МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «Система підтримки прийняття рішень з підбору комплектації комп'ютерної техніки»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ-61 Давиденко Данило Андрійович

Кваліфікаційна робота бакалавра захищена на засіданні ЕК з оцінкою _____ «___» _____2020 р.

Науковий керівник _____ к.т.н., доц., Марченко А. В.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Голова комісії _____ Шифрін Д. М.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Суми-2020

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В. В. Шендрик
«__» _____ 2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Давиденко Данило Андрійович

1 Тема роботи *Система підтримки прийняття рішень з підбору комплектації комп'ютерної техніки*

керівник роботи *Марченко Анна Вікторівна, к.т.н., доцент*,

затверджені наказом по університету від « 14 » травня 2020 р. № 0576-III

2 Строк подання студентом роботи «1» червня 2020 р.

3 Вхідні дані до роботи методика зчитування інформації з інтернет-джерел, методика формування правил сумісності та правил оцінювання характеристик компонентів збірки, методика розробки програмного додатку за шаблоном MVVM.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, проектування системи підтримки прийняття рішень з підбору комплектації комп'ютерної техніки, розробка системи підтримки прийняття рішень з підбору комплектації комп'ютерної техніки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність проблеми, постановка задачі, аналіз аналогів, порівняльна таблиця, контекстна діаграма, діаграма декомпозиції 1-го рівня, діаграма варіантів використання, архітектура додатку, модель бази даних, модель

бази знань, засоби реалізації, головне вікно додатку, головне вікно під час підбору, головне вікно в результаті підбору, вікно бази даних, вікно бази знань, висновки.

6. Консультанти розділів роботи:

Розділ	**Консультант**	**Підпис, дата**	
		Завдання видав	**Завдання прийняв**

7.Дата видачі завдання 01.10.2019

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Створення технічного завдання	01.10.2019-21.10.2019	
2	Планування робіт проекту	22.10.2019-02.12.2019	
3	Огляд останніх досліджень і публікацій	03.12.2019-17.12.2019	
4	Аналіз програмних продуктів – аналогів	18.12.2019-10.01.2020	
5	Постановка задачі проекту	11.01.2020-20.01.2020	
6	Функціональне моделювання бізнес-процесів	21.01.2020-15.02.2020	
7	Проектування концепції системи	16.02.2020-05.03.2020	
8	Створення моделі проектування	06.03.2020-17.03.2020	
9	Проектування моделі бази даних	18.03.2020-30.04.2020	
10	Розробка додатку	01.04.2020-15.05.2020	
11	Тестування додатку	16.05.2020-20.05.2020	
12	Оформлення документації	21.05.2020-31.05.2020	
13	Здача пояснювальної записки	01.06.2020	

Студент _____ Давиденко Д.А.
(підпис)

Керівник роботи _____ к.т.н., доц. Марченко А.В.
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра «Система підтримки прийняття рішень з підбору комплектації комп’ютерної техніки».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 17 найменувань та 4-х додатків. Загальний обсяг роботи − 150 сторінок, у тому числі 65 сторінок основного тексту, 2 сторінки списку використаних джерел, 82 сторінки додатків.

Кваліфікаційну роботу бакалавра присвячено розробці системи підтримки прийняття рішень з підбору комплектації комп’ютерної техніки.

В роботі проведено огляд останніх досліджень і публікацій, на тему підбору компонентів персонального комп’ютера, аналіз аналогічних програмних рішень, проектування системи.

У роботі виконано розробку бази даних, бази знань, і додатку, що містить 3 головних вікна:

- головне вікно додатку для формування збірки;
- вікно бази даних, для внесення змін до бази даних, оновлення її таблиць збереження і відновлення її вмісту;
- вікно бази знань для зміни активності та пояснень до правил сумісності та оцінки характеристик.

Результатом проведеної роботи є створена система підтримки прийняття рішень з підбору комплектації комп’ютерної техніки.

Практичне значення роботи полягає у наданні недосвідченим користувачам інструмента, за допомогою якого вони зможуть швидко та ефективно сформувати комп’ютерну збірку відповідно до встановлених ним потреб.

Ключові слова: персональний комп’ютер, комп’ютерна збірка, комплектуючі, характеристики, MVVM, віконний додаток.

ЗМІСТ

ВСТУП

Вибір комплектуючих для персонального комп’ютера (ПК) – складний процес, що потребує професійного підходу для отримання максимальної продуктивності з мінімальними фінансовими витратами. Необхідно вивчати сумісність компонентів, актуальні моделі в різних цінових категоріях і особливості їх встановлення. Але, очевидно, що не всі люди готові витрачати велику кількість часу для отримання таких навичок. Найпоширенішим виходом з такої ситуації є придбання вже зібраного ПК, ґрунтуючись лише на його загальних характеристиках. Проте він може мати ціну, набагато більшу від фактичної сумарної ціни його комплектуючих, а також, комбінація цих комплектуючих може бути підібрана далеко не досконало.

Головною метою розроблюваного проекту буде створення допоміжного програмного додатку, для підбору комплектуючих ПК, ґрунтуючись на потребах користувача, тобто на тому, для чого буде використовуватись комп’ютер.

Важливою задачею, що постає при реалізації такого додатку є створення максимально повної бази знань. Така база знань повинна містити правила, що встановлюють сумісність, тобто можливість використання, однієї комплектуючої з іншою, а також оцінки значень характеристик комплектуючих, що дозволять правильно визначити відповідність сформованої збірки до обраних користувачем потреб. Ще однією задачею є створення зрозумілого і зручного інтерфейсу з системою пояснень/підказок. Завдяки цьому користувач зможе підвищити свою обізнаність у цій галузі і матиме краще представлення про те, чому додаток пропонує йому саме таку комплектацію. Також, дуже важливо, щоб робота додатку була оптимізована, адже при формуванні збірок або підборі комплектуючих з великої за об’ємом бази даних, слід очікувати уповільнення його роботи, а саме, затримки видачі результатів підбору користувачеві, що, очевидно, зменшить його бажання використовувати додаток наступного разу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Серед літературних джерел на тему підбору комплектуючих ПК переважають статті від інтернет-магазинів, які у доволі короткій та зрозумілій формі намагаються пояснити покупцю на що саме треба орієнтуватися під час формування збірки нового ПК. Одна з таких статей написана Денисом Чумаєвим з компанії Larga [1]. У своїй статті він демонструє приклади конфігурації комп'ютера в залежності від його призначення та надає опис загальної концепції кожної з конфігурацій. Також, розглядається питання вибору кожного окремого компонента збірки для відповідних цілей, і особливостей, які має та чи інша комплектуюча.

Наступним джерелом інформації про формування збірки є статті від незалежних ресурсів, які надають загальне розуміння про технологію підбору компонентів для збірки ПК. Серед таких джерел виділяється сайт «Цифровая жизнь вокруг нас» [2], на якому можна знайти статті про різні аспекти сучасних цифрових технологій, а отже, і про апаратне забезпечення комп'ютера. Стаття, в якій розглядається тема підбору комплектуючих містить інформацію про способи збірки комп'ютера: власноруч, на замовлення тощо, а також переваги та недоліки кожного з варіантів. Приводиться опис типів збірок за градацією ціни та цілей використання. Надається опис окремих комплектуючих збірки, їх видів, переваги та недоліки.

Ще одним, нечисельним типом ресурсів з інформацією про конфігурацію ПК є наукові та дипломні роботи, проте їх знаходження у відкритому доступі доволі рідке явище. Дипломна робота Абрашевой С.Ю. «Аппаратный компоненты офисного ПК» [3] містить відомості про загальні вимоги до комплектуючих комп'ютерів, що використовуються офісними працівниками. У своїй роботі автор надає загальну інформацію про апаратні засоби ПК: основні компоненти та їх характеристика. Проводиться аналіз та оцінка кожного з компонентів збірки, які мають бути

встановлені в офісному комп’ютері. Також, розглядається питання техніки безпеки на робочому місці користувача ПК: організація робочого простору та загальні вимоги до техніки безпеки.

1.2 Аналіз програмних продуктів – аналогів

Для вирішення задачі підбору комплектуючих в залежності від потреб користувача немає окремих програмних додатків у відкритому доступі. Проте деякі інтернет-ресурси, на яких можна придбати комп’ютерне апаратне забезпечення, мають спеціальний модуль, що дозволяє формувати збірку власноруч, або скористатися вже наявними рекомендованими варіантами збірок. Подібних реалізацій в Інтернеті достатньо багато, тому розглянемо лише ті, що мають найбільш розвинений функціонал.

«Конфигуратор компьютера NerdPart» [4]. Вигляд головної сторінки конфігуратора наведено на рисунку 1.1.

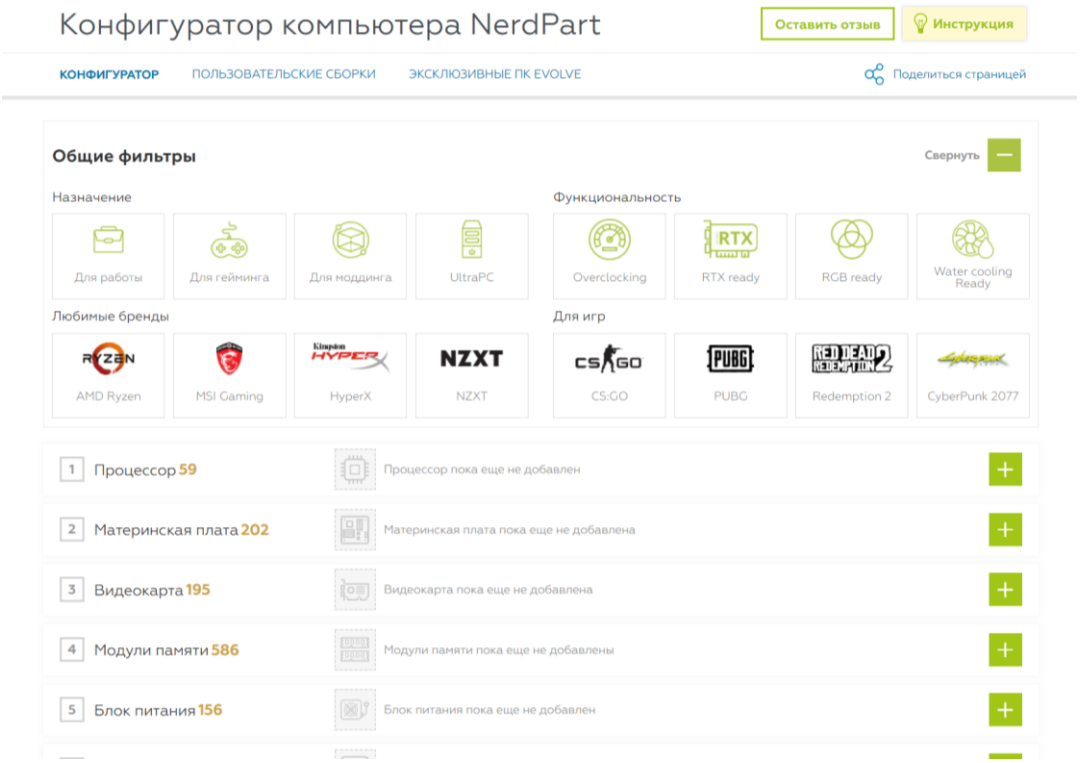


Рисунок 1.1 – Конфігуратор комп’ютера від Telemart.ua

У даного конфігуратора наявний доволі широкий набір фільтрів, що допомагають користувачеві звузити круг комплектуючих, які відповідають його потребам. Є також список збірок рекомендованих іншими користувачами. Наявна перевірка на сумісність компонентів, яка працює таким чином, що при виборі певної комплектуючої списки всіх інших комплектуючих звужується до сумісного з вже обраними компонентами. Проте, даний конфігуратор не відображає ступінь відповідності сформованої збірки до потреб користувача, тобто для цього йому необхідно звертатися до спеціаліста або покладатися на власні знання.

«Конфигуратор системного блока» від інтернет-магазину Ironbook.ru [5]. Вигляд головної сторінки конфігуратора наведено на рисунку 1.2.

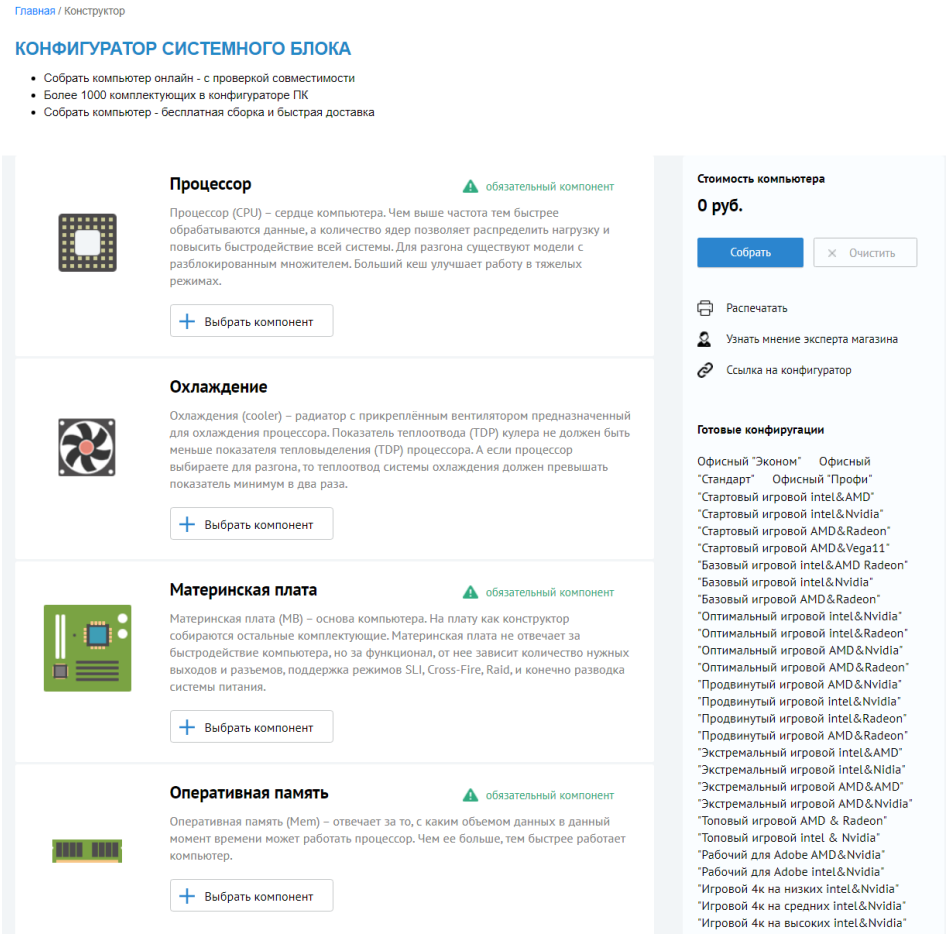


Рисунок 1.2 – Конфігуратор системного блоку від Ironbook.ru

На відміну від вже розглянутого, цей конфігуратор не має фільтрів, що могли б звузити коло пошуку, проте в нього є список збірок, рекомендованих для відповідних потреб користувачів. Приємною особливістю є невеличкий опис призначення кожного з компонентів, що може допомогти користувачеві визначитись з тим, яким комплектуючим слід надати більше уваги при виборі. Також, тут наявна перевірка сумісності комплектуючих, що реалізована аналогічно з минулим прикладом, і відсутня інформація про відповідність сформованої збірки.

«Конфигуратор компьютера», який знаходиться на сайті dns-shop.ru [6]. Вигляд головної сторінки конфігуратора наведено на рисунку 1.3.

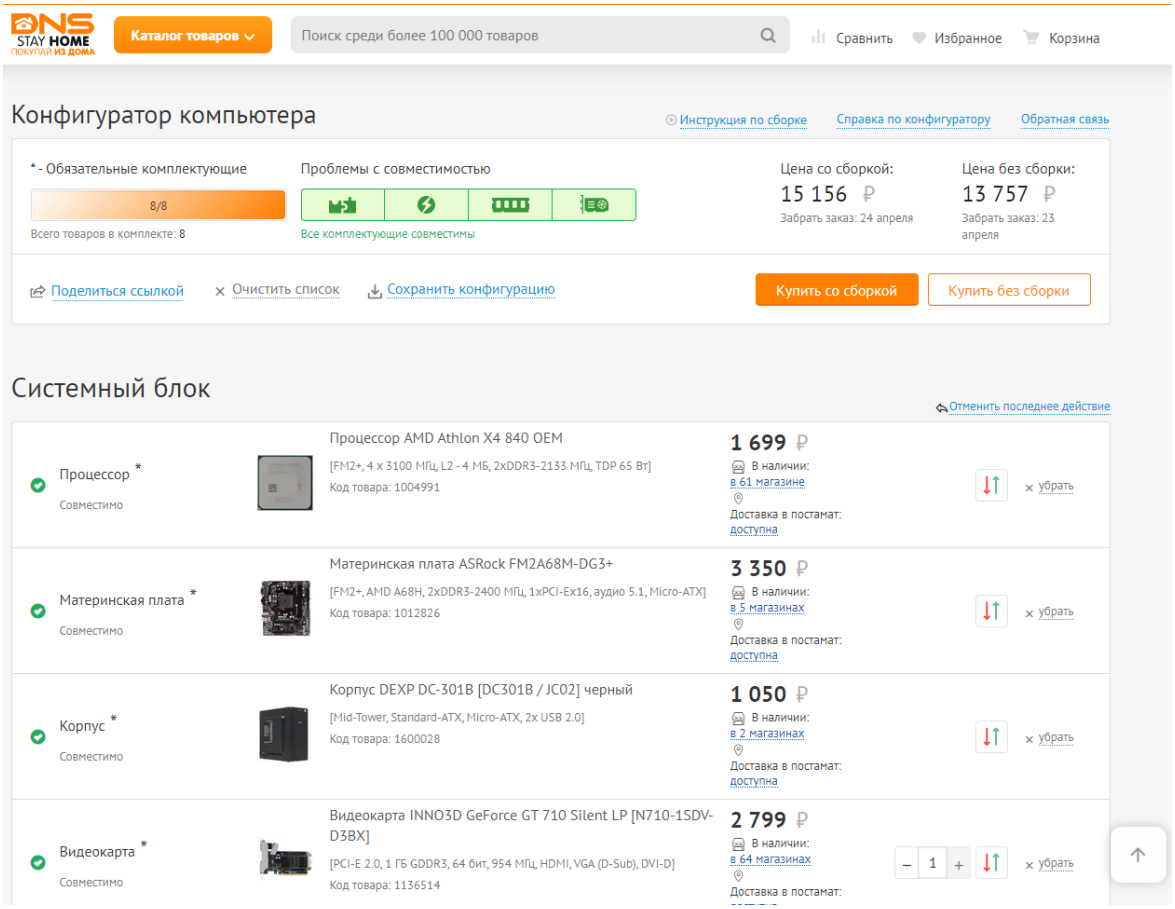


Рисунок 1.3 – Конфігуратор комп’ютера від dns-shop.ru

Зазначений конфігуратор має дуже багато опцій, що допомагають підібрати компоненти: фільтри різних категорій для кожної з комплектуючих, шкала з кількістю обов’язкових компонентів, індикатори, що надають інформацію про сумісність обраних компонентів, і вказують на те, які саме компоненти є несумісними, також, наявна опція збереження збірки для авторизованих користувачів. Після вибору окремого компоненту з’являються підказки, які, зважаючи на характеристики та наявність або відсутність інших компонентів збірки, сповіщають користувача про певні особливості поточної збірки. Даний конфігуратор дозволяє підібрати додаткові компоненти (ті, що не є обов’язковими для функціонування збірки), периферійні пристрої, а також програмне забезпечення (ПЗ). Інформація про загальну відповідність сформованої збірки до потреб користувача не надається.

«Конфигуратор компьютера» від інтернет-магазину can.ua [7]. Вигляд головної сторінки конфігуратора наведено на рисунку 1.4.

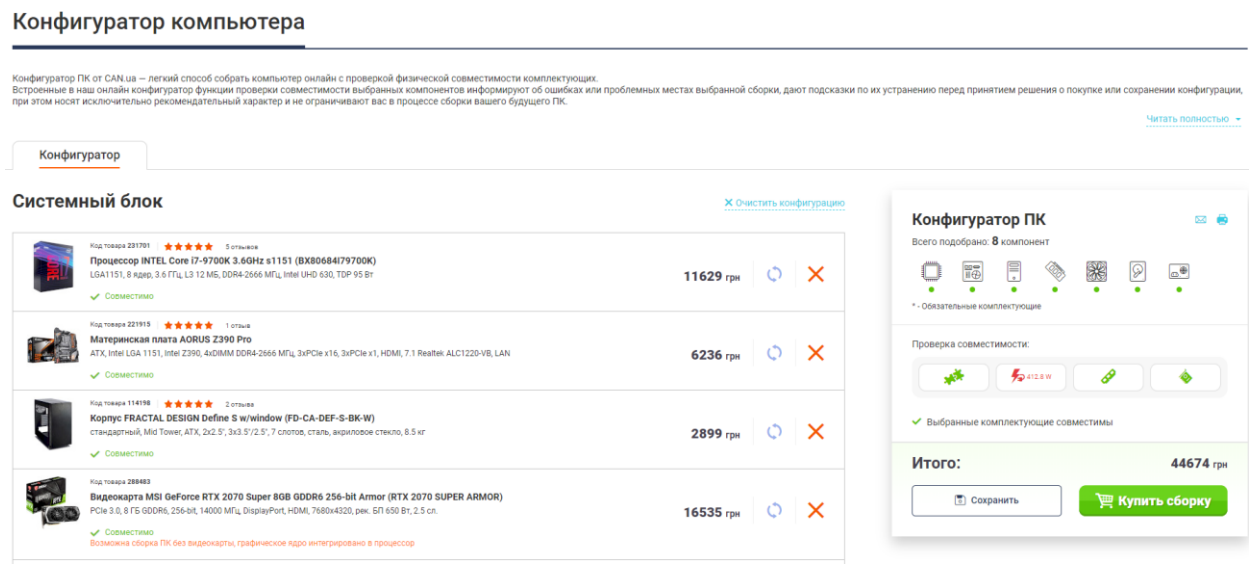


Рисунок 1.4 – Конфігуратор комп’ютера від can.ua

Останній з представлених модулів конфігурації комп’ютера надає широкий набір інструментів для підбору комплектуючих, який майже ідентичний функціоналу минулого прикладу: фільтри, розподілені за категоріями, для кожної комплектуючої, перевірка сумісності, і відображення причини несумісності компонентів (у разі їх виникнення), індикатори, що відображають які обов’язкові компоненти вже були обрані, і які, ще треба підібрати для отримання мінімального функціонуючого набору, для авторизованих користувачів доступна функція збереження сформованої збірки. Наявні опції для підбору периферії та ПЗ. Загальна відповідність створеної збірки до потреб користувача не визначається.

Тепер складемо порівняльну таблицю за визначеними функціональними критеріями (табл. 1.1). Введемо позначення: якщо певний конфігуратор відповідає зазначеному критерію, то він має позначку «+», інакше «-».

Таблиця 1.1 – Порівняння аналогів

	Telemart.ua	Ironbook.ru	dns-shop.ru	can.ua	PCConfigurator
Фільтри для характеристик комплектуючої	+	-	+	+	+
Рекомендовані збірки	+	+	-	-	-
Перевірка сумісності компонентів	+	+	+	+	+
Опис призначення компонентів	-	+	+	-	+
Позначення обов’язкових компонентів збірки	-	+	+	+	+
Збереження сформованої збірки	+	-	+	+	+
Функціонування без доступу до мережі Інтернет	-	-	-	-	+
Оцінка відповідності сформованої збірки	-	-	-	-	+

1.3 Постановка задачі

Метою виконання даної роботи є розробка програмного додатку для системи підтримки прийтяння рішень (СППР), що представлятиме собою інструмент для підбору необхідних компоненти ПК в залежності від потреб користувача. Передбачається, що програмний додаток вирішуватиме ряд задач:

- надання зручного та зрозумілого інтерфейсу для пошуку необхідних компонентів;
- встановлення достатньої кількості фільтрів для звуження кола пошуку необхідної комплектуючої або збірки, відповідно до потреб користувача;
- визначення відповідності сформованої збірки до встановлених потреб;
- збереження сформованої збірки у файл, а також її завантаження з відповідного файлу;
- обґрунтування наданих рекомендацій у лаконічній та достатньо зрозумілій формі.

Для досягнення мети проекту було виділено певні завдання, які необхідно виконати:

- дослідження та аналіз предметної області;
- формування технічного завдання;
- виконання планування проекту;
- проектування моделі створюваного додатку;
- створення бази даних з інформацією про комплектуючі;
- створення бази знань з правилами на сумісність компонентів і оцінками характеристик комплектуючих;
- реалізація додатку;
- тестування додатку;
- створення загальної документації.

Детальний опис мети, задач, а також методи й технології, що використовуються для вирішення задачі наводяться у додатку А. Планування робіт зі створення системи підтримки прийняття рішень наведено у додатку Б.

2 ПРОЕКТУВАННЯ СИСТЕМИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ

2.1 Проектування системи

2.1.1 Модель бізнес-процесів у нотації IDEF0

У даному пункті приводиться модель системи в нотації IDEF0 [8]. Створено контекстну діаграму, а також діаграми декомпозиції 1-го і 2-го рівнів. На вхід системи можуть поступати:

- вимоги користувача, що є основним критерієм для встановлення шаблонів оцінок характеристик, що дозволяють проводити оцінку відповідності сформованої користувачем збірки.

Елементами керування є:

- записи бази даних, що утворюють набори комплектуючих певного типу, які додає користувач до збірки при її формуванні;
- правила бази знань, за якими перевіряється сумісність компонентів збірки, що формується;
- оцінки характеристик, за допомогою яких відбувається розрахунок відсотка відповідності сформованої користувачем збірки до встановлених ним потреб.

В якості механізмів виступають:

- модуль формування збірки, з використанням якого користувач може знаходити необхідні йому компоненти (за фільтрами або без), додавати чи видаляти компоненти з поточної збірки;
- модуль визначення відповідності, з використанням якого відбувається підрахунок компонентів у поточній збірці, визначення типів цих комплектуючих, перевірка сумісності, обчислення відсотка відповідності.

На виході системи ми можемо отримати:

- сформовану збірку, що отримав користувач в результаті підбору комплектуючих;
- оцінку відповідності збірки, визначену, відповідно до потреб користувача і сформованої ним збірки.

На рисунку 2.1 наведено контекстну діаграму (0-го рівня), що представляє головний процес – «Підбір комплектації комп’ютерної техніки».

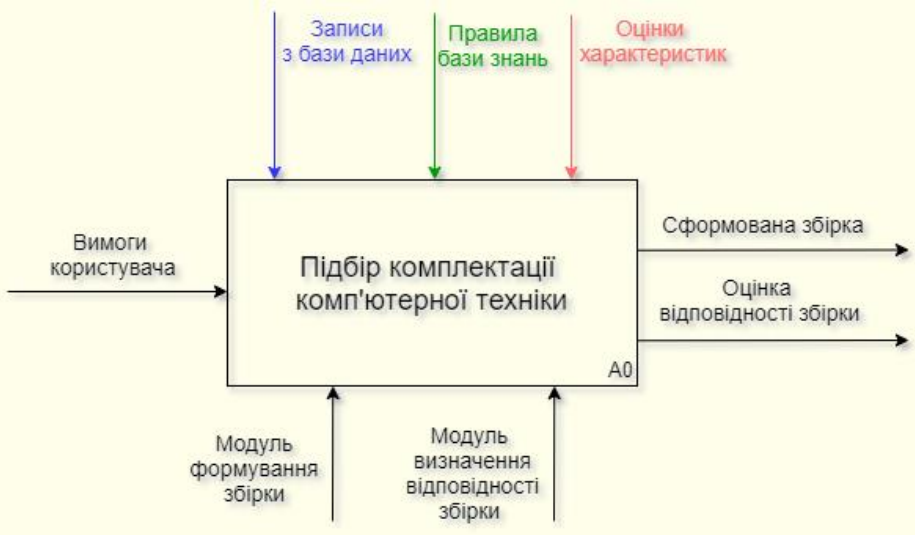


Рисунок 2.1 – Контекстна діаграма

На рисунку 2.2 зображена діаграма декомпозиції 1-го рівня для процесу «Підбір комплектації комп’ютерної техніки» (A0), що складається з двох процесів: «Формування збірки» (A1), «Розрахунок відповідності збірки» (A2).

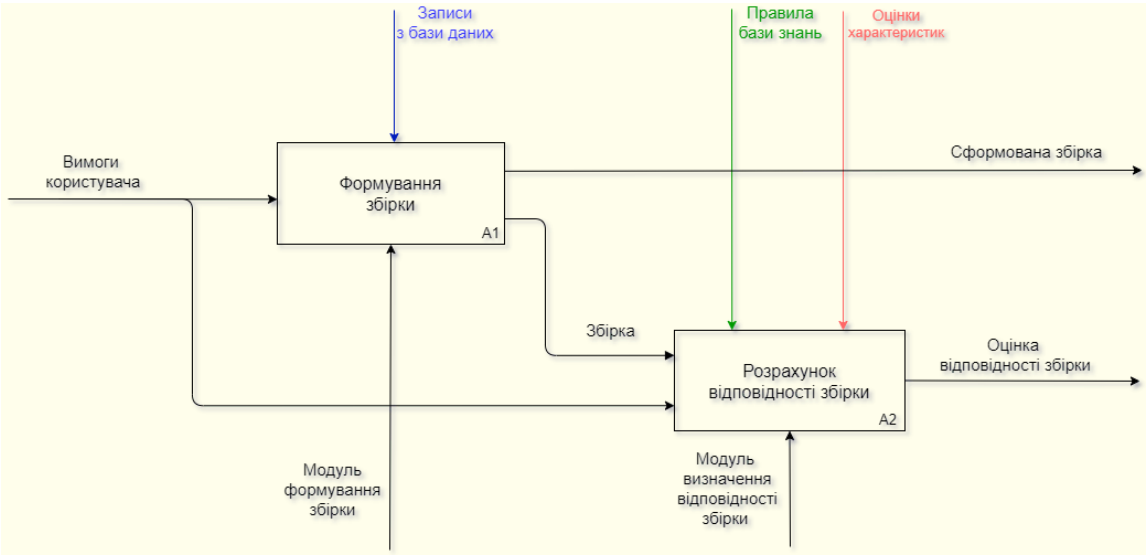


Рисунок 2.2 – Діаграма декомпозиції 1-го рівня

На рисунку 2.3 наведено діаграму декомпозиції 2-го рівня для процесу «Розрахунок відповідності збірки» (A2), що складається з трьох процесів: «Визначення кількості й типів обраних компонентів», «Перевірка сумісності компонентів» і «Обчислення відсотка відповідності».

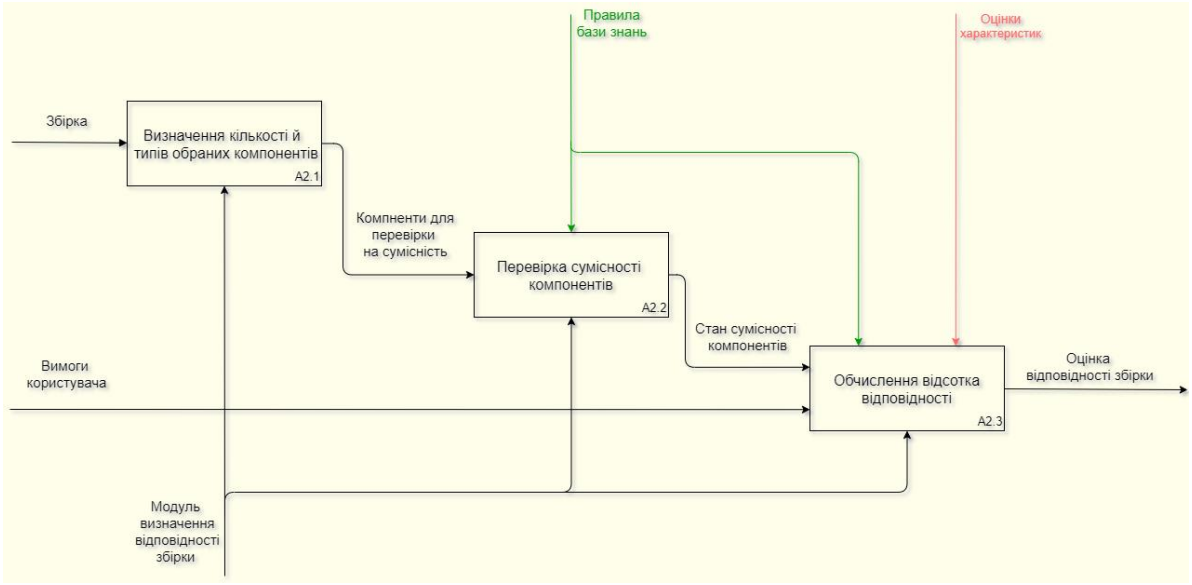


Рисунок 2.3 – Діаграма декомпозиції 2-го рівня процесу «Розрахунок відповідності збірки»

2.1.2 Модель використання системи

Даний пункт містить діаграму варіантів використання (Use Case Diagram) [9], а також короткий опис акторів, прецедентів і артефактів системи наведених у цій діаграмі. Виділено таких акторів (4):

- Клієнт: користувач, що може зберігати/завантажувати і формувати збірки;
- Адміністратор: користувач, якому доступні всі опції клієнта, а також він має доступ до управління базою даних і базою знань;
- MSSQL: база даних комплектуючих;
- База знань: набір правил сумісності, правил оцінювання характеристик компонентів та оцінок їх характеристик з методами їх обробки.

Наступні варіанти використання (ВВ) (16):

- Авторизація: підтвердження того, що користувач є адміністратором, для отримання доступу до бази даних та бази знань;
- Додавання записів до бази даних;
- Редагування записів бази даних;
- Видалення записів з бази даних;
- Створення резервної копії: збереження конфігурації і записів бази даних;
- Відновлення за резервною копією: завантаження конфігурації і записів бази даних;
- Оновлення інформації про компоненти: зчитування інформації про комплектуючі обраного типу з інтернет-ресурсів, і заповнення відповідної таблиці зчитаними записами, при цьому попередні записи таблиці видаляються;
- Зміна активності та підказок правил: зміна стану правил: увімкнене/вимкнене, та встановлення пояснень, що надаються при застосуванні відповідно правила;
- Збереження збірки: створення файлу формату .pcconf (власний формат), в якому зберігається набір комплектуючих, обраних користувачем;
- Завантаження збірки: заповнення області збірки комплектуючими, список яких знаходиться у файлі формату .pcconf;

- Формування збірки: включає в себе можливості додавання, видалення та пошуку компонентів збірки, за умови достатньої сформованості збірки можливо визначення оцінки відповідності;
- Додавання комплектуючої до збірки;
- Видалення комплектуючої зі збірки;
- Фільтрування компонентів: встановлення обмежень на значення характеристик компонентів певного типу для звуження кола пошук комплектуючих цього типу;
- Визначення оцінки відповідності: підрахунок кількості обраних компонентів, визначення їх сумісності та обчислення відсотка відповідності.

Такі артефакти (3):

- Характеристики комплектуючих: набір числових, рядкових і логічних значень для комплектуючих кожного з типів;
- Правила сумісності: статичний набір обмежень на можливість утворення збірки, що накладаються на пари компонентів визначених типів;
- Шаблони оцінок характеристик: коефіцієнти важливості характеристик для кожного з типів комплектуючих відповідно до режимів підбору.

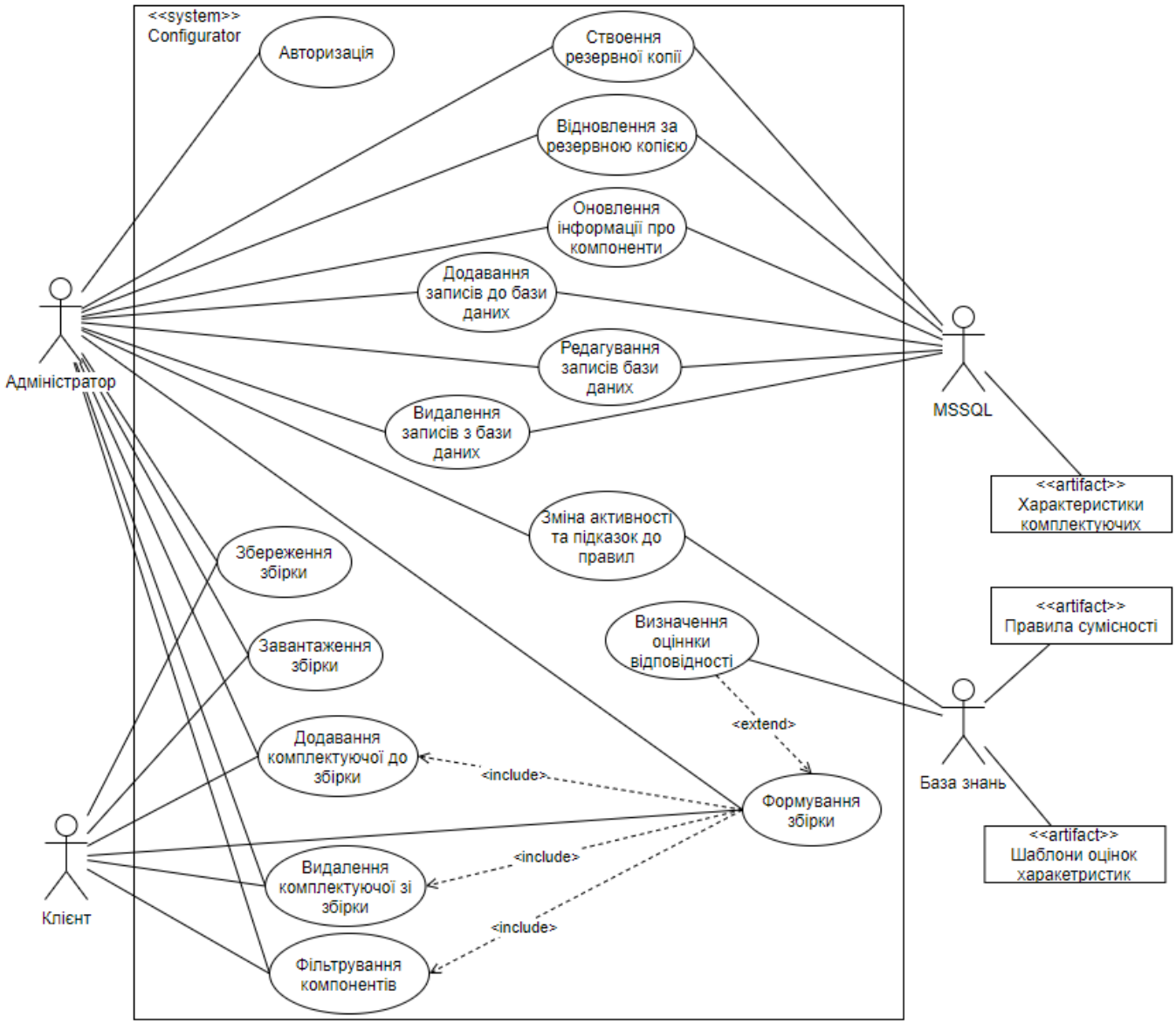


Рисунок 2.4 – Діаграма варіантів використання

2.1.3 Модель проектування

У цьому пункті наводиться діаграма класів створюваного додатку [10], а також діаграми діяльності для кожного з варіантів використання [11].

На рисунку 2.5 приведена повна діаграма класів створюваного додатку.

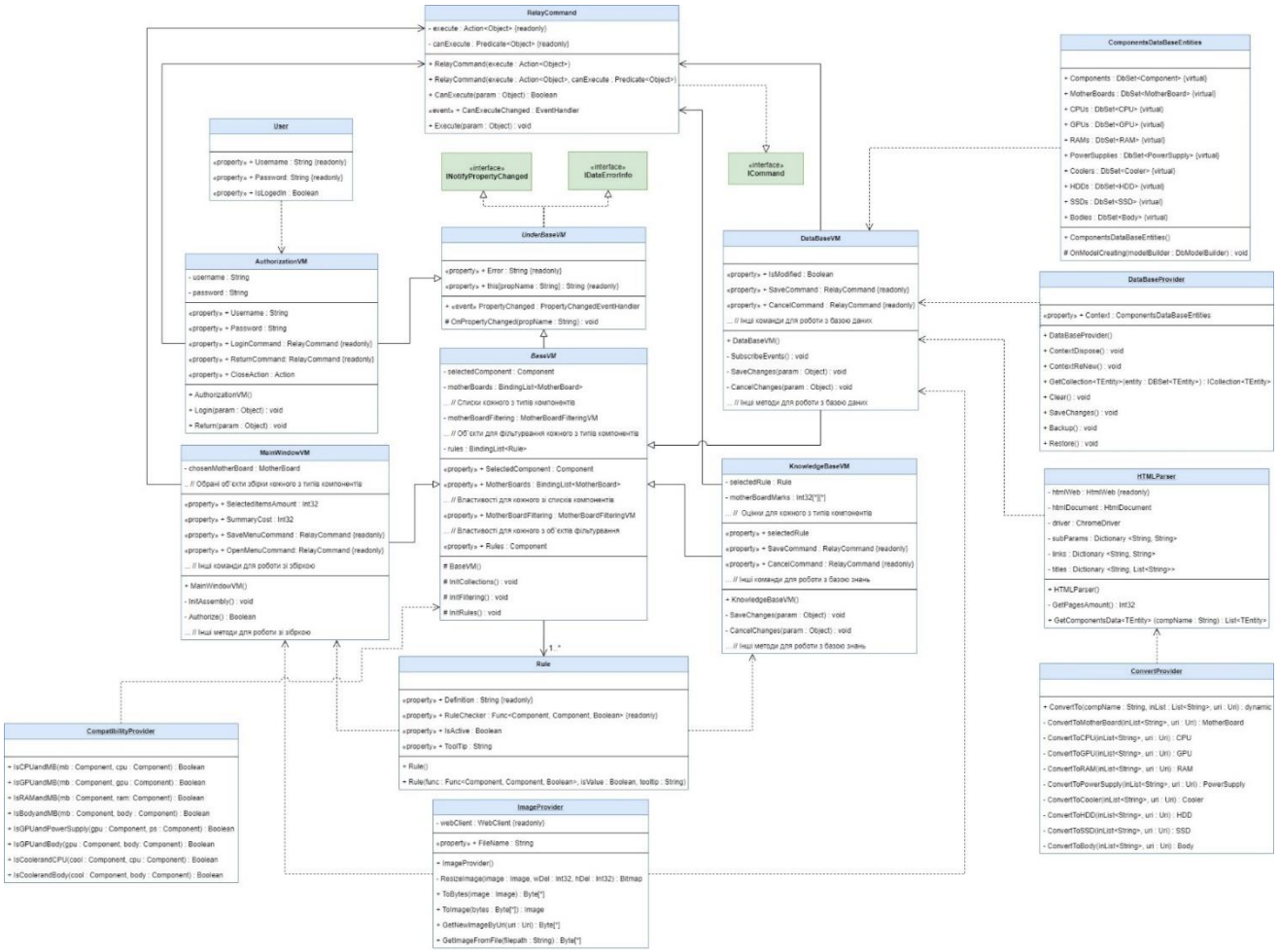


Рисунок 2.5 – Діаграма класів

Також, для більш чіткого розуміння відносин між класами була розроблена спрощена діаграма класів (рисунок 2.6), що не містить інформації про поля, властивості та атрибути класів.

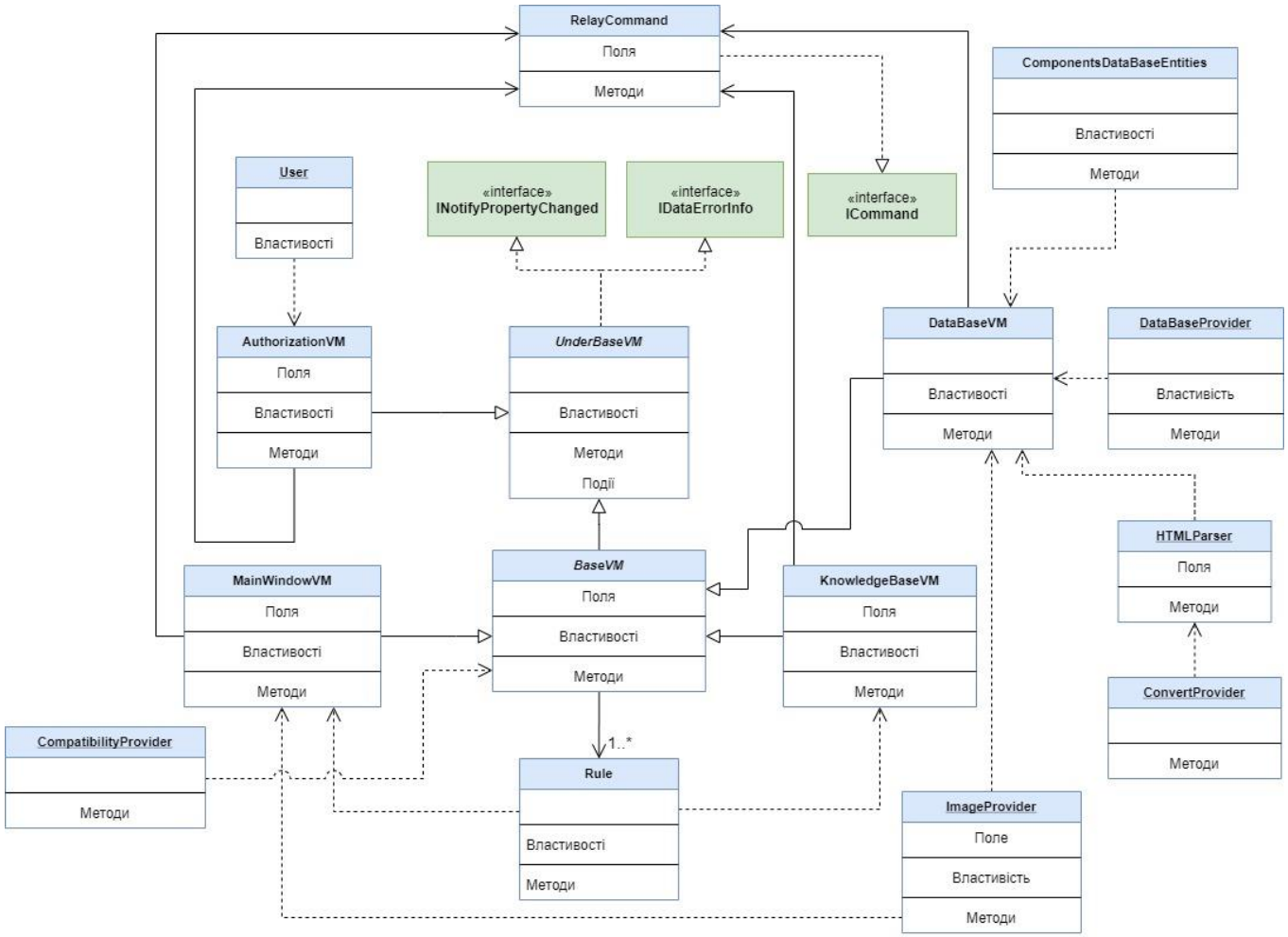


Рисунок 2.6 – Спрощена діаграма класів

Діаграми дільяності для кожного з варіантів використання системи наведені у додатку В.

2.2 Проектування моделі бази даних

Даний розділ містить діаграми потоків даних (DFD) [12] 0-го рівня та декомпозиції 1-го рівня, а також концептуальну і даталогічну моделі бази даних комплектуючих. На рисунках 2.7 і 2.8 наведені діаграми потоків даних 0-го та 1-го рівня.

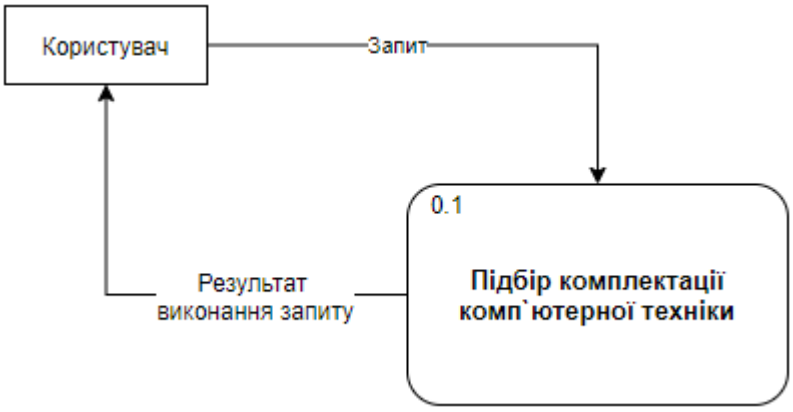


Рисунок 2.7 – Діаграма потоків даних 0-го рівня

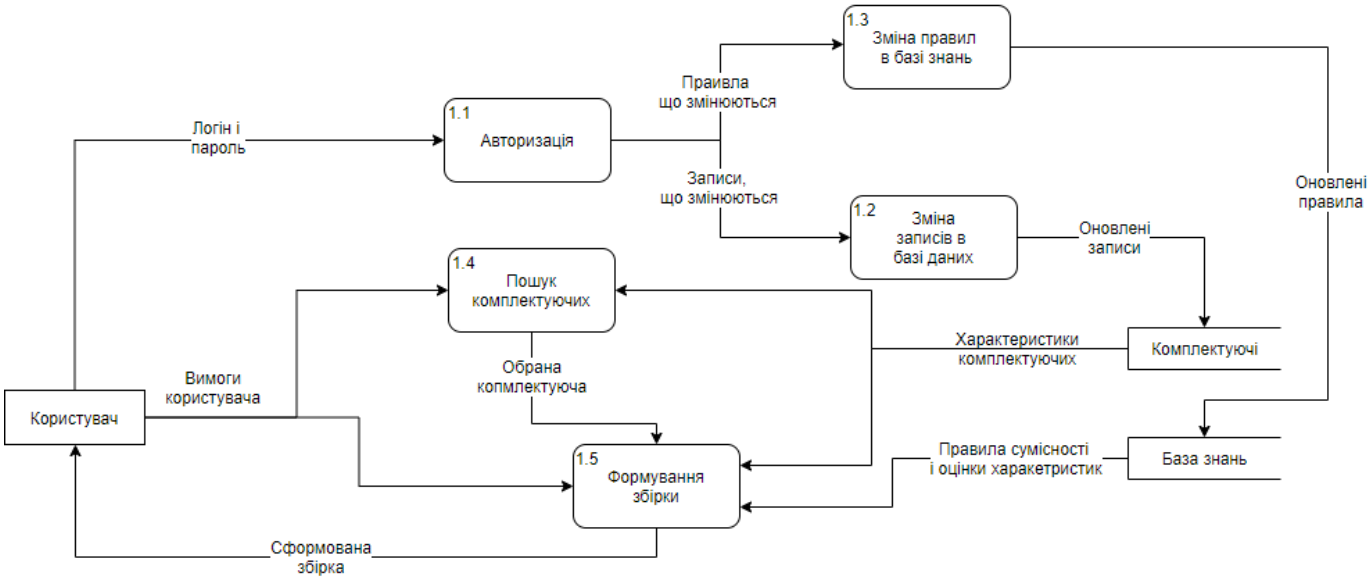


Рисунок 2.8 – Діаграма потоків даних, декомпозиція 1-го рівня

Для проектування бази даних комплектуючих було застосовано підхід TPT (таблиця на тип), що передбачає створення загальної таблиці (сутності), яка зберігає тільки ті властивості, що притаманні всім сутностям нащадкам, при цьому властивості, що відносяться лише до похідних сутностей, зберігаються в окремих таблицях. Таким чином, було створено ряд сутностей (*MotherBoard, CPU, GPU, RAM, PowerSupply, Cooler, HDD, SSD, Body*), що відповідають окремим типам компонентів зі своїми технічними характеристиками, а також єдину, абстрактну сутність *Component*, яка містить загальні для всіх інших типів властивості компонентів.

Наслідування кожної з сутностей типів комплектуючих від загальної сутності відбувається за рахунок неявного створення у кожній з цих таблиць ідентифікатору, що одночасно виступає первинним та зовнішнім ключем, який вказує на первинний ключ у загальній таблиці. На рисунку 2.9 наведено концептуальну модель бази даних.

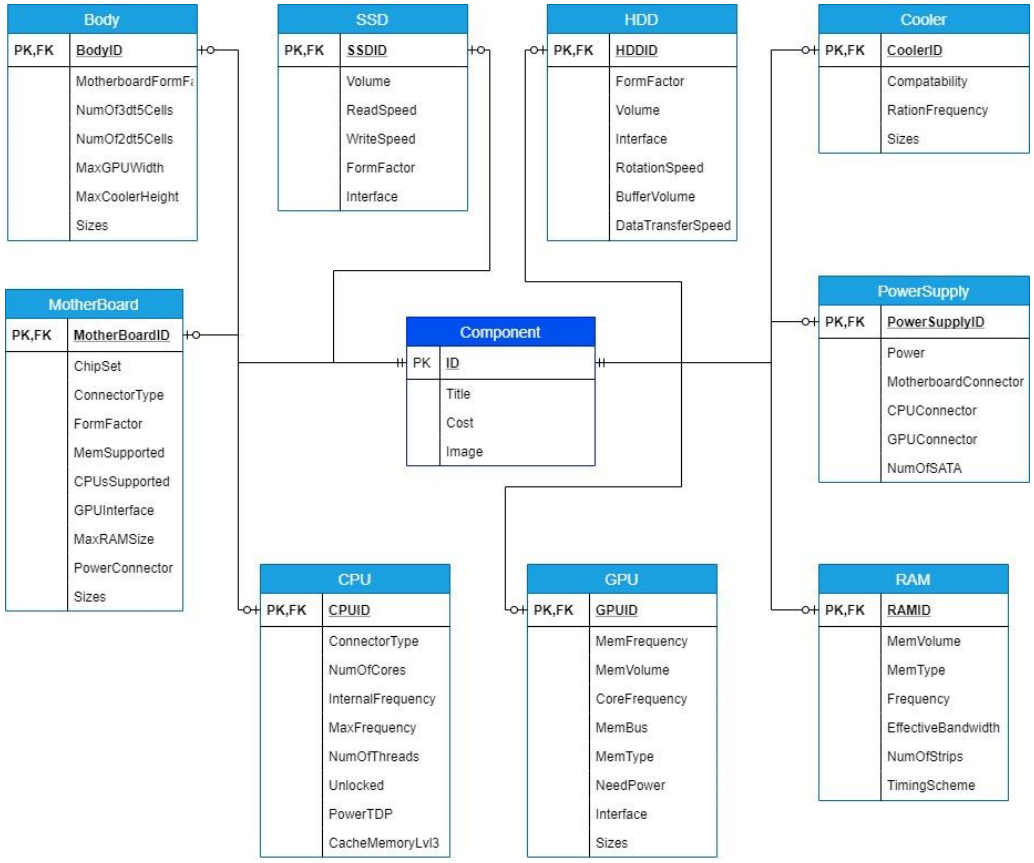


Рисунок 2.9 – Концептуальна модель бази даних комплектуючих

На рисунку 2.10 продемонстровано даталогічну модель бази даних, на якій, окрім відображення зв’язків між сутностями, наведено типи даних властивостей, що зберігаються у кожній з таблиць.

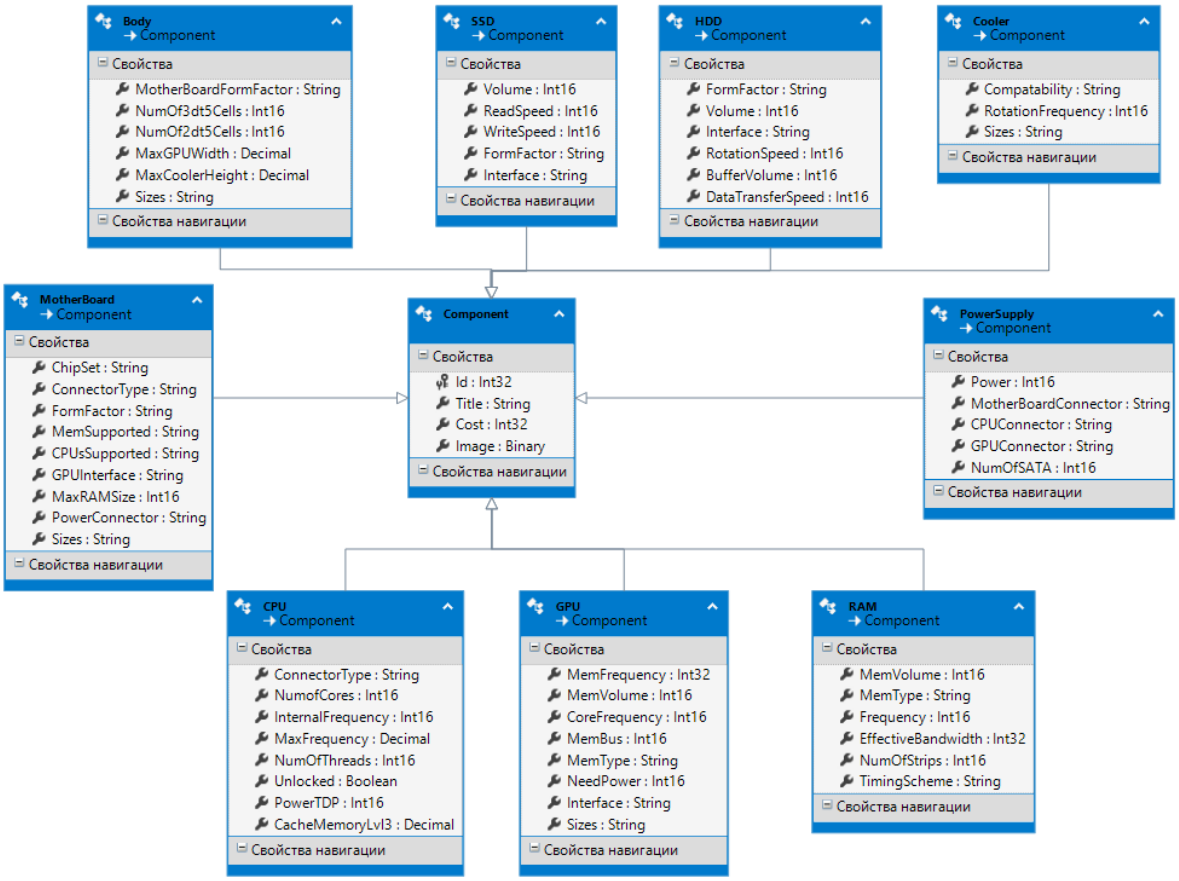


Рисунок 2.10 – Даталогічна модель бази даних комплектуючих

Для зберігання правил сумісності комплектуючих та правил оцінки їх характеристик було створено дві непов’язані між собою сутності: *CompatibilityRule* та *EvaluationRule* (рисунок 2.11). Для перевірки виконання правил використовуються методи на мові програмування, що прив’язуються до відповідних правил на початку виконання програми.

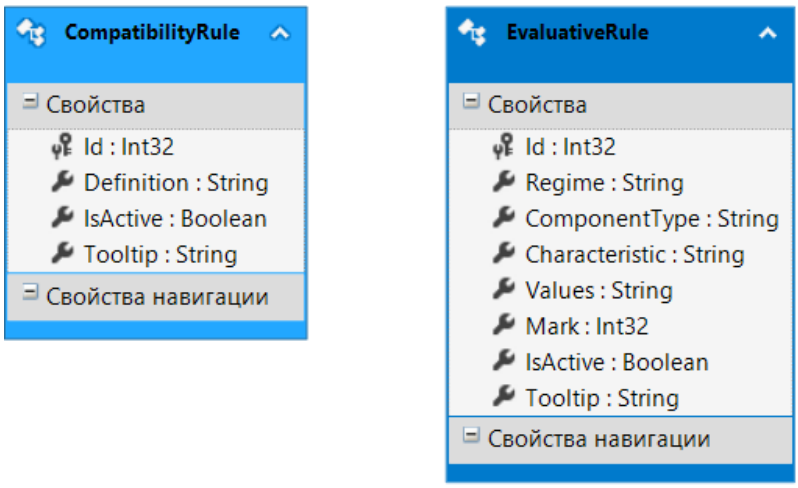


Рисунок 2.11 – Сутності правил бази знань

Перша з них містить такі атрибути (не включаючи ідентифікатор):

- *Definition*: опис правила сумісності, в якому наводяться компоненти, сумісність яких перевіряється, а також характеристика за якою відбувається визначення сумісності;
- *IsActive*: логічне значення, що відповідає за те, чи використовується дане правило при визначенні сумісності;
- *Tooltip*: пояснення до правила, що відображатиметься, якщо правило використовується, а компоненти збірки, що формується, несумісні згідно з цим правилом.

Друга, містить наступні атрибути (не включаючи ідентифікатор):

- *Regime*: режим використання правила, який відповідає певному типу комп’ютерної збірки, що може відповідати потребам користувача;
- *ComponentType*: тип компонента, значення характеристики якого зберігаються;
- *Characteristic*: назва характеристики, значення якої відповідають даному правилу;
- *Values*: значення характеристик (одне значення/перелік/діапазон);
- *Mark*: оцінка відповідна до режиму використання правила;
- *IsActive*: логічне значення, що відповідає за те, чи використовується дане правило при визначенні оцінки відповідності;

- *Tooltip*: пояснення до правила, що відображатиметься, якщо правило використовується, а сформована збірки не цілком відповідає встановленим потребам користувача (не дорівнює 100%).

3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ ДЛЯ СИСТЕМИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ

3.1 Архітектура програмного додатку

Архітектура створюваного додатку (рисунок 3.1) відповідає шаблону проектування MVVM (Model-View-ViewModel) [13], притаманному більшості додатків на базі підсистеми Windows Presentation Foundation (WPF). MVVM передбачає наявність 3-х головних компонентів: моделі, що містить бізнес-логіку роботи додатку; вигляду, який представляє собою користувальницький інтерфейс; модель вигляду, яка впроваджує зв'язок між даними, що надходять з моделі та елементами інтерфейсу, за допомогою яких ці дані відображаються.

Модель передбачає двосторонній обмін даними з базою даних і базою знань, проте зв'язок моделі з базою даних відбувається через ORM (Object-Relational Mapping) Entity Framework [14]. Ця ORM представляє собою технологію програмування, за допомогою якої впроваджується абстракція над доступом до бази даних зі сторони додатку, що реалізує концепції об'єктно-орієнтованої мови програмування. Таким чином, використання цієї абстракції надає можливість працювати з елементами, обраної програмістом, мови програмування, замість елементів реляційної бази даних; DDL (Data Definition Language) та DML (Data Manipulation Language) операції, адаптовані під обрану мову; а також універсальний доступ до будь-якої системи управління базами даних.

Entity Framework (EF) реалізує шаблони «Repository», що надає можливість управління джерелом даних (сутністю бази даних) та «Unit Of Work», який дозволяє мати єдине підключення для всіх репозиторіїв (джерел даних).

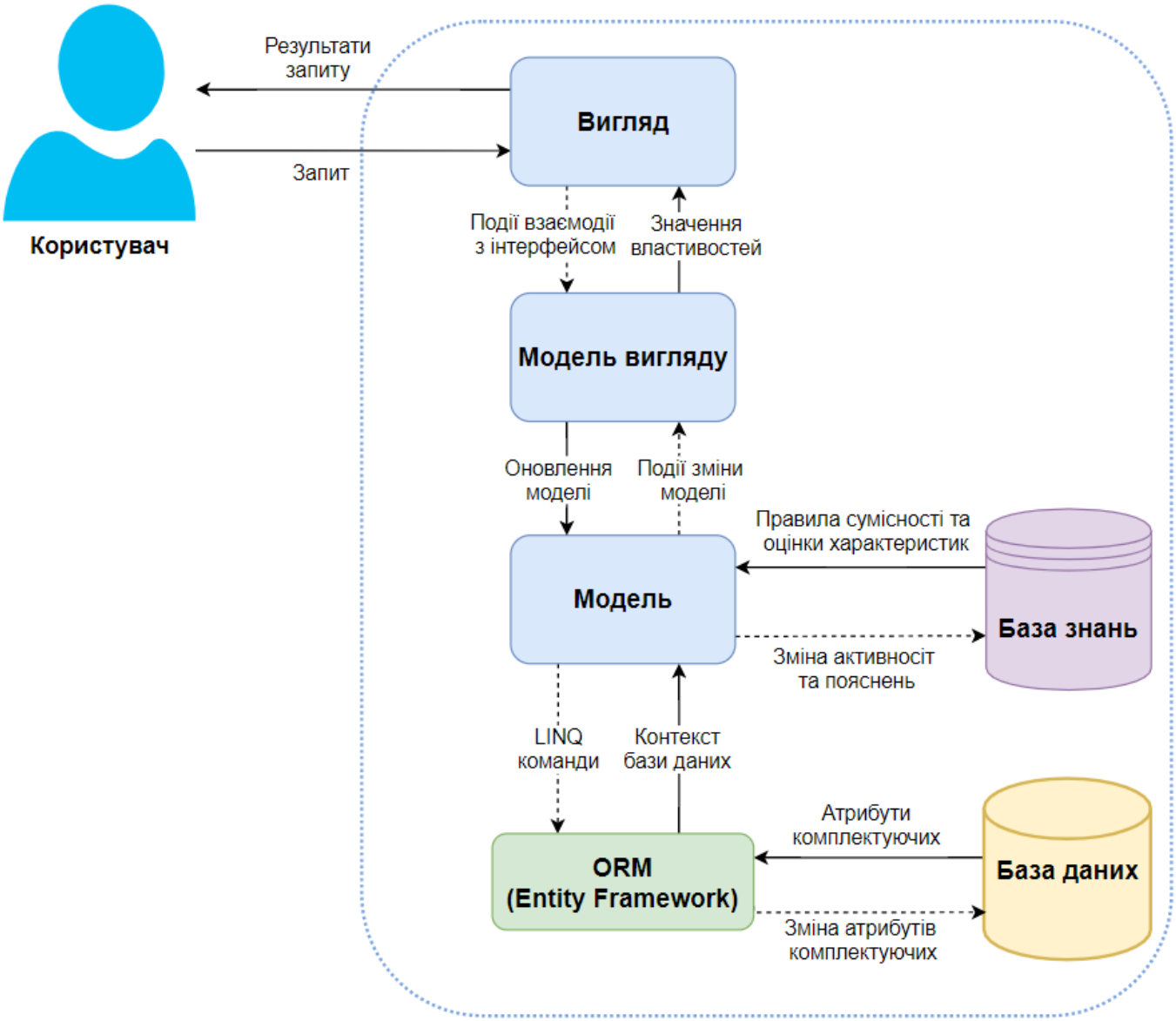


Рисунок 3.1 – Архітектура додатку

3.2 Програмна реалізація

Реалізація програмного додатку відбувалась в інтегрованому середовищі розробки Microsoft Visual Studio 2017. В якості мови програмування було обрано об’єктно-орієнтовану мову C# 7.0 (версія регламентується середовищем розробки) [15]. Для створення інтерфейсу додатку застосовувалась презентаційна підсистема

WPF [16]. Головним елементом, що надає можливість використання зазначених програмної мови й інструменту побудови елементів інтерфейсу виступає платформа .NET Framework 4.6.1. Для роботи з базою даних, як вже зазначалося, використовується ORM Entity Framework.

Visual Studio надає безмежну кількість інструментів для створення додатків, одним з таких інструментів виступає конструктор XAML (eXtensible Application Markup Language), що використовується для створення графічних елементів WPF. Цей конструктор містить дві основні області: дизайнер, в якому відображається макет інтерфейсу, та область з XAML кодом, що містить ієрархічно структурований код розширеної мови розмітки XAML. З використанням цього інструменту було розроблено інтерфейс головного вікна, вікон бази даних, бази знань і авторизації, а також користувальницькі елементи управління (*UserControl*), що були застосовані у цих вікнах. Макет головного вікна додатку у XAML конструкторі наведено на рисунку 3.2.

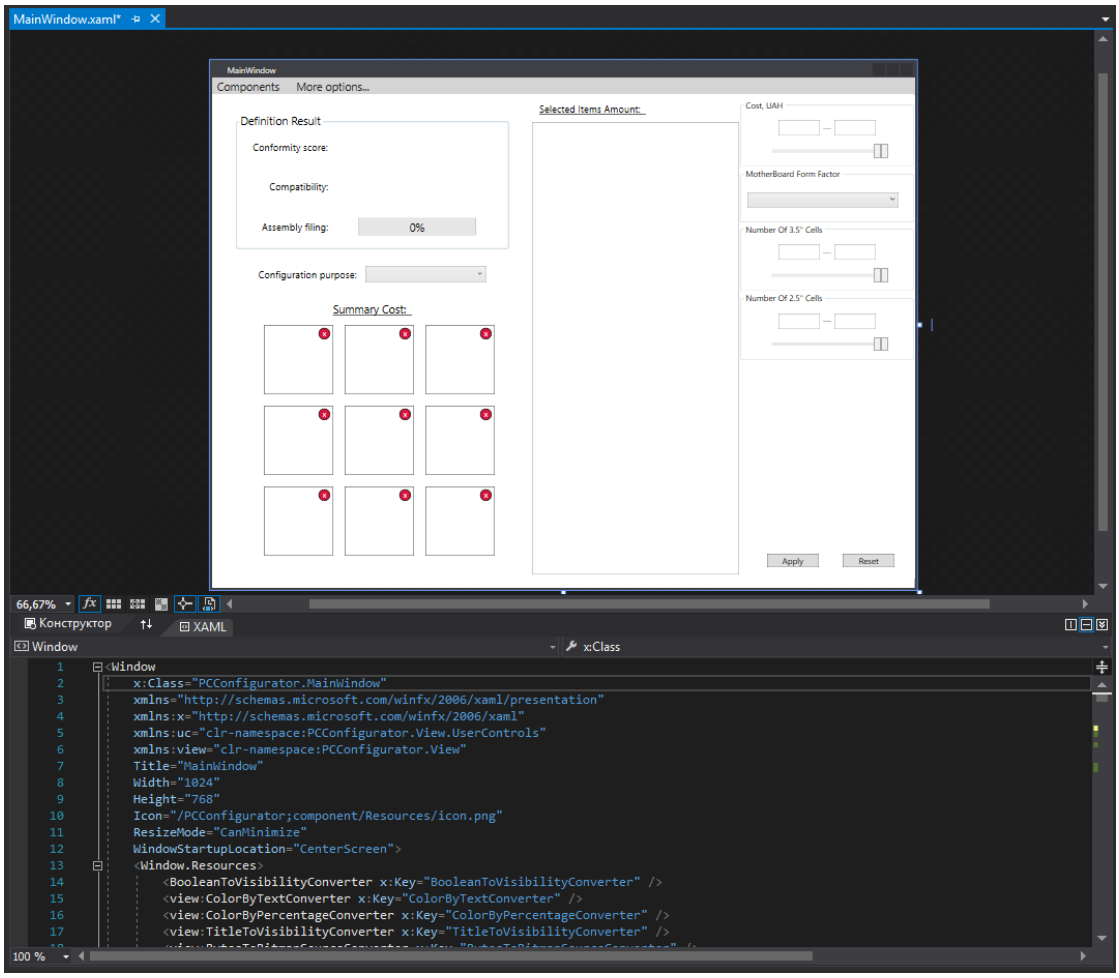
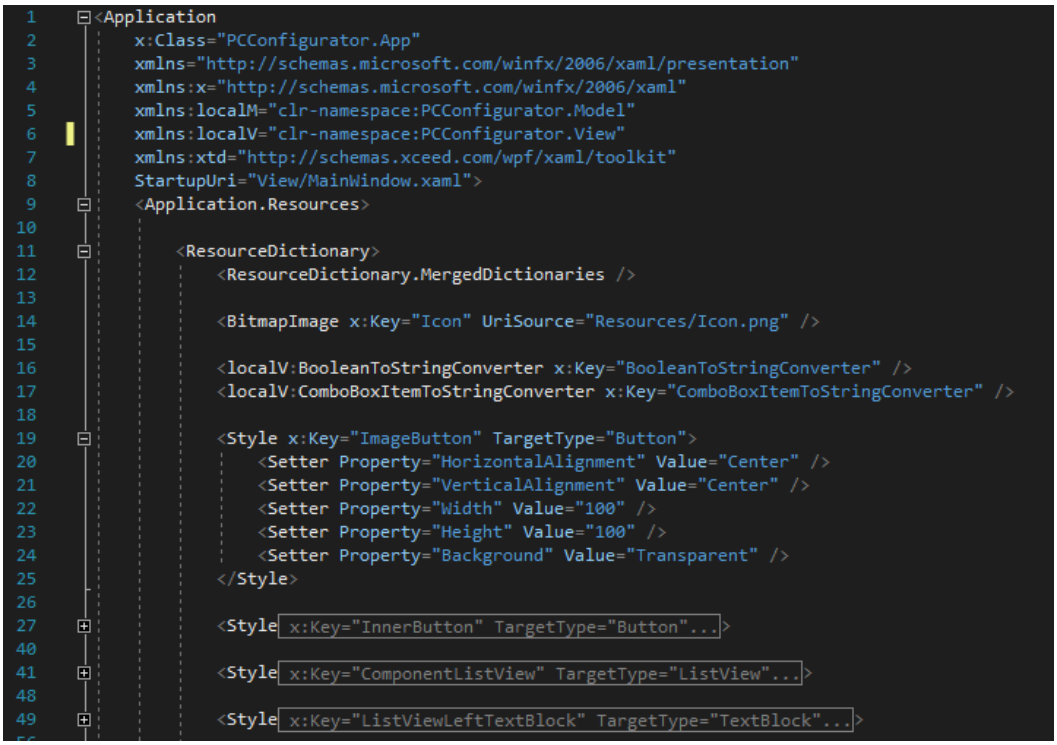


Рисунок 3.2 Макет головного вікна у конструкторі XAML

Створення інтерфейсу в більшості випадків передбачає стилізацію його компонентів. Для того, щоб зменшити кількість XAML коду при оформленні великої кількості однотипних елементів використовуються стилі. Кожен стиль повинен створюватися для певного елементу керування і може бути застосований тільки до нього. Таким чином, створивши необхідні стилі й давши їм унікальні ключові ідентифікатори (*Key*), в файлі розмітки *App.xaml*, шо є загальнодоступним для всього додатку можна застосовувати їх для стилізації відповідних елементів інтерфейсу у будь-якому іншому файлі розмітки, що використовується у цьому додатку. Отже, в розроблюваному додатку застосування стилів відбувається за механізмом зв’язування (*Binding*) певного елементу керування зі статичним ресурсом (*StaticResource*) з файлу *App.xaml*. На рисунку 3.3 наведено фрагмент файлу *App.xaml*.



```
<Application
    x:Class="PCConfigurator.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:localM="clr-namespace:PCConfigurator.Model"
    xmlns:localV="clr-namespace:PCConfigurator.View"
    xmlns:xtd="http://schemas.xceed.com/wpf/xaml/toolkit"
    StartupUri="View/MainWindow.xaml">
    <Application.Resources>

        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries />

            <BitmapImage x:Key="Icon" UriSource="Resources/Icon.png" />

            <localV:BooleanToStringConverter x:Key="BooleanToStringConverter" />
            <localV:ComboBoxItemToStringConverter x:Key="ComboBoxItemToStringConverter" />

            <Style x:Key="ImageButton" TargetType="Button">
                <Setter Property="HorizontalAlignment" Value="Center" />
                <Setter Property="VerticalAlignment" Value="Center" />
                <Setter Property="Width" Value="100" />
                <Setter Property="Height" Value="100" />
                <Setter Property="Background" Value="Transparent" />
            </Style>

            <Style x:Key="InnerButton" TargetType="Button"...>

            <Style x:Key="ComponentListView" TargetType="ListView"...>

            <Style x:Key="ListViewLeftTextBlock" TargetType="TextBlock"...>
```

Рисунок 3.3 Фрагмент файлу *App.xaml*

Реалізація шаблону MVVM передбачає розмежування бізнес-логіки та інтерфейсу, тобто моделі та вигляду, а також впровадження шару, що зв’язує вигляд з моделлю – модель вигляду. Для більш наочного розмежування було створено три директорії, що представлятимуть файли з кодом, що відносяться до вигляду, моделі та моделі вигляду. Отже, далі будуть послідовно розглянуті ці три директорії з наявними в них файлами.

Директорія вигляду (рисунок 3.4) містить такі файли XAML, класи C# та піддиректорія:

- *Authorization.xaml*: макет інтерфейсу вікна авторизації;
- *MainWindow.xaml*: макет інтерфейсу головного вікна;
- *DataBase.xaml*: макет інтерфейсу вінка бази даних;
- *KnowledgeBase.xaml*: макет інтерфейсу вінка бази знань;
- *UserContols*: директорія, що містить макети користувальницьких елементів управління, які представляють собою фільтри до характеристик кожного з типів комплектуючих;

- *BooleanToStringConverter.cs*: конвертація логічного значення до його рядкового представлення;
- *ColorByPercentageConverter.cs*: конвертація десяткового числа, в залежності від діапазону до якого воно належить, у відповідний колів;
- *ColorByTextConverter.cs*: конвертація рядка, в залежності від його значення, у відповідний колів;
- *PasswordBehaviour.cs*: визначення додаткової поведінки елемента керування *PasswordBox*, що дозволяє застосовувати механізм зв'язування до його атрибута *Password*;
- *TabItemToStringConverter.cs*: конвертація вкладки, в залежності від її заголовка, до рядкового представлення;
- *TitleToVisibilityConverter.cs*: конвертація заголовка елемента, в залежності від його значення, до типу видимості елемента керування.

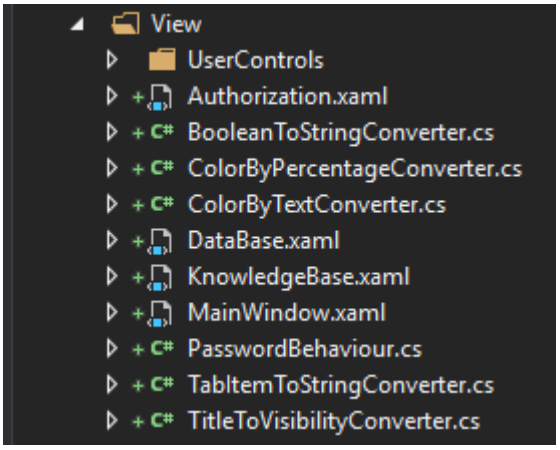


Рисунок 3.4 – Директорія *View* в оглядачі рішень Visual Studio

На рисунку 3.5 наведено лістинг одного з класів для конвертації значень з моделі представлення *ColorByPercentageConverter.cs*, цей клас, як і подібні до нього класи, описані вище, реалізують інтерфейс *IValueConverter*, який визначає 2 методи – для прямої та оберненої конвертації значень. У даному випадку немає необхідності в оберненій конвертації, отже ці класи реалізують тільки один з методів. Необхідність застосування таких класів полягає у тому, що вони дозволяють конвертувати типи

даних, що використовує модель представлення до типів, доступних тільки в моделі, згідно MVVM.

```
using System;
using System.Windows;
using System.Windows.Data;
using System.Globalization;
using System.Windows.Media;

namespace PCConfigurator.View
{
    class ColorByPercentageConverter : IValueConverter
    {
        public Object Convert(Object value, Type targetType, Object parameter, CultureInfo culture)
        {
            Decimal dValue = Decimal.Parse(value.ToString().Replace(',', '.'), CultureInfo.InvariantCulture);

            if (dValue <= 20.0m)
                return Brushes.Red;
            if (dValue > 20.0m && dValue <= 40.0m)
                return Brushes.Orange;
            if (dValue > 40.0m && dValue <= 60.0m)
                return Brushes.Goldenrod;
            if (dValue > 60.0m && dValue <= 80.0m)
                return Brushes.YellowGreen;
            else
                return Brushes.Green;
        }

        public Object ConvertBack(Object value, Type targetType, Object parameter, CultureInfo culture)
        {
            return DependencyProperty.UnsetValue;
        }
    }
}
```

Рисунок 3.5 – Клас *ColorByPercentageConverter.cs*

Директорія моделі (рисунок 3.6) містить такі класи C# та елементи абстрагування доступу до бази даних:

- *CompatibilityProvider.cs*: методи для перевірки сумісності компонентів за відповідними правилами;
- *ComponentsDataBaseEntities.cs*: розширення з репозиторіями кожного з типів комплектуючих;
- *ConvertProvider.cs*: методи для конвертації рядкових значень зчитаних з інтернет-ресурсу атрибутів комплектуючих до типів, що зберігатимуться в базі даних і відповідному форматі;
- *DataBaseProvider.cs*: методи для роботою з бази даних (збереження/скасування змін, резервування та відновлення);

- *EvaluationProvider.cs*: методи для обрахунку оцінок характеристик компонентів сформованої збірки;
- *FileProvider.cs*: методи для збереження та завантаження збірки з файлу;
- *HTMLParser.cs*: зчитування атрибутів комплектуючих з інтернет-ресурсу;
- *ImageProvider.cs*: методи для завантаження зображення з джерел різних типів і його обробки;
- *Rule.cs*: розширення, що містить делегат (посилання на метод) за яким відбувається перевірка правила;
- *User.cs*: представлення користувача-адміністратора системи;
- *ComponentsDataModel.edmx*: модель бази даних, створена з використанням EF, містить усі сутності, та надає абстрагований доступ до бази даних;
- *ComponentsDataModel.edmx.sql*: сценарій створення бази даних за побудованою моделлю.

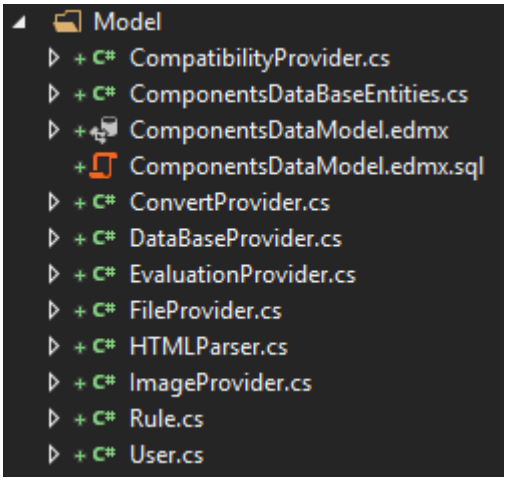


Рисунок 3.6 – Директорія *Model* в оглядачі рішень Visual Studio

Зчитування інформації атрибутів кожного з типів комплектуючих з інтернет-ресурсу (hard.rozetka.com.ua) [17] відбувається за допомогою статичного класу-утиліти *HTMLParser* за таким алгоритмом:

1. Завантаження головної (першої) сторінки з комплектуючими обраного типу і встановленими для них фільтрів.

2. Визначення загальної кількості сторінок з комплектуючими відповідного типу.
3. Прохід по кожній з цих сторінок, який включає:
    a. Завантаження сторінки окремої комплектуючої.
    b. Зчитування її атрибутів.
    c. Конвертація цих атрибутів до відповідних типів, що зберігатимуться в базі даних.
    d. Додавання комплектуючої до вихідного списку комплектуючих відповідного типу.

Для зчитування посилань на кожну з комплектуючих використовується об’єкт класу *WebDriver* з бібліотеки *Selenium*, яка використовується для автоматизації роботи з браузером. Зчитування атрибутів окремих комплектуючих відбувається за допомогою більш простої бібліотеки *HtmlAgilityPack*, що призначена для розпізнавання атрибутів HTML-документів. На рисунку 3.7 наведено фрагмент методу зчитування інформації з інтернет-ресурсу *GetComponentsData(String componentName),* в якому відбувається зчитування атрибутів окремої комплектуючу.

```
// Проход по считаным ссылкам.
foreach (String href in hrefValues)
{
    // Загрузка страницы по ссылке.
    htmlDocument = htmlWeb.Load(href);
    innerList = Enumerable.Repeat("None", titles[componentName].Count + 2).ToList();
    // Считывание названия комплектующей.
    innerList[0] = htmlDocument.DocumentNode.SelectSingleNode("//product-top/div/div[1]/h1").InnerText;
    // Считывание названия комплектующей.
    innerList[innerList.Count - 1] =
        htmlDocument.DocumentNode.SelectSingleNode("//product-main-info/div/div[1]/div/p[1]/text()")?.InnerText ?? "0";
    // Считывание ссылки на загрузку изображения.
    imageUri =
        htmlDocument.DocumentNode.SelectSingleNode("//product-gallery-main/div/div/ul/div/div/li[1]/a/img")
        .Attributes["src"].Value;
    // Переход на страницу характеристик комплектующей.
    htmlDocument = htmlWeb.Load(href + INNERPARAM);
    // Считывание всех характеристик комплектующей.
    htmlValueNodes = htmlDocument.DocumentNode.SelectNodes("//dl/dd/ul");
    // Фильтрация и считывание нужных характеристик.
    for (Int32 j = 0; j < htmlValueNodes.Count; j++)
    {
        featureTitle = htmlValueNodes[j].ParentNode.PreviousSibling.FirstChild.InnerText;
        if (titles[componentName].Contains(featureTitle))
        {
            if (htmlValueNodes[j].ChildNodes == null)
                innerList[titles[componentName].IndexOf(featureTitle) + 1] = htmlValueNodes[j].InnerText;
            else
            {
                String tempStr = "";
                foreach (HtmlNode node in htmlValueNodes[j].ChildNodes)
                    tempStr += node.InnerText + " ";
                innerList[titles[componentName].IndexOf(featureTitle) + 1] = tempStr.TrimStart(' ').TrimEnd(' ');
            }
        }
    }
    // Конвертация и добавление комплектующей к списку.
    outData.Add(ConvertProvider.ConvertTo(componentName, innerList, new Uri(imageUri)));
    innerList.Clear();
}
```

Рисунок 3.7 – Зчитування атрибутів комплектуючої у методі *GetComponentsData(String componentName)*

Як було зазначено, перед тим як додавати компонент з його атрибутами до вихідного списку, зчитана інформація відповідним чином конвертується до необхідних типів даних або певної форми рядкового представлення. Ці операції виконуються статичним класом-утилітою *ConvertProvider*. Він містить ряд методів, що окремо конвертують атрибути комплектуючих певного типу, методи, що конвертують спільні атрибути, та один загальний метод, що в залежності від обраного типу повертає конвертований об’єкт комплектуючої відповідного типу. Фрагмент класу конвертера наведено на рисунку 3.8.

```
internal static class ConvertProvider
{
    // Выбор и переход к конвертации определённого типа копмлекутующих.
    public static dynamic ConvertTo(String componentName, List<String> inList, Uri uri)...

    // Конвертация цены.
    private static Int32 GetCost(String cost)...

    // Конвертация максимального объёма оперативной памяти материнской платы.
    private static Int16 GetMaxRAMSize(String str)...

    // Конвертация размеров материнской платы.
    private static String GetMBSizes(String str)...

    // Конвертация харакетристик матринской платы.
    private static MotherBoard ConvertToMotherBoard(List<String> inList, Uri uri)
    {
        return new MotherBoard
        {
            Title = inList[0].Substring(" Материнская плата".Length).Trim(),
            ConnectorType = inList[1],
            ChipSet = inList[2],
            FormFactor = inList[3].Contains("MiniITX") ? "Mini-ITX" : inList[3],
            MemSupported = inList[4],
            CPUsSupported = inList[5],
            MaxRAMSize = inList[6] != "None" ? GetMaxRAMSize(inList[6]) : (Int16)0,
            GPUInterface = inList[7],
            PowerConnector = inList[8],
            Sizes = inList[9] != "None" ? GetMBSizes(inList[9]) : "None",
            Cost = GetCost(inList[10]),
            Image = ImageProvider.GetNewImageByUri(uri)
        };
    }

    // Конвертация характеристик процессора.
    private static CPU ConvertToCPU(List<String> inList, Uri uri)...

    // Конвертация частоты памяти видеокарты.
    private static Int32 GetMemFrequency(String str)...
```

Рисунок 3.8 – Фрагмент класу *ConvertProvider*

Перевірка виконання правил сумісності відбувається за допомогою статичного класу-утиліти *CompatibilityProvider*, який містить набір методів, по одному для кожного з правил сумісності. Методи цього класу приймають два параметри, що відповідають компонентам збірки між якими перевіряється сумісність і повертають логічне значення, яке і визначає сумісність компонентів. Ці методи зберігаються у масиві делегатів типу *Func<Component, Component, Boolean>*, які при завантаженні додатку приписуються одному з методів відповідного правила сумісності. На рисунку 3.9 наведено фрагмент класу *CompatibilityProvider*.

```
internal static class CompatibilityProvider
{
    // Все правила совместимости (массив делегатов).
    public static Func<Component, Component, Boolean>[] Rules =
    {
        IsCPUandMB, IsGPUandMB, IsRAMandMB, IsBodyandMB,
        IsGPUandPowerSupply, IsGPUandBody, IsCoolerandCPU, IsCoolerandBody
    };

    // Проверка совместимости процессора и материнской платы.
    private static Boolean IsCPUandMB(this Component motherBoard, Component cpu)
    {
        return ((MotherBoard)motherBoard).ConnectorType.Contains(((CPU)cpu).ConnectorType);
    }

    // Проверка совместимости видеокарты и материнской платы.
    private static Boolean IsGPUandMB(this Component motherBoard, Component gpu)
    {
        Boolean isCompatible = false;

        String[] interfaces = Regex.Split(((MotherBoard)motherBoard).GPUInterface, @"\d\s[xx]\s");
        GroupCollection groupMB;
        GroupCollection groupPS =
            Regex.Match(((GPU)gpu).Interface, @"PCI-Express\s?(?:[xx](\d+)\s?)?(\d+).?\w?").Groups;

        for (Int32 i = 1; i < interfaces.Length; i++)
        {
            groupMB = Regex.Match(interfaces[i], @"PCI-E\s?(\d+).?\w?\s?[xx](\d+)").Groups;

            if (groupMB[1].Value == groupPS[2].Value && groupMB[2].Value == groupPS[1].Value)
            {
                isCompatible = true;
                break;
            }

        }

        return isCompatible;
    }

    // Проверка совместимости оперативной памяти и материнской платы.
    private static Boolean IsRAMandMB(this Component motherBoard, Component ram)...
```

Рисунок 3.9 – Фрагмент класу *CompatibilityProvider*

Визначення оцінок характеристик і обчислення відсотка відповідності для компонентів сформованої збірки відбувається у схожому статичному класі-утиліті *EvaluationProvider*. Цей клас містить методи оцінки характеристики певної комплектуючої відповідно до типу збірки за потребами користувача. Вони приймають в якості параметра значення характеристики компонента збірки (тип визначається під час виконання програми) і повертають логічне значення, що визначає, чи задовольняє характеристика компонента, що оцінюється, значенням заданим у правилі. Методи цього класу зберігаються у масиві делегатів типу *Func<dynamic, Boolean>*, які, аналогічно приписуються відповідним правилам оцінки характеристик. Оцінка відповідності розраховується за допомогою функції *Evaluate(Int32 marksPattern)*, яка передбачає обчислення відсоткового відношення

середнього арифметичного оцінок характеристик сформованої збірки до значення оцінки, що відповідає одному з типів збірки. Фрагмент класу *EvaluationProvider* наведено на рисунку 3.10.

```
internal static class EvaluationProvider
{
    // Все правила оценки соответствия (массив делегатов).
    public static Func<dynamic, Boolean>[] Rules;

    // Оценки компонентов сформиврованной сборки.
    private static Int32[] currentMarks;

    // Среднее арифметичесоке оценок сформированной сборки.
    private static Decimal GetAvarage(Int32 marksPattern)
    {
        Int32 sum = 0;
        foreach (Int32 val in currentMarks)
            sum += val > marksPattern ? marksPattern - (val - marksPattern) : sum += val;

        return (Decimal)sum / currentMarks.Length;
    }

    // Оценка соответствия сборки установленному шаблону.
    public static Decimal Evaluate(Int32 marksPattern)
    {
        return GetAvarage(marksPattern) / marksPattern * 100.0m;
    }

    // Проверка требований к значению формфактора материнской платы для типа Gaming.
    private static Boolean GMBFormFactor(String formfactor)
    {
        return formfactor == "ATX" || formfactor == "MicroATX";
    }

    // Проверка требований к значению типа конектора материнской платы для типа Gaming.
    private static Boolean GMBSocket(String socket)
    {
        return socket == "Socket 1151" || socket == "Socket 2066" || socket == "Socket AM4";
    }

    // Проверка требований к значению количества ядер процессора для типа Gaming.
    private static Boolean GCPUNumOfCores(Int32 numOfCores)...

    // Проверка требований к значению разблокированного множителя процессора для типа Gaming.
    private static Boolean GCPUUnlocked(Boolean isUnlocked)...
```

Рисунок 3.10 – Фрагмент класу *EvaluationProvider*

Повний лістинг основних класів моделі, наведено у додатку Г.

Директорія моделі (рисунок 3.11) містить такі класи C# та піддиректорію:

- *UnderBaseVM.cs*: модель представлення, що реалізує можливість відслідковування зміни властивостей;
- *BaseVM.cs*: базова модель представлення, що містить загальні властивості для моделей представлення головного вікна, вінка бази даних і бази знань;
- *AuthorizationVM.cs*: модель представлення вінка авторизації;

- *MainWindowVM.cs*: модель представлення головного вікна додатку;
- *DataBaseVM.cs*: модель представлення вікна бази даних;
- *KnowledgeBaseVM.cs*: модель представлення вікна бази знань;
- *RelayCommand.cs*: реалізація механізму (шаблону) команда;
- *FilteringVMs*: директорія, що містить моделі представлення для кожного з користувальницьких елементів управляння, що представляють собою фільтри до характеристик комплектуючих.

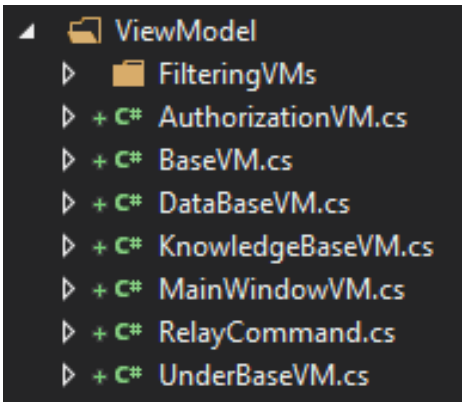


Рисунок 3.11 – Директорія *ViewModel* в оглядачі рішень Visual Studio

Для реалізації механізму зв’язування з властивостями класів моделі представлення, ці класи повинні реалізовувати інтерфейс *INotifyPropertyChanged*. Для того, щоб позбавитись необхідності реалізації цього інтерфейсу у кожному з класів моделі представлення було створено клас абстрактний клас *UnderBaseVM*, який саме і реалізує зазначений інтерфейс. Також, цей клас реалізує інтерфейс *IDataErrorInfo*, який надає механізми для перевірки значень (валідації) властивостей, що використовуються у моделі представлення і зв’язуються з елементами керування. На рисунку 3.12 наведено лістинг класу *UnderBaseVM*.

```
internal abstract class UnderBaseVM : INotifyPropertyChanged, IDataErrorInfo
{
    #region INotifyPropertyChanged members

    public event PropertyChangedEventHandler PropertyChanged;

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

    #endregion

    #region IDataErrorInfo members

    public virtual String Error
    {
        get { throw new NotImplementedException(); }
    }

    public virtual String this[String propertyName]
    {
        get { return String.Empty; }
    }

    #endregion
}
```

Рисунок 3.12 – Клас *UnderBaseVM*

Моделі представлення головного вікна, вікна бази даних і бази знань містять достатньо велику кількість спільних властивостей, і для того, щоб запобігти дублюванню коду у зазначених класах було створено ще один абстрактний клас – *BaseVM*, який успадковується від описаного вище класу *UnderBaseVM*, і тому, відповідно, успадковує реалізацію необхідних інтерфейсів. Цей клас містить властивості для зберігання множин об’єктів комплектуючих певного типу, правил сумісності та оцінки відповідності, а також зберігає моделі представлення для фільтрів, що застосовуються у головному вікна та вікні бази даних. Зазначені раніше класи моделей представлення кожного з вікон успадковуються від даного класу. Фрагмент класу *BaseVM* наведено на рисунку 3.13.

```
internal abstract class BaseVM : UnderBaseVM
{
    Components collections

    Filters values

    #region Selected item
    // Выбранный из списка/таблицы компонент.
    private Model.Component selectedItem;

    public Model.Component SelectedItem
    {
        get { return selectedItem; }
        set
        {
            if (selectedItem != value)
            {
                selectedItem = value;
                OnPropertyChanged(nameof(SelectedItem));
            }
        }
    }

    #endregion

    #region Rules
    // Правила совместимости комплектующих.
    private BindingList<CompatibilityRule> compatibilityRules;

    public BindingList<CompatibilityRule> CompatibilityRules
    {
        get { return compatibilityRules ?? new BindingList<CompatibilityRule>(); }
        set
        {
            compatibilityRules = value;
            OnPropertyChanged(nameof(CompatibilityRules));
        }
    }

    // Правила оценки характеристик.
    private BindingList<EvaluativeRule> evaluativeRules;
```

Рисунок 3.13 – Фрагмент класу *BaseVM*

Повний лістинг основних класів моделі представлення, наведено у додатку Г.

3.3 Використання програмного додатку

Встановивши додаток PCConfigurator на свій комп’ютер, що задовольняє встановленим у технічному завдані вимогам, та завантаживши його, на робочому столі комп’ютера користувача з’являється вікно – головне вікно додатку, що у початковому стані має наступний вигляд (рисунок 3.14):

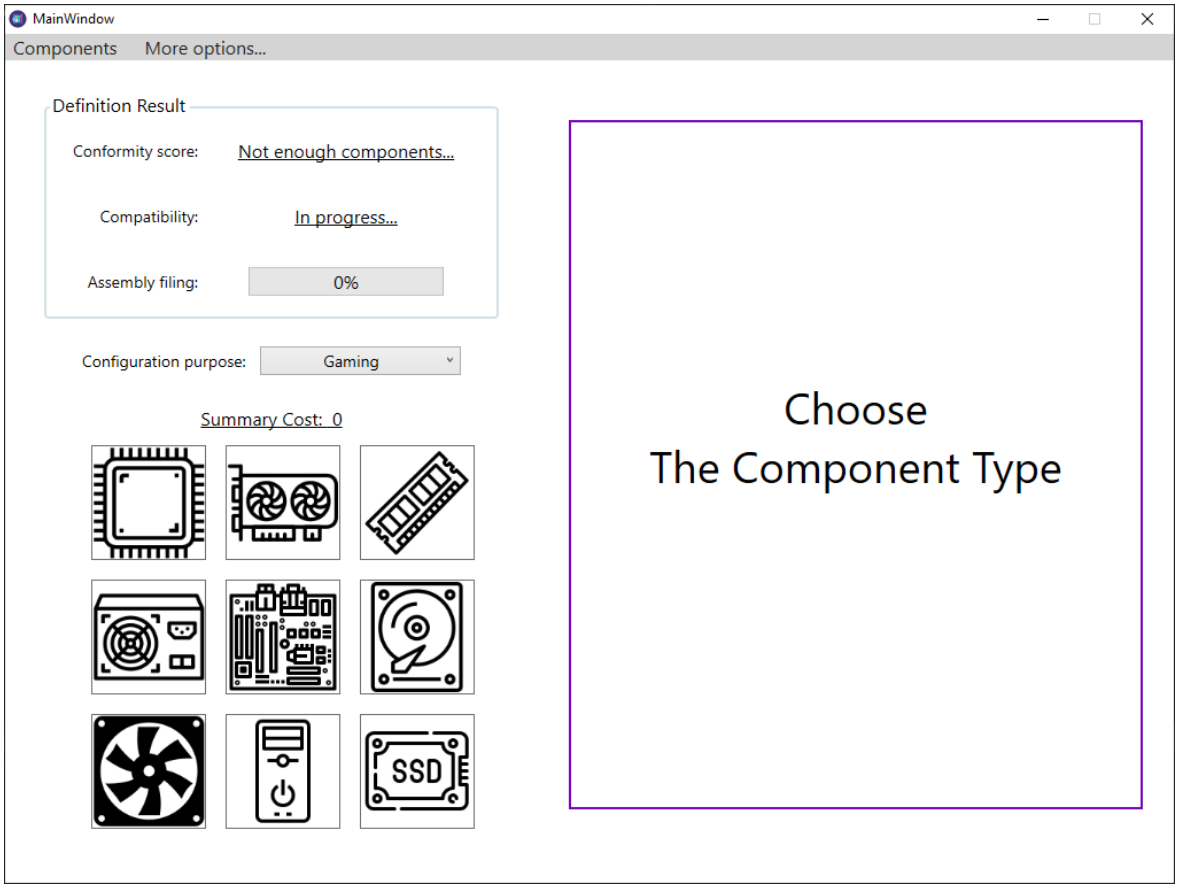


Рисунок 3.14 – Початковий вигляд головного вікна

Для того, що почати формування збірки, користувач натискає на одне з абстрактних зображень комплектуючих, що визначають тип компонента збірки. Так, натиснувши на зображення центрального процесора (верхнє ліве зображення) у правій області вікна з’являється список з компонентами даного типу, а також фільтри для ряду характеристик центрального процесора (рисунок 3.15).

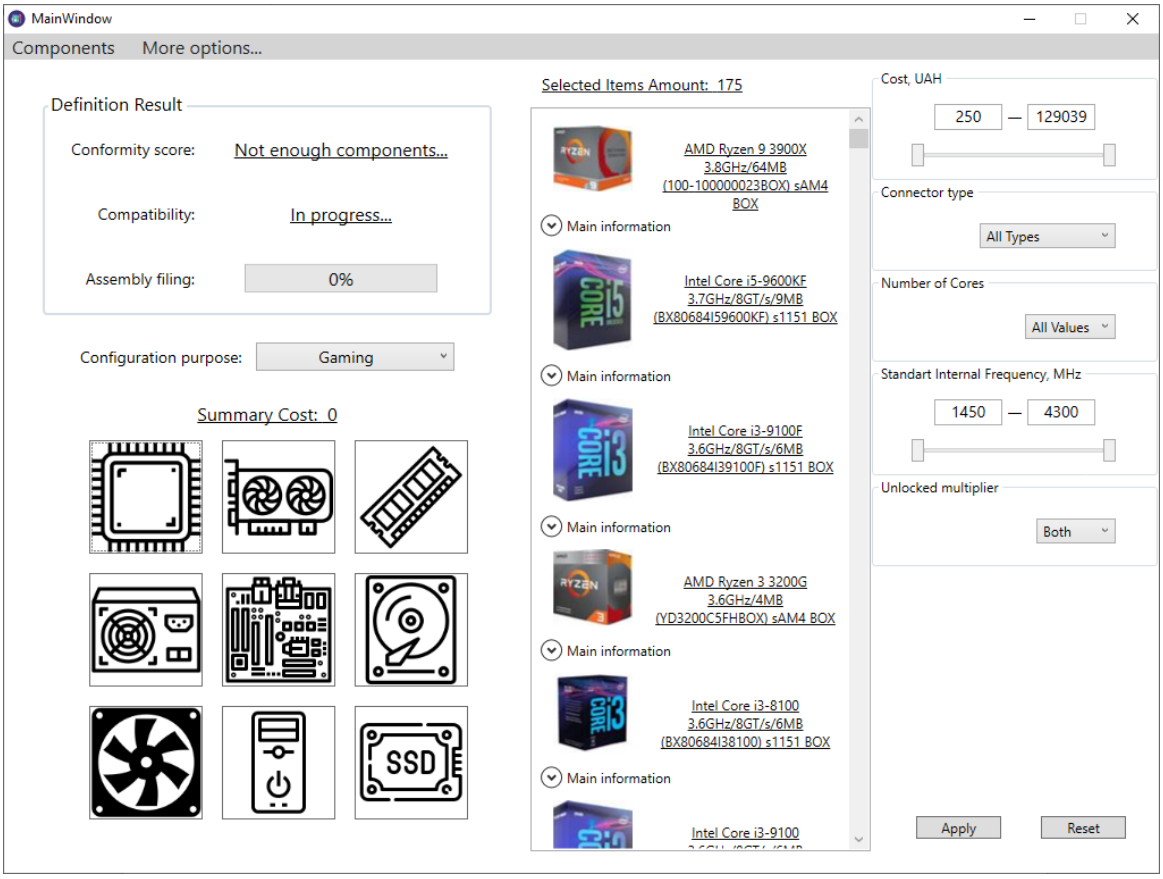


Рисунок 3.15 – Вигляд головного вікна після вибору типу компонента збірки

Встановивши значення фільтрів відповідно до своїх потреб, і натиснувши на кнопку «Apply», список комплектуючих обраного типу звужується, і у відповідній області відображаються тільки ті компоненти, що задовольняють встановленим обмеженням (рисунок 3.16).

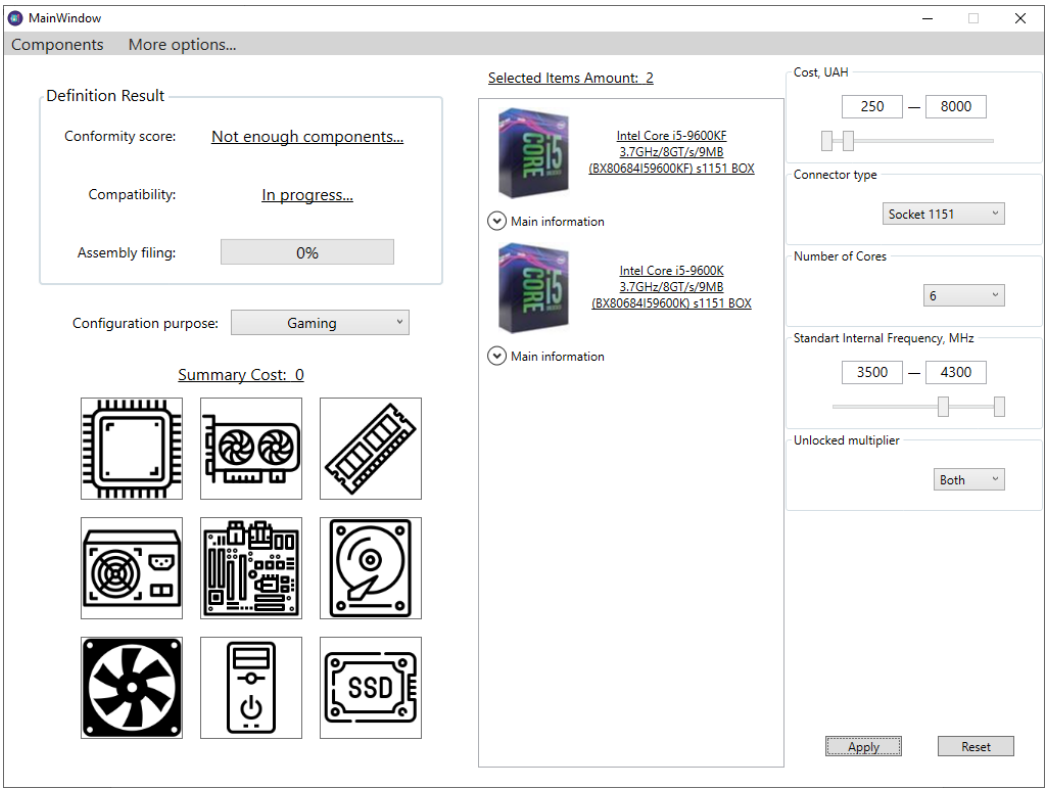


Рисунок 3.16 – Вигляд головного вікна в результаті фільтрування записів

Користувач має можливість переглянути технічні характеристики комплектуючої, розкривши випадаючий список під її зображенням натисканням лівої кнопки миші в області напису «Main information» (рисунок 3.17).

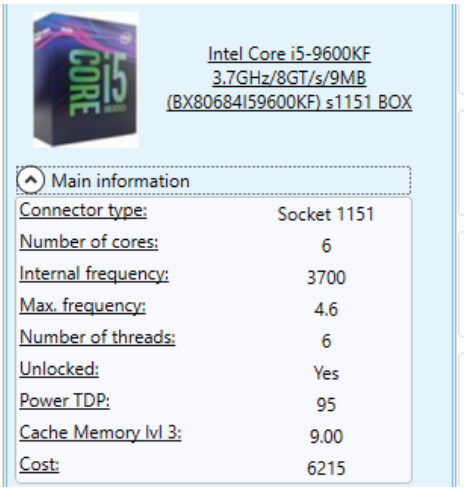


Рисунок 3.17 – Випадаючий список з технічними характеристиками комплектуючої

Обравши комплектуючу зі списку, користувач може додати її до збірки, зробивши подвійне натискання лівої кнопки миші в області списку з відповідною комплектуючою. Таким чином, обрана комплектуюча додається до збірки і абстрактне зображення компонента замінюється відповідним зображенням зі списку, наприклад, користувач додав до збірки материнську плату і центральний процесор (рисунок 3.18). Також, на цьому зображенні з’являється червоний круг, натискання якого приводить до видалення компонента зі збірки, наприклад, було видалено материнську плату (рисунок 3.19).

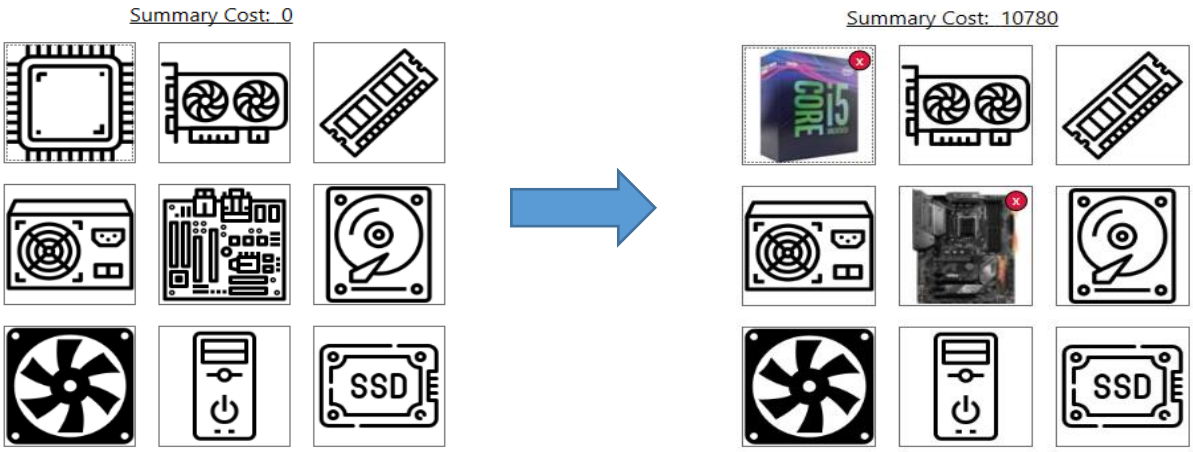


Рисунок 3.18 – Додавання комплектуючих до збірки

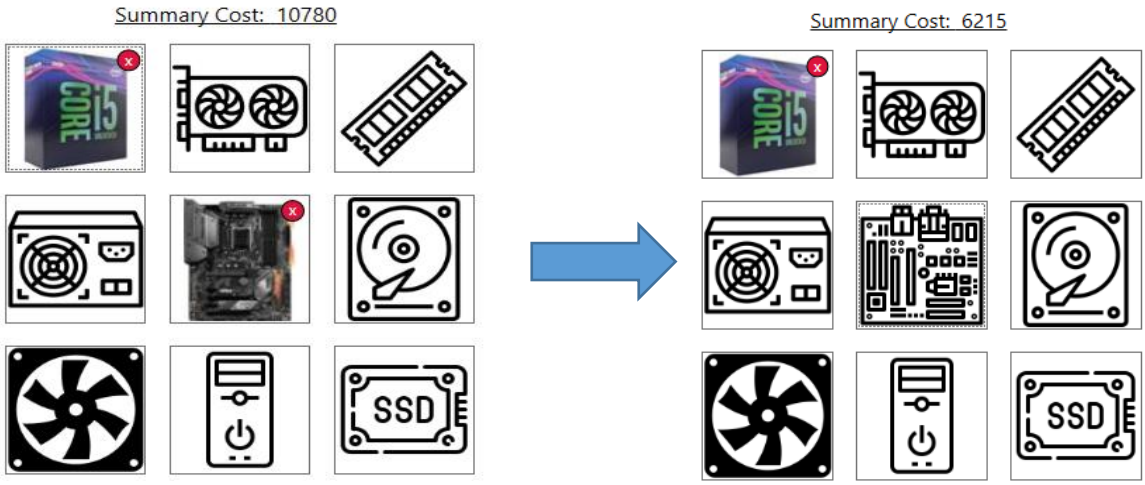


Рисунок 3.18 – Видалення комплектуючої зі збірки

Під час формування збірки відбувається динамічна перевірка сумісності доданих до неї компонентів, а також заповнення (спустошення) шкали прогресу формування збірки. Наприклад, для 3-х доданих компонентів: материнської плати, графічної карти та блока живлення, де показник необхідної потужності блока живлення для графічної карти більший за показник живлення самого блока живлення, отримуємо наступні значення (рисунок 3.20):

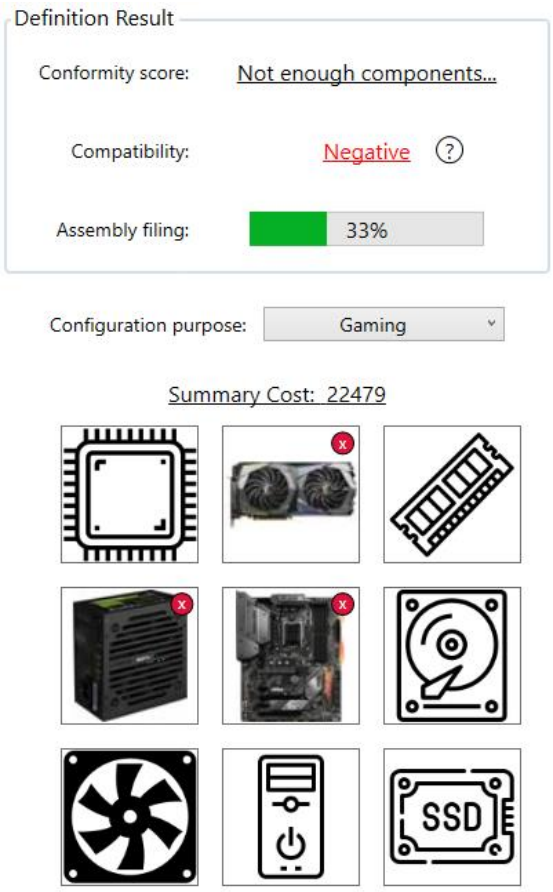


Рисунок 3.20 – Результати перевірки сумісності (негативний) та сформованості збірки

Встановивши курсор миші в область зображення зі знаком питання, що знаходиться біля негативного результату перевірки сумісності, користувач отримує відповідне пояснення: чому компоненти збірки несумісні (рисунок 3.21).

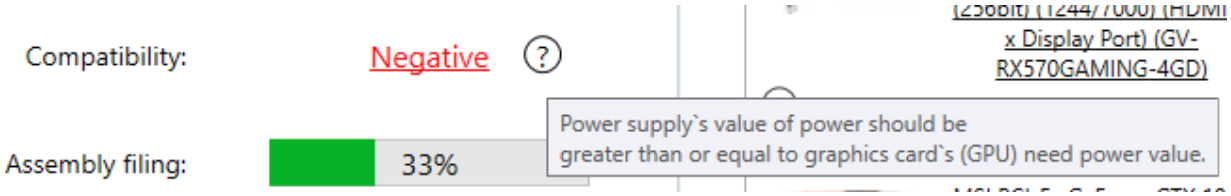


Рисунок 3.21 – Пояснення до результату перевірки сумісності

Тепер, замінивши, наприклад, блок живлення на більш потужний, отримаємо інше значення сумісності компонентів збірки, а також, додавши до збірки жорсткий диск матимемо інше значення відсотка сформованості збірки (рисунок 3.22):

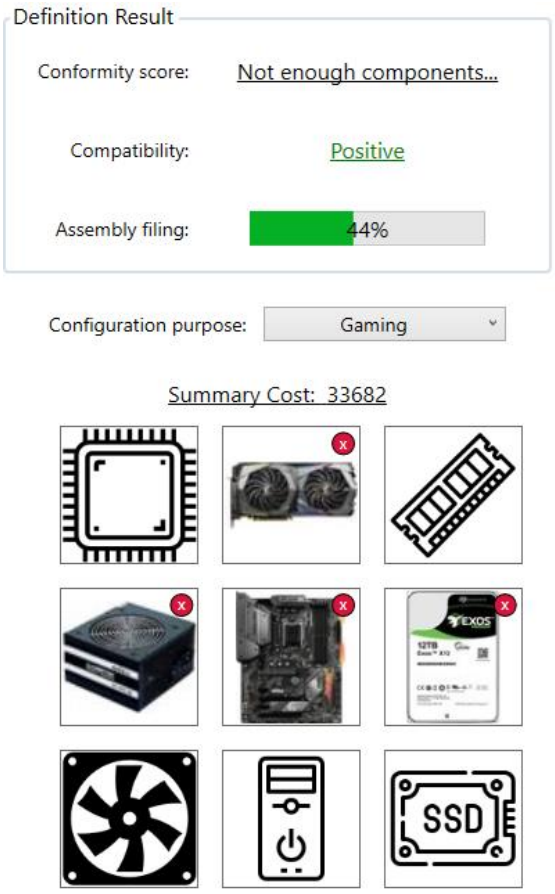


Рисунок 3.22 – Результати перевірки сумісності (позитивний) та сформованості збірки

Сформувавши збірку з набору необхідних для оцінки відповідності компонентів (можливо без компонентів: твердотільний накопичувач та/або кулер), а також встановивши режим підбору з випадаючого списку «Configuration purpose», що

більше відповідає потребам користувача, відбувається динамічний розрахунок оцінки відповідності (у відсотках) сформованої збірки до обраних користувачем потреб. Наприклад, для значення «Gaming» з випадаючого списку «Configuration purpose» та цілком сформованої збірки із сумісних компонентів користувач отримуємо таке значення відсотка відповідності (рисунок 3.23):

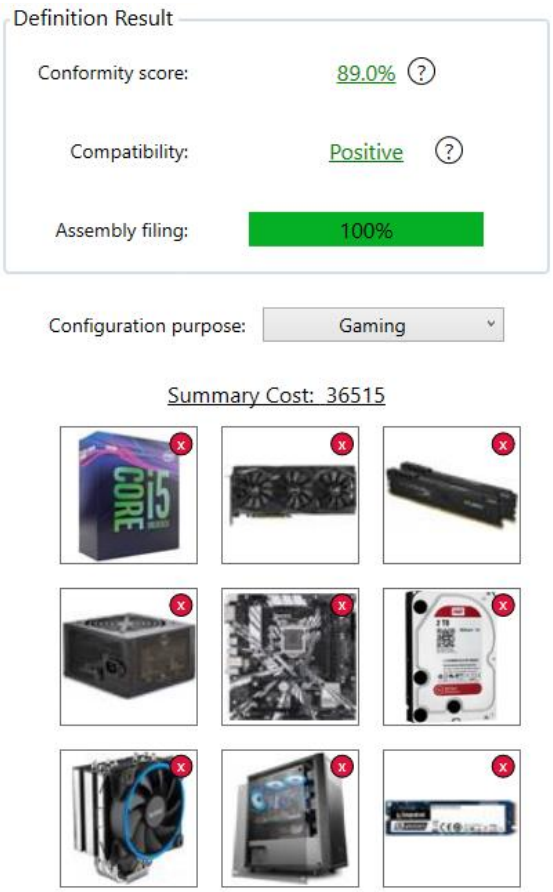


Рисунок 3.23 – Результат оцінки відповідності сформованої збірки для значення потреб користувача «Gaming»

Встановивши курсор миші на зображення зі знаком питання, що знаходиться біля результату оцінювання відповідності, окрім значення повної відповідності (100%), користувач отримує відповідне пояснення: чому значення відсотка відповідності саме таке (рисунок 3.24).

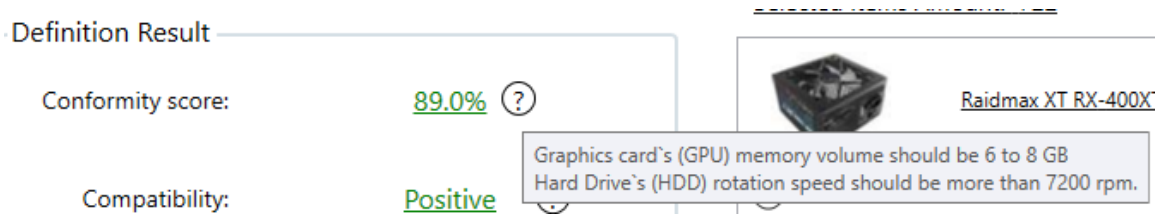


Рисунок 3.24 – Пояснення до отриманої оцінки відповідності

Тепер, залишивши сформовану збірку без змін, змінюємо значення потреб користувача зі списку «Configuration purpose» на «Multimedia/Home», і отримуємо такий результат оцінювання відповідності (рисунок 3.25):

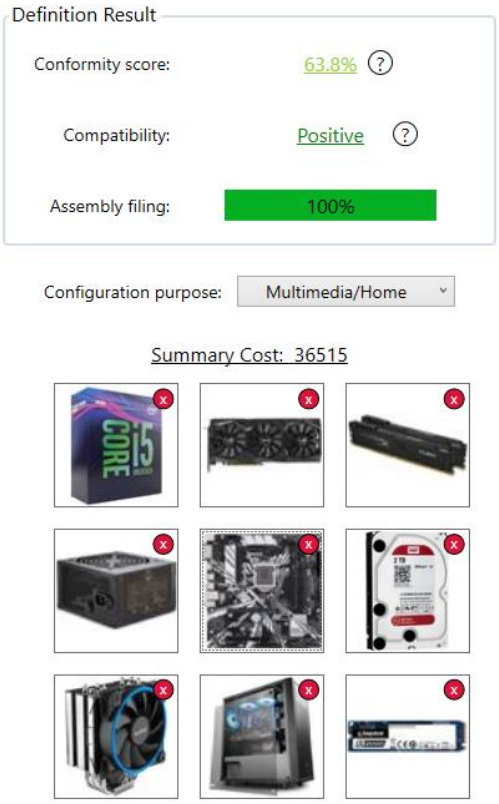


Рисунок 3.25 – Результат оцінки відповідності сформованої збірки для значення потреб користувача «Multimedia/Home»

За необхідності, користувач може зберегти поточну або завантажити наявну конфігурацію сформованої збірки. Операції збереження та завантаження доступні з верхнього горизонтального меню «Components» (рисунок 3.26).

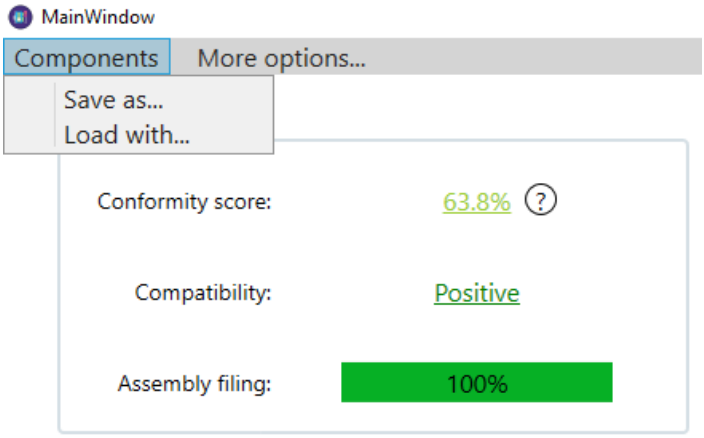


Рисунок 3.26 – Меню збереження та завантаження збірки

Після натискання пункту меню «Save as..» з’являється діалогове вікно для збереження файлу у визначену користувачем директорію (рисунок 3.27). Збірка зберігається у файлі формату .pcconf (власний формат).

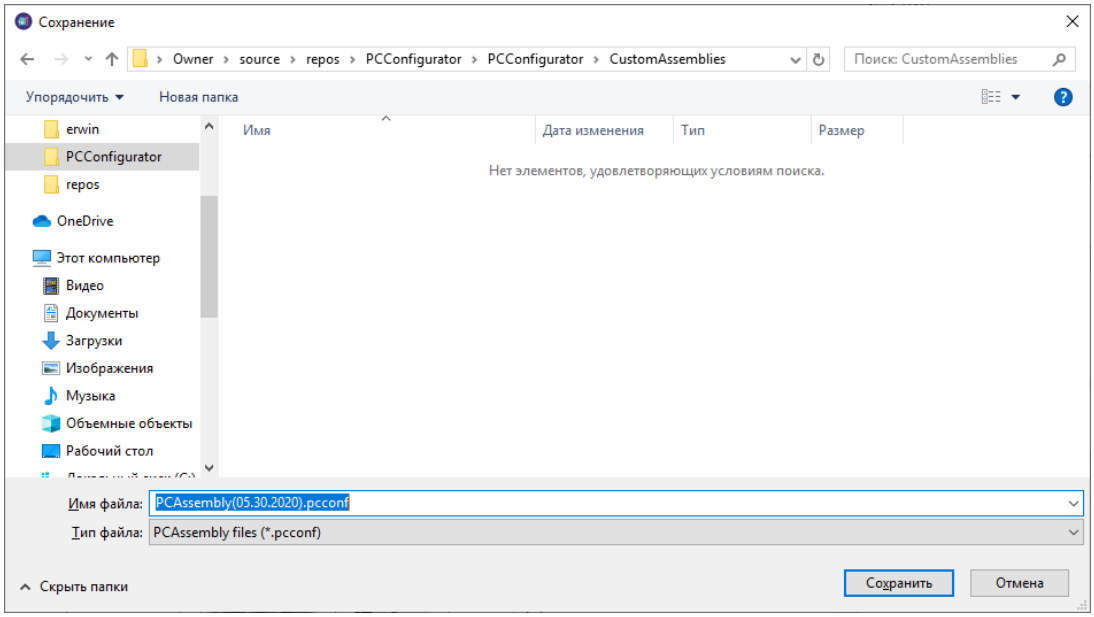


Рисунок 3.27 – Діалогове вікно збереження файлу

Натискання на пункт «Load with…» призводить до появи діалогового вікна для завантаження/відкриття файлу формату .pcconf (рисунок 3.28).

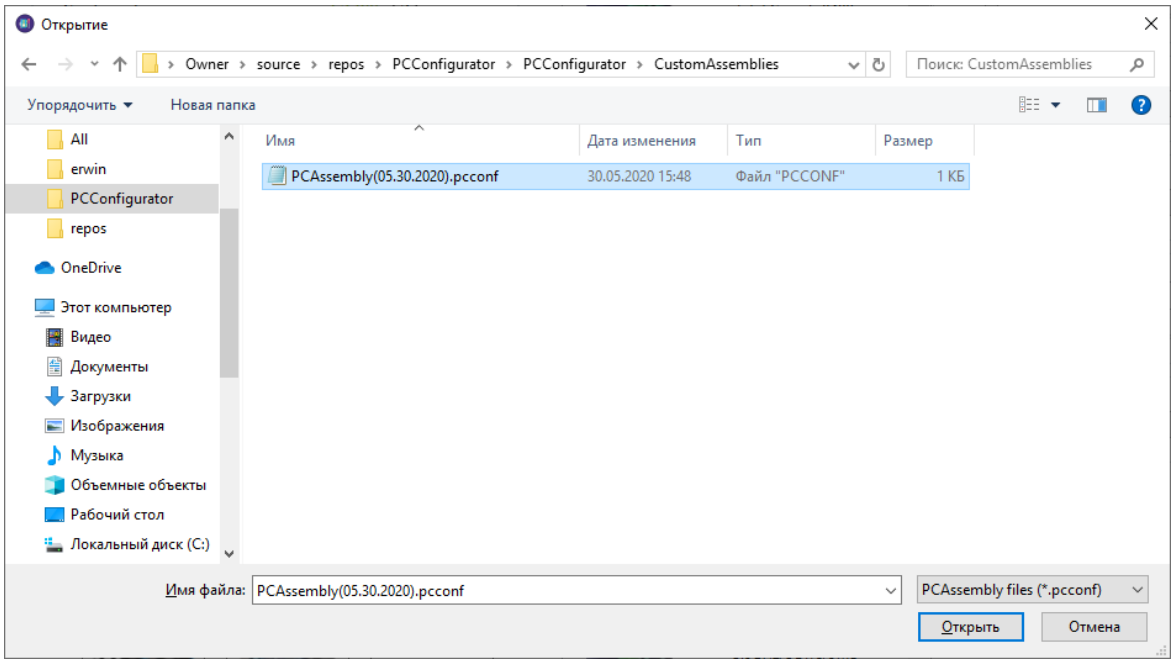


Рисунок 3.28 – Діалогове вікно відкриття файлу

Збірка, що завантажується повністю оновлює вміст поточної збірки, і відповідно визначаються нові значення сумісності та, якщо збірка сформована достатньо, оцінки відповідності (рисунок 3.29).

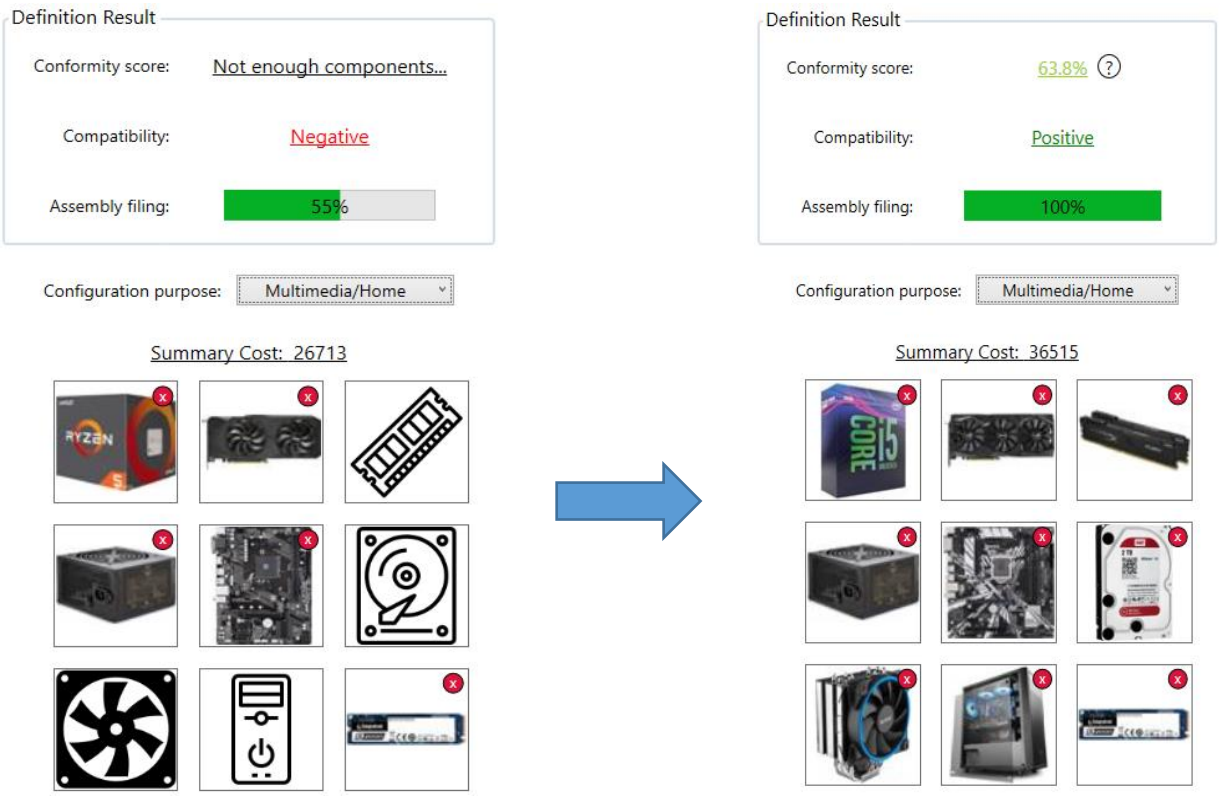


Рисунок 3.29 – Завантаження збірки з файлу

Горизонтальне меню головного вікна, також, передбачає можливість переходу (відкриття) вікон для перегляду/роботи з базою даних або базою знань (рисунок 3.30).

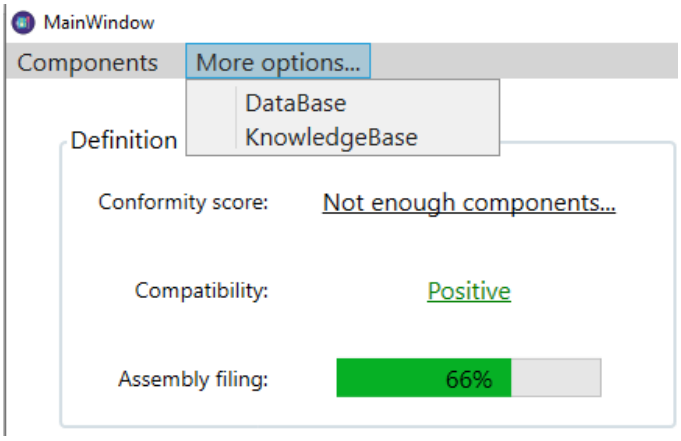


Рисунок 3.30 – Меню переходу до вікна бази даних або бази знань

Після натискання на один з цих пунктів меню з’являється вікно авторизації користувача, тобто перевірка того, чи має поточний користувач доступ до бази даних

і бази знань. Це вікно передбачає введення логіна і пароля, які відомі тільки адміністратору системи (рисунок 3.31).

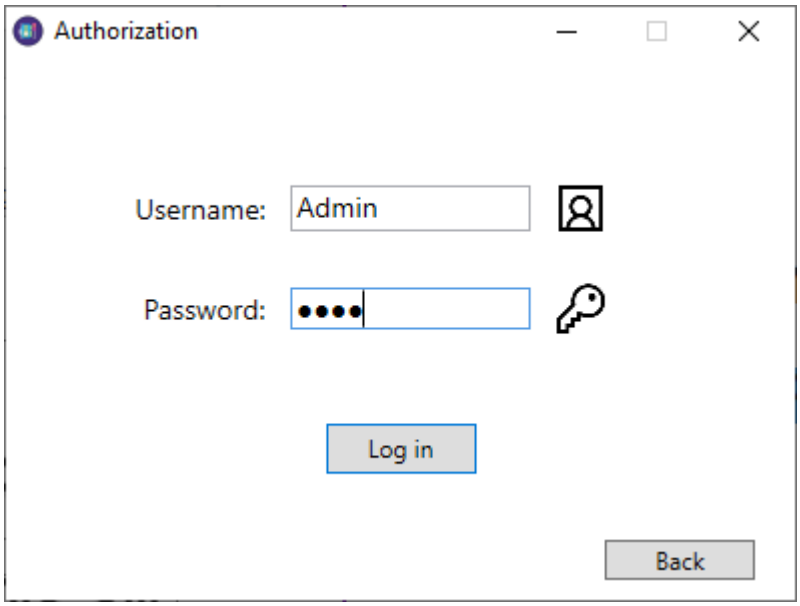


Рисунок 3.31 – Вікно авторизації

Після введення правильного логіна і пароля користувач-адміністратор отримує доступ до вікон бази даних і бази знань, надалі проходити авторизацію йому немає потреби, він може вільно переходити між вікнами додатку.

Вікно бази даних (рисунок 3.32) містить ряд таблиць з характеристиками комплектуючих і наборами фільтрів до цих характеристик, аналогічних тим, що наявні у головному вікні додатку. Також, це вікно надає опції для роботи з таблицями даних (додавання, редагування, видалення записів) і відповідно самою базою даних (збереження, скасування змін, створення/відновлення за резервною копією, оновлення таблиці з комплектуючими обраного типу).

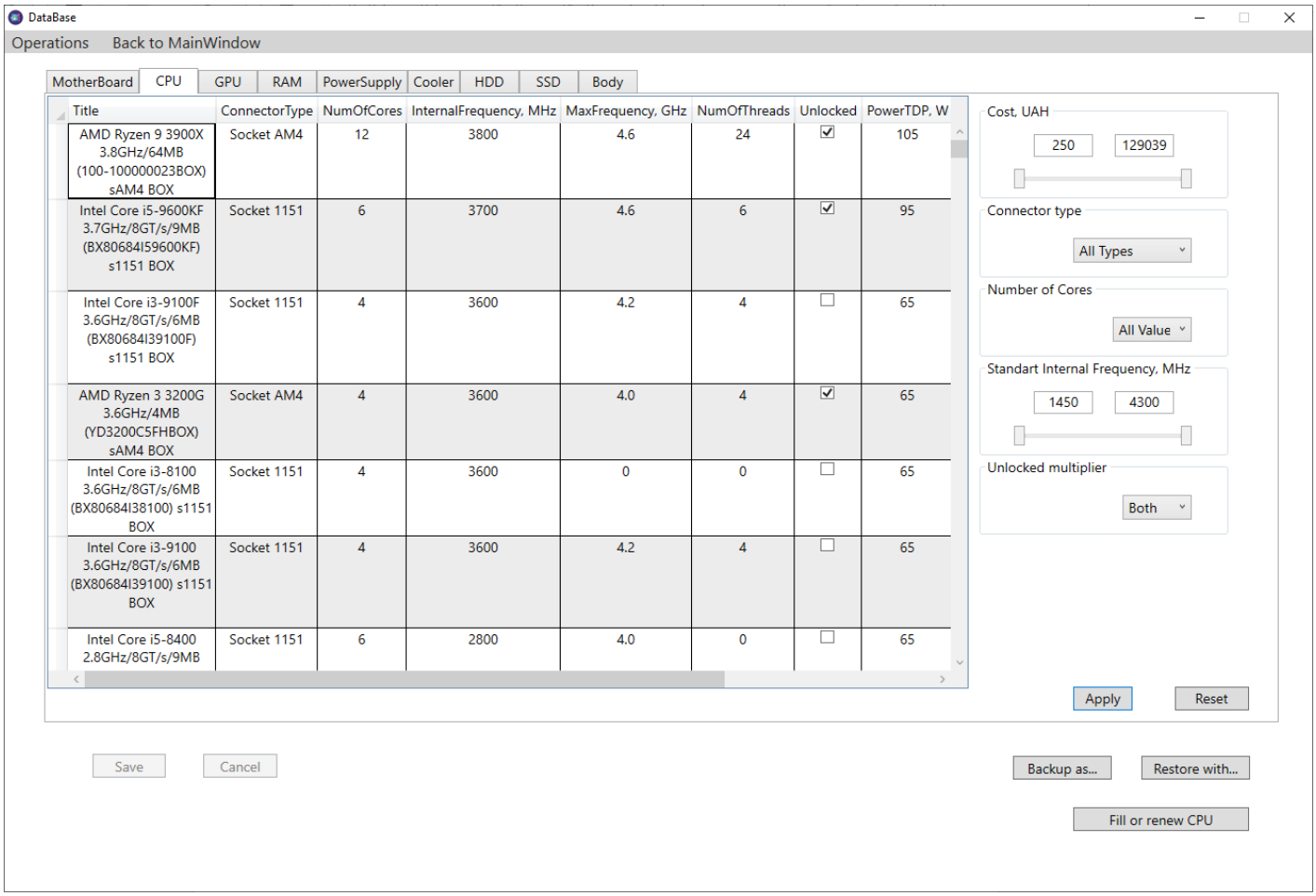


Рисунок 3.32 – Вікно бази даних

Додавання записів до таблиці даних відбувається шляхом встановлення курсора в одну з комірок останнього рядка таблиці і введенням в неї певного значення, інші комірки початково заповнюються значеннями за замовченням. Після внесення будь-яких змін до таблиць даних стають доступними кнопки для збереження або скасування змін, а також пункти меню, що дублюють зазначені опції (рисунок 3.33).

Редагування записів передбачає зміну значень комірок вже наявних в таблиці даних значень (рисунок 3.34).

Видалення записів проводиться шляхом виділення мишею або клавіатурою необхідних записів (одного або більше) і натискання клавіши «Delete» на клавіатурі (рисунок 3.35).

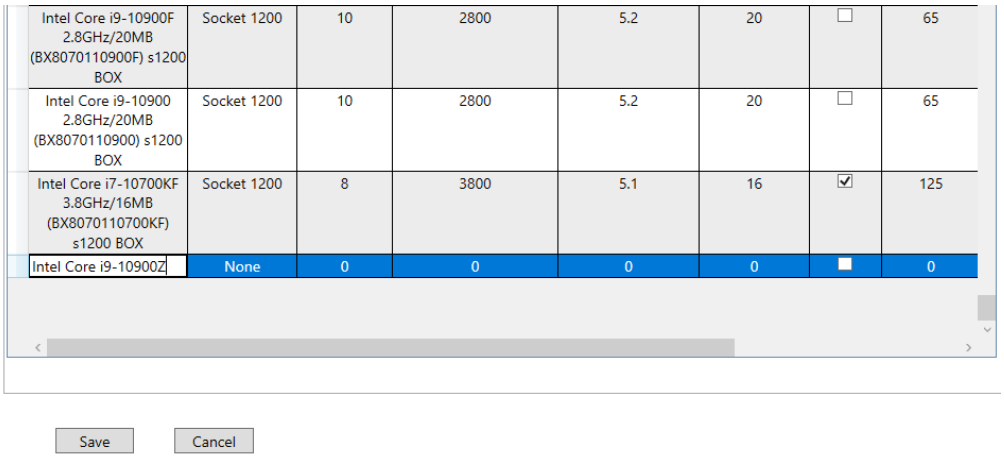


Рисунок 3.33 – Додавання записів да таблиці даних

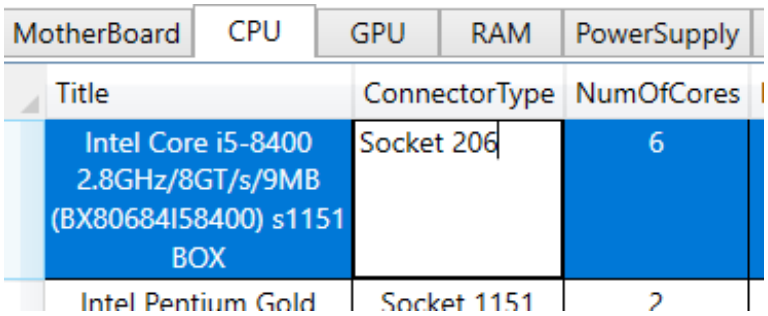


Рисунок 3.34 – Редагування записів таблиці даних

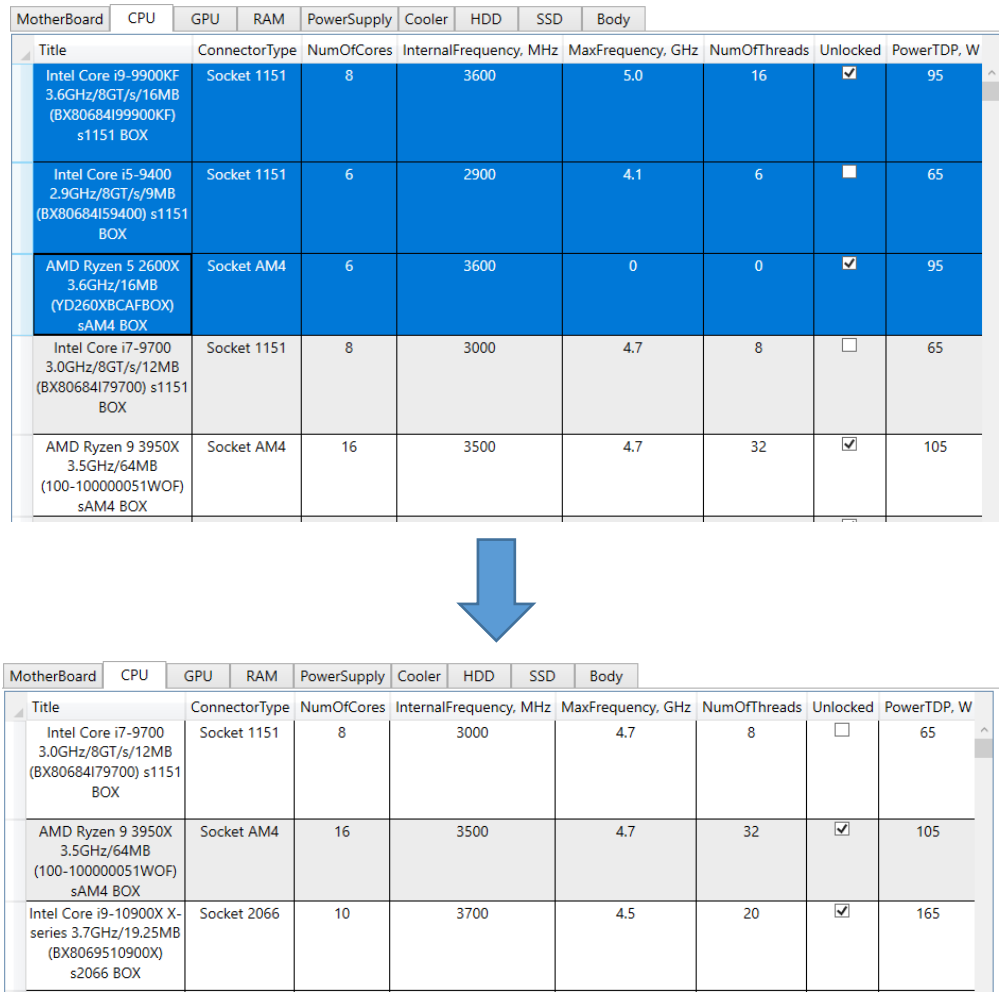


Рисунок 3.35 – Видалення записів з таблиці даних

Операції створення резервної копії та відновлення за резервною копією доступні тільки коли всі зміни, внесені до таблиці даних зафіксовані. Перша з них передбачає відкриття діалогового вікна збереження файлу резервної копії бази даних (формат .bak), і відповідно його збереження в обрану користувачем директорію (рисунок 3.36). При виконанні другої операції з’являється діалогове вікно завантаження/відкриття файлу, і відповідно, відновлення стану бази даних за тою конфігурацію, що зберігаються в обраному файлі резервної копії (рисунок 3.37).

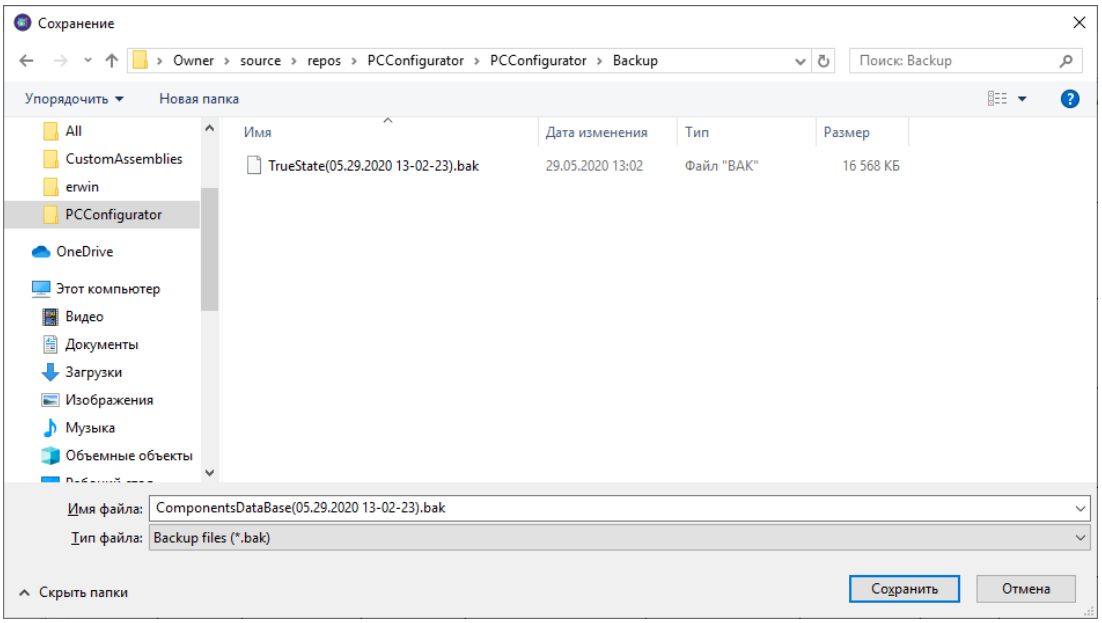


Рисунок 3.36 – Вікно збереження резервної копії

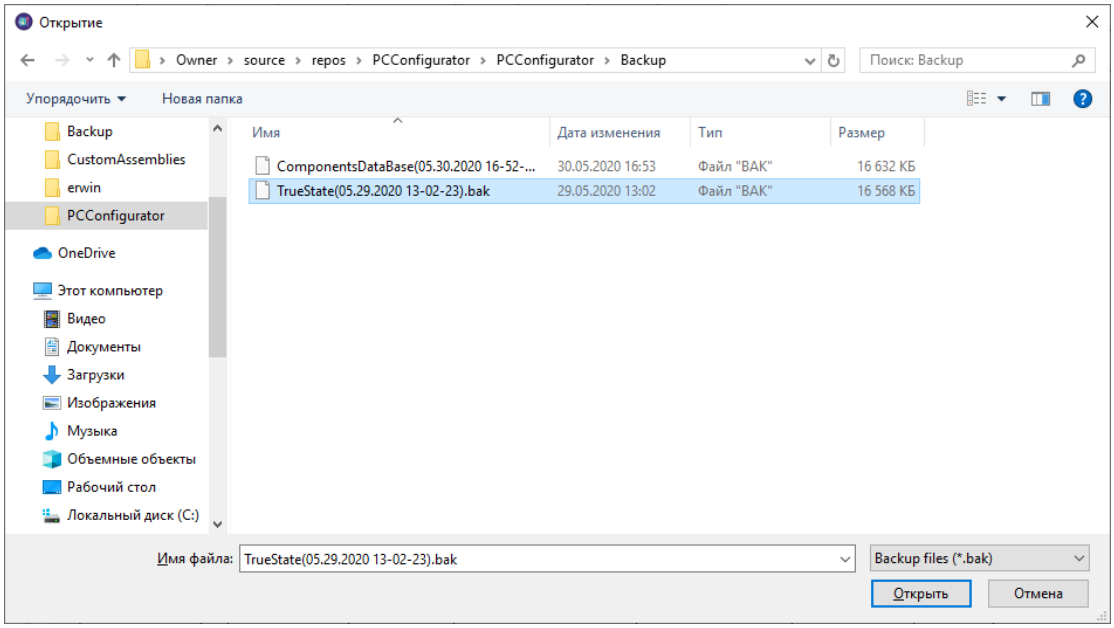


Рисунок 3.37 – Вікно збереження резервної копії

Операція оновлення обраної таблиці даних передбачає вибір однієї з таблиць даних (однієї з вкладок), і натискання кнопки або пункту горизонтального меню «Fill or renew *Component*», де замість слова «*Component*» відображається обраний тип комплектуючих. Наприклад, була обрана вкладка з комплектуючими типу центральний процесор, тоді відбувається відкриття браузера, й інформація про

комплектуючі та їх характеристики автоматично зчитується за визначеним алгоритмом, адміністратор бачить лише переходи браузера по сторінках з наборами по шістдесят комплектуючих обраного типу (рисунок 3.38), і після проходу по всім компонентам браузер закривається а обрана таблиця оновлюється.

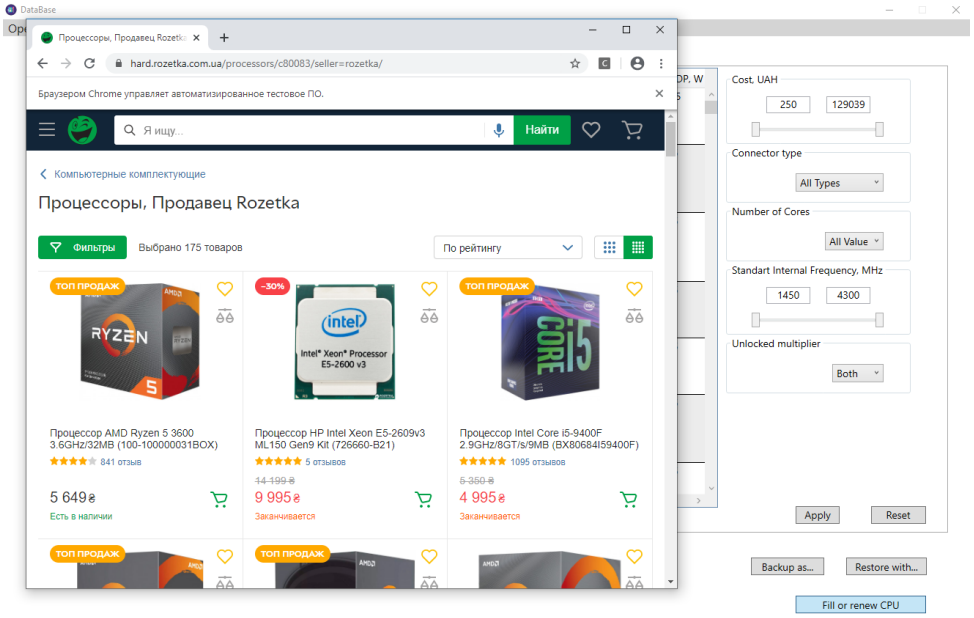


Рисунок 3.38 – Зчитування інформації з інтернет-ресурсу

Вікно бази знань (рисунок 3.39) містить таблицю з правилами сумісності, правилами оцінки характеристик, а також надає опції для збереження і відхилення змін активності (увімкнене/вимкнене) та пояснень до правил.

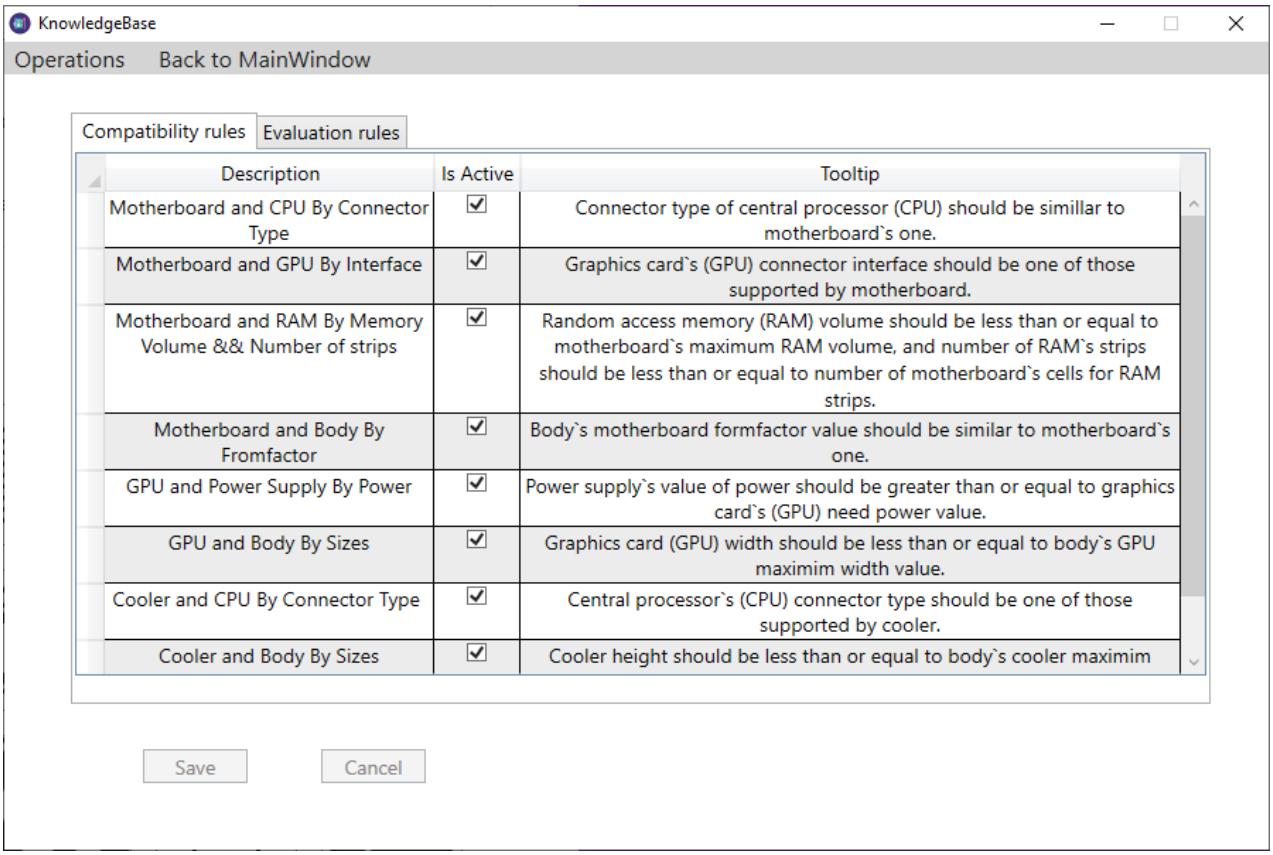


Рисунок 3.39 – Вікно бази знань

Таблиці з правилами передбачають тільки редагування окремих полів: зміна стану перемикача, що відповідає за те чи використовуються відповідне правило у перевірці сумісності або оцінці характеристик, редагування пояснень, що виникають при невідповідності певному правилу. Після зміни значень полів стають доступними кнопки збереження та відхилення змін, а також пункти меню, що дублюють відповідні опції (рисунок 3.40).

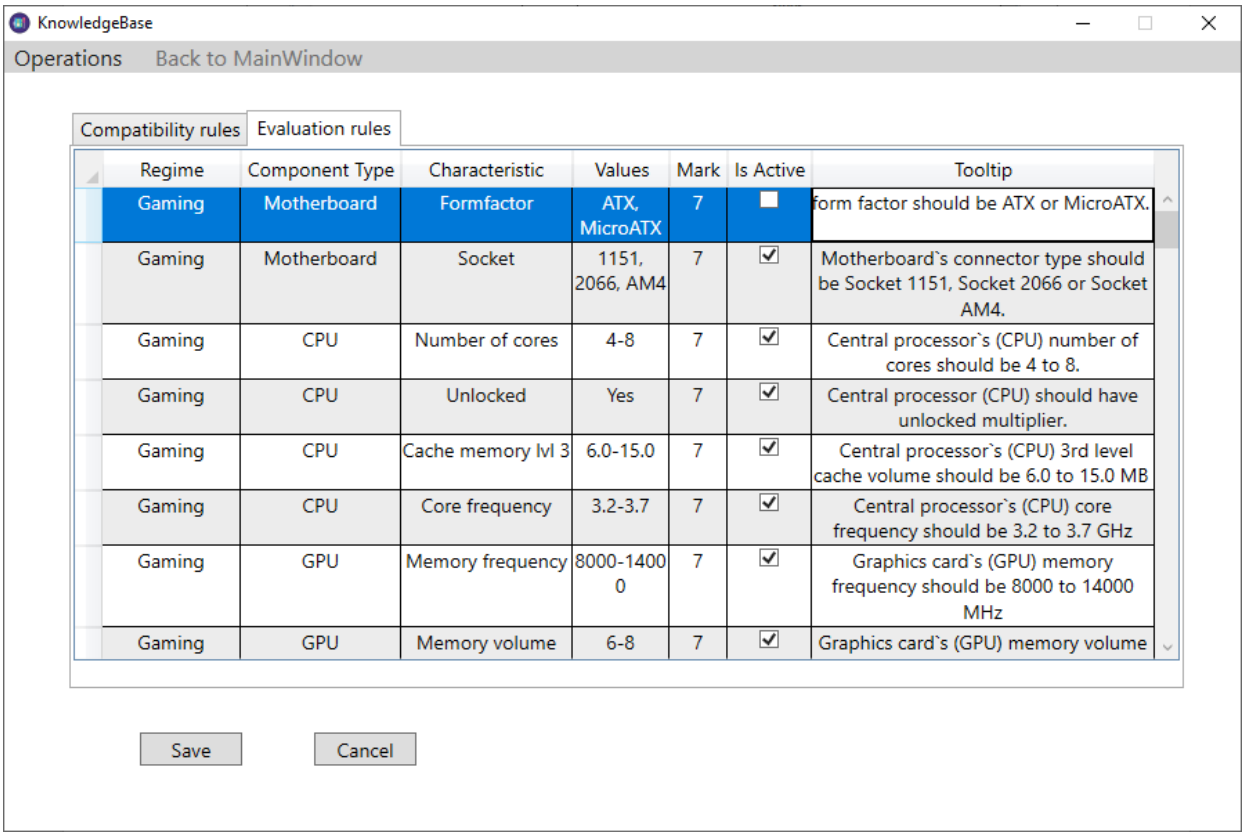


Рисунок 3.40 – Зміна стану та пояснення до правила

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було розроблено систему підтримки прийняття рішень з підбору комплектації комп'ютерної техніки.

Проведено аналіз останніх досліджень і публікацій, проаналізовано програмні продукти, що надають аналогічний або схожий функціонал, сформульовані мета, що полягає у розробці програмного додатку, що представлятиме собою інструмент для підбору необхідних компоненти ПК в залежності від потреб користувача, і задачі дипломного проекту.

Для досягнення мети та виконання поставлених задач було проведено планування робіт проекту: створено структура робіт та виконавців, побудовано матрицю відповідальності, графік виконання робіт, а також визначені ризики та можливі шляхи їх запобігання негативних наслідків.

Спроектовано модель функціонування головного процесу системи у нотації IDEF0: її контекстну діаграму, а також діаграми декомпозиції 1-го і 2-го рівня. Створено модель варіантів використання, побудовано діаграми діяльності для кожного з прецедентів. Для відображення структури та взаємозв'язків між класами було побудовано діаграму класів.

Проведено моделювання процесів передачі даних у системі й створено DFD діаграму 0-го і 1-го рівня. Сформовано структуру бази даних комплектуючих, що містить сутності для зберігання атрибутів комплектуючих, а також визначень правил, за якими відбувається перевірка сумісності та оцінка характеристик компонентів сформованої збірки.

Визначені та застосовані інструменти реалізації додатку: інтегроване середовище розробки Microsoft Visual Studio 2017, мова програмування C#, графічна підсистема WPF, а також особливості його структури – шаблон проектування MVVM.

Створено ряд графічних вікон для підбору комплектуючих, роботи з базою даних і базою знань. Реалізовано функціонал з пошуку елементів і компонування збірки, перевірки сумісності її компонентів та оцінювання відповідності до

встановлених потреб. Також, виконано функціонал для збереження/скасування змін бази даних і бази знань, опції створення резервної копії, відновлення за цією копією та заповнення або оновлення таблиці даних комплектуючих обраного типу.

Дана система може бути дуже корисною для недостатньо обізнаних у галузі комп’ютерного апаратного забезпечення користувачів, яким необхідно без зайвих зусиль сформувати нову комп’ютерну збірку в залежності від своїх потреб, а також для більш досвідчених користувачів, які бажають підтвердити свої знання у цій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Выбор компьютера 2020. Выбираем комплектующие // 27.12.2019 URL: https://larga.ru/article/vybor-kompyutera (дата звернення: 16.01.2020).

2 Выбор компьютера, 2020 год // 08.12.2019 URL: https://www.dxdigitals.info/2013/06/vibor-personalnogo-kompyutera-2013.html (дата звернення: 18.01.2020).

3 Аппаратные компоненты офисного ПК [Електронний ресурс]. – Режим доступу: https://www.gyrnal.ru/library/ru/obshee/2/2/30/ (дата звернення: 21.01.2020).

4 Конфигуратор компьютера NerdPart // 2020 URL: https://telemart.ua/assembly.html (дата звернення: 02.02.2020).

5 Конфигуратор системного блока // 2020 URL: https://www.ironbook.ru/constructor/ (дата звернення: 03.02.2020).

6 Конфигуратор компьютера // 2020 URL: https://www.dns-shop.ru/configurator/ (дата звернення: 05.02.2020).

7 Конфигуратор компьютера // 2020 URL: https://can.ua/configurator/ (дата звернення: 07.02.2020).

8 Разработка функциональной модели // Методология IDEF0 URL: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema6/tema6_2 (дата звернення: 09.02.2020).

9 Модель вариантов использования // Диаграмы вариантов использования URL: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema12/tema12_2 (дата звернення: 16.02.2020).

10 Модель проективроания // Диаграммы классов URL: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema14/tema14_2 (дата звернення: 07.03.2020).

11 Модель проектирования // Диаграммы деятельности URL: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema14/tema14_3 (дата звернення: 10.03.2020).

12 Разработка функциональной модели // Методология DFD URL: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema6/tema6_3 (дата звернення: 22.03.2020).

13 Weil A. Learn WPF MVVM - XAML, C# and the MVVM pattern. lulu.com, 2017. 174 с.

14 Zeeshan H., Larry T., Nitin G., Brian Driscoll., Robert V. Entity Framework 6 Recipes. Apress 2nd Edition, 2013. 548 с.

15 Richter J. CLR via C# 4th Edition. Developer Reference. Microsoft Press, 2012. 896 с.

16 MacDonald M. Pro WPF 4.5 in C. Windows Presentation Foundation in .NET 4.5 4th Edition. Apress, 2012. 1111 с.

17 Компьютерные комплектующие. // 2020 URL: https://hard.rozetka.com.ua/ (дата звернення: 20.05.2020).

ДОДАТОК А
ТЕХНІЧНЕ ЗАВДАННЯ

А.1 Призначення й мета створення системи

А.1.1 Призначення системи

Система має представляти засіб для допомоги у підборі комплектації персонального комп’ютера (ПК), за встановленими користувачем критеріями.

А.1.2 Мета створення системи

Забезпечення недосвідчених користувачів зручним інструментом пошуку комплектуючих для свого ПК, підвищення їх обізнаності у цьому питанні, та скорочення часу при виборі найоптимальніших комплектуючих за встановленими вимогами.

А.1.3 Цільова аудиторія

Недостатньо обізнані користувачі ПК, яким необхідно швидко та ефективно знайти окремі компоненти, або сформувати нову збірку.

А.2 Вимоги до системи

А.2.1 Вимоги до системи в цілому

А.2.1.1 Вимоги до структури й функціонування системи

Системи повинна бути реалізована у вигляді віконного додатку, що працює під операційною системою (ОС) Windows, для використання основних функцій якого, не є обов’язковим наявність Інтернет з’єднання. У додатку повинен бути передбачений функціонал для роботи з базою даних комплектуючих, а також базою знань з правилами сумісності та оцінками характеристик для підбору комплектуючих.

А.2.1.2 Вимоги до персоналу

Для підтримки коректної та ефективної роботи додатку від персоналу вимагається своєчасне (щотижневе) оновлення інформації про існуючі та доступні комплектуючі за допомогою відповідного функціоналу, а отже, базові навички для роботи з віконними додатками ОС Windows. Необхідний достатній рівень обізнаності в питанні підбору комплектуючих для ПК для перегляду наявних правил та оцінок характеристик бази знань при оновленні інформації про наявні комплектуючі та, за необхідності, редагування цих елементів.

А.2.1.3 Вимоги до збереження інформації

Система керування базою даних з інформацією про комплектуючі повинна передбачати автоматичне створення резервної копії при загальному її оновленні, а також функціональну можливість створення резервних копій за потребою адміністратора системи.

А.2.1.4 Вимоги до розмежування доступу

За доступом до даних та функціональних можливостей додатку користувачі поділяються на два типи:

Звичайний користувач, що має доступ до основних функцій додатку – пошук окремих комплектуючих або цілої збірки за встановленими потребами та обмеженнями.

Адміністратор, що має доступ до функцій пропонованих звичайному користувачу, а також функцій для роботи з базою даних комплектуючих – додавання, редагування, видалення записів, оновлення обраної таблиці, створення резервної копії та відновлення за резервною копією; з базою знань для підбору комплектуючих – вмикання та вимикання правил, редагування оцінок характеристик комплектуючих.

А.2.2 Вимоги до функцій, виконуваних додатком

А.2.2.1 Основні вимоги

А.2.2.1.1 Структура додатку

Додаток повинен складатися з вікон розділених за функціональним призначенням, виділяються такі основні вікна:

- Головне вікно: для виконання пошуку комплектуючих за встановленими потребами із отриманням пояснень/рекомендацій, щодо сформованої збірки.
- Вікно бази даних: для адміністрування бази даних, що містить інформацію про комплектуючі, які використовуються при підборі, доступно тільки адміністратору.
- Вікно бази знань: для адміністрування бази знань, що містить набір правил, необхідних для перевірки сумісності компонентів збірки, та оцінок характеристик компонентів, доступно тільки адміністратору.

А.2.2.1.2 Навігація

У додатку повинен бути передбачений користувальницький інтерфейс, що забезпечує інтуїтивно зрозуміле представлення його структури та функціональних можливостей, логічні та достатньо швидкі переходи між вікнами, та перемикання у кожному з вікон. Елементи навігації повинні надавати однозначне розуміння їх змісту: переходи між вікнами повинні мати відповідні підписи, усі графічні елементи навігації повинні мати підписи, і відповідати загальноприйнятим позначенням.

Для навігації у головному вікні повинні використовуватись меню: верхнє горизонтальне меню для переходу до інших функціональних вікон за наявності

доступу. Навігація в інших вікнах (з базою даних та базою знань) відбувається за допомогою кнопок (закриття вікна), що забезпечують повернення до головного вікна.

А.2.2.1.3 Наповнення вікон додатку

Інформація, яку може отримати звичайний користувач на головному вікні додатку при застосуванні доступних йому функцій підбору комплектуючих, повинна формуватися на основі вмісту бази даних та бази знань експертної системи. У вікні з базою даних повинна розташовуватись таблиця, в якій відображатимуться записи з характеристиками комплектуючих обраного типу. Вікно з базою знань має містити список наявних правил і таблиці з оцінками характеристик комплектуючих.

А.2.2.1.4 Система навігації (карта додатку)

Взаємозв'язок між вікнами, а також основні елемент вмісту кожного з вікон представлено на рисунку А.1.

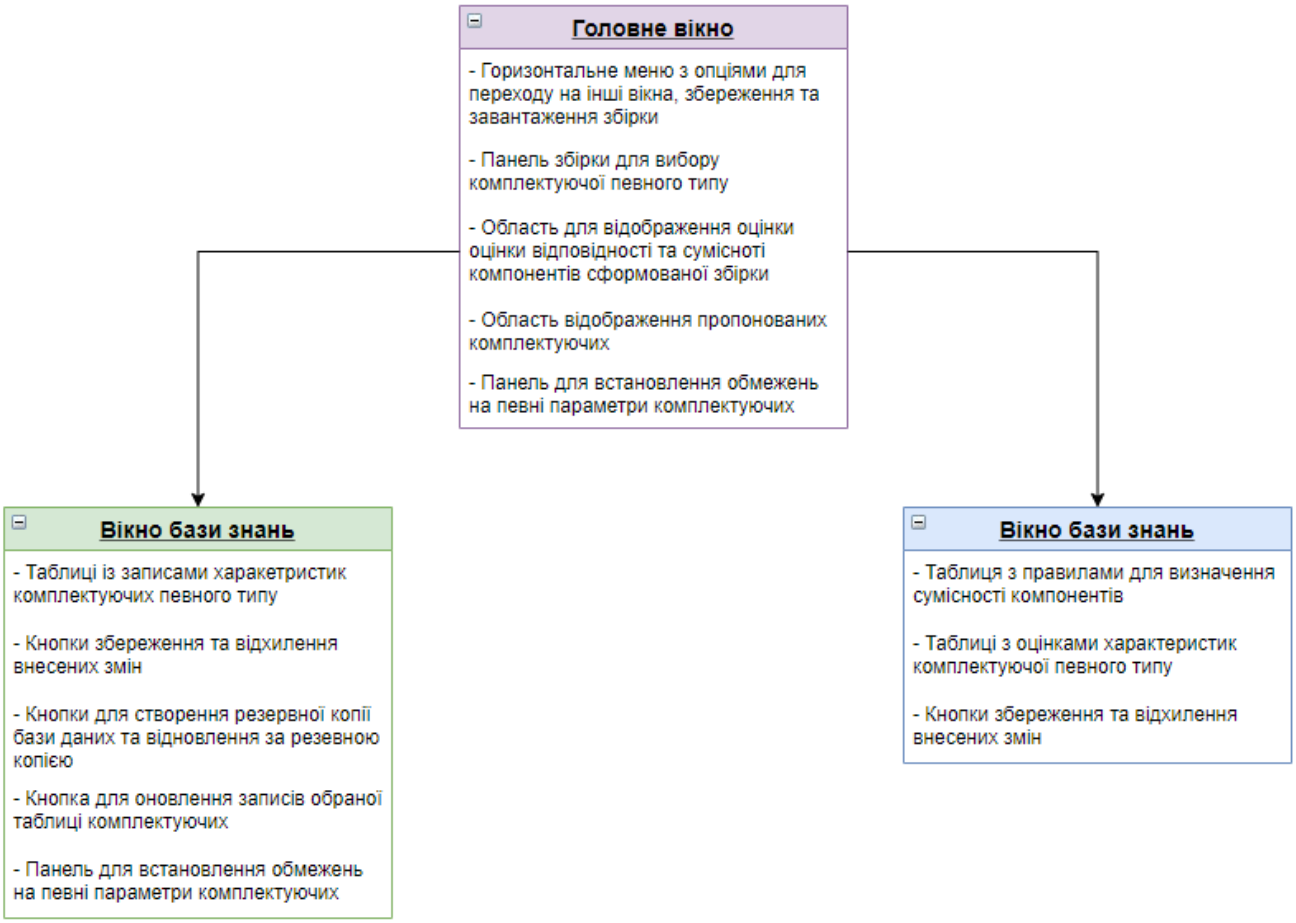


Рисунок А.1 – Карта додатку

А.2.2.1.5 Функціональні можливості вікон

Головне вікно повинно надавати такі функціональні можливості:

- пошук компонентів збірки за встановленими вимогами (режимом підбору);
- фільтрація за характеристиками для окремого компоненту збірки;
- надання інформації про компоненти збірки, їх характеристики та сукупну відповідність рекомендованих компонентів визначеним потребам;
- авторизація для отримання доступу до функціоналу адміністратора.

Вікно бази даних повинно надавати такі функціональні можливості:

- фільтрація записів;
- сортування записів
- додавання записів;
- редагування записів;
- видалення записів;

- оновлення записів обраної таблиці даних;
- створення резервної копії бази даних;
- відновлення бази даних за резервною копією;
- збереження змін;
- відхилення змін (до збереження).

Вікно бази знань повинно надавати такі функціональні можливості:

- перемикання активності правил сумісності;
- встановлення допоміжного тексту для правила сумісності;
- редагування оцінок характеристик комплектуючих;
- встановлення допоміжного тексту для кожного з діапазонів коефіцієнту відповідності;
- збереження змін;
- відхилення змін (до збереження).

А.2.2.1.6 Загальні вимоги

Всі вікна повинні бути виконані у стилі, що поєднує в собі мінімалізм та інформативність. Весь функціонал, що надається звичайному користувачу повинен безвідмовно та коректно працювати без наявності доступу в Інтернет. Повинна бути передбачена обробка виключних ситуацій у випадку некоректного введення інформації користувачем та у разі виникнення проблем з апаратним забезпеченням користувача.

Приблизний вигляд та розташування елементів інтерфейсу головного вікна наведено на рисунку А.2.

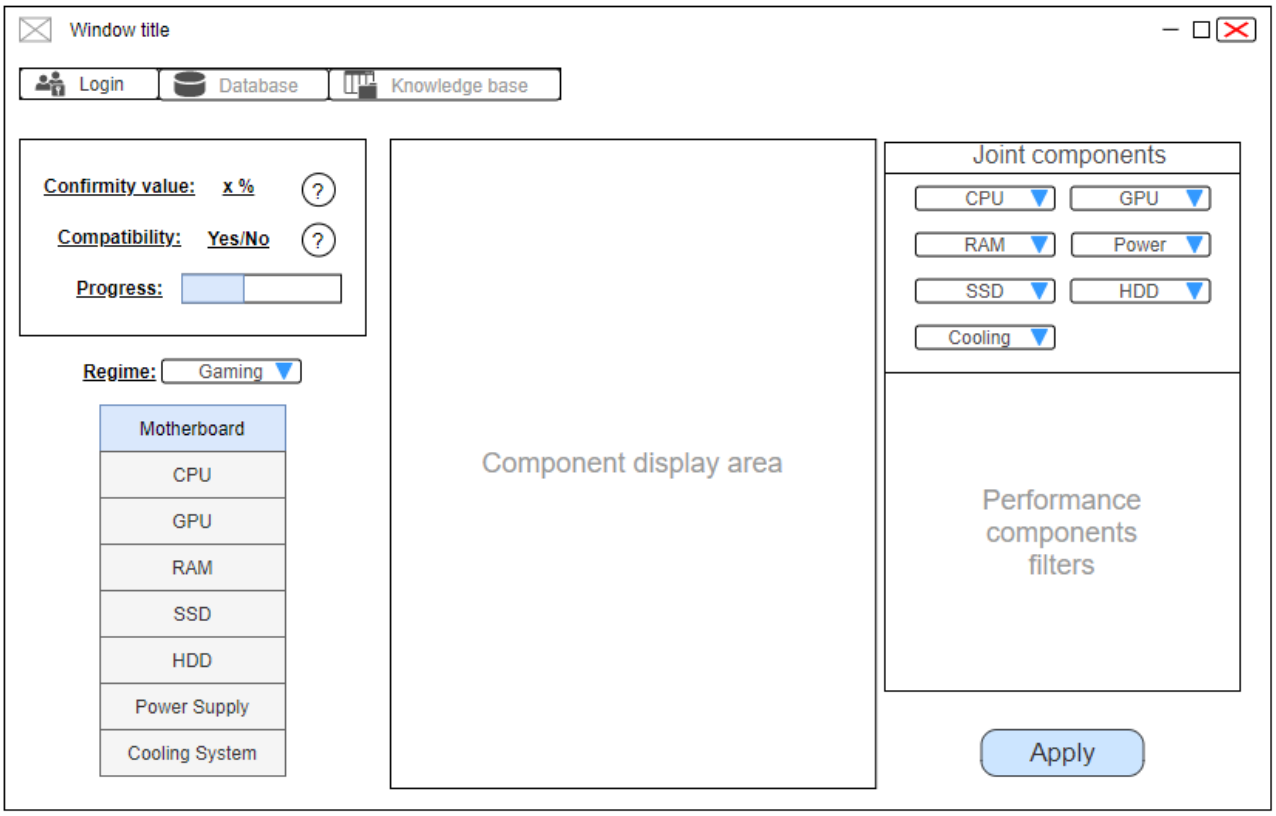


Рисунок А.2 – Головне вікно

А.2.2.1.7 Типові навігаційні та інформаційні елементи

- Верхнє горизонтальне меню
- Область відображення відповідності збірки та сумісності її компонентів
- Панель збірки для вибору комплектуючих
- Область відображення компонентів
- Панель фільтрування

А.2.2.1.8 Верхнє горизонтальне меню

Верхнє горизонтальне меню повинно складатися з пунктів збереження та завантаження збірки, переходу до вікон бази даних і бази знань, причому, для доступу до двох останніх користувач повинен проходити авторизацію у відповідному вікні, після вибору цих пунктів.

А.2.2.1.9 Область відображення відповідності збірки та сумісності її компонентів

Дана область повинна містити значення відповідності сформованої збірки до встановлених користувачем потреб (режиму підбору), інформацію про сумісність обраних компонентів, а також прогрес заповнення збірки.

А.2.2.1.10 Панель збірки для вибору комплектуючих

Панель збірки повинна складатися з елементів, що передбачають встановлення комплектуючої певного типу з можливістю вибору іншої комплектуючої або повернення у початковий стан.

А.2.2.1.11 Область відображення

В область відображення, в залежності від обраного елементу на панелі збірки, повинні відображатися наявні комплектуючі з можливістю перегляду їх параметрів, і можливістю їх додавання до збірки.

А.2.2.1.12 Панель фільтрування

Панель фільтрування повинна, в залежності від обраного елементу панелі збірки, надавати опції для встановлення значень характеристик відповідного компоненту, і при натисканні відповідної кнопки оновлювати список комплектуючих згідно встановлених фільтрів.

А.2.3 Вимоги до видів забезпечення

А.2.3.1 Вимоги до інформаційного забезпечення

Реалізація додатку відбувається з використанням:

- .NET Framework 4.6.1;
- Microsoft Visual Studio 2017 (C# 7.0);
- Windows Presentation Foundation (WPF) 4.5;
- MS SQL 13.0;
- Microsoft SQL Server Management Studio 17.

А.2.3.2 Вимоги до програмного забезпечення

Для коректної роботи додатку програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

- ОС Windows 7/8/8.1/10;
- програмний пакет .NET Framework v.4.6.1 або вище.

А.2.3.3 Вимоги до апаратного забезпечення

Для запуску додатку, апаратне забезпечення клієнтської частини повинно задовольняти таким мінімальним вимогам:

- процесор з мінімальною частотою в 2.0 GHz;
- ОЗУ 2 Gb для 32/64-розрядної версії ОС;
- 5 Gb дискового простору для 32/64-розрядної ОС;
- графічна плата з 3D-прискоренням DirectX v.9 або вище;
- мінімальна роздільна здатність дисплея 800 x 600;
- підключення до інтернету (для функцій адміністрування);
- наявність маніпулятору «миша» та клавіатури.

3 Склад і зміст робіт зі створення додатку

Опис етапів роботи зі створення додатку наведено в табл. А.1.

Таблиця А.1 – Етапи створення додатку

№	Склад і зміст робіт	Строк розробки (робочих днів)
1	База даних комплектуючих: Проектування структури бази даних та взаємозв’язків між сутностями, створення таблиць для всіх типів комплектуючих	2
2	Робота з базою даних: Створення механізму заповнення бази даних та оновлення записів окремої таблиці, механізму створення резервної копії, відновлення за резервною копією, впровадження функцій додавання, редагування та видалення записів, фільтрація записів, а також збереження та скасування внесених змін	5
3	База знань: Проектування структури бази знань, встановлення шаблонів правил, які вона містить, переліку правил сумісності, правил оцінки характеристик	4
4	Робота з базою знань: Впровадження функцій перемикання активності правил, редагування оцінок характеристик кожної з комплектуючих, а також збереження та скасування внесених змін	3

Продовження таблиці А.1

№	Склад і зміст робіт	Строк розробки (робочих днів)
5	Розробка структури та інтерфейсу: Створення головного вікна, вікна бази даних та бази знань, проектування та впровадження інтерфейсу цих вікон	2
6	Формування збірки: Впровадження опцій для вибору комплектуючих певного типу за встановленим режимом підбору та фільтрами до характеристик	2
7	Визначення відповідності збірки: Розрахунок та виведення відсотку відповідності, сумісності компонентів та відсотку сформованості для створюваної користувачем збірки	3
8	Розмежування доступу: Створення додаткового діалогового вікна авторизація для адміністратора, визначення логіна та пароля адміністратора, а також алгоритму шифрування пароля, закриття доступу до бази даних та бази знань для неавторизованого користувача	1
9	Тестування функціоналу та завершення роботи: Перевірка коректності роботи всіх функціональних можливостей додатку, пошук та виправлення помилок в інтерфейсі та неточностей в алгоритмах підбору комплектуючих	2
	Загальна тривалість робіт	24

4 Вимоги до складу й змісту робіт із введення додатку в експлуатацію

Для правильного функціонування додатку на комп’ютері кінцевого користувача передбачається створення інсталяційного пакету, що включатиме сам додаток, а також всі додаткові компоненти, необхідні для коректної та ефективної роботи додатку: .NET Framework 4.6.1 файл/файли з поточною версією записів бази даних комплектуючих, а також файли з набором правил бази знань і оцінками компонентів.

ДОДАТОК Б
ПЛАНУВАННЯ РОБІТ

Б.1 Ідентифікація ідеї проекту

Першою проблемою з якою стикається рядовий користувач при виборі комп’ютера – що саме і де краще придбати. Таким чином породжується питання: «скільки коштів доцільно витратити на покупку ПК?». Найпростіше рішення – брати все найкраще зазвичай відпадає з причини нестачі коштів. Тому більшості людей доводиться обирати конфігурацію комп’ютера виходячи із задач, для яких їм потрібен цей комп’ютер, а потім вже встановлювати компроміс між технічними характеристиками і фінансовими можливостями.

Існує такий вираз – «збалансована конфігурація комп’ютера». Це така комбінація основних комплектуючих (материнської плати, процесора, відеоадаптера, оперативної пам’яті, жорстокого диску, блоку живлення та корпусу), в якій жодна з комплектуючих не є надлишково або недостатньо потужною відносно всіх інших. Очевидно, що підібравши саме таку комбінацію користувач отримує оптимальне співвідношення ціна/продуктивність, але для такого підбору комплектуючих необхідна певна обізнаність в цій області. Далеко не кожен покупець ладен витрачати купу часу на отримання цих знань, проте кожен захоче мати максимально підходящу для його потреб збірку, ще й за розумну ціну.

Отже, на основі саме таких потреб користувачів з’явилася ідея проекту, метою якого є створення додатку, що дозволятиме/допомагатиме, з мінімальними витратами часу та зусиль, обрати компоненти комп’ютера, за встановленими потребами, а також надаватиме підказки/пояснення до зробленого користувачем вибору.

Б.2 Деталізація мети за методикою SMART

Конкретна (Specific). Створити програмний додаток для підбору комплектуючих за потребами користувача.

Вимірювана (Measurable). Процес створення додатку триває не більше 30 днів, витратами ресурсів – мінімальні (без фінансування).

Досяжна (Achievable). Наявна достатня кількість інформації про зазначену область знань, апаратне і програмне забезпечення, а також спеціаліст, спроможний реалізувати необхідний функціонал.

Актуальна (Relevant). Зменшення кількості помилкових рішень при виборі комплектуючих ПК і підвищення обізнаність в цій галузі.

Обмежена у часі (Time-bound). Створено календарний план проекту, що передбачає його вчасне виконання.

Б.3 Опис фази розробки програмного додатку

Б.3.1 Планування змісту робіт для розробки додатку

Для того щоб спланувати будь-який проект, в першу чергу треба зрозуміти з чого він складається, тобто, які роботи треба виконати, щоб досягти цілей проекту. Саме для цього застосовується інструмент, який називається Work Breakdown Structure (WBS). Він орієнтований на основні результати проекту, що визначають його предметну область. Кожен нижній рівень структури є деталізацією вищого рівня проекту. Таким чином, проект деталізується до елементарних робіт, тобто, тих робіт, що можуть бути виконані однією особою.

WBS для розроблюваного програмного додатку наведено на рисунку Б.1.

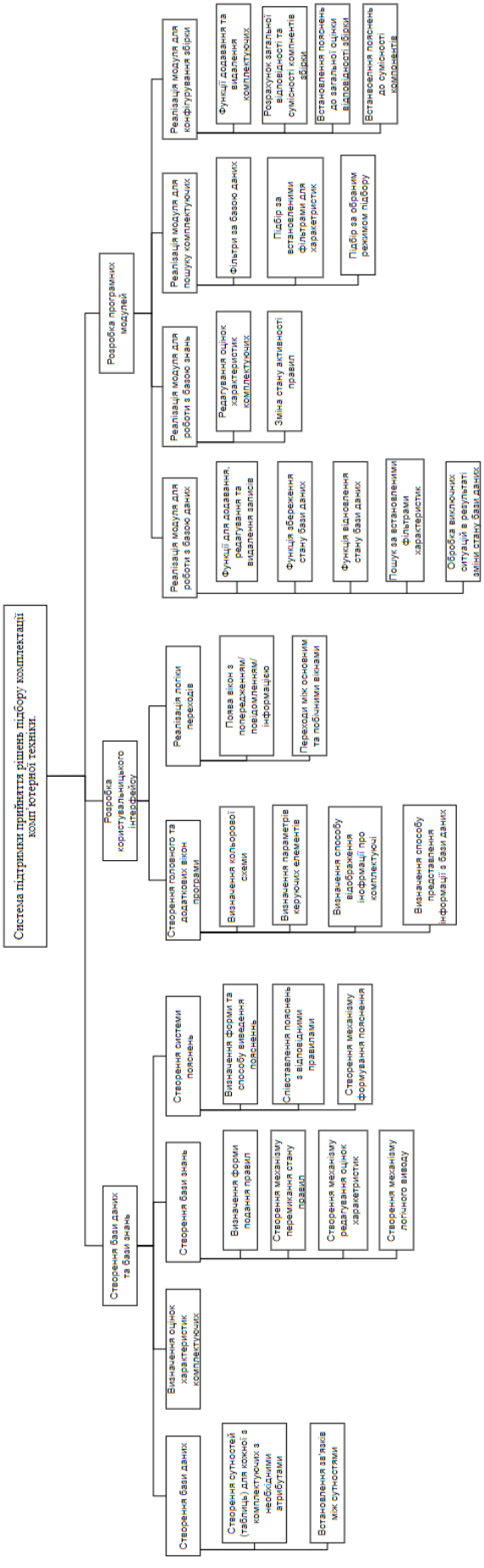


Рисунок Б.1 – WBS

Б.3.2 Планування структури виконавців для розробки додатку

Отримавши декомпозицію робіт проекту за допомогою WBS, членам команди призначаються відповідні до їх спеціалізації роботи, і все це зазначається в організаційній структурі робіт.

Organizational Breakdown Structure (OBS) може будуватися аналогічно ієрархічній структурі робіт, в такому випадку її називають функціональною, тобто, на верхньому рівні відображається організаційна структура (команда, але у даному випадку лише один виконавець), як єдиний елемент, потім відбувається розподіл по відділам, а найнижчі рівні міститимуть окремого виконавця, якому призначена елементарна робота відповідно до WBS.

Функціональна OBS проекту наведена на рисунку Б.2.

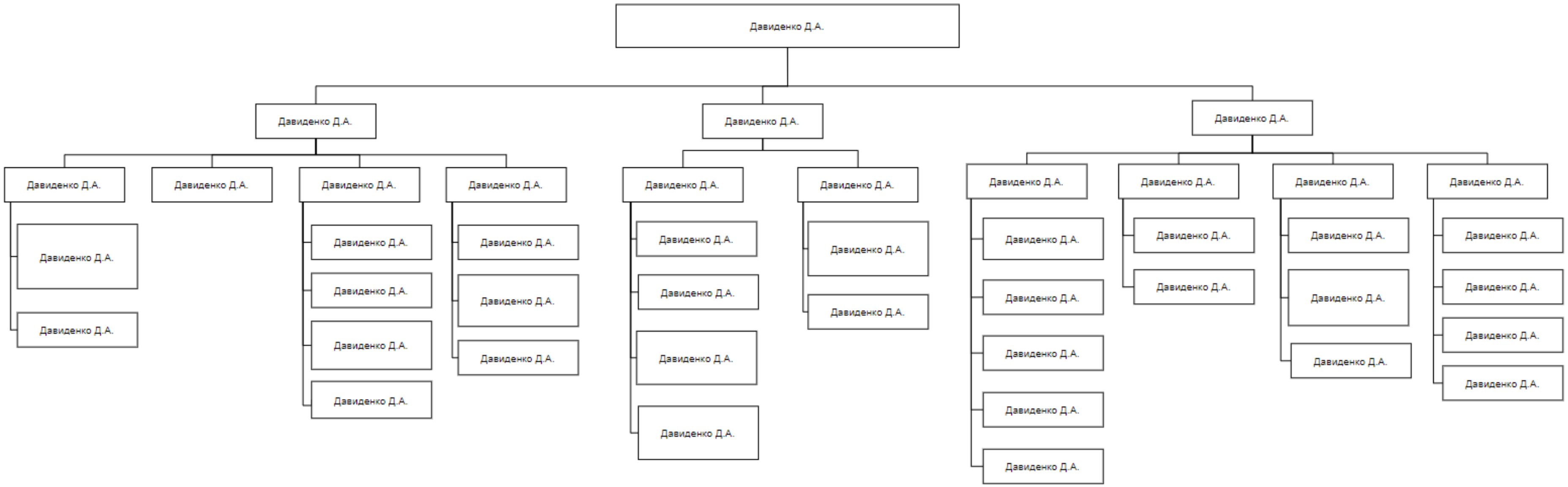


Рисунок Б.2 – OBS

Б.3.3 Побудова матриці відповідальності

Для співставлення набору робіт і відповідних їм виконавців створюється матриця відповідальності.

Responsibility Assignment Matrix (RAM), заповнюється на основі WBS та OBS проекту. RAM описує рольовий розподіл за операціями певного процесу (роботи), її елементами є, заздалегідь визначені, позначення видів діяльності. У найбільш класичному варіанті RAM це RACI-діаграми:

- R-Responsible – відповідальний за виконання роботи;
- A-Accountable – приймає остаточне рішення, щодо виконаної роботи;
- C-Consulted – надає поради, щодо прийняття певних рішень по виконанню роботи;
- I-Informed – проінформований про виконання певної роботи.

Табл. Б.1 представляє матрицю відповідальності створення програмного додатку:

Таблиця Б.1 – Матриця відповідальності

Задачі	Давиденко Д.А.	Марченко А.В.	Шендрик В.В.
Визначення вимог до програмного продукту	R	A	
Створення сутностей (таблиць) для кожної з комплектуючих з необхідними атрибутами	R	I	
Встановлення зв’язків між сутностями	R	I	
Визначення оцінок характеристик комплектуючих	R	I	
Визначення форми подання правил	R	I	

Продовження таблиці Б.1

Задачі	Давиденко Д.А.	Марченко А.В.	Шендрик В.В.
Створення механізму перемикання стану правил	R	I	
Створення механізму редагування оцінок характеристик	R	I	
Створення механізму логічного виводу	R	I	
Визначення форми та способу виведення пояснень	R	I	
Співставлення пояснень з відповідними правилами	R	I	
Створення механізму формування пояснення	R	I	
Визначення кольорової схеми	R	I	
Визначення параметрів керуючих елементів	R	I	
Визначення способу відображення інформації про комплектуючі	R	I	
Визначення способу представлення інформації з бази даних	R	I	
Поява вікон з попередженням/ повідомленням/ інформацією	R	I	

Продовження таблиці Б.1

Задачі	Давиденко Д.А.	Марченко А.В.	Шендрик В.В.
Переходи між основним та побічними вікнами	R	I	
Функції для додавання, редагування та видалення записів	R	I	
Функція збереження стану бази даних	R	I	
Функція відновлення стану бази даних	R	I	
Пошук за встановленими фільтрами характеристик	R	I	
Обробка виключних ситуацій в результаті зміни стану бази даних	R	I	
Редагування оцінок характеристик комплектуючих	R	I	
Зміна стану активності правил	R	I	
ФіФільтри за базою даних	R	I	
Підбір за встановленими фільтрами для харакетристик	R	I	
Підбір за обраним режимом	R	I	
Функції додавання та видалення комплектуючих	R	I	

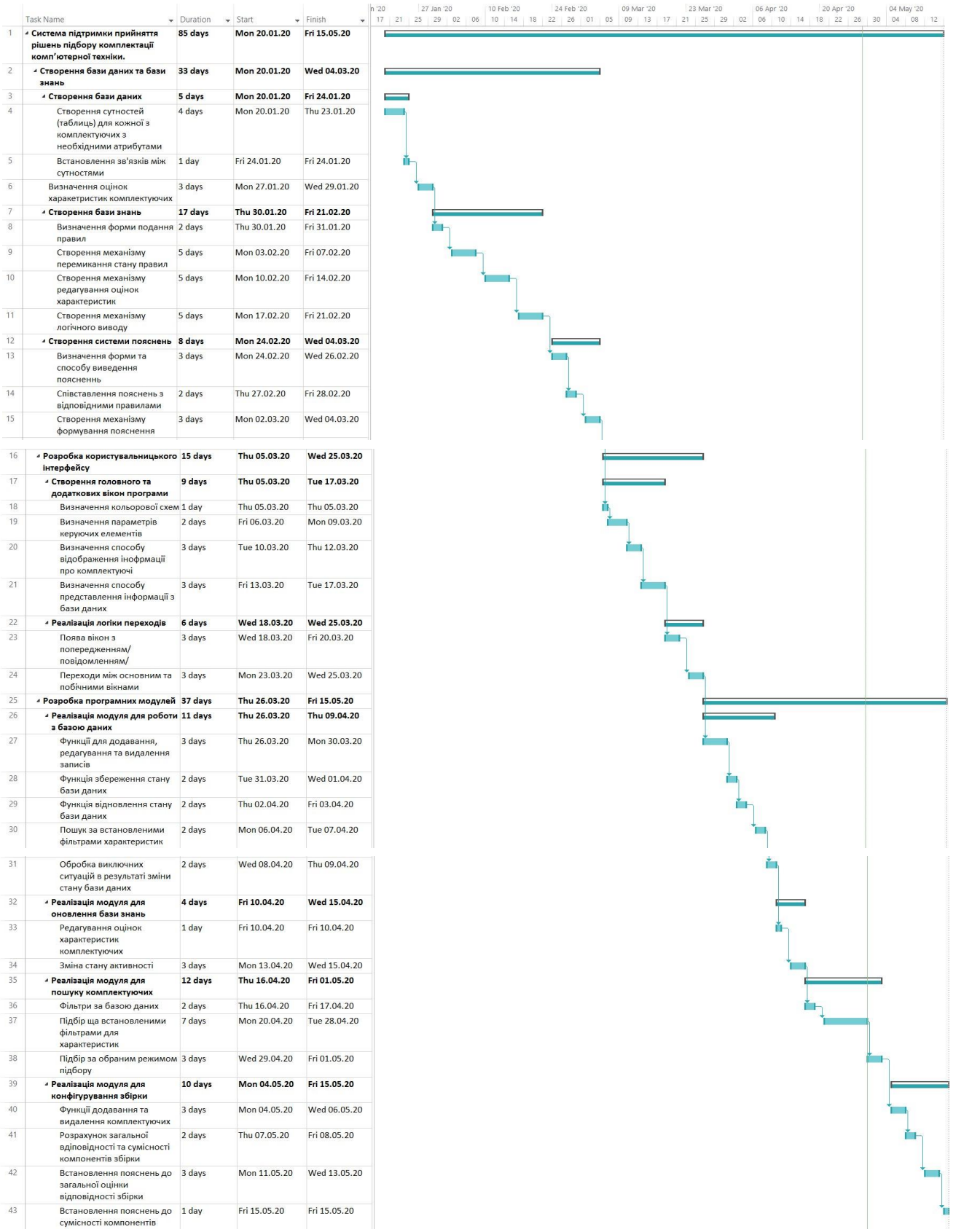
Продовження таблиці Б.1

Задачі	Давиденко Д.А.	Марченко А.В.	Шендрик В.В.
Розрахунок загальної відповідності та сумісності компонентів збірки	R	I	
Встановлення пояснень до загальної оцінки відповідності збірки	R	I	
Встановлення пояснень до сумісності компонентів	R	I	
Демонстрація результату розробки	R	I	A

Б.4 Побудова календарного графіку виконання розробки програмного додатку

Для формування плану (графіку) робіт по проекту використовується така стовпчаста діаграма, яка називається діаграма Ґанта.

Ця діаграма представляє собою відрізки, розташовані на горизонтальній часовій шкалі. Кожен відрізок відповідає окремій задачі, що складають загальний набір робіт проекту, і розташовуються вертикально. Початок, кінець і довжина відрізка на часовій шкалі відповідають датам початку, кінця та тривалості виконання задачі в проекті. Слід зазначити, що діаграма Ґанта не відображає значимість та ресурсомісткість робіт, а також область їх дії. Побудована діаграма Ґанта наведена на рисунку Б.3.



	Task Name	Duration	Start	Finish
1	**Система підтримки прийняття рішень підбору комплектації комп'ютерної техніки.**	**85 days**	**Mon 20.01.20**	**Fri 15.05.20**
2	**Створення бази даних та бази знань**	**33 days**	**Mon 20.01.20**	**Wed 04.03.20**
3	**Створення бази даних**	**5 days**	**Mon 20.01.20**	**Fri 24.01.20**
4	Створення сутностей (таблиць) для кожної з комплектуючих з необхідними атрибутами	4 days	Mon 20.01.20	Thu 23.01.20
5	Встановлення зв'язків між сутностями	1 day	Fri 24.01.20	Fri 24.01.20
6	Визначення оцінок характеристик комплектуючих	3 days	Mon 27.01.20	Wed 29.01.20
7	**Створення бази знань**	**17 days**	**Thu 30.01.20**	**Fri 21.02.20**
8	Визначення форми подання правил	2 days	Thu 30.01.20	Fri 31.01.20
9	Створення механізму перемикання стану правил	5 days	Mon 03.02.20	Fri 07.02.20
10	Створення механізму редагування оцінок характеристик	5 days	Mon 10.02.20	Fri 14.02.20
11	Створення механізму логічного виводу	5 days	Mon 17.02.20	Fri 21.02.20
12	**Створення системи пояснень**	**8 days**	**Mon 24.02.20**	**Wed 04.03.20**
13	Визначення форми та способу виведення пояснень	3 days	Mon 24.02.20	Wed 26.02.20
14	Співставлення пояснень з відповідними правилами	2 days	Thu 27.02.20	Fri 28.02.20
15	Створення механізму формування пояснення	3 days	Mon 02.03.20	Wed 04.03.20
16	**Розробка користувальницького інтерфейсу**	**15 days**	**Thu 05.03.20**	**Wed 25.03.20**
17	**Створення головного та додаткових вікон програми**	**9 days**	**Thu 05.03.20**	**Tue 17.03.20**
18	Визначення кольорової схем	1 day	Thu 05.03.20	Thu 05.03.20
19	Визначення параметрів керуючих елементів	2 days	Fri 06.03.20	Mon 09.03.20
20	Визначення способу відображення інофрмації про комплектуючі	3 days	Tue 10.03.20	Thu 12.03.20
21	Визначення способу представлення інформації з бази даних	3 days	Fri 13.03.20	Tue 17.03.20
22	**Реалізація логіки переходів**	**6 days**	**Wed 18.03.20**	**Wed 25.03.20**
23	Поява вікон з попередженням/ повідомленням/	3 days	Wed 18.03.20	Fri 20.03.20
24	Переходи між основним та побічними вікнами	3 days	Mon 23.03.20	Wed 25.03.20
25	**Розробка програмних модулей**	**37 days**	**Thu 26.03.20**	**Fri 15.05.20**
26	**Реалізація модуля для роботи з базою даних**	**11 days**	**Thu 26.03.20**	**Thu 09.04.20**
27	Функції для додавання, редагування та видалення записів	3 days	Thu 26.03.20	Mon 30.03.20
28	Функція збереження стану бази даних	2 days	Tue 31.03.20	Wed 01.04.20
29	Функція відновлення стану бази даних	2 days	Thu 02.04.20	Fri 03.04.20
30	Пошук за встановленими фільтрами характеристик	2 days	Mon 06.04.20	Tue 07.04.20
31	Обробка виключних ситуацій в результаті зміни стану бази даних	2 days	Wed 08.04.20	Thu 09.04.20
32	**Реалізація модуля для оновлення бази знань**	**4 days**	**Fri 10.04.20**	**Wed 15.04.20**
33	Редагування оцінок характеристик комплектуючих	1 day	Fri 10.04.20	Fri 10.04.20
34	Зміна стану активності	3 days	Mon 13.04.20	Wed 15.04.20
35	**Реалізація модуля для пошуку комплектуючих**	**12 days**	**Thu 16.04.20**	**Fri 01.05.20**
36	Фільтри за базою даних	2 days	Thu 16.04.20	Fri 17.04.20
37	Підбір ща встановленими фільтрами для характеристик	7 days	Mon 20.04.20	Tue 28.04.20
38	Підбір за обраним режимом підбору	3 days	Wed 29.04.20	Fri 01.05.20
39	**Реалізація модуля для конфігурування збірки**	**10 days**	**Mon 04.05.20**	**Fri 15.05.20**
40	Функції додавання та видалення комплектуючих	3 days	Mon 04.05.20	Wed 06.05.20
41	Розрахунок загальної вдіповідності та сумісності компонентів збірки	2 days	Thu 07.05.20	Fri 08.05.20
42	Встановлення пояснень до загальної оцінки відповідності збірки	3 days	Mon 11.05.20	Wed 13.05.20
43	Встановлення пояснень до сумісності компонентів	1 day	Fri 15.05.20	Fri 15.05.20

Рисунок Б.3 - Діаграма Ґанта

Б.5 Управління ризиками проекту

Управління ризиками – процес реагування на події та зміни ризиків у процесі виконання проекту. Основна стратегія управління ризиками полягає в їх мінімізації. Управління ризиками містить у собі такі етапи:

- планування управління ризиками;
- ідентифікація ризиків проекту;
- якісна і кількісна оцінка ризиків;
- планування реагування на виявлені значимі ризики;
- моніторинг і контроль ризиків.

Ідентифікація ризиків – визначення ризиків, які можуть вплинути на проект, і документування їх характеристик. Це ітеративний процес, оскільки в міру розвитку проекту в рамках його життєвого циклу можуть бути виявлені нові ризики.

Тепер, складемо перелік ризиків, їх відносну критичність та можливі способи уникнення цих ризиків у Risk Register – інструмент для документування ризиків и дій по управлінню кожним з них, але спочатку, встановимо шкалу рівнів критичності ризиків (табл. Б.2):

Таблиця Б.2 – Рівні критичності ризиків

1	Незначний
2	Низький
3	Середній
4	Високий
5	Критичний

Визначені ризики проекту представлені у табл. Б.3.

Таблиця Б.3 – Risk Register

№	Опис	Вплив	Ймовірність	Серйозність	Пом’якшення
1	Проект визначений нечітко	4	2	M	Забезпечити чітке визначення мети в статуті проекту
2	Низьке залучення замовника	2	3	M	Пояснити, що чим більше замовник залучений тим кращій продукт він матиме
3	Графік робіт проекту визначений не чітко	3	3	M	Встановити пункт планування робіт на початковій стадії проекту
4	Втрачена актуальність проекту	4	2	M	Аналіз актуальності на початкових стадіях проекту
5	Різка зміна вимог	4	1	M	Зауваження, що вимоги повинні визначатись на початку і, якщо корегуватись, то незначною мірою

Продовження таблиці Б.3

№	Опис	Вплив	Ймовірність	Серйозність	Пом’якшення
6	Затримка на початкових етапах, що зміщує графік наступних робіт	4	3	Н	Переконатись, що графік виконання робіт визначений чітко
7	Незапланована робота, що повинна бути виконана	4	2	М	Задокументувати всі припущення про можливі додаткові роботи при плануванні робіт
8	Вимушене зменшення тривалості виконання певних завдань, або їх незаплановане паралельне виконання	3	2	М	Зауваження, що при виконанні робіт з необґрунтовано високою інтенсивністю або одночасно може привести до погіршення якості продукту
9	Важкі або нездійснені задачі	5	3	Н	Замінити або видалити такі задачі

Продовження таблиці Б.3

№	Опис	Вплив	Ймовірність	Серйозність	Пом’якшення
10	Поломка / Відсутність необхідного обладнання	5	1	M	Передбачення резервного обладнання/Передбачення коштів на закупівлю обладнання
11	Непрацездат-ність розробника	5	1	M	Запобігання екстремальних ситуацій

Далі, наведено матрицю залежності величини втрат від ймовірності виникнення певного ризику (табл. Б.4).

Таблиця Б.4 – Probability / Impact Matrix

Probability\Impact	1	2	3	4	5
1				5	10,11
2			8	1,4,7	
3		2	3	6	9
4					
5					

ДОДАТОК В

ДІАГРАМИ ДІЯЛЬНОСТІ

На рисунках В.1-В.15 наведено діаграми діяльнотсі для кожного з варіантів використання (ВВ) створеної діаграми прецедентів.

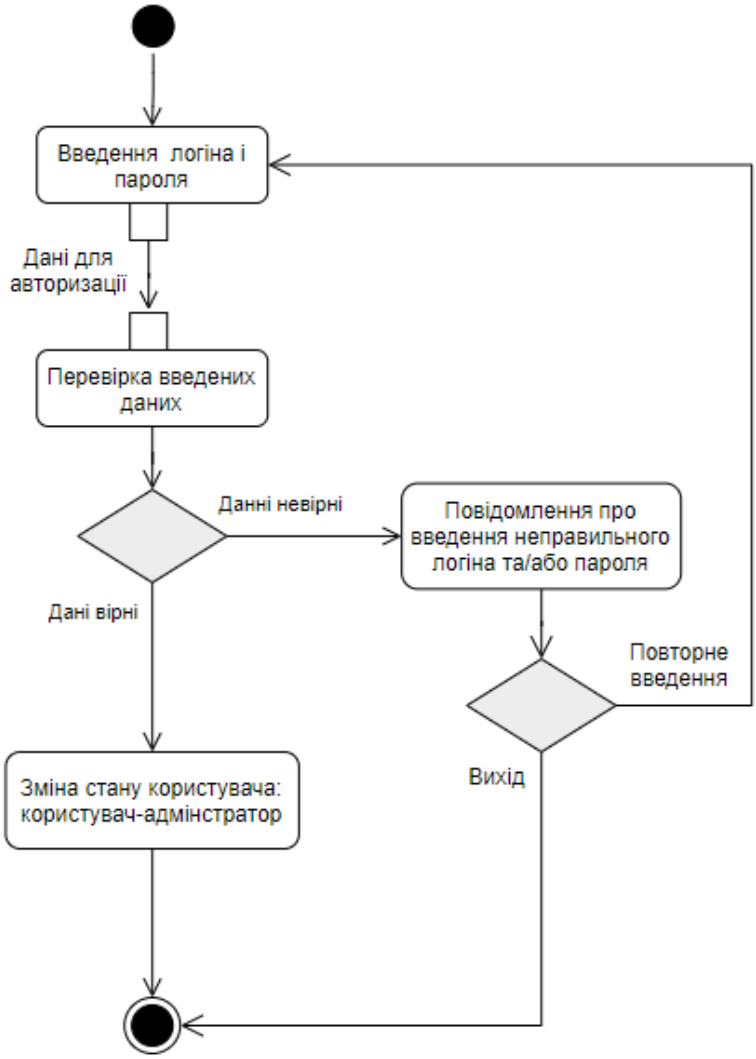


Рисунок В.1 – Діаграма діяльності для ВВ «Авторизація»

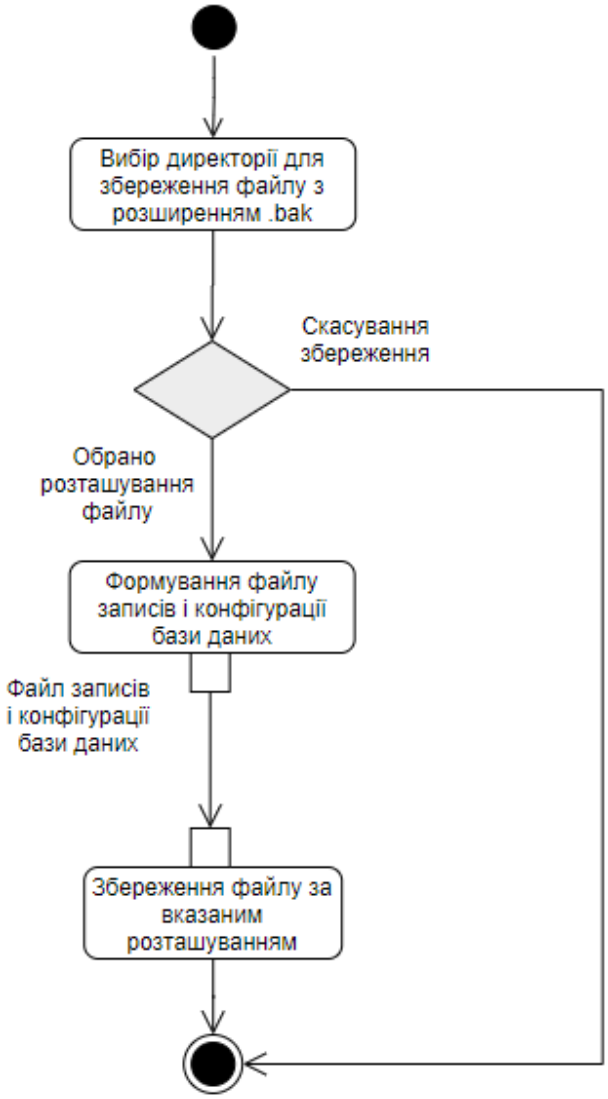


Рисунок В.2 – Діаграма діяльності для ВВ «Створення резервної копії»

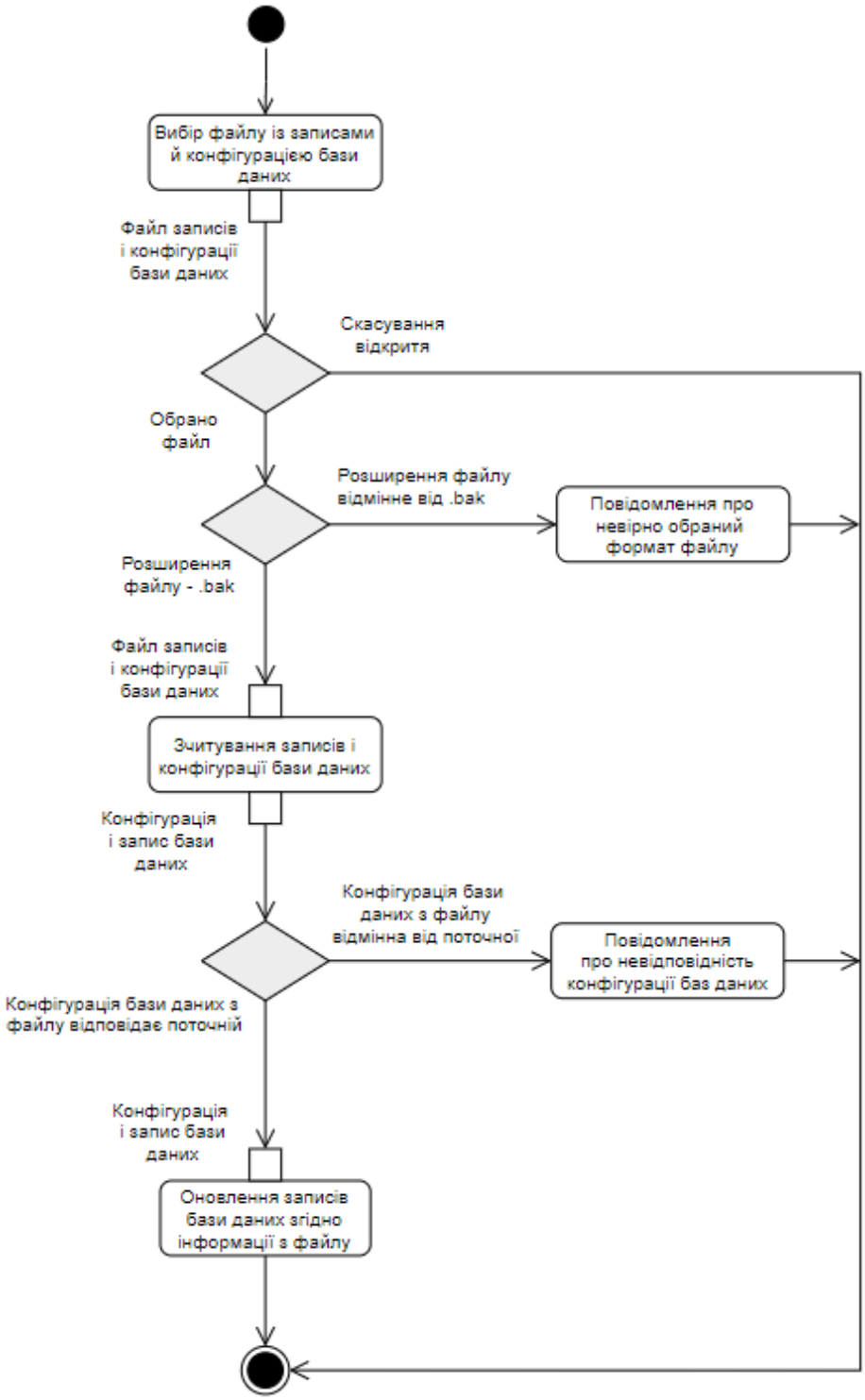


Рисунок В.3 – Діаграма діяльності для ВВ «Відновлення за резервною копією»

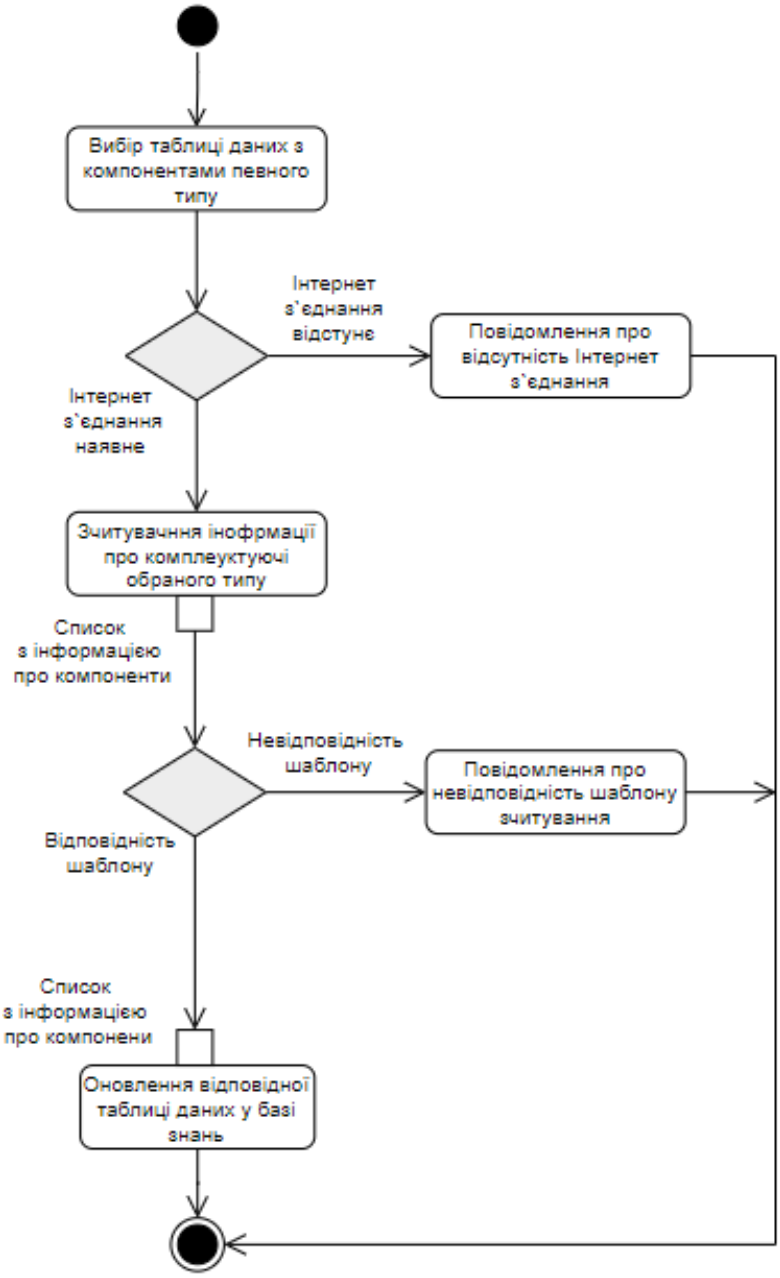


Рисунок В.4 – Діаграма діяльності для ВВ «Оновлення інформації про компоненти»

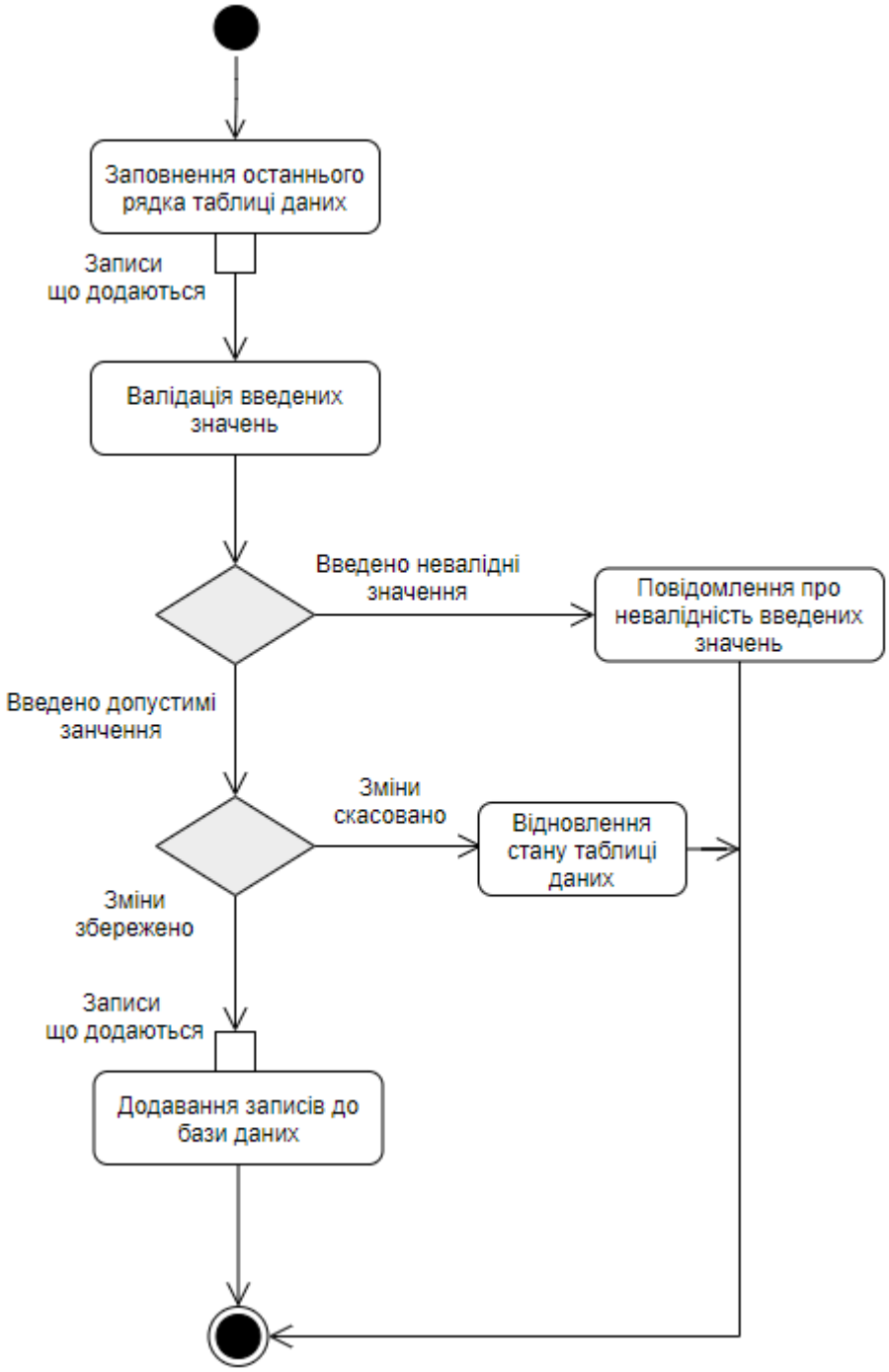


Рисунок В.5 – Діаграма діяльності для ВВ «Додавання записів до бази даних»

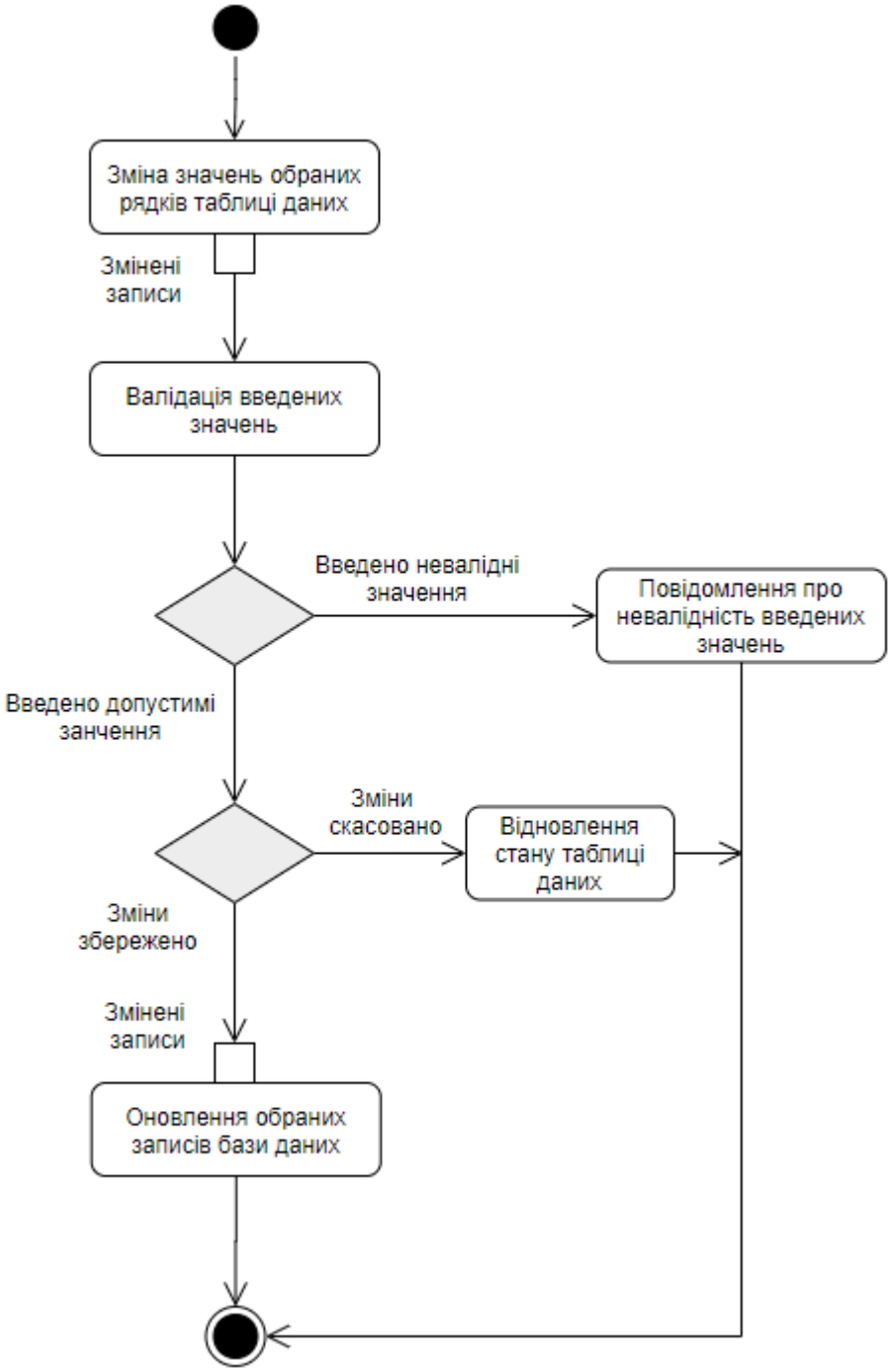


Рисунок В.6 – Діаграма діяльності для ВВ «Редагування записів бази даних»

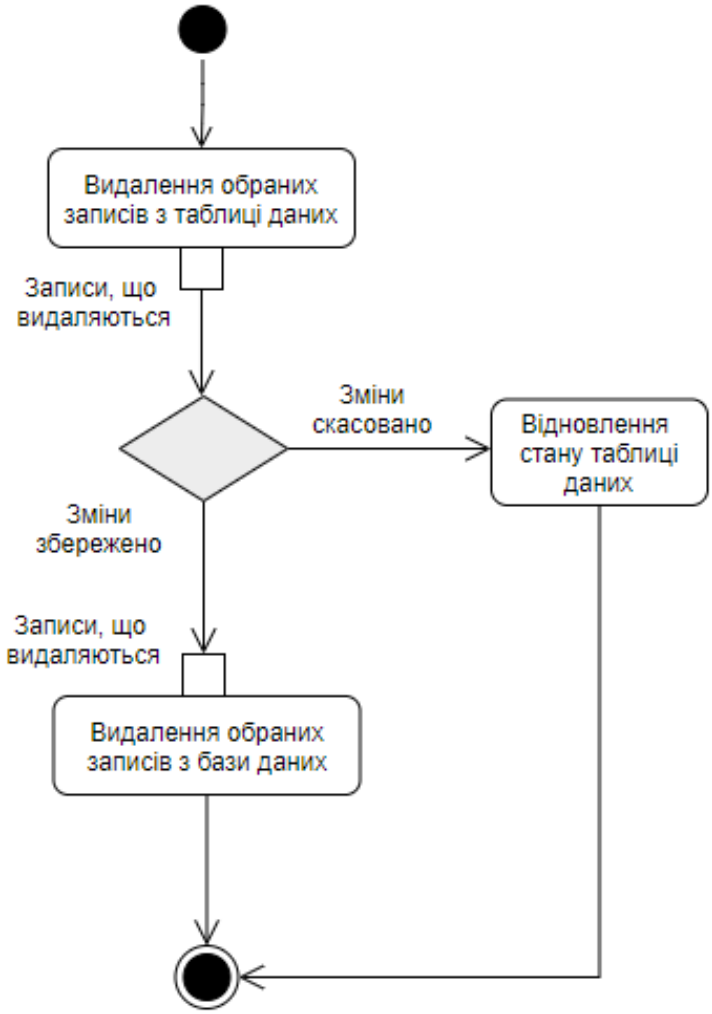


Рисунок В.7 – Діаграма діяльності для ВВ «Видалення записів з бази даних»

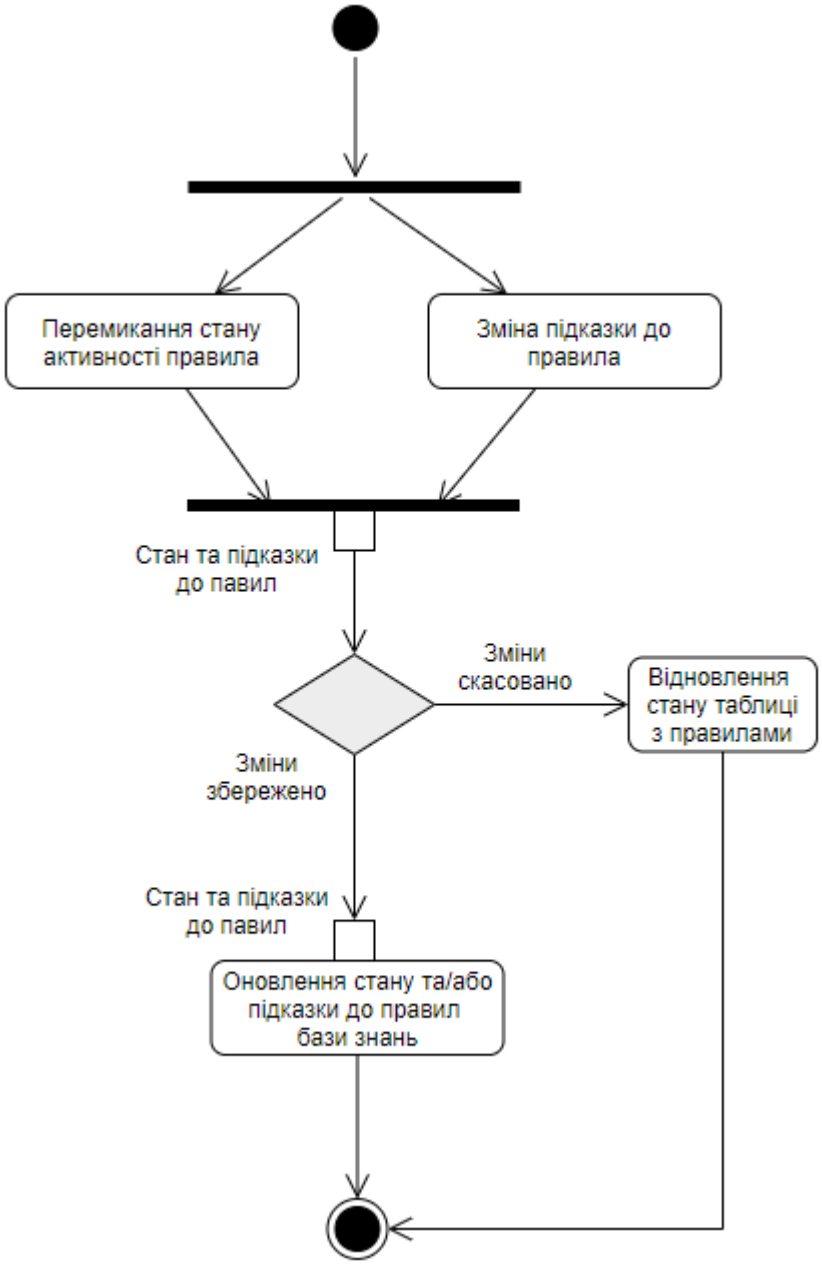


Рисунок В.8 – Діаграма діяльності для ВВ «Зміна активності та підказок до правил»

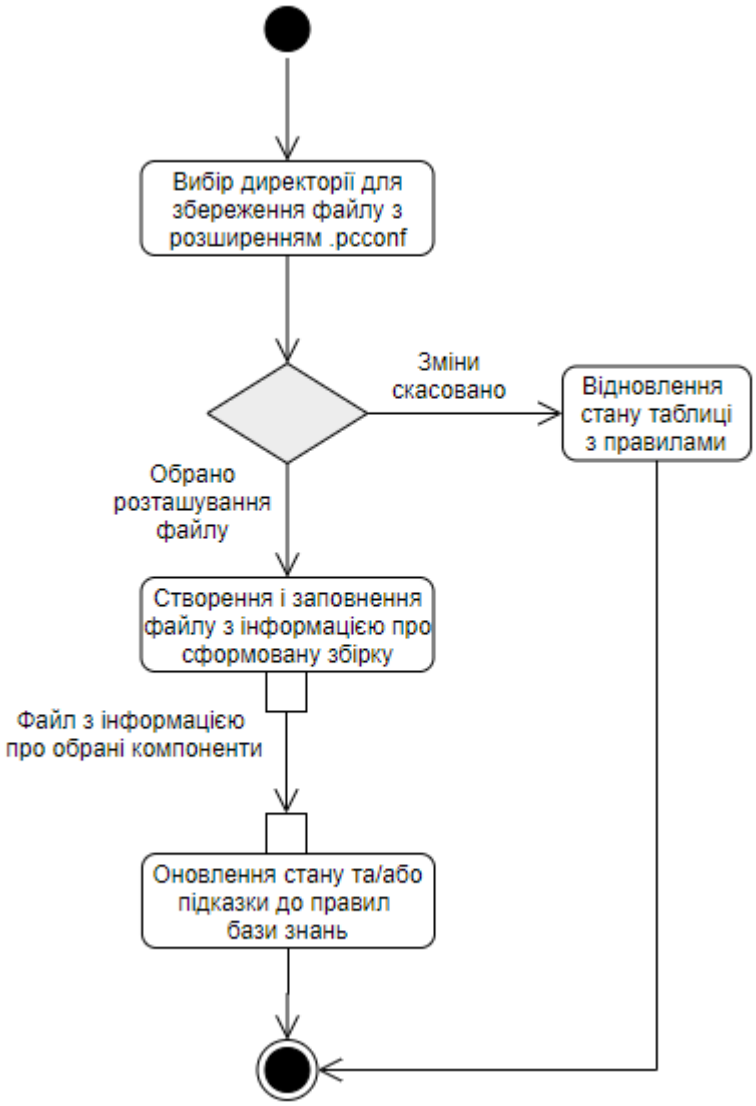


Рисунок В.9 – Діаграма діяльності для ВВ «Збереження збірки»

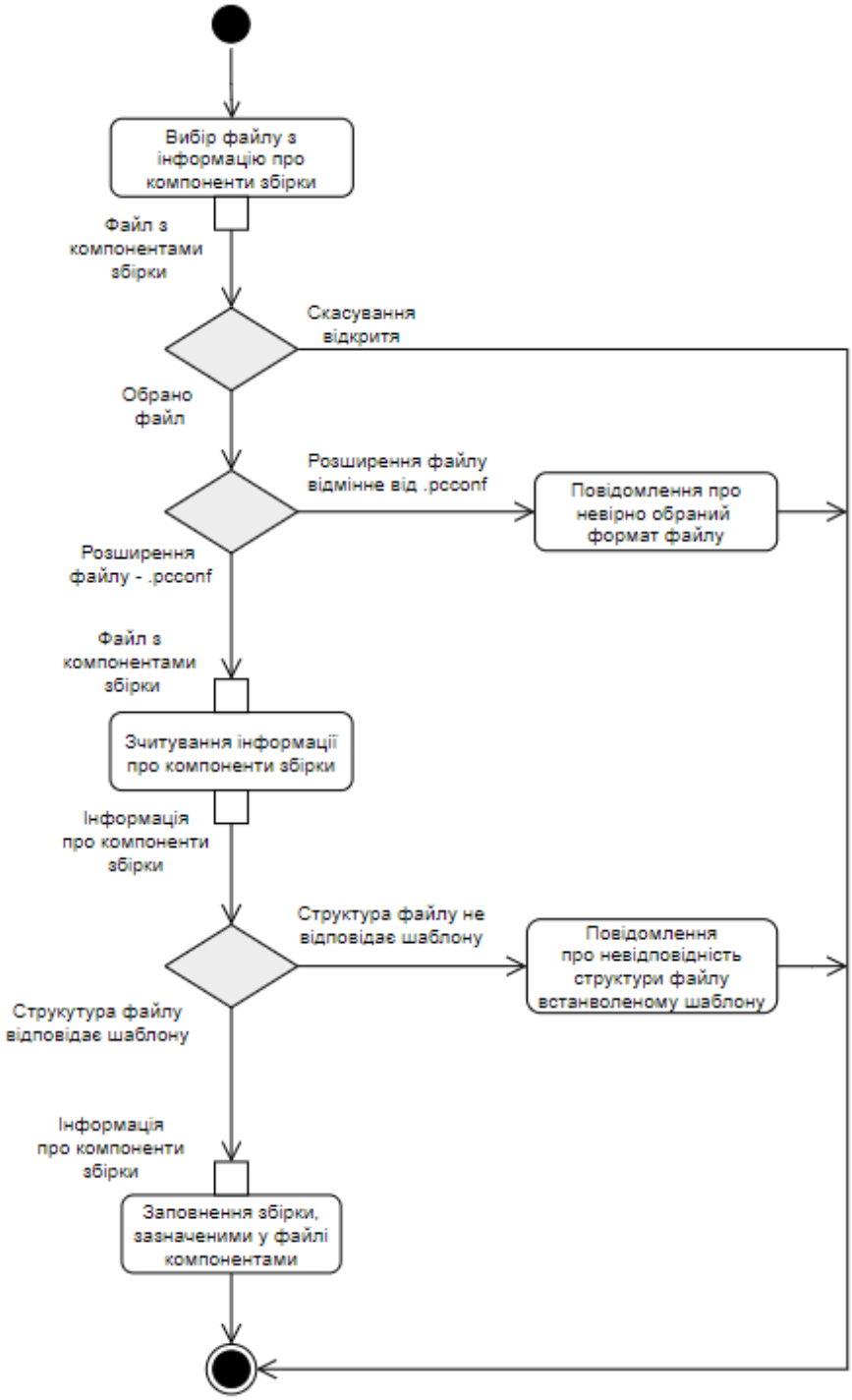


Рисунок В.10 – Діаграма діяльності для ВВ «Завантаження збірки»

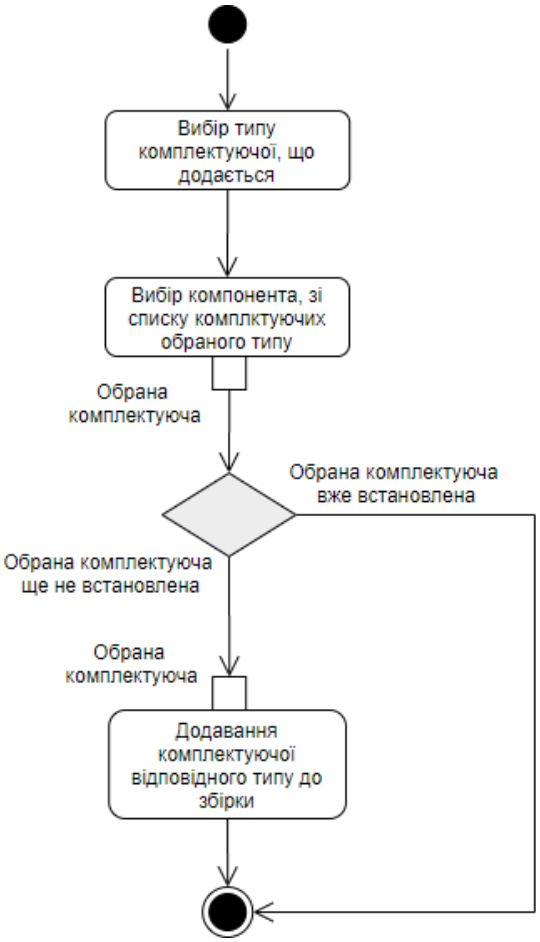


Рисунок В.11 – Діаграма діяльності для ВВ «Додавання комплектуючої до збірки»

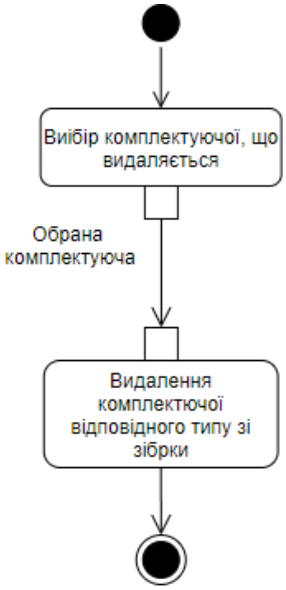


Рисунок В.12 – Діаграма діяльності для ВВ «Видалення комплектуючої зі збірки»

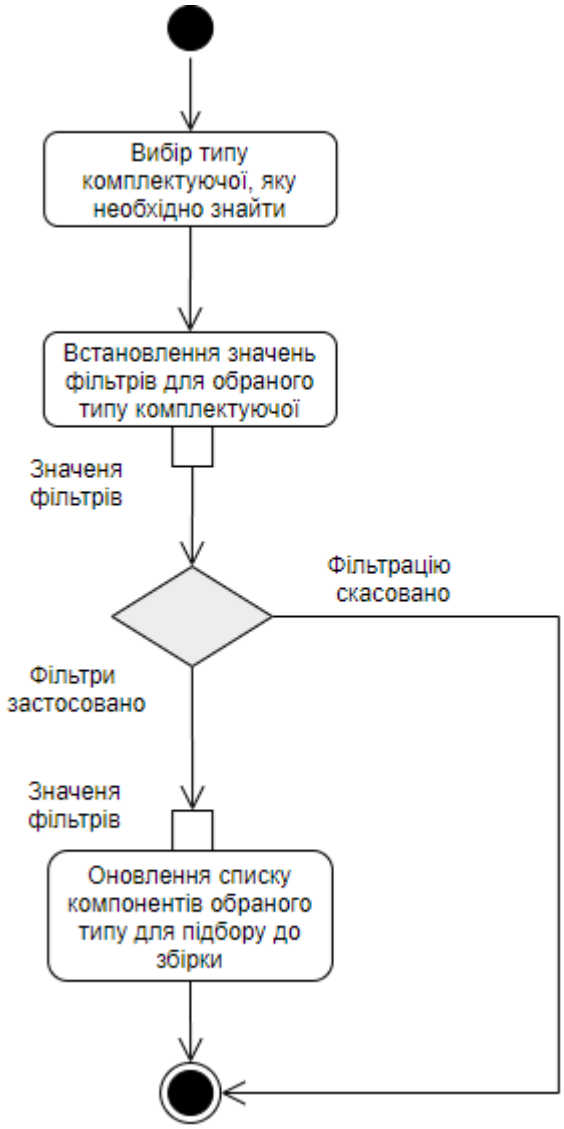


Рисунок В.13 – Діаграма діяльності для ВВ «Фільтрування компонентів»

Рисунок В.14 – Діаграма діяльності для ВВ «Визначення оцінки відповідності»

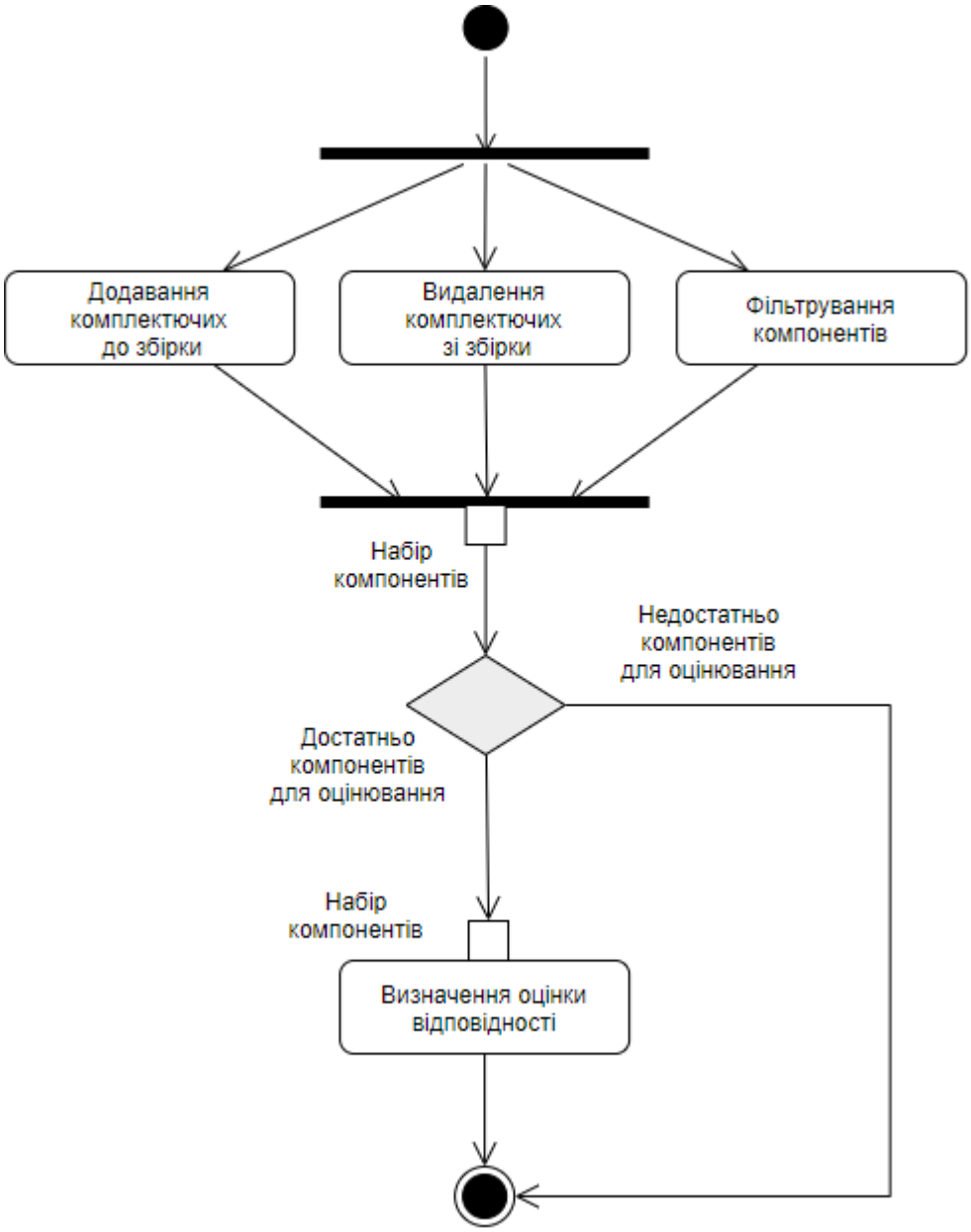


Рисунок В.15 – Діаграма діяльності для ВВ «Формування збірки»

ДОДАТОК Г

ЛІСТИНГ ФАЙЛІВ ПРОГРАМНОГО КОДУ

Лістинг файлів моделі:

CompatibilityProvider.cs.

```
using System;
using System.Linq;
using System.Globalization;
using System.Text.RegularExpressions;

namespace PCConfigurator.Model
{
   internal static class CompatibilityProvider
   {
      // Все правила совместимости (массив делегатов).
      public static Func<Component, Component, Boolean>[] Rules =
      {
         IsCPUandMB, IsGPUandMB, IsRAMandMB, IsBodyandMB,
         IsGPUandPowerSupply, IsGPUandBody, IsCoolerandCPU, IsCoolerandBody
      };
      // Проверка совместимости процессора и материнской платы.
      private static Boolean IsCPUandMB(this Component motherBoard, Component cpu)
      {
         return ((MotherBoard)motherBoard).ConnectorType.Contains(((CPU)cpu).ConnectorType);
      }
      // Проверка совместимости видеокарты и материнской платы.
      private static Boolean IsGPUandMB(this Component motherBoard, Component gpu)
      {
         Boolean isCompatible = false;
         String[] interfaces = Regex.Split(((MotherBoard)motherBoard).GPUInterface, @"\d\s[xx]\s");
         GroupCollection groupMB;
         GroupCollection groupPS =
            Regex.Match(((GPU)gpu).Interface, @"PCI-Express\s?(?:[xx](\d+)\s?)?(\d+).?\w?").Groups;
         for (Int32 i = 1; i < interfaces.Length; i++)
         {
            groupMB = Regex.Match(interfaces[i], @"PCI-E\s?(\d+).?\w?\s?[xx](\d+)").Groups;

            if (groupMB[1].Value == groupPS[2].Value && groupMB[2].Value == groupPS[1].Value)
```

```
            {
                isCompatible = true;
                break;
            }

        }

        return isCompatible;
    }
    // Проверка совместимости оперативной памяти и материнской платы.
    private static Boolean IsRAMandMB(this Component motherBoard, Component ram)
    {
        Boolean isCompatible = true;
        GroupCollection regexGroup = Regex.Match(((MotherBoard)motherBoard).MemSupported,
@"(\d+)?\s?x\s?(DDR\d?)").Groups;
        if (((MotherBoard)motherBoard).MaxRAMSize < ((RAM)ram).MemVolume)
            isCompatible = false;
        if(regexGroup.Count == 3)
            if(Int16.Parse(regexGroup[1].Value) < ((RAM)ram).NumOfStrips ||
!((RAM)ram).MemType.Contains(regexGroup[2].Value))
                isCompatible = false;
        else if(regexGroup.Count == 2)
            if(!((RAM)ram).MemType.Contains(regexGroup[1].Value))
                isCompatible = false;
        return isCompatible;
    }
    // Проверка совместимости корпуса и материнской платы.
    private static Boolean IsBodyandMB(this Component motherBoard, Component body)
    {
        return ((Body)body).MotherBoardFormFactor.ToLower().Split('
').Contains(((MotherBoard)motherBoard).FormFactor.ToLower());
    }
    // Проверка совместимости видеокарты и блока питания.
    private static Boolean IsGPUandPowerSupply(this Component gpu, Component powerSupply)
    {
        return ((GPU)gpu).NeedPower <= ((PowerSupply)powerSupply).Power;
    }
    // Проверка совместимости видеокарты и корпуса.
    private static Boolean IsGPUandBody(this Component gpu, Component body)
    {
        GroupCollection groups = Regex.Match(((GPU)gpu).Sizes, @"(\d+.?(?:\d+)?)\s?[xx]?\s?(\d+.?(?:\d+)?)").Groups;
```

```
            return ((Body)body).MaxGPUWidth > Decimal.Parse(groups[2].Value != "" ? groups[2].Value : groups[1].Value,
CultureInfo.InvariantCulture);
        }
        // Проверка совместимости кулера и процессора.
        private static Boolean IsCoolerandCPU(this Component cooler, Component cpu)
        {
            return Regex.IsMatch(((Cooler)cooler).Compatability, ((CPU)cpu).ConnectorType.Substring("Socket".Length + 1),
RegexOptions.IgnoreCase);
        }

        // Проверка совместимости кулера и корпуса.
        private static Boolean IsCoolerandBody(this Component cooler, Component body)
        {
            GroupCollection groups = Regex.Match(((Cooler)cooler).Sizes,
@"(?:\d+.?(?:\d+)?\s?[xxXX]\s?){2}(\d+.?(?:\d+)?)").Groups;

            return ((Body)body).MaxCoolerHeight > Decimal.Parse(groups[1].Value, CultureInfo.InvariantCulture);
        }

        // Проверка выбран ли компонент зборки.
        public static Boolean IsTaken(this Component component)
        {
            return component.Title != "None";
        }
    }
}
```

ConvertProvider.cs.

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Text.RegularExpressions;

namespace PCConfigurator.Model
{
    internal static class ConvertProvider
    {
        // Выбор и переход к конвертации определённого типа копмлекутующих.
        public static dynamic ConvertTo(String componentName, List<String> inList, Uri uri)
        {
            switch (componentName)
            {
```

```
            case "MotherBoard":
                return ConvertToMotherBoard(inList, uri);
            case "CPU":
                return ConvertToCPU(inList, uri);
            case "GPU":
                return ConvertToGPU(inList, uri);
            case "RAM":
                return ConvertToRAM(inList, uri);
            case "PowerSupply":
                return ConvertToPowerSupply(inList, uri);
            case "Cooler":
                return ConvertToCooler(inList, uri);
            case "HDD":
                return ConvertToHDD(inList, uri);
            case "SSD":
                return ConvertToSSD(inList, uri);
            case "Body":
                return ConvertToBody(inList, uri);
            default:
                return null;
        }
    }
    // Конвертация цены.
    private static Int32 GetCost(String cost)
    {
        GroupCollection groups = Regex.Match(cost, @"(?:\s?(\d+) (\d+))|(?:\s?(\d{2,3}|0))").Groups;

        return groups[3].Value == "" ? Int32.Parse(groups[1].Value) * 1000 + Int32.Parse(groups[2].Value) :
Int32.Parse(groups[3].Value);
    }
    // Конвертация максимального объёма оперативной памяти материнской платы.
    private static Int16 GetMaxRAMSize(String str)
    {
        GroupCollection groups = Regex.Match(str, @"(?:(\d+)\s?(?:ГБ|Гб))|(?:(\d+)\s(?:ТБ|Тб))").Groups;

        return (Int16)(groups[2].Value == "" ? Int16.Parse(groups[1].Value) : Int16.Parse(groups[2].Value) * 1000);
    }

    // Конвертация размеров материнской платы.
    private static String GetMBSizes(String str)
    {
        GroupCollection groups = Regex.Match(str, @"(\d+.?(?:\d+)?\s?[x|x]\s?\d+.?(?:\d+)?)").Groups;
```

```
            return groups[1].Value;
        }
        // Конвертация харакетристик матринской платы.
        private static MotherBoard ConvertToMotherBoard(List<String> inList, Uri uri)
        {
            return new MotherBoard
            {
                Title = inList[0].Substring(" Материнская плата".Length).Trim(),
                ConnectorType = inList[1],
                ChipSet = inList[2],
                FormFactor = inList[3].Contains("MiniITX") ? "Mini-ITX" : inList[3],
                MemSupported = inList[4],
                CPUsSupported = inList[5],
                MaxRAMSize = inList[6] != "None" ? GetMaxRAMSize(inList[6]) : (Int16)0,
                GPUInterface = inList[7],
                PowerConnector = inList[8],
                Sizes = inList[9] != "None" ? GetMBSizes(inList[9]) : "None",
                Cost = GetCost(inList[10]),
                Image = ImageProvider.GetNewImageByUri(uri)
            };
        }
        // Конвертация характеристик процессора.
        private static CPU ConvertToCPU(List<String> inList, Uri uri)
        {
            return new CPU
            {
                Title = inList[0].Substring(" Процессор".Length).Trim(),
                ConnectorType = inList[1],
                NumofCores = inList[2] != "None" ? Int16.Parse(inList[2]) : (Int16)0,
                InternalFrequency = inList[3] != "None" ? Int16.Parse(Regex.Match(inList[3], @"(\d+)\s?МГц").Groups[1].Value) :
(Int16)0,
                MaxFrequency = inList[4] != "None" ? Decimal.Parse(Regex.Match(inList[4],
@"(\d\.?\d*)\s?ГГц").Groups[1].Value, CultureInfo.InvariantCulture) : 0m,
                NumOfThreads = inList[5] != "None" ? Int16.Parse(inList[5]) : (Int16)0,
                Unlocked = inList[6] == "С разблокированным множителем" ? true : false,
                PowerTDP = inList[7] != "None" ? Int16.Parse(Regex.Match(inList[7], @"(\d+)\s?Вт").Groups[1].Value) : (Int16)0,
                CacheMemoryLvl3 = (inList[8] != "None" && inList[8] != "Нет") ? Decimal.Parse(Regex.Match(inList[8],
@"(\d+\.?\d*)\s?МБ").Groups[1].Value, CultureInfo.InvariantCulture) : 0m,
                Cost = GetCost(inList[9]),
                Image = ImageProvider.GetNewImageByUri(uri)
            };
```

```
    }
    // Конвертация частоты памяти видеокарты.
    private static Int32 GetMemFrequency(String str)
    {
        GroupCollection groups = Regex.Match(str, @"(?:(\d+)\s?(?:МГц|МГц|Гц))|(?:(\d+)\sГГц)").Groups;

        return groups[2].Value == "" ? Int32.Parse(groups[1].Value) : (Int32.Parse(groups[2].Value) * 1000);
    }
    // Конвертация объёма памяти видеокарты.
    private static Int16 GetMemVolume(String str)
    {
        GroupCollection groups = Regex.Match(str, @"(?:(\d+)\s?ГБ)|(?:(\d+)\sМБ)").Groups;

        return groups[2].Value == "" ? (Int16)(Int16.Parse(groups[1].Value) * 1000) : Int16.Parse(groups[2].Value);
    }
    // Конвертация размеров видеокарты.
    private static String GetGPUSizes(String str)
    {
        GroupCollection groups = Regex.Match(str,

@"((?:\d+.?(?:\d+)?\s?[xx]\s?){2}\d+.?(?:\d+)?)|(\d+.?(?:\d+)?\s?[xx]\s?\d+.?(?:\d+)?)|(?:(?:[дД]лина\s)?(\d+.?(?:\d+)?))").Gro
ups;

        return groups[1].Value != "" ? groups[1].Value : groups[2].Value != "" ? groups[2].Value : groups[3].Value;
    }
    // Конвертация характеристик видеокарты.
    private static GPU ConvertToGPU(List<String> inList, Uri uri)
    {
        return new GPU
        {
            Title = inList[0].Trim(),
            MemFrequency = inList[1] != "None" ? GetMemFrequency(inList[1]) : 0,
            MemVolume = inList[2] != "None" ? GetMemVolume(inList[2]) : (Int16)0,
            CoreFrequency = inList[3] != "None" ? Int16.Parse(Regex.Match(inList[3],
@"(\d+)\s?(?:МГц|МГц|Мгц)").Groups[1].Value) : (Int16)0,
            MemBus = inList[4] != "None" ? Int16.Parse(Regex.Match(inList[4], @"(\d+)\s?бит").Groups[1].Value) : (Int16)0,
            MemType = inList[5],
            NeedPower = inList[6] != "None" ? Int16.Parse(Regex.Match(inList[6], @"(\d+)\s?\w+").Groups[1].Value) :
(Int16)0,
            Interface = inList[7],
            Sizes = inList[8] != "None" ? GetGPUSizes(inList[8]) : "None",
            Cost = GetCost(inList[9]),
```

```
            Image = ImageProvider.GetNewImageByUri(uri)
        };
    }
    // Конвертация характеристик оперативной памяти.
    private static RAM ConvertToRAM(List<String> inList, Uri uri)
    {
        return new RAM
        {
            Title = inList[0].Substring(" Оперативная память".Length).Trim(),
            MemVolume = inList[1] != "None" ? Int16.Parse(Regex.Match(inList[1], @"(\d+)\s?(?:ГБ|Гб)").Groups[1].Value) :
(Int16)0,
            MemType = inList[2],
            Frequency = inList[3] != "None" ? Int16.Parse(Regex.Match(inList[3], @"(\d+)\s?МГц").Groups[1].Value) :
(Int16)0,
            EffectiveBandwidth = inList[4] != "None" ? Int32.Parse(Regex.Match(inList[4], @"(\d{2,})\s?").Groups[1].Value) :
(Int16)0,
            NumOfStrips = inList[5] != "None" ? Int16.Parse(inList[5]) : (Int16)0,
            TimingScheme = inList[6] != "None" ? Regex.Match(inList[6], @"((?:CL)?\s?\d{1,2}(?:(?:-\d{1,2}-
\d{1,2})|\z))").Groups[1].Value : "None",
            Cost = GetCost(inList[7]),
            Image = ImageProvider.GetNewImageByUri(uri)
        };
    }

    // Конвертация названия блока питания, твердотельного накопителя или корпуса.
    private static String GetTitle(String str)
    {
        Int32 index = str.IndexOf(" + ");

        if (index != -1)
            return str.Substring(0, index);
        return str;
    }
    // Конвертация характеристик блока питания.
    private static PowerSupply ConvertToPowerSupply(List<String> inList, Uri uri)
    {
        return new PowerSupply
        {
            Title = GetTitle(inList[0]).Trim(),
            Power = inList[1] != "None" ? Int16.Parse(Regex.Match(inList[1], @"(\d+)\s?Вт").Groups[1].Value) : (Int16)0,
            MotherBoardConnector = inList[2],
            CPUConnector = inList[3],
```

```
            GPUConnector = inList[4],
            NumOfSATA = inList[5] != "None" ? Int16.Parse(inList[5]) : (Int16)0,
            Cost = GetCost(inList[6]),
            Image = ImageProvider.GetNewImageByUri(uri)
        };
    }
    // Конвертация размеров кулера.
    private static String GetCoolerSizes(String str)
    {
        GroupCollection groups = Regex.Match(str,
            @"((?:\d+.?(?:\d+)?\s?[xxXX]\s?){2}\d+.?(?:\d+)?)").Groups;
        return groups[1].Value;
    }
    // Конвертация характеристик кулера.
    private static Cooler ConvertToCooler(List<String> inList, Uri uri)
    {
        return new Cooler
        {
            Title = inList[0].Substring(" Кулер".Length).Trim(),
            Compatability = inList[1],
            RotationFrequency = inList[2] != "None" ? Int16.Parse(Regex.Match(inList[2],
@"(\d{3,4})\s?(?:об/мин)?").Groups[1].Value) : (Int16)0,
            Sizes = inList[3] != "None" ? GetCoolerSizes(inList[3]) : "None",
            Cost = GetCost(inList[4]),
            Image = ImageProvider.GetNewImageByUri(uri)
        };
    }
    // Конвертация объёма жёсткого диска или твердотельного накопителя.
    private static Int16 GetVolume(String str)
    {
        GroupCollection groups = Regex.Match(str, @"(?:(\d+)\s?ТБ)|(?:(\d+)\s?ГБ)").Groups;

        return groups[2].Value == "" ? (Int16)(Int16.Parse(groups[1].Value) * 1000) : Int16.Parse(groups[2].Value);
    }
    // Конвертация скорости передачи данных жёсткого диска.
    private static Int16 GetTransferSpeed(String str)
    {
        GroupCollection groups = Regex.Match(str,
@"(?:(\d+)\s?M\w?(?:Б|бит)/)|(?:(\d+)(?: |\s)?Г(?:бит|б)\s?/\s?c)").Groups;

        return groups[2].Value == "" ? Int16.Parse(groups[1].Value) : (Int16)(Int16.Parse(groups[2].Value) * 134);
    }
```

```
// Конвертация характеристик жёсткого диска.
private static HDD ConvertToHDD(List<String> inList, Uri uri)
{
   return new HDD
   {
      Title = inList[0].Substring(" Жесткий диск".Length).Trim(),
      FormFactor = inList[1],
      Volume = inList[2] != "None" ? GetVolume(inList[2]) : (Int16)0,
      Interface = inList[3],
      RotationSpeed = inList[4] != "None" ? Int16.Parse(Regex.Match(inList[4],
@"(\d{3,5})\s?об/мин").Groups[1].Value) : (Int16)0,
      BufferVolume = inList[5] != "None" ? Int16.Parse(Regex.Match(inList[5], @"(\d+)\s?МБ").Groups[1].Value) :
(Int16)0,
      DataTransferSpeed = inList[6] != "None" ? GetTransferSpeed(inList[6]) : (Int16)0,
      Cost = GetCost(inList[7]),
      Image = ImageProvider.GetNewImageByUri(uri)
   };
}
// Конвертация характеристик твердотельного накопителя.
private static SSD ConvertToSSD(List<String> inList, Uri uri)
{
   return new SSD
   {
      Title = GetTitle(inList[0]).Trim(),
      Volume = inList[1] != "None" ? GetVolume(inList[1]) : (Int16)0,
      ReadSpeed = inList[2] != "None" ? Int16.Parse(Regex.Match(inList[2], @"(\d+)\s?").Groups[1].Value) : (Int16)0,
      WriteSpeed = inList[3] != "None" ? Int16.Parse(Regex.Match(inList[3], @"(\d+)\s?").Groups[1].Value) : (Int16)0,
      FormFactor = inList[4],
      Interface = inList[5],
      Cost = GetCost(inList[6]),
      Image = ImageProvider.GetNewImageByUri(uri)
   };
}
// Конвертация размеров корпуса.
private static String GetBodySizes(String str)
{
   GroupCollection groups = Regex.Match(str,
      @"(\d+.?(?:\d+)?\s?[xx]\s?\d+.?(?:\d+)?\s?[xx]\s?\d+.?(?:\d+)?)" +

@"|(?:(?:Ш|ширина):\s?(\d+.?(?:\d+)?)\s?(?:мм,)?s\?(?:В|высота):\s?(\d+.?(?:\d+)?)\s?(?:мм,)?s\?(?:Г|глубина):\s?(\d+.?(?:\d
+)?)(?:мм,)?s\?)").Groups;
   return groups[1].Value != "" ? groups[1].Value : groups[2].Value + "x" + groups[3].Value + "x" + groups[4].Value;
```

```
        }
        // Конвертация характеристик корпуса.
        private static Body ConvertToBody(List<String> inList, Uri uri)
        {
            return new Body
            {
                Title = GetTitle(inList[0].Substring(" Корпус".Length)).Trim(),
                MotherBoardFormFactor = inList[1],
                NumOf3dt5Cells = inList[2] != "None" ? Int16.Parse(Regex.Match(inList[2], @"(\d+)").Groups[1].Value) :
(Int16)0,
                NumOf2dt5Cells = inList[3] != "None" ? Int16.Parse(Regex.Match(inList[3], @"(\d+)").Groups[1].Value) :
(Int16)0,
                MaxGPUWidth = inList[4] != "None" ? Decimal.Parse(Regex.Match(inList[4],
@"(\d\.?\d*)\s?\w*").Groups[1].Value, CultureInfo.InvariantCulture) : 0m,
                MaxCoolerHeight = inList[5] != "None" ? Decimal.Parse(Regex.Match(inList[5],
@"(\d\.?\d*)\s?\w*").Groups[1].Value,
                CultureInfo.InvariantCulture) : 0m,
                Sizes = inList[6] != "None" ? GetBodySizes(inList[6]) : "None",
                Cost = GetCost(inList[7]),
                Image = ImageProvider.GetNewImageByUri(uri)
            };
        }
    }
}
```

DataBaseProvider.cs.

```
using System;
using System.Data.Entity;
using System.Collections.Generic;

namespace PCConfigurator.Model
{
    internal static class DataBaseProvider
    {
        // Контекст базы данных (конфигурация и данные).
        public static ComponentsDataBaseEntities Context { get; private set; }

        static DataBaseProvider()
        {
            Context = new ComponentsDataBaseEntities();
        }
        // Удаление контекста.
```

```
public static void ConetextDispose()
{
    if (Context != null)
        Context.Dispose();
}
// Обновление контекста.
public static void ConetextReNew()
{
    ConetextDispose();
    Context = new ComponentsDataBaseEntities();
}
// Загрузка данных из репозитория определённого типа комплеткующих.
public static ICollection<TEntity> GetCollection<TEntity>(this DbSet<TEntity> entity)
    where TEntity : class
{
    entity.Load();
    return entity.Local.ToBindingList();
}
// Удаления данных из репозитория определённого типа.
public static void Clear<TEntity>(this DbSet<TEntity> entity)
    where TEntity : class
{
    entity.RemoveRange(entity);
}
// Сохранение изменений контекста.
public static void SaveChanges()
{
    Context.SaveChanges();

    ConetextReNew();
}
// Создание резервной копии контекста.
public static void Backup(String path)
{
    Context.Database.ExecuteSqlCommand(TransactionalBehavior.DoNotEnsureTransaction,
    @"EXEC [dbo].[BackupAll] @path = '" + path + "'");
}
// Восстановление контекста по резервной копии.
public static void Restore(String path)
{
    Context.Database.ExecuteSqlCommand(TransactionalBehavior.DoNotEnsureTransaction,
      @"EXEC [dbo].[RestoreAll] @path = '" + path + "'");
```

```
            ConetextReNew();
        }
    }
}
```

FileProvider.cs.

```
using System;
using System.IO;
using System.Text;
using System.Collections.Generic;

namespace PCConfigurator.Model
{
    internal static class FileProvider
    {
        // Заголовок файла со сборкой.
        private const String Header = "#&#Assembly components INFO for PCConfigurator app#&#";

        // Сохранение компонентов сборки в файл.
        public static void SaveAssemblyFile(List<Component> components, String filePath)
        {
            String infoToWrite = Header;
            foreach (Component component in components)
                infoToWrite += Environment.NewLine + component.GetType().Name + ": " + component.Title;

            using (StreamWriter streamWriter = new StreamWriter(filePath, false, Encoding.Default))
            {
                streamWriter.Write(infoToWrite);
            }
        }
        // Загрузка компонентов сборки из файла.
        public static String[] LoadAssemblyFile(String filePath, Int32 amount)
        {
            String tempStr = "";
            String[] titles = new String[amount];

            using (StreamReader streamReader = new StreamReader(filePath, Encoding.Default))
            {
                if (streamReader.ReadLine() != Header)
                    throw new FileFormatException("Wrong file format.");

                for (Int32 i = 0; i < titles.Length; i++)
```

```
            {
                tempStr = streamReader.ReadLine();
                titles[i] = (tempStr.Substring(tempStr.IndexOf(':') + 1).Trim());
            }
        }
        return titles;
    }
  }
}
```

HTMLParser.cs.

```
using System;
using System.Linq;
using System.Collections.Generic;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using HtmlAgilityPack;

namespace PCConfigurator.Model
{
  internal static class HTMLParser
  {
    #region Parsing static constants

    // Параметр для перехода на страницу с характеристиками комплектующей.
    private const String INNERPARAM = "characteristics/";
    // Значения фильтров для каждого из типов комплектующих.
    private readonly static Dictionary<String, String> subParams = new Dictionary<String, String>
    {
      ["MotherBoard"] = "seller=rozetka/",
      ["CPU"] = "seller=rozetka/",
      ["RAM"] = "seller=rozetka;21256=3370/",
      ["GPU"] = "seller=rozetka;form-faktor128689=standartnaya/",
      ["PowerSupply"] = "seller=rozetka;tip-98627=kompjyuternyj/",
      ["Cooler"] = "seller=rozetka;21439=4996;21440=5004/",
      ["HDD"] = "seller=rozetka;22680=internal;naznachenie-221460=dlya-kompyutera/",
      ["SSD"] = "seller=rozetka;24749=15091;;naznachenie-193838=dlya-kompyutera/",
      ["Body"] = "seller=rozetka;21393=4502/"
    };
    // Ссылки на страницы каждого из типов комплектующих.
    private readonly static Dictionary<String,String> links = new Dictionary<String, String>
    {
```

```
    ["MotherBoard"] = "https://hard.rozetka.com.ua/motherboards/c80082/",
    ["CPU"] = "https://hard.rozetka.com.ua/processors/c80083/",
    ["RAM"] = "https://hard.rozetka.com.ua/memory/c80081/",
    ["GPU"] = "https://hard.rozetka.com.ua/videocards/c80087/",
    ["PowerSupply"] = "https://hard.rozetka.com.ua/psu/c80086/",
    ["Cooler"] = "https://hard.rozetka.com.ua/coolers/c80099/",
    ["HDD"] = "https://hard.rozetka.com.ua/hdd/c80084/",
    ["SSD"] = "https://hard.rozetka.com.ua/ssd/c80109/",
    ["Body"] = "https://hard.rozetka.com.ua/cases/c80090/"
};
// Названия характеристик каждого из типов комплектуючщих.
private readonly static Dictionary<String, List<String>> titles = new Dictionary<String, List<String>>
{
    ["MotherBoard"] =
    new List<String> {
        "Тип разъема",
        "Чипсет (Северный мост)",
        "Формфактор",
        "Поддержка памяти",
        "Поддержка процессоров",
        "Максимальный объем оперативной памяти",
        "PCI Express x16",
        "Коннектор питания",
        "Физические размеры"
    },
    ["CPU"] =
    new List<String> {
        "Тип разъема",
        "Количество ядер",
        "Внутренняя тактовая частота",
        "Максимальная тактовая частота",
        "Количество потоков",
        "Разблокированный множитель",
        "Мощность TDP",
        "Объем кэш памяти 3 уровня"
    },
    ["RAM"] =
    new List<String> {
        "Объем памяти",
        "Тип памяти",
        "Частота памяти",
        "Эффективная пропускная способность",
```

```
    "Количество планок",
    "Схема таймингов памяти"
},
["GPU"] =
new List<String> {
    "Частота памяти",
    "Объем памяти",
    "Частота ядра",
    "Разрядность шины памяти",
    "Тип памяти",
    "Минимально необходимая мощность БП",
    "Интерфейс",
    "Размеры"
},
["PowerSupply"] =
new List<String> {
    "Мощность",
    "Тип разъема подключения к материнской плате",
    "Тип разъема подключения питания процессора",
    "Тип разъемов дополнительного питания для видеокарт",
    "Количество SATA разъемов"
},
["Cooler"] =
new List<String> {
    "Совместимость",
    "Частота вращения",
    "Размеры"
},
["HDD"] =
new List<String> {
    "Форм-фактор",
    "Емкость накопителя",
    "Интерфейс",
    "Скорость вращения шпинделя",
    "Объем буфера",
    "Скорость передачи данных"
},
["SSD"] =
new List<String> {
    "Объем",
    "Скорость чтения",
    "Скорость записи",
```

```
            "Формфактор",
            "Интерфейс"
        },
        ["Body"] =
        new List<String> {
            "Форм-фактор материнской платы",
            "Количество внутренних отсеков 3.5\"",
            "Количество отсеков 2.5\"",
            "Максимальная длина видеокарты",
            "Максимальная высота кулера ЦП",
            "Размеры"
        }
    };

    #endregion

    #region HTML and Browser parsing tools fields

    // Объект для получения HTML документов (страниц) по протоколу HTTP.
    private readonly static HtmlWeb htmlWeb;
    // Объект для хранения HTML документа.
    private static HtmlDocument htmlDocument;
    // Объект для доступа к браузеру Chrome.
    private static ChromeDriver driver;

    #endregion

    #region Parsing class methods + constructor

    static HTMLParser()
    {
        htmlWeb = new HtmlWeb();
    }
    // Парсинг количества страниц с комплектующими определённого типа.
    private static Int32 GetPagesAmount()
    {
        return
Int32.Parse(htmlDocument.DocumentNode.SelectSingleNode("//ul[@class='pagination__list']/li[last()]/a").InnerText);
    }
    // Парсинг характеристик комплектующих определённого типа.
    public static List<TEntity> GetComponentsData<TEntity>(String componentName)
        where TEntity : class
```

```
{
    // Название характеристики.
    String featureTitle = "";
    // Ссылка на изображение комплектующей.
    String imageUri = "";
    // Комплектующие определённого типа.
    List <TEntity> outData = new List<TEntity>();
    // Характеристики одной комплектующей.
    List<String> innerList = new List<String>();
    // Ссылки на комплектующие с одной страницы.
    List<String> hrefValues = new List<String>();

    // Узлы HTML документа.
    HtmlNodeCollection htmlValueNodes;

    driver = new ChromeDriver();

    // Загрузка основной (первой) страницы с комплектующими определённого типа.
    htmlDocument = htmlWeb.Load(links[componentName] + subParams[componentName]);

    Int32 pages = GetPagesAmount();

    // Проход по всем страницам с комплектующими определённого типа.
    for (Int32 i = 0; i < pages; i++)
    {
        // Загрузка следующей страницы.
        driver.Url = links[componentName] + "page=" + (i + 1).ToString() + ";" + subParams[componentName];
        // Считывание ссылок на комплектующие.
        IReadOnlyCollection<IWebElement> elements =
            driver.FindElements(By.XPath(@"//ul[@class='catalog-grid']/li//app-goods-tile-default/div/div[2]/a[1]"));

        foreach (IWebElement elem in elements)
            hrefValues.Add(elem.GetAttribute("href"));
        // Проход по считаным ссылкам.
        foreach (String href in hrefValues)
        {
            // Загрузка страницы по ссылке.
            htmlDocument = htmlWeb.Load(href);

            innerList = Enumerable.Repeat("None", titles[componentName].Count + 2).ToList();
            // Считывание названия комплектующей.
            innerList[0] = htmlDocument.DocumentNode.SelectSingleNode("//product-top/div/div[1]/h1").InnerText;
```

```
                // Считывание названия комплектующей.
                innerList[innerList.Count - 1] = htmlDocument.DocumentNode.SelectSingleNode("//product-main-
info/div/div[1]/div/p[1]/text()")?.InnerText ?? "0";
                // Считывание ссылки на загрузку изображения.
                imageUri = htmlDocument.DocumentNode.SelectSingleNode("//product-gallery-
main/div/div/ul/div/div/li[1]/a/img").Attributes["src"].Value;
                // Переход на страницу характеристик комплектующей.
                htmlDocument = htmlWeb.Load(href + INNERPARAM);
                // Считывание всех характеристик комплектующей.
                htmlValueNodes = htmlDocument.DocumentNode.SelectNodes("//dl/dd/ul");
                // Фильтрация и считывание нужных характеристик.
                for (Int32 j = 0; j < htmlValueNodes.Count; j++)
                {
                    featureTitle = htmlValueNodes[j].ParentNode.PreviousSibling.FirstChild.InnerText;
                    if (titles[componentName].Contains(featureTitle))
                    {
                        if (htmlValueNodes[j].ChildNodes == null)
                            innerList[titles[componentName].IndexOf(featureTitle) + 1] = htmlValueNodes[j].InnerText;
                        else
                        {
                            String tempStr = "";
                            foreach (HtmlNode node in htmlValueNodes[j].ChildNodes)
                                tempStr += node.InnerText + " ";
                            innerList[titles[componentName].IndexOf(featureTitle) + 1] = tempStr.TrimStart(' ').TrimEnd(' ');
                        }
                    }
                }
                // Конвертация и добавление комплектующей к списку.
                outData.Add(ConvertProvider.ConvertTo(componentName, innerList, new Uri(imageUri)));
                innerList.Clear();
            }
            hrefValues.Clear();
        }
        driver.Quit();
        return outData;
    }

    #endregion

  }
}
```

ImageProvider.cs.

```
using System;
using System.IO;
using System.Net;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Windows;
using System.Windows.Media.Imaging;

namespace PCConfigurator.Model
{
   internal static class ImageProvider
   {
      // Объект для загрузки изображения по Uri.
      private readonly static WebClient webClient;
      // Путь к файлу изображения.
      public static String Filename { get; private set; }

      static ImageProvider()
      {
         webClient = new WebClient();
      }
      // Изменение размера изображения.
      private static Bitmap ResizeImage(Image image, Int32 widthDel, Int32 heightDel)
      {
         Rectangle destRect = new Rectangle(0, 0, image.Width / widthDel, image.Height / heightDel);
         Bitmap destImage = new Bitmap(image.Width / widthDel, image.Height / heightDel);
         destImage.SetResolution(image.HorizontalResolution, image.VerticalResolution);

         using (Graphics graphics = Graphics.FromImage(destImage))
         {
            graphics.CompositingMode = CompositingMode.SourceCopy;
            graphics.CompositingQuality = CompositingQuality.HighQuality;
            graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;
            graphics.SmoothingMode = SmoothingMode.HighQuality;
            graphics.PixelOffsetMode = PixelOffsetMode.HighQuality;

            using (ImageAttributes wrapMode = new ImageAttributes())
            {
               wrapMode.SetWrapMode(WrapMode.TileFlipXY);
               graphics.DrawImage(image, destRect, 0.0f, 0.0f, image.Width, image.Height, GraphicsUnit.Pixel, wrapMode);
            }
```

```
    }

    return destImage;
}
// Преобразование изображения формата JPEG в массив байтов.
public static Byte[] ToBytesJpeg(this Image image)
{
    MemoryStream memoryStream = new MemoryStream();
    image.Save(memoryStream, ImageFormat.Jpeg);
    return memoryStream.ToArray();
}
// Преобразование изображения с зарание определённым форматом в массив байтов.
public static Byte[] ToBytes(this Image image)
{
    MemoryStream memoryStream = new MemoryStream();
    image.Save(memoryStream, image.RawFormat);
    return memoryStream.ToArray();
}
// Преобразование массива байтов в изображение.
public static Image ToImage(this Byte[] byteArray)
{
    MemoryStream memoryStream = new MemoryStream(byteArray);
    return Image.FromStream(memoryStream);
}
// Загрузка изображения по Uri.
public static Byte[] GetNewImageByUri(Uri uri)
{
    Filename = @"Z:\erwin\" + Path.GetFileName(uri.LocalPath);
    webClient.DownloadFile(uri, Filename);

    return GetImageFromFile(Filename);
}
// Загрузка изображения как файла.
public static Byte[] GetImageFromFile(String filepath)
{
    return ResizeImage(Image.FromFile(filepath), 2, 2).ToBytesJpeg();
}
// Преобразование изображения в постоянный набор пикселей.
public static BitmapSource ToBitmapSource(this Image bitmap)
{
    return System.Windows.Interop.Imaging.CreateBitmapSourceFromHBitmap(
        ((Bitmap)bitmap).GetHbitmap(),
```

```
            IntPtr.Zero,
            Int32Rect.Empty,
            BitmapSizeOptions.FromEmptyOptions());
        }
    }
}
```

Лістинг файлів моделі представлення:

BaseVM.cs.

```
using System;
using System.ComponentModel;

using PCConfigurator.Model;

namespace PCConfigurator.ViewModel
{
    internal abstract class BaseVM : UnderBaseVM
    {
        #region Components collections

        // Связующие списки характеристикики компонентов.
        private BindingList<MotherBoard> motherBoards;
        private BindingList<CPU> cpus;
        private BindingList<GPU> gpus;
        private BindingList<RAM> rams;
        private BindingList<PowerSupply> powerSupplies;
        private BindingList<Cooler> coolers;
        private BindingList<HDD> hdds;
        private BindingList<SSD> ssds;
        private BindingList<Body> bodies;

        public BindingList<MotherBoard> MotherBoards
        {
            get { return motherBoards ?? (motherBoards = new BindingList<MotherBoard>()); }
            set
            {
                motherBoards = value;
                OnPropertyChanged(nameof(MotherBoards));
            }
        }
```

```
public BindingList<CPU> CPUs
{
    get { return cpus ?? (cpus = new BindingList<CPU>()); }
    set
    {
        cpus = value;
        OnPropertyChanged(nameof(CPUs));
    }
}
public BindingList<GPU> GPUs
{
    get { return gpus ?? (gpus = new BindingList<GPU>()); }
    set
    {
        gpus = value;
        OnPropertyChanged(nameof(GPUs));
    }
}
public BindingList<RAM> RAMs
{
    get { return rams ?? (rams = new BindingList<RAM>()); }
    set
    {
        rams = value;
        OnPropertyChanged(nameof(RAMs));
    }
}
public BindingList<PowerSupply> PowerSupplies
{
    get { return powerSupplies ?? (powerSupplies = new BindingList<PowerSupply>()); }
    set
    {
        powerSupplies = value;
        OnPropertyChanged(nameof(PowerSupplies));
    }
}
public BindingList<Cooler> Coolers
{
    get { return coolers ?? (coolers = new BindingList<Cooler>()); }
    set
    {
```

```
        coolers = value;
        OnPropertyChanged(nameof(Coolers));
    }
}
public BindingList<HDD> HDDs
{
    get { return hdds ?? (hdds = new BindingList<HDD>()); }
    set
    {
        hdds = value;
        OnPropertyChanged(nameof(HDDs));
    }
}
public BindingList<SSD> SSDs
{
    get { return ssds ?? (ssds = new BindingList<SSD>()); }
    set
    {
        ssds = value;
        OnPropertyChanged(nameof(SSDs));
    }
}
public BindingList<Body> Bodies
{
    get { return bodies ?? (bodies = new BindingList<Body>()); }
    set
    {
        bodies = value;
        OnPropertyChanged(nameof(Bodies));
    }
}

#endregion

#region Filters values
// Фильтры для характеристик.
private MotherBoardFilteringVM motherBoardFiltering;
private CPUFilteringVM cpuFiltering;
private GPUFilteringVM gpuFiltering;
private RAMFilteringVM ramFiltering;
private PowerSupplyFilteringVM powerSupplyFiltering;
private CoolerFilteringVM coolerFiltering;
```

```
private HDDFilteringVM hddFiltering;
private SSDFilteringVM ssdFiltering;
private BodyFilteringVM bodyFiltering;

public MotherBoardFilteringVM MotherBoardFiltering
{
   get { return motherBoardFiltering; }
   set
   {
      if (motherBoardFiltering != value)
      {
         motherBoardFiltering = value;
         OnPropertyChanged(nameof(MotherBoardFiltering));
      }
   }
}

public CPUFilteringVM CPUFiltering
{
   get { return cpuFiltering; }
   set
   {
      if (cpuFiltering != value)
      {
         cpuFiltering = value;
         OnPropertyChanged(nameof(CPUFiltering));
      }
   }
}

public GPUFilteringVM GPUFiltering
{
   get { return gpuFiltering; }
   set
   {
      if (gpuFiltering != value)
      {
         gpuFiltering = value;
         OnPropertyChanged(nameof(GPUFiltering));
      }
   }
}
```

```
public RAMFilteringVM RAMFiltering
{
    get { return ramFiltering; }
    set
    {
        if (ramFiltering != value)
        {
            ramFiltering = value;
            OnPropertyChanged(nameof(RAMFiltering));
        }
    }
}

public PowerSupplyFilteringVM PowerSupplyFiltering
{
    get { return powerSupplyFiltering; }
    set
    {
        if (powerSupplyFiltering != value)
        {
            powerSupplyFiltering = value;
            OnPropertyChanged(nameof(PowerSupplyFiltering));
        }
    }
}

public CoolerFilteringVM CoolerFiltering
{
    get { return coolerFiltering; }
    set
    {
        if (coolerFiltering != value)
        {
            coolerFiltering = value;
            OnPropertyChanged(nameof(CoolerFiltering));
        }
    }
}

public HDDFilteringVM HDDFiltering
{
```

```
    get { return hddFiltering; }
    set
    {
        if (hddFiltering != value)
        {
            hddFiltering = value;
            OnPropertyChanged(nameof(HDDFiltering));
        }
    }
}

public SSDFilteringVM SSDFiltering
{
    get { return ssdFiltering; }
    set
    {
        if (ssdFiltering != value)
        {
            ssdFiltering = value;
            OnPropertyChanged(nameof(SSDFiltering));
        }
    }
}

public BodyFilteringVM BodyFiltering
{
    get { return bodyFiltering; }
    set
    {
        if (bodyFiltering != value)
        {
            bodyFiltering = value;
            OnPropertyChanged(nameof(BodyFiltering));
        }
    }
}

#endregion

#region Selected item
// Выбранный из списка/таблицы компонент.
private Model.Component selectedItem;
```

```
public Model.Component SelectedItem
{
    get { return selectedItem; }
    set
    {
        if (selectedItem != value)
        {
            selectedItem = value;
            OnPropertyChanged(nameof(SelectedItem));
        }
    }
}

#endregion

#region Rules
// Правила совместимости комплектующих.
private BindingList<CompatibilityRule> compatibilityRules;
public BindingList<CompatibilityRule> CompatibilityRules
{
    get { return compatibilityRules ?? new BindingList<CompatibilityRule>(); }
    set
    {
        compatibilityRules = value;
        OnPropertyChanged(nameof(CompatibilityRules));
    }
}

// Правила оценки характеристик.
private BindingList<EvaluativeRule> evaluativeRules;
public BindingList<EvaluativeRule> EvaluativeRules
{
    get { return evaluativeRules ?? new BindingList<EvaluativeRule>(); }
    set
    {
        evaluativeRules = value;
        OnPropertyChanged(nameof(EvaluativeRules));
    }
}

#endregion
```

```
        protected BaseVM()
        {
            InitCollections();
            InitFiltering();
            InitRules();
        }
        // Инициализация связующих списков характеристик комплектующих.
        protected void InitCollections()
        {
            MotherBoards = (BindingList<MotherBoard>)DataBaseProvider.Context.MotherBoards.GetCollection();
            CPUs = (BindingList<CPU>)DataBaseProvider.Context.CPUs.GetCollection();
            GPUs = (BindingList<GPU>)DataBaseProvider.Context.GPUs.GetCollection();
            RAMs = (BindingList<RAM>)DataBaseProvider.Context.RAMs.GetCollection();
            PowerSupplies = (BindingList<PowerSupply>)DataBaseProvider.Context.PowerSupplies.GetCollection();
            Coolers = (BindingList<Cooler>)DataBaseProvider.Context.Coolers.GetCollection();
            HDDs = (BindingList<HDD>)DataBaseProvider.Context.HDDs.GetCollection();
            SSDs = (BindingList<SSD>)DataBaseProvider.Context.SSDs.GetCollection();
            Bodies = (BindingList<Body>)DataBaseProvider.Context.Bodies.GetCollection();
        }
        // Инициализация моделей представлений фильтров.
        protected void InitFiltering()
        {
            MotherBoardFiltering = new MotherBoardFilteringVM();
            CPUFiltering = new CPUFilteringVM();
            GPUFiltering = new GPUFilteringVM();
            RAMFiltering = new RAMFilteringVM();
            PowerSupplyFiltering = new PowerSupplyFilteringVM();
            CoolerFiltering = new CoolerFilteringVM();
            HDDFiltering = new HDDFilteringVM();
            SSDFiltering = new SSDFilteringVM();
            BodyFiltering = new BodyFilteringVM();
        }
        // Инициализация правил совместимости и правил оценки харакетристик комплектующих.
        protected void InitRules()
        {
            CompatibilityRules =
(BindingList<CompatibilityRule>)DataBaseProvider.Context.CompatibilityRules.GetCollection();
            EvaluativeRules = (BindingList<EvaluativeRule>)DataBaseProvider.Context.EvaluativeRules.GetCollection();

            for (Int32 i = 0; i < CompatibilityRules.Count; i++)
                CompatibilityRules[i].RuleChecker = CompatibilityProvider.Rules[i];
```

```
            for (Int32 i = 0; i < EvaluativeRules.Count; i++)
                EvaluativeRules[i].RuleChecker = EvaluationProvider.Rules[i];
        }
    }
}
```

AuthorizationVM.cs.

```
using System;
using PCConfigurator.Model;
using System.Windows;

namespace PCConfigurator.ViewModel
{
    internal sealed class AuthorizationVM : UnderBaseVM
    {
        #region User`s data

        private String username;
        private String password;

        public String Username
        {
            get { return username; }
            set
            {
                if (username != value)
                {
                    username = value;
                    OnPropertyChanged(nameof(Username));
                }
            }
        }

        public String Password
        {
            get { return password; }
            set
            {
                if (password != value)
                {
                    password = value;
                    OnPropertyChanged(nameof(Password));
```

```
            }
        }
    }

    #endregion

    #region Commands

    public RelayCommand LoginCommand { get; }
    public RelayCommand ReturnCommand { get; }
    public Action CloseAction { get; set; }

    #endregion

    public AuthorizationVM()
    {
        LoginCommand = new RelayCommand(Login);
        ReturnCommand = new RelayCommand(Return);
    }
    // Авторизация (проверка логина и пароля).
    private void Login(Object param)
    {
        if (Username == User.Username && Password == User.Password)
        {
            User.IsLogedin = true;
            MessageBox.Show("Completed");
            CloseAction();
        }
        else
            MessageBox.Show("Wrong values");
    }
    // Возвращение на главную страницу.
    private void Return(Object param)
    {
        CloseAction();
    }
  }
}
```

MainWindowVM.cs.

```
using System;
using System.Linq;
```

```
using System.ComponentModel;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using PCConfigurator.Model;
using Microsoft.Win32;
using System.Windows;

namespace PCConfigurator.ViewModel
{
    internal sealed class MainWindowVM : BaseVM
    {
        #region Supporting variables and properties
        // Количесто комплектующих.
        private const Int32 COMPAMOUNT = 9;
        private String selectedComponent;
        // Прогресс заполнения зборки.
        private Int32 progress;
        // Выбранный режим подбора.
        private String modeCBSV;
        // Первичные значения компонентов сборки (абстрактные изображения).
        private static readonly Model.Component[] defaultComponents =
        {
            new MotherBoard() { Image = Properties.Resources.motherboard.ToBytes() },
            new CPU() { Image = Properties.Resources.processor.ToBytes() },
            new GPU() { Image = Properties.Resources.graphics_card.ToBytes() },
            new RAM() { Image = Properties.Resources.ram.ToBytes() },
            new PowerSupply() { Image = Properties.Resources.power.ToBytes() },
            new Cooler() { Image = Properties.Resources.fan.ToBytes() },
            new HDD() { Image = Properties.Resources.harddisk.ToBytes() },
            new SSD() { Image = Properties.Resources.ssd_drive.ToBytes() },
            new Body() { Image = Properties.Resources.computer.ToBytes() }
        };
        // Результаты проверки на совместимость.
        private Boolean[] isCompatibleArr = Enumerable.Repeat(true, 8).ToArray();
        // Оценки для каждого из типов требований пользователя.
        private static readonly Dictionary<String, Int32> markToMode = new Dictionary<String, Int32>
        {
            ["Study"] = 7,
            ["Office"] = 8,
            ["Multimedia/Home"] = 9,
            ["Adv. Multimedia/Home"] = 10,
            ["Gaming"] = 11,
```

```
    ["Adv. Gaming"] = 12,
    ["Professional"] = 13
};
// Режимы побра.
public static String[] Modes { get; } =
{
    "Gaming",
    "Adv. Gaming",
    "Professional",
    "Multimedia/Home",
    "Adv. Multimedia/Home",
    "Study",
    "Office"
};
public String ModeCBSV
{
    get { return modeCBSV ?? "Gaming"; }
    set
    {
        if (modeCBSV != value)
        {
            modeCBSV = value;
            OnPropertyChanged(nameof(ModeCBSV));
        }
    }
}
// Выбранные компоненты (текущая сборка).
public ObservableCollection<Model.Component> ChosenComponents { get; set; }
// Общая цена выбраных компонентов.
public Int32 SummaryCost { get; private set; } = 0;
// Количество комплектующих в списке, из которого выбираются компоненты в сборку.
public Int32 SelectedItemsAmount { get; private set; } = 0;
public String CompatibilityTooltip { get; private set; }
// Отображаемое занчение совместимости.
public String Compatibility { get; private set; } = "In progress...";
// Оценка соответствия.
public Decimal Score { get; private set; }
public String ScoreTooltip { get; private set; }
public Int32 Progress
{
    get { return progress; }
    private set { progress = value * 100 / COMPAMOUNT; }
```

```
}

#endregion

#region Components lists and filters visibility

// Отображение элементов списка, фильтров, заставки и количества комплектующих в списке (Да/Нет).
public Boolean DefaultVisibility { get; private set; } = true;
public Boolean StateElementVisibility { get; private set; } = false;
public Boolean MotherBoardVisibility { get; private set; } = false;
public Boolean CPUVisibility { get; private set; } = false;
public Boolean GPUVisibility { get; private set; } = false;
public Boolean RAMVisibility { get; private set; } = false;
public Boolean PowerSupplyVisibility { get; private set; } = false;
public Boolean CoolerVisibility { get; private set; } = false;
public Boolean HDDVisibility { get; private set; } = false;
public Boolean SSDVisibility { get; private set; } = false;
public Boolean BodyVisibility { get; private set; } = false;

#endregion

#region Commands

public RelayCommand SaveMenuCommand { get; }
public RelayCommand LoadMenuCommand { get; }
public RelayCommand DataBaseMenuCommand { get; }
public RelayCommand KnowledgeMenuCommand { get; }
public RelayCommand ChooseTypeCommand { get; }
public RelayCommand ChooseComponentCommand { get; }
public RelayCommand DeleteComponentCommand { get; }
public RelayCommand ApplyFiltersCommand { get; }
public RelayCommand ResetFiltersCommand { get; }

#endregion

public MainWindowVM() : base()
{
    SaveMenuCommand = new RelayCommand(SaveAssembly);
    LoadMenuCommand = new RelayCommand(LoadAssembly);
    DataBaseMenuCommand = new RelayCommand(ShowDataBase);
    KnowledgeMenuCommand = new RelayCommand(ShowKnowledgeBase);
    ChooseTypeCommand = new RelayCommand(ChooseType);
```

```
    ChooseComponentCommand = new RelayCommand(ChooseComponent);
    DeleteComponentCommand = new RelayCommand(DeleteComponent);
    ApplyFiltersCommand = new RelayCommand(ApplyFilters);
    ResetFiltersCommand = new RelayCommand(ResetFilters);

    InitAssembly();
}

#region Private View-Model methods

// Заполнение сборки первичными элементами.
private void InitAssembly()
{
    if (ChosenComponents != null)
        ChosenComponents.Clear();

    ChosenComponents = new ObservableCollection<Model.Component>(defaultComponents.ToList());
}
// Заполнение сборки по названиям из загруженого файла.
private void FillByTitles(String[] titles)
{
    const String noneValue = "None";
    SelectedItem = null;
    InitAssembly();

    if (titles[0] != noneValue)
        ChosenComponents[0] = MotherBoards.ToList().Find(c => c.Title == titles[0]);
    if (titles[1] != noneValue)
        ChosenComponents[1] = CPUs.ToList().Find(c => c.Title == titles[1]);
    if (titles[2] != noneValue)
        ChosenComponents[2] = GPUs.ToList().Find(c => c.Title == titles[2]);
    if (titles[3] != noneValue)
        ChosenComponents[3] = RAMs.ToList().Find(c => c.Title == titles[3]);
    if (titles[4] != noneValue)
        ChosenComponents[4] = PowerSupplies.ToList().Find(c => c.Title == titles[4]);
    if (titles[5] != noneValue)
        ChosenComponents[5] = Coolers.ToList().Find(c => c.Title == titles[5]);
    if (titles[6] != noneValue)
        ChosenComponents[6] = HDDs.ToList().Find(c => c.Title == titles[6]);
    if (titles[7] != noneValue)
        ChosenComponents[7] = SSDs.ToList().Find(c => c.Title == titles[7]);
    if (titles[8] != noneValue)
```

```
            ChosenComponents[8] = Bodies.ToList().Find(c => c.Title == titles[8]);

        SummaryCost = ChosenComponents.Select(c => c.Cost).ToList().Sum();
        Compatibility = CheckCompatibility() ? "Positive" : "Negative";
        Progress = CountTaken();
    }

    // Сохранение сборки в файлу формата .pcconf.
    private void SaveAssembly(Object param)
    {
        String path = "";
        SaveFileDialog saveFileDialog = new SaveFileDialog
        {
            InitialDirectory = @"Z:\erwin\",
            Filter = "PCAssembly files (*.pcconf)|*.pcconf",
            FileName = "PCAssembly(" + DateTime.Now.ToString("MM/dd/yyyy") + ").pcconf"
        };

        if ((Boolean)saveFileDialog.ShowDialog())
            path = saveFileDialog.FileName;
        else
            return;

        FileProvider.SaveAssemblyFile(ChosenComponents.ToList(), path);
        MessageBox.Show("Assembly has been saved", "Information", MessageBoxButton.OK,
MessageBoxImage.Information);
    }
    // Загрузка сборки из файла формата .pcconf.
    private void LoadAssembly(Object param)
    {
        String path = "";
        OpenFileDialog openFileDialog = new OpenFileDialog
        {
            InitialDirectory = @"Z:\erwin\",
            Filter = "PCAssembly files (*.pcconf)|*.pcconf"
        };

        if ((Boolean)openFileDialog.ShowDialog())
            path = openFileDialog.FileName;
        else
            return;
```

```
            FillByTitles(FileProvider.LoadAssemblyFile(path, COMPAMOUNT));
            MessageBox.Show("Assembly has been loaded", "Information", MessageBoxButton.OK,
MessageBoxImage.Information);
        }
        // Авторизация пользователя.
        private Boolean Authorize()
        {
            if (!User.IsLogedin)
            {
                View.Authorization authorization = new View.Authorization();
                authorization.ShowDialog();
            }

            return User.IsLogedin;
        }
        // Переход на окно базы данных.
        private void ShowDataBase(Object param)
        {
            if (Authorize())
            {
                DataBase dataBase = new DataBase();
                dataBase.ShowDialog();

                DataBaseProvider.ConetextReNew();
                InitCollections();
            }
        }
        // Переход на окно базы знаний.
        private void ShowKnowledgeBase(Object param)
        {
            if (Authorize())
            {
                KnowledgeBase knowledgeBase = new KnowledgeBase();
                knowledgeBase.ShowDialog();
            }
            //DataBaseProvider.ConetextReNew();
            //InitRules();
        }
        // Установка всех списков и фильтров для них в невидимое состояние.
        private void AllInvisible()
        {
            MotherBoardVisibility = false;
```

```
    CPUVisibility = false;
    GPUVisibility = false;
    RAMVisibility = false;
    PowerSupplyVisibility = false;
    CoolerVisibility = false;
    HDDVisibility = false;
    SSDVisibility = false;
    BodyVisibility = false;
}
// Подсчёт количества элементов в списке.
private void CountItems()
{
    switch (selectedComponent)
    {
        case "MotherBoard":
            SelectedItemsAmount = MotherBoards.Count;
            break;
        case "CPU":
            SelectedItemsAmount = CPUs.Count;
            break;
        case "GPU":
            SelectedItemsAmount = GPUs.Count;
            break;
        case "RAM":
            SelectedItemsAmount = RAMs.Count;
            break;
        case "PowerSupply":
            SelectedItemsAmount = PowerSupplies.Count;
            break;
        case "Cooler":
            SelectedItemsAmount = Coolers.Count;
            break;
        case "HDD":
            SelectedItemsAmount = HDDs.Count;
            break;
        case "SSD":
            SelectedItemsAmount = SSDs.Count;
            break;
        case "Body":
            SelectedItemsAmount = Bodies.Count;
            break;
    }
```

```
}
// Подсчёт количества компонентов в сборке.
private Int32 CountTaken()
{
    Int32 taken = 0;

    foreach (Model.Component component in ChosenComponents)
        if (component.IsTaken())
            taken++;
    return taken;
}

// Проверка достаточной сформированности сборки для оценки соответствия.
private Boolean IsEnoughTaken()
{
    Boolean isEnough = true;
    for (Int32 i = 0; i < ChosenComponents.Count; i++)
    {
        if (i == 5 || i == 7)
            continue;
        if (!ChosenComponents[i].IsTaken())
            isEnough = false;
    }
    return isEnough;
}

// Проверка совместимости компонентова текущей сборки.
private Boolean CheckCompatibility()
{
    Boolean compatibility = true;

    if (ChosenComponents[0].IsTaken())
    {
        if (ChosenComponents[1].IsTaken() && CompatibilityRules[0].IsActive)
            isCompatibleArr[0] = CompatibilityRules[0].RuleChecker(ChosenComponents[0], ChosenComponents[1]);
        if (ChosenComponents[2].IsTaken() && CompatibilityRules[1].IsActive)
            isCompatibleArr[1] = CompatibilityRules[1].RuleChecker(ChosenComponents[0], ChosenComponents[2]);
        if (ChosenComponents[3].IsTaken() && CompatibilityRules[2].IsActive)
            isCompatibleArr[2] = CompatibilityRules[2].RuleChecker(ChosenComponents[0], ChosenComponents[3]);
        if (ChosenComponents[8].IsTaken() && CompatibilityRules[3].IsActive)
            isCompatibleArr[3] = CompatibilityRules[3].RuleChecker(ChosenComponents[0], ChosenComponents[8]);
    }
```

```
    if (ChosenComponents[2].IsTaken())
    {
        if (ChosenComponents[4].IsTaken() && CompatibilityRules[4].IsActive)
            isCompatibleArr[4] = CompatibilityRules[4].RuleChecker(ChosenComponents[2], ChosenComponents[4]);
        if (ChosenComponents[8].IsTaken() && CompatibilityRules[5].IsActive)
            isCompatibleArr[5] = CompatibilityRules[5].RuleChecker(ChosenComponents[2], ChosenComponents[8]);
    }

    if (ChosenComponents[5].IsTaken())
    {
        if (ChosenComponents[1].IsTaken() && CompatibilityRules[6].IsActive)
            isCompatibleArr[6] = CompatibilityRules[6].RuleChecker(ChosenComponents[5], ChosenComponents[1]);
        if (ChosenComponents[8].IsTaken() && CompatibilityRules[7].IsActive)
            isCompatibleArr[7] = CompatibilityRules[7].RuleChecker(ChosenComponents[5], ChosenComponents[8]);
    }

    foreach (Boolean isVal in isCompatibleArr)
        if (!isVal)
            compatibility = false;

    return compatibility;
}
// Выбор типа комплектующих.
private void ChooseType(Object param)
{
    AllInvisible();
    DefaultVisibility = false;
    StateElementVisibility = true;
    selectedComponent = param.ToString();
    CountItems();

    switch (selectedComponent)
    {
        case "MotherBoard":
            MotherBoardVisibility = true;
            break;
        case "CPU":
            CPUVisibility = true;
            break;
        case "GPU":
            GPUVisibility = true;
```

```
            break;
        case "RAM":
            RAMVisibility = true;
            break;
        case "PowerSupply":
            PowerSupplyVisibility = true;
            break;
        case "Cooler":
            CoolerVisibility = true;
            break;
        case "HDD":
            HDDVisibility = true;
            break;
        case "SSD":
            SSDVisibility = true;
            break;
        case "Body":
            BodyVisibility = true;
            break;
    }
}
// Добавление комплектующей определённого типа в сборку.
private void ChooseComponent(Object param)
{
    if(param != null)
    {
        switch (param.GetType().Name)
        {
            case "MotherBoard":
                if (SelectedItem is MotherBoard)
                    ChosenComponents[0] = (MotherBoard)SelectedItem;
                break;
            case "CPU":
                if (SelectedItem is CPU)
                    ChosenComponents[1] = (CPU)SelectedItem;
                break;
            case "GPU":
                if (SelectedItem is GPU)
                    ChosenComponents[2] = (GPU)SelectedItem;
                break;
            case "RAM":
                if (SelectedItem is RAM)
```

```
                ChosenComponents[3] = (RAM)SelectedItem;
            break;
        case "PowerSupply":
            if (SelectedItem is PowerSupply)
                ChosenComponents[4] = (PowerSupply)SelectedItem;
            break;
        case "Cooler":
            if (SelectedItem is Cooler)
                ChosenComponents[5] = (Cooler)SelectedItem;
            break;
        case "HDD":
            if (SelectedItem is HDD)
                ChosenComponents[6] = (HDD)SelectedItem;
            break;
        case "SSD":
            if (SelectedItem is SSD)
                ChosenComponents[7] = (SSD)SelectedItem;
            break;
        case "Body":
            if (SelectedItem is Body)
                ChosenComponents[8] = (Body)SelectedItem;
            break;
    }

    SummaryCost = ChosenComponents.Select(c => c.Cost).ToList().Sum();
    Compatibility = CheckCompatibility() ? "Positive" : "Negative";
    Progress = CountTaken();
    if (IsEnoughTaken())
        Score = EvaluationProvider.Evaluate(markToMode[ModeCBSV]);
  }
}

// Удаление компонента из сборки.
private void DeleteComponent(Object param)
{
  if (param != null)
  {
    Int32 index = Int32.Parse(param.ToString());
    ChosenComponents[index] = defaultComponents[index];
  }

  SelectedItem = null;
```

```
    SummaryCost = ChosenComponents.Select(c => c.Cost).ToList().Sum();
    Compatibility = CheckCompatibility() ? "Positive" : "Negative";
    Progress = CountTaken();
    if (IsEnoughTaken())
        Score = EvaluationProvider.Evaluate(markToMode[ModeCBSV]);
}
// Применение установленных фильтров.
private void ApplyFilters(Object param)
{
    switch (selectedComponent)
    {
        case "MotherBoard":
            MotherBoards = new BindingList<MotherBoard>(MotherBoardFiltering.GetFiltersResult());
            break;
        case "CPU":
            CPUs = new BindingList<CPU>(CPUFiltering.GetFiltersResult());
            break;
        case "GPU":
            GPUs = new BindingList<GPU>(GPUFiltering.GetFiltersResult());
            break;
        case "RAM":
            RAMs = new BindingList<RAM>(RAMFiltering.GetFiltersResult());
            break;
        case "PowerSupply":
            PowerSupplies = new BindingList<PowerSupply>(PowerSupplyFiltering.GetFiltersResult());
            break;
        case "Cooler":
            Coolers = new BindingList<Cooler>(CoolerFiltering.GetFiltersResult());
            break;
        case "HDD":
            HDDs = new BindingList<HDD>(HDDFiltering.GetFiltersResult());
            break;
        case "SSD":
            SSDs = new BindingList<SSD>(SSDFiltering.GetFiltersResult());
            break;
        case "Body":
            Bodies = new BindingList<Body>(BodyFiltering.GetFiltersResult());
            break;
    }
    CountItems();
}
// Сборс заначений фильтров до начального состояния.
```

```
        private void ResetFilters(Object param)
        {
            switch (selectedComponent)
            {
                case "MotherBoard":
                    MotherBoardFiltering.ResetValues();
                    break;
                case "CPU":
                    CPUFiltering.ResetValues();
                    break;
                case "GPU":
                    GPUFiltering.ResetValues();
                    break;
                case "RAM":
                    RAMFiltering.ResetValues();
                    break;
                case "PowerSupply":
                    PowerSupplyFiltering.ResetValues();
                    break;
                case "Cooler":
                    CoolerFiltering.ResetValues();
                    break;
                case "HDD":
                    HDDFiltering.ResetValues();
                    break;
                case "SSD":
                    SSDFiltering.ResetValues();
                    break;
                case "Body":
                    BodyFiltering.ResetValues();
                    break;
            }
        }

        #endregion
    }
}
```