НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення інформаційної системи для реєстрації пацієнтів лікарні»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи ст. викладач	_____ (підпис)	**Юрій МІЛОВІДОВ**
Виконав	_____ (підпис)	**Віталій ДЕНИСЕНКО**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 202 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Денисенку Віталію Анатолійовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення інформаційної системи для реєстрації пацієнтів лікарні»

затверджена наказом від «19» грудня 2025 р. № 3196 «С»
Термін подання завершеної роботи на кафедру «___» _____ 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: опис предметної області реєстрації пацієнтів лікарні та існуючих програмних рішень для онлайн-запису до лікарів.
Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області
2. Проєктування інформаційного та програмного забезпечення
3. Розробка інформаційного та програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання «___» _____ 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Юрій МІЛОВІДОВ**

Завдання взяв до виконання _____ (підпис) **Віталій ДЕНИСЕНКО**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – інтерфейс прикладного програмування.

CORS (Cross-Origin Resource Sharing) – механізм обміну ресурсами між різними джерелами.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

DRF (Django REST Framework) – фреймворк для побудови REST API на основі Django.

ERD (Entity-Relationship Diagram) – діаграма сутність-зв’язок.

HTML (HyperText Markup Language) – мова розмітки гіпертексту.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту.

HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передачі гіпертексту.

JS (JavaScript) – мова програмування для веб-розробки.

JSON (JavaScript Object Notation) – формат обміну даними.

JWT (JSON Web Token) – стандарт для безпечної передачі інформації між сторонами.

MVT (Model-View-Template) – патерн проєктування, що використовується у Django.

ORM (Object-Relational Mapping) – технологія зв’язування бази даних з об'єктами програми.

PBKDF2 (Password-Based Key Derivation Function 2) – функція виведення ключа на основі пароля.

RBAC (Role-Based Access Control) – контроль доступу на основі ролей.

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів.

SMTP (Simple Mail Transfer Protocol) – протокол для відправки електронної пошти.

SQL (Structured Query Language) – мова структурованих запитів.

СУБД – система управління базами даних.

UML (Unified Modeling Language) – уніфікована мова моделювання.

URL (Uniform Resource Locator) – уніфікований локатор ресурсів.

БД – база даних.

ПЗ – програмне забезпечення.

ВСТУП

Актуальність теми. У сучасному світі цифровізація охоплює абсолютно всі сфери життя, включаючи охорону здоров'я. Лікарні в Україні стикаються з такими проблемами, як: черги пацієнтів в реєстратурі, складність управління розкладом лікарів без автоматизації, відсутність єдиного електронного журналу записів та обмежений доступ пацієнтів до власних медичних даних. Популярність існуючих систем, таких як Helsi.me та Doc.ua, підтверджує зростаючу потребу в ефективних веб-орієнтованих системах для автоматизації медичних процесів, але кожна з них має деякі недоліки.

Розробка інформаційної системи "Mediculus" дозволяє автоматизувати процеси запису пацієнтів на прийом, зберігання медичних записів та управління розкладом лікарів без необхідності особистого звернення до реєстратури.

Мета роботи полягає у розробці повноцінного веб-застосунку інформаційної системи "Mediculus" для реєстрації пацієнтів лікарні, що забезпечує автоматизацію процесу онлайн-запису до лікарів, зберігання результатів прийомів в електронному вигляді та управління медичними даними з розмежуванням прав доступу для різних типів користувачів.

Для досягнення мети було поставлено такі завдання:

1) Проаналізувати предметну область та існуючі програми-аналоги;
2) Визначити функціональні та нефункціональні вимоги до системи;
3) Побудувати UML-діаграми для візуального представлення предметної області;
4) Спроєктувати базу даних та архітектуру програмної системи;
5) Реалізувати серверну та клієнтську частини системи з використанням сучасних технологій;
6) Провести тестування системи та виконати розгортання на веб-хостингу.

Об'єктом дослідження є процес реєстрації пацієнтів та запису на прийом до лікаря в лікарні.

Предметом дослідження є програмне забезпечення веб-орієнтованої інформаційної системи для реєстрації пацієнтів лікарні.

Практичне значення одержаних результатів. Розроблена система “Mediculus” може бути впроваджена в роботу лікарень для автоматизації процесу запису пацієнтів на прийом. Результатом роботи є повністю функціональний веб-застосунок готовий до використання, який може бути використаний як уже готове рішення, або взятий за основу для подальшого вдосконалення і адаптації під потреби конкретного медичного закладу. Вихідний код проєкту розміщено у публічному репозиторії на GitHub.

Методи та технології. У роботі використовуються: мова програмування Python, фреймворк Django, бібліотека Django REST Framework для побудови REST API, система управління базами даних PostgreSQL, фреймворк Bootstrap 5 та Vanilla JavaScript для клієнтської частини. Авторизація реалізована на основі JWT-токенів з механізмом розмежування прав доступу на основі ролей. Розгортання системи виконано на веб-хостингу Railway з використанням Cloudinary для зберігання медіафайлів та Brevo для відправки електронних листів.

Апробація. Результати дослідження представлено у вигляді тез доповіді “Програмне забезпечення інформаційної системи для реєстрації пацієнтів лікарні” на науково-практичній конференції студентів і аспірантів “Теоретичні та прикладні аспекти розробки комп’ютерних систем” факультету інформаційних технологій НУБіП України, секція “Технології розробки програмного забезпечення та інформаційних управляючих систем”, квітень 2026 року.

Структура записки. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 68 сторінок, кількість використаних джерел – 21, кількість додатків – 5.

У першому розділі проведено аналіз предметної області, визначено функціональні та нефункціональні вимоги до системи, побудовано UML-

діаграми прецедентів, послідовності, активності та основні абстракції системи, а також проведено аналіз існуючих програмних рішень та сформульовано постановку завдання.

У другому розділі виконано проєктування інформаційного та програмного забезпечення: побудовано логічну модель даних у вигляді ER-діаграми, перевірено відповідність нормальним формам, побудовано діаграми класів, кооперацій, пакетів та компонентів.

У третьому розділі обґрунтовано вибір СУБД PostgreSQL, описано розробку інформаційної бази засобами Django ORM, обрано інструментарій серверної та клієнтської частин, а також представлено алгоритми ключових процесів системи з фрагментами коду.

У четвертому розділі описано тестування системи з наведенням автоматизованих та ручних тестових сценаріїв, визначено вимоги до апаратного та програмного забезпечення, наведено діаграму розгортання та інструкцію з розгортання системи на платформі Railway.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Медична сфера є однією з найважливіших у суспільному житті тому, що від того, наскільки швидко і зручно людина може потрапити до лікаря, залежить не лише її комфорт, але й стан здоров'я в загальному. Не дивлячись на розвиток цифрових технологій, у більшості медичних закладів України процес запису пацієнтів досі відбувається вручну: через особисте звернення до реєстратури або телефонний дзвінок [1].

Цей формат роботи має кілька великих недоліків. Черги вранці – звична ситуація для будь-якої лікарні, особливо коли велика кількість пацієнтів бажають записатись одночасно. Ручне ведення журналів підвищує ймовірність таких проблем, як: подвійний запис на один і той самий час, плутанина у прізвищах, загублені дані. У невеликих приватних лікарнях, де немає єдиної електронної системи, доступ до медичної історії пацієнта взагалі може залежати від того, чи знайдуть потрібну паперову картку [2].

Предметна область, яку охоплює дана система, включає взаємодію трьох основних учасників: пацієнтів, лікарів та адміністраторів лікарні.

Пацієнт – це особа, яка звертається за медичними послугами або допомогою. У більшості лікарень для запису на прийом він змушений особисто прийти до реєстратури або зателефонувати, що не завжди вдається зробити з першого разу. Результати прийому фіксуються лікарем у паперовій картці, до якої пацієнт самостійно не має доступу.

Лікар – це фахівець, який веде прийом відповідно до розкладу. Після кожного прийому він заповнює медичну документацію, встановлює діагноз і призначає лікування. Тобто, велика частина робочого часу лікаря витрачається на паперову роботу, а не на роботу з пацієнтом.

Адміністратор – працівник реєстратури, який відповідає за ведення журналу записів, розподіл пацієнтів між лікарями та контроль їхніх розкладів. Коли пацієнтів багато, утримати актуальний розклад у голові чи паперовому журналі стає практично неможливо.

Автоматизація цих процесів є головною метою розробки системи “Mediculus”. Замість черги до реєстратури, пацієнт самостійно обирає лікаря і записується онлайн у будь-який вільний час. Після прийому результати одразу доступні в особистому кабінеті. Лікар отримує зручний інтерфейс для ведення медичних карток у цифровому вигляді. Адміністратор бачить всю необхідну інформацію в одному місці: розклади, записи, статистику [3].

1.2 Аналіз вимог до системи

Щоб визначити завдання та логіку роботи системи, ще до початку розробки, вимоги було поділено на три групи: бізнес-вимоги, функціональні та нефункціональні. [20]

Бізнес-вимоги:

- забезпечити зручний та безпечний онлайн-запис пацієнтів до лікарів без необхідності особистого звернення до реєстратури;
- зменшити навантаження на працівників реєстратури при обробці записів;
- мінімізувати помилки при веденні медичних записів;
- забезпечити зберігання медичних даних та доступ до них в одному місці;
- підвищити якість обслуговування пацієнтів шляхом автоматизації рутинних процесів.

Функціональні вимоги для пацієнтів:

- реєстрація та авторизація в системі з верифікацією електронної пошти;
- пошук лікаря за спеціальністю, прізвищем чи ім’ям;

- перегляд профілю лікаря, його досвіду та розкладу роботи;
- запис на прийом на конкретну дату та час;
- перегляд та управління своїми записами з різними статусами;
- можливість скасування запланованого прийому;
- перегляд результатів прийому та призначеного лікування;
- отримання сповіщень про підтвердження та зміни записів.

Функціональні вимоги для лікарів:

- авторизація в системі з персональним обліковим записом;
- перегляд власного розкладу та списку запланованих пацієнтів;
- завершення прийому з внесенням діагнозу та призначення;
- перегляд статистики прийомів у особистому кабінеті;
- управління фотографією профілю;
- отримання сповіщень про нові записи.

Функціональні вимоги для адміністратора:

- управління обліковими записами лікарів та пацієнтів;
- управління розкладами лікарів та спеціальностями;
- перегляд загальної статистики записів;
- доступ до всіх даних системи.

Нефункціональні вимоги:

Безпека: автентифікація на основі JWT-токенів; розмежування прав доступу відповідно до ролі користувача; шифрування паролів; захист від підбору паролю; перевірка електронної пошти при реєстрації.

Надійність: автоматичне визначення пропущених прийомів; валідація вхідних даних; унікальні обмеження для запобігання дублювання записів.

Продуктивність: час відгуку інтерфейсу не перевищує 2 секунди; пагінація для списків лікарів та записів.

Зручність використання: зрозумілий інтерфейс; панель доступності для людей з особливими потребами; адаптивний дизайн для мобільних пристроїв та комп’ютерів.

Масштабованість: модульна архітектура, що дозволяє розширювати функціональність системи.

1.3 Моделювання предметної області

Щоб краще зрозуміти як влаштована система до початку її реалізації, можна описати основні процеси графічно. Для цього використовується мова моделювання UML (Unified Modeling Language) – стандартний інструмент у розробці програмного забезпечення, який дає змогу описати поведінку системи, взаємодію між учасниками та послідовність дій у вигляді діаграм [5].

У даному підрозділі було побудовано три типи діаграм. Діаграма прецедентів показує, хто взаємодіє з системою і які дії може виконувати. Діаграми послідовності та активності деталізують конкретні сценарії – як саме відбувається той чи інший процес, в якому порядку і що відбувається при різних умовах.

1.3.1 Діаграма прецедентів

Діаграма прецедентів описує предметну область через взаємодію між учасниками та діями, які вони виконують. Дана діаграма показує не конкретну реалізацію в системі, а склад її учасників та доступний їм функціонал. [5]. Саме через це вона будується однією з перших, коли треба розібратись у тому, які саме процеси має реалізовувати майбутня система.

У предметній області системи реєстрації пацієнтів лікарні виділено трьох основних акторів: Пацієнт, Лікар та Адміністратор.

На рис. 1.1 представлено діаграму прецедентів предметної області системи реєстрації пацієнтів лікарні. Вона описує ролі адміністратора, лікаря та пацієнта, а також ключові дії, які кожен з них виконує в межах досліджуваної предметної області.

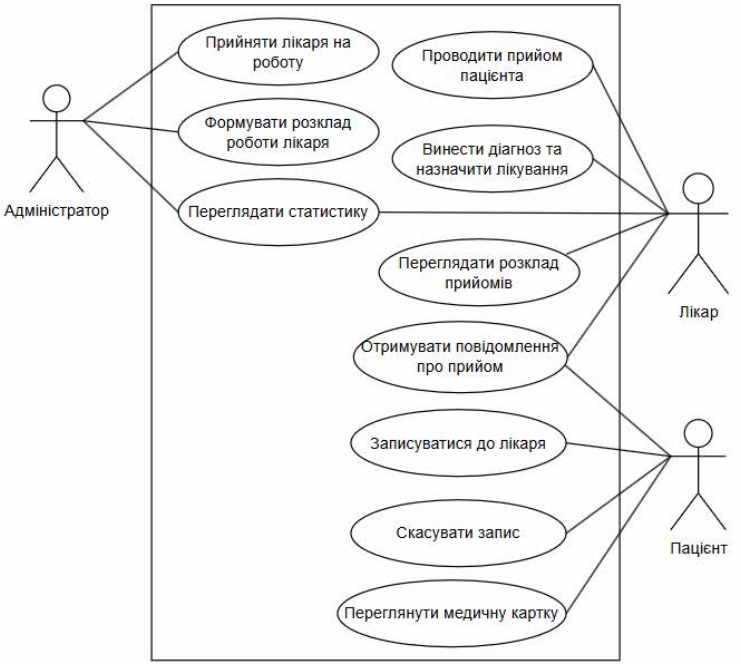

Рис. 1.1 Діаграма прецедентів предметної області

Адміністратор займається керуванням роботи лікарні. Він приймає лікарів на роботу, формує їхні розклади роботи та переглядає загальну статистику.

Лікар працює з пацієнтами: проводить прийоми, виносить діагнози та назначає лікування, за результатами кожного з них. Також, переглядає власний розклад, отримує повідомлення про заплановані прийоми та має доступ до статистики записів пацієнтів конкретно до нього.

Пацієнт звертається до лікарні з конкретною метою – записатись на прийом. За потреби він може скасувати запис або переглянути медичну картку після завершення прийому. Повідомлення про успішний запис на прийом або його скасування отримує так само, як і лікар.

1.3.2 Діаграма послідовності

Діаграма послідовності показує як учасники процесу взаємодіють між собою в часі. Головна відмінність від діаграми прецедентів у тому, що тут видно не просто перелік дій, а конкретний порядок їх виконання та обмін повідомленнями між учасниками [5].

Основні елементи діаграми: учасники, лінії життєвого циклу та повідомлення. Учасники розміщуються горизонтально у верхній частині

діаграми. Від кожного з них вниз іде пунктирна лінія – лінія життєвого циклу, яка показує що учасник існує протягом усього сценарію. Прямокутники на цих лініях позначають моменти активності – коли учасник безпосередньо виконує якусь дію або очікує відповіді. Суцільні стрілки між учасниками – це повідомлення або запити, пунктирні стрілки – відповіді на них [5].

Побудовано діаграму для сценарію “Запис пацієнта до лікаря”, як одного з ключових процесів предметної області. На діаграмі присутні чотири учасники: Пацієнт, Реєстратура, Розклад лікаря та Запис.

На рис. 1.2 наведено діаграму послідовності даного сценарію.

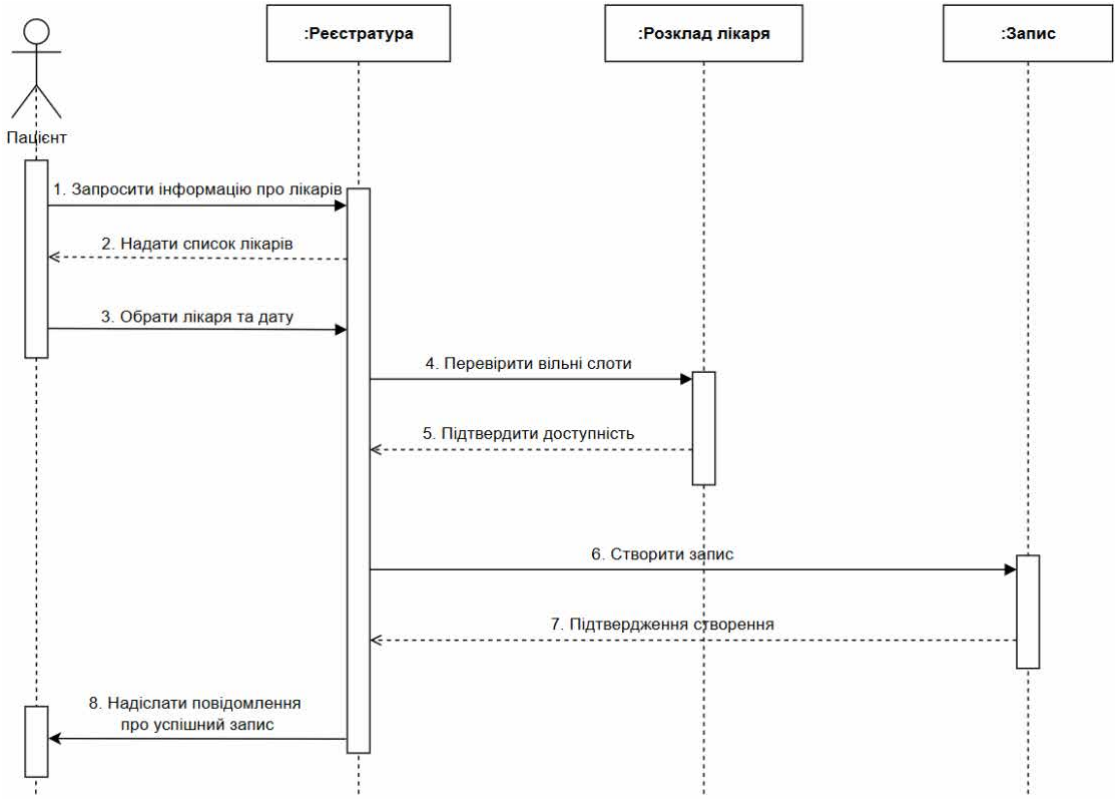

Рис. 1.2 Діаграма послідовності сценарію “Запис пацієнта до лікаря”

Сценарій починається з того, що пацієнт звертається до реєстратури із запитом про доступних лікарів. Реєстратура надає список лікарів, після чого пацієнт обирає конкретного лікаря та бажану дату. Далі реєстратура звертається до розкладу лікаря, щоб перевірити наявність вільного часу на обрану дату. Розклад підтверджує доступність і реєстратура переходить до створення запису. Об’єкт “Запис” зберігає дані про прийом та повертає підтвердження

успішного створення. Сценарій завершується тим, що реєстратура надсилає пацієнту повідомлення про успішний запис на прийом.

1.3.3 Діаграми активності

Діаграма активності показує послідовність дій у межах певного процесу та логіку переходів між ними. Вона добре підходить для опису сценаріїв де є розгалуження, тобто коли залежно від умови процес іде різними шляхами і виконуються різні дії [5]. На відміну від діаграми послідовності, тут акцент не на тому, хто з ким обмінюється повідомленнями, а на тому, як саме виконується процес від початку й до кінця.

Основні елементи діаграми: початкова точка, з якої стартує процес; дії у вигляді прямокутників зі скругленими кутами; стрілки, що показують напрям переходу між діями; ромби, що позначають умовні розгалуження; кінцева точка яка завершує процес. Щоб показати, хто саме виконує ту чи іншу дію, діаграма може ділитись на доріжки – кожна доріжка належить одному актору і містить лише його дії [5].

Побудовано дві діаграми активності для найбільш характерних сценаріїв для даної предметної області.

Перша діаграма охоплює сценарій “Запис пацієнта до лікаря”. Процес починається з того, що пацієнт обирає спеціальність та лікаря і переглядає його розклад. Далі він вказує бажану дату, після чого реєстратура перевіряє наявність вільного часу: якщо вільного часу немає, то пацієнт повертається до вибору іншої дати; якщо ж вільний час є, то пацієнт обирає зручний час та залишає причину звернення. Реєстратура оформлює запис і надсилає пацієнту повідомлення про успішну реєстрацію.

На рис. 1.3 наведено діаграму активності даного сценарію.

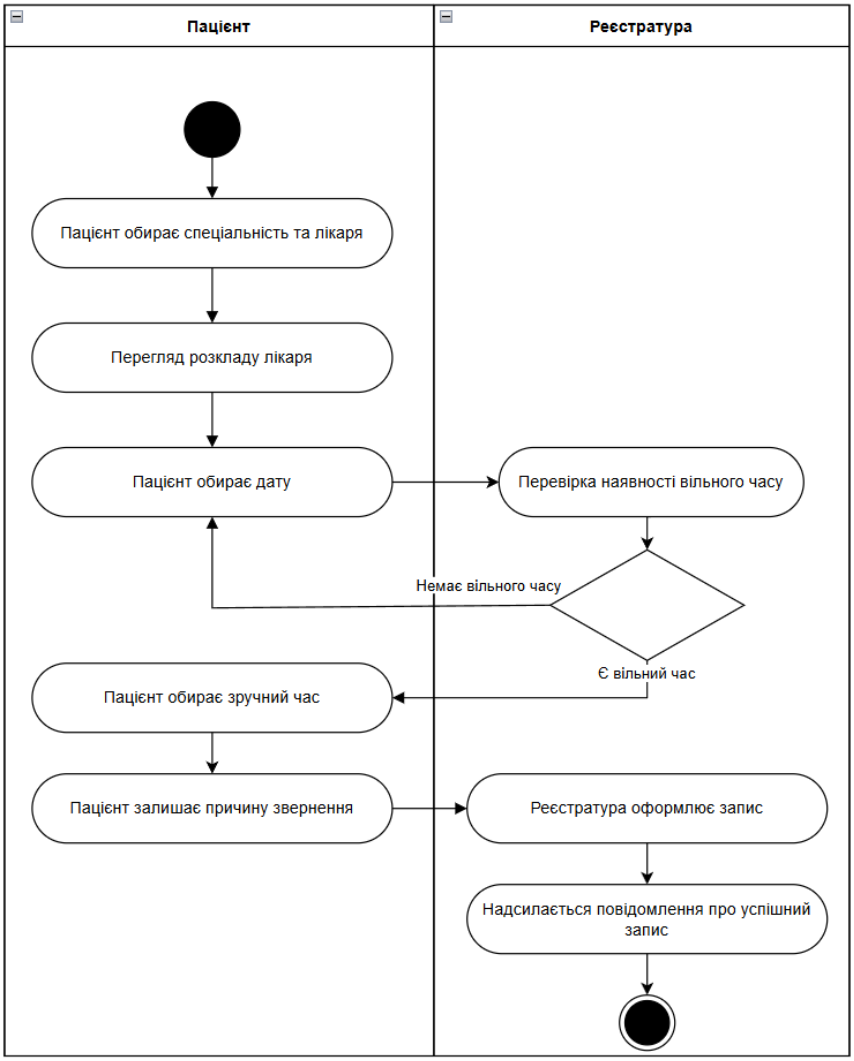

Рис. 1.3 Діаграма активності сценарію "Запис пацієнта до лікаря"

Друга діаграма описує сценарій "Скасування запису". Пацієнт переглядає список своїх записів та обирає той, який хоче скасувати. Реєстратура перевіряє статус запису: якщо він не має статусу "заплановано", скасування недоступне і процес завершується; якщо ж статус підходить, то пацієнт підтверджує скасування, після чого реєстратура змінює статус та надсилає повідомлення і пацієнту і лікарю.

На рис. 1.4 наведено діаграму активності даного сценарію.

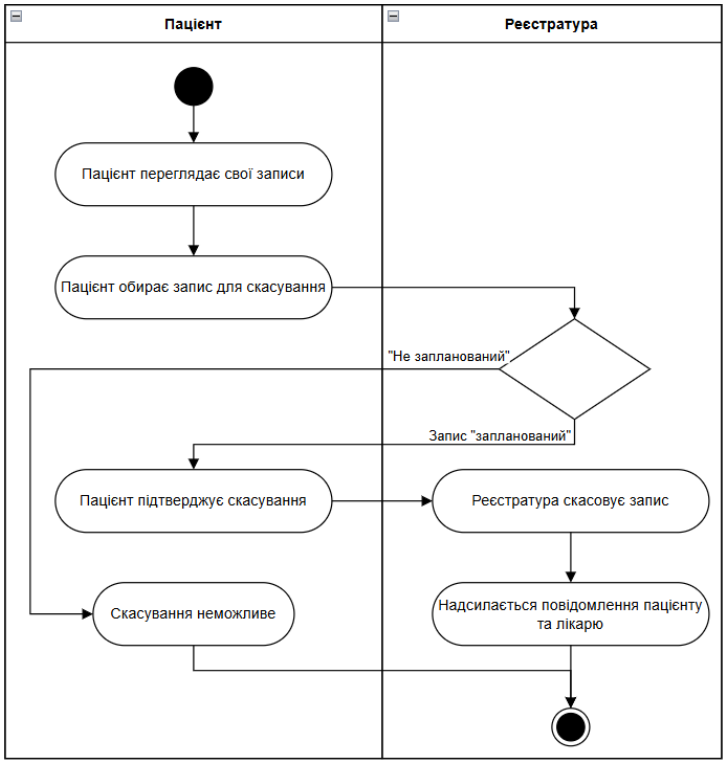

Рис. 1.4 Діаграма активності сценарію "Скасування запису"

1.3.4 Основні абстракції системи

Використання абстракцій в об'єктно-орієнтованому підході дозволяє виділити найважливіші характеристики об'єктів предметної області, відкинувши другорядні деталі. Кожна абстракція описує, які властивості має об'єкт і що він вміє робити [5].

На рис. 1.5 наведено абстракції основних об'єктів предметної області системи "Mediculus".

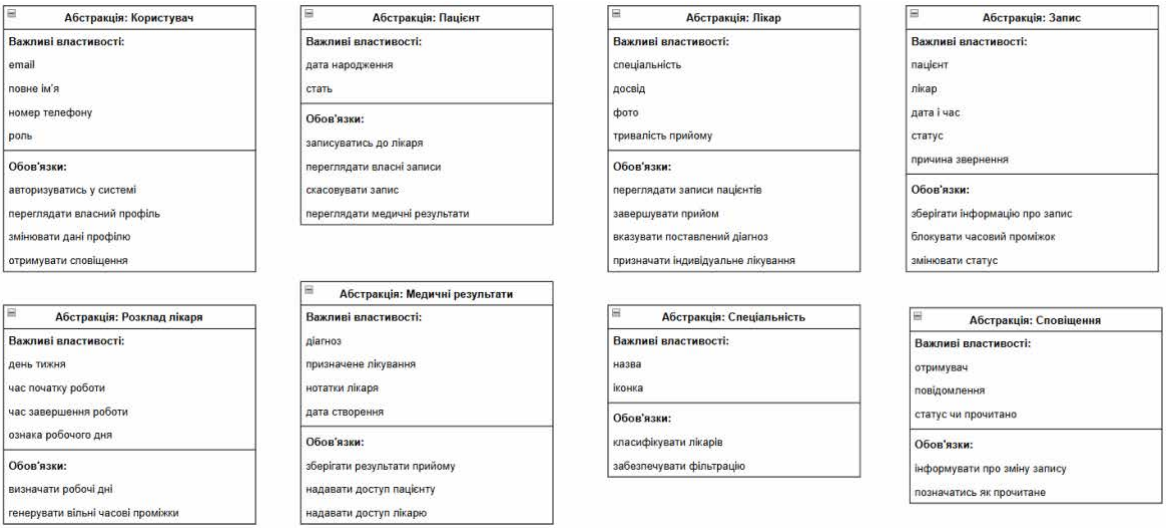

Рис. 1.5 Абстракції предметної області системи "Mediculus"

Центральною є абстракція ”Користувач”, вона є базовою для трьох ролей системи і зберігає спільні дані будь-якого учасника: електронну пошту, повне ім’я, телефон та роль. Абстракції “Пацієнт” і “Лікар” розширюють “Користувача” специфічними властивостями – пацієнт має дату народження та стать, лікар – спеціальність, досвід, фото та тривалість прийому.

Абстракція “Запис” є центральною з точки зору бізнес-логіки, вона зберігає запис конкретного пацієнта до конкретного лікаря на певну дату і час, зберігає поточний статус та причину звернення, і необхідна для блокування часового проміжку та зміни статусу. Абстракція “Розклад лікаря” визначає робочі дні та години лікаря і є основою для генерації вільних часових проміжків.

Абстракція “Медичні результати” зберігає підсумки завершеного прийому, а саме: діагноз, призначене лікування та нотатки лікаря, і надає доступ до цих даних відповідним учасникам. Абстракція “Спеціальність” це просто довідник, який ділить лікарів по категоріях і потрібен для того, щоб працював фільтр на сторінці пошуку. Абстракція “Сповіщення” відповідає за інформування користувачів про зміни статусу їх записів.

1.4 Огляд існуючих рішень

Перед тим як розробляти власну систему, варто розібратись що вже існує на ринку. Для цього проведено огляд чотирьох існуючих програмних рішень, які використовуються в Україні для онлайн-реєстрації пацієнтів: Helsi.me, Doc.ua, Medics.ua та Medikom.ua. Кожна система розглядається з точки зору функціональних можливостей, переваг та недоліків.

1) Helsi.me

Helsi.me – найбільш поширена електронна медична система в Україні, яка офіційно інтегрована з державною системою охорони здоров’я eHealth. Орієнтована як на державні так і на приватні медичні заклади по всій країні. За роки роботи зібрала велику базу лікарів та лікарень і стала звичним рішенням для більшості українців, які шукають лікаря онлайн [8].

На рис. 1.5 наведено головну сторінку Helsi.me.

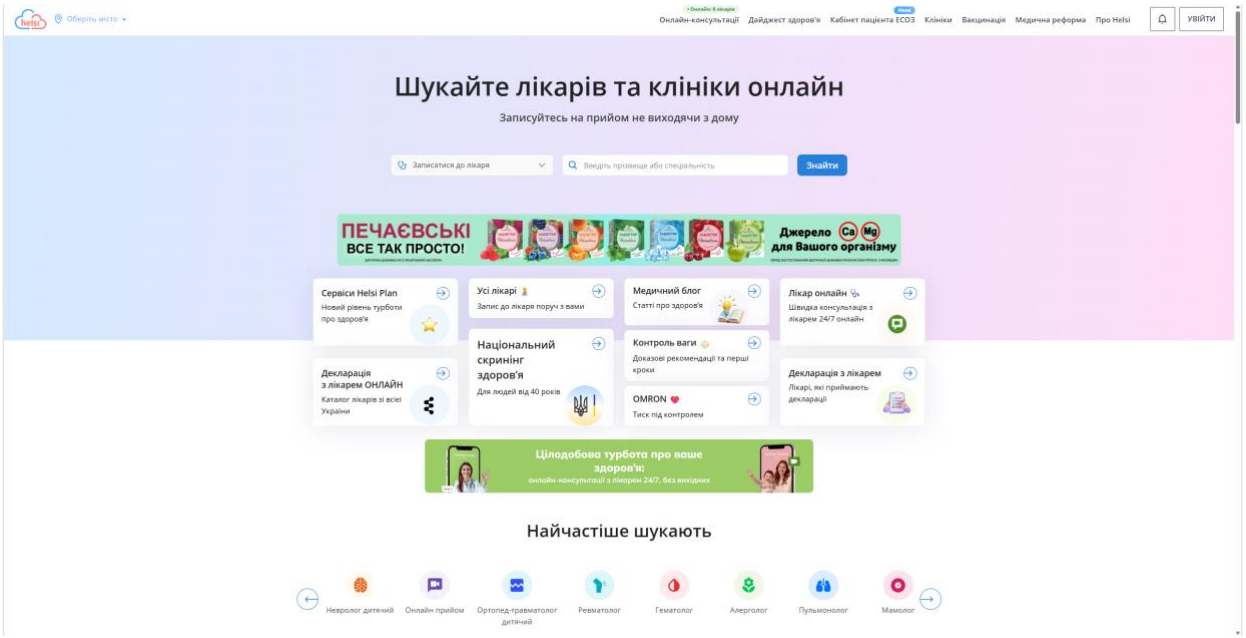

Рис. 1.5 Головна сторінка Helsi.me

Основні функціональні можливості:

- онлайн-запис на прийом до лікаря;
- перегляд електронної медичної картки пацієнта;
- отримання електронного рецепта;
- підтримка електронних декларацій із сімейним лікарем;
- доступ до результатів аналізів та історії прийомів.

Переваги:

- офіційна інтеграція з eHealth;
- велика база лікарів і медзакладів по всій Україні;
- мобільний додаток для пацієнтів.

Недоліки:

- деякі функції обмежені залежно від клініки та її підключення до eHealth;
- авторизація лише за номером телефону з підтвердженням через SMS-код – неможливо увійти через електронну пошту.

2) Doc.ua

Doc.ua є зручною платформою для пошуку лікаря та запису на прийом, орієнтованою переважно на приватні клініки. Підходить для користувачів, які шукають швидкий запис, проте не призначена для зберігання історії лікувань [9].

На рис. 1.6 наведено головну сторінку Doc.ua.

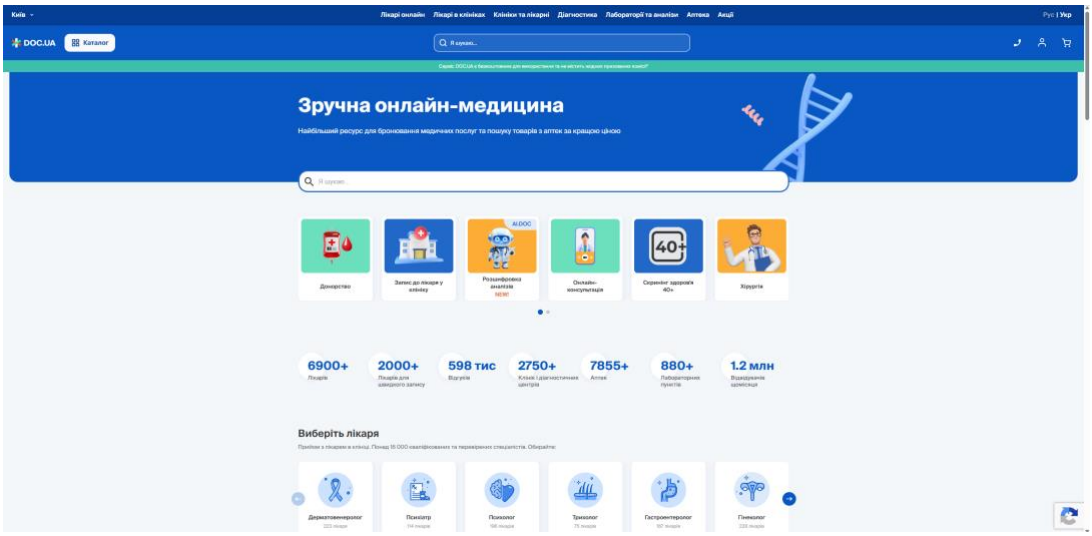

Рис. 1.6 Головна сторінка Doc.ua

Основні функціональні можливості:

- онлайн-запис на прийом до лікаря у приватних та державних клініках;
- пошук лікарів за спеціальністю, рейтингом, стажем та відгуками;
- підтримка онлайн-консультацій з лікарем;
- система відгуків і рейтингів для лікарів;
- SMS-нагадування про запис.

Переваги:

- одразу видно ціну послуги;
- наявність ботів в різних месенджерах для запису;
- велика база лікарів по всій Україні;
- безкоштовне користування для пацієнтів;

- вбудована система пошуку аптек з найнижчими цінами.

Недоліки:

- відсутність електронної медичної картки;
- неможливість запису на конкретний час – лише перша або друга половина дня.

3) Medics.ua

Medics.ua є комплексною електронною медичною системою спрямованою як на лікарів так і на пацієнтів. Основна перевага – офіційна інтеграція з державною системою eHealth [10].

На рис. 1.7 наведено головну сторінку Doc.ua.

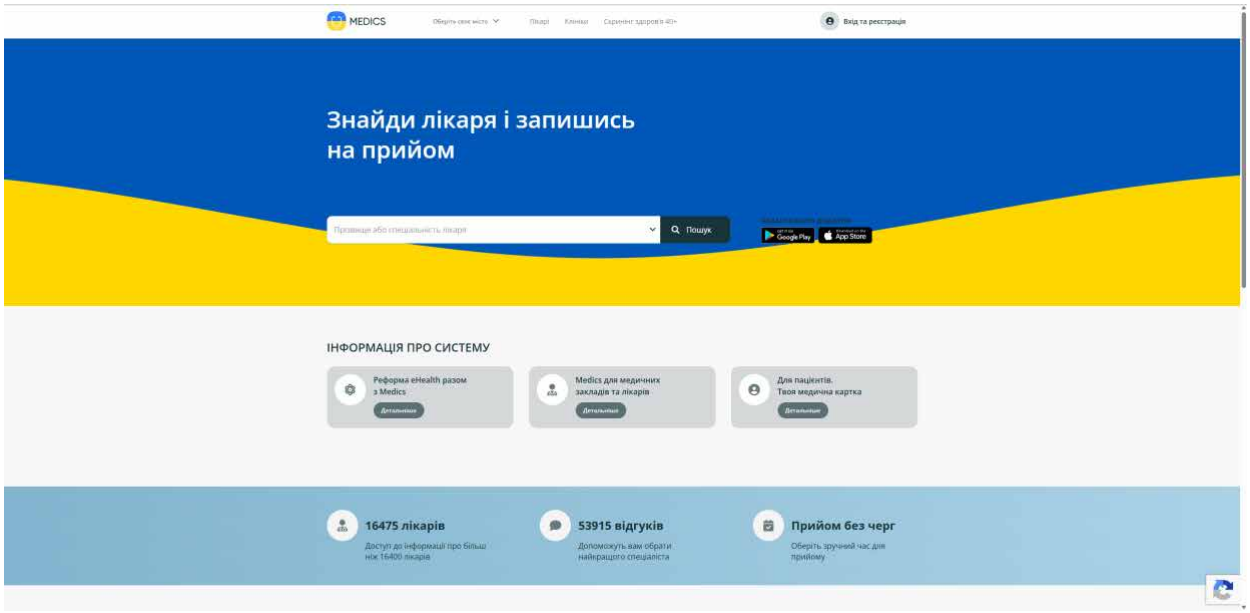

Рис. 1.7 Головна сторінка Medics.ua

Основні функціональні можливості:

- перегляд електронної медичної картки пацієнта;
- онлайн-запис до лікаря;
- отримання електронного рецепта;
- доступ до історії відвідувань і результатів аналізів;
- розширений пошук лікарів за типом лікарні та клінікою.

Переваги:

- інтеграція з eHealth;
- підтримка лікарів і клінік будь-якого типу;
- детальний каталог клінік та місць надання послуг з розкладом роботи та адресами;
- реєстрація як за номером телефону так і за електронною поштою.

Недоліки:

- частина функцій доступна лише для медичних закладів підключених до eHealth;
- не всі приватні клініки представлені в системі.

4) Medikom.ua

Medikom.ua орієнтована на одну мережу клінік. Підходить для приватних лікарень але не може масштабуватися для загального використання [11].

На рис. 1.8 наведено головну сторінку Doc.ua.

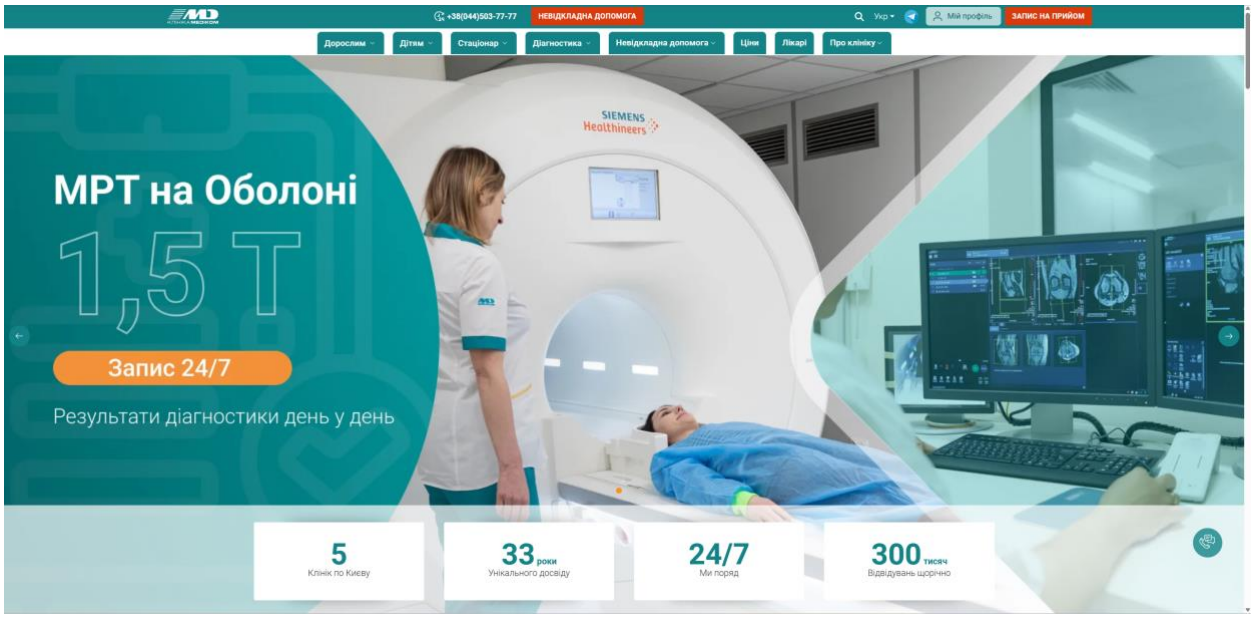

Рис. 1.8 Головна сторінка Medikom.ua

Основні функціональні можливості:

- вибір лікаря, послуги та філіалу;
- особистий кабінет пацієнта;
- оплата послуг онлайн;
- отримання консультацій онлайн.

Переваги:

- інформація про акції, знижки та новини клініки;
- онлайн та офлайн-консультації;
- наявність прайсу на всі послуги.

Недоліки:

- система працює лише для однієї мережі клінік;
- не можливо записатися на прийом без консультації з менеджером;
- немає можливості вибору конкретного часу запису.

Порівняльний аналіз систем наведено в таблиці 1.1.

Таблиця 1.1 Порівняльний аналіз існуючих систем

Критерій	**Helsi.me**	**Doc.ua**	**Medics.ua**	**Medikom.ua**	**Mediculus**
Онлайн-запис	+	+	+	-	+
Вибір конкретного часу	+	-	+	-	+
Електронна медкартка	+	-	+	-	+
Верифікація по email або по номеру телефону	+	+	+	+	+
Відновлення паролю	+	+	+	+	+
Автоскасування прийомів	-	-	-	-	+
Панель доступності	-	-	-	-	+

Після аналізу програм-аналогів стало зрозуміло, що “Mediculus” виграє у своїх конкурентів за кількома основними критеріями. По-перше, система єдина з розглянутих має автоматичне визначення пропущених прийомів та зміну їх статусу. По-друге, реалізована панель доступності для людей з особливими

потребами – це функція, яка відсутня в усіх аналогах. По-третє, на відміну від Helsi.me та Medics.ua, система не прив’язана до державної інфраструктури eHealth, що дає більшу гнучкість при розгортанні в приватних медичних закладах. Також авторизація реалізована через електронну пошту, а не через номер телефону, що для багатьох людей є перевагою.

1.5 Постановка завдання

Метою дипломної роботи є створення веб-застосунку, який автоматизує процес реєстрації пацієнтів лікарні. Основна проблема, яку вирішує система це усунення необхідності особистого звернення до реєстратури для запису.

“Mediculus” належить до класу веб-орієнтованих інформаційних систем реєстрації пацієнтів. Система працює через браузер без встановлення додаткового програмного забезпечення і доступна з будь-якого пристрою підключеного до інтернету.

Система повинна обробляти та зберігати такі дані:

- особисті дані користувачів (ім’я, прізвище, по батькові, електронна пошта, номер телефону, роль);
- дані лікарів (спеціальність, досвід, опис, фотографія, розклад роботи по днях тижня);
- записи на прийом (пацієнт, лікар, дата та час, статус, причина звернення);
- медичні результати (діагноз, призначене лікування, нотатки лікаря);
- внутрішні сповіщення для користувачів.

Повинні бути автоматизовані такі операції:

- генерація вільних місць для запису, орієнтуючись на графік лікаря;
- перевірка унікальності запису, щоб унеможливити подвійний запис на один час;
- автоматична зміна статусу прийому на “пропущено”, якщо пацієнт не з’явився на прийом;

- відправка email-повідомлень через зовнішній сервіс Brevo для верифікації акаунту та відновлення паролю;
- формування внутрішніх сповіщень при зміні статусу запису.

В результаті роботи система повинна надавати наступне:

- розклад лікарів з переліком вільного часу на 14 днів вперед;
- особистий кабінет пацієнта з історією записів та результатами завершених прийомів;
- кабінет лікаря зі списком пацієнтів на поточний день та статистикою прийомів, а саме: кількість запланованих, завершених та скасованих;
- адміністративна панель із загальною статистикою по всіх записах лікарні з можливістю переглядати її по днях, тижнях та місяцях.

Система повинна реалізовувати такий функціонал:

1) Реєстрація та авторизація користувачів трьох ролей: пацієнт, лікар, адміністратор. Реєстрація доступна лише для пацієнтів. Облікові записи лікарів та адміністраторів створюються адміністратором через панель управління. Авторизація здійснюється за допомогою електронної пошти та пароля.

2) Верифікація електронної пошти при реєстрації пацієнта. Після реєстрації на пошту надсилається 6-значний код підтвердження, який дійсний протягом 5 хвилин. Відновлення забутого паролю через посилання на електронну пошту, дійсне 1 годину.

3) Пошук та перегляд лікарів. Публічна сторінка з переліком лікарів з можливістю фільтрації за спеціальністю та пошуком за ПІБ. Детальна сторінка профілю кожного лікаря з описом, досвідом та розкладом роботи.

4) Онлайн-запис на прийом. Автоматична генерація вільного часу, доступного для запису, на основі індивідуального розкладу лікаря на 14 днів вперед. Слоти формуються з кроком, що дорівнює тривалості прийому. Зайняті та пропущені слоти автоматично видаляються із доступних.

5) Управління записами. Пацієнт може переглядати свої записи з фільтрацією за статусом та часовим діапазоном, скасовувати заплановані

прийоми. Лікар бачить список своїх пацієнтів, може завершувати прийоми та вносити результати.

6) Медичні результати. Після завершення прийому лікар вносить діагноз та призначення. Результати зберігаються в електронному вигляді та доступні пацієнту в особистому кабінеті.

7) Автоматичне визначення пропущених прийомів. Система автоматично змінює статус запису з “заплановано” на “пропущено”.

8) Система сповіщень. Внутрішні сповіщення для пацієнтів та лікарів про підтвердження записів, зміни статусів та пропущені прийоми. Email-сповіщення для верифікації акаунту та відновлення паролю.

9) Адміністративна панель з детальною статистикою записів та інструментами для управління системою.

10) Панель доступності для людей з особливими потребами: регулювання розміру шрифту, режим високої контрастності, шрифт для людей з дислексією.

2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування є одним з ключових етапів розробки програмного забезпечення. На цьому етапі визначається структура даних, архітектура системи та взаємодія між її компонентами. Якість прийнятих проєктних рішень дуже сильно впливає на те, наскільки система буде зручною у розробці, підтримці та майбутньому розширенні [7].

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування інформаційного забезпечення починається з побудови логічної моделі даних. Вона описує предметну область через сутності, їхні атрибути та зв’язки між ними, а також без прив’язки до конкретної СУБД чи деталей реалізації. Найпоширенішим інструментом для побудови логічної моделі є ER-діаграма (Entity-Relationship Diagram) – діаграма сутність-зв’язок [6].

В ER-діаграмі кожна сутність відповідає певному об’єкту предметної області і представлена у вигляді прямокутника з іменем та переліком атрибутів. Атрибути поділяються на ключові: ті, що однозначно ідентифікують екземпляр сутності, та неключові – ті, що описують його властивості. Зв’язки між сутностями показують як вони пов’язані між собою і яка кількість екземплярів однієї сутності може відповідати екземплярам іншої [6].

Для побудови логічної моделі даних системи “Mediculus” використано програму ERwin Data Modeler. Модель містить сутності, які охоплюють всі основні об’єкти предметної області.

На рис. 2.1 наведено логічну модель даних системи “Mediculus”.

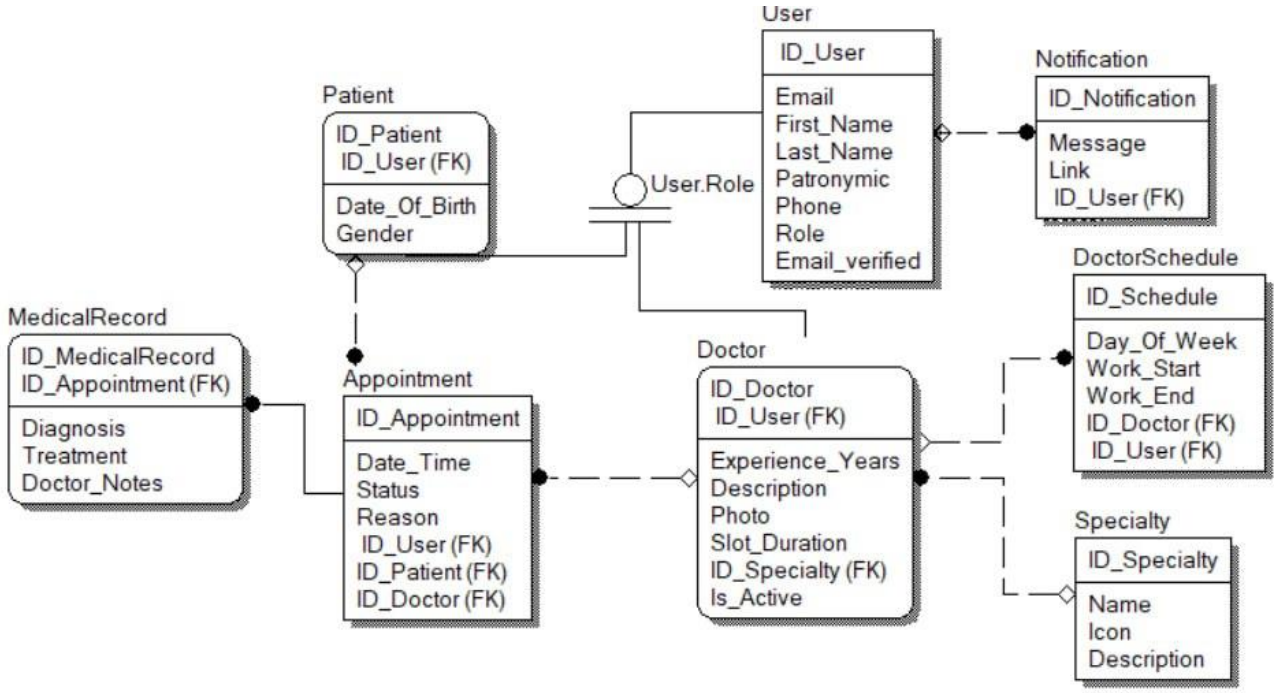

Рис. 2.1 Логічна модель даних системи "Mediculus"

Центральною сутністю моделі є "User" – користувач системи. Вона містить загальні дані про будь-якого користувача: електронну пошту як унікальний ідентифікатор для входу, ім'я, прізвище, по батькові, номер телефону, роль у системі та ознаку верифікації електронної пошти. Роль визначає до якого підтипу належить користувач – пацієнт, лікар або адміністратор. User є супертипом у категоріальному зв'язку з дискримінатором User.Role і виступає основою для двох підтипів: Patient та Doctor.

Сутність "Patient" розширює "User" даними специфічними для пацієнта – датою народження та статтю. Зв'язок між User і Patient є категоріальним – Patient є підтипом супертипу User з дискримінатором Role. Первинний ключ підтипу ID_Patient успадковується від супертипу, що означає що запис пацієнта не може існувати без відповідного користувача.

Сутність "Doctor" також є підтипом "User" у тому ж категоріальному зв'язку. Розширює User такими даними лікаря, як: роки досвіду, опис, фотографія, тривалість одного прийому в хвилинах та ознака активності. Сутність "Doctor" завжди відповідає конкретній сутності "User" з роллю "лікар".

Сутність ”Specialty” є довідником медичних спеціальностей. Містить назву, іконку та опис. Пов’язана з Doctor неідентифікуючим зв’язком – один лікар має одну спеціальність, але одна спеціальність може належати багатьом лікарям. Зовнішній ключ ID_Specialty переходить до неключових атрибутів Doctor.

Сутність ”DoctorSchedule” зберігає розклад роботи лікаря. Для кожного дня тижня окремо вказується час початку та кінця роботи і ознака чи лікар приймає в цей день. Пов’язана з Doctor неідентифікуючим зв’язком – зовнішній ключ ID_Doctor переходить до неключових атрибутів DoctorSchedule.

Центральною з точки зору бізнес-процесу є сутність ”Appointment” – запис на прийом. Вона фіксує факт запису конкретного пацієнта до конкретного лікаря на певну дату і час, зберігає поточний статус запису та причину звернення. Appointment пов’язана з Patient і Doctor неідентифікуючими зв’язками – пацієнт і лікар можуть мати багато записів на прийом, зовнішні ключі ID_Patient та ID_Doctor переходять до неключових атрибутів Appointment.

Сутність ”MedicalRecord” зберігає результати завершеного прийому: діагноз, призначене лікування та нотатки лікаря. Пов’язана з Appointment ідентифікуючим зв’язком – первинний ключ ID_Appointment переходить до первинного ключа MedicalRecord, що означає, що медична картка не може існувати без відповідного запису на прийом і кожен запис може мати лише одну картку.

Сутність ”Notification” зберігає внутрішні сповіщення для користувачів. Містить текст повідомлення та посилання на відповідний об’єкт. Пов’язана з User неідентифікуючим зв’язком – один користувач може отримувати багато сповіщень, зовнішній ключ ID_User переходить до неключових атрибутів Notification.

2.1.1 Перевірка відповідності нормальним формам

Щоб забезпечити ефективне зберігання даних та уникнути їх надлишковості, логічна модель приведена до нормалізованого вигляду.

Нормалізація є процесом організації атрибутів та зв’язків між сутностями таким чином, щоб мінімізувати дублювання даних і забезпечити їх цілісність [21]. У процесі проєктування модель даних системи “Mediculus” було перевірено на відповідність першій, другій та третій нормальним формам.

Перша нормальна форма виконується, оскільки кожен атрибут кожної сутності містить лише атомарне значення [21]. Наприклад прізвище, ім’я та по батькові зберігаються як окремі поля, а не одним рядком. Розклад лікаря зберігається окремим записом для кожного дня тижня, а не переліком днів в одному полі.

Друга нормальна форма виконується, оскільки всі неключові атрибути повністю залежать від первинного ключа своєї сутності [21]. Наприклад, атрибути “діагноз” та “призначення” належать сутності MedicalRecord і залежать виключно від її ключа, а не від ключів Doctor чи Appointment.

Третя нормальна форма виконується, оскільки між неключовими атрибутами відсутні транзитивні залежності [21]. Спеціальність лікаря винесена в окрему сутність Specialty і пов’язана з Doctor через зовнішній ключ, замість того, щоб зберігати назву спеціальності безпосередньо в Doctor. Дані користувача не дублюються в Patient і Doctor, тобто обидві сутності звертаються до User через зовнішній ключ.

2.2 Діаграма класів

Діаграма класів описує структуру програмної системи через класи, їхні атрибути, методи та зв’язки між ними [5]. На відміну від діаграм предметної області, які описують реальний світ, діаграма класів вже відображає проєктне рішення – як саме система буде організована зсередини. Кожен клас на діаграмі має такі секції: назву, атрибути з типами та методи з позначками видимості: + публічний, - приватний.

На рис. 2.2 наведено діаграму класів системи “Mediculus”.

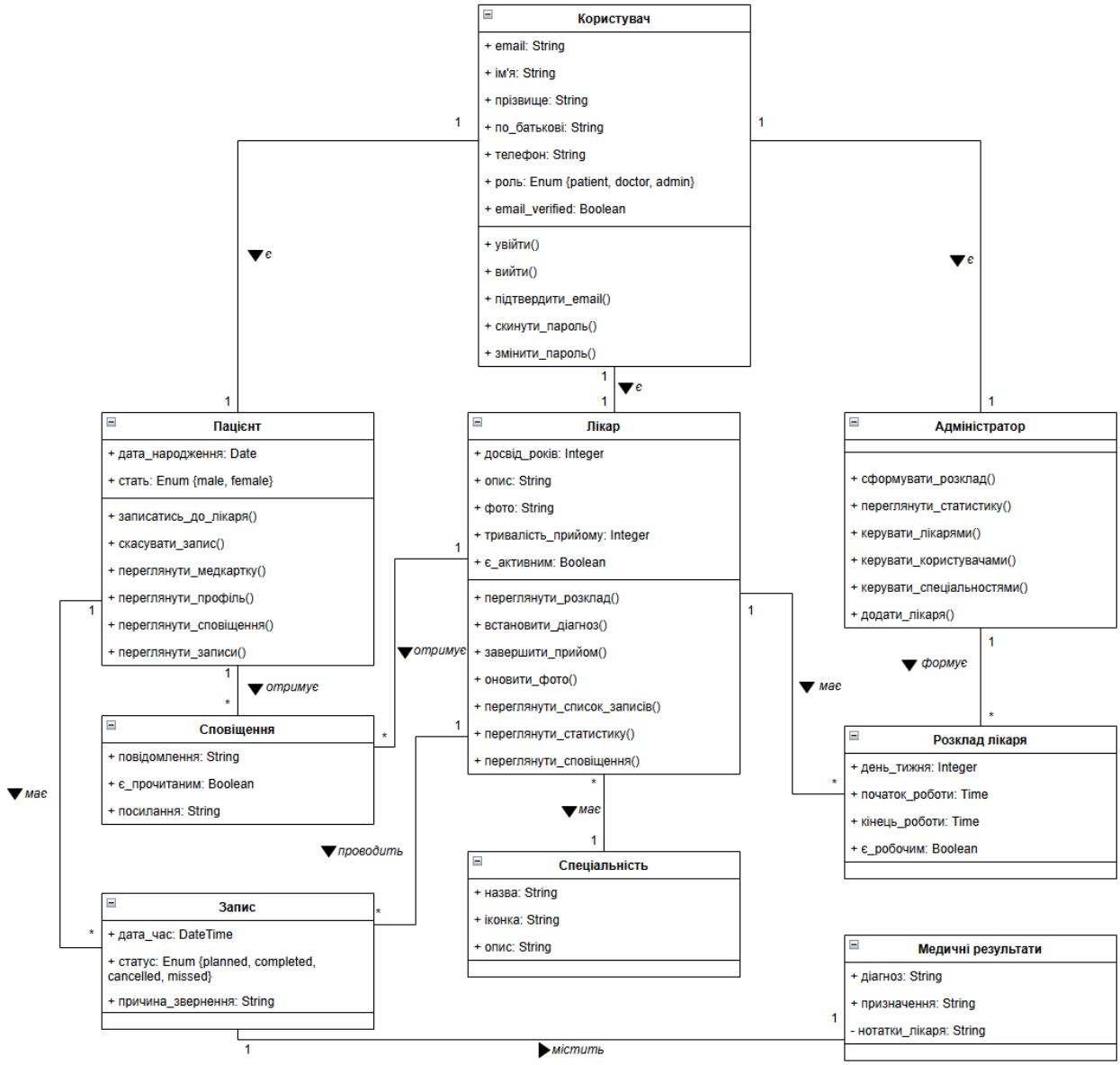

Рис. 2.2 Діаграма класів системи "Mediculus"

Центральним є клас "Користувач" – він зберігає загальні дані будь-якого учасника системи: електронну пошту для входу в систему, ім'я, прізвище, по батькові, телефон, роль та ознаку підтвердження пошти. Клас реалізує базові операції автентифікації: вхід і вихід із системи, підтвердження електронної пошти та керування паролем. Від класу "Користувач" встановлено асоціації "є" до трьох підтипів – "Пацієнт", "Лікар" та "Адміністратор" – з кратністю 1:1 з кожного боку.

Клас "Пацієнт" розширює "Користувача" особистими медичними даними, а саме: датою народження та статтю. Пацієнт виконує основні дії

пов’язані з відвідуванням лікаря: записується на прийом, скасовує записи, переглядає результати прийомів та власний профіль. Пацієнт пов’язаний з класом ”Сповіщення” асоціацією “отримує” з кратністю 1:* та з класом ”Запис” асоціацією “має” з кратністю 1:*.

Клас ”Лікар” містить такі атрибути: роки досвіду, опис, фото та триваліть одного прийому. Лікар виконує такі ключові функції системи: проводить прийоми, встановлює діагнози, заповнює медичні результати та керує власним розкладом і профілем. Лікар пов’язаний з класом ”Спеціальність” асоціацією “має” з кратністю *:1 та з класом ”Запис” асоціацією “проводить” з кратністю 1:*.

Клас ”Адміністратор” не має власних атрибутів, він необхідний для управління системою: він додає лікарів, формує їхні розклади, керує спеціальностями та переглядає загальну статистику. Адміністратор пов’язаний з класом ”Розклад” лікаря асоціацією “формує” з кратністю 1:*. Важливим є те, що в базі даних ”Адміністратор” не має окремої таблиці, це звичайний користувач з роллю “admin”. На діаграмі його виділено окремо, щоб показати важливі функції управління та зв’язки з іншими класами.

Клас ”Спеціальність” є довідником медичних спеціальностей і містить назву, іконку та опис. Одна спеціальність може об’єднувати багатьох лікарів.

Клас ”Розклад лікаря” описує робочий графік лікаря по днях тижня – для кожного дня зберігається час початку та кінця роботи і ознака чи лікар приймає в цей день.

Клас ”Запис” є центральним з точки зору бізнес-процесу, він зберігає запис конкретного пацієнта до конкретного лікаря на певну дату і час, зберігає поточний статус та причину звернення. З класом ”Медичні результати” встановлено асоціацію “містить” з кратністю 1:1, кожен завершений запис має рівно одні медичні результати.

Клас ”Медичні результати” зберігає підсумки прийому: діагноз та призначення як публічні атрибути, та нотатки лікаря як приватний атрибут, який доступний лише в межах класу.

Клас ”Сповіщення” містить текст повідомлення, посилання на відповідний об’єкт системи та ознаку чи повідомлення було прочитане.

2.2.1 Кооперація “Запис пацієнта до лікаря”

Кооперація описує взаємодію між класами під час виконання сценарію запису пацієнта на прийом до лікаря. Кооперація містить чотири класи: Пацієнт, Лікар, Запис та Розклад лікаря.

На рис. 2.3 наведено діаграму даної кооперації.

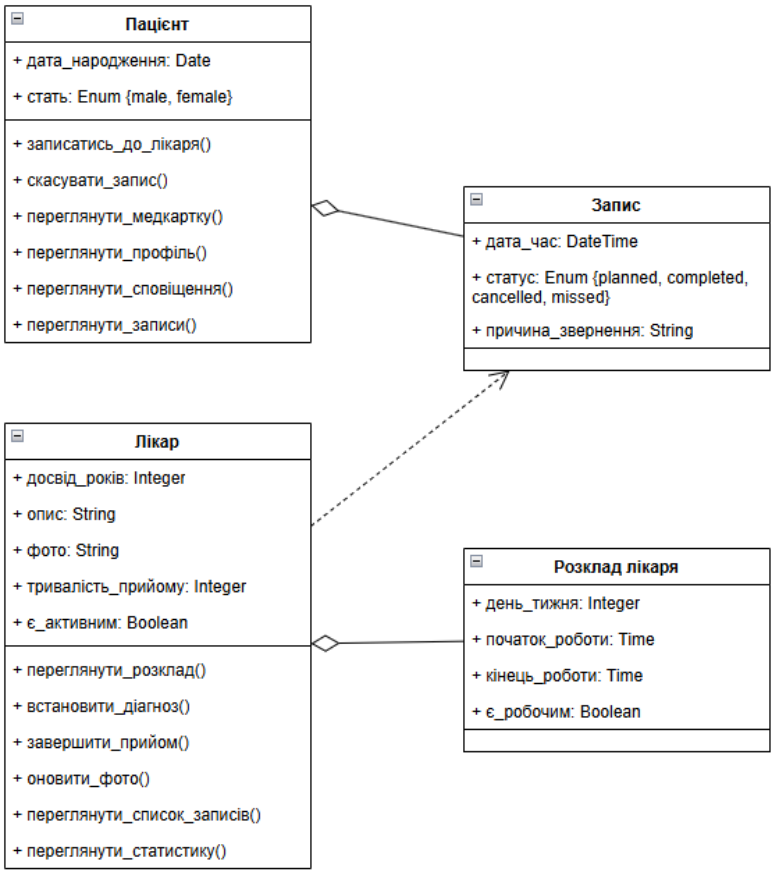

Рис. 2.3 Діаграма кооперації “Запис пацієнта до лікаря”

Зв’язок між класами ”Пацієнт” та ”Запис” є агрегацією, тобто запис є частиною історії пацієнта, проте може існувати окремо після свого створення. Пацієнт ініціює процес через метод “записатись_до_лікаря()” і в подальшому може скасувати запис або переглянути його результати.

Зв’язок між класами ”Лікар” та ”Запис” є залежністю, тобто лікар використовує запис у процесі роботи, переглядає список своїх пацієнтів та змінює статус запису, але при цьому не є його власником.

Зв’язок між класами ”Лікар” та ”Розклад лікаря” є агрегацією, тобто розклад є частиною лікаря як сутності системи. Саме на основі розкладу формуються вільні часові проміжки для запису: система перевіряє для якого дня тижня є запит, зчитує час початку та кінця роботи і генерує доступні проміжки з кроком що дорівнює тривалості прийому лікаря.

2.2.2 Кооперація “Завершення прийому лікарем”

Кооперація описує взаємодію між класами під час завершення лікарем прийому пацієнта та внесення його результатів. Кооперація має три класи: Лікар, Запис та Медичні результати.

На рис. 2.4 наведено діаграму даної кооперації.

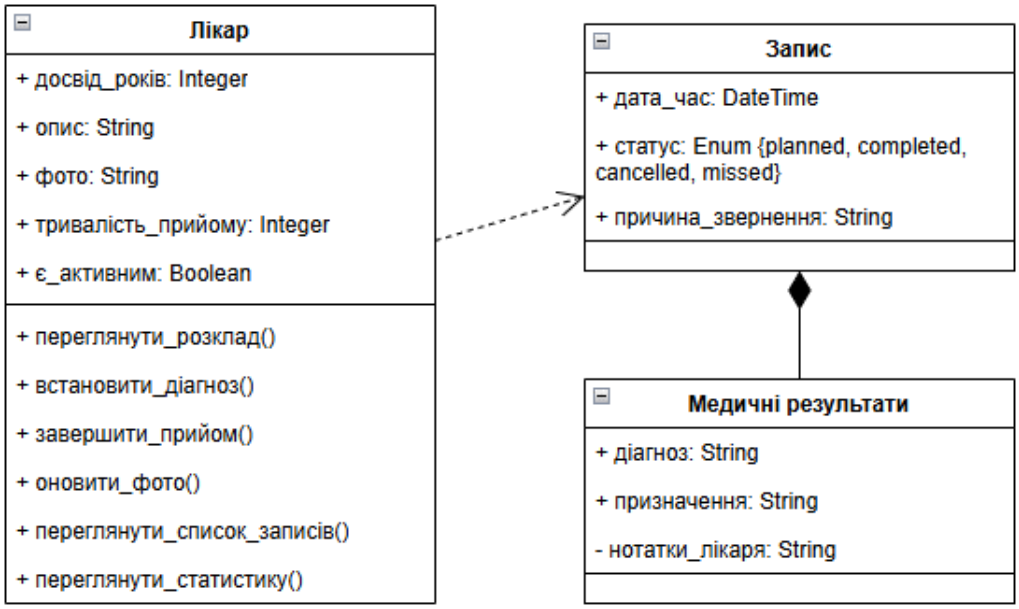

Рис. 2.4 Кооперація “Завершення прийому лікарем”

Зв’язок між класами “Лікар” та “Запис” є залежністю, тобто лікар використовує запис для завершення прийому та встановлення діагнозу, але не є його власником. Через метод “завершити_прийом()” лікар змінює статус запису на “completed” та ініціює створення медичних результатів.

Зв’язок між класами “Запис” та “Медичні результати” є композицією, тобто медичні результати не можуть існувати без відповідного запису на прийом. Якщо запис видаляється, то медичні результати також втрачають сенс.

Після завершення прийому лікар вносить діагноз, призначення та за потреби приватні нотатки, які доступні лише в межах класу.

2.2.3 Кооперація “Формування розкладу роботи лікаря”

Кооперація описує взаємодію між класами під час формування адміністратором розкладу роботи лікаря. Кооперація має три класи: Адміністратор, Лікар та Розклад лікаря.

На рис. 2.5 наведено діаграму даної кооперації.

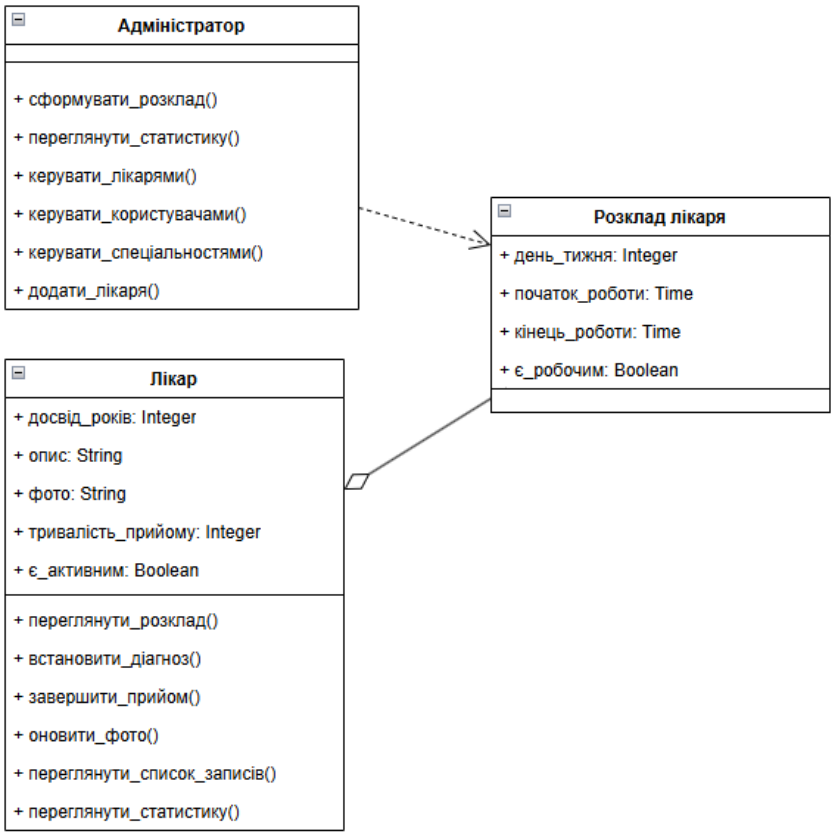

Рис. 2.5 Кооперація “Формування розкладу роботи лікаря”

Зв’язок між класами “Адміністратор” та “Розклад лікаря” є залежністю, тобто адміністратор формує та редагує розклад роботи лікарів через метод “сформувати_розклад()”, але при цьому не є власником цього розкладу. Для кожного дня тижня окремо вказується час початку та кінця роботи і ознака чи лікар приймає в цей день.

Зв’язок між класами “Лікар” та “Розклад лікаря” є агрегацією, тобто розклад є частиною лікаря як сутності системи, оскільки визначає його робочі

години. Лікар може переглядати власний розклад через метод переглянути_розклад(), однак змінювати його може лише адміністратор.

2.3 Діаграма пакетів

Діаграма пакетів показує як система організована на рівні логічних модулів. Пакети групують пов'язані між собою елементи системи і відображають залежності між цими групами [5]. Знак "+" перед назвою пакету означає, що він публічний і доступний ззовні, знак "-" означає, що пакет є внутрішнім і використовується лише в межах батьківського пакету. Пунктирні стрілки між пакетами позначають залежності, де один пакет використовує або потребує інший для своєї роботи.

На рис. 2.6 наведено діаграму пакетів системи "Mediculus".

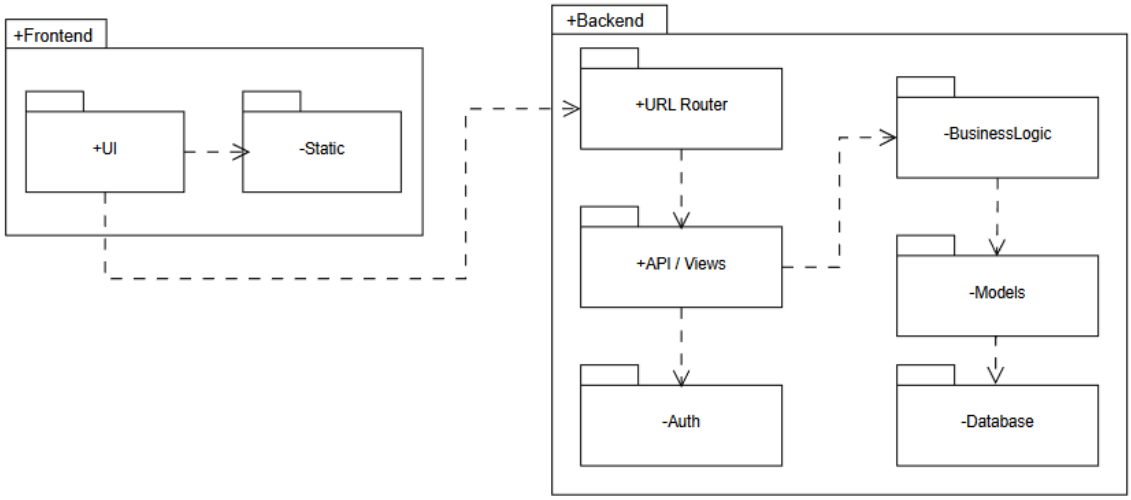

Рис. 2.6 Діаграма пакетів системи "Mediculus"

Діаграма демонструє загальну архітектуру системи розділену на дві основні частини – Frontend (клієнтська частина) та Backend (серверна частина).

Frontend (клієнтська частина)

Підпакет *+UI* є інтерфейсом користувача через який пацієнти, лікарі та адміністратор взаємодіють із системою. Він відображає HTML-сторінки та обробляє дії користувача. При будь-якій дії користувача формується HTTP-запит до серверної частини.

Підпакет -*Static* містить статичні ресурси системи: CSS-стилі на основі Bootstrap 5 для оформлення інтерфейсу, JavaScript-файли з логікою fetch-запитів до API та зображення. Використовується інтерфейсом для коректного відображення сторінок. Статичні файли завантажуються та виконуються на стороні клієнта, а їх доставка здійснюється засобами бібліотеки WhiteNoise на рівні Backend.

Backend (серверна частина)

Підпакет *+URL Router* є маршрутизатором на основі urls.py файлів Django. Він приймає HTTP-запити від клієнтської частини та направляє їх до відповідного обробника залежно від URL-адреси запиту.

Підпакет *+API / Views* є точкою входу до серверної частини та контролером системи. Він приймає маршрутизовані запити від URL Router та координує їх обробку: ініціює перевірку авторизації через підпакет -Auth і після успішної перевірки передає запит до -BusinessLogic.

Підпакет -*Auth* відповідає за авторизацію та розмежування прав доступу. Містить JWT Middleware для перевірки валідності токена в заголовку запиту та кастомні класи прав доступу: IsPatient, IsDoctor, IsAdmin, які перевіряють роль користувача.

Підпакет -*BusinessLogic* реалізує основну бізнес-логіку системи. Отримує вже авторизовані запити та виконує необхідні операції: обробку записів на прийом, генерацію вільних часових проміжків, автоматичну зміну статусів прострочених записів, створення сповіщень та формування медичних результатів. Включає серіалізатори для валідації вхідних даних та перетворення об’єктів моделей у формат JSON.

Підпакет -*Models* містить моделі даних системи: User, Patient, Doctor, Specialty, DoctorSchedule, Appointment, MedicalRecord, Notification. Забезпечує взаємодію між бізнес-логікою та базою даних через Django ORM.

Підпакет -*Database* є базою даних PostgreSQL яка зберігає всі дані системи і є кінцевою ланкою в ланцюжку обробки запиту.

Залежності між пакетами

- UI використовує Static для відображення стилів, скриптів та зображень.
- UI формує HTTP-запити та передає їх до URL Router.
- URL Router аналізує URL адресу та направляє запит до API / Views.
- API / Views передає запит до Auth для перевірки JWT токена та прав
- доступу.
- API / Views після успішної авторизації передає запит до BusinessLogic для
- обробки.
- BusinessLogic звертається до Models для читання та збереження даних.
- Models виконують запити до Database через Django ORM.

2.4 Діаграма компонентів

Діаграма компонентів показує фізичну організацію системи: конкретні програмні компоненти, бібліотеки, зовнішні сервіси та залежності між ними [5]. На відміну від діаграми пакетів, яка описує логічну структуру коду, діаграма компонентів відображає що саме розгорнуто і виконується в системі. Пунктирні стрілки між компонентами позначають залежності, де один компонент використовує або потребує інший.

На рис. 2.7 наведено діаграму компонентів системи “Mediculus”.

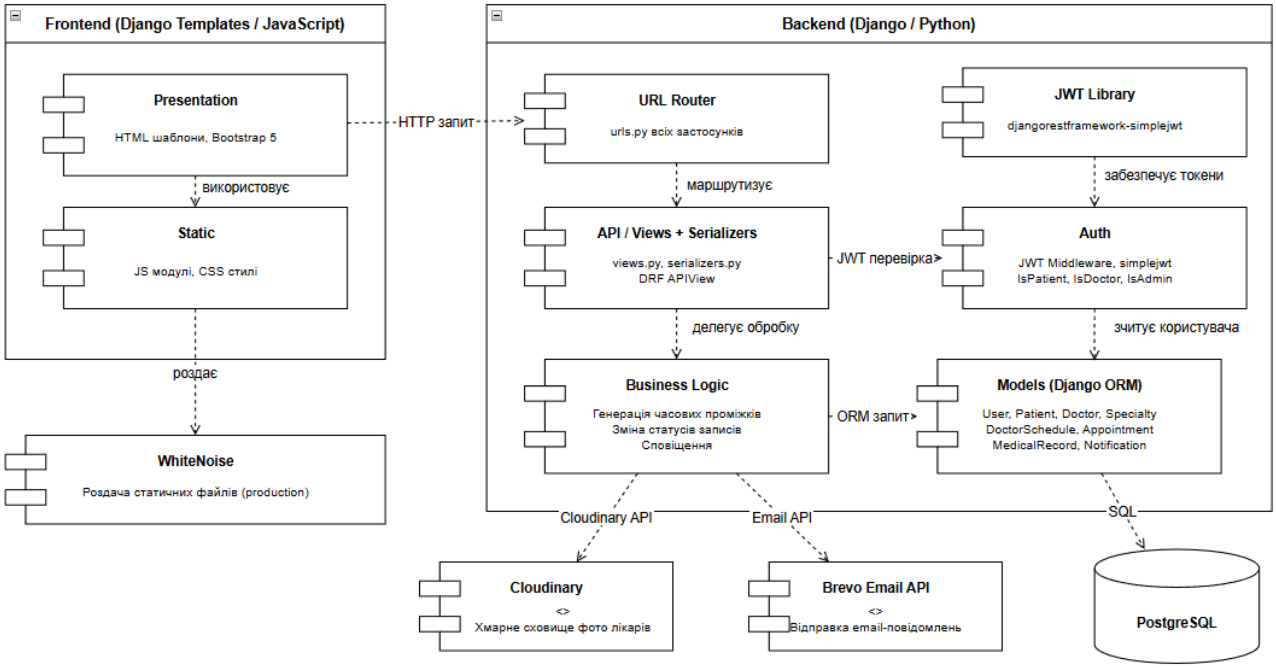

Рис. 2.7 Діаграма компонентів системи "Mediculus"

Система розділена на дві основні частини: Frontend і Backend, та взаємодіє з трьома зовнішніми сервісами.

Frontend (Django Templates / JavaScript)

Компонент *Presentation* відповідає за відображення інтерфейсу користувача. Він містить HTML-шаблони побудовані на основі Django Templates та оформлені за допомогою Bootstrap 5. Саме через цей компонент пацієнти, лікарі та адміністратор взаємодіють із системою – будь-яка дія користувача формує HTTP-запит до Backend.

Компонент *Static* містить JavaScript-модулі та CSS-стилі, які завантажуються браузером і виконуються на стороні клієнта. JavaScript-модулі відповідають за fetch-запити до REST API, динамічне оновлення інтерфейсу та роботу панелі доступності. Роздача цих файлів у production-середовищі здійснюється через компонент WhiteNoise.

Компонент *WhiteNoise* є бібліотекою, яка забезпечує ефективну роздачу статичних файлів безпосередньо з Django-застосунку без потреби в окремому веб-сервері.

Backend (Django / Python)

Компонент *URL Router* приймає всі вхідні HTTP-запити і маршрутизує їх до відповідних обробників на основі URL-адреси. Реалізований через urls.py файли кожного застосунку.

Компонент *API / Views + Serializers* є центральним контролером серверної частини. Він обробляє маршрутизовані запити, делегує перевірку автентифікації до Auth та передає логіку обробки до Business Logic. Серіалізатори відповідають за валідацію вхідних даних та перетворення об’єктів моделей у формат JSON.

Компонент *Auth* відповідає за автентифікацію та розмежування прав доступу. Містить JWT Middleware який перевіряє токен у заголовку кожного запиту, та класи прав доступу IsPatient, IsDoctor, IsAdmin, які визначають, що саме дозволено конкретній ролі. Для роботи з токенами використовує зовнішню бібліотеку JWT Library.

Компонент *JWT Library* – бібліотека djangorestframework-simplejwt яка забезпечує генерацію, перевірку та оновлення JWT-токенів. Виступає зовнішньою залежністю для компонента Auth.

Компонент *Business Logic* реалізує основну бізнес-логіку системи: генерацію вільних часових проміжків на основі розкладу лікаря, автоматичну зміну статусу прострочених записів та формування внутрішніх сповіщень. Також ініціює відправку email-повідомлень через Brevo та завантаження фото через Cloudinary.

Компонент *Models (Django ORM)* містить всі моделі даних системи: User, Patient, Doctor, Specialty, DoctorSchedule, Appointment, MedicalRecord, Notification. Забезпечує взаємодію між бізнес-логікою та базою даних через Django ORM.

Зовнішні сервіси

PostgreSQL – реляційна база даних розгорнута на платформі Railway. Зберігає всі дані системи і є кінцевою ланкою в ланцюжку обробки запиту. Компонент Models звертається до неї через SQL-запити сформовані Django ORM [16].

Cloudinary – хмарний сервіс для зберігання зображень. Використовується для завантаження та зберігання фотографій лікарів через Cloudinary API.

Brevo Email API – зовнішній сервіс для відправки електронних листів. Використовується для верифікації email при реєстрації та відновлення паролю. Запити до Brevo виконуються асинхронно у фоновому потоці, щоб не блокувати основний цикл обробки запитів.

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка інформаційного та програмного забезпечення є центральним етапом створення системи. На цьому етапі проєктні рішення з попереднього розділу перетворюються на реальний працюючий код, де визначається система управління базою даних, розробляється її фізична структура, обирається інструментарій та реалізується бізнес-логіка системи [7].

3.1 Система управління інформаційною базою

Для зберігання даних системи “Mediculus” обрано реляційну систему управління базами даних PostgreSQL. PostgreSQL є офіційно рекомендованою СУБД для Django-проєктів, саме її розробники Django вказують як основну у документації [12].

У проєкті структура бази даних створюється не через написання SQL-запитів вручну, а через Python-код за допомогою системи міграцій Django ORM. Структуру таблиць описано у вигляді Python-класів – моделей, після чого Django автоматично генерує та виконує необхідні SQL-команди для PostgreSQL. Це спрощує розробку та унеможливлює помилки при ручному написанні SQL [12].

PostgreSQL має кілька переваг над альтернативними СУБД, які є дуже важливими саме для даної системи:

По-перше, підтримка часткових індексів з умовою WHERE. У системі реалізовано унікальне обмеження на пару (лікар + дата/час) лише для активних записів зі статусом planned або completed. Якщо пацієнт скасував запис, то інший пацієнт може записатися на цей самий час до того самого лікаря. Така

логіка реалізується через UniqueConstraint з параметром condition і є неможливою в MySQL.

По-друге, повна підтримка типу TIMESTAMPTZ – дата і час з урахуванням часового поясу. Це необхідно для коректного відображення розкладу лікарів при переході між літнім та зимовим часом у часовому поясі Europe/Kyiv, який використовується в системі.

По-третє, PostgreSQL підтримує успадкування таблиць на рівні бази даних. У системі реалізовано патерн, де модель User є базовою, а Patient і Doctor успадковують від неї через зв'язок OneToOneField. Саме PostgreSQL коректно підтримує такий підхід до організації ієрархії сутностей.

По-четверте, PostgreSQL офіційно підтримується платформою Railway, на якій розгорнута система. Railway має вбудовану підтримку PostgreSQL як окремого сервісу, що спрощує розгортання та налаштування бази даних без додаткової інфраструктури. З'єднання між Django-застосунком і базою даних здійснюється через драйвер psycopg2.

3.2 Розробка інформаційної бази

Інформаційна база системи "Mediculus" реалізована засобами Django ORM. На відміну від звичайного підходу, де спочатку проєктується логічна модель, а потім вручну пишуться SQL-запити для створення таблиць, у Django структура бази даних описується безпосередньо у вигляді Python-класів. Команда makemigrations автоматично генерує файли міграцій на основі цих класів, а migrate застосовує їх до бази даних. Типи полів, обмеження, індекси та зв'язки між таблицями визначаються в коді моделей і автоматично перетворюються у відповідні SQL-конструкції PostgreSQL.

База даних містить вісім таблиць, які відповідають моделям системи. Всі первинні ключі мають тип BIGSERIAL, що забезпечує автоматичну генерацію унікальних ідентифікаторів.

Таблиця "accounts_user" є центральною і зберігає дані всіх користувачів системи незалежно від їх ролі. Для входу використовується електронна пошта.

Кожен користувач має визначену роль: пацієнт, лікар або адміністратор. Передбачено механізм верифікації електронної пошти при реєстрації та відновлення забутого паролю через посилання на пошту. Також реалізовано патерн м'якого видалення – видалені користувачі не видаляються фізично з бази даних, а лише позначаються як видалені з фіксацією дати видалення.

Таблиця "accounts_patient" розширює дані користувача інформацією, яка є особливою для пацієнта – датою народження та статтю. Видалення користувача автоматично видаляє відповідний запис пацієнта.

Таблиці "doctors_specialty", "doctors_doctor" та "doctors_doctorschedule" зберігають дані про лікарів. Фотографія лікаря завантажується в хмарне сховище Cloudinary, а у базі даних зберігається лише посилання на це фото. Тривалість одного прийому налаштовується індивідуально для кожного лікаря. Розклад зберігається окремим записом для кожного дня тижня і не може дублюватись.

Таблиця "appointments_appointment" є центральною з точки зору бізнес-логіки. Запис може перебувати в одному з чотирьох станів: запланований, завершений, скасований або пропущений. Реалізовано частковий унікальний індекс, який гарантує, що лікар не може мати два активних записи на один і той самий час, але після скасування часовий проміжок звільняється для нового запису.

Таблиця "appointments_medicalrecord" зберігає результати завершеного прийому, а саме: діагноз, призначене лікування та приватні нотатки лікаря. Кожен запис на прийом може мати лише одні медичні результати, і вони автоматично видаляються разом із записом.

Таблиця "notifications_notification" зберігає внутрішні сповіщення для користувачів з можливістю відстеження прочитаних та непрочитаних повідомлень.

Щоб показати, як саме побудовані моделі, далі наведено фрагменти коду двох основних класів. Більш детальний лістинг моделей наведено у додатку А.

Модель “User” успадковує від AbstractBaseUser, що дозволяє використовувати email як логін замість стандартного username, та визначає три ролі користувача через внутрішній клас Role:

```
class User(AbstractBaseUser, PermissionsMixin):

    class Role(models.TextChoices):
        PATIENT = 'patient', 'Пацієнт'
        DOCTOR  = 'doctor',  'Лікар'
        ADMIN   = 'admin',   'Адміністратор'

    email  = models.EmailField(unique=True)
       role     =  models.CharField(max_length=10,  choices=Role.choices,
default=Role.PATIENT)
    is_deleted = models.BooleanField(default=False)
    deleted_at = models.DateTimeField(null=True, blank=True)

    USERNAME_FIELD = 'email'
```

Лістинг 3.1 Фрагменд коду моделі User

Модель “Appointment” визначає частковий унікальний індекс через UniqueConstraint з параметром condition, завдяки чому лікар не може мати два активних записи на один час, але після скасування часовий проміжок звільняється:

```
    class Meta:
        verbose_name = 'Запис на прийом'
        verbose_name_plural = 'Записи на прийом'
        ordering = ['-date_time']
        constraints = [
            models.UniqueConstraint(
                fields=['doctor', 'date_time'],
                condition=~Q(status__in=['cancelled', 'missed']),
                name='unique_active_appointment'
            )
        ]
```

Лістинг 3.2 Фрагменд коду з частковим унікальним обмеження моделі Appointment

Індекси та обмеження цілісності

Окрім первинних ключів у базі даних створено додаткові індекси для прискорення пошуку та забезпечення цілісності даних. Django автоматично створює індекси для всіх зовнішніх ключів та полів з обмеженням UNIQUE. Найважливішим є частковий унікальний індекс для активних записів, який Django автоматично перетворює у наступний SQL-вираз:

```
CREATE UNIQUE INDEX unique_active_appointment
ON appointments_appointment (doctor_id, date_time)
WHERE status NOT IN ('cancelled', 'missed');
```

У базі даних визначено такі правила каскадного видалення: видалення користувача автоматично видаляє записи пацієнта або лікаря; видалення лікаря видаляє його розклад та всі записи на прийом; видалення запису на прийом видаляє медичні результати; видалення користувача видаляє всі його сповіщення. Для зв’язку між лікарем та спеціальністю застосовано on_delete = models.SET_NULL, що дозволяє видаляти спеціальності, не видаляючи лікаря, а лише знімати прив’язку.

3.3 Вибір інструментарію для створення прикладного ПЗ

3.3.1 Інструментарій серверної частини

Серверна частина системи “Mediculus” реалізована мовою програмування Python версії 3.13. Python є однією з найпоширеніших мов для розробки веб-застосунків завдяки простому синтаксису, великій кількості бібліотек та активній спільноті розробників [13].

Основним фреймворком для розробки серверної частини обрано Django версії 5.2. Django є повнофункціональним веб-фреймворком, який реалізує патерн MVT (Model-View-Template) і надає готові інструменти для роботи з базою даних, автентифікацією, маршрутизацією та адміністративною панеллю. Головною перевагою Django є те, що більшість типових задач веб-розробки вирішується вбудованими засобами без підключення сторонніх бібліотек [12].

Автентифікація користувачів реалізована на основі JWT-токенів за допомогою бібліотеки djangorestframework-simplejwt. При успішному вході користувач отримує два токени: access з терміном дії 2 години та refresh з терміном дії 7 днів. Після закінчення терміну дії access-токена клієнт може отримати новий не проходячи повторну автентифікацію. Для підвищення безпеки було реалізовано механізм регулярного оновлення refresh-токенів, де після кожного використання старий токен блокується через чорний список.

Для побудови REST API використовується Django REST Framework (DRF) – це розширення Django, яке спрощує створення програмних інтерфейсів. DRF надає зручні інструменти для серіалізації даних, валідації вхідних запитів, розмежування прав доступу та автоматичної генерації документації API [14].

Для документування API використовується бібліотека drf-spectacular, яка автоматично генерує OpenAPI-схему на основі визначених у коді серіалізаторів та view-класів.

3.3.2 Інструментарій клієнтської частини

Клієнтська частина системи “Mediculus” реалізована без використання окремих JavaScript-фреймворків, просто на основі Django Templates, Bootstrap 5 та Vanilla JavaScript.

Django Templates є вбудованою системою шаблонів Django, яка дозволяє генерувати HTML-сторінки на сервері з динамічним вмістом. Завдяки підтримці успадкування шаблонів, усі сторінки системи наслідують загальний вигляд від головного шаблону base.html. Він містить базові елементи, такі як: навігаційна панель, футер та панель доступності. [12].

Для оформлення інтерфейсу використовується Bootstrap 5.3. Це популярна CSS-бібліотека, яка надає готові компоненти та адаптивну сітку. Завдяки Bootstrap інтерфейс системи коректно відображається як на комп’ютерах так і на мобільних пристроях без написання додаткового CSS-коду [15].

Динамічна взаємодія з сервером реалізована через Vanilla JavaScript з використанням fetch API для асинхронних запитів до REST API. Кожна сторінка має окремий JS-модуль який відповідає за її логіку. Наприклад, doctors.js для сторінки списку лікарів, doctor-detail.js для сторінки профілю лікаря з бронюванням часу, notifications.js для відображення та оновлення сповіщень в реальному часі. Також реалізовано окремий модуль accessibility.js, який забезпечує роботу панелі доступності з налаштуванням розміру шрифту, контрастності та вибору шрифту.

3.3.3 Зовнішні сервіси

У процесі розробки системи використано три зовнішніх сервіси, які забезпечують функціональність, що виходить за межі серверної логіки.

Cloudinary це хмарний сервіс для зберігання та управління медіафайлами. Використовується для зберігання фотографій лікарів. При завантаженні фото, воно надсилається безпосередньо на сервери Cloudinary через їх API, а в базі даних зберігається лише ідентифікатор зображення. Це дозволяє не навантажувати сервер зберіганням медіафайлів та отримати автоматичну оптимізацію зображень [17].

Brevo це сервіс для відправки електронних листів через HTTP API. Використовується для верифікації електронної пошти при реєстрації та відновлення забутого паролю. Оскільки SMTP-порти заблоковані на платформі Railway, відправка листів реалізована через HTTP-запити до Brevo API у фоновому потоці, щоб не блокувати основний процес обробки запитів [18].

Railway це хмарна платформа для розгортання веб-застосунків. Обрана через просту інтеграцію з GitHub, вбудовану підтримку PostgreSQL, як окремого сервісу, та автоматичне розгортання при кожному push до репозиторію [19].

3.4 Алгоритмізація та програмування програмних модулів

Алгоритмізація потрібна для того, щоб заздалегідь визначити точний порядок дій системи і не плутатися під час написання коду. Блок-схема алгоритму описує послідовність дій, умовні розгалуження та можливі результати виконання процесу у графічному вигляді [7]. У даному підрозділі представлено алгоритми запису пацієнта до лікаря та реєстрації користувача, а також наведено фрагменти коду, що демонструють особливості їх реалізації.

3.4.1 Алгоритм запису пацієнта до лікаря

Одним з головних процесів системи “Mediculus” є запис пацієнта на прийом до лікаря. Алгоритм охоплює як клієнтську частину з вибіром дати та часового проміжку, так і серверну, де відбувається перевірка доступності та збереження запису. На рис. 3.1 наведено блок-схему цього алгоритму.

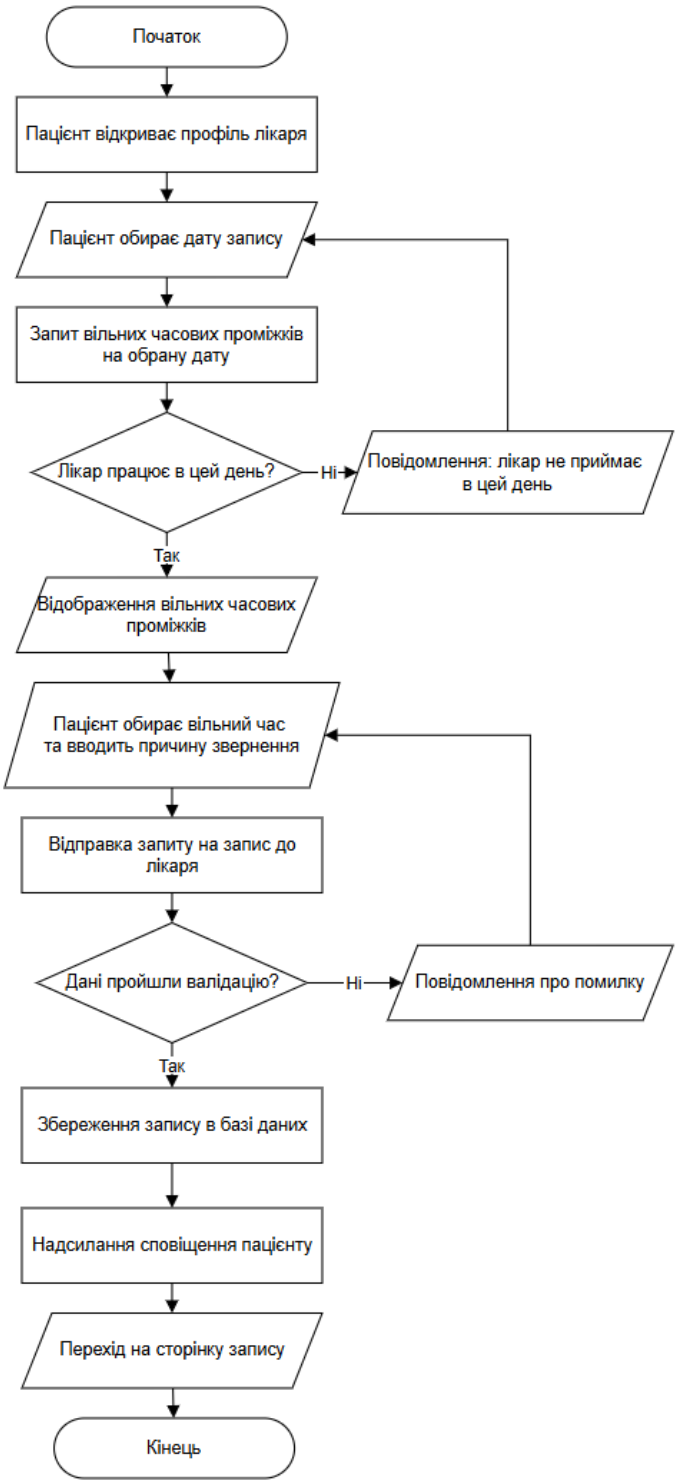

Рис. 3.1 Блок-схема алгоритму запису пацієнта до лікаря

Процес починається з того що пацієнт відкриває сторінку профілю лікаря і обирає бажану дату прийому. Після вибору дати клієнтська частина автоматично надсилає запит до сервера для отримання вільних часових проміжків на обрану дату.

Сервер перевіряє чи лікар працює у вибраний день тижня відповідно до його розкладу. Якщо лікар не приймає в цей день, то користувач отримує відповідне повідомлення і може обрати іншу дату. Якщо лікар працює, то сервер генерує перелік вільного часу, доступного для запису, від початку до кінця робочого дня з кроком, що дорівнює тривалості одного прийому лікаря, після чого фільтрує вже зайняті проміжки часу та ті, що вже минули. Список вільних проміжків відображається на сторінці.

Пацієнт обирає зручний час, за бажанням вводить причину звернення і натискає кнопку для запису. Клієнтська частина перевіряє чи пацієнт авторизований та чи має роль пацієнта, після чого надсилає запит на створення запису до сервера.

Сервер виконує валідацію отриманих даних з перевіркою таких умов: час запису має бути в майбутньому; дата не може бути більше ніж на 14 днів від поточної; лікар має бути активним у системі; обраний час не повинен бути зайнятий іншим пацієнтом; день має бути робочим у розкладі лікаря; час має знаходитись в межах робочих годин лікаря; пацієнт не може мати більше двох активних записів до цього лікаря загалом; пацієнт не може записатись до того самого лікаря двічі на одну дату. Коли хоча б одна з перевірок не пройдена, тоді сервер повертає повідомлення про помилку і пацієнт може обрати інший час для запису.

Якщо всі перевірки пройдені успішно, то сервер зберігає запис у базі даних зі статусом “заплановано” та надсилає внутрішнє сповіщення пацієнту з підтвердженням запису. Після успішного створення запису, пацієнт автоматично переходить на сторінку деталей свого запису.

Ключовим фрагментом серверної логіки є функція генерації вільних часових проміжків. Вона будує перелік можливих слотів від початку до кінця робочого дня лікаря з заданим кроком, після чого виключає вже зайняті:

```
slots = []
        current_time = datetime.combine(target_date, schedule.work_start)
        end_time = datetime.combine(target_date, schedule.work_end)
        slot_delta = timedelta(minutes=doctor.slot_duration)
```

```
while current_time + slot_delta <= end_time:
    slots.append(current_time)
    current_time += slot_delta

from apps.appointments.models import Appointment
booked_times = Appointment.objects.filter(
    doctor=doctor,
    date_time__date=target_date,
    status__in=['planned', 'completed']
).values_list('date_time', flat=True)
```

Лістинг 3.3 Генерація вільних часових проміжків

Повний код серверної валідації запису наведено у додатку Б.

3.4.2 Алгоритм реєстрації користувача

Другим ключовим процесом системи є реєстрація нового користувача з верифікацією електронної пошти. Алгоритм охоплює клієнтську валідацію даних форми, серверну обробку запиту, створення облікового запису та підтвердження email через одноразовий код. На рис. 3.2 наведено блок-схему цього алгоритму.

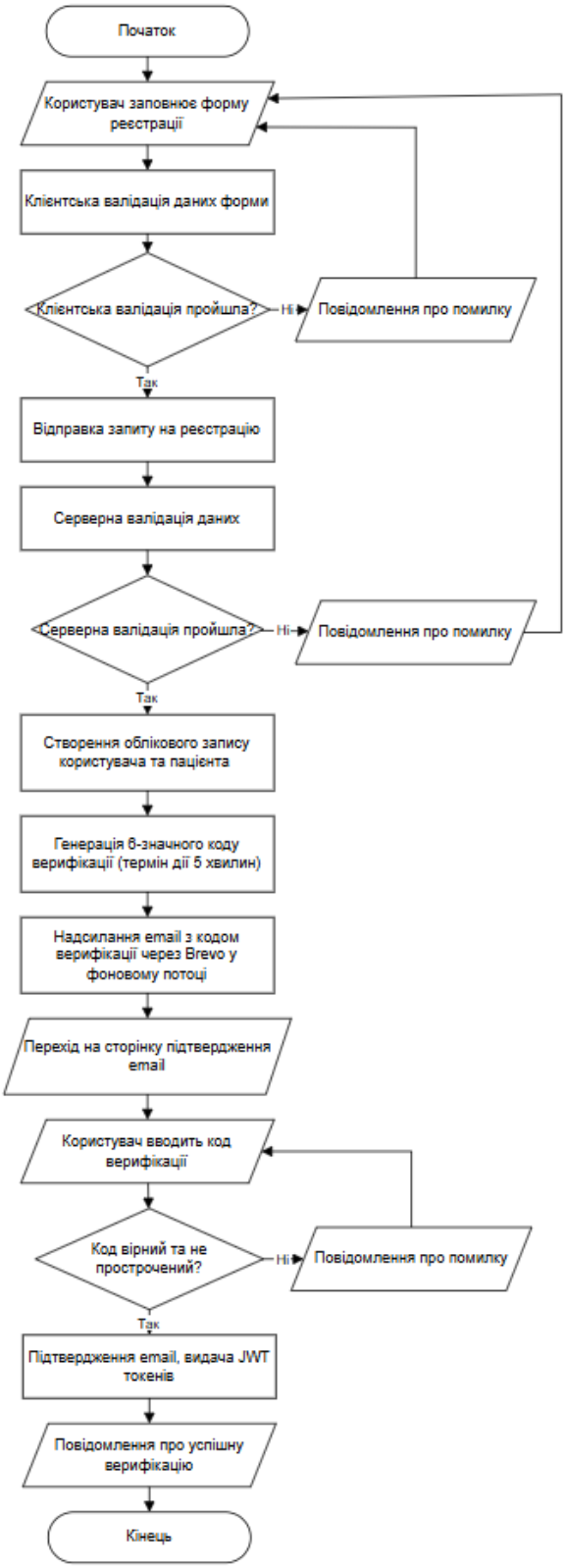

Рис. 3.2 Блок-схема алгоритму реєстрації користувача

Процес починається з того, що користувач заповнює форму реєстрації. Він вводить прізвище, ім’я, по батькові, електронну пошту, телефон, дату народження, стать, пароль та підтвердження паролю. Після натискання кнопки

реєстрації виконується клієнтська валідація даних. Перевіряються такі умови: дата народження не може бути в майбутньому, дата народження не раніше 1900 року і вік користувача має бути не менше одного року; паролі мають співпадати; пароль має містити мінімум 8 символів, хоча б одну літеру та хоча б одну цифру. Якщо клієнтська валідація не пройшла, то користувач отримує повідомлення про помилку і може виправити дані форми.

Після успішної клієнтської валідації дані відправляються на сервер. Сервер виконує власну валідацію: перевіряє унікальність електронної пошти в базі даних, складність паролю та збіг паролів. Якщо серверна валідація не пройшла, то повертається повідомлення про помилку і користувач може виправити дані.

Якщо валідація пройшла успішно, сервер створює акаунт користувача та профіль пацієнта в межах однієї транзакції. Це гарантує, що вони або створяться разом, або не створиться нічого, якщо виникне помилка. Пароль при цьому хешується алгоритмом PBKDF2-SHA256 і зберігається в базі даних у вигляді хешу, а не відкритого тексту.

Після створення облікового запису сервер генерує випадковий 6-значний цифровий код верифікації з терміном дії 5 хвилин і зберігає його в базі даних. Email з кодом надсилається через сервіс Brevo асинхронно у фоновому потоці. Тому сервер не чекає відправки листа і може одразу повернути відповідь. Важлива особливість цього алгоритму полягає в тому, що JWT токени не видаються одразу після реєстрації, а лише після підтвердження email.

Користувач автоматично переходить на сторінку підтвердження email, де вводить отриманий код. Сервер перевіряє чи код вірний та чи не закінчився термін його дії. Якщо код невірний або прострочений, то відображається повідомлення про помилку і користувач може ввести код повторно або запросити новий. Після успішної верифікації сервер підтверджує email користувача і видає JWT токени. З цього моменту користувач вважається авторизованим. На сторінці з’являється повідомлення про успішну верифікацію.

Ключовим фрагментом серверної логіки є функція надсилання email через Brevo у фоновому потоці:

```
def send_email_async(subject, body, from_email, to_list, html=False):
    def _send():
        try:
            api_key = os.environ.get('BREVO_API_KEY')
            url = "https://api.brevo.com/v3/smtp/email"
            headers = {
                "accept": "application/json",
                "api-key": api_key,
                "content-type": "application/json",
            }
            data = {
                "sender": {"email": from_email, "name": "Mediculus"},
                "to": [{"email": email} for email in to_list],
                "subject": subject,
            }
            if html:
                data["htmlContent"] = body
            else:
                data["textContent"] = body
            response = requests.post(url, json=data, headers=headers)
                        print(f"EMAIL   SENT   OK:   {response.status_code}
{response.text}", flush=True)
        except Exception as e:
            print(f"EMAIL ERROR: {e}", flush=True)
    thread = threading.Thread(target=_send, daemon=True)
    thread.start()
```

Лістинг 3.4 Асинхронна відправка email

Повний код серверної валідації реєстрації та логіки верифікації email наведено у додатку В.

4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення це процес перевірки відповідності системи визначеним вимогам та виявлення можливих помилок у її роботі [7]. Якісне тестування є обов'язковою умовою перед розгортанням системи в реальному середовищі тому, що воно дозволяє переконатись, що кожен компонент працює коректно як окремо так і у взаємодії з іншими.

Для тестування системи "Mediculus" застосовано підхід, що відповідає двом нижнім рівням тестової піраміди, а саме: юніт-тестам та інтеграційним тестам. Додатково розроблено тести прав доступу, як окремий підвид інтеграційних тестів, що перевіряє матрицю доступу користувачів. Роль наскрізного тестування виконало ручне тестування через браузер за підготовленими сценаріями. Саме це тестування перевіряє систему з точки зору реального користувача, включаючи інтерфейс та повний користувацький шлях.

Юніт-тести перевіряють окремі методи та класи ізольовано від інших компонентів системи без реальних HTTP-запитів до API. В застосунку accounts перевіряються методи моделі User: формування повного імені, властивості визначення ролі, хешування паролю та значення полів за замовчуванням. Окремо тестується валідація серіалізатора реєстрації – унікальність email, складність паролю, збіг паролів та обов'язковість полів. Юніт-тести модуля doctors перевіряють алгоритм генерації вільних часових проміжків: кількість слотів, їх часові межі та поведінку за різних налаштувань розкладу, фільтрацію зайнятих проміжків залежно від статусу запису та валідацію вхідних параметрів запиту.

Інтеграційні тести перевіряють повні сценарії від HTTP-запиту до змін у базі даних. Вони охоплюють одразу кілька компонентів системи: view, серіалізатор, модель та систему сповіщень. Вони охоплюють повний цикл

реєстрації та верифікації email, від створення облікового запису до отримання JWT токенів після підтвердження пошти. Тестується цикл запису на прийом: створення запису, перевірка збереження в базі даних, відправка сповіщень та звільнення часового проміжку після скасування. Також перевіряється завершення прийому лікарем з автоматичним створенням результатів прийому та сповіщенням пацієнта.

Тести прав доступу є окремим підвидом інтеграційних тестів, які зосереджені на матриці доступу користувачів. Вони перевіряють, що Гість отримує помилку при спробі доступу до захищених ресурсів, пацієнт не може виконати дії лікаря і навпаки, а дані одного користувача недоступні іншому – система повертає помилку, замість чужих даних.

Всі розроблені тести виконуються успішно. Результати тестування підтверджують, що система відповідає визначеним функціональним вимогам та коректно розділяє права доступу між користувачами різних ролей.

Окрім автоматизованого тестування проведено ручне тестування основних сценаріїв використання системи через інтерфейс браузера. Нижче наведено **сценарії ручного тестування:**

1. Реєстрація пацієнта з верифікацією email. Гість відкриває сторінку реєстрації, заповнює форму з коректними даними та паролем, що відповідає вимогам складності, погоджується з політикою конфіденційності і натискає кнопку “Зареєструватись”. Переходить на сторінку верифікації і вводить 6-значний код отриманий на пошту.

Результат: email підтверджено, JWT токени збережено в браузері, користувач може авторизуватись.

2. Авторизація користувача. Пацієнт з підтвердженим email відкриває сторінку входу, вводить коректні дані та натискає кнопку “Увійти”.

Результат: токени збережено, у навігаційній панелі відображається кнопки: мій профіль, дзвоник із сповіщеннями та кнопка для виходу з профілю. При спробі входу з непідтвердженим email, користувач перенаправляється на сторінку для верифікації.

3. Перегляд списку лікарів з фільтрацією. Гість відкриває сторінку лікарів, застосовує фільтр за спеціальністю та вводить прізвище в поле пошуку.

Результат: список звужується відповідно до обраних фільтрів, після їх очищення відображаються всі активні лікарі.

4. Запис пацієнта до лікаря. Пацієнт відкриває профіль лікаря, обирає дату, натискає на вільний часовий проміжок, вводить причину звернення і натискає кнопку “Записатись”.

Результат: перехід на сторінку деталей запису зі статусом “Заплановано”, у розділі сповіщень з’являється підтвердження запису.

5. Скасування запису пацієнтом. Пацієнт відкриває список своїх записів, обирає запис зі статусом “Заплановано” і підтверджує скасування.

Результат: статус запису змінюється на “Скасовано”, часовий проміжок звільняється, лікар отримує внутрішнє сповіщення про скасування.

6. Завершення прийому лікарем із внесенням діагнозу. Лікар відкриває список записів, обирає запланований прийом, час якого вже настав, натискає “Завершити прийом”, вносить діагноз та призначення і зберігає.

Результат: статус змінюється на “Завершено”, результати прийому прив’язані до запису, пацієнт отримує сповіщення про завершення прийому.

7. Перегляд медичних результатів пацієнтом. Пацієнт відкриває завершений запис і переглядає розділ результатів прийому.

Результат: відображаються діагноз, призначення та дата заповнення результатів. Приватні нотатки лікаря пацієнту не відображаються.

8. Автоматичне позначення пропущеного прийому. Пацієнт відкриває сторінку своїх записів після того, як час запланованого прийому минув більше ніж на тривалість прийому плюс 10 хвилин.

Результат: запис автоматично отримує статус “Пропущено”, пацієнт і лікар отримують відповідні сповіщення.

9. Завантаження фото лікарем. Лікар відкриває особистий кабінет, завантажує фото у форматі JPEG, PNG або WebP.

Результат: фото завантажується до Cloudinary, відображається в кабінеті лікаря та на його публічній сторінці. При спробі завантажити файл невірного формату або розміру буде відображено відповідне повідомлення про помилку.

10. Перегляд та позначення сповіщень як прочитаних. Авторизований користувач відкриває розділ сповіщень, переглядає непрочитані повідомлення і натискає “Позначити всі як прочитані”.

Результат: лічильник непрочитаних сповіщень обнуляється, сповіщення з посиланням при натисканні відкриває відповідну сторінку.

Нижче наведено **детальний приклад виконання сценарію тестування реєстрації пацієнта з верифікацією email:**

Гість відкриває сторінку реєстрації та заповнює форму особистими даними. Вводить ім’я, прізвище, електронну пошту, пароль, що відповідає вимогам складності та інші поля. Після заповнення погоджується з політикою конфіденційності та натискає кнопку “Зареєструватись”.

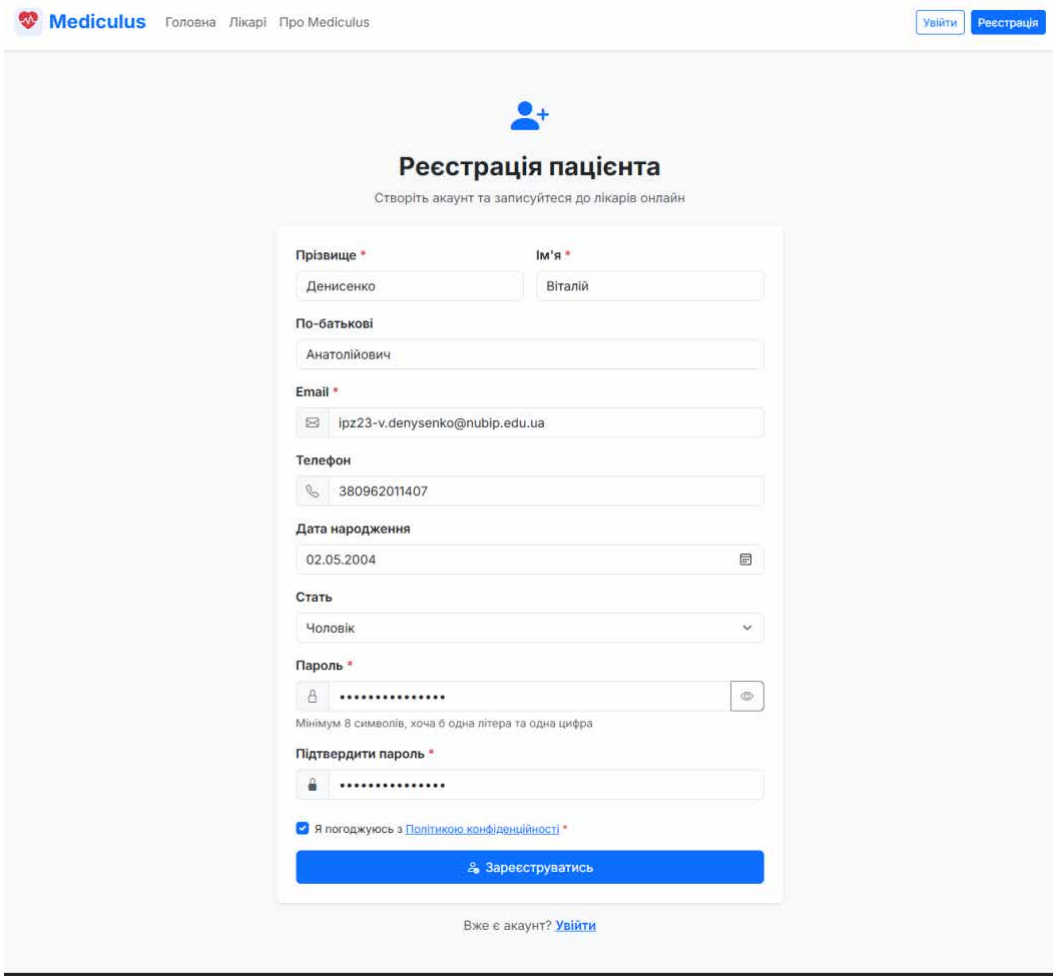

Рис. 4.1 Заповнена форма реєстрації

Система виконує валідацію даних і надсилає запит на сервер. Сервер створює обліковий запис користувача та пов’язаний запис пацієнта, генерує 6-значний код верифікації з терміном дії 5 хвилин і надсилає його на вказану електронну пошту через Brevo у фоновому потоці. Користувач отримує лист від Mediculus з кодом підтвердження.

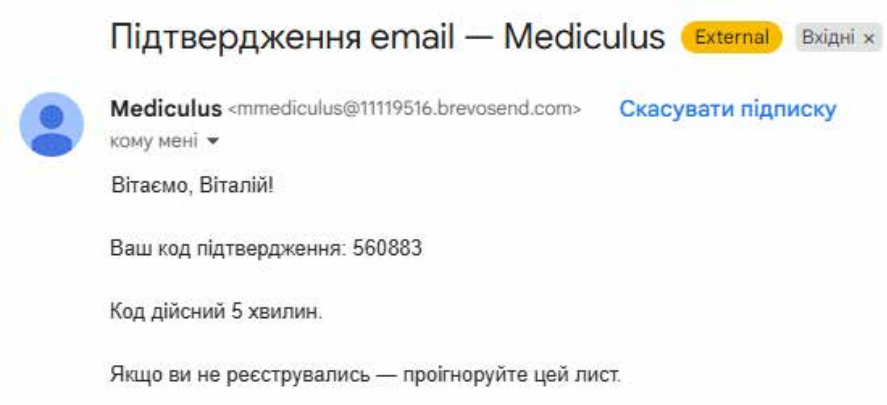

Рис. 4.2 Лист з кодом верифікації на електронній пошті

Користувач переходить на сторінку підтвердження email, вводить отриманий код і натискає кнопку підтвердження. Сервер перевіряє код та його термін дії, після чого підтверджує email і видає JWT токени. На сторінці з’являється повідомлення про успішну верифікацію та кнопка для переходу до профілю.

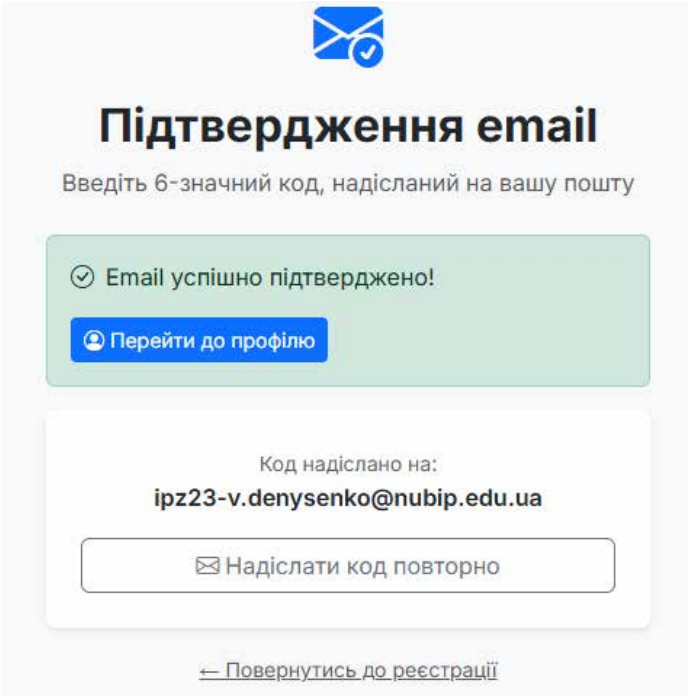

Рис. 4.3 Повідомлення про успішне підтвердження email

Користувач натискає кнопку “Перейти до профілю” і потрапляє до особистого кабінету пацієнта, де доступний повний функціонал системи.

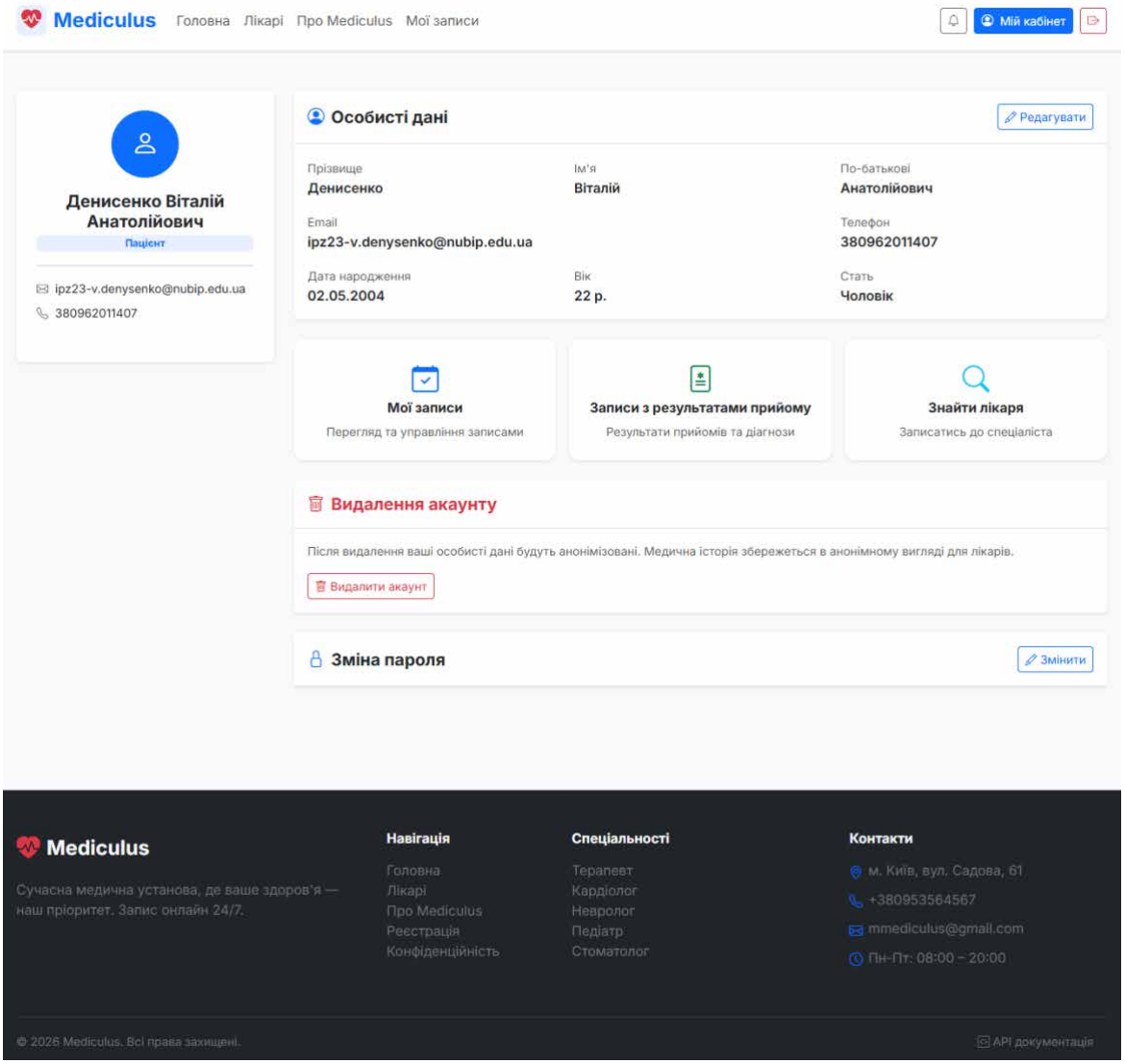

Рис. 4.4 Особистий кабінет пацієнта

Результат: email підтверджено, JWT токени збережено в браузері, користувач успішно авторизований і має доступ до особистого кабінету. Сценарій виконано успішно.

Екранні форми інших сценаріїв тестування наведено у додатку Г.

4.2 Вимоги до апаратного та програмного забезпечення

Система “Mediculus” розгорнута за клієнт-серверною архітектурою на хмарній платформі Railway. Серверна частина обробляє HTTP-запити від

браузерів користувачів, взаємодіє з базою даних PostgreSQL та надає REST API для клієнтської частини. Клієнтська частина є веб-застосунком, який відображається прямо у браузері без встановлення додаткового програмного забезпечення.

Діаграма розгортання показує фізичну топологію системи: на яких пристроях і в яких середовищах виконання розгорнуті компоненти програмного забезпечення [5]. Основними елементи діаграми є вузли, які позначають фізичне або віртуальне обладнання, середовища виконання всередині них та артефакти, які показують конкретні файли і модулі, які розгорнуті в цих середовищах.

На рис. 4.5 наведено діаграму розгортання системи “Mediculus”.

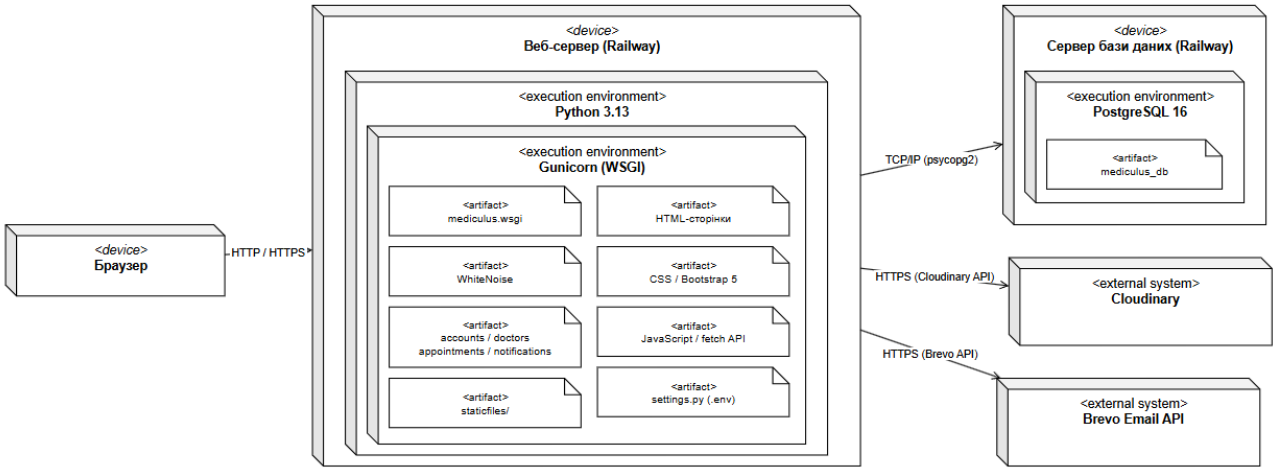

Рис. 4.5 Діаграма розгортання системи “Mediculus”

Система розгорнута на платформі Railway і взаємодіє з двома зовнішніми сервісами. Діаграма містить три вузли пристроїв та два зовнішніх сервіси.

Вузол “Браузер” є клієнтським пристроєм через який користувачі взаємодіють із системою. Браузер надсилає HTTP/HTTPS запити до веб-сервера і отримує у відповідь HTML-сторінки, статичні файли та JSON-відповіді від REST API.

Вузол “Веб-сервер (Railway)” є основним обчислювальним вузлом системи. Всередині нього розгорнуто середовище виконання Python 3.13, в

якому працює Gunicorn. Gunicorn є WSGI-сервером який запускає Django-застосунок через інтерфейс mediculus.wsgi і обробляє вхідні HTTP-запити.

Всередині середовища виконання “Gunicorn” розгорнуті всі артефакти Django-застосунку: mediculus.wsgi як точка входу, WhiteNoise як middleware для роздачі статичних файлів, чотири Django-застосунки (accounts, doctors, appointments, notifications), директорія staticfiles з CSS та JavaScript файлами, HTML-шаблони, та файл конфігурації settings.py з параметрами середовища з .env файлу.

Вузол “Сервер БД (Railway)” містить середовище виконання PostgreSQL 17, в якому розгорнута база даних mediculus_db. Веб-сервер з’єднується з сервером бази даних через протокол TCP/IP за допомогою драйвера psycopg2.

Cloudinary це зовнішній сервіс для зберігання фотографій лікарів. Веб-сервер звертається до нього через HTTPS, використовуючи Cloudinary API при завантаженні або оновленні фото.

Brevo Email API це зовнішній сервіс для відправки електронних листів. Використовується для верифікації email при реєстрації та відновлення паролю. Запити до Brevo виконуються асинхронно у фоновому потоці через HTTPS, щоб не блокувати основний процес обробки запитів.

Вимоги з точки зору власника системи

Апаратні вимоги до веб-сервера визначаються параметрами хмарної платформи Railway, яка надає контейнерне середовище з динамічним розподілом ресурсів. Система розгорнута на тарифному плані Hobby, який включає до 8 віртуальних ядер та 8 ГБ оперативної пам’яті на сервіс і до 5 ГБ дискового простору. Мінімальні фактичні вимоги для коректної роботи системи: процесор з 2 віртуальними ядрами і більше, оперативна пам’ять від 1 ГБ, дисковий простір від 2 ГБ.

Програмне забезпечення веб-сервера базується на Python 3.13 і вище та фреймворку Django 5.2. Для обробки HTTP-запитів використовується WSGI-сервер Gunicorn версії 21.2.0, який налаштований на 2 робочих процеси із затримкою 120 секунд. Роздача статичних файлів забезпечується бібліотекою

WhiteNoise 6.7.0. Для побудови REST API використовується Django REST Framework 3.15.2, для JWT-автентифікації використовується бібліотека djangorestframework-simplejwt 5.3.1. З'єднання з базою даних PostgreSQL здійснюється через драйвер psycopg2-binary 2.9.11.

Окрім серверного програмного забезпечення система використовує два зовнішніх сервіси, які не потребують встановлення на сервері: Cloudinary для зберігання фотографій лікарів та Brevo для відправки email-повідомлень.

Апаратні вимоги до сервера бази даних: процесор з 2 віртуальними ядрами і вище, оперативна пам'ять від 512 МБ, дисковий простір від 1 ГБ. Програмна вимога – PostgreSQL версії 17 і вище. Railway автоматично надає PostgreSQL 17, як окремий сервіс незалежний від веб-сервера.

Вимоги з точки зору користувача системи

Для роботи з системою користувачу достатньо будь-якого пристрою з сучасним браузером та доступом до інтернету. Це може бути персональний комп'ютер, ноутбук або смартфон.

Система підтримує такі браузери: Google Chrome версії 90 і вище, Mozilla Firefox версії 88 і вище, Microsoft Edge версії 90 і вище, Safari версії 14 і вище та Opera версії 76 і вище.

Встановлення будь-якого додаткового програмного забезпечення на стороні користувача не потрібне тому, що система працює у браузері.

4.3 Склад інсталяційного пакету та розгортання системи

Система "Mediculus" є веб-застосунком розгорнутим на хмарній платформі Railway і доступним за адресою https://mediculus.up.railway.app/. Для розгортання системи використовується механізм безперервного розгортання через GitHub. Завдяки цьому при кожному push у репозиторій система автоматично оновлюється на сервері і одразу працює в останній версії.

До складу інсталяційного пакету входять такі компоненти:

- весь вихідний код проєкту у репозиторії GitHub;

- файл requirements.txt з переліком усіх залежностей Python. Railway автоматично встановлює їх при кожному розгортанні;
- файл Procfile який вказує платформі як запускати застосунок – через веб-сервер Gunicorn з відповідними параметрами;
- файл settings.py з налаштуваннями для production-середовища. Всі секретні параметри: параметри бази даних, API-ключі Cloudinary та Brevo – зберігаються окремо у змінних середовища, а не в коді;
- міграції бази даних, які описують структуру всіх таблиць в БД.

Процес розгортання

Спочатку весь проєкт було завантажено до репозиторію на GitHub. На платформі Railway створено новий проєкт і підключено репозиторій, і вже після підключення було автоматично виконано перший деплой.

У розділі змінних середовища сервісу налаштовано параметри підключення до бази даних, секретний ключ Django, параметри JWT-токенів, API-ключі Cloudinary та Brevo, а також налаштування безпеки, який включає перелік дозволених доменів з яких система приймає запити.

Для бази даних у проєкті Railway додано окремий сервіс PostgreSQL 17. Railway автоматично розгорнув його як окремий контейнер, параметри підключення скопійовано до змінних середовища веб-сервісу. Django підключається до бази через драйвер psycopg2-binary, зчитуючи параметри з цих змінних.

При кожному розгортанні Railway автоматично виконує застосування міграцій бази даних та збір статичних файлів, після чого запускає застосунок командою з Procfile. Весь процес відбувається без ручного втручання після виконання команди git push у гілку main.

Railway також автоматично керує SSL-сертифікатами – система доступна через HTTPS без додаткових налаштувань.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи на тему “Програмне забезпечення інформаційної системи для реєстрації пацієнтів лікарні” було виконано повний цикл створення веб-застосунку, від аналізу предметної області та проєктування до реалізації, тестування і розгортання системи на хмарній платформі.

Аналіз предметної області дозволив виявити основні проблеми традиційного підходу до реєстрації пацієнтів у лікарнях: черги до реєстратури, ручне ведення журналів та відсутність зручного доступу пацієнтів до власних медичних даних. Також було проведено детальний аналіз існуючих програмних рішень, що використовуються в Україні: Helsi.me, Doc.ua, Medics.ua та Medikom.ua. Для кожної системи визначено переваги та недоліки, що дозволило краще сформулювати вимоги до розробки власної системи. На основі проведеного аналізу визначено функціональні та нефункціональні вимоги до системи, які охоплюють весь цикл взаємодії користувача з нею, від реєстрації до автоматичного відстеження пропущених прийомів.

Для візуалізації системи було побудовано UML-діаграми, які включають діаграму прецедентів для опису взаємодії з користувачами, діаграми послідовності та активності для головних процесів, а також опис основних абстракцій. Для проєктування інформаційної бази побудовано ER-діаграму логічної моделі даних та перевірено на відповідність нормальним формам. Додатково розроблено діаграми класів, кооперацій, пакетів, компонентів та розгортання, що відображають архітектуру системи на різних рівнях деталізації.

Базу даних спроєктовано з восьми таблиць, що відповідають основним сутностям системи. Для забезпечення коректної логіки бронювання реалізовано

частковий унікальний індекс, який гарантує, що лікар не може мати два активні записи на один час, але дозволяє повторний запис після скасування. Серверну частину реалізовано на основі Python та фреймворку Django з використанням Django REST Framework для побудови REST API. Автентифікація реалізована на основі JWT-токенів з розмежуванням прав доступу залежно від ролі користувача. Клієнтську частину побудовано на Django Templates, Bootstrap та Vanilla JavaScript. Для зберігання фотографій лікарів підключено хмарний сервіс Cloudinary, а для відправки email-повідомлень з кодом верифікації – сервіс Brevo.

Тестування системи проводилось на кількох рівнях. Розроблено автоматизовані тести трьох видів: юніт-тести, що перевіряють логіку окремих компонентів, інтеграційні тести, які перевіряють повні сценарії від запиту до збереження в базі даних, та тести прав доступу, що підтверджують коректне розмежування між ролями. Окрім автоматизованого тестування, проведено ручне тестування основних сценаріїв використання системи. Усі тести було успішно виконано, що підтверджує відповідність розробленої системи визначеним вимогам. Систему розгорнуто на хмарній платформі Railway з налаштуванням автоматичного розгортання при кожному оновленні коду репозиторію.

Поставлена мета повністю досягнута. Система “Mediculus” забезпечує автоматизацію онлайн-запису до лікарів, зберігання результатів прийомів в електронному вигляді та управління медичними даними. Для кожної з трьох ролей користувачів: пацієнта, лікаря та адміністратора, реалізовано окремий набір функціоналу з відповідними правами доступу.

Розроблена система є повністю функціональним веб-застосунком, готовим до використання в реальних умовах. У майбутньому система може бути доповнена автоматичними нагадуваннями пацієнту про запис через email або SMS, можливістю залишати оцінку лікарю після завершеного прийому для формування рейтингу, а також можна розробити мобільний додаток.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Місяці в очікуванні прийому: що не так із системою онлайн-запису до лікарів та що з цим робити – [Електронний ресурс] – режим доступу: https://life.pravda.com.ua/society/misyaci-v-ochikuvanni-priyomu-shcho-ne-tak-iz-sistemoyu-onlayn-zapisu-do-likariv-ta-shcho-z-cim-robiti-304668/ (дата звернення: 15.05.2026)
2. Інформатизація закладу охорони здоров’я та автоматизація робочих процесів – [Електронний ресурс] – режим доступу: https://moz.gov.ua/uk/informatizaciya-zakladu-ohoroni-zdorov-ya-ta-avtomatizaciya-robochih-procesiv-2 (дата звернення: 15.05.2026)
3. МОЗ України відмовляється від паперових медичних карток – [Електронний ресурс] – режим доступу: https://interfax.com.ua/news/general/167015.html (дата звернення: 15.05.2026)
4. Sommerville I. Software Engineering. 10th ed. Pearson, 2016. 816 p.
5. Booch G., Maksimchuk R. A., Engle M. W. та ін. Object-Oriented Analysis and Design with Applications. 3rd ed. Addison-Wesley Professional, 2007. 720 p.
6. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 533 p.
7. Стадії циклу розробки ПЗ – [Електронний ресурс] – режим доступу: https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz/ (дата звернення: 15.05.2026)
8. Helsi – електронна медична система – [Електронний ресурс] – режим доступу: https://helsi.me/ (дата звернення: 15.05.2026)
9. Doc.ua – сервіс онлайн-запису до лікарів – [Електронний ресурс] – режим доступу: https://doc.ua/ (дата звернення: 15.05.2026)
10. Medics.ua – медичний портал – [Електронний ресурс] – режим доступу: https://medics.ua/ (дата звернення: 15.05.2026)

11. Medikom.ua – медична інформаційна система – [Електронний ресурс] – режим доступу: https://medikom.ua/ (дата звернення: 15.05.2026)
12. Django documentation. Django Software Foundation – [Електронний ресурс] – режим доступу: https://docs.djangoproject.com/en/5.2/ (дата звернення: 15.05.2026)
13. Python documentation. Python Software Foundation – [Електронний ресурс] – режим доступу: https://www.python.org/doc/ (дата звернення: 15.05.2026)
14. Django REST Framework documentation. Encode OSS Ltd – [Електронний ресурс] – режим доступу: https://www.django-rest-framework.org/ (дата звернення: 15.05.2026)
15. Bootstrap 5 documentation. The Bootstrap Authors – [Електронний ресурс] – режим доступу: https://getbootstrap.com/docs/5.3/ (дата звернення: 15.05.2026)
16. PostgreSQL documentation. The PostgreSQL Global Development Group – [Електронний ресурс] – режим доступу: https://www.postgresql.org/docs/ (дата звернення: 15.05.2026)
17. Cloudinary documentation. Cloudinary Ltd – [Електронний ресурс] – режим доступу: https://cloudinary.com/documentation (дата звернення: 15.05.2026)
18. Brevo API documentation. Brevo SAS – [Електронний ресурс] – режим доступу: https://developers.brevo.com/ (дата звернення: 15.05.2026)
19. Railway platform documentation – [Електронний ресурс] – режим доступу: https://docs.railway.com/ (дата звернення: 15.05.2026)
20. Функціональні та нефункціональні вимоги до програмного забезпечення – [Електронний ресурс] – режим доступу: https://wezom.com.ua/ua/blog/funktsionalni-ta-nefunktsionalni-vimogi-do-programnogo-zabezpechennya (дата звернення: 15.05.2026)
21. Database Normalization: 1NF, 2NF, 3NF & BCNF Examples – [Електронний ресурс] – режим доступу: https://www.digitalocean.com/community/tutorials/database-normalization (дата звернення: 15.05.2026)

ДОДАТКИ

ДОДАТОК А

Лістинг моделей системи

А.1 Моделі модуля акаунтів

```
class User(AbstractBaseUser, PermissionsMixin):

    class Role(models.TextChoices):
        PATIENT = 'patient', 'Пацієнт'
        DOCTOR = 'doctor', 'Лікар'
        ADMIN = 'admin', 'Адміністратор'

    email = models.EmailField(unique=True, verbose_name='Email')
    first_name = models.CharField(max_length=100, verbose_name='Ім\'я')
    last_name = models.CharField(max_length=100, verbose_name='Прізвище')
    patronymic = models.CharField(max_length=100, blank=True,
verbose_name='По-батькові')
    phone = models.CharField(max_length=20, blank=True,
verbose_name='Телефон')
    role = models.CharField(max_length=10, choices=Role.choices,
default=Role.PATIENT, verbose_name='Роль')
    is_active = models.BooleanField(default=True,
verbose_name='Активний')
    is_staff = models.BooleanField(default=False,
verbose_name='Персонал')
    created_at = models.DateTimeField(default=timezone.now,
verbose_name='Дата реєстрації')
    email_verified = models.BooleanField(default=False,
verbose_name='Email підтверджено')
    email_verification_code = models.CharField(max_length=6, null=True,
blank=True)
    email_verification_code_expires = models.DateTimeField(null=True,
blank=True)
    password_reset_token = models.CharField(max_length=64, null=True,
blank=True)
```

```
        password_reset_token_expires = models.DateTimeField(null=True,
blank=True)
        is_deleted = models.BooleanField(default=False,
verbose_name='Видалено')
        deleted_at = models.DateTimeField(null=True, blank=True)

        objects = UserManager()
        USERNAME_FIELD = 'email'
        REQUIRED_FIELDS = ['first_name', 'last_name']

        class Meta:
            verbose_name = 'Користувача'
            verbose_name_plural = 'Користувачі'
            ordering = ['-created_at']

    class Patient(models.Model):

        class Gender(models.TextChoices):
            MALE = 'male', 'Чоловік'
            FEMALE = 'female', 'Жінка'

        user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name='patient_profile')
        date_of_birth = models.DateField(null=True, blank=True,
verbose_name='Дата народження')
        gender = models.CharField(max_length=10, choices=Gender.choices,
blank=True, verbose_name='Стать')

        class Meta:
            verbose_name = 'Пацієнта'
            verbose_name_plural = 'Пацієнти'
```

А.2 Моделі модуля лікарів

```
    class Specialty(models.Model):
        name = models.CharField(max_length=100, verbose_name='Назва
спеціальності')
```

```
        icon = models.CharField(max_length=50, default='', verbose_name='Іконка')
        slug = models.SlugField(max_length=100, unique=True, verbose_name='URL-slug')
        description = models.TextField(blank=True, verbose_name='Опис')

        class Meta:
            verbose_name = 'Спеціальність'
            verbose_name_plural = 'Спеціальності'
            ordering = ['name']


    class Doctor(models.Model):
        user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='doctor_profile')
        specialty = models.ForeignKey(Specialty, on_delete=models.SET_NULL, null=True, related_name='doctors')
        experience_years = models.PositiveIntegerField(default=0, verbose_name='Досвід роботи (років)')
        description = models.TextField(blank=True, verbose_name='Опис')
        photo = CloudinaryField(blank=True, null=True, resource_type='image', verbose_name='Фото')
        is_active = models.BooleanField(default=True, verbose_name='Приймає пацієнтів')
        slot_duration = models.PositiveIntegerField(default=30, verbose_name='Тривалість прийому (хв)')

        class Meta:
            verbose_name = 'Лікаря'
            verbose_name_plural = 'Лікарі'
            ordering = ['user__last_name', 'user__first_name']


    class DoctorSchedule(models.Model):
        DAY_CHOICES = [
            (0, 'Понеділок'), (1, 'Вівторок'), (2, 'Середа'),
            (3, 'Четвер'), (4, 'П\'ятниця'), (5, 'Субота'), (6, 'Неділя'),
        ]
```

```
        doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='schedules')
        day_of_week = models.IntegerField(choices=DAY_CHOICES,
verbose_name='День тижня')
        work_start = models.TimeField(verbose_name='Початок роботи')
        work_end = models.TimeField(verbose_name='Кінець роботи')
        is_working = models.BooleanField(default=True, verbose_name='Робочий
день')

        class Meta:
            verbose_name = 'Розклад лікаря'
            verbose_name_plural = 'Розклади лікарів'
            unique_together = ['doctor', 'day_of_week']
            ordering = ['day_of_week']
```

А.3 Моделі модуля записів

```
class Appointment(models.Model):

    class Status(models.TextChoices):
        PLANNED = 'planned', 'Заплановано'
        COMPLETED = 'completed', 'Завершено'
        CANCELLED = 'cancelled', 'Скасовано'
        MISSED = 'missed', 'Пропущено'

    patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
related_name='appointments')
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='appointments')
    date_time = models.DateTimeField(verbose_name='Дата та час прийому')
    status = models.CharField(max_length=10, choices=Status.choices,
default=Status.PLANNED)
    reason = models.TextField(blank=True, verbose_name='Причина
звернення')
    created_at = models.DateTimeField(default=timezone.now)

    class Meta:
```

```
            verbose_name = 'Запис на прийом'
            verbose_name_plural = 'Записи на прийом'
            ordering = ['-date_time']
            constraints = [
                models.UniqueConstraint(
                    fields=['doctor', 'date_time'],
                    condition=~Q(status__in=['cancelled', 'missed']),
                    name='unique_active_appointment'
                )
            ]
    class MedicalRecord(models.Model):
        appointment = models.OneToOneField(Appointment,
on_delete=models.CASCADE, related_name='medical_record')
        diagnosis = models.TextField(verbose_name='Діагноз')
        treatment = models.TextField(verbose_name='Призначення / лікування')
        doctor_notes = models.TextField(blank=True, verbose_name='Приватні
нотатки лікаря')
        created_at = models.DateTimeField(default=timezone.now)

        class Meta:
            verbose_name = 'Результат прийому'
            verbose_name_plural = 'Результати прийому'
            ordering = ['-created_at']
```

А.4 Моделі модуля сповіщень

```
    class Notification(models.Model):
        user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='notifications')
        message = models.TextField(verbose_name='Повідомлення')
        is_read = models.BooleanField(default=False,
verbose_name='Прочитано')
        link = models.CharField(max_length=255, blank=True,
verbose_name='Посилання')
        created_at = models.DateTimeField(default=timezone.now)

        class Meta:
            verbose_name = 'Сповіщення'
```

```
        verbose_name_plural = 'Сповіщення'
        ordering = ['-created_at']
```

ДОДАТОК Б

Лістинг коду валідації запису та клієнтської логіки

Серіалізатор створення запису

```
class AppointmentCreateSerializer(serializers.ModelSerializer):
    doctor_id = serializers.IntegerField(write_only=True)
    date_time = serializers.DateTimeField()

    class Meta:
        model = Appointment
        fields = ['doctor_id', 'date_time', 'reason']

    def validate_date_time(self, value):
        today = timezone.now().date()
        if value < timezone.now():
            raise serializers.ValidationError('Час запису має бути в
майбутньому.')
        max_date = today + timedelta(days=14)
        if value.date() > max_date:
            raise serializers.ValidationError('Запис можливий тільки на
найближчі 14 днів.')
        return value

    def validate(self, data):
        try:
            doctor = Doctor.objects.get(pk=data['doctor_id'],
is_active=True)
        except Doctor.DoesNotExist:
            raise serializers.ValidationError({'doctor_id': 'Лікаря не
знайдено або він не приймає.'})

        is_booked = Appointment.objects.filter(
            doctor=doctor,
            date_time=data['date_time'],
            status__in=['planned', 'completed']
        ).exists()
        if is_booked:
            raise serializers.ValidationError({'date_time': 'Цей час вже
зайнятий. Оберіть інший.'})

        target_date = data['date_time'].date()
        day_of_week = target_date.weekday()
        try:
            schedule = DoctorSchedule.objects.get(
                doctor=doctor, day_of_week=day_of_week, is_working=True
            )
        except DoctorSchedule.DoesNotExist:
            raise serializers.ValidationError({'date_time': 'Лікар не
приймає в цей день тижня.'})

        appointment_time = data['date_time'].time()
```

```
            if appointment_time < schedule.work_start or appointment_time >=
schedule.work_end:
                raise serializers.ValidationError(
                    {'date_time': f'Лікар приймає з {schedule.work_start} до
{schedule.work_end}.'}
                )

            user = self.context['request'].user
            try:
                patient = user.patient_profile
            except ObjectDoesNotExist:
                patient = None

            if patient:
                planned_count = Appointment.objects.filter(
                    patient=patient,                                doctor=doctor,
status=Appointment.Status.PLANNED
                ).count()
                if planned_count >= 2:
                    raise serializers.ValidationError(
                        'Ви вже маєте 2 активні записи до цього лікаря.'
                    )
                same_date = Appointment.objects.filter(
                    patient=patient, doctor=doctor,
                    date_time__date=target_date,
status=Appointment.Status.PLANNED
                ).exists()
                if same_date:
                    raise serializers.ValidationError(
                        {'date_time': 'У вас вже є запис до цього лікаря на
цю дату.'}
                    )

            data['doctor'] = doctor
            return data

        def create(self, validated_data):
            validated_data.pop('doctor_id')
            user = self.context['request'].user
            try:
                patient = user.patient_profile
            except ObjectDoesNotExist:
                patient, _ = Patient.objects.get_or_create(user=user)
            return                Appointment.objects.create(patient=patient,
**validated_data)
```

ДОДАТОК В

Лістинг коду реєстрації та верифікації email

В.1 Серіалізатор реєстрації

```
      class RegisterSerializer(serializers.ModelSerializer):
          password = serializers.CharField(write_only=True, min_length=8)
          password_confirm = serializers.CharField(write_only=True)
          date_of_birth         =         serializers.DateField(required=False,
allow_null=True)
          gender    =    serializers.ChoiceField(choices=Patient.Gender.choices,
required=False, allow_blank=True)

          class Meta:
              model = User
              fields = [
                  'email', 'password', 'password_confirm',
                  'first_name', 'last_name', 'patronymic', 'phone',
                  'date_of_birth', 'gender'
              ]

          def validate_email(self, value):
              if User.objects.filter(email=value.lower()).exists():
                  raise  serializers.ValidationError('Користувач  з  таким  email
вже існує.')
              return value.lower()

          def validate(self, data):
              password = data['password']
              if data['password'] != data['password_confirm']:
                  raise         serializers.ValidationError({'password_confirm':
'Паролі не співпадають.'})
              if len(password) < 8:
                  raise  serializers.ValidationError({'password':  'Пароль  має
містити мінімум 8 символів.'})
              if not any(c.isalpha() for c in password):
                  raise  serializers.ValidationError({'password':  'Пароль  має
містити хоча б одну літеру.'})
              if not any(c.isdigit() for c in password):
                  raise  serializers.ValidationError({'password':  'Пароль  має
містити хоча б одну цифру.'})
              return data

          def create(self, validated_data):
              date_of_birth = validated_data.pop('date_of_birth', None)
              gender = validated_data.pop('gender', '')
              validated_data.pop('password_confirm')
              with transaction.atomic():
                  user    =    User.objects.create_user(role=User.Role.PATIENT,
**validated_data)
                  Patient.objects.get_or_create(
                      user=user,
```

```
                    defaults={'date_of_birth':    date_of_birth,    'gender':
gender or ''}
                )
            return user
```

В.2 Обробник верифікації email

```
    class VerifyEmailView(APIView):
        permission_classes = [AllowAny]

        def post(self, request):
            email = request.data.get('email', '').lower().strip()
            code = request.data.get('code', '').strip()

            try:
                user = User.objects.get(email=email)
            except User.DoesNotExist:
                return   Response({'error':   'Користувача   не   знайдено.'},
status=status.HTTP_404_NOT_FOUND)

            if user.email_verified:
                return  Response({'message':  'Email  вже  підтверджено.  Можете
увійти.'})

            if not user.email_verification_code:
                return Response(
                    {'error':   'Код   підтвердження   не   знайдено.   Запросіть
новий.'},
                    status=status.HTTP_400_BAD_REQUEST
                )
            if timezone.now() > user.email_verification_code_expires:
                return Response(
                    {'error': 'Код прострочений. Надішліть новий.'},
                    status=status.HTTP_400_BAD_REQUEST
                )
            if user.email_verification_code != code:
                return Response(
                    {'error': 'Невірний код підтвердження.'},
                    status=status.HTTP_400_BAD_REQUEST
                )

            user.email_verified = True
            user.email_verification_code = None
            user.email_verification_code_expires = None
            user.save(update_fields=['email_verified',
'email_verification_code', 'email_verification_code_expires'])

            refresh = RefreshToken.for_user(user)
            return Response({
                'message': 'Email успішно підтверджено!',
                'user': {
                    'id': user.id,
                    'email': user.email,
                    'full_name': user.get_full_name(),
                    'role': user.role,
                },
                'tokens': {
```

```
            'access': str(refresh.access_token),
            'refresh': str(refresh),
        }
    }, status=status.HTTP_200_OK)
```

ДОДАТОК Г

Екранні форми тестування системи

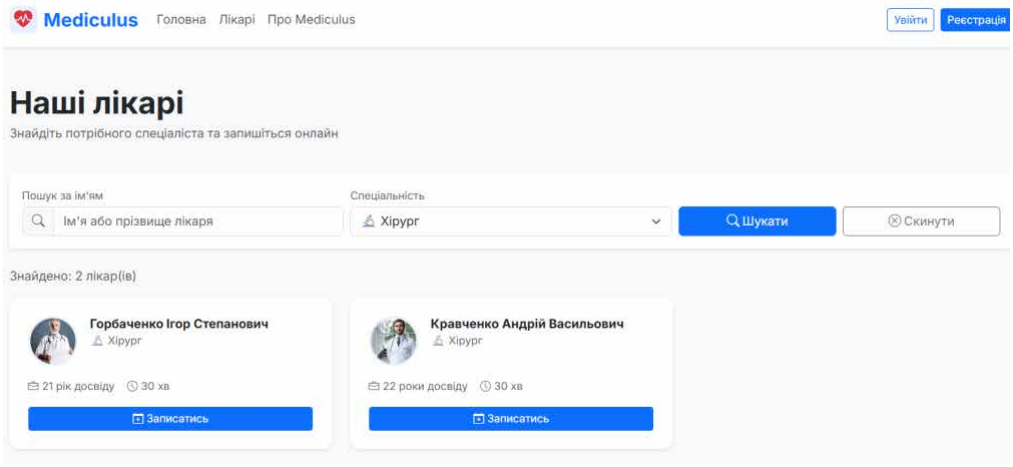

Рис. Г.1 Сторінка зі списком лікарів з фільтрацією за спеціальністю

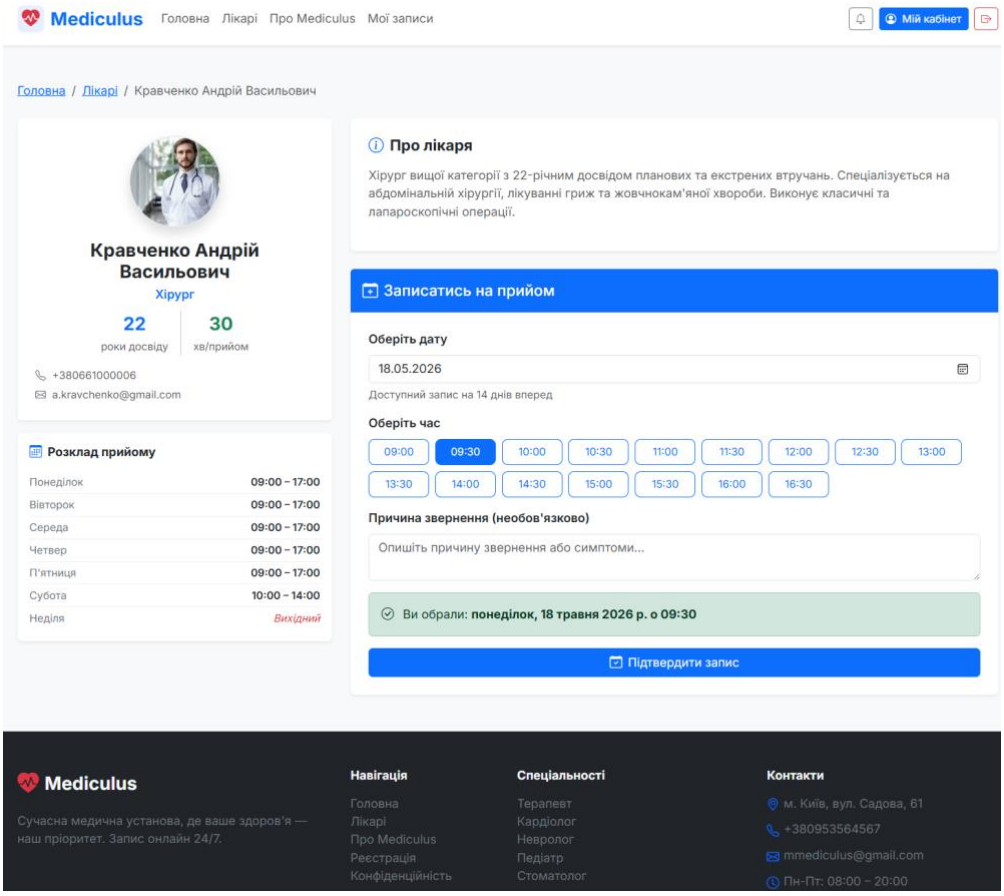

Рис. Г.2 Профіль лікаря з формою вибору дати і часових проміжків

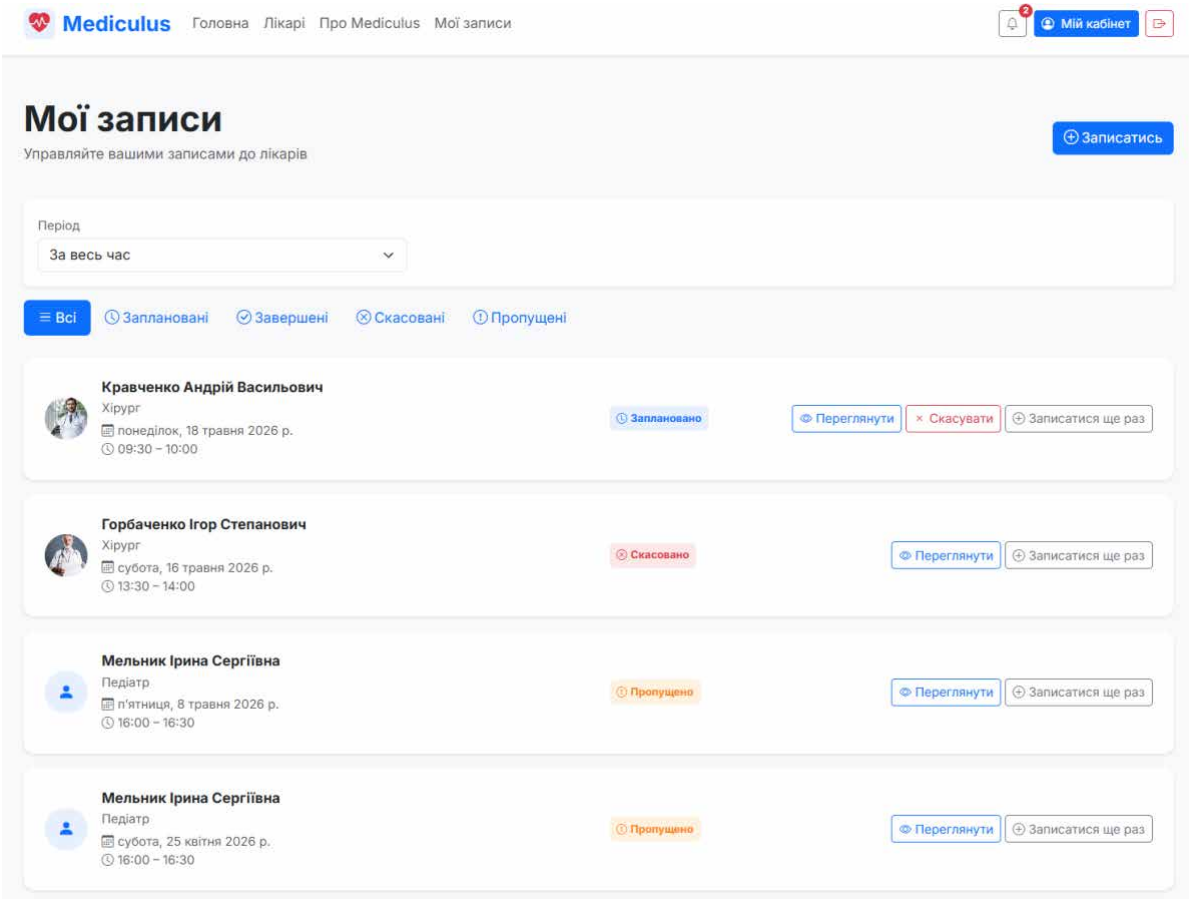

Рис. Г.3 Особистий кабінет пацієнта зі списком записів

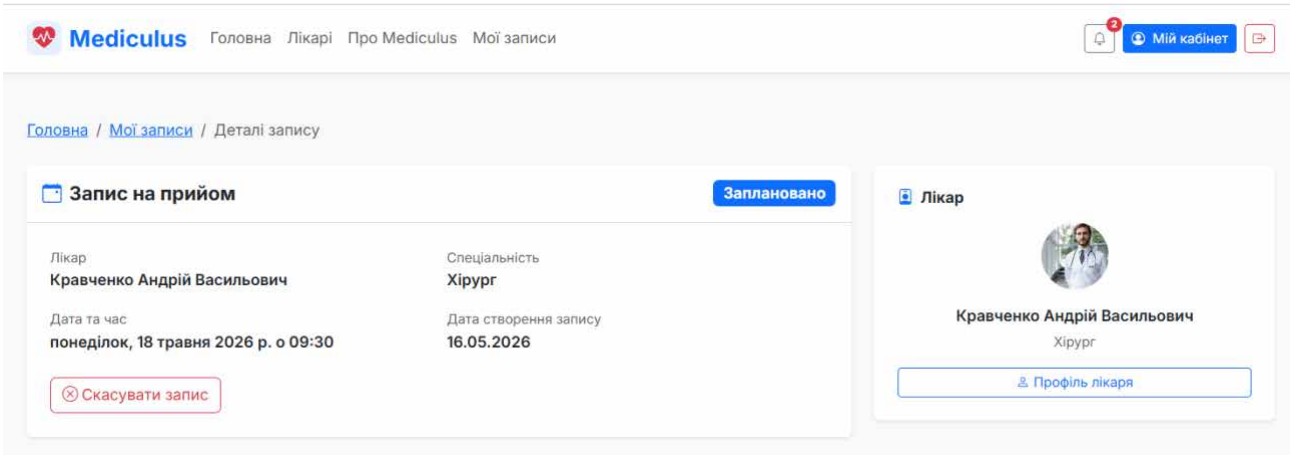

Рис. Г.4 Деталі запису зі статусом і кнопкою скасування

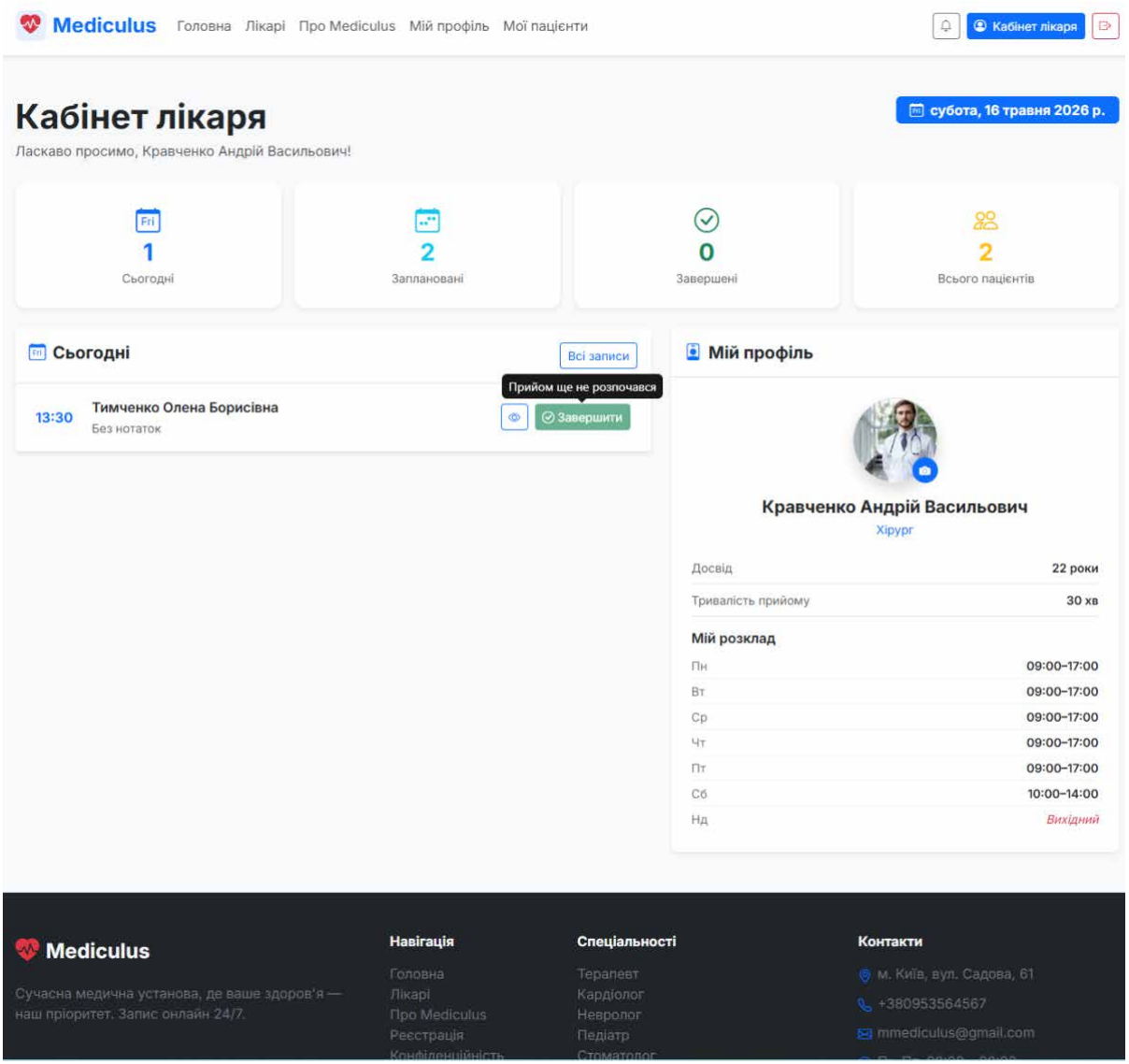

Рис. Г.5 Кабінет лікаря з статистикою і записами на сьогодні

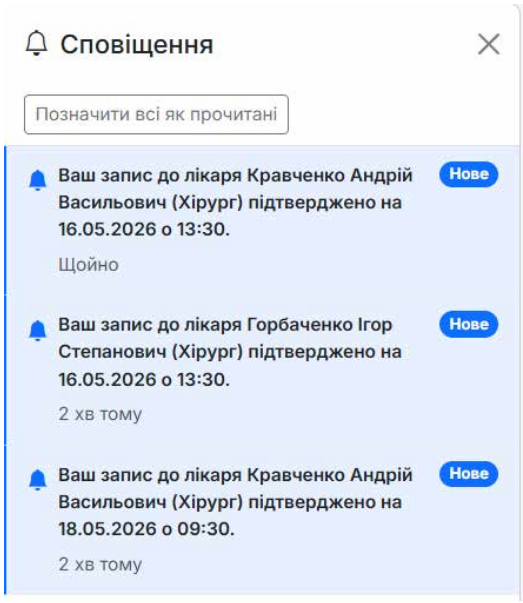

Рис. Г.6 Розділ сповіщень

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Денисенко Віталій Анатолійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення інформаційної системи для реєстрації пацієнтів лікарні» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [+] інструменти ШІ (Gemini та Claude) були використані для:
        - [+] перекладу іншомовних джерел;
        - [+] стилістичного редагування та перевірки граматики;
        - [+] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ **Віталій ДЕНИСЕНКО**
(підпис)