МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти Перший (бакалаврський)
Спеціальність Комп'ютерні науки
Освітня програма Комп'ютерні науки
Група 6.04.122.010.18.1

ДИПЛОМНИЙ ПРОЄКТ

на тему: «Розроблення модуля «Реєстратура медичного закладу» на базі веб-технологій»

Виконала: студентка Юлія РЕШИТНЯКОВА

Керівник: к.е.н., доцент Олена ПЛОХА

Рецензент:
к.т.н, доцент кафедри «Кібербезпеки та інформаційних технологій»
Харківського національного економічного університету ім. С. Кузнеця
Іван МІХЄЄВ

Харків – 2022 рік
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ

РЕФЕРАТ

Пояснювальна записка до дипломного проекту: 63 с., 45 рис., 24 табл., 1 додаток, 35 джерел.

Об’єктом дослідження є процес розроблення та проектування web застосунку для запису пацієнтів до лікаря у медичних закладах шляхом використання web-технологій.

Предметом дослідження є технології та методи реалізації запису до лікаря в online-режимі.

Метою дипломного проекту є організація процедури запису до лікаря шляхом розроблення web-модуля.

Метод проектування – використання програмних систем WebStorm, Ramus Educational, Online Visual Paradigm, Moqups, PgAdmin. У розробці використовується фреймворк Nest.js для back-end частини та бібліотека React.js для front-end частини.

У результаті виконання проекту розроблено модуль програмного продукту під назвою «Your health», який дозволяє полегшити процес запису до лікаря та відстежувати навантаженість лікарів.

Результатами розробки з мінімальними змінами зможе користуватись будь-яка лікарня.

МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, РЕЄСТРАТУРА, CASE ДІАГРАМИ, БАЗА ДАНИХ, POSTGERSQL, REACTJS, NESTJS

ABSTRAC

Explanatory note to the diploma project: 63 pages, 45 figures, 24 tables, 1 appendix, 35 sources.

The object of research is the process of developing and designing a web application for enrolling patients in a medical institution through the use of web technologies.

The subject of the study is the technology and methods of implementing a doctor's appointment online.

The aim of the diploma project is to facilitate the procedure of registration with a doctor by developing a web-module.

Design method - using software systems WebStorm, Ramus Educational, Online Visual Paradigm, Moqups, PgAdmin. The development uses the Nest.js framework for the back-end part and the React.js library for the front-end part.

As a result of the project, a software product module called "Your health" was developed, which allows to facilitate the process of registration with a doctor and

monitor the workload of doctors.

Any hospital will be able to use the results of the development with minimal changes.

MEDICAL INFORMATION SYSTEM, REGISTRATION, CASE DIAGRAMS, DATABASE, POSTGERSQL, REACTJS, NESTJS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних
ВВ – варіант використання
ІЛМ – інформаційно-логічна модель
ІС – інформаційна система
ПЗ – програмне забезпечення
ПП – програмний продукт
СКБД – система керування базами даних
СУП – система, що управляється подіями

ВСТУП

У наш час ще досі існує проблема відсутності веб-застосунків або інших інформаційних систем (ІС) у багатьох лікарнях. У невеликих містах чи селах ще досі потрібно стояти у черзі до реєстратури щоб записатись на прийом чи просто дізнатись графік роботи [23].

Велика кількість медичних ІС є занадто важкими у користуванні, багато неоптимізовані або важкі у розгортанні, а за деякі додатково потрібно вносити плату. Також великою проблемою таких систем є те, що вони не універсальні.

Модуль «Реєстратура медичного закладу» вирішить усі вищесказані проблеми.

Метою дипломного проекту є організація процедури запису до лікаря шляхом розроблення універсального web-застосунку, яким при мінімальних змінах та термінах зможе почати користуватись будь-яка лікарня. Це також дозволить відстежувати навантаженість лікарні та окремих лікарів та комфортно керувати цими процесами.

Об’єктом дослідження є процес розроблення та проектування web застосунку для запису пацієнтів до лікаря у медичних закладах шляхом використання web-технологій.

Предметом дослідження є технології та методи реалізації запису до лікаря

в online-режимі.

В ході дипломного проектування необхідно вирішити наступні задачі: проаналізувати предметну область за допомогою використання методології IDEF0;

порівняти вже існуючі рішення щодо організаційної підтримки вантажних перевезень, визначити їх переваги та недоліки;

побудувати UML-діаграму варіантів використання (ВВ) та розробити специфікації варіантів використання;

розробити зовнішній вигляд екранних форм – розкадровку;

сформувати специфікацію функціональних та нефункціональних вимог до модуля;

скласти словник даних, визначити обмеження та розробити базу даних (БД);

визначити інструменти та засоби програмної реалізації бази даних, шаблони проектування та технології реалізації програмного продукту (ПП); побудувати діаграму класів та діаграму станів;

розробити програмний продукт;

перевірити якість програмного продукту, виконавши його тестування;

провести розгортання програмного продукту;

провести аналіз якості організації і безпеки робочого місця на підприємстві, а також оцінку заходів з охорони праці.

Таким чином, в ході дипломного проектування буде розроблено модуль «Реєстратура для медичного закладу». Система має пройти через всі обов'язкові етапи життєвого циклу програмного забезпечення: аналіз предметної області та визначення вимог до продукту, проектування, розробка, тестування та розгортання.

Матеріали дипломного проєкту пройшли апробацію на міжнародній науково-практичній конференції молодих учених, аспірантів та студентів «Інформаційні технології в сучасному світі: дослідження молодих вчених» 17 – 18 лютого 2022 р. в м. Харкові [23].

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ «РЕЄСТРАТУРА МЕДИЧНОГО ЗАКЛАДУ»

1.1. Коротка характеристика бази практики ТОВ «АЙТІ ПРОСТІР»

ТОВ «АЙТІ ПРОСТІР» – ІТ компанія, що більше 15 років займається розробкою веб-додатків, мобільних додатків та технічною підтримкою своїх клієнтів.

Щодо структурних особливостей бази практики, головний офіс складається з великої кількості відділів: дирекції, менеджменту, відділу документації, бухгалтерії, відділу роботи з клієнтами, підрозділів, що займаються розробкою веб-додатків, підрозділів розробки мобільних додатків, у тому числі додатків доповненої реальності, відділу тестування, а також системних та серверних адміністраторів.

У кожному підрозділі працюють від 5 до 30 працівників, а саме: проектний менеджер, менеджери з роботи з клієнтами та розробники і тестувальники програмного забезпечення.

Організація структурних підрозділів компанії налагоджена на високому рівні. Це зумовлено співпрацею різних відділів для успішного виконання поставлених клієнтами завдань.

Загальна структура підприємства наведена на Рис. 1.1. Головним об’єктом структури є дирекція, яка керує усіма процесами підприємства; відділи розробки веб-додатків, мобільних додатків, додатків доповненої реальності займаються розробкою відповідного програмного забезпечення; відділ системних та серверних адміністраторів відповідає за налагодження та підтримку систем інших відділів та підрозділів; відділ документації займається документообігом підприємства, а бухгалтерія – усім, що пов’язано з фінансами.

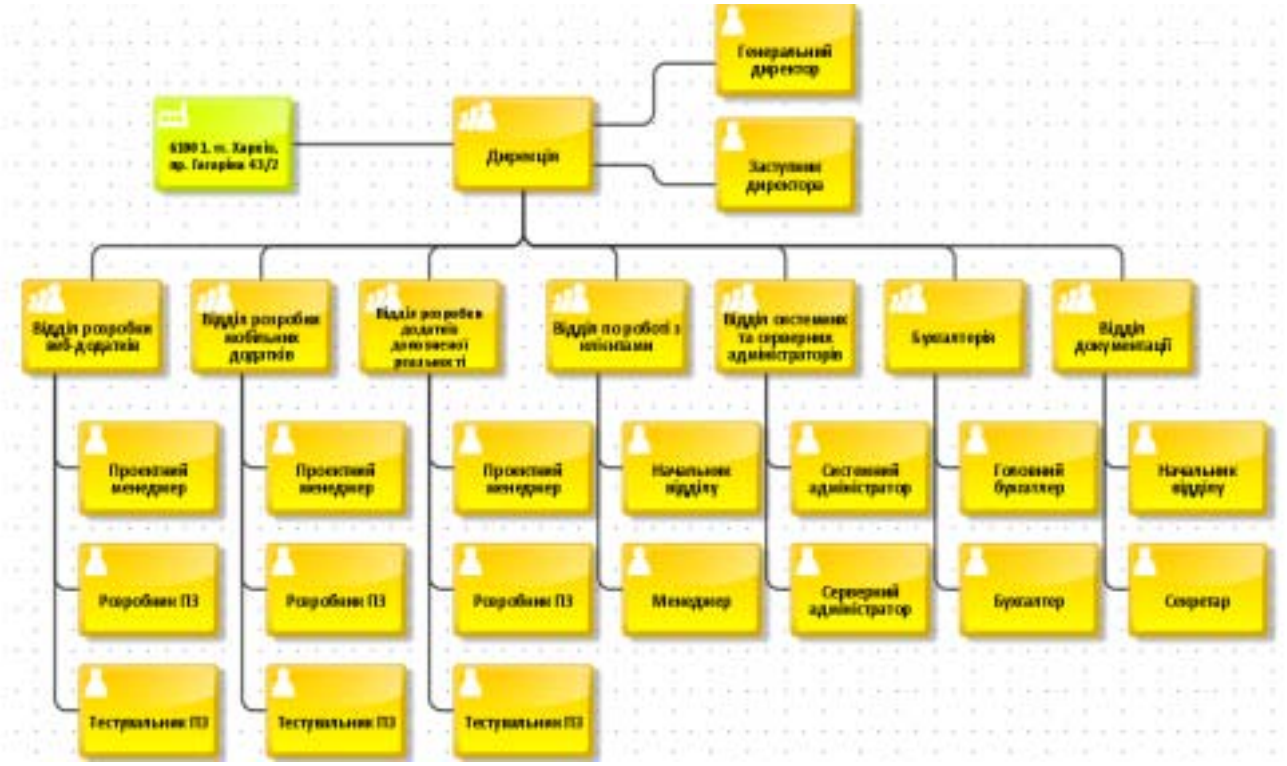

Рис. 1.1. Організаційна структура управління ТОВ «Айті Простір»

Під час проходження практики на підприємстві мене було закріплено за відділом розробки веб-додатків. У відділі працюють близько 25 працівників: проектний менеджер, менеджери по роботі з клієнтами та розробники і тестувальники програмного забезпечення різного рівня. Загальна характеристика структури відділу наведена на Рис. 1.2.

Підприємство ТОВ «АЙТІ ПРОСТІР» займається розробкою програмного забезпечення та його подальшою підтримкою під час використання замовником. Замовниками або клієнтами компанії в основному виступають закордонні бізнес організації, які потребують у програмному забезпечені для власного функціонування або обслуговування своїх клієнтів.

Відділ розробників веб-додатків після отримання замовлення на розробку програмного забезпечення від клієнта будує технічну задачу (обов’язок проектного менеджера), обирає найбільш зручні та ефективні методи та технології її виконання (обов’язок архітектора) та розподіляє між розробниками відділу їх завдання (обов’язок старшого розробника).

Під час щоденних мітингів старший розробник та проектний менеджер вислуховують звіти розробників щодо їх роботи та радяться з кожним працівником відділу щодо покращення ефективності загальної роботи відділу для виконання поставленої перед ним задачі.

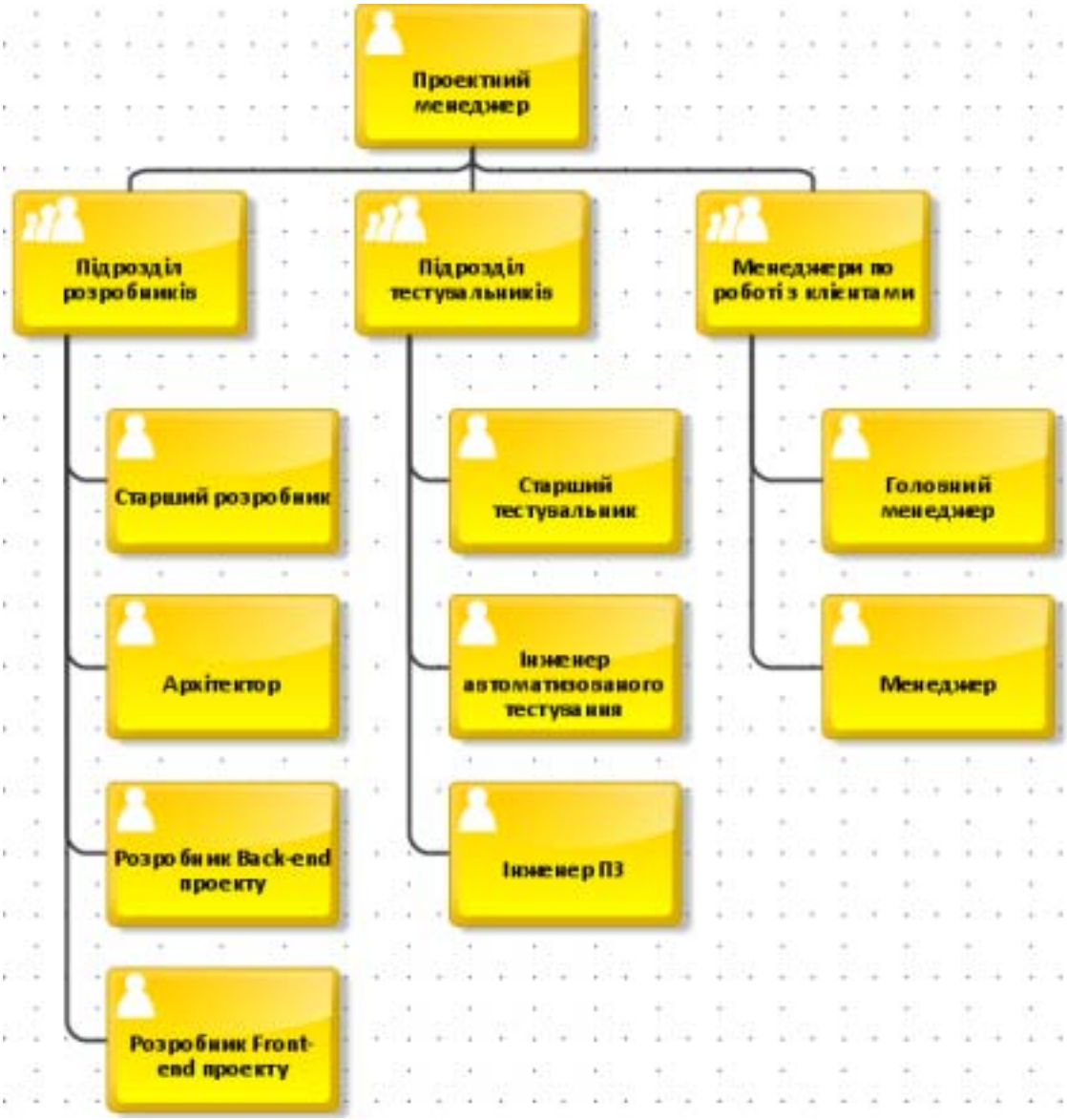

Рис. 1.2. Структура відділу розробки веб-додатків

Щодня проектний менеджер відділу звітує дирекції компанії щодо прогресу виконання завдання та оцінює рівень відповідності поточних досягнень до укладеного до початку розробки проекту графіка виконання ТЗ.

З цього можна зробити висновок, що структура відділу дає змогу отримати найкращий результат під час роботи, а завдяки щоденному спілкування учасників команди їм одразу вдається виявити наявність проблем та почати вживати заходів для їх вирішення.

1.2. Опис предметної області розроблення web-застосунку «Реєстратура медичного закладу»

В рамках дипломного проєкту, що розробляється, було розглянуто тему

«Реєстратура медичного закладу» що допоможе як пацієнтам, так і лікарям не витрачати час, стоячи в чергах та зберегти своє здоров'я, не знаходячись у великому скупченні людей. Web-застосунок «Реєстратура медичного закладу» зможе взяти у користування будь-яка лікарня та полегшити цим життя співробітників та пацієнтів.

Для описання роботи web-застосунку було обрано IDEF0-діаграми [16, 30]. Це – методологія функціонального та графічного описання систем, призначена для формалізації та опису бізнес-процесів. Характеристика бізнес-процесу «Реєстратура медичного закладу» наведена у Таблиця 1.1 [1]. Контекстна діаграма наведена на Рис. 1.3. Декомпозиція бізнес-процесу наведена на Рис. 1.4.

Таблиця 1.1

Характеристика бізнес-процесу «Реєстратура медичного закладу»

Назва характеристики	Значення характеристики
Ім'я бізнес-процесу	«Реєстратура медичного закладу»
Основні учасники	Адміністратор, Лікар, Пацієнт
Вхідна подія	Авторизація
Вхідний документ	Список лікарів, Розклад роботи лікарів, Інформація про пацієнта, Запит пацієнта, Логін та пароль користувача
Вихідна подія	Перегляд записів до лікаря
Вихідний документ	Талон на прийом до лікаря, Список записів до лікаря

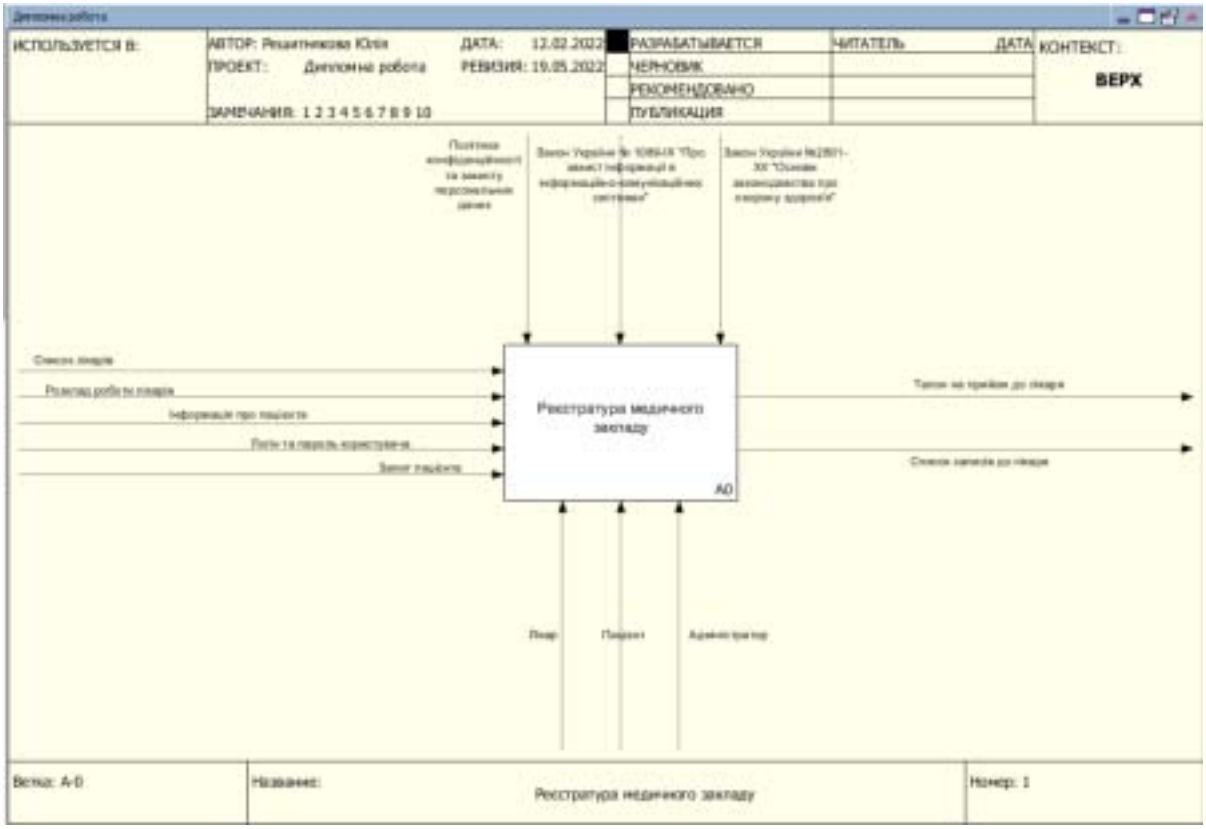

Рис. 1.3. Контекстна діаграма процесу «Реєстратура медичного закладу»

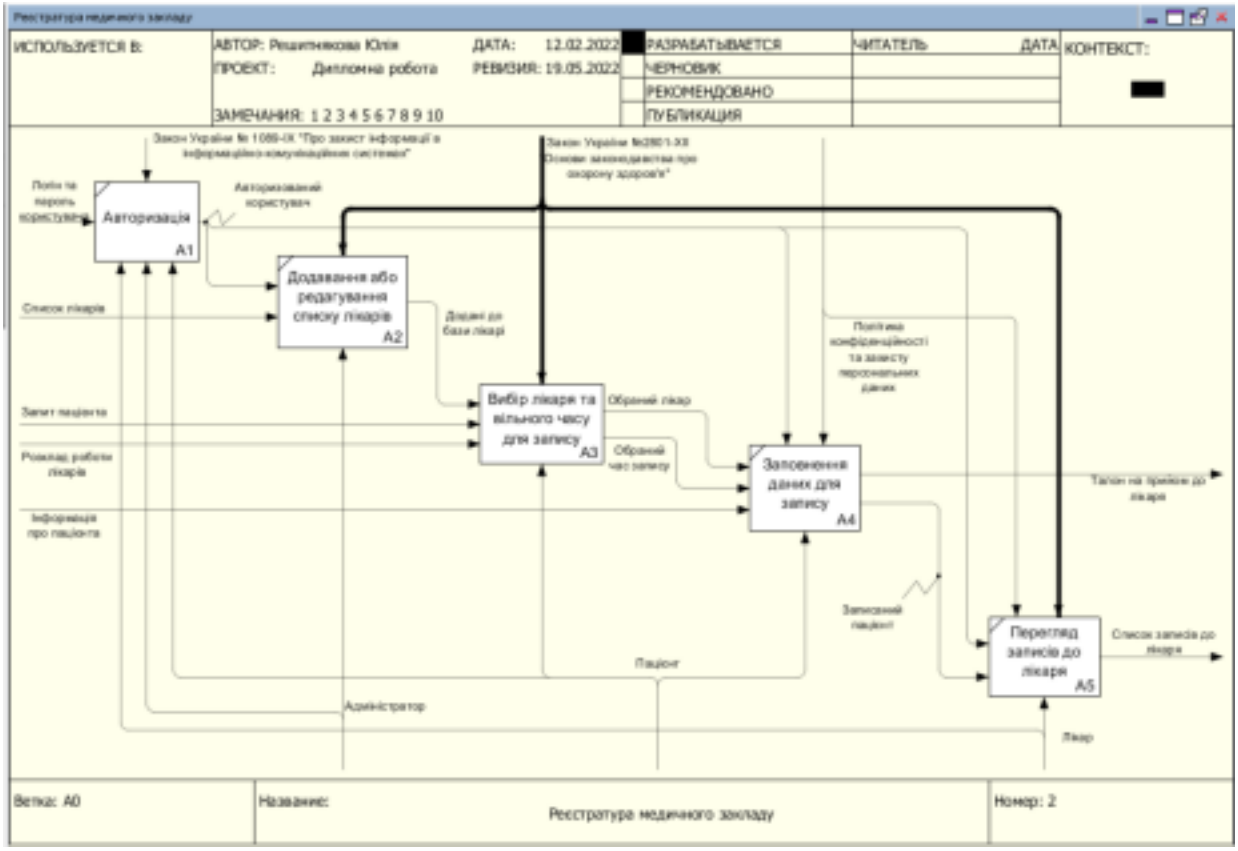

Рис. 1.4. Декомпозиція процесу «Реєстратура медичного закладу»

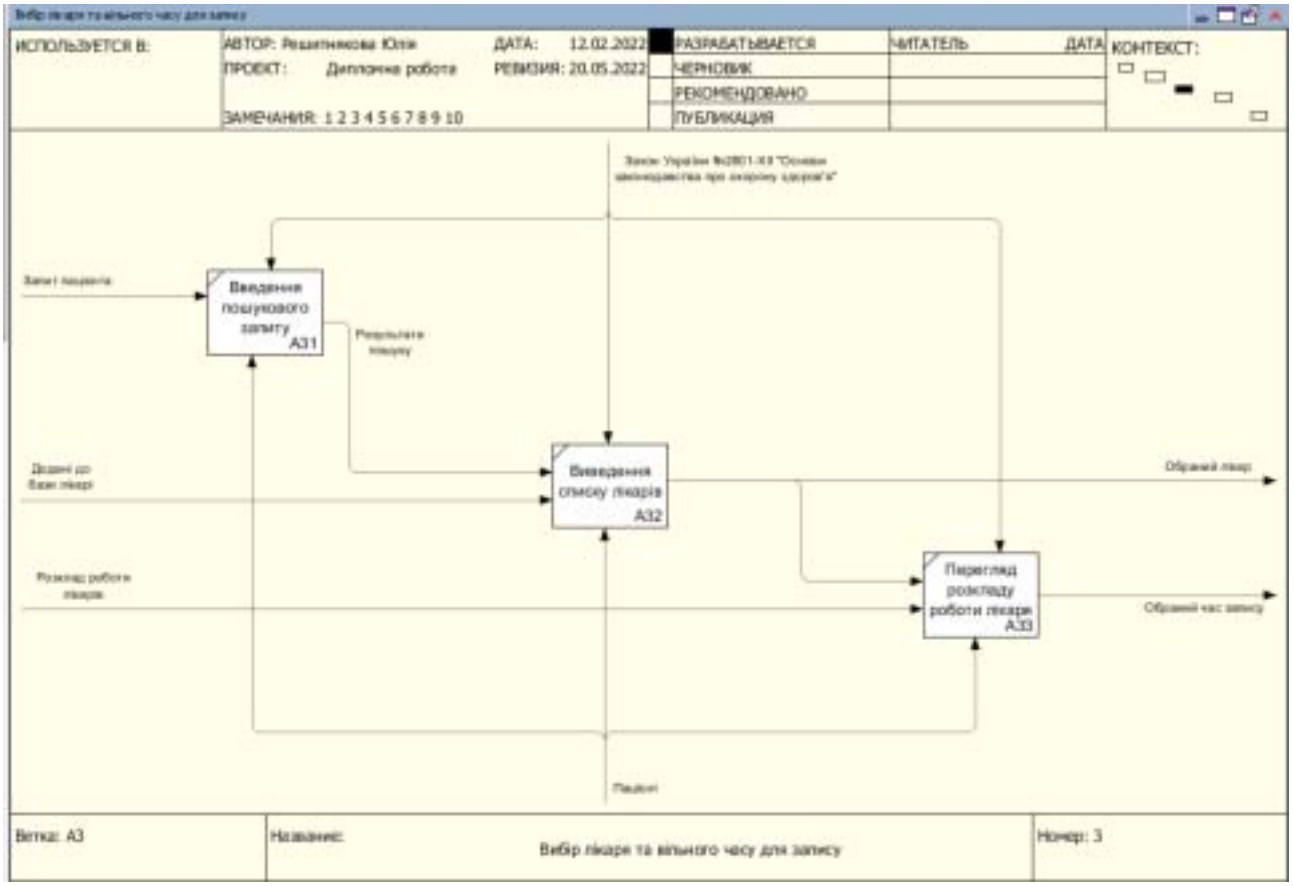

Рис. 1.5. Декомпозиція процесу А3 «Вибір лікаря та вільного часу для запису»

Таким чином, під час аналізу предметної області було виявлено процеси, які потребують автоматизації.

1.3. Огляд і аналіз існуючих аналогів, що реалізують функції предметної області

Після проведення аналізу предметної області було виявлено три аналоги для online-запису пацієнтів. Це:

- медична інформаційна система Helsi [29];
- поліклініка без черг Global New Medicine [21];
- сервіс для запису до лікаря My Med Cabinet [18].

Медична інформаційна система Helsi (helsi.me) базована на державній системі охорони здоров'я eHealth, має багато різного функціоналу, серед якого також є online-запис до лікаря. Головна сторінка Helsi зображена на Рис. 1.6.

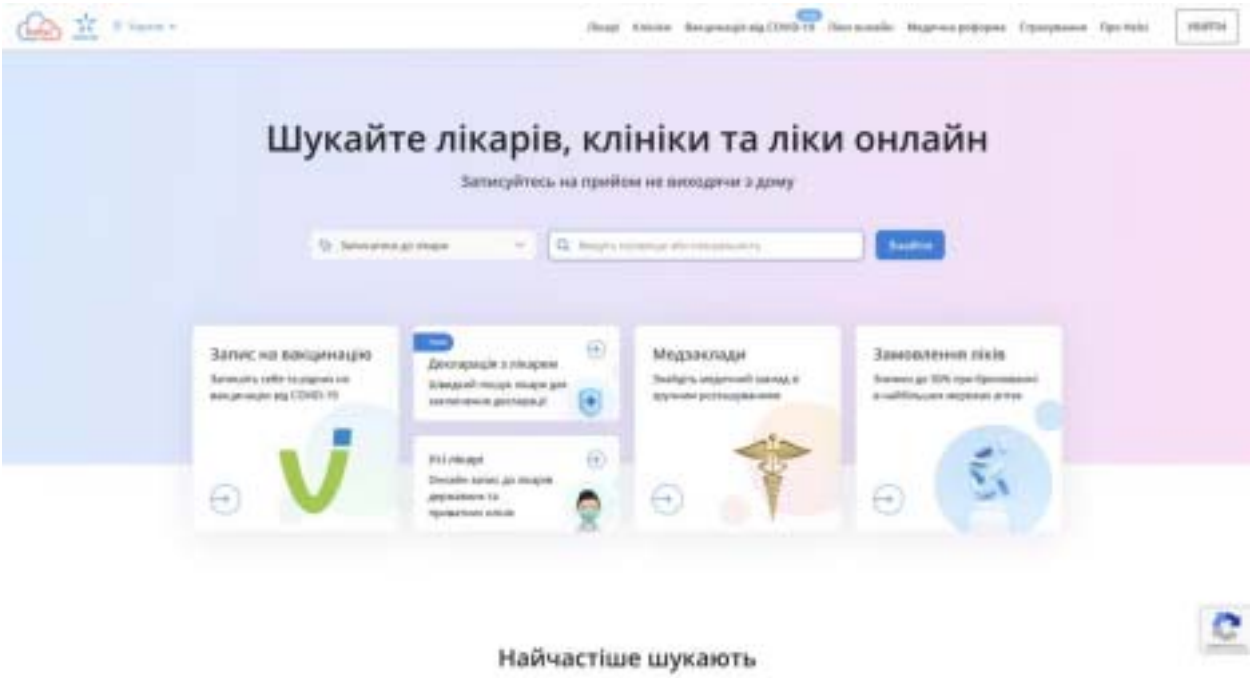

Рис. 1.6. Головна сторінка helsi.me

Є функція пошуку за ПІБ або спеціальністю лікаря. За запитом «Терапевт» маємо такий результат (див. Рис. 1.7). За пошуком також є велика кількість фільтрів.

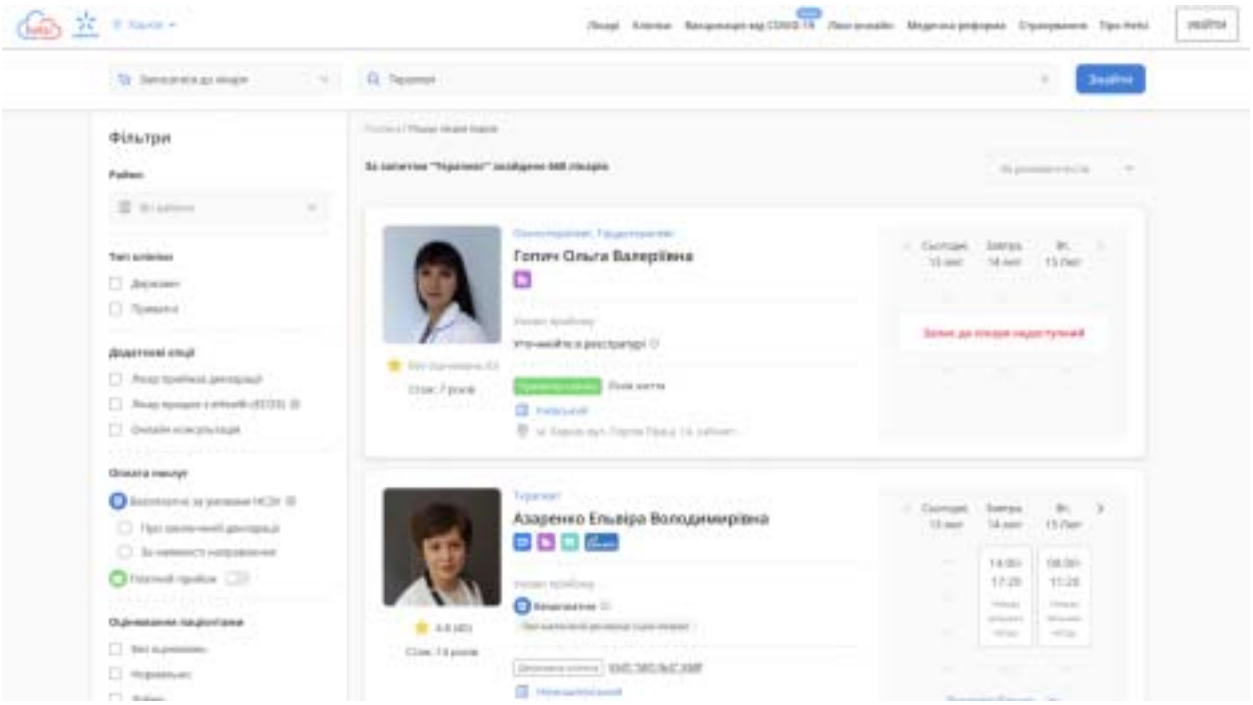

Рис. 1.7. Результати пошуку за запитом «Терапевт» у helsi.me

Для вибору лікаря потрібно авторизуватись у системі. Вікно авторизації знаходиться на Рис. 1.8.

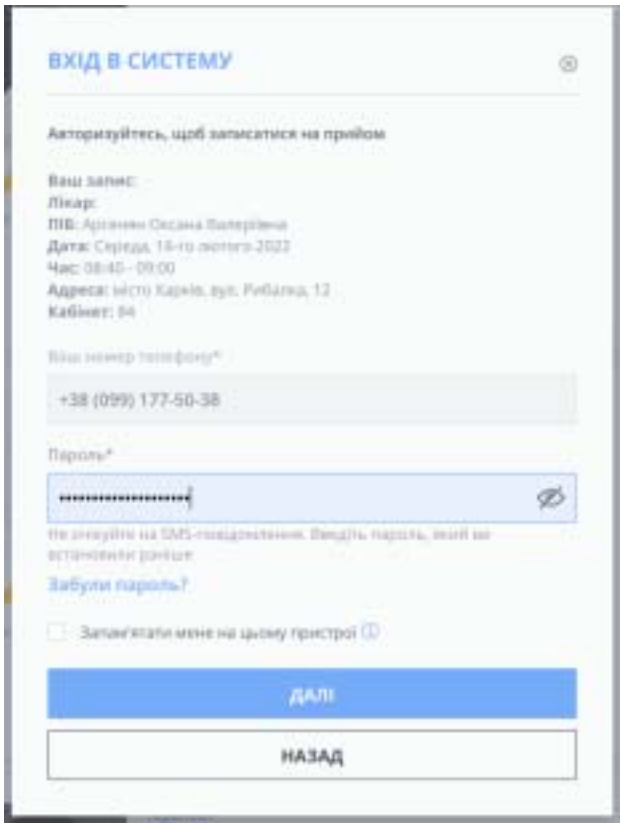

Рис. 1.8. Вікно авторизації у Helsi

В особистому кабінеті можна буде знайти ваш запис та отримати талон (Рис. 1.9).

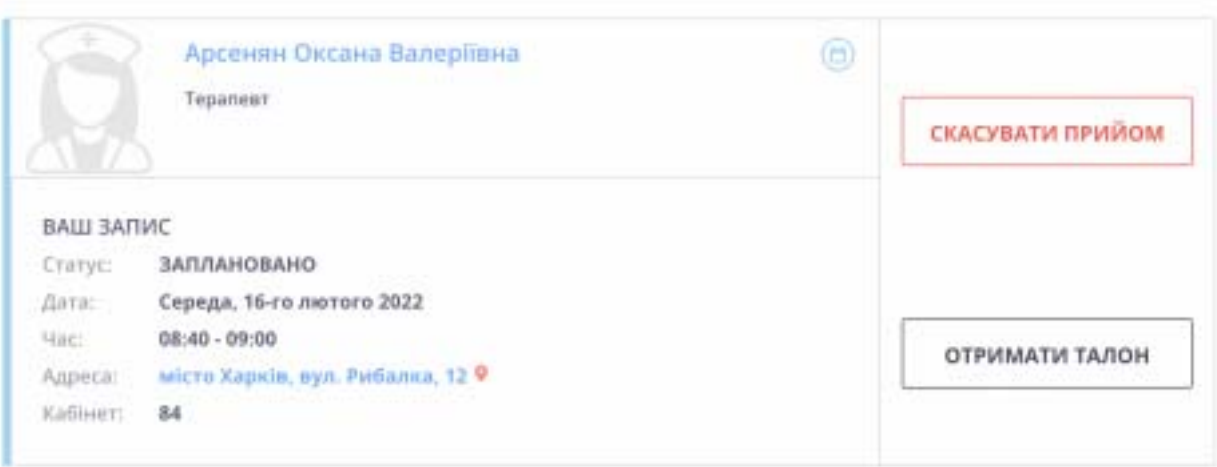

Рис. 1.9. Запис до лікаря у Helsi

З мінусів: ця медична система працює на eHealth [28], тож підключена до Центральної бази даних, а це означає, що ваш медичний заклад повинен пройти деяку перевірку перед підключенням до Helsi. Це тривала процедура. Також не всі приватні лікарні будуть згодні підключатись до Центральної бази даних.

Поліклініка без черг Global New Medicine спеціалізується тільки на online записах до лікаря. Головне меню сайту представлено на Рис. 1.10.

Рис. 1.10. Головна сторінка Global New Medicine

Одразу до мінусів – сайт не запам’ятовує чи був користувач авторизований. Після кожного виходу з сайту потрібно авторизуватись знову шляхом відповіді на телефонний дзвінок. Вікно підтвердження відображене на Рис. 1.11.

Можна вибрати лікарню та записатись до лікаря. Після вибору лікарні вікно запису виглядає таким чином (див. Рис. 1.12). І ще один недолік – після перезавантаження сторінки URL змінюється на сторінку профілю. Вікно з інформацією про запис подане на Рис. 1.13.

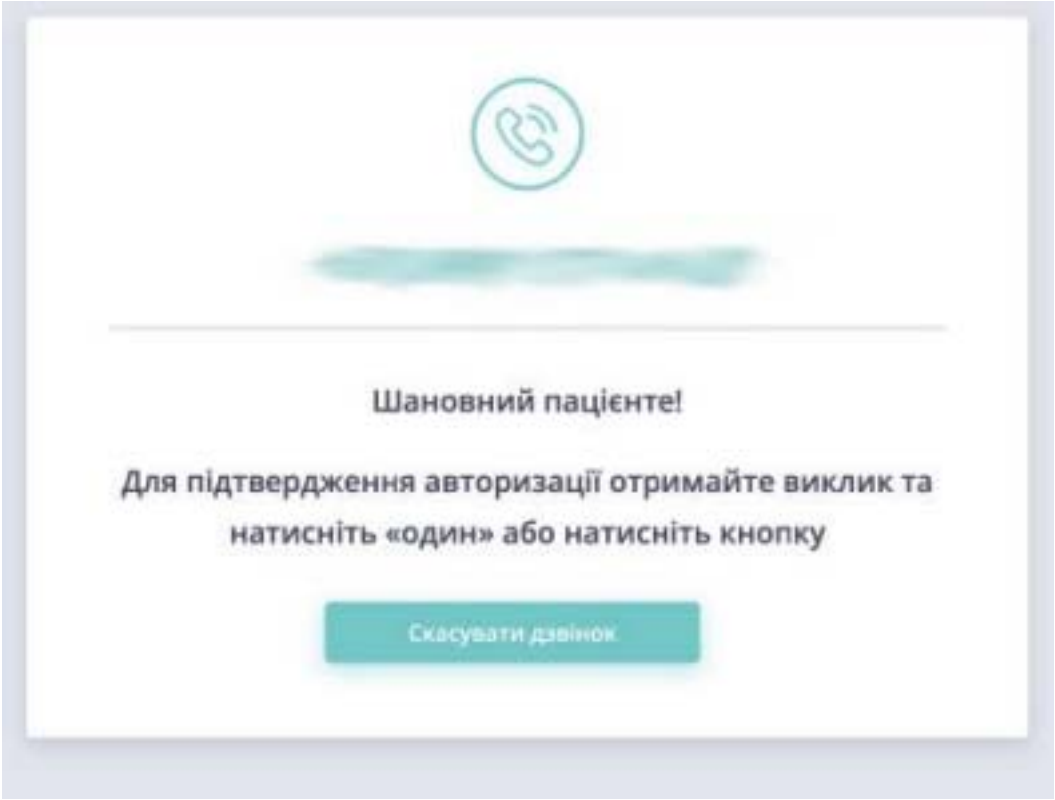

Рис. 1.11. Вікно з підтвердженням авторизації шляхом телефонування (Global New Medicine)

Рис. 1.12. Вікно запису до лікаря

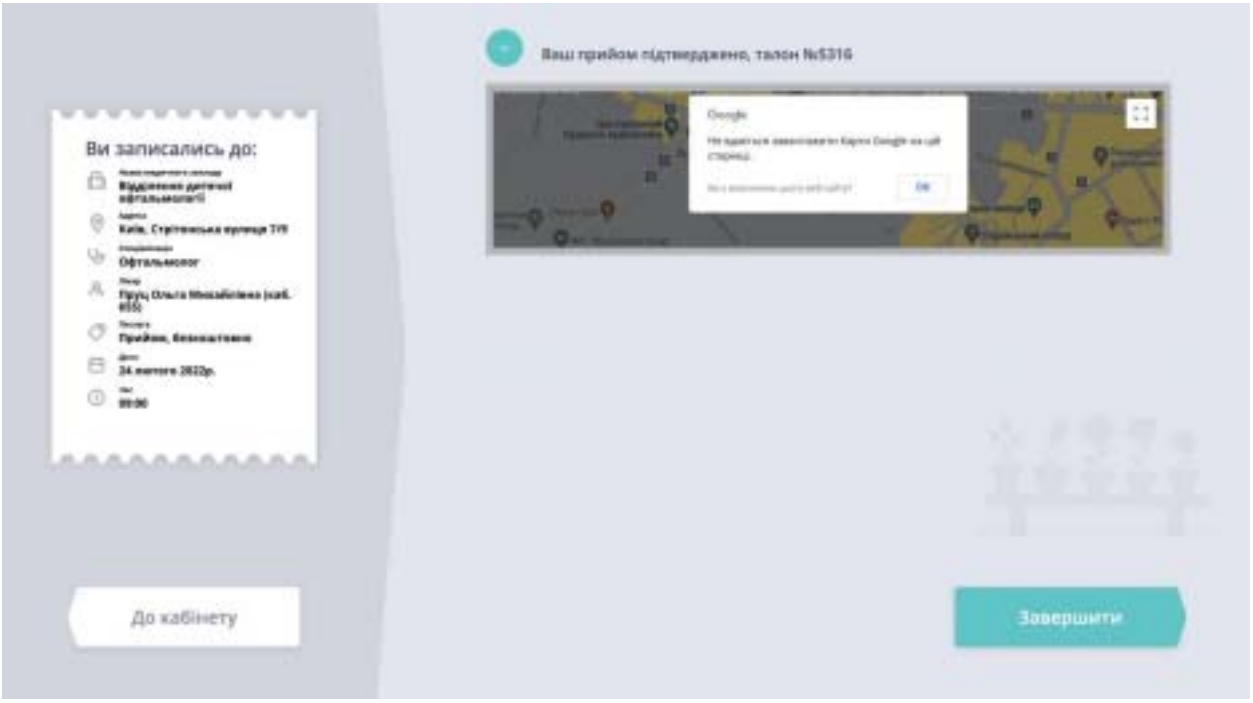

Рис. 1.13. Вікно з інформацією про запис

Також додаток має можливість відмінити запис у профілі користувача (див. Рис. 1.14)

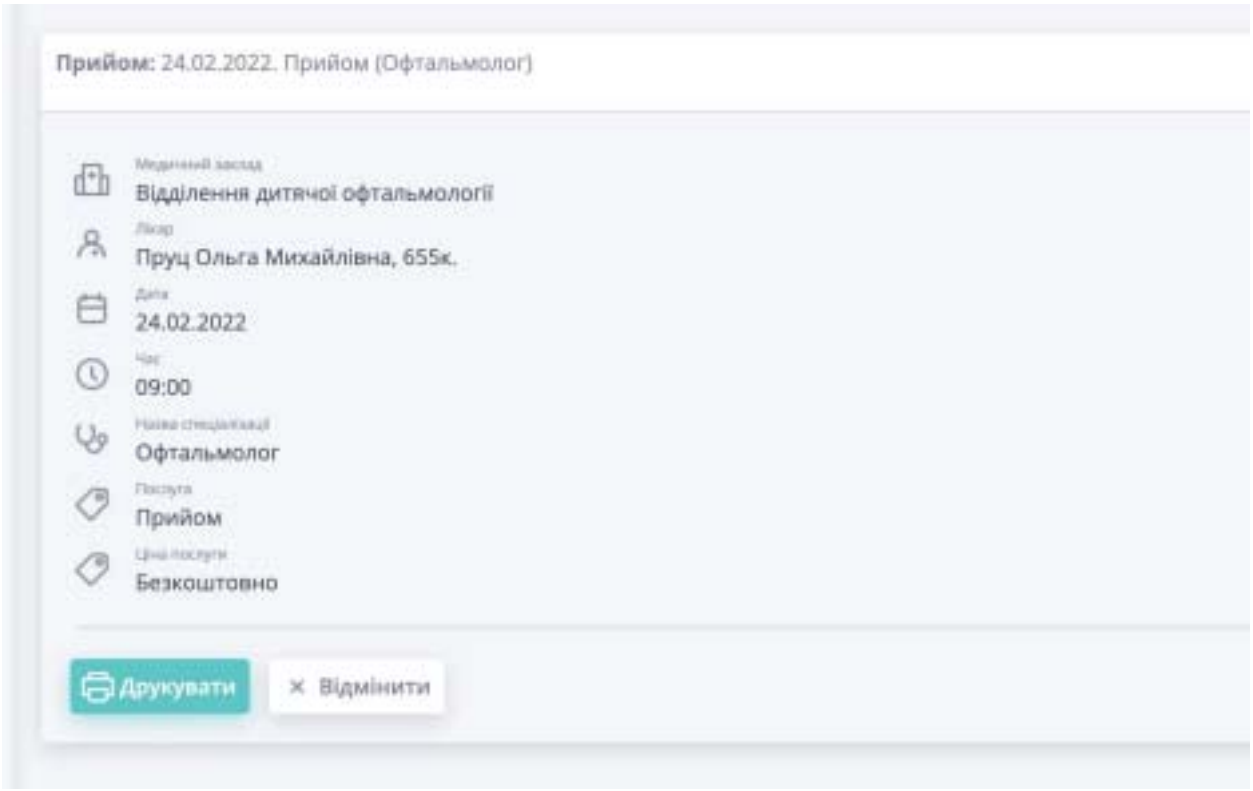

Рис. 1.14. Вікно для відміни запису до лікаря

My Med Cabinet – сервіс для запису до лікаря з додатковими послугами у

виді пошуку аптек з потрібними ліками. Головна сторінка сайту зображена на Рис. 1.15.

Рис. 1.15. Головна сторінка My Med Cabinet

Після вибору лікарні можна обрати лікаря та записатись на вільний час. Функція відхилення запису також присутня. (див. Рис. 1.16).

Рис. 1.16. Запис до лікаря у My Med Cabinet

Порівняльна характеристика web-сайтів наведена в табл. 1.2.

Таблиця 1.2

Порівняльна характеристика web-сайтів

Назва web-сайту	Helsi	Global New Medicine	My Med Cabinet
Посилання	https://helsi.me/	https://global.newmedic ine.com.ua/	https://mymedcabinet.c om.ua/

Коректне відображення сайту в різних браузерах при різних налаштуванняx екрану монітора	+	+	+
Актуальність ключової інформації	+	+	+
Реклама	+	-	-
Перехід по навігації сайту	+	-	+
Пошук	+	+	+
Швидкість розгортання та підключення медичного закладу то медичної системи	-	+	+
Додаткові можливості	+	-	+

Проаналізувавши функціональні та нефункціональні характеристики розглянутих аналогів, можна зробити висновок, що програмні продукти Helsi та My Med Cabinet найбільш універсальні та корисні для використання медичними закладами. В залежності від потреб можна обрати потрібний варіант. Але вони мають в собі такі недоліки, як наявність реклами, погана навігація по сайту та швидкість розгортання.

Модуль «Реєстратура медичного закладу» буде універсальним застосунком, який не потребуватиме довгого навчання для користування ним, матиме зручний та зрозумілий інтерфейс. В найкоротші строки будь-яка лікарня зможе почати користування ним, а це один з найважливіших критеріїв гарної розробки.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО МОДУЛЯ

2.1. Глосарій

Глосарій – це словник термінів, складений для розуміння певної специфічної області знань [6]. Основні терміни та їх визначення, які використовуються у дипломному проекті, наведено у табл. 2.1.

Таблиця 2.1

Глосарій

Термін	Опис терміну
1	2
1. Основні поняття та категорії предметної області та проекту	
Реєстратура	Організаційно-технічна система забезпечення підвищення якості медичних послуг для планування госпіталізації або прийому пацієнтів та безчергового звернення за медичною допомогою
Медична інформаційна система	Спеціалізоване програмне забезпечення, розроблене спеціально для потреб системи охорони здоров'я, яке характеризується одночасним зберіганням та обробкою особистої, демографічної та медичної інформації пацієнтів
Статус запису до лікаря	Актуальна інформація про запис пацієнта до лікаря (відхилено, заплановано, завершено)
2. Користувачі системи	
Пацієнт	Людина, яка хоче звернутись до лікаря за медичною допомогою (лікування, консультування, медичне спостереження та ін.)
Лікар	Спеціаліст з вищою медичною освітою, який займається діагностуванням, профілактикою та лікуванням різного роду хвороб
Адміністратор	Людина, яка керує відображенням лікарського складу у застосунку, яка має право редагувати, додавати, видаляти дані про лікарів, переглядати записи до лікаря та змінювати статус запису пацієнта до лікаря
3. Вхідні та вихідні документи	

Список лікарів	Таблиця з даними про лікарів, такі як ПІБ, електронна пошта, номер телефону, посада, графік роботи та номер кабінету, у якому приймає лікар

Закінчення табл.2.1

1	2
Розклад роботи лікарів	Таблиця з даними, у якій вказаний час роботи кожного з лікарів медичного закладу.
Інформація про пацієнта	Такі дані, як ПІБ, номер телефону, електронна пошта
Логін та пароль користувача	Набір символів, за допомогою якого можна авторизувати та автентифікувати користувача для входу до особистого кабінету
Запит пацієнта	Текст, який пацієнт вводить у спеціальне поле для того, щоб знайти потрібного лікаря. Це може бути ПІБ або посада лікаря
Талон на прийом до лікаря	Документ, за допомогою якого лікар зможе ідентифікувати та прийняти пацієнта
Список записів на прийом до лікаря	Дані, в яких вказано вільний та зайнятий час для запису до лікаря за його розкладом

2.2. Розроблення варіантів використання

Діаграма варіантів використання (ВВ) – діаграма, що описує, який функціонал програмної системи, що розробляється, доступний кожній групі користувачів [2].

ВВ застосовуються для специфікації загальних особливостей поведінки системи або будь-якої іншої сутності предметної області, без розгляду внутрішньої структури цієї сутності. Кожен варіант використання визначає дії, які повинні бути виконані системою, що проектується при її взаємодії з відповідним актором [3].

2.2.1. Розроблення діаграми варіантів використання.

Діаграма варіантів використання представлена на Рис. 2.1

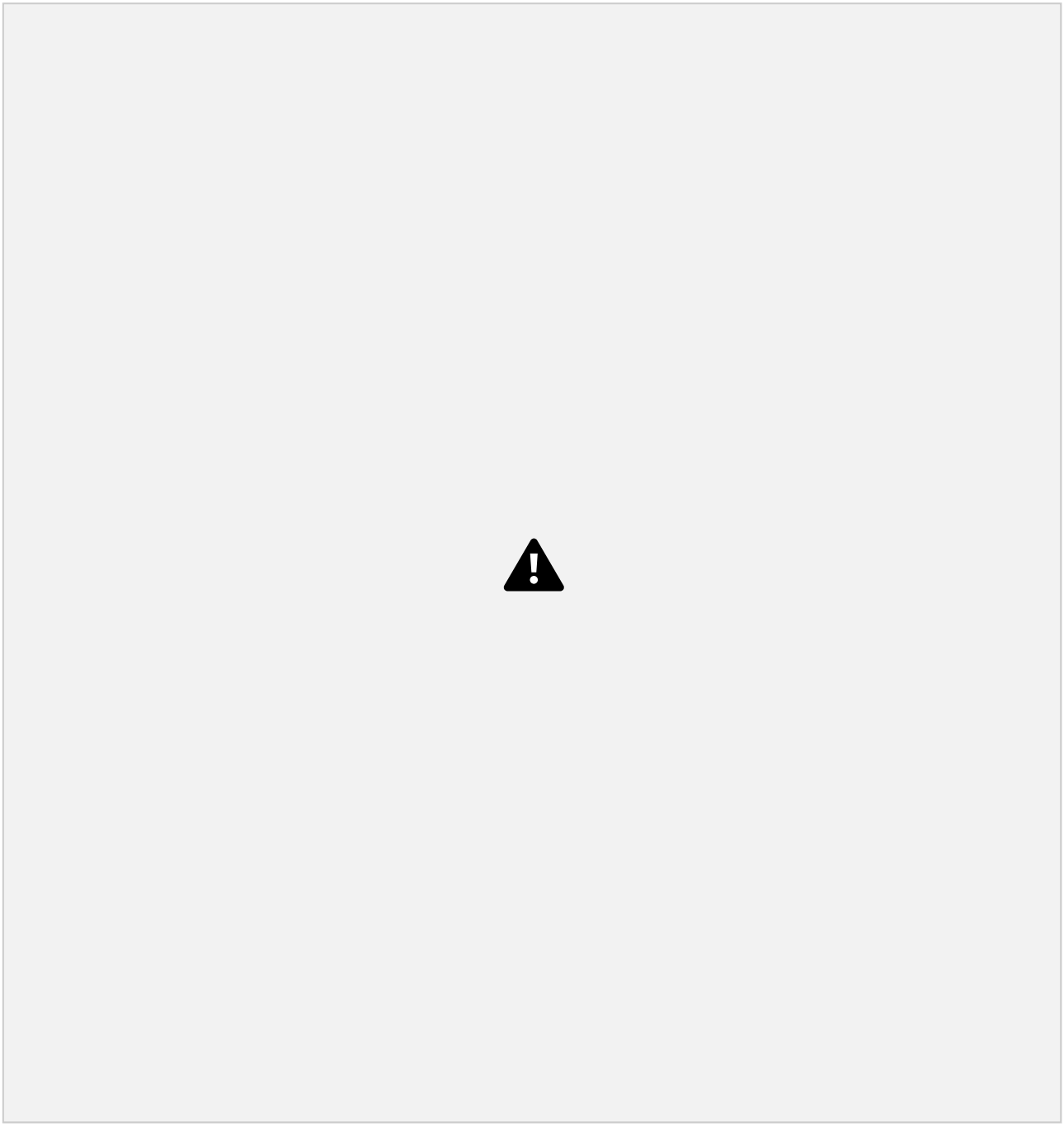

Рис. 2.1. Діаграма варіантів використання

2.2.2. Специфікація варіантів використання

Опис основних варіантів використання наведений у табл. 2.2 – 2.6.

Таблиця 2.2

Варіант використання «Вхід в систему»

Контекст використання	Авторизація в системі

Дійові особи	Пацієнт, адміністратор, лікар
Передумова	Користувач зареєстрований в системі
Тригер	Необхідно увійти на веб-сайт
Сценарій	1 - Введення логіну 2 - Введення паролю 3 - Натискання кнопки «Log in»
Постумова	У випадку виконання варіанта використання користувачу надані можливості системи в межах його ролі; у випадку помилки стан системи залишається незмінним, програма відображає повідомлення, яке містить причину помилки

Таблиця 2.3

Варіант використання «Оформлення запису до лікаря»

Контекст використання	Пацієнт хоче записатись на прийом до лікаря
Дійові особи	Пацієнт
Передумова	Користувач увійшов в систему
Тригер	З’явилась необхідність записатись на прийом до лікаря
Сценарій	1 - Ввести пошуковий запит для пошуку лікаря (ПІБ або посаду) 2 - Натискання кнопки «Search» 3 - Вибір потрібного лікаря зі списку 4 - Вибір вільного для запису часу у лікаря 5 - Перевірка заповнених даних 6 - Натискання кнопки «Make an appointment with a doctor»
Постумова	Відображається повідомлення про успішне створення запису або, якщо створити запис не вдалося, відображається повідомлення з текстом помилки. Також при успішному створенні в кабінеті користувача з’являється інформація про запланований запис

Таблиця 2.4

Варіант використання «Внесення особистої інформації до профілю користувача»

Контекст використання	Збереження даних про пацієнта у системі для майбутнього комфортного запису до лікарів

Дійові особи	Пацієнт
Передумова	Користувач увійшов в систему. Користувач натиснув кнопку «Profile»
Тригер	З’явилась необхідність у додаванні інформації про себе у систему для комфортного користування застосунком
Сценарій	1 - Введення інформації про пацієнта 2 - Натискання кнопки «Save changes»
Постумова	У випадку виконання варіанта використання інформація про перевізника збережена в ІС; у випадку помилки стан системи залишається незмінним, програма відображає повідомлення, яке містить причину помилки

Таблиця 2.5

Варіант використання «Додавання лікарів»

Контекст використання	Додавання інформації про нових лікарів у систему
Дійові особи	Адміністратор
Передумова	Користувач увійшов в систему
Тригер	Необхідно додати інформацію про нового лікаря
Сценарій	1 - Натискання кнопки «Add doctor» 2 - Введення інформації про лікаря у формі, що з’явилась 3 - Натискання кнопки «Add doctor»
Постумова	У випадку успішного виконання ВВ з’являється повідомлення про успішне додавання нового лікаря та оновлюється список лікарів. Якщо додати інформацію не вдалося з’являється повідомлення з причиною помилки

Таблиця 2.6

Варіант використання «Редагування графіку роботи лікаря»

Контекст використання	Редагування графіку роботи лікаря
1	2
Дійові особи	Лікар
Передумова	Користувач увійшов в систему
Тригер	Необхідно відредагувати графік роботи

Закінчення табл. 2.6

1	2
Сценарій	1 - Натискання кнопки «Profile» 2 - Натискання кнопки «Change schedule» 3 - Редагування необхідної інформації у формі, що з’явилась 4 - Натискання кнопки «Save changes»
Постумова	У випадку успішного редагування з’являється повідомлення про успішне редагування графіку. Також оновиться інформація у профілі користувача. Якщо не вдалося відредагувати графік роботи на екрані буде відображено повідомлення з текстом помилки.

2.3. Специфікація функціональних та нефункціональних вимог

Вимоги до програмних продуктів або інформаційних систем можна поділити на великі групи. Це функціональні вимоги (ті, що описують, що необхідно реалізувати в продукті або системі, в тому числі які дії повинні виконувати користувачі при взаємодії з ними) та нефункціональні вимоги (ті, що описують, як має працювати система чи програмний продукт, і якими властивостями чи характеристиками вона повинна володіти) [4].

Функціональні вимоги пояснюють, що має бути зроблено. Вони ідентифікують завдання чи дії, які мають бути виконані чи які мають бути здатними до виконання системою [12]. Як правило, говорячи про нефункціональні вимоги, найчастіше говорять про атрибути якості, тобто вимоги, що визначають якісні характеристики програмного забезпечення або системи, що розробляються, такі як продуктивність, надійність, масштабованість [19].Специфікація функціональних вимог наведена в Таблиця 2.7. Специфікація нефункціональних вимог наведена в Закінчення табл. 2.7

1	2	3	4	5
UC-5	Вибір часу прийому	обов’язкова	низька	Головний лікар
UC-6	Введення персональних даних для запису	обов’язкова	середня	Головний лікар
UC-7	Внесення особистої інформації до профілю користувача	обов’язкова	середня	Головний лікар
UC-8	Перегляд записів пацієнта на	рекомендована	середня	Головний лікар

	прийом			
UC-9	Скасування прийому	рекомендована	низька	Головний лікар
UC-10	Додавання лікарів	обов’язкова	середня	Головний лікар
UC-11	Зміна статусу прийому	обов’язкова	низька	Головний лікар

UC-12	Видалення даних	обов’язкова	середня	Головний лікар
UC-13	Перегляд даних	обов’язкова	середня	Головний лікар
UC-14	Збереження даних	обов’язкова	середня	Головний лікар

Таблиця 2.8.

Таблиця 2.7

Специфікація функціональних вимог

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Складність	Контакт
1	2	3	4	5
UC-1	Вхід в систему	обов’язкова	середня	Головний лікар
UC-2	Оформлення запису до лікаря	обов’язкова	висока	Головний лікар
UC-3	Пошук лікаря	обов’язкова	середня	Головний лікар
UC-4	Вибір лікаря	обов’язкова	низька	Головний лікар

Закінчення табл. 2.7

1	2	3	4	5
UC-5	Вибір часу прийому	обов’язкова	низька	Головний лікар
UC-6	Введення персональних даних для запису	обов’язкова	середня	Головний лікар
UC-7	Внесення особистої інформації до профілю користувача	обов’язкова	середня	Головний лікар
UC-8	Перегляд записів пацієнта на прийом	рекомендована	середня	Головний лікар

UC-9	Скасування прийому	рекомендована	низька	Головний лікар
UC-10	Додавання лікарів	обов’язкова	середня	Головний лікар
UC-11	Зміна статусу прийому	обов’язкова	низька	Головний лікар
UC-12	Видалення даних	обов’язкова	середня	Головний лікар
UC-13	Перегляд даних	обов’язкова	середня	Головний лікар
UC-14	Збереження даних	обов’язкова	середня	Головний лікар

Таблиця 2.8

Специфікація нефункціональних вимог

	Назва вимоги	Атрибути вимог

Ідентифікатор вимоги		Пріоритет	Складність	Контакт
1	2	3	4	5
1. Застосовність				
NFR-1-1	Час, необхідний для навчання користувачів, не повинен перевищувати 5 хвилин	обов’язкова	середня	Головний лікар
NFR-1-2	Зручний та функціональний інтерфейс	обов’язкова	середня	Головний лікар
2. Надійність				
NFR-2-1	Безперебійна цілодобова робота програми	обов’язкова	низька	Головний лікар
NFR-2-2	Максимальна кількість помилок на 1000 рядків коду не більше 1	рекомендована	висока	Головний лікар
3. Робочі характеристики				
NFR-3-1	Місткість системи має бути вищою за 100 000 користувачів	обов’язкова	середня	Головний лікар

Продовження табл. 2.8

1	2	3	4	5
4. Експлуатаційна придатність				
NFR-4-1	Код програми має бути написаний в одному стилі	обов'язкова	низька	Головний лікар
NFR-4-2	Відображення дат має бути у форматі дд.мм.рррр, де дд – це двозначне число дати, мм – це двозначне число місяця, а рррр – це чотиризначне число року	обов'язкова	середня	Головний лікар
5. Проектні обмеження				
NFR-5-1	Всі бібліотеки повинні бути у стабільній версії	обов'язкова	середня	Головний лікар
NFR-5-2	Програмний продукт має бути легко розширюваним	обов'язкова	середня	Головний лікар
6. Вимоги до документації, призначеної для користувача, і до системи допомоги.				
NFR-6-1	Всі повідомлення мають бути в одному стилі	обов'язкова	низька	Головний лікар
7. Куповані компоненти				
NFR-7-1	Система не використовує будь-які куповані компоненти	обов'язкова	середня	Головний лікар
8. Інтерфейс користувача				
NFR-7-1	Інтерфейс має бути дружнім, користувач має бачити на екрані кожну дію від позначки про завантаження, до відображення повідомлень з помилками	обов'язкова	середня	Головний лікар
NFR-7-2	Інтерфейс має бути інтуїтивним та не вимагати більше 5 хвилин для навчання роботі у системі	обов'язкова	середня	Головний лікар
NFR-7-3	Всі об'єкти інтерфейсу мають бути в одному стилі, який буде приємним для ока та не матиме велику кількість кольорів	обов'язкова	середня	Головний лікар

9. Комунікаційні інтерфейси				
NFR-8-1	Передача даних здійснюється за допомогою протоколу HTTPS	обов’язкова	низька	Головний лікар

Закінчення табл. 2.8

1	2	3	4	5
10. Вимоги до ліцензування				
NFR-9-1	Всі використовувані бібліотеки повинні мати Apache, MIT або Microsoft Public ліцензію	обов’язкова	середня	Головний лікар

2.4. Проєктування інтерфейсу користувача

Користувацький інтерфейс – це способи та засоби взаємодії користувача з програмами [20].

Вайрфрейм – це образ дизайну низької точності [20]. Він має чітко показувати:

- основну групу контенту (що?);
- структуру інформації (де?);
- опис та базову візуалізацію взаємодії між інтерфейсом та користувачем (як?).

Вайрфрейми не є безглуздим набором сірих блоків, хоча часто він виглядає саме так. Зазвичай, його розглядають як скелет майбутнього дизайну. Вайрфрейми мають зображати кожну деталь фінального продукту. Вайрфрейм варіанту використання «Вхід в систему» наведений на рис. 2.2.

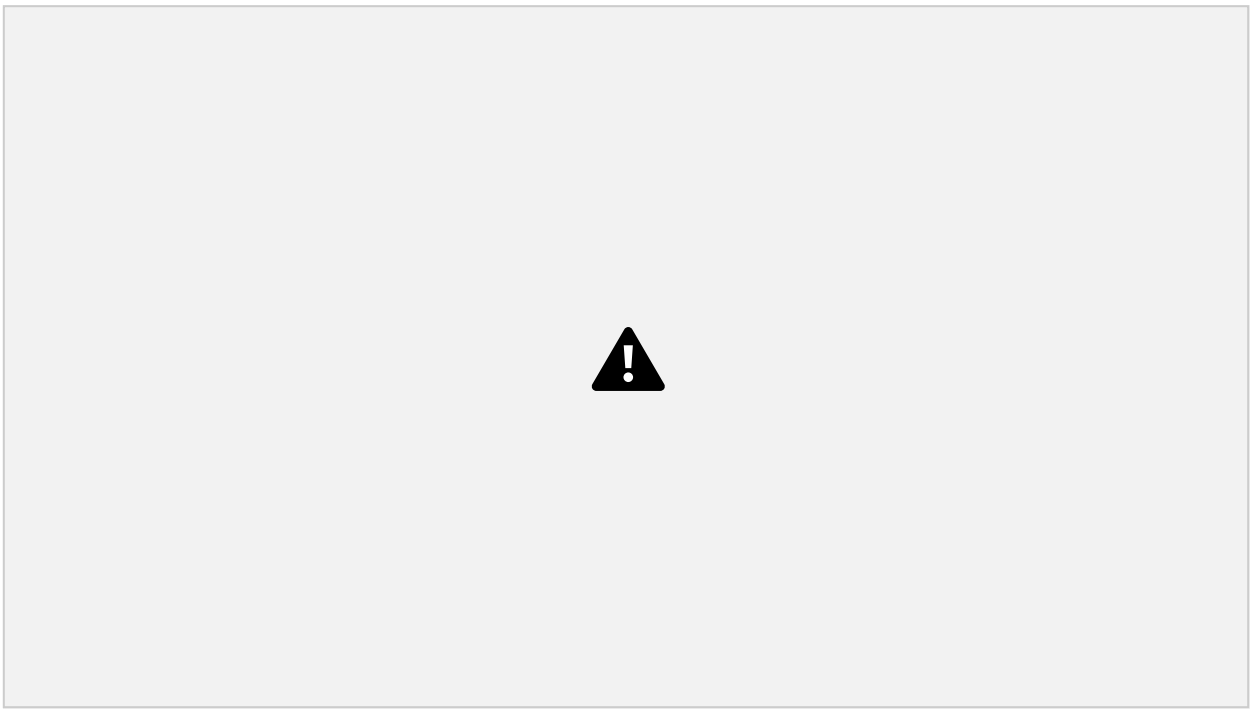

Рис. 2.2. Вайрфрейм «Вхід в систему»

Структура головного меню програми наведена на рис. 2.3.

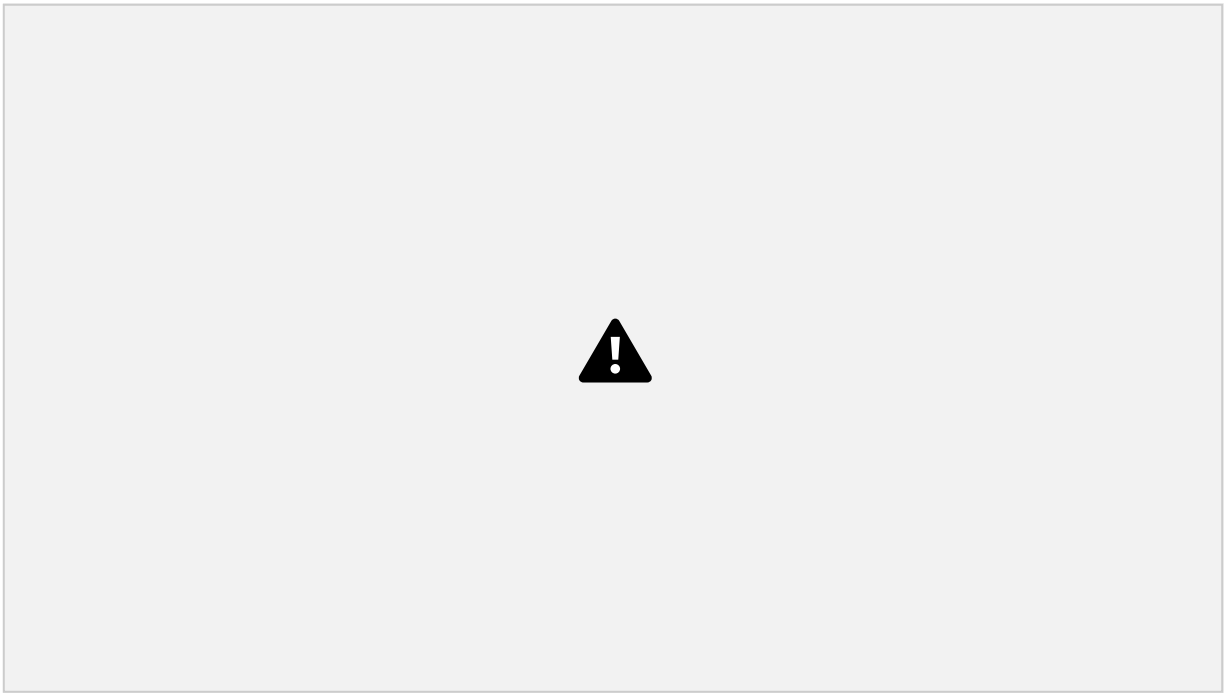

Рис. 2.3. Структура головної сторінки сайту

Вайрфрейми основних ВВ зображені на рис. 2.4 – 2.12

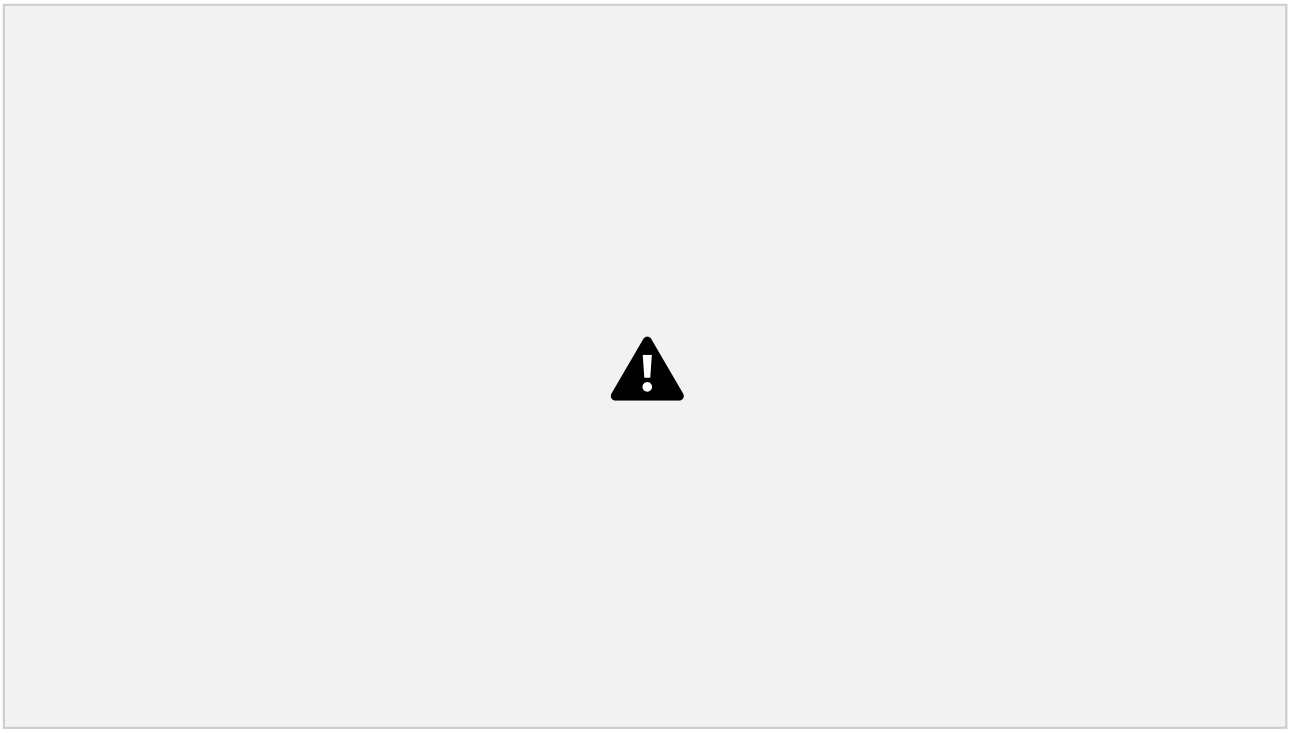

Рис. 2.4. Вайрфрейм «Пошук лікаря»

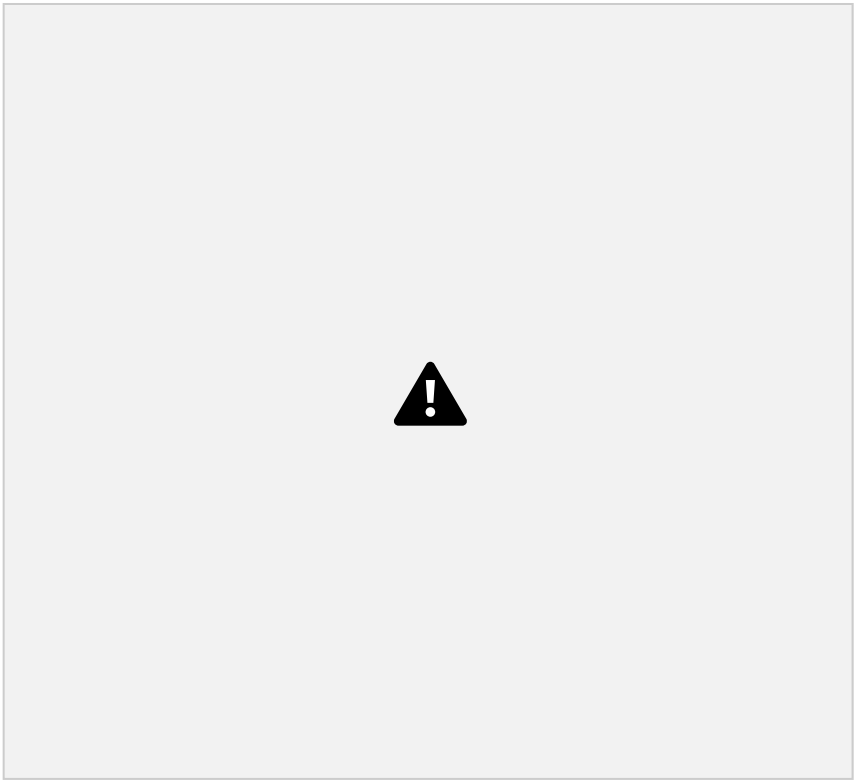

Рис. 2.5. Вайрфрейм «Підтвердження запису до лікаря»

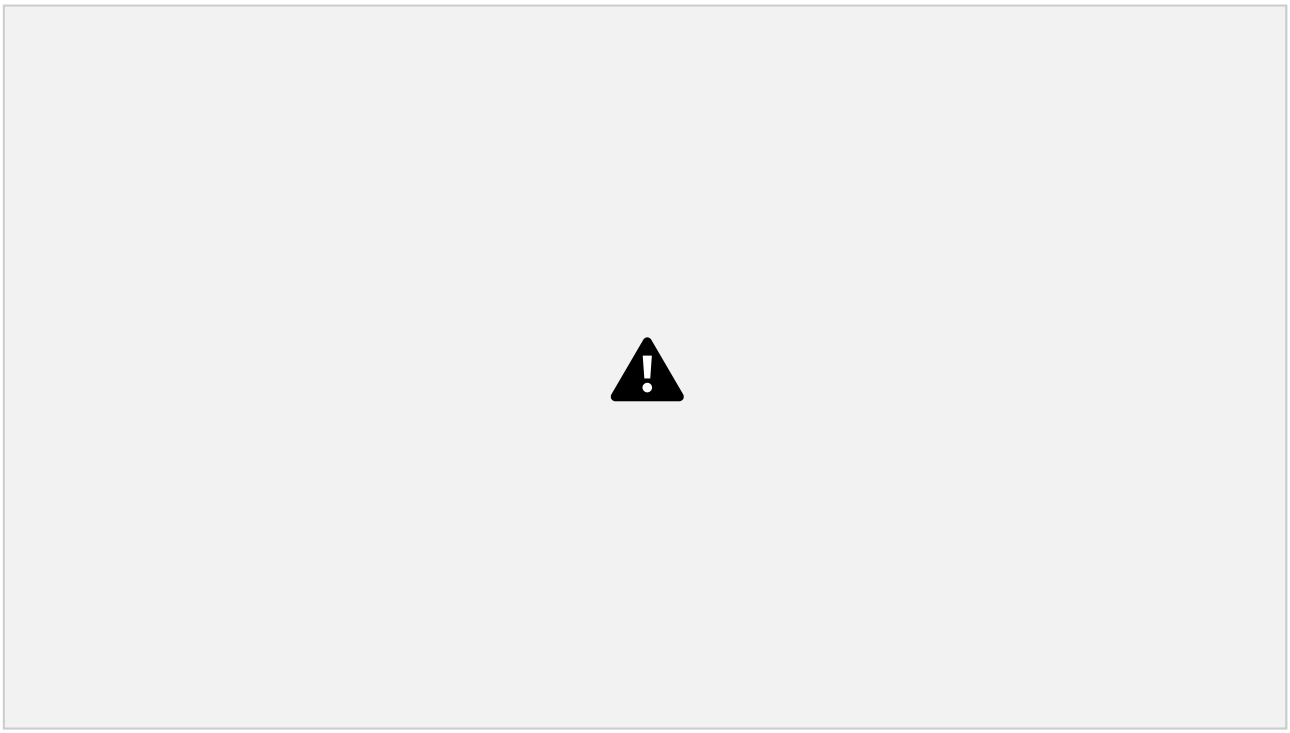

Рис. 2.6. Вайрфрейм «Внесення особистої інформації до профілю користувача»

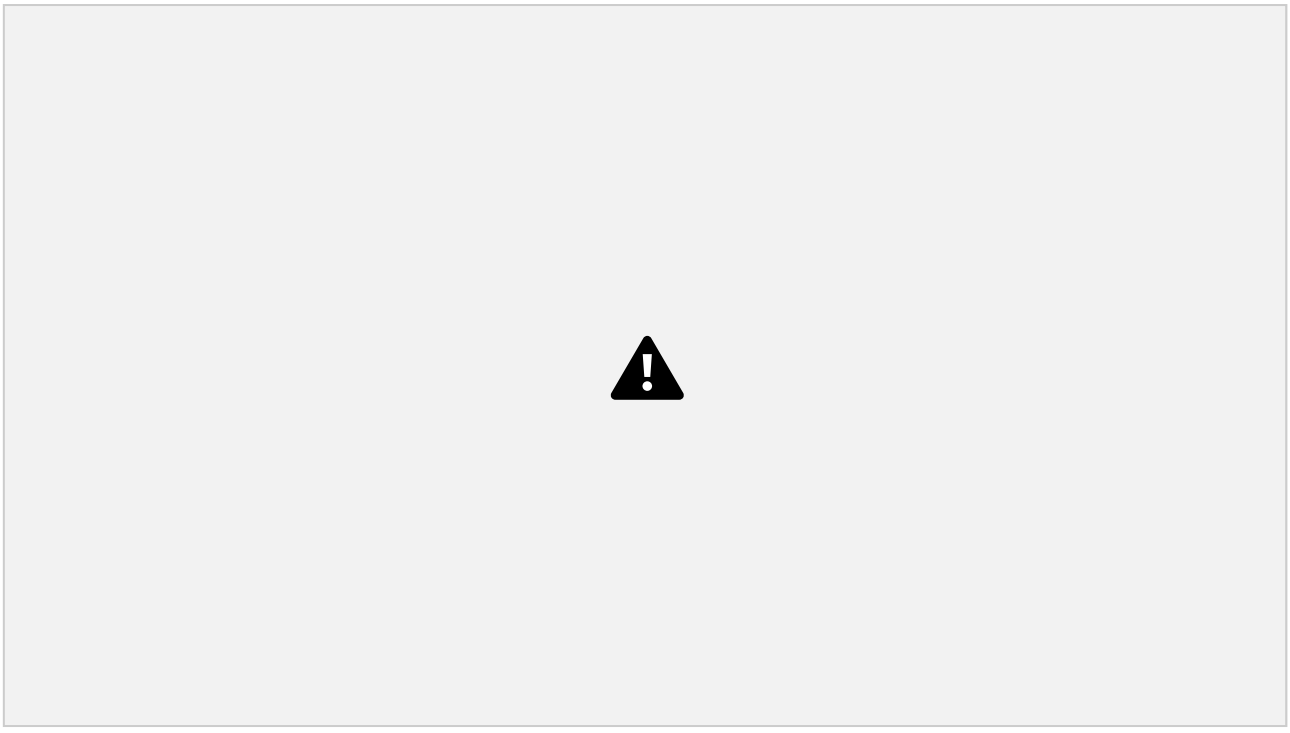

Рис. 2.7. Вайрфрейм «Перегляд записів пацієнта на прийом»

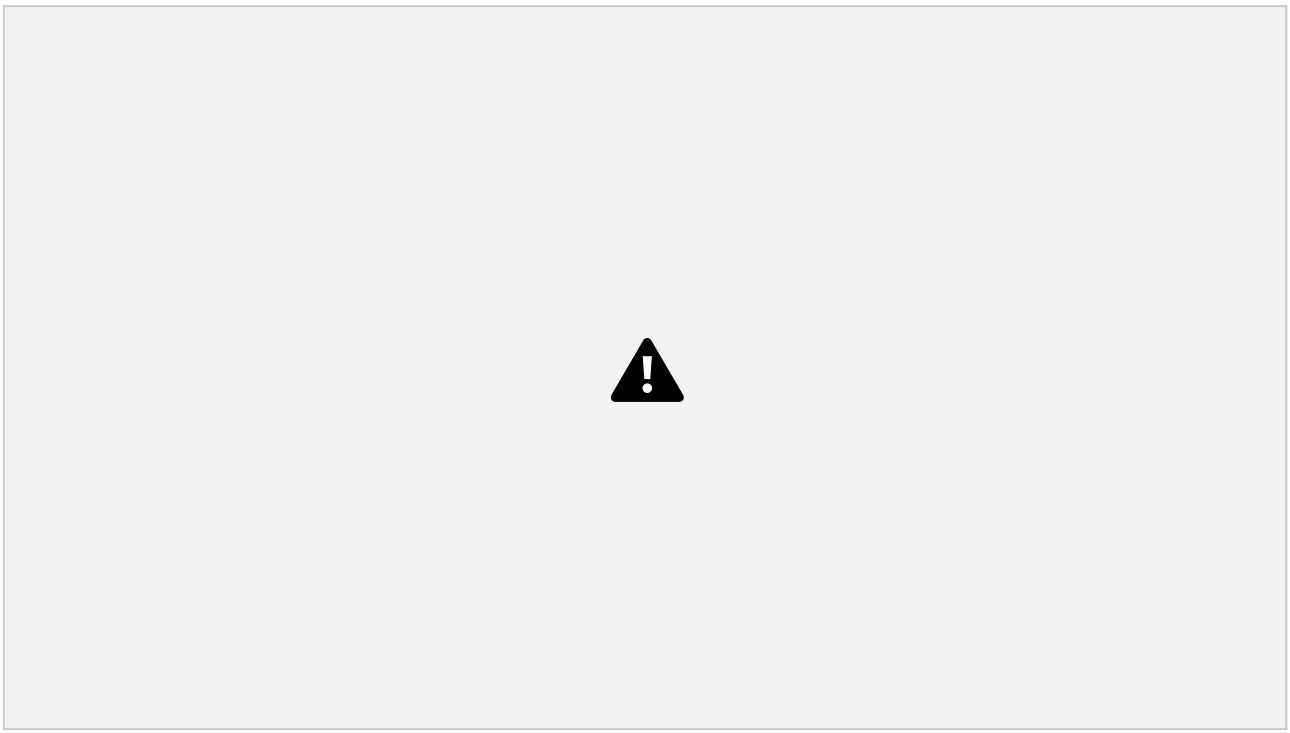

Рис. 2.8. Вайрфрейм «Перегляд списку лікарів»

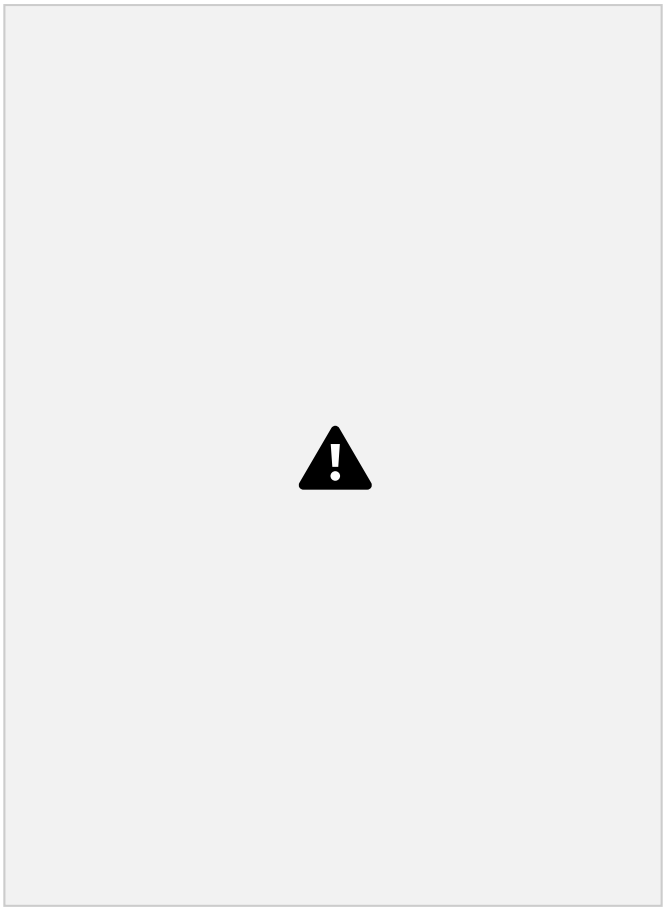

Рис. 2.9. Вайрфрейм «Додавання лікарів»

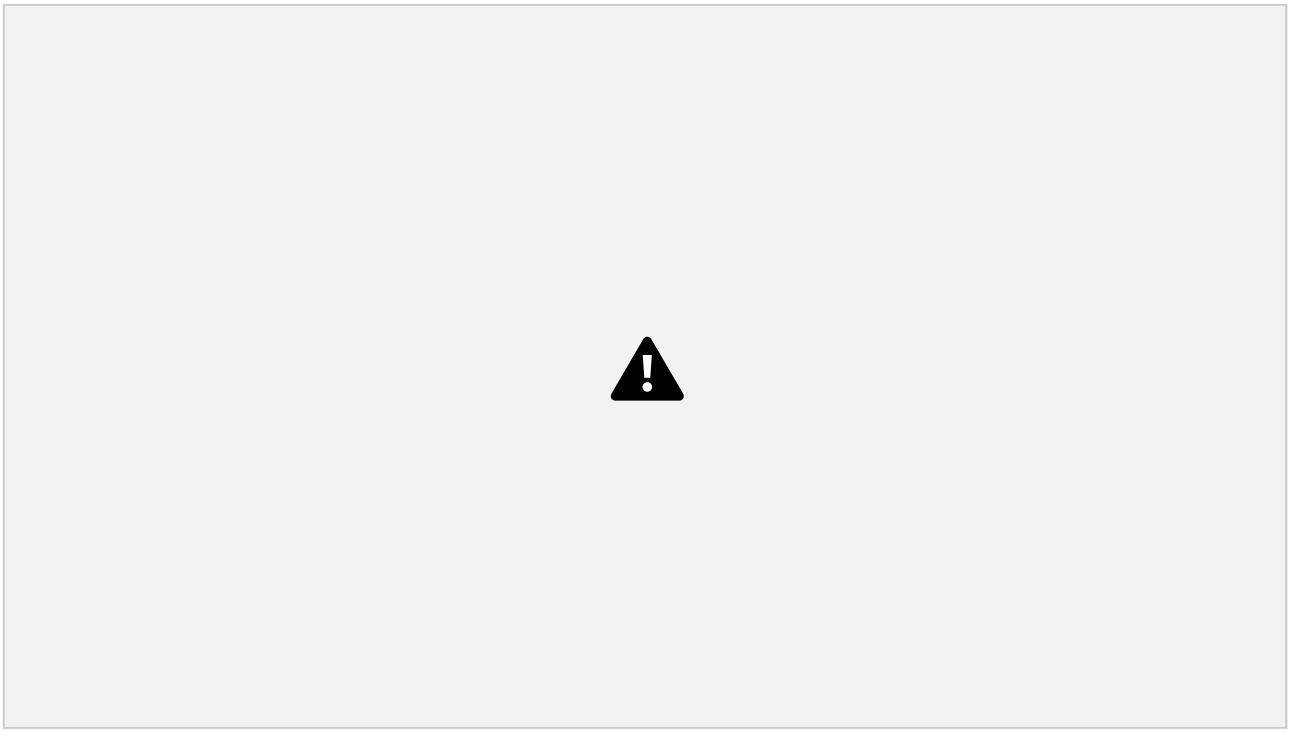

Рис. 2.10. Вайрфрейм «Перегляд списку записаних пацієнтів до лікаря»

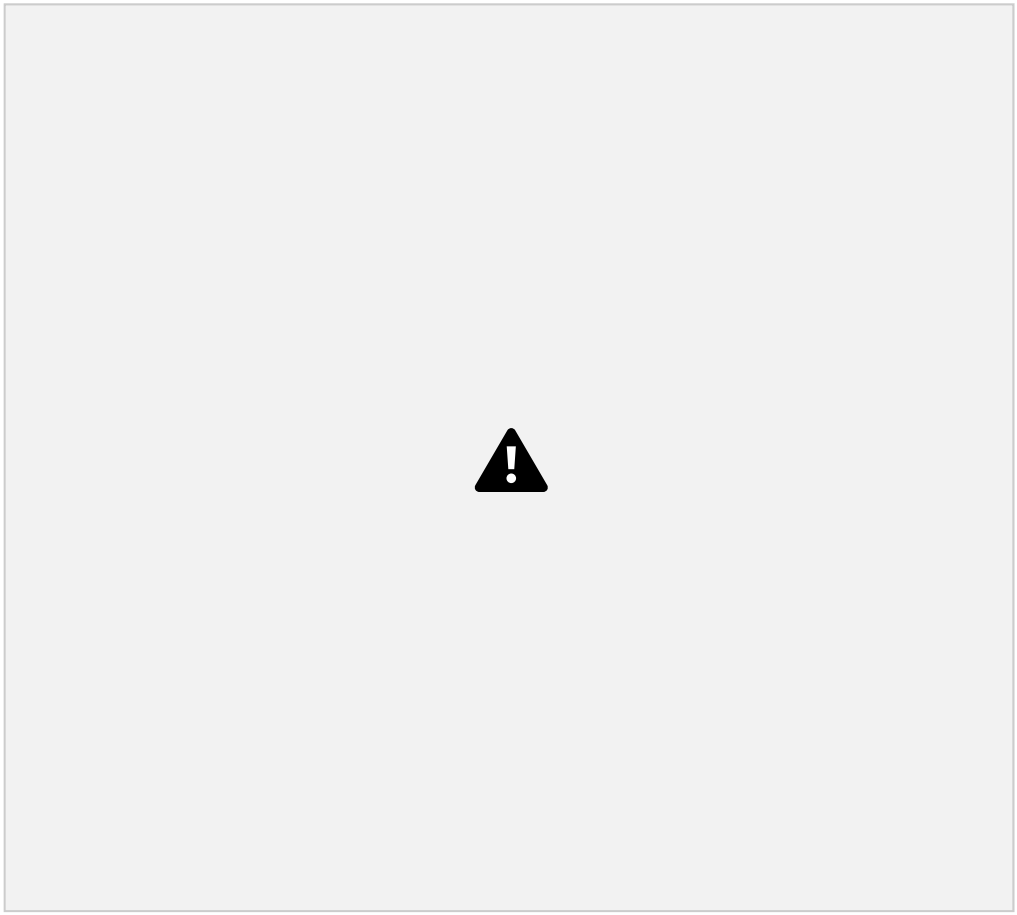

Рис. 2.11. Вайрфрейм «Редагування графіку роботи лікаря»

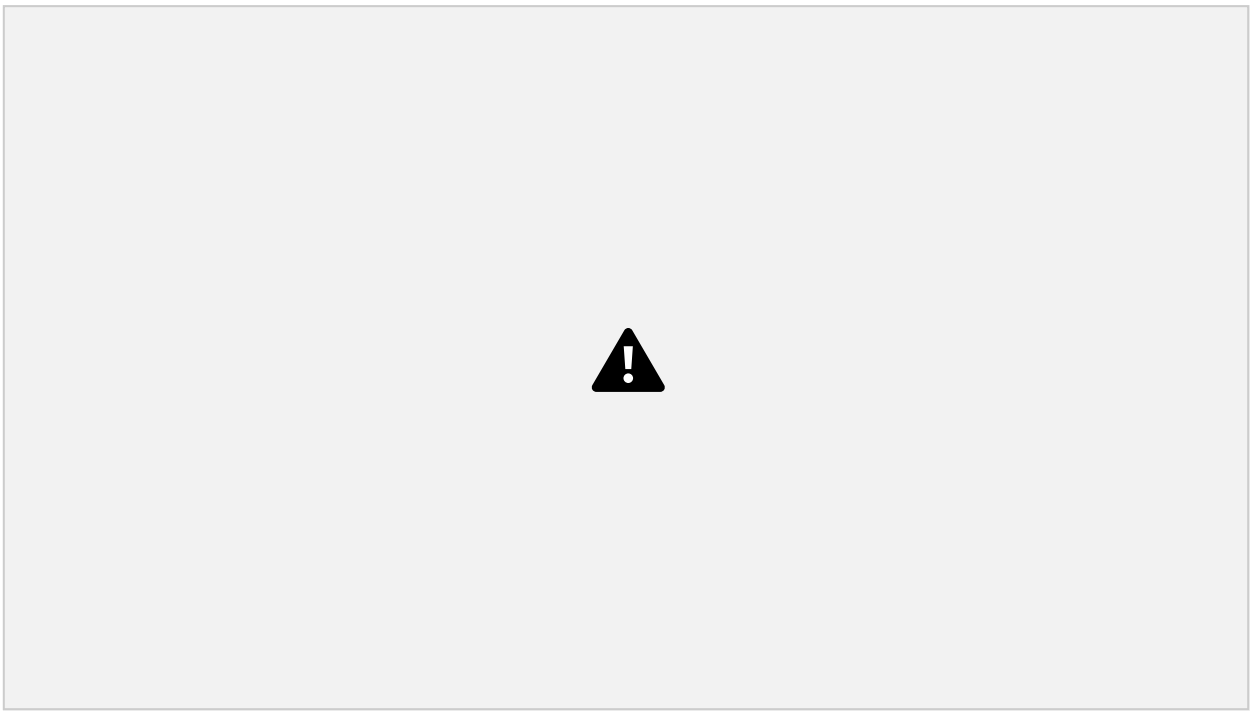

Рис. 2.12. Вайрфрейм «Реєстрація»

Висновки до розділу 2.

У процесі визначення та аналізу вимог до програмного модуля було створено глосарій, побудовано діаграму варіантів використання й описано основні варіанти використання у їх специфікаціях. Було спроєктовано інтерфейсні вікна програми для основної частини процесів. Також була проведена оцінка складності кожного з процесів, визначено функціональні та нефункціональні вимоги.

РОЗДІЛ 3
ПРОЄКТНІ ТА ТЕХНІЧНІ РІШЕННЯ

3.1. Логічна постановка задачі

Логічна постановка надається як опис логіки послідовних операцій у вигляді виконуваних функцій обробки інформації [14].

Застосунок являє собою веб-сайт, за допомогою якого можна відстежувати запис пацієнтів до лікаря.

Користуватись програмним продуктом можна під такими ролями: адміністратор, лікар та пацієнт.

При запуску додатку без авторизації користувач може знайти лікаря за

ім’ям, прізвищем або посадою, також можна переглянути таку інформацію, як графік роботи лікаря, номер кабінету, вільний та зайнятий час для запису та переглянути фото лікаря за наявності.

Для запису до лікаря користувач має авторизуватись як пацієнт. Пацієнт має змогу переглядати свої записи до лікаря та відхиляти заплановані записи. При перегляді можна сортувати записи за давністю та фільтрувати їх за статусом чи певним періодом часу.

Лікар може переглядати свої записи, змінювати статус записів та додавати до них коментарі. Для зручності також можна сортувати записи за давністю, та фільтрувати за статусом та періодом час. Лікар може брати собі вихідні, та переглядати календар вихідних в кабінеті. Також є змога редагування графіку роботи.

Адміністратор має змогу переглядати список всіх існуючих лікарів, редагувати дані про них, видаляти лікарів та додавати нових. Є можливість завантаження списку лікарів з csv-файлу. Адміністратор може додавати та видаляти вихідні для всіх лікарів одразу та переглядати календар робочих днів.

Кожного першого дня місяця дані про минулі вихідні лікарів будуть автоматично видалятись для того, щоб не засмічувати базу даних інформацією, яка ніяк не використовується.

Всі користувачі за всіма ролями мають змогу змінити особисті дані, такі як ім’я, прізвище, електронна пошта, фото тощо, в кабінеті користувача, є можливість змінити пароль. Також користувач може відновити пароль, якщо дані для входу було втрачено. Для цього потрібно ввести свою електронну пошту, на яку був зареєстрований аккаунт, у відповідне поле. Після чого після

переходу за посиланням, яке прийде у повідомленні на вказану пошту, можна змінити пароль.

3.2. Проєктування структури бази даних

Створення бази даних слід розпочинати з її проектування (розробки). У результаті проектування має бути визначено структуру бази, тобто склад таблиць, їхня структура та логічні зв’язки [5].

Структура реляційної таблиці визначається складом стовпців, їх послідовністю, типом даних кожного стовпця та його розміром, а також ключем таблиці. Процес проектування можна здійснювати двома підходами. При першому підході спочатку визначають основні завдання, на вирішення яких

створюється база і попити цих завдань у даних. При другому підході визначають предметну область, здійснюють огляд даних і встановлюють типові об'єкти предметної області. Особливо розумним підходом до проектування бази є поєднання обох підходів [7].

3.2.1. Концептуальне інфологічне проєктування

Інфологічний рівень - це інформаційно-логічна модель (ІЛМ) предметної області, у якій немає надмірних даних та у якій відображені інформаційні особливості об'єкта управління, не враховуючи особливостей і специфіки конкретної СКБД [13].

Мета інфологічного проектування – створити структуровану інформаційну модель ПЗ, для якої розроблятиметься БД [22]. При проектуванні на інфологічному рівні створюється ІЛМ, яка повинна відповідати таким вимогам: 1) коректність схеми БД;

2) простота і зручність використання на всіх етапах проектування, тобто

3) підтримка більшістю відомих СКБД (сіткові, реляційні, ієрархічні) 3)

модель має бути описана мовою, яка буде зрозумілою проектувальникам БД, програмістам, адміністратору та майбутнім користувачам. Основною складовою інфологічної моделі є атрибути, які слід проаналізувати та деяким чином згрупувати для подальшого зберігання у БД. Сутність інфологічного моделювання полягає у виділенні інформаційних об'єктів, що зберігаються в БД, а також визначення характеристик об'єктів та зв'язків між ними.

На основі аналізу вхідних та вихідних документів будується модель відображення безлічі реквізитів вихідних та вхідних документів на безліч елементів даних, що підлягають збереженню в базі даних, потім виконується приведення зібраної інформації до зручного для проектування виду. Для цього складають словник даних.

Словник даних, що містяться у таблицях бази даних, наведений у Таблиця 3.1.

Таблиця 3.1

Словник даних

№ п/п	Найменування елемента	Тип і довжина	Призначення елемента

1	birthday	String(22)	Зберігає дату народження пацієнта
2	cabinet	Integer	Зберігає номер кабінету, у якому приймає лікар
3	dayOfWeek	Integer	Зберігає номер дня тижня для графіку роботи лікаря
4	doctorsComment	String(500)	Зберігає коментар лікаря щодо прийому
5	email	String(50)	Зберігає електронну пошту користувача
6	endTime	Time	Зберігає час кінця робочого дня лікаря
7	firstName	String(30)	Зберігає ім’я користувача
8	gender	String(6)	Зберігає стать пацієнта
9	id	String(36)	Зберігає ІД кожного рядку таблиць
10	isGlobal	Boolean	Зберігає значення чи являється вихідний глобальним (для всіх лікарів лікарні)
11	lastName	String(50)	Зберігає прізвище користувача
12	lunchEnd	Time	Зберігає час кінця обідньої перерви лікаря
13	lunchStart	Time	Зберігає час початку обідньої перерви лікаря
14	password	String(50)	Зберігає пароль для авторизації
15	phoneNumber	String(10)	Зберігає номер телефону користувача
16	photoName	String(30)	Зберігає назву фото
17	position	String(30)	Зберігає посаду лікаря
18	startTime	Time	Зберігає час початку робочого дня лікаря
19	status	String(30)	Зберігає статус прийому
20	time	Date	Зберігає час запису до лікаря у форматі hh:mm

21	weekendDate	Date	Зберігає дату вихідного дня лікаря

Під час проєктування глобальної інфологічної моделі даних потрібно виявити еквівалентні сутності та виконати їхнє злиття, виявити й усунути дублювання атрибутів і зв'язків. Графічне подання глобальної моделі подається у вигляді ERD (нотація IDEF1X), діаграми класів або інших нотацій [17].

Графічне подання глобальної моделі у вигляді ERD на Рис. 3.1:

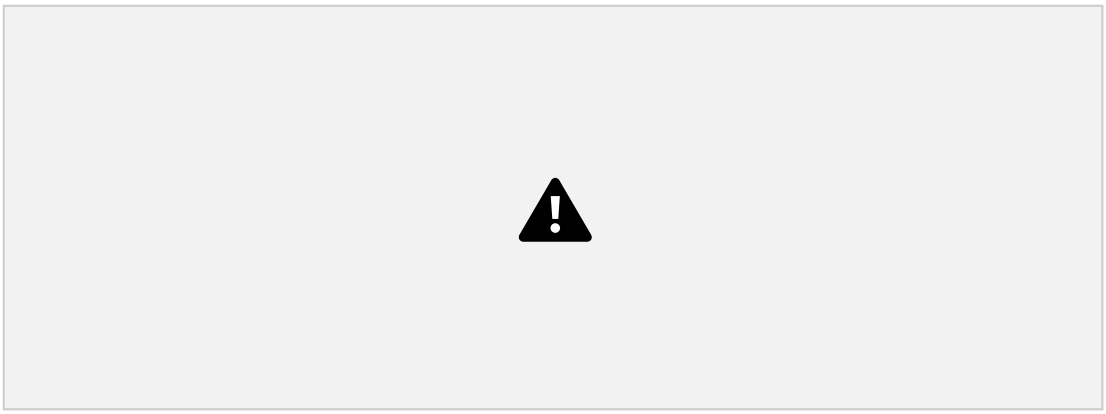

Рис. 3.1. Графічне подання глобальної моделі

Сутність «schedule» зберігає розклад роботи лікарів. Сутність «doctor» зберігає інформацію про лікарів. Сутність «position» зберігає посади лікарів. Сутність «weekend» зберігає дати, у які у лікарів буде вихідний. Сутність «appointment» зберігає дані про прийом у лікаря. Сутність «status» зберігає статус прийому у лікаря. Сутність «patient» зберігає інформацію про пацієнтів. Сутність «admin» зберігає інформацію про адміністраторів.

Згідно специфіки предметної області для сутностей розроблюваної системи визначаються обмеження (див. табл. 3.2).

Таблиця 3.2

Обмеження атрибутів

№ з/п	Ім’я атрибута	Межі або допустимі значення	Структура (формат)	Умова	Значення за замовченням
1	2	3	4	5	6
1	birthday	–	рррр-мм-дд	NULLABLE	null
2	cabinet	cabinet > 0	–	NOT NULL	–
3	dayOfWeek	7≥dayOfWeek ≥ 1	–	NOT NULL	–
4	doctorsComment	Максимальна кількість символів 500	–	NULLABLE	null

Закінчення табл. 3.2

1	2	3	4	5	6
5	email	Максимальна кількість символів 50	*@назва_домену	NOT NULL, UNIQUE	–
6	endTime	–	гг:хх	NOT NULL	–
7	firstName	Кількість символів від 2 до 30	–	NOT NULL	–
8	gender	male або female	–	NULLABLE	null
9	id	–	uuid (v4)	NOT NULL, UNIQUE	–
10	isGlobal	–	–	NOT NULL	false
11	lastName	Кількість символів від 5 до 50	–	NOT NULL	–
12	lunchEnd	–	гг:хх	NOT NULL	–
13	lunchStart	–	гг:хх	NOT NULL	–
14	password	Кількість символів від 6 до 50	–	NOT NULL	–
15	phoneNumber	–	–	NOT NULL	–
16	photoName	–	Назва_файлу.формат (формат має бути jpg, jpeg або png)	NULLABLE	null
17	position	Кількість символів від 5 до 30	–	NOT NULL	–
18	startTime	–	гг:хх	NOT NULL	–
19	status	Planned, canceled або completed	–	NOT NULL	planned
20	time	–	рррр-мм ддТгг:хх:сс+зз:зз (ISO 8601 date-time)	NOT NULL	–
21	weekendDate	–	рррр-мм-дд	NOT NULL	–

3.2.2. Проєктування логічної моделі даних

Проєктування логічної моделі даних містить: обґрунтування вибору СКБД, графічне подання логічної моделі у вигляді ERD (в нотації IDEF1X), діаграми класів тощо [14].

Була створена база даних за допомогою СКБД PostgreSQL. Була обрана саме це СКБД так як вона має такі переваги, які повністю відповідають бізнес завданню, а саме:

1) необмежений максимальний розмір бази даних;
2) висока цілісність даних;
3) безкоштовне ПО з відкритим вихідним кодом;
4) кросплатформеність;

5) висока швидкість простих операцій вибірки даних;
6) низька потреба експлуатаційних витрат;
7) має свою мову структурованих запитів, яка надає широкий функціонал;
8) має безліч типів полів та можливість створювати користувацькі типи.

Структура логічної моделі даних відображена на Рис. 3.2.

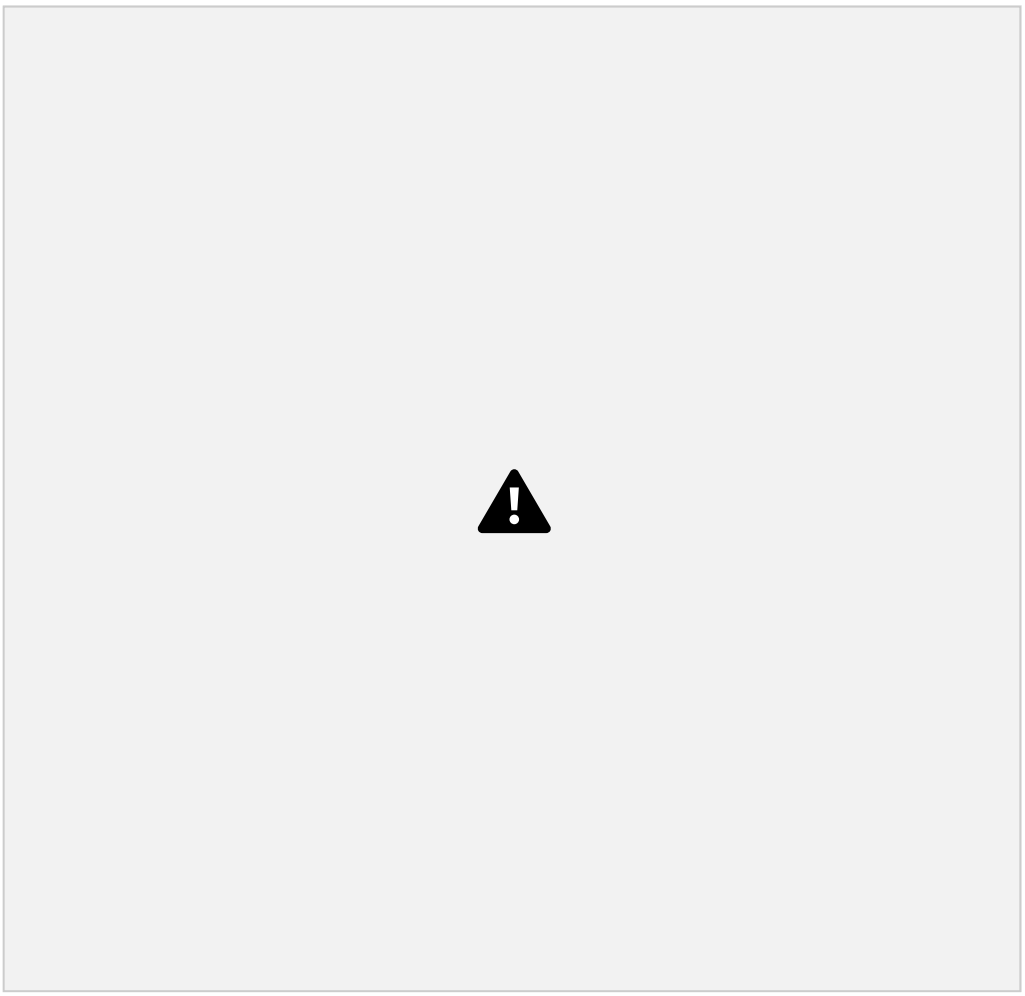

Рис. 3.2. Структура логічної моделі даних

3.2.3. Проєктування фізичної моделі даних

Фізичне проектування бази даних – процес підготовки опису реалізації бази даних на вторинних пристроях. На цьому етапі розглядають основні відносини, а також усі пов'язані з представленими даними обмеження цілісності та засоби захисту [7].

Структура фізичної моделі даних відображена на рис. 3.3.

Побудована фізична модель повністю задовольняє потреби бізнес процесів – вона є безпечною, ефективною та зручною у використанні.

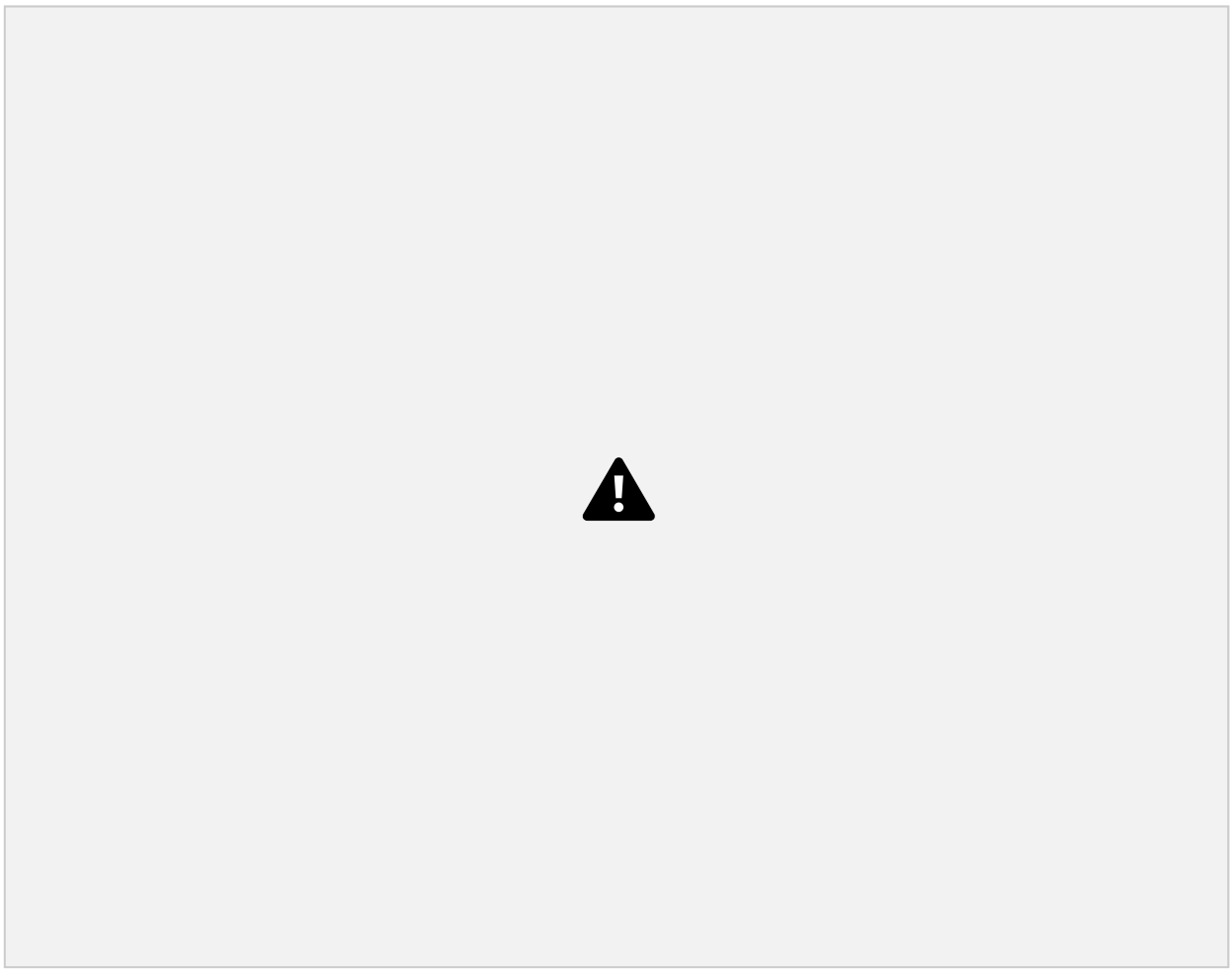

Рис. 3.3. Структура фізичної моделі даних

3.3. Проєктування програмного забезпечення

Діаграма класів визначає типи класів системи та різного роду статичні зв'язки, які існують між ними. На діаграмах класів також зображують атрибути класів, їх операції та обмеження, що накладаються на зв'язки між класами [10].

Вигляд та інтерпретація діаграми класів істотно залежить від точки зору (рівня абстракції): класи можуть представляти сутності предметної галузі (у процесі аналізу) або елементи програмної системи (у процесах проектування та реалізації).

Згідно з предметною областю та обраними технологіями розробки на рис.3.4 представлена діаграма класів з рівнем абстракції, який не передбачає вказування зв’язків між класами.

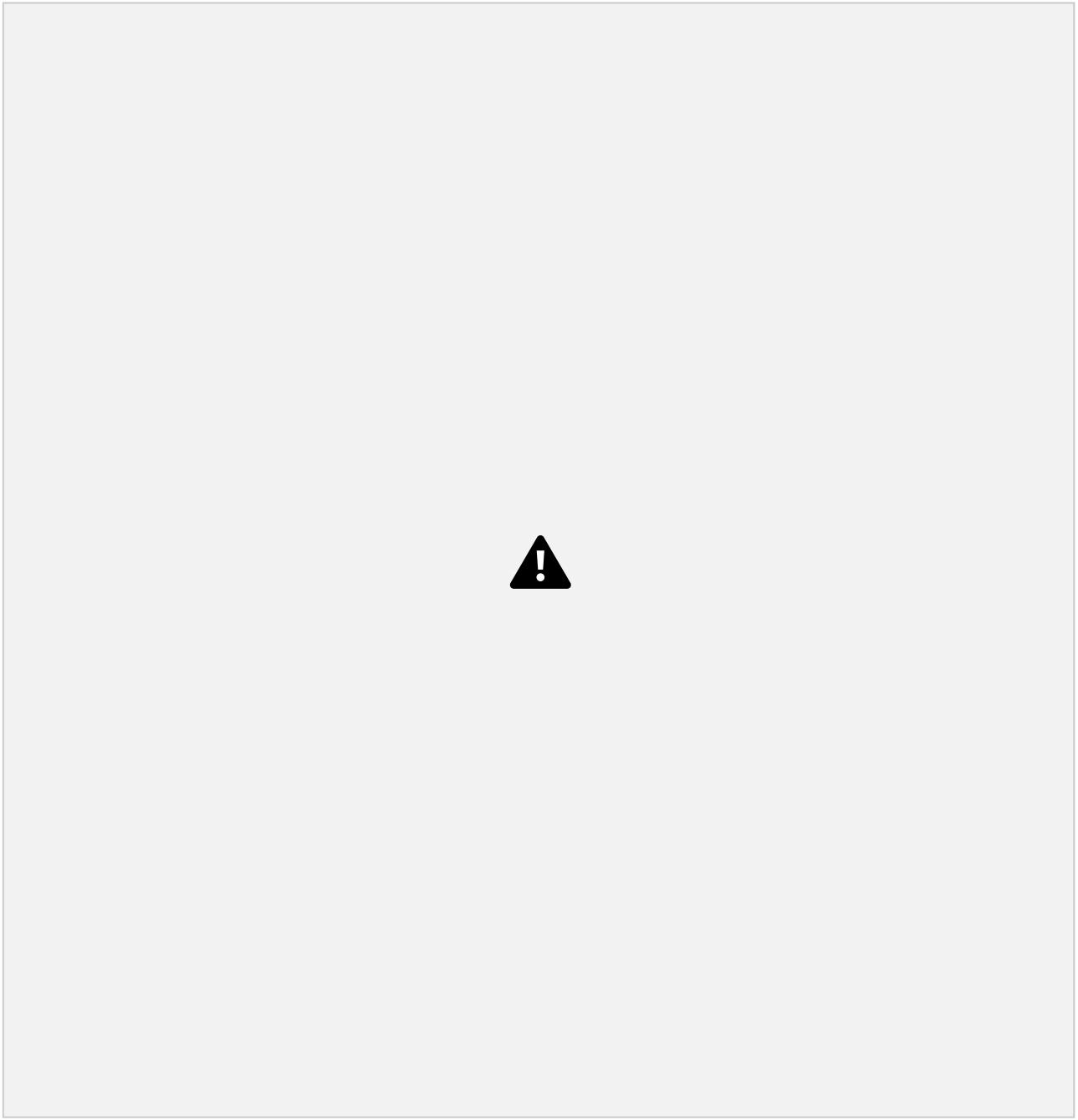

Рис. 3.4. Діаграма класів

В класах «Admin», «Patient», «Doctor», «Schedule», «Appointment», «Weekend», «Status», «Position» знаходяться сутності таблиць БД. Класи з постфіксом «Service» включають в себе логіку роботи з даними. Класи з постфіксом «Controller» включають в себе методи, які відповідають за обробку

запитів. Класи з постфіксом «Module» надають дані, які використовуються для організації структури програми. Класи с постфіксом «Dto» потрібні для передачі даних між підсистемами програми.

Інтерфейс користувача складається з елементів, які можуть перебувати в

різних станах, а також у процесі взаємодії з користувачем програми змінювати свій стан. Він є системою, що управляється подіями (СУП) [12].

Поводження таких систем найкраще характеризується їхньою реакцією на зовнішні події. Здебільшого, СУП перебуває в стані очікування, поки не одержить повідомлення про подію. Після того як СУП відреагує на подію, вона знову переходить у стан очікування наступної події. Для таких систем важливо визначити, насамперед, стійкі стани, події, що ініціюють переходи з одного стану в інший, і дії, що виконуються під час зміни стану.

Для формального опису СУП доцільно використовувати UML-діаграми станів.

Діаграма станів зв'язує події й стани. У ході виникнення події наступний стан системи залежить як від її поточного стану, так і від події. Зміна стану називається переходом. Діаграма станів – це граф, вузли якого моделюють стани, а спрямовані дуги, позначені іменами відповідних подій, – переходи.

На рис. 3.5, рис. 3.6, рис. 3.7 зображені діаграми станів для пацієнта, лікаря та адміністратора відповідно.

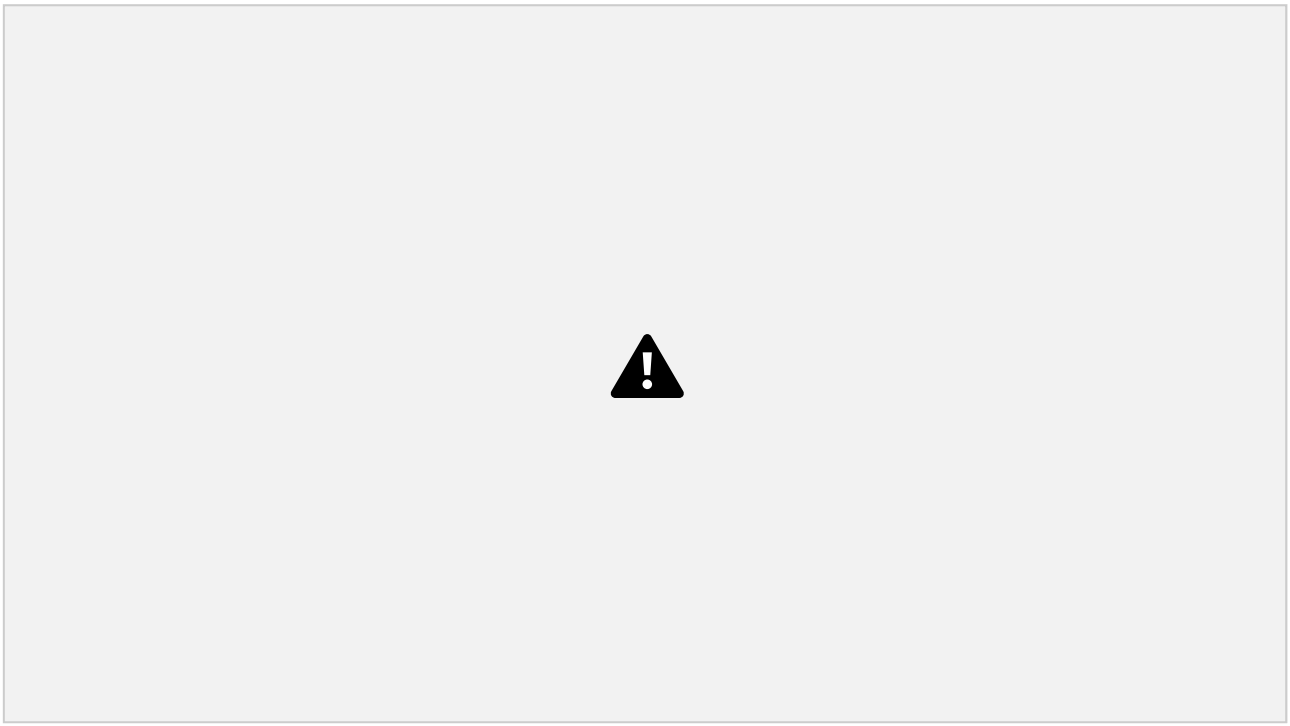

Рис. 3.1. Діаграма станів для пацієнта

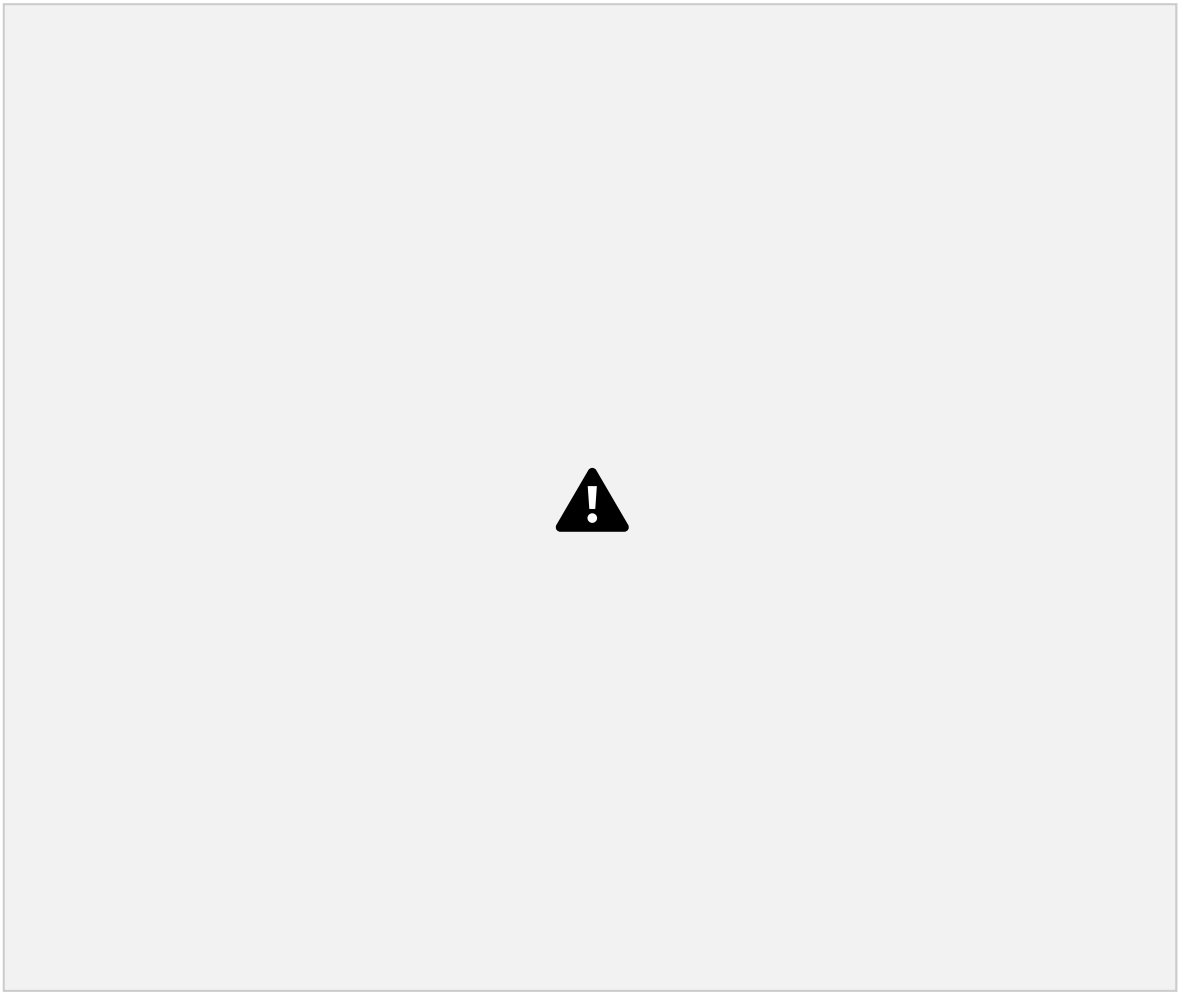

Рис. 3.6. Діаграма станів для лікаря

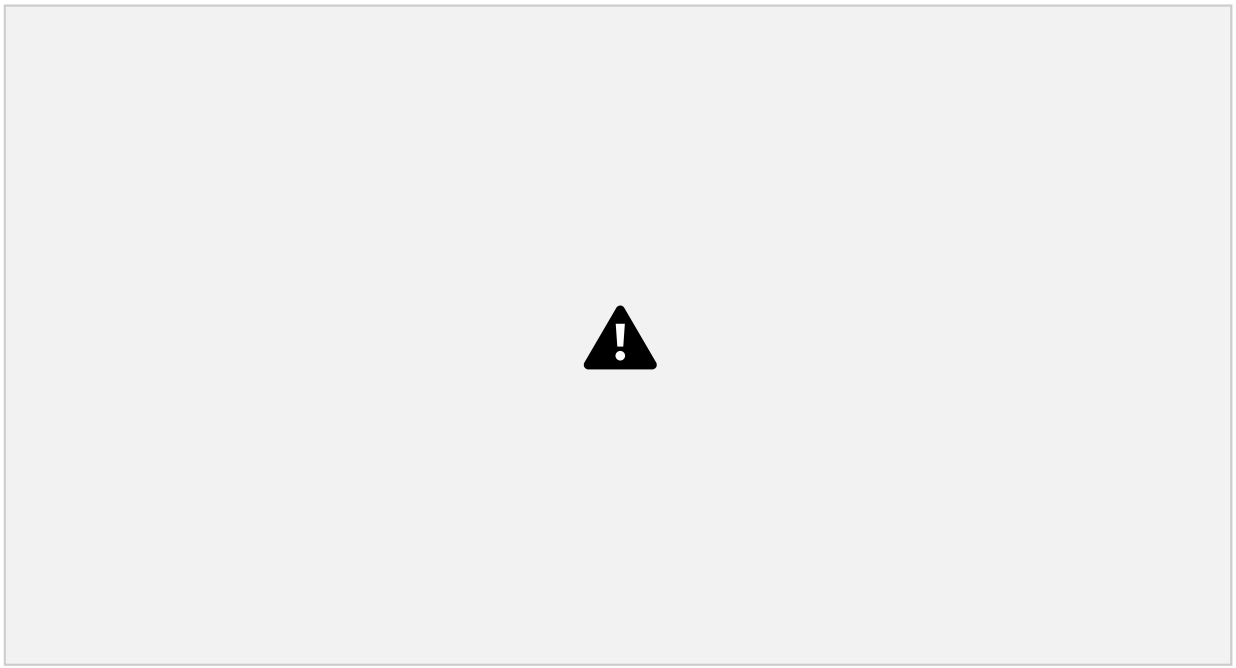

Рис. 3.7. Діаграма станів для адміністратора

Діаграма станів, яка зображена на рис. 3.5, відображає стани інтерфейсу

для пацієнта. Після авторизації пацієнт буде бачити сторінку для пошуку лікаря. Ввівши пошуковий запит, за яким не вдалось знайти жодного лікаря, буде відображено відповідний текст, про відсутність результатів пошуку. Якщо ж результати пошуку є – пацієнт зможе обрати лікаря й дізнатись вільний для запису час. Після вибору часу потрібно переглянути деталі запису та підтвердити його. Також є можливість переглянути свої записи до лікаря та відхилити запланований прийом. Можна переглянути профіль користувача та відредагувати інформацію. За бажання можна додати або видалити фото з профілю.

На рис. 3.6. зображена діаграма станів інтерфейсу для лікаря. Лікар може переглянути список пацієнтів, які записані до нього на прийоми. Можна змінити статус прийому та додати коментар до запису. За потреби коментар можна редагувати. Є можливість перегляду інформації про користувача та її редагування. Також можна додати або видалити фото користувача. Також лікар може переглядати календар робочих днів та, за потреби, додавати собі вихідні (якщо, наприклад, лікар йде у відпустку або бере лікарняний).

Діаграма станів, що зображена на рис. 3.7, відображає стани інтерфейсів для адміністратора. Після авторизації адміністратор переходить до сторінки зі списком лікарів, де він може переглянути детальну інформацію про кожного з них, відредагувати її або ж видалити лікаря зі списку. Також адміністратор має змогу додавати нових лікарів. Це можна зробити після натискання кнопки “Add doctor” та заповнити дані у формі, або завантажити csv-файл зі списком лікарів. Адміністратор може переглядати календар робочих днів лікарів, додавати та видаляти вихідні для всіх лікарів одразу.

3.4. Тестування програмної системи

Тестування програмної системи – це метод перевірки відповідності фактичного програмного продукту очікуваним вимогам, який також необхідний, щоб переконатися, що продукт не містить дефектів [25]. Під дефектом розуміється вада в компоненті або системі, яка може привести компонент або систему до неможливості виконати потрібну опцію, наприклад, неправильний оператор або визначення даних.

Завдання тестування – визначення умов, за яких проявляються дефекти системи, і протоколювання цих умов.

Мета застосування процедури тестування програмного коду – зведення кількості дефектів у застосунку до мінімуму.

Підсумком процесу тестування має стати висновок про якість розробленого текстового редактору, сформований на основі списку протестованих функцій, списку знайдених дефектів і його аналізі .

Для тестування застосунку було застосовано ручне тестування через відсутність суворої специфікації, а також зважаючи на обмеженість ресурсів на формалізацію тестів [26].

Тест-вимоги функціонального тестування наведені у наступному списку: 1) перевірка функціоналу, який має пацієнт:

1.1) при авторизації як пацієнт відбувається перенаправлення на сторінку з пошуком лікаря;

1.2) при пошуку лікаря відображається по 10 лікарів на одній сторінці;

1.3) при пошуку лікаря за запитом, який не дав результатів, відображається текст «The search has not given any results. Try to find a doctor on a different request.»;

1.4) після запису до лікаря, час, який був обраний, відображається як зайнятий;

1.5) після відхилення запланованого запису до лікаря, час, який був обраний, відображається як вільний;

2) перевірка функціоналу, який має лікар:

2.1) при авторизації як лікар відбувається перенаправлення на сторінку з записами пацієнтів до лікаря;

2.2) після додавання коментаря чи зміни статусу запису лікарем у пацієнта в аккаунті відображаються актуальні дані;

3) перевірка функціоналу, який має адміністратор:

3.1) при авторизації як адміністратор відбувається перенаправлення на сторінку зі списком лікарів;

3.2) після додавання адміністратором вихідного для лікарів, у лікарів в календарі робочих днів цей день відображається як вихідний;

3.3) після редагування адміністратором лікаря в профілі лікаря відображається оновлена інформація;

3.4) після додавання нового лікаря, він одразу відображається в списку;

3.5) після додавання лікаря, йому на електронну пошту приходить повідомлення з логіном на паролем для входу.

Далі наведено тест-плани, відповідно до існуючих тест-вимог.

Тестовий приклад: № 1.

Призначення: Перевірка того, що при авторизації як пацієнт відбувається перенаправлення на сторінку з пошуком лікаря. (табл. 3.3)

Тест-вимога, що перевіряється: 1.1.

Передумови для тесту: пацієнт має бути зареєстрований у системі.

Критерій проходження тесту: відображається сторінка з пошуком лікаря.

Таблиця 3.3

Тестовий приклад №1

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	1) Введення логіну від аккаунту пацієнта; 2) Введення паролю від аккаунту пацієнта; 3) Після успішної авторизації отримати сторінку з пошуком лікаря.	Відображається сторінка з пошуком лікаря	Відображається сторінка з пошуком лікаря	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 2.

Призначення: перевірка того, що при пошуку лікаря відображається по 10 лікарів на одній сторінці. (табл. 3.4)

Тест-вимога, що перевіряється: 1.2.

Передумови для тесту: застосунок має бути розгорнутий.

Критерій проходження тесту: кожне зі 100 слів має переклад.

Таблиця 3.4

Тестовий приклад №2

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	1) Введення пошукового запиту у відповідне поле; 2) Отримання списку знайдених лікарів;	На одній сторінці відображається максимум 10 лікарів	На одній сторінці відображається максимум 10 лікарів	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 3.

Призначення: перевірка того, що при пошуку лікаря за запитом, який не дав результатів, відображається текст «The search has not given any results. Try to find a doctor on a different request.». (табл. 3.5)

Тест-вимога, що перевіряється: 1.3.

Передумови для тесту: застосунок має бути розгорнутий.

Критерій проходження тесту: відображення тексту «The search has not given any results. Try to find a doctor on a different request.».

Таблиця 3.5

Тестовий приклад №3

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	1) Введення пошукового запиту у відповідне поле; 2) Отримання відповідного тексту про відсутність результатів;	Відображення тексту «The search has not given any results. Try to find a doctor on a different request.»	Відображення тексту «The search has not given any results. Try to find a doctor on a different request.»	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 4.

Призначення: Перевірка того, що після запису до лікаря, час, який був обраний, відображається як зайнятий. (табл. 3.6)

Тест-вимога, що перевіряється: 1.4.

Передумови для тесту: користувач має бути авторизований як пацієнт.
Критерій проходження тесту: відображення часу, на який був зроблений запис, як зайнятого.

Таблиця 3.6

Тестовий приклад №4

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію

1	2	3	4	5
1	1) Введення пошукового запиту у відповідне поле; 2) Вибір лікаря; 3) Вибір вільного часу у лікаря;	Відображення часу, на який був зроблений запис, як зайнятого.	Відображення часу, на який був зроблений запис, як зайнятого.	Так

Закінчення табл. 3.6

1	2	3	4	5
	4) Підтвердження запису до лікаря; 5) Повторити пункти 1- 2; 6) Час, який був обраний у пункті 3 відображається як зайнятий.			

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 5.

Призначення: перевірка того, що після відхилення запланованого запису до лікаря, час, який був обраний, відображається як вільний. (табл. 3.7)

Тест-вимога, що перевіряється: 1.5.

Передумови для тесту: пацієнт повинен мати хоча б один запланований запис до лікаря.

Критерій проходження тесту: час, на який був запланований запис до лікаря, відображається як вільний.

Таблиця 3.7

Тестовий приклад №5

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію

1	1) Натиснути на кнопку “My appointments” в меню; 2) Натиснути на кнопку “Cancel the appointment”; 3) Перейти до головної сторінки; 4) Ввести пошуковий запит щоб знайти лікаря, до якого був запланований запис; 5) Обрати потрібного лікаря зі списку; 6) Переглянути графік роботи.	Час, на який був запланований запис до лікаря, відображається як вільний.	Час, на який був запланований запис до лікаря, відображається як вільний.	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 6.

Призначення: перевірка того, що при авторизації як лікар відбувається перенаправлення на сторінку з записами пацієнтів до лікаря. (табл. 3.8)
Тест-вимога, що перевіряється: 2.1.

Передумови для тесту: лікар має бути зареєстрований у системі. Критерій проходження тесту: відображається сторінка з записами до лікаря.

Таблиця 3.8

Тестовий приклад №6

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	1) Введення логіну від аккаунту лікаря; 2) Введення паролю від аккаунту лікаря; 3) Після успішної авторизації отримати сторінку з записами до лікаря.	Відображається сторінка з записами до лікаря.	Відображається сторінка з записами до лікаря.	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 7.

Призначення: перевірка того, що після додавання коментаря чи зміни

статусу запису лікарем у пацієнта в аккаунті відображаються актуальні дані. (табл. 3.9)

Тест-вимога, що перевіряється: 2.2.

Передумови для тесту: хоча б один пацієнт повинен мати хоча б один запис до лікаря, лікар має бути авторизований.

Критерій проходження тесту: в кабінеті пацієнта відображаються актуальні дані.

Таблиця 3.9

Тестовий приклад №7

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	1) Змінити статусу прийому; 2) Додати коментаря до статусу; 3) Увійти до системи як пацієнт; 4) Перейти до сторінки з записами до лікарів; 5) Перевірка чи дані являються актуальними.	В кабінеті пацієнта відображаються актуальні дані.	В кабінеті пацієнта відображаються актуальні дані.	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 8.

Призначення: Перевірка того, що при авторизації як адміністратор відбувається перенаправлення на сторінку зі списком лікарів (табл. 3.10).

Тест-вимога, що перевіряється: 3.1

Передумови для тесту: адміністратор має бути зареєстрований в системі.

Критерій проходження тесту: відображення сторінки зі списком лікарів.

Таблиця 3.10

Тестовий приклад №8

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію

1	1) Введення логіну від аккаунту лікаря; 2) Введення паролю від аккаунту лікаря; 3) Після успішної авторизації отримати сторінку з записами до лікаря.	Відображається сторінка зі списком лікарів.	Відображається сторінка зі списком лікарів.	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 9.

Призначення: перевірка того, що після додавання адміністратором вихідного для лікарів, у лікарів в календарі робочих днів цей день відображається як вихідний. (табл. 3.11)

Тест-вимога, що перевіряється: 3.2.

Передумови для тесту: у списку лікарів має бути хоча б один лікар, адміністратор має бути авторизований у системі.

Критерій проходження тесту: в кабінеті у лікаря відображається актуальний календар вихідних.

Таблиця 3.11

Тестовий приклад №9

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5
1	1) Перейти до сторінки з календарем вихідних;	В кабінеті у лікаря відображається актуальний календар вихідних.	В кабінеті у лікаря відображається актуальний календар вихідних.	Так

Закінчення табл. 3.11

1	2	3	4	5

	2) Натиснути кнопку “Add weekend”; 3) Вказати дату вихідного; 4) Підтвердити додавання вихідного; 5) Авторизуватись у системі як лікар; 6) Перейти до сторінки з календарем вихідних.			

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 10.

Призначення: перевірка того, що після редагування адміністратором лікаря в профілі лікаря відображається оновлена інформація. (табл. 3.12)
Тест-вимога, що перевіряється: 3.3.

Передумови для тесту: у списку лікарів має бути хоча б один лікар, адміністратор має бути авторизований у системі.

Критерій проходження тесту: в кабінеті у лікаря відображається актуальна інформація у профілі користувача.

Таблиця 3.12

Тестовий приклад №10

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	1) Натиснути на копку редагування біля відповідного лікаря; 2) Відредагувати будь які поля; 3) Натиснути кнопку “Edit”; 4) Авторизуватись у системі як лікар; 5) Перейти до профілю користувача	В кабінеті у лікаря відображається актуальна інформація у профілі користувача.	В кабінеті у лікаря відображається актуальна інформація у профілі користувача.	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 11.

Призначення: перевірка того, що після додавання нового лікаря, він одразу

відображається в списку. (табл. 3.13)

Тест-вимога, що перевіряється: 3.4.

Передумови для тесту: адміністратор має бути авторизований у системі. Критерій проходження тесту: у списку лікарів відображається інформація про щойно доданого лікаря.

Таблиця 3.13

Тестовий приклад №11

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	1) Натиснути на копку “Add doctor”; 2) У модальному вікні, що з’явиться, заповнити всі поля; 3) Натиснути кнопку “Add doctor”; 4) Закрити модальне вікно.	У списку лікарів відображається інформація про щойно доданого лікаря.	У списку лікарів відображається інформація про щойно доданого лікаря.	Так

Відмітка про проходження тесту: пройдений.

Тестовий приклад: № 12.

Призначення: перевірка того, що після додавання лікаря, йому на електронну пошту приходить повідомлення з логіном на паролем для входу. (табл. 3.14)

Тест-вимога, що перевіряється: 3.5.

Передумови для тесту: адміністратор має бути авторизований у системі. Критерій проходження тесту: на електронній пошті лікаря присутнє повідомлення з логіном та паролем для входу.

Таблиця 3.14

Тестовий приклад №12

№ п/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію

1	1) Натиснути на копку “Add doctor”; 2) У модальному вікні, що з’явиться, заповнити всі поля; 3) Натиснути кнопку “Add doctor”; 4) Закрити модальне вікно. 5) Перевірити наявність повідомлення на електронній пошті, яку було вказано під час додавання лікаря.	На електронній пошті лікаря присутнє повідомлення з логіном та паролем для входу.	На електронній пошті лікаря присутнє повідомлення з логіном та паролем для входу.	Так

На основі наведених вище тестових прикладів, ми можемо зробити висновок про те, що програмний продукт відповідає функціональним вимогам, виходячи з того, що всі тести були пройдені успішно.

3.5. Розгортання програмного продукту

Розгортання – це дії, які потрібно виконати для того, щоб зробити застосунок загальнодоступним [24]. Для даного web-застосунку розгортання відбувається шляхом його розміщення на web-хостингу.

Для розгортання web-застосунку «Реєстратура медичного закладу» потрібно:

- встановлений Node.js версії 14.19.2 або вище [31];
- встановлений PostreSQL версії 14 або вище [32];
- 1,5 гб дискового простору для зберігання вихідних файлів програми;
- 50 мб дискового простору для бази даних.

Командою npm install у папці з front-end частиною та у папці з back-end частиною потрібно встановити всі необхідні бібліотеки та залежності. За допомогою PostgerSQL потрібно створити базу даних з назвою «hospital» або змінити необхідну назву у back-end частині та так само назвати БД у PostgerSQL.

Після розміщення застосунку на хостингу користувачі з доступом в Інтернет матимуть змогу користуватись продуктом.

Висновки до розділу.

У цьому розділі була сформульована логічна постановка задачі, було проведено концептуальне інфологічне проєктування БД, також спроєктовано фізичну та логічну модель БД. У процесі проєктування ПЗ було представлено діаграму класів та діаграму станів графічного інтерфейсу. Під час тестування ПЗ

дефектів виявлено не було. Також були представлені вимоги до розгортання програмного продукту.

ВИСНОВКИ

У процесі виконання дипломного проєкту було розроблено web застосунок «Реєстратура для медичних закладів». Web-сайт буде допомагати пацієнтам банально не стояти у черзі та відстежувати призначення лікарів у своєму аккаунті, лікарям відстежувати активність та комфортно керувати своїм робочим часом та адміністраторам комфортно відстежувати навантаженість лікарів та лікарні в цілому, вести облік лікарів.

Було проведено опис та аналіз предметної області, а саме проведено опис структурних і функціональних особливостей Товариства з обмеженою відповідальністю «АЙТІ ПРОСТІР», також було виконано аналіз аналогів, в яких було оглянуто їх недоліки та переваги, тим самим було виявлено оптимальний варіант роботи. Також був проведений аналіз бізнес-процесів за допомогою IDEF0-діаграм.

Було розроблено специфікацію вимог до модуля, а саме створено глосарій проєкту, розроблені варіанти використання та їх специфікація, розроблена специфікація функціональних та нефункціональних вимог, також був спроєктований інтерфейс користувача.

Під час розробки проєктних та технічних рішень була сформульована логічна постановка задачі, спроєктована глобальна, логічна та фізична моделі даних, спроєктована структура програмного забезпечення. Також було проведене тестування ПЗ, у ході якого не було знайдено ніяких дефектів. Були сформульовані умови до розгортання програмного продукту.

Функціональність модуля полягає в зручності використання та обробки інформації про запис до лікаря.

Переваги продукту:

1) інтуїтивно зрозумілий інтерфейс;

2) швидкість розгортання;

3) швидкість роботи застосунку;

4) багато функцій, які підвищують зручність користування, такі як фільтрація та сортування даних;

5) універсальність, застосунком зможе почати користуватись будь-яка лікарня у найменші строки.

Back-end частина розроблена із застосуванням мови програмування

TypeScript та фреймворку Nest.js [27]. Для зберігання даних використовувалась СКБД PostgreSQL , для зручного користування функціоналом, який надає ця БД використовувалась ORM під назвою Sequelize [35].

61

Front-end частина також розроблялась із застосуванням TypeScript. Для розробки використовувалась бібліотека React.js [33, 34].

Розроблений програмний продукт відповідає поставленим вимогам.

62

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бізнес процеси [Електронний ресурс] – Режим доступу: https://business.diia.gov.ua/cases/sistematizacia-biznes-procesiv/biznes-procesi 2. Варіант використання [Електронний ресурс] – Режим доступу: https://studfile.net/preview/9828815/page:2/

3. Використання діаграми варіантів використання UML під час проектування програмного забезпечення [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/566218/

4. Вимоги до програмного забезпечення [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Вимоги_до_програмного_забезпечення 5. Гайна. Г.А. Основи проектування баз даних: Навчальний посібник. / Г.А. Гайна. − К.: КНУБА, 2018. – 48 с.

6. Глосарій [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Глосарій

7. Етапи проєктування БД [Електронний ресурс] – Режим доступу: https://studfile.net/preview/9295990/page:10/

8. Закон України «Основи законодавства України про охорону здоров'я» // Відомості Верховної Ради України – 1993, № 4, ст.19. 9. Закон України «Про захист інформації в інформаційно комунікаційних системах» із змінами, внесеними згідно із Законом № 1089-IX від 16.12.2020 // Відомості Верховної Ради України. – 1994, № 31, ст.286 10. Застосування UML в дипломних роботах [Електронний ресурс] – Режим доступу: https://dut.edu.ua/ua/news-1-626-7758-zastosuvannya-uml-v diplomnih-robotah_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy 11. Захист персональних даних [Електронний ресурс] – Режим доступу: https://pdp.nacs.gov.ua/pages/zahyst-pers-dannih

12. Інструментальні засоби автоматизованого проектування [Електронний

ресурс] // Lviv Polytechnic National University – Academic Journals and Conferences. – Режим доступу: https://science.lpnu.ua/sites/default/files/journal paper/2018/jul/13597/3klasifikaciyainteraktivnihvzaiemodiykoristuvacha.pdf

13. Канонічна інформаційно-логічна модель предметної області [Електронний ресурс] – Режим доступу: http://um.co.ua/3/3-3/3-38916.html 14. Логічна модель системи [Електронний ресурс] – Режим доступу: https://posibniki.com.ua/post-logichna-model-sistemi

15. Методичні рекомендації до виконання кваліфікаційних робіт для студентів спеціальностей 121 "Інженерія програмного забезпечення", 122

"Комп'ютерні науки", 126 "Інформаційні системи та технології" першого (бакалаврського) рівня [Електронний ресурс] / уклад. Ю. Е. Парфьонов, І. О. Ушакова. – Харків : ХНЕУ ім. С. Кузнеця, 2020. – 62 с.

16. Методологія IDEF0 [Електронний ресурс] – Режим доступу: https://itteach.ru/bpwin/metodologiya-idef0

17. Методологія idef1x [Електронний ресурс] – Режим доступу: https://stud.com.ua/87187/ekonomika/metodologiya_idef1x

18. Мій МедКабінет для турботи про ваше здоров'я [Електронний ресурс] – Режим доступу: https://mymedcabinet.com.ua/

19. Нефункціональні вимоги до програмного продукту [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/231961/

20. Основні стадії розробки дизайну для мобільних пристроїв [Електронний ресурс] – Режим доступу: http://alex devezenko.blogspot.com/2015/01/blog-post.html

21. Поліклініка без черг [Електронний ресурс] – Режим доступу: https://global.newmedicine.com.ua/

22. Принципи відображення предметної області при проектуванні БД [Електронний ресурс] – Режим доступу: https://studfile.net/preview/10045748/page:9/

23. Матеріали Міжнародної науково-практичної конференції молодих учених, аспірантів та студентів — Інформаційні технології в сучасному світі: дослідження молодих вчених : тези доповідей, 17 – 18 лютого 2022 р. – Харків : ХНЕУ імені Семена Кузнеця, 2022. – с. 41.

24. Розгортання програмного забезпечення [Електронний ресурс] – Режим доступу: https://www.wiki.uk ua.nina.az/Розгортання_програмного_забезпечення.html

25. Словник тестувальника [Електронний ресурс] – Режим доступу:

https://qalight.ua/baza-znaniy/slovnik-testuvalnika/

26. Тестування вручну [Електронний ресурс] – Режим доступу: https://uk.education-wiki.com/8800364-manual-testing

27. A progressive Node.js framework for building efficient, reliable and scalable server-side applications. // Nest.js – [Електронний ресурс] – Режим доступу: https://nestjs.com/

28. Електронна система охорони здоров'я в Україні [Електронний ресурс] – Режим доступу: https://ehealth.gov.ua/

29. Helsi – Інформаційна система для пацієнтів [Електронний ресурс] – Режим доступу: https://helsi.me/

30. IDEF0. Знайомство з нотацією та приклади використання [Електронний ресурс] – Режим доступу: https://www.trinion.org/blog/idef0-znakomstvo-s-notaciey-i-primer-ispolzovaniya

31. Node.js® — це JavaScript–оточення побудоване на JavaScript– рушієві Chrome V8. – [Електронний ресурс] – Режим доступу: https://nodejs.org/uk/

32. PostgreSQL: The World's Most Advanced Open Source Relational Database // PostgreSQL – [Електронний ресурс] – Режим доступу: https://www.postgresql.org/

33. React – бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] – Режим доступу: https://ru.reactjs.org/

34. React – початок роботи [Електронний ресурс] – Режим доступу: https://ru.reactjs.org/docs/getting-started.html

35. Sequelize is a modern TypeScript and Node.js ORM for Postgres, MySQL, MariaDB, SQLite and SQL Server, and more. // Sequelize – [Електронний ресурс] – Режим доступу: https://sequelize.org/

ДОДАТОК А
ЛІСТИНГ ОСНОВНИХ ФАЙЛІВ ДОДАТКУ

Файл patient.service.ts

```
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';
import * as bcrypt from 'bcryptjs';

import { Appointment } from 'src/appointment/entities/appointment.entity';
import { ChangePasswordDto } from 'src/auth/dto/change-password.dto';
import { CloudinaryService } from 'src/cloudinary/cloudinary.service';
```

```
import { gendersEnum } from 'src/enums';
import { CreatePatientDto } from './dto/create-patient.dto';
import { UpdatePatientDto } from './dto/update-patient.dto';
import { Patient } from './entities/patient.entity';

const _ = require('lodash');

@Injectable()
export class PatientService {
 constructor(
 @InjectModel(Patient) private patientRepository: typeof Patient,
private cloudinary: CloudinaryService,
 ) {}

 async addPatient(createPatientDto: CreatePatientDto): Promise<Patient> {
const isPatientExist = await this.patientRepository.findOne({  where: {
email: createPatientDto.email },
 });

 if (isPatientExist) {
 throw new HttpException(
 'Patient with this email already exist',
 HttpStatus.BAD_REQUEST,
 );
 }
 return await this.patientRepository.create(createPatientDto);  }

 async getPatientProfile(id: string): Promise<Patient> {
 const patient = await this.patientRepository
 .scope('withoutPassword')
 .findByPk(id);

 if (!patient) {
 throw new HttpException(
 'There is no patient with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 return patient;
 }

 async updatePatient(
 id: string,
 updatePatientDto: UpdatePatientDto,
 ): Promise<Patient> {
 if (_.isEmpty(updatePatientDto)) {
```


Продовження дод. А

```
 throw new HttpException(
 'For updating you need to change at least one property',
HttpStatus.BAD_REQUEST,
 );
 }

 const isPatientExist = await this.patientRepository.findByPk(id);

 if (!isPatientExist) {
```

```
 throw new HttpException(
 'There is no patient with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 if (updatePatientDto.gender) {
 const isGenderCorrect =
 updatePatientDto.gender.toUpperCase() === gendersEnum.MALE ||
updatePatientDto.gender.toUpperCase() === gendersEnum.FEMALE;

 if (!isGenderCorrect) {
 throw new HttpException(
 'Gender should be male or female',
 HttpStatus.BAD_REQUEST,
 );
 }
 }

 const updatedPatient = await this.patientRepository.update( {
...updatePatientDto },
 { where: { id }, returning: true },
 );

 return updatedPatient[1][0];
 }

 async addPatientPhoto(
 id: string,
 photo: Express.Multer.File,
 ): Promise<{ photoName: string }> {
 const patient = await this.patientRepository.findByPk(id);

 if (!patient) {
 throw new HttpException(
 'There is no patient with this id',
 HttpStatus.NOT_FOUND,
 );
 }

 if (patient.photoName) {
 await this.cloudinary.deleteImage(patient.photoName);
 }

 const photoName = await this.cloudinary.uploadImage(photo);
await this.patientRepository.update(
 { photoName },
 { where: { id }, returning: true },
 );
```



```
 return { photoName };
 }

 async deletePatientPhoto(id: string): Promise<void> {
 const patient = await this.patientRepository.findByPk(id);

 if (!patient) {
 throw new HttpException(
 'There is no patient with this ID',
 HttpStatus.NOT_FOUND,
 );
```

```
 }

 await this.cloudinary.deleteImage(patient.photoName);
 await this.patientRepository.update({ photoName: null }, { where: { id } });
return;
 }

 async getPatientByEmail(email: string) {
 return await this.patientRepository.findOne({ where: { email } });  }

 async getPatientAppointments(patientId: string): Promise<Appointment[]> {
const patient = await this.patientRepository.findByPk(patientId, {  include:
'appointments',
 });

 if (!patient) {
 throw new HttpException(
 'There is no patient with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 return patient.appointments;
 }

 async changePatientPassword(
 id: string,
 changePasswordDto: ChangePasswordDto,
 ): Promise<void> {
 const patient = await this.patientRepository.findByPk(id);

 if (!patient) {
 throw new HttpException(
 'There is no patient with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 const passwordEquals = await bcrypt.compare(
 changePasswordDto.oldPassword,
 patient.password,
 );

 if (passwordEquals) {
 const hashNewPassword = await bcrypt.hash(
 changePasswordDto.newPassword,
 5,
 );

 await this.patientRepository.scope('withoutPassword').update(  {
password: hashNewPassword },
 {
```


Продовження дод. А

```
where: { id },
 },
 );
```

```
 return;
 }

 throw new HttpException('Incorrect old password', HttpStatus.BAD_REQUEST);  }

 async restorePatientPassword(
 id: string,
 newPassword: string,
 ): Promise<number[]> {
 const hashPassword = await bcrypt.hash(newPassword, 5);
 return await this.patientRepository.update(
 { password: hashPassword },
 { where: { id } },
 );
 }
}
```

Файл doctor.service.ts

```
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';

import * as bcrypt from 'bcryptjs';
import { classToClassFromExist } from 'class-transformer';
import { Op } from 'Sequelize';

import { Appointment } from 'src/appointment/entities/appointment.entity';
import { ChangePasswordDto } from 'src/auth/dto/change-password.dto';
import { CloudinaryService } from 'src/cloudinary/cloudinary.service';
import { mg } from 'src/emails/emails';
import { Position } from 'src/position/entities/position.entity';
import { CreateDeleteWeekendDto } from 'src/weekend/dto/create-delete
weekend.dto';
import { Weekend } from 'src/weekend/entities/weekend.entity';
import { WeekendService } from 'src/weekend/weekend.service';
import { CreateDoctorDto } from './dto/create-doctor.dto';
import { ReturnDoctorDto } from './dto/return-doctor.dto';
import { UpdateDoctorScheduleDto } from './dto/update-doctor-schedule.dto';
import { UpdateDoctorDto } from './dto/update-doctor.dto';
import { Doctor } from './entities/doctor.entity';
import { Schedule } from './schedule/entities/schedule.entity';
import { ScheduleService } from './schedule/schedule.service';

const _ = require('lodash');
const generator = require('generate-password');

@Injectable()
export class DoctorService {
 constructor(
 @InjectModel(Doctor) private doctorRepository: typeof Doctor,
@InjectModel(Position) private positionRepository: typeof Position,
private scheduleService: ScheduleService,
```



```
private weekendService: WeekendService,
```

```
 private cloudinary: CloudinaryService,
 ) {}

 async addDoctor(createDoctorDto: CreateDoctorDto): Promise<Doctor> {
const isDoctorExist = await this.doctorRepository.findOne({  where: {
email: createDoctorDto.email },
 });

 if (isDoctorExist) {
 throw new HttpException(
 `Doctor with ${createDoctorDto.email} email already exist`,
HttpStatus.BAD_REQUEST,
 );
 }

 const position = await this.positionRepository.findByPk(
 createDoctorDto.positionId,
 );

 if (!position) {
 throw new HttpException(
 'Position is not correct',
 HttpStatus.BAD_REQUEST,
 );
 }

 const password = generator.generate({
 length: 10,
 numbers: true,
 });

 const hashPassword = await bcrypt.hash(password, 5);

 const doctor = await this.doctorRepository.create({
 ...createDoctorDto,
 password: hashPassword,
 });

 await mg.messages
 .create('sandbox3314b96104264ec088e3414f3013f4da.mailgun.org', {
from: 'Mailgun Sandbox
<postmaster@sandbox3314b96104264ec088e3414f3013f4da.mailgun.org>',
to: [createDoctorDto.email],
 subject: 'Your Health doctor account info',
 html: `<h1>Hello, ${createDoctorDto.firstName}
${createDoctorDto.lastName}!</h1>

 <h2>You have been registered in the system <a
href='http://localhost:3000'>Your Health</a> as a doctor.</h2>

 <h3 style='color: #04896c; margin-block-end: 0.3em;'>Your login: <span
style='color: #3a3a3a; text-decoration:
underline'>${createDoctorDto.email}</span></h3>
 <h3 style='color: #04896c; margin-block-start: 0'>Your password: <span
style='color: #3a3a3a; text-decoration: underline'>${password}</span></h3>
 <h3>You can set your own password in your profile.<br/></h3>
```


Продовження дод. А

```
<p style='margin-block-end: 0.3em'>Best,</p>
 <p style='margin-block-start: 0'>Your Health service</p> `,
 })
 .then((msg) => console.log(msg))
 .catch((err) => console.log(err));

 if (createDoctorDto.schedule) {
 for (const oneDaySchedule of createDoctorDto.schedule) {
await this.scheduleService.createSchedule({
 ...oneDaySchedule,
 doctorId: doctor.id,
 });
 }
 }

 return await this.doctorRepository.findByPk(doctor.id, {
 include: ['schedule'],
 });
 }

 async uploadDoctorsFromCsv(csvString: any) {
 const headers = [
 'firstName',
 'lastName',
 'email',
 'positionId',
 'cabinet',
 'duration',
 ];

 const delimiter = ';';

 const rows = csvString
 .slice(csvString.indexOf('') ? csvString.indexOf('') + 1 : 0)
.split('\n');

 const doctors = rows.map(function (row) {
 const values = row.split(delimiter);

 return headers.reduce(function (object, header, index) {
object[header] = values[index];
 return object;
 }, {});
 });

 let createdDoctors = [];

 for (const doctor of doctors) {
 const createdDoctor = await this.addDoctor(doctor);
 createdDoctors.push(createdDoctor);
 }

 return createdDoctors;
 }

 async getDoctorByEmail(email: string) {
 return await this.doctorRepository.findOne({
 where: {
```

```
email,
 },
 });
 }

 async getAllDoctors(
 offset: string = '0',
 ): Promise<{ count: number; rows: ReturnDoctorDto[] }> {
 return await this.doctorRepository
 .scope('withoutPassword')
 .findAndCountAll({
 order: [['firstName', 'ASC']],
 limit: 10,
 offset: +offset,
 });
 }

 async findDoctorByNameOrPosition(
 search: string,
 offset: string = '0',
 ): Promise<{ count: number; rows: ReturnDoctorDto[] }> {
 return await this.doctorRepository
 .scope('withoutPassword')
 .findAndCountAll({
 where: {
 [Op.or]: [
 { firstName: { [Op.substring]: search } },
 { lastName: { [Op.substring]: search } },
 { positionId: { [Op.substring]: search } },
 ],
 },
 order: [['firstName', 'ASC']],
 limit: 10,
 offset: +offset,
 });
 }

 async getDoctorById(id: string): Promise<Doctor> {
 const doctor = await this.doctorRepository
 .scope('withoutPassword')
 .findByPk(id, {
 include: { all: true },
 });

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 return doctor;
 }

 async changeDoctorPassword(
 id: string,
 changePasswordDto: ChangePasswordDto,
```

Продовження дод. А

```
 ): Promise<void> {
 const doctor = await this.doctorRepository.findByPk(id);

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 const passwordEquals = await bcrypt.compare(
 changePasswordDto.oldPassword,
 doctor.password,
 );

 if (passwordEquals) {
 const hashNewPassword = await bcrypt.hash(
 changePasswordDto.newPassword,
 5,
 );

 await this.doctorRepository.update(
 { password: hashNewPassword },
 {
 where: { id },
 returning: true,
 },
 );

 return;
 }

 throw new HttpException('Incorrect old password', HttpStatus.BAD_REQUEST); }

 async restoreDoctorPassword(
 id: string,
 newPassword: string,
 ): Promise<number[]> {
 const hashPassword = await bcrypt.hash(newPassword, 5);
 return await this.doctorRepository.update(
 { password: hashPassword },
 { where: { id } },
 );
 }

 async updateDoctor(
 updateDoctorDto: UpdateDoctorDto,
 id: string,
 ): Promise<Doctor> {
 if (_.isEmpty(updateDoctorDto)) {
 throw new HttpException(
 'For updating you need to change at least one property',
HttpStatus.BAD_REQUEST,
 );
 }
```

Продовження дод. А

```
 const updatedDoctor = await this.doctorRepository.update( {
...updateDoctorDto },
 {
 where: { id },
 returning: true,
 },
 );

 if (!updatedDoctor) {
 throw new HttpException(
 'There is no doctor with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 return updatedDoctor[1][0];
 }

 async addDoctorPhoto(
 id: string,
 photo: Express.Multer.File,
 ): Promise<{ photoName: string }> {
 const doctor = await this.doctorRepository.findByPk(id);

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this id',
 HttpStatus.NOT_FOUND,
 );
 }

 if (doctor.photoName) {
 await this.cloudinary.deleteImage(doctor.photoName);
 }

 const photoName = await this.cloudinary.uploadImage(photo);
await this.doctorRepository.update(
 { photoName },
 { where: { id }, returning: true },
 );
 return { photoName };
 }

 async deleteDoctorPhoto(id: string): Promise<void> {
 const doctor = await this.doctorRepository.findByPk(id);

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 await this.cloudinary.deleteImage(doctor.photoName);
 await this.doctorRepository.update({ photoName: null }, { where: { id } });
```

```
return;
 }

 async deleteDoctor(id: string): Promise<void> {
 const doctor = await this.doctorRepository.findByPk(id, {
include: ['weekends', 'schedule'],
 });
 if (!doctor) {
```


Продовження дод. А

```
throw new HttpException(
 'There is no doctor with this id',
 HttpStatus.NOT_FOUND,
 );
 }

 for (const weekend of doctor.weekends) {
 await this.deleteDoctorWeekend(id, { date: weekend.date }); }

 for (const schedule of doctor.schedule) {
 await this.scheduleService.deleteSchedule(schedule);
 }

 await this.doctorRepository.destroy({ where: { id } });

 return;
 }

 async getDoctorAppointments(id: string): Promise<Appointment[]> {
const doctor = await this.doctorRepository.findByPk(id, { include:
'appointments',
 });

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 return doctor.appointments;
 }

 async getDoctorSchedule(
 id: string,
 ): Promise<{ schedules: Schedule[]; duration: number }> {
 const doctor = await this.doctorRepository.findByPk(id, {
include: 'schedule',
 });

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 return { schedules: doctor.schedule, duration: doctor.duration }; }
```

```
 async updateDoctorSchedule(
 id: string,
 updateDoctorScheduleDto: UpdateDoctorScheduleDto,
 ): Promise<{ schedules: Schedule[]; duration: number }> {
 const updatedDoctor = await this.doctorRepository.update( {
duration: updateDoctorScheduleDto.duration },
 { where: { id } },
 );
```


Продовження дод. А

```
 if (!updatedDoctor[0]) {
 throw new HttpException(
 'There is no doctor with this ID',
 HttpStatus.NOT_FOUND,
 );
 }

 const updatedSchedules = [];

 for (const schedule of updateDoctorScheduleDto.schedules) { const
updatedSchedule = await this.scheduleService.updateSchedule( schedule,
 );
 updatedSchedules.push(updatedSchedule);
 }

 return {
 schedules: updatedSchedules,
 duration: updateDoctorScheduleDto.duration,
 };
 }

 async addDoctorWeekend(
 id: string,
 createDoctorWeekendDto: CreateDeleteWeekendDto,
 ): Promise<Weekend> {
 const doctor = await this.doctorRepository.findByPk(id);

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this id',
 HttpStatus.NOT_FOUND,
 );
 }

 return await this.weekendService.addWeekend(id, createDoctorWeekendDto); }

 async deleteDoctorWeekend(
 id: string,
 deleteDoctorWeekend: CreateDeleteWeekendDto,
 ): Promise<void> {
 const doctor = await this.doctorRepository.findByPk(id);

 if (!doctor) {
 throw new HttpException(
 'There is no doctor with this id',
 HttpStatus.NOT_FOUND,
```

```
 );
 }

 await this.weekendService.deleteWeekend(id, deleteDoctorWeekend);
return;
 }
}
```


Продовження дод. А

Файл admin.service.ts

```
import { HttpException, HttpStatus, Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';
import * as bcrypt from 'bcryptjs';

import { ChangePasswordDto } from 'src/auth/dto/change-password.dto';
import { CreateAdminDto } from './dto/create-admin.dto';
import { UpdateAdminDto } from './dto/update-admin.dto';
import { Admin } from './entities/admin.entity';

const _ = require('lodash');

@Injectable()
export class AdminService {
 constructor(@InjectModel(Admin) private adminRepository: typeof Admin) {}

 async addAdmin(createAdminDto: CreateAdminDto): Promise<Admin> {
const isAdminExist = await this.adminRepository.findOne({ where: {
email: createAdminDto.email },
 });

 if (isAdminExist) {
 throw new HttpException(
 `Admin with ${createAdminDto.email} email already exist`,
HttpStatus.BAD_REQUEST,
 );
 }

 const hashPassword = await bcrypt.hash(createAdminDto.password, 5);

 return await this.adminRepository.create({
 ...createAdminDto,
 password: hashPassword,
 });
 }

 async getAdminProfile(id: string): Promise<Admin> {
 const admin = await this.adminRepository
 .scope('withoutPassword')
 .findByPk(id);

 if (!admin) {
 throw new HttpException(
 'There is no admin with this id',
 HttpStatus.NOT_FOUND,
 );
 }
```

```
return admin;
}

async updateAdmin(
id: string,
updateAdminDto: UpdateAdminDto,
): Promise<Admin> {
if (_.isEmpty(updateAdminDto)) {
throw new HttpException(
'For updating you need to change at least one property',
```


Продовження дод. А

```
HttpStatus.BAD_REQUEST,
);
}

const isAdminExist = await this.adminRepository.findByPk(id);

if (!isAdminExist) {
throw new HttpException(
'There is no admin with this ID',
HttpStatus.NOT_FOUND,
);
}

const updatedAdmin = await this.adminRepository.update(
{ ...updateAdminDto },
{ where: { id }, returning: true },
);
return updatedAdmin[1][0];
}

async getAdminByEmail(email: string): Promise<Admin> {
return await this.adminRepository.findOne({ where: { email } }); }

async changeAdminPassword(
id: string,
changePasswordDto: ChangePasswordDto,
): Promise<void> {
const admin = await this.adminRepository.findByPk(id);

if (!admin) {
throw new HttpException(
'There is no admin with this ID',
HttpStatus.NOT_FOUND,
);
}

const passwordEquals = await bcrypt.compare(
changePasswordDto.oldPassword,
admin.password,
);

if (passwordEquals) {
const hashNewPassword = await bcrypt.hash(
changePasswordDto.newPassword,
5,
```

```
  );

  await this.adminRepository.update(
  { password: hashNewPassword },
  {
  where: { id },
  },
  );

  return;
  }

  throw new HttpException('Incorrect old password', HttpStatus.BAD_REQUEST);
```


Закінчення дод. А

```
  }

  async restoreAdminPassword(id: string, newPassword: string) {
const hashPassword = await bcrypt.hash(newPassword, 5);
  return await this.adminRepository.update(
  { password: hashPassword },
  { where: { id } },
  );
  }
}
```