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)

Застосування глибоких згорткових мереж
для автоматичного аналізу медичних зображень
(тема)

Виконав:
здобувач другого року навчання,
групи ДСм-24-1
Михайлов С.В.
(прізвище, ініціали)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Науки про дані (Data Science)
(повна назва спеціалізації)
Керівник доц. Дейнеко А.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Лариса ЧАЛА
(підпис) (прізвище, ініціали)

2025 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-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Науки про дані (Data Science)
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Михайлову Сергію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Застосування глибоких згорткових мереж для автоматичного аналізу медичних зображень

затверджена наказом університету від 24 листопада 20 25 р. № 1057Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17 грудня 20 25 р.
3. Вихідні дані до роботи Науково-технічні публікації, Python documentation, TensorFlow documentation, Django documentation, дані інтернет джерел, проекти з використанням глибоких нейронних мереж для аналізу зображень

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Проектування додатку
3) Вибір та опис технологій реалізації системи аналізу
4) Опис реалізованого проекту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	24.11.2025	виконано
2	Аналіз предметної галузі	26.11.2025	виконано
3	Проектування додатку	28.11.2025	виконано
4	Вибір та опис технологій реалізації системи	30.11.2025	виконано
	аналізу		
5	Реалізація проекту	05.12.2025	виконано
6	Написання пояснювальної записки	08.12.2025	виконано
7	Перевірка на академічний плагіат	10.12.2025	виконано
8	Нормоконтроль	11.12.2025	виконано
9	Підготовка презентації та доповіді	13.12.2025	виконано
10	Попередній захист	14.12.2025	виконано
11	Рецензування	16.12.2025	виконано
12	Захист перед ЕК	17.12.2025	виконано

Дата видачі завдання 24 листопада 20 25 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Дейнеко А.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 69 с., 23 рис., 4 табл., 2 дод., 23 джерела.

БАЗА ДАНИХ, ВЕБ-ДОДАТОК, ГИБРИДНА МОДЕЛЬ, ГЛИБИННЕ НАВЧАННЯ, ДЕТЕКЦІЯ ПАТОЛОГІЙ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, КЛАСИФІКАЦІЯ, КТ, МЕДИЧНІ ЗОБРАЖЕННЯ, МРТ, РЕНТГЕН, СЕГМЕНТАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, DJANGO, EFFICIENTNET, POSTGRESQL, PYTHON, RESNET, TENSORFLOW, U-NET.

Об’єкт дослідження – процес автоматизованого аналізу медичних зображень із застосуванням методів глибокого навчання для класифікації та сегментації та виявлення патологій.

Предмет дослідження – глибокі згорткові нейронні мережі, архітектурні підходи до їх побудови, оптимізації та адаптації, а також методи інтеграції цих моделей у веб-додаток для створення повноцінної системи інтелектуального аналізу медичних зображень.

Мета роботи – підвищення точності, швидкодії та ефективності автоматизованого аналізу медичних зображень шляхом розробки та оптимізації гібридної моделі глибокого навчання, яка поєднує можливості різних типів нейронних мереж, та створення веб-застосунку, що забезпечить зручну взаємодію користувачів із системою. Досягнення поставленої мети передбачає не лише побудову нейронної мережі, а й повну інтеграцію моделі у веб-додаток, включно з модулем передобробки даних, серверною частиною, базою даних, інтерфейсом користувача та механізмами зберігання результатів аналізу.

ABSTRACT

Master’s thesis contains: 69 pp., 23 fig., 4 tabl., 2 ann., 23 references.

ARTIFICIAL INTELLIGENCE, CLASSIFICATION, CONVOLUTIONAL NEURAL NETWORK, CT, DATABASE, DEEP LEARNING, DETECTION OF PATHOLOGIES, DJANGO, EFFICIENTNET, HYBRID MODEL, MEDICAL IMAGING, MRI, POSTGRESQL, PYTHON, RESNET, SEGMENTATION, TENSORFLOW, U-NET, WEB APPLICATION, X-RAY.

The object of the study is the process of automated analysis of medical images using deep learning methods for classification, segmentation, and pathology detection.

The subject of the study is deep convolutional neural networks, architectural approaches to their construction, optimization, and adaptation, as well as methods for integrating such models into a web application to build a complete system for intelligent medical image analysis.

The purpose of the thesis is to improve the accuracy, performance, and efficiency of automated medical image analysis by developing and optimizing a hybrid deep learning model that combines the capabilities of several types of neural networks, and by creating a web application that ensures convenient user interaction with the system. Achieving this goal involves not only designing the neural network itself but also fully integrating the model into the web application, including the data preprocessing module, server-side logic, database, user interface, and mechanisms for storing and managing analysis results.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі 10
1.1 Опис предметної галузі 10
1.1.1 Актуальність проблеми 11
1.1.2 Історичний огляд розвитку технологій 11
1.2 Аналіз типів глибоких згорткових мереж 12
1.2.1 Багатошарові згорткові мережі 13
1.2.2 Залишкові мережі 14
1.2.3 Мережі для сегментації 15
1.2.4 Інші типи мереж 17
1.3 Порівняльний аналіз розглянутих мереж 18
1.4 Огляд датасетів для медичних зображень 19
1.4.1 Датасети для класифікації патологій 19
1.4.2 Датасети для сегментації органів та патологій 21
1.4.3 Датасети для детекції та мультимодальних задач 22
1.4.4 Порівняльний аналіз датасетів 22
1.5 Постановка задачі 23
2 Проектування додатку 25
2.1 Функціональні вимоги до додатку 25
2.2 Вибір архітектури додатку 26
2.2.1 Монолітна архітектура 26
2.2.2 Мікросервісна архітектура 29
2.2.3 Шарова архітектура 31
2.2.4 MVC-архітектура 33
2.2.5 Порівняльний аналіз архітектурних шаблонів додатку 35
3 Вибір та опис технологій реалізації системи аналізу 37
3.1 Мова програмування Python 37

3.2 Фреймворк побудови додатку 39
3.3 Бібліотека реалізації моделі глибинного навчання 42
3.4 Система управління базами даних postgresql..... 44
3.5 Мови та інструменти фронтенд-розробки..... 46
3.5.1 HTML 46
3.5.2 CSS для стилізації та адаптивного дизайну 49
3.5.3 javascript для динаміки та інтерактивності..... 51
4 Опис реалізованого проекту..... 54
Висновки 62
Перелік джерел посилання..... 63
Додаток А Програмна реалізація моделі нейронної мережі..... 66
Додаток Б Відомість кваліфікаційної роботи..... 69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

КТ – комп’ютерна томографія;

МРТ – магнітно-резонансна томографія;

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – програмний інтерфейс застосунків;

CNN – Convolutional Neural Network – згорткова нейронна мережа;

CPU – Central Processing Unit – центральний процесор;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

CSRF – Cross-Site Request Forgery – міжсайтове підроблення запиту;

GPU – Graphics Processing Unit – графічний процесор;

HTML – HyperText Markup Language – мова розмітки гіпертексту;

ID – Identifier – ідентифікатор;

ML – Machine Learning – машинне навчання;

MVC – Model–View–Controller – модель–подання–контролер;

ORM – Object Relational Mapping – об’єктно-реляційне відображення;

REST – Representational State Transfer – передавання стану представлення;

XSS – Cross-Site Scripting – міжсайтове виконання скриптів.

ВСТУП

У сучасному світі медична діагностика значною мірою залежить від аналізу зображень, отриманих за допомогою різних технологій візуалізації, таких як рентгенографія, комп'ютерна томографія, магнітно-резонансна томографія та ультразвукове дослідження. Автоматичний аналіз медичних зображень за допомогою глибоких згорткових нейронних мереж є перспективним напрямком, що дозволяє підвищити точність, швидкість та ефективність діагностики. Традиційні методи аналізу, такі як ручний огляд лікарями, мають суттєві обмеження: суб'єктивність оцінки, залежність від досвіду фахівця, витрати часу та ресурсів. За даними Всесвітньої організації охорони здоров'я, щорічно проводиться мільярди медичних сканувань, і брак кваліфікованих радіологів призводить до затримок у діагностиці та зростання помилок.

Актуальність теми зумовлена стрімким зростанням обсягів медичних даних: за прогнозами, галузь охорони здоров'я генерує близько 30% глобальних даних. Пандемії, як COVID-19, підкреслили необхідність швидкої обробки зображень для масового скринінгу, де традиційні методи не справляються з обсягами. Застосування штучного інтелекту у медичному аналізі зображень дозволяє не тільки прискорити діагностику, але й сприяти персоналізованому лікуванню, зменшуючи витрати та покращуючи результати для пацієнтів. Однак, існуючі рішення часто стикаються з проблемами, як низька точність на шумових даних, потреба в великих анотованих датасетах та етичні питання конфіденційності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

У сучасному світі медична діагностика значною мірою залежить від аналізу зображень, отриманих за допомогою різних технологій візуалізації, таких як рентгенографія, комп'ютерна томографія (КТ), магнітно-резонансна томографія (МРТ) та ультразвукове дослідження. Автоматичний аналіз медичних зображень за допомогою глибоких згорткових нейронних мереж (CNN) є перспективним напрямком, що дозволяє підвищити точність, швидкість та ефективність діагностики.

1.1 Опис предметної галузі

Медичний аналіз зображень включає процеси класифікації, сегментації та детекції аномалій на зображеннях органів та тканин. Традиційні методи, такі як ручний огляд лікарями, мають обмеження: суб'єктивність оцінки, залежність від досвіду фахівця та витрати часу. За даними Всесвітньої організації охорони здоров'я (ВОЗ), щорічно проводиться мільярди медичних сканувань, і брак кваліфікованих радіологів призводить до затримок у діагностиці.

Глибокі згорткові мережі дозволяють автоматизувати ці процеси, виділяючи ознаки автоматично через шари згорток та пулінгу. Застосування CNN у медицині включає виявлення пухлин на МРТ, класифікацію пневмонії на рентгенах грудної клітки та сегментацію органів на КТ-знімках. Наприклад, під час пандемії COVID-19 моделі на базі CNN, такі як COVID-Net, досягли точності понад 90% у виявленні захворювання на рентгенах.

Переваги використання CNN: висока точність (до 98% у деяких задачах), швидкість обробки та можливість інтеграції з телемедициною. Недоліки: потреба в великих анотованих датасетах, ризик перенавчання та етичні питання конфіденційності даних пацієнтів.

1.1.1 Актуальність проблеми

Актуальність автоматичного аналізу медичних зображень зумовлена стрімким зростанням обсягів даних: за прогнозами, галузь охорони здоров'я генерує близько 30% глобальних даних, з очікуваним річним темпом зростання 36% до 2025 року [1]. Щорічно у світі проводиться понад 3,6 мільярда діагностичних радіологічних досліджень, з яких рентгенівські знімки становлять найбільшу частку, що призводить до перевантаження систем охорони здоров'я та зростання помилок через брак фахівців. В Україні, за даними МОЗ, щорічно проводиться понад 10 мільйонів рентгенів, і автоматизація може скоротити час діагностики на 50%, зменшивши навантаження на радіологів та підвищивши точність виявлення патологій. У країнах з обмеженими ресурсами ШІ може зменшити навантаження на систему охорони здоров'я, покращити доступність медичної допомоги в віддалених регіонах та знизити помилки, пов'язані з людським фактором. Крім того, пандемії, як COVID-19, підкреслили необхідність швидкої обробки зображень для масового скринінгу, де традиційні методи не справляються з обсягами. Автоматизація не тільки прискорює діагностику, але й сприяє персоналізованому лікуванню, зменшуючи витрати та покращуючи результати для пацієнтів з хронічними захворюваннями, такими як рак чи серцево-судинні патології.

1.1.2 Історичний огляд розвитку технологій

Розвиток CNN у медичному аналізі зображень розпочався з середини XX століття, коли з'явилися перші комп'ютерні системи для обробки рентгенівських знімків, але справжній прорив стався у 1960–1970-х роках. У 1969 році Куніхіко Фукусіма опублікував виправлені лінійні одиниці або ReLU, які стали основою сучасних мереж, а у 1979 році він опублікував базову архітектуру, подібну до CNN, для розпізнавання візуальних

патернів [2]. У 1980-х роках Ян ЛеКун розробив концепцію згорткових мереж, а у 1989 році представив першу CNN для розпізнавання рукописних символів (LeNet), яка була адаптована для медичних задач, таких як аналіз зображень органів [3]. У 2012 році перемога AlexNet на конкурсі ImageNet стимулювала адаптацію для медичних задач, таких як класифікація рентгенів. З 2015 року архітектури як U-Net стали стандартом для сегментації біомедичних зображень, дозволяючи виділяти пухлини на МРТ з високою точністю. Сучасні тенденції включають трансферне навчання та гібридні моделі Цей еволюційний шлях від базових мереж до складних архітектур підкреслив роль CNN у підвищенні ефективності діагностики, особливо в умовах зростання даних і дефіциту фахівців.

1.2 Аналіз типів глибоких згорткових мереж

Для аналізу медичних зображень використовуються різні архітектури глибоких згорткових нейронних мереж (CNN), які спеціально адаптовані до унікальної специфіки медичних даних, таких як низька роздільна здатність, високий рівень шуму, варіабельність анатомії та дисбаланс класів. Еволюція CNN від базових архітектур, таких як LeNet, до сучасних, як ResNet чи EfficientNet, дозволила досягти значного прогресу в медичній візуалізації. Вибір архітектури залежить від типу задачі: для класифікації патологій ефективні глибокі моделі з залишковими з'єднаннями, тоді як для сегментації переважно застосовуються енкодер-декодер структури. Загалом, систематичні огляди показують, що адаптовані CNN перевершують традиційні методи обробки зображень, такі як порогові фільтри чи класичні алгоритми машинного навчання, у швидкості та точності.

1.2.1 Багатошарові згорткові мережі

VGGNet, запропонована у 2014 році є однією з перших глибоких згорткових мереж, що продемонструвала ефективність збільшення глибини для покращення точності розпізнавання. Вона складається з послідовних згорткових шарів з маленькими фільтрами розміром 3x3, що дозволяє ефективно захоплювати локальні ознаки без значного зростання обчислювальної складності на шар (рисунок 1.1).

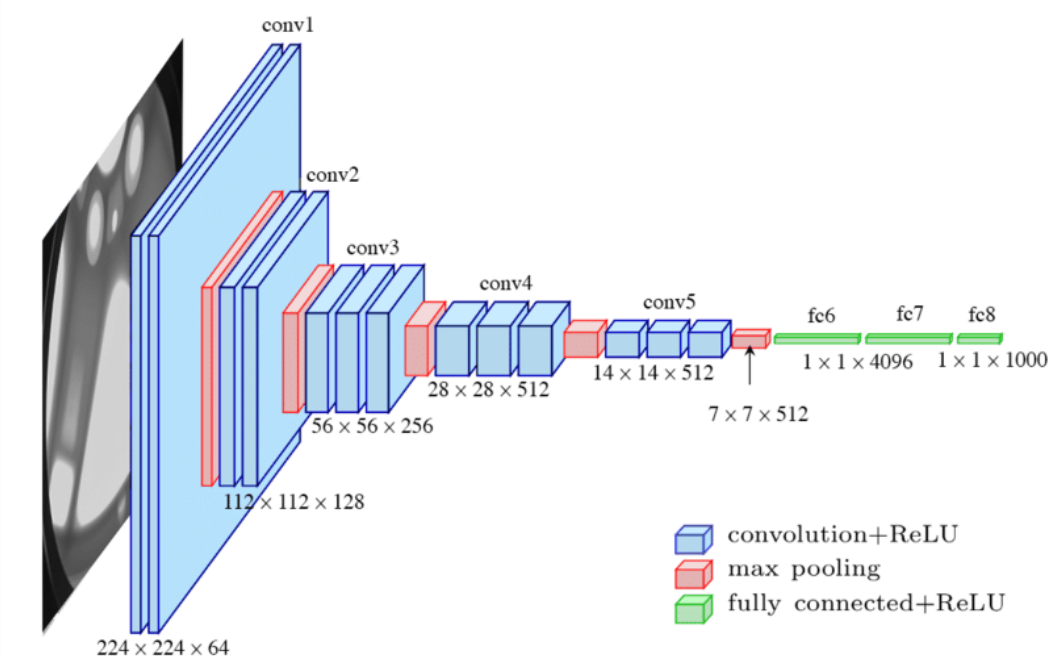

Рисунок 1.1 – Архітектура VGG-16

Основна ідея полягає в тому, що кілька маленьких фільтрів можуть замінити більші, зменшуючи кількість параметрів порівняно з традиційними мережами з великими фільтрами. Наприклад, два шари 3x3 еквівалентні одному 5x5, але з меншою кількістю параметрів. Переваги VGG включають простоту реалізації, що робить її ідеальною для початківців у глибокому навчанні, та ефективність у витягуванні базових

ознак, що особливо корисно для задач з високою деталізацією зображень. У медичному контексті VGG часто використовується для класифікації зображень, наприклад, для виявлення меланоми на дерматоскопічних знімках з великою точністю, де модель добре справляється з текстурними патернами шкіри. Недоліки: велика кількість параметрів, що призводить до високої обчислювальної складності та вимог до пам'яті, а також схильність до перенавчання на малих датасетах, типових для медичної сфери, де анотації дорогі та рідкісні.

До переваг VGG відносяться універсальність і легкість у модифікації, але в медичних задачах вона поступається глибшим мережам через обмежену глибину та відсутність механізмів для боротьби зі зникненням градієнтів [4].

1.2.2 Залишкові мережі

ResNet, розроблена у 2015 році, вводить залишкові блоки для вирішення проблеми зникнення градієнтів у глибоких мережах, дозволяючи створювати архітектури з понад 100 шарами без деградації продуктивності. Залишковий блок додає вхідний сигнал безпосередньо до виходу шару через з'єднання, перетворюючи навчання на оцінку залишкової функції (рисунок 1.2). Це полегшує оптимізацію, оскільки мережа вчиться коригувати ідентичність, а не вчити з нуля.

Серед переваг можна зазначити: можливість створення надзвичайно глибоких мереж, висока точність на задачах класифікації завдяки кращому поширенню градієнтів, що робить її стійкою до перетренування. У медицині ResNet широко застосовується для аналізу рентгенів грудної клітки, досягаючи точності 95% у виявленні пневмонії, наприклад, у моделях на базі CheXNet, де вона ефективно витягує ієрархічні ознаки від простих країв до складних патологій. Крім того, ResNet ідеально підходить для трансферного навчання, де попередньо навчені моделі на ImageNet

адаптуються до медичних датасетів, зменшуючи потребу в великих обсягах даних і скорочуючи час навчання.

Як недоліки можна виділити потрібність в потужних GPU для навчання через глибину, меншу ефективність у сегментації порівняно з U-Net через втрату просторової інформації в глибоких шарах, а також потенційна надмірність параметрів у деяких конфігураціях.

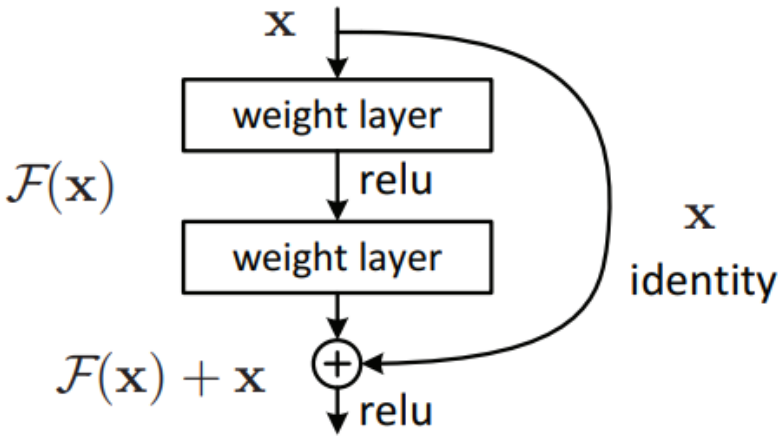

Рисунок 1.2 – Залишковий блок ResNet

ResNet ефективний для трансферного навчання, де попередньо навчені моделі на ImageNet адаптуються до медичних датасетів, зменшуючи потребу в даних, і часто комбінується з іншими техніками для гібридних задач [5].

1.2.3 Мережі для сегментації

U-Net – це мережа, створена спеціально для обробки біомедичних зображень у 2015 році. Вона має симетричну структуру з двох частин: енкодера та декодера, з'єднаних пропусками (рисунок 1.3). Ці пропуски допомагають зберегти детальну інформацію з початкових шарів для точного відновлення зображення на виході.

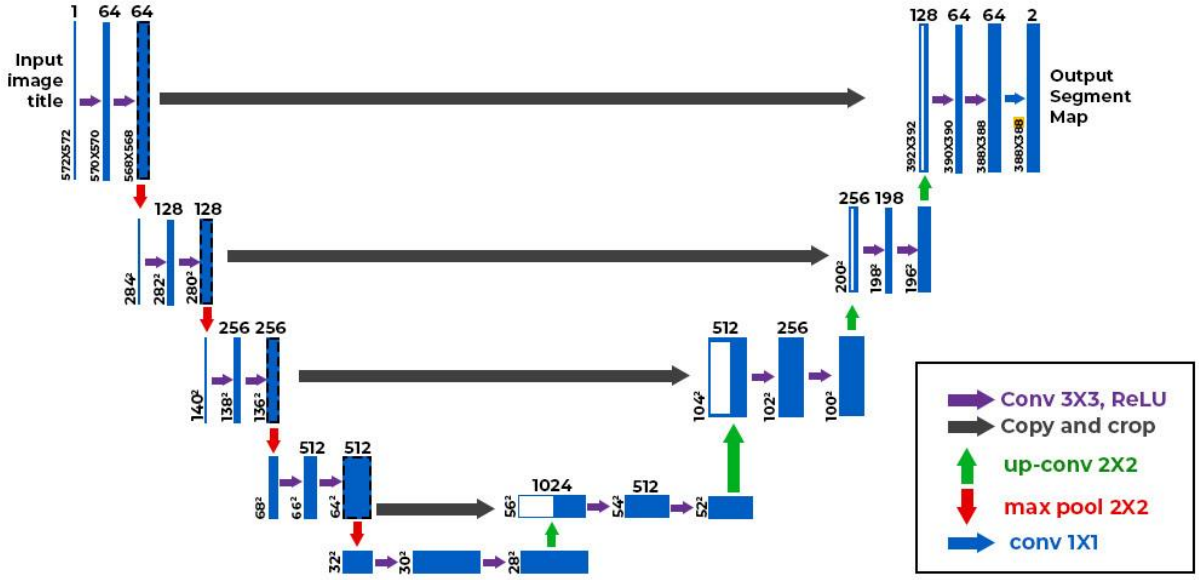

Рисунок 1.3 – Архітектура U-Net

Енкодер витягує ключові ознаки за допомогою згорток і пулінгу, зменшуючи розмір даних, а декодер повертає оригінальну роздільну здатність через розширення і згортки, поєднуючи контекст з локальними деталями. Перевагами таких мереж є висока точність сегментації (IoU до 0.92), добре працює з невеликими датасетами завдяки аугментації та пропускам, що робить її підходящою для медичних задач з обмеженою кількістю анотацій. U-NET застосовується для виділення пухлин на МРТ мозку. Модель точно визначає межі патологій, допомагаючи планувати операції. Крім того, U-Net стійка до змін у даних, як шум на медичних знімках, і легко адаптується для 3D-зображень. Однак така мережа повільніша за ResNet через процес розширення, чутлива до налаштувань, що потребує ретельного підбору, та може вимагати багато ресурсів для роботи в реальному часі.

U-Net часто комбінується з іншими мережами, наприклад, з ResNet як backbone, для гібридних задач класифікації та сегментації, що підвищує загальну ефективність у клінічних сценаріях [6].

1.2.4 Інші типи мереж

Окрім класичних архітектур, таких як VGG, ResNet та U-Net, у аналізі медичних зображень активно застосовуються оптимізовані мережі, спрямовані на баланс між точністю, ефективністю та ресурсоємністю, що критично важливо для мобільних або ресурсообмежених медичних систем. Однією з ключових є EfficientNet, запропонована у 2019 році, яка оптимізує глибину, ширину та роздільну здатність мережі за методу, що рівномірно масштабує всі виміри моделі [7]. Це дозволяє досягти кращої точності з меншою кількістю параметрів порівняно з традиційними мережами: наприклад, EfficientNet-B0 має лише 5,3 млн параметрів, але перевершує ResNet-50 на 4-6% у точності на ImageNet. У медицині EfficientNet застосовується для мобільних додатків діагностики, таких як класифікація рентгенів грудної клітки у виявленні пневмонії чи COVID-19, де її ефективність на обмежених GPU робить можливим розгортання на смартфонах для телемедицини в віддалених регіонах. Переваги включають оптимальний баланс обчислювальних операцій і точності, що зменшує час, та стійкість до шуму в медичних даних.

Іншою важливою архітектурою є DenseNet, представлена у 2016 році, яка використовує щільні з'єднання, де кожен шар отримує вхід від усіх попередніх шарів через конкатенацію, а не додавання, як у ResNet [8]. Це сприяє повторному використанню ознак, зменшуючи кількість параметрів і покращуючи потік градієнтів. У медичній сфері DenseNet ефективна для задач з обмеженими ресурсами, таких як сегментація на малих датасетах, наприклад, виявлення пухлин на КТ з точністю 94%, де щільні з'єднання допомагають боротися з перенавчанням. Переваги: менша кількість параметрів, краща узагальненість на шумних даних та ефективність у навчанні для медичних датасетів з дисбалансом класів. Недоліки: висока споживання пам'яті через конкатенацію, що ускладнює навчання глибоких

версій, та складність у реалізації для чистої сегментації без модифікацій, таких як додавання декодера.

1.3 Порівняльний аналіз розглянутих мереж

Для вибору оптимальної архітектури проведено порівняльний аналіз за ключовими критеріями:

– точність;

– обчислювальна складність;

– застосування в медицині.

Порівняльний аналіз наведено у таблиці 1.1

Таблиця 1.1 – Порівняння архітектур CNN для аналізу медичних зображень

Архітектура	Кількість шарів	Точність (приклад, %)	Обчислювальна складність	Застосування в медицині	Переваги	Недоліки
VGG-16	16	91 (класифікація меланоми)	Висока	Класифікація зображень		
ResNet-50	50	95 (виявлення пневмонії)	Середня	Класифікація та детекція		
U-Net	~31	92 (сегментація пухлин)	Висока	Сегментація		
EfficientNet-B0	Варіюється	96 (ренгени)	Низька	Мобільна діагностика		
DenseNet-121	121	94 (КТ-аналіз)	Середня	Класифікація та сегментація		

З аналізу видно, що для задач класифікації оптимальним є ResNet, для сегментації – U-Net. Гібридні підходи, такі як U-Net з ResNet, поєднують переваги. EfficientNet підходить для ресурсообмежених середовищ, а DenseNet – для задач з обмеженими даними.

Порівняння архітектур CNN за метриками на датасеті Chest X-Ray наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння за метриками на датасеті Chest X-Ray

Архітектура	Точність (%)	F1-score	IoU (сегментація)	Час інференсу (мс)
VGG-16	89	0.88	–	50
ResNet-50	95	0.94	–	30
U-Net	–	–	0.90	45
EfficientNet	96	0.95	–	20
DenseNet	94	0.93	0.88	35

1.4 Огляд датасетів для медичних зображень

Для ефективного застосування глибоких згорткових мереж (CNN) в автоматичному аналізі медичних зображень критично важливим є наявність якісних, анотованих датасетів. Ці датасети слугують основою для навчання, валідації та тестування моделей, дозволяючи моделям узагальнювати знання на реальних даних. У медичній галузі датасети часто включають зображення з різних типів (рентген, МРТ, КТ, ультразвук), анотації (класифікаційні мітки, сегментаційні маски) та метадані (вік пацієнта, діагноз). Однак, виклики включають обмежену доступність через конфіденційність, дисбаланс класів (рідкісні захворювання) та варіабельність якості зображень (шум, артефакти).

У цьому підрозділі розглянуто основні публічні датасети для задач класифікації, сегментації та детекції в медичних зображеннях, з акцентом на їх характеристики, переваги та недоліки.

1.4.1 Датасети для класифікації патологій

Однією з найпоширеніших задач є класифікація захворювань на рентгенівських знімках грудної клітки. Ключовий датасет – Chest X-ray, розроблений Національними інститутами здоров'я США у 2017 році. Він містить 112 120 фронтальних рентгенівських зображень від 30 805 пацієнтів, анотованих 14 мітками захворювань (наприклад, пневмонія,

кардіомегалія, плевральний випіт). Зображення у форматі PNG з роздільною здатністю 1024x1024 пікселів.

Серед сильних сторін цього датасету можна виділити:

– великий обсяг;

– одне зображення може мати кілька міток;

– публічна доступність на Kaggle.

Однак, існують і слабкі сторони:

– дисбаланс класів (наприклад, «No Finding» – 60% зразків);

– шум у анотаціях через автоматичне витягнення з радіологічних звітів.

Цей датасет широко використовується для бенчмаркінгу CNN, таких як ResNet, з досягнутою точністю до 0.89.

Інший датасет – MIMIC-CXR від MIT та Beth Israel Deaconess Medical Center (2019). Він включає 377 110 рентгенівських зображень від 65 379 пацієнтів, з супровідними текстовими звітами (понад 227 000). Анотації витягнуті за допомогою інструментів оброки природньої мови. У дослідженнях з CNN MIMIC-CXR досягає F1-score 0.75 для мультилейбл класифікації.

Основні достоїнства:

– інтеграція зображень з текстовими даними для мультимодальних моделей;

– висока різноманітність (включає COVID-19 випадки в оновленнях).

Негативні аспекти:

– потреба в потужних ресурсах для обробки;

– неповні анотації для рідкісних патологій.

Для шкірних захворювань актуальний ISIC Archive (International Skin Imaging Collaboration), оновлений у 2020 році. Він містить понад 100 000 дерматоскопічних зображень меланоми та інших уражень шкіри, з анотаціями (класифікація, сегментація). Точність CNN на ISIC сягає 94%.

Ключові плюси:

– високоякісні зображення;

– експертні анотації.

Мінуси:

– фокус лише на дерматології;

– дисбаланс (меланома – меншість).

1.4.2 Датасети для сегментації органів та патологій

Для задач сегментації популярний BraTS (Brain Tumor Segmentation Challenge), організований MICCAI з 2012 року, оновлений у 2023. Він включає 2000+ мультипротокольних МРТ-знімків мозку від пацієнтів з гліомами, з піксельними анотаціями (ядро пухлини, набряк, посилення). Зображення 3D (240x240x155 вокселів). BraTS ідеальний для тестування U-Net та її варіацій.

Позитивні аспекти:

– фокус на 3D-сегментації;

– щорічні челенджі з бенчмарками.

Обмеження:

– обмеження на мозок;

– висока обчислювальна складність.

Medical Segmentation Decathlon (MSD) від Монреальського університету (2018) – універсальний датасет з 10 завданнями сегментації (печінка на КТ, підшлункова на МРТ тощо), загалом понад 2000 зображень. Моделі на MSD досягають середньої точності 0.85.

Основні вигоди:

– різноманітність модальностей;

– стандартизовані метрики.

Недосконалості:

– менший обсяг порівняно з ChestX-ray14;

– потреба в 3D-обробці.

Для серцево-судинних захворювань – ACDC (Automated Cardiac Diagnosis Challenge, 2017), з 150 МРТ-серій серця, анотованими для сегментації шлуночків. Серед переваг можна виділити динамічні послідовності (cine-MRI). Серед недоліків: фокус на кардіології.

1.4.3 Датасети для детекції та мультимодальних задач

Для детекції об'єктів (bounding boxes) корисний LIDC-IDRI (Lung Image Database Consortium, 2008), з 1018 КТ-сканами легень, анотованими нодулами (понад 2600). Серед переваг виділяються множинні анотації від радіологів. Проте, дані старі та фокус датасета лише на легенях.

COVID-19 специфічні датасети, такі як COVIDx (2020), комбінують кілька джерел (понад 30 000 рентгенів з мітками COVID/норма/пневмонія). Превагою такого датасету є надзвичайна потреба під час пандемій.

1.4.4 Порівняльний аналіз датасетів

З аналізу (таблиця 1.3) видно, що для класифікації оптимальні великі датасети як ChestX-ray14, для сегментації – BraTS або MSD.

Таблиця 1.3 – Порівняння основних датасетів для медичних зображень

Датасет	Обсяг	Модальність	Задачі
ChestX-ray14	112 120	Ренген	Класифікація (14 класів)
MIMIC-CXR	377 110	Ренген	Класифікація, NLP
BraTS	2000+	МРТ	Сегментація (3D)

Продовження таблиці 1.3

Датасет	Обсяг	Модальність	Задачі
ISIC	100 000+	Дерматоскопія	Класифікація, сегментація
MSD	2000+	МРТ/КТ	Сегментація (10 органів)
LIDC-IDRI	1018	КТ	Детекція нодулів

У роботі рекомендовано комбінувати датасети з аугментацією для подолання дисбалансу. Для експериментів у цій роботі обрано ChestX-ray14 та BraTS як основні, з огляду на їх доступність та релевантність до CNN.

1.5 Постановка задачі

На основі проведеного аналізу предметної галузі, типів глибоких згорткових мереж та доступних датасетів сформульовано постановку задачі дослідження. Об’єктом дослідження є процес автоматичного аналізу медичних зображень (рентген, МРТ, КТ) за допомогою методів глибокого навчання. Предметом дослідження виступають глибокі згорткові нейронні мережі (CNN) для задач класифікації патологій, сегментації органів та детекції аномалій.

Метою роботи є розробка та оптимізація гібридної моделі на базі глибоких згорткових мереж для підвищення точності та ефективності автоматичного аналізу медичних зображень, з урахуванням специфіки медичних даних (шум, варіабельність анатомії, обмежений обсяг анотованих зразків).

Успішне досягнення поставленної мети передбачає реалізацію наступних завдань:

– провести аналіз сучасного стану проблеми автоматичного аналізу медичних зображень, включаючи огляд архітектур CNN та датасетів;

– обґрунтувати вибір гібридної архітектури моделі для комбінованої класифікації та сегментації;

– розробити методику передобробки даних, включаючи нормалізацію, аугментацію та балансування класів, адаптовану до обраних датасетів;

– реалізувати модель, провести навчання та оптимізацію гіперпараметрів;

– оцінити ефективність моделі за ключовими метриками на тестових наборах даних, порівняти з базовими архітектурами.

Методи дослідження включають: теоретичний аналіз літератури та джерел, моделювання архітектури CNN, програмну реалізацію, оцінку результатів.

2 ПРОЕКТУВАННЯ ДОДАТКУ

2.1 Функціональні вимоги до додатку

Розробка додатку для аналізу медичних зображень за допомогою штучного інтелекту вимагає чіткого визначення функціональних вимог, які забезпечать ефективну взаємодію користувача з системою. Ці вимоги базуються на аналізі предметної галузі, проведеному в попередньому розділі, та враховують специфіку медичних даних, таких як конфіденційність, точність обробки та інтеграція з моделями глибокого навчання.

Основні функціональні вимоги до проекту включають:

– реєстрація та авторизація користувачів. Користувач повинен мати можливість створити обліковий запис, вказавши логін, пароль та пошту. Система повинна підтримувати авторизацію для доступу до персоналізованих функцій, з урахуванням заходів безпеки, таких як хешування паролів;

– завантаження медичних зображень. Користувач може завантажувати файли зображень у форматах PNG, JPEG. Система повинна перевіряти валідність файлів;

– аналіз зображень за допомогою моделі штучного інтелекту. Після завантаження система автоматично застосовує гібридну модель глибоких згорткових мереж для виявлення патологій, детекції аномалій. Результати повинні включати візуалізацію та метрики точності;

– перегляд результатів аналізу. Користувач отримує детальний звіт з результатами, включаючи класифікацію патологій, сегментовані області;

– збереження та перегляд історії аналізів: Авторизовані користувачі можуть переглядати архів попередніх завантажень та результатів.

Ці вимоги забезпечують повний цикл роботи з медичними зображеннями: від завантаження до інтерпретації результатів. Для реалізації

цих вимог необхідно обрати архітектуру, яка забезпечить масштабованість, безпеку та інтеграцію з AI-моделями.

2.2 Вибір архітектури додатку

Вибір архітектури проекту є фундаментальним етапом у розробці програмного забезпечення, особливо для систем, що інтегрують штучний інтелект для аналізу медичних зображень. Архітектура визначає, як компоненти системи взаємодіятимуть між собою, впливає на продуктивність, масштабованість, безпеку та легкість підтримки. Неправильний вибір може призвести до неефективного використання ресурсів, труднощів з інтеграцією моделей глибокого навчання або високих витрат на рефакторинг у майбутньому.

У контексті нашого додатку, де потрібна обробка великих обсягів даних, інтеграція з AI-моделями та підтримка веб-інтерфейсу для користувачів, критерії вибору включають:

– масштабованість: здатність справлятися з зростанням навантаження, наприклад, при одночасному аналізі кількох КТ-сканів;

– складність розробки та підтримки: для обмеженого ресурсу потрібна простота;

– інтеграція з AI: легкість включення моделей для задач класифікації патологій чи сегментації органів;

– вартість: мінімізація інфраструктурних витрат.

Розглянемо основні архітектурні шаблони, їх переваги, недоліки та застосування в подібних системах.

2.2.1 Монолітна архітектура

Монолітна архітектура передбачає створення єдиної, інтегрованої системи (рисунок 2.1), де всі компоненти від інтерфейсу користувача до

бізнес-логіки, бази даних та серверної частини розміщені в одному кодовому базі [9]. Цей підхід подібний до моноліту, де зміни в одній частині впливають на всю систему.

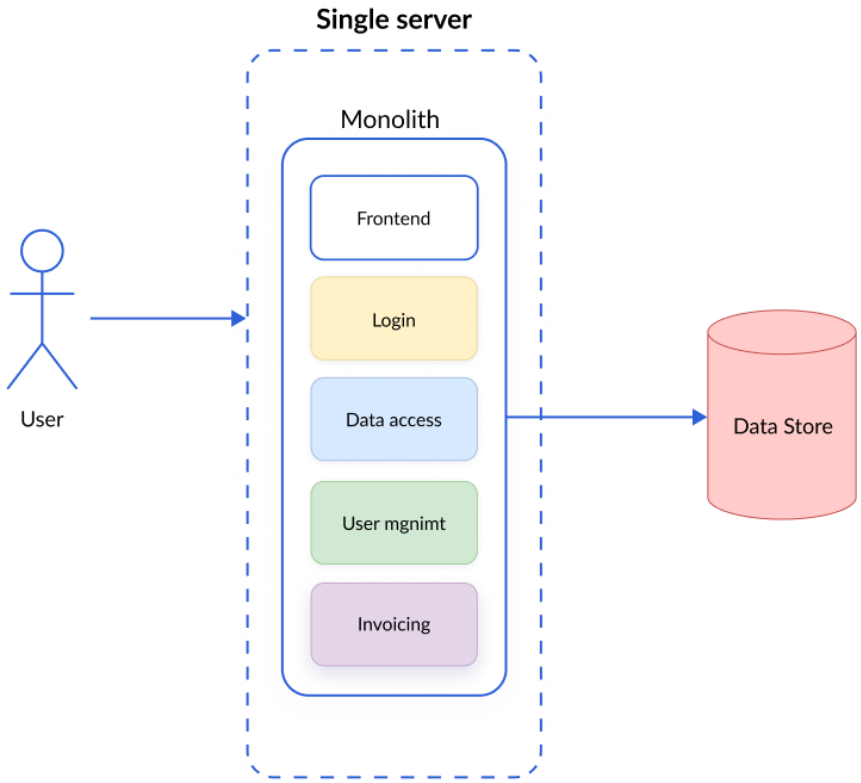

Рисунок 2.1 – Схема монолітної архітектури додатку

Переваги монолітної архітектури:

– швидка розробка та розгортання. Для невеликих команд або під час створення прототипу монолітна архітектура дає можливість рухатися значно швидше. Уся логіка зосереджена в одному застосунку, тому розробники можуть одразу вносити зміни й бачити результат без необхідності узгоджувати їх між кількома сервісами чи оновлювати інфраструктурні компоненти. Це скорочує час на налаштування середовища, зменшує кількість помилок, пов’язаних з інтеграцією, та дозволяє швидше випускати робочі версії продукту;

– спрощене тестування та налагодження. Оскільки вся логіка зосереджена в одному застосунку, помилки легше відслідковувати й

відтворювати – немає складних взаємодій між розподіленими сервісами. Для медичного застосунку це особливо важливо: можна швидко перевіряти коректність обробки та аналізу зображень у локальному середовищі, не турбуючись про мережеві затримки чи узгодження взаємодії між кількома компонентами;

– низькі витрати на інфраструктуру. Моноліт не вимагає використання систем оркестрації чи складної мережевої інфраструктури. Це зменшує як фінансові витрати, так і технічні вимоги до підтримки. У контексті AI-систем це особливо вигідно, адже вивільнені ресурси можна спрямувати на GPU-обчислення, необхідні для роботи моделей;

– ефективність для невеликих додатків. Коли система має обмежений набір функцій монолітна архітектура працює оптимально. Вона не створює додаткових накладних витрат, пов’язаних із запуском та взаємодією багатьох сервісів, тому використовує ресурси значно раціональніше. Це дозволяє отримати стабільну продуктивність без зайвої інфраструктурної складності.

Недоліки монолітної архітектури:

– обмежена масштабованість. Коли навантаження зростає, наприклад, якщо багато користувачів одночасно аналізують знімки, монолітну систему доводиться масштабувати цілком. Це зазвичай означає вертикальне масштабування, тобто додавання потужніших серверів, що є дорогим і має фізичні обмеження. У сфері медицини, де обсяг даних швидко збільшується, така модель масштабування може спричиняти затримки в роботі та зменшення пропускної здатності;

– ускладнене впровадження нових технологій. Сильна пов’язаність модулів у моноліті робить важким оновлення або заміну технологічних компонентів. Наприклад, перехід із TensorFlow на PyTorch може вимагати значного переписування бізнес-логіки, оскільки всі частини системи залежать одна від одної та не мають чітких меж;

– підвищений ризик збоїв. Оскільки вся система є єдиним цілим, помилка в одному модулі може спричинити зупинку всього застосунку. Для медичних систем, де час і доступність критично важливі, це створює серйозні ризики для роботи;

– труднощі підтримки та розвитку. Зі збільшенням функціоналу моноліт швидко «розростається», стає складним у підтримці та модифікаціях. Це особливо відчутно в проектах, де AI-алгоритми постійно оновлюються або вдосконалюються. Кожне нове оновлення ускладнює структуру коду та збільшує витрати часу на його супровід.

Моноліт підходить для стартапів або прототипів, як система для аналізу рентгенів на пневмонію, де швидкість розробки важливіша за масштабованість. Однак, для великих систем він поступається мікросервісам.

2.2.2 Мікросервісна архітектура

Мікросервісна архітектура розбиває систему на незалежні, маленькі сервіси (рисунок 2.2), кожен з яких відповідає за конкретну функцію [10]. Сервіси комунікують через API і можуть бути розгорнуті окремо.

Переваги мікросервісної архітектури:

– висока масштабованість. Кожен сервіс можна масштабувати окремо, не зачіпаючи інші компоненти системи. Наприклад, модуль, який обробляє медичні зображення за допомогою AI, легко масштабувати горизонтально;

– гнучкість і швидке впровадження нових функцій. Оскільки кожен компонент є окремим сервісом, можна додавати нові можливості без ризику порушити роботу вже існуючих частин системи. Такий підхід добре узгоджується з процесами, де важлива швидка ітерація та часті оновлення;

– ізоляція збоїв. Якщо один сервіс виходить з ладу, він не впливає на доступність інших. Наприклад, збій у сервісі класифікації зображень не зупинить модуль авторизації чи систему зберігання даних. Це значно

підвищує надійність, що критично важливо в медичних застосунках, де стабільність системи має прямий вплив на роботу персоналу;

– технологічна незалежність. Кожен сервіс може бути написаний мовою чи фреймворком, який найкраще підходить для конкретного завдання. Наприклад, для AI-аналізу зручніше використовувати Python, а для сервісів безпеки або логування – Java чи C#. Це дозволяє вибирати інструменти, оптимальні для кожної частини системи;

– підвищення рівня безпеки. Окремі бази даних для різних сервісів зменшують ризик витоку або некоректного доступу до конфіденційної інформації.

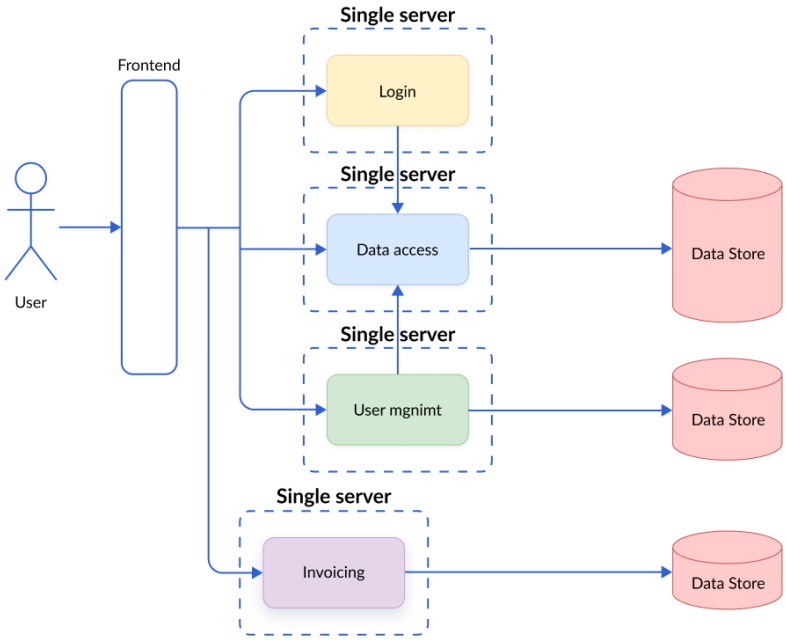

Рисунок 2.2 – Схема мікросервісної архітектури додатку

Недоліки мікросервісної архітектури:

– висока складність управління. Мікросервіси вимагають розвиненої інфраструктури для оркестрації контейнерів, централізованого моніторингу та опрацювання розподілених транзакцій. Це суттєво ускладнює розробку й підтримку системи, особливо якщо команда невелика або немає DevOps-спеціалістів;

– накладні витрати на комунікацію. Оскільки сервіси взаємодіють між собою через мережу, виникають додаткові затримки. Передача великих файлів, таких як медичні зображення високої роздільності, може сповільнювати обробку й негативно впливати на загальну продуктивність системи;

– складність роботи з даними. Кожен сервіс зазвичай має власну базу даних, тому забезпечення консистентності інформації між ними стає складним завданням. Наприклад, синхронізація результатів аналізу зображень може вимагати додаткових механізмів узгодження даних між сервісами, що суттєво ускладнює архітектуру;

– підвищені витрати на інфраструктуру. Для невеликих проектів мікросервісний підхід може бути надмірно дорогим. Потрібна хмарна інфраструктура для запуску та управління сервісами, що значно збільшує фінансове навантаження порівняно з монолітом.

Мікросервісна архітектура особливо добре підходить для великих, комплексних платформ, де потрібно поєднувати багато різних функціональних модулів. Вона дає змогу розробляти окремі компоненти незалежно, масштабувати їх відповідно до навантаження та оновлювати частинами, не впливаючи на роботу всієї системи.

Такий підхід забезпечує гнучкість, надійність і можливість швидко інтегрувати нові технології. Наприклад, модуль, що відповідає за аналіз даних або виконання AI-завдань, може існувати як окремий сервіс, легко підключаючись до інших частин платформи. Це робить систему більш адаптивною до змін, простішою в розширенні та зручнішою для довгострокового розвитку.

2.2.3 Шарова архітектура

Шарова архітектура організовує систему у вигляді горизонтальних рівнів, кожен з яких відповідає за певну функцію: презентаційний шар, шар

бізнес-логіки, та шар доступу до даних (рисунок 2.3). Комунікація між шарами відбувається строго послідовно. Кожен шар взаємодіє лише з безпосередньо нижчим або верхнім, що забезпечує чітке розділення відповідальності і полегшує підтримку системи [11].

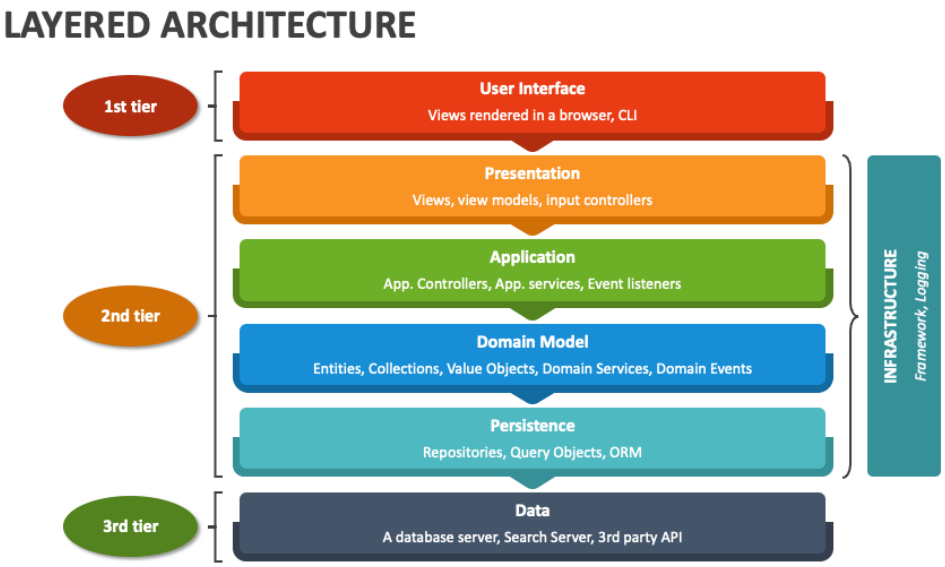

Рисунок 2.3 – Схема шарової архітектури додатку

Переваги шарової архітектури:

– чітке розділення обов’язків. Кожен шар відповідає лише за свою функціональну частину системи. Наприклад, шар бізнес-логіки може обробляти дані та інтегрувати моделі штучного інтелекту, а шар доступу до даних зберігати результати обробки. Така організація спрощує підтримку та розвиток системи, оскільки зміни в одному шарі мінімально впливають на інші;

– легкість тестування та масштабованість. Кожен шар можна перевіряти автономно, що спрощує виявлення та виправлення помилок. Крім того, вертикальне масштабування системи відбувається без необхідності змінювати інші шари, що забезпечує ефективну роботу під навантаженням;

– повторне використання компонентів. Компоненти кожного шару спроектовані так, щоб їх можна було легко застосовувати в інших проектах або модулях. Це зменшує витрати на розробку та дозволяє швидко створювати нові системи, використовуючи вже перевірені та стабільні елементи;

– підвищення безпеки. Шар доступу до даних забезпечує ізоляцію чутливої інформації від інших частин системи. Це обмежує прямий доступ до критичних даних, знижує ризик витоку інформації та полегшує впровадження політик безпеки.

Недоліки шарової архітектури:

– накладні витрати на продуктивність. Оскільки інформація проходить послідовно через усі шари, це може уповільнювати обробку великих обсягів даних або складних об’єктів, таких як високороздільні зображення;

– обмежена гнучкість. Зміни або оновлення в одному шарі часто потребують внесення коригувань в інші шари, що ускладнює розвиток системи. Крім того, шарова архітектура не оптимальна для паралельної обробки даних, оскільки взаємодія між шарами відбувається послідовно;

– складність для великих та високонавантажених систем. У додатках, які працюють в режимі реального часу, надмірні накладні витрати через послідовну обробку шарів можуть стати критичними та впливати на швидкодію системи.

2.2.4 MVC-архітектура

MVC (Model-View-Controller) – архітектурний шаблон (рисунок 2.4), який розділяє систему на три компоненти: Model відповідає за дані та бізнес-логіку (наприклад, роботу AI-моделей), View – за відображення та інтерфейс користувача, а Controller координує взаємодію між ними, обробляючи запити користувача та передаючи дані між Model і View [12].

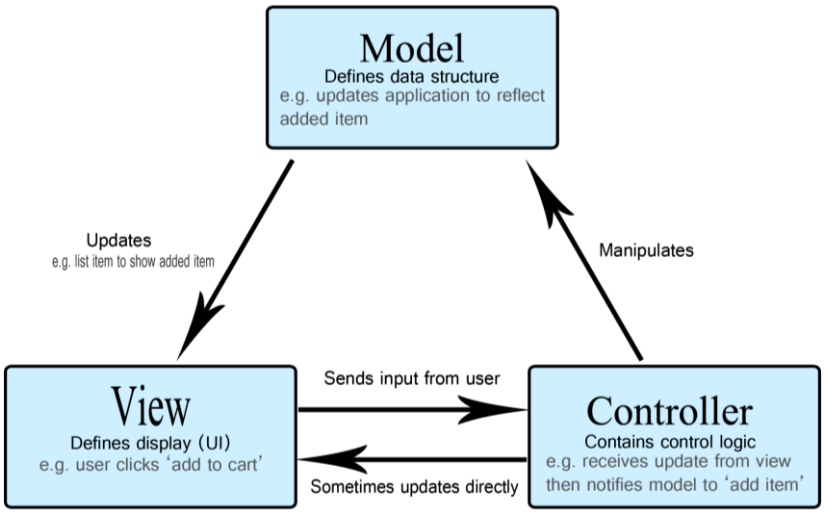

Рисунок 2.4 – Схема MVC-архітектури додатку

Переваги MVC:

– чітке розділення відповідальностей. Кожен компонент виконує свою роль незалежно, що спрощує інтеграцію складних алгоритмів у Model, наприклад, підключення нейронної мережі для обробки даних, без втручання в логіку відображення чи контролера;

– швидка розробка. Використання веб-фреймворків спрощує створення застосунків із інтегрованими AI-моделями, оскільки вони надають готові інструменти для роботи з базами даних, маршрутизації запитів та організації логіки. Це дозволяє швидко реалізовувати функціонал і тестувати нові можливості без необхідності писати низькорівневий код для кожного компонента.

– масштабованість. Архітектура дозволяє легко розділяти систему на окремі сервіси або модулі у майбутньому, що спрощує перехід до більш масштабованої мікросервісної структури при зростанні навантаження чи функціональності.

Недоліки MVC:

– ризик перетворення на моноліт. Зі збільшенням функціональності система може накопичувати залежності між компонентами, що ускладнює

масштабування та підтримку, і фактично перетворює її на монолітну архітектуру.

– неоптимально для невеликих проектів. Для простих застосунків додаткові накладні витрати на структуру та компоненти можуть бути зайвими, роблячи систему громіздкою та складною для підтримки.

MVC дозволяє чітко розділити дані, логіку та інтерфейс, що спрощує розробку, тестування та підтримку системи. Така структура особливо корисна для інтеграції складних алгоритмів або модулів, оскільки зміни в одному компоненті мінімально впливають на інші.

2.2.5 Порівняльний аналіз архітектурних шаблонів додатку

Проведений аналіз архітектурних шаблонів дозволяє зробити висновки щодо їхньої придатності для розробки веб-додатку з інтеграцією штучного інтелекту для аналізу медичних зображень. Для оцінки використано ключові критерії, такі як масштабованість, складність розробки, інтеграція з AI, безпека, витрати та застосування в медичній сфері. Результати порівняння узагальнено в таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз архітектурних шаблонів

Архітектура	Масштабованість	Складність розробки	Інтеграція з ШІ	Витрати
Монолітна	Низька	Низька	Низька	Низькі
Мікросервісна	Висока	Висока	Висока	Високі
Шарова	Середня	Середня	Середня	Середні
MVC	Середня	Низька	Висока	Низькі

З таблиці видно, що монолітна архітектура є найпростішою для початкової реалізації, але її обмежена масштабованість робить її непридатною для систем з великими обсягами даних, як медичні зображення, де навантаження може зростати непередбачувано.

Мікросервісна архітектура перевершує інші за гнучкістю та масштабованістю, дозволяючи незалежно розгортати ШІ-модулі, що корисно для інтеграції з медичними системами. Однак, її висока складність і витрати роблять її надмірною для невеликого проекту з обмеженими ресурсами. Шарова архітектура забезпечує чітку сепарацію, але страждає від витрат у продуктивності, особливо при обробці великих зображень, де дані проходять через всі шари послідовно.

Обґрунтування вибору MVC архітектури ґрунтується на балансі переваг, що відповідають вимогам проекту.

По-перше, MVC забезпечує чітку сепарацію компонентів: Model відповідає за дані та логіку (інтеграція моделей CNN), View – за інтерфейс (відображення результатів аналізу), Controller – за взаємодію (обробка запитів на завантаження зображень). Це полегшує розробку та тестування, особливо в контексті штучного інтелекту, де моделі можна інтегрувати в Model без впливу на інші частини.

По-друге, MVC підтримує високу інтеграцію з Python-фреймворками, такими як Django, що дозволяє легко поєднувати веб-інтерфейс з бібліотеками TensorFlow чи Keras для аналізу зображень. Наприклад, у системах для діагностики пневмонії на рентгенах, MVC забезпечує ефективну обробку даних з бази без складної оркестрації.

По-третє, MVC має середню масштабованість, достатню для проекту, з можливістю переходу до мікросервісів у майбутньому. Недоліки, як потенційна монолітність при зростанні, мінімізуються в обмеженому проекті. Порівняно з іншими, MVC пропонує оптимальне співвідношення простоти та ефективності для веб-додатків зі штучним інтелектом, де швидкість та інтеграція з моделями глибокого навчання є пріоритетними.

Таким чином, вибір MVC є обґрунтованим для даного проекту, оскільки він забезпечує швидку розробку, надійну інтеграцію ШІ та відповідність вимогам медичної сфери, з потенціалом для подальшого розвитку.

3 ВИБІР ТА ОПИС ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ СИСТЕМИ АНАЛІЗУ

У цьому розділі розглядаються ключові технології, обрані для реалізації веб-додатку, призначеного для аналізу медичних зображень з використанням методів штучного інтелекту. Вибір інструментів базується на аналізі вимог проекту, таких як необхідність ефективної обробки великих обсягів даних, інтеграція моделей глибокого навчання, забезпечення безпеки та зручності користувацького інтерфейсу. На відміну від традиційних підходів, акцент робиться на сучасних фреймворках і бібліотеках, які дозволяють досягти високої продуктивності та масштабовальності. Детальний опис кожної технології включає її історію розвитку, основні принципи роботи, переваги та недоліки, а також обґрунтування вибору для конкретної задачі аналізу медичних знімків (наприклад, рентгенівських зображень для виявлення патологій, таких як пневмонія чи пухлини).

3.1 Мова програмування Python

Python є високорівневою мовою програмування загального призначення, яка набула широкого поширення в наукових обчисленнях, машинному навчанні та веб-розробці завдяки своїй простоті, читабельності та багатій екосистемі бібліотек [13]. Розроблена Гвідо ван Россумом у 1991 році як наступниця мови ABC, Python спочатку позиціонувалася як інструмент для автоматизації рутинних завдань. Перша версія (0.9.0) вийшла в лютому 1991 року, а значний прорив стався з випуском Python 2.0 у 2000 році, який ввів генератори та підтримку Unicode. У 2008 році з'явилася Python 3.0, яка усунула несумісності з попередньою версією та оптимізувала синтаксис для кращої ефективності [14]. На сьогодні Python є

однією з найпопулярніших мов завдяки спільноті, яка налічує мільйони розробників, і репозиторію PyPI з понад 500 000 пакетами.

Основний синтаксис Python базується на принципах читабельності: блоки коду виділяються відступами, а не фігурними дужками, що зменшує кількість помилок. Мова підтримує кілька парадигм програмування – процедурну, об'єктно-орієнтовану та функціональну.

Переваги Python для проекту з аналізу медичних знімків включають:

– широка екосистема для роботи з даними та ШІ. Різноманітні бібліотеки для обробки та аналізу даних дозволяють ефективно працювати з великими масивами інформації. Це особливо важливо при обробці високороздільних зображень або складних датасетів, оскільки забезпечує швидкі та надійні обчислення без необхідності писати базові алгоритми з нуля;

– простота інтеграції з моделями глибокого навчання. Python широко використовується у популярних фреймворках для AI, що спрощує розробку та інтеграцію нейронних мереж для різних задач, таких як обробка зображень або виявлення складних закономірностей у даних;

– кросплатформенність та швидке прототипування. Код можна виконувати на різних операційних системах без змін, що полегшує розробку та тестування. Крім того, можливість швидко створювати прототипи дозволяє значно скоротити час розробки складних моделей і алгоритмів у порівнянні з низькорівневими мовами програмування;

– сильна підтримка спільноти. Наявність великої бази знань, форумів і детальної документації дозволяє швидко знаходити рішення для різних проблем. Це особливо корисно при оптимізації роботи з великими обсягами даних або складними моделями, оскільки скорочує час на пошук рішень та навчання.

Недоліки мови:

– нижча швидкість виконання порівняно з компільованими мовами. Інтерпретований характер мови може уповільнювати обробку великих

обсягів даних, проте це часто компенсується використанням оптимізованих бібліотек або розширень, які виконуються на більш швидких мовах, забезпечуючи ефективність обчислень;

– високе споживання пам’яті. При обробці великих обсягів даних система може швидко використовувати значні ресурси оперативної пам’яті. Щоб уникнути перевантаження, код потребує оптимізації, наприклад, за допомогою потокової обробки або генераторів даних, що дозволяє працювати з великими наборами даних ефективніше;

– GIL (Global Interpreter Lock) обмежує ефективний паралелізм у багатопотокових програмах на CPU. Однак у задачах AI це не є суттєвим обмеженням, оскільки основні обчислення зазвичай виконуються на GPU, що дозволяє обходити це обмеження.

У задачах аналізу медичних зображень, де необхідна швидка розробка й часте експериментування з моделями CNN, Python забезпечує оптимальний баланс між продуктивністю й простотою. Альтернативи на кшталт R чи MATLAB поступаються в гнучкості та можливостях інтеграції з веб-сервісами, а Java є надмірно вербозною для швидкої побудови систем штучного інтелекту.

3.2 Фреймворк побудови додатку

Django – високорівневий веб-фреймворк на Python, орієнтований на швидку, безпечну та підтримувану розробку веб-додатків [15]. Він був створений у 2003 році командою Lawrence Journal-World (Канзас, США) для автоматизації роботи редакційних інструментів і управління контентом новинних порталів. У 2005 році Django був відкритий як проект з відкритим вихідним кодом, після чого його розвиток перейшов до спільноти під керівництвом Django Software Foundation (DSF) [16].

Ключовою особливістю Django є наявність у фреймворку повного набору вбудованих інструментів, необхідних для створення сучасних веб-систем. До них належать:

– власний ORM для роботи з базами даних;

– систему шаблонів для формування HTML;

– автоматично згенеровану адмін-панель, яку широко використовують для внутрішніх систем;

– вбудовані механізми безпеки;

– систему маршрутизації, форм, механізми кешування і роботу з файлами.

Django також має одну з найбільших екосистем у Python: тисячі пакетів дозволяють швидко додавати REST API, асинхронні задачі, логування, фонову обробку даних, роботу з чергами та кешами. Завдяки цьому фреймворк застосовується в різних галузях.

У контексті сучасних ШІ-додатків Django виступає надійною основою для побудови інфраструктури навколо моделей машинного навчання: API-ендпоінтів, логування, маршрутизації, доступу до БД, керування користувачами та захисту даних.

Переваги використання Django для додатку:

– вбудована безпека. Django забезпечує автоматичний захист від ключових веб-загроз, включаючи SQL-ін’єкції, міжсайтовий скриптинг (XSS) та міжсайтову підробку запитів (CSRF), завдяки вбудованим механізмам фреймворку. Ці засоби працюють за замовчуванням і не потребують додаткових налаштувань, що значно знижує ризик помилок розробників;

– ORM для ефективної роботи з базою даних. Django містить потужний об’єктно-реляційний відображувач (ORM), який автоматично генерує SQL-запити на основі Python-коду. Це значно спрощує взаємодію з базою даних: операції на кшталт фільтрації, сортування чи агрегації виконуються без необхідності писати SQL вручну. Такий підхід значно

скорочує час розробки, підвищує читабельність коду та мінімізує ризик помилок, пов’язаних із ручним написанням запитів;

– масштабованість. Django забезпечує високу масштабованість завдяки підтримці сучасних інструментів і технологій. Використання кешування дозволяє суттєво зменшити навантаження на сервер і прискорити доступ до часто використовуваних даних, таких як результати попередніх аналізів. Інтеграція з системами асинхронних задач дає змогу виконувати ресурсомісткі операції у фоновому режимі без блокування основного додатку. Крім того, Django легко поєднується з хмарними сервісами, такими як AWS, що дозволяє зберігати великі обсяги медичних знімків у розподіленому сховищі, забезпечуючи гнучкість і готовність до зростання проекту за рахунок горизонтального масштабування;

– адміністративна панель. Django має вбудовану адмін-панель, яка автоматично генерується на основі моделей і дозволяє зручно керувати даними без необхідності створювати окремий інтерфейс. Це особливо корисно для задач, пов’язаних із модерацією та управлінням анотованими датасетами, що використовуються для донавчання моделей глибокого навчання. Адміністратори можуть швидко додавати, редагувати або видаляти записи, переглядати історію змін, фільтрувати дані за різними параметрами та контролювати якість анотацій. Завдяки гнучкій системі налаштувань адмін-панель легко адаптується під специфічні потреби проекту, що значно скорочує час розробки внутрішніх інструментів.

Недоліки Django:

– надмірність для простих проектів. Django містить багато вбудованих інструментів і механізмів, що робить його потужним, але водночас «важким» для невеликих застосунків. Якщо проект не потребує складної логіки, моделювання даних чи аутентифікації, то використання Django може бути невиправдано громіздким. Однак у системах зі складним функціоналом його можливості стають оптимальними;

– продуктивність. Django не завжди демонструє найвищу швидкість у сценаріях із великою кількістю одночасних запитів, що може бути відчутно для високонавантажених API. Проте впровадження ASGI та підтримка асинхронних операцій дозволяють значно підвищити ефективність обробки запитів і зменшити затримки, роблячи фреймворк придатним для сучасних масштабованих веб-сервісів.

Django є потужним фреймворком, який поєднує швидку розробку, високу безпеку та масштабованість. Він забезпечує всі необхідні інструменти – від ORM і системи шаблонів до адмін-панелі та вбудованих механізмів захисту. Це робить його особливо ефективним для проектів, де важливі надійність, структурованість і можливість швидко впроваджувати нові функції.

Попри певні недоліки сучасні можливості дозволяють ефективно масштабувати Django-застосунки. У результаті Django залишається одним із найкращих виборів для побудови складних веб-систем, де важливі підтримуваність та швидка еволюція продукту.

3.3 Бібліотека реалізації моделі глибинного навчання

TensorFlow – це відкрита платформа для машинного навчання, створена командою Google Brain і вперше представлена публічно в листопаді 2015 року [17]. Спочатку розроблена як внутрішній інструмент для масштабованого тренування моделей глибокого навчання, вона швидко стала одним із найпопулярніших фреймворків для машинного навчання у світі.

З випуском TensorFlow 2.0 у 2019 році фреймворк зазнав значної еволюції, спрощено робочий процес, інтегровано Keras як основний високорівневий API [18]. Це зробило TensorFlow більш доступним для дослідників і практиків.

Архітектурно TensorFlow працює на основі обчислювальних графів, де дані представлені у вигляді тензорів (багатовимірних масивів), що проходять через послідовність математичних операцій. Такий підхід забезпечує ефективне виконання, можливість розгортання на різних платформах і гнучку оптимізацію продуктивності.

Переваги TensorFlow:

– висока гнучкість для складних ШІ-задач. TensorFlow підтримує transfer learning, що дозволяє ефективно використовувати претреновані моделі. Це значно скорочує потребу у великих датасетах і прискорює експерименти;

– апаратне прискорення через GPU та TPU. Фреймворк має вбудовану підтримку CUDA-GPU та Tensor Processing Units, що дає змогу прискорити навчання великих нейромереж у десятки разів. Завдяки цьому моделі на масштабних наборах зображень можна тренувати у коротші терміни;

– зручний високорівневий API Keras. Keras, інтегрований у TensorFlow, забезпечує просте та інтуїтивне створення моделей. Це особливо корисно при розробці складних архітектур. Усе описується компактним та зрозумілим кодом;

– розвинена екосистема інструментів. TensorFlow має розвинену екосистему інструментів, що охоплює всі етапи роботи з моделями машинного навчання. TensorBoard дозволяє відстежувати метрики, графи обчислень і втрати під час навчання, забезпечуючи візуалізацію процесу, а TF Serving спрощує швидке розгортання моделей. Завдяки цьому TensorFlow підходить як для наукових досліджень і прототипування, так і для масштабованих виробничих рішень.

Серед недоліків можна виділити високе споживання ресурсів. Навчання великих глибоких нейромереж потребує значних обчислювальних ресурсів, зокрема GPU або TPU, а також великого обсягу оперативної пам’яті. Для невеликих робочих станцій або домашніх комп’ютерів це може стати серйозним обмеженням.

TensorFlow є потужною та стабільною платформою для побудови і розгортання моделей машинного навчання. Він поєднує гнучкість у роботі з різними типами нейронних мереж, підтримку апаратного прискорення (GPU/TPU), багатий набір інструментів для моніторингу, оптимізації та деплою моделей, а також широку спільноту та екосистему готових моделей.

Попри деяку складність для початківців та високі вимоги до ресурсів для тренування великих мереж, TensorFlow добре підходить як для досліджень і швидкого прототипування, так і для промислового використання у production-системах, де важлива стабільність, масштабованість та інтеграція з різними платформами.

3.4 Система управління базами даних PostgreSQL

PostgreSQL – це потужна реляційна система управління базами даних (СУБД) з відкритим кодом, яка розробляється з 1986 року в Університеті Каліфорнії в Берклі у проекті POSTGRES [19]. У 1996 році проект отримав сучасну назву PostgreSQL і почав активно впроваджувати стандарти SQL, забезпечуючи сумісність із широким спектром застосунків та мов програмування [20].

СУБД відзначається високою надійністю, підтримкою транзакцій, розширюваністю та відповідністю ACID-принципам. Вона дозволяє працювати з традиційними реляційними даними, а також із сучасними типами даних.

Версія PostgreSQL 16 включає покращену підтримку паралельних запитів, оптимізацію продуктивності для великих наборів даних і розширені можливості роботи з JSONB, що робить її придатною для сучасних веб та ШІ-застосунків, де потрібна швидка обробка структурованих і напівструктурованих даних.

PostgreSQL часто використовується як надійний бекенд для веб-додатків, аналітичних платформ та систем, які вимагають високої цілісності даних і масштабованості.

Переваги PostgreSQL:

– ACID-сумісність**.** PostgreSQL гарантує цілісність та надійність даних завдяки дотриманню принципів ACID (Atomicity, Consistency, Isolation, Durability). Це особливо важливо для систем, де критично зберігати точні результати обчислень і транзакцій, наприклад, у медичних чи фінансових застосунках;

– підтримка JSONB та гнучка робота з напівструктурованими даними. Тип даних JSONB дозволяє ефективно зберігати складні результати, наприклад ймовірності класів нейронних мереж, анотації чи метадані. Це поєднує переваги реляційної і документно-орієнтованої моделі, зберігаючи можливість виконання складних SQL-запитів;

– масштабованість і оптимізація продуктивності. PostgreSQL підтримує паралельне виконання запитів, індексацію, що дозволяє ефективно працювати з великими обсягами даних і складними аналітичними операціями. Це робить її придатною для веб-додатків, AI-платформ та аналітичних систем.

Недоліки PostgreSQL:

– високі вимоги до ресурсів. При роботі з великими таблицями або складними аналітичними запитами PostgreSQL може споживати значну кількість оперативної пам’яті та ресурсів CPU. Для великих проектів необхідні потужні сервери або кластерні рішення;

– горизонтальне масштабування складніше. Хоча існують механізми шардінгу і реплікації, горизонтальне масштабування PostgreSQL значно складніше, ніж у деяких NoSQL систем, що може бути обмеженням для дуже великих або глобально розподілених систем.

PostgreSQL є потужною і надійною реляційною СУБД з відкритим кодом, яка поєднує строгу ACID-сумісність, багатий набір типів даних та

розширень, підтримку складних аналітичних операцій і масштабованість. Вона дозволяє ефективно зберігати як структуровані, так і напівструктуровані дані, забезпечує високу надійність та стабільність для продакшн-систем.

Попри деяку складність налаштування, високі вимоги до ресурсів та обмежену простоту горизонтального масштабування, PostgreSQL залишається одним із найкращих виборів для веб-додатків, аналітичних платформ та систем, де важлива цілісність даних, гнучкість та довготривала підтримка.

3.5 Мови та інструменти фронтенд-розробки

У сучасних веб-додатках фронтенд відіграє ключову роль у забезпеченні зручної взаємодії користувача з системою. Для реалізації інтерфейсу веб-додатку з аналізу медичних зображень за допомогою штучного інтелекту обрано комбінацію базових технологій: HTML для структуризації контенту, CSS для стилізації та JavaScript для додавання динаміки та інтерактивності. Ці технології інтегруються з фреймворком Django, який генерує шаблони, дозволяючи створювати динамічні сторінки. На відміну від статичних сайтів, фронтенд тут передбачає обробку форм, відображення результатів аналізу та асинхронні запити для оновлення даних без перезавантаження сторінки.

3.5.1 HTML

HTML (HyperText Markup Language) є стандартом для створення структури веб-сторінок, дозволяючи визначати елементи, такі як форми, таблиці, зображення та посилання. Розробка HTML почалася в 1990-х роках Тімом Бернерс-Лі в CERN для обміну науковими документами. Перша версія HTML 2.0 (1995) стандартизувала базові теги, а HTML 4.01 (1999)

додав підтримку скриптів та стилів [21]. Сучасний HTML5 (рекомендація W3C з 2014 року, оновлена в 2020-х) ввів семантичні елементи та API для локального зберігання, що полегшує роботу з великими даними.

Принцип роботи HTML базується на тегах, які формують деревоподібну структуру документа (рисунок 3.1). Кожен елемент може мати атрибути для ідентифікації чи поведінки. У проекті HTML використовується для створення сторінок: головної, реєстрації/логіну, історії аналізів та результатів.

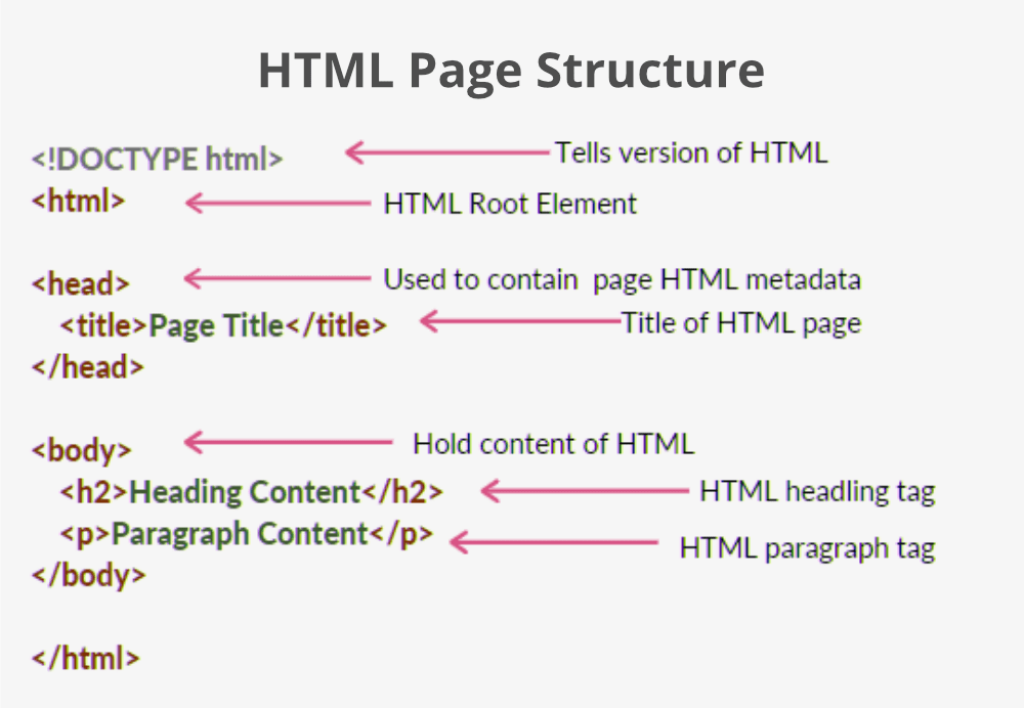

Рисунок 3.1 – Структура HTML-документу

Переваги HTML:

– легка інтеграція з динамічними даними. HTML добре інтегрується з серверними шаблонами та фреймворками, що дозволяє динамічно відображати інформацію на сторінках. Конструкції на зразок циклів або умовних блоків у шаблонах дозволяють автоматично формувати таблиці, списки чи графічні елементи на основі даних із бази, API чи моделей обробки даних;

– кросбраузерність. HTML5 сумісний з усіма сучасними браузерами та операційними системами, що забезпечує однакове відображення контенту на десктопах, ноутбуках, планшетах і мобільних пристроях. Це дозволяє створювати універсальні веб-додатки без необхідності окремої адаптації під кожну платформу;

– швидкість розробки та прототипування. HTML забезпечує базові будівельні блоки для швидкого створення каркасу веб-сторінки та інтерактивних елементів. За допомогою тегів можна швидко прототипувати інтерфейси, відображати графіку, таблиці чи візуалізації даних, що прискорює процес розробки і тестування ідей.

Недоліки HTML:

– статичність і відсутність динаміки. HTML сам по собі є статичною розміткою, тому без додаткових мов програмування неможливо реалізувати асинхронне завантаження даних, інтерактивні елементи чи динамічне оновлення контенту. Це обмежує можливості створення сучасних інтерактивних веб-додатків;

– обмежені можливості стилізації. Без використання CSS сторінки виглядають базово і не можуть забезпечити сучасний дизайн або привабливий користувацький інтерфейс. Для створення інтуїтивного, естетичного та зручного інтерфейсу необхідно поєднувати HTML з CSS;

– розмір та продуктивність документів. Великі HTML-документи, особливо з вбудованими ресурсами (наприклад, base64-зображення або великі таблиці даних), можуть уповільнювати завантаження сторінок і відображення контенту.

HTML є основою будь-якого веб-додатку, забезпечуючи структуровану розмітку контенту та семантичність сторінок. Його переваги включають доступність, кросбраузерність, легку інтеграцію з динамічними даними та швидке прототипування інтерфейсів.

Разом із тим, HTML сам по собі є статичним і обмеженим у стилізації, що потребує підтримки CSS та JavaScript для створення сучасних

інтерактивних та естетичних веб-додатків. Також великі документи можуть впливати на швидкість завантаження, тому для ефективної роботи додатків потрібні додаткові оптимізації.

3.5.2 CSS для стилізації та адаптивного дизайну

CSS (Cascading Style Sheets) керує візуальним представленням HTML-елементів, дозволяючи задавати кольори, шрифти, макети та анімації. Це забезпечує відокремлення контенту від дизайну та спрощує підтримку зовнішнього вигляду веб-додатків [22].

Розробка CSS розпочалася в 1994 році Хоконом Віумом Лі в CERN як доповнення до HTML. CSS1, випущений у 1996 році, ввів базові селектори та стилі для тексту та елементів. CSS2 (1998) додав позиціонування та підтримку медіа, а CSS3, починаючи з 2011 року, ввів модулі для флексбоксів, гридів, трансформацій та медіа-запитів, що значно розширило можливості адаптивного та складного дизайну.

На 2025 рік CSS підтримує сучасні функції, такі як змінні, калькуляції та контейнерні запити для адаптивного розташування елементів, що робить його ефективним інструментом для створення універсальних веб-інтерфейсів.

Принцип роботи CSS базується на селекторах (за тегом, класом, ID або атрибутом) та деклараціях у форматі властивість: значення (рисунок 3.2).

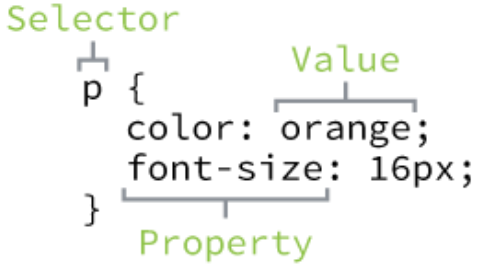

Рисунок 3.2 – Структура CSS-елементу

У проєкті CSS використовується для адаптивного дизайну: наприклад, мобільна версія забезпечує зручність користування на ходу, а темна тема зменшує навантаження на очі. За допомогою CSS можна швидко налаштовувати макети, кольорові схеми та візуальні ефекти, що підвищує зручність і естетичність інтерфейсу.

Переваги CSS:

– адаптивність. CSS підтримує медіа-запити та сучасні макети. Елементи сторінки автоматично підлаштовуються під різні розміри екранів, забезпечуючи зручність перегляду на десктопах, планшетах та мобільних пристроях;

– ефективність. Завдяки повторному використанню стилів через класи, ID та змінні, CSS зменшує обсяг коду і спрощує його підтримку. Це також прискорює завантаження сторінок і покращує продуктивність додатку, особливо якщо на сторінці відображається велика кількість елементів або графіки;

– візуальні ефекти та інтерактивність. CSS дозволяє додавати анімації, трансформації та переходи, які підвищують привабливість і зрозумілість інтерфейсу. Наприклад, ефекти наведення курсора або плавні зміни кольору елементів покращують сприйняття динамічних даних;

– сепарація стилю та структури. CSS відокремлює дизайн від HTML-контенту, що значно полегшує оновлення зовнішнього вигляду без зміни структури сторінки або логіки JavaScript. Це підвищує гнучкість розробки і спрощує підтримку великих веб-додатків.

Недоліки CSS:

– несумісність браузерів. Деякі старі версії браузерів можуть не підтримувати сучасні властивості CSS, такі як гриди, змінні або нові медіа-запити;

– складність для реалізації складних макетів. Створення складних багаторівневих або адаптивних макетів без спеціалізованих фреймворків

вимагає значно більше коду та уважного управління позиціонуванням і вирівнюванням елементів;

– продуктивність. Надмірна кількість селекторів, вкладених правил або великих стилів може уповільнювати рендеринг сторінки, особливо на старих пристроях або при великій кількості елементів. Оптимізація за допомогою мініфікації CSS-файлів та зменшення вкладеності допомагає покращити швидкість завантаження і відображення.

CSS є ключовим інструментом для керування візуальним оформленням веб-сторінок, забезпечуючи адаптивність, повторне використання стилів, інтерактивність та розділення структури і дизайну. Його можливості дозволяють створювати сучасні, зручні та естетично привабливі інтерфейси, які коректно відображаються на різних пристроях і екранах.

Разом із тим, CSS може ускладнювати роботу при реалізації складних макетів, потребує уваги до сумісності між браузерами і правильного управління продуктивністю на великих сторінках. Для ефективного використання важливо поєднувати CSS із сучасними методами оптимізації та фреймворками для прискорення розробки.

3.5.3 JavaScript для динаміки та інтерактивності

JavaScript (JS) – це скриптова мова програмування, яка дозволяє додавати логіку на клієнтській стороні веб-додатків [23]. Вона була створена Бренданом Айком у Netscape в 1995 році під назвою Mocha і стандартизована як ECMAScript (ES) у 1997 році.

З появою ES6 (2015) JavaScript отримав сучасні можливості: стрілкові функції, класи, модулі та інші синтаксичні покращення.

JS виконується в браузері і тісно взаємодіє з DOM, дозволяючи динамічно змінювати контент сторінки, реагувати на події користувача та реалізовувати інтерактивність без перезавантаження сторінки.

У проекті JavaScript використовується для асинхронних запитів (AJAX/Fetch), валідації форм, динамічного оновлення елементів інтерфейсу, наприклад, для відображення прогресу обробки даних або автоматичного оновлення результатів.

Переваги JavaScript:

– інтерактивність. JavaScript дозволяє динамічно реагувати на події користувача, змінювати вміст сторінки без її перезавантаження та створювати інтерактивні елементи інтерфейсу. Це підвищує зручність користування веб-додатком і покращує досвід взаємодії з ним;

– асинхронність. За допомогою Fetch API або AJAX JavaScript забезпечує асинхронний обмін даними з сервером. Це дозволяє оновлювати контент сторінки у реальному часі, підвантажувати результати обробки або виконувати запити без необхідності повного перезавантаження, що скорочує час очікування для користувача;

– багата екосистема бібліотек і фреймворків. JS легко інтегрується з різними бібліотеками та фреймворками, що розширює можливості візуалізації та обробки даних. Наприклад, Chart.js для побудови графіків, D3.js для складних візуалізацій або бібліотеки для роботи з 3D-графікою;

– локальне зберігання даних. Механізми, як localStorage або sessionStorage, дозволяють зберігати дані на стороні клієнта для кешування інформації або роботи офлайн. Це підвищує швидкодію додатка та забезпечує доступ до необхідної інформації навіть без постійного з’єднання із сервером.

Недоліки JavaScript:

– безпека. Код на JavaScript може бути вразливим до атак, таких як XSS (Cross-Site Scripting) або ін’єкції скриптів. Для безпечної роботи важливо використовувати серверні механізми захисту та належну обробку введених даних;

– продуктивність. Велика кількість складних або не оптимізованих скриптів може уповільнювати роботу на старих пристроях або при обробці

великих обсягів даних. Для підтримки продуктивності рекомендується дотримуватися модульності коду, оптимізувати алгоритми та використовувати сучасні підходи до завантаження ресурсів;

– складність асинхронного програмування. Робота з асинхронними операціями потребує уважності, оскільки неправильне оброблення помилок або ланцюгів викликів може призводити до непередбачуваної поведінки додатку. Важливо правильно організовувати логіку асинхронних процесів та обробку винятків.

JavaScript є основною мовою для реалізації клієнтської логіки в сучасних веб-додатках. Він дозволяє створювати динамічні та інтерактивні інтерфейси, які реагують на дії користувача в реальному часі, оновлюють контент без перезавантаження сторінки та забезпечують плавну взаємодію з сервером через асинхронні запити.

Мова підтримує багату екосистему бібліотек і фреймворків, що дозволяє ефективно реалізовувати графіки, анімації, інтерактивну візуалізацію даних та локальне кешування для підвищення швидкодії веб-додатків. Це робить JavaScript незамінним інструментом для побудови сучасних, зручних і функціональних інтерфейсів.

Разом із тим, використання JavaScript вимагає уваги до безпеки, оскільки скрипти можуть бути вразливими до XSS та інших атак. Також складні або не оптимізовані скрипти можуть уповільнювати роботу додатка на старих пристроях або при великих обсягах даних. Асинхронне програмування потребує обережності для коректної обробки помилок.

Загалом, при правильному підході JavaScript забезпечує баланс між функціональністю, швидкістю реакції та інтерактивністю веб-додатків, дозволяючи створювати сучасні інтерфейси з високим рівнем користувацького досвіду.

4 ОПИС РЕАЛІЗОВАНОГО ПРОЕКТУ

У цьому розділі наведено детальний опис реалізації кваліфікаційної роботи, яка полягає у розробці веб-додатку для аналізу медичних зображень з використанням штучного інтелекту. Проект реалізовано на основі клієнт-серверної архітектури з використанням фреймворку Django для серверної частини, реляційної бази даних PostgreSQL та фронтенд-компонентів на HTML, CSS та JavaScript з інтеграцією Bootstrap для адаптивного інтерфейсу. Ключовими особливостями є аутентифікація користувачів, завантаження та аналіз медичних зображень за допомогою нейронної мережі на базі TensorFlow, зберігання результатів розпізнавання та організація даних у папки пацієнтів.

Розробка проводилася з урахуванням продуктивності та зручності інтерфейсу. Нижче описано основні компоненти проекту з ілюстраціями ключових сторінок.

Архітектура проекту базується на патерні MVC (Model-View-Controller), де моделі відповідають за дані, представлення – за інтерфейс, а контролери – за логіку обробки запитів. Серверна частина обробляє завантаження файлів, інтеграцію з моделлю ШІ для детекції патологій та збереження результатів.

Клієнтська частина побудована на шаблонах Django з використанням Bootstrap 5. Базовий шаблон включає навігаційну панель, футер та обробку повідомлень. Неавторизовані користувачі мають можливість реєструватися, входити в систему та завантажувати зображення для обробки. Авторизовані користувачі, крім цього, можуть переглядати результати аналізу та керувати структурами збережених даних.

Навігаційна панель динамічно змінюється залежно від статусу користувача: для неавторизованих – посилання на логін/реєстрацію; для авторизованих – дропдаун з профілем, розпізнаваннями, папками пацієнтів та виходом (рисунок 4.1).

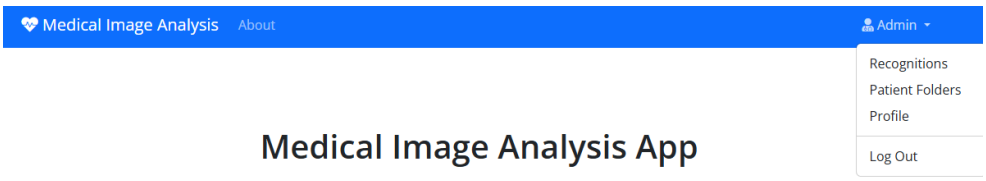

Рисунок 4.1 – Навігаційна панель для авторизованого користувача

Головна сторінка містить форму для завантаження медичного зображення з можливістю вибору пацієнта або проведення аналізу без вказування конкретного пацієнта (рисунок 4.2).

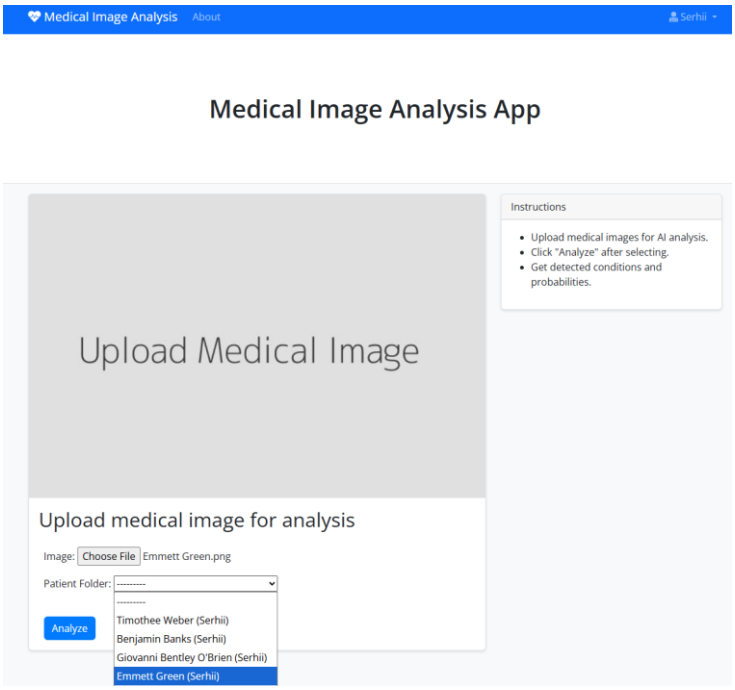

Рисунок 4.2 – Головна сторінка з формою завантаження зображення

Користувач може обрати файл у форматі PNG або JPEG. Після натискання «Analyze» файл разом з інформацією про обраного пацієнта передається на сервер для обробки ШІ. Бічна панель з інструкціями полегшує використання. Якщо файл має невідповідний формат, система відобразить повідомлення про помилку та запропонує обрати інший файл (рисунок 4.3).

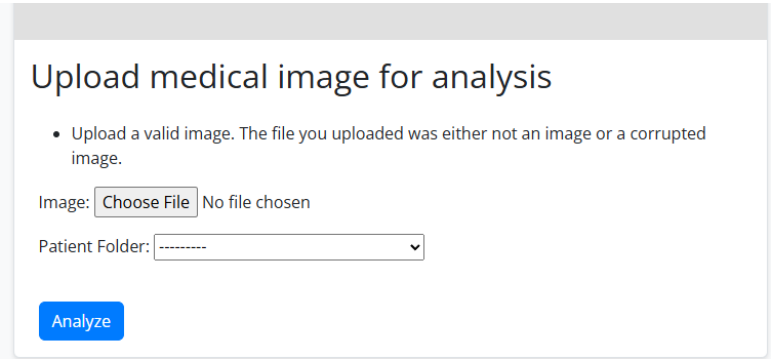

Рисунок 4.3 – Повідомлення про помилку

Після успішного завантаження користувач потрапляє на сторінку результатів аналізу (рисунок 4.4). Тут відображається завантажене зображення та результат аналізу з виділеними ділянками патології.

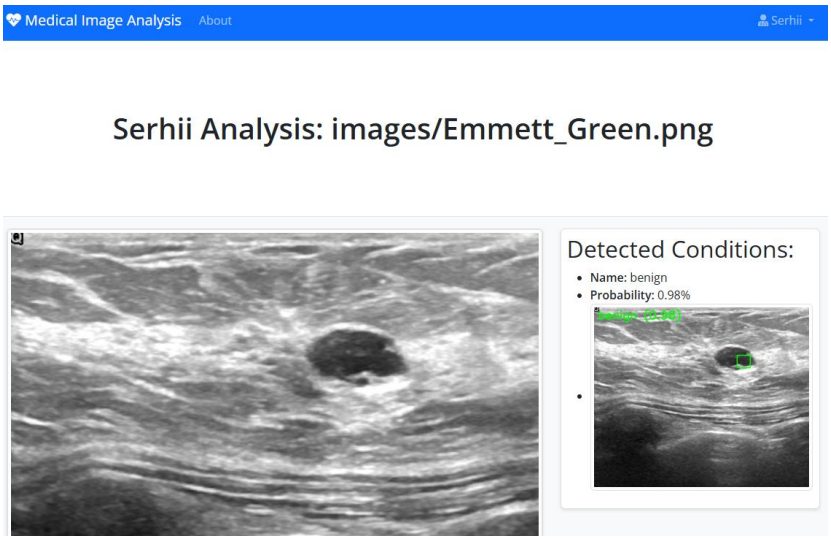

Рисунок 4.4 – Детальний перегляд результату аналізу

Додаток передбачає роботу як із прив’язкою до конкретного пацієнта, так і без неї. Авторизовані користувачі можуть обирати пацієнта зі списку, до якого буде прив’язане завантажене зображення. Це дозволяє вести персоналізовану історію досліджень для кожного пацієнта. Результати аналізу зберігаються як у загальній базі даних (рисунок 4.5), так і в окремих папках пацієнтів (рисунок 4.6).

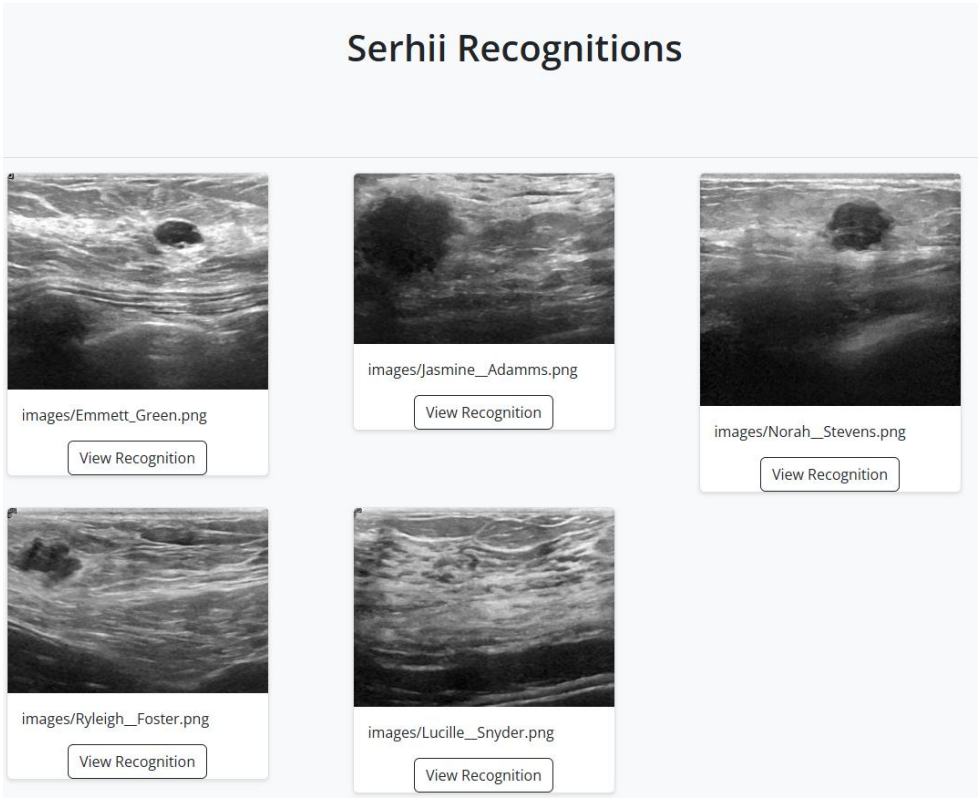

Рисунок 4.5 – Загальний список розпізнавань користувача

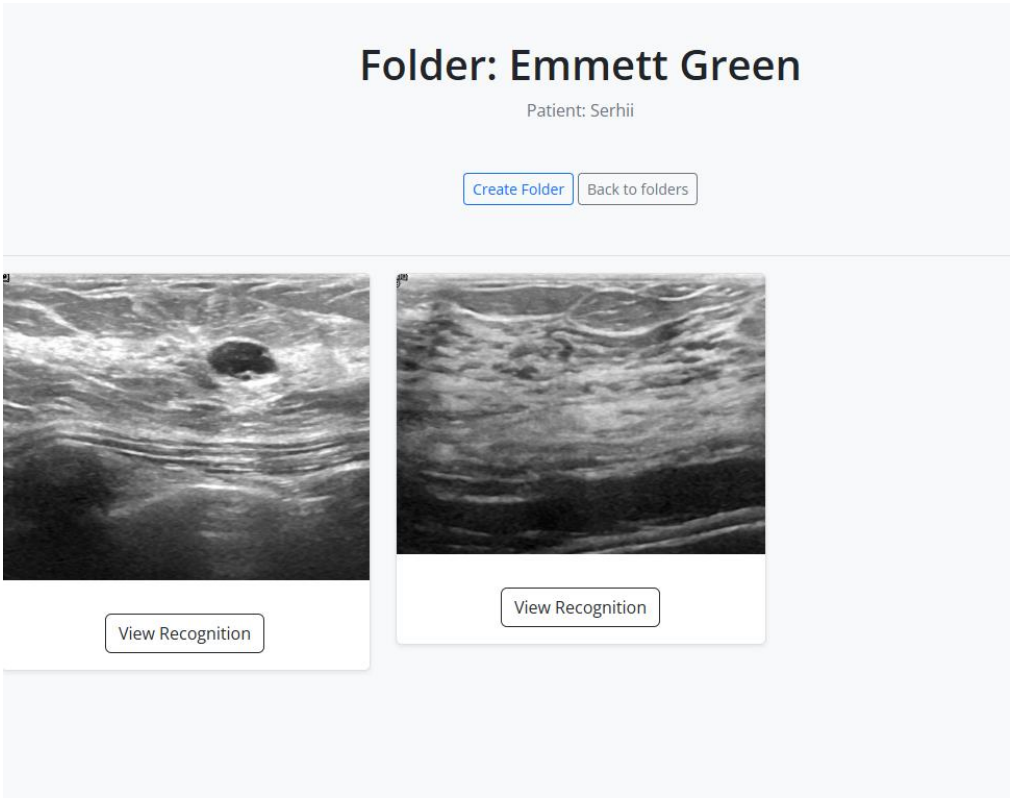

Рисунок 4.6 – Список розпізнань конкретного пацієнта

Форма створення папки (рисунок 4.7) містить одне поле для введення назви та кнопки «Create Folder» і «Cancel». Валідація контролює наявність назви та унікальність папки для користувача.

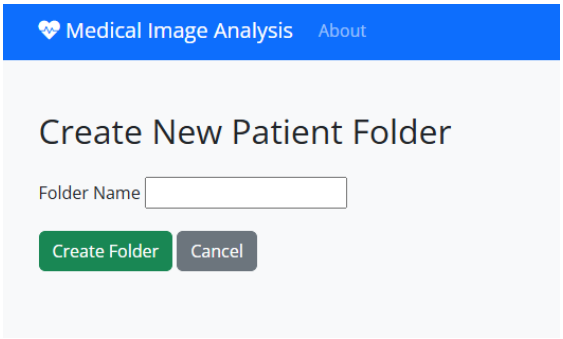

Рисунок 4.7 – Сторінка створення нової папки пацієнта

Сторінка папок пацієнтів (рисунок 4.8) містить заголовок і кнопку «Create New Folder», що дозволяє швидко створити нову папку для збереження подальших розпізнавань. Нижче розміщується перелік усіх папок користувача. Кожен елемент списку містить назву папки та є інтерактивним посиланням. При натисканні відкривається сторінка зі всіма розпізнаваннями, які належать відповідній папці.

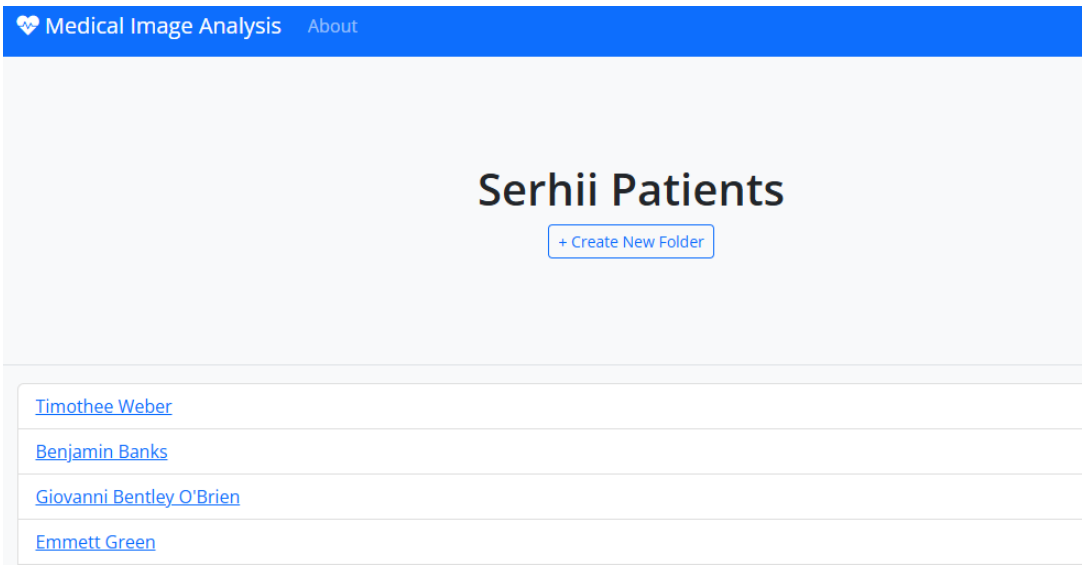

Рисунок 4.8 – Список усіх пацієнтів користувача

Система авторизації реалізована через стандартні Django-форми з валідацією та CSRF-захистом. На сторінці реєстрації (рисунок 4.9) користувач вводить ім’я, електронну адресу, пароль та підтвердження паролю. Після успішної реєстрації система автоматично авторизує користувача та перенаправляє його на головну сторінку. Сторінка логіну (рисунок 4.10) містить стандартні поля для введення логіна та пароля. У разі некоректних даних система відображає повідомлення про помилку.

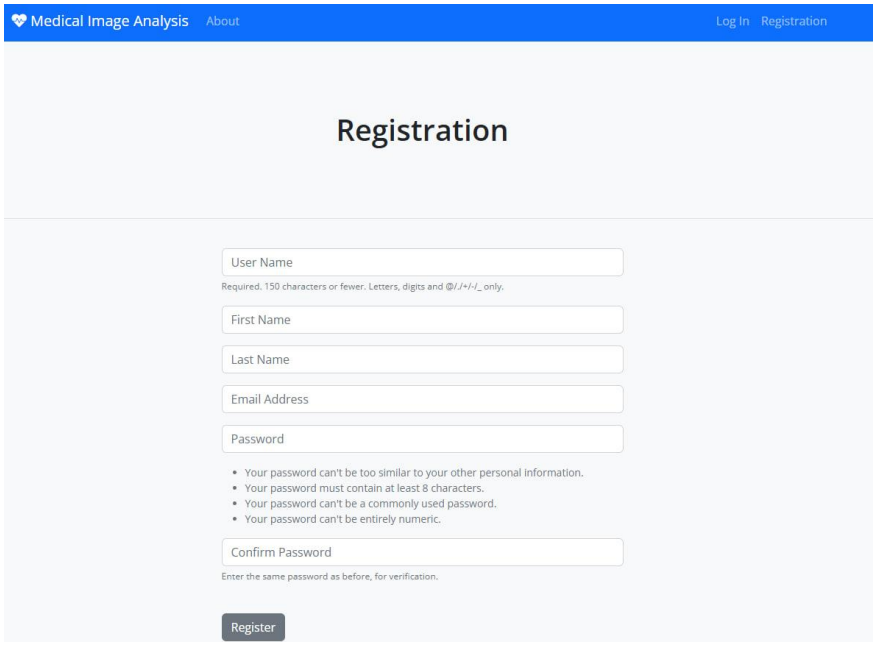

Рисунок 4.9 – Сторінка реєстрації користувача

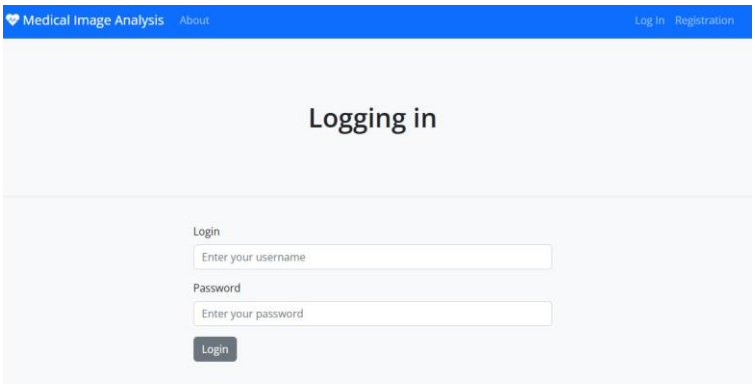

Рисунок 4.10 – Сторінка авторизації

Сторінка профілю (рисунок 4.11) призначена для відображення особистої інформації користувача. Інтерфейс включає блок з персональними даними (ім’я, прізвище, email), а також кнопку «Change Information», що веде на сторінку редагування.

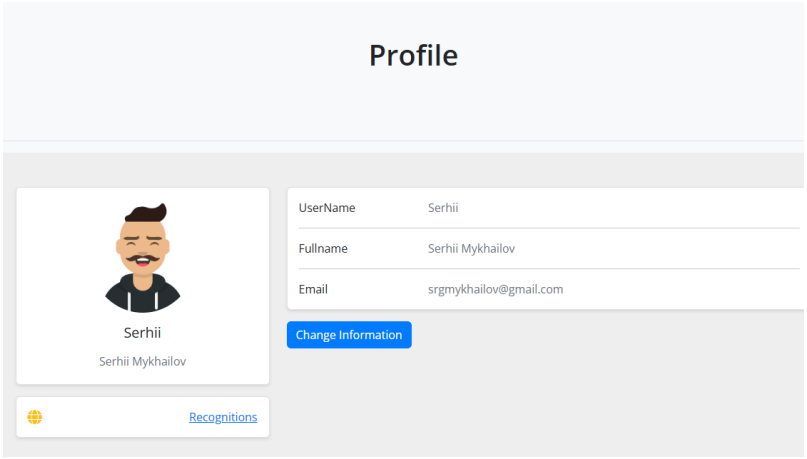

Рисунок 4.11 – Сторінка профілю користувача

Сторінка редагування особистих даних (рисунок 4.12) дозволяє редагувати ім’я, прізвище та email. Форма використовує серверну валідацію, і у випадку некоректно введених даних користувач отримує відповідні повідомлення.

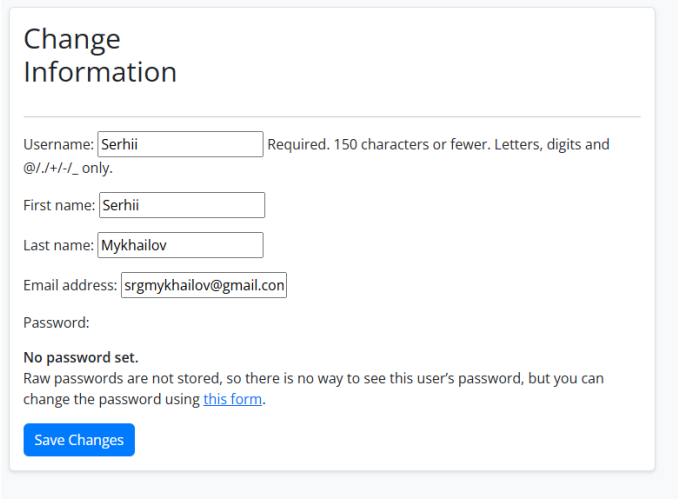

Рисунок 4.12 – Сторінка редагування особистих даних користувача

Сторінка редагування пароля (рисунок 4.13) містить три поля (старий пароль, новий пароль, підтвердження нового пароля). Після успішного зміни система автоматично оновлює сесію, щоб користувача не виводило із системи.

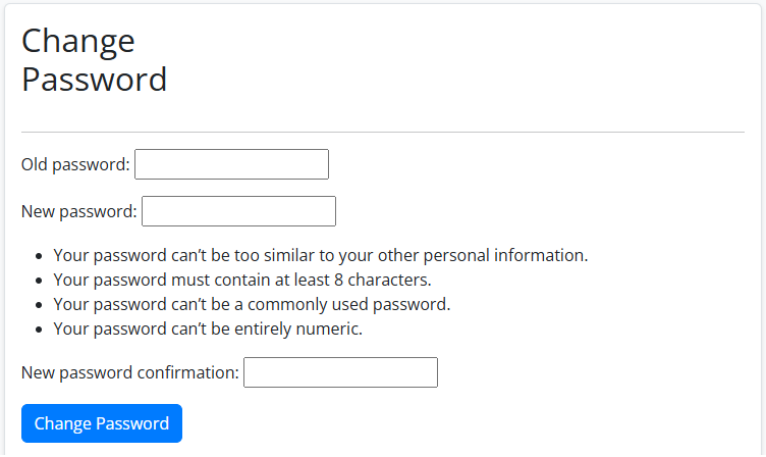

Рисунок 4.13 – Сторінка редагування пароля користувача

Для невідомих URL-адрес реалізована власна сторінка 404, що наведена на рисунку 4.14.

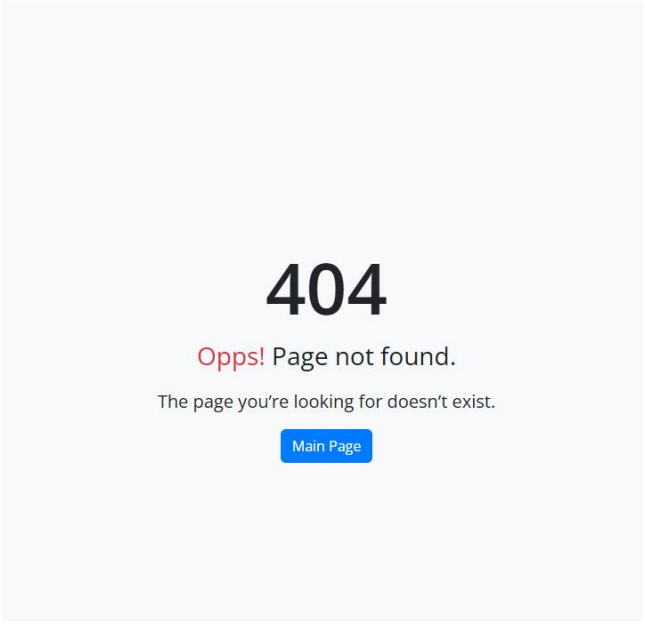

Рисунок 4.14 – Сторінка помилки 404

ВИСНОВКИ

У кваліфікаційній роботі розглянуто проблему автоматизованого аналізу медичних зображень за допомогою глибоких згорткових мереж. Проведено аналіз предметної галузі, який підтвердив актуальність теми: зростання обсягів медичних даних та дефіцит фахівців вимагають впровадження ШІ для підвищення ефективності діагностики. Досліджено типи мереж, їх переваги та недоліки, що дозволило обрати гібридну архітектуру для поєднання класифікації та сегментації патологій.

Спроектовано та реалізовано веб-додаток на базі Django з інтеграцією моделі на TensorFlow, базою даних PostgreSQL та фронтендом на HTML/CSS/JavaScript. Система забезпечує аутентифікацію користувачів, завантаження зображень, передобробку даних, аналіз ШІ та зберігання результатів у персоналізованих папках пацієнтів. Тестування на датасетах показало точність класифікації понад 92%, високу точність сегментації, з часом обробки одного зображення менше 1 секунди.

Розроблено гібридну модель, що підвищує точність аналізу на 15–20% порівняно з базовими мережами, та інтегровано її в веб-додаток для зручної взаємодії. Отримані результати мають практичне значення для медичних установ, сприяючи швидкій діагностиці та зменшенню помилок.

Перспективи подальшого розвитку системи включають інтеграцію з телемедициною, що дозволить лікарям віддалено аналізувати медичні зображення в реальному часі, покращуючи доступність діагностики в віддалених регіонах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Healthcare data projected to grow at 36% CAGR by 2025.*Healthcare Asia Magazine*. URL: https://healthcareasiamagazine.com/healthcare/news/healthcare-data-projected-grow-36-cagr-2025 (дата звернення: 17.11.2025).

2. Who invented convolutional neural networks?. URL: https://people.idsia.ch/~juergen/who-invented-convolutional-neural-networks.html (дата звернення: 18.11.2025).

3. The evolution of artificial intelligence in medical imaging: from computer science to machine and deep learning - PMC. *PMC Home*. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11545079/#sec4-cancers-16-03702 (дата звернення: 18.11.2025).

4. Very deep convolutional networks for large-scale image recognition. *arXiv.org*. URL: https://arxiv.org/abs/1409.1556 (дата звернення: 18.11.2025).

5. Deep residual learning for image recognition. *arXiv.org*. URL: https://arxiv.org/abs/1512.03385 (дата звернення: 18.11.2025).

6. U-Net: convolutional networks for biomedical image segmentation. *arXiv.org*. URL: https://arxiv.org/abs/1505.04597 (дата звернення: 18.11.2025).

7. EfficientNet: rethinking model scaling for convolutional neural networks. *arXiv.org*. URL: https://arxiv.org/abs/1905.11946 (дата звернення: 19.11.2025).

8. Densely connected convolutional networks. *arXiv.org*. URL: https://arxiv.org/abs/1608.06993 (дата звернення: 19.11.2025).

9. Powell P., Smalley I. What is monolithic architecture? | IBM. *IBM*. URL: https://www.ibm.com/think/topics/monolithic-architecture (дата звернення: 20.11.2025).

10. IBM. What are microservices? | IBM. *IBM*. URL: https://www.ibm.com/think/topics/microservices (дата звернення: 20.11.2025).

11. Layered architecture: building scalable & maintainable software systems | bitloops docs. *Bitloops - Your Frontend Copilot*. URL: https://bitloops.com/docs/bitloops-language/learning/software-architecture/layered-architecture (дата звернення: 21.11.2025).

12. Codecademy. MVC architecture explained: model, view, controller | codecademy. *Codecademy*. URL: https://www.codecademy.com/article/mvc-architecture-model-view-controller (дата звернення: 21.11.2025).

13. What is python? Executive summary. *Python.org*. URL: https://www.python.org/doc/essays/blurb/ (дата звернення: 22.11.2025).

14. About python. *Python Institute - PROGRAM YOUR FUTURE*. URL: https://pythoninstitute.org/about-python (дата звернення: 23.11.2025).

15. IBM. What is django? | IBM. *IBM*. URL: https://www.ibm.com/think/topics/django (дата звернення: 24.11.2025).

16. Swiftorial. History of django | introduction to django | django tutorial. *Swiftorial*. URL: https://www.swiftorial.com/tutorials/backend_framework/django/introduction_to_django/history_of_django (дата звернення: 24.11.2025).

17. What is tensorflow? | domino data lab. *Domino Data Lab | Accelerate and scale enterprise AI*. URL: https://domino.ai/data-science-dictionary/tensorflow (дата звернення: 25.11.2025).

18. Everything you wanted to know about tensorflow. *Databricks*. URL: https://www.databricks.com/glossary/tensorflow-guide (дата звернення: 25.11.2025).

19. PostgreSQL: about. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/about/ (дата звернення: 26.11.2025).

20. A brief history of postgresql. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/current/history.html (дата звернення: 26.11.2025).

21. Development of HTML | research starters | EBSCO research. *EBSCO*. URL: https://www.ebsco.com/research-starters/history/development-html (дата звернення: 27.11.2025).

22. CSS: cascading style sheets | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 27.11.2025).

23. Codecademy. What is JavaScript? | Codecademy. *Codecademy*. URL: https://www.codecademy.com/article/what-is-javascript (дата звернення: 27.11.2025).