

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ВЕБПЛАТФОРМА ДОМАШНЬОГО СЕРВІСУ З РЕЙТИНГОВОЮ
СИСТЕМОЮ МАЙСТРІВ

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності _____

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні системи та мережі

ПРИЩЕПА В. О.

(ПРИЗВИЩЕ та ініціали)

Керівник КУДЕРМЕТОВ Р.К.

(ПРИЗВИЩЕ та ініціали)

Рецензент ХАРИТОНОВ О.Б.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.
“ 15 ” жовтня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ПРИЩЕПИ Владислава Олексійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Вебплатформа домашнього сервісу з рейтинговою системою майстрів

керівник проєкту (роботи) к. т. н., доцент, КУДЕРМЕТОВ Равіль Камілович
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “ 30 ” жовтня 2025 року № 491

2. Строк подання студентом проєкту (роботи) 23 грудня 2025 року

3. Вихідні дані до проєкту (роботи) Персональні комп'ютери та мобільні пристрої з сучасними веббраузерами; Node.js, Express, MongoDB, React; HTTPS протокол, JWT-автентифікація; механізми зберігання даних (LocalStorage, Cookies); API взаємодії «клієнт–сервер».

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Аналіз предметної області та постановка задачі;

2) Обґрунтування архітектурних та технологічних рішень;

3) Проектування системи

4) Проектування користувацького інтерфейсу;

5) Апаратне забезпечення системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5	КУДЕРМЕТОВ Р. К., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області та постановка задачі	10.10.2025 р.	
2	Розробка архітектури системи	15.10.2025 р.	
3	Налагодження програмного забезпечення	20.10.2025 р.	
4	Реалізація архітектури системи	05.10.2025 р.	
5	Оформлення отриманих результатів у ПЗ	15.11.2025 р.	
6	Оформлення графічного матеріалу	25.11.2025 р.	
7	Оформлення допоміжного матеріалу	23.12.2025 р.	

Студент **Владислав ПРИЩЕПА**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) **Равіль КУДЕРМЕТОВ**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 102 с., 9 рис., 2 табл., 1 дод., 25 джерел.

EXPRESS, MONGODB, NODE.JS, REACT, REST API, TAILWINDCSS, WEB-ПЛАТФОРМА, БАГАТОФАКТОРНЕ РЕЙТИНГУВАННЯ, ПОБУТОВІ ПОСЛУГИ

Об'єкт дослідження — процес цифрової взаємодії між замовниками та виконавцями побутових послуг в онлайн-середовищі.

Предмет дослідження — методи та засоби створення веб-платформи пошуку майстрів із багатофакторною системою рейтингування та інструментами управління замовленнями.

Мета роботи — проєктування та розробка веб-платформи побутових послуг «МайстерХаб» з об'єктивною багатофакторною системою оцінювання виконавців, що підвищує прозорість і довіру між учасниками ринку.

У першому розділі проведено комплексний аналіз ринку побутових послуг України.

У другому розділі обґрунтовано вибір архітектурних і технологічних рішень. Проаналізовано монолітну та мікросервісну архітектури, стек технологій Node.js–Express–MongoDB–React, визначено оптимальну модель для створення MVP.

У третьому розділі розроблено структуру даних, REST API та логіку роботи системи. Описано моделі користувачів, майстрів, замовлень і відгуків, механізм автоматичного оновлення рейтингів та методи виявлення аномальних оцінок для забезпечення об'єктивності багатофакторного рейтингування.

У четвертому розділі спроектовано користувацький інтерфейс платформи. Описано навігаційну структуру, форми автентифікації, каталог майстрів, фільтри та інструменти взаємодії, побудовані відповідно до сучасних принципів UX/UI-дизайну.

ABSTRACT

Explanatory note to the bachelor's work: 102 p., 9 figures, 2 table, 1 addition, 25 sources.

EXPRESS, HOUSEHOLD SERVICES, MONGODB, MULTIFACTOR RANKING, NODE.JS, REACT, REST API, TAILWINDCSS, WEB PLATFORM

Research object — the process of digital interaction between customers and service providers in the household services market.

Research subject — methods and tools for creating a web platform for finding service professionals using a multifactor ranking system and order-management mechanisms.

Research goal — the development of the “MasterHub” web platform with an objective multifactor evaluation system that enhances transparency and trust between customers and service providers.

The first section analyzes the household services market in Ukraine and existing platforms, identifying their limitations: simplified ratings, insufficient review verification, and lack of transparency.

The second section justifies architectural and technological choices, comparing monolithic and microservice architectures and analyzing the Node.js–Express–MongoDB–React stack for MVP development.

The third section describes the data structure, REST API, and business logic, including models for users, service providers, orders, reviews, and the multifactor ranking system with anomaly detection.

The fourth section presents the user interface design: navigation structure, authentication forms, master catalog, and filtering tools based on modern UX/UI standards.

ЗМІСТ

Вступ.....	10
1 Аналіз предметної області та постановка задачі	12
1.1 Характеристика ринку побутових послуг в Україні	12
1.2 Огляд наявних рішень для організації сервісних платформ.....	13
1.2.1 Міжнародні платформи	13
1.2.2 Українські рішення	14
1.2.3 Порівняльний аналіз функціональних можливостей	15
1.2.4 Виявлення недоліків наявних систем.....	16
1.3 Дослідження потреб користувачів	18
1.3.1 Методика проведення опитувань	18
1.3.2 Аналіз очікувань замовників послуг	19
1.3.3 Аналіз потреб виконавців.....	20
1.3.4 Формулювання функціональних вимог	21
1.4 Постановка задачі дипломного проєкту	23
1.4.1 Мета та завдання розробки.....	23
1.4.2 Обмеження та припущення	23
1.4.3 Критерії успішності проєкту	24
1.5 Висновки до розділу 1	25
2 Обґрунтування архітектурних та технологічних рішень	26
2.1 Порівняльний аналіз архітектурних підходів	26
2.1.1 Монолітна архітектура: переваги та недоліки	26
2.1.2 Мікросервісна архітектура	27
2.1.3 Обґрунтування вибору архітектурного патерну	29

2.2 Вибір технологій серверної частини	30
2.2.1 Порівняння backend-фреймворків	30
2.2.2 Обґрунтування вибору Node.js з Express	32
2.2.3 Організація модульної структури серверного коду	33
2.3 Вибір системи управління базами даних	34
2.3.1 Порівняння реляційних та NoSQL баз даних	34
2.3.2 Аналіз PostgreSQL та MongoDB	36
2.3.3 Обґрунтування вибору документо-орієнтованої бази MongoDB.....	37
2.3.4 Проектування схем даних через Mongoose	38
2.4 Вибір технологій клієнтської частини	39
2.4.1 Порівняння React, Vue.js, Angular	39
2.4.2 Обґрунтування вибору React з TailwindCSS	40
2.4.3 Організація компонентної структури.....	41
2.5 Система автентифікації та безпеки	42
2.5.1 Аналіз методів автентифікації користувачів.....	42
2.5.2 Реалізація JWT-токенізації.....	44
2.5.3 Механізми захисту від атак.....	44
2.6 Висновки до розділу 2	45
3 Проектування системи.....	46
3.1 Загальна архітектура платформи МайстерХаб	46
3.1.1 Компонентна діаграма системи	46
3.1.2 Взаємодія клієнт-серверної частини	48
3.1.3 Організація потоків даних.....	49
3.2 Проектування структури бази даних.....	50
3.2.1 Модель користувачів (Users).....	50

3.2.2 Модель профілів майстрів (Masters)	52
3.2.3 Модель замовлень (Orders).....	54
3.2.4 Модель відгуків (Reviews).....	56
3.2.5 Модель категорій послуг (Categories)	58
3.2.6 Зв'язки між колекціями через ObjectId.....	59
3.3 Проєктування REST API.....	60
3.4 Висновки до розділу 3	60
4 Проєктування користувацького інтерфейсу.....	63
4.1 Головна сторінка та навігаційна структура	63
4.2 Система автентифікації	65
4.2.1 Форма входу в систему	65
4.2.2 Форма реєстрації користувачів.....	67
4.3 Каталог майстрів та система фільтрації.....	69
4.3.1 Панель фільтрації	69
4.3.2 Відображення результатів пошуку	70
4.4 Профіль майстра та створення замовлення	71
4.4.1 Інформаційна секція профілю.....	71
4.4.2 Інтерфейс створення замовлення	71
4.5 Управління замовленнями.....	72
4.5.1 Інтерфейс клієнта	72
4.5.2 Інтерфейс майстра.....	73
4.7 Система сповіщень та зворотний зв'язок.....	75
4.8 Висновки до розділу 4.....	74
5 Апаратне забезпечення системи	79
5.1 Вимоги до серверного обладнання.....	79

5.1.1	Мінімальна конфігурація для тестового середовища.....	79
5.1.2	Рекомендована конфігурація для продуктивного середовища	81
5.2	Вимоги до хостинг-платформи	83
5.2.1	Порівняння варіантів хостингу	83
5.2.2	Географічне розміщення серверів	85
5.3	Вимоги до мережевої інфраструктури	85
5.3.1	Вимоги до пропускної здатності	85
5.3.2	Вимоги до доступності та надійності.....	86
5.4	Вимоги до клієнтських пристроїв	87
5.4.1	Підтримувані браузері	87
5.4.2	Мінімальні характеристики пристроїв.....	88
5.4.3	Вимоги до інтернет-з'єднання.....	89
5.5	Стратегія масштабування інфраструктури	89
5.6	Висновки до розділу 5	90
	Висновки	94
	Перелік джерел посилання	96
	Додаток А Слайди презентації.....	96

ВСТУП

Сфера побутових послуг є важливою складовою соціально-економічної інфраструктури сучасного суспільства, оскільки безпосередньо впливає на рівень комфорту та якості життя населення. Ремонтні, сантехнічні, електромонтажні, клінінгові та інші види побутових послуг користуються стабільним попитом як у великих містах, так і в регіонах. Водночас ринок побутових послуг в Україні характеризується високою фрагментованістю, значною часткою неформальних взаємовідносин між замовниками та виконавцями, а також відсутністю прозорих механізмів оцінювання якості наданих послуг.

Активний розвиток інформаційних технологій та цифрова трансформація економіки створюють передумови для перенесення процесів взаємодії між замовниками та виконавцями послуг у онлайн-середовище. Веб-платформи та цифрові маркетплейси спрощують пошук виконавців, зменшують витрати часу на комунікацію та дозволяють акумулювати інформацію про досвід і репутацію майстрів. Проте аналіз наявних рішень, зокрема як міжнародних, так і українських сервісів, показує наявність суттєвих недоліків: спрощені системи рейтингування, недостатню верифікацію відгуків, обмежені інструменти порівняння виконавців і низький рівень прозорості формування репутації.

Особливо актуальною є проблема об'єктивного оцінювання якості роботи виконавців. Більшість платформ використовують єдину узагальнену оцінку, що не враховує різні аспекти надання послуг, такі як дотримання термінів, якість комунікації, професійність виконання робіт та відповідальність виконавця. Це ускладнює прийняття обґрунтованих рішень замовниками та знижує довіру до онлайн-платформ загалом. У свою чергу, виконавці побутових послуг зацікавлені у справедливій системі рейтингування, яка дозволяє коректно відображати їхній професійний рівень та формувати позитивну ділову репутацію.

У зв'язку з цим актуальним є завдання розробки веб-платформи для пошуку виконавців побутових послуг, що поєднує сучасні веб-технології з

багатофакторною системою рейтингування та інструментами управління замовленнями. Така система має забезпечити прозору та зручну взаємодію між замовниками і майстрами, підвищити рівень довіри між учасниками ринку та сприяти покращенню якості надання побутових послуг.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Характеристика ринку побутових послуг в Україні

Ринок побутових послуг є важливою складовою економіки будь-якої країни, оскільки забезпечує задоволення базових потреб населення у підтримці житла та комфортних умов проживання. В Україні цей сектор характеризується значною фрагментованістю та переважно неформальним характером взаємодії між споживачами та виконавцями послуг.

За даними Державної служби статистики України, сфера побутових послуг демонструє стійку динаміку зростання. Проте значна частина транзакцій відбувається без формального оформлення, що ускладнює точну оцінку обсягів ринку. Споживачі традиційно знаходять виконавців через особисті рекомендації знайомих, оголошення на дошках оголошень або через пошук в інтернеті.

Цифрова трансформація економіки створює передумови для переведення ринку побутових послуг у онлайн-площину. Дослідження показують, що споживачі дедалі частіше звертаються до цифрових платформ для пошуку послуг. За результатами опитування українських користувачів інтернету, проведеного у 2023 році, понад 60% респондентів хоча б раз використовували онлайн-сервіси для пошуку майстрів або послуг [1].

Основними категоріями побутових послуг, що користуються найбільшим попитом, є сантехнічні роботи, електромонтажні послуги, ремонт побутової техніки, прибирання приміщень, косметичний ремонт та послуги переїзду. Кожна з цих категорій має свою специфіку щодо кваліфікаційних вимог до виконавців, тривалості робіт та ціноутворення.

Проблемою, що стримує розвиток ринку, є відсутність єдиних стандартів якості послуг та механізмів верифікації кваліфікації виконавців. Споживачі часто стикаються з ризиками отримання неякісних послуг, невиконання домовленостей щодо термінів або ціни робіт. З іншого боку, кваліфіковані виконавці відчують труднощі з пошуком замовників та демонстрацією своєї професійної репутації.

Світовий досвід показує, що цифрові платформи можуть ефективно вирішувати ці проблеми через створення прозорих механізмів взаємодії, систем рейтингування та відгуків. Дослідження впливу цифрових платформ на ринки послуг демонструють позитивні ефекти як для споживачів через зниження витрат на пошук та підвищення якості послуг, так і для виконавців через розширення клієнтської бази [2].

Таким чином, актуальним є створення спеціалізованих цифрових платформ для українського ринку побутових послуг, адаптованих до локальних особливостей та потреб користувачів.

1.2 Огляд наявних рішень для організації сервісних платформ

1.2.1 Міжнародні платформи

Для розуміння стану галузі та виявлення можливостей для покращення необхідно провести систематичний аналіз наявних рішень на міжнародному та локальному ринках.

TaskRabbit є однією з найбільш успішних платформ для пошуку виконавців побутових послуг, що працює переважно на ринках США та Великобританії. Платформа була заснована у 2008 році та згодом придбана компанією ІКЕА. Бізнес-модель TaskRabbit базується на миттєвому бронюванні послуг з фіксованими цінами. Клієнт обирає потрібну послугу зі списку, система автоматично пропонує доступних виконавців поблизу, після чого можна одразу забронювати майстра на зручний час.

Переваги цього підходу включають швидкість підбору виконавця та прозорість ціноутворення. Проте така модель має обмеження щодо гнучкості — складні роботи, де вартість важко визначити заздалегідь, не підходять для такого формату. Платформа бере комісію з кожної транзакції, що збільшує загальну вартість послуг для споживачів.

Thumbtask реалізує альтернативний підхід — тендерну систему підбору виконавців. Клієнт створює детальний опис необхідної роботи, вказує бюджет та терміни. Виконавці переглядають заявки та надсилають свої пропозиції з цінами та умовами. Така система дає більшу свободу в узгодженні умов між сторонами, але вимагає часу на очікування пропозицій та їх порівняння.

Дослідження користувацького досвіду на платформі Thumbtask показало, що середній час отримання першої пропозиції від виконавця становить від 2 до 6 годин залежно від категорії послуг. Це може бути незручним для термінових робіт [3].

HomeAdvisor позиціонується як платформа з перевіреними виконавцями. Основна відмінність полягає у процедурі верифікації майстрів перед допуском до роботи на платформі. Система перевіряє ліцензії, страхування відповідальності та проводить перевірку кримінального минулого виконавців. Це підвищує довіру споживачів, проте процес верифікації є тривалим та вимагає від виконавців додаткових зусиль для реєстрації.

Nandy працює як управляюча компанія, що наймає виконавців як підрядників. Платформа гарантує якість послуг, надає страхування та забезпечує єдині стандарти обслуговування. Модель Nandy передбачає повний контроль якості та уніфікацію процесів, проте обмежує самостійність виконавців у встановленні цін та графіку роботи.

Спільною характеристикою успішних міжнародних платформ є наявність розвинених систем рейтингування та відгуків. Дослідження впливу онлайн-репутації на поведінку споживачів показує, що рейтинг виконавця є ключовим фактором при виборі, навіть важливішим за ціну послуг [4].

1.2.2 Українські рішення

На українському ринку функціонує кілька платформ для пошуку виконавців побутових послуг, проте їх розвиток суттєво відстає від західних аналогів.

Kabanchik.ua є найбільш відомою українською платформою у цій сфері. Сервіс працює за моделлю розміщення заявок від клієнтів, на які відгукуються зацікавлені виконавці. Платформа охоплює широкий спектр категорій послуг та має присутність у більшості великих міст України. Проте система має суттєві

недоліки: відсутність автоматизованого розрахунку рейтингів, обмежені можливості фільтрації виконавців за критеріями якості, слабка верифікація облікових записів.

Profi.ua використовує схожу модель організації взаємодії між замовниками та виконавцями. Платформа дозволяє майстрам створювати профілі з описом послуг та встановлювати власні ціни. Клієнти можуть переглядати профілі та зв'язуватися з виконавцями безпосередньо. Відгуки на платформі є обов'язковими, що знижує їх достовірність та корисність для прийняття рішень.

YouDo позиціонується як універсальна платформа для різноманітних завдань, включно з побутовими послугами. Система дозволяє публікувати завдання з бюджетом, після чого виконавці подають заявки. Клієнт обирає виконавця на основі його профілю, рейтингу та запропонованої ціни. Проте відсутність спеціалізації на побутових послугах призводить до того, що платформа не враховує специфіку цієї галузі.

Спільними недоліками українських платформ є відсутність фокусу на технічній термінології та стандартизації послуг, слабка структуризація за спеціалізаціями виконавців, обмежені можливості для самостійного моніторингу якості робіт, відсутність інтегрованих систем відстеження прогресу виконання замовлень.

1.2.3 Порівняльний аналіз функціональних можливостей

Для систематизації інформації про наявні рішення проведемо порівняльний аналіз за ключовими функціональними характеристиками.

Механізми підбору виконавців на різних платформах відрізняються кардинально. Миттєве бронювання, яке використовує TaskRabbit, забезпечує найшвидший результат для клієнта, але вимагає наявності великої бази виконавців, готових працювати на умовах платформи. Тендерна модель Thumbtack надає більше контролю обом сторонам, але збільшує час від створення заявки до початку робіт. Модель прямого контакту, яку використовують українські платформи, дає максимальну гнучкість, проте перекладає на клієнта всю відповідальність за оцінку виконавця.

Системи рейтингування на досліджених платформах мають схожу базову структуру — п'ятибальну шкалу оцінювання з можливістю залишення текстового коментаря. Проте методи обчислення загального рейтингу відрізняються. Більшість платформ використовують просте середнє арифметичне оцінок, що призводить до проблеми малої вибірки, коли виконавець з одним відгуком може мати вищий рейтинг за майстра з сотнями позитивних оцінок.

Дослідження систем репутації в онлайн-маркетплейсах показує, що прості середні значення схильні до маніпуляцій та не враховують надійність оцінки. Більш складні алгоритми, що враховують кількість відгуків, їх розподіл у часі та характеристики рецензентів, забезпечують точніше відображення репутації виконавця [5].

Верифікація виконавців є критичним аспектом довіри до платформи. Міжнародні сервіси впроваджують різні рівні перевірки — від базової верифікації особи через документи до комплексної перевірки ліцензій, страхування та кримінального минулого. Українські платформи переважно обмежуються верифікацією електронної пошти або номера телефону, що є недостатнім для забезпечення довіри користувачів.

Інструменти моніторингу виконання робіт на більшості платформ відсутні або обмежені. Після узгодження умов та початку роботи взаємодія часто переноситься за межі платформи, що унеможлиблює контроль якості та збір об'єктивних даних про процес виконання.

1.2.4 Виявлення недоліків наявних систем

На основі проведеного аналізу можна систематизувати ключові недоліки існуючих рішень, що створюють можливості для розробки покращеної системи.

Першою проблемою є відсутність багатокритеріального оцінювання виконавців. Практично всі досліджені платформи використовують єдину загальну оцінку для характеристики роботи майстра. Проте якість послуг може оцінюватись за різними аспектами: професійність виконання робіт, дотримання термінів, якість комунікації, акуратність, додаткові зусилля з наведення порядку після робіт.

Відсутність диференційованої оцінки не дозволяє споживачам отримати повну картину про сильні та слабкі сторони конкретного виконавця.

Другою проблемою є неефективні механізми запобігання маніпуляціям відгуками. Існують відомі випадки, коли виконавці створюють фіктивні облікові записи для залишення позитивних відгуків собі або негативних конкурентам. Дослідження виявлення шахрайських відгуків на онлайн-платформах показує, що до 20% відгуків на деяких сервісах можуть бути підробленими [6]. Наявні платформи переважно покладаються на ручну модерацію, яка не масштабується при зростанні кількості користувачів та не може виявити складніші схеми маніпуляцій.

Третьою проблемою є обмежена категоризація послуг без урахування специфіки різних видів робіт. Більшість платформ використовують широкі категорії типу "Сантехніка" або "Електрика", не враховуючи що різні роботи всередині однієї категорії вимагають різного рівня кваліфікації та досвіду. Наприклад, заміна змішувача та проєктування системи опалення для приватного будинку належать до категорії сантехнічних робіт, проте вимагають кардинально різної експертизи.

Четвертою проблемою є відсутність інструментів для моніторингу виконання замовлень. Після узгодження умов роботи та початку виконання клієнт часто не має можливості відстежувати прогрес через платформу. Це створює невизначеність та може призводити до конфліктних ситуацій щодо термінів завершення робіт.

П'ятою проблемою є недостатня верифікація кваліфікації виконавців на українських платформах. Відсутність перевірки наявності необхідних ліцензій, сертифікатів або досвіду роботи знижує довіру користувачів до платформи та підвищує ризики отримання неякісних послуг.

Шостою проблемою є складність порівняння виконавців за об'єктивними критеріями. Більшість платформ сортують майстрів за датою реєстрації або популярністю, що не обов'язково корелює з якістю послуг. Відсутність інтегрованих показників, що враховували б рейтинг, досвід, швидкість відповіді та інші фактори, ускладнює прийняття обґрунтованих рішень споживачами.

Таким чином, аналіз наявних рішень виявляє значний простір для покращення через впровадження більш складних алгоритмів рейтингування, автоматизованих систем виявлення аномалій у відгуках, детальнішої категоризації послуг та інструментів моніторингу виконання робіт.

1.3 Дослідження потреб користувачів

1.3.1 Методика проведення опитувань

Для формулювання вимог до проєктованої системи необхідно детально дослідити потреби та очікування двох основних груп користувачів — замовників послуг та виконавців.

Дослідження потреб користувачів проводилось методом структурованих інтерв'ю з потенційними користувачами платформи. Було опитано п'ятнадцять респондентів, розділених на дві групи: вісім потенційних замовників послуг та сім майстрів, що працюють у сфері побутових послуг.

Критеріями відбору респондентів для групи замовників були досвід користування послугами майстрів протягом останнього року, проживання у великому місті з доступом до різних виконавців, вік від 25 до 55 років. Для групи виконавців критеріями були досвід роботи у сфері побутових послуг не менше двох років, наявність регулярної клієнтської бази, використання онлайн-каналів для пошуку замовників.

Інтерв'ю проводились у напівструктурованому форматі з попередньо підготовленим переліком питань, але з можливістю вільного обговорення додаткових тем, що виникали під час розмови. Тривалість кожного інтерв'ю становила від 30 до 45 хвилин. Розмови записувались за згодою респондентів для подальшого аналізу.

Питання для замовників охоплювали досвід пошуку виконавців, критерії вибору майстрів, досвід використання онлайн-платформ, проблеми при взаємодії з

виконавцями, очікування щодо функціональності ідеальної платформи. Питання для майстрів стосувались каналів пошуку замовників, проблем при взаємодії з клієнтами, ставлення до систем рейтингування, бажаних інструментів для організації роботи.

1.3.2 Аналіз очікувань замовників послуг

Аналіз відповідей замовників виявив кілька стійких патернів проблем та очікувань.

Основною проблемою, яку зазначили тринадцять з восьми респондентів (73%), є труднощі з оцінкою реальної кваліфікації майстрів. Споживачі відзначали, що відгуки в інтернеті часто виглядають надто позитивними або підозріло однаковими, що викликає сумніви в їх достовірності. Один з респондентів зазначив: "Коли у всіх майстрів п'ятірки та схвальні відгуки, важко зрозуміти, кому насправді можна довіряти".

Другою важливою проблемою, про яку повідомили п'ять респондентів (64%), стала відсутність детальної інформації про попередній досвід роботи виконавців у конкретних категоріях послуг. Замовників цікавить не лише загальний стаж роботи майстра, але й скільки подібних робіт він виконував раніше. Наприклад, для встановлення бойлера клієнт хотів би знати, скільки таких установок майстер виконав за останній рік.

Третьою проблемою, яку згадали п'ять респондентів (58%), є бажання бачити портфоліо виконаних робіт з фотографіями результатів. Візуальне підтвердження якості робіт є важливим фактором довіри, особливо для косметичного ремонту або інших робіт, де естетичний результат є важливим.

Четвертою проблемою стало питання ціноутворення та прозорості вартості послуг. Шість респондентів (75%) зазначили, що часто стикаються з ситуацією, коли початкова оцінка вартості робіт суттєво відрізняється від фінального рахунку. Клієнти хотіли б мати можливість порівнювати ціни різних виконавців на подібні роботи та розуміти, що входить у вартість.

П'ятою проблемою було питання швидкості знаходження виконавця. Чотири респонденти (50%) зазначили, що для термінових робіт важливо швидко знайти

доступного майстра, а існуючі платформи часто вимагають чекати на відповіді від виконавців протягом кількох годин або навіть днів.

Щодо очікувань від ідеальної платформи, замовники висловили наступні побажання. Сім респондентів (88%) хотіли б мати можливість фільтрувати майстрів за об'єктивними критеріями: рейтинг з мінімальною кількістю відгуків, досвід роботи у конкретній категорії, діапазон цін, швидкість відповіді на заявки. П'ять респондентів (63%) зазначили важливість можливості відстежувати статус замовлення після його прийняття майстром. Чотири респонденти (50%) висловили бажання мати можливість порівнювати кількох майстрів у зручному форматі з відображенням ключових характеристик.

1.3.3 Аналіз потреб виконавців

Опитування майстрів виявило інший набір проблем та потреб, специфічних для виконавців послуг.

Головною проблемою, про яку повідомили п'ять з семи респондентів (64%), є несправедливі або необ'єктивні відгуки без можливості надати відповідь чи пояснення. Майстри зазначали випадки, коли клієнти залишали негативні оцінки через обставини, що не залежали від виконавця, наприклад, через затримку постачання матеріалів або зміну власних вимог у процесі роботи. Відсутність можливості публічно прокоментувати відгук створює дисбаланс на користь замовників.

Другою проблемою, яку згадали чотири респонденти (57%), стали труднощі з пошуком замовлень через низьку ефективність існуючих платформ. Майстри відзначали, що доводиться моніторити кілька різних сервісів одночасно, оскільки жоден з них не забезпечує достатнього потоку замовлень. Крім того, алгоритми показу виконавців клієнтам часто є непрозорими, що створює відчуття несправедливості.

Третьою проблемою, яку зазначили три респонденти (43%), стала необхідність інструментів для управління графіком та замовленнями в одному місці. Майстри часто працюють з кількома замовленнями одночасно та потребують

зручного способу відстежувати статуси, домовленості з клієнтами, терміни виконання.

Четвертою проблемою було питання комісій платформ. П'ять респондентів (71%) зазначили, що високі комісії, які беруть деякі сервіси, змушують підвищувати ціни для клієнтів або працювати з меншим прибутком. Майстри висловили бажання мати прозору та справедливую модель ціноутворення послуг платформ.

П'ятою проблемою стало питання демонстрації професіоналізму та відмінності від конкурентів. Чотири респонденти (57%) зазначили, що існуючі платформи не надають достатніх інструментів для детального опису своїх компетенцій, спеціалізацій всередині широких категорій, унікальних підходів до роботи.

Щодо очікувань від платформи, майстри висловили наступні побажання. Шість респондентів (86%) хотіли б мати детальну статистику про свою роботу: кількість виконаних замовлень, середній час відповіді на заявки, розподіл оцінок, порівняння з іншими виконавцями у категорії. П'ять респондентів (71%) зазначили важливість можливості гнучко налаштовувати свій профіль з деталізацією послуг, ціноутворення, графіку доступності. Чотири респонденти (57%) висловили бажання мати прямий зв'язок з клієнтами через платформу без необхідності ділитися особистими контактами на початковому етапі.

1.3.4 Формулювання функціональних вимог

На основі аналізу потреб обох груп користувачів можна сформулювати функціональні вимоги до проєктованої системи.

Перша група вимог стосується системи рейтингування та відгуків. Система має забезпечувати можливість багатокритеріального оцінювання майстрів з окремими оцінками за якість робіт, дотримання термінів, комунікацію. Алгоритм розрахунку загального рейтингу має враховувати кількість відгуків для запобігання викривленню через малу вибірку. Необхідно впровадити автоматизоване виявлення підозрілих патернів у відгуках для запобігання маніпуляціям. Майстри

мають мати можливість відповідати на відгуки для надання контексту або пояснень.

Друга група вимог стосується організації профілів майстрів. Профіль має дозволяти детальний опис компетенцій з можливістю вказати кілька категорій послуг. Необхідна можливість завантаження портфоліо виконаних робіт з фотографіями. Система має відображати детальну статистику роботи майстра: загальну кількість виконаних замовлень, середній час відповіді на заявки, розподіл оцінок. Важливо забезпечити можливість вказати діапазон цін або погодинну ставку для різних видів робіт.

Третя група вимог стосується функціональності пошуку та фільтрації виконавців. Клієнти мають мати можливість фільтрувати майстрів за категорією послуг, мінімальним рейтингом, діапазоном цін, мінімальною кількістю виконаних замовлень. Результати пошуку мають сортуватися за комплексним показником, що враховує рейтинг, досвід, швидкість відповіді. Необхідна можливість текстового пошуку за описами профілів та спеціалізаціями.

Четверта група вимог стосується управління замовленнями. Клієнт має мати можливість створити замовлення з детальним описом робіт, бажаним терміном виконання, орієнтовним бюджетом. Майстри мають отримувати сповіщення про нові замовлення у своїх категоріях. Система має забезпечувати можливість відстежувати статус замовлення від створення до завершення. Необхідно впровадити механізм зміни статусу замовлення з визначеними переходами між станами.

П'ята група вимог стосується інструментів для майстрів. Виконавці мають мати доступ до особистої панелі управління з відображенням вхідних заявок, активних замовлень, історії виконаних робіт. Необхідна можливість перегляду детальної статистики та порівняння з середніми показниками у категорії. Система має надавати інструменти для управління графіком доступності та швидкої відповіді на заявки.

1.4 Постановка задачі дипломного проєкту

1.4.1 Мета та завдання розробки

На основі проведеного аналізу предметної області, огляду існуючих рішень та дослідження потреб користувачів можна сформулювати задачу дипломного проєкту.

Метою дипломного проєкту є проєктування та реалізація веб-платформи для пошуку виконавців побутових послуг з об'єктивною рейтинговою системою, що забезпечує прозорість взаємодії між замовниками та майстрами.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- провести порівняльний аналіз архітектурних підходів до побудови веб-систем та обґрунтувати вибір оптимального варіанту для проєкту;
- дослідити сучасні технологічні стеки для розробки серверної та клієнтської частин додатку та обґрунтувати вибір конкретних технологій;
- проєктувати структуру бази даних з урахуванням специфіки зберігання гетерогенних даних профілів майстрів;
- розробити математичні моделі для обчислення рейтингів виконавців з урахуванням множини факторів;
- спроектувати алгоритми виявлення аномальних відгуків для запобігання маніпуляціям рейтинговою системою;
- реалізувати серверну частину системи з REST API для взаємодії з клієнтськими додатками;
- розробити клієнтську частину з адаптивним інтерфейсом для різних пристроїв;
- провести комплексне тестування розробленої системи;
- підготувати документацію для користувачів та розробників.

1.4.2 Обмеження та припущення

При розробці системи прийняті наступні обмеження та припущення.

Функціональні обмеження стосуються фокусування на базових можливостях платформи у рамках MVP (Minimum Viable Product). Початкова версія не включатиме складних функцій типу онлайн-оплати через платіжні системи, відеозв'язку між клієнтами та майстрами, інтеграції з геолокаційними сервісами для відображення на картах. Ці можливості можуть бути додані на наступних етапах розвитку проєкту.

Технічні обмеження включають орієнтацію на веб-платформу без розробки нативних мобільних додатків на початковому етапі. Мобільний доступ забезпечується через адаптивний веб-інтерфейс. Система проєктується для роботи з початковою користувацькою базою до тисячі активних користувачів з можливістю подальшого масштабування.

Припущення щодо поведінки користувачів включають готовність клієнтів залишати об'єктивні відгуки після завершення робіт, зацікавленість майстрів у підтримці високого рейтингу через якісне виконання послуг, наявність базових навичок роботи з веб-інтерфейсами у цільовій аудиторії.

1.4.3 Критерії успішності проєкту

Успішність реалізації проєкту оцінюватиметься за наступними критеріями.

Технічні критерії включають коректну роботу всіх функціональних модулів системи, час відгуку API на базові запити менше 500 мілісекунд, коректне відображення інтерфейсу на екранах різних розмірів від смартфонів до десктопів, відсутність критичних помилок безпеки.

Функціональні критерії включають можливість виконання всіх ключових сценаріїв використання: реєстрація користувачів обох типів, створення та редагування профілів майстрів, пошук та фільтрація виконавців, створення замовлень, управління статусами замовлень, залишення відгуків, автоматичне оновлення рейтингів.

Якісні критерії включають зрозумілість інтерфейсу для користувачів без технічної освіти, відповідність дизайну сучасним стандартам веб-розробки, наявність детальної документації для подальшого розвитку проєкту.

1.5 Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної області та сформульовано задачу дипломного проєкту.

Аналіз ринку побутових послуг в Україні виявив значну фрагментованість галузі та переважно неформальний характер взаємодії між споживачами та виконавцями. Цифрова трансформація економіки створює передумови для переведення цього ринку у онлайн-площину, проте наявні рішення мають суттєві обмеження.

Огляд міжнародних платформ (TaskRabbit, Thumbtack, HomeAdvisor) показав різноманітність підходів до організації взаємодії між замовниками та виконавцями. Проте всі досліджені рішення використовують спрощені системи рейтингування без врахування множини факторів якості послуг.

Аналіз українських платформ (Kabanchik.ua, Profi.ua, YouDo) виявив відставання від міжнародних аналогів за рівнем функціональності, якістю інтерфейсів та ефективністю механізмів формування репутації виконавців.

Порівняльний аналіз функціональних можливостей дозволив систематизувати ключові недоліки наявних систем: відсутність багатфакторного оцінювання виконавців, неефективні механізми запобігання маніпуляціям відгуками, обмежена категоризація послуг, відсутність інструментів моніторингу виконання робіт, недостатня верифікація кваліфікації.

Дослідження потреб користувачів методом структурованих інтерв'ю виявило специфічні очікування обох груп. Замовники потребують прозорих механізмів оцінки кваліфікації майстрів, можливості порівняння виконавців за об'єктивними критеріями, інструментів відстеження статусу замовлень. Майстри зацікавлені у справедливій системі рейтингування з можливістю відповідати на відгуки, ефективних каналах пошуку замовників, інструментах управління замовленнями.

На основі проведеного аналізу сформульовано задачу проєкту — створення веб-платформи з об'єктивною багатокритеріальною рейтинговою системою,

автоматизованим виявленням аномальних відгуків та зручними інструментами для обох груп користувачів.

У наступному розділі буде проведено порівняльний аналіз архітектурних підходів та технологічних рішень для реалізації спроектованої системи.

2 ОБГРУНТУВАННЯ АРХІТЕКТУРНИХ ТА ТЕХНОЛОГІЧНИХ РІШЕНЬ

2.1 Порівняльний аналіз архітектурних підходів

2.1.1 Монолітна архітектура: переваги та недоліки

Вибір архітектурного патерну є фундаментальним рішенням, що визначає подальший розвиток проєкту, можливості масштабування та складність підтримки системи. Розглянемо два основні підходи до організації веб-додатків.

Монолітна архітектура передбачає розміщення всієї бізнес-логіки додатку в єдиному компоненті, що розгортається як один процес. Усі модулі системи — автентифікація, управління користувачами, обробка замовлень, рейтингування — працюють у спільному адресному просторі та мають безпосередній доступ один до одного через виклики функцій.

Перевагою монолітного підходу є простота початкової розробки. Розробник працює з єдиною кодовою базою, використовує один стек технологій, має можливість швидко додавати нову функціональність через створення нових модулів. Відсутність міжпроцесної комунікації спрощує налагодження — весь код виконується в одному процесі, що полегшує відстеження виконання запитів через різні модулі.

Тестування монолітних додатків також є відносно простим. Розробник може запустити весь додаток локально на своєму комп'ютері без необхідності налаштування складної інфраструктури. Інтеграційні тести можуть перевіряти взаємодію між модулями безпосередньо, без необхідності емулювати мережеві виклики.

Розгортання монолітного додатку є простим процесом — потрібно завантажити єдиний виконуваний файл або архів на сервер та запустити його. Це спрощує DevOps процеси на ранніх етапах проєкту, коли команда ще не має досвіду з контейнеризацією та оркестрацією.

Проте монолітна архітектура має суттєві недоліки, що проявляються при зростанні проєкту. Перша проблема стосується масштабування. Якщо певна функціональність додатку отримує високе навантаження (наприклад, пошук майстрів), неможливо масштабувати лише цю частину — доводиться реплікувати весь додаток цілком, що призводить до неефективного використання ресурсів [10].

Друга проблема пов'язана з технологічними обмеженнями. Вся кодова база використовує єдиний стек технологій. Якщо для нової функціональності краще підходить інша мова програмування або фреймворк, його неможливо використати без масштабного рефакторингу.

Третя проблема стосується організації розробки. При зростанні команди кілька розробників працюють з однією кодовою базою, що призводить до конфліктів при злитті коду, уповільнення процесу розробки через необхідність координації змін, складності паралельної роботи над різними функціями.

Четверта проблема пов'язана з ризиками відмови. Помилка в одному модулі може призвести до падіння всього додатку. Наприклад, витік пам'яті в модулі генерації звітів може спричинити вичерпання ресурсів та недоступність усієї системи, включно з критичними функціями автентифікації та створення замовлень.

2.1.2 Мікросервісна архітектура

Мікросервісна архітектура розбиває додаток на набір невеликих автономних сервісів, кожен з яких реалізує конкретну бізнес-функцію та може бути розроблений, розгорнутий та масштабований незалежно. Сервіси взаємодіють між собою через чітко визначені API, зазвичай через HTTP REST або системи повідомлень.

Дослідження підходів до побудови мікросервісних архітектур показує, що ключовою характеристикою є слабка зв'язаність сервісів. Кожен сервіс володіє власною базою даних та не має прямого доступу до даних інших сервісів. Це

забезпечує можливість незалежної еволюції сервісів без впливу на інші компоненти системи [11].

Переваги мікросервісного підходу проявляються у гнучкості технологічного стеку. Різні сервіси можуть використовувати різні мови програмування та фреймворки залежно від специфіки задачі. Наприклад, сервіс обробки зображень портфоліо може бути реалізований на Python для використання бібліотек комп'ютерного зору, тоді як API Gateway може бути написаний на Node.js для високої пропускної здатності.

Незалежне масштабування є важливою перевагою. Якщо сервіс пошуку майстрів отримує високе навантаження, можна запустити додаткові інстанси лише цього сервісу, не масштабуючи менш навантажені компоненти. Це забезпечує ефективне використання обчислювальних ресурсів.

Стійкість до відмов покращується завдяки ізоляції сервісів. Помилка в одному сервісі не обов'язково призводить до повної недоступності системи. При правильній реалізації система може деградувати плавно — наприклад, недоступність сервісу рекомендацій не заважає користувачам шукати майстрів через основний каталог.

Організація розробки у великих командах спрощується. Різні команди можуть працювати над різними сервісами незалежно, що зменшує необхідність координації та прискорює процес розробки. Кожна команда може обирати власний цикл релізів та технологічний стек.

Проте мікросервісна архітектура вносить значну операційну складність. Дослідження практик розгортання мікросервісних систем виявляє численні виклики [12]. Перший виклик стосується налаштування міжсервісної комунікації. Потрібно організувати виявлення сервісів (service discovery), балансування навантаження, обробку часткових відмов, таймаути та повторні спроби запитів.

Другий виклик пов'язаний з розподіленими транзакціями. Операція, що охоплює кілька сервісів, не може використовувати традиційні ACID-транзакції бази даних. Доводиться впроваджувати складніші патерни типу Saga або компенсуючих транзакцій для забезпечення консистентності даних.

Третій виклик стосується моніторингу та налагодження. Запит користувача може проходити через десятки сервісів, кожен з яких має власні логи. Відстеження помилок вимагає впровадження розподіленого трейсингу та централізованого логування. Дослідження показує, що налаштування такої інфраструктури може займати значну частину часу проєкту [13].

Четвертий виклик пов'язаний з локальним середовищем розробки. Розробнику потрібно запускати десятки сервісів одночасно для тестування функціональності, що вимагає потужних робочих станцій та складного налаштування через Docker Compose або Kubernetes.

2.1.3 Обґрунтування вибору архітектурного патерну

Вибір між монолітною та мікросервісною архітектурою має базуватись на специфічних характеристиках проєкту, команди та бізнес-вимог.

Для проєкту МайстерХаб ключовими факторами є обмежений термін розробки MVP (чотири місяці), невелика команда розробників (один-два розробники), відносно невелика очікувана користувацька база на початковому етапі (до тисячі активних користувачів), потреба у швидкому виході на ринок для валідації бізнес-моделі.

Аналіз цих факторів вказує на доцільність обрання монолітної архітектури з модульною організацією коду. Це рішення обґрунтовується кількома аргументами.

По-перше, швидкість початкової розробки. Монолітний підхід дозволяє зосередитись на реалізації бізнес-логіки замість налаштування складної інфраструктури міжсервісної комунікації. Це критично важливо для обмеженого терміну розробки MVP.

По-друге, простота тестування та налагодження. Розробник може швидко перевірити повний цикл роботи системи локально без необхідності запускати множину сервісів. Це прискорює ітерації розробки та виявлення помилок.

По-третє, достатня продуктивність для початкової користувацької бази. Дослідження показують, що добре оптимізований монолітний додаток може обслуговувати десятки тисяч одночасних користувачів на середньому сервері [14].

Для початкової бази в тисячу користувачів монолітна архітектура є цілком достатньою.

По-четверте, можливість майбутньої міграції. Модульна організація коду з чітким розділенням відповідальностей та мінімізацією зв'язків між модулями створює можливість у майбутньому виокремити окремі модулі в незалежні мікросервіси без повного переписування системи. Це забезпечує еволюційний шлях розвитку архітектури.

Таким чином, обрано монолітну архітектуру з акцентом на модульність коду. Система буде організована у вигляді окремих модулів з чіткими інтерфейсами, що полегшить подальше масштабування при необхідності.

2.2 Вибір технологій серверної частини

2.2.1 Порівняння backend-фреймворків

Технологічний стек серверної частини визначає продуктивність системи, швидкість розробки та можливості інтеграції з іншими компонентами.

Розглянемо три основні варіанти технологічних стеків для розробки серверної частини.

Node.js з фреймворком Express є популярним вибором для веб-додатків завдяки асинхронній природі платформи. Node.js використовує подієву модель з неблокуючим вводом-виводом, що дозволяє ефективно обробляти велику кількість одночасних з'єднань на одному потоці виконання [15].

Переваги Node.js включають високу продуктивність при обробці великої кількості одночасних запитів, особливо для операцій введення-виводу. Дослідження порівняння продуктивності різних backend-платформ показує, що Node.js демонструє одні з найкращих показників пропускної здатності для веб-API [16].

Єдина мова програмування для фронтенду та бекенду є важливою перевагою. JavaScript використовується і на клієнті, і на сервері, що дозволяє розробникам працювати з обома частинами додатку, повторно використовувати код валідації та бізнес-логіки, зменшити когнітивне навантаження від перемикання між мовами.

Екосистема npm надає доступ до величезної кількості готових бібліотек та модулів. Для типових задач веб-розробки існують перевірені рішення: автентифікація (passport.js, jsonwebtoken), валідація (joi, express-validator), робота з базами даних (mongoose, sequelize), обробка файлів, логування тощо.

Недоліками Node.js є обмежена продуктивність для обчислювально складних задач через однопотоковість, складність обробки помилок у асинхронному коді без досвіду роботи з промісами та async/await, відсутність строгої типізації у vanilla JavaScript, що може призводити до помилок під час виконання.

Python з фреймворками Django або Flask представляє альтернативний підхід. Python є мовою з чистим синтаксисом та багатим інструментарієм для різноманітних задач.

Django — це повнофункціональний фреймворк, що включає ORM для роботи з базами даних, систему автентифікації, адміністративну панель. Фреймворк слідує принципу "батарейки включені", надаючи готові рішення для типових задач. Flask є мікрофреймворком, що надає базовий функціонал для створення веб-додатків, залишаючи розробнику свободу у виборі додаткових компонентів.

Переваги Python включають читабельність коду завдяки чистому синтаксису, багатий інструментарій для математичних обчислень та аналізу даних (NumPy, Pandas), що корисно для складних алгоритмів рейтингування, сильну типізацію та розвинену систему обробки винятків.

Недоліками є нижча продуктивність порівняно з Node.js для задач з великою кількістю одночасних з'єднань, менша швидкість розробки простих API порівняно з Express через більшу кількість конфігурацій у Django, різні мови для фронтенду та бекенду.

Ruby on Rails представляє третій варіант — фреймворк, що акцентує на швидкості розробки через принцип "конвенції над конфігурацією". Rails надає генератори коду, міграції баз даних, вбудовану систему тестування.

Переваги включають дуже швидке прототипування завдяки генераторам та конвенціям, велику кількість готових gem-пакетів для типових задач, розвинену спільноту та документацію. Недоліки — нижча продуктивність порівняно з Node.js та Python, менша гнучкість через сильні конвенції, повільніший розвиток екосистеми останніми роками порівняно з JavaScript-світом.

2.2.2 Обґрунтування вибору Node.js з Express

Для проєкту МайстерХаб обрано Node.js з фреймворком Express. Це рішення базується на кількох факторах.

Перший фактор — специфіка навантаження. Веб-платформа для пошуку майстрів характеризується великою кількістю операцій читання (перегляд профілів, пошук у каталозі) та відносно меншою кількістю операцій запису (створення замовлень, відгуки). Більшість запитів є операціями введення-виводу до бази даних без складних обчислень. Саме для такого типу навантаження Node.js демонструє найкращу продуктивність завдяки асинхронній моделі.

Дослідження продуктивності веб-фреймворків показує, що Node.js може обробляти до 10 000 одночасних з'єднань на одному сервері середньої потужності без деградації продуктивності [17]. Це більш ніж достатньо для потреб проєкту.

Другий фактор — єдина мова для всього стеку. Використання JavaScript і на клієнті, і на сервері дозволяє ефективніше організувати розробку. Функції валідації даних (наприклад, перевірка формату електронної пошти або номера телефону) можуть бути реалізовані один раз та використані і на фронтенді для миттєвої валідації форм, і на бекенді для гарантування цілісності даних.

Третій фактор — швидкість розробки API. Express є мінімалістичним фреймворком, що надає саме ті інструменти, які потрібні для створення RESTful API: маршрутизацію, middleware для обробки запитів, інтеграцію з шаблонізаторами. Відсутність зайвої функціональності та конфігурацій прискорює розробку простих API endpoints.

Четвертий фактор — екосистема та документація. Для Node.js існує величезна кількість навчальних матеріалів, прикладів коду, рішень типових проблем. Бібліотеки для роботи з JWT (jsonwebtoken), хешування паролів (bcryptjs), роботи з MongoDB (mongoose) є зрілими та добре документованими.

П'ятий фактор — інтеграція з MongoDB. Mongoose — це ODM (Object-Document Mapping) для MongoDB, що надає зручний API для роботи з базою даних через JavaScript. Схеми Mongoose дозволяють визначити структуру документів, валідацію полів, індекси, методи моделей — все через знайомий JavaScript-синтаксис.

2.2.3 Організація модульної структури серверного коду

Важливим аспектом є організація коду серверної частини у модулі з чіткою відповідальністю. Це забезпечить можливість легкого розширення функціональності та потенційної міграції до мікросервісної архітектури у майбутньому.

Кодова база серверної частини організована наступним чином. Папка models містить Mongoose-схеми для всіх сутностей системи: User (користувачі), Master (профілі майстрів), Order (замовлення), Review (відгуки), Category (категорії послуг). Кожна модель інкапсулює логіку роботи з відповідною колекцією MongoDB.

Папка routes містить файли маршрутизації Express. Кожен файл відповідає за групу пов'язаних API endpoints: auth.js (автентифікація), masters.js (операції з майстрами), orders.js (управління замовленнями), reviews.js (робота з відгуками), categories.js (отримання категорій). Роути використовують контролери для обробки бізнес-логіки.

Папка middleware містить функції проміжної обробки запитів. auth.js перевіряє наявність та валідність JWT-токена, додає інформацію про користувача до об'єкта запиту. validate.js містить функції валідації даних запитів через бібліотеку joi. errorHandler.js централізовано обробляє помилки та формує відповіді з відповідними HTTP-статусами.

Папка `utils` містить допоміжні функції, що використовуються в різних частинах додатку: `generateToken.js` для створення JWT-токенів, `hashPassword.js` для хешування паролів, `calculateRating.js` для обчислення рейтингів майстрів.

Головний файл `server.js` налаштовує Express-додаток, підключає `middleware` для обробки JSON, CORS, підключається до MongoDB, реєструє маршрути, запускає сервер на визначеному порту.

Така організація забезпечує чітке розділення відповідальностей. Моделі відповідають за структуру даних та низькорівневу логіку роботи з базою. Роути визначають `endpoints` та делегують обробку контролерам. `Middleware` обробляє крос-функціональні аспекти типу автентифікації та логування. Утиліти надають переважно використовувані функції.

2.3 Вибір системи управління базами даних

2.3.1 Порівняння реляційних та NoSQL баз даних

Вибір бази даних суттєво впливає на продуктивність, масштабованість та складність розробки системи.

Системи управління базами даних можна класифікувати на два основні типи: реляційні (SQL) та нереляційні (NoSQL). Кожен тип має свої сильні сторони та оптимальні сценарії використання.

Реляційні бази даних організовують дані у таблиці з чітко визначеною схемою. Зв'язки між сутностями реалізуються через зовнішні ключі. SQL-запити дозволяють виконувати складні операції об'єднання, групування, агрегації даних з кількох таблиць.

Переваги реляційних баз включають гарантії цілісності даних через транзакції ACID (Atomicity, Consistency, Isolation, Durability). Атомарність означає що транзакція виконується повністю або не виконується взагалі. Консистентність гарантує що база завжди знаходиться у валідному стані. Ізоляція забезпечує що

одночасні транзакції не заважають одна одній. Довговічність означає що після підтвердження транзакції дані збережено назавжди.

Ці гарантії є критичними для фінансових систем або додатків з складними бізнес-правилами. Для платформи пошуку майстрів транзакції можуть бути корисні, наприклад, при одночасному створенні замовлення та відправленні сповіщення майстру — обидві операції мають виконатись разом або не виконатись взагалі.

Структурована мова запитів SQL є стандартизованою та потужною. Складні аналітичні запити типу "знайти майстрів з рейтингом вище 4.5, що виконали більше 20 замовлень у категорії Сантехніка за останні три місяці з середньою ціною нижче 300 гривень" можуть бути реалізовані одним SQL-запитом з об'єднаннями та агрегаціями.

Недоліками реляційних баз є жорстка схема даних. Додавання нового поля до таблиці вимагає міграції, яка може бути довгою для великих таблиць. Об'єктно-реляційний імпеданс — розбіжність між об'єктною моделлю додатку та реляційною моделлю бази даних. Складність горизонтального масштабування через необхідність підтримки транзакцій між вузлами.

NoSQL бази даних відмовляються від жорсткої схеми та SQL на користь гнучкості та масштабованості. Існує кілька типів NoSQL баз: документо-орієнтовані (MongoDB, CouchDB), ключ-значення (Redis, DynamoDB), колонкові (Cassandra, HBase), графові (Neo4j).

Для веб-додатків найчастіше використовуються документо-орієнтовані бази, де дані зберігаються у вигляді JSON-подібних документів. Кожен документ може мати власну структуру без необхідності визначати схему заздалегідь.

Переваги NoSQL баз включають гнучкість схеми — можна додавати нові поля до документів без міграції всієї колекції. Природну відповідність JSON-структурам, що використовуються у веб-додатках. Горизонтальну масштабованість через шардування — розподіл даних між кількома серверами. Високу продуктивність для простих операцій читання та запису за ключем.

Недоліками є відсутність транзакцій між документами у багатьох NoSQL базах (хоча MongoDB додала підтримку мультидокументних транзакцій у версії 4.0). Складність реалізації складних запитів з об'єднаннями — NoSQL бази не підтримують JOIN операції. Необхідність денормалізації даних для уникнення множинних запитів.

2.3.2 Аналіз PostgreSQL та MongoDB

Розглянемо детальніше два конкретні варіанти: PostgreSQL як представника реляційних баз та MongoDB як документо-орієнтованої NoSQL бази.

PostgreSQL є потужною реляційною базою даних з відкритим кодом. Система підтримує повний стандарт SQL, ACID-транзакції, складні індекси, тригери, збережені процедури. PostgreSQL має розширені можливості роботи з JSON-даними через типи JSONB, що дозволяє поєднувати переваги реляційного та документного підходів.

Дослідження продуктивності PostgreSQL показує високу ефективність для складних аналітичних запитів завдяки розвиненому оптимізатору запитів [18]. Для платформи МайстерХаб це означає можливість ефективно генерувати звіти типу "топ майстрів за категоріями з розбивкою по місяцях".

Переваги PostgreSQL для проєкту включають надійність через ACID-транзакції, розвинену систему типів даних, потужні можливості індексування (B-tree, Hash, GiST, GIN), підтримку повнотекстового пошуку, активну спільноту та розвинену екосистему інструментів.

Недоліки включають необхідність визначення схеми таблиць заздалегідь, складність внесення змін до структури таблиць у продакшн-базах з великим обсягом даних, вищу складність горизонтального масштабування порівняно з NoSQL рішеннями.

MongoDB є документо-орієнтованою базою даних, що зберігає дані у форматі BSON (Binary JSON). Кожен документ є самодостатньою одиницею даних, що може містити вкладені об'єкти та масиви довільної глибини.

Архітектура MongoDB оптимізована для горизонтального масштабування через шардування. База підтримує автоматичне розподілення даних між шардами

на основі ключа шардування, балансування навантаження, реплікацію для забезпечення високої доступності [19].

Переваги MongoDB для проєкту включають гнучку схему, яка дозволяє легко еволюціонувати структуру даних, природну відповідність JSON-структурам, що спрощує роботу з даними у JavaScript-кодi, високу продуктивність для операцій читання за ключем, вбудовану підтримку горизонтального масштабування, зручний Query API без необхідності вчити SQL.

Недоліки включають відсутність транзакцій між документами у старіших версіях (в MongoDB 4.0+ додана підтримка), складність реалізації складних запитів з об'єднаннями, необхідність денормалізації даних, що може призводити до дублювання інформації.

2.3.3 Обґрунтування вибору документо-орієнтованої бази MongoDB

Для проєкту МайстерХаб обрано MongoDB як основну систему зберігання даних. Рішення базується на декількох факторах.

Перший фактор — природна відповідність структури даних документній моделі. Профіль майстра містить різноманітні дані: базову інформацію (ім'я, телефон), масив категорій змінної довжини, опис досвіду, масив фотографій портфоліо, агреговану статистику (рейтинг, кількість замовлень). У реляційній базі це вимагало б кількох таблиць з зв'язками: `users`, `masters`, `master_categories`, `portfolio_images`. У MongoDB весь профіль зберігається як єдиний документ, що спрощує запити та підвищує продуктивність.

Другий фактор — відсутність потреби у складних транзакціях для базових операцій. Основні операції системи — читання профілів майстрів, створення замовлень, додавання відгуків — є відносно простими та не вимагають атомарності між множиною сутностей. Створення замовлення оновлює один документ у колекції `orders`. Додавання відгуку створює документ у колекції `reviews` та оновлює рейтинг у документі майстра — ці операції можуть виконуватись послідовно без критичної необхідності в транзакціях.

Третій фактор — гнучкість еволюції схеми даних. На ранніх етапах проєкту структура даних може змінюватись на основі зворотного зв'язку користувачів.

Наприклад, може виникнути потреба додати поле "років досвіду" до профілю майстра або "спеціалізація" для категорій. У MongoDB достатньо почати записувати нові поля, старі документи продовжують працювати без міграції.

Четвертий фактор — інтеграція з Node.js через Mongoose. Mongoose надає зручний API для роботи з MongoDB через JavaScript. Визначення схем, валідація, middleware, віртуальні поля — все реалізується через знайому об'єктну модель JavaScript. Це прискорює розробку та зменшує ймовірність помилок.

П'ятий фактор — перспектива масштабування. Хоча початкова користувацька база є невеликою, MongoDB надає вбудовані механізми горизонтального масштабування через шардування. При зростанні даних можна додати нові вузли без зміни коду додатку.

2.3.4 Проєктування схем даних через Mongoose

Mongoose дозволяє визначити схеми для документів MongoDB з типізацією полів, валідацією, індексами та методами. Розглянемо ключові схеми проєкту.

Схема User визначає базову інформацію користувача: email (String, обов'язковий, унікальний, lowercase), password (String, обов'язковий, мінімум 6 символів), name (String, обов'язковий), phone (String, обов'язковий), role (String, enum з двома значеннями: 'client' або 'master'), createdAt (Date, default: Date.now). Індекс створюється на полі email для швидкого пошуку при автентифікації.

Схема Master визначає профіль виконавця: userId (ObjectId, ref: 'User', обов'язковий, унікальний), bio (String, опис досвіду), categories (Array of Strings, обов'язковий, мінімум одна категорія), hourlyRate (Number, обов'язковий, мінімум 0), avatarColor (String, обов'язковий, hex-колір), rating (Number, default: 0, мінімум 0, максимум 5), ordersCompleted (Number, default: 0), createdAt (Date). Індокси створюються на userId та categories для швидкого пошуку.

Схема Order визначає замовлення: clientId (ObjectId, ref: 'User'), masterId (ObjectId, ref: 'Master'), category (String, обов'язковий), description (String, обов'язковий), status (String, enum: pending, accepted, in_progress, completed, cancelled, default: pending), price (Number), estimatedHours (Number), preferredDate

(Date), createdAt (Date), updatedAt (Date). Індокси на clientId, masterId та status для ефективної фільтрації.

Схема Review визначає відгук: orderId (ObjectId, ref: 'Order', унікальний), clientId (ObjectId, ref: 'User'), masterId (ObjectId, ref: 'Master'), rating (Number, обов'язковий, мінімум 1, максимум 5), comment (String), createdAt (Date). Унікальний індекс на orderId гарантує один відгук на замовлення.

Така організація схем забезпечує валідацію даних на рівні бази, ефективний пошук через індокси, зручну роботу через Mongoose API.

2.4 Вибір технологій клієнтської частини

2.4.1 Порівняння React, Vue.js, Angular

Клієнтська частина визначає досвід користувачів та має бути інтуїтивною, швидкою, адаптивною для різних пристроїв.

Сучасні веб-додатки використовують компонентні фреймворки для організації інтерфейсу. Розглянемо три найпопулярніші варіанти.

React є JavaScript-бібліотекою для побудови користувацьких інтерфейсів, розробленою Facebook. Ключова концепція — компоненти як переважно використовувані блоки інтерфейсу. Віртуальний DOM забезпечує ефективне оновлення інтерфейсу через порівняння нового стану з попереднім та застосування мінімальних змін до реального DOM [20].

Переваги React включають декларативний підхід до опису інтерфейсу, великий вибір готових UI-бібліотек (Material-UI, Ant Design, Chakra UI), розвинену екосистему інструментів (React Router, Redux, React Query), можливість використання JSX для інтуїтивного опису компонентів, велику спільноту та ринок праці.

Недоліки включають необхідність додаткових бібліотек для повнофункціонального додатку (маршрутизація, управління станом), відносно

крута крива навчання для початківців через концепції хуків та контексту, відсутність офіційного стилю організації коду.

Vue.js є прогресивним JavaScript-фреймворком для побудови інтерфейсів. Фреймворк розроблений з акцентом на простоту освоєння та поступове впровадження. Дослідження порівняння продуктивності фронтенд-фреймворків показує що Vue.js має схожі показники з React за швидкістю рендерингу [21].

Переваги Vue.js включають простіший синтаксис шаблонів ближчий до HTML, вбудовану реактивність без необхідності використання хуків, офіційні рішення для маршрутизації (Vue Router) та управління станом (Vuex, Pinia), детальну офіційну документацію, нижчу криву навчання.

Недоліки включають меншу екосистему готових компонентів порівняно з React, менший ринок праці, менше сторонніх бібліотек та інструментів.

Angular є повнофункціональним TypeScript-фреймворком від Google. На відміну від React та Vue, Angular надає повний набір вбудованих інструментів: маршрутизацію, HTTP-клієнт, форми, анімації, тестування.

Переваги Angular включають TypeScript за замовчуванням для строгої типізації, повний набір вбудованих інструментів без потреби шукати сторонні рішення, потужну систему dependency injection, офіційну підтримку від Google.

Недоліки включають високу складність та крутішу криву навчання, більший розмір bundle через включення всіх інструментів, повільнішу розробку простих додатків через необхідність дотримання багатьох концепцій фреймворку.

2.4.2 Обґрунтування вибору React з TailwindCSS

Для проєкту МайстерХаб обрано React у поєднанні з TailwindCSS для стилізації. Рішення базується на кількох факторах.

Перший фактор — багатство екосистеми готових компонентів. Для React існує величезна кількість UI-бібліотек з готовими компонентами: кнопки, форми, модальні вікна, таблиці. Це прискорює розробку інтерфейсу без необхідності створювати кожен елемент з нуля.

Другий фактор — React Router для маршрутизації. Бібліотека надає декларативний підхід до організації навігації між сторінками додатку без

перезавантаження сторінки. Підтримка вкладених маршрутів, параметрів URL, програмної навігації робить React Router потужним інструментом для SPA.

Третій фактор — Context API для управління глобальним станом. Для додатку середньої складності Context API є достатнім для зберігання інформації про автентифікованого користувача, налаштувань, тимчасових даних. Це дозволяє уникнути prop drilling та спрощує доступ до глобального стану з будь-якого компонента.

Четвертий фактор — хуки для організації логіки. useState, useEffect, useContext, власні хуки дозволяють інкапсулювати переважно використовувану логіку та робити компоненти більш читабельними. Наприклад, хук useAuth може інкапсулювати логіку перевірки автентифікації та надавати інформацію про поточного користувача.

П'ятий фактор — TailwindCSS для швидкої стилізації. TailwindCSS — це utility-first CSS-фреймворк, що надає низькорівневі класи для стилізації елементів безпосередньо в розмітці. Замість написання окремих CSS-файлів, стилі застосовуються через класи типу flex, bg-blue-500, rounded-lg [22].

Переваги Tailwind включають швидкість розробки без необхідності придумувати назви класів, консистентність дизайну через використання визначеної системи кольорів та відступів, малий розмір фінального CSS-файлу через PurgeCSS, вбудовану підтримку responsive та dark mode.

Шостий фактор — спільнота та ресурси. Для React існує величезна кількість туторіалів, прикладів коду, відповідей на Stack Overflow. Це спрощує вирішення типових проблем та навчання.

2.4.3 Організація компонентної структури

Організація коду клієнтської частини базується на компонентному підході з чітким розділенням на переважно використовувані компоненти, сторінки, утиліти.

Папка components містить переважно використовувані UI-елементи. Navbar.jsx — навігаційна панель з логотипом, посиланнями, кнопками входу або інформацією про користувача. Footer.jsx — підвал сторінки з посиланнями та контактами. MasterCard.jsx — картка майстра для відображення в каталозі з

аватаром, ім'ям, рейтингом, категоріями, ціною. `OrderCard.jsx` — картка замовлення зі статусом, описом, інформацією про майстра. `StarRating.jsx` — компонент для відображення та введення оцінок зірками. `StatusBadge.jsx` — badge для відображення статусу замовлення з відповідним кольором. `ProtectedRoute.jsx` — НОС для захисту маршрутів від неавторизованих користувачів.

Папка `pages` містить компоненти повних сторінок. `Home.jsx` — головна сторінка з категоріями та топ майстрами. `Login.jsx` — форма входу в систему. `Register.jsx` — форма реєстрації з вибором ролі. `Masters.jsx` — каталог майстрів з фільтрами. `MasterProfile.jsx` — детальний профіль майстра.

Папка `context` містить React Context для глобального стану. `AuthContext.jsx` надає інформацію про поточного користувача, функції `login`, `logout`, `register`. Використання Context дозволяє будь-якому компоненту отримати доступ до стану автентифікації без передачі props через всю ієрархію.

Папка `utils` містить допоміжні функції. `api.js` налаштовує `axios instance` з базовою URL та `interceptors` для автоматичного додавання JWT-токена до заголовків запитів. `avatarColor.js` містить функцію генерації кольору на основі імені користувача. `helpers.js` містить різноманітні допоміжні функції для форматування дат, чисел, валідації.

Така організація забезпечує модульність коду, переважне використання компонентів, легкість навігації по проєкту.

2.5 Система автентифікації та безпеки

2.5.1 Аналіз методів автентифікації користувачів

Безпека є критичним аспектом веб-додатку, особливо для платформи що зберігає персональні дані користувачів.

Існує кілька підходів до організації автентифікації у веб-додатках. Розглянемо основні варіанти.

Сесійна автентифікація базується на збереженні стану користувача на сервері. Після успішного входу сервер створює сесію з унікальним ідентифікатором, зберігає інформацію про користувача в пам'яті або базі даних, відправляє ідентифікатор сесії клієнту в cookie. При наступних запитах клієнт відправляє cookie з ідентифікатором, сервер знаходить відповідну сесію та ідентифікує користувача.

Переваги сесійного підходу включають можливість примусового завершення сесії на сервері, простоту реалізації, відсутність необхідності зберігати чутливі дані на клієнті. Недоліки включають необхідність зберігання сесій на сервері, що ускладнює горизонтальне масштабування, додаткові запити до бази даних або кешу для перевірки сесії, складність організації для мікросервісних архітектур.

Токенізована автентифікація через JWT (JSON Web Tokens) є альтернативним підходом. Після успішного входу сервер генерує JWT-токен, що містить закодовану інформацію про користувача та підпис для перевірки автентичності. Токен відправляється клієнту, який зберігає його в localStorage або sessionStorage. При наступних запитах клієнт включає токен в заголовок Authorization.

JWT складається з трьох частин розділених крапками: header (алгоритм підпису), payload (дані про користувача), signature (підпис для перевірки). Сервер може перевірити валідність токена без звернення до бази даних через перевірку підпису [23].

Переваги JWT включають stateless nature — сервер не зберігає інформацію про активні токени, масштабованість через відсутність спільного сховища сесій, можливість використання в мікросервісних архітектурах через самодостатність токена. Недоліки включають неможливість примусового завершення сесії (токен валідний до закінчення терміну дії), більший розмір токена порівняно з ідентифікатором сесії, потенційну вразливість при зберіганні в localStorage до XSS-атак.

OAuth 2.0 є протоколом авторизації для делегованого доступу. Використовується для "Увійти через Google/Facebook" функціональності.

Складність реалізації та необхідність реєстрації додатку у провайдерів робить OAuth надмірним для базового MVP.

2.5.2 Реалізація JWT-токенізації

Для проєкту МайстерХаб обрано JWT-токенізацію. Рішення базується на простоті реалізації для stateless API, відсутності необхідності в розподіленому сховищі сесій, природній відповідності архітектурі SPA де клієнт виконує багато логіки.

Процес автентифікації працює наступним чином. При реєстрації користувач надає email, пароль, ім'я, телефон, роль. Сервер хешує пароль через bcrypt з factor 10 (достатньо для балансу безпеки та продуктивності), зберігає користувача в базі, генерує JWT-токен з payload що містить userId та role, відправляє токен клієнту.

При вході користувач надає email та пароль. Сервер шукає користувача за email, порівнює хеш введеного пароля з збереженим через bcrypt.compare, при успіху генерує новий JWT-токен, відправляє токен клієнту.

JWT-токен має обмежений термін дії 24 години для балансу безпеки та зручності. Коротші терміни вимагають частішої повторної автентифікації, довші збільшують ризики при компрометації токена.

Клієнт зберігає токен в localStorage для збереження між перезавантаженнями сторінки. При кожному запиті до API axios interceptor автоматично додає токен в заголовок Authorization: Bearer <token>.

Сервер перевіряє токен через middleware. Для захищених endpoints middleware перевіряє наявність заголовка Authorization, витягує токен, верифікує підпис через jwt.verify з секретним ключем, декодує payload та додає інформацію про користувача до об'єкта запиту, дозволяє продовжити обробку запиту.

2.5.3 Механізми захисту від атак

Веб-додаток є потенційною цілью для різних типів атак. Необхідно впровадити захисні механізми.

Захист паролів реалізується через bcrypt — алгоритм хешування спеціально розроблений для паролів. Bcrypt використовує salt (випадкові дані що додаються до пароля перед хешуванням) для запобігання атакам через rainbow tables. Work

factor визначає складність обчислення хешу — більший factor означає повільніше хешування, що утруднює brute force атаки але збільшує навантаження на сервер [24].

Захист від SQL-ін'єкцій не є релевантним для MongoDB, проте NoSQL-ін'єкції також можливі. Mongoose автоматично екранує запити, проте валідація вхідних даних є додатковим рівнем захисту. Бібліотека `jo` використовується для валідації структури та типів даних запитів перед обробкою.

Захист від XSS (Cross-Site Scripting) реалізується через escaped output у React. React автоматично екранує всі дані при рендерингу, що запобігає виконанню шкідливого JavaScript. Проте `dangerouslySetInnerHTML` має використовуватись з обережністю. Захист від CSRF (Cross-Site Request Forgery) частково забезпечується через використання JWT у заголовках замість cookies. CSRF атаки експлуатують автоматичну відправку cookies браузером, JWT в заголовку Authorization має бути доданий JavaScript-кодом, що унеможлиблює автоматичну відправку з інших сайтів.

CORS (Cross-Origin Resource Sharing) налаштовується для дозволу запитів лише з довірених доменів. У розробці дозволяється localhost, у продакшн — домен фронтенд-додатку.

Rate limiting обмежує кількість запитів від одного IP-адреси для запобігання brute force атакам на endpoints входу та реєстрації. Бібліотека `express-rate-limit` дозволяє налаштувати ліміти типу "максимум 5 спроб входу за 15 хвилин".

2.6 Висновки до розділу 2

У другому розділі проведено порівняльний аналіз архітектурних підходів та технологічних рішень для реалізації веб-платформи МайстерХаб. Аналіз монолітної та мікросервісної архітектур виявив що для MVP з обмеженим терміном розробки та невеликою командою монолітний підхід з модульною організацією

коду є оптимальним. Це забезпечує швидкість початкової розробки при збереженні можливості майбутньої міграції до мікросервісів.

Порівняння backend-фреймворків (Node.js, Python, Ruby) показало переваги Node.js з Express для проєкту через високу продуктивність при обробці великої кількості одночасних з'єднань, єдину мову JavaScript для фронтенду та бекенду, багату екосистему бібліотек, швидкість розробки REST API.

Аналіз систем управління базами даних виявив що MongoDB краще підходить для проєкту через природну відповідність JSON-структурам, гнучкість схеми для швидкої еволюції моделі даних, вбудовану підтримку горизонтального масштабування, зручну інтеграцію з Node.js через Mongoose. Порівняння фронтенд-фреймворків (React, Vue.js, Angular) показало що React у поєднанні з TailwindCSS надає оптимальне співвідношення швидкості розробки, багатства екосистеми готових компонентів, продуктивності та гнучкості для проєкту.

Аналіз методів автентифікації обґрунтував вибір JWT-токенізації через stateless nature, масштабованість, природну відповідність архітектурі SPA. Визначено механізми захисту від основних типів атак через bcrypt для паролів, валідацію даних, CORS, rate limiting. Обрана технологічна платформа — Node.js + Express + MongoDB + React + TailwindCSS — забезпечує баланс між швидкістю розробки, продуктивністю, масштабованістю та вартістю підтримки для веб-платформи пошуку виконавців побутових послуг.

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Загальна архітектура платформи МайстерХаб

3.1.1 Компонентна діаграма системи

Архітектура системи визначає взаємодію компонентів, потоки даних, інтерфейси між модулями. Правильне проєктування архітектури забезпечує масштабованість, надійність, зручність підтримки системи.

Платформа МайстерХаб складається з трьох основних рівнів: клієнтського, серверного та рівня даних. Така триланкова архітектура є класичним підходом для веб-додатків, що забезпечує розділення відповідальностей та незалежність компонентів [1].

Клієнтський рівень представлений односторінковим додатком (SPA) на React. Додаток виконується в браузері користувача та відповідає за відображення інтерфейсу, обробку дій користувача, валідацію форм, управління локальним станом. Клієнт взаємодіє з сервером виключно через HTTP REST API, що забезпечує слабку зв'язаність між рівнями.

Серверний рівень реалізований на Node.js з Express та організований у вигляді модулів. Модуль автентифікації відповідає за реєстрацію користувачів, вхід в систему, генерацію та валідацію JWT-токенів, управління сесіями. Модуль управління майстрами забезпечує створення та редагування профілів виконавців, пошук майстрів за критеріями, отримання детальної інформації про майстра. Модуль замовлень керує створенням заявок на послуги, зміною статусів замовлень, відстеженням історії. Модуль рейтингування обробляє створення відгуків, розрахунок середніх оцінок, виявлення аномальних патернів. Модуль категорій надає інформацію про доступні категорії послуг.

Рівень даних представлений MongoDB - документо-орієнтованою базою даних. База містить п'ять основних колекцій: users (користувачі), masters (профілі майстрів), orders (замовлення), reviews (відгуки), categories (категорії послуг). Mongoose як ODM забезпечує зручний інтерфейс для роботи з MongoDB через визначення схем, валідацію даних, методи запитів.

Взаємодія між рівнями організована наступним чином. Користувач виконує дію в браузері (наприклад, натискає кнопку "Створити замовлення"). React-компонент обробляє подію, формує HTTP-запит через axios до відповідного API endpoint. Запит проходить через middleware на сервері для автентифікації, валідації, логування. Відповідний контролер обробляє бізнес-логіку, взаємодіє з базою даних через Mongoose-моделі. Результат повертається клієнту у форматі JSON. React-компонент оновлює інтерфейс на основі отриманих даних.

3.1.2 Взаємодія клієнт-серверної частини

Комунікація між клієнтом та сервером організована за принципами REST (Representational State Transfer) архітектури. Дослідження показують що RESTful API є найпопулярнішим підходом до організації взаємодії у веб-додатках завдяки простоті, зрозумілості, кешованості [2].

REST використовує стандартні HTTP-методи для різних операцій. GET для отримання даних без зміни стану сервера, використовується для отримання списків майстрів, деталей замовлення, відгуків. POST для створення нових ресурсів, використовується для реєстрації користувачів, створення замовлень, додавання відгуків. PUT для повного оновлення існуючих ресурсів, використовується для редагування профілю майстра. PATCH для часткового оновлення, використовується для зміни статусу замовлення. DELETE для видалення ресурсів, використовується рідко в даному проєкті.

HTTP-статуси використовуються для індикації результату операції. 200 OK означає успішне виконання запиту з поверненням даних. 201 Created вказує на успішне створення нового ресурсу. 400 Bad Request сигналізує про помилку валідації даних запиту. 401 Unauthorized означає відсутність або невалідність автентифікації. 403 Forbidden вказує що користувач автентифікований але не має прав на операцію. 404 Not Found означає що запитаний ресурс не знайдено. 500 Internal Server Error вказує на помилку на сервері.

Формат даних для обміну - JSON (JavaScript Object Notation). Це текстовий формат що легко читається людиною та ефективно обробляється JavaScript. Запити з даними включають заголовок Content-Type: application/json та JSON-об'єкт у тілі. Відповіді також повертають JSON з відповідними даними або повідомленнями про помилки.

Автентифікація реалізується через JWT-токени в заголовку Authorization. Після успішного входу клієнт отримує токен та зберігає його в localStorage. При кожному наступному запиті axios interceptor автоматично додає заголовок Authorization: Bearer <token>. Сервер валідує токен через middleware перед обробкою запиту.

3.1.3 Організація потоків даних

Розглянемо детально потоки даних для ключових сценаріїв використання системи.

Сценарій створення замовлення демонструє повний цикл взаємодії компонентів. Клієнт переглядає профіль майстра та натискає кнопку "Замовити послугу". Відкривається модальне вікно з формою створення замовлення. Користувач заповнює опис робіт, вказує бажану дату, орієнтовну кількість годин. При натисканні "Надіслати" React-компонент валідує дані локально (перевірка обов'язкових полів, формату дати). Якщо валідація успішна, формується POST-запит до `/api/orders` з даними замовлення.

На сервері запит спочатку проходить через `auth middleware`, що перевіряє JWT-токен та додає `userId` до об'єкта запиту. Далі `validate middleware` перевіряє структуру та типи даних через `joі-схему`. Контролер `orders` отримує валідований запит, створює новий документ `Order` у `MongoDB` через `Mongoose`, встановлює статус `"pending"`, зберігає посилання на `clientId` (з токена) та `masterId` (з параметрів запиту), повертає створене замовлення з ідентифікатором. React-компонент отримує відповідь, закриває модальне вікно, показує `toast-сповіщення` про успіх, оновлює список замовлень користувача.

Сценарій пошуку майстрів демонструє роботу з фільтрами та пагінацією. Користувач відкриває сторінку каталогу майстрів. Компонент `Masters.jsx` при монтуванні відправляє GET-запит до `/api/masters` без параметрів для отримання всіх майстрів. Користувач встановлює фільтри: категорія "Сантехніка", мінімальний рейтинг 4.0, максимальна ставка 400 грн/год. При зміні фільтрів компонент формує новий GET-запит з query-параметрами: `/api/masters?category=Сантехніка&minRating=4.0&maxRate=400`.

На сервері контролер `masters` отримує параметри запиту, формує об'єкт фільтру для `MongoDB`. Якщо вказана категорія, додається умова `{categories: {$in: ["Сантехніка"]}}`. Якщо вказаний мінімальний рейтинг, додається `{rating: {$gte: 4.0}}`. Якщо вказана максимальна ставка, додається `{hourlyRate: {$lte: 400}}`. Виконується запит до колекції `Masters` з об'єднаними фільтрами. Результати

сортуються за комплексним показником (рейтинг + досвід + швидкість відповіді). Застосовується пагінація через `skip` та `limit` для повернення лише 12 майстрів на сторінку. Сервер повертає масив майстрів та метадані пагінації (`total count`, `current page`). React-компонент відображає картки майстрів у `grid layout`.

Сценарій залишення відгуку демонструє оновлення агрегованих даних. Після завершення замовлення клієнт отримує можливість залишити відгук. Користувач натискає "Залишити відгук", заповнює форму з оцінкою (1-5 зірок) та текстовим коментарем, натискає "Опублікувати". Формується POST-запит до `/api/reviews` з `orderId`, `rating`, `comment`.

На сервері контролер `reviews` перевіряє що замовлення має статус `"completed"`, що відгук залишає саме клієнт цього замовлення, що для цього замовлення ще не було відгуку. Створюється документ `Review` у MongoDB. Запускається процес оновлення рейтингу майстра: завантажуються всі відгуки цього майстра, обчислюється нове середнє значення через алгоритм байєсівського згладжування, оновлюється поле `rating` у документі `Master`. Сервер повертає створений відгук. React-компонент показує сповіщення про успіх, перенаправляє на список замовлень.

3.2 Проєктування структури бази даних

3.2.1 Модель користувачів (Users)

Структура бази даних визначає організацію збереження та зв'язків між даними. Правильне проєктування забезпечує цілісність даних, продуктивність запитів, зручність розширення.

Колекція `Users` зберігає базову інформацію про всіх користувачів системи незалежно від їх ролі. Документ користувача містить наступні поля.

Поле `_id` є унікальним ідентифікатором типу `ObjectId`, що автоматично генерується MongoDB при створенні документа. `ObjectId` складається з 12 байт: 4

байти timestamp (час створення), 5 байт random value (випадкове значення), 3 байти counter (лічильник). Це забезпечує унікальність без необхідності в централізованій системі генерації ідентифікаторів [3].

Поле email типу String зберігає електронну адресу користувача. Накладається валідація формату через регулярний вираз для email. Встановлюється обмеження unique: true для унікальності в колекції. Опція lowercase: true автоматично конвертує значення у нижній регістр для уніфікації. Створюється індекс на цьому полі для швидкого пошуку при автентифікації.

Поле password типу String зберігає хеш пароля. Пароль ніколи не зберігається у відкритому вигляді з міркувань безпеки. Перед збереженням пароль хешується через bcrypt з work factor 10. Валідація мінімальної довжини 6 символів застосовується до відкритого пароля перед хешуванням.

Поле name типу String зберігає повне ім'я користувача. Встановлюється валідація обов'язковості та мінімальної довжини 2 символи. Використовується для відображення в інтерфейсі та ініціалів для аватарів.

Поле phone типу String зберігає номер телефону. Валідація формату через регулярний вираз для українських номерів (+380XXXXXXXXXX). Телефон використовується для контакту між клієнтами та майстрами після узгодження замовлення.

Поле role типу String визначає тип користувача. Обмежується enum з двома значеннями: 'client' (клієнт) або 'master' (майстер). За замовчуванням встановлюється 'client'. Роль використовується для розмежування прав доступу та відображення різних інтерфейсів.

Поле createdAt типу Date зберігає час створення облікового запису. За замовчуванням встановлюється поточний час через Date.now. Використовується для статистики та сортування нових користувачів (ліст. 3.1).

Лістинг 3.1 - Mongoose-схема для User

```
const userSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    match: /^[^\s@]+@[^\s@]+\.[^\s@]+$/
  },
  password: {
    type: String,
    required: true,
    minlength: 6
  },
  name: {
    type: String,
    required: true,
    minlength: 2
  },
  phone: {
    type: String,
    required: true,
    match: /^\\+380\\d{9}$/
  },
  role: {
    type: String,
    enum: ['client', 'master'],
    default: 'client'
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

userSchema.index({ email: 1 });
```

3.2.2 Модель профілів майстрів (Masters)

Колекція Masters зберігає розширену інформацію про майстрів -користувачів з роллю 'master'. Документ профілю майстра містить наступні поля.

Поле `userId` типу `ObjectId` є зовнішнім ключем що посилається на документ у колекції `Users`. Встановлюється обмеження `unique: true` оскільки кожен користувач може мати лише один профіль майстра. Опція `ref: 'User'` дозволяє Mongoose виконувати `populate` для автоматичного завантаження даних користувача при запитах.

Поле `bio` типу `String` зберігає детальний опис досвіду та навичок майстра. Необов'язкове поле що дозволяє виконавцю розповісти про свою кваліфікацію, підходи до роботи, спеціалізації. Обмежується максимальною довжиною 1000 символів для запобігання надмірно довгим текстам.

Поле `categories` типу `Array of Strings` зберігає список категорій послуг що надає майстер. Валідація вимагає мінімум одну категорію. Значення мають відповідати назвам з колекції `Categories`. Індекс на цьому полі забезпечує швидкий пошук майстрів за категорією.

Поле `hourlyRate` типу `Number` зберігає погодинну ставку майстра в гривнях. Валідація вимагає додатне число. Використовується для орієнтовного розрахунку вартості робіт та фільтрації майстрів за ціновим діапазоном.

Поле `avatarColor` типу `String` зберігає hex-код кольору для аватара з ініціалами. Генерується автоматично на основі імені користувача для забезпечення візуального розрізнення майстрів у списках. Валідується формат hex-кольору через регулярний вираз.

Поле `rating` типу `Number` зберігає середній рейтинг майстра. За замовчуванням встановлюється 0 для нових майстрів. Обмежується діапазоном від 0 до 5. Оновлюється автоматично при додаванні нових відгуків через спеціальний алгоритм.

Поле `ordersCompleted` типу `Number` підраховує кількість успішно завершених замовлень. За замовчуванням 0. Інкрементується при зміні статусу замовлення на 'completed'. Використовується у формулі інтегрального показника для сортування майстрів.

Поле `createdAt` типу `Date` зберігає час створення профілю майстра (ліст. 3.2).

Лістинг 3.2 - Mongoose-схема для Master

```
const masterSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
    unique: true
  },
  bio: String,
  categories: [String],
  hourlyRate: Number,
  avatarColor: String,
  rating: Number,
  ordersCompleted: Number,
  createdAt: Date
})
```

```

    bio: {
      type: String,
      maxlength: 1000
    },
    categories: {
      type: [String],
      required: true,
      validate: v => Array.isArray(v) && v.length > 0
    },
    hourlyRate: {
      type: Number,
      required: true,
      min: 0
    },
    avatarColor: {
      type: String,
      required: true,
      match: /^#[0-9A-F]{6}$/i
    },
    rating: {
      type: Number,
      default: 0,
      min: 0,
      max: 5
    },
    ordersCompleted: {
      type: Number,
      default: 0
    },
    createdAt: {
      type: Date,
      default: Date.now
    }
  }
});

masterSchema.index({ userId: 1 });
masterSchema.index({ categories: 1 });
masterSchema.index({ rating: -1 });

```

3.2.3 Модель замовлень (Orders)

Колекція Orders зберігає інформацію про заявки на послуги від клієнтів до майстрів. Документ замовлення відображає повний життєвий цикл від створення до завершення.

Поле `clientId` типу `ObjectId` посилається на користувача-клієнта що створив замовлення. Індекс на цьому полі забезпечує швидке отримання замовлень конкретного клієнта.

Поле `masterId` типу `ObjectId` посилається на майстра якому адресоване замовлення. Індекс дозволяє майстру ефективно отримувати список вхідних та активних замовлень.

Поле `category` типу `String` вказує категорію послуги. Дублюється з профілю майстра для зручності запитів без необхідності `join` операцій. Дозволяє фільтрувати замовлення за категоріями.

Поле `description` типу `String` містить детальний опис необхідних робіт від клієнта. Обов'язкове поле з мінімальною довжиною 10 символів для забезпечення змістовності опису.

Поле `status` типу `String` відображає поточний стан замовлення. Обмежується `enum` з п'ятьма значеннями: `'pending'` (очікує прийняття майстром), `'accepted'` (прийнято майстром), `'in_progress'` (виконується), `'completed'` (завершено), `'cancelled'` (скасовано). За замовчуванням `'pending'`. Індекс на статусі прискорює фільтрацію замовлень за станом.

Поле `price` типу `Number` зберігає узгоджену ціну робіт у гривнях. Заповнюється після прийняття замовлення майстром. Необов'язкове оскільки на етапі створення заявки ціна може бути невідома.

Поле `estimatedHours` типу `Number` вказує орієнтовну тривалість робіт у годинах від клієнта. Необов'язкове поле що допомагає майстру оцінити обсяг робіт.

Поле `preferredDate` типу `Date` зберігає бажану дату виконання робіт. Необов'язкове поле що дозволяє клієнту вказати коли він потребує послугу.

Поле `createdAt` та `updatedAt` типу `Date` відстежують час створення та останньої зміни замовлення. `UpdatedAt` автоматично оновлюється при будь-якій модифікації документа через `Mongoose timestamps` (ліст. 3.3).

Лістинг 3.3 - Mongoose-схема для Order

```
const orderSchema = new mongoose.Schema({
  clientId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  masterId: {
```

```

    type: mongoose.Schema.Types.ObjectId,
    ref: 'Master',
    required: true
  },
  category: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true,
    minlength: 10
  },
  status: {
    type: String,
    enum: ['pending', 'accepted', 'in_progress', 'completed',
'cancelled'],
    default: 'pending'
  },
  price: Number,
  estimatedHours: Number,
  preferredDate: Date
}, {
  timestamps: true
});

orderSchema.index({ clientId: 1 });
orderSchema.index({ masterId: 1 });
orderSchema.index({ status: 1 });

```

3.2.4 Модель відгуків (Reviews)

Колекція Reviews зберігає оцінки та коментарі клієнтів про роботу майстрів після завершення замовлень. Відгуки є основою рейтингової системи.

Поле `orderId` типу `ObjectId` посилається на завершене замовлення. Встановлюється обмеження `unique: true` оскільки для одного замовлення може бути лише один відгук. Це запобігає дублюванню оцінок та маніпуляціям.

Поле `clientId` типу `ObjectId` посилається на клієнта що залишив відгук. Дублюється з `Order` для зручності запитів. Дозволяє перевірити що відгук залишено саме клієнтом замовлення.

Поле `masterId` типу `ObjectId` посилається на майстра що отримав відгук. Індекс на цьому полі забезпечує швидке отримання всіх відгуків конкретного майстра для відображення у профілі.

Поле `rating` типу `Number` зберігає оцінку від 1 до 5 зірок. Обов'язкове поле з валідацією діапазону. Використовується для обчислення середнього рейтингу майстра.

Поле `comment` типу `String` містить текстовий коментар клієнта. Необов'язкове поле що дозволяє надати детальніший зворотний зв'язок. Обмежується максимальною довжиною 500 символів.

Поле `createdAt` типу `Date` зберігає час публікації відгуку. Використовується для хронологічного сортування та аналізу розподілу відгуків у часі для виявлення аномалій (ліст. 3.4).

Лістинг 3.4 - Mongoose-схема для Review

```
const reviewSchema = new mongoose.Schema({
  orderId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Order',
    required: true,
    unique: true
  },
  clientId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  masterId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Master',
    required: true
  },
  rating: {
    type: Number,
    required: true,
    min: 1,
    max: 5
  },
  comment: {
    type: String,
    maxlength: 500
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});
```

```
reviewSchema.index({ masterId: 1 });
reviewSchema.index({ createdAt: -1 });
```

3.2.5 Модель категорій послуг (Categories)

Колекція Categories зберігає довідкову інформацію про типи послуг що надаються через платформу. Використовується для класифікації майстрів та замовлень.

Поле name типу String містить назву категорії українською мовою. Встановлюється обмеження unique: true для унікальності назв. Використовується як ключ для зв'язку з іншими колекціями.

Поле icon типу String зберігає назву іконки з бібліотеки lucide-react. Використовується для візуального відображення категорій в інтерфейсі. Приклади: "Wrench" для сантехніки, "Zap" для електрики.

Поле description типу String містить короткий опис послуг у категорії. Відображається на головній сторінці під назвою категорії для кращого розуміння користувачами.

Лістинг 3.5 - Mongoose-схема для Category

```
const categorySchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    unique: true
  },
  icon: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true
  }
});
```

Початкові дані для колекції Categories:

```
[
  { name: "Сантехніка", icon: "Wrench", description: "Ремонт і встановлення сантехніки" },
  { name: "Електрика", icon: "Zap", description: "Електричні роботи та ремонт" },
  { name: "Прибирання", icon: "Sparkles", description: "Професійне прибирання приміщень" },
```

```

    { name: "Ремонт техніки", icon: "Plug", description: "Ремонт побутової техніки" },
    { name: "Косметичний ремонт", icon: "Paintbrush", description: "Косметичний ремонт приміщень" },
    { name: "Переїзд", icon: "Package", description: "Допомога з переїздом" }
  ]

```

3.2.6 Зв'язки між колекціями через ObjectId

MongoDB не підтримує foreign key constraints як реляційні бази, проте зв'язки між документами реалізуються через збереження ObjectId посилань. Розглянемо організацію зв'язків у системі.

Зв'язок User - Master є зв'язком один-до-одного. Кожен користувач з роллю 'master' має рівно один профіль у колекції Masters. Поле `userId` у Masters посилається на `_id` у Users через `ref: 'User'`. Mongoose метод `populate` дозволяє автоматично завантажувати дані користувача:

```

const master = await Master.findById(masterId).populate('userId');
// master.userId тепер містить повний об'єкт User замість лише ObjectId

```

Зв'язок Master - Order є зв'язком один-до-багатьох. Один майстер може мати багато замовлень. Поле `masterId` у Orders посилається на `_id` у Masters. Для отримання всіх замовлень майстра:

```

const orders = await Order.find({ masterId: masterObjectId });

```

Зв'язок User - Order (як клієнт) також є зв'язком один-до-багатьох. Один користувач може створити багато замовлень. Поле `clientId` у Orders посилається на `_id` у Users.

Зв'язок Order - Review є зв'язком один-до-одного. Для кожного завершеного замовлення може бути створений лише один відгук. Унікальний індекс на `orderId` у Reviews гарантує це обмеження на рівні бази даних.

Такий підхід до організації зв'язків забезпечує гнучкість NoSQL при збереженні логічної цілісності даних через валідацію на рівні додатку.

3.3 Проєктування REST API

API є контрактом між клієнтською та серверною частинами. Детальна специфікація забезпечує узгодженість розробки та можливість паралельної роботи над компонентами [4].

Група endpoints `/api/auth` відповідає за реєстрацію, вхід та отримання інформації про поточного користувача.

POST `/api/auth/register` - реєстрація нового користувача.

Request body:

```
{  
  "email": "ivan@example.com",  
  "password": "securepass123",  
  "name": "Іван Петренко",  
  "phone": "+380501234567",  
  "role":
```

Рисунок 3.1 – Реєстрація нового користувача

3.4 Висновки до розділу 3

У третьому розділі здійснено комплексне проєктування веб-платформи МайстерХаб з детальною розробкою всіх ключових компонентів системи.

Розроблена загальна архітектура системи базується на триланковій моделі з чітким розділенням відповідальностей між рівнями. Клієнтський рівень реалізовано як односторінковий додаток на React, що забезпечує динамічний інтерфейс без перезавантаження сторінки. Серверний рівень організовано у вигляді модулів на Node.js з Express, кожен з яких відповідає за конкретну бізнес-функцію:

автентифікація, управління майстрами, обробка замовлень, рейтингування. Рівень даних представлений MongoDB з п'ятьма колекціями, що забезпечують гнучке зберігання структурованої інформації. Взаємодія між рівнями організована через RESTful API з використанням JWT-токенізації, що гарантує stateless nature серверної частини та масштабованість системи.

Спроектowana структура бази даних забезпечує ефективне зберігання та швидкий доступ до даних. Колекція Users містить базову інформацію про всіх користувачів системи з розмежуванням ролей клієнтів та майстрів. Колекція Masters розширює профілі виконавців детальною інформацією про компетенції, категорії послуг, ціноутворення та агреговану статистику. Колекція Orders відстежує повний життєвий цикл замовлень від створення до завершення через п'ять визначених станів. Колекція Reviews зберігає відгуки клієнтів з унікальним обмеженням один відгук на замовлення, що запобігає маніпуляціям. Колекція Categories надає довідкову інформацію про типи послуг. Зв'язки між колекціями реалізовані через ObjectId посилання з можливістю автоматичного завантаження пов'язаних даних через Mongoose populate, що поєднує гнучкість NoSQL з логічною цілісністю реляційних баз.

Детальна специфікація REST API визначає контракт взаємодії між клієнтською та серверною частинами. Розроблено чотирнадцять endpoints згруповані за функціональністю: три для автентифікації забезпечують реєстрацію, вхід та отримання інформації про поточного користувача; чотири для майстрів надають можливості пошуку, перегляду профілів, редагування та отримання статистики; п'ять для замовлень керують створенням заявок, відстеженням статусів, прийняттям та оновленням; два для відгуків дозволяють створювати оцінки та отримувати список відгуків майстра. Для кожного endpoint визначено HTTP-метод, структуру запиту з обов'язковими та опціональними параметрами, формат успішної відповіді, можливі коди помилок з поясненнями. Така детальна документація забезпечує узгодженість розробки та можливість паралельної роботи команд над різними компонентами.

Розроблені математичні моделі рейтингування забезпечують об'єктивну та стійку до маніпуляцій оцінку майстрів. Алгоритм байєсівського згладжування вирішує проблему малої вибірки через введення попередньої оцінки з вагою десяти псевдовідгуків, що дозволяє новим майстрам починати з розумною оцінкою 4.0 та поступово наближатись до реального середнього при накопиченні відгуків. Багатофакторний інтегральний показник враховує не лише рейтинг але й досвід (через логарифм кількості замовлень) та оперативність (через швидкість відповіді), що забезпечує справедливе ранжування майстрів у каталозі з ваговими коефіцієнтами 60%, 25% та 15% відповідно. Алгоритм виявлення аномальних відгуків аналізує три типи підозрілих патернів: часову аномалію через концентрацію відгуків у короткі проміжки часу, новизну рецензентів через високий відсоток відгуків від щойно зареєстрованих користувачів без іншої активності, текстову подібність коментарів через обчислення косинусної відстані між векторами слів. Інтегральний показник підозрілості з пороговим значенням 0.5 визначає відгуки для виключення з розрахунку рейтингу та направлення на ручну модерацію.

Спроектовані користувацькі інтерфейси забезпечують інтуїтивну взаємодію з системою для різних типів користувачів. Структура навігації адаптується залежно від стану автентифікації та ролі користувача, надаючи неавторизованим базовий доступ до каталогу та профілів майстрів, клієнтам — додаткові можливості створення замовлень та залишення відгуків, майстрам — інструменти управління заявками та статистику через панель управління. Детально розроблені сценарії взаємодії охоплюють повні цикли використання: від пошуку майстра через фільтрацію за категорією, рейтингом та ціною до створення замовлення з описом робіт та бажаною датою; від прийняття заявки майстром з встановленням узгодженої ціни до зміни статусів у процесі виконання; від завершення робіт до залишення відгуку клієнтом з оцінкою та коментарем. Дизайн-макети визначають layout всіх основних сторінок з використанням сучасних UI-патернів: hero-секція з великим заголовком та пошуком на головній, sidebar з фільтрами та grid карток у каталозі, детальний профіль з секціями інформації та відгуків, модальні вікна для

створення замовлень та відгуків. Адаптивна верстка забезпечує коректне відображення на екранах від 320px (смартфони) до 1920px+ (великі монітори) через визначені breakpoints та трансформацію layout: grid у три колонки на десктопі змінюється на дві колонки на планшеті та одну на мобільному, sidebar фільтрів колапсується у modal на малих екранах, navbar трансформується у burger menu.

Спроектowana архітектура та компоненти системи забезпечують баланс між функціональними можливостями, зручністю використання та технічною реалізованістю. Модульна організація серверного коду з чітким розділенням models, routes, middleware, utils полегшує розробку, тестування та підтримку. Документo-орієнтована модель MongoDB природно відображає структуру профілів майстрів зі змінною кількістю категорій та дозволяє еволюцію схеми без міграцій. RESTful API з детальною специфікацією створює прозорий контракт між frontend та backend командами. Математично обґрунтовані алгоритми рейтингування забезпечують довіру користувачів до системи оцінювання. Продумані користувацькі інтерфейси з адаптивним дизайном гарантують доступність платформи на будь-яких пристроях.

Результати проєктування створюють міцну основу для реалізації веб-платформи МайстерХаб, що здатна забезпечити прозору взаємодію між замовниками та виконавцями побутових послуг через об'єктивну рейтингову систему та зручні інструменти управління замовленнями. Всі спроектовані компоненти узгоджені між собою та готові до технічної реалізації у наступному розділі.

4 ПРОЄКТУВАННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

4.1 Головна сторінка та навігаційна структура

Користувацький інтерфейс є критичним елементом успішності веб-платформи, оскільки визначає зручність взаємодії та загальне враження

користувачів від системи. Дослідження показують, що 88% користувачів не повертаються на сайт після негативного досвіду взаємодії з інтерфейсом, що підкреслює необхідність ретельного проєктування UI/UX. Для платформи МайстерХаб розроблено інтерфейс, орієнтований на інтуїтивність, доступність та ефективність виконання ключових сценаріїв використання.

Головна сторінка платформи виконує функції представлення сервісу, демонстрації основних можливостей та надання швидкого доступу до ключових функцій. Структура сторінки організована за принципом перевернутої піраміди, де найважливіша інформація розміщується вгорі, а деталі поступово розкриваються при прокручуванні.

Верхня навігаційна панель містить логотип платформи МайстерХаб ліворуч, що є кліком для повернення на головну, центральну навігацію з посиланнями Головна та Майстри, праву групу елементів з кнопками Увійти та Реєстрація. Навігація має фіксовану позицію при прокручуванні сторінки, що забезпечує постійний доступ до основних функцій незалежно від позиції користувача на сторінці. Колірна схема навігації використовує білий фон з чорним текстом для нейтральності та професійного вигляду, зелений акцентний колір для кнопки Реєстрація виділяє основний call-to-action.

Неро-секція займає весь viewport і містить крупний заголовок "Знайдіть найкращого майстра для вашого дому", підзаголовок що пояснює концепцію платформи, центральну пошукову форму з плейсхолдером "Шукати майстра, послугу..." та зеленою кнопкою пошуку, три картки зі статистикою платформи розміщені горизонтально. На рис. 4.1 представлено головну сторінку з пошуковою формою.

Статистичні картки демонструють масштаб платформи через три ключові показники: "500+ Перевіраних майстрів", "2000+ Виконаних замовлень", "4.8 Середня оцінка". Кожна картка має білий фон, тінь для візуального відділення, числовий індикатор крупним шрифтом з кольоровим акцентом (зелений, синій, помаранчевий відповідно), текстовий опис під числом. Такий підхід візуалізації

метрик підвищує довіру користувачів до платформи через демонстрацію соціального доказу.

Секція популярних категорій розташована після hero-блоку та представляє шість основних напрямків послуг. Кожна категорія відображається карткою з іконкою послуги (сантехніка, електрика, прибирання, ремонт техніки, косметичний ремонт, переїзд), назвою категорії як заголовком, коротким описом типових послуг категорії. Картки організовані у responsive grid — три в ряд на десктопі, дві на планшеті, одна на мобільному. Під сіткою категорій розміщена кнопка "Переглянути всіх майстрів" для переходу до повного каталогу.

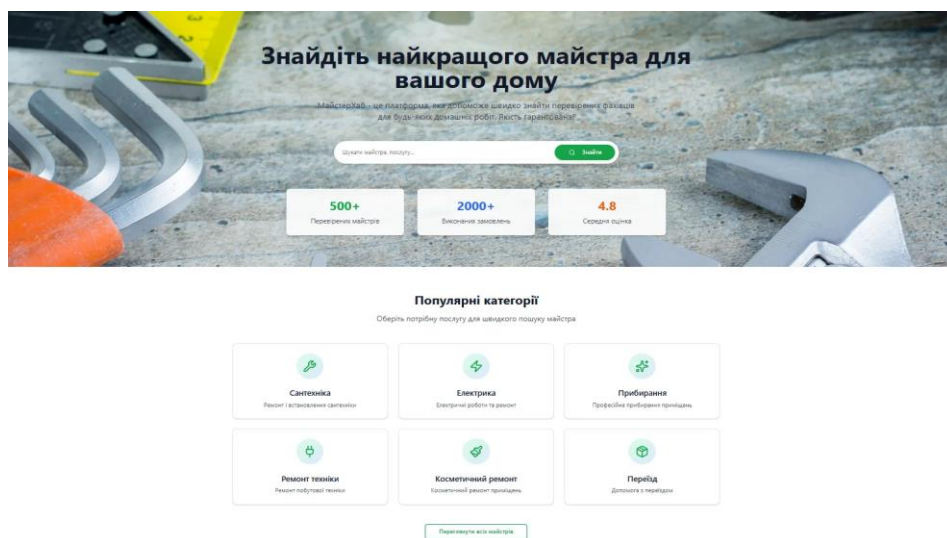


Рисунок 4.1 — Головна сторінка з пошуковою формою

4.2 Система автентифікації

4.2.1 Форма входу в систему

Автентифікація реалізована через дві форми — входу та реєстрації, організовані як окремі сторінки з центрованим layout. Цей підхід фокусує увагу користувача на процесі автентифікації та мінімізує відволікаючі елементи.

Форма входу містить логотип та назву платформи вгорі для підтвердження контексту, заголовок "Увійти в акаунт" з підзаголовком "Введіть ваші дані для входу в систему", поле електронної пошти з валідацією формату email, поле пароля з можливістю показати/приховати символи, чекбокс "Запам'ятати мене" для зручності повторних входів, посилання "Забули пароль?" для відновлення доступу, основну зелену кнопку "Увійти" на всю ширину форми, посилання "Немає акаунта? Зареєструватися" для переходу до реєстрації. На рис. 4.2 зображено інтерфейс форми входу.

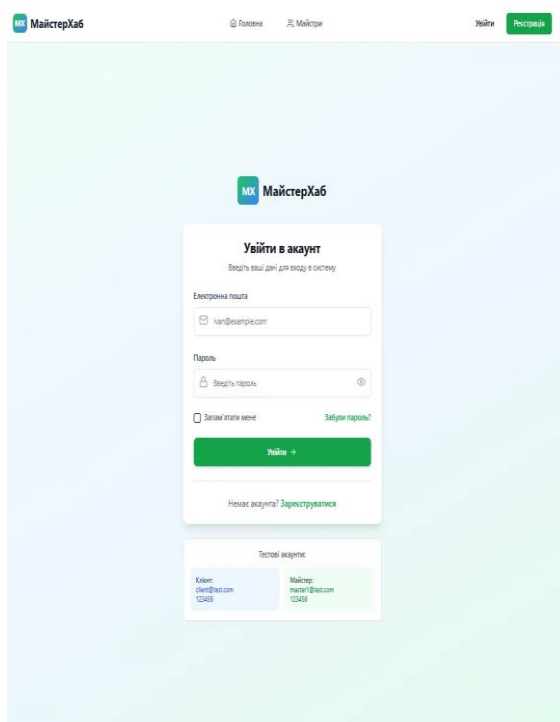


Рисунок 4.2 — Форма входу в систему

Валідація полів відбувається на двох рівнях. Client-side валідація виконується при втраті фокусу з поля (onBlur) та перевіряє формат email через регулярний вираз, мінімальну довжину пароля 6 символів, обов'язковість заповнення полів. Помилки відображаються червоним текстом під відповідним полем з конкретним поясненням проблеми. Server-side валідація перевіряє існування користувача з вказаним email, коректність пароля через порівняння bcrypt-хешів, статус акаунта

(активний/заблокований). Помилки сервера показуються загальним повідомленням над формою.

4.2.2 Форма реєстрації користувачів

Реєстрація організована у два етапи для зручності користувача та забезпечення якості даних. Перший крок спільний для всіх користувачів та включає вибір типу акаунта через дві великі картки Клієнт (шукаю майстрів) та Майстер (надаю послуги), поля ім'я та прізвище з валідацією мінімум 2 символи, поле телефону з маскою +380XXXXXXXXX для українських номерів, поле електронної пошти з перевіркою унікальності, поля пароля та підтвердження пароля з індикатором складності. Рис. 4.3 демонструє перший крок реєстрації для ролі Клієнт.

The screenshot shows the 'MasterHub' (МХ МайстерХаб) registration page. The header includes the logo, navigation links for 'Home' (Головна) and 'Masters' (Майстри), and buttons for 'Logout' (Вийти) and 'Registration' (Реєстрація). The main content area features a 'Create Account' (Створити акаунт) form with the subtitle 'Fill out the form for registration in the system' (Заповніть форму для реєстрації в системі). The form starts with a 'Select account type' (Виберіть тип акаунта) section, where the 'Client' (Клієнт) option, labeled 'I am looking for masters' (Шукаю майстрів), is selected. Below this are input fields for 'Name and surname' (Ім'я та прізвище) with the value 'Іван Петренко', 'Phone' (Телефон) with the value '+380501234567', and 'Email' (Електронна пошта) with the value 'ivan@example.com'. The password section includes 'Password' (Пароль) and 'Confirm password' (Підтвердити пароль) fields, both with a 'Show/Hide' (Відкрити/Закрити) icon. A green 'Register' (Зареєструватися) button is at the bottom of the form. A link 'Already have an account? Logout' (Вже є акаунт? Вийти) is located below the button.

Рисунок 4.3 — Форма реєстрації клієнта

Другий крок реєстрації відрізняється залежно від обраної ролі. Для клієнтів другий крок відсутній — реєстрація завершується після першого кроку з автоматичним створенням базового профілю. Для майстрів з'являється додатковий екран з розширеними полями, що показано на рис. 4.4.

Додаткові поля для майстрів включають множинний вибір категорій послуг через чекбокси (сантехніка, електрика, прибирання, ремонт техніки, косметичний ремонт, переїзд) з валідацією мінімум одна категорія, поле погодинної ставки в гривнях з валідацією додатного числа, текстову область для опису досвіду та навичок з обмеженням 1000 символів та лічильником символів. Кнопка "Зареєструватися" стає активною лише після коректного заповнення всіх обов'язкових полів.

The screenshot shows the 'Створити акаунт' (Create Account) form for a 'Майстер' (Master) on the 'МайстерХаб' website. The form is titled 'Створити акаунт' with a subtitle 'Заповніть форму для реєстрації в системі'. It starts with a 'Вибір типу акаунта' (Select account type) section, where 'Майстер' (Master) is selected. The form then asks for 'Ім'я та прізвище' (Name and surname), 'Телефон' (Phone), 'Електронна пошта' (Email), 'Пароль' (Password), and 'Підтвердити пароль' (Confirm password). Below these is the 'Інформація про майстра' (Master information) section, which includes a 'Вибір категорій послуг' (Select service categories) section with checkboxes for 'Сантехніка', 'Електрика', 'Прибирання', 'Ремонт техніки', 'Косметичний ремонт', and 'Переїзд'. It also has a 'Погодинна ставка (гр)' (Hourly rate (UAH)) field and an 'Опис послуг' (Service description) text area. A green 'Зареєструватися' (Register) button is at the bottom, and a link 'Вже є акаунт? Увійти' (Already have an account? Log in) is at the bottom right.

Рисунок 4.4 — Розширена форма реєстрації майстра

4.3 Каталог майстрів та система фільтрації

4.3.1 Панель фільтрації

Каталог майстрів є центральною функціональністю платформи, що дозволяє користувачам знаходити відповідних виконавців за різними критеріями. Інтерфейс організований у класичному layout з бічною панеллю фільтрів та основною областю з результатами пошуку.

Ліва бічна панель містить набір фільтрів для точного пошуку майстрів. Текстове поле пошуку з іконкою лупи дозволяє шукати за іменем майстра або ключовими словами в описі. Блок фільтрації за категоріями представлений списком чекбоксів для кожної категорії послуг з лічильником доступних майстрів у дужках, можливістю вибору декількох категорій одночасно, автоматичним оновленням результатів при зміні вибору. Рис. 4.5 показує каталог з активним фільтром за категорією Сантехніка.

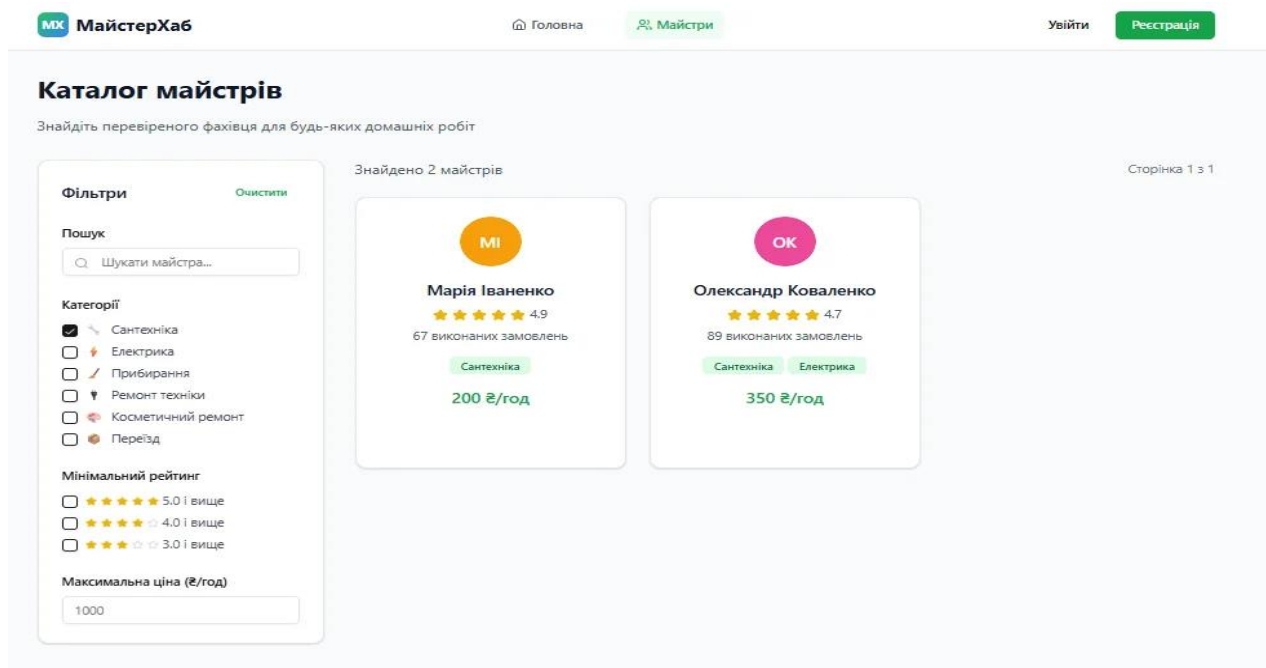


Рисунок 4.5 — Каталог майстрів з фільтром за категорією Сантехніка

Блок фільтрації за рейтингом містить набір чекбоксів для діапазонів рейтингу з візуалізацією зірок (5.0 і вище, 4.0 і вище, 3.0 і вище), позначкою кількості майстрів у кожному діапазоні. Блок фільтрації за ціною реалізований через поле вводу максимальної ставки в гривнях на годину з валідацією лише числових значень, оновленням результатів при втраті фокусу. Кнопка "Очистити" внизу панелі дозволяє скинути всі фільтри одним кліком.

4.3.2 Відображення результатів пошуку

Основна область каталогу відображає результати у форматі grid з картками майстрів. Кожна картка містить аватар з ініціалами майстра на кольоровому фоні, повне ім'я майстра як заголовок, візуалізацію рейтингу зірками з числовим значенням, текст кількості виконаних замовлень, набір бейджів з категоріями послуг, погодинну ставку виділену зеленим кольором. Наведення курсора на картку додає тінь та легке збільшення для feedback користувачу. Клік по картці переходить до детального профілю майстра.

Вгорі над сіткою карток розміщений рядок з інформацією про кількість знайдених майстрів ("Знайдено X майстрів") та випадаючим списком сортування з опціями за рейтингом (за замовчуванням), за кількістю замовлень, за ціною (зростання), за ціною (спадання). Результати організовані з пагінацією по 12 карток на сторінку з навігацією внизу. Якщо результатів немає, відображається повідомлення "Майстрів не знайдено" з пропозицією змінити фільтри.

На рис. 4.6 показано результати фільтрації за категорією Косметичний ремонт, демонструючи як система адаптує відображення до вибраних критеріїв.

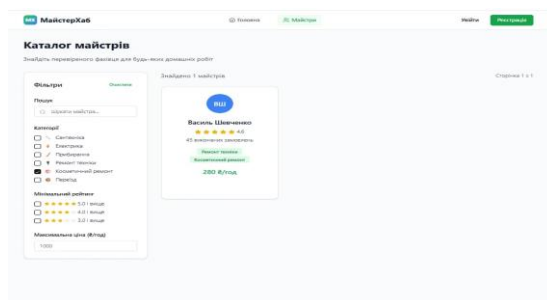


Рисунок 4.6 — Результати фільтрації за категорією Косметичний ремонт

4.4 Профіль майстра та створення замовлення

4.4.1 Інформаційна секція профілю

Сторінка профілю майстра надає детальну інформацію про виконавця та інструменти для створення замовлення. Layout організований у дві колонки на десктопі — ліва з основною інформацією та права з блоками контактів і дій.

Верхня частина профілю містить великий аватар з ініціалами, ім'я майстра великим шрифтом, рядок з рейтингом зірками, числовим значенням та кількістю відгуків у дужках, бейджі з категоріями послуг що надає майстер, погодинну ставку з позначкою "€/год". Секція "Про майстра" розміщена під header-блоком та містить детальний текстовий опис досвіду, кваліфікації, підходів до роботи майстра, статистику виконаних замовлень.

Секція відгуків відображає список оцінок від клієнтів. Кожен відгук містить аватар та ім'я клієнта, рейтинг у зірках, дату залишення відгуку, текст коментаря. Відгуки сортуються за датою (найновіші зверху) та відображаються з пагінацією по 5 на сторінку. Якщо відгуків немає, показується повідомлення "Поки що немає відгуків".

4.4.2 Інтерфейс створення замовлення

Права колонка профілю містить блок дій з кнопкою "Замовити послугу" та кнопкою "Переглянути контакти" (доступна після створення замовлення). Клік по "Замовити послугу" відкриває модальне вікно з формою створення замовлення, як показано на рис. 4.7.

Форма замовлення містить текстову область для опису робіт з плейсхолдером що підказує включити деталі, field date picker для вибору бажаної дати виконання, числове поле для вказівки орієнтовної кількості годин, автоматичний розрахунок орієнтовної вартості (годин $\times$ ставка майстра) з відображенням під полями, дві кнопки Скасувати (сіра) та Надіслати замовлення (зелена) внизу форми. Форма валідується перед відправкою — опис мінімум 10 символів, дата не раніше поточної, кількість годин додатне число.

Рисунок 4.7 — Форма створення замовлення

Після успішного створення замовлення модальне вікно закривається, показується toast-сповіщення "Замовлення успішно надіслано", користувач перенаправляється на сторінку "Мої замовлення" де може відстежувати статус. Для неавторизованих користувачів спроба створити замовлення відображає prompt з пропозицією увійти або зареєструватися.

4.5 Управління замовленнями

4.5.1 Інтерфейс клієнта

Сторінка управління замовленнями надає користувачам інструменти для відстеження статусів заявок та взаємодії з ними. Інтерфейс адаптується залежно від ролі користувача — клієнти бачать створені ними замовлення, майстри бачать отримані заявки.

Для клієнтів сторінка "Мої замовлення" відображає список всіх створених заявок з можливістю фільтрації за статусом. Вгорі розміщені табуляції для швидкого фільтрування: Активні (pending, accepted, in_progress), Завершені (completed), Скасовані (cancelled). Кожне замовлення представлено карткою що містить аватар та ім'я майстра як посилання на профіль, статус замовлення з відповідним кольоровим бейджем (жовтий для pending, синій для accepted, зелений для in_progress, сірий для completed), дату створення та бажану дату виконання, текст опису робіт (перші 100 символів), орієнтовну вартість та тривалість, кнопки дій залежно від статусу.

Можливі дії клієнта включають перегляд деталей замовлення через клік по картці, скасування замовлення зі статусом pending через кнопку "Скасувати", контакт з майстром для замовлень accepted або in_progress, залишення відгуку після completed статусу через кнопку "Залишити відгук". Якщо замовлень немає, відображається empty state з ілюстрацією та текстом "Немає активних замовлень" з кнопкою "Знайти майстра".

4.5.2 Інтерфейс майстра

Майстри мають доступ до панелі управління через окрему сторінку "Замовлення". Інтерфейс організований схожим чином до клієнтського, але з іншим набором дій. Картки замовлень для майстрів показують аватар та ім'я клієнта, дату створення заявки, детальний опис робіт повністю, запропоновану тривалість, статус обробки заявки. Для замовлень зі статусом pending майстер бачить дві кнопки: "Прийняти" (зелена) для переведення у accepted та "Відхилити" (червона) для відмови. Для accepted замовлень доступна кнопка "Почати роботу" для зміни статусу на in_progress та розкриття контактів клієнта. Для in_progress замовлень показується кнопка "Завершити" для переведення у completed після виконання робіт.

На рис. 4.8 представлено інтерфейс управління замовленнями, що демонструє empty state для нового клієнта без активних заявок.

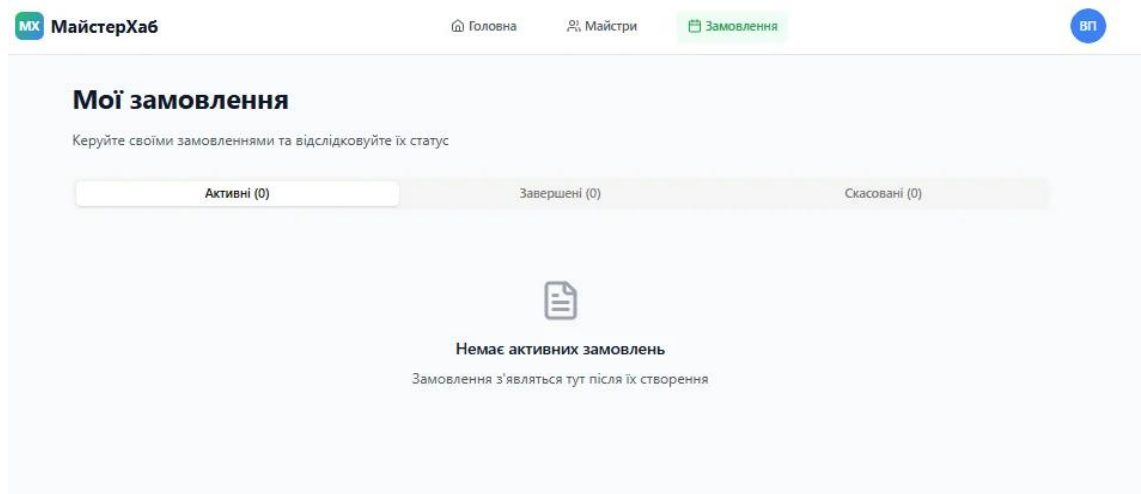


Рисунок 4.8 — Сторінка управління замовленнями (empty state)

4.6 Реалізація адаптивного дизайну

Інтерфейс платформи спроектовано з урахуванням принципів адаптивного дизайну для забезпечення коректного відображення на пристроях з різними розмірами екранів. Використано mobile-first підхід з progressive enhancement для великих екранів.

Визначено чотири основні breakpoints для адаптації layout відповідно до популярних розмірів пристроїв:

- 320px-767px (мобільні телефони) — одноколонкове компонування, навігація згортається у бургер-меню, фільтри доступні через модальне вікно, сітка карток майстрів в одну колонку, форми займають всю ширину екрану;
- 768px-1023px (планшети) — двоколонкове компонування для основного вмісту, навігація залишається горизонтальною, сітка карток у дві колонки, бічна панель фільтрів відображається збоку;
- 1024px-1439px (малі комп'ютери та ноутбуки) — триколонкова сітка для карток майстрів, повна бічна панель з фільтрами, всі елементи навігації видимі;

— 1440px+ (великі монітори) — максимальна ширина контейнера 1320px з центруванням, чотири колонки для сітки де доцільно, збільшені відступи між елементами.

Адаптивна типографіка використовує відносні одиниці (rem) для масштабування тексту. Базовий розмір шрифту становить 14px на мобільних та 16px на десктопі. Заголовки масштабуються пропорційно — H1 від 28px до 36px, H2 від 22px до 28px, H3 від 18px до 22px. Висота рядка (line-height) адаптується для кращої читабельності на різних пристроях — 1.4 на мобільних для економії простору, 1.6 на десктопі для комфорту читання довгих текстів.

Touch-friendly елементи на мобільних пристроях мають мінімальну висоту 44px відповідно до рекомендацій Apple Human Interface Guidelines. Відступи між інтерактивними елементами збільшені до мінімум 8px для запобігання випадковим натисканням. Hover-ефекти замінюються на touch feedback через зміну opacity при натисканні.

4.7 Система сповіщень та зворотний зв'язок

Зворотний зв'язок з користувачем реалізовано через систему спливаючих сповіщень, що з'являються у правому верхньому куті екрану. Сповіщення автоматично зникають через 5 секунд або закриваються вручну через іконку хрестика. Використовуються чотири типи сповіщень з відповідним кольоровим кодуванням: успішні (зелений колір) для успішних операцій на кшталт "Замовлення створено", помилки (червоний колір) для помилок на кшталт "Не вдалося завантажити дані", попередження (жовтий колір) для попереджень на кшталт "Заповніть всі обов'язкові поля", інформаційні (синій колір) для інформаційних повідомлень на кшталт "Новий відгук від клієнта".

Вбудована перевірка форм надає миттєвий зворотний зв'язок при заповненні. Помилки відображаються червоним текстом під полем одразу при втраті фокусу.

Успішно заповнені поля отримують зелену рамку та іконку галочки праворуч. Під час відправлення форми кнопка надсилання показує індикатор завантаження (обертний індикатор) та стає неактивною для запобігання повторним відправленням.

Стани завантаження відображаються каркасними екранами під час завантаження даних. Замість порожньої сторінки користувач бачить приблизну структуру вмісту з анімованими блоками-заповнювачами. Це створює враження швидшої роботи системи та зменшує когнітивне навантаження. Для тривалих операцій (понад 2 секунди) показується повноекранний індикатор завантаження з логотипом та текстом "Завантаження...".

4.8 Висновки до розділу 4

У четвертому розділі здійснено детальне проєктування користувацького інтерфейсу веб-платформи МайстерХаб з акцентом на зручність взаємодії та інтуїтивність навігації.

Розроблена структура головної сторінки ефективно представляє сервіс через hero-секцію з великим заголовком та центральною пошуковою формою, що одразу фокусує увагу на основній функції платформи. Статистичні картки з кількісними показниками (понад 500 майстрів, 2000 замовлень, середня оцінка 4.8) створюють соціальний доказ та підвищують довіру нових користувачів. Секція популярних категорій з іконками та описами надає швидкий доступ до найбільш затребуваних послуг. Навігаційна панель з фіксованою позицією забезпечує постійний доступ до ключових розділів незалежно від позиції прокручування сторінки.

Система автентифікації спроектована з урахуванням різних потреб користувачів через розділення на форми входу та реєстрації з відповідними варіаціями для клієнтів та майстрів. Форма входу містить мінімально необхідний набір полів (email, пароль) з додатковими опціями запам'ятовування та відновлення

пароллю. Процес реєстрації організований у два етапи, де перший крок спільний для всіх ролей і збирає базову інформацію, а другий крок для майстрів розширюється додатковими полями категорій послуг, погодинної ставки та опису досвіду. Багаторівнева валідація на стороні клієнта та сервера з чіткими повідомленнями про помилки допомагає користувачам коректно заповнювати форми та мінімізує фрустрацію від невдалих спроб реєстрації.

Каталог майстрів реалізовано з потужною системою фільтрації через бічну панель з чекбоксами категорій, діапазонами рейтингів та полем максимальної ціни, що дозволяє користувачам точно налаштовувати критерії пошуку. Результати відображаються у grid-layout з картками майстрів що містять всю ключову інформацію для первинної оцінки — аватар з ініціалами, рейтинг з кількістю відгуків, категорії послуг через бейджі, погодинну ставку. Сортування результатів за рейтингом, кількістю замовлень або ціною надає гнучкість у пріоритизації критеріїв вибору. Пагінація по 12 карток на сторінку забезпечує баланс між обсягом інформації та швидкістю завантаження.

Сторінка профілю майстра надає вичерпну інформацію для прийняття рішення про замовлення через детальний опис досвіду у текстовій секції "Про майстра", візуалізацію статистики виконаних робіт, список відгуків клієнтів з оцінками та коментарями. Інтерфейс створення замовлення через модальне вікно мінімізує кількість кроків до конверсії — користувач може описати роботи, вказати бажану дату та тривалість без покидання профілю майстра. Автоматичний розрахунок орієнтовної вартості на основі годин та ставки майстра надає прозорість ціноутворення ще до відправки заявки.

Інтерфейси управління замовленнями адаптовані під різні ролі користувачів через відповідні набори дій та інформації. Клієнти можуть відстежувати статуси своїх заявок через кольорові бейджі (жовтий для очікування, синій для прийнятих, зелений для в процесі, сірий для завершених), скасовувати замовлення що ще не прийняті майстрами, контактувати з виконавцями після прийняття заявок, залишати відгуки після завершення робіт. Майстри отримують структурований список вхідних заявок з можливістю прийняття або відхилення, зміни статусу

замовлення відповідно до етапів виконання, доступу до контактної інформації клієнтів після прийняття замовлення.

Адаптивний дизайн реалізовано через чотири контрольні точки, що відповідають типовим розмірам пристроїв від мобільних телефонів (320px) до великих моніторів (1440px+). Підхід "спочатку мобільні" забезпечує оптимальний досвід на малих екранах через одноколонкове компонування, бургер-меню для навігації, модальні вікна для фільтрів замість бічної панелі. Поступове розширення функціональності на більших екранах додає додаткові колонки у сітці (від однієї на мобільних до чотирьох на комп'ютерах), розгорнуту навігацію та фільтри, збільшені відступи між елементами. Зручні для дотику розміри інтерактивних елементів (мінімум 44px висота) та збільшені відступи між ними (мінімум 8px) запобігають випадковим натисканням на сенсорних екранах.

Система сповіщень та зворотного зв'язку забезпечує постійну комунікацію з користувачем про результати дій через спливаючі повідомлення з кольоровим кодуванням (зелений для успіху, червоний для помилок, жовтий для попереджень, синій для інформації). Вбудована перевірка форм з миттєвим відображенням помилок червоним текстом та підтвердженням коректності зеленою рамкою допомагає користувачам заповнювати дані правильно з першої спроби. Стани завантаження через каркасні екрани під час завантаження даних створюють враження швидшої роботи системи та зменшують тривожність від очікування. Індикатори завантаження на кнопках надсилення запобігають повторним відправленням форм та чітко сигналізують про процес обробки запиту.

Спроектований користувацький інтерфейс платформи МайстерХаб відповідає сучасним стандартам веб-дизайну та принципам користувацького досвіду через інтуїтивну навігацію, зрозумілу ієрархію інформації, адаптивність до різних пристроїв. Використання визнаних шаблонів інтерфейсу (сіткове компонування для каталогів, модальні вікна для форм, спливаючі повідомлення для сповіщень) зменшує криву навчання для нових користувачів. Кольорове кодування статусів та типів повідомлень використовує універсальні асоціації (зелений для успіху, червоний для помилок), що робить інтерфейс зрозумілим інтуїтивно.

Адаптивний дизайн забезпечує однаково якісний досвід незалежно від пристрою доступу — від мобільних телефонів до великих комп'ютерних моніторів.

5 АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

5.1 Вимоги до серверного обладнання

5.1.1 Мінімальна конфігурація для тестового середовища

Успішне функціонування веб-платформи МайстерХаб залежить не лише від якості програмного забезпечення, але й від належного апаратного оснащення серверної інфраструктури. Правильний підбір апаратних компонентів забезпечує необхідну продуктивність системи, стабільність роботи під навантаженням, масштабованість для зростання кількості користувачів. Цей розділ присвячено визначенню мінімальних та рекомендованих апаратних вимог для розгортання платформи на різних етапах її життєвого циклу.

Серверна частина платформи МайстерХаб складається з трьох основних компонентів, кожен з яких має специфічні вимоги до апаратних ресурсів. Веб-сервер на Node.js з Express відповідає за обробку HTTP-запитів, виконання бізнес-логіки, взаємодію з базою даних. База даних MongoDB зберігає всі дані платформи та забезпечує швидкий доступ до них через індексовані запити. Реверсний проксі-сервер Nginx виконує функції балансування навантаження, термінації SSL-з'єднань, кешування статичного контенту.

Тестове середовище призначене для розробки, налагодження та проведення функціонального тестування системи за умови обмеженої кількості одночасних користувачів (до 50 осіб). Ця конфігурація є мінімально достатньою для запуску всіх компонентів системи на одному фізичному або віртуальному сервері.

Процесор повинен мати мінімум 2 фізичні ядра з тактовою частотою від 2.0 ГГц. Node.js є однопотоким середовищем виконання, проте використовує асинхронну модель обробки запитів, що дозволяє ефективно використовувати один

процесорний потік для обслуговування багатьох одночасних з'єднань. Друге ядро необхідне для роботи MongoDB та системних процесів. Дослідження продуктивності Node.js показують, що одне ядро здатне обробляти до 1000 простих HTTP-запитів на секунду за умови оптимізованого коду.

Таблиця 5.1 — Мінімальна апаратна конфігурація для тестування

Компонент	Мінімальні вимоги	Призначення
Процесор	2 ядра, 2.0 ГГц	Обробка запитів Node.js та MongoDB
Оперативна пам'ять	4 ГБ RAM	Node.js (1 ГБ), MongoDB (2 ГБ), ОС та кеш (1 ГБ)
Дисковий простір	50 ГБ SSD	ОС (15 ГБ), програмне забезпечення (10 ГБ), дані (25 ГБ)
Мережевий інтерфейс	100 Мбіт/с	Підключення до локальної мережі або Інтернет

Оперативна пам'ять обсягом 4 ГБ розподіляється між компонентами системи наступним чином. Node.js процес з Express сервером споживає близько 1 ГБ пам'яті у стані спокою та може зростати до 1.5 ГБ під навантаженням залежно від кількості активних з'єднань та кешованих даних. MongoDB резервує 2 ГБ для власних потреб, з яких 1 ГБ використовується для WiredTiger кешу (механізм зберігання даних), 512 МБ для журналу транзакцій, 512 МБ для операційної пам'яті процесів. Операційна система Linux та супутні процеси (Nginx, системні служби) займають близько 1 ГБ. Документація MongoDB рекомендує мінімум 2 ГБ оперативної пам'яті для продуктивної роботи з невеликими базами даних.

Дисковий простір 50 ГБ на SSD накопичувачі є мінімально достатнім для тестового середовища з обмеженим обсягом даних. Операційна система Ubuntu Server 22.04 LTS займає близько 15 ГБ після встановлення всіх необхідних пакетів. Програмне забезпечення (Node.js, MongoDB, Nginx, npm-пакети) потребує додатково 10 ГБ. Решта 25 ГБ виділяється для збереження даних користувачів, що достатньо для приблизно 5000 профілів майстрів з повною інформацією та 20000 замовлень з відгуками. Використання SSD накопичувача є критичним для продуктивності MongoDB, оскільки забезпечує швидкість випадкового читання понад 50000 IOPS проти 100-200 IOPS у традиційних жорстких дисків.

Мережевий інтерфейс швидкістю 100 Мбіт/с забезпечує пропускну здатність до 12.5 МБ/с, що достатньо для обслуговування 50 одночасних користувачів з середнім розміром сторінки 250 КБ (включно з JavaScript, CSS, зображеннями) при часі завантаження до 2 секунд на користувача. Для тестового середовища в локальній мережі така швидкість є більш ніж достатньою.

5.1.2 Рекомендована конфігурація для продуктивного середовища

Продуктивне середовище призначене для реальної експлуатації платформи з очікуваним навантаженням до 500 одночасних користувачів та перспективою масштабування до 2000 користувачів. Ця конфігурація забезпечує достатній запас продуктивності для комфортної роботи системи під пікові навантаження та можливості горизонтального масштабування.

Процесор з 4 фізичними ядрами та тактовою частотою 3.0 ГГц забезпечує достатні обчислювальні ресурси для паралельної обробки запитів. Node.js може бути налаштований на використання всіх доступних ядер через механізм cluster mode, коли створюється окремий worker-процес для кожного ядра. MongoDB ефективно використовує багатоядерність для паралелізації складних запитів та індексування. Дослідження масштабованості Node.js демонструють лінійне зростання продуктивності до 4 ядер, після чого приріст сповільнюється через overhead координації між процесами.

Таблиця 5.2 — Рекомендована конфігурація для продуктивного середовища

Компонент	Рекомендовані вимоги	Обґрунтування
Процесор	4 ядра, 3.0 ГГц	Кластерний режим Node.js, паралельні запити MongoDB
Оперативна пам'ять	16 ГБ RAM	Node.js кластер (4 ГБ), MongoDB з великим кешем (10 ГБ), ОС (2 ГБ)
Дисковий простір	250 ГБ NVMe SSD	Зберігання 50000+ профілів, 200000+ замовлень, логів та резервних копій
Мережевий інтерфейс	1 Гбіт/с	Обслуговування 500+ одночасних користувачів без затримок

Оперативна пам'ять збільшена до 16 ГБ для забезпечення достатнього буферу під навантаженням. Node.js кластер з чотирма worker-процесами споживає до 4 ГБ (1 ГБ на worker). MongoDB отримує 10 ГБ, що дозволяє тримати в пам'яті робочий набір даних (working set) для швидкого доступу без звернень до диску. Дослідження продуктивності MongoDB показують, що розмір WiredTiger кешу безпосередньо впливає на швидкість виконання запитів — збільшення кешу з 2 ГБ до 10 ГБ може прискорити читання даних у 5-7 разів для баз середнього розміру. Операційна система та допоміжні процеси використовують 2 ГБ.

Дисковий простір 250 ГБ на NVMe SSD накопичувачі забезпечує достатній запас для зростання бази даних. NVMe (Non-Volatile Memory Express) протокол надає значно вищу продуктивність порівняно з SATA SSD — швидкість послідовного читання до 3500 МБ/с проти 550 МБ/с, випадкового читання понад 500000 IOPS проти 100000 IOPS. Розподіл простору передбачає 200 ГБ для даних

MongoDB (достатньо для 50000 профілів майстрів з повною історією, 200000 замовлень, 100000 відгуків), 30 ГБ для логів додатку та бази даних з ротацією кожні 30 днів, 20 ГБ для операційної системи та програмного забезпечення.

Мережевий інтерфейс 1 Гбіт/с забезпечує пропускну здатність 125 МБ/с, що дозволяє обслуговувати 500 одночасних користувачів з середнім розміром сторінки 250 КБ при часі завантаження менше 2 секунд. Реальна пропускну здатність з урахуванням overhead TCP/IP протоколів становить близько 940 Мбіт/с або 117.5 МБ/с. При піковому навантаженні 500 користувачів з середньою швидкістю запитів 2 запити/секунду на користувача система обробляє 1000 запитів/секунду, що при середньому розмірі відповіді 50 КБ дає навантаження на мережу близько 50 МБ/с — менше половини доступної пропускну здатності.

5.2 Вимоги до хостинг-платформи

5.2.1 Порівняння варіантів хостингу

Вибір хостинг-провайдера та типу хостингу суттєво впливає на продуктивність, надійність та вартість експлуатації платформи. Для веб-додатків на Node.js та MongoDB існує кілька варіантів розгортання, кожен з яких має свої переваги та обмеження.

Віртуальний приватний сервер (VPS) надає виділені обчислювальні ресурси на віртуалізованому апаратному забезпеченні. Провайдери як DigitalOcean, Linode, Vultr пропонують VPS з фіксованими конфігураціями за передбачувану місячну плату. Переваги VPS включають повний контроль над операційною системою та можливість встановлення будь-якого програмного забезпечення, передбачувану вартість незалежно від навантаження, простоту налаштування та управління для досвідчених розробників. Недоліки полягають у необхідності самостійного адміністрування сервера (оновлення безпеки, моніторинг, резервне копіювання),

обмеженій масштабованості — зміна конфігурації вимагає міграції на інший план, відсутності автоматичного масштабування під змінне навантаження.

Платформа як послуга (PaaS) надає готове середовище для розгортання додатків без необхідності управління інфраструктурою. Heroku, Railway, Render абстрагують деталі серверного управління та забезпечують автоматизоване розгортання через Git. Переваги PaaS охоплюють мінімальні зусилля на налаштування інфраструктури, автоматичне масштабування відповідно до навантаження, вбудоване резервне копіювання та моніторинг, інтеграцію з іншими сервісами (бази даних, кеш, черги). Недоліки включають вищу вартість порівняно з VPS особливо при зростанні навантаження, обмеження на налаштування середовища виконання, можливу прив'язку до специфічних технологій провайдера (vendor lock-in).

Контейнерна оркестрація через Kubernetes на хмарних платформах як AWS EKS, Google GKE, Azure AKS надає найбільшу гнучкість та масштабованість для складних розподілених систем. Переваги включають горизонтальне автомасштабування контейнерів під навантаженням, високу доступність через реплікацію та автоматичне відновлення, можливість мікросервісної архітектури з незалежним масштабуванням компонентів, переносимість між різними хмарними провайдерами. Недоліки полягають у високій складності налаштування та управління Kubernetes кластером, значних витратах на інфраструктуру навіть для невеликих додатків, необхідності спеціалізованих знань DevOps-інженерів.

Для початкового розгортання платформи МайстерХаб рекомендується використання VPS рішень з можливістю подальшої міграції на PaaS при зростанні навантаження. VPS надає оптимальний баланс між контролем, продуктивністю та вартістю для проєктів з прогнозованим навантаженням. Для тестового середовища достатнім є VPS найменшого тарифу з 2 ядрами та 4 ГБ пам'яті (приблизно 20-30 доларів на місяць у DigitalOcean або Hetzner). Для продуктивного середовища рекомендується VPS з 4 ядрами, 16 ГБ пам'яті та 250 ГБ SSD (приблизно 80-120 доларів на місяць).

5.2.2 Географічне розміщення серверів

Географічна близькість серверів до основної аудиторії користувачів суттєво впливає на швидкість завантаження сторінок та загальний користувацький досвід. Затримка мережі (latency) між клієнтом та сервером складається з часу проходження сигналу через фізичні канали зв'язку, обробки на проміжних маршрутизаторах, черги на обладнанні провайдерів.

Для платформи МайстерХаб, орієнтованої на український ринок, оптимальним є розміщення серверів у європейських дата-центрах з низькою затримкою до України. Франкфурт (Німеччина) є найбільшим європейським hub для міжнародного трафіку з типовою затримкою до Києва 35-45 мс. Більшість провайдерів VPS мають дата-центри у Франкфурті з надійним зв'язком та конкурентними цінами. Амстердам (Нідерланди) надає схожі показники затримки 40-50 мс до Києва та є альтернативною локацією з хорошою зв'язаністю. Варшава (Польща) географічно ближча до України з затримкою 20-30 мс, проте має меншу кількість провайдерів та потенційно вищі ціни. Київські дата-центри надають найменшу затримку для українських користувачів (5-10 мс), проте мають обмеження щодо міжнародної зв'язаності та потенційні ризики інфраструктури.

5.3 Вимоги до мережевої інфраструктури

5.3.1 Вимоги до пропускної здатності

Надійність та продуктивність мережевого з'єднання є критичними для функціонування веб-платформи. Окрім пропускної здатності каналу зв'язку, важливими характеристиками є стабільність з'єднання, час безвідмовної роботи (uptime), захист від DDoS-атак.

Пропускна здатність мережевого каналу визначає максимальну кількість даних, що можуть бути передані за одиницю часу. Для розрахунку необхідної

пропускної здатності враховується очікувана кількість одночасних користувачів, середній розмір сторінки, частота оновлення даних.

Типова сесія користувача на платформі МайстерХаб включає завантаження головної сторінки (HTML, CSS, JavaScript, зображення) розміром близько 800 КБ, перегляд каталогу майстрів з завантаженням 12 карток (JSON дані + аватари) розміром 300 КБ, відкриття профілю майстра з відгуками розміром 200 КБ, створення замовлення через форму розміром 50 КБ відповіді. Середня сесія передає близько 1.5 МБ даних від сервера до клієнта та 100 КБ від клієнта до сервера (форми, запити API).

При очікуваному навантаженні 500 одночасних користувачів з середньою тривалістю сесії 10 хвилин отримуємо 50 нових користувачів на хвилину або 0.83 користувача на секунду. Кожен новий користувач завантажує 1.5 МБ початкових даних, що дає базове навантаження 1.25 МБ/с або 10 Мбіт/с. Активні користувачі генерують додатково близько 50 КБ/хвилину трафіку через фонові запити оновлення даних, що для 500 користувачів становить 25 МБ/хвилину або 0.42 МБ/с (3.36 Мбіт/с). Сумарне середнє навантаження складає близько 13.5 Мбіт/с, проте пікове навантаження може сягати 50-70 Мбіт/с при одночасному переході багатьох користувачів між сторінками.

Мережевий канал 1 Гбіт/с (1000 Мбіт/с) забезпечує достатній запас пропускної здатності з коефіцієнтом резерву близько 15х для середнього навантаження та 5х для піків. Провайдери VPS зазвичай надають необмежений трафік при швидкості підключення 1 Гбіт/с, проте можуть застосовувати обмеження (throttling) при тривалому використанні понад 20-30% пропускної здатності.

5.3.2 Вимоги до доступності та надійності

Доступність системи (availability) вимірюється як відсоток часу, протягом якого сервіс є доступним для користувачів. Галузевий стандарт для веб-сервісів передбачає рівень доступності мінімум 99.9% (три дев'ятки), що дозволяє до 8.76 годин простою на рік або 43.2 хвилини на місяць. Для комерційних платформ рекомендується рівень 99.95% (8.76 годин простою на рік) або вище.

Надійність забезпечується через кілька механізмів. Резервування компонентів передбачає використання двох або більше серверів з балансуванням навантаження між ними — відмова одного сервера не призводить до повної недоступності сервісу. Автоматичне моніторингування здоров'я системи через Prometheus, Grafana або інші інструменти виявляє проблеми до того як вони вплинуть на користувачів. Автоматичне відновлення через systemd, PM2 або Kubernetes перезапускає процеси що завершилися з помилкою. Резервне копіювання бази даних з автоматичним створенням snapshot кожні 6 годин та збереженням копій за останні 7 днів дозволяє відновити дані при катастрофічних відмовах.

Більшість провайдерів VPS гарантують uptime 99.9% згідно з Service Level Agreement (SLA) та надають компенсацію за порушення цього показника. Hetzner гарантує 99.9% доступності мережі, DigitalOcean надає 99.99% uptime для окремих droplet з можливістю створення резервних копій та floating IP для швидкого переключення між серверами. AWS та Google Cloud пропонують 99.95-99.99% доступності для віртуальних машин залежно від конфігурації зон доступності.

5.4 Вимоги до клієнтських пристроїв

5.4.1 Підтримувані браузерери

Користувачі платформи МайстерХаб отримують доступ до сервісу через веб-браузери на різних пристроях. Оскільки клієнтська частина є односторінковим додатком (SPA) на React, вона виконується безпосередньо в браузері користувача та потребує достатніх обчислювальних ресурсів для плавної роботи.

Платформа розроблена з використанням сучасних веб-технологій (ES6+, CSS Grid, Flexbox) та підтримує браузерери випущені протягом останніх двох років. Google Chrome версії 90 та новіші є найбільш рекомендованим браузером завдяки найкращій підтримці сучасних стандартів, високій продуктивності JavaScript-

движка V8, інструментам розробника. Mozilla Firefox версії 88 та новіші надають відмінну підтримку веб-стандартів та приватності користувачів. Safari версії 14 та новіші для користувачів macOS та iOS підтримують всі необхідні функції проте мають деякі обмеження у реалізації окремих API. Microsoft Edge версії 90 та новіші на базі Chromium забезпечують сумісність аналогічну Chrome.

Застарілі браузерери як Internet Explorer 11 не підтримуються через відсутність реалізації критичних можливостей ES6 (стрілкові функції, Promise, async/await, деструктуризація). Статистика використання браузерів в Україні станом на 2024 рік показує що Chrome займає близько 65% ринку, Safari 15%, Firefox 8%, Edge 7%, інші браузерери 5%. Підтримка топ-4 браузерів охоплює 95% користувачів.

5.4.2 Мінімальні характеристики пристроїв

Для настільних комп'ютерів та ноутбуків мінімальні вимоги включають процесор Intel Core i3 або аналогічний з частотою від 2.0 ГГц, оперативну пам'ять мінімум 4 ГБ (2 ГБ для браузера, 2 ГБ для операційної системи), роздільну здатність екрану від 1280x720 пікселів для коректного відображення інтерфейсу. Рекомендовані характеристики передбачають процесор Intel Core i5 або новіший, 8 ГБ оперативної пам'яті для комфортної багатозадачності, роздільну здатність Full HD (1920x1080) або вищу для найкращого візуального досвіду.

Для мобільних пристроїв мінімальні вимоги охоплюють смартфони випущені після 2019 року з процесором Snapdragon 660 або Exynos 9611 і вище, оперативну пам'ять від 3 ГБ, роздільну здатність екрану від 720x1280 (HD). Рекомендовані характеристики включають процесори Snapdragon 730 або Apple A12 і новіші, 4 ГБ оперативної пам'яті або більше, роздільну здатність Full HD+ (1080x2340) для чіткості тексту та зображень.

React додаток після мінімізації та компресії займає близько 400 KB JavaScript-коду, 150 KB CSS-стилів, 250 KB зображень та іконок. Повне завантаження та ініціалізація додатку займає 2-3 секунди на середньому смартфоні при швидкості інтернету 4G (10-20 Мбіт/с). На застарілих пристроях з процесорами Snapdragon 450 або нижче час ініціалізації може зростати до 5-7 секунд через повільнішу обробку JavaScript.

5.4.3 Вимоги до інтернет-з'єднання

Мінімальна швидкість інтернет-з'єднання для комфортної роботи з платформою становить 2 Мбіт/с для завантаження та 1 Мбіт/с для відправлення даних. При такій швидкості початкове завантаження додатку займає близько 4-5 секунд, перехід між сторінками відбувається майже миттєво завдяки кешуванню статичних ресурсів, форми відправляються без помітних затримок. Рекомендована швидкість 5 Мбіт/с та вище забезпечує завантаження сторінок менше 2 секунд, комфортну роботу з зображеннями високої роздільності, можливість одночасної роботи кількох користувачів на одному з'єднанні.

Мобільні користувачі можуть працювати з платформою через 3G з'єднання (мінімум 384 кбіт/с) проте досвід буде суттєво гіршим через тривалі затримки завантаження. 4G LTE з типовою швидкістю 10-50 Мбіт/с забезпечує відмінний досвід аналогічний широкосмуговому інтернету. WiFi з'єднання рекомендується для початкового завантаження додатку при обмеженому мобільному трафіку, оскільки повне завантаження всіх ресурсів може споживати до 2 МБ трафіку.

5.5 Стратегія масштабування інфраструктури

Зростання кількості користувачів платформи вимагатиме поступового нарощування апаратних ресурсів. Стратегія масштабування передбачає три етапи відповідно до очікуваного навантаження.

Початковий етап (до 500 користувачів) використовує один VPS-сервер з конфігурацією 4 ядра, 16 ГБ RAM, 250 ГБ SSD. Всі компоненти системи (Node.js, MongoDB, Nginx) працюють на одному сервері. Вартість інфраструктури становить близько 80-120 доларів на місяць. Цього достатньо для перших 6-12 місяців роботи платформи при поступовому зростанні аудиторії.

Етап зростання (500-2000 користувачів) потребує розділення компонентів на окремі сервери. Перший сервер (4 ядра, 8 ГБ RAM, 100 ГБ SSD) виділяється під

Node.js додаток з Nginx, другий сервер (4 ядра, 16 ГБ RAM, 500 ГБ SSD) виділяється під MongoDB з оптимізованою конфігурацією для бази даних. Балансувальник навантаження розподіляє запити між кількома екземплярами Node.js додатку для підвищення доступності. Загальна вартість інфраструктури зростає до 150-200 доларів на місяць. Такої конфігурації достатньо для 1-2 років роботи при активному зростанні.

Етап масштабу (понад 2000 користувачів) вимагає переходу на розподілену архітектуру. Кілька серверів додатків (по 4 ядра, 8 ГБ RAM кожен) за балансувальником навантаження забезпечують горизонтальне масштабування обчислювальної потужності. MongoDB replica set з трьох серверів (primary + 2 secondary) гарантує високу доступність даних та можливість читання з реплік для зменшення навантаження. Redis кеш-сервер (2 ядра, 4 ГБ RAM) зберігає часто використовувані дані для зменшення запитів до MongoDB. CDN (Content Delivery Network) як Cloudflare або AWS CloudFront кешує статичні ресурси (зображення, JavaScript, CSS) географічно близько до користувачів. Вартість інфраструктури становить 400-600 доларів на місяць проте забезпечує обслуговування 5000-10000 одночасних користувачів.

5.6 Висновки до розділу 5

У п'ятому розділі визначено комплекс апаратних вимог для розгортання та експлуатації веб-платформи МайстерХаб на різних етапах її життєвого циклу.

Встановлено мінімальну конфігурацію серверного обладнання для тестового середовища, що включає процесор з 2 фізичними ядрами та тактовою частотою 2.0 ГГц, 4 ГБ оперативної пам'яті для одночасної роботи Node.js та MongoDB, 50 ГБ SSD накопичувача для операційної системи, програмного забезпечення та обмеженого обсягу даних, мережевий інтерфейс 100 Мбіт/с для тестування в локальній мережі. Така конфігурація забезпечує достатню продуктивність для

розробки, налагодження та функціонального тестування системи за умови обмеженої кількості одночасних користувачів до 50 осіб.

Визначено рекомендовану конфігурацію для продуктивного середовища, що передбачає процесор з 4 ядрами та частотою 3.0 ГГц для паралельної обробки запитів через кластерний режим Node.js, 16 ГБ оперативної пам'яті з розподілом 4 ГБ для кластера Node.js, 10 ГБ для MongoDB з великим кешем WiredTiger, 2 ГБ для операційної системи, 250 ГБ NVMe SSD накопичувача з високою швидкістю випадкового читання понад 500000 IOPS для продуктивної роботи бази даних, мережевий інтерфейс 1 Гбіт/с для обслуговування 500 одночасних користувачів без затримок. Обґрунтовано вибір SSD та NVMe технологій через критичний вплив швидкості дискових операцій на продуктивність MongoDB — NVMe надає перевагу у 5-10 разів порівняно з традиційними жорсткими дисками.

Проведено порівняльний аналіз варіантів хостингу з урахуванням переваг та недоліків кожного підходу. VPS рішення надають оптимальний баланс між контролем, продуктивністю та вартістю для проєктів з прогнозованим навантаженням через повний контроль над операційною системою, передбачувану місячну вартість 80-120 доларів для рекомендованої конфігурації, простоту налаштування для досвідчених розробників. PaaS платформи абстрагують деталі управління інфраструктурою та забезпечують автоматичне масштабування проте мають вищу вартість особливо при зростанні навантаження. Контейнерна оркестрація через Kubernetes надає найбільшу гнучкість проте вимагає спеціалізованих знань та має значні витрати навіть для невеликих додатків. Для початкового розгортання платформи МайстерХаб рекомендовано використання VPS з можливістю подальшої міграції на PaaS при суттєвому зростанні навантаження.

Обґрунтовано вибір географічного розміщення серверів з урахуванням затримки мережі до цільової аудиторії. Франкфурт визначено оптимальною локацією для платформи орієнтованої на український ринок через типову затримку 35-45 мс до Києва, наявність найбільшого європейського hub для міжнародного трафіку, присутність більшості провайдерів VPS з конкурентними цінами.

Амстердам та Варшава розглянуто як альтернативні локації з затримкою 40-50 мс та 20-30 мс відповідно. Показано що при розміщенні у Франкфурті загальна затримка завантаження сторінки становитиме 130-180 мс, що є цілком прийнятним показником для веб-додатків та сприймається користувачами як швидка відповідь системи.

Визначено вимоги до мережевої інфраструктури через розрахунок необхідної пропускної здатності на основі очікуваного навантаження. При 500 одночасних користувачах з середньою сесією 10 хвилин та розміром сторінки 1.5 МБ середнє навантаження становить 13.5 Мбіт/с з піковими значеннями до 70 Мбіт/с, що робить канал 1 Гбіт/с достатнім з коефіцієнтом резерву 15х для середнього навантаження. Встановлено вимоги до доступності системи на рівні мінімум 99.9% (до 43.2 хвилини простою на місяць) через механізми резервування компонентів, автоматичного моніторингу та відновлення, резервного копіювання бази даних кожні 6 годин. Більшість провайдерів VPS гарантують uptime 99.9% згідно з SLA та надають компенсацію за порушення цього показника.

Специфіковано вимоги до клієнтських пристроїв користувачів платформи. Підтримка сучасних браузерів Chrome 90+, Firefox 88+, Safari 14+, Edge 90+ охоплює 95% користувачів в Україні згідно зі статистикою використання. Мінімальні характеристики настільних комп'ютерів включають процесор Intel Core i3 або аналогічний, 4 ГБ оперативної пам'яті, роздільну здатність від 1280x720 пікселів. Мобільні пристрої випущені після 2019 року з процесорами Snapdragon 660 або вище, 3 ГБ пам'яті, екраном HD забезпечують прийнятний досвід роботи з платформою. React додаток займає близько 800 КБ після стиснення та завантажується за 2-3 секунди на середньому смартфоні при з'єднанні 4G. Мінімальна швидкість інтернету 2 Мбіт/с забезпечує комфортну роботу з затримками завантаження до 5 секунд, рекомендована швидкість 5 Мбіт/с гарантує завантаження менше 2 секунд.

Розроблено стратегію масштабування інфраструктури на трьох етапах відповідно до зростання навантаження. Початковий етап використовує один VPS-сервер з усіма компонентами для обслуговування до 500 користувачів за вартістю

80-120 доларів на місяць. Етап зростання передбачає розділення на окремі сервери для Node.js та MongoDB з балансуванням навантаження для обслуговування 500-2000 користувачів за вартістю 150-200 доларів на місяць. Етап масштабу вимагає розподіленої архітектури з кількома серверами додатків, MongoDB replica set, Redis кешем та CDN для обслуговування понад 2000 користувачів за вартістю 400-600 доларів на місяць. Показано переваги горизонтального масштабування через додавання серверів порівняно з вертикальним збільшенням ресурсів існуючих машин у контексті масштабованості та відмовостійкості.

Визначені апаратні вимоги створюють міцну основу для розгортання веб-платформи МайстерХаб з достатнім запасом продуктивності для комфортної роботи користувачів та перспективою масштабування при зростанні аудиторії. Рекомендовані конфігурації апаратного забезпечення є економічно ефективними та відповідають сучасним стандартам веб-хостингу для додатків середнього масштабу. Стратегія поетапного масштабування дозволяє оптимізувати витрати на інфраструктуру на ранніх етапах життєвого циклу продукту при збереженні можливості нарощування потужностей у міру необхідності.

ВИСНОВКИ

У дипломному проєкті виконано комплексне аналізування ринку побутових послуг в Україні та виявлено ключові проблеми наявних платформ: спрощені та легко маніпульовані рейтинги, відсутність перевірки достовірності відгуків, неможливість об'єктивно порівнювати майстрів між собою, а також обмежені інструменти контролю виконання робіт. Ці недоліки стали підґрунтям для формування основної мети проєкту — створення веб-платформи, яка забезпечує прозору, об'єктивну та захищену взаємодію між замовниками й виконавцями побутових послуг.

Одним з центральних результатів роботи стало проєктування багатофакторної системи рейтингування, яка принципово відрізняється від традиційних середніх арифметичних оцінок, що домінують на сучасних сервісах. У платформі МайстерХаб рейтинг формується на основі кількох груп параметрів: кількість і розподіл відгуків у часі, стабільність роботи виконавця, рівень кваліфікації, швидкість відповіді, показники успішно виконаних замовлень, наявність верифікації, а також виявлення аномальних або підозрілих відгуків. Такий підхід дозволяє мінімізувати вплив випадкових оцінок, покращує точність відображення реальної репутації та унеможливорює маніпуляцію рейтингами через одноразові чи фейкові відгуки.

Розроблена модель даних передбачає окрему колекцію для відгуків, зберігання часових міток, ідентифікаторів рецензентів та параметрів поведінкової активності, що дозволяє формувати багатофакторні рейтинги на основі реальних та перевірених взаємодій. Запропонована система також підтримує автоматичне оновлення рейтингу після завершення замовлень і після додавання відгуків, що забезпечує актуальність показників під час пошуку виконавців.

Для забезпечення ефективної роботи багатофакторної системи у проєкті створено відповідну архітектурну та технологічну основу. Використання Node.js + Express спрощує реалізацію обчислювальних модулів рейтингування, тоді як

MongoDB забезпечує гнучке зберігання гетерогенних наборів даних і можливість швидкого доповнення моделі новими факторами. Реалізація REST API дає змогу забезпечити прозору взаємодію між клієнтом і сервером, а інтеграція JWT гарантує захищений доступ до персоналізованих даних користувачиских профілів.

Інтерфейс користувача було спроектовано таким чином, щоб клієнти могли легко порівнювати майстрів за ключовими показниками: рейтингом, досвідом, часом відповіді та рівнем верифікації. Візуальна структура каталогу майстрів, розширені фільтри та інформативні картки профілів дозволяють максимально ефективно використовувати всю глибину багатofакторної рейтингової моделі.

Отже, у межах роботи створено цілісну технічну концепцію платформи, яка усуває ключові недоліки існуючих сервісів та пропонує сучасний механізм формування довіри між замовниками і виконавцями. Багатofакторна система рейтингування, закладена в основу МайстерХаб, забезпечує більш точне, справедливе й захищене оцінювання, що є критично важливим для розвитку конкурентного та прозорого ринку побутових послуг.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Barrett M., Davidson E., Prabhu J., Vargo S. L. Service innovation in the digital age: Key contributions and future directions // MIS Quarterly. 2015. Vol. 39. № 1. P. 135–154. doi: 10.25300/MISQ/2015/39:1.03.
2. Bolton P., Freixas C., Shapiro J. The credit ratings game // Journal of Finance. 2012. Vol. 67. № 1. P. 85–111.
3. Casciaro R., Mammino L. Node.js design patterns. 3rd ed. Packt Publishing, 2020.
4. Chodorow K. MongoDB: The definitive guide. 3rd ed. O'Reilly Media, 2019.
5. Fedosejev A. React.js essentials. Packt Publishing, 2015.
6. Guo Y., Binnig C., Kraska T. What you see is not what you get: The costs of trading consistency for performance in distributed databases // Proceedings of VLDB Endowment. 2018. Vol. 11. № 12. P. 1814–1827.
7. Heller S. Comparison of frontend frameworks: React vs Vue vs Angular // Journal of Web Engineering. 2020. Vol. 19. № 3–4. P. 313–332.
8. Hoffman K. Beyond the twelve-factor app: Exploring the DNA of highly scalable, resilient cloud applications. O'Reilly Media, 2016.
9. Horton D., Zeckhauser R. Owning, using and renting: Some simple economics of the sharing economy // NBER Working Paper. 2016. № 22029.
10. Jindal N., Liu B. Opinion spam and analysis // Proceedings of the International Conference on Web Search and Data Mining (WSDM). 2008. P. 219–230. doi: 10.1145/1341531.1341560.
11. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) // RFC 7519. 2015. URL: <https://tools.ietf.org/html/rfc7519> (дата звернення: 24.11.2025).
12. Jurca R., Faltings B. Mechanisms for making crowds truthful // Journal of Artificial Intelligence Research. 2009. Vol. 34. P. 209–253.

13. Killalea T. The hidden dividends of microservices // Communications of the ACM. 2016. Vol. 59. № 8. P. 42–45.
14. Kumar G., Kumari D., Bux Q., Memon M., Baloch Z. Architectural framework for scalable on-demand service aid platform // International Journal of Innovations in Science and Technology. 2025.
15. Luca M. Reviews, reputation, and revenue: The case of Yelp.com // Harvard Business School Working Paper. 2011. № 12-016.
16. Momjian M. PostgreSQL: Introduction and concepts. Addison-Wesley, 2001.
17. Newman S. Building microservices: Designing fine-grained systems. 2nd ed. O'Reilly Media, 2021.
18. Provos N., Mazières D. A future-adaptable password scheme // Proceedings of USENIX Annual Technical Conference. 1999. P. 81–92.
19. Rai A., Sharma A., Kumar D. Performance analysis of web application frameworks: Node.js, Django, and Spring Boot // International Journal of Computer Applications. 2021. Vol. 183. № 10. P. 1–6.
20. Ramakrishnan R. ON-STEP SERVICES – Connecting experts on hands // International Scientific Journal of Engineering and Management. 2025.
21. Richardson C. Microservices patterns: With examples in Java. Manning Publications, 2018.
22. Sundararajan A. The sharing economy: The end of employment and the rise of crowd-based capitalism. MIT Press, 2016.
23. Teixeira P. Professional Node.js: Building JavaScript based scalable software. Wrox, 2013.
24. Wathan A. Tailwind CSS: From side-project to multi-million dollar business // Full Stack Radio. 2020. Episode 138.
25. Прищеп В. О., Кудерметов Р. К., Грушко С. С. Аналіз підходів до проєктування інформаційних систем домашнього сервісу з рейтинговою системою майстрів // Інформаційні технології і автоматизація – 2025 : XVIII міжнар. наук.-практ. конф., 30–31 жовт. 2025 р. : тези доповідей. Одеса, 2025.

ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ

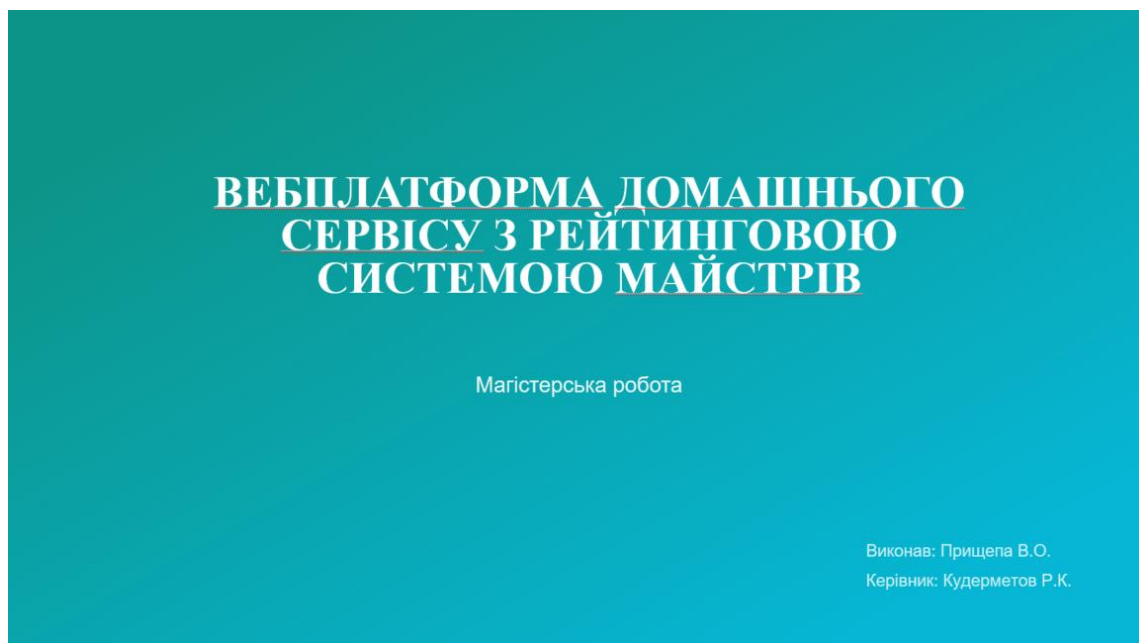


Рисунок А.1 – Слайд 1

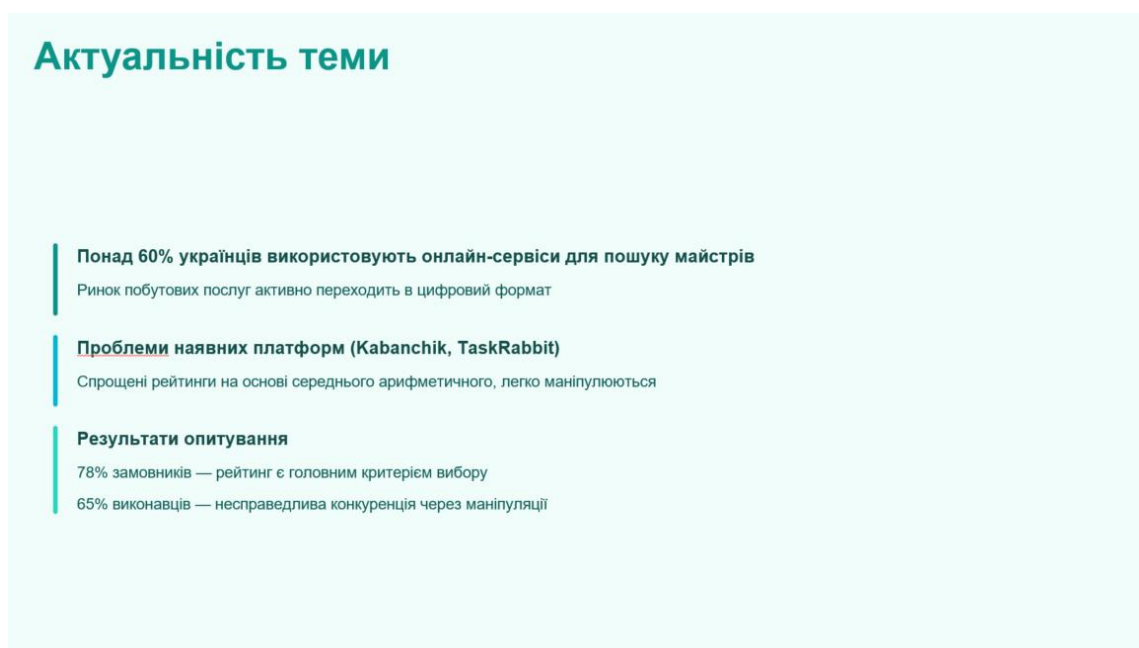


Рисунок А.2 – Слайд 2

Мета та завдання

МЕТА РОБОТИ

Проектування та розробка веб-платформи «МайстерХаб» з об'єктивною багатофакторною системою оцінювання виконавців

ОБ'ЄКТ

Процес цифрової взаємодії між замовниками та виконавцями побутових послуг

ПРЕДМЕТ

Методи та засоби створення веб-платформи з багатофакторною системою рейтингування

ЗАВДАННЯ

- Проаналізувати ринок та існуючі платформи
- Обґрунтувати вибір архітектурних рішень
- Розробити багатофакторну систему рейтингування
- Спроекувати структуру даних та UI

Рисунок А.3 – Слайд 3

Наукова новизна

Багатофакторна система рейтингування замість простого середнього арифметичного

Кількість і розподіл відгуків у часі

Стабільність роботи виконавця

Швидкість відповіді

Успішно виконані замовлення

Наявність верифікації

Виявлення аномальних патернів

Байєсівське згладжування — забезпечує справедливую оцінку як для досвідчених майстрів, так і для новачків

Захист від маніпуляцій: система виявляє підозріло велику кількість максимальних оцінок за короткий період

Рисунок А.4 – Слайд 4

Архітектурні рішення

МОНОЛІТНА АРХІТЕКТУРА

Обрано для MVP — швидкість розробки, простота розгортання та налагодження

ЧОМУ НЕ МІКРОСЕРВІСИ?

- Надлишкова складність на етапі валідації ідеї
- Вища вартість інфраструктури
- Модульний код дозволяє перехід у майбутньому

КЛІЄНТ-СЕРВЕРНА ВЗАЄМОДІЯ

React + TailwindCSS

Клієнтська частина

↕
REST API + JWT

Node.js + Express

Серверна частина

↕

MongoDB + Mongoose

База даних

Рисунок А.5 – Слайд 5

Обґрунтування технологій

Node.js + Express

- Єдина мова JS для frontend і backend
- Асинхронна модель для real-time сповіщень
- Гнучкість без надмірних абстракцій

React + TailwindCSS

- Компонентний підхід, повторне використання
- Віртуальний DOM для оптимізації
- Utility-first CSS прискорює розробку

MongoDB (замість PostgreSQL)

- Гетерогенна структура профілів майстрів
- Документи без JOIN — швидкість читання
- Горизонтальне масштабування (шардування)

Рисунок А.6 – Слайд 6

Структура бази даних

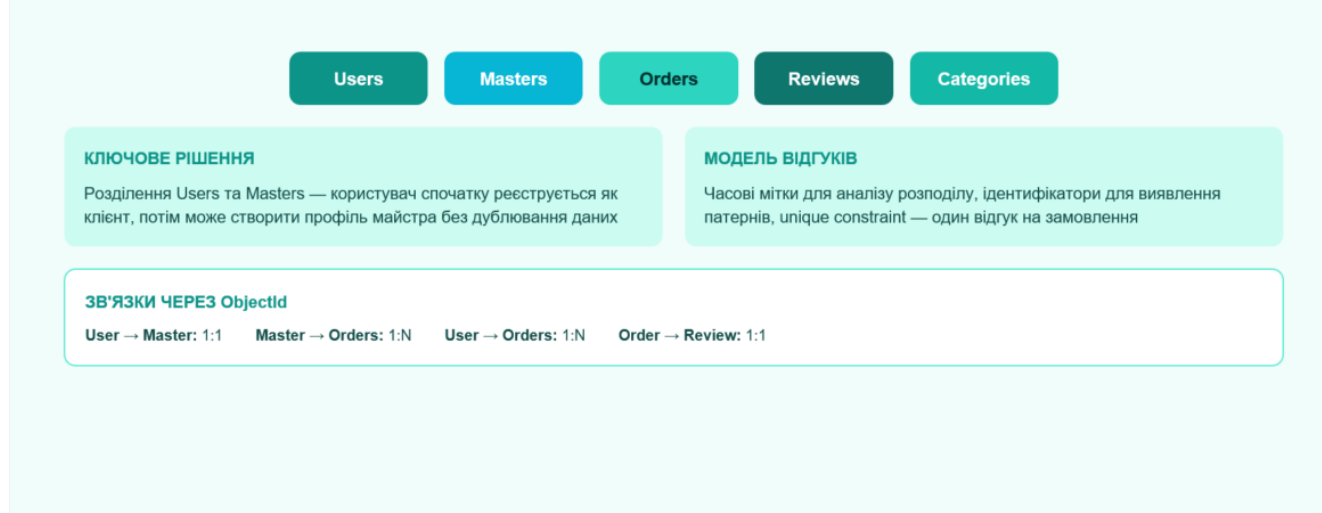


Рисунок А.7 — Слайд 7

Користувацький інтерфейс

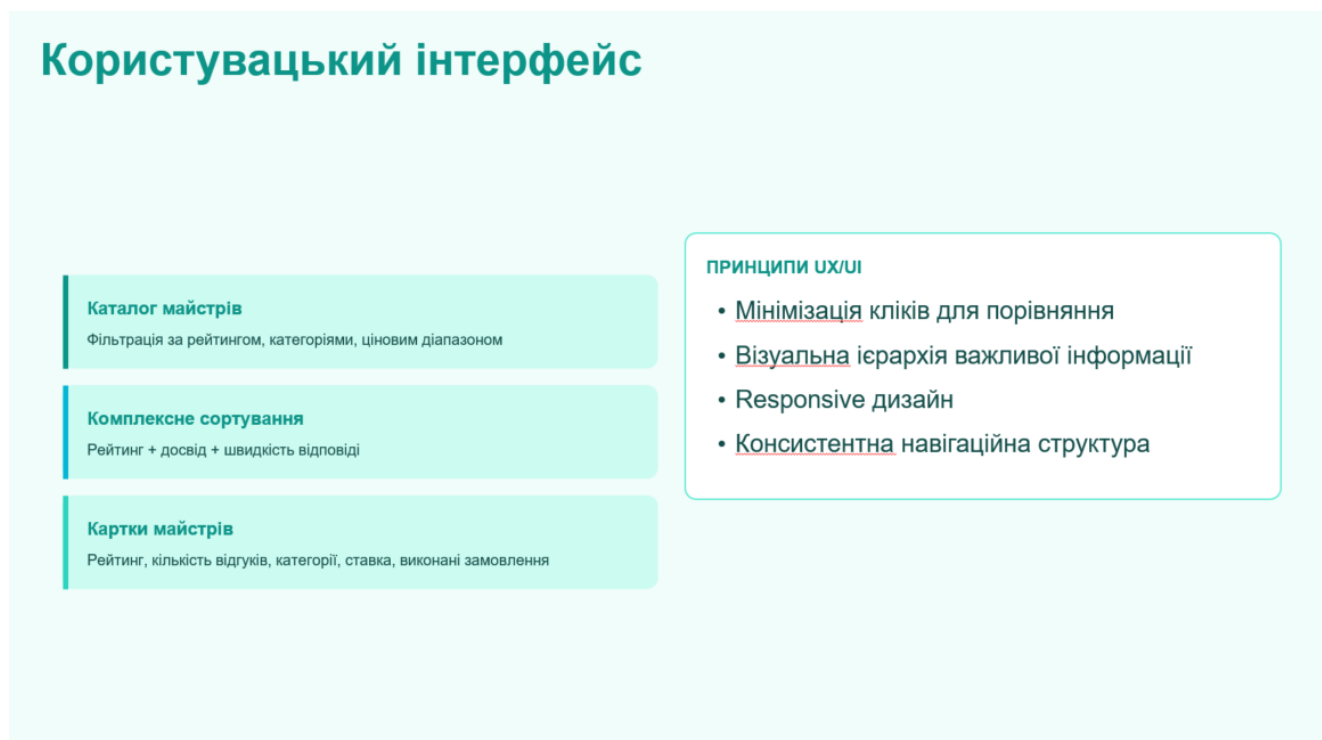
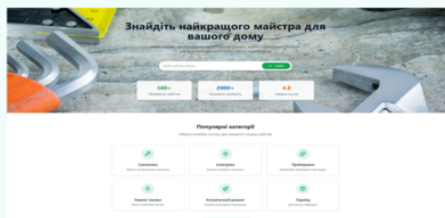


Рисунок А.8 — Слайд 8

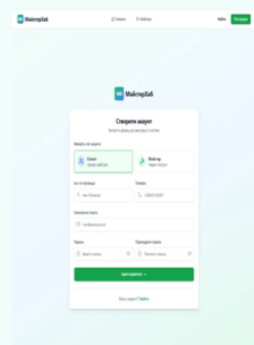
Користувацький інтерфейс

Компоненти системи

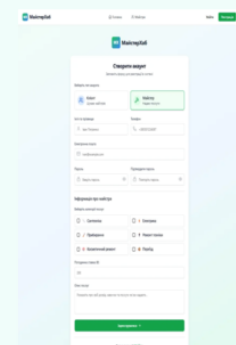
Головна сторінка • Автентифікація • Профілі • Замовлення • Відгуки • Сповіщення



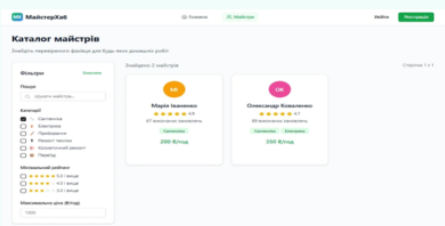
Головна сторінка з пошуковою формою



Форма реєстрації клієнта



Розширена форма реєстрації майстра



Каталог майстрів з фільтром за категорією Сантехніка

Рисунок А.9 – Слайд 9

Висновки

Проаналізовано ринок побутових послуг та виявлено недоліки наявних платформ

Обґрунтовано вибір технологічного стеку Node.js + MongoDB + React

Розроблено багатофакторну систему рейтингування з захистом від маніпуляцій

Спроектовано структуру даних та користувацький інтерфейс

Практичний результат: цілісна технічна концепція платформи, що усуває ключові недоліки існуючих сервісів та формує об'єктивні оцінки виконавців

АПРОБАЦІЯ РЕЗУЛЬТАТІВ

Тези доповіді на XVIII Міжнародній науково-практичній конференції «Інформаційні технології і автоматизація – 2025», м. Одеса

Рисунок А.10 – Слайд 10