ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ «ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ» ІМЕНІ ІВАНА ПУЛЮЯ

Циклова комісія комп'ютерних наук

КУРСОВА РОБОТА

з дисципліни:

«Об’єктно-орієнтоване програмування»

на тему: **«Розробка програмного забезпечення «Конструктор ПК»»**

Студента 3 курсу групи КН-321 спеціальності 122 «Комп’ютерні науки»

Олійник Д.С.
(прізвище та ініціали)

Керівник: викладач Слободян Р.О.

Національна шкала _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії: _____ Р.О. Слободян
(підпис)

_____ Р.Ю. Болюбаш
(підпис)

м. Тернопіль – 2021

ЗМІСТ

ВСТУП

Програмування в звичному розумінні народилося в 60-х роках попереднього сторіччя, а персональні комп'ютери – в 70-х. Тодішні комп'ютерні технології не були сильно поширеними і використовувалися лише в промислових цілях, в тому числі для космічної світової програми. З тих часів багато все змінилося і зараз, коли на порозі видніється кінець першої чверті 21-го сторіччя, комп'ютери є значно доступнішими і кількість користувачів ПК вже перетнула межу за 1 мільярд людей, причому, це не враховуючи сервери, яких ще більше.

Комп'ютер став доступним настільки, що тепер кожен бажаючий може сам створити свій, тепер ПК – модульна конструкція, яка може багаторазово модифікуватися і при цьому бути невеликих розмірів, на відміну від тих самих 80-х років, де їх розмір можна було порівняти з аудиторією навчального закладу. Популярним є збирання свого власного ПК, мережа Інтернет містить величезну кількість інструкцій та керівництв по з'єднанню комплектуючих у єдиний механізм, документації та характеристики сучасних функціональних частин комп'ютера, дуже простим є придбати той чи інший важливий компонент на електронному ринку. Деякі веб-сайти надають таку послугу, як конфігуратор ПК, проте їх не можна назвати об'єктивними через умисне рекламування чи покращення характеристик того чи іншого компоненту, несовісну роботу, мета якої – лише продати продукт, що є нечесним в відношенні користувача або покупця. Окрім цього, майже жоден з таких ресурсів не вміє зберігати роботу користувача.

Прикладом вирішення проблеми є дана курсова робота, яка покликана створити програмне забезпечення, котре буде максимально дружнім та чесним з користувачем, де він сам може керувати своєю базою компонентів, вносити в неї корективи згідно зі знайденою інформацією та отримувати розрахунок потрібних йому величин, таких як ціна та потужність ПК. Окрім цього, важливою перевагою є можливість офлайн-роботи програми, оскільки вона не вимагає доступу до мережі Інтернет. Не дивлячись на сучасність, де Інтернет є доступним майже завжди та усюди, сьогодні все ж стаються випадки, коли такої можливості може й не бути.

На початках студенти вивчають алгоритмічне процедурне програмування на прикладі мови програмування С, але перевіреною часом та значно потужнішою є парадигма об’єктно-орієнтованого програмування, яку підтримує мова програмування С++. Потужним функціональним розширенням слугують фреймворки – інструментальні технологічні надбудови, яким і є Qt, що розробляється суспільством з 1991 року. Ще одною метою розроблення курсової роботи є покращення навичок та знань студента-виконавця, робота над першим проєктом в реальних умовах, яка передбачає використання таких технологій, як системи контролю версій і фреймворку, моделей розробки та патернів ООП, створення реальних та практичних умов розроблення програмного забезпечення.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Найменування та область застосування

Найменування програми – «Конструктор ПК».

Робоча назва – «PC Constructor».

Область застосування програми – конструювання збірок персональних комп'ютерів з різних складових, з перевіркою їх сумісності, розрахуванням потужності і ціни збірки.

1.2 Підстави для розробки

Підставою для проведення розробки являється індивідуальне завдання на курсову роботу з дисципліни «Об'єктно-орієнтоване програмування».

Найменування теми курсової роботи – «Розробка програмного забезпечення «Конструктор ПК»».

Замовник – керівник курсової роботи, викладач Слободян Руслан Олесьович.

Виконавець – студент групи КН-321 Олійник Денис Сергійович.

1.3 Призначення розробки

Призначенням програми є спрощення і вдосконалення процесу відбору комплектуючих для ПК, конструювання власних збірок, моделювання сумісності комплектуючих. Припускається, що основним сегментом користувачів є будь-які люди або компанії, перед якими стоїть задача зібрати ефективний, сумісний і робочий ПК з мінімальними витратами часу та грошей.

Передбачається зручне зберігання необхідних для збірки комплектуючих сучасного ПК та наданні швидкому доступі до кожного потрібного компоненту з можливістю перегляду його зовнішнього вигляду та характеристик.

1.4 Вимоги до програми чи програмного виробу

1.4.1 Вимоги до функціональних характеристик

Виконання поставленої задачі вимагає, щоб конструктор ПК забезпечував зручне та зрозуміле представлення великої маси комплектуючих комп'ютера та їх групування. Також обов'язковим є послідовне конструювання збірки з дотриманням сумісності компонентів, як-от:

– сокет у материнської плати та центрального процесора;

– тип оперативної пам'яті у материнської плати та безпосередньо мікросхеми оперативної пам'яті;

– тип інтерфейсу обміну даними у материнської плати та накопичувача;

– можливість блоку живлення забезпечити усі потреби і споживання інших компонентів збірки.

Програма повинна застерігати користувача від випадкової втрати даних, наприклад, збірки, і вважати конструкцію закінченою лише тоді, коли наявні обов'язкові компоненти:

– материнська плата;

– центральний процесор;

– оперативна пам'ять;

– хоча б один накопичувач;

– якщо центральний процесор не містить графічного ядра, то хоча б одна відеокарта;

– блок живлення.

Вхідні дані повинні опрацьовуватись зрозумілим та раціонально змодельованим графічним інтерфейсом, а вихідні зберігатися у найбільш оптимальному представленні організації даних, як-от база даних.

Програма повинна надавати користувачу можливість самому вносити у базу даних комплектуючі, які його цікавлять, редагувати їх, видаляти зайві.

1.4.2 Вимоги до часових характеристик

Оскільки програма повинна надавати доступ до функціоналу використовуючи графічний інтерфейс, то запуск програми мусить відбуватися за часом до 1 секунди, а відгук інтерфейсу бути майже моментальним, щоб не створювалося відчуття «провисання» програми.

1.4.3 Вимоги до надійності

Конструктор повинен застерігати користувача від введення неправильних вхідних даних, їх випадкового видалення, забезпечувати їх надійне зберігання і виключати можливість втрати даних. Весь функціонал забезпечення надійності повинен супроводжуватися зрозумілими для користувача повідомленнями, які буде виводити графічний інтерфейс.

1.4.4 Умови до експлуатації

Для експлуатації даної програми вистачить базових навичок користування персональним комп’ютером, але для ефективного використання конструктора за призначенням потребується знання архітектури ПК, характеристик його комплектуючих для конструювання збірки та внесення нових компонентів у базу.

1.4.5 Вимоги до складу і параметрів технічних засобів

В склад технічних засобів входять:

– монітор з діагоналлю не менше 13" та роздільністю не менше 1280 на 1024 пікселів, рекомендується широкий формат, на кшталт 16:9 або 16:10;

– комп’ютерні клавіатура та мишка;

– персональний комп’ютер з операційною системою, не старішою за Microsoft Windows 7 або Linux/MacOS (з перезбиранням конструктора на цільових

ОС) з такими мінімальними характеристиками:

- процесор з частотою ядра не менше 2ГГц;
- оперативна пам’ять з не менше ніж 1ГБ пам’яті;
- накопичувач з обсягом не менше 20ГБ.

1.4.6 Вимоги до інформаційної і програмної сумісності

Вихідні коди програми повинні бути реалізовані мовою програмування C++ зі стандартом не старішим за стандарт ISO C++11, а в якості середовища розробки повинне використовуватись середовище, яке підтримує C++11 та володіє можливістю інтегрування фреймворку Qt не старіше версії 5.12.

Системні програмні засоби та утиліти повинні забезпечуватись операційною системою, яка відповідає вимогам технічних засобів.

Вимоги до захисту інформації і програми не пред’являються.

1.5 Вимоги до програмної документації

По закінченню розробки програмного забезпечення потрібно підготувати таку документацію:

- інструкція інсталяції програми;
- загальні відомості про можливості програми;
- інструкція з експлуатації.

1.6 Техніко-економічні показники

Розрахунок економічної ефективності і вартості розробки програмного продукту не проводиться.

Приблизне число використань програми в рік визначити не представляється можливим і є індивідуальним для кожного користувача в залежності від його потреб.

1.7 Стадії та етапи розробки

Розробка конструктора є комплексною і підходить під визначення каскадної моделі розробки. Весь процес розробки програмного забезпечення буде містити такі стадії:

– Аналіз.

Перший етап являє собою безпосередній аналіз поставленої задачі та її предметної області. Формується загальне уявлення, як програма повинна працювати та взаємодіяти з користувачем, як саме вона повинна задовольняти вимогам.

– Проєктування.

На даному етапі формуються макети, вже реальна реалізація того, як приблизно зовнішньо має виглядати графічний інтерфейс програми, розробляються класи, моделюються зв’язки між ними. Застосовуються такі засоби ООП, як спадкування, інтерфейс, поліморфізм, створюється ієрархія класів. Визначається тип підключення об’єктів класів – статичний чи динамічний. Формується структура бази даних та загальний вигляд її таблиць.

– Реалізація.

Логіка та функціонал послідовно описуються мовою програмування, для їх реалізації застосовуються різні засоби C++ та фреймворку Qt. Написаний код на даному етапі повинен поверхнево відповідати вимогам та загальній парадигмі ООП, важливим принципам та/або патернам проєктування, але все ж допускаються виключення, пріоритет робиться на реалізації функціоналу.

– Рефакторинг.

Програмний код переглядається та оптимізується, де це потрібно – переписується. Усі іменування об’єктів, змінних та інших ідентифікаторів даних за потреби перейменовуються так, щоб код «пояснював сам себе». За потреби додаються коментарі. Відбувається усунення дубльованого коду, спрощення невиправдано ускладнених блоків. В останню чергу переглядається та за потреби редагується зовнішня частина програми – вікна графічного інтерфейсу, їх дизайн.

– Тестування.

Останній етап передбачає проходження розробленим ПЗ тестування і налагодження: на вхід подаються найбільш різноманітні вхідні дані, оглядається реакція програми на них, обробка некоректних дій користувача. Аналізується, наскільки програма відповідає вимогам і реалізовує потрібний функціонал. За потреби до програмного коду вносяться корективи. Для пошуку помилкової поведінки програми використовуються попередження компілятора, зневаджувач.

Після проходження етапу тестування оформлюється пояснювальна записка.

1.8 Порядок контролю та прийому

Прийом розробленого програмного забезпечення повинен відбуватися на об’єкті Замовника в терміни, які зазначені в індивідуальному завданні.

Для прийому роботи Виконавець повинен представити:

– діючу програму, яка повністю відповідає даному технічному завданню;

– вихідний програмний код, записаний разом із програмою на носій інформації.

Прийом програмного забезпечення повинен відбуватися перед комісією з двох чоловік (один з яких - Замовник) у такій послідовності:

– доповідь Виконавця про виконану роботу;

– демонстрація Виконавцем роботи програми;

– контрольні випробування роботи програми;

– відповіді на запитання і зауваження комісії.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка загальної структури і варіантів використання програми

Даний та наступні підрозділи послідовно описують розробку ПЗ згідно каскадної моделі розробки та попередньо зазначених етапів.

Перш за все стояла задача дослідити предметну область програми, а саме комп'ютерну інженерію або архітектуру. Було набуто нові знання і закріплено вже існуючі з галузі будови ПК, після чого визначалися ключові питання, відповіді на яких дають змогу моделювати вже безпосередньо взаємодію програми та користувача:

– хто буде виступати в ролі суб'єктів програми, скільки їх буде та яка буде їхня роль, як програма буде з ними взаємодіяти;

– хто буде виступати в ролі користувачів та скільки їх буде;

– який функціонал повинна надавати програма користувачу та у який спосіб.

Послідовно на ці питання було дано відповіді: в ролі діючих суб'єктів (надалі - користувачів) буде виступати будь-яка людина чи група людей, оскільки цільова категорія користувачів є широкою, як було вказано в розділі 1.3. Методом взаємодії програми та користувача був обраний інтерфейс як найбільш зручний засіб, який надає користувачу можливість керування програмою без випадкового внесення змін в конструктор та аналізом, виправленням некоректного користувацького введення.

Також, суб'єктом програми буде операційна система, яка і повинна забезпечити функціонування програми на комп'ютері. Засобом взаємодії програми та ОС виступає мова програмування та/або фреймворк, які дозволяють використовувати системний функціонал та керувати ним для вирішення задачі.

Як і було вказано, в ролі користувачів може виступати будь хто, і оскільки задача не передбачає створення контролю облікових записів (наприклад, адміністратор - користувач) та їх контролю, то з цього випливає, що користувачів

може бути декілька і програма повинна забезпечувати багатокористувацький доступ до функціоналу.

Програма повинна давати користувачу такі можливості згідно технічного завдання:

– створення збірки ПК, її редагування, видалення, перегляд;
– створення декількох збірок ПК та навігація між ними;
– внесення, видалення, редагування, перегляд комплектуючих;
– перевірка сумісності обраних комплектуючих збірки;
– запобігання некоректним діям користувача;
– надійну та безперебійну роботу програми;
– вираховування потужності та ціни кожної збірки.

В свою чергу, користувач буде виконувати такі дії для взаємодії з конструктором, і деякі з них можна розбити на підпункти, як на рисунку 2.1:

– запуск і вимкнення програми;
– створення, редагування, видалення збірок;
– створення, редагування, видалення комплектуючих;
– розрахування потужності та ціни збірки.

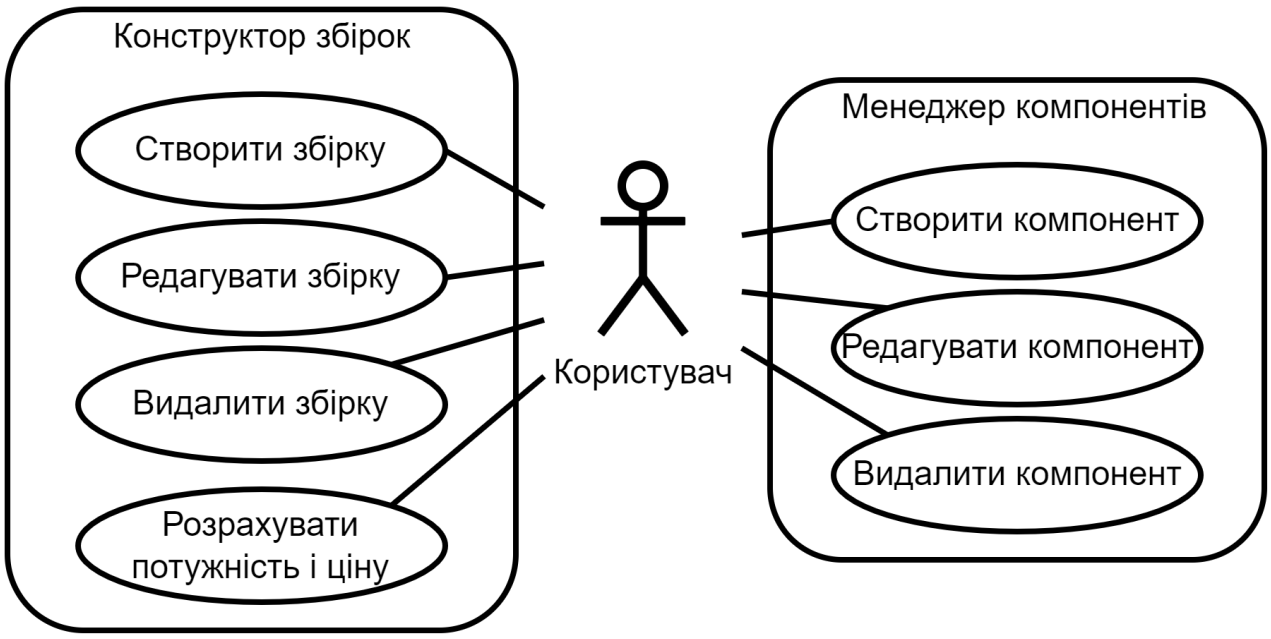

Рисунок 2.1 – Діаграма варіантів використання програми PC Constructor

Кожен з вказаних варіантів потрібно описати, щоб розуміти, як їх розробляти засобами мови програмування.

PC Constructor:

– ***Запустити програму.*** Точка входження у програму. Повинен вивестись інтерфейс Конструктора з можливістю переходу у наступні варіанти або в Менеджер компонентів і його використання.

– ***Вимкнути програму.*** Програма закінчує своє виконання, зберігаючи всі дані та звільняючи зайняту оперативну пам’ять.

Конструктор збірок:

– ***Створити збірку.*** Користувачу пропонується створити збірку і назвати її. Надалі йдуть різні сценарії: якщо користувач вводить назву і погоджується створити збірку і вона створюється, то створена збірка стає активною і з нею можна працювати. Якщо користувач відмовляється, або назва є некоректною, або така збірка вже існує, то збірка не створюється і програма повертається до початкового вікна.

– ***Редагувати збірку.*** При виборі збірки з вже існуючих або після створення збірки така збірка стає активною. Інтерфейс надає елементи для додавання, зміни, видалення компонентів. Сценарії: якщо змінити збірку, то така збірка стає активною, а попередня залишається в так званому «сплячому стані», щоб можна було повернутися для роботи з нею. Послідовно додаються компоненти, якщо якийсь з них не є сумісним, то користувач отримує помилку, якщо він вибере сумісний компонент, то зможе далі збирати ПК. Збірку можна закрити і тоді активної збірки не буде. Виводиться інформація про готовність збірки. Недоступно, якщо немає активної збірки.

– ***Видалити збірку.*** Програма питає користувача, чи точно він хоче видалити збірку. Якщо він погоджується, то збірка видаляється і інформація про неї стирається, а активною збіркою стає остання «спляча» збірка, або, якщо таких немає, то жодна. Якщо відмовляється, то збірка не видаляється і можна продовжувати роботу. Недоступно, якщо немає активної збірки.

– ***Розрахувати потужність і ціну.*** Виводиться інформація про потужність і ціну активної збірки з компонентами, які вже є обраними. Недоступно, якщо немає активної збірки.

Менеджер компонентів:

– ***Створити компонент.*** Попередньо обирається активна група компонентів. Створюється компонент, дані якого беруться зі створених у базі. Якщо компонент з таким ім’ям вже існує, то компонент не створиться.

– ***Редагувати компонент.*** Користувач може змінювати характеристики компонента, проте, якщо він змінить його найменування і воно буде повторювати вже існуюче у базі іншого компонента, то зміна не внесеться. Недоступно, якщо нема активної групи компонентів.

– ***Видалити компонент.*** З’являється вікно, яке питає, чи точно користувач хоче видалити компонент. Якщо він погоджується, компонент видаляється, в іншому випадку залишається. Недоступно, якщо нема активної групи компонентів.

На даному етапі також доцільним є сформувати UML-діаграми дій цих сценаріїв для того, щоб більш конкретизувати їх поведінку та спросити їх реалізацію, як видно на рисунку 2.2 і рисунку 2.3.

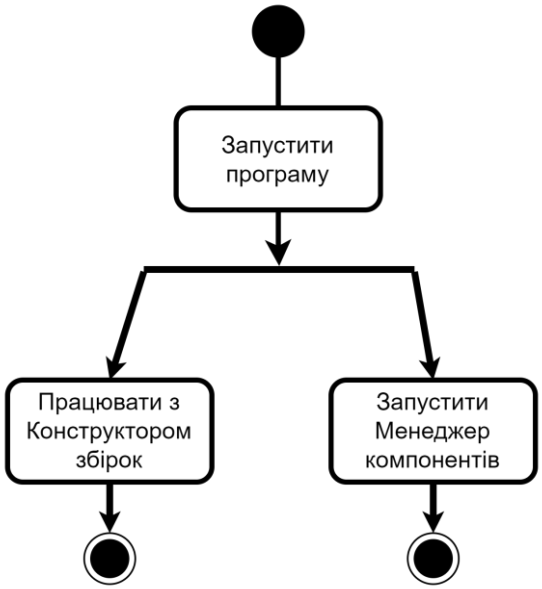

Рисунок 2.2 – Діаграма дій при запуску програми

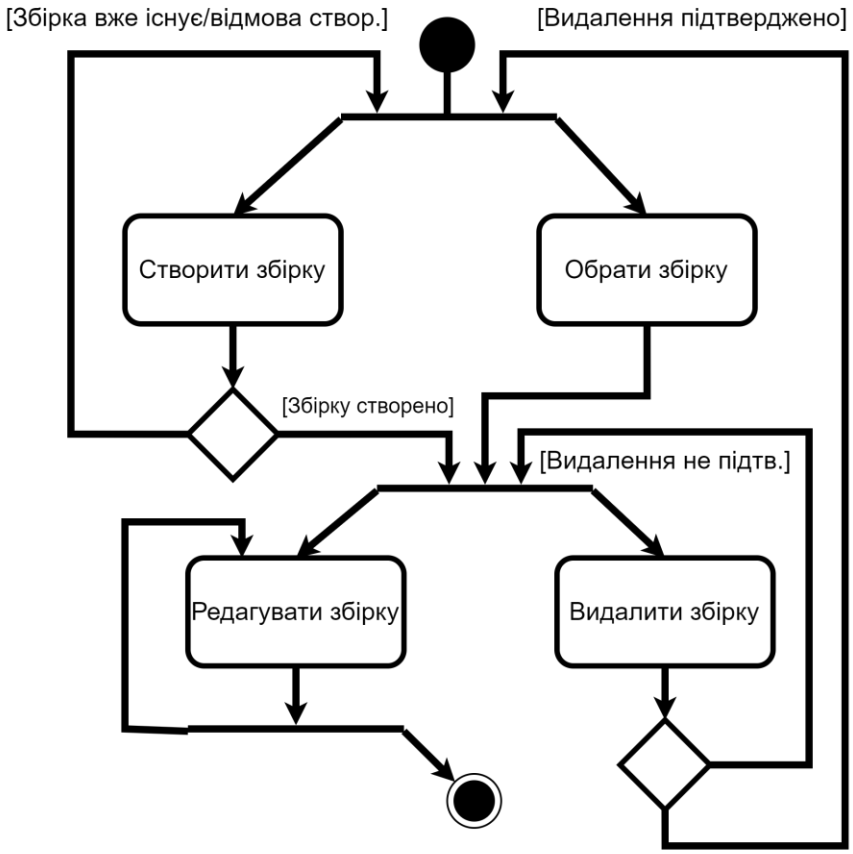

Рисунок 2.3 – Діаграма дій в Конструкторі збірок

2.2 Розробка системи класів

Визначивши загальну модель поведінки програми, будується UML-діаграма класів, яка зображуватиме об’єктно-орієнтовану будову програми та зв’язки між її класами.

Згідно розробленої структури, записуємо ключові ідентифікатори, які будуть використовуватися в програмі та взаємодіяти між собою:

- Головне вікно конструктора;
- База даних, таблиці;
- Менеджер компонентів;
- Діалогове вікно створення збірки;
- Активне ім’я, ідентифікатор збірки;
- Модель таблиці;
- Віджет, список компонентів;

– Конфлікт компонентів;
– Активне ім’я, тип, ідентифікатор компоненту;
– Віджет-переглядач характеристик компоненту;
– Повідомлення про помилки.

Діаграма будується за допомогою програми StarUML, вказуються зв’язки між класами: наслідування, агрегація, композиція, залежність. Записуються усі атрибути та методи, їх модифікатор доступу, чи є той чи інший метод статичним, віртуальним. Створену діаграму класів наведено на рисунку 2.4.

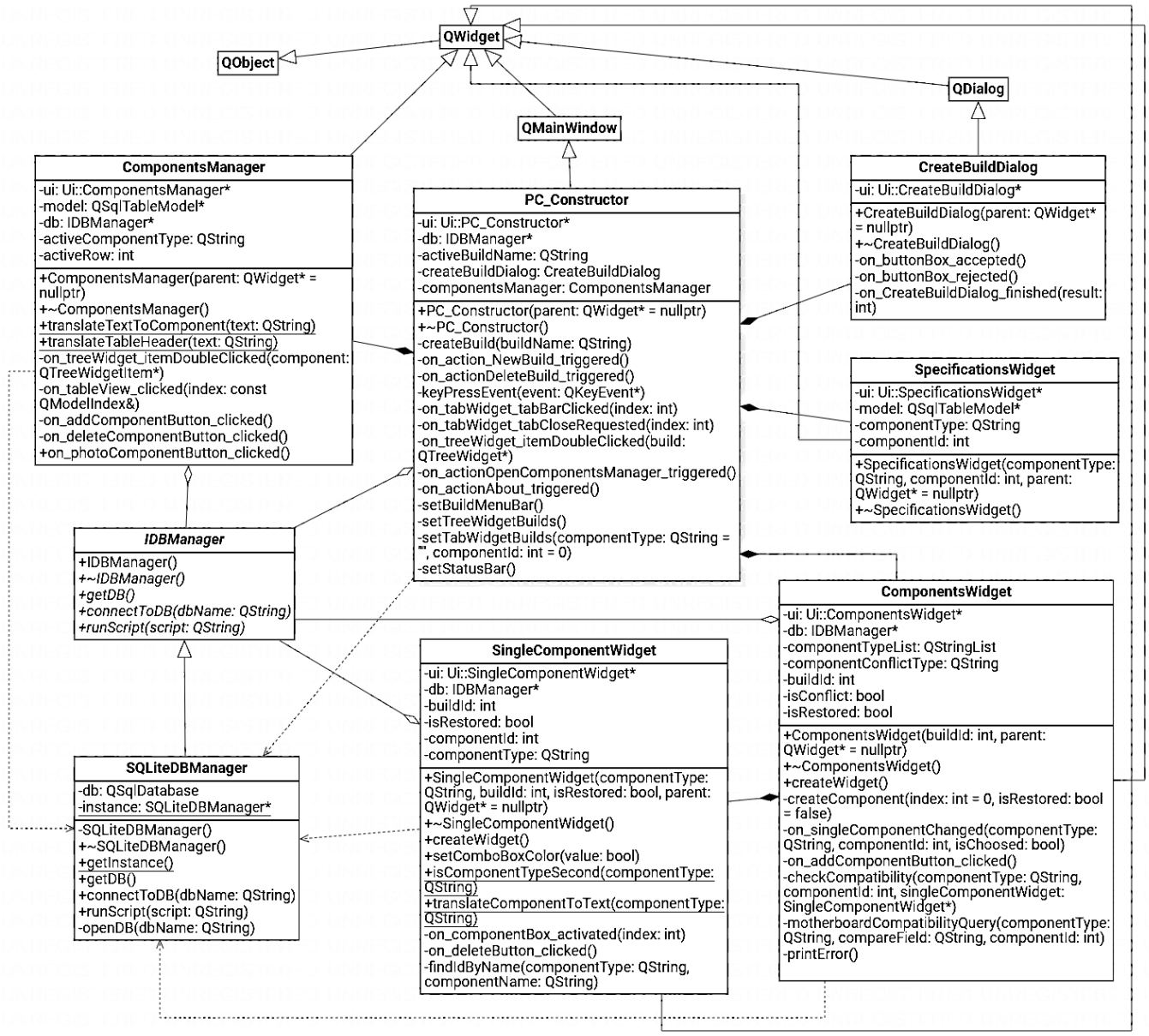

Рисунок 2.4 – UML-діаграма класів програми PC Constructor

Згідно моделі та спроектованих сценаріїв взаємодії користувача і програми, роботи програми, можна виділити, що визначені раніше головні іменники програми сформувалися в одну монолітну структуру і є взаємозв'язаними: наприклад, всі класи, які працюють з базами даних, використовують як оболонку для доступу до бази даних інтерфейс IDBManager, який, в свою чергу, має свою реалізацію у виді похідного класу SQLiteDBManager, що видно на рисунку 2.4. Оскільки інтерфейс – лише сигнал для програми, що є деяка абстракція, яка дає доступ до бази даних, проте існує й інша реалізація, в нашому випадку похідний клас, динамічний об'єкт якого і буде записуватись у вказівник типу IDBManager. Це є зручним, якщо, наприклад, потрібно буде замінити СКБД SQLite на, наприклад, MySQL: достатньо буде лише створити окрему реалізацію інтерфейсу та підключити її один-два рази, не шукаючи слідів іншого менеджеру по всій програмі.

Наглядно видно, що класи реалізовують свій функціонал модульно і кожен має свою зону відповідальності: наприклад, ComponentsManager лише надає можливість працювати безпосередньо з базою даних, а ComponentsWidget і SpecificationsWidget – переглядати та редагувати збірки. І лише в головному класі PC_Constructor вони організовуються у єдину систему.

На основі діаграм, наведених на рис. 2.2, 2.3 і 2.4 вже представляється можливим написання коду програми та реалізації задумки, вирішення задачі.

Створюється проект в інтегрованому середовищі розробки Qt Creator версії 5.0.1, обирається робочим фреймворк Qt версії 6.1 та стандарт мови C++17. Відповідно до визначених моделей створюються C++-класи у вигляді відповідних .h (заголовних, надалі - хедерів) та .cpp (реалізації) файлів [1, с. 365].

Послідовно у хедери вносяться оголошення класів, їх атрибутів і методів, параметрів, вказується тип значень, який повертає кожен метод.

Таким чином, UML-діаграма та діаграма дій реалізовуються вже в програмному коді, де вже більш конкретизована структура програми така: головним вікном, з якого починається вхід у програму, є клас PC_Constructor, який

наслідує QMainWindow і той в свою чергу QWidget. Причому вікно не містить весь безпосередній функціонал програми, а лише надає користувачу доступ до інших віджетів, які вже цей функціонал надають, зв'язує їх. Є одне діалогове вікно CreateBuildDialog, який наслідує QDialog, та всі інші функціональні одиниці конструктора – безпосередньо наслідують QWidget та за задумкою повинні працювати не у ролі діалогових чи окремих вікон (окрім ComponentsManager), а ці віджети «вмонтовуються» безпосередньо в вікно PC_Constructor, який, по суті, є і контейнером віджетів [3, гл. 5, с. 125].

2.3 Розробка методів

У файлах реалізації створюються відповідно до оголошень тіла методів, а також файл main.cpp, який буде містити головну функцію main() та точку входу для програми, в ролі якої виступає об'єкт класу фреймворку QApplication і нашого класу PC_Constructor [7, гл. 1].

Проте, що не дуже є очевидним, реалізація програми починається не з класу головного вікна, а реалізації інтерфейсу бази даних SQLiteDBManager: це пояснюється тим, що, оскільки для вирішення технічної задачі буде достатньо лише однієї бази даних, то і об'єкту, який буде надавати до неї доступ, достатньо буде лише одного (що перегукується з шаблоном проектування Singleton(Одинак)) [6, Singleton pattern 23.07.2010]. Віртуальні методи батьківського класу реалізовуються згідно задумки та створюються декілька допоміжних закритих методів, які усовують деяке дублювання коду і будуть використовуватися безпосередньо екземпляром класу. Конструктор класу при цьому також робиться закритим, а для реалізації одиничного екземпляру класу використовується метод SQLiteDBManager::getInstance(), який повертає статичний вказівник на цей клас.

Наступним кроком є побудова методів класу ComponentsManager: оскільки підключення до бази даних повинно відбуватися лише один раз, з метою запобігання накладання одне на одного підключень з різних методів, то метод SQLiteDBManager::connectToDB() (з вказанням назви файлу-бази, оскільки SQLite

– суто клієнтна СКБД [8, гл. 3], яка працює в межах файлу/файлів) викликається тільки в конструкторі менеджеру компонентів. Тут же і створюються таблиці компонентів, які будуть містити характеристики комплектуючих, потрібних для користувача, їх порівняння, визначення сумісності, про що буде детальніше описано у розділі 2.6. Тут же реалізовується доступ до таблиць компонентів бази даних і можливість їх додавати, редагувати та видаляти.

Наступним на черзі програмується коротеньке діалогове вікно CreateBuildDialog, який приймає від користувача введену назву нової збірки і буде її повертати [5, QDialog Class].

Після цього описується SingleComponentWidget, який буде собою представляти вибір у збірці одного компонента і його запис у збірку (яка пізніше буде представлена як рядок в таблиці збірок тієї ж БД) і моделюються сигнали, які будуть відправлятися в клас ComponentsWidget, що й буде містити до восьми таких віджетів і буде перевіряти їх сумісність. Обрані компоненти записуються у рядок активної збірки у таблиці.

ComponentsWidget вже описується на основі сигналів від попереднього класу віджетів, реалізовуються слоти, які приймають ці сигнали та виконують відповідну перевірку сумісності компонентів. Також моделюється зовнішній вигляд цього віджету і реалізовується динамічне існування об’єктів класу SingleComponentWidget. Цей клас також буде містити сигнали і буде їх надсилати вже у головне вікно, наприклад, з результатами перевірки сумісності – щоб їх можна було виводити не тільки у діалогове вікно, а й у інші елементи інтерфейсу. Таким пізніше обраним буде рядок стану.

Передостаннім пишеться SpecificationsWidget, процес роботи над яким є, напевно, одним з найлегших, тому що від нього потребується всього лиш відображати характеристики обраного компоненту та його ілюстрацію.

І, нарешті, головне вікно PC_Constructor всі створені класи об’єднує у собі, пов’язує їхню взаємодію. Реалізовується робота віджетів з елементами графічного інтерфейсу: рядком стану, деревоподібною панеллю, вікном вкладок, рядком меню

– все це дозволяє працювати одночасно над кількома збірками та перемикатися між ними, про що буде описано у наступному розділі. В конструкторі цього класу створюється заключна таблиця збірок, яка буде містити безпосередньо збірки з обраними компонентами та сумарними характеристиками. Створений функціонал налагоджується та інколи коригується, коли знаходяться кращі рішення тих чи інших проблем.

Варто зазначити, що, в основному замість так званого «винайдення велосипеду» використовуються перевірені часом вбудовані методи фреймворку Qt, деякі, більш структурні рішення моделюються самостійно. Написаний код аналізується, за потреби спрощується, дубльовані блоки виносяться у окремі методи. Основним пластом взаємодії віджетів між собою, але при роботі над збірками та компонентами є SQL-запити до бази даних. Один і той самий екземпляр класу для роботи з базою даних використовується в кожному з віджетів і виконує ці самі запити (здебільшого, для додавання інформації в таблиці) та в разі помилок у роботі сповіщає про це в терміналі інтегрованого середовища, а для більш складних і комплексних запитів, які передбачають витяг даних з таблиць, використовуються тимчасові об’єкти класу QSqlQuery [5, QSqlQuery Class].

2.4 Проєктування і опис інтерфейсу користувача

Не всі класи програми взаємодіють з користувачем, з таких є, наприклад, інтерфейс для роботи з БД та його реалізація IDBManager та SQLiteDBManager відповідно, проте з іншими безпосередньо працює користувач конструктора засобом графічного інтерфейсу.

Виходячи з вже змодельованої структури класів і програми, випливає, що основна взаємодія користувача з програмою буде припадати не на певну кількість звичайних чи діалогових вікон, а з динамічними віджетами, які будуть займати одне й те ж місце в інтерфейсі, проте змінювати свій вміст в залежності від зміни активної збірки.

Було прийнято рішення досягти цього, спираючись на приклад веб-

браузерів або середовищ розробки, які містять такі головні елементи інтерфейсу: панель вкладок, панель елементів, рядок меню та рядок стану.

Другий елемент з переліку було застосовано в Менеджері компонентів, як продемонстровано на рисунку 2.5: ліва панель надає навігацію по таблицям компонентів, при цьому вікно залишається фіксованого розміру.

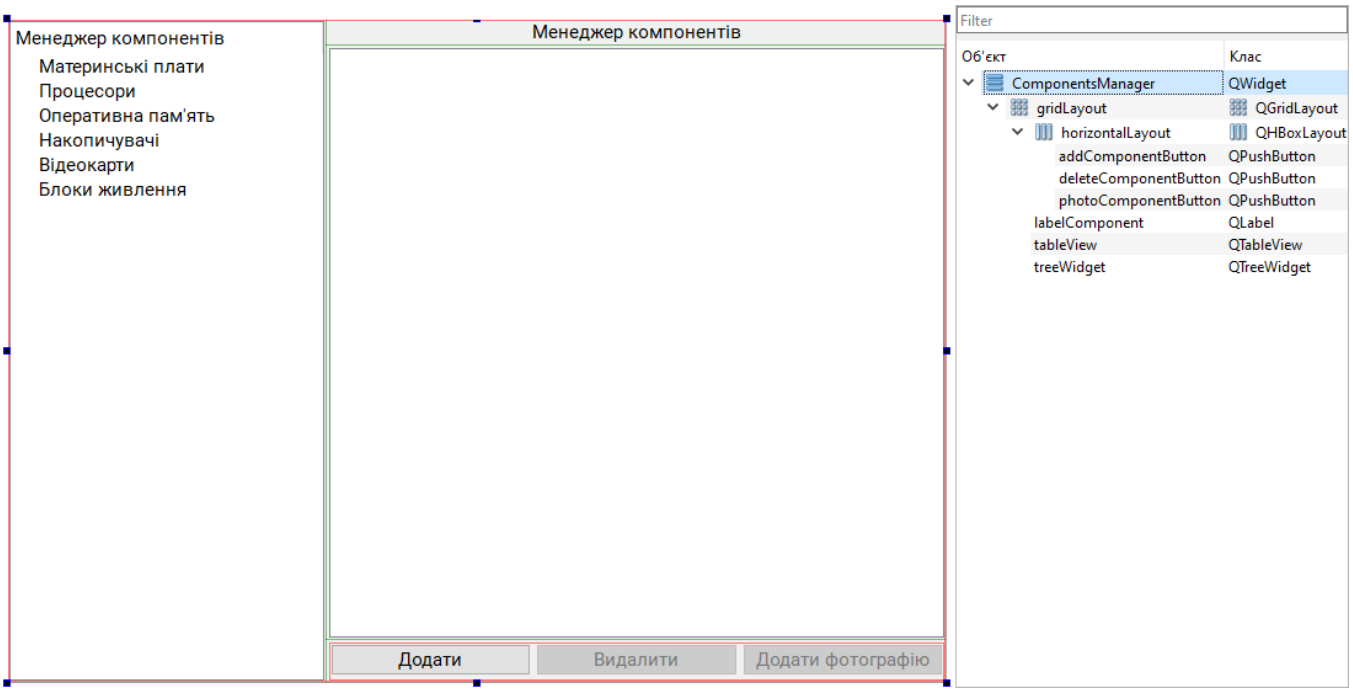

Рисунок 2.5 – Вигляд і будова вікна класу ComponentsManager, яке містить потрібні елементи для роботою з таблицями БД

Для отримання від користувача назви нової збірки використовується діалогове вікно класу CreateBuildDialog, вигляд якого наведено на рисунку 2.6.

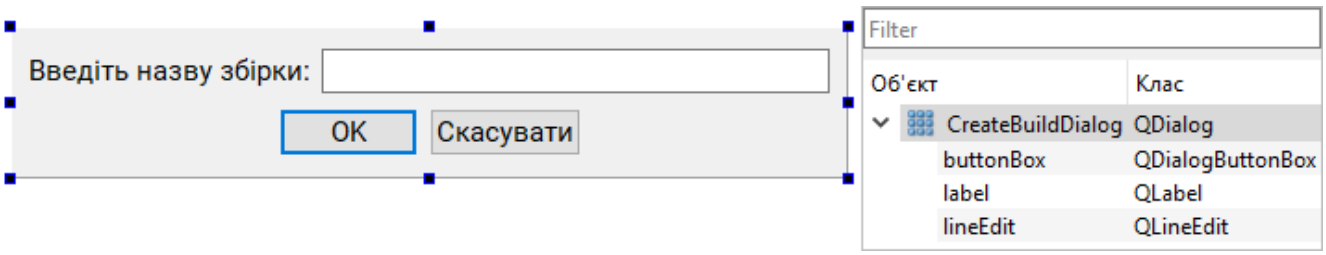

Рисунок 2.6 – Вигляд і будова вікна класу CreateBuildDialog з полем запиту тексту

Оскільки найбільш важливим та функціональним вікном є вікно самого конструктора збірок (див. рис. 2.7), то саме у ньому використовується найбільше елементів графічної бібліотеки фреймворку Qt, і тут же вбудовуються створені віджети SpecificationsWidget, ComponentsWidget, SingleComponentsWidget, які в програмі Qt Designer (надбудова для редагування UI-форм) є порожніми, оскільки елементи в них з'являються динамічно, логіку яких написано у програмному коді (див. лістинги И, І, К).

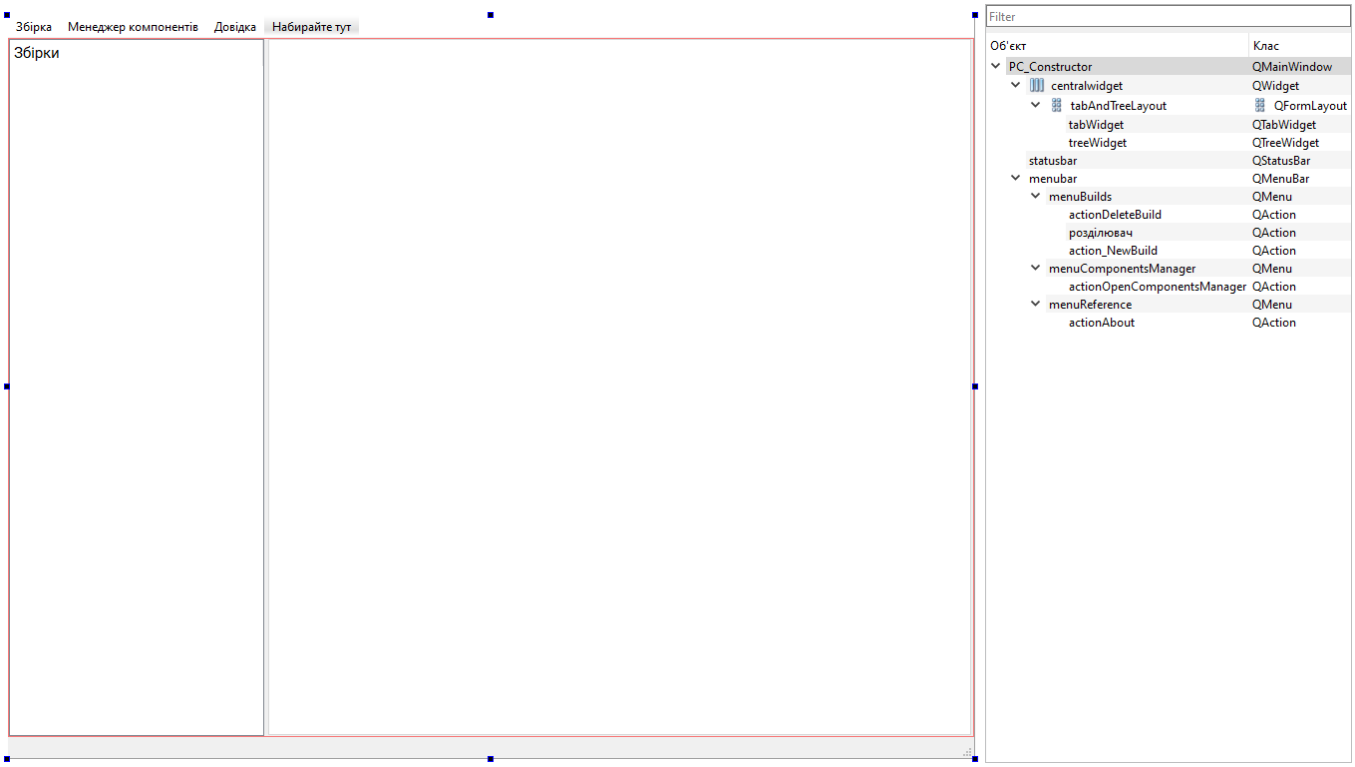

Рисунок 2.7 – Вигляд і будова головного вікна конструктора, яке містить рядки стану і меню, панелі збірок та вкладок

2.5 Опис файлової структури програми

Вихідний код програми представлений в кількості 18 файлів – 8 файлів реалізації та оголошень відповідно для кожного класу, ще одного файлу реалізації main.cpp, який є вхідною точкою програми і не належить до жодного класу, і файлу проекту pc_constructor.pro, який інтегроване середовище розробки Qt Creator використовує для зв'язування усіх інших файлів у єдину структуру. Лістинги

відповідних файлів надано у додатках А-Н. Також, база даних міститься у файлі pc_constructor_db.sqlite і зображення компонентів у папці img з форматами png або jpg/jpeg, окрім цього, користувач може визначити і власні файли зображень та вказати шлях до них.

2.6 Опис структури бази даних програми

В загальному, база даних містить усього 7 таблиць: 6, які містять компоненти, і остання містить збірки. При цьому жодні з цих таблиць не є зв’язаними, оскільки кожен їхній елемент належить лише цій таблиці і ніякій більше, а таблиця збірок взагалі лише посилається на ідентифікатори певних комплектуючих. Це пояснюється тим, що різні збірки можуть мати один і той самий процесор або материнську плату, але при їх видаленні цей самий компонент повинен залишатися в базі, тому як такого зв’язування в програмі немає. Далі наведено структури цих таблиць:

Таблиця 2.1 – Збірки (builds)

Номер поля	Назва поля	Тип даних поля
1	id	INTEGER PK AI
2	name	VARCHAR UK
3	cpu	INTEGER
4	ram	INTEGER
5	rom	INTEGER
6	rom2	INTEGER
7	gpu	INTEGER
8	gpu2	INTEGER
9	powerSupply	INTEGER
10	conflict	INTEGER

На наведеній таблиці 2.1 видно, що збірка містить безпосередньо числові ідентифікатори компонентів і таким чином на них посилається. Для забезпечення унікальності і неповторності збірок поля id і name було модифіковано за допомогою засобів SQLite ключами PRIMARY KEY та UNIQUE KEY відповідно.

З таблиць 2.2 – 2.5 видно, що є сумісність компонентів, а саме процесора, оперативної пам’яті та накопичувача з мат. платою, порівнюючи сокет, тип пам’яті та інтерфейс накопичувача відповідно. Окрім цього, деякі з них містять поля power (потужність) і всі містять ціну, що допоможе розрахувати сумарні характеристики, а також фотографію самого компонента.

Таблиця 2.2 – Материнські плати (motherboard)

Номер поля	**Назва поля**	**Тип даних поля**
1	id	INTEGER PK AI
2	name	VARCHAR UK
3	chipset	VARCHAR
4	socket	VARCHAR
5	ramType	VARCHAR
6	romInterface	VARCHAR
7	price	INTEGER
8	photo	VARCHAR

Таблиця 2.3 – Процесори (cpu)

Номер поля	**Назва поля**	**Тип даних поля**
1	id	INTEGER PK AI
2	name	VARCHAR UK
3	core	INTEGER
4	thread	INTEGER
5	frequency	INTEGER
6	graphics	INTEGER
7	cache	INTEGER
8	socket	VARCHAR
9	power	INTEGER
10	price	INTEGER
11	photo	VARCHAR

Також варто зазначити, що, окрім наведених на таблицях типів даних INTEGER і VARCHAR, що представляють цілі числа та стрічки відповідно, поля id і name містять модифікаторі PK AI і UK. Перший з них – модифікатор PRIMARY KEY AUTOINCREMENT, який робить ідентифікатор первинним ключем таблиці, тобто унікальним і автоінкрементованим (на одиницю більшим від попереднього).

В свою чергу UK – це UNIQUE KEY, який робить поле name також неповторюваним, проте він відрізняється від PRIMARY KEY тим, що перший може містити значення типу NULL, а другий – ні, і в таблиці може бути лише один PK, під роль якого часто відводять саме унікальні числові або хешовані ідентифікатори.

Таблиця 2.4 – Оперативна пам’ять (ram)

Номер поля	**Назва поля**	**Тип даних поля**
1	id	INTEGER PK AI
2	name	VARCHAR UK
3	storage	INTEGER
4	frequency	INTEGER
5	ramType	INTEGER
6	power	VARCHAR
7	price	INTEGER
8	photo	VARCHAR

Таблиця 2.5 – Накопичувачі (rom)

Номер поля	**Назва поля**	**Тип даних поля**
1	id	INTEGER PK AI
2	name	VARCHAR UK
3	type	VARCHAR
4	storage	INTEGER
5	readSpeed	INTEGER
6	writeSpeed	INTEGER
7	romInterface	VARCHAR
8	price	INTEGER
9	photo	VARCHAR

Таблиця 2.6 – Відеокарти (gpu)

Номер поля	**Назва поля**	**Тип даних поля**
1	id	INTEGER PK AI
2	name	VARCHAR UK
3	storage	INTEGER
4	frequency	INTEGER
5	power	INTEGER
6	price	INTEGER
7	photo	VARCHAR

Таблиця 2.7 – Блоки живлення (powerSupply)

Номер поля	**Назва поля**	**Тип даних поля**
1	id	INTEGER PK AI
2	name	VARCHAR UK
3	power	INTEGER
4	price	INTEGER
5	photo	VARCHAR

Варто описати поля сумісності з таблиць 2.3 і 2.6 – якщо, наприклад, певний процесор не має інтегрованого графічного ядра (graphics), то збірка мусить містити відеокарту, і навпаки не повинна, якщо графіка у процесора є. Окрім цього, таблиці 2.3, 2.4, 2.6 та 2.7 мають поле power, і блок живлення з останньої таблиці повинен задовольняти потреби споживання згаданих компонентів. Материнська плата та накопичувачі не враховуються, тому що їх споживання не таке велике, тому з них береться спрощена похибка – 20Вт.

Таким чином, раціональне моделювання бази даних програми спрощує процес розробки та дозволяє уникнути написання непотрібного коду, як-от класу Component, що буде містити інформацію про компонент, або класу Build, який міститиме об’єкти попереднього класу. В цьому просто немає потреби, оскільки робота зі збірками та компонентами відбувається безпосередньо через таблиці бази даних, що економить розмір вихідного коду та оперативну пам’ять, яка могла б бути відведена під об’єкти, кількість яких могла б злічуватися десятками чи сотнями.

3 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ

Програма тестується поступово під час написання коду і перевіряється одразу ж написаний функціонал, проте після закінчення реалізації класів робиться ще одне тестування. Незначні помилки, які не було помічено раніше, усуваються.

При запуску програми створюється єдиний екземпляр головного вікна, з якого можна перейти у Менеджер компонентів, відкрити довідку, створити/редагувати/видалити збірку та їх компоненти (див. рис. 3.1 і рис. 3.2).

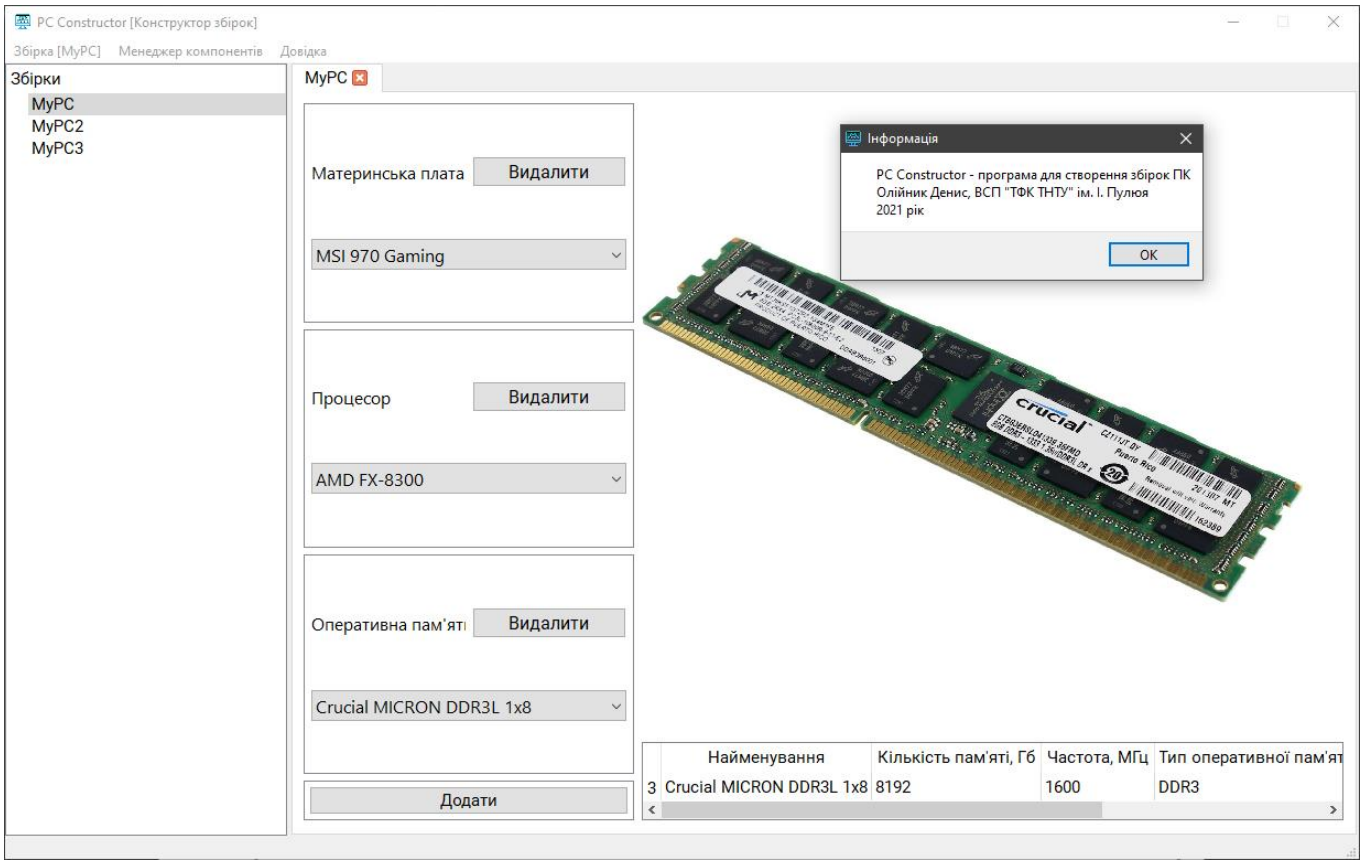

Рисунок 3.1 – Головне вікно конструктора з відкритою збіркою, компонентом і довідкою

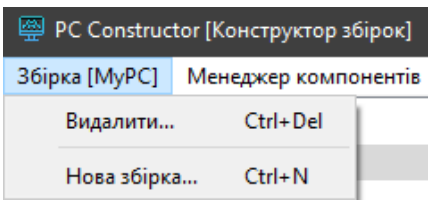

Рисунок 3.2 – Панель з рядку меню з діями створення і видалення збірки

Дії в панелі Збірок рядка меню супроводжуються діалоговими вікнами, як видно на рисунку 3.3 і рисунку 3.4:

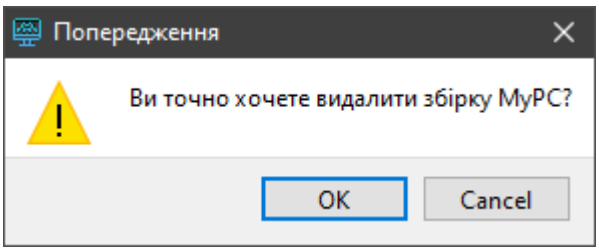

Рисунок 3.3 – Попередження про видалення збірки

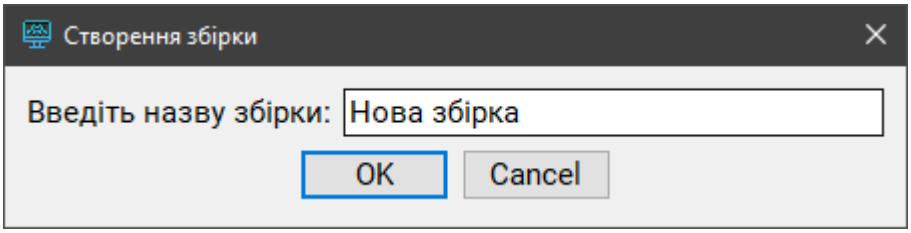

Рисунок 3.4 – Діалогове вікно створення нової збірки

При переході з відповідної дії в меню в Менеджер компонентів відкривається доступ до бази даних компонентів з можливістю їх додавання, видалення, редагування та вибору фотографії:

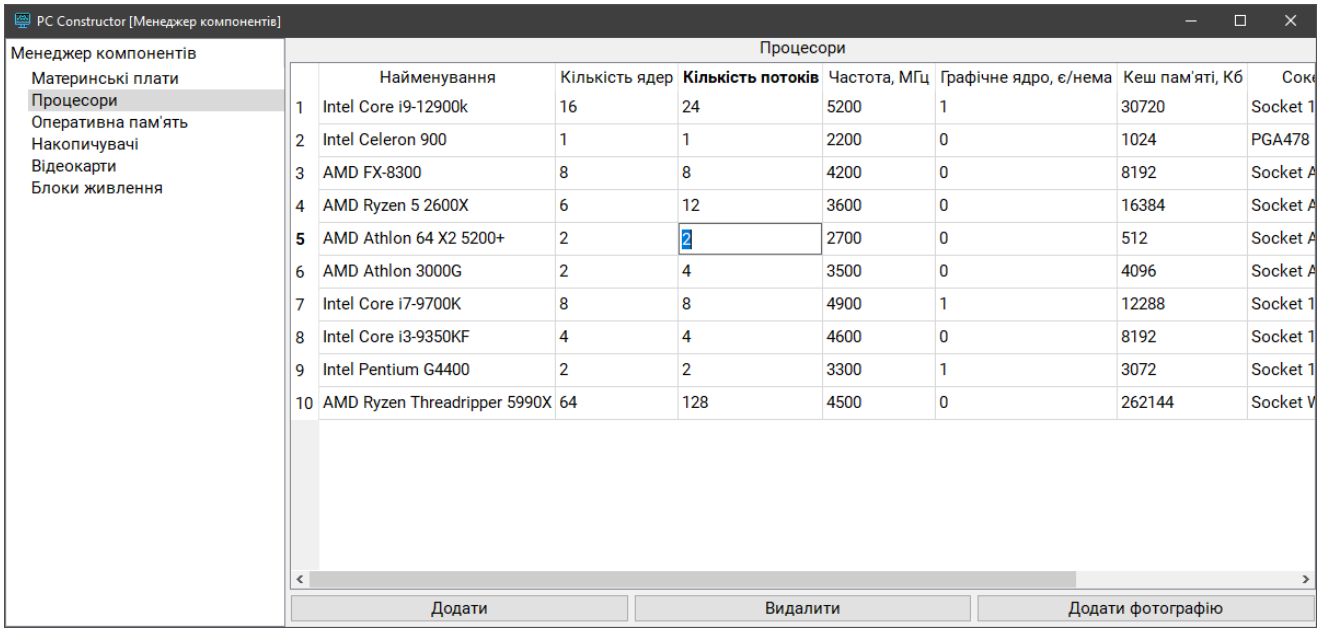

Рисунок 3.5 – Менеджер компонентів

При роботі з конструктором тестується механізм сумісності компонентів (див. рис. 3.6), і, врешті-решт, про успішно зібрану збірку свідчить остання стрічка в рядку стану на рисунку 3.7. Також, ми отримали розраховану потужність і ціну.

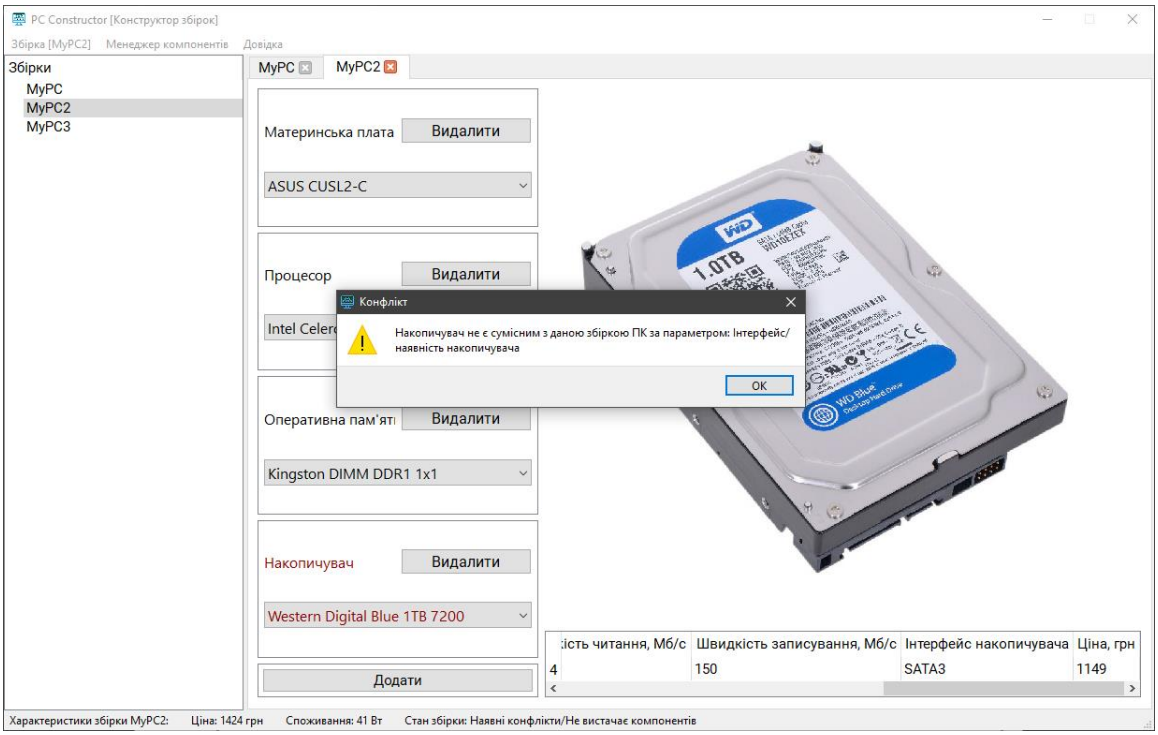

Рисунок 3.6 – Сповіщення про несумісність материнської плати та накопичувача

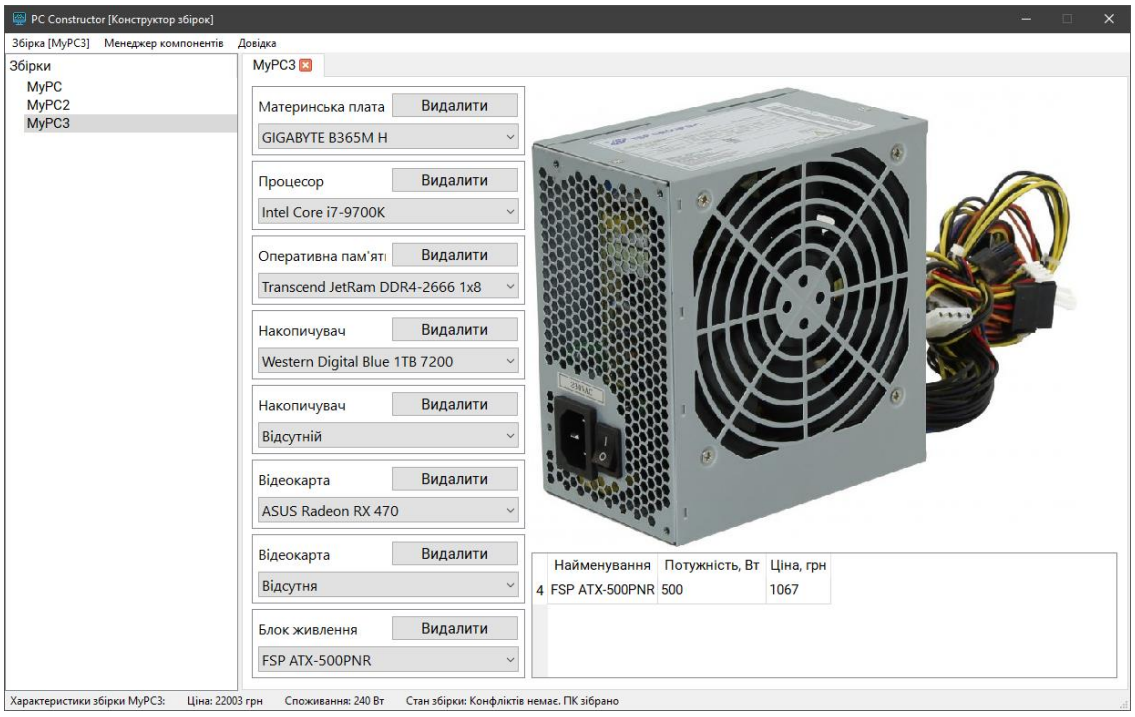

Рисунок 3.7 – Закінчена збірка та її розрахована ціна і потужність

ВИСНОВКИ

Дана курсова робота описує процес розробки програмного забезпечення «Конструктор ПК», який дає змогу створювати власні збірки з комплектуючих та перевіряти їх на сумісність, розраховувати кінцеві характеристики. Розроблене ПЗ відповідає поставленим вимогам ТЗ.

Під час виконання курсової роботи було закріплено і поглиблено знання, отримані за час вивчення дисципліни «Об’єктно-орієнтоване програмування», посилено знання та навички роботи з мовою програмування C++, практично застосовано фреймворк Qt та, частково, бібліотеку STL, яку перевизначає фреймворк, набуто нові знання щодо патернів і принципів ООП, предметної області задачі.

Для розробки даного програмного забезпечення використовувались такі принципи і концепції ООП, як Singleton і DRY, використані інтегровані середовища розробки Microsoft Visual Studio 2022 і Qt Creator 5.0.1, їх функціонал та переваги.

ПЕРЕЛІК ПОСИЛАНЬ

1) Грицюк Ю.І., Рак Т.Є. Об’єктно-орієнтоване програмування мовою C++ : навч. посіб. Львів: Вид-во Львівського ДУ БЖД, 2011. 404 с. (дата звернення: з 24.11.2021 по 28.11.2021)

2) Вайсфельд М. Объектно-ориентированное мышление. СПб.: Питер, 2014. 304 с. (дата звернення: 25.11.2021)

3) Шлее М. Qt 5.3. Профессиональное программирование на C++. СПб.: БХВ-Петербург, 2018. 1074 с. (дата звернення: з 25.11.2021 по 07.12.2021)

4) Веб-сайт EVILEG. URL: https://evileg.com/uk/knowledge/qt (дата звернення: з 27.11.2021 по 07.12.2021)

5) Веб-сайт документація Qt (Qt documentation). URL: https://doc.qt.io (дата звернення: з 25.11.2021 по 07.12.2021)

6) Веб-сайт Stack Overflow. URL: https://stackoverflow.com (дати звернення: 28.11.2021, 30.11.2021)

7) Веб-сайт Ravesli, уроки по Qt. URL: https://ravesli.com/uroki-po-qt5 (дата звернення: з 25.11.2021 по 06.12.2021)

8) Веб-сайт документація SQLite (SQLite documentation). URL: https://www.sqlite.org/docs.html (дати звернення: 28.11.2021, 01.12.2021)

Додаток А

Лістинг файлу «pc_constructor.pro»

```
QT += core gui sql
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
CONFIG += c++17

MOC_DIR = moc
RCC_DIR = rcc
UI_DIR = ui
unix: OBJECTS_DIR = unix
win32: OBJECTS_DIR = win32
macx: OBJECT_DIR = mac

SOURCES += \
    componentsmanager.cpp \
    componentswidget.cpp \
    createbuilddialog.cpp \
    idbmanager.cpp \
    main.cpp \
    pc_constructor.cpp \
    singlecomponentwidget.cpp \
    specificationswidget.cpp \
    sqlitedbmanager.cpp

HEADERS += \
    componentsmanager.h \
    componentswidget.h \
    createbuilddialog.h \
    idbmanager.h \
    pc_constructor.h \
    singlecomponentwidget.h \
    specificationswidget.h \
    sqlitedbmanager.h
```

```
FORMS += \
    componentsmanager.ui \
    componentswidget.ui \
    createbuilddialog.ui \
    pc_constructor.ui \
    singlecomponentwidget.ui \
    specificationswidget.ui

CONFIG(release, debug|release) {
    win32: QMAKE_POST_LINK = $$(QTDIR)/bin/windeployqt $$OUT_PWD/release
}

RC_ICONS = pc_constructor_icon.ico
```

Додаток Б

Лістинг файлу «main.cpp»

```
#include <QApplication>
#include "pc_constructor.h"

int main(int argc, char *argv[])
{
    QApplication application(argc, argv);
    PC_Constructor mainWindow;

    mainWindow.show();

    return application.exec();
}
```

Додаток В

Лістинг файлу «idbmanager.h»

```
#ifndef IDBMANAGER_H
#define IDBMANAGER_H

#include <QSqlDatabase>

//  Інтерфейс менеджера бази даних
class IDBManager
{
public:
    IDBManager();
    virtual ~IDBManager() = 0;

    virtual QSqlDatabase getDB() = 0;
    virtual bool connectToDB(QString dbName) = 0;
    virtual bool runScript(QString script) = 0;
};

#endif // IDBMANAGER_H
```

Додаток Г

Лістинг файлу «idbmanager.cpp»

```
#include "idbmanager.h"

IDBManager::IDBManager() {}

IDBManager::~IDBManager() {}
```

Додаток Ґ

Лістинг файлу «sqlitedbmanager.h»

```
#ifndef SQLITEDBMANAGER_H
#define SQLITEDBMANAGER_H

#include "idbmanager.h"

//  Реалізація інтерфейсу менеджера бази даних
//  Менеджер бази даних SQLite
class SQLiteDBManager : public IDBManager
{
public:
    virtual ~SQLiteDBManager() override;
    static SQLiteDBManager *getInstance();

    virtual QSqlDatabase getDB() override;
    virtual bool connectToDB(QString dbName) override;
    virtual bool runScript(QString script) override;

private:
    SQLiteDBManager();

    QSqlDatabase db;
    static SQLiteDBManager *instance;

    bool openDB(QString dbName);
};

#endif // SQLITEDBMANAGER_H
```

Додаток Д

Лістинг файлу «sqlitedbmanager.cpp»

```
#include "sqlitedbmanager.h"
#include <QDir>
#include <QSqlError>
#include <QSqlQuery>

SQLiteDBManager *SQLiteDBManager::instance = nullptr;

SQLiteDBManager::SQLiteDBManager() {}

SQLiteDBManager::~SQLiteDBManager() {}

SQLiteDBManager *SQLiteDBManager::getInstance()
{
    return (instance ?: instance = new SQLiteDBManager);
}

QSqlDatabase SQLiteDBManager::getDB()
{
    return db;
}

bool SQLiteDBManager::connectToDB(QString dbName)
{
    return openDB(dbName);
}

bool SQLiteDBManager::runScript(QString script)
{
    QSqlQuery tempQuery;
    bool result = tempQuery.exec(script);
```

```
    if (!result)
        qDebug()    <<    tempQuery.lastError()    <<    Qt::endl    <<
tempQuery.lastQuery() << Qt::endl;

    return result;
}

bool SQLiteDBManager::openDB(QString dbName)
{
    db = QSqlDatabase::addDatabase("QSQLITE");
    if (!QDir("db").exists())
        QDir().mkdir("db");

    db.setDatabaseName(QString("db//%1.sqlite").arg(dbName));
    return db.open();
}
```

Додаток Е

Лістинг файлу «createbuilddialog.h»

```
#ifndef CREATEBUILDDIALOG_H
#define CREATEBUILDDIALOG_H

#include <QDialog>

namespace Ui {
class CreateBuildDialog;
}

//  Клас-діалогове вікно, яке запитує ім'я для створення збірки
class CreateBuildDialog : public QDialog
{
    Q_OBJECT

public:
    explicit CreateBuildDialog(QWidget *parent = nullptr);
    ~CreateBuildDialog();

signals:
    void getBuildName(QString buildName);

private slots:
    void on_buttonBox_accepted();
    void on_buttonBox_rejected();
    void on_CreateBuildDialog_finished(int result);

private:
    Ui::CreateBuildDialog *ui;
};

#endif // CREATEBUILDDIALOG_H
```

Додаток Є

Лістинг файлу «createbuilddialog.cpp»

```
#include "createbuilddialog.h"
#include "ui_createbuilddialog.h"

CreateBuildDialog::CreateBuildDialog(QWidget *parent) :
    QDialog(parent), ui(new Ui::CreateBuildDialog)
{
    ui->setupUi(this);
    setModal(true);
}

CreateBuildDialog::~CreateBuildDialog()
{
    delete ui;
}

void CreateBuildDialog::on_buttonBox_accepted()
{
    emit getBuildName(ui->lineEdit->text());
    ui->lineEdit->clear();
}

void CreateBuildDialog::on_buttonBox_rejected()
{
    ui->lineEdit->clear();
}

void CreateBuildDialog::on_CreateBuildDialog_finished(int result)
{
    if (!result)
        ui->lineEdit->clear();
}
```

Додаток Ж

Лістинг файлу «componentsmanager.h»

```
#ifndef COMPONENTSMANAGER_H
#define COMPONENTSMANAGER_H

#include <QSqlTableModel>
#include <QTreeWidgetItem>
#include <QWidget>
#include "sqlitedbmanager.h"

namespace Ui {
class ComponentsManager;
}

//  Клас-вікно, який реалізовує графічний менеджер бази даних і
//  дозволяє вносити/редагувати/видаляти компоненти з таблиць БД
class ComponentsManager : public QWidget
{
    Q_OBJECT

public:
    explicit ComponentsManager(QWidget *parent = nullptr);
    ~ComponentsManager();

    static QString translateTextToComponent(QString text);
    static QString translateTableHeader(QString text);

private slots:
    void on_treeWidget_itemDoubleClicked(QTreeWidgetItem *component);
    void on_tableView_clicked(const QModelIndex &index);
    void on_addComponentButton_clicked();
    void on_deleteComponentButton_clicked();
    void on_photoComponentButton_clicked();
```

```
private:
    Ui::ComponentsManager *ui;
    QSqlTableModel *model;
    IDBManager *db;

    QString activeComponentType;
    int activeRow;
};

#endif // COMPONENTSMANAGER_H
```

Додаток З

Лістинг файлу «componentsmanager.cpp»

```
#include "componentsmanager.h"
#include <QDir>
#include <QFileDialog>
#include <QMessageBox>
#include "ui_componentsmanager.h"

ComponentsManager::ComponentsManager(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::ComponentsManager)
    , model(nullptr)
    , db(SQLiteDBManager::getInstance())
    , activeRow(0)
{
    ui->setupUi(this);
    db->connectToDB("pc_constructor_db");
    db->runScript("CREATE TABLE motherboard"
                  "("
                  "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                  "name VARCHAR NOT NULL, "
                  "chipset VARCHAR NOT NULL, "
                  "socket VARCHAR NOT NULL, "
                  "ramType VARCHAR NOT NULL, "
                  "romInterface VARCHAR NOT NULL, "
                  "price INTEGER NOT NULL, "
                  "photo VARCHAR, "
                  "UNIQUE(name)"
                  ")");
    db->runScript("CREATE TABLE cpu"
                  "("
                  "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                  "name VARCHAR NOT NULL, "
```

```
                "core INTEGER NOT NULL, "
                "thread INTEGER NOT NULL, "
                "frequency INTEGER NOT NULL, "
                "graphics INTEGER NOT NULL, "
                "cache INTEGER NOT NULL, "
                "socket VARCHAR NOT NULL, "
                "power INTEGER NOT NULL, "
                "price INTEGER NOT NULL, "
                "photo VARCHAR, "
                "UNIQUE(name)"
                ")");
    db->runScript("CREATE TABLE ram"
                "("
                "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                "name VARCHAR NOT NULL, "
                "storage INTEGER NOT NULL, "
                "frequency INTEGER NOT NULL, "
                "ramType VARCHAR NOT NULL, "
                "power INTEGER NOT NULL, "
                "price INTEGER NOT NULL, "
                "photo VARCHAR, "
                "UNIQUE(name)"
                ")");
    db->runScript("CREATE TABLE rom"
                "("
                "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                "name VARCHAR NOT NULL, "
                "type VARCHAR NOT NULL, "
                "storage INTEGER NOT NULL, "
                "readSpeed INTEGER NOT NULL, "
                "writeSpeed INTEGER NOT NULL, "
                "romInterface VARCHAR NOT NULL, "
                "price INTEGER NOT NULL, "
                "photo VARCHAR, "
```

```
                "UNIQUE(name)"
                ")");
    db->runScript("CREATE TABLE gpu"
                "("
                "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                "name VARCHAR NOT NULL, "
                "storage INTEGER NOT NULL, "
                "frequency INTEGER NOT NULL, "
                "power INTEGER NOT NULL, "
                "price INTEGER NOT NULL, "
                "photo VARCHAR, "
                "UNIQUE(name)"
                ")");
    db->runScript("CREATE TABLE powerSupply"
                "("
                "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                "name VARCHAR NOT NULL, "
                "power INTEGER NOT NULL, "
                "price INTEGER NOT NULL, "
                "photo VARCHAR, "
                "UNIQUE(name)"
                ")");

    connect(ui->treeWidget,
            &QTreeWidget::itemActivated,
            this,
            &ComponentsManager::on_treeWidget_itemDoubleClicked);

    model = new QSqlTableModel(this, db->getDB());
}

ComponentsManager::~ComponentsManager()
{
    delete model;
```

```
    delete ui;
}

QString ComponentsManager::translateTextToComponent(QString text)
{
    if (text == "Материнські плати")
        return "motherboard";
    if (text == "Процесори")
        return "cpu";
    if (text == "Оперативна пам'ять")
        return "ram";
    if (text == "Накопичувачі")
        return "rom";
    if (text == "Відеокарти")
        return "gpu";
    if (text == "Блоки живлення")
        return "powerSupply";
    return text;
}

QString ComponentsManager::translateTableHeader(QString text)
{
    if (text == "name")
        return "Найменування";
    if (text == "chipset")
        return "Чіпсет";
    if (text == "socket")
        return "Сокет";
    if (text == "ramType")
        return "Тип оперативної пам'яті";
    if (text == "romInterface")
        return "Інтерфейс накопичувача";
    if (text == "price")
        return "Ціна, грн";
```

```
    if (text == "photo")
        return "Фотографія";
    if (text == "core")
        return "Кількість ядер";
    if (text == "thread")
        return "Кількість потоків";
    if (text == "frequency")
        return "Частота, МГц";
    if (text == "graphics")
        return "Графічне ядро, є/нема";
    if (text == "cache")
        return "Кеш пам'яті, Кб";
    if (text == "power")
        return "Потужність, Вт";
    if (text == "storage")
        return "Кількість пам'яті, Гб";
    if (text == "type")
        return "Тип";
    if (text == "readSpeed")
        return "Швидкість читання, Мб/с";
    if (text == "writeSpeed")
        return "Швидкість записування, Мб/с";
    return "";
}

void     ComponentsManager::on_treeWidget_itemDoubleClicked(QTreeWidgetItem
*component)
{
    activeComponentType = translateTextToComponent(component->text(0));

    ui->labelComponent->setText(component->text(0));

    model->clear();
    model->setTable(activeComponentType);
```

```
    model->select();

    for (int i = 0; i < model->columnCount(); i++)
        model->setHeaderData(i,
                             Qt::Horizontal,
                             translateTableHeader(model->headerData(i,
Qt::Horizontal).toString()));

    ui->tableView->setModel(model);
    ui->tableView->horizontalHeader()-
>setSectionResizeMode(QHeaderView::ResizeToContents);

    for (int i = 0; i < model->rowCount(); i++)
        if (model->index(i, 1).data().toString().contains("Відсутн")
            && !model->index(i, 3).data().toInt()) {
            ui->tableView->hideRow(i);
            break;
        }

    ui->tableView->hideColumn(0);
    ui->tableView->hideColumn(model->columnCount() - 1);

    if (ui->labelComponent->text() == "Відеокарти")
        model->setHeaderData(2, Qt::Horizontal, "Кількість пам'яті, Мб");
}

void ComponentsManager::on_tableView_clicked(const QModelIndex &index)
{
    if (!ui->deleteComponentButton->isEnabled())
        ui->deleteComponentButton->setEnabled(true);
    if (!ui->photoComponentButton->isEnabled())
        ui->photoComponentButton->setEnabled(true);

    activeRow = index.row();
```

```
}

void ComponentsManager::on_addComponentButton_clicked()
{
    model->insertRow(model->rowCount());
}

void ComponentsManager::on_deleteComponentButton_clicked()
{
    if (QMessageBox::warning(this,
                             "Попередження",
                             QString("Ви точно хочете видалити компонент
%1?")
                                 .arg(model->index(activeRow,
1).data().toString()),
                             QMessageBox::Ok | QMessageBox::Cancel)
            != QMessageBox::Ok
        || !model->removeRow(activeRow))
        return;
    ui->deleteComponentButton->setEnabled(false);
    ui->photoComponentButton->setEnabled(false);
}

void ComponentsManager::on_photoComponentButton_clicked()
{
    QString tempPhotoPath = QFileDialog::getOpenFileName(
        this,
        QString("Оберіть фотографію для %1").arg(model->index(activeRow,
1).data().toString()));
    db->runScript(QString("UPDATE %1 SET photo = '%2' WHERE id = %3")
                      .arg(activeComponentType,
QDir().relativeFilePath(tempPhotoPath))
                      .arg(model->index(activeRow, 0).data().toInt()));
}
```

Додаток И

Лістинг файлу «specificationswidget.h»

```
#ifndef SPECIFICATIONSWIDGET_H
#define SPECIFICATIONSWIDGET_H

#include <QSqlTableModel>
#include <QWidget>

namespace Ui {
class SpecificationsWidget;
}

//  Клас-віджет, який надає і дозволяє переглядати інформацію
//  про взятий компонент, також показує його зображення
class SpecificationsWidget : public QWidget
{
    Q_OBJECT

public:
    explicit SpecificationsWidget(QString componentType, int componentId,
QWidget *parent = nullptr);
    ~SpecificationsWidget();

private:
    Ui::SpecificationsWidget *ui;
    QSqlTableModel *model;

    QString componentType;
    int componentId;
};

#endif // SPECIFICATIONSWIDGET_H
```

Додаток І

Лістинг файлу «specificationswidget.cpp»

```
#include "specificationswidget.h"
#include "componentsmanager.h"
#include "sqlitedbmanager.h"
#include "ui_specificationswidget.h"

SpecificationsWidget::SpecificationsWidget(QString componentType, int
componentId, QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::SpecificationsWidget)
    , model(new QSqlTableModel(this, SQLiteDBManager::getInstance()-
>getDB()))
    , componentType(componentType)
    , componentId(componentId)
{
    ui->setupUi(this);
    ui->label->hide();

    if (!componentId)
        return;

    model->clear();
    model->setTable(componentType);
    model->select();

    for (int i = 0; i < model->columnCount(); i++)
        model->setHeaderData(i,
                             Qt::Horizontal,
                             ComponentsManager::translateTableHeader(
                                 model->headerData(i,
Qt::Horizontal).toString()));
```

```
    ui->tableView->setModel(model);
    ui->tableView->hideColumn(0);
    ui->tableView->horizontalHeader()-
>setSectionResizeMode(QHeaderView::ResizeToContents);

    int tempPhotoIndex = 0;
    for (int i = model->columnCount() - 1; i >= 0; i--)
        if (model->headerData(i, Qt::Horizontal).toString() == "Фотографія")
{
            ui->tableView->hideColumn(i);
            tempPhotoIndex = i;
            break;
        }

    QString tempPhotoPath;
    for (int i = 0; i < model->rowCount(); i++) {
        if (model->index(i, 0).data().toInt() == componentId) {
            tempPhotoPath                  =                  model->index(i,
tempPhotoIndex).data().toString();
            continue;
        }

        ui->tableView->hideRow(i);
    }

    for (int i = 0; i < model->rowCount(); i++)
        if (model->index(i, 1).data().toString().contains("Відсутн")
            &&   !model->index(i,   3).data().toInt()   &&   !ui->tableView-
>isRowHidden(i)) {
            ui->tableView->hide();
            break;
        }

    if (componentType.startsWith("gpu"))
```

```
        model->setHeaderData(2, Qt::Horizontal, "Кількість пам'яті, Мб");

    ui->label->setPixmap(QPixmap(tempPhotoPath).scaled(600,           600,
Qt::KeepAspectRatio));
    ui->label->show();
}

SpecificationsWidget::~SpecificationsWidget()
{
    delete model;
    delete ui;
}
```

Додаток Ї

Лістинг файлу «singlecomponentwidget.h»

```
#ifndef SINGLECOMPONENTWIDGET_H
#define SINGLECOMPONENTWIDGET_H

#include <QVector>
#include <QWidget>
#include "sqlitedbmanager.h"

namespace Ui {
class SingleComponentWidget;
}

//  Клас-віджет, який графічно представляє один взятий компонент збірки
//  дозволяє його змінювати, реагує на несумісність компонентів
class SingleComponentWidget : public QWidget
{
    Q_OBJECT

public:
    explicit SingleComponentWidget(QString componentType,
                                   int buildId,
                                   bool isRestored,
                                   QWidget *parent = nullptr);
    ~SingleComponentWidget();

    void createWidget();
    void setComboBoxColor(bool value);
    static QString isComponentTypeSecond(QString componentType);
    static QString translateComponentToText(QString componentType);

signals:
    void componentCreated(QString componentType,
```

```
                            int componentId,
                            SingleComponentWidget *singleComponentWidget);
    void componentChanged(QString componentType, int componentId = 0, bool
isDeleted = true);
    void specificationsRequest(QString componentType, int componentId);

private slots:
    void on_componentBox_activated(int index);
    void on_deleteButton_clicked();

private:
    Ui::SingleComponentWidget *ui;
    IDBManager *db;

    int buildId;
    bool isRestored;
    int componentId;
    QString componentType;

    int findIdByName(QString componentType, QString componentName);
};

#endif // SINGLECOMPONENTWIDGET_H
```

Додаток Й

Лістинг файлу «singlecomponentwidget.cpp»

```
#include "singlecomponentwidget.h"
#include <QSqlQuery>
#include "ui_singlecomponentwidget.h"

SingleComponentWidget::SingleComponentWidget(QString componentType,
                                             int buildId,
                                             bool isRestored,
                                             QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::SingleComponentWidget)
    , db(SQLiteDBManager::getInstance())
    , buildId(buildId)
    , isRestored(isRestored)
    , componentId(0)
    , componentType(componentType)
{
    ui->setupUi(this);
}

SingleComponentWidget::~SingleComponentWidget()
{
    delete ui;
}

void SingleComponentWidget::createWidget()
{
    QSqlQuery query(db->getDB());
    ui->label->setText(translateComponentToText(componentType));

    query.exec(
        QString("SELECT name FROM %1 ORDER BY id
```

```
ASC").arg(isComponentTypeSecond(componentType)));
    while (query.next())
        ui->componentBox->addItem(query.value(0).toString());

    if (isRestored) {
        query.exec(QString("SELECT %1 FROM builds WHERE id =
%2").arg(componentType).arg(buildId));
        query.next();

        ui->componentBox->setCurrentIndex(query.value(0).toInt() - 1);
    } else {
        query.exec(QString("SELECT id FROM %1 WHERE name LIKE '%2'")
                        .arg(isComponentTypeSecond(componentType),       ui-
>componentBox->itemText(0)));
        query.next();

        db->runScript(QString("UPDATE builds SET %1 = %2 WHERE id = %3")
                            .arg(componentType)
                            .arg(query.value(0).toInt())
                            .arg(buildId));
    }

    componentId = query.value(0).toInt();

    emit componentCreated(componentType, componentId, this);
}

void SingleComponentWidget::setComboBoxColor(bool value)
{
    QString tempStyleSheetColor = (QString("color: %1").arg(value ? "black"
: "darkred"));

    ui->componentBox->setStyleSheet(tempStyleSheetColor);
    ui->label->setStyleSheet(tempStyleSheetColor);
```

```
}

QString SingleComponentWidget::isComponentTypeSecond(QString componentType)
{
    return                    (componentType.endsWith("2")                    ?
componentType.left(componentType.length() - 1)
                                        : componentType);
}

QString             SingleComponentWidget::translateComponentToText(QString
componentType)
{
    if (componentType == "motherboard")
        return "Материнська плата";
    if (componentType == "cpu")
        return "Процесор";
    if (componentType == "ram")
        return "Оперативна пам'ять";
    if (componentType.startsWith("rom"))
        return "Накопичувач";
    if (componentType.startsWith("gpu"))
        return "Відеокарта";
    if (componentType == "powerSupply")
        return "Блок живлення";
    return componentType;
}

void SingleComponentWidget::on_componentBox_activated(int index)
{
    int  tempComponentId  =  findIdByName(componentType,  ui->componentBox-
>itemText(index));

    if (tempComponentId == componentId) {
        emit specificationsRequest(componentType, componentId);
```

```
        return;
    }

    componentId = tempComponentId;
    db->runScript(QString("UPDATE builds SET %1 = %2 WHERE id = %3")
                      .arg(componentType)
                      .arg(componentId)
                      .arg(buildId));

    emit componentChanged(componentType, componentId, false);
}

void SingleComponentWidget::on_deleteButton_clicked()
{
    emit componentChanged(componentType);
}

int SingleComponentWidget::findIdByName(QString componentType, QString
componentName)
{
    QSqlQuery query(db->getDB());
    query.exec(QString("SELECT id FROM %1 WHERE name LIKE '%2'")
                    .arg(isComponentTypeSecond(componentType),
componentName));
    query.next();

    return query.value(0).toInt();
}
```

Додаток К

Лістинг файлу «componentswidget.h»

```
#ifndef COMPONENTSWIDGET_H
#define COMPONENTSWIDGET_H

#include <QWidget>
#include "singlecomponentwidget.h"
#include "sqlitedbmanager.h"

namespace Ui {
class ComponentsWidget;
}

//  Клас-віджет, який містить в собі віджети SingleComponentWidget
//  візуалізовує компоненти збірки і дозволяє їх змінювати
//  перевіряти їх сумісність
class ComponentsWidget : public QWidget
{
    Q_OBJECT

public:
    explicit ComponentsWidget(int buildId, QWidget *parent = nullptr);
    ~ComponentsWidget();

    void createWidget();

signals:
    void conflictResult(QStringList componentTypes);
    void clearWidget(QString componentType, int componentId);
    void lastComponentCreated();

private slots:
    void createComponent(int index = 0, bool isRestored = false);
```

```
    void on_singleComponentChanged(QString componentType, int componentId,
bool isChoosed);
    void on_addComponentButton_clicked();

private:
    Ui::ComponentsWidget *ui;
    IDBManager *db;

    QStringList componentTypeList;
    QString componentConflictType;
    int buildId;
    bool isConflict;
    bool isRestored;

    void checkCompatibility(QString componentType,
                            int componentId,
                            SingleComponentWidget *singleComponentWidget);
    bool   motherboardCompatibilityQuery(QString   componentType,   QString
compareField, int componentId);
    bool printError();
};

#endif // COMPONENTSWIDGET_H
```

Додаток Л

Лістинг файлу «componentswidget.cpp»

```
#include "componentswidget.h"
#include <QMessageBox>
#include <QSqlQuery>
#include "ui_componentswidget.h"

ComponentsWidget::ComponentsWidget(int buildId, QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::ComponentsWidget)
    , db(SQLiteDBManager::getInstance())
    , componentTypeList({"motherboard", "cpu", "ram", "rom", "rom2", "gpu",
"gpu2", "powerSupply"})
    , buildId(buildId)
    , isConflict(false)
    , isRestored(false)
{
    ui->setupUi(this);
}

ComponentsWidget::~ComponentsWidget()
{
    delete ui;
}

void ComponentsWidget::createWidget()
{
    QSqlQuery query(db->getDB());
    query.exec(QString("SELECT * FROM builds WHERE id = %1").arg(buildId));
    query.next();

    for (int i = 2, deletedCount = 0; i <= 9; i++)
        if (query.value(i).toInt()) {
```

```
            createComponent(i - 2 - deletedCount, true);

            deletedCount++;
        }
}

void ComponentsWidget::createComponent(int index, bool isRestored)
{
    if (printError())
        return;

    this->isRestored = isRestored;

    SingleComponentWidget *singleComponentWidget
        =   new   SingleComponentWidget(componentTypeList[index],   buildId,
isRestored);
    connect(singleComponentWidget,
            &SingleComponentWidget::componentCreated,
            this,
            [this](QString componentType,
                   int componentId,
                   SingleComponentWidget *singleComponentWidget) {
                checkCompatibility(componentType,              componentId,
singleComponentWidget);

                if (componentType == "powerSupply")
                    emit lastComponentCreated();
            });
    connect(singleComponentWidget,
            &SingleComponentWidget::componentChanged,
            this,
            &ComponentsWidget::on_singleComponentChanged);
    connect(singleComponentWidget,
            &SingleComponentWidget::specificationsRequest,
```

```
            this,
            [this](QString componentType, int componentId) {
                emit
clearWidget(SingleComponentWidget::isComponentTypeSecond(componentType),
                                componentId);
            });

    singleComponentWidget->createWidget();

    ui->gridLayout->addWidget(singleComponentWidget);

    componentTypeList.removeFirst();
    if (componentTypeList.empty())
        ui->frame->hide();
}

void ComponentsWidget::on_singleComponentChanged(QString componentType,
                                                 int componentId,
                                                 bool isDeleted)
{
    const QStringList tempComponentList(
        {"motherboard",  "cpu",  "ram",  "rom",  "rom2",  "gpu",  "gpu2",
"powerSupply"});

    for    (int    i    =    0;    i    <    tempComponentList.size()    -
tempComponentList.indexOf(componentType)
                            - (isDeleted ? 0 : 1);
         i++)
        db->runScript(QString("UPDATE builds SET %1 = NULL WHERE id = %2")
                          .arg(*(tempComponentList.crbegin() + i))
                          .arg(buildId));

    emit
clearWidget(SingleComponentWidget::isComponentTypeSecond(componentType),
```

```
componentId);
}

void ComponentsWidget::on_addComponentButton_clicked()
{
    createComponent();
}

void ComponentsWidget::checkCompatibility(QString componentType,
                                          int componentId,
                                          SingleComponentWidget
*singleComponentWidget)
{
    bool result = false;
    int idList[4] = {};

    if (componentType == "motherboard")
        result = true;
    else if (componentType == "cpu")
        result  =  motherboardCompatibilityQuery(componentType,  "socket",
componentId);
    else if (componentType == "ram")
        result  =  motherboardCompatibilityQuery(componentType,  "ramType",
componentId);
    else if (componentType == "rom")
        result        =        motherboardCompatibilityQuery(componentType,
"romInterface", componentId);
    else if (componentType == "rom2") {
        QSqlQuery query(db->getDB());

        query.exec(QString("SELECT   rom2   FROM   builds   WHERE   id   =
%1").arg(buildId));
        query.next();
```

```
        query.exec(QString("SELECT   storage   FROM   rom   WHERE   id   =
%1").arg(query.value(0).toInt()));
        query.next();

        if (!query.value(0).toBool())
            result = true;
        else
            result      =      motherboardCompatibilityQuery(componentType,
"romInterface", componentId);
    } else if (componentType.startsWith("gpu")) {
        QSqlQuery query(db->getDB());

        query.exec(QString("SELECT   cpu   FROM   builds   WHERE   id   =
%1").arg(buildId));
        query.next();

        query.exec(QString("SELECT   graphics   FROM   cpu   WHERE   id   =
%1").arg(query.value(0).toInt()));
        query.next();

        if (!query.value(0).toInt()) {
            if (componentType == "gpu") {
                query.exec(QString("SELECT  storage  FROM  gpu  WHERE  id  =
%1").arg(componentId));
                query.next();

                result = query.value(0).toBool();
            } else if (componentType == "gpu2") {
                query.exec(QString("SELECT  gpu  FROM  builds  WHERE  id  =
%1").arg(buildId));
                query.next();

                query.exec(
                    QString("SELECT   storage   FROM   gpu   WHERE   id   =
```

```
%1").arg(query.value(0).toInt()));
                query.next();
                bool tempGPU2Exists = query.value(0).toBool();

                query.exec(QString("SELECT  storage  FROM  gpu  WHERE  id  =
%1").arg(componentId));
                query.next();

                result = (tempGPU2Exists || query.value(0).toBool());
            }
        } else
            result = true;
    } else if (componentType == "powerSupply") {
        int generalPower = 20;
        QSqlQuery query(db->getDB());

        query.exec(QString("SELECT cpu, ram, gpu, gpu2 FROM builds WHERE id
= %1").arg(buildId));
        query.next();
        for (int i = 0; i < 4; i++)
            idList[i] = query.value(i).toInt();

        query.exec(QString("SELECT    power    FROM    cpu    WHERE    id    =
%1").arg(idList[0]));
        query.next();
        generalPower += query.value(0).toInt();

        query.exec(QString("SELECT    power    FROM    ram    WHERE    id    =
%1").arg(idList[1]));
        query.next();
        generalPower += query.value(0).toInt();

        for (int i = 2; i <= 3; i++) {
            query.exec(QString("SELECT   power   FROM   gpu   WHERE   id   =
```

```
%1").arg(idList[i]));
            query.next();
            generalPower += query.value(0).toInt();
        }

        query.exec(QString("SELECT  power  FROM  powerSupply  WHERE  id  =
%1").arg(componentId));
        query.next();
        result = (query.value(0).toInt() > generalPower);
    }

    singleComponentWidget->setComboBoxColor(result);

    isConflict = !result;

    db->runScript(
        QString("UPDATE  builds  SET  conflict  =  %1  WHERE  id  =
%2").arg(isConflict).arg(buildId));

    if (isConflict) {
        componentConflictType = componentType;

        if (componentConflictType == "powerSupply")
            printError();
    }
}

bool ComponentsWidget::motherboardCompatibilityQuery(QString componentType,
                                                     QString compareField,
                                                     int componentId)
{
    QSqlQuery query(db->getDB());

    query.exec(QString("SELECT  motherboard  FROM  builds  WHERE  id  =
```

```
%1").arg(buildId));
    query.next();
    int motherboardIndex = query.value(0).toInt();

    query.exec(
        QString("SELECT    %1    FROM    motherboard    WHERE    id    =
%2").arg(compareField).arg(motherboardIndex));
    query.next();
    QString compareResult = query.value(0).toString();

    query.exec(QString("SELECT %1 FROM %2 WHERE id = %3")
                    .arg(compareField,
SingleComponentWidget::isComponentTypeSecond(componentType))
                    .arg(componentId));
    query.next();

    if (componentType.startsWith("rom"))
        return compareResult.contains(query.value(0).toString());

    return query.value(0).toString() == compareResult;
}

bool ComponentsWidget::printError()
{
    if (!isConflict)
        return false;

    QMessageBox *a = new QMessageBox(
        QMessageBox::Warning,
        "Конфлікт",
        QString("%1 не є сумісним з даною збіркою ПК за параметром: %2")

.arg(SingleComponentWidget::translateComponentToText(componentConflictType)
,
```

```
                componentConflictType == "cpu"
                    ? "Сокет"
                    : (componentConflictType == "ram"
                           ? "Тип оперативної пам'яті"
                           : (componentConflictType.startsWith("rom")
                                  ? "Інтерфейс/наявність накопичувача"
                                  :
(componentConflictType.startsWith("gpu")
                                          ? "Наявність графічного ядра"
                                          :    (componentConflictType    ==
"powerSupply" ? "Потужність"

: "")))));
    a->show();

    return true;
}
```

Додаток М

Лістинг файлу «pc_constructor.h»

```
#ifndef PC_CONSTRUCTOR_H
#define PC_CONSTRUCTOR_H

#include <QMainWindow>
#include <QTreeWidget>
#include "componentsmanager.h"
#include "createbuilddialog.h"
#include "sqlitedbmanager.h"

QT_BEGIN_NAMESPACE
namespace Ui {
class PC_Constructor;
}
QT_END_NAMESPACE

//  Клас-головне вікно, яке містить у собі всі створені у проєкті
//  віджети та вікна, об'єднує їх у одну систему і з'єднує їх функціонал
class PC_Constructor : public QMainWindow
{
    Q_OBJECT

public:
    PC_Constructor(QWidget *parent = nullptr);
    ~PC_Constructor();

private slots:
    void createBuild(QString buildName);
    void on_action_NewBuild_triggered();
    void on_actionDeleteBuild_triggered();
    void keyPressEvent(QKeyEvent *event);
    void on_tabWidget_tabBarClicked(int index);
```

```
    void on_tabWidget_tabCloseRequested(int index);
    void on_treeWidget_itemDoubleClicked(QTreeWidgetItem *build);
    void on_actionOpenComponentsManager_triggered();
    void on_actionAbout_triggered();

private:
    Ui::PC_Constructor *ui;
    IDBManager *db;

    QString activeBuildName;
    CreateBuildDialog createBuildDialog;
    ComponentsManager componentsManager;

    void setBuildMenuBar();
    void setTreeWidgetBuilds();
    void setTabWidgetBuilds(QString componentType = "", int componentId =
0);
    void setStatusBar();
};

#endif // PC_CONSTRUCTOR_H
```

Додаток Н

Лістинг файлу «pc_constructor.cpp»

```
#include "pc_constructor.h"
#include <QKeyEvent>
#include <QMessageBox>
#include <QSqlQuery>
#include <QTableView>
#include "componentswidget.h"
#include "singlecomponentwidget.h"
#include "specificationswidget.h"
#include "ui_pc_constructor.h"

PC_Constructor::PC_Constructor(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::PC_Constructor)
    , db(SQLiteDBManager::getInstance())
{
    ui->setupUi(this);

    db->runScript("CREATE TABLE builds"
                  "("
                  "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                  "name VARCHAR NOT NULL, "
                  "motherboard INTEGER, "
                  "cpu INTEGER, "
                  "ram INTEGER, "
                  "rom INTEGER, "
                  "rom2 INTEGER, "
                  "gpu INTEGER, "
                  "gpu2 INTEGER, "
                  "powerSupply INTEGER, "
                  "conflict INTEGER DEFAULT 0, "
                  "UNIQUE(name)"
```

```
                ")");

    connect(&createBuildDialog,
            &CreateBuildDialog::getBuildName,
            this,
            &PC_Constructor::createBuild);
    connect(ui->treeWidget,
            &QTreeWidget::itemActivated,
            this,
            &PC_Constructor::on_treeWidget_itemDoubleClicked);

    setTreeWidgetBuilds();
}

PC_Constructor::~PC_Constructor()
{
    delete db;
    delete ui;
}

void PC_Constructor::createBuild(QString buildName)
{
    if (buildName.isEmpty()
        ||      !db->runScript(QString("INSERT      INTO      builds(name)
VALUES('%1')").arg(buildName))) {
        QMessageBox::warning(this,
                             "Помилка",
                             QString("Збірку %1 не створено або збірка вже
існує.").arg(buildName),
                             QMessageBox::Ok);
        return;
    }

    activeBuildName = buildName;
```

```
    QTreeWidgetItem *buildItem = new QTreeWidgetItem;
    buildItem->setText(0, activeBuildName);
    ui->treeWidget->addTopLevelItem(buildItem);

    setBuildMenuBar();
    setTabWidgetBuilds();
    setStatusBar();
}

void PC_Constructor::on_action_NewBuild_triggered()
{
    createBuildDialog.exec();
}

void PC_Constructor::on_actionDeleteBuild_triggered()
{
    if (QMessageBox::warning(this,
                             "Попередження",
                             QString("Ви точно хочете видалити збірку
%1?").arg(activeBuildName),
                             QMessageBox::Ok | QMessageBox::Cancel)
            != QMessageBox::Ok
        || !db->runScript(QString("DELETE FROM builds WHERE name LIKE
'%1'").arg(activeBuildName)))
        return;

    activeBuildName.clear();
    ui->treeWidget->clear();
    on_tabWidget_tabCloseRequested(ui->tabWidget->currentIndex());

    setBuildMenuBar();
    setTreeWidgetBuilds();
    setStatusBar();
```

```
}

void PC_Constructor::keyPressEvent(QKeyEvent *event)
{
    if (event->key() == Qt::Key_Tab && event->modifiers() ==
Qt::ControlModifier) {
        if (ui->tabWidget->currentIndex() + 1 == ui->tabWidget->count())
            ui->tabWidget->setCurrentIndex(0);
        else
            ui->tabWidget->setCurrentIndex(ui->tabWidget->currentIndex() +
1);
    } else if (event->key() == Qt::Key_Backtab) {
        if (!ui->tabWidget->currentIndex())
            ui->tabWidget->setCurrentIndex(ui->tabWidget->count() - 1);
        else
            ui->tabWidget->setCurrentIndex(ui->tabWidget->currentIndex() -
1);
    } else
        return;

    activeBuildName = ui->tabWidget->tabText(ui->tabWidget-
>currentIndex());

    setBuildMenuBar();
    setStatusBar();
}

void PC_Constructor::on_tabWidget_tabBarClicked(int index)
{
    if (index == ui->tabWidget->currentIndex())
        return;

    activeBuildName = ui->tabWidget->tabText(index);
```

```
    setBuildMenuBar();
    setStatusBar();
}

void PC_Constructor::on_tabWidget_tabCloseRequested(int index)
{
    ui->tabWidget->removeTab(index);

    activeBuildName.clear();
    activeBuildName         =         ui->tabWidget->tabText(ui->tabWidget-
>currentIndex());

    setBuildMenuBar();
    setStatusBar();
}

void        PC_Constructor::on_treeWidget_itemDoubleClicked(QTreeWidgetItem
*build)
{
    if (activeBuildName == build->text(0))
        return;

    activeBuildName = build->text(0);

    setBuildMenuBar();
    setStatusBar();

    for (int i = 0; i < ui->tabWidget->count(); i++)
        if (activeBuildName == ui->tabWidget->tabText(i)) {
            ui->tabWidget->setCurrentIndex(i);
            return;
        }

    setTabWidgetBuilds();
```

```
}

void PC_Constructor::on_actionOpenComponentsManager_triggered()
{
    componentsManager.show();
}

void PC_Constructor::on_actionAbout_triggered()
{
    QMessageBox::about(this,
                       "Інформація",
                       "PC Constructor - програма для створення збірок ПК\n"
                       "Олійник Денис, ВСП \"ТФК ТНТУ\" ім. І. Пулюя\n"
                       "2021 рік");
}

void PC_Constructor::setBuildMenuBar()
{
    ui->menuBuilds->setTitle(
        QString("Збірка%1")
            .arg(!activeBuildName.isEmpty()          ?          QString("
[%1]").arg(activeBuildName) : ""));
    ui->actionDeleteBuild->setEnabled(!activeBuildName.isEmpty());
}

void PC_Constructor::setTreeWidgetBuilds()
{
    QSqlQuery query(db->getDB());

    query.exec("SELECT name FROM builds ORDER BY id ASC");
    while (query.next()) {
        QTreeWidgetItem *buildItem = new QTreeWidgetItem;
        buildItem->setText(0, query.value(0).toString());
        if (!buildItem->text(0).isEmpty())
```

```
            ui->treeWidget->addTopLevelItem(buildItem);
    }
}

void    PC_Constructor::setTabWidgetBuilds(QString    componentType,    int
componentId)
{
    QSqlQuery query(db->getDB());

    query.exec(QString("SELECT    id    FROM    builds    WHERE    name    LIKE
'%1'").arg(activeBuildName));
    query.next();

    QWidget *tempTabWidget = new QWidget(this);
    QHBoxLayout *tabHBoxLayout = new QHBoxLayout(tempTabWidget);
    ComponentsWidget            *componentsWidget            =            new
ComponentsWidget(query.value(0).toInt());

    connect(componentsWidget,
            &ComponentsWidget::clearWidget,
            this,
            [this, tempTabWidget](QString componentType, int componentId) {
                ui->tabWidget->removeTab(ui->tabWidget->currentIndex());
                tempTabWidget->deleteLater();

                setTabWidgetBuilds(componentType, componentId);
                setStatusBar();
            });
    connect(componentsWidget,
            &ComponentsWidget::lastComponentCreated,
            this,
            &PC_Constructor::setStatusBar);

    componentsWidget->createWidget();
```

```
    tabHBoxLayout->addWidget(componentsWidget, QSizePolicy::Ignored);
    tabHBoxLayout->addWidget(new        SpecificationsWidget(componentType,
componentId),
                             QSizePolicy::Expanding);
    ui->tabWidget->addTab(tempTabWidget, activeBuildName);
    ui->tabWidget->setCurrentIndex(ui->tabWidget->count() - 1);
}

void PC_Constructor::setStatusBar()
{
    if (activeBuildName.isEmpty()) {
        ui->statusbar->clearMessage();
        return;
    }

    QSqlQuery query(db->getDB());

    int tempPrice = 0;
    int tempPower = 0;

    QStringList tempComponentsList({"cpu", "ram", "gpu", "gpu"});
    QVector<int> ids(4);

    query.exec(
        QString("SELECT cpu, ram, gpu, gpu2 FROM builds WHERE name LIKE
'%1'").arg(activeBuildName));
    query.next();
    for (int i = 0; i < 4; i++)
        ids[i] = query.value(i).toInt();
    for (int i = 0; i < 4; i++) {
        query.exec(
            QString("SELECT     power     FROM     %1     WHERE     id     =
%2").arg(tempComponentsList[i]).arg(ids[i]));
```

```
        if (query.next())
            tempPower += query.value(0).toInt();
    }

    tempComponentsList = {"motherboard", "cpu", "ram", "rom", "rom", "gpu",
"gpu", "powerSupply"};
    ids.clear();
    ids.resize(8);

    query.exec(QString("SELECT conflict FROM builds WHERE name LIKE
'%1'").arg(activeBuildName));
    query.next();
    bool tempConflict = query.value(0).toBool();

    query.exec(QString("SELECT * FROM builds WHERE name LIKE
'%1'").arg(activeBuildName));
    query.next();
    for (int i = 2; i < 10; i++)
        ids[i - 2] = query.value(i).toInt();
    if (!ids.contains(0))
        tempPower += 20;
    else
        tempConflict = true;
    for (int i = 0; i < 8; i++) {
        query.exec(
            QString("SELECT price FROM %1 WHERE id =
%2").arg(tempComponentsList[i]).arg(ids[i]));
        if (query.next())
            tempPrice += query.value(0).toInt();
    }

    ui->statusbar->showMessage(QString("Характеристики збірки %1:"
                                       "        Ціна: %2 грн"
                                       "        Споживання: %3 Вт"
```

```
                                        "        Стан збірки: %4")
                                    .arg(activeBuildName)
                                    .arg(tempPrice)
                                    .arg(tempPower)
                                    .arg(tempConflict ? "Наявні конфлікти/Не
вистачає компонентів"
                                                      :  "Конфліктів  немає.
ПК зібрано"));
}
```

Додаток О

CD-диск із програмним продуктом