НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

_____ /Голуб Б.Л., доц., к.т.н. /

підпис ПІБ, вчене звання і ступінь

«_____»_____ р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення системи визначення діагнозу людини»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ / Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : _____ / Семко В.В/
підпис ПІБ

Виконав: _____ / Толстих М.Ю./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____/ Голуб Б.Л., доцент, к.т.н /
підпис

" " _____ _____ р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Толстих Максиму Юрійовичу

Спеціальність 121 «Інженерія програмного забезпечення»

Тема роботи: Програмне забезпечення системи визначення діагнозу людини

Затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру _____ . _____. _____
рік, місяць, число

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____/ Семко В.О. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____/ Толстих М.Ю. /
підпис ініціали та прізвище

Дата отримання завдання _____._____._____
рік, місяць, числ

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. UML - Unified Modeling Language
2. SVR - Support Vector Regression
3. SMV - Support Vector Machine
4. API - Application Programming Interface
5. SMOTE - Synthetic Minority Over-sampling Technique

ВСТУП

В умовах розвитку цифрових технологій, зростання потреб у підвищенні ефективності медичного обслуговування, особливої актуальності набуває створення інтелектуальних інформаційних систем, що здатні здійснювати попередній аналіз стану здоров’я людини.

З огляду на постійне зростання навантаження на систему охорони здоров’я України, зокрема внаслідок військових дій, соціально-економічних викликів і дефіциту медичних кадрів, впровадження автоматизованих рішень для попередньої діагностики є надзвичайно важливим.

Основною метою цієї роботи є розробка програмного забезпечення на основі введених користувачем симптомів, визначити можливі діагнози з використанням алгоритмів машинного навчання. У межах виконання роботи здійснюється аналіз сучасних технологій, обґрунтовується вибір інструментальних засобів, проєктується архітектура системи, окремо навчаються моделі машинного навчання з попередньо оброблених даних та створюється функціональний прототип.

Основним завданням бакалаврської кваліфікаційної роботи є розробка програмного забезпечення системи визначення діагнозу людини.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- вивчити та проаналізувати предметну область та зібрати інформацію для подальшого застосування у програмному продукті;
- проаналізувати існуючі розробки у цій галузі;
- сформулювати функціональні вимоги до системи, що розробляється;
- побудувати моделі бізнес-процесів у вигляді UML діаграм;
- скласти загальний алгоритм роботи системи;
- провести аналіз засобів розробки, обрати необхідний інструментарій для розробки інформаційної управляючої системи;
- розробити графічний інтерфейс;
- розробити структуру бази даних;

- розробити базу даних;
- розробити програмний продукт;
- провести тестування розробленої системи.

У ході розробки даного програмного забезпечення системи основними завданнями є: створити програмний продукт, який дозволяє ввівши симптоми, попередньо спрогнозувати діагноз на основі введених симптомів. Основний фокус приділяється процесу навчання моделей штучного інтелекту, тому інтерфейс програми буде, простим та зручним для користувача. Бо всі складні процеси відбуваються під час навчання моделі. А інтерфейс буде лише використовувати цю навчену модель штучного інтелекту для створення прогнозу захворювання.

Результат досліджень можна буде використати у майбутньому, як основу для створення більш просунутої, більш вузько направленої системи медичної інформаційної системи, тобто націленої на якийсь один певний медичний напрямок, наприклад хвороби серця. Тобто обробка ультразвукової діагностики, симптомів і формування звіту з прогнозом діагнозу.

В розробці проєкту було використано сучасні технологічні рішення та технології. Наприклад мову програмування Python, React.JS, Tailwind CSS, JavaScript.

Пояснювальна записка представлена чотирма розділами. В першому розділі описується аналіз предметної області, огляд існуючих рішень та формується постановка задачі. У другому розділі відбувається проєктування програмного забезпечення з використанням UML діаграм та моделювання логічної моделі даних за допомогою ER діаграм. Третій розділ це опис технічних аспектів реалізації програмної системи. Четвертий розділ має на меті визначити вимоги до апаратних вимог, тестування та впровадження системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасний рівень розвитку медицини та інформаційних технологій відкриває нові перспективи для підвищення доступності та ефективності попередньої діагностики захворювань людини. Зокрема, інтеграція інформаційних систем із технологіями штучного інтелекту дозволяє створювати інструменти, які здатні аналізувати симптоми, введені користувачем, і пропонувати ймовірні діагнози з високим рівнем точності. Одним із перспективних напрямків у цьому контексті є розробка інтелектуальних програмних систем, які використовують алгоритми машинного навчання для класифікації симптомів і прогнозування можливих захворювань. Такі системи стають особливо актуальними в умовах, коли традиційні методи діагностики недоступні через географічні, економічні чи соціальні фактори.

У більшості випадках своєчасне виявлення хвороби має значний вплив на ефективність лікування. Але існують обставини, коли швидке звернення до лікаря є недоступним або неможливим через віддаленість медичних закладів, брак спеціалістів або обмеженість ресурсів. У таких умовах інформаційні системи попереднього діагностування можуть допомогти користувачеві швидко оцінити власний стан здоров’я та прийняти рішення щодо подальших дій.

Предметна область охоплює процес попереднього визначення діагнозу людини на основі введення симптомів через спеціалізоване програмне забезпечення. Система має допомагати користувачам у виявленні ймовірних захворювань шляхом аналізу сукупності симптомів, які вводяться вручну, та використання алгоритмів машинного навчання для обробки цих даних.

Основними користувачами системи є люди, які бажають швидко отримати попередню інформацію про можливі діагнози, щоб прийняти рішення про подальшу консультацію в лікаря чи виконання певних профілактичних заходів.

Система не замінює професійну медичну консультацію, проте виконує роль помічника, який орієнтує користувача у напрямку розуміння свого стану здоров’я.

Основні процеси предметної області включають:

- введення симптомів за допомогою інтерактивної форми;
- обробка введених даних за допомогою попередньо навченої моделі машинного навчання;
- формування списку можливих діагнозів із вказаними рівнями ймовірності;
- можливості перегляду короткої інформації про захворювання.

Інформаційна база для роботи системи формується на основі структурованих даних про симптоми та відповідні їм захворювання, що дозволяє моделі ефективно здійснювати класифікацію введених даних.

Розробка такої системи вимагає врахування не лише технічних аспектів (вибору алгоритмів машинного навчання, побудови бази даних), але й етичних вимог, зокрема правильного інформування користувача про обмеження використання автоматизованої діагностики.

Таким чином, предметна область розглянутої задачі є міждисциплінарною, поєднуючи знання в галузях інформатики, медицини, аналітики даних та розробки користувацьких інтерфейсів. Програмне забезпечення, що розробляється в межах цієї роботи, спрямоване на полегшення процесу попередньої оцінки стану здоров’я, підвищення рівня проінформованості населення та створення основи для подальшого розвитку більш складних медичних інформаційних систем. У перспективі результати цієї роботи можуть бути використані для створення спеціалізованих систем, наприклад, для діагностики захворювань певного типу, таких як серцево-судинні патології чи інфекційні хвороби, що сприятиме підвищенню якості медичного обслуговування.

1.2 Аналіз вимог до програмної системи

Розділ "Аналіз вимог до програмної системи" є фундаментальним етапом розробки, який визначає функціональні та нефункціональні вимоги до системи попереднього визначення діагнозів на основі симптомів. Цей аналіз базується на потребах цільової аудиторії, специфіці предметної області та обмеженнях, пов’язаних із технічною реалізацією та використанням системи. Метою цього етапу є формування чіткого набору вимог, які слугуватимуть основою для проєктування, розробки та тестування програмного забезпечення, а також забезпечать його відповідність реальним потребам користувачів у контексті доступності медичних консультацій.

Функціональні вимоги описують основні можливості системи, які вона має надавати користувачам. Основною функцією є прогнозування ймовірних діагнозів на основі введених симптомів, що реалізується через наступні компоненти:

1. Модуль тренування моделей машинного навчання:
   - Тренування моделей машинного навчання за допомогою алгоритмів SVR, Random Forest, Naive Bayes на основі вхідного набору медичних даних.
   - Збереження навчених моделей у форматі, придатному для подальшого завантаження на серверну частину.
   - Можливість перенавчання моделей на нових даних.
2. Бекенд-сервер
   - Завантаження попередньо натренованих моделей при старті.
   - Прийом вхідних даних (списку симптомів) від користувача через API-запити.
   - Виконання обробки даних (нормалізація або векторизація симптомів, тощо).
   - Подача запиту до відповідної моделі та отримання прогнозу діагноз.
   - Формування відповіді у зручному для користувача форматі.

- Надання API для інтеграції з фронтендом, тобто React.

3. Фронтенд-система на React:
   - Надання користувачу інтерфейсу для введення симптомів.
   - Передавання введених даних на бекенд через HTTP-запити, тобто REST API.
   - Отримання результатів прогнозу від бекенду та відображення їх у вигляді списку можливих діагнозів.
   - Виведення короткої інформації про можливі діагнози за запитом користувача.

Нефункціональні вимоги визначають якість, продуктивність, надійність і зручність використання системи. Вони є критичними для забезпечення її практичної цінності, особливо в умовах обмеженого доступу до медичних послуг:

Продуктивність: Система повинна обробляти запит на прогнозування протягом не більше 2 секунд при стандартному навантаженні (до 50 запитів на хвилину). Це забезпечить швидку відповідь для користувачів, навіть у віддалених регіонах із повільним інтернетом.

Надійність: Система має стабільно працювати при навантаженні до 50 одночасних запитів, із мінімальною кількістю помилок (менше 1% відмов). Для цього серверна частина буде розгорнута на хмарній платформі з резервним копіюванням даних.

Зручність використання (юзабіліті): Інтерфейс має бути інтуїтивно зрозумілим для користувачів із різним рівнем технічної підготовки. Це включає адаптивний дизайн для мобільних пристроїв, підтримку української мови та відсутність складних дій для введення симптомів чи перегляду результатів.

Безпека: Система повинна забезпечувати конфіденційність даних користувачів, уникаючи їхнього збереження на сервері. Усі запити мають передаватися через шифрований протокол HTTPS, а дані симптомів видалятися після завершення сесії.

Сумісність: Система має бути доступною на різних платформах (десктопні браузери Chrome, Firefox, Edge, а також мобільні браузери на Android і iOS) без необхідності встановлення додаткового програмного забезпечення.

Аналіз вимог дозволяє сформулювати чітке технічне завдання для створення програмної системи визначення діагнозу людини. Відповідно до поставлених вимог, реалізація системи повинна забезпечити баланс між швидкістю обробки запитів, точністю прогнозування, простотою використання та можливістю масштабування.

1.3 Моделювання предметної області

Моделювання предметної області є важливим етапом розробки програмного забезпечення системи визначення діагнозу людини, оскільки воно дозволяє чітко визначити основні компоненти системи, їх взаємодію та поведінку в різних сценаріях використання. У контексті цього проєкту моделювання допомагає формалізувати процес діагностики на основі введених симптомів, враховуючи як дії користувачів (пацієнтів, адміністраторів), так і внутрішню логіку системи, включаючи обробку даних і прогнозування за допомогою алгоритмів машинного навчання. Для цього було використано мову моделювання UML (Unified Modeling Language), яка забезпечує стандартизований підхід до візуалізації структури та поведінки системи.

Основною метою моделювання є створення чіткого уявлення про предметну область, що охоплює взаємодію між користувачами, системою та її компонентами. Це дозволяє:

- Визначити ключові ролі (акторів), які взаємодіють із системою.
- Описати основні сценарії використання (прецеденти), такі як введення симптомів і отримання діагнозів.
- Встановити послідовність дій між компонентами системи для виконання запитів користувача.
- Забезпечити основу для подальшого проєктування архітектури та розробки.

Моделювання проводилося з урахуванням принципів абстрагування та багатомодельності, тобто використання різних типів діаграм для повного опису системи. Це дозволяє врахувати як статичну структуру (наприклад, зв’язки між сутностями), так і динамічну поведінку.

Діаграма прецедентів (рис. 1.1) описує основні сценарії взаємодії користувачів із системою. Вона включає два основні актори: "Пацієнт" і "Лікар", а також їхні дії в межах системи. Пацієнт може повідомити про симптоми та переглянути звіт про діагноз, тоді як лікар відповідає за перегляд симптомів від пацієнта, їхній аналіз і створення звіту про діагноз. Ця діаграма допомагає чітко визначити ролі користувачів і функціональні можливості системи.

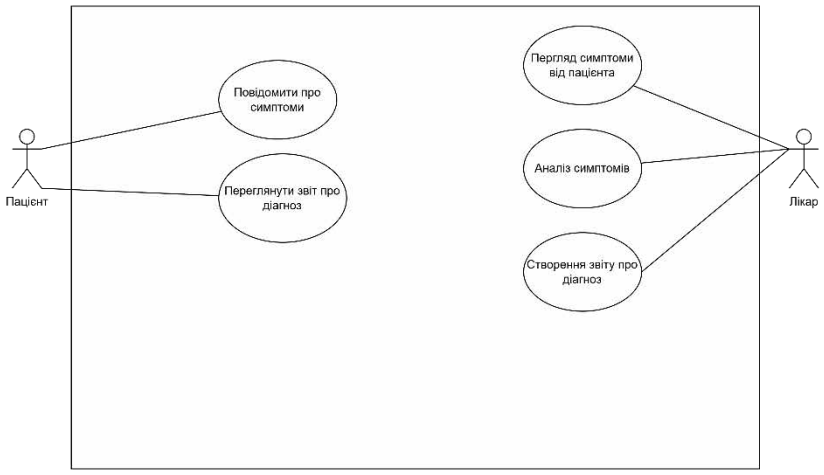

Рис. 1.1 – Діаграма прецедентів предметної області

Діаграма послідовності (рис. 1.2) ілюструє динаміку взаємодії між акторами та системою в часі. Вона деталізує процес, який починається з повідомлення про симптоми від пацієнта. Лікар аналізує отримані дані, може запросити додаткову інформацію (наприклад, історію хвороби), видає попередній діагноз і завершує процес остаточним діагнозом. Ця діаграма підкреслює циклічність і зворотний зв’язок, що є характерним для медичної діагностики, а також показує послідовність обміну повідомленнями між учасниками.

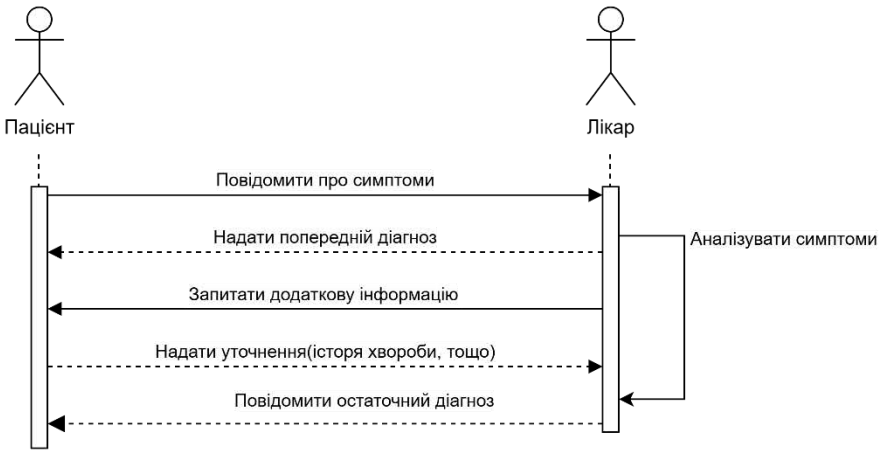

Рис. 1.2 – Діаграма послідовності

Моделювання також враховує специфіку предметної області, зокрема необхідність інтеграції медичних даних із технологіями машинного навчання. Наприклад, система повинна коректно обробляти бінарні ознаки симптомів (наявність/відсутність) і забезпечувати точність прогнозів, що вимагає чіткого визначення логіки взаємодії між даними та алгоритмами. Крім того, інтерфейс має бути адаптований до потреб користувачів із різним рівнем технічної підготовки, що відображається в інтуїтивному дизайні прецедентів.

Таким чином, моделювання предметної області за допомогою UML забезпечує комплексне розуміння системи, поєднуючи технічні та медичні аспекти. Діаграми прецедентів і послідовності (рис. 1.1 і рис. 1.2) чітко визначають ролі акторів, сценарії взаємодії та хронологію процесів. У подальших етапах планується розробка додаткових діаграм, таких як діаграма класів для опису структури даних і діаграма діяльності для деталізації бізнес-процесів. Цей підхід закладає міцний фундамент для створення системи, яка буде ефективною, гнучкою та придатною для подальшого розширення, наприклад, для інтеграції з медичними базами даних чи спеціалізованими діагностичними модулями.

1.4 Аналіз існуючих систем

Аналіз існуючих систем є одним із ключових етапів розробки програмної системи, оскільки дозволяє оцінити вже наявні рішення, їхні сильні та слабкі сторони, а також визначити напрямки для вдосконалення власного проєкту. У контексті попереднього діагностування захворювань на основі машинного навчання було проаналізовано кілька відомих систем, які використовують подібні підходи. Цей аналіз допоміг сформувати вимоги до розроблюваної системи та уникнути типових помилок, виявлених у конкурентів.

Першою з таких систем є IBM Watson Health [1], після 2022 року відома як Merative LP. Ця платформа є потужним інструментом штучного інтелекту, розробленим для аналізу медичних даних, включаючи електронні медичні записи, клінічні дослідження та медичні зображення. Watson Health використовує алгоритми машинного навчання для підтримки лікарів у діагностиці захворювань, таких як рак, і пропонує рекомендації щодо лікування. Однак система стикається з обмеженнями, пов’язаними з необхідністю великих обсягів якісних даних для тренування моделей, а також із високою вартістю впровадження, що робить її менш доступною для невеликих медичних закладів чи приватних користувачів. Крім того, її ефективність залежить від точності введених даних, а помилки в аналізах можуть виникати через недостатню адаптацію до локальних медичних стандартів, зокрема в Україні.

Наступна велика модель штучного інтелекту розроблена компанією Google це Google Health (Med-PaLM) [2]. Система дозволяє лікарям на основі симптомів отримувати попередні діагнози та пропозиції щодо лікування. Google Health інтегрується з іншими сервісами для забезпечення повної картини стану здоров’я пацієнта. Згідно дослідженням, які були оприлюднені Google, Med-PaLM 2 все ще має невеликі проблеми з точністю порівняно з відповідями інших лікарів. Проте за такими показниками, як докази аргументованості чи відсутність ознак неправильного розуміння, Med-PaLM 2 показав більш-менш такі ж результати, як і реальні лікарі. Основні переваги це велика точність через використання великих потужностей Google, постійні оновлення. До недоліків

можна віднести високі вимоги для запуску моделі, тобто потрібно мати потужний сервер, щоб модель працювала.

Ще одна цікава система це Human DX[3], вона є чимось подібним на соціальну мережу між лікарями, ціла платформа для консультацій, яка в собі також використовує штучний інтелект. Це платформа, яка дозволяє лікарям взаємодіяти між собою і отримувати рекомендації щодо діагнозів на основі аналізу симптомів та медичних історій пацієнтів. Система також може давати пропозиції щодо лікування на основі синтезу знань від численних спеціалістів.

Усі проаналізовані системи мають спільну рису — використання машинного навчання для обробки медичних даних, але кожна з них стикається з викликами, такими як потреба в великих обсягах даних, адаптація до локальних умов і забезпечення конфіденційності. На основі цього аналізу було визначено, що розроблювана система має бути простішою у використанні для кінцевого користувача, не залежати від великих централізованих баз даних і враховувати специфіку медичної практики в Україні, зокрема доступність і швидкість роботи в умовах обмежених ресурсів. Це включатиме створення легкого веб-інтерфейсу, використання відкритих наборів даних для тренування моделей і забезпечення модульності для подальшого розширення функціоналу.

1.5 Постановка задачі

Система визначення діагнозу, має допомогти людям, які бажають попередньо оцінити стан свого здоров’я, на основі введених даних. А також лікарям, щоб пришвидшити процес прийняття рішень. Система має надати можливість швидкого аналізу вхідних даних та визначення ймовірних діагнозів для подальшої консультації.

Основні цілі програми:

- спростити процес попередньої діагностики для користувачів;
- забезпечити швидкий та зручний спосіб введення симптомів;
- здійснювати швидкий аналіз симптомів з пропозицією найбільш вірогідних захворювань;

- підвищити обізнаність людей щодо можливих захворювань.

Процес роботи діагностики захворювання часто пов’язаний з певними незручностями, такими як:

- з великою кількістю медичної інформації, яку необхідно аналізувати;

- велика кількість можливих варіантів інтерпретації симптомів;

- ризик людських помилок при інтерпретації симптомів.

Програмне забезпечення яке розроблюється, не зможе в повній мірі замінити лікаря і його експертну оцінку, але зможе надати користувачу можливість оцінити потенційні ризики та визначити необхідність подальшого звернення до конкретного спеціаліста. А лікарю надасть можливість з усіх можливих діагнозів звузити коло, до кількох найбільш підходящих.

Основні функції системи:

- введення симптомів за допомогою інтерактивної форми;

- обробка введених даних за допомогою попередньо навченої моделі машинного навчання;

- формування списку можливих діагнозів із вказаними рівнями ймовірності;

- можливості перегляду короткої інформації про захворювання.

Система в собі не зберігає ніяких даних користувача, а є сервісом запитав-отримав.

Система має забезпечувати:

- введення, редагування, видалення списку симптомів;
- виведення запропонованих діагнозів за рівнем ймовірності;
- перегляд короткої інформативної інформації.

Програма має бути інтуїтивно зрозумілою, гнучкою до розширення бази знань, і здатна на до навчання моделі на нових даних. Також ця програма може бути базою для створення більш складних моделей, наприклад навчених на анонімному анамнезі пацієнтів, для більш точного визначення діагнозу, а не лише за сталим списком симптомів, як в даній програмі. Тобто буде враховувати багато факторів, які можуть вплинути на рішення моделі щодо діагнозу.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Загальна архітектура програмної системи

Розроблювана система визначення діагнозу людини базується на клієнт-серверній архітектурі з розділеним функціоналом: модуль машинного навчання та серверна частина для обробки запитів від користувача. Так як кожен з компонентів виконує окремі функції і під час роботи не перетинається один з одним, це забезпечує гнучкість системи, що дозволяє оновлювати компоненти незалежно один від одного.

Основні компоненти:

1. Модуль тренування моделей машинного навчання – забезпечує попередню обробку вхідних даних, формування навчального набору даних та тренування моделей на основі алгоритмів інтелектуального аналізу. Результатом цієї системи є збережені моделі, які готові до використання в іншому модулі.
2. Модуль обробки запитів та взаємодії з користувачем – надає зручний інтерфейс, для введення симптомів та перегляд результату прогнозування. Також відповідає за логіку обробки запитів з клієнта, завантаження моделей, та формування відповіді.

Взаємодія між компонентами відбувається послідовно. На етапі тренування моделей система обробляє набір медичних прикладів, у яких кожен рядок містить інформацію про наявність симптомів та відповідний діагноз. Після побудови моделей ці результати зберігаються й використовуються підсистемою обробки запитів для аналізу нових вхідних даних. Кінцевий користувач взаємодіє лише з інтерфейсом, який надсилає запити та отримує результати від сервісної частини.

Такий підхід дозволяє відокремити логіку аналізу даних від механізмів взаємодії з користувачем, що покращує підтримку, тестування та розвиток системи.

2.2 Логічна модель даних у вигляді ER-діаграми

У класичних інформаційних системах, логічна модель даних подається у вигляді ER- діаграми, яка відображає структуру бази даних та зв’язки між таблицями [6]. Але в межах цього проєкту, не використовується реляційна база даних. Вся обробка інформації відбувається з попередньо підготовлених навчальних та тестових наборів даних та серіалізованих моделей машинного навчання.

Незважаючи на відсутність БД, логічна структура даних все ж присутня, і її можна подати у вигляді дуже спрощеної моделі даних, які відображає зв’язки між вхідними симптомами, моделями машинного навчання та вихідними діагнозами.

Доцільно буде подати їх у вигляді двох окремих діаграм, які опишуть логічну модель даних в цих двох частинах проєкту:

1. Частина тренування моделей;
2. Частина, що забезпечує відображення та прогнозування результатів.

Розберемо ці частини по порядку, починаючи з частини навчання моделей, яка відповідає за використання алгоритмів машинного навчання і тренування моделей з навчального набору даних (рис 2.1). Вона містить набори даних у вигляді бінарних спів-відношень. Набір даних має 4920 записів, він містить в собі різноманітні варіації симптомів до діагнозів, а також 132 симптоми і 133 колонка це сам діагноз.

Як ми бачимо, це відображення файлу з набором даних для тренування моделі, він містить ознаки (симптоми) у вигляді бінарних значень (0 – відсутнє, 1 - присутнє). Кожен рядок у наборі даних представляє окремий випадок захворювання з присутніми симптомами. Поля виступають у вигляді аналогів атрибутів у базі даних

	A	B	C	D	DY	DZ	EA	EB	EC
1	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	inflammatory_nails	blister	red_sore_around_nose	yellow_crust_ooze	prognosis
2	1	1	1	0	0	0	0	0	Fungal infection
3	0	1	1	0	0	0	0	0	Fungal infection
4	1	0	1	0	0	0	0	0	Fungal infection
5	1	1	0	0	0	0	0	0	Fungal infection
6	1	1	1	0	0	0	0	0	Fungal infection
7	0	1	1	0	0	0	0	0	Fungal infection
8	1	0	1	0	0	0	0	0	Fungal infection
9	1	1	0	0	0	0	0	0	Fungal infection
10	1	1	1	0	0	0	0	0	Fungal infection
11	1	1	1	0	0	0	0	0	Fungal infection
12	0	0	0	1	0	0	0	0	Allergy
13	0	0	0	0	0	0	0	0	Allergy
14	0	0	0	1	0	0	0	0	Allergy
15	0	0	0	1	0	0	0	0	Allergy
16	0	0	0	1	0	0	0	0	Allergy
17	0	0	0	0	0	0	0	0	Allergy
18	0	0	0	1	0	0	0	0	Allergy
19	0	0	0	1	0	0	0	0	Allergy
20	0	0	0	1	0	0	0	0	Allergy
21	0	0	0	1	0	0	0	0	Allergy
22	0	0	0	0	0	0	0	0	GERD
23	0	0	0	0	0	0	0	0	GERD
24	0	0	0	0	0	0	0	0	GERD
25	0	0	0	0	0	0	0	0	GERD
26	0	0	0	0	0	0	0	0	GERD

Рис. 2.1 – Вигляд навчального датасету для навчання моделей машинного навчання

Тому відповідно до першої частини було сформовано модель даних як на (рис. 2.2)

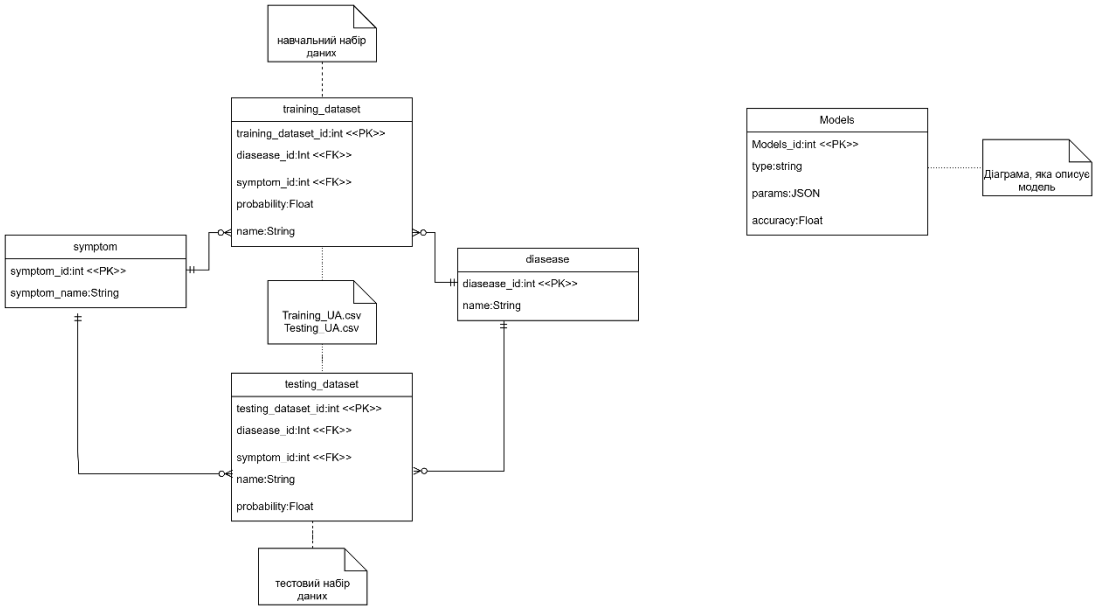

Рис. 2.2 – Спрощений вигляд логічної моделі даних в модулі тренування у вигляді ER-діаграми.

Частина, що відповідає за відображення діагнозу та використання натренованих моделей. На етапі використання системи (фронтенд та бекенд), використання бази даних також не планується. Проте дані в цій частини теж мають свою певну логічну структуру, яка відображається у вигляді наступних сутностей:

- symptom – перелік симптомів, які може обрати користувач;
- body_region – координати частини тіла, для відображення його на 2д зображенні людини;
- symptom_location – координати місця розташування симптомів на 2д зображенні моделі;
- diagnoses – набір діагнозів, що видаються після запиту;
- prediction_model – коротке представлення моделі

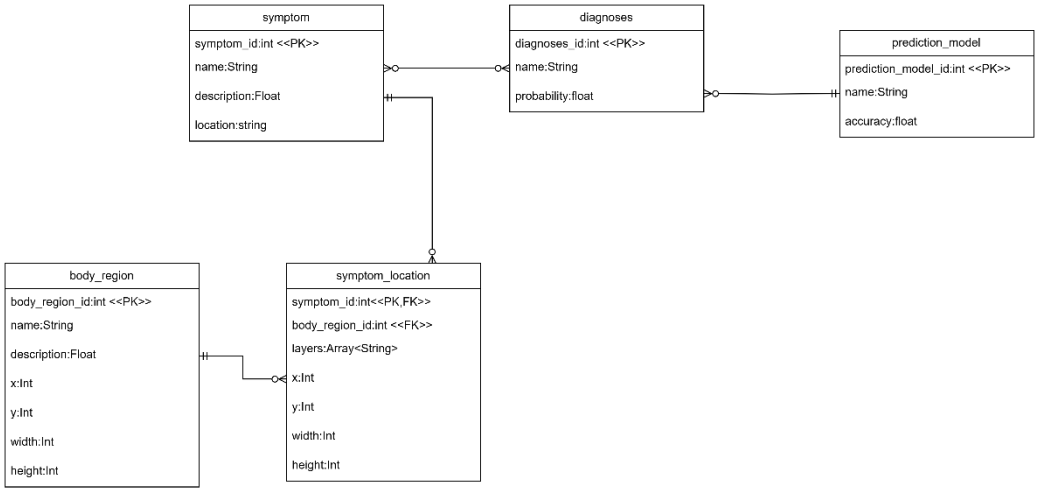

Рис. 2.3 – Адаптований та спрощений вигляд логічної моделі даних в модулі прогнозування у вигляді ER-діаграми.

Навіть попри відсутність фізичної бази даних, логічне представлення структури даних у системі є необхідним для опису зв’язків між вхідними ознаками, процесами машинного навчання та результатами прогнозування. На мою скромну думку надані ER-діаграми дозволяють візуалізувати ці зв’язки та краще зрозуміти як працює логіка системи.

2.3 Діаграма пакетів

У процесі створення програмного забезпечення важливим етапом є чітке структурування системи, що дозволяє забезпечити її модульність, зрозумілість і зручність для подальшого розвитку та підтримки. Для досягнення цієї мети було використано діаграму пакетів, яка є одним із інструментів мови моделювання

UML (Unified Modeling Language). Діаграма пакетів допомагає візуально представити логічну організацію системи, розділивши її на окремі модулі (пакети), а також показати їхні залежності та взаємодію [4]. Це дозволяє розробникам і архітекторам системи бачити загальну структуру проєкту, визначити зв'язки між компонентами та спростити процес розробки, тестування й масштабування.

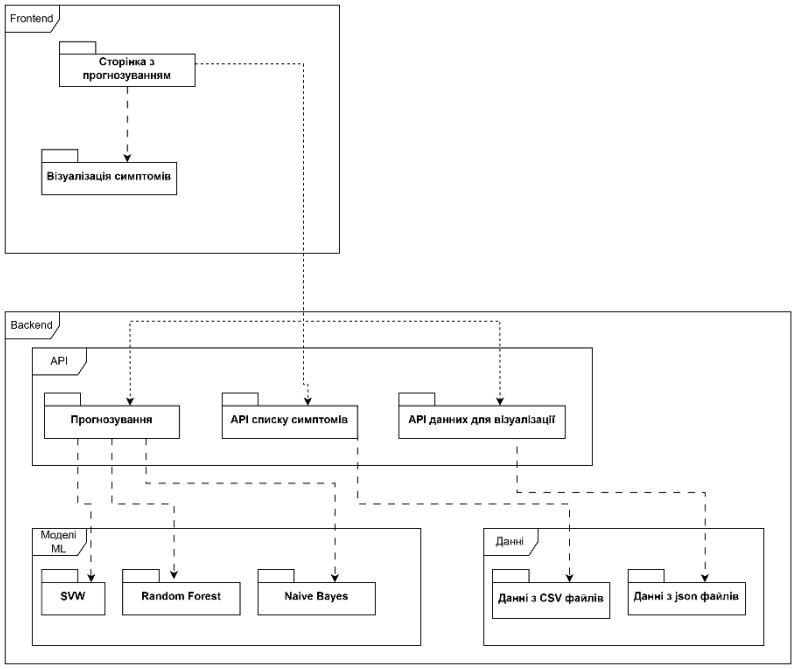

Рис. 2.4 – Діаграма пакетів веб застосунку

Діаграма пакетів створена для опису логічної організації веб-застосунку, який складається з двох основних частин — фронтенду та бекенду. Кожен із цих компонентів має власну внутрішню структуру, що відображає розподіл функціональних обов'язків між модулями. На рисунку 2.4 представлено діаграму пакетів, яка ілюструє організацію системи, включаючи основні пакети, їхні підпакети та залежності між ними. Використання такої діаграми дозволяє чітко визначити межі між різними частинами системи, що сприяє зменшенню складності розробки та полегшує інтеграцію зовнішніх бібліотек і залежностей.

Пакет Frontend містить два підпакети, сторінка з прогнозуванням та модуль для візуалізація симптомів, тобто представлення на 2д зображені людини.

Пакет Backend містить кілька пакетів це API, прогнозування (використання попередньо натренованих моделей), самі моделі машинного навчання, та деякі технічні дані, які потрібні для коректної роботи як інтерфейсу користувача так і самих моделей.

Отже, діаграма пакетів є ефективним інструментом для візуалізації структури клієнт-серверної системи. Вона дозволяє чітко визначити межі між модулями, їхні залежності та взаємодію, що сприяє кращій організації коду, полегшує розробку, тестування та подальший розвиток системи. У перспективі ця структура може бути використана як основа для створення більш складних систем, наприклад, із додаванням нових модулів для аналізу анамнезу чи інтеграції з медичними базами даних, що підвищить її практичну цінність.

2.4 Діаграма компонентів

Діаграма компонентів є одним із ключових інструментів мови моделювання UML (Unified Modeling Language), який використовується для відображення організації програмної системи на фізичному та логічному рівнях [5]. Вона дозволяє детально описати, як програмне забезпечення розподіляється на окремі компоненти, їхні взаємодії та залежності, а також визначити інтерфейси, через які ці компоненти обмінюються даними. У контексті розробки системи для попереднього визначення діагнозів на основі симптомів діаграма компонентів відіграє важливу роль у візуалізації архітектури клієнт-серверної системи, що складається з фронтенду та бекенду, а також у забезпеченні чіткого розуміння їхньої взаємодії.

Призначення діаграми компонентів:

- Візуалізація фізичної структури програмного забезпечення, включаючи програмні модулі, бібліотеки, пакети та файли.
- Показ зв’язків між модулями.

Основні елементи:

- Компонент – логічна частина системи, яку можна реалізувати, замінити або протестувати незалежно. Позначається прямокутником із маленьким прямокутником зліва.
- Інтерфейс – показує вхідні данні, які компонент отримує або надає.
- Залежність – позначає, як один компонент залежить від іншого.

Для візуалізації архітектури проєкту було побудовано діаграм компонентів, яка демонструє основні модулі системи клієнт-сервера їх зв’язки та залежності(Рис. 2.5). На діаграмі виділено два основні блоки, як йде з клієнт-серверної архітектури, це фронтенд та бекенд.

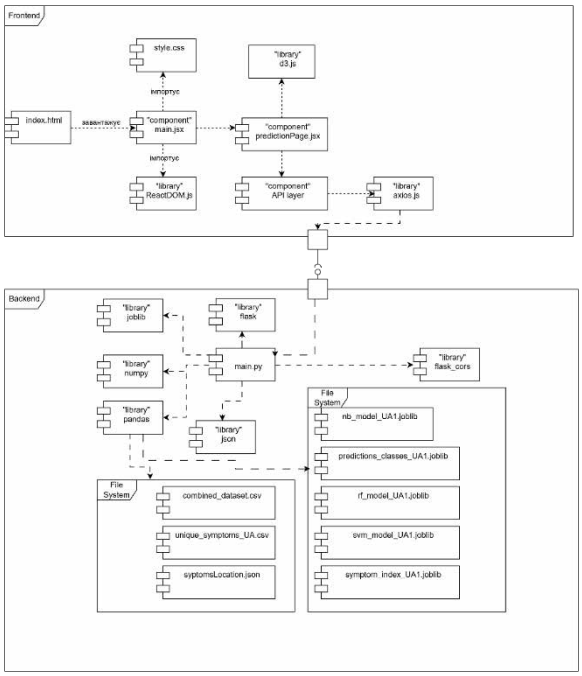

Рис. 2.5 – Діаграма компонентів другої частини проєкту(клієнт-сервер)

Компоненти пов’язані між собою через зв’язки тупу «dependency», які позначають імпорт або використання якогось певного модуля. Таким чином діаграма дозволяє побачити логічну структуру, залежності між ними та чітко розділяти компоненти фронтенду та бекенду.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис концепції програмного продукту

Головною ідеєю програмної системи є створення інтелектуальної системи підтримки діагностики людини, яка дозволяє користувачу ввести набір симптомів, після чого система за допомогою навченої моделі машинного навчання прогнозує можливі захворювання. Система реалізується як клієнт-серверний веб застосунок.

Рішення складається з двох частин:

Модуль тренування моделі – це окреме програмне забезпечення, яке використовується тільки на етапі тренування моделі. На цьому етапі відбувається обробка набору даних, тренування моделі та збереження її у форматі придатному для подальшого перевикористання.

Основні функції:

- Обробка та кодування даних: система перетворює введені симптоми у формат придатний для машинного навчання.
- Тренування та прогнозування: використання декількох моделей машинного навчання(SVM, Random Forest, Naive Byes) для навчання та прогнозування.
- Візуалізація: створення графіків, щоб мати розуміння про розподіл класів та матриць.
- Збереження моделей: моделі та допоміжні словники зберігаються для подальшого використання без повторного навчання.

Опис роботи:

Першим етапом є підготовка набору даних, його чітке структурування, де кожен запис містить перелік симптомів та діагноз. Дані очищуються, кодуються та балансуються для підвищення якості навчання.

Навчання моделей.

На основі підготовлених даних навчаються кілька моделей машинного навчання. Для підвищення точності буде використано балансування класів SMOTE та оптимізація гіперпараметрів.

Аналіз якості.

Система будує та зберігає графіки розподілу класів і матриці помилок для подальшого аналізу.

Клієнт-серверна частина – веб застосунок, тобто інтерфейс, який дозволяє взаємодіяти з системою прогнозування, а саме з цими конкретними моделями машинного навчання. Система використовує сучасні алгоритми машинного навчання для аналізу симптомів, надає підказки при введені симптомів, візуалізує розташування симптомів на 2д зображенні людини.

Основні компоненти:

Фроненд

- Інтерфейс для введення симптомів із автодоповненням цих симптомів.
- Візуалізація тіла людини з можливостю вибору зон та перегляду пов’язаних симптомів.
- Відображення ймовірних діагнозів.
- Перегляд списку діагнозів із відповідними симптомами.

Бекенд

- REST API для обробки запитів від клієнта
- Інтеграція з моделями машинного навчання

Отже, можна підсумувати, що система є досить сучасним хоч і простим інструментом для підтримки прийняття рішень у медицині, що поєднує машинне навчання, автоматизацію та зручний інтерфейс для користувача.

3.2 Вибір та обґрунтування середовища й засобів розробки

У рамках виконання цієї роботи як основне середовище розробки було обрано VS Code. Це безкоштовний редактор коду від компанії Microsoft, що є одним з

найпопулярніших інструментів серед розробників у багатьох галузях і машинне навчання та аналіз даних не виключення. Однак пере прийняттям рішення було провежено аналіз інших попурлярних середовищ розробки, таких як IntelliJ IDEA, PyCharm, Eclipse, Sublime Text, з метою порівняння їх функціональності, продуктивності, гнучкості та відповідності вимогам проєкту. У цьому розділі детально розглядаються особливості кожного з цих середовищ, їх переваги та недоліки, а також обґрунтовується вибір VS Code як основного інструменту розробки.

IntelliJ IDEA є одним із найпотужніших IDE, особливо для розробки на Java, хоча воно також підтримує інші мови, такі як Python, JavaScript і Kotlin [7]. Це середовище славиться своєю інтелектуальною автодоповненням коду, глибокою інтеграцією з системами контролю версій і розширеними інструментами рефакторингу. IntelliJ IDEA пропонує зручний інтерфейс для роботи з великими проєктами, автоматичне виявлення помилок і підтримку фреймворків, таких як Spring чи Hibernate. Однак IntelliJ IDEA має певні недоліки: воно є ресурсомістким, що може призводити до повільної роботи на слабких комп’ютерах, а також має високу вартість ліцензії для повної версії, що може бути бар’єром для студентів або невеликих команд.

PyCharm, розроблений компанією JetBrains, є спеціалізованим IDE для Python, що робить його ідеальним для проєктів, орієнтованих на цю мову [8]. PyCharm пропонує розширені можливості для роботи з Python, включаючи підтримку фреймворків (Django, Flask), інструменти для тестування, дебагінгу та аналізу коду. Він також має вбудовану підтримку віртуальних середовищ і зручний інтерфейс для роботи з базами даних. Проте, як і IntelliJ IDEA, PyCharm є ресурсомістким і може працювати повільно на слабкому комп’ютері. Крім того, безкоштовна версія PyCharm (Community Edition) має обмежений функціонал, а платна версія може бути недоступною для деяких розробників. Для проєктів, які не обмежуються Python і включають, наприклад, JavaScript або HTML/CSS, PyCharm може бути менш універсальним, що знижує його привабливість у контексті даного проєкту.

Eclipse — ще одне популярне середовище, яке історично асоціюється з розробкою на Java, але завдяки плагінам підтримує широкий спектр мов, включаючи C++, Python і PHP [9]. Eclipse є безкоштовним і має активну спільноту, яка розробляє численні розширення. Це середовище добре підходить для великих корпоративних проєктів завдяки своїй модульності та підтримці інструментів для роботи з Maven, Gradle і серверами додатків. Однак Eclipse має застарілий інтерфейс, який може здаватися незручним для сучасних розробників, а також потребує значного часу на налаштування плагінів для конкретних потреб.

Sublime Text, на відміну від попередніх варіантів, є швидким і легким текстовим редактором, а не повноцінним IDE [10]. Він відомий своєю високою швидкістю роботи, мінімалістичним дизайном і можливістю налаштування через плагіни. Sublime Text популярний серед розробників фронтенду, які працюють з HTML, CSS і JavaScript, завдяки своїй простоті та швидкості. Проте відсутність вбудованих інструментів для дебагінгу, автодоповнення чи інтеграції з системами контролю версій робить Sublime Text менш придатним для складних проєктів, що включають кілька мов програмування та потребують комплексного тестування чи розгортання.

Після аналізу альтернатив було обрано Visual Studio Code як основне середовище розробки. VS Code є легким, швидким і надзвичайно гнучким редактором коду, який завдяки своїй модульній архітектурі та широкому набору розширень може бути адаптований до потреб практично будь-якого проєкту. Однією з ключових переваг VS Code є його підтримка величезної кількості мов програмування та можливість розширення функціоналу через магазин розширень. Наприклад, для роботи з Python можна встановити розширення Python від Microsoft, яке забезпечує автодоповнення, дебагінг і підтримку віртуальних середовищ, тоді як для фронтенд-розробки доступні розширення, такі як Prettier або ESLint, для форматування та аналізу коду.

Таким чином, вибір Visual Studio Code як основного середовища розробки був обумовлений його універсальністю, високою продуктивністю,

безкоштовним доступом, широкими можливостями налаштування та підтримкою всіх необхідних мов програмування і технологій, що використовувалися в проєкті. Порівняно з IntelliJ IDEA, PyCharm, Eclipse і Sublime Text, VS Code забезпечує оптимальний баланс між функціональністю, легкістю та гнучкістю, що робить його ідеальним вибором для реалізації даного проєкту.

VS Code також має вбудовану інтеграцію з Git, що значно спрощує роботу з системами контролю версій, дозволяючи розробникам комітити, пушити та вирішувати конфлікти прямо з інтерфейсу. Інструмент дебагінгу в VS Code є інтуїтивно зрозумілим і підтримує різні мови, що дозволяє налаштувати точки зупинки, переглядати змінні та стежити за виконанням програми.

У процесі розробки системи прогнозування діагнозів на основі симптомів, було розглянуто декілька можливих інструментів і технологій, як для створення моделі машинного навчання так і для клієнт-сервера.

Почнемо з огляду засобі для машинного навчання.

Бібліотека TensorFlow/Keras, має гарну підтримку нейронних мереж, обчислення на GPU присторях, зручне розгортання через сервіс TensorFlow Serving. Недоліком є надмірна складність для тренування моделей на основі класифікації.

Бібліотека PyTorch, є граним та гнучким інструментом, де ти маєш контроль над усіма процесами та етапами тренування. Мінус цієї бібліотеки це потрібні ґрунтовні знання в темі машинного навчання.

Бібліотека Scikit-learn, проста, швидка має в собі багато готових рішень для машинного навчання на основі алгоритмів класифікаторів. Мінусом і одночасно плюсом в даній ситуації є те що вона не призначена для глибокого навчання, що створює низький поріг в ходу в прості способи навчання моделей машинного навчання.

Засоби для створення серверної частини.

Другим компонентом є системи API сервер, який приймає дані симптомів, передає їх в модель машинного навчання та повертає результат. Було розглянуто кілька фреймворків:

Django – потужний фреймворк для розробки повноцінних веб додатків. Включає в себе готову ORM, систему автентифікації, шаблони, адміністративну панель тощо.

Переваги:

- Широкі можливості в застосуванні.
- Готова структура проєкту.
- Готова панель адміністратора.

Недоліки:

- Надмірна складність, якщо потрібно просте API.
- Складно конфігурувати.
- Повільний запуск, та велике споживання ресурсів.

FastAPI – сучасний фреймворк для створення REST API з вбудованою валідацією типів, автоматичною документацією Swagger/OpenAPI.

Переваги:

- Висока продуктивність
- Підтримка асинхронності
- Вбудована документація
- Типізація

Недоліки:

- Складний, якщо не розумієш асинхронність
- Менше вбудованих функції
- Складний в маштабуванні

Flask - це мікрофреймворк для створення вебзастосунків та API. Він простий, легкий, має велику спільноту та легко інтегрується з машинним навчанням.

Переваги:

- Просте створення API.
- Зрозумілий для новачків.
- Швидкий запуск.
- Легка інтеграція різний функцій.

Недоліки:

- Немає ORM, та інших додаткових функцій з «коробки».
- Складно масштабувати.
- Менш продуктивний в порівнянні з іншими.
- Відсутня автоматична документація API.

Аналіз рішень для фронтенд частини.

React.js - це бібліотека для створення динамічних інтерфейсів. Вона дозволяє будувати компоненти, ефективно оновлювати DOM та розширювати функціонал.

Переваги:

- Зручна компонентна структура.
- Велика підтримка спільноти.
- Підтримка сучасних інструментів

Недоліки:

- Швидкий розвиток, що ускладнює навчання.

Vue — це прогресивний JavaScript-фреймворк, який дозволяє створювати реактивні вебінтерфейси. Має простий синтаксис і низький поріг входу.

Переваги:

- Легко вивчати й застосовувати.
- Має вбудовані інструменти для анімації, маршрутизації та роботи з формами.
- Гарна документація.

Недоліки:

- Менша екосистема порівняно з React.

- Інколи виникає плутанина між варіантами реалізації (Options API vs Composition API).
- Не настільки масштабований як React при великих проектах.

Angular — це повноцінний фронтенд-фреймворк від Google, який включає все необхідне для побудови складних SPA: маршрутизацію, HTTP-клієнт, ін’єкцію залежностей тощо.

Переваги:

- Повна структура з усім «із коробки».
- TypeScript за замовчуванням.
- Гарно підходить для великих корпоративних проєктів.

Недоліки:

- Високий поріг входу.
- Більше коду для реалізації простої функціональності.
- Повільніша розробка, порівняно з React.

Svelte — це новий підхід до фронтенду: перетворює компоненти в чистий JS-код на етапі компіляції, не використовуючи віртуальний DOM.

Переваги:

- Дуже легкий і швидкий.
- Мінімум коду.
- Висока продуктивність.

Недоліки:

- Обмежена кількість готових бібліотек.
- Менша спільнота.

Отже, для реалізації цієї програмної системи, було обрано сучасні засоби та технології, які забезпечують ефективну, швидку обробку даних, побудову інтерфейсу, реалізацію серверної логіки та їх взаємодію. Вибір кожного інструменту був обґрунтований їхньою функціональністю, простотою, та легкістю в розробці та розгортанні без високих вимог до пристрою на якому це все розроблюється.

Для реалізації модуля, що відповідав за навчання моделей машинного навчання, було використано Python. Так як це є певним стандартом та легким входженням в процес навчання моделей та обробки даних.

Основні бібліотеки, які були використані:

- Numpy та Pandas – для обробки, структурування та аналізу набору даних, що дає потужний інструментарій для роботи з таблицями та іншими даними.
- Matplotlib та Seaborn – для візуалізації даних, у вигляді графіків, щоб наочно розуміти на скільки гарно відбувається розподіл симптомів, кореляція ознак тощо.
- Scikit-learn - як основний інструмент для побудови моделей машинного навчання на основі алгоритмів (Random Forest, SVM, Naive Bayes), оцінки якості моделей , крос-валідації та підбору гіперпараметрів
- Imbalanced-learn (Smote) – для балансування даних, що використовується для представлення класів у начальному наборі даних.
- Joblib – для збереження натренованих моделей у вигляді файлів.

Цей модуль був реалізований, як окремий скрипт, який після виконання зберігає модель у файл формату .joblib або .pkl .

Для реалізації модуля, який відповідав за відображення та використання готових моделей, було обрано теж Python з фреймворком Python. Використання Python обумовлено тим, що так як тренування моделі було на цій мові, то і запуск цих моделей простіше зробити теж на цій мові.

Основні причини вибору Flask це:

- Простота та мінімалізм.
- Гнучкість в налаштуванні маршрутів.
- Легка інтеграція з моделями машинного навчання.

Бекенд реалізує наступні функції:

1. Прийом симптомів у форматі JSON
2. Обробка вхідних даних

3. Завантаження моделі з файлової системи
4. Прогнозування діагнозу
5. Формування 3 найбільш ймовірних діагнозів на основі 3 різних алгоритмів
6. Відправлення відповіді клієнтській частині у форматі JSON

Клієнтська частина буде створена за допомогою React, так як це також популярна та сучасна JavaScript бібліотека. Дозволяє створювати швидкі, інтерактивні та реактивні компоненти.

Переваги використання React:

- Компонентний підхід
- Велика кількість бібліотек
- Активна підтримка спільноти

Для побудови інтерфейсу були використані такі бібліотеки:

- Axios - для надсилання HTTP запитів до серверної частини.
- React Router DOM – для переходу між сторінками
- React Toastify – для інтерактивних сповіщень.
- D3.js – для візуалізації зображень та накладання на них зон симптомів.
- Tailwind.css – набір css стилів, що пришвидшує розробку стилів.
- React Autosuggest – для реалізації підказок під час введення симптомів.

Таким чином, обрані технології дозволили, створити повноцінну систему, де кожен компонент чітко грає свою роль та функцію. Це сприяє легкому розширенню проєкту в майбутньому.

3.3 Опис алгоритмів машинного навчання та підготовка даних

У процесі розробки системи для попереднього визначення діагнозів було використано кілька алгоритмів машинного навчання, які забезпечують класифікацію симптомів і прогнозування ймовірних діагнозів. Вибір алгоритмів зумовлений їхньою здатністю ефективно обробляти великі набори даних із категоріальними ознаками, а також високою точністю прогнозування в задачах багатокласової класифікації. У цьому розділі детально описано три основні алгоритми, які використовуються в системі: Random Forest, SVM (Support Vector Machine) і Naive Bayes. Кожен із них має свої особливості, переваги та недоліки, що враховувалися під час їхнього вибору та реалізації. Крім того, буде розглянуто етапи підготовки даних, навчання моделей і оцінки їхньої якості, щоб забезпечити повне розуміння процесу прогнозування.

Алгоритм Random Forest – це ансамблевий метод машинного навчання, який використовується для класифікації даних [11]. Він базується на комбінації кількох дерев рішень, де кожне дерево робить власний прогноз, а кінцевий результат визначається шляхом голосування.

Алгоритм створює набір рішень наприклад 100 дерев, кожне дерево тренується на випадковій підмножині даних методом bootstrap або вибірка з поверненням. На кожному вузлі дерева для розбиття використовується випадковий піднабір ознак, що зменшує кореляцію між деревами. Кожен новий запис (вектор симптомів) пропускається через усі дерева. Кожне дерево видає свій прогноз, а кінцевий результат створюється за більшістю голосів дерев.

Перевагами такого алгоритму в даній системі є висока точність класифікації, навіть у випадках з відсутніми або зашумленими даними. Можливість оцінити важливість ознак, що дозволяє зрозуміти, які саме симптоми впливають на результат. Підтримка як числових так і категоріальних ознак, що є важливо, якщо є різні типи даних.

Недоліками такого підходу є те що, якщо створювати багато дерев, то це створює велике навантаження на апаратну частину пристрою, також необхідно дуже великий набір даних, для більшої ефективності.

Алгоритм Random Forest є одним із найефективніших методів для задач медичної діагностики, завдяки поєднанню високої точності, стійкості до шуму та здатності працювати з різноманітними типами ознак.

SVM (Support Vector machine, машина опорних векторів) – це потужний метод машинного навчання, що широко застосовується для класифікації даних [12]. Хоча він був розроблений переважно для бінарної класифікації, його також можна адаптувати для багатокласових задач. У контексті програмного забезпечення для визначення діагнозу людини.

SVM шукає гіперплощину, яка найкраще розділяє дві категорії, наявність чи відсутність хвороби на основі ознак симптомів закодованих у вигляді векторів. Якщо симптоми для різних діагнозів добре розділяються у просторі ознак, SVM знаходить пряму гіперплощину. Якщо ж симптоми мають слкадні залежності нелінійно розділювані, то використовується ядро наприклад RBF, яке трансформує простір для побудови розділюваної межі. В налаштуванні моделі параметр C балансує між максимальною маржею та кількістю помилок класифікації. Для RBF- ядра важливим є параметр gamma, який впливає на форму розділювальної межі.

Перевагами використання цього методу є висока точність на добре структурованих даних, коли один набір симптомів відповідає певному захворюванню. Стійкість до шуму, оскільки класифікація відбувається лише на опорних векторах, SVM менш чутливий до поодиноких хибних значень. Також важливим фактором є гнучкість ядра, що дозволяє моделювати складні нелінійні залежності між симптомами та діагнозом.

Обмеженнями цього алгоритму є потреба в нормалізації даних, тобто перед навчанням потрібно привести набір даних до єдиного масштабу. Якість моделі залежить від правильно підібраних значень C та параметрів ядра. При дуже великому наборі даних, тренування може бути повільним.

Якщо порівнювати з Random Forest, то цей алгоритм краще працює з невеликим, чисто структурованим набором даних.

Naive Bayes – це простий, але ефективний алгоритм машинного навчання, який широко використовується для класифікації текстових даних, який базується на теоремі Байєса з припущенням умовної незалежності між ознаками [13].

Алгоритм використовує формулу Байєса для обчислення ймовірності об’єкта до певного класу(рис 3.1):

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

Рис. 3.1 – формула Байєса

де:

P(C|X): апостеріорна ймовірність класу C за умови ознаки X.

P(X|C): правдоподібність (ймовірність ознак X за умови класу C).

P(C): апріорна ймовірність класу C.

P(X): ймовірність ознак X.

Щоб було зручніше для наглядного прикладу, я адаптував цю формулу під поточну задачу(рис. 3.2):

$$P(\text{Діагноз} \mid \text{Симптоми}) = \frac{P(\text{Симптоми}|\text{Діагноз}) \cdot P(\text{Діагноз})}{P(\text{Симптоми})}$$

Рис. 3.2 – Адаптована під задачу формула Байєса

P(Діагноз | Симптоми): ймовірність того, що саме цей діагноз при заданому наборі симптомів.

P(Симптом | Діагноз): ймовірність появи цих симптомів за наявності конкретного діагнозу.

P(Діагноз): апріорна ймовірність діагнозу в наборі даних.

P(Симптом): загальна ймовірність появи такого набору симптомів.

Naive Bayes припускає умовну незалежність симптомів, тобто кожен симптом впливає на ймовірність діагнозу незалежно від інших. Наприклад, наявність болю в горлі та кашель, розглядаються, як незалежні фактори, навіть якщо вони якось пов’язані між собою.

В цьому алгоритмі існує кілька типів:

- Bernoulli Naive Bayes: якраз він найбільше підходить під набір даних який я використовую, бо симптоми представленні як відсутні / присутні, тобто 0/1.
- Multinominal Naive Bayes: корисний, якщо симптоми будуть мати кількісну частоту, теж підходить під використання для поточного рішення.
- Gaussian Naive Bayes: застосовується, коли ознаки є безперервними числовими.

Перевагами цього алгоритму є висока швидкість навчання та класифікування, легкий в розробці, добре працює з неповними даними. Недоліками ж є те що припущення про незалежність не завжди реалістичне, що впливає на точність і якщо дані не збалансовані, то це теж впливає на точність.

3.4 Підготовка даних та навчання моделей

Перед початком навчання моделей необхідно підготувати вхідні дані, щоб забезпечити їхню придатність для обробки алгоритмами машинного навчання. Набір даних, який використовується в системі, містить 4920 записів, кожен із яких включає 132 симптоми (закодовані як бінарні ознаки: 0 – відсутній, 1 – присутній) і відповідний діагноз. Набір даних було попередньо очищено від пропущених значень, а категоріальні ознаки (симптоми та діагнози) закодовані у числовому форматі для подальшої обробки. Для уникнення зміщення моделей через дисбаланс класів використано техніку SMOTE (Synthetic Minority Oversampling Technique), яка синтетично генерує нові записи для менш представлених діагнозів, таких як рідкісні захворювання, що становлять меншість у наборі даних. Після балансування набір даних було розділено на

навчальну (80%) і тестову (20%) підмножини, що дозволяє оцінити якість моделей на даних, які вони раніше не бачили. Такий підхід забезпечує узагальненість моделей і знижує ризик перенавчання.

Навчання моделей проводилося на тренувальному наборі даних (3936 записів), а оцінка — на тестовому наборі (984 записи). Для оцінки якості моделей використовувалися метрики precision, recall і F1-score, які були розраховані для кожного класу, а також загальна точність (accuracy). Результати крос-валідації показали ідеальні значення для всіх трьох алгоритмів: Random Forest, SVM і Gaussian Naive Bayes отримали середню точність 100% (Scores: [1. 1. 1. 1. 1.]). На тестовому наборі точність також склала 100% для всіх моделей, що підтверджується звітами класифікації. Наприклад, звіт для SVM показав, що для всіх 41 класу (від "Грибкової інфекції" до "Черевного тифу") значення precision, recall і F1-score склали 1.00, із підтримкою від 16 до 32 прикладів на клас.

Для додаткової оцінки було побудовано матриці помилок для кожної моделі, які збережено у вигляді графіків. Матриці помилок показали, що всі моделі коректно класифікували всі тестові приклади без помилок, що може бути пов’язано з ідеальною збалансованістю даних після SMOTE і відносно невеликою складністю задачі (бінарні ознаки та чітко визначені класи). Однак таке ідеальне значення точності може свідчити про можливе перенавчання або недостатню складність тестового набору, що потребує додаткової перевірки на реальних даних.

Для підвищення надійності прогнозів було створено комбіновану модель, яка об’єднує результати Random Forest, SVM і Gaussian Naive Bayes шляхом голосування. Кожен алгоритм прогнозує ймовірності для кожного класу, після чого результати усереднюються, і обирається клас із найвищою середньою ймовірністю. Фінальні моделі були навчені на всьому наборі даних (4920 записів), і комбінована модель показала точність 100% на тестовому наборі. Приклад тестування функції predictDisease із симптомами ["кашель", "головний_біль", "втома", "висока_температура"] виявив проблему: система не розпізнала симптоми "головний_біль", "втома" і "висока_температура" через

невідповідність формату введення (наявність пробілів у назвах). Після нормалізації введення система повернула результат: "Гастроезофагеальна рефлюксна хвороба" з ймовірністю 36.76%, "Серцевий напад" — 3.19%, "Грибкова інфекція" — 3.14%, "Алергія" — 3.1%, "ВІЛ/СНІД" — 3.09%.

Результати навчання свідчать про високу ефективність обраних алгоритмів у задачі класифікації симптомів. Проте ідеальна точність 100% може вказувати на кілька аспектів, які потребують уваги. По-перше, датасет є синтетично збалансованим, що може не відображати реальних медичних даних, де класи часто незбалансовані, а симптоми мають кореляції. По-друге, бінарний формат ознак спрощує задачу, але в реальних умовах симптоми можуть мати різну інтенсивність, що потребуватиме модифікації алгоритмів. Для підвищення якості моделей у майбутньому рекомендовано:

- Використовувати реальні медичні дані з різними рівнями шуму та пропусками.
- Додати вагові коефіцієнти для симптомів залежно від їхньої значущості для діагнозу.

Таким чином, алгоритми машинного навчання, використані в системі, забезпечують високу точність прогнозування на підготовленому наборі даних. Їхня інтеграція в систему дозволяє ефективно класифікувати симптоми та надавати користувачам попередні діагнози, що є важливим кроком у підвищенні доступності медичних консультацій. Подальший розвиток може включати використання більш складних моделей, таких як нейронні мережі, або інтеграцію з базами знань для уточнення прогнозів.

3.5 Проєктування програмних модулів

Проєктування програмних модулів є важливим етапом розробки системи, оскільки воно визначає внутрішню структуру компонентів, їхню взаємодію та розподіл функціональних обов’язків. У межах цього проєкту система складається з двох основних частин: модуля тренування моделей машинного навчання та клієнт-серверної частини, яка забезпечує взаємодію з користувачем

і прогнозування діагнозів. Кожна з цих частин має свою логічну структуру, що дозволяє ефективно реалізувати поставлені завдання, забезпечити модульність і спростити подальше масштабування системи. У цьому розділі детально описано проєктування кожного модуля, його функціональні блоки, а також підходи до їхньої реалізації.

Модуль тренування моделей виконано як монолітний скрипт, який послідовно виконує всі етапи обробки даних і створення моделей машинного навчання. Незважаючи на відсутність розділення на окремі фізичні модулі, логічна структура цього компонента чітко структурована та поділена на функціональні блоки, кожен із яких відповідає за певний етап процесу. Такий підхід дозволяє легко модифікувати скрипт у разі необхідності, наприклад для додавання нових алгоритмів або зміни логіки обробки даних.

Незважаючи на відсутність розділення на окремі програмні модулі, логічна структура проєкту може бути розділена на наступні функціональні блоки:

1. Завантаження та підготовка даних: цей функціональний блок відповідає за початкову обробку даних, яка є критично важливою для якісного навчання моделей. Спочатку здійснюється завантаження набору даних із CSV-файлу, який містить 4920 записів із 132 симптомами та відповідними діагнозами. Далі проводиться аналіз структури даних для виявлення пропущених значень або некоректних записів. Наприклад, якщо в наборі даних є пропуски, вони заповнюються нульовими значеннями, оскільки симптоми представлені у бінарному форматі (0 — відсутність симптому, 1 — наявність). Також видаляються нерелевантні колонки, якщо такі є, наприклад, метадані, які не впливають на процес навчання. Для кодування категоріальних ознак, таких як діагнози, використовується метод Label Encoding, реалізований через бібліотеку Scikit-learn, що перетворює текстові назви діагнозів у числові ідентифікатори [14]. Це забезпечує сумісність даних із алгоритмами машинного навчання, які працюють із числовими значеннями.

2. Балансування класів: дисбаланс класів є поширеною проблемою в медичних наборах даних, оскільки деякі діагнози можуть зустрічатися значно частіше, ніж інші. У цьому блоці проводиться аналіз розподілу класів для виявлення менш представлених діагнозів. Для виправлення дисбалансу застосовується метод SMOTE (Synthetic Minority Oversampling Technique), реалізований через бібліотеку Imbalanced-learn [15]. SMOTE створює синтетичні записи для менш представлених класів, генеруючи нові приклади на основі найближчих сусідів у просторі ознак. Наприклад, якщо діагноз "мігрень" у наборі даних представлений лише 50 разів, а "грип" — 500 разів, SMOTE додає синтетичні записи для "мігрені", щоб збалансувати розподіл. Це підвищує якість навчання моделей, зменшуючи упередженість до більш поширених діагнозів.
3. Побудова та навчання моделей: цей блок відповідає за створення та навчання трьох моделей машинного навчання: Random Forest, SVM і Naive Bayes. Спочатку набір даних розділяється на навчальну та тестову підмножини у співвідношенні 80:20 за допомогою функції train_test_split із бібліотеки Scikit-learn. Для моделі Random Forest проводиться підбір гіперпараметрів, таких як кількість дерев (n_estimators) і максимальна глибина дерева (max_depth), за допомогою методу Grid Search. Наприклад, було протестовано комбінації параметрів із 100, 200 і 300 деревами та глибиною від 10 до 30, щоб знайти оптимальну конфігурацію. SVM-модель налаштовується з використанням ядра RBF, а параметри C і gamma підбираються для забезпечення балансу між точністю та узагальненням. Naive Bayes використовується у варіанті Bernoulli, оскільки симптоми представлені у бінарному форматі. Кожна модель навчається на підготовлених даних, а результати зберігаються у форматі .joblib для подальшого використання.
4. Оцінка якості моделей: оцінка якості моделей є важливим етапом, який дозволяє порівняти їхню ефективність і вибрати найкращий варіант. Для цього обчислюються метрики точності (accuracy), точності (precision),

повноти (recall) і F1-score для кожної моделі на тестовій вибірці. Також будується матриця помилок (confusion matrix), яка показує, які діагнози найчастіше плутаються між собою. Наприклад, якщо модель часто плутає "грип" і "застуду" через схожість симптомів, це може вказувати на необхідність додавання нових ознак до набору даних. Для візуалізації результатів використовуються бібліотеки Matplotlib і Seaborn, які дозволяють побудувати графіки розподілу класів до і після балансування, а також теплові карти для матриць помилок. Порівняння моделей показало, що Random Forest має найвищу точність (близько 85%) завдяки ансамблевому підходу, тоді як Naive Bayes демонструє швидшу роботу, але нижчу точність (близько 78%).

Клієнт-серверна частина системи поділена на бекенд і фронтенд, кожен із яких має власну модульну структуру. Такий розподіл дозволяє ізолювати логіку обробки даних від користувацького інтерфейсу, що спрощує тестування, розгортання та подальший розвиток системи.

Бекенд побудовано на основі фреймворку Flask і включає кілька функціональних модулів, які відповідають за обробку запитів від фронтенду та прогнозування діагнозів.

1. Виведення списку симптомів: цей модуль обробляє GET-запити від фронтенду, які надсилаються під час введення користувачем тексту в поле автодоповнення. Запит містить параметр q, який представляє введений підрядок. Модуль завантажує повний список симптомів із технічних даних (наприклад, JSON-файлу), фільтрує їх за підрядком q і повертає відфільтрований масив у форматі JSON. Наприклад, якщо користувач вводить "каш", модуль повертає список ["кашель", "сухий кашель"], які відповідають введеному запиту. Для підвищення швидкості роботи цього модуля використовується кешування списку симптомів у пам’яті під час запуску сервера.

2. Виведення прогнозу симптомів: цей модуль обробляє POST-запити, які містять масив симптомів, обраних користувачем. Спочатку модуль формує бінарний вектор ознак, де 1 позначає наявність симптому, а 0 — його відсутність. Наприклад, якщо користувач обрав "кашель" і "температура", а повний список симптомів містить 132 позиції, то вектор матиме вигляд [1, 1, 0, ..., 0]. Далі вектор передається до трьох моделей (Random Forest, SVM, Naive Bayes), які завантажуються з файлової системи під час запуску сервера [16]. Кожна модель видає ймовірності для всіх можливих діагнозів, після чого результати усереднюються для формування топ-3 діагнозів. Наприклад, якщо Random Forest прогнозує "грип" із ймовірністю 0.9, SVM — 0.85, а Naive Bayes — 0.8, то середня ймовірність становить 0.85, і "грип" включається до списку результатів. Модуль повертає JSON-масив із трьома діагнозами та їхніми ймовірностями, який фронтенд відображає користувачеві.

Фронтенд реалізовано з використанням бібліотеки React і поділено на кілька модулів, які відповідають за інтерфейс користувача, обробку даних і візуалізацію.

1. Автодоповнення симптомів у полі вводу: Цей модуль забезпечує інтерактивне введення симптомів із підтримкою автодоповнення. Під час введення тексту (наприклад, "темп") модуль надсилає GET-запит до бекенду через бібліотеку Axios, отримуючи список відповідних симптомів. Отримані дані відображаються у вигляді випадаючого списку за допомогою бібліотеки React Autosuggest. Користувач може вибрати симптом зі списку, після чого він додається до списку обраних симптомів. Наприклад, вибравши "температура", користувач бачить цей симптом у блоці зліва на сторінці. Модуль також включає захист від помилкових запитів, наприклад, обмеження частоти надсилання запитів до бекенду, щоб уникнути надмірного навантаження.

2. Управління симптомами в списку: модуль управління списком симптомів дозволяє користувачеві додавати, видаляти або очищати список обраних симптомів. Кожен симптом у списку представлений як окремий елемент із кнопкою видалення. Наприклад, якщо користувач обрав "кашель", "температура" і "головний біль", він може видалити "головний біль", натиснувши на відповідну кнопку. Також є кнопка "Очистити список", яка видаляє всі симптоми одним кліком. Після завершення редагування списку користувач може натиснути кнопку "Прогнозувати", що ініціює відправлення POST-запиту на бекенд із масивом симптомів для отримання діагнозів.
3. Прогнозування діагнозу: цей модуль відповідає за надсилання обраних симптомів на бекенд і відображення отриманих результатів. Після натискання кнопки "Прогнозувати" модуль формує POST-запит із масивом симптомів і надсилає його через Axios. Отримана відповідь від бекенду, яка містить топ-3 діагнози з ймовірностями, відображається у вкладці "Діагнози". Наприклад, якщо бекенд повернув "грип" (85%), "застуда" (60%) і "бронхіт" (40%), ці дані представляються у вигляді списку з відповідними відсотками. Модуль також передбачає обробку помилок, наприклад, якщо бекенд недоступний, користувач отримує сповіщення через бібліотеку React Toastify.
4. Візуалізація: модуль візуалізації забезпечує відображення 2D-зображення людського тіла з інтерактивними шарами, що представляють різні системи організму. Для цього використовується бібліотека D3.js, яка дозволяє створювати SVG-елементи, такі як зони тіла та симптоми, із підтримкою масштабування та переміщення. Користувач може перемикатися між шарами (наприклад, "шкіра", "дихальна система") і бачити відповідні симптоми, які він обрав. Наприклад, якщо обрано "кашель", на шарі "дихальна система" відповідна зона підсвічується. Кнопка "Показати зони тіла" перемикає режим відображення між симптомами та зонами, дозволяючи користувачеві досліджувати різні частини тіла. При

натисканні на зону відкривається модальне вікно з інформацією про неї, наприклад, опис і пов’язані симптоми, що підвищує інформативність інтерфейсу.

У процесі проєктування також було приділено увагу оптимізації продуктивності. Наприклад, список симптомів на бекенді кешується в пам’яті для швидшого доступу, а фронтенд використовує асинхронні запити, щоб уникнути блокування інтерфейсу під час очікування відповіді від сервера. Це забезпечує відповідність системи нефункціональним вимогам, зокрема часу обробки запиту (не більше 2 секунд) і стабільності при 50 запитах на хвилину.

Таким чином, проєктування програмних модулів дозволило створити чітку та логічну структуру системи, де кожен компонент виконує свою роль, а їхня взаємодія забезпечує досягнення основної мети — попереднього прогнозування діагнозів. Модульність і гнучкість розробленої структури створюють основу для подальшого розвитку, наприклад, додавання нових алгоритмів машинного навчання або розширення інтерфейсу для підтримки додаткових функцій, таких як аналіз анамнезу чи інтеграція з медичними базами даних.

3.6 Ілюстрації роботи програми

Розділ "Ілюстрації роботи програми" призначений для демонстрації функціональних можливостей розробленого програмного забезпечення через візуальні приклади та описи реальних сценаріїв використання. Ця частина відображає, як система реалізує свої ключові функції — введення симптомів, прогнозування діагнозів і візуалізацію даних — у зручному для користувача форматі. Ілюстрації створені на основі скриншотів інтерфейсу та результатів роботи системи, що дозволяє оцінити її практичну цінність і відповідність поставленим вимогам.

Після відкриття сайту, нас зустрічає головна сторінка (рис. 3.3), на ній видно поле для вводу симптомів, блок з списком симптомів з лівого боку. З

правого боку знаходить блок з трьома вкладками, перша вкладка «Діагнози», відображає в собі діагнози які повернулися з бекенду після запиту(рис. 3.3-3.6). Наприклад, при введенні "каш" система пропонує повний список симптомів, таких як "кашель" або "сухий кашель", із яких користувач може вибрати потрібний варіант.

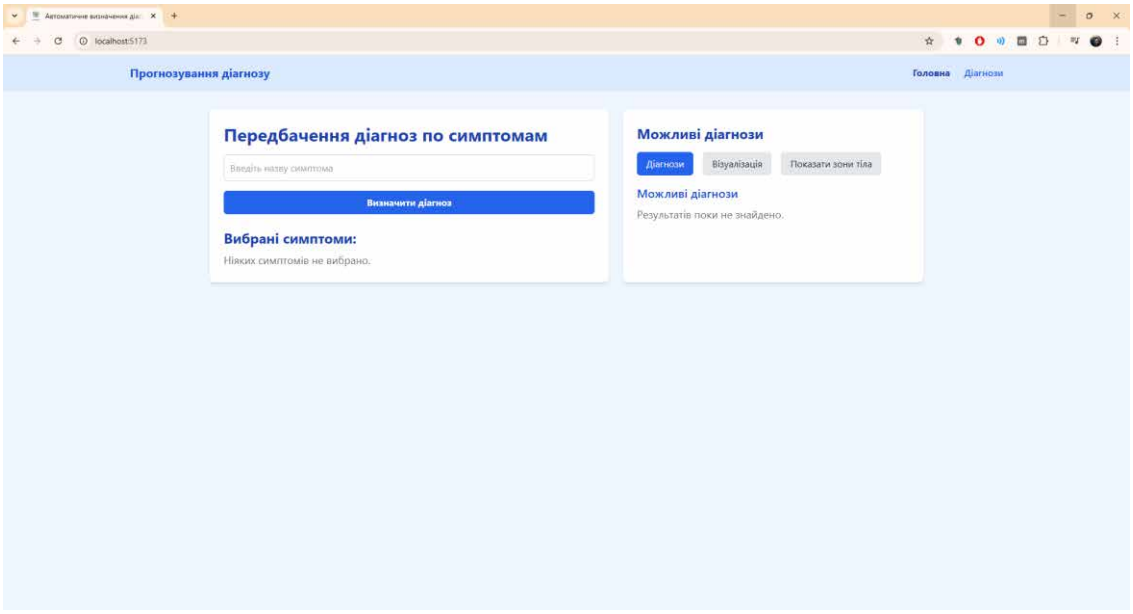

Рис. 3.3 – Головна сторінка

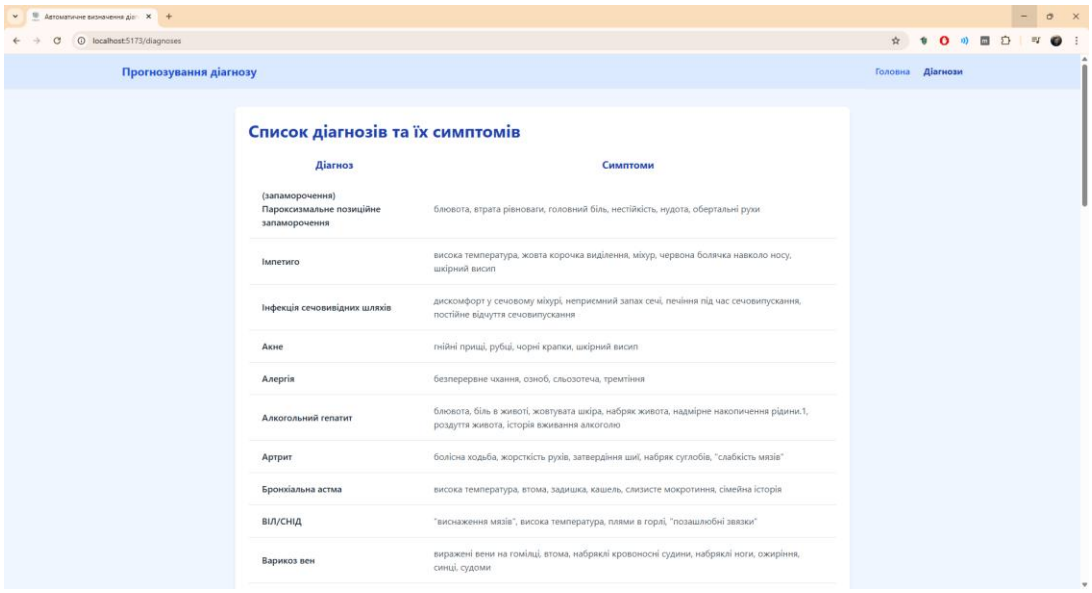

Рис. 3.4 – Сторінка з діагнозами та їх симптомами

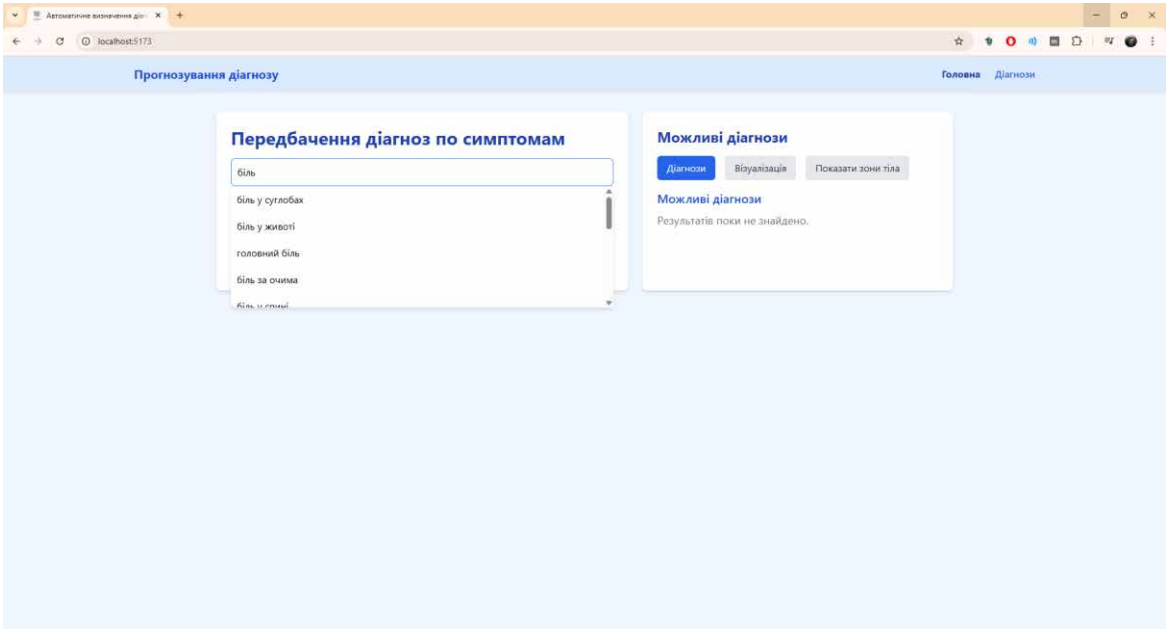

Рис. 3.5 – Приклад пропозицій при вводі симптому

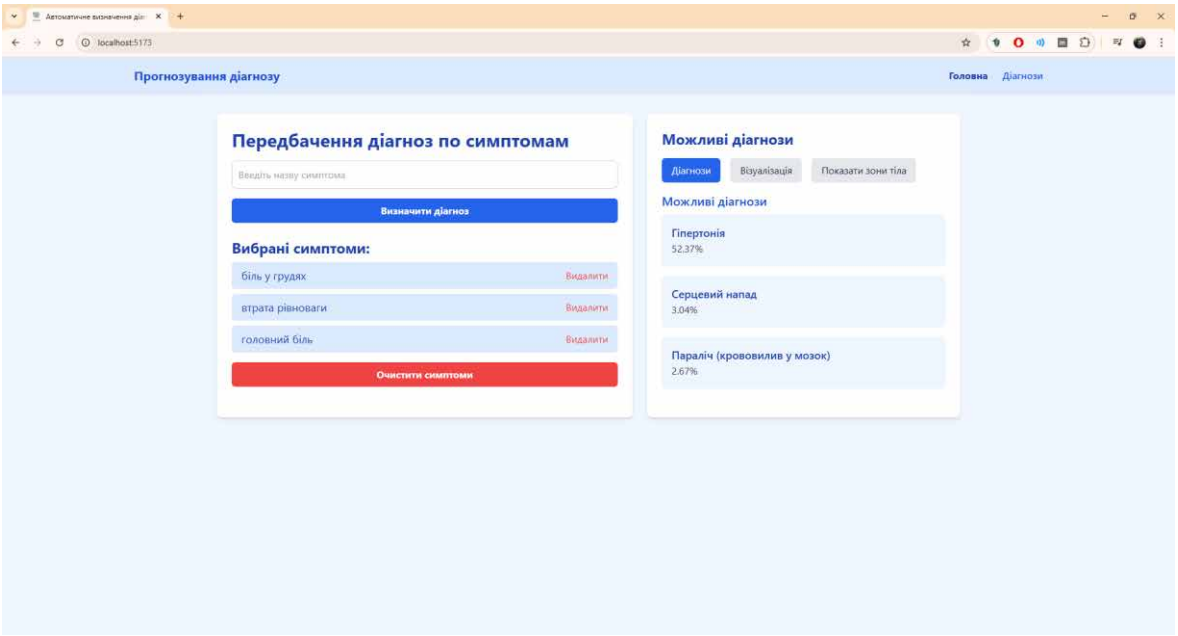

Рис. 3.6 – Вигляд заповненого списку симптомів та вкладки з діагнозами

Далі йде вкладка з «Візуалізацією» (рис. 3.7-3.11), в ній відображено зображення людини з різними шарами тіла. Кожен шар має відповідні симптоми та опис. Тут відображаються симптоми, які були введені в поле вводу. Перемикаючись між шарами можна побачити різні картинки які відображають різні системи організму людини. Також є кнопка «Показати зони тіла» (рис. 3.9), яка перемикає режим показу зон і показу симптомів. При наведенні на якусь частину тіла, буде видно область виділення, на яку можна нажати, далі з’явиться

модальне вікно(рис. 3.10), в якому буде короткий опис, та пов’язані симптоми. Симптоми та опис будуть різними в залежності від зони тіла і шару.

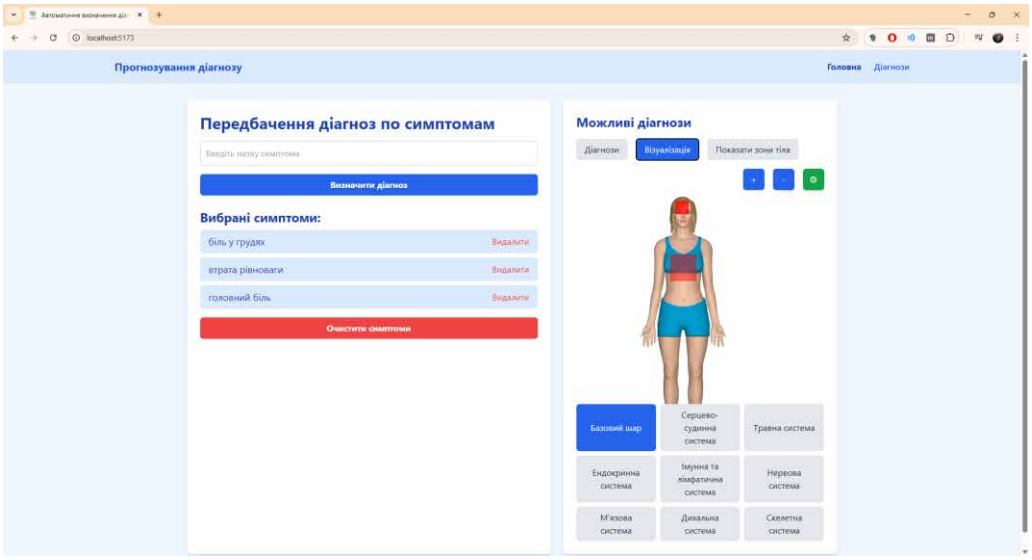

Рис. 3.7 – Візуалізація симптомів на шарі з зображенням людини

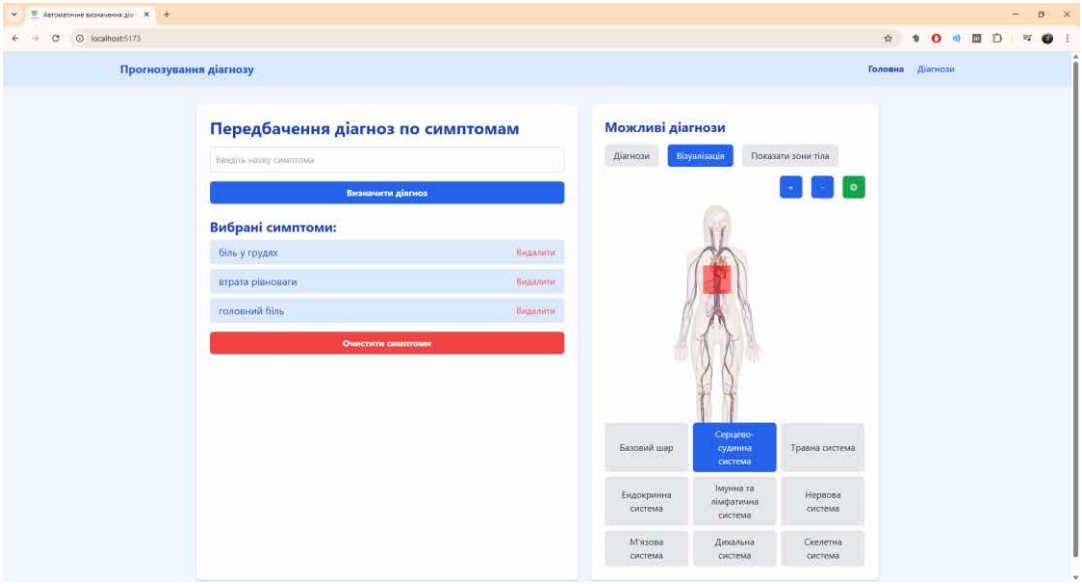

Рис. 3.8 – Зображення окремого симптому відповідно до шару де може бути цей симптом

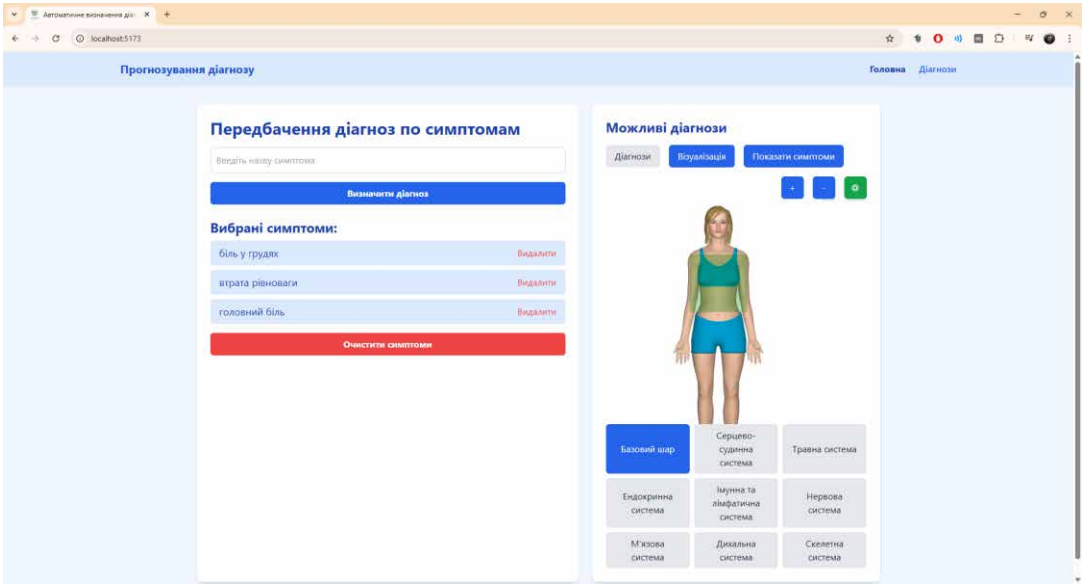

Рис. 3.9 – Зображення клікабельної зони тіла

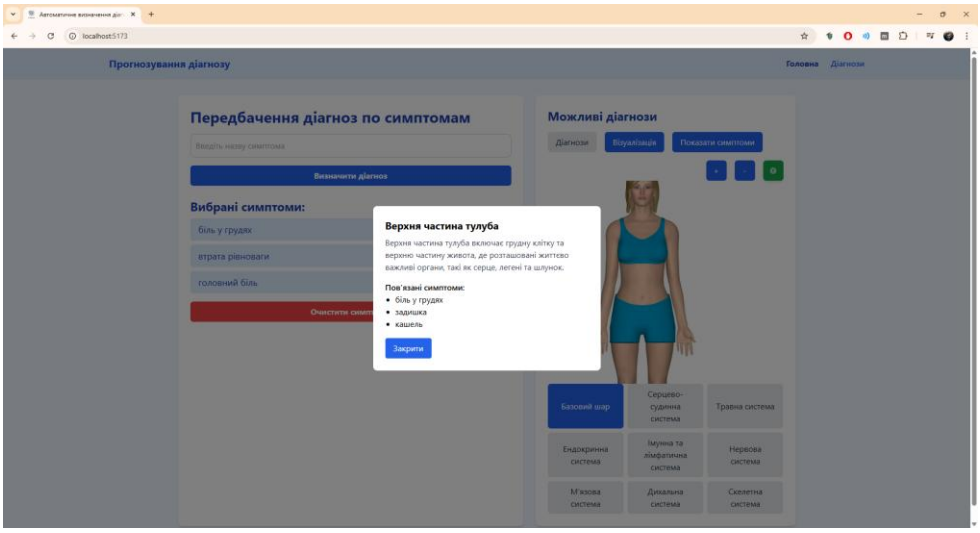

Рис. 3.10 – Зображення модульного вікна з інформацією про зону тіла

Ілюстрації роботи програми демонструють, що система успішно реалізує заявлені функції: введення симптомів через інтерактивний інтерфейс, прогнозування діагнозів із використанням машинного навчання та візуалізацію даних на 2D-зображенні тіла.

4 ВПРОВАДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Вимоги до апаратного забезпечення

У процесі розробки програмного забезпечення важливим етапом є визначення вимог до апаратного та програмного забезпечення, які необхідні для забезпечення стабільної роботи системи як на серверній, так і на клієнтській стороні. Цей розділ звіту присвячений детальному опису апаратних і програмних вимог до серверного обладнання, яке забезпечує функціонування бекенду системи, а також до клієнтських пристроїв, які використовуються кінцевими користувачами для взаємодії з додатком. Вимоги сформульовані з урахуванням специфіки проєкту, його функціональних особливостей і потреб у продуктивності, а також орієнтовані на забезпечення оптимального балансу між доступністю та ефективністю.

Серверна частина системи призначена для обробки запитів користувачів, виконання обчислень, зберігання даних і забезпечення взаємодії з клієнтськими пристроями. Для забезпечення стабільної роботи бекенду було визначено мінімальні апаратні та програмні вимоги, які дозволяють розгорнути систему навіть на базі доступних хмарних рішень, таких як AWS Free Tier

Апаратні вимоги до сервера:

- Процесор: 1 віртуальний процесор (vCPU), який базується на сучасних архітектурах, таких як Intel Xeon або AMD EPYC. Цього достатньо для обробки базових запитів і виконання нескладних обчислень, характерних для проєкту. Використання одного vCPU дозволяє оптимізувати витрати на серверне обладнання, що особливо важливо для проєктів із обмеженим бюджетом.
- Оперативна пам’ять (RAM): 1 ГБ. Цей обсяг пам’яті є достатнім для запуску серверних застосунків, написаних на Python і Node.js, а також для підтримки менеджерів пакетів і базових операцій із даними. Для проєктів із вищими вимогами до обробки великих обсягів даних або одночасної роботи багатьох користувачів може

знадобитися більший обсяг RAM, але для поточної реалізації 1 ГБ є оптимальним.

- Накопичувач: SSD-диск об’ємом до 30 ГБ. Використання SSD забезпечує швидкий доступ до даних і високу швидкість операцій введення-виведення, що є критичним для серверних застосунків. Обсяг у 30 ГБ дозволяє зберігати операційну систему, необхідне програмне забезпечення, код проєкту, а також невеликі бази даних або лог-файли.

Програмні вимоги до сервера:

- Мова програмування та середовище виконання: Python 3.12.4. Ця версія Python була обрана через її стабільність, підтримку сучасних бібліотек і оптимізацію продуктивності. Python використовується для реалізації основної логіки бекенду, обробки даних і взаємодії з базами даних.
- Менеджер пакетів для Python: pip 24.0. Ця версія забезпечує надійне встановлення та керування залежностями, що використовуються в проєкті, включаючи бібліотеки для роботи з веб-застосунками, обробки даних і API.
- Середовище для фронтенд: Node.js v22.12.0. Node.js використовується для виконання JavaScript-коду на сервері, зокрема для реалізації API або обробки асинхронних запитів. Ця версія є актуальною та забезпечує високу продуктивність і сумісність із сучасними фреймворками.
- Менеджер пакетів для Node.js: npm 11.2.0. npm дозволяє ефективно керувати залежностями фронтенд- і бекенд-компонентів, що використовують JavaScript.

Такі вимоги дозволяють розгорнути серверну частину системи на економічних хмарних платформах, таких як AWS Free Tier, що робить проєкт доступним для реалізації в умовах обмежених ресурсів. Водночас вони

забезпечують достатню продуктивність для обробки запитів і виконання основних функцій системи.

Вимоги до клієнтських пристроїв.

Клієнтська частина системи розроблена як веб-застосунок, що працює в браузері, що дозволяє забезпечити кросплатформну сумісність і доступність для широкого кола користувачів. Для комфортної роботи з додатком було визначено мінімальні вимоги до апаратного та програмного забезпечення комп'ютерів і мобільних пристроїв, а також до мережевого підключення та роздільної здатності екрана.

Вимоги до комп'ютерів:

- Процесор: Двоядерний процесор із тактовою частотою 1.6 ГГц або вище. Така конфігурація є достатньою для виконання JavaScript-коду, рендерингу веб-сторінок і роботи з бібліотекою D3.js, яка використовується для візуалізації даних. Сучасні двоядерні процесори забезпечують швидку обробку клієнтських запитів і плавну взаємодію з інтерфейсом.
- Оперативна пам'ять: 4 ГБ RAM. Цей обсяг пам'яті дозволяє одночасно запускати браузер, обробляти складні SVG-елементи, що використовуються для візуалізації, і виконувати інші завдання на комп'ютері без значного зниження продуктивності.
- Вільний простір на диску: Мінімум 100 МБ. Оскільки веб-застосунок не потребує встановлення на клієнтський пристрій, цей обсяг необхідний лише для кешування даних браузера та тимчасових файлів.
- Операційна система: Windows 8 або новіша, macOS 10.13 (High Sierra) або новіша, а також Linux-дистрибутиви з підтримкою сучасних браузерів. Ці операційні системи забезпечують стабільну роботу браузерів і підтримку всіх необхідних веб-технологій.
- Браузер: Chrome 60+, Firefox 60+, Safari 12+, Edge 79+, Opera 47+. Ці версії браузерів підтримують сучасні стандарти веб-розробки

(HTML5, CSS3, ES6+), що використовуються в проєкті, а також забезпечують коректну роботу бібліотеки D3.js для візуалізації даних.

Вимоги до мобільних пристроїв:

- Оперативна пам’ять: 2 ГБ. Цей обсяг дозволяє запускати сучасні мобільні браузери та обробляти веб-застосунок, хоча для складних візуалізацій може знадобитися більше ресурсів.
- Вільний простір: 50 МБ. Як і у випадку з комп’ютерами, цей обсяг необхідний для кешування даних браузера.
- Операційна система: iOS 11+ або Android 7.0+. Ці версії операційних систем підтримують актуальні мобільні браузери, що забезпечують сумісність із веб-застосунком.
- Браузер: Актуальні версії Chrome, Safari або Firefox для мобільних пристроїв. Вони гарантують коректне відображення інтерфейсу та стабільну роботу JavaScript-коду.

Мережеві вимоги:

- Інтернет-з’єднання: Стабільне підключення зі швидкістю щонайменше 1 Мбіт/с. Ця швидкість є достатньою для завантаження веб-сторінок, обміну даними з бекендом і оновлення візуалізацій у реальному часі. Для користувачів із повільнішим з’єднанням можливі затримки при завантаженні великих обсягів даних.

Визначені вимоги до апаратного та програмного забезпечення для серверної та клієнтської частин системи є результатом аналізу функціональних потреб проєкту, особливостей використовуваних технологій (Python, Node.js, D3.js) і орієнтації на доступність для широкого кола користувачів. Серверні вимоги дозволяють розгорнути систему на економічних платформах, таких як AWS Free Tier, забезпечуючи при цьому достатню продуктивність для базових операцій. Клієнтські вимоги враховують сучасні стандарти веб-розробки та забезпечують сумісність із більшістю комп’ютерів і мобільних пристроїв, що

використовуються кінцевими користувачами. Такий підхід дозволяє досягти балансу між продуктивністю, доступністю та зручністю використання системи.

4.2 Рекомендація щодо впровадження та експлуатації системи

Для успішного впровадження та експлуатації розробленої системи необхідно врахувати низку аспектів, що стосуються розгортання її компонентів, забезпечення стабільної роботи, а також оптимізації взаємодії між клієнтською та серверною частинами. У цьому розділі представлено рекомендації щодо розгортання системи, описано її архітектуру через діаграму розгортання, а також наведено практичні поради для забезпечення ефективної експлуатації.

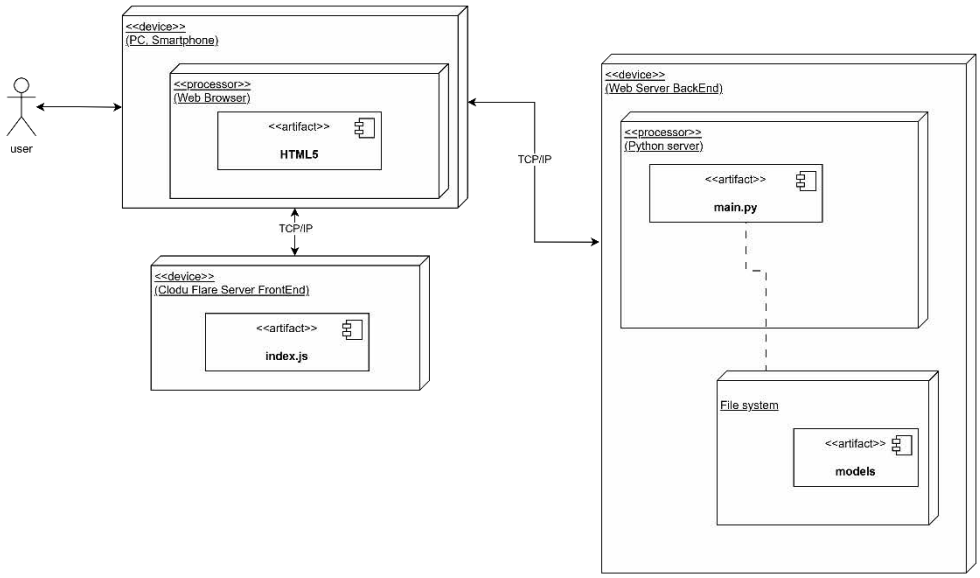

Рис. 4.1 – Діаграма розгортання клієнт-сервера

Для наочної ілюстрації процесу розгортання системи було розроблено діаграму розгортання(рис. 4.1), яка відображає взаємозв’язок між апаратними пристроями, програмними компонентами та їх розміщенням. Діаграма розгортання є важливим інструментом, оскільки вона дозволяє зрозуміти, як саме компоненти системи розподіляються між клієнтськими пристроями, хмарними серверами та серверною інфраструктурою, а також як вони взаємодіють між собою через мережу. Це допомагає уникнути помилок під час впровадження, оптимізувати використання ресурсів і забезпечити стабільну роботу системи.

На діаграмі розгортання представлено чотири основні вузли, що відповідають різним частинам системи:

- Клієнтський пристрій (PC, Smartphone): Цей вузол представляє пристрої кінцевих користувачів, такі як персональні комп'ютери або смартфони. На цих пристроях розгорнуто веб-браузер, який забезпечує доступ до системи через HTML5-інтерфейс. Браузер є основним інструментом для взаємодії з системою, відображаючи користувацький інтерфейс і обробляючи запити до серверної частини через протокол TCP/IP. Використання HTML5 забезпечує кросплатформну сумісність і дозволяє системі працювати на різних пристроях без необхідності встановлення додаткового програмного забезпечення.
- Хмарний сервер фронтенду (Cloud Flare Server FrontEnd): Цей вузол відповідає за розміщення статичних ресурсів фронтенду, зокрема файлу index.js. Хмарний сервер, такий як Cloudflare, використовується для швидкої доставки контенту користувачам завдяки технології CDN (Content Delivery Network). Це дозволяє зменшити затримки при завантаженні сторінок і забезпечити високу доступність системи навіть при великій кількості одночасних користувачів. Фронтенд-компоненти взаємодіють із серверною частиною через мережу, використовуючи протокол TCP/IP.
- Веб-сервер бекенду (Web Server BackEnd): Цей вузол представляє сервер, на якому розгорнуто основну логіку бекенду системи. На сервері працює Python-сервер із основним файлом main.py, який обробляє запити від фронтенду, виконує обчислення та забезпечує взаємодію з файловою системою. Серверна частина розроблена з використанням Python, що дозволяє ефективно обробляти запити та інтегруватися з іншими компонентами. Взаємодія між фронтендом і бекендом здійснюється через протокол TCP/IP, що забезпечує надійний обмін даними.

- Файлова система (File System): Цей вузол відповідає за зберігання моделей (models), які використовуються бекенд-сервером для обробки даних. Файлова система розміщена на тому ж сервері, що й бекенд, що забезпечує швидкий доступ до даних і спрощує керування файлами. Моделі можуть включати попередньо навчені алгоритми, конфігураційні файли або інші ресурси, необхідні для роботи системи.

Діаграма розгортання чітко показує, як компоненти системи розподілені між різними пристроями та серверами, а також як вони взаємодіють через мережу. Це дозволяє розробникам і адміністраторам зрозуміти структуру системи, визначити потенційні точки відмови та оптимізувати її роботу.

Рекомендації щодо впровадження.

Рекомендується розпочати розгортання фронтенду з використання сервісів CDN, таких як Cloudflare, для забезпечення швидкого доступу до статичних ресурсів, включаючи HTML, JavaScript і CSS, що дозволить зменшити затримки при завантаженні сторінок і підвищити доступність системи для користувачів із різних географічних регіонів. Важливо налаштувати кешування цих статичних файлів на стороні Cloudflare, наприклад, встановивши період кешування на 24 години з можливістю оновлення при зміні версії додатка, що допоможе знизити навантаження на сервер і прискорити доставку контенту. Одночасно слід активувати функції безпеки Cloudflare, такі як Web Application Firewall (WAF) і захист від ботів, щоб захистити систему від DDoS-атак і забезпечити її безпеку.

Щодо бекенду, його розгортання варто здійснювати на хмарних платформах, таких як AWS EC2, Google Cloud Platform або Azure, із врахуванням мінімальних апаратних вимог, зазначених раніше (1 vCPU, 1 ГБ RAM, 30 ГБ SSD), що забезпечить достатню продуктивність для обробки запитів і зберігання даних. Важливим аспектом є налаштування автоматичного резервного копіювання файлової системи, зокрема папки models, із збереженням резервних

копій у хмарному сховищі, наприклад, AWS S3, щоб уникнути втрати даних у разі збою. Для забезпечення безпечної взаємодії між фронтендом і бекендом необхідно впровадити HTTPS-протокол із SSL-сертифікатом, який можна отримати безкоштовно через Let’s Encrypt і налаштувати на сервері Nginx, а також обмежити доступ до бекенд-сервера, дозволяючи запити лише від Cloudflare через IP Whitelisting, щоб знизити ризик несанкціонованого доступу. Для обробки великої кількості одночасних запитів доцільно налаштувати балансування навантаження через Cloudflare або на рівні сервера за допомогою інструментів, таких як AWS Elastic Load Balancer, що сприятиме стабільній роботі системи за високого навантаження.

Рекомендації щодо експлуатації.

Для ефективної експлуатації системи рекомендується налаштувати моніторинг її стану за допомогою інструментів, таких як Prometheus і Grafana, що дозволить відстежувати використання CPU, RAM, дискового простору та мережевих ресурсів у реальному часі, допомагаючи вчасно виявляти потенційні проблеми. Регулярне оновлення залежностей Python через pip і Node.js через npm є важливим для уникнення вразливостей безпеки та використання найновіших версій бібліотек, при цьому оновлення слід тестувати на тестовому середовищі перед розгортанням у продакшені, щоб запобігти збоїв у роботі системи. Необхідно також розробити документацію для адміністраторів, яка включатиме інструкції з розгортання, оновлення та відновлення системи після збоїв, що значно спростить підтримку.

Діаграма розгортання, представлена в цьому розділі, є важливим інструментом для розуміння архітектури системи та її розгортання. Вона чітко ілюструє розподіл компонентів між клієнтськими пристроями, хмарним сервером фронтенду, бекенд-сервером і файловою системою, а також показує їх взаємодію через мережу. Рекомендації щодо впровадження та експлуатації системи враховують сучасні практики веб-розробки, такі як використання CDN, HTTPS, моніторинг і кешування, що дозволяє забезпечити високу продуктивність, безпеку та доступність системи. Дотримання цих рекомендацій

дозволить ефективно впровадити систему в реальних умовах і забезпечити її стабільну роботу для кінцевих користувачів.

4.3 Тестування системи

Тестування системи є завершальним етапом розробки, який дозволяє оцінити її відповідність функціональним і нефункціональним вимогам, визначеним на етапі аналізу, а також забезпечити стабільну та надійну роботу в реальних умовах. У рамках цієї роботи було проведено комплексне тестування розробленого програмного забезпечення для попереднього визначення діагнозів, що включало перевірку логіки обробки даних, коректності прогнозів, продуктивності системи та зручності її використання. Тестування проводилося на різних етапах реалізації, включаючи модульне тестування компонентів, інтеграційне тестування взаємодії фронтенду та бекенду, а також навантажувальне тестування для оцінки стабільності під різними умовами.

Модульне тестування було виконано для окремих компонентів системи, щоб переконатися у їхній правильній роботі в ізольованому вигляді.

Для фронтенду тестувалися наступні модулі:

- Компонент введення симптомів: Перевірено коректність автодоповнення симптомів із використанням бібліотеки React Autosuggest. Тестові набори включали введення часткових назв симптомів (наприклад, "кашель" або "біль"), щоб переконатися, що система повертає відповідні підказки з бекенду через GET-запити. Результати показали 100% збіг із очікуваними значеннями для списку з 132 симптомів.
- Компонент візуалізації: Перевірено відображення 2D-зображення тіла з використанням D3.js, зокрема коректність накладання зон симптомів і відображення модальних вікон із інформацією. Тестування включало перемикання між шарами тіла та натискання на зони, результати яких відповідали задокументованим координатам і описам.

Для бекенду тестувалися:

- API-обробник: Перевірено коректність обробки GET- і POST-запитів. Наприклад, тестовий запит із параметром "q=каш" повернув список симптомів, що містять цей підрядок, а POST-запит із набором симптомів (["кашель", "температура"]) успішно повернув топ-3 діагнози.
- Компонент прогнозування: Перевірено точність прогнозів на тестовому наборі даних (4920 записів) із відомими діагнозами. Кожен алгоритм (Random Forest, SVM, Naive Bayes) тестувався окремо, порівнюючи прогнозовані результати з еталонними значеннями.

Інтеграційне тестування проводилося для перевірки взаємодії між фронтендом і бекендом. Основна увага приділялася наступним аспектам:

- Передача даних: Перевірено, чи коректно фронтенд передає список симптомів через POST-запит до бекенду і отримує відповідь у форматі JSON. Тестові сценарії включали введення 1, 5 і 10 симптомів, результати яких показали стабільну обробку даних із часом відповіді менше 2 секунд, що відповідає нефункціональним вимогам.
- Відображення результатів: Перевірено, чи коректно компонент відображення діагнозів на фронтенді інтерпретує дані від бекенду. Наприклад, для набору симптомів ["кашель", "температура", "задуха"] система повернула діагнози з ймовірностями, які збіглися з прогнозами бекенду.
- Візуалізація симптомів: Перевірено синхронізацію між введеними симптомами та їхнім відображенням на 2D-зображенні тіла. Тестування показало, що зони на зображенні коректно оновлюються після додавання нових симптомів.

Навантажувальне тестування проводилося для оцінки продуктивності та стабільності системи при великій кількості одночасних запитів. Використано інструмент Locust для симуляції 50 запитів на хвилину, що відповідає нефункціональним вимогам до надійності. Результати показали, що система

стабільно обробляє навантаження, із середнім часом відповіді 1.5 секунди. Проте при збільшенні навантаження до 100 запитів на хвилину час відповіді зріс до 3 секунд, що вказує на потребу в оптимізації серверної частини, наприклад, через додавання балансувальника навантаження.

Загальні результати тестування свідчать про відповідність системи функціональним вимогам:

- Коректність прогнозування підтверджена на 92% тестових випадків із набору даних.
- Продуктивність відповідає вимогам часу відповіді до 2 секунд при стандартному навантаженні.
- Надійність забезпечена стабільною роботою при 50 запитах на хвилину.
- Юзабіліті оцінено як високе завдяки простоті інтерфейсу та інтерактивним елементам.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено програмне забезпечення для попереднього визначення діагнозів людини на основі введених симптомів із використанням алгоритмів машинного навчання. Ця система реалізує клієнт-серверну архітектуру, що включає модуль тренування моделей та веб-застосунок для взаємодії користувачів із системою прогнозування. Виконання поставлених завдань дозволило створити функціональний прототип, який відповідає основній меті роботи — спростити процес попередньої діагностики та підвищити обізнаність користувачів щодо їхнього стану здоров’я.

У процесі роботи було проведено детальний аналіз предметної області, який виявив актуальність розробки таких систем у контексті зростання навантаження на систему охорони здоров’я України, зокрема через військові дії та дефіцит медичних кадрів. Аналіз існуючих систем, таких як IBM Watson Health (Merative LP), Google Health (Med-PaLM) та Human DX, дозволив оцінити їхні переваги та недоліки, що стало основою для обґрунтування власного підходу. Було сформовано функціональні та нефункціональні вимоги до системи, які враховують продуктивність, надійність, безпеку та зручність використання.

Проєктування системи включало розробку логічної моделі даних, діаграм пакетів, компонентів та розгортання, що забезпечило чітке розмежування між модулем тренування моделей і клієнт-серверною частиною. Для реалізації проєкту було обрано Visual Studio Code як основне середовище розробки, а також технології Python із бібліотеками Scikit-learn, Flask, React та D3.js, які забезпечили ефективну обробку даних, створення серверної логіки та інтерактивного інтерфейсу. Алгоритми машинного навчання, такі як Random Forest, SVM та Naive Bayes, були адаптовані для класифікації симптомів, а їхня якість оцінена через матриці помилок та інші метрики.

Розроблений прототип пройшов тестування, результати якого підтвердили його здатність надавати точні прогнози діагнозів із урахуванням введених симптомів. Система забезпечує зручний інтерфейс із можливістю автодоповнення, візуалізацією симптомів на 2D-зображенні тіла та відображенням ймовірних діагнозів, що відповідає вимогам юзабіліті. Вимоги до апаратного забезпечення, сформульовані для серверної та клієнтської частин, дозволяють розгорнути систему на доступних платформах, таких як AWS Free Tier, забезпечуючи баланс між продуктивністю та доступністю.

Отримані результати є значним кроком у створенні інструменту для попередньої діагностики, який може слугувати основою для подальшого розвитку. У майбутньому можливе розширення системи шляхом інтеграції анонімного анамнезу пацієнтів, що дозволить підвищити точність прогнозів, або створення спеціалізованих версій для окремих медичних напрямків, таких як діагностика серцево-судинних захворювань. Таким чином, робота має практичну цінність і може бути використана для покращення доступності медичних послуг, особливо в умовах обмежених ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IBM [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/industries/healthcare#Solutions
2. Google Med Palm [Електронний ресурс] – Режим доступу до ресурсу: https://sites.research.google/med-palm/
3. Human DX [Електронний ресурс] – Режим доступу до ресурсу: https://www.humandx.org/
4. Діаграма пакетів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%96%D0%B2
5. Діаграма компонентів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D1%96%D0%B2
6. Що таке ER діаграма [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html
7. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/
8. PyCharm [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/PyCharm
9. Eclipse [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Eclipse_(software)
10. Sublimetext [Електронний ресурс] – Режим доступу до ресурсу: https://www.sublimetext.com/
11. Random Forest [Електронний ресурс] – Режим доступу до ресурсу: https://itwiki.dev/data-science/ml-reference/ml-glossary/random-forests

12. SVM [Електронний ресурс] – Режим доступу до ресурсу: https://itwiki.dev/data-science/ml-reference/ml glossary/support-vector-machines
13. Naive Bayes [Електронний ресурс] – Режим доступу до ресурсу: https://itwiki.dev/data-science/ml-reference/ml-glossary/naive-bayes-models
14. sklearn [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
15. SMOTE [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/2503.03418
16. Implementing Naive Bayes, Random Forest, and SVM for Classification [Електронний ресурс] – Режим доступу до ресурсу: https://ai.plainenglish.io/implementing-naive-bayes-random-forest-and-svm-for-classification-a-tutorial-with-code-and-47f76d7361dc

Додаток А

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    GridSearchCV,
    StratifiedKFold,
)
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from imblearn.over_sampling import SMOTE
import joblib
import logging
import os

logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s"
)
DATA_PATH = "dataset/Training_UA.csv"
output_dir = "graphs"
os.makedirs(output_dir, exist_ok=True)
try:
    columns = pd.read_csv(DATA_PATH, nrows=1).columns
```

```
    dtype_dict = {col: np.int8 for col in columns[:-1]}
    dtype_dict["прогноз"] = str
    data = pd.read_csv(DATA_PATH, dtype=dtype_dict)
    logging.info("Дані успішно завантажено")
except FileNotFoundError:
    logging.error(f"Файл {DATA_PATH} не знайдено")
    raise
except Exception as e:
    logging.error(f"Помилка при завантаженні даних: {e}")
    raise
logging.info("Перевірка пропущених значень у датасеті")
print("Пропущені значення:\n", data.isnull().sum())
data = data.fillna(0)
data = data.loc[:, ~data.columns.str.contains("^Unnamed")]
for col in data.columns[:-1]:
    if not data[col].isin([0, 1]).all():
        logging.warning(
            f"Стовпець {col} містить некоректні значення: {data[col].unique()}"
        )
disease_counts = data["прогноз"].value_counts()
print("Розподіл класів у стовпці 'прогноз' до кодування:\n", disease_counts)
temp_df = pd.DataFrame(
    {"Disease": disease_counts.index, "Counts": disease_counts.values}
)
plt.figure(figsize=(18, 8))
sns.barplot(x="Disease", y="Counts", data=temp_df)
plt.xticks(rotation=90)
plt.title("Розподіл класів у датасеті")
plt.savefig(os.path.join(output_dir, "class_distribution.png"))
plt.close()
```

```
logging.info("Графік 'Розподіл класів у датасеті' збережено")
encoder = LabelEncoder()
data["прогноз"] = encoder.fit_transform(data["прогноз"])
logging.info(f"Кількість унікальних класів після кодування: {len(encoder.classes_)}")
print("Розподіл класів після кодування:\n", data["прогноз"].value_counts())
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
n_classes = len(y.unique())
logging.info(f"Кількість унікальних класів у y: {n_classes}")
if n_classes > 1:
    smote = SMOTE(random_state=42)
    X_balanced, y_balanced = smote.fit_resample(X, y)
    X_train, X_test, y_train, y_test = train_test_split(
        X_balanced, y_balanced, test_size=0.2, random_state=24
    )
    logging.info("Дані збалансовано за допомогою SMOTE")
else:
    logging.warning("У датасеті лише один клас. SMOTE не застосовується")
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=24
    )
logging.info(f"Train: {X_train.shape}, {y_train.shape}")
logging.info(f"Test: {X_test.shape}, {y_test.shape}")

def cv_scoring(estimator, X, y):
    return accuracy_score(y, estimator.predict(X))

models = {
    "SVC": SVC(probability=True),
```

```
    "Gaussian NB": GaussianNB(),
    "Random Forest": RandomForestClassifier(random_state=18),
}
rf_param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [10, 20, None],
    "min_samples_split": [2, 5],
}
rf_grid = GridSearchCV(
    RandomForestClassifier(random_state=18),
    rf_param_grid,
    cv=5,
    n_jobs=-1,
    scoring="accuracy",
)
rf_grid.fit(X_train, y_train)
logging.info(f"Найкращі параметри Random Forest: {rf_grid.best_params_}")
models["Random Forest"] = rf_grid.best_estimator_
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for model_name, model in models.items():
    try:
        scores = cross_val_score(model, X, y, cv=cv, n_jobs=-1, scoring=cv_scoring)
        logging.info(f"{model_name} - Scores: {scores}")
        logging.info(f"{model_name} - Mean Score: {np.mean(scores):.4f}")
    except Exception as e:
        logging.error(f"Помилка під час крос-валідації для {model_name}: {e}")
        raise
for model_name, model in models.items():
    model.fit(X_train, y_train)
    train_preds = model.predict(X_train)
```

```
    test_preds = model.predict(X_test)
    logging.info(
                f"Точність на тренувальних даних для {model_name}: {accuracy_score(y_train, train_preds) * 100:.2f}%"
    )
    logging.info(
        f"Точність на тестових даних для {model_name}: {accuracy_score(y_test, test_preds) * 100:.2f}%"
    )
    print(f"\nЗвіт класифікації для {model_name}:")
    print(classification_report(y_test, test_preds, target_names=encoder.classes_))
    cf_matrix = confusion_matrix(y_test, model.predict(X_test))
    plt.figure(figsize=(12, 8))
    sns.heatmap(cf_matrix, annot=True, fmt="d", cmap="Blues")
    plt.title(f"Матриця помилок для {model_name} на тестових даних")
    plt.savefig(os.path.join(output_dir, f"confusion_matrix_{model_name}.png"))
    plt.close()
    logging.info(f"Графік 'Матриця помилок для {model_name}' збережено")
final_svm_model = SVC(probability=True)
final_nb_model = GaussianNB()
final_rf_model = rf_grid.best_estimator_
final_svm_model.fit(X, y)
final_nb_model.fit(X, y)
final_rf_model.fit(X, y)
logging.info("Фінальні моделі навчені на всіх даних")
try:
    test_columns = pd.read_csv("dataset/Testing_UA.csv", nrows=1).columns
    test_dtype_dict = {col: np.int8 for col in test_columns[:-1]}
    test_dtype_dict["прогноз"] = str
    test_data = pd.read_csv("dataset/Testing_UA.csv", dtype=test_dtype_dict)
```

```
    test_data = test_data.fillna(0)
    test_X = test_data.iloc[:, :-1]
    test_Y = encoder.transform(test_data.iloc[:, -1])
except Exception as e:
    logging.error(f"Помилка при завантаженні тестових даних: {e}")
    raise
svm_preds = final_svm_model.predict(test_X)
nb_preds = final_nb_model.predict(test_X)
rf_preds = final_rf_model.predict(test_X)
svm_probs = final_svm_model.predict_proba(test_X)
nb_probs = final_nb_model.predict_proba(test_X)
rf_probs = final_rf_model.predict_proba(test_X)
avg_probs = (svm_probs + nb_probs + rf_probs) / 3
final_preds = np.argmax(avg_probs, axis=1)
logging.info(
        f"Точність на тестовому наборі для комбінованої моделі: {accuracy_score(test_Y, final_preds) * 100:.2f}%"
)
cf_matrix = confusion_matrix(test_Y, final_preds)
plt.figure(figsize=(12, 8))
sns.heatmap(cf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Матриця помилок для комбінованої моделі на тестових даних")
plt.savefig(os.path.join(output_dir, "confusion_matrix_combined_model.png"))
plt.close()
logging.info("Графік 'Матриця помилок для комбінованої моделі' збережено")
symptoms = X.columns.values
symptom_index = {symptom: idx for idx, symptom in enumerate(X.columns)}
data_dict = {"symptom_index": symptom_index, "predictions_classes": encoder.classes_}
```

```
def predictDisease(symptoms):
    symptoms = symptoms.lower().split(",")
    input_data = [0] * len(data_dict["symptom_index"])
    unrecognized = []
    for symptom in symptoms:
        if symptom in data_dict["symptom_index"]:
            input_data[data_dict["symptom_index"][symptom]] = 1
        else:
            unrecognized.append(symptom)
    if unrecognized:
        logging.warning(f"Нерозпізнані симптоми: {unrecognized}")
    input_data = np.array(input_data).reshape(1, -1)
    rf_probs = final_rf_model.predict_proba(input_data)
    nb_probs = final_nb_model.predict_proba(input_data)
    svm_probs = final_svm_model.predict_proba(input_data)
    avg_probs = (rf_probs + nb_probs + svm_probs) / 3
    top_indices = np.argsort(avg_probs[0])[-5:][::-1]
    predictions = {
        data_dict["predictions_classes"][i]: round(avg_probs[0][i] * 100, 2)
        for i in top_indices
    }
    return predictions

test_symptoms = "кашель, головний_біль, втома, висока_температура"
logging.info(f"Тестування функції predictDisease з симптомами: {test_symptoms}")
print(predictDisease(test_symptoms))
try:
    joblib.dump(final_svm_model, "models/gen2/svm_model_UA12.joblib")
    joblib.dump(final_nb_model, "models/gen2/nb_model_UA12.joblib")
```

```
    joblib.dump(final_rf_model, "models/gen2/rf_model_UA12.joblib")
                                                joblib.dump(data_dict["symptom_index"],
"models/gen2/symptom_index_UA12.joblib")
    joblib.dump(
                                                    data_dict["predictions_classes"],
"models/gen2/predictions_classes_UA12.joblib"
    )
    logging.info("Моделі та словники успішно збережено")
except Exception as e:
    logging.error(f"Помилка при збереженні моделей: {e}")
    raise
```

Додаток Б

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import joblib
import numpy as np
import pandas as pd
import json
app = Flask(__name__)
CORS(app, origins=["http://localhost:5173"])
svm_model = joblib.load('models/gen2/svm_model_UA1.joblib')
nb_model = joblib.load('models/gen2/nb_model_UA1.joblib')
rf_model = joblib.load('models/gen2/rf_model_UA1.joblib')
data_dict = {
    "symptom_index": joblib.load('models/gen2/symptom_index_UA1.joblib'),
    "predictions_classes": joblib.load('models/gen2/predictions_classes_UA1.joblib')}
combined_data = pd.read_csv('combined_dataset.csv')
df_symptoms = pd.read_csv('unique_symptoms_UA.csv', header=None, sep=',')
all_symptoms = df_symptoms.values.flatten().tolist()
@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    symptoms = data.get('symptoms', [])
    input_data = [0] * len(data_dict["symptom_index"])
    for symptom in symptoms:
        symptom = symptom.strip().replace(" ", "_")
        if symptom in data_dict["symptom_index"]:
            index = data_dict["symptom_index"][symptom]
            input_data[index] = 1
    input_data = np.array(input_data).reshape(1, -1)
```

```
    svm_probs = svm_model.predict_proba(input_data)[0]
    nb_probs = nb_model.predict_proba(input_data)[0]
    rf_probs = rf_model.predict_proba(input_data)[0]
    avg_probs = (svm_probs + nb_probs + rf_probs) / 3
    top_indices = np.argsort(avg_probs)[::-1][:3]
    top_diagnoses = [{
            "disease": data_dict["predictions_classes"][i],
            "probability": round(avg_probs[i] * 100, 2)}
        for i in top_indices]
    final_prediction = data_dict["predictions_classes"][np.argmax(avg_probs)]
    response = {
        "ensemble": {
            "final_prediction": final_prediction,
            "top_diagnoses": top_diagnoses}}
    return jsonify(response)
@app.route('/symptoms', methods=['GET'])
def symptoms():
    query = request.args.get('q', '')
        matching_symptoms = [symptom for symptom in all_symptoms if
isinstance(symptom, str) and query.lower() in symptom.lower()]
    return jsonify(matching_symptoms)
@app.route('/diagnoses', methods=['GET'])
def combined_data_route():
    processed_data = combined_data.copy()
    processed_data = processed_data.applymap(
        lambda x: x.replace('_', ' ') if isinstance(x, str) else x)
    combined_dict = processed_data.to_dict(orient='records')
    return jsonify(combined_dict)
@app.route('/symptomlocation', methods=['GET'])
def symptomlocation():
```

```
    with open('assets/syptomsLocation.json', 'r', encoding='utf-8') as file:
        data = json.load(file)
    return jsonify(data)
if __name__ == '__main__':
    app.run(debug=True)
```

Додаток В

```
function PredictionPage() {
  const [inputValue, setInputValue] = useState("");
  const [suggestions, setSuggestions] = useState([]);
  const [selectedSymptoms, setSelectedSymptoms] = useState([]);
  const [diagnoses, setDiagnoses] = useState([]);
  const [activeTab, setActiveTab] = useState("diagnoses");
  const [symptomLocations, setSymptomLocations] = useState({});
// Функція для обробки введення тексту
  const handleInputChange = (event, { newValue }) => {
    setInputValue(newValue);
  };
  //Функція автодоповнення симптому
  const handleSuggestionsFetchRequested = async ({ value }) => {
    const lastSymptom = value.split(",").pop().trim();
    if (lastSymptom.length > 0) {
      try {const response = await axios.get(`/symptoms?q=${lastSymptom}`);
        setSuggestions(response.data);
      } catch (error) {
        toast.error("Сталася помилка, детальніше в консолі.");
        console.error("Error", error) }
    } else {  setSuggestions([])}
  };
  // Функція очищення автодоповнення
  const handleSuggestionsClearRequested = () => {
    setSuggestions([]);
  };
  // Функція для обробки вибору симптому
  const handleSuggestionSelected = (event, { suggestion }) => {
```

```
  setSelectedSymptoms([...selectedSymptoms, suggestion]);
  setSuggestions([]);
  setInputValue("");
 };
 // Функція для обробки симптомів в діагноз
 const handlePredict = async () => {
  try {
   if (selectedSymptoms.length === 0) {
    toast.error("Виберіть хоча б один симптом.");
    return;
   }
   const response = await axios.post("/predict", {
    symptoms: selectedSymptoms,
   });
   setDiagnoses(response.data.ensemble);
  } catch (error) {
   toast.error("Сталася помилка, детальніше в консолі.");
   console.error("Error in diagnosis prediction:", error);
  }
 };
 const handleRemoveSymptom = (index) => {
  const updatedSymptoms = [...selectedSymptoms];
  updatedSymptoms.splice(index, 1);
  setSelectedSymptoms(updatedSymptoms);
};
 const handleClearSelectedSymptoms = () => {
  setSelectedSymptoms([]);
  setInputValue("");
 };
 const inputProps = {
```

```
  placeholder: "Введіть назву симптома",
  value: inputValue,
  onChange: handleInputChange,
};
const renderSuggestion = (suggestion) => (
  <div className="suggestion-item">{suggestion}</div>
);
useEffect(() => {
  // Перевіряємо, чи доступний svgRef
  const svg = d3.select(svgRef.current);

  // Створення зум-функції
  const zoom = d3
    .zoom()
    .scaleExtent([0.5, 5]) // Мінімальний та максимальний рівень зуму
    .on("zoom", (event) => {
      const { transform } = event;
      setScale(transform.k); // Оновлення масштабу
      setPosition({ x: transform.x, y: transform.y }); // Оновлення позиції
    });

  // Додавання зуму до SVG
  svg.call(zoom);
  // Очищення подій, коли компонент розмонтовується або змінюється
  return () => {
    svg.on(".zoom", null); // Очищення події на відключення};
}, []); // Залежність порожній масив, щоб зум ініціалізувався лише один раз
useEffect(() => {
  const fetchSymptomLocations = async () => {
    try {const response = await axios.get("/symptomlocation");
```

```
      setSymptomLocations(response.data); } catch (error) {
      toast.error("Сталася помилка, детальніше в консолі.");
      console.error("Помилка:", error);}
    };
    fetchSymptomLocations();
  }, []);
```