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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Створення застосунку з підбору комплектуючих для комп'ютера мовою C#»

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

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

_____ Володимир ПРОЦУН
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Володимир ПРОЦУН

Керівник: Віктор ГРЕБЕНЮК
доктор філософії (PhD)

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Процуну Володимиру Сергійовичу

1. Тема кваліфікаційної роботи: «Створення застосунку з підбору комплектуючих для комп'ютера мовою C#»
керівник кваліфікаційної роботи доктор філософії (PhD), доцент кафедри ІПЗ Віктор ГРЕБЕНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

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

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи моделювання інформаційних систем; онлайн редактор UML-діаграм; IDE Microsoft Visual Studio.

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

1. Комп'ютер як важливий інструмент у житті людини
2. Аналіз наявних засобів та технологій розробки програмного забезпечення для підбору комплектуючих до ПК
3. Моделювання та проектування інформаційної системи

4. Тестування системи.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма класів
6. Діаграма станів меню
7. Відео роботи застосунку
8. Екранні форми.
9. Апробація результатів дослідження

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих алгоритмів для підбору комплектуючих до ПК	14.03-18.03.2024	
4	Проектування архітектури системи	19.03-31.03.2024	
5	Розробка застосунку для підбору комплектуючих до комп'ютера	01.04-18.04.2024	
6	Тестування застосунку	19.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Володимир ПРОЦУН
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ГРЕБЕНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 41 стор., 1 табл., 36 рис., 23 джерело.

Мета роботи – спрощення підбору комплектуючих до комп'ютера

Об'єкт дослідження – процес прийняття рішення для підбору комплектуючих до комп'ютера.

Предмет дослідження – програмне забезпечення для підбору комплектуючих до комп'ютера мовою C#.

Короткий зміст роботи: У даному досліджені було проаналізовано алгоритми та методи для підбору комплектуючих для персонального комп'ютера. Проаналізовано аналогічні рішення конкурентів для підбору комплектуючих до персонального комп'ютера, такі як: Telemart.ua, Elmir.ua. Створено алгоритм функціонування застосунку та програмно реалізовані основні функціональні можливості, такі як: завантаження інформації про компоненти з бази даних, можливість підбору комплектуючих користувачем, алгоритм підбору сумісних один до одного компонентів, можливість експорту збірки. Проведено ручне тестування застосунку. В роботі використано базу даних MySQL, мову C#, інтегроване середовище розробки Microsoft Visual Studio, Windows Forms.

Сферою використання застосунку є будь яка потреба в швидкому та зручному підборі комплектуючих до ПК.

КЛЮЧОВІ СЛОВА: ПІДБІР КОМЛЕКТУЮЧИХ, БАЗА ДАНИХ, АЛГОРИТМ ПІДБОРУ СУМІСНИХ КОМПОНЕНТІВ, ЕКСПОРТ ЗБІРКИ, MySQL, C#, WINDOWS FORMS.

ЗМІСТ

ВСТУП

В наш час персональний комп’ютер став не лише предметом роскоші, але і необхідним інструментом для освіти, роботи, зв’язку та розваг, тож у кожній людині має бути доступний персональний комп’ютер для забезпечення можливостей навчання, розвитку та успішної соціальної і професійної інтеграції. Але при цьому існує проблема, що багато людей не знає, як правильно підібрати комплектуючі, як правильно зібрати ПК, та як ефективно витратити кошти.

На теперішній час існує немало рішень для підбору комплектуючих до ПК, але в своїй основі вони мають більше комерційну основу для продажу цих компонентів та часто мають на меті саме продати потрібні їм комплектуючі, а не об’єктивно дати інструменти та допомогти у збірці персонального комп’ютера.

Отже обрана тема є актуальною.

Об’єктом дослідження є процес прийняття рішення для підбору комплектуючих до персонального комп’ютера.

Предметом роботи є програмне забезпечення для підбору комплектуючих до персонального комп’ютера.

Метою роботи є спрощення підбору комплектуючих до комп’ютера.

В процесі дослідження вирішувалися наступні завдання:

- Аналіз існуючих застосунків для підбору комплектуючих до ПК.
- Вивчення потреб користувачів у підборі комплектуючих.
- Розробка структури даних та їх взаємодія.
- Розробка алгоритму підбору збірки.
- Розробка архітектури програмного забезпечення.
- Реалізація застосунку для підбору комплектуючих до ПК
- Тестування програмного продукту

Методика дослідження: уніфікований процес створення програмного забезпечення.

Практична значущість: цей продукт може бути використаний у будь-яких

галузях, де потрібно зібрати ПК. Його універсальність і зручність у застосуванні дозволяють значно спростити процес збірки комп'ютерів різної складності та конфігурації.

Робота пройшла апробацію на Всеукраїнській науково-технічній конференції “Застосування програмного забезпечення в ІКТ” (24.04.2024, ДУІКТ, м. Київ). За результатами участі опубліковано тези доповідей [22,23]

1 КОМП’ЮТЕР ЯК НЕОБХІДНИЙ ІНСТУМЕНТ У ЖИТТІ ЛЮДИНИ

1.1 Комп’ютер у сучасному світі

Комп'ютери відіграють критичну роль у сучасному суспільстві, забезпечуючи різноманітні переваги, такі як поліпшення комунікації, підвищення ефективності роботи та розширення можливостей в науці та технологіях. Завдяки комп'ютерам сучасне суспільство переживає революцію в інформаційних технологіях, що сприяє зростанню ефективності усіх сфер життя. Вони дозволяють швидко обмінюватися інформацією, спрощують ведення бізнесу та організацію робочих процесів. Комп'ютери є невід'ємною частиною наукових досліджень, що сприяють відкриттю нових знань і технологій. Таким чином, вони є не лише інструментом, але й мотором для прогресу і розвитку суспільства.

Крім того, комп'ютери відіграють важливу роль у освіті, дозволяючи доступ до великої кількості навчальних матеріалів і засобів навчання, що робить процес навчання більш доступним і ефективним. Вони також сприяють розвитку творчих індустрій, таких як мультимедіа, графіка та дизайн, дозволяючи творити й втілювати нові ідеї. Комп'ютери змінюють спосіб, яким ми працюємо, навчаємося і взаємодіємо один з одним, і вони продовжують відігравати ключову роль у подальшому розвитку суспільства.

Загалом, комп'ютери не лише полегшують повсякденні завдання і підвищують ефективність роботи, але й є важливим стимулом для інновацій і розвитку суспільства в цілому.

1.2 Проблематика збірки ПК

Збірка персональних комп'ютерів є складною проблемою для звичайних користувачів через технічну складність, необхідність знань у галузі апаратного

забезпечення та ризик несумісності компонентів, що ускладнює доступ до персоналізованих і оптимально працюючих комп'ютерних систем.

Основними проблемами, з якими зіштовхуються звичайні користувачі під час збірки персональних комп'ютерів, є незрозумілість технічних характеристик компонентів, складність вибору сумісних частин, а також необхідність витрачати значний час на дослідження та вибір оптимальних варіантів. Крім того, недостатня підтримка або відсутність гарантії на результативність і сумісність між компонентами можуть призвести до проблем з надійністю та ефективністю роботи системи.

2 АНАЛІЗ НАЯВНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДБОРУ КОМПЛЕКТУЮЧИХ ДО ПК

2.1 Аналіз існуючих рішень для підбору комплектуючих до ПК

У контексті зростаючого попиту на персональні рішення в сфері збірки персональних комп’ютерів, дослідження конкурентного середовища застосунків для підбору комплектуючих до ПК є критично важливим для розробки ефективного, інноваційного продукту на мові C#, який задовольняє потреби ринку та виокремлюється своїми перевагами серед існуючих рішень.

Це дослідження сприятиме глибшому розумінню ключових функціональних характеристик та інновацій, які впроваджують конкуренти, дозволяючи ідентифікувати потенційні ніші для інновацій та вдосконалення у нашому продукті. Важливість цього аналізу підкріплюється необхідністю розробки застосунку, який не просто конкурує, але пропонує унікальні можливості, що оптимізують процес підбору комплектуючих для ПК, роблячи його більш зручним, доступним та адаптованим до специфічних потреб користувачів.

Найпопулярнішими рішеннями в Україні у цій сфері є:

1. «Telemart.ua»

Telemart.ua являє собою найпопулярніший сайт в Україні для покупки та підбору комплектуючих для ПК. У вкладці підбору комплектуючих, який називається «Конфігуратор ПК». Користувач має можливість підібрати комплектуючі за типом, ціною, його сумісності з іншими компонентами та ефективністю. Також сайт був зроблений досить зручного для багатьох користувачів, що дає змогу не витрачати час для вивчення його інтерфейсу.

Рішення має наступний інтерфейс (Рис. 2.1-2.3):

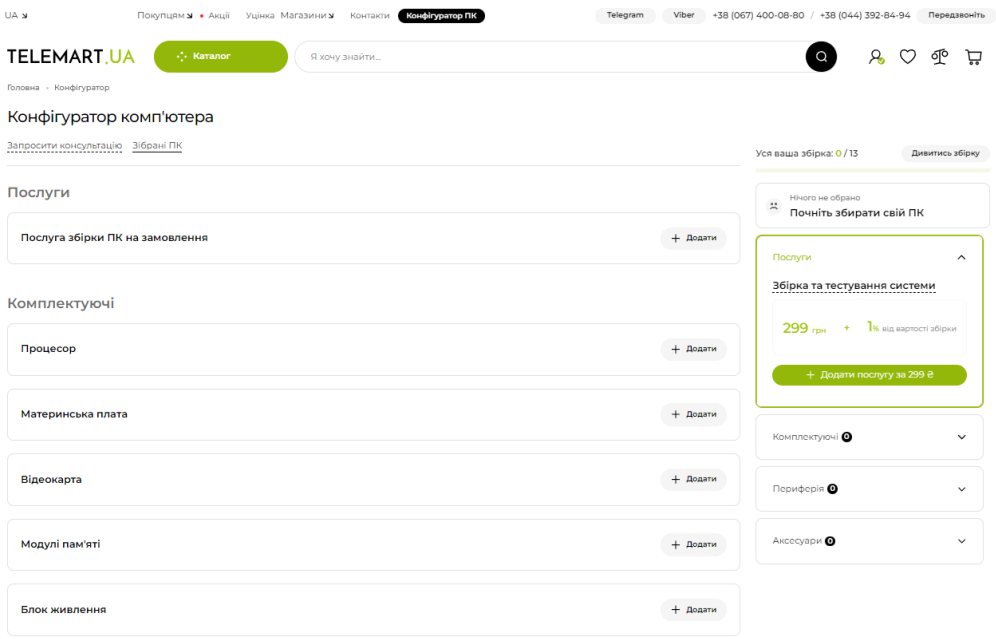


Рис 2.1 інтерфейс «Конфігуратор ПК» від Telemart.ua

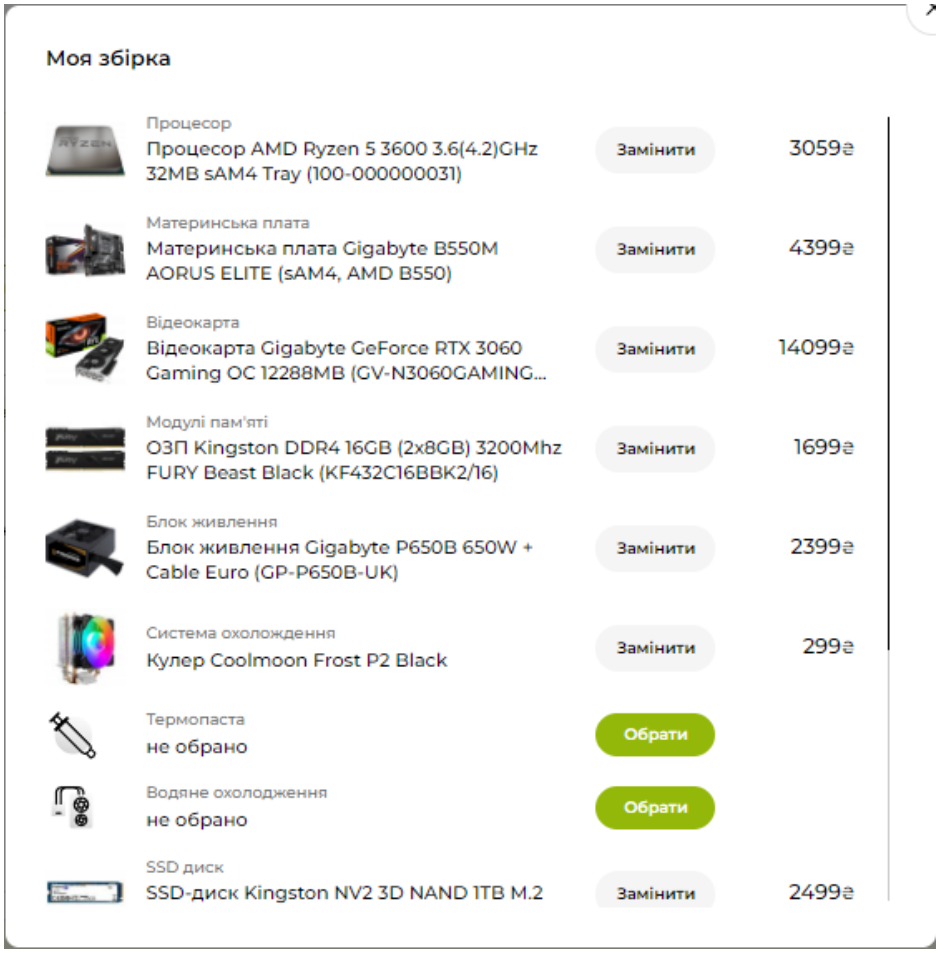


Рис. 2.2 Інтерфейс «Конфігуратор ПК» від Telemart.ua

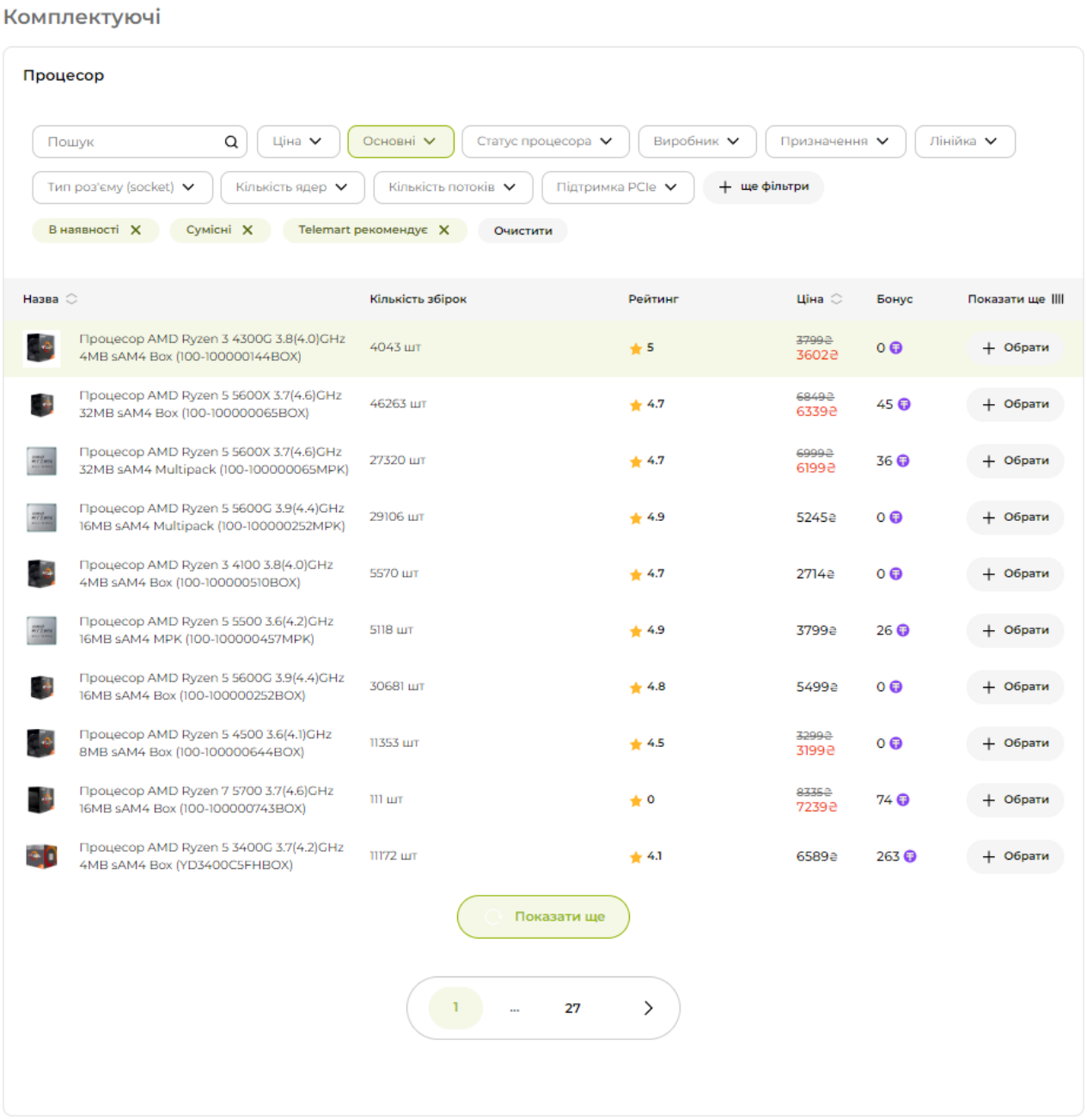


Рис. 2.3 Інтерфейс «Конфігуратор ПК» від Telemart.ua

2. «Elmir.ua»

Elmir.ua теж інтернет магазин для продажу комплектуючих до ПК в Україні, але є менш популярним ніж Telemart.ua. На відмінну від Telemart, цей магазин має, на мою думку, має більш user-frendly інтерфейс у вкладці конфігуратора ПК, можливість подивитися, які обов’язкові компоненти треба підібрати і їх докладні характеристики та зручні фільтри до кожного елементу ПК.

Рішення має наступний інтерфейс. Дивитися рисунки 2.4 - 2.6

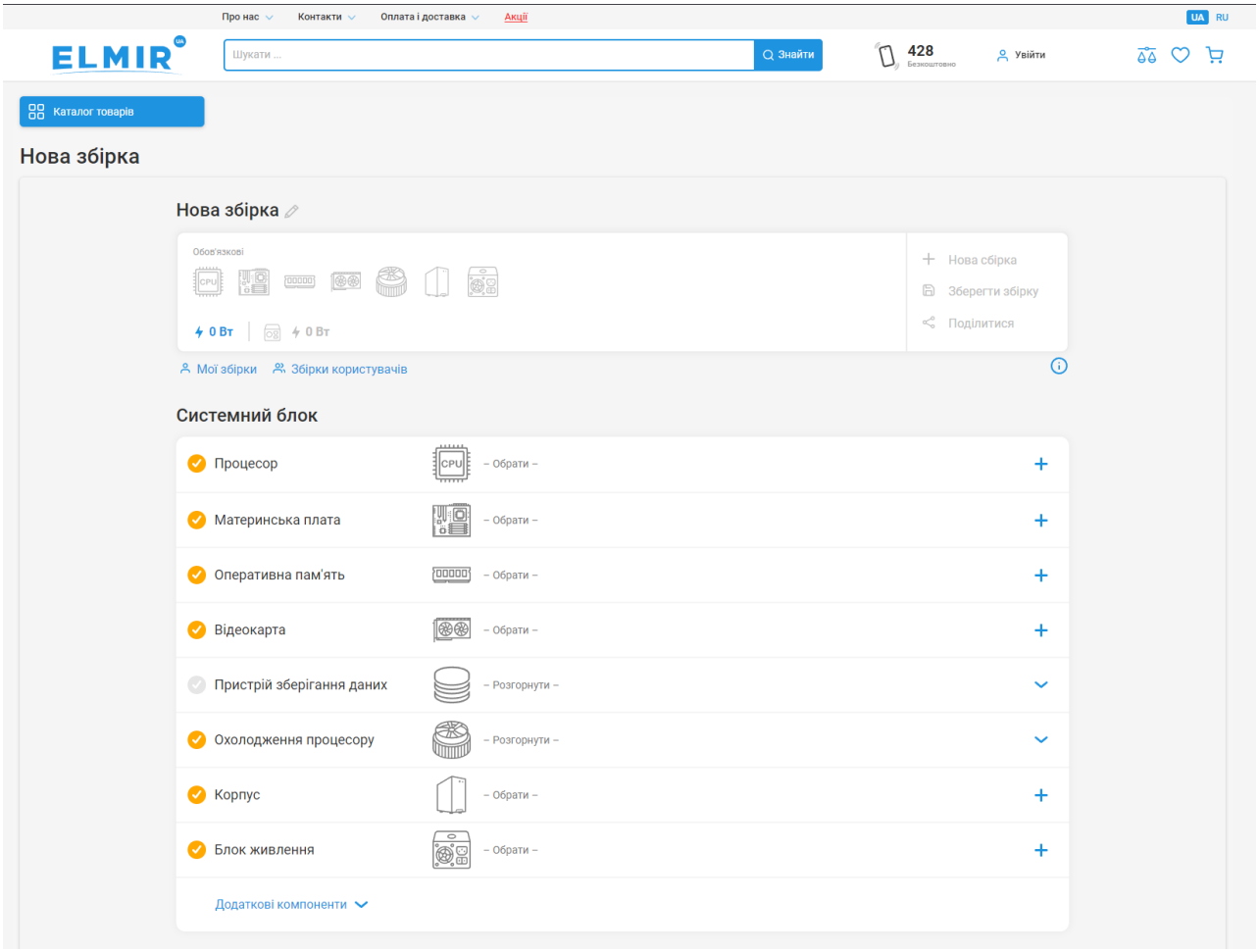


Рис. 2.4 Інтерфейс «Конфігуратор ПК» від Elmir.ua

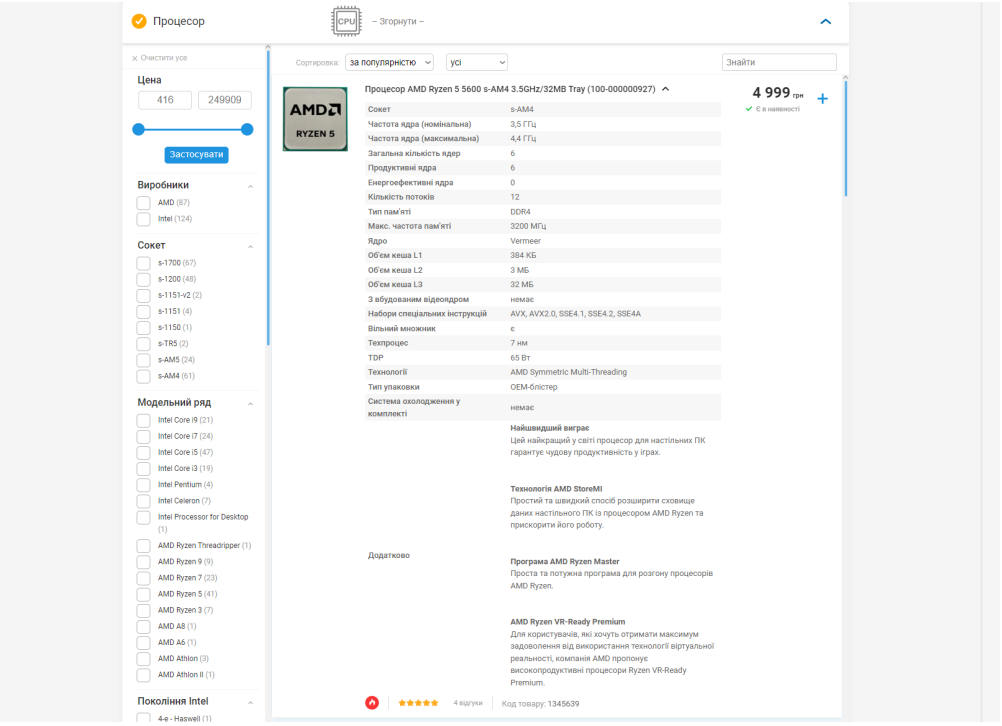


Рис. 2.5 Інтерфейс «Конфігуратор ПК» від Elmir.ua

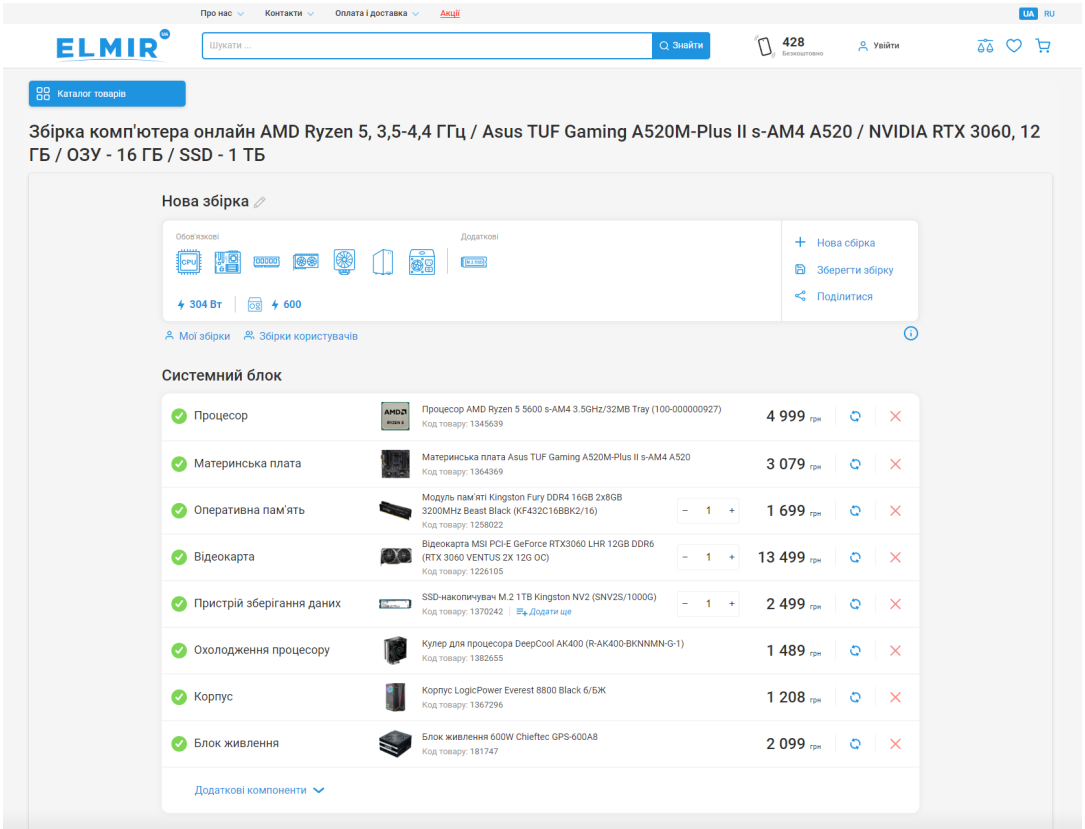


Рис. 2.6 Інтерфейс «Конфігуратор ПК» від Elmir.ua

На таблиці 2.1 можна побачити, які переваги буде мати застосунок для підбору комплектуючих до комп’ютера, який буде реалізовано у даній роботі

Таблиця 2.1

Порівняння існуючих програм-аналогів

Характеристика	Telemart.ua	Elmir.ua	PCComponent
Можливість експорту збірки	-	-	+
Наявність гайдів для збірки	-	-	+
Перевірка сумісності компонентів	+	+	+
Десктопна версія застосунку	-	-	+
Перегляд детальної інформації про компоненти	+	+	+

Отже, аналіз конкурентного середовища дозволив зрозуміти, що існуючі застосунки для підбору комплектуючих до ПК відзначаються різноманітністю функціональних можливостей та рівнем користувацької зручності. На основі цього дослідження можна зробити висновок, що власний застосунок має потенціал для покращення шляхом інтеграції унікальних функцій та оптимізації користувацького досвіду. Такий підхід дозволить нашому продукту зайняти конкурентну позицію на ринку та забезпечити задоволення потреб користувачів у підборі найкращих комплектуючих для їхніх ПК.

2.2 Вибір технологій та засобів розробки програмного забезпечення

При розробці програмного забезпечення для підбору комплектуючих до ПК

важливо обирати технології та засоби розробки, що забезпечують ефективність, швидкість розробки та зручність для користувача. Підбір правильних технологій допомагає забезпечити оптимальну продуктивність та надійність програмного забезпечення, сприяє зниженню витрат часу та ресурсів на розробку і підтримку, а також забезпечує максимальну задоволеність від користування продуктом для кінцевих користувачів.

До засобів розробки програмного забезпечення для підбору комплектуючих до ПК належать: мова програмування, IDE (укр. Інтегроване середовище розробки) та система керування базами даних.

2.2.1 Онлайн редактор UML-діаграм UMLetino

UMLetino являє собою інноваційний інструмент для зручного та ефективного моделювання структур та процесів програмного забезпечення. Цей редактор UML-діаграм дозволяє розробникам створювати, візуалізувати та аналізувати різноманітні аспекти програмного проекту, запезпечуючи зручний інтерфейс та багатий функціонал. У результаті, використання UMLetino сприяє прискоренню процесу розробки та підвищенню якості розроблених рішень. Крім того, UMLetino є відкритим програмним забезпеченням, що сприяє зростанню його популярності та поширенню серед спільноти розробників.

Цей редактор підтримує різноманітні типи UML-діаграм, такі як діаграми класів, послідовності, діяльності тощо, що робить його універсальним інструментом для роботи з будь-якими аспектами програмного забезпечення, дозволяючи їм ефективно моделювати та аналізувати складні системи.

Редактор має наступний вигляд на рисунку 2.7.

Рис 2.7 Інтерфейс UMLetino

2.2.2 База даних MySQL

MySQL – це надійний інструмент для зберігання, організації та ефективного управління структурною інформацією в сучасних програмних додатках та веб-сторінках, який забезпечує широкий спектр можливостей, включаючи зручний інтерфейс, потужність в роботі з великим обсягом даних, підтримку різних типів даних та високий рівень безпеки.

База даних MySQL використовується в різних галузях, від веб-розробки до корпоративних систем, що підтверджує її значення як одного з найпопулярніших та надійних рішень для зберігання та управління даними.

База даних має наступний вигляд на рисунках 2.8-2.9.

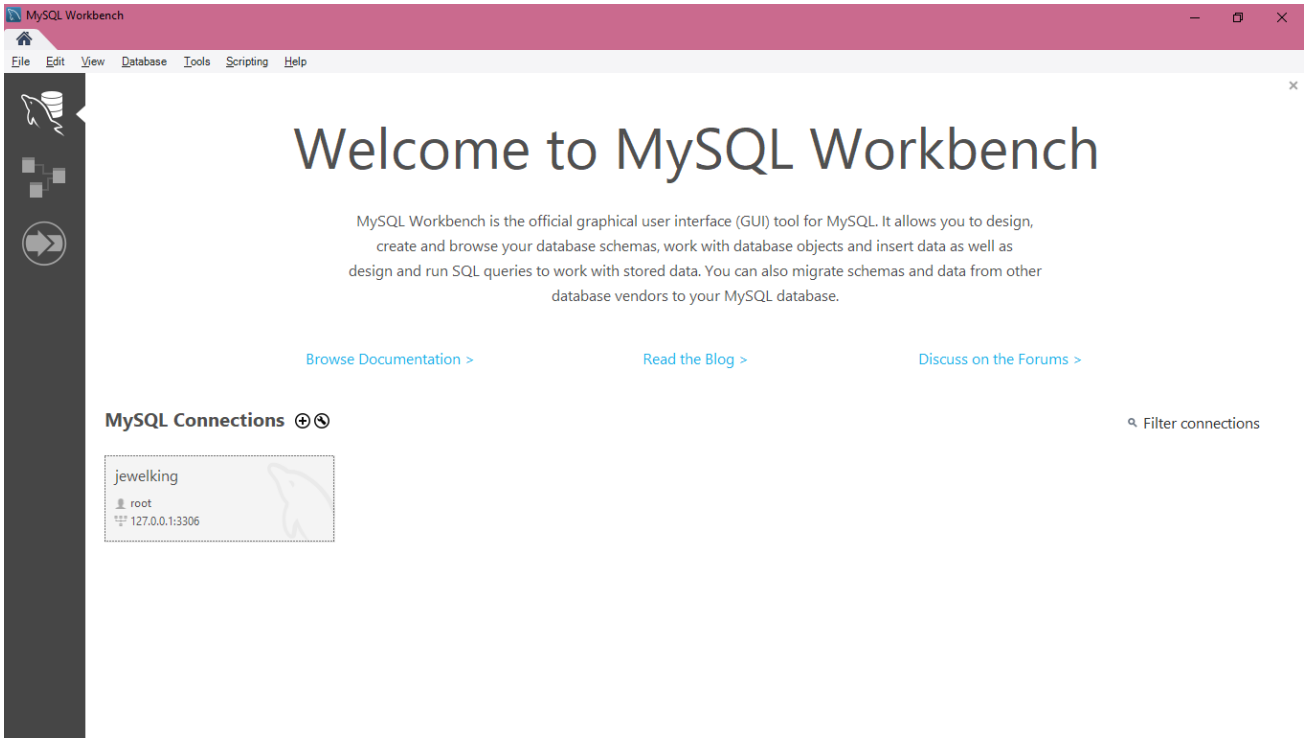


Рис 2.8 Інтерфейс MySQL

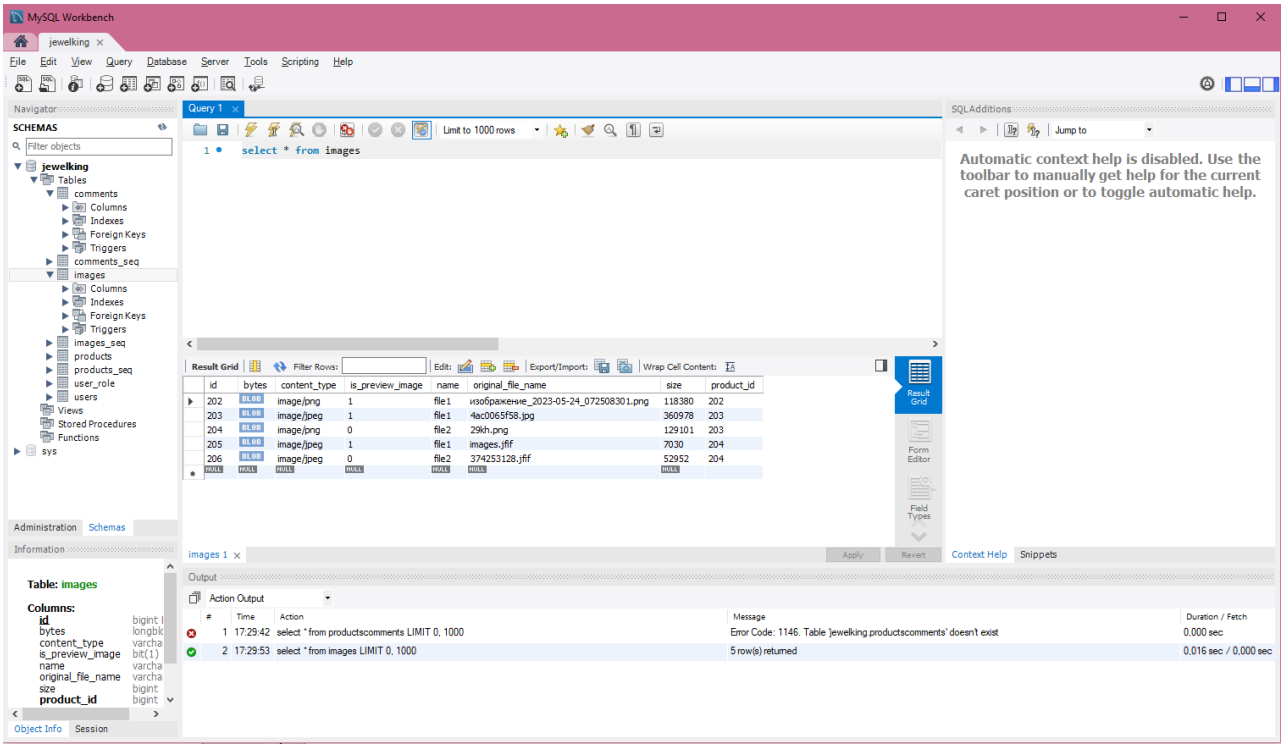


Рис 2.9 Інтерфейс MySQL

2.2.3 Мова програмування C#

Мова програмування C# – це високорівнева, об’єктно-орієнтована мова програмування, розроблена корпорацією Microsoft. Вона була випущена у 2000 році як частина ініціативи Microsoft .NET Framework та є однією з найбільш популярних мов програмування в світі, особливо в області розробки додатків для платформ Windows.

C# поєднує в собі потужність, гнучкість, високий рівень безпеки та простоту використання, що й робить її одну з найпопулярніших мов програмування у світі.

Основні особливості мови C# включають в себе об’єктно-орієнтоване програмування, асинхронні операції, делегати та події, вбудовану підтримку для керування пам’яттю, а також велику кількість бібліотек і фреймворків, доступних для розробників, завдяки інтеграції з платформою .NET.

Хоча C# і має багато переваг і є дуже популярною серед розробників, вона також має свої недоліки, такі як:

- Обмежена кросплатформенність: Хоча C# підтримується на платформах Windows, Linux і macOS через проекти, такі як .NET Core та .NET 5+, але це не так кросплатформенно, як у деяких інших мов програмування
- Залежність від екосистеми Microsoft: Розробка в середовищі C# зазвичай пов’язана з використанням інструментів та технологій, що розроблені або підтримуються компанією Microsoft, що може обмежувати вибір розробників.
- Обмежені можливості на мобільних платформах: Хоча існують фреймворки, що дозволяють розробляти мобільні додатки на мові C#, але це може бути не так ефективно та зручно, як використання мов, спеціально призначених для цієї цілі.

Незважаючи на ці недоліки, C# залишається однією з найпопулярніших мов програмування завдяки своїм перевагам та активній спільноті розробників.

2.2.4 Інтегроване середовище розробки Microsoft Visual Studio

Microsoft Visual Studio — це інтегроване середовище розробки (Integrated Development Environment, IDE), розроблене компанією Miscrosoft для програмування, налагодження та керування програмними проектами. Це один з найпопулярніших інструментів серед розробників програмного забезпечення.

Microsoft Visual Studio надає широкі можливості для роботи з різними мовами програмування, такими як C#, C++, Python, Visual Basic тощо. Воно включає в себе різноманітні інструменти, такі як редактор коду з підсвічуванням синтаксису, система керування версіями коду, інтегрована система налагодження, візуальний дизайнер інтерфейсу користувача, засоби тестування, а також можливості для роботи з веб-розробкою, мобільною розробкою, хмарними сервісами та іншими технологіями

Це інтегроване середовище розробки є найкращім рішенням для розробки програмного забезпечення на мові C# за його ефективність, зручність та надійність за рахунок широкого спектру інструментів та автоматизованим процесом розгортання та налагодження, що сприяє прискоренню та поліпшенню якості програмного продукту.

Інтегроване середовище має наступний вигляд на рисунках 2.10-2.11

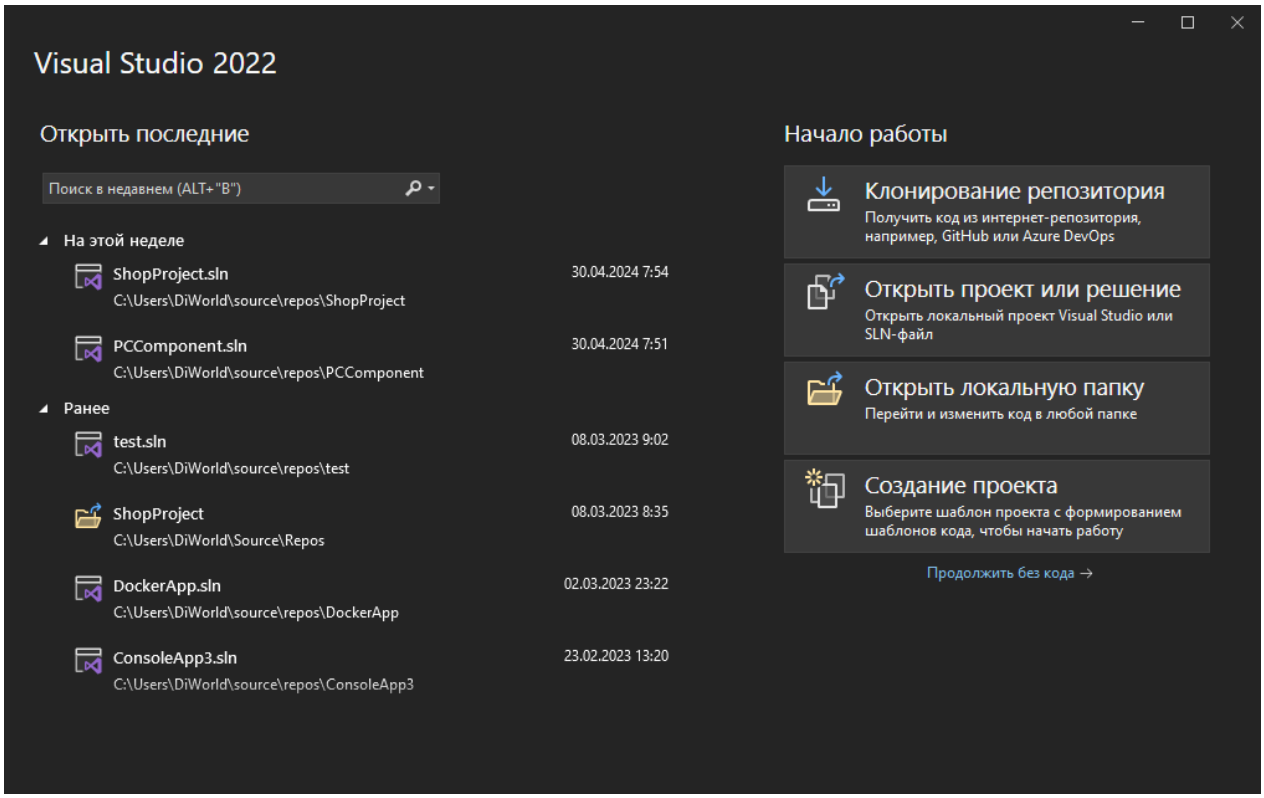


Рис. 2.10 Інтерфейс Visual Studio

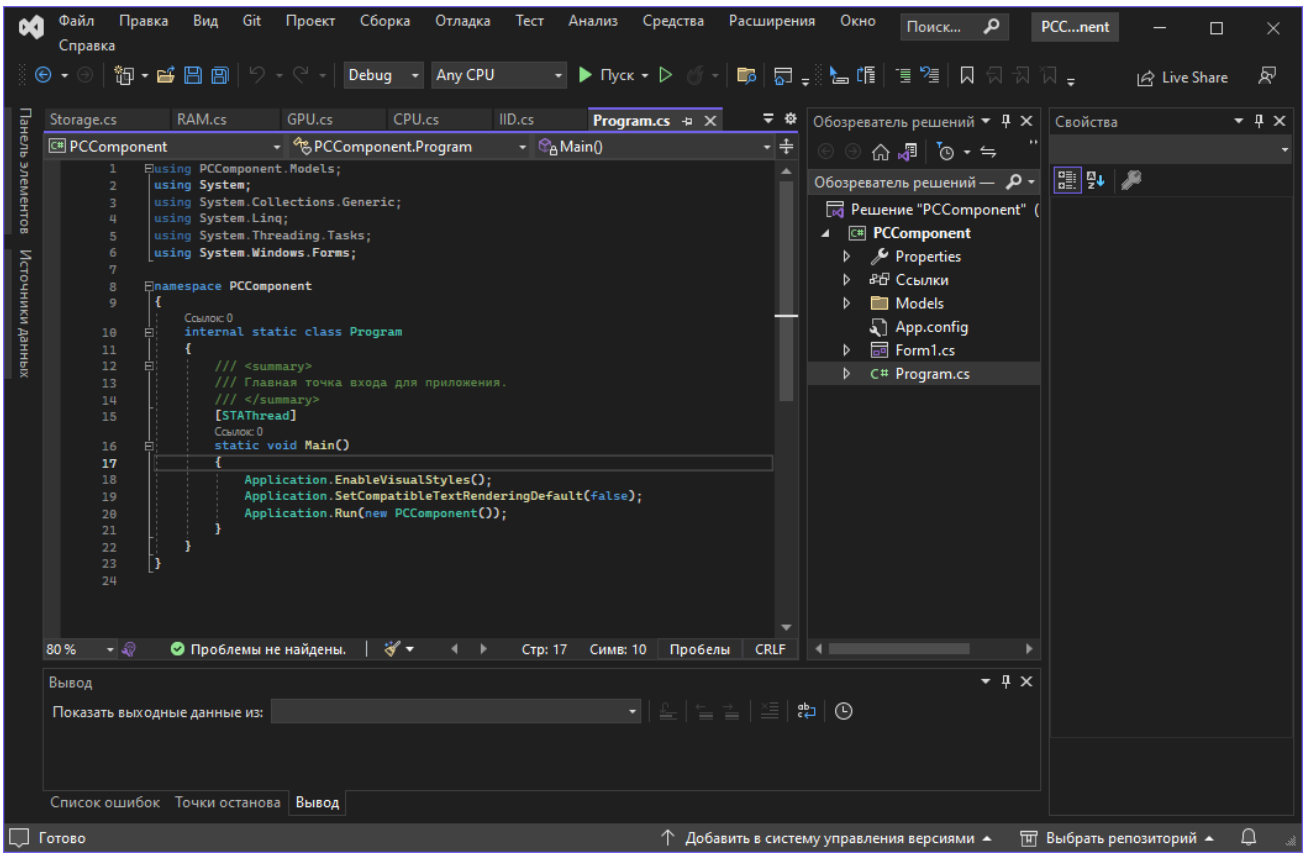


Рис. 2.11 Інтерфейс Visual Studio

2.2.5 Платформа .NET Framework

.NET Framework — це розвиток платформи Microsoft для розробки та виконання програмного забезпечення. Він надає зручне середовище для створення різноманітних додатків, включаючи веб-додатки, десктопні програми, служби та багато іншого. Платформа має вбудовану підтримку для мов програмування, таких як C#, C++ та Visual Basic, що дозволяє розробникам використовувати їх для створення програм.

.NET Framework також має багато вбудовану бібліотек класів та фреймворків, які спрощують розробку програм та забезпечують високу продуктивність та безпеку. Бібліотека класів — це набір готових класів, методів та інших компонентів, які можна використовувати для розробки програмного забезпечення. Вона зазвичай містить реалізацію загальних завдань, таких як робота з рядками, робота з файлами, мережеві операції тощо. Фреймворк — це великий набір заздалегідь написаного програмного забезпечення, який надає розробникам засоби для розробки та виконання програм. Він включає в себе бібліотеки класів, компоненти, інструменти для розробки, документації та інші корисні ресурси, які допомагають створювати програмне забезпечення.

2.2.6 Інтерфейс користувача Windows Forms

Windows Forms — це один з наборів інтерфейсу користувача (UI) для створення клієнтських десктопних програм у середовищі розробки Visual Studio. Вони надають інструменти для створення графічного користувача (GUI) у програмах для операції системи Windows.

Windows Forms базуються на мові програмування C# і дозволяють розміщувати на формі різноманітні елементи керування, такі як кнопки, тексти, списки, таблиці тощо. Вони також підтримують події, які виникають при взаємодії користувача з програмою, такі як натискання кнопок або введення

тексту. Ще вони мають простий у використанні інтерфейс, що дозволяє швидко створювати та редагувати інтерфейс програми за допомогою перетягування та розміщення елементів у вікні форми.

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Модель предметної галузі

Діаграма предметної галузі (Domain Model) - це графічне представлення ключових понять і відносин в межах певної предметної області. Вона використовується для моделювання структур даних, які відображають сутності і їх взаємозв'язки в реальному світі. Цей інструмент дозволяє розробникам, аналітикам та іншим зацікавленим сторонам зрозуміти основні компоненти і взаємодії в системі.

До основних компонентів діаграми належать такі компоненти: Сутності, Атрибути та Зв’язки.

Діаграма предметної галузі має наступний вигляд на рисунку 3.1.

На даній діаграмі видно, що клас Motherboard використовує велику кількість інших класів, які описують характеристики компонентів ПК, а саме класи Case, Cooler, GPU, RAM, CPU , Storage та PowerSupply .

Можна звернути увагу, що Motherboard використовую декілька класів RAM та Storage, бо комп’ютери мають в собі декілька таких компонентів.

Реалізація моделей за допомогою коду на основі діаграми предметної галузі має наступний вигляд на рисунках 3.2 - 3.9

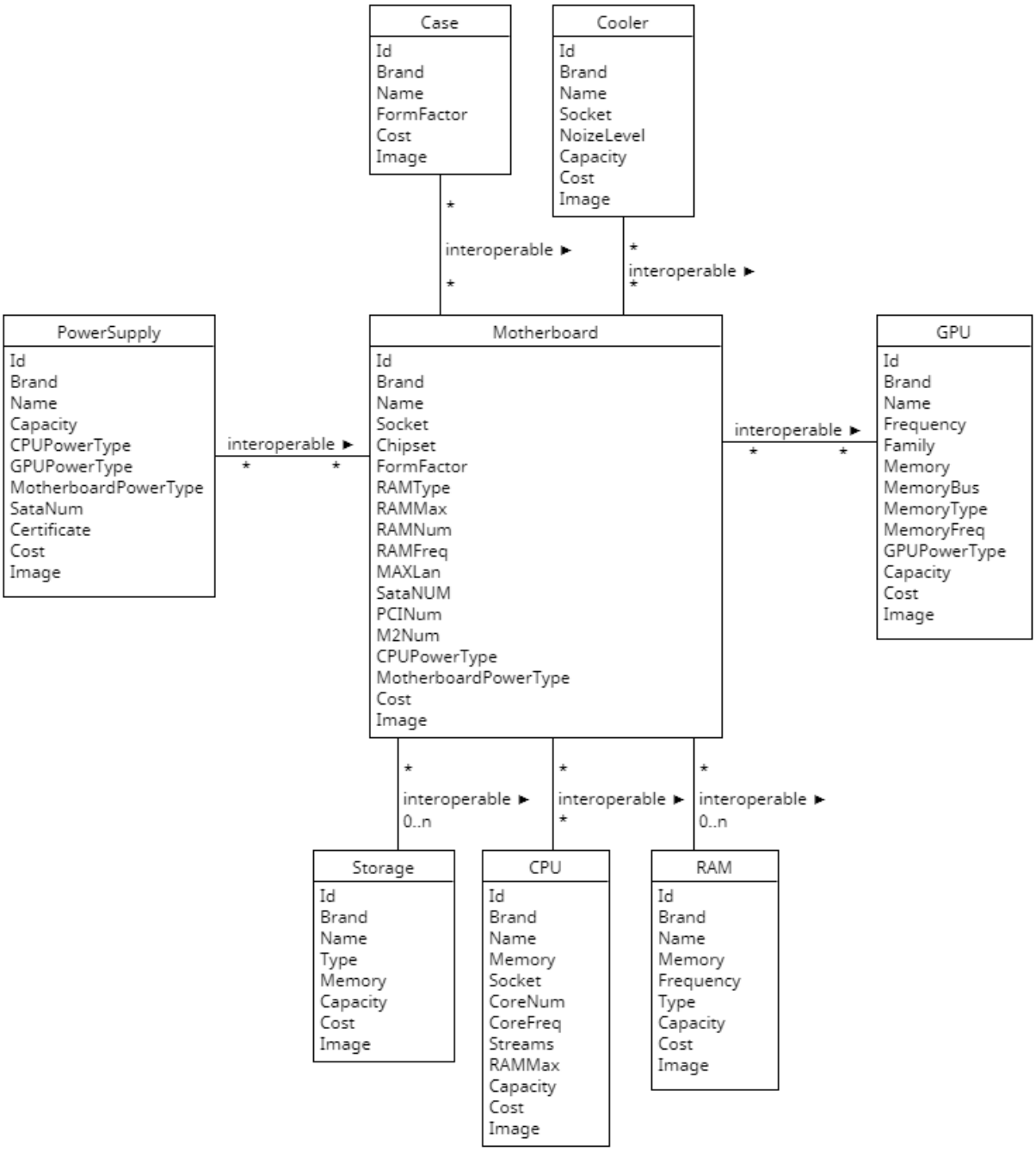


Рис. 3.1 Діаграма предметної галузі

```
internal class MotherBoard : IID
{
    Ссылок: 4
    public int ID { get; set; }
    Ссылок: 2
    public string Brand { get; set; }
    Ссылок: 4
    public string Name { get; set; }
    Ссылок: 4
    public string Socket { get; set; }
    Ссылок: 2
    public string Chipset { get; set; }
    Ссылок: 3
    public string FormFactor { get; set; }
    Ссылок: 3
    public string RAMType { get; set; }
    Ссылок: 2
    public int RAMMax { get; set; }
    Ссылок: 3
    public int RAMNum { get; set; }
    Ссылок: 2
    public float RAMFreq { get; set; }
    Ссылок: 2
    public int MaxLAN { get; set; }
    Ссылок: 2
    public int SATANum { get; set; }
    Ссылок: 2
    public int PCINum { get; set; }
    Ссылок: 4
    public int M2Num { get; set; }
    Ссылок: 4
    public string CPUPowerType { get; set; }
    Ссылок: 4
    public string MotherboardPowerType { get; set; }
    Ссылок: 3
    public int Cost { get; set; }
    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.2 Реалізація діаграми предметної галузі

```
internal class Case : IID
{
    Ссылок: 4
    public int ID { get; set; }
    Ссылок: 2
    public string Brand { get; set; }
    Ссылок: 3
    public string Name { get; set; }
    Ссылок: 3
    public string FormFactor { get; set; }
    Ссылок: 3
    public int Cost { get; set; }
    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.3 Реалізація діаграми предметної галузі

```
internal class Cooler : IID
{
    Ссылок: 4
    public int ID { get; set; }
    Ссылок: 2
    public string Brand { get; set; }
    Ссылок: 3
    public string Name { get; set; }
    Ссылок: 3
    public string Socket { get; set; }
    Ссылок: 2
    public string NoizeLevel { get; set; }
    Ссылок: 3
    public int Capacity { get; set; }
    Ссылок: 3
    public int Cost { get; set; }
    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.4 Реалізація діаграми предметної галузі

```
internal class GPU : IID
{
    Ссылок: 4
    public int ID { get; set; }
    Ссылок: 2
    public string Brand { get; set; }
    Ссылок: 3
    public string Name { get; set; }
    Ссылок: 2
    public float Frequency { get; set; }
    Ссылок: 2
    public string Family { get; set; }
    Ссылок: 2
    public int Memory { get; set; }
    Ссылок: 2
    public int MemoryBus { get; set; }
    Ссылок: 2
    public string MemoryType { get; set; }
    Ссылок: 2
    public float MemoryFreq { get; set; }
    Ссылок: 4
    public string GPUPowerType { get; set; }
    Ссылок: 3
    public int Capacity { get; set; }
    Ссылок: 3
    public int Cost { get; set; }
    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.5 Реалізація діаграми предметної галузі

```
internal class RAM
{
    Ссылок: 6
    public int ID { get; set; }
    Ссылок: 2
    public string Brand { get; set; }
    Ссылок: 6
    public string Name { get; set; }
    Ссылок: 2
    public int Memory { get; set; }
    Ссылок: 2
    public float Frequency { get; set; }
    Ссылок: 3
    public string Type { get; set; }
    Ссылок: 3
    public int Capacity { get; set; }
    Ссылок: 3
    public int Cost { get; set; }
    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.6 Реалізація діаграми предметної галузі

```
internal class CPU : IID
{
    Ссылок: 4
    public int ID { get; set; }

    Ссылок: 2
    public string Brand { get; set; }

    Ссылок: 3
    public string Name { get; set; }

    Ссылок: 2
    public float Memory { get; set; }

    Ссылок: 3
    public string Socket { get; set; }

    Ссылок: 2
    public int CoreNum { get; set; }

    Ссылок: 2
    public float CoreFreq { get; set; }

    Ссылок: 2
    public int Streams { get; set; }

    Ссылок: 2
    public int RAMMax { get; set; }

    Ссылок: 3
    public int Capacity { get; set; }

    Ссылок: 3
    public int Cost { get; set; }

    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.7 Реалізація діаграми предметної галузі

```
internal class Storage : IID
{
    Ссылок: 6
    public int ID { get; set; }
    Ссылок: 2
    public string Brand { get; set; }
    Ссылок: 5
    public string Name { get; set; }
    Ссылок: 2
    public string Type { get; set; }
    Ссылок: 2
    public int Memory { get; set; }
    Ссылок: 3
    public int Capacity { get; set; }
    Ссылок: 3
    public int Cost { get; set; }
    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.8 Реалізація діаграми предметної галузі

```
internal class PowerSupply : IID
{
    Ссылок: 4
    public int ID { get; set; }
    Ссылок: 2
    public string Brand { get; set; }
    Ссылок: 3
    public string Name { get; set; }
    Ссылок: 3
    public int Capacity { get; set; }
    Ссылок: 4
    public string CPUPowerType { get; set; }
    Ссылок: 4
    public string GPUPowerType { get; set; }
    Ссылок: 4
    public string MotherboardPowerType { get; set; }
    Ссылок: 2
    public int SATANum { get; set; }
    Ссылок: 2
    public string Certificate { get; set; }
    Ссылок: 3
    public int Cost { get; set; }
    Ссылок: 2
    public byte[] Image { get; set; }
}
```

Рис. 3.9 Реалізація діаграми предметної галузі

3.2 Модель варіантів використання

Діаграма варіантів використання (Use Case Diagram) — це один із типів діаграм UML (Unified Modeling Language), який використовується для моделювання функціональних вимог системи. Вона показує взаємодію між акторами (користувачами або іншими системами) і системою для виконання певних дій або варіантів використання (use cases).

До основних компонентів діаграми належать такі компоненти: Актори, Варіанти використання, Відносини та Система.

Діаграма варіантів використання має наступний вигляд на рисунку 3.10.

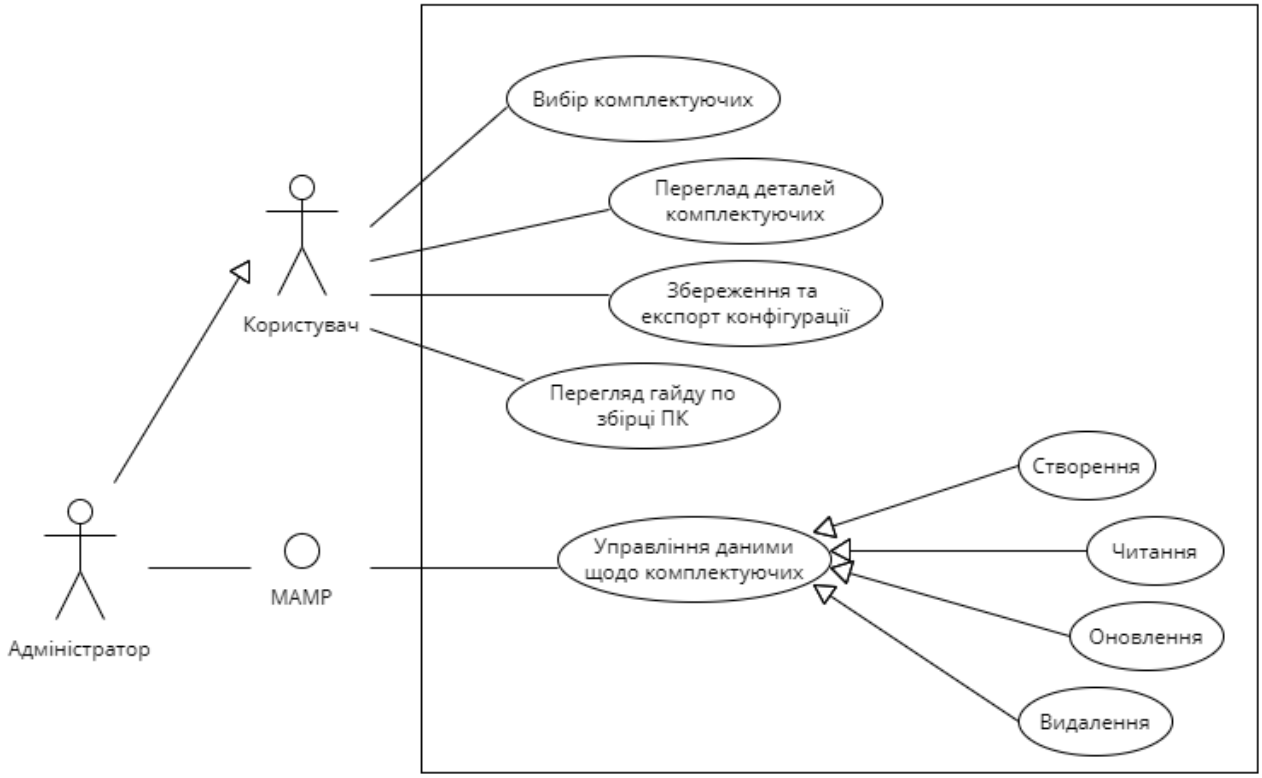


Рис. 3.10 Діаграма варіантів використання

Проаналізувавши дану діаграму варіантів використання можна побачити, що в ній є два актори: Користувач та Адміністратор.

У сутності Користувач є чотири варіанти використання, а саме: «Вибір комплектуючих», «Перегляд деталей комплектуючих», «Збереження та експорт конфігурації» та «Перегляд гайду по збірці ПК».

У сутності «Адміністратор» є один варіант використання «Управління даними щодо комплектуючих», використовуючи інтерфейс MAMP.

Також сутність «Адміністратор» має можливість використовувати варіанти використання сутності «Користувач».

Варіант використання «Управління даними щодо комплектуючих» має розширення: Створити компонент, знайти компонент, оновити компонент та видалити компонент.

Реалізація варіанту використання «Управління даними щодо комплектуючих» у інтерфейсі MAMP має зображено на рисунках 3.11 - 3.13.

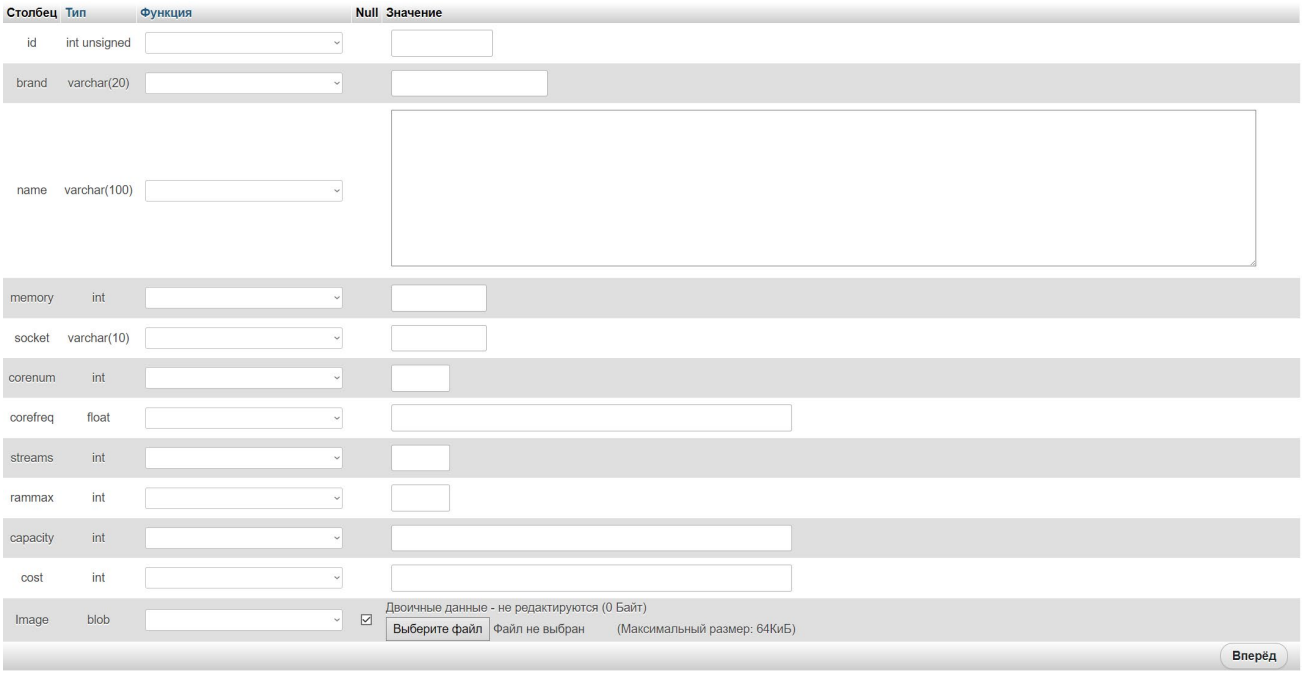


Рис. 3.11 Створення нового компоненту CPU

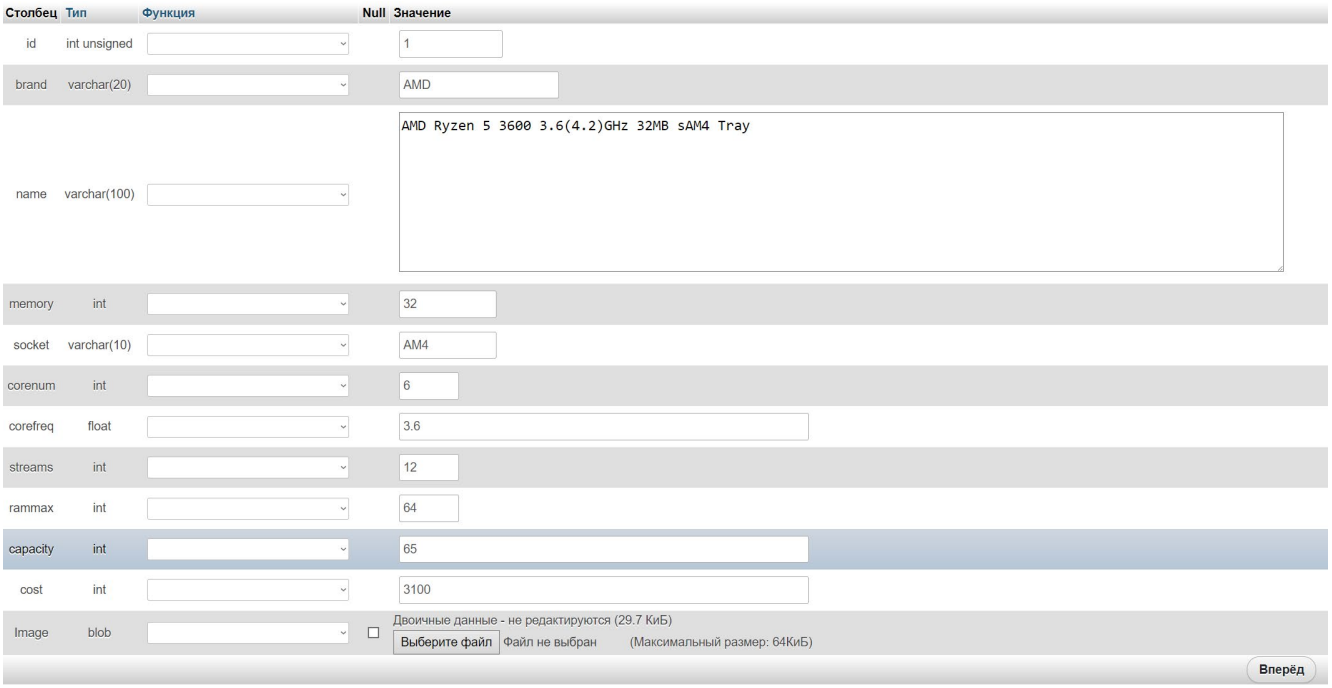


Рис. 3.12 Редагування компоненту CPU

Рис. 3.13 Перегляд компонентів CPU

3.3 Нефункціональні вимоги

У даному розділі описано всі вимоги які не увійшли до опису варіантів використання, а саме: Надійність, продуктивність, зручність використання, можливість розширення та інші.

3.3.1 Надійність

Необхідно забезпечити безвідмовність системи на будь які запити користувачів та можливість резервного копіювання при критичному завершенні програми.

3.3.2 Продуктивність

Застосунок повинен забезпечувати середній час відгуку не більше 2 секунд для всіх основних функцій (пошук компонентів, перевірка сумісності, експорт та

інше).

3.3.3 Зручність використанням

Інтерфейс користувача повинен бути інтуїтивно зрозумілий та мати мінімалістичний дизайн.

3.3.4 Можливість розширення

Застосунок повинен мати змогу до розширення функціоналу без переробки основних функцій та можливість обробляти нові дані у базі даних.

3.3.5 Інші вимоги

Застосунок повинен відповідати всім вимогам законодавства України та враховувати всі необхідні нормативно-правові акти.

3.4 Модель проектування

3.4.1 Проектування діяльності

Діаграма діяльності (Activity Diagram) — це один з типів діаграм у мові моделювання UML , який використовується для моделювання динамічних аспектів систем. Вона відображає послідовність дій або потік контролю в системі або бізнес-процесі. Діаграма діяльності часто використовується для опису поведінки об'єктів і взаємодії між ними.

Діаграма діяльності підбору комплектуючих має наступний вигляд на рисунку 3.14

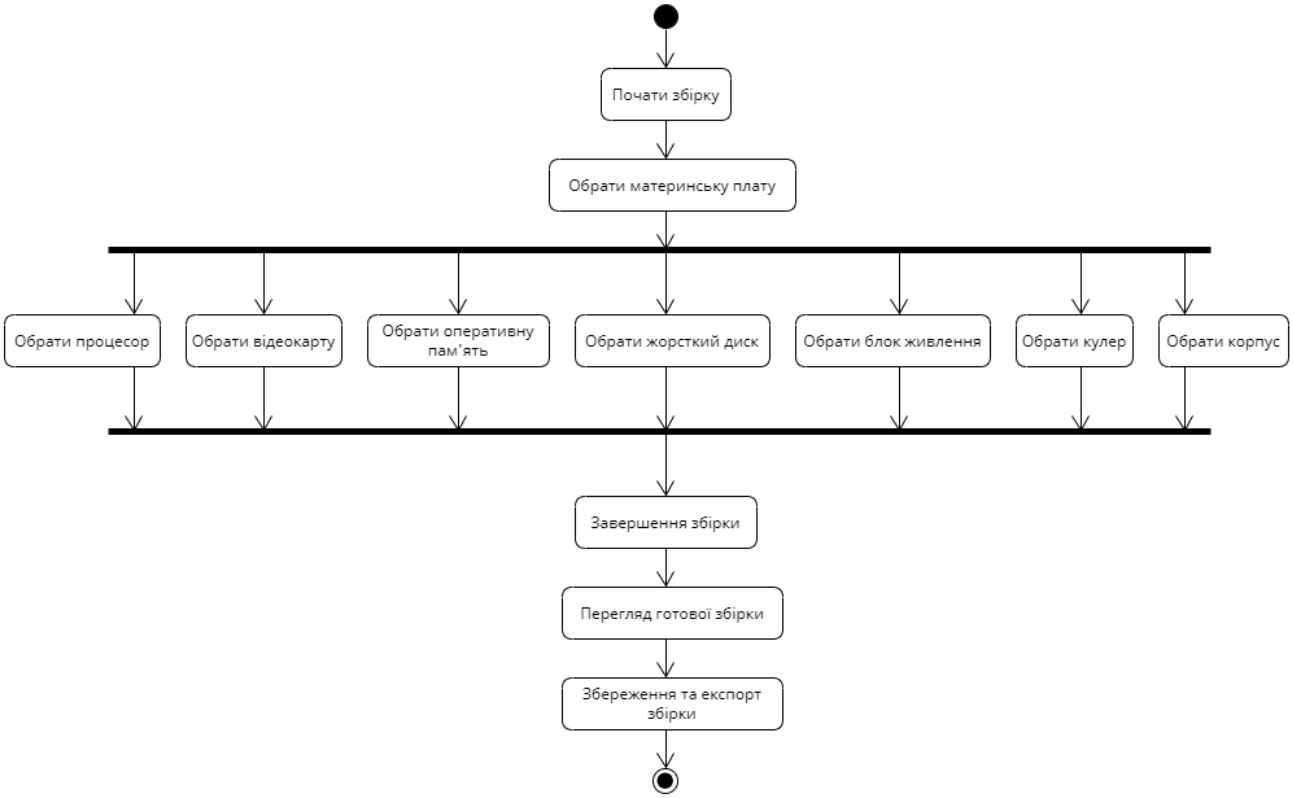


Рис. 3.14 Діаграма діяльності підбору комплектуючих

Дана діаграма описує процес створення нової збірки користувачем і можливості її збереження та експорту.

Розглянемо детальніше, як працює цей процес за кроками:

1. Початок
2. Початок збірки
3. Вибір материнської плати
4. Паралельно з цим потрібно обрати процесор, відеокарту, оперативну пам’ять, жорсткий диск, блок живлення, кулер та корпус
5. Паралельні процеси переходять в завершення збірки
6. Перегляд нової збірки
7. Збереження та експорт збірки
8. Кінець

У даній діаграмі немає розгалужень помилок в збірці, бо в сама програма

має алгоритм, який протидіє помилкам та показує тільки сумісні компоненти до материнської плати.

Діаграма діяльності окремого підбору материнської плати та процесору має наступний вигляд на рисунку 3.15.

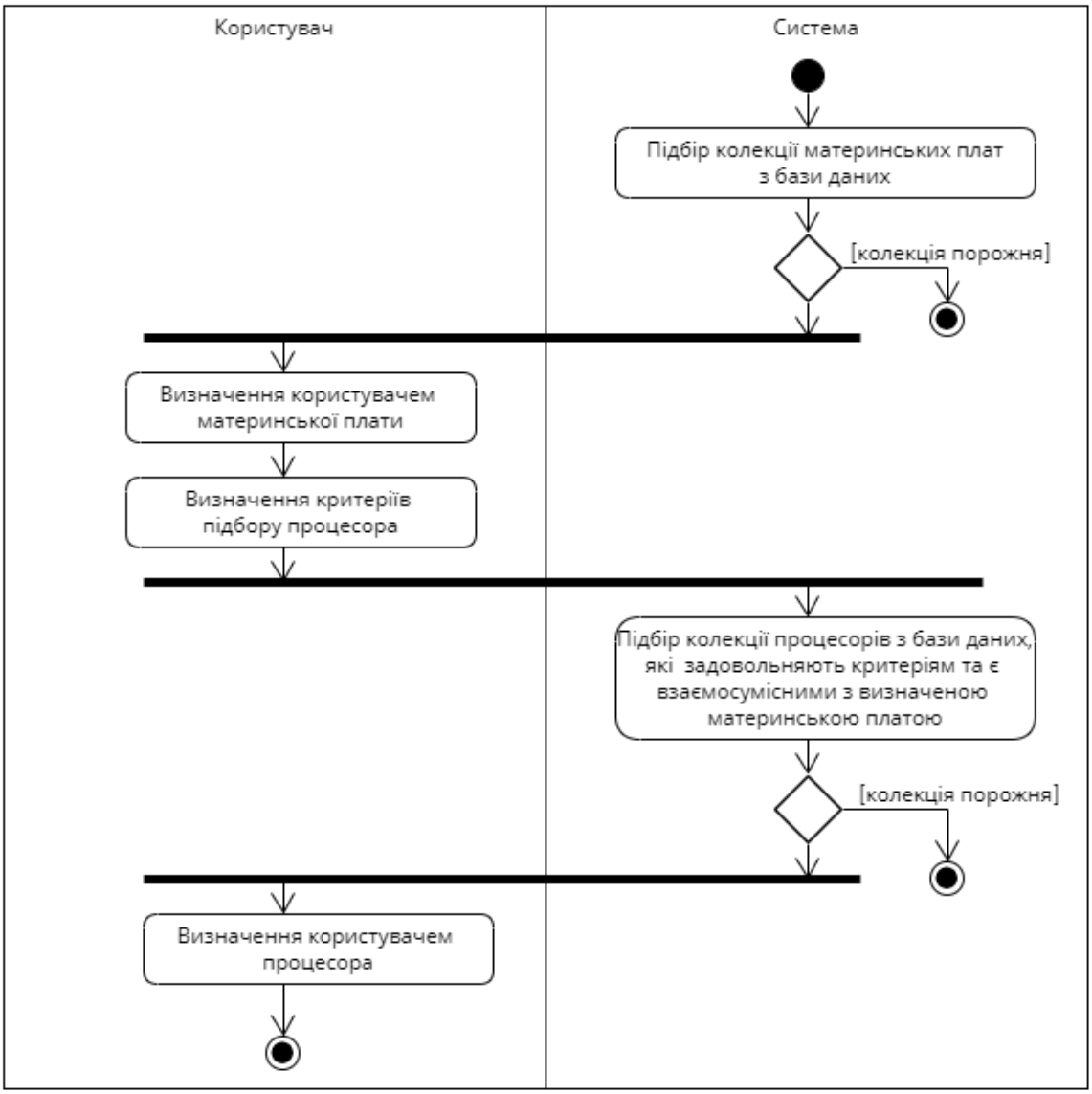


Рис. 3.15 Діаграма діяльності підбору материнської плати та процесора

На даній діаграмі видно, що система підбирає колекції материнських плат з бази даних, та завантажує її списком до інтерфейсу користувача. Далі користувач обирає потрібну йому материнську плату. Програма підбирає за критеріями та сумісності процесори, на основі обраної материнської плати, з бази даних, та

завантажує її списком до інтерфейсу користувача.

Дана діаграма є аналогічною і до інших компонентів системи, таких як: Відеокарта, Оперативна пам’ять, Диски, Корпус, Кулер та Блок Живлення

Реалізація діаграм діяльності підбору комплектуючих у вигляді коду має наступний вигляд на рисунках 3.16 — 3.18.

```
Ссылок: 1
private void LoadMotherBoardList()
{
    DB db = new DB();
    List<MotherBoard> motherBoardList = db.GetMotherBoard();

    MotherboardList.DisplayMember = "Name";
    MotherboardList.ValueMember = "ID";
    MotherboardList.DataSource = motherBoardList;
}
```

Рис. 3.16 Підбір колекції материнської плати

```
Ссылок: 1
private void LoadCPUList()
{
    List<CPU> cpuList = SortedCPUByMotherboard();

    CPUList.DisplayMember = "Name";
    CPUList.ValueMember = "ID";
    CPUList.DataSource = cpuList;
}
```

Рис. 3.17 Підбір колекції процесора

3.4.2 Проектування послідовності

Діаграма послідовності (Sequence Diagram) є одним з типів діаграм, що використовуються в мові моделювання UML. Вона описує, як об'єкти взаємодіють один з одним у певній послідовності дій. Основне призначення діаграм послідовності — моделювати потік повідомлень між об'єктами в рамках

сценарію або операції.

Основними компонентами діаграми послідовності є: Актори, Об’єкти, Лінії життя, Повідомлення, Активність та Фрагменти.

Діаграма послідовності підбору материнської плати має наступний вигляд на рисунку 3.19

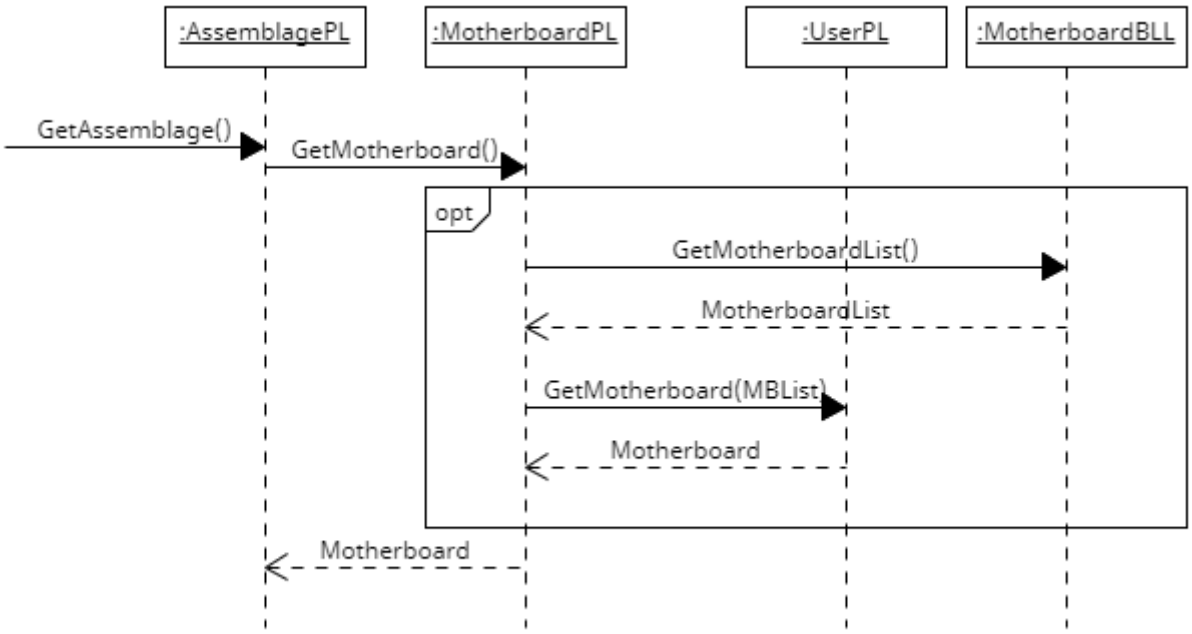


Рис. 3.19 Діаграма послідовності підбору материнської плати

На даній діаграмі видно, що коли викликається метод GetAssemblage(), тобто початок збірки, тоді він викликає метод GetMotherboard(), який відкривається в класі MotherboardPL та завантажує список материнських плат з класу MotherboardBLL. Цей список використовується в класі UserPL, де користувач обирає зі списку потрібну материнську плату. Дана плата і стає обраною користувачем і повертається до класу AssemblagePL.

Реалізація діаграми послідовності підбору материнської плати у вигляді коду має наступний вигляд на рисунках 3.20 - 3.21

```
public List<MotherBoard> GetMotherBoardList()
{
    List<MotherBoard> motherBoardList = new List<MotherBoard>();

    openConnection();

    string query = "SELECT * FROM motherboard";
    using (MySqlCommand cmd = new MySqlCommand(query, connection))
    {
        using (MySqlDataReader reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                motherBoardList.Add(new MotherBoard
                {
                    ID = reader.GetInt32("id"),
                    Brand = reader.GetString("brand"),
                    Name = reader.GetString("name"),
                    Socket = reader.GetString("socket"),
                    Chipset = reader.GetString("chipset"),
                    FormFactor = reader.GetString("formfactor"),
                    RAMType = reader.GetString("ramtype"),
                    RAMMax = reader.GetInt32("rammax"),
                    RAMNum = reader.GetInt32("ramnum"),
                    RAMFreq = reader.GetFloat("ramfreq"),
                    MaxLAN = reader.GetInt32("maxlan"),
                    SATANum = reader.GetInt32("satanum"),
                    PCINum = reader.GetInt32("pcinum"),
                    M2Num = reader.GetInt32("m2num"),
                    CPUPowerType = reader.GetString("cpupowertype"),
                    MotherboardPowerType = reader.GetString("motherboardpowertype"),
                    Cost = reader.GetInt32("cost"),
                    Image = reader.IsDBNull(reader.GetOrdinal("Image")) ? null : reader["image"] as byte[]
                });
            }
        }
    }
    closeConnection();
    return motherBoardList;
}
```

Рис. 3.20 Реалізація діаграми послідовності підбору материнської плати

```
private MotherBoard GetMotherBoard()
{
    return MotherboardList.SelectedItem as MotherBoard;
}
```

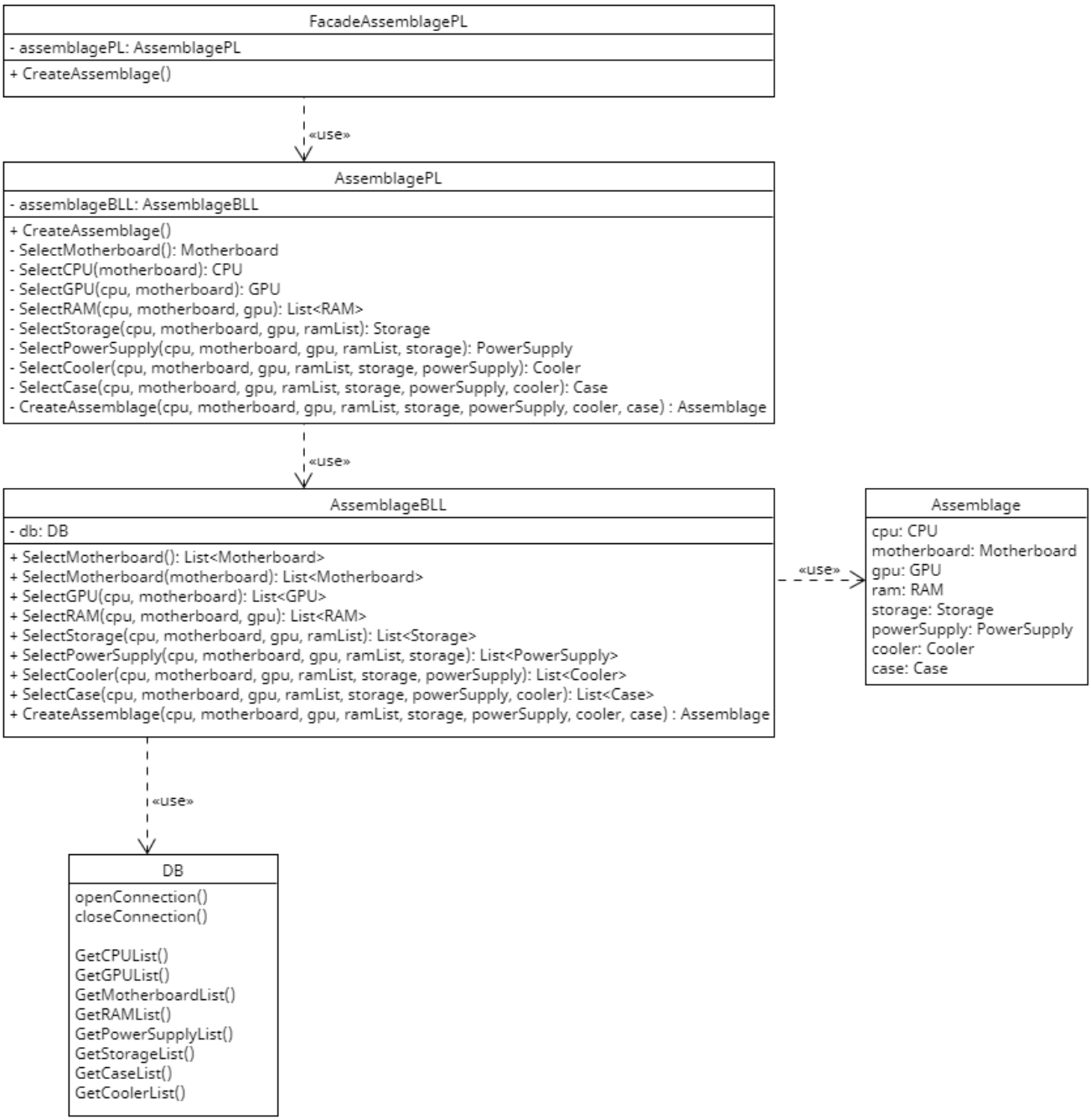
Рис. 3.21 Реалізація діаграми послідовності підбору материнської плати

3.4.3 Проектування класів

Діаграма класів — це один з типів діаграм, використовуваних в об'єктно-орієнтованому моделюванні для зображення структури системи. Вона є частиною мови UML і використовується для візуалізації класів, їхніх атрибутів, методів та відносин між ними.

Основними компонентами діаграми класів є: Класи, Атрибути, Методи,

Відносини та Мультиплікації.



Діаграма класів повної збірки має наступний вигляд на рисунку 3.22

Рис. 3.22 Діаграма класів повної збірки

Проаналізувавши дану діаграму, можна побачити, що система модульно розподіляє функціональність по окремих класах, таких як AssemblagePL, AssemblageBLL, і DB, що забезпечує легке оновлення та розширення кожного

модуля. Всі основні компоненти комп'ютера (CPU, Motherboard, GPU, RAM, Storage, PowerSupply, Cooler, Case) мають свої відповідні методи вибору в класі AssemblageBLL, що забезпечує високу гнучкість в конфігурації.

Клас AssemblageBLL відповідає за бізнес-логіку збірки, тоді як клас DB відповідає за взаємодію з базою даних. Така архітектура полегшує тестування та налагодження системи, оскільки логіка додатка і доступ до даних розділені на різні рівні.

Клас Assemblage представляє собою комплексний об'єкт, що включає всі необхідні компоненти комп'ютера. Використання класу Assemblage дозволяє системі легко управляти різними конфігураціями комп'ютера, забезпечуючи при цьому цілісність та узгодженість даних.

Методи класу AssemblageBLL використовують методи класу DB для отримання необхідних даних з бази даних, а також об'єднують ці дані для створення об'єкта типу Assemblage. Така взаємодія забезпечує чітке розмежування завдань між різними рівнями системи, що сприяє кращій організації коду.

Реалізація класів DB та Assemblage у вигляді коду зображено на рисунках 3.23 — 3.24

```
namespace PCComponent
{
    Ссылок: 42
    class DB
    {
        MySqlConnection connection = new MySqlConnection("server=localhost; port=3306; username=root; password=Lolukrnet2003; database=pccomponent");

        Ссылок: 0
        public void openConnection()
        {
            if (connection.State == System.Data.ConnectionState.Closed)
            {
                connection.Open();
            }
        }

        Ссылок: 0
        public void closeConnection()
        {
            if (connection.State == System.Data.ConnectionState.Open)
            {
                connection.Close();
            }
        }

        Ссылок: 0
        public MySqlConnection getConnection()
        {
            return connection;
        }
```

Рис. 3.23 Реалізація класу DB

```
internal class Assemblage
{

    CPU cpu { get; set; }

    MotherBoard motherboard { get; set; }

    GPU gpu { get; set; }

    RAM ram { get; set; }

    Storage storage { get; set; }

    PowerSupply powerSupply { get; set; }

    Cooler cooler { get; set; }

    Case _case { get; set; }
}
```

Рис. 3.24 Реалізація класу Assemblage

3.4.4 Проектування станів

Діаграма станів — це графічне представлення поведінки об'єкта в різних станах у відповідь на події. Вона відображає стани об'єкта, переходи між цими станами, а також події або умови, що спричиняють ці переходи. Діаграми станів часто використовуються в розробці програмного забезпечення та систем для моделювання поведінки об'єктів або систем.

Основними компонентами діаграми станів є: Стани, Переходи, Події, Дії та Вкладені Стани.

Діаграма станів меню застосунку має наступний вигляд на рисунку 3.24

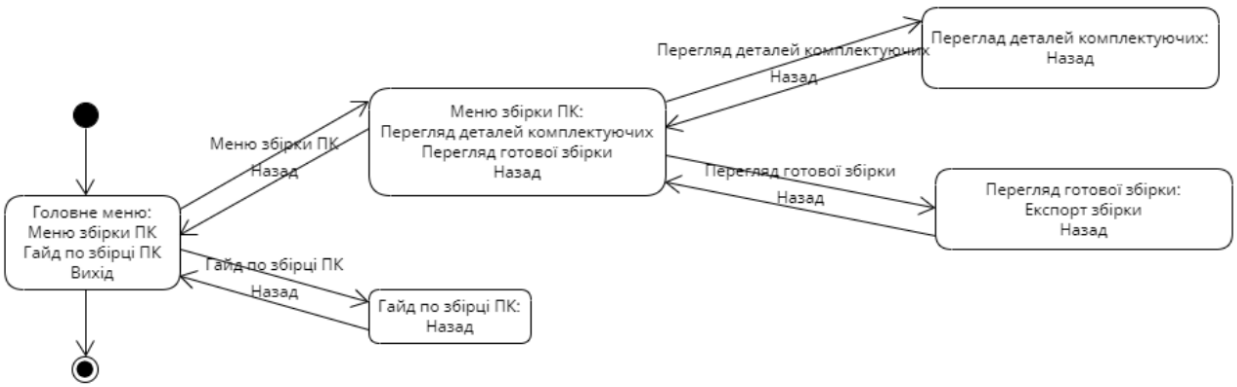


Рис. 3.24 Діаграма станів меню застосунку

Приведена діаграма показує можливість використання програмного продукту від головного меню до повної збірки ПК та його експорту. Найважливішими об’єктами діаграми можна виділити Меню збірки ПК, в якій зібрані всі основні функції (Функції для збірки ПК, Перегляд деталей комплектуючих, Перегляд готової збірки)

Система має централізовану структуру, тобто користувач має змогу потрапити до всіх основних розділів програми через головне меню, яке виступає центральним вузлом програми.

3.5 Архітектура системи

3.5.1 Діаграма артефактів

Діаграма артефактів — це засіб моделювання, що використовується в розробці програмного забезпечення для представлення і опису різних артефактів, які створюються, змінюються або використовуються в процесі розробки. Артефакти можуть включати документи, моделі, код, тести, конфігураційні файли та інші продукти діяльності розробників.

Діаграма артефактів є частиною мови моделювання UML (Unified Modeling Language) і служить для ілюстрації структури артефактів, їх взаємозв'язків та залежностей. Вона допомагає розробникам зрозуміти, як різні частини системи пов'язані між собою і як вони взаємодіють.

Основними компонентами діаграми артефактів є: Артефакти, Вузли та Залежності.

Діаграма артефактів меню застосунку має наступний вигляд на рисунку 3.25

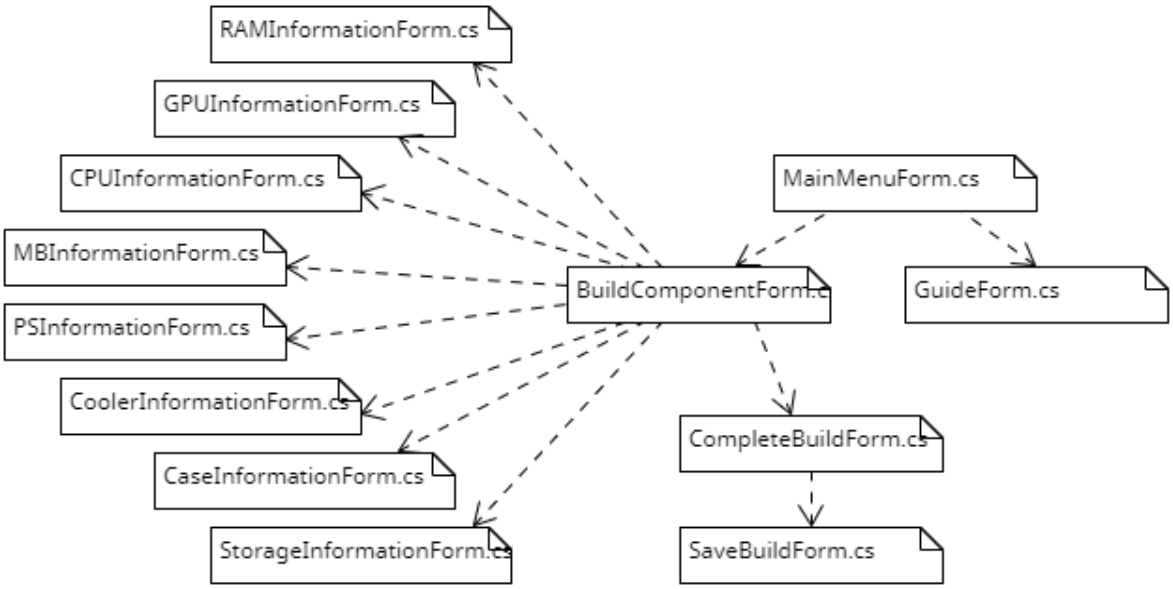


Рис. 3.24 Діаграма артефактів меню застосунку

На цій діаграмі можна побачити архітектуру меню застосунку. До головного меню, яке має назву MainMenuForm.cs належать ще два меню від назвами BuildComponentForm.cs та GuideForm.cs.

До форми BuildComponentForm.cs належать всі форми перегляду інформації про компоненти, такі як RAMInformationForm.cs, GPUInformationForm.cs, CPUInformationForm.cs, MBInformationForm.cs, PSInformationForm.cs, CoolerInformationFrom.cs, CaseInformationFrom.cs та StorageInformationFrom.cs. Також до BuildComponentForm.cs належить форма, яка відповідає за кінцеву збірку комп’ютера, яка має назву CompleteBuildForm.cs.

4 ТЕСТУВАННЯ СИСТЕМИ

Тестування програмного забезпечення (Software Testing) – це процес оцінки та перевірки програмного забезпечення з метою забезпечення його якості, надійності та відповідності вимогам. Основні цілі тестування включають виявлення помилок, перевірку функціональності, продуктивності, безпеки та сумісності програмного забезпечення.

Основними типами тестування програмного забезбечення вважають:

- Модульне тестування: тестування окремих компонентів або модулів програми;
- Інтеграційне тестування: перевірка взаємодії між різними модулями;
- Системне тестування: тестування всієї системи як єдиного цілого;
- Приймальне тестування: перевірка відповідності вимогам замовника та готовності до використання.

Є два методи тестування, це: Статичні методи та Динамічні методи.

Статичні методи тестування (Static Testing) – це методи перевірки програмного забезпечення, які здійснюються без виконання коду. Ці методи орієнтовані на аналіз артефактів проекту, таких як вимоги, дизайн-документація, та вихідний код, з метою виявлення потенційних помилок або невідповідностей на ранніх етапах розробки.

Динамічні методи тестування (Dynamic Testing) – це методи перевірки програмного забезпечення, які здійснюються під час виконання коду. Ці методи орієнтовані на оцінку поведінки системи, її продуктивності, функціональності та інших характеристик в реальному часі.

Тестування системи було виконано власноруч по крокам по всім функціям та взаємодіям.

Висновок по результатам ручного тестування додатку для підбору

комплектуючих до ПК:

Основні функції додатку (вибір комплектуючих, перевірка сумісності, експорт результатів, перегляд інформації про компоненти, перегляд гайду) працюють корректно.

- Всі елементи інтерфейсу правильно відображаються та функціонують
- Інтерфейс користувача інтуїтивно зрозумілий і зручний у використанні.
- Час відгуку додатку оптимальний, затримки в роботі відсутні.

Загалом, додаток продемонстрував високу функціональність та стабільність роботи. Всі основні вимоги виконані, і додаток готовий до використання користувачами.

ВИСНОВКИ

В результаті виконаної роботи було створено десктопний додаток для підбору комплектуючих до комп'ютера мовою програмування C#. У процесі розробки цього програмного рішення були виконані поставлені задачі, а саме:

1. Розроблено структуру даних компонентів, алгоритми підбору комплектуючих та мінімалістичний зручний дизайн
2. Розроблено взаємодію додатку з Базою Даних MySQL
3. Додана значна кількість комплектуючих до бази даних
4. Реалізовано збереження та імпорт збірки
5. Створено модель предметної галузі та варіантів використання
6. Було проведено проектування роботи, послідовності викликів методів, станів та класів.
7. Опрацьовано архітектуру системи
8. Проведено ручне тестування системи, яке забезпечило стабільність роботи програмного забезпечення. Також було забезпечено можливість масштабування системи.

Робота пройшла апробацію на конференціях, за результатом апробації опубліковано тези доповідей:

1. Процун В.С., Замрій І.В. АНАЛІЗ КОНКУРЕНТНОГО СЕРЕДОВИЩА ДЛЯ ЗАСТОСУНКУ ДЛЯ ПІДБОРУ КОМПЛЕКТУЮЧИХ ДО ПК. *Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез*. 24 квітня 2024 року, ДУІКТ, м. Київ, С. 206;
2. Процун В.С., Замрій І.В. ВИЗНАЧЕННЯ ЗАСОБІВ РОЗРОБКИ ЗАСТОСУНКУ ДЛЯ ПІДБОРУ КОМПЛЕКТУЮЧИХ ДО ПК НА МОВІ C#. *Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез*. 24 квітня 2024 року, ДУІКТ, м. Київ, С. 202.

ПЕРЕЛІК ПОСИЛАНЬ

1. Як самостійно зібрати комп’ютер: покрокова інструкція від Foxtrot.ua URL: https://blog.foxtrot.com.ua/pk-i-noutbuky/yak-samostijno-zibrati-kompyuter-pokrokova-instrukcziya.html.

2. Як грамотно підібрати комплектуючі до ПК. URL: https://maxnet.ua/blog/yak-gramotno-pidibrati-komplektuyuchi-dlya-modernizaciyi-pk.

3. How to Build a PC: The Ultimate Guide. URL: https://www.pcmag.com/how-to/how-to-build-a-pc-the-ultimate-beginners-guide.

4. Computer Basics – Inside a Computer. URL: https://edu.gcfglobal.org/en/computerbasics/inside-a-computer/1.

5. Microsoft C# Guide. URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/

6. Інтернет магазин техніки та комплектуючих для ПК Telemart.ua. URL: https://telemart.ua/ua.

7. Інтернет магазин техніки та комплектуючих для ПК Elmir.ua. URL: https://elmir.ua.

8. Інтернет магазин та комплектуючих для ПК IT-BLOK. URL: https://it-blok.com.ua/ua.

9. UMLet - Free UML Tools for fast UML diagrams URL: https://www.umlet.com.

10. Introduction to Visual Studio 2019. URL: https://www.geeksforgeeks.org/introduction-to-visual-studio.

11. What is .NET Framework? URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework.

12. UML Use Case Diagram Tutorial. URL: https://www.lucidchart.com/pages/uml-use-case-diagram.

13. UML Activity Diagram Tutorial. URL: https://www.lucidchart.com/pages/uml-activity-diagram.

14. Lesson 4 – UML – Domain Model. URL: https://www.ictdemy.com/software-design/uml/uml-domain-model.

15. Nonfunctional Requirements in Software Engineering: Examples, Types, Best Practices. URL: https://www.altexsoft.com/blog/non-functional-requirements.

16. What is Sequence Diagram? URL:

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram.

17. UML Class Diagram Tutorial. URL: https://www.lucidchart.com/pages/uml-class-diagram.

18. Діаграма станів. *Кафедра ММСА КПІ.* URL: http://mmsa.kpi.ua/sites/default/files/disciplines/Розробка%20і%20тестування%20програм/didkovska_m_v_testing_lecture_4.pdf.

19. UML Artifact. URL: https://www.uml-diagrams.org/artifact.html.

20. What is Software Testing? URL: https://www.geeksforgeeks.org/software-testing-basics/.

21. Software Testing Self-Paced. *Epam Campus*. URL: https://training.epam.ua/ua/training/3506.

22. Процун В.С., Замрій І.В. АНАЛІЗ КОНКУРЕНТНОГО СЕРЕДОВИЩА ДЛЯ ЗАСТОСУНКУ ДЛЯ ПІДБОРУ КОМПЛЕКТУЮЧИХ ДО ПК. *Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез*. 24 квітня 2024 року, ДУІКТ, м. Київ, С. 206;

23. Процун В.С., Замрій І.В. ВИЗНАЧЕННЯ ЗАСОБІВ РОЗРОБКИ ЗАСТОСУНКУ ДЛЯ ПІДБОРУ КОМПЛЕКТУЮЧИХ ДО ПК НА МОВІ C#. *Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез*. 24 квітня 2024 року, ДУІКТ, м. Київ, С. 202.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Створення застосунку для підбору комплектуючих для комп’ютера мовою C#

Виконав студент 4 курсу
групи ПД-42
Процун Володимир Сергійович
Керівник роботи
Доктор філософських наук (PhD), доцент кафедри ІПЗ Гребенюк Віктор Вікторович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - спрощення підбору комплектуючих до персонального комп’ютера
- **Об’єкт дослідження** - процес прийняття рішення для підбору комплектуючих до персонального комп’ютера
- **Предмет дослідження** - програмне забезпечення для підбору комплектуючих до персонального комп’ютера

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Аналіз існуючих застосунків для підбору комплектуючих до ПК
2. Вивчення потреб користувачів у підборі комплектуючих
3. Розробка структури даних та їх взаємодія
4. Розробка алгоритму підбору збірки
5. Розробка архітектури програмного забезпечення
6. Реалізація застосунку для підбору комплектуючих до ПК
7. Тестування програмного продукту

АНАЛІЗ АНАЛОГІВ

Характеристика	Telemart.ua	Elmir.ua	PCComponent
Можливість експорту збірки	-	-	+
Наявність гайдів для збірки	-	-	+
Перевірка сумісності компонентів	+	+	+
Десктопна версія застосунку	-	-	+
Перегляд детальної інформації про компоненти	+	+	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. **Фунціональні вимоги**

   - Здійснення підбору комплектуючих.

   - Збереження та експорт результатів.

   - Алгоритм сумісності комплектуючих.

2. **Нефунціональні вимоги**

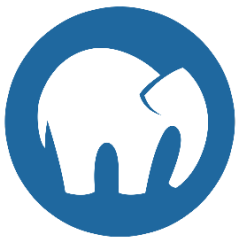
   - Реагування на запити не довже ніж 2 секунди.

   - Зручність використання.

   - Здатність архітектури застосунку підтримувати додавання нового функціоналу.

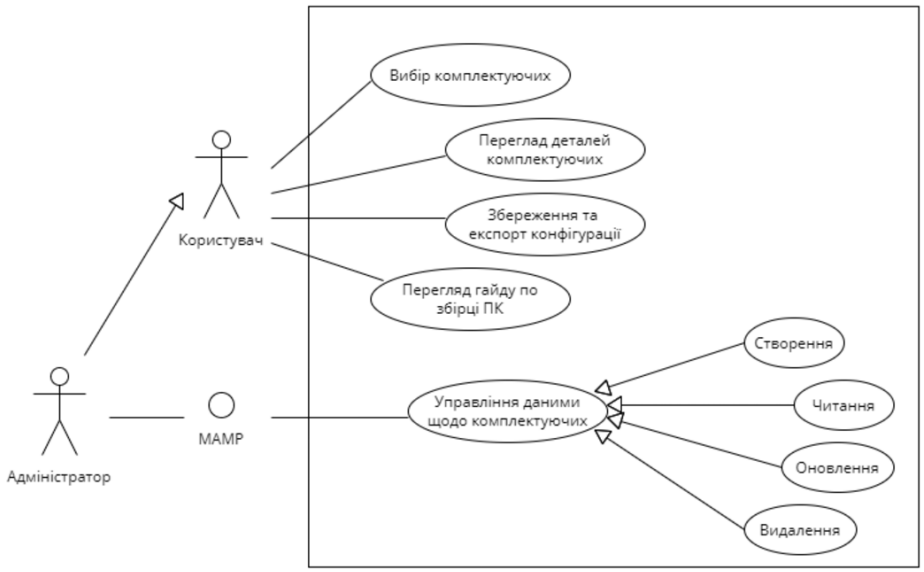
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Visual Studio



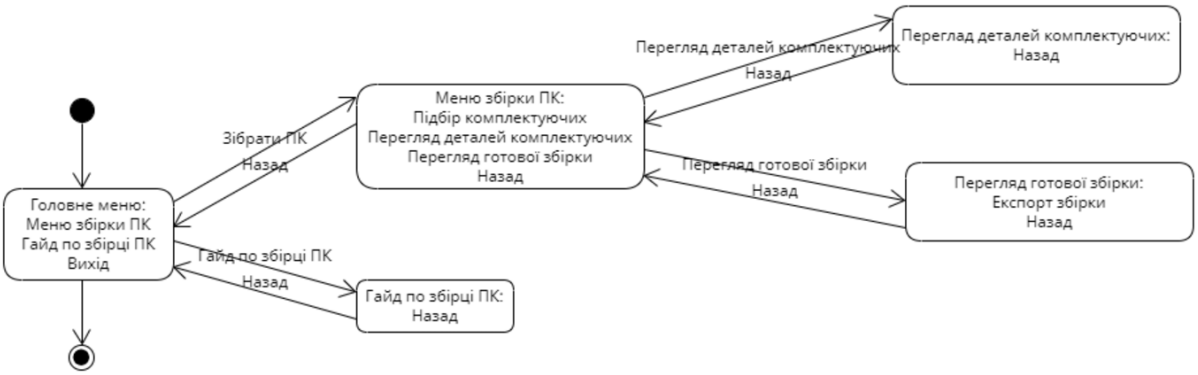
MAMP

Діаграма варіантів використання



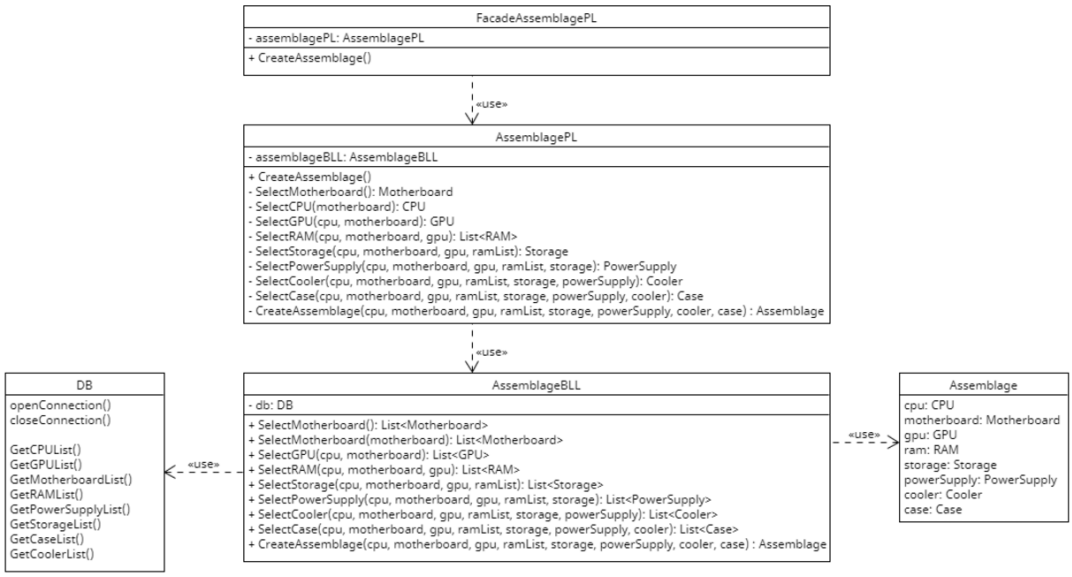
7

Діаграма станів меню



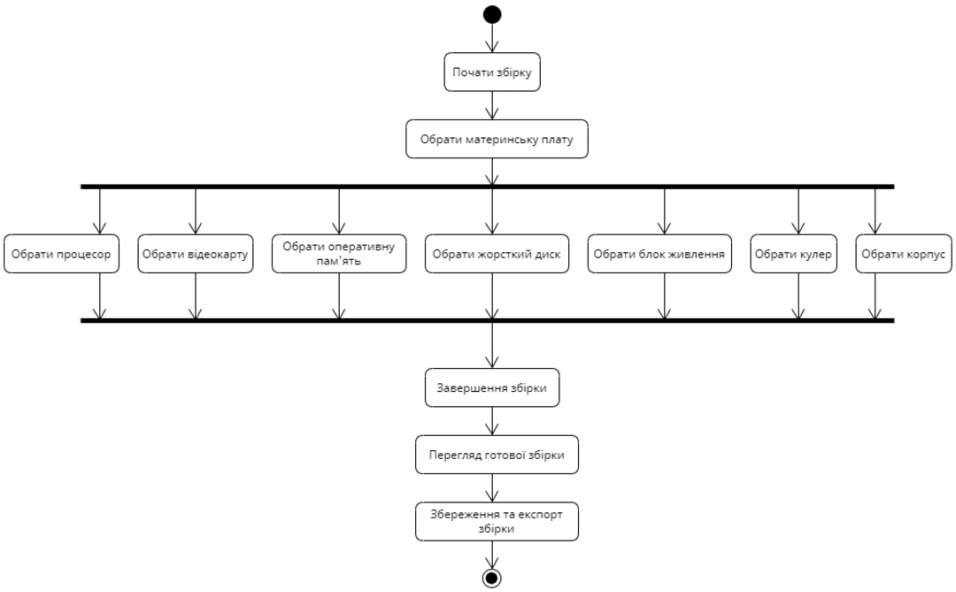
8

Діаграма класів збірки



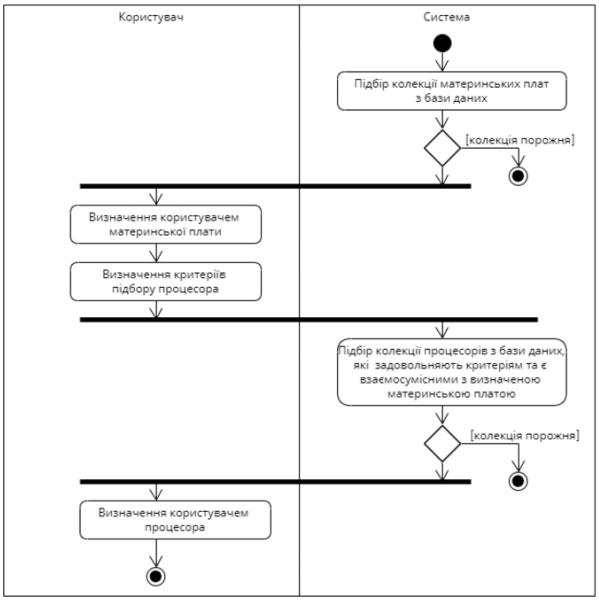
9

Діаграма діяльності підбору комплектуючих



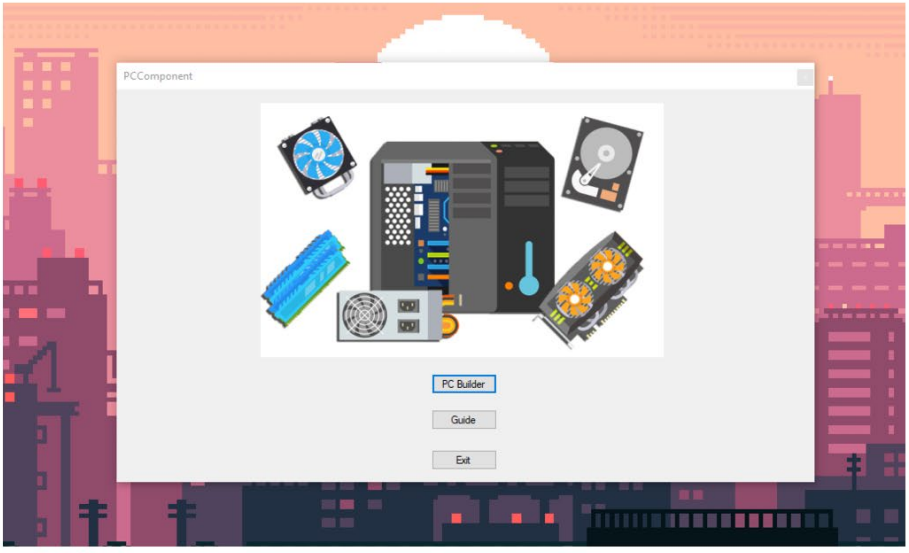
10

Діаграма діяльності підбору материнської плати



11

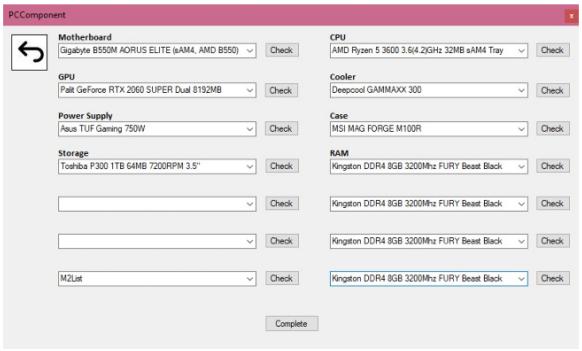
Відео роботи застосунку



12

ЕКРАННІ ФОРМИ

Головне меню



Меню збірки ПК

ЕКРАННІ ФОРМИ

Інформація про компонент

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Процун В.С., Замрій І.В. АНАЛІЗ КОНКУРЕНТНОГО СЕРЕДОВИЩА ДЛЯ ЗАСТОСУНКУ ДЛЯ ПІДБОРУ КОМПЛЕКТУЮЧИХ ДО ПК. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2024 року, ДУІКТ, м. Київ, С. 206.

Процун В.С., Замрій І.В. ВИЗНАЧЕННЯ ЗАСОБІВ РОЗРОБКИ ЗАСТОСУНКУ ДЛЯ ПІДБОРУ КОМПЛЕКТУЮЧИХ ДО ПК НА МОВІ C#. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2024 року, ДУІКТ, м. Київ, С. 202.

15

ВИСНОВКИ

1. Проаналізовано існуючі рішення для підбору комплектуючих до ПК
2. Проаналізовано потреби користувачів у підборі комплектуючих
3. Розроблено структуру даних компонентів, алгоритми підбору комплектуючих та дизайн
4. Розроблено взаємодію додатку з базою даних MySQL
5. Розроблено архітектуру програмного забезпечення
6. Створено моделі предметної галузі та варіантів використання
7. Розроблено застосунок для підбору комплектуючих до ПК
8. Протестовано кінцевий програмний продукт

16

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

```
internal class MotherBoard
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public string Socket { get; set; }
        public string Chipset { get; set; }
        public string FormFactor { get; set; }
        public string RAMType { get; set; }
        public int RAMMax { get; set; }
        public int RAMNum { get; set; }
        public float RAMFreq { get; set; }
        public int MaxLAN { get; set; }
        public int SATANum { get; set; }
        public int PCINum { get; set; }
        public int M2Num { get; set; }
        public string CPUPowerType { get; set; }
        public string MotherboardPowerType { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
    }
 internal class CPU
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public float Memory { get; set; }
        public string Socket { get; set; }
        public int CoreNum { get; set; }
        public float CoreFreq { get; set; }
        public int Streams { get; set; }
        public int RAMMax { get; set; }
        public int Capacity { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
    }

internal class GPU
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public float Frequency { get; set; }
        public string Family { get; set; }
        public int Memory { get; set; }
        public int MemoryBus { get; set; }
        public string MemoryType { get; set; }
        public float MemoryFreq { get; set; }
        public string GPUPowerType { get; set; }
        public int Capacity { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
    }

internal class RAM
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public int Memory { get; set; }
        public float Frequency { get; set; }
        public string Type { get; set; }
        public int Capacity { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
    }

internal class Storage : IID
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public string Type { get; set; }
        public int Memory { get; set; }
        public int Capacity { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
    }

internal class PowerSupply : IID
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public int Capacity { get; set; }
        public string CPUPowerType { get; set; }
        public string GPUPowerType { get; set; }
        public string MotherboardPowerType { get; set; }
        public int SATANum { get; set; }
        public string Certificate { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
    }

internal class Case : IID
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public string FormFactor { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
```

```
    }

internal class Cooler : IID
    {
        public int ID { get; set; }
        public string Brand { get; set; }
        public string Name { get; set; }
        public string Socket { get; set; }
        public string NoizeLevel { get; set; }
        public int Capacity { get; set; }
        public int Cost { get; set; }
        public byte[] Image { get; set; }
    }

class DB
    {
        MySqlConnection connection = new
MySqlConnection("server=localhost; port=3306;
username=root; password=Lolukrnet2003;
database=pccomponent");

        public void openConnection()
        {
            if (connection.State ==
System.Data.ConnectionState.Closed)
            {
                connection.Open();
            }
        }

        public void closeConnection()
        {
            if (connection.State ==
System.Data.ConnectionState.Open)
            {
                connection.Close();
            }
        }

        public MySqlConnection getConnection()
        {
            return connection;
        }

        public List<CPU> GetCPU()
        {
            List<CPU> cpuList = new List<CPU>();

            openConnection();

            string query = "SELECT * FROM cpu";
            using (MySqlCommand cmd = new
MySqlCommand(query, connection))
            {
                using (MySqlDataReader reader =
cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        cpuList.Add(new CPU
                        {
                            ID = reader.GetInt32("id"),
                            Brand = reader.GetString("brand"),
                            Name = reader.GetString("name"),
                            Memory = reader.GetFloat("memory"),
                            Socket = reader.GetString("socket"),
                            CoreNum = reader.GetInt32("corenum"),
                            CoreFreq = reader.GetFloat("corefreq"),
                            Streams = reader.GetInt32("streams"),
                            RAMMax = reader.GetInt32("rammax"),
                            Capacity = reader.GetInt32("capacity"),
                            Cost = reader.GetInt32("cost"),
                            Image =
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                        });
                    }
                }
            }
            closeConnection();
            return cpuList;
        }

        public List<GPU> GetGPU()
        {
            List<GPU> gpuList = new List<GPU>();

            openConnection();

            string query = "SELECT * FROM gpu";
            using (MySqlCommand cmd = new
MySqlCommand(query, connection))
            {
                using (MySqlDataReader reader =
cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        gpuList.Add(new GPU
                        {
                            ID = reader.GetInt32("id"),
                            Brand = reader.GetString("brand"),
                            Name = reader.GetString("name"),
                            Frequency = reader.GetFloat("frequency"),
```

```
                    Family = reader.GetString("family"),
                    Memory = reader.GetInt32("memory"),
                    MemoryBus =
reader.GetInt32("memorybus"),
                    MemoryType =
reader.GetString("memorytype"),
                    MemoryFreq =
reader.GetFloat("memoryfreq"),
                    GPUPowerType =
reader.GetString("gpupowertype"),
                    Capacity = reader.GetInt32("capacity"),
                    Cost = reader.GetInt32("cost"),
                    Image =
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                });
            }
        }
    }
    closeConnection();
    return gpuList;
}

public List<MotherBoard> GetMotherBoard()
{
    List<MotherBoard> motherBoardList = new
List<MotherBoard>();

    openConnection();

    string query = "SELECT * FROM motherboard";
    using (MySqlCommand cmd = new
MySqlCommand(query, connection))
    {
        using (MySqlDataReader reader =
cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                motherBoardList.Add(new MotherBoard
                {
                    ID = reader.GetInt32("id"),
                    Brand = reader.GetString("brand"),
                    Name = reader.GetString("name"),
                    Socket = reader.GetString("socket"),
                    Chipset = reader.GetString("chipset"),
                    FormFactor =
reader.GetString("formfactor"),
                    RAMType = reader.GetString("ramtype"),
                    RAMMax = reader.GetInt32("rammax"),
                    RAMNum = reader.GetInt32("ramnum"),
                    RAMFreq = reader.GetFloat("ramfreq"),
                    MaxLAN = reader.GetInt32("maxlan"),
                    SATANum = reader.GetInt32("satanum"),
                    PCINum = reader.GetInt32("pcinum"),
                    M2Num = reader.GetInt32("m2num"),
                    CPUPowerType =
reader.GetString("cpupowertype"),
                    MotherboardPowerType =
reader.GetString("motherboardpowertype"),
                    Cost = reader.GetInt32("cost"),
                    Image =
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                });
            }
        }
    }
    closeConnection();
    return motherBoardList;
}

public List<PowerSupply> GetPowerSupply()
{
    List<PowerSupply> powerSupplyList = new
List<PowerSupply>();

    openConnection();

    string query = "SELECT * FROM powersupply";
    using (MySqlCommand cmd = new
MySqlCommand(query, connection))
    {
        using (MySqlDataReader reader =
cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                powerSupplyList.Add(new PowerSupply
                {
                    ID = reader.GetInt32("ID"),
                    Brand = reader.GetString("brand"),
                    Name = reader.GetString("name"),
                    Capacity = reader.GetInt32("capacity"),
                    CPUPowerType =
reader.GetString("cpupowertype"),
                    GPUPowerType =
reader.GetString("gpupowertype"),
                    MotherboardPowerType =
reader.GetString("motherboardpowertype"),
                    SATANum = reader.GetInt32("satanum"),
                    Certificate = reader.GetString("certificate"),
                    Cost = reader.GetInt32("cost"),
                    Image =
```

```
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                    });
                }
            }
        }
        closeConnection();
        return powerSupplyList;
    }

    public List<Storage> GetStorage()
    {
        List<Storage> storageList = new List<Storage>();

        openConnection();

        string query = "SELECT * FROM storage";

        using (MySqlCommand cmd = new
MySqlCommand(query, connection))
        {
            using (MySqlDataReader reader =
cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    storageList.Add(new Storage
                    {
                        ID = reader.GetInt32("ID"),
                        Brand = reader.GetString("brand"),
                        Name = reader.GetString("name"),
                        Type = reader.GetString("type"),
                        Memory = reader.GetInt32("memory"),
                        Capacity = reader.GetInt32("capacity"),
                        Cost = reader.GetInt32("cost"),
                        Image =
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                    });
                }
            }
        }
        closeConnection();
        return storageList;
    }

    public List<RAM> GetRAM()
    {
        List<RAM> ramList = new List<RAM>();

        openConnection();

        string query = "SELECT * FROM ram";
        using (MySqlCommand cmd = new
MySqlCommand(query, connection))
        {
            using (MySqlDataReader reader =
cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    ramList.Add(new RAM
                    {
                        ID = reader.GetInt32("ID"),
                        Brand = reader.GetString("brand"),
                        Name = reader.GetString("name"),
                        Memory = reader.GetInt32("memory"),
                        Frequency = reader.GetFloat("frequency"),
                        Type = reader.GetString("type"),
                        Capacity = reader.GetInt32("capacity"),
                        Cost = reader.GetInt32("cost"),
                        Image =
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                    });
                }
            }
        }
        closeConnection();
        return ramList;
    }

    public List<Case> GetCase()
    {
        List<Case> caseList = new List<Case>();

        openConnection();

        string query = "SELECT * FROM pccase";
        using (MySqlCommand cmd = new
MySqlCommand(query, connection))
        {
            using (MySqlDataReader reader =
cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    caseList.Add(new Case
                    {
                        ID = reader.GetInt32("ID"),
                        Brand = reader.GetString("brand"),
                        Name = reader.GetString("name"),
```

```
                    FormFactor =
reader.GetString("formfactor"),
                    Cost = reader.GetInt32("cost"),
                    Image =
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                });
            }
        }
    }
    closeConnection();
    return caseList;
}

public List<Cooler> GetCooler()
{
    List<Cooler> coolerList = new List<Cooler>();

    openConnection();

    string query = "SELECT * FROM cooler";
    using (MySqlCommand cmd = new
MySqlCommand(query, connection))
    {
        using (MySqlDataReader reader =
cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                coolerList.Add(new Cooler
                {
                    ID = reader.GetInt32("ID"),
                    Brand = reader.GetString("brand"),
                    Name = reader.GetString("name"),
                    Socket = reader.GetString("socket"),
                    NoizeLevel =
reader.GetString("noizelevel"),
                    Capacity = reader.GetInt32("capacity"),
                    Cost = reader.GetInt32("cost"),
                    Image =
reader.IsDBNull(reader.GetOrdinal("Image")) ? null :
reader["image"] as byte[]
                });
            }
        }
    }

    closeConnection();
    return coolerList;
}
}
```

```
public Case GetCaseByID(int id)
    {
        var caseList = db.GetCase();

        foreach (var selectCase in caseList)
        {
            if (selectCase.ID == id)
            {
                return selectCase;
            }
        }

        return null;
    }

public Cooler GetCoolerByID(int id)
    {
        var coolerList = db.GetCooler();

        foreach (var cooler in coolerList)
        {
            if (cooler.ID == id)
            {
                return cooler;
            }
        }

        return null;
    }

public CPU GetCPUByID(int id)
    {
        var cpuList = db.GetCPU();

        foreach (var cpu in cpuList)
        {
            if (cpu.ID == id)
            {
                return cpu;
            }
        }

        return null;
    }

public GPU GetGPUByID(int id)
    {
        var gpuList = db.GetGPU();

        foreach (var gpu in gpuList)
        {
            if (gpu.ID == id)
            {
                return gpu;
            }
        }
```

```
            return null;
        }

public MotherBoard GetMBByID(int id)
        {
            var mbList = db.GetMotherBoard();

            foreach (var motherBoard in mbList)
            {
                if (motherBoard.ID == id)
                {
                    return motherBoard;
                }
            }

            return null;
        }

public PowerSupply GetPSByID(int id)
        {
            var powerSupplyList = db.GetPowerSupply();

            foreach (var powerSupply in powerSupplyList)
            {
                if (powerSupply.ID == id)
                {
                    return powerSupply;
                }
            }

            return null;
        }

 public RAM GetRAMByID(int id)
        {
            var ramList = db.GetRAM();

            foreach (var ram in ramList)
            {
                if (ram.ID == id)
                {
                    return ram;
                }
            }
            return null;
        }

 public Storage GetStorageByID(int id)
        {
            var storageList = db.GetStorage();

            foreach (var storage in storageList)
            {
                if (storage.ID == id)
                {
                    return storage;
                }
            }

            return null;
        }

private void LoadCPUList()
        {
            DB db = new DB();
            List<CPU> cpuList = db.GetCPU();

            CPUList.DisplayMember = "Name";
            CPUList.ValueMember = "ID";
            CPUList.DataSource = cpuList;
        }

        private void LoadGPUList()
        {
            DB db = new DB();
            List<GPU> gpuList = db.GetGPU();

            GPUList.DisplayMember = "Name";
            GPUList.ValueMember = "ID";
            GPUList.DataSource = gpuList;
        }

        private void LoadMotherBoardList()
        {
            DB db = new DB();
            List<MotherBoard> motherBoardList =
db.GetMotherBoard();

            MotherboardList.DisplayMember = "Name";
            MotherboardList.ValueMember = "ID";
            MotherboardList.DataSource = motherBoardList;
        }

        private void LoadPowerSupplyList()
        {
            DB db = new DB();
            List<PowerSupply> powerSupplyList =
db.GetPowerSupply();
            PowerSupplyList.DisplayMember = "Name";
            PowerSupplyList.ValueMember = "ID";
            PowerSupplyList.DataSource = powerSupplyList;
        }

        private void LoadStorageList()
        {
            DB db = new DB();
            List<Storage> storageList = db.GetStorage();
            StorageList.DisplayMember = "Name";
            StorageList.ValueMember = "ID";
            StorageList.DataSource = storageList;
        }

        private void LoadStorageList1()
        {
            DB db = new DB();
```

```
    List<Storage> storageList = db.GetStorage();
    StorageList1.DisplayMember = "Name";
    StorageList1.ValueMember = "ID";
    StorageList1.DataSource = storageList;
}

private void LoadStorageList2()
{
    DB db = new DB();
    List<Storage> storageList = db.GetStorage();
    StorageList2.DisplayMember = "Name";
    StorageList2.ValueMember = "ID";
    StorageList2.DataSource = storageList;
}


private void LoadRAMList()
{
    DB db = new DB();
    List<RAM> ramList = db.GetRAM();
    RAMList.DisplayMember = "Name";
    RAMList.ValueMember = "ID";
    RAMList.DataSource = ramList;
}


private void LoadRAMList1()
{
    DB db = new DB();
    List<RAM> ramList = db.GetRAM();
    RAMList1.DisplayMember = "Name";
    RAMList1.ValueMember = "ID";
    RAMList1.DataSource = ramList;
}


private void LoadRAMList2()
{
    DB db = new DB();
    List<RAM> ramList = db.GetRAM();
    RAMList2.DisplayMember = "Name";
    RAMList2.ValueMember = "ID";
    RAMList2.DataSource = ramList;
}

private void LoadRAMList3()
{
    DB db = new DB();
    List<RAM> ramList = db.GetRAM();
    RAMList3.DisplayMember = "Name";
    RAMList3.ValueMember = "ID";
    RAMList3.DataSource = ramList;
}

private void LoadCaseList()
    {
        DB db = new DB();
        List<Case> caseList = db.GetCase();
        CaseList.DisplayMember = "Name";
        CaseList.ValueMember = "ID";
        CaseList.DataSource = caseList;
    }

    private void LoadCoolerList()
    {
        DB db = new DB();
        List<Cooler> coolerList = db.GetCooler();
        CoolerList.DisplayMember = "Name";
        CoolerList.ValueMember = "ID";
        CoolerList.DataSource = coolerList;
    }

    private MotherBoard GetMotherBoard()
    {
        return MotherboardList.SelectedItem as
MotherBoard;
    }
    private CPU GetCPU()
    {
        return CPUList.SelectedItem as CPU;
    }
    private GPU GetGPU()
    {
        return GPUList.SelectedItem as GPU;
    }
    private Case GetCase()
    {
        return CaseList.SelectedItem as Case;
    }
    private Cooler GetCooler()
    {
        return CoolerList.SelectedItem as Cooler;
    }
    private PowerSupply GetPowerSupply()
    {
        return PowerSupplyList.SelectedItem as
PowerSupply;
    }
    private RAM GetRAM()
    {
        return RAMList.SelectedItem as RAM;
    }
    private Storage GetStorage()
    {
        return StorageList.SelectedItem as Storage;
    }
```