ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РОЗРОБКА САЙТУ «КОНФІГУРАТОР ПК» З АНАЛІЗОМ ПРОДУКТИВНОСТІ»

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Березенко Роман Мирославович

_____ «___» _____ 202_ р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Чілікіна Тетяна Василівна

_____ «___» _____ 202_ р.
(підпис)

Рецензент _____

ПОЛТАВА 2025

ЗАТВЕРДЖУЮ
Завідувач кафедри _____ Олена ОЛЬХОВСЬКА
«____» вересня 202__ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка сайту «Конфігуратор ПК» з аналізом продуктивності»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра
Прізвище, ім'я, по батькові Березенко Роман Мирославович
Затверджена наказом ректора № ___-Н від «__» _____ 202__ р.
Термін подання студентом роботи «__» _____ 202__ р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки.
Зміст пояснювальної записки:
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Огляд конфігураторів ПК
2.2. Підсумки аналізу конфігураторів
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Фізична та логічна структура веб-додатку
3.2. Обґрунтування вибору середовища розробки
3.3. Обґрунтування вибору технологій розробки
3.4. Семантична розмітка та доступність
4. ПРАКТИЧНА ЧАСТИНА
4.1. Завантаження та налаштування необхідних інструментів
4.2. Створення бази даних
4.3. Розробка серверної частини
4.4. Розробка клієнтної частини
4.5. Інструкція користувача
4.6. Аналіз адаптивності та швидкодії сайту
ВИСНОВКИ
СПИСОК ЛІТЕРАТУРИ
ДОДАТОК А
Перелік графічного матеріалу: 73 рисунки.

Консультанти розділів кваліфікаційної роботи

Розділ	ПІБ, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі			
Інформаційний огляд			
Теоретична частина			
Практична реалізація			

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка завдання		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доопрацювання (за необхідності), рецензування		

Дата видачі завдання «____»_____ 202__ р.

Здобувач вищої освіти Березенко Роман Мирославович

Науковий керівник к. ф.-м. н., Чілікіна Тетяна Василівна

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____»_____ 202__ р.

Секретар ЕК _____ _____

(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к. ф.-м. н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р

Погоджено
Науковий керівник _____
_____
«____» _____ 202_ р

План

кваліфікаційної роботи на тему
«Розробка сайту «Конфігуратор ПК» з аналізом продуктивності»
зі спеціальності 122 Комп’ютерні науки
освітня програма 122 «Комп’ютерні науки»
ступеня бакалавр
Прізвище, ім’я, по батькові Березенко Роман Мирославович

Здобувач вищої освіти _____ Роман БЕРЕЗЕНКО
«____» _____ 202_ р.

РЕФЕРАТ

Записка: 65 стр., 73 рис., 1 додаток, 23 джерела.

КОНФІГУРАЦІЯ ПК, АНАЛІЗ ПРОДУКТИВНОСТІ, ВЕБ-РОЗРОБКА, ВИЗНАЧЕННЯ СУМІСНОСТІ, ОЦІНКА ПІДХОДУ

Об’єкт розробки – розробка веб-додатку для конфігурування ПК з оцінкою продуктивності обраної конфігурації.

Мета роботи – розробка веб-додатку з теми “Конфігуратор ПК з аналізом продуктивності”, що дозволяє користувачам створювати конфігурації ПК та отримувати прогнозовану оцінку продуктивності обраних компонентів.

Методи дослідження та інформаційне забезпечення – середовище розробки Visual Studio Code, мова програмування TypeScript, JavaScript бібліотека React, CSS препроцесор SCSS, Node.js, методи розробки інтерфейсів користувача, а також використання баз даних MySQL для зберігання та управління даними про комплектуючі та конфігурації.

Результати дослідження. У межах роботи проаналізовано сучасні методи створення вебсайтів і підходи до розробки конфігураторів ПК, визначено переваги та недоліки існуючих рішень. Розроблено алгоритм оцінки продуктивності комплектуючих для формування рекомендацій користувачам. Створено інтерфейс конфігуратора з перевіркою сумісності компонентів і реалізовано вебзастосунок з функціями підбору, пошуку та збереження конфігурацій. Додано автентифікацію користувачів і персональний кабінет. Забезпечено адаптивність інтерфейсу для різних екранів і оптимізацію веб-додатку.

В результаті тестування виявлено, що система коректно перевіряє сумісність компонентів та надає точну оцінку продуктивності для різних конфігурацій ПК.

Ключові слова: конфігуратор ПК, вебзастосунок, бібліотека React, TypeScript, Node.js, MySQL, адаптивний дизайн.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
ПК	персональний комп’ютер
TDP	thermal design power
FPS	frames per second
UX/UI	user interface / user experience
API	application programming interface
Бенчмарк	набір тестів, який використовується для вимірювання продуктивності
TBW	total bytes written
GPU	graphics processing unit
БЖ	блок живлення
SEO	search engine optimization
JWT	json web token
HTTP	hyper text transfer protocol
JSON	java script object notation
СУБД	система управління базами даних

ВСТУП

Стрімкий розвиток інтернет-технологій суттєво змінив підходи до пошуку інформації та взаємодії з цифровими ресурсами. Веб-застосунки стали невід'ємною частиною повсякденного життя, забезпечуючи швидкий та зручний доступ до різноманітних сервісів. Одним із популярних напрямів є інструменти для підбору комп’ютерних компонентів, що особливо актуально для користувачів, які самостійно комплектують персональні комп’ютери.

Актуальність цієї роботи зумовлена зростаючою потребою користувачів у зручних інструментах для індивідуального підбору комп’ютерних комплектуючих. Із розширенням асортименту апаратного забезпечення та зростанням зацікавленості в самостійному складанні ПК виникає необхідність у сервісах, які автоматизують цей процес. Традиційний підхід вимагає глибоких технічних знань і значних часових витрат, тоді як веб-додаток, що забезпечує перевірку сумісності деталей і оцінку їхньої продуктивності, значно спрощує завдання та мінімізує ризик помилок при виборі конфігурації.

Мета кваліфікаційної роботи – створення інтерактивного веб-додатку, який дозволяє користувачам самостійно формувати конфігурацію персонального комп’ютера, підбираючи сумісні компоненти та отримуючи орієнтовну оцінку їх спільної продуктивності. Такий інструмент має на меті спростити процес складання ПК, зробити його доступним для широкого кола користувачів та зменшити ймовірність технічних помилок при виборі апаратного забезпечення.Об’єкт дослідження — процес розробки веб-застосунку, орієнтованого на підбір та аналіз апаратного забезпечення комп’ютера.

Об’єкт дослідження — веб-застосунок, призначений для конфігурування апаратного забезпечення персонального комп’ютера.

Предмет дослідження — програмні механізми перевірки сумісності компонентів та формування оптимальної конфігурації ПК у межах веб-застосунку.

Новизна роботи — полягає у комплексному підході до вирішення задачі конфігурування ПК, з інтеграцією перевірки сумісності компонентів та попередньої продуктивності системи в єдиному застосунку.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновку, списку використаної літератури та додатків.Обсяг пояснювальної записки становить 105 сторінок, з них основна частина — 64 сторінки, додатки — 41 сторінка, джерела — 23 назви.

1. ПОСТАНОВКА ЗАДАЧІ

У межах кваліфікаційної роботи розроблено веб-додаток для індивідуального конфігурування персонального комп’ютера. Основна мета — підбір сумісних комплектуючих, формування збірки та оцінка її технічного потенціалу. Інструмент орієнтований як на новачків, так і досвідчених користувачів, спрощуючи процес вибору компонентів.

Перед розробкою проаналізовано наявні аналоги та визначено вимоги до системи. Серед ключових вимог:

- адаптивність — коректна робота інтерфейсу на смартфонах, планшетах і ПК;
- зрозуміла структура — логічна навігація з розділами: реєстрація, авторизація, конфігуратор, перегляд збірок;
- перевірка сумісності — автоматичне виявлення несумісних комплектуючих;
- висока швидкодія — швидке завантаження сторінок і обробка запитів;
- кросбраузерність — підтримка всіх популярних браузерів;
- масштабованість — можливість додавання нового функціоналу без змін у базовій архітектурі;
- сучасний дизайн — стильний і зручний інтерфейс за принципами UX/UI.

Відповідно до розробленої структури, додаток включає чотири основні сторінки: головну (яка одночасно виконує функції каталогу комплектуючих та конструктора збірки), сторінки автентифікації, а також особистого профілю.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

Сучасний ринок комп'ютерних комплектуючих вражає своєю різноманітністю та стрімким розвитком технологій. Для користувача, який прагне зібрати власний персональний комп'ютер, вибір оптимальної конфігурації може стати складним завданням. Необхідно враховувати безліч факторів: сумісність компонентів, технічні характеристики, продуктивність у різних сценаріях використання.

У цьому контексті особливої актуальності набувають інструменти, що здатні спростити та оптимізувати процес конфігурації ПК. Одним з таких інструментів є конфігуратор ПК – веб-додаток, який дозволяє користувачам обирати необхідні компоненти, автоматично перевіряти їхню сумісність та отримувати інформацію про потенційну продуктивність зібраної системи.

Існуючі підходи до розробки конфігураторів ПК варіюються від простих веб-інтерфейсів з каталогами комплектуючих до складних систем з інтегрованими рекомендаційними механізмами та інструментами порівняння. Однак, часто такі рішення обмежуються лише переліком технічних характеристик, не надаючи аналізу очікуваної продуктивності в реальних умовах експлуатації.

У зв'язку з цим, метою даної кваліфікаційної роботи є розробка конфігуратора ПК, що не лише забезпечуватиме зручний процес вибору компонентів та перевірку їхньої сумісності, але й надаватиме користувачам комплексний аналіз продуктивності майбутньої системи в різних сценаріях використання. Такий підхід дозволить користувачам приймати більш обґрунтовані рішення при складанні ПК, оптимізувати свої витрати та досягти бажаного рівня продуктивності для конкретних завдань.

2.1. Огляд конфігураторів ПК

Розглянемо найпопулярніші конфігуратори персональних комп’ютерів.

Elmir.ua — онлайн-інструмент для складання персонального комп'ютера з можливістю вибору комплектуючих з їхнього каталогу. Конфігуратор акцентує увагу на зручності вибору та автоматичній перевірці сумісності компонентів.

Інтерфейс конфігуратора Elmir.ua є інтуїтивно зрозумілим, з чітким поділом на категорії компонентів (рис. 2.1).

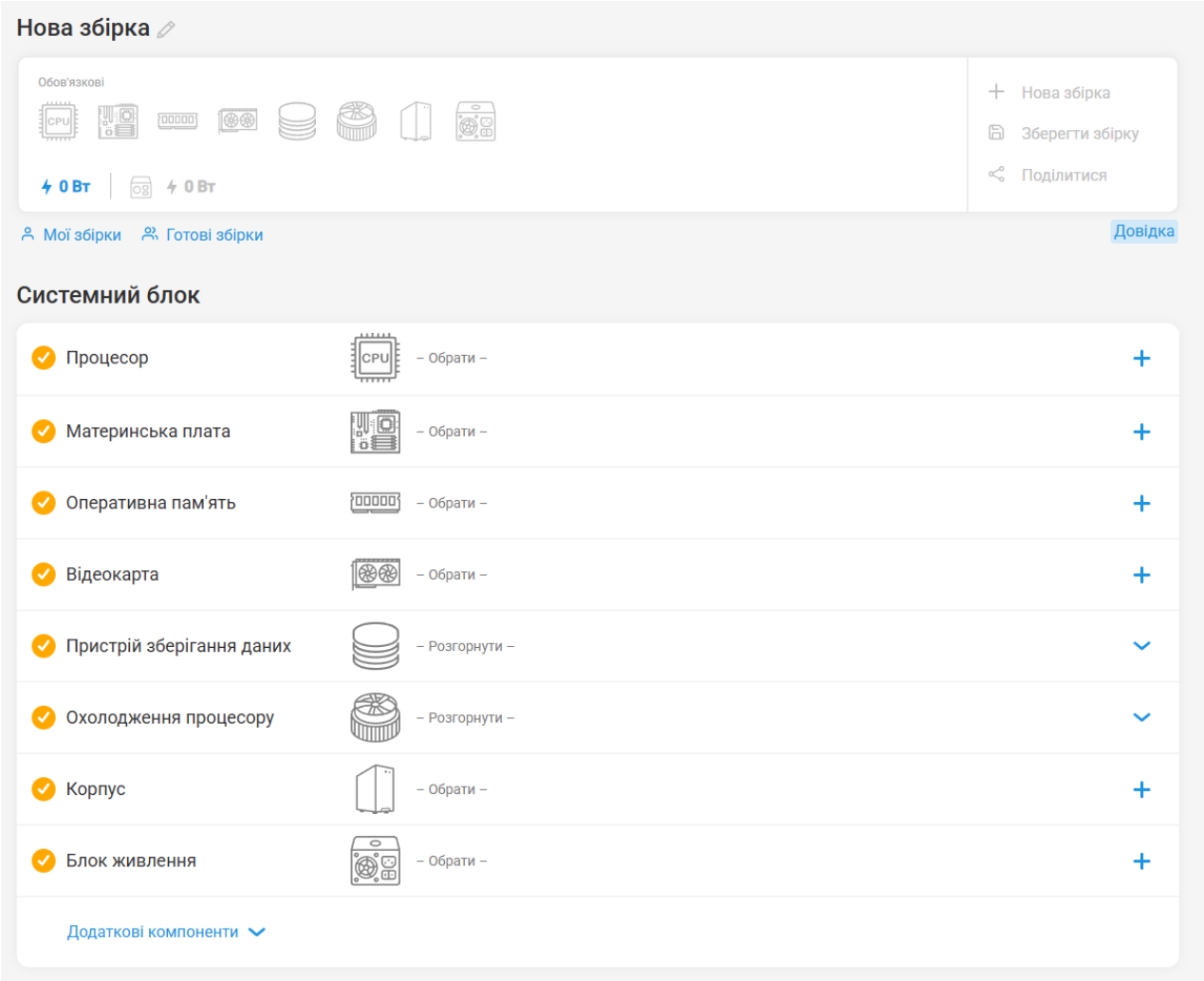


Рисунок 2.1 – Інтерфейс конфігуратора Elmir.ua

Процес вибору є послідовним, а підказки щодо сумісності відображаються в реальному часі (рис. 2.2).

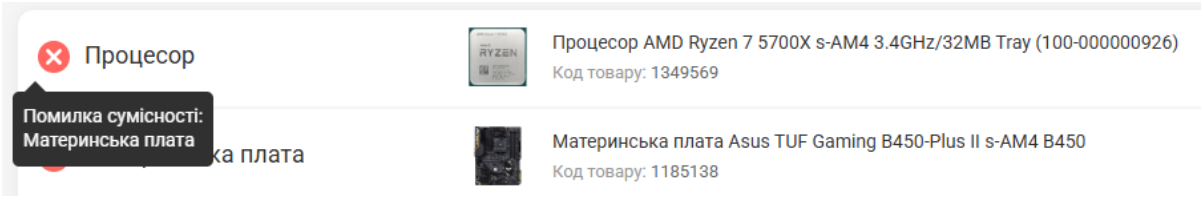


Рисунок 2.2 – Підказки щодо сумісності

Користувачам надається доступ до широкого асортименту комплектуючих з детальними описом та характеристиками (рис. 2.3 – 2.4).

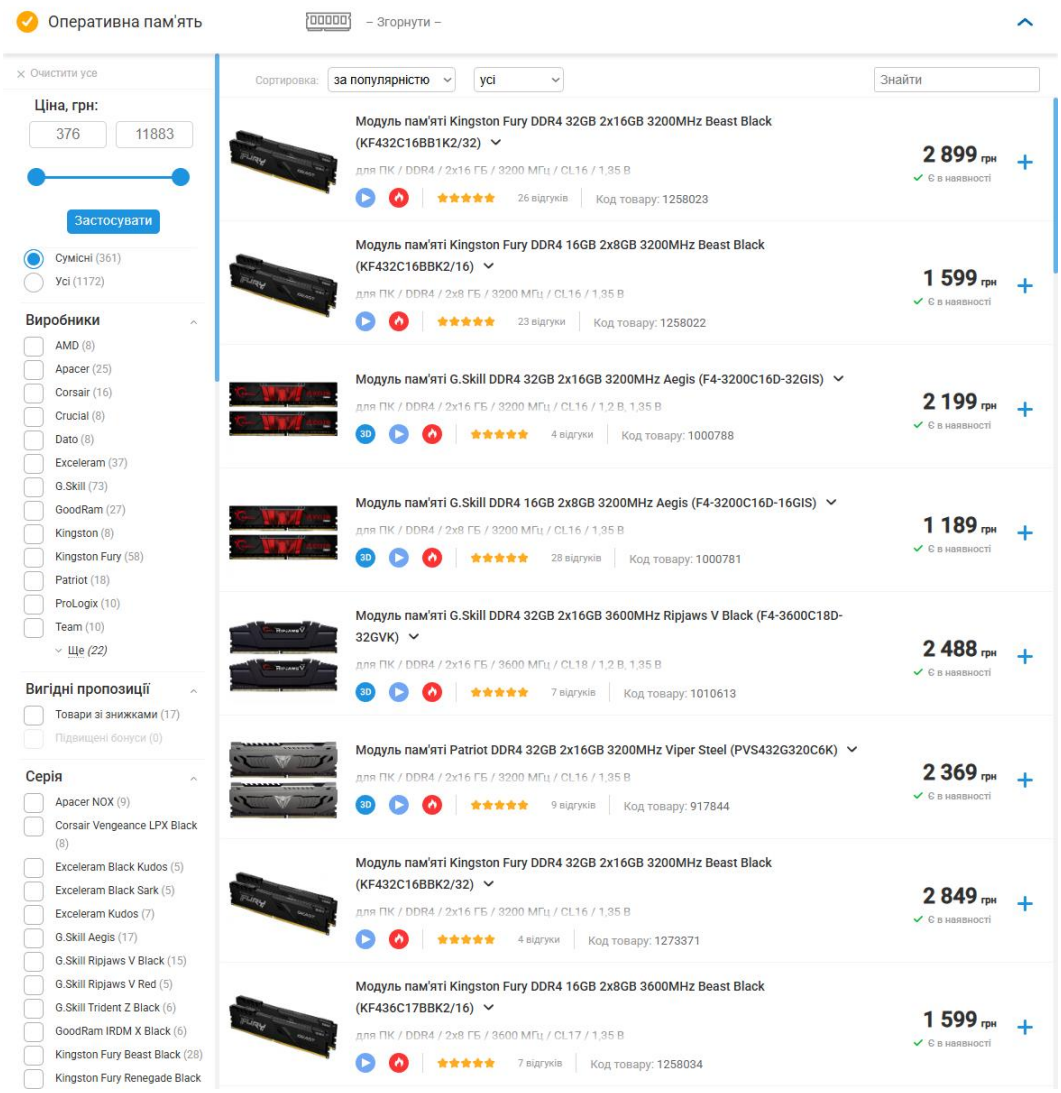


Рисунок 2.3 – Асортимент комплектуючих

Рисунок 2.4 – Характеристики комплектуючого

На сайті представлені готові конфігурації ПК різного призначення та цінових категорій, які можна використовувати як основу або відправну точку для власної збірки (рис. 2.5).

Рисунок 2.5 – Готові конфігурації

На даний момент, відсутні інструменти прогнозування продуктивності зібраної конфігурації у вигляді бенчмарків, орієнтовного FPS в іграх або рекомендацій щодо продуктивності в конкретних завданнях. Інформація обмежується технічними характеристиками компонентів.

Розглянемо наступний конструктор сайтів.

Versum.ua — онлайн-сервіс, орієнтований на створення геймерських комп’ютерів. Конфігуратор на сайті має вузьку спеціалізацію: складання

потужних систем, оптимізованих для ігор і навантажених задач, таких як відеомонтаж, стримінг або робота з графікою (рис. 2.6).

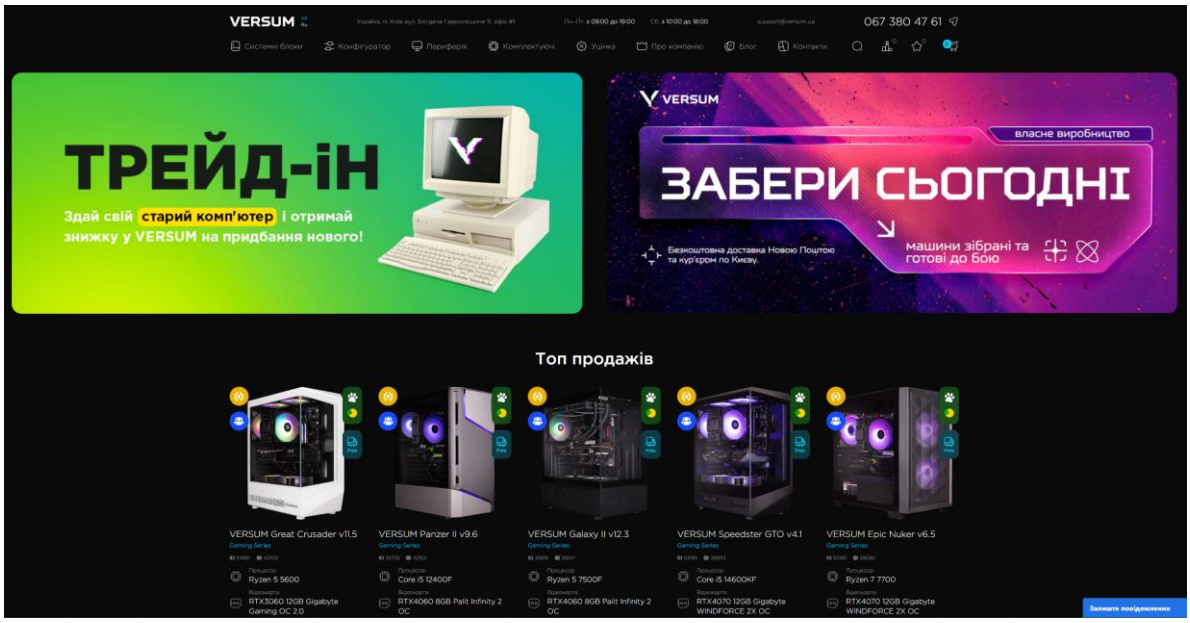


Рисунок 2.6 – Головна сторінка

Конфігуратор має сучасний, адаптивний інтерфейс, орієнтований на швидке обслуговування. Присутня послідовна структура вибору компонентів у вигляді вертикального списку, де кожен блок можна відкривати окремо, що забезпечує лінійний і зрозумілий процес конфігурації (рис. 2.7).

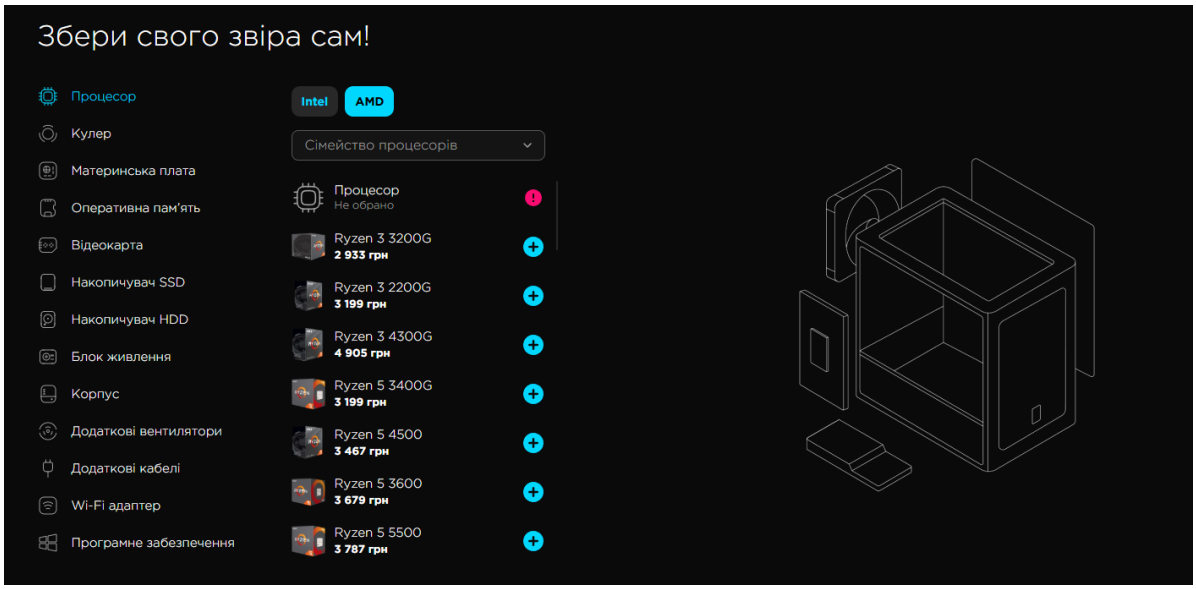


Рисунок 2.7 – Інтерфейс конфігуратора Versum.ua

Під час вибору комплектуючого відображаються його зображення, опис і характеристики, що дозволяє швидко оцінити параметри без переходу на окремі сторінки (рис. 2.8).

Рисунок 2.8 – Структура вибору комплектуючого

Після вибору компонентів система автоматично формує зовнішній вигляд ПК у вибраному корпусі з урахуванням форм-фактора, підсвітки та охолодження (рис. 2.9), що зручно для користувачів, які цінують дизайн.

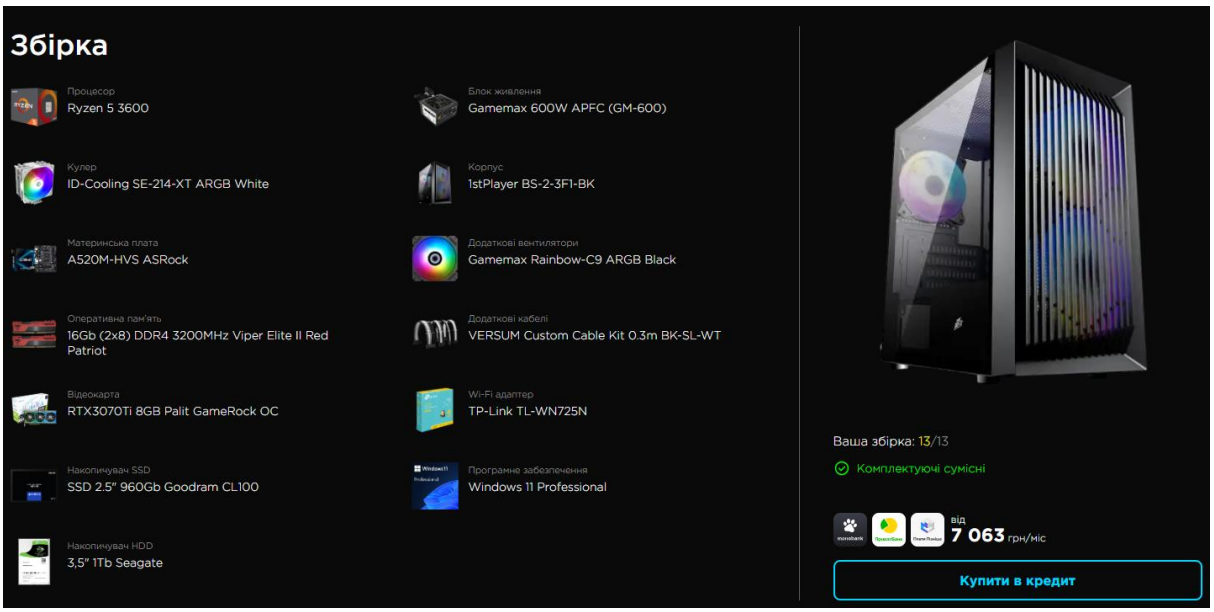


Рисунок 2.9 – Підсумкове зображення зібраної системи в корпусі

2.2. Підсумки аналізу конфігураторів

Переваги Elmir.ua:

- широкий асортимент компонентів — платформа пропонує величезний вибір комплектуючих для складання ПК, що дає користувачам можливість зібрати систему на будь-який смак і бюджет;
- інтуїтивно зрозумілий інтерфейс — розподіл на категорії компонентів дозволяє легко знаходити потрібні елементи і поетапно складати комп’ютер;
- перевірка сумісності в реальному часі — під час вибору компонентів система автоматично перевіряє їх сумісність, що значно зменшує ризик помилок при складанні системи;
- готові конфігурації — для користувачів, які не хочуть витрачати час на самостійний підбір, надає готові конфігурації ПК для різних завдань (геймерські, офісні, для роботи з графікою тощо).

Недоліки Elmir.ua:

- відсутність прогнозу продуктивності — немає інструментів для оцінки FPS, бенчмарків або тестів, що ускладнює розуміння потужності системи для конкретних задач;
- обмежений аналіз сумісності — деякі деталі можуть не враховувати усі нюанси сумісності, як-от обмеження по габаритам або роз’ єми для відеокарт.

Переваги Versum.ua:

- фокус на геймерських і продуктивних збірках — платформа орієнтована на створення потужних ПК для ігор і професійного використання;
- простий і автоматизований підбір — система допомагає швидко сформувати конфігурацію без глибоких технічних знань;
- візуалізація збірки — дає змогу оцінити як технічні характеристики, так і зовнішній вигляд ПК у вибраному корпусі.

Недоліки Versum.ua:

- обмежений вибір компонентів — асортимент переважно складається з популярних топових моделей, що ускладнює збірку з менш поширених або нестандартних комплектуючих;

- відсутність ручного налаштування сумісності — немає можливості вручну встановлювати технічні фільтри (сокет, форм-фактор, TDP), що знижує гнучкість для досвідчених користувачів;

- немає оцінки продуктивності — як і багато аналогів, платформа не пропонує інструментів для прогнозування FPS, бенчмарків чи ефективності в реальних задачах.

Обидва конфігуратори, Elmir.ua і Versum.ua, мають свої переваги та обмеження. Elmir.ua підходить для широкої аудиторії завдяки великому вибору компонентів і зручному інтерфейсу, але йому бракує інструментів для детального прогнозування продуктивності — відсутні бенчмарки, оцінка FPS у іграх та рекомендації для різних сценаріїв використання. Крім того, перевірка сумісності не завжди враховує всі технічні нюанси, як-от обмеження за габаритами чи специфічні вимоги до живлення.

Versum.ua, у свою чергу, орієнтований на потужні геймерські ПК та простоту використання, проте має обмежений асортимент комплектуючих, що ускладнює збірки з нестандартних чи менш популярних деталей. Також відсутні можливості ручного налаштування ключових параметрів сумісності (наприклад, тип сокету, форм-фактор плати чи TDP процесора), що знижує гнучкість для досвідчених користувачів. Як і Elmir.ua, цей сервіс не пропонує жодних засобів для аналізу реальної продуктивності збірки.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Фізична та логічна структура веб-додатку

Фізична структура проекту — це організація файлів і папок на файловій системі, яка визначає, як компоненти додатку розподіляються між різними каталогами для зручності їх управління, доступу та модифікації. Вона є важливою частиною архітектури програмного забезпечення, оскільки забезпечує ефективну структуру для розробників, що працюють над його створенням та супроводом. Добре спроектована фізична структура дозволяє зберігати порядок у коді, підвищує швидкість розробки і полегшує підтримку в майбутньому (рис. 3.1 – 3.2).

Рисунок 3.1 – Фізична структура клієнту

Рисунок 3.2 – Фізична структура серверу

Логічна структура сайту визначає взаємозв’язки між основними сторінками та елементами інтерфейсу, що дозволяє чітко формувати порядок навігації та сценарії користувацької взаємодії. Вона включає чітке розмежування функціональних блоків, що полегшує орієнтування на сайті. Це дозволяє користувачам легко переходити між різними етапами процесу без зайвих зусиль і забезпечує інтуїтивно зрозумілий інтерфейс.

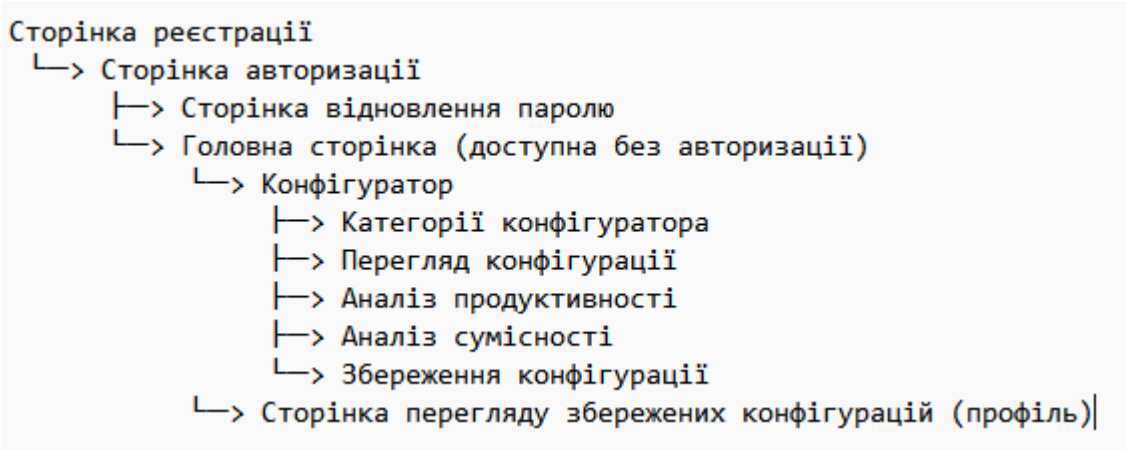


Рисунок 3.3 – Логічна структура сайту

3.2. Обґрунтування вибору середовища розробки

У процесі створення сайту були обрані програмні засоби, які забезпечують ефективну, зручну та швидку розробку вебдодатків. Основними критеріями вибору були: підтримка необхідних мов програмування, зручність

у використанні, наявність вбудованих інструментів та розширень, що полегшують роботу розробника, а також можливості для роботи з базою даних. Саме тому було використано Visual Studio Code для програмування та MySQL Workbench для роботи з базами даних.

Для розробки сайту було використано Visual Studio Code — популярне середовище програмування, створене компанією Microsoft [1]. Воно є безкоштовним, кросплатформенним та підтримує широкий спектр мов програмування. Visual Studio Code має зручний інтерфейс, розширювану архітектуру та велику кількість плагінів, що дозволяють адаптувати середовище під потреби розробника.

Основні переваги редактора:

- підтримка HTML, CSS, JavaScript та інших мов;
- інтелектуальне автодоповнення (IntelliSense);
- вбудований термінал і Git-інтеграція;
- підтримка розширень, зокрема Live Server, Prettier, ESLint.

Ці можливості забезпечують зручну та ефективну роботу над проектом, прискорюють розробку та зменшують кількість помилок у коді.

Для роботи з базою даних використовувалось MySQL Workbench — офіційне графічне середовище для проектування, адміністрування та виконання запитів до MySQL [2]. Дає змогу створювати нові схеми, редагувати структуру таблиць, виконувати SQL-запити та переглядати дані.

MySQL Workbench був обраний для виконання таких завдань:

− створення структури бази даних;

− візуального проектування зв’язків між таблицями;

− написання SQL-запитів для перевірки роботи серверної логіки;

− моніторингу збережених даних після запитів з клієнта.

MySQL Workbench зручний у розробці завдяки візуальному інтерфейсу, валідації схеми в реальному часі та інструментам аналізу запитів, що спрощує пошук помилок і підвищує ефективність роботи.

3.3. Обґрунтування вибору технологій розробки

Для реалізації веб-додатку було обрано набір сучасних технологій, які найбільш оптимально відповідають вимогам проекту. Ці технології забезпечують швидку розробку, зручність масштабування, підтримку компонентного підходу, адаптивного дизайну та ефективного зберігання даних. Їх використання дозволило створити повноцінний застосунок із клієнтською та серверною частинами, що легко підтримується та розширюється.

У проекті були використані такі технології:

− Vite — збирач проекту та локальний сервер;

− React — бібліотека для створення інтерфейсу користувача;

− SCSS — препроцесор для стилів;

− Material UI (MUI) — бібліотека інтерфейсних компонентів;

− Redux Toolkit — інструмент керування станом застосунку;

− Node.js — середовище для серверної частини;

− Express — фреймворк для REST API;

− MySQL — СУБД для зберігання даних.

Для збирання клієнтської частини використовувався Vite [3]. Його було обрано завдяки дуже швидкому старту проекту та миттєвій перезбірці під час розробки — це значно економить час. Крім того, Vite має підтримку сучасного JavaScript і TypeScript з коробки, дозволяє легко інтегрувати SCSS, JSX та інші модулі. Порівняно з Webpack, він значно простіший у налаштуванні та забезпечує кращу продуктивність.

Для ініціалізації проекту достатньо виконати команду npm create vite@latest (рис. 3.4).

```
npm create vite@latest
```

Рисунок 3.4 – Ініціалізація проекту за допомогою Vite

У процесі налаштування слід вказати назву проекту, обрати шаблон (наприклад, react або react-swc), після чого перейти до створеної директорії, встановити залежності та запустити проект за допомогою команди npm run dev (рис. 3.5).

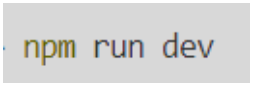


Рисунок 3.5 – Запуск проекту у режимі розробки

Після запуску вебсайт буде доступний у браузері за адресою http://localhost:5173.

Цей підхід забезпечує швидке відображення змін у коді в режимі реального часу без потреби в ручному перезавантаженні сторінки, що значно пришвидшує процес розробки.

React — це JavaScript-бібліотека, що широко використовується для побудови інтерфейсів користувача завдяки своїй компонентній архітектурі, яка дозволяє створювати незалежні та багаторазово використовувані елементи інтерфейсу, як-от кнопки, форми чи блоки списків [4]. У рамках цього проекту було доцільно обрати саме React, оскільки він забезпечує зручний і масштабований підхід до організації коду. Окрім цього, бібліотека підтримує управління станом за допомогою hooks, що спрощує оновлення даних та реакцію на дії користувача.

Завдяки використанню віртуального DOM React [5] забезпечує високу продуктивність — зміни в інтерфейсі відбуваються без повного перезавантаження сторінки, що особливо корисно при створенні односторінкових застосунків (SPA). Ще однією перевагою є JSX — спеціальний синтаксис, який дозволяє комбінувати HTML-подібний розмітковий код з JavaScript-логікою, що значно полегшує процес створення динамічних компонентів (рис. 3.6).

```jsx
import { useState } from "react";

function Counter() {
  // Ініціалізація початкового стану для лічильника
  const [count, setCount] = useState(0);

  // Функції для збільшення та зменшення лічильника
  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);

  return (
    <div>
      <h1>Лічильник: {count}</h1>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
    </div>
  );
}

export default Counter;
```

Рисунок 3.6 – Приклад використання JSX синтаксису

Для зручного інтерфейсу обрали бібліотеку Material UI (MUI) [6], що базується на Google Material Design. Вона містить багато готових компонентів (кнопки, форми, таблиці, меню), що пришвидшує розробку. Також MUI дозволяє легко налаштовувати дизайн через теми, адаптуючи інтерфейс під потреби проекту (рис. 3.7).

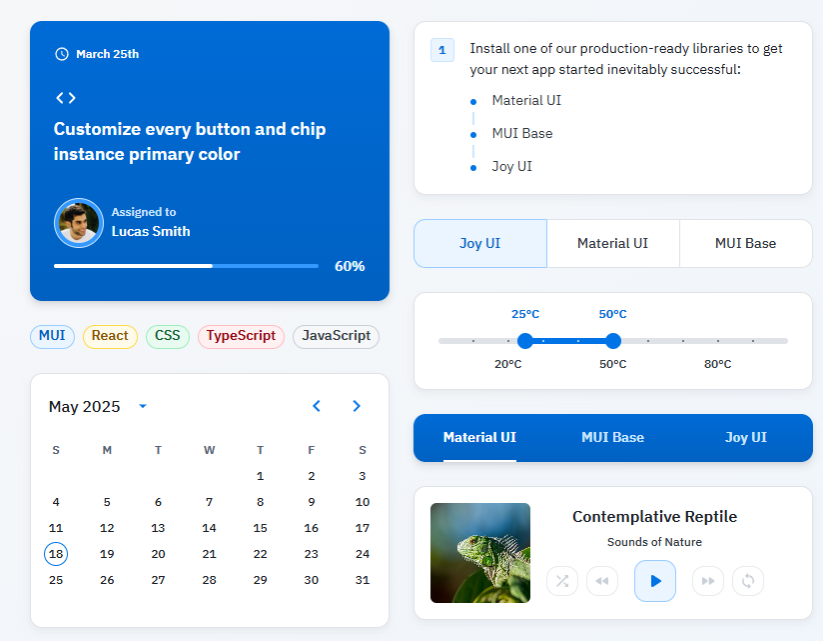


Рисунок 3.7 – Компоненти дизайну MUI

Бібліотека підтримує адаптивність, завдяки чому інтерфейс коректно відображається на різних пристроях і розмірах екрану. Крім того, MUI має велику спільноту користувачів і розробників, що забезпечує наявність численних прикладів, документації та швидку підтримку при виникненні питань. Гнучкість у кастомізації стилів і вбудована підтримка анімацій роблять застосунок більш сучасним і зручним для користувача.

Для стилізації інтерфейсу використано SCSS — розширення мови CSS [7], яке суттєво розширює її можливості та спрощує процес написання стилів (рис. 3.8).

```scss
@import "./normalize.scss";
@import "./variables.scss";
@import "./mixins.scss";

html {
  &::-webkit-scrollbar {
    display: none;
  }
}

body {
  background-color: $bg;
  min-height: 100vh;
  height: 100%;
  font-family: "Inter", sans-serif;
}

.container {
  max-width: calc(1568px + (80px * 2));
  width: 100%;
  margin: 0 auto;
  padding: 0 80px;

  @include Laptop {
    padding: 0 32px;
  }
}
```

Рисунок 3.8 – Приклад використання SCSS синтаксису

Його використання дає змогу зручно організовувати стилі завдяки модульній структурі: стилі можна розділяти на окремі файли, що полегшує підтримку та масштабування коду. SCSS підтримує змінні, функції, міксини та вкладеність, що дозволяє уникати дублювання й швидко вносити зміни до кольорів, шрифтів або розмірів. Такий підхід особливо ефективний при роботі з великими або динамічними інтерфейсами, де важлива гнучкість та повторне використання стилів (рис. 3.9).

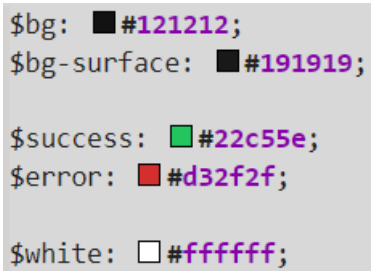


Рисунок 3.9 – Приклад використання змінних в SCSS

Для створення серверу було використано Node.js. Це серверне середовище виконання JavaScript, яке забезпечує створення швидких і масштабованих веб-сервісів [8]. Завдяки асинхронній обробці запитів, Node.js дозволяє ефективно працювати з великою кількістю одночасних з’єднань без блокування. Використання однієї мови програмування JavaScript як на сервері, так і на клієнті значно спрощує процес розробки. Крім того, екосистема npm надає широкий вибір готових пакетів, що значно пришвидшує початок роботи. Node.js гнучко підходить як для створення REST API, так і для реалізації WebSocket-сервісів.

Express — мінімалістичний фреймворк для Node.js [9], який забезпечує швидке створення REST API [10] та ефективну обробку HTTP-запитів (рис. 3.10). Express відомий своєю простотою та гнучкістю, що дозволяє швидко розробляти масштабовані веб-додатки.

```
import express from "express";

const app = express();
const PORT = 3000;

app.use(express.json()); // Для парсингу JSON тіла запиту

// Імітація бази даних
let users = [
  { id: 1, name: "Іван" },
  { id: 2, name: "Артем" },
  { id: 3, name: "Роман" },
];

// GET /users — отримати список усіх користувачів
app.get("/users", (_, res) => {
  res.status(200).json(users);
});

// GET /users/:id — отримати одного користувача за ID
app.get("/users/:id", (req, res) => {
  const user = users.find((u) => u.id === parseInt(req.params.id));
  if (user) res.json(user);
  else res.status(404).json({ error: "Користувача не знайдено" });
});

// Запуск сервера
app.listen(PORT, () => {
  console.log(`Сервер запущено на http://localhost:${PORT}`);
});
```

Рисунок 3.10 – Приклад базового REST API

Використання Express у цьому проекті було обране через його вбудовані інструменти для маршрутизації, обробки запитів, сесій та інтеграції з базами даних, що значно прискорює розробку. Крім того, мінімальна конфігурація та зрозумілий API дозволяють швидко налаштувати сервер без зайвих ускладнень.

MySQL — це реляційна система управління базами даних, яка забезпечує надійне і ефективне зберігання інформації за допомогою підтримки транзакцій, зв’язків між таблицями та складних SQL-запитів [11]. Вибір MySQL обґрунтований її стабільністю, високою продуктивністю при обробці великих обсягів даних, а також широкими можливостями індексації,

що суттєво прискорюють операції пошуку та вибірки інформації. Завдяки великій спільноті та розвиненій екосистемі MySQL залишається одним із найпопулярніших інструментів для роботи з реляційними базами даних.

Redux — це бібліотека для керування глобальним станом JavaScript-застосунків, що часто використовується разом із React. Вона дозволяє централізовано зберігати і змінювати стан, особливо коли дані мають бути доступними для багатьох компонентів [12]. Основними перевагами Redux є централізоване зберігання стану в одному об’єкті store, що спрощує контроль, налагодження і масштабування додатка, а також однонаправлений потік даних, який робить логіку передбачуваною та зручною для тестування. Крім того, Redux підтримує асинхронну логіку через інструменти на кшталт createAsyncThunk, що розширює можливості обробки запитів і оновлення даних.

Вибір Redux обґрунтований його ефективністю у підтримці складного стану застосунку, зручністю організації логіки та широкою підтримкою в екосистемі React.

3.4. Семантична розмітка та доступність

Семантична розмітка є основою доступного вебінтерфейсу. Вона дозволяє структурувати контент так, щоб його могли правильно інтерпретувати всі користувачі, зокрема ті, які використовують допоміжні технології — наприклад, екранні читачі (screen readers). Крім того, правильна семантика позитивно впливає на SEO-оптимізацію та зручність навігації [13].

Семантичні теги — це елементи HTML, які чітко вказують на смислову роль контенту. Їх використання дозволяє:

- покращити розуміння структури сторінки людьми;
- забезпечити коректну роботу допоміжних технологій;
- зробити код більш читабельним і підтримуваним.

До таких тегів належать:

- <header> — верхній блок сторінки або секції, зазвичай містить заголовки й навігацію;
- <footer> — нижня частина сторінки;
- <main> — основний унікальний контент сторінки;
- <section> — смислова частина сторінки, об’єднана тематикою;
- <article> — незалежний, самодостатній блок контенту (напр., публікація, новина);
- <aside> — додатковий контент, що не є основним (наприклад, реклама або блок рекомендацій).

Теги <h1>–<h6> створюють логічну ієрархію заголовків. Рекомендується використовувати один <h1> на сторінку, що описує основну тему, і дотримуватися послідовності (наприклад, після <h1> має йти <h2>, а не <h3>), щоб зберегти структуру документа.

Теги і виділяють текст жирним або курсивом, але також передають важливість або емоційний відтінок. Екранні читачі розпізнають їх і озвучують з відповідною інтонацією.Кожне зображення має містити атрибут alt з описом змісту. Це дозволяє:

- користувачам із вадами зору отримати інформацію про зображення через читач;
- користувачам з повільним інтернетом зрозуміти, що мало б бути завантажено.

ARIA (Accessible Rich Internet Applications) — це набір атрибутів, що підвищують доступність динамічних елементів. Наприклад:

- aria-label — надає текстову мітку елементу;
- aria-hidden="true" — приховує елемент від читачів екрана;
- role="button" або role="navigation" — вказує на функціональну роль елемента.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Завантаження та налаштування необхідних інструментів

Для створення клієнтної частини проекту використано інструмент Vite з шаблоном React та TypeScript. Ось як виглядав процес налаштування:

Спочатку було створено новий проект за допомогою команди npm create vite@latest configurator --template react-ts, що ініціює шаблон React з підтримкою TypeScript (рис. 4.1).

Рисунок 4.1 – Створення проекту за допомогою Vite

Перейшовши до директорії проекту, було виконано команду npm install для інсталяції всіх необхідних залежностей для запуску проекту (рис. 4.2).

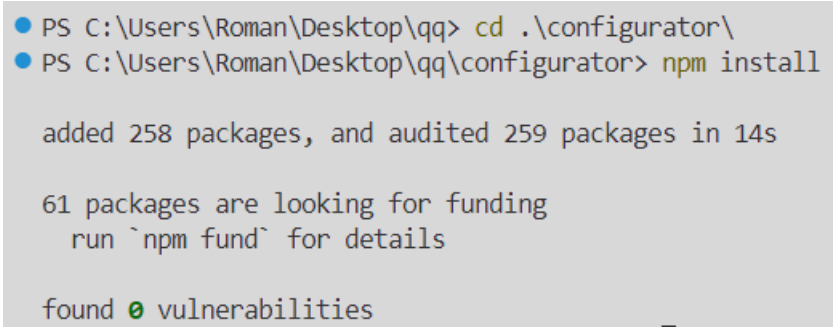


Рисунок 4.2 – Встановлення залежностей

Під час створення проекту автоматично генерується структура файлів, включаючи необхідні залежності для запуску та розробки.

Залежності зберігаються у файлі package.json, який містить всі необхідні бібліотеки та скрипти для подальшої роботи з проектом (рис. 4.3).

```
"dependencies": {
  "@emotion/react": "^11.13.3",
  "@emotion/styled": "^11.13.0",
  "@hookform/resolvers": "^4.1.3",
  "@mui/icons-material": "^6.1.6",
  "@mui/material": "^6.1.6",
  "@reduxjs/toolkit": "^2.3.0",
  "@types/lodash.debounce": "^4.0.9",
  "axios": "^1.7.7",
  "classnames": "^2.5.1",
  "framer-motion": "^11.11.11",
  "lodash.debounce": "^4.0.8",
  "qs": "^6.14.0",
  "react": "^18.3.1",
  "react-dom": "^18.3.1",
  "react-hook-form": "^7.54.2",
  "react-hot-toast": "^2.5.2",
  "react-redux": "^9.1.2",
  "react-router-dom": "^6.26.2",
  "yup": "^1.6.1"
},
```

Рисунок 4.3 – Файл залежностей package.json

Для зручності розробки та розширення функціональності проекту, були встановлені додаткові бібліотеки за допомогою команди npm install axios classnames framer-motion lodash.debounce qs yup react-redux react-hook-form react-hot-toast @reduxjs/toolkit @mui/material @mui/icons-material (рис. 4.4).

```
PS C:\Users\Roman\Desktop\qq\configurator> npm install axios classnames framer-motion lodash
.debounce qs yup react-redux react-hook-form react-hot-toast @reduxjs/toolkit @mui/material
@mui/icons-material

added 63 packages, and audited 322 packages in 19s

74 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 4.4 – Встановлення додаткових бібліотек

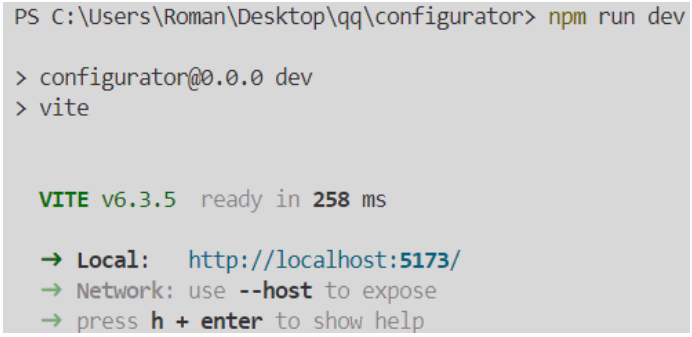
Також було налаштовано файл конфігурації TypeScript — tsconfig.json (рис. 4.5).

```
{
  "compilerOptions": {
    "tsBuildInfoFile": "./node_modules/.tmp/tsconfig.app.tsbuildinfo",
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "verbatimModuleSyntax": true,
    "moduleDetection": "force",
    "noEmit": true,
    "jsx": "react-jsx",

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "erasableSyntaxOnly": true,
    "noFallthroughCasesInSwitch": true,
    "noUncheckedSideEffectImports": true
  },
  "include": ["src"]
}
```

Рисунок 4.5 – Налаштування tsconfig.json

Це дозволило використовувати TypeScript для забезпечення статичної типізації, зменшення кількості помилок під час розробки та покращення зручності при написанні коду [14].

Після завершення інсталяції, проект було запущено на сервері розробки для перевірки коректності налаштувань (рис. 4.6).



```
PS C:\Users\Roman\Desktop\qq\configurator> npm run dev

> configurator@0.0.0 dev
> vite


  VITE v6.3.5  ready in 258 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Рисунок 4.6 – Запуск проекту

Для створення серверної частини проекту була створена окрема директорія за допомогою команди mkdir server. Після створення директорії, було ініціалізовано новий проект в цій директорії за допомогою команди npm init -y [15] (рис 4.7).

```
PS C:\Users\Roman\Desktop\configurator\server> npm init -y
Wrote to C:\Users\Roman\Desktop\configurator\server\package.json:

{
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Рисунок 4.7 – Ініціалізація серверної частини

4.2. Створення бази даних

Для зберігання інформації про комп’ютерні комплектуючі була створена база даних у середовищі MySQL Workbench. На першому етапі було спроектовано структуру таблиць, яка відображає характеристики основних компонентів ПК [16].

Таблиця cpu використовується для зберігання інформації про процесори: назву, сокет, кількість ядер і потоків, базову та максимальну частоти, об’єм кешу L3, архітектуру, тип підтримуваної пам’яті, TDP та інші технічні характеристики (рис. 4.8).

```
CREATE TABLE cpu (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL, image varchar(255) NOT NULL,
  ruler varchar(45) DEFAULT NULL, socket varchar(45) NOT NULL, cores int NOT NULL,
  threads int NOT NULL, frequency int NOT NULL, max_frequency int DEFAULT NULL,
  cache_l3 int DEFAULT NULL, architecture_code_name varchar(100) DEFAULT NULL,
  micro_architecture varchar(45) DEFAULT NULL, graphic_core_name varchar(60) DEFAULT NULL,
  support_pcie varchar(32) DEFAULT NULL, is_unlocked_multiplier tinyint DEFAULT NULL,
  memory_type varchar(6) NOT NULL, memory_frequency int NOT NULL, tdp int NOT NULL,
  performance int NOT NULL, max_memory int NOT NULL, PRIMARY KEY (id)
);
```

Рисунок 4.8 – Структура таблиці для процесорів

Таблиця motherboard зберігає дані про материнські плати: назву, сокет, чипсет, тип і кількість пам’яті, підтримуваний обсяг і частоту, форм-фактор, інтерфейси (SATA, USB, PCIe), мережу та роз’єми M.2 (рис. 4.9).

```
CREATE TABLE motherboard (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
  image varchar(255) NOT NULL, socket varchar(32) NOT NULL, chipset varchar(32) NOT NULL,
  memory_type varchar(16) NOT NULL, count_memory_slots int NOT NULL,
  count_memory_channels int NOT NULL, max_memory int NOT NULL,
  min_frequency_memory int NOT NULL, max_frequency_memory int NOT NULL,
  lan varchar(96) DEFAULT NULL, count_sata int DEFAULT NULL, support_pcie16x int DEFAULT NULL,
  count_usb_v2 int DEFAULT NULL, count_usb_v3 int DEFAULT NULL, count_m2_slots int DEFAULT NULL,
  support_pcie_m2_v3 int DEFAULT NULL, form_factor varchar(10) NOT NULL, PRIMARY KEY (id)
);
```

Рисунок 4.9 – Структура таблиці для материнських плат

Таблиця gpu містить дані про відеокарти: назву, зображення, характеристики пам’яті, серію, частоти, роздільну здатність, продуктивність, інтерфейс, підтримку стандартів, розміри, TDP, охолодження та роз’єми (рис. 4.10).

```
CREATE TABLE gpu (
  id INT NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
  image varchar(255) NOT NULL, memory varchar(50) NOT NULL,
  bus varchar(50) DEFAULT NULL, type_memory varchar(100) DEFAULT NULL,
  series varchar(100) DEFAULT NULL, memory_frequency varchar(50) NOT NULL,
  video_frequency varchar(50) DEFAULT NULL, max_resolution varchar(50) DEFAULT NULL,
  performance int NOT NULL, interface varchar(50) DEFAULT NULL,
  count_fans int DEFAULT NULL, support_standard varchar(100) DEFAULT NULL,
  support_cuda tinyint(1) DEFAULT NULL, cuda_cores int DEFAULT NULL,
  length int DEFAULT NULL, height int DEFAULT NULL, tdp int NOT NULL,
  additional_connector varchar(100) DEFAULT NULL, support_screens int DEFAULT NULL,
  PRIMARY KEY (id)
);
```

Рисунок 4.10 – Структура таблиці для відеокарт

Таблиця ram містить дані про оперативну пам’ять: назву, зображення, тип, обсяг, кількість модулів, частоту, пропускну здатність, таймінги та напругу (рис. 4.11).

```
CREATE TABLE ram (
  id INT NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
  image varchar(255) NOT NULL, memory_type varchar(50) NOT NULL,
  memory_capacity int NOT NULL, count_modules int NOT NULL,
  frequency int NOT NULL, passing_capacity int DEFAULT NULL,
  latency varchar(50) DEFAULT NULL, timings varchar(50) DEFAULT NULL,
  voltage varchar(50) DEFAULT NULL, PRIMARY KEY (id)
)
```

Рисунок 4.11 – Структура таблиці для оперативної пам’яті

Таблиця psu зберігає дані про блоки живлення: назву, зображення, форм-фактор, потужність і сертифікацію, що допомагає обрати відповідне джерело для системи (рис. 4.12).

```
CREATE TABLE psu (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
  image varchar(255) NOT NULL, form_factor varchar(255) NOT NULL,
  power int NOT NULL, certificate varchar(255) DEFAULT NULL,
  PRIMARY KEY (id)
)
```

Рисунок 4.12 – Структура таблиці для блоків живлення

Таблиця air_cooling містить дані про повітряні кулери CPU: модель, сумісність із сокетами, габарити, TDP, вентилятор, теплові трубки, шум, швидкість, підключення та тип охолодження (рис. 4.13).

```
CREATE TABLE air_cooling (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
  image varchar(255) NOT NULL, socket varchar(255) NOT NULL,
  diameter int DEFAULT NULL, type varchar(255) DEFAULT NULL,
  connector varchar(255) DEFAULT NULL, speed varchar(255) DEFAULT NULL,
  max_tdp int NOT NULL, noise_level int DEFAULT NULL,
  count_heat_pipes int DEFAULT NULL, height int DEFAULT NULL,
  sizes varchar(255) DEFAULT NULL, PRIMARY KEY (id)
)
```

Рисунок 4.13 – Структура таблиці для повітряного охолодження

Таблиця water_cooling містить дані про водяні системи охолодження: модель, тип, сумісність із сокетами, вентилятори, шум, підключення, швидкість помпи та габарити (рис. 4.14).

```
CREATE TABLE water_cooling (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
  image varchar(255) NOT NULL, type varchar(255) DEFAULT NULL,
  socket varchar(255) NOT NULL, diameter int DEFAULT NULL,
  count_fans int DEFAULT NULL, speed varchar(255) DEFAULT NULL,
  noise_level int DEFAULT NULL, connector varchar(255) DEFAULT NULL,
  pump_speed varchar(255) DEFAULT NULL, sizes varchar(255) DEFAULT NULL,
  PRIMARY KEY (id)
)
```

Рисунок 4.14 – Структура таблиці для водяного охолодження

Таблиця hdd містить інформацію про жорсткі диски: модель, форм-фактор, обсяг пам’яті, інтерфейс підключення, швидкість передачі даних, обертання шпинделя та рівень шуму під час роботи (рис. 4.15).

```
CREATE TABLE hdd (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) DEFAULT NULL,
  image varchar(255) DEFAULT NULL, form_factor varchar(255) DEFAULT NULL,
  memory varchar(255) DEFAULT NULL, interface varchar(255) DEFAULT NULL,
  speed int DEFAULT NULL, rotating_speed int DEFAULT NULL,
  noise_level int DEFAULT NULL, PRIMARY KEY (id)
)
```

Рисунок 4.15 – Структура таблиці для HDD

Таблиця ssd зберігає дані про твердотільні накопичувачі: модель, форм-фактор, обсяг і тип пам’яті, інтерфейс, підтримку NVMe, швидкість читання/запису та показник TBW (рис. 4.16).

```
CREATE TABLE ssd (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
  image varchar(255) NOT NULL, form_factor varchar(255) DEFAULT NULL,
  memory varchar(255) DEFAULT NULL, type_memory varchar(255) DEFAULT NULL,
  interface varchar(255) DEFAULT NULL, is_nvme tinyint(1) DEFAULT NULL,
  read_speed int DEFAULT NULL, write_speed int DEFAULT NULL,
  tbw int DEFAULT NULL, PRIMARY KEY (id)
)
```

Рисунок 4.16 – Структура таблиці для SSD

Таблиця case зберігає дані про комп'ютерні корпуси: назву, тип, сумісність з форм-фактором плати, підтримку рідинного охолодження, висоту кулера, кількість відсіків під HDD, довжину відеокарти та габарити (рис. 4.17).

```
CREATE TABLE case (
  id int NOT NULL AUTO_INCREMENT, name varchar(255) DEFAULT NULL,
  image varchar(255) DEFAULT NULL, type varchar(255) DEFAULT NULL,
  form_factor_motherboard varchar(255) DEFAULT NULL,
  is_support_water_cooling int DEFAULT NULL,
  max_height_air_cooling int DEFAULT NULL,
  hdd_slots varchar(255) DEFAULT NULL, max_length_gpu int DEFAULT NULL,
  sizes varchar(255) DEFAULT NULL, PRIMARY KEY (id)
)
```

Рисунок 4.17 – Структура таблиці для комп'ютерних корпусів

Таблиця users використовується для зберігання облікових записів користувачів. Вона містить унікальний ідентифікатор, адресу електронної пошти, яка також є унікальною, та хешований пароль (рис. 4.18).

```
CREATE TABLE users (
  id int NOT NULL AUTO_INCREMENT, email varchar(60) NOT NULL,
  password_hash varchar(255) NOT NULL, PRIMARY KEY ("id"),
  UNIQUE KEY "email_UNIQUE" ("email")
)
```

Рисунок 4.18 – Структура таблиці для облікових записів

Таблиця reset_password_tokens зберігає ID, токен, user_id і дату завершення дії для безпечного скидання пароля (рис. 4.19).

```
CREATE TABLE reset_password_tokens (
  id int NOT NULL AUTO_INCREMENT, user_id int NOT NULL,
  token varchar(255) NOT NULL, expires datetime NOT NULL,
  PRIMARY KEY (id), UNIQUE KEY "id_UNIQUE" ("id")
)
```

Рисунок 4.19 – Структура таблиці для токенів відновлення

Таблиця user_configurations містить інформацію про зібрані користувачами конфігурації ПК: вибрані комплектуючі (CPU, материнську

плату, ОЗП, GPU, БЖ, охолодження, SSD, HDD та корпус). Записи пов’язані з користувачами через user_id, що дозволяє зберігати конфігурації для повторного використання (рис. 4.20).

```
CREATE TABLE user_configurations (
  id int NOT NULL AUTO_INCREMENT, user_id int NOT NULL,
  cpu_id int NOT NULL, motherboard_id int NOT NULL,
  ram_id int NOT NULL, ram_count int NOT NULL,
  gpu_id int NOT NULL, psu_id int NOT NULL,
  air_cooling_id int DEFAULT NULL, water_cooling_id int DEFAULT NULL,
  ssd_id int DEFAULT NULL, ssd_count int DEFAULT NULL,
  hdd_id int DEFAULT NULL, hdd_count int DEFAULT NULL,
  case_id int NOT NULL, PRIMARY KEY (id)
)
```

Рисунок 4.20 – Структура таблиці для конфігурацій користувачів

4.3. Розробка серверної частини

Серверна частина застосунку реалізована у вигляді окремого Node.js-проекту в директорії server. Для організації коду було обрано модульну структуру (рис. 4.21), яка дозволяє чітко розділити логіку обробки запитів, маршрутизації, роботи з базою даних та допоміжними функціями. Це значно спростило масштабування та підтримку проекту.

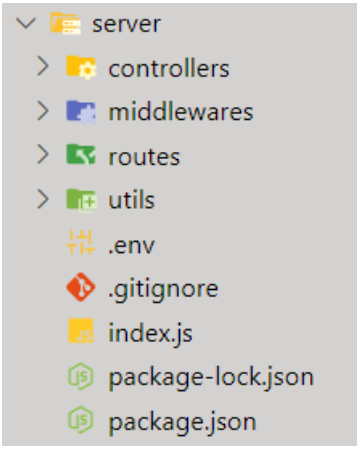


Рисунок 4.21 – Файлова структура серверу

Основні елементи файлової структури:

− index.js — головний файл, де ініціалізується сервер, підключаються middleware та маршрути, запускається прослуховування порту;

− controllers/ — директорія з логікою обробки запитів, поділених за функціоналом: користувачі, автентифікація, компоненти ПК.routes/ — окрема директорія для визначення маршрутів API, де кожен маршрут передає запит відповідному контролеру;

- middlewares/ — тут зберігаються проміжні функції, наприклад, перевірка автентифікації користувача через JWT;

- utils/ — допоміжні модулі, зокрема для підключення до бази даних;

- .env — файл конфігурацій змінних середовища.

Основна логіка запуску серверної частини реалізована у файлі index.js. У цьому файлі виконується налаштування основних middleware, підключення маршрутів та запуск HTTP-сервера (рис. 4.22).

```
import express from "express";
import cors from "cors";
import cookieParser from "cookie-parser";

//routes
import componentsRouter from "./routes/components.js";
import configurationsRouter from "./routes/configurations.js";
import authRouter from "./routes/auth.js";
import userRouter from "./routes/user.js";

const app = express();
const PORT = process.env.PORT || 3000;
const prefix = "/api/v1";

app.use(
  cors({
    origin: "http://localhost:5173",
    credentials: true,
  })
);
app.use(express.json());
app.use(cookieParser());

app.use(`${prefix}/components`, componentsRouter);
app.use(`${prefix}/auth`, authRouter);
app.use(`${prefix}/user`, userRouter);
app.use(`${prefix}/configurations`, configurationsRouter);

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Рисунок 4.22 – Файл index.js

На початку відбувається імпорт необхідних бібліотек:

- express — фреймворк для побудови веб-серверу;
- cors — для налаштування крос-доменного доступу, що дозволяє клієнтській частині взаємодіяти з API;
- cookie-parser — для роботи з cookie.

Після цього створюється екземпляр застосунку Express та визначається порт прослуховування. Для зручності маршрути API мають префікс /api/v1, що дозволяє легко версіонувати API в майбутньому.

Далі налаштовуються middleware:

- cors() — дозволяє клієнту, надсилати запити до API з урахуванням передачі cookie;
- express.json() — автоматично обробляє вхідні запити з JSON-тілом;
- cookieParser() — дозволяє зчитувати cookie з запитів.

Після цього відбувається підключення чотирьох основних груп маршрутів:

- /components — робота з комплектуючими ПК;
- /auth — маршрути для реєстрації та авторизації користувачів;
- /user — маршрути, пов’язані з даними користувача;
- /configurations — маршрути для створення та збереження конфігурацій ПК.

Для забезпечення зберігання та обробки даних у серверній частині було реалізовано підключення до бази даних MySQL. Для цього використано бібліотеку mysql2/promise, яка забезпечує асинхронну роботу з базою даних, що є необхідним для продуктивної взаємодії з API.

У файлі підключення db.js (рис. 4.23) спочатку завантажуються змінні середовища з .env файлу за допомогою бібліотеки dotenv. Це дозволяє зберігати конфіденційні дані, такі як логін, пароль, хост та назву бази даних, поза межами відкритого коду.

```
const { DB_HOST, DB_NAME, DB_PORT, DB_LOGIN, DB_PASS } = process.env;

const pool = mysql.createPool({
  host: DB_HOST,
  user: DB_LOGIN,
  password: DB_PASS,
  database: DB_NAME,
  port: DB_PORT,
  connectTimeout: 60000,
});

export const query = async (sql, params) => {
  const connection = await pool.getConnection();
  try {
    const [results] = await connection.execute(sql, params);
    return results;
  } finally {
    connection.release();
  }
};
```

Рисунок 4.23 – Підключення до бази даних

Створюється пул з'єднань через mysql.createPool() для зменшення навантаження на сервер і ефективнішого керування підключеннями до бази даних. Це дозволяє уникнути постійного відкриття й закриття з'єднань при кожному запиті. Функція query відкриває з'єднання з пулу, виконує SQL-запит і повертає результат, забезпечуючи централізовану, безпечну та уніфіковану обробку запитів у проекті.

Для реалізації реєстрації користувачів було створено функцію register (рис. 4.24). Вона перевіряє, чи існує в базі користувач із вказаною електронною поштою, і якщо ні — хешує пароль за допомогою бібліотеки bcrypt та зберігає нового користувача до таблиці users. Також ця функція може бути доповнена логікою для перевірки введених даних і обробки можливих помилок при збереженні.

```
export const register = async (req, res) => {
  const { email, password } = req.body;
  const isExist = await query("SELECT * FROM users WHERE email = ?", [email]);
  if (isExist.length) {
    return res.status(400).json({ message: "Користувач з таким email вже існує" });
  }
  const hashedPassword = await bcrypt.hash(password, 10);
  await query("INSERT INTO users (email, password_hash) VALUES (?, ?)", [email, hashedPassword]);
  await res.status(200).json({ message: "Успішна реєстрація" });
};
```

Рисунок 4.24 – Функція реєстрації

Для авторизації користувача використовується функція login (рис. 4.25). Вона перевіряє, чи існує користувач із вказаною електронною поштою у базі даних. Якщо такий користувач знайдений, функція порівнює введений пароль з хешованим паролем, збереженим у базі, використовуючи метод bcrypt.compare.

```
export const login = async (req, res) => {
  const { email, password } = req.body;

  const [user] = await query("SELECT id, password_hash FROM users WHERE email = ?", [email]);

  if (!user || !bcrypt.compareSync(password, user.password_hash)) {
    return res.status(403).json({ message: "Не вірні дані" });
  }

  const accessToken = jwt.sign({ email, userId: user.id }, JWT_ACCESS_SECRET, { expiresIn: "1h" });
  const refreshToken = jwt.sign({ email, userId: user.id }, JWT_REFRESH_SECRET, {
    expiresIn: "14d",
  });

  res.cookie("accessToken", accessToken, { httpOnly: true, secure: false, sameSite: "Strict" });
  res.cookie("refreshToken", refreshToken, { httpOnly: true, secure: false, sameSite: "Strict" });

  res.status(200).json({ message: "Вхід виконано успішно" });
};
```

Рисунок 4.25 – Функція авторизації

У разі успішної перевірки генерується JWT-токен доступу, який містить основну інформацію про користувача (наприклад, ID або email).

Цей токен підписується секретним ключем та надсилається клієнту у вигляді HTTP-only cookie, що забезпечує захист від XSS-атак. Надалі токен використовується для ідентифікації користувача під час виконання запитів до захищених маршрутів. Такий підхід дозволяє реалізувати безпечну систему автентифікації, уникаючи зберігання сесій на сервері.

Функція getComponents отримує список комплектуючих з бази даних, враховуючи тип, фільтрацію за пошуком і пагінацію. Вона перевіряє, чи передано правильний тип, здійснює пошук за назвою, враховує сторінки та кількість елементів на сторінці, а також підраховує загальну кількість елементів для пагінації (рис. 4.26).

```
import { query } from "../utils/db.js";

export const getComponents = async (req, res) => {
  const { type, page = 1, limit = 10, search } = req.query;
  const offset = (page - 1) * limit;

  const allowedTables = [
    "motherboard",
    "cpu",
    "gpu",
    "ram",
    "hdd",
    "ssd",
    "case",
    "psu",
    "air_cooling",
    "water_cooling",
  ];

  if (!allowedTables.includes(type)) {
    return res.status(400).json({ message: "Invalid type parameter" });
  }

  const tableName = type === "case" ? "`case`" : type;

  try {
    let queryString = `SELECT * FROM ${tableName}`;
    let queryParams = [];

    if (search) {
      queryString += ` WHERE LOWER(name) LIKE LOWER(?)`;
      queryParams.push(`%${search}%`);
    }

    queryString += ` LIMIT ${parseInt(limit, 10)} OFFSET ${parseInt(offset, 10)}`;

    const data = await query(queryString, queryParams);

    let countQuery = `SELECT COUNT(*) as count FROM ${tableName}`;
    let countParams = [];

    if (search) {
      countQuery += ` WHERE LOWER(name) LIKE LOWER(?)`;
      countParams.push(`%${search}%`);
    }

    const [totalCount] = await query(countQuery, countParams);
    const totalPages = Math.ceil(totalCount.count / limit);

    return res.status(200).json({ data, totalPages });
  } catch (error) {
    console.error(`Error fetching data from ${type}:`, error);
    return res.status(500).json({ message: "Internal server error" });
  }
};
```

Рисунок 4.26 – Функція отримання комплектуючих

Для запитів до API використовуються маршрути, побудовані за допомогою роутера Express [18], що дозволяє обробляти запити, пов’язані з авторизацією, скиданням пароля та перевіркою токенів (рис. 4.27). Роутер структурує логіку за функціональністю, а кожен маршрут пов’язаний з контролером, який перевіряє дані, працює з базою або генерує токени. Для захищених маршрутів реалізована перевірка JWT-токена перед обробкою запиту.

```
import express from "express";

// controllers
import * as UserController from "../controllers/UserController.js";

// middlewares
import { authMiddleware } from "../middlewares/auth.js";

const router = express.Router();

router.get("/get", authMiddleware, UserController.getUser);
router.post("/send-reset-password", UserController.sendResetPasswordLetter);
router.post("/reset-password", UserController.resetPassword);
router.post("/check-reset-token", UserController.checkResetToken);

export default router;
```

Рисунок 4.27 – API маршрути, що стосуються користувача

4.4. Розробка клієнтної частини

Клієнтну частину веб-застосунку реалізовано з використанням сучасного стеку технологій: React, TypeScript, Redux, SCSS та бібліотеки Material UI (MUI). Для ініціалізації проекту використано Vite, що забезпечило високу швидкість запуску локального середовища та швидке оновлення під час розробки. Уся логіка додатка зосереджена у папці src, яка містить підрозділи для компонентів, сторінок, стилів, API-сервісів, утиліт, типів та конфігурацій. Це дозволяє підтримувати чисту архітектуру з розділенням відповідальностей (рис. 4.28).

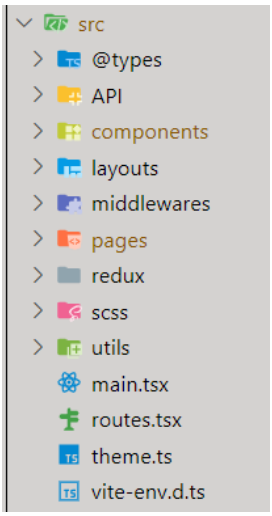


Рисунок 4.28 – Структура папки src

Початкова точка застосунку розміщується у файлі main.tsx (рис. 4.29), де підключаються глобальні стилі, провайдер глобального стану (Redux Provider), а також обгортка для маршрутизації (BrowserRouter), яка забезпечує навігацію між сторінками без перезавантаження сторінки. У цьому файлі також підключено тему Material UI та базові стилі SCSS.

```tsx
import { createRoot } from "react-dom/client";
import { RouterProvider } from "react-router-dom";
import { ThemeProvider } from "@mui/material";
import { Provider } from "react-redux";
import { Toaster } from "react-hot-toast";

//styles
import "@/scss/main.scss";

//routes
import router from "./routes.tsx";

//theme
import { theme } from "./theme.ts";

//store
import { store } from "./redux/store.ts";

createRoot(document.getElementById("root")!).render(
  <ThemeProvider theme={theme}>
    <Provider store={store}>
      <RouterProvider router={router} />
      <Toaster
        position="bottom-center"
        reverseOrder={false}
        toastOptions={{ style: { borderRadius: 10, background: "#333", color: "#fff" } }}
      />
    </Provider>
  </ThemeProvider>
);
```

Рисунок 4.29 – Підключення провайдерів та теми

Для реалізації маршрутизації було створено файл routes.tsx, у якому описано логіку переходів між сторінками. За допомогою бібліотеки react-router-dom [19] визначені основні шляхи, такі як /, /login, /register, /profile (рис. 4.30).

```
const router = createBrowserRouter([
  { path: "/", element: <ConfiguratorPage /> },
  { path: "/register", element: <RegisterPage /> },
  { path: "/login", element: <LoginPage /> },
  { path: "/profile", element: <ProfilePage /> },
  { path: "/reset-password", element: <ResetPasswordPage /> },
]);
```

Рисунок 4.30 – Реалізація маршрутизації

Усі маршрути пов’язані з окремими сторінками, які зберігаються у папці pages (рис. 4.31).

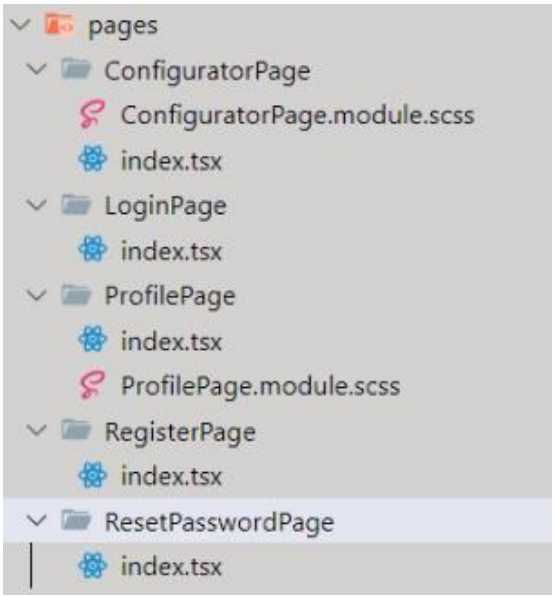


Рисунок 4.31 – Структура папки pages

Кожен файл у цій папці відповідає окремому маршруту, що забезпечує зручну організацію структури проекту. Це дозволяє легко керувати навігацією між сторінками, а також пришвидшує розробку та обслуговування вебзастосунку.

Основний функціонал додатку — це конфігуратор ПК, який дає змогу користувачеві зібрати індивідуальну комп'ютерну систему відповідно до власних потреб. Його реалізовано через набір спеціалізованих компонентів, серед яких основним є Configurator (рис. 4.32).

```
<div className={styles.configurator}>
  <h2>Комплектуючі</h2>
  <div className={styles.configurator__list}>
    {componentTypes.map(({ type, name }) => {
      const isDisabled = !!(
        (type === ComponentTypes.AIR_COOLING &&
          selectedComponents[ComponentTypes.WATER_COOLING]) ||
        (type === ComponentTypes.WATER_COOLING &&
          selectedComponents[ComponentTypes.AIR_COOLING])
      );

      return (
        <ConfiguratorItem
          key={type}
          disabled={isDisabled}
          searchValue={searchValues[type]}
          onSearchChange={handleSearchChange(type)}
          name={name}
          type={type}
          data={components[type].data}
          totalPages={components[type].totalPages}
          onPageChange={(newPage) => handlePageChange(type, newPage)}
          selected={selectedComponents[type]}
          onSelect={(component) => dispatch(selectComponent({ type, component }))}
          isOpen={openedItems[type]}
          onToggle={() => handleToggleItem(type)}
        />
      );
    })}
  </div>
</div>
```

Рисунок 4.32 – Компонент Configurator

Configurator відповідає за загальну структуру та відображення конфігуратора, забезпечуючи послідовний вибір складових комп'ютера.

Ще одним важливим компонентом є ConfiguratorItem, який відповідає за відображення та вибір окремих комплектуючих, таких як процесор, оперативна пам'ять, накопичувачі тощо (рис. 4.33).

```
<div className={cn(styles.item, disabled ? styles.disabled : "")}>
  <div className={styles.item__wrapper}>
    <div className={styles.item__head} onClick={() => !disabled && onToggle()}>
      <p className={styles.item__head__title}>{name}</p>
      <div className={styles.item__head__selected}>
        {selected && (
          <>
            <img src={selected?.image} />
            <p>{selected?.name}</p>
            {countingTypes.includes(type) && (
              <Counter
                count={selectedComponents[type]?.count || 1}
                setCount={(newCount: number) =>
                  dispatch(updateCount({ type, count: newCount }))
                }
              />
            )}
          </>
        )}
      </div>
      <Button
        className={styles.item__head__close}
        onClick={(e) => {
          e.stopPropagation();
          if (selected) {
            onSelect(null);
          } else {
            onToggle();
          }
        }}
        variant="iconary"
        variantColor={"default"}
      >
        {selected ? (
          <Close />
        ) : (
          <ArrowDropDown
            className={cn(styles.item__head__close__arrow, { [styles.open]: isOpen })}
          />
        )}
      </Button>
    </div>
    <motion.div
      className={styles.item__content}
      initial="hidden"
      variants={variants}
      animate={isOpen ? "visible" : "hidden"}
      style={{ overflow: "hidden" }}
    >
      <div className={styles.item__content__head}>
        <SearchBar value={searchValue} onChange={onSearchChange} />
      </div>
      <ComponentsList
        data={data}
        totalPages={totalPages}
        onPageChange={onPageChange}
        selected={selected?.id}
        onSelect={(item) => {
          onSelect(item);
          onToggle();
        }}
      />
    </motion.div>
  </div>
</div>
```

Рисунок 4.33 – Компонент ConfiguratorItem

Для зручності взаємодії з користувачем також реалізовано модальні вікна — AnalyzeCompatibilityModal і AnalyzePerformanceModal. Вони відображаються у разі запиту аналізу конфігурації на сумісність або продуктивність, використовуючи утиліти з utils/ (рис. 4.34).

```
const analysisPoints = [
  {
    name: "Сумісність процесора та материнської плати",
    checkFunction: () => isCompatibleCPUAndMotherboard(cpu, motherboard),
  },
  {
    name: "Сумісність ОЗП та процесора",
    checkFunction: () => isCompatibleRAMAndCPU(ram, cpu),
  },
  {
    name: "Сумісність ОЗП та материнської плати",
    checkFunction: () => isCompatibleRAMAndMotherboard(ram, motherboard),
  },
  {
    name: "Сумісність блоку живлення",
    checkFunction: () => isCompatiblePowerSupplyAndGPUAndCPU(psu, gpu, cpu),
  },
  ...(air_cooling?.id
    ? [
        {
          name: "Сумісність повітряного охолодження та процесора",
          checkFunction: () => isCompatibleAirCoolingAndCPU(air_cooling, cpu),
        },
      ]
    : []),
  ...(water_cooling?.id
    ? [
        {
          name: "Сумісність водяного охолодження та процесора",
          checkFunction: () => isCompatibleWaterCoolingAndCPU(water_cooling, cpu),
        },
      ]
    : []),
  ...(ssd?.id
    ? [
        {
          name: "Сумісність материнської плати та SSD",
          checkFunction: () => isCompatibleMotherboardAndSSD(motherboard, ssd),
        },
      ]
    : []),
  {
    name: "Сумісність корпусу з компонентами",
    checkFunction: () => isCompatibleCase(pcCase, motherboard, gpu, air_cooling, water_cooling),
  },
];
```

Рисунок 4.34 – Перевірка сумісності комплектуючих в AnalyzeCompabilityModal

Уся інформація про користувача, вибрані комплектуючі, зберігається в централізованому сховищі Redux [20]. У папці redux зберігаються відповідні slices: userSlice.ts, configuratorSlice.ts, componentsSlice.ts. Це дозволяє будь-якому компоненту в застосунку отримувати чи змінювати глобальний стан без дублювання логіки (рис. 4.35).

```
const initialState: IUserInitialState = {
    user: null,
    isAuthenticated: false,
    status: Statuses.LOADING
}

const userSlice = createSlice({
    name: "user",
    initialState,
    reducers: {},
    extraReducers: (builder) => {
        builder
            .addCase(fetchUser.pending, (state) => {
                state.status = Statuses.LOADING;
            })
            .addCase(fetchUser.fulfilled, (state, action) => {
                state.user = action.payload;
                state.isAuthenticated = true;
                state.status = Statuses.SUCCESS;
            })
            .addCase(fetchUser.rejected, (state) => {
                state.status = Statuses.ERROR;
            })
            .addCase(logoutUser.pending, (state) => {
                state.status = Statuses.LOADING;
            })
            .addCase(logoutUser.fulfilled, (state) => {
                state.isAuthenticated = false;
                state.status = Statuses.SUCCESS;
            })
            .addCase(logoutUser.rejected, (state) => {
                state.status = Statuses.ERROR;
            });
    }
});
```

Рисунок 4.35 – Структура userSlice.tsx

Для взаємодії з сервером реалізовано набір сервісів у папці API/, кожен з яких відповідає за певну групу функцій. Наприклад, authService.ts виконує реєстрацію, логін, відновлення паролю (рис. 4.36); componentsService.ts — отримання комплектуючих за типом і фільтром пошуку; configurationsService.ts — роботу з готовими конфігураціями.

```ts
import axios from "@/middlewares/axios";

// types
import { TLoginBody, TRegisterBody } from "@/@types/auth";

export const register = async (body: TRegisterBody) => {
  await axios.post("/auth/register", body);
};

export const login = async (body: TLoginBody) => {
  await axios.post("/auth/login", body);
};

export const logout = async () => {
  await axios.post("/auth/logout");
};

export const refreshTokens = async () => {
  await axios.post("/auth/refresh");
};
```

Рисунок 4.36 – Взаємодія з API для роботи з користувачем

Запити виконуються через Axios [21], який налаштований в окремому файлі middlewares/axios.ts, де автоматично додаються токени авторизації до кожного запиту (рис. 4.37).

```ts
import axios from "axios";

// API
import { refreshTokens } from "@/API/authService";

const baseURL =
  process.env.NODE_ENV === "development" ? "http://localhost:3001" : "https://production.com";

const instance = axios.create({
  baseURL: `${baseURL}/api/v1`,
  withCredentials: true,
});
```

Рисунок 4.37 – Налаштування axios

Використання TypeScript підвищило точність роботи з даними. У папці @types зберігаються інтерфейси для типізації користувача, компонентів, відповідей з сервера тощо, що допомагає виявляти помилки ще під час розробки та спрощує підтримку коду. Наприклад, тип TCPU описує модель процесора з полями id, name, socket, cores, threads, base_clock, tdp, що полегшує роботу з даними при формуванні конфігурації або відображенні списку процесорів (рис. 4.38).

```ts
export type TComponentDefault = {
  id: number;
  name: string;
  image: string;
  isCompatible: boolean;
  type: ComponentTypes;
};

export type TCPU = TComponentDefault & {
  ruler: string;
  socket: string;
  cores: number;
  threads: number;
  frequency: number;
  max_frequency: number;
  cache_l3: number;
  architecture_code_name: string;
  micro_architecture: string;
  graphic_core_name: string;
  support_pcie: string;
  is_unlocked_multiplier: number;
  memory_type: string;
  memory_frequency: number;
  tdp: number;
  performance: number;
  max_memory: number;
};
```

Рисунок 4.38 – Типізація полів об’єкту процесора

4.5. Інструкція користувача

Щоб розпочати роботу з конфігуратором, користувач має створити обліковий запис, заповнивши форму реєстрації, яка включає електронну пошту, пароль і підтвердження пароля (рис. 4.39).

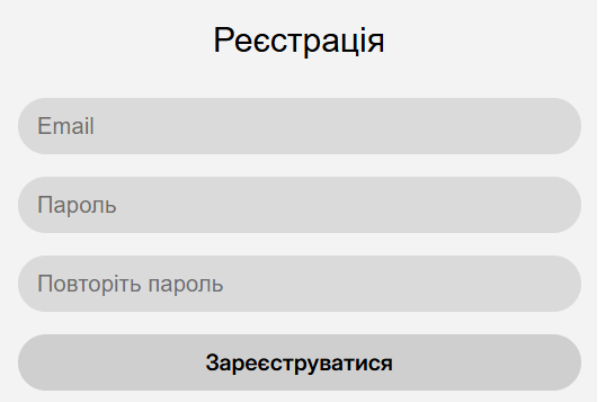


Рисунок 4.39 – Форма реєстрації

Після реєстрації потрібно увійти в систему через сторінку авторизації, ввівши електронну пошту та пароль. Якщо пароль втрачено, можна скористатися функцією відновлення (рис. 4.40).

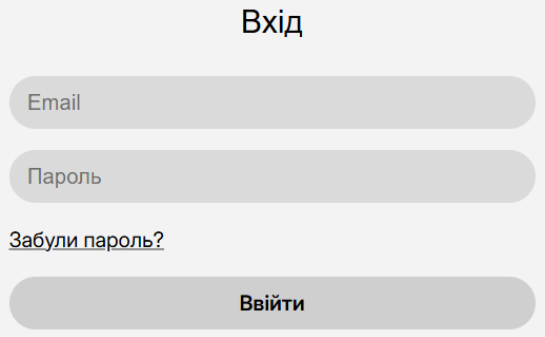


Рисунок 4.40 – Форма авторизації

Головна сторінка конфігуратора дозволяє обрати категорії комплектуючих, таких як процесори, материнські плати, відеокарти тощо (рис. 4.41).

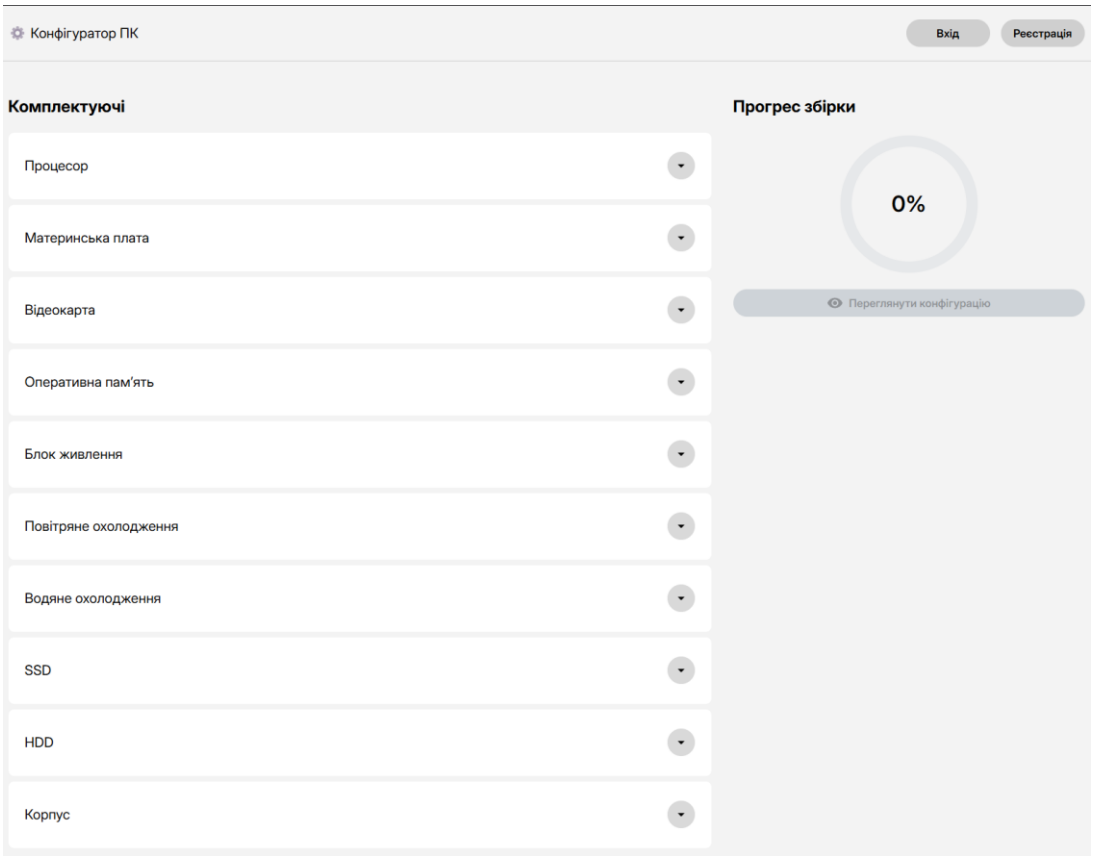


Рисунок 4.41 – Головна сторінка конфігуратора

Кожна категорія представлена у вигляді інтерактивних блоків, що ведуть до списку товарів відповідного типу (рис. 4.42).

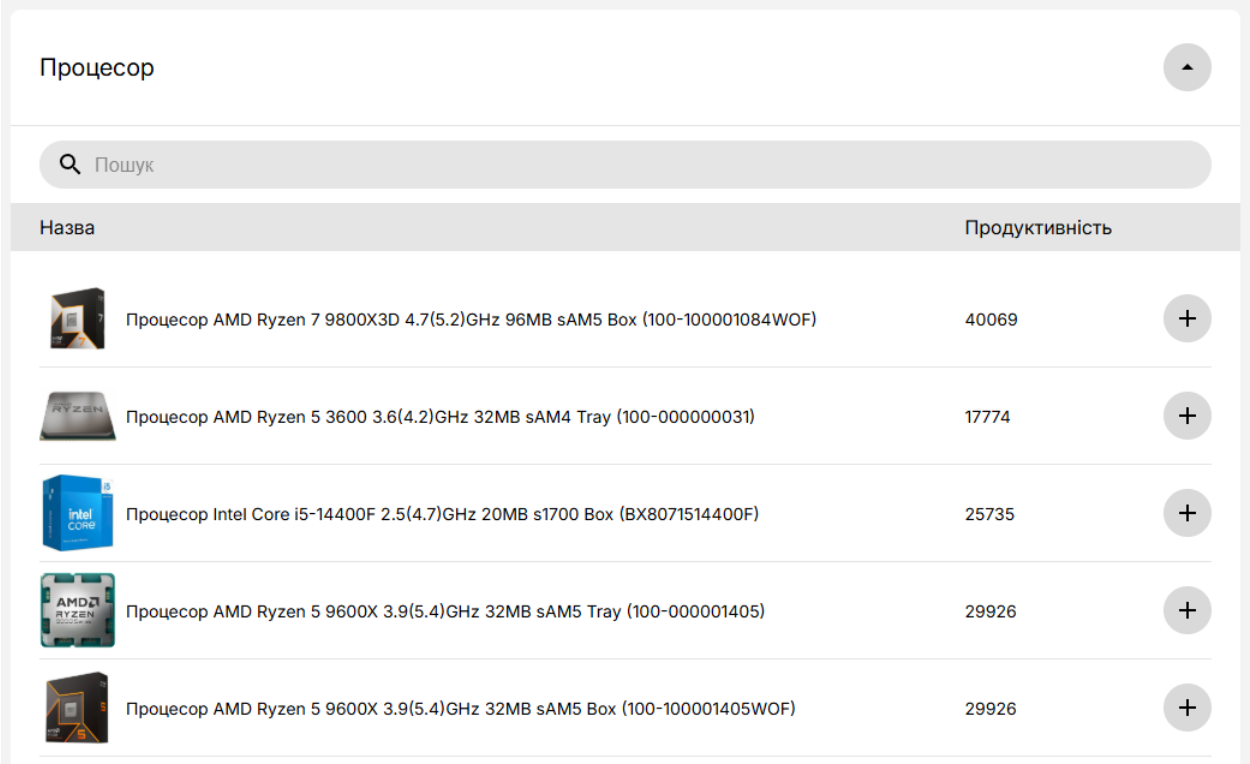


Рисунок 4.42 – Компонент категорії

Усередині категорії є поле пошуку для швидкого доступу до потрібного комплектуючого (рис. 4.43).

Рисунок 4.43 – Компонент пошуку

Натиснувши на комплектуюче, користувач переходить до вікна з детальною технічною інформацією про обраний компонент. У цьому вікні відображаються характеристики, такі як тип, модель, технічні параметри зображення та інші важливі особливості (рис. 4.44).

Процесор Intel Core i9-12900KF 3.2(5.2)GHz 30MB s1700 Box (BX8071512900KF)

Лінійка	Intel Core i9
Роз'єм процесора (Socket)	LGA1700
Кількість ядер	16 ядер
Кількість потоків	24
Тактова частота процесора	3200 GHz
Максимальна тактова частота	-
Об'єм кешу	30 MB
Кодова назва мікроархітектури	Alder Lake
Назва графічного ядра	Без вбудованої графіки
Підтримка PCIe	PCIe 5.0
Розблокований множник	+
Тип пам'яті	DDR4: 3200 MHz
Термопакет	125 Вт
Продуктивність	41341

Рисунок 4.44 – Технічна інформація комплектуючого

Під час вибору система автоматично перевіряє сумісність з іншими компонентами. У разі сумісності з’являється відповідна позначка (рис. 4.45).

Рисунок 4.45 – Позначка сумісності

Обрані комплектуючі відображаються у відповідних блоках інтерфейсу — з назвою та зображенням, що полегшує орієнтацію в збірці (рис. 4.46).

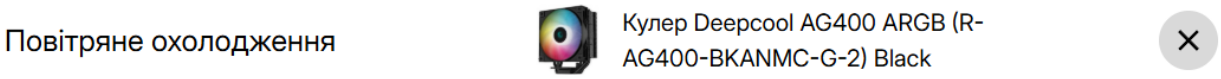


Рисунок 4.46 – Вигляд обраного комплектуючого

Коли вибрано всі необхідні деталі, шкала заповнюється повністю, візуально сигналізуючи про завершення етапу вибору компонентів. У цей момент стає активною кнопка переходу до перегляду конфігурації (рис. 4.47).

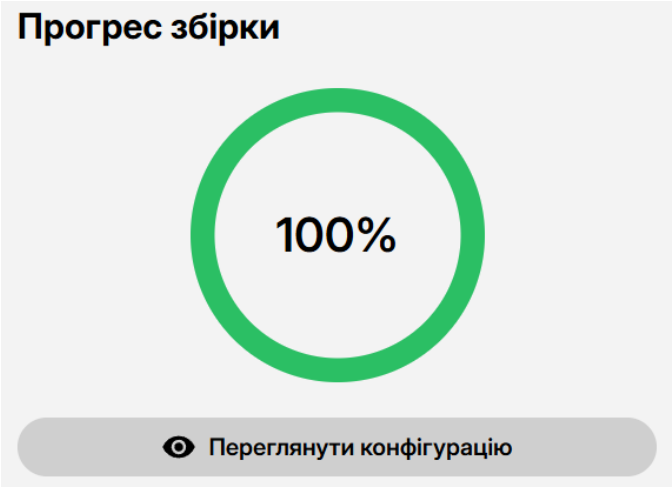


Рисунок 4.47 – Секція прогресу збірки

У вікні перегляду показано список комплектуючих з короткими характеристиками, оцінку продуктивності за шкалою PassMark, а також кнопки для збереження збірки, аналізу сумісності та загального аналізу (рис. 4.48).

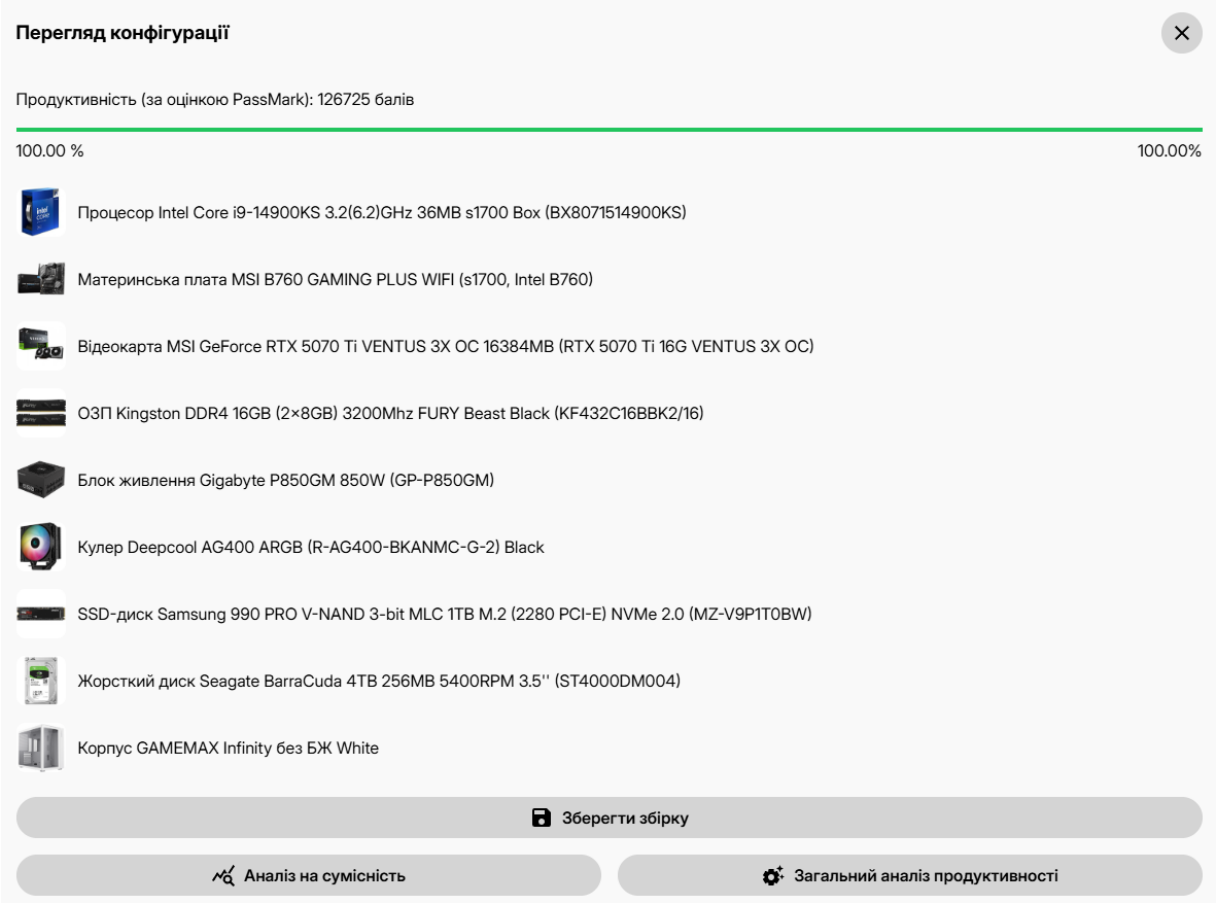


Рисунок 4.48 – Вікно перегляду конфігурації

Після збереження збірка стає доступною на окремій сторінці, де можна переглядати всі конфігурації користувача (рис. 4.49).

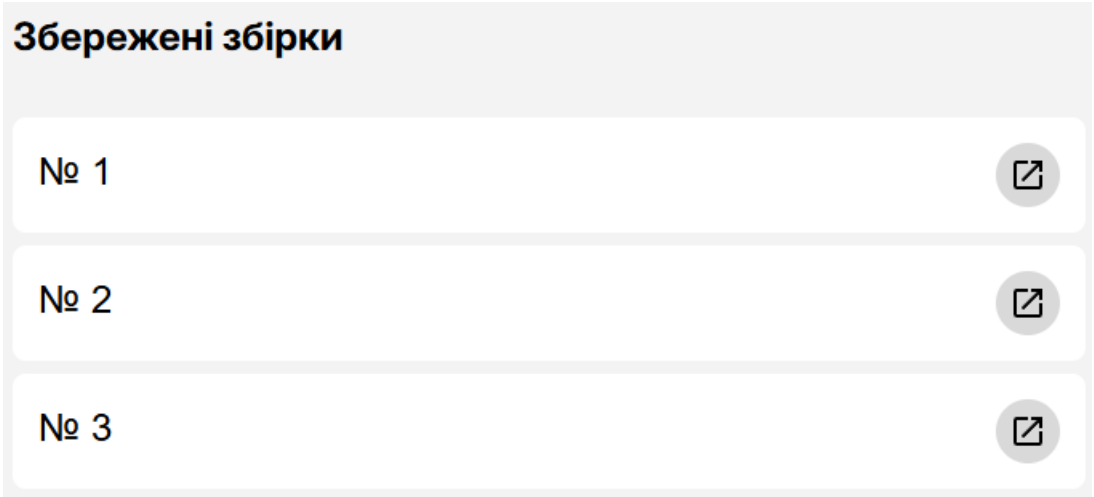


Рисунок 4.49 –Сторінка «Збережені збірки»

Функція аналізу сумісності перевіряє технічні параметри, такі як сокети, частоти, інтерфейси, об’єм пам’яті, форм-фактори та енергоспоживання. Вона автоматично порівнює вибрані компоненти між собою та повідомляє користувача про можливі конфлікти або несумісності (рис. 4.50).

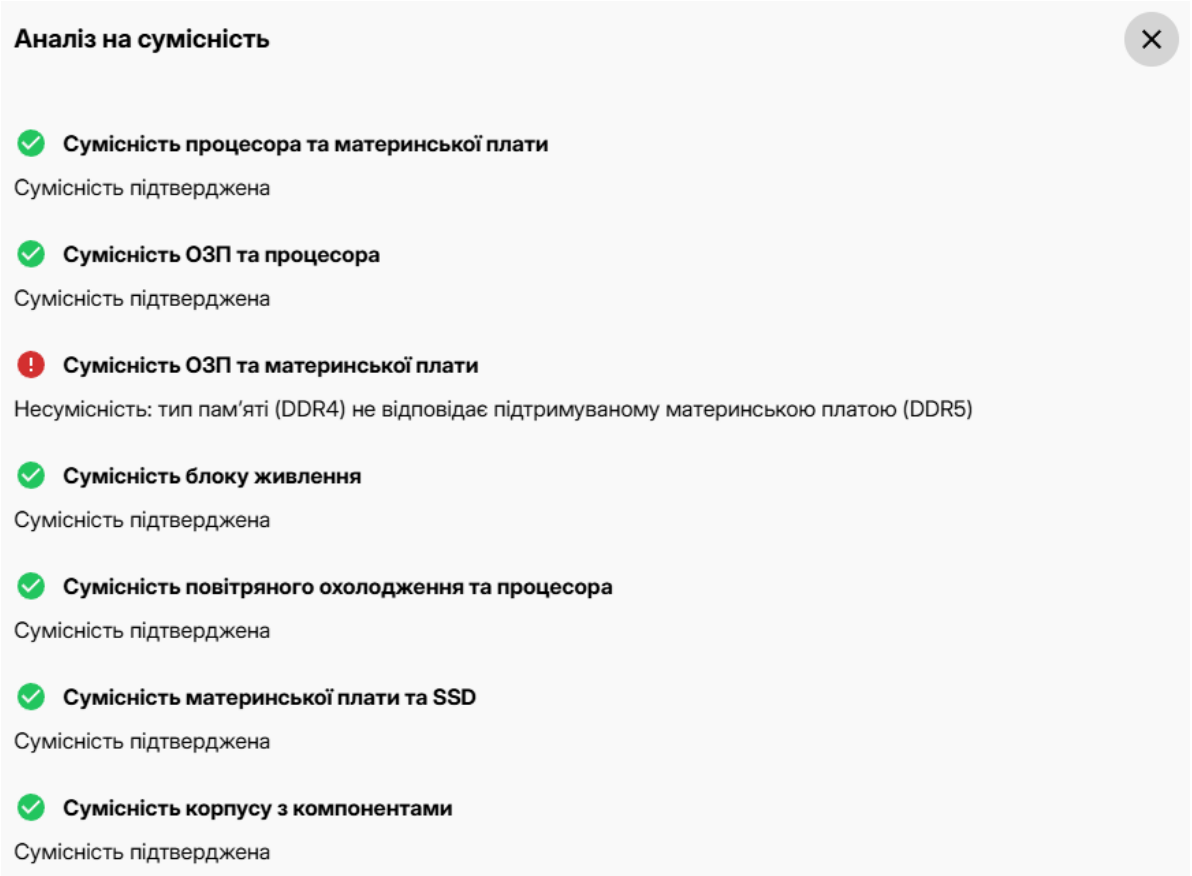


Рисунок 4.50 – Модальне вікно аналізу сумісності

Це дозволяє уникнути помилок при складанні системи та забезпечує правильну роботу ПК після збирання.

Аналіз продуктивності оцінює загальну потужність зібраної системи на основі вибраних компонентів, таких як процесор, відеокарта, обсяг оперативної пам’яті та інші параметри. На основі цієї оцінки система пропонує можливі сценарії використання конфігурації — для офісної роботи, навчання, ігор, професійного монтажу чи рендерингу (рис. 4.51).

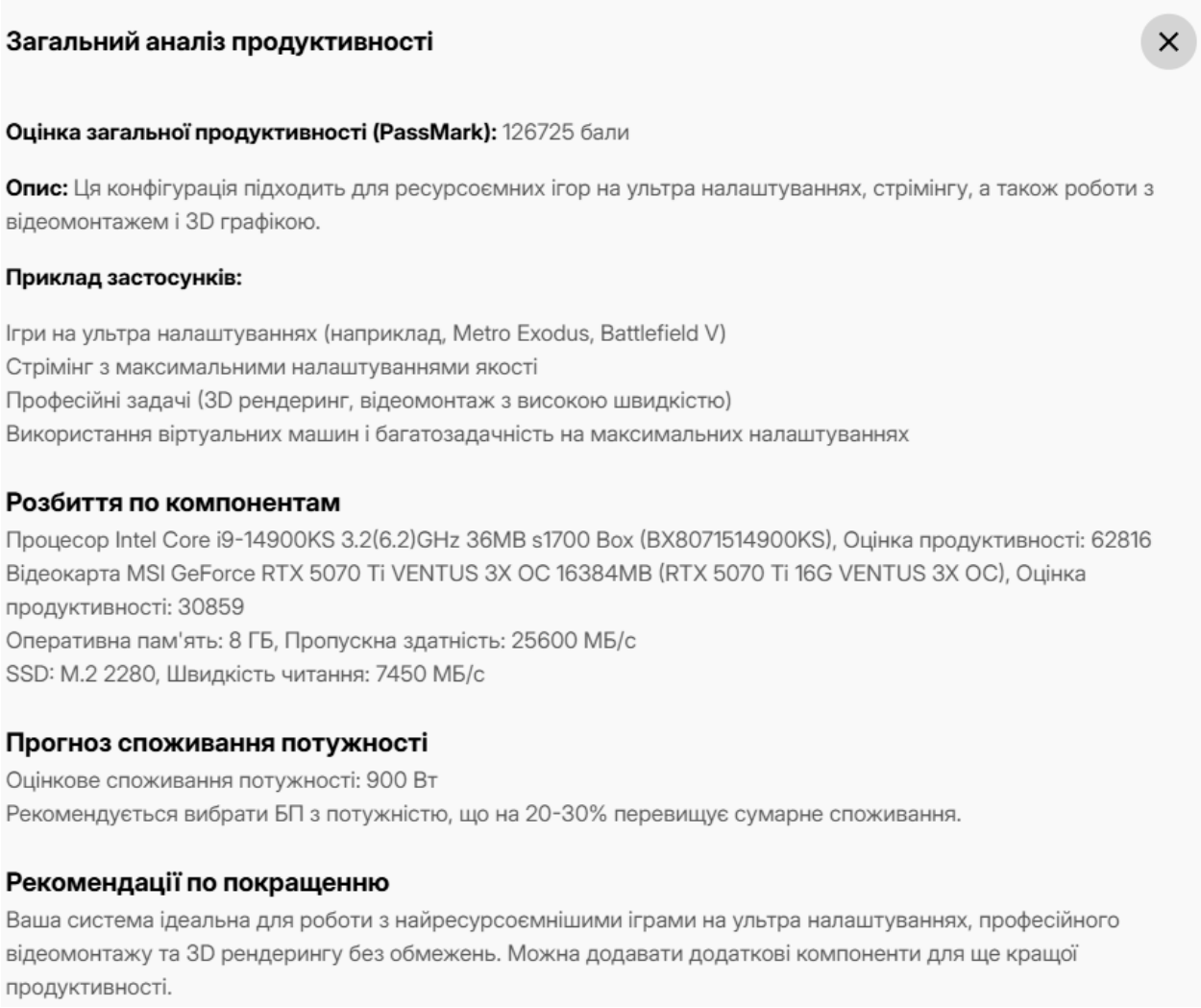


Рисунок 4.51 – Модальне вікно аналізу продуктивності

Також надається прогноз енергоспоживання всієї системи, що допомагає правильно підібрати блок живлення. Крім того, користувач отримує поради щодо покращення конфігурації для досягнення вищої продуктивності або енергоефективності.

4.6. Аналіз адаптивності та швидкодії сайту

Сайт адаптується під різні пристрої і коректно відображається на різних пристроях. На прикладі нижче представлено вигляд інтерфейсу в браузері

Safari на операційній системі iOS (рис. 4.52), що підтверджує зручність використання та читабельність на мобільних платформах.

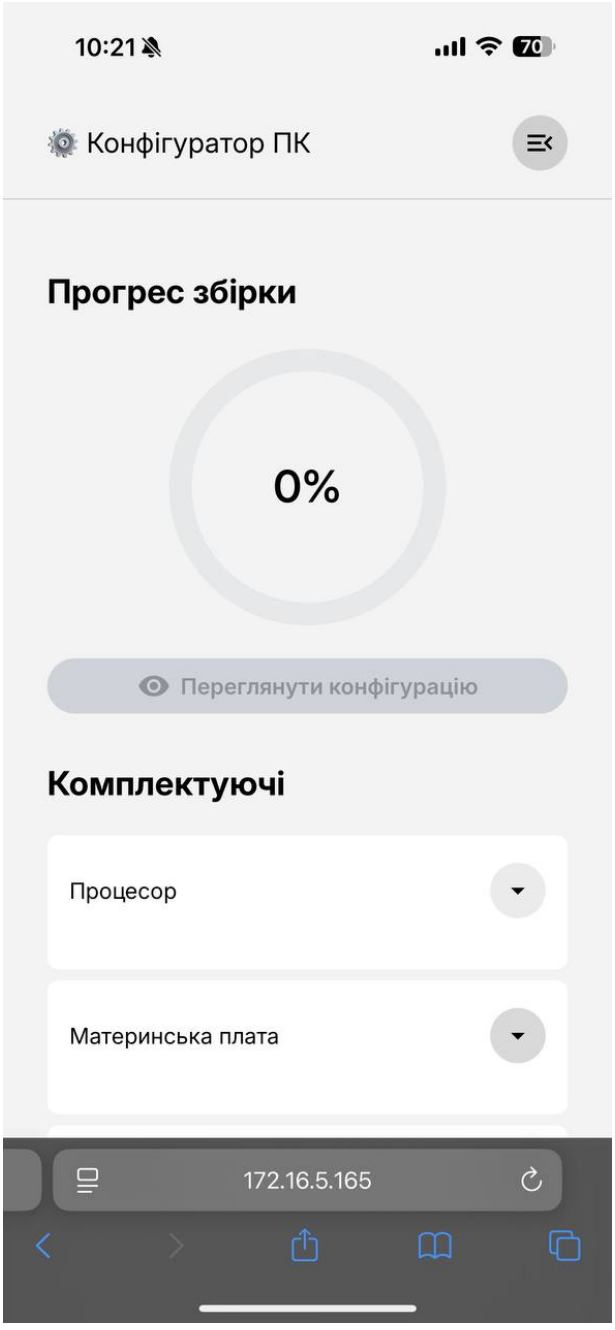


Рисунок 4.52 – Робота сайту на iOS

Також інтерфейс сайту був протестований у браузері Google Chrome на операційній системі Windows, де він так само відображається стабільно та коректно, з правильною адаптацією елементів до розміру вікна (рис. 4.53).

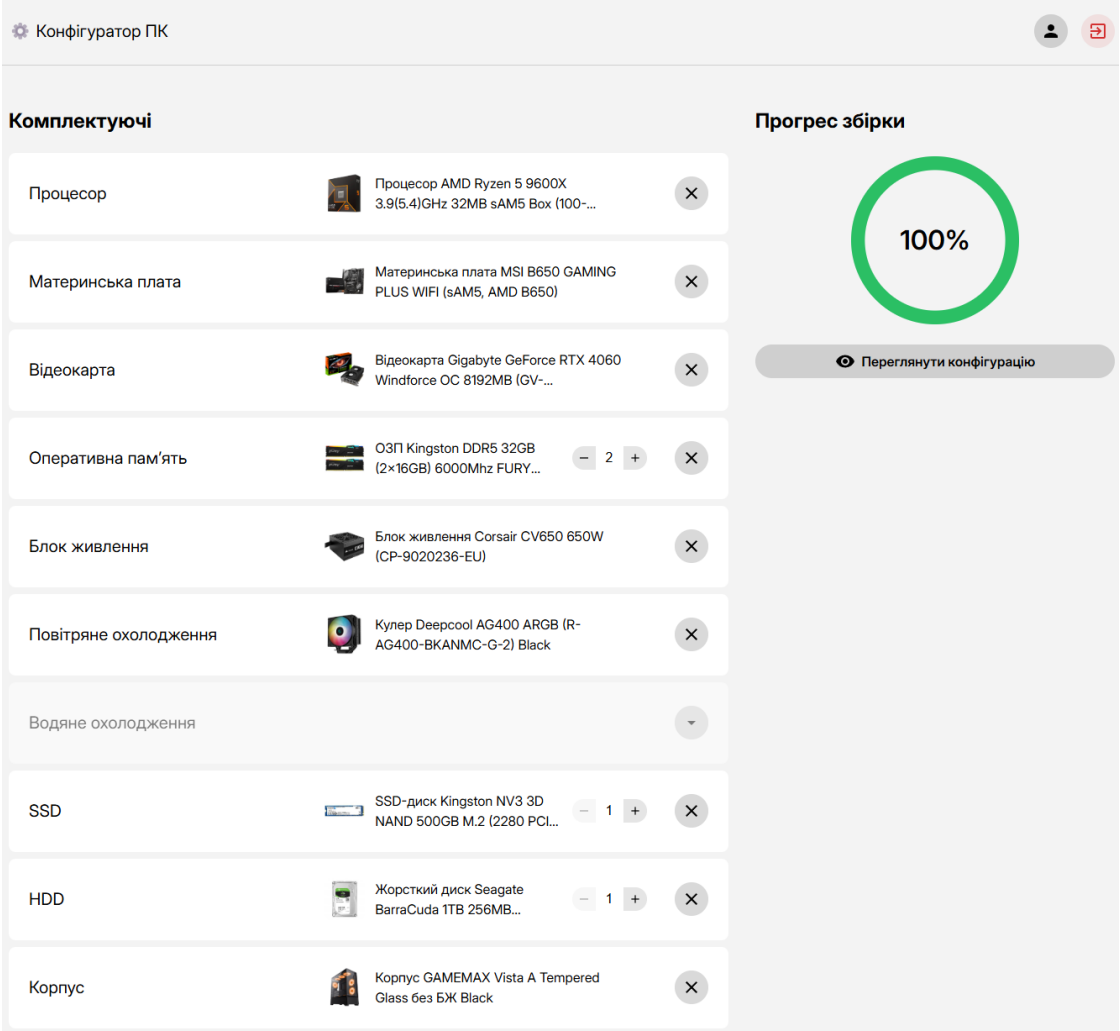


Рисунок 4.53 – Робота сайту на Windows

Сайт коректно працює на різних пристроях, а інтерфейс адаптується під малі екрани, забезпечуючи зручність. Робота ресурсу швидка та стабільна.

Для оцінки основних показників було використано вбудований інструмент LightHouse у браузері Google Chrome. Результати перевірки засвідчили високий рівень продуктивності сайту [22] — максимальні бали були отримані за доступність, оптимізацію для пошукових систем та відповідність найкращим практикам розробки (рис. 4.54).

Рисунок 4.54 – Звіт Lighthouse

ВИСНОВКИ

У процесі розробки сайту "Конфігуратор ПК з аналізом продуктивності" були реалізовані ключові аспекти для зручності користувачів і швидкої роботи додатку, що дозволяє ефективно обирати компоненти для складання персонального комп'ютера. Створено інтуїтивно зрозумілий інтерфейс, який дозволяє легко вибирати та комбінувати компоненти, враховуючи їхню сумісність. Простий процес вибору і наочний пошук допомагають користувачам швидко знаходити необхідні елементи для конфігурації.

Веб-додаток побудовано за допомогою React.js та TypeScript, що покращує керування станом і забезпечує точний контроль типів. Адаптивна верстка на SCSS підлаштовує сайт під різні пристрої, забезпечуючи високу продуктивність і зручність на будь-якому екрані. Для серверної частини використано Node.js та Express.js, що гарантує швидку обробку запитів і високу продуктивність.

Застосовано методи, які зменшують час завантаження сайту, зокрема мінімізація бандлів JavaScript за допомогою Vite, вибір необхідних бібліотек і оптимізація компонентів. Використано Redux для керування станом додатку та MUI для створення сучасного і зручного інтерфейсу.

Інструмент для порівняння продуктивності компонентів дає змогу користувачам вибирати найбільш відповідні варіанти для їхніх потреб, що допомагає створити збалансовану і потужну конфігурацію ПК. Завдяки комплексному підходу проект став зручним і швидким інструментом для вибору компонентів, який відповідає вимогам сучасних користувачів.

Окрім технічної реалізації, результати кваліфікаційної роботи пройшли апробацію у вигляді доповіді та тез на конференції «XLVIII Міжнародна наукова студентська конференція».

СПИСОК ЛІТЕРАТУРИ

1. Visual Studio Code – Режим доступу: https://code.visualstudio.com (дата звернення: 12.05.2025).

2. Томсон В. Розробка веб-додатків за допомогою PHP та MySQL. Київ: Світ, 2017. 768 с.

3. Vite – Режим доступу: https://vite.dev (дата звернення: 12.05.2025).

4. React – Режим доступу: https://reactjs.org (дата звернення: 12.05.2025).

5. Банзал А.: React в дії. Київ: Наш Формат, 2020. 328 с.

6. Material UI – Режим доступу: https://mui.com (дата звернення: 12.05.2025).

7. Крамер Б.: Sass and Compass in Action. Shelter Island: Manning Publications, 2013. 312 p.

8. Паверс Ш.: Вивчаємо Node. Переходимо на сторону сервера. 2-е видання, основи Node-модулів та Node Package Manager. Запоріжжя: Дія, 2020. 55-86 с.

9. Express.js – Режим доступу: https://expressjs.com (дата звернення: 12.05.2025).

10. Річардсон Л., Рубі С.: RESTful Web Services. Київ: Діалектика, 2009. 448 с.

11. Фуше Л.: MySQL для початківців. Львів: Піраміда, 2019. 312 с.

12. Redux Toolkit – Режим доступу: https://redux-toolkit.js.org (дата звернення: 12.05.2025).

13. Байер Е.: Семантичне ядро сайту: створення, аналіз, оптимізація. Львів: Астролябія, 2021. 198 с.

14. TypeScript – Режим доступу: https://www.typescriptlang.org (дата звернення: 12.05.2025).

15. Node.js – Режим доступу: https://nodejs.org (дата звернення:

12.05.2025).

16. Фаррелл Д.: Вивчаємо SQL. Практичний підхід. Одеса: Фенікс, 2021. 400 с.

17. Гріффітс Н.: Безпечна аутентифікація: JWT, OAuth 2.0 та OpenID Connect. Львів: Розумна книга, 2021. 264 с.

18. Гольдберг А.: Node.js і Express.js: програмування серверної частини вебзастосунків. Харків: Фактор, 2021. 368 с.

19. React Router – Режим доступу: https://reactrouter.com (дата звернення: 12.05.2025).

20. Абрамов Д., Мезенцев М.: Освоєння Redux: керування станом React-додатків. Київ: ІТ-книга, 2021. 220 с.

21. Річардс А.: HTTP-запити у веброзробці: Axios, Fetch та REST API. Харків : Фактор, 2022. 256 с.

22. Дакс М.: Семантична верстка і SEO: створення доступних і оптимізованих вебзастосунків. Київ: ВебКод, 2021. 312 с.

23. Ольховська О.: Методичні рекомендації до виконання кваліфікаційної роботи. Полтава: РВВ ПУЕТ 2025. 58с.

ДОДАТОК А. ПРОГРАМНИЙ КОД

КОД ГОЛОВНОЇ СТОРІНКИ

```
import styles from "./ConfiguratorPage.module.scss";
import PrimaryLayout from "@/layouts/PrimaryLayout";
import Configurator from "@/components/Configurator";
import ConfigureProgress from "@/components/ConfigureProgress";
const ConfiguratorPage: React.FC = () => {
  return (
    <PrimaryLayout>
      <div className="container">
        <div className={styles.content}>
          <div className={styles.content__wrapper}>
            <Configurator />
            <ConfigureProgress />
          </div>
        </div>
      </div>
    </PrimaryLayout>
  );
};
export default ConfiguratorPage;
```

КОД СТОРІНКИ РЕЄСТРАЦІЇ

```
import { Box, TextField, Typography } from "@mui/material";
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import { useNavigate } from "react-router-dom";
import { useState } from "react";
import * as yup from "yup";
import PrimaryLayout from "@/layouts/PrimaryLayout";
import Button from "@/components/Button";
import { register as registerAccount } from "@/API/authService";
import { TRegisterBody } from "@/@types/auth";
import toast from "react-hot-toast";
const schema = yup.object().shape({
  email: yup.string().required("Поле обов' язкове").email("Введіть коректний email"),
  password: yup
    .string()
    .required("Поле обов' язкове")
    .min(8, "Пароль повинен містити мінімум 8 символів"),
  confirmPassword: yup
    .string()
    .oneOf([yup.ref("password"), undefined], "Паролі не співпадають")
    .required("Поле обов'язкове"),
});
const RegisterPage: React.FC = () => {
  const [isLoading, setIsLoading] = useState<boolean>(false);
  const navigate = useNavigate();
```

```
const {
  register,
  handleSubmit,
  formState: { errors },
} = useForm({
  resolver: yupResolver(schema),
});
const onSubmit = async (body: TRegisterBody) => {
  try {
    setIsLoading(true);
    await registerAccount(body);
    toast.success("Успішна реєстрація");
    navigate("/login");
  } catch (err: any) {
    toast.error(err?.response.data.message);
  } finally {
    setIsLoading(false);
  }
};
return (
  <PrimaryLayout>
    <div className="container">
      <Box
        display="flex"
        justifyContent="center"
        alignItems="center"
        minHeight="calc(100vh - 80px)"
      >
        <Box display="flex" flexDirection="column" gap={3} maxWidth={400} width="100%">
          <Typography component="h2" variant="h5" textAlign="center">
            Реєстрація
          </Typography>
          <form onSubmit={handleSubmit(onSubmit)}>
            <Box display="flex" flexDirection="column" gap={2}>
              <TextField
                label="Email"
                autoComplete="off"
                variant="outlined"
                size="small"
                fullWidth
                {...register("email")}
                error={!!errors.email}
                helperText={errors.email?.message}
              />
              <TextField
                label="Пароль"
                autoComplete="new-password"
                variant="outlined"
                size="small"
                type="password"
                fullWidth
                {...register("password")}
```

```
              error={!!errors.password}
              helperText={errors.password?.message}
             />
             <TextField
              label="Повторіть пароль"
              autoComplete="new-password"
              variant="outlined"
              size="small"
              type="password"
              fullWidth
              {...register("confirmPassword")}
              error={!!errors.confirmPassword}
              helperText={errors.confirmPassword?.message}
             />
             <Button type="submit" variant="primary" isLoading={isLoading}>
              Зареєструватися
             </Button>
            </Box>
           </form>
          </Box>
         </Box>
        </div>
      </PrimaryLayout>
    );
};
export default RegisterPage;
```

КОД СТОРІНКИ АВТОРИЗАЦІЇ

```
import { Box, Link, TextField, Typography } from "@mui/material";
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import toast from "react-hot-toast";
import { useNavigate } from "react-router-dom";
import { useState } from "react";
import * as yup from "yup";
import PrimaryLayout from "@/layouts/PrimaryLayout";
import Button from "@/components/Button";
import { login } from "@/API/authService";
import { TRegisterBody } from "@/@types/auth";
const schema = yup.object().shape({
  email: yup.string().required("Поле обов' язкове").email("Введіть коректний email"),
  password: yup
    .string()
    .required("Поле обов' язкове")
    .min(8, "Пароль повинен містити мінімум 8 символів"),
});
const LoginPage: React.FC = () => {
  const [isLoading, setIsLoading] = useState<boolean>(false);
  const navigate = useNavigate();
  const {
    register,
    handleSubmit,
```

```
  formState: { errors },
} = useForm({
  resolver: yupResolver(schema),
});
const onSubmit = async (body: TRegisterBody) => {
  try {
    setIsLoading(true);
    await login(body);
    toast.success("Успішний вхід");
    navigate("/");
  } catch (err: any) {
    toast.error(err?.response.data.message);
  } finally {
    setIsLoading(false);
  }
};
return (
  <PrimaryLayout>
    <div className="container">
      <Box
        display="flex"
        justifyContent="center"
        alignItems="center"
        minHeight="calc(100vh - 80px)"
      >
        <Box display="flex" flexDirection="column" gap={3} maxWidth={400} width="100%">
          <Typography component="h2" variant="h5" textAlign="center">
            Вхід
          </Typography>
          <form onSubmit={handleSubmit(onSubmit)}>
            <Box display="flex" flexDirection="column" gap={2}>
              <TextField
                label="Email"
                autoComplete="off"
                variant="outlined"
                size="small"
                fullWidth
                {...register("email")}
                error={!!errors.email}
                helperText={errors.email?.message}
              />
              <TextField
                label="Пароль"
                autoComplete="new-password"
                variant="outlined"
                size="small"
                type="password"
                fullWidth
                {...register("password")}
                error={!!errors.password}
                helperText={errors.password?.message}
              />
```

```
            <Link color="#000000" href="/reset-password">
              Забули пароль?
            </Link>
            <Button type="submit" variant="primary" isLoading={isLoading}>
              Ввійти
            </Button>
          </Box>
        </form>
      </Box>
    </Box>
  </div>
 </PrimaryLayout>
 );
};
export default LoginPage;
```

СТОРІНКА ВІДНОВЛЕННЯ ПАРОЛЮ

```
import { Box, Typography } from "@mui/material";
import { useLocation } from "react-router-dom";
import { useEffect, useState } from "react";
import qs from "qs";
import PrimaryLayout from "@/layouts/PrimaryLayout";
import SubmitEmailForm from "@/components/Forms/SubmitEmailForm";
import ResetPasswordForm from "@/components/Forms/ResetPasswordForm";
import { checkResetToken } from "@/API/userService";
const ResetPasswordPage = () => {
 const [mode, setMode] = useState<"email" | "password">("email");
 const location = useLocation();
 const queryParams = qs.parse(location.search, { ignoreQueryPrefix: true });
 const token = queryParams.token as string | undefined;
 useEffect(() => {
  if (!token) {
   setMode("email");
  }
  (async () => {
   try {
    await checkResetToken(token as string);
    setMode("password");
   } catch (err) {
    setMode("email");
   }
  })();
 }, [token]);
 return (
  <PrimaryLayout>
   <div className="container">
    <Box
     display="flex"
     justifyContent="center"
     alignItems="center"
     minHeight="calc(100vh - 80px)"
    >
```

```
        <Box display="flex" flexDirection="column" gap={3} maxWidth={400} width="100%">
          <Typography component="h2" variant="h5" textAlign="center">
            {mode === "email" ? "Забули пароль?" : "Зміна паролю"}
          </Typography>
          {mode === "email" ? <SubmitEmailForm /> : <ResetPasswordForm token={token} />}
        </Box>
      </Box>
      s
    </div>
  </PrimaryLayout>
 );
};
export default ResetPasswordPage;
```

СТОРІНКА ПРОФІЛЮ

```
import React, { useEffect, useState } from "react";
import toast from "react-hot-toast";
import { Box, Skeleton, Stack, Typography } from "@mui/material";
import styles from "./ProfilePage.module.scss";
import Button from "@/components/Button";
import PreviewConfigure from "@/components/PreviewConfigure";
import PrimaryLayout from "@/layouts/PrimaryLayout";
import { getConfigurations } from "@/API/configurationsService";
import { TConfiguration } from "@/@types/configuration";
import { OpenInNew } from "@mui/icons-material";
const ProfilePage: React.FC = () => {
 const [isLoading, setIsLoading] = useState<boolean>(true);
 const [isPreviewOpened, setIsPreviewOpened] = useState<boolean>(false);
 const [configurations, setConfigurations] = useState<TConfiguration[]>([]);
 const [selectedConfiguration, setSelectedConfiguration] = useState<TConfiguration |
null>(null);
 useEffect(() => {
  (async () => {
   try {
    const data = await getConfigurations();
    setConfigurations(data);
   } catch (err) {
    console.log(err);
    toast.error("Виникла помилка при отриманні конфігурацій");
   } finally {
    setIsLoading(false);
   }
  })();
 }, []);
 const handlePreview = (config: TConfiguration) => {
  setSelectedConfiguration(config);
  setIsPreviewOpened(true);
 };
 return (
  <>
   <PrimaryLayout>
    <Box className="container">
```

```
          <Box className={styles.content}>
            <div className={styles.content__wrapper}>
              <h2>Збережені збірки</h2>
              <div className={styles.content__configurations}>
                {!isLoading
                  ? configurations.map((c) => (
                      <Stack
                        sx={{
                          backgroundColor: "#ffffff",
                          padding: 2,
                          borderRadius: 2,
                          flexDirection: "row",
                          justifyContent: "space-between",
                        }}
                        key={c.config_id}
                      >
                        <Typography fontSize={24} color="#black">
                          № {c.config_id}
                        </Typography>
                        <Button
                          onClick={() => handlePreview(c)}
                          variant="iconary"
                          variantColor="default"
                        >
                          <OpenInNew />
                        </Button>
                      </Stack>
                    ))
                  : [...Array(6)].map((_, idx) => (
                      <Skeleton key={idx} variant="rounded" height={40} />
                    ))}
              </div>
            </div>
          </Box>
        </Box>
      </PrimaryLayout>
      {selectedConfiguration && (
        <PreviewConfigure
          data={selectedConfiguration}
          isOpened={isPreviewOpened}
          handleClose={() => setIsPreviewOpened(false)}
        />
      )}
    </>
  );
};
export default ProfilePage;
```

API ШЛЯХИ ПОВ’ЯЗАНІ З АУТЕНТИФІКАЦІЄЮ

```
import axios from "@/middlewares/axios";
import { TLoginBody, TRegisterBody } from "@/@types/auth";
export const register = async (body: TRegisterBody) => {
```

```
  await axios.post("/auth/register", body);
};
export const login = async (body: TLoginBody) => {
  await axios.post("/auth/login", body);
};
export const logout = async () => {
  await axios.post("/auth/logout");
};
export const refreshTokens = async () => {
  await axios.post("/auth/refresh");
};
```

API ШЛЯХИ ПОВ'ЯЗАНІ З КОРИСТУВАЧЕМ

```
import axios from "@/middlewares/axios";
import { TResetPasswordBody } from "@/@types/user";
export const getUser = async () => {
  const { data } = await axios.get("/user/get");
  return data;
};
export const sendResetPasswordLetter = async (email: string) => {
  const { data } = await axios.post("/user/send-reset-password", { email });
  return data;
};
export const checkResetToken = async (token: string) => {
  const { data } = await axios.post("/user/check-reset-token", { token });
  return data;
};
export const resetPassword = async (body: TResetPasswordBody) => {
  const { data } = await axios.post("/user/reset-password", body);
  return data;
};
```

КОМПОНЕНТ КОНФІГУРАТОР

```
import { useState, useCallback, useEffect, useRef } from "react";
import { useSelector } from "react-redux";
import { useAppDispatch } from "@/redux/store";
import styles from "./Configurator.module.scss";
import ConfiguratorItem from "../ConfiguratorItem";
import { getComponents } from "@/API/componentsService";
import { configureSelector } from "@/redux/slices/configure/selectors";
import { selectComponent } from "@/redux/slices/configure/slice";
import { ComponentTypes } from "@/@types/enums";
import { TComponent } from "@/@types/components";
import {
  isAirCooling,
  isCase,
  isCPU,
  isGPU,
  isMotherboard,
  isPSU,
  isRAM,
  isWaterCooling,
```

```
} from "@/utils/typeGuards";
const componentTypes = [
  { type: ComponentTypes.CPU, name: "Процесор" },
  { type: ComponentTypes.MOTHERBOARD, name: "Материнська плата" },
  { type: ComponentTypes.GPU, name: "Відеокарта" },
  { type: ComponentTypes.RAM, name: "Оперативна пам’ять" },
  { type: ComponentTypes.PSU, name: "Блок живлення" },
  { type: ComponentTypes.AIR_COOLING, name: "Повітряне охолодження" },
  { type: ComponentTypes.WATER_COOLING, name: "Водяне охолодження" },
  { type: ComponentTypes.SSD, name: "SSD" },
  { type: ComponentTypes.HDD, name: "HDD" },
  { type: ComponentTypes.CASE, name: "Корпус" },
];
const Configurator: React.FC = () => {
  const dispatch = useAppDispatch();
  const { selectedComponents } = useSelector(configureSelector);
  const timeoutRef = useRef<NodeJS.Timeout | null>(null);
  const initialComponents = componentTypes.reduce(
    (acc, { type }) => ({
      ...acc,
      [type]: { data: [], totalPages: 0, currentPage: 1 },
    }),
    {} as Record<ComponentTypes, { data: any[]; totalPages: number; currentPage: number }>
  );
  const initialSearchValues = componentTypes.reduce(
    (acc, { type }) => ({
      ...acc,
      [type]: "",
    }),
    {} as Record<ComponentTypes, string>
  );
  const [components, setComponents] = useState(initialComponents);
  const [searchValues, setSearchValues] = useState(initialSearchValues);
  const [openedItems, setOpenedItems] = useState<Record<ComponentTypes, boolean>>(
    componentTypes.reduce(
      (acc, { type }) => ({ ...acc, [type]: false }),
      {} as Record<ComponentTypes, boolean>
    )
  );
  const fetchComponent = useCallback(
    async (type: ComponentTypes, search?: string, page?: number) => {
      try {
        const response = await getComponents({ type, page, search });
        setComponents((prevState) => ({
          ...prevState,
          [type]: {
            data: response.data.map((component: TComponent) => ({
              ...component,
              isCompatible: checkCompatibility(type, component),
              type,
            })),
            totalPages: response.totalPages,
```

```
          currentPage: prevState[type].currentPage,
        },
      }));
    } catch (error) {
      console.error(`Error fetching ${type}:`, error);
    }
  },
  []
);
const checkCompatibility = useCallback(
  (type: ComponentTypes, component: TComponent): boolean => {
    const { cpu, motherboard, gpu, air_cooling, water_cooling } = selectedComponents;
    switch (type) {
      case ComponentTypes.CPU:
        if (!motherboard || !isMotherboard(motherboard) || !isCPU(component)) return false;
        return motherboard.socket === component.socket;
      case ComponentTypes.MOTHERBOARD:
        if (!cpu || !isCPU(cpu) || !isMotherboard(component)) return false;
        return cpu.socket === component.socket;
      case ComponentTypes.RAM:
        if (
          !motherboard ||
          !cpu ||
          !isMotherboard(motherboard) ||
          !isCPU(cpu) ||
          !isRAM(component)
        )
          return false;
        return (
          motherboard.memory_type === component.memory_type &&
          motherboard.count_memory_slots >= component.count_modules &&
          motherboard.min_frequency_memory <= component.frequency &&
          motherboard.max_frequency_memory >= component.frequency &&
          cpu.memory_frequency >= component.frequency
        );
      case ComponentTypes.PSU:
        if (!gpu || !cpu || !isGPU(gpu) || !isCPU(cpu) || !isPSU(component)) return false;
        return gpu.tdp * 0.75 + cpu.tdp <= component.power;
      case ComponentTypes.AIR_COOLING:
        if (!cpu || !isCPU(cpu) || !isAirCooling(component)) return false;
        return component.socket.includes(cpu.socket) && cpu.tdp <= component.max_tdp;
      case ComponentTypes.WATER_COOLING:
        if (!cpu || !isCPU(cpu) || !isWaterCooling(component)) return false;
        return component.socket.includes(cpu.socket);
      case ComponentTypes.CASE:
        const caseComponent = component;
        if (!motherboard || !isMotherboard(motherboard) || !isCase(caseComponent)) {
          return false;
        }
        const isFormFactorCompatible = (
          caseFormFactor: string,
          motherboardFormFactor: string
```

```
        ): boolean => {
          const caseFactors = caseFormFactor?.split("/").map((f) => f.trim().toLowerCase());
          const motherboardFactor = motherboardFormFactor.trim().toLowerCase();
          return caseFactors?.includes(motherboardFactor);
        };
        const hasAirCooling = !!air_cooling && isAirCooling(air_cooling);
        const hasWaterCooling = !!water_cooling && isWaterCooling(water_cooling);
        const supportsAirCooling = hasAirCooling
          ? caseComponent.max_height_air_cooling >= air_cooling.height
          : true;
        const supportsWaterCooling = hasWaterCooling
          ? typeof caseComponent.is_support_water_cooling === "number"
          : true;
        return (
          isFormFactorCompatible(
            caseComponent.form_factor_motherboard,
            motherboard.form_factor
          ) &&
          supportsWaterCooling &&
          supportsAirCooling
        );
      default:
        return false;
    }
  },
  [selectedComponents]
);
const handleSearchChange = (type: ComponentTypes) => (value: string | undefined) => {
  setSearchValues((prevState) => ({
    ...prevState,
    [type]: value || "",
  }));
  setComponents((prevState) => ({
    ...prevState,
    [type]: { ...prevState[type], currentPage: 1 },
  }));
};
const handlePageChange = (type: ComponentTypes, newPage: number) => {
  setComponents((prevState) => ({
    ...prevState,
    [type]: { ...prevState[type], currentPage: newPage },
  }));
  fetchComponent(type, searchValues[type], newPage);
};
const handleToggleItem = (type: ComponentTypes) => {
  setOpenedItems((prevState) => {
    const isOpened = !prevState[type];
    if (isOpened && components[type].data.length === 0) {
      fetchComponent(type);
    }
    return { ...prevState, [type]: isOpened };
  });
```

```
};
useEffect(() => {
  if (timeoutRef.current) {
    clearTimeout(timeoutRef.current);
  }
  timeoutRef.current = setTimeout(() => {
    Object.keys(searchValues).forEach((key) => {
      const type = key as ComponentTypes;
      const search = searchValues[type];
      if (search !== "") {
        fetchComponent(type, search, 1);
      } else {
        fetchComponent(type, "", 1);
      }
    });
  }, 350);
  return () => {
    if (timeoutRef.current) {
      clearTimeout(timeoutRef.current);
    }
  };
}, [searchValues, fetchComponent]);
useEffect(() => {
  componentTypes.forEach(({ type }) => {
    fetchComponent(type);
  });
}, [fetchComponent]);
useEffect(() => {
  setComponents((prevState) => {
    const updatedComponents = { ...prevState };
    Object.keys(updatedComponents).forEach((key) => {
      const type = key as ComponentTypes;
      updatedComponents[type] = {
        ...updatedComponents[type],
        data: updatedComponents[type].data.map((component) => ({
          ...component,
          isCompatible: checkCompatibility(type, component),
        })),
      };
    });
    return updatedComponents;
  });
}, [selectedComponents, checkCompatibility]);
return (
  <div className={styles.configurator}>
    <h2>Комплектуючі</h2>
    <div className={styles.configurator__list}>
      {componentTypes.map(({ type, name }) => {
        const isDisabled = !!(
          (type === ComponentTypes.AIR_COOLING &&
            selectedComponents[ComponentTypes.WATER_COOLING]) ||
          (type === ComponentTypes.WATER_COOLING &&
```

```
            selectedComponents[ComponentTypes.AIR_COOLING])
          );
          return (
            <ConfiguratorItem
              key={type}
              disabled={isDisabled}
              searchValue={searchValues[type]}
              onSearchChange={handleSearchChange(type)}
              name={name}
              type={type}
              data={components[type].data}
              totalPages={components[type].totalPages}
              onPageChange={(newPage) => handlePageChange(type, newPage)}
              selected={selectedComponents[type]}
              onSelect={(component) => dispatch(selectComponent({ type, component }))}
              isOpen={openedItems[type]}
              onToggle={() => handleToggleItem(type)}
            />
          );
        })}
      </div>
    </div>
  );
};
export default Configurator;
```

КОМПОНЕНТ ЕЛЕМЕНТУ КОНФІГУРАТОРА

```
import { motion } from "framer-motion";
import { useSelector } from "react-redux";
import { useAppDispatch } from "@/redux/store";
import cn from "classnames";
import styles from "./ConfiguratorItem.module.scss";
import Button from "../Button";
import SearchBar from "../Search";
import ComponentsList from "../ComponentsList";
import Counter from "../Counter";
import { configureSelector } from "@/redux/slices/configure/selectors";
import { countingTypes, updateCount } from "@/redux/slices/configure/slice";
import { TComponent } from "@/@types/components";
import { ComponentTypes } from "@/@types/enums";
import { ArrowDropDown, Close } from "@mui/icons-material";
type TConfiguratorItemProps<T extends TComponent> = {
  name: string;
  disabled?: boolean;
  searchValue?: string;
  onSearchChange: (value: string | undefined) => void;
  data: T[];
  type: ComponentTypes;
  totalPages: number;
  selected: T | null;
  onPageChange: (page: number) => void;
  onSelect: (item: T | null) => void;
```

```
  isOpen: boolean;
  onToggle: () => void;
};
const ConfiguratorItem: React.FC<TConfiguratorItemProps<TComponent>> = ({
  name,
  onSearchChange,
  searchValue,
  data,
  totalPages,
  selected,
  onPageChange,
  onSelect,
  isOpen,
  type,
  disabled,
  onToggle,
}) => {
  const dispatch = useAppDispatch();
  const { selectedComponents } = useSelector(configureSelector);
  const variants = {
    hidden: { opacity: 0, maxHeight: 0, transition: { duration: 0.5 } },
    visible: { opacity: 1, maxHeight: 1200, transition: { duration: 0.8 } },
  };
  return (
    <div className={cn(styles.item, disabled ? styles.disabled : "")}>
      <div className={styles.item__wrapper}>
        <div className={styles.item__head} onClick={() => !disabled && onToggle()}>
          <p className={styles.item__head__title}>{name}</p>
          <div className={styles.item__head__selected}>
            {selected && (
              <>
                <img src={selected?.image} />
                <p>{selected?.name}</p>
                {countingTypes.includes(type) && (
                  <Counter
                    count={selectedComponents[type]?.count || 1}
                    setCount={(newCount: number) =>
                      dispatch(updateCount({ type, count: newCount }))
                    }
                  />
                )}
              </>
            )}
          </div>
          <Button
            className={styles.item__head__close}
            onClick={(e) => {
              e.stopPropagation();
              if (selected) {
                onSelect(null);
              } else {
                onToggle();
```

```
            }
          }}
          variant="iconary"
          variantColor={"default"}
        >
          {selected ? (
            <Close />
          ) : (
            <ArrowDropDown
              className={cn(styles.item__head__close__arrow, { [styles.open]: isOpen })}
            />
          )}
        </Button>
      </div>
      <motion.div
        className={styles.item__content}
        initial="hidden"
        variants={variants}
        animate={isOpen ? "visible" : "hidden"}
        style={{ overflow: "hidden" }}
      >
        <div className={styles.item__content__head}>
          <SearchBar value={searchValue} onChange={onSearchChange} />
        </div>
        <ComponentsList
          data={data}
          totalPages={totalPages}
          onPageChange={onPageChange}
          selected={selected?.id}
          onSelect={(item) => {
            onSelect(item);
            onToggle();
          }}
        />
      </motion.div>
    </div>
  </div>
 );
};
export default ConfiguratorItem;
```

СПИСОК КОМПЛЕКТУЮЧИХ

```
import { Pagination } from "@mui/material";
import useMediaQuery from "@mui/material/useMediaQuery";
import styles from "./ComponentsList.module.scss";
import ComponentsListItem from "../ComponentsListItem";
import EmptyState from "../EmptyState";
import { TComponent } from "@/@types/components";
type TComponentsListProps<T extends TComponent> = {
 data: T[];
 totalPages: number;
 selected: number | null | undefined;
```

```
  onPageChange: (page: number) => void;
  onSelect: (item: T) => void;
};
const ComponentsList: React.FC<TComponentsListProps<TComponent>> = ({
  data,
  totalPages,
  onPageChange,
  onSelect,
  selected,
}) => {
  const isMobile = useMediaQuery("(max-width:480px)");
  return (
    <div className={styles.list}>
      <div className={styles.list__wrapper}>
        <div className={styles.list__head}>
          <p>Назва</p>
          <p>Продуктивність</p>
          <div />
        </div>
        <div className={styles.list__main}>
          {data?.length > 0 ? (
            data.map((i, idx) => (
              <ComponentsListItem key={idx} selected={selected} data={i} onSelect={onSelect} />
            ))
          ) : (
            <EmptyState />
          )}
        </div>
        {totalPages > 1 && (
          <Pagination
            count={totalPages}
            onChange={(_, val) => onPageChange(val)}
            siblingCount={isMobile ? 0 : 1}
            boundaryCount={isMobile ? 1 : 2}
          />
        )}
      </div>
    </div>
  );
};
export default ComponentsList;
```

ЕЛЕМЕНТ СПИСКУ КОМПЛЕКТУЮЧИХ

```
import { Box } from "@mui/material";
import { useState } from "react";
import styles from "./ComponentsListItem.module.scss";
import Button from "../Button";
import PreviewComponent from "../PreviewComponent";
import { TComponent } from "@/@types/components";
import { ComponentTypes } from "@/@types/enums";
import { Add, Check } from "@mui/icons-material";
```

```
type TComponentsListItem<T extends TComponent> = {
  data: T;
  onSelect?: (item: T) => void;
  selected?: number | null | undefined;
  mode?: "view" | "select";
};
const ComponentsListItem: React.FC<TComponentsListItem<TComponent>> = ({
  data,
  onSelect,
  selected,
  mode = "select",
}) => {
  const [isPreviewOpened, setIsPreviewOpened] = useState<boolean>(false);
  return (
    <Box className={styles.item__wrapper}>
      <Box className={styles.item}>
        <Box onClick={() => setIsPreviewOpened(true)} className={styles.item__title}>
          <img src={data?.image} />
          <p>{data?.name}</p>
        </Box>
        {mode === "select" && (
          <>
            <p className={styles.item__rating}>
              {data?.type === ComponentTypes.CPU || data?.type === ComponentTypes.GPU
                ? data?.performance
                : "—"}
            </p>
            <Box sx={{ position: "relative" }}>
              <Button
                onClick={() => onSelect && onSelect(data)}
                className={styles.item__add}
                variantColor={data.id === selected ? "success" : "default"}
                variant="iconary"
              >
                {data.id === selected ? <Check /> : <Add />}
              </Button>
              {data.isCompatible && data.id !== selected && (
                <Box
                  sx={{
                    position: "absolute",
                    top: -8,
                    right: -6,
                    display: "flex",
                    alignItems: "center",
                    justifyContent: "center",
                    backgroundColor: "success.main",
                    width: 18,
                    height: 18,
                    borderRadius: "100%",
                  }}
                >
                  <Check sx={{ color: "#000000", width: 16, height: 16 }} />
```

```
            </Box>
          )}
        </Box>
      </>
    )}
  </Box>
  <PreviewComponent
    open={isPreviewOpened}
    data={data}
    handleClose={() => setIsPreviewOpened(false)}
  />
 </Box>
 );
};
export default ComponentsListItem;
```

ВІКНО ПЕРЕГЛЯДУ КОМПЛЕКТУЮЧОГО

```
import { alpha, Box, Drawer, List, ListItem, ListItemText, Stack, Typography } from
"@mui/material";
import styles from "./PreviewComponent.module.scss";
import Button from "../Button";
import { TComponent } from "@/@types/components";
import { ComponentTypes } from "@/@types/enums";
import { Close } from "@mui/icons-material";
type TPreviewComponentProps<T extends TComponent> = {
 open: boolean;
 handleClose: () => void;
 data: T;
};
const PreviewComponent: React.FC<TPreviewComponentProps<TComponent>> = ({
 open,
 data,
 handleClose,
}) => {
 const componentFields = [
  ...(data.type === ComponentTypes.CPU
   ? [
      { label: "Лінійка", value: data.ruler },
      { label: "Роз'єм процесора (Socket)", value: data.socket },
      { label: "Кількість ядер", value: `${data.cores} ядер` },
      { label: "Кількість потоків", value: data.threads },
      { label: "Тактова частота процесора", value: `${data.frequency} GHz` },
      { label: "Максимальна тактова частота", value: `${data.max_frequency} GHz` },
      { label: "Об'єм кешу", value: `${(data.cache_l3 / 1024).toFixed(0)} MB` },
      { label: "Кодова назва мікроархітектури", value: data.architecture_code_name },
      { label: "Назва графічного ядра", value: data.graphic_core_name },
      { label: "Підтримка PCIe", value: data.support_pcie },
      { label: "Розблокований множник", value: `${data.is_unlocked_multiplier ? "+" : "-"}` },
      { label: "Тип пам'яті", value: `${data.memory_type}: ${data.memory_frequency} MHz`
},
      { label: "Термопакет", value: `${data.tdp} Вт` },
      { label: "Продуктивність", value: data.performance },
```

```
        ]
      : []),
    ...(data.type === ComponentTypes.MOTHERBOARD
      ? [
          { label: "Роз'єм процесора (Socket)", value: data.socket },
          { label: "Чипсет", value: data.chipset },
          { label: "Тип пам'яті", value: data.memory_type },
          { label: "Кількість слотів для пам'яті", value: data.count_memory_slots },
          { label: "Кількість каналів пам'яті", value: data.count_memory_channels },
          { label: "Максимальний обсяг пам'яті", value: `${data.max_memory} GB` },
          { label: "Мінімальна частота пам'яті", value: `${data.min_frequency_memory} MHz` },
          { label: "Максимальна частота пам'яті", value: `${data.max_frequency_memory} MHz`
},
          { label: "LAN", value: data.lan },
          { label: "Кількість SATA", value: data.count_sata },
          { label: "Підтримка PCIe 16x", value: data.support_pcie16x ? "Так" : "Ні" },
          { label: "Кількість USB 2.0", value: data.count_usb_v2 },
          { label: "Кількість USB 3.0", value: data.count_usb_v3 },
          { label: "Кількість M.2 слотів", value: data.count_m2_slots },
          { label: "Підтримка PCIe M.2 v3", value: data.support_pcie_m2_v3 ? "Так" : "Ні" },
          { label: "Форм-фактор", value: data.form_factor },
        ]
      : []),
    ...(data.type === ComponentTypes.GPU
      ? [
          { label: "Об'єм пам'яті", value: `${data.memory} MB` },
          { label: "Шина пам'яті", value: data.bus },
          { label: "Тип пам'яті", value: data.type_memory },
          { label: "Серія", value: data.series },
          { label: "Частота пам'яті", value: `${data.memory_frequency} MHz` },
          { label: "Частота відеоядра", value: `${data.video_frequency} MHz` },
          { label: "Максимальна роздільна здатність", value: data.max_resolution },
          { label: "Інтерфейс", value: data.interface },
          { label: "Кількість вентиляторів", value: data.count_fans },
          { label: "Підтримка стандартів", value: data.support_standard },
          { label: "Підтримка CUDA", value: data.support_cuda ? "Так" : "Ні" },
          { label: "Кількість ядер CUDA", value: data.cuda_cores },
          { label: "Довжина", value: `${data.length} мм` },
          { label: "Висота", value: `${data.height} мм` },
          { label: "TDP", value: `${data.tdp} Вт` },
          { label: "Додатковий роз'єм живлення", value: data.additional_connector },
          { label: "Підтримка дисплеїв", value: `${data.support_screens} монітори` },
          { label: "Продуктивність", value: data.performance },
        ]
      : []),
    ...(data.type === ComponentTypes.RAM
      ? [
          { label: "Тип пам'яті", value: data.memory_type },
          { label: "Об'єм пам'яті", value: `${data.memory_capacity} GB` },
          { label: "Кількість модулів", value: data.count_modules },
          { label: "Частота пам'яті", value: `${data.frequency} MHz` },
          { label: "Пропускна здатність", value: `${data.passing_capacity} GB/s` },
```

```
      { label: "Затримка (CL)", value: data.latency },
      { label: "Таймінги", value: data.timings },
      { label: "Напруга", value: `${data.voltage} В` },
    ]
  : []),
...(data.type === ComponentTypes.PSU
  ? [
      { label: "Потужність", value: `${data.power} Вт` },
      { label: "Сертифікація", value: data.certificate },
      { label: "Форм-фактор", value: data.form_factor },
    ]
  : []),
...(data.type === ComponentTypes.AIR_COOLING
  ? [
      { label: "Висота кулера", value: `${data.height} мм` },
      { label: "Сумісність з Socket", value: data.socket.split("/").join(", ") },
      { label: "Кількість охолоджуючих трубок", value: data.count_heat_pipes },
      { label: "Швидкість обертання", value: `${data.speed} об/хв` },
      { label: "Максимальний рівень шуму", value: `${data.noise_level} дБ` },
    ]
  : []),
...(data.type === ComponentTypes.WATER_COOLING
  ? [
      { label: "Діаметр", value: `${data.diameter} мм` },
      { label: "Сумісність з Socket", value: data.socket.split("/").join(", ") },
      { label: "Кількість вентиляторів", value: data.count_fans },
      { label: "Швидкість обертання", value: `${data.speed} об/хв` },
      { label: "Швидкість помпи", value: `${data.pump_speed} об/хв` },
      { label: "Підключення", value: data.connector },
      { label: "Максимальний рівень шуму", value: `${data.noise_level} дБ` },
    ]
  : []),
...(data.type === ComponentTypes.SSD
  ? [
      { label: "Об'єм пам'яті", value: data.memory },
      { label: "Форм-фактор", value: data.form_factor },
      { label: "Інтерфейс", value: data.interface },
      { label: "Максимальна швидкість читання", value: `${data.read_speed} МБ/с` },
      { label: "Максимальна швидкість запису", value: `${data.write_speed} МБ/с` },
      { label: "Ресурс TBW", value: `${data.tbw} ТБ` },
    ]
  : []),
...(data.type === ComponentTypes.HDD
  ? [
      { label: "Об'єм пам'яті", value: data.memory },
      { label: "Форм-фактор", value: data.form_factor },
      { label: "Максимальна швидкість", value: `${data.speed} МБ/с` },
      { label: "Швидкість обертання шпинделя", value: `${data.rotating_speed} об/хв` },
      { label: "Інтерфейс", value: data.interface },
      { label: "Максимальний рівень шуму", value: `${data.noise_level} дБ` },
    ]
  : []),
```

```
    ...(data.type === ComponentTypes.CASE
      ? [
          { label: "Форм-фактор", value: data.form_factor_motherboard.split("/").join(", ") },
          { label: "Можливість встановлення СВО", value: `${data.is_support_water_cooling}
мм` },
          { label: "Максимальна висота кулера", value: `${data.max_height_air_cooling} мм` },
          { label: "Кількість 3.5″ внутрішніх відсіків ", value: data.hdd_slots },
          { label: "Максимальна довжина відеокарти", value: `${data.max_length_gpu} мм` },
          { label: "Розміри", value: data.sizes },
        ]
      : []),
  ];
  return (
    <Drawer
      open={open}
      onClose={handleClose}
      sx={{ "& .MuiDrawer-paper": { width: 600, maxWidth: "100%" }, zIndex: 9999 }}
    >
      <Stack
        className={styles.component}
        px={3}
        py={5}
        direction="column"
        gap={2.5}
        alignItems="center"
      >
        <Box sx={{ position: "absolute", top: 16, right: 16 }}>
          <Button onClick={handleClose} variant="iconary" variantColor={"default"}>
            <Close />
          </Button>
        </Box>
        <Box className={styles.component__image}>
          <img src={data.image} />
        </Box>
        <Typography fontSize={20}>{data.name}</Typography>
        <List className={styles.component__specs} sx={{ width: "100%" }}>
          {componentFields.map((field, index) => (
            <ListItem
              key={index}
              sx={{
                py: 1,
                display: "flex",
                gap: 2,
                borderBottom: 1,
                borderColor: alpha("#ffffff", 0.1),
              }}
            >
              <ListItemText
                primary={field.label}
                sx={{ flex: "1 1 50%", textAlign: "left", fontWeight: 700 }}
              />
              <ListItemText
```

```
              primary={!field.value?.toString().includes("null") ? field.value : "-"}
              sx={{ flex: "1 1 50%", textAlign: "left" }}
            />
          </ListItem>
        ))}
      </List>
    </Stack>
  </Drawer>
 );
};
export default PreviewComponent;
```

ПРОГРЕС ЗБІРКИ

```
import { alpha, CircularProgress } from "@mui/material";
import { useState } from "react";
import { useSelector } from "react-redux";
import styles from "./ConfigureProgress.module.scss";
import Button from "../Button";
import PreviewConfigure from "../PreviewConfigure";
import { configureSelector } from "@/redux/slices/configure/selectors";
import { ComponentTypes } from "@/@types/enums";
import { Visibility } from "@mui/icons-material";
const ConfigureProgress: React.FC = () => {
 const { selectedComponents } = useSelector(configureSelector);
 const [isPreviewOpened, setIsPreviewOpened] = useState<boolean>(false);
 const totalComponents = Object.keys(selectedComponents).length;
 let countSelectedComponents = Object.values(selectedComponents).filter(
  (value) => value !== null
 ).length;
 const hasStorage = Boolean(
  selectedComponents[ComponentTypes.SSD] || selectedComponents[ComponentTypes.HDD]
 );
 const hasCooling = Boolean(
  selectedComponents[ComponentTypes.AIR_COOLING] ||
   selectedComponents[ComponentTypes.WATER_COOLING]
 );
 if (selectedComponents[ComponentTypes.SSD] &&
selectedComponents[ComponentTypes.HDD]) {
  countSelectedComponents--;
 }
 const requiredComponents = totalComponents - (hasStorage ? 1 : 0) - (hasCooling ? 1 : 0);
 const completionPercentage = Math.round((countSelectedComponents / requiredComponents)
* 100);
 return (
  <div className={styles.progress}>
   <h2>Прогрес збірки</h2>
   <div className={styles.progress__bar}>
    <CircularProgress
     color="success"
     size={200}
     variant="determinate"
     value={completionPercentage}
```

```
        />
        <CircularProgress
          sx={{
            color: alpha("#ffffff", 0.1),
            position: "absolute",
            top: 0,
            left: 0,
          }}
          size={200}
          variant="determinate"
          value={100}
        />
        <p className={styles.progress__bar__percentage}>{completionPercentage}%</p>
      </div>
      <Button
        disabled={completionPercentage < 100}
        onClick={() => setIsPreviewOpened(true)}
        variant="primary"
      >
        <Visibility />
        Переглянути конфігурацію
      </Button>
      <PreviewConfigure isOpened={isPreviewOpened} handleClose={() =>
setIsPreviewOpened(false)} />
    </div>
  );
};
export default ConfigureProgress;
```

ТИПІЗАЦІЯ КОМПЛЕКТУЮЧИХ

```
import { ComponentTypes } from "./enums";
export type TGetComponentsParams = {
  type: ComponentTypes;
  limit?: number;
  page?: number;
  search?: string;
};
export type TComponentDefault = {
  id: number;
  name: string;
  image: string;
  isCompatible: boolean;
  type: ComponentTypes;
};
export type TCPU = TComponentDefault & {
  ruler: string;
  socket: string;
  cores: number;
  threads: number;
  frequency: number;
  max_frequency: number;
  cache_l3: number;
```

```
  architecture_code_name: string;
  micro_architecture: string;
  graphic_core_name: string;
  support_pcie: string;
  is_unlocked_multiplier: number;
  memory_type: string;
  memory_frequency: number;
  tdp: number;
  performance: number;
  max_memory: number;
};
export type TMotherboard = TComponentDefault & {
  socket: string;
  chipset: string;
  memory_type: string;
  count_memory_slots: number;
  count_memory_channels: number;
  max_memory: number;
  min_frequency_memory: number;
  max_frequency_memory: number;
  lan: string;
  count_sata: number;
  support_pcie16x: number;
  count_usb_v2: number;
  count_usb_v3: number;
  count_m2_slots: number;
  support_pcie_m2_v3: number;
  form_factor: string;
};
export type TGPU = TComponentDefault & {
  memory: string;
  bus: string;
  type_memory: string;
  series: string;
  memory_frequency: string;
  video_frequency: string;
  max_resolution: string;
  performance: number;
  interface: string;
  count_fans: number;
  support_standard: string;
  support_cuda: number;
  cuda_cores: number;
  length: number;
  height: number;
  tdp: number;
  additional_connector: string;
  support_screens: number;
};
export type TRAM = TComponentDefault & {
  memory_type: string;
  memory_capacity: number;
```

```
  count_modules: number;
  frequency: number;
  passing_capacity: number;
  latency: string;
  timings: string;
  voltage: string;
};
export type TPowerSupply = TComponentDefault & {
  form_factor: string;
  power: number;
  certificate: string;
};
export type TAirCooling = TComponentDefault & {
  socket: string;
  diameter: number;
  type: string;
  connector: string;
  speed: string;
  max_tdp: number;
  noise_level: number;
  count_heat_pipes: number;
  height: number;
  sizes: string;
};
export type TWaterCooling = TComponentDefault & {
  type: string;
  socket: string;
  diameter: number;
  count_fans: number;
  speed: string;
  noise_level: number;
  connector: string;
  pump_speed: string;
  sizes: string;
};
export type TSSD = TComponentDefault & {
  form_factor: string;
  memory: string;
  type_memory: string;
  interface: string;
  is_nvme: number | null;
  read_speed: number;
  write_speed: number;
  tbw: number;
};
export type THDD = TComponentDefault & {
  form_factor: string;
  memory: string;
  interface: string;
  speed: number;
  rotating_speed: number;
  noise_level: number;
```

```
};
export type TCase = TComponentDefault & {
  type: string;
  form_factor_motherboard: string;
  is_support_water_cooling: number;
  max_height_air_cooling: number;
  hdd_slots: string;
  max_length_gpu: number;
  sizes: string;
};
export type TComponent =
  | (TCPU & { type: ComponentTypes.CPU })
  | (TMotherboard & { type: ComponentTypes.MOTHERBOARD })
  | (TGPU & { type: ComponentTypes.GPU })
  | (TRAM & { type: ComponentTypes.RAM })
  | (TPowerSupply & { type: ComponentTypes.PSU })
  | (TAirCooling & { type: ComponentTypes.AIR_COOLING })
  | (TWaterCooling & { type: ComponentTypes.WATER_COOLING })
  | (TSSD & { type: ComponentTypes.SSD })
  | (THDD & { type: ComponentTypes.HDD })
  | (TCase & { type: ComponentTypes.CASE });
```

ВІКНО ПЕРЕВІРКИ СУМІСНОСТІ

```
import { Modal } from "@mui/material";
import { useState, useEffect } from "react";
import styles from "./AnalyzeCompabilityModal.module.scss";
import Button from "../Button";
import {
  isCompatibleAirCoolingAndCPU,
  isCompatibleCase,
  isCompatibleCPUAndMotherboard,
  isCompatibleMotherboardAndSSD,
  isCompatiblePowerSupplyAndGPUAndCPU,
  isCompatibleRAMAndCPU,
  isCompatibleRAMAndMotherboard,
  isCompatibleWaterCoolingAndCPU,
} from "@/utils/compabilityUtils";
import { CheckCircle, Close, Error } from "@mui/icons-material";
import { TModalProps } from "@/@types/base";
import { TConfiguration } from "@/@types/configuration";
type TAnalyzeCompabilityProps = TModalProps & { data: TConfiguration };
const AnalyzeCompabilityModal: React.FC<TAnalyzeCompabilityProps> = ({
  isOpened,
  handleClose,
  data,
}) => {
  const { cpu, motherboard, ram, gpu, psu, air_cooling, water_cooling, ssd, case: pcCase } =
data;
  const [componentsStatus, setComponentsStatus] = useState<
    Record<string, { success: boolean; message: string }>
  >({});
```

```
  const analysisPoints = [
    {
      name: "Сумісність процесора та материнської плати",
      checkFunction: () => isCompatibleCPUAndMotherboard(cpu, motherboard),
    },
    {
      name: "Сумісність ОЗП та процесора",
      checkFunction: () => isCompatibleRAMAndCPU(ram, cpu),
    },
    {
      name: "Сумісність ОЗП та материнської плати",
      checkFunction: () => isCompatibleRAMAndMotherboard(ram, motherboard),
    },
    {
      name: "Сумісність блоку живлення",
      checkFunction: () => isCompatiblePowerSupplyAndGPUAndCPU(psu, gpu, cpu),
    },
    ...(air_cooling?.id
      ? [
          {
            name: "Сумісність повітряного охолодження та процесора",
            checkFunction: () => isCompatibleAirCoolingAndCPU(air_cooling, cpu),
          },
        ]
      : []),
    ...(water_cooling?.id
      ? [
          {
            name: "Сумісність водяного охолодження та процесора",
            checkFunction: () => isCompatibleWaterCoolingAndCPU(water_cooling, cpu),
          },
        ]
      : []),
    ...(ssd?.id
      ? [
          {
            name: "Сумісність материнської плати та SSD",
            checkFunction: () => isCompatibleMotherboardAndSSD(motherboard, ssd),
          },
        ]
      : []),
    {
      name: "Сумісність корпусу з компонентами",
      checkFunction: () => isCompatibleCase(pcCase, motherboard, gpu, air_cooling,
water_cooling),
    },
  ];
  useEffect(() => {
    if (!isOpened) return;
    const initialStatus: Record<string, { success: boolean; message: string }> = {};
    analysisPoints.forEach(({ name, checkFunction }) => {
      const result = checkFunction();
```

```
      initialStatus[name] = {
        success: result.success,
        message: result.message,
      };
    });
    setComponentsStatus(initialStatus);
  }, [data, isOpened]);
  return (
    <Modal open={isOpened} onClose={handleClose}>
      <div className={styles.modal}>
        <div className={styles.modal__wrapper}>
          <div className={styles.modal__head}>
            <h3>Аналіз на сумісність</h3>
            <Button variant="iconary" onClick={handleClose}>
              <Close />
            </Button>
          </div>
          <div className={styles.modal__main}>
            <ul className={styles.modal__main__list}>
              {Object.entries(componentsStatus).map(([key, { success, message }]) => (
                <li className={styles.modal__main__list__item} key={key}>
                  <div className={styles.modal__main__list__item__head}>
                    {success ? (
                      <CheckCircle color="success" />
                    ) : (
                      <Error sx={{ color: "#d32f2f" }} />
                    )}
                    <p>{key}</p>
                  </div>
                  <p>{message}</p>
                </li>
              ))}
            </ul>
          </div>
        </div>
      </div>
    </Modal>
  );
};
export default AnalyzeCompabilityModal;
```

ВІКНО АНАЛІЗУ ПРОДУКТИВНОСТІ

```
import { useEffect, useState } from "react";
import { Modal } from "@mui/material";
import styles from "./AnalyzePerformanceModal.module.scss";
import Button from "../Button";
import { TModalProps } from "@/@types/base";
import { TComponent } from "@/@types/components";
import { TConfiguration } from "@/@types/configuration";
import { isCPU, isGPU } from "@/utils/typeGuards";
import { Close } from "@mui/icons-material";
```

```
const getUsageRecommendation = (totalPerformance: number) => {
  if (totalPerformance < 20000) {
    return {
      recommendation:
        "Цей пристрій підходить для навчання, роботи з офісними програмами та базових
задач.",
      suitableFor: ["Навчання", "Офісні програми", "Базові задачі"],
      exampleApplications: [
        "Microsoft Word, Excel, Google Docs",
        "Перегляд відео на YouTube",
        "Прості ігри (наприклад, Minecraft, старі ігри)",
      ],
    };
  } else if (totalPerformance >= 20000 && totalPerformance < 40000) {
    return {
      recommendation:
        "Підходить для легких ігор, роботи з мультимедіа та багатозадачності. Можливо, деякі
ігри на середніх налаштуваннях.",
      suitableFor: ["Навчання", "Легкі ігри", "Багатозадачність", "Мультимедіа"],
      exampleApplications: [
        "Мультимедійні редактори (наприклад, Adobe Photoshop, Premiere Pro)",
        "Ігри на середніх налаштуваннях (наприклад, The Witcher 3, GTA V)",
        "Багатозадачність (відкриття кількох вкладок браузера, робота з офісними
програмами)",
      ],
    };
  } else if (totalPerformance >= 40000 && totalPerformance < 85000) {
    return {
      recommendation:
        "Хороша система для більшості ігор на високих налаштуваннях, стрімінгу та
професійних задач, таких як відеомонтаж і 3D моделювання.",
      suitableFor: [
        "Ігри на високих налаштуваннях",
        "Стрімінг",
        "Професійні задачі",
        "3D моделювання",
      ],
      exampleApplications: [
        "Ігри на високих налаштуваннях (наприклад, Cyberpunk 2077, Red Dead Redemption
2)",
        "Стрімінг на платформах (Twitch, YouTube)",
        "Відеомонтаж і рендеринг (Adobe Premiere, Final Cut Pro)",
        "3D моделювання і рендеринг (Blender, Autodesk Maya)",
      ],
    };
  } else {
    return {
      recommendation:
        "Ця конфігурація підходить для ресурсоємних ігор на ультра налаштуваннях,
стрімінгу, а також роботи з відеомонтажем і 3D графікою.",
      suitableFor: [
        "Ігри на ультра налаштуваннях",
```

```
        "Стрімінг",
        "Професійні задачі",
        "Відеомонтаж",
        "3D графіка",
      ],
      exampleApplications: [
        "Ігри на ультра налаштуваннях (наприклад, Metro Exodus, Battlefield V)",
        "Стрімінг з максимальними налаштуваннями якості",
        "Професійні задачі (3D рендеринг, відеомонтаж з високою швидкістю)",
        "Використання віртуальних машин і багатозадачність на максимальних
налаштуваннях",
      ],
    };
  }
};
const calculatePowerConsumption = (cpu: TComponent | null, gpu: TComponent | null) => {
  if (!cpu || !gpu || !isCPU(cpu) || !isGPU(gpu)) return 0;
  const cpuPower = cpu?.tdp || 0;
  const gpuPower = gpu?.tdp || 0;
  const totalPower = cpuPower + gpuPower;
  return totalPower;
};
type TAnalyzePerformanceModalProps = TModalProps & {
  totalPerformance: number;
  data: TConfiguration;
};
const AnalyzePerformanceModal: React.FC<TAnalyzePerformanceModalProps> = ({
  isOpened,
  handleClose,
  totalPerformance,
  data,
}) => {
  const { cpu, gpu, ram, ssd } = data;
  const [recommendation, setRecommendation] = useState<string>("");
  const [suitableFor, setSuitableFor] = useState<string[]>([]);
  const [exampleApplications, setExampleApplications] = useState<string[]>([]);
  const [powerConsumption, setPowerConsumption] = useState<number>(0);
  useEffect(() => {
    if (cpu && gpu && ram && ssd) {
      const { recommendation, suitableFor, exampleApplications } =
        getUsageRecommendation(totalPerformance);
      setRecommendation(recommendation);
      setSuitableFor(suitableFor);
      setExampleApplications(exampleApplications);

      const power = calculatePowerConsumption(cpu, gpu);
      setPowerConsumption(power);
    }
  }, [data]);
  return (
    <Modal open={isOpened} onClose={handleClose}>
      <div className={styles.modal}>
```

```
      <div className={styles.modal__wrapper}>
        <div className={styles.modal__head}>
          <h3>Загальний аналіз продуктивності</h3>
          <Button variant="iconary" onClick={handleClose}>
            <Close />
          </Button>
        </div>
        <div className={styles.modal__main}>
          <div className={styles.modal__main__summary}>
            <p>
              <span>Оцінка загальної продуктивності (PassMark):</span> {totalPerformance}
бали
            </p>
            <p>
              <span>Опис:</span> {recommendation}
            </p>
            <p>
              <span>Приклад застосунків:</span>
            </p>
            <ul>
              {exampleApplications.map((app, index) => (
                <li key={index}>{app}</li>
              ))}
            </ul>
          </div>
          <div className={styles.performanceBreakdown}>
            <h3>Розбиття по компонентам</h3>
            <ul>
              <li>
                {cpu?.name}, Оцінка продуктивності: {cpu?.performance}
              </li>
              <li>
                {gpu?.name}, Оцінка продуктивності: {gpu?.performance}
              </li>
              <li>
                Оперативна пам'ять: {ram?.memory_capacity} ГБ, Пропускна здатність: 
                {ram?.passing_capacity} МБ/с
              </li>
              <li>
                SSD: {ssd?.form_factor}, Швидкість читання: {ssd?.read_speed} МБ/с
              </li>
            </ul>
          </div>
          <div className={styles.powerConsumption}>
            <h3>Прогноз споживання потужності</h3>
            <p>Оцінкове споживання потужності: {powerConsumption} Вт</p>
            <p>
              Рекомендується вибрати БП з потужністю, що на 20-30% перевищує сумарне
споживання.
            </p>
          </div>
          <div className={styles.performanceSuggestions}>
```

```
            <h3>Рекомендації по покращенню</h3>
            {totalPerformance < 20000 && (
              <p>
                Для покращення продуктивності, розгляньте можливість оновлення процесора
та
                відеокарти.
              </p>
            )}
            {totalPerformance >= 20000 && totalPerformance < 40000 && (
              <p>
                Якщо ви плануєте ігри на середніх або високих налаштуваннях або стрімінг,
                рекомендую оновити відеокарту до більш потужної моделі.
              </p>
            )}
            {totalPerformance >= 40000 && totalPerformance < 70000 && (
              <p>
                Для комфортної роботи з більшості сучасних ігор на високих налаштуваннях та
                стрімінгу, можна покращити систему охолодження та збільшити оперативну
пам'ять для
                підвищення багатозадачності.
              </p>
            )}
            {totalPerformance >= 70000 && totalPerformance < 110000 && (
              <p>
                Система готова для найвищих налаштувань у іграх, 3D рендерингу та
професійних
                задач, але для ще більшої ефективності розгляньте можливість покращення SSD
та процесора.
              </p>
            )}
            {totalPerformance >= 110000 && (
              <p>
                Ваша система ідеальна для роботи з найресурсоємнішими іграми на ультра
                налаштуваннях, професійного відеомонтажу та 3D рендерингу без обмежень.
                Можна додавати додаткові компоненти для ще кращої продуктивності.
              </p>
            )}
          </div>
        </div>
      </div>
    </div>
  </Modal>
 );
};
export default AnalyzePerformanceModal;
```

ОСНОВНИЙ ФАЙЛ СЕРВЕРУ

```
import express from "express";
import cors from "cors";
```

```
import cookieParser from "cookie-parser";
import componentsRouter from "./routes/components.js";
import configurationsRouter from "./routes/configurations.js";
import authRouter from "./routes/auth.js";
import userRouter from "./routes/user.js";
const app = express();
const PORT = process.env.PORT || 3000;
const prefix = "/api/v1";
app.use(
  cors({
    origin: "http://localhost:5173",
    credentials: true,
  })
);
app.use(express.json());
app.use(cookieParser());
app.use(`${prefix}/components`, componentsRouter);
app.use(`${prefix}/auth`, authRouter);
app.use(`${prefix}/user`, userRouter);
app.use(`${prefix}/configurations`, configurationsRouter);
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

ПІДКЛЮЧЕННЯ ДО БАЗИ ДАНИХ

```
import mysql from "mysql2/promise";
import dotenv from "dotenv";
dotenv.config();
const { DB_HOST, DB_NAME, DB_PORT, DB_LOGIN, DB_PASS } = process.env;
const pool = mysql.createPool({
  host: DB_HOST,
  user: DB_LOGIN,
  password: DB_PASS,
  database: DB_NAME,
  port: DB_PORT,
  connectTimeout: 60000,
});
export const query = async (sql, params) => {
  const connection = await pool.getConnection();
  try {
    const [results] = await connection.execute(sql, params);
    return results;
  } finally {
    connection.release();
  }
};
```

КОНТРОЛЕР АУТЕНТИФІКАЦІЇ

```
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
import dotenv from "dotenv";
import { query } from "../utils/db.js";
```

```
dotenv.config();
const { JWT_ACCESS_SECRET, JWT_REFRESH_SECRET } = process.env;
export const register = async (req, res) => {
  const { email, password } = req.body;
  const isExist = await query("SELECT * FROM users WHERE email = ?", [email]);
  if (isExist.length) {
    return res.status(400).json({ message: "Користувач з таким email вже існує" });
  }
  const hashedPassword = await bcrypt.hash(password, 10);
  await query("INSERT INTO users (email, password_hash) VALUES (?, ?)", [email,
hashedPassword]);
  await res.status(200).json({ message: "Успішна реєстрація" });
};
export const login = async (req, res) => {
  const { email, password } = req.body;
  const [user] = await query("SELECT id, password_hash FROM users WHERE email = ?",
[email]);
  if (!user || !bcrypt.compareSync(password, user.password_hash)) {
    return res.status(403).json({ message: "Не вірні дані" });
  }
  const accessToken = jwt.sign({ email, userId: user.id }, JWT_ACCESS_SECRET, { expiresIn:
"1h" });
  const refreshToken = jwt.sign({ email, userId: user.id }, JWT_REFRESH_SECRET, {
    expiresIn: "14d",
  });
  res.cookie("accessToken", accessToken, { httpOnly: true, secure: false, sameSite: "Strict" });
  res.cookie("refreshToken", refreshToken, { httpOnly: true, secure: false, sameSite: "Strict" });
  res.status(200).json({ message: "Вхід виконано успішно" });
};
export const refreshTokens = async (req, res) => {
  const refreshToken = req.cookies.refreshToken;
  if (!refreshToken) {
    return res.status(404).json({ message: "Токен відсутній" });
  }
  try {
    const decoded = jwt.verify(refreshToken, JWT_REFRESH_SECRET);
    const accessToken = jwt.sign(
      { email: decoded.email, userId: decoded.userId },
      JWT_ACCESS_SECRET,
      {
        expiresIn: "1h",
      }
    );
    const newRefreshToken = jwt.sign(
      { email: decoded.email, userId: decoded.userId },
      JWT_REFRESH_SECRET,
      {
        expiresIn: "14d",
      }
    );
    res.cookie("accessToken", accessToken, { httpOnly: true, secure: true, sameSite: "Strict" });
    res.cookie("refreshToken", newRefreshToken, {
```

```
      httpOnly: true,
      secure: true,
      sameSite: "Strict",
    });
    res.status(200).json({ message: "Токени оновлені" });
  } catch (err) {
    return res.status(403).json({ message: "Недійсний токен" });
  }
};
export const logout = async (_, res) => {
  res.clearCookie("accessToken", { httpOnly: true, secure: true, sameSite: "Strict" });
  res.clearCookie("refreshToken", { httpOnly: true, secure: true, sameSite: "Strict" });
  res.status(200).json({ message: "Logged out" });
};
```

КОНТРОЛЕР КОМПЛЕКТУЮЧИХ

```
import { query } from "../utils/db.js";
export const getComponents = async (req, res) => {
  const { type, page = 1, limit = 10, search } = req.query;
  const offset = (page - 1) * limit;
  const allowedTables = [
    "motherboard",
    "cpu",
    "gpu",
    "ram",
    "hdd",
    "ssd",
    "case",
    "psu",
    "air_cooling",
    "water_cooling",
  ];
  if (!allowedTables.includes(type)) {
    return res.status(400).json({ message: "Invalid type parameter" });
  }
  const tableName = type === "case" ? "`case`" : type;
  try {
    let queryString = `SELECT * FROM ${tableName}`;
    let queryParams = [];
    if (search) {
      queryString += ` WHERE LOWER(name) LIKE LOWER(?)`;
      queryParams.push(`%${search}%`);
    }
    queryString += ` LIMIT ${parseInt(limit, 10)} OFFSET ${parseInt(offset, 10)}`;
    const data = await query(queryString, queryParams);
    let countQuery = `SELECT COUNT(*) as count FROM ${tableName}`;
    let countParams = [];
    if (search) {
      countQuery += ` WHERE LOWER(name) LIKE LOWER(?)`;
      countParams.push(`%${search}%`);
    }
    const [totalCount] = await query(countQuery, countParams);
```

```
    const totalPages = Math.ceil(totalCount.count / limit);
    return res.status(200).json({ data, totalPages });
  } catch (error) {
    console.error(`Error fetching data from ${type}:`, error);
    return res.status(500).json({ message: "Internal server error" });
  }
};
```

КОНТРОЛЕР КОНФІГУРАЦІЙ

```
import { query } from "../utils/db.js";
export const saveConfiguration = async (req, res) => {
  try {
    const { userId } = req.user;
    const { cpu, motherboard, gpu, ram, psu, air_cooling, water_cooling, ssd, hdd, pcCase } =
      req.body;
    if (!cpu || !motherboard || !gpu || !ram || !ram.count || !psu || !pcCase) {
      return res.status(400).json({ message: "Всі обов'язкові компоненти повинні бути
вибрані!" });
    }
    if ((air_cooling && water_cooling) || (!air_cooling && !water_cooling)) {
      return res
        .status(400)
        .json({ message: "Повинний бути обраний тільки один тип охолодження!" });
    }
    if ((!ssd && !hdd) || (ssd && hdd && (ssd.count <= 0 || hdd.count <= 0))) {
      return res.status(400).json({ message: "Необхідно вибрати хоча б один диск (SSD або
HDD)!" });
    }
    await query(
      `INSERT INTO user_configurations (
          user_id, cpu_id, motherboard_id, gpu_id, ram_id, ram_count, psu_id, air_cooling_id,
          water_cooling_id, ssd_id, ssd_count, hdd_id, hdd_count, case_id
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,
      [
        userId,
        cpu,
        motherboard,
        gpu,
        ram.id,
        ram.count,
        psu,
        air_cooling,
        water_cooling,
        ssd?.id || null,
        ssd?.count || null,
        hdd?.id || null,
        hdd?.count || null,
        pcCase,
      ]
    );
    res.status(200).json({ message: "Конфігурація успішно збережена" });
  } catch (err) {
```

```
    console.log(err);
    res.status(500).json({ message: "Помилка серверу" });
  }
};
export const getConfigurations = async (req, res) => {
  try {
    const { userId } = req.user;
    const configurations = await query(
      `SELECT
        uc.id AS config_id,
        uc.user_id,
        uc.cpu_id,
        uc.motherboard_id,
        uc.ram_id, uc.ram_count,
        uc.gpu_id,
        uc.psu_id,
        uc.air_cooling_id,
        uc.water_cooling_id,
        uc.ssd_id, uc.ssd_count,
        uc.hdd_id, uc.hdd_count,
        uc.case_id
      FROM user_configurations uc
      WHERE uc.user_id = ?`,
      [userId]
    );
    const formattedConfigurations = await Promise.all(
      configurations.map(async (config) => {
        const [cpu, motherboard, ram, gpu, psu, air_cooling, water_cooling, ssd, hdd, pc_case] =
          await Promise.all([
            query(`SELECT * FROM cpu WHERE id = ?`, [config.cpu_id]),
            query(`SELECT * FROM motherboard WHERE id = ?`, [config.motherboard_id]),
            query(`SELECT * FROM ram WHERE id = ?`, [config.ram_id]),
            query(`SELECT * FROM gpu WHERE id = ?`, [config.gpu_id]),
            query(`SELECT * FROM psu WHERE id = ?`, [config.psu_id]),
            query(`SELECT * FROM air_cooling WHERE id = ?`, [config.air_cooling_id]),
            query(`SELECT * FROM water_cooling WHERE id = ?`, [config.water_cooling_id]),
            query(`SELECT * FROM ssd WHERE id = ?`, [config.ssd_id]),
            query(`SELECT * FROM hdd WHERE id = ?`, [config.hdd_id]),
            query(`SELECT * FROM \`case\` WHERE id = ?`, [config.case_id]),
          ]);
        return {
          config_id: config.config_id,
          cpu: { ...cpu[0], type: "cpu" } || {},
          motherboard: { ...motherboard[0], type: "motherboard" } || {},
          ram: { ...ram[0], count: config.ram_count, type: "ram" } || {},
          gpu: { ...gpu[0], type: "gpu" } || {},
          psu: { ...psu[0], type: "psu" } || {},
          air_cooling: { ...air_cooling[0], type: "air_cooling" } || {},
          water_cooling: { ...water_cooling[0], type: "water_cooling" } || {},
          ssd: { ...ssd[0], count: config.ssd_count, type: "ssd" } || {},
          hdd: { ...hdd[0], count: config.hdd_count, type: "hdd" } || {},
          case: { ...pc_case[0], type: "case" } || {},
```

```
      };
    })
  );
  res.status(200).json(formattedConfigurations);
 } catch (err) {
  console.log(err);
  res.status(500).json({ message: "Помилка серверу" });
 }
};
```

КОНТРОЛЕР КОРИСТУВАЧА

```
import bcrypt from "bcryptjs";
import crypto from "crypto";
import nodemailer from "nodemailer";
import { query } from "../utils/db.js";
export const getUser = async (req, res) => {
 try {
  const { email } = req.user;
  if (!email) {
   return res.status(400).json({ message: "Invalid token: missing email" });
  }
  const user = (await query("SELECT id, email FROM users WHERE email = ?", [email]))[0];
  if (!user) {
   return res.status(404).json({ message: "User not found" });
  }
  res.status(200).json(user);
 } catch (err) {
  res.status(500).json({ message: "Помилка серверу" });
 }
};
export const sendResetPasswordLetter = async (req, res) => {
 try {
  const { email } = req.body;
  const { EMAIL, EMAIL_PASS, RESET_PASSWORD_URL } = process.env;
  if (!email) {
   return res.status(404).json({ message: "Не вірні дані" });
  }
  const user = (await query("SELECT * FROM users WHERE email = ?", [email]))[0];
  if (!user) {
   return res.status(404).json({ message: "Користувача не знайдено" });
  }
  const token = crypto.randomBytes(20).toString("hex");
  const expires = new Intl.DateTimeFormat("sv-SE", {
   timeZone: "Europe/Kyiv",
   year: "numeric",
   month: "2-digit",
   day: "2-digit",
   hour: "2-digit",
   minute: "2-digit",
   second: "2-digit",
  })
   .format(new Date(Date.now() + 60 * 60 * 1000))
```

```
    .replace(" ", "T");
  await query("INSERT INTO reset_password_tokens (token, expires, user_id) VALUES (?, ?,
?)", [
    token,
    expires,
    user.id,
  ]);
  const transporter = nodemailer.createTransport({
    service: "gmail",
    host: "smtp.gmail.com",
    port: 465,
    secure: true,
    auth: {
      user: EMAIL,
      pass: EMAIL_PASS,
    },
  });
  const resetPasswordUrl = `${RESET_PASSWORD_URL}/reset-password?token=${token}`;
  const mailOptions = {
    to: email,
    from: "PC Configurator",
    subject: "Запит на скидання пароля",
    html: `
    <table role="presentation" width="100%" cellspacing="0" cellpadding="0" border="0"
      style="background-color: #121212; padding: 20px; text-align: center;">
      <tr>
        <td align="center">
          <table role="presentation" width="500" cellspacing="0" cellpadding="0" border="0"
            style="background-color: #1e1e1e; color: #ffffff; padding: 20px; font-family: Arial,
sans-serif; border-radius: 8px;">
            <tr>
              <td align="center">
                <h2 style="color: #ffcc00;">Скидання пароля</h2>
                <p>Ви отримали цей лист, тому що ви (або хтось інший) надіслали запит на
скидання пароля.</p>
                <p>Будь ласка, натисніть на наступне посилання або вставте його у свій
браузер, щоб завершити процес:</p>
                <p>
                  <a href="${resetPasswordUrl}" style="
                    display: inline-block;
                    background-color: #ffcc00;
                    color: #121212;
                    padding: 12px 20px;
                    text-decoration: none;
                    font-weight: bold;
                    border-radius: 5px;
                    margin-top: 10px;
                  ">Скинути пароль</a>
                </p>
                <p style="margin-top: 20px; font-size: 14px; color: #bbbbbb;">
                  Якщо ви не надсилали запит, проігноруйте цей лист.
                </p>
```

```
            </td>
           </tr>
          </table>
         </td>
        </tr>
       </table>
    `,
    };
    await transporter.sendMail(mailOptions);
    res.status(200).json({ message: "Лист відправлено. Перевірте свій email" });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Помилка серверу" });
  }
};
export const checkResetToken = async (req, res) => {
  try {
    const { token } = req.body;
    if (!token) {
      return res.status(403).json({ message: "Немає доступу" });
    }
    const data = (await query("SELECT * FROM reset_password_tokens WHERE token = ?",
[token]))[0];
    if (!data || new Date(data.expires) < new Date()) {
      return res.status(403).json({ message: "Немає доступу" });
    }
    res.status(201).json({});
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Помилка серверу" });
  }
};
export const resetPassword = async (req, res) => {
  try {
    const { token, password } = req.body;
    if (!token || !password) {
      return res.status(404).json({ message: "Не вірні дані" });
    }
    const {
      user_id,
      token: resetToken,
      expires,
    } = (await query("SELECT * FROM reset_password_tokens WHERE token = ?",
[token]))[0];
    if (!resetToken || new Date(expires) < new Date()) {
      return res.status(404).json({ message: "Час дії сплинув. Спробуйте ще раз" });
    }
    const hashedPassword = await bcrypt.hash(password, 10);
    await query("UPDATE users SET password_hash = ? WHERE id = ?", [hashedPassword,
user_id]);
    await query("DELETE FROM reset_password_tokens WHERE user_id = ?", [user_id]);
    res.status(200).json({ message: "Пароль успішно змінено" });
```

```
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Помилка серверу" });
  }
};
```

ФУНКЦІЯ ПОСЕРЕДНИК ДЛЯ ПЕРЕВІРКИ АВТОРИЗАЦІЇ

```
import jwt from "jsonwebtoken";
export const authMiddleware = (req, res, next) => {
  const token = req.cookies.accessToken;
  if (!token) {
    return res.status(401).json({ message: "Unauthorized" });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWT_ACCESS_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: "Access denied" });
  }
};
```

API МАРШРУТИ ДЛЯ АВТЕНТИФІКАЦІЇ

```
import express from "express";
import * as AuthController from "../controllers/AuthController.js";
import { authMiddleware } from "../middlewares/auth.js";
const router = express.Router();
router.post("/register", AuthController.register);
router.post("/login", AuthController.login);
router.post("/refresh", AuthController.refreshTokens);
router.post("/logout", authMiddleware, AuthController.logout);
export default router;
```

API МАРШРУТИ ДЛЯ КОМПЛЕКТУЮЧИХ

```
import express from "express";
import * as ComponentsController from "../controllers/ComponentsController.js";
const router = express.Router();
router.get("/get", ComponentsController.getComponents);
export default router;
```

API МАРШРУТИ ДЛЯ КОРИСТУВАЧА

```
import express from "express";
import * as UserController from "../controllers/UserController.js";
import { authMiddleware } from "../middlewares/auth.js";
const router = express.Router();
router.get("/get", authMiddleware, UserController.getUser);
router.post("/send-reset-password", UserController.sendResetPasswordLetter);
router.post("/reset-password", UserController.resetPassword);
router.post("/check-reset-token", UserController.checkResetToken);
export default router;
```