ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Бакалавр»

«Web-застосунок підбору комплектуючих комп'ютера»
(тема кваліфікаційної роботи українською мовою)

«Web application for selection of computer components»
(тема кваліфікаційної роботи англійською мовою)

Виконав(ла): здобувач(ка) денної/заочної форми навчання
спеціальності 122 Комп'ютерні науки
(код, назва спеціальності)
Освітня програма Комп'ютерні науки
(назва)
Латиш Артур Володимирович
(прізвище, ім'я, по-батькові здобувача)

Керівник к.т.н., доцент Єпік М.О. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)
Рецензент к.ф.-м.н., доцент Шугайло Ю.Б.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 20___ р.

Завідувач(ка) кафедри
_____ _____
(підпис) (прізвище, ім'я)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/___ ___/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ _____
(підпис) (прізвище, ім'я)

Одеса 2024

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці web-застосунку для конфігурування комп’ютерів з використанням дерева рішення по підбору комплектуючих комп’ютера.

Мета проєкту – створення зручного та інтуїтивного інструменту, який допоможе користувачам підібрати оптимальну конфігурацію комп’ютера відповідно до їхніх вимог, потреб та бюджету.

У ході роботи проведено аналіз предметної області та огляд існуючих рішень для конфігурування комп’ютерів, визначено основні вимоги до веб-застосунку та його функціональні можливості. Розроблено алгоритм дерева рішень для підбору комплектуючих з урахуванням різних критеріїв та обмежень, спроєктовано архітектуру веб-застосунку, структуру бази даних та інтерфейс користувача. Описано вибір технологій для реалізації, імплементацію дерева рішень та інтеграцію бази даних комплектуючих. Проведено тестування застосунку для забезпечення коректної роботи та зручності використання.

ABSTRACT

Bachelor's thesis is devoted to the development of a web application for configuring computers using a decision tree for computer’s components selection.

The project’s goal is to create a convenient and intuitive tool that will help users choose the optimal computer configuration according to their requirements, needs, and budget.

In the process, we analysed the subject area and reviewed existing computer configuration solutions, identified the main requirements for the web application and its functionality. A decision tree algorithm for the selection of components was developed, taking into account various criteria and constraints, and the architecture of the web application, the structure of the database and the user interface were designed. The choice of technologies for implementation, implementation of the decision tree and integration of the component database are described. The application is tested to ensure correct operation and usability.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

БД – база даних

ПЗ – програмне забезпечення

ПК – персональний комп'ютер

ОЗУ – оперативна пам'ять

SSD – твердотільний накопичувач (Solid-State Drive)

HDD – жорсткий диск (Hard Disk Drive)

CPU – центральний процесор (Central Processing Unit)

GPU – графічний процесор (Graphics Processing Unit)

RAID – надлишковий масив недорогих дисків (Redundant Array of Independent Disks)

DIMM – модуль пам'яті (Dual In-line Memory Module)

PSU – блок живлення (Power Supply Unit)

UPS – джерело безперебійного живлення (Uninterruptible Power Supply)

UI – користувацький інтерфейс (User Interface)

UX – досвід користувача (User Experience)

CRUD – створення, читання, оновлення та видалення даних (Create, Read, Update, Delete)

HTML – мова розмітки гіпертексту (HyperText Markup Language)

CSS – каскадні таблиці стилів (Cascading Style Sheets)

JS – JavaScript

PHP – Hypertext Preprocessor

AJAX – асинхронний JavaScript і XML (Asynchronous JavaScript and XML)

SQL – структурована мова запитів (Structured Query Language)

API – прикладний програмний інтерфейс (Application Programming Interface)

ORM – відображення об'єктно-реляційної моделі (Object-Relational Mapping)

CI/CD – безперервна інтеграція/безперервне розгортання (Continuous Integration/Continuous Deployment

ВСТУП

Актуальність теми. В епоху цифрових технологій персональні комп’ютери стали невід’ємною частиною повсякденного життя як для роботи, так і для розваг. З постійним технологічним прогресом зростають і вимоги до продуктивності комп'ютерів, що змушує користувачів регулярно оновлювати або модернізувати свої системи. Вибір правильної конфігурації комп’ютера може бути складним завданням, особливо для тих, хто не має глибоких знань у цій галузі. Тому виникає потреба у зручному і доступному інструменті, який би допомагав користувачам підібрати оптимальну конфігурацію комп’ютера відповідно до їхніх потреб та бюджету.

Веб-застосунок для конфігурування комп’ютерів з використанням дерева рішень є актуальним рішенням для задоволення цієї потреби. Такий застосунок дозволить користувачам легко визначати необхідні комплектуючі для збирання власного комп’ютера або модернізації існуючої системи, надаючи рекомендації на основі дерева рішень. Це забезпечить зручний та інтуїтивно зрозумілий спосіб підбору конфігурації, враховуючи різні критерії, такі як ціна, продуктивність, енергоспоживання та сумісність компонентів [7].

Дерево рішень є потужним інструментом, який використовується в різних галузях, таких як машинне навчання, аналіз даних та системи підтримки прийняття рішень. Воно являє собою ієрархічну структуру, де кожен вузол представляє певну умову або критерій, а гілки відображають можливі результати цієї умови. Шляхом проходження від кореневого вузла до листкових вузлів, можна отримати рекомендацію або рішення на основі заданих вхідних даних.

У контексті конфігурування комп’ютерів дерево рішень може враховувати різноманітні параметри, такі як бюджет користувача, цільове призначення комп'ютера (ігри, офісна робота, мультимедіа тощо), вимоги до продуктивності, енергоспоживання, розміри корпусу та інші критерії. На основі цих вхідних даних система зможе рекомендувати оптимальну

конфігурацію комп'ютера, що містить процесор, оперативну пам'ять, відеокарту, накопичувачі даних та інші компоненти.

Одна з головних переваг використання дерева рішень для конфігурування комп'ютерів полягає в його гнучкості та можливості адаптації до різних вимог користувачів. Дерево можна легко модифікувати, додаючи нові критерії або змінюючи існуючі, що забезпечить актуальність системи з плином часу та появою нових технологій.

Крім того, веб-застосунок на основі дерева рішень буде доступний для використання з будь-якого пристрою з підключенням до Інтернету, що робить його зручним інструментом як для домашніх користувачів, так і для професіоналів та бізнесу. Це особливо актуально в умовах зростаючої популярності віддаленої роботи та онлайн-навчання, коли потреба в потужних комп'ютерах зростає.

Мета і завдання проєкту. Метою даного проєкту є розробка веб-застосунку для конфігурування комп'ютерів з використанням дерева рішень для підбору комплектуючих, що дозволить користувачам легко та ефективно підібрати оптимальну конфігурацію комп'ютера відповідно до їхніх потреб та бюджету.

Основними завданнями проєкту є:

1. Провести аналіз предметної області та визначити вимоги до веб-застосунку.
2. Розробити алгоритм дерева рішень для підбору комплектуючих комп'ютера з урахуванням різних критеріїв та обмежень.
3. Спроєктувати структуру та інтерфейс веб-застосунку з урахуванням принципів зручності використання та досвіду користувача.
4. Реалізувати веб-застосунок з використанням сучасних технологій веб-розробки, забезпечивши його швидкодію, масштабованість та безпеку.
5. Інтегрувати базу даних комплектуючих та забезпечити регулярне оновлення інформації про нові моделі та ціни.

6. Протестувати веб-застосунок з використанням різних методів тестування для забезпечення коректної роботи та зручності використання [12].
7. Розробити інструкцію з використання веб-застосунку для кінцевих користувачів.

Об'єкт дослідження. Об'єктом дослідження є процес конфігурування персональних комп'ютерів з урахуванням різноманітних вимог та обмежень.

Предмет дослідження. Предметом дослідження є веб-застосунок для конфігурування комп'ютерів з використанням дерева рішень для підбору комплектуючих.

Методи дослідження. У процесі розробки веб-застосунку будуть використані наступні методи дослідження

1. Огляд літератури та інтернет-джерел для вивчення предметної області та існуючих рішень.
2. Моделювання процесів та даних з використанням уніфікованої мови моделювання UML.
3. Проєктування інтерфейсу веб-застосунку з урахуванням принципів зручності користування та досвіду користувача.
4. Реалізація веб-застосунку з використанням мов програмування, фреймворків та бібліотек для веб-розробки.
5. Тестування веб-застосунку з використанням різних видів тестування (модульне, інтеграційне, функціональне, зручності тощо).
6. Збір та аналіз зворотного зв'язку від користувачів для подальшого вдосконалення веб-застосунку.

Практична цінність проєкту.

Розроблений веб-застосунок для конфігурування комп'ютерів з використанням дерева рішень для підбору комплектуючих матиме практичну цінність для широкого кола користувачів, які бажають придбати або модернізувати персональний комп'ютер.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ І ОГЛЯД ПІДХОДІВ ДО ВИРІШЕННЯ ЗАВДАННЯ

1.1 Опис предметної області

Предметна область даного проєкту охоплює процес конфігурування персональних комп'ютерів, який включає в себе вибір та комбінування різноманітних апаратних компонентів для створення повноцінної комп'ютерної системи, що відповідає конкретним вимогам та потребам користувача.

Персональний комп’ютер (ПК) є складним пристроєм, який складається з багатьох взаємопов’язаних компонентів. Основними компонентами ПК є:

1. Центральний процесор (CPU): «мозок» комп'ютера, який відповідає за виконання обчислень та інструкцій. Продуктивність процесора визначає швидкість обробки даних та загальну продуктивність системи.
2. Оперативна пам'ять (ОЗУ): тимчасове сховище для даних та інструкцій, які використовуються процесором під час роботи. Більший обсяг ОЗУ дозволяє одночасно виконувати більше завдань та працювати з об'ємними даними.
3. Материнська плата: основна плата, на якій розташовані роз'єми для підключення інших компонентів, таких як процесор, ОЗУ, відеокарта, накопичувачі даних тощо. Вона забезпечує зв'язок між всіма компонентами та визначає можливості розширення системи.
4. Накопичувачі даних: пристрої для зберігання даних та програмного забезпечення. До них належать жорсткі диски (HDD), твердотільні накопичувачі (SSD) та оптичні приводи (DVD, Blue-Ray).
5. Відеокарта (GPU): спеціалізований процесор, призначений для обробки графічних даних та візуалізації зображень. Потужна відеокарта

необхідна для задач, пов'язаних з відеомонтажем, 3D-моделюванням, іграми тощо.

6. Блок живлення (PSU): забезпечує живлення всіх компонентів комп'ютера електричною енергією. Потужність блоку живлення визначає можливості розширення системи та підключення додаткових компонентів.
7. Корпус: захищає всі внутрішні компоненти комп'ютера та забезпечує розміщення та кріплення для них.
8. Система охолодження: містить вентилятори та радіатори для відведення тепла, що виділяється компонентами комп'ютера під час роботи, та запобігання їх перегріву.

Крім основних компонентів, сучасні комп’ютери можуть містити додаткові пристрої, такі як звукові карти, мережеві адаптери, пристрої введення/виведення (клавіатура, миша, принтер тощо), а також різноманітні периферійні пристрої (веб-камери, мікрофони, навушники тощо).

Процес конфігурування комп’ютера полягає у виборі та комбінуванні відповідних компонентів для створення системи, яка відповідає конкретним потребам користувача. Ці потреби можуть містити:

1. Цільове призначення: чи буде комп’ютер використовуватися для офісної роботи, ігор, мультимедіа, професійних додатків (відеомонтаж, 3D-моделювання) або для інших спеціалізованих завдань.
2. Вимоги до продуктивності: деякі завдання, такі як редагування відео або обробка великих обсягів даних, вимагають потужних процесорів, великої кількості оперативної пам'яті та високопродуктивних накопичувачів.
3. Бюджет: ціни на комп’ютерні компоненти можуть значно варіюватися, що впливає на загальну вартість конфігурації.
4. Розміри та форм-фактор: користувачі можуть мати різні вимоги до розмірів комп'ютера, від компактних настільних систем до повнорозмірних корпусів для ігрових або професійних конфігурацій.

5. Енергоспоживання та шумність: для деяких користувачів важливими факторами можуть бути низьке енергоспоживання та рівень шуму, що генерується системою.
6. Сумісність компонентів: під час конфігурування комп’ютера необхідно забезпечити сумісність між компонентами, такими як процесор, материнська плата, оперативна пам'ять, відеокарта тощо.
7. Можливості розширення: деякі користувачі можуть бажати створити конфігурацію, яку можна легко оновлювати або розширювати в майбутньому шляхом додавання нових компонентів.

Правильне конфігурування комп’ютера є важливим завданням, оскільки воно забезпечує оптимальне поєднання продуктивності, функціональності та вартості для задоволення конкретних потреб користувача. Неправильний вибір або комбінація компонентів може призвести до низької продуктивності, несумісності, неефективного використання ресурсів або надмірних витрат.

Традиційно процес конфігурування комп’ютерів вимагав глибоких технічних знань та досвіду у цій галузі. Користувачі повинні були ретельно вивчати специфікації та характеристики різних компонентів, а також розуміти їх сумісність та взаємодію між собою. Це могло бути складним завданням, особливо для нетехнічних користувачів або тих, хто не стежив за швидким розвитком комп’ютерних технологій [4].

З появою веб-технологій та розвитком інтернету з’явилася можливість створення онлайн-інструментів та веб-застосунків, які спрощують процес конфігурування комп'ютерів та роблять його більш доступним для широкого кола користувачів. Такі веб-застосунки можуть використовувати різноманітні алгоритми та методи для аналізу вимог користувача, вибору відповідних компонентів та перевірки їхньої сумісності.

1.2 Аналіз існуючих рішень

На ринку вже існує низка веб-застосунків та онлайн-інструментів, що пропонують користувачам конфігурувати персональні комп'ютери відповідно до їхніх потреб та бюджету. Ці рішення використовують різні підходи та методи для підбору компонентів та надання рекомендацій.

Одним із популярних рішень є PCPartPicker (рис. 1.1). Це веб-застосунок, який дозволяє користувачам вибирати та комбінувати різні компоненти для створення комп'ютерної конфігурації. Основною перевагою PCPartPicker є його велика база даних із сумісними компонентами, що допомагає уникнути проблем несумісності. Крім того, сайт надає інформацію про ціни та наявність компонентів у різних онлайн-магазинах, що полегшує процес придбання. Однак PCPartPicker не використовує дерево рішень або подібні алгоритми для автоматичного підбору конфігурації на основі вимог користувача. Замість цього, користувач самостійно вибирає компоненти з доступного асортименту [34].

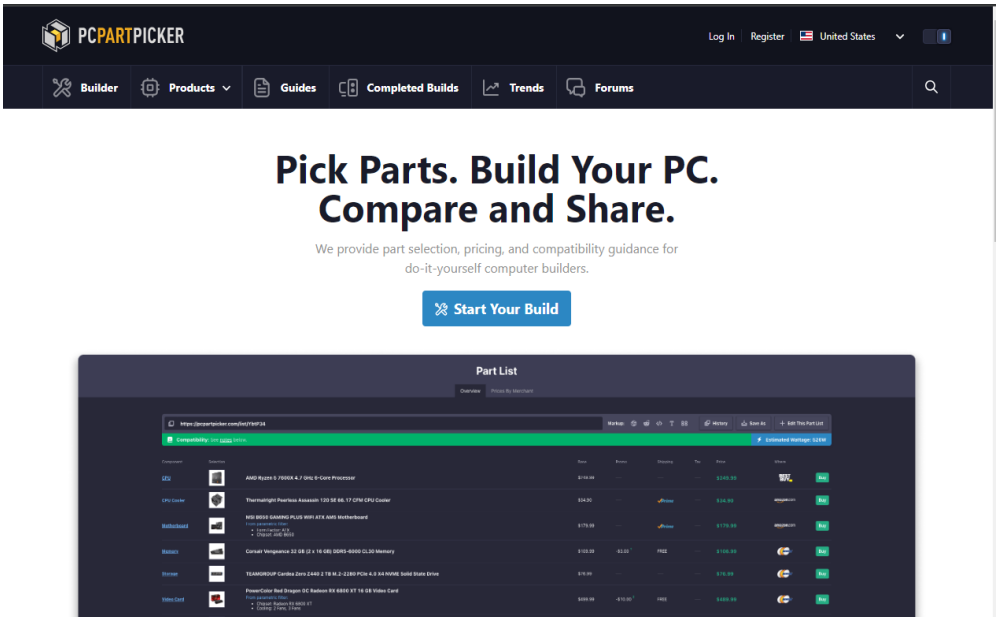

Рисунок 1.1 – Веб-сервіс PCPartPicker
Джерело: https //pcpartpicker.com/ [34].

Іншим веб-застосунком, що заслуговує на увагу, є BLD (рис. 1.2). Він розроблений компанією NZXT, яка спеціалізується на виробництві комп'ютерних корпусів та аксесуарів. BLD пропонує користувачам пройти крізь серію запитань про цільове використання комп'ютера, ігрові вимоги, бюджет та інші преференції. На основі цих відповідей застосунок рекомендує готову конфігурацію, яку можна придбати або налаштувати за бажанням. Перевагою BLD є його інтуїтивний та зручний інтерфейс, що робить процес конфігурування простим і доступним для непрофесійних користувачів. Однак він має обмежений вибір компонентів, оскільки орієнтований переважно на продукцію NZXT та партнерські бренди [30].

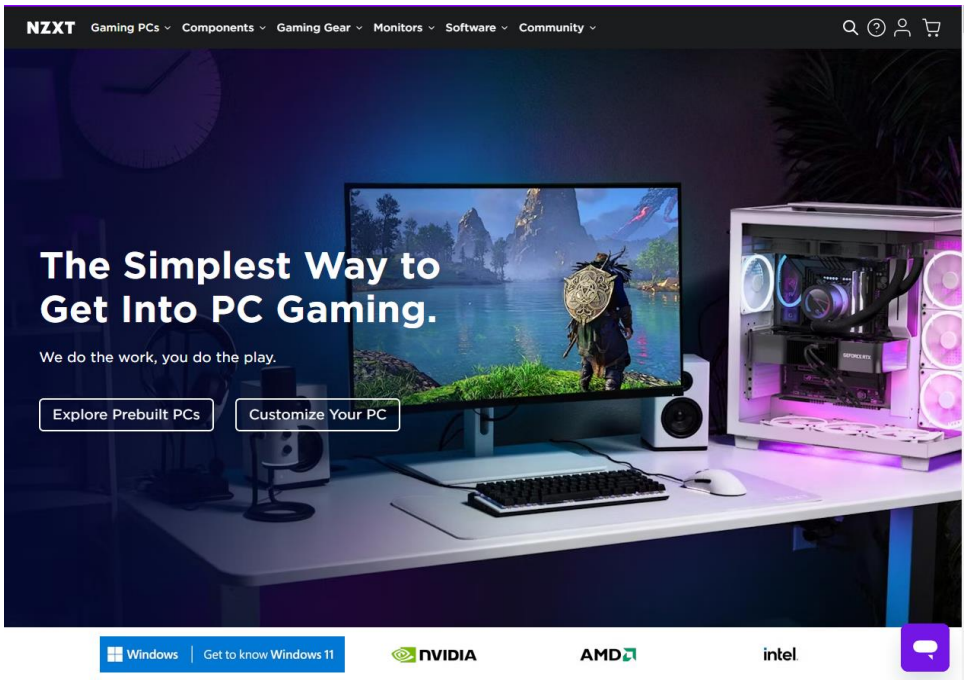

Рисунок 1.2 – Веб-застосунок NZXT
Джерело: https //www.nzxt.com/bldnow [30]

CyberPowerPC – це веб-сайт, що дозволяє користувачам створювати власні конфігурації комп'ютерів або вибирати з попередньо налаштованих варіантів (рис. 1.3). Застосунок використовує дерево рішень для визначення вимог користувача та пропонує відповідні компоненти. Перевагою Cyber PowerPC є широкий вибір компонентів та можливість налаштування конфігурацій відповідно до індивідуальних потреб. Однак інтерфейс може

бути складним для непрофесійних користувачів, а процес конфігурації може виявитися занадто деталізованим [31].

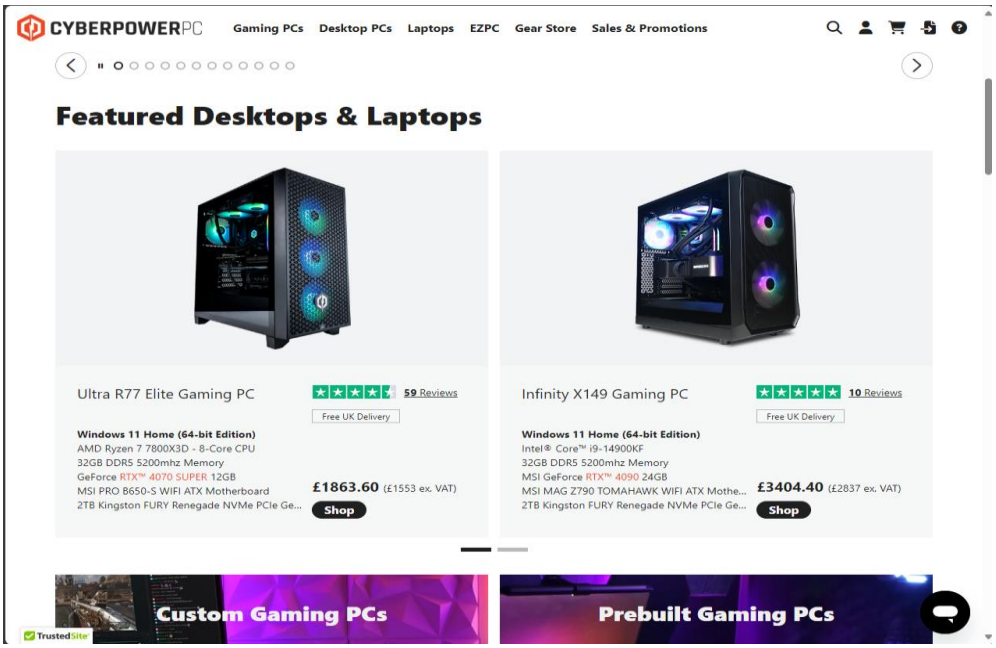

Рисунок 1.3 – Веб-сайт CyberPowerPC
Джерело: https //www.cyberpowerpc.com/ [31]

Puget Systems – це веб-сайт, орієнтований на професійних користувачів, таких як дизайнери, інженери та творці контенту. Застосунок пропонує конфігурації, оптимізовані для конкретних робочих навантажень та програмного забезпечення. Puget Systems використовує власні алгоритми та тести продуктивності для визначення найбільш ефективних компонентів для певних задач [32]. Однак це рішення може бути занадто спеціалізованим для звичайних домашніх користувачів, а також має обмежений вибір доступних конфігурацій.

iBUYPOWER – це ще один веб-застосунок для конфігурування та придбання комп'ютерів (рис. 1.4). Він пропонує користувачам пройти крізь серію запитань для визначення їхніх потреб, після чого рекомендує відповідні готові конфігурації або можливість створити власну. iBUYPOWER має широкий вибір компонентів та варіантів конфігурацій, проте його інтерфейс може бути складним для непрофесійних користувачів [33].

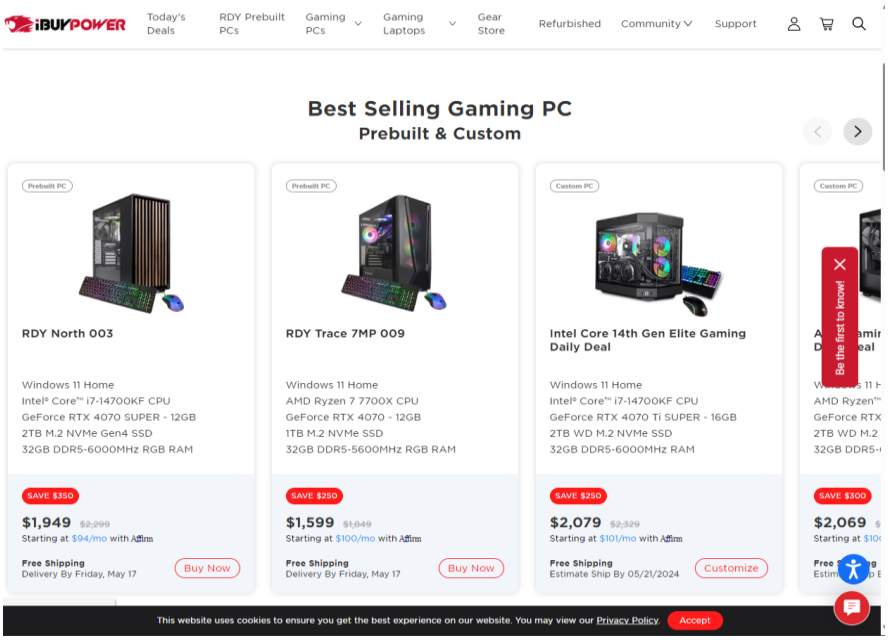

Рисунок 1.4 – Веб-сайт iBUYPOWER
Джерело: https //www.ibuypower.com/ [33]

Серед відкритих рішень, заснованих на спільноті, можна виділити сайт LogicalIncrements (рис. 1.5). Це веб-застосунок, який надає рекомендації щодо конфігурацій комп'ютерів, розподілених за різними рівнями продуктивності та бюджетами. Рекомендації базуються на внесках та досвіді спільноти користувачів, а не на автоматизованих алгоритмах. Перевагою цього підходу є постійне оновлення та адаптація до нових технологій, однак відсутність чіткої системи може призвести до неоднозначних або суперечливих рекомендацій [35].

Важливо зазначити, що більшість із цих існуючих рішень пропонують користувачам придбати готові конфігурації або замовити збирання комп'ютера у спеціалізованих компаніях. Це може бути зручним варіантом для тих, хто не бажає займатися самостійним придбанням та збиранням компонентів. Однак такий підхід може обмежувати гнучкість та можливості налаштування конфігурації відповідно до індивідуальних потреб та бюджету.

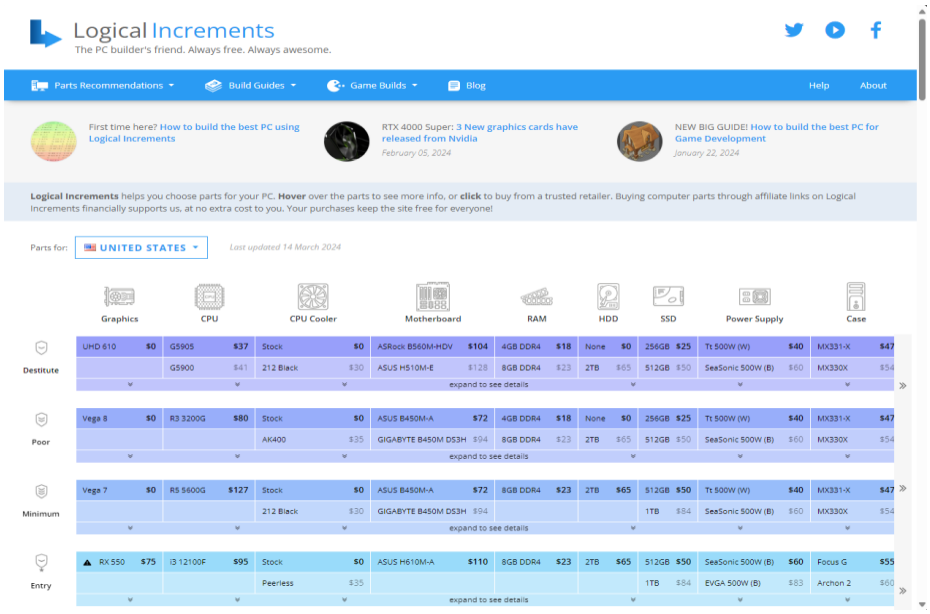

Рисунок 1.5 – Веб-сайт LogicalIncrements
Джерело: https //logicalincrements.com/ [35]

Отже, хоча на ринку існують різноманітні веб-застосунки для конфігурування комп'ютерів, кожен із них має свої переваги та недоліки. Деякі рішення пропонують зручні інтерфейси та інтуїтивні процеси конфігурації, але обмежені вибір компонентів або відсутність автоматизованих алгоритмів для підбору конфігурацій. Інші застосунки надають більше можливостей для налаштування та вибору компонентів, але можуть бути складними для непрофесійних користувачів [6].

Розробка нового веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень для підбору комплектуючих може стати актуальним рішенням, що поєднає переваги існуючих застосунків та усуне їхні недоліки. Дерево рішень є потужним інструментом, який може забезпечити автоматизований та персоналізований підбір конфігурацій на основі заданих критеріїв та вимог користувача.

1.3 Обґрунтування актуальності розробки

Розробка веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень для підбору комплектуючих є актуальним та

своєчасним рішенням, що відповідає сучасним тенденціям у галузі інформаційних технологій та задовольняє зростаючі потреби користувачів.

Актуальність даного проєкту обумовлена стрімким розвитком технологій та постійним оновленням апаратного забезпечення. Комп’ютерні компоненти, такі як процесори, відеокарти, накопичувачі даних та інші, швидко застарівають, і користувачі змушені регулярно модернізувати свої системи, щоб відповідати вимогам сучасного програмного забезпечення та додатків. Вибір правильної конфігурації комп'ютера стає все більш складним завданням, оскільки з'являються нові технології, стандарти та вимоги до продуктивності.

Крім того, зростаюча популярність віддаленої роботи, дистанційного навчання та електронної комерції призводить до збільшення попиту на потужні та ефективні комп'ютерні системи. Користувачі прагнуть до оптимального поєднання продуктивності, енергоефективності, компактності та вартості. Веб-застосунок з деревом рішень для підбору комплектуючих допоможе задовольнити ці потреби, надаючи персоналізовані рекомендації на основі заданих критеріїв та обмежень.

Ще одним важливим фактором, що обґрунтовує актуальність даного проєкту, є зростаюча популярність ігрової індустрії та мультимедійних додатків. Сучасні комп'ютерні ігри та програми для редагування відео, 3D-моделювання та анімації вимагають потужного апаратного забезпечення, здатного забезпечити високу продуктивність та якість візуалізації. Використання дерева рішень для підбору комплектуючих дозволить користувачам, які працюють у цих сферах, створити оптимальну конфігурацію, що відповідає їхнім вимогам до продуктивності та якості.

Не менш важливим є те, що веб-застосунок з деревом рішень для конфігурування комп'ютерів може стати корисним інструментом для малого та середнього бізнесу. Підприємства часто потребують надійних та ефективних комп'ютерних систем для виконання різноманітних завдань, таких як офісна робота, обробка даних, розробка програмного забезпечення та

багато іншого. Використання веб-застосунку дозволить їм швидко та легко підібрати оптимальні конфігурації комп'ютерів відповідно до їхніх бізнес-вимог та бюджету, що сприятиме підвищенню продуктивності та ефективності роботи.

Окрім практичних переваг, розробка веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень також матиме освітнє та навчальне значення. Такий інструмент може бути корисним для студентів та ентузіастів, які цікавляться комп’ютерними технологіями та бажають розширити свої знання в цій галузі. Веб-застосунок надасть їм можливість експериментувати з різними конфігураціями, вивчати взаємодію компонентів та отримувати рекомендації на основі заданих критеріїв.

Важливо також відзначити, що веб-застосунок, заснований на дереві рішень, матиме переваги з точки зору масштабованості та гнучкості. Дерево рішень можна легко оновлювати та модифікувати, додаючи нові критерії або змінюючи існуючі, що забезпечить актуальність системи з плином часу та появою нових технологій. Крім того, веб-інтерфейс забезпечить доступність застосунку з будь-якого пристрою, підключеного до Інтернету, що зробить його зручним інструментом як для домашніх користувачів, так і для професіоналів та бізнесу.

З точки зору економічної доцільності, розробка веб-застосунку для конфігурування комп'ютерів з використанням дерева рішень може забезпечити ряд переваг. По-перше, це дозволить користувачам оптимізувати витрати на придбання комп'ютерних компонентів, уникаючи непотрібних або надлишкових витрат. Веб-застосунок враховуватиме бюджетні обмеження користувача та пропонуватиме конфігурації, що відповідають його фінансовим можливостям.

Крім того, використання веб-застосунку може сприяти економії часу та зусиль, які зазвичай витрачаються на ретельне вивчення специфікацій та характеристик різних компонентів, а також на перевірку їх сумісності. Замість

цього, користувачі отримають персоналізовані рекомендації, засновані на автоматизованих алгоритмах та дереві рішень [3].

Нарешті, веб-застосунок може забезпечити економію ресурсів та енергії, пропонуючи енергоефективні конфігурації, що відповідають потребам користувачів. Це може бути особливо важливим для підприємств, які прагнуть скоротити витрати на електроенергію та зменшити свій вуглецевий слід.

Підсумовуючи, розробка веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень для підбору комплектуючих є актуальним та своєчасним рішенням, що відповідає сучасним тенденціям у галузі інформаційних технологій та задовольняє зростаючі потреби користувачів. Такий застосунок забезпечить зручний та персоналізований спосіб підбору оптимальної конфігурації комп'ютера, враховуючи різні критерії, такі як продуктивність, енергоефективність, бюджет та сумісність компонентів.

1.4 Постановка завдання

Метою даного проєкту є розробка веб-застосунку для конфігурування комп’ютерів, який дозволить користувачам легко та ефективно підбирати оптимальну конфігурацію персонального комп’ютера відповідно до їхніх потреб, вимог та бюджету. Для досягнення цієї мети необхідно вирішити низку ключових завдань. Проаналізувати та описати предметну область, а саме процес конфігурування комп’ютерів та взаємозв’язки між різними апаратними компонентами.

Важливим завданням є визначення основних критеріїв, які будуть враховуватися під час підбору конфігурації комп’ютера. Ці критерії можуть містити:

1. Цільове призначення комп’ютера (наприклад, офісна робота, ігри, мультимедіа, професійні завдання тощо).

2. Вимоги до продуктивності, такі як швидкість обробки даних, графічна потужність, багатозадачність тощо.
3. Бюджетні обмеження користувача.
4. Енергоефективність та рівень шумності.
5. Розміри та форм-фактор комп'ютера.
6. Можливості розширення та модернізації в майбутньому.

Розробити структуру дерева рішень, визначити послідовність запитань та критеріїв, а також розробити логіку прийняття рішень на основі відповідей користувача.

Спроєктувати загальну архітектури веб-застосунку та структуру бази даних. Це містить визначення основних модулів та компонентів веб-застосунку, їх взаємодії та потоків даних, а також розробку ефективної та масштабованої структури бази даних для зберігання інформації про доступні компоненти, їхні характеристики та ціни. Реалізувати веб-застосунок з використанням відповідних технологій веб-розробки: мови програмування, фреймворки та бібліотеки. Розробити серверну та клієнтську частини веб-застосунку. Розробити базу даних та наповнити її інформацією про доступні комп’ютерні компоненти, їхні характеристики та ціни. Це може бути зроблено шляхом інтеграції з API онлайн-магазинів комп'ютерних компонентів або ручного введення даних.

Провести ретельне тестування для виявлення та усунення будь-яких помилок або недоліків: модульне тестування окремих компонентів, інтеграційне тестування взаємодії між компонентами, функціональне тестування для перевірки коректності роботи веб-застосунку відповідно до вимог, а також тестування зручності та досвіду користувача.

Розробити детальну інструкцію з використання веб-застосунку для кінцевих користувачів.

Успішне виконання всіх цих завдань дозволить створити ефективний та зручний веб-застосунок для конфігурування комп’ютерів, який забезпечить користувачам персоналізований підхід до вибору оптимальної конфігурації

відповідно до їхніх потреб та бюджету. Використання дерева рішень для підбору комплектуючих забезпечить точність та релевантність рекомендацій, а зручний інтерфейс та регулярне оновлення бази даних зроблять веб-застосунок привабливим та корисним інструментом для широкого кола користувачів.

Висновки до розділу 1

У розділі розглянуто предметну область конфігурування персональних комп’ютерів, проаналізовано існуючі рішення на ринку веб-застосунків та обґрунтовано актуальність і необхідність розробки нового веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень для підбору комплектуючих.

Обґрунтовано актуальність розробки веб-застосунку з використанням дерева рішень для підбору комплектуючих. Проєкт є актуальним та перспективним рішенням, яке поєднує переваги існуючих застосунків та усуває їхні недоліки. Дерево рішень забезпечить автоматизований та персоналізований підбір конфігурацій на основі заданих критеріїв та вимог користувача, а веб-інтерфейс забезпечить зручність використання та доступність з будь-якого пристрою. Такий веб-застосунок матиме широкий спектр застосування, від домашніх користувачів та ентузіастів до професіоналів, малого та середнього бізнесу, а також освітньої галузі.

Визначено основну мету кваліфікаційної роботи та описано усі завдання, які потрібно виконати для досягнення мети.

Загалом, розробка веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень для підбору комплектуючих є актуальним та необхідним проєктом, який відповідає сучасним тенденціям у галузі інформаційних технологій та задовольняє зростаючі потреби користувачів.

2 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ

2.1 Вимоги до веб-застосунку

Веб-застосунок для конфігурування комп’ютерів з використанням дерева рішень є комплексною системою, яка має відповідати низці функціональних та нефункціональних вимог для забезпечення ефективної роботи, зручності використання та масштабованості. Детальний опис цих вимог є ключовим для успішної розробки та впровадження веб-застосунку.

Функціональні вимоги:

1. Зручний та інтуїтивний інтерфейс для послідовного вибору компонентів комп'ютера (CPU, материнська плата, RAM, Storage, GPU, PSU, Case, система охолодження). Інтерфейс має бути логічно структурованим, з чітким розподілом компонентів за категоріями та можливістю фільтрації доступних варіантів за різними критеріями.
2. Використання дерева рішень для визначення сумісності компонентів та формування рекомендацій на основі заданих критеріїв (бюджет, цільове призначення, продуктивність, енергоспоживання тощо). Дерево рішень має бути гнучким та легко модифікованим для додавання нових критеріїв або зміни існуючих.
3. Перевірка сумісності вибраних компонентів між собою та повідомлення користувача про виявлені несумісності з пропозицією альтернатив. Система повинна автоматично визначати потенційні конфлікти сумісності та попереджати користувача перед остаточним підтвердженням конфігурації.
4. Наявність бази даних з детальною інформацією про комплектуючі, їх характеристики, сумісність та ціни. База даних має бути оптимізованою для швидкого пошуку та фільтрації записів.
5. Зручний інтерфейс для адміністрування та оновлення бази даних. Адміністратори повинні мати можливість легко додавати, змінювати та

видаляти записи про компоненти, завантажувати їх характеристики та зображення.

6. Можливість розрахунку вартості зібраної конфігурації з відображенням загальної ціни та вартості окремих компонентів.
7. Інтеграція з платіжними системами для оформлення замовлення (опціонально). Користувачі можуть мати можливість придбати зібрану конфігурацію безпосередньо через веб-застосунок з використанням безпечних платіжних шлюзів.
8. Реєстрація та авторизація користувачів для збереження історії переглядів та конфігурацій (опціонально). Це дозволить користувачам повертатися до попередніх варіантів та легко модифікувати їх.

Нефункціональні вимоги:

1. Зручний та інтуїтивно зрозумілий інтерфейс користувача (UI/UX) з адаптивним дизайном для різних пристроїв. Інтерфейс повинен бути привабливим, але без надмірностей, що можуть відволікати від основного функціоналу. Навігація має бути простою та логічною.
2. Швидка робота та своєчасне оновлення інформації навіть за умови високого навантаження. Веб-застосунок має використовувати ефективні технології та оптимізовані методи обробки даних для забезпечення належної продуктивності.
3. Масштабованість системи для розширення функціональності та додавання нових компонентів. Архітектура веб-застосунку повинна бути модульною, що дозволить легко додавати або змінювати окремі компоненти без зупинки роботи всієї системи.
4. Належний рівень безпеки для захисту даних користувачів та запобігання несанкціонованому доступу. Необхідно застосовувати сучасні методи шифрування, захисту від вебзагроз (XSS, CSRF, SQL-ін'єкції тощо) та регулярно оновлювати систему для усунення виявлених вразливостей.
5. Доступність веб-застосунку з будь-якого пристрою з підключенням до Інтернету та сучасним веб-браузером. Веб-застосунок має бути

розгорнутим на захищеному веб-сервері з належною підтримкою різних протоколів та стандартів.

6. Підтримка різних браузерів (Chrome, Firefox, Safari, Edge тощо) та операційних систем (Windows, macOS, Linux). Веб-застосунок повинен бути розроблений з використанням кросбраузерних технологій та методів для забезпечення коректного відображення та функціонування на різних платформах.
7. Стійкість до збоїв та помилок, механізми резервного копіювання та відновлення даних. Необхідно передбачити обробку помилок та винятків, а також розробити процедури резервного копіювання та відновлення бази даних для запобігання втраті даних у разі непередбачуваних ситуацій.
8. Сумісність та можливість інтеграції з існуючими системами та сервісами (наявність API). Веб-застосунок має підтримувати відкриті стандарти та протоколи для інтеграції з іншими додатками та сервісами, наприклад, через RESTful API.
9. Можливість регулярного оновлення та вдосконалення функціоналу. Процес оновлення веб-застосунку повинен бути зручним та мінімізувати простої системи. Необхідно передбачити можливість поетапного розгортання оновлень та забезпечити сумісність з попередніми версіями.
10. Належна документація та навчальні матеріали. Розробники та адміністратори веб-застосунку повинні мати доступ до детальної технічної документації, керівництв користувача та навчальних матеріалів для забезпечення належної експлуатації та підтримки системи.
11. Забезпечення високого рівня юзабіліті (usability) та доступності (accessibility) для різних груп користувачів, включаючи людей з обмеженими можливостями.

12. Впровадження ефективних механізмів SEO-оптимізації для покращення видимості веб-застосунку в пошукових системах та залучення більшої кількості користувачів.
13. Використання сучасних технологій та фреймворків для розробки веб-застосунків, що забезпечить легкість підтримки, розширення та оновлення системи в майбутньому.

Цей розширений перелік функціональних та нефункціональних вимог забезпечить створення ефективного, зручного та масштабованого веб-застосунку для конфігурування комп'ютерів, який задовольнить потреби різних груп користувачів та буде відповідати сучасним стандарт

На рисунку 2.1 представлено діаграму прецедентів веб-застосунку.

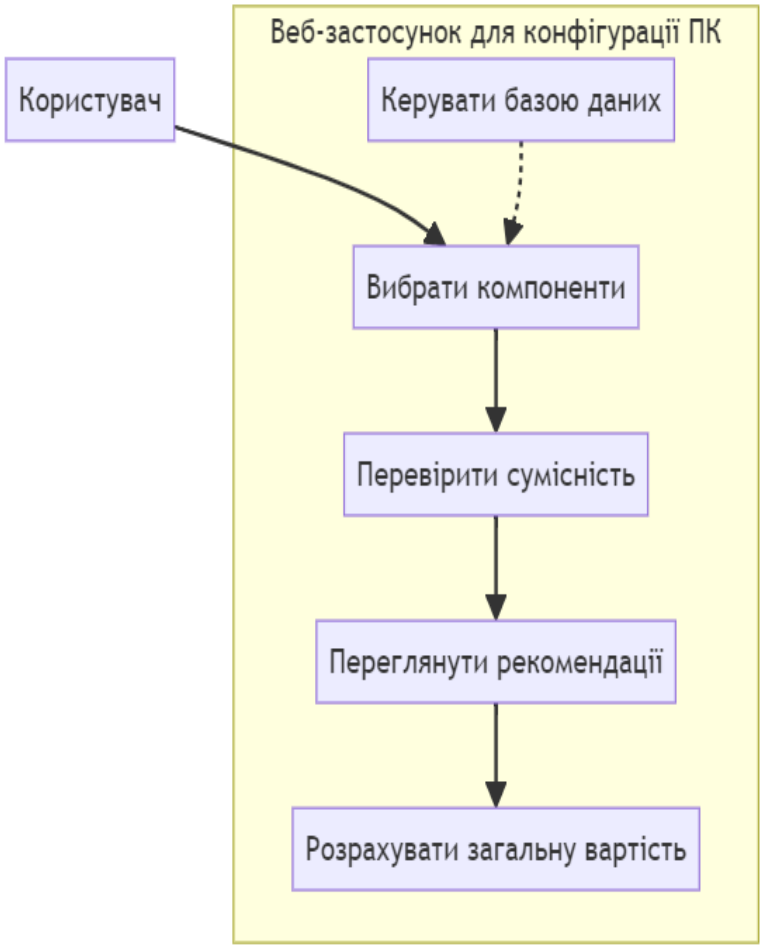

Рисунок 2.1 – Діаграма прецедентів (Use Case Diagram)
Джерело: розроблено автором

2.2 Використання дерева рішень для підбору комплектуючих

Дерево рішень є потужним і наочним інструментом, який ідеально підходить для задачі підбору оптимальної конфігурації комп’ютера на основі заданих критеріїв та обмежень. Його структура має ієрархічну будову, де кожен вузол представляє певний критерій або умову, а гілки відображають можливі результати цієї умови. У проєкті дерево рішень допомагає користувачам вибрати оптимальні комплектуючі для комп’ютера на основі бюджету та призначення.

На рисунку 2.2 представлено дерево рішень для підбору комплектуючих комп’ютера.

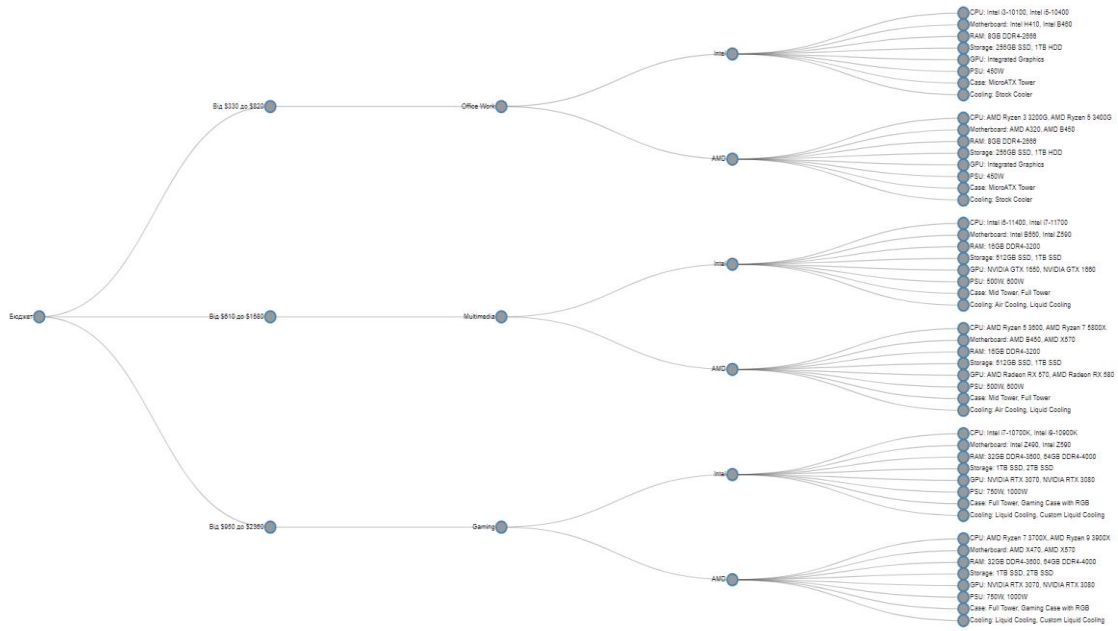

Рисунок 2.2 – Дерево рішень для підбору комплектуючих

Джерело: розроблено автором

Дерево рішень для підбору комплектуючих комп’ютера починається з визначення бюджету користувача. Спочатку користувач має вибрати бюджет, який він готовий витратити на придбання комп’ютера. Бюджет поділяється на три категорії: від $330 до $820, від $610 до $1580, та від $950 до $2360. Вибір

бюджету визначає подальший напрямок дерева рішень, оскільки різні бюджети вимагають різних підходів до підбору комплектуючих.

Наступний крок полягає у виборі призначення комп’ютера. В залежності від обраного бюджету, користувач визначає, для чого саме він буде використовувати комп’ютер: для офісної роботи, мультимедіа чи ігор. Вибір призначення допомагає визначити необхідний рівень продуктивності та відповідні компоненти, які найкраще підійдуть для конкретних завдань. Так, наприклад, комп’ютер для офісної роботи може вимагати менш потужних комплектуючих порівняно з комп’ютером для ігор або мультимедійних завдань. Після визначення бюджету і призначення комп’ютера, дерево рішень працює за наступним алгоритмом:

1. Крок 1: Вибір процесора (CPU)
    - На цьому кроці користувачеві буде запропоновано вибрати центральний процесор (CPU) для своєї майбутньої конфігурації.
    - Доступні різні моделі процесорів від провідних виробників, таких як Intel та AMD, з різними характеристиками продуктивності, кількістю ядер, тактовою частотою та технологіями.
    - Вибір процесора є критично важливим, оскільки він відіграє ключову роль у загальній продуктивності системи та визначає сумісність з іншими компонентами.
2. Крок 2: Вибір материнської плати
    - Після вибору процесора користувачеві потрібно вибрати сумісну материнську плату.
    - Материнська плата визначає підтримувані типи оперативної пам'яті, роз'єми для накопичувачів, слоти розширення для відеокарт та інших компонентів.
3. Крок 3: Вибір оперативної пам'яті (RAM)
    - На цьому кроці користувач вибере обсяг та тип оперативної пам'яті для своєї системи.

- Буде запропоновано різні комплекти пам’яті DDR4 (залежно від підтримки материнської плати) від провідних виробників.
- Обсяг пам'яті впливає на продуктивність системи в багатозадачному режимі та під час роботи з ресурсомісткими додатками.

4. Крок 4: Вибір накопичувача (Storage)
   - Користувач вибирає накопичувач для зберігання даних та встановлення операційної системи.
   - Буде запропоновано різні типи накопичувачів, такі як твердотільні накопичувачі (SSD) та жорсткі диски (HDD), з різними обсягами пам’яті та швидкостями передачі даних.
5. Крок 5: Вибір відеокарти (GPU)
   - На цьому кроці користувач вибирає відеокарту для своєї системи, якщо планує використовувати її для ігор, обробки відео, 3D-моделювання або інших графічно-інтенсивних завдань.
   - Буде запропоновано різні моделі відеокарт від NVIDIA та AMD з різними характеристиками продуктивності, обсягом відеопам'яті та підтримкою технологій, таких як трасування променів.
6. Крок 6: Вибір блока живлення (PSU)
   - Блок живлення є важливим компонентом для забезпечення стабільної роботи всієї системи.
   - Інформація про кожний блок живлення міститиме його потужність (у ватах).
   - Буде надано рекомендації щодо вибору блока живлення відповідної потужності на основі конфігурації системи.
7. Крок 7: Вибір корпусу (Case)
   - Корпус є захисним корпусом для всіх компонентів комп'ютера та впливає на його загальний зовнішній вигляд.
   - Користувачеві буде запропоновано основні моделі корпусів від виробників.

8. Крок 8: Вибір системи охолодження
    - Система охолодження відповідає за відведення тепла від процесора, відеокарти та інших компонентів для забезпечення їх стабільної роботи.
    - Користувачеві буде запропоновано різні варіанти систем охолодження, такі як повітряні кулери для процесора, рідинні системи охолодження (AIO) або спеціалізовані вентиляторні системи для корпусу.
    - Буде надано рекомендації ціни щодо вибору системи охолодження на основі конфігурації системи та очікуваних навантажень.

Після завершення всіх кроків, веб-застосунок складе остаточну конфігурацію комп'ютера на основі вибраних користувачем компонентів. Ця конфігурація буде відображена у "Final Configuration

Однак, слід враховувати певні обмеження та недоліки використання дерев рішень:

- Чутливість до незбалансованих або зашумлених даних: якість дерева рішень значною мірою залежить від якості та збалансованості навчальних даних. Незбалансовані або зашумлені дані можуть призвести до перенавчання або неточних рішень.
- Складність для задач з великою кількістю змінних: для задач з дуже великою кількістю критеріїв або змінних дерева рішень можуть стати надто громіздкими та складними для інтерпретації.
- Нестабільність: невеликі зміни у навчальних даних можуть призвести до значних змін у структурі дерева, що робить його нестабільним.

Для подолання цих обмежень можуть застосовуватись додаткові методи:

- Випадкові ліси: ансамблевий метод, який будує множину дерев рішень на різних підмножинах даних та об'єднує їх результати для підвищення точності та стабільності.

- Бустинг: ітеративний підхід, який послідовно будує дерева рішень, фокусуючись на важких прикладах, які були неправильно класифіковані на попередніх етапах.
- Обрізка дерева: процес видалення або об'єднання окремих вузлів чи гілок для зменшення складності та підвищення узагальнюваності дерева.
- Комбінування з іншими алгоритмами: створення ансамблевих моделей, які поєднують дерева рішень з іншими алгоритмами машинного навчання, такими як нейронні мережі чи метод опорних векторів.

Незважаючи на ці обмеження, для більшості типових задач конфігурування комп'ютерів дерево рішень залишається ефективним та зрозумілим рішенням. Воно забезпечує наочність, гнучкість та можливість врахування різноманітних критеріїв, що є критично важливим для якісного підбору комплектуючих.

Загалом, використання дерева рішень для підбору комплектуючих є перспективним підходом, який дозволяє створити потужний і зрозумілий інструмент для конфігурування персональних комп'ютерів. Подальше вдосконалення алгоритмів та інтеграція з іншими технологіями, такими як ансамблеві моделі чи нейронні мережі, може значно підвищити ефективність та точність рекомендацій веб-застосунку.

Діаграма діяльності ілюструє взаємодію користувача з системою, дозволяє уявити послідовність дій, які користувач виконує, та визначає можливі альтернативні шляхи взаємодії. Використовується для швидкого огляду процесу, ідентифікації варіантів поведінки користувача та виявлення областей для оптимізації та поліпшення ефективності.

Рисунок 2.3 надає візуальне представлення можливих шляхів взаємодії користувача з веб-застосунком для конфігурування комп’ютерів.

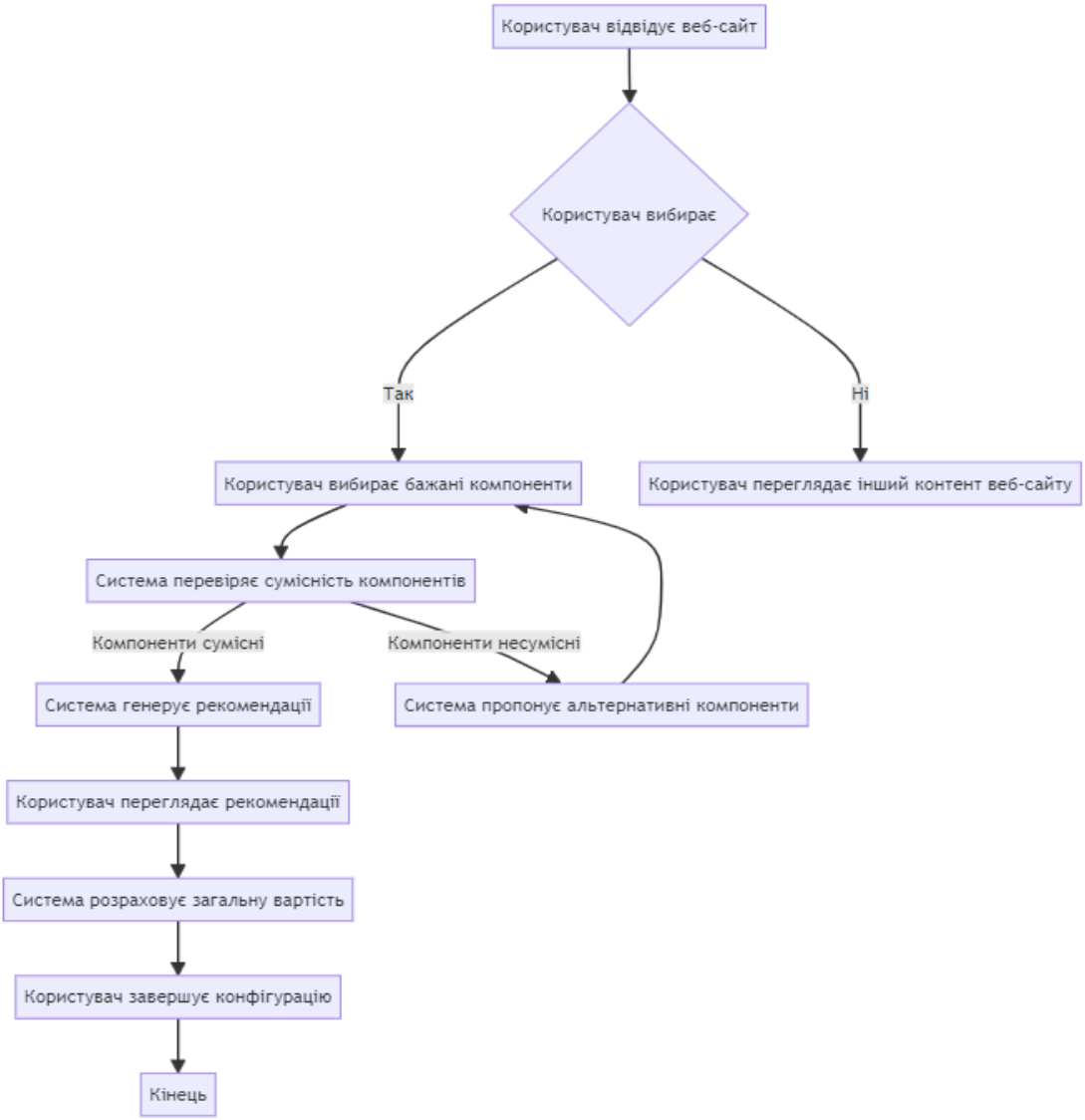

Рисунок 2.3 – Діаграма діяльності (Activity Diagram)
Джерело: розроблено автором

2.3 Проєктування інтерфейсу користувача

Веб-застосунок складається з наступних основних частин: головної сторінки, сторінки конфігурації, сторінки перегляду зібраної конфігурації та сторінки оформлення замовлення.

На головній сторінці розміщено привітальний заголовок, коротку інструкцію та велику кнопку «Розпочати конфігурацію». Ця сторінка служить входом до веб-застосунку для користувачів, які бажають розпочати процес підбору комплектуючих (рис. 2.4).

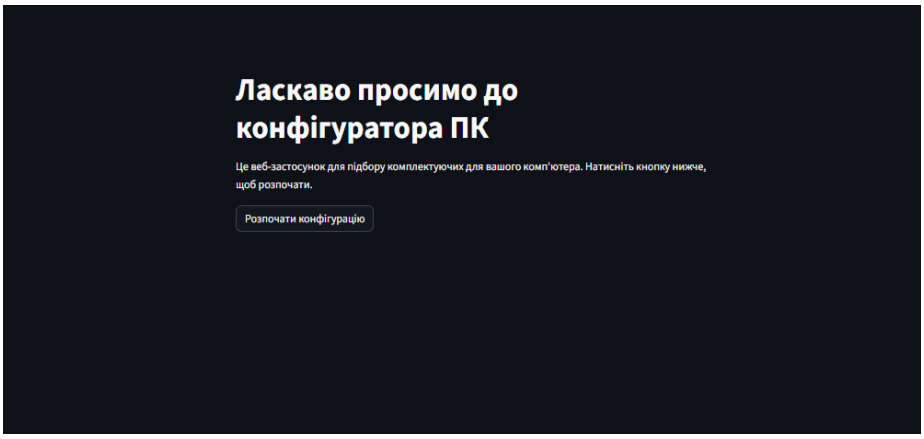

Рисунок 2.4 – Головна сторінка
Джерело: розроблено автором

Після натискання кнопки «Розпочати конфігурацію» користувач переходить на сторінку конфігурації. Ця сторінка є серцем веб-застосунку і містить інтерактивне дерево рішень, яке керує процесом підбору комплектуючих. На початку користувачеві пропонується вказати свій бюджет та призначення майбутнього комп’ютера (офісна робота, ігри, мультимедіа тощо). Залежно від вибраних параметрів, дерево рішень динамічно відображає наступні кроки та питання, допомагаючи користувачеві поступово звужувати коло варіантів до остаточної конфігурації (рис. 2.5- 2.7).

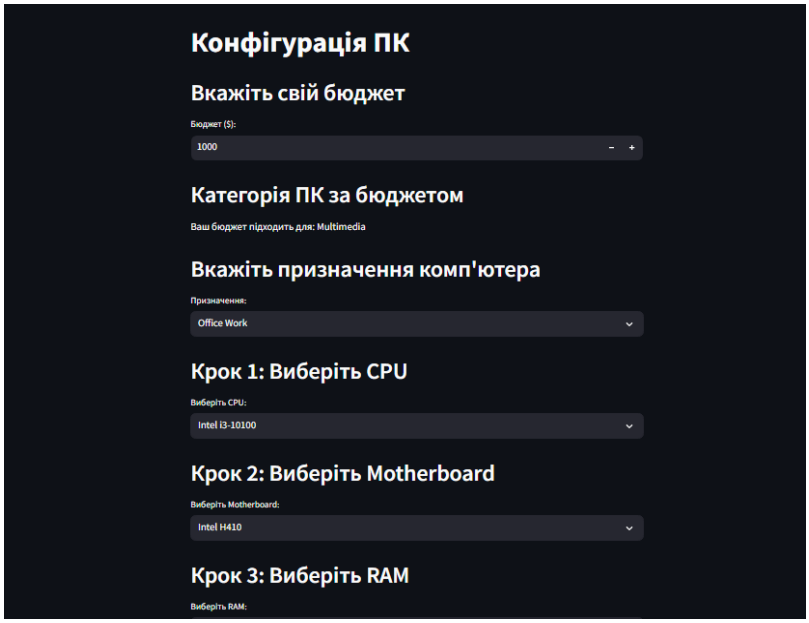

Рисунок 2.5 – Робота програми сторінка 1
Джерело: розроблено автором

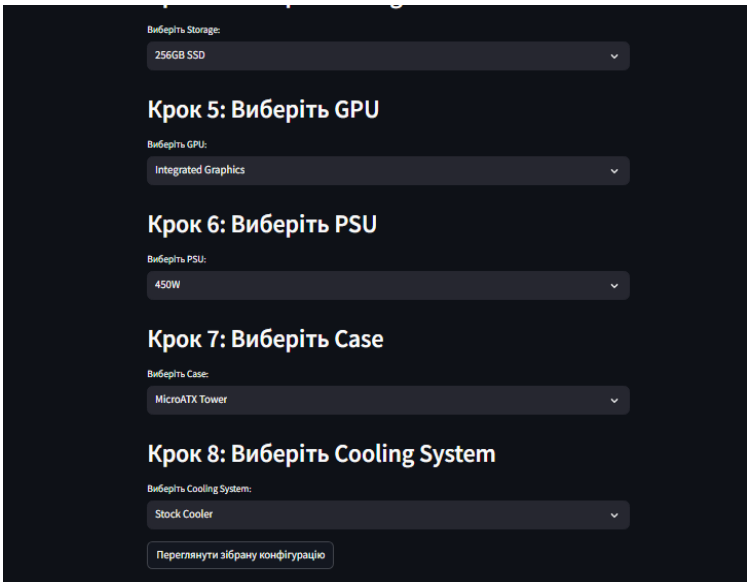

Рисунок 2.6 – Робота програми сторінка 2
Джерело: розроблено автором

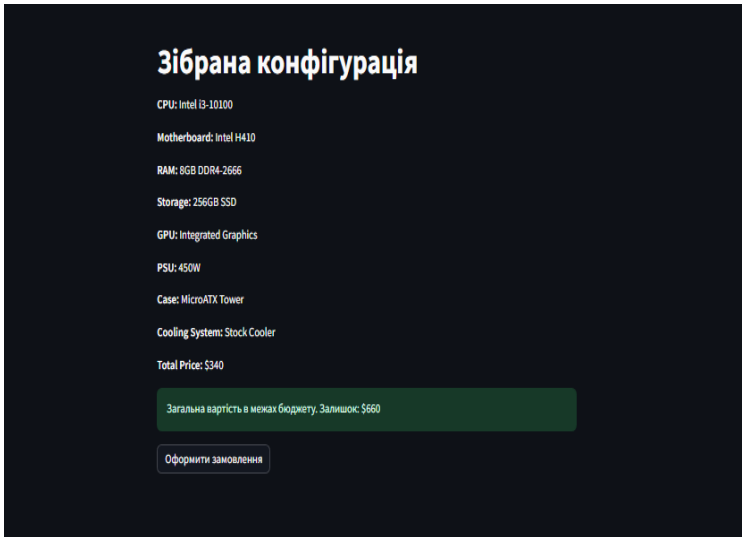

Рисунок 2.7 – Робота програми сторінка 3
Джерело: розроблено автором

Під час процесу конфігурації користувач може переглядати поточний стан зібраної конфігурації у бічній панелі. Ця панель також містить загальну вартість комплектуючих та дозволяє користувачеві переглянути залишок його бюджету (рис. 2.8).

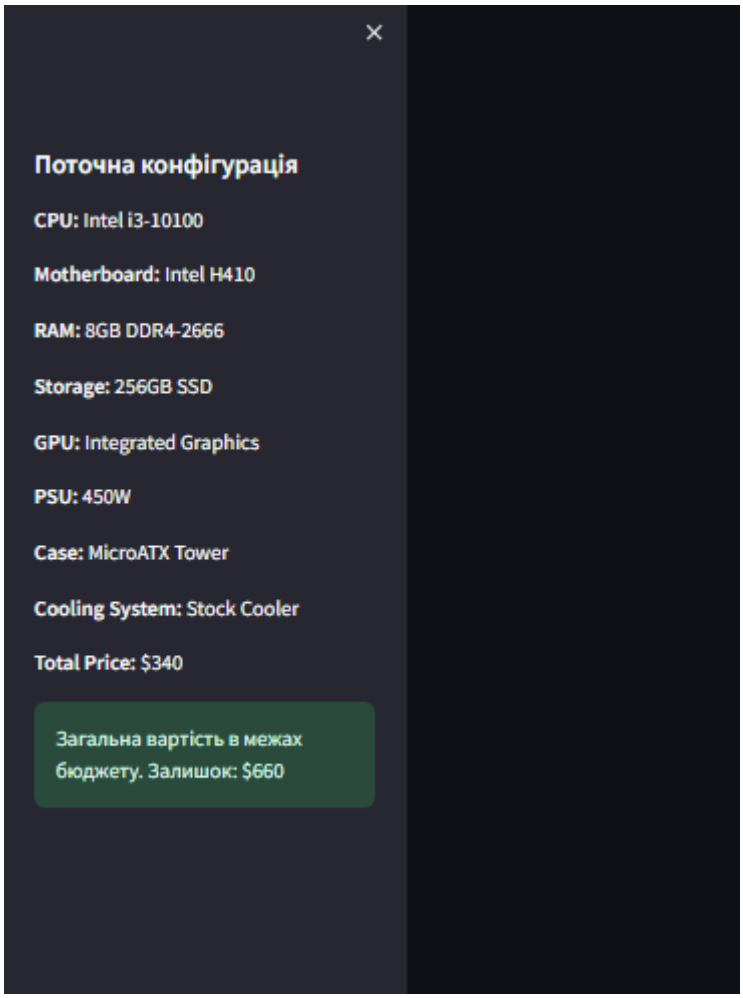

Рисунок 2.8 – Робота програми сторінка 3
Джерело: розроблено автором

Після завершення процесу конфігурації користувач переходить на сторінку перегляду зібраної конфігурації. Ця сторінка надає детальну інформацію про всі обрані комплектуючі, їхні специфікації та сумісність. Користувач може переглянути зображення компонентів, технічні характеристики та рекомендовані периферійні пристрої.

Нарешті, якщо користувач задоволений зібраною конфігурацією, він може перейти на сторінку оформлення замовлення. Ця сторінка містить форму, де користувач вводить свої контактні дані та інформацію про доставку.

Для розуміння всієї концепції дизайну веб-застосунку для конфігурування комп’ютерів спочатку було створено прототип, який постійно вдосконалювався. Прототип дозволив перевірити зручність користувацького

інтерфейсу, виявити потенційні проблеми та отримати відгуки від потенційних користувачів на ранніх етапах розробки.

На рисунку 2.9 представлено Interface Structure Diagram веб-застосунку для конфігурування комп’ютерів.

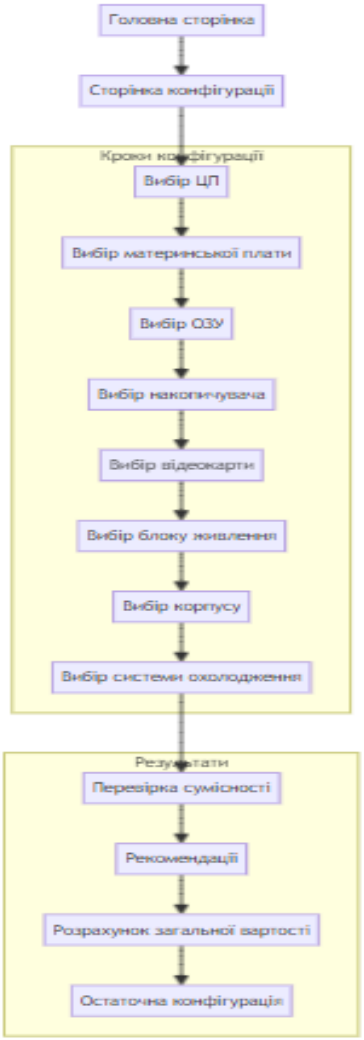

Рисунок 2.9 – Діаграма структури інтерфейсу (Interface Structure Diagram)
Джерело: розроблено автором

Діаграма структури інтерфейсу складається з ієрархічно організованих елементів, що відображають структуру веб-застосунку. Кожен елемент представляє окрему сторінку або компонент інтерфейсу, а стрілки між ними вказують на можливі переходи користувача.

На вершині діаграми знаходиться головна сторінка, яка є точкою входу для користувачів. Від неї розгалужуються дві гілки: одна веде до сторінки конфігурації, а інша – до сторінки про застосунок (з інформацією, FAQ тощо).

Сторінка конфігурації є центральним елементом діаграми та містить кілька вкладених компонентів, таких як дерево рішень, панель перегляду конфігурації та кнопки навігації. Від сторінки конфігурації користувач може перейти до сторінки перегляду зібраної конфігурації або повернутися на головну сторінку. Сторінка перегляду зібраної конфігурації дозволяє користувачеві детально розглянути обрані комплектуючі та їхні специфікації. З цієї сторінки користувач може перейти до сторінки оформлення замовлення або повернутися до сторінки конфігурації для внесення змін.

Сторінка оформлення замовлення містить форму для введення контактних даних та інформації про доставку. Після успішного оформлення замовлення користувач переходить на сторінку підтвердження замовлення. Діаграма структури інтерфейсу також відображає додаткові компоненти, такі як сторінка про застосунок, сторінка налаштувань та сторінка входу/реєстрації для авторизованих користувачів.

Використання діаграми структури інтерфейсу допомагає забезпечити логічну та зрозумілу організацію веб-застосунку, полегшуючи навігацію та взаємодію користувачів. Вона дозволяє розробникам візуалізувати та аналізувати взаємозв'язки між різними компонентами інтерфейсу, що сприяє ефективному проєктуванню та вдосконаленню користувацького досвіду.

Висновки до розділу 2

У цьому розділі розглянуто ключові аспекти проєктування веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень. Спроєктовано архітектуру системи.

Описано використання дерева рішень як потужного інструменту для підбору оптимальної конфігурації комп'ютера на основі заданих критеріїв та

обмежень. Розглянуто структуру дерева рішень, його складові елементи, переваги та обмеження використання. Наведено приклад дерева рішень для підбору комплектуючих з урахуванням бюджету, призначення комп'ютера та інших критеріїв.

Спроєктовано зручний інтерфейс користувача, який забезпечує інтуїтивну взаємодію з веб-застосунком. Створено прототип інтерфейсу та описано його основні компоненти: головну сторінку, сторінку конфігурації з інтерактивним деревом рішень, сторінку перегляду зібраної конфігурації та сторінку оформлення замовлення.

Розроблено діаграму структури інтерфейсу (Interface Structure Diagram), яка візуально представляє взаємозв’язки між різними компонентами веб-застосунку та можливі шляхи переходів користувача. Ця діаграма допомагає зрозуміти та проаналізувати логічну організацію інтерфейсу, полегшуючи процес проєктування та вдосконалення користувацького досвіду.

Загалом, у розділі описано ключові елементи проєктування веб-застосунку для конфігурування комп’ютерів, що забезпечить зручний та інтуїтивний процес підбору комплектуючих для широкого кола користувачів.

3 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

3.1 Вибір технологій для розробки

У веб-застосунку для конфігурування комп’ютерів використано бібліотеку Streamlit для створення клієнтської частини (Front-end). Streamlit – це інноваційна бібліотека для створення веб-додатків мовою Python, яка дозволяє швидко розробляти інтерактивні інтерфейси користувача за допомогою простого та інтуїтивно зрозумілого коду.

Вибір Streamlit, як технології для розробки клієнтської частини веб-застосунку, має ряд переваг:

1. Дозволяє створювати веб-інтерфейси за допомогою Python, який відомий своїм простим і зрозумілим синтаксисом. Це значно знижує криву навчання для розробників, особливо для тих, хто не має глибоких знань у веб-технологіях.
2. Прискорює процес розробки веб-застосунків, оскільки розробники можуть зосередитись на логіці додатку, а не на деталях реалізації інтерфейсу користувача. Бібліотека автоматично перетворює Python-код на веб-сторінки, скорочуючи час, необхідний для створення інтерфейсу.
3. Забезпечує широкий набір компонентів та можливостей для створення інтерактивних інтерфейсів користувача. Це дозволяє легко реалізувати такі елементи, як спливаючі списки, перемикачі, слайдери та інші інтерактивні елементи керування.
4. Має вбудовану підтримку для візуалізації даних, що дозволяє легко створювати різноманітні діаграми, графіки та таблиці прямо у веб-інтерфейсі.
5. Можна легко інтегрувати веб-інтерфейс з існуючим кодом на Python, використовуючи популярні бібліотеки та фреймворки, такі як NumPy, Pandas, Scikit-Learn та інші.

6. Веб-застосунки, створені за допомогою Streamlit, є крос-платформними та можуть бути розгорнуті на різних операційних системах та середовищах, включаючи локальний комп'ютер, хмарні платформи або сервери.

Однак, варто зазначити, що Streamlit має певні обмеження порівняно з традиційними фреймворками для розробки веб-інтерфейсів, такими як React, Angular або Vue.js. Ці фреймворки забезпечують більше можливостей для налаштування інтерфейсу, реактивності та організації коду, але вимагають більше часу та зусиль для освоєння.

Для реалізації серверної частини (Back-end) веб-застосунку в наведеному коді використовується Python, що надає можливості для вибору різних фреймворків для розробки веб-застосунків мовою Python, таких як Django, Flask або FastAPI [19].

Вибір конкретного фреймворку для серверної частини залежить від вимог до продуктивності, масштабованості, функціональності та уподобань розробників. Наприклад, Django є потужним фреймворком з багатим набором функцій та вбудованих рішень для веб-розробки, але може бути надмірно складним для невеликих проєктів. Flask, з іншого боку, є більш легковаговим та гнучким фреймворком, який дозволяє легко розширювати функціональність за допомогою плагінів.

Для зберігання даних використовується вкладений словник Python. Однак, для більш складних вимог зберігання даних, таких як підтримка транзакцій, забезпечення цілісності даних та масштабованості, рекомендується використовувати систему управління базами даних (СУБД). Потенційними варіантами СУБД для веб-застосунку можуть бути PostgreSQL або MySQL, які є популярними та надійними реляційними базами даних з відкритим кодом. PostgreSQL відомий своєю потужністю, надійністю та численними розширеннями, тоді як MySQL відрізняється високою продуктивністю та простотою у використанні.

Крім того, для деяких випадків використання, наприклад, для зберігання неструктурованих даних або забезпечення високої масштабованості, можна розглянути використання NoSQL баз даних, таких як MongoDB або Redis. Вибір конкретної СУБД залежить від вимог до продуктивності, масштабованості, типу даних, які потрібно зберігати, та досвіду розробників. Для забезпечення належної безпеки веб-застосунку необхідно впровадити відповідні заходи безпеки, такі як захист від вразливостей (XSS, CSRF, SQL-ін'єкцій), шифрування конфіденційних даних, авторизація та автентифікація користувачів, а також регулярне оновлення використовуваних бібліотек та фреймворків [22].

Загалом, вибір технологій для розробки веб-застосунку для конфігурування комп’ютерів є розумним і обґрунтованим. Streamlit забезпечує швидкість та простоту розробки клієнтської частини, а Python дозволяє гнучко обирати фреймворки та інструменти для реалізації серверної частини та роботи з даними. Однак, при масштабуванні проекту та розширенні функціональності, може виникнути потреба в перегляді або доповненні вибраного стеку технологій для задоволення нових вимог.

3.2 Реалізація дерева рішень

Одним із ключових компонентів веб-застосунку є реалізація дерева рішень для підбору сумісних комплектуючих. У проєкті дерево рішень реалізовано за допомогою структури даних – вкладеного словника Python [26].

Коренем дерева рішень у коді є ключ «CPU», який представляє першу умову – вибір процесора. Для кожного типу процесора (Intel i3-10100, Intel i5-10400, AMD Ryzen 3 3200G, AMD Ryzen 5 3400G) існує окремий вкладений словник, що містить ціну процесора та можливі варіанти вибору для наступних компонентів: материнської плати, оперативної пам'яті, накопичувача, відеокарти, блоку живлення, корпусу та системи охолодження.

Кожний компонент представлений як ключ словника, а його значення містить список доступних варіантів та відповідні ціни. Наприклад, для процесора «Intel i5-10400» доступні дві материнські плати: «Intel H410» і «Intel B460» з цінами 70 та 90 відповідно. Така структура дерева рішень дозволяє легко обробляти вибір користувача на кожному кроці та визначати сумісні варіанти для наступних компонентів. Після проходження всіх рівнів дерева, система може сформувати остаточну рекомендовану конфігурацію комп'ютера та підрахувати загальну ціну.

На рисунку 3.1 представлено структуру дерева рішень.

```
decision_tree = {
    "Office Work": {
        "CPU": {
            "Intel i3-10100": 100,
            "Intel i5-10400": 150,
            "AMD Ryzen 3 3200G": 100,
            "AMD Ryzen 5 3400G": 150
        },
        "Motherboard": {
            "Intel H410": 70,
            "Intel B460": 90,
            "AMD A320": 60,
            "AMD B450": 80
        },
        "RAM": {
            "8GB DDR4-2666": 40,
            "16GB DDR4-3200": 70,
            "32GB DDR4-3600": 130
        },
        "Storage": {
            "256GB SSD": 50,
            "512GB SSD": 80,
            "1TB SSD": 150,
            "1TB HDD": 40
        },
        "GPU": {
            "Integrated Graphics": 0,
            "NVIDIA GTX 1050": 120,
            "NVIDIA GTX 1650": 150,
            "AMD Radeon RX 560": 130
```

Рисунок 3.1 – Структура дерева рішень
Джерело: розроблено автором

Перевірка сумісності між компонентами здійснюється за допомогою функції check_compatibility (рис. 3.2).

```python
def check_compatibility(cpu, motherboard):
    if "Intel" in cpu and "Intel" not in motherboard:
        return False, "Selected motherboard is not compatible with the selected CPU."
    if "AMD" in cpu and "AMD" not in motherboard:
        return False, "Selected motherboard is not compatible with the selected CPU."
    return True, ""
```

Рисунок 3.2 – Функція check_compatibility
Джерело: розроблено автором

Дана функція перевіряє, чи вибрана материнська плата сумісна з обраним процесором, на основі наявності рядків «Intel» або «AMD» у їх назвах. Якщо материнська плата несумісна з процесором, функція повертає False та повідомлення про помилку. В іншому випадку, вона повертає True, вказуючи на сумісність компонентів.

Для взаємодії з користувачем та вибору компонентів використовується бібліотека Streamlit. Користувач послідовно проходить через вісім кроків, на кожному з яких обирає необхідний компонент за допомогою елемента st.selectbox (рис. 3.3).

```python
# Крок 1: Вибір CPU
st.header("Крок 1: Виберіть CPU")
cpu_options = list(decision_tree[selected_purpose]["CPU"].keys())
selected_cpu = st.selectbox("Виберіть CPU:", cpu_options)
st.session_state.selected_cpu = selected_cpu
```

Рисунок 3.3 – Функція st.selectbox
Джерело: розроблено автором

Після вибору всіх компонентів система перевіряє їхню сумісність за допомогою функції check_compatibility. Якщо компоненти сумісні, обчислюється загальна ціна конфігурації та виводиться остаточний результат у зручному вигляді (рис. 3.4).

```
compatible, message = check_compatibility(selected_cpu, selected_motherboard)
if not compatible:
    st.error(message)
else:
    st.session_state.total_price = current_total

    # Відображення поточної конфігурації
    st.sidebar.header("Поточна конфігурація")
    st.sidebar.write(f"**CPU:** {selected_cpu}")
    st.sidebar.write(f"**Motherboard:** {selected_motherboard}")
    st.sidebar.write(f"**RAM:** {selected_ram}")
    st.sidebar.write(f"**Storage:** {selected_storage}")
    st.sidebar.write(f"**GPU:** {selected_gpu}")
    st.sidebar.write(f"**PSU:** {selected_psu}")
    st.sidebar.write(f"**Case:** {selected_case}")
    st.sidebar.write(f"**Cooling System:** {selected_cooling}")
    st.sidebar.write(f"**Total Price:** ${current_total}")
```

Рисунок 3.4 – Функція check_compatibility
Джерело розроблено автором

Така реалізація дерева рішень є гнучкою та легко масштабованою. Додавання нових компонентів або варіантів вибору може бути здійснено шляхом модифікації структури словника. Крім того, дана реалізація забезпечує простоту та зрозумілість для розробників, полегшуючи подальшу підтримку та розвиток веб-застосунку [9].

3.3 Реалізація інтерфейсу користувача

Для розробки інтерфейсу у даному проєкті використовується Streamlit – потужна бібліотека для створення інтерактивних веб-додатків мовою Python. Streamlit дозволяє швидко розробляти інтуїтивні та привабливі інтерфейси, забезпечуючи широкий набір компонентів та можливостей для візуалізації даних. На рисунку 3.5 представлено головну сторінку веб-застосунку.

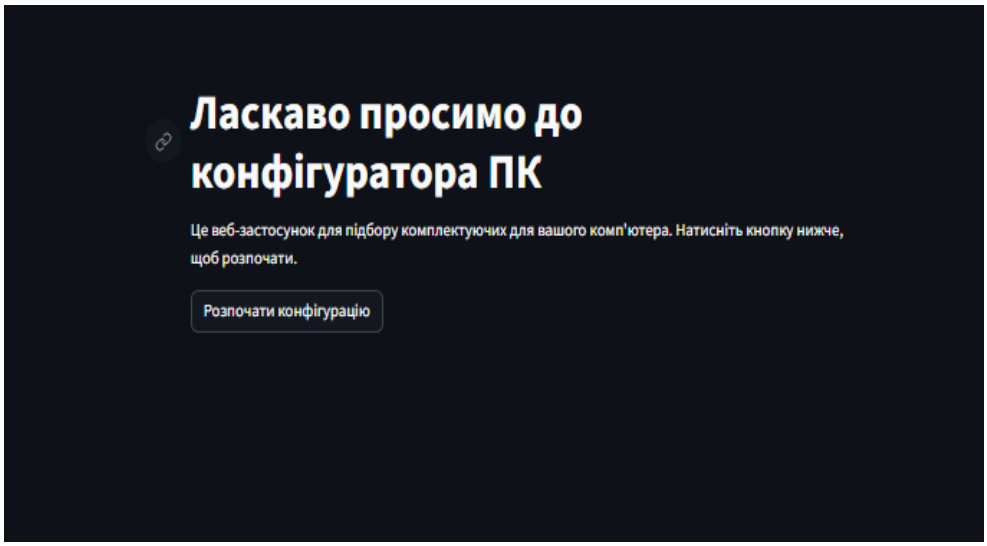

Рисунок 3.5 – Інтерфейс програми
Джерело: розроблено автором

Код розпочинається з імпорту бібліотеки Streamlit та визначення структури дерева рішень для конфігурування комп’ютера. Ця структура містить інформацію про доступні компоненти, їхні характеристики та ціни, що буде використовуватися для формування рекомендацій та розрахунку загальної вартості. Далі визначаються дві допоміжні функції check_compatibility для перевірки сумісності обраних компонентів та apply_dark_mode для застосування темного режиму до інтерфейсу веб-застосунку. Остання функція використовує вбудовані можливості Streamlit для додавання стилів CSS та зміни кольорової схеми інтерфейсу.

Основний код інтерфейсу користувача починається з заголовка та короткого вступного тексту, встановлених за допомогою функцій st.title та st.write. Після цього слідує серія кроків, на кожному з яких користувачу пропонується вибрати певний компонент комп’ютера [5].

Для вибору компонентів використовується функція st.selectbox, яка відображає меню, що випадає, з доступними опціями (рис. 3.6). Наприклад, для вибору процесора (CPU) потрібно виконати дії, описані нижче.

```
st.header("Крок 1: Виберіть CPU")
cpu_options = list(decision_tree[selected_purpose]["CPU"].keys())
selected_cpu = st.selectbox("Виберіть CPU:", cpu_options)
st.session_state.selected_cpu = selected_cpu
```

Рисунок 3.6 – Функція st.selectbox
Джерело: розроблено автором

Спочатку встановлюється заголовок кроку за допомогою st.header. Далі список доступних опцій для процесора отримується з дерева рішень, після чого користувачу пропонується вибрати одну опцію за допомогою st.selectbox. Обраний варіант зберігається у змінній selected_cpu та відображається на сторінці за допомогою st.write.

Аналогічний процес повторюється для вибору інших компонентів, таких як материнська плата, оперативна пам'ять, накопичувачі даних, відеокарта, блок живлення, корпус та система охолодження.

Після вибору всіх компонентів виконується перевірка їхньої сумісності за допомогою функції check_compatibility. Якщо виявлено несумісність, користувачеві відображається повідомлення про помилку за допомогою st.error. В іншому випадку, код обчислює загальну вартість обраної конфігурації та відображає її разом з остаточним переліком обраних компонентів [10].

Наприкінці, за допомогою функцій st.header та st.write відображається заголовок «Final Configuration» та деталі обраної конфігурації комп'ютера, включаючи назви компонентів та загальну вартість.

Інтерфейс користувача, створений за допомогою Streamlit, є простим та інтуїтивно зрозумілим. Він чітко структурований шляхом поділу на кроки, на кожному з яких користувачеві пропонується зробити вибір з доступних опцій. Випадаючі меню забезпечують зручність вибору, не перевантажуючи інтерфейс зайвими елементами.

Однак, поточна реалізація має кілька обмежень та потенційних можливостей для вдосконалення. По-перше, інтерфейс не надає детальної

інформації про характеристики кожного компонента, що може ускладнити прийняття рішення для користувачів, які не мають глибоких технічних знань. Додавання стислих описів або специфікацій компонентів може значно покращити користувацький досвід.

По-друге, інтерфейс не надає жодної візуалізації або попереднього перегляду обраної конфігурації. Додавання фотографій або 3D-моделей компонентів, а також можливості переглянути їх у зібраному вигляді, може зробити процес більш наочним та привабливим для користувачів.

Крім того, існує можливість для додавання додаткових функцій, таких як порівняння декількох конфігурацій, збереження та завантаження попередніх конфігурацій, інтеграція з магазинами комплектуючих для безпосереднього замовлення обраних компонентів тощо.

Загалом, наведений код демонструє основні принципи проєктування інтерфейсу користувача для веб-застосунку з використанням дерева рішень для конфігурування комп'ютерів. Він забезпечує структурований та інтуїтивний процес вибору компонентів, перевірку їх сумісності та розрахунок загальної вартості конфігурації. Однак, існує значний простір для вдосконалення інтерфейсу, додавання нових функцій та покращення візуалізації, щоб забезпечити ще кращий користувацький досвід.

3.4 Тестування веб-застосунку

Тестування є невід’ємною частиною процесу розробки програмного забезпечення, оскільки воно дозволяє виявляти та усувати дефекти, перевіряти відповідність функціональним вимогам, забезпечувати належну якість коду та гарантувати коректну роботу системи в різних умовах. Для веб-застосунку конфігурування комп’ютерів необхідно провести ретельне тестування на різних рівнях, включаючи модульне тестування, інтеграційне тестування, та системне тестування.

Модульне тестування (Unit Testing) полягає у перевірці окремих модулів або функцій програми на коректність роботи та відповідність очікуваним результатам. Для веб-застосунку Для конфігурування комп’ютерів використано вбудований фреймворк для модульного тестування unittest або популярний фреймворк pytest.

Наприклад, створено модульний тест для перевірки функції check_compatibility, яка визначає сумісність процесора та материнської плати:

```
import unittest
from compatibility import check_compatibility
class TestCompatibility(unittest.TestCase):
    def test_compatible_cpu_motherboard_intel(self):
        cpu = "Intel i5-10400"
        motherboard = "Intel B460"
        is_compatible, _ = check_compatibility(cpu, motherboard)
        self.assertTrue(is_compatible)
    def test_compatible_cpu_motherboard_amd(self):
        cpu = "AMD Ryzen 5 3400G"
        motherboard = "AMD B450"
        is_compatible, _ = check_compatibility(cpu, motherboard)
        self.assertTrue(is_compatible)
    def test_incompatible_cpu_motherboard_intel_amd(self):
        cpu = "Intel i5-10400"
        motherboard = "AMD B450"
        is_compatible, error_message = check_compatibility(cpu, motherboard)
        self.assertFalse(is_compatible)
        self.assertEqual(error_message, "Selected motherboard is not compatible with the selected CPU.")
    def test_incompatible_cpu_motherboard_amd_intel(self):
        cpu = "AMD Ryzen 5 3400G"
        motherboard = "Intel B460"
        is_compatible, error_message = check_compatibility(cpu, motherboard)
        self.assertFalse(is_compatible)
        self.assertEqual(error_message, "Selected motherboard is not compatible with the selected CPU.")
if __name__ == "__main__":  unittest.main()
```

У цьому прикладі створено клас TestCompatibility, який успадковує від unittest.TestCase. Всередині класу визначено два методи тестування test_compatible_cpu_motherboard та test_incompatible_cpu_motherboard. Перший метод перевіряє, що функція check_compatibility повертає True для сумісних процесора та материнської плати, а другий метод перевіряє, що функція повертає False та відповідне повідомлення про помилку для несумісних компонентів.

Модульні тести допомагають виявляти помилки на ранніх стадіях розробки та забезпечують швидку зворотний зв'язок про коректність роботи окремих модулів програми [7].

Інтеграційне тестування (Integration Testing) полягає у перевірці взаємодії між різними модулями або компонентами системи. Для веб-застосунку створено тести для перевірки інтеграції між серверною частиною (API) та клієнтською частиною (інтерфейсом користувача), а також інтеграції з базою даних.

Наприклад, використано бібліотеку requests для відправки HTTP-запитів до API та перевірки відповідей:

```
import requests
import unittest
class TestIntegration(unittest.TestCase):
    BASE_URL = "http://127.0.0.1:8501/api"
    def test_get_components(self):
        response = requests.get(f"{self.BASE_URL}/components")
        self.assertEqual(response.status_code, 200)
        components = response.json()
        self.assertGreater(len(components), 0)
        self.assertIn("Office Work", components)
        self.assertIn("Gaming", components)
        self.assertIn("Multimedia", components)
    def test_create_configuration(self):
        data = {
```

```
            "cpu": "Intel i5-10400",
            "motherboard": "Intel B460",
            "ram": "16GB DDR4-3200",
            "storage": "512GB SSD",
            "gpu": "NVIDIA GTX 1650",
            "psu": "500W",
            "case": "Mid Tower",
            "cooling": "Air Cooling"
        }
        response = requests.post(f"{self.BASE_URL}/configurations", json=data)
        self.assertEqual(response.status_code, 201)
        config_id = response.json()["id"]
        response = requests.get(f"{self.BASE_URL}/configurations/{config_id}")
        self.assertEqual(response.status_code, 200)
        config = response.json()
        self.assertEqual(config["cpu"], "Intel i5-10400")
        self.assertEqual(config["motherboard"], "Intel B460")
        self.assertEqual(config["ram"], "16GB DDR4-3200")
        self.assertEqual(config["storage"], "512GB SSD")
        self.assertEqual(config["gpu"], "NVIDIA GTX 1650")
        self.assertEqual(config["psu"], "500W")
        self.assertEqual(config["case"], "Mid Tower")
        self.assertEqual(config["cooling"], "Air Cooling")
if __name__ == "__main__":
    unittest.main()
```

У цьому прикладі тестується API для отримання списку доступних компонентів та створення нової конфігурації комп'ютера. Перший тест перевіряє, що API повертає список компонентів з очікуваними категоріями, такими як процесор та материнська плата. Другий тест створює нову конфігурацію комп’ютера за допомогою HTTP POST запиту, перевіряє, що API повернув успішний статус-код 201 (Created), а потім виконує GET запит для перевірки збереженої конфігурації [5].

Інтеграційне тестування допомагає виявляти помилки на рівні взаємодії між компонентами та забезпечує коректну роботу системи в цілому.

Висновки до розділу 3

Для реалізації веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень обрано Streamlit для створення клієнтської частини веб-застосунку завдяки простоті розробки інтерактивних інтерфейсів.

Обґрунтовано використання фреймворків Django, Flask або FastAPI для розробки серверної частини, а також СУБД, таких як PostgreSQL або MySQL, для зберігання даних.

Реалізовано дерево рішень за допомогою вкладеного словника Python, що забезпечує гнучкість та масштабованість.

Наведено приклади коду для створення модульних тестів з використанням unittest або pytest, а також інтеграційних тестів для перевірки взаємодії компонентів. Визначено інструменти автоматизованого тестування, для перевірки інтерфейсу користувача. Підкреслено важливість системного тестування, включаючи тестування продуктивності, навантаження, стрес-тестування, тестування безпеки та сумісності.

Проведено тестування ключових компонентів веб-застосунку для забезпечення коректної роботи та відповідності функціональним вимогам.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено веб-застосунок для конфігурування комп’ютерів, який дозволяє користувачам підібрати сумісні комплектуючі та сформувати остаточну конфігурацію комп’ютера відповідно до їхніх вимог та бюджету.

Розглянуто предметну область конфігурування персональних комп’ютерів та проаналізовано існуючі рішення на ринку веб-застосунків. Обґрунтовано актуальність і необхідність розробки нового веб-застосунку для конфігурування комп’ютерів з використанням дерева рішень для підбору комплектуючих.

Спроєктовано архітектуру системи та зручний інтерфейс користувача з використанням дерева рішень як потужного інструменту для підбору оптимальної конфігурації.

Розроблено діаграму структури інтерфейсу, що візуально представляє взаємозв’язки між компонентами веб-застосунку та можливі шляхи переходів користувача.

Для реалізації клієнтської частини обрано Streamlit завдяки простоті розробки інтерактивних інтерфейсів. Дерево рішень реалізовано за допомогою вкладеного словника Python, що забезпечує гнучкість та масштабованість. Наведено приклади модульного та інтеграційного тестування, а також визначено інструменти автоматизованого тестування інтерфейсу.

Підкреслено важливість системного тестування, що містить тестування продуктивності, навантаження, стрес-тестування, тестування безпеки та сумісності. Проведено тестування ключових компонентів веб-застосунку для забезпечення коректної роботи та відповідності функціональним вимогам.

Таким чином, розроблений програмний продукт є зручним та ефективним інструментом, що дозволяє користувачам швидко і точно підібрати сумісні комплектуючі відповідно до їхніх потреб і бюджету.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Буй Д.Б., Цідило І.М., Бурачок О.В. "Основи тестування програмного забезпечення". Навчальний посібник. – К. ДУТ, 2018. – 212 с.
2. Глушков С.В. "Тестування програмного забезпечення". Навчальний посібник. – Харків ХНУ імені В.Н. Каразіна, 2018. – 160 с.
3. Голіков І.В. "Тестування веб-застосунків". Навчальний посібник. – Дніпро ДВНЗ УДХТУ, 2019. – 128 с.
4. Грицюк Ю.І., Грицюк П.Ю. "Тестування програмного забезпечення. Практикум". – Рівне НУВГП, 2017. – 128 с.
5. Корольков В.В. "Python для веб-розробки". Навчальний посібник. – Київ КПІ ім. Ігоря Сікорського, 2021. – 256 с.
6. Кузьменко О.В., Сапунов В.В. "Тестування програмного забезпечення". Підручник. – Київ КПІ ім. Ігоря Сікорського, 2020. – 360 с.
7. Лавріщева К.М. "Програмна інженерія". Підручник. – Київ Академперіодика, 2008. – 312 с.
8. Ляшенко О.М., Коваленко Є.В. "Тестування програмного забезпечення". Навчальний посібник. – Запоріжжя ЗНТУ, 2019. – 152 с.
9. Маєвський Д.А. "Тестування програмного забезпечення теорія та практика". – Львів Видавництво Львівської політехніки, 2021. – 320 с.
10. Мороз Г.Б. "Тестування програмного забезпечення". Навчальний посібник. – Вінниця ВНТУ, 2017. – 176 с.
11. Онай М.В., Коваль Г.І. "Основи тестування програмного забезпечення". Підручник. – Львів Видавництво Львівської політехніки, 2020. – 328 с.
12. Оржеховська А.В. "Тестування програмного забезпечення стратегії та методи". Монографія. – Київ Наукова думка, 2018. – 360 с.
13. Пілічева М.О., Ройко В.В. "Тестування веб-додатків". Навчальний посібник. – Одеса ОНПУ, 2020. – 160 с.
14. Стасюк О.І. "Тестування програмного забезпечення". Підручник. – Львів Видавництво Львівської політехніки, 2019. – 400 с.

15. Федасюк Д.В., Федорчук А.В. "Основи тестування програмного забезпечення". Навчальний посібник. – Луцьк СНУ ім. Лесі Українки, 2018. – 160 с.

16. Андон П.І., Яцишин В.В., Луцький М.Г. "Тестування та діагностика компонентів розподілених систем". Підручник. – Львів Видавництво Львівської політехніки, 2017. – 360 с.

17. Браницький І.І. "Тестування програмного забезпечення на основі моделей". Монографія. – Київ Видавництво НАУ, 2020. – 288 с.

18. Головатий Р.Я., Драчук В.О. "Тестування веб-застосунків з використанням хмарних технологій". Навчальний посібник. – Тернопіль ТНТУ ім. І. Пулюя, 2019. – 160 с.

19. Дудкін В.П., Сукач Є.І. "Тестування програмного забезпечення інструменти та методи". Навчальний посібник. – Харків ХНУ ім. В.Н. Каразіна, 2021. – 208 с.

20. Калитюк В.А., Петрик М.Р., Бевз С.В. "Тестування та верифікація програмного забезпечення". Підручник. – Львів Видавництво Львівської політехніки, 2020. – 400 с.

21. Карпов Ю.Г., Романко В.С. "Тестування та якість програмного забезпечення". Навчальний посібник. – Київ Інтерсервіс, 2018. – 240 с.

22. Лапська О.В., Муравйова І.Г. "Тестування та діагностика програмного забезпечення". Підручник. – Одеса ОНПУ, 2019. – 328 с.

23. Лісовичок Ю.Ю. "Тестування програмного забезпечення в розподілених системах". Монографія. – Львів Видавництво Львівської політехніки, 2021. – 304 с.

24. Мамчур В.М., Пархоменко А.В. "Тестування та метрики програмного забезпечення". Підручник. – Дніпро ДВНЗ УДХТУ, 2018. – 272 с.

25. Микитюк С.О., Яцишин В.В. "Тестування та діагностика програмного забезпечення". Навчальний посібник. – Тернопіль ТНТУ ім. І. Пулюя, 2020. – 192 с.

26. Павловський Ю.А., Литвиненко В.І. "Тестування веб-застосунків". Навчальний посібник. – Харків ХНУ ім. В.Н. Каразіна, 2019. – 160 с.

27. Петрик М.Р., Мельник А.О. "Тестування та забезпечення якості програмного забезпечення". Підручник. – Львів Видавництво Львівської політехніки, 2017. – 368 с.

28. Романкевич В.О., Русин Б.П. "Тестування та діагностика розподілених систем". Монографія. – Київ Наукова думка, 2019. – 312 с.

29. Сидоренко В.М., Ліпко І.В. "Тестування програмного забезпечення в агільних методологіях розробки". Навчальний посібник. – Київ КПІ ім. Ігоря Сікорського, 2021. – 192 с.

30. NZXT BLD. URL https //www.nzxt.com/bldnow

31. CyberPowerPC. URL https //www.cyberpowerpc.com/

32. Puget Systems. URL https //www.pugetsystems.com/

33. IBuyPower. URL https //www.ibuypower.com/

34. PCPartPicker. URL https //pcpartpicker.com/

35. Logical Increments. URL https //logicalincrements.com/

ДОДАТОК А ФРАГМЕНТИ ЛІТИНГУ

```
import streamlit as st
# Структура даних дерева рішень
decision_tree = {
    "Office Work": {
        "CPU": {
            "Intel i3-10100": 100,
            "Intel i5-10400": 150,
            "AMD Ryzen 3 3200G": 100,
            "AMD Ryzen 5 3400G": 150
        },
        "Motherboard": {
            "Intel H410": 70,
            "Intel B460": 90,
            "AMD A320": 60,
            "AMD B450": 80
        },
        "RAM": {
            "8GB DDR4-2666": 40,
            "16GB DDR4-3200": 70,
            "32GB DDR4-3600": 130
        },
        "Storage": {
            "256GB SSD": 50,
            "512GB SSD": 80,
            "1TB SSD": 150,
            "1TB HDD": 40
        },
        "GPU": {
            "Integrated Graphics": 0,
            "NVIDIA GTX 1050": 120,
            "NVIDIA GTX 1650": 150,
            "AMD Radeon RX 560": 130
        },
        "PSU": {
            "450W": 40,
            "500W": 50,
            "600W": 60
        },
        "Case": {
            "MicroATX Tower": 40,
            "Mid Tower": 60
        },
        "Cooling": {
            "Stock Cooler": 0,
            "Air Cooling": 30
        }
    },
    "Gaming": {
```

```
    "CPU": {
      "Intel i5-10600K": 250,
      "Intel i7-10700K": 350,
      "AMD Ryzen 7 3700X": 320,
      "AMD Ryzen 9 3900X": 450
    },
    "Motherboard": {
      "Intel Z490": 150,
      "Intel Z590": 200,
      "AMD X470": 150,
      "AMD X570": 200
    },
    "RAM": {
      "16GB DDR4-3200": 70,
      "32GB DDR4-3600": 130,
      "64GB DDR4-4000": 250
    },
    "Storage": {
      "512GB SSD": 80,
      "1TB SSD": 150,
      "2TB SSD": 300,
      "2TB HDD": 60
    },
    "GPU": {
      "NVIDIA GTX 1660 Super": 250,
      "NVIDIA RTX 2060": 400,
      "NVIDIA RTX 3070": 700,
      "AMD Radeon RX 5700 XT": 450,
      "AMD Radeon RX 6800 XT": 650
    },
    "PSU": {
      "600W": 60,
      "750W": 90,
      "850W": 110
    },
    "Case": {
      "Mid Tower": 60,
      "Full Tower": 100,
      "Gaming Case with RGB": 150
    },
    "Cooling": {
      "Air Cooling": 30,
      "Liquid Cooling": 100,
      "Custom Liquid Cooling": 200
    }
  },
  "Multimedia": {
    "CPU": {
      "Intel i5-11400": 180,
      "Intel i7-11700": 320,
      "AMD Ryzen 5 3600": 200,
      "AMD Ryzen 7 5800X": 340
```

```
        },
        "Motherboard": {
            "Intel B560": 110,
            "Intel Z590": 200,
            "AMD B450": 100,
            "AMD X570": 200
        },
        "RAM": {
            "8GB DDR4-2666": 40,
            "16GB DDR4-3200": 70,
            "32GB DDR4-3600": 130
        },
        "Storage": {
            "256GB SSD": 50,
            "512GB SSD": 80,
            "1TB SSD": 150,
            "2TB HDD": 60
        },
        "GPU": {
            "NVIDIA GTX 1650": 150,
            "NVIDIA GTX 1660": 220,
            "NVIDIA RTX 3060": 500,
            "AMD Radeon RX 570": 140,
            "AMD Radeon RX 580": 200
        },
        "PSU": {
            "450W": 40,
            "500W": 50,
            "600W": 60
        },
        "Case": {
            "MicroATX Tower": 40,
            "Mid Tower": 60,
            "Full Tower": 100
        },
        "Cooling": {
            "Stock Cooler": 0,
            "Air Cooling": 30,
            "Liquid Cooling": 100
        }
    }
}

# Функція перевірки сумісності компонентів
def check_compatibility(cpu, motherboard):
    if "Intel" in cpu and "Intel" not in motherboard:
        return False, "Selected motherboard is not compatible with the selected CPU."
    if "AMD" in cpu and "AMD" not in motherboard:
        return False, "Selected motherboard is not compatible with the selected CPU."
    return True, ""

# Головна сторінка
```

```
def main_page():
    st.title("Ласкаво просимо до конфігуратора ПК")
    st.write("Це веб-застосунок для підбору комплектуючих для вашого комп'ютера. Натисніть кнопку нижче, щоб розпочати.")
    if st.button("Розпочати конфігурацію"):
        st.session_state.page = "configuration"

# Сторінка конфігурації
def configuration_page():
    st.title("Конфігурація ПК")

    # Введення бюджету
    st.header("Вкажіть свій бюджет")
    budget = st.number_input("Бюджет ($):", min_value=300, max_value=3000, value=1000, step=50)
    st.session_state.budget = budget

    # Визначення категорії в залежності від бюджету
    if budget >= 330 and budget <= 820:
        budget_category = "Office Work"
    elif budget >= 610 and budget <= 1580:
        budget_category = "Multimedia"
    elif budget >= 950 and budget <= 2360:
        budget_category = "Gaming"
    else:
        budget_category = "Unknown"

    st.header("Категорія ПК за бюджетом")
    st.write(f"Ваш бюджет підходить для: {budget_category}")

    # Введення призначення комп'ютера
    st.header("Вкажіть призначення комп'ютера")
    purpose_options = list(decision_tree.keys())
    selected_purpose = st.selectbox("Призначення:", purpose_options)
    st.session_state.selected_purpose = selected_purpose

    # Крок 1: Вибір CPU
    st.header("Крок 1: Виберіть CPU")
    cpu_options = list(decision_tree[selected_purpose]["CPU"].keys())
    selected_cpu = st.selectbox("Виберіть CPU:", cpu_options)
    st.session_state.selected_cpu = selected_cpu

    selected_cpu_price = decision_tree[selected_purpose]["CPU"][selected_cpu]
    current_total = selected_cpu_price

    # Фільтрація материнських плат
    if "Intel" in selected_cpu:
        motherboard_options = [mb for mb in decision_tree[selected_purpose]["Motherboard"].keys() if "Intel" in mb]
    elif "AMD" in selected_cpu:
        motherboard_options = [mb for mb in decision_tree[selected_purpose]["Motherboard"].keys() if "AMD" in mb]
```

```
else:
    motherboard_options = list(decision_tree[selected_purpose]["Motherboard"].keys())

# Крок 2: Вибір Motherboard
st.header("Крок 2: Виберіть Motherboard")
selected_motherboard = st.selectbox("Виберіть Motherboard:", motherboard_options)
st.session_state.selected_motherboard = selected_motherboard
current_total += decision_tree[selected_purpose]["Motherboard"][selected_motherboard]

# Крок 3: Вибір RAM
st.header("Крок 3: Виберіть RAM")
ram_options = list(decision_tree[selected_purpose]["RAM"].keys())
selected_ram = st.selectbox("Виберіть RAM:", ram_options)
st.session_state.selected_ram = selected_ram
current_total += decision_tree[selected_purpose]["RAM"][selected_ram]

# Крок 4: Вибір Storage
st.header("Крок 4: Виберіть Storage")
storage_options = list(decision_tree[selected_purpose]["Storage"].keys())
selected_storage = st.selectbox("Виберіть Storage:", storage_options)
st.session_state.selected_storage = selected_storage
current_total += decision_tree[selected_purpose]["Storage"][selected_storage]

# Крок 5: Вибір GPU
st.header("Крок 5: Виберіть GPU")
gpu_options = list(decision_tree[selected_purpose]["GPU"].keys())
selected_gpu = st.selectbox("Виберіть GPU:", gpu_options)
st.session_state.selected_gpu = selected_gpu
current_total += decision_tree[selected_purpose]["GPU"][selected_gpu]

# Крок 6: Вибір PSU
st.header("Крок 6: Виберіть PSU")
psu_options = list(decision_tree[selected_purpose]["PSU"].keys())
selected_psu = st.selectbox("Виберіть PSU:", psu_options)
st.session_state.selected_psu = selected_psu
current_total += decision_tree[selected_purpose]["PSU"][selected_psu]

# Крок 7: Вибір Case
st.header("Крок 7: Виберіть Case")
case_options = list(decision_tree[selected_purpose]["Case"].keys())
selected_case = st.selectbox("Виберіть Case:", case_options)
st.session_state.selected_case = selected_case
current_total += decision_tree[selected_purpose]["Case"][selected_case]

# Крок 8: Вибір Cooling System
st.header("Крок 8: Виберіть Cooling System")
cooling_options = list(decision_tree[selected_purpose]["Cooling"].keys())
selected_cooling = st.selectbox("Виберіть Cooling System:", cooling_options)
st.session_state.selected_cooling = selected_cooling
current_total += decision_tree[selected_purpose]["Cooling"][selected_cooling]
```

```
    compatible, message = check_compatibility(selected_cpu, selected_motherboard)
    if not compatible:
        st.error(message)
    else:
        st.session_state.total_price = current_total

        # Відображення поточної конфігурації
        st.sidebar.header("Поточна конфігурація")
        st.sidebar.write(f"**CPU:** {selected_cpu}")
        st.sidebar.write(f"**Motherboard:** {selected_motherboard}")
        st.sidebar.write(f"**RAM:** {selected_ram}")
        st.sidebar.write(f"**Storage:** {selected_storage}")
        st.sidebar.write(f"**GPU:** {selected_gpu}")
        st.sidebar.write(f"**PSU:** {selected_psu}")
        st.sidebar.write(f"**Case:** {selected_case}")
        st.sidebar.write(f"**Cooling System:** {selected_cooling}")
        st.sidebar.write(f"**Total Price:** ${current_total}")

        if current_total > budget:
            st.sidebar.error(f"Загальна вартість перевищує бюджет на ${current_total - budget}!")
        else:
            st.sidebar.success(f"Загальна вартість в межах бюджету. Залишок: ${budget - current_total}")

        if st.button("Переглянути зібрану конфігурацію"):
            st.session_state.page = "review"

# Сторінка перегляду зібраної конфігурації
def review_page():
    st.title("Зібрана конфігурація")
    st.write(f"**CPU:** {st.session_state.selected_cpu}")
    st.write(f"**Motherboard:** {st.session_state.selected_motherboard}")
    st.write(f"**RAM:** {st.session_state.selected_ram}")
    st.write(f"**Storage:** {st.session_state.selected_storage}")
    st.write(f"**GPU:** {st.session_state.selected_gpu}")
    st.write(f"**PSU:** {st.session_state.selected_psu}")
    st.write(f"**Case:** {st.session_state.selected_case}")
    st.write(f"**Cooling System:** {st.session_state.selected_cooling}")
    st.write(f"**Total Price:** ${st.session_state.total_price}")

    if st.session_state.total_price > st.session_state.budget:
        st.error(f"Загальна вартість перевищує бюджет на ${st.session_state.total_price - st.session_state.budget}!")
    else:
        st.success(f"Загальна вартість в межах бюджету. Залишок: ${st.session_state.budget - st.session_state.total_price}")

    if st.button("Оформити замовлення"):
        st.session_state.page = "order"

def order_page():
    st.title("Оформлення замовлення")
```

```
    st.write("Будь ласка, введіть ваші контактні дані для оформлення замовлення.")
    user_name = st.text_input("Ім'я")
    user_email = st.text_input("Email")
    user_address = st.text_area("Адреса доставки")

    if st.button("Підтвердити замовлення"):
        st.write(f"Дякуємо, {user_name}! Ваше замовлення було оформлене.")


if "page" not in st.session_state:
    st.session_state.page = "main"

if st.session_state.page == "main":
    main_page()
elif st.session_state.page == "configuration":
    configuration_page()
elif st.session_state.page == "review":
    review_page()
elif st.session_state.page == "order":
    order_page()
```