Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

# Пояснювальна записка

До кваліфікаційної роботи бакалавра
За спеціальності 123 – Комп’ютерна інженерія

На тему: ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИБОРУ КОМП’ЮТЕРНИХ КОМПОНЕНТІВ

Проєктував _____ С. С. Буряк

Керівник роботи _____ І. О. Музика

Нормконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2025

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти бакалавр
Спеціальність 123 «Комп'ютерна інженерія»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри, голова циклової комісії

_____А. І. Купін

«____»_____20___року

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Буряк Станіслав Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи_____

Керівник роботи_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "___"_____ 20__року №__

2. Строк подання студентом роботи_____

3. Вихідні дані до роботи_____

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

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. консультанти розділів роботи:

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| 1 | | | |
| 2 | | | |
| 3 | | | |
| | | | |

7. Дата видачі_____

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ Буряк С. С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Музика І. О.
(підпис) (прізвище та ініціали)

# РЕФЕРАТ

Пояснювальна записка: 73 сторінки, 30 рисунків, 1 таблиця, 20 джерел, 5 додатків.

Метою кваліфікаційної роботи є створення програмного забезпечення для вибору комп’ютерних компонентів, яке дозволяє користувачам формувати сумісні конфігурації ПК із використанням локальної бази даних.

Проєкт складається з трьох розділів.

Перший розділ присвячено аналізу існуючих систем для підбору комплектуючих, виявленню їхніх обмежень та обґрунтуванню вибору технологій розробки, серед яких C#, Windows Forms, PostgreSQL і бібліотека Npgsql.

У другому розділі здійснено проєктування архітектури програмного забезпечення, інтерфейсу користувача та структури бази даних із урахуванням вимог до функціональності, зручності, автономності й масштабованості.

Третій розділ містить опис реалізації основних компонентів застосунку, логіки перевірки сумісності, роботи з базою даних, а також результати тестування програми з візуальним супроводом і прикладами вихідного коду.

C#, Database, GPU, Motherboard, .NET Framework, Npgsql, PostgreSQL, Processor, PSU, RAM, SQL, UI Layer, Windows Forms, Модульна архітектура, Нефункціональні та функціональні вимоги, Продуктивність, Стабільність, Трирівнева структура.

Explanatory note: 73 pages, 30 figures, 1 table, 20 sources, 5 appendices.

The purpose of the qualification work is to create software for selecting computer components, which allows users to form compatible PC configurations using a local database.

The project consists of three sections.

The first section is devoted to the analysis of existing systems for selecting components, identifying their limitations and justifying the choice of development technologies, including C#, Windows Forms, PostgreSQL and the Npgsql library.

The second section designs the software architecture, user interface and database structure, taking into account the requirements for functionality, convenience, autonomy and scalability.

The third section contains a description of the implementation of the main components of the application, the logic of checking compatibility, working with the database, as well as the results of testing the program with visual support and examples of source code.

C#, Database, GPU, Modular architecture, Motherboard, .NET Framework, Npgsql, Non-functional and functional requirements, Performance, PostgreSQL, Processor, PSU, RAM, SQL, Stability, Three-tier structure, UI Layer, Windows Forms.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

**AM4/5** - Сокет для процесорів AMD (AMD Socket 4/5)
**API** - Інтерфейс програмування додатків (Application Programming Interface)
**ATX** - Стандарт форм-фактора (Advanced Technology Extended)
**CPU** – Центральний процесор (Central Processing Unit)
**DDR4**/5 - Тип оперативної пам’яті 4-го покоління (Double Data Rate 4/5)
**GPU** – Графічний процесор (Graphics Processing Unit)
**HDD** – Жорсткий диск (Hard Disk Drive)
**IDE** – Інтегроване середовище розробки (Integrated Development Environment)
**LGA** – Контактний масив (Land Grid Array)
**NVMe** – (Non-Volatile Memory Express)
**PCIe** – Інтерфейс периферійних пристроїв (Peripheral Component Interconnect Express)
**PSU** – Блок живлення (Power Supply Unit)
**RAM** – Оперативна пам’ять (Random Access Memory)
**SQL** - Мова структурованих запитів (Structured Query Language)
**SSD** – Твердотільний накопичувач (Solid State Drive)
**TDP** - Тепловиділення (Thermal Design Power)
**UI** - Користувацький інтерфейс (User Interface)
**VRAM** - Відеопам’ять (Video Random Access Memory)
**БД** – База даних
**ПЗ** – Програмне забезпечення
**ПК** – Персональний комп’ютер
**СУБД** – Система управління базами даних

# ВСТУП

Сучасне суспільство дедалі більше залежить від інформаційних технологій, і комп'ютерна техніка стала невід'ємною частиною повсякденного життя людини. Комп'ютери використовуються в освіті, медицині, виробництві, наукових дослідженнях, розвагах та повсякденній діяльності. Водночас, зростає потреба у точному підборі та конфігурації комп'ютерних систем відповідно до конкретних вимог користувача. Від правильного вибору компонентів залежить ефективність роботи комп'ютера, його продуктивність, енергоспоживання, сумісність та надійність.

Більшість користувачів, які прагнуть зібрати власний комп'ютер, зіштовхуються з проблемою складності вибору сумісних комплектуючих. Різноманіття процесорів, материнських плат, оперативної пам'яті, відеокарт та інших компонентів ускладнює цей процес навіть для досвідчених користувачів. Крім того, важливо враховувати технічні характеристики, стандарти, ціновий діапазон, енергоспоживання та інші параметри, що впливають на сумісність і доцільність обраних елементів.

Незважаючи на існування низки онлайн-сервісів, таких як PCPartPicker або локальні каталоги (наприклад, Hotline.ua, Rozetka.ua), не всі з них пропонують достатній рівень гнучкості, інтерактивності та зручності використання. Часто такі системи орієнтовані на ринок певної країни, мають обмежений набір функцій або не дозволяють повноцінно фільтрувати компоненти за ключовими характеристиками. Це створює потребу в розробці нових, інтуїтивно зрозумілих програмних засобів, які б дозволяли легко і швидко підібрати комп'ютерні комплектуючі, враховуючи їхню сумісність.

У зв'язку з цим тема кваліфікаційної роботи є актуальною, оскільки передбачає створення спеціалізованого програмного забезпечення, яке дозволить автоматизувати процес підбору комп'ютерних компонентів відповідно до технічних вимог та бюджету користувача. Така система має бути зручною у використанні, мати візуальний інтерфейс, забезпечувати фільтрацію за різними параметрами та перевірку сумісності обраних елементів.

Для реалізації поставленої задачі у межах даної роботи було обрано мову програмування C# та технологію Windows Forms для розробки графічного інтерфейсу користувача. Також для зберігання та обробки інформації про комп’ютерні компоненти буде використано реляційну систему управління базами даних PostgreSQL. Такий вибір забезпечує гнучкість і розширюваність програмного забезпечення, а також дозволяє використовувати надійні інструменти для роботи з даними.

Мета роботи - розробити програмне забезпечення для зручного вибору сумісних комп’ютерних компонентів з використанням мови програмування C# та бази даних PostgreSQL.

**Завдання роботи:**

- Провести аналіз існуючих рішень у сфері підбору комп’ютерних компонентів.
- Визначити вимоги до функціональності програмного забезпечення.
- Спроєктувати архітектуру системи та базу даних.
- Реалізувати графічний інтерфейс користувача засобами Windows Forms.
- Розробити механізм перевірки сумісності компонентів.
- Провести тестування програмного продукту.
- Сформулювати висновки щодо ефективності реалізованої системи.

# 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

## 1.1. Аналіз існуючих систем підбору комп'ютерних компонентів

У сфері персонального складання комп'ютерів (DIY – Do It Yourself) дедалі більше користувачів звертаються до онлайн-інструментів, які спрощують процес вибору та оцінки сумісності комп'ютерних комплектуючих. Такі платформи дозволяють не лише підібрати оптимальний набір компонентів для збірки ПК, але й врахувати їх технічну сумісність, ціни та доступність на ринку. Серед найвідоміших і найпоширеніших інструментів можна виділити такі: **PCPartPicker**, **Telemart.ua**, **Hotline.ua** та **e-Katalog**. Нижче наведено детальний аналіз кожного з них, з акцентом на їх функціональні можливості, переваги, недоліки та доцільність використання, особливо з урахуванням потреб українського користувача.

- **PCPartPicker [7]**

Це провідний міжнародний сервіс для підбору комп'ютерних комплектуючих, який широко використовується у США, Європі та інших регіонах. Платформа дозволяє створювати повноцінні конфігурації ПК, перевіряти сумісність компонентів, порівнювати ціни з різних магазинів і ділитися збірками зі спільнотою. Вона орієнтована на користувачів різного рівня підготовки – від новачків до ентузіастів.

**Особливості**

PCPartPicker є повноцінним конфігуратором із глибоким аналізом сумісності. Наприклад, платформа автоматично перевіряє, чи сумісний сокет процесора з материнською платою, чи достатньо потужності блоку живлення для обраної відеокарти, а також чи відповідає форм-фактор материнської плати корпусу. Крім того, вона інтегрується з великою кількістю міжнародних магазинів, що дозволяє користувачам бачити актуальні ціни та наявність товарів.

**Функціональні можливості**

- Створення повноцінних конфігурацій ПК із вибором усіх основних компонентів (процесор, материнська плата, оперативна пам'ять, відеокарта, блок живлення, корпус, накопичувачі, охолодження).
- Автоматична перевірка сумісності (сумісність сокета, форм-фактора, потужності блоку живлення, розміру компонентів).

- Порівняння цін із різних магазинів (здебільшого американських і європейських).
- Збереження та порівняння збірок, можливість ділитися ними з іншими користувачами через форуми.
- Рекомендації компонентів на основі бюджету та потреб (наприклад, для геймінгу, роботи чи офісних завдань).
- Оцінка загального енергоспоживання (Wattage) конфігурації.

**Переваги**

- Глибока перевірка сумісності, що зменшує ризик помилок при виборі компонентів.
- Широка база даних із комплектуючими від провідних світових брендів (Intel, AMD, NVIDIA, Corsair тощо).
- Інтеграція зі спільнотою: користувачі можуть переглядати готові збірки, залишати відгуки та отримувати поради.
- Зручний і сучасний інтерфейс із детальними технічними характеристиками компонентів.
- Можливість зберігати та порівнювати кілька збірок.

**Недоліки**

- Обмежена орієнтація на український ринок: ціни вказані в доларах або євро, а магазини переважно з США та Європи, що ускладнює доставку в Україну.
- Відсутність локальних даних про наявність товарів в українських магазинах.
- Потребує ручного введення цін із локальних джерел для українських користувачів.
- Англійськомовний інтерфейс, що може бути незручним для користувачів, які не володіють мовою.

**Доцільність для українського користувача**

PCPartPicker є ідеальним інструментом для ознайомлення з глобальними трендами, технічними характеристиками та створення збірок із високим рівнем сумісності. Однак для практичного використання в Україні платформа вимагає додаткових зусиль: користувачам потрібно самостійно перевіряти наявність компонентів у місцевих магазинах і конвертувати ціни в гривні. Це робить її більш придатною для досвідчених користувачів, які готові витратити час на додаткові перевірки.

- **Telemart.ua [8]**

Telemart.ua – це український інтернет-магазин комп'ютерної техніки, який пропонує інтегрований інструмент для підбору сумісних комплектуючих. Платформа орієнтована на продаж товарів, але має базові функції конфігуратора, що дозволяють підбирати сумісні компоненти, наприклад, материнські плати до обраного процесора.

**Особливості**

Telemart.ua не є універсальним конфігуратором у повному розумінні. Її основна мета – спростити процес покупки, а не забезпечити детальний технічний аналіз. Платформа пропонує базовий підбір компонентів, враховуючи основні параметри сумісності (наприклад, сокет або тип пам'яті), але не проводить глибокої перевірки технічних деталей.

**Функціональні можливості**

- Підбір материнської плати до обраного процесора (на основі сумісності сокета).
- Підбір оперативної пам'яті до материнської плати (враховує тип пам'яті, наприклад, DDR4).
- Автоматичне оновлення загальної ціни конфігурації в процесі вибору компонентів.
- Можливість одразу перейти до оформлення замовлення повної збірки.

**Переваги**

- Локалізований інтерфейс українською мовою, що робить платформу зручною для місцевих користувачів.
- Актуальні ціни в гривнях, що відповідають реаліям українського ринку.
- Простий і інтуїтивно зрозумілий підбір компонентів, який підходить для новачків.
- Можливість швидкого оформлення покупки з доставкою по Україні.

**Недоліки**

- Відсутність глибокої перевірки сумісності: платформа не враховує такі параметри, як кількість слотів для пам'яті, частота пам'яті, TDP процесора чи потужність блоку живлення.
- Мінімум технічних пояснень: користувач не отримує попереджень про можливі проблеми сумісності.
- Немає можливості зберігати або порівнювати кілька збірок.
- Обмежена фільтрація за характеристиками (немає гнучких фільтрів за частотою пам'яті, форм-фактором тощо).
- Асортимент обмежений порівняно з міжнародними платформами.

**Доцільність для українського користувача**

Telemart.ua є зручним рішенням для новачків, які хочуть швидко підібрати базові комплектуючі та оформити покупку. Локалізація та актуальні ціни роблять платформу практичною для українського ринку. Однак через відсутність глибокої перевірки сумісності та обмежений функціонал вона менш придатна для створення складних або високопродуктивних збірок.

- **Hotline.ua [9]**

Hotline.ua – це український сервіс порівняння цін, який агрегує пропозиції від різних інтернет-магазинів по всій Україні. Платформа дозволяє шукати комп'ютерні комплектуючі, порівнювати ціни, читати відгуки та перевіряти акції. Вона більше орієнтована на порівняння пропозицій, ніж на створення повноцінних конфігурацій.

**Особливості**

Hotline.ua не є конфігуратором у прямому сенсі, але допомагає користувачам знайти комплектуючі, порівняти їх характеристики та вибрати найвигіднішу пропозицію. Платформа не перевіряє сумісність компонентів, але пропонує зручний пошук і фільтрацію товарів.

**Функціональні можливості**

- Пошук комплектуючих із фільтрацією за ціною, брендом, характеристиками (наприклад, обсяг пам'яті, частота).
- Порівняння цін від різних магазинів, включно з локальними українськими.
- Перегляд відгуків покупців і рейтингів товарів.
- Сортування за популярністю, ціною чи акційними пропозиціями.
- Інформація про наявність товарів у магазинах.

**Переваги**

- Широкий вибір магазинів, включно з локальними українськими, що забезпечує актуальні ціни в гривнях.
- Зручний пошук і базова фільтрація за характеристиками.
- Наявність відгуків, які допомагають оцінити якість товарів.
- Можливість швидко знайти найвигіднішу пропозицію завдяки порівнянню цін.

**Недоліки**

- Відсутність автоматичної перевірки сумісності між компонентами.
- Інформація залежить від даних магазинів, які можуть бути неточними (наприклад, застаріла інформація про наявність).
- Фокус на порівнянні цін, а не на створенні цілісної збірки.
- Обмежена інформативність щодо технічних деталей (немає попереджень про сумісність).

**Доцільність для українського користувача**

Hotline.ua є відмінним інструментом для моніторингу цін і вибору постачальника в Україні. Він дозволяє швидко знайти комплектуючі за вигідною ціною, але вимагає від користувача самостійної перевірки сумісності. Платформа підходить для тих, хто вже має уявлення про потрібну конфігурацію і шукає найкращу ціну.

- **e-Katalog [10]**

e-Katalog – ще один український сервіс порівняння цін, який об'єднує пропозиції від тисяч магазинів, включно з локальними та міжнародними. Платформа дозволяє порівнювати характеристики комплектуючих, перевіряти ціни, читати відгуки та використовувати фільтри для пошуку.

**Особливості**

e-Katalog подібний до Hotline.ua за своєю функціональністю, але має ширшу базу даних і кращі можливості для порівняння технічних характеристик. Проте, як і Hotline.ua, він не є конфігуратором і не перевіряє сумісність компонентів.

**Функціональні можливості**

- Пошук комплектуючих із фільтрацією за параметрами (ціна, бренд, технічні характеристики).
- Порівняння характеристик між кількома компонентами (наприклад, порівняння двох відеокарт).
- Перегляд цін у гривнях від різних магазинів.
- Читання відгуків і перегляд рейтингів товарів.
- Рекомендації схожих товарів на основі обраного компонента.

**Переваги**

- Велика база даних із пропозиціями від локальних і міжнародних магазинів.
- Зручний інтерфейс для порівняння технічних характеристик.
- Оновлення цін у реальному часі.
- Наявність відгуків і рейтингів для оцінки товарів.

**Недоліки**

- Відсутність автоматичної перевірки сумісності компонентів.
- Іноді застаріла інформація про наявність товарів у магазинах.
- Обмежена гнучкість фільтрації для специфічних технічних параметрів (наприклад, підтримка PCIe 4.0).
- Немає можливості створювати чи зберігати повноцінні конфігурації.

**Доцільність для українського користувача**

e-Katalog є ефективним інструментом для пошуку комплектуючих і порівняння цін на українському ринку. Він допомагає знайти вигідні пропозиції та порівняти характеристики, але потребує від користувача самостійного аналізу сумісності. Платформа підходить для тих, хто шукає конкретні компоненти за вигідною ціною.

Кожна з розглянутих платформ має свої сильні та слабкі сторони, які впливають на їхню доцільність для українських користувачів:

- **PCPartPicker** є лідером за функціональністю, пропонуючи глибоку перевірку сумісності, збереження збірок і широку базу даних. Однак через орієнтацію на міжнародний ринок (ціни в доларах/євро, відсутність локальних магазинів) він менш практичний для України без додаткових зусиль із боку користувача.
- **Telemart.ua** пропонує базовий підбір сумісних компонентів і швидке оформлення покупки, що робить її зручною для новачків. Проте відсутність глибокої перевірки сумісності та обмежений функціонал роблять її менш придатною для складних збірок.
- **Hotline.ua і e-Katalog** є ефективними інструментами для порівняння цін і пошуку комплектуючих на українському ринку. Вони пропонують актуальні ціни в гривнях і великий вибір магазинів, але не мають функцій конфігуратора, що вимагає від користувача самостійного аналізу сумісності.

Для українських користувачів ідеальним рішенням могла б стати платформа, яка поєднує переваги всіх систем: автоматичну перевірку сумісності (як у **PCPartPicker**), локалізовані ціни та магазини (як у **Telemart.ua**, **Hotline.ua**, **e-Katalog**), а також можливість зберігати та порівнювати збірки. Наразі користувачам доводиться комбінувати ці інструменти, наприклад, використовувати **PCPartPicker** для створення конфігурації, а **Hotline.ua** чи **e-Katalog** – для пошуку цін і покупки в Україні.

Це створює підґрунтя для розробки власного програмного забезпечення, яке:

- матиме інтуїтивно зрозумілий інтерфейс,
- включатиме вбудовану базу даних компонентів,
- автоматично перевірятиме їх сумісність,
- буде локалізованим і не залежатиме від сторонніх веб-сервісів

## 1.2. Вибір мови програмування та технологій розробки

Після аналізу існуючих систем підбору комп'ютерних компонентів було визначено, що жодна з них не задовольняє повною мірою потреби українських користувачів щодо зручного, локалізованого, технічно грамотного та автономного інструменту підбору ПК-компонентів. Тому для реалізації власного програмного забезпечення необхідно обрати такі технології, які забезпечать швидку розробку, гнучкість, розширюваність та зручність підтримки проєкту.

У процесі розробки програмного забезпечення для вибору комп'ютерних компонентів було прийнято рішення використовувати мову програмування C# у середовищі Windows Forms, а також систему керування базами даних PostgreSQL із драйвером Npgsql. Такий вибір базується на вимогах до функціональності, продуктивності та зручності реалізації інтерфейсу користувача.

- **Вибір мови програмування**

**Мова програмування:** C#
**Середовище:** .NET Framework

Мова програмування C# є однією з провідних у екосистемі .NET, створеній компанією Microsoft. Вона була обрана для розробки цього застосунку завдяки своїм численним перевагам, які роблять її ідеальним вибором для створення настільних застосунків під операційну систему Windows. C# поєднує в собі об'єктно-орієнтований підхід із сучасними можливостями, такими як підтримка асинхронного програмування, делегатів, лямбда-виразів і LINQ (Language Integrated Query), що значно спрощує роботу з даними та логікою застосунку.

**Переваги C#:**

- **Багата бібліотека стандартних класів:** .NET Framework надає широкий набір вбудованих класів і методів для роботи з файлами, мережею, потоками, графічними інтерфейсами, базами даних тощо. Наприклад, класи System.Data і System.Windows.Forms дозволяють легко обробляти дані та створювати інтерфейси.
- **Активна підтримка Microsoft:** C# постійно оновлюється, і Microsoft надає численні інструменти для розробки, такі як Visual Studio, яка є основним середовищем для роботи з цією мовою. Visual Studio пропонує інтуїтивний інтерфейс із автодоповненням коду, дебагером, профайлером і вбудованими шаблонами проєктів.

- **Стабільність і продуктивність:** C# компілюється в проміжний код (IL), який виконується на CLR (Common Language Runtime), що забезпечує стабільність, безпеку типів і ефективне управління пам’яттю через механізм збирання сміття (Garbage Collection).
- **Зручна обробка подій і винятків:** C# має розвинений механізм обробки подій (через делегати та події), що ідеально підходить для створення інтерактивних графічних застосунків. Наприклад, обробка натискання кнопки чи вибір елемента зі списку реалізовується через просту модель подій. Також C# підтримує структуровану обробку винятків (try-catch), що дозволяє ефективно управляти помилками, наприклад, при з’єднанні з базою даних.
- **Робота з графічними інтерфейсами та даними:** C# забезпечує зручні інструменти для створення графічних інтерфейсів (наприклад, через Windows Forms або WPF) і роботи з базами даних через Npgsql.

**Середовище розробки:**

Розробка на C# відбувається у Visual Studio, яка є потужним інтегрованим середовищем розробки (IDE). Visual Studio має зручний інтерфейс із панеллю інструментів, вікном для редагування коду, дизайнером форм (для Windows Forms), а також вікнами для перегляду структури проєкту, властивостей об’єктів і виводу помилок. Дизайнер форм дозволяє "перетягувати" елементи управління (кнопки, текстові поля, списки) на форму, автоматично генеруючи відповідний код.

**Чому саме C#?**

C# обрано через його універсальність, підтримку Microsoft, велику спільноту розробників і зручність для створення настільних застосунків, орієнтованих на Windows. Альтернативи, такі як Java чи Python, могли б бути розглянуті, але Java менш інтегрована з Windows, а Python, хоча й універсальний, поступається C# у продуктивності та зручності роботи з графічними інтерфейсами на Windows.

- **Вибір платформи для створення інтерфейсу**

**Платформа:** Windows Forms
**Середовище:** Visual Studio (дизайнер форм)

Для створення графічного інтерфейсу застосунку було обрано технологію Windows Forms – фреймворк, що входить до складу .NET Framework і призначений для створення настільних застосунків із графічним інтерфейсом користувача (GUI). Windows Forms дозволяє розробникам створювати вікна, кнопки, таблиці, списки, текстові поля та інші елементи управління у візуальному режимі, що значно прискорює процес розробки.

**Переваги Windows Forms:**

- **Візуальний дизайн:** Windows Forms підтримує drag-and-drop інтерфейс у Visual Studio, що дозволяє розробникам швидко створювати форми, розміщуючи елементи управління (наприклад, кнопки, списки, таблиці) безпосередньо на формі. Кожному елементу можна задавати властивості (розмір, колір, текст) через вікно Properties у Visual Studio.
- **Простота моделі подій:** Windows Forms використовує просту і зрозумілу модель обробки подій. Наприклад, для кнопки можна призначити подію Click, і Visual Studio автоматично згенерує відповідний обробник події в коді, куди розробник може додати свою логіку (наприклад, завантаження даних у таблицю після натискання).
- **Оптимальність для проєктів середньої складності:** Windows Forms ідеально підходить для застосунків середньої складності, таких як цей проєкт – програма для підбору комп'ютерних компонентів. Вона не вимагає вивчення додаткових мов розмітки (наприклад, XAML у WPF) і має низький поріг входження для розробників.
- **Сумісність із традиційним дизайном Windows:** Інтерфейси, створені за допомогою Windows Forms, мають класичний вигляд, який відповідає стилю Windows (наприклад, Windows 10 або 11). Це забезпечує знайомий користувачу досвід із типовими елементами, такими як кнопки, меню, діалогові вікна тощо.
- **Швидка розробка:** Завдяки готовим елементам управління (DataGridView для таблиць, ListBox для списків, Button для кнопок) і простій інтеграції з C#, Windows Forms дозволяє швидко створювати функціональні інтерфейси без необхідності писати складний код для рендерингу.

**Інтерфейс застосунку:**
Інтерфейс застосунку, створений у Windows Forms, складається з кількох вкладок (TabControl), кожна з яких відповідає за певний функціонал:

- **Вкладка "Підбір комплектуючих":** Містить таблицю (DataGridView) для відображення доступних компонентів, список (ListBox) для обраних компонентів, кнопки для додавання/заміни компонентів і скидання збірки, а також текстові мітки (Label) для відображення загальної вартості та попереджень.
- **Вкладка "Рекомендовані збірки":** Включає список рекомендованих збірок (ListBox), деталі обраної збірки (ще один ListBox) і кнопки для завантаження або перегляду деталей.
- **Вкладка "Збережені збірки":** Аналогічна за структурою до попередньої, але з додатковими кнопками для видалення та оновлення списку збірок. Кольори кнопок (наприклад, зелений для "Додати", червоний для "Скинути") і шрифти (Arial, розмір 10-12) відповідають стандартам Windows, забезпечуючи зручність і читабельність.

**Чому саме Windows Forms?**

Альтернативи, такі як WPF (Windows Presentation Foundation) або .NET MAUI, пропонують сучасніший дизайн і підтримку кросплатформності, але мають вищий поріг входження. WPF вимагає знання XAML для створення інтерфейсів, а .NET MAUI більше підходить для кросплатформних застосунків, що не є пріоритетом для цього проєкту. Windows Forms була обрана за її простоту, швидкість розробки та відповідність вимогам проєкту.

- **Вибір бази даних**

**СУБД:** PostgreSQL

Для зберігання інформації про комп’ютерні компоненти (категорії, технічні характеристики, ціни, залежності сумісності) було обрано реляційну систему управління базами даних (СУБД) PostgreSQL. Це потужна система з відкритим вихідним кодом, яка широко використовується в проєктах різного масштабу завдяки своїй надійності, гнучкості та підтримці стандартів SQL.

**Переваги PostgreSQL:**

- **Підтримка складних зв’язків:** PostgreSQL дозволяє створювати таблиці з відношеннями "один до багатьох" і "багато до багатьох", що ідеально підходить для цього проєкту. Наприклад, таблиця **processors** (процесори) може бути пов’язана з таблицею motherboards (материнські плати) через поле **socket**, забезпечуючи перевірку сумісності.

- **Гнучкий синтаксис SQL:** PostgreSQL підтримує розширені запити, включаючи агрегатні функції, підзапити, транзакції та тригери, що дозволяє ефективно обробляти дані про компоненти. Наприклад, можна легко отримати список усіх материнських плат, сумісних із певним процесором, за допомогою запиту типу **SELECT * FROM motherboards WHERE socket = 'AM4'.**
- **Висока стабільність і масштабованість:** PostgreSQL відома своєю надійністю навіть при роботі з великими обсягами даних. Вона підтримує індекси, транзакції та резервне копіювання, що забезпечує безпеку даних.
- **Відкритий код:** PostgreSQL є безкоштовною, що знижує витрати на розробку, і має велику спільноту, яка надає підтримку та додаткові інструменти.
- **Логічна структура даних:** У проєкті PostgreSQL використовується для створення взаємопов'язаних таблиць:
    - Таблиця **processors** (процесори): містить поля **id**, **name**, **socket**, **price**.
    - Таблиця **motherboards** (материнські плати): містить поля **id**, **name**, **socket**, **memory_type**, **price**.
    - Таблиця **config_components** (компоненти збірок): пов'язує користувацькі збірки з компонентами через зовнішні ключі. Така структура відповідає об'єктно-реляційній моделі, де об'єкти C# (наприклад, клас **Component**) відображаються на записи в таблицях.

**Чому саме PostgreSQL?**

Альтернативи, такі як MySQL або SQLite, також могли б бути розглянуті. MySQL є схожою за функціоналом, але PostgreSQL переважає за підтримкою складних запитів і типів даних (наприклад, JSON). SQLite підходить для легких застосунків, але менш ефективна для складних зв'язків і великих обсягів даних. PostgreSQL обрано за її універсальність, стабільність і відповідність потребам проєкту.

- **Вибір засобу підключення до бази даних**

**Бібліотека:** Npgsql

Для взаємодії між застосунком на C# і базою даних PostgreSQL було обрано бібліотеку Npgsql – офіційний .NET-драйвер для PostgreSQL. Npgsql забезпечує інтеграцію між .NET-застосунками та PostgreSQL, дозволяючи виконувати SQL-запити, обробляти результати та працювати з транзакціями.

**Переваги Npgsql:**

- Простота підключення: Для підключення до PostgreSQL достатньо вказати рядок з'єднання, наприклад:

```
"Host=localhost;Username=postgres;Password=buriak;Database=pc_co
mponent_selector"
```

Це дозволяє швидко налаштувати з'єднання без складної конфігурації.

- Виконання SQL-запитів: Npgsql підтримує виконання як простих запитів (наприклад, **SELECT * FROM processors**), так і параметризованих запитів для безпеки (наприклад, **SELECT * FROM motherboards WHERE socket = @socket**).
- Обробка результатів: Npgsql дозволяє легко конвертувати результати запитів у об'єкти C#. Наприклад, результати запиту можна завантажити в об'єкт **DataTable** для відображення в таблиці **DataGridView**.
- Робота з транзакціями: Npgsql підтримує транзакції, що дозволяє безпечно виконувати кілька операцій (наприклад, збереження конфігурації користувача та пов'язаних компонентів).
- Документація та спільнота: Npgsql має детальну документацію та активну спільноту, що полегшує вирішення проблем і навчання.
- Інтеграція з Windows Forms: Npgsql легко інтегрується з Windows Forms через ADO.NET, дозволяючи завантажувати дані в елементи управління, такі як **DataGridView** або **ListBox**. Наприклад, список компонентів можна відобразити так:

```
using (var conn = new NpgsqlConnection(connectionString))
{
    conn.Open();
    var cmd = new NpgsqlCommand("SELECT * FROM processors",
conn);
    var adapter = new NpgsqlDataAdapter(cmd);
    var table = new DataTable();
    adapter.Fill(table);
    dataGridView.DataSource = table;
}
```

**Чому саме Npgsql?**

Npgsql є стандартним вибором для роботи з PostgreSQL у .NET, оскільки це офіційна бібліотека, яка забезпечує високу продуктивність і безпеку. Альтернативи, такі як Entity Framework (з Npgsql-провайдером), могли б бути використані, але вони додають зайву складність для проєкту середнього розміру. Npgsql обрано за його простоту, швидкість і пряму роботу з SQL-запитами.

C# і .NET Framework обрано за їх універсальність, підтримку Microsoft і зручність для створення настільних застосунків на Windows, а Visual Studio забезпечує комфортне середовище розробки з інтуїтивним інтерфейсом і потужними інструментами. Windows Forms є оптимальним вибором для швидкої розробки графічного інтерфейсу середньої складності, а інтерфейс застосунку, створений у Windows Forms, має класичний вигляд Windows із вкладками, таблицями, списками та кнопками, що забезпечує зручність для користувача. PostgreSQL обрано за її надійність, підтримку складних зв’язків і гнучкість у роботі з даними, а структура бази даних із взаємопов’язаними таблицями ідеально відповідає потребам проєкту. Npgsql забезпечує просту й ефективну взаємодію між C# і PostgreSQL, дозволяючи легко завантажувати й обробляти дані в інтерфейсі Windows Forms. Цей набір технологій дозволяє створити стабільний, функціональний і зручний застосунок для підбору комп’ютерних компонентів, орієнтований на користувачів Windows.

# 2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1. Вимоги програмного забезпечення

Розроблюване програмне забезпечення призначене для спрощення процесу підбору комп'ютерних комплектуючих, забезпечуючи зручність, ефективність і надійність. Воно має допомагати користувачам, які можуть не мати глибоких технічних знань, створювати сумісні конфігурації ПК, враховуючи ключові технічні характеристики компонентів. На основі аналізу предметної області (складання ПК) та типових сценаріїв використання (вибір компонентів, перевірка сумісності, збереження конфігурацій) було сформульовано детальний перелік функціональних і нефункціональних вимог до системи. Ці вимоги стануть основою для проєктування архітектури, інтерфейсу та бази даних у наступних етапах розробки.

**Функціональні вимоги**

Функціональні вимоги визначають, які саме дії та можливості має надавати система для виконання своїх основних задач. Вони охоплюють ключові аспекти взаємодії користувача з програмою та обробки даних.

- **Завантаження та відображення списку компонентів**

Система повинна забезпечувати зручний доступ до каталогу комплектуючих, дозволяючи користувачу переглядати доступні компоненти.

- **Категоризація компонентів:** Користувач зможе вибирати комплектуючі за категоріями, такими як процесори, материнські плати, відеокарти, оперативна пам'ять, блоки живлення, корпуси, накопичувачі та системи охолодження. Наприклад, вибір категорії "Процесори" у випадаючому списку (ComboBox) відкриває список усіх доступних процесорів.
- **Відображення характеристик:** Для кожного компонента мають бути показані основні технічні характеристики, які допоможуть користувачу зробити усвідомлений вибір. Наприклад, для процесора це можуть бути назва моделі (Intel Core i5-12400), виробник (Intel), сокет (LGA 1700), частота (2.5–4.4 ГГц), кількість ядер (6), ціна (6500 грн). Для відеокарти – назва (NVIDIA GeForce RTX 3060), обсяг пам'яті (12 ГБ), рекомендована потужність блоку живлення (550 Вт). Ці дані відображаються в таблиці (DataGridView), де кожен рядок відповідає одному компоненту, а стовпці – його характеристикам.

- **Фільтрація та сортування компонентів**

Для спрощення пошуку потрібного компонента система має надавати інструменти для фільтрації та сортування даних.

- **Фільтрація за параметрами:** Користувач може задавати критерії для звуження списку компонентів. Наприклад, для оперативної пам'яті можна відфільтрувати модулі за обсягом (16 ГБ), типом (DDR4), частотою (3200 МГц) або ціною (до 3000 грн). Для процесорів – за типом сокету (AM4) або виробником (AMD). Фільтрація може бути реалізована через додаткові елементи управління, такі як текстові поля або випадаючі списки, які дозволяють вводити чи вибирати потрібні значення.
- **Сортування:** Система має підтримувати сортування компонентів за кількома критеріями:
  - **За ціною** (від найдешевшого до найдорожчого або навпаки).
  - **За алфавітом** (за назвою моделі чи виробником).
  - **За продуктивністю** (наприклад, для процесорів – за частотою або кількістю ядер, для відеокарт – за обсягом пам'яті). Сортування реалізується через клік на заголовок стовпця в таблиці або через спеціальні кнопки/меню.

- **Формування конфігурації ПК**

Основна функція програми – створення повноцінної конфігурації ПК із перевіркою сумісності між компонентами.

- **Вибір компонентів:** Користувач має змогу вибрати по одному компоненту з кожної категорії для створення збірки. Наприклад, після вибору процесора (AMD Ryzen 5 5600X) система дозволяє перейти до вибору материнської плати, відеокарти тощо. Обрані компоненти відображаються в окремому списку (ListBox), де користувач бачить поточну конфігурацію.
- **Перевірка сумісності:** Система автоматично перевіряє сумісність між обраними компонентами, враховуючи ключові технічні залежності:
  - Відповідність сокета процесора та материнської плати (наприклад, AM4 для Ryzen 5 5600X).
  - Підтримка типу пам'яті материнською платою (наприклад, DDR4).
  - Достатність потужності блоку живлення для всіх компонентів (наприклад, якщо відеокарта вимагає 650 Вт, а блок живлення забезпечує лише 500 Вт, система видасть попередження).

- Сумісність форм-фактора материнської плати (ATX, Micro-ATX) із корпусом. Якщо виявлено несумісність, система виводить повідомлення (наприклад, через Label із червоним текстом), пояснюючи проблему (наприклад, "Сокет процесора (LGA 1700) не сумісний із материнською платою (AM4)").

- **Збереження та редагування збірки**

Користувач має мати можливість зберігати свої конфігурації для подальшого використання чи редагування.

- **Збереження конфігурації:** Після створення збірки користувач може зберегти її під унікальною назвою (наприклад, "Моя ігрова збірка"). Це реалізується через діалогове вікно, де користувач вводить назву, а система зберігає дані в базу даних (наприклад, у таблиці user_configs).
- **Завантаження та редагування:** Користувач може завантажити раніше збережену збірку, переглянути її складові та внести зміни – замінити компоненти, додати нові або видалити старі. Наприклад, у вкладці "Збережені збірки" відображається список назв збірок, і після вибору однієї з них її компоненти завантажуються в список для редагування.

- **Перегляд підсумкової інформації**

Система має надавати користувачу підсумкову інформацію про створену конфігурацію, щоб він міг оцінити її перед завершенням.

- **Загальна вартість:** Система автоматично підраховує і відображає загальну вартість усіх обраних компонентів у гривнях (наприклад, "Загальна вартість: 35 000 грн"). Це відображається в текстовій мітці (Label) у нижній частині вікна.
- **Короткий звіт щодо сумісності:** Користувач отримує звіт про сумісність компонентів, який включає попередження про можливі проблеми (наприклад, "Недостатня потужність блоку живлення") або підтвердження, що всі компоненти сумісні ("Конфігурація повністю сумісна"). Цей звіт може відображатися в окремій текстовій області або у вигляді спливаючого повідомлення (MessageBox).

**Нефункціональні вимоги**

Нефункціональні вимоги визначають якісні характеристики системи, які впливають на її зручність, продуктивність і масштабованість. Вони не стосуються конкретних функцій, а описують загальні властивості програми.

- **Зручний графічний інтерфейс**

Інтерфейс користувача має бути інтуїтивно зрозумілим, щоб навіть користувачі без технічних знань могли легко ним користуватися.

- **Чітка навігація:** Інтерфейс структурований за допомогою вкладок (TabControl), кожна з яких відповідає за окремий функціонал: вибір компонентів, перегляд рекомендованих збірок, управління збереженими збірками. Наприклад, вкладка "Підбір комплектуючих" містить випадаючий список для вибору категорії, таблицю з компонентами та кнопки для додавання/заміни.
- **Візуальна ясність:** Елементи управління (кнопки, списки, таблиці) мають чіткі підписи українською мовою (наприклад, "Додати до збірки", "Скинути"). Кольори кнопок допомагають швидко зрозуміти їхню функцію: зелений для позитивних дій (додати, зберегти), червоний для негативних (скинути, видалити).
- **Логічне розташування:** Наприклад, таблиця компонентів розташована у верхній частині вікна, список обраних компонентів – унизу, а підсумкова інформація (вартість, попередження) – у нижній частині, щоб користувач бачив її постійно.

- **Модульність**

Код програми має бути структурованим і розбитим на логічні модулі для полегшення підтримки та розширення.

- **Окремі модулі:** Програма поділена на кілька класів:
  - **DatabaseManager** – відповідає за роботу з базою даних (завантаження компонентів, збереження збірок).
  - **PCBuildManager** – обробляє логіку сумісності та управління конфігурацією (перевірка сокетів, підрахунок вартості).
  - **MainForm** – містить код інтерфейсу користувача (Windows Forms).
- **Переваги модульності:** Такий підхід дозволяє легко змінювати один модуль, не зачіпаючи інші. Наприклад, якщо потрібно змінити логіку перевірки сумісності, достатньо відредагувати PCBuildManager, не торкаючись коду інтерфейсу.

- **Масштабованість**

Система має бути спроєктована так, щоб її можна було розширити в майбутньому без значних змін у архітектурі.

- **Розширювана база даних:** Структура бази даних (PostgreSQL) передбачає додавання нових категорій компонентів (наприклад, "Звукові карти") або нових параметрів (наприклад, підтримка Wi-Fi для материнських плат). Для цього достатньо додати нову таблицю або поле, а також оновити код для їх обробки.
- **Гнучка логіка:** Логіка роботи з компонентами (у класі **PCBuildManager**) побудована так, щоб підтримувати нові категорії без зміни основного коду. Наприклад, список категорій зберігається в словнику **ComponentTableMap**, до якого легко додати нову категорію.

- **Автономність**

Програма має працювати автономно, без залежності від підключення до Інтернету.

- **Локальна база даних:** Усі дані про компоненти зберігаються в локальній базі даних PostgreSQL, встановленій на комп'ютері користувача. Це дозволяє програмі працювати офлайн, завантажуючи дані з локального сервера PostgreSQL (наприклад, через **Host=localhost**).
- **Відсутність зовнішніх API:** Система не залежить від зовнішніх вебсервісів чи API, що забезпечує її незалежність від мережі.

- **Продуктивність**

Програма має працювати швидко, забезпечуючи комфортний досвід користувача навіть при великій кількості даних.

- **Швидке відкриття вікон:** Завантаження головної форми та перемикання між вкладками (наприклад, між "Підбір комплектуючих" і "Збережені збірки") має відбуватися без затримок – не більше 1-2 секунд.
- **Ефективна фільтрація:** Фільтрація компонентів (наприклад, за ціною чи сокетом) має виконуватися миттєво, навіть якщо в базі даних є тисячі записів. Це досягається за допомогою індексів у PostgreSQL і оптимізованих SQL-запитів (наприклад, **SELECT * FROM processors WHERE socket = 'AM4'**).
- **Перевірка сумісності:** Перевірка сумісності між компонентами (наприклад, сокета чи потужності блоку живлення) має відбуватися швидко – протягом 0.1-0.5 секунди, щоб користувач не відчував затримок під час додавання компонентів до збірки.

Сформульовані функціональні та нефункціональні вимоги створюють основу для розробки програмного забезпечення, яке відповідає потребам користувачів, які хочуть зібрати комп'ютерну конфігурацію без необхідності глибокого вивчення технічних деталей. Функціональні вимоги забезпечують базові можливості: завантаження та відображення компонентів, їх фільтрацію та сортування, формування конфігурації з перевіркою сумісності, збереження збірок і перегляд підсумкової інформації. Нефункціональні вимоги гарантують зручність (інтуїтивний інтерфейс), модульність (структурований код), масштабованість (можливість розширення), автономність (офлайн-робота) і високу продуктивність (швидка робота). На основі цих вимог у наступних етапах буде спроєктовано архітектуру системи, графічний інтерфейс і структуру бази даних, що забезпечить ефективну реалізацію всіх заявлених функцій.

## 2.2. Проєктування архітектури програми

Розробка будь-якого програмного забезпечення є складним і багатоступеневим процесом, який потребує попереднього визначення загальної архітектури. Архітектура визначає взаємодію основних модулів системи, розподіляє їхні відповідальності, встановлює спосіб обміну даними між компонентами та забезпечує зручний механізм взаємодії з користувачем. Для проєкту програмного забезпечення, призначеного для підбору комп'ютерних компонентів, було вирішено обрати модульну архітектуру, яка базується на чіткому розділенні функціональності на незалежні частини. Такий підхід дозволяє оптимізувати процес розробки, полегшує тестування, підтримку та подальше розширення системи. Архітектура розробляється з урахуванням функціональних і нефункціональних вимог, описаних у попередньому підпункті, таких як зручність інтерфейсу, модульність, масштабованість і продуктивність.

### Загальна структура системи

Система умовно поділяється на три основні рівні, кожен із яких виконує специфічну роль у загальному функціонуванні програми. Ця трирівнева архітектура сприяє чіткому розподілу обов'язків і мінімізує залежності між компонентами.

- **Рівень інтерфейсу користувача (UI Layer)**

Цей рівень відповідає за пряму взаємодію з користувачем через графічні форми, реалізовані за допомогою технології Windows Forms. Він включає всі елементи управління, такі як кнопки (Button), списки (ListBox), таблиці (DataGridView), текстові мітки (Label) і випадаючі списки (ComboBox), які відображають дані та реагують на дії користувача. Наприклад, таблиця DataGridView використовується для показу списку доступних компонентів із їхніми характеристиками, а кнопка "Додати до збірки" запускає подію, яка додає обраний компонент до поточної конфігурації. Інтерфейс розробляється з урахуванням інтуїтивної навігації та візуальної чіткості, щоб забезпечити комфортний досвід навіть для новачків.

- **Логічний рівень (Business Logic Layer)**

Логічний рівень реалізує основну функціональність програми, включаючи обробку даних, логіку перевірки сумісності компонентів, обчислення загальної вартості конфігурації та управління збереженням збірок. Цей рівень виступає посередником між UI та рівнем доступу до даних, забезпечуючи обробку бізнес-логіки без прямого доступу до бази даних. Наприклад, він перевіряє, чи сумісний сокет процесора з материнською платою, і передає результат у вигляді повідомлення для відображення в інтерфейсі. Логічний рівень не виконує SQL-запити, а покладається на рівень доступу до даних для отримання необхідної інформації, що сприяє його незалежності та перевикористанню.

- **Рівень доступу до даних (Data Access Layer)**

Цей рівень відповідає за взаємодію з базою даних PostgreSQL через бібліотеку Npgsql. Він забезпечує відкриття з’єднання із сервером бази даних, виконання SQL-запитів для отримання або збереження даних, обробку результатів і перетворення їх у об’єкти, які використовуються логічним рівнем. Наприклад, запит **SELECT * FROM processors WHERE socket = 'AM4'** повертає список процесорів із відповідним сокетом, а дані формуються у вигляді об’єктів або таблиць для подальшої обробки. Цей рівень ізолює логіку доступу до даних від інших частин системи, що полегшує модифікацію чи заміну бази даних у майбутньому.

**Основні модулі системи**

Архітектура включає три ключові модулі, кожен із яких відповідає за певну групу завдань. Ці модулі реалізуються як окремі класи чи набори класів у коді програми.

- **Модуль роботи з базою даних**

Цей модуль реалізує всі операції, пов’язані з доступом до бази даних PostgreSQL.

- **Відкриття підключення:** Виконується ініціалізація з’єднання з базою даних за допомогою Npgsql із використанням рядка з’єднання (наприклад,"Host=localhost;Username=postgres;Password=buriak;Database=pc_component_selector"). Підключення може бути налаштовано як одноразове для всієї сесії або відкриватися/закриватися для кожного запиту для оптимізації ресурсів.
- **Отримання списку компонентів за категоріями:** Модуль виконує запити для завантаження даних із таблиць, таких як processors, motherboards, rams, і передає їх у вигляді об’єктів або таблиць (DataTable) для відображення в інтерфейсі. Наприклад, вибір категорії "Процесори" викликає запит SELECT * FROM processors.
- **Виконання фільтрації:** Підтримує динамічні SQL-запити для фільтрації компонентів за параметрами, такими як ціна (WHERE price < 5000) чи сокет (WHERE socket = 'LGA 1700'), з використанням параметризованих запитів для безпеки.
- **Збереження/зчитування конфігурацій:** Модуль обробляє операції збереження конфігурацій користувача в таблиці user_configs і пов’язаних таблицях (наприклад, config_components), а також завантаження збережених збірок для редагування.

- **Модуль логіки сумісності**

Цей модуль відповідає за перевірку технічної сумісності компонентів і розрахунок ключових параметрів конфігурації.

- **Перевірка відповідності сокетів процесора і материнської плати:** Порівнює значення поля socket у таблиці processors із відповідним полем у motherboards. Наприклад, якщо процесор має сокет AM4, а материнська плата – LGA 1700, модуль видає помилку сумісності.
- **Сумісність оперативної пам’яті з платою:** Перевіряє тип пам’яті (DDR4, DDR5) і підтримувану частоту (наприклад, до 3200 МГц) між таблицями rams і motherboards.
- **Розрахунок навантаження на блок живлення:** Аналізує сумарну потужність, необхідну для всіх компонентів (з урахуванням рекомендованих значень із gpus і processors), і порівнює її з потужністю блоку живлення (psus). Наприклад, якщо відеокарта вимагає 500 Вт, а блок живлення забезпечує 400 Вт, видається попередження.

- **Визначення підтримуваних інтерфейсів:** Перевіряє сумісність інтерфейсів, таких як PCIe (для відеокарт), SATA (для накопичувачів) і M.2 (для SSD), на основі даних із таблиць motherboards і storages.

- **Модуль управління інтерфейсом**

Цей модуль забезпечує зв'язок між логікою програми та графічним інтерфейсом, а також обробку дій користувача.

- **Відображення списків компонентів:** Завантажує дані з логічного рівня в елементи управління, такі як DataGridView (для доступних компонентів) і ListBox (для обраних компонентів). Наприклад, після фільтрації таблиця оновлюється новими даними.
- **Виведення повідомлень про помилки або несумісність:** Використовує Label або MessageBox для відображення попереджень (наприклад, "Недостатня потужність блоку живлення").
- **Формування підсумкової конфігурації:** Збирає інформацію про обрані компоненти в єдиний список і передає її для відображення (наприклад, у нижній частині форми показується загальна вартість).
- **Збереження обраних даних у файл чи БД:** Ініціює збереження конфігурації через виклик відповідного методу логічного рівня, який у свою чергу звертається до модуля бази даних.

**Принципи побудови архітектури**

Архітектура базується на кількох ключових принципах, які забезпечують її ефективність і гнучкість.

- **Розділення відповідальностей (Separation of Concerns)**

Кожен модуль відповідає лише за свою специфічну область функціональності. Наприклад, модуль інтерфейсу займається лише відображенням і обробкою подій, логічний модуль – перевіркою сумісності, а модуль даних – доступом до бази. Це дозволяє розробникам працювати над окремими частинами системи незалежно, зменшуючи ризик помилок і спрощуючи дебагінг.

- **Слабке зв'язування (Loose Coupling)**

Модулі не залежать один від одного напряму. Наприклад, інтерфейс викликає методи логічного рівня через абстрактні інтерфейси, не знаючи, як саме реалізовано з'єднання з базою даних. Це досягається через використання класів із чітко визначеними методами (наприклад, **PCBuildManager.GetCompatibilityReport()**), що мінімізує залежності й полегшує заміну модулів у майбутньому.

- **Розширюваність (Extensibility)**

Архітектура спроєктована так, щоб у майбутньому до системи можна було додати нові функціональності без значних змін у поточній структурі. Наприклад, додавання нової категорії компонентів (наприклад, "Звукові карти") вимагає лише створення нової таблиці в базі даних, оновлення модуля фільтрації та додавання відповідного елемента в інтерфейс. Логіка сумісності також легко розширюється для нових параметрів (наприклад, перевірка підтримки Thunderbolt).

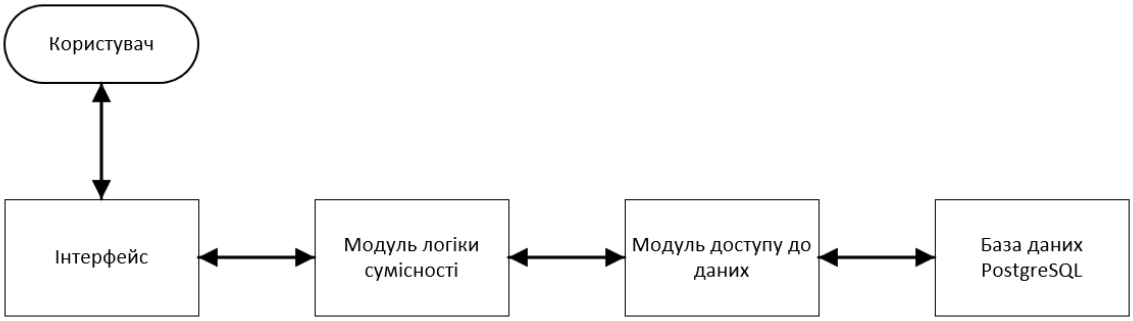


**Рисунок 2.1** – Узагальнена схема взаємодії модулів

Схема відображає взаємодію трьох рівнів і модулів. Наприклад, користувач натискає кнопку "Додати до збірки" в інтерфейсі (**UI Layer**), що викликає метод логічного рівня (**Business Logic Layer**) для перевірки сумісності. Логічний рівень звертається до модуля доступу до даних (**Data Access Layer**), який виконує SQL-запит до PostgreSQL і повертає результат. Потім логіка оновлює стан конфігурації, а інтерфейс відображає оновлений список і повідомлення про сумісність.

Запропонована модульна архітектура з трьома рівнями (UI Layer, Business Logic Layer, Data Access Layer) і трьома основними модулями (робота з базою даних, логіка сумісності, управління інтерфейсом) забезпечує чітку структуру та логічний поділ обов’язків між частинами системи. Принципи розділення відповідальностей, слабкого зв’язування та розширюваності гарантують зручність розробки, тестування та подальшої підтримки застосунку. Такий підхід дозволяє легко адаптувати систему до нових вимог, наприклад, додавання нових категорій компонентів чи вдосконалення логіки сумісності. У наступних підпунктах буде детально розглянуто реалізацію графічного інтерфейсу та структуру бази даних, що базується на цій архітектурі, забезпечуючи практичну основу для подальшої розробки.

## 2.3. Проєктування інтерфейсу користувача

Інтерфейс користувача (UI) є ключовим елементом розроблюваного програмного забезпечення для підбору комп'ютерних компонентів, оскільки він забезпечує пряму взаємодію користувача з усіма функціями системи. У контексті цього проєкту мета інтерфейсу полягає в наданні зручного, інтуїтивно зрозумілого та функціонального середовища для вибору, перегляду, фільтрації, аналізу сумісності та збереження конфігурацій ПК.

Оскільки розробка здійснюється на платформі Windows Forms у межах .NET Framework, інтерфейс реалізується через набір графічних форм із елементами керування, створеними за допомогою дизайнера форм у Visual Studio. Раніше код проєкту було поділено на три основні частини: MainForm.cs (інтерфейс і основна логіка), DatabaseManager.cs (доступ до бази даних) і PCBuildManager.cs (логіка сумісності та управління конфігурацією), що відображає модульну архітектуру, описану раніше. Цей підхід дозволяє чітко розділити відповідальності та полегшує подальшу підтримку та розширення системи.

**Основні вимоги до інтерфейсу**

Інтерфейс розробляється з урахуванням специфічних потреб користувачів, які можуть мати різний рівень технічних знань, і враховує реальні реалізації, присутні в коді проєкту. Вимоги сформульовані на основі функціональності, уже втіленої в **MainForm.cs.**

- **Простота та інтуїтивність:** Інтерфейс має бути простим для освоєння, із чіткими українськими підписами на кнопках і мітках (наприклад, "Додати до збірки", "Загальна вартість"). Навігація здійснюється через вкладки (TabControl), що дозволяє користувачу швидко переключатися між основними функціями без складних меню.
- **Чіткий поділ на категорії компонентів:** Інтерфейс поділено на три вкладки – "Підбір комплектуючих", "Рекомендовані збірки" і "Збережені збірки", кожна з яких відповідає окремому етапу роботи. Наприклад, вкладка "Підбір комплектуючих" фокусується на виборі компонентів, а "Збережені збірки" – на управлінні збереженими конфігураціями.
- **Можливість перегляду технічних характеристик:** Користувачі можуть **переглядати** основні характеристики компонентів у таблиці DataGridView (наприклад, модель, ціна, сокет для процесорів), а детальні дані доступні через окреме вікно "Детальна інформація", яке викликається кнопкою.

- **Підказки або попередження у разі несумісності:** Система відображає попередження про несумісність через мітку powerWarningLabel, наприклад, "Недостатня потужність блоку живлення", якщо обрана конфігурація не відповідає вимогам.
- **Відображення підсумкової вартості та конфігурації:** Загальна вартість відображається в мітці totalPriceLabel (наприклад, "Загальна вартість: 0.00 грн"), яка оновлюється в реальному часі, а список обраних компонентів показується в selectedListBox.

Ці вимоги відображають реальну реалізацію в коді та спрямовані на забезпечення зручності та інформативності для користувача.

**Основні екрани/вікна застосунку**

Інтерфейс реалізовано через кілька ключових екранів, які відповідають функціональним можливостям, описаним у MainForm.cs. Кожен екран спроєктовано для виконання конкретного завдання, із чітким розподілом елементів керування.

**Головне вікно**

Це основне робоче вікно, яке об’єднує всі функції програми й доступне одразу після запуску.

- **Верхнє меню або вкладки для навігації по категоріях:** Навігація реалізується через компонент **TabControl**, який містить три вкладки:
  - "Підбір комплектуючих" – для вибору та додавання компонентів.
  - "Рекомендовані збірки" – для перегляду і завантаження готових конфігурацій.
  - "Збережені збірки" – для управління власними конфігураціями. Кожна вкладка має власний набір елементів, наприклад, **componentGrid** у вкладці "Підбір комплектуючих".
- **Панель пошуку або фільтрації:** У вкладці "Підбір комплектуючих" розміщено **componentTypeCombo** – випадаючий список, який дозволяє вибирати категорії (наприклад, "Процесори", "Материнські плати"). Фільтрація здійснюється автоматично при зміні категорії, але ручної фільтрації за параметрами (ціна, сокет) у поточній версії немає.
- **Таблиця з компонентами (DataGridView):** У вкладці "Підбір комплектуючих" використовується **componentGrid**, яка відображає список компонентів із полями, такими як name, **price** і **socket** (залежно від категорії). Наприклад, для процесорів відображаються модель і ціна, а стовпець **id** прихований через обробник **DataBindingComplete**. Користувач може вибрати рядок подвійним кліком або через кнопку "Додати до збірки".

- **Кнопки:** У вкладці "Підбір комплектуючих" є кнопки **addToBuild** ("Додати до збірки"), **replaceComponentButton** ("Замінити компонент"), **resetButton** ("Скинути збірку") і **saveBuildButton** ("Зберегти збірку"). У вкладках "Рекомендовані збірки" і "Збережені збірки" є кнопки **loadButton** ("Завантажити"), **detailsButton** ("Детальна інформація") і **deleteButton** ("Видалити"), кожна з яких має обробник подій у коді.

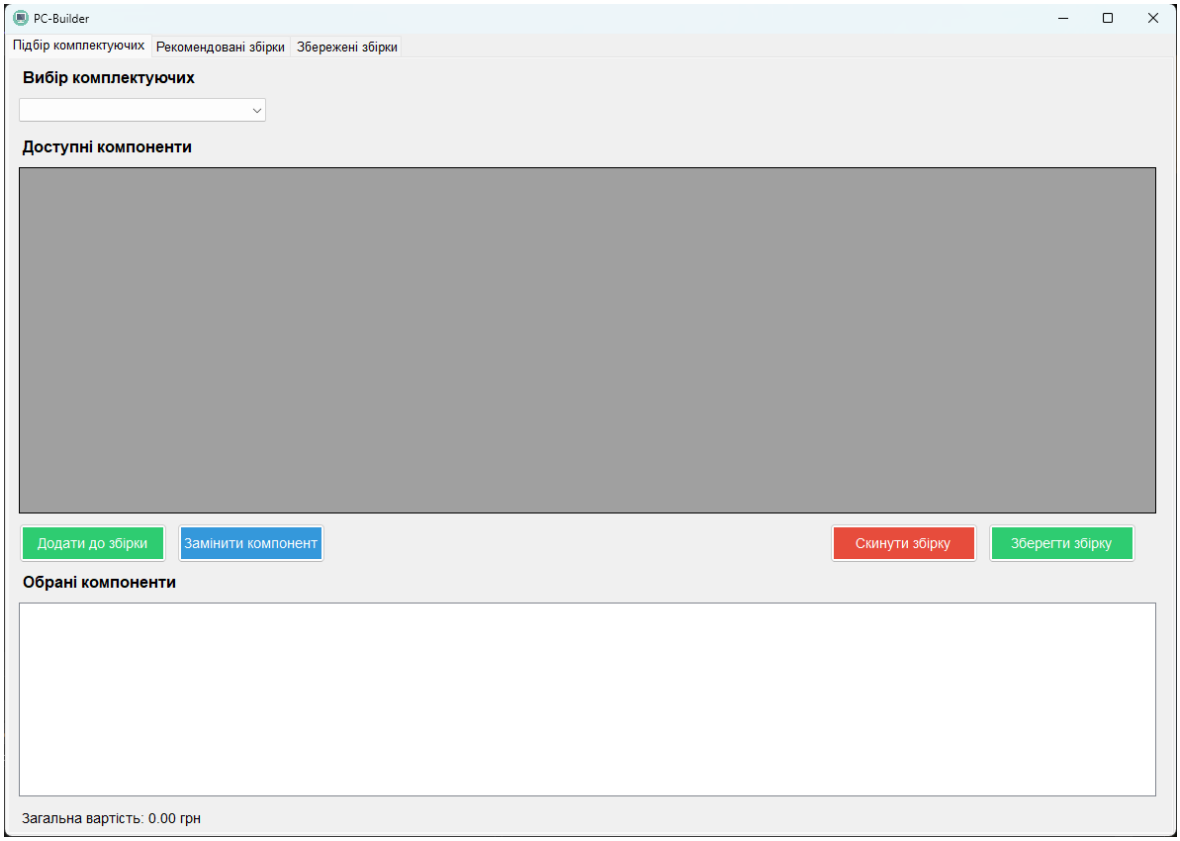


**Рисунок 2.2** – Зображення інтерфейсу головного вікна (Підбір комплектуючих)

**Вікно перегляду конфігурації**

Ця функціональність інтегрована в головне вікно, зокрема в вкладку "Підбір комплектуючих".

- **Виведення всіх обраних компонентів:** Список обраних компонентів відображається в selectedListBox, де кожен рядок показує назву компонента (наприклад, "Процесор: Intel Core i5-12400").
- **Повідомлення про сумісність/несумісність:** Мітка powerWarningLabel відображає попередження, наприклад, "Потужність блоку живлення недостатня", якщо PCBuildManager виявляє проблему.
- **Загальна ціна:** Мітка totalPriceLabel показує поточну вартість (наприклад, "Загальна вартість: 35 000 грн"), яка оновлюється методом AddComponentToSelection у PCBuildManager.

- **Кнопки:** Кнопки "Зберегти збірку" (saveBuildButton) викликає діалогове вікно для введення назви, "Редагувати" реалізується через подвійний клік на selectedListBox для заміни компонентів, а "Очистити" (resetButton) скидає всю конфігурацію.

**Модальне вікно з характеристиками компонента**
Реалізовано через метод **ShowDetailedInfo** у **MainForm.cs**, який відкриває окреме вікно.

- **Детальна інформація:** Вікно відображає деталі компонента, отримані з бази даних через DatabaseManager. Наприклад, для процесора це може бути "Процесор: Intel Core i5-12400 (Socket: LGA 1700, Ціна: 6500 грн)". Дані виводяться в мітках (Label) із автоматичним налаштуванням розміру для довгих текстів.
- **Можливість перегляду зображення (опціонально):** У поточній версії зображення не реалізовано, але код передбачає можливість додавання (наприклад, через PictureBox), якщо будуть доступні файли зображень.

**Вікно збереження/завантаження збірки**
Реалізовано як частина вкладки "Збережені збірки".

- **Оберігає місце збереження:** Зберігання відбувається в базі даних PostgreSQL через **dbManager.SaveBuild**, де користувач вводить назву збірки в діалоговому вікні, викликаному **ShowInputDialog**.
- **Перегляд збережених конфігурацій:** Список збережених збірок відображається в **buildsListBox**, який оновлюється методом **dbManager.LoadBuildsList**.
- **Завантаження для редагування:** Кнопка **loadButton** викликає **dbManager.LoadBuild**, що завантажує обрану збірку в **selectedListBox** для редагування.

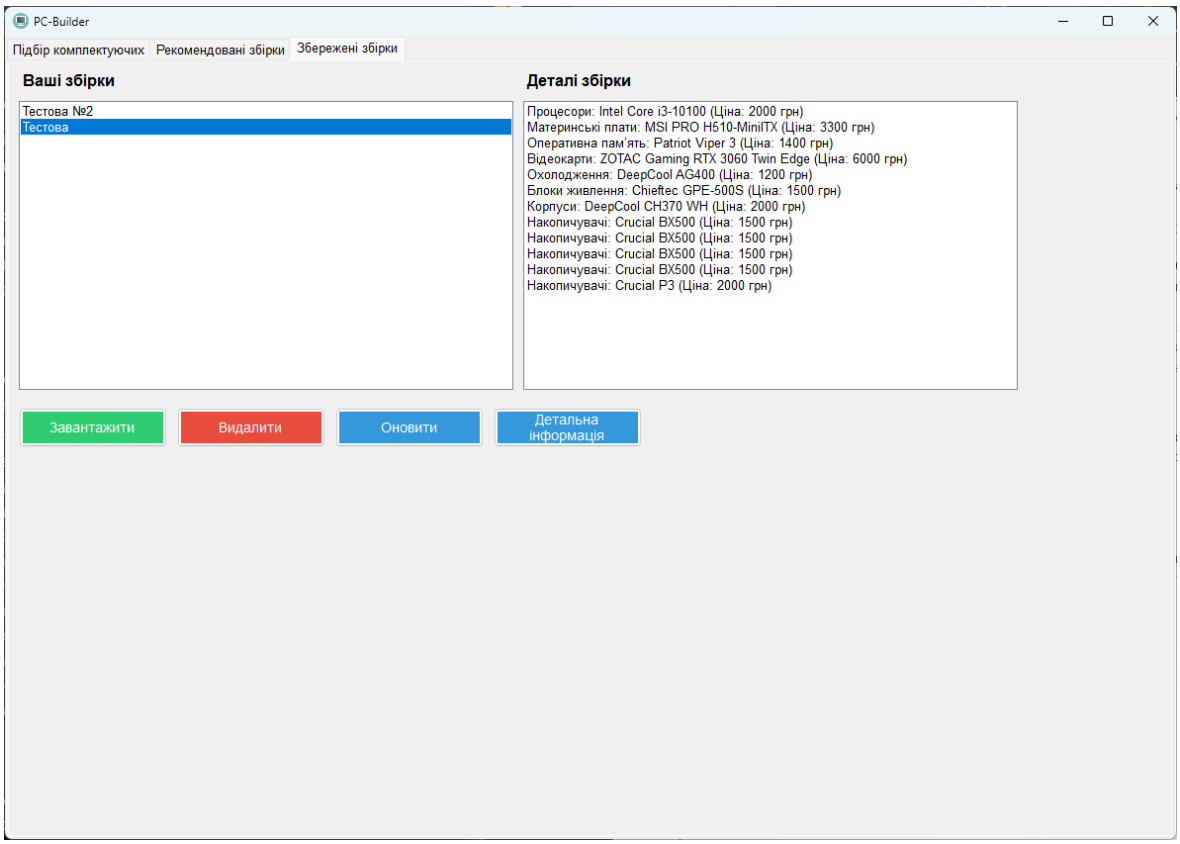


**Рисунок 2.3** - Зображення інтерфейсу вікна (Збережені збірки)

**Основні елементи інтерфейсу**

Інтерфейс побудовано на стандартних елементах Windows Forms, які інтегровані з логікою, реалізованою в MainForm.cs, DatabaseManager.cs і PCBuildManager.cs.

- **DataGridView:** Використовується як **componentGrid** для відображення списку компонентів у вкладці "Підбір комплектуючих". Таблиця має стовпці, такі як **name** і **price**, а обробник **DoubleClick** дозволяє додавати компоненти до збірки.
- **ComboBox / CheckBox: componentTypeCombo** дозволяє вибирати категорії, викликаючи оновлення **componentGrid** через **dbManager.LoadComponentData**. **CheckBox** у поточній версії не використовується, але може бути доданий для фільтрації.
- **Label / TextBox: totalPriceLabel** показує вартість, **powerWarningLabel** – попередження, а **ShowInputDialog** використовує TextBox для введення назви збірки.
- **Button:** Кнопки, такі як **addToBuild** і **saveBuildButton**, мають обробники подій, наприклад, **addToBuild_Click**, який викликає **buildManager.AddComponentToSelection**.
- **TabControl або MenuStrip: TabControl** із трьома вкладками забезпечує навігацію, а MenuStrip не реалізовано, але може бути доданий для розширення функціоналу.

**Реалізація та інтеграція**

Інтерфейс розробляється в Visual Studio за допомогою дизайнера форм. Елементи, такі як **componentGrid** і **selectedListBox**, "перетягуються" на форму, а їхні властивості (розмір, текст) налаштовуються через Properties. Обробники подій, наприклад, **TabControl_SelectedIndexChanged**, інтегрують інтерфейс із логікою **PCBuildManager** і **DatabaseManager**. Наприклад, метод **AddComponentToSelection** оновлює **selectedListBox** і **totalPriceLabel**.

(Приклад коду з **MainForm.cs** для кнопки "Додати до збірки":

```
private void addToBuild_Click(object sender, EventArgs e)
{
    if (componentGrid.SelectedRows.Count > 0)
    {
        string category =
componentTypeCombo.SelectedItem.ToString();
        DataRowView selectedRow =
(DataRowView)componentGrid.SelectedRows[0].DataBoundItem;
        buildManager.AddComponentToSelection(category,
selectedRow.Row, selectedListBox, totalPriceLabel,
powerWarningLabel,
            () => dbManager.LoadComponentData(category ==
"Накопичувачі" ? "storages" : "rams",
buildManager.SelectedComponents, componentGrid));
    }
    else
    {
        MessageBox.Show("Оберіть компонент у таблиці.",
"Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

Цей код ілюструє, як кнопка "Додати до збірки" додає компонент до списку, оновлює ціну й перевіряє сумісність, інтегруючись із **PCBuildManager** і **DatabaseManager**.)

Запроєктований інтерфейс, реалізований у **MainForm.cs**, забезпечує всі необхідні функції для конфігурації ПК – від перегляду компонентів у **componentGrid** до збереження збірок через **saveBuildButton**. Поділ на вкладки ("Підбір комплектуючих", "Рекомендовані збірки", "Збережені збірки") і інтеграція з **PCBuildManager** (перевірка сумісності) та **DatabaseManager** (доступ до даних) відповідають модульній архітектурі. Використання Windows Forms у Visual Studio дозволяє швидко створювати та тестувати дизайн, а реальні приклади коду та скріншоти полегшують розуміння його структури. Цей інтерфейс відповідає вимогам простоти, інформативності та зручності, забезпечуючи ефективну роботу з проєктом.

## 2.4. Проєктування бази даних

База даних є центральною складовою програмного забезпечення, призначеного для підбору комп'ютерних компонентів, оскільки саме вона забезпечує централізоване зберігання, організацію та доступ до великого обсягу інформації про пристрої, необхідні для збирання персонального комп'ютера. Правильне проєктування структури бази даних (БД) відіграє ключову роль у забезпеченні швидкого пошуку потрібних компонентів, ефективної фільтрації за технічними параметрами, автоматизованої перевірки сумісності між різними типами обладнання та надійного збереження користувацьких конфігурацій. У процесі розробки враховувалися вимоги до гнучкості, розширюваності та здатності підтримувати логіку сумісності, що дозволило створити систему, яка може адаптуватися до змін у специфікаціях компонентів і зростаючого обсягу даних. Зокрема, було вирішено відмовитися від використання єдиної універсальної таблиці для всіх типів компонентів на користь спеціалізованої структури з окремими таблицями для кожного типу обладнання, що забезпечує кращу організацію даних і спрощує їх обробку.

Особлива увага приділялася реалізації механізмів перевірки сумісності між компонентами, які є критичними для коректного підбору конфігурації ПК. Наприклад, система повинна перевіряти відповідність сокета процесора та материнської плати (наприклад, AM4 для Ryzen чи LGA 1700 для Intel), сумісність типу та частоти оперативної пам'яті (DDR4 або DDR5), відповідність розмірів відеокарти та кулера корпусу, а також достатність потужності блоку живлення для забезпечення роботи відеокарти та інших компонентів. Ця логіка частково реалізується через SQL-запити та умовні фільтри в коді застосунку (наприклад, у класі **PCBuildManager.cs**), а частково – через тригери в PostgreSQL, які автоматично перевіряють допустимість додавання несумісних записів до конфігурації під час збереження. Такий підхід дозволяє поєднати швидкість виконання запитів із забезпеченням цілісності даних.

Структура бази даних реалізована в системі управління базами даних PostgreSQL, яка була обрана завдяки своїй високій продуктивності, підтримці складних SQL-запитів, тригерів, індексів і розширень, а також відкритості коду, що сприяє гнучкості та безкоштовному використанню. Для взаємодії з базою даних із боку застосунку використовується бібліотека Npgsql – офіційний .NET-драйвер для PostgreSQL, який забезпечує швидкий, безпечний і зручний обмін даними між програмою та сервером бази даних. Npgsql підтримує параметризовані запити, транзакції та обробку результатів у вигляді об'єктів C#, що ідеально вписується в архітектуру застосунку з трьома рівнями (**UI Layer, Business Logic Layer, Data Access Layer**).

У результаті сформована модель бази даних дозволяє ефективно зберігати, обробляти й аналізувати великий обсяг інформації про комп'ютерні компоненти, слугуючи надійною основою для роботи всіх модулів програмного забезпечення, включаючи модуль доступу до даних (**DatabaseManager.cs**) та модуль логіки сумісності (**PCBuildManager.cs**).

Ця підсистема призначена для зберігання інформації про комп'ютерні компоненти, їхні технічні характеристики, а також про користувацькі та рекомендовані збірки ПК. База даних має відповідати наступним ключовим вимогам:

- **Збереження інформації про всі типи компонентів:** Кожен тип обладнання (процесори, материнські плати, оперативна пам'ять тощо) має бути представлений у власній таблиці з унікальними полями для характеристик.
- **Можливість фільтрації за сумісними параметрами:** Система має підтримувати фільтрацію за критеріями, такими як тип сокета, тип пам'яті, форм-фактор материнської плати чи корпусу, що реалізується через SQL-запити з умовами (наприклад, **WHERE socket = 'AM4'**).
- **Підтримка збереження конфігурацій ПК:** Користувацькі збірки повинні зберігатися з можливістю подальшого завантаження та редагування, що досягається через зв'язані таблиці.
- **Масштабованість:** Структура БД спроєктована так, щоб дозволяти додавання нових типів компонентів (наприклад, звукові карти) або додаткових параметрів (наприклад, підтримка Wi-Fi) без значних змін у архітектурі.

**Опис предметної області**

Предметна область охоплює зберігання даних про типові компоненти персонального комп'ютера, включаючи процесори, материнські плати, оперативну пам'ять, відеокарти, накопичувачі, блоки живлення, корпуси та системи охолодження. Для кожного типу компонента зберігаються не лише базові дані, такі як назва, виробник і ціна, але й детальні технічні параметри, які мають вирішальне значення для забезпечення сумісності. Наприклад, для процесора важливими є сокет (AM4, LGA 1700), кількість ядер і тепловиділення (TDP), для материнської плати – підтримувані типи пам'яті (DDR4, DDR5) і кількість слотів M.2, для відеокарти – рекомендована потужність блоку живлення та максимальна довжина. Ці дані використовуються для автоматичної перевірки сумісності під час складання конфігурації.

Крім того, передбачено реалізацію механізму перевірки сумісності між деякими типами компонентів, що є основою функціональності застосунку. Наприклад:

- **Сокет процесора та материнської плати:** Перевірка на відповідність значення поля **socket** у таблиці **processors** із полем **socket** у таблиці **motherboards**.
- **Тип і частота оперативної пам'яті:** Порівняння поля **memory_type** і **frequency** у таблиці **rams** із відповідними полями в **motherboards**.
- **Потужність блоку живлення:** Аналіз сумарного енергоспоживання всіх компонентів (зокрема, поля **recommended_psu_watts** у таблиці **gpus**) і порівняння з полем **wattage** у таблиці **psus**.
- **Розміри компонентів:** Перевірка максимальної довжини відеокарти (**gpu_max_length**) і висоти кулера (**cooler_max_height**) щодо відповідних параметрів корпусу (**case_gpu_max_length**, **case_cooler_max_height**).

Ця логіка частково реалізується через SQL-запити (наприклад, **SELECT * FROM motherboards WHERE socket = (SELECT socket FROM processors WHERE id = 1)**), а частково – через код у **PCBuildManager.cs**, що викликає методи **DatabaseManager.cs** для отримання даних.

**Структура бази даних**

База даних спроєктована як набір взаємопов'язаних таблиць, кожна з яких відповідає певному типу компонента або служить для зберігання конфігурацій. Нижче описано ключові аспекти структури:

- **Таблиця User_Configs:** Ця таблиця зберігає метадані про користувацькі конфігурації, такі як ідентифікатор (**id**, автоінкремент), назва (**name**, наприклад, "Моя робоча збірка") і дата створення (**created_at**). Вона слугує лише для позначення існування конфігурації і не містить деталей про компоненти.
- **Таблиця Config_Components:** Ця таблиця встановлює зв'язок між **User_Configs** і компонентами, включаючи поля **config_id** (зовнішній ключ до **User_Configs**), component_type (наприклад, "**processor**"), component_id (посилання на ідентифікатор компонента в відповідній таблиці) і **quantity** (зазвичай 1, але може бути використано для пам'яті чи накопичувачів).
- **Таблиця Recommended_Configs:** Аналогічна **User_Configs**, але призначена для зберігання рекомендованих збірок (наприклад, "Бюджетна збірка", "Середня збірка"). Має поля **id**, **name, description** і **created_at**.

- **Таблиця Recommended_Config_Components:** Подібна до Config_Components, але пов’язана з Recommended_Configs, із полями config_id, component_type, component_id і quantity.

Крім того, для кожного типу компонента створено окремі таблиці з унікальними полями:

- **processors:** id, name, socket, tdp, cores, base_clock, boost_clock, price.
- **motherboards:** id, name, socket, memory_type, max_ram_slots, form_factor, pcie_slots, sata_ports, m2_slots, power_connectors, price.
- **rams:** id, name, memory_type, size_gb, frequency, module_count, price.
- **gpus:** id, name, interface_type, length_mm, power_draw, psu_recommendation, price, power_connectors.
- **psus:** id, name, power_wattage, power_connector, efficiency, price.
- **cases:** id, name, supported_form_factors, gpu_max_length, cooler_max_height, psu_form_factor, price.
- **coolers:** id, name, socket_support, max_tdp, height_mm, noise_level_db, price.
- **storages:** id, name, interface_type, capacity_gb, form_factor, nvme_support, price.

Ця структура дозволяє зберігати деталізовані дані про кожен компонент і забезпечувати зв’язок із конфігураціями через ідентифікатори.

**Табл. 2.1** – Структура БД

| Таблиця | Призначення |
|---|---|
| Processors, Motherboards, … | Основні дані по всіх типах компонентів |
| User_Configs | Збережені користувачем конфігурації ПК |
| Config_Components | Конкретні компоненти в кожній конфігурації |
| Recommended_Configs | Готові збірки за бюджетом або призначенням |
| Recommended_Config_Components | Компоненти, що входять у рекомендовану збірку |

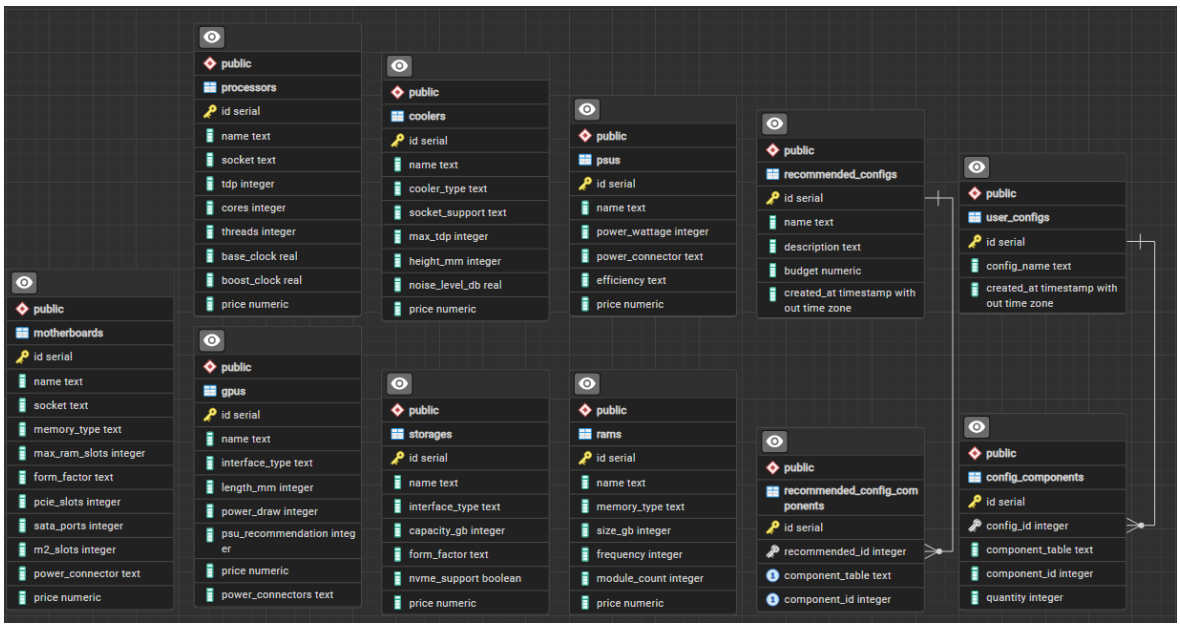


**Рисунок 2.4** – ER-діаграма БД

# 3. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1. Реалізація клієнтської частини

Клієнтська частина програми є основним інтерфейсом взаємодії користувача із системою підбору компонентів для персонального комп'ютера. Вона розроблена з використанням технології Windows Forms у середовищі C#, що забезпечує створення інтуїтивного та функціонального графічного інтерфейсу. Основна мета клієнтської частини — надати користувачу зручний спосіб вибору компонентів, перегляду рекомендованих і збережених збірок, а також керування процесом складання ПК. Інтерфейс включає три основні вкладки, які дозволяють виконувати різні завдання: підбір компонентів, перегляд рекомендованих конфігурацій і роботу із збереженими збірками.

### Структура інтерфейсу

Клієнтська частина реалізована в класі **MainForm** (файл **MainForm.cs**), який створює головне вікно програми розміром 1200x850 пікселів. Вікно ініціалізується з іконкою (**Pc.ico**) та назвою "PC-Builder". Основним елементом інтерфейсу є **TabControl**, який займає всю площу вікна (**Dock = DockStyle.Fill**) і містить три вкладки, створені методами **CreateComponentSelectionTab**, **CreateRecommendedBuildsTab** і **CreateUserBuildsTab**. Кожна вкладка має власний набір елементів управління, таких як мітки (**Label**), списки (**ListBox**), таблиці (**DataGridView**), кнопки (**Button**) і випадаючі списки (**ComboBox**), що забезпечують виконання специфічних функцій.

Цікавим аспектом є модульна організація коду: кожен метод створення вкладки повертає об'єкт **TabPage**, що дозволяє легко додавати нові вкладки або модифікувати існуючі. Наприклад, у методі **MainForm_Load** ініціалізується **TabControl**, до якого додаються вкладки:

```
tabControl = new TabControl { Dock = DockStyle.Fill, Font = new
Font("Arial", 10) };
tabControl.TabPages.Add(CreateComponentSelectionTab());
tabControl.TabPages.Add(CreateRecommendedBuildsTab());
tabControl.TabPages.Add(CreateUserBuildsTab());
Controls.Add(tabControl);
```

Використання **DockStyle.Fill** забезпечує адаптивність інтерфейсу до розмірів вікна, а єдиний шрифт (**Arial**, 10) створює консистентний вигляд усіх елементів.

**Вкладка «Підбір комплектуючих»**

Вкладка "Підбір комплектуючих", створена методом **CreateComponentSelectionTab**, є основним робочим простором для складання ПК. Вона містить кілька ключових елементів:

- **Випадаючий список категорій (**ComboBox**)**: Дозволяє обрати категорію компонентів (процесори, материнські плати тощо). Список заповнюється ключами зі словника ComponentTableMap класу PCBuildManager:

```
componentTypeCombo.Items.AddRange(buildManager.ComponentTableMap
.Keys.ToArray());
```

При зміні вибору в ComboBox викликається подія SelectedIndexChanged, яка завантажує дані для обраної категорії через метод DatabaseManager.LoadComponentData. Це забезпечує динамічне оновлення таблиці компонентів:

```
componentTypeCombo.SelectedIndexChanged += (s, e) =>
{
    string selected =
componentTypeCombo.SelectedItem.ToString();
    string tableName = buildManager.ComponentTableMap[selected];
    Console.WriteLine($"Завантаження даних для таблиці:
{tableName}");
    dbManager.LoadComponentData(tableName,
buildManager.SelectedComponents, componentGrid);
};
```

**Технічний момент:** використання **Console.WriteLine** для дебагу дозволяє відстежувати, яка таблиця бази даних завантажується, що полегшує діагностику.

- **Таблиця компонентів (DataGridView)**: Відображає доступні компоненти для обраної категорії. Таблиця (**componentGrid**) має розміри 1150x350 пікселів і автоматично підлаштовує ширину стовпців (**AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill**). Стовпець **id** приховується для зручності користувача:

```
componentGrid.DataBindingComplete += (s, e) =>
{
    if (componentGrid.Columns.Contains("id"))
    {
        componentGrid.Columns["id"].Visible = false;
    }
};
```

Подвійне клацання на рядку таблиці додає компонент до збірки через метод **AddComponentToSelection**. Це рішення підвищує зручність, дозволяючи швидко додавати компоненти без натискання окремої кнопки.

- **Список обраних компонентів (**ListBox**)**: Відображає компоненти, додані до поточної збірки. Оновлення списку відбувається в методі AddComponentToSelection, який очищає selectedListBox і заповнює його заново:

```
selectedListBox.Items.Clear();
foreach (var kv in SelectedComponents)
{
    foreach (var row in kv.Value)
    {
        string name = row["name"].ToString();
        selectedListBox.Items.Add($"{kv.Key}: {name}");
    }
}
```

Це забезпечує актуальність відображення, навіть якщо компоненти додаються чи видаляються.

- **Кнопки керування**: Кнопки "Додати до збірки", "Замінити компонент", "Скинути збірку" і "Зберегти збірку" мають кольорове оформлення для кращої візуальної ідентифікації (зелений для додавання/збереження, синій для заміни, червоний для скидання). Наприклад, кнопка "Додати до збірки" викликає той же метод AddComponentToSelection, але з перевіркою вибору рядка:

```
addToBuild.Click += (s, e) =>
{
    if (componentGrid.SelectedRows.Count > 0)
    {
        string category =
componentTypeCombo.SelectedItem.ToString();
        DataRowView selectedRow =
(DataRowView)componentGrid.SelectedRows[0].DataBoundItem;
        buildManager.AddComponentToSelection(category,
selectedRow.Row, selectedListBox, totalPriceLabel,
powerWarningLabel, () => dbManager.LoadComponentData(category ==
"Накопичувачі" ? "storages" : "rams",
buildManager.SelectedComponents, componentGrid));
    }
    else
    {
        MessageBox.Show("Оберіть компонент у таблиці.",
"Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
};
```

Для категорій "Накопичувачі" та "Оперативна пам’ять" викликається делегат **reloadComponentData**, який оновлює таблицю компонентів після додавання, враховуючи обмеження (наприклад, кількість слотів M.2 або RAM).

- **Мітки стану**: Мітка **totalPriceLabel** відображає загальну вартість збірки, а **powerWarningLabel** попереджає про недостатню потужність блока живлення. Оновлення цих міток відбувається в методі **UpdateBuildInfo** класу **PCBuildManager**.

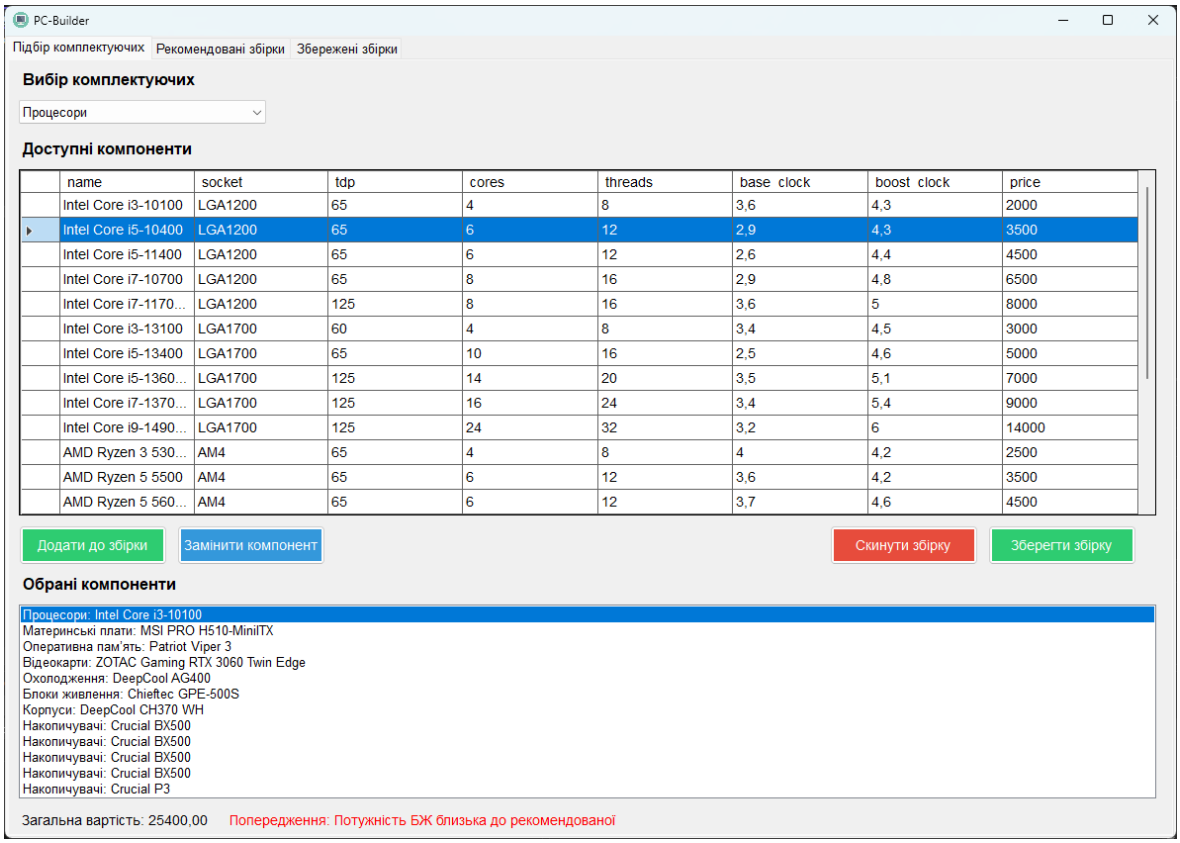


**Рисунок 3.1** – Зображення вкладки «Підбір комплектуючих»

**Вкладка «Рекомендовані збірки»**

Вкладка, створена методом **CreateRecommendedBuildsTab**, дозволяє переглядати три попередньо визначені конфігурації: бюджетну, середню та преміум. Основні елементи:

- **Список збірок (**ListBox**)**: Містить назви збірок, які додаються статично:

```
recommendedBuildsListBox.Items.AddRange(new[] { "Бюджетна
збірка", "Середня збірка", "Преміум збірка" });
```

- **Список деталей (ListBox)**: Відображає компоненти обраної збірки з цінами, оновлюючись при зміні вибору через метод **ShowRecommendedBuildDetails**.
- **Кнопки "Завантажити збірку" та "Детальна інформація"**: Кнопка "Завантажити" викликає **LoadRecommendedBuild**, а "Детальна інформація" відкриває окреме вікно з деталями через метод **ShowDetailedInfo**. Метод **ShowDetailedInfo** підтримує як рекомендовані, так і користувацькі збірки, використовуючи параметр **isRecommended**:

```
detailsButton.Click += (s, e) =>
{
    if (recommendedBuildsListBox.SelectedItem != null)
    {

ShowDetailedInfo(recommendedBuildsListBox.SelectedItem.ToString(
), true);
    }
    else
    {
        MessageBox.Show("Оберіть збірку для перегляду детальної
інформації!", "Попередження", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
    }
};
```

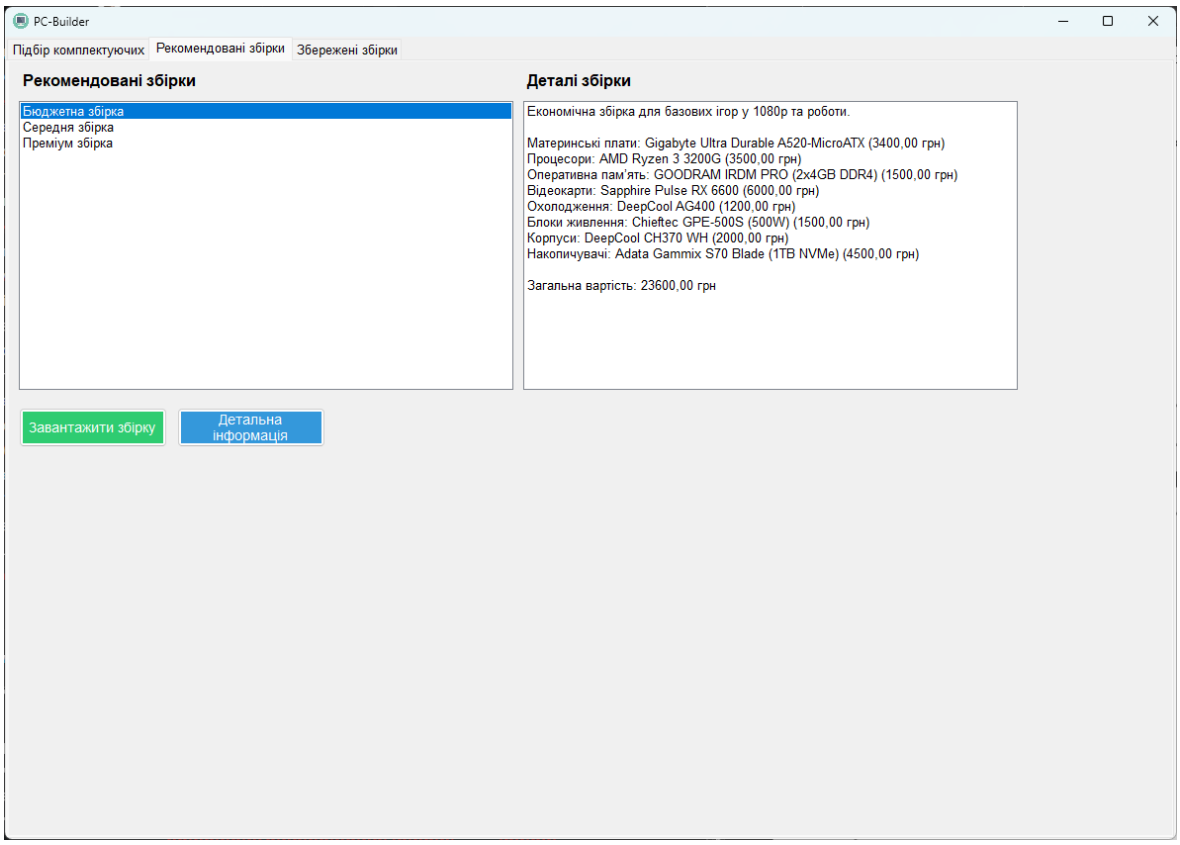


**Рисунок 3.2** – Зображення вкладки «Рекомендовані збірки»

**Вкладка «Збережені збірки»**

Вкладка, створена методом **CreateUserBuildsTab**, дозволяє керувати збереженими користувацькими збірками. Вона включає:

- **Список збірок (**ListBox**)**: Заповнюється через DatabaseManager.LoadBuildsList при завантаженні вкладки.
- **Список деталей (**ListBox**)**: Оновлюється при виборі збірки через ShowBuildDetails.
- **Кнопки керування**: "Завантажити", "Видалити", "Оновити" та "Детальна інформація". Кнопка "Видалити" має підтвердження через MessageBox, що підвищує безпеку:

```
deleteButton.Click += (s, e) =>
{
    if (buildsListBox.SelectedItem != null)
    {
        var result = MessageBox.Show(
            $"Ви впевнені, що хочете видалити збірку
'{buildsListBox.SelectedItem}'?",
            "Підтвердження видалення",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question
        );
        if (result == DialogResult.Yes)
        {

dbManager.DeleteBuild(buildsListBox.SelectedItem.ToString());
            dbManager.LoadBuildsList(buildsListBox);
            buildDetailsListBox.Items.Clear();
        }
    }
    else
    {
        MessageBox.Show("Оберіть збірку для видалення!",
"Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
};
```

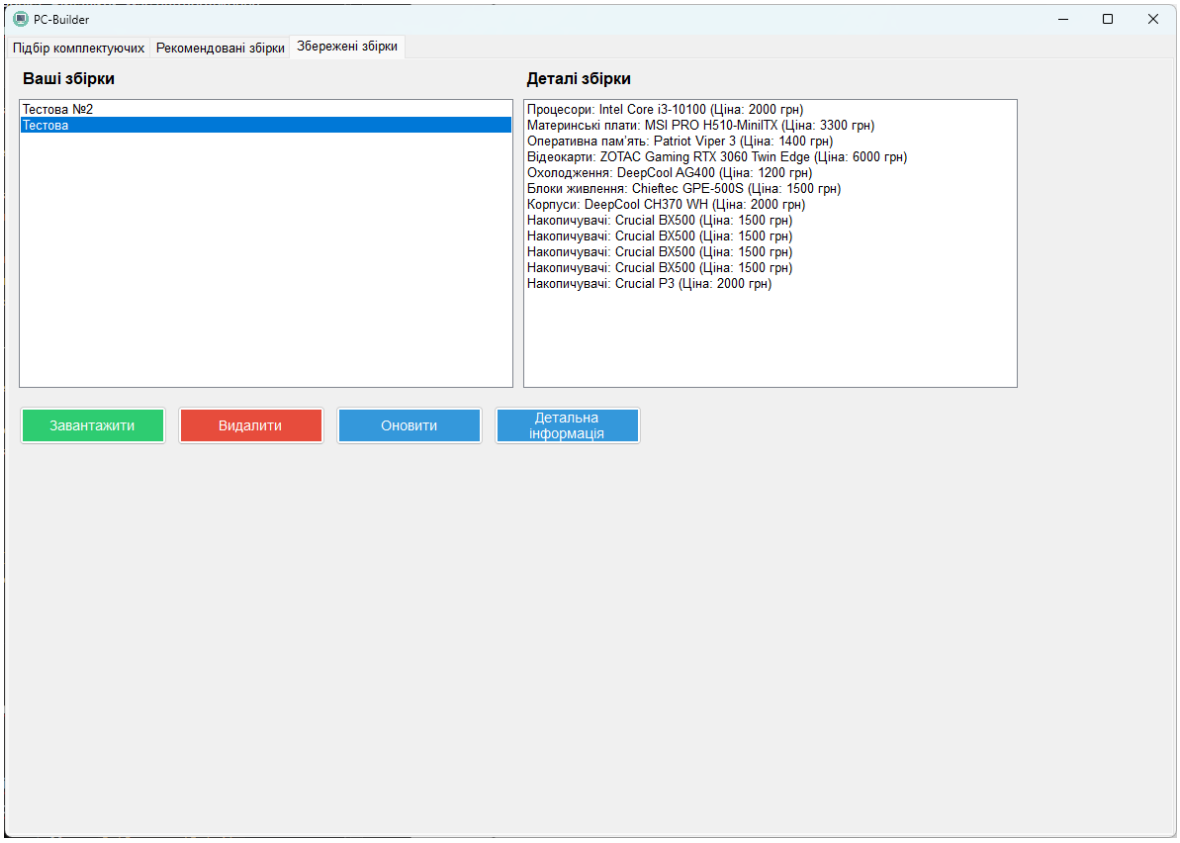


**Рисунок 3.3** – Зображення вкладки «Збережені збірки»

**Технічні рішення вкладки «Збережені збірки»**

- **Динамічне оновлення даних**: Використання делегата в AddComponentToSelection для оновлення таблиці компонентів після додавання накопичувачів або RAM дозволяє враховувати обмеження (наприклад, кількість слотів). Це підвищує точність фільтрації сумісних компонентів.
- **Гнучке відображення деталей**: Метод GetComponentDetails у ShowDetailedInfo адаптує формат відображення залежно від типу компонента (наприклад, для процесорів вказується сокет, для відеокарт — рекомендована потужність БЖ):

```
case "processors":
    details = $"Процесор: {reader["name"]} (Socket:
{reader["socket"]}, Ціна: {reader["price"]} грн)";
    break;
case "gpus":
    details = $"Відеокарта: {reader["name"]} (Рекомендована
потужність БЖ: {reader["psu_recommendation"]} W, Ціна:
{reader["price"]} грн)";
    break;
```

- **Кастомний діалог введення**: Метод ShowInputDialog створює власне модальне вікно для введення назви збірки, із зеленим "OK" і червоним "Скасувати", що відповідає стилю програми.
- **Обробка винятків**: Використання MessageBox для повідомлень про помилки (наприклад, при відсутності обраного компонента) забезпечує зворотний зв’язок із користувачем.

Ці рішення роблять інтерфейс зручним, надійним і адаптивним до потреб користувача.

### 3.2. Реалізація доступу до бази даних через Npgsql

База даних є центральним компонентом програми, що забезпечує зберігання інформації про комп’ютерні компоненти, користувацькі збірки та їх характеристики. Для взаємодії з базою даних використано СУБД PostgreSQL та бібліотеку Npgsql, яка дозволяє інтегрувати C#-програму з PostgreSQL через ADO.NET. Клас **DatabaseManager** (файл **DatabaseManager.cs**) відповідає за всі операції з базою даних, включаючи завантаження даних компонентів, збереження збірок, відображення рекомендованих конфігурацій і керування користувацькими збірками.

**Організація бази даних та підключення**

База даних **pc_component_selector** містить таблиці для кожного типу компонентів: **processors**, **motherboards**, **rams**, **gpus**, **coolers**, **psus**, **cases**, **storages**, а також таблиці **user_configs** і **config_components** для збереження користувацьких збірок. Підключення до бази даних здійснюється через константу **ConnectionString**, визначену в класі **DatabaseManager**:

```
private const string ConnectionString =
"Host=localhost;Username=postgres;Password=buriak;Database=pc_co
mponent_selector";
```

Ця конфігурація вказує на локальний сервер PostgreSQL із заданими обліковими даними. Для кожної операції створюється нове з'єднання через **NpgsqlConnection**, яке відкривається та закривається в блоці **using**, що гарантує коректне звільнення ресурсів. Наприклад, у методі **LoadComponentData**:

```
using (var conn = new NpgsqlConnection(ConnectionString))
{
    conn.Open();
    // Логіка запиту
}
```

Використання **using** запобігає витокам пам'яті та забезпечує надійне закриття з'єднань навіть у разі винятків.

**Ключові методи доступу до бази даних**

- **Завантаження даних компонентів (LoadComponentData)**

Метод **LoadComponentData** є основним для відображення компонентів у таблиці **DataGridView** на вкладці "Підбір комплектуючих". Він приймає назву таблиці (наприклад, **processors**), словник обраних компонентів і таблицю для відображення даних. Метод формує SQL-запит із динамічними умовами для фільтрації сумісних компонентів. Наприклад, якщо обрано материнську плату, для процесорів додається умова відповідності сокету:

```
if (tableName == "processors" &&
selectedComponents.ContainsKey("Материнські плати") &&
selectedComponents["Материнські плати"].Count > 0 &&
selectedComponents["Материнські плати"].Count < 2)
{
    string socket = selectedComponents["Материнські
плати"][0]["socket"].ToString();
    conditions.Add($"socket = @socket");
    parameters.Add(new NpgsqlParameter("socket", socket));
}
```

метод підтримує складні фільтри, такі як перевірка кількості слотів M.2 і SATA для накопичувачів. Якщо слоти вичерпано, додається умова 1=0, що блокує відображення нових накопичувачів:

```
if (selectedM2 >= maxM2Slots && selectedSata >= maxSataPorts)
{
    conditions.Add("1=0");
}
```

Для виконання запиту використовується **NpgsqlDataAdapter**, який заповнює **DataTable**, що прив'язується до **DataGridView**. Обробка винятків із виведенням повідомлення через **MessageBox** підвищує зручність:

```
catch (Exception ex)
{
    MessageBox.Show($"Помилка підключення до БД: {ex.Message}",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

- **Збереження користувацької збірки (SaveBuild)**

Метод **SaveBuild** зберігає поточну збірку в таблицях **user_configs** і **config_components**. Він перевіряє унікальність назви збірки, створює запис у **user_configs** і додає компоненти до **config_components** у межах транзакції:

```
using (var transaction = conn.BeginTransaction())
{
    try
    {
        string insertConfigSql = "INSERT INTO user_configs
(config_name) VALUES (@configName) RETURNING id";
        int configId;
        using (var cmd = new NpgsqlCommand(insertConfigSql,
conn, transaction))
        {
            cmd.Parameters.AddWithValue("configName",
buildName);
            var result = cmd.ExecuteScalar();
            if (result == null)
                throw new Exception("Не вдалося отримати ID
нової збірки.");
            configId = Convert.ToInt32(result);
        }
        // Додавання компонентів
        transaction.Commit();
    }
    catch (Exception ex)
    {
        transaction.Rollback();
```

```
        MessageBox.Show($"Помилка збереження збірки:
{ex.Message}", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}
```

Використання транзакцій забезпечує атомарність операції: якщо хоча б одна дія (наприклад, додавання компонента) завершується невдало, усі зміни скасовуються. Формування підсумкового повідомлення з переліком компонентів і вартістю додає зручності:

```
buildSummary += $"\n\nЗагальна вартість: {totalPrice:F2} грн";
MessageBox.Show($"{buildSummary}\n\nЗбірку збережено успішно!",
"Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

- **Відображення рекомендованих збірок (ShowRecommendedBuildDetails)**

Метод відображає деталі рекомендованих збірок (бюджетна, середня, преміум) у списку **ListBox**. Він отримує компоненти через метод **GetRecommendedBuildComponents** і перевіряє ціни в базі даних, використовуючи резервні ціни зі словника **builds**, якщо дані відсутні:

```
string priceSql = $"SELECT price FROM \"{componentTable}\" WHERE
id = @id";
using (var priceCmd = new NpgsqlCommand(priceSql, conn))
{
    priceCmd.Parameters.AddWithValue("id", componentId);
    var result = priceCmd.ExecuteScalar();
    if (result != null && result != DBNull.Value)
    {
        price = Convert.ToDecimal(result);
    }
    else
    {
        price = component.Item5;
        Console.WriteLine($"Ціна не знайдена в БД для ID
{componentId} у таблиці {componentTable}, використовується
резервна ціна: {price}");
    }
}
```

Метод сортує компоненти, щоб накопичувачі (**storages_**) відображалися після інших категорій, що покращує читабельність:

```
var sortedComponents = buildComponents.OrderBy(c =>
c.Item1.StartsWith("storages_") ? 1 : 0).ToList();
```

- **Завантаження збірок (LoadBuild і LoadRecommendedBuild)**

Методи **LoadBuild** і **LoadRecommendedBuild** завантажують користувацькі та рекомендовані збірки відповідно, оновлюючи словник **SelectedComponents** і список **selectedListBox**. Для користувацьких збірок спочатку отримується **config_id**, а потім компоненти з таблиці **config_components**:

```
string sql = "SELECT component_table, component_id FROM
config_components WHERE config_id = @configId";
```

Для рекомендованих збірок дані беруться зі словника **builds**, але перевіряються в базі даних, що забезпечує актуальність.

**Технічні рішення**

- **Динамічна фільтрація сумісності**

Метод LoadComponentData використовує складну логіку для фільтрації компонентів. Наприклад, для накопичувачів перевіряється кількість доступних слотів M.2 і SATA, що залежить від материнської плати. Це реалізовано через окремі запити для отримання m2_slots і sata_ports:

```
string portSql = "SELECT m2_slots, sata_ports FROM motherboards
WHERE id = @motherboardId";
```

Ця логіка забезпечує точну сумісність, уникаючи вибору несумісних компонентів.

- **Використання параметризованих запитів**

Усі SQL-запити використовують параметри (**NpgsqlParameter**), що захищає від SQL-ін'єкцій і підвищує продуктивність:

```
parameters.Add(new NpgsqlParameter("socket", socket));
```

- **Обробка відсутності даних**

Для рекомендованих збірок передбачено резервні ціни в словнику builds, що дозволяє відображати деталі навіть при відсутності компонента в базі даних. Це додає гнучкості та стійкості до помилок.

- **Дебаг-логування**

Використання Console.WriteLine для виведення інформації про запити та стани (наприклад, selectedM2={selectedM2}) полегшує діагностику під час розробки.

- **Транзакційна цілісність**

Застосування транзакцій у SaveBuild гарантує, що збірка зберігається повністю або не зберігається взагалі, що критично для збереження цілісності даних.

Реалізація доступу до бази даних через Npgsql у класі **DatabaseManager** забезпечує ефективну взаємодію з PostgreSQL, підтримуючи завантаження даних, збереження збірок і перевірку сумісності компонентів. Використання параметризованих запитів, транзакцій і обробки винятків робить систему надійною, а динамічна фільтрація сумісності підвищує її функціональність. Ці рішення дозволяють користувачу легко підбирати компоненти та керувати збірками, зберігаючи дані в структурованій базі.

### 3.3. Розробка базових функцій

Базові функції програми забезпечують її основну функціональність: підбір компонентів для складання персонального комп'ютера з урахуванням їх сумісності, збереження конфігурацій і керування збірками. Ці функції включають завантаження даних компонентів із бази даних, пошук і фільтрацію сумісних компонентів, а також формування та редагування збірок ПК. Реалізація здійснюється через взаємодію класів **DatabaseManager**, **PCBuildManager** і **MainForm**, що забезпечує тісну інтеграцію між інтерфейсом користувача, логікою обробки даних і доступом до бази даних.

**Формування збірки ПК**

Функція формування збірки ПК дозволяє користувачу створювати, редагувати та зберігати конфігурації ПК. Основна логіка реалізована в класі **PCBuildManager** (файл **PCBuildManager.cs**). Ключові методи:

- **Додавання компонента (AddComponentToSelection)**

Метод додає компонент до словника **SelectedComponents**, перевіряючи обмеження. Наприклад, для більшості категорій дозволяється лише один компонент:

```
if (category != "Накопичувачі" && category != "Оперативна
пам'ять")
{
    if (SelectedComponents[category].Count >= 1)
    {
        MessageBox.Show($"Можна додати лише один {category}!",
"Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }
}
```

Для оперативної пам’яті враховується кількість слотів на материнській платі, отримана через метод **GetMaxRamSlots**:

```
int maxRamSlots = new
DatabaseManager().GetMaxRamSlots(motherboardId);
int totalModules = SelectedComponents.ContainsKey("Оперативна
пам’ять") ? SelectedComponents["Оперативна пам’ять"].Sum(row =>
row["module_count"] != DBNull.Value ?
Convert.ToInt32(row["module_count"]) : 1) : 0;
if (totalModules + moduleCount > maxRamSlots)
{
    MessageBox.Show($"Додавання цього набору ({moduleCount}
модулів) перевищить ліміт слотів RAM ({maxRamSlots}).",
"Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    return;
}
```

Після додавання компонента оновлюється список **selectedListBox** і викликається **UpdateBuildInfo** для перерахунку вартості та перевірки сумісності блока живлення з відеокартою.

- **Заміна компонента (ReplaceComponent)**

Метод дозволяє замінити обраний компонент у збірці, перевіряючи відповідність категорії:

```
if (selectedCategory != category &&
!(selectedCategory.StartsWith("Накопичувачі_") && category ==
"Накопичувачі"))
{
    MessageBox.Show("Обраний компонент для заміни належить до
іншої категорії!", "Попередження", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
    return;
}
```

- **Скидання збірки (ResetBuild)**

Очищає словник **SelectedComponents** і список **selectedListBox**, оновлюючи мітки вартості та попереджень.

- **Рекомендовані збірки**

Словник **builds** містить три попередньо визначені конфігурації, які можна завантажити через **GetRecommendedBuildComponents**.

Метод **UpdateBuildInfo** перевіряє сумісність блока живлення з відеокартою, виводячи попередження, якщо різниця потужностей менша за 50 Вт:

```
if (psuPower - gpuPowerRecommendation <= 50)
{
    powerWarningLabel.Text = "Попередження: Потужність БЖ
близька до рекомендованої для GPU!";
}
```

У класі **MainForm** метод **CreateComponentSelectionTab** забезпечує інтеграцію цих функцій із інтерфейсом, дозволяючи додавати компоненти через подвійне клацання на **DataGridView** або кнопку "Додати до збірки".

**Технічні рішення**

- **Динамічна фільтрація в реальному часі**
  Метод **LoadComponentData** застосовує фільтри сумісності на основі обраних компонентів, що оновлюються при кожній зміні категорії чи додаванні компонента. Це забезпечує актуальність відображених даних.
- **Параметризовані SQL-запити**
  Використання NpgsqlParameter у всіх запитах (**LoadComponentData**, **SaveBuild**) захищає від SQL-ін'єкцій і підвищує безпеку.
- **Обмеження кількості компонентів**
  Логіка в A**ddComponentToSelection** обмежує вибір одного компонента для більшості категорій, а для RAM і накопичувачів враховує фізичні обмеження (слоти), що запобігає несумісним конфігураціям.
- **Автоматичне оновлення інтерфейсу**
  Використання делегата в **AddComponentToSelection** для оновлення **DataGridView** після додавання накопичувачів або RAM забезпечує коректне відображення доступних компонентів.
- **Перевірка сумісності живлення**
  Метод **UpdateBuildInfo** автоматично попереджає про недостатню потужність блока живлення, що підвищує надійність збірок.

Розробка базових функцій забезпечує повний цикл роботи з компонентами ПК: від завантаження даних і фільтрації сумісних варіантів до створення та редагування збірок. Інтеграція між класами **DatabaseManager**, **PCBuildManager** і **MainForm** дозволяє створювати зручний і надійний інструмент для користувачів. Технічні рішення, такі як динамічна фільтрація та перевірка сумісності, підвищують точність і ефективність процесу складання ПК.

## 3.4. Тестування

Тестування програми проведено для перевірки коректності роботи основних функцій, сумісності компонентів і стабільності інтерфейсу користувача. Метою тестування було переконатися, що програма дозволяє завантажувати дані компонентів, фільтрувати сумісні варіанти, формувати та зберігати збірки ПК без помилок. Тестові сценарії охоплюють ключові аспекти функціоналу, реалізованого в класах **MainForm**, **PCBuildManager** і **DatabaseManager**, включаючи взаємодію з базою даних PostgreSQL через Npgsql, обробку обмежень і відображення попереджень. У цьому розділі наведено описи тестових сценаріїв, результати їх виконання та технічні рішення, які сприяють надійності програми. Для кожного сценарію передбачено місце для скріншота, що демонструє результат тестування.

**Тестові сценарії та відображення:**

**Тестовий сценарій №1:** Завантаження рекомендованої збірки
**Мета**: Перевірити коректність завантаження рекомендованої збірки та відображення її компонентів у списку обраних компонентів.

**Кроки**:

- **Запустити** програму та перейти на вкладку "Рекомендовані збірки".
- **Обрати** "Середня збірка" у списку **recommendedBuildsListBox**.
- **Натиснути** кнопку "Завантажити збірку".
- **Перевірити** список **selectedListBox** на вкладці "Підбір комплектуючих" і мітку **totalPriceLabel**.

**Очікуваний результат**: У списку **selectedListBox** відображаються компоненти "Середньої збірки", наприклад, "Процесор: AMD Ryzen 5 7600X", "Материнська плата: Gigabyte Eagle X670E-ATX" тощо. Мітка **totalPriceLabel** показує загальну вартість 44 700 грн. Попередження про сумісність (**powerWarningLabel**) відсутнє, оскільки блок живлення (850 Вт) значно перевищує вимоги відеокарти (650 Вт).

**Фактичний результат**: Збірка завантажується коректно, список selectedListBox містить усі компоненти "Середньої збірки", загальна вартість відображається як 44 700 грн, попередження про сумісність не з’являється.

Скріншоти поступового виконання наведені в: **Додаток №1**


**Рисунок 3.4** – Скріншот результату тестування №1

**Тестовий сценарій №2:** Фільтрація сумісних компонентів
**Мета**: Перевірити коректність фільтрації процесорів за сумісністю з обраною материнською платою.

**Кроки**:

- На вкладці "Підбір комплектуючих" обрати категорію "Материнські плати" у **componentTypeCombo**.
- Вибрати материнську плату "ASUS ROG Z790-EATX" (сокет LGA1700) у таблиці **componentGrid** і додати її до збірки кнопкою "Додати до збірки".
- Змінити категорію на "Процесори" у **componentTypeCombo**.
- Перевірити вміст таблиці **componentGrid**.

**Очікуваний результат**: Таблиця componentGrid містить лише процесори з сокетом LGA1700, наприклад, "Intel Core i9-14900K". Процесори з іншими сокетами (AM4, AM5) не відображаються.

**Фактичний результат**: Таблиця відображає лише сумісні процесори з сокетом LGA1700, включаючи "Intel Core i9-14900K". Несумісні процесори, такі як "AMD Ryzen 5 7600X" (AM5), відсутні.

Скріншоти поступового виконання наведені в: **Додаток №2**

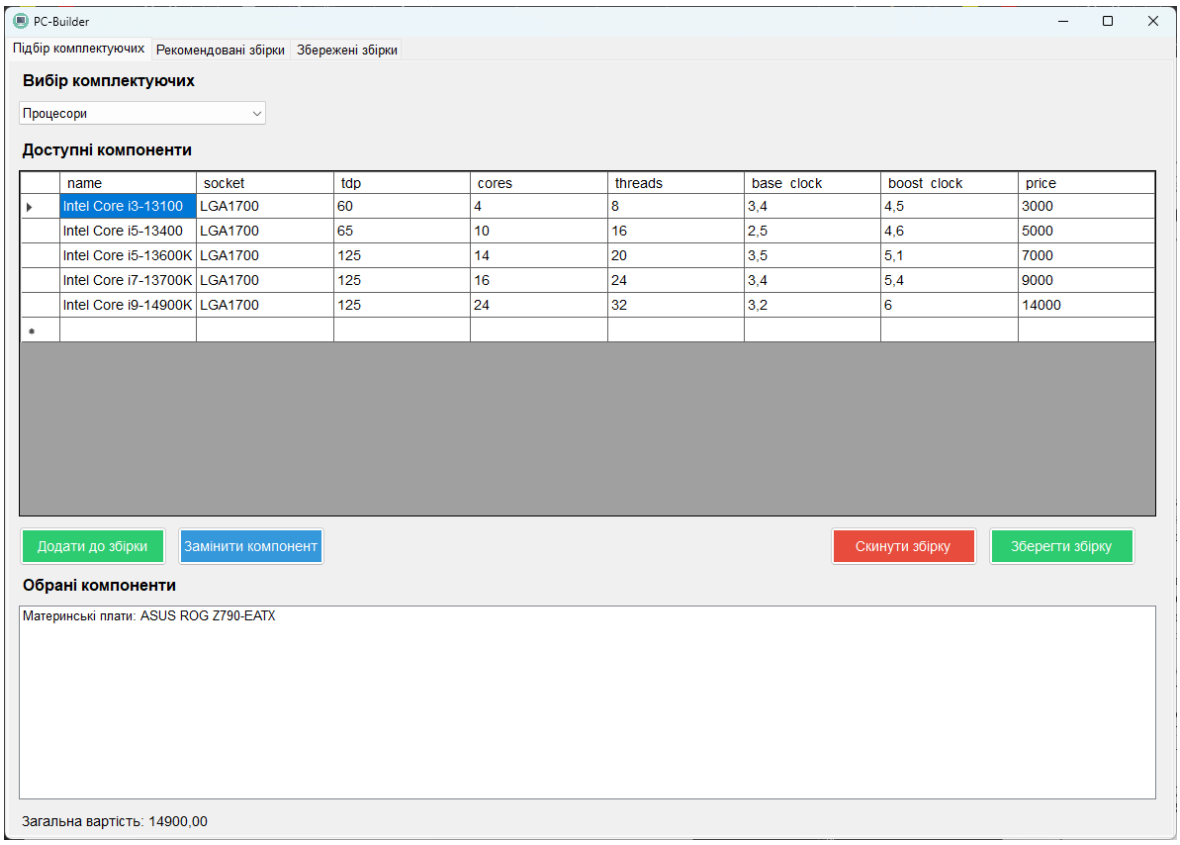


**Рисунок 3.5** – Скріншот результату тестування №2

**Тестовий сценарій 3:** Перевірка попередження про сумісність БЖ
**Мета**: Перевірити відображення попередження, якщо потужність БЖ близька до рекомендованої для відеокарти.

**Кроки**:

- На вкладці "Підбір комплектуючих" обрати категорію "Відеокарти" та додати "ZOTAC Gaming RTX 3060 Twin Edge" (рекомендована потужність 450 Вт).
- Обрати категорію "Блоки живлення" та додати "Chieftec GPE-500S" (потужність 500 Вт).
- Перевірити вміст мітки **powerWarningLabel**.

**Очікуваний результат**: Мітка **powerWarningLabel** відображає текст "Попередження: Потужність БЖ близька до рекомендованої для GPU!", оскільки різниця між потужністю блока живлення (500 Вт) і вимогами відеокарти (450 Вт) становить менше 50 Вт.

**Фактичний результат**: Попередження відображається коректно з указаним текстом.

Скріншоти поступового виконання наведені в: **Додаток №3**

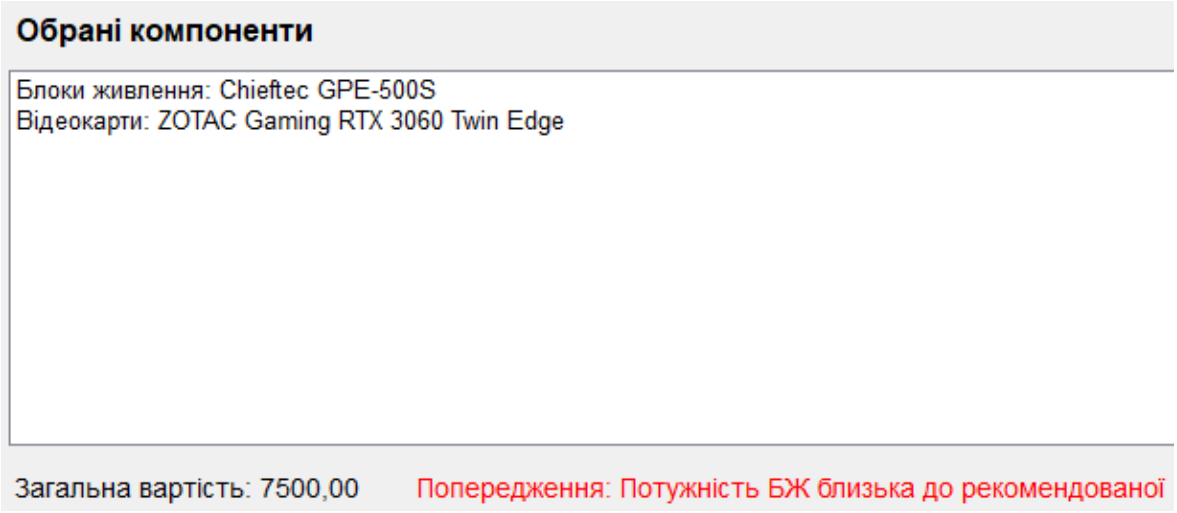


**Рисунок 3.6** – Скріншот результату тестування №3

**Тестовий сценарій 4:** Збереження та завантаження користувацької збірки
**Мета**: Перевірити коректність збереження збірки в базі даних і її завантаження.

**Кроки**:

- На вкладці "Підбір комплектуючих" додати компоненти: процесор "AMD Ryzen 3 5300G", материнську плату "Gigabyte Ultra Durable A520-MicroATX", оперативну пам’ять "GOODRAM IRDM PRO (2x8GB DDR4)".
- Натиснути кнопку "Зберегти збірку", у діалоговому вікні ввести назву "Тестова збірка 1".
- Перейти на вкладку "Збережені збірки" та перевірити наявність "Тестова збірка 1" у списку **buildsListBox**.
- Обрати збірку та натиснути "Завантажити".

**Очікуваний результат**: Після збереження з’являється повідомлення про успішне збереження, "Тестова збірка 1" відображається у списку збережених збірок. Після завантаження компоненти збірки з’являються в **selectedListBox** на вкладці "Підбір комплектуючих", а **totalPriceLabel** показує правильну суму (наприклад, 7400 грн для зазначених компонентів).

**Фактичний результат**: Збірка зберігається, повідомлення про успіх відображається, назва з’являється у списку. Після завантаження усі компоненти коректно відображаються, загальна вартість відповідає очікуваній.

Скріншоти поступового виконання наведені в: **Додаток №4**

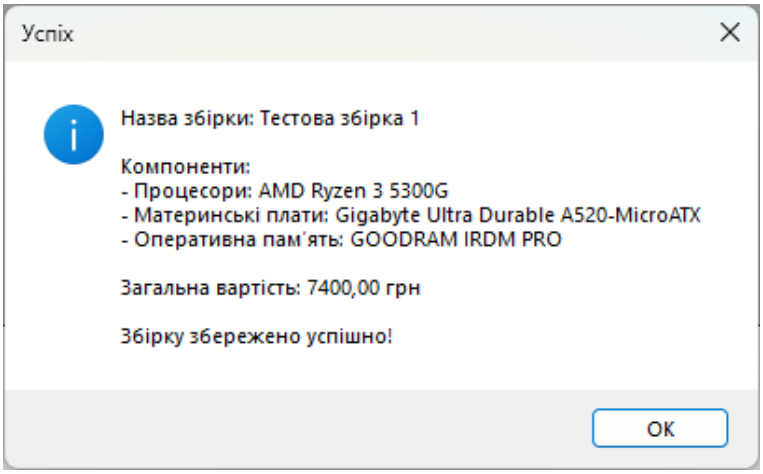


**Рисунок 3.7** – Скріншот результату тестування №4

**Тестовий сценарій 5:** Видалення користувацької збірки
**Мета**: Перевірити коректність видалення збірки з бази даних.

**Кроки**:

- На вкладці "Збережені збірки" обрати "Тестова збірка 1" у списку **buildsListBox**.
- Натиснути кнопку "Видалити".
- У діалоговому вікні підтвердити видалення, натиснувши "Так".
- Перевірити список **buildsListBox**.

**Очікуваний результат**: З'являється діалогове вікно підтвердження, після натискання "Так" відображається повідомлення про успішне видалення, а "Тестова збірка 1" зникає зі списку **buildsListBox**.

**Фактичний результат**: Діалогове вікно з'являється, після підтвердження збірка видаляється, повідомлення про успіх відображається, список оновлюється без "Тестова збірка 1".

Скріншоти поступового виконання наведені в: **Додаток №5**

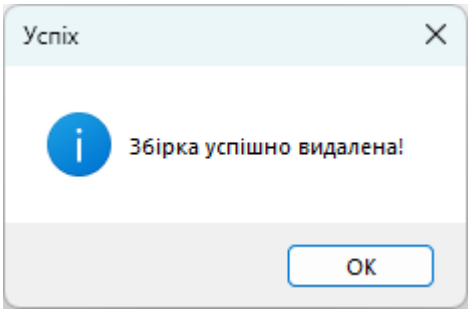


**Рисунок 3.8** – Скріншот результату тестування №5

**Технічні рішення**

- **Автоматизована перевірка сумісності**
  Метод **UpdateBuildInfo** у **PCBuildManager** автоматично перевіряє сумісність блока живлення з відеокартою, виводячи попередження при недостатній потужності, що знижує ризик помилок.

- **Динамічна фільтрація даних**
  Метод **LoadComponentData** у **DatabaseManager** застосовує SQL-фільтри на основі обраних компонентів, забезпечуючи відображення лише сумісних варіантів.

- **Транзакційна цілісність**
  Метод **SaveBuild** використовує транзакції для збереження збірок, гарантуючи, що дані записуються повністю або не записуються взагалі.

- **Обробка винятків**
  Усі методи доступу до бази даних (**LoadComponentData**, **SaveBuild**, **LoadBuild**) включають обробку винятків із виведенням повідомлень через **MessageBox**, що підвищує зручність діагностики.

- **Підтвердження критичних дій**
  Метод видалення збірки в **MainForm** використовує діалогове вікно підтвердження, що запобігає випадковому видаленню даних.

Тестування підтвердило коректність роботи основних функцій програми, включаючи завантаження рекомендованих і користувацьких збірок, фільтрацію сумісних компонентів, перевірку сумісності живлення та керування збереженими конфігураціями. Усі сценарії виконалися успішно, а технічні рішення, такі як автоматизовані перевірки та транзакції, забезпечують надійність і зручність роботи системи. Скріншоти результатів тестування ілюструють успішне виконання кожного сценарію.

# ВИСНОВОК

Розробка програми для підбору компонентів ПК завершилася створенням функціональної системи, яка дозволяє користувачам складати сумісні конфігурації комп’ютерів із зручним графічним інтерфейсом. Використовуючи C# і Windows Forms, реалізовано інтуїтивний інтерфейс із трьома вкладками для вибору компонентів, перегляду рекомендованих і збережених збірок, що включають елементи керування, такі як списки, таблиці та кнопки. Інтеграція з базою даних PostgreSQL через бібліотеку Npgsql забезпечує надійне завантаження даних, фільтрацію сумісних компонентів за характеристиками, наприклад, сокетом процесора чи типом пам’яті, а також збереження конфігурацій із застосуванням транзакцій для цілісності.

Основні функції програми, включаючи динамічну фільтрацію, формування збірок із перевіркою обмежень, таких як кількість слотів RAM чи потужність блока живлення, і підтримку трьох рекомендованих конфігурацій, працюють коректно, що підтверджено п’ятьма тестовими сценаріями. Ці тести показали стабільність системи під час завантаження збірок, відображення попереджень і керування даними. Програма повністю відповідає поставленій меті створення зручного інструменту для підбору компонентів ПК, забезпечуючи сумісність, простоту використання та надійність.

Для подальшого вдосконалення пропонується розширити базу даних новими типами компонентів, додати фільтри за ціною чи продуктивністю, реалізувати хмарну синхронізацію збірок, розробити мобільний додаток і впровадити автоматизовані тести для перевірки граничних випадків. Такі зміни зроблять програму більш універсальною та привабливою для ширшої аудиторії.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft Docs. Windows Forms Documentation [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/ (дата звернення: 7.04.2025).
2. Microsoft Docs. ADO.NET Overview [Електронний ресурс]. URL: https://learn.microsoft.com/dotnet/framework/data/adonet/ (дата звернення: 7.04.2025).
3. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/ (дата звернення: 7.04.2025).
4. Npgsql Documentation. Npgsql – .NET Data Provider for PostgreSQL [Електронний ресурс]. URL: https://www.npgsql.org/doc/ (дата звернення: 7.04.2025).
5. Python Software Foundation. Python Official Documentation [Електронний ресурс]. URL: https://docs.python.org/3/ (дата звернення: 7.04.2025).
6. SQLite Documentation. [Електронний ресурс]. URL: https://www.sqlite.org/ (дата звернення: 7.04.2025).
7. PCPartPicker. System Builder [Електронний ресурс]. URL: https://pcpartpicker.com/ (дата звернення: 9.04.2025).
8. Telemart.ua [Електронний ресурс]. URL: https://telemart.ua/ (дата звернення: 9.04.2025).
9. Hotline.ua [Електронний ресурс]. URL: https://hotline.ua/ (дата звернення: 9.04.2025).
10. e-Katalog [Електронний ресурс]. URL: https://ek.ua/ (дата звернення: 9.04.2025).
11. Intel Corporation. Processor Specifications [Електронний ресурс]. URL: https://www.intel.com/content/www/us/en/ark.html (дата звернення: 10.04.2025).
12. AMD. Ryzen Processors Technical Details [Електронний ресурс]. URL: https://www.amd.com/en/products/processors/desktops/ryzen.html (дата звернення: 10.04.2025).
13. NVIDIA. GeForce RTX Series Specifications [Електронний ресурс]. URL: https://www.nvidia.com/en-us/geforce/graphics-cards/compare/ (дата звернення: 10.04.2025).
14. Stack Overflow [Електронний ресурс]. URL: https://stackoverflow.com/ (дата звернення: 17.04.2025).
15. TechPowerUp. GPU Database [Електронний ресурс]. URL: https://www.techpowerup.com/gpu-specs/ (дата звернення: 17.04.2025).
16. Tom’s Hardware. CPU Benchmarks and Hierarchy [Електронний ресурс]. URL: https://www.tomshardware.com/reviews/cpu-hierarchy,4312.html (дата звернення: 17.04.2025).

17. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE). ISO, 2011. 34 p.
18. IEEE Standard for Software and System Test Documentation. IEEE Std 829-2019. IEEE, 2019. 96 p.
19. IEEE Standard for Software Testing Documentation. IEEE Std 829-2008. IEEE, 2008. 118 p.
20. PassMark Software. CPU Benchmark [Електронний ресурс]. URL: https://www.cpubenchmark.net/ (дата звернення: 24.04.2025).

# ДОДАТКИ

## Додаток №1 – Скріншоти поетапного тестування (№1)

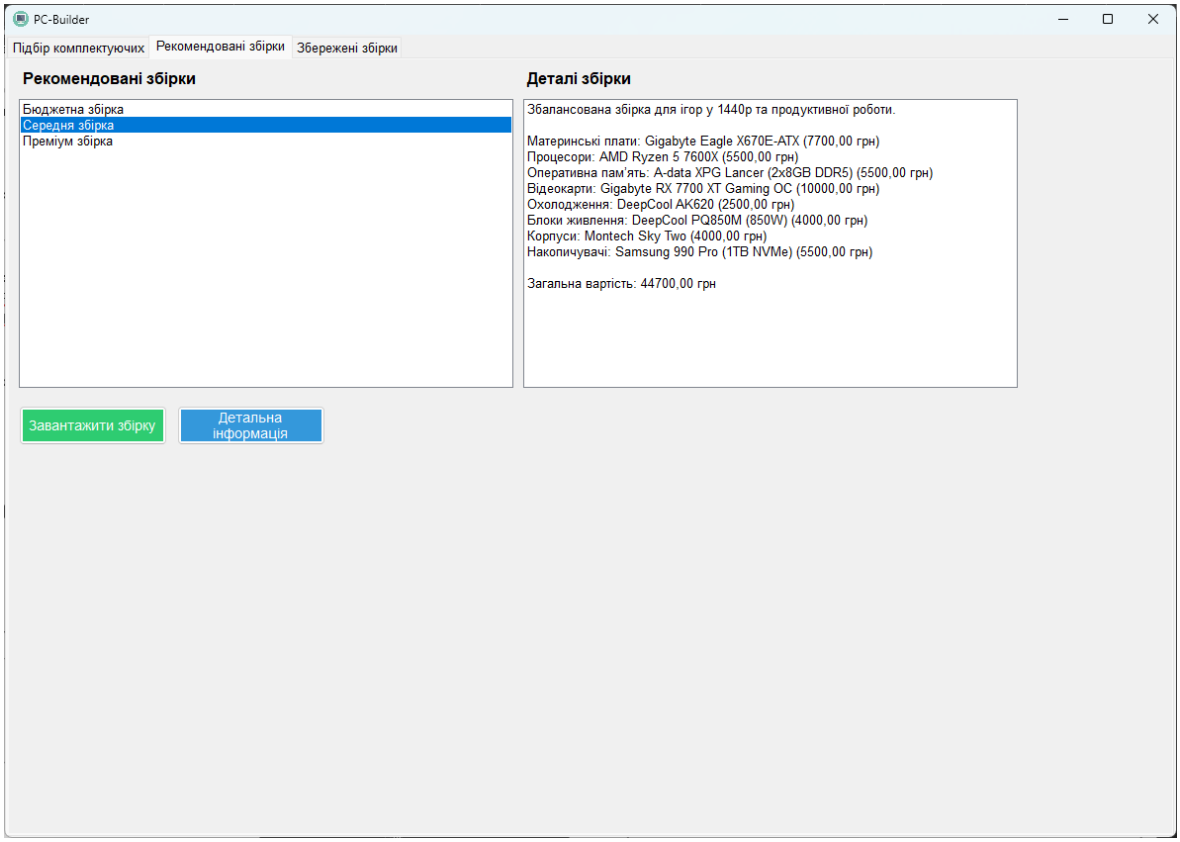


**Рисунок Д.1.1** – Скріншот виконання тестування №1


**Рисунок Д.1.2** – Скріншот виконання тестування №1

## Додаток №2 - Скріншоти поетапного тестування (№2)

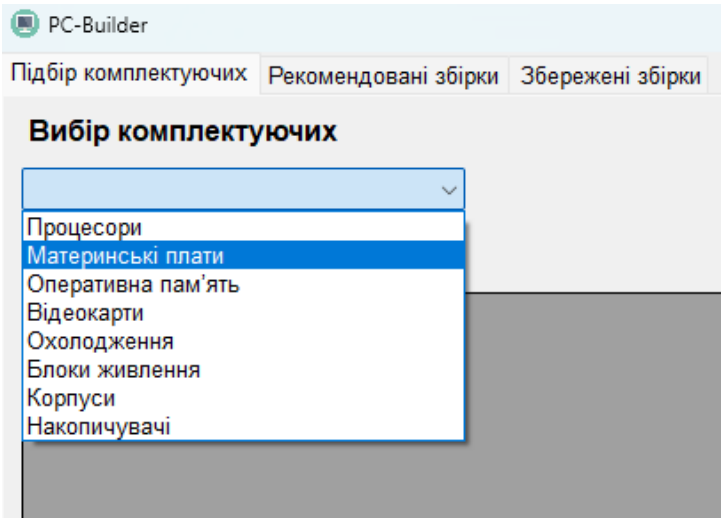


**Рисунок Д.2.1** – Скріншот виконання тестування №2

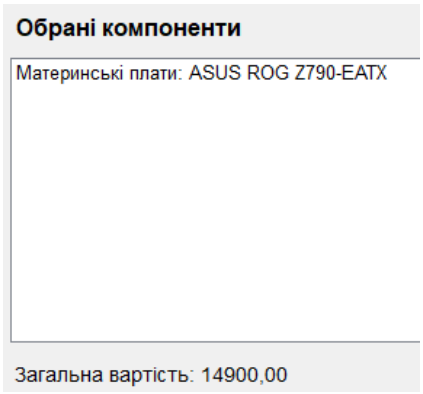


**Рисунок Д.2.2** – Скріншот виконання тестування №2

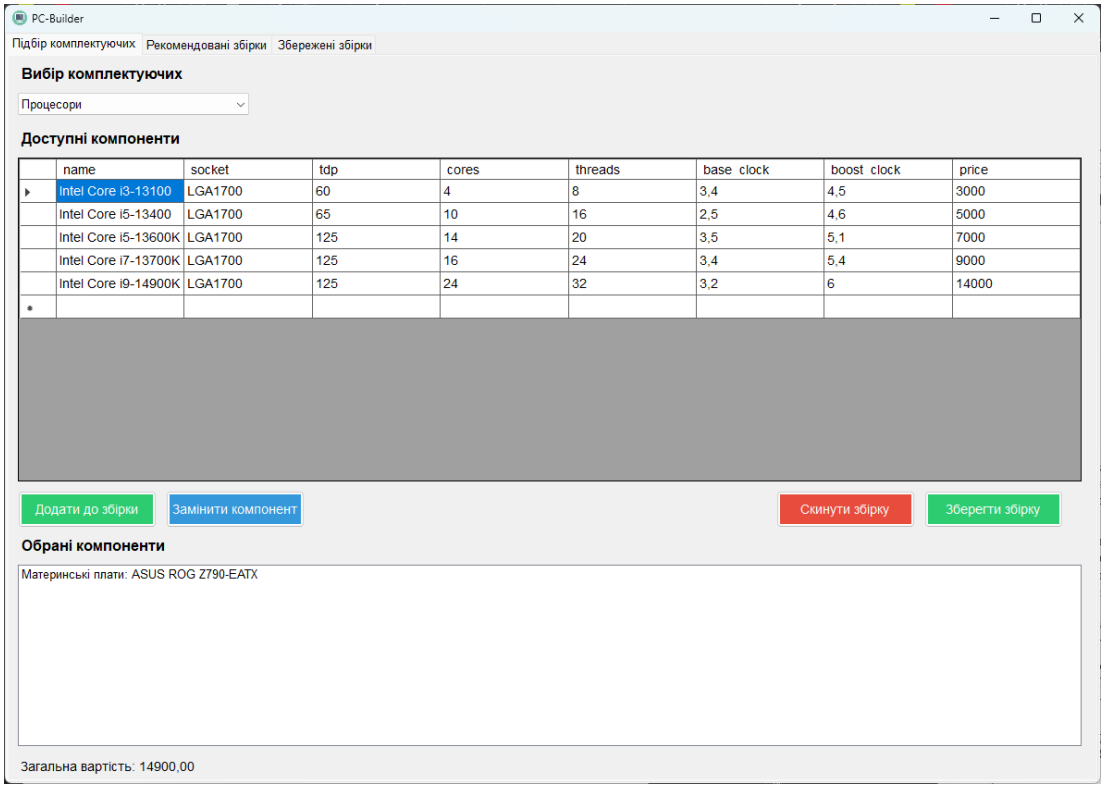


**Рисунок Д.2.3** – Скріншот виконання тестування №2

## Додаток №3 - Скріншоти поетапного тестування (№3)

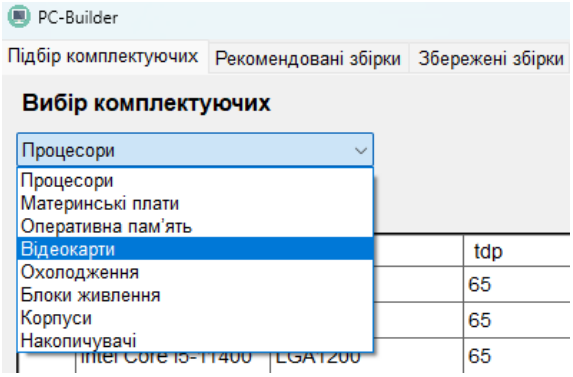


**Рисунок Д.3.1** – Скріншот виконання тестування №3

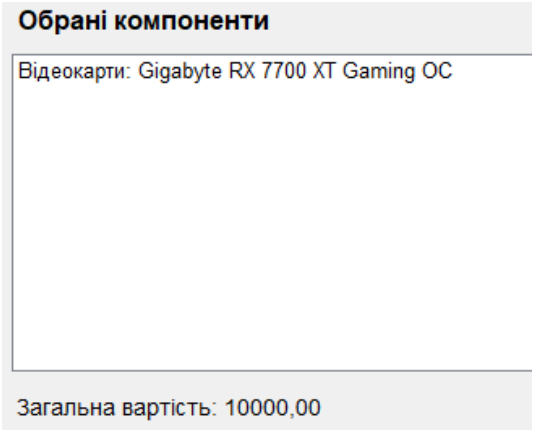


**Рисунок Д.3.2** – Скріншот виконання тестування №3

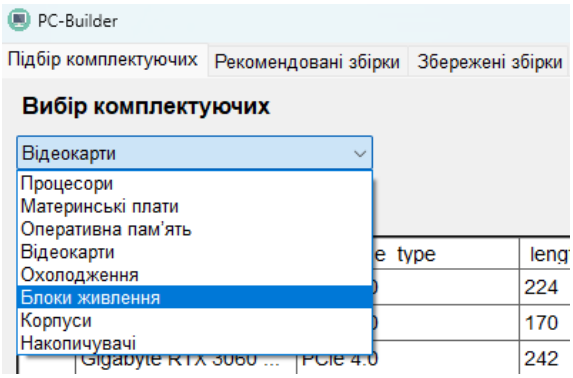


**Рисунок Д.3.3** – Скріншот виконання тестування №3

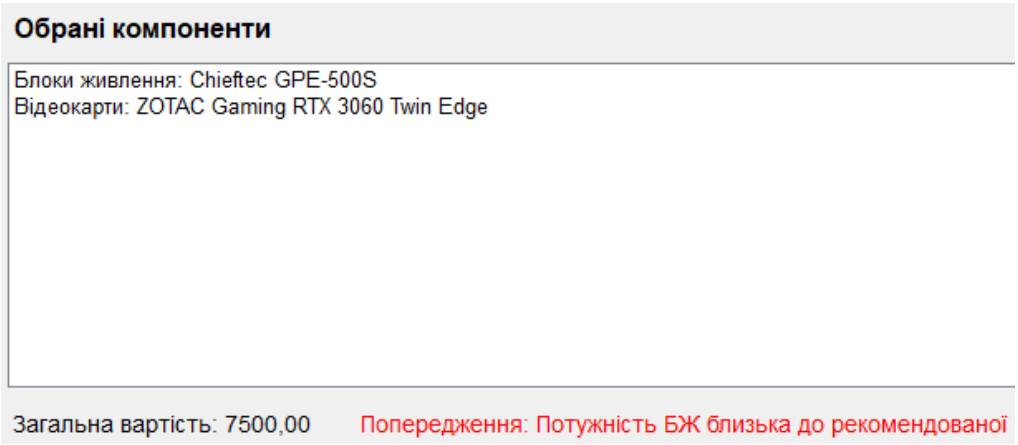


**Рисунок Д.3.4** – Скріншот виконання тестування №3

## Додаток №4 - Скріншоти поетапного тестування (№4)


**Рисунок Д.4.1** – Скріншот виконання тестування №4

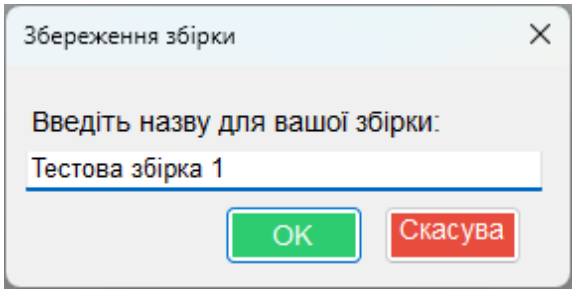


**Рисунок Д.4.2** – Скріншот виконання тестування №4

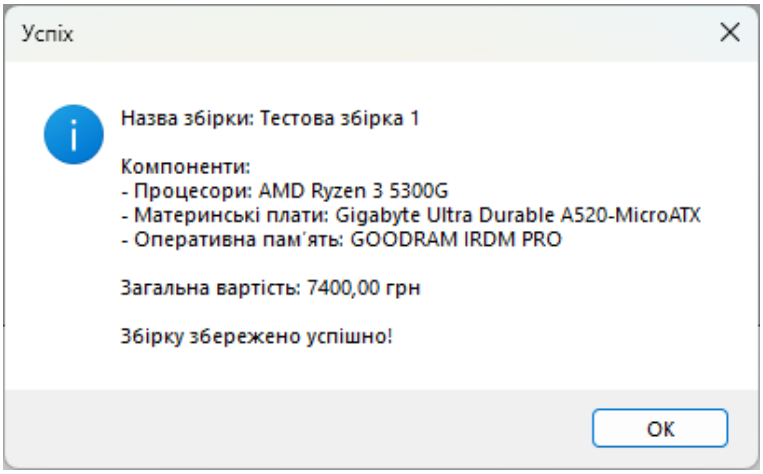


**Рисунок Д.4.3** – Скріншот виконання тестування №4

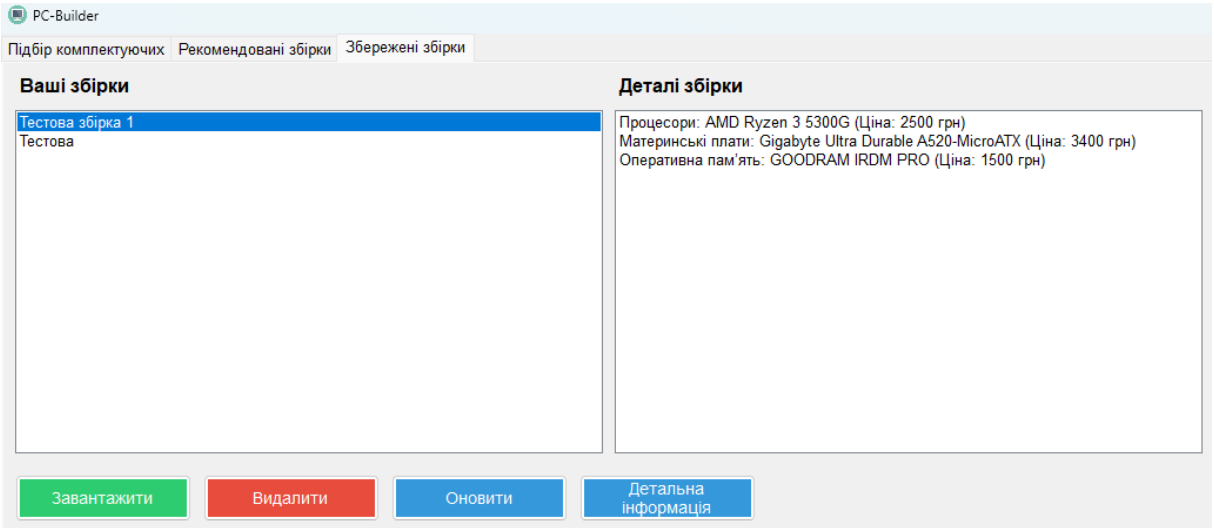


**Рисунок Д.4.4** – Скріншот виконання тестування №4

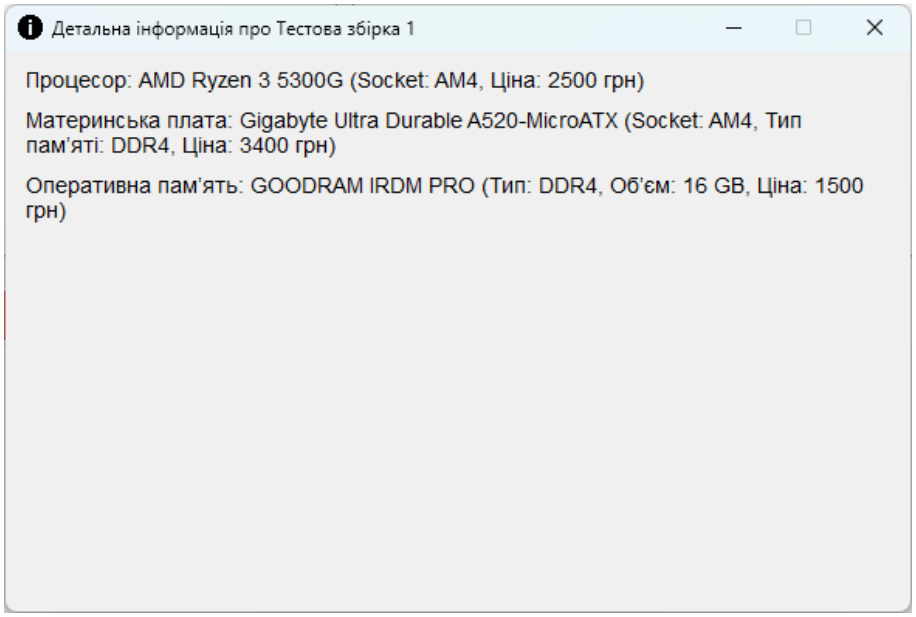


**Рисунок Д.4.5** – Скріншот виконання тестування №4

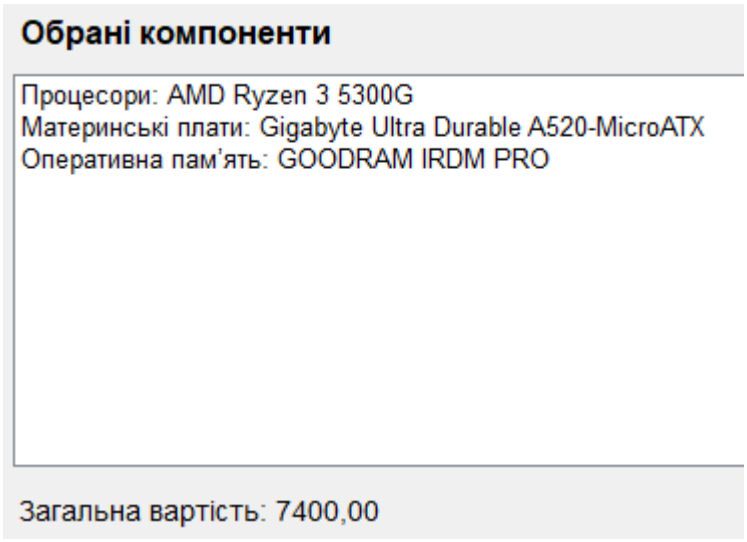


**Рисунок Д.4.6** – Скріншот виконання тестування №4

## Додаток №5 - Скріншоти поетапного тестування (№5)

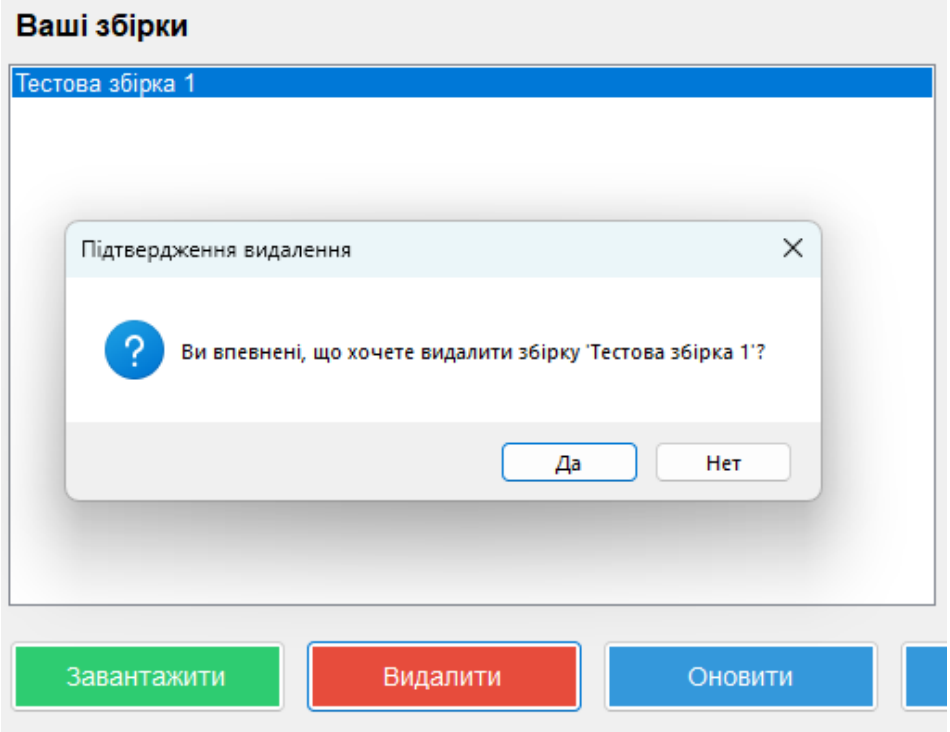


**Рисунок Д.5.1** – Скріншот виконання тестування №5

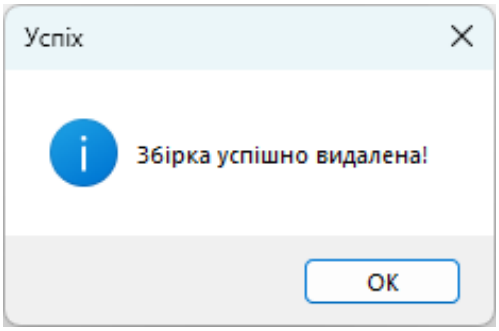


**Рисунок Д.5.2** – Скріншот виконання тестування №5

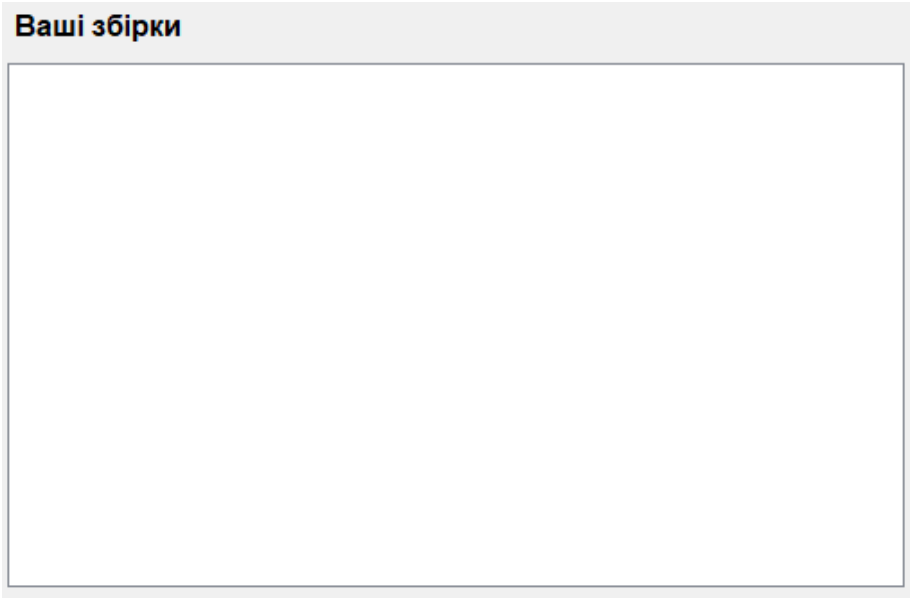


**Рисунок Д.5.3** – Скріншот виконання тестування №5