**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інженерії програмного забезпечення**

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____Є. О. Давиденко

*підпис*

**« » _____20___р.**

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

# СЕРВІС БРОНЮВАННЯ ГОТЕЛІВ НА ОСНОВІ ТЕХНОЛОГІЙ REACT ТА MONGODB

Спеціальність «Інженерія програмного забезпечення»

121 – КРМ – 608м.21810124

**Здобувач:** _____ В. М. Хоменко

*підпис*

«__» _____ 20__ р.

**Керівник: д-р техн. наук, проф.** _____ І. М. Журавська

*підпис*

«__» _____ 20__ р.

**Миколаїв – 2024**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інженерії програмного забезпечення**

ЗАТВЕРДЖУЮ
Зав. кафедри
_____ Є. О. Давиденко
«_____» _____ 2023 р.

# ЗАВДАННЯ

## на виконання магістерської кваліфікаційної роботи

**Видано студенту групи 608 факультету комп'ютерних наук**

_____Хоменку Віктору Максимовичу_____

*(прізвище, ім'я, по батькові студента)*

1. Тема кваліфікаційної роботи:

___Сервіс бронювання готелів на основі технологій React та MongoDB_____

Затверджена наказом по ЧНУ від «11» листопада 2023 р. № 234

2. Строк представлення кваліфікаційної роботи «____» _____ 2024 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є прототип вебсервісу з підвищеною ефективністю та зручністю процесу бронювання готелів онлайн.

4. Перелік питань, що підлягають розробці

− дослідження предметної галузі та аналіз застосунків-аналогів;

− визначення проблем та пошук рішень;

− моделювання та проєктування ПЗ;

− розробка функціоналу сервісу бронювання готелів;

− впровадження технологій React та MongoDB;

− тестування прототипу вебсервіс для бронювання готелів;

− аналіз результатів розробки.

5. Перелік графічних матеріалів

Презентація.

Керівник роботи д-р техн. наук, проф. Журавська Ірина Миколаївна
*(посада, прізвище, ім'я, по батькові)*

_____
*(підпис)*

Завдання прийнято до виконання
Хоменко Віктор Максимович
*(прізвище, ім'я, по батькові студента)*

_____
*(підпис)*

Дата видачі завдання «___» _____ 2023 р.

**КАЛЕНДАРНИЙ ПЛАН**
**виконання кваліфікаційної магістерської роботи**

Тема:_ Сервіс бронювання готелів на основі технологій React та MongoDB

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1. | Розробка та затвердження завдання на виконання МКР | *01.09.2023* | *15.09.2023* | *Виконав* |
| 2. | Огляд літератури за темою роботи | *15.09.2023* | *01.10.2023* | *Виконав* |
| 3. | Складання календарного плану МКР | *01.10.2023* | *03.10.2023* | *Виконав* |
| 4. | Аналіз предметної області | *05.10.2023* | *31.10.2023* | *Виконав* |
| 5. | Розробка проєктних рішень | *31.10.2023* | *10.11.2023* | *Виконав* |
| 6. | Моделювання | *10.11.2023* | *15.11.2023* | *Виконав* |
| 7. | Розробка ПЗ | *15.11.2023* | *20.12.2023* | *Виконав* |
| 8. | Перевірка працездатності, тестування та апробація розробленого ПЗ | *20.12.2023* | *05.01.2024* | *Виконав* |
| 9. | Аналіз результатів тестування | *05.01.2024* | *10.01.2024* | *Виконав* |
| 10. | Розробка керівництва користувача | *10.01.2024* | *13.01.2024* | *Виконав* |
| 11. | Відгук керівника МКР | *05.02.2024* | *15.02.2024* | *Виконав* |
| 12. | Оформлення МКР та презентації | *07.02.2024* | *08.02.2024* | *Виконав* |
| 13. | Попередній захист | *05.02.2024* | *11.02.2024* | *Виконав* |
| 14. | Рецензування | *12.02.2024* | *18.02.2024* | *Виконав* |
| 15. | Завершення оформлення МКР та презентації | *01.02.2024* | *15.02.2024* | *Виконав* |
| 16. | Захист кваліфікаційної роботи | *__.02.2024* | *__.02.2024* | *Виконав* |

Розробив здобувач Хоменко Віктор Максимович_____ _____
(*прізвище, ім'я, по батькові)* *(підпис)*
«__» _____ 20__ р.

Керівник роботи зав. каф. КІ, проф. Журавська Ірина Миколаївна____ _____
*(підпис)*

*«__» _____ 20__ р.*

**АНОТАЦІЯ**
**магістерської кваліфікаційної роботи**
**студента групи 608 ЧНУ ім. Петра Могили**

**Хоменка Віктора Максимовича**
**Тема: «Сервіс бронювання готелів на основі технологій React та MongoDB»**

Актуальність даного дослідження обумовлена необхідністю створення ефективного та зручного сервісу для бронювання готелів. В сучасному світі швидкість і якість обслуговування клієнтів у готельному бізнесі відіграють ключову роль.

Об'єкт кваліфікаційної роботи: процес бронювання готелів та розробки вебсервісів.

Предмет кваліфікаційної роботи: засоби та технології розробки вебсервісів для бронювання готелів.

Мета кваліфікаційної роботи: підвищення ефективності та зручності процесу бронювання готелів онлайн за допомогою розробки спеціалізованого вебсервісу.

У першому розділі представлено аналіз предметної області та існуючих сервісів бронювання готелів. У другому розділі описана розробка проєктних рішень, що забезпечують специфікацію вимог. У третьому розділі описано архітектуру ПЗ, вибір технологій та мови програмування, вибір компонентів ПЗ. У четвертому розділі описано процес реалізації сервісу бронювання готелів.

В результаті виконання кваліфікаційної роботи магістра було реалізовано вебсервіс для бронювання готелів.

Кваліфікаційна робота магістра викладена на __ с., містить __ розділи, __ рис., __ табл., __ джерел в переліку посилань, ___ додатка.

***Ключові слова****: бронювання готелів, вебсервіс, HTML, CSS, Yarn, JavaScript, TypeScript, SPA, React, Next.js, NEST, MongoDB.*

# ABSTRACT

**Master's Thesis of a student of group 608 Petro Mohyla BSNU**

**Khomenko Victor Maksymovych**

**Topic: «Hotel booking service based on React and MongoDB technologies»**

The relevance of this study is driven by the need to create an efficient and convenient service for hotel reservations. In the modern world, the speed and quality of customer service in the hotel business play a key role. The object of the qualification work is the processes of hotel booking and web application development.

The subject of the qualification work is tools and technologies for the development of web services for hotel reservations.

The purpose of the qualification work is to increase the efficiency and convenience of the online hotel booking process by developing a specialized web service.

The first chapter presents an analysis of the subject area and existing hotel booking services. The second section describes the development of design solutions that ensure the requirements specification. The third describes the software architecture, the choice of technologies and programming languages, and the selection of software components. The fourth chapter describes the process of implementing the hotel booking service.

As a result of the master's qualification work, the web service for hotel reservations was implemented.

The master’s thesis is presented on __ pages, it contains __ sections, __ figures, __ tables, __ sources in the list of references, __ appendices.



***Keywords:*** *hotel reservation, web application, HTML, CSS, Yarn, JavaScript, TypeScript, SPA, React, Next.js, NEST, Mongo*

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

КРМ – кваліфікаційна робота магістра

ПЗ – програмне забезпечення

СКБД – система керування БД

CSS – Cascading Style Sheets

FCP – First Contentful Paint

GDPR – General Data Protection Regulation

HTML – HyperText Markup Language

ISR – Incremental Static Regeneration

LCP – Largest Contentful Paint

NPM – Node Package Manager

ORM – Object-Relational Mapping

SSG – Static Site Generation

SSR – Server-Side Rendering

TTFB – Time to First Byte

TTI – Time to Interactive

USPs – Unique Selling Propositions, Unique Selling Points

UX – User Experience

# ВСТУП

У сучасному світі, де цифрові технології впливають на кожен аспект нашого життя, індустрія туризму та гостинності переживає значні зміни. Особливу увагу привертають онлайн-сервіси бронювання готелів, які відіграють ключову роль у спрощенні процесу планування подорожей. Ця дипломна робота присвячена розробці такого сервісу, з використанням сучасних технологій React і MongoDB, які дозволяють створювати високопродуктивні та легкі у використанні вебзастосунки.

Актуальність даного дослідження випливає з зростаючого попиту на ефективні цифрові рішення у сфері туризму. Мета роботи – розробити сервіс, який би був не тільки функціональним і надійним, але й зручним та інтуїтивно зрозумілим для користувачів. Застосування технологій React і MongoDB дозволяє реалізувати ці вимоги, забезпечуючи швидкість роботи і гнучкість управління даними.

Ця робота також підкреслює важливість інновацій у веброзробці та базах даних для створення ефективних рішень у туристичній галузі. Завдяки дослідженню і розробці такого сервісу можливо не тільки відповісти на поточні потреби ринку, але й внести свій вклад у розвиток цифрового туризму.

У підсумку, ця дипломна робота відкриває шлях для нових можливостей у створенні вебсервісів бронювання готелів, демонструючи потенціал сучасних технологій у поліпшенні якості туристичних послуг.

**Об’єкт** кваліфікаційної роботи – процес розробки та реалізації вебсервісу для бронювання готелів.

**Предметом** кваліфікаційної роботи – програмні засоби та методи створення ефективного вебсервісу з використанням сучасних технологій, таких як React та MongoDB.

**Мета** кваліфікаційної роботи – підвищення ефективності та зручності процесу бронювання готелів онлайн за допомогою розробки сучасного,

інтуїтивно зрозумілого та надійного спеціалізованого вебсервісу для бронювання готелів.

Для досягнення цієї мети було поставлено наступні задачі:

– провести аналіз ринку вебсервісів для бронювання готелів та визначити основні потреби користувачів;

– аналізувати сучасні методології розробки вебзастосунків та вибрати оптимальні для даного проєкту;

– вивчити особливості та можливості технологій React та MongoDB у контексті розробки вебсервісів;

– розробити архітектуру та дизайн вебсервісу, забезпечуючи зручний інтерфейс та високу швидкість роботи;

– реалізувати серверну частину вебсервісу з використанням MongoDB для ефективного управління даними;

– розробити клієнтську частину з використанням React, зосередившись на зручності інтерфейсу та інтуїтивності користування;

– провести комплексне тестування вебсервісу, щоб забезпечити його надійність та безпеку.

Практична цінність роботи полягає в тому, що вона відкриває нові можливості для покращення системи бронювання готелів та визначення основних потреб користувачів. Завдяки використанню технологій React та MongoDB можливо створити вебсервіс, який забезпечить ефективний моніторинг та аналіз інформації щодо потреби в готельних послугах. Це дозволить оптимізувати завантаження номерів/поверхів готелів, забезпечуючи точне визначення потреб постояльців та вчасну реакцію на зміни в попиті.

Робота пройшла апробацію під час XV Міжнародної науково-практичній конференції «Free and Open Source Software» (13–14 лютого 2024 р., м. Харків, Національний економічний університет імені Семена Кузнеця) [1].

# 1 АНАЛІЗ РИНКУ СЕРВІСІВ БРОНЮВАННЯ ГОТЕЛІВ

## 1.1 Опис предметної галузі

У наш час Інтернет і мобільні пристрої стали невід'ємними атрибутами нашого повсякденного життя, сприяючи зростанню попиту на онлайн-сервіси. Бронювання готелів через Інтернет також не стало винятком, перетворившись на ключовий елемент сучасної подорожі. Платформи онлайн-бронювання надають потужний інструмент для ретельного планування подорожі. Користувачі можуть з легкістю переглядати, порівнювати ціни та бронювати готелі з будь-якої точки світу, роблячи процес бронювання зручнішим і доступнішим.

У 2021 році обсяг світового ринку онлайн-сервісів для бронювання подорожей оцінювався в 519,1 мільярда доларів США і, як очікується, зростатиме зі середньорічним темпом приросту (CAGR) 9,0 % у період з 2022 р. по 2030 р. Зростання ринку можна пояснити зміною споживчої поведінки, зростаючим впливом соціальних мереж, збільшенням наявного доходу та зростаючою схильністю до пригодницьких подорожей по всьому світу. Більше того, зростання кількості мандрівників по всьому світу є рушійною силою індустрії в останні кілька років. Згідно зі статистикою, опублікованою

Всесвітньою туристичною організацією ООН, та розрахунками співробітників МВФ, кількість міжнародних туристів зросла до понад 1,5 мільярда до кінця 2019 року з 680 мільйонів у 2000 році. Розширення онлайн-платформ і надійних продавців туристичних путівок, готелів та інших послуг по всьому світу відкриває нові можливості для галузі, що, як очікується, прискорить її зростання впродовж прогнозованого періоду [1].

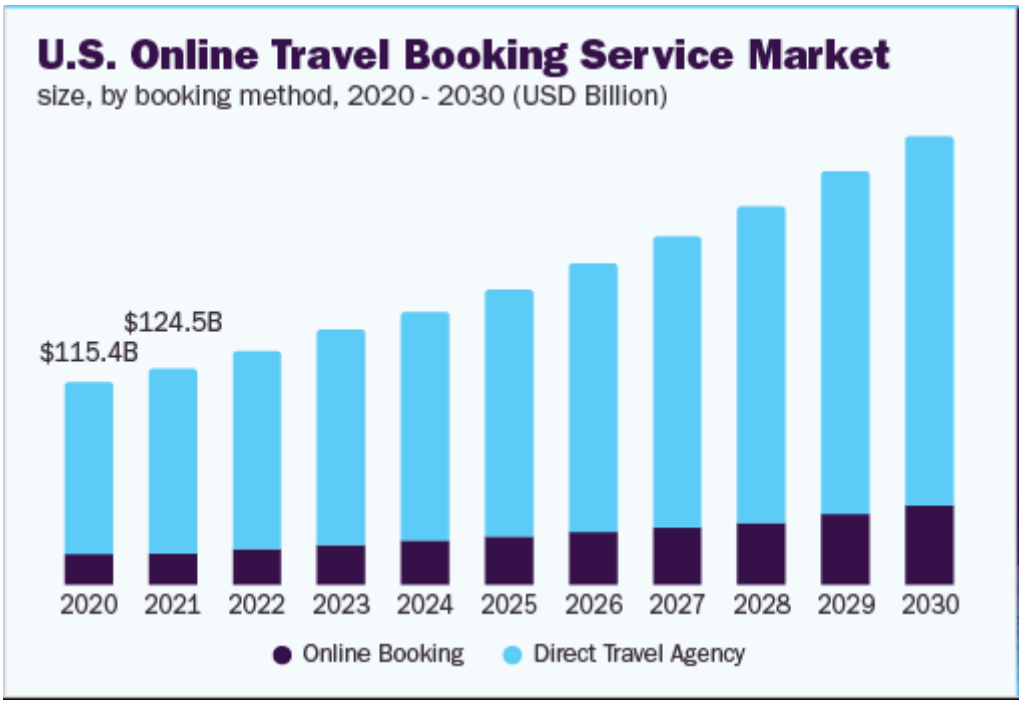


Рисунок 1.1 – Обсяг світового ринку онлайн-сервісів

Вибір готелю клієнтом залежить від безлічі факторів. Очікується, що номери будуть затишними, чистими і добре обладнаними з необхідними зручностями. Місцезнаходження відіграє значну роль; близькість до визначних пам'яток, транспортних вузлів або ділових центрів може зробити готель більш привабливим. Наявність додаткових послуг, таких як спа, ресторани, басейни та фітнес-центри, також важлива.

Процес бронювання зазвичай охоплює такі етапи:

– підготовка. Клієнт встановлює свої пріоритети та критерії для вибору готелю, які можуть включати локацію, дати проживання, кількість гостей, бюджет, клас комфорту, наявність додаткових опцій і так далі;

– дослідження та аналіз. Починаючи з пошуку на онлайн-платформах для резервації, клієнти вводять свої параметри і отримують перелік готелів, що відповідають їх вимогам. Вони аналізують варіанти, звертаючи увагу на оцінки та відгуки інших відвідувачів;

– вибір. Після порівняльного аналізу гості обирають готель, який найкраще відповідає їх потребам;

– резервація. На цьому етапі клієнти вносять свої дані для бронювання: імена, контактну інформацію, деталі проживання, тип номеру і т. д., обираючи при цьому додаткові сервіси, які хочуть отримати під час відвідування;

– оплата. Завершуючи реєстрацію, клієнти оплачують проживання за допомогою одного з доступних способів оплати, як-от аванс, повну або часткову оплату при заїзді або негайно через інтернет;

– заїзд та реєстрація. Прибувши в готель, гості проходять процедуру реєстрації, надають всі необхідні дані, отримують ключі від номера та інструкції стосовно свого перебування;

– проживання та виїзд. Використовуючи готельні зручності, гості насолоджуються перебуванням до моменту виїзду, коли вони повертають ключі та можуть залишити відгук про свій досвід.

Розглянувши ці кроки, можна дійти висновку, що система бронювання повинна бути зручною та простою для користувачів, з легкозрозумілим інтерфейсом. Клієнтам має бути легко знаходити потрібні готелі, порівнювати їх, вибирати найкращий варіант та здійснювати бронювання без зусиль.

Важливо забезпечити детальну та актуальну інформацію про кожен готель, включно з фото, описом послуг, оцінками та відгуками, щоб клієнти могли робити обдуманий вибір. Гнучкість у виборі номерів, додаткових послуг та спеціальних пропозицій також є ключовим аспектом. Конфіденційність та безпека особистих даних, надійність процедури бронювання, захист інформації та безпечні методи оплати є критично важливими. Актуальність інформації про наявність номерів, ціни та підтримка клієнтів через різноманітні канали зв'язку також мають велике значення для забезпечення високого рівня задоволення клієнтів.

## 1.2 Аналіз критеріїв ефективності сервісу бронювання готелів

Для аналізу ефективності сервісу бронювання готелів, вибір ключових критеріїв ефективності та формула їх врахування базуються на комплексному

підході, який включає час відгуку системи (T), відсоток успішних бронювань (S%), навантажувальну здатність (N), задоволеність користувача (U), та вартість обслуговування на одного користувача (C), з відповідними вагами критеріїв.

Обрані критерії ефективності мають міцне теоретичне підґрунтя у сфері інформаційних систем та електронної комерції. Дослідження в області управління вебсервісами підкреслюють важливість швидкості відгуку та надійності сервісу як ключових факторів, що впливають на задоволеність та лояльність користувачів. Масштабованість і стабільність системи, визначені через навантажувальну здатність, є вирішальними для забезпечення високої продуктивності та доступності сервісу. Економічна ефективність, виміряна через вартість обслуговування, безпосередньо впливає на загальну ефективність та конкурентоспроможність сервісу.

Між вибраними критеріями існує взаємозв'язок, який важливо враховувати при оцінці ефективності. Наприклад, покращення часу відгуку системи може збільшити задоволеність користувачів та підвищити відсоток успішних бронювань, оскільки користувачі менше чекатимуть на обробку своїх запитів. Водночас, інвестиції в масштабованість та оптимізацію продуктивності можуть знизити вартість обслуговування на одного користувача через ефективніше використання ресурсів.

Методологія збору даних:

1) час відгуку системи (T): вимірювання за допомогою інструментів моніторингу продуктивності, таких як New Relic або Datadog, які дозволяють відстежувати час відповіді сервера на запити користувачів в реальному часі;

2) відсоток успішних бронювань (S%): аналіз логів системи бронювання для визначення кількості успішно завершених транзакцій проти загальної кількості спроб;

3) навантажувальна здатність (N): використання тестування навантаженням з інструментами як Apache JMeter, щоб визначити максимальну кількість одночасних запитів, які система може обробити;

4) задоволеність користувача (U): проведення опитувань серед користувачів, аналіз відгуків і рейтингів на платформах відгуків або соціальних мережах;

5) вартість обслуговування на одного користувача (C): аналіз фінансових звітів для визначення загальних витрат на підтримку та розвиток сервісу, поділених на кількість активних користувачів.

Ваги критеріїв будуть такими:

1) час відгуку системи ($T$) або $W1 = 0{,}1$;
2) відсоток успішних бронювань ($S\%$) або $W2 = 0{,}3$;
3) навантажувальна здатність ($N$) або $W3 = 0{,}3$;
4) задоволеність користувача ($U$) або $W4 = 0{,}2$;
5) вартість обслуговування на одного користувача ($C$): або $W5 = 0{,}1$.

Отже маємо формулу:

$$E = \left(W_1 \times \left(\frac{1}{T}\right)\right) + (W_2 \times S\%) + (W_3 \times N) + (W_4 \times U) - (W_5 \times C).$$

На основі аналізу можуть бути рекомендовані такі стратегії для оптимізації:

– технічні покращення: оптимізація коду, використання кешування, асинхронного програмування для покращення часу відгуку; масштабування інфраструктури для підтримки більшої кількості запитів;

– маркетингові ініціативи: розробка програм лояльності та залучення користувачів через персоналізовані пропозиції для підвищення задоволеності та залученості.

Аналіз показав, що для успішного розвитку та конкурентоспроможності сервісу бронювання готелів необхідно зосередитися на покращенні часу відгуку системи, надійності сервісу, масштабованості, задоволеності

користувачів, та економічної ефективності. Реалізація рекомендованих стратегій оптимізації та покращення дозволить досягти цих цілей і забезпечити високу якість обслуговування

### 1.3 Аналіз аналогічних систем

Для розробки ефективної системи бронювання готелів важливо провести аналіз існуючих аналогічних сервісів. Це дозволить виявити ключові вимоги ринку та потреби користувачів, а також визначити сильні та слабкі сторони конкурентів.

Основні цілі аналізу включають:

1) визначення позиції в ніші. Аналіз конкурентів допомагає зрозуміти, які стратегії використовуються для залучення клієнтів. Важливо оцінити, які унікальні пропозиції (англ. Unique Selling Propositions, Unique Selling Points – USPs) пропонують інші сервіси бронювання готелів та як вони позиціонують себе на ринку;

2) підвищення рівню лояльності клієнтів. Вивчення стратегій конкурентів може виявити нові можливості для інновацій і поліпшень у сервісі. Це може бути пов'язано з розширенням функціоналу, покращенням інтерфейсу користувача або наданням додаткових послуг;

3) визначення сильних і слабких сторін. Знання сильних і слабких сторін конкурентів дозволяє вам адаптувати свій продукт таким чином, щоб він був максимально привабливим для потенційних користувачів. Це також може вказати на важливі аспекти безпеки даних, швидкості обробки запитів, зручності інтерфейсу тощо.

Давайте поглянемо на провідні програмні продукти у галузі готельного бізнесу, аналізуючи популярні онлайн-сервіси для резервування готелів. Один із знакових прикладів – Booking.com, який служить мостом між клієнтами і готелями для бронювання онлайн [4]. Готелі розміщують свої пропозиції на

сайті, а відвідувачі вибирають варіанти, що відповідають їхнім вимогам (рис. 1.2).


Рисунок 1.2 – Домашня сторінка Booking

Переваги сервісу Booking.com:

1) обширний асортимент готелів: Наявність обширної бази даних готелів у різних регіонах та містах, дозволяючи клієнтам підібрати проживання за своїми уподобаннями і фінансовими можливостями;

2) рецензії та оцінки користувачів: Функціонал для відгуків та оцінок допомагає іншим клієнтам у виборі, надаючи інформацію від тих, хто вже мав досвід проживання в конкретних готелях;

Недоліки сервісу Booking.com:

1) недоліки дизайну: Погано реалізовано мобільний дизайн сайту. Одразу ж на домашній сторінці у нас є "горизонтальний скрол", який заважає нормально користуватися сайтом;

2) проблема монетизації: Велика залежність від комісій з продажу може створювати конфлікт інтересів та впливати на об'єктивність рекомендацій.

3) зовнішні та внутрішні нестикування: Іноді інформація на сайті не відповідає дійсності, що може бути пов'язано як з недостатньою перевіркою даних, так і з надмірною рекламою з боку готелів;

4) обмежена взаємодія з готелем: Іноді виникають труднощі у спілкуванні між клієнтами та готелями через посередництво сайту;

5) недоробки інтерфейсу: На сайті іноді бракує додаткових опцій для фільтрації, які б допомогли знайти ідеальне проживання.

Expedia виступає як широко відомий онлайн-ресурс, де можна резервувати житло на відпочинок, включаючи готелі та інші варіанти розміщення [5]. Сервіс надає функціонал для пошуку, порівняння вартості та безпосереднього бронювання (рис. 1.3). Крім того, пропонуються додаткові послуги, такі як підбір авіаперельотів і організація комплексних турів.

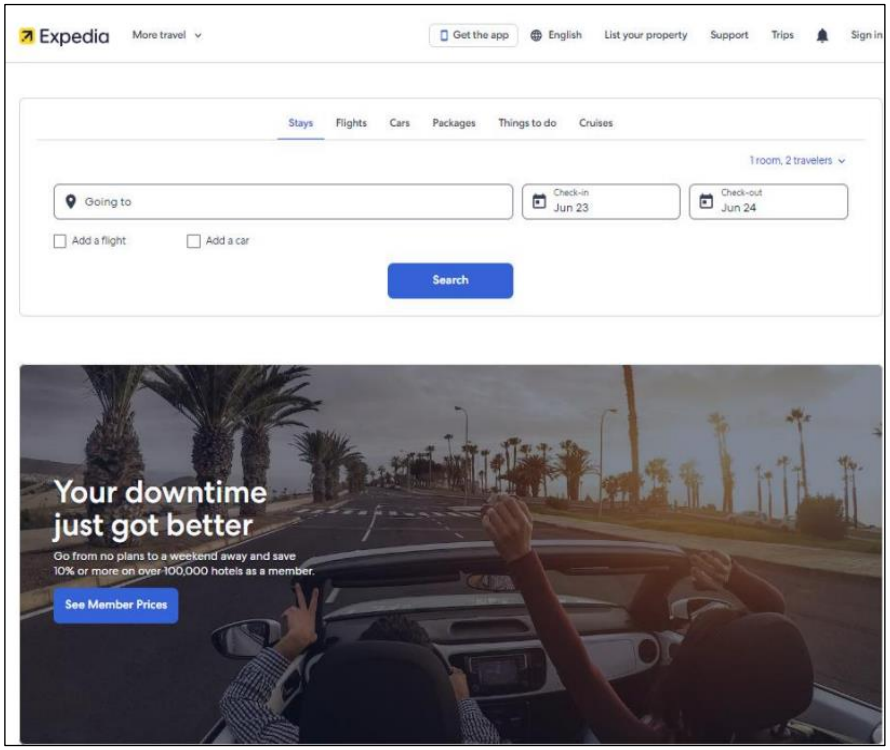


Рисунок 1.3 – Домашня сторінка сайту Expedia

На рис. 1.4 зображено результати пошуку за заданими параметрами (місто, в яке планується подорож, дата в’їзду та виїзду, кількість

дорослих/дітей та необхідна кількість номерів) Expedia. За реєстрацію на сайті пропонується знижка в 15 відсотків.

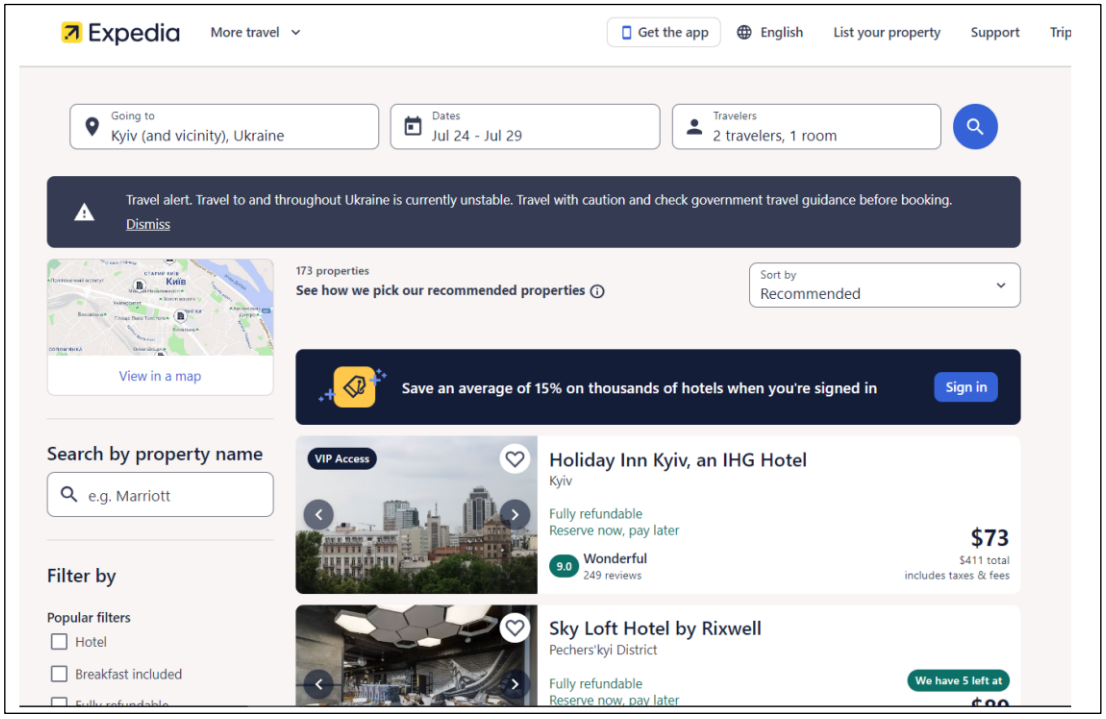


Рисунок 1.4 – Процес пошуку на Expedia

Основні переваги, які пропонує сервіс бронювання Expedia:

1) розмаїтість варіантів проживання: Сайт «Expedia» надає широкий асортимент варіантів для розміщення, дозволяючи вибрати готелі, апартаменти або вілли у різних куточках світу;

2) можливість порівняти ціни: Користувачі можуть аналізувати вартість пропозицій від різних готелів та постачальників, що допомагає знайти найвигідніші умови;

3) комплексні пропозиції: Сайт пропонує комплексні пакети, які включають бронювання готелю разом з авіаквитками або орендою автомобіля, що є зручним і вигідним;

4) відгуки користувачів: Можливість залишати відгуки та оцінки допомагає іншим визначитися з вибором, надаючи об'єктивну інформацію про готелі.

Тим не менш, у Expedia є деякі недоліки:

1) проблеми монетизації: В деяких випадках сервіс може стягувати плату за бронювання, збільшуючи вартість перебування;

2) зовнішні та внутрішні нестиковки: Іноді інформація на сайті не відповідає дійсності, що може бути пов'язано як з недостатньою перевіркою даних, так і з надмірною рекламою з боку готелів;

3) обмежений вибір: Часом сервіс не пропонує доступ до всіх готелів на ринку, оскільки деякі з них використовують альтернативні канали бронювання;

4) недоробки інтерфейсу: Користувачі мають проблеми з пошуком специфічних функцій або інформації;

5) залежність від постачальників: Наявність житла та можливості зміни бронювання можуть обмежуватися угодами з готельними постачальниками.

Agoda – це онлайн-платформа для резервування різноманітних типів помешкань. Цей сервіс пропонує доступ до великої кількості готелів, включаючи малі сімейні заклади та розкішні курорти. Переваги використання Agoda для бронювання готелів:

На рис. 1.5 зображено головну сторінку сайту Agoda.

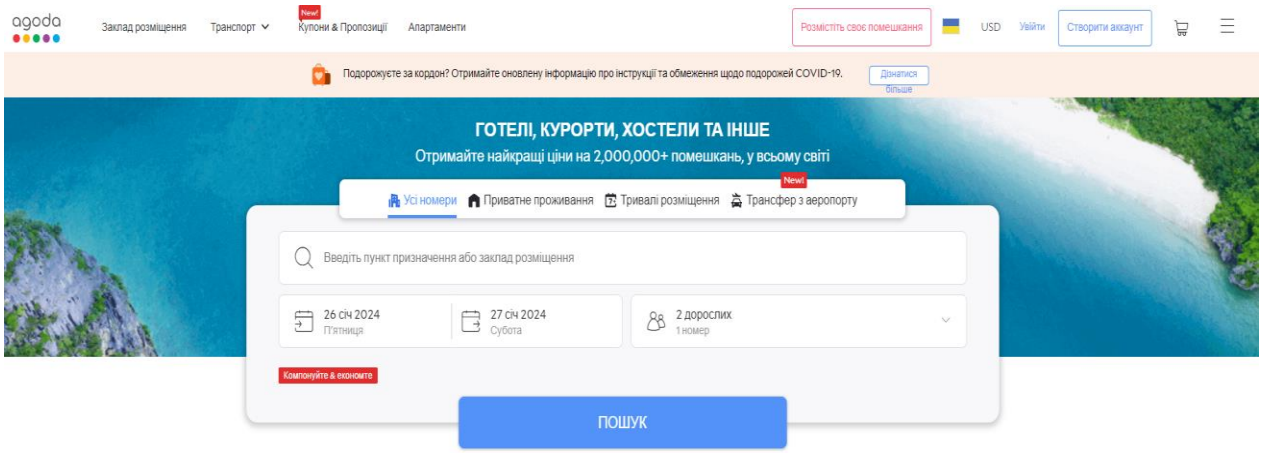


Рисунок 1.5 – Головна сторінка сайту Agoda

На рис. 1.6 зображено результати пошуку за заданими параметрами (місто, в яке планується подорож, дата в'їзду та виїзду, кількість дорослих/дітей та необхідна кількість номерів), також доступний пошук безпосередньо на мапі міста.

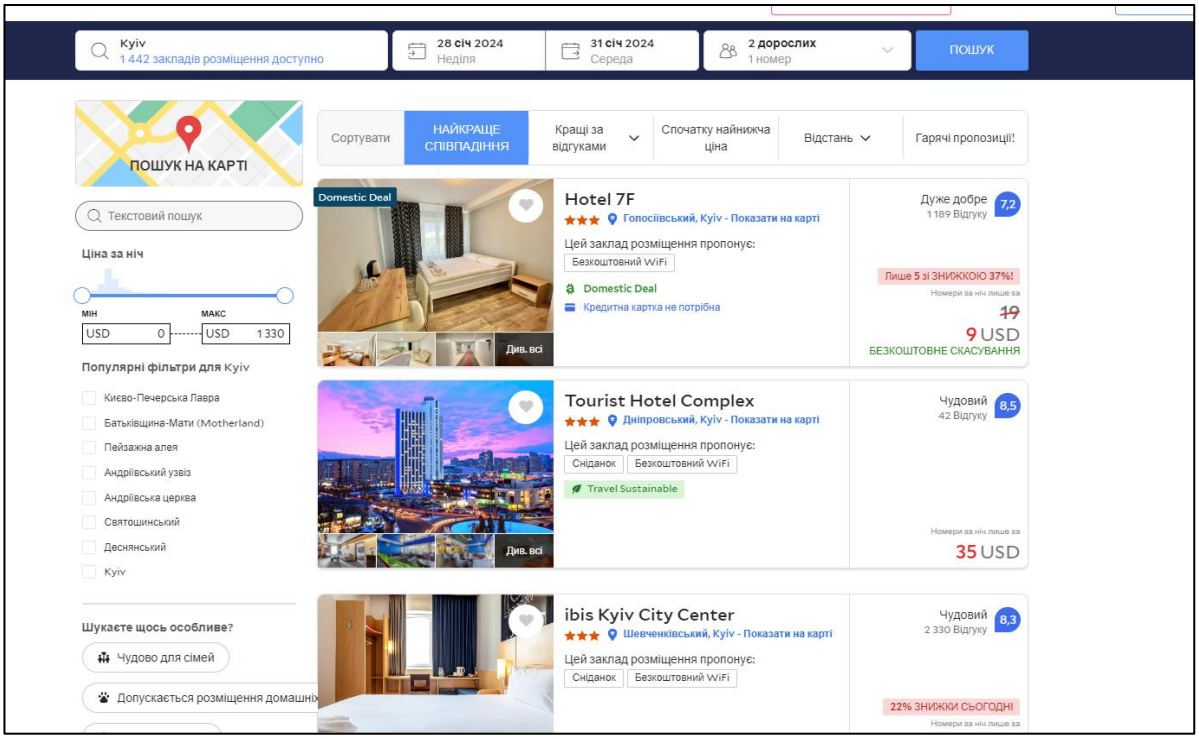


Рисунок 1.6 – Результати пошуку на сайти Agoda

Основні переваги Agoda включають:

1) різноманітність варіантів: Agoda має обширну базу готелів по всьому світу, що надає користувачам можливість вибору з численних варіантів проживання;

2) функціональні фільтри та відгуки: Сервіс надає ефективні інструменти для фільтрації та вибору готелів за індивідуальними вимогами. Також на сайті доступні рейтинги та відгуки від користувачів;

Недоліки сервісу Agoda:

1) різниця у якості готелів: Деякі користувачі відзначають високий рівень обслуговування, в той час як інші стикаються з проблемами пов'язаними з якістю послуг або чистотою;

2) проблема монетизації: Іноді можуть виникати ситуації, коли Agoda показує різні ціни на один і той же готель на різних платформах, що може бути збиваючим;

3) недоробки дизайну: В деяких випадках, інтеграція карт для перегляду розташування готелів може бути недосконалою.

**Шляхи покращення**

1) оптимізація дизайну: Спрощення та очищення інтерфейсу від зайвої інформації;

2) поліпшення функціоналу пошуку та фільтрації: Зробити більш інтуїтивно зрозумілими інструменти для пошуку та фільтрації;

3) вдосконалення синхронізації даних: Покращити механізми обміну даними з партнерами для забезпечення актуальності інформації;

4) зменшення залежності від комісій: Розробка додаткових моделей монетизації, що не залежать від комісій з продажу. Підвищення точності описів: Впровадження строгішої перевірки інформації, яку надають готелі.

У порівнянні з такими сервісами, як Booking.com, Airbnb, та Expedia, власна система бронювання готелю має ряд переваг:

1) контроль над брендом: Власний сайт дозволяє готелю контролювати дизайн та взаємодію з клієнтами, зміцнюючи бренд і впізнаваність;

2) гнучкість ціноутворення: Власний сайт дає можливість гнучко налаштовувати ціни і спеціальні пропозиції;

3) пряма комунікація: Готель може безпосередньо спілкуватися з клієнтами, надавати детальну інформацію та персоналізовані послуги;

4) зниження комісій: Використання власної системи дозволяє уникати високих комісій, які стягують сервіси бронювання;

5) збір даних про клієнтів: Власна система дозволяє збирати цінну інформацію про клієнтів для подальшого аналізу та вдосконалення сервісу.

## 1.4 Специфікація вимог до програмного забезпечення

### 1 Призначення та межі проєкту

#### 1.1 Призначення системи (застосунку), для якої розробляється програмне забезпечення

Система являє собою вебплатформу для бронювання готелів, яка дає змогу користувачам знаходити та бронювати номери в готелях по всьому світу.

Основною метою є надання зручного, швидкого та надійного способу для пошуку та бронювання готельних номерів. Система має забезпечувати можливості для пошуку готелів за різними параметрами (наприклад, місце розташування, ціна, рейтинг, доступні зручності), перегляду детальної інформації про готель, фотографій номерів, відгуків від інших користувачів, а також процес бронювання та оплати проживання.

#### 1.2 Погодження, що ухвалені в програмній документації

Систему розробляють із використанням бібліотеки React для фронтенда і бази даних MongoDB для зберігання даних.

Інтерфейс користувача буде адаптивним і підтримуватиме різні типи пристроїв, включно з мобільними телефонами та планшетами.

#### 1.3 Межі проєкту ПЗ

Проєкт обмежується розробкою вебплатформи для бронювання готелів і не передбачає створення мобільного застосунку. Проєкт не передбачає розроблення власної платіжної системи або складної системи управління цінами та доступністю номерів, передбачається інтеграція з наявними рішеннями на ринку.

### 2 Загальний опис

#### 2.1 Сфера застосування

Платформа орієнтована на приватних осіб, які шукають житло під час подорожей, та бізнес-клієнтів, які планують службові поїздки.

### 2.2 Характеристики користувачів

Система буде пристосована для двох основних типів користувачів: гості, які шукають житло, і адміністратори готелів, які надають інформацію про доступність номерів та ціни.

### 2.3 Загальна структура і склад системи

Система містить кілька основних компонентів:

– клієнтська частина (Frontend): реалізована за допомогою React, забезпечує користувачам графічний інтерфейс для взаємодії з сервісом;

– серверна частина (Backend): забезпечує логіку обробки даних, управління користувацькими сесіями, аутентифікацію та авторизацію; Завдяки Next.js ми можемо безпосередньо звертатися до бази даних без серверної частини, тому що вона буде написана за допомогою Next.js actions, за авторизацію також відповідатиме NextAuth;

– база даних: використовується MongoDB для зберігання даних про користувачів, готелі, бронювання та відгуки.

### 2.4 Загальні обмеження

Сервіс має враховувати обмеження, пов'язані з обробкою персональних даних користувачів у відповідності з GDPR та іншими регулятивними вимогами. Також важливо забезпечити високий рівень безпеки системи для захисту від зовнішніх загроз та вразливостей.

Система повинна бути масштабованою для підтримки зростаючої кількості користувачів та обсягу даних.

## 3 Функції системи

### 3.1 Функція додавання об'єкту розміщення

#### 3.1.1 Опис функції

Дозволяє користувачам створювати нові об'єкти розміщення з детальною інформацією.

#### 3.1.2 Вхідна і вихідна інформація

Вхідні дані: заголовок, опис, зображення, категорія, кількість кімнат,

ванних кімнат, місця для гостей, локація, ціна.

Вихідні дані: деталі створеного об'єкту розміщення.

**3.1.3 Функціональні вимоги**

Перевірка автентифікації користувача.

Збереження об'єкту розміщення в базі даних.

Валідація вхідних даних.

**3.2 Функція видалення об'єкту розміщення**

**3.2.1 Опис функції**

Дозволяє користувачам видаляти власні об'єкти розміщення.

**3.2.2 Вхідна і вихідна інформація**

Вхідні дані: ID об'єкту розміщення.

Вихідні дані: підтвердження видалення об'єкту.

**3.2.3 Функціональні вимоги**

Перевірка автентифікації користувача.

Перевірка права користувача на видалення об'єкту.

Видалення об'єкту з бази даних.

**3.3 Функція отримання детальної інформації про об'єкт розміщення**

**3.3.1 Опис функції**

Надає детальну інформацію про конкретний об'єкт розміщення, включаючи інформацію про власника.

**3.3.2 Вхідна і вихідна інформація**

Вхідні дані: ID об'єкту розміщення.

Вихідні дані: детальна інформація про об'єкт, включаючи інформацію про власника.

**3.3.3 Функціональні вимоги**

Валідація ID об'єкту розміщення.

Повернення детальної інформації про об'єкт та власника.

### 3.4 Функція реєстрації користувача

#### 3.4.1 Опис функції

Дозволяє новим користувачам створювати обліковий запис у системі.

#### 3.4.2 Вхідна і вихідна інформація

Вхідні дані: електронна пошта, ім'я, пароль (або інформація для соціальних мереж).

Вихідні дані: деталі облікового запису користувача.

#### 3.4.3 Функціональні вимоги

Валідація вхідних даних.

Перевірка наявності електронної пошти в системі.

Збереження облікового запису користувача в базі даних.

### 3.5 Функція авторизації користувача

#### 3.5.1 Опис функції

Дозволяє користувачам входити в систему, використовуючи електронну пошту та пароль або через соціальні мережі.

#### 3.5.2 Вхідна і вихідна інформація

Вхідні дані: електронна пошта та пароль або інформація для соціальних мереж;

Вихідні дані: токен авторизації та деталі облікового запису користувача;

#### 3.5.3 Функціональні вимоги

Перевірка відповідності електронної пошти та пароля.

Генерація та повернення токена авторизації.

### 3.6 Функція додавання готелю до «улюблених»

#### 3.6.1 Опис функції

Дозволяє користувачам додавати обрані готелі до списку "улюблених" для швидкого доступу в майбутньому.

#### 3.6.2 Вхідна і вихідна інформація

Вхідні дані: ID готелю;

Вихідні дані: оновлений список «улюблених» готелів користувача;

### 3.6.3 Функціональні вимоги

Перевірка автентифікації користувача;

Валідація ID готелю;

Оновлення списку «улюблених» в базі даних користувача;

## 3.7 Функція видалення готелю із «улюблених»

### 3.7.1 Опис функції

Дозволяє користувачам видаляти готелі зі свого списку «улюблених».

### 3.7.2 Вхідна і вихідна інформація

Вхідні дані: ID готелю.

Вихідні дані: оновлений список «улюблених» готелів користувача без видаленого готелю.

### 3.7.3 Функціональні вимоги

Перевірка автентифікації користувача;

Валідація ID готелю;

Оновлення списку «улюблених» в базі даних користувача;

# 4 Вимоги до програмного забезпечення

## 4.1 Архітектура програмної системи

Програмна система повинна реалізовувати клієнт-серверну архітектуру

## 4.2 Системне програмне забезпечення

Сервіс буде розроблений для роботи в середовищі Node.js, тому для його використання потрібно встановити Node.js та npm (Node Package Manager) як менеджер пакетів.

## 4.3 Мережеве програмне забезпечення

Для забезпечення комунікації між клієнтською та серверною частинами буде використовуватися протокол HTTP/HTTPS через REST API для ефективної взаємодії та обміну даними.

### 4.4 Мережеве програмне забезпечення

В якості СКБД використовується MongoDB, що дозволяє зберігати структуровані та неструктуровані дані. MongoDB Atlas може бути використаний для хмарного розміщення бази даних з високою доступністю та безпекою.

### 4.5 Мова і технологія розробки ПЗ

Фронтенд розробляється на React з використанням Next.js для серверного рендерингу та оптимізації SEO, а основною мовою програмування є TypeScript.

TypeScript забезпечує строгу типізацію, що сприяє підвищенню якості коду та зменшенню кількості помилок під час розробки. Це особливо корисно при роботі з великими та складними проєктами, де потрібна висока надійність коду.

Завдяки використанню Next.js, значна частина логіки бекенду, включаючи безпосередній доступ до бази даних, може бути виконана на стороні клієнта за допомогою Next.js API routes. Це дозволяє створювати серверні функції, які виконуються під час збірки або на сервері на етапі запиту.

Для управління базою даних використовується Prisma як ORM-інструмент, що забезпечує ефективну роботу з MongoDB. Prisma полегшує виконання запитів до бази даних, міграції, а також роботу з моделями даних у TypeScript, забезпечуючи типізацію на рівні схеми бази даних.

Авторизація користувачів реалізована за допомогою NextAuth.js.

## 5 Вимоги до зовнішніх інтерфейсів

### 5.1 Інтерфейс користувача

Інтерфейс користувача має бути інтуїтивно зрозумілим і легким у використанні. Він повинен адаптуватися під різні типи пристроїв (десктопні, планшети, мобільні) та містити такі основні функції, як пошук готелів, перегляд деталей об'єкта бронювання, управління бронюваннями, перегляд та управління обраними об'єктами.

### 5.2 Програмний інтерфейс

Система використовує MongoDB як базу даних для зберігання інформації про користувачів, об'єкти бронювання, бронювання та відгуки. React використовується для розробки інтерфейсу користувача, а Next.js застосовується для серверного рендерингу та управління маршрутизацією.

### 5.3 Програмний інтерфейс

Для комунікації між клієнтом та сервером використовується HTTP/HTTPS протокол. API запити до сервера здійснюються через REST). Аутентифікація користувачів здійснюється через сесії або токени (наприклад, JWT).

## 6 Властивості програмного забезпечення

### 6.1 Доступність

Система має бути доступною 24/7 з використанням cloud-based хостингу, забезпечуючи високу доступність та автоматичне масштабування під час піків навантаження.

### 6.2 Супроводжуваність

Код має слідувати сучасним стандартам розробки

### 6.3 Переносимість

Система має бути сумісна з усіма основними операційними системами та браузерами

## Висновок до розділу 1

У цьому розділі здійснено всебічний аналіз предметної області сервісів бронювання готелів, що включає вивчення технологічних трендів, оцінку поточного стану ринку та аналіз конкурентних продуктів. Особлива увага приділена перевагам використання стеку технологій React та MongoDB у контексті розробки вебсервісів бронювання готелів, зокрема, їх високій продуктивності, гнучкості управління даними, зручності масштабування та спрощення процесу розробки.

Акцент зроблено на аналізі вимог користувачів до сервісів бронювання готелів, включаючи інтуїтивний інтерфейс, швидкість завантаження сторінок, наявність детальної інформації про готелі, зручність вибору та бронювання номерів, а також безпеку транзакцій та конфіденційності даних. Визначено ключові аспекти, які повинен враховувати ефективний вебсервіс бронювання готелів, зокрема, гнучкість пошуку, персоналізацію пропозицій, інтеграцію з соціальними мережами та системами відгуків.

Далі проведено аналіз переваг та недоліків існуючих сервісів бронювання готелів, визначивши основні напрямки для покращення. На основі цього аналізу були сформульовані технічні та функціональні вимоги до нового сервісу, який базується на технологіях React та MongoDB. Зокрема, було підкреслено важливість створення адаптивного дизайну, що забезпечує однаково високу якість користувацького досвіду на різних пристроях, від стаціонарних комп'ютерів до мобільних телефонів.

У результаті цього дослідження, було виявлено, що використання комбінації React та MongoDB дозволяє створити високопродуктивний, масштабований та легко адаптований вебсервіс бронювання готелів, який відповідає сучасним вимогам ринку та забезпечує високий рівень задоволення користувачів.

# 2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ РОБОТИ

## 2.1 Вибір цільової платформи системи

У наш час використання інтернет-технологій та смартфонів стало звичайною практикою, сприяючи росту запиту на різноманітні онлайн-сервіси. Зокрема, це стосується онлайн-резервування готелів, яке вже стало ключовим елементом сучасних подорожей. Ці інтернет-платформи перетворились на важливий інструмент для планування поїздок, дозволяючи користувачам легко обирати та бронювати номери у готелях, незалежно від їхнього місцезнаходження, що зробило процес більш зручним та доступним.

Багато готелів вибирають зовнішні платформи для демонстрації своїх послуг, але це створює залежність від цих систем, які контролюють доступ клієнтів до готелів і вимагають певний відсоток за бронювання. Ця тенденція посилила конкуренцію у готельній індустрії, особливо в онлайн-сегменті, змушуючи готелі знижувати ціни, щоб залучити більше клієнтів, що в кінцевому результаті знижує їхню маржу [2].

Крім того, залежність від цих платформ може обмежувати пряме спілкування готелів з клієнтами, ускладнюючи персоналізацію послуг та розуміння потреб клієнтів, що впливає на їх загальне задоволення та вірність бренду. Високі комісії, стягувані платформами, також негативно впливають на фінансову стабільність готелів.

Відмова від онлайн-бронювання може призвести до втрати конкурентних переваг і обмеження ринкового охоплення. Більшість сучасних клієнтів використовують інтернет для пошуку та бронювання готелів, тому відсутність онлайн-бронювання може обмежити можливості реклами, просування та зменшити дохід готелів.

При створенні сервісу бронювання готелів на основі React та MongoDB першочерговим завданням є визначення цільової платформи та аудиторії. Метою системи є надання зручного інструменту для користувачів при виборі та

бронюванні готельних номерів. Основною проблемою, яку вирішує сервіс – це спрощення та прискорення процесу пошуку та бронювання готелів.

Користувацький досвід (UX) відіграє ключову роль в успіху сервісу бронювання. Необхідно створити інтуїтивно зрозумілий та ефективний інтерфейс, який би включав зручну систему фільтрації, пошуку, вибору номерів, а також процес бронювання і оплати.

**Особливості вибору платформи:**

– **вебзастосунки.** Вебзастосунок забезпечує легкий доступ користувачів з будь-якого пристрою, що має браузер, не вимагаючи завантаження додаткового програмного забезпечення [3];

– **адаптивність.** Важливою частиною розробки є створення адаптивного дизайну, який оптимізує відображення та взаємодію на різних пристроях (десктоп, планшет, мобільний телефон);

– **інтеграція з MongoDB.** MongoDB є гнучкою нереляційною базою даних, що дозволяє легко зберігати та обробляти великі обсяги даних, що є важливим для сервісів бронювання готелів. Ця технологія сприяє ефективному управлінню даними про готелі, відгуки користувачів, історію бронювань та іншу важливу інформацію;

– **оптимізація швидкості роботи.** Швидкість завантаження та відгуку сервісу є критичною для забезпечення хорошого користувацького досвіду. Використання React дозволяє оптимізувати взаємодію з користувачем, роблячи інтерфейс більш чуйним;

– **масштабованість.** Система має бути розроблена з урахуванням потенційного зростання та розширення функціоналу. MongoDB і архітектура React сприяють легкому масштабуванню сервісу.

Враховуючи вищевказані аспекти, можна зробити висновок, що така система повинна забезпечувати не тільки ефективний користувацький досвід, але й гнучкість у розробці та управлінні даними, що є важливим для динамічно змінюваних умов ринку готельних послуг.

## 2.2 Архітектурна модель

У веброзробці цього застосунку було вибрано модель взаємодії клієнт-сервер, яка дуже популярна для створення вебзастосунків. Згідно з цією моделлю, клієнтська та серверна частини працюють разом через мережу, обмінюючись даними та обробляючи запити й відповіді.

Інтерфейс користувача, який є клієнтською частиною, може бути реалізований у формі браузера, мобільного застосунку чи іншого програмного забезпечення для доступу до ресурсів сервера. Ця частина відповідає за візуалізацію інформації, збір та обробку даних від користувача та надсилання запитів до сервера.

З іншого боку, сервер займається обробкою цих запитів, виконанням бізнес-операцій, доступом до баз даних та надсиланням відповідей назад клієнту. Сервер може бути як фізичним, так і віртуальним комп'ютером, на якому працює серверний застосунок або вебсервер.

Комунікація між клієнтом та сервером відбувається за допомогою протоколу HTTP. Клієнт формулює запити з необхідними даними, такими як URL, параметри запиту, заголовки, а сервер їх обробляє, виконує потрібні дії та відсилає відповідь (рис. 2.1). Використання клієнт-серверної моделі дає можливість ефективно розподілити завдання між двома частинами, забезпечуючи таким чином гнучкість, масштабованість та високу швидкість роботи застосунку, а також полегшує розробку, підтримку безпеки і контроль доступу.

Цей підхід широко використовується у вебзастосунках, оскільки дозволяє різноманітним пристроям доступатися до серверних ресурсів, а сервери забезпечують централізований доступ до ресурсів та бізнес-логіки. Така модель дозволяє створювати потужні застосунки, які можуть обслуговувати велику кількість користувачів одночасно.

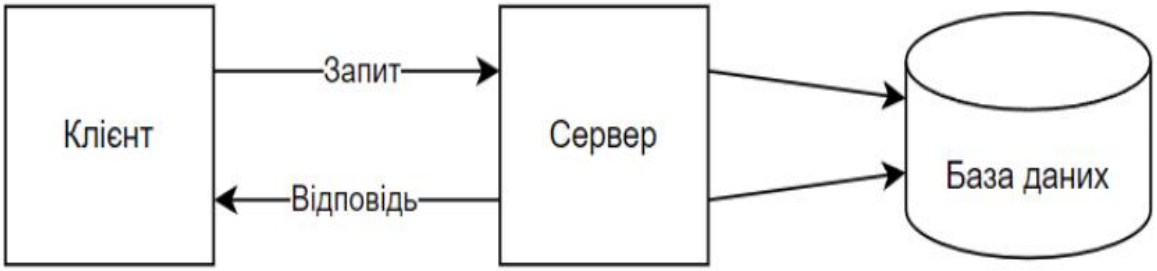


Рисунок 2.1 – Схема клієнт-серверної архітектури

Основні переваги архітектури клієнт-сервер включають:

1) відділення ролей клієнта та сервера: інтерфейс та користувацькі взаємодії обробляються на клієнтській стороні, в той час як сервер займається обробкою інформації, бізнес-логікою та зберіганням даних.

2) масштабованість системи: залежно від потреб, сервер можна апгрейдити, додавши ресурси (вертикальне масштабування) або встановивши додаткові сервери (горизонтальне масштабування), що забезпечує ефективність при високих навантаженнях;

3) високий рівень безпеки: клієнти не мають безпосереднього доступу до сервера, що знижує ризик несанкціонованого доступу до даних. Використання методів аутентифікації та авторизації додатково захищає конфіденційність інформації;

4) використання різноманітних технологій: на клієнтській та серверній сторонах можна використовувати різні платформи та технології, що дає змогу створювати застосунки, сумісні з різними пристроями та ОС;

5) співпраця у командах: розробка клієнтської та серверної частин може вестися незалежно, навіть різними командами або компаніями, сприяючи розподілу робочих процесів, прискоренню розробки та забезпеченню якості ПЗ.

## 2.3 Моделювання предметної області: опис об’єкту дослідження, учасники процесу бронювання

До розробки архітектури сервісу бронювання готелів доцільно провести моделювання предметної області, використовуючи технології React для фронтенду та MongoDB для бекенду.

Об’єктом дослідження у даному випадку є процес бронювання номерів в готелях через онлайн платформу. Цей процес включає в себе взаємодію між кінцевими користувачами (клієнтами), системою бронювання та самими готелями. Основною метою є забезпечення зручного та ефективного механізму для вибору та бронювання номерів, що відповідають специфічним потребам та вимогам користувачів.

**Процес бронювання залучає кілька ключових учасників:**

1) клієнти – особи або організації, які шукають та бронюють готельні номери через онлайн платформу;

2) готелі – постачальники послуг проживання, які надають інформацію про доступні номери, ціни, умови проживання тощо;

3) платіжні системи – забезпечують обробку фінансових транзакцій між клієнтами та сервісом бронювання/готелями;

4) адміністратори системи – особи, які відповідають за управління та підтримку платформи бронювання.

Моделювання предметної області вимагає створення функціональних та інформаційних моделей, які дозволять детально описати процес бронювання та взаємодії між учасниками. Система бронювання готелів організована навколо основних моделей: «User», «Account», «Listing», та «Reservation», які взаємопов'язані та відображають ключові аспекти бізнес-процесу:

– модель «User» представляє користувачів системи, з полями для зберігання основної інформації, такої як ім'я, електронна пошта, зображення профілю, хешований пароль, та історію створення та оновлення профілю;

– модель «Account» забезпечує інтеграцію зі сторонніми сервісами автентифікації, включаючи поля для зберігання токенів доступу, типу акаунту, та іншої відповідної інформації;

– модель «Listing» відображає інформацію про об'єкти проживання, які можуть бути забронювані через систему. Ця модель містить деталі, такі як назва, опис, кількість кімнат, ванних кімнат, максимальну кількість гостей, розташування, вартість за ніч, та інші;

– модель «Reservation» управляє бронюваннями, зв'язуючи користувачів із об'єктами проживання. Вона зберігає інформацію про дати початку та закінчення проживання, загальну вартість, та має зв'язок з відповідними користувачем та об'єктом проживання.

Використання детально визначених моделей у системі бронювання готелів дозволяє не тільки чітко структурувати зберігання та обробку даних, але й гнучко налаштовувати бізнес-логіку відповідно до вимог користувачів та умов ринку.

**Взаємодія між моделями:**

− кожен користувач може мати кілька акаунтів для різних методів входу та ідентифікації, а також може опублікувати кілька об'єктів проживання (listings) і мати кілька бронювань (reservations);

− об'єкти проживання (listings) прив'язані до конкретних користувачів, які їх опублікували, та можуть мати багато бронювань від різних користувачів;

− бронювання (reservations) забезпечує зв'язок між користувачем, який бронює, та конкретним об'єктом проживання, вказуючи період та загальну вартість проживання.

Для наочного представлення взаємодій між учасниками процесу бронювання та компонентами системи було розроблено ER-діаграму за допомогою сервісу «Mermaid» (рис. 2.2).

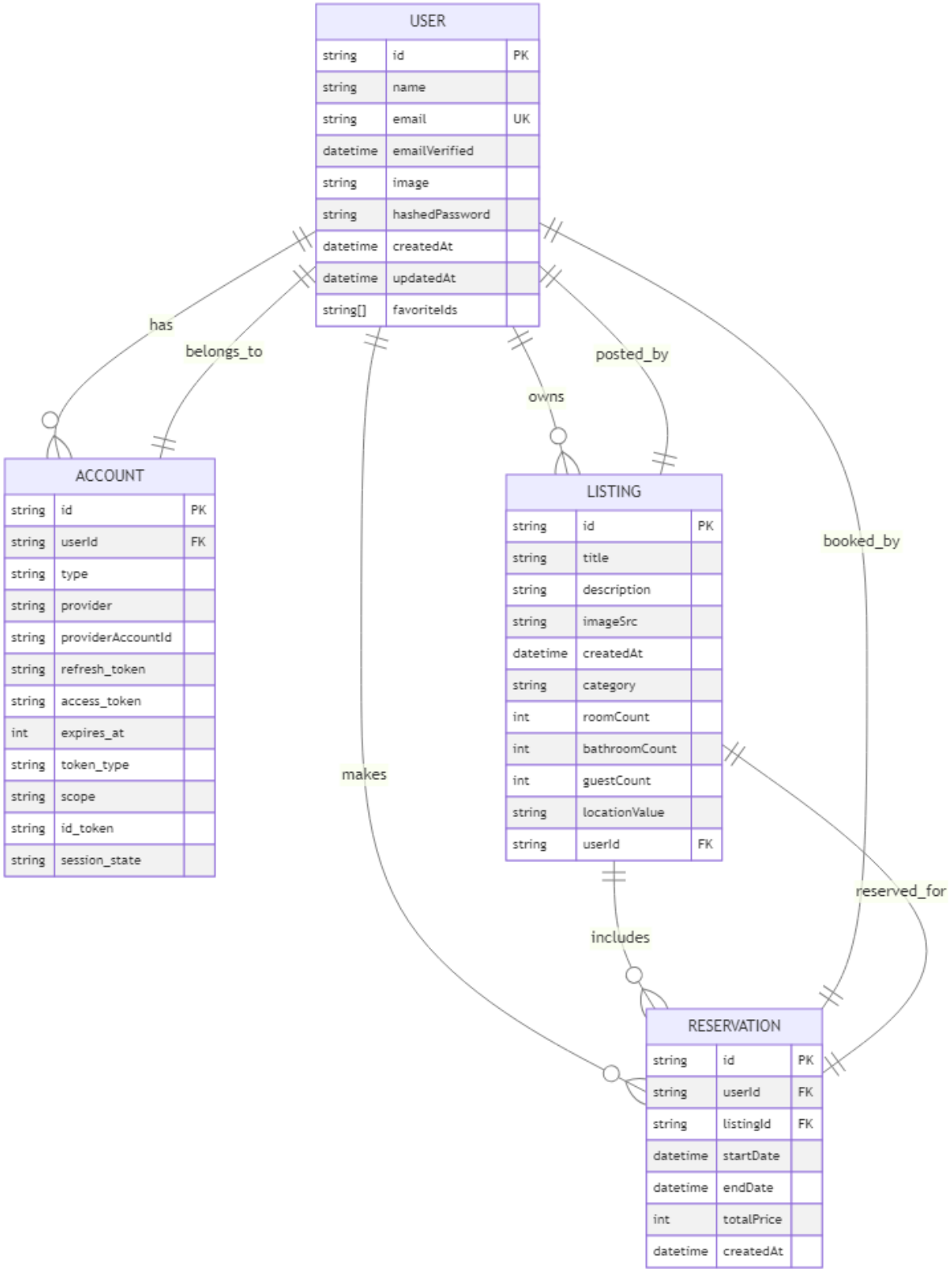


Рисунок 2.2 – ER-діаграма системи

## 2.4 Процеси та алгоритми роботи сервісу: опис основних процесів бізнес-логіки

На основі зібраних даних можна дійти висновку, що ефективна робота готельного бізнесу в сучасних умовах неможлива без впровадження онлайн-системи резервування. Ціль розробки програмної підсистеми резервування полягає в оптимізації процесу замовлення номерів з боку клієнтів та мінімізації витрат на її обслуговування з боку власників готелів. В процесі резервування можна виділити наступні ключові етапи, які повинні бути інтегровані в онлайн-систему:

1) відображення списку номерів, доступних для бронювання на поточний момент;

2) функціонал пошуку та фільтрації для відбору номера за певними параметрами або вимогами;

3) надання повної інформації про номер, включаючи фотографії, ціни та опис;

4) можливість вибору дат заїзду та виїзду;

5) процес реєстрації, під час якого клієнт заповнює персональні дані: ім'я, контактні дані, спосіб оплати, інформацію про платіжну картку та інші необхідні деталі для бронювання;

6) отримання підтвердження про резервування від менеджера готелю після заповнення всіх даних;

7) здійснення оплати через сайт у випадку, якщо клієнт вибрав такий спосіб оплати;

8) видача клієнту всієї необхідної інформації про бронювання, включаючи дати проживання, тип номера, запропоновані послуги та інші важливі деталі.

Також у процесі було розроблено концептуальну блок-схему роботи вебсервісу. Блок-схема головного циклу показана на рис. 2.2.

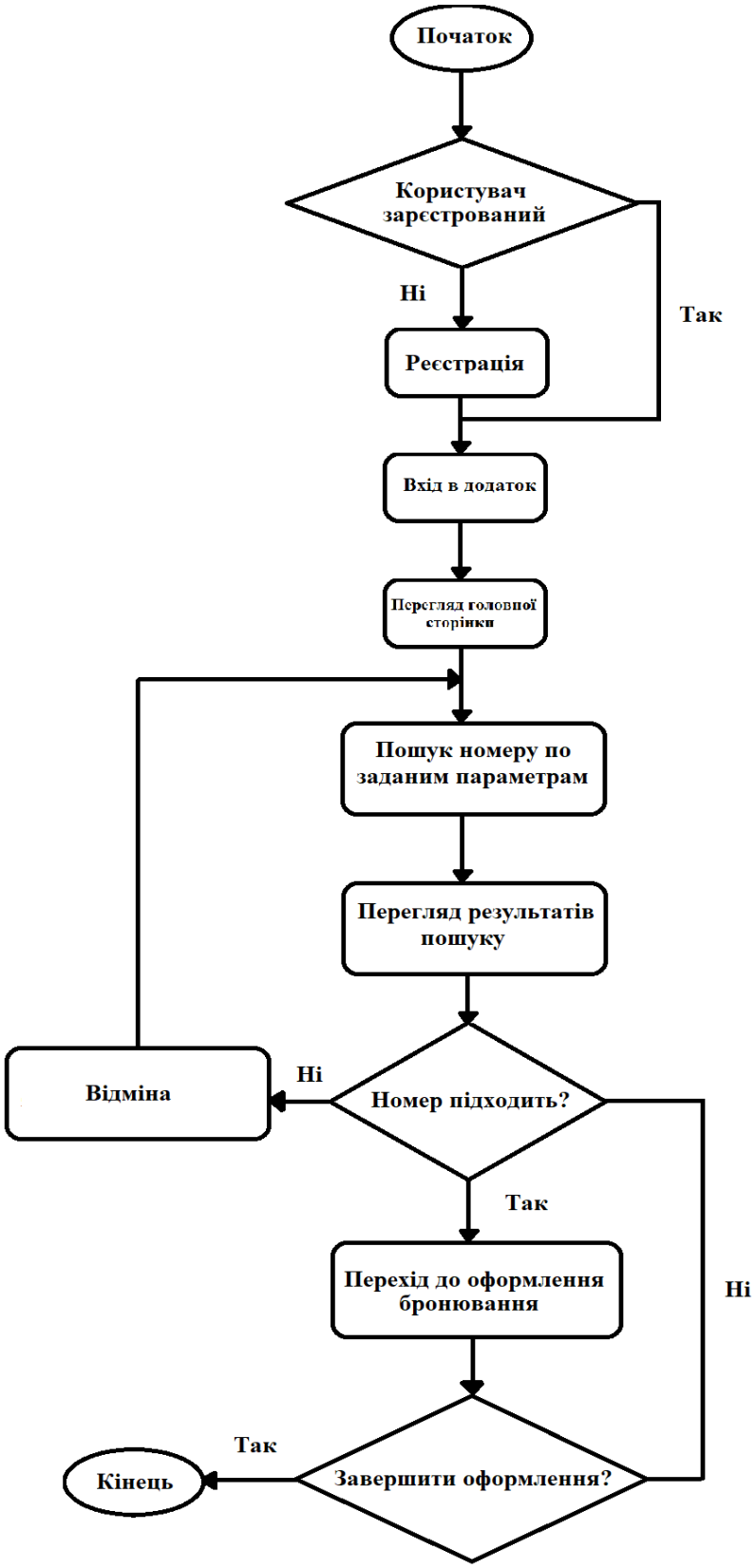


Рисунок 2.2 – Блок-схема алгоритму основного циклу

Для аналізу створеного програмного продукту були розроблені діаграми BPMN для окремих бізнес-процесів, втілених у програмному забезпеченні.

Бізнес-процес представляє собою серію взаємопов'язаних дій, спрямованих на досягнення певної мети або результату в межах компанії. Цей процес охоплює різноманітні етапи, дії та задачі, що забезпечують взаємодію між різними учасниками процесу в організації.

Використання нотації BPMN для моделювання бізнес-процесів дозволяє візуалізувати та формалізувати ці процеси для кращого розуміння і документування. Головною ціллю застосування BPMN є створення чіткого та уніфікованого зображення бізнес-процесів, що полегшує обмін інформацією між зацікавленими сторонами, такими як аналітики, програмісти, керівники проєктів та інші учасники.

За допомогою моделі BPMN можна детально представити послідовність дій та задач, які відбуваються протягом бізнес-процесу. Це сприяє кращому розумінню процесу, ідентифікації потенційних проблемних зон або шляхів для поліпшення. Нотація BPMN пропонує стандартизований підхід до представлення елементів процесу та їх взаємодій, що забезпечує однакове розуміння процесів серед усіх учасників. Це може слугувати основою для автоматизації бізнес-процесів через використання спеціалізованого програмного забезпечення для управління бізнес-процесами (BPM), забезпечуючи більш ефективне виконання задач, автоматизацію передачі завдань між співробітниками та моніторинг процесів. Такий підхід не лише поліпшує ефективність роботи, але й надає інструменти для аналізу, оптимізації та удосконалення бізнес-процесів, виявляючи непотрібні етапи або затримки, що дозволяє вносити корективи для підвищення продуктивності організації.

**Бізнес-процес «Реєстрації користувача в системі»**

Процес «Реєстрація користувача» починається на стартовій сторінці:

1) користувач обирає «Реєстрація», після чого відбувається переадресація на сторінку з формою для створення акаунта;

2) тут користувач вводить свої особисті дані у форму: прізвище, ім'я, по батькові, нікнейм, email, дату народження і пароль;

3) у процесі заповнення форми система перевіряє коректність даних і обов'язковість заповнення всіх полів. Помилки підсвічуються, а кнопка «Реєстрація» стає активною тільки після виправлення всіх недоліків і повного заповнення форми;

4) у разі успішного заповнення даних користувач може натиснути на «Реєстрація»;

5) натискання кнопки ініціює запит до сервера, і за відсутності помилок, користувач переходить у свій особистий кабінет.

Система також перевіряє унікальність логіна та електронної пошти. У разі виявлення вже наявного акаунта з такими даними, виводиться повідомлення про помилку.

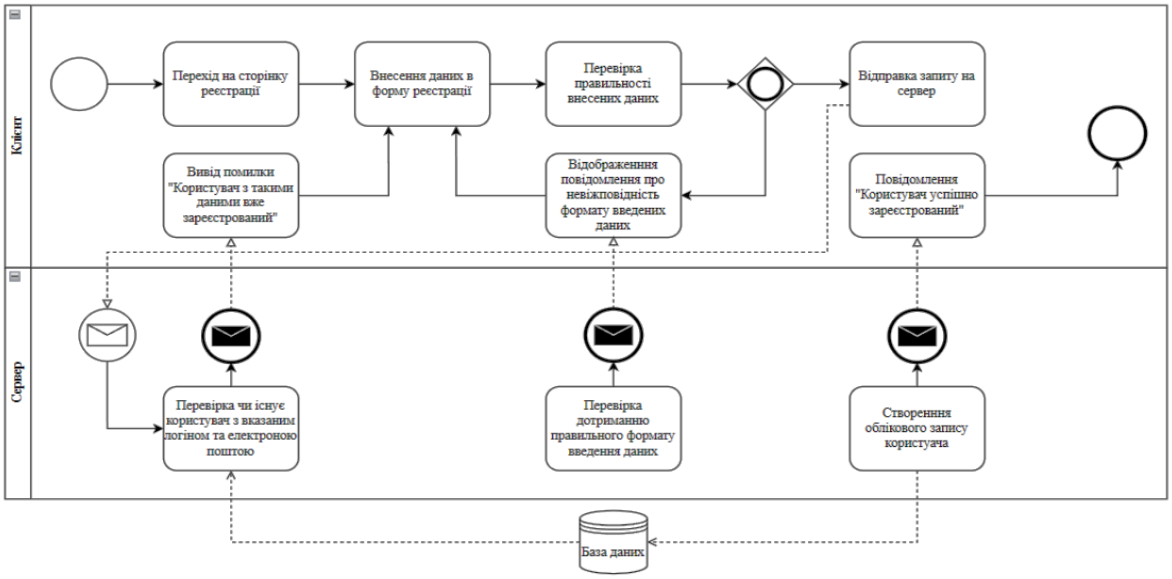


Рисунок 2.3 – Бізнес-процес «Реєстрації користувача в системі»

**Бізнес-процес «Отримання результатів пошуку номерів за заданими параметрами» (рис. 2.4):**

1) залогінений користувач вказує потрібні деталі для пошуку (назву місця проживання або регіону майбутньої поїздки, дати прибуття та від'їзду, кількість осіб та дітей) у спеціальній формі на сайті;

2) ініціюється запит до сервера з метою знаходження інформації про кімнати, що можуть бути заброньовані;

3) сервер відповідає переліком номерів, які відповідають заданим критеріям і наразі є вільними;

4) клієнт оглядає пропоновані варіанти з цього списку;

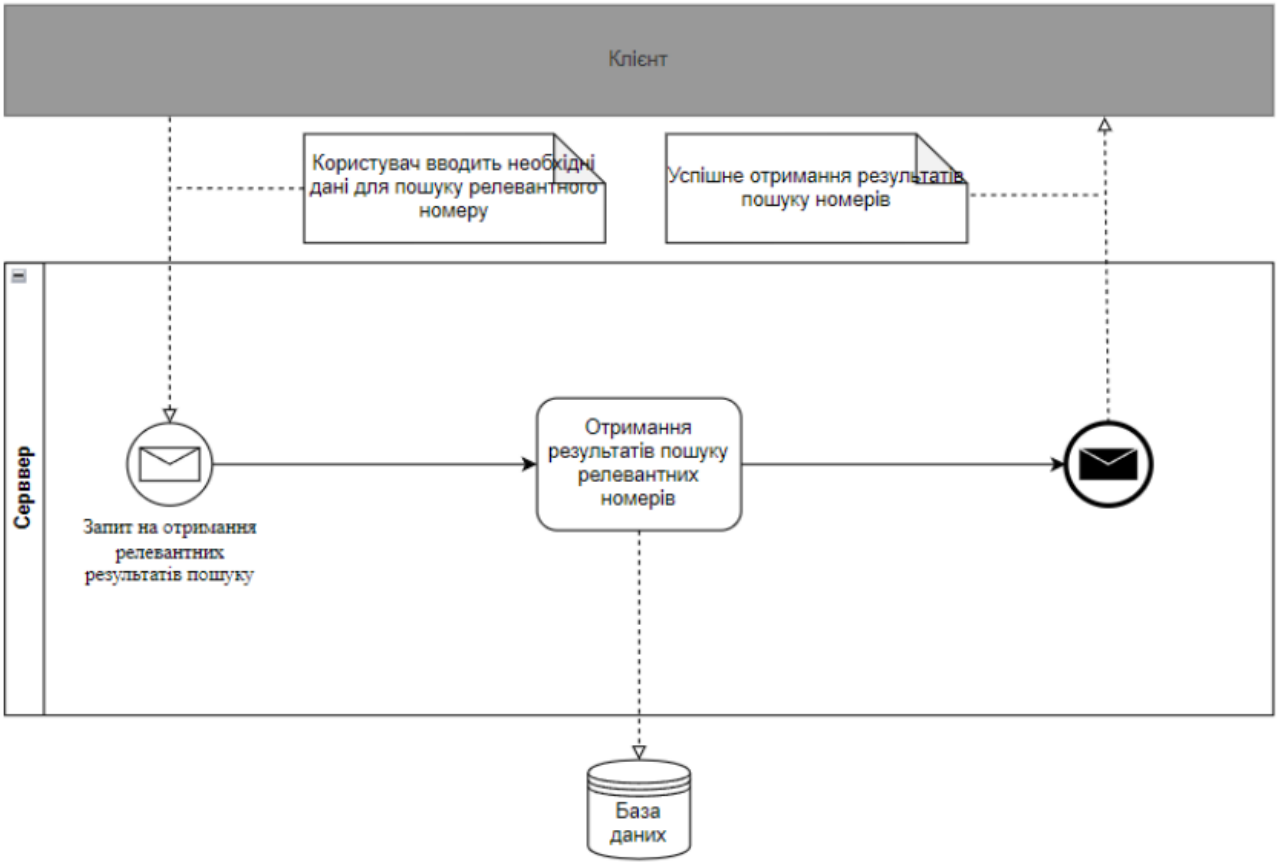


Рисунок 2.4 – Бізнес-процес «Отримання результатів пошуку номерів за заданими параметрами»

## Висновок до розділу 2

У даному розділі підсумовуються ключові аспекти проєктування та реалізації онлайн-платформи. Значний акцент робиться на інноваційному підході до взаємодії між клієнтами та готелями, забезпеченні зручного користувацького інтерфейсу та ефективного управління даними за допомогою обраного стека технологій.

Архітектурна модель, обрана для розробки застосунку, заснована на клієнт-серверній взаємодії, що є стандартом для сучасних вебзастосунків. Це забезпечує гнучкість, масштабованість та високий рівень безпеки, розділяючи ролі між клієнтською (React) та серверною (MongoDB) частинами. Такий підхід дозволяє не тільки ефективно обробляти запити користувачів, але й легко адаптуватися до змінних потреб ринку.

Моделювання предметної області визначає ключових учасників процесу бронювання та їхні взаємовідносини, що дозволяє структуровано підходити до розробки функціоналу сервісу. Застосування технологій React та MongoDB сприяє створенню зручного користувацького інтерфейсу та ефективному управлінню великими обсягами даних, відповідно.

Опис процесів та алгоритмів роботи сервісу демонструє глибоке розуміння бізнес-логіки онлайн-бронювання та забезпечує чіткі кроки для користувачів від пошуку до фіналізації бронювання. Використання BPMN діаграм допомагає візуалізувати та оптимізувати ці процеси, забезпечуючи ефективне взаємодію між усіма учасниками системи.

Важливою перевагою розробленого рішення є його адаптивність та масштабованість, що дозволяє сервісу ефективно розширюватися відповідно до зростання числа користувачів та обсягу доступних для бронювання об'єктів. Особлива увага приділена оптимізації швидкості роботи та користувацького досвіду, що є критичним для забезпечення високої конверсії та задоволення користувачів.

# 3 АРХІТЕКТУРА, МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Архітектура ПЗ

Основна мова програмування, що використовується в проєкті, це TypeScript. TypeScript, розроблений і підтримуваний Microsoft, є строго типізованою надмножиною JavaScript, що пропонує розширені можливості для розробки масштабованих і стійких вебзастосунків. Вибір TypeScript для розробки вебсервісу бронювання готелів зумовлений кількома ключовими факторами, які роблять його кращим порівняно з традиційним JavaScript.

Основна відмінність і головна перевага над Javascript це те, що TypeScript надає сувору типізацію, що дозволяє розробникам визначати типи змінних, функцій, об'єктів та інших структур даних. Це забезпечує вищий рівень безпеки під час виконання, оскільки більшість помилок, пов'язаних з типами, виявляється на етапі компіляції. На відміну від JavaScript, де помилки типів часто виявляються тільки під час виконання, використання TypeScript сприяє ідентифікації та усуненню таких помилок заздалегідь, покращуючи якість і надійність коду.

Також, у поєднанні з сучасними фреймворками і бібліотеками, Typescript забезпечує більш організовану структуру коду, що сприяє кращому управлінню станом і архітектурою застосунків. Це особливо важливо для розроблення складних вебзастосунків, як сервіси бронювання готелів, де потрібне опрацювання великої кількості даних і взаємодія з різними компонентами інтерфейсу.

**Фронтенд**

Фронтенд застосунку будується з використанням React і Next.js, де Next.js надає зручні абстракції для серверного рендерингу (SSR) та інкрементально-статичної генерації сторінок (ISG), а також для побудови API маршрутів:

– сторінки та компоненти: файли в директорії pages автоматично стають маршрутами в застосунку. Компоненти з папки app/components використовуються для побудови користувацького інтерфейсу, організовуючи повторно використовувані елементи UI;

– Next.js API Routes: У папці app/api знаходяться маршрути API, створені за допомогою функціональності API Routes від Next.js. Ці маршрути слугують бекенд-частиною застосунку, обробляючи HTTP-запити (наприклад, POST для створення бронювання) і взаємодіючи з базою даних через Prisma.

**Бекенд**

Prisma: Prisma використовується як ORM для роботи з MongoDB. Вона дає змогу визначати схему бази даних і автоматично генерувати клієнт для виконання запитів до бази даних із коду програми. Prisma забезпечує типізацію і безпеку запитів до бази даних.

API Routes: Бекенд-логіка вбудована безпосередньо в Next.js через API Routes, ці маршрути обробляють аутентифікацію, CRUD-операції, взаємодію із зовнішніми API і логіку бізнес-процесів. Файли в app/api являють собою серверні функції, які приймають HTTP-запити і повертають відповіді, взаємодіючи з базою даних через Prisma.

**Взаємодія**

Запити з фронтенду: коли користувач взаємодіє з інтерфейсом (наприклад, надсилає форму бронювання), фронтенд відправляє запит до відповідного API маршруту, використовуючи fetch API або інші HTTP-клієнти.

Обробка на бекенді: API маршрут на бекенді приймає запит, обробляє його (наприклад, створює нове бронювання в базі даних), і відправляє відповідь назад на фронтенд.

Відповідь фронтенду: фронтенд обробляє відповідь від сервера (наприклад, показує повідомлення про успішне створення бронювання або помилку).

Архітектура добре структурована і розділена на логічні частини, що спрощує розробку і підтримку. Використання Prisma для взаємодії з MongoDB, Zustand для управління станом і NextAuth для автентифікації дає змогу створити безпечний, масштабований вебзастосунок, який легко підтримувати.

## 3.2 Вибір технологій для розробки клієнтської частини

### 3.2.1 Порівняння React та інших фреймворків для створення SPA-інтерфейсів користувача

Розвиток інформаційних технологій спонукає до постійного вдосконалення інструментів веброзробки. Вибір технологій для створення вебзастосунків має вирішальне значення для успіху проєкту. Серед сучасних фреймворків і бібліотек для фронтенд-розробки особливе місце займає React, розроблений компанією Facebook.

Для порівняння фреймворків React, Vue і Angular ми можемо вибрати такі критерії, ґрунтуючись на об'єктивних даних і характеристиках, які можуть бути виміряні та порівнянні: продуктивність, гнучкість, спільноту, а також простоту навчання і використання. Важливо підкреслити, що вибір фреймворка або бібліотеки багато в чому залежить від конкретних вимог проєкту, уподобань розробника і контексту завдання (табл. 3.1).

Табл. 3.1 підкреслює ключові відмінності між React, Vue і Angular, даючи змогу зробити поінформований вибір залежно від вимог проєкту. React вирізняється своєю продуктивністю, гнучкістю і підтримкою великої

спільноти, що робить його привабливим вибором для розроблення сучасних вебзастосунків.

Таблиця 3.1 – Порівняння основних фреймворків для створення інтерфейсів користувача

| Критерій | React | Vue | Angular |
|---|---|---|---|
| **Продуктивність** | Висока завдяки віртуальному DOM | Висока завдяки віртуальному DOM і оптимізаціям | Висока завдяки AOT компіляції та change detection |
| **Гнучкість** | Висока, мінімалістичний підхід з можливістю вибору додаткових бібліотек | Середня, надає детально налаштований набір функцій з можливістю розширення | Низька, суворий і комплексний фреймворк з безліччю вбудованих можливостей |
| **Спільнота** | Дуже велика й активна | Велика і зростаюча | Велика, але з тенденцією до зменшення популярності |
| **Простота навчання** | Середня | Висока, легко почати роботу завдяки простому API | Низька, складний для новачків через обширність і складність |
| **Ідеально підходить для** | Великі проєкти, що вимагають гнучкості та масштабованості | Проєкти середнього розміру, що вимагають швидкого прототипування та розроблення | Великі, складні застосунки з необхідністю суворої архітектури та інтеграції на стороні клієнта |

Популярність React також можна спостерігати в статистиці використовуваних фреймворків за останній рік (рис. 3.1).

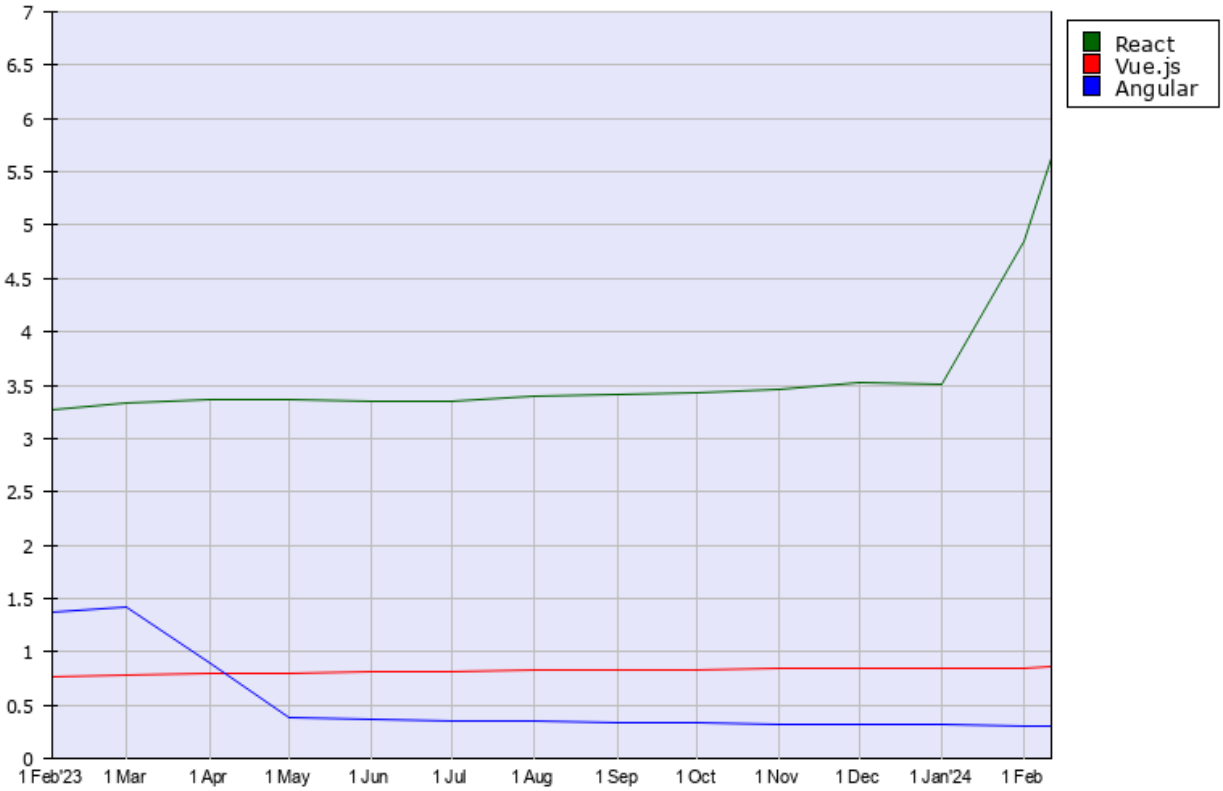


Рисунок 3.1 – Використання React/Vue/Angular для вебсайтів за 2023 рік

Ці фактори роблять React привабливим вибором для розроблення сучасних вебзастосунків, особливо в умовах, коли потрібна масштабованість, висока продуктивність і гнучкість у виборі технологій.

### 3.2.2 Вибір фреймворку для React

Розробники, які працюють із React, мають кілька варіантів для створення повноцінних вебзастосунків, але серед них виділяються три основні фреймворки: «Astro», «Remix» та «Next.js». Кожен з них пропонує унікальний набір функціональності, оптимізацію продуктивності та зручність розробки, але для вибору найбільш підходящого інструменту необхідно глибоко зрозуміти їхні ключові особливості та відмінності (табл. 3.2).

Таблиця 3.2 – Порівняння основних фреймворків React

| **Критерій** | **Astro** | **Remix** | **Next.js** |
|---|---|---|---|
| Гнучкість | React, статичний сайт | Edge native, фулл-стек | React, серверний та статичній |
| Рендеринг | SSG, SSR через адаптери | SSR | SSR, SSG, ISR |
| Особливості | Партійна гідратація, мінімалізм JavaScript | Підтримка форм, маршрутизація | Маршрутизація React Server Components |
| Продуктивність | Висока за рахунок мінімізації JavaScript | Оптимізоване завантаження даних | Висока, оптимізована за допомогою Turbopack |
| Підтримка стилів | Підтримка більшості CSS бібліотек | Вбудована підтримка стилів | Вбудована підтримка CSS і SASS |

«Next.js» фреймворк пропонує найкраще співвідношення гнучкості, продуктивності та зручності розробки. «Next.js» забезпечує широкі можливості для створення як статичних, так і серверно-рендерних вебзастосунків, підтримуючи при цьому сучасні практики веброзробки, такі як асинхронні компоненти та оптимізацію завантаження. Особливо важливим є його здатність до оптимізації продуктивності за допомогою таких інструментів, як Turbopack, та підтримка різних стратегій рендерингу (SSR, SSG, ISR), що робить його відмінним вибором для широкого спектру проєктів. Враховуючи ці аспекти, Next.js видається найкращим варіантом для розробників, які шукають надійний, гнучкий та продуктивний фреймворк для створення сучасних вебзастосунків на основі React.

### 3.2.3 Вибір бібліотеки для керуванням стану клієнтської частини

У світі розробки програмного забезпечення управління станом є вирішальним аспектом, який може зробити або зруйнувати успіх програми. Воно визначає, як дані зберігаються, оновлюються та отримують доступ до них протягом усього життєвого циклу програми. З'явилося багато бібліотек, які допомагають розробникам ефективно працювати з даними. Два відомі рішення - Redux і MobX – добре відомі серед розробників.

Однак, нова бібліотека Zustand заявляє про кращу альтернативу. З ростом застосунків управління сховищем Redux, діями та редукторами може стати громіздким і призвести до зниження продуктивності розробників, а іноді це змушує нас забувати деякі правила, тому що їх було занадто багато. З іншого боку MobX – страждає від проблем з продуктивністю, особливо при роботі з великими і глибоко вкладеними державними структурами. Zustand є мінімалістичним і легким рішенням, яке використовує простоту і знайомство з хуками React для управління станами у функціональних компонентах.

В порівнянні з Redux і MobX, Zustand пропонує чисте та ефективне рішення для керування станом у JavaScript-застосунках. Його легкість, простота і безшовна інтеграція з React роблять його цінним інструментом для розробників. Використовуючи Zustand та його патерн проєктування, розробники можуть спростити управління станами, покращити організацію коду та підвищити загальну продуктивність своїх застосунків [6].

## 3.3 Патерни рендерингу

У сучасному світі веброзробки існує багато різних патернів рендерингу застосунка: клієнтський рендеринг (CSR), серверний рендеринг (SSR), статична генерація (SSG) тощо. Для того щоб створити швидкий і стабільний вебзастосунок вже на етапі проєктування потрібно вибрати які патерни рендерингу застосунок буде використовувати і які це дасть переваги.

### 3.3.1 Клієнтський рендеринг

При клієнтському рендерингу (CSR) базовий HTML-контейнер відображається на сервері, а логіка, дані, шаблони та маршрутизація обробляються на стороні клієнта. CSR дозволяє створити односторінковий застосунок з хорошим користувацьким досвідом і чітким поділом клієнтського і серверного коду.

Проте, CSR також має свої недоліки. SEO може бути складнішим, оскільки пошукові роботи можуть мати обмежене розуміння JavaScript. Крім того, затримка завантаження сторінки може вплинути на користувацький досвід, особливо якщо вміст великий або клієнт має обмежену обчислювальну потужність. Обслуговування коду може також бути проблемою, коли деякі елементи повторюються між клієнтом і сервером. Отримання даних також може бути складним, оскільки вони отримуються на основі подій і можуть затримувати час загрузки сторінки [7].

Клієнтський рендеринг здебільшого використовують у застосунках без фрейморків, тобто на «чистому» React. Цей патерн не підходить через перелічені вище проблеми.

### Статичний рендеринг

Статичний рендеринг намагається вирішити проблеми TTFB, FCP , LCP та TTI, надаючи клієнту попередньо відрендерений HTML-контент. Статичний HTML-файл генерується заздалегідь для кожного маршруту, доступного користувачу. Ці файли можуть бути кешовані, що покращує продуктивність і відмовостійкість. Завантаження скриптів на стороні клієнта є мінімальним, що призводить до швидшого FCP і TTI.

Цей метод підходить для сторінок зі статичним контентом, але для деяких типів контенту, як блоги або сторінки продуктів, потрібна динаміка. Використання SSG має свої переваги, включаючи покращену продуктивність і

SEO. Однак, велика кількість HTML-файлів, залежність від хостингу і необхідність оновлення сайту при зміні контенту можуть бути проблемою [8].

**Серверний рендеринг**

SSR генерує повний HTML-код вмісту сторінки у відповідь на запит користувача. Вміст може включати дані зі сховища даних або зовнішнього API. Операції підключення та вибірки виконуються на сервері. HTML, необхідний для форматування вмісту, також генерується на сервері. Таким чином, завдяки SSR ми можемо уникнути додаткових обходів для отримання даних і шаблонування. Таким чином, код рендерингу на клієнті не потрібен, і відповідний JavaScript не потрібно надсилати клієнту. З SSR кожен запит обробляється незалежно і буде оброблятися сервером як новий запит. Навіть якщо результати двох послідовних запитів не дуже відрізняються, сервер обробить і згенерує їх з нуля [9].

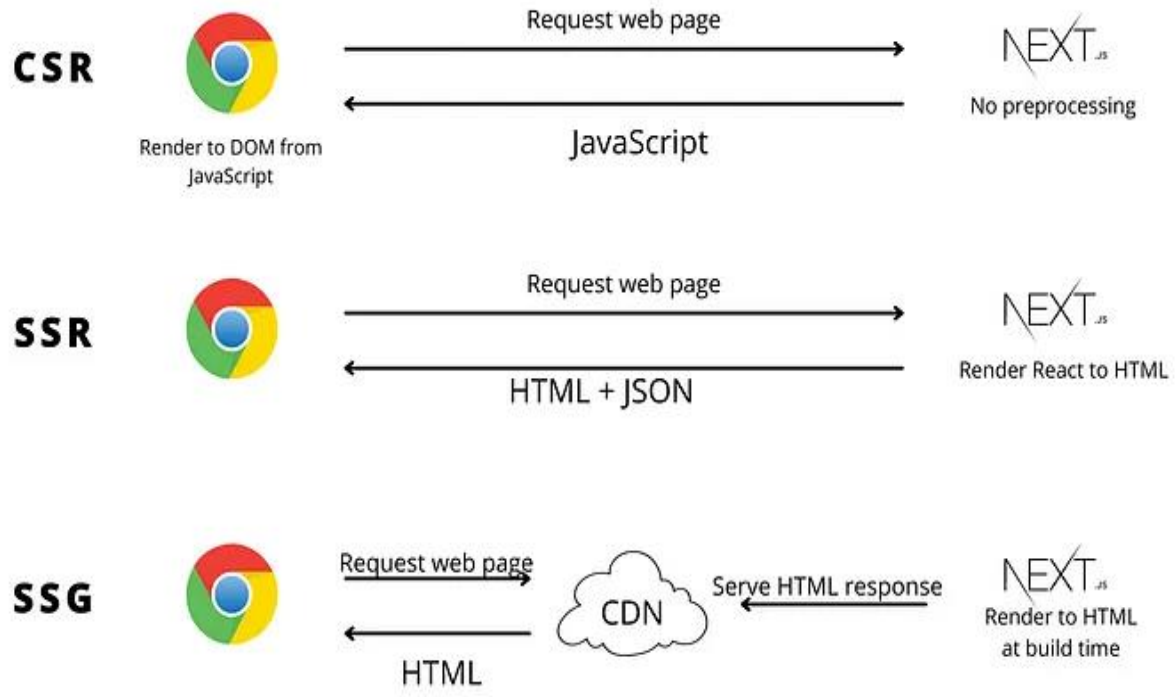


Рисунок 3.2 – Робота клієнтського, серверного та статичного рендеру.

### Інкрементна статична генерація

Інкрементна статична генерація (ISG) є оновленням статичної генерації (SSG) і допомагає розв'язати проблему динамічних даних на статичних сайтах. ISG дозволяє оновлювати існуючі сторінки і додавати нові шляхом попереднього рендерингу підмножини сторінок у фоновому режимі навіть під час надходження нових запитів на сторінки.

Процес оновлення сторінок в ISG полягає у встановленні таймауту для перебудови сторінки. Поки сторінка перевіряється на оновлення, користувач бачить попередню версію, що гарантує неперервність відображення. ISG має ряд переваг, включаючи здатність підтримувати динамічні дані без перебудови сайту, швидкість обробки даних, доступність найсвіжіших версій сторінок та послідовність регенерації на сервері.

ISG є покращенням SSG і допомагає статичним сайтам масштабуватися для великих обсягів даних, що часто змінюються, забезпечуючи швидке та ефективне оновлення сторінок [10].

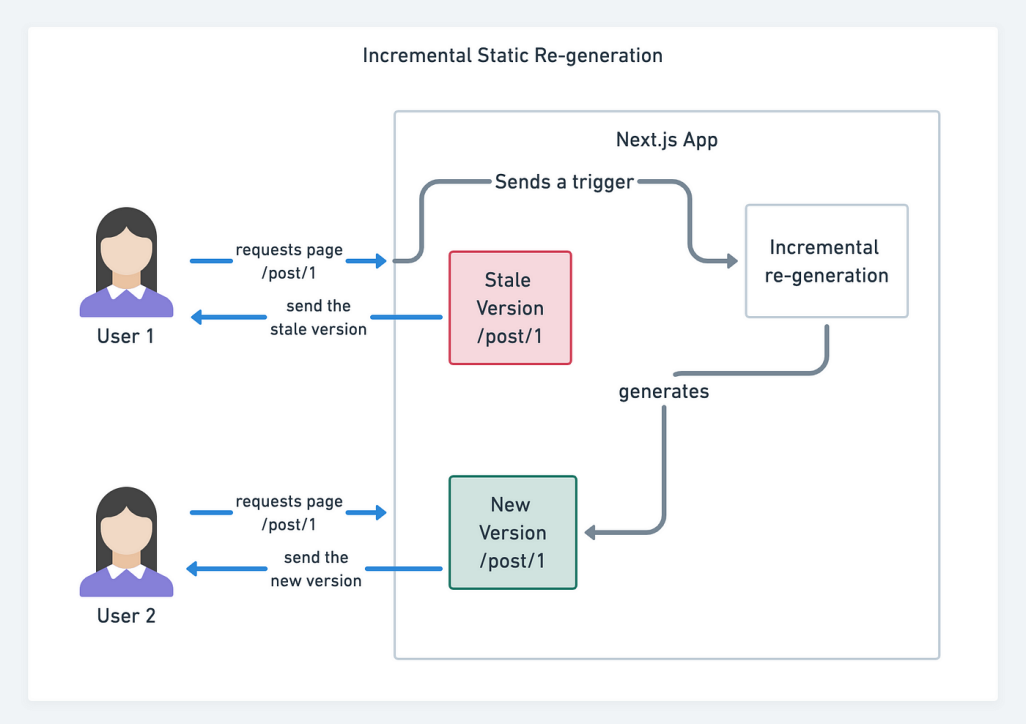


Рисунок 3.3 – Робота інкрементально-статичного рендерингу

Отже, у вебсервісі бронювання готелів використовується комбінацію серверного та інкрементально-статичного рендерингу. На сторінках, які потребують постійної динаміки, як-от, наприклад, сторінка пошуку готелів, буде використовуватися серверний рендеринг. На сторінках, які можуть бути тимчасово статичними, як наприклад, сторінки окремих готелів буде використовуватися інкрементально-статичний рендеринг.

## 3.4 Опис використовуваних технологій для розробки серверної частини

### 3.4.1 Платформа Node.js

Node.js – це платформа, яка дозволяє запускати код JavaScript на стороні сервера. Вона побудована на двигуні V8 від Google, який використовується для запуску JavaScript в браузері Chrome, дозволяючи розробникам створювати серверні програми на JavaScript [11].

Особливістю Node.js є його асинхронна обробка вводу/виводу. Це дозволяє ефективно обробляти численні запити одночасно, не блокуючи інші процеси. Завдяки цьому, Node.js ідеально підходить для створення застосунків, які потребують одночасного з'єднання у великій кількості, таких як вебсервери або мережеві програми.

Node.js включає в себе багато модулів, що дозволяють легко взаємодіяти з різними компонентами системи, такими як операційна система, мережа, файлова система.

Це надає розробникам інструменти для створення повнофункціональних серверних застосунків. До того ж, в Node.js є npm, потужний інструмент для управління бібліотеками та залежностями, що робить його затребуваним серед розробників завдяки легкості інтеграції сторонніх бібліотек. Node.js підтримує створення мережевих застосунків, які використовують такі протоколи, як TCP/IP, HTTP і WebSocket. Це дає можливість створювати вебсервери, API, мікросервіси та інші типи мережевих програм.

### 3.4.2 ORM Prisma

Prisma – це ORM наступного покоління з відкритим вихідним кодом. Вона складається з наступних частин:

– Prisma Client: Автоматично згенерований та типобезпечний конструктор запитів для Node.js та TypeScript;

– Prisma Migrate: Система міграції;

– Prisma Studio: Графічний інтерфейс для перегляду та редагування даних у базі даних.

Prisma Client можна використовувати в будь-якому бекенд-застосунку на Node.js (підтримувані версії) або TypeScript (включаючи безсерверні застосунки і мікросервіси). Це може бути REST API, GraphQL API, gRPC API або будь-що інше, що потребує бази даних. Основна мета Prisma – зробити розробників застосунків більш продуктивними при роботі з базами даних.

Серверні застосунки зазвичай є серверами API, які надають доступ до операцій з даними за допомогою таких технологій, як REST, GraphQL або gRPC. Вони зазвичай будуються як мікросервіси або монолітні застосунки і розгортаються за допомогою довготривалих серверів або безсерверних функцій. Prisma чудово підходить для всіх цих моделей застосунків і розгортання (рис. 3.4).

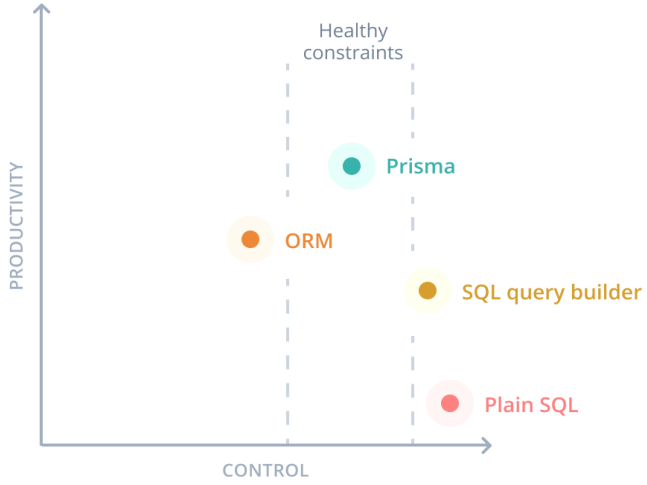


Рисунок 3.4 – Prisma балансує між продуктивністю та контролем

### 3.4.3 Авторизація за допомогою Next.Auth

Використання Next.Auth у проєкті є важливим рішенням для забезпечення безпеки та зручності авторизації користувачів. Next.Auth пропонує ряд ключових переваг і плюсів, які роблять його ідеальним вибором для впровадження систем авторизації у сучасних вебзастосунках.

NextAuth.js – це повноцінне рішення для автентифікації з відкритим вихідним кодом для Next.js застосунків, розроблене для підтримки Next.js і Serverless.

Головні плюси:

- розроблений для роботи з будь-яким сервісом OAuth;
- вбудована підтримка багатьох популярних служб входу;
- підтримує як JSON Web Tokens, так і сеанси роботи з базами даних;
- NextAuth.js можна використовувати з базою даних або без неї;
- автоматично генерує симетричні ключі підпису та шифрування для зручності розробників.

### 3.4.4 NoSQL база даних MongoDB

MongoDB є системою зберігання даних, що працює за принципом NoSQL та використовує документо-орієнтований підхід. Вона ефективно працює з даними, представленими у форматі, схожому на JSON, що робить її адаптивною та зручною для різноманітних застосунків, особливо тих, що потребують швидкісної обробки та можливості масштабування.

Головна перевага MongoDB полягає у її гнучкій структурі даних. Різноманітність форматів документів та полів сприяє легкості зберігання та обробки динамічних даних, що зробило її вибором номер один для проєктів, де вимоги до даних часто змінюються [13].

Ця база даних відрізняється високою швидкістю доступу до даних завдяки використанню алгоритмів хешування в пам'яті та підтримує горизонтальне масштабування, дозволяючи ефективно розподіляти

навантаження між серверами. MongoDB пропонує широкий спектр можливостей для роботи з даними, включно з індексацією, складними запитами, агрегацією, текстовим пошуком та іншими функціями. Вона також підтримує роботу з геоданими та забезпечує високу доступність і надійність через реплікацію даних.

В контексті порівняння SQL та NoSQL, важливо розуміти їх відмінності. SQL-бази даних базуються на строгій структурі та відносинах між таблицями, в той час як NoSQL, як MongoDB, пропонують більш гнучкий підхід з різними типами даних, що дозволяє краще адаптуватися до специфіки проєкту [14]. Вибір між SQL та NoSQL базами даних залежить від потреб проєкту. SQL ідеально підходить для випадків, коли потрібна сувора цілісність даних і складні запити, тоді як NoSQL є кращим вибором для проєктів з великою кількістю неструктурованих даних, що вимагають високої масштабованості та гнучкості. Іноді, комбінація обох типів може бути найефективнішим рішенням.

### 3.4.5 Система розрахунків Stripe

Stripe – це система, що забезпечує інфраструктуру для проведення онлайн-операцій з коштами. Вона дозволяє компаніям приймати оплату через різноманітні канали, включаючи платіжні карти, електронні гаманці, банківські перекази та інші способи оплати. Її API є зручним та простим у використанні, дозволяючи розробникам інтегрувати платіжні опції у свої програми. Це робить процес створення платіжних форм, обробки оплат, управління підписками та здійснення повернень коштів легким [15].

Однією з ключових особливостей Stripe є його високий рівень безпеки та дотримання стандартів індустрії платежів. Система захищає від шахрайства, включно з захистом даних платіжних карток і використанням двофакторної автентифікації. Stripe відповідає нормам безпеки PCI DSS, дозволяючи компаніям обробляти оплати без необхідності зберігання чутливої інформації про картки.

## Висновок до розділу 3

У даному розділі описана архітектура та відображено підхід до розробки вебсервісу бронювання готелів, заснований на сучасних технологіях та практиках. Вибір TypeScript як основної мови програмування обумовлений його строгою типізацією та сумісністю з JavaScript, що забезпечує високий рівень безпеки та легкість утримання коду. Використання React і Next.js для фронтенду дозволяє створювати інтерактивні користувацькі інтерфейси з оптимізованим завантаженням і високою продуктивністю за рахунок серверного рендерингу (SSR) та інкрементально-статичної генерації сторінок (ISG).

Бекенд-частина, побудована на Node.js і використовує Prisma як ORM для роботи з базою даних MongoDB, підкреслює ефективність асинхронного програмування та легкість управління даними. Це забезпечує швидку відповідь сервера на запити клієнта і легку інтеграцію з різними вебсервісами.

Управління станом з Zustand і автентифікація через NextAuth забезпечують зручність користувачів та безпеки даних. Особлива увага приділена патернам рендерингу, де комбінація SSR і ISG використовується для оптимізації швидкодії та забезпечення актуальності даних без необхідності повного перегенерування сторінок.

Технологічний вибір підкріплюється порівняльним аналізом інструментів фронтенду та бекенду, що дозволяє зробити висновок про їх адекватність вимогам проєкту. Використання MongoDB як NoSQL бази даних надає гнучкості управління даними.

# 4 РОЗРОБКА СЕРВІСУ БРОНЮВАННЯ ГОТЕЛІВ

## 4.1 Розробка моделей

### 4.1.1 Модель User

Модель User являє собою основу для зберігання інформації про користувачів у базі даних (рис. 4.1).

```
model User {
  id             String @id @default(auto()) @map("_id") @db.ObjectId
  name           String?
  email          String?   @unique
  emailVerified  DateTime?
  image          String?
  hashedPassword String?
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
  favoriteIds    String[] @db.ObjectId

  accounts Account[]
  listings Listing[]
  reservations Reservation[]
}
```

Рисунок 4.1 – Модель User

Докладний опис кожного поля моделі User:

– id: Унікальний ідентифікатор користувача, генерується автоматично;

– name: Ім'я користувача. Опціональне поле типу String, що означає, що ім'я може бути не вказано;

– email: Електронна пошта користувача. Опціональне поле типу String. Унікальне для кожного користувача;

– emailVerified: Дата і час підтвердження електронної пошти. Опціональне поле типу DateTime, що дає змогу відстежувати, чи було підтверджено адресу електронної пошти користувача;

– image: Посилання на зображення профілю користувача. Опціональне поле типу String;

– hashedPassword: Хешований пароль користувача. Опціональне поле типу String. Використовується для безпечного зберігання паролів користувачів;

– createdAt: Дата і час створення запису про користувача. Значення поля генерується автоматично під час створення запису;

– updatedAt: Дата і час останнього оновлення запису про користувача. Поле оновлюється автоматично при кожному оновленні запису;

– favoriteIds: Масив ідентифікаторів (у форматі ObjectId), що представляють обрані елементи користувача. Це можуть бути ідентифікатори товарів, статей або інших об'єктів, доданих користувачем в обране;

– accounts: Зв'язок один до багатьох з таблицею Account, що представляє акаунти соціальних мереж або інші зовнішні акаунти, пов'язані з користувачем;

– listings: Зв'язок один до багатьох з таблицею Listing, що представляє оголошення або лістинги, створені користувачем;

– reservations: Зв'язок один до багатьох із таблицею Reservation, що представляє бронювання або резервації, зроблені користувачем.

### 4.1.2 Модель Account

Модель Account являє собою схему облікового запису, яка використовується для зберігання даних про аутентифікацію та авторизацію користувача в системі (рис. 4.2).

```
model Account {
  id String @id @default(auto()) @map("_id") @db.ObjectId
  userId             String   @db.ObjectId
  type               String
  provider           String
  providerAccountId  String
  refresh_token      String?  @db.String
  access_token       String?  @db.String
  expires_at         Int?
  token_type         String?
  scope              String?
  id_token           String?  @db.String
  session_state      String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerAccountId])
}
```

Рисунок 4.2 – Модель Account

Докладний опис кожного поля моделі Account:

– id: Унікальний ідентифікатор облікового запису. Це поле автоматично генерується;

– userId: Ідентифікатор користувача, пов'язаний з обліковим записом. Це поле також використовує формат ObjectId і вказує на конкретного користувача в системі;

– type: Тип облікового запису. Поле може містити інформацію про те, до якого типу належить обліковий запис

– provider: Постачальник облікового запису, що вказує на сервіс або платформу, через яку здійснюється аутентифікація;

– providerAccountId: Ідентифікатор облікового запису, наданий постачальником. Це унікальний ідентифікатор, який використовується постачальником для ідентифікації облікового запису користувача;

– refresh_token: Токен оновлення, який використовується для отримання нового доступу без повторного входу користувача;

– access_token: Токен доступу, що надає тимчасовий доступ до ресурсів або функцій на стороні постачальника;

– expires_at: Час закінчення терміну дії токена доступу, виражений у цілих числах;

– token_type: Тип токена, наприклад «Bearer»;

– scope: Область дії токена, що визначає, до яких ресурсів або операцій дозволено доступ;

– id_token: Ідентифікаційний токен, що містить дані ідентифікації користувача, який може використовуватися для верифікації;

– session_state: Стан сесії, який може використовуватися для відстеження поточного стану сесії аутентифікації.

Додатково, модель Account містить зв'язок із моделлю User через поле userId, забезпечуючи зв'язок між обліковим записом і користувачем. Під час видалення користувача із системи всі пов'язані з ним облікові записи також буде видалено завдяки каскадному видаленню (onDelete: Cascade).

### 4.1.3 Модель Listing

Модель Listing пов'язана із розміщенням оголошень про здавання в оренду житла, номерів або апартаментів. Модель Listing володіє всіма необхідними атрибутами для створення, управління та подання оголошень про розміщення в системі бронювання або оренди житла (рис. 4.3).

```
model Listing {
  id             String @id @default(auto()) @map("_id") @db.ObjectId
  title String
  description String
  imageSrc String
  createdAt DateTime @default(now())
  category  String
  roomCount Int
  bathroomCount Int
  guestCount Int
  locationValue String
  userId String @db.ObjectId
  price Int

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  reservations Reservation[]
}
```

Рисунок 4.3 – Модель Listing

Докладний опис кожного поля моделі Listing:

– id: Унікальний ідентифікатор облікового запису. Це поле автоматично генерується;

– title: Заголовок оголошення, що містить короткий опис пропонованої нерухомості;

– description: Детальний опис об'єкта розміщення, включно з особливостями, зручностями та іншими важливими деталями;

– imageSrc: Посилання на зображення об'єкта розміщення;

– createdAt: Дата і час створення оголошення;

– category: Категорія розміщення, наприклад, квартира, будинок, кімната тощо, що дає змогу класифікувати оголошення за типом житла;

– roomCount: Кількість кімнат в об'єкті розміщення;

– bathroomCount: Кількість ванних кімнат в об'єкті розміщення;

– guestCount: Максимальна кількість гостей, яка може розміститися в даному житлі;

– locationValue: Значення, що описує місце розташування об'єкта розміщення, наприклад, адреса або координати;

– userId: Ідентифікатор користувача, який розмістив оголошення;

– price: Ціна оренди за певний період;Зв'язки:

– user (User): Зв'язок між оголошенням і користувачем, що вказує на власника оголошення. onDelete: Cascade означає, що під час видалення користувача буде видалено і всі його оголошення.

– reservations (Reservation[]): Масив бронювань, пов'язаних із цим оголошенням. Дозволяє відстежувати всі бронювання об'єкта розміщення.

### 4.1.4 Модель Reservation

Модель Reservation являє собою схему бронювання, що використовується в системах, де необхідно зарезервувати будь-які ресурси або послуги, наприклад, у системах оренди житла, бронювання готелів або інших послуг (рис. 4.4).

```
model Reservation {
  id String @id @default(auto()) @map("_id") @db.ObjectId
  userId String @db.ObjectId
  listingId String @db.ObjectId
  startDate DateTime
  endDate DateTime
  totalPrice Int
  createdAt DateTime @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  listing Listing @relation(fields: [listingId], references: [id], onDelete: Cascade)
}
```

Рисунок 4.4 – Модель Reservation

Зв'язки з іншими моделями (User і Listing) реалізовано через поля user і listing відповідно. Ці зв'язки дають змогу звертатися до даних користувача, який здійснив бронювання, і до об'єкта бронювання безпосередньо з моделі Reservation, полегшуючи таким чином доступ до пов'язаних даних та їх обробку.

Докладний опис кожного поля моделі Reservation:

– id: Унікальний ідентифікатор бронювання, що автоматично генерується під час створення запису;

– userId: Ідентифікатор користувача, який створив бронювання. Тип даних String, також використовує @db.ObjectId. Зв'язок з моделлю User вказується через @relation, забезпечуючи цілісність даних при видаленні користувача (опція onDelete: Cascade);

– listingId: Ідентифікатор об'єкта бронювання (наприклад, житла, місця або послуги), тип даних String, з @db.ObjectId. Аналогічно userId, пов'язаний з моделлю Listing через @relation, з каскадним видаленням;

– startDate і endDate: Дати початку та закінчення бронювання. Використовують тип даних DateTime для зберігання точних дат і часу;

– createdAt: Дата і час створення оголошення;

– totalPrice: Загальна вартість бронювання;

– createdAt: Дата і час створення запису про бронювання, автоматично встановлюється поточним часом завдяки @default(now()).

Таким чином, модель Reservation забезпечує зручне та ефективне управління процесами бронювання в системі, даючи змогу зберігати всі необхідні дані про бронювання та їхніх учасників у структурованому та взаємопов'язаному вигляді.

### 4.2 Реалізація реєстрації

Процес створення нового користувача в системі реалізовано з використанням фреймворку Next.js та ORM Prisma для роботи з базою даних. Для забезпечення безпеки паролів використовується бібліотека bcrypt, яка дозволяє хешувати паролі перед зберіганням їх в базі даних.

Процес реєстрації розпочинається з отримання даних користувача – електронної пошти, імені та пароля – через запит типу POST. Після цього введений користувачем пароль хешується за допомогою bcrypt, що забезпечує додаткову безпеку, оскільки в базі даних зберігається не сам пароль, а його хеш (рис. 4.5).

```
import { NextResponse } from "next/server";
import bcrypt from "bcrypt";

import prisma from "@/app/libs/prismadb";

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

   const hashedPassword = await bcrypt.hash(password, 12);

   const user = await prisma.user.create({
    data: {
      email,
      name,
      hashedPassword,
    }
  });

  return NextResponse.json(user);
}
```

Рисунок 4.5 – Код методу для реєстрації

Наступним кроком є створення запису користувача в базі даних з використанням Prisma, де зазначаються електронна пошта, ім'я та хешований пароль. В кінці процесу реєстрації, дані нового користувача повертаються у відповіді на запит у форматі JSON.

Цей процес є ключовим для реалізації системи реєстрації та аутентифікації у вебзастосунках, оскільки він не тільки дозволяє створювати нові облікові записи, але й забезпечує захист користувацьких даних за допомогою надійного методу хешування паролів. Використання Prisma як ORM спрощує роботу з базою даних, роблячи код більш читабельним та легшим для підтримки.

### 4.3 Реалізація авторизації

Процес аутентифікації в застосунку реалізовано за допомогою бібліотеки NextAuth для фреймворка Next.js. Він містить адаптер Prisma для роботи з базою даних через ORM Prisma, а також надає кілька способів автентифікації: через акаунти GitHub, Google і з використанням локальних облікових даних (email і пароль). Адаптер Prisma: Інтегрує NextAuth з базою даних через Prisma, даючи змогу керувати користувачами та сесіями безпосередньо через ORM.

Сесія: Конфігурація сесії використовує стратегію JSON Web Tokens (JWT), що є поширеним вибором для управління сесіями у вебзастосунках.

Провайдери аутентифікації:

– GitHub: Дозволяє користувачам входити в систему через їхні акаунти GitHub, використовуючи OAuth;

– Google: Аналогічно надає можливість аутентифікації через акаунти Google;

– Credentials: Цей провайдер використовується для аутентифікації за email і паролем. При цьому відбувається перевірка існування користувача в базі даних і порівняння хешованих паролів за допомогою бібліотеки bcrypt (рис. 4.6).

```
async authorize(credentials) {
  if (!credentials?.email || !credentials?.password) {
    throw new Error('Invalid credentials');
  }

  const user = await prisma.user.findUnique({
    where: {
      email: credentials.email
    }
  });

  if (!user || !user?.hashedPassword) {
    throw new Error('Invalid credentials');
  }

  const isCorrectPassword = await bcrypt.compare(
    credentials.password,
    user.hashedPassword
  );

  if (!isCorrectPassword) {
    throw new Error('Invalid credentials');
  }

  return user;
}
```

Рисунок 4.6 – Метод для авторизації

Next.Auth являє собою потужну і гнучку основу для системи аутентифікації в сучасних вебзастосунках, забезпечуючи безпеку і зручність використання як для розробників, так і для кінцевих користувачів.

### 4.4 Реалізація додавання та видалення до списку «улюблених» готелів

Спочатку, для додавання готелю до улюблених, використовується HTTP метод POST. В цьому методі спочатку здійснюється спроба ідентифікації поточного користувача. Якщо користувач не ідентифікований, метод повертає помилку. Далі перевіряється валідність ID об'єкту, який користувач бажає додати до свого списку улюблених. Якщо ID є валідним, воно додається до списку «улюблених» ID поточного користувача, після чого оновлюється запис користувача в базі даних з новим списком улюблених (рис. 4.7).

```
export async function POST(
  request: Request,
  { params }: { params: IParams }
) {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return NextResponse.error();
  }

  const { listingId } = params;

  if (!listingId || typeof listingId !== 'string') {
    throw new Error('Invalid ID');
  }

  let favoriteIds = [...(currentUser.favoriteIds || [])];

  favoriteIds.push(listingId);

  const user = await prisma.user.update({
    where: {
      id: currentUser.id
    },
    data: {
      favoriteIds
    }
  });

  return NextResponse.json(user);
}
```

Рисунок 4.7 – Код методу додавання готелів до списку «улюблених»

Для видалення готелю з улюблених використовується HTTP-метод DELETE: ідентифікація користувача, перевірка валідності ID готелю, видалення ID зі списку улюблених, і оновлення запису користувача в базі даних (рис. 4.8).

```
export async function DELETE(
  request: Request,
  { params }: { params: IParams }
) {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return NextResponse.error();
  }

  const { listingId } = params;

  if (!listingId || typeof listingId !== 'string') {
    throw new Error('Invalid ID');
  }

  let favoriteIds = [...(currentUser.favoriteIds || [])];

  favoriteIds = favoriteIds.filter((id) => id !== listingId);

  const user = await prisma.user.update({
    where: {
      id: currentUser.id
    },
    data: {
      favoriteIds
    }
  });

  return NextResponse.json(user);
}
```

Рисунок 4.8 – Код методу видалення готелів з списку «улюблених»

## 4.5 Опис інтерфейсу системи

### Форма реєстрації

Система надає інтерфейс для реєстрації нових користувачів через модальне вікно, яке забезпечує інтуїтивно зрозумілий та зручний процес створення облікового запису. Реалізація модального вікна реєстрації використовує функціональний компонент «RegisterModal», розроблений на

основі «React» та «Next.js» для оптимізації взаємодії користувача з веб-додатком.

Модальне вікно реєстрації має сучасний дизайн, з чітко структурованими елементами вводу та навігації. Використання компонентів, таких як «Modal», «Input», «Heading», і «Button», допомагає створити консистентний та естетично привабливий інтерфейс. Візуальні елементи, такі як іконки для входу через «Google» та «GitHub», сприяють зручності використання (рис. 4.9).

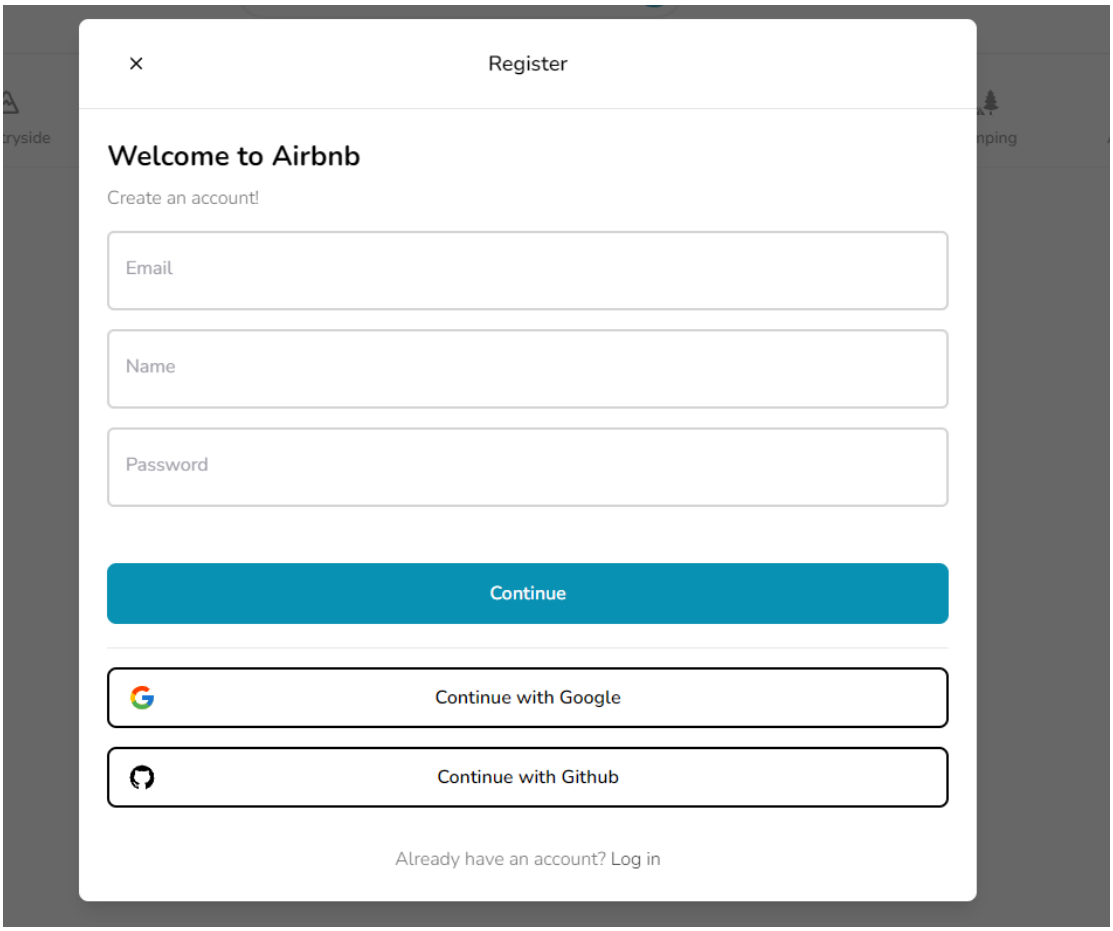


Рисунок 4.9 – Форма рєстрації

**Форма авторизації**

Форма авторизації є ключовим елементом інтерфейсу користувача, який дозволяє користувачам увійти в систему, використовуючи свої персональні дані. Ця форма реалізована як модальне вікно, що забезпечує зручність та ефективність взаємодії з користувачем. Розробка форми здійснена на платформі «React» з використанням бібліотеки «next-auth» для автентифікації,

а також інших сучасних технологій та інструментів для розробки інтерактивних веб-інтерфейсів (рис. 4.10).

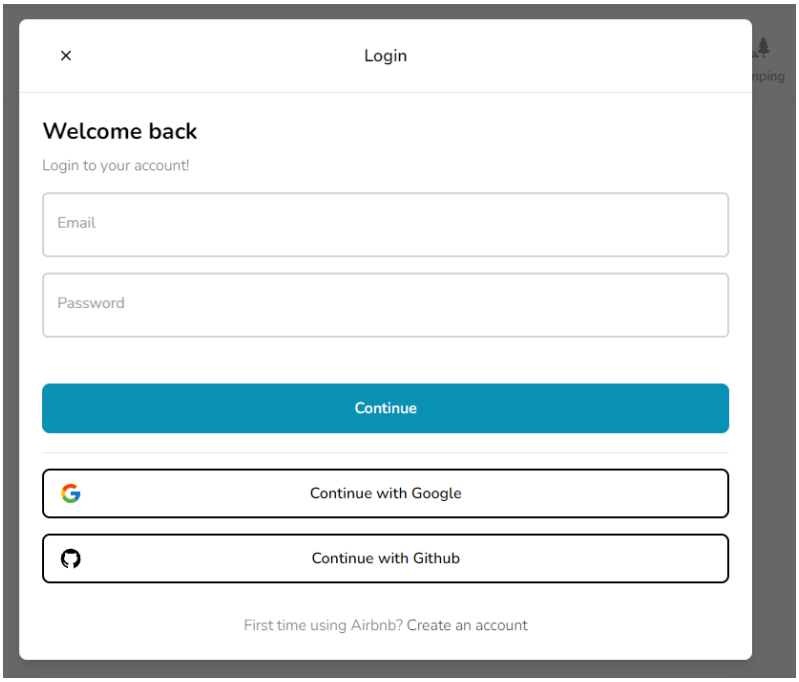


Рисунок 4.10 – Форма авторизації

Основні компоненти форми включають:

– вхідні поля: Для електронної пошти та пароля, з можливістю відображення помилок введення. Кожне поле містить мітку та підтримку необхідних атрибутів для взаємодії з користувачем;

– кнопки для соціальної автентифікації: Дозволяють користувачам продовжити авторизацію через «Google» або «Github», використовуючи іконки «FcGoogle» та «AiFillGithub» відповідно, для забезпечення візуального розпізнавання;

– перехід на реєстрацію: Посилання для переходу до форми реєстрації, якщо користувач відвідує сайт вперше.

**Навігація**

У навігаційному меню розміщено пошуковий модуль, використовуючи ключові слова, дати, кількість гостей тощо. Для зручності використання, пошуковий модуль адаптований до різних розмірів екрану.

Меню користувача розташоване в правій частині навігаційної панелі та надає доступ до особистого кабінету користувача, де він може керувати своїми подорожами, улюбленими місцями, бронюваннями та нерухомістю. Якщо користувач не авторизований, він може легко увійти у свій акаунт або зареєструватися через це меню. Також присутня кнопка для додавання нового місця, що спрощує для власників нерухомості процес публікації їхніх об'єктів на сайті.

Розділ категорій, розташований безпосередньо під панеллю навігації, дає змогу користувачам швидко переходити до різних сегментів сайту, як-от популярні напрямки, спеціальні пропозиції тощо, що робить процес пошуку зручнішим та ефективнішим (рис. 4.11).

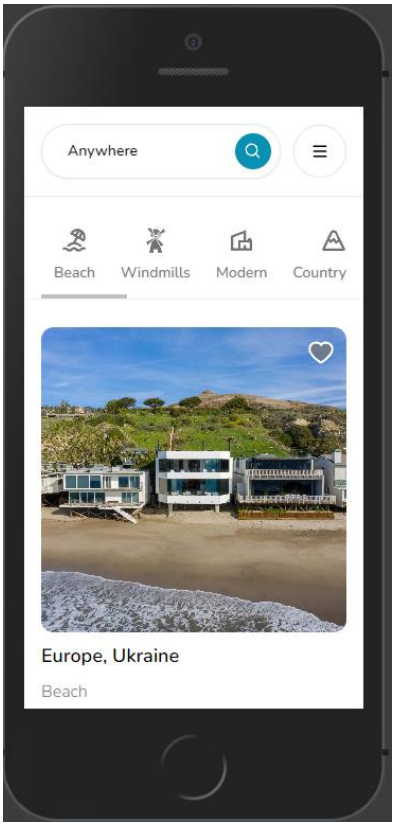


Рисунок 4.11 – Навігація сайту в мобільній версії сайту

**Сторінка орендуємого об'єкта**

На сторінці орендуємого об'єкту відображається інформація про об'єкт: власник, назва об'єкта, додатковий опис, на скількох людей розраховано, кількість спальних кімнат і ванних. Також на сторінці відображається

категорія, до якої належить об'єкт, наприклад: "Пляж", "Кемпінг", "Люкс", "Сучасні" тощо (рис. 4.12).

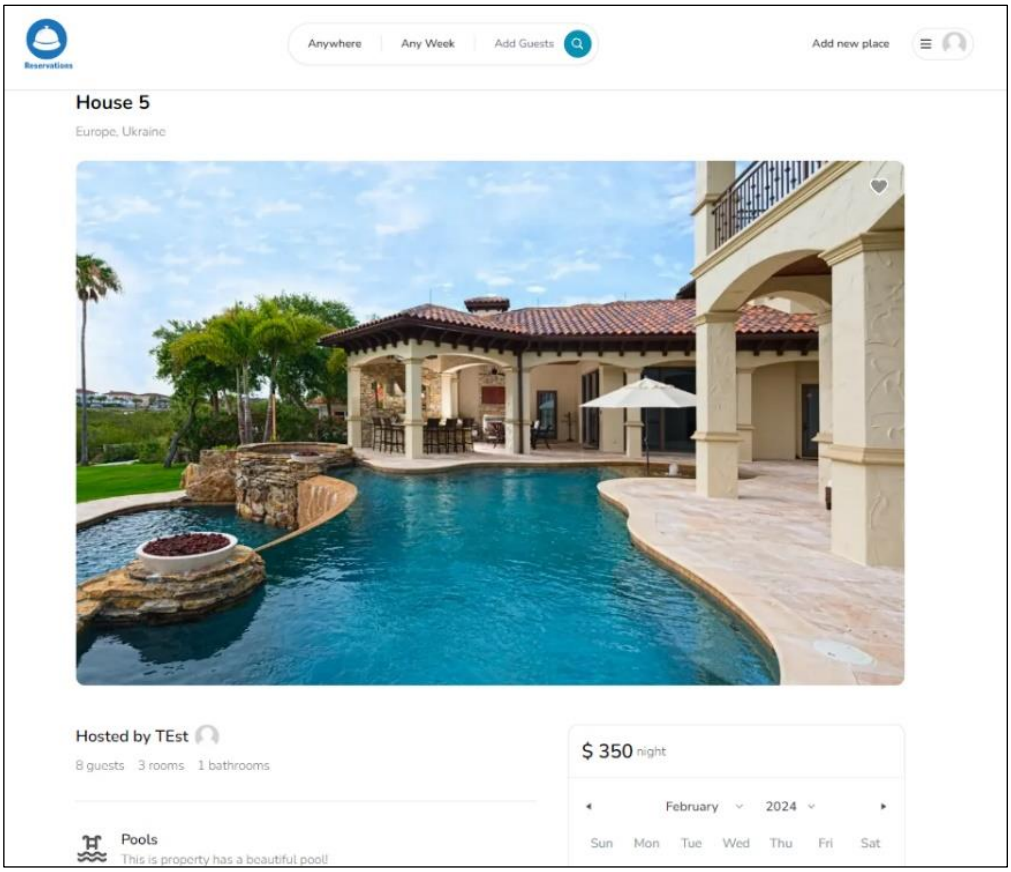


Рисунок 4.12 – Сторінка орендуємого об'єкту, верхня частина

На сторінці також є мапа, щоб дізнатися, де знаходиться об'єкт, календар для резервації, а також ціна за обраний період резервації (рис. 4.13).

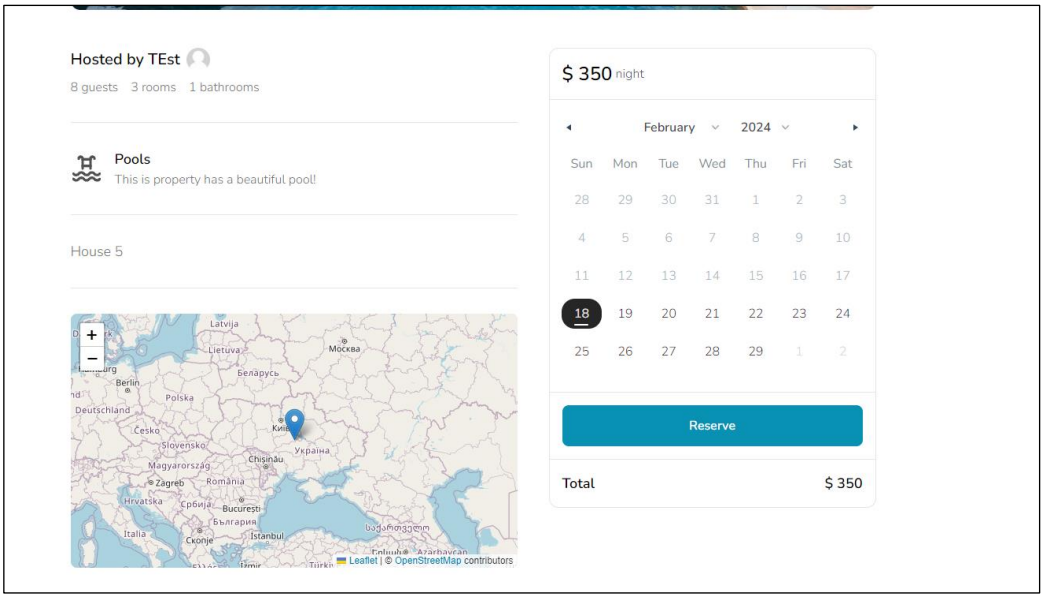


Рисунок 4.13 – Сторінка орендуємого об'єкту, нижня частина

## 4.6 Тестування розробленої системи

Тестування розробленої системи - це невід'ємний крок у процесі створення програмного забезпечення, який дає змогу перевірити правильність функціонування системи та виявити потенційні помилки до її запуску у виробництво. Щоб забезпечити високу якість продукту, тестування має проводитися ретельно й організовано.

Першим і найважливішим кроком у перевірці надійності та безпеки будь-якої системи є детальний аналіз процесу реєстрації та авторизації користувачів. Цей процес є ключовим для забезпечення доступу до персоналізованих функцій та захисту конфіденційної інформації. В рамках цього дослідження, ми ініціюємо створення тестового облікового запису, що дозволяє відслідкувати всі етапи взаємодії користувача з системою від початку і до кінця. На першому етапі, необхідно зосередитись на ретельній реєстрації користувача, що включає в себе заповнення форми з основними даними. Важливо звернути увагу на те, як система обробляє та валідує введену інформацію, а також які механізми захисту від помилкових або шкідливих введень активовані (рис. 4.14).

Після успішного завершення реєстрації, потрібно проаналізувати, які дані були передані та збережені в базі даних, з особливою увагою до їх захищеності та шифрування. Наступний крок полягає у верифікації процесу авторизації користувача, який є критично важливим для забезпечення безпечного доступу до облікового запису. Вхід в систему здійснюється за допомогою тестового облікового запису, щоб переконатися у правильності роботи механізмів ідентифікації та автентифікації. Завдяки цьому комплексному підходу до перевірки процесів реєстрації та авторизації можливо не тільки забезпечити високий рівень безпеки системи, але й виявити потенційні слабкі місця для їх подальшого усунення. Цей етап є невід'ємною частиною загального процесу аудиту безпеки, спрямованого на створення надійної та ефективної інформаційної системи.

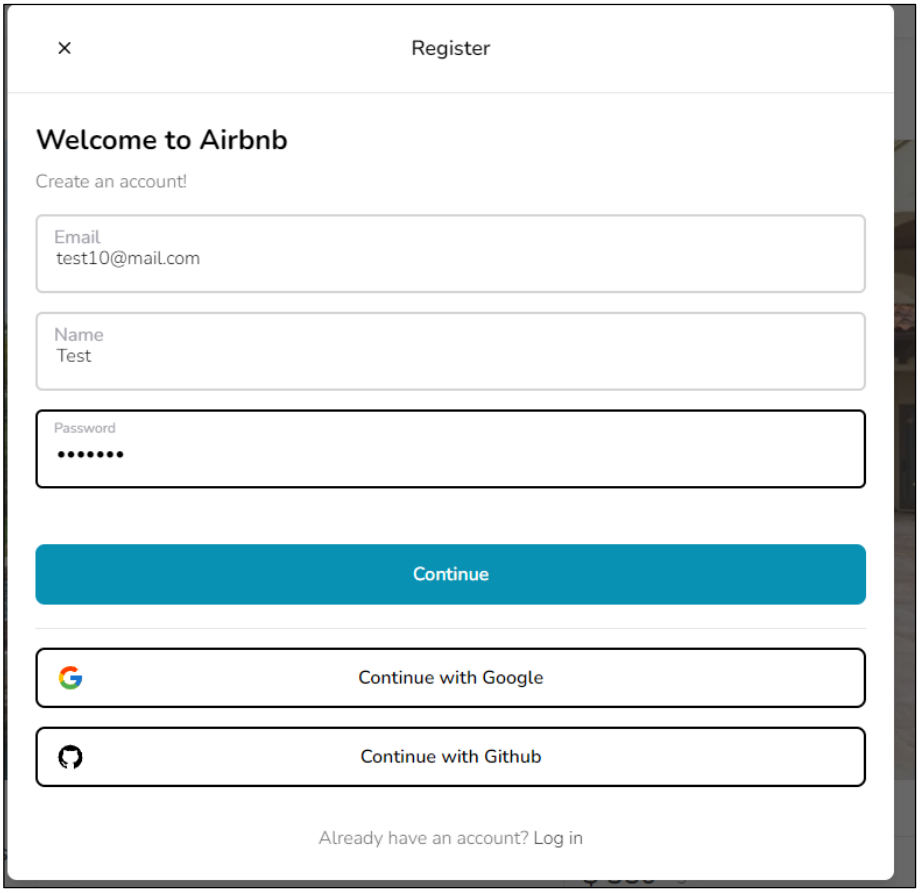


Рисунок 4.14 – Заповнення форми реєстрації тестовими даними

Тестовий акаунт коректно додано до бази даних, пароль залишається конфенденційним, оскільки перед надсиланням форми на сервер пароль було зашифровано (рис. 4.15).

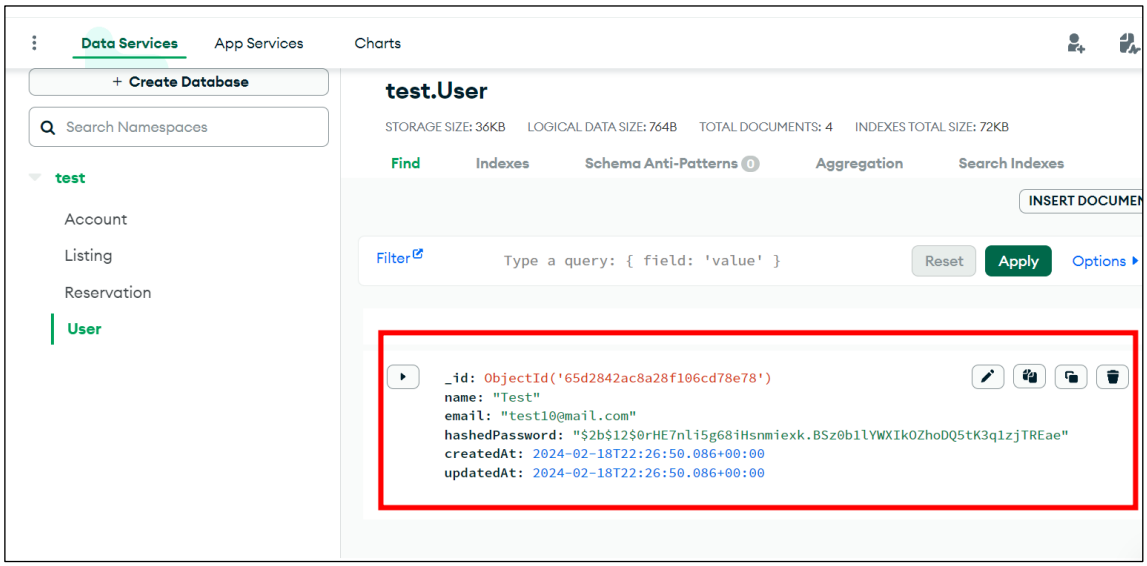


Рисунок 4.15 – Дані зареєстрованого тестового користувача в базі даних

Далі протестуємо авторизацію, для цього вводимо дані тестового користувача у формі авторизації (рис. 4.16).

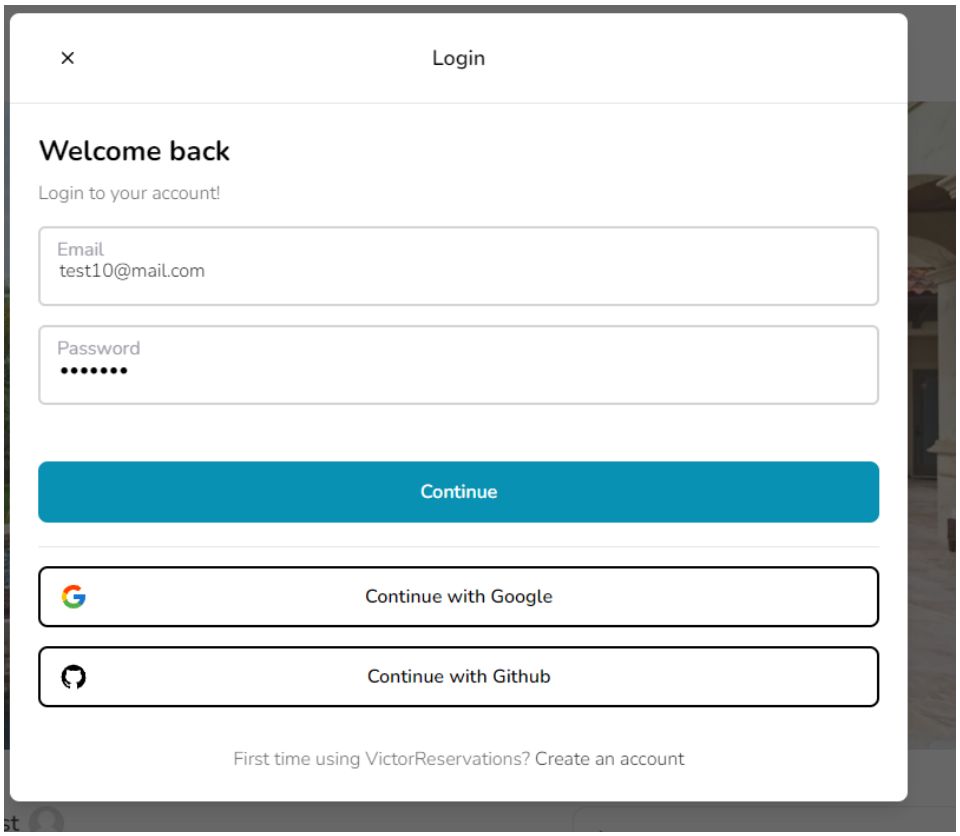


Рисунок 4.16 – Процес авторизації

Після успішної авторизації потрапляємо на сторінку списку існуючих об'єктів для оренди, а також у меню навігації з'явилися посилання на нові доступні сторінки (рис. 4.17).

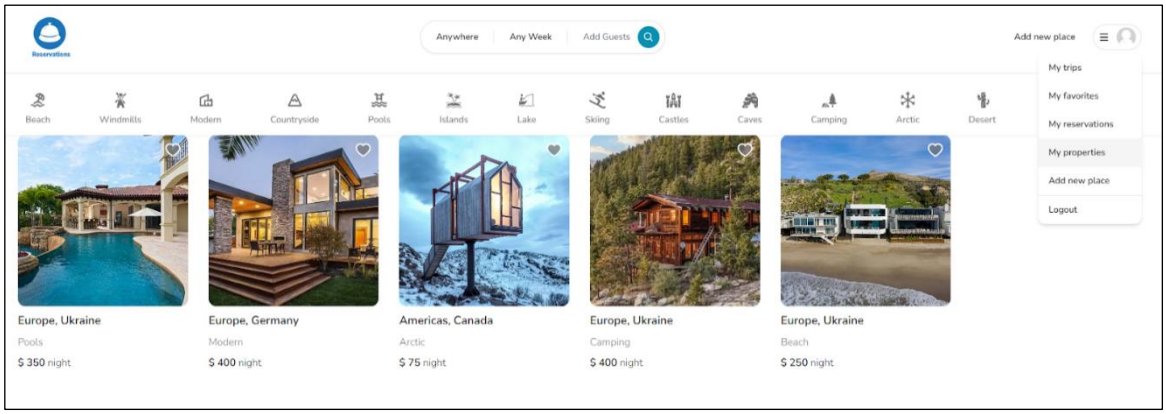


Рисунок 4.17 – Головна сторінка після успішного логування

## Висновок до розділу 4

В розділі «Розробка сервісу бронювання готелів» детально описано створення та імплементацію ключових компонентів системи, яка дозволяє користувачам ефективно взаємодіяти з платформою для резервації житла. Розроблені моделі "User", "Account", "Listing", та "Reservation" формують основу бази даних і визначають взаємовідносини між різними елементами системи, забезпечуючи зберігання та обробку інформації про користувачів, їхні облікові записи, оголошення про оренду та бронювання.

Процеси реєстрації та авторизації користувачів використовують сучасні технології та практики безпеки, зокрема хешування паролів та інтеграцію з соціальними мережами через NextAuth, що забезпечує зручність та безпеку входу в систему. Механізми додавання та видалення готелів зі списку улюблених реалізовані через HTTP методи POST та DELETE, що дозволяє користувачам персоналізувати свій досвід користування платформою.

Інтерфейс користувача, розроблений на основі React та Next.js, надає інтуїтивно зрозумілі та зручні інструменти для реєстрації, авторизації, пошуку та бронювання житла. Навігаційна панель та форми реєстрації/авторизації, а також сторінки оголошень і бронювань, оптимізовані для різних типів пристроїв, підвищуючи зручність використання сервісу.

Комплексне тестування системи, включаючи аналіз процесів реєстрації, авторизації та взаємодії з інтерфейсом, показало високу надійність та ефективність розробленої платформи.

Таким чином, розробка сервісу бронювання готелів є комплексним рішенням, яке враховує усі аспекти взаємодії користувачів з системою, від реєстрації та авторизації до пошуку, бронювання та управління улюбленими готелями, забезпечуючи зручність, безпеку та високу якість обслуговування.

# ВИСНОВКИ

У цій кваліфікаційній роботі була висвітлена актуальна проблема покращення онлайн-сервісу бронювання готелів, які відіграють ключову роль у спрощенні процесу планування подорожей, Мета роботи полягала в підвищення ефективності та зручності процесу бронювання готелів онлайн за допомогою розробки сучасного, інтуїтивно зрозумілого та надійного спеціалізованого вебсервісу для бронювання готелів.

Для досягнення цієї мети були вирішені конкретні завдання:

– проведено аналіз ринку вебсервісів для бронювання готелів та визначено основні потреби користувачів;

– проаналізовано сучасні методології розробки вебзастосунків та вибрати оптимальні для даного проєкту;

– проаналізовано особливості та можливості технологій React та MongoDB у контексті розробки вебсервісів;

– розроблено архітектуру та дизайн вебсервісу, забезпечуючи зручний інтерфейс та високу швидкість роботи;

– реалізовано серверну частину вебсервісу з використанням MongoDB для ефективного управління даними;

– розроблено клієнтську частину з використанням React, зосередившись на зручності інтерфейсу та інтуїтивності користування;

– проведено комплексне тестування вебсервісу, щоб забезпечити його надійність та безпеку.

Інноваційність розробки полягає в унікальній інтеграції технологій React і MongoDB для створення масштабованого сервісу бронювання готелів, що пропонує персоналізований користувацький досвід через інтуїтивні функції управління обраними об'єктами та бронюваннями. Цей підхід дає змогу динамічно адаптуватися до запитів користувачів, забезпечуючи високу продуктивність і безпеку даних. Особлива увага приділяється поліпшенню інтерфейсу та користувацької взаємодії, що робить сервіс не тільки

функціональним, а й зручним для кінцевого користувача, виділяючи його на тлі стандартних рішень у сфері бронювання.

Практична цінність роботи полягає в тому, що вона відкриває нові можливості для поліпшення системи бронювання готелів. Використання технологій React та MongoDB дозволяє забезпечити ефективний моніторинг та аналіз інформації щодо потреб ймовірних постояльців, що в свою чергу допомагає оптимізувати наповнення готелів та відкриває шлях до ще більш ефективного та інноваційного планування наповнення готелів та обслуговування відпочиваючих.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Хоменко В. М., Журавська І. М. Сервіс бронювання готелів на основі технологій React та MongoDB. *Free and Open Source Software (FOSS'2024)* : тези доп. XV Міжнар. наук.-практ. конф. / Харків. нац. економ. ун-т ім. Семена Кузнеця. Харків, 13–14 лютого 2024 р. Харків : ХНЕУ ім. Семена Кузнеця, 2024.

2. Online Travel Booking Service Market Size, Share & Trends Analysis Report By Service Type (Vacation Packages, Transportation Booking), By Booking Method, By Device, By Region, And Segment Forecasts, 2022–2030. URL: https://www.grandviewresearch.com/industry-analysis/online-travel-booking-service-market-report (Last accessed: 16.01.2024).

3. Менеджмент туризму. URL: https://library.if.ua (дата звернення: 30.12.2023).

4. Kintonova A. J., Suleimenova B. B., Yensebaev N. A. Web application development technologies. *Yessenov Science Journal.* Dec. 2023. DOI: 10.56525/ZESO4158.

5. Booking.com. URL: https://www.booking.com/ (Last accessed: 30.12.2023).

6. Expedia. URL: https://www.expedia.com/ (Last accessed: 30.12.2023).

7. Zustand: the real definition of elegant state. URL: https://medium.com/inside-meteor/zustand-the-real-definition-of-elegant-state-320413c69541 (Last accessed: 30.12.2023).

8. Client-side Rendering. URL: https://www.patterns.dev/react/client-side-rendering (Last accessed: 30.12.2023).

9. Static Rendering. URL: https://www.patterns.dev/react/static-rendering (Last accessed: 30.12.2023).

10. Server-side Rendering: https://www.patterns.dev/react/server-side-rendering (Last accessed: 30.12.2023).

11. Incremental Static Generation. URL: https://www.patterns.dev/react/incremental-static-rendering (Last accessed: 30.12.2023).

12. NoSQL and SQL DB’s: What's the real difference? URL: https://www.ibm.com/cloud/blog/sql-vs-nosq (Last accessed: 30.12.2023).

13. What is the virtual DOM in React? URL: https://blog.logrocket.com/virtual-dom-react/ (Last accessed: 30.12.2023).

14. Node.js – Official Website. URL: https://nodejs.org/en/ (Last accessed: 30.12.2023).

15. Express.js – Official website. URL: https://expressjs.com/ (Last accessed: 30.12.2023).

16. What is MongoDB and Why to use it? URL: https://www.dotnettricks.com/learn/mongodb/what-ismongodb-and-why-to-use-it (Last accessed: 30.12.2023).

17. Stripe | Get started with the Stripe API. URL: https://stripe.com/docs/development/get-started (Last accessed: 30.12.2023).

18. VS Code Getting started. URL: https://code.visualstudio.com (Last accessed: 30.12.2023).

19. Wirya M. S., Widiantara G. A. B., Susila M. G. D. Application of the OPERA information system in hotel reservations to increase the productivity of the front office department (Case study at the Grand Hyatt Hotel Bali). *Jurnal Manajemen Pelayanan Hotel.* 2023. Vol. 7(2):744. DOI: 10.37484/jmph.070209.

20. Rinayanthi N. M. Impact of hotel reservation site reviews on the decision to visit Andaz Bali. *Jurnal Manajemen Pelayanan Hotel*. Dec. 2023. 7(2):972. DOI: 10.37484/jmph.070218.

21. Balalle H., Prasad M. L. D.,Madhuwantha S. M. A. Personalized hotel reservation system for various occasions. March 2022. 9 p. DOI: 10.13140/RG.2.2.11322.26568.

# ДОДАТОК А

## Код програми

### А.1 Код “обгортки” *layout.tsx*

```
import { Nunito } from 'next/font/google'

import Navbar from '@/app/components/navbar/Navbar';
import LoginModal from '@/app/components/modals/LoginModal';
import RegisterModal from '@/app/components/modals/RegisterModal';
import SearchModal from '@/app/components/modals/SearchModal';
import RentModal from '@/app/components/modals/RentModal';

import ToasterProvider from '@/app/providers/ToasterProvider';

import './globals.css'
import ClientOnly from './components/ClientOnly';
import getCurrentUser from './actions/getCurrentUser';

export const metadata = {
  title: 'VictorReservations',
  description: 'VictorReservations Clone',
}

const font = Nunito({
  subsets: ['latin'],
});

export default async function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  const currentUser = await getCurrentUser();

  return (
    <html lang="en">
      <body className={font.className}>
        <ClientOnly>
          <ToasterProvider />
          <LoginModal />
          <RegisterModal />
          <SearchModal />
          <RentModal />
          <Navbar currentUser={currentUser} />
        </ClientOnly>
        <div className="pb-20 pt-28">
          {children}
        </div>
      </body>
    </html>
  )
}
```

```
import { Nunito } from 'next/font/google'

import Navbar from '@/app/components/navbar/Navbar';
import LoginModal from '@/app/components/modals/LoginModal';
import RegisterModal from '@/app/components/modals/RegisterModal';
import SearchModal from '@/app/components/modals/SearchModal';
import RentModal from '@/app/components/modals/RentModal';

import ToasterProvider from '@/app/providers/ToasterProvider';

import './globals.css'
import ClientOnly from './components/ClientOnly';
import getCurrentUser from './actions/getCurrentUser';

export const metadata = {
  title: 'VictorReservations',
  description: 'VictorReservations Clone',
}

const font = Nunito({
  subsets: ['latin'],
});

export default async function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  const currentUser = await getCurrentUser();

  return (
    <html lang="en">
      <body className={font.className}>
        <ClientOnly>
          <ToasterProvider />
          <LoginModal />
          <RegisterModal />
          <SearchModal />
          <RentModal />
          <Navbar currentUser={currentUser} />
        </ClientOnly>
        <div className="pb-20 pt-28">
          {children}
        </div>
      </body>
    </html>
  )
}
```

## А.2 Код домашньої сторінки /app/page.tsx

```
import Container from "@/app/components/Container";
import ListingCard from "@/app/components/listings/ListingCard";
import EmptyState from "@/app/components/EmptyState";

import getListings, {
  IListingsParams
} from "@/app/actions/getListings";
import getCurrentUser from "@/app/actions/getCurrentUser";
import ClientOnly from "./components/ClientOnly";

interface HomeProps {
  searchParams: IListingsParams
};

const Home = async ({ searchParams }: HomeProps) => {
  const listings = await getListings(searchParams);
  const currentUser = await getCurrentUser();

  if (listings.length === 0) {
    return (
      <ClientOnly>
        <EmptyState showReset />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <Container>
        <div
          className="
            pt-24
            grid
            grid-cols-1
            sm:grid-cols-2
            md:grid-cols-3
            lg:grid-cols-4
            xl:grid-cols-5
            2xl:grid-cols-6
            gap-8
          "
        >
          {listings.map((listing: any) => (
            <ListingCard
              currentUser={currentUser}
              key={listing.id}
              data={listing}
            />
```

```
          ))}
        </div>
      </Container>
    </ClientOnly>
  )
}

export default Home;
```

```
import Container from "@/app/components/Container";
import ListingCard from "@/app/components/listings/ListingCard";
import EmptyState from "@/app/components/EmptyState";

import getListings, {
  IListingsParams
} from "@/app/actions/getListings";
import getCurrentUser from "@/app/actions/getCurrentUser";
import ClientOnly from "./components/ClientOnly";

interface HomeProps {
  searchParams: IListingsParams
};

const Home = async ({ searchParams }: HomeProps) => {
  const listings = await getListings(searchParams);
  const currentUser = await getCurrentUser();

  if (listings.length === 0) {
    return (
      <ClientOnly>
        <EmptyState showReset />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <Container>
        <div
          className="
            pt-24
            grid
            grid-cols-1
            sm:grid-cols-2
            md:grid-cols-3
            lg:grid-cols-4
            xl:grid-cols-5
            2xl:grid-cols-6
            gap-8
          "
        >
          {listings.map((listing: any) => (
```

```
            <ListingCard
              currentUser={currentUser}
              key={listing.id}
              data={listing}
            />
          ))}
        </div>
      </Container>
    </ClientOnly>
  )
}

export default Home;
```

## А.3 Код сторінки /app/favorites/page.tsx

```
import EmptyState from "@/app/components/EmptyState";
import ClientOnly from "@/app/components/ClientOnly";

import getCurrentUser from "@/app/actions/getCurrentUser";
import getFavoriteListings from "@/app/actions/getFavoriteListings";

import FavoritesClient from "./FavoritesClient";

const ListingPage = async () => {
  const listings = await getFavoriteListings();
  const currentUser = await getCurrentUser();

  if (listings.length === 0) {
    return (
      <ClientOnly>
        <EmptyState
          title="No favorites found"
          subtitle="Looks like you have no favorite listings."
        />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <FavoritesClient
        listings={listings}
        currentUser={currentUser}
      />
    </ClientOnly>
  );
}

export default ListingPage;
```

```
import EmptyState from "@/app/components/EmptyState";
import ClientOnly from "@/app/components/ClientOnly";

import getCurrentUser from "@/app/actions/getCurrentUser";
import getFavoriteListings from
"@/app/actions/getFavoriteListings";

import FavoritesClient from "./FavoritesClient";

const ListingPage = async () => {
  const listings = await getFavoriteListings();
  const currentUser = await getCurrentUser();
```

```
  if (listings.length === 0) {
    return (
      <ClientOnly>
        <EmptyState
          title="No favorites found"
          subtitle="Looks like you have no favorite listings."
        />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <FavoritesClient
        listings={listings}
        currentUser={currentUser}
      />
    </ClientOnly>
  );
}

export default ListingPage;
```

## А.4 Код компоненту FavoritesClient.tsx

```
import { SafeListing, SafeUser } from "@/app/types";

import Heading from "@/app/components/Heading";
import Container from "@/app/components/Container";
import ListingCard from "@/app/components/listings/ListingCard";

interface FavoritesClientProps {
  listings: SafeListing[],
  currentUser?: SafeUser | null,
}

const FavoritesClient: React.FC<FavoritesClientProps> = ({
  listings,
  currentUser
}) => {
  return (
    <Container>
      <Heading
        title="Favorites"
        subtitle="List of places you favorited!"
      />
      <div
        className="
          mt-10
          grid
          grid-cols-1
          sm:grid-cols-2
          md:grid-cols-3
          lg:grid-cols-4
          xl:grid-cols-5
          2xl:grid-cols-6
          gap-8
        "
      >
        {listings.map((listing: any) => (
          <ListingCard
            currentUser={currentUser}
            key={listing.id}
            data={listing}
          />
        ))}
      </div>
    </Container>
   );
}

export default FavoritesClient;
```

```
import { SafeListing, SafeUser } from "@/app/types";

import Heading from "@/app/components/Heading";
import Container from "@/app/components/Container";
import ListingCard from "@/app/components/listings/ListingCard";

interface FavoritesClientProps {
  listings: SafeListing[],
  currentUser?: SafeUser | null,
}

const FavoritesClient: React.FC<FavoritesClientProps> = ({
  listings,
  currentUser
}) => {
  return (
    <Container>
      <Heading
        title="Favorites"
        subtitle="List of places you favorited!"
      />
      <div
        className="
          mt-10
          grid
          grid-cols-1
          sm:grid-cols-2
          md:grid-cols-3
          lg:grid-cols-4
          xl:grid-cols-5
          2xl:grid-cols-6
          gap-8
        "
      >
        {listings.map((listing: any) => (
          <ListingCard
            currentUser={currentUser}
            key={listing.id}
            data={listing}
          />
        ))}
      </div>
    </Container>
  );
}

export default FavoritesClient;
```