний ключ. Type – поле для збереження типу обвісу, наприклад стандарт, широкий, повний, бампер, карбон та інші. Price – ціна за відповідний тип обвісу. Таблиця BodyKit представлена на рис. 3.12.

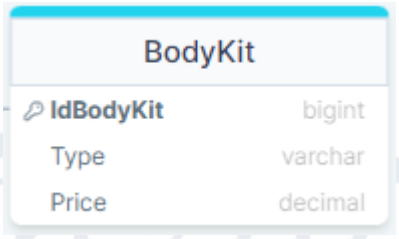

Рисунок 3.12 Таблиця BodyKit

Основна таблиця Car (автомобіль), вона зберігає цілий автомобіль з відповідними можливими модифікаціями. В ній є сім полів. IdCar – унікальний ідентифікатор та первинний ключ. IdCarModel – зовнішній ключ, який вказує модель автомобіля, тип зв'язку один до багатьох. IdEngine – зовнішній ключ, який вказує на двигун автомобіля, тип зв'язку один до багатьох. IdTransmission – зовнішній ключ, який вказує на трансмісію автомобіля, тип зв'язку один до багатьох. Colours – поле для збереження списку Id можливих кольорів для даного автомобіля. Wheels – поле для збереження списку Id можливих коліс для даного автомобіля. BodyKits – поле для збереження списку Id можливих обвісів для даного автомобіля. Таблиця Car представлена на рис. 3.13. Повна структура бази даних та зв'язків між таблицями представлені на рис. 3.14.

Наступний параметр для опису архітектури програми – це компоненти програми. Програма має бути розбита на окремі папки з модулями, які виконують свої певні функції. Використовуючи шаблон MVP мають бути папки з назвами Models, Presenters та Views. Також можна додати папки для збереження зображень і тому подібному. Для збереження міграцій бази даної моє бути папка Migrations.

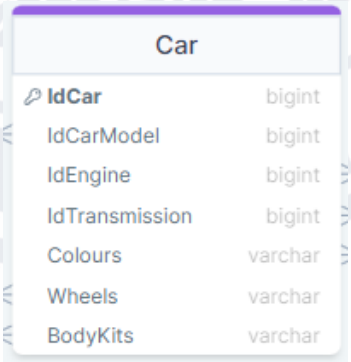

Рисунок 3.13 Таблиця Car

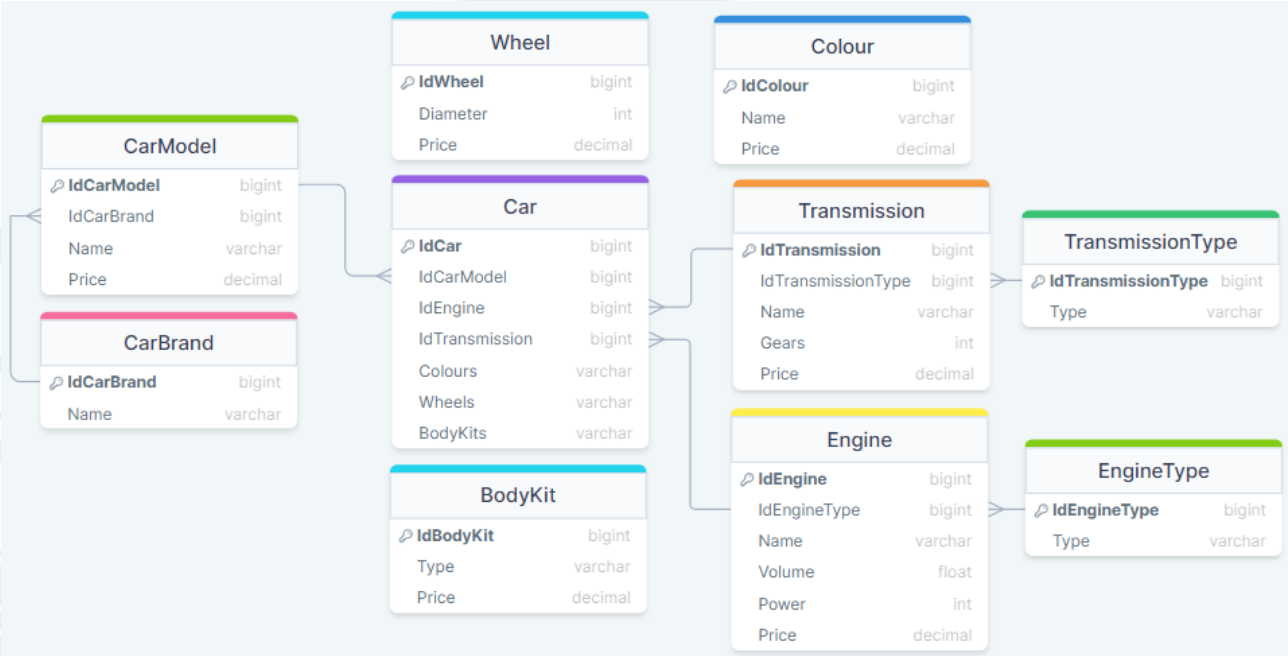

Рисунок 3.14 Повна структура бази даних та зв’язків між таблицями

3.2 Структура програми

Програма складається з модулів, які потрібні для реалізації архітектурного шаблону MVP. Це модулі моделей, пред’явників та виглядів. Модулі моделей це представлення таблиць бази даних у виді класів. Використовуючи EntityFramework є три різні варіанти створення цих моделей. ModelFirst – спочатку дозволяє створити нову модель за допомогою конструктора Entity Framework, а потім створити схему бази даних із моделей. CodeFirst – спочатку пишеться код таблиць, міграцій та контексту, потім за допомогою менеджеру пакетів NuGet та пакету EntityFramework створюється база даних на основі цих файлів. DatabaseFirst автоматично за допомогою менеджеру пакетів NuGet “EntityFramework”, ”EntityFramework.Core”, “EntityFramework.Core.Design”, “EntityFramework.Core.SqlServer”, “EntityFramework.Core.Tools”. Створюються файли моделей таблиць, файл контексту та файл міграції. Був обраний спосіб DatabaseFirst, тому що він є найлегший у реалізації.

Для створення потрібно ввести відповідну команду у консоль менеджера пакетів. Це команда “Scaffold-DbContext” після цього потрібно ввести параметри для підключення до бази, шаблон такий “"Data Source=“посилання на ядро бази”;Initial Catalog=“назва бази даних”; Trusted_Connection=“True/False”; TrustServerCertificate=“True/False ”;"”. Data Source – це шлях до ядра бази даних, Initial Catalog – назва бази даних з якою буде взаємодія, Trusted_Connection – True, якщо підключення до бази є довіреним, або False, якщо підключення до бази не є довіреним, у цьому випадку взаємодія не відбудеться, TrustServerCertificate – True, якщо сертифікат сервера є довіреним, або False, якщо сертифікат сервера не є довіреним, у цьому випадку взаємодія не відбудеться. Файли моделей та контексту представлені на рис. 3.15. Файли міграцій представлені на рис. 3.16.

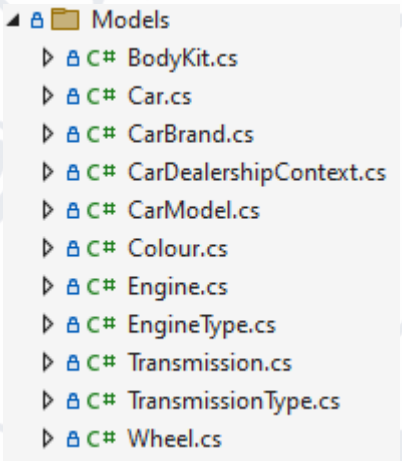

Рисунок 3.15 Файли моделей та контексту

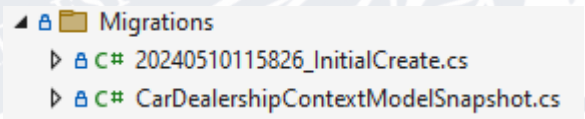

Рисунок 3.16 Файли створених міграцій

Наступний крок – це створення виглядів для показу інтерфейсу користувачеві. При написанні програми на C# WindowsForms ці вигляди

створюються на основі форм, тобто вікон, які будуть відображатись користувачеві. Файли виглядів представлені на рис. 3.17.

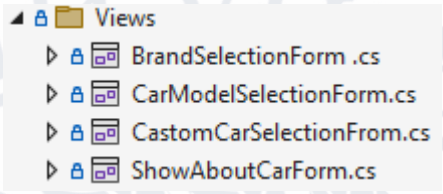

Рисунок 3.17 Файли створених виглядів

Наступний крок – це створення файлів пред'явників, які будуть об'єднувати моделі та вигляди між собою. Файли пред'явників представлені на рис. 3.18.

Останній крок – це папка Images, яке призначена для зберігання файлів фотографій автомобілів. Вони будуть використовуватись для показу користувачеві ви обрані автомобіля та його кольору. Файли фотографій автомобілів представлені на рис. 3.19.

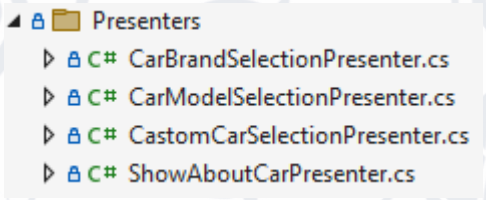

Рисунок 3.18 Файли створених пред'явників

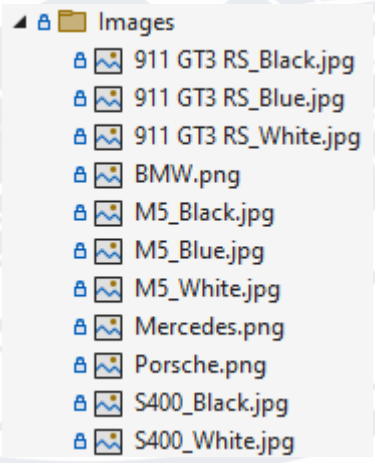

Рисунок 3.19 Файли фотографій автомобілів

3.3 Реалізація основних модулів

Всі модулі шаблону MVP створюються за одним шаблоном з додаванням індивідуального функціоналу для окремих випадків.

Моделі для реалізації даної програми створюються за таким шаблоном:

1. Створюється клас таблиці в базі даних
2. Створюються змінні для кожного поля таблиці
3. Якщо є зв'язки з іншими таблицями створюються віртуальні колекції для навігації по таблиці
4. Якщо є зовнішні ключи то створюються віртуальні змінні, які являють собою об'єкт класу який є зовнішнім ключем

Приклад коду для реалізації класу CarModel:

```
[Table("CarModel")]
public partial class CarModel
{
    [Key]
    public long IdCarModel { get; set; }
    public long IdCarBrand { get; set; }
    [StringLength(50)]
    public string? Name { get; set; }
    [Column(TypeName = "decimal(18, 0)")]
    public decimal? Price { get; set; }
    [InverseProperty("IdCarModelNavigation")]
    public virtual ICollection<Car> Cars { get; set; } = new
List<Car>();
    [ForeignKey("IdCarBrand")]
    [InverseProperty("CarModels")]
    public virtual CarBrand IdCarBrandNavigation { get; set; } =
null!;
}
```

У першому ряду написаний атрибут який показує до якої таблиці належить даний клас. У другому рядку написані права доступу до класу та його назва. Далі атрибут який вказує на зміну класу яка є первинним ключем, рядком і його

довжина, колонка типу даних, зовнішній ключ, поле для навігації та інші потрібні атрибути. Потім під кожним атрибутом оголошується відповідна зміна. За даним прикладом створюються всі моделі таблиць з бази даних

Пред'явники для реалізації даної програми створюються за таким шаблоном:

1. Створюється клас конкретного пред'явника
2. Створюється змінна інтерфейс вигляду яка доступна лише для читання
3. В конструктор класу пред'явника передається інтерфейс вигляду за який відповідає даний пред'явник. В конструкторі в змінну створену на попередньому кроці записується відповідний вигляд
4. Реалізується функція для передачі у вигляд інформації яка має бути представлена користувачеві
5. Реалізується функція яка реагує на дії користувача та обробляє

Приклад коду для реалізації пред'явника CarBrandSelectionPresenter:

```
public class CarBrandSelectionPresenter
{
    private readonly IBrandSelectionView _view;
    public CarBrandSelectionPresenter(IBrandSelectionView view)
    {
        _view = view;
        LoadBrands();
    }
    private void LoadBrands()
    {
        using (CarDealershipContext db = new
CarDealershipContext())
        {
            var brands = db.CarBrands.OrderBy(x =>
x.IdCarBrand).ToList();
            _view.ShowBrands(brands);
        }
    }
    public void BrandSelected(CarBrand selectedBrend)
    {
```

```
        CarModelSelectionForm cmsf = new
CarModelSelectionForm(selectedBrend);
        cmsf.Show();
    }
}
```

Перший рядок це створення відповідного класу з потрібним рівнем доступу до нього. Далі змінна інтерфейсу вигляду. Потім конструктор з потрібними параметрами. В конструкторі записується в зміну інтерфейсу вигляду вигляд яким буде керувати даний пред'явник. Далі йде виклик функції для передачі даних вигляду. У цій функції йде обробка даних з моделі та передача у вигляд.

Наступна функція відповідає за обробку запиту користувача. У цьому випадку йде перехід на наступний вигляд.

За таким принципом створені всі пред'явники з мінімальними особливими відмінностями.

Вигляд для реалізації даної програми створюється за таким шаблоном:

1. Створення форми тобто вікна програми
2. Створення інтерфейсу який відповідає даному вигляду та написання в інтерфейсі функцій які повинні бути реалізовані у коді.
3. Створення класу форми з потрібним рівнем доступу, який успадковується від системного класу форми та інтерфейсу який був описаний у минулому пункті
4. Створюється змінна пред'явника для взаємодії з ним
5. Створюються додаткові потрібні змінні
6. Оголошується конструктор даного класу в якому ініціалізуються форма, тобто вікно програми, та пред'явник в який передається вигляд.
7. Реалізуються функції які потребує унаслідуваний інтерфейс
8. Реалізується інші потрібні функції

Приклад коду для реалізації коду вигляду BrandSelectionForm:

```
public interface IBrandSelectionView
{
    void ShowBrands(List<CarBrand> brands);
```

```
}
public partial class BrandSelectionForm : Form,
IBrandSelectionView
{
    private CarBrandSelectionPresenter _presenter;
    public List<CarBrand> _brands = new List<CarBrand>();
    public BrandSelectionForm()
    {
        InitializeComponent();
        _presenter = new CarBrandSelectionPresenter(this);
    }
    public void ShowBrands(List<CarBrand> brands)
    {
        _brands = brands;
    }
    private void pictureBox1_Click(object sender, EventArgs e)
    {
        _presenter.BrandSelected(_brands[0]);
    }
    private void pictureBox2_Click(object sender, EventArgs e)
    {
        _presenter.BrandSelected(_brands[1]);
    }
    private void pictureBox3_Click(object sender, EventArgs e)
    {
        _presenter.BrandSelected(_brands[2]);
    }
}
```

Спочатку створюється інтерфейс IBrandSelectionView з публічним рівнем доступу, щоб пред’явник міг ним користуватись. В ньому описаний метод для демонстрації інформації про бренди для користувача. Далі створений клас BrandSelectionForm, який унаслідує Form та IBrandSelectionView. Створюється змінна пред’явника. Створюється змінна типу список для зберігання списку доступних брендів авто. Створюється конструктор який ініціалізує вікно та

пред’явник в який передається дане вікно. Потім йде реалізація функції, яку потребує інтерфейс IBrandSelectionView, для прийому інформації від пред’явника. Потім йде обробка взаємодії з елементами вікна, а саме вибору бренду. В представлених функціях викликається функція пред’явника для обробки вибору.

За таким принципом створені всі вигляди з мінімальними особливими відмінностями.

Для отримання даних з бази використовується оператор “Using”, він забезпечує правильне використання віддалених об'єктів. Даний оператор гарантує виклик екземпляра інтерфейсу IDisposable, навіть якщо у блоку Using було викликано виключення. IDisposable – надає механізм для звільнення пам’яті некерованих ресурсів.

Приклад коду з використанням оператора Using:

```
private void LoadBrands()
{
    using (CarDealershipContext db = new CarDealershipContext())
    {
        var brands = db.CarBrands.OrderBy(x =>
x.IdCarBrand).ToList();
        _view.ShowBrands(brands);
    }
}
```

Функція LoadBrands, що використовується для формування даних та її відправки на вигляд з пред’явника. Створюється екземпляр класу CarDealershipContext для роботи з даними у базі. Отримується список представлених брендів і записуються у зміну brands. Потім за допомогою функції ShowBrands вони відправляються на вигляд.

3.4 Інтерфейс користувача

Розроблений інтерфейс користувача є мінімалістичним та потребує доопрацювання. Завдяки правильно структурованого коду, який можна

масштабувати та легко підтримувати, це можна буде зробити з легкістю та без застосування великих зусиль.

Першим вікном, яке бачить користувач, є вікном вибору бренду автомобіля. На формі представленні наявні у базі даних бренди автомобілів, моделі яких можна замовити. Воно формується завдяки передачі на об’єкт PictureBox зображення бренду автовиробника. Щоб обрати бренд потрібно натиснути на його зображення. Пісня чого відбудеться перехід на ступну стадію вибору. Приклад вікна вибору бренду показано на рис. 3.20.

Рисунок 3.20 Вікно вибору бренду автомобіля

Друге вікно, яке бачить користувач, є вікном вибору моделі автомобіля обраного бренду. На формі представлені моделі автомобілів, моделі яких наявні в базі даних. На формі є об’єкти TableLayoutPanel та Button. Button для повернення на минуле вікно. TableLayoutPanel містить в собі PictureBox та Button, кількість цих елементів змінюється автоматично відносно кількості доступних моделей автомобіля. PictureBox для показу зображення автомобіля

відповідної моделі, Button для вибору відповідного автомобіля. Приклад вікна вибору моделі представлено на рис. 3.21.

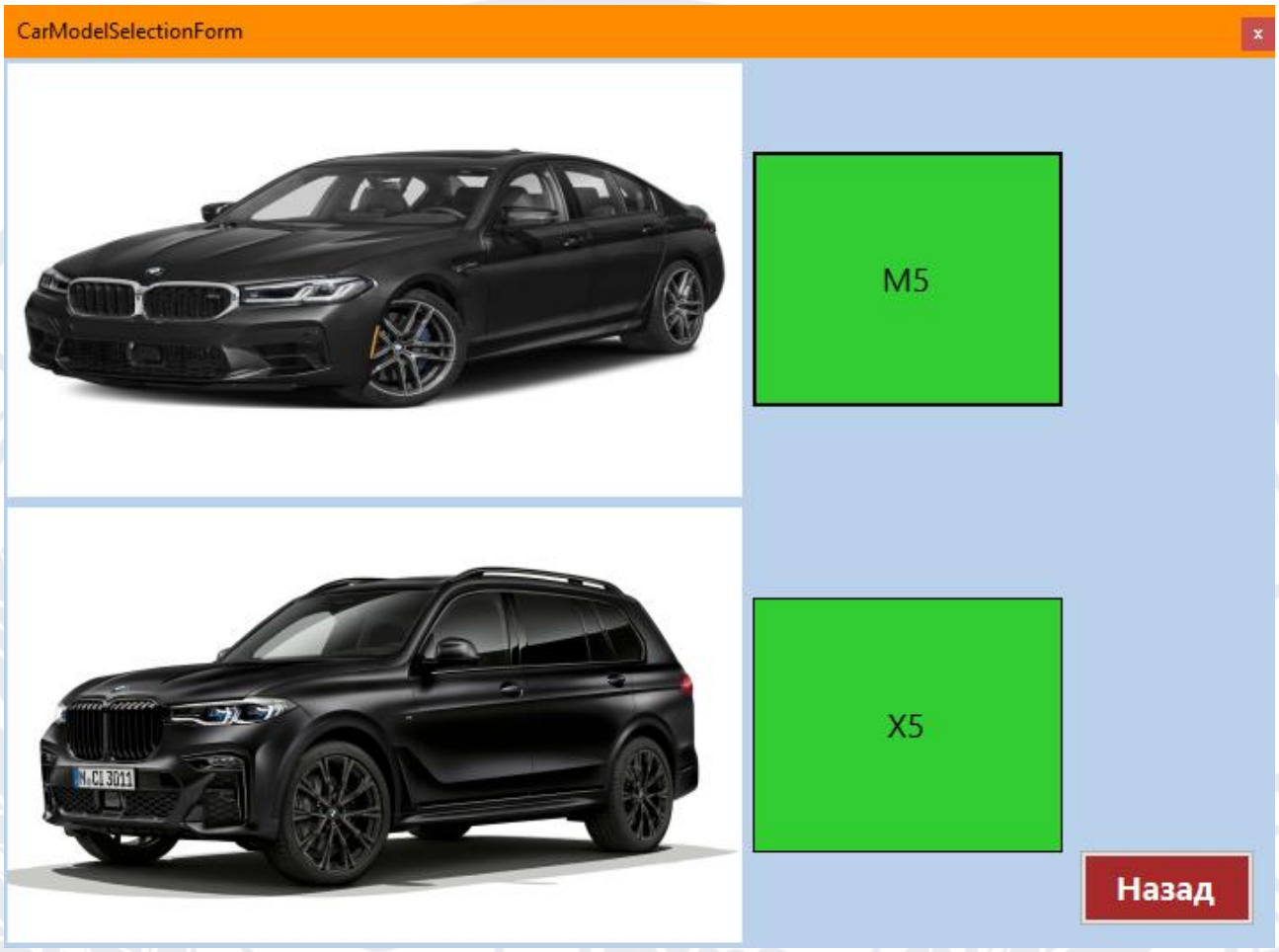

Рисунок 3.21 Вікно вибору моделі автомобіля

Третє вікно, яке бачить користувач, є вікном вибору комплектації автомобіля обраної моделі. На формі представлений список доступних комплектацій автомобіля, комплектації якого наявні в базі даних. На формі є об’єкти TableLayoutPanel та Button. Button для повернення на минуле вікно. TableLayoutPanel містить в собі PictureBox, Lable та Button. PictureBox для показу зображення автомобіля, Lable для виводу тексту з інформацією про комплектацію, Button для вибору авто відповідної комплектації. Приклад вікна вибору моделі представлено на рис. 3.22.

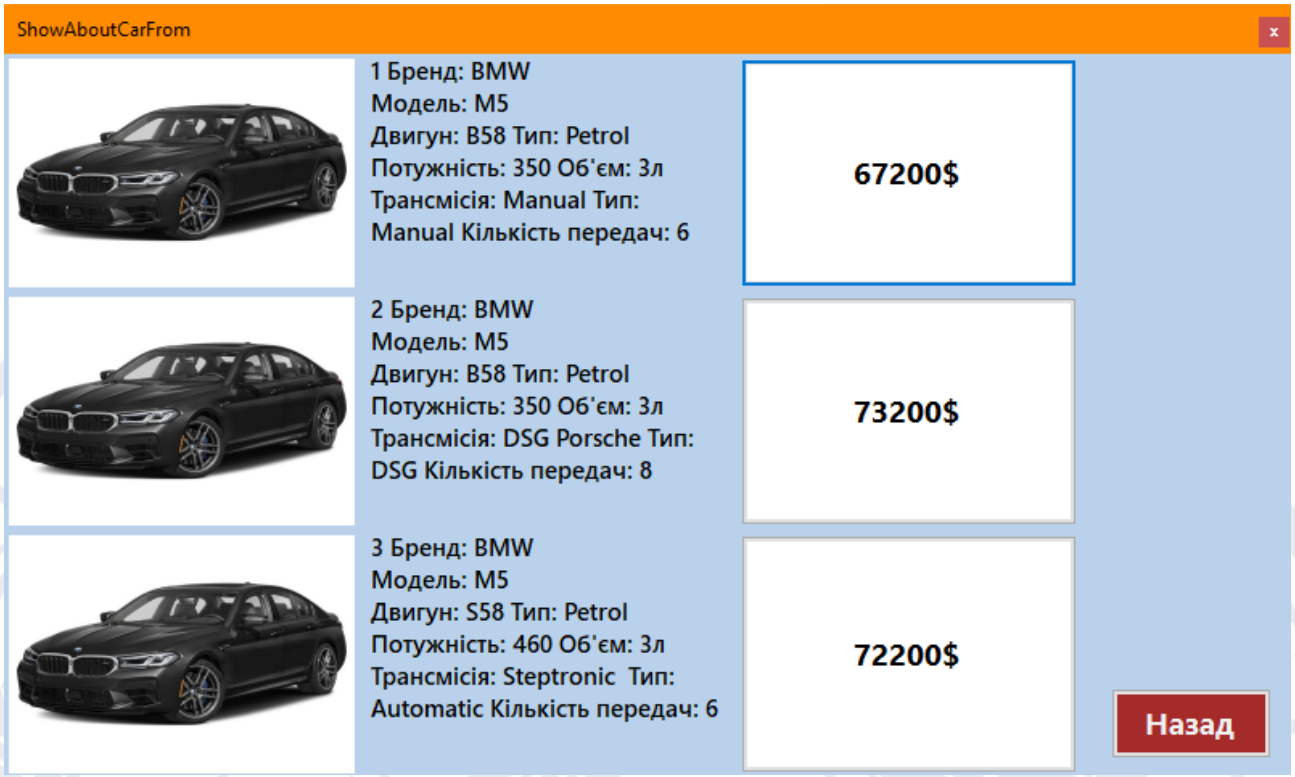

Рисунок 3.22 Вікно вибору комплектації автомобіля

Четверте вікно, яке бачить користувач, є вікном вибору модифікацій для автомобіля, які наявні в базі даних. На формі є об’єкти PictureBox, GroupBox, Lable, ImageList та Button. PictureBox для показу, як виглядає автомобіль при змінах обраних модифікацій, наприклад колір та інше. GroupBox створений для об’єднання кнопок (Button), їх два, перший для кольорів та другий для обрання дисків. Також у кожному з них є Lable який містить напис, що описує за що відповідає кожен GroupBox. Кнопки у GroupBox містять в зображення, кнопки з зображенням кольорів відповідають за зміну кольору автомобіля, а кнопки з зображенням колісних дисків відповідають за зміну колісних дисків на зображені. За відображення цих зображень на кнопках є два ImageList. ImageList – це список зображень, які можуть використовувати об’єкти на формі. Перший ImageList містить в собі зображення кольорів, які використовуються для передачі на кнопки у першому GroupBox. Другий ImageList містить в собі зображення колісних дисків, які використовуються для передачі на кнопки у другому GroupBox. Також є окремі Label та Button. Label використовується для виводу ціни при змінах під час вибору модифікацій. Button зроблений для

підтвердження замовлення після усіх бажаних змін, що зробив користувач. Список змін запам’ятовується та записується у окремий файл.

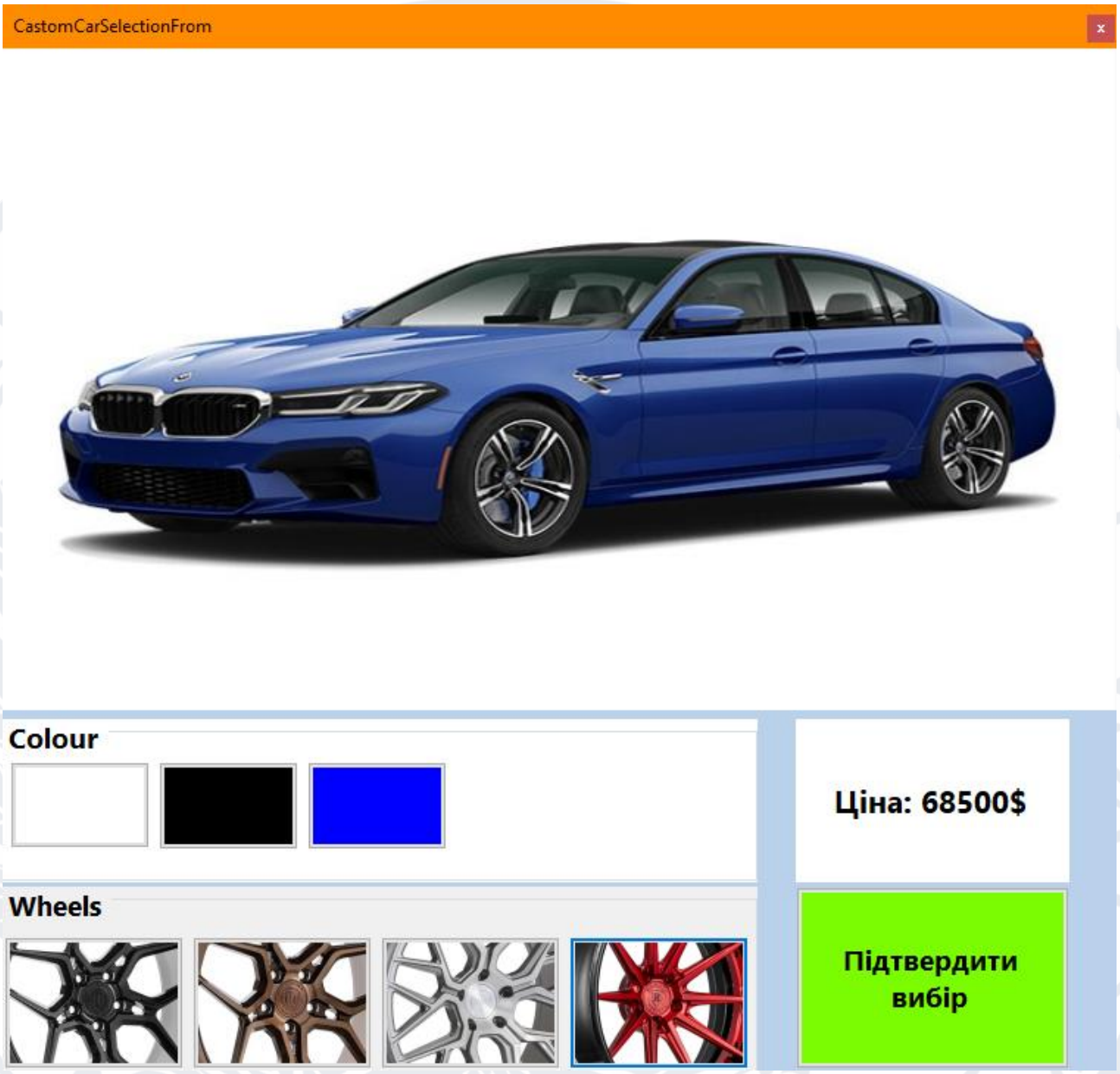

Рисунок 3.23 Вікно вибору компонентів автомобіля

Висновки до розділу 3

У даному розділі були описані основна архітектура та структура програми. Реалізація основних модулів та інтерфейс користувача. З наведеними прикладами коду, бази даних та зображеннями інтерфейсу.

РОЗДІЛ 4. Тестування та верифікація

У даному розділі будуть описані основні підходи до тестування програмного забезпечення. Буде описана реалізація тестування для створеного додатку. Буде проведена верифікація відповідності використаному архітектурному шаблону.

4.1 Підходи до тестування

Тестування програмного забезпечення має довгу історію, воно еволюціонувало разом з розвитком комп’ютерної техніки та програмування [56].

Перші тести відбувалися вручну. У 1940-1950-ті роки, коли виникли перші комп’ютери, такі як ENIAC, Mark I та інші. Тестування здійснювалось безпосередньо на апаратному рівні. Тестування відбувалось шляхом перевірки результатів виконання програм. Основною метою було впевнитися, що комп'ютер обчислює правильні результати.

У 1960-тих роках відбулися перші спроби формалізувати тестування. З’явились перші методології тестування. Це було пов’язано із зростанням складності програм.

У 1970-тих роках почалась стандартизація та розвиток методів тестування. Почали розробляти стандарти тестування, такі як IEEE 829 (Стандарт для документації тестових випадків). З’явився метод тестування під назвою “Тестування чорної скриньки”, це метод, коли тестер не знає внутрішньої структури програми.

У 1980-ті відбулось вдосконалення технік тестування. Почали розвивати методи “Тестування білої скрині”, вони, в свою чергу, вже вимагали знань внутрішньої структури програмного забезпечення. Почали з’являтись більш складні інструменти для автоматизованого тестування та інтегрованих середовищ розробки (IDE), які полегшували процес тестування.

У 1990-ті почався розвиток вже існуючих та нових методологій та інструментів для тестування. З появою та розвитком гнучких методологій для розробки програмного забезпечення, наприклад Agile, Scrum та інші, змінився

підхід до тестування. Було вирішено зосередитись на постійному інтегруванні та тестуванні. Почали впроваджувати нові системи для управління дефектами (bug tracking systems), такі як JIRA, Bugzilla, що полегшили процеси відстеження та виправлення помилок.

У 2000-ті та до зараз почали з’являтись відповідні посади), такі як DevOps та Continuous Integration/Continuous Deployment (CI/CD), для інтеграції процесів розробки та операцій розробки з постійним тестуванням і розгортанням, наприклад (Jenkins, Travis CI). Почалось широке застосування інструментів для автоматизації тестування (Selenium, Appium). Також зросла увага до безпеки та продуктивності. Тому з’явились тести безпеки та продуктивності програмного забезпечення. Почали використовувати машинне навчання та штучний інтелект для підвищення якості процесів тестування, автоматизації генерації тестів і виявлення дефектів.

Сучасне тестування програмного забезпечення є комплексним і невід'ємним елементом розробки програмного забезпечення, що включає безліч підходів, методологій та інструментів для забезпечення якості та надійності програмних продуктів.

Тестування програмного забезпечення є критично важливою частиною процесу розробки, оскільки воно допомагає забезпечити якість, надійність та безпеку програмного продукту. Існує кілька основних підходів до тестування програмного забезпечення, кожен з яких має свої особливості та сфери застосування. Основні з них це [57–59]:

1. Функціональне тестування
2. Нефункціональне тестування
3. Статичне тестування
4. Динамічне тестування
5. Чорна скриня (Black Box Testing)
6. Біла скриня (White Box Testing)
7. Регресійне тестування
8. Автоматизоване тестування

9. Ручне тестування

Функціональне тестування – це тип тестування програмного забезпечення, спрямований на перевірку відповідності функцій програми вимогам та специфікаціям. Мета функціонального тестування полягає у перевірці того, що програмне забезпечення виконує всі свої заплановані функції правильно і без помилок. Функціональне тестування має декілька видів, такі як модульне тестування (Unit Testing), інтеграційне тестування (Integration Testing), системне тестування (System Testing) та приймальне тестування (Acceptance Testing).

Модульне тестування фокусується на перевірці окремих компонентів або модулів коду. Це може бути окрема функція або метод. Ціль цього тестування – це виявлення помилок у найменших частинах коду. Це тестування виконує зазвичай розробник.

Інтеграційне тестування перевіряє взаємодію між окремими модулями або компонентами після їх об'єднання. Ціль – це виявлення проблем у взаємодії між модулями. Цим тестуванням замається тестувальники або розробники.

Системне тестування перевіряє всю систему в цілому для забезпечення правильного функціонування всіх компонентів разом. Його ціль – це перевірка всієї системи на відповідність вимогам. Виконує тестування його кінцеві користувачі або тестувальники.

Приймальне тестування перевіряє, чи відповідає програма вимогам кінцевих користувачів та чи готова вона до релізу. Ціль цього тестування полягає у переконані, що система задовольняє бізнес-вимоги. Виконує тестування його також, як і системне тестування, кінцеві користувачі або тестувальники.

Існують декілька методів для функціонального тестування – це тестування на основі вимог (Requirements-Based Testing), тестування на основі сценаріїв (Scenario-Based Testing), тестування на основі еквівалентних класів (Equivalence Partitioning), тестування на основі граничних значень (Boundary Value Analysis) та тестування приймальних критеріїв (Acceptance Criteria Testing).

Тестування на основі вимог передбачає створення тестів на основі специфікацій та вимог до ПЗ. Ціль – забезпечити, що всі вимоги виконані. Наприклад тестування функції логіну з різними комбінаціями логіну та пароля.

Тестування на основі сценаріїв моделює реальні ситуації, з якими можуть зіткнутися користувачі. Його ціль – це перевірити функціональність системи в реальних умовах. Наприклад перевірка процесу покупки в інтернет-магазині від додавання товару до кошика до оплати.

Тестування на основі еквівалентних класів, цей метод розділяє вхідні дані на класи, які вважаються рівноцінними для тестування. Ціль – це зменшити кількість тестів, зберігаючи покриття тестами більшості коду. Наприклад для поля вводу віку перевірити діапазони 0-17, 18-64, 65+.

Тестування на основі граничних значень зосереджується на перевірці граничних значень вхідних даних. Ціль – виявлення помилок на границях вхідних діапазонів. Наприклад для поля вводу від 1 до 100 перевірити значення 0, 1, 100, 101.

Тестування приймальних критеріїв перевіряє, чи задовольняє система приймальні критерії, визначені в специфікаціях. Ціль – переконатися, що всі умови приймання виконані. Наприклад перевірка функціональності, визначеної в користувацьких історіях.

Існує багато інструментів, які допомагають автоматизувати процес функціонального тестування, такі як:

1. Selenium: Використовується для автоматизації веб-додатків.
2. JUnit: Інструмент для модульного тестування Java-додатків.
3. TestNG: Потужний фреймворк для тестування на Java.
4. Postman: Використовується для тестування API.
5. SoapUI: Інструмент для тестування веб-сервісів.

Функціональне тестування має свої переваги та недоліки. Переваги це:

1. Виявлення дефектів: Дозволяє виявляти помилки на ранніх стадіях розробки.

2. Забезпечення якості: Гарантує, що програма відповідає специфікаціям і вимогам.
3. Підвищення задоволеності користувачів: Забезпечує функціональність, яка відповідає очікуванням користувачів.

Недоліки:

1. Обмеженість покриття: Не перевіряє нефункціональні аспекти, такі як продуктивність чи безпека.
2. Витрати часу і ресурсів: Може бути трудомістким процесом, особливо для великих систем.

Функціональне тестування є незамінною частиною процесу забезпечення якості програмного забезпечення, допомагаючи переконатися, що продукт виконує всі свої функції коректно та відповідає очікуванням користувачів.

Нефункціональне тестування – це тип тестування програмного забезпечення, який зосереджується на нефункціональних аспектах системи, таких як продуктивність, надійність, безпека, юзабіліті тощо. Мета нефункціонального тестування полягає у перевірці якості системи в різних умовах експлуатації, забезпеченні її стабільності та здатності відповідати вимогам користувачів у різних сценаріях використання.

Воно має декілька основних видів:

1. Тестування продуктивності (Performance Testing)
2. Тестування надійності (Reliability Testing)
3. Тестування безпеки (Security Testing)
4. Тестування юзабіліті (Usability Testing)
5. Тестування сумісності (Compatibility Testing)
6. Тестування на зручність супроводу (Maintainability Testing)

Тестування продуктивності оцінює, наскільки швидко система реагує і наскільки ефективно працює під різними навантаженнями. Ціль – це визначення меж продуктивності системи та її поведінки при максимальному навантаженні. Методи – це стресове тестування, навантажувальне тестування, тестування масштабованості.

Тестування надійності, цей тип тестування перевіряє стабільність і надійність системи протягом тривалого періоду часу. Ціль – це переконатися, що система стабільно працює без збоїв протягом заданого часу. Методи – тестування вразливостей, тестування проникнення, тестування аутентифікації.

Тестування безпеки оцінює здатність системи захищати дані та підтримувати конфіденційність, цілісність та доступність інформації. Ціль – виявлення і усунення потенційних загроз безпеці. Методи – тестування вразливостей, тестування проникнення, тестування аутентифікації.

Тестування юзабіліті оцінює зручність використання системи, її інтерфейс та взаємодію з користувачами. Ціль – переконатися, що система є зручною та інтуїтивно зрозумілою для користувачів. Методи – дослідження юзабіліті, тестування з користувачами, тестування на основі сценаріїв.

Тестування сумісності, цей тип тестування перевіряє, як система взаємодіє з різними операційними системами, браузерами, апаратним забезпеченням та іншими системами. Ціль – забезпечити коректну роботу системи у різних середовищах. Методи – тестування крос-браузерної сумісності, тестування крос-платформної сумісності.

Тестування на зручність супроводу, цей тип тестування оцінює, наскільки легко систему можна модифікувати, виправляти та вдосконалювати. Ціль – забезпечити, що система легко піддається супроводу та модернізації. Методи – тестування читабельності коду, тестування структурованості.

Існує багато інструментів, які допомагають автоматизувати нефункціональне тестування:

1. LoadRunner: Інструмент для тестування продуктивності та навантаження.
2. JMeter: Популярний інструмент для навантажувального та стресового тестування.
3. Burp Suite: Інструмент для тестування безпеки веб-додатків.
4. OWASP ZAP: Інструмент для тестування вразливостей веб-додатків.
5. Selenium: Інструмент для тестування крос-браузерної сумісності.

6. Apache Benchmark (ab): Інструмент для тестування продуктивності веб-серверів.

Переваги нефункціонального тестування:

1. Покращення якості: Забезпечує високу якість системи, перевіряючи різні нефункціональні аспекти.
2. Підвищення надійності: Дозволяє виявляти потенційні проблеми, що можуть виникнути при тривалому використанні.
3. Покращення продуктивності: Оптимізує продуктивність системи, забезпечуючи її стабільну роботу під навантаженням.
4. Підвищення безпеки: Виявляє та усуває вразливості, що можуть загрожувати безпеці даних.

Недоліки нефункціонального тестування:

1. Високі витрати: Може потребувати значних ресурсів та часу для проведення комплексного тестування.
2. Складність виконання: Вимагає спеціалізованих знань та інструментів для ефективного проведення.
3. Обмежене охоплення: Може не охоплювати всі можливі сценарії використання системи, особливо в умовах реального світу.

Нефункціональне тестування є важливою частиною процесу забезпечення якості програмного забезпечення, оскільки воно допомагає забезпечити, що система не тільки працює правильно, але й відповідає всім вимогам щодо продуктивності, надійності, безпеки та зручності використання.

Статичне тестування – це тип тестування програмного забезпечення, при якому аналізуються артефакти (код, документація, дизайн) без виконання самого програмного коду. Воно проводиться на ранніх етапах розробки, дозволяючи виявити та виправити дефекти до етапу динамічного тестування або навіть перед початком кодування. Це сприяє зниженню вартості виправлення помилок і покращує якість кінцевого продукту.

Основні методи статичного тестування:

1. Рецензування (Review)

2. Статичний аналіз коду (Static Code Analysis)
3. Аналіз вимог (Requirements Analysis)

Рецензування – це процес систематичного аналізу документації та коду з метою виявлення помилок, порушень стандартів, неточностей або пропусків. Є два види рецензування, такі як рецензування коду (Code Review) та Рецензування документації (Document Review). Рецензування коду – це перегляд коду іншими розробниками для виявлення потенційних помилок або оптимізації. Рецензування документації – це аналіз документації, специфікацій, вимог тощо.

Статичний аналіз коду – це автоматизований аналіз коду за допомогою спеціальних інструментів, які перевіряють код на відповідність стандартам, виявляють потенційні помилки, уразливості та можливі проблеми продуктивності.

Аналіз вимог – це перевірка вимог до програмного забезпечення на повноту, коректність, несуперечність і можливість тестування. Мета – це переконатися, що вимоги повністю зрозумілі, однозначні та можливі для реалізації та тестування.

Процес статичного тестування складається з планування, підготовки, виконання, аналізу результатів, звітування та коригування. Планування містить у собі визначення цілей статичного тестування, вибір методів та інструментів для проведення аналізу, розподіл ролей і відповідальності. Підготовка містить у собі збір артефактів для аналізу (вимоги, дизайн, код), налаштування інструментів статичного аналізу, підготовка чек-листів або контрольних списків для рецензування. Виконання містить у собі проведення рецензування коду або документації, виконання автоматизованого статичного аналізу коду, аналіз вимог на відповідність критеріям якості. Аналіз результатів містить у собі визначення дефектів та проблем, оцінка важливості та пріоритетності виявлених дефектів, обговорення результатів у команді. Звітування та коригування містить у собі створення звітів про виявлені дефекти, виправлення виявлених помилок та повторний аналіз при необхідності, документування процесу та результатів для подальшого використання.

Існує багато інструментів, які автоматизують процес статичного аналізу коду:

1. SonarQube: Інструмент для безперервної інтеграції та аналізу якості коду.
2. Pylint: Аналізатор коду для Python, що перевіряє відповідність стандартам та виявляє потенційні проблеми.
3. Checkstyle: Інструмент для перевірки стилю коду в Java.
4. FindBugs: Статичний аналізатор для Java, який виявляє потенційні дефекти.
5. ESLint: Аналізатор коду для JavaScript, що допомагає дотримуватися стандартів та знаходити помилки.

Переваги статичного тестування:

1. Раннє виявлення дефектів: Дозволяє виявляти помилки на ранніх етапах розробки, що знижує вартість їх виправлення.
2. Поліпшення якості коду: Допомагає дотримуватися стандартів кодування та виявляти потенційні проблеми з продуктивністю та безпекою.
3. Зниження ризиків: Допомагає виявляти критичні проблеми ще до виконання коду, знижуючи ризик виникнення серйозних помилок у робочій системі.
4. Підвищення ефективності команди: За допомогою рецензування коду та документації підвищується загальна якість роботи команди, оскільки всі члени дотримуються однакових стандартів та практик.

Недоліки статичного тестування:

1. Обмежена ефективність: Не може виявити всі типи дефектів, зокрема ті, що проявляються лише під час виконання коду.
2. Залежність від досвіду рецензентів: Якість рецензування значною мірою залежить від досвіду та уваги рецензентів.
3. Часові витрати: Процес рецензування та статичного аналізу може бути трудомістким, особливо для великих проєктів.

Статичне тестування є важливою частиною процесу забезпечення якості програмного забезпечення, оскільки дозволяє виявляти дефекти на ранніх стадіях розробки, покращуючи загальну якість коду та документації.

Чорна скриня (Black Box Testing) – це метод тестування програмного забезпечення, при якому тестувальник оцінює функціональність системи, не маючи знання про її внутрішню структуру або код. Цей підхід зосереджується на перевірці входів та виходів системи, відповідності вимогам і специфікаціям, а також поведінці системи у різних умовах. Основні характеристики чорної скрині – це фокус на функціональності, незалежність від реалізації, сценарії використання. Фокус на функціональності – це тестування базується на вимогах та специфікаціях, а не на внутрішній структурі коду. Незалежність від реалізації – це, коли тестувальники не потребують знань про внутрішню логіку чи архітектуру системи. Сценарії використання – це тестування імітує реальні умови використання системи кінцевими користувачами.

Види тестування чорної скрині:

1. Тестування на основі вимог (Requirements-Based Testing)
2. Тестування на основі сценаріїв (Scenario-Based Testing)
3. Тестування на основі еквівалентних класів (Equivalence Partitioning)
4. Тестування на основі граничних значень (Boundary Value Analysis)
5. Тестування на основі приймальних критеріїв (Acceptance Testing)

Процес тестування чорної скрині:

1. Аналіз вимог і специфікацій
2. Розробка тестових випадків
3. Виконання тестів
4. Аналіз результатів
5. Звітування та виправлення

Переваги тестування чорної скрині:

1. Не потребує знання коду: Тестувальникам не потрібні технічні знання про реалізацію системи.

2. Перевірка з точки зору користувача: Фокус на функціональності, що важлива для кінцевих користувачів.
3. Широке охоплення: Може бути застосовано до будь-якої системи незалежно від технології та реалізації.

Недоліки тестування чорної скрині:

1. Обмежене охоплення внутрішньої логіки: Можливі дефекти у внутрішніх структурах залишаються непоміченими.
2. Висока ймовірність пропущення дефектів: Без знання внутрішньої структури важко покрити всі можливі шляхи та стани.
3. Залежність від специфікацій: Якість тестування сильно залежить від якості та повноти вимог та специфікацій.

Інструменти для автоматизації тестування чорної скрині:

1. Selenium: Інструмент для автоматизації тестування веб-додатків.
2. QTP/UFT: Інструмент для функціонального автоматизованого тестування.
3. Appium: Інструмент для автоматизації мобільних додатків.
4. TestComplete: Комплексний інструмент для автоматизації тестування настільних, веб- та мобільних додатків.

Тестування чорної скрині є важливою складовою забезпечення якості програмного забезпечення, дозволяючи перевірити його функціональність та відповідність вимогам з точки зору кінцевого користувача.

Біла скриня (White Box Testing) – це метод тестування програмного забезпечення, при якому тестувальник має доступ до внутрішньої структури коду і використовує ці знання для розробки тестових випадків. Цей підхід дозволяє перевіряти внутрішні операції системи, а не тільки її функціональність з точки зору користувача.

Основні характеристики білої скрині:

1. Глибоке розуміння коду: Тестувальники повинні розуміти структуру коду, алгоритми та логіку роботи системи.

2. Перевірка внутрішніх процесів: Тестування орієнтоване на верифікацію внутрішніх шляхів, умов, циклів і структур даних.
3. Прямий доступ до коду: Тестувальники мають доступ до вихідного коду, що дозволяє проводити детальний аналіз і перевірку.

Основні методи білої скрині – це покриття коду (Code Coverage), аналіз потоків даних (Data Flow Testing), тестування циклів (Loop Testing) та модульне тестування (Unit Testing).

Покриття коду, перевірка того, наскільки добре тестами покритий вихідний код програми. Методи покриття, такі як покриття операторів (Statement Coverage) – визначає, чи всі оператори в коді були виконані, покриття гілок (Branch Coverage) – перевіряє, чи всі гілки в умовних операторів були пройдені, покриття умов (Condition Coverage) – перевірка всіх умовних виразів на істинність та хибність, покриття шляхів (Path Coverage) – перевірка всіх можливих шляхів через програму.

Аналіз потоків даних – аналіз і перевірка шляхів передачі даних між змінними в програмі. Ціль – виявлення аномалій, таких як використання неініціалізованих змінних, непотрібні змінні або неправильно визначені змінні.

Тестування циклів – перевірка коректності виконання циклічних конструкцій у програмі. Ціль – переконатися, що цикли працюють правильно при різних умовах, таких як нульові і граничні значення.

Модульне тестування – перевірка окремих модулів або компонентів системи для забезпечення їхньої коректної роботи. Ціль – виявлення дефектів у невеликих частинах коду ізольовано від решти системи.

Процес тестування білої скрині складається з п’яти етапів, а саме: аналіз коду, розробка тестових випадків, виконання тестів, аналіз результатів, звітування та виправлення.

Переваги тестування білої скрині:

1. Високе покриття: Забезпечує високе покриття коду та виявлення дефектів, які можуть бути пропущені при тестуванні Чорного ящика.

2. Оптимізація коду: Допомагає виявляти неефективні частини коду та можливості для його оптимізації.
3. Поглиблений аналіз: Дозволяє глибше аналізувати логіку програми та виявляти складні дефекти.

Недоліки тестування білої скрині:

1. Висока вартість: Потребує значних ресурсів та часу, оскільки вимагає детального аналізу коду.
2. Технічна складність: Потребує глибоких технічних знань та досвіду роботи з кодом.
3. Обмежена перевірка користувацьких сценаріїв: Фокус на внутрішній логіці може призводити до недостатньої уваги до реальних умов використання системи.

Існує багато інструментів, які допомагають автоматизувати процес тестування білої скрині:

1. JUnit: Інструмент для модульного тестування на мові Java.
2. NUnit: Інструмент для модульного тестування на мові C#.
3. CppUnit: Інструмент для модульного тестування на мові C++.
4. PyTest: Інструмент для модульного тестування на мові Python.
5. JaCoCo: Інструмент для вимірювання покриття коду в Java.

Тестування білої скрині є важливою складовою забезпечення якості програмного забезпечення, оскільки дозволяє глибоко аналізувати та перевіряти внутрішню логіку та структуру коду, забезпечуючи високу якість і надійність кінцевого продукту.

Регресійне тестування – це тип тестування програмного забезпечення, що проводиться для перевірки того, що зміни у коді (як-от виправлення помилок, додавання нових функцій або оптимізація існуючого коду) не вплинули негативно на вже існуючу функціональність системи. Основна мета регресійного тестування – виявлення помилок, які могли бути ненавмисно введені в результаті модифікацій.

Основні характеристики регресійного тестування:

1. Перевірка стабільності: Переконується, що нові зміни не порушують роботу існуючого функціоналу.
2. Періодичність: Проводиться після кожної значної зміни або виправлення у програмному забезпеченні.
3. Автоматизація: Часто автоматизується для зменшення часу та зусиль, необхідних для повторного виконання тестів.

Є три види регресійного тестування, такі як повне регресійне тестування, вибіркове регресійне тестування (Selective Regression Testing), прогресивне регресійне тестування (Progressive Regression Testing).

Повне регресійне тестування включає повторне виконання всіх тестових випадків для всієї системи. Переваги – забезпечує максимальне покриття та виявлення потенційних дефектів. Недоліки – вимагає багато ресурсів та часу, особливо для великих систем.

Вибіркове регресійне тестування – включає виконання тільки вибраних тестових випадків, які є найбільш критичними або які можуть бути найбільш імовірними для впливу змін. Переваги – зменшує час і ресурси, необхідні для тестування. Недоліки – може пропустити деякі дефекти, якщо вибір тестів не є достатньо ретельним.

Прогресивне регресійне тестування включає виконання тестів, розроблених спеціально для нових або змінених частин коду, а також тестів, які можуть бути потенційно вплинуті змінами. Переваги – фокусує увагу на нових та змінених компонентах, що дозволяє швидко виявити дефекти. Недоліки – може не виявити дефекти в інших частинах системи, якщо вони не були враховані.

Процес регресійного тестування є стандартним та включає в себе визначення обсягу тестування, розробку тестових випадків, виконання тестів, аналіз результатів, звітування та виправлення.

Переваги регресійного тестування:

1. Забезпечення стабільності: Переконується, що зміни в коді не вводять нові помилки в існуючу функціональність.

2. Підвищення якості: Допомагає підтримувати високу якість програмного забезпечення, постійно перевіряючи його функціональність.
3. Ефективність автоматизації: Автоматизовані тести значно знижують час і ресурси, необхідні для повторного тестування.

Недоліки регресійного тестування

1. Часові витрати: Повторне виконання великої кількості тестів може бути дуже часозатратним.
2. Ресурсозатратність: Вимагає значних ресурсів, особливо для повного регресійного тестування.
3. Підтримка тестів: Потрібно постійно оновлювати та підтримувати тестові випадки, щоб вони відповідали поточному стану коду.

Існує багато інструментів, які допомагають автоматизувати процес регресійного тестування:

1. Selenium: Інструмент для автоматизації тестування веб-додатків.
2. JUnit: Інструмент для модульного тестування на мові Java.
3. TestNG: Інструмент для тестування на мові Java, який підтримує паралельне виконання тестів.
4. QTP/UFT: Інструмент для функціонального автоматизованого тестування.
5. Jenkins: Інструмент для безперервної інтеграції, що дозволяє автоматизувати виконання тестів після кожної збірки.

Регресійне тестування є критично важливим елементом забезпечення якості програмного забезпечення, оскільки воно допомагає зберегти стабільність і надійність системи після внесення змін. Використання автоматизованих інструментів може значно підвищити ефективність цього процесу.

Автоматизоване тестування – це процес використання спеціальних програмних інструментів для автоматичного виконання тестових випадків і порівняння фактичних результатів з очікуваними. Цей підхід допомагає знизити

людські помилки, скоротити час тестування та підвищити ефективність перевірки програмного забезпечення.

Основні характеристики автоматизованого тестування

1. Автоматизація рутинних завдань: Виконання тестових випадків автоматично за допомогою скриптів або програм.
2. Повторюваність: Тести можуть виконуватися багаторазово з однаковими результатами.
3. Швидкість: Автоматизовані тести виконуються набагато швидше, ніж ручні тести.
4. Масштабованість: Легко масштабувати тестування на великі обсяги даних або комплексні системи.

Автоматизоване тестування включає в себе різни види тестувань, а саме: функціональне тестування – перевірка функціональності програмного забезпечення відповідно до специфікацій, регресійне тестування – перевірка, що зміни у коді не вплинули на існуючий функціонал, навантажувальне тестування – перевірка продуктивності системи під різними умовами навантаження, тестування інтерфейсу користувача (UI Testing) – перевірка роботи графічного інтерфейсу користувача, API тестування – Перевірка функціональності API (інтерфейсів прикладного програмування), інтеграційне тестування – перевірка взаємодії між різними компонентами системи.

Процес включає в себе вибір інструменту для тестування, розробку тестових випадків, налаштування середовища тестування, виконання тестів, аналіз результатів, звітування та виправлення, повторне тестування.

Переваги автоматизованого тестування:

1. Швидкість і ефективність: Автоматизовані тести виконуються швидше, ніж ручні, що скорочує час тестування.
2. Точність: Виключає людські помилки при виконанні повторюваних завдань.
3. Економія часу і ресурсів: Можливість повторного використання автоматизованих тестів для різних версій ПЗ.

4. Безперервне тестування: Інтеграція з CI/CD системами дозволяє проводити тестування після кожної зміни коду.

Недоліки автоматизованого тестування:

1. Висока початкова вартість: Вимагає часу та ресурсів на розробку автоматизованих тестів і налаштування середовища.
2. Підтримка і оновлення: Необхідно постійно підтримувати і оновлювати автоматизовані тести, щоб вони відповідали актуальному стану ПЗ.
3. Обмеження на деякі види тестів: Деякі види тестів (наприклад, тестування юзабіліті) важко автоматизувати.

Популярні інструменти для автоматизованого тестування:

1. Selenium: Відкритий інструмент для автоматизації тестування веб-додатків.
2. JUnit: Інструмент для автоматизованого модульного тестування на мові Java.
3. Apache JMeter: Інструмент для навантажувального тестування.
4. TestComplete: Комерційний інструмент для автоматизації тестування настільних, веб- та мобільних додатків.
5. QTP/UFT: Комерційний інструмент для функціонального автоматизованого тестування.
6. Appium: Інструмент для автоматизації тестування мобільних додатків.
7. Postman: Інструмент для автоматизованого тестування API.
8. Jenkins: Інструмент для безперервної інтеграції, що дозволяє автоматизувати виконання тестів.

Автоматизоване тестування є важливою складовою сучасного процесу розробки програмного забезпечення, оскільки дозволяє значно підвищити ефективність та якість тестування, знижуючи витрати часу і ресурсів на рутинні завдання.

Ручне тестування – це процес тестування програмного забезпечення, при якому тестувальник вручну виконує тестові випадки без використання

автоматизованих інструментів. Цей підхід дозволяє ретельно перевірити функціональність, юзабіліті та інші аспекти системи з точки зору кінцевого користувача.

Основні характеристики ручного тестування:

1. Виконання вручну: Тестувальники виконують тестові сценарії без використання автоматизованих скриптів.
2. Аналіз користувацького досвіду: Дозволяє оцінити програму з точки зору кінцевого користувача.
3. Гнучкість: Легко адаптувати тестування під нові вимоги або зміни в програмному забезпеченні.

Ручне тестування включає в себе різни види тестувань, а саме: функціональне тестування, юзабіліті тестування, інтеграційне тестування, системне тестування, прийнятне тестування (Acceptance Testing), тестування на відповідність.

Процес ручного тестування є стандартним та включає в себе аналіз вимог, розробку тестових випадків, налаштування середовища тестування, виконання тестів, аналіз результатів, звітування та виправлення, потворне тестування.

Переваги ручного тестування:

1. Гнучкість: Легко адаптувати тестування під зміни у вимогах або нові функції.
2. Оцінка юзабіліті: Дозволяє оцінити зручність використання та взаємодії з інтерфейсом.
3. Креативність: Тестувальники можуть використовувати креативні підходи для виявлення дефектів, які можуть бути пропущені автоматизованими тестами.
4. Мінімальні витрати на початку: Не потребує значних початкових інвестицій в інструменти або налаштування.

Недоліки ручного тестування:

1. Часозатратність: Виконання тестів вручну займає більше часу, особливо для великих систем.

2. Висока ймовірність людських помилок: Тестувальники можуть помилятися при виконанні повторюваних завдань.
3. Неможливість масштабування: Ручне тестування важко масштабувати для великих обсягів тестів або частого виконання.
4. Обмежене покриття: Можливе неповне покриття тестів через обмежений час та ресурси.

Інструменти для допомоги в ручному тестуванні:

1. JIRA: Інструмент для управління дефектами та завданнями, що дозволяє документувати та відстежувати дефекти.
2. TestRail: Інструмент для управління тестуванням, що дозволяє створювати, виконувати та відстежувати тестові випадки.
3. Excel/Google Sheets: Використовуються для створення та відстеження тестових випадків та результатів тестування.
4. Bugzilla: Система для відстеження дефектів, що дозволяє документувати та управляти дефектами.
5. Confluence: Інструмент для створення та зберігання документації, включаючи тестові плани та звіти.

Ручне тестування залишається важливою складовою процесу забезпечення якості програмного забезпечення, оскільки дозволяє гнучко реагувати на зміни, забезпечувати високий рівень юзабіліті та виявляти дефекти, які можуть бути пропущені автоматизованими тестами.

Проаналізувавши різні типи тестування програмного забезпечення було вирішено проводити тестування білої скриньки та використовувати метод модульного тестування (Unit Testing).

4.2 Результати тестування

Приклад тесту

```
[TestClass]
public class UnitTest1
{
```

```
    [TestMethod]
    public void SetShowAboutCarView()
    {
        IShowAboutCarView view = new ShowAboutCarView();
        ShowAboutCarView sacv = new ShowAboutCarView();
        sacv.SetView(view);
        var expected = view;
        var actual = sacv._view;
        Assert.AreEqual(expected, actual);
    }
}
```

Результати тестувань наведені на рис. 4.1.

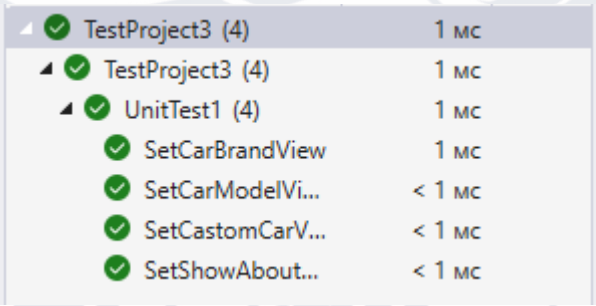

Рисунок 4.1 результати тестування

4.3 Верифікація відповідності архітектурному шаблону

Верифікація відповідності архітектурному шаблону — це процес перевірки, чи відповідає система або її компоненти заданому архітектурному шаблону або стандарту. Цей процес включає в себе оцінку того, наскільки дизайн системи дотримується визначених архітектурних принципів, структур і методологій.

Вимоги до реалізації архітектурного шаблону MVP:

1. Model
    1) Модель відповідає за управління даними додатка. Вона містить бізнес-логіку та правила для маніпулювання даними.
    2) Вимоги

a) Забезпечує доступ до даних (отримання та збереження даних з бази даних, веб-сервісів тощо).
b) Не повинна містити код, пов’язаний з відображенням даних (UI).
c) Повинна надавати інтерфейси для пред’явника для виконання операцій CRUD (створення, читання, оновлення, видалення).

2. View
   1) Вигляд відповідає за відображення даних користувачеві та передачу команд користувача до пред’явника.
   2) Вимоги
      a) Відповідає за відображення інтерфейсу користувача.
      b) Не повинна містити бізнес-логіку або безпосередньо взаємодіяти з моделлю.
      c) Повинна мати мінімальний обсяг логіки, пов'язаної з відображенням.
      d) Має бути інтерактивною, тобто вміти реагувати на дії користувача та передавати ці дії пред’явника.
3. Presenter
   1) Пред’явник виступає посередником між моделлю та виглядом. Він отримує дані з моделі, обробляє їх і передає вигляду для відображення.
   2) Вимоги
      a) Отримує команди від вигляду та запити до моделі для отримання/збереження даних.
      b) Обробляє дані та бізнес-логіку, перед тим як передати дані до вигляду.
      c) Повинен бути незалежним від специфічного виду інтерфейсу користувача.
      d) Має забезпечувати синхронізацію стану між Моделлю та вигляду.

е) Забезпечує оновлення вигляду у відповідь на зміни у Моделі.

Проаналізувавши створений додаток усі вимоги до реалізації обраного архітектурного шаблону MVP.

Висновки до розділу 4

Був проведений огляд існуючих підходів до тестування, проведений їх аналіз та обраний вид та метод тестування. Було проведене успішне тестування розробленої програми. Була проведена верифікація розробленої програми відповідності архітектурному шаблону.

ВИСНОВКИ

Огляд існуючих аналогів показав, що вони не є ідеальними та потребують удосконалення. А з розвитком комп’ютерних технологій це неминуче.

Ознайомлення з існуючими архітектурними шаблонами показав їх різноманітність та функціональність у всіх сферах їх використання. Та показав їх необхідність при розробці програмного забезпечення.

Проектування програми не є легкою задачею, воно має багато кроків та нюансів у кожному з них. Правильно спроектовані програми мають високі шанси на їх успішну реалізацію та отримання великої популярності.

Реалізація програми є важливою частиною подальшого успіху програмного забезпечення. Вона має великий вплив на майбутнє удосконалення та розширення функціоналу програми. Що з набуттям популярності є неминучим.

Тестування та верифікація програми один з найважливіших аспектів для її майбутнього. Важливо щоб тестування було успішне та покривало якнайбільшу її частину. Верифікація повинна проводитись завжди, щоб уникнути конфліктів у коді при її розробці та подальшому удосконаленню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О. П. Савич Сучасні тенденції і перспективи розвитку глобального автомобільного ринку, 2016. URL: http://www.economy.nayka.com.ua/?op=1&z=5178
2. Стаття “найбільші автовиробники світу” URL: https://largesthq.com/najbilshi-avtovirobniki-svitu/
3. Стаття “ Що таке тюнінг автомобіля?” URL: https://avtoblog.in.ua/scho-take-tyuninh-avtomobilya/
4. Стаття “ Тюнінг-ательє ” URL: https://uk.wikipedia.org/wiki/Тюнінг-ательє
5. Стаття “ Bonneville Motorcycle Speed Trials ” URL: https://topmoto.pro/ua/blog/bonneville-salt-flats-motorcycle
6. Стаття “A bluffer's guide to the Bonneville Salt Flats” URL: https://www.redbull.com/int-en/bonneville-salt-flats-land-speed-records-history
7. Кеттвелл, Майк. «Індіанаполіс: найбагатша раса в світі», у Northey, Tom, ed. Світ автомобілів (Лондон: Orbis, 1974), том 9, 1012 с.
8. Стаття “ The Greatest Car Tuners To Ever Exist ” URL: https://www.hotcars.com/the-greatest-car-tuners-to-ever-exist/
9. Стаття “ The History: SUCCESSES IN THE REARVIEW MIRROR” URL: https://www.mercedes-amg.com/en/footer/about-us/history.html
10. Стаття “ AC Schnitzer Chronicle ” URL: https://www.ac-schnitzer.de/en/about-us/history/
11. Стаття “ABOUT BRABUS” URL: https://www.sandown-brabus.co.uk/about/
12. Стаття “ Take a look inside Liberty Walk’s mind-blowing HQ ” URL: https://www.topgear.com/car-news/tgs-guide-japan/take-look-inside-liberty-walks-mind-blowing-hq
13. Стаття “Here’s The History of Ruf, One of The Greatest Supercar Builders” URL: https://www.thedrive.com/guides-and-gear/what-is-ruf-porsche
14. Стаття “Rev Your Engines: The History of JDM Culture” URL: https://jdm-obsession.com/blogs/jdm-lifestyle-blog/rev-your-engines-the-history-of-jdm-culture
15. Стаття “Car tuning guide: everything you need to know” URL: https://www.carvertical.com/blog/car-tuning
16. Конфігуратор Porsche URL: https://configurator.porsche.com/en-WW/model-start/
17. Конфігуратор BMW URL: https://www.bmw.co.uk/en/configurator.html
18. Конфігуратор Mercedes-Benz URL: https://www.mbusa.com/en/vehicles/build
19. Офіційний сайт Mercedes-Benz URL: https://www.mercedes-benz.com/en/

20. Стаття “Історія компанії Mercedes-Benz” URL: https://group.mercedes-benz.com/company/tradition/company-history/
21. Офіційний сайт BMW URL: https://www.bmw.com/en/index.html
22. Стаття “Історія компанії BMW” URL: https://www.bmw.com/en/automotive-life/BMW-name-meaning-and-history.html
23. Офіційний сайт Porsche URL: https://www.porsche.com/
24. Стаття “Історія компанії Porsche” URL: https://www.porsche.com/international/aboutporsche/christophorusmagazine/archive/379/articleoverview/article14/
25. Додаток на телефон “Car Scanner ELM OBD2” URL: https://play.google.com/store/apps/details?id=com.ovz.carscanner&hl=ua
26. Сервіс для підбору автомобіля “CarFinder” URL: https://carfinder.ua/uk/
27. Компанія з автопідбору авто “ArchiLowPodbor”URL: https://archilowpodbor.com/
28. Побудова автомобіля в конфігураторі Mercedes-Benz URL: https://www.mbusa.com/en/vehicles/build/c-class/sedan/c43w4
29. Стаття “Архітектура програмного забезпечення: все що треба знати” URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya
30. Стаття “Історія патернів” URL: https://refactoring.guru/uk/design-patterns/history
31. Книга Еріх Гамма, Джон Вліссідес, Річард Гелм, Ральф Джонсон “Design Patterns: Elements of Reusable Object-Oriented Software”, 1994 – 416 с.
32. Документація GitHub URL: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/about-merge-conflicts
33. Офіційний сайт Git URL: https://git-scm.com/
34. Книга Скот Чакон, Беном Страуб “GitPro”, 2009 – 440 с.
35. Офіційний сайт GitLab URL: https://about.gitlab.com/
36. Стаття “ GitLab 14 Delivers Modern DevOps in One Platform ” URL: https://www.devprojournal.com/news/gitlab-acquires-unreview-to-expand-its-open-devops-platform-with-machine-learning-capabilities-2/
37. Офіційний сайт BitBucket URL: https://bitbucket.org/
38. Офіційний сайт Jira URL: https://www.atlassian.com/software/jira
39. Офіційний сайт Replit URL: https://replit.com/
40. Книга Бред Вілсон, К Скот Ален, Девід Метсон, Джон Галловей “Professional ASP.NET MVC 5”, 2014 – 624 с.
41. Книга Грегор Хопе, Боббі Вулф “Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions”, 2011 – 688 с.

42. Книга Пітер Найс “The MVVM Pattern in .NET MAUI: The definitive guide to essential patterns, best practices, and techniques for cross-platform app development” 2023 – 386 с.
43. Книга Габріель Баптіста, Франческо Аббруцезе “Software Architecture with C# 12 and .NET 8 - Fourth Edition: Build enterprise applications using microservices, DevOps, EF Core, and design patterns for Azure”, 2024 – 756 с.
44. Книга Тхуан Тхай, Хоанг Лам “.NET Framework Essentials: Introducing the .NET Framework”, 2003 – 380 с.
45. Книга Кріс Селлс “Windows Forms Programming in C#”, 2003 734 с.
46. Стаття “When Was C# Created? A Brief History” URL: https://csharp-station.com/when-was-c-sharp-created-a-brief-history/
47. Офіційний сайт Java URL: https://www.java.com/
48. Офіційний сайт Turbo Pascal URL: https://turbopascal.org/
49. Офіційний сайт Delphi URL: https://www.embarcadero.com/products/delphi
50. Офіційний сайт Visual Studio URL: https://visualstudio.microsoft.com/
51. Офіційний сайт Azure URL: https://azure.microsoft.com/
52. Офіційний сайт Microsoft SQL Server URL: https://www.microsoft.com/en-us/sql-server
53. Стаття “What is SQL?” URL: https://aws.amazon.com/what-is/sql/
54. Entity Framework documentation hub URL: https://learn.microsoft.com/en-us/ef/
55. Language Integrated Query (LINQ) URL: https://learn.microsoft.com/en-us/dotnet/csharp/linq/
56. Книга Авраменко А.С., Авраменко В.С. Косенюк Г.В. “ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ” Навчальний посібник, 2017 – 284 с.
57. Книга Дороті Грем Foundations of Software Testing, 2006 – 243 с.
58. Книга Срінівасан Десікан, Гопаласвами Рамеш Software Testing: Principles and Practices, 2009 – 480 с.
59. Книга Брет Петтіхорд, Сем Канер, Джеймс Бах Lessons Learned in Software Testing: A Context-Driven Approach, 2001 – 320 с.

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)