

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа мережі барбершопів»

Виконав студент групи ПЗ-22-1

СИНГАЇВСЬКИЙ Максим Олександрович

Керівник роботи:

ЧИЖМОТРА Олексій Володимирович

Рецензент

ВАКАЛЮК Тетяна Анатоліївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СИНГАЇВСЬКИЙ Максим Олександрович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Вебплатформа мережі барбершопів»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: Вебплатформа мережі барбершопів для спрощення ведення бізнес-процесів та їх аналітики із рольовим доступом, онлайн-записом, форумом та системою оцінок.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	02.03.2026	02.03.2026
2	Чижмотря О.В.	02.04.2026	02.04.2026
3	Чижмотря О.В.	02.05.2026	02.05.2026

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра, складається з вебдодатку «Вебплатформа мережі барбершопів», вступу, 3-х розділів, висновків, списку використаних джерел із 30 найменувань (2 стор.), 3 додатки (17 стор.), 27 таблиць та 34 ілюстрації по тексту. Повний обсяг роботи становить 87 сторінок, з них 70 основного тексту.

Мета роботи – розробка та моделювання вебсайту для вебплатформи мережі барбершопів, що забезпечує автоматизацію основних бізнес-процесів, таких як управління записами клієнтів, облік послуг, робота з майстрами та адміністрування.

Використано методи веброзробки, проєктування баз даних, алгоритмізації, UX/UI дизайну та розподілу доступу до функціоналу. Отримані результати включають розроблену архітектуру вебплатформи, спроектовану базу даних, реалізацію модулів управління записами, послугами, майстрами та системи онлайн-бронювання.

Рекомендовано впровадження розробленої вебплатформи у мережах барбершопів для покращення організації роботи, спрощення запису клієнтів та підвищення якості обслуговування.

КЛЮЧОВІ СЛОВА: ВЕБПЛАТФОРМА, БАРБЕРШОП, БРОНЮВАННЯ, МОДЕЛЮВАННЯ, ІНТЕРФЕЙС

					ІПЗ.КР.Б-121-26-ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Сингаївський М.О.			Вебплатформа мережі барбершопів			Літ.	Арк.	Аркушів	
Керівник		Чижмотря О.В.								3	87
Рецензент		Вакалюк Т.А.						Житомирська політехніка, група ІПЗ-22-1			
Зав.каф.		Вакалюк Т.А.									

ABSTRACT

The bachelor's final qualification work consists of the web application "Web Platform for a Barbershop Network", an introduction, 3 chapters, conclusions, a list of references with 30 titles (2 pages), 3 appendices (17 pages), 27 tables, and 34 illustrations throughout the text. The total volume of the work is 87 pages, of which 70 pages constitute the main body of text.

The purpose of the work is the development and modeling of a website for a barbershop network web platform that provides automation of core business processes, such as managing client appointments, accounting for services, working with barbers, and administration.

The work employs methods of web development, database design, algorithmization, UX/UI design, and role-based access control. The obtained results include the developed architecture of the web platform, the designed database, the implementation of modules for managing appointments, services, and barbers, and an online booking system.

The implementation of the developed web platform in barbershop networks is recommended to improve work organization, simplify client appointment booking, and enhance service quality.

KEYWORDS: WEB PLATFORM, BARBERSHOP, BOOKING, MODELING, INTERFACE

					ИПЗ.КР.Б-121-26-ПЗ	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБПЛАТФОРМИ МЕРЕЖІ БАРБЕРШОПІВ.....	9
1.1. Постановка задачі.....	9
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури вебплатформи	15
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	19
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБДОДАТКУ.....	25
2.1. Визачення варіантів використання та структури системи.....	25
2.2. Розробка структури бази даних	33
2.3. Проектування та реалізація алгоритмів роботи системи	38
2.4. Реалізація вебплатформи мережі барбершопів	41
Висновки до другого розділу.....	47
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ...	49
3.1. Порядок встановлення та налаштування параметрів системи	49
3.2. Структура інтерфейсу	52
3.3. Тестування роботи програмного комплексу	62
Висновки до третього розділу	66
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	69
ДОДАТКИ.....	71

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – прикладний програмний інтерфейс

БД – база даних

ПЗ – програмне забезпечення

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						6

ВСТУП

Актуальність теми полягає в тому, що цифровізація сфери послуг стрімко зростає, а клієнти очікують швидкого та зручного онлайн-доступу до сервісів. Особливо це актуально для мереж барбершопів, де важливо забезпечити ефективний запис клієнтів, управління розкладом майстрів, контроль завантаженості філій та зручну комунікацію з відвідувачами.

Вебплатформа мережі барбершопів дозволяє автоматизувати процес бронювання послуг, оптимізувати внутрішні бізнес-процеси та підвищити рівень обслуговування клієнтів. Така система забезпечує централізоване управління всіма філіями мережі, що є особливо важливим для масштабованого бізнесу.

Метою даної роботи є розробка алгоритмів функціонування та реалізація вебплатформи мережі барбершопів.

Сформульована мета визначає наступні завдання:

- аналіз теоретичних та практичних аспектів розробки вебплатформи;
- визначення структури та архітектури системи;
- аналіз існуючих аналогів у сфері онлайн-запису;
- обґрунтування вибору технологій реалізації;
- розробка структури бази даних;
- реалізація ролей користувачів;
- тестування системи.

Об'єктом дослідження є методи та засоби розробки веборієнтованих інформаційних систем.

Предметом дослідження є використання вебтехнологій для реалізації платформи управління мережею барбершопів.

У роботі використано методи об'єктно-орієнтованого проектування, вебпрограмування та аналізу бізнес-процесів.

Розроблена вебплатформа може бути застосована у практичній господарській діяльності мереж барбершопів для автоматизації основних бізнес-процесів. Її впровадження дозволяє організувати онлайн-запис клієнтів

без телефонних дзвінків, централізовано керувати філіями, послугами та розкладом майстрів, а також вести облік відгуків і формувати аналітичну звітність про завантаженість персоналу, популярність послуг і дохід мережі. Завдяки цьому власники та керівники отримують зручний інструмент для управління мережею й ухвалення обґрунтованих рішень, барбери – для організації власного робочого часу, а клієнти – для швидкого та зручного запису на послуги. Платформа є універсальною й може бути впроваджена в будь-якій мережі барбершопів незалежно від кількості філій, що підвищує рівень обслуговування та ефективність організації роботи закладів.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБПЛАТФОРМИ МЕРЕЖІ БАРБЕРШОПІВ

1.1. Постановка задачі

Вебплатформа мережі барбершопів – це онлайн-система, створена з метою автоматизації процесів управління закладами та забезпечення зручного онлайн-запису клієнтів на послуги. Суть проекту полягає у створенні єдиного цифрового середовища, яке дозволяє централізовано керувати мережею барбершопів, оптимізувати розклад роботи майстрів та забезпечити клієнтам швидкий і зручний доступ до сервісів.

Говорячи про подібні платформи, зазвичай вони включають такі основні компоненти:

- інформаційний ресурс для зберігання та відображення актуальної інформації про барбершопи мережі, перелік послуг, їх вартість, тривалість виконання та графік роботи майстрів;
- система онлайн-запису, що дозволяє клієнтам обирати філію, барбера, послугу, дату та час візиту;
- особисті кабінети користувачів, де клієнти можуть переглядати історію записів, скасовувати або редагувати бронювання, а також керувати власними персональними даними;
- адміністративна панель, яка забезпечує можливість керування барберами, послугами, філіями, графіками роботи та записами клієнтів;
- аналітичний модуль, що дозволяє формувати звіти про завантаженість майстрів, кількість записів, популярність послуг та інші показники діяльності мережі;
- система повідомлень та сповіщень, яка інформує клієнтів про підтвердження запису, зміни в розкладі або нагадування про візит.

Першим кроком реалізації вебплатформи є вибір технологій, на цьому етапі визначається стек технологій для розробки вебплатформи, вибираються фреймворки для клієнтської та серверної частини, а також забезпечується

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						9

масштабованість та безпека майбутньої системи.

Передбачається проєктування структури бази даних для зберігання інформації про філії барбершопів, барберів, клієнтів, послуг, записів та адміністраторів. Крім того, на цьому етапі здійснюється вибір та налаштування системи управління базами даних та забезпечується цілісність та узгодженість даних.

Окрема увага приділена інтерфейсу користувача, інтерфейс розробляється з використанням HTML, CSS та JavaScript, при цьому реалізується адаптивний дизайн для коректного відображення на різних пристроях, включаючи персональні комп'ютери, планшети та смартфони.

Функціонал включає реєстрацію та авторизацію користувачів, впровадження механізму онлайн-запису та перевірку доступності часу для запису, форум для спілкування користувачів на різні теми, профілі для барберів де вони можуть викладати свої роботи для портфолію, можливість залишати відгуки, система нагадувань. Також передбачено реалізацію управління розкладом барберів та формування статистичних даних та звітів.

Для функціоналу керування контентом розробляється панель управління для адміністратора та менеджера, з розміжними можливостями для обох ролей, яка дозволяє додавати, редагувати та видаляти інформацію про послуги, барберів та філії, а також керувати цінами, тривалістю послуг та переглядати зведену аналітику по кожній філії.

Перед використанням система проходить тестування та налагодження, перевіряється працездатність функціональних модулів, тестується взаємодія клієнтської та серверної частин, а також виявляються та усуваються помилки.

Завершальним аспектом є безпека, вона забезпечується шляхом шифрування паролів користувачів, контролю доступу відповідно до ролі, резервного копіювання бази даних та захисту від несанкціонованого доступу.

Підсумовуючи наведене, переді мною постають такі завдання:

- аналіз існуючих аналогів вебплатформ для барбершопів;
- збір та систематизація вимог до системи;

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						10

- проектування структури бази даних;
- реалізація ролей користувачів (Клієнт, Барбер, Адміністратор, Менеджер);
- розробка основного функціоналу онлайн-запису;
- тестування та оцінка працездатності системи.

1.2. Аналіз аналогів програмного продукту

Аналіз аналогів є важливим етапом розробки вебплатформи, оскільки дозволяє визначити сильні та слабкі сторони існуючих рішень, оцінити їх функціональні можливості та виявити напрями для вдосконалення власного програмного продукту.

У межах даної роботи було проаналізовано три вебсайти барбершопів:

1. BarberKod – <http://www.barberkod.team/>
2. DANDY Barbershop – <https://dandy.ua/>
3. Black Peppers Barbershop – <https://www.blackpeppers-barbershop.com/>

Метою аналізу є визначення функціональних можливостей, зручності використання, наявності системи онлайн-запису, адаптивності та інших характеристик, які можуть бути реалізовані у власній вебплатформі.

BarberKod є вебсайтом барбершопу, який презентує мережу закладів, їх послуги та команду майстрів, а його головна сторінка виконана в сучасному стильному оформленні (рис. 1.1).

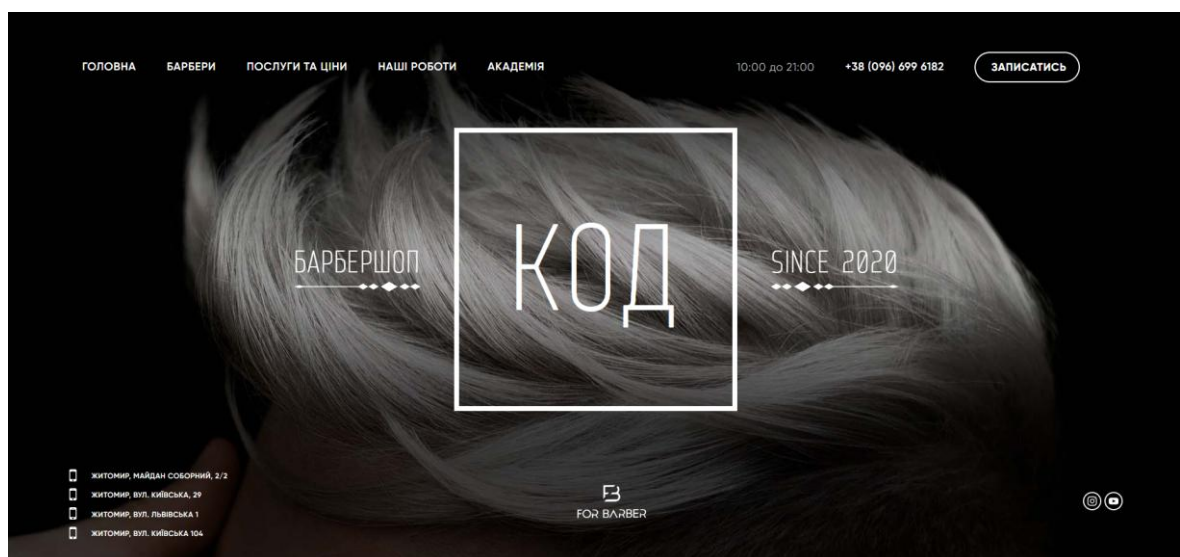


Рис. 1.1. Вигляд головної сторінки сайту BarberKod

Серед основних функцій сайту можна виділити перегляд інформації про барбершоп та його філії, ознайомлення з переліком послуг і їх вартістю, онлайн-запис на стрижку, а також перегляд інформації про барберів. Окрім цього, на сайті наведено контактну інформацію та карту розташування закладів, а також реалізовано інтеграцію із соціальними мережами. Команда майстрів представлена на окремій сторінці з фотографіями барберів (рис. 1.2).

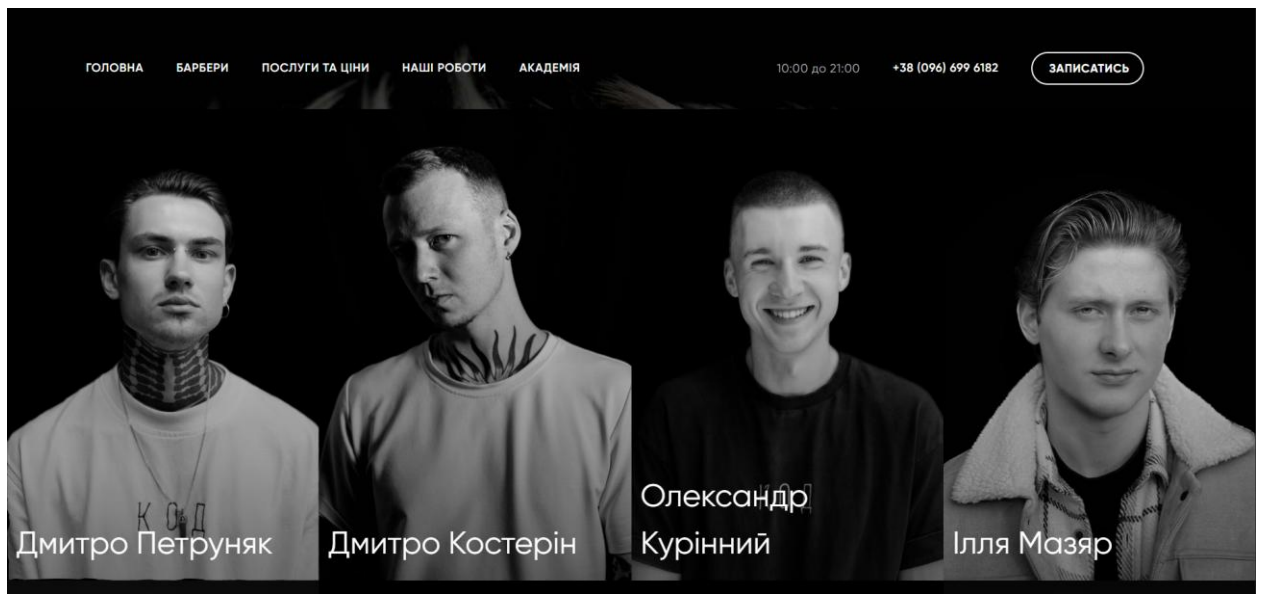


Рис. 1.2. Сторінка з командою майстрів сайту BarberKod

До переваг цього рішення належать наявність онлайн-запису, сучасний дизайн і зручна навігація, детальний опис послуг та адаптивність під мобільні пристрої. Також варто відзначити візуальну презентацію команди майстрів та інтеграцію з Instagram, що позитивно впливає на впізнаваність бренду.

На сайті відсутній особистий кабінет клієнта, а також немає можливості залишати відгуки безпосередньо на сайті. Крім того, спостерігаються обмежені можливості адміністрування для централізованої мережі, відсутність аналітичних інструментів для керівництва та відсутність форуму для обговорення робіт барберів.

DANDY є мережею барбершопів, яка має стильний корпоративний сайт із можливістю онлайн-запису (рис. 1.3).

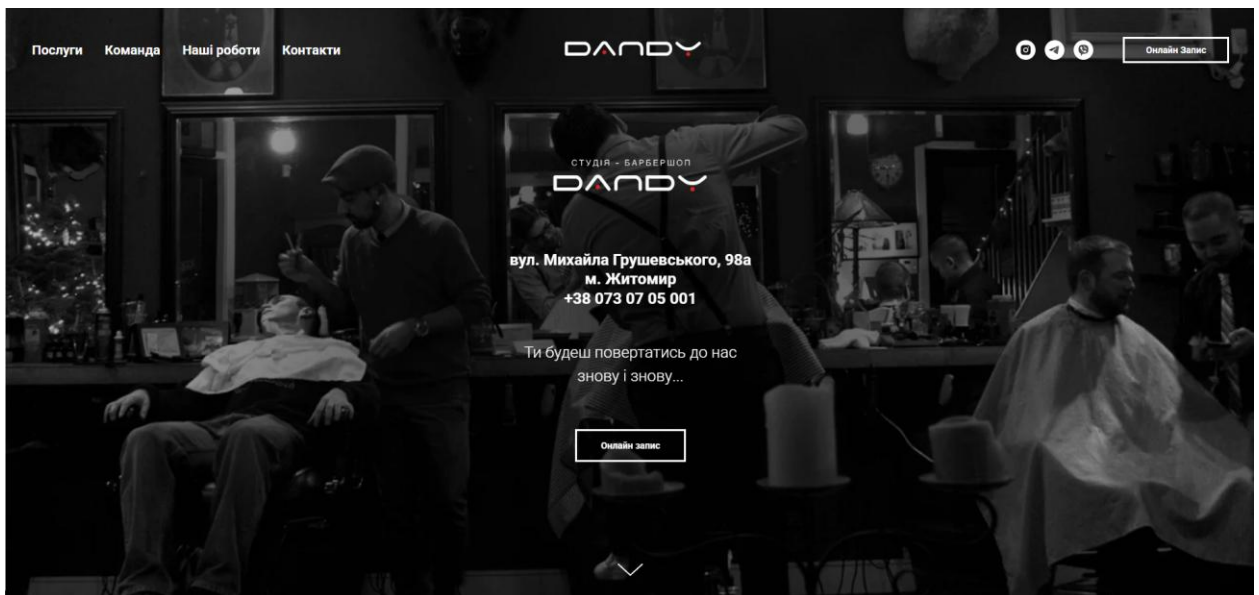


Рис. 1.3. Вигляд сайту DANDY Barbershop

Серед основних функцій сайту можна виділити онлайн-запис через інтегровану систему бронювання, перегляд філії на мапі разом із контактною інформацією, а також ознайомлення з послугами та цінами. Окрім цього, користувач має змогу переглянути команду барберів та отримати інформацію про акції, а сам сайт виконано з використанням адаптивного дизайну.

До переваг цього рішення належать чітка структура сайту, зручний процес запису та гарна оптимізація для мобільних пристроїв, що забезпечує комфортну взаємодію з ресурсом.

Водночас сайт має низку недоліків: відсутній особистий кабінет клієнта та система онлайн-оплати, а також не передбачено розширеного функціоналу для адміністраторів. Крім того, відсутні статистика або звітність для керівництва та форуму для обговорення робіт барберів.

Black Peppers є барбершопом із акцентом на бренд та стиль, який має інформативний сайт із можливістю запису, а його головна сторінка виконана у темному стильному оформленні (рис. 1.4). Серед основних функцій сайту можна виділити онлайн-запис, перелік послуг та інформацію про команду.

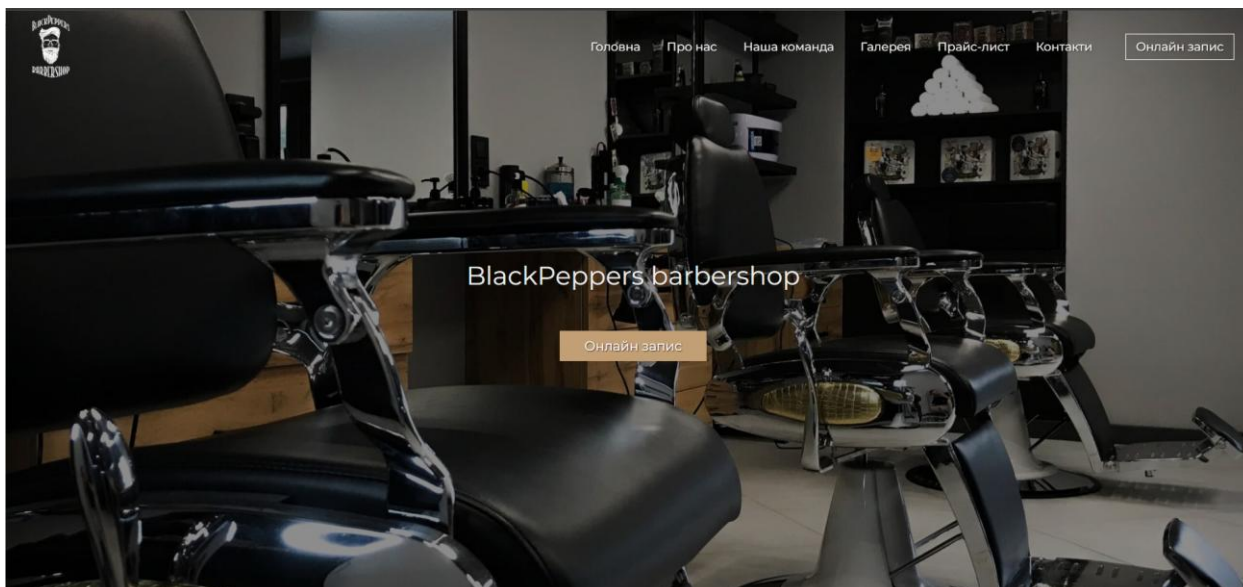


Рис. 1.4. Вигляд головної сторінки сайту Black Peppers Barbershop

Окрім цього, на сайті наведено контактні дані й посилання на соціальні мережі, а також передбачено галерею фотографій, де представлено заклад і його команду майстрів (рис. 1.5).

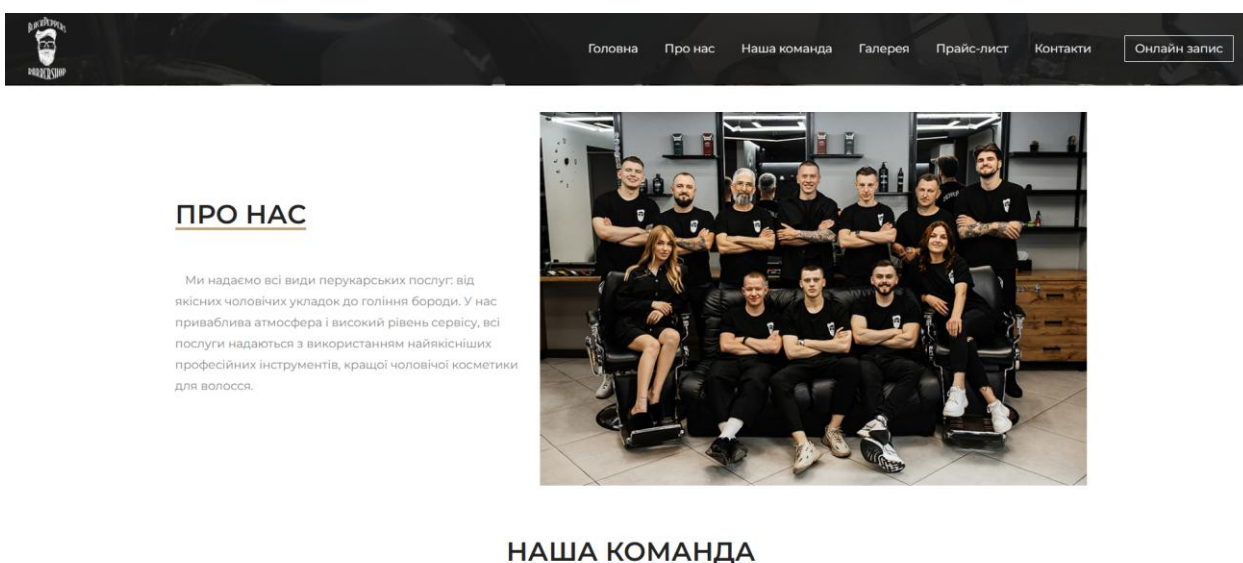


Рис. 1.5. Сторінка «Про нас» та команда сайту Black Peppers Barbershop

До переваг цього рішення належать візуально привабливий дизайн, простий процес запису та інтуїтивно зрозумілий інтерфейс, а також адаптація під мобільні пристрої, що забезпечує комфортну взаємодію з ресурсом.

На сайті відсутні реєстрація та авторизація користувачів, а також особистий кабінет. Крім того, немає аналітики, спостерігається обмежений функціонал для мережі з декількома філіями, відсутня онлайн-оплата та форум

для обговорення робіт барберів.

Таблиця 1.1

Порівняльний аналіз аналогів програмного продукту

Назва функції	BarberKod	DANDY	Black Peppers
Онлайн-запис	+	+	+
Перелік послуг з цінами	+	+	+
Інформація про барберів	+	+	+
Наявність декількох філій	+	+	-
Особистий кабінет клієнта	-	-	-
Відгуки на сайті	-	-	-
Адаптивність під мобільні пристрої	+	+	+
Адміністративна панель	+ -	+ -	-
Аналітика та звітність	-	-	-
Інтеграція з соцмережами	+	+	+

1.3. Вибір архітектури вебплатформи

Вибір архітектури веборієнтованої системи є одним із найважливіших етапів проєктування програмного продукту, оскільки саме архітектурне рішення визначає структуру системи, спосіб взаємодії її компонентів, рівень продуктивності, безпеки та можливості подальшого масштабування [16]. Для вебплатформи мережі барбершопів архітектура повинна враховувати специфіку предметної області, зокрема наявність кількох філій, розмежування ролей користувачів, організацію онлайн-запису клієнтів, централізоване управління розкладом майстрів, збереження історії записів та формування аналітичної інформації для адміністрації.

Під час вибору архітектури можливі підходи оцінювалися за кількома ключовими критеріями, а саме масштабованістю, зручністю супроводу й внесення змін, продуктивністю під час обробки запитів, рівнем безпеки даних, а також швидкістю та простотою розробки в умовах обмежених ресурсів.

Остаточне рішення приймалося як компроміс між цими критеріями, що найкраще відповідає масштабу та призначенню системи [16].

Правильно обрана архітектура дозволяє забезпечити стабільну роботу системи навіть при зростанні кількості клієнтів і розширенні мережі барбершопів. Крім того, вона впливає на швидкість обробки запитів, зручність супроводу програмного забезпечення та можливість подальшого вдосконалення функціоналу. У випадку неефективного архітектурного рішення можуть виникати труднощі із масштабуванням, зростати витрати на підтримку та ускладнюватися процес внесення змін до програмного коду.

Під час проєктування вебплатформи було розглянуто кілька підходів до організації архітектури системи. Монолітна архітектура передбачає реалізацію всіх компонентів програмного забезпечення в межах єдиного проєкту, який розгортається як цілісна система. У такому випадку презентаційний рівень, бізнес-логіка та механізми доступу до даних функціонують у межах одного серверного застосунку. Даний підхід є відносно простим у реалізації та зручним для невеликих і середніх проєктів, оскільки спрощує процес розробки, тестування та розгортання, а виклики між частинами системи відбуваються безпосередньо в межах процесу без додаткових мережових затримок [12]. Водночас при значному зростанні функціоналу система може ускладнюватися, а внесення змін потребуватиме більш ретельного контролю залежностей між модулями.

Мікросервісна архітектура, на відміну від монолітної, передбачає розподіл функціоналу на окремі незалежні сервіси, кожен з яких відповідає за конкретну частину системи. Такий підхід дозволяє масштабувати окремі компоненти незалежно один від одного та підвищує гнучкість розвитку програмного продукту. Проте впровадження мікросервісної архітектури потребує складнішої інфраструктури, налаштування взаємодії між сервісами, забезпечення узгодженості даних та додаткових ресурсів для підтримки [17]. З огляду на те, що вебплатформа для мережі барбершопів розробляється як інформаційна система середнього масштабу з тісно пов'язаними даними,

застосування мікросервісної архітектури на початковому етапі є недоцільним.

Також було розглянуто шаровий підхід до побудови системи, який передбачає логічний поділ програмного забезпечення на рівні відповідальності. У межах такого підходу виділяється презентаційний рівень, що відповідає за взаємодію з користувачем, рівень бізнес-логіки, який реалізує основні правила функціонування системи, та рівень доступу до даних, що забезпечує роботу з базою даних. Шарова структура дозволяє впорядкувати код, підвищити його читабельність та спростити подальшу модифікацію.

У розроблюваній системі шаровий підхід реалізовано безпосередньо у структурі серверної частини: презентаційному рівню відповідають маршрути та контролери, рівень бізнес-логіки реалізує правила роботи системи, зокрема перевірку доступності часу й розмежування прав доступу, а рівень доступу до даних відповідає за виконання запитів до бази через єдиний модуль підключення. Такий поділ забезпечує слабку зв'язаність складових, підвищує тестованість і полегшує супровід, оскільки зміни на одному рівні не вимагають переписування інших [16, 19].

З урахуванням масштабу проєкту та поставлених функціональних вимог серверну частину вебплатформи доцільно реалізувати із застосуванням монолітної архітектури з внутрішнім поділом на логічні шари. Таке рішення є оптимальним для системи, яка не передбачає надвеликих навантажень, але водночас повинна забезпечувати стабільну роботу онлайн-запису, облік клієнтів, керування розкладом барберів та формування звітності. Монолітна структура спрощує розробку, тестування та розгортання, а також дозволяє швидко впроваджувати новий функціонал без необхідності налаштування складної взаємодії між окремими сервісами.

Архітектура вебплатформи є клієнт-серверною. Клієнтська частина реалізується за допомогою сучасних вебтехнологій і відповідає за формування інтерфейсу користувача, відображення інформації про послуги, майстрів і розклад, а також за передачу запитів до серверної частини. Серверна частина забезпечує обробку цих запитів, реалізацію механізмів авторизації та

аутентифікації, перевірку доступності часу для запису, збереження даних у базі та формування відповідей для клієнта. Взаємодія між клієнтом і сервером здійснюється через API, що забезпечує чітке розмежування відповідальності між компонентами системи.

Клієнтська частина побудована як односторінковий застосунок на основі компонентного підходу, за якого інтерфейс складається з окремих повторно використовуваних блоків і обмінюється з сервером лише необхідними даними без повного перезавантаження сторінки [1]. Взаємодія організована за архітектурним стилем REST, у якому сутності системи подаються як ресурси з уніфікованими адресами, а кожен запит містить усю інформацію для його обробки й не залежить від збереженого на сервері стану. Прямим наслідком цього є реалізація автентифікації на основі підписаних токенів: сервер не зберігає сесій у пам'яті, що спрощує розподіл навантаження та закладає основу для подальшого горизонтального масштабування [28].

Рівень доступу до даних реалізовано на основі реляційної бази даних, що забезпечує надійне та узгоджене зберігання інформації про клієнтів, записи, послуги й філії та підтримує транзакції, критично важливі для уникнення конфліктів під час одночасного бронювання [4]. Питання безпеки розглядалося як невід'ємна складова архітектури: на серверному рівні реалізовано розмежування прав доступу за ролями, перевірку дійсності токенів, зберігання паролів виключно у вигляді хешів та контроль доступу з інших джерел, причому ці механізми винесено у проміжні обробники, що узгоджується із шаровою структурою системи [28].

Обрана архітектура дозволяє забезпечити достатній рівень масштабованості шляхом збільшення обчислювальних ресурсів сервера у разі зростання навантаження. Крім того, вона створює передумови для подальшої модернізації системи, зокрема можливості поступового переходу до більш розподіленої моделі у разі розширення мережі барбершопів. Таким чином, клієнт-серверна модель із монолітною серверною реалізацією та шаровою структурою є обґрунтованим і доцільним рішенням для розробки

вебплатформи мережі барбершопів, оскільки забезпечує баланс між простотою реалізації, продуктивністю та можливістю подальшого розвитку системи.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Під час розробки вебплатформи мережі барбершопів вибір інструментальних засобів здійснювався з урахуванням функціональних вимог системи, перспектив масштабування бізнесу, необхідності стабільної роботи онлайн-запису клієнтів, а також зручності подальшої підтримки програмного забезпечення. Оскільки платформа передбачає багатокористувацький режим роботи, розмежування ролей (клієнт, барбер, адміністратор), управління розкладом, облік філій та збереження історії записів, було обрано сучасний стек вебтехнологій, що забезпечує надійність, продуктивність і гнучкість.

Однією з визначальних переваг обраного стеку є використання єдиної мови програмування JavaScript як на клієнтській, так і на серверній частині, що спрощує розробку та повторне використання коду. Вибір Node.js як серверного середовища додатково обумовлений його подієво-орієнтованою моделлю з неблокувальним введенням-виведенням, яка добре підходить для вебзастосунків із великою кількістю одночасних з'єднань і переважно операціями вводу-виводу [11]. Оскільки робота платформи полягає здебільшого в обробці запитів до бази даних і формуванні відповідей, така модель дозволяє ефективно обслуговувати багатьох користувачів без надмірного споживання ресурсів сервера.

Клієнтську частину системи доцільно реалізувати з використанням React. Дана технологія базується на компонентному підході, що дозволяє структурувати інтерфейс у вигляді незалежних функціональних блоків [1]. Для вебплатформи барбершопів це особливо важливо, оскільки інтерфейс містить різні логічні частини: каталог послуг, сторінки окремих філій, профілі барберів, модуль онлайн-запису, особистий кабінет користувача та адміністративну панель. Компонентна структура спрощує повторне

використання елементів інтерфейсу, підвищує читабельність коду та полегшує внесення змін у майбутньому.

React використовує механізм віртуального DOM, що забезпечує ефективне оновлення сторінок без повного перезавантаження. Це дозволяє створити швидкий і зручний інтерфейс для користувача, що особливо актуально при виборі дати та часу запису, фільтрації майстрів або оновленні статусу бронювання. Плавна взаємодія з інтерфейсом без затримок позитивно впливає на користувацький досвід та формує довіру до сервісу.

Додатковою перевагою React є розвинена екосистема, завдяки якій для типових завдань існують готові перевірені рішення [9]. Зокрема, для навігації між сторінками односторінкового застосунку використано бібліотеку маршрутизації, що реалізує переходи без перезавантаження сторінки, а для обміну даними із сервером – клієнтську бібліотеку HTTP-запитів. Це пришвидшує розробку та підвищує надійність, оскільки замість власних рішень застосовуються широко вживані компоненти.

Для побудови серверної частини разом із Node.js доцільно використовувати фреймворк Express. Даний інструмент забезпечує зручну організацію серверної логіки та спрощує обробку HTTP-запитів [3]. Express дозволяє швидко створювати API, через які клієнтська частина взаємодіє з сервером, що є важливим для реалізації таких функцій, як авторизація користувачів, створення записів, отримання списку послуг або розкладу майстрів.

Завдяки гнучкій структурі Express можна легко організувати проєкт за принципом розділення відповідальностей, виділивши окремі модулі для роботи з користувачами, записами, барберами та філіями. Це підвищує зрозумілість коду та спрощує його подальшу підтримку. Крім того, Express підтримує використання middleware, що дозволяє реалізувати перевірку автентифікації, обробку помилок, логування запитів та інші допоміжні функції.

Окрім основних технологій, у проєкті використано низку допоміжних

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						20

бібліотек: для автентифікації застосовано підписані токени, паролі зберігаються у захешованому вигляді, завантаження зображень реалізовано через спеціалізований проміжний обробник, а доступ до бази даних здійснюється через офіційний драйвер із підтримкою параметризованих запитів і пулу з'єднань. Розробка велася в сучасному інтегрованому середовищі з керуванням версіями через систему контролю версій і віддалений репозиторій, а залежності обох частин проєкту встановлювалися за допомогою стандартного менеджера пакетів [25].

Використання Express у поєднанні з Node.js дозволяє створити стабільний та масштабований серверний додаток, який ефективно обробляє запити клієнтів і забезпечує швидку взаємодію з базою даних PostgreSQL. Це особливо важливо для вебплатформи барбершопів, де необхідно оперативно обробляти запити на бронювання, уникати конфліктів у розкладі та гарантувати коректність даних.

Для збереження та обробки даних обрано систему керування базами даних PostgreSQL. Вибір обумовлений її надійністю, підтримкою транзакцій та високим рівнем цілісності даних [4]. У контексті вебплатформи мережі барбершопів база даних повинна забезпечувати коректне збереження інформації про клієнтів, записи, послуги, розклад майстрів, філії та фінансові операції. PostgreSQL підтримує складні запити, механізми блокування транзакцій та гнучку систему керування доступом, що є критично важливим для запобігання конфліктам під час одночасного бронювання часу різними користувачами. Додатковими чинниками на користь цієї СУБД стали вбудовані засоби забезпечення цілісності, зокрема зовнішні ключі та перелічувані типи, підтримка операцій для роботи з часовими інтервалами, що використовуються під час перевірки доступності часу, а також те, що вона є вільним програмним забезпеченням з відкритим кодом [13].

Щодо вимог до апаратного забезпечення серверної частини, система повинна функціонувати на сервері з обсягом оперативної пам'яті не менше 16 Гб, що забезпечить стабільну роботу при одночасному доступі декількох

десятків активних користувачів. Необхідний обсяг вільного дискового простору становить не менше 100 Гб для зберігання бази даних, резервних копій та журналів системи. Процесор із тактовою частотою не нижче 1 GHz забезпечить достатню продуктивність для обробки запитів та виконання серверної логіки. Обов'язковою умовою є наявність стабільного доступу до мережі Інтернет, оскільки система функціонує у вебсередовищі. Як операційна система може використовуватися Windows або Unix-подібні системи. Водночас вимоги до клієнтського пристрою є значно нижчими: для роботи з платформою достатньо будь-якого сучасного комп'ютера, планшета чи смартфона з актуальним вебпереглядачем і доступом до Інтернету, оскільки основне обчислювальне навантаження припадає на сервер.

Таким чином, обраний технологічний стек відповідає сучасним стандартам веброзробки, забезпечує стабільність, масштабованість і зручність підтримки системи. Використання React, Node.js з Express та PostgreSQL дозволяє створити продуктивну та надійну вебплатформу для мережі барбершопів із можливістю подальшого розширення функціоналу та інтеграції нових сервісів.

Висновки до першого розділу

У першому розділі проаналізовано напрями використання інформаційних технологій для реалізації вебплатформи мережі барбершопів та сформовано теоретичне підґрунтя для подальшої розробки системи. У межах постановки задачі визначено основне призначення платформи, яке полягає в автоматизації процесів управління закладами мережі та забезпеченні зручного онлайн-запису клієнтів на послуги. Окреслено ключові компоненти, що мають увійти до складу системи, а саме інформаційний ресурс із відомостями про заклади та послуги, модуль онлайн-запису, особисті кабінети користувачів, адміністративну панель, аналітичний модуль і систему сповіщень. Також сформульовано технічні вимоги до вибору технологій, проєктування бази даних, розробки інтерфейсу, реалізації основного функціоналу, управління контентом, тестування та забезпечення безпеки [19].

Окрему увагу приділено аналізу наявних аналогів програмного продукту, у ході якого розглянуто три діючі сайти барбершопів – BarberKod, DANDY Barbershop та Black Peppers Barbershop. Дослідження їхнього функціоналу показало, що всі ці рішення забезпечують базовий онлайн-запис, перегляд переліку послуг із цінами та презентацію команди майстрів, проте мають низку спільних обмежень. Зокрема, у них відсутні особистий кабінет клієнта, система лояльності, онлайн-оплата, інструменти аналітики для керівництва та форум для обговорення робіт барберів. Результати порівняння функціональних можливостей цих сайтів дозволили обґрунтувати доцільність розробки власної вебплатформи, яка усуває виявлені недоліки й пропонує ширший набір можливостей для всіх категорій користувачів з урахуванням принципів зручності використання та інтуїтивності інтерфейсу [22, 23].

На основі вивчення предметної області обґрунтовано вибір архітектури системи. Порівняно монолітний, мікросервісний та шаровий підходи до організації програмного забезпечення, після чого для серверної частини обрано клієнт-серверну модель із монолітною реалізацією та внутрішнім поділом на логічні шари [16, 17]. Взаємодію між клієнтською та серверною частинами вирішено організувати за принципами REST, що забезпечує чітке розмежування відповідальності між компонентами системи [18, 30]. Таке рішення забезпечує баланс між простотою розробки, продуктивністю й можливістю подальшого масштабування, що відповідає рівню проекту як інформаційної системи середнього масштабу та дозволяє за потреби поступово переходити до більш розподіленої моделі під час розширення мережі.

Крім того, обґрунтовано вибір інструментальних засобів та визначено вимоги до апаратного забезпечення серверної частини. Для реалізації клієнтської частини обрано бібліотеку React, серверну частину вирішено будувати на основі середовища Node.js із фреймворком Express, а для зберігання й обробки даних – систему керування базами даних PostgreSQL [1, 3, 4]. Установлено, що такий технологічний стек відповідає сучасним

стандартам веброзробки та забезпечує надійність, продуктивність і зручність подальшого супроводу системи [12, 15]. Отримані в розділі результати створюють підґрунтя для переходу до проєктування вебдодатку, що розглядається в наступному розділі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБДОДАТКУ

2.1. Визачення варіантів використання та структури системи

Для забезпечення стабільної та безперебійної роботи вебплатформи мережі барбершопів необхідно визначити загальні вимоги, яким має відповідати система. Ці вимоги спрямовані на те, щоб платформа була зручною у використанні, зрозумілою для різних категорій користувачів і водночас допомагала автоматизувати основні процеси роботи барбершопу. Головною метою розробки є спрощення запису клієнтів на послуги, налагодження зручної взаємодії між клієнтами та працівниками, а також покращення управління всією мережею барбершопів. У результаті має бути створено вебсайт, який об'єднує в собі всі необхідні інструменти для роботи клієнтів, барберів, адміністраторів і менеджерів.

Користувачі платформи повинні мати можливість швидко створити обліковий запис і виконати вхід у систему, після чого їм відкривається доступ до основного функціоналу. Вони зможуть переглядати список барбершопів, ознайомлюватися з доступними послугами та дізнаватися інформацію про барберів і їхній графік роботи. Однією з ключових можливостей є онлайн-запис, під час якого користувач обирає послугу, конкретного барбера та зручний час. Система також повинна дозволяти змінювати або скасовувати запис у разі потреби та переглядати історію відвідувань, що робить користування більш зручним і прозорим.

Працівники, які мають доступ до адміністративної частини, повинні отримати інструменти для керування всією інформацією в системі. Це передбачає можливість додавати нові локації барбершопу, змінювати дані про них, працювати зі списком послуг, редагувати їх або видаляти. Окрему увагу приділено управлінню графіками барберів, адже саме від цього залежить коректність записів клієнтів. Адміністратори повинні бачити всі записи, мати можливість їх підтверджувати та контролювати завантаженість персоналу. Крім того, система повинна надавати базові звіти про кількість клієнтів, популярність послуг та рівень зайнятості барберів, що допомагає приймати

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						25

обґрунтовані управлінські рішення.

З погляду користувача вебплатформа має бути максимально простою і зручною, оскільки рівень інтуїтивності інтерфейсу безпосередньо впливає на досвід взаємодії та довіру до сервісу [22, 23]. Людина повинна без труднощів знайти потрібний барбершоп, обрати послугу, переглянути доступний час і записатися. Інтерфейс має працювати однаково добре як на комп'ютері, так і на телефоні чи планшеті, оскільки значна частина користувачів послуговуватиметься системою саме з мобільних пристроїв. Важливо, щоб усі елементи були зрозумілими, а сам процес запису займав мінімум часу.

Функціонально система повинна забезпечувати реєстрацію користувачів і розподіл ролей, адже різні типи користувачів мають різні можливості. Уся інформація про барбершопи, послуги, барберів, клієнтів і записи має зберігатися в базі даних і бути доступною для обробки. Онлайн-запис повинен працювати в реальному часі, щоб уникати ситуацій, коли той самий час одночасно намагаються зайняти кілька клієнтів, що вимагає коректного опрацювання конкурентних запитів на рівні бази даних [4]. Адміністративна частина має дозволяти змінювати дані без складних дій, щоб працівники могли швидко реагувати на зміни.

Система реалізує низку додаткових можливостей, що розширюють взаємодію користувачів із платформою. Зокрема, передбачено форум для обговорень, де користувачі можуть створювати теми, коментувати їх і оцінювати дописи вподобайками, систему відгуків та оцінок роботи барберів, а також портфоліо барберів із роботами у форматі «до/після». Важливу роль відіграє механізм сповіщень, який інформує користувачів про підтвердження запису, його зміни чи нагадування про візит [22].

Важливу роль відіграють і нефункціональні вимоги, система повинна бути простою у використанні, щоб новий користувач міг розібратися з нею за короткий час, а відгук на дії користувача має бути швидким, без помітних затримок. Також необхідно забезпечити стабільну роботу системи навіть за одночасного використання великою кількістю людей та передбачити захист

У результаті аналізу всіх цих вимог можна визначити, як саме користувачі взаємодіятимуть із платформою, які дії виконуватимуть і які можливості для цього потрібні. Для наочного подання цієї взаємодії побудовано діаграми варіантів використання за нотацією UML, які окремо описують поведінку кожної ролі – клієнта, барбера, адміністратора та менеджера [16]. Загальну структуру системи та зв'язки між її основними класами, зокрема ієрархію ролей користувачів і пов'язані з ними сутності, подано у вигляді діаграми класів (додаток А).

```

graph LR
    Client((Клієнт))
    UC1(Реєстрація / вхід)
    UC2(Перегляд барбершопів)
    UC3(Перегляд послуг)
    UC4(Перегляд барберів)
    UC5(Перегляд портфоліо барбера)
    UC6(Онлайн-запис)
    UC7(Перегляд історії записів)
    UC8(Скасування запису)
    UC9(Залишити відгук та оцінку)
    UC10(Користування форумом)
    UC11(Отримання сповіщень)
    UC12(Керування профілем)
    UC13(Вибір дати та часу)
    UC14(Перевірка доступності)
    UC15(Оцінювання лайки)
    UC16(Коментування)
    UC17(Створення поста)

    Client --> UC1
    Client --> UC2
    Client --> UC3
    Client --> UC4
    Client --> UC5
    Client --> UC6
    Client --> UC7
    Client --> UC8
    Client --> UC9
    Client --> UC10
    Client --> UC11
    Client --> UC12

    UC6 -.->|«include»| UC13
    UC6 -.->|«include»| UC14
    UC10 -.->|«include»| UC15
    UC10 -.->|«include»| UC16
    UC10 -.->|«include»| UC17
  
```

					ИПЗ.КР.Б-121-26-ПЗ	Арк.
						27

Функціональні можливості барбера відображено на діаграмі варіантів використання барберської частини (рис. 2.2). Після входу барбер отримує доступ до робочого середовища, де основна увага приділяється розкладу та записам клієнтів. Барбер може переглядати власний графік і записи клієнтів із відомостями про послугу, час та відвідувача, а також підтверджувати записи, контролюючи актуальність бронювань. Через відношення «include» редагування власного графіку охоплює встановлення вихідних днів та зміну робочих годин, що безпосередньо впливає на доступність часу для запису. Окремо передбачено ведення портфолію з додаванням робіт у форматі «до/після», перегляд отриманих відгуків і власного рейтингу, а також участь в обговореннях на форумі.



Рис. 2.2. Діаграма варіантів використання барберської частини

Можливості адміністратора подано на діаграмі варіантів використання адміністративної частини (рис. 2.3). Ця роль має розширений доступ до керування системою. Адміністратор управляє послугами, що через відношення «include» охоплює їх додавання, редагування, видалення та керування категоріями послуг, а також управляє барберами, зокрема додає нових працівників і змінює їхні дані. Окремо виділено управління записами клієнтів із можливістю їх підтвердження та скасування. Адміністратор також керує видимістю відгуків і здійснює модерацію форуму, що включає закріплення та блокування постів і видалення коментарів, а також формує звіти про роботу системи.



Рис. 2.3. Діаграма варіантів використання адміністративної частини

Функціонал менеджера відображено на діаграмі варіантів використання менеджерської частини (рис. 2.4). Менеджер відповідає за глобальніше управління мережею та через відношення «include» керує барбершопами, тобто додає нові філії, редагує інформацію про них або видаляє їх, а також

налаштовує графік роботи філій. Крім того, він переглядає аналітику, яка охоплює аналіз популярності послуг та відвідуваності й дозволяє оцінити ефективність роботи мережі. Менеджер також формує фінансові звіти для контролю прибутків і витрат та контролює завантаженість персоналу, що допомагає рівномірно розподіляти навантаження між барберами й уникати перевантажень або простоїв.

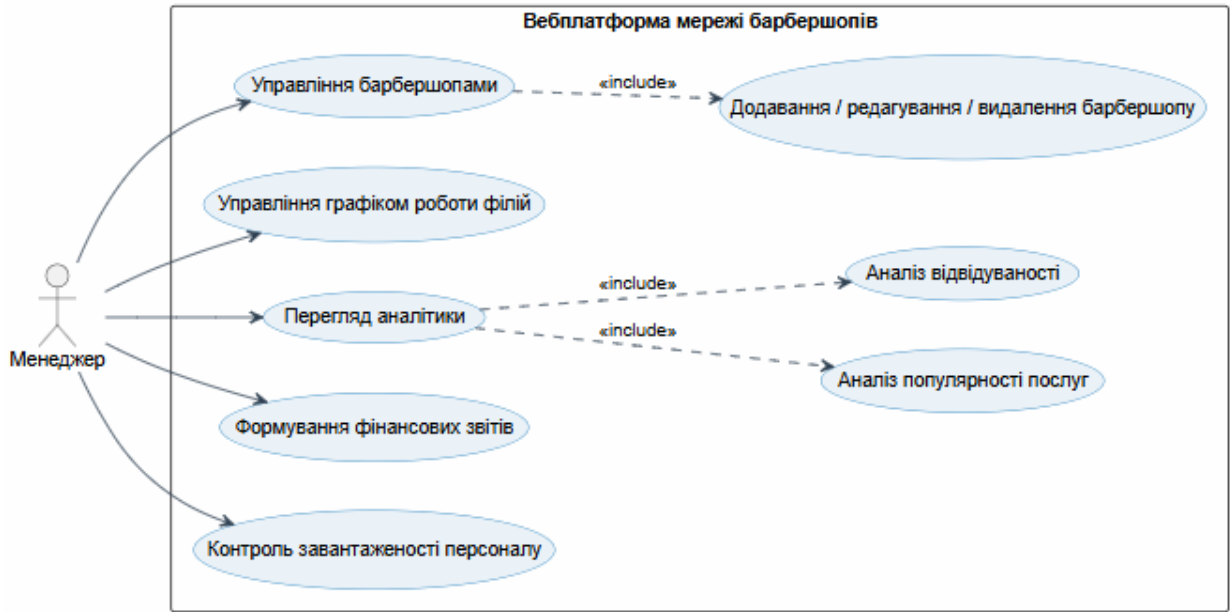


Рис. 2.4. Діаграма варіантів використання менеджерської частини

Таким чином, кожна з ролей у системі має чітко визначений набір можливостей, що дозволяє розмежувати доступ і зробити роботу платформи зрозумілою та ефективною. Побудовані діаграми варіантів використання наочно показують, як саме користувачі взаємодіють із системою, і допомагають краще зрозуміти логіку її роботи ще на етапі проєктування, що є підґрунтям для подальшої розробки структури бази даних.

У системі виділено 4 основних актори, нижче надано таблицю з основними сценаріями використання вебплатформи мережі барбершопів (табл. 2.1.)

Основні сценарії використання

Основний актор	Найменування	Формулювання
Клієнт	Авторизація	Процес реєстрації або входу в систему за допомогою електронної пошти та пароллю для отримання доступу до відповідного функціоналу
	Перегляд барбершопів та послуг	Перегляд списку доступних філій мережі з відображенням назви, адреси та переліку послуг із цінами
	Вибір барбера	Вибір конкретного барбера для переходу до детальної інформації про його спеціалізацію, рейтинг та доступний час
	Перегляд портфоліо барбера	Ознайомлення з виконаними роботами барбера у форматі «до/після» для оцінки якості його послуг
	Вибір дати та часу	Відображення доступних часових слотів обраного барбера в реальному часі з їх статусами
	Онлайн-запис	Процес резервування обраного часу та підтвердження бронювання послуги після перевірки доступності інтервалу
	Перегляд історії записів	Відображення списку всіх попередніх та активних записів клієнта із зазначенням статусу кожного
	Скасування запису	Процес відміни активного бронювання з вивільненням зарезервованого часу
	Залишення відгуку та оцінки	Створення відгуку з числовою оцінкою роботи барбера після відвідування
	Користування форумом	Створення тем, написання коментарів та оцінювання дописів вподобайками в межах тематичних обговорень
	Отримання сповіщень	Отримання повідомлень про підтвердження запису, його зміни або нагадування про візит
	Керування профілем	Перегляд та редагування персональних даних власного облікового запису
Барбер	Авторизація	Процес входу в систему за допомогою електронної пошти та пароллю для отримання доступу до відповідного функціоналу
	Перегляд власного розкладу	Відображення персонального графіку роботи барбера із зазначенням усіх запланованих записів на обраний період
	Перегляд записів клієнтів	Отримання детальної інформації про клієнтів, записаних до барбера, із зазначенням послуги, дати та часу
	Підтвердження запису	Опрацювання запиту клієнта, за якого барбер підтверджує бронювання, після чого система оновлює статус та повідомляє клієнта

	Редагування власного графіку	Встановлення або зміна робочих годин і вихідних днів барбера для коректного відображення доступності в системі
	Ведення портфоліо	Додавання та оформлення власних робіт у форматі «до/після» для формування портфоліо
	Перегляд відгуків та рейтингу	Перегляд отриманих від клієнтів відгуків та власного середнього рейтингу
	Участь у форумі	Створення дописів та коментарів у тематичних обговореннях
Адміністратор	Авторизація	Процес входу в систему за допомогою електронної пошти та пароллю для отримання доступу до відповідного функціоналу
	Управління послугами	Додавання, редагування та видалення послуг, а також керування їх категоріями
	Управління барберами	Додавання нових барберів та редагування інформації про них
	Управління записами	Перегляд усіх записів клієнтів із можливістю їх підтвердження та скасування
	Управління відгуками	Керування видимістю відгуків клієнтів для модерації контенту
	Модерація форуму	Закріплення та блокування публікацій, а також видалення коментарів, що порушують правила
	Формування звітів	Формування звітів про роботу системи, кількість клієнтів та популярність послуг
Менеджер	Авторизація	Процес входу в систему за допомогою електронної пошти та пароллю для отримання доступу до відповідного функціоналу
	Управління барбершопами	Додавання нових філій мережі, редагування їх інформації або закриття існуючих
	Управління графіком роботи філій	Налаштування годин роботи кожної філії за днями тижня для коректного відображення доступності
	Перегляд аналітики	Перегляд зведеної аналітики по філії – аналіз популярності послуг та динаміки відвідуваності за обраний період
	Формування фінансових звітів	Генерація фінансової звітності по філії з деталізацією доходів за послугами та барберами за визначений часовий проміжок
	Контроль завантаженості персоналу	Моніторинг рівня завантаженості кожного барбера філії для оптимізації розподілу робочого навантаження

Наведена специфікація варіантів використання дозволяє чітко окреслити межі відповідальності кожної ролі та визначити повний перелік дій, доступних користувачам системи. Клієнт орієнтований на пошук послуг, онлайн-запис і взаємодію зі спільнотою через відгуки та форум, барбер зосереджений на веденні власного розкладу, опрацюванні записів і формуванні портфоліо, тоді

як адміністратор і менеджер відповідають за керування контентом та управління мережею на операційному й аналітичному рівнях відповідно. Такий розподіл функцій забезпечує розмежування прав доступу, унеможлиблює виконання дій поза межами повноважень ролі та робить логіку роботи платформи прозорою. Сформований перелік варіантів використання є основою для проєктування структури бази даних і подальшої програмної реалізації системи, оскільки безпосередньо визначає, які дані мають зберігатися та які операції над ними необхідно підтримувати.

2.2. Розробка структури бази даних

Зберігання всіх даних, які використовуються у системі, відбувається за допомогою системи керування базами даних PostgreSQL. Це реляційна СУБД, яка підтримує транзакції, зовнішні ключі та користувацькі перелічувані типи, що забезпечує високий рівень цілісності й узгодженості даних [4].

Усього в базі даних налічується 22 таблиці: users, client_profiles, refresh_tokens, barbershops, barbershop_hours, barbers, barber_services, barber_schedules, barber_absences, barber_portfolio, barber_rating_cache, service_categories, services, appointments, reviews, forum_categories, forum_posts, forum_post_photos, forum_comments, forum_comment_history, forum_likes та notifications. Загальну структуру бази даних і зв'язки між таблицями наведено на ER-діаграмі (Додаток В).

Для полів, що можуть набувати лише обмеженого набору значень, застосовано перелічувані типи PostgreSQL. Тип user_role визначає роль користувача та розмежовує рівні доступу, тип appointment_status описує статус запису протягом його життєвого циклу, а тип notification_type задає категорію сповіщення. Опис основних таблиць бази даних наведено в таблицях 2.2–2.12.

Таблиця 2.2

Опис таблиці users

Назва поля	Тип	Опис
id	integer (PK)	Унікальний ідентифікатор користувача
email	varchar(255)	Електронна пошта (логін)
password_hash	varchar(255)	Хешований пароль

продовж. табл. 2.2

role	user_role	Роль користувача
first_name	varchar(100)	Ім'я
last_name	varchar(100)	Прізвище
phone	varchar(20)	Номер телефону
avatar_url	text	Посилання на аватар
is_active	boolean	Ознака активності облікового запису
barbershop_id	integer (FK)	Філія (для барберів і менеджерів)
created_at	timestamptz	Дата і час створення
updated_at	timestamptz	Дата і час оновлення

Таблиця 2.3

Опис таблиці client_profiles

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор профілю
user_id	integer (FK)	Посилання на користувача
birthdate	date	Дата народження
preferred_barber_id	integer (FK)	Улюблений барбер
notes	text	Нотатки
updated_at	timestamptz	Дата і час оновлення

Таблиця 2.4

Опис таблиці barbershops

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор філії
name	varchar(200)	Назва барбершопу
address	text	Адреса
city	varchar(100)	Місто
phone	varchar(20)	Номер телефону
description	text	Опис
photo_url	text	Посилання на фото
latitude	numeric	Географічна широта
longitude	numeric	Географічна довгота
is_active	boolean	Ознака активності
created_at	timestamptz	Дата і час створення
updated_at	timestamptz	Дата і час оновлення

Таблиця 2.5

Опис таблиці barbers

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор барбера
user_id	integer (FK)	Посилання на обліковий запис

продовж. табл. 2.5

barbershop_id	integer (FK)	Філія, до якої належить барбер
bio	text	Біографія
experience_years	smallint	Стаж роботи, років
instagram_url	text	Посилання на Instagram
is_active	boolean	Ознака активності
created_at	timestamptz	Дата і час створення
updated_at	timestamptz	Дата і час оновлення

Таблиця 2.6

Опис таблиці barber_services

Назва поля	Тип	Опис
barber_id	integer (PK, FK)	Ідентифікатор барбера
service_id	integer (PK, FK)	Ідентифікатор послуги

Таблиця 2.7

Опис таблиці services

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор послуги
category_id	integer (FK)	Категорія послуги
name	varchar(200)	Назва послуги
description	text	Опис
duration	smallint	Тривалість, хвилин
price	numeric	Ціна
photo_url	text	Посилання на фото
is_active	boolean	Ознака активності
is_extra	boolean	Ознака додаткової послуги
created_at	timestamptz	Дата і час створення

Таблиця 2.8

Опис таблиці appointments

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор запису
client_id	integer (FK)	Клієнт
barber_id	integer (FK)	Барбер
service_id	integer (FK)	Послуга
barbershop_id	integer (FK)	Філія
start_time	timestamptz	Час початку сеансу
end_time	timestamptz	Час завершення сеансу
status	appointment_status	Статус запису
notes	text	Нотатки
price_paid	numeric	Вартість наданої послуги
cancelled_at	timestamptz	Час скасування
cancel_reason	text	Причина скасування

продовж. табл. 2.8

created_at	timestampz	Дата і час створення
updated_at	timestampz	Дата і час оновлення

Таблиця 2.9

Опис таблиці reviews

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор відгуку
client_id	integer (FK)	Клієнт-автор відгуку
barber_id	integer (FK)	Барбер, якому адресовано відгук
appointment_id	integer (FK)	Пов'язаний запис
rating	smallint	Числова оцінка
comment	text	Текст відгуку
is_visible	boolean	Ознака видимості
created_at	timestampz	Дата і час створення

Таблиця 2.10

Опис таблиці forum_posts

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор публікації
author_id	integer (FK)	Автор публікації
category_id	integer (FK)	Категорія форуму
title	varchar(300)	Заголовок
body	text	Текст публікації
photo_url	text	Посилання на фото
is_pinned	boolean	Ознака закріплення
is_locked	boolean	Ознака блокування
views_count	integer	Кількість переглядів
created_at	timestampz	Дата і час створення
updated_at	timestampz	Дата і час оновлення

Таблиця 2.11

Опис таблиці forum_comments

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор коментаря
post_id	integer (FK)	Публікація, до якої залишено коментар
author_id	integer (FK)	Автор коментаря

продовж. табл. 2.11

parent_id	integer (FK)	Батьківський коментар для відповідей
body	text	Текст коментаря
is_deleted	boolean	Ознака видалення
is_edited	boolean	Ознака редагування
edited_at	timestamptz	Час редагування
created_at	timestamptz	Дата і час створення
updated_at	timestamptz	Дата і час оновлення

Таблиця 2.12

Опис таблиці notifications

Назва поля	Тип	Опис
id	integer (PK)	Ідентифікатор сповіщення
user_id	integer (FK)	Отримувач сповіщення
type	notification_type	Тип сповіщення
title	varchar(255)	Заголовок
body	text	Текст сповіщення
is_read	boolean	Ознака прочитання
link	text	Посилання на пов'язаний об'єкт
created_at	timestamptz	Дата і час створення

Решта таблиць, які не наведені окремо (refresh_tokens, barbershop_hours, barber_schedules, barber_absences, barber_portfolio, barber_rating_cache, service_categories, forum_categories, forum_post_photos, forum_comment_history, forum_likes), мають аналогічну структуру й виконують допоміжну роль, зокрема зберігають графіки роботи, періоди відсутності, портфоліо, кеш рейтингів та елементи форуму.

Більшість таблиць пов'язані з таблицею users, оскільки практично кожен запис у системі створюється конкретним користувачем, що реалізовано через поля user_id, author_id або client_id. Таблиця barber_services реалізує зв'язок «багато до багатьох» між барберами та послугами за допомогою складеного первинного ключа. Центральна таблиця appointments об'єднує клієнта, барбера, послугу й філію та зберігає відомості про кожне бронювання, а таблиця reviews додатково посиляється на конкретний запис, підтверджуючи

2.3. Проектування та реалізація алгоритмів роботи системи

Найскладнішим сценарієм системи є запис клієнта на послугу, послідовність кроків якого подано на рис. 2.5.

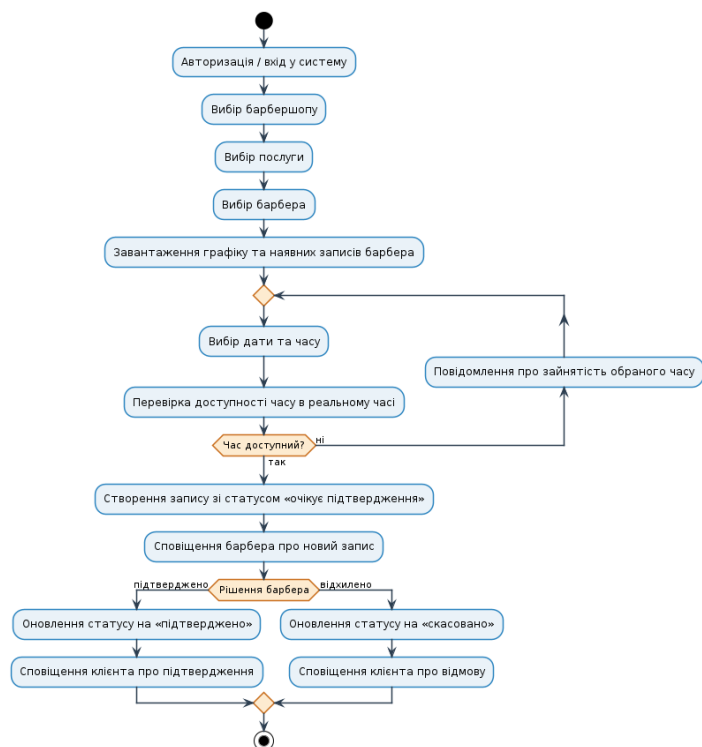


Рис. 2.5. Діаграма активності процесу онлайн-запису

Спершу користувач проходить авторизацію, а далі по черзі вказує барбершоп, потрібну послугу та майстра. Одразу після вибору барбера система звертається до його робочого графіка й уже наявних бронювань, щоб сформувати перелік вільних інтервалів. Коли клієнт обирає конкретну дату й час, виконується миттєва перевірка того, чи цей проміжок ще не зайнятий. У ситуації, коли обраний час уже заброньовано, користувач отримує відповідне повідомлення та повертається до вибору іншого проміжку, завдяки чому виключається повторне бронювання того самого слоту. Якщо ж час вільний, у системі фіксується новий запис зі статусом «очікує підтвердження», а майстер одночасно отримує сповіщення про нього. Подальша доля запису залежить від реакції барбера: після підтвердження статус змінюється на «підтверджено», у разі відмови – на «скасовано», і в обох випадках клієнта повідомляють про прийняте рішення.

Коректність процесу запису забезпечується кількома допоміжними механізмами. Робочі години майстра на обрану дату обчислюються за рівнями пріоритету: спершу враховується період його відсутності, далі особистий тижневий графік, потім графік роботи філії, а за відсутності цих даних – типові години за замовчуванням; на основі отриманого діапазону та зайнятих інтервалів формується набір вільних слотів. Під час створення запису тривалість сеансу визначається як сума основної та додаткових послуг, після чого система перевіряє на рівні бази даних, чи новий інтервал не перетинається з наявними записами майстра. Використання засобів СКБД для виявлення перетину часових проміжків гарантує коректність навіть за майже одночасних звернень кількох користувачів, унеможливлюючи подвійне бронювання [4]. Життєвий цикл запису описується чітко визначеним переліком статусів, між якими можливі лише дозволені переходи.

Не менш важливим є механізм входу в систему та розмежування повноважень. Користувач передає на сервер свою електронну пошту й пароль, після чого введений пароль звіряється зі збереженим хешем, а не з відкритим значенням. За умови успішної перевірки система видає захищений токен, у

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						39

якому зашифровано роль користувача, і саме за цим токеном надалі визначається, чи має запит право на доступ до тієї чи іншої операції. Така схема гарантує, що клієнт, барбер, адміністратор і менеджер працюють лише з призначеним для них функціоналом, а будь-яка спроба вийти за межі своєї ролі блокується ще до виконання дії.

Перевірка прав доступу побудована у вигляді ланцюжка проміжних обробників: перший підтверджує дійсність токена та додає дані користувача до запиту, а другий звіряє його роль із переліком дозволених для конкретної операції [28]. Окремо реалізовано обмеження для менеджера, за якого вибірки автоматично звужуються до його філії. Завдяки цьому одні й ті самі програмні інтерфейси повертають різний обсяг даних залежно від ролі: клієнт бачить лише власні записи, барбер – звернення до себе, менеджер – дані своєї філії, а адміністратор – інформацію по всій мережі.

Окремо побудовано логіку роботи з відгуками та підтримання рейтингу майстрів в актуальному стані. Завершивши відвідування, клієнт може оцінити роботу барбера й залишити коментар, який зберігається із прив'язкою до відповідного запису. Перед збереженням система перевіряє, що цей запис справді належить клієнту й має статус «виконано», а повторне оцінювання того самого запису блокується, що запобігає накрученню рейтингу. У момент додавання нового відгуку середній бал майстра та кількість оцінок переобчислюються й записуються до спеціальної таблиці-кешу, завдяки чому рейтинг відображається миттєво, без повторного перебирання всіх відгуків. Якщо адміністратор приховує певний відгук, той автоматично перестає враховуватися під час обчислення підсумкової оцінки.

Окремий блок алгоритмів забезпечує роботу форуму. Коментарі підтримують деревоподібну структуру завдяки посиланню кожного з них на батьківський, що дозволяє вибудовувати гілки відповідей, а під час редагування попередній текст коментаря зберігається в окремій таблиці історії. Видалення виконується як позначення коментаря прихованим без фізичного стирання, щоб не порушувати цілісність обговорення, а додавання

коментаря чи відповіді супроводжується сповіщенням для автора публікації та автора батьківського коментаря.

Завершує перелік ключових алгоритмів механізм сповіщень, що відповідає за своєчасне інформування користувачів. Частина повідомлень створюється безпосередньо в момент настання події – зміни статусу запису, появи нового бронювання чи відповіді на форумі, для чого передбачено єдину допоміжну функцію, що викликається з відповідних обробників. Нагадування про майбутній візит формує фонове завдання, яке періодично перевіряє наближення запланованих сеансів і створює нагадування лише для тих записів, яким його ще не надсилали, що унеможливорює дублювання. Кожне сповіщення містить ознаку прочитання та посилання на пов'язаний об'єкт, що дає користувачеві змогу одразу перейти до потрібного запису або обговорення.

Отже, розроблені алгоритми описують усі основні процеси платформи: авторизацію з розмежуванням доступу, онлайн-запис із перевіркою доступності, опрацювання відгуків, роботу форуму та розсилання сповіщень і забезпечують стабільну та передбачувану поведінку системи для кожної категорії користувачів, що є основою для її подальшої програмної реалізації та супроводу.

2.4. Реалізація вебплатформи мережі барбершопів

Вебплатформу реалізовано за клієнт-серверною архітектурою, що передбачає чіткий поділ системи на дві самостійні частини, які взаємодіють між собою через визначений програмний інтерфейс. Серверна частина відповідає за зберігання та обробку даних, перевірку прав доступу й виконання бізнес-логіки, тоді як клієнтська частина забезпечує відображення інформації та взаємодію з користувачем. Обмін даними між ними побудовано за принципами REST поверх протоколу HTTP, а як формат передавання використано JSON, що є компактним, зручним для обробки та підтримується обома частинами системи [18]. Такий поділ дозволяє розробляти, тестувати й розвивати кожну частину незалежно та спрощує подальший супровід

програмного забезпечення.

Серверну частину побудовано на платформі Node.js із застосуванням фреймворку Express, який бере на себе маршрутизацію запитів та обробку HTTP-звернень [3]. Для впорядкування коду застосовано підхід із розділенням відповідальностей, за якого кожен функціональний шар виконує власну роль [16]. Шар маршрутів лише визначає адреси кінцевих точок і прив'язує до них відповідні обробники, шар контролерів зосереджує в собі бізнес-логіку та роботу з базою даних, а проміжні обробники виконують допоміжні дії, такі як перевірка автентифікації чи прав доступу, ще до того, як запит досягне контролера. Завдяки цьому структура проєкту залишається прозорою, а окремі модулі, що відповідають за користувачів, записи, барберів, послуги чи форум, можна змінювати незалежно один від одного.

Запуск і налаштування серверного застосунку зосереджено в головному файлі, де створюється екземпляр Express та послідовно під'єднуються необхідні проміжні обробники. Передусім підключається механізм підтримки міждоменних запитів, який дозволяє клієнтській частині, розгорнутій на іншому джерелі, безперешкодно звертатися до сервера. Далі активується обробник, що автоматично розбирає тіло запитів у форматі JSON та робить отримані дані доступними в зручному вигляді, а також налаштовується віддача статичних файлів, завдяки чому завантажені зображення стають доступними за прямим посиланням. Після цього кожна група маршрутів прив'язується до власного префікса адреси, що дозволяє логічно згрупувати споріднені операції, наприклад усі дії із записами зосереджено в межах одного шляху. Окремо передбачено обробники помилок, які повертають зрозумілі повідомлення у разі звернення до неіснуючої адреси або виникнення внутрішньої помилки сервера. Під час старту застосунок не лише починає очікувати з'єднання на заданому порту, а й запускає фонове завдання, що періодично формує нагадування користувачам про заплановані візити.

Взаємодію з базою даних організовано через окремий модуль, у якому налаштовується пул з'єднань із PostgreSQL. Застосування пулу дозволяє

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						42

повторно використовувати вже відкриті підключення замість встановлення нового для кожного окремого запиту, що суттєво підвищує продуктивність за умови одночасної роботи багатьох користувачів [4]. Параметри підключення, зокрема адреса сервера, назва бази, ім'я користувача та пароль, зчитуються зі змінних оточення, а не зберігаються безпосередньо в коді, що відповідає вимогам безпеки та спрощує перенесення системи між середовищами. Усі звернення до бази виконуються через єдину допоміжну функцію, причому значення параметрів завжди передаються окремо від тексту запиту. Такий спосіб формування запитів унеможлиблює впровадження стороннього коду через введені користувачем дані та забезпечує захист від атак типу SQL-ін'єкція.

Особливу увагу під час реалізації приділено підсистемі автентифікації та керування доступом. Під час реєстрації нового користувача система спершу перевіряє, чи не зайнята вказана електронна пошта, після чого пароль перетворюється на криптографічний хеш за допомогою алгоритму bcrypt із дванадцятьма раундами обчислення. Відкритий пароль ніде не зберігається, а під час наступних входів введені значення не розшифровується, а лише порівнюється з раніше збереженим хешем, що унеможлиблює відновлення пароля навіть у разі витоку бази даних. Одразу після створення облікового запису для користувача автоматично формується профіль клієнта. Для подальшої роботи застосовано механізм автентифікації на основі JSON Web Token, за якого користувачеві видаються два токени [28]. Короткоживучий токен доступу супроводжує звичайні запити та підтверджує особу користувача, а довготривалий токен оновлення дозволяє продовжити сеанс без повторного введення облікових даних. Токени оновлення зберігаються в окремій таблиці бази даних разом із терміном дії, що дає змогу контролювати активні сесії та за потреби відкликати їх.

Розмежування прав доступу реалізовано за допомогою двох проміжних обробників, які працюють у зв'язці. Перший із них перевіряє наявність та дійсність токена доступу: він зчитує токен із відповідного заголовка запиту,

перевіряє його підпис і термін дії, а в разі успіху додає розшифровані відомості про користувача до об'єкта запиту, роблячи їх доступними для подальшої обробки. Якщо токен відсутній, прострочений або підроблений, запит відхиляється ще до того, як досягне контролера. Другий обробник відповідає за перевірку ролі та приймає перелік ролей, яким дозволено виконати певну операцію. Запит пропускається далі лише тоді, коли роль поточного користувача входить до цього переліку, інакше повертається відмова через брак повноважень. Таке поєднання дозволяє коротко й однозначно описувати рівень захисту кожного маршруту, надаючи, наприклад, доступ до адміністративних дій лише адміністраторам і менеджерам. Додатково для ролі менеджера передбачено механізм обмеження видимості даних межами закріпленої за ним філії, завдяки чому менеджер працює лише з інформацією свого барбершопу, тоді як адміністратор має доступ до всієї мережі.

Центральне місце в бізнес-логіці посідає процес створення запису клієнта, у якому реалізовано перевірку доступності обраного часу. На основі моменту початку та сумарної тривалості обраних послуг система обчислює час завершення сеансу, після чого звертається до бази даних, щоб з'ясувати, чи немає в цього барбера інших записів, інтервал яких перетинається з бажаним. Для виявлення такого накладання використано вбудовані засоби PostgreSQL для роботи з часовими проміжками. Якщо виявлено хоча б один конфлікт, операція переривається з повідомленням про зайнятість часу, що повністю виключає можливість подвійного бронювання одного інтервалу різними клієнтами. За відсутності перетинів новий запис додається до бази зі статусом очікування підтвердження, а відповідний барбер одразу отримує сповіщення про появу нового бронювання. Подальша зміна статусу запису контролюється з урахуванням ролі: клієнтові дозволено лише скасувати власний запис, тоді як підтвердження, завершення чи позначення неявки доступні працівникам.

Окрім ключового сценарію запису, серверна частина містить низку інших контролерів, що реалізують повний набір можливостей платформи. Контролери послуг, барберів і барбершопів забезпечують додавання,

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						44

редагування та видалення відповідних записів, причому для довідкових даних передбачено захист від видалення за наявності пов'язаних записів. Підсистема відгуків не лише зберігає оцінки клієнтів, а й під час кожного нового відгуку автоматично перераховує середній бал барбера та кількість оцінок, записуючи оновлені значення до спеціальної допоміжної таблиці. Завдяки цьому рейтинг відображається миттєво, без потреби щоразу обробляти всі відгуки. Модуль форуму реалізує створення тем, написання коментарів із підтримкою деревоподібних відповідей та оцінювання дописів, а також можливості модерації. Підсистема сповіщень формує повідомлення як безпосередньо під час зміни статусу записів, так і за допомогою фонових завдань, що завчасно нагадує про майбутні візити. Окремий аналітичний модуль готує зведені дані про популярність послуг, відвідуваність і доходи, які використовуються менеджером для оцінювання роботи мережі.

Завантаження зображень, зокрема аватарів користувачів та робіт для портфолію барберів, реалізовано за допомогою спеціалізованого проміжного обробника, що опрацьовує дані форм, призначені для передавання файлів [12]. У його налаштуваннях задано папку для збереження та правило формування імені файлу, за яким до поточної позначки часу додається випадкове число, що виключає збіги назв при одночасному завантаженні. Додатково встановлено фільтр, який пропускає лише файли зображень припустимих форматів, та обмеження на максимальний розмір файлу, що захищає сервер від завантаження надмірно великих або сторонніх даних.

Клієнтську частину реалізовано як односторінковий застосунок на основі бібліотеки React, що дозволяє оновлювати вміст сторінки без її повного перезавантаження та забезпечує плавну взаємодію з інтерфейсом [1]. Інтерфейс побудовано з окремих компонентів, кожен з яких відповідає за певну логічну частину, наприклад сторінки барбершопів, профілю барбера, оформлення запису, форуму чи адміністративних розділів. Навігацію між сторінками організовано засобами клієнтської маршрутизації, завдяки чому переходи відбуваються миттєво в межах того самого застосунку. Усі

звернення до сервера здійснюються через єдиний попередньо налаштований HTTP-клієнт, для якого визначено перехоплювачі запитів і відповідей. Перехоплювач запитів автоматично додає токен доступу до кожного звернення, тож окремим компонентам не потрібно дбати про автентифікацію самостійно. Перехоплювач відповідей відстежує ситуацію, коли термін дії токена спливає, і непомітно для користувача звертається за новим токеном, після чого повторює початковий запит, а в разі невдачі перенаправляє користувача на сторінку входу.

Глобальний стан автентифікації на клієнті зберігається за допомогою механізму контексту React, що робить відомості про поточного користувача доступними будь-якому компоненту без необхідності передавати їх вручну через властивості [9]. Під час входу отримані токени зберігаються в локальному сховищі браузера, а повні дані користувача завантажуються із сервера й записуються у спільний стан, який згодом використовується для відображення інтерфейсу відповідно до ролі. На основі цього стану працює компонент захищеного маршруту, що не допускає неавторизованих відвідувачів до закритих сторінок і перенаправляє користувача залежно від наявних у нього повноважень. Такий підхід забезпечує узгодженість між серверною перевіркою прав і поведінкою інтерфейсу, оскільки користувач навіть візуально не отримує доступу до розділів, призначених для інших ролей.

Таким чином, у межах реалізації створено повноцінну серверну та клієнтську частини вебплатформи, які спільно охоплюють весь передбачений функціонал – від реєстрації та автентифікації з розмежуванням ролей до онлайн-запису з перевіркою доступності, ведення відгуків, форуму, сповіщень і аналітики. Застосовані технічні рішення забезпечують безпеку даних, стабільність роботи за умови одночасного доступу багатьох користувачів та зручність подальшого розширення системи, що повністю відповідає вимогам, сформульованим на етапі проектування.

Висновки до другого розділу

У другому розділі виконано проектування та програмну реалізацію вебплатформи мережі барбершопів. На етапі визначення варіантів використання окреслено чотири ролі користувачів – клієнта, барбера, адміністратора та менеджера – і для кожної з них побудовано діаграми варіантів використання, що чітко розмежують доступний функціонал. Сформована специфікація дала змогу встановити повний перелік дій, які має підтримувати система, та стала підґрунтям для подальшого проектування.

На основі визначених вимог розроблено структуру бази даних під керуванням PostgreSQL, що налічує двадцять дві таблиці, об'єднані в логічні модулі користувачів, барбершопів, барберів, послуг, записів, відгуків, форуму та сповіщень. Спроектовану модель подано у вигляді ER-діаграми та описано на рівні окремих таблиць із зазначенням полів, типів даних і ключів, а цілісність даних забезпечено застосуванням зовнішніх ключів, перелічуваних типів і продуманих правил видалення.

Окрему увагу приділено алгоритмам роботи системи, серед яких детально розглянуто процес онлайн-запису з перевіркою доступності часу, механізм авторизації з розмежуванням прав доступу, опрацювання відгуків із підтриманням актуального рейтингу та формування сповіщень. Динаміку ключового процесу подано за допомогою діаграми активності, що наочно відображає послідовність дій і умовні розгалуження під час бронювання.

Завершальним етапом стала програмна реалізація серверної та клієнтської частин платформи. Серверну частину побудовано на Node.js із фреймворком Express та організовано за принципом розділення відповідальностей між маршрутами, контролерами й проміжними обробниками, а автентифікацію реалізовано на основі технології JSON Web Token із хешуванням паролів. Клієнтську частину створено засобами бібліотеки React із використанням контексту для керування станом і HTTP-клієнта axios для взаємодії із сервером. Отримані результати підтверджують працездатність обраних проектних рішень і створюють основу для тестування

та оцінювання системи, що розглядається в наступному розділі.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ

3.1. Порядок встановлення та налаштування параметрів системи

Щоб система працювала коректно, її необхідно правильно встановити та налаштувати, розгорнувши кожен її компонент і задавши потрібні параметри. Для опису фізичної архітектури вебплатформи та наочного подання розподілу програмного забезпечення на апаратних ресурсах побудовано діаграму розгортання, наведену на рис. 3.1. Така діаграма відображає, як саме компоненти системи розміщуються на фізичних вузлах і яким чином вони взаємодіють між собою на апаратному рівні [27].

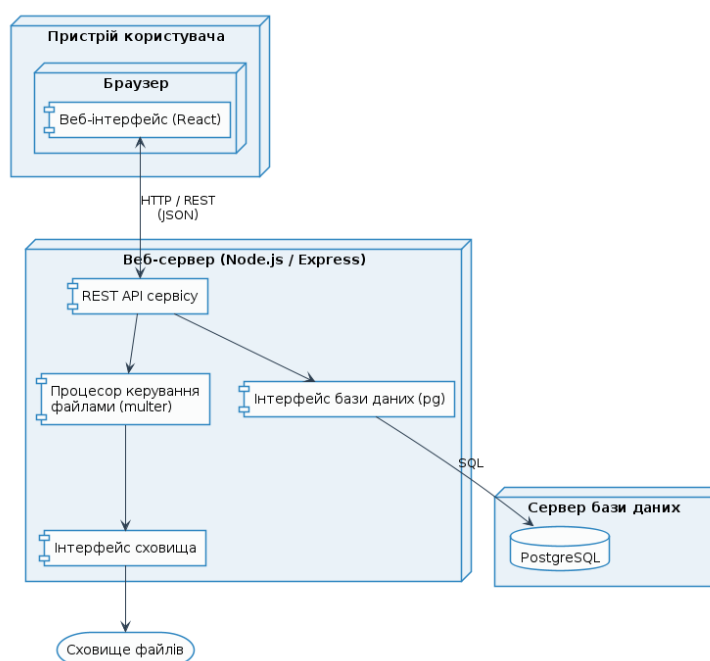


Рис. 3.1. Діаграма розгортання вебплатформи мережі барбершопів

Архітектура системи охоплює чотири основні вузли. На пристрої користувача працює браузер, у якому виконується клієнтський застосунок, реалізований засобами React, – саме через цей вебінтерфейс відвідувач взаємодіє із системою. Вебсервер, побудований на Node.js із фреймворком Express, є центральним вузлом, що приймає та опрацьовує запити користувачів. У його межах функціонує кілька логічних компонентів: REST API, який обробляє звернення клієнтської частини, процесор керування файлами на основі бібліотеки multer, інтерфейс бази даних для виконання

SQL-запитів та інтерфейс сховища для роботи із завантаженими файлами. Окремий вузол утворює сервер бази даних під керуванням PostgreSQL, де зберігаються всі дані системи, а ще один вузол – файлове сховище, у якому розміщуються завантажені зображення, зокрема аватари та роботи портфолію.

Взаємодія між вузлами відбувається за чітко визначеною схемою. Браузер обмінюється даними з вебсервером через протокол HTTP за принципами REST, передаючи запити та отримуючи відповіді у форматі JSON. Вебсервер, своєю чергою, звертається до сервера бази даних через інтерфейс бази даних, виконуючи SQL-запити, а для роботи із зображеннями використовує процесор керування файлами, що через інтерфейс сховища забезпечує доступ до файлового сховища. Такий розподіл функцій між вузлами дозволяє зрозуміти, як основні можливості системи відокремлені одна від одної та як компоненти доповнюють один одного під час обробки запиту.

Слід зазначити, що подане розмежування є логічним, тому залежно від умов експлуатації вузли можуть розташовуватися як на одній фізичній машині, так і на окремих серверах. На етапі розробки та для невеликих мереж усі компоненти зручно розгортати на одному комп'ютері, тоді як у разі зростання навантаження сервер бази даних доцільно винести на окрему машину, а кількість екземплярів вебсервера – збільшити. Завдяки тому, що серверна частина не зберігає стан сесій, такий розподіл не потребує суттєвих змін у коді й забезпечує можливість поступового масштабування системи.

Для розгортання системи на сервері або робочій станції необхідно попередньо встановити середовище виконання Node.js, яке забезпечує роботу серверної частини, та систему керування базами даних PostgreSQL для зберігання даних [3, 4]. Разом із Node.js встановлюється менеджер пакетів, що використовується для завантаження залежностей, а для отримання вихідного коду застосовується система контролю версій, за допомогою якої проєкт копіюється з віддаленого репозиторію [25]. Окрім цього, на пристрої користувача має бути встановлений сучасний вебпереглядач, через який здійснюється доступ до платформи.

Першим кроком безпосереднього налаштування є підготовка бази даних. У PostgreSQL створюється нова база, після чого застосовується схема, яка містить описані в попередньому розділі перелічувані типи та таблиці разом із зовнішніми ключами й правилами цілісності. Для виконання цих дій можна скористатися як консольним клієнтом, так і графічним засобом адміністрування, що спрощує початкове розгортання та подальше обслуговування бази. Після створення структури база готова до приймання даних, які накопичуватимуться під час роботи системи.

Після підготовки бази даних у файлі конфігурації серверної частини задаються параметри підключення, а саме адреса хоста, порт, назва бази, ім'я користувача та пароль, які зчитуються через змінні оточення й не зберігаються безпосередньо в коді. Окрім параметрів бази даних, у конфігурації серверної частини вказуються секретні ключі для формування токенів доступу та оновлення разом із термінами їхньої дії, порт, на якому очікує запити сервер, а також адреса клієнтської частини, що використовується для налаштування політики доступу з інших джерел. Винесення цих значень у змінні оточення є важливим з погляду безпеки, оскільки конфіденційні дані не потрапляють до вихідного коду й до репозиторію, а секретні ключі мають бути достатньо складними, щоб унеможливити підроблення токенів [28].

Клієнтська частина налаштовується окремо й спрямовується на адресу серверного API, після чого для серверної та клієнтської частин встановлюються залежності за допомогою менеджера пакетів. Папка для збереження завантажених зображень створюється автоматично під час запуску сервера, тому додаткового ручного налаштування сховища файлів не потребує. Завдяки організації проєкту з окремими складовими для сервера й клієнта встановлення та оновлення залежностей виконується незалежно для кожної частини, що спрощує супровід.

Після задання всіх параметрів система запускається: серверний застосунок починає очікувати з'єднання на визначеному порту й одночасно активує фонове завдання формування нагадувань, а клієнтський застосунок

стає доступним у браузері. Працездатність серверної частини можна перевірити через службовий запит стану, що повертає підтвердження її роботи, після чого платформа відкривається у вебпереглядачі. Оскільки під час реєстрації новий користувач за замовчуванням отримує роль клієнта, для початкового адміністрування системи відповідному обліковому запису потрібно вручну призначити роль адміністратора в базі даних, після чого через адміністративну панель створюються філії, послуги та облікові записи барберів і менеджерів.

Для використання системи в реальних умовах експлуатації клієнтську частину збирають у статичний набір файлів, який віддається користувачам, а серверний застосунок переводиться у виробничий режим роботи [12]. У такому разі доцільно налаштувати захищене з'єднання за протоколом HTTPS та зворотний проксі-сервер, що спрямовує запити до вебсервера, а також передбачити регулярне резервне копіювання бази даних і ведення журналів роботи. Дотримання цих рекомендацій забезпечує безпеку, надійність і відмовостійкість системи під час її промислової експлуатації.

Таким чином, процес встановлення та налаштування охоплює підготовку середовища виконання й бази даних, задання параметрів підключення та безпеки, встановлення залежностей і запуск обох частин системи. За умови коректного виконання всіх кроків платформа готова до роботи, а користувач може отримати доступ до неї через вебінтерфейс.

3.2. Структура інтерфейсу

У цьому підрозділі детально розглянуто, який вигляд має інтерфейс вебплатформи, з яких елементів він складається та як користувач взаємодіє із системою. Доступ до перегляду основного вмісту сайту відкритий для будь-якого відвідувача, а повний набір можливостей розблоковується після авторизації. Незмінним елементом усіх сторінок є верхня панель навігації, що містить логотип, посилання на основні розділи – барбершопи, барберів і форум, а для авторизованого користувача додатково відображає посилання на його записи, особистий кабінет, піктограму сповіщень та кнопку виходу.

Неавторизованому відвідувачу натомість пропонуються кнопки входу та реєстрації.

Початковою точкою взаємодії є головна сторінка, що відкривається одразу при заході на сайт (рис. 3.2). На ній розміщено привітальний блок із коротким описом сервісу та закликом розпочати роботу, а нижче – секцію переваг платформи. Тут же подано перелік барбершопів мережі та кнопки переходу до запису, що дозволяють відвідувачу одразу перейти до бронювання.

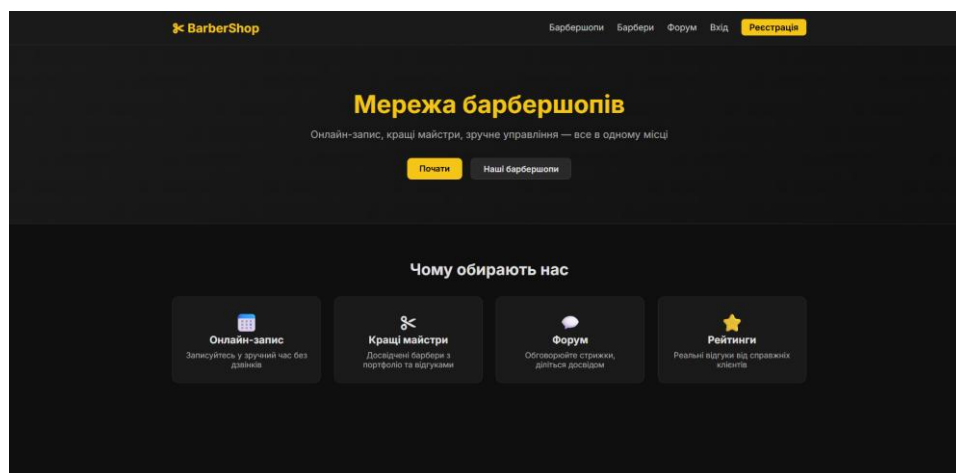


Рис. 3.2. Головна сторінка сайту

Щоб скористатися повним функціоналом, користувач має пройти авторизацію на сторінці входу (рис. 3.3). Ця сторінка містить поля для введення електронної пошти та пароля, кнопку входу й посилання для переходу до реєстрації.

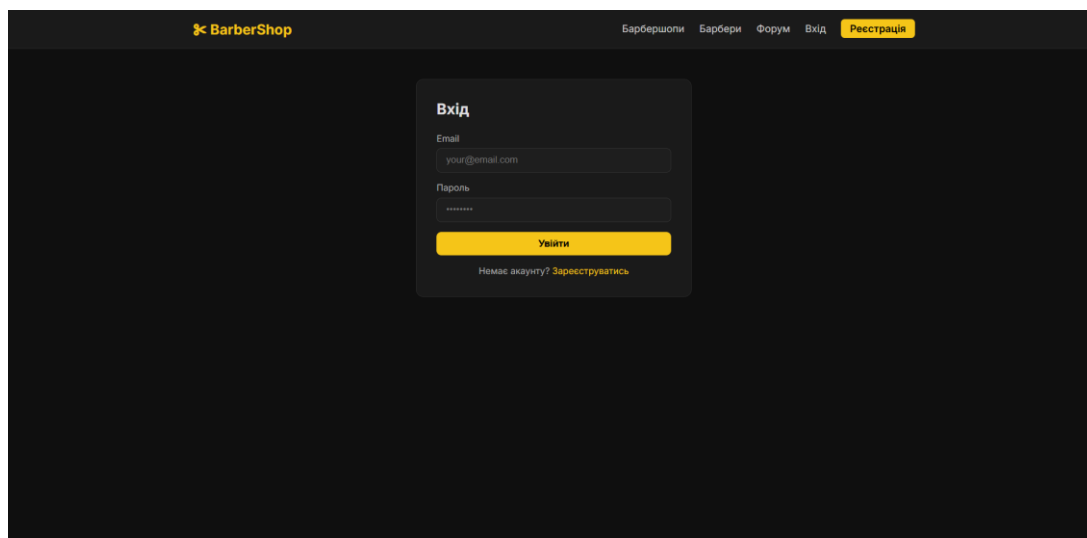


Рис. 3.3. Сторінка входу

Якщо облікового запису ще немає, відвідувач переходить на сторінку реєстрації (рис. 3.4), де заповнює поля з іменем, прізвищем, телефоном, електронною поштою та паролем із його повторним підтвердженням, після чого система перевіряє коректність даних і створює новий обліковий запис.

Рис. 3.4. Сторінка реєстрації

Перегляд закладів мережі здійснюється на сторінці барбершопів (рис. 3.5). На ній відображається перелік філій із їхніми назвами, адресами та фотографіями, а користувачеві пропонується обрати зручну філію для подальшого запису.

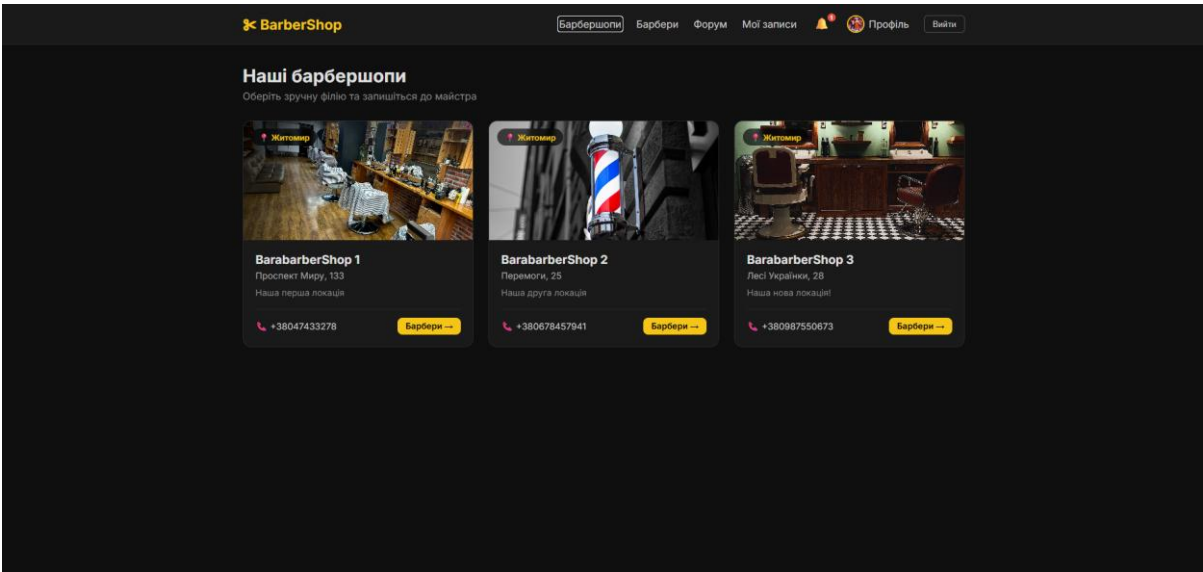


Рис. 3.5. Сторінка барбершопів

Перехід до переліку майстрів відбувається на сторінці барберів (рис.

3.6), де подано список усіх барберів мережі разом із короткою інформацією про них та їхнім рейтингом.

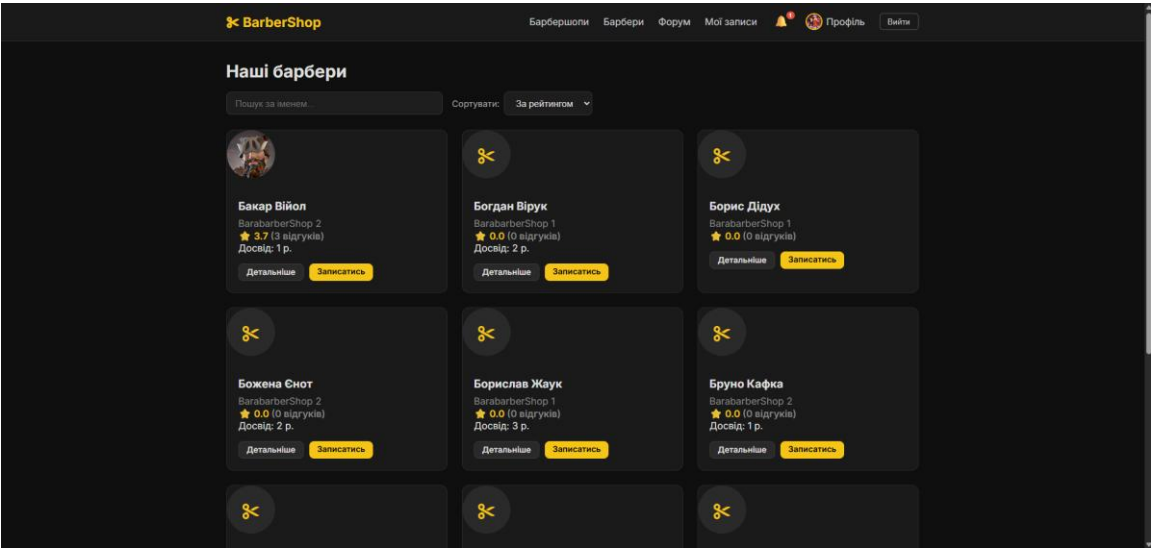


Рис. 3.6. Сторінка барберів

Детальні відомості про конкретного майстра наведено на сторінці барбера (рис. 3.7). Вона містить інформацію про досвід майстра, його графік роботи за днями тижня, перелік послуг, які він надає, портфолію з виконаними роботами та блок відгуків клієнтів із можливістю залишити власну оцінку й коментар. Звідси ж користувач може одразу перейти до запису, натиснувши відповідну кнопку.

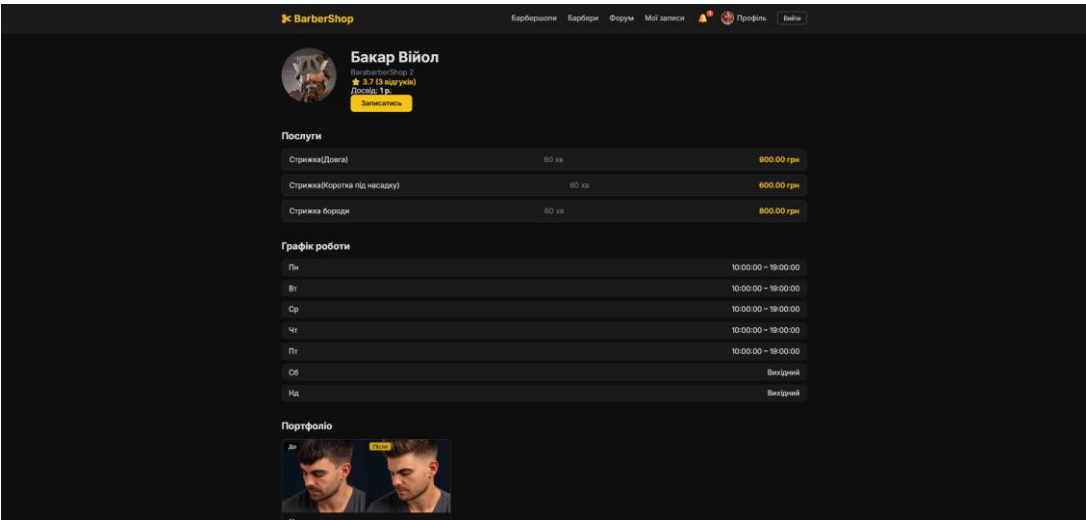


Рис. 3.7. Сторінка майстра з портфолію та відгуками

Ключовим процесом є онлайн-запис, що виконується на сторінці бронювання (рис. 3.8). Користувач послідовно обирає основну послугу та за бажанням додаткові, після чого вказує дату й вільний час із доступних слотів,

а за потреби залишає коментар. Праворуч у режимі реального часу формується підсумок запису із зазначенням майстра, обраних послуг, дати, часу та підсумкової вартості, а після підтвердження система повідомляє про успішне створення запису.

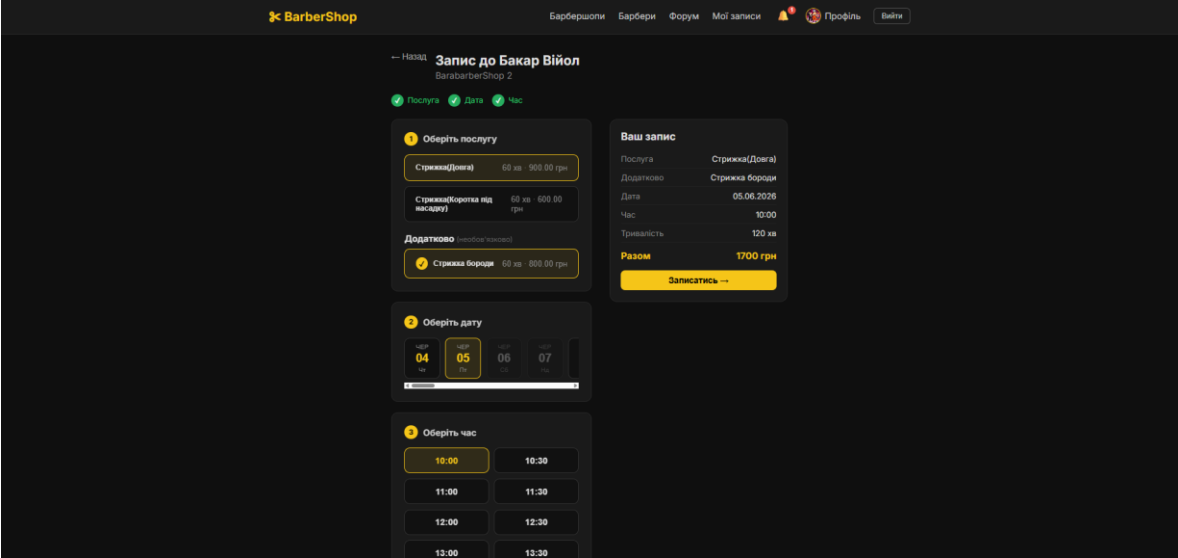


Рис. 3.8. Сторінка онлайн-запису

Усі створені бронювання користувач переглядає на сторінці своїх записів (рис. 3.9). Записи відображаються із зазначенням статусу, який може набувати значень «очікує», «підтверджено», «виконано», «скасовано» або «не прийшов». Активний запис клієнт може скасувати, а після завершення відвідування – залишити відгук із оцінкою безпосередньо з цієї сторінки.

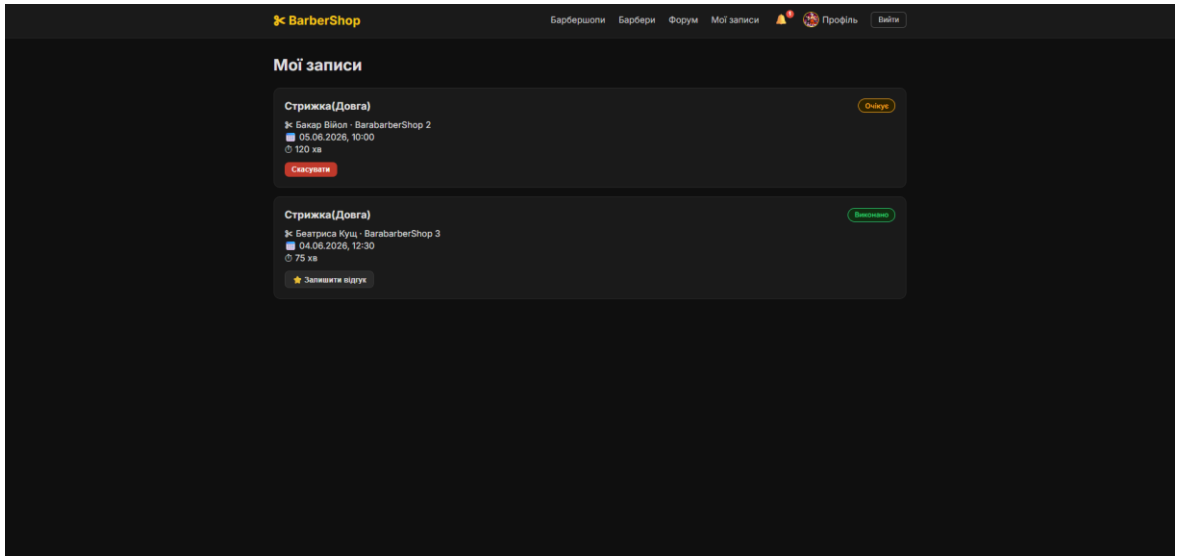


Рис. 3.9. Сторінка «Мої записи»

Керування власними даними здійснюється на сторінці профілю (рис.

3.10). Тут користувач переглядає та редагує свої персональні відомості й аватар, а також бачить позначку своєї ролі в системі. Для майстра ця сторінка розширюється додатковими блоками – особистим графіком роботи, переліком наданих послуг, портфоліо з можливістю додавати нові роботи, а також списком власних відгуків і дописів на форумі.

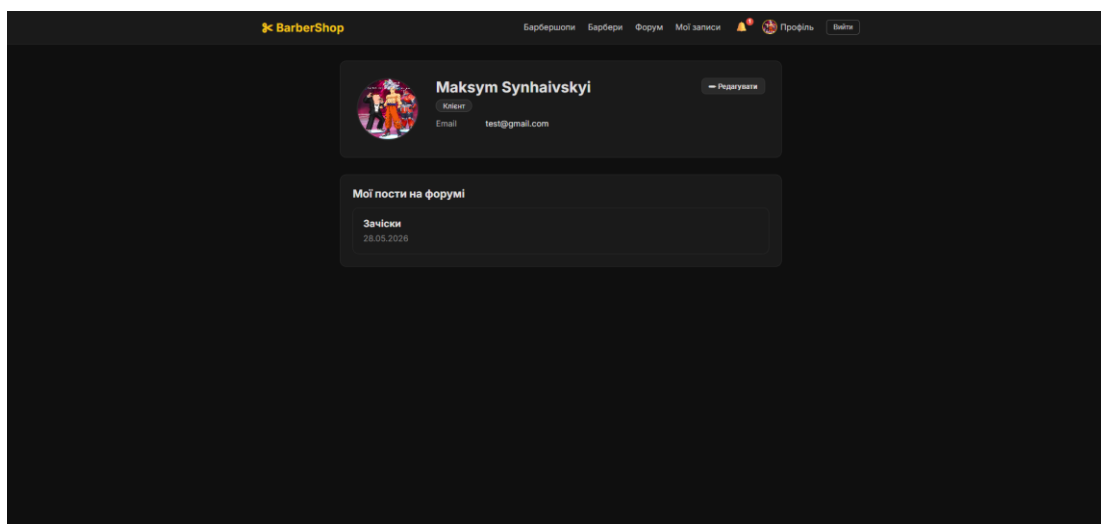


Рис. 3.10. Сторінка профілю користувача

Спілкування користувачів відбувається на сторінці форуму (рис. 3.11). На ній подано перелік публікацій, причому закріплені теми відображаються вище за інші, а авторизований користувач може створити новий допис, указавши заголовок, текст і за бажанням категорію.

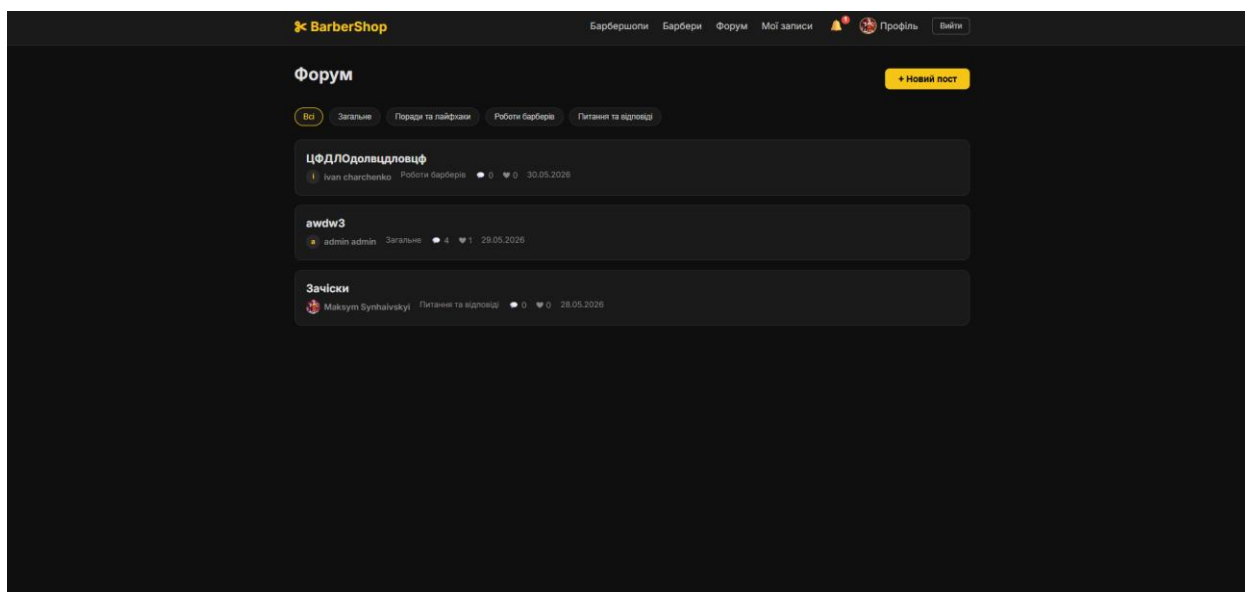


Рис. 3.11. Сторінка форуму

Перехід до окремої публікації відкриває сторінку поста (рис. 3.12), де

відображається його повний текст і гілка коментарів із можливістю відповідати, редагувати власні коментарі з переглядом історії змін та видаляти їх.

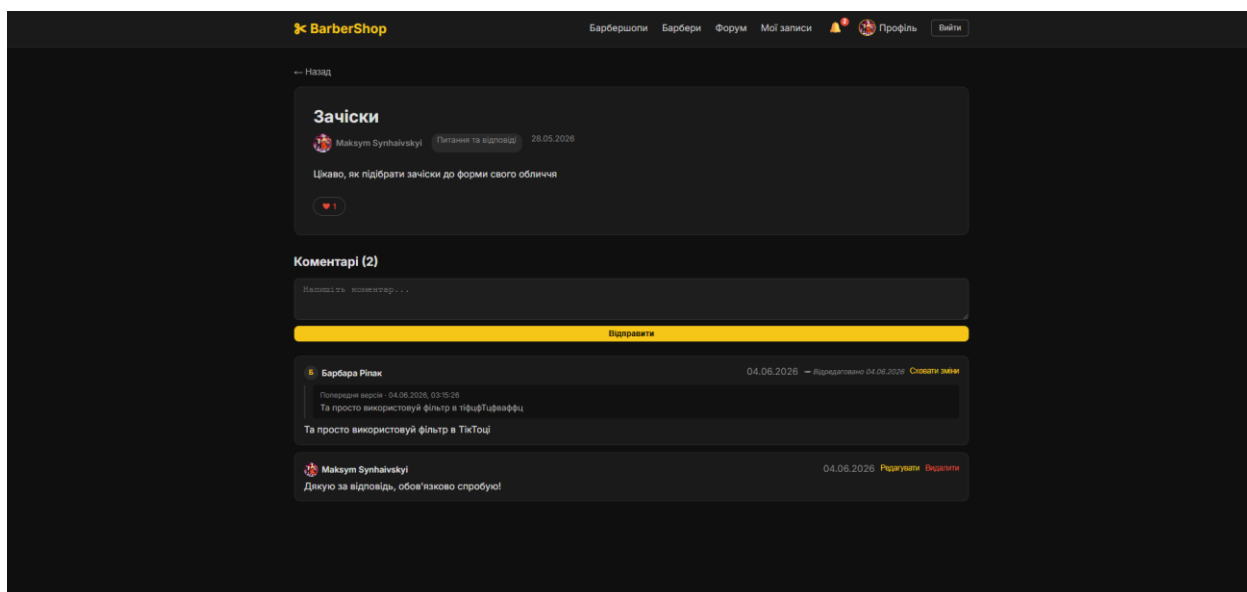


Рис. 3.12. Сторінка публікації форуму

Своєчасне інформування користувача забезпечує панель сповіщень, що розгортається з піктограми у верхній панелі навігації (рис. 3.13). У ній відображаються повідомлення про підтвердження чи зміну записів і нагадування про візити, а також передбачено можливість позначити всі сповіщення як прочитані.

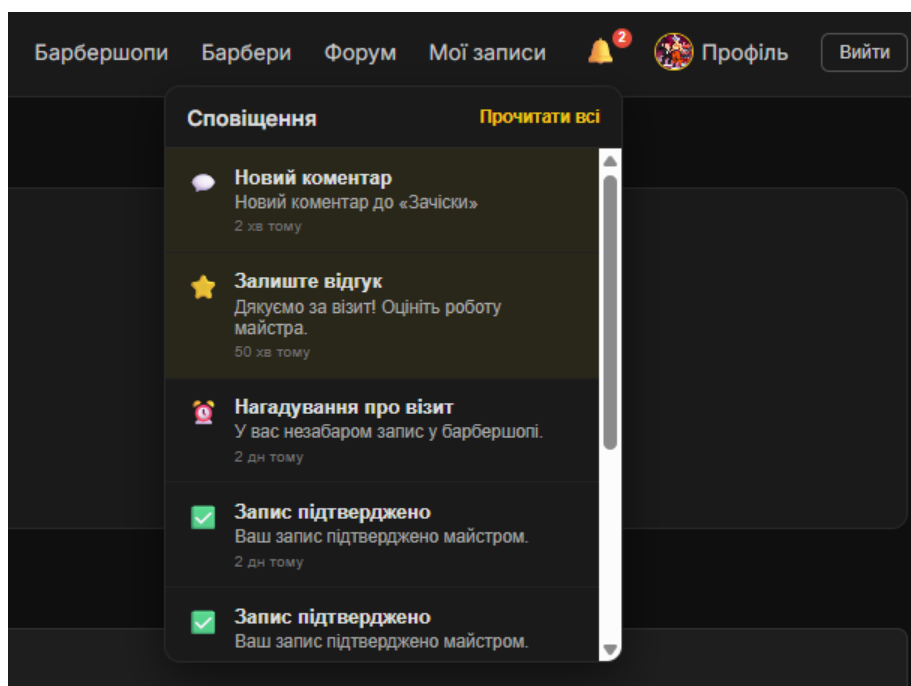


Рис. 3.13. Панель сповіщень

Робоче середовище майстра реалізовано у вигляді кабінету барбера (рис. 3.14). У ньому барбер переглядає свої записи, поділені на найближчі та минулі, керує їхніми статусами, зокрема підтверджує або скасовує бронювання,

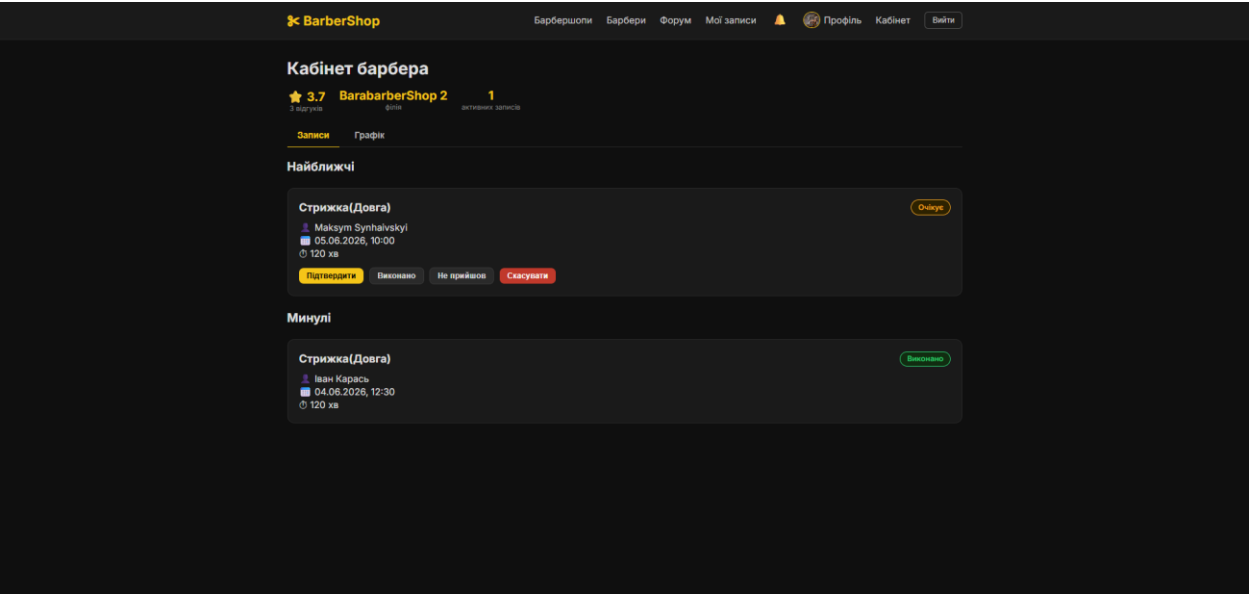


Рис. 3.14. Кабінет барбера

А також барбер може працювати з власним графіком роботи (рис. 3.15.). Це дозволяє майстрові контролювати завантаженість і підтримувати актуальність розкладу.

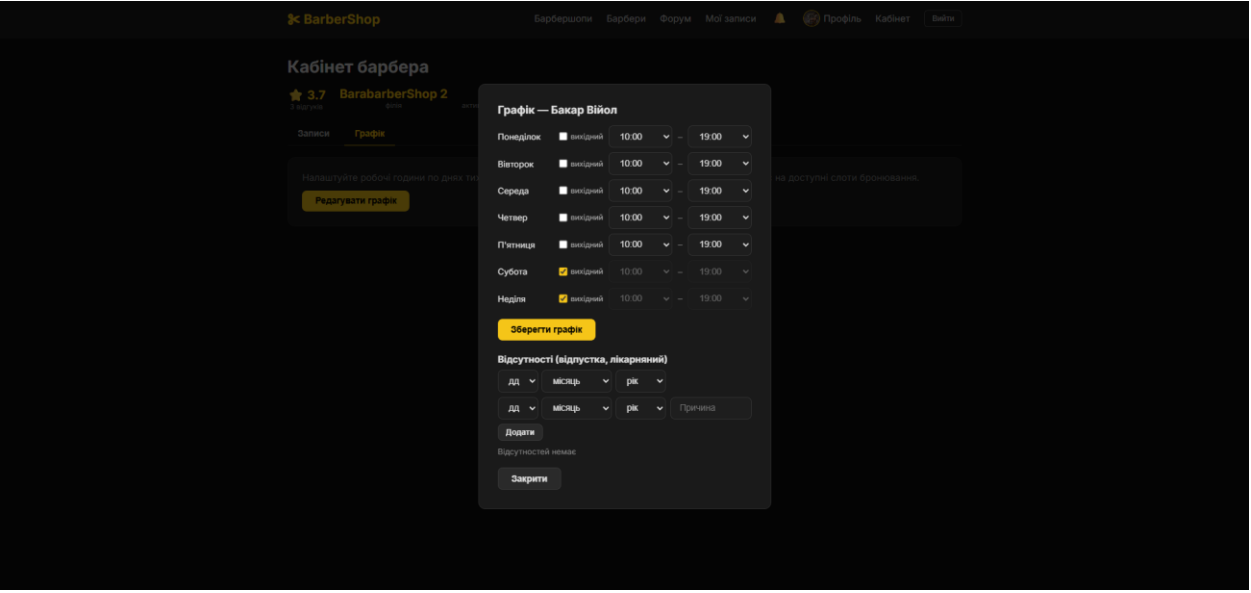


Рис. 3.15. Кабінет барбера(Редагування графіку)

Управління системою зосереджено в адміністративній панелі, доступній адміністраторові та менеджеру (рис. 3.16). Панель має бічне меню з розділами керування барбершопами, послугами, барберами, записами та аналітикою.

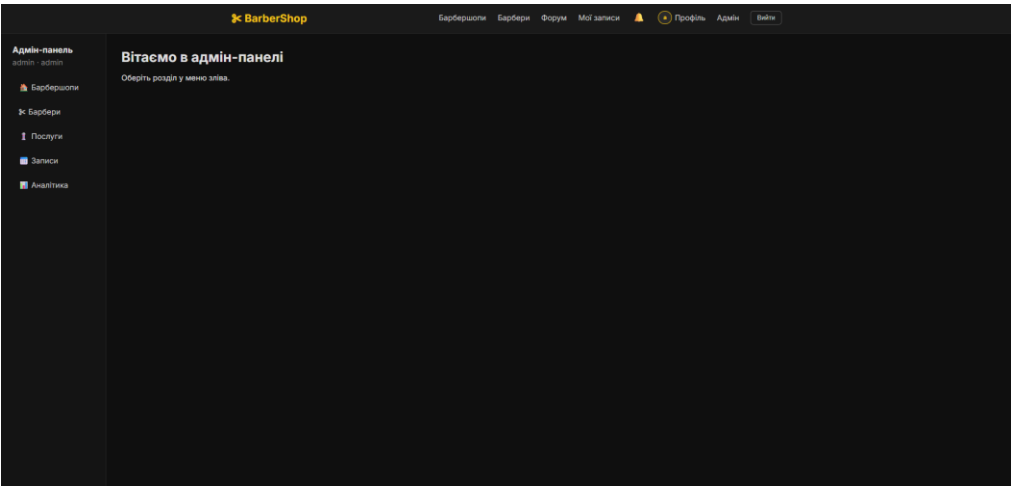


Рис. 3.16. Загальний вигляд адмін-панелі

У розділі барбершопів(рис. 3.17.) адміністратор додає, редагує та деактивує філії, зазначаючи їхню назву, адресу, місто й фотографію.

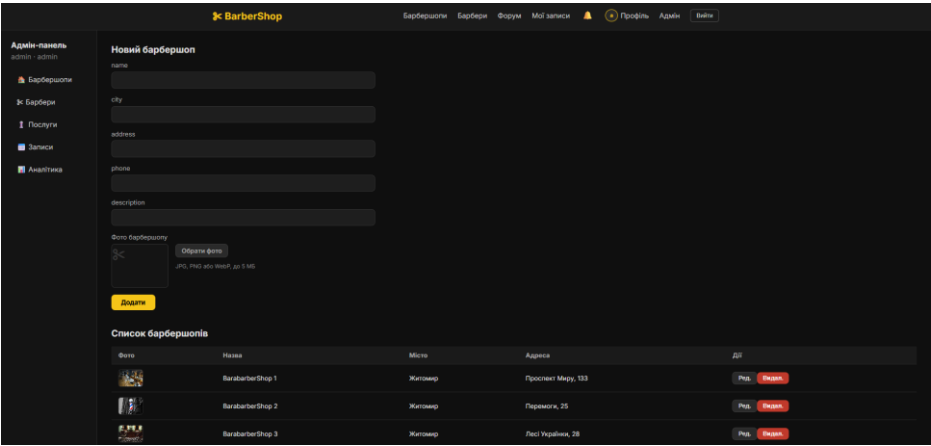


Рис. 3.17. Адмін-панель, барбершопи

Розділ послуг(рис. 3.18.) дозволяє керувати переліком послуг за категоріями, такими як стрижка, гоління, догляд за бородою, укладка та комплексні пакети, із зазначенням тривалості й ціни.

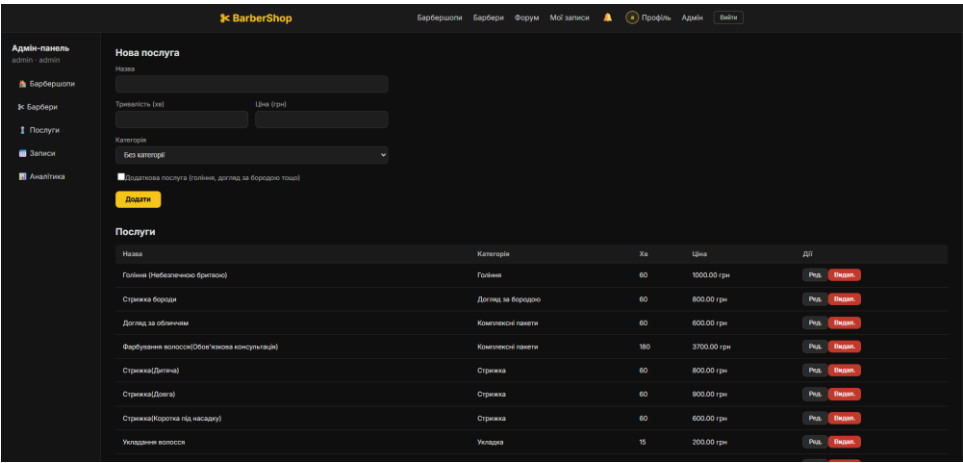


Рис. 3.18. Адмін-панель, послуги

У розділі барберів(рис. 3.19.) додаються нові майстри та налаштовуються надавані ними послуги, а розділ записів надає доступ до всіх бронювань системи з можливістю змінювати їхні статуси.

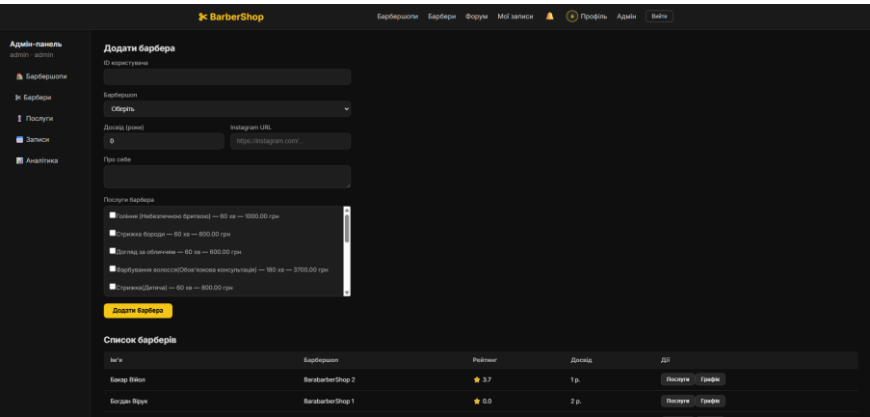


Рис. 3.19. Адмін-панель, барбери

Окремо варто виділити розділ аналітики, орієнтований передусім на менеджера мережі (рис. 3.20). У ньому за обраний період відображаються зведені показники діяльності – сумарний дохід, кількість замовлень і клієнтів, розподіл записів за статусами, динаміка доходу, дохід у розрізі філій та завантаженість за годинами.

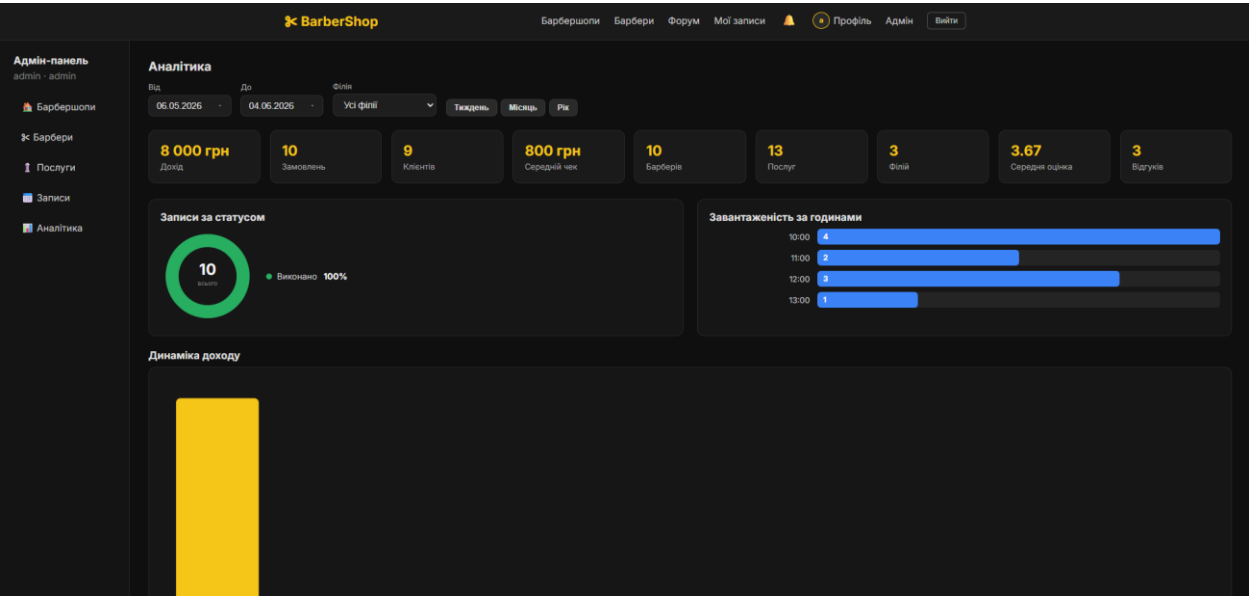


Рис. 3.20. Адмін-панель, аналітика

Для менеджера дані автоматично обмежуються його філією(рис. 3.21.), тоді як адміністратор бачить показники всієї мережі, що дозволяє оцінювати ефективність роботи та приймати обґрунтовані управлінські рішення.

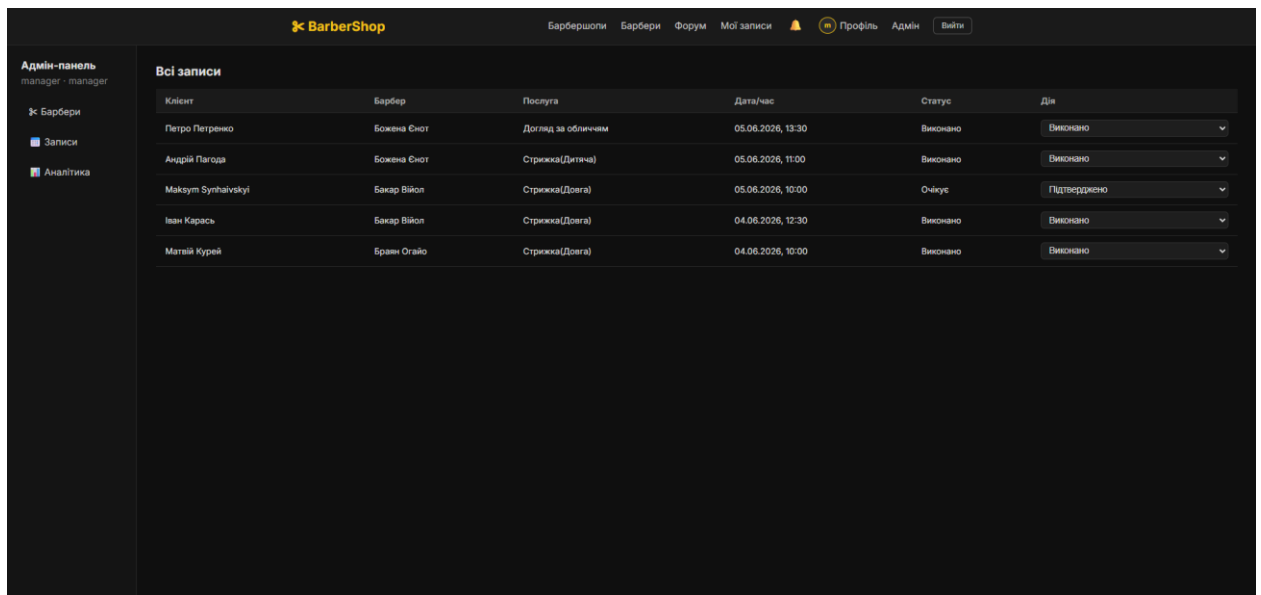


Рис. 3.21. Адмін-панель, записи, але для менеджера однієї з локацій

3.3. Тестування роботи програмного комплексу

Тестування є важливою складовою життєвого циклу розробки програмного забезпечення, оскільки саме воно дає змогу виявити дефекти та переконатися у відповідності продукту висунутим вимогам [19]. Під тестуванням розуміють процес перевірки програмного продукту, спрямований на оцінку його якості та відповідності очікуваній поведінці. Для перевірки розробленої вебплатформи застосовано ручний метод, за якого тест-кейси виконуються без використання автоматизованих інструментів. Тест-кейс являє собою детально описаний набір кроків, умов і очікуваних результатів, який дозволяє встановити, чи працює система та її окремі компоненти коректно. Нижче наведено основні тест-кейси, що охоплюють ключовий функціонал платформи.

У першому тест-кейсі перевіряється розмежування доступу, а саме спроба неавторизованого користувача відкрити сторінки, призначені лише для зареєстрованих користувачів (табл. 3.1).

Таблиця 3.1

Тест-кейс 1 – Розмежування доступу неавторизованого користувача

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Перехід за адресою /appointments	Перенаправлення на сторінку входу	Як і очікувалось	Виконано

продовж. табл. 3.1

2	Перехід за адресою /profile	Перенаправлення на сторінку входу	Як і очікувалось	Виконано
3	Перехід за адресою /admin	Перенаправлення на головну сторінку	Як і очікувалось	Виконано
4	Перехід за адресою /barber	Перенаправлення на головну сторінку	Як і очікувалось	Виконано

Другий тест-кейс присвячено перевірці процесів реєстрації та авторизації за різних умов введення даних (табл. 3.2).

Таблиця 3.2

Тест-кейс 2 – Реєстрація та авторизація

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Відправлення пустої форми реєстрації	Повідомлення про обов'язкові поля, реєстрація не виконується	Як і очікувалось	Виконано
2	Реєстрація з некоректним форматом пошти	Повідомлення про помилку введення	Як і очікувалось	Виконано
3	Введення паролів, що не співпадають	Повідомлення «Паролі не співпадають»	Як і очікувалось	Виконано
4	Реєстрація з уже зайнятою поштою	Повідомлення «Email вже зареєстровано»	Як і очікувалось	Виконано
5	Реєстрація з коректними даними	Успішне створення акаунта та вхід у систему	Як і очікувалось	Виконано
6	Вхід із невірним паролем	Повідомлення «Невірний email або пароль»	Як і очікувалось	Виконано
7	Вхід із коректними даними	Успішний вхід у систему	Як і очікувалось	Виконано

Третій тест-кейс перевіряє основний процес системи – онлайн-запис, зокрема коректність перевірки доступності часу (табл. 3.3).

Таблиця 3.3

Тест-кейс 3 – Онлайн-запис на послугу

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Вибір послуги, дати й вільного часу та підтвердження	Повідомлення «Запис створено», статус «очікує»	Як і очікувалось	Виконано
2	Додавання додаткової послуги	Збільшення тривалості та підсумкової вартості в записі	Як і очікувалось	Виконано
3	Спроба запису на вже зайнятий час	Повідомлення «Цей час вже зайнятий»	Як і очікувалось	Виконано
4	Вибір дня, коли майстер не працює	Відсутність вільних слотів, повідомлення «Майстер не працює»	Як і очікувалось	Виконано

Арк.

ІПЗ.КР.Б-121-26-ПЗ

63

продовж. табл. 3.3

5	Перевірка сповіщення барбера про новий запис	Барбер отримує сповіщення про створений запис	Як і очікувалось	Виконано
---	--	---	------------------	----------

Четвертий тест-кейс охоплює керування записами та зміну їхніх статусів різними ролями (табл. 3.4).

Таблиця 3.4

Тест-кейс 4 – Керування записами та статусами

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Барбер підтверджує запис	Статус «підтверджено», клієнт отримує сповіщення	Як і очікувалось	Виконано
2	Барбер позначає запис виконаним	Статус «виконано»	Як і очікувалось	Виконано
3	Клієнт скасовує власний запис	Статус «скасовано»	Як і очікувалось	Виконано
4	Клієнт намагається змінити чужий запис	Відмова «Недостатньо прав»	Як і очікувалось	Виконано

П'ятий тест-кейс перевіряє роботу системи відгуків та підтримання рейтингу майстрів (табл. 3.5).

Таблиця 3.5

Тест-кейс 5 – Відгуки та рейтинг

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Залишення відгуку після виконаного запису	Відгук успішно додано	Як і очікувалось	Виконано
2	Перевірка перерахунку середнього рейтингу барбера	Рейтинг та кількість відгуків оновлено	Як і очікувалось	Виконано
3	Адміністратор приховує відгук	Відгук перестає враховуватися й відображатися	Як і очікувалось	Виконано

Шостий тест-кейс присвячено функціоналу форуму (табл. 3.6).

Таблиця 3.6

Тест-кейс 6 – Робота форуму

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Створення нового поста із заголовком і текстом	Пост успішно опубліковано	Як і очікувалось	Виконано
2	Додавання коментаря до поста	Коментар з'являється під постом	Як і очікувалось	Виконано
3	Відповідь на наявний коментар	Створюється вкладений коментар	Як і очікувалось	Виконано

продовж. табл. 3.6

4	Редагування власного коментаря	Зміни збережено, доступна історія редагувань	Як і очікувалось	Виконано
5	Видалення власного коментаря	Коментар позначається як видалений	Як і очікувалось	Виконано
6	Оцінювання поста вподобайкою	Лічильник уподобань оновлюється	Як і очікувалось	Виконано

Сьомий тест-кейс перевіряє адміністративний функціонал та обмеження доступу за ролями (табл. 3.7).

Таблиця 3.7

Тест-кейс 7 – Адміністрування системи

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Додавання нового барбершопу	Філія з'являється у списку	Як і очікувалось	Виконано
2	Редагування послуги	Зміни збережено	Як і очікувалось	Виконано
3	Деактивація філії	Філія стає неактивною	Як і очікувалось	Виконано
4	Додавання нового барбера	Майстер з'являється у списку	Як і очікувалось	Виконано
5	Перегляд аналітики менеджером	Відображаються дані лише його філії	Як і очікувалось	Виконано
6	Спроба клієнта відкрити адмін-панель	Перенаправлення на головну сторінку	Як і очікувалось	Виконано

Восьмий тест-кейс перевіряє завантаження зображень із контролем формату та розміру файлів (табл. 3.8).

Таблиця 3.8

Тест-кейс 8 – Завантаження зображень

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Завантаження аватара у форматі jpg	Зображення завантажуються й відображається	Як і очікувалось	Виконано
2	Додавання роботи портфолію у форматі png	Зображення успішно зберігається	Як і очікувалось	Виконано
3	Спроба завантажити файл невідповідного формату	Відхилення з повідомленням про дозволені формати	Як і очікувалось	Виконано
4	Спроба завантажити файл розміром понад 5 МБ	Завантаження відхиляється	Як і очікувалось	Виконано

Отже, у ході тестування перевірено всі основні сценарії роботи вебплатформи, у яких могли б виникнути непередбачувані ситуації. За результатами перевірки встановлено, що система та її окремі компоненти

працюють коректно й відповідають висунутим вимогам.

Висновки до третього розділу

У третьому розділі розглянуто інтерфейс вебплатформи та порядок роботи з нею. У першому підрозділі описано порядок встановлення й налаштування параметрів системи, а також за допомогою діаграми розгортання детально розкрито, як розгортається кожен компонент системи та як вони взаємодіють між собою на апаратному рівні.

У другому підрозділі розглянуто структуру інтерфейсу вебплатформи. Інтерфейс розроблено з урахуванням принципів зручності та інтуїтивності, що дозволяє користувачам легко орієнтуватися в системі без спеціальної підготовки, а кожен його компонент забезпечує швидкий доступ до відповідних функцій та інформації для всіх категорій користувачів – клієнтів, барберів, адміністраторів і менеджерів.

У третьому підрозділі проведено тестування роботи програмного комплексу за допомогою ручного методу. Перевірку виконано для всіх ключових процесів системи, зокрема авторизації з розмежуванням доступу, онлайн-запису з контролем доступності часу, керування записами, відгуків, форуму, адміністрування та завантаження файлів. За результатами тестування підтверджено високу робочу надійність системи та відсутність дефектів у роботі основного функціоналу.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи досягнуто її основної мети – розроблено та реалізовано вебплатформу мережі барбершопів, що забезпечує автоматизацію основних бізнес-процесів та їх аналітику. Робота складалася з кількох послідовних етапів, які охопили аналіз предметної області, проєктування, програмну реалізацію й тестування системи.

У першому розділі проаналізовано напрями використання інформаційних технологій для реалізації подібних систем та визначено призначення майбутньої платформи. Розглянуто три наявні аналоги – сайти BarberKod, DANDY Barbershop і Black Peppers Barbershop, що дозволило виявити їхні сильні та слабкі сторони. Установлено, що наявним рішенням бракує особистого кабінету клієнта, системи відгуків, аналітичних інструментів і форуму для спілкування, що обґрунтувало доцільність розробки власної платформи з розширеним функціоналом. За результатами порівняння архітектурних підходів обрано клієнт-серверну модель із монолітною серверною реалізацією та поділом на логічні шари, а як технологічний стек – бібліотеку React, середовище Node.js із фреймворком Express та систему керування базами даних PostgreSQL.

У другому розділі виконано детальне проєктування та програмну реалізацію системи. На основі аналізу вимог визначено чотири ролі користувачів – клієнта, барбера, адміністратора й менеджера – та побудовано для них діаграми варіантів використання, що чітко розмежовують доступний функціонал. Розроблено структуру бази даних із двадцяти двох таблиць, подану у вигляді ER-діаграми та описану на рівні окремих таблиць із зазначенням полів, типів і ключів. Спроєктовано й описано ключові алгоритми роботи системи, серед яких онлайн-запис із перевіркою доступності часу, авторизація з розмежуванням прав доступу, опрацювання відгуків із підтриманням рейтингу та формування сповіщень. Реалізацію серверної частини виконано засобами Node.js та Express із автентифікацією на основі технології JSON Web Token, а клієнтську частину – засобами React із

використанням контексту для керування станом.

У третьому розділі розглянуто інтерфейс системи та порядок роботи з нею. За допомогою діаграми розгортання описано фізичну архітектуру платформи та порядок її встановлення й налаштування. Докладно проаналізовано структуру інтерфейсу, який побудовано з урахуванням принципів зручності та інтуїтивності й дозволяє користувачам легко орієнтуватися в системі без спеціальної підготовки. Проведено тестування програмного комплексу ручним методом, у ході якого перевірено всі ключові процеси системи, зокрема розмежування доступу, реєстрацію та авторизацію, онлайн-запис із захистом від подвійного бронювання, керування записами, відгуки, форум, адміністрування й завантаження зображень. Результати тестування підтвердили коректну роботу основного функціоналу та високу надійність розробленої системи.

Таким чином, у роботі реалізовано всі необхідні етапи створення працездатної вебплатформи мережі барбершопів, що відповідає поставленим вимогам. Отримані результати мають практичну цінність і можуть бути впроваджені в діяльність мереж барбершопів для покращення організації роботи, спрощення запису клієнтів та підвищення якості обслуговування, а також становлять основу для подальшого вдосконалення системи й розширення її функціональних можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React: офіційна документація. Meta Open Source. URL: <https://react.dev>
2. Node.js Documentation. OpenJS Foundation. URL: <https://nodejs.org/en/docs>
3. Express – Node.js web application framework. OpenJS Foundation. URL: <https://expressjs.com>
4. PostgreSQL 16 Documentation. The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/>
5. MDN Web Docs: JavaScript. Mozilla Foundation. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
6. MDN Web Docs: HTML. Mozilla Foundation. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML>
7. MDN Web Docs: CSS. Mozilla Foundation. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS>
8. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. 3rd ed. San Francisco: No Starch Press, 2018.
9. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol: O'Reilly Media, 2020.
10. Wieruch R. The Road to React: The React.js with Hooks in JavaScript Book. Leanpub, 2022.
11. Casciaro M., Mammino L. Node.js Design Patterns. 3rd ed. Birmingham: Packt Publishing, 2020.
12. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack. 2nd ed. Sebastopol: O'Reilly Media, 2019.
13. Obe R., Hsu L. PostgreSQL: Up and Running. 3rd ed. Sebastopol: O'Reilly Media, 2017.
14. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol: O'Reilly Media, 2020.
15. Kleppmann M. Designing Data-Intensive Applications. Sebastopol: O'Reilly Media, 2017.

16. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002.
17. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021.
18. Richardson L., Ruby S. RESTful Web Services. Sebastopol: O'Reilly Media, 2007.
19. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Boston: Prentice Hall, 2008.
20. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994.
21. Date C. J. An Introduction to Database Systems. 8th ed. Boston: Addison-Wesley, 2003.
22. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Berkeley: New Riders, 2014.
23. Norman D. The Design of Everyday Things: Revised and Expanded Edition. New York: Basic Books, 2013.
24. Nielsen J. Usability Engineering. San Francisco: Morgan Kaufmann, 1993.
25. Chacon S., Straub B. Pro Git. 2nd ed. New York: Apress, 2014.
26. ECMAScript 2023 Language Specification (ECMA-262). Geneva: ECMA International, 2023. URL: <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>
27. Fielding R., Reschke J. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230. IETF, 2014. URL: <https://www.rfc-editor.org/info/rfc7230>
28. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. IETF, 2015. URL: <https://www.rfc-editor.org/info/rfc7519>
29. Web Content Accessibility Guidelines (WCAG) 2.1. W3C Recommendation. World Wide Web Consortium, 2018. URL: <https://www.w3.org/TR/WCAG21/>
30. Architectural Styles and the Design of Network-based Software Architectures, 2000. URL: <https://ics.uci.edu/~fielding/pubs/dissertation/top.htm>

ДОДАТКИ

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						71

Діаграма класів

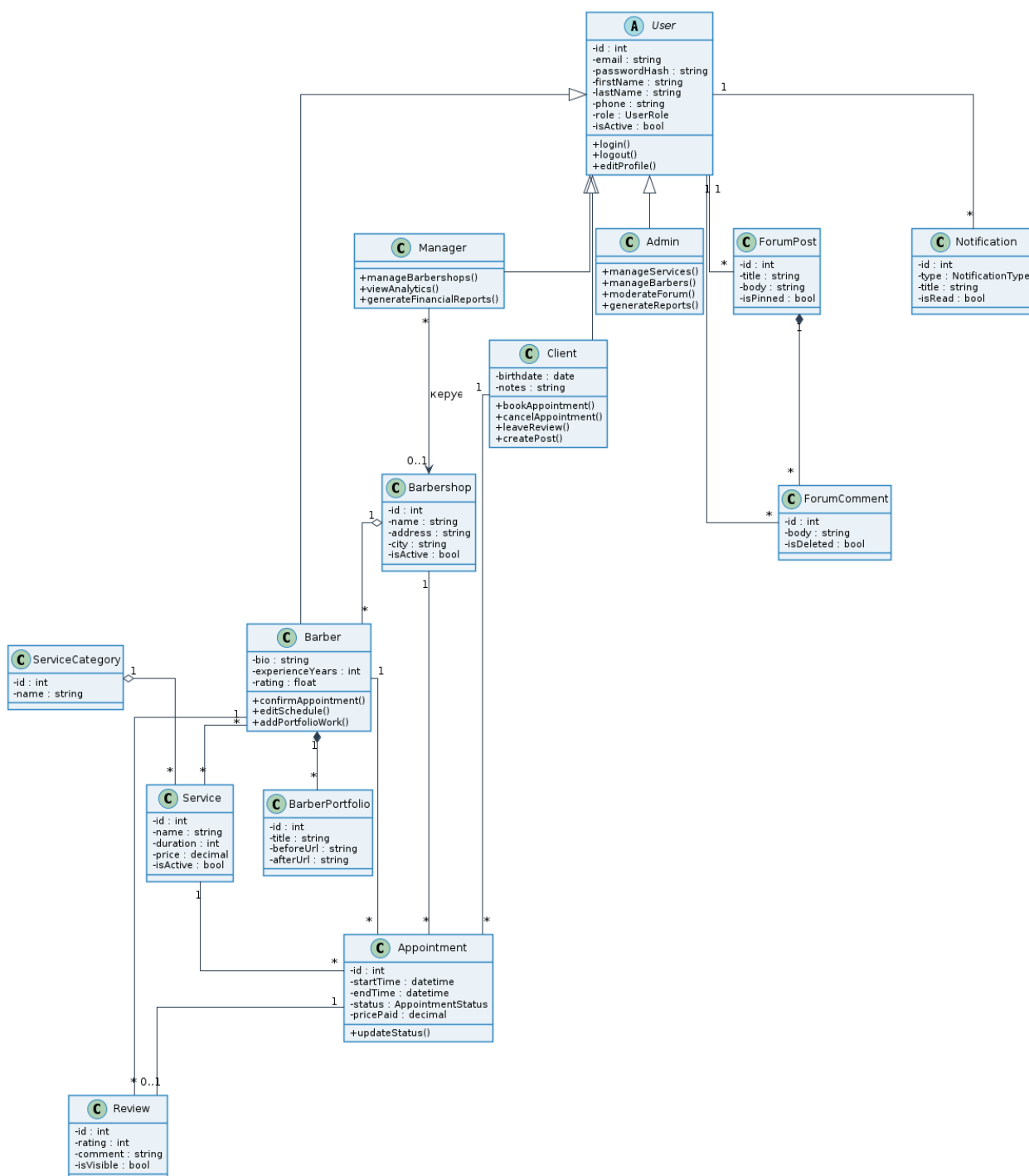


Рис. А.1. Діаграма класів

Діаграма активності

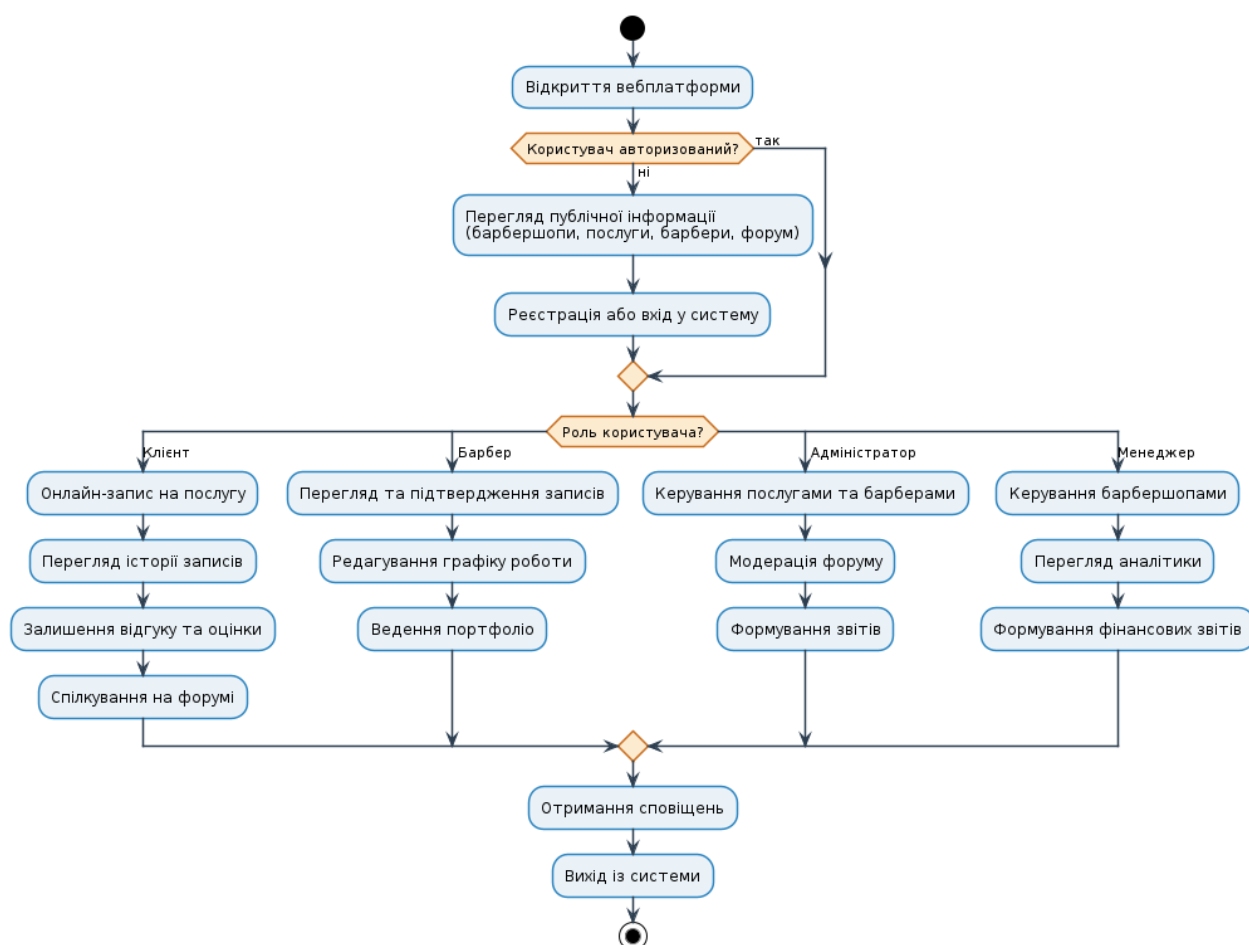


Рис. Б.1. Діаграма активності

ER-діаграма

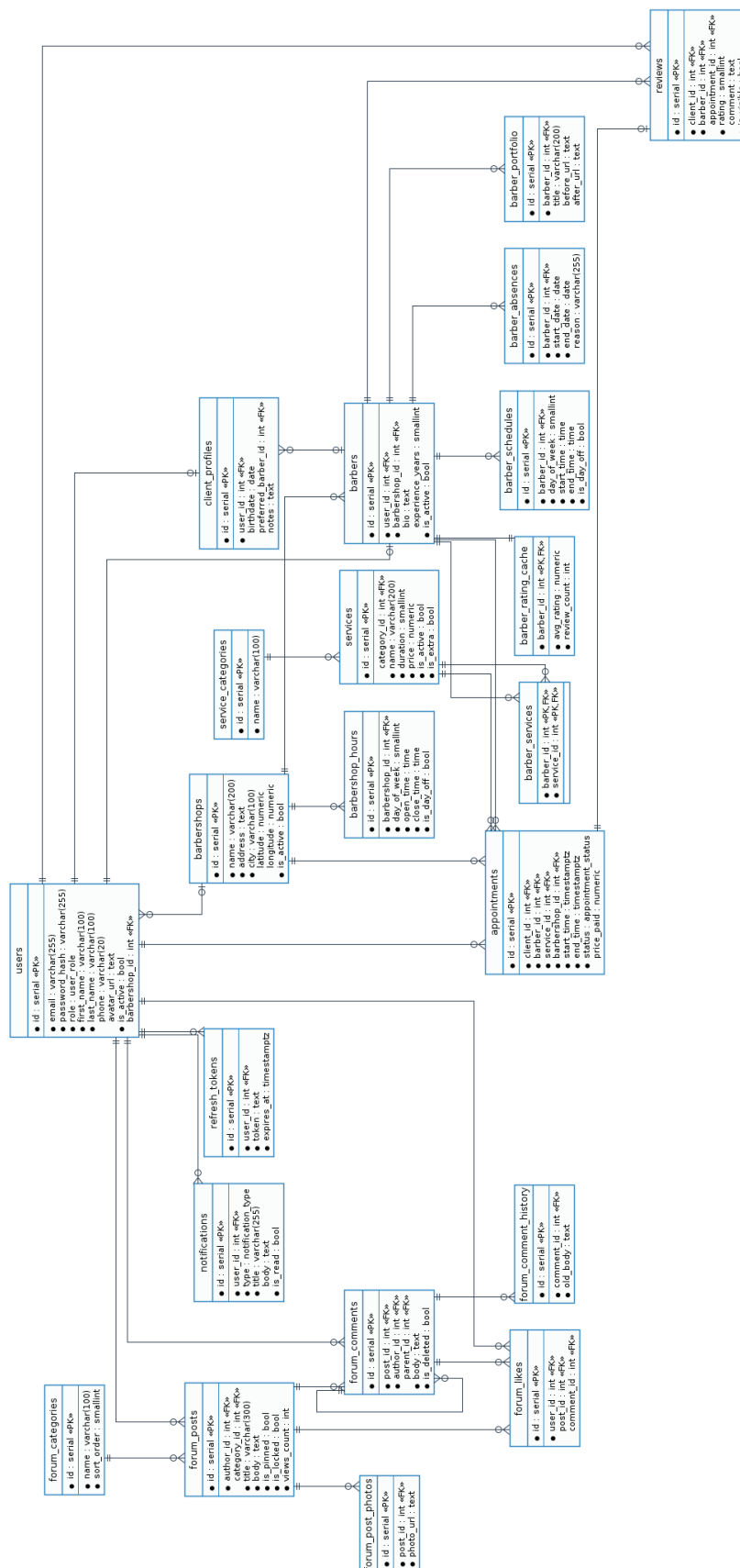


Рис. В.1. ER-діаграма

Лістинг програми index.js

```

const express = require('express');
const cors = require('cors');
const path = require('path');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;

app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:3000',
  credentials: true,
}));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

app.use('/api/auth', require('./routes/auth'));
app.use('/api/users', require('./routes/users'));
app.use('/api/barbershops', require('./routes/barbershops'));
app.use('/api/barbers', require('./routes/barbers'));
app.use('/api/services', require('./routes/services'));
app.use('/api/appointments', require('./routes/appointments'));
app.use('/api/reviews', require('./routes/reviews'));
app.use('/api/forum', require('./routes/forum'));
app.use('/api/notifications', require('./routes/notifications'));
app.use('/api/analytics', require('./routes/analytics'));

// Health check
app.get('/api/health', (req, res) => {
  res.json({ status: 'OK', timestamp: new Date() });
});

app.use((req, res) => {
  res.status(404).json({ error: 'Роут не знайдено' });
});

app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Внутрішня помилка сервера' });
});

const { startReminderJob } = require('./jobs/reminders');

app.listen(PORT, () => {
  console.log(`🚀 Сервер запущено: http://localhost:${PORT}`);
  startReminderJob(); // фонові нагадування про візити
});

```

Лістинг програми config/db.js

```
const { Pool } = require('pg');
require('dotenv').config();

const pool = new Pool({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
});

pool.on('connect', () => {
  console.log('✓ Підключено до PostgreSQL');
});

pool.on('error', (err) => {
  console.error('✗ Помилка PostgreSQL:', err.message);
  process.exit(1);
});

const query = (text, params) => pool.query(text, params);

module.exports = { pool, query };
```

Лістинг програми middleware/auth.js

```
const jwt = require('jsonwebtoken');
require('dotenv').config();

const authenticate = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ error: 'Токен відсутній' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded; // { id, email, role }
    next();
  } catch (err) {
    if (err.name === 'TokenExpiredError') {
      return res.status(401).json({ error: 'Токен прострочений' });
    }
    return res.status(401).json({ error: 'Недійсний токен' });
  }
};

module.exports = { authenticate };
```

Лістинг програми middleware/roles.js

```
const requireRole = (...roles) => (req, res, next) => {
  if (!req.user) {
    return res.status(401).json({ error: 'Не авторизовано' });
  }
  if (!roles.includes(req.user.role)) {
    return res.status(403).json({ error: 'Недостатньо прав' });
  }
  next();
};

module.exports = { requireRole };
```

Лістинг програми authController.js

```
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { query } = require('../config/db');
require('dotenv').config();

const generateTokens = (user) => {
  const payload = { id: user.id, email: user.email, role: user.role };

  const accessToken = jwt.sign(payload, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN,
  });
  const refreshToken = jwt.sign(payload, process.env.JWT_REFRESH_SECRET, {
    expiresIn: process.env.JWT_REFRESH_EXPIRES_IN,
  });
  return { accessToken, refreshToken };
};

const register = async (req, res) => {
  const { email, password, first_name, last_name, phone } = req.body;

  if (!email || !password || !first_name || !last_name) {
    return res.status(400).json({ error: "Заповніть усі обов'язкові поля" });
  }

  try {
    const existing = await query('SELECT id FROM users WHERE email = $1', [email]);
    if (existing.rows.length > 0) {
      return res.status(409).json({ error: 'Email вже зареєстровано' });
    }

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

    const result = await query(
      `INSERT INTO users (email, password_hash, first_name, last_name, phone)
      VALUES ($1, $2, $3, $4, $5)
      RETURNING id, email, role, first_name, last_name, phone, avatar_url`,
      [email, password_hash, first_name, last_name, phone || null]
    );
  }
};
```

```

const user = result.rows[0];

await query('INSERT INTO client_profiles (user_id) VALUES ($1)', [user.id]);

const { accessToken, refreshToken } = generateTokens(user);

const expiresAt = new Date();
expiresAt.setDate(expiresAt.getDate() + 7);
await query(
  'INSERT INTO refresh_tokens (user_id, token, expires_at) VALUES ($1, $2, $3)',
  [user.id, refreshToken, expiresAt]
);

return res.status(201).json({ user, accessToken, refreshToken });
} catch (err) {
  console.error('register error:', err.message);
  return res.status(500).json({ error: 'Помилка сервера' });
}
};

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

  if (!email || !password) {
    return res.status(400).json({ error: 'Введіть email та пароль' });
  }

  try {
    const result = await query(
      'SELECT id, email, role, first_name, last_name, phone, avatar_url, password_hash, is_active
FROM users WHERE email = $1',
      [email]
    );

    if (result.rows.length === 0) {
      return res.status(401).json({ error: 'Невірний email або пароль' });
    }

    const user = result.rows[0];

    if (!user.is_active) {
      return res.status(403).json({ error: 'Обліковий запис деактивовано' });
    }

    const valid = await bcrypt.compare(password, user.password_hash);
    if (!valid) {
      return res.status(401).json({ error: 'Невірний email або пароль' });
    }

    const { accessToken, refreshToken } = generateTokens(user);

    const expiresAt = new Date();
    expiresAt.setDate(expiresAt.getDate() + 7);
    await query(
      'INSERT INTO refresh_tokens (user_id, token, expires_at) VALUES ($1, $2, $3)',
      [user.id, refreshToken, expiresAt]
    );
  }
};

```

```

    const { password_hash, ...safeUser } = user;
    return res.json({ user: safeUser, accessToken, refreshToken });
  } catch (err) {
    console.error('login error:', err.message);
    return res.status(500).json({ error: 'Помилка сервера' });
  }
};

const refresh = async (req, res) => {
  const { refreshToken } = req.body;
  if (!refreshToken) return res.status(400).json({ error: 'Токен відсутній' });

  try {
    const decoded = jwt.verify(refreshToken, process.env.JWT_REFRESH_SECRET);

    const stored = await query(
      'SELECT id FROM refresh_tokens WHERE token = $1 AND expires_at > NOW()',
      [refreshToken]
    );
    if (stored.rows.length === 0) {
      return res.status(401).json({ error: 'Токен недійсний або прострочений' });
    }

    const userResult = await query(
      'SELECT id, email, role FROM users WHERE id = $1',
      [decoded.id]
    );
    const user = userResult.rows[0];
    const { accessToken, refreshToken: newRefresh } = generateTokens(user);

    await query('DELETE FROM refresh_tokens WHERE token = $1', [refreshToken]);
    const expiresAt = new Date();
    expiresAt.setDate(expiresAt.getDate() + 7);
    await query(
      'INSERT INTO refresh_tokens (user_id, token, expires_at) VALUES ($1, $2, $3)',
      [user.id, newRefresh, expiresAt]
    );

    return res.json({ accessToken, refreshToken: newRefresh });
  } catch (err) {
    return res.status(401).json({ error: 'Недійсний токен' });
  }
};

const logout = async (req, res) => {
  const { refreshToken } = req.body;
  if (refreshToken) {
    await query('DELETE FROM refresh_tokens WHERE token = $1', [refreshToken]);
  }
  return res.json({ message: 'Вихід виконано' });
};

module.exports = { register, login, refresh, logout };

```

Лістинг програми appointmentsController

```

const { query } = require('../config/db');
const { createNotification } = require('../notificationsController');
const { managerBranch } = require('../utils/scope');

const getAll = async (req, res) => {
  try {
    let sql = `
      SELECT a.id, a.start_time, a.end_time, a.status, a.notes, a.price_paid,
        a.barber_id,
        u.first_name || ' ' || u.last_name AS client_name,
        ub.first_name || ' ' || ub.last_name AS barber_name,
        s.name AS service_name,
        ROUND(EXTRACT(EPOCH FROM (a.end_time - a.start_time)) / 60) AS duration,
        bs.name AS barbershop_name,
        EXISTS (
          SELECT 1 FROM reviews r
          WHERE r.appointment_id = a.id AND r.client_id = a.client_id
        ) AS has_review
      FROM appointments a
      JOIN users u ON u.id = a.client_id
      JOIN barbers b ON b.id = a.barber_id
      JOIN users ub ON ub.id = b.user_id
      JOIN services s ON s.id = a.service_id
      JOIN barbershops bs ON bs.id = a.barbershop_id
    `;
    const params = [];

    if (req.user.role === 'client') {
      params.push(req.user.id);
      sql += ` WHERE a.client_id = ${params.length}`;
    } else if (req.user.role === 'barber') {
      const barberRow = await query('SELECT id FROM barbers WHERE user_id = $1', [req.user.id]);
      if (barberRow.rows.length === 0) return res.json([]);
      params.push(barberRow.rows[0].id);
      sql += ` WHERE a.barber_id = ${params.length}`;
    } else if (req.user.role === 'manager') {
      const mb = await managerBranch(req);
      params.push(mb);
      sql += ` WHERE a.barbershop_id = ${params.length}`;
    }
    // admin бачить усі

    sql += ' ORDER BY a.start_time DESC LIMIT 100';
    const result = await query(sql, params);
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
};

const getOne = async (req, res) => {
  try {
    const result = await query(
      `SELECT a.*, u.first_name || ' ' || u.last_name AS client_name,

```

						ІПЗ.КР.Б-121-26-ПЗ	Арк.
							80

```

        s.name AS service_name, bs.name AS barbershop_name
    FROM appointments a
    JOIN users u ON u.id = a.client_id
    JOIN services s ON s.id = a.service_id
    JOIN barbershops bs ON bs.id = a.barbershop_id
    WHERE a.id = $1`,
    [req.params.id]
  );
  if (result.rows.length === 0) return res.status(404).json({ error: 'Не знайдено' });

  const appt = result.rows[0];
  // Клієнт може бачити лише свої записи
  if (req.user.role === 'client' && appt.client_id !== req.user.id) {
    return res.status(403).json({ error: 'Недостатньо прав' });
  }
  res.json(appt);
} catch (err) {
  res.status(500).json({ error: 'Помилка сервера' });
}
};

const create = async (req, res) => {
  const { barber_id, service_id, barbershop_id, start_time, notes, total_duration } = req.body;
  if (!barber_id || !service_id || !barbershop_id || !start_time) {
    return res.status(400).json({ error: 'barber_id, service_id, barbershop_id, start_time – обов\'язкові' });
  }

  try {
    const svc = await query('SELECT duration, price FROM services WHERE id = $1', [service_id]);
    if (svc.rows.length === 0) return res.status(404).json({ error: 'Послугу не знайдено' });

    const { duration, price } = svc.rows[0];
    const effectiveDuration = total_duration && total_duration > duration ? total_duration : duration;
    const start = new Date(start_time);
    const end = new Date(start.getTime() + effectiveDuration * 60 * 1000);

    const conflict = await query(
      `SELECT id FROM appointments
      WHERE barber_id = $1
      AND status NOT IN ('cancelled', 'no_show')
      AND (start_time, end_time) OVERLAPS ($2::timestampz, $3::timestampz)`,
      [barber_id, start, end]
    );
    if (conflict.rows.length > 0) {
      return res.status(409).json({ error: 'Цей час вже зайнятий' });
    }

    const result = await query(
      `INSERT INTO appointments
      (client_id, barber_id, service_id, barbershop_id, start_time, end_time, notes, price_paid)
      VALUES ($1,$2,$3,$4,$5,$6,$7,$8)
      RETURNING *`,
      [req.user.id, barber_id, service_id, barbershop_id, start, end, notes, price]
    );
    const appt = result.rows[0];

```

```

const barberUser = await query('SELECT user_id FROM barbers WHERE id = $1', [barber_id]);
if (barberUser.rows.length) {
  await createNotification(
    barberUser.rows[0].user_id,
    'system',
    'Новий запис',
    'Клієнт створив новий запис. Деталі – у розділі «Записи».',
    '/appointments'
  );
}

res.status(201).json(appt);
} catch (err) {
  res.status(500).json({ error: 'Помилка сервера' });
}
};

const updateStatus = async (req, res) => {
  const { status } = req.body;
  const allowed = ['confirmed', 'completed', 'cancelled', 'no_show'];

  if (!allowed.includes(status)) {
    return res.status(400).json({ error: 'Недозволений статус' });
  }

  try {
    // Клієнт може лише скасовувати власний запис
    if (req.user.role === 'client') {
      if (status !== 'cancelled') {
        return res.status(403).json({ error: 'Недостатньо прав' });
      }
      const own = await query(
        'SELECT id FROM appointments WHERE id = $1 AND client_id = $2',
        [req.params.id, req.user.id]
      );
      if (own.rows.length === 0) {
        return res.status(403).json({ error: 'Недостатньо прав' });
      }
    }

    const extra = status === 'cancelled'
      ? ', cancelled_at = NOW(), cancel_reason = $3'
      : '';
    const params = status === 'cancelled'
      ? [status, req.params.id, req.body.cancel_reason || null]
      : [status, req.params.id];

    const result = await query(
      `UPDATE appointments SET status = $1 ${extra} WHERE id = $2 RETURNING *`,
      params
    );
    if (result.rows.length === 0) return res.status(404).json({ error: 'Не знайдено' });

    const appt = result.rows[0];
    // Сповіщення про зміну статусу запису
    if (status === 'confirmed') {
      await createNotification(appt.client_id, 'appointment_confirmed',

```

```

        'Запис підтверджено', 'Ваш запис підтверджено майстром.', '/appointments');
    } else if (status === 'completed') {
        await createNotification(appt.client_id, 'review_request',
            'Залиште відгук', 'Дякуємо за візит! Оцініть роботу майстра.', '/appointments');
    } else if (status === 'cancelled') {
        if (req.user.role === 'client') {
            // Клієнт скасував – повідомляємо барбера
            const bu = await query('SELECT user_id FROM barbers WHERE id = $1', [appt.barber_id]);
            if (bu.rows.length) {
                await createNotification(bu.rows[0].user_id, 'appointment_cancelled',
                    'Запис скасовано', 'Клієнт скасував свій запис.', '/appointments');
            }
        } else {
            // Скасував персонал – повідомляємо клієнта
            await createNotification(appt.client_id, 'appointment_cancelled',
                'Запис скасовано', 'Ваш запис було скасовано.', '/appointments');
        }
    }
    res.json(appt);
} catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
}
};

module.exports = { getAll, getOne, create, updateStatus };

```

Лістинг програми analyticsController

```

const { query } = require('../config/db');
const { managerBranch } = require('../utils/scope');

const getAnalytics = async (req, res) => {
    const today = new Date();
    const to = req.query.to || today.toISOString().slice(0, 10);
    const from = req.query.from || new Date(today.getTime() - 29 * 864e5).toISOString().slice(0, 10);
    const mb = await managerBranch(req);
    const barbershopId = mb !== undefined
        ? mb
        : (req.query.barbershop_id ? parseInt(req.query.barbershop_id, 10) : null);

    const days = Math.round((new Date(to) - new Date(from)) / 864e5) + 1;
    const bucket = days > 92 ? 'month' : 'day';

    const buildFilter = () => {
        const params = [from, to];
        let clause = `a.status = 'completed' AND a.start_time::date BETWEEN $1 AND $2`;
        if (barbershopId) {
            params.push(barbershopId);
            clause += ` AND a.barbershop_id = ${params.length}`;
        }
        return { clause, params };
    };

    const buildFilterAll = () => {
        const params = [from, to];
        let clause = `a.start_time::date BETWEEN $1 AND $2`;

```

```

    if (barbershopId) {
        params.push(barbershopId);
        clause += ` AND a.barbershop_id = ${params.length}`;
    }
    return { clause, params };
};

try {
    const fTotals = buildFilter();
    const fClients = buildFilter();
    const fBarbers = buildFilter();
    const fServices = buildFilter();
    const fBranch = buildFilter();
    const fSeries = buildFilter();
    const fStatus = buildFilterAll();
    const fLoad = buildFilterAll();

    const [totals, topClients, topBarbers, topServices, byBranch, series, statusBreak, loadByHour,
counts] = await Promise.all([
    query(
        `SELECT COALESCE(SUM(a.price_paid),0)::float AS revenue,
            COUNT(*)::int AS orders,
            COUNT(DISTINCT a.client_id)::int AS clients
            FROM appointments a WHERE ${fTotals.clause}`,
        fTotals.params
    ),
    query(
        `SELECT u.id, u.first_name || ' ' || u.last_name AS name,
            COUNT(*)::int AS orders,
            COALESCE(SUM(a.price_paid),0)::float AS spent
            FROM appointments a
            JOIN users u ON u.id = a.client_id
            WHERE ${fClients.clause}
            GROUP BY u.id, name
            ORDER BY spent DESC, orders DESC
            LIMIT 10`,
        fClients.params
    ),
    query(
        `SELECT b.id, ub.first_name || ' ' || ub.last_name AS name,
            COUNT(*)::int AS orders,
            COALESCE(SUM(a.price_paid),0)::float AS earned
            FROM appointments a
            JOIN barbers b ON b.id = a.barber_id
            JOIN users ub ON ub.id = b.user_id
            WHERE ${fBarbers.clause}
            GROUP BY b.id, name
            ORDER BY earned DESC, orders DESC
            LIMIT 10`,
        fBarbers.params
    ),
    query(
        `SELECT s.name,
            COUNT(*)::int AS count,
            COALESCE(SUM(a.price_paid),0)::float AS revenue
            FROM appointments a
            JOIN services s ON s.id = a.service_id

```

```

WHERE ${fServices.clause}
GROUP BY s.id, s.name
ORDER BY count DESC, revenue DESC
LIMIT 10`,
fServices.params
),
query(
`SELECT bs.name AS branch,
    COUNT(*)::int AS orders,
    COALESCE(SUM(a.price_paid),0)::float AS revenue
FROM appointments a
JOIN barbershops bs ON bs.id = a.barbershop_id
WHERE ${fBranch.clause}
GROUP BY bs.id, bs.name
ORDER BY revenue DESC`,
fBranch.params
),
query(
`SELECT to_char(date_trunc('${bucket}', a.start_time), 'YYYY-MM-DD') AS period,
    COALESCE(SUM(a.price_paid),0)::float AS revenue,
    COUNT(*)::int AS orders
FROM appointments a
WHERE ${fSeries.clause}
GROUP BY period
ORDER BY period`,
fSeries.params
),
query(
`SELECT status, COUNT(*)::int AS count
FROM appointments a WHERE ${fStatus.clause}
GROUP BY status`,
fStatus.params
),
query(
`SELECT EXTRACT(HOUR FROM a.start_time)::int AS hour, COUNT(*)::int AS count
FROM appointments a
WHERE ${fLoad.clause} AND a.status NOT IN ('cancelled','no_show')
GROUP BY hour ORDER BY hour`,
fLoad.params
),
query(
`SELECT
    (SELECT COUNT(*) FROM barbers WHERE is_active)::int AS barbers,
    (SELECT COUNT(*) FROM services WHERE is_active)::int AS services,
    (SELECT COUNT(*) FROM barbershops WHERE is_active)::int AS branches,
    (SELECT COUNT(*) FROM reviews WHERE is_visible)::int AS reviews,
    COALESCE((SELECT ROUND(AVG(rating),2) FROM reviews WHERE is_visible),0)::float
AS avg_rating`,
[]
),
]);

const t = totals.rows[0];
res.json({
    range: { from, to, bucket },
    totals: {
        revenue: t.revenue,

```

```

        orders: t.orders,
        clients: t.clients,
        avg_check: t.orders ? +(t.revenue / t.orders).toFixed(2) : 0,
    },
    top_clients: topClients.rows,
    top_barbers: topBarbers.rows,
    top_services: topServices.rows,
    by_branch: byBranch.rows,
    series: series.rows,
    status_breakdown: statusBreak.rows,
    load_by_hour: loadByHour.rows,
    counts: counts.rows[0],
  });
} catch (err) {
  console.error('analytics error:', err.message);
  res.status(500).json({ error: 'Помилка сервера' });
}
};

module.exports = { getAnalytics };

```

Лістинг програми reminders.js

```

const { query } = require('../config/db');
const { createNotification } = require('../controllers/notificationsController');

const REMINDER_HOURS = 24;
const CHECK_INTERVAL_MS = 15 * 60 * 1000;

const runReminderCheck = async () => {
  try {
    const { rows } = await query(
      `SELECT a.id, a.client_id
      FROM appointments a
      WHERE a.status IN ('pending', 'confirmed')
      AND a.start_time > NOW()
      AND a.start_time <= NOW() + INTERVAL '${REMINDER_HOURS} hours'
      AND NOT EXISTS (
        SELECT 1 FROM notifications n
        WHERE n.user_id = a.client_id
        AND n.type = 'appointment_reminder'
        AND n.link = '/appointments#' || a.id
      )`
    );

    for (const r of rows) {
      await createNotification(
        r.client_id,
        'appointment_reminder',
        'Нагадування про візит',
        'У вас незабаром запис у барбершопі.',
        `/appointments#${r.id}`
      );
    }

    if (rows.length) console.log(`🔔 Створено нагадувань: ${rows.length}`);
  } catch (err) {
    console.error('Reminder check failed:', err);
  }
};

```

```

        console.error('reminder job error:', err.message);
    }
};

const startReminderJob = () => {
    setTimeout(runReminderCheck, 30 * 1000);
    setInterval(runReminderCheck, CHECK_INTERVAL_MS);
    console.log('□ Планувальник нагадувань запущено');
};

module.exports = { startReminderJob, runReminderCheck };

```