**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інженерії програмного забезпечення**

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

___________ Євген Давиденко

«___» _________ 2026 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

# Інформаційна система автосервісу

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

**Здобувач** _______________ **Максим ЛАВРЕНЮК**

«__» ________ 20__ р.

**Керівник роботи**

PhD,

Ст.викладач _______________ **Ігор КАНДИБА**

«__» ________ 20__ р.

**Миколаїв – 2026**

**Завдання на виконання кваліфікаційної роботи**

Чорноморський національний університет імені Петра Могили

| | |
|---|---|
| Факультет | Комп'ютерних наук |
| Кафедра | Інженерії програмного забезпечення |
| Рівень вищої освіти | Перший (бакалаврський) |
| Освітній ступінь | Бакалавр |
| Спеціальність | 121 Інженерія програмного забезпечення |
| Освітня програма | Інженерія програмного забезпечення |

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_________ Євген ДАВИДЕНКО
«___» __________ 2026 р.

**ЗАВДАННЯ**

**на кваліфікаційну бакалаврську роботу здобувача**

**Лавренюка Максима**

1. Тема кваліфікаційної роботи інформаційна система автосервісу затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.

2. Строк представлення кваліфікаційної роботи «___» червня 2026 р.

3. Очікуваним результатом роботи є створення веборієнтованої інформаційної системи для комплексної автоматизації процесів автосервісу.

4. Перелік питань, що підлягають розробці:

– аналіз предметної області та формування вимог до системи;

– проєктування архітектури та структури бази даних;

– розробка алгоритмів автоматизованого планування та бронювання;

– реалізація функціоналу управління замовленнями та онлайн-консультаціями;

– інтеграція із зовнішніми сервісами для оплати, авторизації та зберігання медіаданих;

– впровадження адміністративної панелі для керування даними системи;

– розробка серверної частини та клієнтського інтерфейсу;

– здійснення тестування розробленого продукту.

5. Презентація.

6. Консультанти:

| **Консультант** | **Кафедра (організація)** | **Частина роботи** |
|---|---|---|
| | | |
| | | |
| | | |

Дата видачі завдання «26» грудня 2025 р.

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

Тема: **Інформаційна система автосервісу**

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1. | Розробка та затвердження завдання на виконання КБР | 20.12.2025 | 26.12.2025 | Виконано |
| 2. | Огляд літератури за темою роботи | 27.12.2025 | 30.12.2025 | Виконано |
| 3. | Складання календарного плану КБР | 02.02.2026 | 03.02.2026 | Виконано |
| 4. | Аналіз предметної області | 04.02.2026 | 11.02.2026 | Виконано |
| 5. | Проєктування архітектури системи | 11.02.2026 | 09.03.2026 | Виконано |
| 6. | Моделювання та конструювання ПЗ | 10.03.2026 | 27.03.2026 | Виконано |
| 7. | Реалізація бекенд-логіки та тестування | 30.03.2026 | 15.04.2026 | Виконано |
| 8. | Реалізація клієнтської частини | 16.04.2026 | 03.05.2026 | Виконано |
| 9. | Відгук керівника КБР | 04.05.2025 | 05.05.2025 | Виконано |
| 10. | Оформлення КБР та презентації | 07.05.2026 | 20.06.2026 | Виконано |
| 11. | Попередній захист | 27.05.2026 | 27.05.2026 | Виконано |
| 12. | Завершення оформлення КБР та презентації | 04.06.2026 | 09.06.2026 | Виконано |
| 13. | Рецензування | | | |
| 14. | Захист кваліфікаційної роботи | | | |

**Здобувач** ____________________ **Максим ЛАВРЕНЮК**
«__» ___________ 20__ р.

**Керівник роботи**
PhD, ст.викладач ____________________ **Ігор КАНДИБА**
«__» ___________ 20__ р.

# АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

**Інформаційна система автосервісу**

Здобувач 409 гр.: Лавренюк Максим

Керівник: PhD, ст. викладач Кандиба Ігор

Актуальність роботи пов’язана зі зростанням потреби в автоматизації інформаційної взаємодії між учасниками сервісу. Сучасні вимоги до галузі передбачають розробку інформаційних систем, здатних синхронізувати внутрішні робочі процеси, забезпечувати прозорий контроль виконання заявок та впровадження механізмів автоматизованого інформування та взаємодії з користувачами.

Метою роботи є розробка веборієнтованої інформаційної системи автосервісу для спрощення комунікації з клієнтами.

Об’єктом роботи є процеси взаємодії користувачів в інформаційній системі автосервісного обслуговування.

Предметом роботи є методи та програмні засоби розробки інформаційної системи автосервісу для спрощення комунікації з клієнтами.

Система забезпечує автоматизоване формування сервісних замовлень, координацію роботи персоналу та контроль виконання робіт, створюючи єдине цифрове середовище для ефективного обслуговування та зручної взаємодії користувачів.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність теми, визначено мету та завдання кваліфікаційної роботи, а також охоплено об’єкт і предмет роботи.

Перший розділ присвячено аналізу існуючих рішень у сфері автосервісу, виявленню їхніх функціональних та архітектурних обмежень, а також формуванню ключових напрямків проєктування нової системи.

У другому розділі проведено аналіз сучасного стану інструментарію, моделей та методів проєктування інформаційних систем на основі актуальних наукових публікацій. Сформовано детальну специфікацію вимог до програмного забезпечення.

Третій розділ присвячено етапам проєктування системи, включно з моделюванням бізнес-процесів, розробкою сценаріїв використання, побудовою UML-діаграм різних типів та створенням макетів користувацького інтерфейсу застосунку.

Четвертий розділ містить опис практичної реалізації вебзастосунку, зокрема розробку серверної та клієнтської частин, реалізацію бази даних, backend-логіки та API, а також опис процесу тестування програмного забезпечення.

У висновках узагальнено результати кваліфікаційної роботи, що охоплюють аналіз предметної області, формування вимог до системи, моделювання програмного забезпечення, розробку інформаційної системи та перевірку коректності її функціонування засобами тестування.

Кваліфікаційна робота викладена на 77 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 17 найменувань та 2 додатків. Праця містить 8 таблиць та 34 рисунка.

*Ключові слова: автоматизація, автосервіс, інформаційна взаємодія, інформаційна система, вебзастосунок.*

**ABSTRACT**

to the qualifying bachelor's thesis

**Car service information system**

Student, Group 409: Lavreniuk Maksym

Supervisor: PhD, senior lecturer Kandyba Ihor

The relevance of the work is associated with the growing need for automation of information interaction between service participants. Modern requirements for the industry imply the development of information systems capable of synchronizing internal work processes, providing transparent control over the execution of requests, and implementing mechanisms for automated informing and interaction with users.

The purpose of the work is to develop a web-based information system for a car service to simplify communication with clients.

The object of the work is the processes of user interaction in the car service information system.

The subject of the study is the methods and software tools for developing an auto service information system to simplify communication with clients.

The system provides automated creation of service orders, coordination of staff activities, and control over task execution, forming a unified digital environment for efficient service delivery and convenient user interaction.

The qualification work consists of an introduction, 4 sections, conclusions and a list of references.

The introduction substantiates the relevance of the topic, defines the aim and objectives of the qualification work, and covers the object and subject of the work.

The first chapter is devoted to the analysis of existing solutions in the field of automotive service systems, identification of their functional and architectural limitations, as well as the formation of key directions for designing a new system.

The second chapter presents an analysis of the current state of tools, models, and methods for designing information systems based on up-to-date scientific publications. A detailed specification of software requirements has been developed.

The third chapter is devoted to the system design stages, including business process modeling, development of use-case scenarios, construction of various types of UML diagrams, and creation of user interface mockups for the application.

The fourth chapter contains a description of the practical implementation of the web application, including the development of the server-side and client-side parts, implementation of the database, backend logic and API, as well as a description of the software testing process.

The conclusions summarize the results of the qualification work, including the analysis of the subject area, formation of system requirements, software modeling, development of the information system, and verification of the correctness of its functioning through testing.

The qualification work is presented on 77 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 17 titles and 2 appendices. The work contains 8 tables and 34 figures.

*Keywords: automation, car service, information interaction, information system, web application..*

# ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ .......... 4
ВСТУП .......... 5
1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ .......... 7
1.1 Аналіз предметної області .......... 7
1.2 Огляд існуючих інформаційних систем .......... 8
1.3 Переваги запропонованої системи .......... 16
Висновки до розділу 1 .......... 17
2 ОГЛЯД ІНСТРУМЕНТАРІЮ, МОДЕЛЕЙ ТА ВИМОГ ДО ПРОЄКТУВАННЯ СИСТЕМИ .......... 19
2.1 Аналіз та обґрунтування вибору методів .......... 19
2.2 Специфікація вимог до програмного забезпечення .......... 23
Висновки до розділу 2 .......... 33
3 МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .......... 35
3.1 UML-моделювання системи .......... 35
3.1.1 Діаграма прецедентів .......... 35
3.1.2 Визначення сутностей та їх зв’язків системи автосервісу .......... 37
3.1.3 Визначення класів системи .......... 39
3.1.4 Діаграма послідовності .......... 41
3.1.5 Проєктування діаграми розгортання .......... 43
3.2 Розробка макетів інтерфейсу користувача системи автосервісу .......... 45
Висновки до розділу 3 .......... 49
4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОСЕРВІСУ .......... 50
4.1 Розробка моделі бази даних .......... 50
4.2 Організація моделей сутностей .......... 52
4.3 Розробка серверної архітектури та бізнес-логіки .......... 53
4.4 Тестування програмного забезпечення .......... 58

4.5 Опис інтерфейсу та функціональних можливостей системи ........... 65
Висновки до розділу 4 ........................................................................... 73
ВИСНОВКИ.............................................................................................. 74
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ......................................................... 76
ДОДАТОК А ЛІСТИНГ КОДУ СЕРВЕРНОЇ ЧАСТИНИ ............................... 78
ДОДАТОК Б ЛІСТИНГ КОДУ КЛІЄНТСЬКОЇ ЧАСТИНИ ........................... 86

# ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення

СТО – станція технічного обслуговування

ACID – Atomicity Consistency Isolation Durability

API – Application Programming Interface

CDN – Content Delivery Network

ER – Entity-Relationship

ERP – Enterprise Resource Planning

gRPC – Google Remote Procedure Call

JWT – JSON Web Token

JSON – JavaScript Object Notation

LINQ – Language Integrated Query

OBD-II – On-Board Diagnostics II

ORM – Object-Relational Mapping

RBAC – Role-Based Access Control

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

UML – Unified Modeling Language

XML – Extensible Markup Language

# ВСТУП

Ефективність сучасного автосервісу залежить не лише від якості ремонту, а й від швидкості взаємодії з клієнтом. Проте більшість станцій технічного обслуговування в Україні досі використовують ручне ведення записів, що призводить до затримок у роботі та не відповідає сучасним вимогам до сервісу.

Актуальність роботи пов'язана зі зростанням потреби в автоматизації процесів інформаційної взаємодії учасників сервісу. Сучасні вимоги до галузі передбачають розробку інформаційних систем, здатних синхронізувати внутрішні робочі процеси, забезпечувати прозорий контроль виконання заявок та впровадження механізмів автоматизованого інформування та взаємодії з користувачами.

**Метою роботи** є розробка веборієнтованої інформаційної системи автосервісу для спрощення комунікації з клієнтами.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз предметної області та існуючих сервісів для обслуговування авто з метою виявлення функціональних прогалин;

– визначити функціональні та технічні вимоги до системи, розробити сценарії використання для клієнтів, адміністраторів та майстрів;

– розробити ER-діаграму бази даних;

– забезпечити можливості майстрам змінювати статус виконання робіт з автоматичним відображенням змін для інформування клієнтів;

– створити функціонал автоматичного відображення та оновлення графіків роботи майстрів на основі актуальних замовлень клієнта;

– реалізувати функціонал відображення доступних часових інтервалів з урахуванням завантаженості майстрів;

– виконати функціонал онлайн-консультацій та формування рекомендацій за результатами огляду транспортного засобу;

– розробити адміністративну панель для керування даними системи, що забезпечить додавання, редагування та видалення послуг, майстрів і категорій послуг;

– розробити серверну частину з використанням технології REST API;

– забезпечити адаптивний клієнтський інтерфейс для зручного користування системою на різних пристроях;

– провести тестування розробленого рішення.

**Об’єктом роботи** є процеси взаємодії користувачів в інформаційній системі автосервісного обслуговування.

**Предметом роботи** є методи та програмні засоби розробки інформаційної системи автосервісу для спрощення комунікації з клієнтами.

Значущість одержаних результатів роботи полягає в поєднанні теоретичних підходів до автоматизації сервісних процесів з їх практичною реалізацією у вигляді діючого програмного продукту.

Розроблений застосунок дозволить клієнтам та автосервісу без зайвих зусиль взаємодіяти між собою, оформлювати записи на ремонт та контролювати весь цикл співпраці.

# 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 1.1 Аналіз предметної області

Автосервіс – складна ремонтно-технічна система, продуктивність якої безпосередньо залежить від організації внутрішніх робочих процесів. Зростання кількості транспортних засобів і ускладнення їхнього технічного оснащення поступово роблять методи управління менш придатними для сучасних умов. Цифрові інструменти значною мірою замінили застарілі методи ведення записів.

Більшість виробничих процесів в автосервісі мають виразний динамічний характер. На організацію робіт впливають тип несправності, доступність діагностичного обладнання, кваліфікація конкретного спеціаліста та поточне завантаження. Через це планування робіт потребує постійних коригувань, а фіксований розклад швидко втрачає актуальність.

Процес обслуговування автомобіля охоплює кілька послідовних етапів: прийом заявки, діагностику, узгодження вартості, розподіл завдань між майстрами, безпосереднє виконання робіт і фінансове закриття замовлення. Кожен із них пов'язаний з обробкою певного обсягу інформації та прийняттям рішень. Помилки на стадії діагностики змінюють обсяг робіт уже в ході виконання, що безпосередньо позначається на графіку й завантаженні інших спеціалістів.

Дослідження у сфері управління підприємствами підтверджують, що впровадження інформаційних систем підвищує ефективність роботи завдяки централізації даних та скороченню кількості ручних операцій [1]. Прозорість процесів зростає, а управлінські рішення спираються на актуальну інформацію. Для автосервісу це означає можливість відстеження стану кожного замовлення в реальному часі, швидкого доступу до сервісної історії та точнішого планування завантаження персоналу.

Автоматизація логістики обслуговування охоплює управління потоками завдань і розподіл навантаження між виконавцями. На відміну від класичної логістики, тут ідеться не про фізичне переміщення матеріальних ресурсів, а про координацію інформації й дій між учасниками процесу. Налагоджена організація цих потоків зменшує кількість простоїв, усуває накладки у розкладі й підвищує ефективність використання робочого часу.

Інформаційна система автосервісу в такому контексті виконує роль не просто електронного журналу послуг, а повноцінного інструмента координації. Її функціональна модель забезпечує прозорий зв'язок між запитом клієнта та реальними можливостями станції [2]. Система об'єднує всі етапи обслуговування в єдиний інформаційний простір, де кожна дія користувача або працівника автоматично відображається на загальному стані процесу.

## 1.2 Огляд існуючих інформаційних систем

Розглядаються дві категорії програмних рішень, поширених на ринку, – комплексні хмарні ERP-системи та клієнтські вебсервіси. Кожен підхід має власну архітектуру і по-різному впливає на процеси автосервісу, визначаючи рівень автоматизації, гнучкість управління та ефективність взаємодії з клієнтами.

Порівняльний аналіз зазначених категорій розкриває їхні функціональні можливості, виявляє обмеження у застосуванні та характерні підходи до організації внутрішніх і зовнішніх процесів. Отримані висновки формують основу для подальших вимог до інформаційної системи автосервісу.

Першу категорію представляють хмарні ERP-системи. Серед рішень для малого та середнього бізнесу найбільш відомим є Microsoft Dynamics 365 Business Central – повноцінне середовище для керування всіма внутрішніми процесами підприємства в єдиному інтерфейсі [3].

Стосовно потреб автосервісу, платформа автоматизує внутрішню логістику. Модуль управління забезпечує детальний опис ремонтних замовлень: тип робіт, витрачені матеріали, відповідальний фахівець. Після завершення ремонту адміністратор фіксує витрачений час та списує запчастини з фактичних залишків.

Завдяки інтеграції між модулями складська система автоматично коригує залишки після кожної операції, а фінансовий модуль одразу формує рахунок для замовника (рис. 1.1). Платформа також підтримує автоматизацію через Power Automate – зокрема, відправку SMS-повідомлень про готовність авто та аналітику завантаженості персоналу через Power BI.

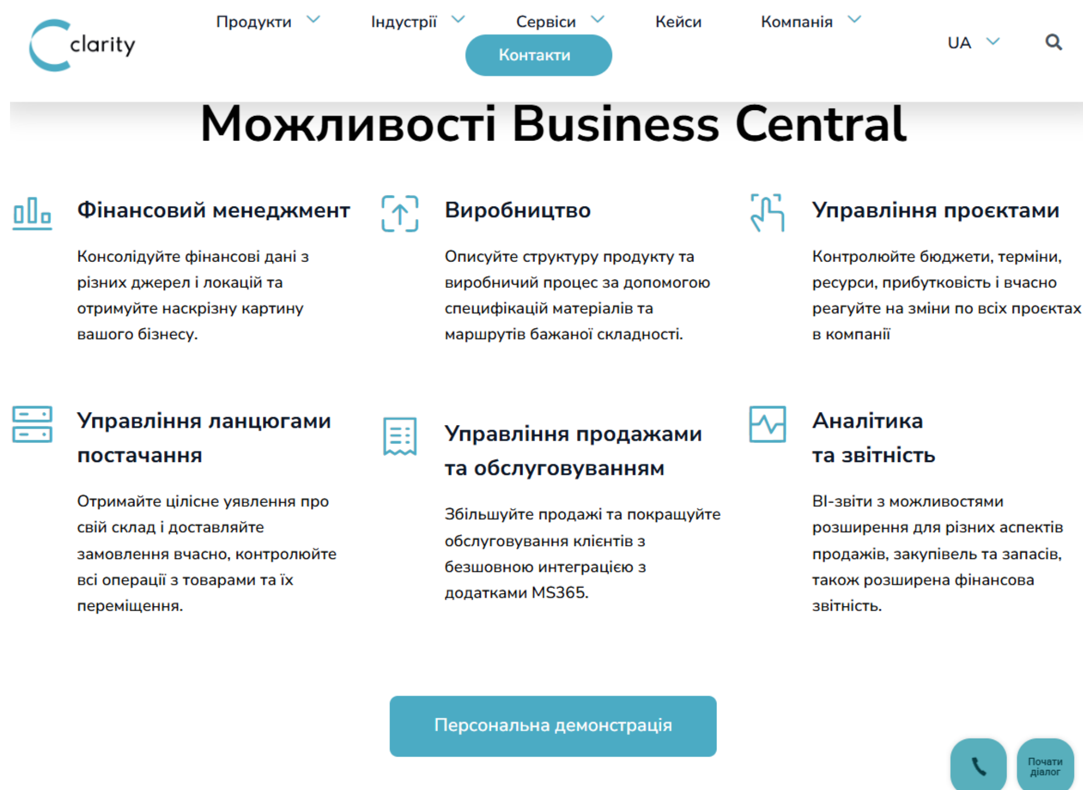


Рисунок 1.1 – Інтерфейс модулів управління Microsoft Dynamics 365 Business Central [3]

Сучасні версії платформи відкриті для зовнішніх інтеграцій – на відміну від класичних десктопних програм. Інструмент Power Pages відкриває адміністраторам доступ до публікації вебпорталів для клієнтів. Попри

широкий набір інструментів, Dynamics 365 не підходить як фундамент для розроблюваної системи з інженерної точки зору.

В екосистемі Microsoft клієнтська частина та бізнес-логіка механічно розділені. Power Pages не підтримує реалізацію складної логіки обробки запитів. Підключення власної логіки вимагає розширення ERP-системи мовою AL, прив'язаною виключно до продуктів Microsoft. Жорстка залежність від вендора обмежує переносимість і підтримуваність проєкту в довгостроковій перспективі.

Крім того, створення клієнтського вебсайту поверх бази даних ERP має певні обмеження. База даних Business Central має складну структуру, розраховану на фінансові транзакції. У разі запиту з зовнішнього клієнтського сайту система звертається до API для отримання інформації. Оскільки система накладає суворі обмеження на кількість API-запитів, для уникнення зниження продуктивності необхідно використовувати проміжну архітектуру з кешуванням даних.

Другу групу становлять веборієнтовані сервіси, через які клієнти звертаються за послугами автосервісу. Платформи різняться рівнем функціональної складності та ступенем автоматизації обробки запитів. Для порівняння обрано три рішення з різними підходами до взаємодії з клієнтом: AutoDoc, AutoTech та VIDI. Логіка роботи у кожного відрізняється – від простих форм подачі заявки до розвинених систем керування бронюванням.

Порівняння виявляє основні обмеження наявних рішень у сфері автоматизації автосервісу та окреслює напрямки вдосконалення в контексті цифровізації управління ремонтом.

Першим прикладом є AutoDoc – мінімалістичний сервіс для збору контактних даних і базової інформації про потрібну послугу [4]. Інтерфейс містить просту форму для запису на станцію технічного обслуговування (рис. 1.2). Кількість обов'язкових полів зведена до мінімуму, а авторизація перед відправленням запиту не потрібна.

Звернення оформлюється за кілька кліків без реєстрації та зайвих кроків – зручний варіант для разового контакту з незнайомою станцією. Мінімальний поріг входу і висока швидкість оформлення роблять сервіс придатним для першого звернення.

Запис на відвідування СТО AutoDoc

Ім'я *

Телефон *

Оберіть бажані послуги (можна декілька) *

- Технічне обслуговування
- Ремонт двигуна
- Кузовний ремонт
- Діагностика автомобіля
- Ремонт ходової
- Развал - Сходження

Модель авто

Я надаю згоду на обробку та використання моїх персональних даних відповідно до Закону України «Про захист персональних даних». Володільцем персональних даних є ТОВ Компанія «Автодок» (Україна, 02093, місто Київ, вулиця Бориспільська, будинок 30)

Рисунок 1.2 – Базова форма для подачі заявки сервісу AutoDoc [4]

На сайті розміщено опис послуг автосервісу, завдяки чому користувач може заздалегідь ознайомитися з переліком робіт і приблизно визначити потрібний вид обслуговування. Проте можливості сервісу залишаються обмеженими. Дані клієнта не зберігаються між зверненнями, тому під час кожного нового запису необхідно повторно вводити контактну інформацію та відомості про автомобіль.

Після оформлення заявки користувач не має доступу до інформації про її подальшу обробку. Відсутня можливість переглянути статус запису,

самостійно змінити дату або скасувати візит. У результаті уточнення деталей виконується через телефонний зв’язок із менеджером, що ускладнює взаємодію та збільшує час обробки звернень (табл. 1.1).

Таблиця 1.1 – Переваги та недоліки системи AutoDoc

| Переваги | Недоліки |
|---|---|
| – Клієнт може залишити базову заявку на ремонт у кілька кліків без зайвої реєстрації;<br>– наявність розгорнутого тексту допомагає клієнту самостійно визначити потрібну послугу;<br>– сервіс працює швидко та стабільно на будь-яких пристроях. | – Система не зберігає профіль користувача та дані його автомобіля, що виключає можливість формування історії обслуговування;<br>– після відправки форми клієнт втрачає контроль над нею і не може скасувати запис без дзвінка на станцію;<br>– клієнт не отримує орієнтовної вартості послуг до приїзду на станцію. |

Більш функціональним рішенням є мережа автосервісів AutoTech [5]. Сервіс пропонує розширений набір функцій порівняно з базовими системами запису. Окрім стандартної форми запису на ремонт (рис. 1.3), користувач замовляє додаткові послуги – сезонний технічний огляд, виклик евакуатора та самостійно обирає зручну дату і час бронювання. Розширений каталог частково впорядковує взаємодію замовника зі станцією ще до безпосереднього звернення.

Система забезпечує цілодобову доступність сервісу, що підвищує зручність для користувачів та зменшує залежність від робочого графіку операторів. Крім того, послуги логічно згруповані за категоріями, що спрощує навігацію та дозволяє швидше знайти необхідний тип обслуговування. Таким

чином, інтерфейс орієнтований на базову зручність користувача та мінімізацію часу оформлення запиту.

Рисунок 1.3 – Залишання заявки на послуги сервісу AutoTech [5]

Клієнт не отримує жодного підтвердження про те, що його заявку побачили або прийняли в роботу. Тобто сервіс фактично виконує роль інтерфейсу, що виконує функцію каталогу із формою зворотного зв'язку, але він не керує самим процесом обслуговування на станції (табл. 1.2).

З боку адміністрації СТО така заявка надходить просто як чергове повідомлення, що вимагає від менеджера ручного перенесення даних у реальний робочий графік та подальшого дзвінка клієнту для підтвердження запису. Такий підхід фактично імітує цифровізацію бізнес-процесів,

створюючи ілюзію автоматизації на клієнтській стороні, але залишаючи всю рутинну роботу з координації за персоналом.

Таблиця 1.2 – Переваги та недоліки системи AutoTech

| Переваги | Недоліки |
|---|---|
| – Об'єднання ремонту з додатковими послугами (евакуатор, шиномонтаж) в одному місці створює зручність для користувача;<br>– категоризація послуг допомагає швидше знайти потрібний тип робіт без плутанини. | – Відсутність реєстрації не дозволяє створити єдину картку клієнта, всі запити обробляються як розрізнені листи;<br>– після натискання кнопки «Відправити» клієнт не дізнається, чи прийнята його заявка в роботу адміністрацією або системою. |

Найбільш розвиненим варіантом серед розглянутих є офіційний сервіс VIDI [6]. Його головна відмінність полягає в тому, що тут є реєстрація та повноцінний особистий кабінет. Завдяки цьому система запам'ятовує клієнта та дані його автомобіля. Через кабінет можна самостійно записуватися, змінювати час або скасувати заявки в будь-який час доби. Також тут є зручний інструмент для попереднього підрахунку вартості робіт (рис. 1.4).

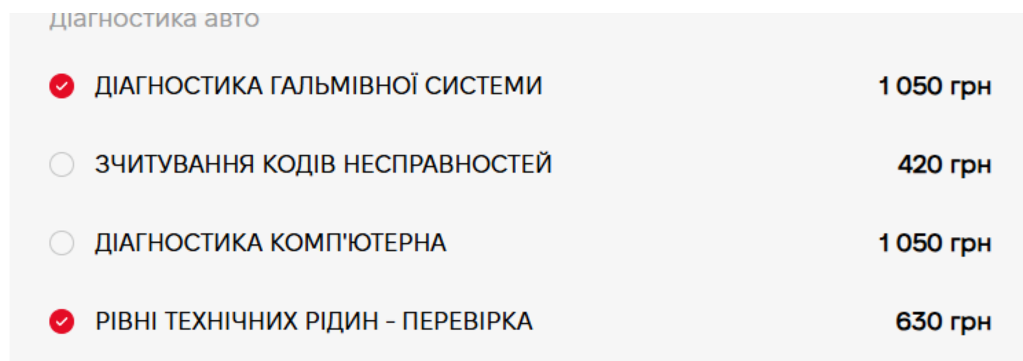


Рисунок 1.4 – Загальний функціонал вибір послуг сервісу VIDI [6]

Аналіз процесу запису показує високий рівень автоматизації цієї платформи. Клієнт обирає послуги та ініціює запис, на сервері створюється

заявка, яка одразу обробляється алгоритмом планування. З технічної точки зору VIDI реалізує більш ефективний підхід до управління ресурсами СТО. Алгоритм автоматично аналізує вільні часові інтервали майстрів, їхню відповідність типу робіт та зайнятість ремонтних постів. Планування орієнтоване на нормативну тривалість робіт та завантаженість персоналу, що дозволяє формувати максимально ефективний графік роботи.

Завдяки такій автоматизації процес призначення відбувається системно, що суттєво мінімізує ручну роботу менеджерів та прискорює обробку заявок (рис. 1.5).

Рисунок 1.5 – Загальний функціонал бронювання сервісу VIDI [6]

Одним із недоліків системи є відсутність гнучкого підходу до нетипових ситуацій, з якими щоденно стикаються автосервіси. В системі  можна обрати

лише ті послуги, що чітко визначені в каталозі. Проте дуже часто клієнт не знає точної причини несправності і йому потрібна попередня консультація майстра. У системі VIDI немає окремого інструменту для запиту на консультації. Клієнт не може просто описати симптоми поломки і чекати реакції майстра. Він змушений або вгадувати потрібну послугу з прайс-листа, або одразу телефонувати на станцію, що знову переводить процес у ручний режим.

### 1.3 Переваги запропонованої системи

Аналіз програмних рішень, розглянутих у попередньому підрозділі, виявив помітну невідповідність між клієнтською частиною та серверною обробкою даних. На ринку домінують два підходи, кожен із яких має власні обмеження. Вебсервіси, зокрема AutoDoc та AutoTech, забезпечують зручну взаємодію для користувачів, проте слабко інтегровані з внутрішніми процесами СТО. ERP-системи, наприклад Microsoft Dynamics, підтримують широкий набір функцій, але залишаються надто складними для оперативної роботи з клієнтами.

Сучасний бізнес потребує переходу від простого запису заявок до активного управління процесами і ресурсами. Часткові зміни наявних рішень проблему не вирішать – більшість платформ мають обмеження.

На основі аналізу архітектурних та функціональних обмежень існуючого ПЗ можна визначити чотири основні напрями запропонованої інформаційної системи:

1) система повинна поєднувати можливості обробки даних, характерні для ERP-рішень, зі швидкістю вебсервісів. Чіткий поділ внутрішньої логіки за принципами чистої архітектури – основа модульності. Дослідження підтверджують: ізоляція бізнес-логіки – зокрема відділення розподілу завдань від модуля оплат – мінімізує залежності між компонентами [7].

2) На відміну від закритих систем, запропоноване рішення має забезпечувати доступ до даних через стандартні API [8]. Платіжні шлюзи, хмарні сховища та сервіси підключаються до єдиного інтерфейсу без додаткової адаптації.

3) Ключова відмінність від наявних рішень – відмова від ручного призначення майстрів менеджером. Алгоритм автоматично аналізує навантаження фахівців, їхню кваліфікацію та нормативну тривалість послуг. Клієнт отримує вільний час запису після автоматичної перевірки графіка на накладки [9].

4) Особистий профіль втрачає сенс, якщо підтвердження запису все одно вимагає ручного дзвінка. Статуси замовлення змінюються автоматично на основі алгоритмічних правил або дій користувача. Зміни відображаються в реальному часі без перезавантаження сторінки.

Визначені напрями формують основу проєктування системи. Вони свідчать про те, що розробка ефективної системи автосервісу потребує аналізу сучасних методів проєктування архітектури, алгоритмів складання розкладів та інструментальних засобів, що дозволить обґрунтувати вибір конкретних технологій для реалізації поставлених завдань у наступному розділі.

## Висновки до розділу 1

Проведений аналіз існуючих рішень у сфері автосервісу показав наявність суттєвих обмежень, пов’язаних з їхніми архітектурними особливостями. Аналіз базових клієнтських сервісів AutoDoc, AutoTech показав, що їхній інтерфейс є недостатньо інформативним для користувача, а процес підтвердження запису потребує ручного дзвінка менеджеру. Комплексні ERP-системи забезпечують якісний внутрішній облік, проте залишаються складними у використанні та мають обмеження для інтеграції. Навіть найбільш просунуті сервіси VIDI, що реалізували алгоритми автоматичного призначення майстрів, мають проблеми з плануванням і не

завжди підтримують обробки нетипових заявок, через що клієнтам доводиться звертатися телефоном.

Визначено, що ключовими недоліками аналогічних систем є відсутність гнучкого автоматичного розподілу завдань між майстрами з урахуванням їхнього навантаження та кваліфікації, обмежені можливості інтеграції з іншими сервісами, а також неповна підтримка життєвого циклу замовлення.

У межах роботи обґрунтовано переваги розроблюваної системи, які полягають у підвищенні рівня автоматизації процесів обслуговування, зменшенні залежності від ручних операцій та покращенні зручності взаємодії користувачів із системою. Запропоновані підходи дозволяють забезпечити більш точне планування робіт, ефективний розподіл ресурсів та замовленнями, що в цілому сприяє підвищенню якості обслуговування клієнтів автосервісу.

# 2 ОГЛЯД ІНСТРУМЕНТАРІЮ, МОДЕЛЕЙ ТА ВИМОГ ДО ПРОЄКТУВАННЯ СИСТЕМИ

Другий розділ розглядає інструментарій та методології, придатні для реалізації поставлених завдань. Аналіз наукових публікацій охоплює актуальні архітектурні рішення, алгоритми планування ресурсів та протоколи комунікації. Виявлені переваги та обмеження оцінюються з урахуванням особливостей предметної області. Специфікація вимог до ПЗ [10], сформована за результатами аналізу, задає основу для подальших етапів розробки.

## 2.1 Аналіз та обґрунтування вибору методів і моделей проєктування системи

Проєктування сучасної веборієнтованої системи автосервісу потребує вибору методів та моделей, які б забезпечили баланс між швидкістю розробки, надійністю та можливістю подальшого масштабування. Обрані рішення мають враховувати недоліки існуючих аналогів, проаналізованих у першому розділі. Першим і найважливішим етапом є вибір архітектурного підходу.

Монолітна архітектура приваблює простотою на старті. Проте з часом тісний зв'язок компонентів перетворює підтримку на складний процес. Мікросервісна альтернатива потребує складної інфраструктури та значних ресурсів, що є надмірним для масштабів типового автосервісу (рис. 2.1). Дослідження [11] доводить: постійний обмін даними між безліччю дрібних сервісів генерує зайве навантаження. Проєкти, що не обслуговують тисячі користувачів одночасно, від такої декомпозиції лише втрачають у швидкості.

Оптимальним рішенням є використання архітектури модульного моноліту. Як зазначають дослідники [12], цей підхід поєднує переваги обох підходів: система розгортається як єдиний застосунок, що спрощує розробку на початкових етапах, але внутрішньо поділяється на ізольовані модулі. Це

дозволяє змінювати або масштабувати окремий функціонал без впливу на інші компоненти системи [7].

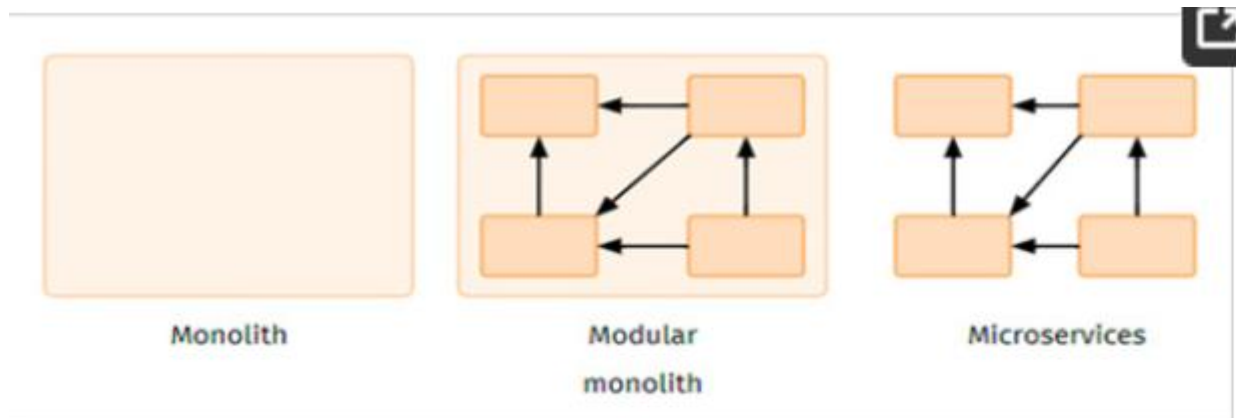


Рисунок 2.1 – Архітектурні підходи до організації серверної частини інформаційних систем [12]

Якість коду та зручність підтримки підвищуються за допомогою моделі Clean Architecture. Побудова програмного комплексу за принципом чистої архітектури повністю ізолює бізнес-логіку від інфраструктурних деталей – баз даних чи вебфреймворків [13]. Компоненти системи втрачають жорстку взаємозалежність і можуть змінюватися незалежно один від одного. Мінімізація ризиків під час модифікації коду відповідає сучасним вимогам до управління життєвим циклом ПЗ [14].

Автоматизоване планування завантаженості фахівців становить одну з головних задач. Формування запису вимагає аналізу графіка роботи, кваліфікації персоналу та нормативних часових показників. Розв'язання проблеми базується на моделі складання розкладу у багаторесурсному середовищі. Робота [9] описує систему призначення зустрічей за форматом multi-doctor/multi-services. Наведена концепція ідеально адаптується до реалій СТО. Алгоритм зіставляє вільні інтервали часу спеціалістів із тривалістю обраних клієнтом процедур. Відвідувач отримує виключно актуальні пропозиції. Накладки у розкладі усуваються автоматично, а ефективність використання робочого часу зростає [15].

Для організації взаємодії між клієнтською частиною та сервером необхідно обрати протокол обміну даними. Сучасні дослідження [8] проводять порівняльний аналіз продуктивності RESTful API та gRPC (табл. 2.1). Хоча gRPC демонструє вищу швидкість завдяки бінарному формату та підтримці потокової передачі, його використання доцільне переважно для внутрішньої комунікації між мікросервісами.

Для розроблюваної системи автосервісу важливо забезпечити роботу динамічного вебклієнта та безшовної інтеграції із зовнішніми сервісами. Згідно з [8], RESTful API добре підтримується браузерами та використовує формат JSON, що є стандартом для сучасних клієнтських фреймворків.

Таблиця 2.1 – Порівняльний аналіз протоколів взаємодії gRPC та RESTful API

| **Критерій порівняння** | **gRPC** | **RESTful API** |
|---|---|---|
| Формат даних | Protobuf (бінарний формат). | JSON, XML (текстові формати). |
| Валідація даних | Автоматична валідація повідомлень. | Потрібна додаткова валідація (наприклад, JSON Schema). |
| Патерн взаємодії | Підтримка різних режимів (унінарний, серверний/клієнтський потоковий, двонаправлений). | Класична модель (Request-Response). |
| Генерація коду | Нативна підтримка генерації клієнтського коду для багатьох мов. | Відсутня нативна підтримка (потребує зовнішніх інструментів, наприклад, OpenAPI/Swagger). |

Кінець таблиці 2.1

| Кросбраузерна сумісність | Потребує додаткового рівня (gRPC-Web) або проксі-сервера. | Вбудована в усі браузери. |
|---|---|---|
| Основне застосування | Внутрішня комунікація між мікросервісами. | Публічні API, простота інтеграції з зовнішніми сервісами. |

Сформовані архітектурні принципи формують технологічний стек. Серверну частину побудовано на платформі ASP.NET Core та мові C#. Фреймворк входить до числа найпродуктивніших рішень для вебзастосунків, нативно підтримуючи концепцію чистої архітектури [13] завдяки вбудованому механізму впровадження залежностей. Строга типізація C# сприяє ефективній реалізації модульного моноліту [12]. Бізнес-логіка чітко ізолюється від інфраструктури. Фронтенд створюється на базі бібліотеки React. Компонентний підхід радикально спрощує розробку інтерфейсу та впровадження елементів.

Роль сховища даних відведено реляційній СКБД MySQL. Гарантія жорсткої цілісності транзакцій ACID є критичною для платформ із фінансовими операціями. Крос-платформність ASP.NET Core у поєднанні з MySQL відкриває можливість розгортання на економічновигідних Linux-серверах. Безпека конфіденційної інформації також є пріоритетом. Впровадження сучасних механізмів захисту в MySQL ефективно блокує несанкціонований доступ [16]. Продуктивність при цьому залишається високою, що життєво важливо для обробки фінансових та персональних даних.

Мостом між бекендом та базою даних виступає сучасний ORM-фреймворк. Інструмент відокремлює бізнес-логіку від SQL-запитів, дотримуючись принципів чистої архітектури [11]. Реалізація модульного

моноліта передбачає делегування складних задач хмарним сервісам через REST API.

Основний застосунок розвантажується, а його надійність зростає. Безпечна обробка платежів інтегрована через платіжний шлюз LiqPay. Процес авторизації спирається на Google Authentication. Медіаконтент обробляється хмарним сервісом Cloudinary. Оптимізація, стиснення та швидка доставка зображень забезпечуються мережею Content Delivery Network.

## 2.2 Специфікація вимог до програмного забезпечення

## 1 Призначення та межі проєкту

### 1.1 Призначення системи

Інформаційна система автосервісу призначена для автоматизації бізнес-процесів станції технічного обслуговування. Система усуває потребу в ручному плануванні навантаження майстрів, підвищує прозорість процесів ремонту для клієнтів та забезпечує збереження повної історії обслуговування автомобілів.

### 1.2 Погодження, ухвалені в програмній документації

– система розроблена з урахуванням вимог до чистої архітектури та архітектури модульного моноліта, що забезпечує незалежність бізнес-логіки від баз даних та інфраструктури;

– затверджено використання сучасного вебфреймворку React та REST API для забезпечення адаптивності та швидкої взаємодії;

– обрано підхід делегування складних функцій, таких як зберігання медіа, обробка платежів та авторизація, зовнішнім спеціалізованим сервісам через API.

### 1.3 Межі проєкту

– проєкт охоплює розробку серверної та клієнтської частини, але межі проєкту виключають розробку нативних мобільних застосунків для iOS та Android;

– не передбачається апаратна інтеграція з діагностичним обладнанням автомобілів OBD-II сканерами;

– не передбачено взаємодії з державними реєстрами, страховими системами або сторонніми ERP-платформами.

## 2 Загальний опис

### 2.1 Сфера застосування

Система застосовується для організації взаємодії між клієнтами та працівниками автосервісу. Вона забезпечує запис на обслуговування, контроль статусу виконання ремонтних робіт, а також проведення онлайн-консультацій між клієнтом і майстром. Додатково реалізовано підтримку керування замовленнями та інформацією про послуги з боку адміністрації. Рішення орієнтоване на клієнтів автосервісу та працівників, які беруть участь у процесі обслуговування автомобілів.

### 2.2 Характеристики користувачів

– клієнти: власники автомобілів, які можуть переглядати перелік доступних послуг автосервісу, створювати запити на обслуговування, записуватися на ремонт, завантажувати фото або інші матеріали для уточнення проблеми, а також отримувати онлайн-консультації та оплачувати замовлення через систему;

– майстри: отримують призначені замовлення, оновлюють їх статус у процесі виконання, надають консультації клієнтам та формують рекомендації щодо подальшого обслуговування автомобіля;

– адміністратори: здійснюють загальне управління системою, включаючи ведення переліку послуг, керування прайс-листом, облік та розподіл майстрів, а також вирішення організаційних питань і контроль роботи сервісу.

## 2.3 Загальна структура і склад системи

– серверна частина: API, реалізоване на базі ASP.NET Core, яке забезпечує взаємодію з базою даних MySQL через ORM-рівень доступу до даних;

– клієнтська частина: багатосторінковий інтерфейс на базі React, що забезпечує динамічне оновлення даних без повного перезавантаження сторінок. Система включає окремі панелі для різних ролей користувачів, а також функціонал перегляду та управління послугами;

– інтеграція з хмарними сервісами Cloudinary для роботи з медіа, сервісом LiqPay для проведення платежів та Google Authentication для авторизації;

– адміністративна панель: окремий інтерфейс управління системою, який може бути реалізований як розширення функціоналу в подальших версіях проєкту;

– панель майстра: окремий інтерфейс, призначений для роботи майстрів автосервісу, що забезпечує перегляд і опрацювання замовлень, оновлення їх статусу, доступ до інформації про послуги та взаємодію з клієнтськими заявками в межах системи.

## 2.4 Загальні обмеження

– функціонування системи можливе лише за умови стабільного підключення до мережі Інтернет;

– швидкість доставки сповіщень та оновлення даних може змінюватися залежно від поточного навантаження системи та кількості активних користувачів;

– продуктивність системи та кількість одночасно активних користувачів залежать від конфігурації серверної інфраструктури та рівня навантаження;

– робота з медіафайлами обмежується політиками зовнішніх хмарних сервісів.

# 3 Функції системи

## 3.1 Реєстрація та авторизація

### 3.1.1 Опис функції

Дає змогу користувачам створювати обліковий запис та авторизуватися через локальну форму або за допомогою аутентифікації через Google.

### 3.1.2 Вхідна і вихідна інформація

– вхідна: email, пароль (для локальної реєстрації), токен Google (для Google auth);

– вихідна: JWT-токен, базова інформація про користувача (ID, email, ім'я, роль).

### 3.1.3 Функціональні вимоги

– система повинна зберігати паролі у захищеному вигляді;

– система повинна надавати засоби для підтримки сесії авторизованого користувача.

## 3.2 Управління даними профілю та автомобілем

### 3.2.1 Опис функції

Дозволяє авторизованому клієнту переглядати дані свого профілю та редагувати персональні дані автомобіля.

### 3.2.2 Вхідна і вихідна інформація

– вхідна: ідентифікатор, дані для оновлення (ім'я, телефон, дані авто, фото авто);

– вихідна: оновлені дані профілю, повідомлення про помилки.

### 3.2.3 Функціональні вимоги

– система повинна відхиляти введення некоректних даних та повідомляти про помилки;

– система повинна зберігати та відображати прив'язані до профілю медіафайли автомобіля;

– система повинна дозволяти перегляд профілю лише власнику.

## 3.3 Створення запису

### 3.3.1 Опис функції

Дає змогу клієнту створювати запис на обслуговування автомобіля з автоматичним підбором вільного часу та розподілом навантаження між майстрами.

### 3.3.2 Вхідна і вихідна інформація

– вхідна: ідентифікатори послуг, бажаний час, дані клієнта;

– вихідна: ідентифікатор створеного замовлення, перелік пов’язаних послуг, призначені майстри, час виконання.

### 3.3.3 Функціональні вимоги

– система повинна використовувати механізм розподілу навантаження між майстрами на основі їх доступності та графіка роботи;

– система повинна формувати єдине замовлення, що об’єднує декілька послуг;

– система повинна забезпечувати контроль переходів статусів замовлення;

## 3.4 Онлайн-консультація

### 3.4.1 Опис функції

Дає змогу клієнту створювати запит на попередню діагностику автомобіля з можливістю додавання опису проблеми та мультимедійних матеріалів.

### 3.4.2 Вхідна та вихідна інформація

– вхідна: опис проблеми, фото та відео матеріали, бажані дата і час;

– вихідна: статус створеної консультації, підтвердження її реєстрації.

### 3.4.3 Функціональні вимоги

– система повинна обмежувати створення лише однієї активної консультації протягом визначеного періоду;

– система повинна дозволяти завантажувати медіафайли до консультації та зберігати їх протягом життя замовлення;

– система повинна забезпечувати запис консультації з прив’язкою до користувача.

## 3.5 Перегляд та управління графіком майстра

### 3.5.1 Опис функції

Дає змогу майстру переглядати свій персональний графік роботи, включаючи заплановані записи на обслуговування автомобілів, а також керувати статусами виконання робіт.

### 3.5.2 Вхідна та вихідна інформація

– вхідна: ідентифікатор майстра, період перегляду графіка;

– вихідна: розклад роботи майстра, список запланованих записів, поточні статуси виконання робіт.

### 3.5.3 Функціональні вимоги

– система повинна забезпечувати відображення персонального графіка роботи майстра;

– система повинна відображати всі заплановані записи, закріплені за майстром;

– система повинна підтримувати зміну статусів виконання робіт (початок роботи, завершення роботи, скасування);

– система повинна виключати можливість некоректної зміни статусів поза дозволеною логікою процесу.

## 3.6 Формування та надсилання рекомендацій клієнту

### 3.6.1 Опис функції

Дає змогу майстру формувати список рекомендацій для клієнта, а клієнту – переглядати їх та приймати рішення щодо виконання.

### 3.6.2 Вхідна і вихідна інформація

– вхідна: ідентифікатор поточного замовлення, перелік обраних послуг від клієнта;

– вихідна: оновлений статус рекомендацій, ідентифікатори новостворених додаткових замовлень на основі прийнятих рекомендацій.

#### 3.6.3 Функціональні вимоги

– система повинна підтримувати режим попереднього перегляду до зберігання даних;

– система повинна атомарно створювати нові замовлення на основі прийнятих рекомендацій.

### 3.7 Проведення оплати

#### 3.7.1 Опис функції

Дає змогу користувачу ініціювати оплату послуг через зовнішню платіжну систему.

#### 3.7.2 Вхідна і вихідна інформація

– вхідна: ідентифікатори замовлення або консультації;

– вихідна: статус транзакції, результат проведення оплати.

#### 3.7.3 Функціональні вимоги

– система повинна перевіряти можливість оплати відповідно до статусу замовлення.

### 3.8 Адміністрування системи

#### 3.8.1 Опис функції

Дає змогу адміністратору здійснювати управління послугами системи, включаючи створення, редагування та видалення інформації про послуги автосервісу.

#### 3.8.2 Вхідна і вихідна інформація

– вхідна: дані послуги (назва, опис, ціна, тривалість), ідентифікатор послуги для редагування або видалення;

– вихідна: оновлений список послуг, підтвердження виконаних операцій.

#### 3.8.3 Функціональні вимоги

– система повинна забезпечувати можливість створення нових послуг;

– система повинна забезпечувати редагування існуючих послуг;

– система повинна забезпечувати видалення послуг із системи;

– система повинна забезпечувати актуальне відображення змін у каталозі послуг.

# 4 Вимоги до інформаційного забезпечення

## 4.1 Джерела і зміст вхідної інформації

– персональні дані користувачів (ПІБ, телефон, email, пароль) вводяться під час реєстрації та редагування профілю;

– дані про автомобіль (марка, модель, рік випуску, фото) надаються клієнтом та можуть оновлюватися;

– інформація про записи на обслуговування формується клієнтами або майстрами в межах системи;

– дані про виконані роботи та статуси замовлень оновлюються майстрами;

– інформація про оплату надходить із зовнішнього платіжного сервісу;

– дані консультацій формуються майстрами на основі звернень клієнтів.

## 4.2 Нормативно-довідкова інформація

– внутрішні правила роботи автосервісу;

– політика обробки персональних даних;

– платіжні стандарти (ISO 20022).

## 4.3 Вимоги до зберігання та обробки даних

– дані зберігаються у реляційній базі даних MySQL;

– конфіденційна інформація підлягає шифруванню;

– всі критичні операції виконуються у транзакційному режимі.

# 5 Вимоги до технічного забезпечення

– сучасні браузери на ПК та мобільних пристроях;

– сервери загального призначення з можливістю масштабування;

– інтеграція із зовнішніми сервісами.

# 6 Вимоги до програмного забезпечення

## 6.1 Вимоги до зберігання та обробки даних

– клієнт-серверна архітектура;
– SPA для клієнтської частини;
– REST API для взаємодії між клієнтом і сервером;
– модульна структура системи.

## 6.2 Системне програмне забезпечення

– операційна система серверної частини: Linux (Ubuntu Server);
– вебсервер: Nginx.

## 6.3 Мережне програмне забезпечення

– всі запити виконуються через HTTPS;

## 6.4 Програмне забезпечення ведення інформаційної бази

– СКБД: MySQL;
– зберігання медіафайлів здійснюється із використанням хмарного сервісу Cloudinary.

## 6.5 Мови та технології розробки

– frontend: React, JavaScript, HTML, CSS;
– backend: ASP.NET Core;
– база даних: MySQL;
– інструменти: Docker (за потреби розгортання).

# 7 Вимоги до зовнішніх інтерфейсів

## 7.1 Інтерфейс користувача

– адаптивний вебінтерфейс для різних роздільних здатностей;
– розмежування доступу за ролями (клієнт, майстер, адміністратор);
– інтерфейс повинен забезпечувати зручну навігацію між функціональними модулями.

## 7.2 Апаратний інтерфейс

– спеціального обладнання не потребується;
– підтримується робота на стандартних пристроях користувача.

### 7.3 Програмний інтерфейс

– інтеграція з платіжним сервісом LiqPay через API для ініціювання та обробки платіжних транзакцій;

– система повинна підтримувати автентифікацію користувачів з використанням протоколу OAuth 2.0;

– інтеграція з хмарним сервісом Cloudinary через API для завантаження, зберігання та отримання медіафайлів.

### 7.4 Комунікаційний протокол

– всі запити здійснюються через HTTPS;

– обмін даними реалізується у форматі REST API-запитів.

## 8 Властивості програмного забезпечення

### 8.1 Доступність

Система повинна забезпечувати стабільний доступ користувачів до основних функцій вебзастосунку в будь-який час за наявності інтернет-з’єднання. Допускається короткочасна недоступність системи під час технічного обслуговування або оновлення.

### 8.2 Супроводжуваність

Програмна система повинна мати модульну структуру, що забезпечує можливість внесення змін, розширення функціоналу та виправлення помилок без порушення роботи інших компонентів. Код повинен бути структурованим і придатним до подальшого супроводу та розвитку.

### 8.3 Переносимість

Система повинна підтримувати можливість розгортання як у локальному середовищі розробки, так і на серверній інфраструктурі без суттєвих змін у кодовій базі. Це забезпечує гнучкість у використанні різних платформ для запуску застосунку.

### 8.4 Продуктивність

– система повинна забезпечувати обробку запитів користувачів із часом відповіді не більше 2 секунд для 95% запитів;

– система повинна підтримувати одночасну роботу не менше 50 активних користувачів.

При збільшенні навантаження до зазначеного рівня система повинна зберігати час відповіді в межах встановлених значень.

### 8.5 Надійність

Система повинна забезпечувати коректну роботу навіть у разі часткових збоїв окремих модулів. Дані повинні зберігатися без втрат, а критичні операції виконуватися з використанням механізмів транзакцій та резервного копіювання.

### 8.6 Безпека

Система повинна забезпечувати захист даних користувачів шляхом використання автентифікації та авторизації, а також обмеження доступу до функцій відповідно до ролей. Передбачено захист від типових вебзагроз (SQL-ін’єкції, XSS, CSRF) та використання захищеного з’єднання HTTPS для передачі даних.

## 9 Інші вимоги

– система повинна забезпечувати можливість подальшої інтеграції з додатковими зовнішніми сервісами;

– система повинна бути орієнтована на подальший розвиток та розширення функціональних можливостей без порушення існуючої архітектури.

## Висновки до розділу 2

Проведений аналіз дозволив сформувати цілісне уявлення про підходи до проєктування веборієнтованої інформаційної системи автосервісу та визначити ключові принципи її побудови. У процесі аналізу узагальнено сучасні методи організації архітектури програмних систем, підходи до вибору технологічного стеку та принципи взаємодії між компонентами, що дало змогу

сформувати обґрунтоване бачення структури майбутнього програмного рішення.

Окрему увагу приділено вибору моделей і технологій реалізації, які забезпечують баланс між гнучкістю, продуктивністю, надійністю та можливістю подальшого розширення функціональних можливостей системи. Виконано порівняння різних архітектурних підходів, за результатами якого обґрунтовано доцільність використання модульного моноліту як архітектурної основи програмного забезпечення. Також визначено переваги застосування REST API для обміну даними між клієнтською та серверною частинами системи та принципів чистої архітектури для відокремлення бізнес-логіки від інфраструктурних компонентів.

Сформована специфікація вимог до програмного забезпечення узагальнює результати аналізу та визначає основні функціональні й нефункціональні характеристики системи, забезпечуючи єдине розуміння її цілей і обмежень та слугуючи базою для подальшого проєктування.

# 3 МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 UML-моделювання системи

Розробка програмного коду завжди потребує попереднього проєктування архітектури. Уніфікована мова моделювання UML є стандартним засобом опису програмних систем, що дозволяє графічно відображати їхню структуру, функціональну поведінку та взаємодію між окремими компонентами. Застосування моделювання дозволяє перетворити нечіткі бізнес-вимоги на чітку формалізовану модель. Моделювання грає роль моста між абстрактною ідеєю та її програмним втіленням, мінімізуючи ризики логічних помилок ще до написання першого рядка коду.

### 3.1.1 Діаграма прецедентів

Функціональне моделювання розпочинається з діаграми варіантів. Вона презентує застосунок у вигляді «чорної скриньки», фокусуючи увагу на зовнішніх акторах та доступному їм інтерфейсі. У системі визначено чотири ролі: гість, клієнт, майстер та адміністратор (рис. 3.1).

Фундаментом для більшості операцій виступає прецедент «Авторизація». Залежність "include" жорстко прив'язує до нього всі дії, що передбачають модифікацію даних. Невдала спроба входу автоматично понижує доступ до статусу гостя.

Клієнтська взаємодія обертається навколо формування замовлень. Базовий сценарій розширюється залежністю "include" за рахунок автоматичного підбору майстрів. Алгоритм одразу розраховує доступних фахівців під час комбінування різних послуг. Альтернативні шляхи реалізовано через зв'язок "extend". Наприклад, майстром ініційовано прецедент «Консультація», якщо виникає потреба запропонувати клієнту додаткові роботи після огляду авто. Після отримання розширеного переліку

з'являється умовний сценарій «Оплата замовлення». Фінансова транзакція стає доступною лише після фінального узгодження списку робіт.

Діяльність майстра тісно пов'язана з автоматизованим розкладом. Оновити статус виконання можна, звернувшись до прецеденту «Перегляд індивідуального графіка». Подібне архітектурне рішення гарантує цілісність обробки кожного запису. Спеціаліст спочатку знаходить свою задачу, і лише потім змінює її стан.

Керування базовими даними сервісу лежить на адміністраторі. Редагування послуг чи призначення спеціалізацій обов'язково вимагають попереднього перегляду поточного масиву інформації. Безпечний доступ до цих інструментів також захищений подвійною залежністю від прецедента «Авторизація», повністю блокуючи їх для неавтентифікованих користувачів.

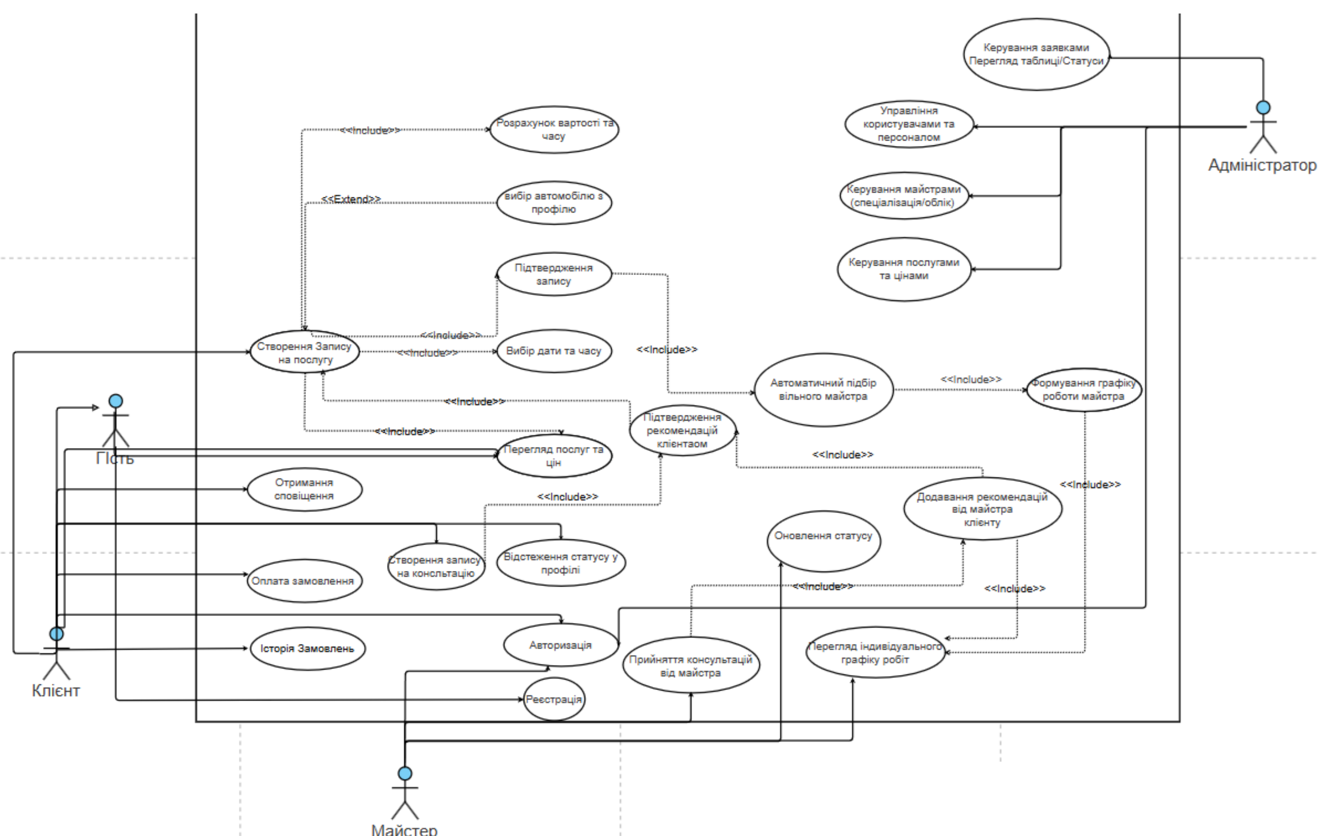


Рисунок 3.1 – Діаграма  використання системи автосервісу

Побудована діаграма прецедентів дозволила визначити основні функціональні можливості системи, ролі користувачів та взаємозв’язки між

окремими сценаріями роботи. Вона відображає загальну логіку взаємодії клієнта, майстра та адміністратора із системою автосервісу, а також фіксує залежності між ключовими прецедентами.

Отримані результати стали основою для подальшого проєктування внутрішньої структури системи, архітектури та структури бази даних.

### 3.1.2 Визначення сутностей та їх зв'язків системи автосервісу

Перенести бізнес-процеси автосервісу на рівень бази даних без попереднього моделювання неможливо – система перетвориться на хаос із дубльованими записами. Щоб цього уникнути, побудовано ER-модель (рис. 3.2). Вона показує, з яких саме «цеглинок» складатимуться дані і як ці блоки спираються один на одного.

За основу взято таблиця "Users", яка зосереджена виключно на аутентифікаційних даних. Відомості про транспортні засоби зберігаються ізольовано у сутності "User_profiles". Між двома цими об'єктами встановлено зв'язок один-до-одного.

Роль майстра вимагає окремої уваги через специфіку роботи СТО. Один фахівець може володіти кількома напрямками одночасно. Моделювання цього процесу вимагає зв'язку багато-до-багатьох. Щоб база не ламалася при збереженні таких перетинів, між майстрами та спеціалізаціями поставлено проміжну таблицю "Master_specialization". Вона просто фіксує, хто до чого має допуск.

Самі послуги розбито на два рівні. Загальні назви лежать у "Services", а конкретні позиції з їхньою ціною та тривалістю – у "ServiceItems". Завдяки такій ієрархії змінювати вартість окремої операції можна без ризику зачепити інші категорії.

Найскладніший вузол моделі – це замовлення. Сутність "Bookings" зберігає загальну картину: хто приходить, коли і який статус має візит. Якщо прив'язувати роботи прямо до замовлення, структура втратить гнучкість. Тому

з'явилася таблиця "Booking_services". Головна її особливість у тому, що саме тут до кожної окремої послуги прив'язується конкретний майстер. Без цього кроку автоматичний розподіл навантаження, був би неможливий технічно.

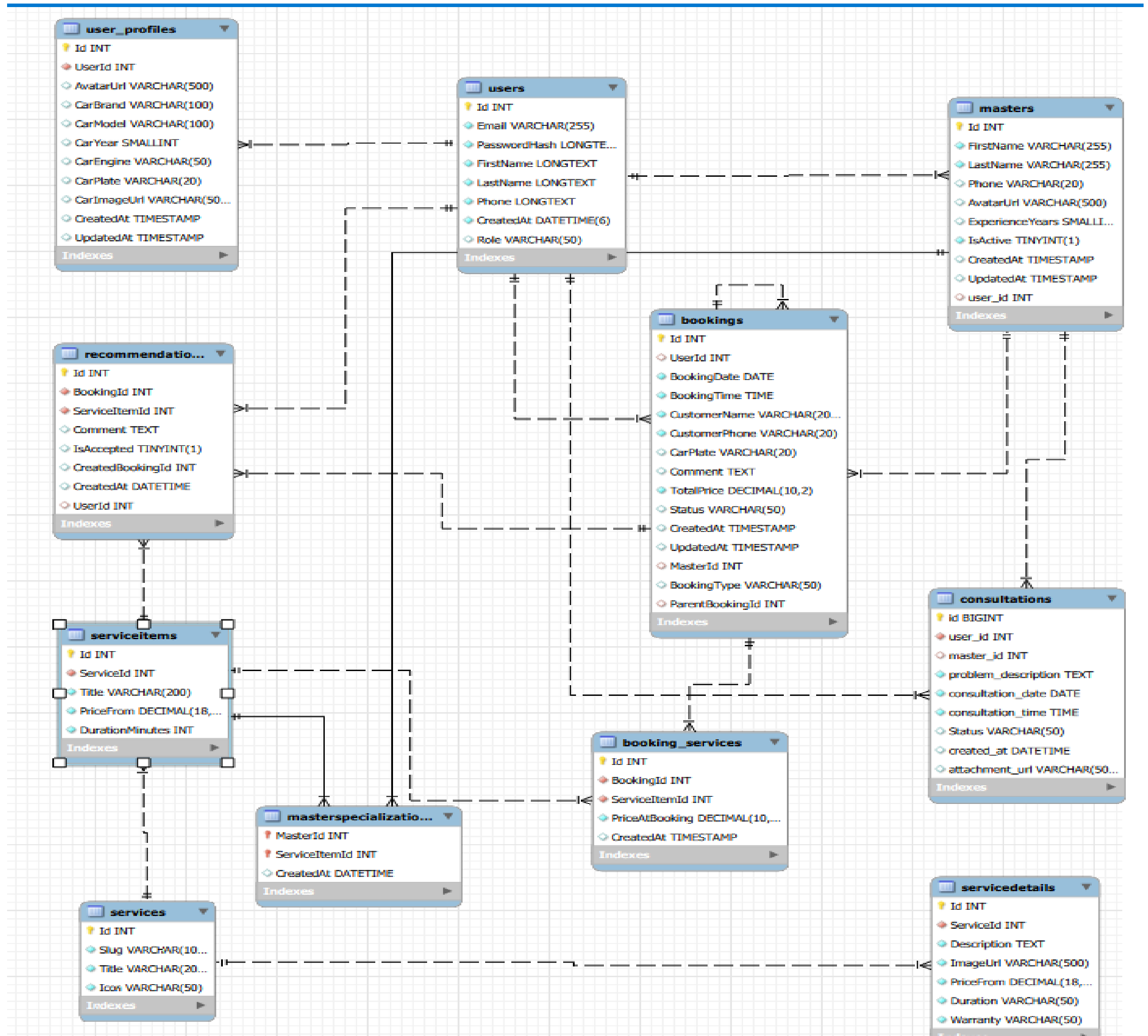


Рисунок 3.2 – ER-Діаграма

Окрема гілка моделі відповідає за діагностику. Коли клієнтові невідома причина поломки, він створює запис у "Consultations". Після огляду майстер формує перелік того, що треба зробити. Ці ініційовані фахівцем роботи передаються у "Recommendations" із посиланнями на ті ж "ServiceItems". Як тільки власник авто натискає "Погодити" у своєму кабінеті, система бере ці рекомендації та автоматично конвертує їх у класичний запис у "Bookings".

### 3.1.3 Визначення класів системи

Структурна архітектура системи формується не набором окремих сутностей, а топологією їхніх зв'язків, що фіксується на діаграмі класів (рис. 3.3). На відміну від ER-моделі, яка описує сховище, ця діаграма проектує поведінку об'єктів застосунку через типізовані відношення.

Фундаментом рольової моделі слугують зв'язки узагальнення. Базовий клас *User* передає свої властивості похідним сутностям. Запроваджений поліморфізм дозволяє звертатися до будь-якої ролі через єдиний інтерфейс батьківського компонента. Втім, специфічні методи стають доступними лише за умови приведення об'єкта до типу *Client*. Виділення класу *Guest* в окрему гілку наслідування суттєво спрощує валідацію прав доступу.

Взаємодія між рівнями ієрархії та відповідними даними реалізується через систему асоціативних зв'язків. Яскравим прикладом слугує композиція між *Client* та *UserProfile* – специфічний варіант відношення «один до одного» із жорстким контролем життєвого циклу. Існування профілю поза контекстом конкретного клієнта заблоковано на рівні логіки. Видалення власника акаунта автоматично запускає каскадне знищення пов'язаного запису. Найвищу складність має конфігурація зв'язків навколо класу *Booking.*

Рисунок 3.3 – Діаграма класів системи

У цій частині реалізовано кілька різних підходів до обробки даних:

1) направлена асоціація від *Client* до *Booking*. Метод *createbooking()* у клієнта виступає ініціатором зв’язку та ініціює створення об’єкта замовлення як контейнера для подальших послуг. Кожен екземпляр Booking зберігає зворотне посилання на свого ініціатора, що забезпечує фіксацію відповідальності за створення замовлення.

2) Агрегація між *Booking* та *Service*. Замовлення містить набір послуг, при цьому клас *Service* залишається незалежним. Такий тип зв’язку забезпечує, що зміна вартості послуги адміністратором не впливає на вже створені замовлення, оскільки в них зберігаються лише ідентифікатори послуг.

3) Асоціація між *master* та *schedule*. Майстер є власником розкладу, але з специфікою лише для читання та модифікації статусів. Саме цей зв'язок відображає технічну реалізацію попередньої діаграми прецедентів: оскільки майстер не створює записи в розкладі самостійно, клас master не містить методів типу *generateschedule(),* маючи лише методи оновлення статусів отриманих об'єктів.

### 3.1.4 Діаграма послідовності

Для детального вивчення динамічної поведінки системи в часі використовується діаграма послідовності. На відміну від статичної діаграми класів, вона показує процес обміну повідомленнями між об'єктами у рамках конкретного сценарію. Це дозволяє верифікувати те, як сервісний шар виконує складні алгоритмічні рішення, зокрема патерн «Стратегія» для автоматичного пошуку вільного майстра та кінцевий автомат для контролю зміни статусів бронювання.

Детальний процес того, як саме сервісний шар виконує алгоритм формування запису та автоматичного підбору майстрів, демонструє діаграма послідовності (рис. 3.4). На ній відображено повний ланцюжок взаємодій –

від моменту, коли клієнт надсилає запит через інтерфейс, до фіксації готового запису в базі даних. Це дає змогу побачити не лише які компоненти задіяні, а й у якій саме послідовності вони обмінюються даними та яку роль кожен із них відіграє.

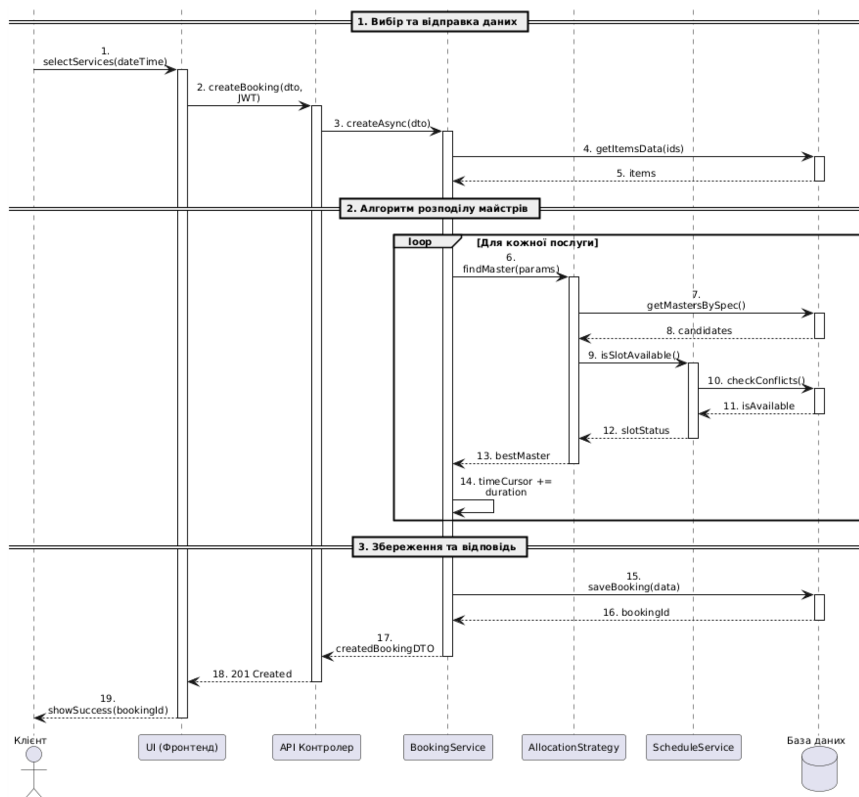


Рисунок 3.4 – Діаграма послідовності: формування запису та розподіл майстрів

Цей процес доцільно поділити на три послідовні етапи:

1) клієнт через інтерфейс ініціює перегляд послуг. Запит проходить через API-контролер безпосередньо до бази даних, після чого актуальний каталог повертається на фронтенд для відображення;

2) після того, як клієнт обирає конкретні послуги та бажаний час, інтерфейс формує об'єкт передачі даних та відправляє його на сервер разом із токеном авторизації. Сервісний шар приймає запит і завантажує повну інформацію про обрані послуги з бази;

3) оскільки в автосервісі одне замовлення може включати послуги з різних спеціалізацій, система запускає цикл обробки для кожної обраної

позиції окремо. До роботи підключається клас *AllocationStrategy*. Спочатку він запитує базу даних і отримує список майстрів, які мають відповідну кваліфікацію. Далі кожен кандидат передається до сервісу розкладу, який перевіряє наявність конфліктів у графіку на вказаний час. Якщо слот вільний, стратегія фіксує цього майстра за послугою, а сервіс бронювання зміщує внутрішній часовий маркер. Це необхідно для того, щоб пошук майстра для наступної послуги в цьому ж замовленні розпочинався з урахуванням тривалості попередньої. Після успішного проходження циклу для всіх обраних послуг, система фіксує сформований запис у базі даних та повертає клієнту підтвердження.

### 3.1.5 Проєктування діаграми розгортання та опис архітектури системи

Фінальним етапом моделювання є визначення архітектури програмного забезпечення. На етапі реалізації системи сформовано діаграму розгортання, яка відображає загальну архітектурну структуру. Відповідно до аналізу, проведеного у підрозділі 2.1, діаграма дозволяє наочно представити розподіл основних компонентів системи, зокрема клієнтської частини, серверної логіки, бази даних та зовнішніх сервісів, а також визначити характер їх взаємодії під час обробки запитів користувача (рис. 3.5). Такий підхід забезпечує розуміння принципів побудови системи, її масштабованості та стабільності роботи в умовах навантаження.

Інтерфейсний рівень реалізовано у вигляді клієнтської частини, яка виконується безпосередньо в браузері користувача на базі React-компонентів. Його основним завданням є забезпечення інтерактивної взаємодії без повторного завантаження вебсторінок. Усі сформовані дії трансформуються у HTTP-запити, які спрямовуються до вузла Nginx. Вебсервер виконує функції зворотного проксі-сервера, приймаючи на себе обслуговування статичних

ресурсів фронтенду та маршрутизуючи запити, що потребують виконання обчислень, на наступний рівень обробки.

Основою системи є прикладний сервер, побудований на технології .NET Web API. Саме тут виконується вся основна бізнес-логіка: від валідації вхідних даних до роботи алгоритмів автоматичного підбору майстрів. Для постійного зберігання інформації про клієнтів, послуги, розклад та історію бронювань підключено сервер баз даних MySQL.

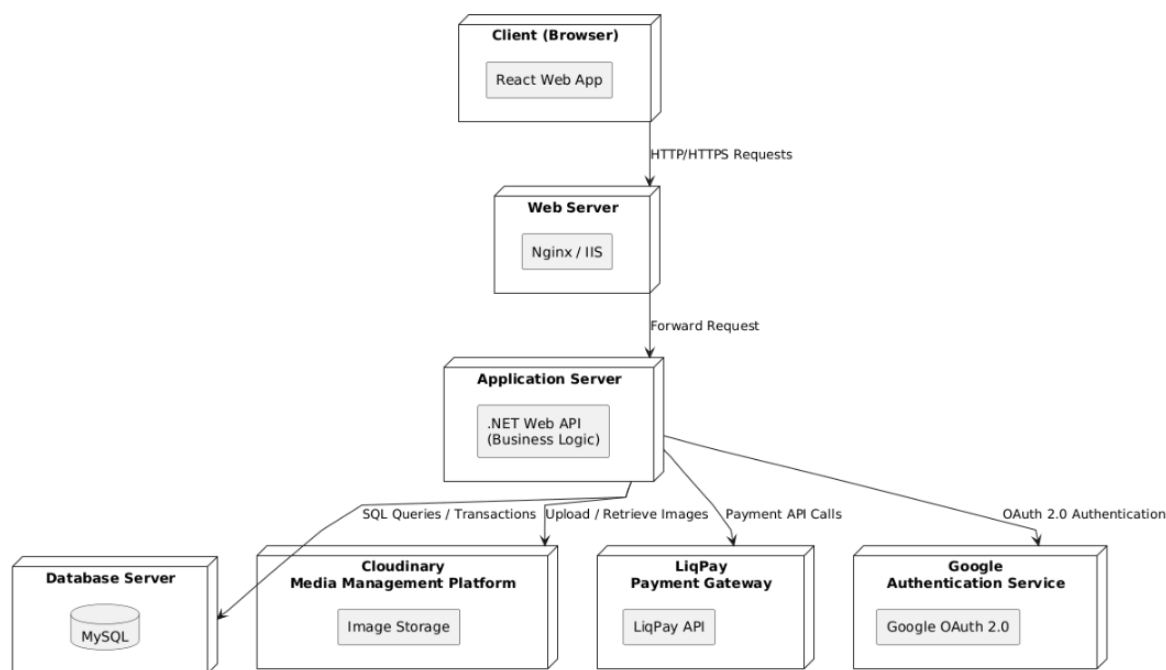


Рисунок 3.5 – Діаграма розгортання системи

Окремий сегмент інфраструктури складають сторонні сервіси, інтеграція з якими здійснюється через мережеві виклики зі сторони API:

– зберігання графічного контенту передано платформі Cloudinary, що дозволяє вивантажити медіа-файли з основного сервера;

– фінансові операції обробляються через платіжний шлюз LiqPay, що виключає пряму обробку банківських реквізитів на стороні автосервісу;

– процедура автентифікації частково передається сервісу Google Authentication, що спрощує реєстраційну процедуру та зменшує ризики, пов’язані із самостійним управлінням паролями користувачів.

Процес обробки даних у системі відбувається послідовно. Дія користувача в інтерфейсі React ініціює HTTP-запит, який через Nginx передається до .NET API. Сервер обробляє бізнес-логіку, взаємодіє з базою даних MySQL або зовнішніми сервісами та повертає результат на клієнтську частину. Така архітектура дозволяє легко масштабувати окремі компоненти системи без її зупинки.

## 3.2 Розробка макетів інтерфейсу користувача системи автосервісу

Створення інтерфейсу займає ключове місце в процесі створення системи, оскільки саме від нього залежить зручність і швидкість взаємодії користувачів із сервісом. На цьому етапі підготовлені макети основних сторінок, що дало змогу заздалегідь сформувати структуру інтерфейсів, логіку переходів та розташування функціональних елементів.

Форма запису на обслуговування (рис. 3.6) реалізує функцію комбінованого замовлення. Інтерфейс дозволяє обирати декілька послуг одночасно, переглядати їхню вартість та формувати єдиний запис.

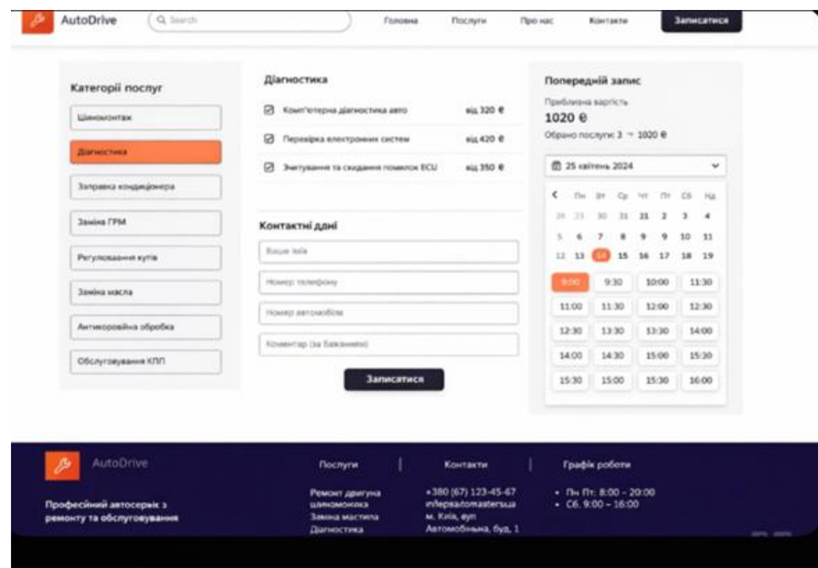


Рисунок 3.6 – Макет форми створення запису на обслуговування автомобіля

Роль центрального інформаційного вузла виконує головна сторінка системи (рис. 3.7). Тут розміщено навігаційне меню, короткий опис сервісу та

перелік основних послуг. Перехід між розділами реалізовано без зайвих вкладень, що спрощує доступ до ключових функцій і скорочує час взаємодії з інтерфейсом.

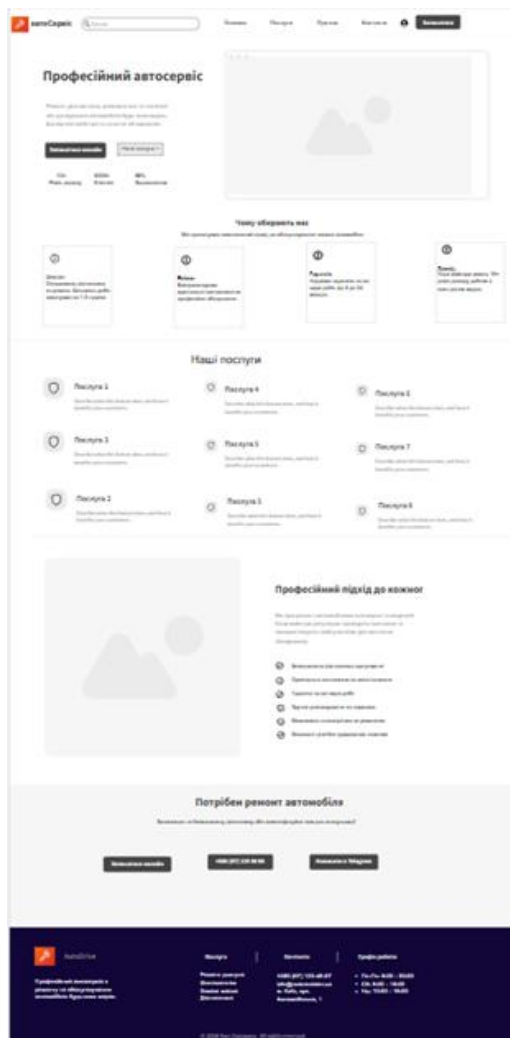


Рисунок 3.7 – Макет головної сторінки системи автосервісу

Користувач може швидко перейти до потрібного розділу, переглянути доступні категорії робіт або перейти до запису на обслуговування. Структура сторінки побудована таким чином, щоб основні дії доступні без зайвих переходів.

Окрему логіку має робочий простір майстра (рис. 3.8), який базується на автоматично сформованому розкладі. У центральній частині відображається перелік запланованих робіт із прив’язкою до часу виконання. Для кожного замовлення доступна інформація про клієнта, транспортний засіб, тривалість

та статус виконання. Додатково в інтерфейсі відображаються консультаційні звернення від клієнтів, які надходять майстру для попереднього уточнення деталей або погодження обсягу робіт.

Майстер може змінювати стан заявки, переглядати деталі, опрацьовувати консультації та фіксувати завершення робіт, однак ручне редагування розкладу не передбачене, оскільки планування здійснюється автоматично.

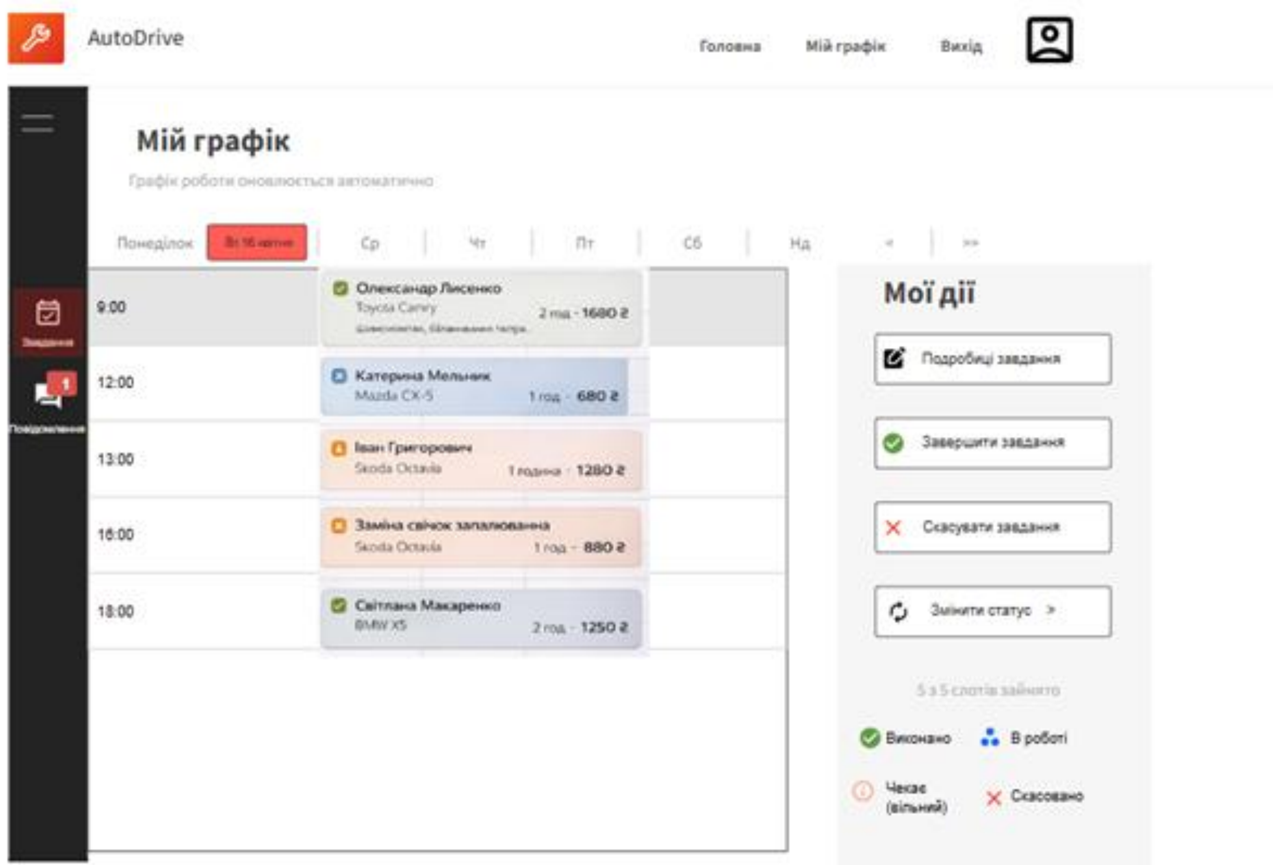


Рисунок 3.8 – Макет робочого інтерфейсу майстра з автоматично сформованим розкладом робіт

Особистий кабінет клієнта (рис. 3.9) призначений для перегляду історії обслуговування та керування активними записами. У ньому зберігаються дані про автомобіль, попередні звернення, обрані послуги та статуси виконання замовлень. Додатково відображаються дата обслуговування, вартість робіт і призначений виконавець.

Повідомлення про результати діагностики, зміни статусів або рекомендації майстра виводяться в окремому блоці, що дозволяє оперативно отримувати зворотний зв’язок без додаткових звернень до сервісу.

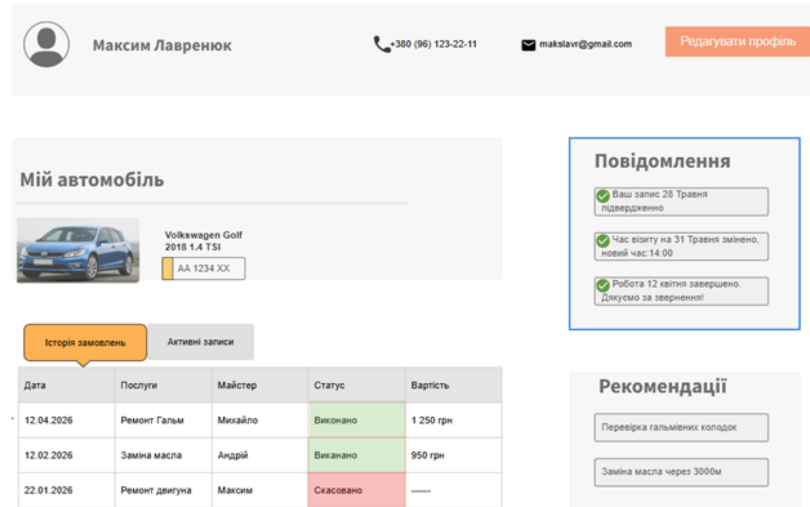


Рисунок 3.9 – Макет особистого кабінету клієнта для перегляду історії обслуговування та статусів замовлень

Адміністративна панель (рис. 3.10) призначена для контролю роботи системи та обробки заявок. Інтерфейс дозволяє переглядати всі записи на обслуговування, виконувати пошук і фільтрацію за датою, категорією послуг або конкретним майстром.

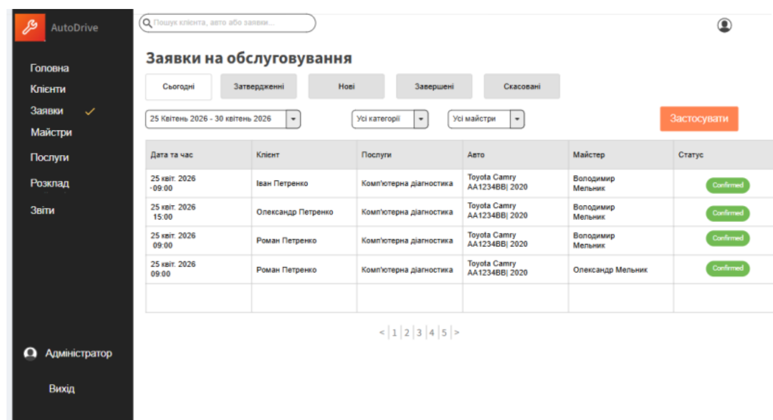


Рисунок 3.10 – Макет адміністративної панелі

Для кожної заявки відображається інформація про клієнта, транспортний засіб, виконавця та поточний стан виконання. Окремо

реалізовано функції управління довідниками послуг і спеціалізаціями працівників. Такий підхід спрощує контроль за роботою сервісу та дозволяє швидко працювати з великим обсягом замовлень.

Під час створення макетів основний акцент зроблено на простоті взаємодії та зрозумілому розташуванні елементів. Інтерфейси не перевантажені зайвими деталями. Це дозволяє користувачам швидше орієнтуватися в системі та зменшує кількість помилок під час роботи зі системою.

## Висновки до розділу 3

У третьому розділі сформовано загальну модель програмного забезпечення системи автосервісу, що охоплює основні аспекти її структури та логіки роботи. Узагальнено підходи до побудови функціональної та структурної складових системи, що дозволило визначити цілісну організацію майбутнього програмного рішення.

Окрему увагу приділено моделюванню основних компонентів системи та способів їх взаємодії. Визначено ключові елементи архітектури, які забезпечують узгоджену роботу користувацької частини, серверної логіки та обробки даних.

Сформовано уявлення про принципи реалізації основних сценаріїв використання та організацію інтерфейсу користувача, що забезпечує зв’язок між функціональними можливостями системи та способом їх застосування.

Отримані результати створюють основу для подальшої реалізації програмного забезпечення та забезпечують цілісне бачення структури та принципів функціонування.

# 4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОСЕРВІСУ

## 4.1 Розробка моделі бази даних

У межах проєкту реалізовано реляційну структуру бази даних, що відображає ключові бізнес-сутності системи автосервісу та зв'язки між ними. Фізична модель побудована з урахуванням принципів цілісності даних, нормалізації та забезпечення підтримки основних сценаріїв роботи системи: реєстрації користувачів, формування замовлень, управління послугами та розподілу навантаження між майстрами.

У таблиці 4.1 наведено ключові атрибути кожної сутності та їхню характеристику змінності. Частина полів має фіксований характер, оскільки визначається системою або бізнес-правилами. Інші атрибути є динамічними та можуть змінюватися в процесі життєвого циклу записів, що забезпечує гнучкість моделі даних.

Таблиця 4.1 – Проєктні рішення моделі бази даних

| **Сутність** | **Атрибути** | **Характеристика** |
|---|---|---|
| Users | id (int), email (varchar), passwordHash (longtext), firstName (longtext), lastName (longtext), phone (int), createdAt (datetime), role (varchar) | Поле role визначається фіксованим переліком значень (admin, client, master). Інші атрибути є змінними. |
| Services | id (int), slug (varchar), title (varchar), icon (varchar) | slug – унікальний незмінний ідентифікатор. |
| ServiceItems | id (int), serviceId (int), title (varchar), priceFrom (decimal), durationMinutes (int) | durationMinutes – базове значення для планування. |

Кінець таблиці 4.1

| Masters | id (int), firstName (varchar), lastName (varchar), phone (varchar), avatarUrl (varchar), experienceYears (smallint), isActive (tinyint), createdAt (timestamp), updatedAt (timestamp), user_id (int) | Поле isActive ініціалізується значенням за замовчуванням. Системні часові мітки створюються автоматично. Інші дані можуть змінюватися. |
|---|---|---|
| Bookings | id (int), userId (int), bookingDate (date), bookingTime (time), customerName (varchar), customerPhone (varchar), carPlate (varchar), comment (text), totalPrice (decimal), status (varchar), masterId (int), bookingType (varchar), parentBookingId (int), createdAt (timestamp), updatedAt (timestamp) | Поле status має початкове значення *pending*. Часові мітки формуються автоматично. Інші атрибути є змінними. |
| Booking Services | id (int), bookingId (int), serviceItemId (int), priceAtBooking (decimal), createdAt (timestamp) | Поле priceAtBooking фіксує історичну вартість на момент створення запису. Поле createdAt є системним і не змінюється. |

Основу структури складають базові сутності, які забезпечують функціонування системи. Сутність *Users* відповідає за зберігання даних користувачів та їхніх ролей у системі. *Bookings* відображає процес створення та обробки замовлень на обслуговування автомобіля. Каталог послуг реалізовано через сутності *Services* та *ServiceItems*, що дозволяє розділити загальні категорії послуг і конкретні операції з визначеними характеристиками

виконання. Сутність *Masters* відображає виконавців робіт та їхні професійні характеристики, що дозволяє реалізувати механізми розподілу навантаження та призначення відповідальних за виконання замовлень. Зв’язувальна таблиця *BookingServices* забезпечує багатозв’язну структуру між замовленнями та послугами, а також фіксує історичну вартість виконаних робіт, що є важливим для подальшого аналізу та звітності.

Загалом представлена структура бази даних забезпечує узгоджене зберігання інформації, підтримує цілісність зв’язків між сутностями та дозволяє ефективно реалізувати основні сценарії роботи системи автосервісу.

## 4.2 Організація моделей сутностей

Для взаємодії між реляційною базою даних MySQL та об’єктно-орієнтованою логікою .NET сформовано шар доменних моделей (рис. 4.1). На відміну від прямого використання SQL-запитів, ці класи виступають строго типізованими контейнерами даних. Вони підключаються до контексту *ApplicationContext* через властивості *DbSet<T>*, що дозволяє сервісному шару працювати із записами таблиць за допомогою LINQ-виразів, делегуючи генерацію SQL-коду ORM.

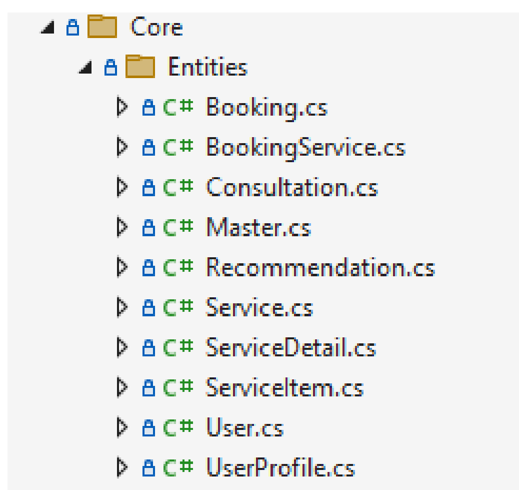


Рисунок 4.1 – Структура моделей у проєкті

Структура файлів моделей відображає принцип розділення відповідальностей, закладений у структурі бази даних. Наприклад,

авторизаційні дані винесені в окремий клас *User*, тоді як інформація про профіль користувача виділена в *UserProfile*. Таке розділення дає змогу налаштовувати навігаційні властивості таким чином, щоб додаткові дані завантажувалися лише за необхідності, що зменшує навантаження на пам'ять під час базових операцій авторизації.

Найскладнішу систему зв'язків реалізовано в групі моделей *Booking*, *BookingService*, *Consultation* та *Recommendation*, які відповідають за бізнес-логіку автосервісу. Зокрема, в моделі *Booking* використано ієрархічний зв'язок через властивість *ParentBooking*. Це рішення дозволяє зберігати загальне замовлення як батьківський запис, а окремі роботи для різних майстрів – як дочірні сутності, пов'язані через *ParentBookingId*.

Каталожні сутності, такі як *Service*, S*erviceItem* та *ServiceDetail*, містять навігаційні колекції для реалізації зв'язків «багато-до-багатьох», зокрема для визначення переліку послуг, які може виконувати майстер. Ці класи не містять бізнес-логіки та виконують роль проміжного шару для мапінгу даних між сервісною логікою та базою даних.

### 4.3 Розробка серверної архітектури та бізнес-логіки

Серверна частина побудована на основі чіткого поділу відповідальностей між рівнями системи. Контролери виконують роль точок входу для HTTP-запитів і відповідають лише за прийом даних та базову перевірку коректності. Уся прикладна логіка винесена в сервіси, які інкапсулюють роботу з даними, зовнішніми ресурсами та бізнес-правилами. Зв'язування компонентів реалізовано через інтерфейси з централізованою реєстрацією залежностей у Program.cs.

Ізоляція обробки запитів від деталей реалізації робить код зручнішим для читання. Бізнес-правила зберігаються разом. Модулі перестають залежати один від одного, а структура виглядає набагато акуратніше. Створення об'єктів відбувається автоматично. Програма сама керує їхнім життєвим циклом.

Заміна будь-якої залежності тепер не ламає сусідні процеси. Ізольовані шари набагато легше піддавати автоматизованому тестуванню. Додавання нових можливостей вимагатиме лише підключення класу.

У системі виділено окремий шар інтерфейсів, який відокремлює реалізації сервісів від їх використання. Сервіси взаємодіють не з конкретними класами, а з абстракціями. Відповідність між контролерами, сервісними інтерфейсами та їх реалізаціями наведено в таблиці 4.2.

Таблиця 4.2 – Відповідність контролерів, сервісних інтерфейсів та їх реалізацій

| **Контролери** | **Інтерфейси** | **Реалізації сервісів** |
|---|---|---|
| AdminController | IAdminServiceService, IBookingService | AdminServiceService, BookingService |
| AuthController | IUserService, ITokenService | UserService, TokenService |
| BookingsController | IBookingService | BookingService |
| ConsultationsController | IConsultationService, IFileStorageService | ConsultationService, CloudinaryStorageService |
| MastersController | IMasterService, IMasterAllocationStrategy, IScheduleService | MasterService, LeastLoadStrategy, ScheduleService, MasterAllocationService |
| ProfileController | IUserService, IFileStorageService | UserService, CloudinaryStorageService |
| RecommendController | IRecommendationService | RecommendationService |
| ServicesController | IServiceService | ServiceService |

Наприклад, IFileStorageService описує поведінку роботи з файлами, тоді як CloudinaryStorageService відповідає за збереження медіа у хмарному

сховищі (рис. 4.2). Такий підхід дозволяє змінювати спосіб зберігання файлів без впливу на інші частини системи.

```
namespace AutoServ.Core.Interfaces
{
    Ссылок: 8
    public interface IFileStorageService
    {
        Ссылок: 5
        Task<string> UploadFileAsync(Stream fileStream, string folder, string fileName);
        Ссылок: 3
        Task DeleteFileAsync(string fileUrl);
    }
}
```

Рисунок 4.2 – Один із прикладів інтерфейсів

Ключова бізнес-логіка створення замовлень реалізована у *BookingService*. При виборі кількох послуг формується послідовний план виконання, у межах якого для кожної послуги підбирається доступний майстер з урахуванням поточного часу та тривалості попередніх операцій (рис. 4.3). На кожному кроці алгоритм звертається до *IMasterAllocationStrategy*, враховуючи доступний час і тривалість попередніх операцій.

Після формування повного набору завдань запускається транзакція. Спочатку створюється основний запис замовлення, після чого додаються дочірні елементи з типом *service_task*, кожен із яких прив’язується до конкретного майстра через *MasterId* та посилається на *ParentBookingId*.

```
    var servicesList = dto.Services.Where(s => dbServices.ContainsKey(s.Id)).ToList();
    var plan = new List<(Master Master, List<BookingServiceItemDto> Services, TimeSpan StartTime)>();
    TimeSpan chainCursor = parsedTime;

    foreach (var svc in servicesList)
    {
        int duration = dbServices[svc.Id].DurationMinutes;
        var master = await _allocationStrategy.FindMasterAsync(svc.Id, currentDate, chainCursor, duration,

        if (master == null)
            return Result<BookingCreatedResultDto>.Failure($"на жаль, не вдалося знайти вільного майстра на

        plan.Add((master, new List<BookingServiceItemDto> { svc }, chainCursor));
        chainCursor = chainCursor.Add(TimeSpan.FromMinutes(duration));
    }

    return await SaveBookingPlanWithParent(plan, dto, currentDate);
}
```

Рисунок 4.3 – Формування черги виконання послуг із розподілом майстрів

Підтвердження змін виконується лише після успішного завершення всіх операцій. У разі помилки виконується відкат транзакції, що виключає появу неповних або частково збережених замовлень.

Окремий сценарій стосується перетворення рекомендацій після консультацій у реальні замовлення. У RecommendationService реалізовано метод пакетної обробки *AcceptBatchAsync*, який дозволяє обробляти одразу набір рекомендацій. Перед створенням записів виконується відбір коректних і ще не оброблених елементів. Далі, у межах однієї транзакції, формується набір пов'язаних завдань за тим самим принципом, що й у процесі бронювання.

Зміна статусів замовлень контролюється через механізм керування переходами між станами, реалізований у класі *BookingStateTransitions*. Замість перевірок, розосереджених по сервісах, допустимі переходи зібрані в одному місці у вигляді словника *_transitions* (рис. 4.4). Для кожного значення перерахування *BookingStatus* визначено перелік статусів, до яких дозволено виконати перехід.

```
Ссылок: 2
public static class BookingStateTransitions
{
    private static readonly Dictionary<BookingStatus, List<BookingStatus>> _transitions = new()
    {

        { BookingStatus.Pending, new List<BookingStatus> { BookingStatus.Confirmed, BookingStatus.InProgress, Booki

        { BookingStatus.Confirmed, new List<BookingStatus> { BookingStatus.InProgress, BookingStatus.Cancelled } },
        { BookingStatus.InProgress, new List<BookingStatus> { BookingStatus.Completed, BookingStatus.Cancelled } },
        { BookingStatus.Completed, new List<BookingStatus> { } }
    };

    Ссылок: 1
    public static bool CanTransition(BookingStatus current, BookingStatus next)
    {
        return _transitions.TryGetValue(current, out var allowedStates) && allowedStates.Contains(next);
    }
```

Рисунок 4.4 – Реалізація словника допустимих переходів між статусами у класі BookingStateTransitions

Перевірка коректності переходу виконується у методі *UpdateTaskStatusAsync* сервісу BookingService (рис. 4.5). Після отримання нового статусу із запиту система завантажує відповідне завдання з бази даних та викликає метод *CanTransition,* передаючи поточний і новий стани. Якщо запитуваний перехід відсутній у словнику дозволених переходів, метод повертає помилку через *Result.Failure*, а виклик *SaveChangesAsync* не виконується.

Після успішної перевірки статус завдання оновлюється. Якщо запис є дочірнім елементом складеного замовлення, додатково аналізуються всі пов'язані завдання. Коли всі дочірні записи отримують статус *Completed,* батьківське замовлення автоматично переводиться у стан *Completed*. Якщо хоча б одне завдання перебуває у стані *InProgress*, батьківський запис також отримує статус *InProgress*. Такий підхід забезпечує узгодженість станів між окремими роботами та загальним замовленням.

```
Ссылок: 2
public async Task<Result> UpdateTaskStatusAsync(int taskId, string newStatus)
{
    if (!Enum.TryParse<BookingStatus>(newStatus.Replace("_", ""), ignoreCase: true, out var parsedStatus))
        return Result.Failure("unknown status");

    var task = await _context.Bookings.Include(b => b.ParentBooking).FirstOrDefaultAsync(b => b.Id == taskId);
    if (task == null) return Result.Failure("task not found");

    if (!BookingStateTransitions.CanTransition(task.Status, parsedStatus))
        return Result.Failure(BookingStateTransitions.GetErrorMessage(task.Status, parsedStatus));

    task.Status = parsedStatus;

    if (task.ParentBookingId.HasValue)
    {
        var allTasks = await _context.Bookings.Where(b => b.ParentBookingId == task.ParentBookingId).ToListAsync();
        var parent = task.ParentBooking;
        if (parent != null)
        {
            bool allCompleted = allTasks.All(t => t.Status == BookingStatus.Completed);
            bool anyInProgress = allTasks.Any(t => t.Status == BookingStatus.InProgress);

            if (allCompleted) parent.Status = BookingStatus.Completed;
            else if (anyInProgress) parent.Status = BookingStatus.InProgress;
        }
    }

    await _context.SaveChangesAsync();
    return Result.Success();
}
```

Рисунок 4.5 – Перевірка допустимості переходу та синхронізація статусів у методі UpdateTaskStatusAsync

Таке рішення запобігає виконанню недопустимих переходів ще на рівні бізнес-логіки. Навіть якщо користувач спробує надіслати модифікований HTTP-запит із некоректним статусом, сервіс відхилить операцію до формування SQL-запиту та внесення змін у базу даних.

Доступ до API обмежується атрибутами авторизації. Контролер BookingsController вимагає наявності JWT-токена, а окремі операції додатково обмежені ролями. Ідентифікатор користувача отримується з токена через ClaimsPrincipalExtensions, що виключає можливість підміни ідентифікатора в тілі запиту.

## 4.4 Тестування програмного забезпечення

Тестування програмного забезпечення є важливим етапом життєвого циклу розробки, що дозволяє верифікувати відповідність реалізованого функціоналу початковим специфікаціям вимог. У контексті системи особливої уваги потребує перевірка не лише базових операцій вводу-виведення даних, але й складної бізнес-логіки, зокрема алгоритмів автоматичного планування ресурсів та обробки нетипових сценаріїв взаємодії з клієнтом. Результати тестування, подані у вигляді таблиць 4.3–4.5, відображають послідовність дій акторів, позитивні шляхи виконання сценаріїв та розширення, що моделюють виняткові ситуації.

Таблиця 4.3 – Обробка нетипової заявки на консультацію із медіафайлом

| Діючі актори | Клієнт, система, майстер |
|---|---|
| Мета | Сформувати запит на попередню діагностику із додаванням відео або фотофіксації проблеми |
| Передумова | Клієнт авторизований, точна причина несправності автомобіля невідома |
| Успішний сценарій | 1) Клієнт переходить до форми створення консультації та описує симптоми поломки;<br>2) клієнт додає медіафайл як додаток до опису;<br>3) система валідує розмір файлу та його формат;<br>4) система передає файл до хмарного сховища Cloudinary;<br>5) хмарне сховище повертає CDN-посилання на збережений файл;<br>6) система фіксує заявку у базі даних із прив'язкою до отриманого посилання та присвоює статус очікування. |

Кінець таблиці 4.3

| Сценарій успішний. Заявку створено, медіаконтент доступний для перегляду майстром. | |
|---|---|
| Розширення | |
| 1a | Перевищення допустимого обсягу файлу (понад 100 МБ). Серверна логіка перериває обробку запиту на етапі валідації. |
| 2a | Спроба створення повторної консультації на ту саму дату тим самим користувачем. Система виявляє порушення бізнес-правила (лише одна консультація на день). |
| Усі сценарії розширення успішно виконані. | |

Наведені результати тестування підтверджують коректність роботи механізму створення консультацій із використанням медіафайлів, а також стабільність обробки виняткових ситуацій, пов’язаних із порушенням встановлених обмежень системи.

Таблиця 4.4 – Формування запису на обслуговування

| Діючі актори | Клієнт, система |
|---|---|
| Мета | Створити бронювання з автоматичним призначенням майстрів на основі їхньої кваліфікації та поточного завантаження |
| Передумова | Клієнт авторизований у системі, сформовано список послуг |

Кінець таблиці 4.4

| | |
|---|---|
| Успішний сценарій | 1) Клієнт ініціює процес запису, обравши послуги та бажаний час бронювання;<br>2) система розраховує загальну тривалість виконання робіт та ініціює алгоритм пошуку;<br>3) для кожної послуги алгоритм фільтрує майстрів за наявністю відповідної спеціалізації;<br>4) система аналізує завантаженість кандидатів на вказану дату з урахуванням поточних транзакцій;<br>5) обрано фахівців з найнижчим показником завантаженості, що мають вільні часові інтервали;<br>6) система формує ієрархічну структуру замовлення та зберігає дані;<br>7) клієнту повертається підтвердження із зазначенням призначених майстрів та точного часу початку робіт. |
| Сценарій успішний. Бронювання створено, гарантовано відсутність накладок у графіку майстрів. | |
| Розширення | |
| 1a | Відсутність доступних фахівців із потрібною кваліфікацією на обраний час. Алгоритм не знаходить кандидата для однієї з послуг. |
| 2a | Під час одночасного бронювання одного й того самого часового інтервалу виникає конфлікт запитів. Система використовує механізм транзакцій бази даних, який не дозволяє виконати два однакові записи одночасно. |
| Усі сценарії розширення успішно виконані | |

Таблиця 4.5 – Формування переліку рекомендацій майстром за результатами консультації

| Діючі актори | Майстер, система |
|---|---|
| Мета | Формування переліку рекомендацій майстром за результатами консультації |
| Передумова | Майстер прийняв заявку на консультацію, провів огляд автомобіля та має картку даного звернення відкритою |
| Успішний сценарій | 1) Майстер відкриває інтерфейс роботи з завершеною консультацією у робочому кабінеті;<br>2) майстер здійснює пошук та обирає з каталогу одну або кілька послуг, необхідних для усунення виявленої несправності;<br>3) майстер ініціює відправку сформованого переліку системі;<br>4) система створює записи типу «Рекомендація»;<br>5) система оновлює статуси створених об'єктів та відображає їх у особистому кабінеті клієнта як пропозиції до погодження. |
| Сценарій успішний. Перелік рекомендацій сформовано та успішно передано клієнту для подальшого прийняття рішення. | |
| Розширення | |
| 1а | Спроба додавання послуги, ідентифікатор якої відсутній в активному каталозі. Система фіксує порушення посилальної цілісності. |
| Усі сценарії розширення успішно виконані. | |

Подальшим етапом перевірки програмного забезпечення стало тестування RESTful API, яке забезпечує взаємодію між клієнтською частиною системи та серверною логікою. Даний рівень тестування є важливим, оскільки

саме через API реалізуються основні операції обробки даних, верифікації прав доступу та обміну складними структурами інформації. З огляду на архітектурні особливості системи безпеки, фундаментальною передумовою для доступу до будь-яких захищених ресурсів є наявність валідного токена доступу (рис. 4.6), що визначило логіку початкового етапу інтеграційної верифікації.

Відображено процес автентифікації користувача через інтерфейс Swagger UI. Для цього до кінцевої точки */api/auth/login* сформовано POST-запит із JSON-пейлоадом, що містить облікові дані користувача.

Request URL

```
https://localhost:7064/api/Auth/login
```

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
jAwNS8wNS9pZGVudGl0eS9jbGFpbXMvZW1haWxhZGRyZXNzIjoib
sImlzcyI6IkF1dG9TZXJ2QVBJIiwiYXVkIjoiQXV0b1NlcnZDbGI
  "user": {
    "id": 27,
    "email": "master8@gmail.com",
    "firstName": "Микатa",
    "lastName": "Майстер",
    "phone": "+380634444443",
    "role": "master"
  }
}
```


Response headers

```
content-type: application/json; charset=utf-8
date: Wed,03 Jun 2026 08:51:47 GMT
server: Kestrel
```

Responses

| Code | Description |
|---|---|
| 200 | OK |

Рисунок 4.6 – POST-запит отримання токена користувачів системи

У результаті успішної перевірки облікових даних серверна частина повернула HTTP-відповідь зі статусом 200 OK. Тіло відповіді містило згенерований криптографічний JWT-токен та інформаційний об'єкт користувача, що включав його ідентифікатор та призначену роль. Отриманий

токен слугував базовим інструментом для всіх подальших перевірок контролю доступу.

Отриманий рядок токена впроваджено у глобальні налаштування середовища Swagger шляхом його додавання до спеціалізованого вікна авторизації за схемою (рис. 4.7).

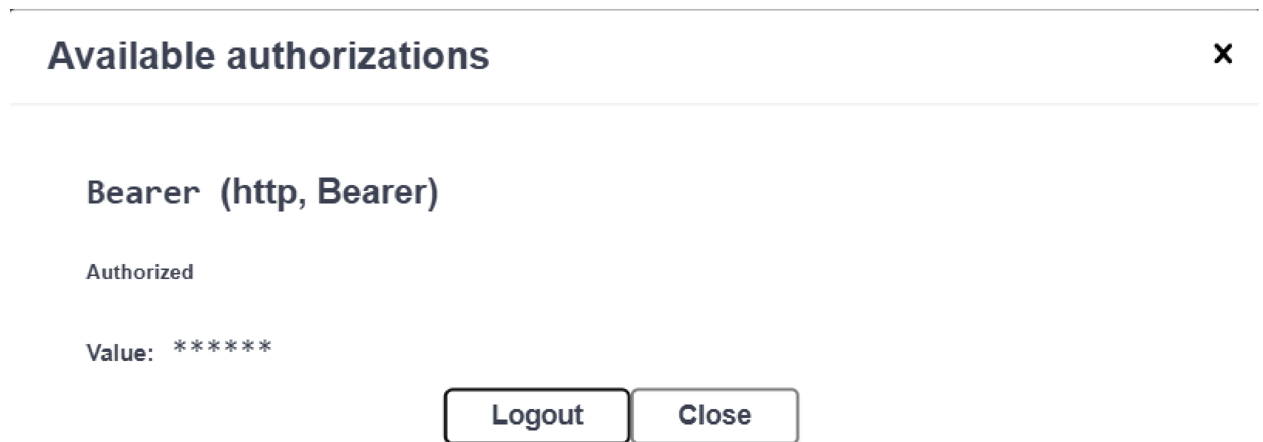


Рисунок 4.7 – Налаштування JWT авторизації в Swagger UI

Після активації даних налаштувань інструмент автоматично додавав заголовок *Authorization: Bearer <token>* до всіх подальших запитів, що дозволило відтворити реальну поведінку клієнтського вебзастосунку.

На основі сформованого авторизаційного контексту проведено серію тестів для верифікації доступу на основі RBAC [17]. З використанням токена користувача з роллю «клієнт» імітувалися спроби звернення до адміністративних кінцевих точок, зокрема до методів отримання статистики завантаженості майстрів *(/api/Admin/masters-stats)* та керування прайс-листом *(/api/Admin/services)*.

Декларативні обмеження, задані на рівні контролерів, успішно перехоплювали такі запити. Система не допускала їх виконання на рівні бізнес-логіки, повертаючи клієнту статус-код *403 Forbidden*, що підтверджує повну ізоляцію привілейованих операцій.

Окремої уваги набуло тестування захисту від несанкціонованого витікання даних. Здійснено спроби отримання сторонніх даних шляхом прямої маніпуляції числовими ідентифікаторами у URL-адресах.

Наприклад, під час виклику методу */api/Profile/{userId}* із підстановкою ідентифікатора іншого користувача, логіка контролера порівнювала цей параметр із ідентифікатором, зашифрованим у JWT-токені (рис. 4.8). Зафіксовано, що будь-яка невідповідність призводить до миттєвої блокування запиту та генерації відповіді 403 Forbid до звернення до бази даних.

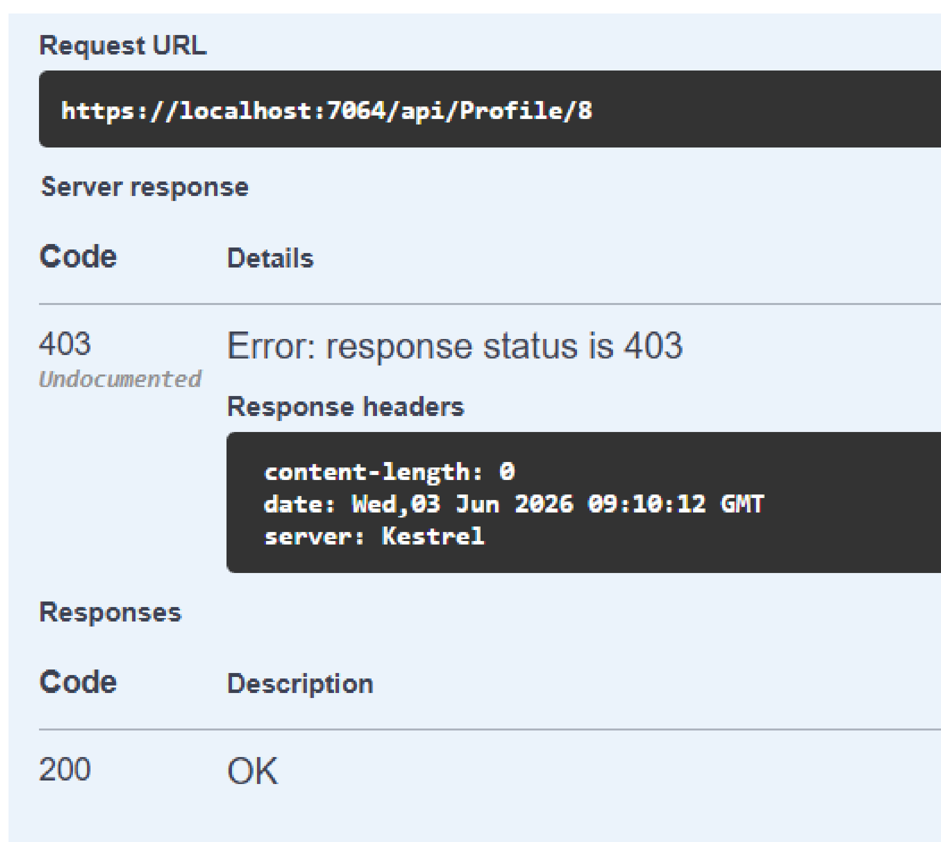


Рисунок 4.8 – Результат тестування доступу до ресурсу /api/Profile/{userId} з перевіркою Insecure Direct Object Reference

Окрім перевірки авторизаційних механізмів, на рівні API проаналізовано інфраструктурну обробку конфліктних ситуацій, зокрема конкурентного доступу, який вже розглядався під час функціонального тестування (розширення 2а, табл. 4.3). У результаті перевірки встановлено, що механізми транзакцій бази даних *BeginTransaction/Rollback* та серверна перевірка

часових перетинів виключають виникнення подвійного бронювання, гарантуючи цілісність розкладу.

У процесі аналізу поведінки API зафіксовано дотримання кодів стану HTTP:

– 200 OK – успішне виконання запитів на читання та модифікацію;
– 201 Created – успішне створення ресурсу;
– 400 Bad Request – помилка валідації;
– 401 Unauthorized – відсутність або недійсність токена;
– 403 Forbidden – недостатній рівень доступу;
– 404 Not Found – звернення до неіснуючого ресурсу.

Для підвищення рівня надійності програмного забезпечення частину перевірок реалізовано у вигляді автоматизованих модульних тестів із використанням фреймворку xUnit. Автоматизація застосовувалася для верифікації окремих компонентів бізнес-логіки, зокрема механізмів визначення доступних часових інтервалів, перевірки перетину бронювань та алгоритмів розрахунку завантаженості майстрів.

### 4.5 Опис інтерфейсу та функціональних можливостей системи

Візуальна складова розробленої системи побудована на логіці послідовного проведення користувача через усі етапи сервісного обслуговування. Графічний інтерфейс орієнтований на те, щоб приховати складність серверних обчислень та запропонувати зрозумілі інструменти для вирішення конкретних завдань.

Головна сторінка системи реалізована як центральний навігаційний елемент взаємодії користувача із сервісом (рис. 4.9). Її структура побудована за принципом поетапного ознайомлення з можливостями автосервісу та швидкого переходу до ключових функцій системи.

У верхній частині інтерфейсу розташовано навігаційну панель, яка містить логотип сервісу, основні пункти меню та елементи авторизації

користувача. Навігація забезпечує швидкий доступ до розділів послуг, консультацій та контактної інформації. Додатково у правій частині панелі розміщено кнопку входу до особистого кабінету.

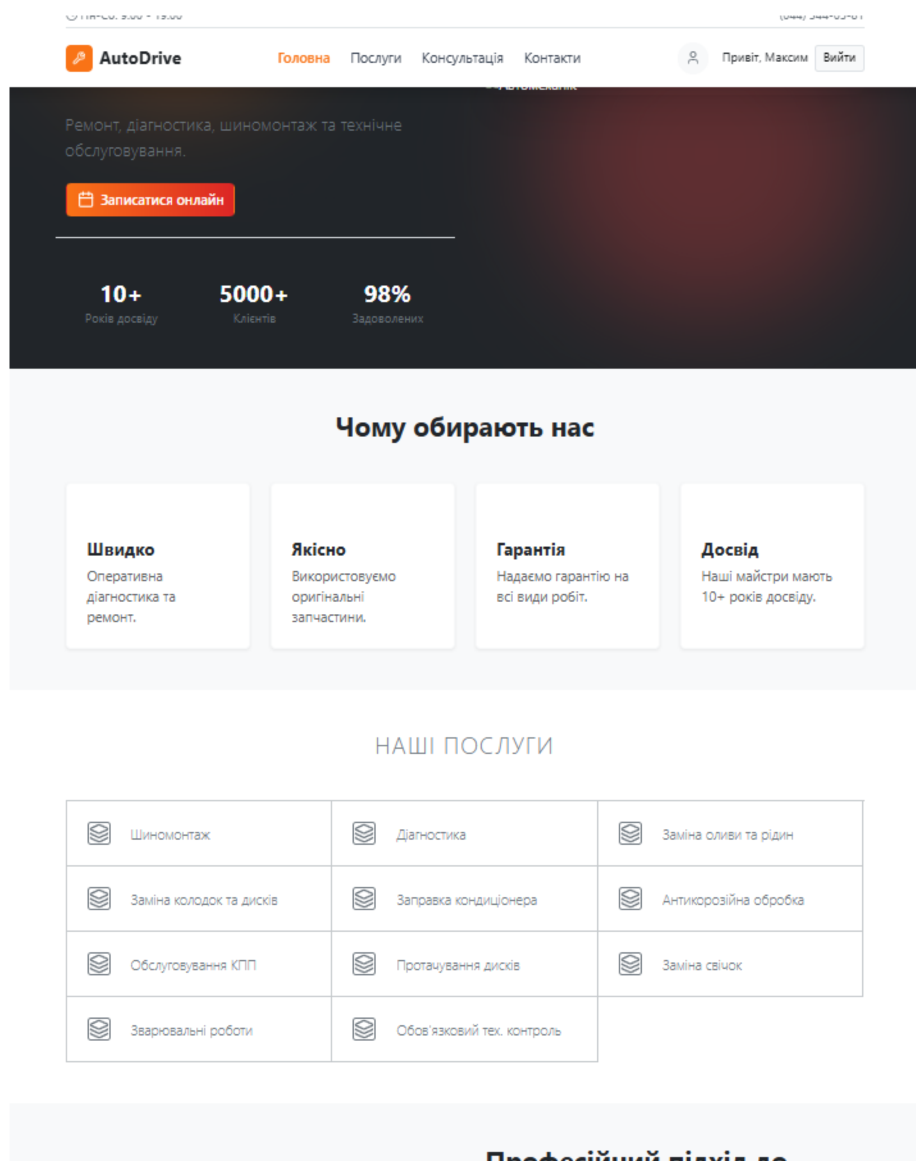


Рисунок 4.9 – Головна сторінка інформаційної системи автосервісу

Після обрання категорії користувач потрапляє на сторінку каталогу (рис. 4.10). Інтерфейс розділено на дві логічні блоки. У лівій частині виведено перелік конкретних робіт із зазначенням фіксованої вартості. Права частина представлена з формою попереднього запису. Важливою особливістю цього інтерфейсу є наявність календаря зі списком часових інтервалів. Оскільки розподіл завантаження майстрів відбувається на сервері, клієнту

пропонуються для вибору виключно ті записи, які дійсно вільні. Це виключає можливість бронювання зайнятого часу та запобігає помилкам у розкладі.

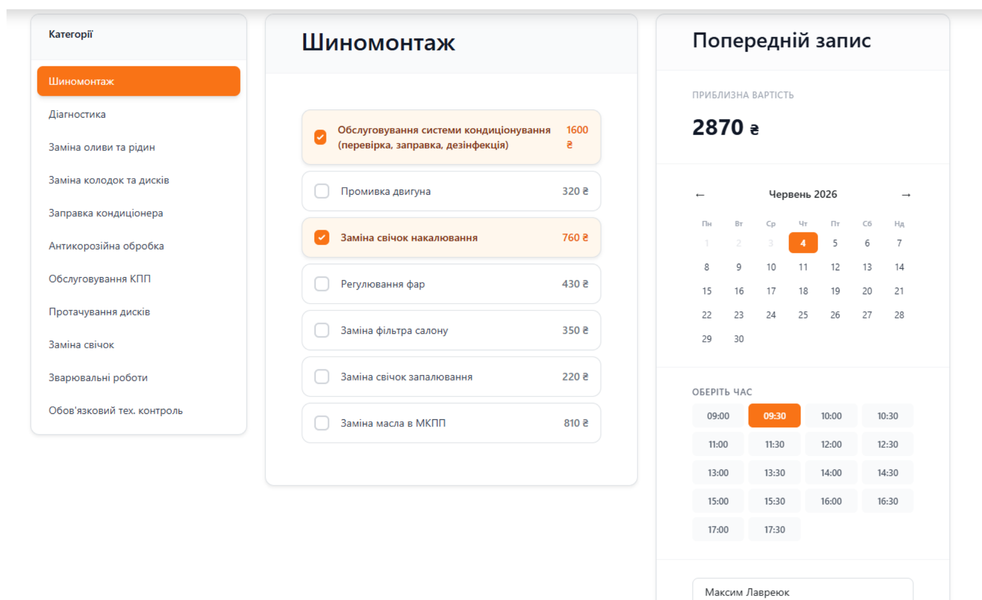


Рисунок 4.10 – Інтерфейс вибору та бронювання  послуг

У ситуаціях, коли клієнту невідома точна причина несправності автомобіля, передбачено альтернативний маршрут взаємодії – сторінку онлайн-консультації. Інтерфейс зосереджений на зборі максимального обсягу інформації про проблему: тут розташоване розширене текстове поле для детального опису симптомів, а також реалізована можливість завантажити медіафайли.

Після прийняття заявки та аналізу наданих матеріалів майстром проводиться попередній огляд ситуації. На основі цих даних система дозволяє фахівцю сформувати структурований перелік рекомендацій – конкретних послуг, необхідних для усунення виявленої несправності (рис. 4.11).

Центром управління індивідуальними даними та моніторингу процесів виступає персональний кабінет (рис. 4.11). Верхня частина інтерфейсу агрегує технічну інформацію про транспортний засіб користувача, що автоматично підставляється у всі подальші заявки. Нижче розміщено хронологічну таблицю завершених та поточних візитів. Кожен запис деталізує перелік наданих послуг, фінансовий підсумок та актуальний статус обробки

замовлення. Таке групування дозволяє клієнту повністю контролювати життєвий цикл обслуговування свого автомобіля в межах одного інтерфейсу.

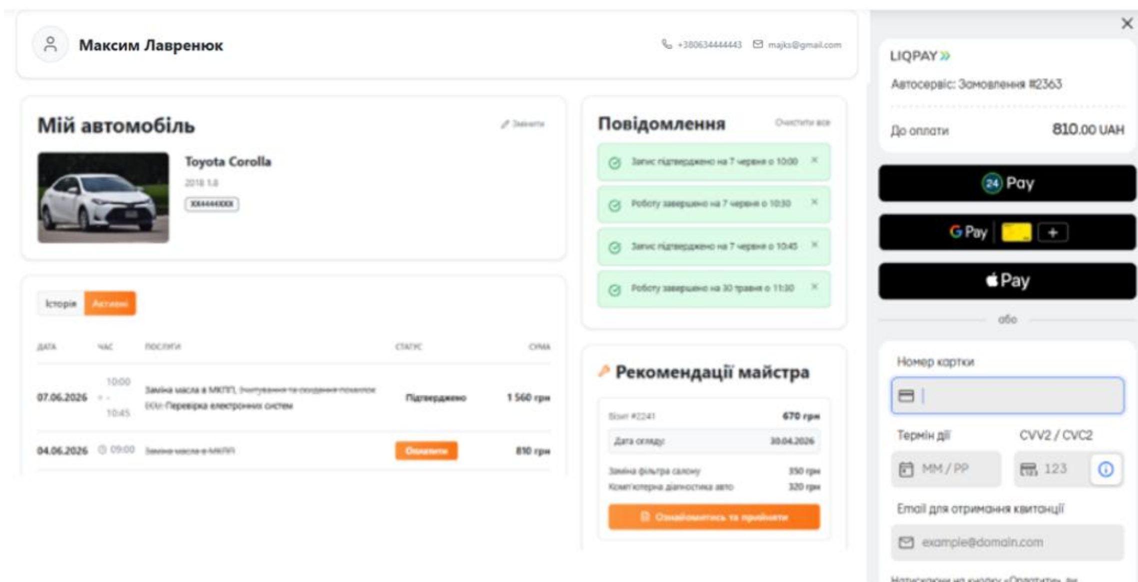


Рисунок 4.11 – Інтерфейс профілю користувача з історією записів та оплатою

Окремої уваги заслуговує бічна панель кабінету, де реалізовано блок обробки рекомендацій. Саме тут результати попередніх консультацій: якщо майстер виявив необхідність додаткових робіт, система автоматично генерує відповідні пропозиції (рис. 4.12).

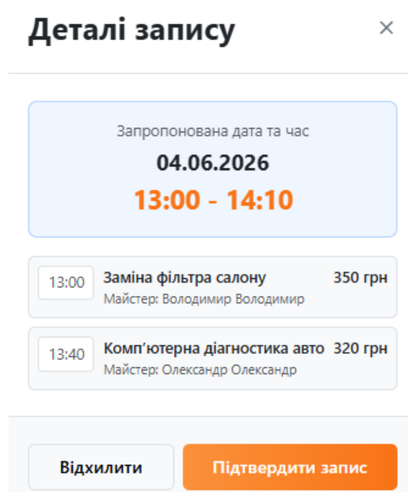


Рисунок 4.12 – Інтерфейс блоку рекомендацій від майстра

Окрім клієнтської частини, інформаційна система передбачає спеціалізований інтерфейс для фахівців станцій.

Робочий простір майстра розпочинається з модуля керування консультаціями (рис. 4.13). Даний інтерфейс відображає чергу активних запитів від користувачів, яким потрібна попередня діагностика.

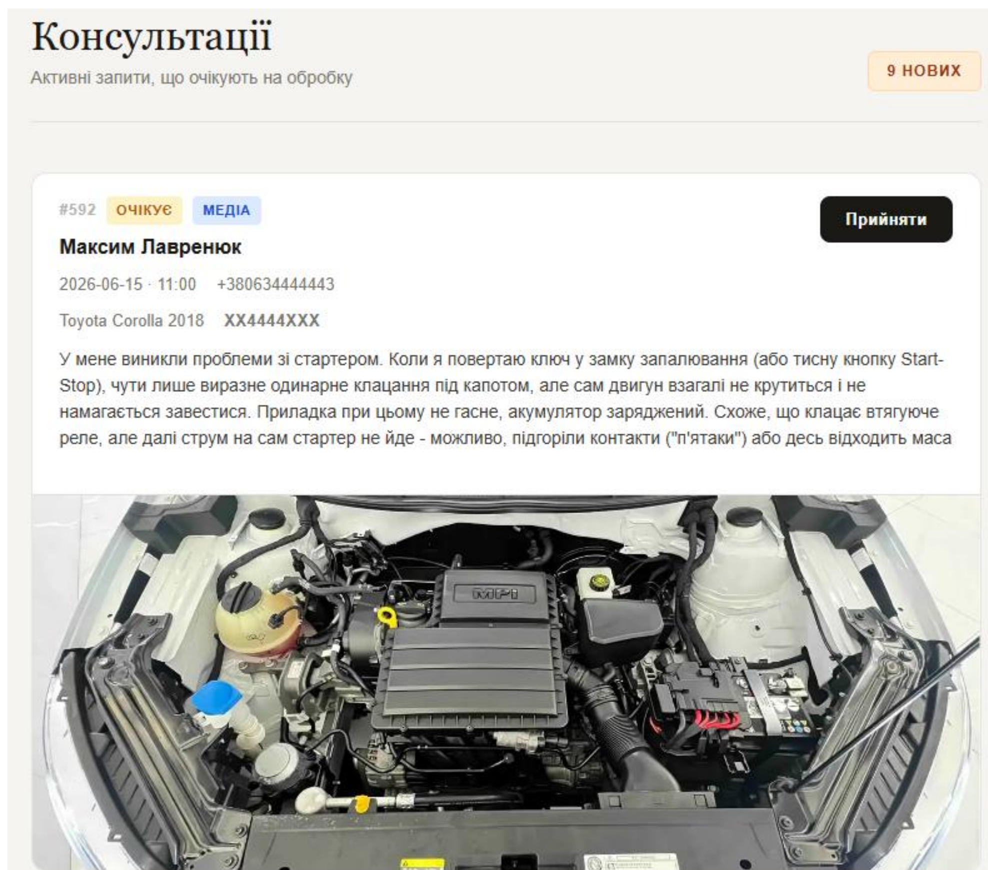


Рисунок 4.13 – Інтерфейс модуля керування консультаціями майстра

Для кожного елемента списку інтерфейс демонструє текстовий опис проблеми, запропонований клієнтом час та, що найважливіше, візуалізує завантажені медіафайли. Наявність вбудованого переглядача зображень суттєво прискорює процес попередньої оцінки ситуації.

Основним інструментом моніторингу та управління завданнями є інтерфейс робочого графіка (рис. 4.14). Він реалізований у вигляді хронологічної шкали, яка наочно ілюструє послідовність виконання замовлень протягом обраного робочого дня. Для кожного запису система виводить назву обраної послуги та її тривалість. Інтерфейс інтегровано з механізмом зміни станів: коли майстер фізично завершує ремонтні роботи, він ініціює відповідний статус. Це запускає серверну обробку, яка оновлює дані в базі та

автоматично змінює статус замовлення в особистому кабінеті клієнта. Наявність кольорової легенди статусів на панелі дозволяє фахівцю візуально контролювати хід виконання всіх призначених на день операцій.

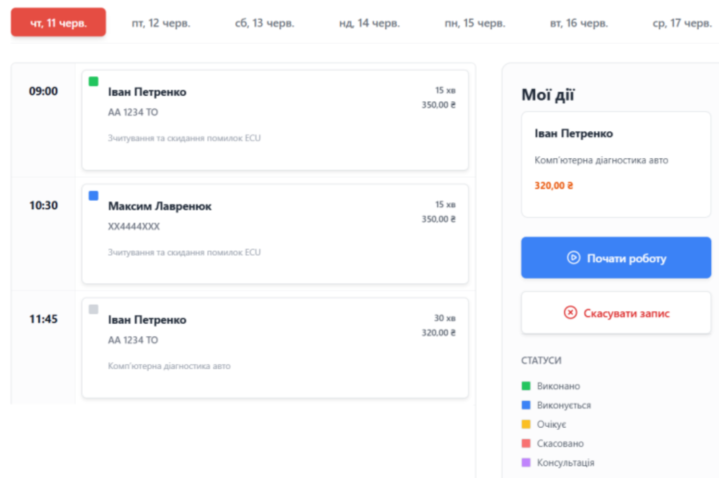


Рисунок 4.14 – Інтерфейс графіка майстра з управлінням статусами

Для управління процесами автосервісу реалізовано адміністративну панель (рис. 4.15). Верхній сегмент інтерфейсу містить модуль статистики, де у вигляді інформаційних карток виведені ключові показники: кількість активних майстрів, загальне число записів за день та їх розподіл за статусами обробки. Це дозволяє адміністратору миттєво оцінювати поточне навантаження на станцію.

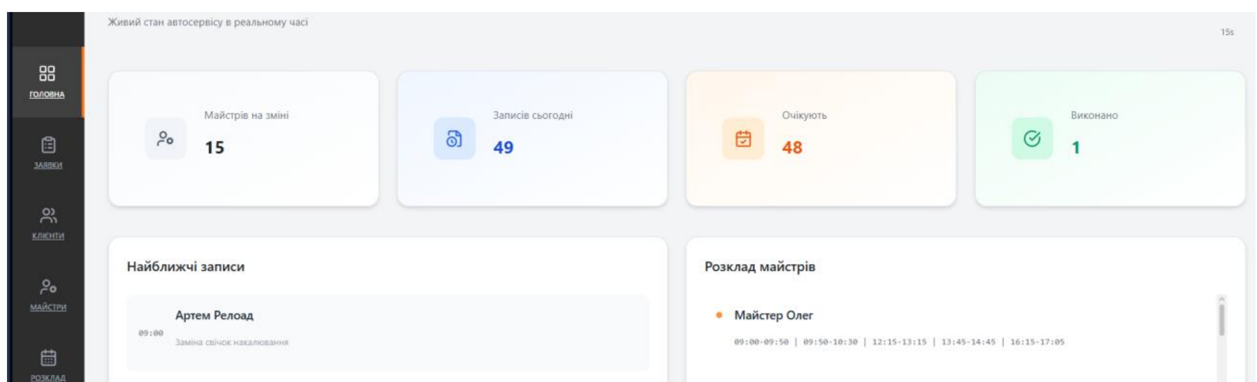


Рисунок 4.15 – Інтерфейс адміністративної панелі

Для контролю за всіма замовленнями станції реалізовано інтерфейс управління сервісними заявками (рис. 4.16). Основним елементом даного

інтерфейсу є інформаційна таблиця, яка агрегує повний перелік сформованих звернень. Для зручності аналізу та пошуку дані структуровані за ключовими атрибутами.

Рисунок 4.16 – Інтерфейс управління сервісними заявками

Для ефективного розподілу ресурсів та контролю персоналу в системі передбачено окремий модуль управління профілями фахівців (рис. 4.17). Інтерфейс представлено у вигляді структурованої таблиці, яка агрегує детальну інформацію про кожного майстра.

Рисунок 4.17 – Інтерфейс модуля управління майстрами

Особливістю даного інтерфейсу є наявність аналітичних колонок, що відображають завантаженість фахівця. Система автоматично розраховує загальну кількість вільних часових інтервалів та демонструє конкретні проміжки вже заброньованого часу. Інтерфейс інтегровано з механізмами управління доступом: за допомогою кнопки «Деактивувати» можна швидко перевести обліковий запис у неактивний, а функція «Додати майстра» забезпечує можливість додавати нових працівників.

Узагальнений розподіл завдань та координація роботи фахівців забезпечується через модуль загального графіка майстрів (рис. 4.18). Даний інтерфейс реалізований у календарному форматі, що дає змогу адміністратору в режимі реального часу контролювати зайнятість працівників станції протягом тижня. Верхня частина інтерфейсу містить елементи навігації з фіксацією обраного періоду для швидкого перемикання між датами.

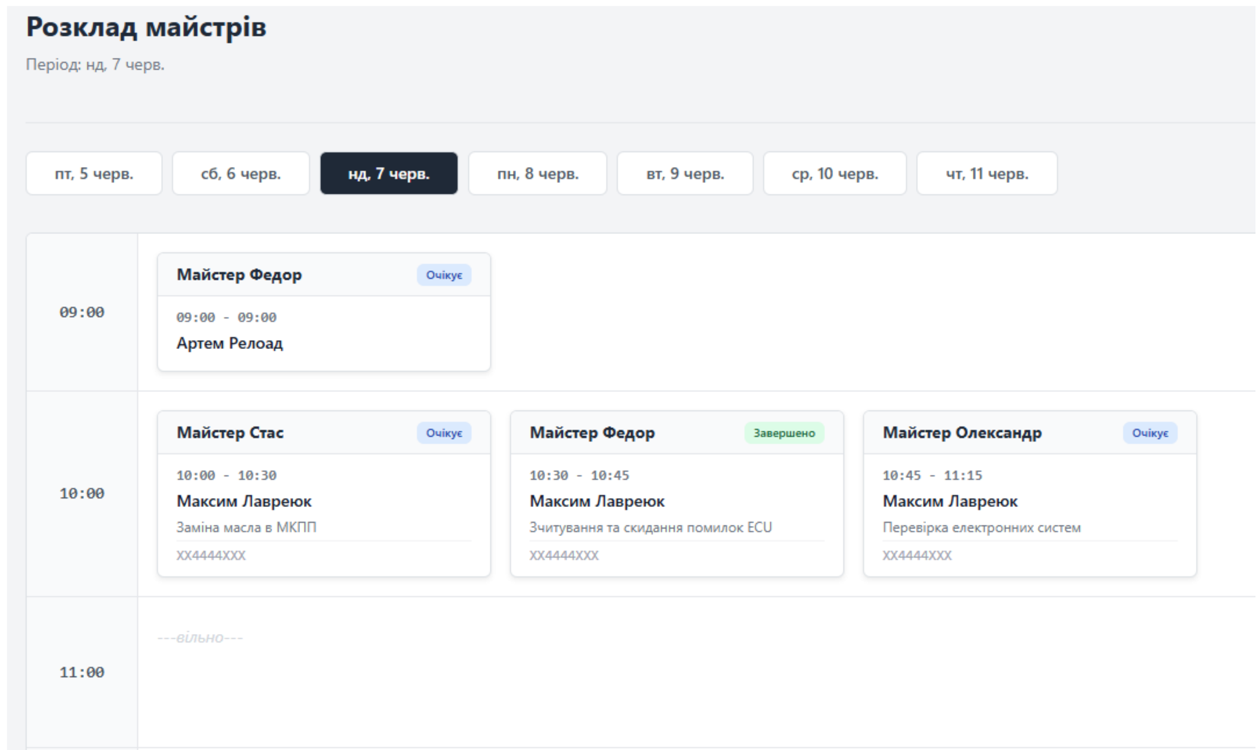


Рисунок 4.18 – Інтерфейс перегляду графіка майстрів

Основна частина інтерфейсу побудована за принципом вертикальної часової шкали. У межах обраного дня система візуалізує заброньовані часові проміжки у вигляді структурованих карток. Для кожного запису інтерфейс агрегує ключову інформацію, необхідну для оперативної оцінки.

**Висновки до розділу 4**

У четвертому розділі реалізовано серверну архітектуру та клієнтський інтерфейс інформаційної системи автосервісу. Запропоновано підходи до побудови сервісного шару з використанням принципу інверсії залежностей, що дозволило організувати бекенд-логіку та відокремити прийом даних від виконання бізнес-правил, а також забезпечити підключення зовнішніх сервісів, зокрема хмарного сховища Cloudinary для обробки медіафайлів та платіжної системи LiqPay для проведення фінансових операцій.

Проведено перевірку програмного забезпечення шляхом функціонального та інтеграційного тестування. Підтверджено коректність обробки граничних ситуацій та конкурентних спроб бронювання. Також перевірено ефективність захисту програмного інтерфейсу від несанкціонованого доступу на основі ролей та токенів, у тому числі під час використання механізму автентифікації через обліковий запис Google.

Візуальна складова системи розроблена з урахуванням специфічних завдань трьох користувацьких ролей. Інтерфейс клієнта інтегрує каталог послуг із календарем доступних часових проміжків та модуль обробки рекомендацій та консультацій. Робочий простір майстра містить інструменти для перегляду медіафайлів під час діагностики та динамічний календар для керування статусами виконання робіт. Адміністративна панель об'єднує статистичні показники завантаження системи, табличний перелік сервісних заявок та загальний графік фахівців. Отримані результати свідчать про готовність створеного програмного рішення до практичної експлуатації.

# ВИСНОВКИ

У процесі виконання кваліфікаційної бакалаврської роботи вирішено завдання автоматизації процесів взаємодії між автосервісом та його клієнтами. Розроблено веборієнтовану інформаційну систему, яка усуває недоліки ручного ведення записів та забезпечує прозорий контроль над виконанням ремонтних робіт.

На етапі аналізу предметної області виявлено суттєві обмеження наявних програмних рішень. Більшість вебсервісів виконують лише функцію прийому заявок. У свою чергу, ERP-системи мають складну структуру та не забезпечують достатньо швидкої взаємодії з клієнтами. Визначено, що ключовою проблемою галузі є відсутність гнучкого автоматичного розподілу завдань між фахівцями з урахуванням їхнього поточного навантаження та кваліфікації.

Виявлені недоліки допомогли скласти чіткий перелік вимог до нової системи. Визначено, що саме входитиме в проєкт та користувачі будуть користуватися сервісом. Також сформульовані технічні умови, які треба врахувати перед початком розробки архітектури.

Для побудови системи обрано архітектуру модульного моноліту з використанням принципів чистої архітектури, що дозволило відокремити бізнес-логіку від інфраструктурних деталей. За допомогою UML-моделювання сформовано структуру даних, де замовлення поділяються на батьківські та дочірні записи для фіксації робіт різних майстрів у межах одного візиту. Дослідження в галузі планування ресурсів допомогли адаптувати концепцію багаторесурсного призначення до специфіки функціонування станції технічного обслуговування. На основі цього розроблено алгоритм послідовного розподілу завдань, який зіставляє доступні часові інтервали фахівців із сумарною тривалістю обраного комплексу послуг.

Окрім стандартного алгоритму бронювання, реалізовано обробку нетипових звернень, коли точна причина несправності автомобіля невідома.

Для цього передбачено модуль попередніх консультацій із можливістю передавати медіафайли. За результатами дистанційного та фізичного огляду майстер формує структурований перелік рекомендацій, який клієнт може прийняти або відхилити.

Практична реалізація виконана на технологічному стеку ASP.NET Core та React. Для оптимізації роботи програмного забезпечення та зменшення навантаження на основний сервер частину функцій передано зовнішнім сервісам. Обробка медіафайлів під час попередніх діагностичних консультацій делегована хмарному сховищу Cloudinary. Фінансові операції інтегровано через платіжний шлюз LiqPay, що виключає пряму обробку банківських реквізитів на стороні автосервісу. Процедура входу до системи спрощена за рахунок підключення механізму автентифікації через обліковий запис Google.

Верифікація розробленого рішення підтвердила коректність роботи серверної логіки. Функціональне тестування довело відсутність конфліктів при конкурентному бронюванні часу завдяки використанню транзакцій. Інтеграційні перевірки API підтвердили надійність захисту від несанкціонованого доступу на основі ролей та токенів, а також неможливість перегляду сторонніх даних шляхом підміни ідентифікаторів.

Завдання роботи виконано повністю. Система побудована з використанням патернів проєктування, що робить її код зручним для підтримки та придатним до подальшого масштабування. Запропоноване рішення забезпечує автоматизоване планування ресурсів та надає зручний інструмент управління замовленнями.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dolgova L., Yamnenko H. Use of information system tools for automation of business processes of the enterprise. Economic Analysis. 2021. No. 31(2). P. 90–97. DOI: https://doi.org/10.35774/econa2021.02.090.

2. Koval N. Functional models of information technology and information system architecture. Bulletin of Lviv National Agrarian University Agroengineering Research. 2021. № 25. P. 157–166. DOI: https://doi.org/10.31734/agroengineering2021.25.157.

3. Microsoft Dynamics 365 Business Central URL: https://clarity-team.com/business_central/ (дата звернення: 22.04.2026).

4. Автосервіс та сервіс технічного обслуговування автомобілів AutoDoc URL: https://www.autodoc.in.ua (дата звернення: 19.01.2026).

5. Autotech. Сервіс та ремонт автомобілів. URL: https://autotech.kiev.ua/uk/ (дата звернення: 19.01.2026).

6. VIDI. Портал сервісів обслуговування автомобілів. URL: https://vidi.ua/ua/service/ (дата звернення: 19.01.2026).

7. Хоршков В., Ліщина Н., Сичук В. Модульний моноліт: архітектурний підхід для побудови високодоступної системи управління зарядними станціями. Computer-integrated technologies: education, science, production. 2025. № 57. С. 31–42. DOI: https://doi.org/10.36910/6775-2524-0560-2024-57-05.

8. Maksymenko V., Frolov O. "Comparative performance analysis of GRPC and RESTFUL API microservices using mongodb and ms sql server". Scientific papers of donetsk national technical university. Series: informatics, cybernetics and computer science. 2025. Vol. 1, no. 40. P. 51–59. Doi: https://doi.org/10.31474/1996-1588-2025-1-40-51-59.

9. Lukić I., Köhler M., Kiralj E. Appointment scheduling system in multi doctor/multi services environment. International journal of electrical and computer

engineering systems. 2021. Vol. 12, № 3. P. 171–176. DOI: https://doi.org/10.32985/ijeces.12.3.6.

10. Laplante P. A., Kassab M. H. Requirements Engineering: A Road Map to the Future. Requirements Engineering for Software and Systems. 4th ed. New York, 2022. P. 295–305. DOI: https://doi.org/10.1201/9781003129509-12.

11. Blinowski G., Ojdowska A., Przybyłek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access. 2022. Vol. 10. P. 20357–20374. DOI: https://doi.org/10.1109/access.2022.3152803.

12. Al-Qora'n L. F., Al-Said Ahmad A. Modular Monolith Architecture: A Systematic Literature Review. Future Internet. 2025. Vol. 17, no. 11. P. 496. DOI: https://doi.org/10.3390/fi17110496.

13. Бирин А., Котенко Н. Важливість побудови чистої архітектури в процесі розробки програмного забезпечення. Наука і техніка сьогодні. 2025. № 3(44). DOI: https://doi.org/10.52058/2786-6025-2025-3(44)-989-997.

14. Ліщина Н., Бойко Л., Гульчук Ю. Оцінка ризиків та їх вплив на життєвий цикл розробки програмного забезпечення. Herald of khmelnytskyi national university. Technical sciences. 2024. Т. 343, № 6(1). С. 141–145. DOI: https://doi.org/10.31891/2307-5732-2024-343-6-21.

15. O. I. Efficiency use of working time in car service enterprises. The national Transport University Bulletin. 2021. Vol. 1, no. 50. P. 92–103. DOI: https://doi.org/10.33744/2308-6645-2021-3-50-092-103.

16. Terencio N., Sony A. Enhancing MySQL database security with MySQL enterprise transparent data encryption. Journal of Logistics, Informatics and Service Science. 2023. Vol. 10, № 4. P. 154–173. DOI: https://doi.org/10.33168/jliss.2023.0411.

17. Ямнич А., Коробейнікова Т. Модель контролю доступу персоналу до підприємств на основі RBAC та технології blockchain. Herald of khmelnytskyi national university. Technical sciences. 2024. Т. 343, № 6(1). С. 380–386. Doi: https://doi.org/10.31891/2307-5732-2024-343-6-56.

# ДОДАТОК А

# ЛІСТИНГ КОДУ СЕРВЕРНОЇ ЧАСТИНИ

Листинг коду моделі користувача User.cs та профілю UserProfile.cs:

```
namespace AutoServ.Core.Entities
{
    public class User
    {
        public int Id { get; set; }
        public string Email { get; set; } = null!;
        public string PasswordHash { get; set; } = null!;
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Phone { get; set; } = null!;
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public string Role { get; set; } = "client";


        public UserProfile? Profile { get; set; }


        public ICollection<Booking> Bookings { get; set; } = new List<Booking>();
    }
}

namespace AutoServ.Core.Entities
{
    public class UserProfile
    {
        public int Id { get; set; }
        public int UserId { get; set; }
        public string? AvatarUrl { get; set; }
        public string? CarBrand { get; set; }
        public string? CarModel { get; set; }
        public int? CarYear { get; set; }
        public string? CarEngine { get; set; }
        public string? CarPlate { get; set; }
        public string? CarImageUrl { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
        public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

        public User User { get; set; } = null!;
    }
      }
```

Лістинг коду моделі майстра (Master.cs):

```
using System.ComponentModel.DataAnnotations.Schema;

namespace AutoServ.Core.Entities
{
    public class Master
    {
        public int Id { get; set; }
        public string FirstName { get; set; } = string.Empty;
        public string LastName { get; set; } = string.Empty;
        public string? Phone { get; set; }
        public bool IsActive { get; set; } = true;
        public int ExperienceYears { get; set; } = 0;
```

```
        [Column("user_id")]
        public int? UserId { get; set; }
        public User? User { get; set; }

        public ICollection<Booking>? Bookings { get; set; }
        public ICollection<ServiceItem> ServiceItems { get; set; } = new
List<ServiceItem>();
    }
      }
```

Лістинг коду MasterService.cs:

```
using AutoServ.Core.Interfaces;
using AutoServ.Core.Models;
using AutoServ.DTOs.Masters;
using AutoServ.Infrastructure.Data;
using Microsoft.EntityFrameworkCore;

namespace AutoServ.Infrastructure.Services
{
    public class MasterService : IMasterService
    {
        private readonly ApplicationContext _context;

        public MasterService(ApplicationContext context) { _context = context; }

        public async Task<List<MasterDto>> GetAllActiveAsync()
        {
            return await _context.Masters.Where(m => m.IsActive)
                .Select(m => new MasterDto { Id = m.Id, FullName = $"{m.FirstName}
{m.LastName}", Phone = m.Phone, ExperienceYears = m.ExperienceYears, IsActive =
m.IsActive })
                .ToListAsync();
        }

        public async Task<object?> GetByIdAsync(int id)
        {
            var master = await _context.Masters.Include(m =>
m.ServiceItems).FirstOrDefaultAsync(m => m.Id == id);
            if (master == null) return null;

            return new
            {
                master.Id,
                FullName = $"{master.FirstName} {master.LastName}",
                master.Phone,
                master.ExperienceYears,
                master.IsActive,
                Specializations = master.ServiceItems.Select(s => new { s.Id,
s.Title, s.PriceFrom, s.DurationMinutes })
            };
        }

        public async Task<Result> UpdateSpecializationsAsync(int id,
UpdateMasterSpecializationsDto dto)
        {
            var master = await _context.Masters.Include(m =>
m.ServiceItems).FirstOrDefaultAsync(m => m.Id == id);
            if (master == null) return Result.Failure("master not found");

            master.ServiceItems = await _context.ServiceItems.Where(s =>
dto.ServiceItemIds.Contains(s.Id)).ToListAsync();
            try { await _context.SaveChangesAsync(); return Result.Success(); }
```

```
            catch (DbUpdateException ex) { return Result.Failure("error while 
saving: " + ex.InnerException?.Message); }
        }

        public async Task<Result> AddSpecializationAsync(int id, int serviceItemId)
        {
            var master = await _context.Masters.Include(m => 
m.ServiceItems).FirstOrDefaultAsync(m => m.Id == id);
            if (master == null) return Result.Failure("master not found");
            if (master.ServiceItems.Any(s => s.Id == serviceItemId)) return 
Result.Failure("ця спеціалізація вже є у майстра");

            var service = await _context.ServiceItems.FindAsync(serviceItemId);
            if (service == null) return Result.Failure("service not found");

            master.ServiceItems.Add(service);
            await _context.SaveChangesAsync();
            return Result.Success();
        }
    }
      }
```

Лістинг коду BookingsController.cs:

```
using AutoServ.Core.Interfaces;
using AutoServ.DTOs.Bookings;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace AutoServ.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    [Authorize]
    public class BookingsController : ControllerBase
    {
        private readonly IBookingService _bookingService;

        public BookingsController(IBookingService bookingService)
        {
            _bookingService = bookingService;
        }

        [HttpGet]
        [Authorize(Roles = "admin")]
        public async Task<IActionResult> GetAll() => Ok(await 
_bookingService.GetAllAsync());

        [HttpGet("{id}")]
        public async Task<IActionResult> GetById(int id)
        {
            var booking = await _bookingService.GetByIdAsync(id);
            if (booking == null) return NotFound(new { message = $"бронювання №{id} 
NotFound" });

            var currentUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
            if (!User.IsInRole("admin") && booking.UserId.ToString() != 
currentUserId) return Forbid();

            return Ok(new
            {
                booking.Id,
```

```
                booking.UserId,
                booking.CustomerName,
                booking.CustomerPhone,
                booking.CarPlate,
                booking.BookingDate,
                booking.BookingTime,
                booking.TotalPrice,
                booking.Status,
                booking.ParentBookingId,
                MasterName = booking.Master != null ? $"{booking.Master.FirstName}
{booking.Master.LastName}" : "Не призначено",
                Services = booking.Services.Select(s => new { s.ServiceItemId, Name
= s.ServiceItem?.Title ?? "Послугу видалено", s.PriceAtBooking }),
                Recommendations = booking.Recommendations.Select(r => new { r.Id,
r.ServiceItemId, ServiceTitle = r.ServiceItem?.Title ?? "Послугу видалено",
r.Comment, r.IsAccepted, r.CreatedBookingId })
            });
        }

        [HttpGet("user/{userId}")]
        public async Task<IActionResult> GetByUser(int userId)
        {
            var currentUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
            if (currentUserId != userId.ToString() && !User.IsInRole("admin"))
return Forbid();

            var result = await _bookingService.GetUserHistoryAsync(userId);
            return Ok(result.Value);
        }

        [HttpGet("my-schedule")]
        [Authorize(Roles = "master, admin")]
        public async Task<IActionResult> GetMySchedule([FromQuery] int userId,
[FromQuery] DateTime? date)
        {
            var currentUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
            if (currentUserId != userId.ToString() && !User.IsInRole("admin"))
return Forbid();

            var result = await _bookingService.GetMasterScheduleAsync(userId, date
?? DateTime.UtcNow);
            return Ok(result.Value);
        }

        [HttpPost]
        public async Task<IActionResult> Create(CreateBookingDto dto)
        {
            var userIdString = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
            if (string.IsNullOrEmpty(userIdString)) return Unauthorized(new {
message = "помилка авторизації" });

            dto.UserId = int.Parse(userIdString);

            var result = await _bookingService.CreateAsync(dto);

            if (!result.IsSuccess) return BadRequest(new { message = result.Error
});

            return Ok(new { message = "замовлення успішно створено.", bookings =
result.Value });
        }

        [HttpPatch("{id}/status")]
        [Authorize(Roles = "master, admin")]
```

```
        public async Task<IActionResult> UpdateStatus(int id, [FromBody]
UpdateStatusDto dto)
        {
            var result = await _bookingService.UpdateTaskStatusAsync(id,
dto.Status);
            return result.IsSuccess ? Ok(new { message = "status upd" }) :
BadRequest(new { message = result.Error });
        }
    }
}
```

Лістинг коду AuthController.cs:

```
using AutoServ.Core.Interfaces;
using AutoServ.DTOs.Auth;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace AutoServ.Controllers
{
    [ApiController, Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly IUserService _userService;

        public AuthController(IUserService userService) { _userService =
userService; }

        [HttpPost("register")]
        public async Task<IActionResult> Register(RegisterDto dto)
        {
            var result = await _userService.RegisterAsync(dto);
            return result.IsSuccess ? Ok(result.Value) : BadRequest(new { message =
result.Error });
        }

        [HttpPost("register-master")]
        [Authorize(Roles = "admin")]
        public async Task<IActionResult> RegisterMaster([FromBody] RegisterMasterDto
dto)
        {
            var result = await _userService.RegisterMasterAsync(dto);
            return result.IsSuccess ? Ok(result.Value) : BadRequest(new { message =
result.Error });
        }

        [HttpPost("login")]
        public async Task<IActionResult> Login(LoginDto dto)
        {
            var result = await _userService.LoginAsync(dto);
            return result.IsSuccess ? Ok(result.Value) : Unauthorized(new { message
= result.Error });
        }
    }
}
```

Лістинг коду контексту бази даних ApplicationContext.cs:

```
using AutoServ.Core.Entities;
using AutoServ.Core.Enums;
using Microsoft.EntityFrameworkCore;

namespace AutoServ.Infrastructure.Data
```

```
{
    public class ApplicationContext : DbContext
    {
        public ApplicationContext(DbContextOptions<ApplicationContext> options) :
base(options) { }

        public DbSet<User> Users => Set<User>();
        public DbSet<UserProfile> UserProfiles => Set<UserProfile>();
        public DbSet<Service> Services => Set<Service>();
        public DbSet<ServiceItem> ServiceItems => Set<ServiceItem>();
        public DbSet<ServiceDetail> ServiceDetails => Set<ServiceDetail>();
        public DbSet<Booking> Bookings => Set<Booking>();
        public DbSet<BookingService> BookingServices => Set<BookingService>();
        public DbSet<Master> Masters => Set<Master>();
        public DbSet<Recommendation> Recommendations => Set<Recommendation>();
        public DbSet<Consultation> Consultations => Set<Consultation>();

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<User>().ToTable("users");
            modelBuilder.Entity<UserProfile>().ToTable("user_profiles");
            modelBuilder.Entity<Service>().ToTable("services");
            modelBuilder.Entity<ServiceItem>().ToTable("serviceitems");
            modelBuilder.Entity<ServiceDetail>().ToTable("servicedetails");
            modelBuilder.Entity<Booking>().ToTable("bookings");
            modelBuilder.Entity<BookingService>().ToTable("booking_services");
            modelBuilder.Entity<Master>().ToTable("masters");
            modelBuilder.Entity<Recommendation>().ToTable("recommendations");
            modelBuilder.Entity<Consultation>().ToTable("consultations");

            modelBuilder.Entity<Booking>().Property(b =>
b.TotalPrice).HasPrecision(18, 2);
            modelBuilder.Entity<BookingService>().Property(bs =>
bs.PriceAtBooking).HasPrecision(18, 2);


            modelBuilder.Entity<Booking>()
                .Property(b => b.Status)
                .HasConversion(
                    v => v.ToString().ToLower(),
                    v => (BookingStatus)Enum.Parse(typeof(BookingStatus),
v.Replace("_", ""), true)
                );
            modelBuilder.Entity<Booking>().HasOne(b => b.Master).WithMany(m =>
m.Bookings).HasForeignKey(b => b.MasterId).OnDelete(DeleteBehavior.SetNull);
            modelBuilder.Entity<BookingService>().HasOne(bs =>
bs.Booking).WithMany(b => b.Services).HasForeignKey(bs => bs.BookingId);
            modelBuilder.Entity<BookingService>().HasOne(bs =>
bs.ServiceItem).WithMany().HasForeignKey(bs => bs.ServiceItemId);
            modelBuilder.Entity<Recommendation>().HasOne(r => r.Booking).WithMany(b
=> b.Recommendations).HasForeignKey(r => r.BookingId);
            modelBuilder.Entity<Recommendation>().HasOne(r =>
r.ServiceItem).WithMany().HasForeignKey(r => r.ServiceItemId);
            modelBuilder.Entity<Consultation>().HasOne(c =>
c.User).WithMany().HasForeignKey(c => c.UserId).OnDelete(DeleteBehavior.Restrict);
            modelBuilder.Entity<Consultation>().HasOne(c =>
c.Master).WithMany().HasForeignKey(c =>
c.MasterId).OnDelete(DeleteBehavior.SetNull);

            modelBuilder.Entity<Master>().HasMany(m => m.ServiceItems).WithMany(s =>
s.Masters).UsingEntity<Dictionary<string, object>>(
                "masterspecializations",
```

```
                j =>
j.HasOne<ServiceItem>().WithMany().HasForeignKey("ServiceItemId"),
                j => j.HasOne<Master>().WithMany().HasForeignKey("MasterId"),
                j => {
j.Property<DateTime>("CreatedAt").HasDefaultValueSql("CURRENT_TIMESTAMP");
j.HasKey("MasterId", "ServiceItemId"); });
        }
    }
      }
```

Лістинг коду Program.cs:

```
using AutoServ.Core.Interfaces;
using AutoServ.Infrastructure.Data;
using AutoServ.Infrastructure.Services;
using AutoServ.Infrastructure.Strategies;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using System.Text;
using System.Text.Json;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString)));

//
builder.Services.AddScoped<IMasterAllocationStrategy, LeastLoadStrategy>();
builder.Services.AddScoped<IScheduleService, ScheduleService>();
builder.Services.AddScoped<IBookingService, BookingService>();
builder.Services.AddScoped<IRecommendationService, RecommendationService>();
builder.Services.AddScoped<IConsultationService, ConsultationService>();
builder.Services.AddScoped<IMasterAllocationService, MasterAllocationService>();

builder.Services.AddScoped<ITokenService, TokenService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IMasterService, MasterService>();
builder.Services.AddScoped<IServiceService, ServiceService>();
builder.Services.AddScoped<IAdminServiceService, AdminServiceService>();


builder.Services.AddScoped<IPaymentService, LiqPayService>();


builder.Services.AddSingleton<IFileStorageService, CloudinaryStorageService>();

var jwtSettings = builder.Configuration.GetSection("JwtSettings");
var secretKey = jwtSettings["SecretKey"];
if (string.IsNullOrEmpty(secretKey)) throw new Exception("JWT SecretKey не
знайдено!");

builder.Services.AddAuthentication(options => { options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme; options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme; })
.AddJwtBearer(options => { options.TokenValidationParameters = new
TokenValidationParameters { ValidateIssuer = true, ValidateAudience = true,
ValidateLifetime = true, ValidateIssuerSigningKey = true, ValidIssuer =
```

```
jwtSettings["Issuer"], ValidAudience = jwtSettings["Audience"], IssuerSigningKey =
new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey)) }; });

builder.Services.AddAuthorization();

builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.PropertyNameCaseInsensitive = true;
        options.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles;
        options.JsonSerializerOptions.Converters.Add(new
JsonStringEnumConverter(JsonNamingPolicy.SnakeCaseLower));
    });
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c => { c.SwaggerDoc("v1", new OpenApiInfo { Title =
"AutoServ API", Version = "v1" }); c.AddSecurityDefinition("Bearer", new
OpenApiSecurityScheme { Name = "Authorization", In = ParameterLocation.Header, Type
= SecuritySchemeType.Http, Scheme = "bearer" }); c.AddSecurityRequirement(new
OpenApiSecurityRequirement { { new OpenApiSecurityScheme { Reference = new
OpenApiReference { Type = ReferenceType.SecurityScheme, Id = "Bearer" } }, new
string[] { } } }); });

builder.Services.AddCors(options => { options.AddDefaultPolicy(policy =>
policy.WithOrigins("http://localhost:3000",
"https://localhost:3000").AllowAnyHeader().AllowAnyMethod()); });
builder.Services.Configure<Microsoft.AspNetCore.Http.Features.FormOptions>(options
=>
{
    options.MultipartBodyLengthLimit = 100 * 1024 * 1024;
});

builder.WebHost.ConfigureKestrel(serverOptions =>
{
    serverOptions.Limits.MaxRequestBodySize = 100 * 1024 * 1024;
});
var app = builder.Build();

if (app.Environment.IsDevelopment()) { app.UseSwagger(); app.UseSwaggerUI(); }

app.UseHttpsRedirection();

app.UseStaticFiles(new StaticFileOptions
{
    OnPrepareResponse = ctx =>
    {
        ctx.Context.Response.Headers.Append("Access-Control-Allow-Origin", "*");
    }
});

app.UseCors();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

# ДОДАТОК Б

# ЛІСТИНГ КОДУ КЛІЄНТСЬКОЇ ЧАСТИНИ

Лістинг коду useBookings.js:

```
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
import api from "../api/api";
import { BOOKING_STATUS, statusMap } from "../constants/statuses";

function getMinutes(timeStr) {
  if (!timeStr) return 0;
  const parts = timeStr.split(':');
  return parseInt(parts[0], 10) * 60 + parseInt(parts[1], 10);
}

export const useBookings = (userId, ordersTab) => {
  const queryClient = useQueryClient();

  const { data: bookings = [], isLoading } = useQuery({
    queryKey: ['userBookings', userId],
    queryFn: () => api.get(`/Bookings/user/${userId}`),
    enabled: !!userId
  });

  const handleMockPayment = useMutation({
    mutationFn: (idsString) => {
      alert("Оплата успішно пройшла! Дякуємо.");
      const targetIds = idsString.split(', ').map(Number);
      queryClient.setQueryData(['userBookings', userId], (oldData) =>
        oldData.map(b => targetIds.includes(b.id) ? { ...b, status:
BOOKING_STATUS.PAID } : b)
      );
    }
  });


  const displayBookings = (() => {
    const parentIdsWithChildren = new Set();
    bookings.forEach(b => { if (b.parentBookingId)
parentIdsWithChildren.add(b.parentBookingId); });

    const allGroups = bookings.reduce((acc, b) => {
      if (b.bookingType === 'service_order' && parentIdsWithChildren.has(b.id))
return acc;
      const groupKey = b.parentBookingId ?
`parent_${b.parentBookingId}_date_${b.bookingDate}` : `self_${b.id}`;
      if (!acc[groupKey]) acc[groupKey] = { bookingDate: b.bookingDate, times: [],
statuses: [], ids: [], serviceObjects: [], totalPrice: 0, bookingTypes: [] };
      if (b.bookingTime) { const timeShort = b.bookingTime.substring(0, 5); if
(!acc[groupKey].times.includes(timeShort)) acc[groupKey].times.push(timeShort); }
      acc[groupKey].ids.push(b.id); acc[groupKey].statuses.push(b.status);
acc[groupKey].totalPrice += b.totalPrice || 0;
      if (b.bookingType && !acc[groupKey].bookingTypes.includes(b.bookingType))
acc[groupKey].bookingTypes.push(b.bookingType);
      if (b.services && b.services.length > 0) { b.services.forEach(s => { const
sName = s.title || s.serviceItemTitle || "Послуга"; if
(!acc[groupKey].serviceObjects.find(x => x.name === sName))
acc[groupKey].serviceObjects.push({ name: sName, status: b.status }); }); }
      if (b.bookingType === "consultation" && !acc[groupKey].serviceObjects.find(x
=> x.name === "Консультація")) acc[groupKey].serviceObjects.push({ name:
"Консультація", status: b.status });
```

```
      return acc;
    }, {});

    return Object.values(allGroups).map(g => {
      let displayTime = "---";
      if (g.times.length > 0) { const sorted = g.times.sort((a, b) => getMinutes(a)
- getMinutes(b)); displayTime = sorted[0] === sorted[sorted.length - 1] ? sorted[0]
: `${sorted[0]} - ${sorted[sorted.length - 1]}`; }

      let aggregateStatus = BOOKING_STATUS.PENDING;
      const uniqueStatuses = [...new Set(g.statuses)];
      if (uniqueStatuses.includes(BOOKING_STATUS.CANCELLED)) aggregateStatus =
BOOKING_STATUS.CANCELLED;
      else if (uniqueStatuses.every(s => s === BOOKING_STATUS.PAID)) aggregateStatus
= BOOKING_STATUS.PAID;
      else if (uniqueStatuses.every(s => s === BOOKING_STATUS.COMPLETED))
aggregateStatus = BOOKING_STATUS.COMPLETED;
      else if (uniqueStatuses.includes(BOOKING_STATUS.IN_PROGRESS)) aggregateStatus
= BOOKING_STATUS.IN_PROGRESS;
      else if (uniqueStatuses.includes(BOOKING_STATUS.CONFIRMED)) aggregateStatus =
BOOKING_STATUS.CONFIRMED;

      const st = statusMap[aggregateStatus] || statusMap[BOOKING_STATUS.PENDING];
      return { ids: g.ids.join(", "), bookingDate: g.bookingDate, displayTime,
serviceObjects: g.serviceObjects, displayPrice: g.totalPrice > 0 ?
`${g.totalPrice.toLocaleString("uk-UA")} грн` : "Безкоштовно", statusLabel:
st.label, statusBg: st.bg, aggregateStatus };
    }).filter(g => {
      if (ordersTab === "active") return [BOOKING_STATUS.PENDING,
BOOKING_STATUS.CONFIRMED, BOOKING_STATUS.IN_PROGRESS,
BOOKING_STATUS.COMPLETED].includes(g.aggregateStatus);
      return g.aggregateStatus === BOOKING_STATUS.PAID || g.aggregateStatus ===
BOOKING_STATUS.CANCELLED;
    });
  })();

  return { bookings, displayBookings, handleMockPayment: handleMockPayment.mutate,
isLoading };
};
```

Лістинг коду useCarData.js:

```
import { useState, useEffect } from "react";
import api from "../api/api";

export const useCarData = (userId) => {
  const [carData, setCarData] = useState({
    carBrand: "", carModel: "", carYear: "", carEngine: "", carPlate: ""
  });

  useEffect(() => {
    if (!userId) return;

    api.get(`/Profile/${userId}`).then(data => {
      setCarData({
        carBrand: data?.carBrand || "",
        carModel: data?.carModel || "",
        carYear: data?.carYear?.toString() || "",
        carEngine: data?.carEngine || "",
        carPlate: data?.carPlate || ""
      });
    }).catch(err => console.error("Помилка завантаження даних авто:", err));
  }, [userId]);
```

```
  return { carData, setCarData };
};
```

Лістинг коду useRecommendations.js:

```
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
import api from "../api/api";

export const useRecommendations = (userId) => {
  const queryClient = useQueryClient();

  const { data: recommendations = [] } = useQuery({
    queryKey: ['recommendations', userId],
    queryFn: () => api.get(`/Recommendations/user/${userId}`).then(data =>
      (data || []).map(r => ({
        ...r,
        price: r.price || r.priceFrom || 0,
        title: r.title || "Послуга",
        id: r.id,
        status: r.status || "pending",
        durationMinutes: r.durationMinutes || 60
      }))
    ),
    enabled: !!userId
  });
  const groupedRecommendations = recommendations.reduce((acc, rec) => {
    if (rec.status === "booked" || rec.status === "declined") return acc;
    if (!acc[rec.bookingId]) {
      acc[rec.bookingId] = {
        bookingId: rec.bookingId,
        bookingDate: rec.bookingDate, // "dd.MM.yyyy"
        masterId: rec.masterId,
        items: [],
        recIds: []
      };
    }
    acc[rec.bookingId].items.push(rec);
    acc[rec.bookingId].recIds.push(rec.id);
    return acc;
  }, {});

  const previewBatch = async ({ recommendationIds, timeStrategy }) => {
    try {
      const response = await api.post(`/Recommendations/preview-batch`, {
recommendationIds, timeStrategy });
      return { isSuccess: true, ...response };
    } catch (error) {
      return { isSuccess: false, error: error?.response?.data?.message || "Не
вдалося підібрати час." };
    }
  };
  // 2. final save
  const acceptBatchMutation = useMutation({
    mutationFn: (data) => api.post(`/Recommendations/accept-batch`, data),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['recommendations', userId] });
      queryClient.invalidateQueries({ queryKey: ['userBookings', userId] });
    }
  });

  const acceptRecommendationsBatch = async ({ recommendationIds, timeStrategy }) =>
{
    try {
```

```
      const result = await acceptBatchMutation.mutateAsync({ recommendationIds,
timeStrategy });
      return { isSuccess: true, ...result };
    } catch (error) {
      return { isSuccess: false, error: error?.response?.data?.message || "Помилка
запису." };
    }
  };
  return {
    groupsList: Object.values(groupedRecommendations),
    hasProcessedRecs: recommendations.some(r => r.status === "booked"),
    previewBatch,
    acceptRecommendationsBatch,
    fetchRecommendations: () => queryClient.invalidateQueries({ queryKey:
['recommendations', userId] }),
    isAccepting: acceptBatchMutation.isPending
  };
};
```

Лістинг коду сторінки MasterSchedulePage.jsx:

```
import HeroSection from "../components/home/HeroSection";
import FeaturesSection from "../components/home/FeaturesSection";
import ServicesSection from "../components/home/ServicesSection";
import WhyUsSection from "../components/home/WhyUsSection";
import CTASection from "../components/home/CTASection";

export default function Home() {
  return (<><HeroSection /><FeaturesSection /><ServicesSection /><WhyUsSection
/><CTASection /></>);
}
import React, { useState } from 'react';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-
import ScheduleGrid from '../components/master/ScheduleGrid';
import ActionsPanel from '../components/master/ActionsPanel';
import api from '../api/api';
import { useAuth } from '../context/AuthContext';

export default function MasterSchedulePage() {
  const { user } = useAuth();
  const queryClient = useQueryClient();
  const generateDays = () => {
    const days = [];
    const options = { weekday: 'short', day: 'numeric', month: 'short' };
    for (let i = 0; i < 7; i++) {
      const date = new Date(); date.setDate(date.getDate() + i);
      days.push({ label: date.toLocaleDateString('uk-UA', options), dateValue:
date.toISOString().split('T')[0] });
    }
    return days;
  };

  const days = generateDays();
  const [selectedDate, setSelectedDate] = useState(days[0]);
  const [selectedTask, setSelectedTask] = useState(null);

  const { data: scheduleData = [], isLoading: loading } = useQuery({
    queryKey: ['schedule', user?.id, selectedDate.dateValue], // унікальний ключ
    queryFn: () => api.get(`/Bookings/my-
schedule?userId=${user.id}&date=${selectedDate.dateValue}`),
    refetchInterval: 30000,
  });
```

```
  const updateStatusMutation = useMutation({
    mutationFn: ({ taskId, newStatus }) => api.patch(`/Bookings/${taskId}/status`, {
status: newStatus }),
    onSuccess: () => {

      queryClient.invalidateQueries({ queryKey: ['schedule'] });
    },
    onError: (err) => alert(`Помилка: ${err.message}`)
  });
  const handleStatusUpdate = (newStatus) => {
    if (!selectedTask) return;
    updateStatusMutation.mutate({ taskId: selectedTask.id, newStatus });
  };
  return (
    <div className="w-full">
      <div className="mb-8">
        <h1 className="text-3xl font-bold text-gray-900">Мій графік</h1>
        <p className="text-gray-400 text-sm mt-1">Графік роботи на
{selectedDate.label}</p>
      </div>
      <div className="flex items-center gap-2 mb-6 border-b border-gray-100 pb-2
overflow-x-auto">
        {days.map((day) => (
          <button
            key={day.dateValue}
            onClick={() => setSelectedDate(day)}
            className={`px-4 py-2 text-sm font-medium transition-all rounded-md
whitespace-nowrap ${
              selectedDate.dateValue === day.dateValue ? 'bg-[#E54D43] text-white
shadow-md' : 'text-gray-500 hover:bg-gray-50'
            }`}
          >
            {day.label}
          </button>
        ))}

      </div>
      {loading ? (
        <p className="text-gray-500 text-center py-10">Завантаження...</p>
      ) : scheduleData.length > 0 ? (
        <div className="flex flex-col lg:flex-row gap-8">

          <ScheduleGrid data={scheduleData} selectedTask={selectedTask}
onSelectTask={setSelectedTask} /
          <ActionsPanel
            selectedTask={selectedTask}
            onUpdateStatus={handleStatusUpdate}
            isUpdating={updateStatusMutation.isLoading}
          />
        </div>
      ) : (
        <div className="flex flex-col lg:flex-row gap-8">
          <div className="flex-1 text-center py-20 text-gray-400 border border-
dashed rounded-xl">Немає замовлень</div>
          <ActionsPanel selectedTask={null} />
        </div>
      )}
    </div>
  );
}
```

Лістинг коду файлу маршрутизації App.jsx:

```
import { Routes, Route, Navigate } from "react-router-dom";
import MainLayout from "./layouts/MainLayout";
import MasterLayout from "./layouts/MasterLayout";
import AdminLayout from "./layouts/AdminLayout";

import Home from "./pages/Home";
import Services from "./pages/Services";
import ServiceDetailPage from "./components/home/ServiceDetailPage";
import Profile from "./pages/Profile";
import OnlineConsultation from "./pages/OnlineConsultation";
import MasterPage from "./pages/MasterPage";
import MasterSchedulePage from "./pages/MasterSchedulePage";

import AdminDashboard from "./pages/admin/AdminDashboard";
import AdminBookings from "./pages/admin/AdminBookings";
import AdminClients from "./pages/admin/AdminClients";
import AdminMasters from "./pages/admin/AdminMasters";
import AdminSchedule from "./pages/admin/AdminSchedule";
import AdminServices from "./pages/admin/AdminServices";

import ProtectedRoute from "./components/ProtectedRoute";
import { useAuth } from "./context/AuthContext";

export default function App() {
  const { user } = useAuth();
  const isMaster = user?.role === "master";
  const isAdmin = user?.role === "admin";
  return (
    <Routes>
      {/* --- main page clients --- */}
      <Route element={<MainLayout />}>
        <Route path="/" element={isMaster ? <Navigate to="/master/schedule" replace
/> : (isAdmin ? <Navigate to="/admin/dashboard" replace /> : <Home />)} />
        <Route path="/services" element={<Services />} />
        <Route path="/services/:id" element={<ServiceDetailPage />} />
        <Route path="/booking" element={<Navigate to="/services" replace />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/consultation" element={<OnlineConsultation />} />
      </Route>

      {/* --- page master --- */}
      <Route path="/master" element={<ProtectedRoute
allowedRole="master"><MasterLayout /></ProtectedRoute>}>
        <Route index element={<Navigate to="schedule" replace />} />
        <Route path="schedule" element={<MasterSchedulePage />} />
        <Route path="requests" element={<MasterPage />} />
      </Route>
      <Route path="/admin" element={<ProtectedRoute allowedRole="admin"><AdminLayout
/></ProtectedRoute>}>
        <Route index element={<Navigate to="dashboard" replace />} />
        <Route path="dashboard" element={<AdminDashboard />} />
        <Route path="bookings" element={<AdminBookings />} />
        <Route path="clients" element={<AdminClients />} />
        <Route path="masters" element={<AdminMasters />} />
        <Route path="schedule" element={<AdminSchedule />} />
        <Route path="services" element={<AdminServices />} />
      </Route>
      <Route path="*" element={<Navigate to="/" replace />} />
    </Routes>
  );
}
```