Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян
(назва відділення)

Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра
(освітньо-професійного ступеня)

на тему: **Розробка веб-сайту квест-кімнат «Quest rooms»**

Виконав: студент IV курсу, групи КІ-406
Спеціальності 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

_____ Владислав КАЛАШНІКОВ
(ім'я та прізвище)

Керівник Олександра МАРЦЮК
(ім'я та прізвище)

Рецензент _____
(ім'я та прізвище)

Тернопіль – 2026

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

Відділення **інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян**
Циклова комісія **комп'ютерної інженерії**
Освітньо-професійний ступінь **фаховий молодший бакалавр**
Освітньо-професійна програма: **Обслуговування комп'ютерних систем і мереж**
Спеціальність: **123 Комп'ютерна інженерія**
Галузь знань: **12 Інформаційні технології**

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"30" березня 2026 року

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

Калашнікову Владиславу Євгенійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: **Розробка веб-сайту квест-кімнат «Quest rooms»**

керівник роботи Марцюк Олександра Василівна
(прізвище, ім'я, по батькові)

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя» від 27.03.2026р № 4/9-167.

2. Строк подання студентом роботи: 15 червня 2026 року.

3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: Ruby, JavaScript.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Охорона праці та безпека життєдіяльності.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
- структурна схема файлів веб-сайту;
- структурна схема головної сторінки;
- текст програми;
- таблиця техніко-економічних показників.

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Богдана МАРТИНЮК викладач		
Охорона праці та безпека життєдіяльності	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	01.04	
2	Збір і узагальнення інформації	08.05	
3	Написання першого розділу	15.05	
4	Розробка технічного та робочого проекту	22.05	
5	Написання спеціального розділу	28.05	
6	Розрахунок економічної частини	1.06	
7	Написання розділу охорони праці	2.06	
8	Виконання графічної частини	8.06	
9	Оформлення проєкту	10.06	
10	Погодження нормоконтролю	12.06	
11	Попередній захист роботи	13.06	
12	Захист кваліфікаційної роботи9		

7. Дата видачі завдання: 31 березня 2026 року

Студент _____ Владислав КАЛАШНІКОВ
(підпис) (ім'я та прізвище)

Керівник роботи _____ Олександра МАРЦЮК
(підпис) (ім'я та прізвище)

ЗМІСТ

АНОТАЦІЯ 6
ANNOTATION 7
ВСТУП 8
1.ЗАГАЛЬНИЙ РОЗДІЛ 9
1.1 Аналітичний огляд існуючих рішень 9
1.2 Підбір технологій та засобів розробки веб сайту 12
1.3 Технічне завдання на розробку «Quest Rooms» 14
1.3.1 Призначення системи 15
1.3.2 Функціональні вимоги 15
1.3.3 Вимоги до структури та оформлення 15
1.3.4 Техніко–економічне обґрунтування 16
1.3.5 План виконання розробки 17
1.3.6 Порядок перевірки та здачі роботи 18
2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ 19
2.1 Розробка структури, архітектури та маршрутизації веб–сайту «Quest rooms» 19
2.2 Створення компонентної структури інтерфейсу та адаптивна верстка сторінок 21
2.3 Проєктування схеми даних, сутностей та взаємодії з СУБД 25
2.3.1 Взаємодія додатка з СУБД та пулінг підключень 28
2.4 Програмування клієнтської та серверної логіки 29
2.4.1 Реалізація серверних компонентів та API–маршрутів 29
2.4.2 Реалізація клієнтських форм та управління станом бронювання 31
2.4.3 Розробка інтерактивної клієнтської логіки, стану та валідації форм 34
2.4.4 Архітектура головної сторінки панелі адміністратора 37

3 СПЕЦІАЛЬНИЙ РОЗДІЛ 42
3.1 Написання юніт–тестів та інтеграційне тестування компонентів 42
3.2 Інструкція з локального розгортання, оптимізації та деплою 46
3.3 Аналіз якості коду, лінтинг та оптимізація продуктивності веб–сайту 49
4 ЕКОНОМІЧНИЙ РОЗДІЛ 54
4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР 54
4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи 55
4.3 Розрахунок матеріальних витрат 58
4.4 Розрахунок витрат на електроенергію 59
4.5 Розрахунок суми амортизаційних відрахувань 60
4.6 Обчислення накладних витрат 61
4.7 Складання кошторису витрат та визначення собівартості НДР 61
4.8 Розрахунок ціни НДР 62
4.9 Визначення економічної ефективності і терміну окупності капітальних вкладень 64
5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ 65
5.1 Ергономіка робочої пози та профілактика захворювань опорно-рухового апарату розробника програмного забезпечення 65
5.2 Психофізіологічні аспекти праці: вплив стрес-факторів на безпеку розробки в умовах обмеженого часу в ІТ-сфері 68
5.3 Гігієнічне нормування та використання систем штучного освітлення для організації робочого місця веброзробника 71
ВИСНОВКИ 74
ПЕРЕЛІК ПОСИЛАНЬ 75
ДОДАТКИ 76
Додаток А. Код route.ts 76
Додаток Б. Тести findErrors 78

АНОТАЦІЯ

Тема кваліфікаційної роботи: Розробка веб–сайту квест–кімнат «Quest rooms».

Метою даного проекту є розробка веб–сайту квест–кімнат.

Пояснювальна записка складається з 5 розділів.

У загальній частині описуються аналітичний огляд існуючих рішень, підбір технологій та засобів для розробки та аналіз технічного завдання.

У другому розділі представлено процес створення веб сайту, опис та обґрунтування вибору структури та методу організації вхідних та вихідних даних, інформаційних зв'язків, зовнішнє проектування програми, написання кодів веб–сайту.

В спеціальній частині описано процес написання системних тестів. Також розписано інструкцію з розгортання інтернет–магазину на хостингу. Описано тестування якості розроблюваного веб–сайту.

Розрахунок вартості розробки та економічної ефективності приведено в економічній частині.

Основні питання охорони праці та техніки безпеки розглянуто в п'ятому розділі.

Обсяг пояснювальної записки 75 сторінки.

До складу дипломного проекту входить графічна частина, яка складається з структурної схеми головної сторінки, техніко–економічних показників, тексту модуля програми, та структурну схему файлів веб-сайту, що виконані на окремих аркушах формату А1.

ANNOTATION

Qualification Project Topic: Development of a website for escape rooms, “Quest Rooms”.

The purpose of this project is to develop a website for escape rooms.

The explanatory report consists of five sections.

The general section includes an analytical review of existing solutions, the selection of technologies and development tools, and an analysis of the technical requirements.

The second section presents the process of website development, including a description and justification of the chosen structure and methods for organizing input and output data, information relationships, external software design, and the implementation of the website code.

The special section describes the process of writing system tests. It also provides instructions for deploying the website on a hosting platform. In addition, it describes the quality testing of the developed website.

The calculation of development costs and economic efficiency is presented in the economic section.

The main issues of occupational health and safety are discussed in the fifth section.

The explanatory report consists of 75 pages.

The diploma project also includes a graphical section consisting of a structural diagram of the main page, technical and economic indicators, the source code of a software module, and a structural diagram of the website files, each presented on separate A1-format sheets.

ВСТУП

У сучасних умовах цифровізації сфера розваг активно переходить в онлайн. Для успішної роботи квест–кімнат наявність зручного веб–сайту є обов'язковою умовою, оскільки саме через інтернет користувачі шукають нові враження та планують своє дозвілля. Створення якісної платформи для презентації ігрових сценаріїв дозволяє задовольнити попит на цікавий та технологічний відпочинок.

Актуальність теми зумовлена потребою у веб–сайті з інтуїтивно зрозумілим інтерфейсом, де клієнти можуть швидко обрати квест і безпечно забронювати час.

Метою дипломного проєкту є розробка веб сайту квест–кімнат «Quest Rooms». Система повинна забезпечувати: зручний перегляд каталогу квестів, ознайомлення з характеристиками (сюжет, складність, кількість гравців) та онлайн–бронювання ігор для звичайних користувачів. А також для адміністраторів інструменти для керування, редагування описів кімнат та контролю замовлень.

Використання сучасних веб технологій при розробці веб сайту дозволить підвищити його якість та забезпечити конкурентоспроможність. Реалізація сайту стане важливим кроком до автоматизації послуг у сфері розваг та покращення взаємодії з клієнтами в цифровому середовищі.

1. ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Сьогодні сфера розваг в Україні, зокрема індустрія квест–кімнат, перебуває на етапі активної цифрової трансформації. Для сучасного користувача веб сайт є не просто візитівкою, а основним інструментом взаємодії: від вибору сюжету квесту до бронювання ігрового сеансу. Успішність та затребуваність розроблюваного проєкту «Quest Rooms» залежить від того, наскільки ефективно система зможе поєднати інформативність, швидкість роботи та зручність інтерфейсу.

Для визначення функціональних вимог було проведено аналіз трьох популярних майданчиків, що працюють на ринку Західної України.

1. Factoria;
2. Questroom.com.ua;
3. Quest.lviv.ua.

Отже розпочнемо порівняння із веб сайту Factoria. Це сайт великого розважального центру, де квести є лише частиною послуг. Тож він лиш частково має схожий функціонал та аудиторію, відносно розроблюваного веб–сайту.

До сильних сторін можна віднести високоякісний візуальний контент, чіткий поділ на вікові категорії та наявність пакетних пропозицій.

Проте слід зауважити, що інтерфейс перевантажений маркетинговими елементами, що ускладнює шлях користувача безпосередньо до бронювання конкретного квесту. Окрім цього відсутня глибока фільтрація за складністю.

Наступним розглянемо один із найбільших веб–сайтів квестів в Україні Questroom.com.ua.

До сильних сторін цього веб сайту можна віднести:

– Хороша система фільтрації (жанр, кількість гравців, рівень страху, складність).

– За рахунок розмаху проекту має велику база відгуків та рейтингів.

Проте можемо виділити і слабкі сторони, а саме:

– шаблонний дизайн з низьким рівнем унікальності (UI/UX 2010–х років).

– Через велику кількість зовнішніх посилань на різних операторів процес підтвердження бронювання іноді є заплутаним. [11]

Додатково на рисунку 1.1 наведено зовнішній вигляд головної сторінки Questroom.

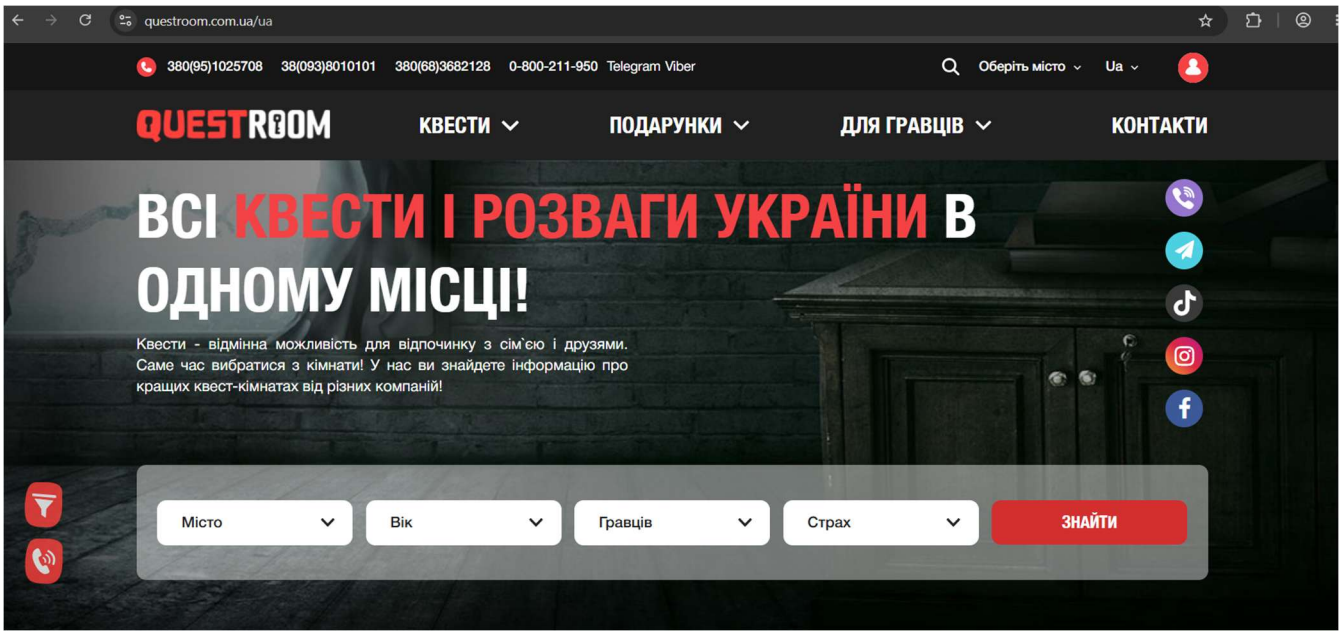

Рисунок 1.1 – Головна сторінка Questroom

Останнім у списку є Quest.lviv.ua, який є сайтом локального оператора з акцентом на атмосферність. [10]

Позитивні сторони це лаконічність, фокус на сюжеті кожного квесту та проста структура навігації, останню наведено на рисунку 1.2.

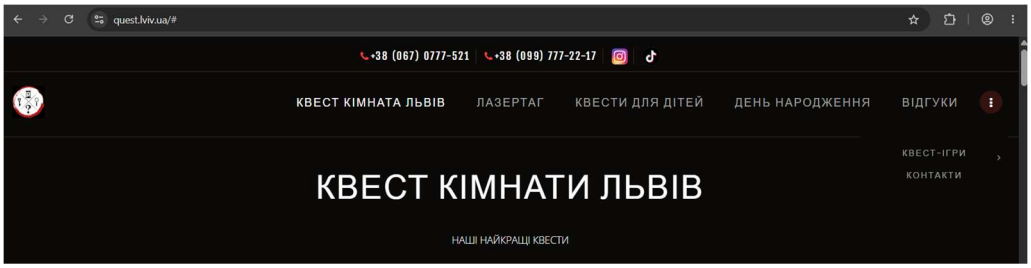

Рисунок 1.2 – Навігація в Quest.lviv

В той ж час є недоліки, які проявляються в обмеженій функціональності (відсутність особистого кабінету), мінімальні можливості фільтрації та відсутність підтримки кількох мов у зручному форматі.

Порівняння існуючих рішень та розроблюваного проєкту наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльне дослідження характеристик веб–ресурсів

Критерій	Factoria	Questroom (Агрегатор)	Quest.lviv.ua	«Quest Rooms» (Проєкт)
Цільова аудиторія	Сім'ї, корпоративи	Широке коло	Молодь, студенти	Універсальна
Система бронювання	Форма зворотного зв'язку	Онлайн–календар (зовнішній)	Онлайн–календар	Інтерактивні слоти + API
Багатомовність (i18n)	Часткова (UA)	Наявна (UA/RU)	Відсутня/Обмежена	Повна підтримка
Особистий кабінет	Відсутній	Відсутній	Відсутній	Наявний
Технологічний стек	Класичний CMS	PHP, Legacy JS	WordPress, jQuery	Next.js 14 , React (SPA)
Візуальний стиль	Рекламний	Інформаційний	Атмосферний	Сучасний Dark Mode Minimalism

Проєкт «Quest Rooms» має на меті усунути недоліки конкурентів, поєднавши атмосферність локальних операторів із функціональною потужністю сучасних вебдодатків. Використання Next.js 14 дозволяє створити «безшовний»

інтерфейс, де перемикання між квестами та мовами інтерфейсу відбувається миттєво.

Основний акцент буде зроблено на:

1. Швидкості завдяки серверному рендерингу;
2. Зручності, а саме впровадження особистого кабінету.
3. Глобалізації – обов’язково буде додана підтримка англійської мови через i18n для залучення іноземних користувачів, чого бракує більшості локальних конкурентів.

Це робить розроблювану платформу конкурентоспроможною та актуальною для сучасного ринку розваг.

1.2 Підбір технологій та засобів розробки веб сайту

Для створення веб сайту квест кімнат «Quest Rooms» було обрано стек технологій, який дозволяє швидко розробити надійний продукт із можливістю подальшого розширення.

Основою для зберігання даних обрано СУБД PostgreSQL. Для дипломного проєкту це оптимальний варіант, оскільки система є безкоштовною, але при цьому дуже надійною. Окрім цього вона все ще не є надто складною у використанні.

Для зручного керування базою на локальному комп'ютері використовується додаток pgAdmin. Він дозволяє візуально переглядати таблиці, перевіряти зв'язки та виконувати тестові запити без написання складного коду в терміналі.

Взаємодія між кодом сайту та базою даних відбувається через Prisma ORM. Це дозволяє працювати з таблицями як зі звичайними об'єктами в коді, що значно прискорює розробку. [10]

Зовнішній вигляд програми pgAdmin наведено на рисунку 1.3.

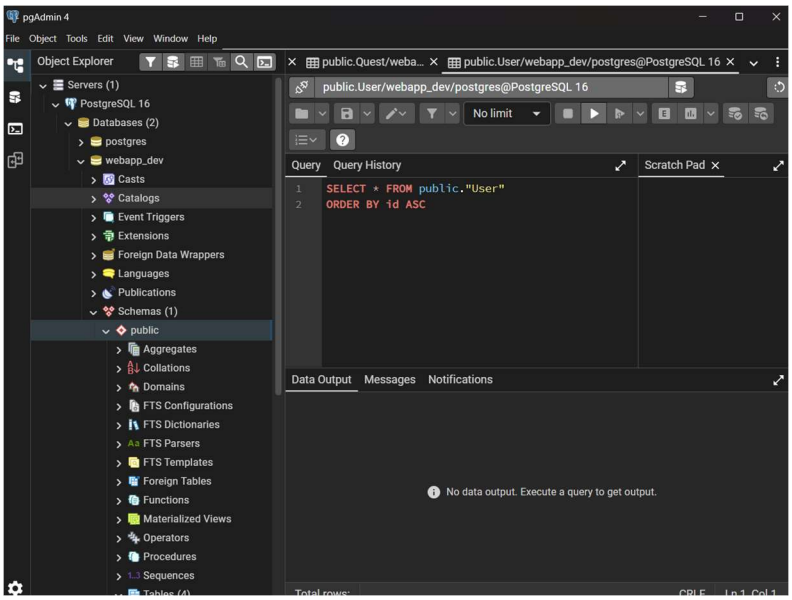

Рисунок 1.3 – вигляд pgAdmin

Додаткове порівняння PostgreSQL з іншими наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння обраної СУБД з аналогами

Характеристика	PostgreSQL	MySQL	MongoDB
Тип бази	Реляційна	Реляційна	Неструктурована
Цілісність даних	Висока	Середня	Обмежена
Складні запити	Підтримує відмінно	Підтримує добре	Працює повільно
Висновок	Оптимально для проєкту	Можливо	Не підходить для бронювань

Далі розглянемо програмну частину розроблюваного веб сайту квест кімнат.

Сайт буде побудовано на фреймворку Next.js 14 та мові TypeScript. Головна перевага такого вибору – це безпека коду. Оскільки TypeScript допомагає уникати помилок ще на етапі написання програми, підказуючи правильні типи даних. [5]

Також у проєкті використано:

1. NextAuth.js – для реалізації входу на сайт;
2. Tailwind CSS – для швидкої верстки дизайну, який автоматично підлаштовується під розміри екранів смартфонів та ноутбуків;
3. Zod – бібліотека, яка перевіряє дані, що вводить користувач у формах.

Для написання коду та управління проектом було обрано середовище розробки Visual Studio Code. Його обрано через те, що він відрізняється своєю гнучкістю та наявністю великої кількості розширень, які полегшують роботу з TypeScript та базою даних. У редакторі використовуються вбудовані інструменти налагодження та інтегрований термінал, що дозволяє виконувати всі операції – від написання функцій до запуску міграцій Prisma – в одному вікні. Окрім цього є поноцінна інтеграція із системою контролю версій – git.

Для фіксації змін у програмі та безпечного зберігання коду використовується система Git. Весь процес розробки супроводжується регулярними комітами, що дозволяє у разі виникнення критичних помилок швидко повернутися до попередньої стабільної версії проекту. Репозиторій проекту розміщено на платформі GitHub, що забезпечує надійне хмарне зберігання коду та візуалізацію історії розробок. Також для спрощення процесів було використано Github Desktop, що дозволило спростити процес комітментів.

1.3 Технічне завдання на розробку «Quest Rooms»

Метою даної роботи є розробка веб–сайту «Quest Rooms», який допоможе автоматизувати процес запису клієнтів на квести. Платформа має замінити ручну реєстрацію через телефон на зручну онлайн–форму, що працює цілодобово. Після чого адміністратори зможуть опрацьовувати заявки самостійно та змінювати їх статус.

1.3.1 Призначення системи

Проєкт створюється для вирішення наступних завдань:

1. Надання інформації за рахунок якої клієнт може ознайомитися з описом квесту, рівнем складності;
2. Онлайн–бронювання – користувач обирає бажаний день та годину в календарі й миттєво резервує слот.
3. Багатомовність – сайт підтримуватиме кілька мов (українську та англійську), що робить його доступним для більшої кількості відвідувачів.
4. Адміністрування – власник квест–кімнати має окремий інтерфейс для керування замовленнями та оновлення інформації на сайті.

1.3.2 Функціональні вимоги

Система повинна забезпечувати стабільну роботу для двох основних груп користувачів: клієнтів та адміністраторів.

Для клієнтів:

– Зручний пошук квестів за категоріями;

– Можливість забронювати квест–кімнату на бажаний час;

– Перегляд деталей квесту, що дозволить краще зрозуміти чи він підходить користувачу.

Для адміністраторів:

– Додавання квестів на веб сайт.

– Перегляд списку всіх заявок та можливість їх редагування.

Окрім цього сайт має мати можливості для подальшого розширення.

1.3.3 Вимоги до структури та оформлення

Код проєкту має бути організований за сучасними стандартами:

– Використання класичної структури для створення проекту, його папок та файлів;

– Винесення всіх текстів у окремі файли перекладів формату .json, щоб легко додавати нові мови;

– Розділення інтерфейсу на окремі компоненти (для зручного повторного використання;

– обов'язкове ведення історії розробки через Git. Кожен логічний етап має бути зафіксований окремим пул реквестом у репозиторії;

– Середовище розробки – використання VS Code з налаштованими правилами форматування коду (Prettier/ESLint), щоб структура файлів та відступи були однаковими в усьому проекті. Що також спрощує саму роботу розробника, за рахунок підказок в самому середовищі розробки.

1.3.4 Техніко–економічне обґрунтування

Доцільність розробки веб сайту «Quest Rooms» на обраному технологічному стеку підтверджується економічними та технічними перевагами використаних інструментів. Головною особливістю є те, що всі засоби розробки мають відкритий початковий код, що повністю виключає витрати на придбання дорогого ліцензійного програмного забезпечення.

Економічний ефект досягається завдяки таким факторам:

1. Зниження витрат на розробку за рахунок використання фреймворку Next.js, що дозволяє створювати і фронтенд, і бекенд в рамках одного проєкту. Цей варіант скорочує час на написання коду та полегшує його підтримку, що в реальних умовах означає меншу кількість витраченого часу.

2. Відсутність ліцензійних платежів. СУБД PostgreSQL та бібліотека Prisma ORM є професійними інструментами, які за функціоналом не поступаються комерційним аналогам, але є абсолютно безкоштовними для використання.

3. Оптимізація дизайну з Tailwind CSS. Оскільки цей інструмент дозволяє відмовитися від написання громіздких стилів вручну. Швидкість завантаження сайту зростає, що позитивно впливає на конверсію відвідувачів у реальні бронювання.

4. Масштабованість через i18n. Впровадження багатомовності на етапі розробки дозволяє системі працювати з іноземними клієнтами без додаткових інвестицій у переробку ядра сайту в майбутньому.

Таким чином, обраний стек дозволяє створити конкурентоспроможний продукт із мінімальними фінансовими вкладеннями, забезпечуючи при цьому високу надійність та швидкість роботи, що є критично важливим для сфери послуг.

1.3.5 План виконання розробки

Розробка розділена на такі етапи:

1. Аналіз даних, що включає в себе проектування таблиць бази даних та опис зв'язків між ними.

2. Підготовка середовища – встановлення редактора VS Code, ініціалізація локального Git–репозиторію та створення першої структури папок Next.js.

3. Розробка інтерфейсу – верстка сторінок каталогу та форми вибору часу.

4. Програмування логіки – налаштування серверних функцій для збереження бронювань у базі.

5. Авторизація – підключення системи входу користувачів.

6. Тестування – перевірка роботи сайту на різних пристроях та виправлення помилок перед запуском.

7. Проведення фінального тестування, завантаження останньої версії коду на GitHub та підготовка до розгортання на сервері.

Після проходження всіх етапів сайт можна було розгортати на хостингу.

1.3.6 Порядок перевірки та здачі роботи

Оцінка готовності веб сайту квест кімнат «Quest Rooms» до експлуатації проводиться шляхом комплексного тестування її функціональних модулів. Основними етапами перевірки якості є:

1. Валідація динамічної навігації.

Контроль коректності формування шляхів для окремих сторінок квестів за їхніми унікальними ID. Система повинна стабільно відображати контент при переході між різними сторінками без помилок.

2. Повна перевірка всіх текстових блоків, системних повідомлень та елементів навігації на предмет наявності перекладів. Жоден елемент інтерфейсу не має залишатися без локалізації при зміні мовної версії.

3. Інтеграційне тестування логіки бронювання. Для цього має бути проведена імітація реального запису на гру для перевірки зв’язку між фронтенд–формою та базою даних PostgreSQL.

4. Перевірка коректності відображення інтерфейсу на різних роздільних здатностях – такзваний контроль адаптивності та UX.

5. Тестування безпеки та автентифікації. Для цього має бути проведена превірка надійності входу через NextAuth.js, контроль валідації полів (через Zod) та захист адміністративної панелі від несанкціонованого доступу.

6. Тестування інструментів адміністратора, зокрема можливості оперативного редагування інформації про квести, їх додавання, зміни цін та моніторингу замовлень.

Після завершення всіх етапів перевірки, виправлення виявлених недоліків та заповнення бази даних актуальною інформацією про квест–кімнати, веб сайт вважається готовим до використання та передається в експлуатацію.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури, архітектури та маршрутизації веб–сайту «Quest rooms»

Під час проєктування архітектурного каркаса для веб–платформи квест–кімнат «Quest rooms» першочерговим завданням було створення гнучкої, зрозумілої та модульної системи, яку можна легко розширювати в майбутньому. Як технологічну основу для реалізації проєкту було обрано сучасний фреймворк Next.js 14, який працює у зв'язці з мовою програмування TypeScript та використовує нову парадигму App Router. Такий вибір дозволяє закрити одразу кілька важливих питань, а саме: забезпечити високу швидкість рендерингу сторінок, зробити зручну навігацію та розмежувати клієнтську і серверну логіку без необхідності піднімати два окремі репозиторії.

В основі архітектури розроблюваного сайту лежить гібридний підхід до обробки та відображення інформації. Оскільки Next.js 14 за замовчуванням розглядає всі компоненти як серверні, це дозволило оптимізувати роботу з базою даних.

Усю логіку взаємодії веб–сайту можна розділити на три взаємопов'язані рівні:

1. Рівень відображення та маршрутизації src/app. Тут містяться безпосередньо сторінки та інтерфейси. Більшість сторінок рендеряться на сервері, завдяки чому користувач одразу отримує готовий HTML–код. Клієнтські компоненти, які використовують директиву «use client», застосовуються лише там, де є жива інтерактивність – форми, модалки, кнопки.

2. Рівень серверних дій src/actions це одна з головних фішок 14–ї версії фреймворку, яку було активно використано в проєкті. Замість того, щоб під кожну дію писати купу стандартних API–контролерів, у папці actions/ створено прямі серверні функції, наприклад – createOrder.ts для замовлень, createQuest.ts

для адмінки, signIn.ts/signUp.ts для профілю. Вони безпечно викликаються прямо з фронтенду, але виконуються суворо на бекенді.

3. Рівень API–маршрутів src/app/api, та класичні Route Handlers, які відповідають за обробку стандартних HTTP–запитів.

Також, щоб сайт не створював під час кожного запиту нове підключення до бази даних, у модулі src/modules/db.ts було реалізовано глобальний клієнт підключення, який працює за принципом Singleton.

Для того, щоб у коді було легко орієнтуватися, структуру каталогу src було чітко розбито за функціональним призначенням. Нижче в таблиці 2.1 наведено детальний опис того, за що відповідає кожна папка.

Таблиця 2.1 – Папки та їх призначення

Папка	Опис та призначення	Основний вміст / Підгрупи
actions	папка, де зосереджена бізнес–логіка мутації даних.	Функції для реєстрації/авторизації, виходу із системи, логіка створення квестів та обробки замовлень.
app	Скелет сайту	Вкладені папки з файлами page.tsx, які формують нові сторінки в браузері
components	Зберігає всі візуальні елементи	Розбиті на підгрупи: admin, footer, header, forms, ui
constants	Статичні константи проєкту	Список категорій квестів, масиви посилань для навігації, початкові стани для очищення форм
lib	Допоміжні утиліти, конфігурації та скрипти	Валідація полів, обробка помилок форм, генерація випадкових картинок,
locales	Локалізація	JSON–файли перекладів
types	Типізація проєкту для запобігання помилкам.	Файли з розширенням .ts, що описують сутності квесту, категорій, замовлень та перекладів

Особливу увагу під час розробки було приділено логіці переходів між сторінками та покращенню користувацького досвіду. У проєкті реалізовано три складні механізми маршрутизації.

Динамічна багатомовність (i18n) реалізована завдяки конфігу i18nConfig.ts та файлу middleware.ts. Таким чином веб сайт автоматично визначає мову користувача і підставляє її в URL–шлях.

Для каталогу квестів реалізовано маршрут, що дозволяє фільтрувати кімнати за жанрами через URL. А для відкриття конкретного квесту створено динамічний маршрут detailed–quest/id/page.tsx, де замість id підставляється унікальний номер квесту, за яким сервер дістає з бази опис та доступні слоти для бронювання.

Для процесу входу на сайт було реалізовано просунутий UX–паттерн за допомогою папок @auth. Тобто якщо користувач натискає «Увійти», перебуваючи, наприклад, на головній сторінці, Next.js 14 перехоплює цей крок. Замість повного перезавантаження і переходу на іншу сторінку, форми SignInForm або SignUpForm красиво відкриваються в модальному вікні поверх поточного контенту, але адреса в браузері все одно змінюється на /sign–in. Якщо ж людина просто скопіює це посилання і відкриє його в новій вкладці, фреймворк відпрацює стандартно і відкриє повноцінну сторінку.

Таким чином, розроблена структура та архітектура дозволили створити швидкий, сучасний веб–сайт, який легко піддається масштабуванню, повністю захищений з боку сервера завдяки Server Actions та має адаптивний і зручний інтерфейс для користувачів із будь–якої країни.

2.2 Створення компонентної структури інтерфейсу та адаптивна верстка сторінок

Проєктування користувацького інтерфейсу веб–сайту «Quest rooms» виконано відповідно до компонентно–орієнтованого підходу. Головною

технологічною базою для розробки інтерфейсу обрано фреймворк Next.js 14 із використанням архітектури App Router, декларативної бібліотеки React та інструментарію Tailwind CSS для побудови адаптивних стилів.

У межах розробки архітектури додатка було організовано чітку ієрархічну структуру сторінок та ізольованих інтерфейсних елементів. Next.js використовує файлову систему для організації маршрутизації, що дозволило розподілити зони відповідальності між глобальними шаблонами, статичними сторінками, API–маршрутами та специфічними для кожного розділу підкомпонентами.

Глобальна архітектура підпапок папки src/app має такий вигляд:

– layout.tsx – кореневий сервісний шар, що відповідає за ініціалізацію мовного середовища, глобальні шрифти, підключення контекстних систем сповіщень та базову геометрію сторінки;

– page.tsx – головна сторінка веб–сайту;

– quests – розділ каталогу квест–кімнат, що містить власну систему фільтрації за категоріями та локальну папку приватних компонентів;

– auth – паралельний маршрут призначений для реалізації модальних вікон авторизації без зміни поточного контексту сторінки.

Основою інтерфейсної ієрархії є кореневий файл макета RootLayout, який визначає глобальну структуру документа фрагмент коду наведен у лістингу 2.1.

```
<body className={`relative ${raleway.className}`}>
  <div className="min–h–full flex flex–col bg–backgroundPrimary text–textWhite">
    <Header locale={locale} />
    <main className="flex–auto">
      <Toaster position="top–right" />
      {auth} {children}
    </main>
```

Лістинг 2.1 – Фрагмент коду структури макету

У наведеному фрагменті застосовано семантичну блочну модель Flexbox, де елемент main має властивість flex–auto. Це гарантує притискання футеру сайту до нижньої межі екрана навіть на сторінках із мінімальним обсягом контенту. Також у макет інтегровано глобальний провайдер спливаючих сповіщень Toaster.

Для забезпечення ідентичності дизайну на всіх екранах додатка, у файлі конфігурації tailwind.config.ts було розширено базову тему та визначено унікальну колірну палітру, що відповідає стилістиці квест–кімнат, а саме темні атмосферні відтінки з неоновими акцентами. Фрагмент новостворених конфігурацій наведено в лістингу 2.2.

```
theme: {
  container: {
    center: true,
  },
  colors: {
    ...colors,
    brandMagenta: "#FF1780",
    white: "#ffffff",
    textWhite: "#E6E6E6",
    backgroundPrimary: "#1C1C1C",
    formsBackground: "#141414",
    darkTransparentBackground: "#3D3333",
  },
```

Лістинг 2.2 – Додаткові налаштування кольорів

Використання семантичних назв кольорів наприклад, bg–backgroundPrimary або text–textWhite дозволяє гнучко керувати стилями, відокремлюючи безпосередні HEX коди від кодової бази компонентів.

Також при розробці сторінок було застосовано методологію адаптивного дизайну, що базується на контрольних точках Tailwind CSS. Це забезпечує коректне відображення як на мобільних пристроях з невеликою шириною екрана, так і на широкоформатних моніторах.

Як приклад реалізації адаптивної посадкової сторінки розглянемо компонент Home наведений у лістингу 2.3.

```
<section className="flex container mx–auto gap–12 py–[120px] justify–center h–screen">
  <div className="flex flex–col gap–7 items–center w–full h–full">
    <h2 className="font–bold text–4xl text–textWhite">{t("heading")}</h2>
    <p className="text–xl text–center text–[#707C87] text–balance">{t("subheading")}</p>
    <Link
      href="/quests"
      className="bg–brandMagenta bg–opacity–25 rounded–md p–2 text–center text–2xl font–bold flex items–center justify–center gap–2 hover:bg–brandMagenta bg–opacity–75 transition–colors"
    >
      <span className="inline–block">{t("actionButtonText")}</span>
      <ArrowRight />
    </Link>
    <ImageSlider />
  </div>
</section>
```

Лістинг 2.3 – Компонент Home

Утиліта «container mx–auto» автоматично встановлює максимальну ширину контенту на основі поточного розширення екрана та вирівнює блок по

центру. Використання класу «text–balance» для текстового опису запобігає появі поодиноких слів на новому рядку, забезпечуючи естетичний розподіл тексту на мобільних дисплеях. Інтерактивні елементи, такі як посилання–кнопка, містять класи «transition–colors» та «hover:bg–opacity–75» для плавного нативного відгуку інтерфейсу на дії користувача.

Для складних інтерфейсів керування даними, таких як адміністративна панель, реалізовано адаптивність для табличних даних, які є критичними до горизонтального простору. Оскільки класичні HTML–таблиці на смартфонах виходять за межі екрана, руйнуючи макет, було використано обгортку з утилітою автоматичного скролу як наведено у лістингу 2.4.

```
<div className="overflow–x–auto">
  <table className="min–w–full divide–y divide–gray–200">
```

Лістинг 2.4 – Елемент скролу таблиці

Клас «overflow–x–auto» дозволяє контейнеру ізолювати широку таблицю всередині мобільного екрана, активуючи плавне горизонтальне гортання лише для цього блоку, зберігаючи при цьому загальну цілісність сторінки. Елементи керування масштабуються під мобільні пальці завдяки збільшеним внутрішнім відступам, що підвищує рівень мобільної ергономіки.

Таким чином, розроблена компонентна структура та використання Tailwind CSS забезпечили високу модульність інтерфейсу, швидкість завантаження сторінок за рахунок відсутності зайвого CSS–коду та повну адаптивність системи «Quest rooms».

2.3 Проєктування схеми даних, сутностей та взаємодії з СУБД.

Для забезпечення надійного збереження інформації, швидкого доступу до неї та підтримки цілісності даних веб сайту «Quest rooms» було спроектовано

реляційну базу даних. Як систему керування базами даних обрано PostgreSQL – потужну об'єктно–реляційну систему з відкритим вихідним кодом, що забезпечує високу стійкість до навантажень, підтримку складних типів даних та відповідність вимогам ACID.

Для взаємодії між кодовою базою Next.js та СУБД PostgreSQL використано сучасний інструмент Prisma ORM. Це дозволило застосувати підхід Database–First / Schema–First, де структура бази даних описується декларативною мовою Prisma, після чого автоматично генеруються типи TypeScript та SQL–скрипти міграцій. [9]

У ході аналізу предметної області веб сайту було виділено три ключові сутності: Quest – квест–кімната, Order – заявка та User – користувач чи адміністратор. Конфігурація джерела даних та частковий опис моделей у файлі schema.prisma наведено у лістингу 2.5.

```
datasource db {
  provider  = "postgresql"
  url       = env("POSTGRES_PRISMA_URL")
  directUrl = env("POSTGRES_URL_NON_POOLING")
}
model Quest {
  id          Int      @id @default(autoincrement())
  title       String
  description  String
  previewImg   String
  coverImg     String
  type         String
  level        String
  peopleCount  Int[]
```

Лістинг 2.5 – Конфігурація бази даних

Сутність Quest описує квест–кімнату, доступну для бронювання. Поле id є первинним ключем з автоінкрементом. Особливістю є використання поля peopleCount із типом Int. Це специфічна можливість PostgreSQL, яка дозволяє зберігати діапазон допустимої кількості учасників в одному полі без створення додаткових проміжних таблиць зв'язку. Поля previewImg та coverImg зберігають текстові шляхи до графічних ресурсів.

Сутність Order зберігає інформацію про сеанси бронювання. Крім базових контактних даних name, phone та прапорця згоди з правилами isLegal, модель містить поле scheduledAt з типом DateTime для фіксації дати й часу квесту, а також поле стану status, значення за замовчуванням – «pending», що дозволяє адміністратору керувати життєвим циклом заявки через інтерфейс системи.

Сутність User призначена для автентифікації. Поле email має директиву unique, яка створює унікальний індекс на рівні СУБД, унеможливлюючи реєстрацію двох облікових записів на одну поштову адресу. Пароль зберігається у вигляді захищеного хешу.

На основі розробленої схеми інструментарій Prisma ORM автоматично транслює абстрактні моделі у фізичні таблиці реляційної бази даних за допомогою механізму міграцій. Приклад SQL–коду представлено у лістингу 2.6.

```
    "id" SERIAL NOT NULL,
    "name" TEXT NOT NULL,
    "peopleCount" INTEGER NOT NULL,
    "phone" TEXT NOT NULL,
    "isLegal" BOOLEAN NOT NULL,
    "scheduledAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "status" TEXT NOT NULL DEFAULT 'pending',
    CONSTRAINT "Order_pkey" PRIMARY KEY ("id")
```

Лістинг 2.6 – SQL–код створення таблиці заявок

Під час виконання цієї міграції PostgreSQL виконує такі операції:

1. Створюється таблиця «Order», де замість абстрактного типу Int з автоінкрементом використовується системний тип даних SERIAL, який автоматично створює послідовність для генерації унікальних ідентифікаторів.

2. Рядові типи даних String та Boolean транслюються у відповідні нативні типи SQL: TEXT та BOOLEAN.

3. Створюється обмеження первинного ключа CONSTRAINT «Order_pkey» PRIMARY KEY («id»), що забезпечує унікальність ідентифікаторів на рівні фізичного диска та автоматично будує B–Tree індекс для прискорення операцій вибірки за id.

2.3.1 Взаємодія додатка з СУБД та пулінг підключень

Оскільки додаток розгорнуто у хмарному середовищі із безсерверною архітектурою, класичне постійне утримання з'єднань із базою даних є неефективним та може призвести до вичерпання ліміту доступних сокетів СУБД при великій кількості одночасних запитів.

Для вирішення цієї проблеми у конфігурації datasource db розділено шлях підключення на два потоки:

1. url = env("POSTGRES_PRISMA_URL") – основне підключення, яке використовує механізм пулінгу з'єднань через спеціалізований проксі–сервер. Це дозволяє сотням безсерверних функцій Next.js повторно використовувати обмежену кількість фізичних з'єднань із PostgreSQL, не перевантажуючи сервер БД.

2. directUrl = env("POSTGRES_URL_NON_POOLING") – пряме з'єднання з базою даних в обхід пулера. Воно використовується виключно під час виконання міграцій, оскільки операції зміни структури таблиць вимагають монопольного доступу до сесії та не можуть виконуватися через загальний пул.

Завдяки такій схемі взаємодії, розроблена база даних веб сайту «Quest rooms» є повністю захищеною від ін'єкцій, типізованою на етапі компіляції додатка та оптимізованою під високі навантаження.

2.4 Програмування клієнтської та серверної логіки

Програмний комплекс веб сайту «Quest rooms» побудовано на базі сучасного фреймворку Next.js, який поєднує в собі можливості клієнтського рендерингу та виконання асинхронного коду безпосередньо на стороні сервера. Завдяки використанню архітектури App Router, логіка системи розділена на два рівні:

1. Рівень представлення та серверної генерації – асинхронні компоненти, які виконуються виключно на сервері, мінімізуючи обсяг JavaScript–коду, що надходить у браузер користувача.
2. Рівень інтеграційного API – набір ізольованих безсерверних HTTP–ендпоінтів, які забезпечують зв'язок між клієнтським інтерфейсом, адміністративною панеллю та базою даних PostgreSQL через Prisma ORM.

Використання вказаного стеку дозволяє досягти оптимального балансу між продуктивністю та зручністю обслуговування системи.

2.4.1 Реалізація серверних компонентів та API–маршрутів

Усі сторінки додатка за замовчуванням є серверними компонентами. Це дозволяє перенести важкі операції, такі як зчитування файлової системи, завантаження конфігурацій перекладів i18n та первинну збірку HTML, безпосередньо на сервер.

Як приклад серверного компонента розглянемо реалізацію головної сторінки додатка наведену у лістингу 2.7.

```
return (
  <section className="flex container mx–auto gap–12 py–[120px] justify–center h–screen">
    <div className="flex flex–col gap–7 items–center w–full h–full">
      <h2 className="font–bold text–4xl text–textWhite">{t("heading")}</h2>
      <p className="text–xl text–center text–[#707C87] text–balance">
        {t("subheading")}
      </p>
```

Лістинг 2.7 – Код головної сторінки

Ключові переваги такої реалізації на рівні сервера:

– Асинхронність – компонент визначено як асинхронну функцію, що дозволяє виконувати операцію await initTranslations прямо під час генерації розмітки, усуваючи необхідність у клієнтських хуках useEffect чи станах завантаження;

– Оптимізація продуктивності – браузер клієнта отримує вже сформований HTML–файл. Клієнтська частина додатка не завантажує пакети, необхідні для парсингу локалізацій, що суттєво покращує метрики First Contentful Paint (FCP) та Largest Contentful Paint (LCP).

Для забезпечення гнучкої взаємодії з даними та обробки запитів від адміністративної панелі, у папці src/app/api/ було розгорнуто REST API архітектуру. Згідно з правилами Next.js, кожен маршрут ізольовано у відповідній директорії у файлі route.ts. Розроблена структура наступні ендпоінти:

1. api/auth/sign–in/route.ts – автентифікація користувачів;
2. api/auth/sign–up/route.ts – реєстрація нових облікових записів;
3. api/create–order/route.ts – створення нових заявок на бронювання;
4. api/create–quest/route.ts – додавання нових квест–кімнат у систему;

5. api/detailed–quest/[id]/route.ts – отримання детальної інформації про конкретний квест за його ідентифікатором;

6. api/orders/route.ts та api/orders/[id]/route.ts – отримання, модифікація (PATCH) та видалення заявок через адмін–панель.

Розглянемо детальну реалізацію логіки перевірки облікових даних у файлі api/auth/sign–in/route.ts, наведену в додатку А.

Метод POST приймає стандартний об'єкт запиту Request, з якого за допомогою await request.json() десеріалізується тіло JSON–запиту з клієнта.

Далі запит db.user.findFirst генерує оптимізований SQL–запит до СУБД PostgreSQL. Пошук відбувається за унікальним індексом email, що забезпечує швидкість виконання операції за часовою складністю.

У випадку відсутності запису в БД або невідповідності паролів, система перериває виконання функції та повертає структуровану помилку через об'єкт NextResponse.json() із встановленням відповідного специфікації HTTP коду помилки 401. Це дозволяє клієнтській стороні чітко диференціювати причину відмови.

У разі збігу всіх параметрів, сервер повертає статус 200 та дані користувача для подальшого створення сесії доступу на рівні серверних дій.

Такий поділ на серверні компоненти й API–маршрути дозволив досягти високої швидкості завантаження публічної частини сайту, надійно захистити логіку авторизації від клієнтських маніпуляцій та організувати чисту REST–архітектуру для управління даними веб сайту.

2.4.2 Реалізація клієнтських форм та управління станом бронювання

Для забезпечення взаємодії з відвідувачами сайту та збору заявок на бронювання квест–кімнат було розроблено клієнтський компонент форми створення замовлення.

Управління станом відправки даних та отримання відповідей від сервера реалізовано за допомогою спеціального хука useFormState. Фрагмент коду ініціалізації наведено у лістингу 2.8.

```
import { useFormState } from "react–dom";
import { createOrder } from "@/actions/createOrder";
import { formInitialState } from "@/constants/formInitialState";

// Всередині компонента CreateOrderForm:
const [state, formAction] = useFormState(createOrder, formInitialState);
```

Лістинг 2.8 – Ініціалізація стану форми замовлення

Хук useFormState приймає серверну дію createOrder та початковий об'єкт стану formInitialState. Він повертає поточний стан відповіді сервера та formAction, який передається безпосередньо в атрибут action HTML–тегу <form>. Це дозволяє відправляти дані на сервер асинхронно, без необхідності вручну створювати обробники подій onSubmit та викликати preventDefault.

Для того щоб користувач миттєво бачив, яке саме поле заповнено некоректно, у компоненті викликається findErrors. Програмний аналіз помилок для кожного поля наведено у лістингу 2.9.

```
const nameErrors = findErrors("name", state.message);
const phoneErrors = findErrors("phone", state.message);
const peopleCountErrors = findErrors("peopleCount", state.message);
const scheduledAtErrors = findErrors("scheduledAt", state.message);
const isLegalErrors = findErrors("isLegal", state.message);
```

Лістинг 2.9 – Логіка визначення помилок для полів введення

Після кожної спроби відправки форми, якщо серверна валідація Zod виявила порушення, масив помилок записується в state.message. Результат передається у кастомний UI–компонент <ErrorMessages errors={...} />, який рендерить текст помилки безпосередньо під відповідним полем введення.

У разі успішного опрацювання заявки сервером, інтерфейс повинен зреагувати та повідомити про це користувача. Реалізацію функції зворотного зв'язку наведено у лістингу 2.10.

```
import toast from "react–hot–toast";
import { useRouter } from "next/navigation";

const router = useRouter();

function handleSuccess(message: string) {

  if (typeof message === "string") {
    toast.success(message);
    router.back();
  }

  return;
}
```

Лістинг 2.10 – Функція обробки успішного створення замовлення

Коли сервер повертає позитивний статус, у тілі форми спрацьовує тригер, який викликає функцію handleSuccess. За допомогою бібліотеки react–hot–toast на екрані з'являється спливаюче вікно з підтвердженням успішного бронювання, вигляд якого наведено на рисунку 2.1).

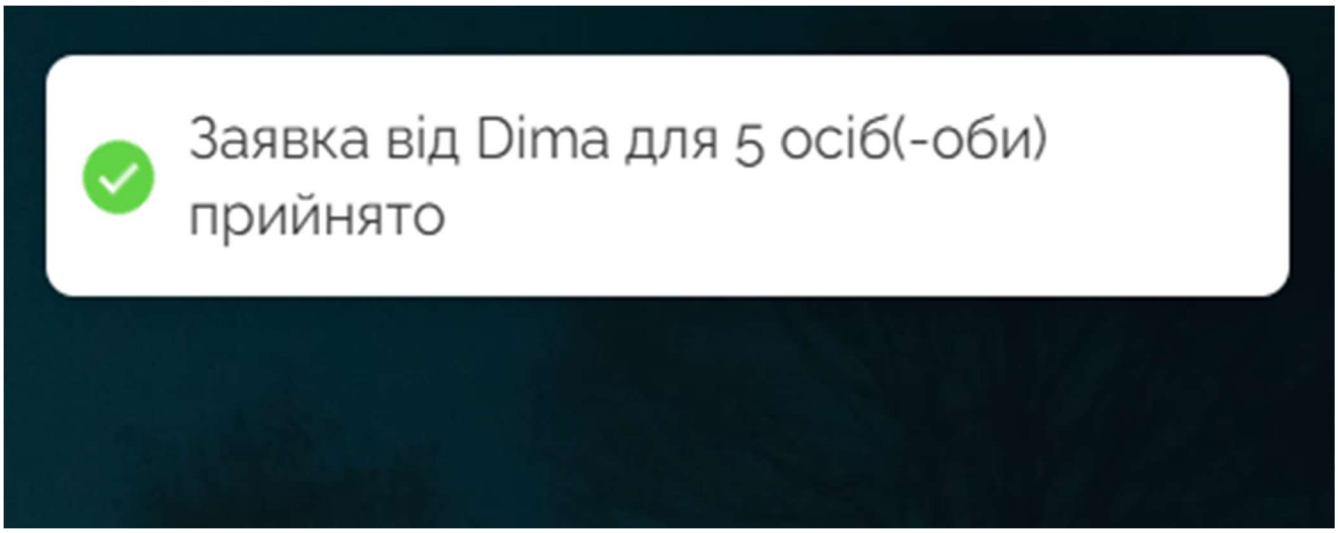

Рисунок 2.1 – Сповіщення про успішне замовлення

Такий поділ логіки форми на ізольовані клієнтські операції у зв'язці із Server Actions дозволив створити гнучкий, швидкий та відмовостійкий інтерфейс, який мінімізує навантаження на клієнтську частину додатка.

2.4.3 Розробка інтерактивної клієнтської логіки, стану та валідації форм

Для забезпечення високого рівня інтерактивності та надійності клієнтської частини додатка під час взаємодії користувача з формами (вхід, реєстрація, бронювання квестів) було реалізовано комплексну систему керування станом, реактивного відображення помилок валідації та запобігання аварійним збоям інтерфейсу.

Zod дозволяє декларативно описувати структуру даних та правила їх перевірки, що гарантує відповідність інформації, яка надходить від користувача, бізнес-вимогам системи ще до моменту звернення до бази даних. [8]

Під час відправки даних через форми, бібліотека валідації Zod генерує масив об'єктів типу ZodIssue. Для того щоб розібрати цей масив і відобразити специфічний текст помилки під кожним відповідним полем введення, було розроблено утилітарну функцію findErrors наведену у лістингу 2.11.

```
export const findErrors = (fieldName: string, errors: ZodIssue[] | string) => {
  if (typeof errors === "string") { return []; }
  return errors
    .filter((item) => {
      return item.path.includes(fieldName);
    })
```

Лістинг 2.11 – Функція findErrors

Ця функція ізолює логіку фільтрації помилок за допомогою строгої типізації TypeScript. Метод filter перевіряє масив path у структурі помилки Zod, зіставляючи його з технічним ім'ям поля. Метод map повертає виключно рядки повідомлень, які потім рендериться у вигляді червоного тексту під полем введення, створюючи миттєвий реактивний відгук на некоректні дії користувача без перезавантаження сторінки.

Окрім локальних помилок полів, існують глобальні помилки сервера. Для запобігання дублюванню спливаючих вікон та уніфікації сповіщень розроблено модуль handleFormError наведений в лістингу 2.12.

```
export function handleFormError(
  message: string | ZodIssue[],
  toastShownRef?: React.Ref<boolean>
) {
  if (typeof message === "string" && message) {
    if (!(toastShownRef as RefObject<boolean>).current) {
      toast.error(message);
    }
  }
  return;}
```

Лістинг 2.12 – Код модуля handleFormError

Використання посилання React.Ref дозволяє зберігати мутабельний стан прапорця toastShownRef між рендерами компонента. Це вирішує поширену проблему React–додатків – розсинхронізацію та появу кількох однакових повідомлень про помилку під час швидких повторних кліків користувача на кнопку відправки форми.

Попри наявність клієнтських перевірок, кінцевий захист бази даних реалізовано безпосередньо в API–ендпоінті створення заявки api/create–order/route.ts. Сервер виконує жорстку перевірку на наявність усіх обов'язкових полів, код цих перевірок наведено у лістингу 2.13.

```
export async function POST(request: Request) {
  const body = await request.json();
  const { name, peopleCount, phone, isLegal, scheduledAt } = body;

  if (![name, peopleCount, phone, isLegal, scheduledAt].every(Boolean)) {
    return NextResponse.json({ message: "Введіть дані у всі поля" },
        { status: 400 });
  }
  const order = await db.order.create({
    data: {name,peopleCount,phone,isLegal,scheduledAt: new
Date(scheduledAt),},
});
  if (!order) {
    return NextResponse.json(
      { message: "Виникла помилка при створенні заявки" }, { status: 400 }
    );
  }
```

Лістинг 2.13 – Код серверних перевірок

Метод every є захисним бар'єром, тобто якщо зловмисник надішле прямий HTTP–запит в обхід графічного інтерфейсу з порожніми значеннями, сервер заблокує операцію, повернувши статус 400 , що гарантує цілісність даних у СУБД.

Для того щоб уся система не виходила з ладу через локальну помилку рендерингу, на рівні архітектури Next.js реалізовано спеціальний клієнтський компонент відловлювання критичних помилок error.tsx. Цей компонент виконує функцію запобіжника. Коли в дочірньому дереві компонентів стається критичний збій, Next.js ізолює його й замість повної зупинки сайту чи відображення системного білого екрана монтує даний інтерфейс. Функція reset дозволяє користувачеві здійснити спробу повторного рендерингу компонента без повної перезагрузки всього веб сайту.

Таким чином, розроблена клієнтська логіка, що базується на типізованих інструментах TypeScript, зв'язці useRef + react–hot–toast та Next.js Error Boundaries, забезпечує високу відмовостійкість системи, безперебійну роботу інтерфейсу та безпечну валідацію користувацького введення.

2.4.4 Архітектура головної сторінки панелі адміністратора

Для централізованого керування веб сайтом було розроблено захищену сторінку адміністратора. Вона реалізована як асинхронний серверний компонент, що дозволяє робити прямі запити до бази даних через серверні дії .

Головна сторінка адмін–панелі об'єднує в собі три основні інтерактивні елементи:

1. Таблицю поточних замовлень від клієнтів OrdersTable;
2. Таблицю керування існуючими квест–кімнатами QuestsTable;
3. Спойлер–компонент із вбудованою формою додавання нових квестів.

Фрагмент коду, що відповідає за отримання, форматування даних та архітектуру сторінки, наведено у лістингу 2.14.

```
export default async function AdminPage() {
  const orders = await getOrders();
  const quests = await getQuests();
  const formattedOrders = orders.map(order => ({
    ...order,
    scheduledAt: order.scheduledAt.toLocaleDateString('uk–UA'),
  }));
  return (
    <OrdersTable orders={formattedOrders} />
    <QuestsTable quests={quests} />
    <details className="rounded–xl border border–[rgb(72_0_66)] bg–formsBackground shadow–sm">
      <summary className="cursor–pointer px–5 py–4">Розгорнути форму додавання</summary>
      <CreateQuestForm />
    </details>
  );
}
```

Лістинг 2.14 – Логіка сторінки адміністратора

Нижче наведено графічний інтерфейс кожного з трьох інтегрованих елементів керування панелі адміністратора:

1. Модуль керування замовленнями <OrdersTable /> приймає масив відформатованих заявок і виводить їх у вигляді інтерактивного списку. Тут адміністратор бачить ім'я клієнта, телефон, обрану дату, час та кількість людей. Вигляд цього модуля наведено на рисунку 2.2.

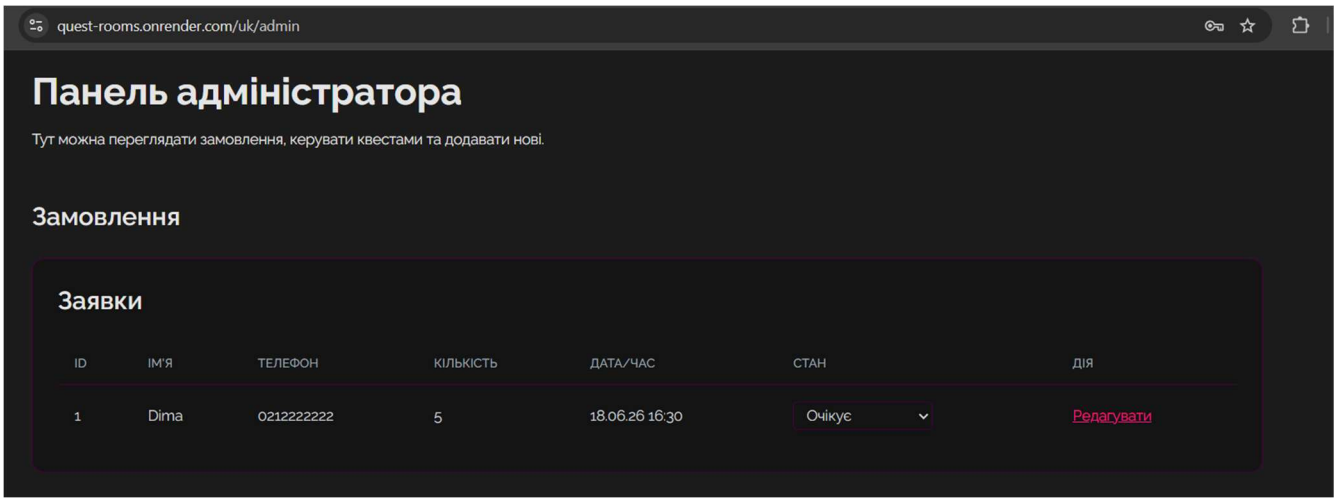

Рисунок 2.2 – Інтерфейс таблиці замовлень «OrdersTable» в адмін–панелі

2. Модуль моніторингу існуючих квестів <QuestsTable /> відповідає за відображення списку всіх кімнат, які зараз активні на сайті. Він дозволяє оперативно оцінити поточний каталог і перейти до їх редагування. Вигляд модуля моніторингу квестів наведено на рисунку 2.3.

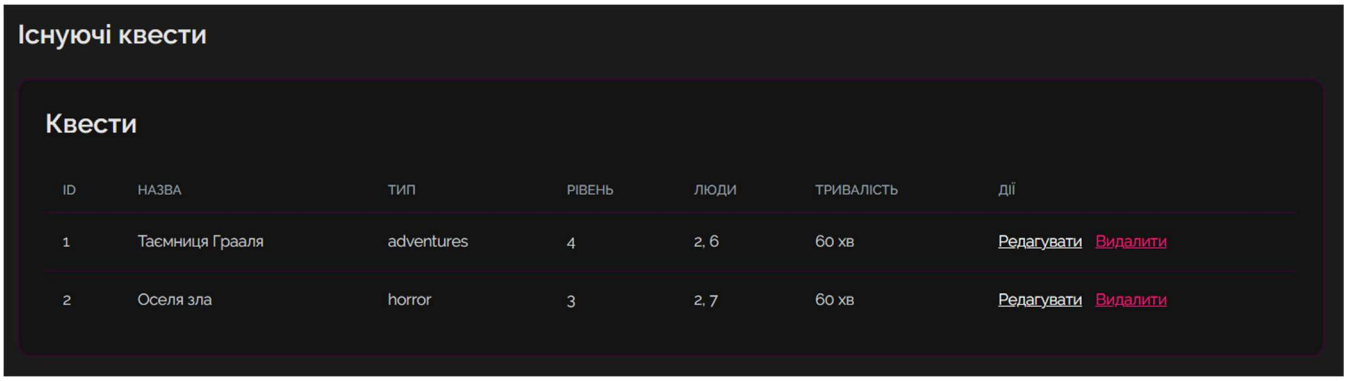

Рисунок 2.3 – Інтерфейс списку існуючих квестів «QuestsTable»

3. Елемент додавання квесту. Для оптимізації простору робочої області адміністратора, форму <CreateQuestForm /> було загорнуто в семантичний HTML–тег <details>. Це дозволило сховати велику форму за замовчуванням. Вона розгортається плавно та без перезавантаження сторінки лише після кліку на плашку «Розгорнути форму додавання». Вигляд сторінки із згорнутим елементом наведено на рисунку 2.4

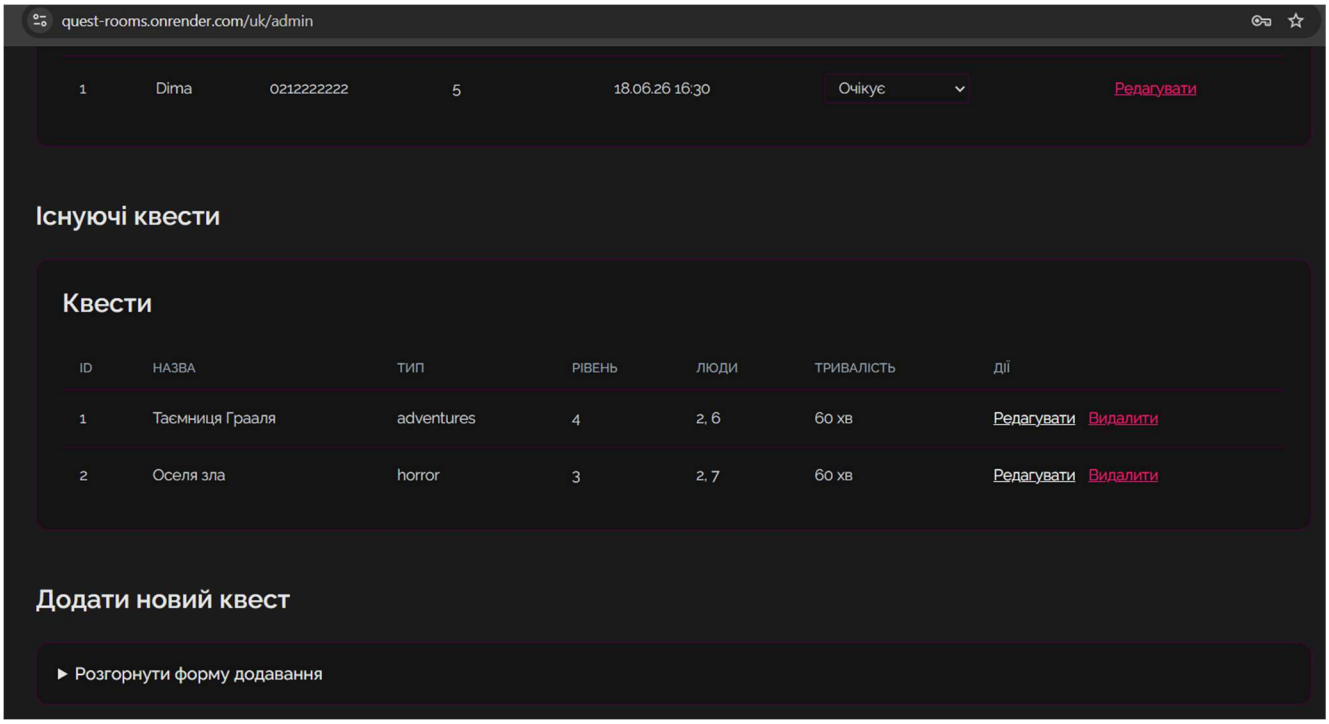

Рисунок 2.4 – Адмін сторінка із згорнутою формою

Форму в розгорнутому стані наведено на рисунку 2.5

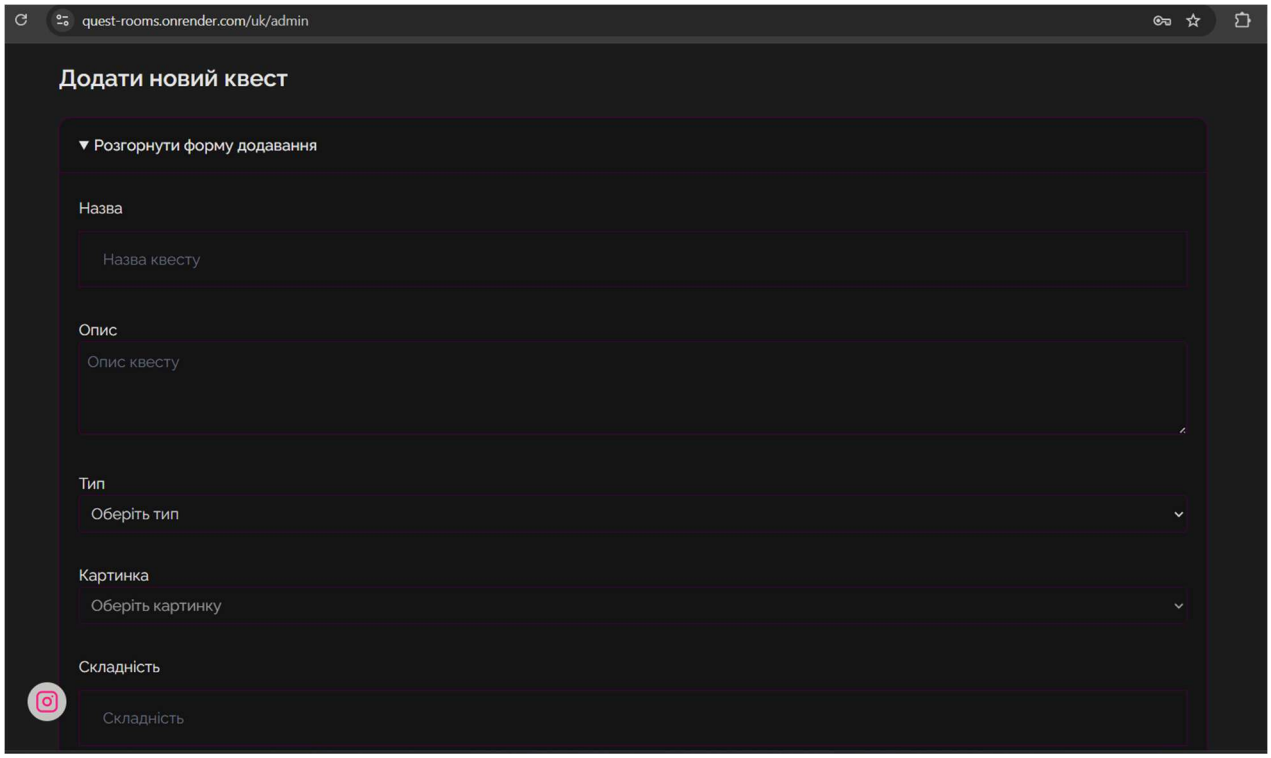

Рисунок 2.5 – Компонент форми додавання нового квесту

Така модульна структура сторінки забезпечує високу швидкість рендерингу (завдяки SSR), чистий код через розділення інтерфейсу на окремі компоненти та зручний UX–досвід для модератора системи.

Окрім цього варто зазначити що в формі додавання квесту використано кілька випадаючих полів, а саме:

1. Тип – має готовий набір типів квест–кімнат, що спрощує додавання нових квестів та уніфікує їх типи, що дозволяє уникнути помилок та краще підтримувати фільтрування;

2. Зображення – задля економії місця було попередньо завантажено набір зображень та погруповано їх відповідно до типу квесту. Даний вибір залежить від раніше обраного типу квесту. Таким чином зменшено кількість пам’яті що використовується для зберігання зображень, а також спрощує процес додавання квестів. Дропдаун для вибору зображення наведено на рисунку 2.6.

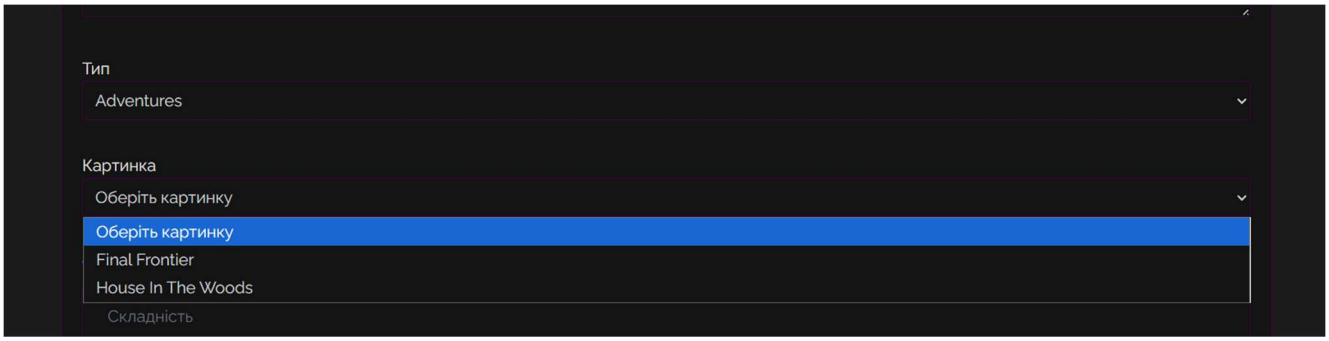

Рисунок 2.6 – Поле вибору зображення

Кожне із полів має приклад заповнення, що є критично важливим задля уникнення помилок в заповненні та економії часу на навчанні роботі із системою.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Написання юніт–тестів та інтеграційне тестування компонентів

Важливою складовою життєвого циклу розробки веб сайту квест–кімнат «Quest rooms» є етап тестування, спрямований на верифікацію коректності роботи алгоритмів, ізольованих функцій, а також інтеграційної взаємодії між модулями додатка та зовнішніми інтерфейсами.

Для побудови тестового оточення було обрано фреймворк Vitest, який є сучасним високопродуктивним інструментом, оптимізованим для роботи з проєктами на базі Vite та Next.js, що підтримує повну типізацію TypeScript та вбудовані механізми імітації. Таким чином даний фреймворк ідеально підходить під технології розроблюваного веб–сайту. У межах роботи було застосовано два рівні тестування:

1. Модульне тестування – перевірка поведінки чистих ізольованих функцій та утиліт без зовнішніх залежностей.
2. Інтеграційне тестування – перевірка серверної логіки, що включає імітацію мережевих HTTP–запитів до API та контроль за життєвим циклом передачі даних від форми до сховища. [12]

Для перевірки коректності роботи клієнтського обробника помилок, що розбирає помилки валідації бібліотеки Zod, було створено набір юніт–тестів для функції findErrors у файлі findErrors.test.ts. Головна мета цих тестів – перевірити, що функція правильно фільтрує масив помилок за ключовим полем і повертає лише необхідні текстові повідомлення користувачу.

Перший тест перевіряє граничну умову: якщо замість очікуваного масиву помилок ZodIssue сервер повертає звичайний рядок із загальною помилкою, функція не повинна викликати збій додатка, а має повернути порожній масив.

Уривок коду модульних тестів представлено у лістингу 3.1.

```
it("returns an empty array when errors is a string", () => {
  expect(findErrors("title", "Some error message")).toEqual([]);
 });
 it("filters Zod issues by field path and returns messages", () => {
  const issues: z.ZodIssue[] = [
   { code: "invalid_type", path: ["title"], message: "Title is required" },
   { code: "too_small", path: ["description"], message: "Description is too short" },
   { code: "invalid_type", path: ["title", "nested"], message: "Nested title failed" },
  ];
  expect(findErrors("title", issues)).toEqual(["Title is required", "Nested title failed"]);
 });
```

Лістинг 3.1 – Фрагмент коду модульних тестів

Другий ж тест перевіряє базовий функціонал фільтрації. У даному випадку конструкція expect().toEqual() перевіряє, що з трьох тестових об'єктів утиліта відібрала лише два, і успішно трансформувала їх у масив текстових рядків.

Результат виконання тестів наведено на рисунку 3.1.

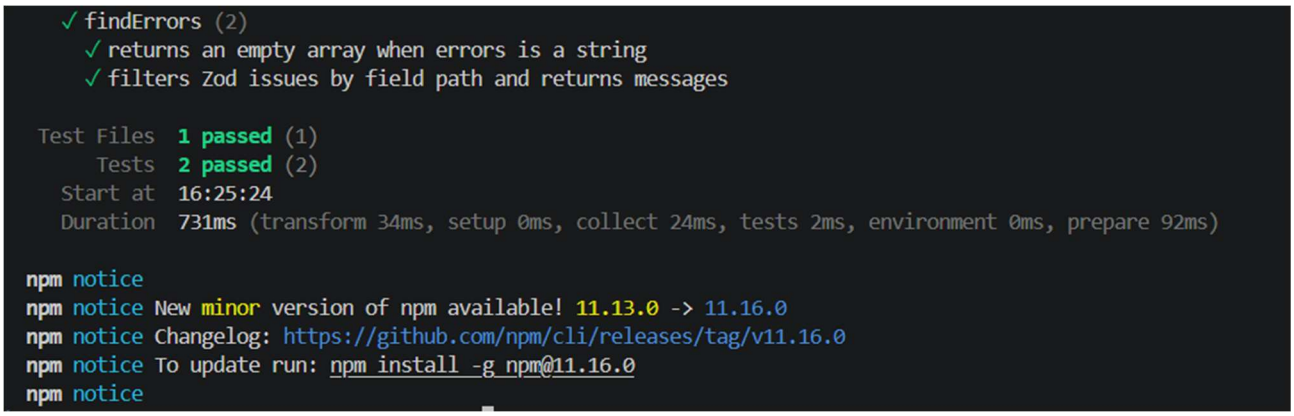

Рисунок 3.1 – Результат виконання юніт тестів

Відповідно до рисунку 3.1 бачимо що тести успішно проходять та перевірки працюють коректно.

Тестування серверної дії створення квесту createQuest є інтеграційним, оскільки воно охоплює перевірку взаємодії між внутрішньою схемою валідації форми та мережевим шаром виконання запитів fetch. Для забезпечення ізоляції тесту від реального сервера та бази даних, у файлі тестування було застосовано механізм підміни глобальних об'єктів за допомогою методу vi.stubGlobal.

Фрагмент коду інтеграційних тестів для createQuest наведено у лістингу 3.2, а весь код даних тестів наведено в додатку Б.

```
it("returns validation errors when form data is invalid", async () => {
 const formData = new FormData();
 formData.append("title", "");
 formData.append("description", "");

 const result = await createQuest({}, formData);

 expect(result.success).toBe(false);
 expect(result.message).toBeInstanceOf(Array);
});
```

Лістинг 3.2 – Фрагмент коду інтеграційних тестів

Перед кожним тестом створюється ізольоване середовище, де глобальний метод fetch замінюється на порожню spy функцію vi.fn(), а також ініціалізується змінна оточення process.env.API_BASE_URL. Після кожного тесту стан повністю відновлюється, що гарантує відсутність взаємного впливу тестів один на одного.

Розглянемо детальніше сценарії для яких написані тести:

1. Помилка валідації форми – цей тест перевіряє поведінку системи у випадку надсилання порожнього об'єкта FormData. Оскільки поля не проходять внутрішню перевірку схеми Zod, функція createQuest повинна миттєво повернути прапорець success: false та масив об'єктів помилок, навіть не намагаючись виконувати мережевий запит.

2. Успішне створення квесту – за допомогою методу mockFetch.mockResolvedValue імітується успішна відповідь від віддаленого API. Тест перевіряє, чи функція правильно збирає текстове повідомлення про успіх, а твердження expect(mockFetch).toHaveBeenCalledWith додатково перевіряє, що мережевий запит було надіслано на правильну URL–адресу із використанням HTTP–методу POST.

3. Обробка помилок сервера – в даному випадку це інтеграційний тест, який верифікує, що блок try/catch всередині серверної дії успішно перехоплює цю помилку, дістає з тіла відповіді повідомлення «Bad request» і безпечно повертає його клієнту, запобігаючи падінню всього додатка.

Загальні результати цих трьох сценаріїв наведено на рисунку 3.2.

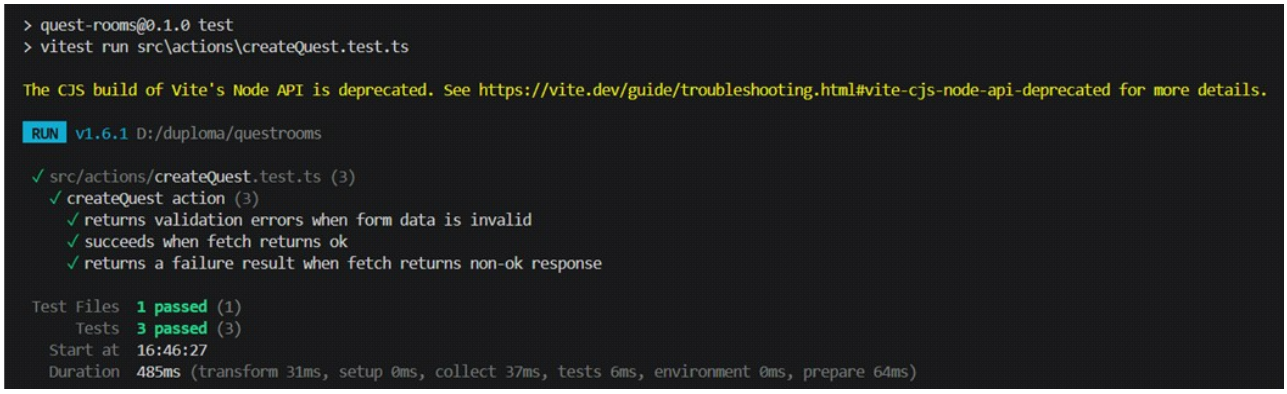

Рисунок 3.2 – Результат запуску тестових сценаріїв

Впровадження автоматизованого тестування за допомогою Vitest на ранніх етапах розробки дозволило гарантувати стабільність обробки форм, захистити додаток від регресійних помилок під час рефакторингу коду веб сайту «Quest rooms».

3.2 Інструкція з локального розгортання, оптимізації та деплою

Перед тим як завантажувати останню версію коду на GitHub, для деплойменту було перевірено, чи правильно вказана команда запуску міграцій перед збіркою, конфігурацію наведено у лістингу 3.3.

```
"scripts": {
  "dev": "next dev",
  "build": "prisma generate && prisma migrate deploy && next build",
  "start": "next start"
}
```

Лістинг 3.3 – Конфігурація перед деплойментом

У даному випадку команда prisma migrate deploy застосує всі створені SQL–міграції до бази даних на Render під час деплою, а prisma generate збереже правильні типи TypeScript. Перевіривши це – код закинуто на github та розпочато процес деплойменту.

Для початку пройдено реєстрацію на хостингу Render.com. Для цього використано метод реєстрації через github акаунт. Першим ділом створено базу даних, при цьому вказано наступні параметри: name, database, user, region, instance type. Цей етап наведено на рисунку 3.3.

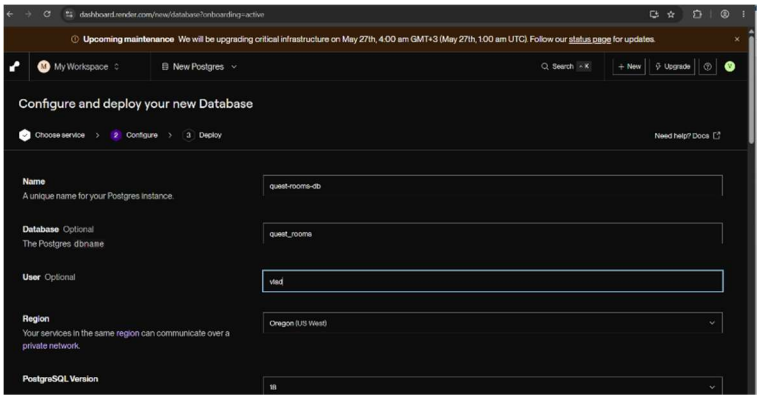

Рисунок 3.3 – Створення бази даних на Render.com

Після успішного створення бази даних в налаштуваннях скопійовано External Database URL, значення якого буде використано при подальших кроках розгортання вже веб сервісу.

Далі створено веб сервіс і оскільки безсерверні функції Next.js на безкоштовному тарифі Render автоматично не працюють, розгортатиметься Next.js як постійно запущений Node.js сервер.

Для спрощення роботи обрано опцію – «Build and deploy from a Git repository». При цьому встановлено доступ саме до розроблюваного проекту та його репозиторію. Вікно налаштування наведено на рисунку 3.4.

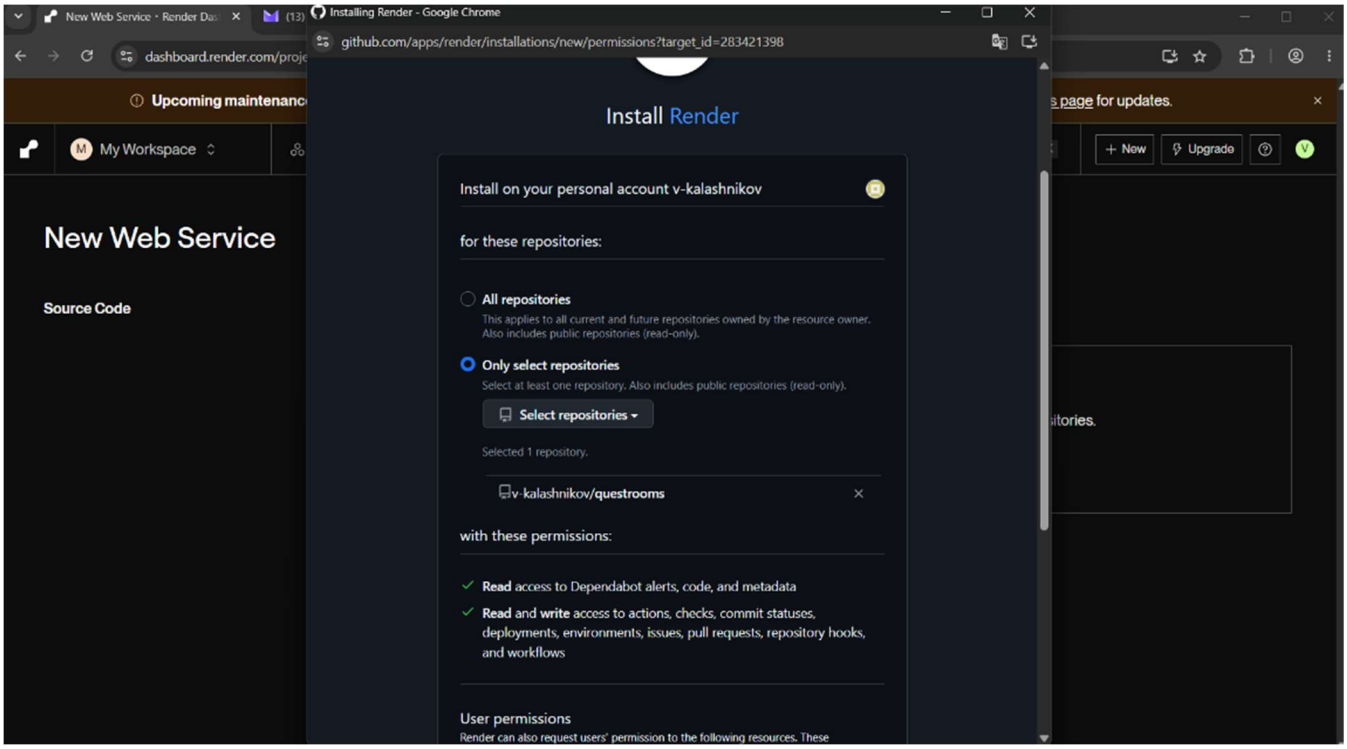

Рисунок 3.4 – Розгортання через github акаунт

Далі налаштовано параметри сервісу: name, region, branch, language, instance type. Обрано регіон відповідно до того що був для бази даних. Результати заповнення цих необхідних полів наведено на рисунку 3.5.

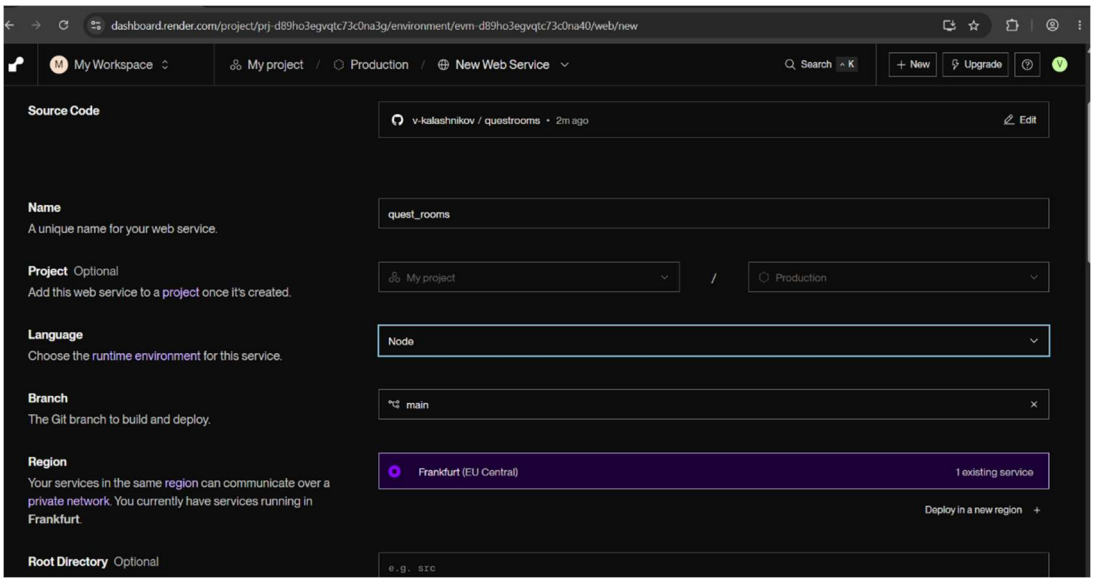

Рисунок 3.5 – Процес налаштування параметрів

Далі обов’язково налаштовано команди розділу «Build and Deploy», як наведено на рисунку 3.6.

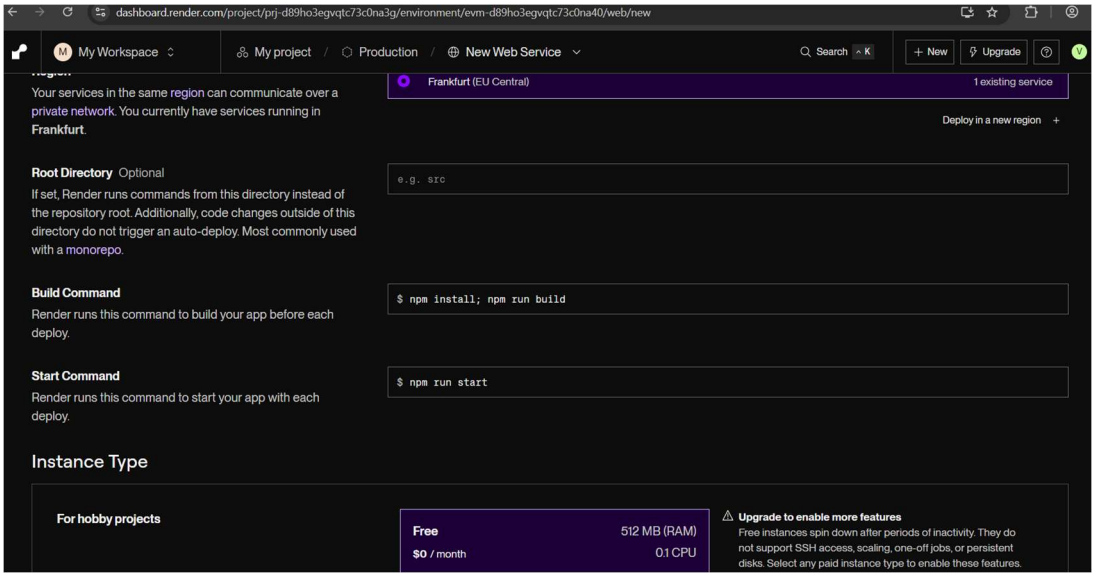

Рисунок 3.6 – Налаштування «Build and Deploy»

На тій же сторінці налаштування Web Service внизу в Advanced налаштуваннях додано змінні API_BASE_URL, POSTGRES_PRISMA_URL та POSTGRES_URL_NON_POOLING, які використовуватиме Prisma. Для зв’язку з базою даних.

Після цього підтверджено створення веб сервісу та починається процес деплою самим Render. Даний процес наведено на рисунку 3.7.

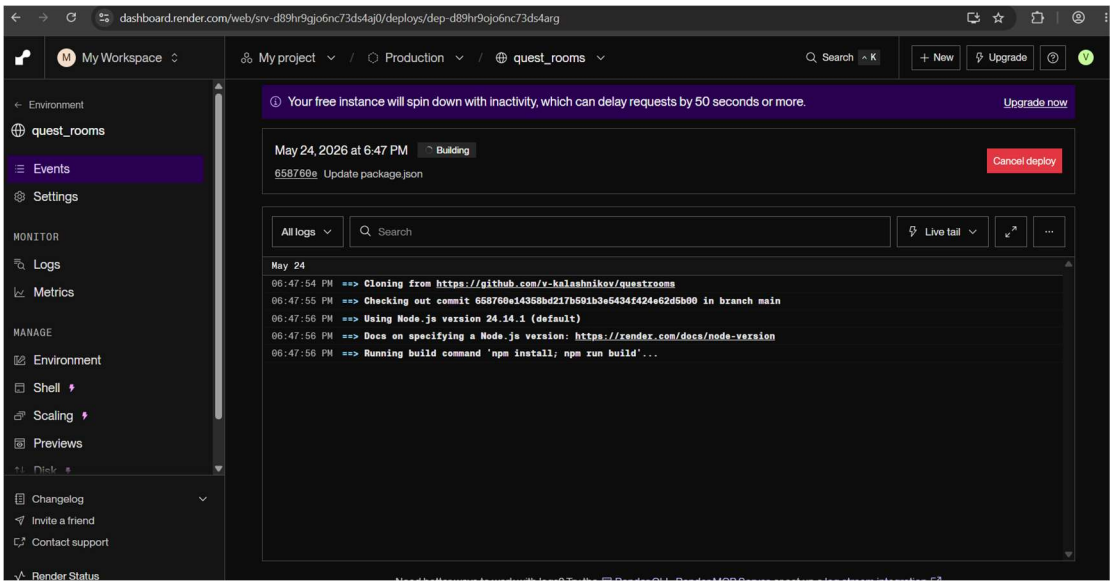

Рисунок 3.7 – Початок деплойменту

В процесі в логах буде видно, як виконались міграції, та останнім буде повідомлення про успішний деплоймент.

3.3 Аналіз якості коду, лінтинг та оптимізація продуктивності веб–сайту

Для забезпечення довгострокової підтримки кодової бази веб сайту «Quest rooms» та досягнення максимальної швидкості завантаження сторінок було впроваджено комплексний підхід до аналізу якості коду та оптимізації продуктивності.

Контроль якості коду на етапі розробки автоматизовано за допомогою інтеграції інструментів ESLint та суворої конфігурації TypeScript. Це дозволило виявляти синтаксичні помилки, потенційні витоки пам'яті та невідповідності архітектурним стандартам ще до етапу компіляції.

У межах проєкту було задіяно такі правила та інструменти статичного аналізу:

1. У конфігураційному файлі tsconfig.json активовано режим «strict»: true – тобто включено строгу типізацію. Це зобов'язує явно описувати типи для всіх об'єктів та параметрів функцій, повністю блокуючи використання небезпечного типу any. Це унеможливлює виникнення поширених помилок під час виконання коду, таких як «Cannot read property of undefined».

2. Використано автоматизований лінтинг з базовим конфіг eslint–config–next, розширений правилами для перевірки відповідності стандартам React–компонентів.

Eslint автоматично блокує:

– Наявність імпортованих, але невикористаних модулів або змінних (запобігання забрудненню пам'яті);

– Неправильне використання масивів залежностей у клієнтських хуках useEffect та useMemo;

– Порушення семантики HTML–розмітки та правил доступності інтерфейсів.

Окрім цього сам фреймворк Next.js 14 має вбудовані механізми оптимізації, які були повністю задіяні під час проєктування системи «Quest rooms» для досягнення найвищих балів у системі оцінювання Google Lighthouse.

Оптимізація графічного контенту була обов’язковим етапом через те, що веб сайт квест–кімнат містить велику кількість медіафайлів, класичні HTML–теги <img> було замінено на системний компонент <Image/> із модуля next/image. Це забезпечило:

1. Автоматичний ресайзинг, бо сервер Next.js на льоту стискає та масштабує зображення під фізичну роздільну здатність екрана клієнта.

2. Зображення автоматично конвертуються у формати нового покоління, що зменшує вагу файлів на 30–50% порівняно з JPEG/PNG без втрати

якості. Це особливо важливо з урахуванням використання безкоштовного хостингу з малою кількістю пам’яті.

3. Ледаче завантаження, яке означає, що зображення, які знаходяться поза зоною видимості екрана, не завантажуються доти, доки користувач не прокрутить сторінку до них, що суттєво економить трафік і прискорює початковий рендеринг.

Використання Tailwind CSS додатково оптимізувало продуктивність. Утилітарний підхід дозволив згенерувати єдиний CSS–файл, який містить лише ті класи, що реально використовуються в коді. Зайві стилі повністю видаляються на етапі збірки.

Завдяки використанню React Server Components, клієнт отримує чистий HTML. Весь важкий JavaScript код, необхідний для генерації сторінки, залишається і виконується на сервері. [6]

Для інтерактивних елементів, таких як форми або таблиця заявок OrdersTable, Next.js автоматично застосовує технологію автоматичного розділення коду. Браузер завантажує JavaScript скрипт для конкретної форми лише тоді, коли користувач переходить на сторінку з цією формою.

Для валідації розроблюваного веб сайту було виконано аналіз розгорнутої версії за допомогою інструменту Google Lighthouse. Він працює шляхом проведення автоматизованого аудиту веб-сторінки.

Результати аудиту представлені у вигляді звіту, який містить детальну інформацію про сильні та слабкі сторони оцінюваного сайту, а також конкретні рекомендації щодо покращення. [7]

Ефективність – оцінює швидкість рендерингу та візуалізації контенту. В даному випадку вони відповідають вимогам і мають високі показники, а саме 100 балів.

Результати для мобільної версії наведено на рисунку 3.8.

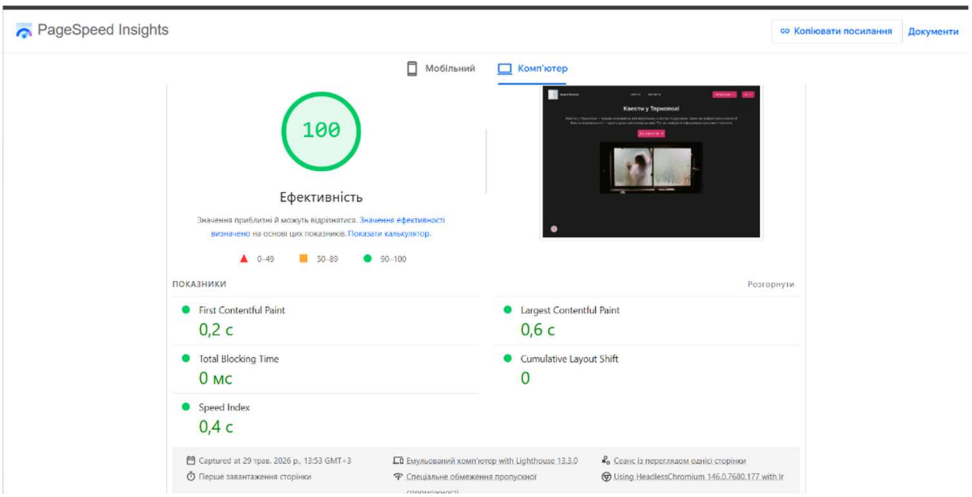

Рисунок 3.8 – Результати «Ефективності»

1. Доступність – перевіряє зручність сайту для людей з обмеженими можливостями та навігацію з клавіатури;

2. Оптимальні практики – аналізує відповідність стандартам безпеки. Результати для мобільної версії наведено на рисунку 3.9.

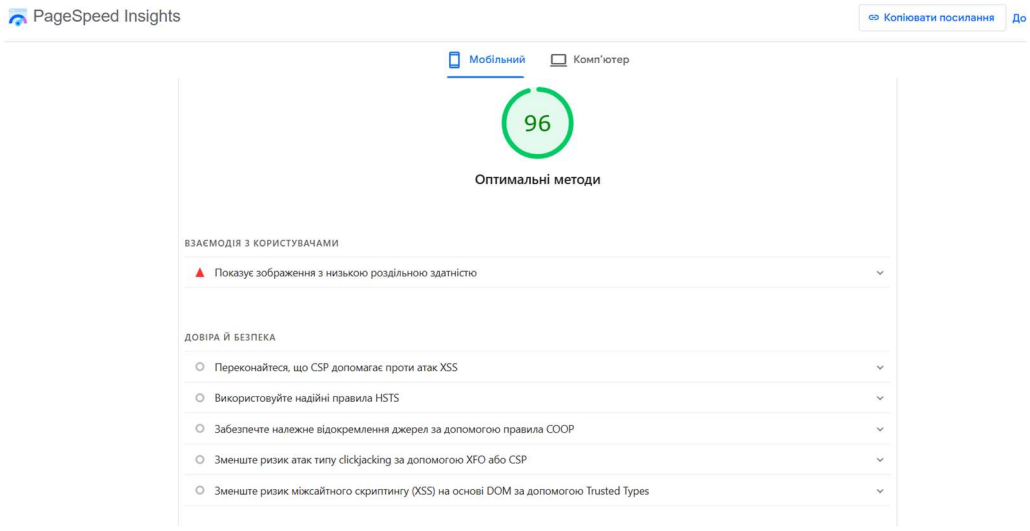

Рисунок 3.9 – Результати аналізу «Оптимальних методів»

3. Пошукова оптимізація – перевіряє наявність мета–тегів, семантичної розмітки та файлів конфігурації пошукових роботів.

Загальні результати аналізу веб версії наведено на рисунку 3.10.

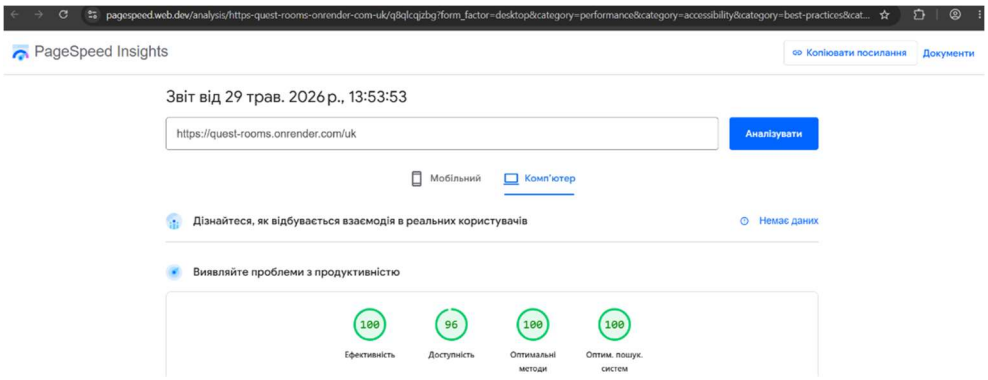

Рисунок 3.10 – Результати для веб версії

Результати тестування мобільної версії наведено на рисунку 3.11.

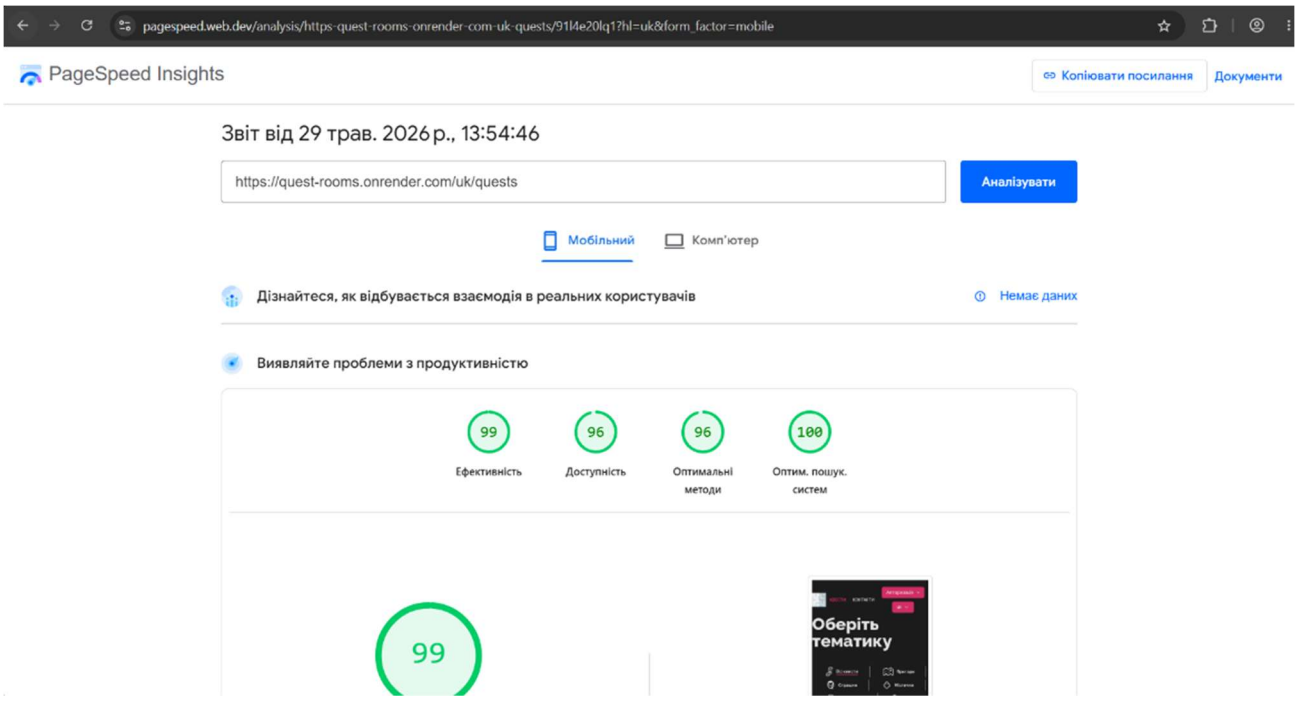

Рисунок 3.11 – Результати для мобільної версії

Впровадження суворих правил лінтингу разом із нативними інструментами оптимізації Next.js дозволило створити не просто функціональний, а швидкий, оптимізований та легкий у розширенні вебсервіс, готовий до високих реальних навантажень.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини дипломного проекту «Розробка веб-сайту квест-кімнат «Quest rooms»» є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності розробки, і прийняття рішення про її подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки або робіт в даному напрямку.

Об’єктом розробки є програма з веб інтерфейсом.

Розрахунок вартості розробки виконується в декілька етапів:

– описати технологічний процес розробки із зазначенням трудомісткості кожної операції;

– визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;

– визначити суму матеріальних затрат;

– обчислити витрати на електроенергію;

– розрахувати транспортні витрати;

– нарахувати суму амортизаційних відрахувань;

– визначити суму накладних витрат;

– скласти кошторис та визначити собівартість робіт;

– розрахувати ціну робіт;

– визначити економічну ефективність та термін окупності.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

Базуючись на даних із сформованого технічного завдання виділено 9 стадій розробки програмного продукту. Дані про витрати часу на проведення робіт наведені в таблиці 4.1.

Таблиця 4.1 - Середній час виконання робіт по обслуговуванню та стадії (операції) технологічного процесу

№ п/п	Назва операції(стадії)	Виконавець	Середній час виконання операції, год.
1	Підготовка	Кер. проекту	2
2	Формування та постановка задачі	Програміст	6
3	Розробка архітектури	Кер. проекту	7
4	Розробка алгоритму програми	Програміст	12
5	Написання коду програми	Програміст	68
6	Відлагодження програми	Програміст	17
7	Тестування програми	Тестувальник	13
8	Документування	Програміст	10
9	Здача програми	Програміст	3
		Кер. проекту	2
Разом			140

Отже, виходячи з даних таблиці, сумарний час виконання операцій технічного процесу становить 140 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Заробітна плата це - винагорода за виконану роботу відповідно до встановлених норм праці (норми часу, виробітку, обслуговування, посадові обов'язки). Вона встановлюється у вигляді тарифних ставок (окладів) і відрядних розцінок для робітників та посадових окладів для службовців.

Відповідно до Закону України «Про оплату праці» заробітна плата – це «винагорода, обчислена, як правило, у грошовому виразі, яку власник або уповноважений ним орган виплачує працівникові за виконану ним роботу».

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с \cdot К_г, \qquad (4.1)$$

де: T_c – тарифна ставка, грн.;

$K_г$ – кількість відпрацьованих годин.

Рекомендовані тарифні ставки: керівник проекту – 435 грн/год., програміст (джуніор)– 140 грн/год., тестер(джуніор) – 130 грн/год.

Отже основна заробітна плата становитиме:

– керівник проекту Зосн1 = 11 * 435 = 4785,00 грн.

– програміст: Зосн2 = 116 * 140 = 16240,00 грн.

– тестувальник: $З_{осн3}$ = 13 * 130 = 1690,00 грн.

Сумарна основна заробітна плата становить:

$$З_{осн.} = 4785{,}00 + 16240{,}00 + 1690{,}00 = 22715{,}00 \text{ грн}$$

Додаткова заробітна плата це – це винагорода за понаднормативну працю, трудові успіхи та винахідливість і за особливі умови праці. Вона включає доплати, надбавки, гарантії та компенсації, передбачені чинним законодавством, премії, пов'язані з виконанням виробничих завдань та функцій.

Додаткова заробітна плата становить 10 - 15% від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.}, \qquad (4.2)$$

де $K_{допл.}$ – коефіцієнт додаткових виплат працівникам, візьмемо 0,15.

Отже додаткова заробітна плата по категоріях працівників становить:

– керівника проекту Здод1 = 4785 * 0,12 = 574,20 грн;
– програміста Здод2 = 16240* 0,12 = 1948,80 грн;
– лаборанта Здод3 = 1690 * 0,12 = 202,80 грн.

Загальна додаткова заробітна плата становить:

$$З_{\text{дод.}} = 574{,}20 + 1948{,}80 + 202{,}80 = 2725{,}80 \text{ грн.} К_{\text{допл.}}$$

Звідси загальні витрати на оплату праці ($B_{о.п.}$) визначаються за формулою:

$$В_{\text{о.п.}} = З_{\text{осн.}} + З_{\text{дод.}}, \qquad (4.3)$$

$$В_{\text{о.п.}} = 22715{,}00 + 2725{,}80 = 25440{,}80 \text{ грн.}$$

Крім того, слід визначити відрахування на соціальні заходи:

– єдиний соціальний внесок – 22%

Отже, сума відрахувань на соціальні заходи буде становити:

$$ФОП = B_{\text{о.п}} \cdot 0{,}22, \qquad (4.4)$$

де ФОП – фонд оплати праці, грн.

$$ФОП = 25440{,}80 \cdot 0{,}22 = 5596{,}98 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці зведемо у таблицю 4.2.

Таблиця 4.2 - Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	Нарахув. на ФОП, грн.	Всього витрати на оплату праці, грн
		Тарифна ставка, грн.	К-сть від-працьов. год.	Фактично нарах. з/пл., грн.			
1	2	3	4	5	6	7	8
1	Кер.проекту	435	11	4785	574,20	-	-
2	Програміст	140	116	16240	1948,80	-	-
3	Тестуваль-ник	130	13	1690	202,80	-	-
Разом				22715,00	2725,80	5596,98	31037,78

Отже, загальні витрати на оплату праці становлять 31037,78 грн.

4.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$M_{Bi} = q_{i.} \cdot p_{i.}, \qquad (4.5)$$

де:

q_i – кількість витраченого матеріалу *i*-го виду;

p_i – ціна матеріалу *i*-го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum M_{Bi} \qquad (4.6)$$

Проведені розрахунки занесемо у таблицю 4.3.

Таблиця 4.3 - Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна 1-ці, грн.	Загальна сума витрат, грн.
1	CD-накопичувачі	шт.	1	17,00	17,00
2	Друк	лист.	100	3,0	300,00
3	Друк ватманів	лист.	4	120,00	480,00
Р а з о м					797,00

Отже, загальна сума матеріальних витрат на розробку програми становить 797,00 грн.

4.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S, \qquad (4.7)$$

де:

W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

В нашій системі є 1 ноутбук. Споживана потужність якого – 0,06 кВт/год., і кількість годин його роботи – 140. На час розрахунку вартість кіловат-години електроенергії становить 15,94 грн.

$$З_в = 0{,}06 \cdot 140 \cdot 15{,}94 = 133{,}90 \text{ грн.}$$

Загальні витрати на електроенергію становлять $З_в = 133{,}90$ грн.

4.5 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Комп'ютери та оргтехніка належать до четвертої групи основних фондів.

Амортизація на них нараховується лише в випадку, якщо їх вартість перевищує 2500 грн., а мінімально допустимі строки їх корисного використання 2 роки.

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{Б_В \cdot Н_А}{150\%}, \qquad (4.8)$$

де:

А – амортизаційні відрахування за звітний період, грн.;

$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

$Н_А$ – норма амортизації, %.

Для програмування було використано ноутбук вартістю 40000 грн. Враховуючи, що ноутбук працює над даним проектом 140 год.,

$$А_{пк} = \frac{40000 \cdot 0{,}04}{150} \cdot 140 = 1493{,}33 \text{ грн.}$$

4.6 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$Н_в = В_{о.п.} \cdot 0{,}2 \ldots 0{,}6, \qquad (4.9)$$

де $Н_В$ – накладні витрати.

Приймемо коефіцієнт накладних витрат – 40%.

$$Н_в = 25440{,}80 \cdot 0{,}4 = 10176{,}32 \text{ грн.}$$

Значення накладних витрат за таких умов складає 10176,32 грн.

4.7 Складання кошторису витрат та визначення собівартості НДР

Результати проведених вище розрахунків зведено у таблиці 4.4.

Таблиця 4.4 - Кошторис витрат

Зміст витрат	Сума, грн.	В % до загальної суми
Витрати на оплату праці	25440, 80	58,30
Відрахування на соціальні заходи	5596,98	12,83
Матеріальні витрати	797,00	1,83
Витрати на електроенергію	133,90	0,31
Амортизаційні відрахування	1493,33	3,42
Накладні витрати	10176,32	23,32
Собівартість	43638,33	100%

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$С_В = В_{о.п.} + В_{с.з.} + З_{м.в.} + З_е + А + Н_в \qquad (4.10)$$

Отже, собівартість дорівнює $С_В$ = 43638,33 грн.

4.8 Розрахунок ціни НДР

Ціну робіт можна визначити за формулою:

$$Ц = \frac{С_В \cdot (1 + Р_{рен}) + К \cdot В_{н.і.}}{К} \cdot (1 + ПДВ), \qquad (4.11)$$

де:

$Р_{рен.}$ – рівень рентабельності, (30 %);

К – кількість замовлень, од.;

$В_{і.н.}$ – вартість носія інформації, грн.;

ПДВ – ставка податку на додану вартість, (20 %).

$$Ц = 43638,33 * (1 + 0,3) * (1 + 0,2) = 68075,79 \text{ грн.}$$

Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ), та термін окупності (Ток).

$$ЧТВ = К_В + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t} \qquad (4.12)$$

де:

$К_В$ – затрати на проект;

$Г_п$ – грошовий потік за t – ий рік;

t – відповідний рік проекту;

i - величина дисконтної ставки (10…15%).

$$ЧТВ = -43638{,}33 + \frac{24437{,}46}{(1+0{,}1)} + \frac{24437{,}46}{(1+0{,}1)^2} + \frac{24437{,}46}{(1+0{,}1)^3} = 17134{,}03 \text{ грн.}$$

Термін окупності - це час, необхідний для того, щоб чистий прибуток від інвестиційного проєкту повністю покрив суму початкових вкладень.

Термін окупності визначається за формулою:

$$Т_{ок} = Т_{ПВ} + \frac{Н_В}{Г_{ПР}} \qquad (4.13)$$

де:

$Т_{ПВ}$ – період до повного відшкодування витрат, років;

$Н_В$ – невідшкодовані витрати на початок року, грн.;

$Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ок} = 2 + \frac{1226{,}20}{24437{,}46} = 2{,}1$$

Всі дані розрахунків внесено в зведену таблицю 4.5 техніко-економічних показників розробки веб-сайту.

Таблиця 4.5 - Техніко-економічні показники розробки програми

№ п/п	Показник	Значення
1.	Собівартість, грн.	43638,33
2.	Плановий прибуток, грн.	24437,46
3.	Ціна, грн.	68075,79
4.	Чиста теперішня вартість, грн.	17134,03
5.	Термін окупності, рік	2,1

4.9 Визначення економічної ефективності і терміну окупності капітальних вкладень

Підрахунок економічних показників інтернет-магазину показали що термін окупності складає 2,1 роки, ціна складає 68075,79 грн., прибуток даного продукту становить 24437,46 грн/ Отже, розробка веб-сайту є доцільною.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

5.1 Ергономіка робочої пози та профілактика захворювань опорно-рухового апарату розробника програмного забезпечення

Ергономіка - це наука про взаємодію людини з навколишнім середовищем, яка враховує особливості людської фізіології під час проєктування предметів, машин і систем. Мета ергономіки - створення ефективних, комфортних і безпечних умов роботи.

У межах ергономіки виокремилися такі основні напрями: фізична ергономіка, когнітивна та організаційна. Фізична ергономіка розглядає антропометричні, анатомічні, фізіологічні, біомеханічні характеристики та їх вплив на фізичну діяльність людини. Дослідження зосереджені на робочій позі, вантажних роботах, монотонних рухах, роботі, яка викликає розлади опорно-рухової системи; організації робочого місця, безпеці та здоров'ї людини.

На рисунку 5.1 показане нейтральне положення голови щодо природнього напрямку погляду й рекомендована відстань від очей до екрана комп'ютера, яка чим більше, тим краще.

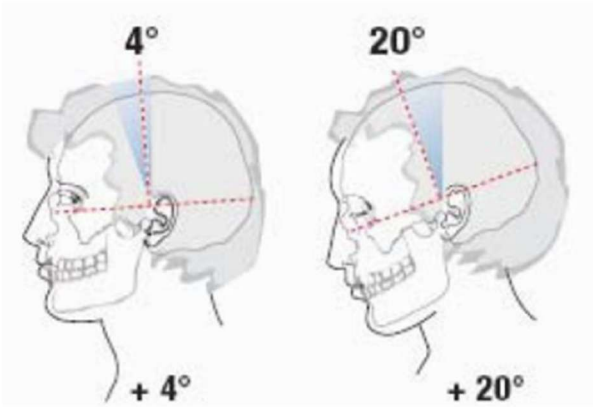

Рисунок 5.1 – Нейтральне положення голови щодо природнього напрямку погляду: а – нейтральне положення; б – максимальний нахил уперед відносно нейтрального положення

При необхідності можна збільшувати розмір шрифту, якщо рівень комфортності не вписується в загальні рекомендації. Фокусування на близьких об’єктах може викликати напруження очей. Щоб зосередитися на близьких об’єктах, екстраокулярні м’яза повертають очні яблука усередину, і ресничні м’яза працюють, утворюючи лінзу. [1]

Тривале розглядання близьких об’єктів приводить до зорової напруги й візуального дискомфорту. Одним з розв’язків є розташування близького об’єкта (екрана комп’ютера) набагато нижче рівня ока користувача. Хоча це може бути ефективним для зменшення напруги око користувача, це, на жаль, може змусити користувача згинати шию, викликаючи скелетно-м’язовий дискомфорт. На щастя, установка монітора на мінімальній відстані огляду й на відповідній висоті щодо очей користувача є ефективною в усуненні й візуального, і скелетно-м’язового дискомфорту для користувачів комп’ютерів.

Сучасні дослідження й технічні стандарти рекомендують, щоб висота монітора визначалася висотою рівня ока користувача; верхня частина екрана 56 повинна розташовуватися не вище рівня очей, а центр екрана повинен перебувати приблизно на 15 – 30º нижче рівня очей користувача. Простим способом оцінки кута між рівнем очей і центром екрана є вимір відстані між очима користувача й екраном, а потім – відстані нижче горизонтального рівня око до центру екрана. Відстань до центру екрана нижче рівня ока повинна становити приблизно половину відстані огляду.

Одним з основних напрямків ергономіки є виконання фізіологічних і психологічних вимог при конструюванні машин та іншого устаткування, організації та плануванні робочих місць. При конструюванні машин повинні бути передбачені заходи щодо усунення зайвих рухів працюючого, ліквідації нахилів тулуба і переходів. Правильне розташування і компонування робочого місця, забезпечення зручної пози і свободи трудових рухів, використання обладнання, що відповідає вимогам ергономіки та інженерної психології,

забезпечують найбільш ефективний трудовий процес, зменшують стомлюваність і запобігають небезпеці виникнення професійних захворювань.

Робота користувача ПК вимагає тривалого статичного напруження м'язів спини, шиї, рук і ніг, що призводить до втоми і специфічних скарг. Ушкодження хребта є результатом недостатнього рівня ергономічності робочого місця користувача, тобто крісло неправильно підтримує згин хребта. Плечі і шия напружуються і затікають унаслідок неприродного положення і виникають болі в ділянці шиї, спини і голови. В середньому працівник, який використовує ПК, просиджує в такому положенні за все своє життя до 80000 годин (8 років).

Неправильне положення рук при введенні даних за допомогою клавіатури (зап'ястя при наборі підняті вверх) призводить до перетискання нервів у вузьких місцях зап'ястя (тунель Карпаля).

Синдром RSI (хронічне розтягнення зв'язок) - це пошкодження, що виникає в результаті постійного напруження м'язів кистей рук як результат неправильно обладнаного з погляду ергономіки робочого місця при використанні ПК. Це хронічне захворювання може непомітно розвиватися протягом декількох років. Такі перевантаження призводять до перенапруження всієї м'язової системи людини. Найбільш небезцечним є те, що внаслідок концентрації уваги на екрані монітора притуплюється своєчасне попередження про болі, які є тривожним сигналом для тіла. Захворювання рук,і кистей рук спостерігається у працюючих за ПК у 7-12 разів частіше, ніж у інших, і досить часто помилково діагностується як запалення сухожиль. [3]

Основою профілактики є ергономічне облаштування робочого місця, регулярні фізичні вправи та динамічні паузи для зняття статичного напруження. Це допомагає запобігти тунельному синдрому зап'ястка, остеохондрозу, грижам міжхребцевих дисків та болям у шиї.

Вправи для відновлення гостроти зору та профілактики опорно-рухової системи:

– Масаж закритих очей внутрішньою поверхнею долоні 20-60 секунд.

– Сильно заплющити очі на 3-5 секунд, відкрити очі на 1-2 секунди, знову заплющити, зробити цю вправу декілька раз.

– Сконцентрувати погляд на кінчику носа на 3-5 секунд, розвести очі, знову звести на кінчик носа.

– Кругові оберти очима ліворуч, праворуч.

– Розминати м'язи спини, зменшувати навантаження на міжхребцеві диски.

– Не перевантажувати хребет, не робити різких рухів.

– Тримати спину рівно, міняти положення тіла.

– Робити гімнастику під час перерви, кожні 40-45 хвилин роботи за комп’ютером.

Профілактика тунельного синдрому:

– Сильно стиснути пальці в кулак, та сильно розтиснути.

– Стиснути кулаки, повертати спочатку в одну сторону, потім в іншу.

– Притиснувши долоні одну до одної, розвести лікті в сторони. Передпліччя в такому положенні знаходяться паралельно підлозі. Опустити долоні як найнижче не розмикаючи і залишаючи лікті високо. Зробити цю вправу декілька раз.

5.2 Психофізіологічні аспекти праці: вплив стрес-факторів на безпеку розробки в умовах обмеженого часу в ІТ-сфері

При аналізі психофізіологічних небезпечних та шкідливих чинників велике значення приділяється стресу, що виникає внаслідок тривалого впливу на працюючого комбінованої дії психоемоційних перевантажень та небезпечних виробничих чинників.

На сьогоднішній день психологи відзначають, що все гострішою постає проблема емоційного стану людей через підвищені стрес і тривожність. Саме ці

два подразники не дають змогу працювати ефективно та із задоволенням і відчувати психологічне благополуччя.

Дослідження, проведене компанією Mental Health America, показало, що 72% працівників IT-компаній відчувають стрес на роботі, а 58% повідомляють про ознаки вигорання.

Дослідження Indeed показало, що 60% працівників IT-компаній думали про те, щоб залишити роботу через проблеми з ментальним здоров'ям.

Важливість стресостійкості для працівників IT-компаній є зрозумілою, оскільки ця галузь характеризується відсутністю стабільності, постійними дедлайнами, великим обсягом відповідальності, ненормованим робочим графіком, постійною сидячою роботою за комп'ютером. У сучасному інформаційному суспільстві, де технологічний прогрес швидко розвивається, працівники IT-сфери знаходяться під постійною напругою. Уміння ефективно управляти стресом є фактором для досягнення успіху в IT-індустрії. Висока конкуренція, швидкі темпи розвитку технологій, постійна необхідність оновлення знань і навичок та інші 200 фактори створюють умови для появи стресових ситуацій.

Робота за ПК — це робота з особливо відчутною монотонністю: більше ніж 600 однакових дій упродовж 75% робочого часу за 1 годину. Монотонність роботи, неергономічність робочого місця, електромагнітні випромінювання призводять до захворювань за-гальноневротичного характеру у вигляді підвищеної загальної втоми, головного болю, відчуття важкості голови, поганого сну. Стійкі нервово-психічні порушення у вигляді підвищеної роздратованості, відчуття неспокою, метушливості (збуджений тип), депресивних станів, загальної скутості в роботі, зменшення швидкості реакцій (гальмівний тип), ймовірно, викликані електромагнітними хвилями, які випромінює ПК і монітор

Стрес характеризують як захисне явище, як вісник захворювання, як причину порушень низки життєво важливих психофізіологічних функцій.

Стрес проявляється як необхідна і корисна реакція організму на різке збільшення загального зовнішнього навантаження. Він характеризується зростанням біоелектричної активності мозку, підвищенням частоти серцебиття, ростом потоку крові, розширенням кровоносних судин, збільшенням вмісту лейкоцитів у крові, тобто цілим рядом фізіологічних змін в організмі, що сприяють підвищенню його енергетичних можливостей, успішності виконання складних і небезпечних дій. Тому стрес є не тільки доцільною захисною реакцією людського організму, але й механізмом, який сприяє успіху трудової діяльності в умовах перешкод, труднощів і небезпек.

Між рівнем стресу і активацією нервової системи, яка породжується ним, з одного боку, та результативністю трудової діяльності з іншого, немає пропорційної залежності. Відомо, що з ростом активації нервової системи до певного рівня продуктивність праці підвищується, тоді як при подальшому зростанні активації вона починає падати, і рівень небезпеки зростає.

Як зазначалося раніше, стресові впливи можуть стати причиною виникнення фізіологічних і психологічних змін, що призводять до небезпечних ситуацій та нещасних випадків.

Фізіологічні порушення можуть супроводжуватися розладами нервової та серцево-судинної систем, шлунково-кишкового тракту та ін.

До психологічних розладів належать агресивність, фрустрація, нервозність, роздратування, тривога, нерішучість, швидкий розвиток утоми тощо. Фрустрація (лат. «обман», марне чекання) – мотивація досягти мети за існуючої сильної перешкоди.

Крім того, стрес є причиною багатьох психосоматичних захворювань: психозів, неврозів, захворювань судин мозку, серцево-судинних захворювань та інфаркту міокарда, гіпертонічної хвороби, виразково-дистрофічних уражень шлунково-кишкового тракту, нейроциркуляторної дистонії, зниження імунітету, онкологічних захворювань. Стрес впливає на статеві функції, генетичний апарат клітин, призводячи до вроджених порушень розвитку дітей, тощо. Вчені

висловлюють припущення про існування зв’язку між стресовими навантаженнями та спонтанним абортом.

З точки зору медицини, для профілактики, попередження та реабілітації наслідків психоемоційного стресу рекомендується застосовувати вправи, що включають психотерапію, фізичні, водно-повітряні процедури, фізіотерапевтичні процедури, масаж, адекватне харчування, приймання вітамінів та мінеральних речовин, релаксуючу музику та вправи, медитацію, аутогенне тренування тощо. [4]

5.3 Гігієнічне нормування та використання систем штучного освітлення для організації робочого місця веброзробника

У виробничих приміщеннях на робочих місцях з комп'ютерною технікою мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й швидкості руху повітря

У приміщеннях з КТ має бути забезпечений 3-кратний обмін повітря за годину. Для забезпечення постійних параметрів мікроклімату (температури, вологості, швидкості руху і чистоти повітря) у приміщеннях можуть бути встановлені побутові кондиціонери типу БК-1500, БК-2000, БК-2500 та ін.

Вимоги до освітлення приміщень та робочих місць Приміщення з КТ повинні мати природне та штучне освітлення. При незадовільному освітленні знижується продуктивність праці користувачів КТ, можлива поява короткозорості, швидка стомлюваність.

Система освітлення повинна відповідати таким вимогам:

– освітленість на робочому місці повинна відповідати характеру зорової роботи, який визначається трьома параметрами: об'єктом розрізнення - найменшим розміром об'єкта, що розглядається на моніторі персонального комп'ютера (ПК) та робочої станції (РС); фоном, який характеризується коефіцієнтом відбиття; контрастом об'єкта і фону;

– необхідно забезпечити достатньо рівномірний розподіл яск^ равості на робочій поверхні монітора, а також в межах навколишнього простору;

– на робочій поверхні повинні бути відсутні різкі тіні;

– у полі зору не повинно бути відблисків (підвищеної яскравості поверхонь, які світяться та викликають осліплення);

– величина освітленості повинна бути постійною під час роботи;

– слід обирати оптимальну спрямованість світлового потоку і необхідний склад світла.

Штучне освітлення в приміщенні і на робочих місцях повинно забезпечувати хорошу видимість інформації на екрані, машинописного і рукописного тексту, оптимальні співвідношення яскравості робочих і оточуючих поверхонь, виключення відбиття від екрана і клавіатури.

Штучне освітлення має здійснюватися системою загального рівномірного освітлення, а в разі необхідності і комбінованого освітлення. При цьому світильники місцевого освітлення слід встановити таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана не повинна перевищувати 300 лк, крім того вони повинні мати просвічуючий відбивач із захисним кутом не менше 40 градусів.

Нормована освітленість на поверхні робочого столу в зоні розміщення документів має бути – 300–500 лк.

Система загального освітлення має становити суцільні або переривчасті лінії світильників, розташовані з боку робочих місць (переважно ліворуч), паралельно лінії зору працюючих.

Для штучного освітлення застосовують люмінесцентні лампи типу ЛБ або світильники серії ЛПО із дзеркальними ґратами, укомплектовані високочастотними пускорегулювальними апаратами. Лампи розжарювання можна застосовувати для місцевого освітлення. Допускається використання світильників П, Н і В класу світлорозподілу.

Слід передбачити обмеження прямої близькості від джерел освітлення, при цьому яскравість відблисків на екрані ВДТ не повинна перевищувати 40 кд/м^2, а яскравість стелі – 200 кд/м^2. Показник освітленості не повинний перевищувати у виробничих приміщеннях – 20, а показник дискомфорту в адміністративно-громадських приміщеннях – 40. При цьому співвідношення яскравості робочих поверхонь не повинна перевищувати 3:1, а робочих поверхонь і поверхонь стін – 5:1. Зменшенню відблисків сприяє застосування захисних дашків та приекранних фільтрів.

Основні світлотехнічні характеристики:

– коефіцієнт запасу = 1,4;

– коефіцієнт пульсацій < 5% .

Для забезпечення нормованих значень освітленості шибки і світильники необхідно чистити двічі на рік. З метою усунення відблисків поверхня екрану обробляється різними засобами (кислотою, нанесенням розсіювальних покриттів тощо) або використовують спеціальні фільтри (скляні, пластмасові, сітчасті).

Таким чином, для створення комфортних умов праці, запобігання втомленості очей і запобіганню професійним захворюванням освітлення повинно:

– відповідати нормованим значенням освітленості на кожному робочому місці;

– бути рівномірним і постійним;

– не створювати тіней;

– не засліплювати;

– зменшувати до мінімуму стробоскопічний ефект;

– дотримувати необхідний контраст об'єктів і фону на екрані ВДТ;

– забезпечуватися правильно підібраними світильниками;

– уникати відблисків на екрані;

– зменшувати тепловий ефект від інсоляції. [2]

ВИСНОВКИ

Під час виконання дипломного проєкту було успішно розроблено та розгорнуто веб-сайт квест-кімнат «Quest Rooms». Проєкт було розміщено на віддаленому хостингу, що забезпечило його доступність у мережі інтернет. Якість та стабільність веб–сайту були підтверджені внутрішнім тестуванням, перевіркою коректності введення даних та тестуванням адаптивності інтерфейсу під різні типи пристроїв.

Сайт розроблено в повній відповідності до технічного завдання, використовуючи сучасний, надійний та ефективний стек технологій.

Для реалізації проєкту було обрано сучасний фреймворк Next.js 14 у зв'язці з мовою TypeScript. Це дозволило створити швидкий, оптимізований додаток із високим рівнем безпеки коду, оскільки сувора типізація TypeScript допомагає уникати помилок ще на етапі написання програми.

Робота з даними: Побудову надійної архітектури та збереження інформації про квести, користувачів і бронювання забезпечено реляційною СУБД PostgreSQL. Взаємодія між кодом сайту та базою даних реалізована через Prisma ORM, що значно прискорило розробку завдяки роботі з таблицями як з об'єктами.

Отже, під час виконання дипломного проєкту було успішно закріплено та заглиблено знання зі створення сучасних вебсайтів на базі фреймворку Next.js, проєктування реляційних баз даних, роботи з ORM–системами, а також практичні навички використання інструментів контролю версій Git та розгортання готового веб–сайту в мережі. Створений сайт повністю вирішує завдання автоматизації послуг «Quest Rooms», роблячи процес вибору та бронювання квест-кімнат максимально зручним для клієнтів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Абракітов В. Е., Ткаченко І.О. «Ергономіка робочих місць». Харків : ХНУМГ ім. О. М. Бекетова, 2017 – 78 с.
2. Вимоги до освітлення робочого місця при роботі з пк. Штучне освітлення за устроєм і призначенням яке буває URL: https://studfile.net/preview/10057092/page:11/ (дата доступу: 20.05.2026)
3. Охорона праці при роботі з комп'ютерною технікою URL: https://studfile.net/preview/12714142/page:15/ (дата доступу: 20.05.2026)
4. Охорона праці та цивільний захист: конспект лекцій URL: https://ela.kpi.ua/server/api/core/bitstreams/8a99aa20-e49f-4c01-b27d-bf10ed8e1fe5/content (дата доступу: 21.05.2026)
5. Розбираємо фреймворк Next.JS: визначення, призначення та переваги URL: https://wezom.com.ua/ua/blog/rozbirajemo-freymvork-nextjs-viznachennya-priznachennya-ta-perevagi (дата звернення: 20.05.2026)
6. Чому React Server Components — майбутнє веброзробки URL: https://dou.ua/forums/topic/46138/ (дата звернення: 10.05.2026)
7. Що таке Google Lighthouse і як він може покращити UX вашого сайту URL: https://tto-studio.com.ua/shcho-take-google-lighthouse-i-yak-vin-mozhe-pokrashchyty-ux-vashoho-saytu/ (дата звернення: 19.05.2026).
8. Basic usage URL: https://zod.dev/ecosystem (дата звернення: 19.05.2026).
9. PostgreSQL URL: https://www.prisma.io/docs/orm/core-concepts/supported-databases/postgresql (дата звернення: 13.05.2026).
10. Quest.lviv URL: https://quest.lviv.ua/ (дата звернення: 10.05.2026).
11. Questroom URL: https://questroom.com.ua/(дата звернення: 10.05.2026)
12. Why Vitest URL: https://vitest.dev/ (дата звернення: 25.05.2026)

ДОДАТКИ

Додаток А. Код route.ts

```
import { NextResponse } from "next/server";
import db from "@/modules/db";

export async function POST(request: Request) {
  const body = await request.json();
  const { email, password } = body;

  // Пошук користувача в базі даних за email
  const user = await db.user.findFirst({
    where: { email },
  });

  // Перевірка на існування користувача
  if (!user) {
    return NextResponse.json(
      { message: "Невірний email" },
      { status: 401 }
    );
  }

  // Перевірка відповідності пароля
  const passwordMatch = password === user.password;

  if (!passwordMatch) {
    return NextResponse.json(
```

```
      { message: "Невірний пароль" },
      { status: 401 }
    );
  }

  // Успішна відповідь із поверненням об'єкта користувача
  return NextResponse.json(user, {
    status: 200,
  });
}
```

Додаток Б. Тести findErrors

```
import { describe, expect, it } from "vitest";
import { findErrors } from "./findErrors";
import { z } from "zod";

describe("findErrors", () => {
  it("returns an empty array when errors is a string", () => {
    expect(findErrors("title", "Some error message")).toEqual([]);
  });

  it("filters Zod issues by field path and returns messages", () => {
    const issues: z.ZodIssue[] = [
      { code: "invalid_type", path: ["title"], message: "Title is required" },
      { code: "too_small", path: ["description"], message: "Description is too short" },
      { code: "invalid_type", path: ["title", "nested"], message: "Nested title failed" },
    ];

    expect(findErrors("title", issues)).toEqual(["Title is required", "Nested title failed"]);
  });
});
```