НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
(назва кафедри)

_____ Голуб Б.Л.
(підпис) (ПІБ)

"3" червня 2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення інформаційної системи для закладів охорони здоров'я»

Спеціальність 121 - «Інженерія програмного забезпечення»

Гарант освітньої програми

К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи

ст. викладач _____ Міловідов Ю.О.
(науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав: _____ Домашенко Ярослав Олегович
(підпис) (ПІБ студента)

КИЇВ - 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук
Голуб Б.Л.

" 16 " грудня 2024 р

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту

Домашенку Ярославу Олеговичу
(прізвище, ім'я, по батькові)

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення інформаційної системи для закладів охорони здоров'я

затверджена наказом ректора НУБіП України від " 16 " грудня 2024 р. №2249

Термін подання завершеної роботи на кафедру 2025.05.30
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи

Опис предмету дослідження, опис програмного забезпечення

Перелік питань, які потрібно розробити:

Системний аналіз предметної області

Аналіз предметної області

Розробка програмного забезпечення

Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання " 16 " грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи

ст. викладач		Міловідов Ю.О.
(науковий ступень та вчене звання)	(підпис)	(ПІБ)

Завдання прийняв до виконання _____ Домашенко Я.О.
(підпис) (ПІБ)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IT - Інформаційні технології

СУБД - Система управління базами даних

JS - JavaScript

БД - База даних

UML - Unified Modeling Language

ER - Entity-Relationship

LINQ - Language Integrated Query

SQL - Structured Query Language

API - Application Programming Interface

ORM - Object-Relational Mapping

CI/CD - Continuous Integration / Continuous Deployment

HTTP - HyperText Transfer Protocol

SMTP - Simple Mail Transfer Protocol

ВСТУП

У сучасному цифровому світі, де технології охоплюють практично всі сфери життя, галузь охорони здоров’я стикається з новими викликами. Зокрема, це зростання обсягів інформації, потреба в швидкому обміні даними та ефективному адмініструванні. Традиційні підходи до обробки медичної інформації вже не відповідають сучасним вимогам, що зумовлює потребу у впровадженні інноваційних інформаційних систем. Саме тому створення програмного забезпечення для автоматизації медичних процесів є надзвичайно актуальним.

Однією з головних проблем, що постають перед медичними інформаційними системами, є забезпечення безпечного, доступного та ефективного управління різноплановими даними - від інформації про пацієнтів і лікарів до медичних послуг, діагнозів і призначень. Важливою також є інтеграція з іншими електронними системами, простота адміністрування та надійне зберігання даних, зокрема файлів медичної документації. Розроблювана система має бути зручною для користувачів різних категорій, враховувати їхні потреби та відповідати сучасним стандартам безпеки.

Ця дипломна робота присвячена створенню інформаційної системи для автоматизації роботи медичного закладу. Розроблена система включає функціонал обліку користувачів і пацієнтів, планування процедур, управління розкладом лікарів, обробку електронних повідомлень, збереження медичних файлів та інструменти адміністрування. Проєкт реалізовано у вигляді веб-застосунку з окремими інтерфейсами для адміністратора, медичного персоналу та пацієнтів.

У реалізації застосовано сучасні технології: ASP.NET Core для побудови серверної частини, React разом із бібліотекою Material UI для створення інтерфейсу користувача, MinIO для організації зберігання файлів та i18next для підтримки багатомовності. Система також реалізує механізми аутентифікації, обміну повідомленнями та управління файлами відповідно до вимог безпеки і масштабованості.

У дипломному проєкті детально описано всі етапи створення програмної системи - від аналізу предметної області до демонстрації функціоналу. Структура роботи охоплює чотири основні розділи: аналіз проблемної сфери й вимог до системи, архітектурне проєктування, реалізацію функціоналу з використанням обраних технологій, а також тестування, вимоги до середовища та формування інсталяційного пакета.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасному світі охорона здоров’я є однією з критично важливих галузей, що потребує ефективної цифровізації для підвищення якості та доступності медичних послуг. Особливої актуальності набуває впровадження інформаційних систем, які дозволяють автоматизувати облік пацієнтів, керування розкладом лікарів, призначення процедур та збереження медичної інформації.

Предметною областю даного проєкту є діяльність медичного закладу, пов’язана з прийомом пацієнтів, веденням електронних медичних записів, управлінням медичним персоналом і послугами. У типовому медичному закладі до основних процесів належать:

1. Реєстрація пацієнтів, зберігання їхніх особистих і контактних даних;
2. Ведення електронної історії хвороби, включаючи діагнози, призначення, медичні файли;
3. Планування розкладу роботи лікарів і керування записами пацієнтів на прийом;
4. Управління медичними послугами, які можуть включати консультації, процедури, лабораторні дослідження;
5. Адміністрування системи - створення облікових записів, призначення ролей, контроль доступу.

Традиційно такі процеси в багатьох закладах охорони здоров’я здійснювалися в ручному режимі або з використанням застарілих локальних систем, які не дозволяють централізовано зберігати та обробляти дані, не мають веб-інтерфейсу і часто не підтримують багатокористувацький режим. Це призводить до втрати часу, неефективної взаємодії між пацієнтами та медичним персоналом, а також до проблем з безпекою та збереженням даних.

Сучасна інформаційна система повинна забезпечити:

1. Уніфіковану структуру зберігання даних пацієнтів, лікарів, послуг;

2. Можливість керування розкладом та призначеннями в режимі реального часу;
3. Автоматичну генерацію супровідної документації (листів, сповіщень, підтверджень запису);
4. Інтеграцію з іншими сервісами, зокрема електронною поштою та сховищами файлів;
5. Гнучкий контроль доступу з урахуванням ролей (адміністратор, лікар, пацієнт).

Таким чином, розробка програмної системи для автоматизації внутрішніх процесів медичного закладу є логічним рішенням для підвищення ефективності роботи персоналу, зниження навантаження на адміністративну частину та покращення якості обслуговування пацієнтів.

1.2 Аналіз вимог до програмного забезпечення

Розробка інформаційної системи для закладу охорони здоров’я передбачає чітке визначення вимог, що висуваються до функціональності, продуктивності, безпеки та інших характеристик програмного забезпечення. Аналіз вимог є ключовим етапом, який дозволяє сформувати цілісне бачення системи та уникнути помилок на подальших етапах проєктування і реалізації.

Бізнес-вимоги

Система має відповідати наступним стратегічним цілям:

1. **Оптимізація процесу запису на прийом** – спрощення взаємодії між пацієнтом і медичним персоналом через онлайн-платформу;
2. **Цифровізація медичних записів** – забезпечення зручного електронного доступу до історії хвороби, результатів обстежень, діагнозів та призначень;
3. **Підвищення якості обслуговування** – завдяки інтерфейсу для перегляду розкладу лікарів, доступу до інформації про послуги тощо;

4. **Інтеграція з медичними установами** – забезпечення можливості ведення медичних записів лікарями в електронній формі.

Функціональні вимоги

Для пацієнтів:

1. Реєстрація, вхід у систему та відновлення паролю;
2. Пошук лікарів за ПІБ, спеціалізацією, стажем;
3. Запис на прийом із вибором дати та часу;
4. Перегляд історії хвороби, результатів обстежень;
5. Скасування прийому;

Для лікарів:

1.Перегляд списку призначених прийомів;

2.Ведення електронної медичної картки пацієнта: додавання діагнозів, призначень, результатів аналізів;

3.Зміна та оновлення розкладу прийому.

Для адміністрації:

1. Управління обліковими записами користувачів;
2. Контроль за доступами і правами;
3. Моніторинг роботи системи.

Нефункціональні вимоги

1. Продуктивність: Інтерфейс повинен оперативно відповідати на дії користувача та забезпечувати швидке завантаження вмісту основних сторінок не повинно перевищувати 3 секунд.
2. Надійність: система повинна бути доступною 99.9% часу; дані мають зберігатися з можливістю аварійного відновлення.
3. Безпека:
    1. захист даних від несанкціонованого доступу;
    2. шифрування персональної інформації;
    3. захист від атак типу SQL Injection, XSS, CSRF.

4. Масштабованість: система має підтримувати розширення.
5. Зручність використання: Інтерфейс повинен бути простим у користуванні та не викликати складнощів у розумінні навіть у нових користувачів.

Архітектурні вимоги

На основі аналізу функціональних і нефункціональних вимог визначено потребу в реалізації системи у вигляді вебзастосунку з чітким розмежуванням ролей користувачів.

1.3 Моделювання предметної області

Моделювання предметної області є одним із ключових етапів процесу створення програмного забезпечення для медичних закладів, який дає змогу чітко сформулювати, систематизувати та формалізувати головні процеси, сутності та взаємозв’язки між ними в межах функціонування установи. Якісне моделювання дозволяє уникнути помилок на наступних етапах розробки, оскільки всі подальші дії - створення бази даних, написання програмного коду, реалізація бізнес-логіки - базуються саме на результатах цього етапу.

Предметна область системи охоплює комплекс процесів і взаємодій, пов’язаних із забезпеченням ефективного функціонування медичного закладу, і включає три ключові категорії користувачів: пацієнтів, лікарів та адміністративний персонал. Діяльність цих категорій організовується навколо таких ключових медичних процесів, як реєстрація та запис пацієнтів на прийом, створення та ведення медичної документації, встановлення діагнозів і призначення лікування, формування та адміністрування графіків роботи медичних працівників, а також надійне зберігання та управління історіями хвороб пацієнтів.

Основні сутності предметної області:

1. **Пацієнт** - фізична особа, зареєстрована в системі, яка має змогу записуватись на прийоми до лікарів, переглядати власну медичну інформацію, отримувати автоматичні нагадування та повідомлення щодо своїх прийомів та результатів лікування.
2. **Лікар** - медичний працівник, що має власний профіль у системі, відповідальний за ведення медичних записів, постановку діагнозів, внесення результатів обстежень, а також перегляд і редагування графіку своїх прийомів.
3. **Адміністратор** - співробітник з розширеними правами, який відповідає за управління системою, включаючи створення та редагування профілів користувачів (лікарів та пацієнтів), контроль доступу до конфіденційних даних, а також загальний моніторинг і технічне адміністрування програмного середовища.
4. **Прийом** - конкретна подія у вигляді консультації або огляду, яка має чітко визначені дату, час, пов’язана із конкретним пацієнтом та лікарем, і включає медичні записи, зроблені за результатами консультації.
5. **Медична картка** - структурована інформація про історію хвороби пацієнта, що включає діагнози, результати досліджень, призначені лікарські препарати та інші медичні записи, необхідні для комплексного аналізу стану пацієнта.

Основні процеси, які були формалізовані під час моделювання:

1. Запис пацієнтів на прийом до лікаря з урахуванням актуального графіку роботи та вільних часових слотів.
2. Ведення та перегляд історії хвороби пацієнта, включаючи діагнози, призначення, результати аналізів.

3. Створення лікарем нових медичних записів після завершення консультації з пацієнтом.
4. Формування та адміністрування графіків прийомів лікарів з можливістю гнучкого управління.
5. Перевірка та управління доступами до конфіденційної медичної інформації відповідно до прав користувачів.

У межах проведеного моделювання були створені наступні UML-діаграми, що використовуються для візуалізації моделі предметної області:

1. **Діаграма прецедентів** показує, як основні актори взаємодіють із системою, відображаючи її ключові функції та можливості.
2. **Діаграма діяльності** ілюструє логічний потік основних бізнес-процесів, детально описуючи, наприклад, процес запису пацієнта.
3. **Діаграма послідовності** детально відображає обмін повідомленнями між об'єктами системи під час виконання типових сценаріїв, таких як обробка та збереження медичних даних.

Усі ці моделі створено з використанням стандартної нотації UML та у відповідності до сучасних міжнародних стандартів моделювання інформаційних систем. У подальших розділах представлено розгорнутий опис кожної з моделей цієї роботи.

Результати цього моделювання допомогли чітко визначити функціональні зв'язки між різними частинами системи, заклавши основу для проектування бази даних та алгоритмів реалізації основних функцій у коді. Таким чином, цей етап став фундаментом для подальшої розробки, значно покращивши якість і надійність майбутнього програмного забезпечення.

1.4 Діаграма прецедентів

Діаграма прецедентів є візуальним інструментом, який допомагає відобразити взаємодію між користувачами системи та її функціональними

можливостями. Така діаграма дозволяє ще на етапі проектування зрозуміти, які основні дії можуть виконувати різні типи користувачів, і як ці дії пов’язані з підсистемами. Це важливо для узгодження вимог до програмного продукту та визначення його структури.

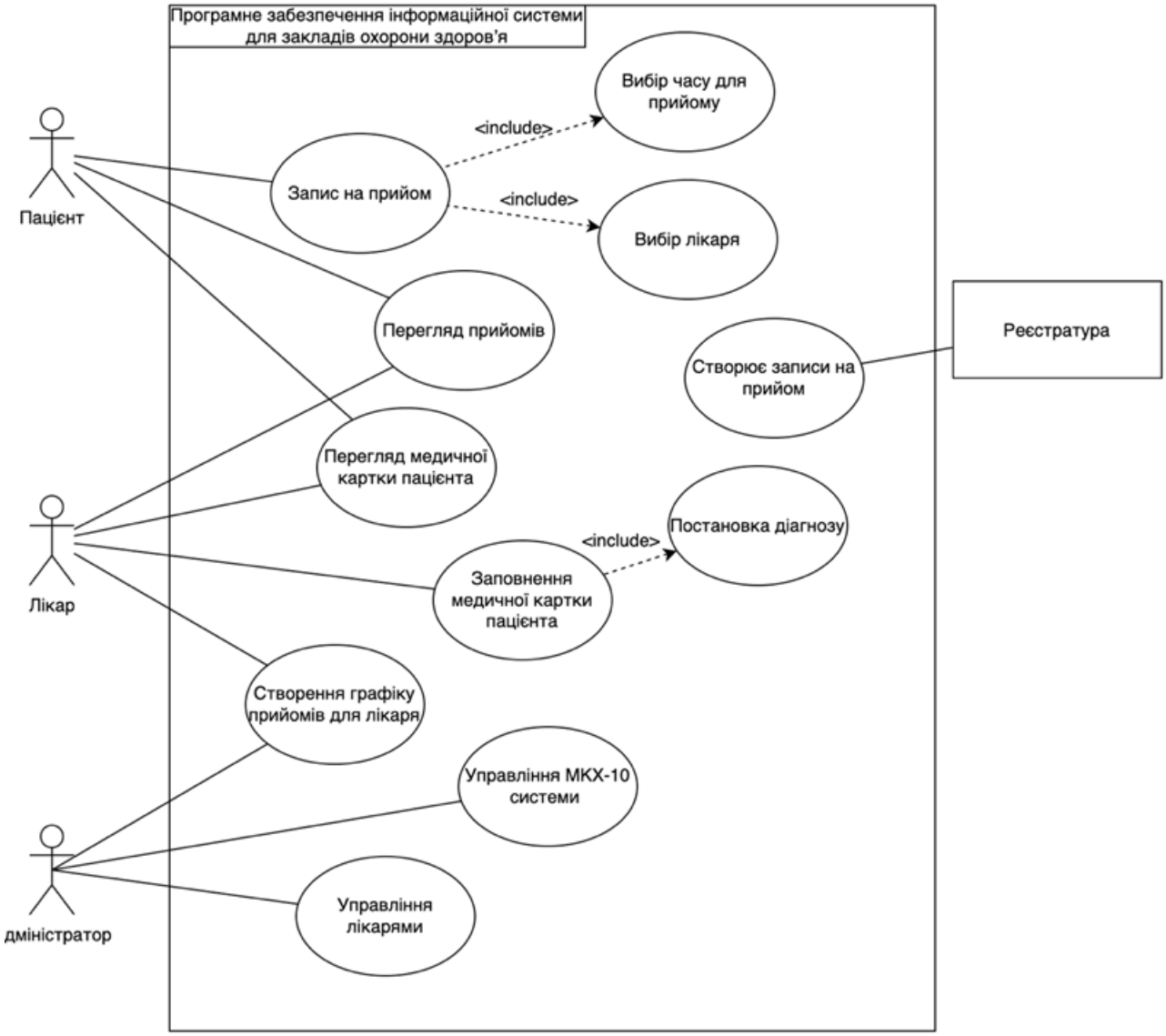


Рис 1. Діаграму прецедентів.

На рис. 1 зображено діаграму прецедентів розроблюваної медичної інформаційної системи. У моделі присутні три основні актори:

1. **Пацієнт** - має можливість записатися на прийом до лікаря, переглядати історію своїх прийомів і медичну картку.

2. **Лікар** - створює графік прийомів, переглядає прийоми та медичні картки пацієнтів, а також заповнює медичну інформацію, зокрема встановлює діагнози.
3. **Адміністратор** - відповідає за управління користувачами (лікарями) та класифікатором хвороб МКХ-10.

Основні прецеденти системи згруповані за ролями:

1. **Пацієнт** виконує запис на прийом, який включає вибір лікаря та часу. Після цього формується відповідний запис у реєстратурі.
2. **Лікар** працює з медичною карткою пацієнта, заповнюючи її, а також встановлюючи діагноз. Окрім того, він має можливість створювати особистий розклад прийомів.
3. **Адміністратор** здійснює керування списком лікарів і наповнення довідника МКХ-10, необхідного для стандартизації діагнозів.

Використання включень дозволяє відобразити обов'язкові події, які є складовими основних процесів, зокрема вибір лікаря чи постановку діагнозу як частину ширшого прецеденту.

1.5 Діаграма діяльності

Діаграма діяльності дозволяє змоделювати алгоритм виконання певної послідовності дій у межах системи. Це зручний інструмент для опису бізнес-процесів - зокрема, тих, які передбачають участь кількох користувачів або підсистем.

Загальна структура процесу дозволяє чітко розмежувати відповідальність: користувач ініціює дії, серверна логіка забезпечує обробку запитів, а база даних виконує збереження та перевірку даних. Це дає змогу ефективно спроєктувати архітектуру компонентів системи.

У межах розробки медичної інформаційної системи важливим функціональним процесом є запис пацієнта на прийом до лікаря. Цей процес має чітко визначену послідовність дій, що охоплює взаємодію користувача з

інтерфейсом, обробку запитів серверною частиною системи, а також оновлення інформації в базі даних.

Компоненти, що беруть участь:

1. **Пацієнт** - ініціює запит і взаємодіє з інтерфейсом;
2. **Реєстратура** - умовна роль, яка виконує логіку на рівні сервера;
3. **Контейнер бази даних** - обробляє запити на читання й запис даних;

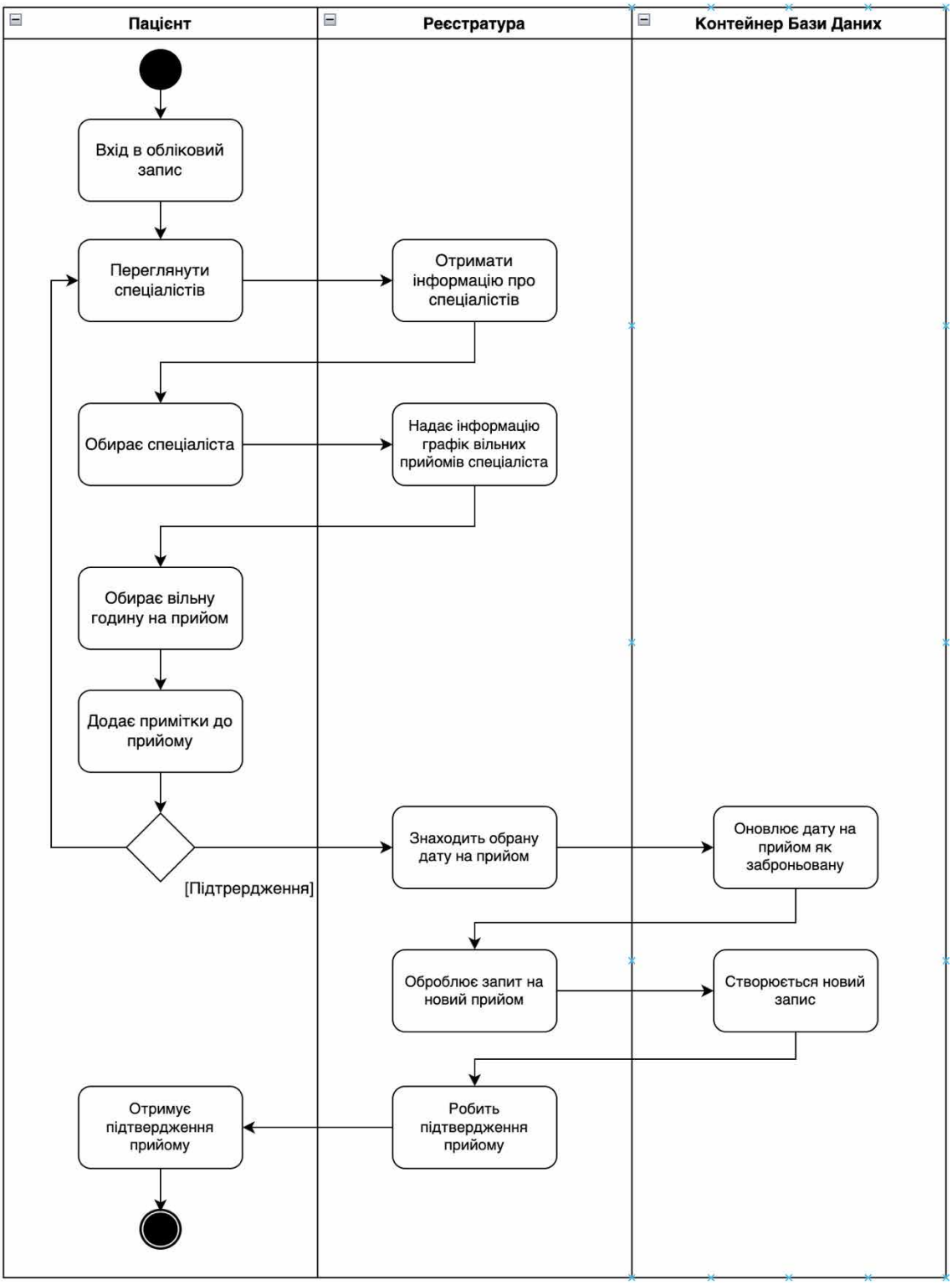


Рис. 2 Діаграма діяльності

На рис. 2 представлено діаграму діяльності, яка ілюструє весь цикл запису на прийом, розділений між трьома учасниками: Пацієнт, Реєстратура (сервісна логіка) та Контейнер бази даних.

Опис послідовності дій:

1. Пацієнт входить у свій обліковий запис через веб-інтерфейс.
2. Він переглядає список доступних спеціалістів, що ініціює відповідний запит до реєстратури.
3. Реєстратура обробляє запит і повертає інформацію про спеціалістів.
4. Пацієнт обирає спеціаліста, після чого система надає йому графік вільних годин прийому.
5. Пацієнт обирає зручний для нього час.
6. Він має можливість додати примітки до прийому - наприклад, опис симптомів або побажання щодо консультації.
7. Після цього система очікує підтвердження запису.

Якщо пацієнт підтверджує прийом:

8. Реєстратура перевіряє, чи обрана дата на той момент є вільною.
9. У випадку актуальності - вона змінює статус обраної години у контейнері бази даних на "забронbована".
10. У базі створюється новий запис про майбутній прийом пацієнта.
11. Після успішного створення запису система робить підтвердження прийому.
12. Пацієнт отримує підтвердження у вигляді повідомлення на сторінці або листа на електронну пошту.

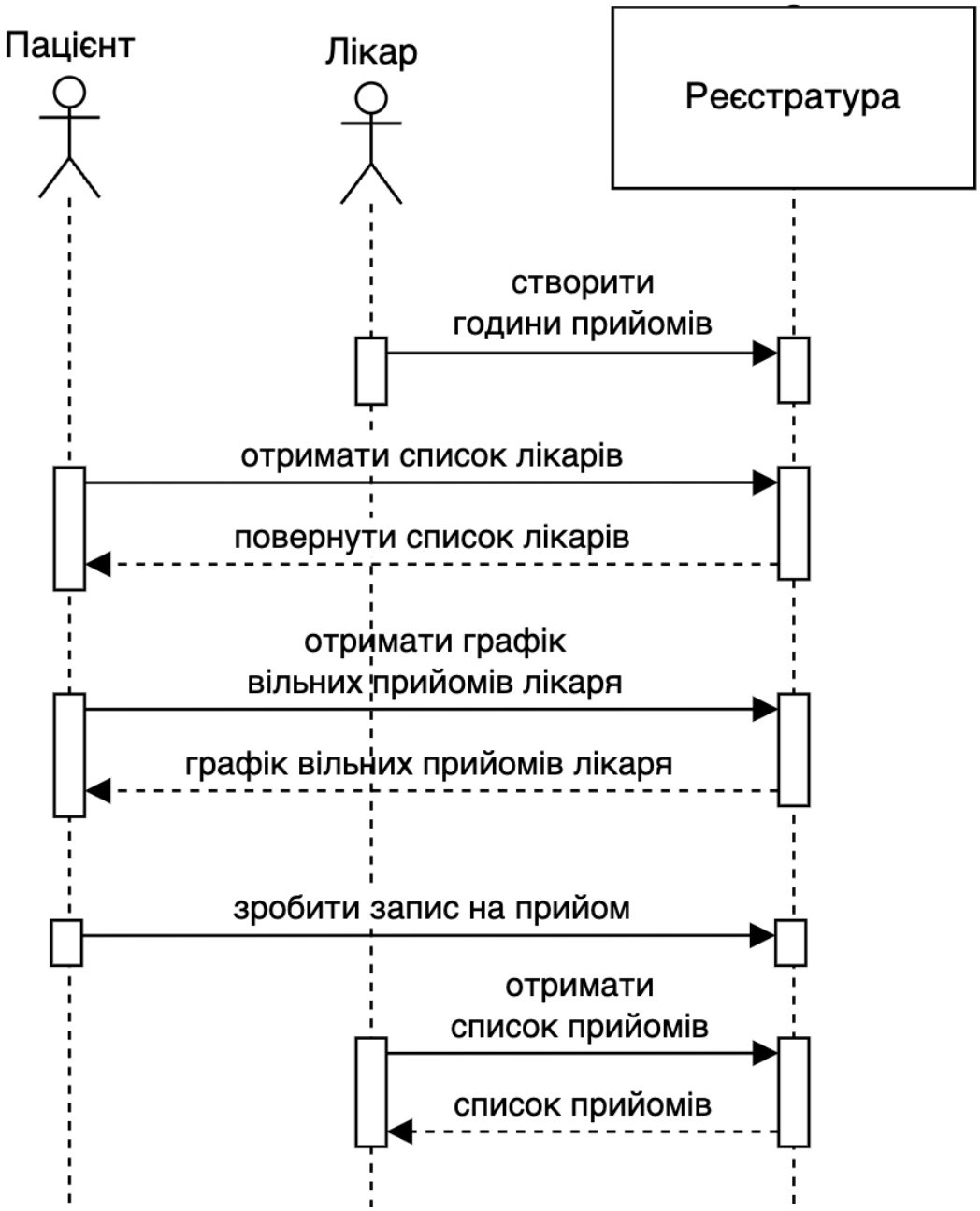
Ця діаграма дозволяє наочно відобразити всі основні кроки, розмежувати відповідальність між користувачем, бізнес-логікою і базою даних, а також забезпечити коректне технічне проектування функціоналу. Подальші етапи розробки системи базуються на цій моделі - вона слугує орієнтиром для розробників фронтенду, бекенду та для реалізації API-зв’язків між компонентами.

1.6 Діаграма послідовності

Діаграма послідовності ілюструє, як об’єкти системи взаємодіють між собою протягом виконання певного сценарію. У контексті медичної

інформаційної системи, прикладом може бути процес запису пацієнта на прийом.

У цьому проєкті діаграма послідовності ілюструє сценарій запису пацієнта до лікаря, з урахуванням побудови графіка доступних прийомів та



запитів до списку спеціалістів.

Рис. 3 Діаграма послідовності

На рис. 3 наведено відповідну діаграму, яка охоплює взаємодію між трьома головними учасниками: Пацієнтом, Лікарем та Реєстратурою (як логікою системи).

Опис взаємодії:

1. Лікар ініціює процес створення власного розкладу, надсилаючи запит до системи - створюються години прийомів, доступні для запису пацієнтів.
2. Пацієнт надсилає запит на отримання списку лікарів. Система обробляє запит і повертає відповідь із доступним переліком спеціалістів.
3. Після вибору лікаря пацієнт надсилає запит на отримання графіка вільних прийомів, який формується на основі розкладу, створеного лікарем.
4. Система повертає пацієнту актуальний графік прийомів.
5. Пацієнт обирає зручну дату і надсилає запит на запис на прийом, що передається до реєстратури.
6. Реєстратура обробляє запит і додає його до бази прийомів.
7. На завершення, пацієнт отримує підтвердження, здійснюючи запит на список прийомів, і отримує оновлений список прийомів, який вже містить щойно створений запис.

Ця діаграма наочно демонструє послідовність дій у процесі взаємодії між користувачем, медичним персоналом і логікою системи. Вона допомагає краще зрозуміти, які API-запити та сервіси потрібно реалізувати, як здійснюється асинхронна взаємодія, і де можуть виникати потенційні точки відмови або необхідність у валідації.

Діаграма послідовності є важливою частиною проєктної документації, оскільки забезпечує чітке уявлення про логіку обміну повідомленнями між об’єктами у часі, і може бути використана як основа для створення RESTful API, контролерів у серверному коді та сценаріїв інтеграційного тестування.

1.7 Аналіз існуючих рішень

До початку проєктування власної системи доцільно дослідити вже наявні програмні рішення, що використовуються в медичній сфері. Це дозволяє виявити переваги, недоліки та ключові функціональні компоненти, які користуються попитом серед медичних закладів та пацієнтів. На основі такого аналізу можна сформувати вимоги до розроблюваної системи, орієнтуючись на найкращі практики та уникаючи типових недоліків конкурентів.

У процесі дослідження було розглянуто три програмні продукти - Helsi, Patient Access та Docplanner. Вони охоплюють широкий спектр функцій для автоматизації медичних процесів, однак кожен має власні обмеження і специфіку.

Helsi (Україна, ТОВ «Хелсі ЮА»)

Helsi - українська платформа для електронної медицини, що активно використовується державними та приватними закладами. Вона має повноцінний функціонал для запису на прийом, ведення електронної медичної картки, виписки рецептів, спілкування між лікарем і пацієнтом, CRM-функціонал для медичних центрів.

Переваги:

- Інтуїтивно зрозумілий інтерфейс;
- Мобільний додаток;
- Можливість керування персоналом, пацієнтами, аптечними запасами;
- Підтримка повідомлень та зворотного зв’язку.

Недоліки:

- Складна інтеграція з іншими системами;
- Обмеження по доступності функцій без підтвердженого акаунта.

Patient Access (Велика Британія, NHS Digital)

Patient Access - онлайн-сервіс для пацієнтів Національної служби здоров’я Великої Британії (NHS). Дає змогу записатися на прийом, отримати рецепт, зв’язатися з лікарем, здійснити оплату послуг.

Переваги:

- Пряме підключення до NHS-сервісів;
- Зручний мобільний додаток;
- Проста авторизація.

Недоліки:

- Доступний лише громадянам Великої Британії;
- Недоступний українським користувачам.

Docplanner (Docplanner Group)

Docplanner - міжнародна платформа для пошуку та бронювання прийомів до лікарів у понад 25 країнах. Сервіс включає функції онлайн-консультацій, медичних карток, рейтингу лікарів, електронних рецептів тощо.

Переваги:

- Широке міжнародне покриття;
- Онлайн-консультації;
- Мобільний додаток;
- Зручний та адаптивний інтерфейс.

Недоліки:

- Недоступність в Україні;
- Частина функціоналу орієнтована на приватну медицину Західної Європи.

Порівняльна таблиця:

Функціонал	**Helsi**	**Patient Access**	**Docplanner**
Запис на прийом	+	+	+
Онлайн-спілкування з лікарем	+	+	–
Електронні рецепти	+	+	+

Функціонал	Helsi	Patient Access	Docplanner
Ведення медичної картки	+	+	+
Мобільний додаток	+	+	+
Оплата медичних послуг	+	+	+
Підтримка українських закладів	+	–	–

Висновки з аналізу

На основі проведеного порівняння можна визначити, що більшість існуючих рішень охоплюють базовий функціонал, пов’язаний із записом до лікаря, веденням картки пацієнта та електронними рецептами. Водночас, жодна з платформ не є повністю відкритою для приватного використання в Україні, окрім Helsi.

Це дозволяє сформулювати наступні ключові завдання для власної системи:

- Бути універсальною платформою для приватних клінік в Україні;
- Підтримувати гнучку рольову модель (пацієнт, лікар, адміністратор);
- Забезпечувати повноцінний цикл обслуговування пацієнта: від запису до прийому до ведення медичних записів;
- Бути інтегровною з іншими сервісами через REST API;
- Мати веб-інтерфейс і мобільну адаптацію;
- Підтримувати резервне копіювання та шифрування даних для відповідності вимогам безпеки.

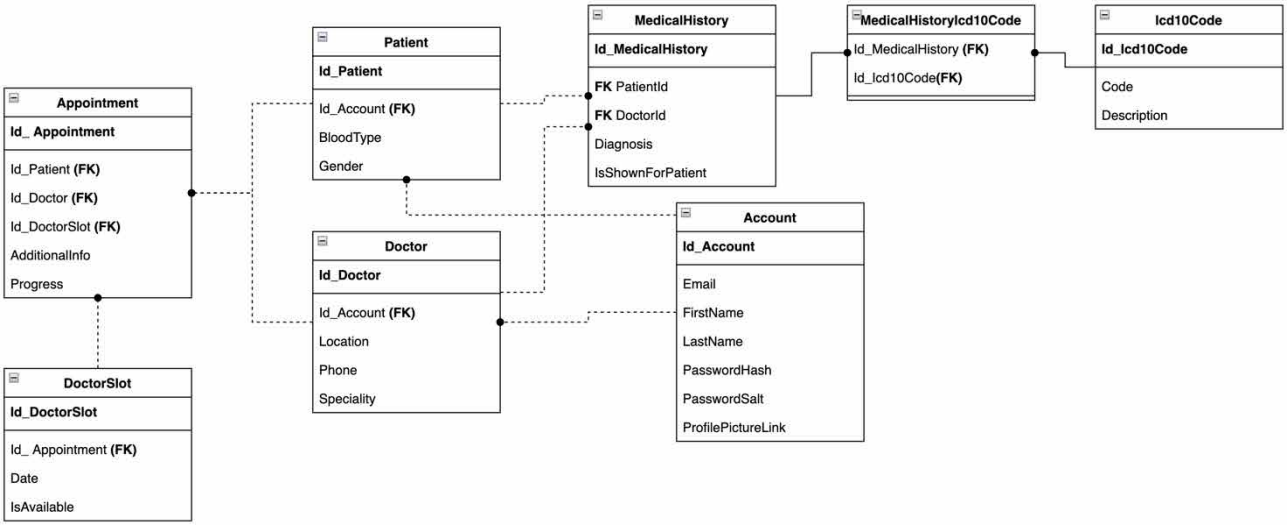
2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

ER-діаграма - це візуальне представлення логічної моделі бази даних, яке описує основні об’єкти, їх характеристики та взаємозв’язки. Вона використовується для формалізації структури даних предметної області та слугує основою при побудові реляційної бази даних.

ER-модель відображає:

1. Сутності (таблиці) - прямокутники з назвою та атрибутами;
2. Зв’язки між сутностями - стрілки або лінії з позначенням зв’язків (один-до-одного, один-до-багатьох тощо);



3. Ключі - первинні (PK) і зовнішні (FK).

Рис. 4 ER-діаграма

На рис. 4 представлена логічна ER-модель даних медичної інформаційної системи, яка реалізує зберігання інформації про користувачів, прийоми, медичну історію та розклад лікарів.

Опис основних сутностей:

1. Account - сутність, що зберігає облікові дані користувачів системи (як пацієнтів, так і лікарів).

1.1. Id_Account (PK) - унікальний ідентифікатор.

1.2. Email, FirstName, LastName - основні атрибути облікового запису.

1.3. PasswordHash, PasswordSalt - поля для безпечного зберігання пароля.

1.4. ProfilePictureLink - посилання на аватар користувача.

2. Patient - представляє користувача з роллю “пацієнт”.

2.1. Id_Patient (PK) - унікальний ідентифікатор пацієнта.

2.2. Id_Account (FK) - зв’язок з обліковим записом.

2.3. BloodType, Gender - додаткові медичні характеристики.

3. Doctor - представляє користувача з роллю “лікар”.

3.1. Id_Doctor (PK) - унікальний ідентифікатор лікаря.

3.2. Id_Account (FK) - зв’язок з обліковим записом.

3.3. Location, Phone, Speciality - адреса прийому, контакт і спеціалізація.

4. Appointment - сутність, що зберігає інформацію про конкретні прийоми пацієнтів.

4.1. Id_Appointment (PK) - ідентифікатор прийому.

4.2. Id_Patient (FK), Id_Doctor (FK) - зв’язки з пацієнтом і лікарем.

4.3. Id_DoctorSlot (FK) - посилання на час прийому.

4.4. AdditionalInfo - додаткові побажання пацієнта.

4.5. Progress - статус прийому (наприклад, “заплановано”, “завершено”).

5. DoctorSlot - визначає конкретні слоти часу, в які лікар приймає пацієнтів.

5.1. Id_DoctorSlot (PK) - ідентифікатор слоту.

5.2. Id_Appointment (FK) - прив’язка до запису (може бути null, якщо слот ще вільний).

5.3. Date - час прийому.

5.4. IsAvailable - ознака, чи доступний слот.

6. MedicalHistory - зберігає медичні записи, створені лікарем для конкретного пацієнта.

6.1. Id_MedicalHistory (PK) - ідентифікатор запису.

6.2. PatientId (FK), DoctorId (FK) - зв’язки з пацієнтом і лікарем.

6.3. Diagnosis - текстовий опис діагнозу.

6.4. IsShownForPatient - позначка, чи пацієнт бачить цей запис.

7. Icd10Code - таблиця з класифікатором МКХ-10 (Міжнародна класифікація хвороб).

7.1. Id_Icd10Code (PK) - ідентифікатор коду.

7.2. Code - код захворювання (наприклад, J45.9).

7.3. Description - опис діагнозу.

8. MedicalHistoryIcd10Code - проміжна таблиця для реалізації зв’язку “багато-до-багатьох” між MedicalHistory і Icd10Code.

8.1. Id_MedicalHistory (FK)

8.2. Id_Icd10Code (FK)

Ця структура забезпечує нормалізацію бази даних, логічне розділення ролей і чітке розмежування доступу до даних. ER-модель послужила основою для фізичного моделювання бази даних у системі управління БД.

2.2 Діаграма абстракцій

Діаграма абстракцій є узагальненим представленням предметної області в термінах об’єктно-орієнтованого підходу. Вона використовується для попереднього формування структури класів системи, опису їхніх основних властивостей та обов’язків, без заглиблення у технічні деталі реалізації. Така діаграма є проміжним кроком між логічною ER-моделлю та UML-діаграмою класів і допомагає перейти від структур бази даних до об’єктної моделі програмного забезпечення.

На рис. 5 зображено діаграму абстракцій розроблюваної медичної інформаційної системи. Вона включає ключові класи: Пацієнт, Лікар, Адміністратор, Прийом, Графік прийомів, Медична картка, Класифікатор МКХ-10. Для кожного з них наведено основні атрибути (властивості) та відповідні обов’язки (відповідальності в системі).

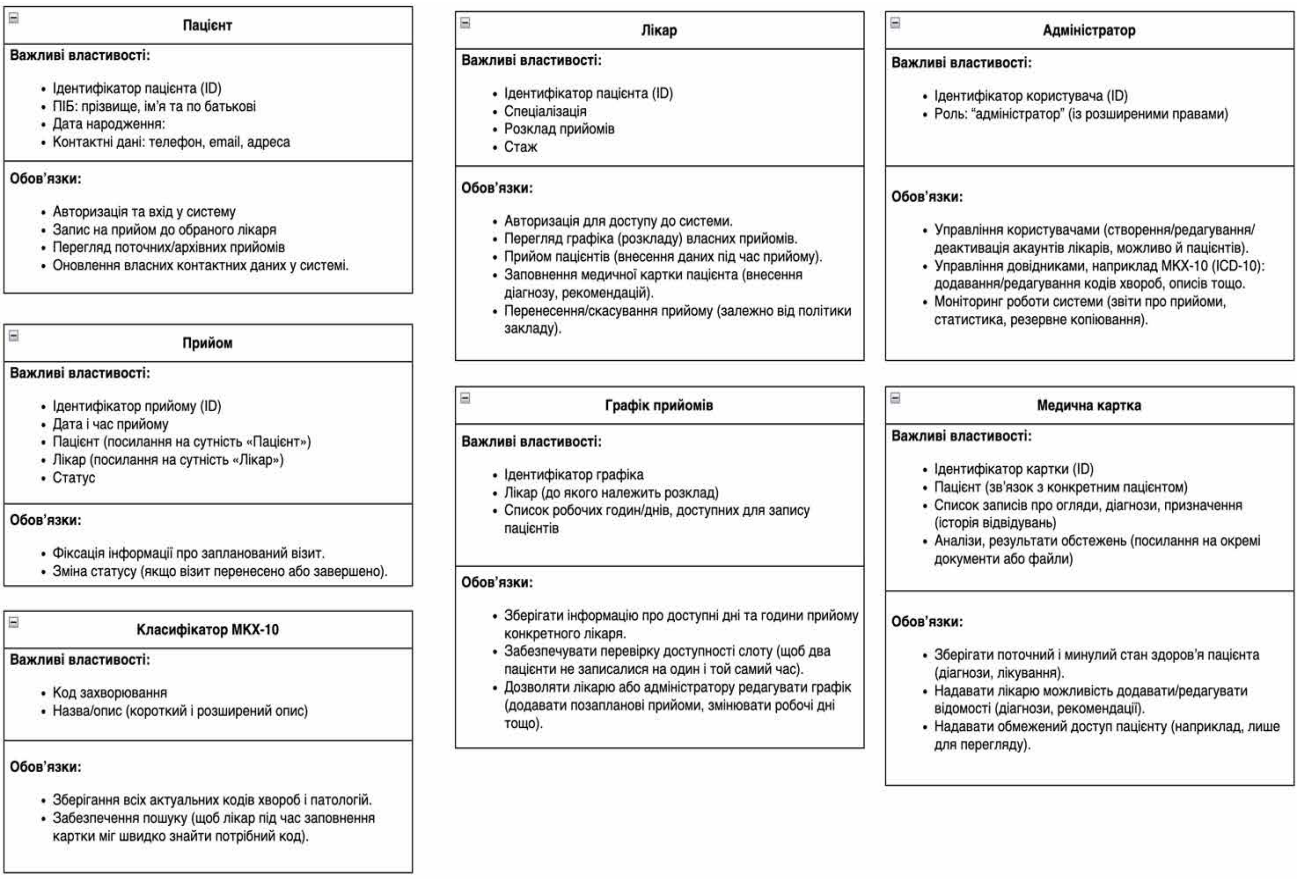


Рис. 5 Діаграма абстракцій

Короткий опис кожної абстракції:

1. **Пацієнт**

• Важливі властивості: Ідентифікатор, ПІБ, дата народження, контактна інформація.

• Обов’язки: Авторизація, запис на прийом, перегляд прийомів, оновлення профілю.

2. **Лікар**

• Важливі властивості: Ідентифікатор, спеціалізація, розклад прийомів, стаж.

• Обов’язки: Вхід у систему, ведення медичної картки, управління прийомами, перенесення/скасування.

3. **Адміністратор**

• Важливі властивості: Ідентифікатор користувача, роль з розширеними правами.

• Обов’язки: Управління користувачами, довідниками, статистикою та резервним копіюванням.

4. **Прийом**

• Важливі властивості: Ідентифікатор, дата і час, посилання на пацієнта і лікаря, статус.

• Обов’язки: Фіксація і зміна статусу прийому.

5. **Графік прийомів**

• Важливі властивості: Ідентифікатор графіка, лікар, список робочих днів/годин.

• Обов’язки: Визначення доступності, редагування графіку, уникнення конфліктів при записі.

6. Медична картка

• Важливі властивості: Ідентифікатор, зв’язок із пацієнтом, записи про огляди, діагнози, аналізи.

• Обов’язки: Зберігання історії здоров’я, додавання/редагування записів, обмеження доступу для пацієнтів.

7. Класифікатор МКХ-10

• Важливі властивості: Код хвороби, опис.

• Обов’язки: Зберігання актуальних кодів, пошук під час діагностування.

Ця діаграма дозволяє розробнику на початковому етапі візуалізувати головні сутності та функціональні зони відповідальності кожної ролі в системі. Вона також формує основу для подальшого проєктування класів, інтерфейсів та логіки взаємодії між об’єктами.

2.3 Діаграма класів

Зобразивши об'єкти системи як класи з їхніми властивостями, методами/операціями та різними типами зв'язків, діаграма класів слугує важливим засобом для моделювання систем, використовуючи нотацію UML.

Основною метою діаграми класів є формалізація структури об’єктної моделі майбутньої системи, що дозволяє чітко визначити складові програмного забезпечення на етапі проєктування, спростити подальшу реалізацію та тестування.

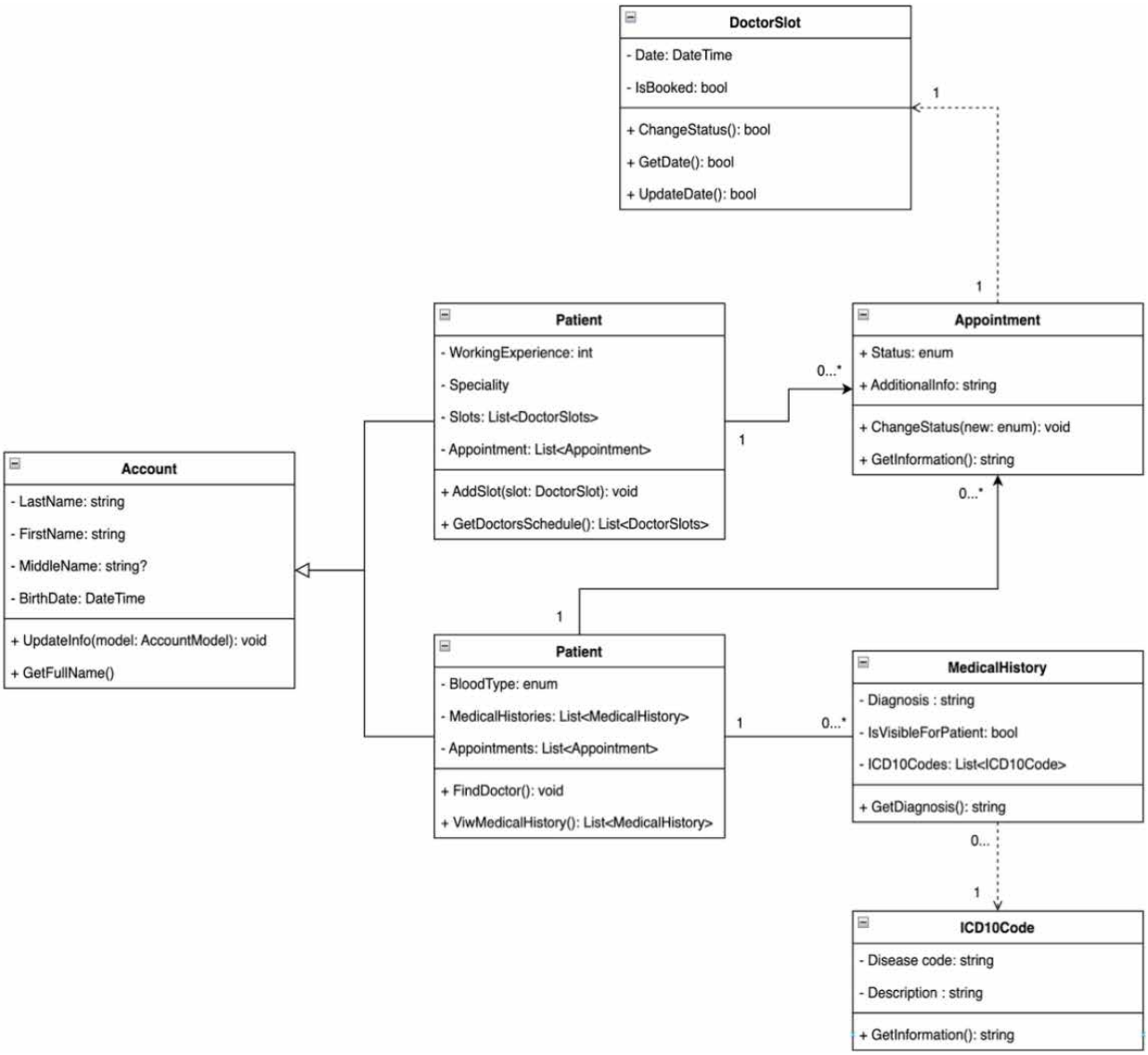


Рис. 6 Діаграма класів

На рис. 6 зображено діаграму класів медичної інформаційної системи, побудовану відповідно до логічної структури бази даних і абстракцій предметної області.

Опис основних класів:

1. **Account** - є базовим класом, від якого успадковуються функціональні класи користувачів - Пацієнт і Лікар.

• Атрибути: LastName, FirstName, MiddleName (опціонально), BirthDate.

• Методи: UpdateInfo(model: AccountModel): void - оновлення інформації облікового запису. GetFullName(): string - отримання повного імені користувача.

2. **Patient** - має можливість записуватися на прийоми, переглядати свою історію лікування та зберігати результати прийомів.

• Атрибути: BloodType, колекції MedicalHistories та Appointments.

• Методи: FindDoctor(): void - пошук лікаря. ViewMedicalHistory(): List<MedicalHistory> - перегляд своєї медичної історії.

3. **Doctor**

• Атрибути: WorkingExperience, Speciality, Slots (розклад прийомів), Appointments.

• Методи: AddSlot(slot: DoctorSlot): void - додавання нового слота в розклад. GetDoctorsSchedule(): List<DoctorSlot> - отримання розкладу прийомів.

4. **DoctorSlot** - слоти відображають доступні години лікарів для запису пацієнтів

• Атрибути: Date (дата і час прийому), IsBooked (заброньований чи ні).

• Методи: ChangeStatus(): bool - змінити статус слоту. GetDate(): DateTime - отримати дату слоту. UpdateDate(): bool - оновити час прийому.

5. **Appointment** - кожен прийом має конкретного лікаря, пацієнта та час проведення

• Атрибути: Status (стан прийому), AdditionalInfo (примітки пацієнта).

• Методи: ChangeStatus(new: enum): void - змінити статус прийому. GetInformation(): string - отримати інформацію про прийом.

6. **MedicalHistory** - медична історія фіксує дані про обстеження пацієнта, включаючи встановлені діагнози.

• Атрибути: Diagnosis (опис діагнозу), IsVisibleForPatient (доступність пацієнту).

• Методи: GetDiagnosis(): string - отримати інформацію про діагноз.

7. **ICD10Code** - класифікатор МКХ-10 дозволяє стандартизувати медичні діагнози, що використовуються в картках пацієнтів.

• Атрибути: DiseaseCode (код захворювання), Description (опис).

• Методи: GetInformation(): string - отримати інформацію про код захворювання.

Огляд зв’язків:

1. Клас Patient має зв’язки з класами Appointment та MedicalHistory типу “один до багатьох” (1 → 0..*).
2. Клас Appointment асоційований з DoctorSlot та MedicalHistory через зв’язки з відповідними об’єктами.
3. Клас MedicalHistory має асоціацію з множиною об’єктів ICD10Code, що відображає використання кількох кодів для одного запису.

Усі зв’язки логічно відповідають реальним процесам у медичному закладі: запис на прийом, ведення медичної історії, облік вільного часу лікаря.

2.4 Діаграма пакетів

Діаграма пакетів в UML описує логічну організацію програмного забезпечення на рівні модулів (пакетів) та залежності між ними. Вона дозволяє структурувати код, розділити відповідальності та забезпечити високу модульність і масштабованість розроблюваної системи.

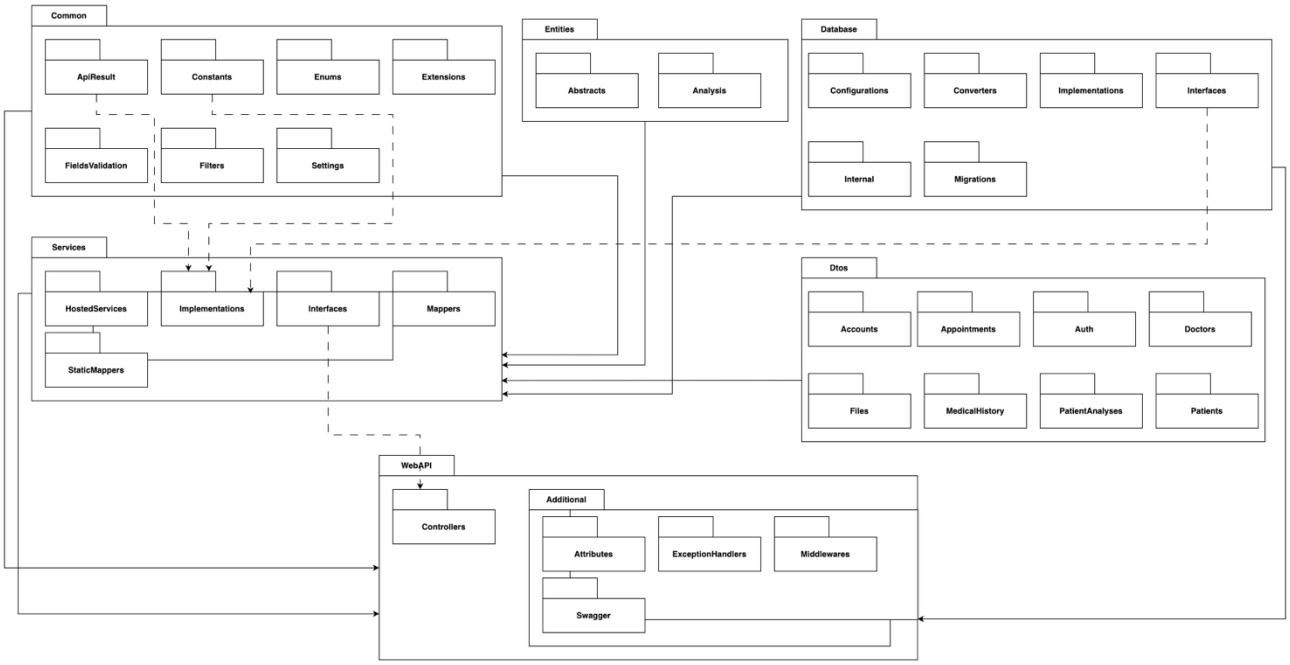


Рис. 7 Діаграма пакетів

На рис. 7 представлена діаграма пакетів медичної інформаційної системи, що реалізована на базі ASP.NET Core. Вона відображає основні компоненти додатку, їхню взаємодію та організацію внутрішніх підсистем.

Опис основних пакетів:

Common - пакет загального призначення:

1. ApiResult - стандартна форма відповіді на запити.
2. Constants, Enums - збереження констант та перерахувань, що використовуються у різних частинах системи.
3. Extensions - розширення функціональності існуючих типів.
4. FieldsValidation, Filters, Settings - засоби валідації даних, фільтрації запитів та зберігання налаштувань.

Entities - пакет, що містить основні бізнес-сутності:

1. Abstracts - базові абстрактні класи для побудови моделей.
2. Analysis - сутності для представлення результатів аналізів пацієнтів.

Database - пакет для взаємодії з базою даних:

1. Configurations - конфігурації сутностей для ORM (Entity Framework Core).
2. Converters - перетворювачі специфічних типів даних.
3. Implementations, Interfaces - реалізації та контракти доступу до бази даних.
4. Internal, Migrations - службові файли для міграцій та допоміжних операцій.

Dtos -пакет об’єктів передачі даних (Data Transfer Objects), що використовуються для обміну інформацією між клієнтом і сервером:

1. Accounts, Appointments, Auth, Doctors, Files, MedicalHistory, PatientAnalyses, Patients - групування DTO за контекстами.

Services - основний бізнес-шар системи:

1. Interfaces - контракти для сервісів.
2. Implementations - реалізації бізнес-логіки.
3. HostedServices - фонові служби.
4. Mappers, StaticMappers - логіка перетворення сутностей у DTO та навпаки.

WebAPI - пакет, що відповідає за обробку вхідних HTTP-запитів:

1. Controllers - контролери, що реалізують API-ендпоїнти для взаємодії з клієнтом.
2. Attributes - атрибути для розширення поведінки API-контролерів або методів.

3. ExceptionHandlers - обробники помилок.
4. Middlewares - проміжні обробники запитів.
5. Swagger - інтеграція документування API.

Огляд залежностей між пакетами:

1. WebAPI безпосередньо залежить від Services, Dtos та Common.
2. Services взаємодіють з Entities та Database для реалізації бізнес-логіки.
3. Dtos використовується для серіалізації та передачі даних між шарами.
4. Common надає допоміжні інструменти для всієї системи.
5. Database реалізує зберігання і доступ до даних.
6. Additional розширює можливості WebAPI через middleware, swagger-конфігурацію та глобальні обробники помилок.

Такий розподіл дозволяє забезпечити високу ізоляцію модулів, що сприяє простоті тестування, гнучкості та підтримці системи.

2.5 Діаграма компонентів

Діаграма компонентів використовується для моделювання фізичної архітектури системи, відображення основних складових програмного забезпечення (компонентів) та взаємодії між ними. Вона описує, як програмні модулі об’єднані для формування повного рішення.

Опис основних компонентів:

Web API Service.exe - це виконуваний компонент, відповідальний за прийом та обробку HTTP-запитів через контролери, які виступають точками доступу для користувачів:

1. AuthController - обробка автентифікації користувачів.
2. AccountsController - керування обліковими записами.
3. AppointmentsController - обробка записів на прийом.
4. DoctorsController - робота з інформацією про лікарів.

5. MedicalHistoryController - робота з медичними записами пацієнтів.
6. PatientsController - обробка інформації про пацієнтів.

Кожен контролер напряму викликає відповідний сервіс для обробки бізнес-логіки.

Services.dll - бібліотека бізнес-логіки. Складається з двох частин:

1. Implementations (реалізації сервісів):
2. AuthService, AccountsService, AppointmentsService, DoctorsService, MedicalHistoryService, PatientsService - сервіси, що реалізують основну бізнес-логіку відповідно до своїх доменних областей.
3. Interfaces (контракти сервісів):
4. IAuthService, IAccountsService, IAppointmentsService, IDoctorsService, IMedicalHistoryService, IPatientsService - інтерфейси, які визначають абстрактні методи для відповідних сервісів.

Контролери взаємодіють з інтерфейсами сервісів через механізм залежностей (Dependency Injection).

DatabaseContext.dll - бібліотека доступу до бази даних.

1. Implementations: DatabaseContext - клас, що забезпечує взаємодію з базою даних за допомогою технології Entity Framework Core.
2. Interfaces: DbContext - абстрактний базовий клас, що визначає базові операції роботи з даними. IUnitOfWork - інтерфейс патерну “Одиниця роботи”, який дозволяє об’єднувати кілька операцій над базою даних в одну транзакцію.

Сервіси використовують DatabaseContext для збереження та отримання даних, через абстракцію IUnitOfWork, що забезпечує більшу контрольованість і тестованість коду.

Огляд взаємодії компонентів:

1. Клієнт надсилає HTTP-запит, який потрапляє до відповідного контролера в Web API Service.
2. Контролер викликає методи з відповідного сервісу через інтерфейс, визначений у Services.dll.
3. Сервіси взаємодіють з базою даних через DatabaseContext.dll, використовуючи патерн “Одиниця роботи” та ORM Entity Framework Core.

Така структура реалізує принципи розділення обов’язків (Separation of Concerns) та інверсії залежностей (Dependency Inversion Principle), що підвищує гнучкість, розширюваність і тестованість архітектури системи.

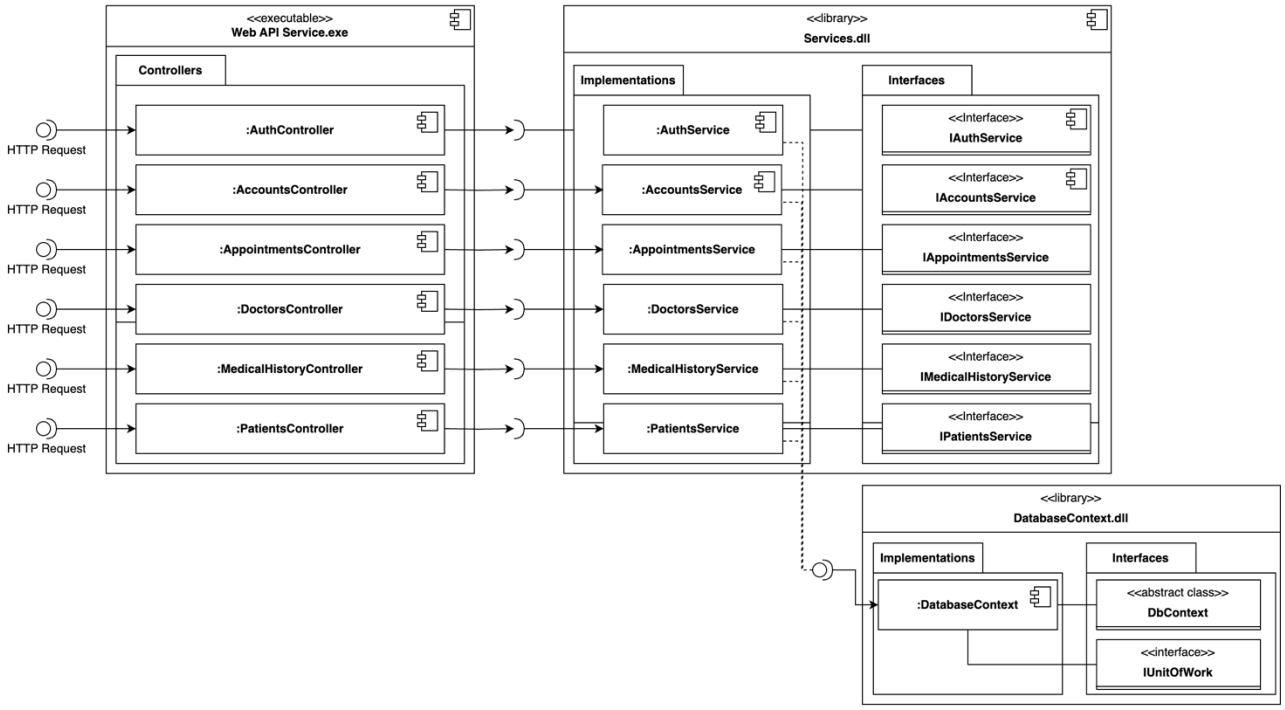


Рис. 8 Діаграма компонентів

На рис. 8 представлено діаграму компонентів медичної інформаційної системи, що демонструє її поділ на контролери, бізнес-логіку (сервіси) та доступ до бази даних.

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір системи управління базою даних та її реалізація

Одним із ключових етапів розробки будь-якої інформаційної системи є вибір відповідної системи управління базами даних (СУБД), яка забезпечуватиме зберігання, доступність, цілісність і безпеку даних. В умовах медичних інформаційних систем, які оперують персональними та конфіденційними даними, вибір СУБД має критичне значення.

Для реалізації системи в якості СУБД обрано Microsoft SQL Server, що виправдано її технічною надійністю, високим рівнем захисту та сумісністю з іншими технологіями проєкту.

Основні переваги MSSQL Server

1. Надійність і масштабованість. MSSQL Server забезпечує високу стабільність роботи системи в умовах великого навантаження. Підтримка транзакцій, реплікацій, журналювання змін та резервного копіювання дозволяє уникнути втрати даних та забезпечити високу доступність. Кластеризація дає змогу масштабувати систему як у напрямку розширення ресурсів (вертикально), так і шляхом додавання нових вузлів (горизонтально).
2. Потужна підтримка безпеки. У MSSQL реалізовано сучасні засоби захисту, зокрема:
   2.1. Шифрування даних як на рівні збереження (Transparent Data Encryption), так і під час передачі;
   2.2. Гнучка система ролей і прав доступу до об’єктів бази;
   2.3. Аудит змін у критичних таблицях.
3. Ідеальна інтеграція з .NET-екосистемою. MSSQL Server є нативним рішенням для розробників у середовищі ASP.NET Core. Він безшовно працює з Entity Framework Core - ORM, яка забезпечує автоматичну генерацію таблиць, зв’язків, міграцій та спрощує взаємодію з БД через LINQ-запити.

4. Розвинуті інструменти адміністрування. SQL Server Management Studio забезпечує зручну роботу з базою даних на всіх етапах:

4.1. Написання запитів, моніторинг продуктивності;

4.2. Налаштування прав доступу;

4.3. Автоматизоване резервне копіювання та відновлення;

4.4. Діагностика проблем у роботі запитів (Execution Plan, Profiler).

5. Microsoft SQL Server має широку підтримку хмарної інфраструктури: його легко розгорнути як локально, так і у хмарі (Azure), що забезпечує безперервний доступ до даних і гнучке масштабування без додаткових витрат.

6. Документованість та підтримка. Microsoft надає широке документування, спільноти та підтримку, що дозволяє швидко вирішувати проблеми та впроваджувати нові можливості.

Враховуючи вищевказані фактори, Microsoft SQL Server був обраний як оптимальна платформа для побудови структурованої, надійної та захищеної бази даних, яка стане фундаментом для всієї медичної інформаційної системи.

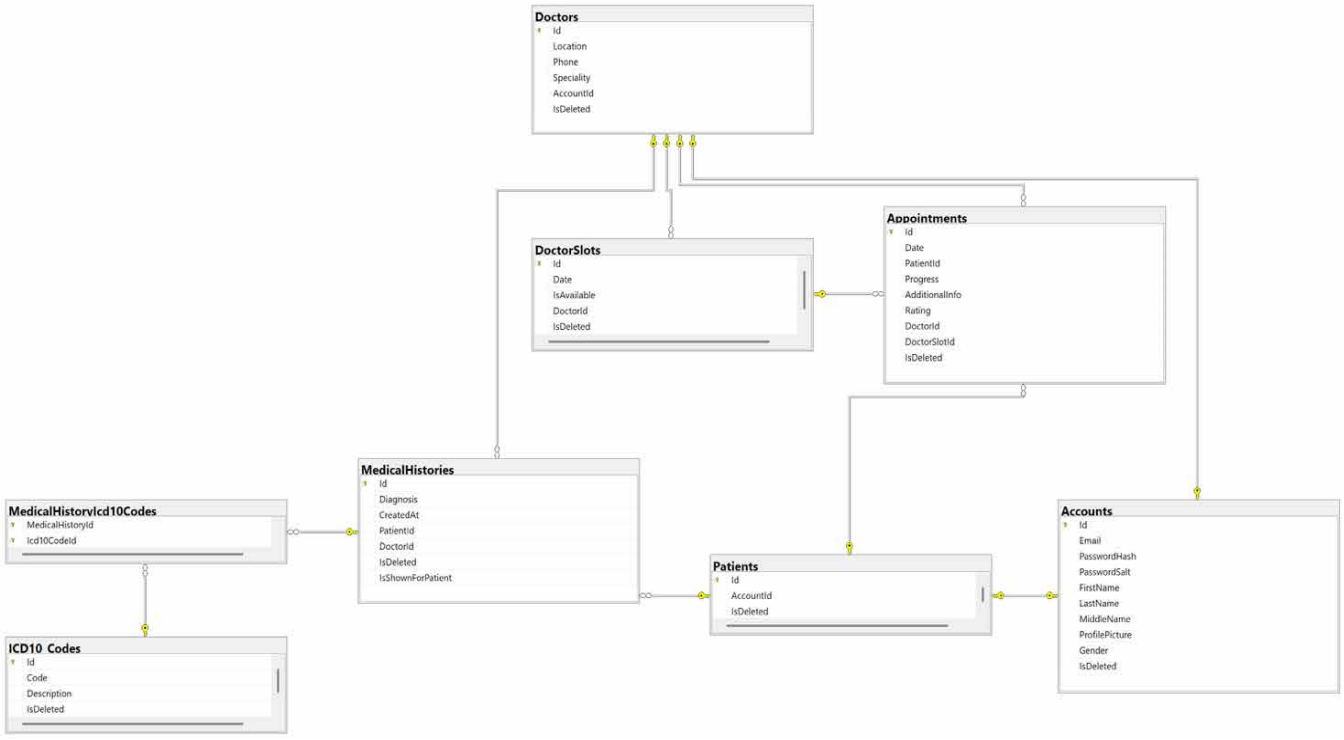


Рис. 9 Фізична модель даних

На рис. 9 зображено фізичну модель бази даних, реалізовану в середовищі MSSQL Server.

Основні сутності:

1. Accounts - зберігає облікову інформацію всіх користувачів системи: ім’я, прізвище, email, пароль (хеш і сіль), стать, фото тощо.
2. Patients - таблиця пацієнтів, яка містить зв’язок із обліковим записом (AccountId).
3. Doctors - таблиця лікарів з даними про спеціалізацію, місцезнаходження, телефон та зовнішній ключ до облікового запису.
4. Appointments - інформація про прийоми пацієнтів, включаючи дату, лікаря, пацієнта, слоти, додаткову інформацію, прогрес прийому та оцінку.
5. DoctorSlots - таблиця, що містить розклад доступних прийомів для лікаря. Поле IsAvailable визначає доступність слоту.
6. MedicalHistories - записи про медичні картки пацієнтів, де зберігаються діагнози, лікар, дата створення та атрибут IsShownForPatient.
7. ICD10Codes - довідник кодів захворювань відповідно до міжнародної класифікації хвороб (МКХ-10).
8. MedicalHistoryIcd10Codes - зв’язуюча таблиця “багато до багатьох” між медичними записами та ICD10-кодами.

Реалізаційні особливості:

1. Всі основні таблиці містять поле IsDeleted, що реалізує м’яке видалення (soft delete), дозволяючи логічне видалення записів без фізичного очищення таблиць.
2. Впроваджено зовнішні ключі для забезпечення цілісності даних між сутностями.
3. Поля з типами bool, DateTime, string ретельно підібрані відповідно до домену кожної сутності.
4. Реалізація відповідних індексів забезпечує ефективний пошук записів (наприклад, по AccountId, DoctorId, PatientId).

5. Через Entity Framework Core були створені міграції, які автоматично генерують таблиці, зв’язки та обмеження в MSSQL.

Завдяки використанню MSSQL Server у поєднанні з Entity Framework Core, реалізація бази даних забезпечує високу продуктивність, структурованість та гнучкість в обслуговуванні системи. Обрана СУБД відповідає сучасним вимогам до надійного та безпечного зберігання медичних даних, забезпечуючи основу для ефективного функціонування всієї інформаційної системи.

3.2 Інструменти для розміщення програмного забезпечення в контейнерах

Контейнеризація - це сучасний підхід до розгортання програмного забезпечення, який дозволяє упакувати застосунок разом із його залежностями в ізольоване середовище. Найбільш поширеним інструментом для цього є Docker. У межах дипломного проєкту використано

Docker Compose - це засіб для одночасного запуску кількох взаємопов’язаних компонентів у вигляді контейнерів у межах одного проєкту. Файл docker-compose.yml описує компоненти: веб-інтерфейс (React), API на ASP.NET Core, базу даних Microsoft SQL Server, сховище MinIO для файлів, а також проксі-сервер NGINX.

Такий підхід гарантує стабільність роботи незалежно від середовища, спрощує розгортання системи на будь-якому сервері та підвищує зручність обслуговування. Завдяки цьому програмна система може бути швидко протестована, перенесена або масштабована за потреби.

У межах даного проєкту для контейнеризації програмної системи обрано Docker - одну з найпопулярніших платформ у світі DevOps і хмарних обчислень.

Docker дозволяє:

1. Упаковувати застосунок разом з усіма бібліотеками, конфігураціями, службами у єдиний контейнер, що працює однаково на будь-якому сервері або ОС;

2. Знизити залежність від середовища розробника або операційної системи;
3. Пришвидшити розгортання - система може бути запущена за лічені секунди за допомогою простих команд docker run чи docker-compose up;
4. Створювати багатоконтейнерні інфраструктури;
5. Масштабувати застосунки і керувати ними через інструменти оркестрації (наприклад, Kubernetes).

Основні компоненти Docker:

1. Docker Engine - рушій, що відповідає за створення та запуск контейнерів.
2. Dockerfile - текстовий файл, у якому описується, як побудувати образ (image) застосунку: з яких шарів, базових образів, команд, копій файлів тощо.
3. Docker Image - шаблон контейнера, який можна багаторазово запускати на різних машинах.
4. Docker Container - працююча інстанція образу, що містить додаток та все необхідне для його виконання.
5. Docker Compose - інструмент, який дозволяє керувати кількома контейнерами одночасно через один YAML-файл.

Для розгортання програмної системи у середовищі Docker використовується конфігураційний файл docker-compose.yml, який дозволяє одночасно запускати декілька пов’язаних між собою сервісів. Це значно спрощує інфраструктуру проєкту та забезпечує ефективне масштабування, відлагодження й підтримку. Зміст файлу docker-compose.yml зображений на Рис.10.1 і 10.2.

docker-compose.yml

```yaml
version: '3.8'
name: health-app-nubip
services:
  healthapp.webapi:
    image: health-app-nubip.api
    container_name: healthapp_webapi_container
    build:
      context: BE
      dockerfile: Dockerfile
    ports:
      - '5050:5050'
    extra_hosts:
      - "host.docker.internal:host-gateway"
    depends_on:
      - healthapp.sqlserver
      - healthapp.minio
    networks:
      - health-network
    environment:
      ASPNETCORE_ENVIRONMENT: Production
      DatabaseSettings__ConnectionString: "Server=healthapp.sqlserver;Database=HealthAppNubip;User Id=sa;Password=Qwerty123!;TrustServerCertificate=
      MinioSettings__AccessKey: "minioadmin"
      MinioSettings__SecretKey: "minioadmin"
      MinioSettings__BucketName: "healthapp"
      MinioSettings__Endpoint: "healthapp.minio:9000"
      ReCaptchaSettings__PublicKey: "6Lcaqf4qAAAAAOEb-6wjiFR7F2IkpUP5MIgHH-jW"
      ReCaptchaSettings__SecretKey: "6Lcaqf4qAAAAAI6PcTxffTqz0SITAQJVNo9tBFaC"

  healthapp.ui:
    image: health-app-nubip.ui
    container_name: healthapp_ui_container
    build:
      context: FE
      dockerfile: Dockerfile
      args:
        REACT_APP_API_URL: "/api/"
        REACT_APP_MINIO_URL: "/healthapp/"
    networks:
      - health-network

  healthapp.nginx:
    image: nginx:alpine
    container_name: healthapp_nginx_container
    ports:
      - "80:80"
    volumes:
      - ./FE/default.conf:/etc/nginx/conf.d/default.conf
      - ./FE/build:/usr/share/nginx/html
    depends_on:
      - healthapp.ui
      - healthapp.webapi
      - healthapp.minio
    networks:
      - health-network
```

Рис. 10.1. Зміст файлу docker-compose.yml.

```
  healthapp.sqlserver:
    image: mcr.microsoft.com/mssql/server:latest
    container_name: healthapp_sqlserver_container
    environment:
      - ACCEPT_EULA=Y
      - MSSQL_SA_PASSWORD=Qwerty123!
      - MSSQL_PID=Express
    ports:
      - "1433:1433"
    volumes:
      - sql_data:/var/opt/mssql
    networks:
      - health-network

  healthapp.minio:
    image: minio/minio
    container_name: healthapp_minio_container
    command: server /data --console-address ":9001"
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      - MINIO_ROOT_USER=minioadmin
      - MINIO_ROOT_PASSWORD=minioadmin
    volumes:
      - minio_data:/data
    networks:
      - health-network

volumes:
  sql_data:
  minio_data:

networks:
  health-network:
    driver: bridge
```

Рис. 10.2 Вміст файлу docker-compose.yml. Продовження.

У наведеному проєкті використовується п’ять основних контейнерів:

1. **healthapp.webapi** - серверна частина ASP.NET Core
   1.1. Образ створюється на основі Dockerfile з директорії BE.
   1.2. Проксі-доступ відкривається на порт 5050.
   1.3. Має залежності від сервісів бази даних healthapp.sqlserver та файлового сховища healthapp.minio.
   1.4. Через змінні середовища передаються налаштування до: бази даних MSSQL та сховища файлів MinIO;
2. **healthapp.ui** - фронтенд частина на React
   2.1. Образ будується з директорії FE, яка містить фронтенд-додаток.
   2.2. Під час складання передаються параметри оточення для інтеграції з API та MinIO.
   2.3. Працює у спільній мережі з іншими компонентами.
3. **healthapp.nginx** - зворотний проксі сервер

3.1. Використовується стандартний образ nginx:alpine.

3.2. Відповідає за маршрутизацію запитів: перенаправляє HTTP-запити на фронтенд або API.

3.3. Монтує default.conf для конфігурації маршрутизації та папку build як кореневий сайт.

4. **healthapp.sqlserver** - база даних Microsoft SQL Server

4.1. Використовується офіційний образ mcr.microsoft.com/mssql/server:latest.

4.2. Проброшено порт 1433 для зовнішнього доступу.

4.3. Задано пароль адміністратора sa, прийнято ліцензійну угоду.

4.4. Дані зберігаються у зовнішньому volume sql_data.

5. **healthapp.minio** - файлове сховище

5.1. Образ minio/minio піднімає сервер для зберігання файлів (наприклад, результатів обстежень, зображень).

5.2. Порти 9000 (інтерфейс доступу до файлів) і 9001 (консоль адміністратора) відкриті.

5.3. Ініціалізується root-доступом (minioadmin/minioadmin).

5.4. Дані зберігаються у volume minio_data.

Переваги використання Docker Compose:

1. Швидке локальне розгортання: достатньо однієї команди docker-compose up для запуску всієї системи.

2. Легкість у супроводі: вся інфраструктура описана декларативно.

3. Повна ізоляція компонентів: зменшується ризик конфліктів залежностей.

4. Можливість CI/CD-інтеграції: легко включити до конвеєра автоматичного тестування і деплою.

3.3 Вибір інструментарію для створення програмного забезпечення

3.4.1 Back-end додаток. Серверна частина програмного забезпечення реалізована з використанням сучасного технологічного стеку на базі платформи .NET 9. Це кросплатформне рішення від Microsoft, яке об’єднує в собі високу продуктивність, стабільність, багаторічну підтримку та зручну інтеграцію з іншими технологіями екосистеми .NET.

Обґрунтування вибору:

.NET 9 є стабільною та перспективною платформою, яка підтримується як у локальному, так і в хмарному середовищі (Azure, AWS, Docker). Вона забезпечує чудову підтримку багатопоточності, ефективне управління пам’яттю, а також високу швидкодію, що критично важливо для масштабованих веб-сервісів.

Основною мовою програмування виступає C#, яка має сучасний синтаксис, підтримує об’єктно-орієнтовану та функціональну парадигми, дозволяє ефективно структурувати код, впроваджувати шаблони проєктування та дотримуватись принципів SOLID.

API-сервер реалізовано з використанням ASP.NET Core — продуктивного фреймворку для створення REST-сервісів, що легко масштабується та підтримує сучасну архітектуру. Його модульна структура дозволяє реалізовувати мікросервісну архітектуру, налаштовувати маршрутизацію, реалізовувати автентифікацію, логування, обробку помилок та інші middleware-рішення.

Сховище даних:

1. У якості основної реляційної СУБД використовується Microsoft SQL Server. Ця система забезпечує:
2. Високу надійність зберігання структурованих даних;
3. Підтримку транзакцій та складних запитів;

4. Можливість створення процедур, тригерів, індексів для підвищення продуктивності;
5. Засоби резервного копіювання та безпеки, що є критичними при роботі з медичною інформацією.
6. Таблиці, зв’язки та обмеження реалізовано через ORM-бібліотеку Entity Framework Core, яка дозволяє будувати запити до БД через LINQ, виконувати автоматичні міграції та підтримувати “code-first” підхід у проєктуванні структури бази даних.

Обробка файлів:

- Для зберігання неструктурованих даних (медичні зображення, результати аналізів, документи тощо) використовується об’єктне файлове сховище MinIO. Це легковажна, продуктивна S3-сумісна система з відкритим вихідним кодом, яка:

1. Підтримує масштабування на великі обсяги файлів;
2. Інтегрується з API через бібліотеки .NET;
3. Дозволяє керувати доступом до файлів
4. Розгортається у Docker-контейнері, що спрощує інфраструктурну підтримку.

Архітектурні особливості:

Уся логіка розподілена за класичним принципом “контролер – сервіс – репозиторій”:

1. Контролери приймають HTTP-запити й передають їх у сервіси;
2. Сервіси реалізують бізнес-логіку та взаємодіють з репозиторіями;
3. Репозиторії забезпечують доступ до БД через EF Core.
4. Для конфігурації використано підхід “strongly typed settings”, що дозволяє безпечно працювати з параметрами з appsettings.json, зокрема для:
5. Підключення до MSSQL Server;
6. Роботи з MinIO;

7. Інтеграції з Google ReCaptcha.
8. Реалізовано підтримку токен-автентифікації, обмеження доступу за ролями, валідацію моделей та обробку винятків за допомогою власних middleware.

3.3.2 Front-end додаток. Клієнтська частина інформаційної системи реалізована з використанням React - популярної бібліотеки для побудови інтерфейсів користувача, розробленої компанією Meta (Facebook). Вибір цієї технології зумовлений її високою продуктивністю, компонентною архітектурою та широкими можливостями для створення сучасних, адаптивних вебзастосунків.

Основні переваги React:

1. Компонентний підхід. Уся логіка інтерфейсу поділяється на багаторазово використовувані компоненти (наприклад: AppointmentCard, PatientForm, DoctorList), що спрощує підтримку, масштабування та повторне використання коду.
2. Віртуальний DOM. React оптимізує оновлення сторінки, мінімізуючи кількість змін у реальному DOM, що суттєво підвищує швидкодію навіть на слабких пристроях.
3. Розвинена екосистема. Платформа підтримує інтеграцію з великою кількістю додаткових бібліотек, зокрема:

Використані бібліотеки:

1. Material UI - бібліотека UI-компонентів, що реалізує дизайн-систему Google Material Design. Завдяки їй створено сучасний, інтуїтивно зрозумілий і адаптивний інтерфейс: кнопки, форми, вкладки, модальні вікна, таблиці, теми, кастомні компоненти на основі MUI (TextField, DatePicker, DataGrid).

2. Redux Toolkit - використовується для централізованого зберігання та керування станом застосунку. Redux дозволяє:

- Синхронізувати стан між різними компонентами;
- Передавати глобальні дані;
- Реалізовувати асинхронні запити;

3. Axios - бібліотека для HTTP-запитів до API. Axios надає зручний синтаксис для GET/POST/PUT/DELETE-запитів, обробку помилок, заголовків та інтерцепторів. У проєкті Axios використовується для взаємодії з бекендом.

4. Yup + React Hook Form - для валідації форм використовується бібліотека Yup, яка дозволяє гнучко описувати правила перевірки даних. У зв'язці з react-hook-form забезпечено:

- Обробку вводу користувача в реальному часі;
- Показ помилок без перезавантаження;
- Динамічне формування складних форм;

Архітектурні особливості front-end частини:

1. Кожен функціональний блок реалізований як окремий маршрут за допомогою React Router (/login, /appointments, /profile, /patients, /history).

2. Для роботи з ролями реалізовано авторизаційні маршрути - користувачі бачать лише ті сторінки, які дозволено їх роллю (наприклад, пацієнт не може редагувати профілі лікарів).

3. Усі форми мають візуальний зворотний зв'язок, валідацію та підказки.

4. Інтерфейс адаптовано під різні розміри екранів, що забезпечує зручну роботу з системою на ноутбуках і планшетах.

5. Додано можливість перегляду медичних карток, результатів прийомів, запису до лікаря та вибору часу прийому у зручному візуальному форматі.

3.3.3 Середовище розробки. У процесі розробки програмного забезпечення важливу роль відіграє вибір ефективного, зручного та функціонального середовища розробки, яке дозволяє прискорити написання коду, полегшує навігацію проєктом, інтегрується з системами контролю версій і підтримує інструменти налагодження, форматування, аналізу коду та профілювання.

Для реалізації серверної частини проєкту було обрано середовище JetBrains Rider. Це сучасне інтегроване середовище розробки для .NET-платформи, яке поєднує у собі переваги ReSharper та IntelliJ-платформи. Rider забезпечує високу швидкість роботи навіть у великих проєктах, має розвинену підтримку C#, ASP.NET Core, Entity Framework та Docker. Вбудовані інструменти для перегляду бази даних, виконання SQL-запитів, рефакторингу та автоматичного форматування коду дозволяють підвищити ефективність розробки. Rider також зручно інтегрується з git, що забезпечує повний контроль над змінами у проєкті.

Для створення клієнтської частини (фронтенду), яка розроблялася на основі React, було використано редактор коду Visual Studio Code. Цей редактор обрано через його легкість, швидкодію, велику кількість корисних розширень та гнучкість налаштувань. Завдяки плагінам для роботи з JavaScript, TypeScript, React, Redux та Material UI, а також підтримці інтеграції з системами контролю версій і терміналом, VS Code забезпечив комфортну і швидку розробку користувацького інтерфейсу. Наявність автодоповнення, linting-аналізу, форматування коду та можливості запуску React-додатку безпосередньо з редактора значно спростили цикл розробки.

Таким чином, обрані середовища JetBrains Rider та Visual Studio Code повністю покрили потреби серверної і клієнтської частини відповідно, забезпечивши ефективну, зручну та стабільну розробку всієї інформаційної системи.

4. ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Діаграма розгортання.

Діаграма розгортання належить до UML-нотації та ілюструє, як окремі програмні модулі фізично розміщені на пристроях. Вона допомагає зрозуміти, які компоненти запускаються на яких серверах, яким чином вони взаємодіють і які канали зв’язку при цьому використовуються.

На рис.11 представлено архітектурну схему, що демонструє, як компоненти інформаційної системи для медичної платформи розміщуються у продакшн-середовищі, організованому за допомогою Docker Compose. Це дозволяє забезпечити ізольовану, контрольовану та масштабовану інфраструктуру.

Взаємодія з користувачем здійснюється через браузер, з якого надсилаються HTTPS-запити до NGINX-сервера. NGINX виконує роль зворотного проксі та роздає статичні файли клієнтської частини (React), а також перенаправляє запити до ASP.NET Web API. Веб-застосунок, що працює на порту 5050, реалізує всю бізнес-логіку системи та обробляє запити до бази даних, що працює на основі Microsoft SQL Server.

Для зберігання файлів використовується об’єктне сховище MinIO, до якого надсилаються прямі запити з бекенду або клієнта через HTTP. Крім цього, для надсилання поштових повідомлень (наприклад, підтвердження реєстрації чи відновлення пароля) ASP.NET-додаток взаємодіє з SMTP-сервером через протокол Simple Mail Transfer Protocol.

Завдяки використанню Docker Compose всі ці компоненти - бекенд, фронтенд, NGINX, MinIO, SQL Server - розгортаються як окремі контейнери у єдиній внутрішній мережі health-network. Це спрощує налаштування сервісів, зменшує кількість ручної конфігурації та забезпечує зручне масштабування в майбутньому.

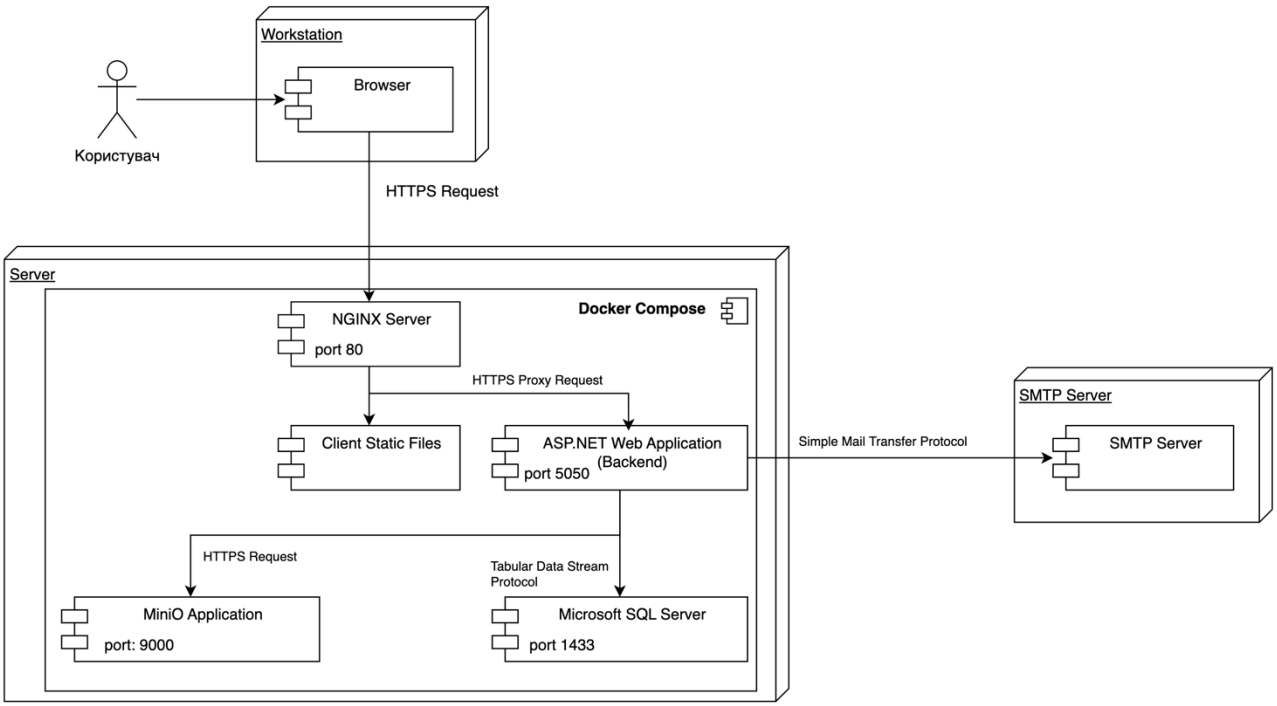


Рис. 11 Діаграма розгортання

Основні компоненти діаграми:

1. Workstation (Користувач)

Користувач взаємодіє з системою через браузер, надсилаючи HTTPS-запити до сервера.

2. Серверна частина (реалізована в Docker Compose):

2.1. NGINX (порт 80) - фронтовий вхідний шлюз, що:

- обробляє всі вхідні запити;
- обслуговує статичні клієнтські файли (з React);
- перенаправляє API-запити до ASP.NET backend.

2.2. Client Static Files - побудований фронтенд (React + MUI) розміщений як статичний вміст на NGINX.

2.3. ASP.NET Web Application - серверна частина додатку, написана на C# (.NET 9), що обробляє бізнес-логіку, запити до бази даних, авторизацію, запис на прийом тощо.

2.4. Microsoft SQL Server - реляційна база даних, яка зберігає всю критичну інформацію (користувачі, лікарі, прийоми, медичні картки тощо).Підключення відбувається через протокол Tabular Data Stream.

2.5. MinIO - внутрішнє файлове сховище, що використовується для збереження файлів (аналізи, фото профілів, медичні документи тощо). Додаток використовує MinIO через API-запити.

3. SMTP Server - Розташований поза контейнерною мережею. Отримує листи з ASP.NET backend через SMTP протокол - наприклад, для відновлення пароля, підтвердження email чи сповіщень.

Основні переваги такого розгортання:

1. Модульність: кожен компонент - окремий сервіс.
2. Швидке масштабування та деплой: достатньо змінити docker-образ або конфіг, щоб перезапустити.
3. Ізольованість та безпека: сервіси не залежать від глобальної ОС, а працюють в контейнерах.
4. Зручна CI/CD інтеграція: можливість легко автоматизувати розгортання в хмарі або локально.

4.2 Вимоги до апаратного та програмного забезпечення

Щоб забезпечити стабільну роботу створеної системи, слід дотримуватись визначених технічних умов як на серверній, так і на клієнтській стороні. Враховуючи використання контейнеризації, сучасного стеку технологій та об’єктного сховища файлів, інфраструктура повинна забезпечувати достатню продуктивність, стабільність та резерв потужності для масштабування системи в майбутньому.

Вимоги до серверної частини:

Сервер повинен мати достатньо обчислювальних ресурсів для одночасного виконання кількох контейнерів: ASP.NET Web API, Microsoft SQL Server, MinIO, NGINX, а також забезпечувати швидкий доступ до дискової системи для обробки файлів пацієнтів. Рекомендовані мінімальні характеристики:

• Процесор: не менше 4 фізичних ядер з частотою від 2.4 ГГц;

• Оперативна пам’ять: від 8 ГБ, рекомендовано 16 ГБ для стабільної роботи MSSQL Server та MinIO;

• Накопичувач: SSD-диск об’ємом від 100 ГБ, бажано з підтримкою резервного копіювання;

• Мережеве з’єднання: стабільне з’єднання зі швидкістю не менше 100 Мбіт/с.

Вимоги до програмного забезпечення сервера:

• Операційна система: Linux (Ubuntu 20.04+ / Debian / CentOS) або Windows Server 2019+;

• Docker Engine (версія 20.10 або новіша);

• Docker Compose (версія 1.29 або новіша);

• .NET 9 SDK та Runtime;

• SQL Server;

• NGINX (вбудований контейнером);

• MinIO (вбудований контейнером);

• SMTP-сервер.

Вимоги до клієнтської частини (користувача):

Враховуючи, що система реалізована у вигляді веб-додатку, користувачеві достатньо мати сучасний інтернет-браузер і стабільне підключення до мережі. Установка додаткового ПЗ не потрібна.

4.3 Склад інсталяційного пакету

Інсталяційний пакет системи розроблено таким чином, щоб забезпечити швидке розгортання всієї інфраструктури за допомогою Docker Compose. Усі необхідні компоненти згруповані у вигляді окремих директорій або

контейнерних образів. Пакет передбачає повну автоматизацію розгортання та не потребує ручної інсталяції залежностей на сервері.

Інсталяційний пакет містить такі елементи:

1. docker-compose.yml - основний конфігураційний файл, що описує усі сервіси (бекенд, фронтенд, SQL Server, MinIO, NGINX) та взаємозв'язки між ними;
2. Каталог BE/ (Backend) - вихідний код ASP.NET Web API-додатку, а також Dockerfile для збірки контейнера;
3. Каталог FE/ (Frontend) - зібрана React-програма, файл Dockerfile для створення образу та конфігураційний файл default.conf для налаштування маршрутизації у NGINX;
4. Файли конфігурацій середовища (.env, змінні середовища у docker-compose.yml) - містять параметри для підключення до бази даних, MinIO, SMTP тощо;
5. База даних - ініціалізується автоматично через Entity Framework Core (після першого запуску), тому окремі SQL-скрипти не потребуються;
6. Образи сервісів (опціонально) - якщо використовується CI/CD або стороннє розгортання, можуть бути надані заздалегідь зібрані Docker-образи у форматі .tar.

Після запуску команди docker-compose up -d система автоматично розгортає всі необхідні сервіси, виконує міграції бази даних і стає доступною для користувачів у локальній або публічній мережі. Усі дані зберігаються в окремих томах, що дозволяє зберігати стійкий стан між перезапусками.

4.4 Демонстрація роботи програмної системи

Після запуску команди docker-compose up система автоматично створює та ініціалізує всі необхідні контейнери: бекенд-додаток на базі ASP.NET, фронтенд-додаток на основі React, реляційну базу даних Microsoft SQL Server,

файлове сховище MinIO, а також сервер NGINX, що виконує функції зворотного проксі.

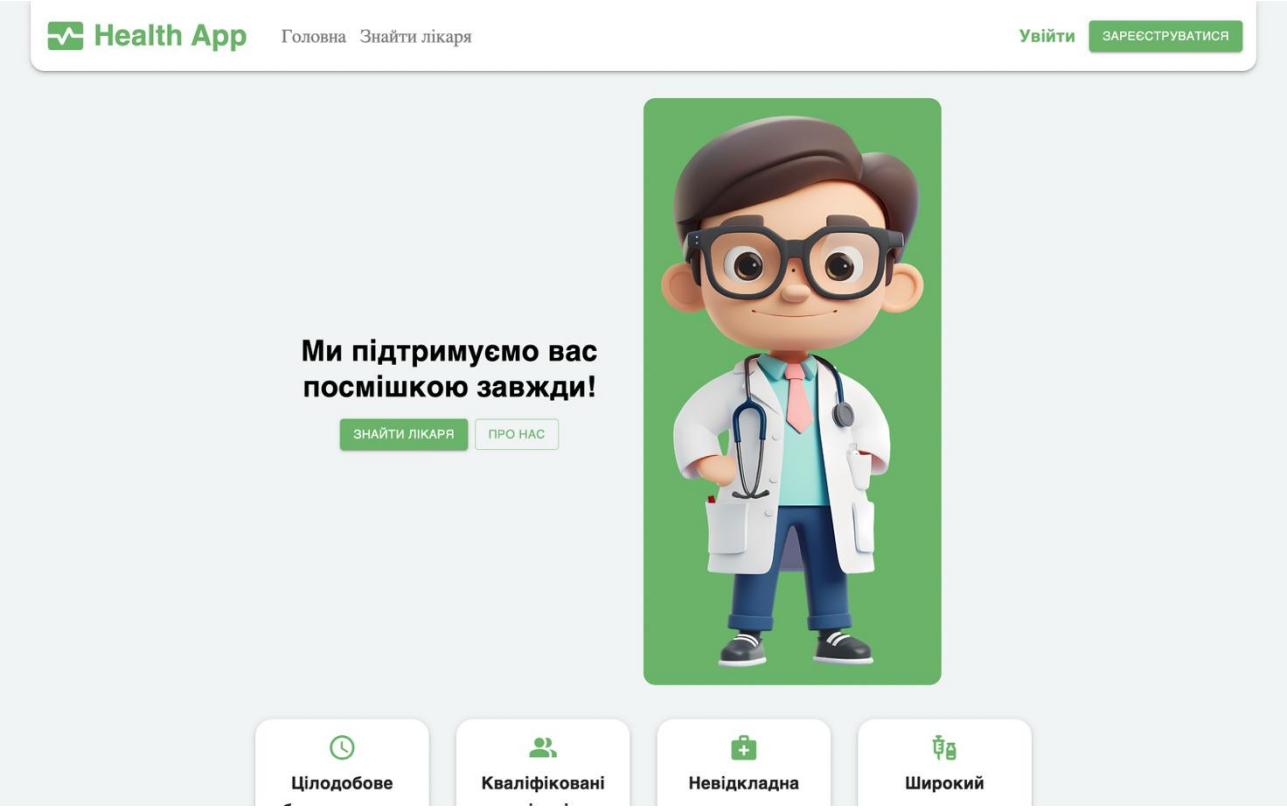


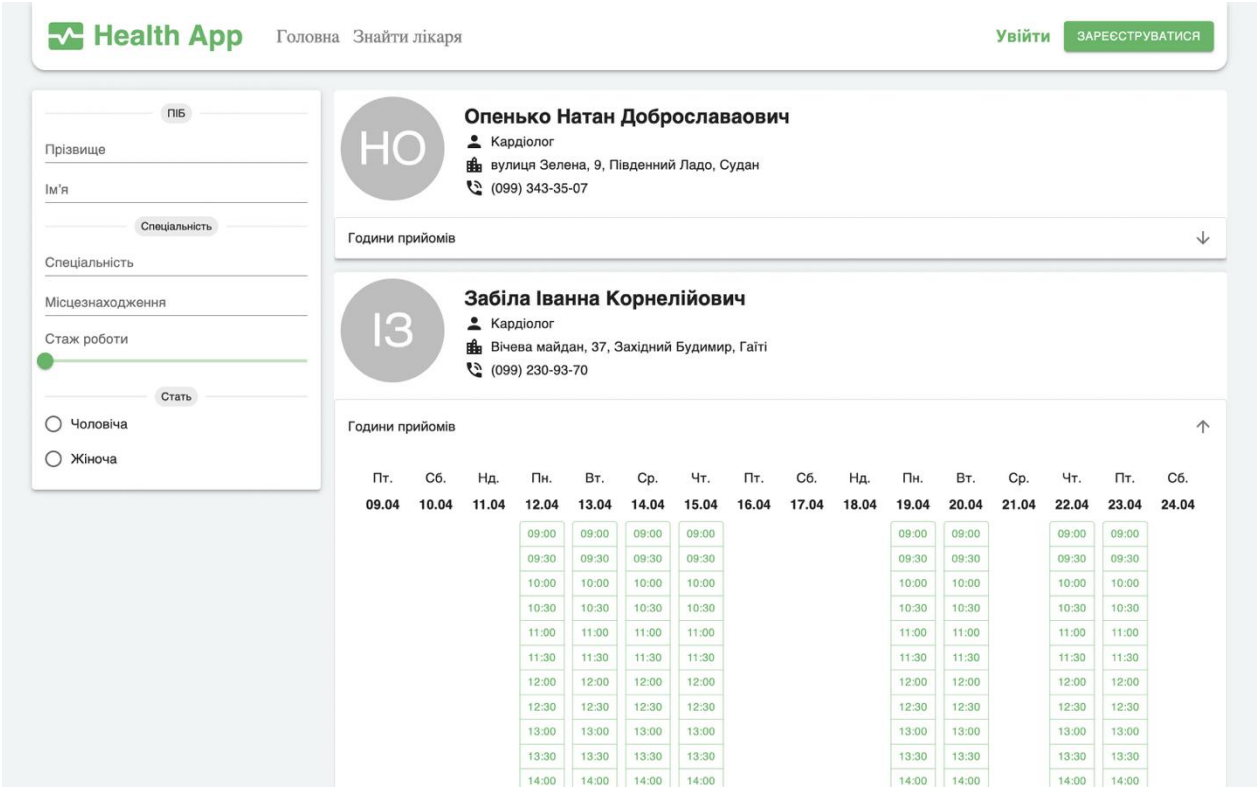
Рис. 12 Головна сторінка додатку

Після успішного запуску система стає доступною за адресою сервера. Користувач потрапляє на головну сторінку застосунку, що зображена на рис. 10. Інтерфейс оформлено у світлих тонах, з акцентами зеленого кольору, що асоціюється зі здоров'ям і турботою. Сторінка містить привітальне повідомлення, заклик до дії та навігаційне меню з можливістю авторизації або реєстрації.

Окрім головної сторінки, користувач може перейти до розділу "Знайти лікаря" (рис.13), де реалізовано функціонал пошуку медичних спеціалістів за різними критеріями. На сторінці представлено зручну форму фільтрації, яка дозволяє шукати лікарів за прізвищем, ім'ям, спеціальністю, місцезнаходженням, стажем роботи та статтю.

У результаті пошуку відображаються картки лікарів з персональною інформацією: ПІБ, спеціалізація, адреса прийому, контактний номер телефону. Поряд з кожним лікарем представлено графік доступних прийомів, реалізований у вигляді табличного календаря з виділеними вільними слотами за

кожен день. Це дозволяє пацієнтам у зручній візуальній формі ознайомитися з



розкладом і вибрати відповідний час для запису.

Рис. 13 Пошук лікаря

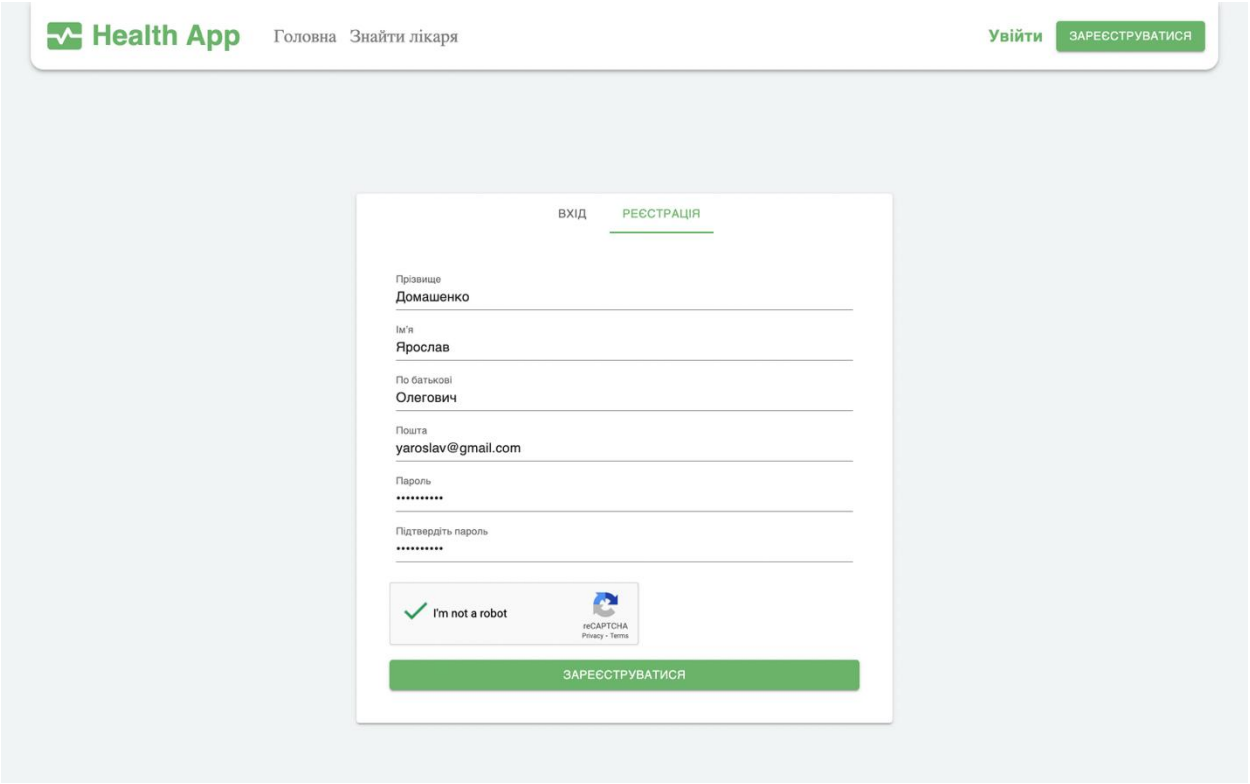


Рис. 14 Реєстрація користувача

Серед базових можливостей системи реалізовано функціонал реєстрації нового користувача, що дозволяє створити особистий обліковий запис для подальшої роботи з платформою. Форма реєстрації (рис. 14) містить обов’язкові поля для введення персональних даних: прізвища, імені, по батькові, адреси електронної пошти, пароля та його підтвердження.

Для захисту від автоматизованих запитів інтегровано Google reCAPTCHA, яка запобігає зловживанням та створенню бот-акаунтів. Усі поля мають клієнтську валідацію через бібліотеку Yup, з миттєвим відображенням помилок при неправильному введенні. Після успішної реєстрації користувач автоматично додається до системи та отримує можливість авторизації.

Передбачена перевірка правильності заповнення форми, мінімальних вимог до складності пароля, а також унікальності електронної пошти.

Цей етап є першим кроком користувача у взаємодії з системою - після чого відкривається доступ до розширеного функціоналу.

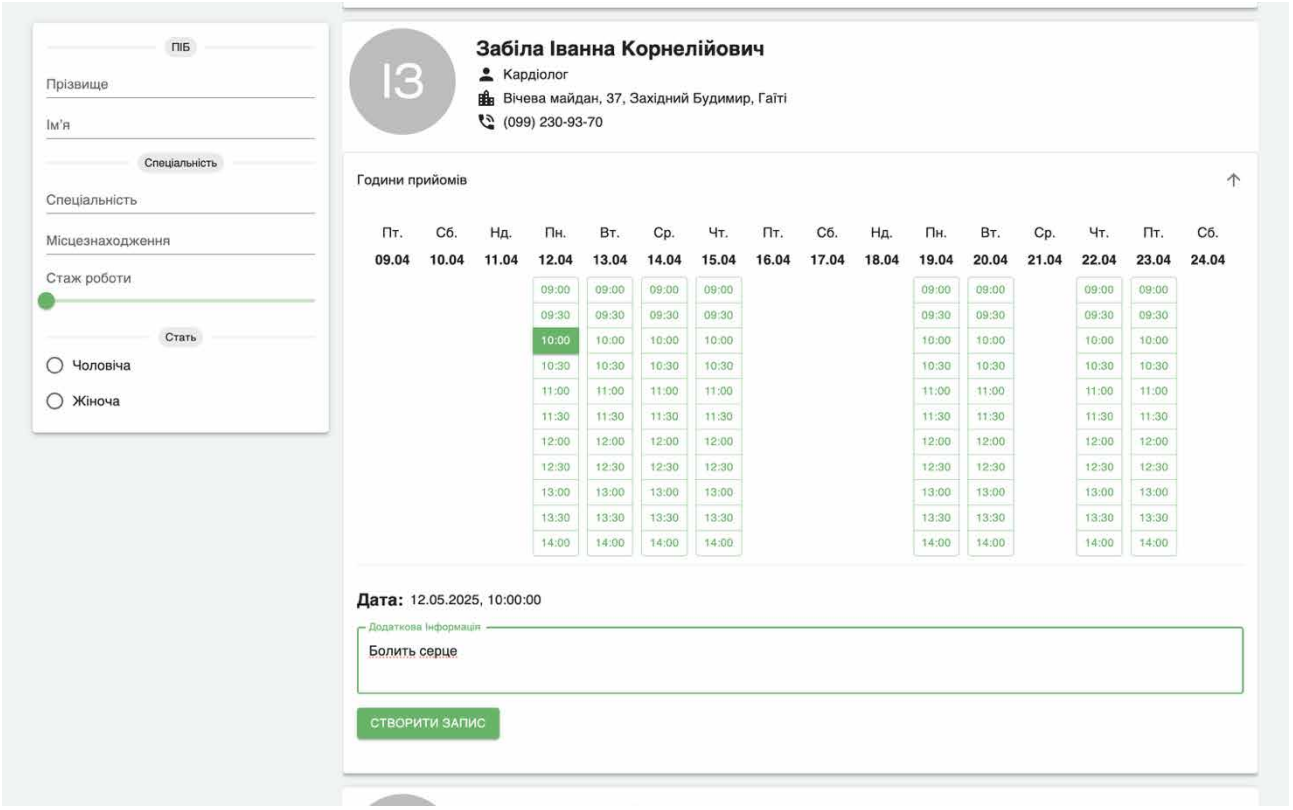


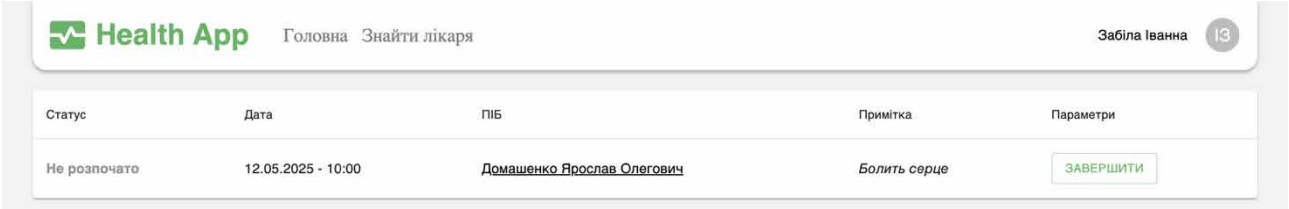
Рис. 15 Запис на вільний прийом

Ключовою функціональною можливістю для пацієнта є запис на прийом до лікаря. Після вибору спеціаліста зі списку, користувач бачить календар із

доступними годинами прийому (рис. 15). Система дозволяє обрати конкретну дату й час, які ще не зайняті іншими пацієнтами. При цьому реалізовано перевірку доступності слоту - один і той самий час не може бути вибраний одночасно кількома користувачами.

Після вибору слоту, пацієнт має можливість додати коментар або додаткову інформацію. Це поле є необов'язковим, але дозволяє зробити комунікацію між пацієнтом і лікарем ефективнішою. Після натискання кнопки "Створити запис", система зберігає дані в базу й оновлює розклад прийомів у режимі реального часу.

Запис реалізований таким чином, щоб бути максимально інтуїтивно зрозумілим, швидким і доступним навіть для користувачів з мінімальним

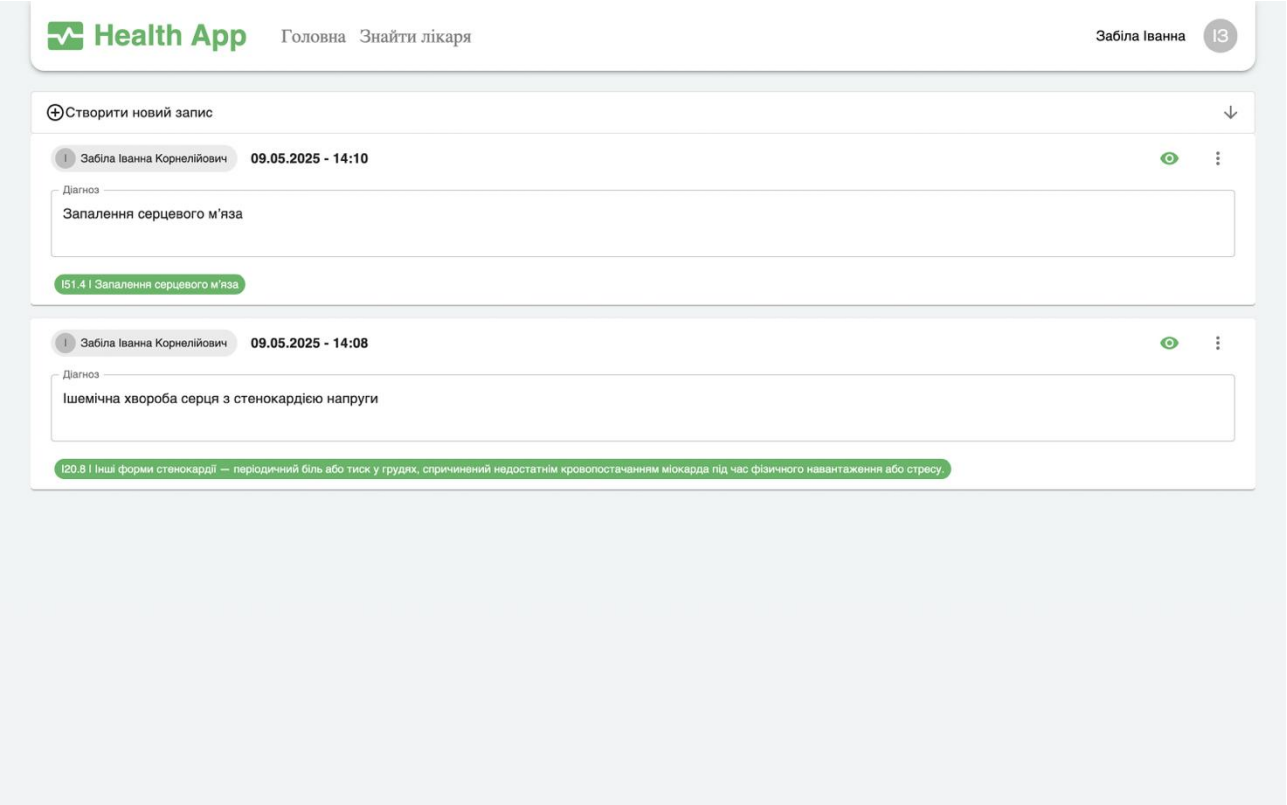


досвідом взаємодії з онлайн-системами.

Рис. 16 Перегляд лікарських прийомів

Після створення запису пацієнта, лікар має можливість переглянути всі свої прийоми у спеціальному розділі - "Список прийомів" (рис. 16). У цьому інтерфейсі відображається таблична структура з основною інформацією: статус прийому, дата й час, ПІБ пацієнта, а також коротка примітка, залишена ним під час запису.

Цей список дозволяє лікарю оперативно орієнтуватися у своїх запланованих візитах, переглядати дані пацієнтів, а також керувати статусом прийому. При натисканні кнопки "ЗАВЕРШИТИ", прийом позначається як завершений, що дозволяє формувати динамічну історію відвідувань і виключає повторне редагування або зміну цього слоту в майбутньому.



Функціонал реалізовано таким чином, щоб забезпечити лікареві простий, швидкий і зручний доступ до усіх майбутніх записів, з можливістю фільтрації, перегляду деталей та відмітки про виконання прийому.

Рис.17 Медична картка пацієнта

Окремий розділ системи присвячений медичній картці пацієнта - централізованому сховищу всієї медичної історії користувача (рис. 17). У цьому інтерфейсі лікар має змогу переглядати всі завершені прийоми, призначати діагнози, додавати коментарі та прикріплювати відповідні коди за класифікатором МКХ-10.

Кожен запис містить такі дані:

- ПІБ лікаря, який поставив діагноз;
- Дата та час прийому, коли було сформовано запис;
- Формулювання діагнозу;
- Офіційний код МКХ-10;
- Короткий опис захворювання, відповідно до міжнародного класифікатора.

Система дозволяє створювати кілька діагнозів на один прийом або вести хронологію звернень. Кожен запис має ідентифікатор, можливість перегляду деталей, а також статус видимості для пацієнта.

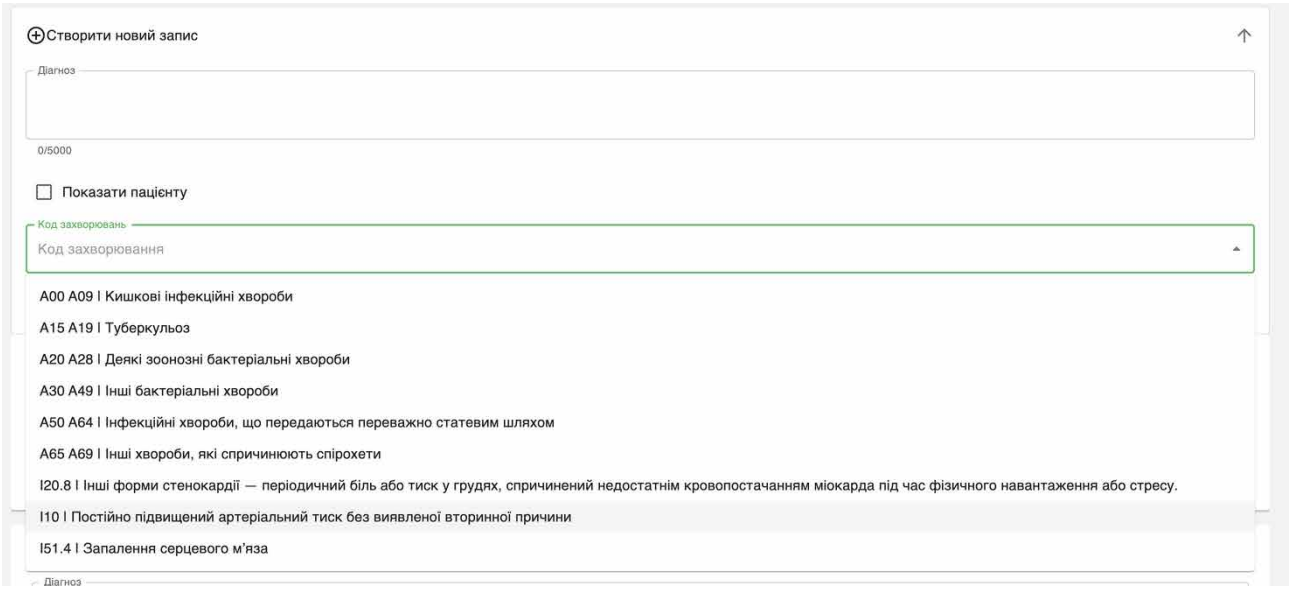


Рис. 18 Створення запису в медичній картці

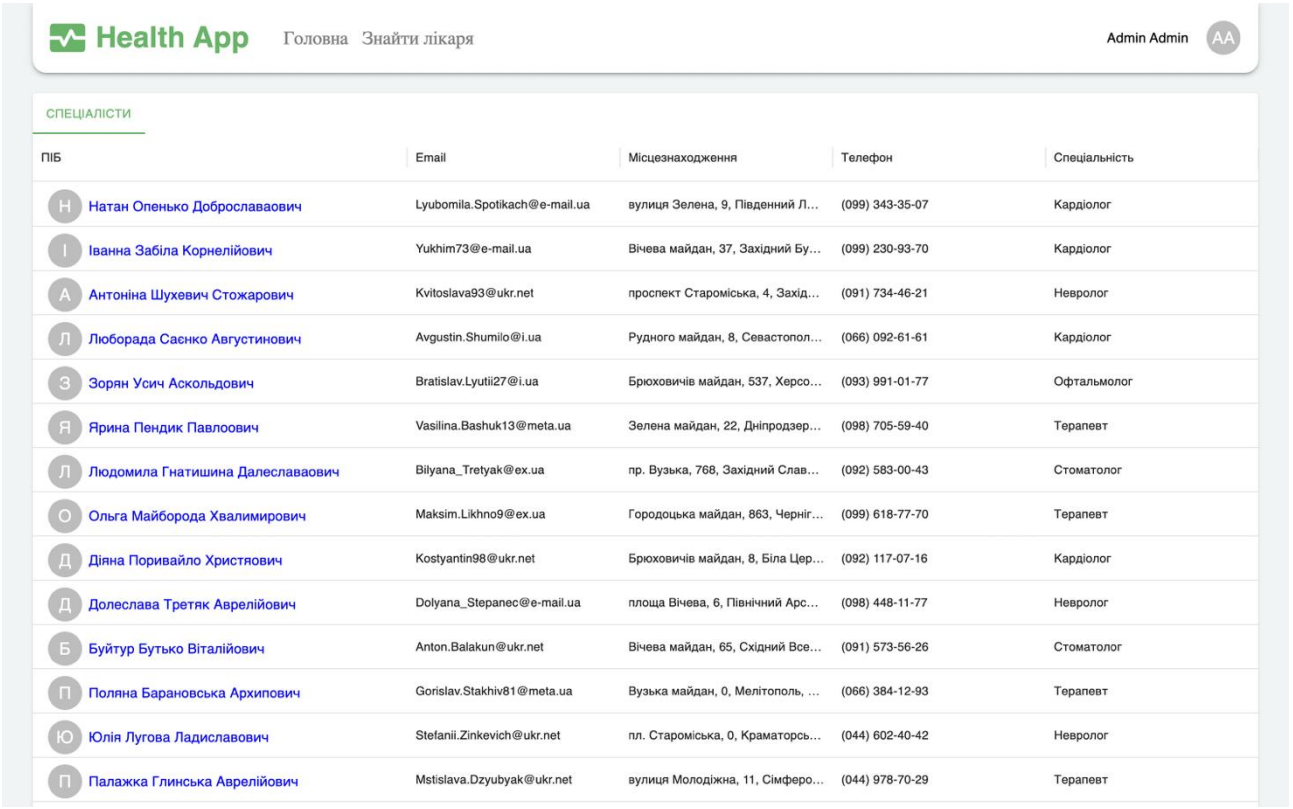
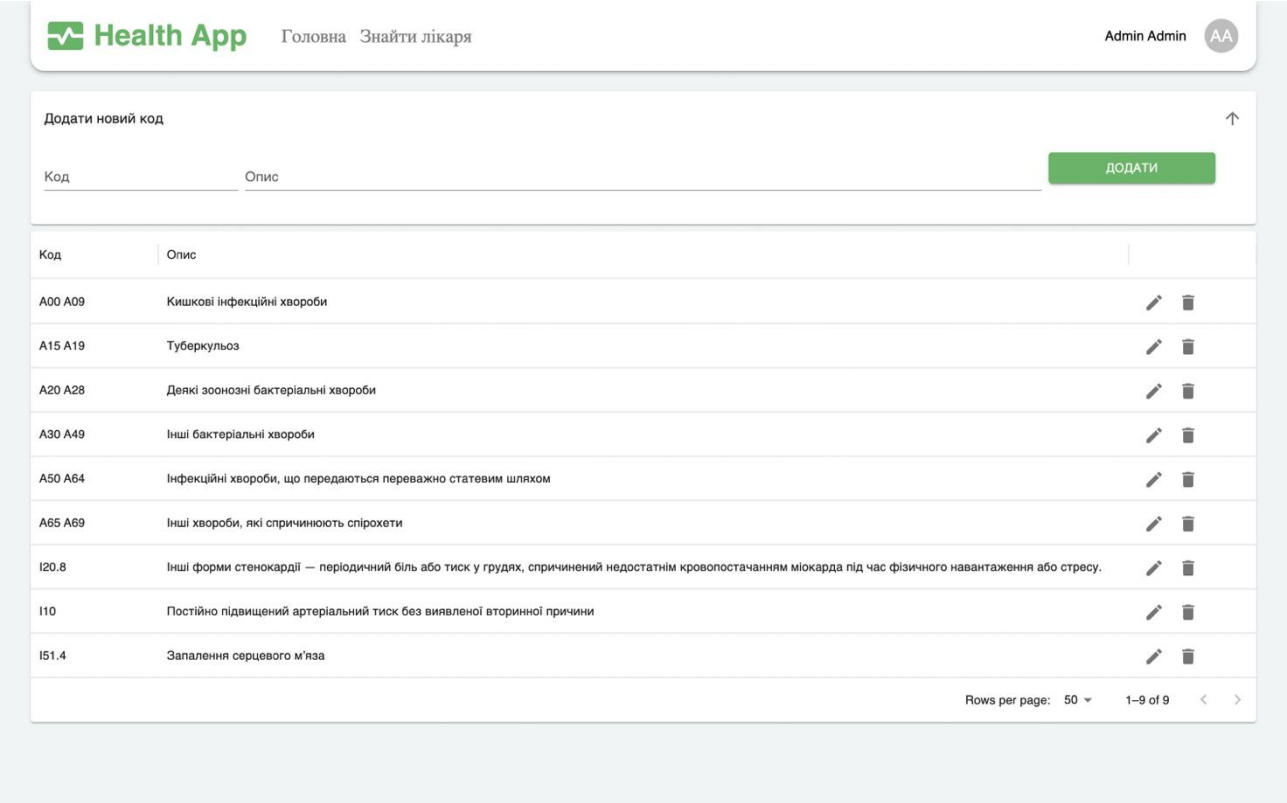


Рис. 19 Управління спеціалістами системи

У системі передбачено окремий інтерфейс для адміністратора, який має розширені права доступу до перегляду та управління обліковими записами медичних працівників (рис. 19). На сторінці налаштувань спеціалістів відображається повний список зареєстрованих лікарів з ключовою інформацією: прізвище, ім’я та по батькові, email, місцезнаходження, номер



телефону та спеціалізація.

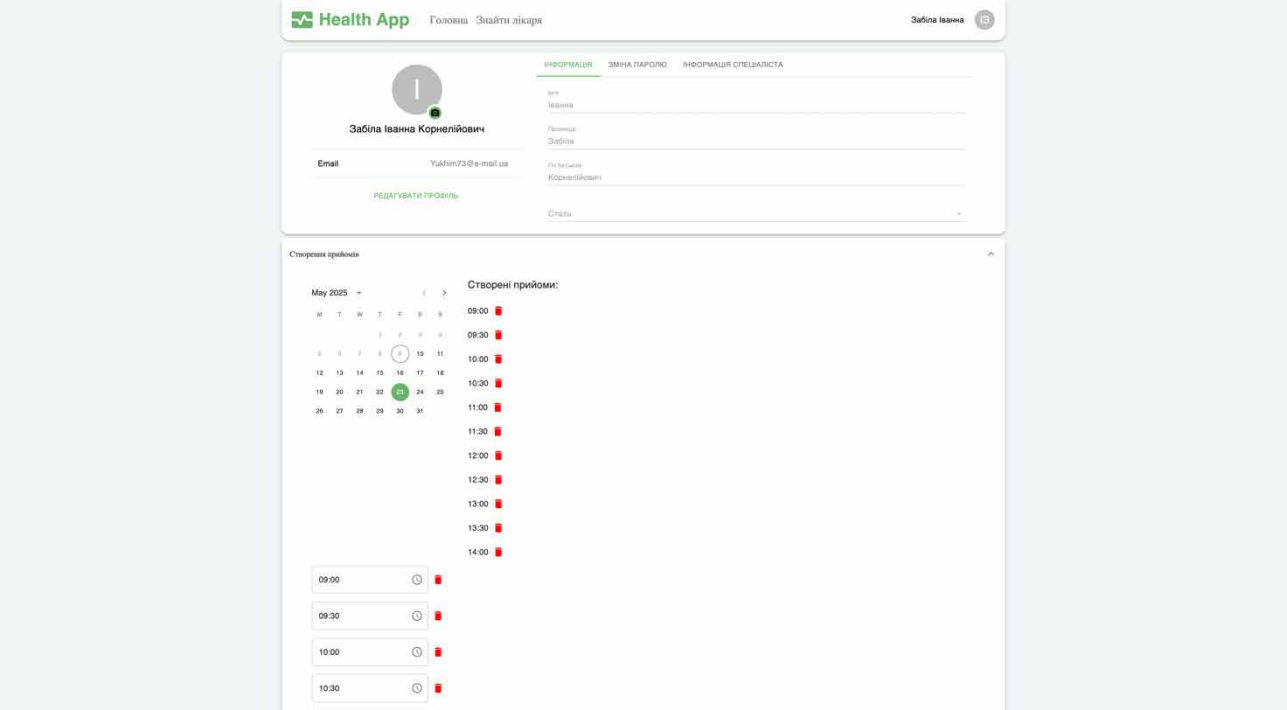
Рис. 20 Управління МКХ-10

Система містить окремий функціонал для керування класифікатором захворювань МКХ-10 (рис. 20), доступний лише адміністраторам. Цей модуль дозволяє підтримувати актуальну базу медичних діагнозів відповідно до міжнародного стандарту.

Інтерфейс дозволяє:

• Переглядати наявні записи з кодами та описами захворювань;

• Додавати нові коди вручну - із зазначенням алфавітно-цифрового позначення та медичного опису;



- Редагувати існуючі записи для уточнення або оновлення термінології;
- Видаляти застарілі або помилкові записи.

Цей підхід дає можливість підтримувати гнучку, розширювану структуру довідника, яка використовується лікарями при заповненні медичних карток. Усі зміни набирають чинності миттєво, а обмеження на дублювання кодів гарантують цілісність даних.

Рис. 21 Створення прийомів лікарем

У системі реалізовано зручний функціонал для самостійного створення розкладу прийомів лікарем (рис. 21). Кожен медичний працівник після входу в особистий кабінет має доступ до вкладки, де може вказати дати й конкретні години, у які він буде доступний для пацієнтів.

Інтерфейс складається з інтерактивного календаря та списку часових слотів, які лікар може додавати або видаляти за потреби. Для кожної дати створюється набір прийомів у вибрані години, які згодом відображаються пацієнтам під час пошуку лікаря й запису на прийом.

Завдяки такому підходу забезпечується автономність лікаря в плануванні роботи та зменшується навантаження на адміністративний персонал.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було повністю реалізовано інформаційну систему, призначену для автоматизації діяльності медичного закладу. Робота охоплювала повний цикл створення програмного забезпечення - від аналізу предметної області до побудови архітектури, моделювання бази даних, розробки користувацького інтерфейсу, впровадження та демонстрації ключових функцій. На етапі дослідження було вивчено існуючі аналоги медичних інформаційних систем, сформульовано технічні та функціональні вимоги до проєкту, а також визначено доцільність розробки нового рішення з урахуванням специфіки користувацьких сценаріїв взаємодії між пацієнтом, лікарем та адміністративною частиною закладу.

Вдалося створити повнофункціональну веб-систему, яка дозволяє пацієнтам проходити реєстрацію, шукати лікарів за спеціальністю, локацією або іншими критеріями, переглядати графік вільних годин та записуватись на прийом. З боку лікаря передбачено інструменти для створення власного розкладу, управління прийомами, формування діагнозів з використанням класифікатора МКХ-10, а також ведення електронної медичної картки пацієнта. Адміністративна частина системи реалізована як окремий функціональний блок, який дозволяє створювати, редагувати та видаляти профілі лікарів, підтримувати актуальність довідника МКХ-10, а також здійснювати загальне конфігураційне управління системою.

Під час реалізації було використано сучасні технології: серверна частина написана з використанням .NET 9 та ASP.NET Core, що забезпечує високу продуктивність і розширюваність API. Клієнтська частина побудована на основі React з використанням бібліотек Material UI для швидкого проєктування інтерфейсів, Redux для централізованого управління станом і Axios для організації запитів до бекенду. У якості системи управління базами даних застосовано Microsoft SQL Server, який забезпечує надійне зберігання структурованої інформації. Для роботи з файлами інтегровано об’єктне сховище MinIO. Усі сервіси розгортаються в контейнерах через Docker

Compose, що забезпечує простоту деплою, гнучке масштабування і стабільність середовища виконання.

Результати тестування підтвердили, що система працює стабільно, швидко реагує на дії користувача, захищає персональні дані та підтримує багаторівневий доступ залежно від ролі. Графічний інтерфейс вийшов інтуїтивно зрозумілим, адаптивним і придатним для використання на різних пристроях. Реалізований функціонал демонструє повну відповідність поставленим вимогам і забезпечує реальну користь для автоматизації базових процесів у медичних установах.

Отриманий результат має практичну цінність і може використовуватись як у навчальних цілях, так і як основа для реального впровадження в клінічну практику. У перспективі можлива розробка мобільного застосунку, розширення можливостей персоналізованої взаємодії між пацієнтом і лікарем, інтеграція з державними реєстрами медичних даних, а також додавання функціоналу телемедицини. Усі поставлені в роботі завдання були реалізовані в повному обсязі, а результати підтвердили доцільність і ефективність обраної архітектури та технологій розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UML Class Diagram Tutorial [Електронний ресурс]. - Режим доступу: https://www.lucidchart.com/pages/uml-class-diagram
2. Activity Diagrams – Unified Modeling Language (UML) [Електронний ресурс]. - Режим доступу: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/
3. What is Unified Modeling Language (UML)? [Електронний ресурс]. - Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
4. What is Class Diagram? [Електронний ресурс]. - Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/
5. All about UML package diagrams [Електронний ресурс]. - Режим доступу: https://www.lucidchart.com/pages/uml-package-diagram
6. Microsoft Learn.[Електронний ресурс]. – Режим доступу: https://learn.microsoft.com
7. Docker Docs [Електронний ресурс]. – Режим доступу: https://docs.docker.com/
8. Comprehensive Guide to UML Package Diagrams [Електронний ресурс]. – Режим доступу: https://www.archimetric.com/comprehensive-guide-to-uml-package-diagrams/

ДОДАТОК А

ПРОГРАМНИЙ КОД

Сторінок - 6

Бізнес логіка сервісу спеціалістів

```
internal class DoctorsService : IDoctorsService
{
    private readonly IUnitOfWork _uow;
    private readonly ICurrentUser _currentUser;
    private readonly IHttpContextAccessor _httpContextAccessor;
    private readonly IFilesSettingsService _filesSettings;

    public DoctorsService(
        IUnitOfWork uow,
        ICurrentUser currentUser,
        IHttpContextAccessor httpContextAccessor,
        IFilesSettingsService filesSettings)
    {
        _uow = uow;
        _currentUser = currentUser;
        _httpContextAccessor = httpContextAccessor;
        _filesSettings = filesSettings;
    }

    public async Task<ApiResult<DoctorDto>> Get(Guid id)
    {
        var result = new ApiResult<DoctorDto>();

        var doctor = await _uow.Repository<Doctor>()
            .Query()
            .WhereIf(_currentUser.IsAdmin, x => !x.IsDeleted)
            .Select(DoctorStaticMapper.DoctorProjection)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (doctor == null)
        {
            result.Status = StatusCodes.Status404NotFound;
            return result;
        }

        result.Status = StatusCodes.Status200OK;

        result.Data = doctor;

        return result;
    }

    public async Task<ApiResult<List<DoctorDto>>> Get(DoctorsFilter filter)
    {
        var result = new ApiResult<List<DoctorDto>>();

        var doctors = await _uow.Repository<Doctor>()
            .Query()
```

```
            .FilterDoctors(filter)
            .Select(DoctorStaticMapper.DoctorProjection)
            .PagedById(filter)
            .ToListAsync();

        foreach (var doctorDto in doctors)
        {
            doctorDto.Slots = await _uow.Repository<DoctorSlot>()
                .Query()
                .FilterSlots(doctorDto.Id)
                .Select(DoctorStaticMapper.DoctorSlotProjection)
                .GroupBy(x => x.Date.Date)
                .ToDictionaryAsync(x => x.Key, x => x.OrderBy(y =>
y.Date).ToList());
        }

        result.Data = doctors;

        result.Status = StatusCodes.Status200OK;

        return result;
    }

    public async Task<ApiResult<List<DoctorTableDto>>> GetTable(DoctorsFilter
filter)
    {
        var result = new ApiResult<List<DoctorTableDto>>();

        var query = _uow.Repository<Doctor>()
            .Query()
            .FilterDoctors(filter)
            .Select(DoctorStaticMapper.DoctorsTableProjection);

        result.Data = await query
            .PagedById(filter)
            .ToListAsync();

        var totalCount = await query.CountAsync();

        _httpContextAccessor.AddResponseHeader(ResponseHeaders.TotalCount,
totalCount.ToString());

        result.Status = StatusCodes.Status200OK;

        return result;
    }

    public async Task<ApiResult<Dictionary<DateTime, List<DoctorSlotDto>>>>
GetSlots(Guid doctorId)
    {
```

```
        var result = new ApiResult<Dictionary<DateTime, List<DoctorSlotDto>>>();

        result.Data = await _uow.Repository<DoctorSlot>()
            .Query()
            .Where(x => x.DoctorId == doctorId && x.Date >=
DateTime.UtcNow.Date)
            .Select(DoctorStaticMapper.DoctorSlotProjection)
            .GroupBy(x => x.Date.Date)
            .ToDictionaryAsync(x => x.Key, x => x.OrderBy(y =>
y.Date).ToList());

        result.Status = StatusCodes.Status200OK;

        return result;
    }

    public async Task<ApiResult> CreateDoctor(Guid id)
    {
        var result = new ApiResult();

        var account = await _uow.Repository<Account>()
            .Query()
            .Include(x => x.Doctor)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (account == null)
        {
            result.Status = StatusCodes.Status404NotFound;
            return result;
        }

        if (account.Doctor != null)
        {
            account.Doctor.IsDeleted = false;

            await _uow.SaveChangesAsync();

            result.Status = StatusCodes.Status204NoContent;
            return result;
        }

        account.Doctor = new Doctor
        {
            Location = "Change me!",
            Phone = "Change me!",
            Speciality = "Change me!",
        };

        await _uow.SaveChangesAsync();
```

```
        result.Status = StatusCodes.Status204NoContent;

        return result;
    }

    public async Task<ApiResult> CreateDoctorSlots(Guid doctorId,
List<DoctorSlotDto> dtoSlots)
    {
        var result = new ApiResult();

        if (_currentUser.DoctorId != doctorId && !_currentUser.IsAdmin)
        {
            result.Status = StatusCodes.Status403Forbidden;
            return result;
        }

        var doctor = await _uow.Repository<Doctor>()
            .Query()
            .FirstOrDefaultAsync(x => x.Id == doctorId);

        if (doctor == null)
        {
            result.Status = StatusCodes.Status404NotFound;
            return result;
        }

        var slots = dtoSlots
            .DistinctBy(x => x.Date)
            .Select(x => new DoctorSlot
            {
                Doctor = doctor,
                Date = x.Date.ToUniversalTime(),
                IsAvailable = true,
            })
            .ToList();

        var exists = _uow.Repository<DoctorSlot>()
            .Query()
            .Where(x => x.DoctorId == doctorId)
            .Any(x => slots.Select(y => y.Date).Contains(x.Date));

        if (exists)
        {
            result.Message = "Одна з дат вже зарезервована";
            return result;
        }

        _uow.Repository<DoctorSlot>().Add(slots);
```

```
        await _uow.SaveChangesAsync();

        result.Status = StatusCodes.Status201Created;

        return result;
    }

    public async Task<ApiResult> DeleteDoctor(Guid id)
    {
        var result = new ApiResult();

        var account = await _uow.Repository<Account>()
            .Query()
            .Include(x => x.Doctor)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (account == null)
        {
            result.Status = StatusCodes.Status404NotFound;
            return result;
        }

        if (account.Doctor == null)
        {
            return result;
        }

        account.Doctor.IsDeleted = true;

        await _uow.SaveChangesAsync();

        result.Status = StatusCodes.Status204NoContent;

        return result;
    }
}
```

Контрелер прийомів HTTP-запитів для прийомів

```
[Authorize]
public class AppointmentsController : BaseApiController
{
    private readonly IAppointmentsService _appointmentsService;

    public AppointmentsController(IAppointmentsService appointmentsService)
    {
        _appointmentsService = appointmentsService;
    }

    [HttpPost]
    [ProducesResponseType(StatusCodes.Status201Created, Type =
typeof(PatientAppointmentDto))]
    public async Task<IActionResult> Create([FromBody] AppointmentCreationDto
model)
    {
        return await _appointmentsService.CreateAsync(model);
    }

    [HttpPost("{id:guid}/rating")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> Rate(Guid id, [FromQuery] int rating)
    {
        return await _appointmentsService.RateAppointmentAsync(id, rating);
    }

    [HttpPost("{id:guid}/complete")]
    [HasPermission(IsDoctor = true)]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> Rate(Guid id)
    {
        return await _appointmentsService.CompleteAsync(id);
    }

    [HttpDelete("{id:guid}")]
    [HasPermission(IsDoctor = true)]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> Delete(Guid id)
    {
        return await _appointmentsService.DeleteAsync(id);
    }
}
```