КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Програмна реалізація генерації конфігурації персонального комп'ютера»

Виконав
(Підпис)
Дацюк Євгеній Геннадійович
(прізвище, ім'я, по батькові)

Керівник Шолохов Олексій Вікторович
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри Плескач В.Л. _____
(Підпис) (Прізвище, ініціали) (Дата)

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань. Унікальність тексту 93%.

Київ - 2022

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: «Програмна реалізація генерації конфігурації персонального комп'ютера»

Освітня програма: Прикладне програмування
Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Дацюк Євгеній Геннадійович	

Назва роботи українською та англійською мовами

Програмна реалізація генерації конфігурації персонального комп'ютера.
Software implementation of personal computer configuration generation.

Мета бакалаврської кваліфікаційної роботи, завдання

Мета роботи: Створити застосунок, за допомогою якої, користувач зможе підібрати для себе комплектуючі стаціонарного комп’ютеру.

План роботи:

1. Підходи до розроблення застосунків для комплектації ПК
2. Аналіз програмно-технологічних рішень реалізації
3. Реалізація та впровадження застосунку створення конфігурації за заданими параметрами

Кандидат фізико-математичних наук,

доцент кафедри прикладних інформаційних систем **Шолохов О.В.**: 

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Номер	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2022	виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2022	виконано
9.	Подання роботи у першому варіанті	28.04.2022	виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**23.05.2022**	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2022	Виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	10.06.2022	виконано
14.	Захист кваліфікаційної роботи бакалавра	22.06.2022 23.06.2022 24.06.2022	

Здобувач вищої освіти  (підпис)

Керівник 

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Завдання до дипломної роботи(календарний план дипломної роботи)	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
Розділ 1	7
Розділ 2	18
Розділ 3	10
Висновок	1
Перелік використаних джерел	1
Додатки	10

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Десктоп - Настільний комп'ютер, стаціонарний персональний комп'ютер.

Winforms - Windows Forms — інтерфейс програмування програм, який відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework.

EUD - Розробка для кінцевих користувачів

ЗМІСТ

АНОТАЦІЯ

Кваліфікаційна робота: 58 с., 23 рис., 7 джерел, 1 дод.

Об’єктом дослідження програмний застосунок.

Предметом дослідження розробка застосунку для підбору комплектуючих.

Проблему дослідження можна сформулювати у питанні: Як підвищити ефективність витрачення часу та створення потрібної конфігурації ПК за допомогою розробленого застосунку?

Метою роботи є побудова застосунку для допомоги підбору комплектуючих персонального комп’ютера.

У ході дослідження було проведено: аналіз аналогів рішень, контент-аналіз веб-застусунків для підбору конфігурації.

Новизна роботи полягає в тому, що аналіз аналогів виявив відсутність цього продукту на ринку, а також перевагою такого рішення для комерційних компаній, які мають за ціль подальшу роботу над проектами комплектування персональних комп’ютерів.

Практичне значення полягає в тому, що створюється унікальний продукт, що забезпечує ефективність використаного часу, пониження порогу входу для розуміння в комплектуючих та комфорт використання продукту.

Ключові слова: десктоп-застосунок, конфігурація, персональний комп’ютер.

ANNOTATION

Qualification work: 58 pp., 23 figs., 7 sources, 1 appendix.

The object of study is a software application.

The subject of research is the development of applications for the selection of components.

The research problem can be formulated in the question: How to increase the efficiency of time and create the desired PC configuration with the help of the developed application?

The purpose of the work is to build an application to help select personal computer components.

In the course of the research the following was conducted: analysis of analog solutions, content analysis of web applications for configuration selection.

The novelty of the work is that the analysis of analogues revealed the absence of this product on the market, as well as the advantage of this solution for commercial companies that aim to further work on projects to complete personal computers.

The practical significance lies in the fact that a unique product is created that ensures the efficiency of the time used, lowering the entry threshold for understanding the components and the comfort of using the product.

Keywords: desktop application, configuration, personal computer.

ВСТУП

На сучасному етапі технологічного розвитку суспільство кожен день робить кроки, які все більше наближують людство до відповідей на запит на комфорт, та що найголовніше – ефективність використання часу, бо це один з тих ресурсів, що ми не можемо витрачати марно. Наразі кожен з нас усвідомлює, що використання комп'ютерних інформаційних технологій – це найбільш корисна навичка. Актуальність теми достатньо обґрунтувати тим, що персональний комп'ютер надає багато можливостей для розвитку, відпочинку, бізнесу та, звісно, роботи.

Якщо ми заглянемо у недалеке минуле, то побачимо, що більшість людей, що хотіли придбати персональний комп'ютер робили свій вибір на користь спеціалізованих на цій сфері комерційних компаній, що з одного боку мали й великий та якісний досвід, супроводжали цикл від абстракцій на основі характеристик ПК, що вони хотіли отримати, аж до кінцевого результату у вигляді вже зібраного рішення, а з другої – значну переоцінку за свої навички. То б то виходило так, що компанії, які розумілися на, здавалося б, доволі простих речах, могли отримувати несумірний результат у еквіваленті грошового ресурсу.

Першочерговою проблемою, на яку треба звернути увагу – це різниця кінцевої суми витрачених коштів. Користуючись додатком є можливість збільшити цю різницю до цифр, які будуть доволі впливовими, тим паче у процентному співвідношенні.

Мета дослідження – побудова застосунку для допомоги підбору комплектуючих персонального комп'ютера.

Об'єкт дослідження – програмний застосунок.

Предмет дослідження – розробка застосунку для підбору комплектуючих.

Також, новизна полягає в тому, що питання цільової аудиторії, не закінчується розробкою тільки B2C цілей. На мою думку, перевага такого

рішення є те, що комерційні компанії, які мають за ціль подальшу роботу над проектами комплектування персональних комп’ютерів теж мають нагоду використання продукту. З метою зменшення порогу входу у розуміння концепції можна підкреслити, що використання застосунку є відмінним рішенням.

РОЗДІЛ 1 ПІДХОДИ ДО РОЗРОБЛЕННЯ ЗАСТОСУНКІВ ДЛЯ КОМПЛЕКТАЦІЇ ПК

Під час вивчення та аналізу питання, як саме можна реалізувати такий додаток, було вирішено, що його буде реалізовано у вигляді десктопного застосунку, це дасть змогу подальшої змоги масштабувати , а також зручно використовувати його в різних напрямках реалізації на практиці.

Після чого були поставлені такі завдання:

- Проаналізувати актуальність та перспективи розробки додатку;
- Проаналізувати складність написання даного додатку та відібрати кращі варіанти для полегшення розробки;
- Розробити застосунок використовуючи Microsoft Visual Studio;

Для написання веб-застосунку я обрав C# - об’єктно-орієнтована мова програмування. Також використовувалась технологія Window Forms, що є складовою платформи .NET. для створення графічних інтерфейсів. Для створення графічних програм на C# було використовуване безкоштовне та повнофункціональне середовище розробки - Visual Studio Community 2022 (Рисунок 1.1.1).

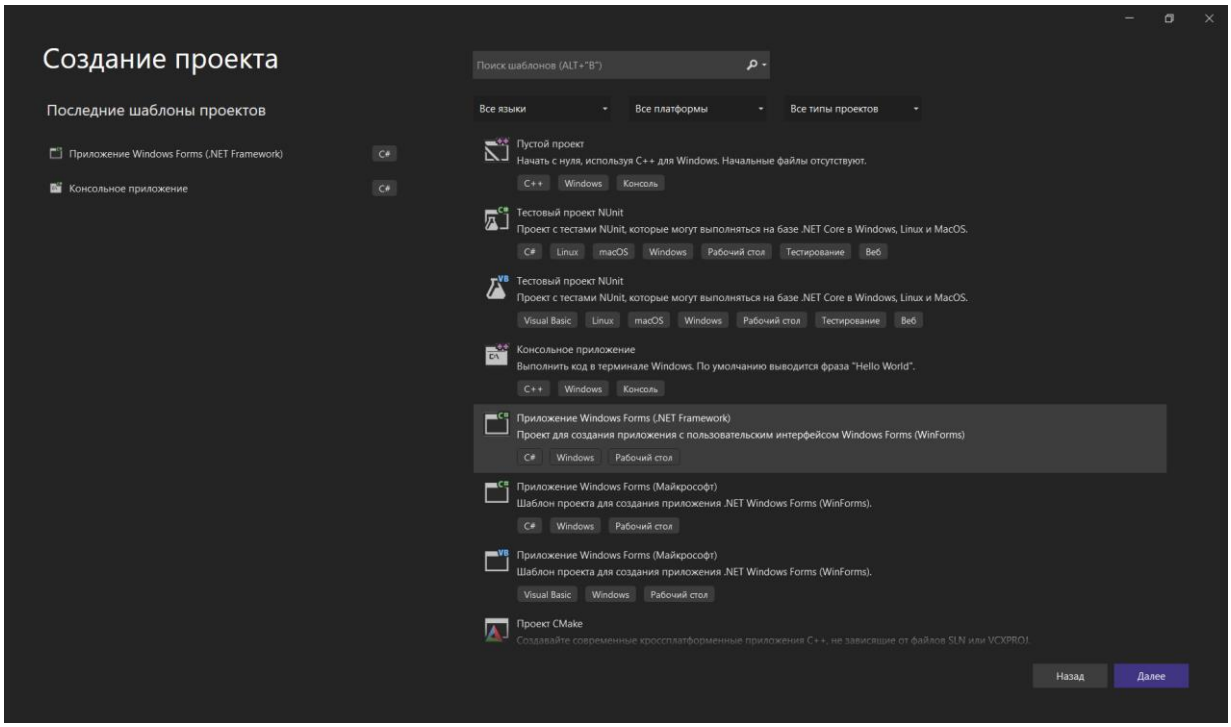


Рисунок 1.1.1 - Visual Studio Community 2022 (WinForm Creating)

Застосунки, що розроблялись для десктопу — це повнофункціональні програми, що працюють незалежно від інших програм та потребують присутності оператора. Для їх роботи необхідні достатні апаратні ресурси комп'ютера, сам додаток та набір функцій, що працюють із додатком. Такі програми розміщуються на машині усіх, хто хоче її використовувати. Вони, у відмінності від веб-застосунків, не потребують вайфаю або Ethernet для роботи, взаємодіють з користувачем через стандартний інтерфейс. Треба підкреслити, що продуктивність, залежить тільки від комплектації та потужностей, що використовується.

C# — об’єктно-орієнтована мова програмування, яка базується на декількох основних парадигмах таких як:
- Інкапсуляція зводить дані і код, що маніпулює цими даними, а також максимізує і забезпечує більшу закритість і безпеку. У результаті - можуть маніпулювати тільки частиною способів об'єкта. В результаті це по суті концепція закритої реалізації програмних елементів для захищеності та більшого забезпечення безпеки, як правило, шляхом застосування модифікаторів доступу;

- Успадкування – якщо говорити конкретно, то це грубо кажучи створення нового об’єкту на основі старого. То б то об’єкт успадкований може приймати властивості іншого. Практичне застосування ця парадигма показує при побудові ієрархії;

- Поліморфізм – «ефект» успадкування, коли ми маємо змогу поповнювати об’єкт конкретним функціоналом та його можливостями. Це надає змогу об’єкту перебувати у різних формах;

- Абстракція – об’єкту надаються характеристики, акцентуючи увагу на значних його характеристиках, та виключають менш значні.

Отже на цьому етапі ми дали відповідь на питання підходу до розробки застосунків для десктопів, окреслили стек, та розуміння технологій для реалізації проекту.

1.1 Аналоги розробки застосунків

Для приведення аналогів обов’язково треба зауважити на веб-застосунках, які по суті є конкурентними для десктоп застусунків у цій сфері. Наразі цей напрям щодня збільшує свою аудиторію, це приведено нижче на рисунку 1.1.2, за графіком зростяння кількості користувачів всесвітньою павутиною.

Рисунок 1.1 - Соціально-демографічна структура “регулярних” інтернет-користувачів

Якщо ми торкаємося теми аналогів розробки, то б то технологій, які в пріоритеті могли б задати той самий вектор, що обраний мною, то треба описати можливі технології, наприклад фреймворки для разробки.

WPF

Технологія, що стала популярна на початку 2000-х для реалізації застосунків. Це технологічна структурована в рамках .NET функціональна технологія, яка за свою основу створена для створення графічної частини десктопної програми. WPF є однією конструкцією зі стеку .NET з початку 2000-х, вона завоювала серця великої кількості кодерів. Однією з найважливіших властивостей WPF є можливість уніфікувати різноманітні елементи частини графічного інтерфейсу. Загалом вона потрібна для створення інтерфейсу програмного забезпечення кінцевого користувача. Далі зображена діаграма WPF у .NET Framework

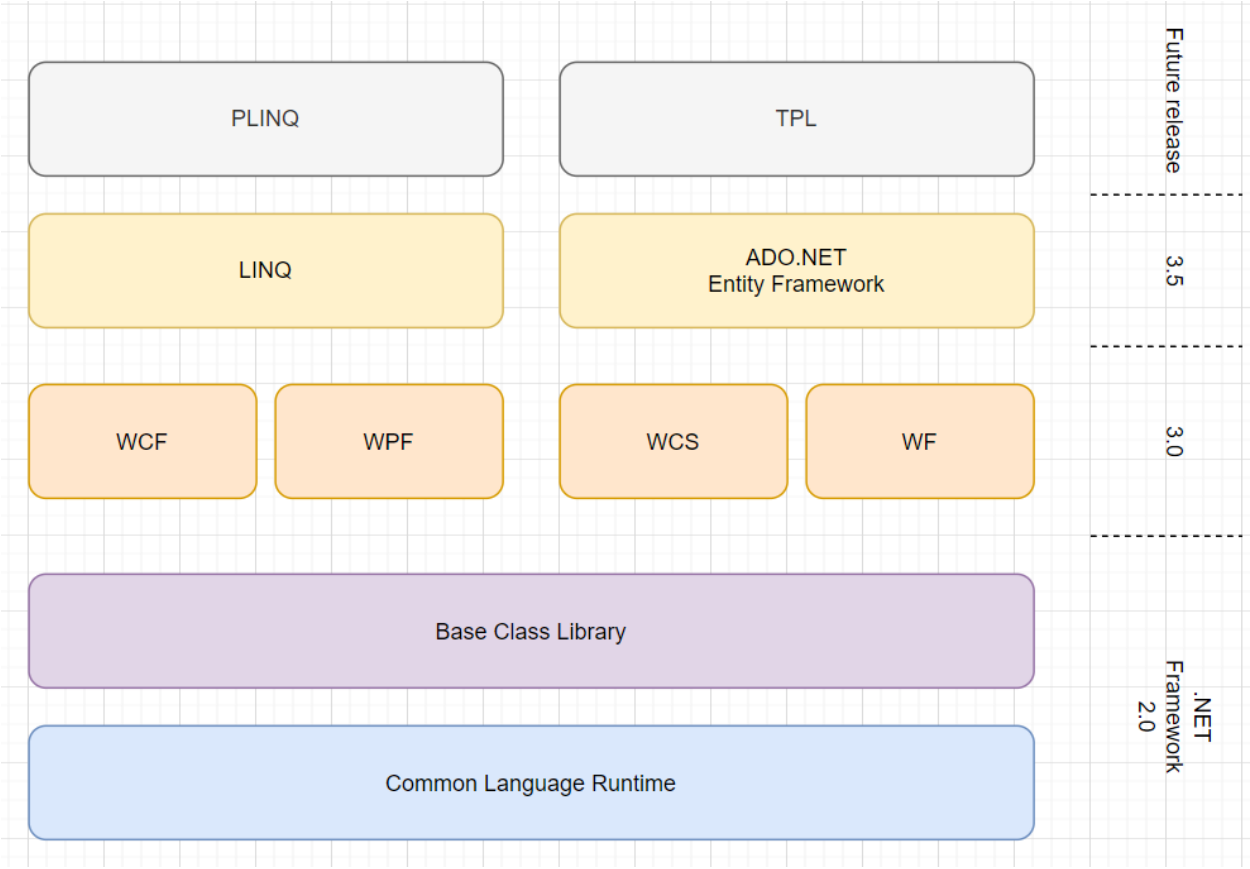


Рисунок 1.1.2 - Діаграма структури інструменту WPF на платформі .NET

1.2 Аналоги реалізованих системних рішень

Одним з аналогів програмної реалізації для допомоги комплектації персонального комп'ютеру є Конфігуратор ПК на сайті компанії Telemart.

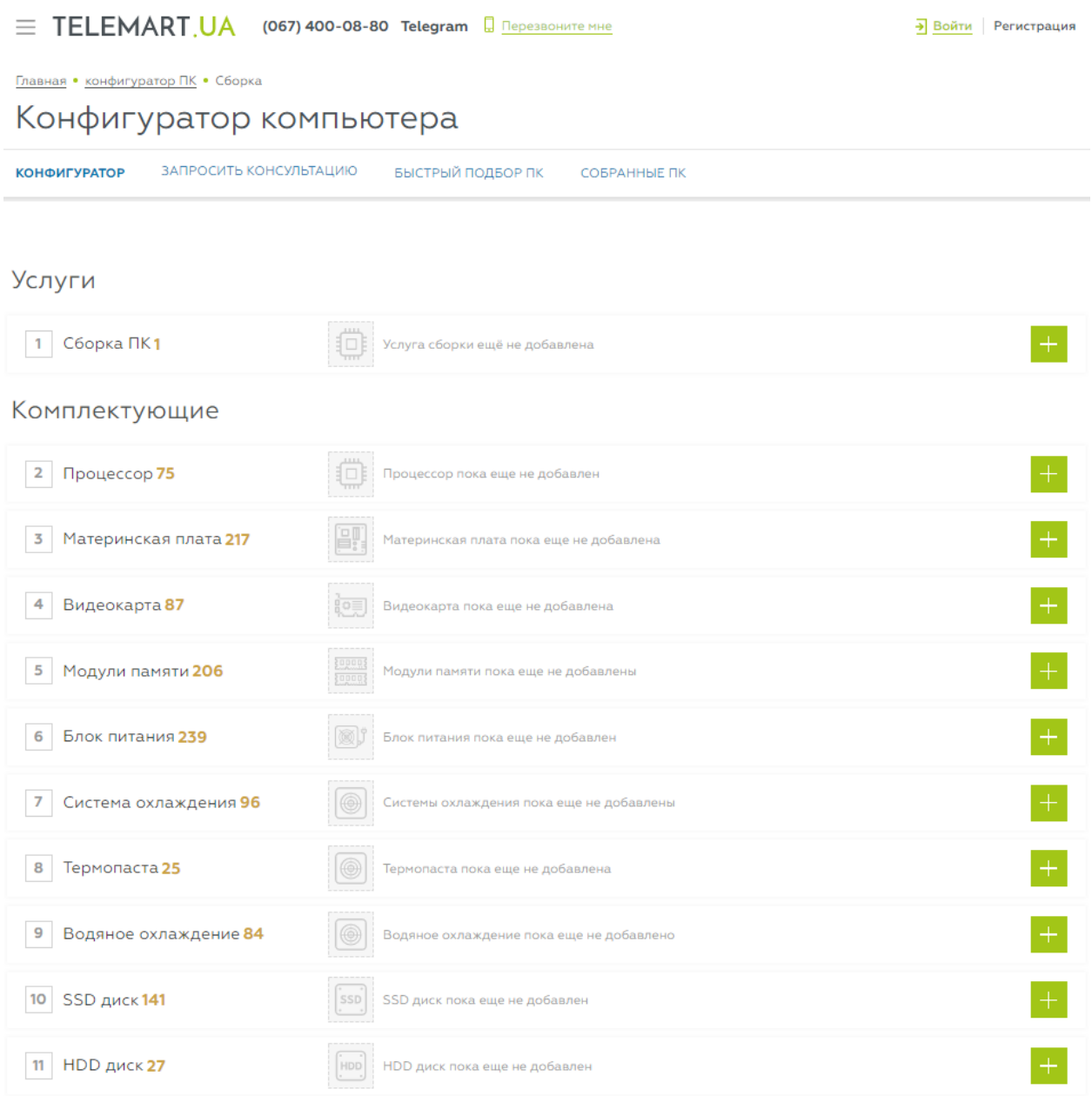


Рисунок 1.2.1 - Конфігуратор ПК на сайті компанії Telemart

Треба відмітити те, що розроблена ця система у веб-формі. То б то для користування достатньо зайти на сайт, але ця концепція має й свої мінуси. Далі йде порівняння веб-застосунку да десктопного застосунку.

Установка, оновлення:

Веб-додаток не вимагає встановлення, всі оновлення відбуваються на сервері та відразу доставляються користувачам – вам просто потрібно перезавантажити сторінку або вийти, а потім знову увійти у свій обліковий запис. Але іноді для його роботи потрібно буде встановити додаткові

бібліотеки або використовувати захищені мережеві протоколи. Програму для настільних комп’ютерів необхідно встановити на комп’ютер або мобільний пристрій і оновлюватиметься щоразу, коли виходить нова версія. Незважаючи на те, що більшість процесу автоматизовано, він все одно вимагає від користувачів часу та ресурсів пристрою. Вам також потрібно буде відстежувати версії на кожному комп’ютері, смартфоні та планшеті.

Опублікувати / розмістити:

Веб-додаток публікується на локальному або хмарному сервері, і там відбувається процес оновлення. У цьому випадку, навіть якщо рішення дуже просте, сервер у будь-якому випадку знадобиться. Зрештою, крім інтерфейсу, який користувачі запускають через браузер, потрібно десь розмістити бекенд. Додаток для настільних комп’ютерів потрібно буде встановити вручну на кожному пристрої. У компанії з великою кількістю робочих місць це може зайняти багато часу. Хороша новина полягає в тому, що якщо немає рішення клієнт/сервер, немає потреби вибирати сервер або шукати джерела для публікації.

Надійність

Продуктивність веб-додатка залежить не тільки від того, наскільки добре воно розроблено та характеристик пристрою користувача, а й від швидкості підключення до Інтернету та продуктивності віддаленого сервера. Настільна програма працює в автономному режимі, тому якість коду та стабільність обладнання, на якому працює цей код, мають першочергове значення. Але якщо потрібен зв'язок з сервером, то такі ж проблеми виникають і з «конкурентом».

Доступність

До веб-додатку можна отримати доступ з будь-якої точки світу, з будь-якого пристрою, а файли користувача завжди під рукою. Але тільки якщо у вас є підключення до Інтернету або можливість працювати в автономному режимі,

а також завантажувати та завантажувати дані. Програма для комп’ютера завжди доступна, але лише на пристрої, на якому вона встановлена. Щоб працювати з різними пристроями, вам потрібно встановити їх на кожному з них, а також визначити, де зберігати файли, щоб ви завжди могли отримати до них доступ.

Кроссплатформенність

Веб-додаток працює так само добре на будь-якому пристрої, будь то настільний комп’ютер, ноутбук, планшет або смартфон, тому що насправді воно не залежить від апаратного забезпечення чи операційної системи. Головне - правильний браузер. Як правило, Google Chrome, Mozilla Firefox, Apple Safari або браузер Windows (Microsoft Edge / Internet Explorer) сумісні з більшістю веб-клієнтів. Настільний додаток залежить від операційної системи, процесора, відеокарти та ряду інших параметрів. Потрібно враховувати нюанси кожного середовища (в тому числі «ловити» помилки), писати код з урахуванням можливих варіантів, наймати окремих розробників або навіть цілі команди для версій для різних операційних систем.

РОЗДІЛ 2 АНАЛІЗ ПРОГРАМНО-ТЕХНОЛОГІЧНИХ РІШЕНЬ РЕАЛІЗАЦІЇ

2.1 C# Основи та історія

C# - це об'єктно-орієнтована мова програмування. Він був створений у 1998-2002 роках командою інженерів Microsoft на чолі з Андерсом Хейлсбергом та Скоттом Вілтаумотом.

Мова належить до сімейства C-подібних мов. Синтаксис близький до Java та C++. Його особливості:

- статистичний лист;
- підтримується поліморфізм;
- підтримується завантаження оператора;
- існуюче делегування, атрибути, події, узагальнені типи та анонімні функції;

C# популярний через свою «простоту». Простота для сучасних програмістів та великих команд розробників, вони можуть створювати функціональні та ефективні програми у короткі терміни. Цьому сприяють нетипові мовні конструкції та специфічний синтаксис, які допомагають максимально органічно реалізувати задумані функції.

Ще однією важливою перевагою є популярність мови. Його просувають багато шанувальників C#. Це також позитивно позначиться збільшення кількості вакансій, пов'язаних з розвитком мови Microsoft. Незважаючи на зростаючу кількість програмістів, які вільно володіють C#, попит у галузі є.

Зрозумілий синтаксис C# значно спрощує розробку, а й інші важливі аспекти спільної роботи, такі як читання чужого коду. Це спрощує процес відновлення та виправлення помилок під час роботи над програмами у великих спільнотах.

Слід також відзначити нижню межу доступу. C# - популярна і дуже проста в освоєнні технологія. Через півроку можна намагатися розробляти та створювати повноцінні програми.

Якщо ви новачок у комп'ютерному програмуванні і не знаєте, яку мову програмування вивчити раніше, то можете бути трохи заплутаними. Є рекомендації досвідчених програмістів! Ви можете почати свій шлях програмування, вивчивши C#, одну з найстаріших, найнадійніших і найбільш широко використовуваних мов комп'ютерного програмування. Коли був створений C# і чому він такий популярний? Подивіться на коротку історію цієї популярної мови програмування.

Коли було створено C #? 2002. Правда, мові кодування C# лише 17 років. Якби це був чоловік, він міг би водити машину чи голосувати! Але в цій мові програмування немає нічого дитячого. Фактично, C# вважається зрілою мовою комп'ютерного програмування і широко використовується в усьому світі.

Існує багато мов комп'ютерного програмування, як і багато різних розмовних мов. Не помиляйтеся; вивчення мови комп'ютерного програмування так само складно, як і вивчення інших мов. C# унікальний, оскільки насправді є однією з найпростіших мов програмування для вивчення. Більшість програмістів вивчають кодування на C# та інших підтримуваних C# програмах. Ви можете розглядати C# як перший крок або базову навичку. Багато мов програмування тепер оновлені та змодельовані за поточними ітераціями тонко налаштованого C#.

Мова C # унікальна і в інших аспектах, тому ми зібрали все, що вам потрібно знати про C #.

Це мова програмування зі специфічним походженням. Коли було створено C #? У період з 2000 по 2002 рік Microsoft створила цю мову програмування для власного використання. Спочатку мова програмування C# була випущена разом із Microsoft Visual Studio 2002 у відповідь на мову програмування Java, що

використовується з Java Script. C# і Java були обидві мовами програмування на перших порах персонального комп’ютера, і обидві змагалися за визнання. Фактично, C# і Java багато років запозичили один у одного, перш ніж були перенаправлені.

Відтоді C# був затверджений Common Language Infrastructure як міжнародний стандарт комп’ютерного програмування та технічний стандарт комунікації комп’ютерної платформи. C# також прийнято Ecma International як поширена мова програмування комп'ютера. При використанні переважно в програмному забезпеченні Microsoft ви також можете побачити C#, який використовується для створення клієнтських програм і навіть інтернет-платформ.

Версії

Існує багато версій C#. Кожна версія C# зазнала оновлення мови програмування, яке продовжує урізноманітнити використання мови. Перша версія C# була розроблена як проста багатоцільова загальна мова програмування, і в порівнянні з поточним виглядом C# ця перша версія була майже в цілому м'якою. Можна подумати, що перша версія C# була схожа на BLT без бекону, салату чи помідорів.

Версія 2

Друга версія C# була випущена в 2005 році і почала відходити від об'єктно-орієнтованого. Це оновлення включає в себе можливість для програмістів використовувати ітератори, що в основному дозволяє програмістам бачити більшу картину даних за раз.

Версія 3

Третя версія C# була випущена в 2007 році і була сповнена нових функцій, таких як дерева виразів, методи розширення, а також вирази запитів і лямбда.

Саме з цією версією C# вона дійсно почала відрізнятися від Java, а C# став найсучаснішою мовою програмування.

Версія 4

Четверта версія C# була випущена в 2010 році і представила вбудовані типи інтерлопів і динамічні ключові слова, які вирішили проблеми і надали C# певної елегантності, якої не вистачало іншим мовам програмування в той час.

Версія 5

П’ята версія C# була випущена в 2012 році, зосереджена на асинхронному програмуванні. Оновлення C# asych і standby розглядаються як мовні функції, які підвищують простоту використання віддалених операційних кодів.

Версія 6

Шоста версія C # була випущена в 2013 році і зосереджена на перетворенні C # на ефективну мову програмування. У цій версії були представлені нові фільтри та ініціализатори, а також спільна команда команд, що спростило введення коду, ніж будь-коли.

Версія 7

Остаточна і поточна версія C # була випущена в 2017 році. На додаток до революційних функцій, які очищають і спрощують кодування, ця версія C# також розширила мову кодування, оскільки цю версію можна використовувати з .NET Core та операційними системами. для хмарних та інших потреб сховища.

2.2 Microsoft Visual Studio

Інтегроване середовище розробки (IDE) - це багатофункціональна програма, яка підтримує багато аспектів розробки програмного забезпечення. Інтегроване середовище розробки Visual Studio – це панель запуску для написання коду, налагодження та компіляції, а також публікації програм. Крім стандартного редактора і відладчика, доступних у більшості середовищ IDE,

Visual Studio включає компілятори, інструменти для заповнення коду, графічні дизайнери і багато інших функцій для поліпшення процесу програмування.

Деякі популярні функції Visual Studio, що підвищують ефективність розробки програмного забезпечення:

- Хвилясті лінії та швидкі рухи:

хвилясті лінії вказують на помилки або можливі проблеми з кодом під час введення. Ці візуальні символи допомагають вам вирішувати проблеми негайно, не чекаючи на створення або виконання помилок.

- Очистити код:

Можна одним кліком відформатувати код та застосувати налаштування стилю коду, угоди у файлі .editorconfig та/або виправлення, рекомендовані аналізаторами Roslyn.

- Рефакторинг:

Рефакторинг включає такі операції, як розумне перейменування змінних, видалення однієї або декількох рядків коду в новий метод і зміна порядку параметрів методу.

2.3 Кінцеві користувачі застосунком

Актуальність теми достатньо обґрунтувати тим, що персональний комп’ютер надає багато можливостей для розвитку, відпочинку, бізнесу та, звісно, роботи.

Якщо ми заглянемо у недалеке минуле, то побачимо, що більшість людей, що хотіли придбати персональний комп’ютер робили свій вибір на користь спеціалізованих на цій сфері комерційних компаній, що з одного боку мали й великий та якісний досвід, супроводжали цикл від абстракцій на основі характеристик ПК, що вони хотіли отримати, аж до кінцевого результату у

вигляді вже зібраного рішення, а з другої – значну переоцінку за свої навички. То б то виходило так, що компанії, які розумілися на, здавалося б, доволі простих речах, могли отримувати несумірний результат у еквіваленті грошового ресурсу.

Першочерговою проблемою, на яку треба звернути увагу – це різниця кінцевої суми витрачених коштів. Користуючись додатком є можливість збільшити цю різницю до цифр, які будуть доволі впливовими, тим паче у процентному співвідношенні.

Кількість і різноманітність користувачів комп'ютерів швидко зроста. До них належать менеджери, бухгалтери, інженери, будівельники будинку, вчителі, науковці, медичні працівники, страховики, продавці та адміністративні помічники. Більшість із цих людей працюють над завданнями, які швидко змінюються на щорічній, щомісячній і навіть щоденній основі. Отже, їхні потреби в програмуванні різноманітні, складні та часто змінюються. Професійні розробники програмного забезпечення можуть бути не в змозі безпосередньо задовольнити всі ці потреби через їх обмежені знання предметної області та через те, що їхні процеси розробки дуже повільні.

Розробка для кінцевих користувачів (EUD) може допомогти вирішити цю проблему. EUD — це набір методів, прийомів та інструментів, які дозволяють користувачам програмних систем, які працюють як непрофесійні розробники програмного забезпечення, створювати, змінювати або розширювати програмний артефакт у будь-який момент. Зокрема, EUD дозволяє кінцевим користувачам проектувати або налаштовувати функціональність користувальницького інтерфейсу та програмного забезпечення. Це цінно, оскільки кінцеві користувачі знають свій контекст і потреби краще за інших, і вони часто знають про зміни у своїх доменах у реальному часі. Через EUD кінцеві користувачі можуть налаштувати програмне забезпечення ближче до своїх вимог, ніж це можливо без EUD. Крім того, оскільки кінцевих користувачів у 30-1 раз більше, ніж професійних розробників програмного забезпечення, EUD

«розширює» діяльність з розробки програмного забезпечення, дозволяючи брати участь більшій частині людей.

Однак EUD радикально відрізняється від традиційної розробки програмного забезпечення, і спроби підтримати EUD шляхом наслідування традиційних підходів часто недостатні для досягнення успішних результатів. Кінцеві користувачі, як правило, не володіють експертними мовами програмування, формальними процесами розробки або навчанням моделювати та діаграмні символи. Крім того, у кінцевих користувачів часто не вистачає часу або мотивації для вивчення цих традиційних методів, оскільки кінцеві користувачі зазвичай ставлять перед собою короткострокові або середньострокові цілі замість створення довгострокового програмного забезпечення, яке генерує постійний потік доходу. Вони пишуть код. Отже, підтримка EUD вимагає надання відповідних інструментів, соціальних структур і процесів розробки, які є дуже зручними, швидко навчаються та легко інтегруються в практику домену.

EUD сумісний з двома подібними концепціями: програмування кінцевого користувача та розробка програмного забезпечення для кінцевого користувача. Програмування кінцевих користувачів (EUP) дозволяє кінцевим користувачам створювати власні програми. Ця невелика колекція EUD є найзрілішою з точки зору досліджень і практики, тому наступну частину цієї статті ми зосередимо на цій частині EUD. Різниця між EUP і EUD полягає в тому, що методи, методи та інструменти EUD охоплюють весь життєвий цикл розробки програмного забезпечення, включаючи не тільки фазу «створення», але також модифікацію та розширення програмного забезпечення.

2.4 Вибір платформи реалізації, опис аналогів

WPF

Технологія стала популярною на початку 2000-х для реалізації додатків. Це технологічна структура, яка побудована в рамках .NET на основі створення графічної частини настільного додатка. WPF був одним із стеків .NET з початку 2000-х років і завоював серця багатьох програмістів. Однією з найважливіших особливостей WPF є його здатність комбінувати різні елементи графічного інтерфейсу. Вам необхідно створити програмний інтерфейс для кінцевого користувача. Нижче представлена схема WPF у .NET Framework. Нижче (рис. 2.4.1) наведено графік порівняння продуктивності Winforms з WPF.

	Windows (ms)		WPF (ms)
Object Creation	309	Parsing of XAML	453
Comctrl32/User32 Control Creation	1823	Layout Formation	3920
Create Window	1308	Window Layout	0.14
Other Clr Calls	985		
Render	1217	Render	145
Total	5642	Total	4518

Рисунок 2.4.1 - графік порівняння продуктивності Winforms з WPF

1) WPF використовує збережене відтворення, а Windows Forms використовує швидке відтворення.

2) Відтворення WPF виконується в потоці композиції, відмінному від потоку інтерфейсу користувача.

3) У програмах WPF використовується GPU, що прискорює процес візуалізації.

Вищевказані тести проводилися на Windows 10 і процесорі I3, результати можуть відрізнятися більш старих операційних системах.

Swing

Ще один аналог – поворотна рама. Найбільш важливим застосуванням Swing є створення візуально зручнішого графічного інтерфейсу для настільних додатків. Ця технологія також має функцію імітації дизайну. Але додатковою особливістю технології, що полегшує використання рішень, є можливість

поставки варіативних елементів компонентів графічного інтерфейсу. Крім того, він є мультиплатформним, тому рішення на основі Swing будуть рішеннями, які можна розмістити на іншій базовій платформі. Це технологія, що базується на мові програмування Java, призначена для реалізації настільної програми. Ієрархія Swing API Java показана малюнку 2.4.2 нижче.

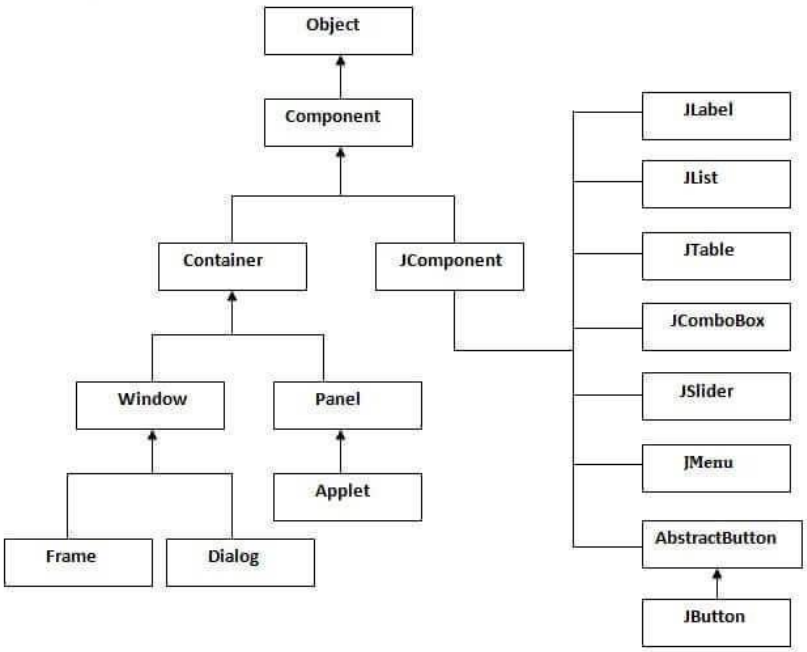


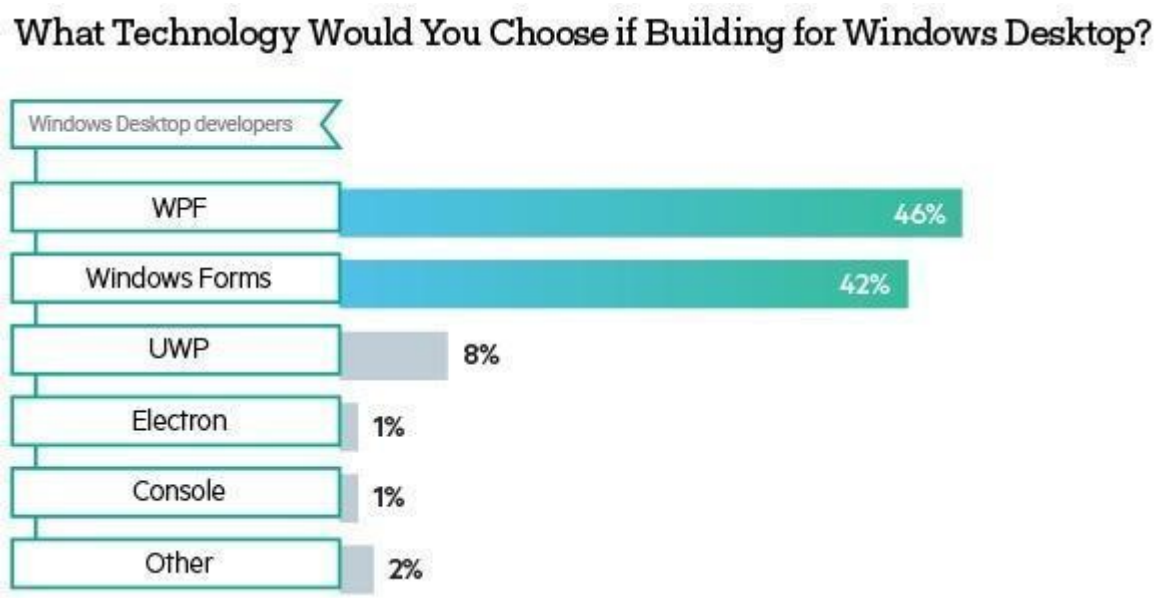
Рисунок 2.4.2 - Ієрархія Swing API Java показана малюнку

Universal Windows Platform (UWP)

UWP також є однією з цінних IT-платформ для розробки настільних програм. Це також технологія, пов'язана із широким використанням .NET. Але це пов'язано з тим, що він дозволяє розробникам продавати мультиплатформні додатки.

Надалі програма зможе працювати на мобільних пристроях і все без винятку навіть на консольних пристроях. Це фреймворк, який значно покращив функціональність VS. З погляду масштабу ця технологія має великий потенціал. При реалізації настільної програми платформа дозволяє масштабувати роботу на більшості пристроїв у визначеній мережі пристроїв.

Зокрема, UWP пропонує підходи та програми, які можуть працювати практично в будь-якій операційній системі Microsoft. Ваше програмне забезпечення може реалізувати потенціал локально на кількох різних пристроях. Загалом це пов'язано з універсальним алгоритмом реалізації програм у Windows. Нижче наведено статистичні дані, які відповідають питання вибору більшості розробників (рис. 1.4.3).



Риуснок 2.4.3 - статистичні дані вибору інструменту для написання десктоп додатку

Сасао

Сосоа – аналог цієї теми розробки, але вже під MacOS. Взявши за основу цю технологію можна не перевизначати її, а навіть анімувати як невід'ємну частину інтерфейсу. Усі без винятку елементи інструментів реалізації через систему Сосоа ліцензовані Apple та підтримуються технологіями. Сосоа – це технологічна платформа для реалізації в операційній системі. Він об'єктно-орієнтований та використовується для створення графічних інтерфейсів кінцевих користувачів для macOS та iOS. Суть у тому, що ця технологія також забезпечує кроссплатформенність. А можливість розвитку цієї технології не тільки додає більше функціональності простору взаємодії з користувачем, а й відкриває можливість додати до інтерфейсу «більше фарб».

Electron JS

Electron JS – це великомасштабна мультиплатформна технологія, розроблена GitHub. Його розподіл за аудиторією розробників показано на статистичному графіку на рис. 1.4.3. Цей фреймворк ґрунтується на Node.JS. Є багато комерційних структур, таких як Oracle та десятки студій, які створюють свої продукти на основі цієї технології. Головною його особливістю є реалізація мультиплатформних настільних програм.

На основі цієї концепції технологія здатна надати інструмент для роботи з дуже високорівневими структурами блоків процесу розробки програмного забезпечення.

2.5 Види інтерфейсів

На рисунку нижче показано складові інтерфейсу користувача (Рисунок 2.5.1)

Рисунок 2.5.1 - Складові інтерфейсу користувача

Інтерфейс прямої обробки — це назва загального класу інтерфейсів користувача, які дозволяють користувачам маніпулювати представленими ним об'єктами за допомогою дій, які, принаймні, відповідають фізичному світу.

Графічні інтерфейси користувача (GUI) отримують дані від таких пристроїв, як комп'ютерні клавіатури і миші, і забезпечують графічний висновок на моніторі комп'ютера. У дизайні графічного інтерфейсу широко використовуються як мінімум два різних принципи: об'єктно-орієнтовані інтерфейси (OOUI) і інтерфейси, орієнтовані на додатки.

Веб-інтерфейси користувача або веб-інтерфейси користувача (WUI), які забезпечують введення та виведення шляхом створення веб-сторінок, що передаються через Інтернет та переглядаються користувачем за допомогою

програми веб-браузера. Нові агрегати використовують: PHP, Java, JavaScript, AJAX, Apache Flex, NET Framework або аналогічні технології, що забезпечують керування в реальному часі в окремій програмі, що усуває необхідність відновлення традиційного веб-браузера на основі HTML. Адміністративні веб-інтерфейси для веб-серверів, серверів та мережевих комп'ютерів зазвичай називають панелями управління.

Сенсорні екрани – це дисплеї, до яких можна отримати доступ пальцем або стилусом. Вони збільшуються в кількості мобільних пристроїв і використовуються в багатьох комерційних транспортних засобах, промислових процесах та машинах, пристроях самообслуговування тощо.

Інтерфейси командного рядка, в яких користувач надає вхідний сигнал, звертаючись до командного рядка за допомогою клавіатури комп'ютера, а система забезпечує висновок, відображаючи текст на моніторі комп'ютера. Використовується програмістами та системними адміністраторами в інженерному та науковому середовищі, а також технічно просунутими користувачами персональних комп'ютерів.

Сенсорний інтерфейс користувача — це графічний інтерфейс, який використовує сенсорну панель або сенсорний екран як комбінований пристрій введення та виведення. Вони доповнюють або замінюють інші форми виведення методами тактичного зворотного зв'язку. Використовується у комп'ютерних симуляторах та ін.

Апаратні інтерфейси - це фізичні просторові інтерфейси, які присутні в продуктах повсякденного життя, від тостерів до панелей приладів автомобілів або кабін літаків. Зазвичай вони є комбінацією кнопок, ручок, повзунків, клавіш і сенсорних екранів.

Ретельні інтерфейси користувача. Управляйте увагою користувача такими способами: дозволяючи затримку користувача, типи попереджень та рівень деталізації повідомлень, що надаються користувачеві.

Пакетні інтерфейси - це неінтерактивні інтерфейси користувача, в яких користувач заздалегідь встановлює всі деталі пакетної задачі для пакетної обробки і отримує висновок після завершення всієї обробки. Після початку обробки комп'ютер не вимагає додаткового введення.

Діалогові інтерфейси дозволяють користувачам керувати своїми комп'ютерами за допомогою простого тексту (наприклад, текстових повідомлень або чатів) або голосових команд замість графіки. Ці інтерфейси найчастіше імітують розмови людей.

Агенти діалогового інтерфейсу намагаються персоналізувати комп'ютерний інтерфейс у вигляді анімованої людини, робота або іншого персонажа (наприклад, скріпки Microsoft Clippy) та представляти взаємодії у формі діалогу.

Інтерактивні інтерфейси — це графічні інтерфейси, основне завдання яких — не орієнтуватися на інструкції, а виходити за межі.

Для створення застосунку, одним з перших кроків – це зазначення цілей рішення. При чіткому розумінні результату, якого ми намагаємося досягнути, буде чітке розуміння подальших кроків, що будуть являти собою макро та мікро модулі. Одним з перших таких кроків буде вирішення набору інструментів, що буду максимально оптимальним у відношенні час\якість.

.NET Framework – це платформа для розробки. Була створена компанією Microsoft у початку 2000-х, на Community Version вийшла попередньо як бета-версія. Цей фреймворк широко використовується для написання веб, десктопних та мобільних застосунків.

Сама по собі структура містить велику кількість бібліотек классів(FCL). Також застосунки, що були створені на цій платформі, виконуються у середі, що називається Common Language Runtime. Common Language Runtime (Рисунок 2.5.2)

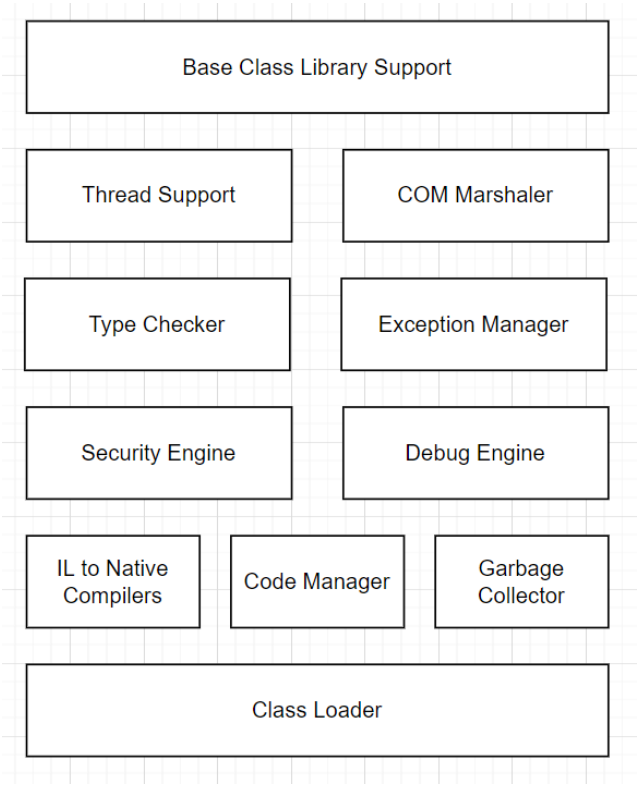


Рисунок 2.5.2 - Common Language Runtime

Загалом .NET Framework підтримує більше 40 мов програмування, наприклад C#, VB.NET, Perl, ML, Pascal, Python тощо.

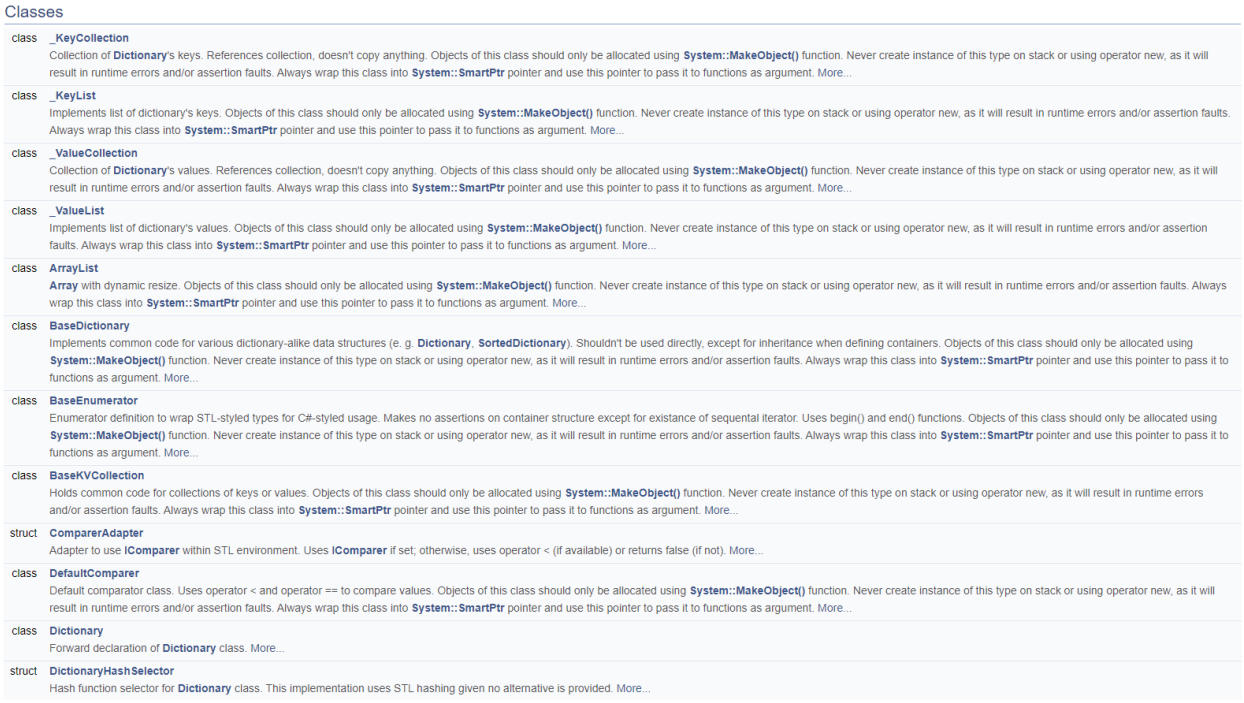
2.6 Використані бібліотеки

У проекті були використані широко розповсюджені бібліотеки, такі як:

- System.Data.SQLite (SQLite);
- System.Linq;
- System.Windows.Forms;
- System.IO;
- System.Drawing;
- System.Diagnostics;
- System.Collections.Generic;
- System.Text;
- System.ComponentModel;

System.Collections.Generic Namespace

Він включає інтерфейси і класи (Рисунок 2.6.1), що визначають загальні пакети, що дозволяє користувачам створювати важко зароблені пакети, що забезпечують кращу безпеку і продуктивність, ніж пакети без загального доступу.



Classes

class	**_KeyCollection** Collection of **Dictionary**'s keys. References collection, doesn't copy anything. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**_KeyList** Implements list of dictionary's keys. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**_ValueCollection** Collection of **Dictionary**'s values. References collection, doesn't copy anything. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**_ValueList** Implements list of dictionary's values. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**ArrayList** **Array** with dynamic resize. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**BaseDictionary** Implements common code for various dictionary-alike data structures (e. g. **Dictionary**, **SortedDictionary**). Shouldn't be used directly, except for inheritance when defining containers. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**BaseEnumerator** Enumerator definition to wrap STL-styled types for C#-styled usage. Makes no assertions on container structure except for existance of sequental iterator. Uses begin() and end() functions. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**BaseKVCollection** Holds common code for collections of keys or values. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
struct	**ComparerAdapter** Adapter to use **IComparer** within STL environment. Uses **IComparer** if set; otherwise, uses operator < (if available) or returns false (if not). More...
class	**DefaultComparer** Default comparator class. Uses operator < and operator == to compare values. Objects of this class should only be allocated using **System::MakeObject()** function. Never create instance of this type on stack or using operator new, as it will result in runtime errors and/or assertion faults. Always wrap this class into **System::SmartPtr** pointer and use this pointer to pass it to functions as argument. More...
class	**Dictionary** Forward declaration of **Dictionary** class. More...
struct	**DictionaryHashSelector** Hash function selector for **Dictionary** class. This implementation uses STL hashing given no alternative is provided. More...

Рисунок 2.6.1 - інтерфейси і класи System.Collections.Generic Namespace

У наступному переліку знаходяться функції, що є довідником з простору імен:

Рисунок 2.6.2 - довідник з простору імен

System.Linq Namespace

Надає класи та інтерфейси, що підтримують запити, що використовують інтегрований до мови запит (LINQ).

System.Text Namespace

ASCII та Unicode включають класи, що представляють кодування символів; абстрактні базові класи для перетворення блоків символів та блоків байтів; та допоміжний клас, який керує об'єктами String та форматує їх без створення проміжних копій String.

System.Text Namespace містить класи, які представляють різні кодування символів і перетворення, а також надає допоміжні класи для маніпулювання та форматування об'єктів String. Клас StringBuilder можна використовувати в поєднанні з класом String для керування рядком. Цей клас корисний у ситуаціях, коли нам потрібно змінити вміст рядка без створення нового рядка — додавання, заміна чи видалення символів. Ми використовуємо методи Insert, Replace і Remove класу StringBuilder для виконання необхідних дій. Доступ до символу в рядку забезпечується властивістю Chars, що дозволяє нам маніпулювати рядками для кожного символу.

SQLite

Це невелика, швидка та вбудована база даних SQL, що базується на файловій системі з відкритим вихідним кодом. У ньому немає окремого серверного компонента, як і традиційних базах даних. Читає та записує дані безпосередньо у файли на диску. База даних SQLite інтегрована з програмою, яка входить до бази даних. Формат файлу бази даних SQLite є кросплатформним і може переміщатися між 32-розрядними та 64-розрядними файловими системами. Завдяки безсерверній архітектурі, розробникам не потрібно встановлювати SQLite перед його використанням. Усі транзакції SQLite повністю відповідають вимогам ACID; це означає, що всі дослідження та розробки є атомарними, послідовними, ізольованими та довгостроковими. Вихідний код SQLite відкритий для всіх і безкоштовний для будь-яких цілей, комерційних чи особистих.

System.Threading.Tasks Namespace

В епоху багатоядерних машин, які дозволяли паралельно виконувати кілька процесів, стандартних функцій .NET було недостатньо. Таким чином, бібліотека паралельних завдань TPL (Task Parallel Library) була додана до платформи .NET, основні функції якої розташовані в полі імені System.Threading.Tasks. Ця бібліотека дозволяє легко працювати з багатопроцесорними багатоядерними системами. Це також спрощує процес створення нової пряжі. Тому, хоча стандартні інструменти та клас Thread все ще широко використовуються, рекомендується використовувати TPL та його класи для створення багатопоточних програм. Бібліотека TPL заснована на концепції завдань, кожна з яких описує окрему довготривалу операцію. У бібліотеці класів .NET функція представлена спеціальним класом, який є класом Task, розташованим у полі імені System.Threading.Tasks. Цей клас описує одну функцію, яка виконується асинхронно на одній із ниток у пулі ниток. Хоча він також може працювати синхронно з поточною заготовкою.

Бібліотека, що пропонує типи, що спрощують написання паралельного та асинхронного коду. Основними типами є Task <TResult>, що є асинхронною операцією, яка може очікувати і скасовувати. Клас TaskFactory надає статичні методи створення та виконання завдань, а клас TaskScheduler надає основну інфраструктуру планування теми.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ ЗАСТОСУНКУ СТВОРЕННЯ КОНФІГУРАЦІЇ ЗА ЗАДАНИМИ ПАРАМЕТРАМИ

3.1 Розробка програмного коду

Зараз розберемо алгоритми, що відповідають за функціонал застосунку.

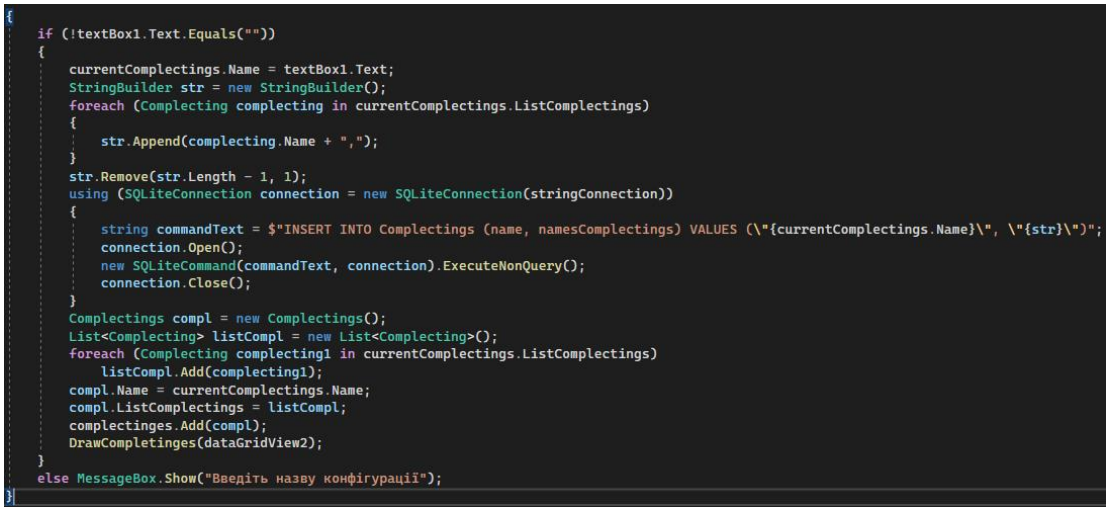
На стартовому етапі розробки проводиться ініціалізація методу для підтримки конструктор. За допомогою властивості інкапсуляції через методи get та sat відкриваємо з’єднання за допомогою знайдених параметрів та коли з'єднання з базою даних закрито, всі команди, пов'язані з цим з'єднанням, автоматично скидаються. Отже основною задачею цієї функції є ініціалізація локальних змінних, що за допомогою методів, які визначають чи рівні значення цього екземпляра чи об'єкта для запису найменувань. (Рисунок 3.1.1)

```
InitializeComponent();
comboBox1.SelectedIndex = 0;
using (SQLiteConnection connetion = new SQLiteConnection(StringConnection)) {

    string commandText = "SELECT * FROM Complecting";
    connetion.Open(); // открыть command
    new SQLiteDataAdapter(commandText, connetion).Fill(dataSet);
    connetion.Close(); // закрыть соединение
    foreach (DataRow row in dataSet.Tables[0].Rows) {
        object[] items = row.ItemArray;
        string[] notCompletings = ((string)items[5]).Split(',');
        allComplectings.Add( new Complecting((string)items[1], (Types)int.Parse(string.Format("{0}", items[2])),
           (string)items[3], double.Parse(string.Format("{0}", items[4])), notCompletings.ToList()));
    }
    string commandText1 = "SELECT * FROM Complectings";
    connetion.Open(); // открыть command
    dataSet.Tables.Clear();
    new SQLiteDataAdapter(commandText1, connetion).Fill(dataSet);
    connetion.Close(); // закрыть соединение
    foreach (DataRow row in dataSet.Tables[0].Rows)
    {
        object[] items = row.ItemArray;
        string[] tempNames = ((string)items[1]).Split(',');
        List<Complecting> tempComplectings = new List<Complecting>();
        for (int i = 0; i < tempNames.Length; i++)
        {
            foreach(Complecting compl in allComplectings)
            {
                if (tempNames[i].Equals(compl.Name)) tempComplectings.Add(compl);
            }
        }
        Complectinges.Add(new Complectings(tempComplectings, (string)items[2]));
    }
    suitableComplecting = allComplectings;
}
```

Рисунок 3.1.1 - ініціалізація локальних змінних, які визначають чи рівні значення цього екземпляра чи об'єкта для запису найменувань

Наступний функціонал, зазначений на скріншоті (Рисунок 3.1.2) є частиною класу, що робить перевірку на заповнення поля назви конфігурації за допомогою методів класів, що використовують параметри за значеннями. Також перевіряє на відсутність деталей у конфігурації та за допомогою дій, що виконуються при отриманні значень (метод get), та дій, що виконуються при встановленні значень (метод set). Отже основна задача цієї функції - надавати змогу зберегти зібрану комплектацію.



```
{
    if (!textBox1.Text.Equals(""))
    {
        currentComplectings.Name = textBox1.Text;
        StringBuilder str = new StringBuilder();
        foreach (Complecting complecting in currentComplectings.ListComplectings)
        {
            str.Append(complecting.Name + ",");
        }
        str.Remove(str.Length - 1, 1);
        using (SQLiteConnection connection = new SQLiteConnection(stringConnection))
        {
            string commandText = $"INSERT INTO Complectings (name, namesComplectings) VALUES (\"{currentComplectings.Name}\", \"{str}\")";
            connection.Open();
            new SQLiteCommand(commandText, connection).ExecuteNonQuery();
            connection.Close();
        }
        Complectings compl = new Complectings();
        List<Complecting> listCompl = new List<Complecting>();
        foreach (Complecting complecting1 in currentComplectings.ListComplectings)
            listCompl.Add(complecting1);
        compl.Name = currentComplectings.Name;
        compl.ListComplectings = listCompl;
        complectinges.Add(compl);
        DrawCompletinges(dataGridView2);
    }
    else MessageBox.Show("Введіть назву конфігурації");
}
```

Рисунок 3.1.2 - Надання змоги зберегти зібрану комплектацію

На наступних скріншотах (Рисунок 3.1.3 та 3.1.4) реалізований приклад функціоналу, що буде видаляти комплектації зі списку комплекцій (Рисунок 2.1.3). Для цього реалізований ітеративний прохід по списку компектуючих (ListComplecting[i]). Як наслідок, рішення має пришвидшення обробки та оптимізації рішення.

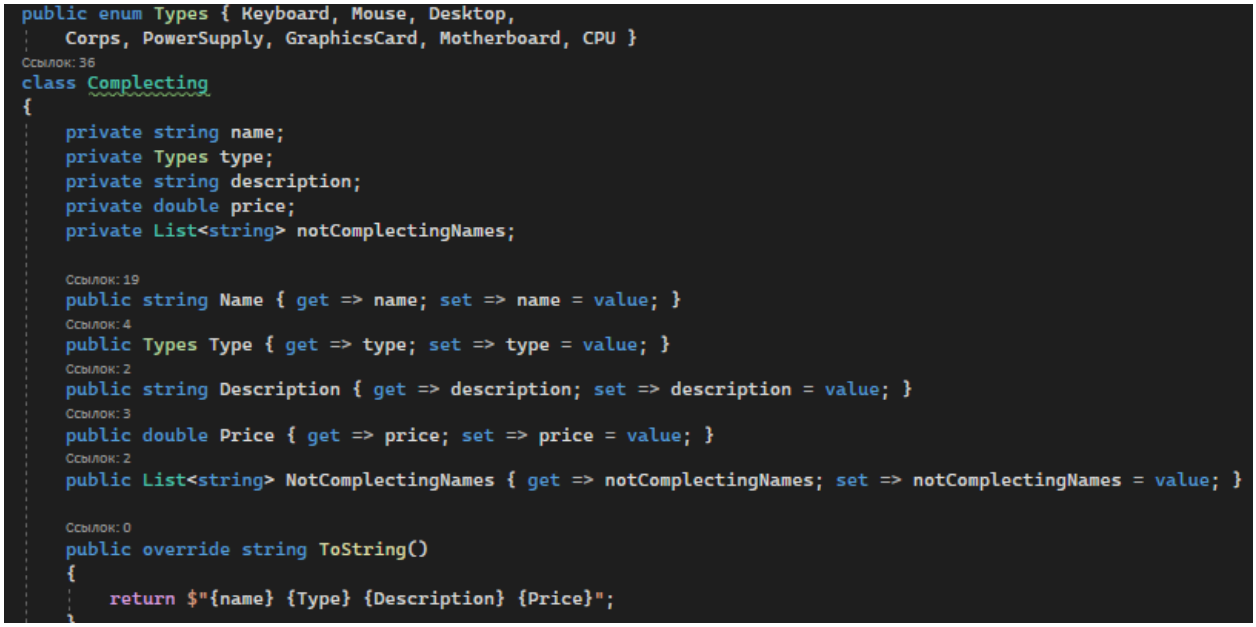
```
private void button6_Click(object sender, EventArgs e)
{
    List<Complecting> complectings = new List<Complecting>();
    for(int i = 0; i < dataGridView3.Rows.Count; i++)
    {
        foreach(Complecting complecting in allComplectings)
        {
            if(complecting.Name.Equals(dataGridView3.Rows[i].Cells[1].Value.ToString()))
            {
                complectings.Add(complecting);
            }
        }
    }if (complectinges.Count != 0)
    {
        bool equal = true;
        if (currentComplectings.ListComplectings.Count == complectings.Count)
        {
            for (int i = 0; i < currentComplectings.ListComplectings.Count; i++)
            {
                if (!currentComplectings.ListComplectings[i].Equals(complectings[i])) equal = false;
            }
        }
        else { equal = false; };
        if (equal)
        {
            currentComplectings.ListComplectings.Remove(complectings[dataGridView3.SelectedCells[0].RowIndex]);
            DrawOnDGV(dataGridView3, currentComplectings.ListComplectings);
            totalPriceL.Text = currentComplectings.AllCost().ToString();
            resetSuitable(currentComplectings);
        }
```

Рисунок 3.1.3 - Функціонал для видалення комплектації зі списку комплекцій

```
        else
        {
            StringBuilder beforeRemove = new StringBuilder();
            foreach(Complecting complecting in complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings)
            {
                beforeRemove.Append(complecting.Name + ",");
            }
            beforeRemove.Remove(beforeRemove.Length - 1, 1);
            complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings.RemoveAt(dataGridView3.SelectedCells[0].RowIndex);
            DrawOnDGV(dataGridView3, complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings);
            totalPriceL.Text = complectinges[dataGridView2.SelectedCells[0].RowIndex].AllCost().ToString();
            StringBuilder afterRemove = new StringBuilder();
            foreach (Complecting complecting in complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings)
            {
                afterRemove.Append(complecting.Name + ",");
            }
            afterRemove.Remove(afterRemove.Length - 1, 1);
            using (SQLiteConnection connection = new SQLiteConnection(stringConnection))
            {
                string commandText = $"UPDATE Complectings SET namesComplectings = \"{afterRemove}\" " +
                    $"WHERE namesComplectings = \"{beforeRemove}\"";
                connection.Open();
                new SQLiteCommand(commandText, connection).ExecuteNonQuery();
                connection.Close();
            }
        }
    }
    else
    {
        MessageBox.Show("В конфігурації немає елементів");
    }
```

Рисунок 3.1.4 - Функціонал для видалення комплектації зі списку комплекцій

На наступному скріні (Рисунок 3.1.5) коду реалізовано оголошення констант, визначення класу Complecting, закриті та відкриті поля. Далі методи get\set. Для того щоб продовжувати притримуватися концепції реалізації алгоритм наступних кроків це – перевизначення стандартних реалізацій toString для повернення значень.



```
public enum Types { Keyboard, Mouse, Desktop,
    Corps, PowerSupply, GraphicsCard, Motherboard, CPU }
Ссылок: 36
class Complecting
{
    private string name;
    private Types type;
    private string description;
    private double price;
    private List<string> notComplectingNames;

    Ссылок: 19
    public string Name { get => name; set => name = value; }
    Ссылок: 4
    public Types Type { get => type; set => type = value; }
    Ссылок: 2
    public string Description { get => description; set => description = value; }
    Ссылок: 3
    public double Price { get => price; set => price = value; }
    Ссылок: 2
    public List<string> NotComplectingNames { get => notComplectingNames; set => notComplectingNames = value; }

    Ссылок: 0
    public override string ToString()
    {
        return $"{name} {Type} {Description} {Price}";
    }
```

Рисунок 3.1.5 - Перевизначення стандартних реалізацій toString для повернення значень

Далі йде Equals, яка приймає екземпляр для повернення булевого значення, яке залежить від результату порівнянь створених та перевизначених змінних. Конструктор, у якому визначаються початкові значення та ініціалізуємо поля. (Рисунок 3.1.6)

```
//desrapt
ссылка: 1
public override bool Equals(object obj)
{
    return obj is Complecting complecting &&
           name == complecting.name &&
           type == complecting.type &&
           description == complecting.description &&
           price == complecting.price &&
           EqualityComparer<List<string>>.Default.Equals(notComplectingNames, complecting.notComplectingNames);
}

ссылка: 1
public Complecting(string name, Types type, string description, double price, List<string> notComplectingNames)
{
    this.name = name;
    this.type = type;
    this.description = description;
    this.price = price;
    this.NotComplectingNames = notComplectingNames;
}
```

Рисунок 3.1.6 - Конструктор, у якому визначаються початкові значення та ініціалізуємо поля

3.2 Вигляд функціоналу та взаємодія з ним з точки зору користувача

Проект реалізується за допомогою Visual Studio 2022 Community на мові C# з використанням інтерфейсу програмування додатків Windows Forms.

Windows Forms — це технологія інтерфейсу для .NET, що представляє собою набір керованих бібліотек, які спрощують виконання стандартних завдань, таких як читання з файлової системи і запис в неї. За допомогою середовища розробки, наприклад Visual Studio, можна створювати інтелектуальні клієнтські програми Windows Forms, які відображають інформацію, запитують введення користувача і взаємодіють з віддаленими комп'ютерами по мережі.

Рисунок 3.2.1 - Блок для перегляду та обрання комплектуючих

Блок інформації щодо обраної комплектуючої, її ціни, а також кнопок управління комплектацій(Рисунок 3.2.2):

Рисунок 3.2.2 - Блок інформації

У доповненні вказані властивості, характеристики та ціна комплектуючої(Рисунок 3.2.3):

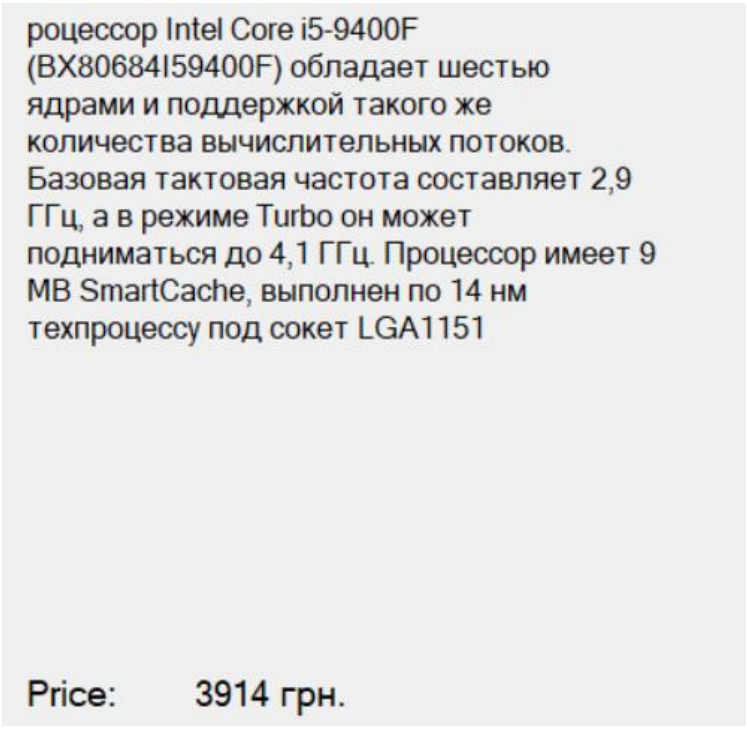


Рисунок 3.2.3 - Характеристики та ціна комплектуючої

На наступному рисунку зображений мінімальний функціонал для взаємодії (Рисунок 3.2.4):

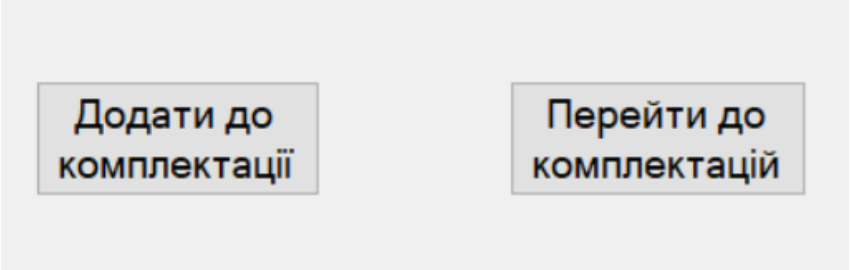


Рисунок 3.2.4 - Функціонал для взаємодії

Далі (Рисунок 3.2.5) знаходиться блок управління поточною конфігурацією, що вміщує в собі слідуючи пункти: «Повернутися до поточної конфігурації», «Зберегти конфігурацію», «Видалити комплектуючу», «Повернутися до вибору комплектуючих» та поле для вводу Найменування збереженої конфігурації.

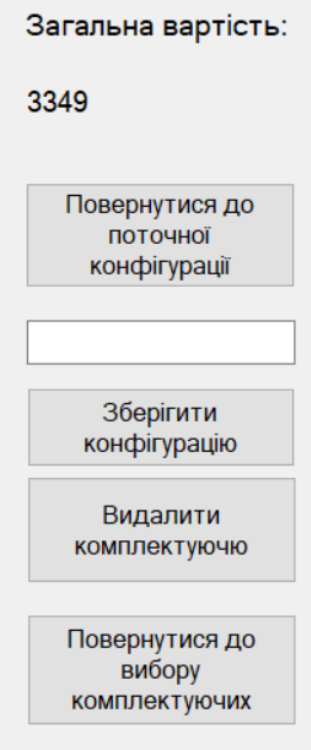


Рисунок 3.2.5 - Блок управління поточною конфігурацією

При переході на сторінку створених конфігурацій був створений блок перегляду збережених конфігурацій, що містять самі назви конфігурацій та їх заповнення (Рисунок 3.2.5):

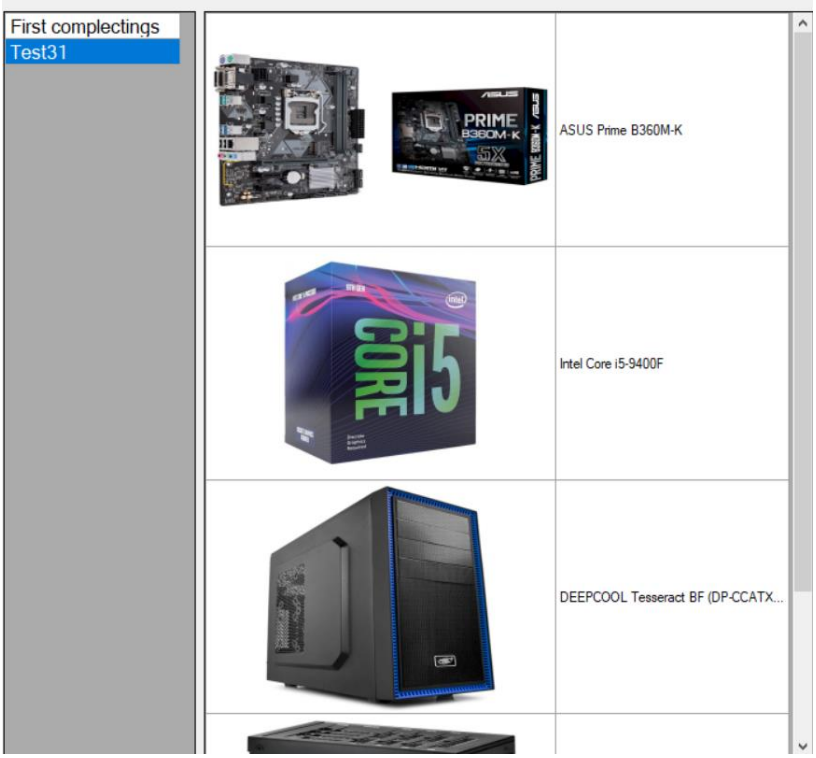


Рисунок 3.2.5 - блок перегляду збережених конфігурацій

3.3 Технічні вимоги

3.3.1 Вимоги до складу програмного продукту

1) Функція додавання та видалення комплектуючих із конфігуратора;

2) Функція зберігання готової конфігурації;

3) Функція перегляду готових конфігурацій;

3.3.2 Вимоги до надійності

1) При несправності під час користування програмою, збій не повинен приводити до втрати даних;

2) При несправності додатку, яка привела до її відключення, програма має зберігати конфігурацію, яка змінювалась в реальному часі;

3.3.3 Умови експлуатації

Для роботи з додатком потрібен PC на базі ОС Windows XP, 7, 8, 10.

Інтерфейс має бути інтуїтивно зрозумілим для будь якого користувача.

3.3.4 Вимоги до складу і параметрів технічних засобів

Для використання продукту потрібен PC або ноутбук на базі ОС Windows з такими мінімальними характеристиками:

- ОС Windows XP/7/8/10
- Процесор: Intel Celeron / AMD Athlon
- Оперативна пам’ять: не менше 2 ГБ
- Пам’ять на диску: не менше 10 ГБ
- Відеокарта: GeForce GTX 760 1ГБ / Radeon R7 260X 2ГБ

ВИСНОВКИ

У результаті проходження виробничої практики було вивчено функціональні можливості сучасної системи створення додатків за допомогою новітніх технологій розробки. Були вивчені теоретичні основи бібліотек, які є збірник підпрограм або об'єктів, що використовуються для розробки програмного забезпечення, а також придбані практичні навички роботи з ними. Було засвоєно технологію роботи на платформі .NET Framework із застосуванням комплексних підходів розробки, що дозволили масштабувати як теоретичні знання так і практичні навички.

Під час розробки застосунку були пройдені всі етапи: від бібліотек до створення концепцій взаємодії з інтерфейсами. Також ми ознайомилися з інструментами, що дозволяють не підіймаючи серверні активи розробити БД, яка подалі є масштабованою та ефективною.

На заданому індивідуальному завданні ми мали змогу попрактикуватися у сфері повноцінної розробки та використати знання, які ми накопили на навчанні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Темпи зростання інтернет-користувачів. Ukrinform. Режим доступу : https://www.ukrinform.ru/rubric-technology/2797153-v-ukraine-kolicestvo-internetpolzovatelej-vozroslo-do-23-millionov.html

2. Microsoft .NET Framework і чому вона встановлена на моєму комп'ютері? .ua.phhsnews : https://ua.phhsnews.com/articles/howto/what-is-the-microsoft-net-framework-and-why-is-it-installed-on-my-pc.html

3. Руководство по классическим приложениям (Windows Forms .NET). docs.microsoft.com :

https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-6.0

4. Інтерфейс користувача. Nina.az. https://www.wiki.uk-ua.nina.az/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0.html

5. System::Collections::Generic Namespace Reference. Aspose : https://reference.aspose.com/pub/cpp/namespace/system.collections.generic

6. Офіційний сайт Telemart.ua : https://telemart.ua/assembly.html

7. Паралельне програмування. Metanit: https://metanit.com/sharp/tutorial/12.1.php

Додаток А

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Data.SQLite;



namespace WindowsFormsApp1
{
   public partial class Form1 : Form
   {
      private static string stringConnection = $"Data Source = {Environment.CurrentDirectory}\\Complecting.sqlite;
Version=3;";
      private static DataSet dataSet = new DataSet();
      private static List<Complecting> allComplectings = new List<Complecting>();
      private static Complectings currentComplectings = new Complectings();
      private static List<Complecting> suitableComplecting = new List<Complecting>();
      private static List<Complectings> complectinges = new List<Complectings>();
      private static int complectingNumber = -1;

      internal static Complectings CurrentComplectings { get => currentComplectings; }
      public static string StringConnection { get => stringConnection; }
      internal static List<Complectings> Complectinges { get => complectinges; set => complectinges = value; }

      public Form1()
      {
```

```
InitializeComponent();
comboBox1.SelectedIndex = 0;
using (SQLiteConnection connetion = new SQLiteConnection(StringConnection)) {

   string commandText = "SELECT * FROM Complecting";
   connetion.Open(); // открыть command
   new SQLiteDataAdapter(commandText, connetion).Fill(dataSet);
   connetion.Close(); // закрыть соединение
   foreach (DataRow row in dataSet.Tables[0].Rows) {
      object[] items = row.ItemArray;
      string[] notCompletings = ((string)items[5]).Split(',');
      allComplectings.Add( new Complecting((string)items[1], (Types)int.Parse(string.Format("{0}", items[2])),
        (string)items[3], double.Parse(string.Format("{0}", items[4])), notCompletings.ToList()));
   }
   string commandText1 = "SELECT * FROM Complectings";
   connetion.Open(); // открыть command
   dataSet.Tables.Clear();
   new SQLiteDataAdapter(commandText1, connetion).Fill(dataSet);
   connetion.Close(); // закрыть соединение
   foreach (DataRow row in dataSet.Tables[0].Rows)
   {
      object[] items = row.ItemArray;
      string[] tempNames = ((string)items[1]).Split(',');
      List<Complecting> tempComplectings = new List<Complecting>();
      for (int i = 0; i < tempNames.Length; i++)
      {
         foreach(Complecting compl in allComplectings)
         {
            if (tempNames[i].Equals(compl.Name)) tempComplectings.Add(compl);
         }
      }
      Complectinges.Add(new Complectings(tempComplectings, (string)items[2]));
   }
   suitableComplecting = allComplectings;
}
dataGridView1.Rows.Clear();
```

```
        foreach (Complecting compl in suitableComplecting)
        {
            if ((int)compl.Type == 0)
            {
                DataGridViewRow row = new DataGridViewRow();
                row.CreateCells(dataGridView1);
                MemoryStream ms = new MemoryStream();

                row.Cells[0].Value = Image.FromFile($"{Environment.CurrentDirectory}\\images\\{compl.Name}.jpg");
                row.Cells[1].Value = compl.Name;
                row.Height = 200;
                dataGridView1.Rows.Add(row);
            }
        }
        DrawCompletinges(dataGridView2);
    }

    private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
    {
        foreach(Complecting complecting in allComplectings)
        {
            if (dataGridView1.Rows[dataGridView1.SelectedCells[0].RowIndex].Cells[1].Value.Equals(complecting.Name))
            {
                describeL.Text = complecting.Description;
                priceL.Text = complecting.Price + " грн.";
            }
        }

    }

    private void button1_Click(object sender, EventArgs e)
    {

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////

        if (complectingNumber == -1)
```

```
        {
            foreach (Complecting complecting in allComplectings)
            {
                if
(complecting.Name.Equals(dataGridView1.Rows[dataGridView1.SelectedCells[0].RowIndex].Cells[1].Value))
                {
                    CurrentComplectings.Add(complecting);
                }
            }
            resetSuitable(currentComplectings);
        }
        else
        {
            StringBuilder beforeUpdate = new StringBuilder();
            foreach(Complecting compl in complectinges[complectingNumber].ListComplectings)
            {
                beforeUpdate.Append(compl.Name + ",");
            }
            beforeUpdate.Remove(beforeUpdate.Length - 1, 1);
            foreach (Complecting complecting in allComplectings)
            {
                if
(complecting.Name.Equals(dataGridView1.Rows[dataGridView1.SelectedCells[0].RowIndex].Cells[1].Value))
                {
                    complectinges[complectingNumber].Add(complecting);
                }
            }
            StringBuilder afterUpdate = new StringBuilder();
            foreach (Complecting compl in complectinges[complectingNumber].ListComplectings)
            {
                afterUpdate.Append(compl.Name + ",");
            }
            afterUpdate.Remove(afterUpdate.Length - 1, 1);
            using (SQLiteConnection connection = new SQLiteConnection(stringConnection))
            {
                string commandText = $"UPDATE Complectings SET namesComplectings = \"{afterUpdate}\" " +
```

```
                $"WHERE namesComplectings = \"{beforeUpdate}\"";
            connection.Open();
            new SQLiteCommand(commandText, connection).ExecuteNonQuery();
            connection.Close();
        }
        resetSuitable(currentComplectings);
    }
}
private void DrawOnDGV(DataGridView dataGrid, List<Complecting> complectings)
{
    dataGrid.Rows.Clear();
    foreach (Complecting compl in complectings)
    {
        DataGridViewRow row = new DataGridViewRow();
        row.CreateCells(dataGrid);
        MemoryStream ms = new MemoryStream();

        row.Cells[0].Value = Image.FromFile($"{Environment.CurrentDirectory}\\images\\{compl.Name}.jpg");
        row.Cells[1].Value = compl.Name;
        row.Height = 200;
        dataGrid.Rows.Add(row);
    }
}

private void button2_Click(object sender, EventArgs e)
{
    createComplectingP.Visible = false;
    workWithComplectationsP.Visible = true;
    DrawOnDGV(dataGridView3, CurrentComplectings.ListComplectings);
    totalPriceL.Text = CurrentComplectings.AllCost().ToString();
}

private void resetSuitable(Complectings complectings)
{
    suitableComplecting = allComplectings;
    foreach (Complecting complecting in complectings.ListComplectings)
```

```
    {

        for(int i = suitableComplecting.Count - 1; i >= 0 ; i--)
        {
            if (complecting.NotComplectingNames.Contains(allComplectings[i].Name))
            {
                suitableComplecting.Remove(allComplectings[i]);
            }
        }
    }
    dataGridView1.Rows.Clear();
    foreach (Complecting compl in suitableComplecting)
    {
        if ((int)compl.Type == comboBox1.SelectedIndex)
        {
            DataGridViewRow row = new DataGridViewRow();
            row.CreateCells(dataGridView1);
            MemoryStream ms = new MemoryStream();

            row.Cells[0].Value = Image.FromFile($"{Environment.CurrentDirectory}\\images\\{compl.Name}.jpg");
            row.Cells[1].Value = compl.Name;
            row.Height = 200;
            dataGridView1.Rows.Add(row);
        }
    }

}

private void dataGridView2_CellContentClick(object sender, DataGridViewCellEventArgs e)
{

    int index = dataGridView2.SelectedCells[0].RowIndex;
    complectingNumber = index;
    DrawOnDGV(dataGridView3,complectinges[index].ListComplectings);
    totalPriceL.Text = Complectinges[index].AllCost().ToString();
}
```

```
        private void button3_Click(object sender, EventArgs e)
        {
            DrawOnDGV(dataGridView3, CurrentComplectings.ListComplectings);
            totalPriceL.Text = CurrentComplectings.AllCost().ToString();
            complectingNumber = -1;
            resetSuitable(currentComplectings);
        }

        private void button4_Click(object sender, EventArgs e)
        {
            if (currentComplectings.ListComplectings.Count() == 0) {
                MessageBox.Show("Пуста комплектація");
            }
            else
            {
                if (!textBox1.Text.Equals(""))
                {
                    currentComplectings.Name = textBox1.Text;
                    StringBuilder str = new StringBuilder();
                    foreach (Complecting complecting in currentComplectings.ListComplectings)
                    {
                        str.Append(complecting.Name + ",");
                    }
                    str.Remove(str.Length - 1, 1);
                    using (SQLiteConnection connection = new SQLiteConnection(stringConnection))
                    {
                        string commandText = $"INSERT INTO Complectings (name, namesComplectings) VALUES
(\"{currentComplectings.Name}\", \"{str}\")";
                        connection.Open();
                        new SQLiteCommand(commandText, connection).ExecuteNonQuery();
                        connection.Close();
                    }
                    Complectings compl = new Complectings();
                    List<Complecting> listCompl = new List<Complecting>();
                    foreach (Complecting complecting1 in currentComplectings.ListComplectings)
```

```
            listCompl.Add(complecting1);
        compl.Name = currentComplectings.Name;
        compl.ListComplectings = listCompl;
        complectinges.Add(compl);
        DrawCompletinges(dataGridView2);
      }
      else MessageBox.Show("Введіть назву конфігурації");
    }
    currentComplectings.ListComplectings.Clear();
    currentComplectings.Name = "";
    dataGridView3.Rows.Clear();
}

public static void DrawCompletinges(DataGridView dataGrid)
{
    dataGrid.Rows.Clear();
    foreach (Complectings complectings in Complectinges)
    {
      DataGridViewRow row = new DataGridViewRow();
      row.CreateCells(dataGrid);
      row.Cells[0].Value = complectings.Name;
      dataGrid.Rows.Add(row);
    }
}

private void button5_Click(object sender, EventArgs e)
{
    if(complectingNumber != -1)
    {
      resetSuitable(complectinges[complectingNumber]);
    }
    createComplectingP.Visible = true;
    workWithComplectationsP.Visible = false;
}

private void button6_Click(object sender, EventArgs e)
```

```
    {
        List<Complecting> complectings = new List<Complecting>();
        for(int i = 0; i < dataGridView3.Rows.Count; i++)
        {
            foreach(Complecting complecting in allComplectings)
            {
                if(complecting.Name.Equals(dataGridView3.Rows[i].Cells[1].Value.ToString()))
                {
                    complectings.Add(complecting);
                }
            }
        }if (complectinges.Count != 0)
        {
            bool equal = true;
            if (currentComplectings.ListComplectings.Count == complectings.Count)
            {
                for (int i = 0; i < currentComplectings.ListComplectings.Count; i++)
                {
                    if (!currentComplectings.ListComplectings[i].Equals(complectings[i])) equal = false;
                }
            }
            else { equal = false; };
            if (equal)
            {
                currentComplectings.ListComplectings.Remove(complectings[dataGridView3.SelectedCells[0].RowIndex]);
                DrawOnDGV(dataGridView3, currentComplectings.ListComplectings);
                totalPriceL.Text = currentComplectings.AllCost().ToString();
                resetSuitable(currentComplectings);
            }
            else
            {
                StringBuilder beforeRemove = new StringBuilder();
                foreach(Complecting complecting in
complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings)
                {
                    beforeRemove.Append(complecting.Name + ",");
```

```
            }
            beforeRemove.Remove(beforeRemove.Length - 1, 1);

complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings.RemoveAt(dataGridView3.SelectedCells[0
].RowIndex);
            DrawOnDGV(dataGridView3, complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings);
            totalPriceL.Text = complectinges[dataGridView2.SelectedCells[0].RowIndex].AllCost().ToString();
            StringBuilder afterRemove = new StringBuilder();
            foreach (Complecting complecting in
complectinges[dataGridView2.SelectedCells[0].RowIndex].ListComplectings)
            {
               afterRemove.Append(complecting.Name + ",");
            }
            afterRemove.Remove(afterRemove.Length - 1, 1);
            using (SQLiteConnection connection = new SQLiteConnection(stringConnection))
            {
               string commandText = $"UPDATE Complectings SET namesComplectings = \"{afterRemove}\" " +
                  $"WHERE namesComplectings = \"{beforeRemove}\"";
               connection.Open();
               new SQLiteCommand(commandText, connection).ExecuteNonQuery();
               connection.Close();
            }
         }
      }
      else
      {
         MessageBox.Show("В конфігурації немає єлементів");
      }

   }

   private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
   {
      dataGridView1.Rows.Clear();
      foreach (Complecting compl in suitableComplecting)
      {
         if ((int)compl.Type == comboBox1.SelectedIndex)
```

```
            {
                DataGridViewRow row = new DataGridViewRow();
                row.CreateCells(dataGridView1);
                MemoryStream ms = new MemoryStream();

                row.Cells[0].Value = Image.FromFile($"{Environment.CurrentDirectory}\\images\\{compl.Name}.jpg");
                row.Cells[1].Value = compl.Name;
                row.Height = 200;
                dataGridView1.Rows.Add(row);
            }
        }
    }

  }
}
```