МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 17.00.00.000 ПЗ

Група ІПм-24-1

Гуйдаш Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Гуйдаш Владислав Андрійович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та засоби підключень до баз даних

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Гуйдаш В.А.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Козак Олексій Федорович, к.т.н., ст.викладач

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Освітній рівень магістр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ
Гуйдашу Владиславу Андрійовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі, методи та засоби підключень до баз даних”

керівник проєктів (роботи) Козак Олексій Федорович, к.т.н., ст. викл.

затверджені наказом закладу вищої освіти від “ 05 ” листопада 2025 р. № 695/7

2. Строк подання студентом проєктів (роботи) 15 грудня 2025 р.

3. Вихідні дані до проєктів (роботи) Теоретичні відомості та практичні матеріал методів засобів підключення до баз даних, принципи клієнт-серверної архітектури

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

1. Теоретичні відомості, поняття та класифікація баз даних. Їх практичне застосування, моделі та засоби їх використання
2. Огляд існуючих концепцій , рішень та сервісів в даній області
3. Побудова моделі або алгоритму власного рішення даних. Застосування підключень баз даних на прикладі веб-ресурсу приватної клініки.
4. Документування реалізації власного оригінального рішення вибраними засобами. Практична демонстрація програмного продукту.

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

1. Безпека баз даних (рис. 1.1, ст. 14)
2. Моделі баз даних (рис. 1.2, ст. 15)
3. Клієнт-серверна архітектура підключення до БД (рис. 1.3, ст. 18)
4. Рівень прикладної логіки (рис. 1.4, ст. 22)
5. Схема бази даних (рис. 4.1, ст. 47)

6. Консультанти розділів проєктів (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.		

7. Дата видачі завдання 02 вересня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєктів (роботи)	Строк виконання етапів проєктів	Примітка
1	Визначення та обгрунтування теми роботи	20.09.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	01.10.2025	виконано
3	Побудова моделі або алгоритму власного рішення	12.10.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.10.2025	виконано
5	Формулювання вимог та алгоритмів функціонування системи	05.11.2025	виконано
6	Програмна реалізація рішення	22.11.2025	виконано
7	Оформлення пояснювальної записки кваліфікаційної роботи	15.12.2025	виконано

Студент _____Гуйдаш В.А._____
(підпис)

Керівник роботи _____Козак О.Ф_____
(підпис)

АНОТАЦІЯ

Магістерська робота: 76с., 19 рис., 38 джерел.

Тема: Моделі, методи та засоби підключень до баз даних

Об’єкт дослідження: Бази даних та системи керування базами даних як основа зберігання, обробки й управління структурованою інформацією в сучасних програмних системах.

Мета роботи: Дослідження, аналіз і вдосконалення моделей та методів підключення до баз даних, а також розробка оптимізованого механізму взаємодії програмного застосунку з системою керування базами даних з урахуванням вимог безпеки.

Предмет дослідження: Методи та засоби реалізації, підключення й адміністрування баз даних, а також механізми забезпечення цілісності, безпеки та продуктивності при роботі з даними.

Результати дослідження:

У роботі проаналізовано сучасні методи та засоби реалізації баз даних, параметризовані підключення, пулінг з’єднань, ORM-технології та клієнт-серверні протоколи взаємодії. Визначено їхні переваги. Практична реалізація й перевірка запропонованих рішень здійснена на прикладі інформаційної системи приватної клініки з урахуванням специфіки роботи з медичними даними.

Висновок:

У результаті проведеного дослідження розроблено та обґрунтовано вдосконалену модель підключення й адміністрування бази даних, яка оптимізує взаємодію між застосунком і СКБД, зменшує втрати продуктивності та підвищує рівень захищеності даних.

ПІДКЛЮЧЕННЯ ДО БАЗ ДАНИХ, МОДЕЛІ ДОСТУПУ, ПУЛІНГ З’ЄДНАНЬ, ORM, API, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ТРАНЗАКЦІЇ, БЕЗПЕКА ДАНИХ.

ANNOTATION

Master's work: 76p., 19 fig., 38 sources

Topic: Models, Methods, and Tools for Database Connectivity

Object of the study: Databases and database management systems as the basis for storage, processing, and management of structured information in modern software systems.

Subject of the study: Methods and tools for database implementation, connectivity, and administration, as well as mechanisms for ensuring data integrity, security, and performance during data processing.

Purpose of the work: To study, analyze, and improve models and methods of database connectivity, as well as to develop an optimized mechanism for interaction between a software application and a database management system, taking into account security, scalability.

Research results:

The paper analyzes modern methods and tools for database implementation, including parameterized connections, connection pooling, Their advantages, disadvantages, and areas of appropriate application are identified. Practical implementation and validation of the proposed solutions were carried out using the example of an information system for a private clinic, considering the specifics of handling medical data.

Conclusion:

As a result of the conducted research, an improved model of database connectivity and administration was developed and substantiated. The proposed solution optimizes interaction between the application and the DBMS, reduces performance overhead, and enhances the level of data security.

DATABASE CONNECTIVITY, ACCESS MODELS, CONNECTION POOLING, ORM, API, CLIENT–SERVER ARCHITECTURE, TRANSACTIONS, DATA SECURITY.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API — прикладний програмний інтерфейс.

RESTful API — прикладний програмний інтерфейс передачі репрезентативного стану (REST).

SQL — мова структурованих запитів.

СУБД — система управління базами даних.

БД — база даних.

ORM — об’єктно-реляційне відображення.

JDBC — Java Database Connectivity, інтерфейс підключення Java-застосунків до БД.

ODBC — Open Database Connectivity, універсальний інтерфейс доступу до БД.

ACID — набір властивостей транзакцій: атомарність, узгодженість, ізольованість, довговічність.

CRUD — базові операції з даними: створення, читання, оновлення, видалення.

HTTP — протокол передавання гіпертексту.

HTTPS — захищений протокол передавання гіпертексту (HTTP + шифрування).

TCP — протокол керування передаванням даних.

IP — протокол інтернету.

DNS — система доменних імен.

URI — уніфікований ідентифікатор ресурсу.

URL — уніфікований локатор ресурсу.

JSON — текстовий формат обміну даними на основі JavaScript-нотації.

XML — розширювана мова розмітки.

JWT — JSON Web Token, токен для автентифікації та авторизації.

SSL/TLS — криптографічні протоколи захисту з’єднання.

ЗМІСТ

РОЗДІЛ 4

ВСТУП

Актуальність роботи

З невпинним розвитком сучасних цифрових технологій та активним впровадженням інформаційних систем у різні сфери суспільного життя питання ефективного зберігання, обробки та захисту даних набувають особливої актуальності. Значна кількість сервісів функціонує у веб-середовищі та обслуговує великі обсяги інформації, що зумовлює необхідність використання надійних методів проєктування й реалізації баз даних, а також безпечних механізмів доступу до них.

Особливо чутливою є сфера охорони здоров’я, де інформаційні системи працюють з персональними та медичними даними пацієнтів. У таких умовах бази даних повинні забезпечувати не лише цілісність і доступність інформації, але й відповідати вимогам конфіденційності, встановленим законодавством та міжнародними стандартами. Порушення правил зберігання або обробки даних може призвести до витоку інформації, фінансових санкцій і втрати довіри користувачів.

Сучасні веб-ресурси медичних закладів будуються на основі клієнт-серверної архітектури, де база даних є центральним елементом системи. Саме вона забезпечує зберігання інформації про пацієнтів, лікарів, записи на прийом, історії звернень та інші критично важливі дані. Тому вибір методів реалізації баз даних, способів підключення до них, організації структури даних і контролю доступу має ключове значення для стабільної та безпечної роботи таких систем.

Додатковим викликом є зростання вимог до механізмів авторизації та аутентифікації користувачів веб-ресурсів.

У межах даної роботи розглядаються методи та засоби реалізації баз даних у веб-ресурсі приватної медичної клініки. Практична частина присвячена розробці та програмній імплементації бази даних і механізмів взаємодії з нею у складі веб-системи, що забезпечує обробку даних користувачів, контроль доступу та можливість масштабування.

Мета і задачі дослідження

Метою даної роботи є дослідження, аналіз та вдосконалення моделей і методів підключення програмних застосунків до баз даних з урахуванням сучасних вимог до безпеки, продуктивності та масштабованості. У процесі роботи передбачається вивчення підходів до організації взаємодії між серверною частиною застосунку та системою керування базами даних, аналіз наявних технологічних рішень і вибір оптимальних інструментів для реалізації такого зв’язку. Окремим завданням є:

1. розробка та опис оптимізованого механізму доступу до даних;
2. забезпечення надійної обробки запитів;
3. захист інформації від несанкціонованого доступу;
4. стабільна робота системи в умовах зростаючого навантаження.
5. реалізація веб-застосунку клініки із підключенням бази даних.

Об’єкт і предмет дослідження

Об’єктом дослідження у даній роботі є бази даних та системи керування базами даних як фундаментальна складова сучасних програмних систем, призначена для зберігання, обробки, оновлення та управління структурованою інформацією. У рамках дослідження розглядаються бази даних як централізовані сховища даних, що забезпечують узгодженість, доступність і надійність інформації.

Предметом дослідження є методи та засоби реалізації підключення до баз даних, їх адміністрування та використання у складі програмних систем, а також механізми забезпечення цілісності, безпеки та ефективності роботи з даними. У межах предмету досліджуються технології організації доступу до баз даних, способи управління користувачами та правами доступу, методи захисту інформації, а також підходи до оптимізації запитів і підвищення продуктивності СКБД.

Методи дослідження

У роботі застосовано системний підхід до побудови архітектури веб-системи, аналіз і порівняння сучасних технологій баз даних, моделювання структури даних, а також експериментальну перевірку працездатності реалізованого програмного рішення.

Наукова новизна

Запропоновано підхід до реалізації бази даних веб-ресурсу медичного закладу, що поєднує оптимізовану структуру зберігання даних, безпечні методи підключення та механізми контролю доступу.

Практичне значення

Розроблене програмне рішення може бути використане як основа для впровадження або модернізації веб-ресурсів приватних клінік з метою підвищення ефективності роботи з даними та рівня інформаційної безпеки.

Особистий внесок

Автором виконано аналіз існуючих методів реалізації баз даних у веб-додатках, спроєктовано структуру бази даних приватної медичної клініки та здійснено її програмну імплементацію у складі веб-ресурсу.

Структура магістерської роботи

Магістерська робота викладена на сторінках 76 друкованого тексту та складається зі вступу, чотирьох розділів, висновків і списку використаних джерел (38 найменувань). Робота містить 19 рисунків.

РОЗДІЛ 1
ПОНЯТТЯ ТА КЛАСИФІКАЦІЯ БАЗ ДАНИХ

1.1 Поняття та тлумачення бази даних

У процесі розвитку інформаційних технологій обсяги даних, що створюються, обробляються та зберігаються, зростають надзвичайно швидкими темпами. Для ефективного управління цією інформацією виникла необхідність у спеціальних засобах її організації, збереження та доступу. Саме такою формою зберігання інформації є база даних[2].

База даних — це впорядкована сукупність логічно пов’язаних між собою даних, що зберігаються в електронному вигляді та призначені для багаторазового використання різними користувачами й програмними засобами. Дані в базі організовуються таким чином, щоб забезпечити їхню цілісність, актуальність, захищеність і швидкий доступ.

На відміну від традиційних способів зберігання інформації у вигляді окремих файлів, база даних дозволяє уникнути надмірності інформації, дублювання записів і помилок під час оновлення. Це досягається завдяки централізованому зберіганню даних та використанню єдиних правил доступу до них.

Важливою характеристикою бази даних є її незалежність від прикладних програм. Це означає, що зміни у структурі даних можуть відбуватися без необхідності суттєвого переписування програмного коду, який використовує ці дані. Такий підхід підвищує гнучкість інформаційних систем і спрощує їх супровід.

База даних не є лише сховищем інформації, а виступає складовою частиною інформаційної системи, що функціонує разом із системою управління базами даних (СУБД). СУБД забезпечує створення, модифікацію, пошук, оновлення та видалення даних, а також контролює доступ користувачів і підтримує цілісність інформації.

Однією з ключових особливостей баз даних є структурованість інформації. Дані організовуються відповідно до певної моделі, яка визначає спосіб їх зберігання та взаємозв’язки між окремими елементами. Найпоширенішою є реляційна модель, у якій дані подаються у вигляді таблиць, що складаються з рядків і стовпців. Кожен рядок таблиці відповідає окремому запису, а кожен стовпець — певному атрибуту.

Бази даних забезпечують підтримку цілісності даних, яка полягає у збереженні коректності та узгодженості інформації. Для цього використовуються спеціальні механізми, зокрема первинні та зовнішні ключі, обмеження цілісності, тригери та транзакції. Завдяки цьому система запобігає появі суперечливих або помилкових даних.

Ще однією важливою характеристикою баз даних є багатокористувацький режим роботи. Декілька користувачів можуть одночасно працювати з одними й тими ж даними без ризику їх пошкодження. Це забезпечується механізмами блокування, керування транзакціями та журналювання змін.

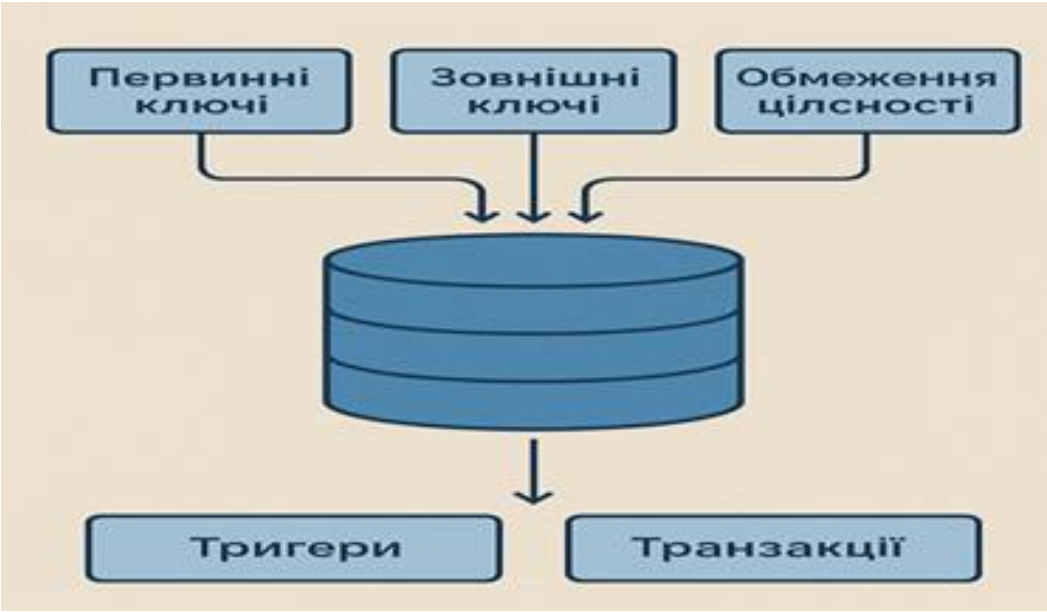

Рис. 1.1. Безпека баз даних

З точки зору безпеки, база даних передбачає контроль доступу до інформації. Користувачам надаються різні рівні прав, що дозволяє обмежити перегляд, зміну або видалення даних відповідно до їхніх повноважень. Це особливо важливо для корпоративних і державних інформаційних систем, де зберігається конфіденційна інформація.

Таким чином, база даних є ключовим елементом сучасних інформаційних систем, який забезпечує ефективне зберігання, обробку та захист великих обсягів інформації. Використання баз даних дозволяє підвищити надійність, масштабованість та продуктивність програмних рішень, а також створює основу для розвитку аналітичних і автоматизованих систем управління.

1.2 Класифікація баз даних та їх ієрархія

Різноманітність задач, для розв’язання яких використовуються бази даних, зумовлює існування великої кількості їх типів. Класифікація баз даних дозволяє систематизувати підходи до зберігання та обробки інформації, а також обрати оптимальну модель бази даних відповідно до конкретних вимог інформаційної системи. Бази даних класифікуються за кількома основними ознаками, серед яких модель даних, рівень ієрархії, спосіб зберігання, призначення та архітектура.

Рис. 1.2. Моделі баз даних

Однією з ключових ознак класифікації є модель даних, яка визначає логічну структуру бази даних та взаємозв’язки між її елементами[1].

Ієрархічна модель базується на деревоподібній структурі, у якій кожен елемент має одного батьківського вузла та може мати декілька дочірніх. Така організація чітко відображає підпорядкованість даних і є ефективною для систем із жорстко визначеною структурою. Недоліком ієрархічних баз даних є складність реалізації зв’язків типу «багато до багатьох» та низька гнучкість при зміні структури.

Мережева модель є розширенням ієрархічної та дозволяє кожному елементу мати кілька батьківських зв’язків. Це забезпечує більш гнучку структуру та можливість опису складних взаємозв’язків між даними. Водночас така модель є складною для проєктування та адміністрування.

Реляційна модель є найпоширенішою в сучасних інформаційних системах. Дані в ній організовані у вигляді таблиць, між якими встановлюються зв’язки за допомогою ключів. Основними перевагами реляційних баз даних є простота використання, наочність структури, підтримка стандартної мови SQL та висока надійність.

У таких базах даних інформація зберігається у вигляді об’єктів, що поєднують дані та методи їх обробки. Ця модель використовується переважно в складних програмних системах, де важливо зберігати складні структури даних.

NoSQL бази даних виникли як відповідь на потребу обробки великих обсягів неструктурованих або напівструктурованих даних. До них належать документоорієнтовані, графові, колонкові та key-value бази даних. Вони забезпечують високу масштабованість і продуктивність, однак часто жертвують строгими правилами цілісності.

Ієрархія баз даних розглядається також з точки зору рівнів подання та абстракції даних. Класично виділяють три рівні ієрархії:

Описує подання даних для конкретних користувачів або груп користувачів. Кожен користувач бачить лише ту частину бази даних, яка необхідна для виконання його завдань.

Відображає логічну структуру всієї бази даних, включаючи типи даних, зв’язки між ними та обмеження цілісності. Цей рівень не залежить від фізичного способу зберігання інформації.

Визначає спосіб фізичного зберігання даних на носіях інформації, структуру файлів, індекси та методи доступу. Саме на цьому рівні оптимізується продуктивність роботи бази даних.

Така ієрархічна структура забезпечує незалежність даних і спрощує супровід інформаційних систем.

1.3 Методи підключень баз даних, специфіка та їх програмне адміністрування.

Реалізація баз даних є одним із ключових етапів створення інформаційної системи, оскільки саме на цьому етапі концептуальні та логічні моделі даних перетворюються на функціонуюче програмно-апаратне рішення. Від правильності реалізації бази даних, обраних методів підключення та організації адміністрування залежить продуктивність, надійність і безпека всієї інформаційної системи.

Реалізація бази даних полягає у фізичному створенні структури зберігання даних відповідно до логічної моделі, обраної на етапі проєктування. Вона включає створення таблиць, визначення типів даних, встановлення ключів, зв’язків та обмежень цілісності.

На цьому етапі здійснюється вибір конкретної системи управління базами даних, що визначається вимогами до продуктивності, масштабованості, безпеки та інтеграції з іншими системами. Найпоширенішими є реляційні СУБД, які реалізують табличну модель з використанням мови структурованих запитів SQL.

Фізична реалізація бази даних також передбачає оптимізацію структури зберігання інформації. Для цього застосовуються індекси, які пришвидшують доступ до даних, а також механізми розподілу даних на файли та табликові простори. Важливою складовою реалізації є забезпечення цілісності даних, що

досягається шляхом використання первинних і зовнішніх ключів, унікальних обмежень та перевірок допустимих значень.

Окрім структури даних, реалізація бази даних включає створення службових об’єктів, таких як представлення, збережені процедури та тригери. Вони дозволяють централізувати бізнес-логіку, зменшити навантаження на прикладний рівень та підвищити безпеку доступу до інформації.

Специфіка підключень до баз даних6:

Підключення до бази даних є процесом встановлення логічного з’єднання між прикладною програмою та СУБД з метою виконання операцій над даними. Специфіка підключень визначається архітектурою системи, типом бази даних та вимогами до безпеки.

У сучасних інформаційних системах найчастіше використовується клієнт-серверна або багаторівнева архітектура, у якій доступ до бази даних здійснюється через прикладний сервер.

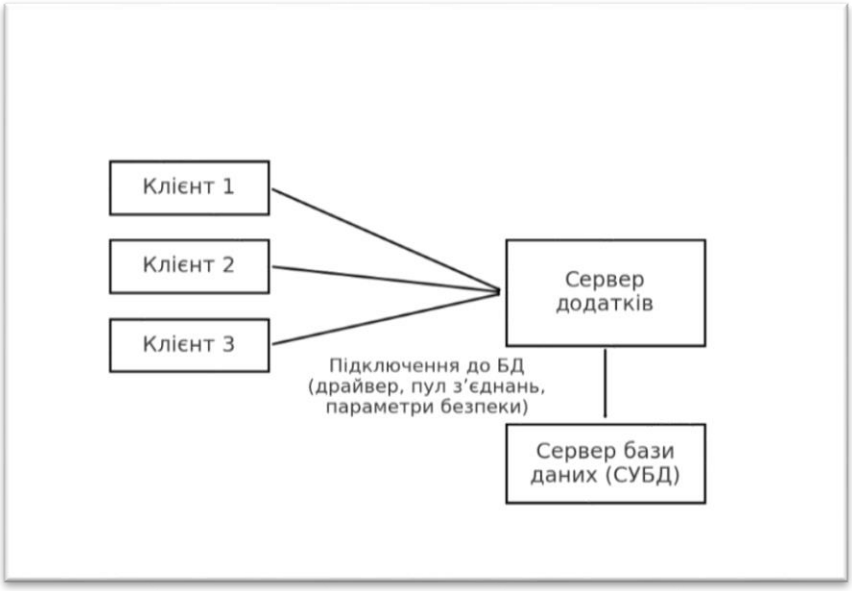

Рис. 1.3. Клієнт-серверна архітектура підключення до БД

Такий підхід дозволяє обмежити прямий доступ користувачів до даних і забезпечити додатковий рівень захисту.

Підключення реалізується за допомогою спеціальних драйверів і програмних інтерфейсів доступу, які забезпечують взаємодію між програмним кодом і СУБД. До параметрів підключення належать адреса сервера, порт, ім’я бази даних, облікові дані користувача та параметри безпеки.

Важливим аспектом є керування сесіями підключення. У багатокористувацьких системах застосовується механізм пулінгу з’єднань, який дозволяє повторно використовувати вже відкриті підключення, зменшуючи навантаження на сервер бази даних і підвищуючи продуктивність системи.

Особливу увагу приділяють безпеці підключень. Для цього використовуються захищені канали передачі даних, механізми шифрування та обмеження доступу за ролями. Некоректно організовані підключення можуть стати причиною витоку інформації або порушення цілісності даних.

Адміністрування баз даних є безперервним процесом, спрямованим на підтримку стабільної роботи СУБД, забезпечення безпеки даних та оптимальної продуктивності. Основна відповідальність за ці процеси покладається на адміністратора баз даних.

До ключових завдань адміністрування належить встановлення та налаштування СУБД, створення та супровід баз даних, а також керування користувачами та їх правами доступу. Розмежування доступу дозволяє мінімізувати ризик несанкціонованих дій та помилок користувачів.

Невід’ємною частиною адміністрування є резервне копіювання та відновлення даних. Регулярне створення резервних копій дозволяє відновити інформацію у разі апаратних збоїв, програмних помилок або зовнішніх загроз. Процеси резервного копіювання мають бути автоматизованими та перевіреними на практиці.

Моніторинг продуктивності бази даних дозволяє виявляти вузькі місця в роботі системи. Для цього аналізуються запити, навантаження на сервер, використання пам’яті та дискових ресурсів. За результатами моніторингу виконуються заходи з оптимізації, зокрема перегляд структури індексів і оптимізація запитів.

Адміністрування також включає забезпечення цілісності та узгодженості даних. Для цього застосовуються механізми транзакцій, журналювання змін та автоматичного відновлення після збоїв. Вони гарантують збереження коректного стану бази даних навіть у разі аварійного завершення роботи системи.

Взаємозв’язок реалізації, підключень та адміністрування: реалізація бази даних, організація підключень та адміністрування є взаємопов’язаними складовими єдиного процесу функціонування інформаційної системи. Неправильно реалізована структура даних ускладнює адміністрування та негативно впливає на продуктивність підключень. У свою чергу, неефективне адміністрування може нівелювати переваги якісного проєктування та реалізації.

Комплексний підхід до реалізації баз даних передбачає узгодженість усіх етапів — від проєктування до експлуатації. Це забезпечує стабільність роботи системи, захист інформації та можливість подальшого розвитку інформаційної інфраструктури.

1.4 Засоби та моделі підключення баз даних до середовищ.

Підключення баз даних до програмних середовищ є ключовим етапом розробки інформаційних систем, оскільки саме від обраних засобів і моделей взаємодії залежить ефективність обробки даних, масштабованість системи, рівень безпеки та стабільність її функціонування. У сучасних програмних рішеннях бази даних зазвичай інтегруються з прикладними середовищами за допомогою стандартизованих інтерфейсів і протоколів, що забезпечують незалежність прикладної логіки від конкретної системи керування базами даних.

Найпоширенішими засобами підключення баз даних є програмні інтерфейси доступу до даних, які реалізують уніфіковані способи виконання SQL-запитів і обробки результатів. До таких засобів належать ODBC (Open Database Connectivity), JDBC (Java Database Connectivity) та сучасні драйвери

доступу до баз даних, що входять до складу веб-фреймворків і серверних платформ.

ODBC є універсальним стандартом, який дозволяє прикладним програмам взаємодіяти з різними СКБД без необхідності зміни прикладного коду. JDBC використовується у середовищах, побудованих на платформі Java, і забезпечує об’єктно-орієнтований підхід до роботи з базами даних. У веб-додатках, реалізованих із використанням мов програмування Python, PHP або JavaScript, широко застосовуються нативні драйвери та бібліотеки, що оптимізовані під конкретні СКБД, такі як MySQL, PostgreSQL або Microsoft SQL Server.

Окрім низькорівневих драйверів, дедалі більшої популярності набувають ORM-засоби (Object-Relational Mapping), які забезпечують відображення реляційних структур бази даних у вигляді об’єктних моделей прикладного середовища.

Використання ORM спрощує розробку, зменшує кількість помилок у запитах і підвищує читабельність коду, однак може впливати на продуктивність у системах з великими обсягами даних.

Моделі підключення баз даних визначають архітектурний підхід до організації взаємодії між прикладним середовищем і СКБД. Однією з базових моделей є модель прямого підключення, за якої клієнтський додаток безпосередньо встановлює з’єднання з базою даних. Такий підхід є простим у реалізації, проте має обмеження щодо безпеки та масштабованості, особливо у веб-середовищах з великою кількістю одночасних користувачів.

Більш поширеною є клієнт-серверна модель, у якій доступ до бази даних здійснюється через сервер прикладної логіки. У цьому випадку клієнт взаємодіє з веб-сервером або API, а вже сервер виконує підключення до бази даних, обробляє запити та повертає результати у зручному форматі.

Така модель дозволяє централізовано керувати доступом до даних, реалізовувати механізми кешування та підвищувати рівень захисту інформації.

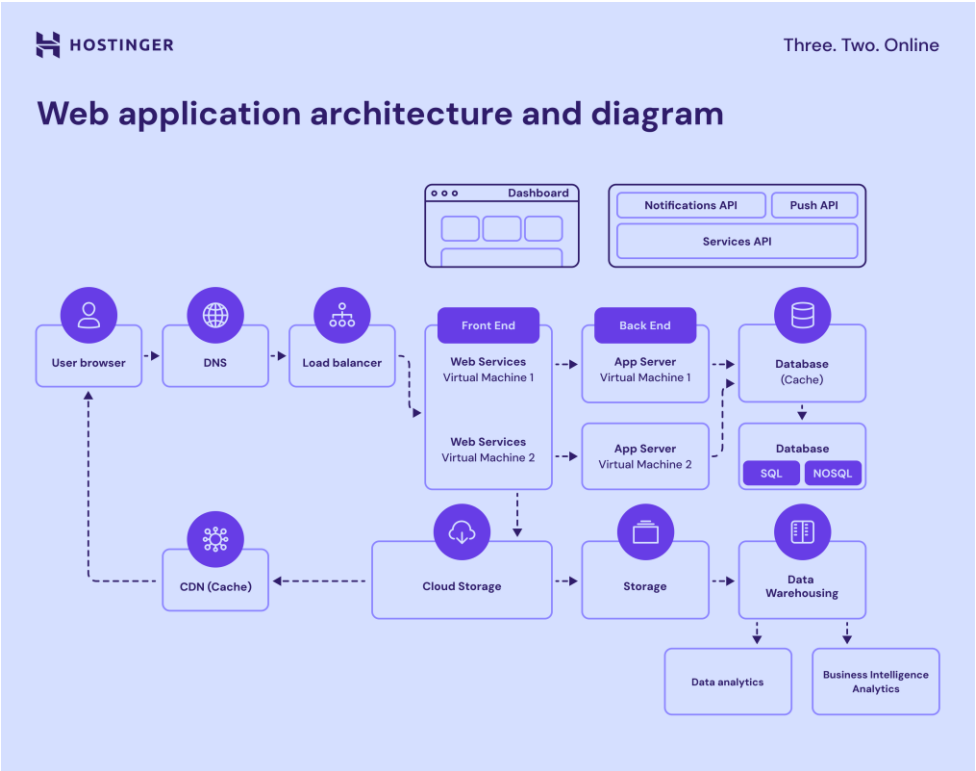

Рис. 1.4. Рівень прикладної логіки

У складніших системах застосовується багаторівнева архітектура, зокрема трирівнева модель, яка передбачає чітке розмежування між рівнем представлення, бізнес-логіки та рівнем доступу до даних. У межах цієї моделі підключення до бази даних інкапсулюється у спеціалізованому модулі або сервісі, що спрощує супровід системи та її подальше розширення.

Важливим аспектом підключення баз даних є управління з'єднаннями, яке включає використання пулів з'єднань. Пул з'єднань дозволяє повторно використовувати встановлені з'єднання з базою даних, зменшуючи накладні витрати на їх створення та закриття. Це особливо актуально для веб-ресурсів, які обслуговують значну кількість паралельних запитів.

Безпека підключення до бази даних забезпечується застосуванням механізмів автентифікації, розмежування прав доступу, шифрування з'єднань та захисту від SQL-ін'єкцій. Рекомендованим підходом є використання параметризованих запитів і збережених процедур, а також обмеження доступу прикладного середовища до бази даних лише необхідним набором операцій.

У сучасних веб-додатках підключення баз даних реалізується з урахуванням вимог масштабованості та надійності. Для цього використовуються середовища контейнеризації, хмарні сервіси та конфігурації, що дозволяють змінювати параметри підключення без модифікації програмного коду. Такий підхід є доцільним для веб-ресурсів медичних закладів, де необхідно забезпечити безперебійну роботу системи та захист конфіденційних даних користувачів.

Таким чином, вибір засобів і моделей підключення баз даних повинен здійснюватися з урахуванням особливостей предметної області, вимог до безпеки та очікуваного навантаження на систему. Оптимальне поєднання сучасних інтерфейсів доступу до даних і архітектурних моделей дозволяє створювати ефективні та надійні веб-орієнтовані інформаційні системи.

1.5 Висновок до першого розділу

У ході дослідження встановлено, що бази даних забезпечують ефективне, впорядковане та безпечне зберігання великих обсягів інформації, а також її оперативну обробку в багатокористувацькому режимі. Класифікація баз даних за моделями організації, способом зберігання та архітектурою дозволяє обрати оптимальне рішення залежно від специфіки задачі та вимог до інформаційної системи.

Особливу увагу приділено процесу реалізації баз даних, який охоплює проєктування структури даних, вибір системи управління базами даних, створення фізичної моделі та оптимізацію доступу до інформації. Розглянуто специфіку підключень до баз даних, що визначається архітектурою системи, методами доступу та вимогами до безпеки, а також роль адміністрування у забезпеченні стабільності, продуктивності та захисту даних.

Результати роботи підтверджують, що ефективне функціонування баз даних можливе лише за умови комплексного підходу, який поєднує грамотну реалізацію, безпечну організацію підключень і системне адміністрування.

Такий підхід забезпечує надійність інформаційних систем, зменшує ризики втрати даних та створює умови для подальшого розвитку й масштабування програмних рішень.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ ТА МЕТОДИ РЕАЛІЗАЦІЇ

2.1. Вибір та обґрунтування засобів підключень баз даних

У межах розроблення та дослідження інформаційної системи одним із ключових етапів є вибір програмних засобів і методів реалізації підключення до бази даних, оскільки саме цей рівень визначає коректність збереження та обробки інформації, швидкодію прикладних сценаріїв, стійкість до помилок, захищеність від несанкціонованого доступу та можливість подальшого масштабування. Практика проєктування програмних систем показує, що помилки у виборі технологічного стека та механізмів доступу до даних призводять до підвищення складності супроводження, появи критичних вразливостей, складності оптимізації запитів та зростання ризику втрати даних. Тому обґрунтування вибору технологій повинно враховувати як функціональні вимоги (операції CRUD, формування звітів, робота з пов’язаними сутностями), так і нефункціональні (безпека, продуктивність, надійність, керованість, переносимість, підтримка транзакцій, можливість відновлення після збоїв).

У даному дослідженні обрано стек Python–Django–MySQL як основу серверної частини із застосуванням HTML і CSS для представлення даних у вебінтерфейсі. Така комбінація є збалансованою з позиції розроблення повноцінних вебзастосунків, де необхідно забезпечити стабільний доступ до даних, реалізувати бізнес-логіку та надати користувачу зручний інтерфейс взаємодії. При цьому вибір не обмежується лише популярністю технологій, а враховує їхню архітектурну сумісність, зрілість інструментів, наявність усталених підходів до безпеки та оптимізації, а також практичну доцільність для задач магістерського рівня, де важливо продемонструвати не тільки роботу системи, а й аргументацію інженерних рішень.

Python як мова реалізації серверної логіки забезпечує швидке створення прототипів, підтримку об’єктно-орієнтованої моделі та широкі можливості інтеграції з різними бібліотеками. З точки зору доступу до даних Python підтримує стандартизовані інтерфейси підключення до СКБД, а також інструменти для логування, профілювання та тестування, що є важливими для дослідження якості реалізації. Окрім цього, екосистема Python дозволяє застосовувати підходи до написання чистого коду, статичного аналізу, юніт-тестування та інтеграційного тестування взаємодії з базою даних. У результаті підвищується керованість коду, зменшується кількість дефектів та спрощується контроль відповідності програмного забезпечення вимогам.

Django використано як фреймворк, який забезпечує структуровану організацію проєкту та містить комплексні механізми для взаємодії з базою даних. Важливою підставою для вибору Django є вбудований ORM, що реалізує об’єктно-реляційне відображення та дозволяє описувати структуру даних через моделі, які відображаються на таблиці бази даних. На практиці це означає, що розробник може оперувати сутностями предметної області (наприклад, користувач, замовлення, запис, категорія, товар тощо) як об’єктами, а фреймворк автоматично трансформує операції над ними у SQL-команди з урахуванням конфігурації підключення. Такий підхід підвищує продуктивність розроблення, уніфікує доступ до даних, полегшує зміну схеми бази даних та сприяє підтримці принципів цілісності.

Додатковою перевагою ORM є можливість поступового ускладнення запитів: від простих вибірок до складних агрегацій, фільтрації, сортування, обмеження вибірки, попереднього завантаження пов’язаних об’єктів і оптимізації кількості SQL-запитів. Для дослідницької частини це важливо, оскільки дозволяє порівнювати підходи: використання ORM-операцій, побудову складних запитів засобами QuerySet, застосування аннотацій і агрегатних функцій, а за потреби — виконання “сирих” SQL-запитів. Таким чином забезпечується баланс між зручністю високорівневого доступу до даних і

можливістю низькорівневої оптимізації у випадках, де це потрібно для продуктивності.

Суттєвим елементом реалізації є система міграцій Django, яка забезпечує керовану еволюцію схеми бази даних. У контексті магістерського дослідження це дає можливість аргументувати, що зміни структури даних виконуються відтворювано, контрольовано та прозоро: кожна зміна фіксується у вигляді міграційного файлу, може бути застосована або відкотена, а також використана для синхронізації середовищ (розробка, тестування, продуктивне середовище). Це мінімізує ризики “ручного” втручання у схему та зменшує ймовірність невідповідностей між версіями бази даних у різних середовищах.

Вибір MySQL як СКБД зумовлений її зрілістю, поширеністю та достатньою функціональністю для задач з реляційними даними. MySQL підтримує транзакції (у межах відповідних механізмів зберігання), індекси, зовнішні ключі, обмеження цілісності та механізми оптимізації виконання запитів. Реляційна модель доцільна у випадках, коли дані мають чітку структуру, а зв’язки між сутностями є важливими для коректної бізнес-логіки. Використання зовнішніх ключів дозволяє підтримувати цілісність зв’язків, а застосування індексів — пришвидшувати доступ до даних за найбільш типових сценаріїв пошуку, фільтрації та приєднання таблиць. У межах дослідження це дає можливість обґрунтувати рішення щодо проєктування таблиць, нормалізації, вибору типів даних і стратегій індексації.

Важливим аспектом підключення до MySQL є наявність відповідного драйвера та коректна конфігурація параметрів з’єднання. У Django підключення визначається через налаштування бази даних, де задаються тип СКБД, ім’я бази, користувач, пароль, хост та порт. З позиції інженерної практики критичним є забезпечення безпеки таких параметрів: їх рекомендовано зберігати у змінних середовища або у конфігураційних файлах, доступ до яких обмежений. Це дозволяє розмежувати конфігурації для різних середовищ та мінімізувати ризик витоку облікових даних у репозиторій. Додатково враховується керування з’єднаннями та пулінг: у вебсередовищі застосунок створює багато

короткочасних запитів, тому важливо налаштувати параметри стабільності з’єднань, таймаути та поведінку при повторному підключенні. Для дослідницької частини доцільним є аналіз типових помилок підключення, їхніх причин (мережеві обмеження, помилки автентифікації, несумісність драйвера, неправильні параметри кодування) та методів усунення.

Безпека доступу до бази даних у вибраному стеку забезпечується як на рівні Django, так і на рівні MySQL. З боку Django значну роль відіграє параметризоване формування запитів у ORM, що знижує ризик SQL-ін’єкцій, а також механізми аутентифікації, хешування паролів та керування сесіями. З боку MySQL важливим є принцип мінімально необхідних привілеїв: користувачу бази даних, з яким працює застосунок, надаються лише ті права, які потрібні для виконання функцій системи (наприклад, SELECT/INSERT/UPDATE/DELETE лише у межах конкретної схеми). У науковому дослідженні доцільно підкреслити, що безпека не обмежується захистом від ін’єкцій, а включає контроль доступу, резервне копіювання, аудит змін, захист каналу передачі (за необхідності — використання SSL/TLS), а також політики обробки помилок, щоб уникати витоку технічних деталей у користувацький інтерфейс.

Продуктивність системи при роботі з базою даних визначається не лише обраною СКБД, а й підходами до формування запитів і організації даних. ORM спрощує доступ, однак може призводити до надмірної кількості запитів, зокрема при роботі з пов’язаними моделями. Тому у межах дослідження обґрунтованим є застосування прийомів оптимізації: попереднє завантаження пов’язаних об’єктів (щоб зменшити кількість запитів), вибірка лише необхідних полів, агрегації на рівні бази даних, використання індексів, аналіз планів виконання запитів та профілювання. Важливо також враховувати, що надмірне ускладнення логіки на рівні застосунку може бути замінене ефективнішими операціями на рівні БД (наприклад, групуванням, підрахунками, фільтрацією). Таким чином, дослідження має демонструвати, що продуктивність досягається комплексно — за рахунок правильного проєктування моделі даних і ефективного формування запитів.

Представлення даних користувачу реалізується за допомогою HTML і CSS, які формують клієнтський рівень вебзастосунку. HTML дозволяє описати структуру сторінок, форми введення, таблиці та елементи відображення результатів. CSS забезпечує узгоджене візуальне оформлення, що є важливим для сприйняття інформації, зменшення помилок користувача та покращення взаємодії з системою. У поєднанні з шаблонізатором Django HTML-сторінки можуть динамічно наповнюватися даними, отриманими з бази даних, при цьому логіка обробки запитів, перевірки даних і керування доступом залишається на серверному боці. Такий підхід відповідає принципам розподілу відповідальності: клієнтський рівень відповідає за подання, серверний — за обробку та доступ до даних. Для магістерського дослідження це важливо як демонстрація коректної архітектури та підтримки керованості системи.

Методологічно реалізацію підключення до бази даних у запропонованому стеку доцільно розглядати як поєднання конфігураційного підходу (налаштування доступу до БД і параметрів з’єднання), об’єктного моделювання (опис сутностей через ORM), транзакційного підходу (гарантування цілісності при виконанні груп операцій) та застосування вбудованих механізмів контролю доступу. Також суттєвим є забезпечення тестованості реалізації: використання тестових баз, фікстур, перевірка поведінки при помилках підключення, валідації даних та коректності транзакцій. У сукупності це дозволяє аргументувати, що обрані програмні засоби забезпечують не лише реалізацію підключення як технічного факту, а й створюють основу для надійної експлуатації системи в умовах реальних навантажень і ризиків.

Отже, вибір Python, Django та MySQL у поєднанні з HTML і CSS є обґрунтованим з позиції реалізації повного циклу роботи з даними у веборієнтованій інформаційній системі. Запропонований технологічний стек забезпечує високий рівень абстракції та зручність розроблення завдяки ORM і міграціям, підтримує принципи безпеки через параметризовані запити та контроль доступу, дозволяє оптимізувати продуктивність за рахунок індексації та ефективного формування запитів, а також забезпечує зрозуміле представлення

результатів користувачу через стандартизовані засоби вебінтерфейсу. У контексті магістерського дослідження це дозволяє продемонструвати системний підхід до вибору технологій та методів реалізації підключення до баз даних, а також сформувати підґрунтя для подальших експериментів із продуктивністю, безпекою та масштабованістю розробленого програмного забезпечення.

2.2 Підключення до бази даних, кроки реалізації, схеми роботи та реалізація MySQL та django

Підключення Django-проєкту до бази даних MySQL складається з трьох логічних частин: розуміння, як працює зв’язок застосунку з БД, проєктування схеми роботи (архітектури) і безпосередня технічна реалізація в MySQL та Django.

Спочатку варто розібратися з базовою моделлю взаємодії. Django працює через свій ORM (Object-Relational Mapping): замість “чистих” SQL-запитів ви описуєте дані як Python-класи (моделі), а Django перетворює їх у таблиці та SQL-оператори. На нижньому рівні Django використовує драйвер бази даних (для MySQL це, як правило, mysqlclient або mysql-connector-python), який встановлює TCP-з’єднання з сервером MySQL, виконує запити, повертає результати та керує транзакціями. Важливо розуміти, що для кожного HTTP-запиту до Django застосовується приблизно така схема: веб-сервер (Gunicorn/uWSGI + Nginx або вбудований dev-сервер) приймає запит, Django виконує бізнес-логіку, ORM звертається до MySQL через драйвер, отримує/записує дані, формує відповідь і повертає її клієнту.

Щоб спроєктувати схему роботи, починають з вимог: які сутності є в системі (користувачі, товари, замовлення тощо), які між ними зв’язки (один-до-одного, один-до-багатьох, багато-до-багатьох), які типи операцій будуть найбільш частими (читання звітів, транзакції в реальному часі, масові оновлення).

На цій основі будують ER-діаграму: таблиці, ключі, індекси, зовнішні ключі. Для високого навантаження важливо одразу продумати індекси під типові фільтри та JOIN-и, а також структурувати БД так, щоб мінімізувати дублювання даних, але не жертвувати продуктивністю.

Схема роботи бекенду з БД зазвичай виглядає так: Django-моделі описують таблиці, міграції відповідають за створення та зміну структури БД, в'юхи/серіалізатори/сервіси реалізують бізнес-логіку, а на рівні інфраструктури налаштовується пул з'єднань з MySQL, резервне копіювання, реплікація (якщо треба) та моніторинг. Реалізація MySQL починається з встановлення сервера, створення окремої бази та користувача для проєкту з мінімально необхідними правами. Наприклад, у MySQL це може виглядати так (команди виконуються в консолі MySQL):

```
CREATE DATABASE myproject CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

CREATE USER 'myproject_user'@'%' IDENTIFIED BY 'strong_password';

GRANT ALL PRIVILEGES ON myproject.* TO 'myproject_user'@'%';

FLUSH PRIVILEGES;
```

Рис. 2.1. Реалізація SQL

utf8mb4 бажаний, щоб коректно зберігати емодзі та всі юнікод-символи. Далі в my.cnf варто налаштувати параметри, пов'язані з продуктивністю: розмір буферів (innodb_buffer_pool_size), тип рушія (InnoDB для транзакційності), таймаути з'єднань, логування повільних запитів.

Для продакшну часто налаштовують реплікацію (master-replica) та регулярні дампи за допомогою mysqldump або логічних/фізичних бекап-інструментів.

На боці Django конфігурація БД задається у файлі settings.py в змінній DATABASES.

Для MySQL типовий приклад виглядає так:Django згенерує та виконає SQL-інструкції для MySQL, створивши таблиці відповідно до моделей.

Нижче, на рис.2.2, проілюстровано конфігурацію підключення Django-застосунку до бази даних MySQL із використанням строгого режиму SQL для забезпечення цілісності та коректності даних. Показано словник DATABASES, у якому визначено параметри підключення для основної (дефолтної) бази даних проєкту. Як рушій бази даних вказано django.db.backends.mysql, що означає використання MySQL як СУБД. Назва бази даних задана як "myproject", а для доступу до неї використовується окремий користувач "myproject_user" із відповідним паролем.

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "myproject",
        "USER": "myproject_user",
        "PASSWORD": "strong_password",
        "HOST": "127.0.0.1",   # або IP/домен сервера MySQL
        "PORT": "3306",
        "OPTIONS": {
            "init_command": "SET sql_mode='STRICT_TRANS_TABLES'",
        },
    }
}
```

Рис. 2.2. Конфігурація DB

У схемі роботи важливо також передбачити, як застосунок буде поводитися при високому навантаженні та помилках підключення. У продакшні зазвичай використовують WSGI/ASGI-сервер (Gunicorn або Daphne/Uvicorn), балансувальник навантаження, пул з’єднань до MySQL (частково Django робить це самостійно, але можна додатково контролювати через параметри CONN_MAX_AGE тощо), кешування (Redis/Memcached) для зменшення

кількості запитів до БД та централізований логінг помилок. Django дозволяє описати повторні звернення до БД через ORM, але для критично важких місць можна додатково використовувати “сирий” SQL (raw() або connection.cursor()), якщо потрібно тонке ручне налаштування запитів.

Загалом реалізація MySQL + Django зводиться до того, щоб правильно поєднати три рівні: логічну модель даних (моделі та ER-діаграма), фізичну реалізацію (MySQL-схема з індексами, типами полів та оптимізацією) та додаток (Django-налаштування, ORM-логіка, оптимізація запитів і кешування). Якщо всі три рівні спроєктовані узгоджено, підключення до бази даних стає прозорим, а сама схема роботи системи — стійкою до росту кількості користувачів і даних.

Додатково слід зазначити, що важливим аспектом підключення до бази даних є забезпечення безпеки облікових даних та передаваних запитів. У практичних реалізаціях не рекомендується зберігати параметри доступу до бази даних безпосередньо в коді застосунку. Натомість використовуються змінні середовища або спеціалізовані сховища конфігурацій, що дозволяє мінімізувати ризики витоку логінів і паролів, а також спрощує розгортання застосунку в різних середовищах (локальному, тестовому, промисловому).

Окрему увагу варто приділити керуванню транзакціями. Django та MySQL підтримують механізм ACID-транзакцій, який гарантує атомарність, узгодженість, ізольованість і довговічність операцій з даними. Використання транзакцій особливо важливе для систем, що працюють з фінансовою або медичною інформацією, де будь-яка помилка під час запису може призвести до втрати або спотворення критично важливих даних. Django надає інструменти для явного керування транзакціями, що дозволяє розробнику контролювати складні бізнес-операції.

Також доцільно враховувати питання масштабування системи. У міру зростання кількості користувачів та обсягів даних виникає потреба у горизонтальному або вертикальному масштабуванні бази даних. У випадку MySQL це може реалізовуватися шляхом використання реплік для читання, розподілу навантаження між серверами або винесення окремих компонентів у

мікросервісну архітектуру. Django, у свою чергу, добре інтегрується з такими підходами завдяки гнучким налаштуванням підключень та можливості розділення читання і запису.

Ще одним важливим аспектом є оптимізація запитів. Навіть за наявності ORM необхідно аналізувати згенеровані SQL-запити, уникати проблеми N+1 та використовувати механізми попереднього завантаження пов’язаних об’єктів (select_related, prefetch_related). Це дозволяє суттєво зменшити кількість звернень до бази даних і підвищити загальну продуктивність застосунку.

Таким чином, підключення Django-застосунку до бази даних MySQL є не лише технічною задачею налаштування з’єднання, а комплексним процесом, що включає питання безпеки, транзакційності, масштабування та оптимізації. Урахування цих факторів на етапі проєктування дозволяє створити надійну й ефективну інформаційну систему.

2.3 Висновок до другого розділу

У результаті проведеного дослідження було проаналізовано підходи до підключення програмних застосунків до баз даних, а також обґрунтовано вибір технологій і методів реалізації взаємодії між серверною частиною та системою керування базами даних. Встановлено, що ефективне підключення до бази даних є не лише технічною операцією встановлення з’єднання, а комплексним процесом, який включає проєктування структури даних, налаштування параметрів доступу, організацію безпечної взаємодії, реалізацію бізнес-логіки та забезпечення стабільності роботи системи в умовах реального навантаження.

У межах роботи було визначено основні кроки реалізації підключення до бази даних, які охоплюють вибір відповідної СКБД, конфігурацію параметрів з’єднання, опис моделей предметної області, налаштування механізмів міграцій та реалізацію операцій доступу до даних. Особливу увагу приділено логіці побудови схем роботи системи, де чітко розмежовуються рівні представлення, прикладної логіки та збереження даних. Такий підхід забезпечує

структурованість програмного коду, спрощує його супроводження та створює умови для подальшого масштабування інформаційної системи.

Реалізація підключення до бази даних на основі MySQL та Django продемонструвала доцільність використання об’єктно-реляційного відображення як основного механізму взаємодії з даними. Застосування ORM дозволяє абстрагуватися від низькорівневих SQL-запитів, забезпечує цілісність і узгодженість даних, а також підвищує рівень безпеки за рахунок параметризованого формування запитів. Використання MySQL як реляційної системи керування базами даних забезпечує стабільне зберігання інформації, підтримку транзакцій і механізмів контролю цілісності, що є критично важливим для побудови надійних інформаційних систем.

Проаналізовані схеми роботи застосунку з базою даних підтверджують, що поєднання Django та MySQL дозволяє ефективно реалізувати повний цикл обробки даних: від прийняття запиту користувача та виконання серверної логіки до збереження результатів у базі даних і відображення їх у вебінтерфейсі. Чітка організація взаємодії між компонентами системи, використання міграцій для керування схемою бази даних і централізоване налаштування підключень забезпечують керованість і відтворюваність процесів розроблення.

Отже, результати дослідження підтверджують, що обрані програмні засоби та методи реалізації підключення до бази даних відповідають сучасним вимогам до розроблення веборієнтованих інформаційних систем. Реалізація підключення до бази даних за допомогою Django та MySQL є обґрунтованим і ефективним рішенням, яке поєднує зручність розроблення, високий рівень безпеки, продуктивність і можливість подальшого розширення функціональності системи, що робить його доцільним для використання в межах магістерського дослідження.Ці інструменти в сукупності надають великі можливості для розробки, забезпечуючи високу продуктивність, масштабованість та безпеку проєктів.

РОЗДІЛ 3
СИСТЕМНИЙ АНАЛІЗ ОБ'ЄКТА ДОСЛІДЖЕННЯ

Сучасний етап розвитку інформаційних технологій характеризується активним упровадженням веборієнтованих інформаційних систем у сферу охорони здоров'я, що зумовлено зростанням обсягів медичних даних, підвищенням вимог до якості медичних послуг та необхідністю оптимізації внутрішніх процесів медичних закладів. Приватні клініки, як динамічні суб'єкти медичної діяльності, дедалі частіше використовують вебсайти з інтегрованими базами даних не лише як інформаційні ресурси, а як повноцінні інструменти управління пацієнтами, медичними записами та адміністративними процесами. У таких умовах особливої актуальності набуває системний підхід до аналізу об'єкта дослідження, який дозволяє сформувати обґрунтовану модель функціонування інформаційної системи ще на етапі проєктування.

Необхідність проведення системного аналізу зумовлена специфікою предметної області, адже медичні інформаційні системи працюють з персональними та чутливими даними, для яких критично важливими є питання достовірності, захисту та контрольованого доступу. Помилки на етапі аналізу можуть призвести до некоректного проєктування структури бази даних, порушення логіки бізнес-процесів, зниження продуктивності системи та підвищення ризиків інформаційної безпеки. Саме тому у даному розділі системний аналіз розглядається як методологічна основа для подальшого проєктування логічної та фізичної моделей бази даних вебзастосунку.

У межах розділу виконано декомпозицію цілей функціонування системи шляхом побудови дерева цілей, що дозволяє узгодити стратегічні завдання клініки з конкретними функціональними можливостями вебсайту та бази даних. Також здійснено концептуальне моделювання роботи системи, яке відображає основні ролі користувачів, інформаційні потоки та взаємодію між клієнтською частиною, серверною логікою та базою даних. Окрему увагу приділено побудові

ієрархії процесів, що дозволяє формалізувати логіку обробки даних і визначити послідовність виконання ключових операцій у межах системи.

Таким чином, даний розділ створює теоретичне та аналітичне підґрунтя для наступних етапів магістерського дослідження, зокрема для проєктування структури бази даних, вибору методів підключення до СКБД та реалізації вебзастосунку приватної клініки з використанням сучасних програмних засобів.

3.1 Дерево цілей

Дерево цілей є одним із базових інструментів системного аналізу, який застосовується для формалізації цілей створення інформаційної системи та їх ієрархічного впорядкування. Використання даного методу дозволяє перейти від абстрактної загальної мети функціонування системи до конкретних, вимірюваних і технічно реалізованих завдань, що безпосередньо впливають на структуру бази даних і логіку роботи вебзастосунку.

У межах розробки вебсайту приватної клініки дерево цілей використовується як аналітичний інструмент для узгодження потреб медичного закладу, очікувань користувачів та технічних можливостей інформаційної системи. Основна мета системи формулюється як створення інтегрованої веборієнтованої інформаційної системи з базою даних, яка забезпечує ефективне керування медичною та адміністративною інформацією, підвищує якість обслуговування пацієнтів і оптимізує внутрішні процеси клініки.

На першому рівні декомпозиції основна мета деталізується на стратегічні підцілі, що відображають ключові напрями функціонування системи. До таких підцілей належать автоматизація обліку пацієнтів, підтримка процесу запису на прийом, забезпечення роботи лікарів з медичними даними, адміністрування інформаційних ресурсів та гарантування безпеки й конфіденційності даних. Кожна з цих підцілей має безпосередній вплив на проєктування структури бази даних, зокрема на визначення основних сутностей, їхніх атрибутів і зв’язків між ними.

На другому рівні дерева цілей відбувається подальша деталізація стратегічних підцілей у вигляді функціональних цілей. Наприклад, автоматизація обліку пацієнтів включає такі функціональні цілі, як реєстрація користувачів, збереження персональних даних, формування електронних медичних карток та підтримка історії звернень. Підціль підтримки запису на прийом деталізується через перевірку доступності лікаря, керування розкладом, створення та редагування записів, а також інформування користувачів про зміни. Цілі, пов’язані з безпекою, включають автентифікацію, авторизацію, розмежування прав доступу та контроль операцій з медичними даними.

На нижньому рівні дерева цілей формуються операційні цілі, які безпосередньо реалізуються у вигляді програмних модулів, таблиць бази даних, обмежень цілісності та механізмів доступу. Саме цей рівень визначає конкретні технічні рішення, зокрема структуру таблиць, використання зовнішніх ключів, індексів, механізмів журналювання та логіки доступу до даних. Таким чином, дерево цілей виступає сполучною ланкою між концептуальним баченням системи та її практичною реалізацією.

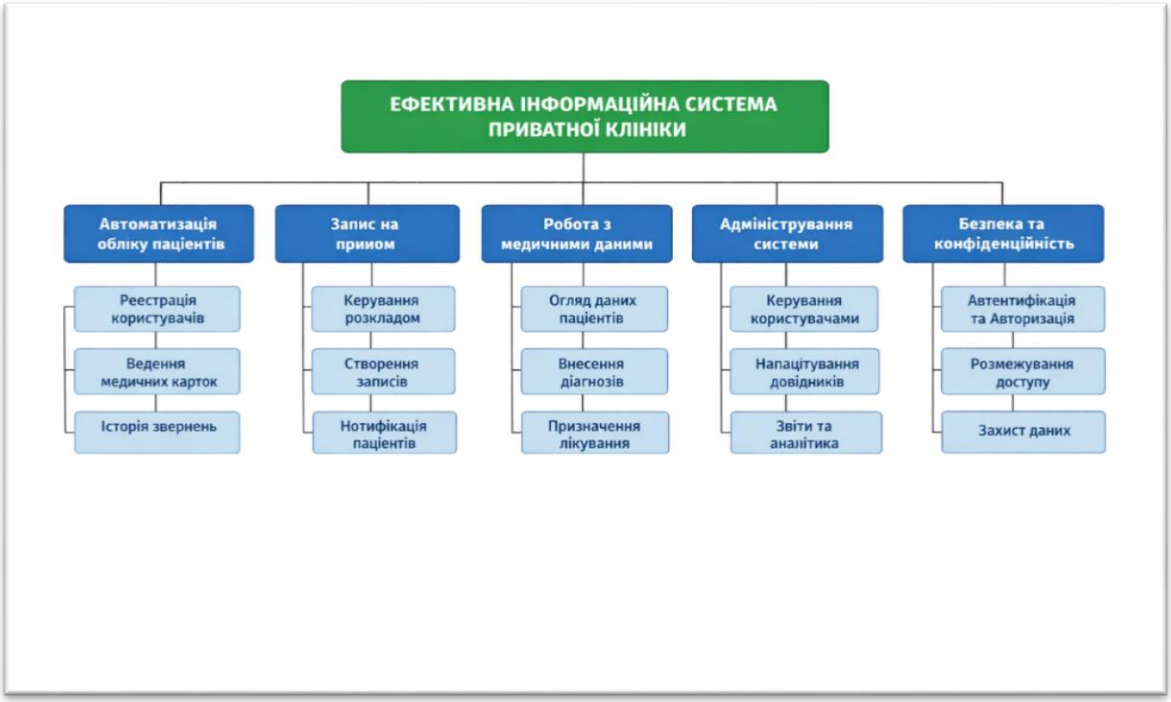

Рис. 3.1. Дерево цілей

Застосування дерева цілей у процесі системного аналізу дозволяє забезпечити логічну цілісність проєктування, уникнути дублювання функцій, чітко визначити межі відповідальності кожного компонента системи та створити основу для подальшої побудови ієрархії процесів. У контексті медичної інформаційної системи це особливо важливо, оскільки помилки на рівні постановки цілей можуть призвести до порушення логіки обробки даних або створення вразливостей у системі безпеки.

3.2 Концептуальне моделювання роботи системи

Концептуальне моделювання роботи системи є одним із ключових етапів системного аналізу, оскільки дозволяє сформувати узагальнене уявлення про функціонування інформаційної системи, визначити основні компоненти, ролі користувачів та характер інформаційних потоків між ними. На цьому рівні система розглядається незалежно від конкретних програмних засобів реалізації, що дає змогу зосередитися на логіці взаємодії елементів та структурі даних, які обробляються в межах предметної області.

У межах даного дослідження вебсайт приватної клініки розглядається як багаторівнева інформаційна система, що об’єднує клієнтський рівень, серверну логіку та базу даних в єдине інформаційне середовище. Основним призначенням системи є забезпечення централізованого зберігання та обробки медичної й адміністративної інформації, а також підтримка взаємодії між пацієнтами, медичним персоналом і адміністрацією клініки.

Концептуальна модель системи передбачає наявність кількох основних ролей користувачів, кожна з яких має власні функціональні повноваження та інформаційні потреби. До таких ролей належать пацієнт, лікар та адміністратор. Пацієнт взаємодіє з системою з метою отримання загальної інформації про клініку, перелік медичних послуг, спеціалістів та графік їх роботи, а також для здійснення електронного запису на прийом. У межах концептуальної моделі

пацієнт не має прямого доступу до медичних даних інших осіб і може переглядати лише інформацію, що стосується власних записів і прийомів.

Лікар є користувачем системи з розширеними правами доступу, який використовує вебзастосунок для роботи з медичними даними. У концептуальній моделі лікар отримує доступ до розкладу прийомів, інформації про пацієнтів, електронних медичних карток та результатів попередніх консультацій. Також лікар здійснює внесення нових медичних записів, фіксує діагнози, призначення та рекомендації. Взаємодія лікаря з базою даних відбувається виключно через серверну частину системи, що забезпечує контроль цілісності та коректності збережених даних.

Адміністратор системи відповідає за підтримку та коректне функціонування інформаційної системи в цілому. У концептуальній моделі адміністратор здійснює керування обліковими записами користувачів, налаштування довідкової інформації, контроль розкладу роботи лікарів, а також адміністрування системних параметрів. Доступ адміністратора до бази даних також реалізується опосередковано через серверну логіку, що дозволяє реалізувати розмежування доступу та забезпечити безпеку даних.

Центральним елементом концептуальної моделі є база даних, яка виконує функцію ядра системи та забезпечує збереження всієї інформації, необхідної для роботи клініки. У базі даних зберігаються відомості про пацієнтів, лікарів, медичні послуги, записи на прийом, результати оглядів та службова інформація. Концептуальна модель передбачає чітке розмежування типів даних і зв’язків між ними, що є передумовою для подальшого логічного проєктування структури бази даних.

Взаємодія між користувачами та базою даних здійснюється через серверну частину вебзастосунку, яка реалізує бізнес-логіку системи. Серверна логіка відповідає за обробку запитів користувачів, перевірку прав доступу, валідацію введених даних, виконання операцій збереження та оновлення інформації, а також формування відповідей для клієнтського рівня. Такий підхід

дозволяє уникнути прямого доступу клієнтської частини до бази даних, що є критично важливим для забезпечення конфіденційності медичної інформації.

Концептуальна модель також відображає основні інформаційні потоки в системі, зокрема потоки введення даних користувачами, обробки запитів серверною частиною та збереження результатів у базі даних. Усі операції з даними розглядаються як контрольовані процеси, що виконуються відповідно до встановлених правил доступу та логіки предметної області. Це забезпечує цілісність даних, відстежуваність змін і можливість подальшого аналізу інформації.

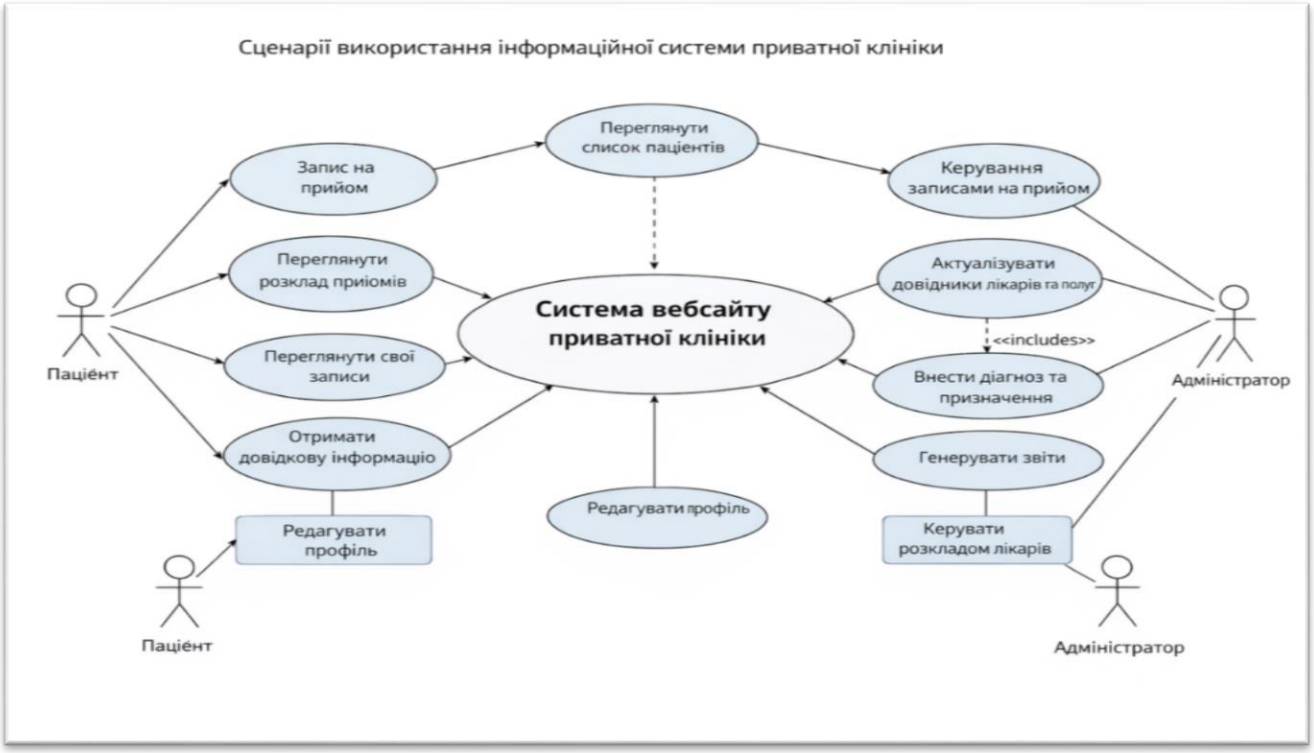

Рис. 3.2. UML-діаграма ролей

Таким чином, концептуальне моделювання роботи системи дозволяє сформувати узгоджене уявлення про функціонування вебсайту приватної клініки як цілісної інформаційної системи, визначити ключові ролі користувачів, структуру взаємодії компонентів та основні інформаційні потоки. Отримана концептуальна модель слугує основою для подальшої побудови ієрархії

процесів, логічного та фізичного проєктування бази даних, а також реалізації серверної частини вебзастосунку.

3.3 Побудова ієрархії процесів

Розробка ієрархії процесів є важливим етапом системного аналізу, який дозволяє формалізувати функціонування інформаційної системи шляхом послідовної декомпозиції її діяльності на взаємопов’язані процеси різного рівня деталізації. На відміну від дерева цілей, яке відображає логіку досягнення стратегічних та функціональних цілей, ієрархія процесів описує практичну реалізацію цих цілей у вигляді конкретних дій та операцій, що виконуються системою та її користувачами.

У межах даного дослідження вебсайт приватної клініки розглядається як складна інформаційна система, у якій одночасно реалізуються медичні, адміністративні та сервісні процеси. Побудова ієрархії процесів дозволяє визначити межі відповідальності кожного компонента системи, встановити послідовність обробки даних і створити основу для подальшого проєктування серверної логіки та структури бази даних.

На верхньому рівні ієрархії виділяється узагальнений процес функціонування вебсайту приватної клініки, який охоплює всі дії, пов’язані з взаємодією користувачів із системою, обробкою запитів та доступом до бази даних. Цей процес включає отримання запитів від користувачів через вебінтерфейс, їх обробку серверною частиною, виконання операцій збереження або вибірки даних та формування відповідей для клієнтського рівня. Такий інтегрований процес забезпечує цілісність роботи системи та узгодженість взаємодії між її складовими.

На другому рівні ієрархії здійснюється декомпозиція загального процесу на основні групи бізнес-процесів відповідно до ролей користувачів і функціонального призначення системи. До таких груп процесів належать процеси керування користувачами, процеси організації запису на прийом,

процеси ведення медичних даних, процеси адміністрування системи та процеси інформаційного обслуговування користувачів. Кожна з цих груп реалізує окремий напрям функціонування системи та безпосередньо пов’язана з відповідними підцілями дерева цілей.

Процеси керування користувачами охоплюють реєстрацію, автентифікацію та авторизацію користувачів, а також керування ролями доступу. У межах цих процесів здійснюється перевірка облікових даних, визначення рівня доступу до функцій системи та забезпечення безпеки персональної й медичної інформації. Дані процеси мають критичне значення для функціонування всієї системи, оскільки вони визначають, які операції можуть виконувати різні категорії користувачів.

Процеси організації запису на прийом включають керування розкладом лікарів, перевірку доступності спеціалістів, створення, редагування та скасування записів, а також інформування пацієнтів про зміни у графіку. Ці процеси забезпечують автоматизацію одного з ключових сервісів приватної клініки та тісно пов’язані зі структурою бази даних, у якій зберігається інформація про лікарів, часові інтервали та записи пацієнтів.

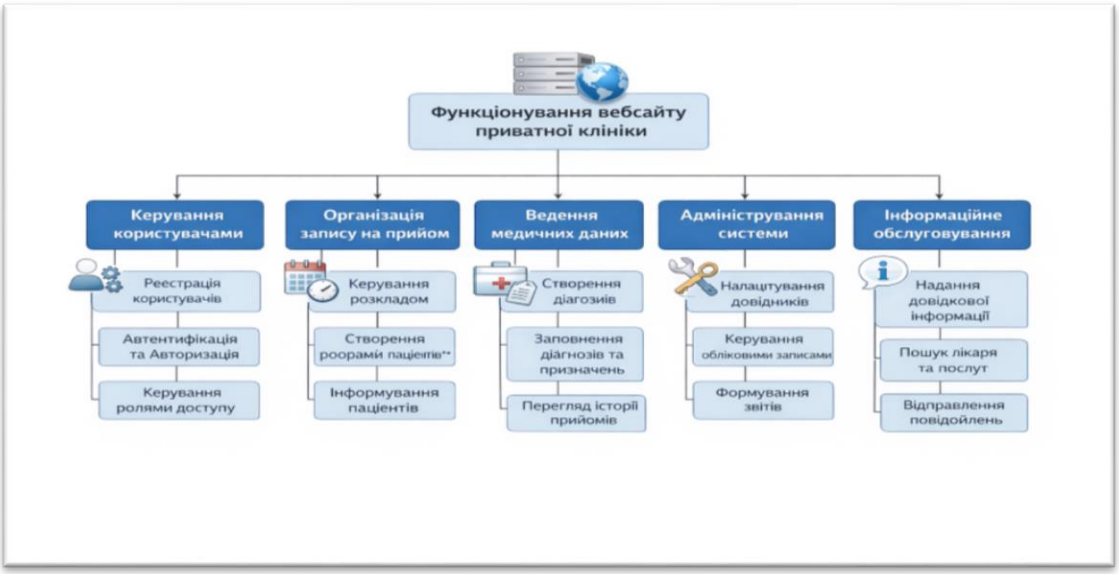

Рис. 3.3. Ієрархія процесів клінічного управління

Процеси ведення медичних даних є центральними з точки зору предметної області та охоплюють створення та ведення електронних медичних карток, збереження історії звернень, внесення діагнозів, призначень і рекомендацій. У межах цих процесів реалізується взаємодія лікаря з базою даних, при цьому особлива увага приділяється контролю цілісності даних, історичності записів та обмеженню доступу до чутливої інформації.

Процеси адміністрування системи включають керування довідковими даними, обліковими записами користувачів, налаштування параметрів системи та формування звітів. Дані процеси забезпечують стабільне функціонування системи, її актуальність та відповідність вимогам медичного закладу. У межах ієрархії процесів адміністрування розглядається як окремий функціональний блок із розширеними правами доступу.

Процеси інформаційного обслуговування користувачів спрямовані на надання пацієнтам і персоналу актуальної довідкової інформації, зокрема про перелік послуг, спеціалістів, правила користування системою та контактні дані клініки. Хоча ці процеси не пов’язані безпосередньо з медичними записами, вони відіграють важливу роль у забезпеченні зручності та зрозумілості роботи з вебсайтом.

На нижчих рівнях ієрархії кожен з основних процесів деталізується на підпроцеси та окремі операції, які реалізуються у вигляді програмних модулів, методів серверної логіки та відповідних операцій доступу до бази даних. Така деталізація дозволяє встановити відповідність між бізнес-процесами та елементами програмної реалізації, що є необхідним для подальшого проєктування архітектури вебзастосунку.

3.4 Висновок до третього розділу

У результаті виконання системного аналізу об’єкта дослідження вебсайт приватної клініки було розглянуто як цілісну багаторівневу інформаційну систему, що забезпечує автоматизацію медичних і адміністративних процесів та

централізоване керування даними. Проведений аналіз дозволив сформувати структуроване уявлення про цілі створення системи, її функціональне призначення та логіку взаємодії між основними компонентами й користувачами.

У ході дослідження було визначено основну мету функціонування інформаційної системи та виконано її декомпозицію на взаємопов’язані функціональні завдання, що відображають потреби приватної клініки. Це дало змогу узгодити вимоги предметної області з технічними рішеннями та забезпечити логічну послідовність подальшого проєктування структури бази даних і програмної реалізації вебзастосунку.

Також сформовано узагальнену модель роботи системи, яка відображає взаємодію між користувачами різних ролей, серверною логікою та базою даних як центральним елементом збереження й обробки інформації. Такий підхід дозволяє забезпечити контроль доступу до медичних даних, їхню цілісність і конфіденційність, що є критично важливим для інформаційних систем медичного призначення.

Крім того, функціонування вебсайту приватної клініки було формалізовано у вигляді впорядкованої сукупності процесів, що відображають основні напрями діяльності системи. Побудована ієрархія процесів дозволяє чітко розмежувати відповідальність між компонентами системи, забезпечити узгодженість між бізнес-логікою та структурою бази даних, а також створити основу для подальшого проєктування серверної частини та реалізації інформаційної системи.

Отже, результати системного аналізу створюють теоретичне й методологічне підґрунтя для наступних етапів магістерського дослідження, зокрема для логічного та фізичного проєктування бази даних, вибору засобів підключення та реалізації веборієнтованої інформаційної системи приватної клініки з використанням сучасних програмних технологій.

РОЗДІЛ 4
ПРАКТИЧНА РЕАЛІЗАЦІЯ

4.1 Опис створеного програмного засобу.

У цьому розділі здійснено науково обґрунтований аналіз структури та функціональних характеристик веб-орієнтованої інформаційної системи, розробленої з метою забезпечення інформаційної та сервісної підтримки діяльності приватного медичного закладу. Основним призначенням даного програмного засобу є організація безперервного доступу користувачів до актуальних відомостей про роботу клініки, склад медичного персоналу, перелік та вартість медичних послуг, а також контактної й довідкової інформації.

Архітектура веб-ресурсу сформована відповідно до принципів логічної структуризації та зручної навігації і включає сукупність функціональних розділів, зокрема: головну сторінку, інформаційний модуль лікарського персоналу, розділ медичних послуг і цін, інформаційний блок про діяльність закладу та контактний модуль. Головна сторінка виконує роль узагальнювального інформаційного елемента, що містить стислий опис діяльності клініки, її ключові переваги та засоби переходу до основних підсистем. Модуль, присвячений медичним спеціалістам, забезпечує відображення відомостей щодо їх професійної кваліфікації, спеціалізації та засобів комунікації, що створює умови для обґрунтованого вибору лікаря пацієнтом. Розділ медичних послуг реалізує систематизоване представлення процедур із зазначенням вартості, що сприяє підвищенню прозорості інформаційної взаємодії між закладом і користувачами. Інформаційний блок «Про заклад» містить відомості щодо організаційної структури, місії та принципів функціонування клініки, тоді як контактний модуль забезпечує доступ до адресних даних, режиму роботи, каналів зв’язку та інтерактивних засобів навігації.

Суттєвим функціональним компонентом розробленої системи є реалізація механізму дистанційного запису пацієнтів на прийом до лікаря шляхом використання електронної форми. Запровадження даного механізму дозволяє автоматизувати процес подання заявок, надати користувачам можливість самостійного вибору спеціаліста та часу візиту, а також зменшити навантаження на реєстратуру медичного закладу. Для адміністративного персоналу система забезпечує централізований облік і керування записами, що сприяє оптимізації процесів планування та зниженню ймовірності виникнення організаційних помилок.

4.2. Структура бази даних

Під час проєктування бази даних для інформаційної системи приватної клініки було виділено набір прикладних сутностей, що відображають основні бізнес-процеси: ведення довідника медичного персоналу, опис послуг, формування графіків прийому та реєстрацію звернень пацієнтів.

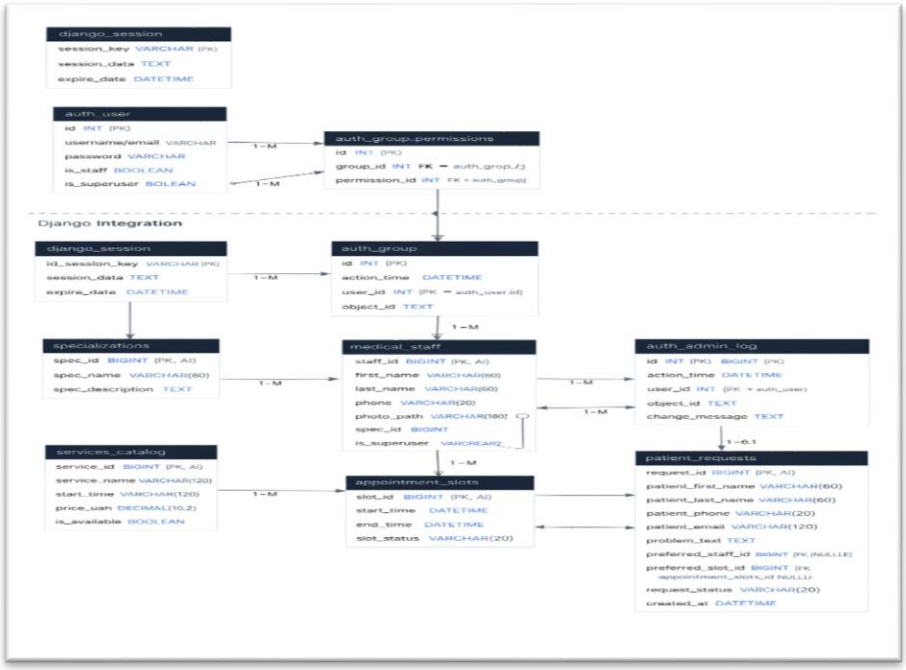

Рис 4.1. Схема бази даних

1) Таблиця clinic_profile — профіль клініки

Призначення: централізоване зберігання довідкових даних для сторінок «Про нас» та «Контакти».

- clinic_id BIGINT, PK, AI
- clinic_name VARCHAR(60), NOT NULL
- address_text VARCHAR(160), NOT NULL
- workdays_hours VARCHAR(60)
- saturday_hours VARCHAR(60)
- sunday_hours VARCHAR(60)
- phone_primary VARCHAR(20), NOT NULL
- phone_secondary VARCHAR(20)
- email_primary VARCHAR(120), NOT NULL
- email_secondary VARCHAR(120)
- social_facebook_url VARCHAR(160)
- social_instagram_url VARCHAR(160)
- social_linkedin_url VARCHAR(160)
- map_embed TEXT
- director_fullname VARCHAR(80)

2) Таблиця specializations — довідник спеціалізацій

Призначення: нормалізація даних (одна спеціалізація — багато лікарів).

- spec_id BIGINT, PK, AI
- spec_name VARCHAR(80), NOT NULL, UNIQUE
- spec_description TEXT

3) Таблиця medical_staff — медичний персонал (лікарі)

Призначення: зберігання інформації для відображення на сторінці «Лікарі» та для запису.

- staff_id BIGINT, PK, AI
- first_name VARCHAR(60), NOT NULL
- last_name VARCHAR(60), NOT NULL
- phone VARCHAR(20)

- photo_path VARCHAR(180)
- spec_id BIGINT, FK → specializations.spec_id, NOT NULL
- is_active TINYINT(1), NOT NULL, DEFAULT 1

Зв’язок: specializations (1) → (M) medical_staff

4) Таблиця services_catalog — каталог послуг

Призначення: формування сторінки «Послуги та ціни», підтримка актуальності цін.

- service_id BIGINT, PK, AI
- service_name VARCHAR(120), NOT NULL
- service_description TEXT
- price_uah DECIMAL(10,2), NOT NULL
- is_available TINYINT(1), NOT NULL, DEFAULT 1

5) Таблиця appointment_slots — розклад (вікна прийому)

Призначення: забезпечення коректного онлайн-запису не “в повітря”, а у конкретні часові інтервали.

- slot_id BIGINT, PK, AI
- staff_id BIGINT, FK → medical_staff.staff_id, NOT NULL
- start_time DATETIME, NOT NULL
- end_time DATETIME, NOT NULL
- slot_status VARCHAR(20), NOT NULL

Зв’язок: medical_staff (1) → (M) appointment_slots

6) Таблиця patient_requests — заявки/запис пацієнтів

Призначення: фіксація звернень із форми запису, контроль статусів обробки.

- request_id BIGINT, PK, AI
- patient_first_name VARCHAR(60), NOT NULL
- patient_last_name VARCHAR(60), NOT NULL
- patient_phone VARCHAR(20), NOT NULL
- patient_email VARCHAR(120), NOT NULL
- problem_text TEXT, NOT NULL

- preferred_staff_id BIGINT, FK → medical_staff.staff_id, NULL
- preferred_slot_id BIGINT, FK → appointment_slots.slot_id, NULL
- request_status VARCHAR(20), NOT NULL
- created_at DATETIME, NOT NULL

4.3 Опис механізмів роботи

Розроблена інформаційна система реалізує функціональність централізованого управління процесом запису пацієнтів на прийом до лікарів у приватному медичному закладі та функціонує у вигляді серверного веб-застосунку. Архітектурна модель системи орієнтована на обробку клієнтських запитів у єдиному обчислювальному середовищі з подальшим збереженням та обробкою даних у реляційній базі даних, а також наданням доступу до функціональних можливостей через веб-інтерфейс і прикладний програмний інтерфейс. Такий підхід забезпечує гнучкість реалізації, масштабованість та можливість подальшого розширення системи.

Функціональне ядро системи реалізовано мовою програмування Python із використанням веб-фреймворку Django, який забезпечує реалізацію серверної логіки, керування запитами та взаємодію з базою даних. Для організації обміну даними між клієнтською та серверною частинами застосовано архітектурні підходи Django Rest Framework, що дозволяє формувати уніфіковані REST-орієнтовані програмні інтерфейси. Представлення користувацького інтерфейсу здійснюється з використанням стандартних веб-технологій розмітки, що гарантує коректне відображення інформації у сучасних веб-браузерах.

Вхідною інформацією для серверного модуля є HTTP-запити, сформовані клієнтською частиною та передані у структурованому форматі JSON. У процесі обробки запитів система звертається до реляційної системи керування базами даних MySQL для отримання, перевірки та збереження необхідних даних. Результати виконання запитів поділяються на інформацію, що зберігається у базі даних для подальшого використання, та дані, які повертаються клієнту у вигляді

HTTP-відповідей у форматі JSON, що відповідає сучасним вимогам до побудови веб-сервісів і міжсистемної інтеграції.

Доступ до функціональних можливостей системи здійснюється шляхом звернення до відповідних серверних контролерів після ініціалізації веб-сервера. Взаємодія між окремими компонентами системи, а також з зовнішніми інформаційними ресурсами, реалізується через мережу Інтернет із використанням загальноприйнятих протоколів передачі даних

Клієнтська частина інформаційної системи реалізована з використанням фреймворку Vue.js, який забезпечує побудову інтерактивного користувацького інтерфейсу та динамічне оновлення даних без необхідності повного перезавантаження веб-сторінок. Застосування сучасних фреймворків і технологій, зокрема Django, Django Rest Framework та Vue.js, дозволяє досягти високого рівня продуктивності, зручності експлуатації та масштабованості розробленого програмного рішення, забезпечуючи ефективну організацію процесу запису пацієнтів у приватній клініці.

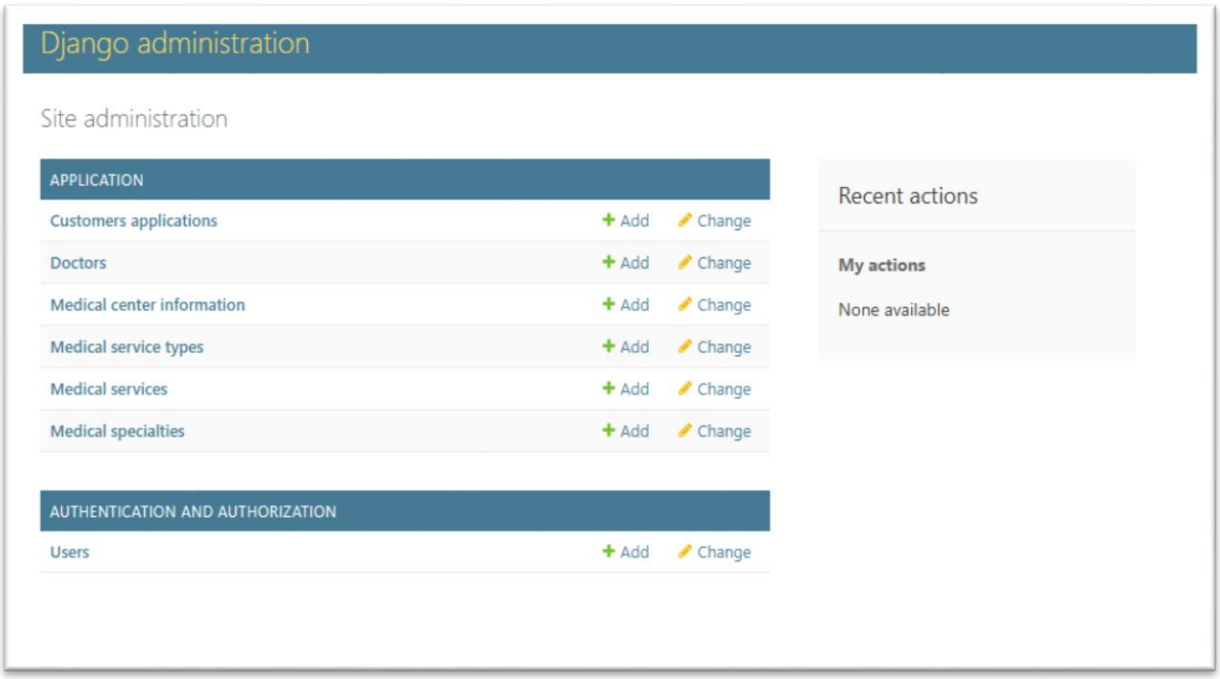

Рис. 4.2. Панель адміністрування Django

Лістинг 4.1

```
# admin.py
from django.contrib import admin
from django.contrib.auth import get_user_model
from django.contrib.auth.admin import UserAdmin as
DjangoUserAdmin
User = get_user_model()
@admin.register(User)
class UserAdmin(DjangoUserAdmin):
    list_display = ("username", "email", "is_active",
"is_staff", "is_superuser", "last_login")
    list_filter = ("is_active", "is_staff", "is_superuser")
    search_fields = ("username", "email", "first_name",
"last_name")
    ordering = ("-is_superuser", "-is_staff", "username")
    readonly_fields = ("last_login", "date_joined")
    fieldsets = (
        ("Основні дані", {"fields": ("username",
"password")}),
        ("Персональні дані", {"fields": ("first_name",
"last_name", "email")}),
        ("Права доступу", {"fields": ("is_active", "is_staff",
"is_superuser", "groups", "user_permissions")}),
   ("Системні дати", {"fields": ("date_joined", "last_login")}),
    )
```

Для керування обліковими записами користувачів у розробленій інформаційній системі використано вбудований адміністративний механізм веб-фреймворку Django, який реалізується за допомогою модуля django.contrib.admin. З метою адаптації стандартної адміністративної панелі до потреб проєкту було здійснено розширене налаштування класу адміністрування користувачів на основі базового класу UserAdmin.

У якості моделі користувача застосовується узагальнений механізм отримання користувацької моделі через функцію get_user_model(), що

забезпечує гнучкість системи та можливість використання як стандартної, так і розширеної моделі користувача без зміни логіки адміністрування.

У списковому представленні користувачів адміністративної панелі відображаються ключові атрибути облікових записів, зокрема ім’я користувача, електронна адреса, статус активності, належність до персоналу або адміністративної ролі, а також дата останнього входу в систему. Такий підхід дозволяє швидко оцінювати стан облікових записів і здійснювати оперативний контроль доступу до системи.

Лістинг 4.2

```
from django.contrib import admin
from .models import (
    ClinicProfile,
    Specialization,
    MedicalStaff,
    Service,
    AppointmentSlot,
    PatientRequest,
)
@admin.register(ClinicProfile)
class ClinicProfileAdmin(admin.ModelAdmin):
    list_display = ("clinic_name", "phone_primary",
"email_primary", "address_text")
    search_fields = ("clinic_name", "phone_primary",
"email_primary", "address_text")
    ordering = ("clinic_name",
@admin.register(Specialization)
class SpecializationAdmin(admin.ModelAdmin):
    list_display = ("spec_name",)
    search_fields = ("spec_name",)
    ordering = ("spec_name",)
class AppointmentSlotInline(admin.TabularInline):
переходу в окремий розділ.
    """
```

```
        model = AppointmentSlot
        extra = 0
        fields = ("start_time", "end_time", "slot_status")
        ordering = ("start_time",)

    @admin.register(MedicalStaff)
    class MedicalStaffAdmin(admin.ModelAdmin):
        list_display = ("last_name", "first_name",
"specialization", "phone", "is_active")
        list_filter = ("is_active", "specialization")
        search_fields = ("first_name", "last_name", "phone",
"specialization__spec_name")
        ordering = ("last_name", "first_name")

        autocomplete_fields = ("specialization",)
        inlines = (AppointmentSlotInline,)

        def get_queryset(self, request):
            qs = super().get_queryset(request)
            return qs.select_related("specialization")
    @admin.register(Service)
    class ServiceAdmin(admin.ModelAdmin):
        list_display = ("service_name", "price_uah",
"is_available")
        list_filter = ("is_available",)
        search_fields = ("service_name",)
        ordering = ("service_name",)
    @admin.register(AppointmentSlot)
    class AppointmentSlotAdmin(admin.ModelAdmin):
        list_display = ("doctor", "start_time", "end_time",
"slot_status")
        list_filter = ("slot_status", "doctor")
        search_fields = ("doctor__first_name",
"doctor__last_name")
        ordering = ("-start_time",)
        autocomplete_fields = ("doctor",)
```

```
        def get_queryset(self, request):
            qs = super().get_queryset(request)
            return qs.select_related("doctor")
    @admin.register(PatientRequest)
    class PatientRequestAdmin(admin.ModelAdmin):
        list_display = ("created_at", "patient_last_name",
"patient_first_name", "patient_phone",
                        "preferred_doctor", "request_status")
        list_filter = ("request_status", "preferred_doctor")
        search_fields =Ф ("patient_first_name",
"patient_last_name", "patient_phone", "patient_email")
        ordering = ("-created_at",)
        autocomplete_fields = ("preferred_doctor",
"preferred_slot")
        readonly_fields = ("created_at",)

        fieldsets = (
            ("Дані пацієнта", {"fields": ("patient_first_name",
"patient_last_name", "patient_phone", "patient_email")}),
            ("Звернення", {"fields": ("problem_text",)}),
            ("Параметри запису", {"fields": ("preferred_doctor",
"preferred_slot", "request_status")}),
            ("Системна інформація", {"fields": ("created_at",)}),
        )
        def get_queryset(self, request):
            qs = super().get_queryset(request)
            return qs.select_related("preferred_doctor",
"preferred_slot")
```

Для забезпечення зручного керування прикладними даними інформаційної системи приватної клініки використано стандартний адміністративний інтерфейс Django, який було адаптовано під структуру розробленої бази даних. У файлі admin.py реалізовано реєстрацію основних прикладних моделей системи, зокрема профілю клініки, довідника спеціалізацій, медичного персоналу, каталогу послуг, розкладу прийомів та заявок пацієнтів.

Для кожної моделі визначено перелік відображуваних полів у списковому поданні, налаштовано механізми пошуку, фільтрації та сортування записів, що дозволяє адміністраторам швидко знаходити та редагувати необхідну інформацію. З метою підвищення продуктивності при роботі зі зв’язаними сутностями застосовано оптимізацію вибірок даних шляхом використання попереднього завантаження пов’язаних об’єктів.

Окрему увагу приділено керуванню медичним персоналом і розкладом прийомів. Для цього реалізовано інлайн-відображення часових слотів безпосередньо в адміністративній формі лікаря, що спрощує редагування графіків і зменшує кількість операцій, необхідних для супроводу системи. Адміністрування заявок пацієнтів забезпечує контроль статусів обробки звернень, збереження історії записів та захист системних полів від редагування.

Таким чином, налаштована адміністративна частина забезпечує централізоване, ефективне та безпечне управління даними інформаційної системи, що сприяє стабільній роботі сервісу запису пацієнтів і спрощує його подальший супровід та розвиток.

4.4 Інструкція користувача

Розроблений веб-застосунок призначений для інформаційного забезпечення діяльності приватної медичної клініки та надання користувачам зручного доступу до актуальних відомостей про медичні послуги. Основною ідеєю створення даного програмного продукту є формування єдиного цифрового середовища, яке дозволяє пацієнтам швидко орієнтуватися у спектрі послуг клініки, ознайомлюватися з інформацією про лікарів, їхню спеціалізацію, кваліфікацію та графік роботи, а також здійснювати запис на прийом у зручний для них час.

Метою функціонування веб-застосунку є підвищення якості взаємодії між пацієнтом і медичним закладом шляхом автоматизації процесів інформування та попереднього запису. Система спрямована на оптимізацію процесу вибору

медичних послуг, зменшення часових витрат користувачів і підвищення загального рівня організованості медичного обслуговування. Завдяки структурованому поданню інформації веб-додаток сприяє прийняттю обґрунтованих рішень щодо вибору лікаря та виду медичної допомоги.

Функціональні можливості веб-застосунку передбачають зручний перегляд переліку медичних послуг із детальними описами, доступ до інформації про лікарів та їх спеціалізації, а також реалізацію механізму онлайн-запису на прийом. Інтерфейс системи орієнтований на інтуїтивну взаємодію з користувачем, що забезпечує простоту навігації, можливість пошуку необхідної інформації та швидкий доступ до ключових розділів сайту без потреби у спеціальних технічних знаннях.

З технічної точки зору веб-застосунок реалізований із використанням сучасних веб-технологій та відповідає принципам клієнт-серверної архітектури. Серверна частина системи розроблена мовою програмування Python із застосуванням фреймворків Django та Django Rest Framework, що забезпечують модульність, масштабованість і безпеку обробки даних. Використання REST-архітектури дозволяє організувати ефективний обмін даними між сервером і клієнтською частиною.

Клієнтський інтерфейс створено на основі фреймворку Vue.js, що дає змогу реалізувати динамічну взаємодію з користувачем, швидке оновлення даних без перезавантаження сторінки та покращений користувацький досвід. Для розмітки та стилізації сторінок застосовуються мови HTML і CSS. Зберігання структурованих даних здійснюється у реляційній базі даних MySQL, яка забезпечує надійність, цілісність і ефективну обробку інформації.

До основних класів завдань, які вирішує веб-застосунок, належать:

- надання оперативного доступу до інформації про медичні послуги та персонал клініки;
- підтримка процесу попереднього запису пацієнтів на прийом до лікаря;

- відображення актуальних цін на медичні послуги;
- інформування про графік роботи та професійну кваліфікацію лікарів.

Реалізація зазначених завдань сприяє зменшенню навантаження на адміністративний персонал клініки та підвищує ефективність організації медичного обслуговування.

Розроблена інформаційна система є комплексним програмним інструментом, орієнтованим на спрощення доступу пацієнтів до медичних послуг. Ключовою особливістю системи є інтеграція функцій інформування та запису в єдиному веб-середовищі. Інтуїтивно зрозумілий інтерфейс дозволяє користувачам швидко знаходити потрібну інформацію, порівнювати послуги та обирати оптимальні варіанти лікування відповідно до власних потреб.

Система також може бути використана як основа для подальшого розширення функціональності, зокрема впровадження особистих кабінетів пацієнтів, електронних медичних карт або інтеграції з зовнішніми медичними сервісами.

Функціонування веб-застосунку безпосередньо залежить від наявності стабільного підключення до мережі Інтернет, оскільки всі дані обробляються та передаються в онлайн-режимі. У разі відсутності мережевого з’єднання доступ до сторінок сайту та його функціональних можливостей є неможливим.

Крім того, для коректної роботи системи користувач повинен використовувати сучасний веб-браузер, який підтримує актуальні веб-стандарти та технології. Недотримання цих вимог може призвести до некоректного відображення інтерфейсу або обмеження доступу до окремих функцій веб-застосунку.

4.5 Аналіз контрольного приладу

Після переходу на головну сторінку веб-сайту медичного центру «MedCare Plus» користувачі потрапляють у візуально структуроване та професійно спроєктоване цифрове середовище. Інтерфейс сторінки побудований

із застосуванням сучасних дизайнерських підходів, що забезпечують зручність сприйняття інформації та простоту навігації для користувачів різного рівня цифрової підготовки.

Центральне місце на головній сторінці займає велике тематичне зображення, яке виконує роль візуального акценту та підкреслює орієнтацію медичного закладу на якість, надійність і доступність медичних послуг.

Навігаційні та інформаційні блоки, реалізовані у вигляді гексагональних елементів, вирізняються нестандартною геометричною формою, що підвищує рівень візуальної привабливості сторінки. Таке дизайнерське рішення не лише привертає увагу користувачів, але й забезпечує швидкий доступ до основних розділів веб-сайту, зокрема: «Лікарі», «Послуги та ціни», «Про клініку» та «Контактна інформація».

Завдяки продуманій структурі головної сторінки користувачі можуть оперативно отримати необхідні відомості та безперешкодно перейти до потрібного розділу, що сприяє позитивному користувацькому досвіду та підвищує ефективність взаємодії з веб-застосунком.

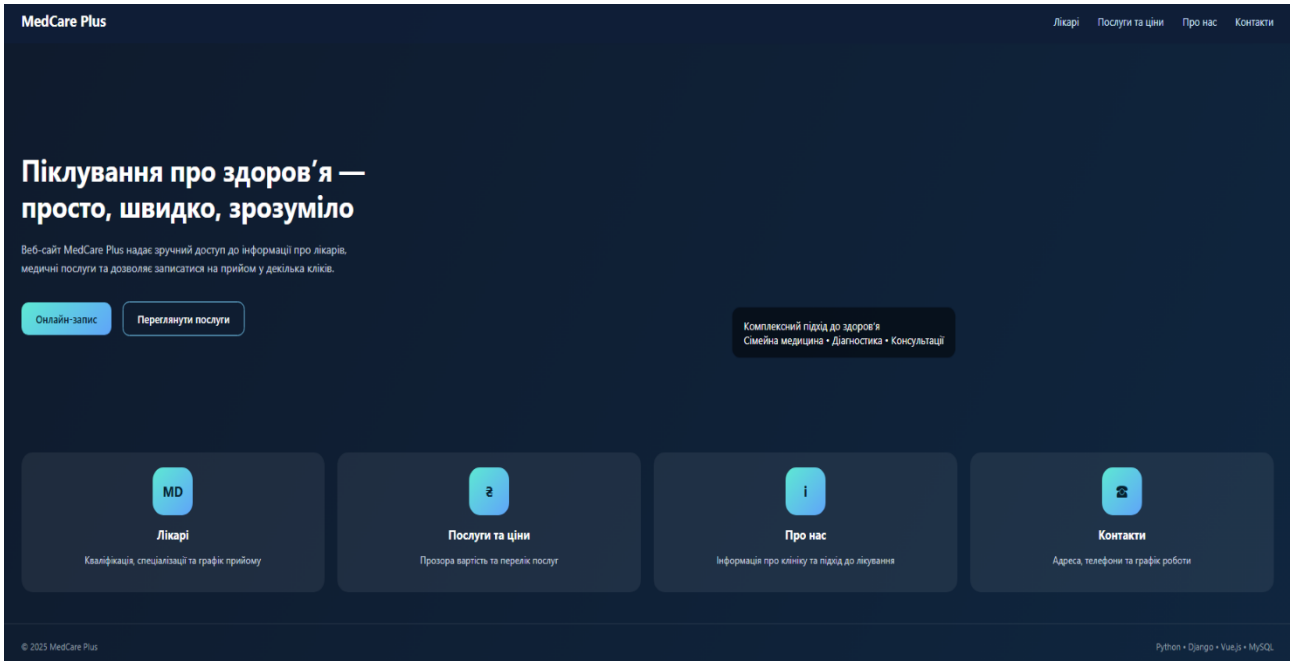

Рис. 4.3. Головна сторінка

На другому скріншоті веб-сайту медичного центру «MedCare Plus» представлено інформаційний розділ, присвячений переліку та різноманітності медичних послуг. Інтерфейс сторінки виконано в стриманому, професійному стилі, що забезпечує зручне сприйняття інформації та швидку навігацію. На сторінці наведено огляд основних напрямів медичної допомоги, серед яких дерматологія, кардіологія, неврологія, психіатрія, ревматологія, ультразвукова діагностика та інші спеціалізації.

Текстовий блок акцентує увагу на сучасному підході закладу до організації медичного обслуговування: застосуванні актуальних методик діагностики й лікування, орієнтації на індивідуальні потреби пацієнта та створенні комфортних умов перебування. Сукупність зазначених аспектів спрямована на підвищення ефективності лікування та формування позитивного користувацького досвіду під час взаємодії з веб-застосунком.

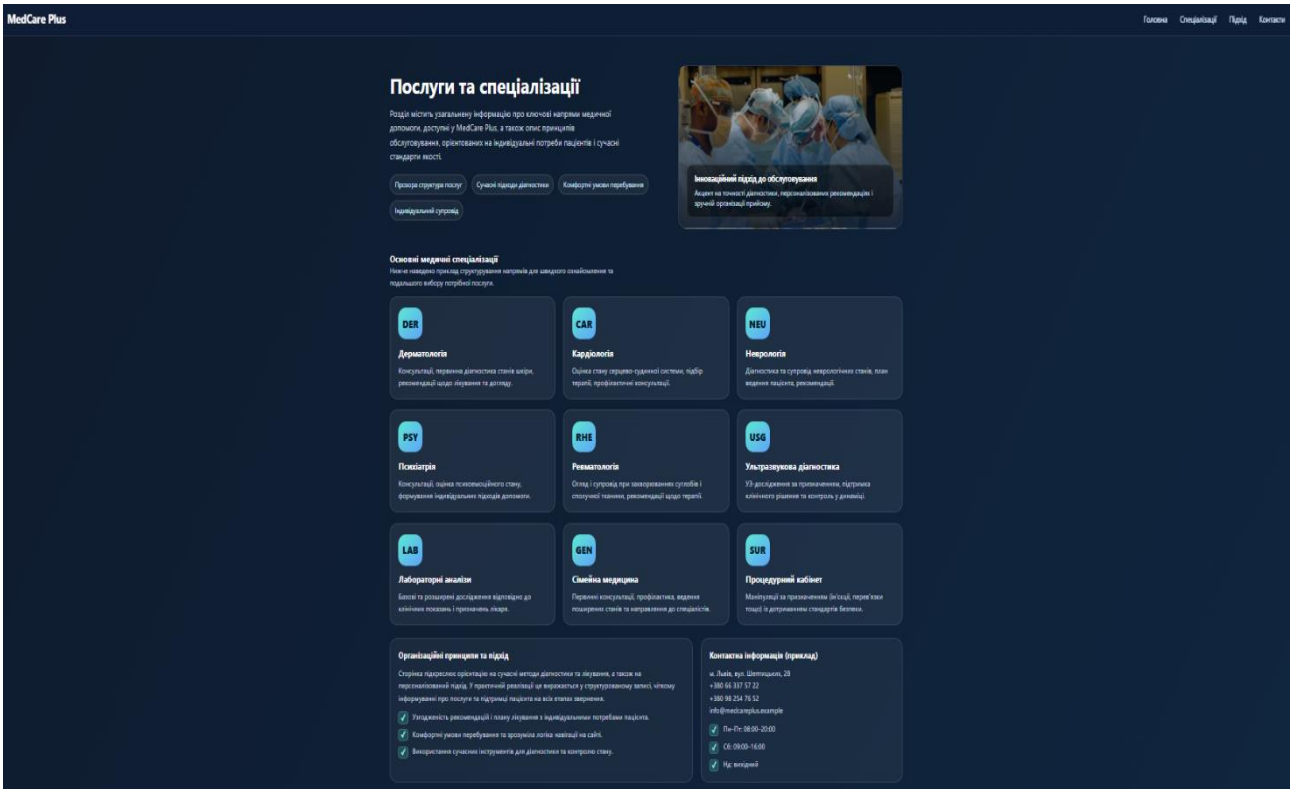

Рис. 4.4. Послуги клініки

Далі наведено розділ «Лікарі» веб-сайту медичного центру «MedCare Plus». У цьому розділі систематизовано інформацію про медичних фахівців, які працюють у клініці. Для кожного спеціаліста відображається профільна картка з фотографією, прізвищем та ім’ям, напрямом спеціалізації, а також контактним номером телефону для оперативного зв’язку. Такий формат подання даних забезпечує зручний пошук необхідного лікаря та підвищує інформативність веб-інтерфейсу для користувачів.

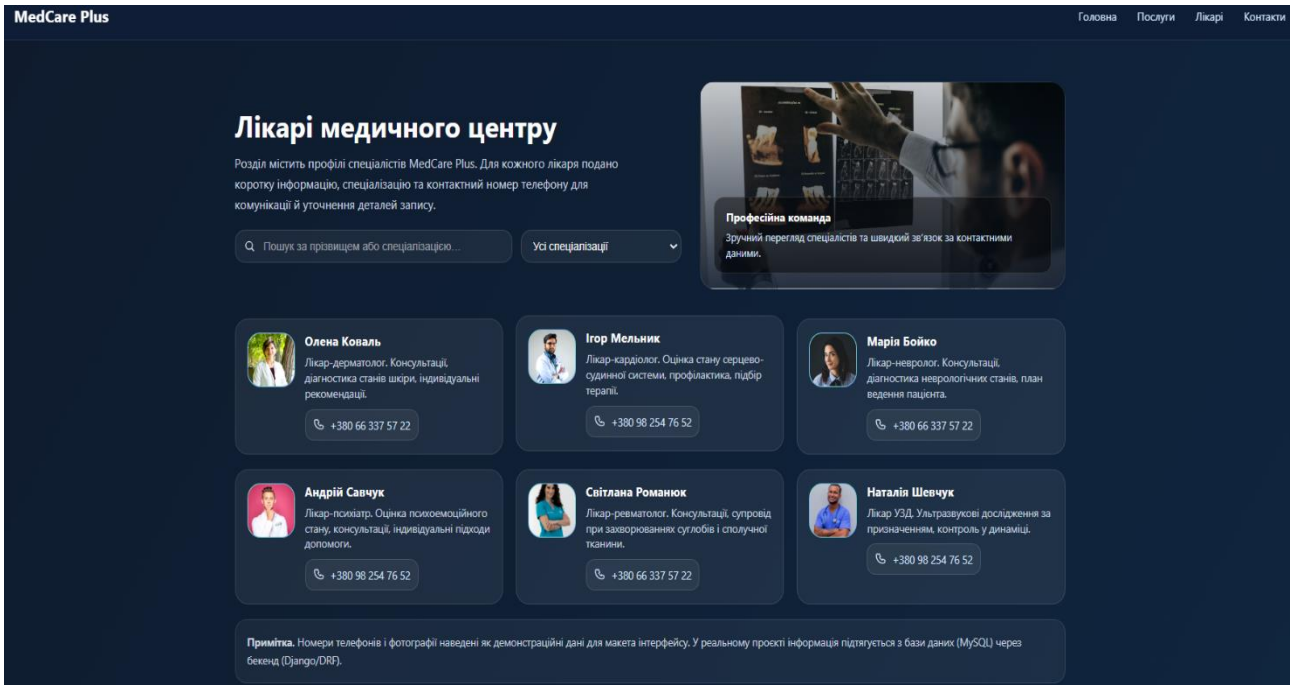

Рис. 4.5. Лікарі

Коваль Олена Вікторівна є лікарем-дерматологом медичного центру «MedCare Plus». Основним напрямом її діяльності є діагностика та лікування захворювань шкіри, а також надання індивідуальних рекомендацій щодо догляду та терапії з урахуванням особливостей стану кожного пацієнта.

Мельник Ігор Сергійович працює лікарем-кардіологом у медичному центрі «MedCare Plus». Він спеціалізується на обстеженні серцево-судинної системи, профілактиці та лікуванні кардіологічних захворювань із застосуванням сучасних клінічних підходів.

Бойко Марія Андріївна є лікарем-неврологом медичного центру «MedCare Plus». Вона проводить консультації пацієнтів з неврологічними порушеннями, здійснює діагностику стану нервової системи та формує індивідуальні плани медичного супроводу.

Савчук Андрій Михайлович працює лікарем-психіатром у медичному центрі «MedCare Plus». Його діяльність спрямована на оцінку психоемоційного стану пацієнтів, надання консультативної допомоги та формування рекомендацій відповідно до сучасних стандартів психіатричної практики.

Романюк Світлана Петрівна є лікарем-ревматологом медичного центру «MedCare Plus». Вона спеціалізується на діагностиці та лікуванні захворювань суглобів і сполучної тканини, забезпечуючи комплексний підхід до ведення пацієнтів.

Шевчук Наталія Ігорівна працює лікарем ультразвукової діагностики в медичному центрі «MedCare Plus». Вона проводить ультразвукові дослідження для уточнення діагнозів та контролю стану пацієнтів у процесі лікування.

Кожен профіль містить професійну фотографію лікаря, що дозволяє пацієнтам легко ідентифікувати спеціаліста. Профілі дизайновані чисто і професійно, з лаконічними відомостями про кожного лікаря, підсилюючи довіру та професіоналізм клініки.

Розділ «Про нас» веб-сайту медичного центру «MedCare Plus» містить узагальнену інформацію про діяльність закладу, його місію, принципи роботи та підхід до надання медичних послуг. Основною метою функціонування медичного центру є забезпечення якісної, доступної та сучасної медичної допомоги з орієнтацією на індивідуальні потреби кожного пацієнта.

У своїй діяльності медичний центр дотримується принципів доказової медицини, системного підходу до діагностики та лікування, а також безперервного професійного розвитку медичного персоналу. Значна увага приділяється створенню комфортних умов для пацієнтів, оптимізації процесів

запису на прийом та забезпеченню зрозумілої комунікації між лікарем і пацієнтом.

Використання сучасних інформаційних технологій у роботі веб-застосунку дозволяє структурувати інформацію про медичні послуги, лікарів та організаційні процеси, що сприяє підвищенню ефективності взаємодії користувачів із медичним закладом. Таким чином, веб-сайт виступає не лише інформаційним ресурсом, а й інструментом підтримки управлінських та комунікаційних процесів у сфері медичного обслуговування.

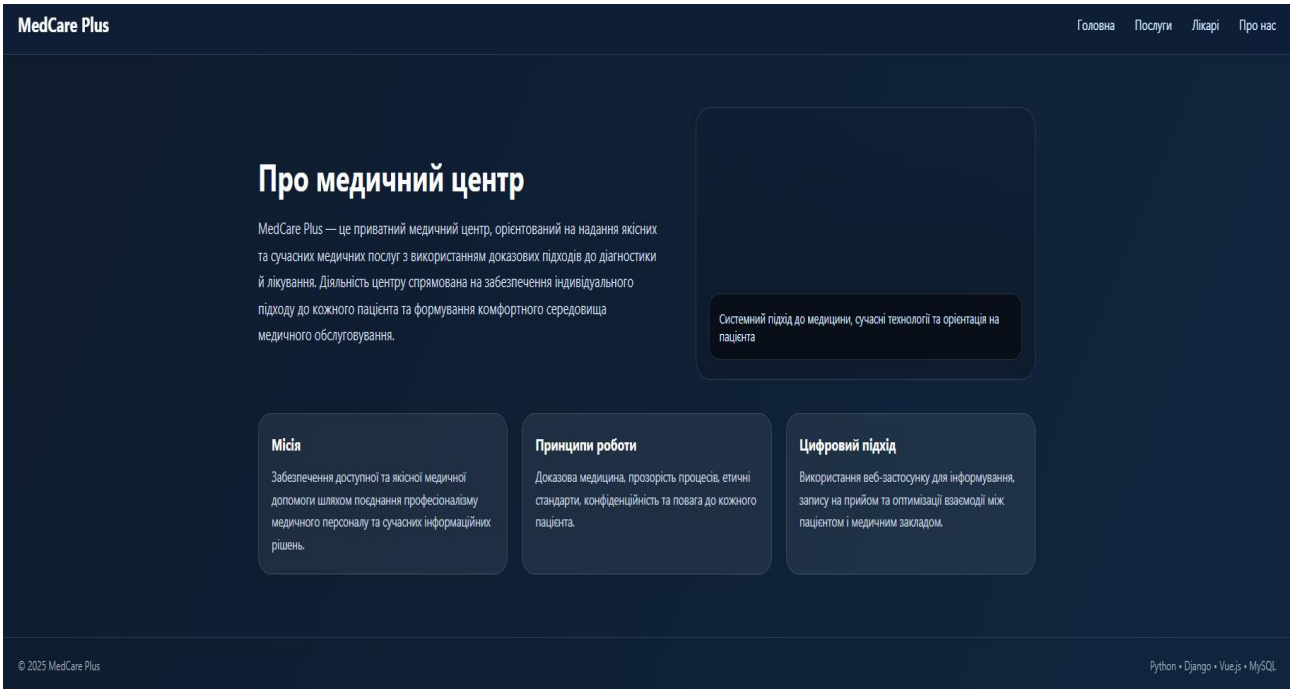

Рис. 4.6. Про нас

Розділ «Контакти» веб-сайту медичного центру «MedCare Plus» призначений для забезпечення швидкого та однозначного доступу користувачів до актуальної контактної інформації закладу. Сторінка містить структуровані відомості про місцезнаходження медичного центру, канали зв’язку (телефонні номери та електронну пошту), а також графік роботи. Візуальна організація даних реалізована таким чином, щоб мінімізувати час пошуку необхідної інформації та підвищити зручність взаємодії користувача з веб-інтерфейсом.

Додатково сторінка включає блок з інтерактивним поданням місця розташування (макет карти) та форму зворотного зв’язку, що сприяє оперативній комунікації між пацієнтом і адміністрацією медичного центру. Такий підхід підвищує функціональну цінність веб-застосунку, забезпечуючи не лише інформування, а й підтримку комунікаційних процесів у рамках сервісу запису та консультацій.

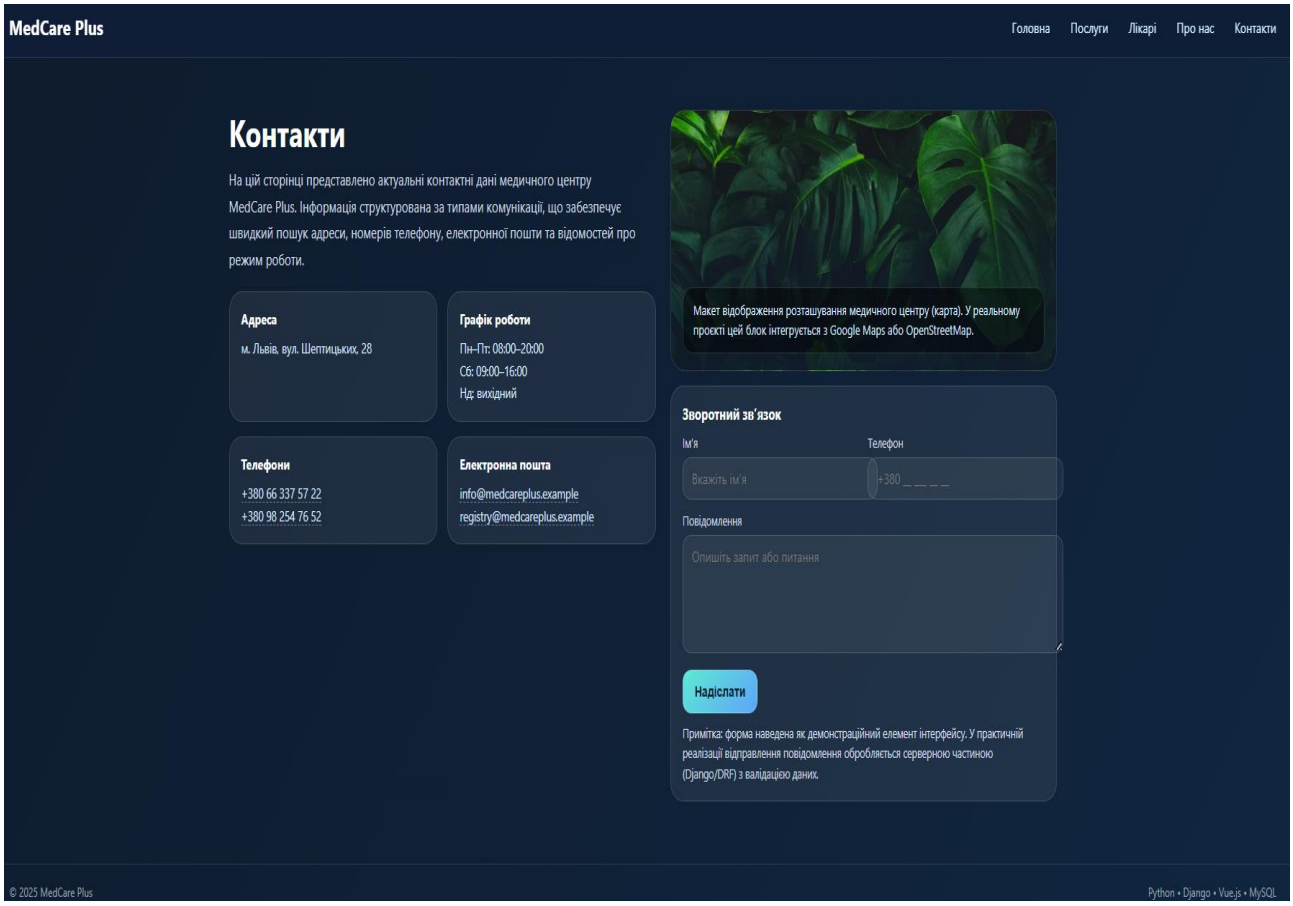

Рис. 4.7. Контакти

Розділ «Запис на прийом» веб-сайту медичного центру «MedCare Plus» призначений для автоматизації процесу попереднього запису пацієнтів на консультацію до лікаря. Основною функціональною метою даної сторінки є спрощення та оптимізація взаємодії між пацієнтом і медичним закладом шляхом надання зручного інтерфейсу для вибору спеціаліста, дати та часу прийому.

Сторінка побудована за принципом покрокового введення даних, що дозволяє мінімізувати кількість помилок під час заповнення форми та забезпечує зрозумілу логіку взаємодії з користувачем. Користувач має можливість обрати лікаря або спеціалізацію, вказати контактні дані та залишити додаткову інформацію щодо свого запиту. Уся введена інформація використовується для формування заявки на прийом, яка в реальній реалізації обробляється серверною частиною системи.

Інтеграція сторінки запису з серверною частиною веб-застосунку (Django та Django Rest Framework) дозволяє реалізувати механізм перевірки доступності лікарів, обробки заявок та подальшого інформування пацієнтів. Таким чином, розділ «Запис на прийом» є ключовим елементом функціональної структури веб-сайту та забезпечує підвищення ефективності організації медичного обслуговування.

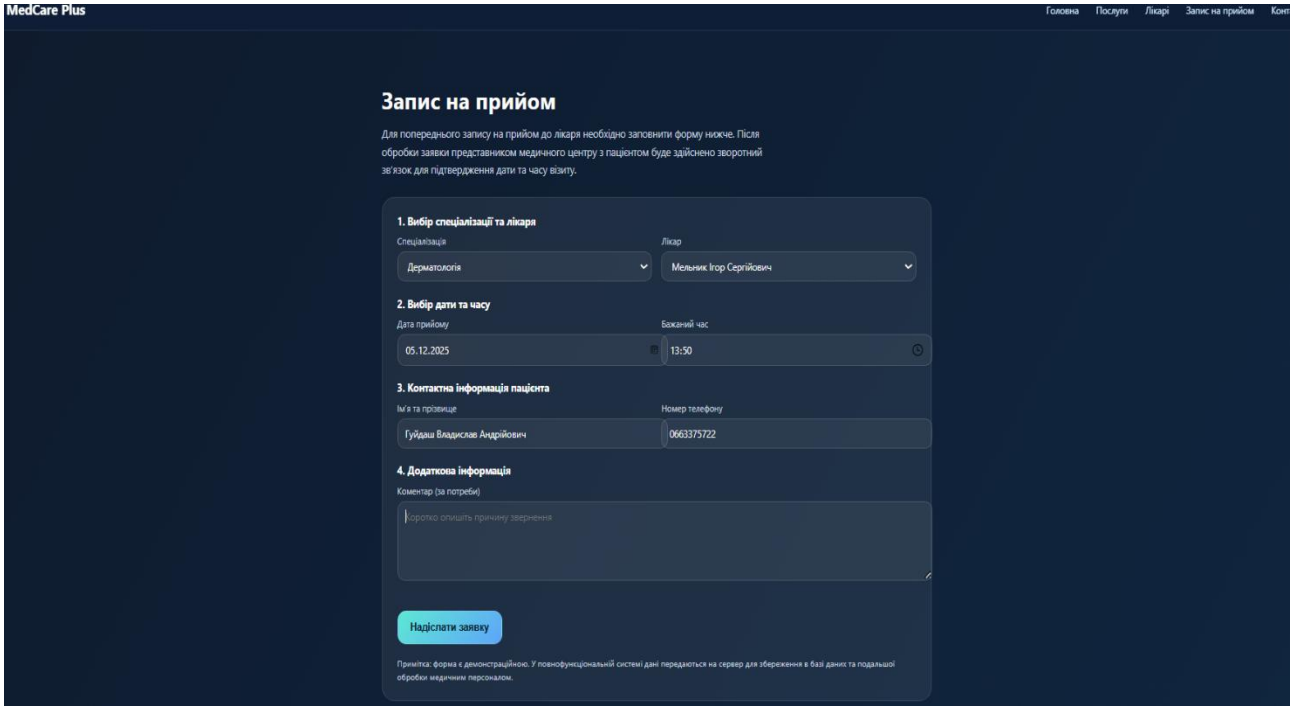

Рис. 4.8. Запис на прийом

Інтеграція серверної частини веб-сайту медичного центру MedCare Plus реалізована з використанням фреймворку Django, який забезпечує надійну

архітектуру, масштабованість та безпеку веб-застосунку. Django використовується як основний інструмент для обробки бізнес-логіки, управління даними та взаємодії з клієнтською частиною сайту.

Вкладка «Послуги» у вебзастосунку медичної клініки спроєктована як функціональний модуль користувацького інтерфейсу, орієнтований на швидкий доступ до переліку медичних послуг, їхніх характеристик та процедур онлайн-запису. Основною метою модуля є підвищення зручності взаємодії користувача з інформаційною системою за рахунок структурованої подачі даних, мінімізації кількості кроків до цільової дії та забезпечення прозорості вартості й доступності послуг.

Інформаційна архітектура сторінки реалізована за принципом ієрархічного групування: послуги згруповано за медичними напрямами (наприклад, консультації, діагностика, лабораторні дослідження), що зменшує когнітивне навантаження та спрощує навігацію. Для підтримки сценаріїв пошуку передбачено механізми фільтрації та сортування, які дають змогу відбирати записи за напрямом, ціновим діапазоном і часовою доступністю. Такий підхід узгоджується з вимогами до сучасних інформаційних систем у сфері охорони здоров’я, де критичними є швидкість отримання інформації, зрозумілість інтерфейсу та коректність відображення актуальних даних.

Функціонально модуль «Послуги» підтримує дві ключові операції: (1) ознайомлення з параметрами послуги (опис, тривалість, підготовка, вартість) та (2) ініціювання запису на прийом через інтеграцію з модулем розкладу. Вибір послуги виступає вхідною подією для процесу бронювання, під час якого система формує доступні часові слоти відповідно до графіків лікарів і правил надання послуг. Це забезпечує узгодженість даних між каталогом послуг і підсистемою планування та зменшує ризик помилок, пов’язаних із людським фактором.

Окрему увагу приділено вимогам юзабіліті та доступності: використано виразні візуальні акценти для основних дій, передбачено зрозумілі підказки й стан інтерфейсу, а також уніфіковано подання інформації у вигляді карток для

пришвидшення порівняння послуг. Візуальна стилістика сторінки витримана в єдиній дизайн-системі вебзастосунку (темна палітра, контрастні акценти, м’які переходи), що підвищує цілісність сприйняття та підтримує впізнаваність сервісу. У підсумку розроблений модуль забезпечує надійну та масштабовану основу для подальшого розширення каталогу послуг і покращення сервісних сценаріїв взаємодії пацієнтів із медичним закладом.

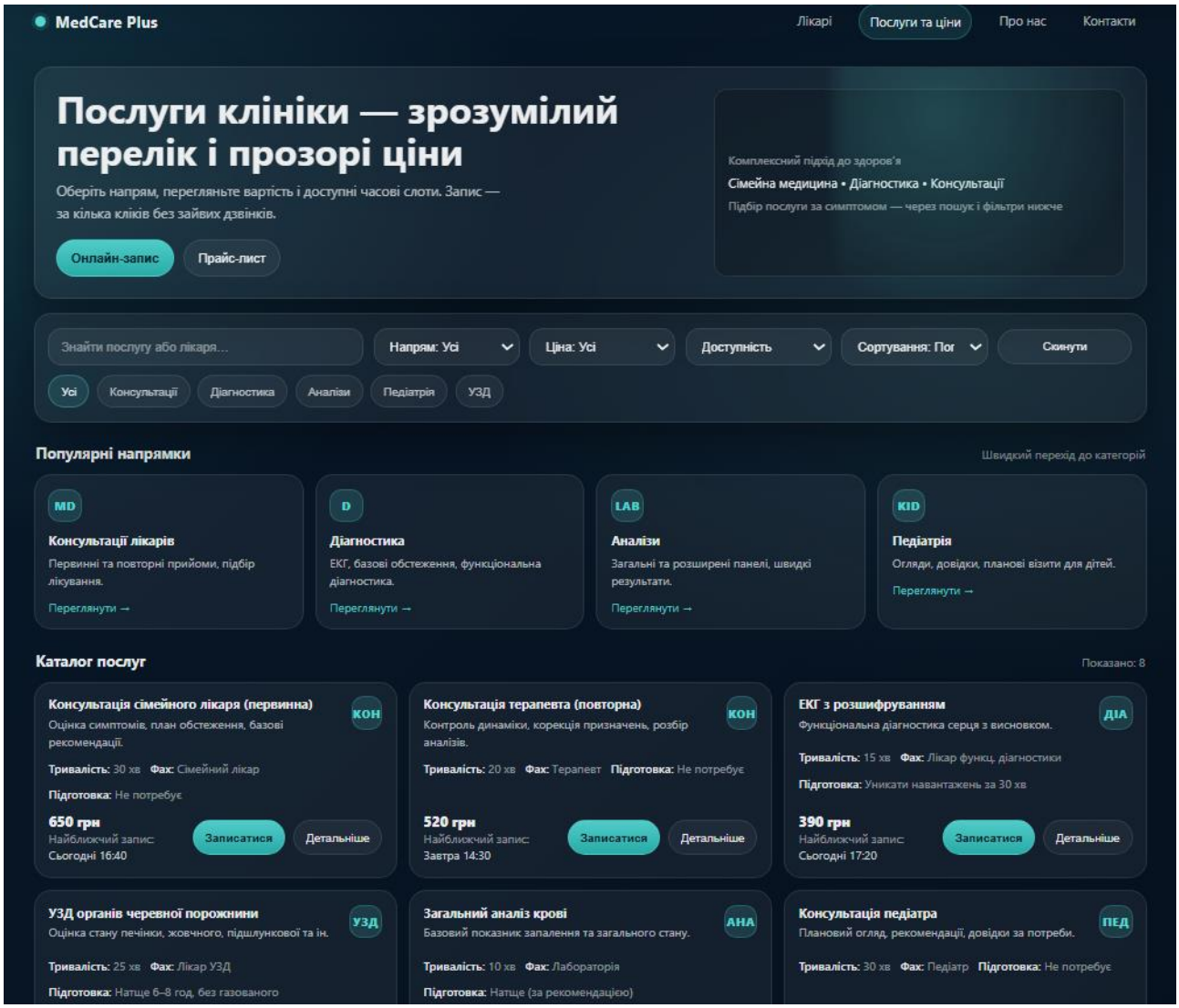

Рис. 4.9. Послуги і ціни із записом

Архітектура застосунку побудована за принципом Model–View–Controller (MVC), реалізованого в Django у вигляді шаблону Model–View–Template (MVT). Такий підхід дозволяє чітко розділити відповідальність між

компонентами системи, що підвищує підтримуваність коду та спрощує подальше розширення функціоналу.

Моделі Django використовуються для опису основних сутностей предметної області, зокрема лікарів, спеціалізацій, медичних послуг, заявок на запис до лікаря та контактної інформації клініки. Кожна модель відповідає окремій таблиці в базі даних MySQL, що забезпечує структуроване зберігання інформації та підтримку реляційних зв’язків між даними.

Завдяки використанню Django ORM забезпечується абстракція від прямого написання SQL-запитів, що зменшує ймовірність помилок та підвищує рівень безпеки. ORM автоматично генерує оптимізовані запити до бази даних, а також підтримує механізми міграцій, які дозволяють керувати змінами структури БД протягом життєвого циклу проєкту.

Для взаємодії серверної частини з фронтендом використовується Django Rest Framework (DRF). За його допомогою реалізовано REST-API, яке забезпечує обмін даними у форматі JSON між сервером та клієнтським інтерфейсом сайту. Такий підхід дозволяє відокремити логіку представлення від логіки обробки даних та забезпечує гнучкість у виборі технологій фронтенду.

Форми, зокрема форма запису на прийом, передають введені користувачем дані на сервер, де вони проходять валідацію, зберігаються в базі даних та можуть бути оброблені адміністраторами або медичним персоналом. Це забезпечує цілісність даних та контроль доступу до інформації.

Для управління вмістом сайту використовується стандартна адміністративна панель Django, яка дозволяє здійснювати CRUD-операції (створення, читання, оновлення та видалення) над усіма ключовими сутностями системи. Через адміністративний інтерфейс здійснюється управління профілями лікарів, переліком медичних послуг, спеціалізаціями та заявками пацієнтів.

Наявність адміністративної панелі значно спрощує супровід системи, оскільки не потребує додаткової розробки окремого інтерфейсу для внутрішнього управління даними.

Django забезпечує вбудовані механізми безпеки, зокрема захист від CSRF-атак, SQL-ін’єкцій та XSS-уразливостей. Система автентифікації та авторизації дозволяє розмежовувати права доступу між адміністраторами, персоналом та іншими користувачами. Це є критично важливим аспектом для веб-застосунків у сфері охорони здоров’я, де обробляється конфіденційна інформація.

Рис. 4.10. Панель Customers applications

Панель Customers applications у адміністративному інтерфейсі Django є функціональним модулем для централізованого управління зверненнями (заявками) користувачів веб-сайту медичного центру MedCare Plus. Її призначення полягає у забезпеченні повного циклу роботи із заявками: від фіксації факту звернення пацієнта через форму на сайті до подальшої обробки, підтвердження, уточнення деталей та контролю виконання. Такий підхід дозволяє систематизувати комунікацію між пацієнтом і медичним закладом та мінімізувати ризики втрати інформації.

У контексті веб-застосунку заявки формуються під час взаємодії користувача з інтерфейсом, зокрема при заповненні форми запису на прийом або запиту консультації. Після надсилання дані заявки передаються на сервер,

проходять валідацію та зберігаються в базі даних MySQL. У результаті в адміністративній панелі формується структурований запис, який відображається у розділі Customers applications. Адміністратор або відповідальний працівник центру отримує можливість оперативно переглянути нові звернення, провести їх первинний аналіз та прийняти рішення щодо подальших дій.

Як правило, кожен запис у Customers applications містить набір полів, що описують заявку та дозволяють однозначно ідентифікувати звернення. До таких даних належать персональні відомості пацієнта (наприклад, ПІБ або ім’я), контактний номер телефону та/або електронна пошта, обрана спеціалізація або конкретний лікар, бажана дата й час прийому, а також текстовий коментар, у якому пацієнт може уточнити причину звернення. Додатково в системі можуть фіксуватися службові атрибути, наприклад дата та час створення заявки, її поточний статус (нова, в роботі, підтверджена, завершена, скасована), а також відповідальний працівник, який здійснює обробку.

Адміністративний інтерфейс Django у межах цієї панелі підтримує стандартні операції керування даними: створення записів (у разі необхідності внесення заявки вручну), перегляд, редагування та видалення. Водночас основною практичною функцією є саме оперативна обробка звернень, що включає перевірку коректності контактних даних, уточнення часу прийому, перевірку доступності лікаря, а також фіксацію факту підтвердження заявки. Наявність механізму редагування дозволяє вносити зміни до запису у випадках, коли пацієнт змінює запит або коли адміністратор коригує дату, час чи спеціаліста після телефонного узгодження.

Для підвищення ефективності роботи з великим обсягом заявок у Django Admin зазвичай використовується функціональність фільтрації, сортування та пошуку. Це дозволяє, наприклад, швидко відібрати заявки за статусом, датою створення, обраною спеціалізацією або конкретним лікарем. Сортування за датою дає змогу контролювати актуальність звернень і забезпечувати їх обробку в порядку надходження. Пошук за номером телефону або прізвищем спрощує повторну ідентифікацію пацієнта та роботу з повторними зверненнями.

З позиції інформаційної безпеки модуль Customers applications має критичне значення, оскільки працює з персональними даними. Тому доступ до нього обмежується механізмами авторизації Django, а права користувачів налаштовуються на рівні груп і дозволів. Це забезпечує розмежування доступу між адміністраторами, медичним персоналом і операторами контакт-центру, а також знижує ризики несанкціонованого перегляду або модифікації інформації.

У підсумку, панель Customers applications виступає ключовим елементом серверної частини веб-застосунку MedCare Plus, оскільки забезпечує структуроване збирання та контроль заявок, інтегрує процес запису на прийом з адміністративним управлінням і підвищує загальну організаційну ефективність медичного сервісу.

4.6 Висновок до четвертого розділу

У ході розробки та аналізу адміністративного модуля Customers applications було встановлено, що даний розділ є ключовим елементом серверної частини веб-застосунку медичного центру MedCare Plus. Його функціональне призначення полягає у централізованому зборі, зберіганні та обробці заявок пацієнтів, що надходять через веб-інтерфейс сайту. Реалізація цього модуля на базі фреймворку Django забезпечує надійну інтеграцію процесу онлайн-запису з внутрішніми адміністративними процесами медичного закладу.

Використання адміністративної панелі Django дозволяє ефективно керувати заявками за допомогою стандартних операцій перегляду, редагування та контролю статусу звернень, що сприяє підвищенню оперативності обробки запитів та зменшенню навантаження на персонал. Наявність механізмів фільтрації, сортування та пошуку забезпечує зручність роботи з великим обсягом даних і підвищує загальну продуктивність управлінських процесів.

Окрему роль у функціонуванні модуля відіграють вбудовані засоби безпеки та контролю доступу, які гарантують захист персональних даних пацієнтів і відповідність сучасним вимогам до інформаційної безпеки в

медичних інформаційних системах. Завдяки цьому панель Customers applications не лише виконує роль інструменту реєстрації звернень, але й слугує важливим компонентом комплексної системи організації медичного обслуговування.

Таким чином, реалізація модуля Customers applications є обґрунтованим та ефективним рішенням, що забезпечує цілісність інформаційних потоків, покращує комунікацію між пацієнтом і медичним центром та підвищує якість управління процесом запису на прийом у веб-застосунку MedCare Plus.

ВИСНОВКИ

У межах даного наукового дослідження основну увагу приділено процесу проєктування та впровадження інформаційної системи для приватного медичного закладу. Розробка системи базувалася на аналізі наявних інформаційних рішень у галузі охорони здоров’я та визначенні специфічних функціональних і організаційних вимог, характерних для діяльності клініки. Такий підхід дозволив обґрунтувати доцільність використання сучасних програмних технологій, спрямованих на оптимізацію внутрішніх процесів та підвищення ефективності взаємодії з пацієнтами.

У першому розділі роботи здійснено огляд сучасного стану інформаційних систем, що застосовуються в медичних установах, у результаті чого було виявлено низку проблемних аспектів, пов’язаних із зручністю експлуатації та безпекою даних. Особливу увагу зосереджено на питаннях захисту персональної інформації пацієнтів і створення умов для комфортної роботи медичного персоналу.

Другий розділ присвячено системному аналізу предметної області, який охоплює розробку функціональної структури та навігаційної моделі інформаційної системи. Акцент зроблено на проєктуванні інтерфейсу користувача з урахуванням специфіки професійної діяльності медичних працівників. Побудова детальної моделі взаємодії між окремими модулями системи забезпечила їх узгоджену роботу та підвищила загальну ефективність функціонування програмного продукту.

У третьому розділі наведено обґрунтування вибору програмних засобів і технологій, використаних під час реалізації проєкту. Зокрема, застосування фреймворку Django для серверної частини та бібліотеки Vue.js для клієнтського інтерфейсу дозволило створити гнучку, надійну та масштабовану платформу.

Такий технологічний стек сприяв оптимізації обробки даних і забезпечив високу швидкодію системи при роботі з користувацькими запитами.

Четвертий розділ містить опис практичної реалізації інформаційної системи, включаючи аналіз її архітектури, основних компонентів та механізмів інтеграції у щоденну діяльність медичного закладу. Розглянуто вплив впровадженої системи на організацію робочих процесів клініки та покращення якості обслуговування пацієнтів.

Узагальнюючи результати проведеного дослідження, можна стверджувати, що створена інформаційна система для приватної клініки «MedCare Plus» є прикладом ефективного застосування сучасних інформаційних технологій у медичній сфері. Її впровадження сприяє вирішенню організаційних і технічних завдань, підвищує рівень сервісу та забезпечує позитивний досвід взаємодії пацієнтів із медичним закладом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bernstein P. A., Newcomer E. Principles of Transaction Processing. — Morgan Kaufmann, 2009Codd E. F. A Relational Model of Data for Large Shared Data Banks. — Communications of the ACM, 1970.
2. Pavlo A. et al. Self-Driving Database Management Systems. — CIDR Conference, 2017.
3. Gray J., Reuter A. Transaction Processing: Concepts and Techniques. — Morgan Kaufmann, 1993.
4. Viega J., McGraw G. Building Secure Software. — Addison-Wesley, 2002.
5. Fowler M. Patterns of Enterprise Application Architecture. — Addison-Wesley, 2002.
6. Bernstein P. A., Newcomer E. Principles of Transaction Processing. — Morgan Kaufmann, 2009
7. DRF Official Documentation. URL: https://www.django-rest-framework.org/
8. Simple is Better Than Complex – DRF. URL: https://simpleisbetterthancomplex.com/
9. Django-Cors-Headers Documentation. URL: https://pypi.org/project/django-cors-headers/
10. MySQLClient Documentation. URL: https://mysqlclient.readthedocs.io/
11. Mozilla Developer Network – HTML. URL: https://developer.mozilla.org/en-US/docs/Web/HTML
12. W3Schools – HTML. URL: https://www.w3schools.com/html/
13. Mozilla Developer Network – CSS. URL: https://developer.mozilla.org/en-US/docs/Web/CSS
14. W3Schools – CSS. URL: https://www.w3schools.com/css/
15. Vue.js Official Documentation. URL: https://vuejs.org/guide/introduction
16. Vue School. URL: https://vueschool.io/
17. Vue Router Documentation. URL: https://vuejs.org/guide/scaling-up/routing
18. Font Awesome Documentation. URL: https://docs.fontawesome.com/

19. Google Fonts. URL: https://fonts.google.com/
20. Planet MySQL. URL: https://planet.mysql.com/
21. SQL Shack. URL: https://www.sqlshack.com/
22. Database Journal. URL: https://www.databasejournal.com/
23. Simple Programmer - Django. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django
24. HTML Dog. URL: https://www.htmldog.com/
25. SitePoint HTML & CSS. URL: https://www.sitepoint.com/html-css/html/
26. Web Designer Depot. URL: https://www.webdesignerdepot.com/
27. Python for Beginners. URL: https://www.pythonforbeginners.com/
28. LearnDjango. URL: https://learndjango.com/
29. Django Central. URL: https://djangocentral.com/
30. HTML5 Up. URL: https://html5up.net/
31. Quackit. URL: https://www.quackit.com/
32. CSS Author. URL: https://cssauthor.com/
33. VueDose. URL: https://vuedose.tips/
34. Vue Mastery. URL: https://www.vuemastery.com/
35. Kircher M., Jain P., Corsaro A. The Three-Tier Architecture Pattern: Language-Level Support for the Three-Tier Architecture Pattern. EuroPLoP, 2001. (PDF).
36. Yusmita J. C. Optimizing Database Access Strategy: Performance Benchmarking Prisma ORM vs Optimized Raw SQL on PostgreSQL. Procedia Computer Science, 2025.
37. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3 : RFC 8446. IETF, 2018. URL: (RFC 8446).
38. Fielding R., Reschke J. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing : RFC 7230. RFC Editor, 2014.