Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Інтелектуальна дерматоскопічна система виявлення меланом

(тема)

Виконав:
студент 2 курсу, групи СШІм-21-1
Кондратюк О. С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Аврунін О. Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кондратюк Олені Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальна дерматоскопічна система виявлення меланом

затверджена наказом університету від 31 березня 20 23 р. № 306Ст

2. Термін подання студентом роботи до екзаменаційної комісії 16 травня 20 23 р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел та відомих наукових проектів щодо розробки та дослідження систем машинного зору з використанням штучного інтелекту, Microsoft Developer Network (MSDN) documentation

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Методи та алгоритми машинного зору
3) Експериментальне моделювання та навчання моделі
4) Розробка інтелектуальної дерматоскопічної системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	06.02 – 19.02.2023	виконано
2	Опис системи	20.02 – 05.03.2023	виконано
3	Моделювання та навчання моделі	06.03 – 27.03.2023	виконано
4	Визначення проблем при розробці системи	28.03 – 04.04.2023	виконано
5	Знаходження і вирішення проблем	05.04 – 10.04.2023	виконано
6	Розробка інтелектуальної дерматоскопічної системи	01.04 – 15.04.2023	виконано
7	Написання пояснювальної записки	10.04 – 20.04.2023	виконано
8	Попередній захист	10.05.2023	виконано
9	Захист перед ЕК	16.05.2023	виконано

Дата видачі завдання 3 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Аврунін О. Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 117 с., 41 рис., 7 табл., 3 додатка, 23 джерела.

БАЗА ДАНИХ, ДЕРМАТОСКОПІЧНА СИСТЕМА, МЕЛАНОМА, НЕВУС, НЕЙРОННА МЕРЕЖА, ШТУЧНИЙ ІНТЕЛЕКТ, C#, ENTITY FRAMEWORK CORE, MVC, NEURAL NETWORK, SQL, .NET

Об'єктом дослідження є дерматоскопічна система.

Предметом дослідження є інтелектуальна дерматоскопічна система яка має функцію виявлення меланом за фото.

Мета роботи – розробити підсистему для автоматизації роботи дерматологічної клініки з підтримкою аналізу новоутворень на шкірі по фотографії.

Методи розробки базуються на можливостях мови програмування C# та будуть використані для аналізу фотографій новоутворень на шкірі, а також автоматизації обліку записів до дерматологічної клініки.

Підсистема, що розроблюється, дозволяє автоматизувати облік записів до дерматологічної клініки та має функцію аналізу фотографій для виявлення меланом зпоміж доброякістніх невусів. Такий аналіз проводиться базуючись на фото новоутворення на шкірі. За допомогою веб-сайту дерматологічної клініки, користувачі можуть переглядати доступні послуги та список лікарів і робити запис до дерматологічної клініки. А також, скориставшись функцією аналізу новоутворень за фото, одразу визначити передбачуваний діагноз.

ABSTRACT

Explanatory note: 117 p., 41 fig., 7 tabl., 3 ann., 23 sources.

ARTIFICIAL INTELLIGENCE, C#, DATABASE, DERMATOSCOPIC SYSTEM, ENTITY FRAMEWORK CORE, MELANOMA, MVC, NAEVUS, NEURAL NETWORK, NEURON NETWORK, SQL, .NET

The object of research is the dermatoscopic system.

The subject of the research is an intelligent dermatoscopic system that has the function of detecting melanoma by photo.

The purpose of the work is to develop a subsystem for automating the work of a dermatology clinic with support for the analysis of neoplasms on the skin by photography.

The development methods are based on the capabilities of the C# programming language and will be used to analyze photos of neoplasms on the skin, as well as to automate the accounting of entries to the dermatology clinic.

The subsystem under development allows to automate the accounting of records to the dermatology clinic and has the function of photo analysis to detect melanoma among benign nevi. Such an analysis is carried out based on a photo of a neoplasm on the skin. Using the dermatology clinic website, users can view the available services and list of doctors and make an appointment with the dermatology clinic. And also, using the function of analyzing neoplasms by photo, immediately determine the expected diagnosis.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

СУБД – Система управління базою даних;

ІС – Інформаційна система;

CRUD – Create-Read-Update-Delete – Абревіатура складена з перших літер;

DFD – Data Flow Diagram – Діаграма потоків даних;

DNN – Deep Neural Network – Глибинна нейронна мережа;

ERD – Entity-Relationship diagram – ДІаграма «сутність-зв'язок»;

IDEF0 – Integration Definition for Function Modeling – Нотація опису бізнес-процесів. Основана на методології SADT;

ISIC – International Skin Imaging Collaboration – Абревіатура складена з перших літер;

MVC – Model-View-Controller – Модель-Представлення-Контролер;

SADT – Structured Analysis and Design Technique – Технологія структурного аналізу та проектування, графічне позначення і підхід до опису систем;

SOLID – Single responsibility principle, Open/closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle – Абревіатура складена з перших літер п'яти базових принципів об'єктно-орієнтованого програмування.

ВСТУП

У наш час стрімко розвиваються інформаційні технології та все більше людей надають перевагу тому, щоб вирішувати більшість завдань за допомогою Інтернету. В нашій країні є багато лікарень та приватних дерматологічних клінік, але в більшість з них, щоб не чекати в черзі, потрібно прийти заздалегідь щоб записатись на прийом або зателефонувати.

Цю проблему міг би вирішити зручний веб-звстосунок, що полегшував би процес запису до дерматологічної клініки, а також надавав би можливість ознайомитись з усім асортиментом послуг не виходячи з дому.

Ця робота є актуальною, адже щорічно, кількість подібних сервісів в Інтернеті збільшується тому, що це не тільки зручно для клієнтів, але й вигідно власникам таких сервісів. Також, такий додаток дозволяє записатись до дерматологічної клініки не виходячи з дому, що є зараз актуальним. Через автоматизацію процесу створення і формування запису, підвищується надійність за рахунок зменшення впливу людського фактору, а також зменшується кількість та трудомісткість дій персоналу.

Крім того, клієнти мають доступ до сайту цілодобово, це зручно для клієнтів, адже вони можуть записатись до дерматологічної клініки у будь який час.

У наш час, багато дерматологічних клінік мають подібний веб-застосунок, що дозволяє записуватися на прийом за допомогою Інтернету, і для того, щоб привабити ще більше нових клієнтів, подібні сервіси намагаються виділитися з поміж інших, надаючи якісь унікальні можливості, які б не надавали конкуренти.

Веб-застосунок, що розроблюється, дозволятиме клієнтам ознайомитись з асортиментом послуг, переглянути інформацію щодо лікарів та записатись до дерматологічної клініки через Інтернет, а також,

матиме унікальну з поміж інших подібних сервісів функцію: аналіз новоутворення на шкірі по фотографії та висування припущення щодо діагнозу.

Завдяки додатковій функції аналізу новоутворень по фотографії, підсистема, що розроблюється, буде вигідно виділятись з поміж інших подібних веб-застосунків, ажде користувачі можуть не виходячи з дому завантажити фотографію турбуючого невуса або іншого новоутворення у систему і отримати результат чи є це новоутворення доброякісним або злоякісним.

У наш час, коли інтернет є доступнім майже кожному, багато людей у разі поганого самопочуття перш за все намагаються самостійно визначити свій діагноз шукаючи свої симптоми в інтернеті, такий спосіб може приводити до того, що людина відносить свої симптоми до рідкісної смертельної хвороби, якої в неї немає, або до хвороби, яка не потребує лікування і проходить сама з часом, коли на справді, має хворобу яку потрібно негайно почтаи лікувати щоб уникнути погіршення стану. Хоча діагноз, який савить веб-застосунок, що розроблюється, у ході аналізу фотографії – не є сто відсотковим і у будь якому разі якщо є турбуючи новоутворення на шкірі – потрібно звернутись до лікаря, такий діагноз ставиться базуючись на конкретній фотографії використовуючи технології штучного інтелекту, що робить його більш надійним ніж спроба знайти свій діагноз в інтернеті за допомогою текстового опису симптомів не маючи спеціальних медичних знань.

Інформація, яку надає підсистема, що розроблюється, допоможе людині прийняти рішення швидше записатись на прийом у разі виявлення відхилень від норми, або у разі, якщо відхилень від норми не виявлено – заспокоїти її на час очікування прийому.

Для проектування було обрано мову програмування C#, ASP.NET Core MVC, адже у ній чітко розділена логіка програми, а це спрощує роботу над великими проектами.

Для зберігання даних будо обрано систему управління базами даних (СУБД) Microsoft SQL.

Для роботи зі штучним інтелектом було обрано використовувати бібліотеки мови програмування C#, такі як:

– бібліотека Microsoft.ML для машинного навчання, яка надає інтерфейси для створення, тренування та використання моделей машинного навчання. Вона включає в себе різноманітні алгоритми машинного навчання, які можна використовувати для розробки додатків з різноманітними функціональними можливостями.

– microsoft.ML.ImageAnalytics та Microsoft.ML.Vision, що є додатковими компонентами для Microsoft.ML, які надають інструменти для обробки та аналізу зображень. Вони дозволяють використовувати різноманітні алгоритми машинного навчання для обробки зображень, такі як класифікація зображень, виявлення облич, розпізнавання тексту на зображенні та інші.

– sciSharp.TensorFlow.Redist – пакет для .NET, який дозволяє використовувати TensorFlow, популярну бібліотеку машинного навчання, у додатках на .NET. TensorFlow надає інструменти для розробки та тренування різноманітних моделей машинного навчання, які можуть бути використані для різних задач.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Дерматологічна клініка – це медичнаі установа, яка спеціалізуються на діагностиці та лікуванні захворювань шкіри, волосся та нігтів. У цих клініках працюють дерматологи, які пройшли спеціалізовану підготовку з діагностики та лікування шкірних захворювань.

У дерматологічній клініці пацієнти можуть отримати широкий спектр медичних та косметологічних послуг. Деякі з найпоширеніших медичних послуг, які пропонуються в дерматологічних клініках, включають скринінг раку шкіри, видалення родимок, лікування таких захворювань, як псоріаз, екзема та розацеа, а також лікування випадіння волосся.

Косметичні дерматологічні послуги, які можуть бути доступні в дерматологічній клініці, включають ін’єкції ботокса, дерматологічні філлери, лазерне лікування для відновлення шкіри та видалення волосся, а також хімічний пілінг. Дерматологічні клініки також можуть запропонувати спеціалізоване лікування дитячих шкірних захворювань, таких як родимки, екзема та акне.

Дерматологічні клініки можуть працювати самостійно або бути частиною більшого медичного закладу. Зазвичай вони оснащені спеціальним обладнанням для діагностики та лікування різних захворювань шкіри. Наприклад, у дерматологічній клініці може бути дерматоскоп, який є спеціальним інструментом, який дерматологи використовують для детального огляду уражень шкіри. Вони також можуть мати доступ до передових методів лікування та процедур, таких як операція Мооса, яка є високоефективним методом видалення раку шкіри.

Найпоширеніші медичні послуги, які пропонують дерматологічні клініки:

– скринінг раку шкіри: дерматологи проводять скринінг, щоб виявити ранні ознаки раку шкіри. Під час обстеження дерматолог огляне шкіру на наявність будь-яких підозрілих родимок, уражень або наростів. У разі виявлення будь-яких підозрілих плям може бути проведена біопсія, щоб визначити, чи є вони раковими.

– видалення родимок: дерматологи можуть рекомендувати видалення родимок, які є підозрілими або потенційно раковими, або родимок, які косметично небажані.

– лікування псоріазу. Псоріаз – це хронічне аутоімунне захворювання, яке спричиняє надто швидкий ріст клітин шкіри, що призводить до появи товстих лускатих плям на шкірі. Дерматологи можуть запропонувати різноманітні методи лікування псоріазу, включаючи місцеві креми, світлотерапію та пероральні препарати.

– лікування екземи. Екзема – це хронічне захворювання, яке викликає сухість, свербіж і запалення шкіри. Дерматологи можуть рекомендувати різноманітні методи лікування екземи, включаючи місцеві кортикостероїди, зволожуючі засоби та пероральні препарати.

– лікування розацеа. Розацеа – це хронічне захворювання шкіри, яке викликає почервоніння, гіперемію та нерівності на обличчі, схожі на прищі. Дерматологи можуть запропонувати різноманітні методи лікування розацеа, включаючи місцеві креми, пероральні антибіотики та лазерну терапію.

– лікування випадіння волосся. Випадіння волосся може бути спричинено різними захворюваннями. Лікар-дерматолог діагностує та лікує захворювання волосся та шкіри голови. Підрозділ дерматології, який вивчає ці хвороби, називається трихологія. До послуг трихолога необхідно звертатись у тих випадках, коли потрібно визначити себорею, трихолазію або різні види облисіння. Варіанти лікування можуть включати місцеві або пероральні препарати, операцію з пересадки волосся або терапію збагаченої тромбоцитами плазми (PRP).

Найпоширеніші косметичні та спеціалізовані послуги дитячої дерматології, які пропонуються в дерматологічних клініках:

– ін’єкції ботокса: ботокс – це очищена форма ботулотоксину, яка може тимчасово паралізувати м’язи обличчя, щоб зменшити появу зморшок і тонких ліній. Дерматологи можуть запропонувати ін’єкції ботокса як нехірургічне косметичне лікування, щоб допомогти пацієнтам досягти більш молодого вигляду.

– дерматологічні філлери – це речовини для ін’єкцій, які можна використовувати для заповнення зморшок і додання об’єму обличчю. Поширені типи філлерів включають наповнювачі з гіалуроновою кислотою, які можна використовувати для заповнення носогубних складок і підтягування губ, і наповнювачі, що стимулюють колаген, які можна використовувати для додання об’єму щокам і скроням.

– лазерне лікування: можна використовувати для різноманітних косметичних цілей, зокрема для відновлення шкіри, видалення волосся та лікування проблем пігментації. Дерматологи можуть запропонувати ряд лазерних процедур, включаючи абляційні та неабляційні лазери.

– хімічний пілінг: це вид косметичного лікування, який передбачає нанесення на шкіру розчину для видалення верхніх шарів мертвих клітин шкіри. Це може допомогти зменшити появу тонких ліній, зморшок і шрамів від прищів, а також може покращити текстуру та тонус шкіри.

– дитяча дерматологія. Педіатрична дерматологія – це спеціалізована область дерматології, яка зосереджена на діагностиці та лікуванні захворювань шкіри у дітей. Дерматологи, які спеціалізуються на дитячій дерматології, можуть запропонувати лікування широкого кола захворювань, включаючи родимі плями, екзему, вугри та бородавки.

Дерматологи навчені діагностувати та лікувати широкий спектр захворювань, що впливають на шкіру, волосся та нігті. Вони можуть допомогти пацієнтам впоратися зі своїми симптомами, покращити стан шкіри та знизити ризик розвитку таких серйозних захворювань, як рак

шкіри. Зазвичай, вони проходять тривалу підготовку з дерматології та володіють знаннями та навичками для діагностики та лікування широкого спектру захворювань шкіри, від акне та псоріазу до раку шкіри та інших дерматологічних захворювань. Щоб стати дерматологом, людина зазвичай закінчує чотирирічний ступінь бакалавра, після чого чотири роки навчання в медичній школі, однорічне стажування та трирічну ординатуру з дерматології. Деякі дерматологи також можуть пройти додаткову стипендію в спеціалізованій галузі дерматології. Під час ординатури з дерматології лікар проходить спеціалізовану підготовку з діагностики та лікування захворювань шкіри, волосся та нігтів. Це включає в себе навчання з дерматопатології, яка є вивченням захворювань шкіри на мікроскопічному рівні. Лікар також пройде навчання з косметичної дерматології, яка включає такі процедури, як ін'єкції ботокса, дерматологічні філлери та лазерне лікування.

Дерматологічні клініки є важливою частиною системи охорони здоров'я та відіграють вирішальну роль у діагностиці та лікуванні широкого спектру захворювань шкіри і допомагають пацієнтам досягти оптимального здоров'я шкіри:

– діагностика: Візуальний аналіз медичного зображення дозволяє лікарю зробити попередній висновок про стан шкіри в конкретний період захворювання [1]. Також, дерматологи навчені використовувати спеціалізоване обладнання та методи для діагностики захворювань шкіри, включаючи рак шкіри, екзему, псоріаз, акне та розацеа. Дерматологічні клініки також можуть пропонувати скринінг раку шкіри, який може допомогти виявити рак шкіри на ранніх стадіях, коли він найбільш піддається лікуванню або конфокальну скануючу лазерну мікроскопію шкіри [2].

– лікування: дерматологи пропонують низку методів лікування захворювань шкіри, включаючи ліки, що відпускаються за рецептом, місцеві креми та процедури в офісі, такі як кріотерапія, видалення родимок

і фототерапія. Косметичні дерматологічні послуги, такі як ін’єкції ботокса, дерматологічні філлери, лазерне лікування та хімічні пілінги, також доступні в дерматологічних клініках, щоб допомогти пацієнтам отримати більш молодий і оновлений вигляд. Дерматологічні клініки забезпечують постійний догляд і лікування хронічних захворювань шкіри, таких як псоріаз та екзема. Це може включати регулярні огляди, коригування доз ліків і надання порад щодо того, як контролювати симптоми та уникати тригерів.

– надання рекомендацій: дерматологи надають пацієнтам поради щодо того, як підтримувати здорову шкіру та уникати її пошкоджень. Це може включати рекомендації щодо засобів догляду за шкірою, поради щодо захисту від сонця та інформацію про фактори способу життя, які можуть вплинути на здоров’я шкіри.

Для того, щоб скористатися послугами дерматологічної клініки потрібно записатися на прийом заздалегідь або одразу прийти, і в разі, якщо немає черги, можна буде скористатись послугами. Записатись заздалегідь можна по телефону або за допомогою веб-сайту обраної дерматологічної клініки, якщо такий існує. У цьому випадку є можливість переглянути каталог всіх послуг та інформацію щодо працюючих в клініці лікарів перед тим, як записатись, що дозволяє одразу дізнатись чи надає саме ця клініка потрібні послуги і полегшує вибір для клієнта. Перевагою також є те, що сайт працює цілодобово, що дає можливість робити запис у будь-який час навіть тоді, коли клініка вже зачинена.

Запис до дерматологічної клініки через сайт значно економить час клієнта, адже йому не потрібно виходити з дому, щоб записатись, а також, через автоматизацію процесу створення і формування запису, підвищується надійність за рахунок зменшення впливу людського фактору, а також зменшується кількість та трудомісткість дій персоналу.

Унікальною особливістю з поміж інших подібних підсистем, буде функція аналізу фотографії новоутворення на шкірі клієнта і постановка передбачуваного діагнозу одразу на сайті клініки.

Хоча для того, щоб пересвідчитись у тому, що новоутворення є доброякісним – потрібно робити біопсію, доброякісні родимки зазвичай відрізняються від потенційно раковиих візуально. Дерматоскопія це метод безконтактної діагностики, що дозволяє достовірно визначити природу новоутворення на шкірі. Особливість даного методу полягає в тому, що він дозволяє встановити ступінь небезпеки будь-якого новоутворення на ранній стадії [3].

Доброякісні невуси (також відомі як родимки) зазвичай являють собою невеликі коричневі або чорні плями на шкірі, які зазвичай нешкідливі. Вони викликані скупченням пігментних клітин, і більшість людей мають принаймні кілька таких на шкірі.

Меланома – це тип раку шкіри, який розвивається в клітинах, що виробляють пігмент (меланоцитах). На відміну від доброякісних невусів, меланома може бути небезпечною для життя, якщо її не виявити та не лікувати вчасно. Меланома часто виглядає як темна родимка неправильної форми, яка з часом росте та змінюється.

Є кілька ключових відмінностей між доброякісними невусами та меланомою, зокрема:

– зовнішній вигляд: доброякісні невуси зазвичай невеликі, округлої або овальної форми та мають чітко окреслену межу. Зазвичай вони мають рівномірний колір та можуть бути плоскими або випуклими. З іншого боку, меланоми часто мають неправильну форму та межі, а також можуть бути асиметричними. Вони також можуть мати кілька кольорів, включаючи відтінки коричневого, чорного, синього або червоного.

– ріст: доброякісні невуси, як правило, з часом залишаються незмінними за розміром і формою або можуть рости дуже повільно.

Меланоми, з іншого боку, можуть швидко рости та ставати досить великими.

– структура: доброякісні невуси, як правило, гладкі та правильної текстури. Меланоми можуть мати нерівну поверхню і мати бугорці.

– симптоми: доброякісні невуси зазвичай не викликають жодних відчуттів, таких як біль або свербіж. Меланоми можуть викликати свербіж, кровотечу або утворення кірок.

Якщо на шкірі є родимка або пляма, яка турбує, важливо звернутися до медичного працівника, щоб визначити, чи є вони доброякісними або потенційно раковими.

Приклад доброякісного невуса зображено на рисунку 1.1, а приклад злоякістного невуса зображено на рисунку 1.2.

Рисунок 1.1 – Приклад доброякісного невуса

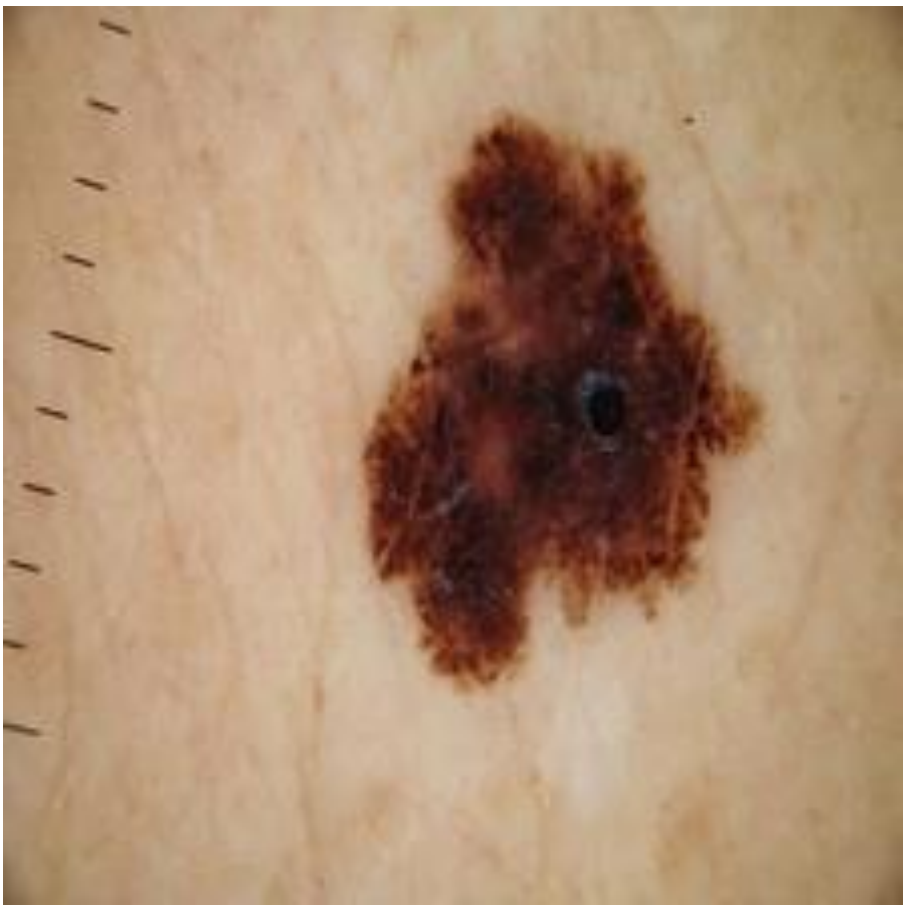


Рисунок 1.2 – Приклад злоякістного невуса

Визначення характеру новоутворення на шкірі за фотографією відбувається з використанням технологій штучного інтелекту. Для цього використовується модель машинного навчання, якак була навчена на попередньо обробленому наборі даних розрізняти меланоми та невуси за ознаками на знімках.

Для такого машинного навчання перш за все потрібно мати набір фотографій, на яких буде проводитись навчання моделі. Для застосунка, що розроблюється, у якості даних для навчання було використано датасет International Skin Imaging Collaboration (ISIC) 2017.

Набір даних International Skin Imaging Collaboration (ISIC) – це колекція зображень уражень шкіри, що доступна для дослідницьких цілей у галузі дерматології та машинного навчання. Набір даних містить тисячі зображень різних типів уражень шкіри, які були анотовані дерматологами.

Набір даних ISIC був створений у відповідь на потребу в кращих інструментах діагностики раку шкіри, який є найпоширенішим видом раку в усьому світі. Набір даних був розроблений, щоб полегшити розробку алгоритмів машинного навчання для раннього виявлення раку шкіри. Набір даних ISIC використовувався для багатьох дослідницьких проектів, пов'язаних із діагностикою уражень шкіри та машинним навчанням. Було показано, що алгоритми машинного навчання, навчені на базі даних ISIC, можуть досягти високого рівня точності в діагностиці уражень шкіри, демонструючи потенціал цих алгоритмів для підвищення точності та ефективності діагностики раку шкіри.

Після того, як датасет було зібрано, данні в ньому потрібно позначити відповідним класом, у даному випадку всі фотографії було поділено на дві папки: доброякістні на злоякістні і імя цих папок було використано у якості класу зображення.

Далі, дані потрібно перемішати щоб переконатися, що приклади не впорядковані певним чином, що може внести упередження в модель.

Після попередньої обробки даних їх потрібно розділити на набори для навчання, перевірки та тестування. Навчальний набір використовується для навчання моделі, набір перевірки використовується для налаштування гіперпараметрів, а набір для тестування використовується для оцінки продуктивності моделі.

Мета обробки даних полягає в тому, щоб переконатися, що набір даних є репрезентативним, неупередженим і містить достатньо прикладів, на яких модель може вчитися. Після того, як дані підготовані, можна переходити до навчання моделі.

Для навчання моделі було використано алгоритм DNN (Deep Neural Network) – це тип алгоритму машинного навчання, який базується на штучних нейронних мережах із кількома рівнями. DNN складається з кількох прихованих шарів, кожен з яких складається з кількох штучних нейронів, які використовуються для вилучення ознак із вхідних даних.

Вихідні дані кожного рівня проходять через функцію нелінійної активації, яка дозволяє мережі вивчати складні шаблони в даних.

DNN продемонстрував надзвичайний успіх у широкому діапазоні програм, таких як комп’ютерне бачення, обробка природної мови та розпізнавання мови. Однією з ключових переваг DNN є здатність автоматично витягувати високорівневі характеристики з необроблених даних без необхідності явної розробки функцій.

Навчання DNN включає процес, званий зворотним розповсюдженням, який використовується для оновлення вагових коефіцієнтів нейронної мережі на основі помилки між прогнозованим виходом і фактичним виходом. Цей процес передбачає обчислення градієнта помилки щодо ваг мережі та використання цього градієнта для оновлення ваг у протилежному напрямку градієнта, щоб мінімізувати помилку.

Після навчання, модель зможе визначити доброякісний невус чи злоякісний зображений на фотографії.

1.2 Визначення бізнес-процесів, що вимагають автоматизації

Неавторизований користувач, зайшовши на сайт, може скористатись функцією аналізу фотографії новоутворення і ознайомитися з каталогом послуг та інформацією щодо лікарів з головної сторінки, щоб обрати послугу та бажаного лікаря. Після цього, щоб записатись до дерматологічної клініки, якщо клієнт ще не має облікового запису у системі, він повинен пройти процедуру реєстрації, або, якщо він вже має обліковий запис – авторизуватися.

Далі, якщо клієнт хоче оформити запис, він повинен перейти на відповідну сторінку на сайті та вибрати бажані процедури, лікаря та дату і час. Після того, як заявка на запис буде сформована, її підтвердить адміністратор, переглянути свої активні записи клієнт може на сторінці

особистого кабінету. Для перевірки новоутворення на шкірі онлайн клієнт повинен перейти на відповідну сторінку та завантажити фотографію турбуючого новоутворення, після чого система видасть результат.

Автоматизація обліку записів до клініки прибере зайвий документообіг та зменшить обсяг та трудомісткість дій персоналу, зменшивши ймовірність помилки при обробці записів клієнтів. Перелік функцій, що входять в межі предметної області:

Функції усіх користувачів:

1) перегляд інформації з сайту (надавані послуги, їх опис, ціни, інформація щодо лікарів);
2) реєстрація на сайті;
3) аналіз новоутворення за фото.

Функції зареєстрованих користувачів:

1) вхід на сайт;
2) запис до клініки;
3) перегляд списку своїх записів;
4) вихід.

1.3 Постановка завдання

Метою роботи є розробка підсистеми для автоматизації роботи дерматологічної клініки з функцією аналізу новоутворення на шкірі за фото. Підсистема, що розроблюється, повинна автоматизувати облік записів до дерматологічної клініки та мати функцію розпізнавання злоякістніх та доброякісних новоутворень на шкірі.

Для програмної реалізації використовується: мова програмування C#, Microsoft Visual Studio 2022, технології ASP.NET Core MVC та Entity Framework Core.

Для машинного навчання було використано Microsoft.ML, що є це бібліотекою машинного навчання з відкритим кодом, розробленою

Microsoft. Вона створениа як гнучка і масштабована структура, яку можна використовувати для створення широкого діапазону моделей машинного навчання, від простої лінійної регресії до складних моделей глибокого навчання.

Також була використана SciSharp.TensorFlow.Redist, що є бібліотекою, яка надає розробникам .NET доступ до різноманітних функцій TensorFlow. ЇЇ підтримка широкого спектру моделей глибокого навчання, розширюваність і підтримка розподіленого навчання роблять її привабливим вибором для тих, хто працює в екосистемі .NET.

2 ПРОЕКТУВАННЯ СИСИТЕМИ

2.1 Визначення функціональних вимог до системи

2.1.1 Опис функціональних вимог у вигляді концептуальної діаграми

Щоб створити функціональну модель інформаційної системи, необхідно використовувати стандарт IDEF0 як точку зору IT-адміністратора. Це дозволить врахувати бізнес-процеси системи, знизити ризики неправильного впровадження автоматизованих бізнес-процесів. Модель, створена за методологією SADT, являє собою діаграму, фрагмент тексту та глосарій із посиланнями один на одного.

Діаграма – це графічне зображення системи або процесу. Вона складається з блоків і дуг, які представляють різні аспекти системи. Місце з'єднання дуги з блоком визначає тип інтерфейсу.

Контрольна інформація надходить у блок зверху, тоді як інформація, яка підлягає обробці, відображається зліва від блоку, а результати виведення – з правого боку.

Щоб змоделювати систему, потрібно розробити концептуальну модель і модель декомпозиції. Метою моделювання є визначення та специфікація функціональних вимог до інформаційної системи. У системі були реалізовані такі основні функції: вибір клієнтом бажаних послуг, лікаря, дати та часу, створення заявки на запис до дерматологічної клініки і підтвердження її адміністратором. Адміністратор (редактор) був обраний як точка зору.

Контекстна діаграма у стандарті IDEF0 представлена на рисунку 2.1.

Концептуальна діаграма представлена на рисунку 2.2

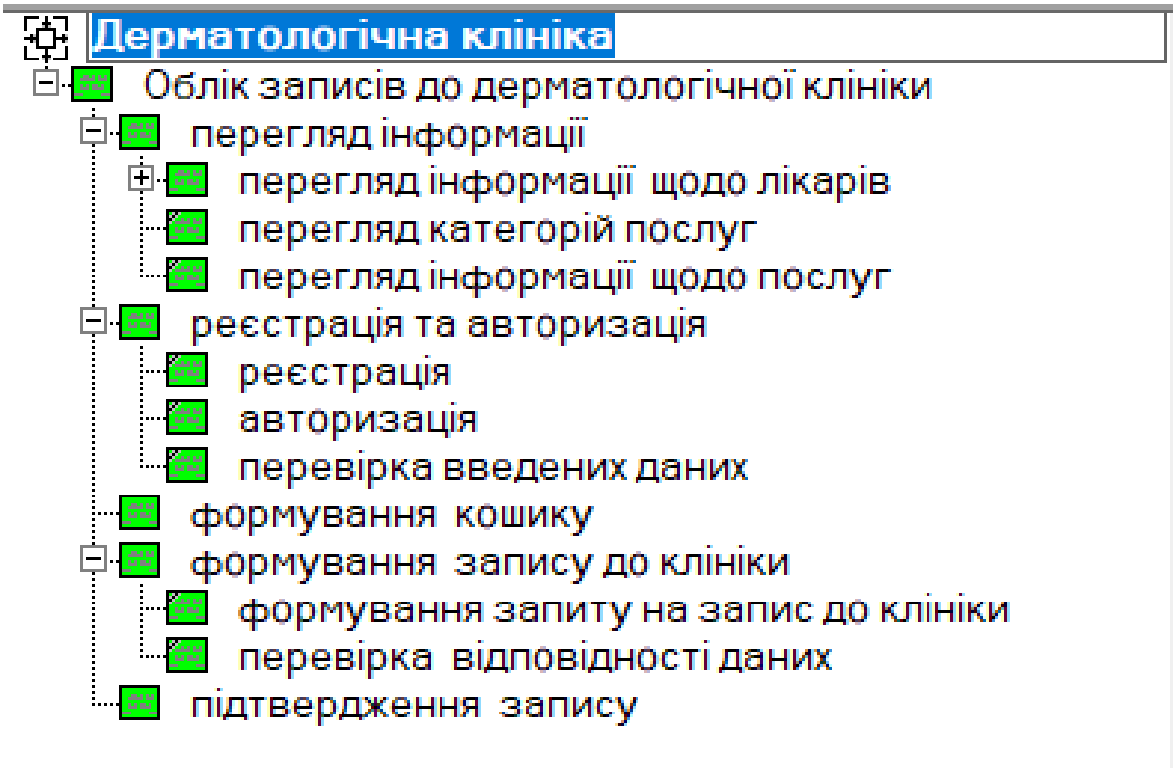


Рисунок 2.1 – Контекстна діаграма

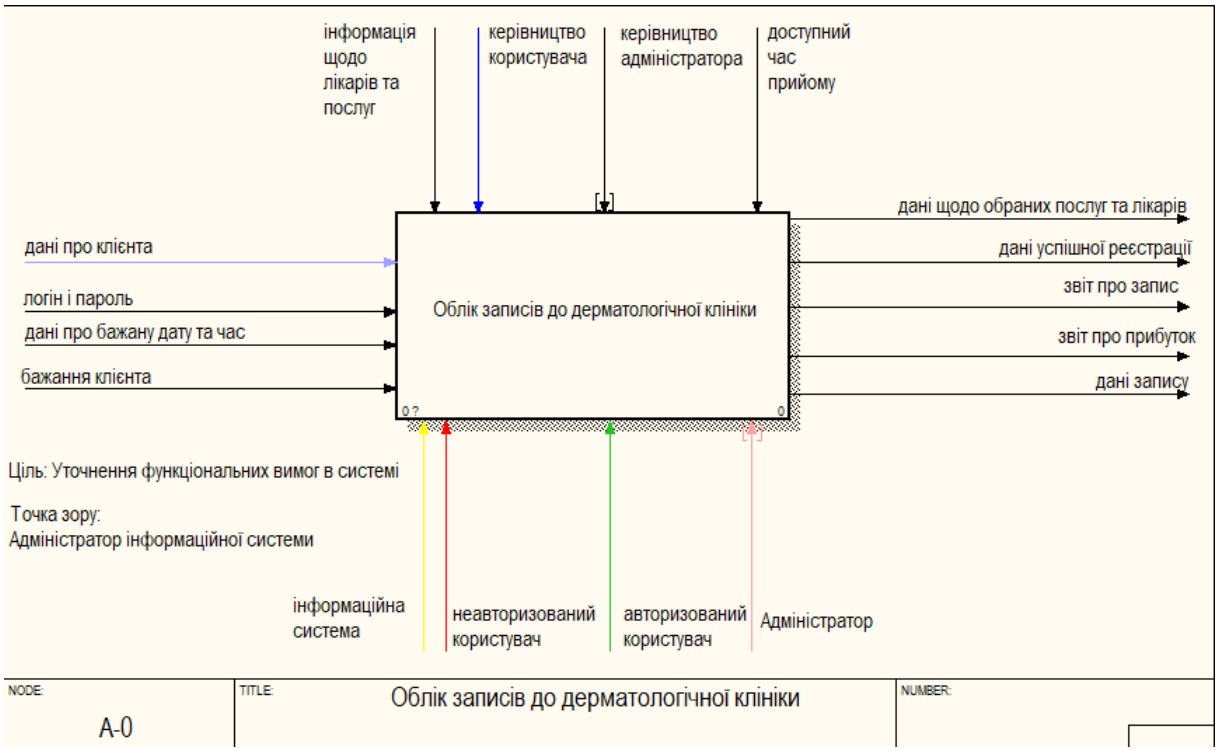


Рисунок 2.2 – Концептуальна діаграма

На схемі показано, що інформація про клієнта – його логін та пароль, дані про бажану дату та час і побажання клієнта подається на вхід.

Виходом системи, її результатом роботи, є інформація щодо обраних послуг, дані щодо запису, дані успішної реєстрації (логін та пароль), якщо вона відбулась, а також звіт про виконання послуг та звіт про прибуток.

Функціонування системи реалізується за допомогою нормативної документації – керівництво користувача та керівництво адміністратора.

Модель SADT (Серія діаграм і документації) забезпечує візуальне представлення складного об’єкта, розбиваючи його на складові частини. Різні блоки на діаграмах відповідають конкретним деталям кожного компонента, а вся модель організована таким чином, що можна побачити, як кожна частина впливає на ціле.

Кожна з наведених нижче окремих діаграм є декомпозицією блоку з більш загальної діаграми. Батьківська діаграма є більш детальною діаграмою, і кожна діаграма ілюструє внутрішню структуру блоку в межах батьківської діаграми. Дуги, що входять і виходять з блоку на діаграмі верхнього рівня, такі ж, як і дуги, що входять і виходять з діаграми нижчого рівня, оскільки блок і діаграма представляють ту саму частину системи.

Для того, щоб вказати положення будь-якої діаграми чи блоку в ієрархії, використовуються номери діаграм. Наприклад, A21 є діаграмою, що деталізує блок 1 на діаграмі A2.

Кожен блок на діаграмі має свій номер. Блок будь-якої діаграми може бути далі описаний діаграмою нижнього рівня, що, у свою чергу, може бути далі деталізована за допомогою необхідного числа діаграм. Таким чином, формується ієрархія діаграм.

Для більш детального опису роботи системи створюється функціональна декомпозиція.

Декомпозицією попередньої діаграми є діаграма A0 (рисунок 2.3).

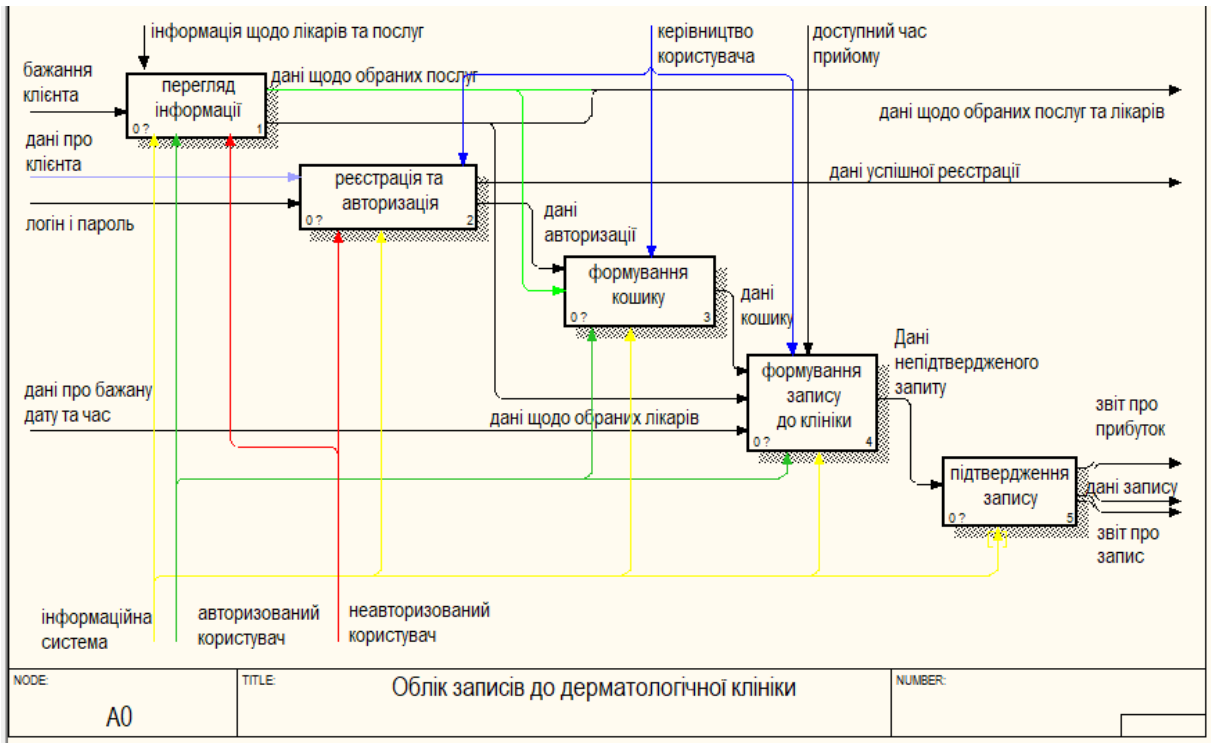


Рисунок 2.3 – Діаграма декомпозиції A0

Декомпозиція функціонального блоку «перегляд інформації» наведено на рисунку 2.4.

На вході є бажання клієнта, він переглядає інформацію щодо лікарів та обирає бажану категорію послуг і переглядає послуги цієї категорії. На виході є знання клієнта щодо обраних лікарів та знання клієнта щодо обраних послуг.

Декомпозиція функціонального блоку «реєстрація та авторизація» наведено на рисунку 2.5.

На вході є дані про клієнта, які знадобляться для реєстрації та логін і пароль. Клієнт реєструється або авторизується, введені дані перевіряються. Уразі якщо данні введено не вірно – виводиться сповіщення про помилку і дію потрібно повторити, у разі коли дані введено вірно – реєстрація чи авторизація пройшли успішно. На виході є дані авторизації або дані успішної реєстрації.

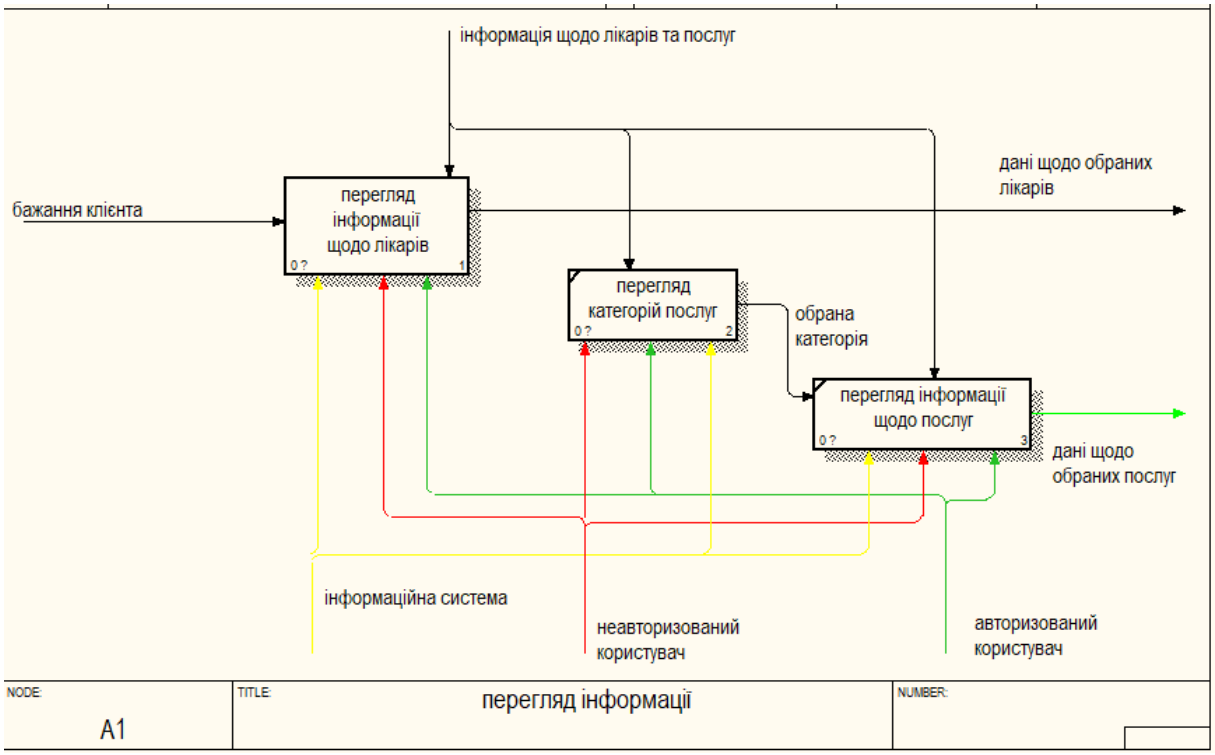


Рисунок 2.4 – Діаграма декомпозиції A1

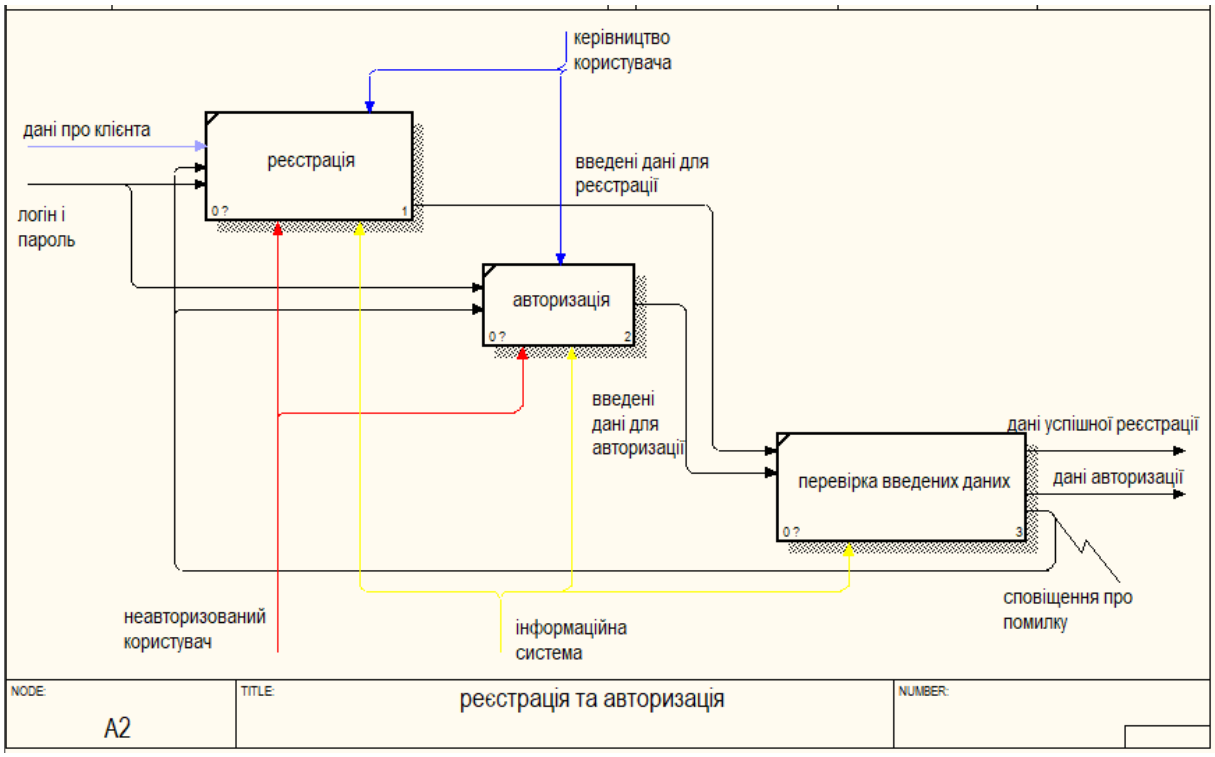


Рисунок 2.5 – Діаграма декомпозиції A2

Декомпозиція функціонального блоку «формування запису до клініки» наведено на рисунку 2.6.

На вході є інформація щодо конкретного користувача, дані щодо обраного лікаря, дані кошику (послуг, які було обрано), та дані про бажану дату та час. Відповідність даних перевіряється і у разі якщо все гаразд створюється запис зі статусом «Не підтверджено». На виході є дані непідтвердженого запиту на запис до дерматологічної клініки.

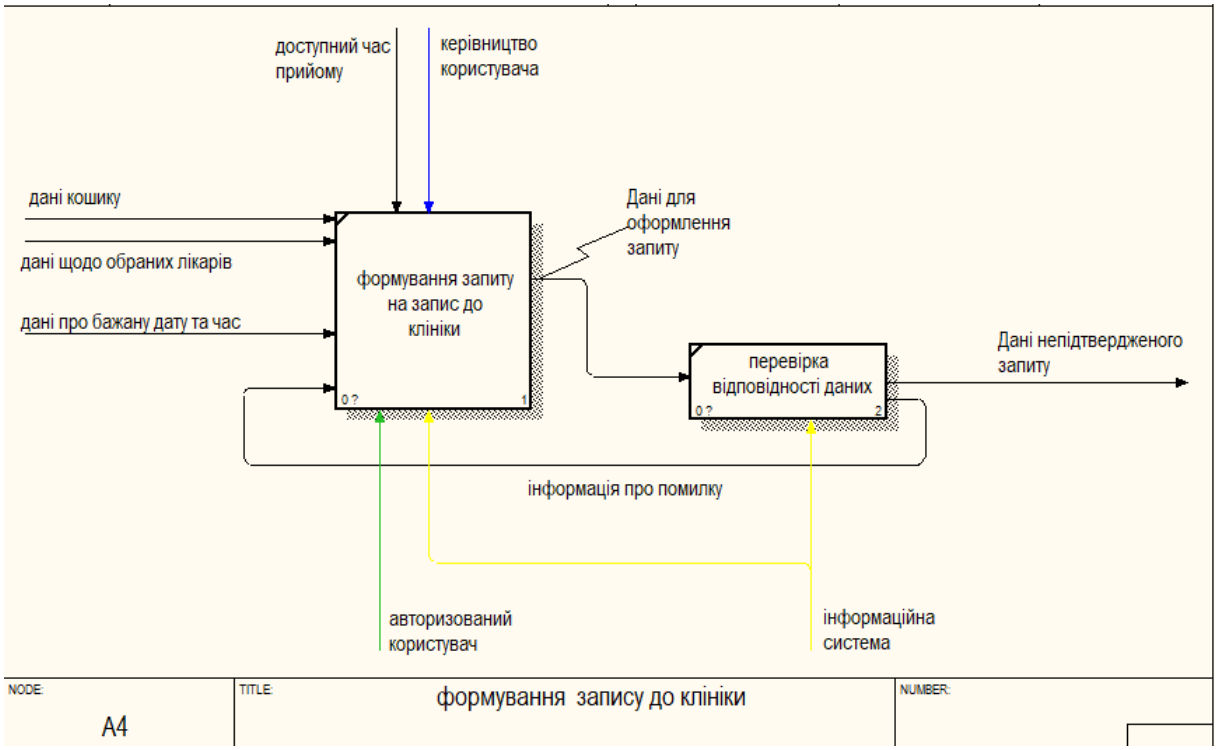


Рисунок 2.6 – Діаграма декомпозиції A4

При моделюванні системи важливо враховувати всі аспекти предметної області. Зі збільшенням рівня деталізації в описі системи стає більш ймовірним, що будуть виявлені недоліки, які не були помічені на початку проекту.

Діаграми було створено в середовищі ERWin Process modeler, яке є програмним інструментом для детального документування бізнес-процесів.

2.1.2 Опис функціональних вимог у вигляді DFD діаграми

Діаграми потоків даних DFD (Data Flow Diagram) є корисним доповненням до до IDEF0-діаграм для опису документообігу та обробки інформації. Вони дуже детально описують потоки даних, дозволяючи відстежувати, як відбувається обмін інформацією в системі між бізнес-процесами та між самою системою та зовнішнім середовищем.

Основні компоненти DFD-діаграм це зовнішні сутності, системи/підсистеми, процеси, накопичувачі даних, потоки даних.

Контекстна діаграма потоків даних задачі «Облік записів до дерматологічної клініки» зображена на рисунку 2.7.

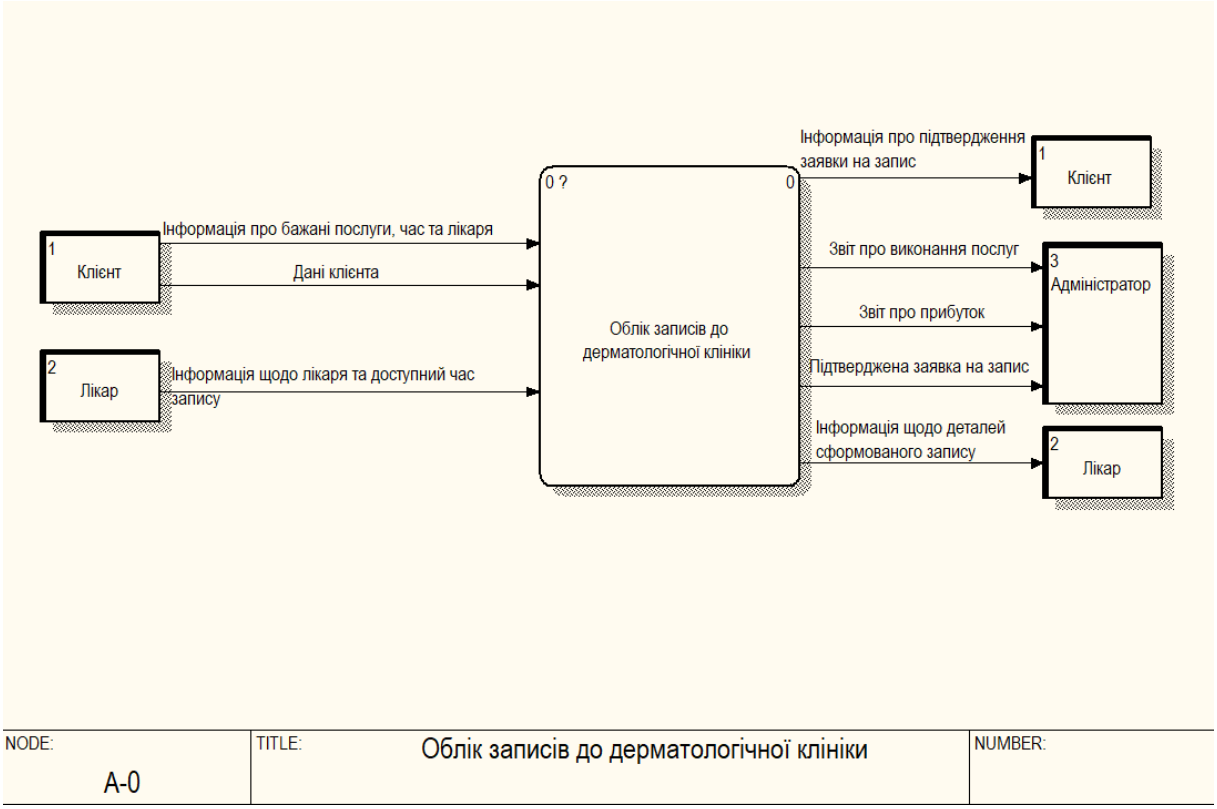


Рисунок 2.7 – Контекстна діаграма потоків даних задачі «Облік записів до дерматологічної клініки»

Контекстні діаграми є способом ілюстрації того, як різні частини інформації взаємодіють одна з одною.

Найпростіша контекстна діаграма – це структура у формі зірки, в середині якої розташований головний процес.

Така діаграма показує основний спосіб введення інформації в систему, а також способи її виведення. Інші контекстні діаграми можна побудувати на основі цієї, щоб проілюструвати, як різні частини інформації взаємодіють одна з одною.

Діаграма потоків даних задачі «Облік записів до дерматологічної клініки» зображена на рисунку 2.8.

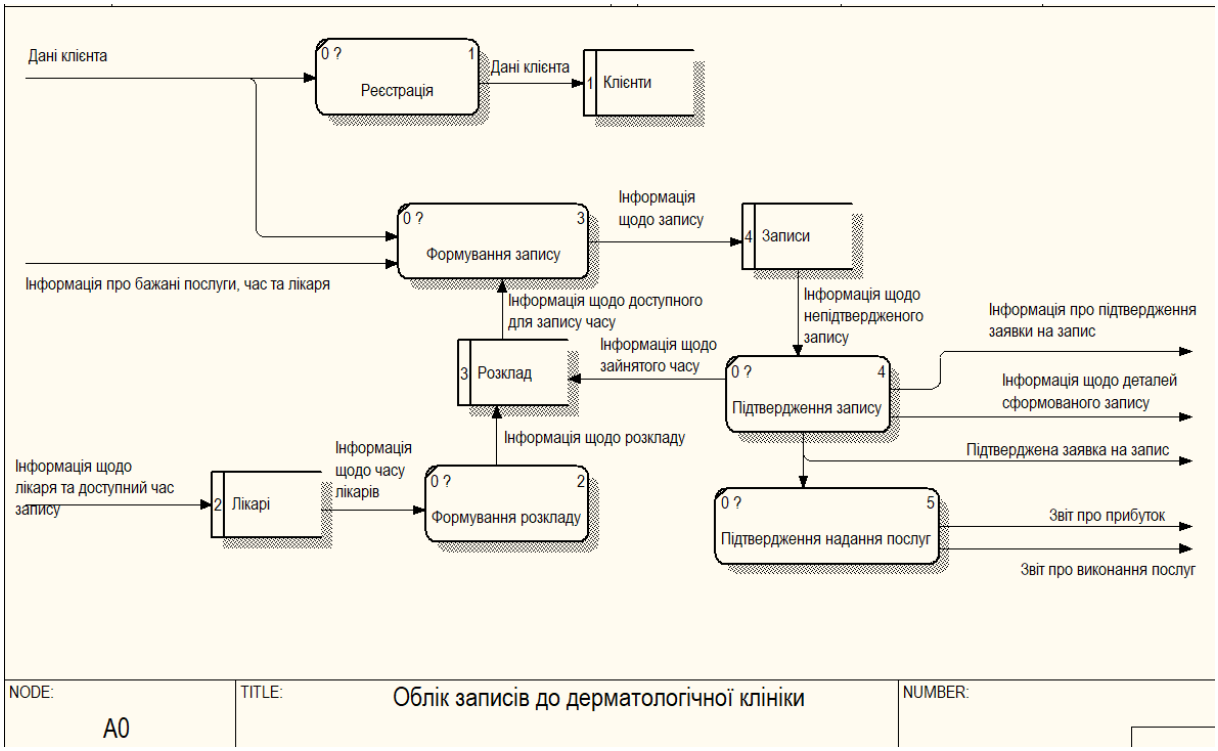


Рисунок 2.8 – Діаграма потоків даних задачі «Облік записів до дерматологічної клініки»

2.1.3 Опис функціональних вимог у вигляді UseCase діаграми

Діаграма прецедентів (UseCase діаграма) – це візуальне представлення того, як люди взаємодіють із системою, яке часто використовується як частина моделі варіантів використання. Він складається з набору акторів, прецедентів (варіантів використання), обмежених межами системи, асоціацій між акторами та прецедентами, відносин між прецедентами та зв’язків узагальнення між акторами. Діаграми прецедентів можуть бути корисними для розуміння того, як люди використовують систему, і для планування вдосконалень.

Діаграма прецедентів показує, як спроектована система, з багатьма об’єктами або акторами, які взаємодіють з нею та використовують варіанти використання для визначення послуг, які система надає. Варіант використання (use case) – це техніка моделювання вимог до програмного забезпечення або системи. Вона використовується для опису того, як користувачі будуть взаємодіяти з системою і як система повинна реагувати на їхні запити та дії.

Кожен варіант використання описує конкретний сценарій взаємодії між користувачем та системою, що включає в себе послідовність кроків, які користувач повинен здійснити, і як система повинна реагувати на кожен крок.

Такі описи варіантів використання можуть використовуватися для розробки вимог до програмного забезпечення, для тестування системи, а також для забезпечення зрозумілості між учасниками проекту щодо того, яким чином система має працювати в конкретних умовах.

Варіанти використання допомагають представити, чого конкретний актор хоче досягти за допомогою системи. Розуміючи випадки використання, ми можемо краще зрозуміти, що намагається зробити актор і як система може допомогти йому цього досягти.

У системі, що розроблюється, можна виділити три актори: Адміністратор, Неавторизований користувач та Авторизований користувач.

На рисунку 2.9 зображена діаграма прецедентів, яка відображає процеси, пов'язані з роботою дерматологічної клініки.

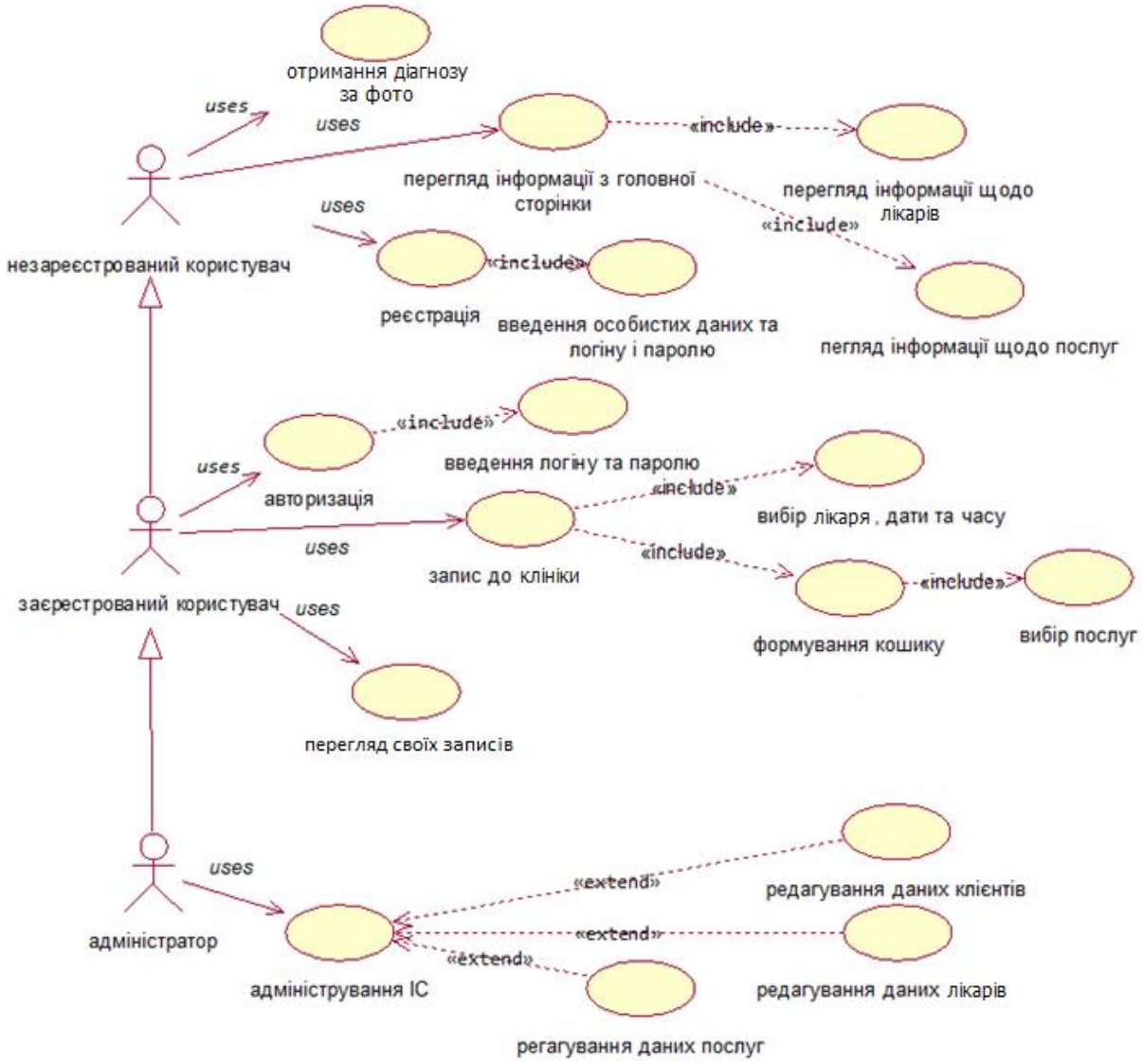


Рисунок 2.9 – Діаграма прецедентів

2.2 Моделювання бази даних системи

Моделювання даних – це спосіб представлення структури та поведінки бази даних у спосіб, який можна легко відобразити в будь-якій системі баз даних.

Моделі сутності та зв’язку (ERD) є поширеним засобом моделювання даних. З їх допомогою визначаються важливі для предметної області об'єкти (сутності), їх властивості (атрибути) і відносини один з одним (зв'язки). ERD часто використовуються для розробки реляційних баз даних.

Перший крок моделювання – виділення сутностей.

Кожна сутність у моделі повинна мати унікальний ідентифікатор, і кожен екземпляр сутності має бути чітко ідентифікований серед усіх інших екземплярів того самого типу сутності. Кожна сутність повинна мати деякі властивості, включаючи ім’я та інтерпретацію, яка завжди застосовується до того самого імені. Не можна однаково тлумачити різні імена, якщо вони не є псевдонімами. Кожна сутність може мати будь-яку кількість зв’язків з іншими сутностями в моделі.

Наступним кроком моделювання є ідентифікація зв'язків.

Зв'язок – це зв'язок між двома сутностями. Кожна сутність у зв’язку має набір пов’язаних екземплярів. Екземпляр дочірньої сутності може існувати, лише якщо існує батьківська сутність.

Можуть бути наведені назви зв’язку, що виражається граматичним оборотом дієслова і ставиться біля лінії зв’язку. Імена кожного зв’язку між двома заданими сутностями мають бути унікальними, але імена зв’язків у моделі не обов’язково мають бути унікальними. Ім’я відношення завжди формується з точки зору батька, тому речення можна сформувати шляхом поєднання імені батьківського об’єкта, імені відношення, виразу ступеня та імені нащадкового об’єкта.

Останнім кроком моделювання є ідентифікація атрибутів.

Атрибут представляє тип або характеристики властивостей, пов’язаних із набором реальних або абстрактних об’єктів (людей, місць, подій, станів, ідей, пар об’єктів тощо). Екземпляр атрибута визначається типом ознаки та значенням атрибута, яке називається значенням атрибута. У моделі ERD атрибути пов’язані з конкретними сутностями. Таким чином, сутність повинна мати єдине визначення значення для пов’язаного атрибута.

Атрибут може бути обов’язковим або необов’язковим (атрибут не може приймати невизначених значень – null values). Атрибут може бути або описовим, або входити до складу унікального ідентифікатора (первинного ключа).

У разі повної ідентифікації кожен екземпляр сутності цього типу ідентифікується своїми унікальними ключовими атрибутами, інакше в її ідентифікації також беруть участь атрибути іншої батьківської сутності.

Кожен атрибут ідентифікується унікальною назвою, яка виражається граматичним оборотом іменника, що описує характеристику, представлену атрибутом. Атрибути відображаються як список імен у середині пов’язаного блоку сутності, причому кожен атрибут займає окремий рядок. Атрибути, що визначають первинний ключ, розміщуються у верхній частині списку та розділяються знаком «#».

Для створення логічної моделі даних була використана програма ERWin Data Modeler. Логічна модель даних – це візуальне представлення структур даних, їхніх атрибутів і зв’язків у системі. Конструкція логічної моделі має бути незалежною від платформи та мови реалізації або способу використання даних у системі і не повинна прив’язуватися до конкретної СУБД. Логічна модель також повинна бути початковим прототипом для майбутньої бази даних.

Для моделювання використовується стандарт IDEF1X.

Для створення логічної моделі спочатку треба визначити сутності, які будуть у базі даних.

Були створені такі сутності:

- «User»: там зберігаються дані які потрібні для входу на сайт;
- «Role»: роль користувача у системі;
- «ServiceType»;
- «Service»: в ній є інформація про послугу та про те, до якого типу вона належить;
- «Employee»: в ній є інформація щодо лікарів;
- «Reception»;
- «Reception-Service»: в ній зіставляються номери записів та послуги.

На рисунку 2.10 зображена логічна модель бази даних.

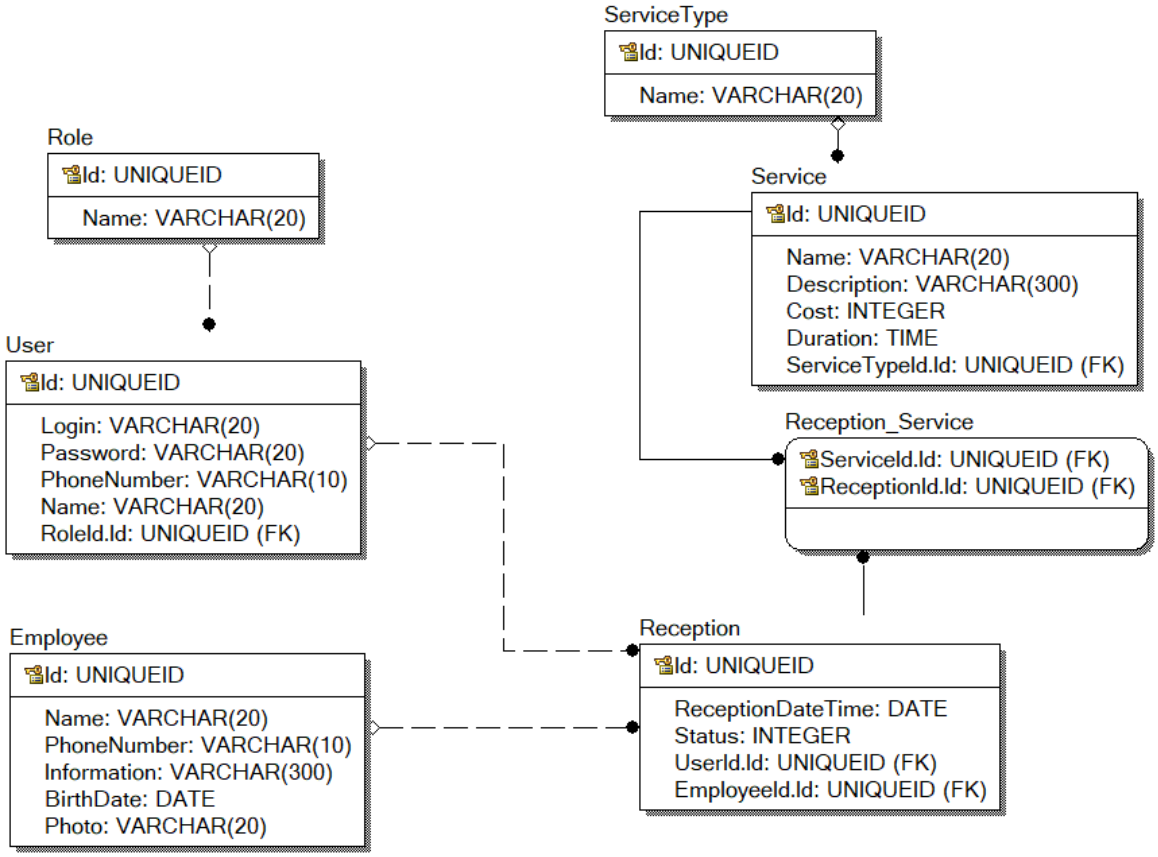


Рисунок 2.10 – Логічна модель бази даних

На основі логічної моделі створюється фізична модель. Ця модель містить інформацію про всі об'єкти бази даних. Модель залежить від конкретної реалізації СУБД. Тому одна і та ж логічна модель може мати різні фізичні моделі. Головною відмінністю від логічної моделі є зв’язок «багато до багатьох». Цей зв’язок дозволений, лише якщо існує третя сутність, яка зберігає первинні ключі двох сполучних сутностей.

На рисунку 2.11 зображена фізична модель бази даних.

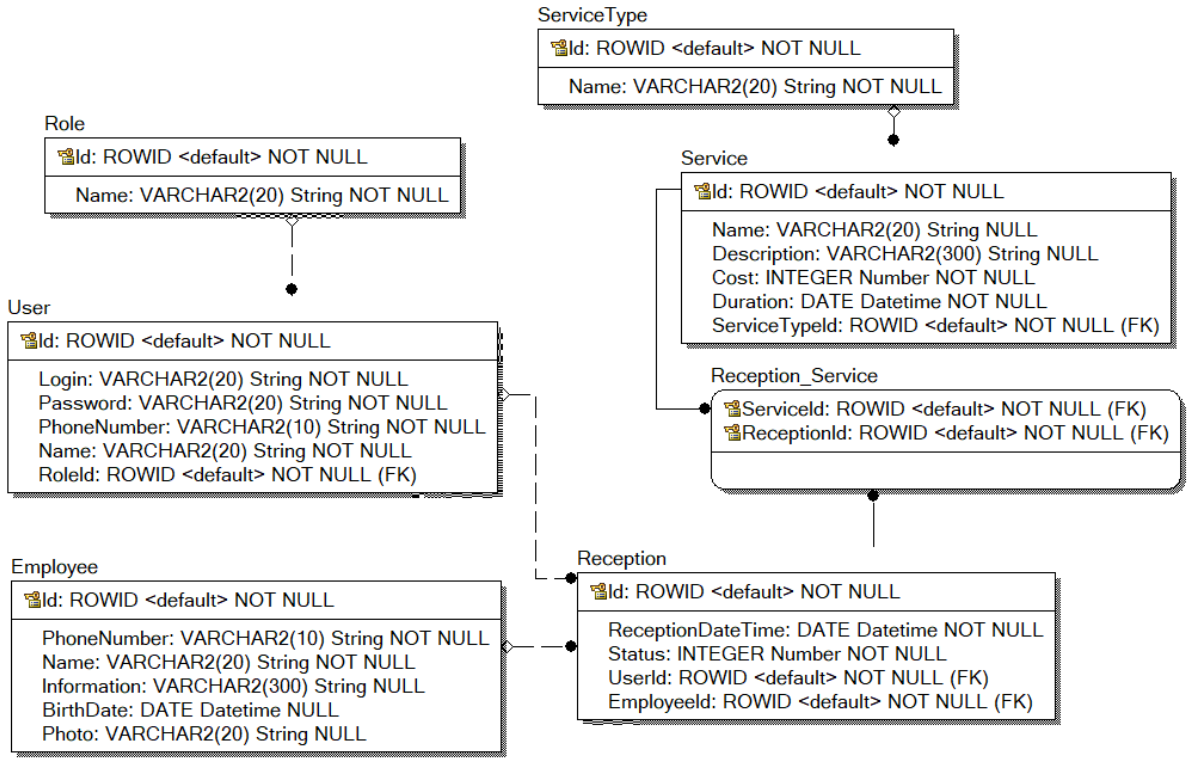


Рисунок 2.11 – Фізична модель бази даних

2.3 Опис функціональних вимог у вигляді UserStories

UserStory – це опис того, що клієнт хоче від продукту. Цей опис розбивається на менші, досяжні цілі. Це дозволяє розробляти та модифікувати продукт таким чином, щоб він був гнучким відповідно до мінливих потреб клієнта. Крім того, історії користувачів – це швидкий і простий спосіб задокументувати вимоги клієнтів.

UserStory найкраще створювати в межах концепції завершених послідовних подій, що описані приблизно так: початкові умови (контекст ситуації) коли (я роблю щось) тоді (відбувається така подія) коли (я роблю щось інше) тоді (відбувається інша подія).

Викладений у структурованому стислий опис проекту допоможе клієнту зосередитися на його найважливіших аспектах, а також відстежувати прогрес проекту в досягненні його цілей.

Опис усіх UserStories наведено у таблицях 2.1, 2.2, 2.3, 2.4, 2.5, 2.6 та 2.7.

Таблиця 2.1 – Опис UserStory «Перегляд інформації з головної сторінки»

Дійові особи	Клієнт, Система
Цілі	Клієнт: заходить на сайт та переглядає інформацію з головної сторінки; Система: надає запитувану інформацію.
Успішний сценарій: 1. Клієнт запускає додаток. Система відкриває сесію користувача, клієнт опиняється на головній сторінці. 2. Клієнт натискає на бажаний розділ головної сторінки (перегляд послуг або лікарів). 3. Система надає запитувану інформацію.	
Результат	Клієнт успішно ознайомився з інформацією з головної сторінки.
Розширення:	
*а	Немає доступу до бази даних (БД). Система видає повідомлення про відповідну помилку. Результат: клієнт не має доступу до списку послуг та лікарів.

Таблиця 2.2 – Опис UserStory «Отримання діагнозу за фото»

Дійові особи	Клієнт, Система
Цілі	Клієнт: завантажує фото новоутворення на шкірі; Система: видає припущення щодо діагнозу, чи є новоутворення доброякісним чи злоякісним.
Успішний сценарій: 1. Клієнт запускає додаток. Система відкриває сесію користувача, пропонує завантажити фото для видачі припущення щодо діагнозу. 2. Клієнт завантажує фото. 3. Система видає припущення щодо діагнозу, чи є новоутворення доброякісним чи злоякісним.	
Результат	Клієнт отримав припущення щодо діагнозу.
Розширення:	
1а	Вибрано пошкоджений файл з фото. Результат: припущення щодо діагнозу не надається, система видає відповідне повідомлення користувачу.

Таблиця 2.3 – Опис UserStory «Реєстрація»

Дійові особи	Клієнт, Система
Цілі	Клієнт: реєструється в системі; Система: додає дані користувача до бази даних і встановлює його права.

Продовження таблиці 2.3

Успішний сценарій: 1. Клієнт запускає додаток. Система відкриває сесію користувача, пропонує зареєструватися за допомогою вводу імені, номеру телефону, логіну, паролю та повтору паролю. 2. Клієнт вводить данні. 3. Система перевіряє чи пароль співпадає з повторним вводом паролю. 4. Система перевіряє унікальність логіну. 5. Система перевіряє валідність інших введених даних. 6. Система додає дані користувача до бази даних.	
Результат	Клієнт успішно зареєстрований у системі.
Розширення:	
*a	Немає доступу до БД. Система видає повідомлення про відповідну помилку. Результат: клієнт не може зареєструватися.
1a	Введені паролі не співпадають. Система видає повідомлення з відповідним текстом. Результат: клієнт не може зареєструватися.
1a	Введені паролі не співпадають. Система видає повідомлення з відповідним текстом. Результат: клієнт не може зареєструватися.
2a	Введений логін вже існує в системі. Система видає повідомлення з відповідним текстом. Результат: клієнт не може зареєструватися.
3a	Введенні дані не валідні. Система видає повідомлення з відповідним текстом. Результат: клієнт не може зареєструватися.

Таблиця 2.4 – Опис UserStory «Вхід»

Дійові особи	Клієнт, Система
Цілі	Клієнт: авторизуватися в системі і отримує доступ до списку своїх записів; Система: ідентифікувати клієнта і його права.
Успішний сценарій: 1. Клієнт відкриває форму азотизації. 2. Клієнт вводить логін і пароль. 3. Система перевіряє логін і пароль. 4. Система надає клієнтові доступ до його записів.	
Результат	Клієнт успішно авторизований і може працювати з системою.
Розширення:	
*а	Немає доступу до БД. Система видає повідомлення про відповідну помилку. Результат: клієнт не може увійти до свого облікового запису.
1а	Користувача з введеними логіном і паролем не знайдено. Результат: відмова в авторизації. Система видає повідомлення з відповідним текстом.

Таблиця 2.5 – Опис UserStory «Створення запису»

Дійові особи	Клієнт, Система
Цілі	Клієнт: створює заявку на запис до дерматологічної клініки; Система: формує запис та додає його до списку записів, які очікують підтвердження.

Продовження таблиці 2.5

Успішний сценарій: 1. Клієнт відкриває сторінку запису до клініки. Система пропонує обрати бажані послуги та лікаря. 2. Клієнт обирає бажані послуги та лікаря. 3. Система пропонує обрати бажаний час та дату серед доступних для даного лікаря. 4. Клієнт обирає бажані час та дату. 5. Система формує запис та зберігає його у базі даних.	
Результат	Клієнт успішно створив зв’язку на запис до дерматологічної клініки.
Розширення:	
*a	Немає доступу до БД. Система видає повідомлення про відповідну помилку. Результат: клієнт не може створив зв’язку на запис.
1a	Загальна тривалість обраних послуг перевищує 8 годин. Система видає повідомлення з відповідним текстом. Результат: створення запису неможливо.
2a	Обрано такий час, що час завершення надання послуг перевищує час завершення робочого дня. Система видає повідомлення з відповідним текстом. Результат: створення запису неможливо.

Таблиця 2.6 – Опис UserStory «Видалення запису»

Дійові особи	Клієнт, Система
Цілі	Клієнт: видаляє не підтверджений запис до дерматологічної клініки; Система: видаляє запис.

Продовження таблиці 2.6

Успішний сценарій: 1. Клієнт відкриває сторінку зі своїми записами. 2. Клієнт обирає не підтверджений запис і натискає кнопку «Delete». 3. Система видаляє запис.	
Результат	Клієнт успішно видалив не підтверджений запис до дерматологічної клініки.
Розширення:	
*a	Немає доступу до БД. Система видає повідомлення про відповідну помилку. Результат: клієнт не може змінити особисту інформацію.

Таблиця 2.7 – Опис UserStory «Редагування даних дерматологічної клініки»

Дійові особи	Користувач з правами адміністратора, Система
Цілі	Адміністратор: змінює дані щодо послуг та лікарів; Система: зберігає оновлені дані.
Успішний сценарій: 1. Адміністратор натискає кнопку «Edit». 2. Адміністратор додає, видаляє або редагує інформацію щодо послуг та лікарів. 3. Адміністратор натискає кнопку «Save». 4. Система перевіряє нові дані. 5. Система зберігає зміни до бази даних.	
Результат	Адміністратор успішно змінив бажані дані.

Продовження таблиці 2.7

Розширення:	
*a	Немає доступу до БД. Система видає повідомлення про відповідну помилку. Результат: адміністратор не може відредагувати інформацію.
1a	Введені дані не валідні. Система видає повідомлення з відповідним текстом. Результат: зміни не зберігаються.

2.4 Моделювання алгоритму аналізу фотографії

Аналіз фотографії для визначення характеру новоутворення на шкірі відбувається з використанням технологій штучного інтелекту. Для цього використовується модель машинного навчання, якак була навчена на попередньо обробленому наборі даних розрізняти меланоми та невуси за ознаками на знімках.

Класифікація зображень – це завдання в області штучного інтелекту, яке передбачає призначення однієї або кількох міток або категорій зображенню на основі його візуальних особливостей. Метою класифікації зображень є розробка алгоритмів або моделей, які можуть автоматично ідентифікувати та класифікувати зображення за заздалегідь визначеними категоріями. В цій роботі була навчена модель, що може класифікувати зображення новоутворень на шкірі на злоякісні та доброякісні.

В роботі було використано Microsoft.ML, що є це бібліотекою машинного навчання з відкритим кодом, розробленою Microsoft. Вона створениа як гнучка і масштабована структура, яку можна використовувати для створення широкого діапазону моделей машинного навчання, від простої лінійної регресії до складних моделей глибокого

навчання. Та SciSharp.TensorFlow.Redist, що є бібліотекою, яка надає розробникам .NET доступ до різноманітних функцій TensorFlow. ЇЇ підтримка широкого спектру моделей глибокого навчання, розширюваність і підтримка розподіленого навчання роблять її привабливим вибором для тих, хто працює в екосистемі .NET.

Для виконання класифікації зображень потрібно виконати наступні кроки:

– збір даних. Першим кроком у класифікації зображень є збір зображень, які потрібно класифікувати. Цей набір даних має бути різноманітним і репрезентативним для класів, що підлягають класифікації. Обробка зображень представляє собою форму обробки інформації, для якої вхідні дані представлені зображенням, наприклад, фотографіями або відеокадрами [4]. Для застосунка, що розроблюється, у якості даних для навчання було використано датасет International Skin Imaging Collaboration (ISIC) 2017 – колекція зображень уражень шкіри, що доступна для дослідницьких цілей у галузі дерматології та машинного навчання, він містить тисячі зображень різних типів уражень шкіри, які були анотовані дерматологами.

– попередня обробка даних: після того як набір даних зібрано, потрібно попередньо обробити зображення. Зібраний дата сет було перемішано щоб переконатися, що приклади не впорядковані певним чином, що може внести упередження в модель та розділено на набори для навчання, перевірки та тестування. Навчальний набір використовується для навчання моделі, набір перевірки використовується для налаштування гіперпараметрів, а набір для тестування використовується для оцінки продуктивності моделі.

– виділення ознак: наступним кроком є виділення ознак із зображень. Для цього було вказано пайплайн операцій для вилучення функцій і застосування алгоритму машинного навчання. Бібліотека Microsoft.ML підтримує чотири варіанти: ResnetV2101, InceptionV3,

MobilenetV2, ResnetV250, всі вони згорткові нейронні мережі, які зазвичай використовуються для завдання класифікації зображень.

Inception створено, щоб зменшити обчислювальний тягар глибоких нейронних мереж із одночасним досягненням найсучаснішої продуктивності. У міру заглиблення мережі обчислювальна ефективність також буде знижуватися, тому автори Inception були зацікавлені в пошуку рішення для розширення нейронних мереж без збільшення обчислювальних витрат.

Навідміну від Inception, що зосереджується на обчислювальних витратах, ResNet зосереджується на точності обчислень. Більш глибокі мережі не повинні працювати гірше, ніж менші, але на практиці більш глибокі мережі працюють гірше, ніж менші мережі, що спричинене проблемою оптимізації – чим глибша мережа, тим важче її оптимізувати.

Ідея MobileNet полягає в тому, щоб використовувати згортки, що розділяються по глибині, для побудови легших глибоких нейронних мереж. У звичайному згортковому шарі ядро або фільтр згортки застосовується до всіх каналів вхідного зображення шляхом зваженої суми вхідних пікселів за допомогою фільтра, а потім ковзає до наступних вхідних пікселів по зображеннях. MobileNet використовує цю згортку лише на першому рівні.

У за стосунку, що розроблюється, було обрано до використання ResnetV250 тому, що вона є найглибшою з чотирьох моделей, має 250 шарів і може досягти високої точності для широкого спектру завдань класифікації зображень, хоча це і вимагає значних обчислювальних ресурсів і часу на навчання.

– навчання: після виділення функцій можна використовувати їх для навчання моделі машинного навчання. Ця модель вчиться класифікувати зображення на основі виділених ознак. Для навчання моделі використовується метод Fit(), що викликається на пайплайні. Навчання

моделі триває деякий час, після чого модель може бути збережена для подальшого використання.

На рисунку 2.12 зображено знімок екрану під час навчання моделі.

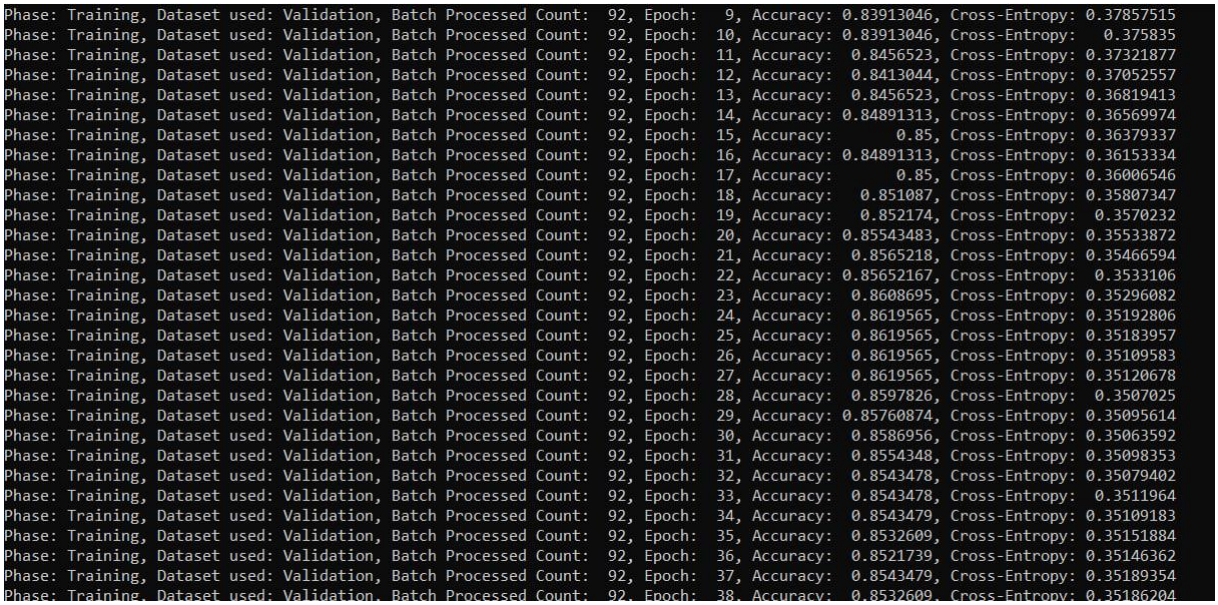


Рисунок 2.12 – Процес навчання моделі

– тестування: після навчання моделі можна перевірити її точність на окремому наборі зображень. Це допоможе оцінити, наскільки добре працює модель. Для цього можна викликати метод Evaluate з бібліотеки Microsoft.ML. В результаті будуть отримані метрики, такі як MicroAccuracy та MacroAccuracy, MicroAccuracy це звичайна точність – просто кількість правильних прогнозів, поділена на загальну кількість елементів. MacroAccuracy – це середня точність у класах для прогнозування.

На рисунку 2.13 зображено знімок екрану під час тестування моделі.

У ході тестування MacroAccuracy склала 0.8925551102204409.

```
Accuracy: 0.8925551102204409
Classifying single image
Image: 424.jpg | Actual Value: benign | Predicted Value: benign
Classifying multiple images
Image: 424.jpg | Actual Value: benign | Predicted Value: benign
Image: 1644.jpg | Actual Value: benign | Predicted Value: benign
Image: 1302.jpg | Actual Value: benign | Predicted Value: benign
Image: 1076.jpg | Actual Value: benign | Predicted Value: benign
Image: 1383.jpg | Actual Value: benign | Predicted Value: benign
Image: 256.jpg | Actual Value: benign | Predicted Value: benign
Image: 668.jpg | Actual Value: benign | Predicted Value: benign
Image: 22.jpg | Actual Value: benign | Predicted Value: malignant
Image: 1024.jpg | Actual Value: benign | Predicted Value: benign
Image: 1513.jpg | Actual Value: benign | Predicted Value: benign
Image: 1351.jpg | Actual Value: benign | Predicted Value: benign
Image: 885.jpg | Actual Value: benign | Predicted Value: benign
Image: 440.jpg | Actual Value: benign | Predicted Value: benign
Image: 1308.jpg | Actual Value: benign | Predicted Value: malignant
Image: 1048.jpg | Actual Value: benign | Predicted Value: benign
Image: 756.jpg | Actual Value: benign | Predicted Value: benign
Image: 1676.jpg | Actual Value: benign | Predicted Value: malignant
Image: 811.jpg | Actual Value: benign | Predicted Value: benign
Image: 1061.jpg | Actual Value: benign | Predicted Value: benign
Image: 716.jpg | Actual Value: benign | Predicted Value: benign
Image: 839.jpg | Actual Value: benign | Predicted Value: benign
Image: 1030.jpg | Actual Value: malignant | Predicted Value: malignant
Image: 643.jpg | Actual Value: benign | Predicted Value: benign
Image: 996.jpg | Actual Value: benign | Predicted Value: benign
Image: 298.jpg | Actual Value: benign | Predicted Value: benign
Image: 783.jpg | Actual Value: benign | Predicted Value: benign
```

Рисунок 2.13 – Оцінка точності моделі

– розгортання: коли модель навчена та протестована, її можна розгортати для використання в програмах. Це може включати її інтеграцію у більшу програмну систему або надання її як окремого інструменту.

Після того, як модель була навчена і в ході тестування була отримана задовільна точність, вона зберігається в папці «workspace» і в подальшому використовується в застосунку для класифікації зображень новоутворень на злоякісні та доброякісні.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Архітектура застосунку

Додаток було розроблено з використанням «Onion» архітектури.

Архітектура Onion ділить програму на рівні. Найнижчий рівень є ядром архітектури, і він розташований у центрі. Усі інші рівні залежать від цього основного рівня. Другий рівень залежить від першого, третій – від другого і так далі. Тобто рівні шаруються навколо ядра. Кількість рівнів може бути різною, але в центрі завжди знаходиться модель домену (Domain Model). Рівні можуть бути замінені без необхідності щось змінювати в попередньому шарі.

Архітектура відповідає принципам SOLID, які підкреслюють модульність, відокремлення завдань і згуртованість. Це дозволяє повторно використовувати код і кращу організацію, що призводить до більш надійного та ефективного програмного забезпечення:

– (Single responsibility principle): Кожен об'єкт має виконувати лише один обов'язок;

– (Open/closed principle): програмні сутності повинні бути відкритими для розширення, але закритими для змін;

– (Liskov substitution principle): об'єкти в програмі можуть бути замінениими їх нащадками без зміни коду програми;

– (Interface segregation principle): багато спеціалізованих інтерфейсів краще за один універсальний;

– (Dependency inversion principle): залежності всередині системи будуються на основі абстракцій, що не повинні залежати від деталей.

Розроблений додаток складається з чотирьох проектів:

DermatologyClinic.Core є основою програми. Він містить моделі та інтерфейси, які можна використовувати на інших рівнях програми.

На рисунку 3.1 зображено проект DermatologyClinic.Core.

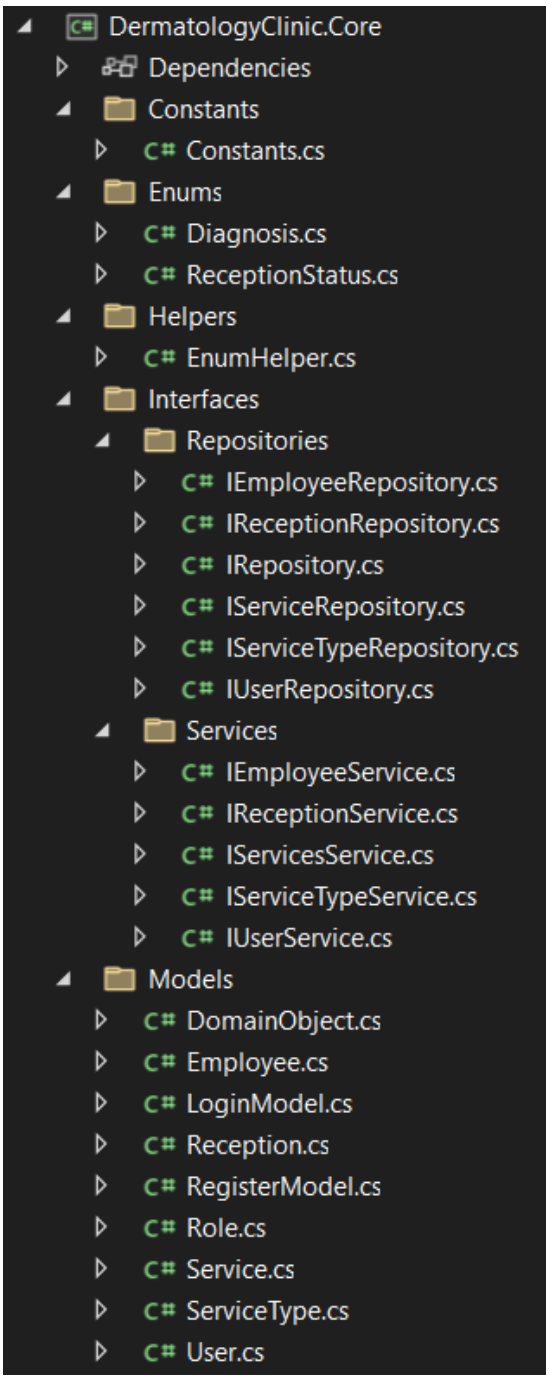


Рисунок 3.1 – Проект DermatologyClinic.Core

DermatologyClinic.DAL (Data access layer) – є наступним рівнем у програмі, і він залежить лише від першого рівня. Тут реалізований доступ до бази даних, також цей проект містить клас контексту даних і реалізацію репозиторіїв, які використовуються для роботи з базою даних.

На рисунку 3.2 зображено проект DermatologyClinic.DAL.

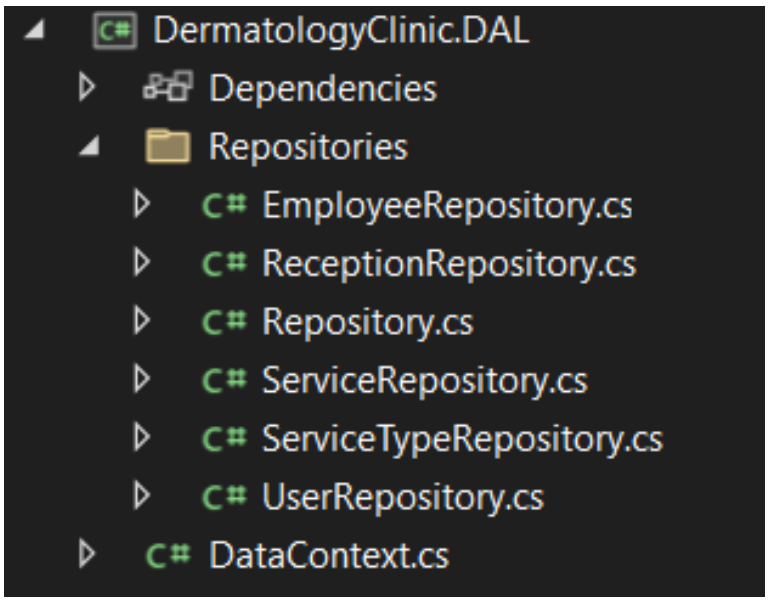


Рисунок 3.2 – Проект DermatologyClinic.DAL

DermatologyClinic.BLL (Business logic layer) – є наступним рівнем програми, і він залежить від рівнів DermatologyClinic.DAL і DermatologyClinic.Core. Тут реалізована бізнес-логіка програми, яка використовує сервіси, які мають методи виконання різних функцій. Ці сервіси можуть використовувати методи сховища для доступу до даних. Використання репозиторіїв реалізовано через інтерфейси.

На рисунку 3.3 зображено проект DermatologyClinic.BLL.

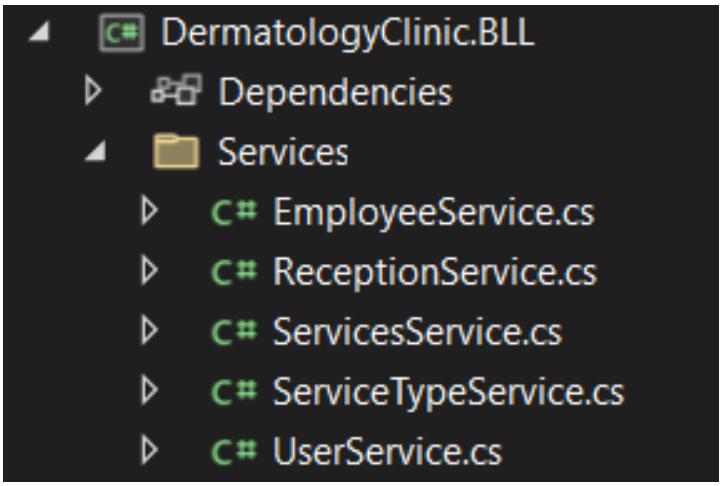


Рисунок 3.3 – Проект DermatologyClinic.BLL

DermatologyClinic – це останній рівень застосунку і основний проект, він залежить від DermatologyClinic.BLL та DermatologyClinic.Core. Він відповідає за представлення даних користувачам у зручний спосіб і визначення точки входу для інших проектів. Проект розроблено з використанням шаблону Model-View-Controller (MVC).

MVC передбачає поділ системи на три взаємопов'язані частини: моделі даних (Model), сторінки (інтерфейс користувача) (View) та контролери (Controller).

На рисунку 3.4 зображено проект DermatologyClinic.

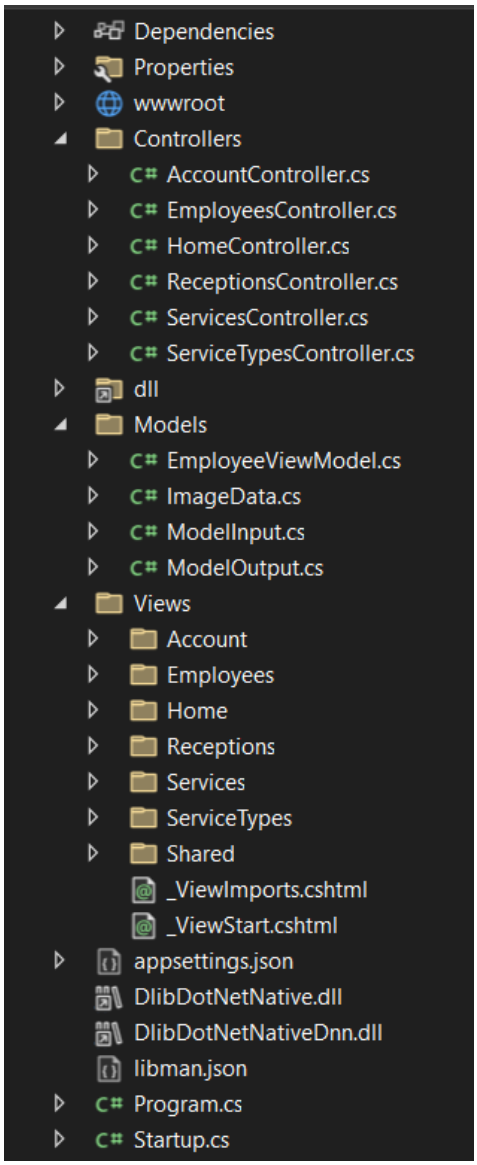


Рисунок 3.4 – Проект DermatologyClinic

3.2 Програмна реалізація

3.2.1 Реалізація алгоритму класифікації зображень

Для аналізу фото використовується бібліотека Microsoft.ML, та виконуються такі кроки: спочатку зображення, на яких буде навчатись модель збираються в об’єкт IDataView, після чого перемішуються та діляться на тестовий та валідаційний набори. Для цього було використано інструменти Microsoft.ML: mlContext.Data.ShuffleRows(imageData) для перемішування зображень та mlContext.Data.TrainTestSplit(data: preProcessedData, testFraction: 0.3) для його поділу.

Далі потрібно налаштувати пайплайн, тут можна вказати архітектуру, кількість епох та інші параметри.

На рисунку 3.5 зображено код для налаштування Pipeline.

```
var classifierOptions = new ImageClassificationTrainer.Options()
{
    FeatureColumnName = "Image",
    LabelColumnName = "LabelAsKey",
    ValidationSet = validationSet,
    Arch = ImageClassificationTrainer.Architecture.ResnetV250,
    MetricsCallback = (metrics) => Console.WriteLine(metrics),
    TestOnTrainSet = false,
    ReuseTrainSetBottleneckCachedValues = true,
    ReuseValidationSetBottleneckCachedValues = true,
    Epoch = 1000,
    WorkspacePath = workspaceRelativePath
};

var trainingPipeline = mlContext.MulticlassClassification.Trainers.ImageClassification(classifierOptions)
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));
```

Рисунок 3.5 – Налаштування Pipeline

Щоб розпочати навчання моделі, викликається метод Fit(): var trainedModel = trainingPipeline.Fit(trainSet);.

Після тренування моделі за допомогою методу Fit(), можна використати модель для передбачення результатів на нових даних за допомогою методу Predict().

Коли навчання завршено, можна оцінити результат навчання моделі, Для цього використовується метод Evaluate з бібліотеки Microsoft.ML,

var metrics = mlContext.MulticlassClassification.Evaluate (predictions, labelColumnName: "LabelAsKey");.

Метод Evaluate() в Microsoft.ML використовується для оцінки продуктивності моделі машинного навчання на тестових даних. Цей метод дозволяє визначити, наскільки точно модель передбачає правильні відповіді на нових даних. Метод Evaluate() приймає на вхід об'єкт типу IDataView, що містить тестові дані, та повертає об'єкт типу IDictionary<string, IDataView>, який містить метрики, що характеризують продуктивність моделі на тестових даних. Такі метрики можуть включати значення середнього квадратичного відхилення (Mean Squared Error), коефіцієнт детермінації (R-Squared), точність (Accuracy), тощо. Оцінка продуктивності моделі на тестових даних дозволяє визначити, наскільки добре модель передбачає правильні відповіді на нових даних, і може використовуватися для налаштування моделі та визначення її параметрів.

Коли модель навчена та має достатню точність – то її можна зберегти для подальшого використання методами CreatePredictionEngine та predictionEngine.Predict(image).

На рисунку 3.6 зображено приклад використання моделі для класифікації зображення.

```
1 reference
private static ModelOutput ClassifyImage(MLContext mlContext, byte[] data, ITransformer trainedModel)
{
    PredictionEngine<ModelInput, ModelOutput> predictionEngine = mlContext.Model.CreatePredictionEngine<ModelInput, ModelOutput>(trainedModel);
    ModelInput image = new() { Image = data };

    return predictionEngine.Predict(image);
}
```

Рисунок 3.6 – Використання моделі для класифікації зображення

На рисунку 3.7 зображена діаграма структури програмного коду для класифікації зображення.

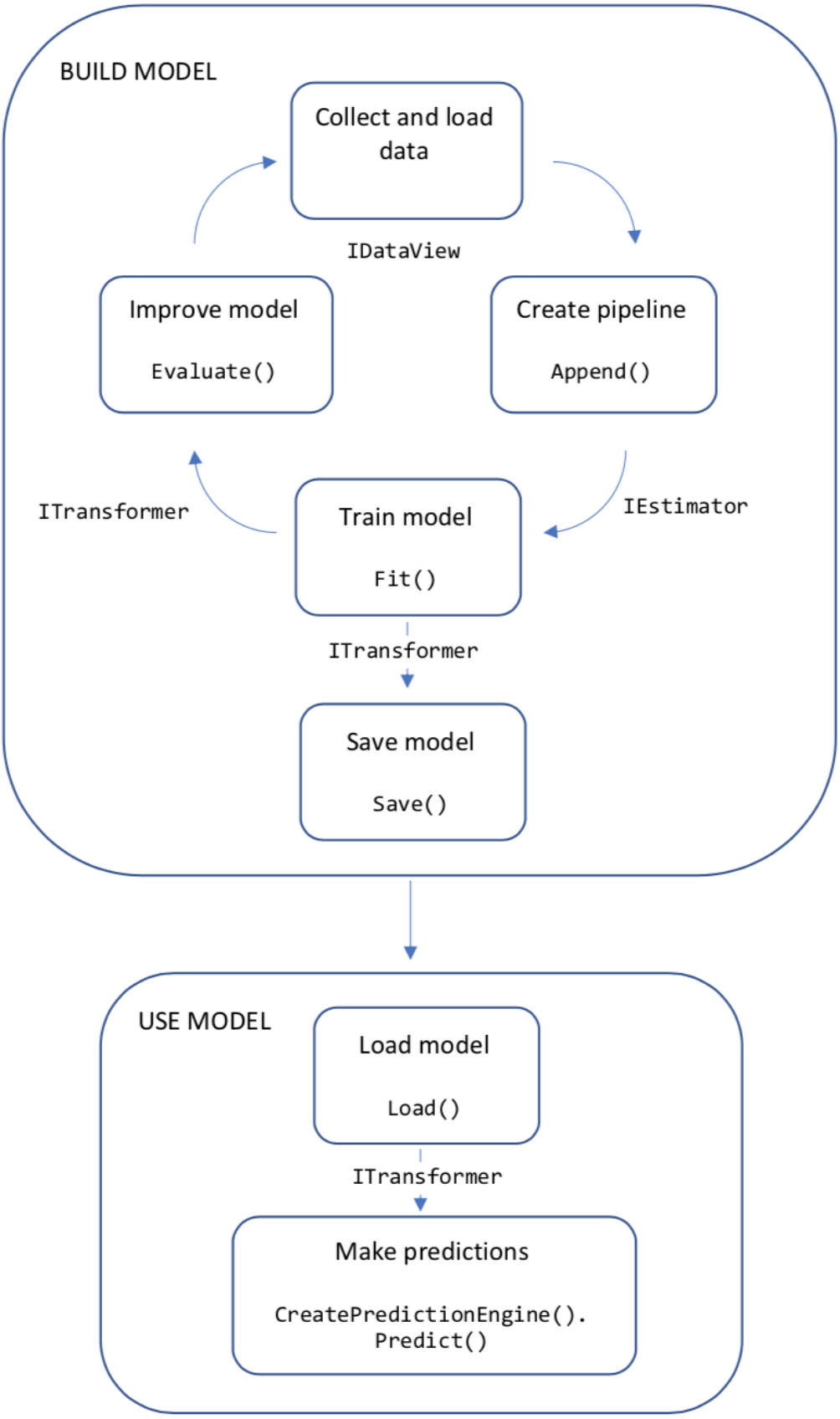


Рисунок 3.7 – Діаграма структури програмного коду для класифікації зображення

3.2.2 База даних

Усі моделі в базі даних є нащадками класу DomainObject. Цей клас має поле з ідентифікатором типу GUID (Globally Unique Identifier). Для бази даних були створені наступні моделі: User, Role, Employee, ServiceType, Service, Reception.

При створенні таких моделей використовувались атрибути, наприклад:

– [Display(Name = "Name")] – визначає напис для відображення;

– [Required(ErrorMessage = "Provide the phone number")] – означає що таке поле обов’язкове, у разі його не заповнення буде показано відповідне повідомлення;

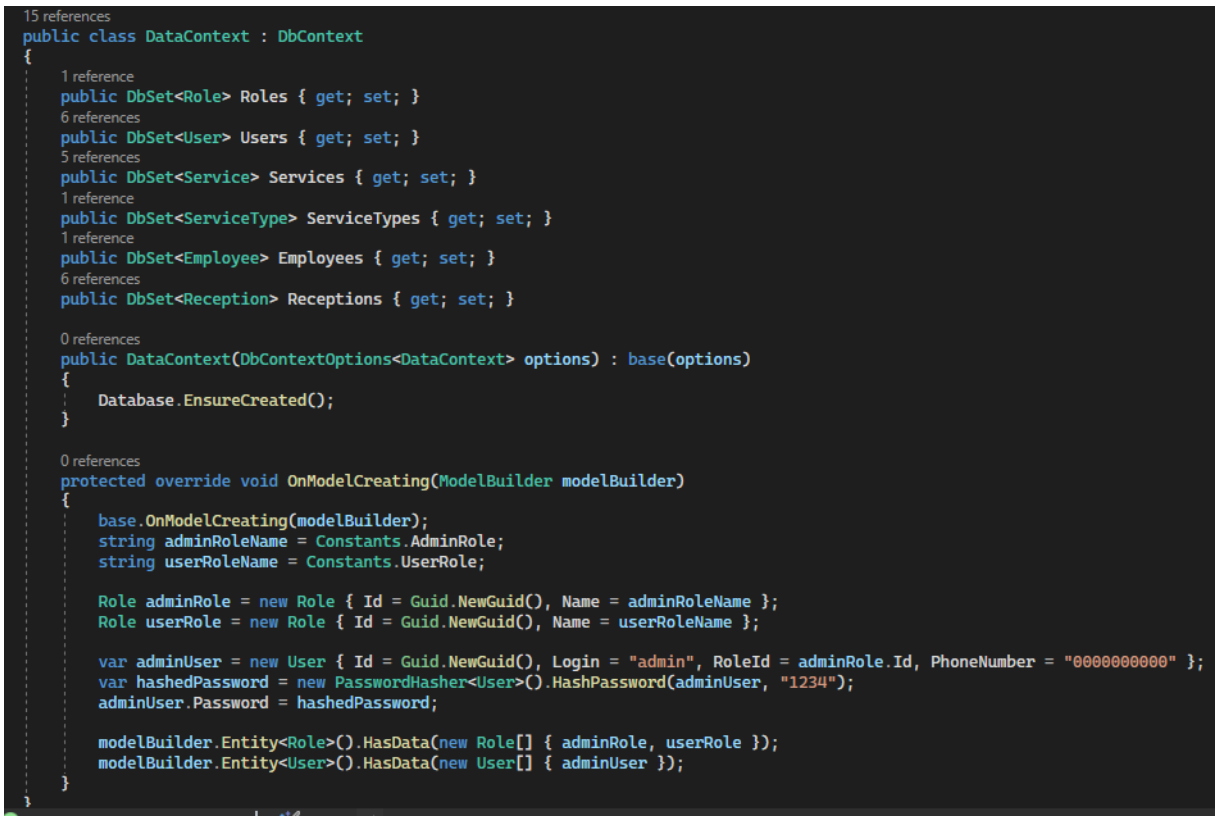
– [StringLength(10, MinimumLength = 10, ErrorMessage = " Phone number should contain 10 numbers ")] – визначає запитувану довжину рядка.

Також, було створено дві моделі для представлень: LoginModel та RegisterModel.

Для реалізації роботи з базою даних було створено клас DataContext (рисунок 3.8). В цьому класі було визначено DbSet-ти. Якщо база даних не створена, то вона створиться автоматично за допомогою команди Database.EnsureCreated(); в конструкторі. В методі OnModelCreating, при створенні бази даних, до неї відразу додаються дві ролі та користувач-адміністратор. Також, при створенні бази даних можна одразу створити користувача – адміністратора.

Оскільки паролі не повинні зберігатися в базі даних явно, пароль користувача хешується та зберігається в базі даних. Потім хешований пароль перетворюється на хеш за допомогою методу HashPassword із бібліотеки Identity.Core.

DataContext додається в класі Startup з рядком підключення з файлу appsettings.json. Код DataContext зображено на рисунку 3.8.



```
15 references
public class DataContext : DbContext
{
    1 reference
    public DbSet<Role> Roles { get; set; }
    6 references
    public DbSet<User> Users { get; set; }
    5 references
    public DbSet<Service> Services { get; set; }
    1 reference
    public DbSet<ServiceType> ServiceTypes { get; set; }
    1 reference
    public DbSet<Employee> Employees { get; set; }
    6 references
    public DbSet<Reception> Receptions { get; set; }

    0 references
    public DataContext(DbContextOptions<DataContext> options) : base(options)
    {
        Database.EnsureCreated();
    }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        string adminRoleName = Constants.AdminRole;
        string userRoleName = Constants.UserRole;

        Role adminRole = new Role { Id = Guid.NewGuid(), Name = adminRoleName };
        Role userRole = new Role { Id = Guid.NewGuid(), Name = userRoleName };

        var adminUser = new User { Id = Guid.NewGuid(), Login = "admin", RoleId = adminRole.Id, PhoneNumber = "0000000000" };
        var hashedPassword = new PasswordHasher<User>().HashPassword(adminUser, "1234");
        adminUser.Password = hashedPassword;

        modelBuilder.Entity<Role>().HasData(new Role[] { adminRole, userRole });
        modelBuilder.Entity<User>().HasData(new User[] { adminUser });
    }
}
```

Рисунок 3.8 – Клас DataContext

3.2.3 Репозиторії

Було створено один загальний репозиторій, який у якості «T» приймає будь який клас наслідник від DomainObject. У ньому були реалізовані основні CRUD операції.

Було створено один загальний репозиторій, який приймає будь-який нащадковий клас із DomainObject як "T". У ньому реалізовані базові операції Create-Read-Update-Delete (CRUD).

На рисунку 3.9 зображено клас Repository<T>.

Даний клас реалізовує відповідний інтерфейс IRepository<T> (рисунок 3.10).

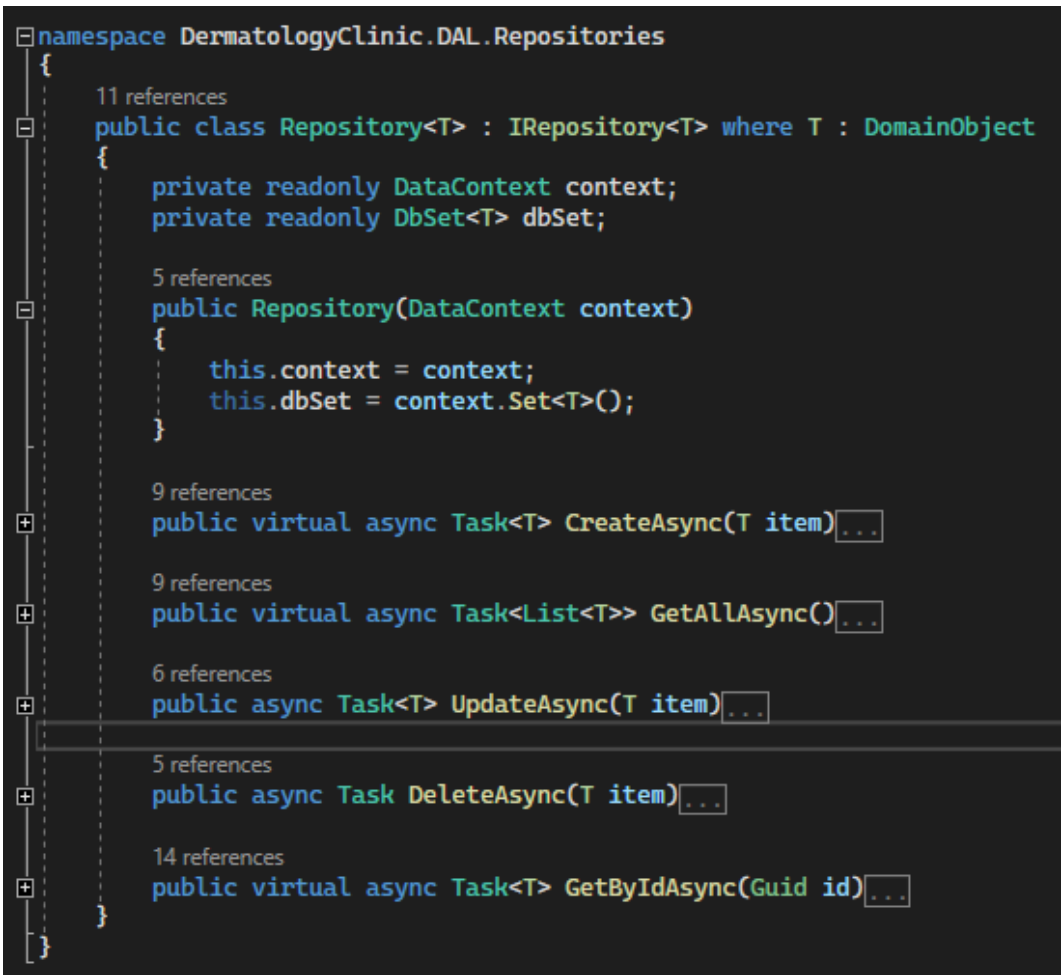


Рисунок 3.9 – Клас Repository<T>

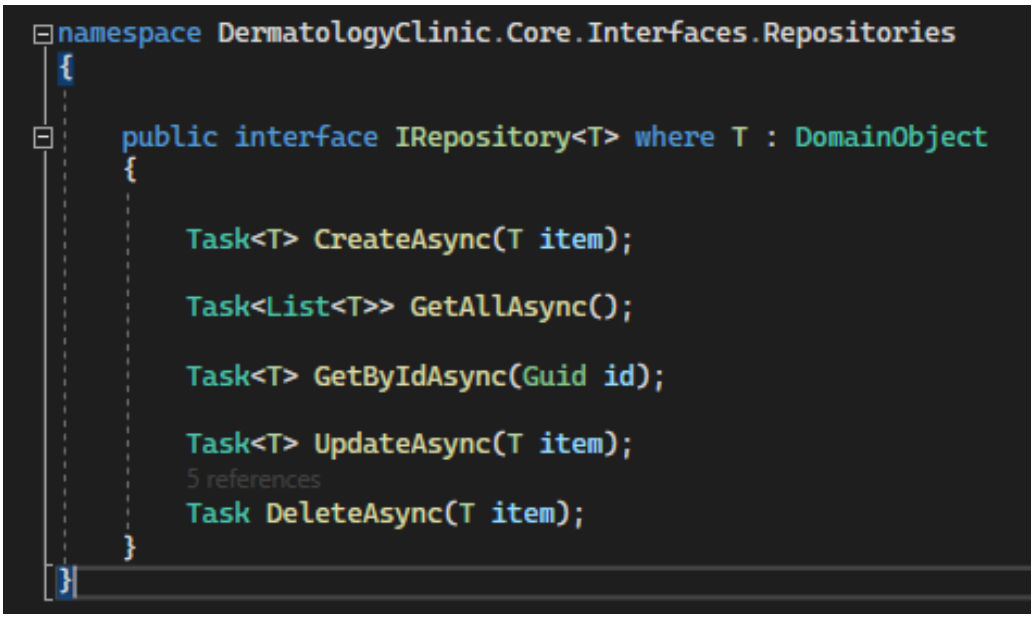


Рисунок 3.10 – Інтерфейс IRepository<T>

Програма також реалізувала репозиторії для певного типу, такі як EmployeeRepository, ReceptionRepository, ServiceRepository, ServiceTypeRepository та UserRepository. Усі ці репозиторії є нащадками класу Repository, але заміть «T» підставляється відповідна модель. У такому класі може бути реалізовано перевизначення батьківського методу, якщо потрібна певна нестандартна обробка даних.

Працюючи з таким репозиторієм можна викликати або батьківський метод з відповідною моделлю, або його override метод. Також, усі вони є реалізаціями відповідних інтерфейсів.

3.2.4 Сервіси

В програмі було реалізовано такі сервіси:

EmployeeService – це клас, у якому реалізовані необхідні методи для підтримки контролера. Інтерфейс IEmployeeRepository передається конструктору, і методи цього інтерфейсу будуть використовуватися при реалізації методів сервісу. Цей клас є реалізацією інтерфейсу IEmployeeService. На рисунку 3.11 зображено інтерфейс IEmployeeService.

```
namespace DermatologyClinic.Core.Interfaces.Services
{

    public interface IEmployeeService
    {

        Task<Employee> CreateAsync(Employee employee);

        Task<List<Employee>> GetAllAsync();

        Task<Employee> GetByIdAsync(Guid id);

        Task<Employee> UpdateAsync(Employee employee);

        Task DeleteAsync(Guid id);
    }
}
```

Рисунок 3.11 – Інтерфейс IEmployeeService

UserService – цей клас реалізує інтерфейс IUserService, який надає методи для керування користувачами. Інтерфейси IUserRepository та IHttpContextAccessor передаються конструктору, і методи цих інтерфейсів будуть використовуватися під час реалізації методів сервісу.

На рисунку 3.12 зображено інтерфейс IUserService.

```
namespace DermatologyClinic.Core.Interfaces.Services
{
    8 references
    public interface IUserService
    {
        2 references
        Task LogInAsync(LoginModel loginModel);
        2 references
        Task LogOutAsync();
        2 references
        Task RegisterAsync(RegisterModel registerModel);
        3 references
        Guid GetCurrentUserId();
        2 references
        Task<List<User>> GetAllClientsAsync();
        2 references
        Task<User> GetByIdAsync(Guid id);
    }
}
```

Рисунок 3.12 – Інтерфейс IUserService

ServiceTypeService – цей клас надає методи для роботи з типами послуг. Інтерфейси IServiceRepository та IServiceTypeRepository передаються конструктору, а методи цих інтерфейсів використовуються під час реалізації методів сервісу.

На рисунку 3.13 зображено інтерфейс IServiceTypeService.

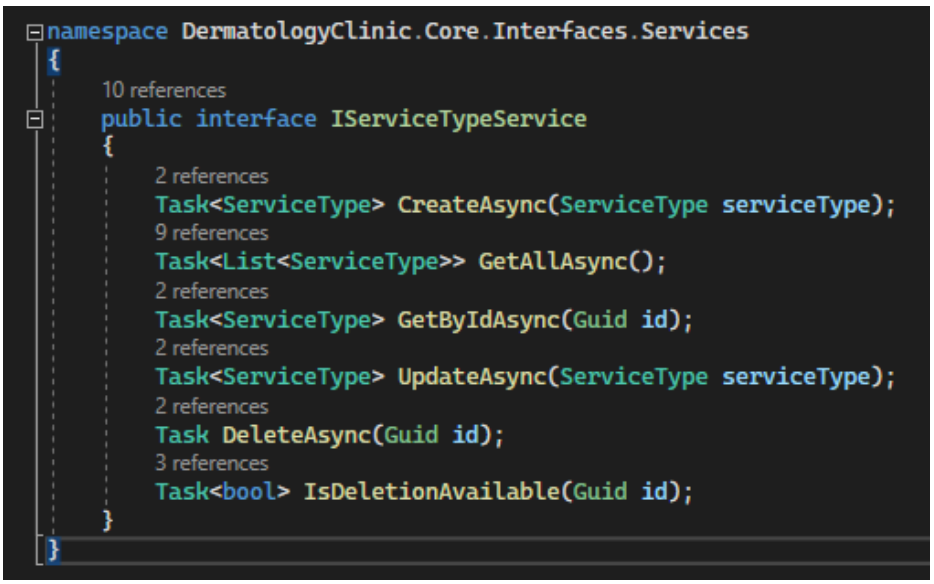


Рисунок 3.13 – Інтерфейс IServiceTypeService

ServicesService – цей клас забезпечує реалізацію інтерфейсу IServicesService, який можна використовувати для доступу до послуг, які надає програма. Він налаштований за допомогою об’єкта IServiceRepository, який забезпечує доступ до методів сервісу.

На рисунку 3.14 зображено інтерфейс IServicesService.

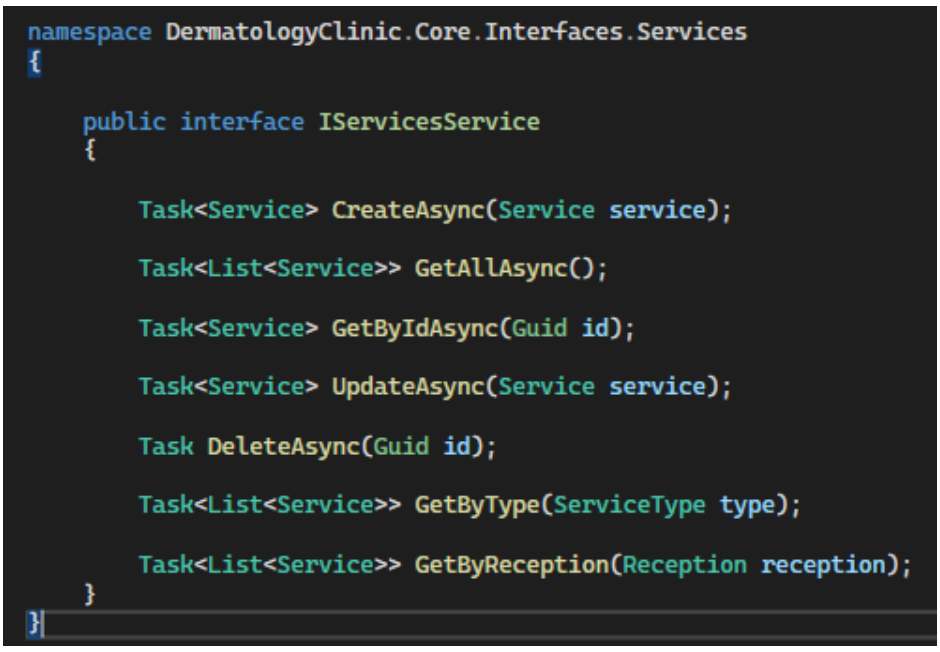


Рисунок 3.14 – Інтерфейс IServicesService

ReceptionService – тут знаходиться основна логіка оброки записів до клініки. Інтерфейси IReceptionRepository, IServicesService, IUserService та IEmployeeService передаються конструктору, а методи цих інтерфейсів використовуватимуться під час реалізації методів сервісу.

Тут міститься список усіх годин, доступних для запису (з 9:00 до 19:00). Метод GetAvailableTimeAsync об’єкта Appointment визначає список вільних годин для запису на певну дату до конкретного лікаря з урахуванням загальної тривалості вибраних послуг, закінчення робочого дня та вже наявних записів до цього лікаря. на цей день. Наприклад, якщо загальна тривалість вибраних послуг становитиме п’ять години, а обраний лікар має запис на прийом з 11:00 до 13:00, то вільні години для прийому будуть лише 13:00, 13:30 та 14:00.

Цей клас є реалізацією відповідного інтерфейсу IReceptionService.

Інтерфейс IReceptionService зображено на рисунку 3.15.

```
namespace DermatologyClinic.Core.Interfaces.Services
{

    public interface IReceptionService
    {

        Task<Reception> CreateAsync(Reception reception);

        Task<List<Reception>> GetAllCompletedAsync();

        Task<List<Reception>> GetCompletedByUserAsync(Guid userId);

        Task<Reception> GetByIdAsync(Guid id);

        Task<Reception> UpdateAsync(Reception reception);

        Task<Reception> ConfirmAsync(Reception reception);

        Task DeleteAsync(Guid id);

        Task<Reception> AddServiceAsync(Guid serviceId);

        Task<Reception> RemoveServiceAsync(Guid serviceId);

        Task<Reception> GetCurrentReceptionAsync();
        2 references
        Task<List<string>> GetAvailableTimeAsync(DateTime date, TimeSpan duration, Guid employeeId);
    }
}
```

Рисунок 3.15 – Інтерфейс IReceptionService

3.2.5 Контролери

Основний проект програми створено за допомогою шаблону Model View Controller (MVC). Представленнями керують відповідні контролери.

Контролер отримує дані, що ввів користувач, обробляє їх і відправляє результат обробки, наприклад, у вигляді подання (view).

Контролери містять методи (action methods), які обробляють запити для певної URL-адреси. Можна використовувати відповідні атрибути, такі як [HttpGet], [HttpPost], [HttpDelete] або [HttpPut], щоб визначити, які методи представляють методи контролера, що обробляють запити для цієї URL-адреси.

Методи контролера або навіть увесь контролер можуть мати атрибут Authorize, що означає, що лише авторизовані користувачі можуть отримати доступ до дії. Також можна передати назву ролі в атрибут Authorize, і тоді лише користувачі цієї ролі матимуть доступ. Ця функція надається бібліотекою AspNetCore.Authorization.

Програмний код усіх контролерів наведено у додатку Б.

HomeController – відповідає за головну сторінку застосунку. Тут міститься метод Index, він повертає ViewResult View(employees), де employees, це колекція лікарів, які контролер отримує з асинхронного методу this. employeesService.GetAllAsync() за допомогою IEmployeesService. контролер отримує список усіх лікарів та передає їх представленню за допомогою ViewBag.

Також в цьому контролері міститься функціонал для припущення діагнозу за фото. Обране фото користувач завантажує у відповідне поле, і далі, за допомогою методу PredictDiagnosis, відбувається аналіз новоутворення на фотографії з використанням технологій штучного інтелекту.

За допомогою методів бібліотеки Microsoft.ML відбувається навчання моделі класифікувати новоутворення на шкірі на злоякісні та

доброякісні, після чого, за допомогою метода CreatePredictionEngine видається припущення щодо діагнозу.

Контролер HomeController зображено на рисунку 3.16.

```
namespace DermatologyClinic.Controllers
{
    2 references
    public class HomeController : Controller
    {
        private readonly IServicesService servicesService;
        private readonly IServiceTypeService serviceTypeService;
        private readonly IEmployeeService employeeService;
        private readonly IHostingEnvironment env;

        0 references
        public HomeController(IServicesService servicesService,
            IServiceTypeService serviceTypeService,
            IEmployeeService employeeService,
            IHostingEnvironment env)...

        1 reference
        public async Task<IActionResult> Index(string error)...

        [HttpPost]
        0 references
        public async Task<ActionResult> Img(IFormFile imageFile)...

        1 reference
        private Diagnosis PredictDiagnosis(byte[] data)...

        1 reference
        private static ModelOutput ClassifyImage(MLContext mlContext, byte[] data, ITransformer trainedModel)...

        1 reference
        private static IEnumerable<ImageData> LoadImagesFromDirectory(string folder, bool useFolderNameAsLabel = true)...
    }
}
```

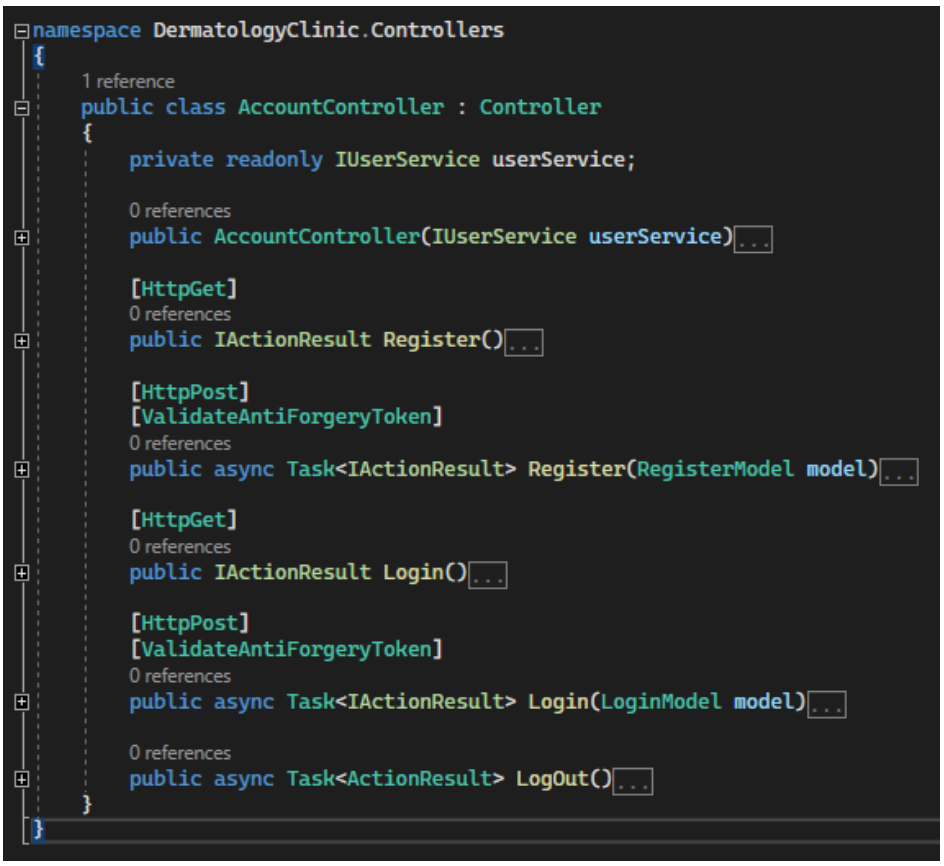
Рисунок 3.16 – Контролер HomeController

AccountController призначений для керування обліковими записами користувачів. Він містить методи, які дозволяють користувачам входити, реєструватися та виходити з системи. Ці завдання виконуються за допомогою методів об’єкта IUserService, який передається конструктору.

Контролер AccountController зображено на рисунку 3.17.

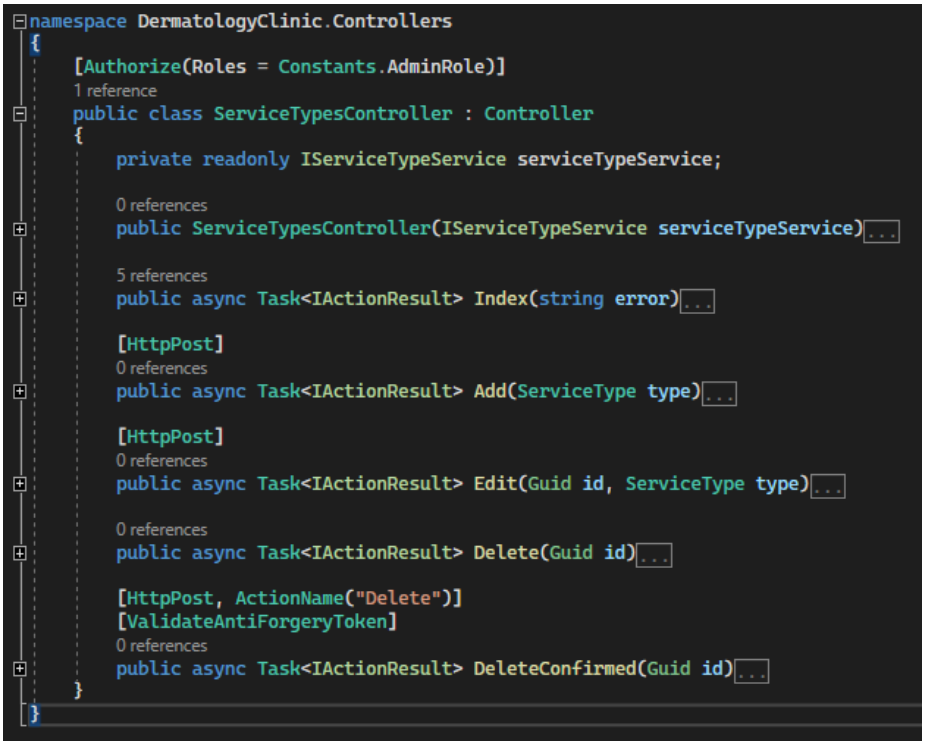
ServiceTypesController – тут виконуються CRUD операції з типами послуг, контролер використовує методи інтерфейсу IServiceTypeService для виконання цих функцій.

Контролер ServiceTypesController зображено на рисунку 3.18.



```
namespace DermatologyClinic.Controllers
{
    1 reference
    public class AccountController : Controller
    {
        private readonly IUserService userService;

        0 references
        public AccountController(IUserService userService)...

        [HttpGet]
        0 references
        public IActionResult Register()...

        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Register(RegisterModel model)...

        [HttpGet]
        0 references
        public IActionResult Login()...

        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Login(LoginModel model)...

        0 references
        public async Task<ActionResult> LogOut()...
    }
}
```

Рисунок 3.17 – Контролер AccountController



```
namespace DermatologyClinic.Controllers
{
    [Authorize(Roles = Constants.AdminRole)]
    1 reference
    public class ServiceTypesController : Controller
    {
        private readonly IServiceTypeService serviceTypeService;

        0 references
        public ServiceTypesController(IServiceTypeService serviceTypeService)...

        5 references
        public async Task<IActionResult> Index(string error)...

        [HttpPost]
        0 references
        public async Task<IActionResult> Add(ServiceType type)...

        [HttpPost]
        0 references
        public async Task<IActionResult> Edit(Guid id, ServiceType type)...

        0 references
        public async Task<IActionResult> Delete(Guid id)...

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> DeleteConfirmed(Guid id)...
    }
}
```

Рисунок 3.18 – Контролер ServiceTypesController

ServicesController відповідає за взаємодію із послугами. Йому надаються інтерфейси IServicesService та IServiceTypeService, які надають методи, за допомогою яких контролер може отримати всі послуги та типи послуг, щоб відобразити їх у представленнях, і виконувати певні дії, такі як створення, редагування, видалення або перегляд деталей про послугу.

Можна використовувати властивість ModelState.IsValid, щоб визначити, чи дійсні дані, введені в презентацію. Якщо ні, то подальші дії не відбуватимуться, а на представленні з’явиться відповідне повідомлення.

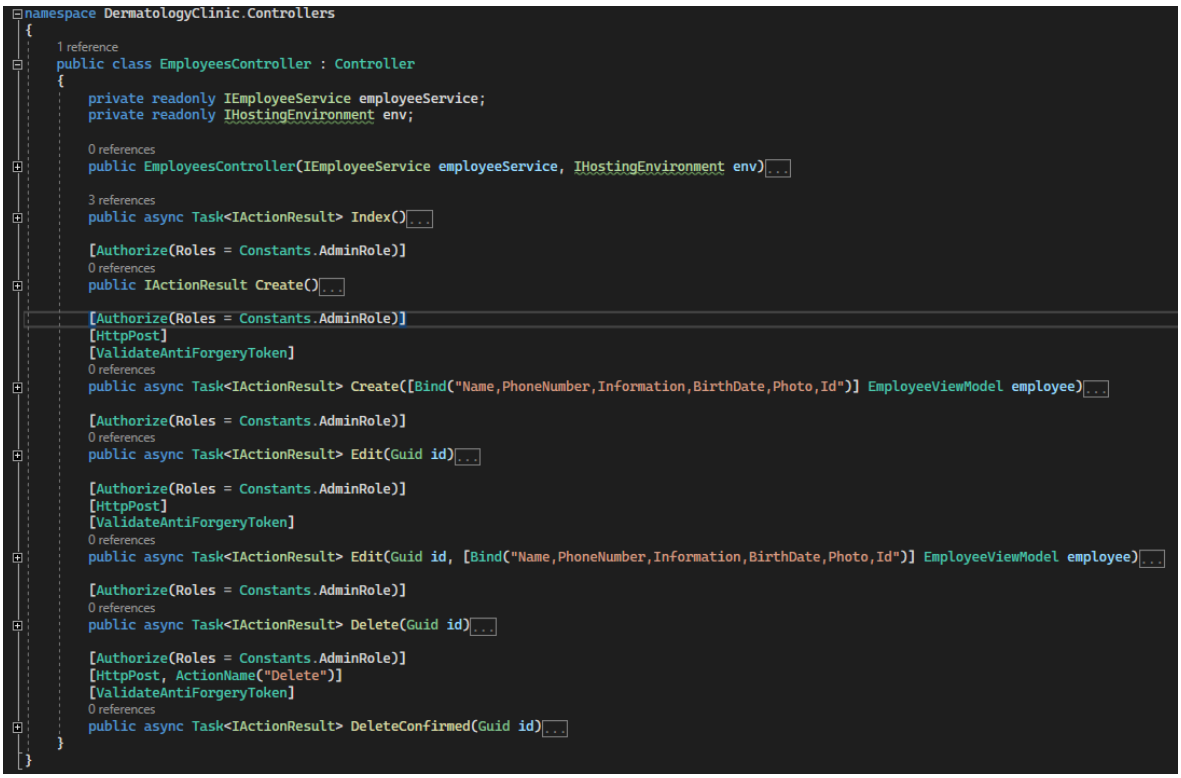
Контролер ServicesController зображено на рисунку 3.19.

```
namespace DermatologyClinic.Controllers
{
    1 reference
    public class ServicesController : Controller
    {
        private readonly IServicesService servicesService;
        private readonly IServiceTypeService serviceTypeService;

        0 references
        public ServicesController(IServicesService servicesService, IServiceTypeService serviceTypeService)...

        3 references
        public async Task<IActionResult> Index()...

        0 references
        public async Task<IActionResult> Details(Guid id)...

        [Authorize(Roles = Constants.AdminRole)]
        0 references
        public async Task<IActionResult> Create()...

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Create(int duration, Service service)...

        [Authorize(Roles = Constants.AdminRole)]
        0 references
        public async Task<IActionResult> Edit(Guid id)...

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Edit(Guid id, int duration, [Bind("Name,Description,Cost,Duration,ServiceTypeId,Id")] Service service)...

        [Authorize(Roles = Constants.AdminRole)]
        0 references
        public async Task<IActionResult> Delete(Guid id)...

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> DeleteConfirmed(Guid id)...
    }
}
```

Рисунок 3.19 – Контролер ServicesController

EmployeesController обробляє операції CRUD (створення, читання, оновлення та видалення) з лікарями. Для цього він використовує методи IEmployeeService.

Контролер EmployeesController зображено на рисунку 3.20.



```
namespace DermatologyClinic.Controllers
{
    1 reference
    public class EmployeesController : Controller
    {
        private readonly IEmployeeService employeeService;
        private readonly IHostingEnvironment env;

        0 references
        public EmployeesController(IEmployeeService employeeService, IHostingEnvironment env)...

        3 references
        public async Task<IActionResult> Index()...

        [Authorize(Roles = Constants.AdminRole)]
        0 references
        public IActionResult Create()...

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Create([Bind("Name,PhoneNumber,Information,BirthDate,Photo,Id")] EmployeeViewModel employee)...

        [Authorize(Roles = Constants.AdminRole)]
        0 references
        public async Task<IActionResult> Edit(Guid id)...

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Edit(Guid id, [Bind("Name,PhoneNumber,Information,BirthDate,Photo,Id")] EmployeeViewModel employee)...

        [Authorize(Roles = Constants.AdminRole)]
        0 references
        public async Task<IActionResult> Delete(Guid id)...

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> DeleteConfirmed(Guid id)...
    }
}
```

Рисунок 3.20 – Контролер EmployeesController

ReceptionsController здійснює запис на прийом до дерматологічної клініки. В конструктор контролера передаються такі сервіси: IReceptionService, IEmployeeService, IUserService, IServicesService, та IServiceTypeService. Вони всі потрібні для методів цього контролера, адже тут знаходиться основний, з точки зору бізнес логіки, метод: створення нового запису до клініки.

Щоб створити запис, за допомогою ViewBag на подання передається дата, список послуг та лікарів. Список вільних годин для запису буде оновлено автоматично на основі наданої інформації. Якщо не вибрано жодної послуги, з’явиться повідомлення про помилку. Також, тут є метод Index, який відкриває сторінку зі списком усіх записів, з цієї сторінки користувач може видалити непідтверджений запис методом Delete, Адміністратор може видалити його або підтвердити, після чого запис змінює статус на «Confirmed».

Контролер ReceptionsController зображено на рисунку 3.21.

```
public class ReceptionsController : Controller
{
    private readonly IReceptionService receptionService;
    private readonly IEmployeeService employeeService;
    private readonly IUserService userService;
    private readonly IServicesService servicesService;
    private readonly IServiceTypeService serviceTypeService;

    0 references
    public ReceptionsController(IReceptionService receptionService,
        IEmployeeService employeeService,
        IUserService userService,
        IServicesService servicesService,
        IServiceTypeService serviceTypeService)...

    [Authorize]
    5 references
    public async Task<IActionResult> Index()...

    [Authorize]
    1 reference
    public async Task<IActionResult> AddService(Guid serviceId)...

    [Authorize]
    0 references
    public async Task<IActionResult> RemoveService(Guid serviceId)...

    [Authorize]
    0 references
    public async Task<IActionResult> SelectEmployee(Guid employeeId)...

    [Authorize]
    0 references
    public async Task<IActionResult> SelectDate(DateTime date)...

    [Authorize]
    0 references
    public async Task<IActionResult> SelectTime(DateTime time)...

    [Authorize]
    5 references
    public async Task<IActionResult> Create()...

    [Authorize]
    [HttpPost]
    [ValidateAntiForgeryToken]
    5 references
    public async Task<IActionResult> Create(Guid userId)...

    [Authorize]
    0 references
    public async Task<IActionResult> Confirm(Guid id, string error)...

    [Authorize]
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public async Task<IActionResult> ConfirmationConfirmed(Guid id)...

    [Authorize]
    0 references
    public async Task<IActionResult> Delete(Guid id)...

    [Authorize]
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    0 references
    public async Task<IActionResult> DeleteConfirmed(Guid id)...

    7 references
    private async Task SetViewBagAsync(Reception reception)...
}
```

Рисунок 3.21 – Контролер ReceptionsController

3.2.6 Представлення

Представлення відповідають за зовнішній вигляд веб-сайту та написані за допомогою Razor. Синтаксис Razor дозволяє писати код C# безпосередньо в розмітку за допомогою символу "@".

Представлення впорядковано в папки відповідно до їхніх контролерів, а також є спільна папка Shared, яка містить основний файл розмітки

Файл _Layout.cshtml є головним, в ньому знаходиться тег html, а всі інші представлення вбудовуються у цю розмітку за допомогою команди @RenderBody(). Саме в цьому файлі відбувається завантаження усіх скриптів та стилів. У цьому файлі визначено верхній і нижній колонтитули, які будуть видимі на всіх сторінках програми.

Також, тут використовується partial: <partial name="_LoginPartial" />. На цьому місці буде вставлена розмітка відповідного файлу. В ньому визначається чи користувач вже авторизований і в залежності від цього вгорі сторінки показується або кнопки входу та реєстрації, або ім’я користувача та вихід. Таке визначення реалізовано за допомогою «if (User.Identity.IsAuthenticated)». Подібні умови для відображення для певних кнопок та полів також є в інших представленнях.

За допомогою «asp-controller="Home" asp-action="Index"» можна вказати, що клік по такому посиланні викличе метод Index з контролера HomeController. На представленнях слово «Controller» можна опустити під час написання назви контролера, достатньо буде написати «Home».

Для створення нового запису можна використати: «<form asp-action="Create">», кнопка підтвердження даної форми викличе метод Create відповідного контролера, він тут не вказаний явно, тому буде викликано відповідний згідно іменування. Також, у цей метод, в якості параметрів, буде передано значення полів даної форми.

В застосунку використовується ще один спосіб звернення до методу контролера: кнопці задається значення onclick, яке дорівнює, наприклад, "location.href='@Url.Action("Index", "Employees")'", це викличе метод Index контролера Employees.

Для створення інтерфейсу використовувався Bootstrap, а також оригінальні стилі. Оригінальні стилі знаходяться у файлі site.css та підключаються до представлень за допомогою «<link rel="stylesheet" href="~/css/site.css" />». Для іконок було використано шрифт «Awesome».

Для застосування стилів у розмітці додаються класи для тегу, приклад червоної кнопки з іконкою видалення:

<a asp-action="Delete" class="btn btn-outline-danger">Delete.

Також, були написані скрипти на мові JavaScript для деяких анімацій, таких як інтерактивний фон на сторінці логіну, автоматичні скроли та інше. Такі скрипти зберігається в розмітці або в папці "js" та підключається за допомогою «<script src="~/js/site.js" asp-append-version="true"></script>».

Приклад використання скрипта: <a onclick="toggle('@item.Id')">.

Структура представлень зображена на рисунку 3.22.

У класі «Startup» зазначено, що при старті програми буде викликано метод Index із контролера «Home». У результаті буде відкрито представлення «Index.cshtml» із папки «Home» та передано йому певну модель даних. У розмітці, за допомогою «foreach», буде створено певні теги з відповідними написами для кожного елементу колекції «ViewBag.Employees», яка була отримана з контролера. Таким чином у розмітці будуть сформовані стилізовані теги для кожного лікаря.

Також, на головній сторінці є інша інформація щодо додатку та поле для завантаження фото, щоб отримати діагноз про новоутворення на шкірі.

У результаті така сторінка буде виглядати так: (рисунок. 3.23).

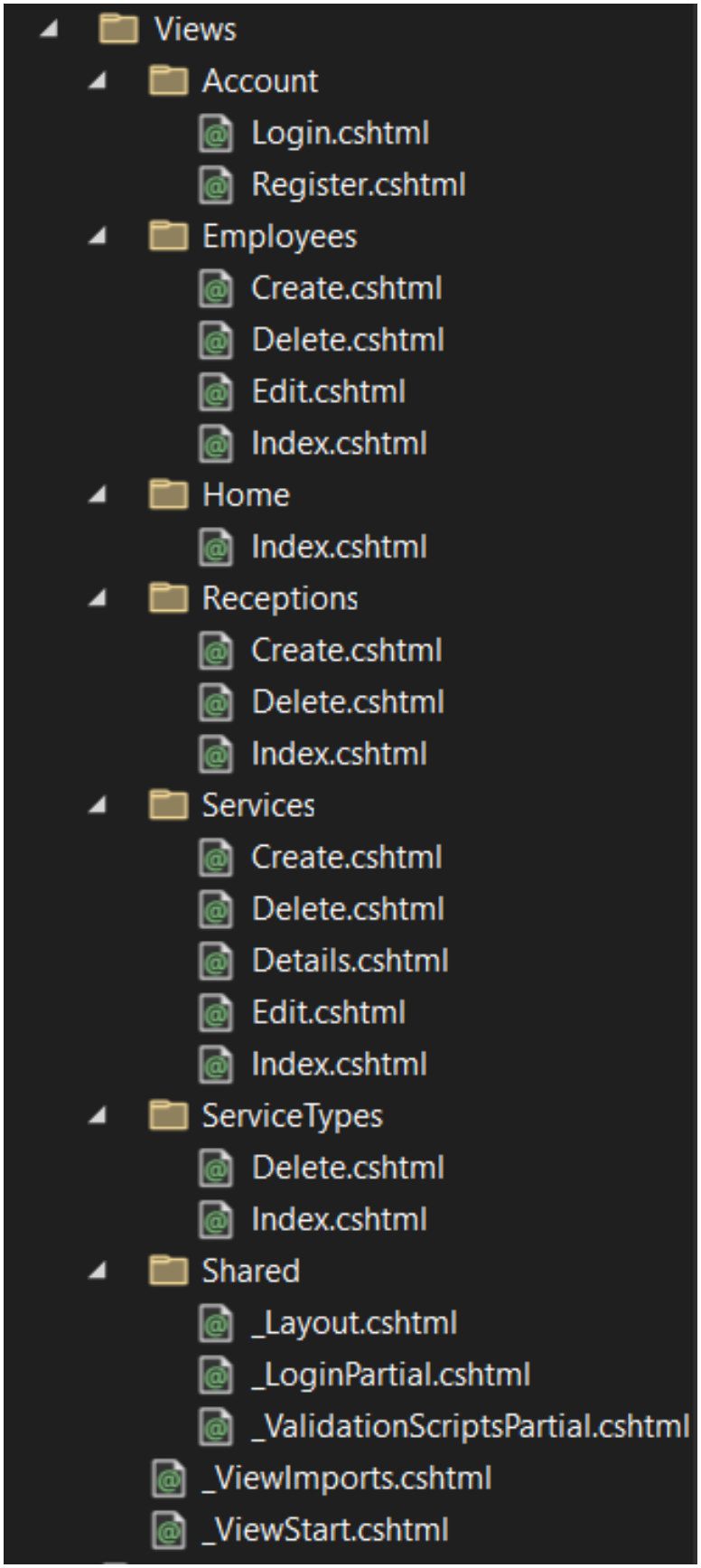


Рисунок 3.22 – Структура представлень

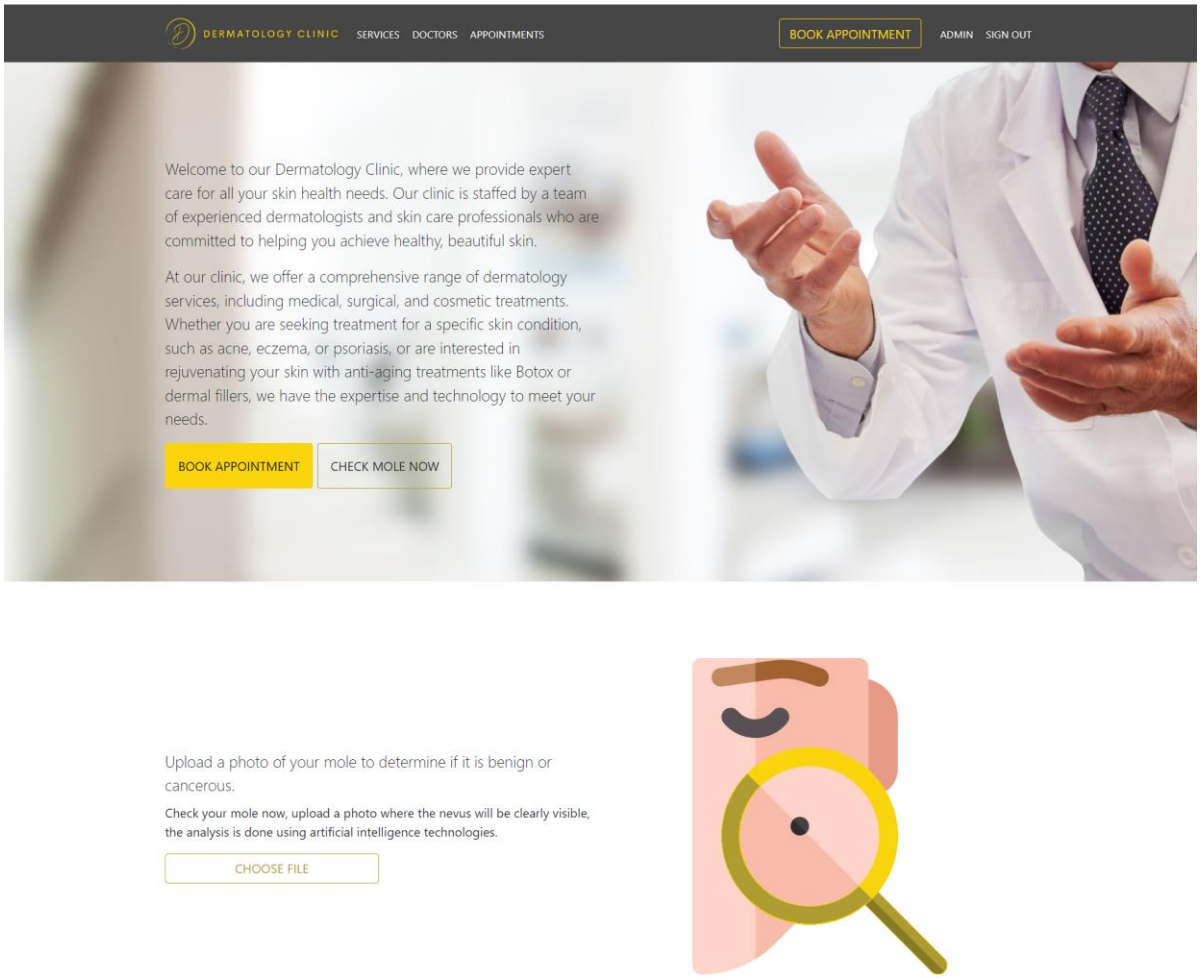


Рисунок 3.23 – Інтерфейс користувача головної сторінки

Представлення сторінки перегляду послуг: тут міститься заголовок та послуги, сортовані за категоріями, а також доступні лише адміністратору кноки створення нової категорії послуг і створення нової послуги.

Створення та редагування категорій послуг відбувається на окремій сторінці, що доступна лише адміністратору. Видалити категорію послуг він може лише після того, як видалить усі послуги цієї категорії.

Усі послуги сортовані за категоріями та містяться в відповідних секціях, вони завантажуються на сторінку за допомогою «foreach» і знаходяться в «ViewBag.ServiceTypes» та «ViewBag.Services».

Для кожної категорії є окрема секція, при натисканні на яку, карточка з іменем секції перевенеться і на її іншій стороні можна буде побачити послуги, що до неї відносяться. Такі анімації було створено за допомогою CSS, які містяться в окремому файлі.

Для перегляду деталей щодо послуги на послугу в списку потрібно натиснути, після чого відкриється відповідна сторінка. Навпроти кожної послуги є дві додаткові кнопки, які доступні лише для адміністратора: видалити та редагування внформації щодо послуги, які відкриють відповідні сторінки.

На сторінці деталей щодо послуги клієнт може побачити детальну інформацію про неї, таку яку і’мя, категорія, ціна, тривалість та детальний опис. Адміністратор додатково має можливість видалити або редагувати обрану послугу з цієї сторінки.

При натисканні кнопки видалити буде відкрито окрему сторінку з підтвердженням видаленя щоб запобігти випадковому видалення даних. Така сторінка з підтвердженням відкривається не тільки при спробі видалення послуги, а й при видаленні інформації про лікарів та записів.

Інтерфейс користувача сторінки перегляду послуг зображено на рисунку 3.24.

Рисунок 3.24 – Інтерфейс користувача сторінки перегляду послуг

Хоча ознайомитись з лікарями, до яких клієнт може записатися, можні з головної сторінки додатку, буда створе окрема сторінка для відображення списку усіх лікарів. Також, на цій сторінці адміністратор може додати нового лікаря в систему, редагувати інформацію про вже існуючих лікарів або видалити їх.

В системі зберігається фотографія лікаря, його ім’я, вік, детальна інформація та номер телефону, який може бачити лише адміністратор.

У результаті ця сторінка виглядає так: (рисунок. 3.25).

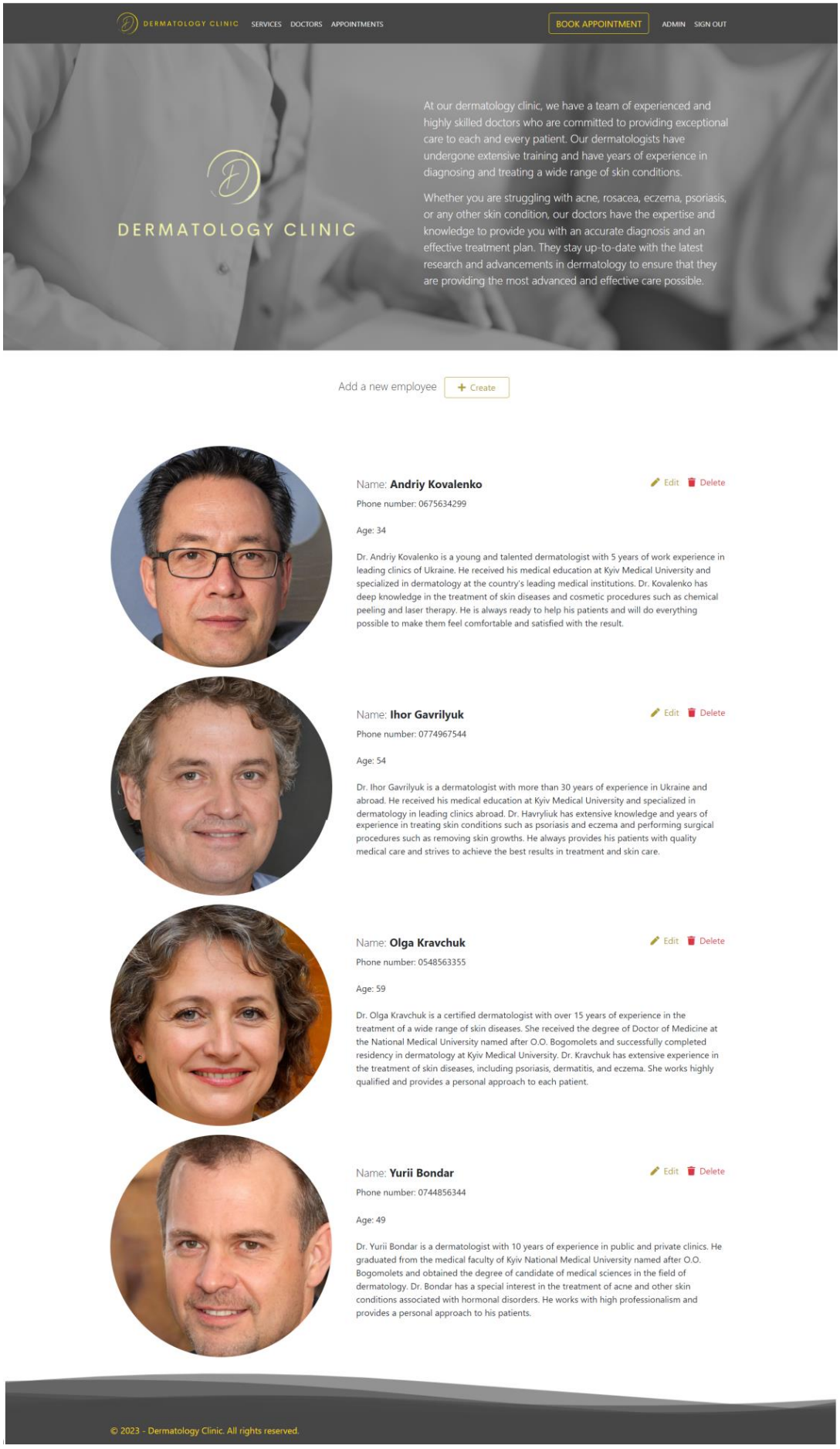


Рисунок 3.25 – Інтерфейс користувача сторінки перегляду лікарів

Записатись до клініки клієнт може на сторінці створення запису, щоб потрапити на цю сторінку потрібно натиснути відповідну кнопку в заголовці кожної сторінки, або з головної сторінки, тому що там дублюється ця кнопка. Адміністатор має можливість записати до клініки обраного клієнта.

На сторінці створення запису зліва міститься список послуг, які доступні для вибору, після вибору послуги пропонується обрати лікаря та клієнта (для адміністратора). Коли послуги та лікар обрані, можна обрати бажану дату прйому та час зпоміж доступних варінтів. Доступні години для запису залежать від того, чи немає на цнй самий час інших записів до цього лікаря та враховується тривалість послуг таким чином, щоб вони будли виконані до закінчення робочого дня та до початну наступного запису іншого клієнта.

Створюючи запис, можна одразу побачити загальну вартість та тривалість обраних послуг поруч з кнопкою створення запису

Адміністратор може записати клієнта на будь-яку послугу, самостійно клієнт може записатись лише на послугу, яка відноситься до категорії консультацій, так як інші медичні процедури рекомендується проводити лише за назначенням лікаря.

Після відвідування консультації лікар визначить подальший курс лікування і менеджер зможе записати клієнта на потрібні послуги. Для цього він обере потрібні послуги, час, дату та лікаря, а також обере клієнта, якого записує. Ця функція є доступною лише для обікового запису з правами адміністратора (редактора) системи.

На сторінці власного кабінету клієнт зможе бачити детальну інформацію щодо його записів і тих що він робив самостійно, і тих які робив адміністратор для нього.

Сторінка створення запису для адміністратора виглядатиме так: (рисунок. 3.26).

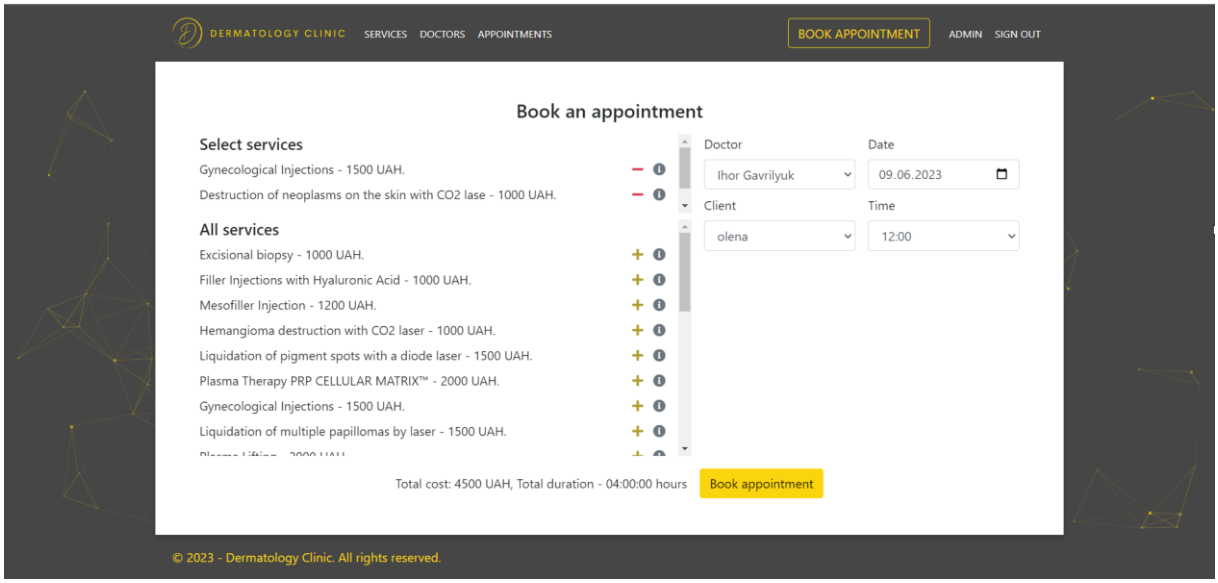


Рисунок 3.26 – Інтерфейс користувача сторінки запису до клініки

Скріншоти усіх сторінок застосунку знаходяться у додатку А.

ВИСНОВКИ

В ході виконання даної роботи було проведено аналіз предметної області та постановлено задачі до виконання. Було визначено функціонал, що підлягає реалізації та визначено методи та засоби для створення підсистеми.

Вимоги до ситеми було сформовано у вигляді UseCase діаграми та надано опис кожного прецеденту у вигляді UserStories. Було проведено функціональне моделювання, використовуючи стандарт методології функціонального моделювання IDEF0, а для визначення функціональної структури була побудована концептуальна діаграма потоків даних та її декомпозиція.

Було проведено логічне та фізичне моделювання бази даних з використанням стандарту IDEF1X і сформовані логічна та фізична моделі даних «сутність-зв’язок».

Було реалізовано додаток для автоматизації роботи дерматологічної клініки з функцією аналізу новоутворення на шкірі за фото, за допомогою якого користувачі можуть зробити запис до клініки переглянувши доступні послуги з інформацією, такою як ціна та тривалість процедури, та обравши бажаного лікаря. Також, підсистема має функцію розпізнавання злоякістніх та доброякісних новоутворень на шкірі за фотографією, цією функцією може скористатись кожен користувач додатку з головної сторінки та перевірити турбуюче новоутворення не виходячи з дому. Завдяки такій додатковій функції, розроблений веб-застосунок вигідно виділяється з поміж інших подібних застосунків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ісаєва О. А., Аврунін О. Г., Трубіцин О. О. Метод відеодерматоскопії при оцінці стану шкіри під час атопічного дерматиту / Матеріали II Міжнародної науково-практичної інтернет-конференції «Інтеграція освіти, науки та бізнесу в сучасному середовищі: літні диспути». - Дніпро, Україна, 2020. 219 с.

2. Трубіцин О. О., Ісаєва О. А., Кліменко В. А., Аврунін О. Г. Інструментальні методи оцінки стану шкіри при атопічному дерматиті / Наука та виробництво: міжвуз.темат. зб. наук. пр. / ДВНЗ «ПДТУ». Вип.. 20. – Маріуполь, ПДТУ, 2019. 188 с.

3. Ісаєва О. А., Аврунін О. Г. Можливості застосування 3D-сканування при визначенні площі уражених ділянок шкіри / Матеріали I Міжнародної науково-практичної конференції "Авіація, промисловість, суспільство", Ч. 1. Кременчук, КЛК ХНУВД, – 2020. 513 с.

4. Ісаєва О. А., Аврунін О. Г. Обробка зображень для відеодерматоскопії / Наукові праці IV Міжнародної науково-практичної конференції Безпека на транспорті - основа ефективної інфраструктури: проблеми та перспективи, Харків, ХНАДУ. – 2019. 230 с.

5. Аврунін О. Г., Трубіцин О. О., Ісаєва О. А., Кліменко В. А., Possibilities for assessing the effectiveness of treatment of atopic dermatitis based on analysis of color characteristics of video dermatoscopic images / Innovative Technologies and Scientific Solutions for Industries, Харків, ХНАДУ, 2020, 133 p.

6. Steven Sanderson Pro ASP.NET MVC 2 Framework / Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, Berlin, 2010. 560 с.

7. MSDN Library. URL: http://msdn.microsoft.com/ru-ru/library/default.aspx (дата звернення: 17.03.2023).

8. ASP.NET Core. URL: https://metanit.com/sharp/aspnet5/1.1.php – Назва з екрана (дата звернення: 17.03.2023).

9. Довідник з C#. URL: https://msdn.microsoft.com/ru-ru/library/618ayhy6 .aspx (дата звернення: 17.03.2023).

10. Andrew Troelsen C# and .NET library. / Apress Berkeley, CA, 2007. 900 p.

11. ISIC-archive. URL: https://www.isic-archive.com/ (дата звернення: 17.03.2023).

12. Борисенко В. П., Левикін В. М., Пономарьов Ю. А. Об'єктно-орієнтований аналіз та проектування ІВС на основі UML / Харків: ХНУРЕ, 2004. 80 с.

13. Томашевський О. М. Інформаційні технології та моделювання бізнес процесів: практ. посіб. Харків, 2012. 296 с.

14. Практичний посібник зі створення UML-діаграм. URL: https://flexberry.github.io/ru/gpg_practical-guides-uml.html (дата звернення: 17.03.2023).

15. Gao J., Guo Y., Wang Z. Matrix neural networks. Advances in neural networks / Proceedings of the 14th International Symposium on Neural Networks. Sapporo, Japan, 2017. 10 p.

16. Pehamberger, H.; Steiner, A.; Wolff, K. In vivo epiluminescence microscopy of pigmented skin lesions. I. Pattern analysis of pigmented skin lesions. / J. Am. Acad. Dermatol. 1987, 583 p.

17. Stolz, W. ABCD rule of dermatoscopy: A new practical method for early recognition of malignant melanoma. / Eur. J. Der-Matol. 1994, 527 p.

18. Pacheco, A.G.; Krohling, R.A. The impact of patient clinical information on automated skin cancer detection. Comput. Biol. Med. 2020, 116 p.

19. Bodyanskiy Y., Boiko O., Pliss I., Kopaliani D., Volkova V. 2D-Deep Neural Network and Its Online Rapid Learning. Proceedings of the 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Metz, France, 2019, 307 p.

20. Huang C. ReLU networks are universal approximators via piecewise linear or constant functions. Neural computation. URL: https://doi.org/10.1162/neco_a_01316 (дата звернення: 17.03.2023).

21. Murugan, A.; Nair, S.A.H.; Preethi, A.A.P.; Kumar, K.P.S. Diagnosis of skin cancer using machine learning techniques. Microprocess. Microsystems 2020, 81 p.

22. Haenssle, H.A.; Fink, C.; Schneiderbauer, R.; Toberer, F.; Buhl, T.; Blum, A.; Kalloo, A.; Hassen, A.B.H.; Thomas, L.; Enk, A.; et al. Man against machine: Diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. / Ann. Oncol. Off. J. Eur. Soc. Med. Oncol. 2018, 29 p.

23. Brinker, T.J.; Hekler, A.; Enk, A.H.; Klode, J.; Hauschild, A.; Berking, C.; Schilling, B.; Haferkamp, S.; Schadendorf, D.; Fröhling, S.; et al. A convolutional neural network trained with dermoscopic images performed on par with 145 derma-tologists in a clinical melanoma image classification task. / Eur. J. Cancer 2019, 154 p.

ДОДАТОК А

Скриншоти візуального інтерфейсу

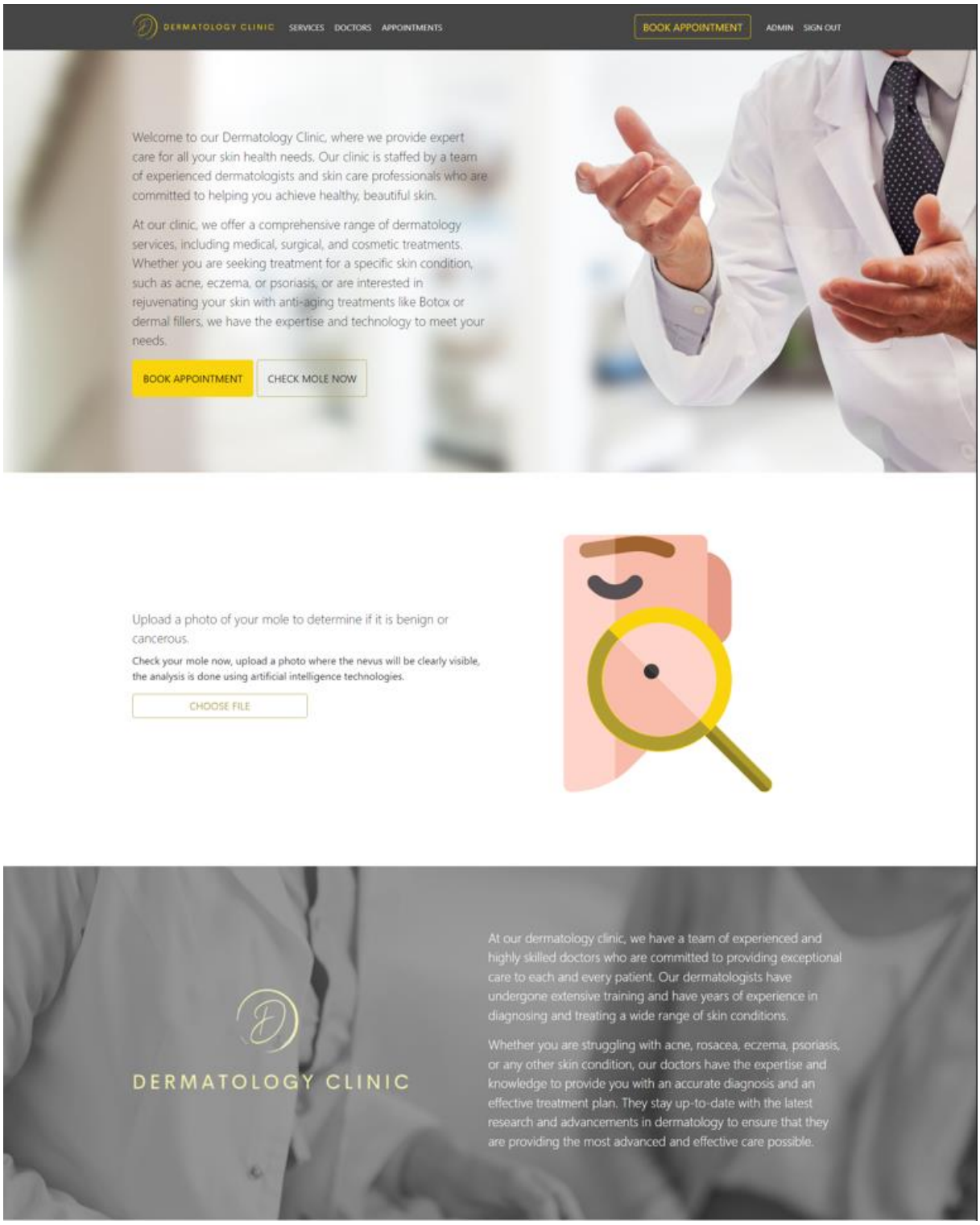


Рисунок А.1 – Дизайн головної сторінки

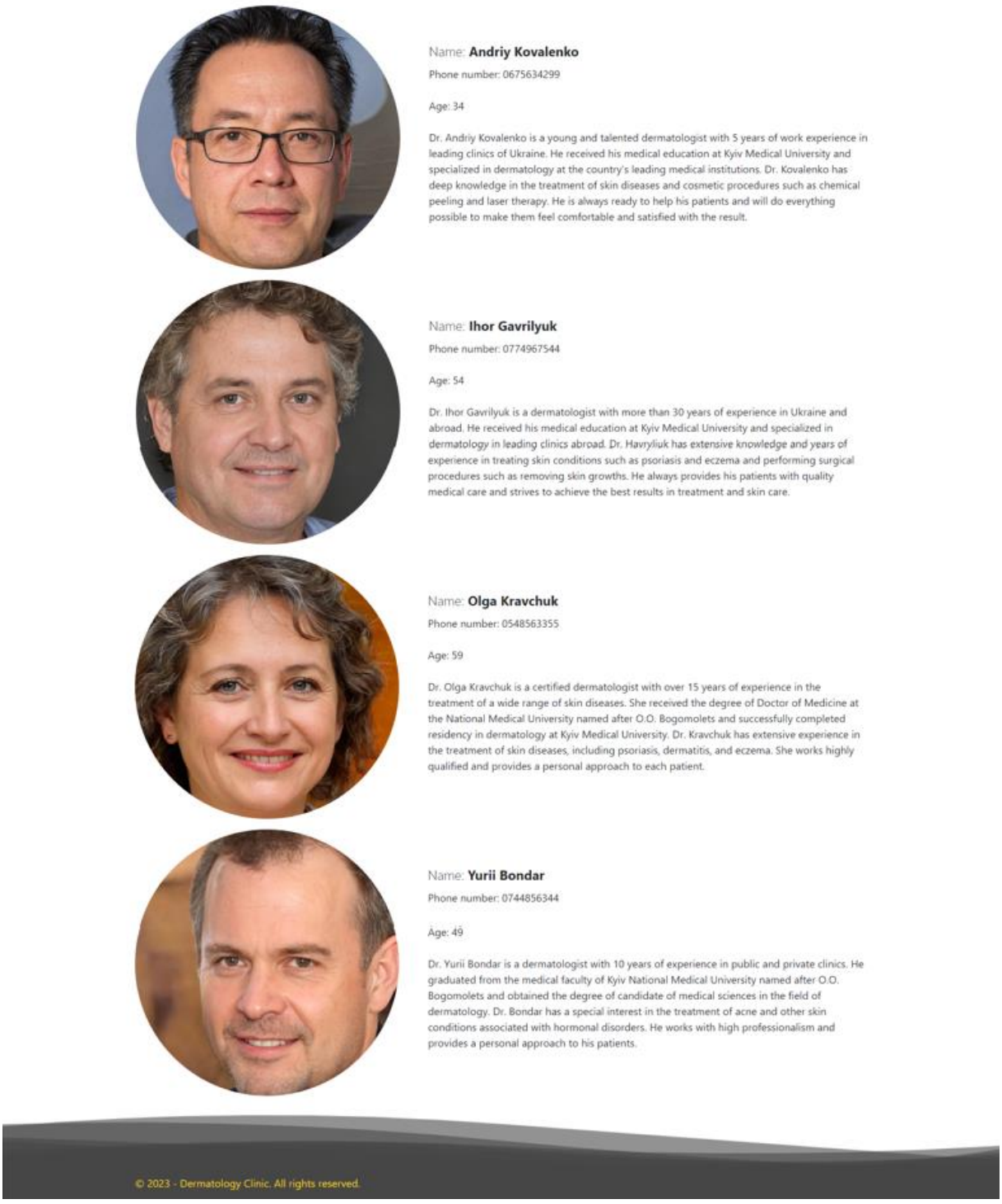


Рисунок А.2 – Дизайн головної сторінки (продовження)

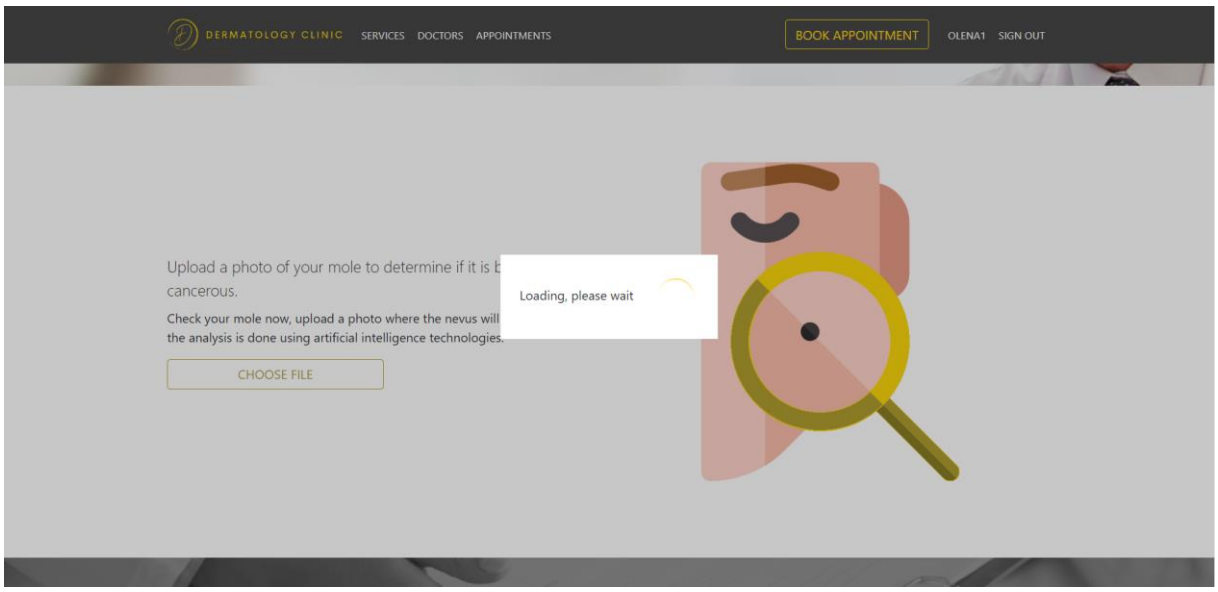


Рисунок А.3 – Дизайн сторінки аналізу новоутворення на шкірі (завантаження)

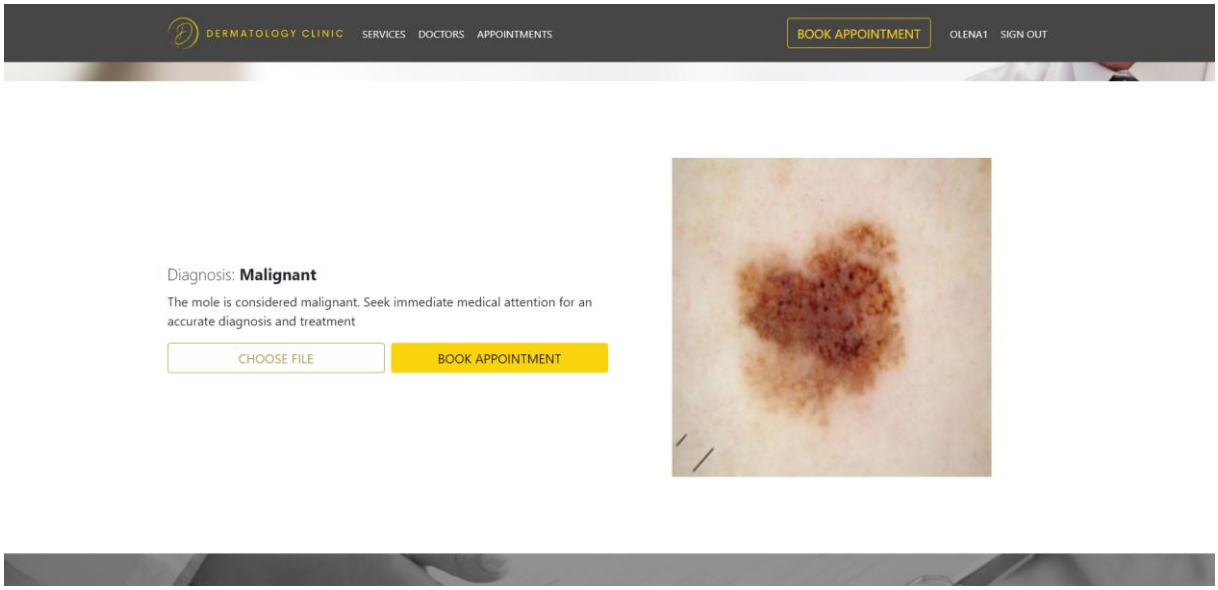


Рисунок А.4 – Дизайн сторінки аналізу новоутворення на шкірі (злоякісне новоутворення)

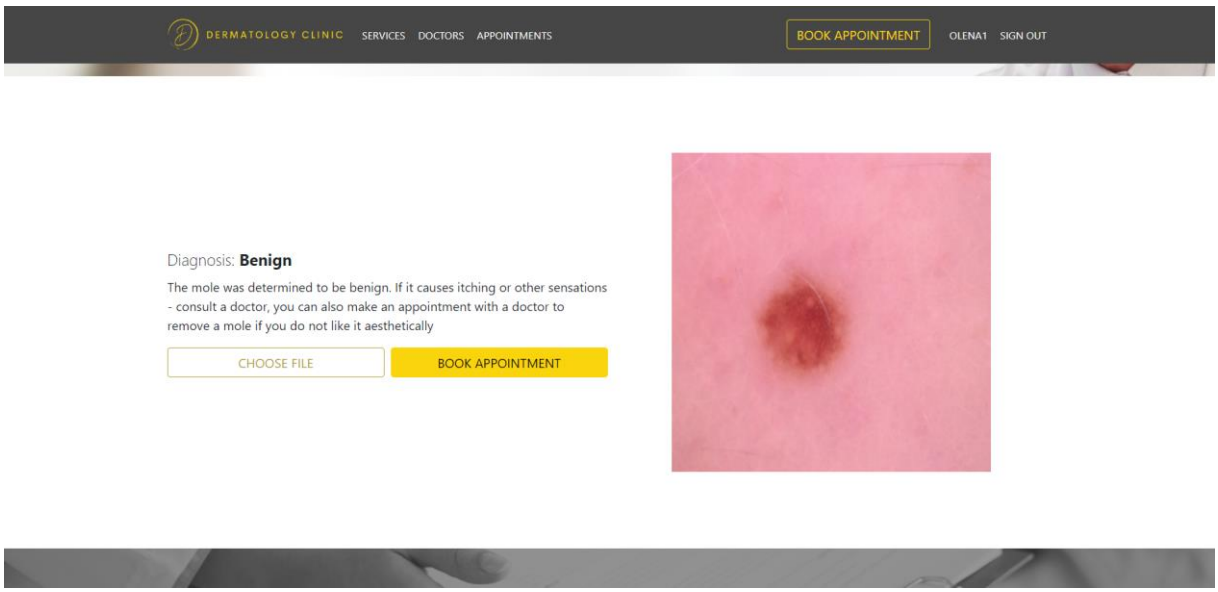


Рисунок А.5 – Дизайн сторінки аналізу новоутворення на шкірі (доброякісне новоутворення)

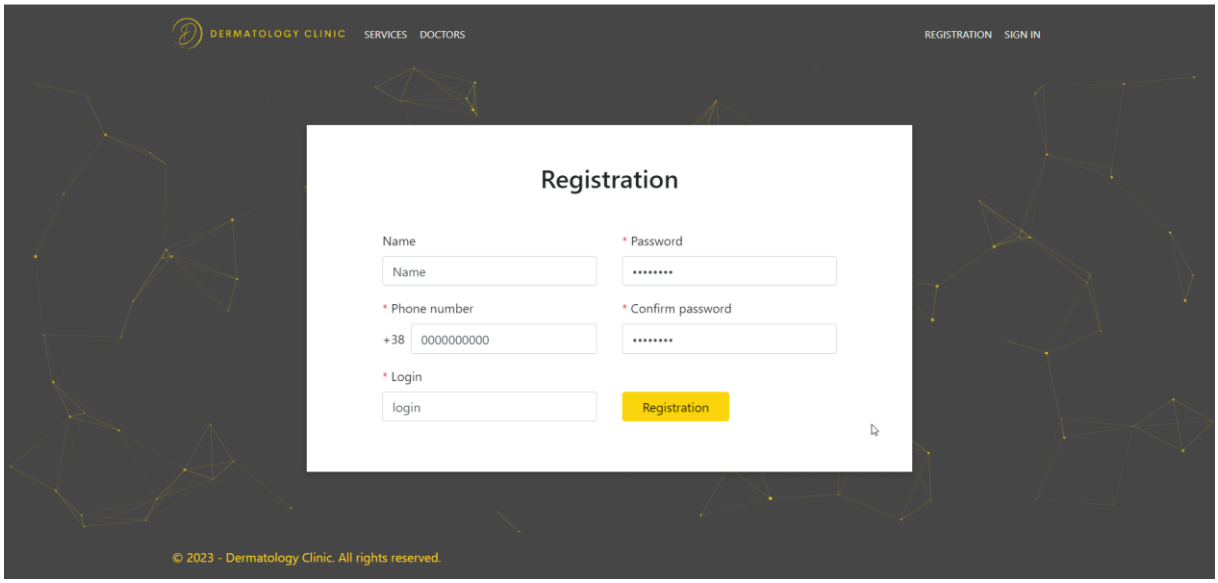


Рисунок А.6 – Дизайн сторінки реєстрації

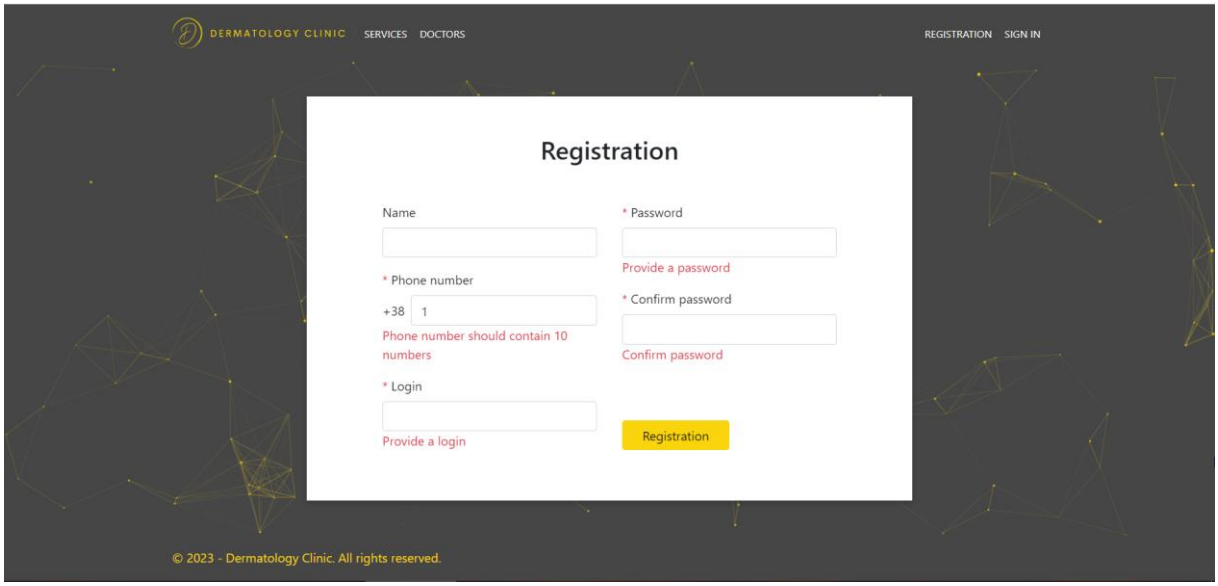


Рисунок А.7 – Дизайн сторінки реєстрації (введено некоректні дані)

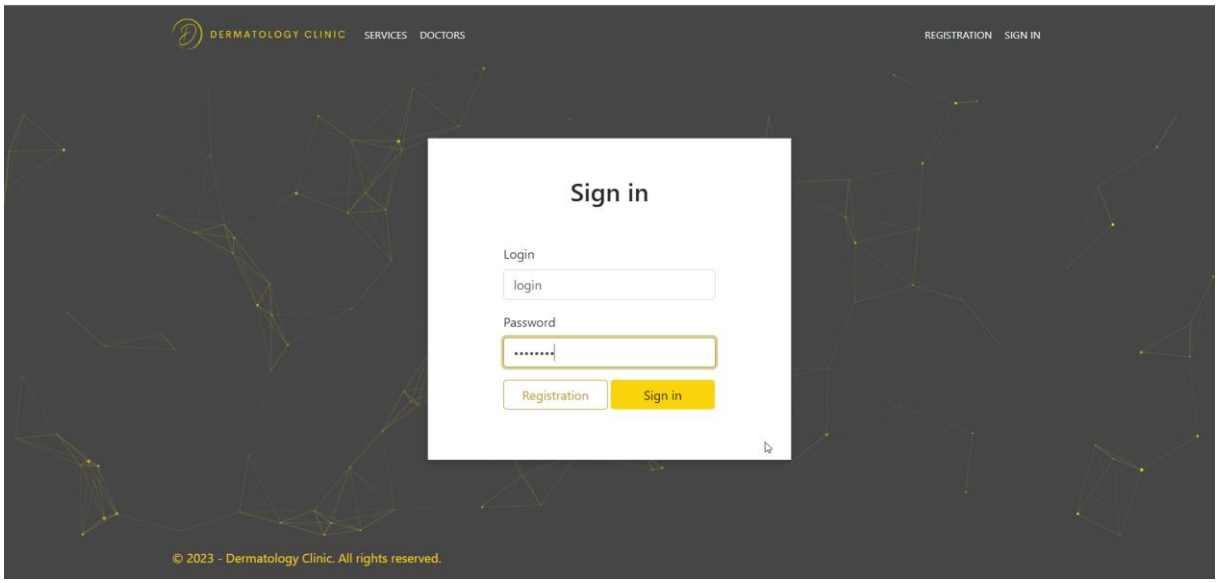


Рисунок А.8 – Дизайн сторінки входу

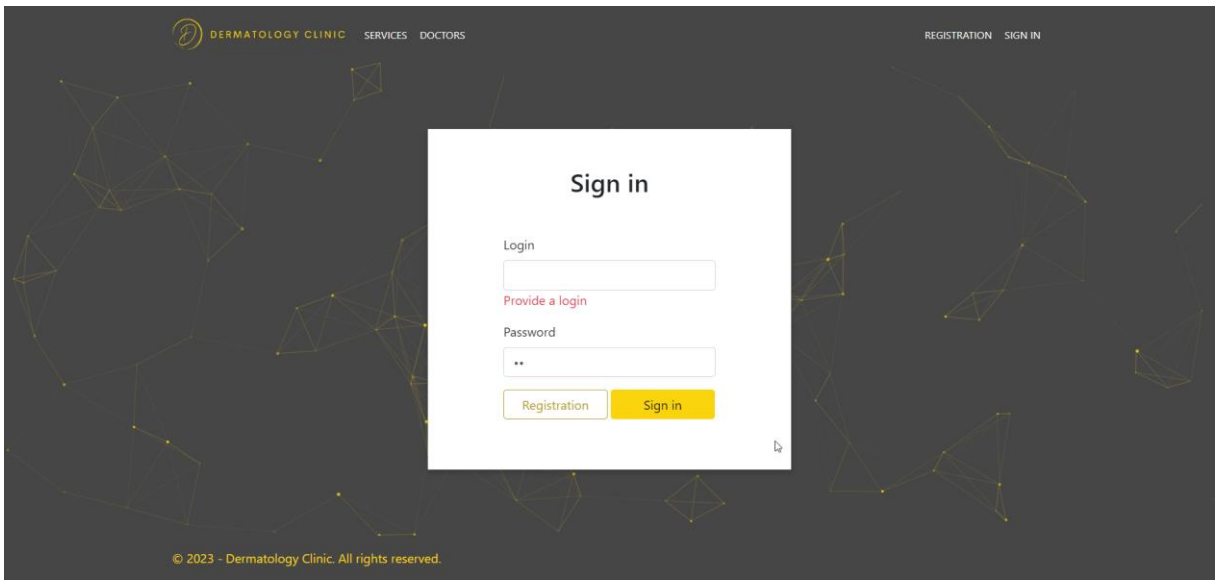


Рисунок А.9 – Дизайн сторінки входу (введено некоректні дані)

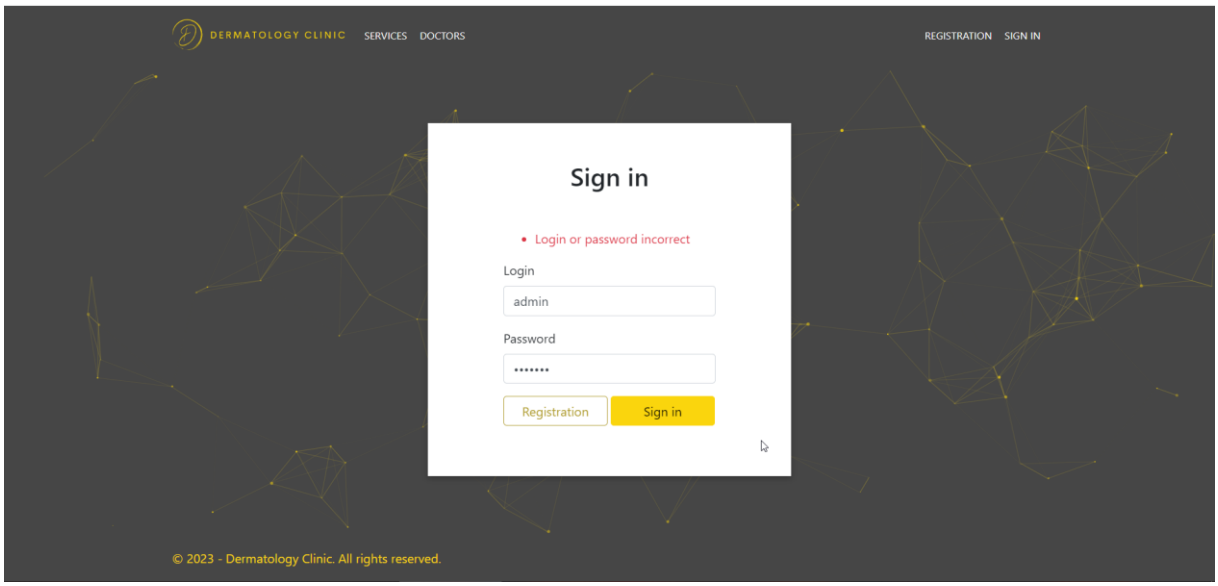


Рисунок А.10 – Дизайн сторінки входу (введено неправльний пароль)

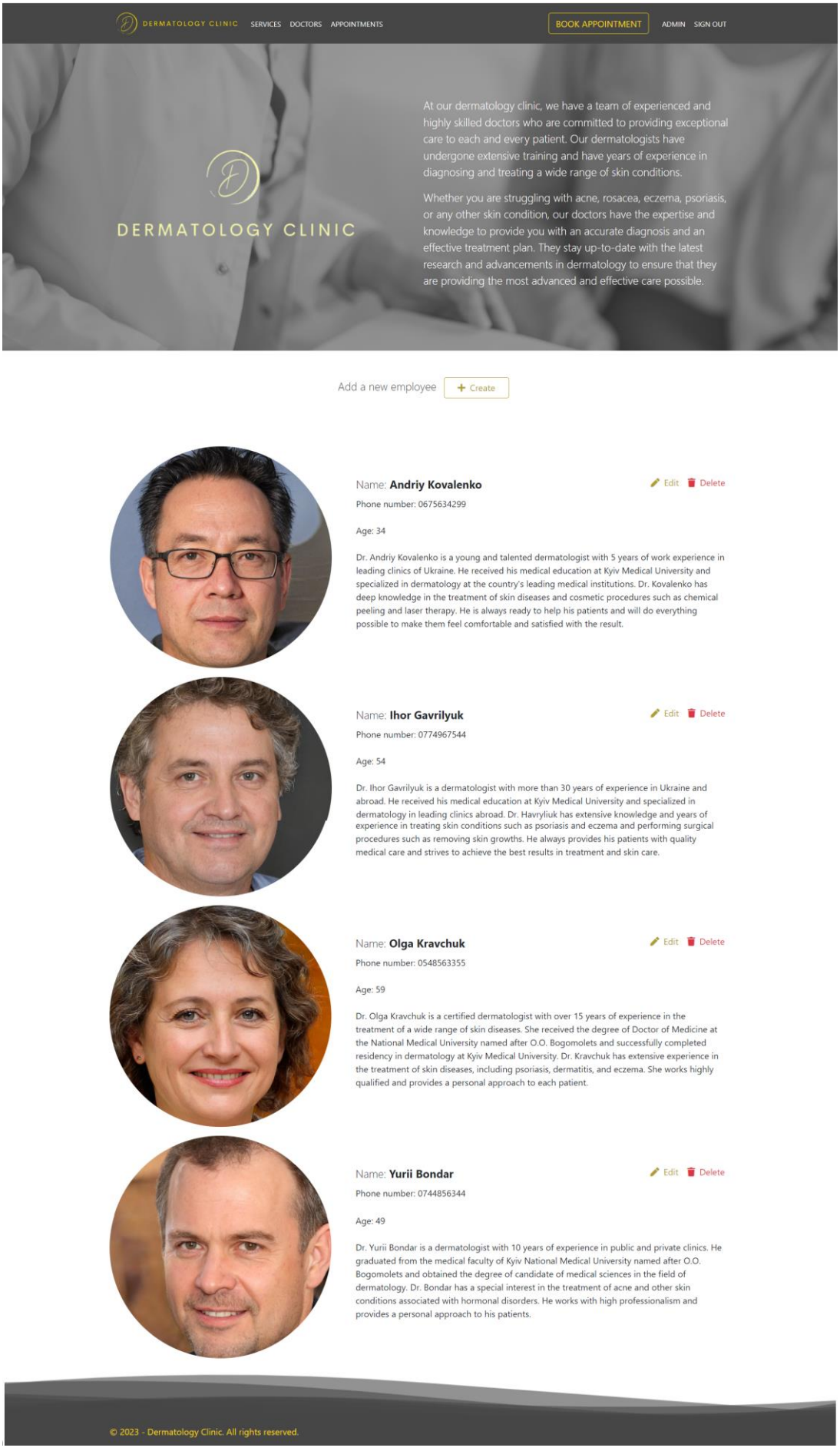


Рисунок А.11 – Дизайн сторінки списку лікарів (редактор)

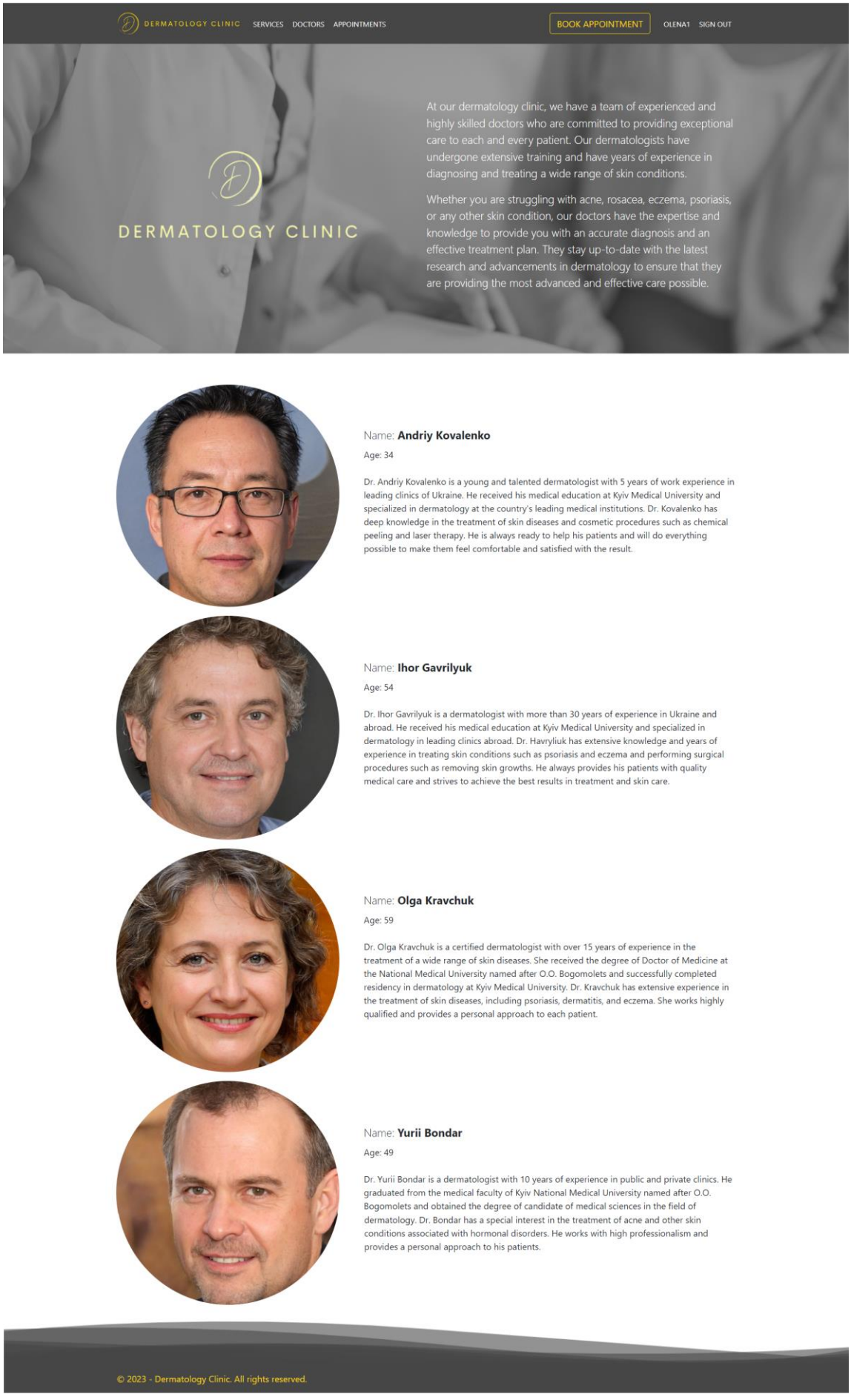


Рисунок А.12 – Дизайн сторінки списку лікарів (звичайний користувач)

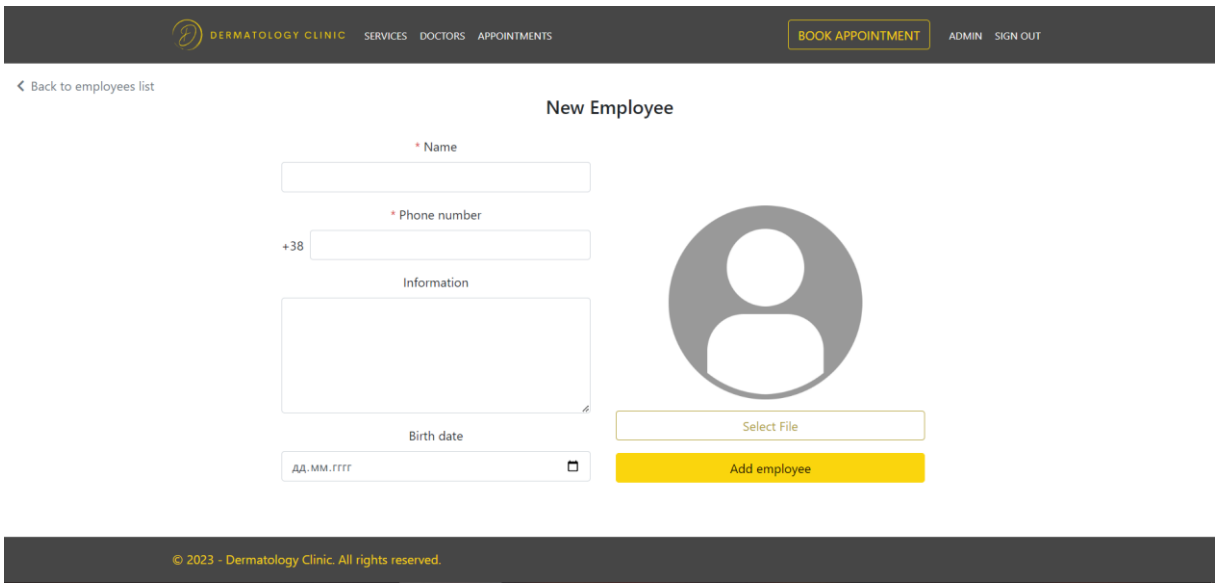


Рисунок А.13 – Дизайн сторінки створення нового лікаря

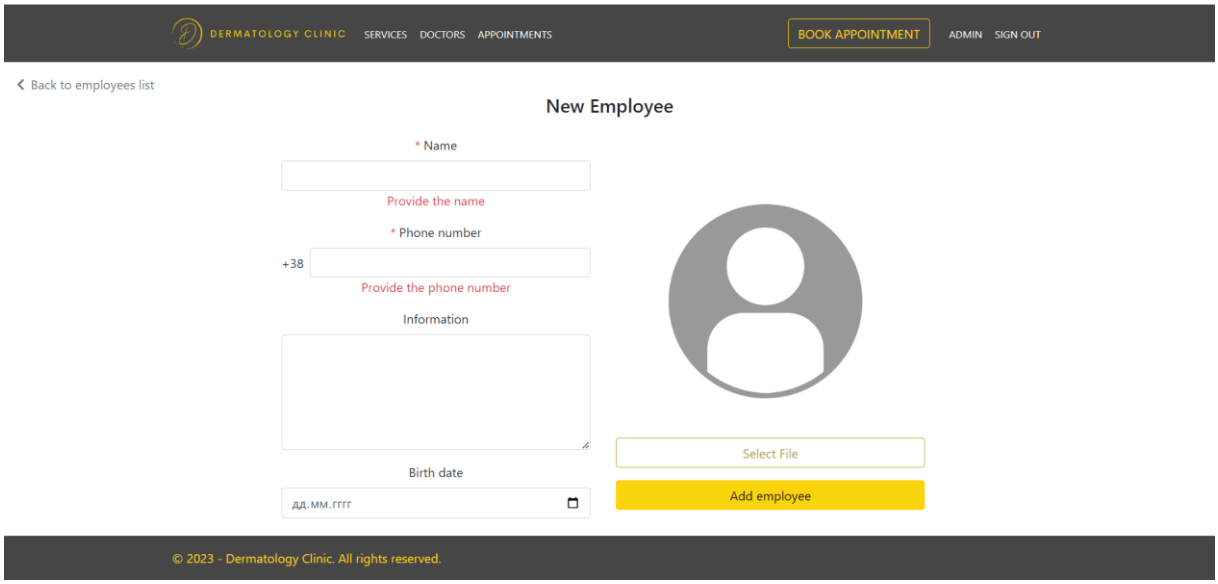


Рисунок А.14 – Дизайн сторінки створення нового лікаря (введено некоректні дані)

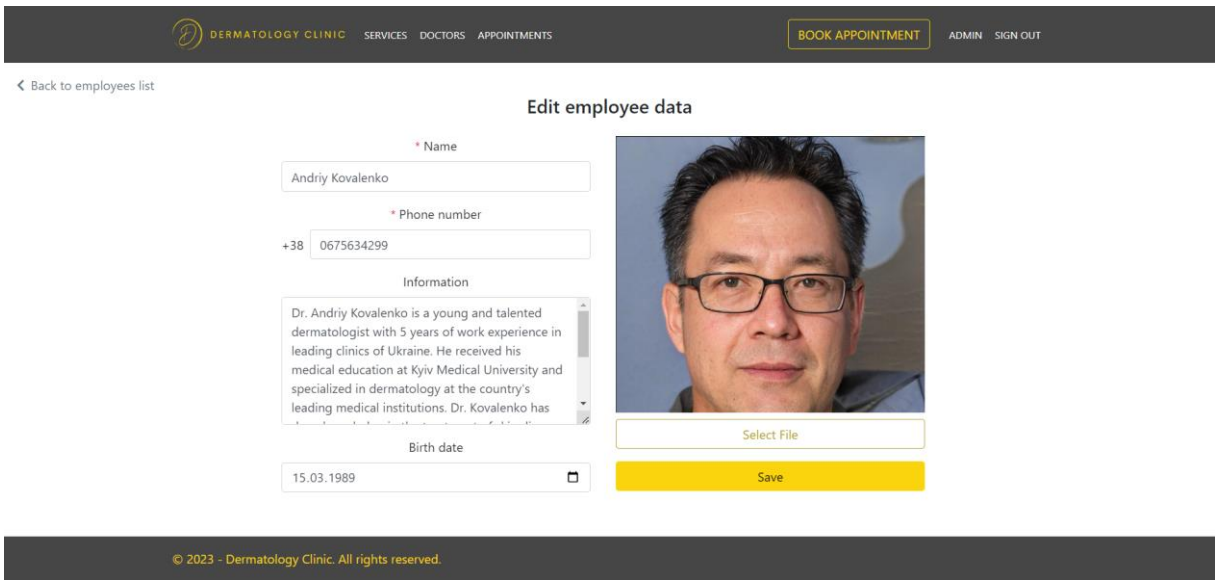


Рисунок А.15 – Дизайн сторінки редагування інформації щодо лікаря

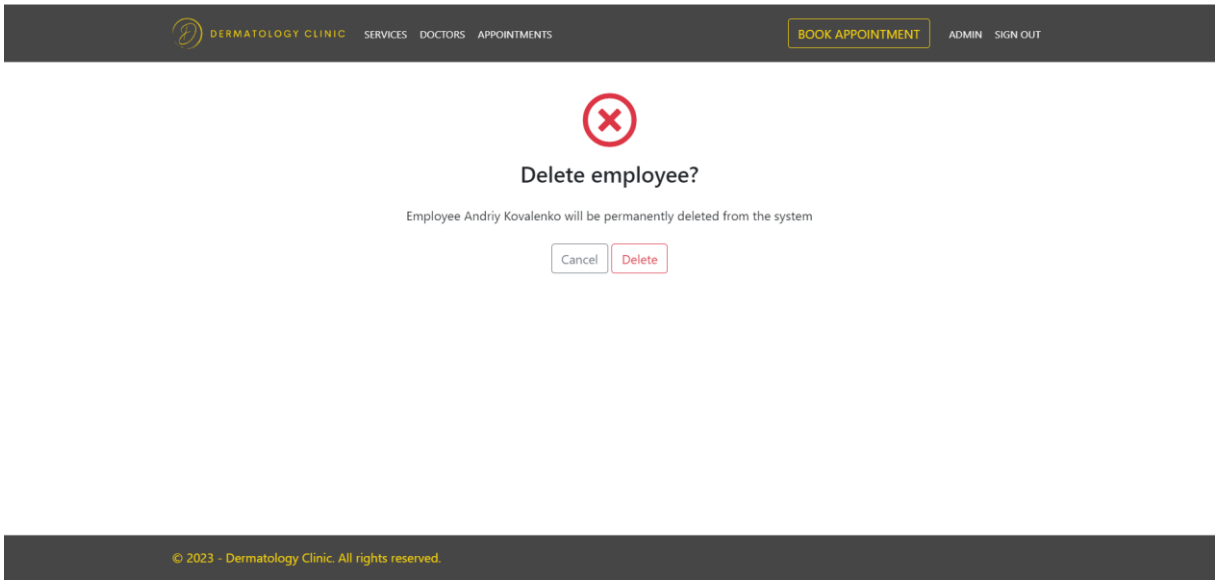


Рисунок А.16 – Дизайн сторінки видалення лікаря

Рисунок А.17 – Дизайн сторінки списку послуг (редактор)

Рисунок А.18 – Дизайн сторінки списку послуг (звичайний користувач)

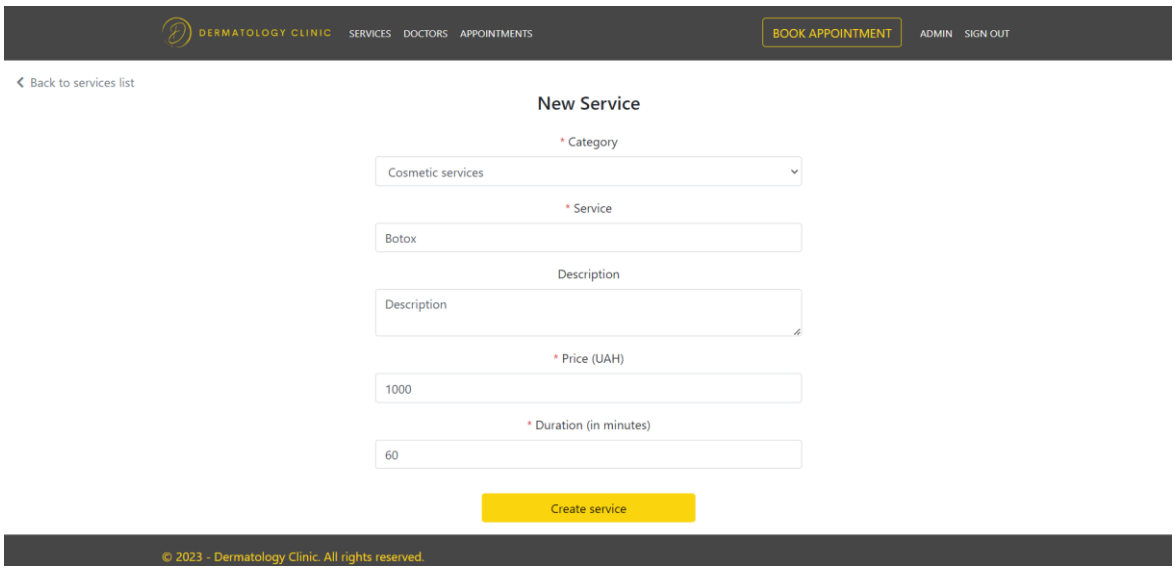


Рисунок А.19 – Дизайн сторінки створення нової послуги

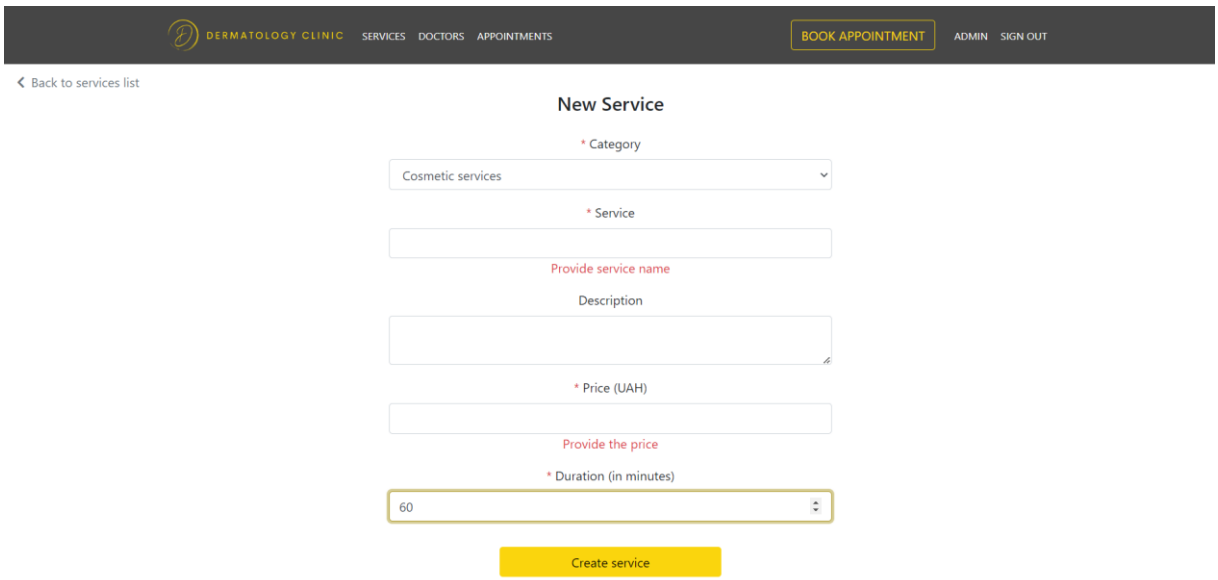


Рисунок А.20 – Дизайн сторінки створення нової послуги (введено некоректні дані)

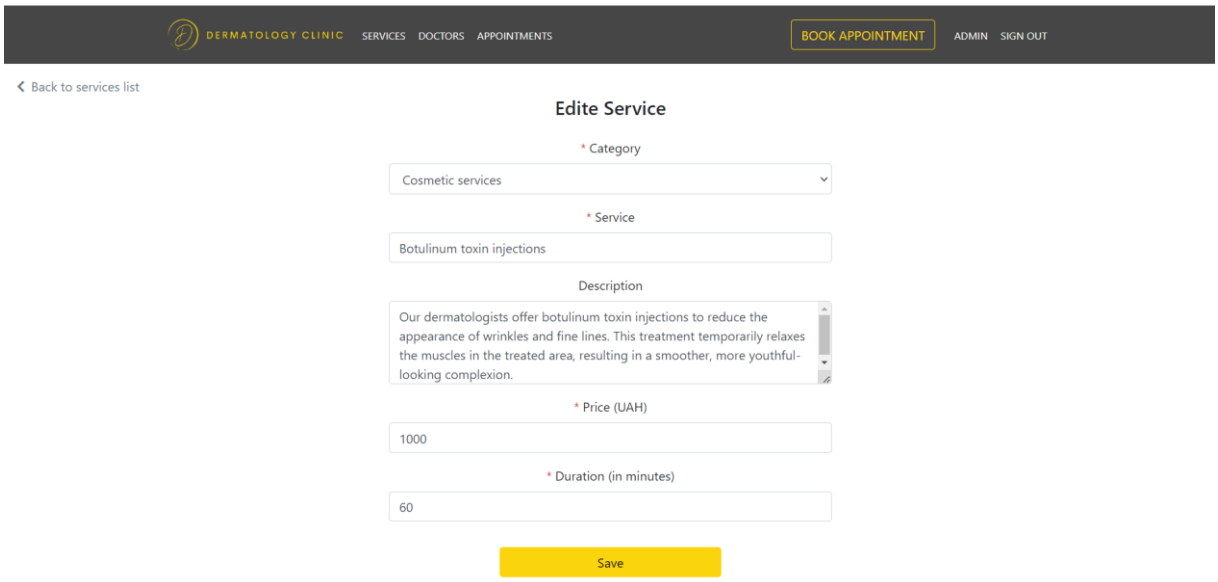


Рисунок А.21 – Дизайн сторінки редагування інформації щодо послуги

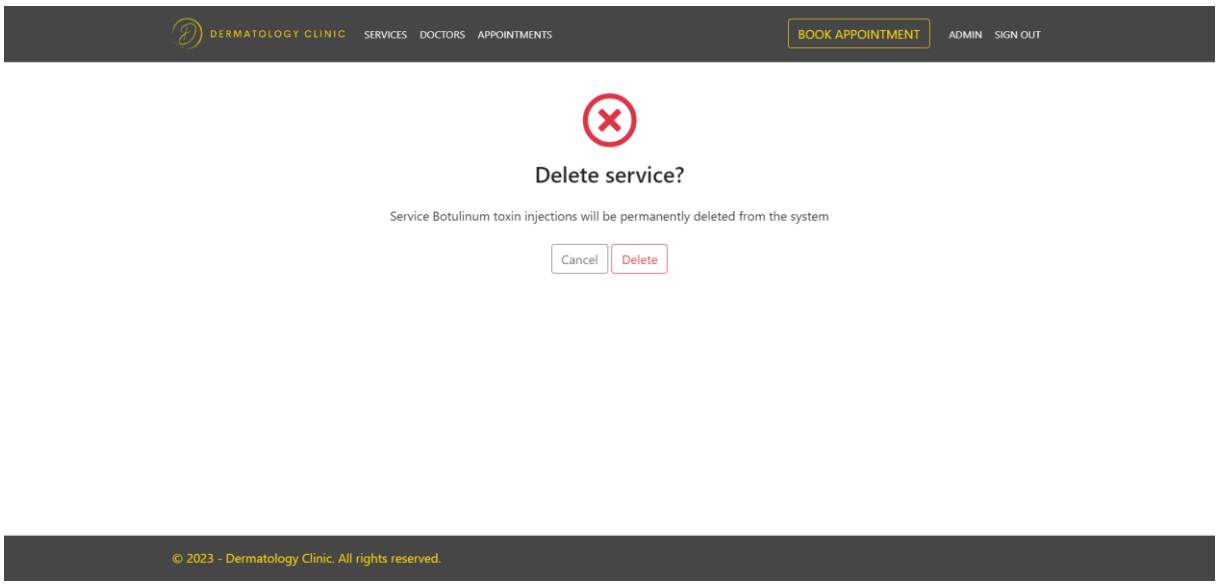


Рисунок А.22 – Дизайн сторінки видалення послуги

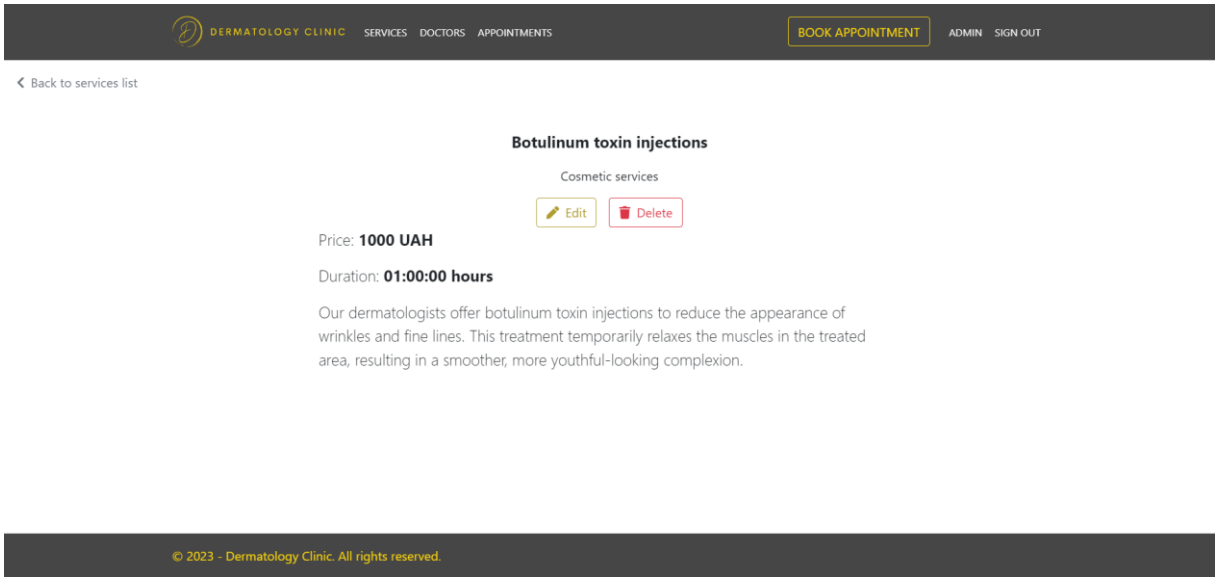


Рисунок А.23 – Дизайн сторінки деталей щодо послуги (редактор)

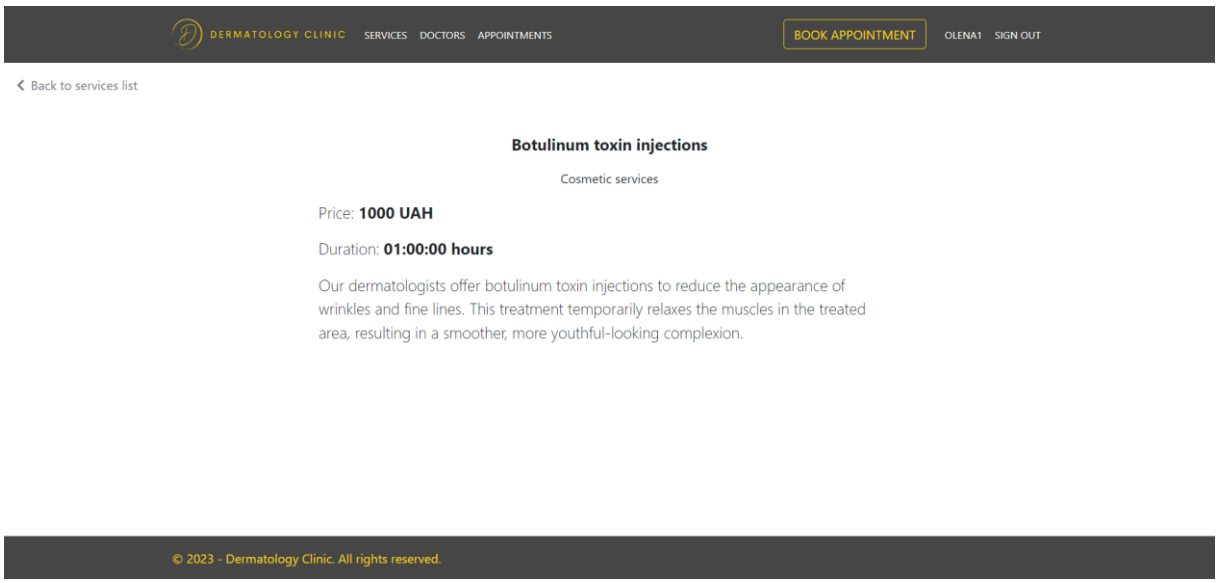


Рисунок А.24 – Дизайн сторінки деталей щодо послуги (звичайний користувач)

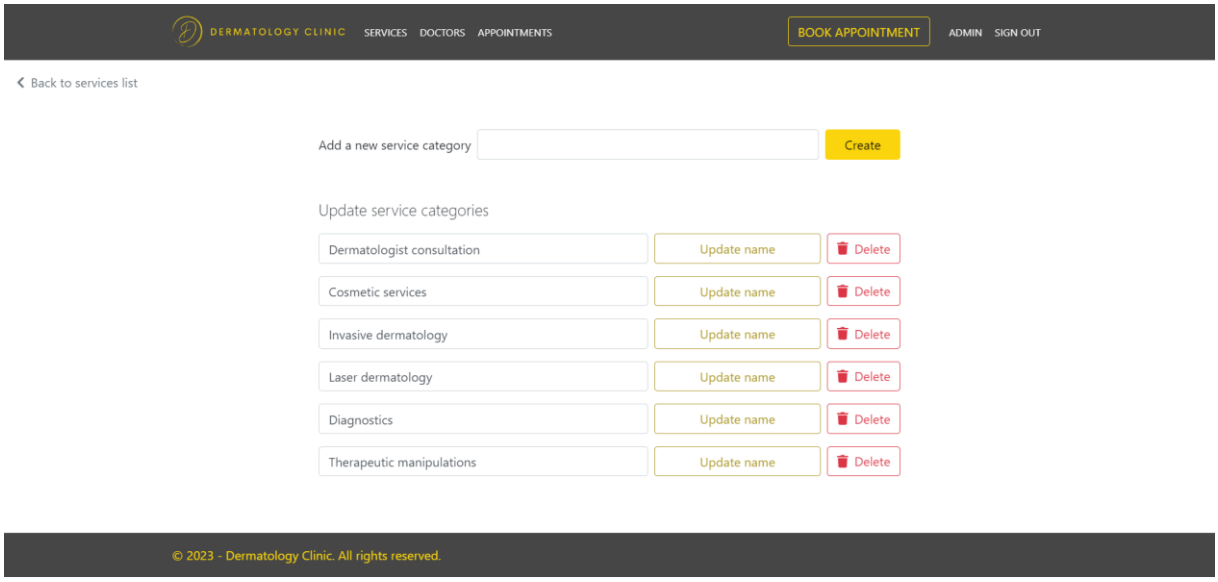


Рисунок А.25 – Дизайн сторінки редагування категорій послуг

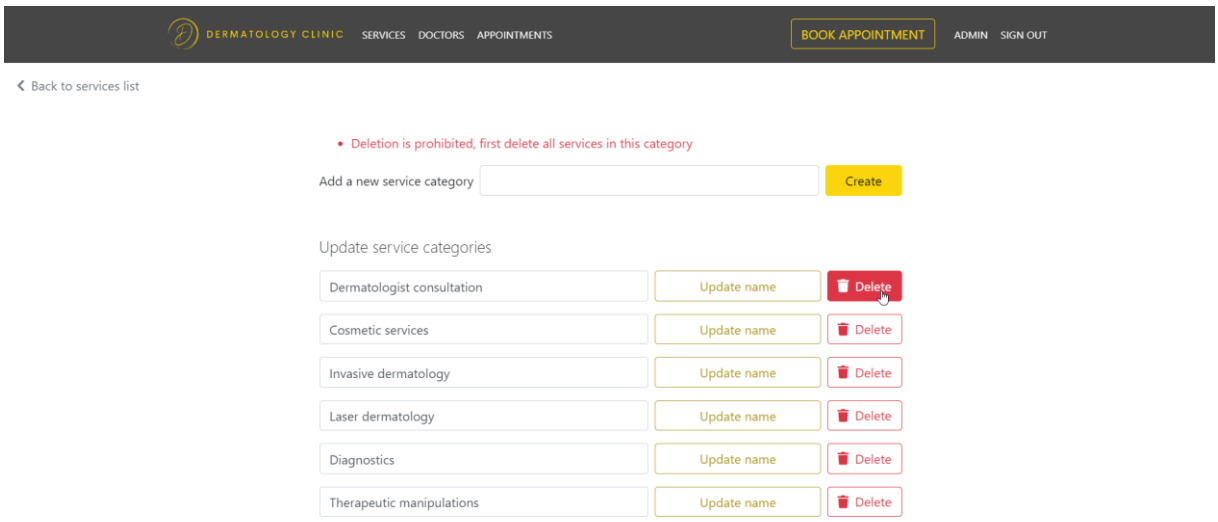


Рисунок А.26 – Дизайн сторінки редагування категорій послуг (видалення неможливо)

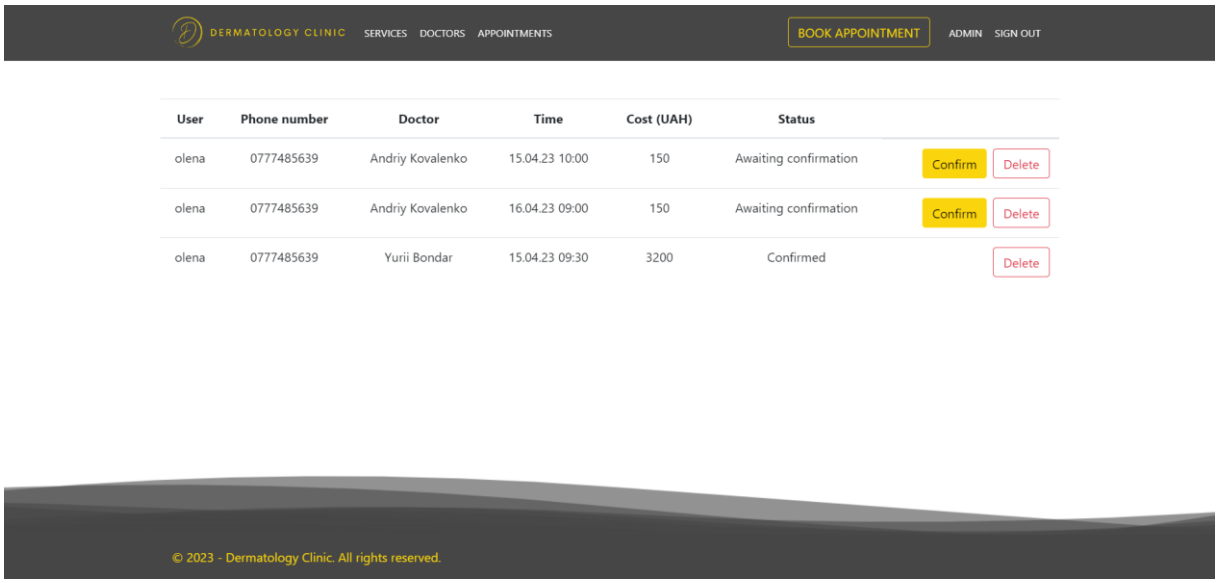


Рисунок А.27 – Дизайн сторінки списку записів (редактор)

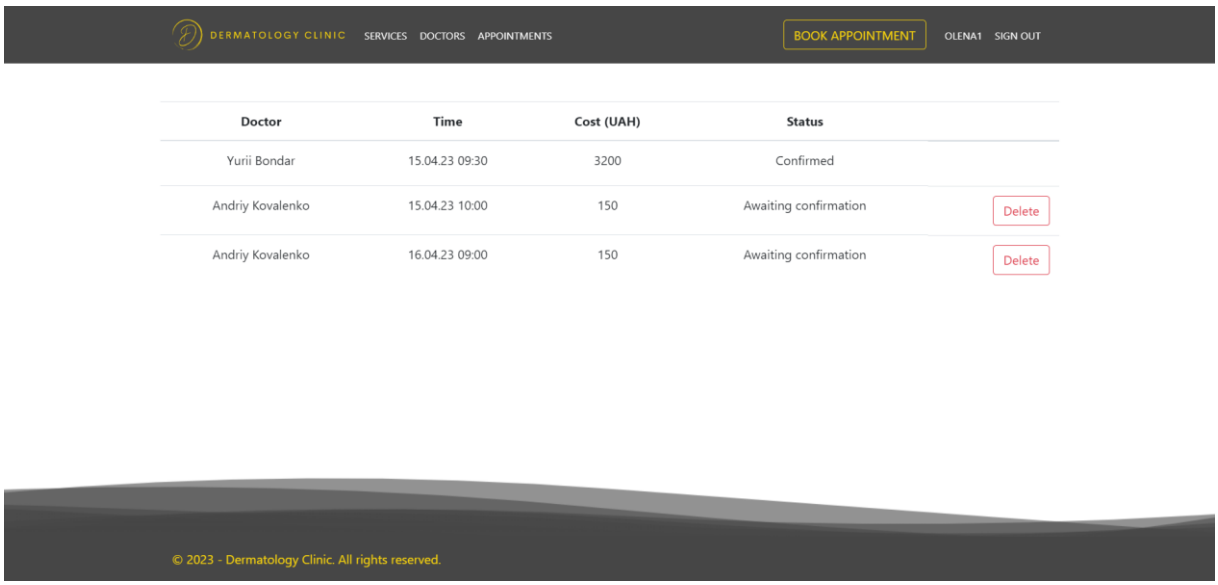


Рисунок А.28 – Дизайн сторінки списку записів (звичайний користувач)

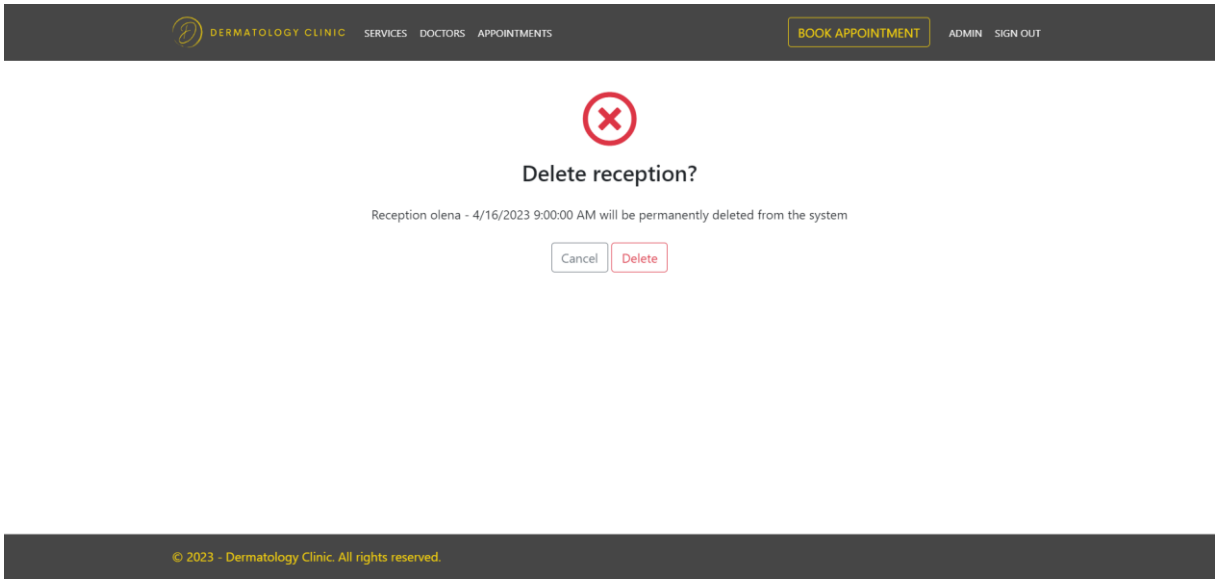


Рисунок А.29 – Дизайн сторінки видалення запису

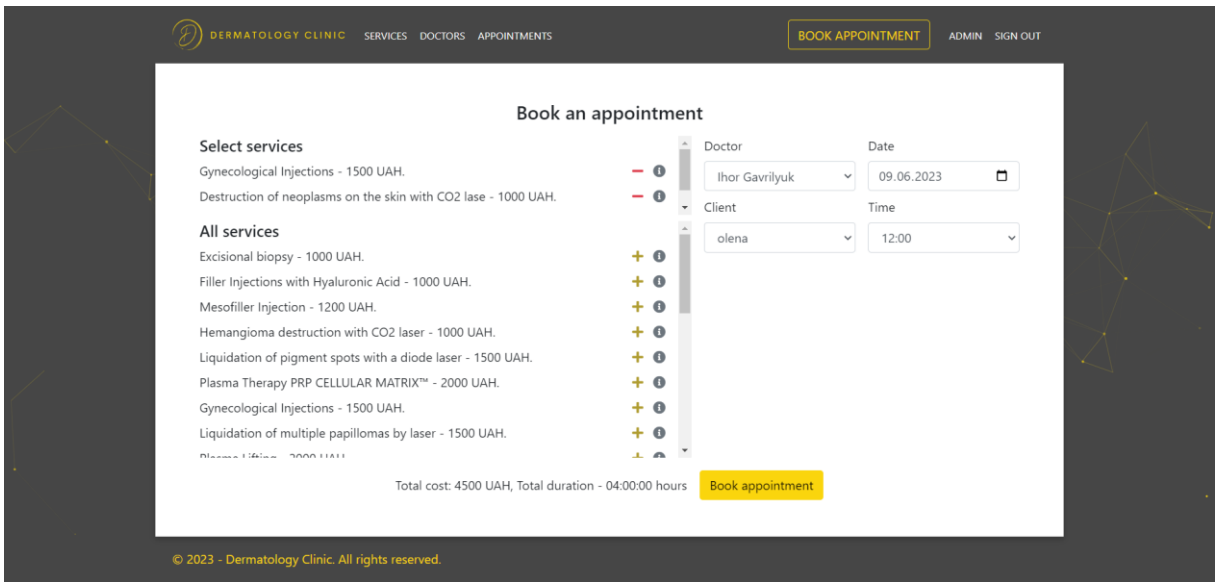


Рисунок А.30 – Дизайн сторінки створення запису (редактор)

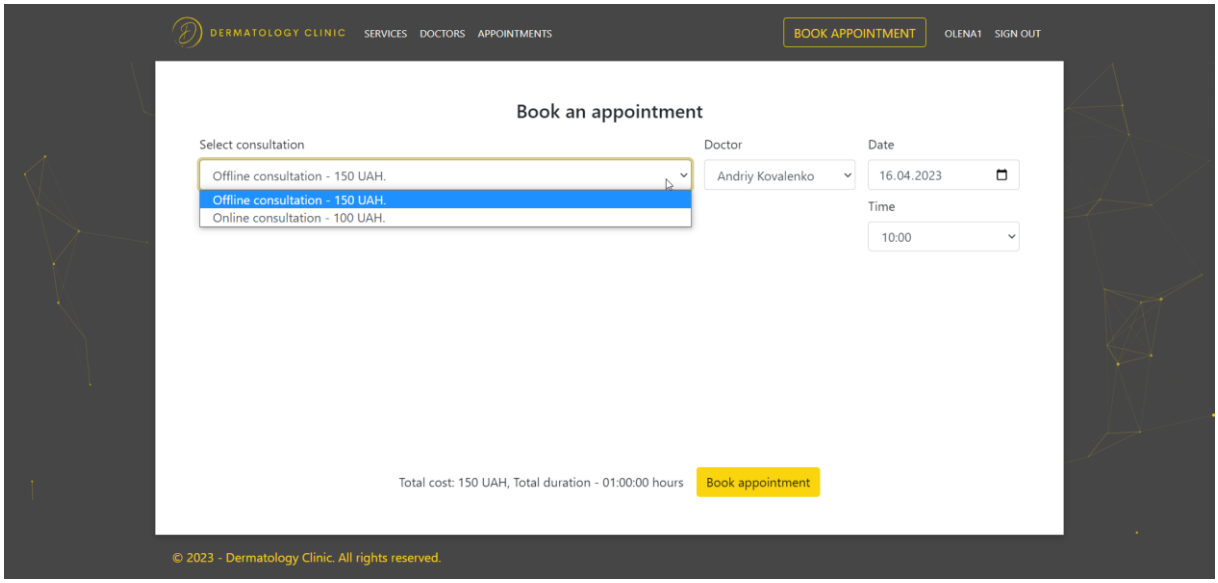


Рисунок А.31 – Дизайн сторінки створення запису (звичайний користувач)

ДОДАТОК Б

Програмний код усіх контролерів

Лістинг Б.1 – Програмний код AccountController

```
public class AccountController : Controller
    {
        private readonly IUserService userService;

        public AccountController(IUserService userService)
        {
            this.userService = userService;
        }

        [HttpGet]
        public IActionResult Register()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Register(RegisterModel
model)
        {
            if (ModelState.IsValid)
            {
                try
                {
                    await this.userService.RegisterAsync(model);
                    return RedirectToAction("Index", "Home");
                }
                catch (Exception e)
                {
                    ModelState.AddModelError("", e.Message);
                }
            }

            return View(model);
        }

        [HttpGet]
        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult> Login(LoginModel model)
        {
            if (ModelState.IsValid)
            {
                try
                {
                    await this.userService.LogInAsync(model);
                    return RedirectToAction("Index", "Home");
                }
                catch (Exception e)
                {
                    ModelState.AddModelError("", e.Message);
                }
            }

            return View(model);
        }

        public async Task<ActionResult> LogOut()
        {
            await this.userService.LogOutAsync();
            return RedirectToAction("Index", "Home");
        }
    }
```

Лістинг Б.2 – Програмний код EmployeesController

```
public class EmployeesController : Controller
    {
        private readonly IEmployeeService employeeService;
        private readonly IHostingEnvironment env;

        public EmployeesController(IEmployeeService employeeService,
IHostingEnvironment env)
        {
            this.employeeService = employeeService;
            this.env = env;
        }

        public async Task<IActionResult> Index()
        {
            return View(await this.employeeService.GetAllAsync());
        }

        [Authorize(Roles = Constants.AdminRole)]
        public IActionResult Create()
        {
            return View();
        }

        [Authorize(Roles = Constants.AdminRole)]
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Name,PhoneNumber,Information,BirthDate,Photo,Id")]
EmployeeViewModel employee)
        {
            if (ModelState.IsValid)
            {
                byte[] data;
                if (employee.Photo == null)
                {
                    data =
System.IO.File.ReadAllBytes(env.WebRootFileProvider.GetFileInfo("img
/avatar.png")?.PhysicalPath);
                }
                else
                {
                    using var memoryStream = new MemoryStream();
                    employee.Photo.CopyTo(memoryStream);
                    data = memoryStream.ToArray();
                }

                var dbEmployee = new Employee()
                {
                    Id = employee.Id,
                    BirthDate = employee.BirthDate,
                    Information = employee.Information,
                    Name = employee.Name,
                    PhoneNumber = employee.PhoneNumber,
                    Photo = data
                };
                await this.employeeService.CreateAsync(dbEmployee);
                return RedirectToAction(nameof(Index));
            }

            return View(employee);
        }

        [Authorize(Roles = Constants.AdminRole)]
        public async Task<IActionResult> Edit(Guid id)
        {
            var employee = await
this.employeeService.GetByIdAsync(id);

            if (employee == null)
            {
                return NotFound();
            }

            var employeeViewModel = new EmployeeViewModel()
            {
```

```
                Id = employee.Id,
                BirthDate = employee.BirthDate,
                Information = employee.Information,
                Name = employee.Name,
                PhoneNumber = employee.PhoneNumber,
                PhotoData = employee.Photo
            };
            return View(employeeViewModel);
        }

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(Guid id,
[Bind("Name,PhoneNumber,Information,BirthDate,Photo,Id")]
EmployeeViewModel employee)
        {
            if (id != employee.Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {

                var dbEmployee = new Employee()
                {
                    Id = employee.Id,
                    BirthDate = employee.BirthDate,
                    Information = employee.Information,
                    Name = employee.Name,
                    PhoneNumber = employee.PhoneNumber
                };

                if (employee.Photo == null)
                {
                    dbEmployee.Photo =
System.IO.File.ReadAllBytes(env.WebRootFileProvider.GetFileInfo("img
/avatar.png")?.PhysicalPath);
                }
                else
                {
                    using var memoryStream = new MemoryStream();
                    employee.Photo.CopyTo(memoryStream);
                    dbEmployee.Photo = memoryStream.ToArray();
                }
                await this.employeeService.UpdateAsync(dbEmployee);
                return RedirectToAction(nameof(Index));
            }
            return View(employee);
        }
```

```
        [Authorize(Roles = Constants.AdminRole)]
        public async Task<IActionResult> Delete(Guid id)
        {
            var employee = await
this.employeeService.GetByIdAsync(id);

            if (employee == null)
            {
                return NotFound();
            }

            return View(employee);
        }

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(Guid id)
        {
            await this.employeeService.DeleteAsync(id);
            return RedirectToAction(nameof(Index));
        }
    }
```

Лістинг Б.3 – Програмний код ServiceTypesController

```
    [Authorize(Roles = Constants.AdminRole)]
    public class ServiceTypesController : Controller
    {
        private readonly IServiceTypeService serviceTypeService;

        public ServiceTypesController(IServiceTypeService
serviceTypeService)
        {
            this.serviceTypeService = serviceTypeService;
        }

        public async Task<IActionResult> Index(string error)
        {
            if (!string.IsNullOrEmpty(error))
            {
                ModelState.AddModelError("", error);
            }

            var types = await this.serviceTypeService.GetAllAsync();

            return View(types);
        }

        [HttpPost]
```

```
        public async Task<IActionResult> Add(ServiceType type)
        {
            if (ModelState.IsValid)
            {
                await this.serviceTypeService.CreateAsync(type);
                return RedirectToAction(nameof(Index));
            }

            return View(type);
        }

        [HttpPost]
        public async Task<IActionResult> Edit(Guid id, ServiceType
type)
        {
            if (id != type.Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                await this.serviceTypeService.UpdateAsync(type);

                return RedirectToAction(nameof(Index));
            }

            return View(type);
        }

        public async Task<IActionResult> Delete(Guid id)
        {
            if (!await
this.serviceTypeService.IsDeletionAvailable(id))
            {
                return RedirectToAction(nameof(Index), new { error =
"Deletion is prohibited, first delete all services in this category"
});
            }

            var type = await
this.serviceTypeService.GetByIdAsync(id);

            if (type == null)
            {
                return NotFound();
            }

            return View(type);
        }
```

```
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(Guid id)
        {
            try
            {
                await this.serviceTypeService.DeleteAsync(id);
                return RedirectToAction(nameof(Index));
            }
            catch (Exception e)
            {
                return RedirectToAction(nameof(Index), new { error =
e.Message });
            }
        }
    }
```

Лістинг Б.4 – Програмний код ServicesController

```
public class ServicesController : Controller
    {
        private readonly IServicesService servicesService;
        private readonly IServiceTypeService serviceTypeService;

        public ServicesController(IServicesService servicesService,
IServiceTypeService serviceTypeService)
        {
            this.servicesService = servicesService;
            this.serviceTypeService = serviceTypeService;
        }

        public async Task<IActionResult> Index()
        {
            ViewBag.ServiceTypes = await
this.serviceTypeService.GetAllAsync();
            var services = await this.servicesService.GetAllAsync();
            return View(services);
        }

        public async Task<IActionResult> Details(Guid id)
        {
            var service = await
this.servicesService.GetByIdAsync(id);
            if (service == null)
            {
                return NotFound();
            }

            return View(service);
        }
```

```
        [Authorize(Roles = Constants.AdminRole)]
        public async Task<IActionResult> Create()
        {
            ViewBag.ServiceTypes = await
this.serviceTypeService.GetAllAsync();
            return View();
        }

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(int duration,
Service service)
        {
            if (ModelState.IsValid)
            {
                service.Duration = TimeSpan.FromMinutes(duration);
                await this.servicesService.CreateAsync(service);
                return RedirectToAction(nameof(Index));
            }

            ViewBag.ServiceTypes = await
this.serviceTypeService.GetAllAsync();
            return View(service);
        }

        [Authorize(Roles = Constants.AdminRole)]
        public async Task<IActionResult> Edit(Guid id)
        {
            var service = await
this.servicesService.GetByIdAsync(id);
            if (service == null)
            {
                return NotFound();
            }

            ViewBag.ServiceTypes = await
this.serviceTypeService.GetAllAsync();
            return View(service);
        }

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(Guid id, int duration,
[Bind("Name,Description,Cost,Duration,ServiceTypeId,Id")] Service
service)
        {
            if (id != service.Id)
            {
                return NotFound();
```

```
            }

            if (ModelState.IsValid)
            {
                service.Duration = TimeSpan.FromMinutes(duration);
                await this.servicesService.UpdateAsync(service);
                return RedirectToAction(nameof(Index));
            }

            ViewBag.ServiceTypes = await
this.serviceTypeService.GetAllAsync();
            return View(service);
        }

        [Authorize(Roles = Constants.AdminRole)]
        public async Task<IActionResult> Delete(Guid id)
        {

            var service = await
this.servicesService.GetByIdAsync(id);
            if (service == null)
            {
                return NotFound();
            }

            return View(service);
        }

        [Authorize(Roles = Constants.AdminRole)]
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(Guid id)
        {
            await this.servicesService.DeleteAsync(id);
            return RedirectToAction(nameof(Index));
        }
    }
```

Лістинг Б.5 – Програмний код ReceptionsController

```
public class ReceptionsController : Controller
    {
        private readonly IReceptionService receptionService;
        private readonly IEmployeeService employeeService;
        private readonly IUserService userService;
        private readonly IServicesService servicesService;
        private readonly IServiceTypeService serviceTypeService;

        public ReceptionsController(IReceptionService
receptionService,
            IEmployeeService employeeService,
```

```
            IUserService userService,
            IServicesService servicesService,
            IServiceTypeService serviceTypeService)
        {
            this.receptionService = receptionService;
            this.employeeService = employeeService;
            this.userService = userService;
            this.servicesService = servicesService;
            this.serviceTypeService = serviceTypeService;
        }

        [Authorize]
        public async Task<IActionResult> Index()
        {
            var receptions = new List<Reception>();
            if (User.IsInRole(Constants.AdminRole))
            {
                receptions = await
this.receptionService.GetAllCompletedAsync();
            }
            else
            {
                receptions = await
this.receptionService.GetCompletedByUserAsync(this.userService.GetCu
rrentUserId());
            }


            return View(receptions);
        }

        [Authorize]
        public async Task<IActionResult> AddService(Guid serviceId)
        {
            ViewBag.Id = serviceId;
            Console.WriteLine("33");
            var reception = await
this.receptionService.AddServiceAsync(serviceId);
            await this.SetViewBagAsync(reception);

            return View(nameof(Create), reception);
        }

        [Authorize]
        public async Task<IActionResult> RemoveService(Guid
serviceId)
        {
            var reception = await
this.receptionService.RemoveServiceAsync(serviceId);
            await this.SetViewBagAsync(reception);
```

```
            return View(nameof(Create), reception);
        }

        [Authorize]
        public async Task<IActionResult> SelectEmployee(Guid
employeeId)
        {
            var reception = await
this.receptionService.GetCurrentReceptionAsync();
            reception.EmployeeId = employeeId;
            reception.Employee = await
this.employeeService.GetByIdAsync(employeeId);
            await this.receptionService.UpdateAsync(reception);
            await this.SetViewBagAsync(reception);

            return View(nameof(Create), reception);
        }

        [Authorize]
        public async Task<IActionResult> SelectDate(DateTime date)
        {
            var reception = await
this.receptionService.GetCurrentReceptionAsync();
            reception.ReceptionDateTime = date.Date +
reception.ReceptionDateTime.TimeOfDay;
            await this.receptionService.UpdateAsync(reception);
            await this.SetViewBagAsync(reception);

            return View(nameof(Create), reception);
        }

        [Authorize]
        public async Task<IActionResult> SelectTime(DateTime time)
        {
            var reception = await
this.receptionService.GetCurrentReceptionAsync();
            reception.ReceptionDateTime =
reception.ReceptionDateTime.Date + time.TimeOfDay;
            await this.receptionService.UpdateAsync(reception);
            await this.SetViewBagAsync(reception);

            return View(nameof(Create), reception);
        }

        [Authorize]
        public async Task<IActionResult> Create()
        {
            try
            {
                var reception = await
this.receptionService.GetCurrentReceptionAsync();
```

```
                await this.SetViewBagAsync(reception);
                if (ViewBag.Id == Guid.Empty)
                {
                    await AddService((ViewBag.UserServices as
IEnumerable<Service>).FirstOrDefault().Id);
                }
                return View(reception);
            }
            catch (Exception e)
            {
                return RedirectToAction(nameof(Index),
nameof(HomeController), new { error = e.Message });
            }
        }

        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(Guid userId)
        {
            var reception = await
this.receptionService.GetCurrentReceptionAsync();
            if (!reception.Services.Any())
            {
                ModelState.AddModelError("", "Select service");
                await this.SetViewBagAsync(reception);
                return View(reception);
            }

            if (User.IsInRole(Constants.AdminRole))
            {
                reception.UserId = userId;
                reception.User = await
this.userService.GetByIdAsync(userId);
                await this.receptionService.UpdateAsync(reception);
            }

            reception.Status = ReceptionStatus.Completed;
            await this.receptionService.UpdateAsync(reception);

            return RedirectToAction(nameof(Index));
        }

        [Authorize]
        public async Task<IActionResult> Confirm(Guid id, string
error)
        {
            if (!string.IsNullOrEmpty(error))
            {
                ModelState.AddModelError("", error);
            }
```

```
            var reception = await
this.receptionService.GetByIdAsync(id);
            if (reception == null)
            {
                return NotFound();
            }

            if (User.IsInRole(Constants.AdminRole))
            {
                await this.receptionService.ConfirmAsync(reception);
                return RedirectToAction(nameof(Index));
            }

            return View(reception);
        }

        [Authorize]
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> ConfirmationConfirmed(Guid
id)
        {
            var reception = await
this.receptionService.GetByIdAsync(id);
            await this.receptionService.ConfirmAsync(reception);
            return RedirectToAction(nameof(Index));
        }

        [Authorize]
        public async Task<IActionResult> Delete(Guid id)
        {
            var reception = await
this.receptionService.GetByIdAsync(id);
            if (reception == null)
            {
                return NotFound();
            }

            return View(reception);
        }

        [Authorize]
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(Guid id)
        {
            await this.receptionService.DeleteAsync(id);
            return RedirectToAction(nameof(Index));
        }
```

```
        private async Task SetViewBagAsync(Reception reception)
        {
            ViewBag.Id = reception.Services.FirstOrDefault()?.Id ??
Guid.Empty;
            ViewBag.Employees = new SelectList(await
this.employeeService.GetAllAsync(), "Id", "Name",
reception.EmployeeId);
            ViewBag.Users = new SelectList(await
this.userService.GetAllClientsAsync(), "Id", "Name",
reception.UserId);
            ViewBag.AllServices = await
this.servicesService.GetAllAsync();
            ViewBag.UserServices = (ViewBag.AllServices as
List<Service>).Where(x => x.ServiceType.Name == "Dermatologist
consultation");
            ViewBag.AvailableTime = new SelectList(await
this.receptionService.GetAvailableTimeAsync(reception.ReceptionDateT
ime, TimeSpan.FromMinutes(reception.Services.Sum(x =>
x.Duration.TotalMinutes)), reception.EmployeeId),
reception.ReceptionDateTime.ToString("H:mm"));
        }
    }
```

Лістинг Б.6 – Програмний код HomeController

```
public class HomeController : Controller
    {
        private readonly IServicesService servicesService;
        private readonly IServiceTypeService serviceTypeService;
        private readonly IEmployeeService employeeService;
        private readonly IHostingEnvironment env;

        public HomeController(IServicesService servicesService,
            IServiceTypeService serviceTypeService,
            IEmployeeService employeeService,
            IHostingEnvironment env)
        {
            this.servicesService = servicesService;
            this.serviceTypeService = serviceTypeService;
            this.employeeService = employeeService;
            this.env = env;
        }

        public async Task<IActionResult> Index(string error)
        {
            if (!string.IsNullOrEmpty(error))
            {
                ModelState.AddModelError("", error);
            }
```

```
            ViewBag.ServiceTypes = await
this.serviceTypeService.GetAllAsync();
            ViewBag.Employees = await
this.employeeService.GetAllAsync();
            var services = await this.servicesService.GetAllAsync();
            ViewBag.Image =
System.IO.File.ReadAllBytes(env.WebRootFileProvider.GetFileInfo("img
/mole3.png")?.PhysicalPath);

            return View(services);
        }

        [HttpPost]
        public async Task<ActionResult> Img(IFormFile imageFile)
        {
            if (imageFile == null || imageFile.Length == 0)
            {
                return BadRequest();
            }

            using var memoryStream = new MemoryStream();
            imageFile.CopyTo(memoryStream);

            var data = memoryStream.ToArray();
            ViewBag.Diagnosis = this.PredictDiagnosis(data);
            ViewBag.Image = data;
            ViewBag.ServiceTypes = await
this.serviceTypeService.GetAllAsync();
            ViewBag.Employees = await
this.employeeService.GetAllAsync();
            ViewBag.Scroll = "analisys";
            var services = await this.servicesService.GetAllAsync();

            return View(nameof(Index), services);
        }

        private Diagnosis PredictDiagnosis(byte[] data)
        {
            var projectDirectory =
Path.GetFullPath(Path.Combine(AppContext.BaseDirectory,
"../../../wwwroot/"));
            var workspaceRelativePath =
Path.Combine(projectDirectory, "workspace");
            var assetsRelativePath = Path.Combine(projectDirectory,
"assets");

            MLContext mlContext = new();

            var images = LoadImagesFromDirectory(folder:
assetsRelativePath, useFolderNameAsLabel: true);
```

```
            var imageData =
mlContext.Data.LoadFromEnumerable(images);

            var shuffledData =
mlContext.Data.ShuffleRows(imageData);

            var preprocessingPipeline =
mlContext.Transforms.Conversion.MapValueToKey(
                    inputColumnName: "Label",
                    outputColumnName: "LabelAsKey")
                .Append(mlContext.Transforms.LoadRawImageBytes(
                    outputColumnName: "Image",
                    imageFolder: assetsRelativePath,
                    inputColumnName: "ImagePath"));

            var preProcessedData =
preprocessingPipeline.Fit(shuffledData).Transform(shuffledData);

            var trainSplit = mlContext.Data.TrainTestSplit(data:
preProcessedData, testFraction: 0.3);

            var trainSet = trainSplit.TrainSet;
            var validationSet = trainSplit.TestSet;

            var classifierOptions = new
ImageClassificationTrainer.Options()
            {
                FeatureColumnName = "Image",
                LabelColumnName = "LabelAsKey",
                ValidationSet = validationSet,
                Arch =
ImageClassificationTrainer.Architecture.ResnetV250,
                MetricsCallback = (metrics) =>
Console.WriteLine(metrics),
                TestOnTrainSet = false,
                ReuseTrainSetBottleneckCachedValues = true,
                ReuseValidationSetBottleneckCachedValues = true,
                Epoch = 1000,
                WorkspacePath = workspaceRelativePath
            };

            var trainingPipeline =
mlContext.MulticlassClassification.Trainers.ImageClassification(clas
sifierOptions)

.Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabe
l"));

            var trainedModel = trainingPipeline.Fit(trainSet);

            // Evaluate the accuracy of the model
```

```
            //var testDataWithKey =
mlContext.Transforms.Conversion.MapValueToKey(
            //    inputColumnName: "Label",
            //    outputColumnName: "LabelAsKey")
            //    .Fit(preProcessedData)
            //    .Transform(preProcessedData);

            //var predictions =
trainedModel.Transform(testDataWithKey);
            //var metrics =
mlContext.MulticlassClassification.Evaluate(predictions,
labelColumnName: "LabelAsKey");
            //Console.WriteLine($"Test set accuracy:
{metrics.MacroAccuracy}");


            var prediction = ClassifyImage(mlContext, data,
trainedModel);

            var diagnosis = Diagnosis.Undefined;
            if (prediction.PredictedLabel == "benign" &&
prediction.Score[0] > 0.8)
            {
                diagnosis = Diagnosis.Benign;
            }
            else if (prediction.PredictedLabel == "malignant" &&
prediction.Score[1] > 0.8)
            {
                diagnosis = Diagnosis.Malignant;
            }

            return diagnosis;
        }

        private static ModelOutput ClassifyImage(MLContext
mlContext, byte[] data, ITransformer trainedModel)
        {
            PredictionEngine<ModelInput, ModelOutput>
predictionEngine =
mlContext.Model.CreatePredictionEngine<ModelInput,
ModelOutput>(trainedModel);
            ModelInput image = new() { Image = data };

            return predictionEngine.Predict(image);
        }

        private static IEnumerable<ImageData>
LoadImagesFromDirectory(string folder, bool useFolderNameAsLabel =
true)
        {
            var files = Directory.GetFiles(folder, "*",
```

```
                searchOption: SearchOption.AllDirectories);

            foreach (var file in files)
            {
                if ((Path.GetExtension(file) != ".jpg") &&
(Path.GetExtension(file) != ".png"))
                    continue;

                var label = Path.GetFileName(file);

                if (useFolderNameAsLabel)
                    label = Directory.GetParent(file).Name;
                else
                {
                    for (int index = 0; index < label.Length;
index++)
                    {
                        if (!char.IsLetter(label[index]))
                        {
                            label = label.Substring(0, index);
                            break;
                        }
                    }
                }

                yield return new ImageData()
                {
                    ImagePath = file,
                    Label = label
                };
            }
        }
    }
```

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	117 с.
	Інші документи	
2.	Презентаційні матеріали	плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Кондратюк О.С.		16.05.23	Інтелектуальна дерматоскопічна система виявлення меланом
Перевірив		Аврунін О.Г.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-21-1	122
ХНУРЕ кафедра ШІ	