ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Сайт для пошуку майстрів з використанням Spring boot»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Назар ФОМІН
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти гр. КНД-42
Назар ФОМІН

Керівник: Віктор ВИШНІВСЬКИЙ
науковий ступінь, вчене звання д.т.н., професор

Рецензент: _____
науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Фоміну Назару Сергійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Сайт для пошуку майстрів з використанням Spring boot»

керівник кваліфікаційної роботи Віктор ВИШНІВСЬКИЙ д.т.н., професор
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» 02.2025 р. № 56

2. Строк подання кваліфікаційної роботи «31» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, мови програмування Java, HTML, CSS, використання фреймворку Spring Boot, бази даних MySQL, технології REST API, моделі клієнт-серверної архітектури, принципи веб-безпеки, інструменти тестування та проєктування інформаційних систем.

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

4.1.Обгрунтування вимог до системи аналізу текстів.
4.2. Визначення архітектури проекту.
4.3 Реалізація проекту на Spring Boot.

5. Перелік графічного матеріалу: *презентація*
    1. Концепція платформи MasterPoint та актуальність проекту.
    2. Основні проблеми ринку та запропоновані рішення.
    3. Функціональність платформи: пошук, фільтри, відгуки, замовлення.
    4. Командна структура та організація розробки.
    5. План реалізації проекту по етапах.
    6. Аналіз ризиків платформи.

6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури та сучасних онлайн-сервісів	24.02-09.03.25	виконано
2	Дослідження архітектурних підходів та вибір технологій (Spring Boot, MySQL, REST)	09.03-16.03.25	виконано
3	Проектування структури бази даних та модулів платформи.	17.03-23.03.25	виконано
4	Реалізація основної функціональності: реєстрація, пошук, створення замовлень	24.03-29.03.25	виконано
5	Розробка системи фільтрації, рейтингу, відгуків	01.04-07.04.25	виконано
6	Тестування, оптимізація продуктивності, безпека.	08.04-24.04.25	виконано
7	Оформлення роботи: вступ, висновки, реферат	25.04-02.05.25	виконано
8	Розробка демонстраційних матеріалів	03.05-08.05.25	виконано

Здобувач вищої освіти _____ Назар ФОМІН
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Віктор ВИШНІВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина бакалаврської роботи: 78 с., 8 рис., 3 табл., 30 джерел.

Наукове завдання – розробка веб-платформи для ефективної взаємодії клієнтів і майстрів.

Мета роботи – створення сервісу на базі Spring Boot для пошуку майстрів та організації замовлень.

Об’єкт дослідження – процес взаємодії між клієнтами та майстрами в онлайн-середовищі.

Предмет дослідження – методи та технології розробки клієнт-серверної веб-платформи.

Методи дослідження – аналіз рішень, об'єктно-орієнтоване проектування, тестування.

Сфера застосування – онлайн-сервіси побутових і професійних послуг.

Короткий зміст роботи:

Розроблено платформу MasterPoint з архітектурою на Spring Boot. Реалізовано реєстрацію, REST API, пошук і фільтрацію майстрів, рейтинги та відгуки, а також систему авторизації. Проведено тестування та підготовку до розгортання. Платформа готова до впровадження та може адаптуватися під різні галузі.

Ключові слова: Spring Boot, Java, веб-розробка, маркетплейс, REST API, рейтинг, клієнт-сервер, MVC

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (Інтерфейс програмування додатків)

AJAX – Asynchronous JavaScript and XML (Асинхронний JavaScript та XML)

ACID – Atomicity, Consistency, Isolation, Durability (Атомарність, Узгодженість, Ізольованість, Довговічність)

CDN – Content Delivery Network (Мережа доставки контенту)

CI/CD – Continuous Integration / Continuous Deployment (Безперервна інтеграція / Безперервне розгортання)

CQRS – Command Query Responsibility Segregation (Розподіл відповідальності команд та запитів)

CRUD – Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення)

CSRF – Cross-Site Request Forgery (Підробка міжсайтових запитів)

CSS – Cascading Style Sheets (Каскадні таблиці стилів)

DAO – Data Access Object (Об'єкт доступу до даних)

DDD – Domain-Driven Design (Доменно-орієнтоване проектування)

DI – Dependency Injection (Впровадження залежностей)

DTO – Data Transfer Object (Об'єкт передачі даних)

EDA – Event-Driven Architecture (Подійно-орієнтована архітектура)

GDPR – General Data Protection Regulation (Загальний регламент захисту даних)

HTML – HyperText Markup Language (Мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (Протокол передачі гіпертексту)

HTTPS – HyperText Transfer Protocol Secure (Захищений протокол передачі гіпертексту)

IoC – Inversion of Control (Інверсія управління)

JPA – Java Persistence API (Інтерфейс програмування додатків для збереження даних Java)

JSON – JavaScript Object Notation (Нотація об'єктів JavaScript)

JWT – JSON Web Token (Веб-токен JSON)

MVC – Model-View-Controller (Модель-Представлення-Контролер)

MVVM – Model-View-ViewModel (Модель-Представлення-МодельПредставлення)

NoSQL – Not Only SQL (Не тільки SQL)

ORM – Object-Relational Mapping (Об'єктно-реляційне відображення)

PWA – Progressive Web Application (Прогресивний веб-додаток)

REST – Representational State Transfer (Передача репрезентативного стану)

SEO – Search Engine Optimization (Оптимізація для пошукових систем)

SQL – Structured Query Language (Мова структурованих запитів)

СУБД – Система Управління Базами Даних

TDD – Test-Driven Development (Розробка через тестування)

UX/UI – User Experience / User Interface (Досвід користувача / Інтерфейс користувача)

XSS – Cross-Site Scripting (Міжсайтовий скриптинг)

2FA – Two-Factor Authentication (Двофакторна автентифікаці

ВСТУП

У сучасному світі, що характеризується швидким темпом життя та зростаючою спеціалізацією праці, знаходження кваліфікованих спеціалістів для вирішення побутових та професійних завдань стає все більш актуальною проблемою. Традиційні методи пошуку майстрів, такі як рекомендації знайомих або оголошення, часто виявляються неефективними, оскільки не забезпечують достатнього рівня прозорості, надійності та зручності. Водночас, розвиток інформаційних технологій та поширення інтернету створюють можливості для розробки інноваційних рішень, які можуть суттєво покращити процес пошуку та взаємодії між клієнтами та виконавцями послуг.

Платформи для пошуку спеціалістів стають все більш популярними як у світі, так і в Україні, демонструючи стабільне зростання аудиторії та обсягів транзакцій. Такі сервіси допомагають вирішувати проблему асиметрії інформації на ринку послуг, коли клієнтам складно оцінити якість роботи майстра до її виконання, а майстрам – знайти потенційних замовників. Створення надійної репутаційної системи, зручних інструментів пошуку та комунікації, а також забезпечення безпеки взаємодії є ключовими факторами успіху таких платформ.

Розробка системи MasterPoint є актуальною відповіддю на зростаючий попит на зручні та ефективні онлайн-рішення для пошуку майстрів. Ця платформа покликана з'єднати клієнтів, які потребують виконання певних робіт, з кваліфікованими спеціалістами у відповідних галузях, забезпечуючи прозорість, надійність та зручність процесу. Використання сучасних технологій веб-розробки, зокрема фреймворку Spring Boot, дозволяє створити масштабоване, надійне та продуктивне рішення, здатне задовольнити потреби різних категорій користувачів.

Метою даної кваліфікаційної роботи є розробка веб-платформи для пошуку майстрів з використанням технології Spring Boot, яка забезпечить ефективну взаємодію між клієнтами та виконавцями послуг. Для досягнення цієї мети були поставлені наступні завданн

- Провести аналіз існуючих платформ для пошуку спеціалістів, визначити їх переваги та недоліки.
- Дослідити та обґрунтувати вибір технологій для розробки веб-додатку, зокрема Spring Boot.
- Розробити архітектуру системи, спроектувати базу даних та структуру проекту.
- Реалізувати моделі даних та бізнес-логіку системи.
- Розробити контролери та REST API для забезпечення взаємодії клієнтської та серверної частин.
- Реалізувати систему авторизації та аутентифікації користувачів.
- Створити інтерфейс користувача та налаштувати його взаємодію з бекендом.
- Реалізувати механізми пошуку та фільтрації майстрів за різними критеріями.
- Розробити систему відгуків та рейтингу майстрів.
- Провести тестування та оптимізацію продуктивності системи.
- Розробити стратегію розгортання та впровадження системи.

Об'єктом дослідження є процес пошуку та взаємодії між клієнтами та майстрами в онлайн-середовищі

Предметом дослідження є методи та технології розробки веб-платформи для пошуку майстрів з використанням Spring Boot

Практична значущість роботи полягає у створенні готового до впровадження програмного продукту, який може бути використаний для з'єднання клієнтів та майстрів у різних галузях побутових та професійних послуг. Розроблена платформа може стати основою для комерційного проєкту або бути адаптована для використання в інших сферах, де потрібно забезпечити взаємодію між замовниками та виконавцями послуг.

Структура роботи відповідає поставленим завданням та включає три розділи. Перший розділ присвячений теоретичним основам розробки сервісів для пошуку майстрів, аналізу існуючих рішень та обґрунтуванню вибору технологій. Другий розділ розкриває процес проєктування та розробки системи MasterPoint, включаючи визначення вимог, проєктування бази даних, реалізацію моделей даних, бізнес-логіки, контролерів та системи авторизації. Третій розділ описує реалізацію та тестування системи, включаючи інтерфейс користувача, механізми пошуку та фільтрації, систему відгуків та рейтингу, а також процеси тестування, оптимізації та розгортання системи.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ СЕРВІСІВ ДЛЯ ПОШУКУ МАЙСТРІВ

1.1 Аналіз сучасних платформ для пошуку спеціалістів

У сучасному світі пошук кваліфікованих спеціалістів для вирішення побутових і професійних завдань став невід'ємною частиною повсякденного життя. Ринок онлайн-платформ, що з'єднують замовників з виконавцями, демонструє стабільне зростання протягом останнього десятиліття. Такі сервіси допомагають ефективно вирішувати проблему асиметрії інформації, коли замовнику складно оцінити якість послуг до їх отримання, а спеціалістам — знайти потенційних клієнтів.

Розглядаючи наявні рішення на ринку табл. 1.1, можна виділити кілька категорій платформ для пошуку спеціалістів. Перша категорія — вертикальні маркетплейси, орієнтовані на конкретну галузь. Наприклад, TaskRabbit спеціалізується на побутових послугах, а Thumbtack пропонує широкий спектр професійних послуг від репетиторства до ремонтних робіт. Такі платформи зазвичай мають глибоку спеціалізацію в певній ніші і пропонують детальну категоризацію послуг.

Таблиця 1.1 Порівняння популярних платформ для пошуку спеціалістів

Тип платформи	**Приклади**	**Переваги**	**Недоліки**	**Модель монетизації**
Вертикальні маркетплейси	TaskRabbit, Thumbtack	Глибока спеціалізація в ніші, детальна категоризація послуг	Обмежений спектр послуг	Комісія з транзакцій (10-15%)
Горизонтальні платформи	Upwork, Freelancer	Широкий спектр послуг, велика аудиторія	Менша ефективність у специфічних галузях	Комісія з транзакцій (5-20%), підписка на преміум-функції

Продовження таблиці 1.1

Локальні сервіси	Kabanchik (Україна), Яндекс.Услуги (Росія)	Адаптація до місцевого ринку, врахування регіональних особливостей	Обмежене географічне охоплення	Комісія з транзакцій (5-15%), платне розміщення оголошень

Друга категорія — горизонтальні платформи, такі як Upwork або Freelancer, які охоплюють широкий спектр професійних послуг і спеціалізацій. Ці платформи пропонують гнучкі інструменти для пошуку, відбору та оплати послуг спеціалістів різних профілів. Їхня перевага — велика аудиторія та різноманітність пропозицій, але часто вони менш ефективні в специфічних галузях порівняно з вертикальними маркетплейсами.

Третя категорія — локальні сервіси, орієнтовані на певний географічний регіон. Такі платформи враховують особливості місцевого ринку, культурні та мовні особливості, а також регіональні регуляторні вимоги. Прикладом такого сервісу є Kabanchik в Україні або Яндекс.Услуги в Росії. Їхня перевага полягає в кращому розумінні місцевого ринку та потреб локальних користувачів.

Аналізуючи функціональність успішних платформ, можна виділити ключові механізми, що забезпечують ефективну взаємодію між замовниками і виконавцями. Перш за все, це розвинена система профілів з детальною інформацією про спеціалістів, їхню кваліфікацію, досвід і портфоліо виконаних робіт. Необхідним елементом є система рейтингів і відгуків, яка дозволяє формувати репутацію виконавців на основі реального досвіду їхніх клієнтів.

Значущим компонентом сучасних платформ є алгоритми пошуку та підбору спеціалістів, які враховують багато факторів: спеціалізацію, географічну близькість, рейтинг, вартість послуг і доступність. Деякі платформи впроваджують елементи штучного інтелекту для покращення релевантності результатів пошуку та персоналізації рекомендацій [1].

Система комунікації між замовниками та виконавцями також є критично необхідною. Більшість сучасних платформ пропонують вбудовані месенджери для обговорення деталей проекту, відправки файлів і погодження умов співпраці. Деякі рішення інтегрують інструменти для проведення відеоконференцій, що особливо актуально в контексті пандемії COVID-19.

Монетизація платформ для пошуку спеціалістів переважно ґрунтується на кількох моделях: комісія з транзакцій, передплата за преміум-функції, реклама та платне розміщення. Найпоширенішою є модель комісії з успішних угод, коли платформа утримує певний відсоток від суми, яку замовник сплачує виконавцю. Розмір комісії варіюється від 5% до 20% залежно від категорії послуг та конкретної платформи.

Вагомим аспектом функціонування сучасних платформ є забезпечення безпеки та захисту інтересів усіх учасників. Для цього впроваджуються механізми верифікації спеціалістів, захищені платежі, страхування відповідальності та служби підтримки для вирішення конфліктних ситуацій. Деякі платформи пропонують гарантію повернення коштів, якщо якість послуг не відповідає очікуванням.

Технологічна складова сучасних платформ набуває дедалі більшого значення. Користувачі очікують інтуїтивно зрозумілий інтерфейс, швидке завантаження, відсутність технічних помилок і безперервний доступ до сервісу через різні пристрої. Тому лідери ринку інвестують значні ресурси в розробку та підтримку стабільної технічної інфраструктури.

Мобільні додатки стали невід'ємною частиною стратегії розвитку платформ для пошуку спеціалістів. Вони дозволяють користувачам отримувати сповіщення в реальному часі, швидко реагувати на запити та керувати замовленнями, перебуваючи в будь-якому місці. Більшість успішних платформ пропонують нативні додатки для iOS та Android з функціоналом, близьким до веб-версії.

Тенденцією останніх років стало впровадження додаткових сервісів, що розширюють екосистему платформ. Це можуть бути інструменти для управління проектами, автоматизації виставлення рахунків, укладання контрактів, ведення

календаря зустрічей та інтеграції з CRM-системами. Такі додаткові функції створюють більш цілісний досвід користування та підвищують лояльність аудиторії.

Нові технологічні рішення, такі як блокчейн, починають впроваджуватися в платформи для пошуку спеціалістів з метою підвищення прозорості транзакцій та захисту від шахрайства. Хоча ці технології ще перебувають на початковому етапі впровадження, вони мають потенціал для трансформації галузі в майбутньому.

Правові аспекти функціонування платформ також викликають значний інтерес. У різних країнах введено регуляторні вимоги, що стосуються оподаткування, захисту даних користувачів, трудових відносин між платформами та спеціалістами. Наприклад, у Європейському Союзі GDPR встановлює суворі вимоги до обробки персональних даних, а в деяких країнах починають розробляти спеціалізоване законодавство для регулювання діяльності онлайн-платформ [2].

Аналіз конкурентів показує, що успішні платформи постійно еволюціонують, адаптуючись до змін у поведінці користувачів, технологічних інновацій та конкурентного ландшафту. Вони активно збирають дані про взаємодію користувачів з платформою і використовують їх для оптимізації бізнес-процесів та покращення користувацького досвіду.

1.2 Огляд технологій для створення веб-додатків

Сучасний стек технологій для розробки веб-додатків надзвичайно різноманітний, що дозволяє обирати оптимальні інструменти для вирішення конкретних завдань. Вибір технологій впливає на швидкість розробки, масштабованість, продуктивність і підтримку проекту в довгостроковій перспективі. Для створення платформи пошуку спеціалістів на кшталт MasterPoint необхідно розглянути різні аспекти технологічного стеку.

Серверні мови програмування є фундаментом, на якому будується бекенд-частина веб-додатку. Серед найпопулярніших мов для веб-розробки можна виділити Java, Python, JavaScript (Node.js), PHP, Ruby, C# та Go. Кожна мова має свої переваги: Java відзначається високою продуктивністю, надійністю та багатою

екосистемою фреймворків; Python привабливий простотою синтаксису та широкими можливостями для інтеграції з системами аналітики та штучного інтелекту; Node.js дозволяє використовувати JavaScript як на клієнті, так і на сервері, що спрощує розробку [3].

Фреймворки серверної частини значно прискорюють процес розробки, надаючи готові шаблони вирішення типових завдань. У світі Java популярними фреймворками є Spring Boot, Hibernate, Play Framework; для Python — Django, Flask, FastAPI; для JavaScript (Node.js) — Express.js, Nest.js, Koa; для PHP — Laravel, Symfony; для Ruby — Ruby on Rails; для C# — ASP.NET Core; для Go — Gin, Echo. Вибір фреймворку залежить від специфіки проекту, вимог до продуктивності, масштабованості та наявних компетенцій команди розробників.

Клієнтські технології визначають, як користувач взаємодіятиме з веб-додатком. Основними мовами фронтенд-розробки залишаються HTML5, CSS3 та JavaScript. З розвитком JavaScript з'явилось багато фреймворків і бібліотек, які полегшують створення динамічних і реактивних інтерфейсів. Найпопулярніші з них — React, Angular, Vue.js, Svelte. Ці фреймворки використовують компонентний підхід до розробки інтерфейсів, що спрощує повторне використання коду та дозволяє створювати складні інтерактивні додатки.

Бази даних є критично необхідним компонентом веб-додатків, особливо для платформ із значним обсягом даних про користувачів і угоди. Реляційні бази даних, такі як MySQL, PostgreSQL, Oracle, Microsoft SQL Server, оптимальні для структурованих даних і забезпечують ACID-транзакції. NoSQL-рішення, такі як MongoDB, Cassandra, Redis, Elasticsearch, краще підходять для сценаріїв з великим обсягом неструктурованих даних, швидким масштабуванням і гнучкою схемою даних. Для платформи пошуку спеціалістів часто використовується гібридний підхід: реляційні бази даних для транзакційних даних і NoSQL-рішення для пошукових індексів і кешування.

Архітектурні підходи до розробки веб-додатків також відіграють головну роль. Монолітна архітектура, де весь додаток розгортається як єдине ціле, простіша в розробці та розгортанні на початкових етапах. Мікросервісна

архітектура, що розділяє додаток на незалежні сервіси, забезпечує кращу масштабованість і гнучкість, але вимагає більше ресурсів на управління інфраструктурою. Для нових проектів часто обирають підхід "монолітний перший", з подальшим переходом до мікросервісів у міру зростання проекту і команди.

API (Application Programming Interface) є головною частиною сучасних веб-додатків, особливо при реалізації архітектури, де фронтенд і бекенд розділені. RESTful API залишається найпоширенішим підходом завдяки своїй простоті та універсальності. GraphQL набуває популярності як альтернатива, що дозволяє клієнтам запитувати саме ті дані, які їм потрібні, зменшуючи надмірність передачі даних. gRPC є ефективним вибором для комунікації між мікросервісами завдяки високій продуктивності бінарних протоколів [4].

Аутентифікація та авторизація є критичними аспектами безпеки веб-додатків, особливо для платформ, де відбуваються фінансові транзакції. Сучасні рішення включають JWT (JSON Web Tokens), OAuth 2.0 для делегування доступу, OpenID Connect для аутентифікації. Багатофакторна аутентифікація стає стандартом для додаткового рівня захисту. Для управління користувачами часто використовуються готові рішення, такі як Keycloak, Auth0, Firebase Authentication. Керування станом на клієнтській стороні ускладнюється з ростом функціональності фронтенд-додатків. Бібліотеки Redux, MobX, Vuex, NgRx допомагають організувати і підтримувати стан додатку в передбачуваний спосіб. Реактивне програмування з використанням бібліотек типу RxJS дозволяє елегантно працювати з асинхронними потоками даних, що особливо корисно для реагування на дії користувача та обробки результатів API-запитів.

Контейнеризація та оркестрація суттєво змінили підходи до розгортання та масштабування веб-додатків. Docker дозволяє упаковувати додатки та їхні залежності в контейнери, забезпечуючи однакове середовище виконання незалежно від інфраструктури. Kubernetes став де-факто стандартом для оркестрації контейнерів, забезпечуючи автоматичне масштабування,

самовідновлення та балансування навантаження. Ці технології спрощують процес безперервної інтеграції та доставки (CI/CD).

Хмарні сервіси надають гнучку інфраструктуру для розгортання веб-додатків. Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform пропонують широкий спектр сервісів: від віртуальних машин до готових рішень для аутентифікації, аналітики, машинного навчання. Функції як сервіс (FaaS) і безсерверні архітектури, такі як AWS Lambda, Azure Functions, дозволяють розробникам зосередитися на коді, не турбуючись про інфраструктуру.

Тестування є невід'ємною частиною процесу розробки якісних веб-додатків. Модульне тестування (Unit Testing) перевіряє окремі компоненти коду за допомогою фреймворків, таких як JUnit, Jest, Mocha. Інтеграційне тестування перевіряє взаємодію між компонентами. End-to-end тестування симулює поведінку користувача від початку до кінця за допомогою інструментів, таких як Selenium, Cypress, Puppeteer. Автоматизоване тестування є необхідним компонентом CI/CD-пайплайнів.

Продуктивність веб-додатків безпосередньо впливає на користувацький досвід і конверсію. Оптимізація часу завантаження через мінімізацію й об'єднання файлів, використання CDN (Content Delivery Network), кешування є стандартними практиками. Прогресивні веб-додатки (PWA) дозволяють забезпечити досвід, близький до нативних мобільних додатків, з підтримкою офлайн-режиму та швидким завантаженням. Технології серверного рендерингу (Server-Side Rendering) і попереднього рендерингу (Pre-rendering) покращують сприйняття швидкості завантаження та SEO.

Моніторинг і аналітика допомагають розуміти поведінку користувачів та виявляти проблеми в реальному часі. Інструменти, такі як Google Analytics, Mixpanel, дозволяють аналізувати користувацьку активність. Сервіси моніторингу, як New Relic, Datadog, Prometheus з Grafana, надають детальну інформацію про продуктивність додатку. Сервіси логування, такі як ELK Stack (Elasticsearch, Logstash, Kibana), Graylog, допомагають агрегувати та аналізувати логи з різних частин системи [5].

Безпека веб-додатків вимагає комплексного підходу. Захист від OWASP Top 10 вразливостей, включаючи SQL-ін'єкції, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), є базовим рівнем безпеки. Шифрування даних при передачі (HTTPS) і в стані спокою стало стандартом. Регулярні аудити безпеки, тестування на проникнення (Penetration Testing) і відповідність нормативним вимогам, таким як GDPR, CCPA, допомагають підтримувати високий рівень захисту користувацьких даних.

1.3 Обґрунтування вибору технології Spring Boot для розробки

Spring Boot є одним із найпопулярніших фреймворків табл. 1.2 для розробки веб-додатків на мові Java, який значно спрощує процес створення і налаштування Spring-застосунків. Цей фреймворк було розроблено як відповідь на зростаючу складність конфігурації традиційних Spring-додатків і необхідність прискорення процесу розробки. Основна філософія Spring Boot – "convention over configuration" (конвенція замість конфігурації), що дозволяє розробникам зосередитися на бізнес-логіці, а не на налаштуванні інфраструктури.

Таблиця 1.2 Порівняння фреймворків для розробки веб-додатків

Критерій	**Spring Boot**	**Django (Python)**	**Express.js (Node.js)**	**Laravel (PHP)**
Швидкість розробки	Висока завдяки автоконфігурації	Висока завдяки "batteries included" підходу	Середня, вимагає більше ручного налаштування	Висока завдяки елегантному синтаксису
Продуктивність	Висока	Середня	Висока (асинхронна модель)	Середня
Масштабованість	Відмінна, підтримка мікросервісів	Хороша, але вимагає додаткових зусиль	Відмінна для I/O-bound операцій	Хороша з правильною архітектурою

Продовження таблиці 1.2

Екосистема	Надзвичайно розвинена	Розвинена	Найбільша екосистема пакетів (npm)	Розвинена
Безпека	Вбудована підтримка через Spring Security	Вбудований захист від типових атак	Потребує додаткових бібліотек	Вбудований захист від типових атак
Документація	Вичерпна	Вичерпна	Хороша, але фрагментована	Вичерпна
Крива навчання	Крута, потребує розуміння Java та Spring	Помірна	Плавна, особливо для тих, хто знає JavaScript	Помірна

Однією з ключових переваг Spring Boot є автоконфігурація, яка дозволяє фреймворку самостійно налаштовувати компоненти на основі залежностей, доданих до проекту. Наприклад, якщо до проекту додано залежність spring-boot-starter-data-jpa, фреймворк автоматично налаштує DataSource, EntityManager та інші компоненти, необхідні для роботи з базою даних через JPA. Це суттєво зменшує кількість коду, який розробники мають написати для налаштування проекту, і дозволяє швидше переходити до реалізації функціоналу.

Spring Boot Starters – це набори залежностей, які спрощують інтеграцію різних технологій у Spring-додаток. Наприклад, spring-boot-starter-web включає все необхідне для розробки веб-додатків, включаючи вбудований сервер Tomcat і Spring MVC. Це дозволяє уникнути необхідності явно вказувати всі залежності та їх версії, забезпечуючи їх сумісність. Для проекту MasterPoint це особливо необхідно, оскільки він потребує інтеграції багатьох компонентів: бази даних, системи аутентифікації, RESTful API тощо.

Вбудований веб-сервер є ще однією перевагою Spring Boot. Завдяки цьому додаток можна запускати як звичайний Java-додаток, без необхідності налаштовувати окремий сервер. За замовчуванням використовується Apache Tomcat, але можна легко перейти на інші сервери, такі як Jetty або Undertow. Це

спрощує розгортання додатка і робить його більш портативним, що необхідно для забезпечення однакової роботи в різних середовищах – від локальної розробки до промислової експлуатації.

Модуль Spring Data JPA значно спрощує роботу з базами даних, надаючи високорівневі абстракції для доступу до даних. Розробники можуть створювати репозиторії шляхом простого визначення інтерфейсів, а Spring автоматично генерує реалізацію. Це зменшує кількість шаблонного коду для операцій CRUD (Create, Read, Update, Delete) і дозволяє зосередитися на бізнес-логіці. Для платформи пошуку спеціалістів, де є складна модель даних з багатьма взаємопов'язаними сутностями, це суттєво прискорює розробку.

Spring Security надає потужний і гнучкий механізм для аутентифікації та авторизації користувачів. Він підтримує різні методи аутентифікації, включаючи формову аутентифікацію, Basic Authentication, OAuth 2.0, JWT тощо. Spring Security також дозволяє визначати детальні правила доступу до ресурсів на основі ролей користувачів та інших параметрів. Для MasterPoint, де є різні типи користувачів (клієнти, майстри, адміністратори) з різними рівнями доступу, це критично необхідний компонент.

Підтримка RESTful API вбудована в Spring Boot через Spring MVC і Spring Data REST. Це дозволяє легко створювати API-ендпоінти, використовуючи анотації для визначення HTTP-методів, параметрів запитів, валідації даних тощо. Spring також надає засоби для документування API за допомогою інтеграції з інструментами, такими як Swagger/OpenAPI. Для веб-платформи, де клієнтська частина взаємодіє з сервером через API, це є ключовим функціоналом.

Тестування додатків на Spring Boot спрощується завдяки вбудованим інструментам для модульного та інтеграційного тестування. Spring Test надає підтримку для тестування компонентів з різними рівнями ізоляції, а також для тестування веб-шару за допомогою MockMvc. Можливо також запускати інтеграційні тести з використанням вбудованої бази даних, такої як H2, що забезпечує швидке виконання тестів без залежності від зовнішніх систем. Якісне тестування критично головне для забезпечення надійності платформи.

Підтримка моніторингу та метрик через Spring Boot Actuator дозволяє легко відстежувати стан додатка і його продуктивність у режимі реального часу. Actuator надає ендпоінти для перевірки стану додатка, його метрик, аудиту, логування тощо. Це особливо необхідно для виробничого середовища, де необхідно швидко реагувати на проблеми та забезпечувати безперебійну роботу сервісу для користувачів.

Екосистема Spring є однією з найбільших і найактивніших у світі Java. Це забезпечує доступ до величезної кількості бібліотек, інструментів і навчальних ресурсів. Крім того, Spring має велику спільноту розробників, які активно діляться досвідом і допомагають вирішувати проблеми. Це знижує ризики при розробці і підтримці проекту, оскільки можна легко знайти рішення для більшості типових проблем.

Продуктивність Spring Boot додатків достатньо висока для більшості сценаріїв використання. Хоча існують більш швидкі фреймворки, такі як Micronaut або Quarkus, Spring Boot забезпечує гарний баланс між продуктивністю і функціональністю. Для покращення продуктивності можна використовувати такі техніки, як кешування, асинхронна обробка запитів, оптимізація запитів до бази даних тощо. Spring Boot також підтримує профілювання, що дозволяє виявляти вузькі місця в додатку.

Масштабованість додатків на Spring Boot забезпечується через підтримку мікросервісної архітектури, контейнеризації з Docker і оркестрації з Kubernetes. Spring Cloud надає інструменти для розробки розподілених систем, включаючи сервісне виявлення, конфігурацію, балансування навантаження тощо. Це дозволяє платформі MasterPoint розвиватися і масштабуватися в міру зростання кількості користувачів і обсягу даних.

Інтеграція з іншими системами спрощується завдяки широким можливостям Spring Integration та Spring Batch. Ці модулі дозволяють легко взаємодіяти з зовнішніми сервісами через різні протоколи, обробляти повідомлення з черг, імпортувати і експортувати дані тощо. Для платформи MasterPoint це важливо для інтеграції з сервісами платежів, геолокації, сповіщень тощо.

Безпека є пріоритетом для команди Spring, яка регулярно випускає оновлення для виправлення виявлених вразливостей. Spring Security пропонує широкий спектр функцій для захисту додатків від найпоширеніших атак, таких як CSRF, XSS, SQL-ін'єкції тощо. Крім того, велика спільнота розробників забезпечує швидке виявлення і реагування на проблеми безпеки [6]. Для

платформи, де зберігаються персональні дані користувачів і проводяться фінансові операції, це критично важливо.

Підтримка всіх фаз життєвого циклу додатка – від розробки до промислової експлуатації – є ще однією перевагою Spring Boot. У фазі розробки допомагають такі функції, як автоматичне перезавантаження при зміні коду (Developer Tools), а у фазі експлуатації – інструменти для моніторингу, логування та налаштування без простою. Це забезпечує ефективність роботи команди на всіх етапах проекту і зменшує час від розробки до випуску нових функцій.

1.4 Архітектурні патерни проектування для веб-додатків

Архітектурні патерни є фундаментальними принципами організації програмного забезпечення, які визначають структуру, взаємодію компонентів і загальний підхід до розробки веб-додатків. Вибір правильних архітектурних патернів критично необхідний для забезпечення якості, масштабованості, підтримки та розвитку веб-додатку. Розглянемо основні архітектурні патерни, які можуть бути застосовані при розробці платформи пошуку майстрів MasterPoint.

Модель-Вид-Контролер (Model-View-Controller, MVC) є одним з найпоширеніших архітектурних патернів для веб-додатків. Він розділяє додаток на три взаємопов'язані компоненти: Модель (відповідає за дані та бізнес-логіку), Вид (відповідає за відображення даних користувачу) і Контролер (обробляє запити користувача, взаємодіє з Моделлю і вибирає Вид для відображення). У контексті Spring Boot, цей патерн реалізується через Spring MVC, де контролери обробляють HTTP-запити, сервіси та репозиторії формують шар моделі, а шаблони Thymeleaf або інші технології використовуються для представлення даних.

Модель-Вид-ВидМодель (Model-View-ViewModel, MVVM) є розширенням патерну MVC, де ВидМодель служить посередником між Моделлю і Видом, забезпечуючи конвертацію даних у формат, зручний для відображення. Цей патерн особливо популярний у клієнтських додатках і фреймворках, таких як Angular, Vue.js, React. Для MasterPoint цей патерн може бути використаний для організації клієнтської частини додатку, особливо якщо вона розробляється як одностранична програма (SPA) з використанням сучасних JavaScript-фреймворків. Багатошарова архітектура (Layered Architecture) організовує додаток у горизонтальні шари, кожен з яких виконує певну роль. Типовими шарами є: презентаційний шар (обробка запитів користувача), шар бізнес-логіки (реалізація бізнес-правил), шар доступу до даних (взаємодія з базою даних та іншими джерелами даних) і шар домену (основні сутності та бізнес-концепти). В Spring Boot ця архітектура реалізується через контролери (презентаційний шар), сервіси (шар бізнес-логіки), репозиторії (шар доступу до даних) і доменні об'єкти (шар домену). Багатошарова архітектура забезпечує чіткий розподіл відповідальності та покращує тестованість додатку.

Мікросервісна архітектура (Microservices Architecture) розділяє додаток на невеликі, незалежні сервіси, кожен з яких відповідає за певну функціональність і може розроблятися, розгортатися та масштабуватися незалежно від інших. Для MasterPoint можна виділити такі мікросервіси, як сервіс користувачів, сервіс замовлень, сервіс пошуку, сервіс повідомлень, сервіс платежів тощо. Spring Boot і Spring Cloud надають потужні інструменти для розробки мікросервісів, включаючи Spring Cloud Config для централізованої конфігурації, Eureka для сервісного виявлення, Feign для спрощення HTTP-клієнтів, Ribbon для балансування навантаження та Hystrix для забезпечення стійкості до збоїв. Подійно-орієнтована архітектура (Event-Driven Architecture, EDA) базується на створенні, виявленні, споживанні та реагуванні на події. Події можуть розглядатися як значущі зміни стану, які публікуються видавцями і споживаються підписниками. Цей патерн особливо корисний для системи MasterPoint, де багато дій можна моделювати як події: створення нового замовлення, відгук майстра на

замовлення, вибір майстра клієнтом, завершення роботи тощо. Spring підтримує EDA через Spring ApplicationEvents для внутрішніх подій і Spring Cloud Stream для інтеграції з брокерами повідомлень, такими як Apache Kafka або RabbitMQ [7].

Доменно-орієнтоване проектування (Domain-Driven Design, DDD) – це підхід до розробки складних програмних систем, який фокусується на створенні моделі домену, яка відображає бізнес-концепти та правила. DDD включає такі концепції, як сутності, агрегати, сховища, фабрики, сервіси домену тощо. Для платформи MasterPoint, де є складна бізнес-логіка, пов'язана з пошуком майстрів, створенням замовлень, управлінням відгуками та рейтингами, DDD може допомогти створити чітку і зрозумілу модель, яка відображає реальні бізнес-процеси.

Інверсія управління (Inversion of Control, IoC) і Впровадження залежностей (Dependency Injection, DI) – це патерни, які сприяють слабкому зв'язуванню компонентів, полегшуючи тестування і заміну реалізацій. Spring Framework створений навколо цих принципів, надаючи контейнер IoC, який керує життєвим циклом об'єктів (бінів) і їх залежностями. В контексті MasterPoint ці патерни дозволяють створити гнучку архітектуру, де, наприклад, сервіс пошуку майстрів може легко замінити свою реалізацію без впливу на інші компоненти системи.

Репозиторій (Repository) – це патерн, який надає абстракцію для доступу до даних, приховуючи деталі збереження та отримання об'єктів. Spring Data JPA реалізує цей патерн, дозволяючи розробникам визначати інтерфейси репозиторіїв без необхідності писати реалізації. Для MasterPoint репозиторії можуть бути створені для кожної основної сутності: користувачів, замовлень, відгуків тощо. Цей патерн забезпечує чистий інтерфейс між бізнес-логікою і шаром доступу до даних, полегшує тестування і дозволяє змінювати деталі зберігання без впливу на бізнес-логіку.

Команда-Запит-Розподіл відповідальності (Command Query Responsibility Segregation, CQRS) розділяє операції запису (команди) і читання (запити) на різні моделі. Це особливо корисно для систем з високим навантаженням, де розділення

може підвищити продуктивність і масштабованість. Для платформи MasterPoint CQRS може бути застосований для розділення операцій створення і редагування замовлень (команди) від операцій пошуку і фільтрації майстрів (запити), які можуть вимагати оптимізованих структур даних і запитів. Матеріалізований вид (Materialized View) – це патерн, який зберігає результати складних запитів у окремій таблиці або кеші для швидкого доступу. Це особливо корисно для операцій, які вимагають агрегації даних з різних джерел. У контексті MasterPoint цей патерн може бути використаний для реалізації швидкого пошуку майстрів за різними критеріями (регіон, категорія, рейтинг тощо), зберігаючи попередньо обчислені результати в окремій структурі даних.

API Gateway – це патерн, який надає єдину точку входу для клієнтів, приховуючи внутрішню структуру системи. Gateway може виконувати такі функції, як маршрутизація запитів, аутентифікація і авторизація, моніторинг, обмеження швидкості тощо. Для MasterPoint використання API Gateway, такого як Spring Cloud Gateway або Zuul, може спростити взаємодію клієнтської частини з різними мікросервісами і забезпечити єдиний інтерфейс для зовнішніх клієнтів.

Кеш-асаїд (Cache-Aside) – це патерн кешування, де додаток спочатку перевіряє кеш перед зверненням до основного джерела даних. Якщо дані не знайдені в кеші, вони отримуються з джерела даних і додаються до кешу для майбутніх запитів. Spring підтримує кешування через Spring Cache, який може бути інтегрований з різними провайдерами кешу, такими як Redis, Ehcache, Caffeine тощо. Для MasterPoint кешування може значно покращити продуктивність частих операцій, таких як отримання списку категорій, регіонів, популярних майстрів тощо.

Шаблонний метод (Template Method) – це поведінковий патерн, який визначає скелет алгоритму, дозволяючи підкласам перевизначати певні кроки без зміни загальної структури. В Spring цей патерн широко використовується в таких компонентах, як JdbcTemplate, RestTemplate, TransactionTemplate тощо. Для MasterPoint шаблонний метод може бути використаний для стандартизації

обробки різних типів запитів або операцій, таких як створення замовлення, обробка платежів, відправка повідомлень тощо.

Спостерігач (Observer) – це патерн, де об'єкт (спостерігач) автоматично отримує повідомлення про зміни стану іншого об'єкта (суб'єкта). В контексті веб-додатків цей патерн часто реалізується через механізми подій. Spring підтримує цей патерн через ApplicationEventPublisher і ApplicationListener. Для MasterPoint патерн спостерігача може бути корисним для реалізації сповіщень (наприклад, клієнти можуть отримувати сповіщення, коли майстер відповідає на їхнє замовлення, а майстри – коли з'являються нові замовлення в їхньому регіоні та категорії) [8].

Стратегія (Strategy) – це патерн, який дозволяє вибирати алгоритм під час виконання з сімейства алгоритмів, інкапсульованих в окремі класи. В Spring цей патерн часто використовується для реалізації різних стратегій аутентифікації, обробки запитів, валідації тощо. У контексті MasterPoint патерн стратегії може бути застосований для реалізації різних алгоритмів пошуку і сортування майстрів, різних методів оплати, різних способів сповіщення користувачів тощо.

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ "MASTERPOINT"

2.1 Функціональні та нефункціональні вимоги до системи

Функціональні вимоги визначають, які конкретні функції та можливості повинна надавати система користувачам табл. 2.1. Для платформи MasterPoint, орієнтованої на пошук майстрів та забезпечення їх взаємодії з клієнтами, ключові функціональні вимоги мають охоплювати всі аспекти використання системи різними категоріями користувачів. Правильне визначення функціональних вимог є критично необхідним для успішної реалізації проекту, оскільки вони формують основу для розробки та тестування програмного продукту.

Таблиця 2.1 Функціональні вимоги до системи MasterPoint за пріоритетами

Пріоритет	**Функціональні вимоги**	**Категорія користувачів**	**Складність реалізації**
Критичний	Реєстрація та автентифікація користувачів	Всі	Середня
Критичний	Створення та управління замовленнями	Клієнти	Висока
Критичний	Пошук та фільтрація замовлень	Майстри	Висока
Критичний	Відгуки на замовлення та комунікація	Клієнти, Майстри	Середня
Високий	Вибір майстра та укладання угоди	Клієнти	Середня
Високий	Рейтинги та відгуки	Клієнти, Майстри	Середня
Високий	Адміністрування системи	Адміністратори	Висока
Середній	Система сповіщень	Всі	Середня
Середній	Геолокація та відображення на карті	Клієнти, Майстри	Висока
Низький	Інтеграція з платіжними системами	Клієнти, Майстри	Висока
Низький	Аналітичні інструменти	Клієнти, Майстри, Адміністратори	Середня

Продовження таблиці 2.1

Низький	API для інтеграції з іншими системами	Всі	Висока
Низький	Імпорт та експорт даних	Всі	Низька
Низький	Підтримка мультимовності та локалізації	Всі	Середня

Реєстрація та автентифікація користувачів є базовою функціональною вимогою для системи MasterPoint. Система повинна забезпечити можливість реєстрації трьох типів користувачів: клієнтів, майстрів та адміністраторів. Процес реєстрації має включати збір необхідної інформації, валідацію введених даних, підтвердження електронної пошти та створення профілю користувача. Автентифікація повинна підтримувати різні методи входу, включаючи електронну пошту з паролем, а також, опціонально, авторизацію через соціальні мережі. Система також має забезпечувати функціонал відновлення паролю та управління сесіями користувачів.

Управління профілями користувачів є головною функціональною вимогою, що дозволяє налаштовувати особисту інформацію та параметри. Для клієнтів система має забезпечувати можливість редагування особистих даних, налаштування параметрів приватності та способів комунікації. Для майстрів додатково потрібно забезпечити можливість створення детального профілю з інформацією про спеціалізацію, досвід, регіон роботи, опис послуг та ціни, а також завантаження портфоліо виконаних робіт. Адміністратори повинні мати можливість управляти профілями всіх користувачів, включаючи їх блокування або видалення в разі необхідності.

Створення та управління замовленнями є центральною функціональною вимогою для клієнтів системи. Користувачі повинні мати можливість створювати нові замовлення, вказуючи категорію послуг, детальний опис завдання, бажані терміни виконання, бюджет та місце проведення робіт. Функціонал повинен дозволяти редагування та скасування створених замовлень, а також перегляд статусу їх виконання. Клієнти мають отримувати сповіщення про відгуки майстрів

на їхні замовлення та мати можливість обрати відповідного виконавця серед тих, хто відгукнувся.

Пошук та фільтрація замовлень є ключовою функціональною вимогою для майстрів. Система повинна надавати інструменти для пошуку релевантних замовлень на основі категорії послуг, регіону, бюджету та інших параметрів. Майстри повинні мати можливість налаштувати фільтри для відображення тільки тих замовлень, які відповідають їхній спеціалізації та регіону роботи. Для зручності користувачів необхідно реалізувати сортування результатів пошуку за різними критеріями, такими як дата створення, терміновість, бюджет тощо. Система також повинна надавати рекомендації щодо замовлень, які найбільш відповідають профілю конкретного майстра.

Відгуки на замовлення та комунікація між користувачами є головними функціональними вимогами для забезпечення взаємодії між клієнтами та майстрами. Майстри повинні мати можливість переглядати деталі замовлень та залишати відгуки з пропозиціями щодо виконання роботи, включаючи вартість, терміни та інші умови. Система має забезпечувати зручний інтерфейс для обміну повідомленнями між клієнтом та майстром, включаючи можливість прикріплення файлів. Комунікація повинна зберігатися в системі для можливості подальшого перегляду та вирішення спірних ситуацій.

Вибір майстра та укладання угоди є функціональною вимогою, яка дозволяє формалізувати відносини між клієнтом та виконавцем. Клієнт повинен мати можливість обрати майстра серед тих, хто відгукнувся на замовлення, на основі запропонованих умов, рейтингу та відгуків інших користувачів. Система має забезпечувати формування простої угоди з основними умовами виконання робіт, яку обидві сторони можуть прийняти. Після вибору майстра замовлення має змінювати свій статус, і всі сторони повинні отримувати відповідні сповіщення.

Рейтинги та відгуки є функціональними вимогами, які забезпечують формування репутації користувачів. Після завершення робіт клієнти повинні мати можливість оцінити якість послуг майстра за різними критеріями (якість роботи, дотримання термінів, комунікація тощо) та залишити текстовий відгук. Майстри

також мають отримати можливість оцінити клієнта як замовника. Рейтинг користувача повинен розраховуватися на основі всіх отриманих оцінок і бути доступним для перегляду в профілі. Система повинна забезпечувати захист від фальшивих відгуків та маніпуляцій з рейтингом [9].

Система сповіщень є головними функціональним компонентом, який забезпечує своєчасне інформування користувачів про важливі події. Система повинна підтримувати різні типи сповіщень: нові відгуки на замовлення, повідомлення від інших користувачів, зміна статусу замовлення, нагадування про терміни тощо. Користувачі мають отримати можливість налаштовувати способи отримання сповіщень (електронна пошта, SMS, push-сповіщення в браузері) та їх періодичність. Для зручності користувачів необхідно реалізувати центр сповіщень з можливістю перегляду історії та відмітки про прочитання.

Адміністрування системи є функціональною вимогою, яка забезпечує контроль та управління платформою. Адміністратори повинні мати доступ до панелі управління з можливістю перегляду статистики використання системи, управління користувачами (блокування, видалення, зміна ролей), модерації контенту (перевірка відгуків, скарг), управління категоріями послуг та іншими довідниками. Система повинна забезпечувати логування дій адміністраторів для аудиту та безпеки.

Геолокація та відображення на карті є функціональною вимогою, яка покращує користувацький досвід. Система повинна підтримувати визначення географічного розташування користувачів (з їхньої згоди) та відображення майстрів на карті. Клієнти мають отримати можливість шукати майстрів у певному радіусі від заданої точки або свого поточного місцезнаходження. Для реалізації цієї функціональності необхідна інтеграція з картографічними сервісами та API геолокації.

Інтеграція з платіжними системами є функціональною вимогою для забезпечення фінансових транзакцій. Система повинна підтримувати різні способи оплати послуг, включаючи банківські карти, електронні гаманці та, можливо, інші методи. Необхідно реалізувати функціонал для безпечного

збереження платіжних реквізитів, проведення транзакцій та формування чеків. Також потрібно передбачити механізми для роботи з депозитами, частковими платежами та поверненнями коштів у випадку спірних ситуацій.

Аналітичні інструменти є функціональною вимогою, яка дозволяє користувачам отримувати інформацію про їхню активність. Майстри повинні мати доступ до статистики переглядів їх профілю, кількості отриманих замовлень, рівня конверсії та динаміки рейтингу. Клієнти можуть отримувати аналітику щодо використання системи, включаючи історію замовлень, витрат та взаємодії з майстрами. Адміністратори повинні мати доступ до загальної аналітики платформи, включаючи статистику реєстрацій, активності користувачів, кількості замовлень та їх розподілу за категоріями та регіонами.

API для інтеграції з іншими системами є функціональною вимогою, яка розширює можливості використання платформи. Система повинна надавати програмний інтерфейс для інтеграції з CRM-системами, маркетинговими інструментами, аналітичними платформами та іншими зовнішніми сервісами. API має бути добре документованим, безпечним та масштабованим для підтримки різних сценаріїв використання.

Імпорт та експорт даних є функціональною вимогою, яка забезпечує інтероперабельність системи. Користувачі повинні мати можливість експортувати свої дані (історію замовлень, повідомлень, відгуків) у стандартних форматах, таких як CSV, XML або JSON. Майстри мають отримати можливість імпортувати інформацію про свої послуги з інших систем або таблиць. Адміністратори повинні мати інструменти для масового імпорту та експорту даних для цілей міграції, резервного копіювання чи аналізу.

Підтримка мультимовності та локалізації є функціональною вимогою для забезпечення доступності системи для користувачів з різних регіонів. Інтерфейс повинен підтримувати переклад на різні мови, з можливістю легкого додавання нових мовних версій. Система також має враховувати регіональні особливості, такі як формат дат, валюта, одиниці вимірювання тощо [10]. Контент, створений

користувачами, може залишатися в оригінальній мові, але система може надавати рекомендації щодо використання мов, популярних у певному регіоні.

Нефункціональні вимоги визначають якісні характеристики системи, такі як продуктивність, безпека, масштабованість тощо. Для платформи MasterPoint головне визначити такі нефункціональні вимоги: система повинна забезпечувати високу доступність (не менше 99.9% часу); час відгуку на запити користувачів має не перевищувати 2 секунди для основних операцій; система повинна підтримувати одночасну роботу не менше 10 000 користувачів без деградації продуктивності; дані користувачів мають бути захищені відповідно до вимог законодавства про захист персональних даних; інтерфейс повинен бути адаптивним для різних пристроїв, включаючи мобільні телефони та планшети.

2.2 Проектування бази даних та структури проекту

Проектування бази даних є одним із ключових аспектів розробки веб-додатку, особливо для системи, яка вимагає зберігання та обробки значних обсягів структурованих даних, таких як платформа MasterPoint. Правильно спроектована база даних забезпечує ефективне зберігання інформації, швидкий доступ до неї та підтримує цілісність даних. Для розробки системи MasterPoint було обрано реляційну базу даних, оскільки вона найкраще відповідає вимогам щодо структурованості даних, підтримки транзакцій та зв'язків між сутностями.

Центральними сутностями в моделі даних MasterPoint є User (Користувач), який може мати одну з трьох ролей: Client (Клієнт), Master (Майстер) або Admin (Адміністратор). Сутність User містить основну інформацію про користувача, таку як email, ім'я, захешований пароль та роль. Для кожної ролі створено окрему таблицю (ClientProfile, MasterProfile), яка пов'язана з основною таблицею User відношенням "один до одного". Такий підхід дозволяє зберігати специфічну інформацію для кожного типу користувачів, зберігаючи при цьому загальні дані у єдиній таблиці User.

Таблиця MasterProfile містить розширену інформацію про майстрів, включаючи опис їхніх послуг, регіон роботи, досвід, фото та інші атрибути, які

важливі для представлення майстра на платформі. Також ця таблиця зберігає зв'язок з категоріями послуг, які надає майстер, через проміжну таблицю master_categories (зв'язок "багато до багатьох") [11]. Категорії послуг зберігаються в окремій таблиці Category, яка містить назву, опис та ієрархічну структуру (через поле parent_id, що вказує на батьківську категорію).

Таблиця ClientProfile містить інформацію, специфічну для клієнтів, таку як історія замовлень, рейтинг як замовника, контактні дані для виконавців тощо. Хоча ця таблиця має менше полів порівняно з MasterProfile, вона все одно головна для підтримки функціоналу платформи, пов'язаного зі створенням замовлень та взаємодією з майстрами.

Замовлення (Order) є ще однією центральною сутністю в системі. Таблиця Order містить інформацію про замовлення клієнта, включаючи категорію робіт, опис завдання, бюджет, бажану дату виконання, регіон та статус (активне, закрите тощо). Ця таблиця має зв'язок з таблицею ClientProfile (клієнт, який створив замовлення) через зовнішній ключ client_id. Також вона може мати зв'язок з таблицею MasterProfile через поле selected_master_id, яке вказує на майстра, обраного для виконання замовлення (це поле може бути null для замовлень, де майстра ще не обрано).

Відгуки майстрів на замовлення зберігаються в таблиці OrderResponse, яка містить зв'язки з таблицями Order (замовлення, на яке відгукнувся майстер) та MasterProfile (майстер, який залишив відгук). Ця таблиця також містить текст повідомлення, яке майстер залишив для клієнта, та додаткову інформацію, таку як запропонована ціна та терміни виконання. Один майстер може залишити тільки один відгук на конкретне замовлення, що забезпечується унікальним індексом на полях order_id та master_id.

Рейтинги та відгуки клієнтів про майстрів зберігаються в таблиці Review, яка містить оцінку (зазвичай за шкалою від 1 до 5), текстовий відгук та дату створення. Ця таблиця має зв'язки з таблицями MasterProfile (майстер, якого оцінюють) та ClientProfile (клієнт, який залишає відгук), а також з таблицею Order (замовлення, на основі якого залишається відгук). Аналогічно, відгуки майстрів

про клієнтів можуть зберігатися в окремій таблиці або в тій самій таблиці з додатковим полем для визначення напрямку відгуку.

Для забезпечення комунікації між користувачами створено таблиці Message (для зберігання індивідуальних повідомлень) та Conversation (для групування повідомлень у розмови). Таблиця Message містить текст повідомлення, дату надсилання, зв'язки з відправником та отримувачем (через таблицю User), а також зв'язок з конкретною розмовою. Таблиця Conversation, у свою чергу, зберігає загальну інформацію про розмову, таку як її тема, дату створення та, опціонально, зв'язок із замовленням, якщо розмова стосується конкретного замовлення.

Для реалізації системи сповіщень створено таблицю Notification, яка містить тип сповіщення, його зміст, дату створення, статус (прочитано/не прочитано) та зв'язок з користувачем-отримувачем. Додатково може бути створена таблиця NotificationSettings для зберігання налаштувань користувачів щодо отримання різних типів сповіщень через різні канали (email, SMS, веб-сповіщення).

Для підтримки функціональності геолокації створено таблиці Region (для зберігання інформації про регіони, міста тощо) та, опціонально, GeoLocation для зберігання конкретних географічних координат (широта, довгота) [12]. Таблиця Region може мати ієрархічну структуру (через поле parent_id) для представлення адміністративних одиниць різних рівнів (країна, область, місто тощо).

Для забезпечення безпеки та аудиту створено таблиці AuditLog (для логування дій користувачів у системі) та LoginAttempt (для відстеження спроб входу в систему, включаючи невдалі спроби). Ці таблиці допомагають виявляти підозрілу активність та розслідувати інциденти безпеки.

Для оптимізації запитів до бази даних створено відповідні індекси. Наприклад, для таблиці Order створено індекси на полях category_id, region, preferred_date для прискорення пошуку замовлень за цими критеріями. Також створено складений індекс на полях category_id, region, closed для швидкого пошуку активних замовлень у певній категорії та регіоні.

Структура проекту Spring Boot для системи MasterPoint організована згідно з принципами модульності та розділення відповідальності. На верхньому рівні

проект розділений на кілька основних пакетів: com.example.masterpoint.model (для класів-сутностей JPA), com.example.masterpoint.repository (для інтерфейсів доступу до бази даних), com.example.masterpoint.service (для бізнес-логіки), com.example.masterpoint.controller (для обробки HTTP-запитів), com.example.masterpoint.config (для конфігурації), com.example.masterpoint.security (для компонентів безпеки) та com.example.masterpoint.util (для утилітних класів).

Пакет model містить класи-сутності, які представляють таблиці бази даних. Кожен клас анотований з @Entity та містить поля, які відповідають колонкам таблиці, з відповідними анотаціями JPA для визначення зв'язків між таблицями (@OneToOne, @OneToMany, @ManyToOne, @ManyToMany). Класи також містять гетери та сетери для доступу до полів. Для представлення статичних даних, таких як статуси замовлень або категорії послуг, використовуються енумерації (enum).

Пакет repository містить інтерфейси для доступу до бази даних, які розширюють JpaRepository або CrudRepository. Ці інтерфейси надають базові методи для операцій CRUD, а також можуть містити кастомні методи, оголошені за допомогою анотації @Query або іменних конвенцій Spring Data JPA. Наприклад, інтерфейс OrderRepository може містити метод findByCategoryInAndRegionAndClosedFalse для пошуку активних замовлень у певних категоріях та регіоні.

Пакет service містить класи, які реалізують бізнес-логіку додатка. Кожен сервіс відповідає за певну функціональну область, наприклад, UserService, OrderService, NotificationService тощо. Ці класи використовують репозиторії для доступу до даних, але також містять додаткову логіку, таку як валідація, обробка бізнес-правил, перетворення даних тощо. Сервіси анотовані з @Service та впроваджуються в інші компоненти через механізм залежностей Spring.

Пакет controller містить класи, які обробляють HTTP-запити та формують відповіді. Контролери анотовані з @Controller (для класичних контролерів, які повертають шаблони) або @RestController (для RESTful API, які повертають дані у форматі JSON). Методи контролерів анотовані з @GetMapping, @PostMapping тощо для визначення типу HTTP-запиту та шляху [13]. Контролери

використовують сервіси для виконання бізнес-операцій та формування відповідей клієнтам.

Пакет config містить класи для конфігурації різних аспектів додатка, таких як налаштування бази даних, безпеки, кешування, асинхронного виконання завдань тощо. Ці класи зазвичай анотовані з @Configuration та містять методи, анотовані з @Bean для створення та налаштування компонентів Spring.

2.3 Реалізація моделей даних та бізнес-логіки

Реалізація моделей даних для системи MasterPoint базується на використанні Java Persistence API (JPA) у поєднанні з Hibernate як провайдера ORM (Object-Relational Mapping). Цей підхід дозволяє представити таблиці бази даних у вигляді Java-класів, полегшуючи роботу з даними та забезпечуючи об'єктно-орієнтований підхід до проектування. Кожен клас-сутність анотований за допомогою @Entity, що вказує на його відповідність таблиці в базі даних рис. 2.1.

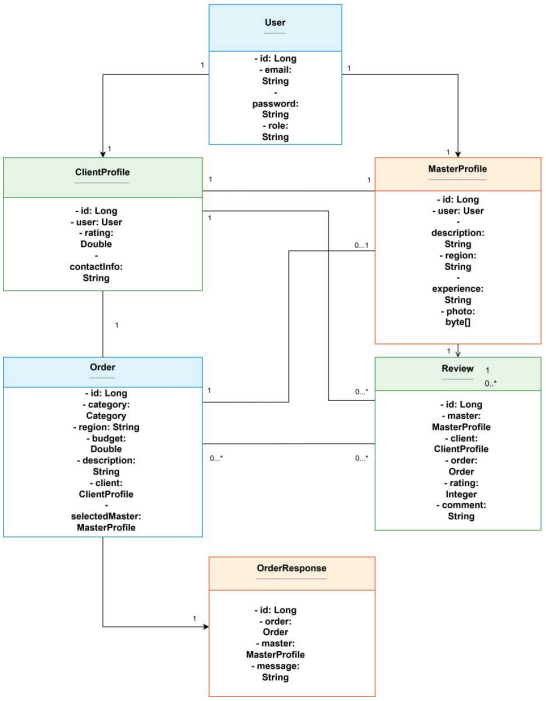

Рисунок 2.1 - Діаграма класів MasterPoint

Центральною моделлю системи є клас User, який представляє користувача платформи. Цей клас містить основні атрибути, такі як ідентифікатор, електронна пошта, пароль та роль. Ідентифікатор анотований як @Id для вказівки на

первинний ключ та @GeneratedValue для автоматичної генерації значень. Поле електронної пошти має обмеження унікальності, що забезпечується анотацією @Column(unique = true). Роль користувача представлена у вигляді рядка, хоча в реальній системі може використовуватися більш складний механізм управління ролями та дозволами.

Для представлення профілів різних типів користувачів створено класи ClientProfile та MasterProfile. Обидва класи мають зв'язок "один-до-одного" з класом User, що реалізується через анотацію @OneToOne та відповідне поле для зберігання зовнішнього ключа. У класі MasterProfile містяться додаткові поля для зберігання інформації про регіон роботи, опис послуг, досвід майстра та його фотографію. Для зберігання фото використовується тип byte[] з анотацією @Lob, що вказує на зберігання великого об'єкта.

Категорії послуг представлені в системі за допомогою перерахування (enum) Category, яке містить набір заздалегідь визначених категорій, таких як САНТЕХНІКА, ЕЛЕКТРИКА, РЕМОНТ тощо. Таке рішення спрощує роботу з категоріями, але обмежує можливість додавання нових категорій без зміни коду. В більш гнучкій реалізації категорії могли б бути представлені окремим класом-сутністю з можливістю зберігання ієрархії категорій [14].

Для зберігання інформації про замовлення в системі використовується клас Order. Цей клас містить поля для зберігання категорії послуг, регіону, бюджету, бажаної дати виконання, опису завдання, а також зв'язки з клієнтом (через поле типу ClientProfile) та обраним майстром (через поле типу MasterProfile, яке може бути null для замовлень, де майстра ще не обрано). Також клас містить булеве поле closed, яке вказує на статус замовлення.

Відгуки майстрів на замовлення представлені класом OrderResponse. Цей клас містить зв'язки з замовленням та майстром, який залишив відгук, а також текст повідомлення. Зв'язки реалізовані через анотації @ManyToOne, оскільки одне замовлення може мати багато відгуків, і один майстер може залишити відгуки на багато замовлень. Для забезпечення унікальності відгуків (один

майстер може залишити тільки один відгук на конкретне замовлення) можна додатково використати анотацію @UniqueConstraint на рівні класу.

Бізнес-логіка системи реалізована через сервіси , які є проміжним шаром між контролерами та репозиторіями. Основним сервісом є UserService, який відповідає за реєстрацію користувачів різних типів, аутентифікацію та управління профілями. Цей сервіс використовує UserRepository для доступу до даних та PasswordEncoder для шифрування паролів. Метод registerClient створює нового користувача з роллю CLIENT та відповідний профіль клієнта, а метод registerMaster створює користувача з роллю MASTER та профіль майстра з додатковою інформацією [15]. Сервіс OrderService відповідає за створення та управління замовленнями рис. 2.2. Він містить методи для створення нових замовлень, отримання списків замовлень за різними критеріями, вибору майстра для виконання замовлення тощо. Метод createOrder приймає об'єкт замовлення та електронну пошту клієнта, знаходить відповідного користувача, встановлює зв'язок між замовленням та клієнтом і зберігає замовлення в базі даних. Метод getRelevantOrdersForMaster повертає список замовлень, що відповідають регіону та категоріям послуг конкретного майстра.

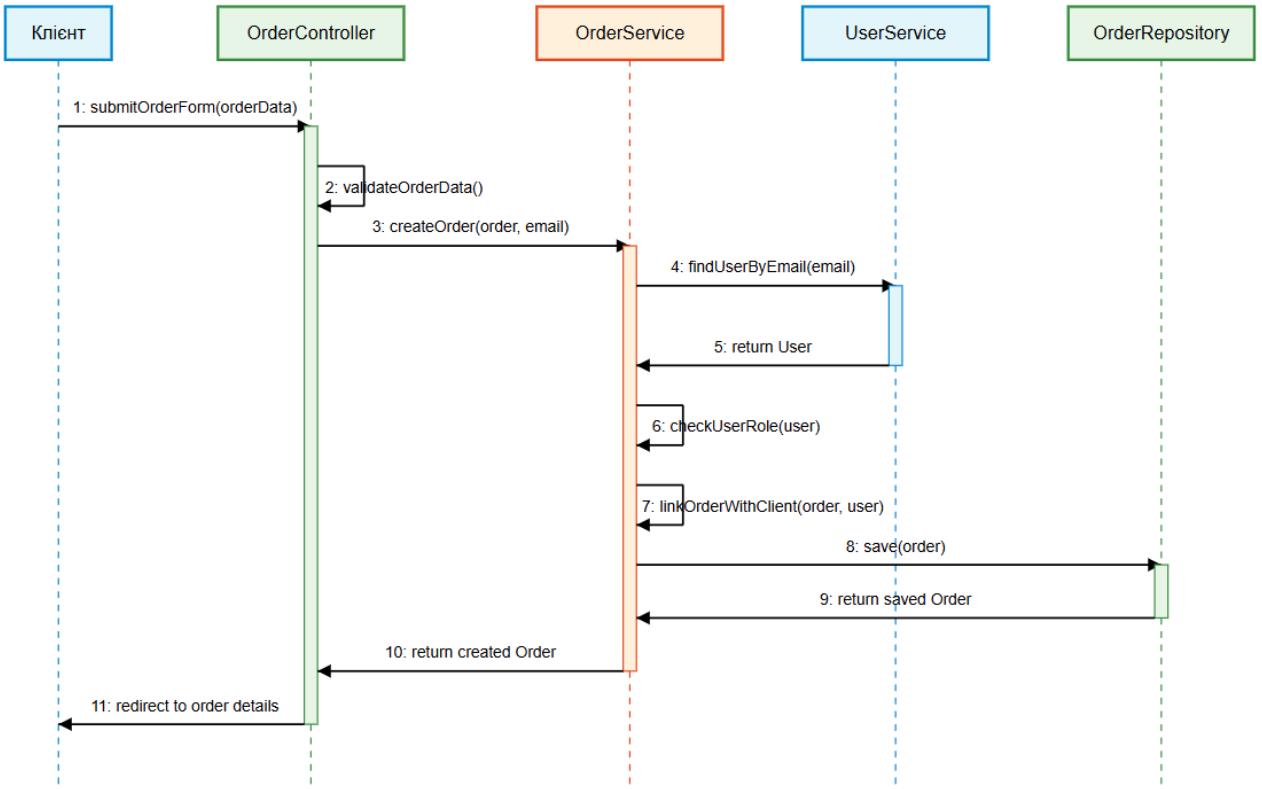

Рисунок 2.2 - Діаграма послідовності для створення замовлення

Тепер розглянемо всю структуру проекту рис. 2.3. Метод respondToOrder в сервісі OrderService дозволяє майстру залишити відгук на замовлення. Він приймає ідентифікатор замовлення, електронну пошту майстра та текст повідомлення, створює новий об'єкт OrderResponse з відповідними зв'язками та зберігає його в базі даних. Цей метод також може містити додаткову логіку для перевірки, чи не залишав майстер відгук раніше, чи активне замовлення тощо.

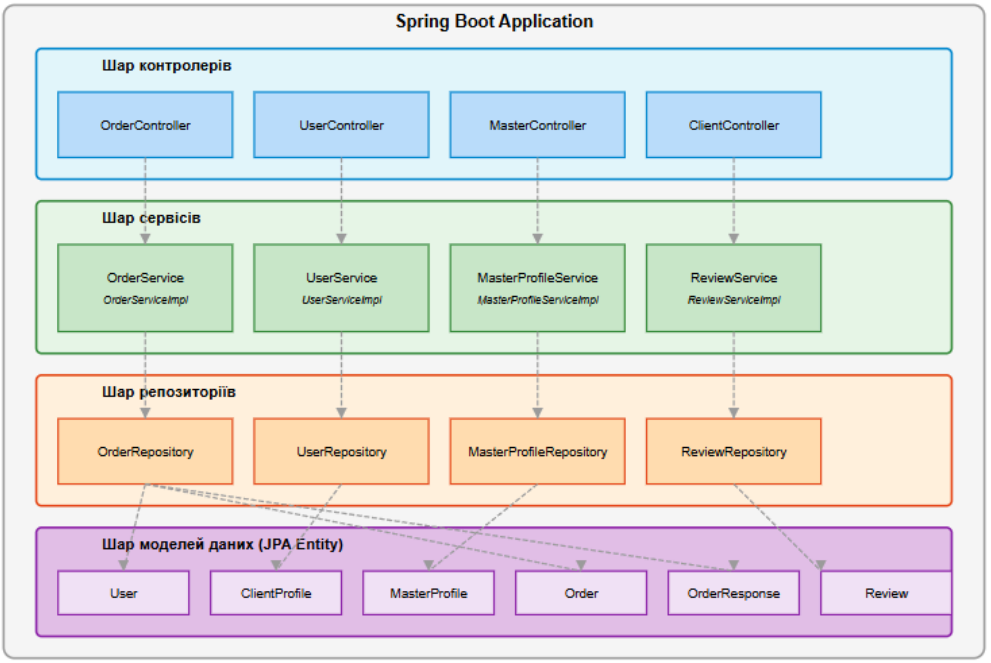

Рисунок 2.3 - Діаграма компонентів системи MasterPoint з відображенням багатошарової архітектури

Метод chooseMaster дозволяє клієнту обрати майстра для виконання замовлення. Він приймає ідентифікатори замовлення та майстра, встановлює зв'язок між замовленням та обраним майстром та змінює статус замовлення на "закрите". В реальній системі цей метод може містити додаткову логіку для формування угоди, сповіщення учасників, оновлення статистики тощо.

Абстрактний метод rejectMaster в сервісі OrderService призначений для реалізації в дочірніх класах і дозволяє клієнту відхилити відгук майстра. В конкретній реалізації OrderServiceImpl цей метод видаляє відгук майстра на замовлення. Використання абстрактного методу дозволяє гнучко змінювати реалізацію цієї функціональності без зміни інтерфейсу сервісу.

Сервіс MasterProfileService відповідає за роботу з профілями майстрів. Він містить методи для отримання профілю за електронною поштою користувача та оновлення інформації в профілі. Метод getProfile знаходить користувача за електронною поштою та повертає його профіль майстра, якщо такий існує. Метод updateProfile оновлює інформацію в профілі майстра, включаючи регіон роботи, опис послуг, категорії послуг та фото.

Сервіс ClientService відповідає за функціональність, пов'язану з клієнтами. Він використовує UserRepository для доступу до даних про користувачів та OrderService для роботи з замовленнями. Метод getOrdersByClientEmail повертає список замовлень, створених конкретним клієнтом. Він знаходить користувача за електронною поштою, перевіряє наявність профілю клієнта та використовує OrderService для отримання замовлень за ідентифікатором клієнта [16]. Для подальшого розвитку системи можуть бути реалізовані додаткові сервіси, такі як NotificationService для управління сповіщеннями, MessageService для комунікації між користувачами, ReviewService для роботи з відгуками та рейтингами, PaymentService для інтеграції з платіжними системами тощо. Кожен з цих сервісів буде відповідати за певну функціональну область та використовувати відповідні репозиторії для доступу до даних.

Також вагомим аспектом реалізації бізнес-логіки є обробка транзакцій. У Spring Boot транзакції можуть бути керовані через анотацію @Transactional на рівні методів або класів. Ця анотація забезпечує атомарність операцій, тобто або всі операції в межах транзакції виконуються успішно, або жодна з них. Це особливо важливо для операцій, які змінюють пов'язані дані в різних таблицях, наприклад, створення користувача та його профілю, вибір майстра для замовлення тощо.

2.4 Розробка контролерів та REST API

Розробка контролерів та REST API є критичним етапом створення веб-додатку, оскільки саме ці компоненти забезпечують інтерфейс взаємодії між клієнтською частиною та серверною бізнес-логікою. У системі MasterPoint контролери реалізовані з використанням Spring MVC, який є частиною фреймворку Spring Boot. Контролери обробляють HTTP-запити, викликають відповідні методи сервісів для виконання бізнес-операцій і формують відповіді для клієнтів.

Головним контролером системи є OrderController, який відповідає за функціональність, пов'язану з замовленнями. Цей контролер анотований з

@Controller і містить базовий маршрут @RequestMapping("/orders"), що визначає префікс для всіх URL-шляхів, пов'язаних з замовленнями. Контролер використовує інжекцію залежностей для отримання екземплярів сервісів OrderService, OrderResponseService та UserService, які необхідні для реалізації бізнес-логіки.

Метод createOrderForm контролера OrderController обробляє GET-запити на шлях /orders/create і відповідає за відображення форми створення нового замовлення. Він використовує об'єкт Model для передачі порожнього об'єкта Order у шаблон, який буде використаний для заповнення форми. Метод повертає ім'я шаблону "createOrder", який буде оброблений шаблонізатором Thymeleaf для генерації HTML-сторінки [17].

Метод createOrder контролера OrderController обробляє POST-запити на той самий шлях /orders/create і відповідає за обробку даних, введених користувачем у форму створення замовлення. Він приймає об'єкт Order, заповнений даними з форми, та об'єкт HttpServletRequest для отримання інформації про аутентифікованого користувача. Метод отримує електронну пошту користувача з об'єкта Principal, який представляє аутентифікованого користувача, і використовує сервіс OrderService для створення нового замовлення. Після успішного створення замовлення, метод перенаправляє користувача на сторінку кабінету клієнта.

Метод respondToOrder контролера OrderController обробляє POST-запити на шлях /orders/{orderId}/respond, де {orderId} є ідентифікатором замовлення. Цей метод дозволяє майстрам залишати відгуки на замовлення. Він приймає ідентифікатор замовлення як параметр шляху, текст повідомлення як параметр запиту і об'єкт HttpServletRequest для отримання інформації про аутентифікованого користувача. Метод використовує сервіс OrderService для створення нового відгуку на замовлення і перенаправляє користувача на сторінку кабінету майстра.

Метод chooseMaster контролера OrderController обробляє POST-запити на шлях /orders/{orderId}/choose/{masterId}, де {orderId} та {masterId} є ідентифікаторами замовлення та майстра відповідно. Цей метод дозволяє клієнтам

обирати майстра для виконання замовлення. Він використовує сервіс OrderService для встановлення зв'язку між замовленням та обраним майстром і перенаправляє користувача на сторінку кабінету клієнта.

Метод rejectMaster контролера OrderController обробляє POST-запити на шлях /orders/{orderId}/reject/{masterId} і дозволяє клієнтам відхиляти відгуки майстрів. Він використовує сервіс OrderService для видалення відгуку майстра на замовлення і перенаправляє користувача на сторінку деталей замовлення.

Метод viewOrder контролера OrderController обробляє GET-запити на шлях /orders/{id} і відповідає за відображення деталей конкретного замовлення. Він приймає ідентифікатор замовлення як параметр шляху, об'єкт Model для передачі даних у шаблон і об'єкт Principal для отримання інформації про аутентифікованого користувача. Метод отримує замовлення за ідентифікатором, список відгуків на нього, визначає, чи є аутентифікований користувач клієнтом, який створив замовлення, та чи залишав він вже відгук на це замовлення, і передає цю інформацію у шаблон для відображення [18]. Контролер MasterController відповідає за функціональність, пов'язану з майстрами. Він анотований з @Controller і має базовий маршрут @RequestMapping("/master"). Контролер використовує сервіси MasterProfileService та OrderService для реалізації бізнес-логіки.

Метод dashboard контролера MasterController обробляє GET-запити на шлях /master/dashboard і відповідає за відображення кабінету майстра з списком релевантних замовлень. Він отримує електронну пошту аутентифікованого користувача, використовує сервіс OrderService для отримання списку замовлень, що відповідають регіону та категоріям послуг майстра, і передає цей список у шаблон "master-dashboard".

Метод profile контролера MasterController обробляє GET-запити на шлях /master/profile і відповідає за відображення профілю майстра. Він отримує електронну пошту аутентифікованого користувача, використовує сервіс MasterProfileService для отримання профілю майстра і передає його у шаблон "masterProfile".

Метод updateProfile контролера MasterController обробляє POST-запити на шлях /master/profile і відповідає за оновлення інформації в профілі майстра. Він приймає об'єкт MasterProfile, заповнений даними з форми, використовує сервіс MasterProfileService для оновлення профілю і перенаправляє користувача на сторінку кабінету майстра.

Метод respond контролера MasterController обробляє POST-запити на шлях /master/respond/{orderId} і є альтернативним способом для майстрів залишати відгуки на замовлення. Він приймає ідентифікатор замовлення як параметр шляху, текст повідомлення як параметр запиту і об'єкт HttpServletRequest для отримання інформації про аутентифікованого користувача. Метод використовує сервіс OrderService для створення нового відгуку на замовлення і перенаправляє користувача на сторінку кабінету майстра.

Контролер ClientController відповідає за функціональність, пов'язану з клієнтами. Він анотований з @Controller і має базовий маршрут @RequestMapping("/client"). Контролер використовує сервіси OrderService та ClientService для реалізації бізнес-логіки.

Метод dashboard контролера ClientController обробляє GET-запити на шлях /client/dashboard і відповідає за відображення кабінету клієнта з списком його замовлень. Він отримує електронну пошту аутентифікованого користувача, використовує сервіс ClientService для отримання списку замовлень, створених цим клієнтом, і передає цей список у шаблон "client-dashboard" [19]. Методи createOrderForm та createOrder контролера ClientController є альтернативними способами для клієнтів створювати нові замовлення і працюють аналогічно відповідним методам контролера OrderController. Метод orderDetails обробляє GET-запити на шлях /client/order/{id} і відповідає за відображення деталей конкретного замовлення. Метод chooseMaster обробляє POST-запити на шлях /client/order/{id}/choose/{masterId} і є альтернативним способом для клієнтів обирати майстра для виконання замовлення.

Контролер AuthController відповідає за функціональність, пов'язану з аутентифікацією та реєстрацією користувачів. Він анотований з @Controller і не

має базового маршруту. Контролер використовує сервіс UserService для реалізації бізнес-логіки.

2.5 Реалізація системи авторизації та аутентифікації користувачів

Реалізація системи авторизації та аутентифікації користувачів є ключовим аспектом безпеки веб-додатку. У системі MasterPoint ця функціональність реалізована з використанням Spring Security – потужного та гнучкого фреймворку для захисту додатків на основі Spring. Spring Security забезпечує всебічний захист, включаючи аутентифікацію, авторизацію, захист від атак CSRF (Cross-Site Request Forgery), XSS (Cross-Site Scripting) та інших поширених вразливостей.

Конфігурація безпеки системи MasterPoint зосереджена в класі SecurityConfig, який анотований з @Configuration для позначення його як класу конфігурації Spring. Цей клас містить основні налаштування безпеки, включаючи правила доступу до різних URL-адрес, конфігурацію форми входу, параметри виходу з системи, тощо. Клас також надає бін PasswordEncoder, який використовується для шифрування паролів користувачів перед їх збереженням у базі даних.

Метод filterChain в класі SecurityConfig є основним методом конфігурації безпеки. Він приймає об'єкт HttpSecurity і налаштовує різні аспекти безпеки. Насамперед, метод налаштовує правила доступу до URL-адрес за допомогою методу authorizeHttpRequests. Публічні ресурси, такі як головна сторінка, сторінки входу та реєстрації, статичні ресурси (CSS, JavaScript), доступні всім користувачам без аутентифікації через виклик permitAll() [20]. Адреси, що починаються з /client/, доступні тільки користувачам з роллю CLIENT, адреси, що починаються з /master/, – тільки користувачам з роллю MASTER, а адреси, що починаються з /admin/, – тільки користувачам з роллю ADMIN.

Налаштування форми входу в систему виконується через метод formLogin. URL-адреса форми входу встановлюється як /login, і після успішного входу користувач перенаправляється на головну сторінку. Усі користувачі мають доступ до форми входу. Налаштування виходу з системи виконується через метод logout.

Після успішного виходу користувач перенаправляється на сторінку входу з параметром ?logout, який може бути використаний для відображення повідомлення про успішний вихід.

Для запобігання атакам CSRF у високозахищених середовищах використовується захист CSRF, хоча в деяких випадках, особливо під час розробки або для певних типів API, цей захист може бути вимкнений через виклик csrf().disable() [21]. У виробничому середовищі рекомендується залишати захист CSRF увімкненим.

Клас SecurityConfig також надає бін AuthenticationManager, який керує процесом аутентифікації користувачів. Цей бін використовується Spring Security для перевірки облікових даних користувачів під час входу в систему. Для шифрування паролів користувачів використовується BCryptPasswordEncoder – реалізація PasswordEncoder, яка використовує алгоритм BCrypt для створення захищених хешів паролів. BCrypt є рекомендованим алгоритмом шифрування паролів, оскільки він спеціально розроблений для повільного виконання, що ускладнює атаки перебору.

Клас UserDetailsServiceConfig відповідає за налаштування сервісу, який завантажує дані користувачів для аутентифікації. Цей клас надає бін UserDetailsService, який використовується Spring Security для отримання інформації про користувачів з бази даних. Реалізація цього сервісу шукає користувача за електронною поштою (яка використовується як ім'я користувача) у базі даних через UserRepository. Якщо користувач знайдений, створюється об'єкт UserDetails з електронною поштою, зашифрованим паролем та роллю користувача. Якщо користувач не знайдений, викидається виняток UsernameNotFoundException.

При реєстрації нових користувачів система MasterPoint забезпечує надійне збереження паролів. Метод registerClient в сервісі UserService отримує дані нового клієнта, шифрує пароль за допомогою PasswordEncoder, встановлює роль "CLIENT", створює профіль клієнта та зберігає дані в базі даних. Аналогічно, метод registerMaster реєструє нового майстра, шифрує пароль, встановлює роль "MASTER", пов'язує користувача з профілем майстра та зберігає дані.

Система також забезпечує захист конфіденційних операцій, таких як створення замовлень, відгуки на замовлення, вибір майстра для виконання замовлення, тощо. Ці операції доступні тільки аутентифікованим користувачам з відповідними ролями. Наприклад, тільки клієнти можуть створювати замовлення, тільки майстри можуть відгукуватися на замовлення, і тільки клієнт, який створив замовлення, може обрати майстра для його виконання [22].

У контролерах система використовує об'єкт Principal для отримання інформації про аутентифікованого користувача. Наприклад, метод createOrder в контролері OrderController отримує електронну пошту аутентифікованого користувача через request.getUserPrincipal().getName() і передає її сервісу OrderService для пов'язування замовлення з конкретним клієнтом.

Для надання контекстно-залежного інтерфейсу система використовує інформацію про аутентифікованого користувача при відображенні сторінок. Наприклад, метод viewOrder в контролері OrderController визначає, чи є аутентифікований користувач клієнтом, який створив замовлення, та чи залишав майстер вже відгук на це замовлення, і передає цю інформацію у шаблон для відображення відповідних елементів інтерфейсу. У шаблонах Thymeleaf використовуються спеціальні вирази для перевірки ролей користувачів та відображення відповідних елементів інтерфейсу. Наприклад, вираз th:if="${#authorization.expression('hasRole("CLIENT")')}" перевіряє, чи має аутентифікований користувач роль CLIENT, і відображає елементи інтерфейсу, доступні тільки клієнтам. Аналогічно, вирази для ролей MASTER та ADMIN дозволяють відображати елементи, доступні тільки майстрам та адміністраторам.

Система також забезпечує логування необхідних подій, пов'язаних з безпекою, таких як успішні та невдалі спроби входу, зміни ролей користувачів, блокування та розблокування облікових записів. Це допомагає виявляти підозрілу активність та розслідувати інциденти безпеки.

Для забезпечення безпеки даних під час передачі система використовує протокол HTTPS, який шифрує дані між клієнтом та сервером. Це захищає конфіденційну інформацію, таку як паролі, особисті дані користувачів, деталі

замовлень, від перехоплення під час передачі через мережу. Система також реалізує механізми для обмеження кількості невдалих спроб входу, блокування облікових записів після певної кількості невдалих спроб, та надсилання сповіщень про підозрілу активність. Це допомагає захистити облікові записи користувачів від атак методом перебору [23].

Для подальшого покращення безпеки система може бути розширена для підтримки двофакторної аутентифікації (2FA), яка вимагає від користувачів надати додатковий фактор аутентифікації, крім пароля, такий як код з мобільного додатку або SMS. Це значно підвищує безпеку облікових записів, навіть якщо пароль користувача був скомпрометований.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ "MASTERPOINT"

3.1 Інтерфейс користувача та його взаємодія з бекендом

Інтерфейс користувача системи MasterPoint розроблений відповідно до принципів зручності використання (UX) та привабливого візуального дизайну (UI), що забезпечує інтуїтивно зрозумілу взаємодію з платформою для всіх типів користувачів. Інтерфейс створений за допомогою фреймворку Thymeleaf у поєднанні з HTML5, CSS3 та JavaScript, що дозволяє створювати динамічні веб-сторінки, які отримують дані з бекенду та реагують на дії користувача. Thymeleaf обраний як шаблонізатор, оскільки він добре інтегрується зі Spring Boot і дозволяє створювати шаблони, які виглядають як звичайні HTML-документи, що спрощує процес розробки.

Структура інтерфейсу складається з кількох ключових компонентів, які повторюються на всіх сторінках для забезпечення послідовного досвіду користувача. Хедер містить логотип сайту, навігаційне меню та елементи авторизації (кнопки "Увійти" та "Зареєструватися" для неавторизованих користувачів або ім'я користувача та кнопка "Вийти" для авторизованих). Навігаційне меню адаптується залежно від ролі користувача: клієнти бачать опції для створення нових замовлень та перегляду своїх замовлень, майстри — для пошуку релевантних замовлень та управління своїм профілем, а адміністратори мають доступ до панелі адміністрування [24].

Головна сторінка системи представляє основні функції платформи та містить секції для реєстрації нових користувачів (як клієнтів, так і майстрів), перегляду популярних категорій послуг та активних замовлень. Для неавторизованих користувачів доступний пошук замовлень та майстрів, але для повноцінного використання платформи потрібна реєстрація. Головна сторінка також містить блок з відгуками задоволених клієнтів та статистику платформи (кількість зареєстрованих майстрів, виконаних замовлень тощо), що підвищує довіру нових користувачів.

Сторінка реєстрації користувачів розділена на два варіанти: для клієнтів та для майстрів. Форма реєстрації клієнта включає базові поля: ім'я, електронну пошту, пароль та підтвердження пароля. Форма реєстрації майстра, крім базових полів, містить додаткові поля для вибору категорій послуг, регіону роботи, опису досвіду та можливості завантаження фото. Обидві форми мають валідацію на стороні клієнта за допомогою JavaScript та на стороні сервера з використанням Bean Validation в Spring Boot.

Сторінка входу в систему містить просту форму з полями для електронної пошти та пароля, а також опцію "Запам'ятати мене" та посилання для відновлення забутого пароля. При невдалій спробі входу користувач отримує відповідне повідомлення про помилку. Після успішного входу користувач перенаправляється на відповідну сторінку залежно від його ролі: клієнти — на сторінку зі списком своїх замовлень, майстри — на сторінку з релевантними для них замовленнями, адміністратори — на панель адміністрування (рис.3.1).

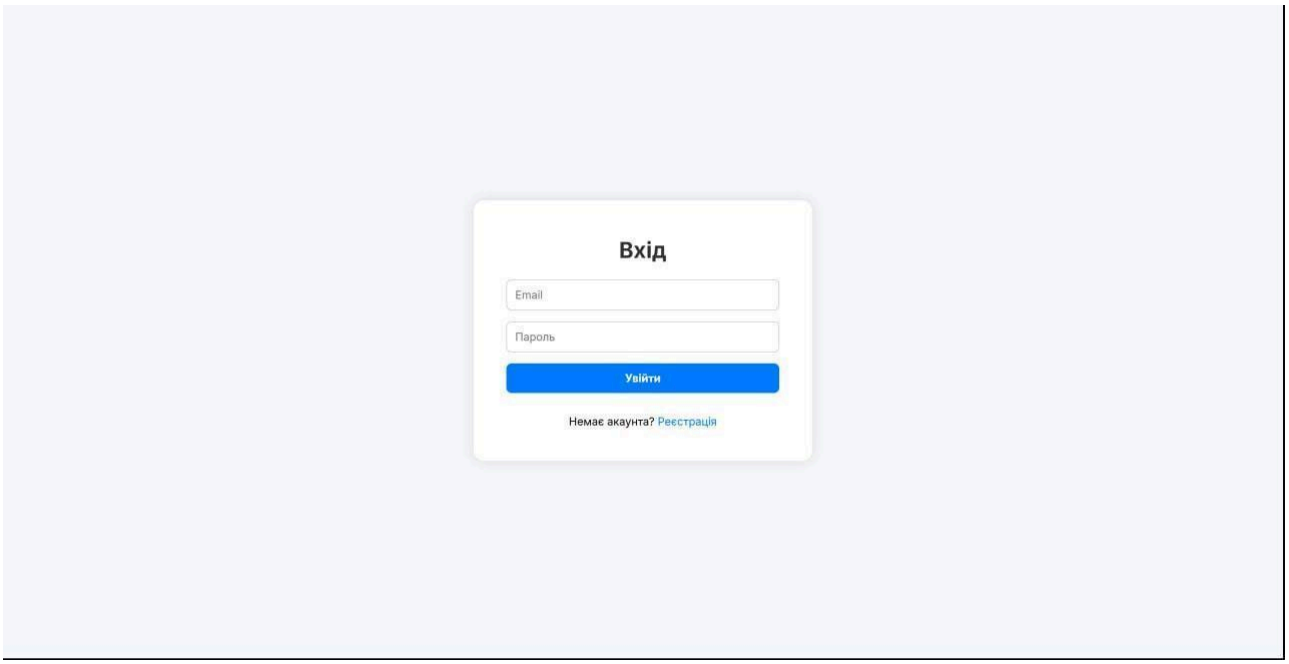

Рисунок 3.1 — Сторінка входу до системи MasterPoint з формою автентифікації для введення облікових даних користувачів та можливістю переходу до реєстрації нових облікових записів

Сторінка створення нового замовлення доступна клієнтам і містить форму з полями для вибору категорії послуг, регіону, бюджету, бажаної дати виконання та детального опису завдання. Поля форми мають підказки та обмеження для спрямування користувача до заповнення коректної інформації. Після успішного створення замовлення клієнт перенаправляється на сторінку деталей цього замовлення, де він може відстежувати відгуки майстрів (рис 3.2-3.3).

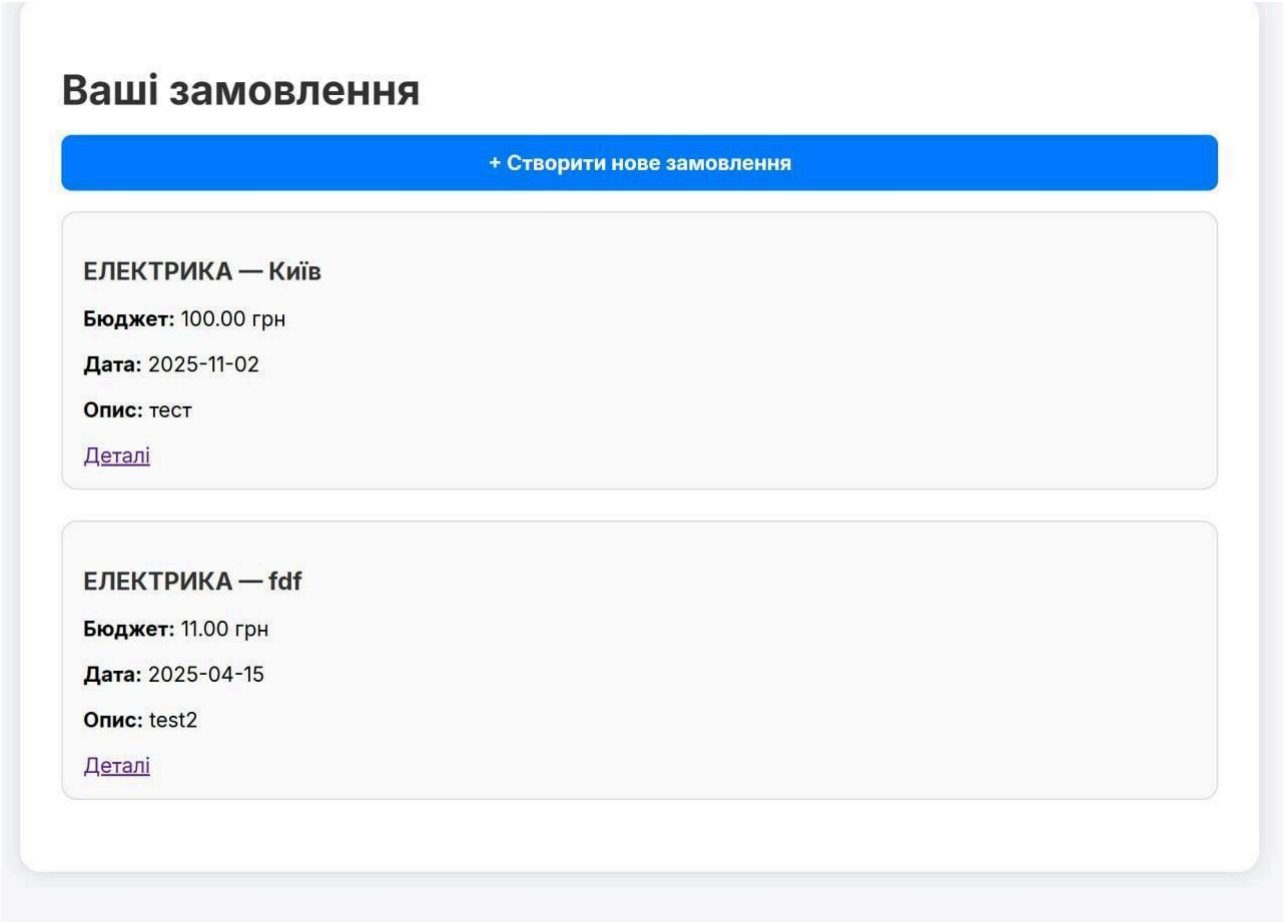

Рисунок 3.2 — Особистий кабінет клієнта системи MasterPoint зі списком створених замовлень у категорії "ЕЛЕКТРИКА" та кнопкою для створення нового замовлення

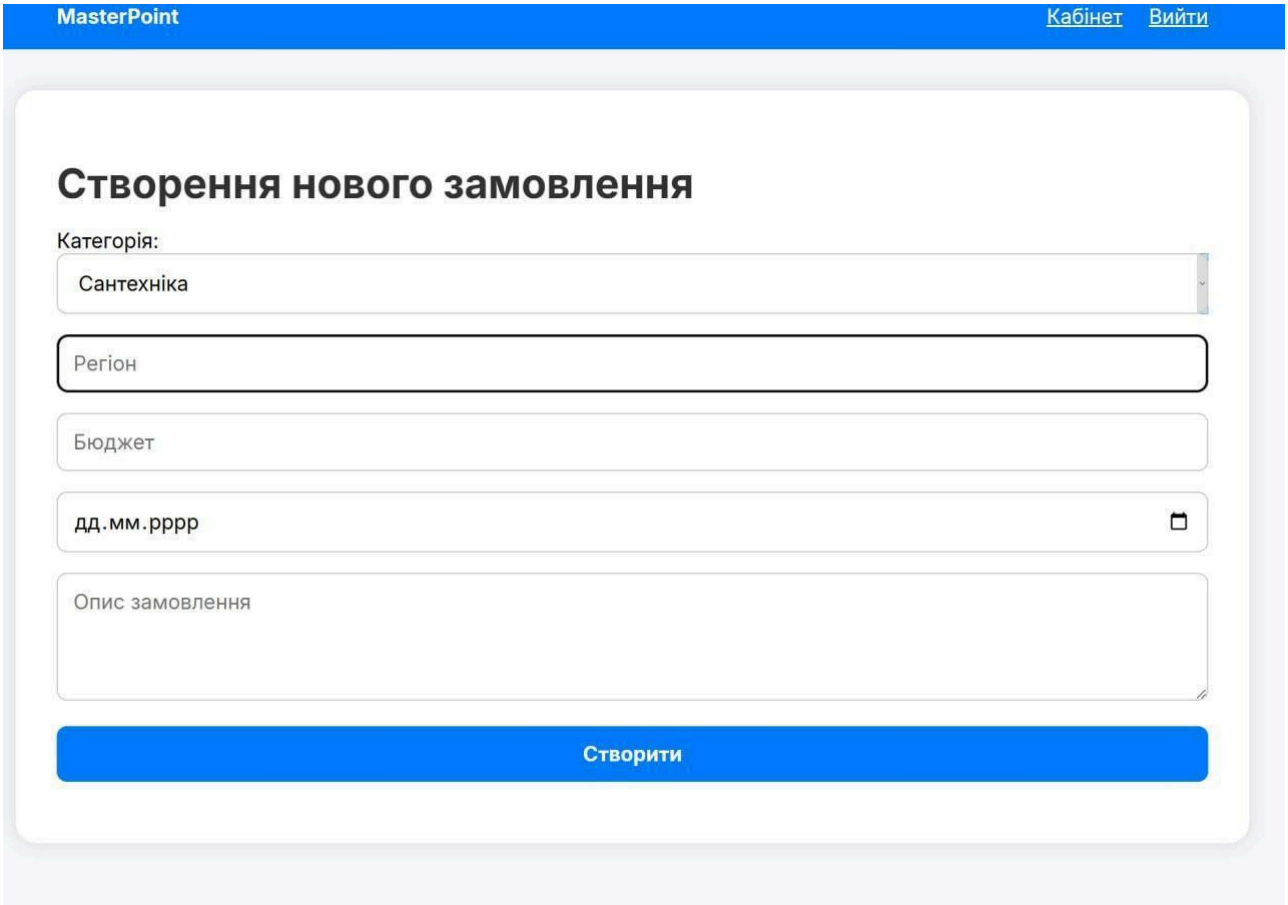

Рисунок 3.3 — Форма створення нового замовлення в системі MasterPoint з полями для вибору категорії, регіону, бюджету, дати виконання та опису завдання

Сторінка деталей замовлення відображає всю інформацію про замовлення: категорію, регіон, бюджет, дату, опис, а також список відгуків від майстрів. Для клієнта, який створив замовлення, доступні кнопки для вибору майстра та відхилення відгуків. Для майстрів доступна форма для залишення відгуку, якщо вони ще не відгукувалися на це замовлення. Сторінка також відображає статус замовлення (активне чи закрите) та, якщо майстра вже обрано, інформацію про нього (рис 3.4).

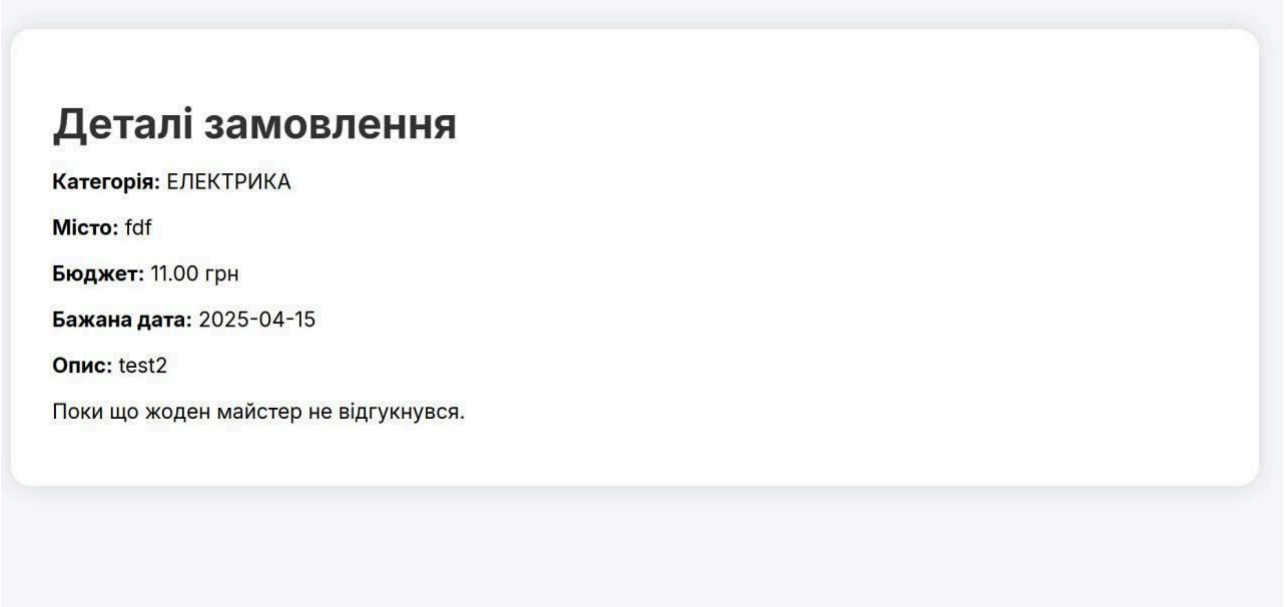

Рисунок 3.4 — Сторінка детального перегляду замовлення в системі MasterPoint з інформацією про категорію, місце, бюджет, дату виконання, опис та статус відгуків майстрів

Сторінка профілю майстра містить детальну інформацію про майстра: його ім'я, категорії послуг, регіон роботи, опис, фото, рейтинг та відгуки від клієнтів. Майстер може редагувати свій профіль, а клієнти можуть переглядати профілі майстрів для оцінки їхньої кваліфікації перед вибором для виконання замовлення. Профіль містить також статистику виконаних замовлень та середній рейтинг майстра.

Взаємодія інтерфейсу з бекендом відбувається за допомогою HTTP-запитів, які обробляються контролерами на стороні сервера. Для отримання даних використовуються GET-запити, а для відправки даних на сервер — POST-запити. Наприклад, коли клієнт створює нове замовлення, форма відправляє POST-запит на URL "/orders/create" з даними замовлення, контролер OrderController обробляє цей запит, викликає метод createOrder сервісу OrderService для збереження замовлення в базі даних, і повертає відповідну сторінку або перенаправлення [25].

Для підвищення інтерактивності інтерфейсу та покращення користувацького досвіду система використовує AJAX-запити для оновлення даних без перезавантаження сторінки. Наприклад, на сторінці деталей замовлення майстер

може залишити відгук через AJAX-запит, і новий відгук з'явиться в списку без перезавантаження сторінки. Це забезпечується за допомогою JavaScript та jQuery на стороні клієнта та відповідних ендпоінтів на стороні сервера, які повертають дані у форматі JSON.

3.2 Механізми пошуку та фільтрації майстрів

Ефективні механізми пошуку та фільтрації майстрів є ключовим компонентом системи MasterPoint, оскільки вони дозволяють клієнтам знаходити найбільш підходящих спеціалістів для вирішення їхніх завдань. Реалізація цих механізмів базується на комбінації серверної логіки з використанням SQL-запитів та клієнтської частини з інтерактивними елементами інтерфейсу. Завдяки цьому користувачі можуть легко знаходити потрібних майстрів за різними критеріями та порівнювати їх для прийняття обґрунтованого рішення.

Основним механізмом пошуку майстрів є фільтрація за категоріями послуг. У системі MasterPoint категорії організовані в ієрархічну структуру з кількома рівнями: головні категорії (наприклад, "Ремонт", "Електрика", "Сантехніка") та підкатегорії (наприклад, "Встановлення кондиціонерів", "Заміна розеток", "Ремонт змішувачів"). Користувачі можуть вибирати категорії з випадаючих списків або через інтерактивну візуальну навігацію, де категорії представлені іконками. Пошук за категоріями реалізовано через SQL-запит, який знаходить майстрів, пов'язаних з обраною категорією через проміжну таблицю master_categories. Географічний фільтр дозволяє знаходити майстрів, які працюють у конкретному регіоні. Користувачі можуть вибрати регіон зі списку або використати своє поточне місцезнаходження для пошуку майстрів поблизу. Система підтримує різні рівні географічної деталізації: країна, область, місто, район. Для визначення відстані між клієнтом і майстром використовується формула гаверсинусів на основі географічних координат. Цей підхід дозволяє клієнтам знаходити майстрів, які фізично ближче розташовані, що особливо важливо для термінових замовлень або послуг, які вимагають особистої присутності майстра.

Фільтрація за рейтингом є головним механізмом для оцінки якості послуг майстрів. Користувачі можуть фільтрувати майстрів за мінімальним рейтингом (наприклад, показати тільки майстрів з рейтингом не менше 4 зірок). Рейтинг майстра розраховується як середнє арифметичне всіх оцінок, отриманих від клієнтів після виконання замовлень. Система також дозволяє сортувати майстрів за рейтингом у порядку спадання, щоб найвище оцінені спеціалісти з'являлися першими у результатах пошуку.

Додатковим критерієм фільтрації є досвід роботи майстра, виражений у кількості виконаних замовлень або роках роботи. Клієнти можуть фільтрувати майстрів за мінімальною кількістю виконаних замовлень або за мінімальним досвідом роботи в роках. Ця інформація зберігається в профілі майстра та оновлюється автоматично при успішному виконанні нових замовлень. Такий фільтр допомагає клієнтам знаходити досвідчених спеціалістів для складних завдань. Система також підтримує фільтрацію за доступністю майстрів. Користувачі можуть вказати бажану дату виконання замовлення, і система покаже тільки тих майстрів, які доступні в цей період. Ця функціональність реалізована через порівняння бажаної дати з календарем зайнятості майстра, який містить інформацію про поточні замовлення та періоди недоступності, вказані самим майстром. Фільтр за доступністю особливо корисний для замовлень, які мають бути виконані в конкретний термін.

Для більш точного пошуку система надає можливість текстового пошуку за ключовими словами. Цей пошук охоплює імена майстрів, описи їхніх послуг, навички та інші текстові поля. Для реалізації ефективного текстового пошуку використовується повнотекстовий пошук (Full-Text Search) в базі даних або зовнішня пошукова система, така як Elasticsearch. Повнотекстовий пошук дозволяє знаходити релевантні результати навіть при частковому збігу або помилках у написанні.

Система пропонує гнучкі можливості сортування результатів пошуку. Користувачі можуть сортувати знайдених майстрів за різними критеріями: рейтингом, кількістю відгуків, досвідом роботи, ціновою категорією або датою

реєстрації на платформі. За замовчуванням результати сортуються за рейтингом у порядку спадання, але користувачі можуть змінити критерій сортування відповідно до своїх пріоритетів. Сортування реалізовано як на стороні сервера (через SQL-запити з ORDER BY), так і на стороні клієнта (через JavaScript) для динамічної зміни порядку без перезавантаження сторінки.

Для покращення користувацького досвіду система надає інтерактивний інтерфейс фільтрації з миттєвим оновленням результатів. Коли користувач змінює параметри фільтрації, система автоматично оновлює список майстрів без перезавантаження сторінки, використовуючи AJAX-запити до серверного API (рис 3.5).

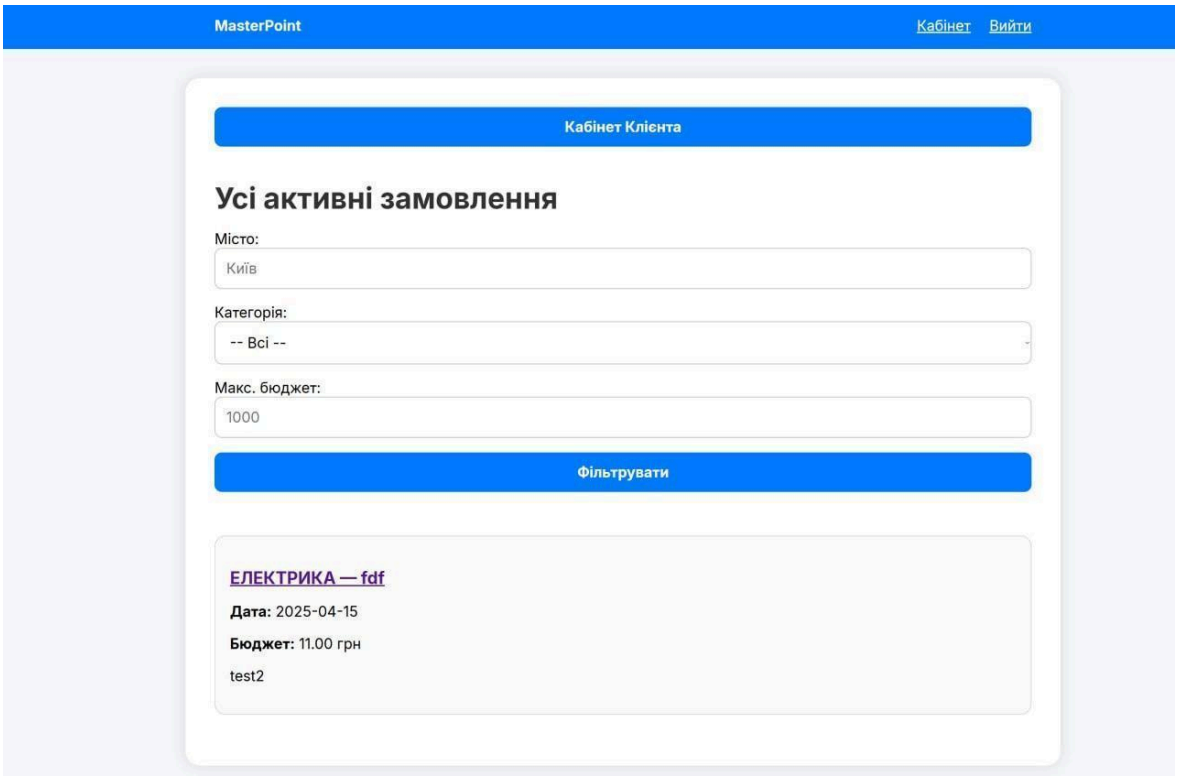

Рисунок 3.5 — Інтерфейс пошуку замовлень у системі MasterPoint з кнопкою фільтрування та картка замовлення в категорії "ЕЛЕКТРИКА" із зазначенням дати, бюджету та коротким описом

Це забезпечує швидкий та плавний процес пошуку, де користувачі можуть легко експериментувати з різними комбінаціями фільтрів для знаходження найкращого майстра.

Система також пропонує функцію "Рекомендовані майстри", яка автоматично показує майстрів, які найкраще відповідають потребам клієнта на основі його попередніх замовлень, уподобань та поведінки на платформі. Рекомендаційна система використовує алгоритми машинного навчання для аналізу взаємодій користувачів з платформою та виявлення патернів, які можуть вказувати на уподобання щодо певних типів майстрів. Це забезпечує персоналізований досвід пошуку, де кожен клієнт отримує рекомендації, адаптовані до його конкретних потреб.

Для мобільних пристроїв система адаптує інтерфейс пошуку та фільтрації, щоб забезпечити зручне використання на маленьких екранах. Фільтри організовані у компактне меню, яке можна розгорнути за потреби, а результати пошуку представлені у вигляді карток, оптимізованих для перегляду на мобільних пристроях. Також мобільна версія використовує геолокацію пристрою для автоматичного визначення місцезнаходження користувача та пошуку найближчих майстрів.

3.3 Система відгуків та рейтингу майстрів

Система відгуків та рейтингу майстрів є критично необхідним компонентом платформи MasterPoint, оскільки вона забезпечує механізм формування репутації спеціалістів та допомагає клієнтам приймати обґрунтовані рішення при виборі виконавця. Ця система розроблена з урахуванням психології користувачів та принципів соціального доказу, коли люди більше довіряють тим, хто має позитивні оцінки від інших. Технічна реалізація системи відгуків та рейтингу базується на використанні реляційної бази даних для зберігання оцінок та відгуків, із відповідною бізнес-логікою на рівні сервісів та інтерактивним інтерфейсом на клієнтській стороні.

Архітектурно система відгуків базується на моделі даних, що включає таблицю Review, яка зберігає всі відгуки клієнтів про майстрів. Кожен запис у цій таблиці містить унікальний ідентифікатор відгуку, ідентифікатори клієнта та майстра, ідентифікатор замовлення, за яким залишено відгук, числову оцінку (зазвичай за шкалою від 1 до 5), текстовий коментар, дату створення відгуку та, опціонально, фото результатів роботи. Ця структура даних дозволяє ефективно зберігати всю необхідну інформацію про відгуки та швидко отримувати статистику по конкретному майстру.

Бізнес-правила системи відгуків встановлюють, що клієнт може залишити відгук про майстра тільки після завершення замовлення, на яке було призначено цього майстра. Це обмеження забезпечується на рівні сервісу ReviewService, який перевіряє статус замовлення та зв'язок між замовленням, клієнтом та майстром перед створенням нового відгуку. Такий підхід запобігає зловживанням та гарантує, що відгуки залишають тільки реальні клієнти, які фактично взаємодіяли з майстром. Для забезпечення об'єктивності та якості відгуків система використовує механізм валідації контенту. Цей механізм включає автоматичну перевірку на наявність нецензурної лексики, спаму та іншого недопустимого контенту, а також можливість ручної модерації відгуків адміністраторами платформи. Система також дозволяє користувачам повідомляти про неприйнятні відгуки, що допомагає швидко виявляти та видаляти контент, який порушує правила платформи.

Інтерфейс створення відгуку розроблений таким чином, щоб заохочувати користувачів залишати детальні та інформативні відгуки. Форма містить поля для загальної оцінки за шкалою від 1 до 5 зірок, а також додаткові поля для оцінки окремих аспектів роботи майстра, таких як якість виконання, дотримання термінів, професіоналізм, комунікабельність та співвідношення ціна/якість. Крім того, форма містить текстове поле для детального опису досвіду співпраці з майстром та можливість завантаження фотографій результатів роботи.

Рейтинг майстра розраховується як середньозважене значення всіх оцінок, отриманих від клієнтів, з урахуванням їхньої актуальності та повноти. Для цього

використовується алгоритм, який надає більшої ваги нещодавнім відгукам та відгукам з детальним текстовим описом і фотографіями. Це забезпечує більш точне відображення поточного рівня послуг майстра та мотивує його підтримувати високу якість роботи. Розрахунок рейтингу відбувається автоматично при додаванні нового відгуку або зміні існуючого.

Відображення відгуків на сторінці профілю майстра реалізовано таким чином, щоб надати клієнтам максимум корисної інформації при прийнятті рішення про вибір виконавця. Відгуки відображаються у хронологічному порядку (за замовчуванням – від найновіших до найстаріших), з можливістю фільтрації за оцінкою та сортування за різними критеріями. Кожен відгук містить ім'я клієнта, дату, оцінку, текстовий коментар та, якщо є, фотографії. Також відображається інформація про категорію послуг, за якою було виконано замовлення, що дозволяє клієнтам оцінити досвід майстра в конкретній сфері. Система також підтримує можливість відповіді майстра на отримані відгуки. Це дозволяє майстрам пояснити ситуацію у випадку негативного відгуку або подякувати клієнту за позитивний відгук. Відповіді майстра відображаються безпосередньо під відповідним відгуком, що забезпечує контекст та дозволяє іншим клієнтам ознайомитися з обома сторонами ситуації. Такий діалог сприяє прозорості та підвищенню довіри до платформи.

Для запобігання маніпуляціям з рейтингом система MasterPoint впроваджує ряд захисних механізмів. Наприклад, система виявляє та фільтрує підозрілі патерни активності, такі як велика кількість позитивних відгуків за короткий період або відгуки з однієї IP-адреси. Крім того, система може тимчасово приховувати відгуки, які значно відхиляються від середньої оцінки майстра, до їх перевірки модераторами. Ці заходи допомагають підтримувати чесність та об'єктивність рейтингової системи.

Для заохочення клієнтів залишати відгуки система використовує різні мотиваційні механізми. Після завершення замовлення клієнт отримує електронного листа з нагадуванням залишити відгук про майстра. Також можуть використовуватися елементи гейміфікації, такі як бейджі за певну кількість

залишених відгуків або невеликі бонуси на наступні замовлення. Ці стимули підвищують активність користувачів та забезпечують більшу кількість відгуків, що робить рейтингову систему більш репрезентативною.

Окрім індивідуальних оцінок майстрів, система також генерує агреговану статистику, яка допомагає клієнтам прийняти рішення та надає майстрам інформацію для покращення своїх послуг. Ця статистика включає середній рейтинг за різними категоріями послуг, динаміку рейтингу з часом, порівняння з середніми показниками по платформі, а також аналіз сильних та слабких сторін на основі текстових відгуків. Для аналізу текстових відгуків можуть використовуватися алгоритми обробки природної мови для виділення ключових тем та сентименту.

Система відгуків та рейтингу також інтегрована з алгоритмами пошуку та рекомендацій. Майстри з вищим рейтингом отримують перевагу в результатах пошуку, а система рекомендацій враховує рейтинг при підборі найбільш підходящих виконавців для конкретного клієнта. Це створює позитивний зворотний зв'язок, де якісне виконання замовлень призводить до вищого рейтингу, що, в свою чергу, збільшує видимість майстра на платформі та кількість нових замовлень. Для нових майстрів, які ще не мають достатньої кількості відгуків, система використовує спеціальні алгоритми для справедливого відображення їх у результатах пошуку. Наприклад, може використовуватися байєсівська оцінка, яка враховує невелику кількість оцінок, або нові майстри можуть отримувати тимчасове підвищення видимості, щоб набрати перші відгуки. Ці заходи допомагають новим майстрам інтегруватися в платформу і конкурувати з більш досвідченими виконавцями [26].

Система також надає майстрам аналітичні інструменти для відстеження та покращення свого рейтингу. В особистому кабінеті майстра доступна детальна статистика по отриманих відгуках, включаючи середні оцінки за різними критеріями, динаміку рейтингу, порівняння з конкурентами у тій самій категорії послуг тощо. Ці дані допомагають майстрам виявляти слабкі місця у своїй роботі та цілеспрямовано покращувати якість послуг.

Конфіденційність є вагомим аспектом системи відгуків. Клієнти можуть вибирати, чи відображати своє реальне ім'я при залишенні відгуку, або використовувати псевдонім. Також система дозволяє клієнтам редагувати або видаляти свої відгуки протягом певного періоду після їх створення, що забезпечує гнучкість та контроль над своїми даними. При цьому система зберігає історію змін відгуків для запобігання зловживанням цією функціональністю.

3.4 Тестування та оптимізація продуктивності системи

Процес тестування системи MasterPoint побудований за принципом багаторівневої перевірки, що охоплює всі аспекти функціонування платформи – від окремих компонентів до комплексного взаємодії всіх модулів. Основою підходу є концепція піраміди тестування, де більшість тестів зосереджено на нижніх рівнях (юніт-тести), меншу кількість складають інтеграційні тести, і найменша частка припадає на складні end-to-end тести. Така стратегія забезпечує оптимальний баланс між швидкістю виконання тестів та їхньою здатністю виявляти дефекти.

Модульне (юніт) тестування застосовується для перевірки коректності роботи окремих компонентів системи, таких як сервіси, контролери та утилітні класи. Для цього використовується фреймворк JUnit у поєднанні з Mockito для створення мок-об'єктів, що імітують залежності тестованого компонента. Наприклад, при тестуванні сервісу OrderService створюються моки для його залежностей, таких як OrderRepository та UserRepository, що дозволяє ізолювати тестований компонент та перевірити його логіку незалежно від інших частин системи. Загальне покриття коду модульними тестами становить не менше 80%, що є хорошим показником для проекту такої складності [27].

Інтеграційне тестування фокусується на перевірці взаємодії між різними компонентами системи. Для цього використовується Spring Boot Test, який дозволяє запускати тести в контексті Spring, але з можливістю конфігурації тестового середовища. Типовим прикладом є тестування взаємодії контролерів з сервісами та репозиторіями, де перевіряється коректність обробки HTTP-запитів,

перетворення даних та взаємодії з базою даних. Для роботи з базою даних у тестах використовується вбудована база даних H2, яка дозволяє швидко створювати та заповнювати тестові таблиці без необхідності налаштування зовнішньої бази даних.

End-to-end тестування перевіряє роботу системи в цілому, включаючи інтерфейс користувача та взаємодію з зовнішніми сервісами. Для автоматизації таких тестів використовується Selenium WebDriver, який дозволяє емулювати дії користувача в браузері та перевіряти відображення елементів інтерфейсу. Сценарії тестування охоплюють основні користувацькі сценарії, такі як реєстрація нового користувача, створення замовлення, пошук майстрів та залишення відгуків. Хоча end-to-end тести є найбільш наближеними до реального використання системи, вони також є найбільш ресурсомісткими та вразливими до змін в інтерфейсі, тому їх кількість обмежена найбільш критичними сценаріями.

Для перевірки безпеки платформи MasterPoint проводяться спеціалізовані тести безпеки, спрямовані на виявлення таких вразливостей, як SQL-ін'єкції, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) та інші з переліку OWASP Top 10. Для цього використовуються як автоматизовані інструменти (OWASP ZAP, SonarQube Security), так і ручне тестування, де досвідчені тестувальники намагаються знайти та експлуатувати потенційні вразливості. Результати тестування безпеки регулярно аналізуються та впроваджуються відповідні заходи захисту для виявлених вразливостей.

Навантажувальне тестування є критично необхідним для забезпечення стабільної роботи платформи при високому рівні активності користувачів. Для цього використовується інструмент Apache JMeter, який дозволяє симулювати різні сценарії навантаження: поступове збільшення кількості користувачів, пікове навантаження, тривале навантаження тощо. Особлива увага приділяється найбільш ресурсомістким операціям, таким як пошук майстрів за складними критеріями, обробка одночасних запитів на створення замовлень, розрахунок рейтингів майстрів. Результати навантажувального тестування допомагають

визначити максимальну кількість користувачів, яку система може обслуговувати з прийнятним часом відгуку, та виявити вузькі місця, які потребують оптимізації.

Вагомим аспектом тестування є використання методології TDD (Test-Driven Development), де тести пишуться перед розробкою функціональності. Це забезпечує високу якість коду та мінімізує кількість дефектів на ранніх стадіях розробки. Крім того, всі тести інтегровані в систему безперервної інтеграції (CI/CD), що дозволяє автоматично запускати їх при кожному внесенні змін до репозиторію коду та запобігати появі регресій.

Після завершення етапу активного тестування та виявлення проблемних місць у системі розпочинається процес оптимізації продуктивності. Першим кроком є профілювання додатку для визначення найбільш ресурсомістких операцій. Для цього використовуються інструменти, такі як VisualVM, Java Flight Recorder та YourKit Java Profiler, які дозволяють аналізувати використання CPU, пам'яті, часу виконання методів та інші показники продуктивності. На основі результатів профілювання визначаються пріоритетні напрямки оптимізації.

Оптимізація запитів до бази даних є одним з ключових аспектів підвищення продуктивності системи. Аналіз виконання SQL-запитів за допомогою інструментів, таких як Hibernate Statistics, SQL Explain Plans та моніторинг логів СУБД, дозволяє виявити повільні запити та оптимізувати їх. Основні заходи оптимізації включають створення відповідних індексів, перегляд структури запитів, використання кешування результатів часто виконуваних запитів, оптимізацію стратегії завантаження пов'язаних сутностей (eagerly vs. lazily) та, в деяких випадках, використання нативних SQL-запитів замість ORM для особливо складних операцій.

Кешування є ефективним способом зменшення навантаження на базу даних та прискорення відгуку системи. У платформі MasterPoint використовується багаторівнева система кешування: на рівні бази даних (кешування запитів СУБД), на рівні ORM (кешування результатів запитів Hibernate), на рівні сервісів (кешування результатів бізнес-операцій з використанням Spring Cache) та на рівні HTTP (кешування відповідей на повторювані запити). Для реалізації кешування на

рівні сервісів та HTTP використовується Redis як розподілений кеш, що дозволяє зберігати узгодженість даних при горизонтальному масштабуванні системи.

Асинхронна обробка операцій застосовується для покращення користувацького досвіду та зменшення навантаження на систему. Довготривалі операції, такі як генерація складних звітів, обробка завантажених файлів, розсилка сповіщень великій кількості користувачів, реалізуються з використанням асинхронних механізмів. Spring Framework надає зручні інструменти для асинхронної обробки, такі як @Async анотації для методів, CompletableFuture для компонування асинхронних операцій та системи черг повідомлень (RabbitMQ, Apache Kafka) для реалізації складних сценаріїв обробки даних.

Оптимізація фронтенду також відіграє значну роль у забезпеченні швидкої та плавної роботи інтерфейсу користувача. Основні заходи включають мінімізацію та об'єднання JavaScript та CSS файлів, оптимізацію завантаження зображень (зменшення розміру, використання форматів WebP та AVIF, ледаче завантаження зображень, які не видно на екрані), використання CDN для статичних ресурсів, впровадження кешування на стороні клієнта та оптимізацію рендерингу сторінок. Для оцінки ефективності цих заходів використовуються інструменти, такі як Google PageSpeed Insights, Lighthouse та WebPageTest.

Горизонтальне масштабування системи є ключовим фактором забезпечення продуктивності при зростанні кількості користувачів. Архітектура MasterPoint побудована з урахуванням можливості горизонтального масштабування: всі компоненти системи є безстановими (stateless), що дозволяє запускати кілька екземплярів додатку за балансувальником навантаження. Для розподілу навантаження між екземплярами використовується Nginx як балансувальник, а для забезпечення узгодженості даних між екземплярами використовуються розподілені системи кешування та сесій на основі Redis. Крім того, використання Docker та Kubernetes дозволяє автоматично масштабувати кількість екземплярів додатку залежно від поточного навантаження.

Оптимізація JVM та налаштування контейнерів є вагомими аспектами тонкого налаштування продуктивності. Для Java-додатку це включає оптимізацію

параметрів Garbage Collector (вибір відповідного GC алгоритму, налаштування розміру пам'яті, частоти збору сміття тощо), налаштування пулу потоків, розміру кешів та інших JVM-параметрів. Для Docker-контейнерів оптимізуються налаштування виділених ресурсів (CPU, пам'ять, диск, мережа), використовуються оптимізовані базові образи (наприклад, Alpine-based для зменшення розміру), налаштовуються параметри логування та моніторингу.

Моніторинг продуктивності в режимі реального часу є невід'ємною частиною стратегії оптимізації. Для цього використовуються інструменти, такі як Prometheus для збору метрик, Grafana для візуалізації даних та Alert Manager для сповіщення про аномалії. Відстежуються ключові показники продуктивності: час відгуку API, використання CPU та пам'яті, кількість активних сесій, частота помилок, час виконання запитів до бази даних, стан Garbage Collector, активність мережі тощо. Аналіз цих даних дозволяє оперативно виявляти проблеми продуктивності та приймати рішення щодо подальшої оптимізації [28].

Автоматизація тестування та оптимізації є необхідним аспектом підтримки стабільної та ефективної роботи системи з часом. Регресійне тестування, навантажувальне тестування та перевірка безпеки інтегровані в CI/CD pipeline, що дозволяє автоматично виявляти проблеми при внесенні змін до коду. Крім того, встановлені граничні значення для ключових показників продуктивності, і якщо ці значення перевищуються, відповідні тести не проходять, запобігаючи випуску неоптимізованого коду в виробниче середовище.

3.5 Розгортання та впровадження системи

Розгортання системи MasterPoint реалізовано з використанням сучасних принципів DevOps та CI/CD (Continuous Integration / Continuous Deployment), що забезпечує автоматизацію процесів розробки, тестування та розгортання. Такий підхід дозволяє швидко та надійно впроваджувати нові функції, виправлення помилок та оптимізації, мінімізуючи людський фактор та ризик виникнення проблем при переході від розробки до експлуатації. Основою реалізації цього

підходу є набір інструментів та практик, які забезпечують повний цикл розгортання, від комміту коду до запуску в продакшн-середовищі.

Для управління версіями коду використовується система контролю версій Git, а саме GitHub, який забезпечує зручний інтерфейс для колаборації розробників, перегляду змін, проведення код-рев'ю та автоматизації процесів. Розробка ведеться за принципом Git Flow, де основний код знаходиться в гілці main, а розробка нових функцій здійснюється в окремих гілках feature, які потім об'єднуються з основною гілкою через Pull Request. Такий підхід забезпечує контроль якості коду, оскільки кожен Pull Request проходить перевірку іншими розробниками та автоматизованими тестами перед об'єднанням з основною гілкою.

Для автоматизації процесів збірки, тестування та розгортання використовується GitHub Actions – інтегрований інструмент CI/CD платформи GitHub. Для кожного Pull Request автоматично запускаються юніт-тести, інтеграційні тести, статичний аналіз коду та перевірка на наявність вразливостей. Лише після успішного проходження всіх перевірок зміни можуть бути об'єднані з основною гілкою. Після об'єднання змін автоматично запускається процес збірки та розгортання додатку в тестове середовище, де проводяться додаткові тести, включаючи end-to-end тести та перевірку продуктивності.

Для пакування та розгортання додатку використовується технологія контейнеризації Docker, яка забезпечує консистентність середовища виконання на всіх етапах – від розробки до промислової експлуатації. Додаток пакується у Docker-контейнер, який містить всі необхідні залежності, конфігурації та сам додаток, що забезпечує його ізоляцію від операційної системи та інших додатків. Для збірки та публікації Docker-образів використовується Docker Registry, де зберігаються всі версії образів з відповідними тегами, що дозволяє легко повертатися до попередніх версій у разі необхідності [29].

Управління контейнерами у промисловому середовищі здійснюється за допомогою Kubernetes – платформи для оркестрації контейнерів, яка забезпечує автоматичне масштабування, самовідновлення при збоях, балансування

навантаження та інші функції, необхідні для надійної роботи системи. Кластер Kubernetes налаштований на хмарній платформі Amazon Web Services (AWS), що забезпечує високу доступність, захист від відмов та можливість глобального масштабування. Конфігурації Kubernetes (deployment, service, ingress тощо) зберігаються в Git-репозиторії та застосовуються автоматично при розгортанні нової версії додатку.

Управління конфігураціями системи реалізовано через механізм ConfigMaps та Secrets у Kubernetes, що дозволяє зберігати різні конфігурації для різних середовищ (розробка, тестування, продакшн) та безпечно керувати секретними даними, такими як паролі, ключі API тощо. Конфігурації для різних середовищ зберігаються в Git-репозиторії у зашифрованому вигляді (з використанням інструменту SOPS) та автоматично розшифровуються при розгортанні. Такий підхід забезпечує прозорість конфігурацій, можливість відстеження змін та легке переключення між різними середовищами.

Базою даних для системи MasterPoint обрано PostgreSQL – надійну та продуктивну реляційну СУБД з підтримкою складних запитів, транзакцій та інших функцій, необхідних для системи такого рівня. В продакшн-середовищі база даних розгорнута як керований сервіс Amazon RDS, який забезпечує автоматичне резервне копіювання, моніторинг, високу доступність та масштабованість. Для забезпечення надійності та доступності даних використовується реплікація бази даних, де основний екземпляр приймає запити на запис, а реліки обслуговують запити на читання, що дозволяє розподілити навантаження та забезпечити швидкий доступ до даних. Для кешування та зберігання сесій використовується Redis – високопродуктивне сховище даних в пам'яті, яке забезпечує швидкий доступ до часто використовуваних даних. Redis розгорнуто як керований сервіс Amazon ElastiCache, що забезпечує високу доступність та автоматичне масштабування. Використання Redis для кешування дозволяє значно зменшити навантаження на базу даних та прискорити відповіді на запити користувачів, особливо для операцій, які часто виконуються, таких як пошук майстрів за певними критеріями, отримання деталей замовлення тощо.

Для зберігання статичних файлів, таких як фотографії майстрів, результати виконаних робіт та інші медіа-файли, використовується Amazon S3 – масштабоване об'єктне сховище з високою доступністю та низькою вартістю зберігання. Для прискорення доставки статичного контенту користувачам налаштовано Amazon CloudFront – службу доставки контенту (CDN), яка кешує статичні файли на серверах по всьому світу, забезпечуючи швидкий доступ користувачам з різних географічних регіонів. Це особливо важливо для платформи MasterPoint, яка орієнтована на користувачів з різних міст і регіонів.

Моніторинг та логування системи реалізовано за допомогою комплексу інструментів, включаючи Prometheus для збору метрик, Grafana для візуалізації даних, ELK Stack (Elasticsearch, Logstash, Kibana) для збору та аналізу логів, та New Relic для моніторингу продуктивності додатку на рівні коду. Ці інструменти дозволяють оперативно виявляти проблеми, аналізувати причини їх виникнення та приймати рішення щодо оптимізації. Крім того, налаштовано сповіщення про критичні події (високе навантаження на систему, помилки, аномалії в метриках) через Slack та електронну пошту, що забезпечує швидку реакцію команди на проблеми.

Масштабування системи при зростанні кількості користувачів забезпечується на кількох рівнях. На рівні додатку використовується горизонтальне масштабування через Kubernetes, який автоматично збільшує кількість контейнерів (pods) при зростанні навантаження. На рівні бази даних використовується вертикальне масштабування (збільшення ресурсів сервера) та горизонтальне масштабування через реплікацію та шардинг, що дозволяє розподілити навантаження між кількома серверами. Також використовується кешування часто використовуваних даних в Redis та на рівні CDN, що зменшує навантаження на базу даних та прискорює відповіді на запити.

Безпека системи забезпечується на кількох рівнях. На рівні мережі використовується Amazon VPC (Virtual Private Cloud) для ізоляції ресурсів та AWS WAF (Web Application Firewall) для захисту від типових атак на веб-додатки. На рівні додатку використовується Spring Security для аутентифікації та авторизації

користувачів, захисту від CSRF, XSS та інших типових вразливостей. Для захисту даних використовується шифрування в стані спокою (encryption at rest) для бази даних та сховища S3, а також шифрування під час передачі (encryption in transit) через HTTPS. Регулярно проводяться аудити безпеки та тести на проникнення для виявлення та усунення вразливостей.

Впровадження системи в експлуатацію проводиться поетапно, починаючи з тестового запуску з обмеженою кількістю користувачів (бета-версія). Цей підхід дозволяє виявити та виправити проблеми в реальних умовах експлуатації без ризику для широкої аудиторії користувачів. Під час бета-тестування збирається зворотний зв'язок від користувачів, аналізуються патерни використання системи, виявляються вузькі місця та оптимізується продуктивність. Після успішного завершення бета-тестування система поступово відкривається для все більшої кількості користувачів, з постійним моніторингом продуктивності та стабільності.

Стратегія оновлення системи базується на принципі безперервної інтеграції та доставки (CI/CD), що дозволяє регулярно випускати нові версії з мінімальним ризиком. Для цього використовується підхід Blue-Green Deployment, де нова версія розгортається паралельно з поточною версією, і після успішного проходження всіх тестів трафік перенаправляється з поточної на нову версію. Такий підхід забезпечує нульовий час простою (zero downtime) при оновленні системи та можливість швидкого повернення до попередньої версії у разі виявлення проблем після розгортання.

Документація системи є невід'ємною частиною процесу розгортання та впровадження. Вона включає технічну документацію для розробників та адміністраторів (архітектура системи, опис API, інструкції з розгортання та налаштування), а також документацію для користувачів (керівництво користувача, FAQ, відеоінструкції). Вся документація зберігається в Git-репозиторії, що забезпечує контроль версій та можливість колаборації. Технічна документація генерується автоматично з коду за допомогою інструментів, таких як Swagger для API та JavaDoc для Java-коду, що забезпечує її актуальність при оновленні системи.

Підтримка системи після впровадження включає кілька рівнів: технічна підтримка для користувачів (вирішення проблем, допомога у використанні системи), моніторинг та оперативне реагування на проблеми, регулярні оновлення системи з новими функціями та виправленнями помилок. Для технічної підтримки користувачів створено службу підтримки з багатоканальним доступом (електронна пошта, чат на сайті, телефон), а також базу знань з відповідями на часті запитання. Для внутрішньої підтримки системи створено команду DevOps, яка забезпечує моніторинг, оптимізацію та вирішення технічних проблем [30].

Таким чином, розгортання та впровадження системи MasterPoint реалізовано з використанням сучасних практик DevOps та CI/CD, що забезпечує надійність, масштабованість, безпеку та можливість гнучкого розвитку системи відповідно до потреб користувачів. Використання хмарних технологій та контейнеризації дозволяє ефективно управляти ресурсами та забезпечувати високу доступність системи навіть при значному зростанні кількості користувачів. Поетапний підхід до впровадження, документація та служба підтримки забезпечують плавний перехід від розробки до експлуатації та позитивний досвід користувачів при взаємодії з системо

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено веб-платформу MasterPoint для пошуку майстрів з використанням технології Spring Boot. Система успішно реалізує функціональні можливості, необхідні для ефективної взаємодії клієнтів та майстрів, забезпечуючи зручний інтерфейс та надійну роботу на всіх етапах користування сервісом.

У ході дослідження було проаналізовано сучасні платформи для пошуку спеціалістів, визначено їх переваги та недоліки, що дозволило сформулювати вимоги до системи MasterPoint. Було обґрунтовано вибір технології Spring Boot як основи для розробки, враховуючи її надійність, масштабованість, широку екосистему та високу продуктивність.

Під час проєктування системи були розроблені моделі даних, які ефективно відображають предметну область – користувачів з різними ролями (клієнти, майстри, адміністратори), замовлення, категорії послуг, відгуки та рейтинги. Архітектура системи побудована за принципами багатошаровості та розділення відповідальності, що забезпечує гнучкість, тестованість та підтримуваність коду.

Реалізація бізнес-логіки та контролерів дозволила забезпечити всі необхідні функціональні можливості: реєстрація та аутентифікація користувачів, створення та управління замовленнями, пошук та фільтрація майстрів, комунікація між користувачами, система відгуків та рейтингів. Особливу увагу було приділено безпеці системи, включаючи захист від атак, контроль доступу та захист даних користувачів.

Інтерфейс користувача розроблено з урахуванням принципів UX/UI дизайну, забезпечуючи інтуїтивно зрозумілу навігацію та зручну взаємодію для всіх типів користувачів. Реалізовано адаптивний дизайн, що дозволяє використовувати систему на різних пристроях – від десктопних комп'ютерів до мобільних телефонів.Розроблена система пошуку та фільтрації майстрів дозволяє користувачам ефективно знаходити виконавців за різними критеріями: категорією

послуг, регіоном, рейтингом, досвідом роботи тощо. Реалізовано різні способи сортування та фільтрації результатів, що забезпечує гнучкість та зручність пошуку.

Система відгуків та рейтингу майстрів надає можливість формування об'єктивної оцінки якості послуг, що сприяє підвищенню довіри до платформи та мотивує майстрів підтримувати високу якість роботи. Реалізовано механізми захисту від маніпуляцій з рейтингом та забезпечення об'єктивності оцінок.

Тестування та оптимізація продуктивності системи проведені на всіх рівнях – від юніт-тестів окремих компонентів до комплексного тестування всієї системи. Реалізовано ряд оптимізацій для підвищення швидкодії та масштабованості, включаючи оптимізацію запитів до бази даних, кешування, асинхронну обробку операцій та оптимізацію фронтен

Розгортання та впровадження системи реалізовано з використанням сучасних практик DevOps та CI/CD, що забезпечує надійність, масштабованість та безпеку в умовах реальної експлуатації. Використання контейнеризації та хмарних технологій дозволяє гнучко управляти ресурсами та забезпечувати високу доступність системи навіть при значному зростанні кількості користувачів.

Розроблена платформа MasterPoint успішно вирішує поставлені задачі та має потенціал для подальшого розвитку та масштабування. Система може бути розширена новими функціональними можливостями, такими як інтеграція з платіжними системами, розширена аналітика для майстрів та клієнтів, мобільні додатки тощо. Також можливе географічне розширення платформи на нові регіони та країни.

Таким чином, розроблена система MasterPoint є практичним та ефективним рішенням для з'єднання клієнтів та майстрів, яке відповідає сучасним вимогам до веб-додатків та має потенціал для успішного впровадження на ринку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Азьмук Н. А. Сучасні виклики ринку праці при переході до цифрової економіки. 2020.
2. Варіс І., Кравчук О., Спіріна К. Цифрове середовище розвитку бренду роботодавця // Економіка та суспільство. 2022. №36.
3. Кисилевич В., Вербівський Д. С. Огляд технологій для створення односторінкових вебдодатків // Актуальні питання сучасної інформатики: матеріали доп. VI Всеукр. наук.-практ. конф. з міжнар. участю "Сучасні інформаційні технології в освіті та науці" (18–19 листопада 2021 р.). 2021.
4. Астістова Т. І. Огляд популярних фреймворків Python для веб розробки // Київ. нац. ун-т технологій та дизайну. 2024.
5. Комендат О. Р. Інформаційні технології аналізу та візуалізації метеоданих на базі веб-додатків. 2018.
6. Гевод В. С. Методи та засоби back-end розробки веб-застосунку “Кабінет студенту УАН”. 2024.
7. Стешко В. Ю. Дослідження архітектурних рішень та методів оптимізації для підвищення продуктивності додатків на Node.js і Vue.js. 2023.
8. Мирець Р. В. Розроблення програмного забезпечення клієнт-серверного додатка з використанням патернів проектування. 2023.
9. Панченко Ю. С. Розробка функціональних та нефункціональних вимог до системи обміну інформацією з використанням квантового розподілу ключів. 2022.
10. Логінов Д. О. Розробка інформаційної системи для магазину побутової техніки. 2024.
11. Редько І. В., Лисенко О. М. Композиційні засади проектування баз даних. 2019.
12. Демиденко М. А. Введення в сучасні бази даних. 2020.
13. Лосєв М. Ю., Федько В. В., Лосев М. Ю. Бази даних. 2019.

14. Первушин А. Г. Дослідження моделей та компонентів бізнес логіки у міжплатформеній мобільній розробці // The 11th International scientific and practical conference “Actual problems of learning and teaching methods” (December 06-09, 2022), Vienna, Austria. International Science Group. 2022.

15. Матвійчук Є. Є. Реалізація веб-орієнтованого програмного рішення для навчання студентів програмуванню // Міжнар. наук. інтернет-конф. "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення". 2021.

16. Дідук В. А., Гриценко В. Г., Єрьоменко А. Д. Побудова моделі мережевої взаємодії складових мультиагентної системи мобільних роботів // Eastern-European Journal of Enterprise Technologies. 2020.

17. Олексійчук Ю. Ф. та ін. Проєктування, розробка та тестування web-сервісу для вибору тем дипломних робіт. 2024.

18. Лунякіна В. Розробка REST API для системи запису на вибіркові. 2021.

19. Єрмішин Д. Є. Розробка REST API фітнес-центрів з використанням Laravel. 2023.

20. Прокопчук Р. С. Дослідження та програмна реалізація системи автентифікації користувачів в інтернет-мережах на основі FreeRADIUS. 2024.

21. Фібрук Р. Програмний додаток аутентифікації та авторизації користувача. 2023.

22. Шпильовий Р. П. Сервіс аутентифікації та авторизації для систем з розподіленою архітектурою. 2022.

23. Сметанка Ю. А. Архітектура системи автентифікації користувачів інформаційної системи на основі віддаленого виділеного сервера. Магістер. дис. 2019.

24. Валсамакі В. Д. Інформаційна система «Адміністрування готельної мережі». 2024.

25. Шевчук П. Т. Розробка онлайн-платформи для освітнього блогу. Івано-Франківськ: Ун-т Короля Данила, 2024.

26. Киричок А. В. Інформаційна система організації замовлення косметичних послуг. Магістер. дис. Сумський держ. ун-т, 2022.

27. Ткачук В. А. Метод оптимізації продуктивності систем інтелектуальних мереж. 2025.

28. Косенко Б. А. Дослідження методів оптимізації та проектувальних рішень для створення складних ігрових програмних систем. 2024.

29. Бєлоха Г. С., Тараба М. О. Транзактивні локальні електроенергетичні системи: особливості функціонування та перспективи розвитку. 2023.

30. Шепета О. В. Система безперервного розгортання проектів для оптимізації роботи компанії зі створення комерційного програмного забезпечення. 2024.

ДОДАТКИ

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>MasterPoint</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>MasterPoint</name>
    <description>MasterPoint</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
```

```
    <url/>
</scm>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web-services</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity6</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
</dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

Концепція проекту

MasterPoint - веб-платформа для пошуку майстрів з використанням Spring Boot. Ця кваліфікаційна робота розроблена для створення ефективного онлайн-середовища взаємодії між клієнтами та спеціалістами, представлена у 2025 році.

Актуальність теми

У сучасному світі швидко зростає потреба в якісних онлайн-сервісах для пошуку спеціалістів. Існує проблема довіри та прозорості при виборі майстрів, а традиційні методи пошуку часто неефективні. MasterPoint вирішує ці питання, надаючи єдину платформу для взаємодії, з чіткою системою рейтингів, що сприяє розвитку ринку послуг та підтримці малого бізнесу.

Мета та завдання

Основна мета роботи – розробка веб-платформи для пошуку майстрів з використанням технології Spring Boot, яка забезпечить ефективну взаємодію між клієнтами та виконавцями послуг. Для досягнення мети розроблено архітектуру системи та базу даних, реалізовано моделі, бізнес-логіку та контролери, створено систему авторизації та інтерфейс користувача, впроваджено механізми пошуку та рейтингової системи.

Об'єкт та предмет дослідження

Об'єктом дослідження виступає процес пошуку та взаємодії між клієнтами та майстрами в онлайн-середовищі. Предметом дослідження є методи та технології розробки веб-платформи для пошуку майстрів з використанням Spring Boot.

Аналіз існуючих рішень

Дослідження ринку виявило три основні типи платформ: вертикальні маркетплейси (TaskRabbit, Thumbtack) з глибокою спеціалізацією, але обмеженим спектром послуг; горизонтальні платформи (Upwork, Freelancer) з широким охопленням, але меншою ефективністю у специфічних галузях; та локальні сервіси (Kabanchik), адаптовані до місцевих ринків, але з обмеженим географічним охопленням.

Обґрунтування вибору технологій

Для розробки MasterPoint обрано стек технологій, що забезпечує надійність, масштабованість та зручність розробки. Spring Boot надає автоконфігурацію, велику екосистему компонентів та готові рішення для безпеки. Java забезпечує надійність, крос-платформеність та потужну типізацію. Thymeleaf ідеально інтегрується зі Spring, а MySQL пропонує оптимальне поєднання продуктивності та надійності для зберігання даних.

Проектування бази даних

- Візуальна схема бази даних з основними таблицями:
•User (користувачі)
•ClientProfile (профілі клієнтів)
•MasterProfile (профілі майстрів)
•Order (замовлення)
•Category (категорії послуг)
•OrderResponse (відгуки на замовлення)
•Review (відгуки про майстрів)
- Показані поля та зв'язки між таблицями

Архітектура системи

Система побудована за багатошаровою архітектурою, що відповідає принципам Spring MVC. Клієнтський рівень представлений інтерфейсом на HTML, CSS, JavaScript та Thymeleaf. Контролери обробляють HTTP-запити, сервіси містять бізнес-логіку, а репозиторії забезпечують доступ до даних. Такий підхід забезпечує чіткий розподіл відповідальності та масштабованість.

Реалізація системи авторизації

Безпека системи забезпечується Spring Security з використанням BCryptPasswordEncoder для шифрування паролів. Впроваджено систему ролей (CLIENT, MASTER, ADMIN) з контролем доступу до різних URL-шляхів. Процес аутентифікації включає перевірку облікових даних, створення сесії та керування правами доступу відповідно до ролі користувача.

Інтерфейс користувача

Інтерфейс MasterPoint розроблений з фокусом на зручність, адаптивність та інтуїтивність. Основні сторінки включають форму входу, кабінети клієнта і майстра, сторінку створення та перегляду замовлень. Особливістю є динамічне оновлення контенту через AJAX та чітка валідація форм, що забезпечує позитивний досвід користувача на будь-якому пристрої.

Механізми пошуку та фільтрації

Система пропонує гнучкі можливості пошуку майстрів за категоріями, регіоном, рейтингом, досвідом та доступністю. Технічна реалізація базується на оптимізованих SQL-запитах через JPA та використанні індексів для підвищення продуктивності. Інтерактивний інтерфейс дозволяє користувачам миттєво бачити результати фільтрації без перезавантаження сторінки.

Система відгуків та рейтингу

Репутаційна система є важливим елементом MasterPoint. Клієнти можуть оцінювати роботу майстрів за 5-бальною шкалою, залишати текстові відгуки та фотографії. Рейтинг розраховується як зважене середнє оцінок з урахуванням їх актуальності. Впроваджені механізми захисту від маніпуляцій, а система заохочення відгуків підвищує кількість оцінок та їх репрезентативність.

Тестування та оптимізація

Якість системи забезпечується комплексним підходом до тестування: модульні тести з JUnit та Mockito, інтеграційні тести через Spring Boot Test, та end-to-end тестування з Selenium. Навантажувальне тестування з JMeter показало стабільну роботу при 500 одночасних користувачах із середнім часом відповіді 1.2 секунди. Оптимізації включають кешування, асинхронну обробку та оптимізацію SQL-запитів.

Розгортання та впровадження

Для розгортання використано підхід DevOps з контейнеризацією через Docker та оркестрацією Kubernetes. CI/CD процес автоматизовано через GitHub Actions. Архітектура розгортання включає балансувальник навантаження, кластер серверів Spring Boot, реплікацію бази даних MySQL та Redis для кешування. Впровадження відбувається поетапно, починаючи з бета-тестування обмеженою групою користувачів.

Висновки

Розроблена платформа MasterPoint успішно вирішує поставлені завдання, забезпечуючи ефективну взаємодію між клієнтами та майстрами. Перспективи розвитку проекту включають інтеграцію з платіжними системами, розробку мобільних додатків, впровадження алгоритмів рекомендацій на основі машинного навчання та розширення на нові ринки. Платформа має потенціал для комерціалізації та може бути адаптована для різних галузей професійних послуг.