Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему РОЗРОБКА ТА ДОСЛІДЖЕННЯ СИСТЕМИ ПРОГНОЗУВАННЯ РИЗИКІВ ЗАХВОРЮВАНЬ НА ОСНОВІ МЕТОДІВ МАШИННОГО НАВЧАННЯ
DEVELOPMENT AND RESEARCH OF A DISEASE RISK PREDICTION SYSTEM BASED ON MACHINE LEARNING METHODS

Виконав: студент(ка) 2 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Системи штучного інтелекту

ІНДИК А.А.
(ПРІЗВИЩЕ та ініціали)
Керівник ЛЕОЩЕНКО С.Д.
(ПРІЗВИЩЕ та ініціали)
Рецензент ПОЛЯКОВ М.О.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ІНДИКА Андрія Андрійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка та дослідження системи прогнозування ризиків захворювань на основі методів машинного навчання. Development and Research of a Disease Risk Prediction System Based on Machine Learning Methods
керівник проєкту (роботи) д-р. філос., доцент, ЛЕОЩЕНКО Сергій Дмитрович
(науковий ступінь, вчене звання, прізвище, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 30 " вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Вибір і обґрунтування структури системи.	4–5 тижні	Розділ 2
4	Основні рішення щодо реалізації компонентів	6–8 тижні	Розділ 3
	системи.		
5	Експлуатація, тестування та експериментальне	9 тиждень	Розділ 4
	дослідження програми.		
6	Оформлення пояснювальної записки та документів	10–11 тижні	Додатки
	до неї		
7	Нормоконтроль та рецензування.	12 тиждень	
8	Захист роботи.	12 тиждень	

Студент(ка) _____ Андрій ІНДИК
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Сергій ЛЕОЩЕНКО
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 122 с., 7 табл., 33 рис., 3 дод., 26 джерел.

ПРОГНОЗУВАННЯ, МАШИННЕ НАЧАННЯ, РИЗИКИ, ЗАХВОРЮВАННЯ, ВЕБЗАСТОСУНОК, PYTHON.

Об’єкт дослідження – процес прогнозування захворювань методами машинного навчання.

Предмет дослідження – методи та засоби для прогнозування ризиків захворювань.

Мета роботи – підвищення точності прогнозування захворювань методами машинного навчання.

Матеріали, методи та технічні засоби: структурне та об’єктно-орієнтоване програмування, клієнт-серверна архітектура, мови розмітки HTML і CSS, мови програмування Python, JavaScript, вибірка даних із медичними показниками для навчання моделей машинного навчання, персональний комп’ютер з процесором Intel Core I7 під управлінням операційної системи Microsoft Windows 11.

Результати. Створено програмний вебзастосунок, що забезпечує аналіз і прогнозування стану користувача за наданими медичними показниками за допомогою моделей машинного навчання.

Висновки. Розроблено вебзастосунок, що здійснює оброблення та аналіз медичних показників користувача із застосуванням моделей машинного навчання, що дає змогу визначати його стан і підтримувати процес моніторингу здоров’я.

Галузь використання – медична діагностика та моніторинг стану здоров’я користувачів у медичних закладах лікарями або в домашніх умовах пацієнтами.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 122 pages, 7 tables, 33 figures, 3 appendixes, 26 sources.

PREDICTION, MACHINE LEARNING, RISKS, DISEASE, WEB APPLICATION, PYTHON.

The object of study is a process of predicting diseases using machine learning methods.

The subject of study are methods and tools for predicting disease risks.

The purpose of the work is to increase the accuracy of disease prediction using machine learning methods.

Materials, methods, and tools: structured and object-oriented programming, client–server architecture, markup languages HTML and CSS, programming languages Python and JavaScript, dataset with medical indicators for training machine learning models, personal computer with an Intel Core i7 processor running Microsoft Windows 11.

Results. A web application has been created which enables analysis and prediction of a user's condition based on provided medical indicators using machine learning models.

Conclusions. A web application has been developed that processes and analyses user's medical indicators with machine learning models, allowing assessment of their condition and supporting health monitoring.

The field of use is medical diagnostics and health monitoring of users in healthcare institutions by doctors or at home by patients.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ACID – Atomicity, Consistency, Isolation, Durability;

API – Application Programming Interface;

AUC – Area Under the Curve;

CNN – Convolutional Neural Network;

CRUD – Create, Read, Update, and Delete;

CSS – Cascading Style Sheets;

CSV – Comma-separated Values;

FPR – False Positive Rate;

HTML – Hypertext Markup Language;

KNN – K-nearest Neighbors;

NB – Naïve Bayes;

ORM – Object-Relational Mapping;

RF – Random Forest;

ROC – Receiver Operating Characteristic;

SQL – Structured Query Language;

SVM – Support Vector Machine;

TPR – True Positive Rate;

СКБД – система керування базами даних.

ВСТУП

У сучасному світі машинне навчання активно застосовується у різних сферах комп’ютерних технологій – від обробки великих даних і розпізнавання образів до кібербезпеки та фінансового прогнозування. Особливо стрімко ці методи розвиваються у медицині, де щороку накопичується величезна кількість даних про пацієнтів: результати лабораторних аналізів, історії хвороб, дані медичних оглядів, показники пристроїв тощо. Ці дані мають великий потенціал для побудови систем, здатних автоматично оцінювати стан здоров’я людини та передбачати ризики розвитку захворювань.

Актуальність теми зумовлена зростанням потреби у створенні інтелектуальних медичних систем підтримки прийняття рішень, які допомагають лікарям своєчасно виявляти ризики захворювань і підвищують якість діагностики. Часто лікарі стикаються з проблемами, пов’язаними з неоднозначністю симптомів, похибками у даних або нестачею часу для аналізу всієї доступної інформації [1]. Використання методів машинного навчання дозволяє частково автоматизувати цей процес і забезпечити більш об’єктивну оцінку ризиків на основі статистичних закономірностей у даних. Таким чином, впровадження таких систем є важливим кроком до розвитку персоналізованої медицини та профілактичного підходу до охорони здоров’я.

Проблема прогнозування ризиків захворювань активно досліджується. Провідні роботи присвячено використанню методів логістичної регресії, випадкових лісів, підтримувальних векторних машин, нейронних мереж та ансамблевих підходів. Зокрема, у сучасних наукових публікаціях [2] наведено результати застосування цих методів для виявлення ризиків серцево-судинних, онкологічних, діабетичних та інфекційних захворювань. Більшість рішень орієнтовані на конкретні види патологій або закриті медичні бази даних, що обмежує можливості їх практичного використання у більш загальних або публічних системах.

Об’єктом дослідження є процес прогнозування ризиків захворювань за допомогою методів машинного навчання.

Предметом дослідження є моделі та алгоритми машинного навчання, що забезпечують побудову систем прогнозування на основі медичних показників користувача.

Метою роботи є підвищення точності прогнозування захворювань методами машинного навчання.

Для досягнення поставленої мети у роботі вирішувалися такі завдання:

– провести аналіз сучасних методів і систем прогнозування захворювань, розроблених на основі методів машинного навчання;

– розробити структуру та архітектуру вебзастосунку для прогнозування ризиків захворювань;

– реалізувати програмну модель системи та провести експериментальні дослідження її точності;

– оцінити отримані результати та сформулювати рекомендації щодо подальшого вдосконалення системи.

Методика дослідження базується на використанні методів машинного навчання, статистичного аналізу даних та моделювання. Для реалізації системи передбачається використання сучасних бібліотек Python для машинного навчання, а для створення вебінтерфейсу – фреймворк Flask. Ефективність моделі оцінюється на відкритих медичних наборах даних із застосуванням метрик точності, повноти, F1-міри та AUC-ROC.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області

Прогнозування ризиків захворювань є одним з ключових напрямів сучасної медичної інформатики. Його метою є виявлення ймовірності виникнення певних патологічних станів у пацієнтів на основі аналізу індивідуальних медичних даних, факторів ризику та статистичних закономірностей. Такий підхід дає змогу не лише виявляти хвороби на ранніх стадіях, а й розробляти персоналізовані профілактичні та лікувальні стратегії, що підвищує ефективність медичної допомоги та знижує навантаження на систему охорони здоров’я.

Машинне навчання відіграє важливу роль у реалізації систем прогнозування ризиків, адже воно дозволяє автоматично виявляти складні нелінійні взаємозв’язки між ознаками, які складно або неможливо описати традиційними статистичними методами. Завдяки алгоритмам машинного навчання медичні системи можуть навчатися на великих масивах історичних даних, а саме електронних медичних записах, результатах лабораторних аналізів, зображеннях медичної візуалізації тощо, і формувати прогностичні моделі, здатні узагальнювати нові випадки з високою точністю.

Методи машинного навчання класифікуються на три категорії відповідно до процесів навчання, що використовуються. Категорії: навчання з учителем, навчання без учителя та навчання з підкріпленням. У навчанні з учителем для навчання використовуються зразки даних із мітками категорій. Прикладами навчання з учителем є моделі класифікації та регресії. Алгоритми, що використовуються в цьому підході, включають дерево рішень, наївний баєсівський класифікатор тощо. У навчанні без учителя для навчання безпосередньо використовуються зразки даних без міток категорій. Основними прикладами підходу без учителя є методи кластеризації. Навчання з підкріпленням є сумішшю двох попередніх підходів. Воно

використовує агента, який знаходить правильну дію для досягнення загальної мети застосування [3].

Модель прогнозування характеризується як модель, що надає спосіб оцінити індивідуальний ризик пацієнта для наслідків захворювання. Зі зростанням таких прогностичних моделей виникає питання, коли, що і як їх використовувати. Ці моделі можуть навчатися з часом, задовольняючи потреби компанії, щоб реагувати на нову інформацію або погляди [2].

Існує два типи моделей прогнозування: моделі класифікації, які передбачають клас результату, та моделі регресії, які передбачають зв’язок між залежною змінною Y та незалежною змінною X. Різні базові та просунуті (наприклад нейронні мережі, регресія з підкріпленням, навчання з підкріпленням, векторні машини підтримки, класифікатор Баєса, K-найближчих сусідів, лінійна регресія) алгоритми проводять аналіз даних, статистичний аналіз та виявляють тенденції й закономірності. Машинне навчання та аналіз прогнозування відкривають широкі можливості для впровадження, але їхнє безсистемне використання значно обмежить здатність надавати корисні висновки, якщо не враховувати інтеграцію в повсякденні операції. Має бути забезпечена відповідна архітектура для підтримки цих інструментів, а також високоякісні дані для їхнього навчання, щоб повною мірою скористатися перевагами аналітикою прогнозування та машинного навчання [4].

У процесі створення моделі прогнозування ризиків захворювань важливим є систематичне проходження кількох взаємопов’язаних етапів, кожен з яких має власну роль і сутність. Спершу здійснюється збір, інтеграція та попередня обробка даних, що є основним ресурсом для будь-якої моделі. Вже на початку потрібно впоратись із проблемою неоднорідності та обсягу даних, а також забезпечити відповідність формату і якості для подальшого аналізу [5].

Далі відбувається етап попередньої обробки та підготовки даних: видалення або заповнення пропущених значень, нормалізація чи

стандартизація ознак, перетворення категоріальних змінних, видалення аномальних спостережень або корекція шуму. Неправильна або недостатня підготовка може суттєво погіршити результати моделі, знижуючи її надійність чи можливість узагальнювати [5].

Наступним ключовим етапом є вибір і створення ознак та зменшення розмірності даних. В сучасних підходах часто використовують методи зменшення розмірності чи кластеризації як допоміжні етапи перед побудовою моделі. Під цим мається на увазі відбір тих змінних, які мають найбільшу інформативність, а також формування нових змінних або агрегатів (наприклад, середній рівень показника за останні декілька місяців) [5].

Далі відбувається власне побудова моделі та навчання алгоритму машинного навчання чи глибинного навчання. На цьому етапі визначається структура моделі, підбираються гіперпараметри, здійснюється тренування на навчальній вибірці. Важливими є аспекти, такі як збалансованість класів, обробка незбалансованих даних і захист від перенавчання, оскільки моделі повинні працювати на нових, невідомих даних [5].

Після навчання моделі слідує етап оцінювання її ефективності. точність, F1-міра, специфічність і чутливість, часто застосовуються для вимірювання якості моделей прогнозування в охороні здоров’я. Оцінювання має проводитися не лише на навчальній, але й на тестовій або зовнішній вибірці, щоб встановити узагальнюваність [5].

І фінальним етапом є впровадження моделі в клінічну практику або систему підтримки прийняття рішень і супроводжується моніторингом її роботи та адаптацією в умовах нових даних. Цей етап може включати інтеграцію з інформаційними системами медичного закладу, забезпечення інтерпретованості моделі для лікарів, контроль за дотриманням етичних та правових стандартів, а саме приватності пацієнтів, та регулярне оновлення моделі новими даними [5].

Незважаючи на вдосконалення методів машинного навчання та систематизацію процесу моделювання, практичне застосування таких систем

супроводжується рядом викликів. Однією з ключових є неповнота, нерівномірність та зашумленість даних. У сфері охорони здоров’я дані часто надходять із різних джерел – електронних медичних записів, лабораторних систем, пристроїв моніторингу, соціальних опитувань тощо – і мають різну структуру, рівень точності та формат. Це створює труднощі для уніфікації, інтеграції та забезпечення цілісності набору даних, а також може призводити до систематичних похибок під час навчання моделей. Крім того, пропущені або некоректно зафіксовані значення, дублікати записів і відсутність часової узгодженості ускладнюють відтворюваність результатів і призводять до потенційного зміщення прогнозів [5].

Другою групою викликів є етичні та правові аспекти використання медичних даних. Системи прогнозування мають відповідати вимогам конфіденційності, безпеки та згоди пацієнтів, адже доступ до персональних медичних записів може становити ризик для приватності. Проблеми анонімізації даних, контроль доступу, а також питання етичного використання алгоритмів, здатних виявляти чутливі кореляції, залишаються предметом активних дискусій у медичній спільноті. Зазначається також, що моделі машинного навчання можуть відтворювати або навіть посилювати соціальні упередження, наявні в початкових даних, що створює ризики дискримінації у прийнятті клінічних рішень [5].

Великою проблемою є інтерпретованість моделей. Складні алгоритми часто працюють як «чорні скриньки», тобто видають результат без очевидного пояснення механізму його отримання. Відсутність прозорості знижує рівень довіри медичних фахівців до таких систем і може ускладнити їх впровадження у клінічну практику [5].

Не такою значною, але все одно вартої уваги є проблема узагальнюваності моделей. Системи, навчальні на даних однієї клініки або регіону, можуть виявляти суттєве погіршення ефективності при застосуванні в інших умовах [5].

1.2 Огляд і аналіз існуючих рішень

Для глибшого розуміння сучасного стану проблеми доцільно розглянути наявні наукові підходи та системи прогнозування ризиків захворювань, розроблені на основі методів машинного навчання.

Для прогнозування пацієнтів із діабетом, дослідники у роботі [6] використали фреймворк для розробки та оцінки моделей класифікації машинного навчання, таких як логістична регресія, KNN, SVM та RF. Метод машинного навчання був реалізований на базі даних Pima Indian Diabetes Database, яка містить 768 рядків і 9 стовпців. Точність прогнозування становить 83% [6]. Результати підходу до реалізації вказують на те, що логістична регресія перевершила інші алгоритми машинного навчання, крім того, було обрано лише структурований набір даних, але неструктуровані дані не враховувалися, модель також повинна бути реалізована в інших галузях охорони здоров'я, таких як серцеві захворювання та COVID-19, нарешті, для прогнозування діабету слід враховувати інші фактори, як-от сімейна історія діабету, звички куріння та фізична неактивність [6].

У роботі [7] розробили модель машинного навчання для прогнозування виникнення цукрового діабету 2 типу в наступному році (Y + 1) на основі факторів поточного року (Y). Набір даних було отримано у вигляді електронних медичних записів з приватного медичного інституту за період з 2013 по 2018 рік. Автори застосували логістичну регресію, RF, SVM, XGBoost та ансамблеві алгоритми машинного навчання для прогнозування результатів: не діабет, прediабет та діабет. Було використано вибір ознак для ефективного розрізнення трьох класів. Серед відібраних ознак: рівень глюкози, індекс маси тіла, стать, вік, сечова кислота, куріння, вживання алкоголю, фізична активність, сімейна історія діабету та ін. Згідно з експериментальними результатами, максимальна точність склала 73% [7] для RF, тоді як найнижча – 71% [7] для моделі логістичної регресії. Автори представили модель, яка використовувала лише один набір даних. Як

наслідок, для перевірки розроблених моделей слід застосовувати додаткові джерела даних [7].

В [8] запропонували використовувати гібридну інтелектуальну класифікацію для класифікації медичної інформації з метою прогнозування раку. Модель включає RF, класифікаційні дерева та дерево регресії, а також нейронну мережу min-max. Техніка RF застосовується для створення ансамблю моделей дерев класифікації та регресії. Нечіткий min-max використовується для навчання. Дерево класифікації та регресії застосовується для вилучення правил. Точність цієї моделі для прогнозування раку склала 98,84% [8].

Колектив дослідників у [9] використав метод видобутку даних для прогнозування серцевих захворювань. Спочатку він попередньо обробив пропущену інформацію за допомогою техніки середнього значення та моди, а також використав модель багатошарового перцептрона для мапування інформації. Для аналізу бази даних серцевих захворювань були застосовані NB, нейронні мережі та дерева рішень. Він зібрав 303 документи з репозиторію серцевих захворювань Клівленда і використав їх як навчальний набір, а також зібрав 270 документів як тестову інформацію з репозиторію Statlog Heart Disease. Набір даних складається з атрибутів, вхідних даних та ключового передбачуваного значення. Його модель забезпечує 90,74% [9] точності для NB [9].

Декілька дослідників у [10] запропонували ідею діагностики серцевих захворювань шляхом поєднання NB та кластеризації k-середніх із різними методами вибору початкових центроїдів. Даний підхід використовує набір даних Cleveland Clinic Foundation. Дослідження показало, що точність комбінованого методу k-середніх та NB складає 84,5% [10], що є значно кращим результатом порівняно з традиційними алгоритмами. Важливим аспектом є вибір початкових центроїдів для кластеризації k-середніх, що впливає на кінцевий результат. В експериментах було розглянуто кілька методів вибору центроїдів, таких як випадкові атрибути, випадкові рядки, а

також методи на основі інтервалів, внутрішніх та зовнішніх точок, що показали різні рівні точності у діагностиці хвороб серця [10].

З метою підвищення надійності та точності прогнозів для діабету 1 типу дослідники у [11] запропонували новий метод на основі CNN та глибинного навчання. Метою було визначити, як виявити поведінкові патерни. Для заповнення прогалин у даних використовувалося численні спостереження за однаковими поведінковими реакціями. Запропоновану модель навчали та перевіряли за допомогою даних 759 осіб з діабетом 1 типу, які відвідували Sheffield Teaching Hospitals у період з 2013 по 2015 рік. Кожен елемент у навчальному наборі складався з результату тесту на діабет 1 типу, демографічних даних (вік, стать, кількість років із діабетом) та останніх 84 днів (12 тижнів) самостійних вимірювань рівня глюкози в крові, що передували тесту. За словами авторів, точність прогнозів погіршується у разі недостатності даних та певних фізіологічних особливостей [11].

Перейдемо до прямого порівняння основних існуючих рішень, виберемо їх сильні та слабкі сторони та зведемо їх у табл. 1.1.

Більшість рішень використовують популярні методи, такі як SVM, NB та ансамблеві методи, для класифікації пацієнтів. Деякі підходи, зокрема використання комбінованих методів, таких як поєднання кластеризації k-середніх та NB, показали значне підвищення точності в порівнянні з традиційними методами. Те саме можна сказати про гібридну систему прогнозування раку. У ряді робіт підкреслено, що ансамблеві методи або гібридні підходи дають змогу компенсувати слабкі сторони окремих моделей, але водночас збільшують складність реалізації та залежність від коректної підготовки ознак. Водночас виявлено, що вибір початкових даних і специфіка даних (наприклад, відсутність інформації або фізіологічні особливості пацієнтів) можуть суттєво впливати на ефективність моделей. Моделі, що працюють із повними та добре структурованими наборами даних, демонструють високу точність , тоді як системи, побудовані на одноманітних

або обмежених вибірках, мають нижчі результати та потребують додаткової перевірки на різних даних.

Таблиця 1.1 – Порівняння існуючих рішень [6]-[11]

Система	Автор	Моделі	Точність	Обмеження
Прогнозування діабету	Крішнамурті та ін.	LR, KNN, SVM, RF	~83%	Використано лише структуровані дані; не враховано розширені клінічні фактори
Прогнозування діабету 2 типу	Деберне та Кім	LR, RF, SVM, XGBoost, ансамблі	до 73%	Модель перевірена лише на одному наборі електронних медичних записів
Прогнозування раку	Сіра та Лім	Гібрид RF, DT та min-max нейронної мережі	98,84%	Специфічність набору даних; складність моделі
Виявлення серцевих захворювань	Дангаре та ін.	NB, DT, нейронні мережі	90,74%	Використовуються два різні набори даних; можливий вплив різниці у збірці даних
Комбінований метод для діагностики серцевих патологій	Шуман, Тернер та Стокер	кластеризація K-means та NB	84,5%	Чутливість до вибору початкових центроїдів; нестабільність у складних просторах ознак

1.3 Постановка задачі та загальна методика дослідження

У межах цього дослідження розглядається задача прогнозування ризику захворювань на основі медичних даних пацієнтів. З огляду на доступність даних і клінічну значущість, основну увагу приділено випадку серцево-судинних захворювань. Вхідними даними є сукупність ознак, що відображають фізіологічний стан людини та результати обстежень. До них належать показники віку, статі, індексу маси тіла, рівня артеріального тиску, рівня глюкози й холестерину в крові, а також наявність шкідливих звичок. Кожен запис у вибірці характеризує одного пацієнта і представлений у

вигляді вектора числових та категоріальних ознак. На основі цих даних система має визначити вихідне значення, а саме рівень ризику розвитку захворювання. Результат може бути поданий як числова оцінка ймовірності або як одна з категорій ризику.

Задача прогнозування ризику захворювання формулюється як задача класифікації, у якій на основі вхідного вектора ознак *x* необхідно визначити цільову змінну *y*, що представляє клас ризику. Метою є знайти таку залежність між вхідними ознаками (вік, тиск, рівень холестерину та ін.) та вихідною змінною (рівнем ризику) щоб модель могла робити правильні передбачення для нових пацієнтів. Процес навчання полягає у поступовому підборі параметрів моделі так, щоб різниця між прогнозованими та реальними результатами була якомога меншою.

Для оцінювання ефективності побудованих моделей використовуються стандартні показники якості класифікації, а саме точність класифікації, прецизійність, повнота, F1-міра та AUC-ROC. Показник точності відображає частку правильно класифікованих прикладів, проте за суттєвого дисбалансу класів більш інформативними є прецизійність та повнота, які характеризують, відповідно, точність і повноту прогнозів. Комплексна оцінка здійснюється за допомогою F1-міри, тоді як AUC-ROC дозволяє оцінити здатність моделі відрізняти пацієнтів із різним рівнем ризику незалежно від вибраного порогу.

Загальна методика дослідження передбачає кілька послідовних етапів. Спочатку виконується попередня обробка даних, що охоплює очищення від пропусків, нормалізацію числових ознак і кодування категоріальних змінних. Після цього дані поділяються на навчальну й тестову вибірки, що дозволяє оцінити здатність моделі узагальнювати. Далі здійснюється навчання моделей із налаштуванням гіперпараметрів для досягнення оптимальної точності. На заключному етапі проводиться оцінювання ефективності кожного алгоритму за зазначеними метриками та порівняльний аналіз

результатів із метою визначення найпридатнішого підходу для прогнозування ризику серцево-судинних захворювань.

1.4 Опис вхідних даних

Для проведення дослідження було розглянуто декілька доступних наборів даних, що містять медичні показники, пов’язані з ризиком розвитку серцево-судинних захворювань. Попередній аналіз здійснювався з урахуванням їхнього обсягу, репрезентативності, актуальності, структури та якості даних, а також відповідності завданню побудови моделі прогнозування ризиків захворювань.

Перший набір даних, який було розглянуто – це набір даних серцевих захворювань (UCI) [12]. Цей набір є одним з найвідоміших і найчастіше використовуваних у дослідженнях машинного навчання. Він містить комбіновані дані з чотирьох незалежних джерел: Фундації клініки Клівленда, Угорського інституту кардіології, Швейцарського університетського шпиталю та Медичного центру ВА Лонг-Біч. Незважаючи на наявність добре структурованих ознак і численні приклади використання цього набору у науковій літературі, його недоліком є застарілість даних – останні записи датуються 1988 роком, що знижує репрезентативність вибірки в наш час. Окрім цього, зазначено, що цей набір містить пропущені значення та можливий шум. Еволюція методів діагностики, зміни стандартів лікування та загального рівня здоров’я населення роблять застосування цього набору менш доцільним для сучасних моделей прогнозування [12].

Другий – набір факторів ризику серцевих захворювань [13]. Цей набір має гарну структуру, кількість ознак та ідеальний розподіл за статями, віком. Після аналізу інформації про походження даних цього набору, було прийнято відхилити його використання через низьку якість і штучну природу даних. Набір було згенеровано з використанням бібліотеки NumPy, що призвело до відсутності реальних клінічних залежностей та статистичних

закономірностей, властивих справжнім медичним вимірюванням [13]. Використання таких синтетичних даних у дослідженні підвищило б ризик перенавчання моделей та спотворило результати оцінювання.

Останнім варіантом для розгляду став розширений набір даних для системи класифікації серцевих хвороб на базі Ербільського політехнічного університету та Університету Курдистану Гевлер, опублікований на платформі Mendeley Data у 2022 році [14]. Його ми надалі і розглядатимемо для нашого дослідження.

Метою створення набору було формування повної бази даних факторів, що спричиняють розвиток серцевого нападу (інфаркту міокарда). Вибірка включає 1319 спостережень, кожне з яких відповідає окремому пацієнту та 8 вхідних ознак [14]:

– вік (age) – числовий показник віку пацієнта у роках;

– стать (gender) – числовий показник статі (1 – чоловіча, 0 – жіноча);

– частота серцевих скорочень (Heart Rate / impulse) – кількість ударів серця за хвилину;

– систолічний артеріальний тиск (Systolic Blood Pressure / pressurehigh) – максимальний тиск у момент скорочення серцевого м’яза;

– діастолічний артеріальний тиск (Diastolic Blood Pressure / pressurelow) – мінімальний тиск у фазі розслаблення;

– рівень глюкози у крові (Blood Sugar / glucose) – концентрація цукру у плазмі крові натще;

– рівень креатинкінази-МВ (CK–MB / kcm) – біомаркер серцевого нападу, що характеризує пошкодження міокарда;

– рівень тропоніну (Test–Troponin / troponin) – біомаркер серцевого нападу, що характеризує пошкодження міокарда.

Вихідною ознакою є клас (class) наявності серцевого нападу з двома можливими символьними значеннями: наявний (positive) та відсутній (negative) [14].

Автори набору зазначили, що для забезпечення узгодженості та коректності даних перевірили відповідність записів клінічним карткам пацієнтів у лікарняному архіві, звірили їх із результатами лабораторних аналізів і провели консультації з медичними спеціалістами. Додатково було виконано перевірку на наявність пропущених значень і проведено нормалізацію числових параметрів у діапазоні від 0 до 1, що гарантує рівномірний вплив усіх ознак під час навчання моделі [14].

Дані сформовано у формат ARFF (Attribute-Relation File Format), який використовується переважно програмним забезпеченням Weka. Можливе подальша трансформація його у більш зручний формат типу CSV, або додаткова обробка у початковому форматі в зазначеній Weka.

1.5 Методи машинного навчання

Після того, як ми розглянули вже існуючі системи прогнозування ризиків захворювань, варто провести аналіз методів машинного навчання, які ці системи використовували та обрати ті, що підійдуть для використання у нашій роботі.

Логістична регресія, також відома як логістична модель, застосовується для дослідження взаємозв’язку між множиною незалежних змінних та категоріальною залежною змінною, а також для оцінювання ймовірності настання певної події шляхом апроксимації даних логістичною кривою. Значення залежної змінної у цьому методі мають бути дискретними та бінарними, тобто набувати лише двох можливих результатів: істина або хиба, 0 або 1, так або ні. Логістична регресія використовується у випадках, коли необхідно здійснити прогнозування категоріальних змінних та розв’язати задачі класифікації [15].

До переваг логістичної регресії належать простота інтерпретації та здатність моделі коректно відображати лінійний зв’язок між незалежними та залежною змінними, що забезпечує високу якість отримуваних результатів.

Разом з тим метод має і певні обмеження: він здатний прогнозувати лише числові значення ймовірностей, є малоефективним у випадках нелінійних залежностей у даних та демонструє чутливість до викидів, що може знижувати точність моделі [16].

Одним із найпоширеніших підходів залишається використання SVM (Support Vector Machine) для класифікації пацієнтів за рівнем ризику. Цей алгоритм виявив ефективність у прогнозуванні серцево-судинних захворювань, діабету та хвороб печінки завдяки здатності відокремлювати класи у багатовимірному просторі ознак навіть при обмеженому обсязі даних [2].

SVM – це лінійна модель, яка пропонує рішення для задач як лінійних, так і нелінійних. Її основою є ідея обчислення запасу (margin). Набір даних поділяється на кілька груп, щоб побудувати відношення між ними [15].

SVM ідеально працює як з напівструктурованими, так і з неструктурованими даними. Крім того, узагальнення SVM має менший ризик перерозподілу. З іншого боку, SVM має багато недоліків. Час навчання моделі збільшується на великому наборі даних. Вибір правильної функції ядра також є складним процесом. Також, він погано працює з даними з шумом [17].

Дерева рішень – це техніка навчання з учителем, яка використовується для класифікації. Воно комбінує значення атрибутів на основі їхнього порядку, як висхідного, так і низхідного. Як стратегія на основі дерева, Метод визначає кожен шлях, починаючи від кореня, за допомогою послідовності розділення даних, доки не буде досягнуто булевого висновку у листовому вузлі. Він є ієрархічним представленням взаємодій знань, що містить вузли та зв’язки. Коли відношення використовуються для класифікації, вузли відображають мету [17].

Дерева рішень мають різні недоліки, такі як збільшення складності зі зростанням номенклатури, невеликі зміни, які можуть призвести до іншої архітектури, та більший час обробки для навчання даних [17].

Випадковий ліс (Random Forest – RF) – це базова техніка, яка більшу частину часу дає правильні результати. Її можна використовувати для класифікації та регресії. Програма створює ансамбль дерев рішень і об'єднує їх [5].

У класифікаторі RF, чим більше дерев у лісі, тим точніші результати. Таким чином, RF генерує колекцію дерев рішень, яка називається лісом, і об'єднує їх для досягнення точніших прогнозів. У RF кожне дерево рішень будується лише на частині наданого набору даних і навчається на наближеннях. RF об'єднує кілька дерев рішень, щоб досягти оптимального рішення [15].

RF має значні переваги. По-перше, його можна використовувати для визначення важливості змінних у задачах регресії та класифікації. Ця важливість вимірюється за шкалою на основі зменшення нечистоти (impurity) у кожному вузлі, використаному для сегментації даних. По-друге, він автоматично обробляє пропущені значення в даних і вирішує проблему перенавчання дерев рішень. RF ефективно працює з великими наборами даних. З іншого боку, RF має недоліки. Наприклад, він потребує більше обчислень і ресурсів для генерації результатів, а також вимагає зусиль на навчання через залучення кількох дерев рішень [15].

K-найближчих сусідів (K-nearest neighbors – KNN) – це алгоритм «навчання на основі екземплярів» або неузагальненого навчання, який часто відомий як алгоритм «лінивого навчання». KNN використовується для розв'язання задач класифікації. Щоб передбачити цільову мітку нових тестових даних, KNN визначає відстань найближчих міток класів навчальних даних до нової тестової точки за наявності значення K. Потім він обчислює кількість найближчих точок даних за допомогою значення K і визначає мітку класу нових тестових даних. Для визначення кількості найближчих за

відстанню навчальних точок KNN зазвичай встановлює значення K як корінь розмір набору даних [17].

KNN має багато переваг. Наприклад, він досить потужний, якщо розмір навчальних даних великий. Він також простий і гнучкий завдяки атрибутам і функціям відстані. Крім того, він може працювати з багатокласовими наборами даних. KNN має багато недоліків, таких як складність вибору відповідного значення K, дуже трудомісткий вибір типу функції відстані для конкретного набору даних, а також дещо висока обчислювальна вартість через розрахунок відстаней між усіма точками навчальних даних [17].

Наївний Баєс (Naïve Bayes – NB) базується на ймовірнісній моделі теореми Баєса і простий у налаштуванні, оскільки складна рекурсивна оцінка параметрів фактично відсутня, що робить його придатним для великих наборів даних. NB визначає ступінь належності до класу на основі заданого позначення класу. Він сканує дані один раз, тому класифікація є легкою. Простіше кажучи, класифікатор NB припускає, що немає зв’язку між наявністю певної ознаки в класі та наявністю будь-якої іншої характеристики. Він переважно орієнтований на індустрію текстової класифікації [18].

NB має значні переваги, такі як легкість реалізації, здатність давати хороші результати навіть при використанні меншої кількості навчальних даних, можливість працювати як з неперервними, так і з дискретними даними, є ідеальним для розв’язання задач прогнозування з кількома класами, а також те, що нерелевантна ознака не впливає на прогноз. З іншого боку, NB має такі недоліки: він припускає, що всі ознаки незалежні, що не завжди реально в реальних задачах, страждає від проблеми нульової частоти, а прогноз NB зазвичай не є точним [17].

Згорткові нейронні мережі (Convolutional neural network – CNN) – це особливі типи нейронних мереж, натхненні зоровою корою людини та використовувані в комп’ютерному зорі. Це автоматична нейронна мережа

прямого поширення, в якій інформація передається виключно в прямому напрямку. CNN часто застосовується в розпізнаванні облич, локалізації органів людини, аналізі тексту та розпізнаванні біологічних зображень [19].

З моменту створення CNN у 1989 році, протягом останніх трьох десятиліть вони показали хороші результати в діагностиці захворювань. Кожен шар мережі приймає вихід попереднього шару як вхід і передає його наступному шару в шарах екстракції ознак. Типова архітектура CNN складається з трьох типів шарів: згортки, агрегування та класифікації. На нижніх і середніх рівнях мережі є два типи шарів: згорткові шари та шари агрегування. Парні шари використовуються для згорток, тоді як непарні – для операцій агрегування. Вихідні вузли шарів згортки та агрегування групуються в двовимірній площині, яка називається картою ознак. Кожен рівень шару зазвичай генерується шляхом комбінування одного або кількох попередніх шарів [19].

CNN мають багато переваг, зокрема систему обробки, подібну до людського зору, значно вдосконалену структуру обробки 2D- та 3D-зображень, а також ефективність у навчанні та вилученні абстрактної інформації з 2D-даних. Шар максимального агрегування в CNN ефективно враховує анізотропію форми. Крім того, вони побудовані на розріджених зв’язках із парними вагами та містять значно менше параметрів, ніж повністю з’єднана мережа такого ж розміру. CNN навчаються за допомогою алгоритму навчання на основі градієнта і менш схильні до проблеми зникаючого градієнта, оскільки градієнтний підхід навчає всю мережу безпосередньо зменшувати критерій помилки, що дозволяє CNN забезпечувати високооптимізовані ваги [19].

Перейдемо до прямого порівняння розглянутих методів, виберемо їх сильні та слабкі сторони та зведемо їх у табл. 1.2.

Підсумовуючи аналіз існуючих підходів, можна відзначити, що різні методи машинного навчання мають як свої переваги, так і суттєві обмеження, які визначають доцільність їх застосування у конкретних медичних задачах.

Логістична регресія, хоча й ефективна для бінарної класифікації, має суттєві обмеження у випадках складних, нелінійних залежностей, що часто спостерігаються в медичних даних.

Таблиця 1.2 – Порівняння існуючих рішень [5]

Метод	Переваги	Недоліки
Логістична регресія	Простота інтерпретації. Ефективність для бінарної класифікації. Невисокі вимоги до обчислювальних ресурсів	Обмежена здатність моделювати складні нелінійні залежності. Чутливість до викидів. Потреба у відсутності сильної взаємної кореляції між ознаками
Дерева рішень	Можуть обробляти категоріальні ознаки. Лише кілька параметрів для налаштування. Добре працюють з великою кількістю ознак	Сумнівна інтерпретованість ансамблю
RF	Висока точність навіть з поганими даними. Добре узагальнюється на нових даних. Інтерпретованість. Масштабується до великих наборів	Обчислювальна складність. Чутливість до гіперпараметрів
SVM	Високовимірний простір для введення. Мало нерелевантних ознак	Збір даних є часозатратним
KNN	Простий алгоритм	Необхідна кількість сусідів. Обчислювальна складність
NB	Простий і зрозумілий	Очікує менший навчальний набір. Припускає незалежність ознак
CNN	Чудово справляється з класифікацією зображень, виявленням об’єктів та сегментацією. Може обробляти шум	Вимагає великих наборів даних. Обчислювальна складність навчання

CNN, хоча й демонструють високу ефективність у сфері комп’ютерного зору, практично не використовуються у розглянутих

реальних моделях прогнозування, оскільки в цьому випадку не передбачається обробка зорової інформації, а також через потребу у великих наборах даних і значні обчислювальні витрати. Дерева рішень і RF забезпечують добру узагальнюваність та можливість роботи з різними типами ознак, проте характеризуються високою чутливістю до налаштувань і складністю масштабування. Водночас SVM залишається найпопулярнішим методом у сучасних дослідженнях прогнозування ризиків захворювань, що пояснюється його високою точністю, стійкістю до шуму, здатністю ефективно працювати у високовимірних просторах і помірними вимогами до обсягів даних. Саме тому у межах цього дослідження доцільно обрати SVM як базовий метод побудови прогностичної моделі, з можливістю доповнити його порівняльним аналізом альтернативних алгоритмів RF та NB для оцінки їхньої точності, узагальнюваності та інтерпретованості.

1.6 Методи оцінювання ефективності моделей

Методи оцінювання ефективності моделей машинного навчання забезпечують можливість об’єктивного аналізу результатів їх роботи та визначення відповідності отриманих прогнозів поставленим завданням. Оцінювання є необхідним етапом дослідження, оскільки воно дозволяє визначити не лише рівень точності моделі, а й збалансованість її рішень, стійкість до помилок і здатність узагальнювати знання на нових даних. У цьому підрозділі ми розглянемо принципи та підходи до вимірювання ефективності моделей машинного навчання.

Точність класифікації (Accuracy) – це частка всіх передбачень моделі, які були правильними, тобто як позитивних, так і негативних [20]. Метрика представляється формулою (1.1):

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (1.1)$$

де *TP* – кількість істинно позитивних випадків;

TN – істинно негативних;

FP – хибно позитивних;

FN – хибно негативних.

За збалансованої вибірки точність може служити базовим показником якості моделі. Проте у випадку незбалансованих даних, коли один клас значно переважає інший (наприклад, лише 5% позитивних прикладів), ця метрика може бути оманливою. Наприклад, модель, яка завжди передбачає негативний клас, може показати 95% точності, але не мати жодної практичної користі [20].

Повнота (Recall, або TPR, True Positive Rate) – це частка реальних позитивних прикладів, які модель правильно ідентифікувала як позитивні. Повнота показує, наскільки ефективно модель виявляє всі випадки позитивного класу [20]. Метрика представляється формулою (1.2):

$$Recall = \frac{TP}{TP+FN}. \quad (1.2)$$

У медичних задачах, зокрема при прогнозуванні захворювань, цей показник має особливе значення, оскільки пропущений позитивний випадок (хибно негативний результат) може мати серйозні наслідки. Тому для таких завдань частіше оптимізують саме повноту.

Частота хибно позитивних результатів (FPR, False Positive Rate) – це частка негативних прикладів, які модель помилково класифікувала як позитивні. Цей показник також називають ймовірністю хибного спрацьовування. Він корисний у випадках, коли важливо мінімізувати кількість помилкових попереджень [20]. Метрика представляється формулою (1.3):

$$FPR = \frac{FP}{FP+TN}. \quad (1.3)$$

Точність позитивних передбачень (Precision) – це частка передбачених як позитивних прикладів, що справді є позитивними. Висока точність означає, що більшість позитивних рішень моделі є правильними [20]. Метрика представляється формулою (1.4):

$$Precision = \frac{TP}{TP+FP}. \quad (1.4)$$

Цей показник важливий, коли помилкові спрацьовування коштують дорого – наприклад, у разі діагностики рідкісного захворювання, де кожне помилкове виявлення призводить до зайвих обстежень пацієнта.

F1-міра – це збалансована оцінка, вона є гармонійним середнім між точністю та повнотою. Ця метрика дозволяє оцінити баланс між виявленням позитивних випадків і точністю передбачень. Вона особливо корисна для незбалансованих вибірок, де проста точність не відображає реальної ефективності моделі [20]. Метрика представляється формулою (1.5):

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall} = \frac{2TP}{2TP+FP+FN}. \quad (1.5)$$

Окрім базових метрик, що розраховуються для одного фіксованого порогу, важливо оцінювати модель у межах усіх можливих порогових значень. Для цього застосовуються графічні методи, а саме ROC-крива (Receiver Operating Characteristic) та площа під кривою (AUC, Area Under the Curve).

ROC-крива є графічним відображенням співвідношення між TPR і FPR при зміні порогу класифікації. Ідеальна модель має TPR = 1 та FPR = 0, тобто повністю відділяє позитивні приклади від негативних [21].

На практиці ROC-крива будується шляхом послідовного обчислення TPR та FPR для низки можливих порогових значень класифікації, після чого результати зображуються на координатній площині, де вісь X відповідає

FPR, а вісь Y – TPR. Ідеальна модель, яка для певного порогу повністю відділяє позитивні приклади від негативних, характеризується точкою у координатах (0;1), тобто має максимальну чутливість без жодної кількості хибних спрацьовувань [21].

Ключовим узагальненим показником, який отримують на основі ROC-кривої, є площа під нею – AUC. Значення AUC показує ймовірність того, що модель призначить вищу оцінку випадково обраному позитивному прикладу, ніж випадково обраному негативному. Таким чином, AUC = 1.0 відповідає ідеальному класифікатору, що завжди коректно розрізняє обидва класи (рис. 1.1), тоді як AUC = 0.5 відображає поведінку, еквівалентну випадковим здогадкам або підкиданню монети (рис. 1.2). Чим більша площа під кривою, тим кращою вважається загальна якість моделі, незалежно від конкретного порогу класифікації [21].

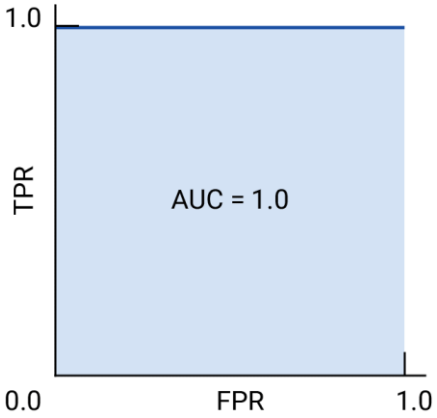


Рисунок 1.1 – Гіпотетична ідеальна модель [21]

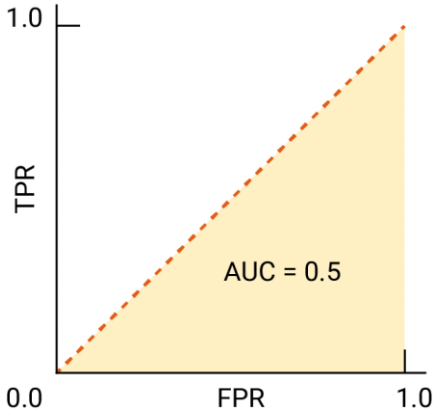


Рисунок 1.2 – Випадкові здогадки [21]

ROC-крива та показник AUC є зручними для аналізу моделей на збалансованих вибірках, де кількість позитивних і негативних прикладів приблизно однакова. Проте у випадках значного дисбалансу класів, наприклад у медичних даних, де позитивні випадки трапляються значно рідше, інтерпретація ROC-кривої може бути спотвореною. У таких випадках доцільно застосовувати альтернативний інструмент – криву Precision-Recall (PRC), що будується на співвідношенні точності та повноти (рис. 1.3) і краще відображає ефективність моделі при роботі з рідкісним позитивним класом [21].

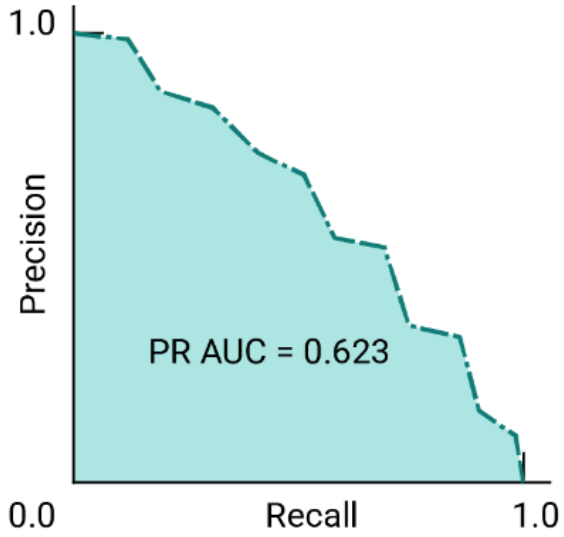


Рисунок 1.3 – Precision-Recall-крива [21]

Підсумовуючи, вибір метрик оцінювання залежить від специфіки задачі та наслідків помилок. У прогнозуванні серцевих захворювань особливо важливо мінімізувати пропущені випадки, тому перевагу надають повноті та F1-мірі. Для загальної оцінки якості моделі використовують ROC-криву та AUC, що показують здатність моделі розрізняти класи незалежно від порогу. Поєднання кількох метрик дає збалансоване уявлення про ефективність моделі.

1.7 Висновки до розділу 1

У цьому розділі було проведено аналіз предметної області прогнозування ризиків захворювань та визначено роль методів машинного навчання у сучасній медичній практиці, було проведено системний аналіз вихідних даних, методів машинного навчання та способів оцінювання ефективності моделей, що є основою для подальшого етапу практичної реалізації дослідження. Показано, що прогнозування ґрунтується на виявленні закономірностей у медичних даних з метою оцінювання імовірності розвитку патологій, а машинне навчання забезпечує підвищення точності та об’єктивності таких оцінок. Розглянуто основні етапи побудови моделей прогнозування й типові труднощі, пов’язані з неповнотою даних, етичними аспектами та потребою у зрозумілості моделей.

Огляд сучасних наукових робіт засвідчив, що для розв’язання подібних задач застосовують різні алгоритми та їх суміші – RF, NB та SVM, глибоке навчання. Комбіновані методи показали значне підвищення точності в порівнянні з традиційними методами. Виявлено вплив специфіки даних на ефективність моделей.

Сформульовано загальну постановку задачі прогнозування, визначено структуру вхідних і вихідних даних, обрано основні критерії оцінювання ефективності моделей. Як предметну область дослідження передбачено прогнозування ризику серцево-судинних захворювань. Отримані результати створюють теоретичне підґрунтя для реалізації практичної частини роботи, де буде здійснено побудову, навчання й тестування моделей.

Проведений огляд існуючих наборів даних дав змогу обґрунтовано обрати сучасну та якісну вибірку, яка містить актуальні медичні показники, що безпосередньо впливають на ризик розвитку серцевих захворювань. Аналіз наявних алгоритмів машинного навчання дозволив визначити найбільш придатні методи для задачі прогнозування – серед них найбільш доцільним обрано SVM завдяки його високій точності, здатності працювати з

обмеженими даними та стійкості до шуму. Також було систематизовано підходи до оцінювання ефективності моделей, що забезпечують об’єктивну інтерпретацію результатів. Зокрема, розглянуті метрики точності, повноти, F1-міри, а також інструменти ROC-кривої, AUC і Precision–Recall-кривої, які дозволяють всебічно оцінити збалансованість і якість моделі.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вимоги до програмної системи

У цьому підрозділі буде представлено детальний аналіз вимог до програмної системи, що розробляється. Зокрема, розглянемо функціональні вимоги, що включають завдання та функції програми; нефункціональні, що включають вимоги до зручності, безпеки, продуктивності; а також технічні вимоги, що стосуються апаратного забезпечення, операційної системи, використовуваних бібліотек і середовища розробки.

Функціональні вимоги визначають основні можливості та завдання, які має виконувати програмна система для забезпечення коректної та ефективної роботи користувачів. У нашому випадку програмний продукт повинен реалізовувати наступні ключові функції:

– реєстрація та авторизація користувачів, що дозволить захистити персональні дані та гарантувати індивідуальний підхід до роботи із системою;

– обробка введених користувачем даних, включаючи валідацію форм, перевірку коректності та збереження інформації у базі даних;

– використання моделей машинного навчання для аналізу та прогнозування на основі введених даних;

– збереження результатів прогнозів із прив’язкою до користувача та часу їх отримання для подальшого перегляду та аналізу;

– логування основних дій користувачів для відстеження роботи системи та виявлення можливих помилок або зловживань;

– забезпечення можливості зміни моделі машинного навчання і повторного аналізу введених даних, а також оновлення моделей машинного навчання.

Ці функції покривають основні сценарії використання системи і визначають межі її функціональності, що є базою для подальшої розробки і

тестування. Забезпечення їхньої реалізації є пріоритетом для задоволення потреб кінцевих користувачів і досягнення поставлених цілей проєкту.

Нефункціональні вимоги визначають загальні характеристики та властивості системи, які забезпечують комфортне, безпечне та ефективне користування програмою. Вони охоплюють аспекти безпеки, продуктивності, стабільності, масштабованості, зручності інтерфейсу та сумісності з різними платформами. Ці вимоги є важливими для підтримки належного рівня якості та надійності роботи вебзастосунку.

– забезпечення високого рівня безпеки даних користувачів, включно з захистом паролів за допомогою надійних алгоритмів хешування;

– забезпечення стабільної роботи системи при помірному навантаженні без значних затримок у відповіді;

– оптимізація продуктивності при обробці запитів і взаємодії з моделями машинного навчання для швидкого отримання результатів;

– забезпечення кросплатформеності роботи застосунку, сумісність із сучасними операційними системами та вебпереглядачами;

– підтримка масштабованості архітектури для можливості подальшого розширення функціоналу та збільшення кількості користувачів;

– організація зручної системи повідомлень і помилок для покращення користувацького досвіду та швидкого виявлення проблем;

– організація простого і зрозумілого інтерфейсу для навігації між різними функціональними модулями системи;

– відображення результатів у зрозумілому вигляді, що забезпечує зручний та наочний інтерфейс для користувачів.

Таким чином, нефункціональні вимоги створюють основу для високої якості програмного забезпечення та позитивного користувацького досвіду.

Технічні вимоги визначають оптимальні характеристики апаратного та програмного забезпечення, необхідні для коректної роботи проєкту, а також середовище розробки, яке забезпечує ефективний процес створення та підтримки системи.

Для підтримки розробки та експлуатації проєкту необхідні такі технічні характеристики:

– операційна система Windows 11 Pro;
– процесор Intel Core i7-13620H;
– оперативна пам’ять обсягом 16 ГБ;
– монітор з роздільною здатністю 1920x1080 пікселів;
– вебпереглядач на основі Chromium, версії 142, який підтримує всі необхідні технології для коректного відображення;
– середовище розробки PyCharm Community Edition версії 2025.2.4;
– інтерпретатор мови Python версії 3.12.10.

Працездатність проєкту, а також можливість подальшого розширення і модернізації гарантовані при дотриманні цих технічних вимог. Водночас, застосунок може коректно функціонувати і на системах зі схожими, нижчими або вищими характеристиками.

2.2 Вибір та обґрунтування інструментів розробки

2.2.1 Вибір мови програмування

Під час проєктування вебзастосунку важливим етапом є визначення мови програмування, на основі якої будуватиметься серверна логіка системи. Вибір мови впливає на продуктивність, зручність розробки, доступність бібліотек, можливості інтеграції та масштабування.

Розробка серверної частини вебзастосунку – це критичний процес, що охоплює такі завдання, як управління базами даних, конфігурація серверів та створення логіки застосунку. Запити користувачів передаються на серверну частину через клієнтську частину, де вони обробляються для отримання необхідної інформації з бази даних або для збереження нової інформації в базі даних [22].

Мови та технології, що використовуються в розробці бекенду, варіюються залежно від потреб і вимог проєкту. Популярні мови, такі як

Python, Java та Node.js, вирізняються своїми унікальними перевагами та сферами застосування. Наприклад, Python ідеально підходить для швидкого прототипування та аналізу даних, тоді як Java пропонує надійну платформу для великомасштабних корпоративних додатків. Node.js часто обирають для додатків реального часу та проєктів, що вимагають високої продуктивності [22]. Розглянемо використання цих варіантів у більших деталях.

Python – це універсальна та потужна мова програмування, яка дедалі більше набирає популярності у своїй галузі. Її легкий для читання синтаксис, широка підтримка бібліотек та різноманітні фреймворки зробили її привабливим варіантом як для початківців, так і для досвідчених розробників. Її застосовність у широкому спектрі додатків, як від великомасштабних вебзастосунків, так і до проєктів аналізу даних, робить Python ідеальним інструментом для сучасної розробки серверної частини [22].

Фреймворки є ключовим фактором успіху Python у розробці серверної частини. Популярні фреймворки, такі як Django та Flask, надають розробникам інструменти, необхідні для швидкої та безпечної розробки застосунків. Ці фреймворки значно зменшують навантаження на розробників, спрощуючи складні завдання, як-от операції з базами даних, авторизація користувачів та створення API. Крім того, велика спільнота цих фреймворків полегшує пошук рішень для наявних проблем [22].

Python також часто обирають для проєктів, орієнтованих на дані. Бібліотеки, такі як Pandas та NumPy, пропонують потужні інструменти для маніпуляції та аналізу даних. Це дозволяє легко виконувати завдання серверної частини, як-от обробка даних, створення звітів та аналіз. Крім того, бібліотеки, такі як TensorFlow та PyTorch, зробили Python незамінною мовою для застосунків машинного навчання та штучного інтелекту [22].

Java існує вже багато років. Це зріла та надійна мова, яка посідає значне місце у світі програмування. Її часто обирають для розробки великомасштабних корпоративних застосунків, фінансових систем та

мобільних застосунків на базі Android. Незалежність від платформи, потужне керування пам’яттю та широка підтримка бібліотек роблять Java ідеальним вибором для складних проєктів [22].

Java є об’єктно-орієнтованою мовою програмування, що полегшує написання модульного та повторно використовуваного коду. Її потужна статична типізація допомагає виявляти помилки під час розробки, а автоматичне керування пам’яттю запобігає проблемам, як-от витоки пам’яті. Завдяки віртуальній машині Java (JVM) вона може безперебійно працювати на різних платформах за принципом «напиши один раз – запускай будь-де» [22].

Екосистема Java пропонує потужні фреймворки, такі як Spring та Hibernate. Spring є комплексним фреймворком для застосунків корпоративного рівня, що надає розширені функції, як-от впровадження залежностей та аспектно-орієнтоване програмування. Hibernate є інструментом об’єктно-реляційного відображення, який спрощує операції з базами даних. Ці фреймворки прискорюють розробку та допомагають створювати більш підтримувані застосунки [22].

Недоліки Java включають необхідність писати більше коду та довші часи компіляції порівняно з іншими мовами. Проте ці недоліки компенсуються перевагами в надійності, продуктивності та масштабованості. Java досі вважається одним із найкращих варіантів, особливо для складних і критичних систем [22].

Node.js – це середовище виконання на основі JavaScript з архітектурою, що керується подіями, та неблокуючою моделлю вводу-виводу. Це ідеальний вибір для застосунків, що вимагають високої продуктивності та інтенсивно використовують одночасні з'єднання. Завдяки неблокуючій моделі вводу-виводу Node.js може одночасно обробляти кілька запитів в одному потоці, що робить його надзвичайно ефективним для застосунків реального часу [22].

Node.js особливо популярний у застосунках, як от чат-застосунки, онлайн-ігри, платформи для потокової передачі даних у реальному часі та застосунки інтернету речей. Наприклад, коли серверна частина онлайн-гри розроблена з Node.js, дії та взаємодії гравців можуть миттєво передаватися іншим гравцям. Це значно покращує користувацький досвід [22].

Популярність Node.js також зумовлена його широкою відомістю та використанням. Розробники клієнтської частини можуть легко працювати над серверною частиною завдяки Node.js, що розширює їхні можливості повностекової розробки. Одна й та сама мова може використовуватися як для клієнтської, так і для серверної частини. Це прискорює процес розробки та полегшує координацію в команді. Крім того, спільнота Node.js надзвичайно активна та підтримуюча, що допомагає швидко знаходити рішення проблем, з якими стикаються під час розробки [22].

Наведені варіанти мов програмування демонструють широкий спектр можливостей для створення серверної частини вебзастосунку, однак кожна з них оптимально підходить для різних контекстів використання. Java вирізняється високою надійністю, потужною екосистемою фреймворків та здатністю підтримувати складні багаторівневі корпоративні системи. Попри це, її застосування зазвичай доцільне тоді, коли передбачаються значні обсяги навантаження, масштабування та складна бізнес-логіка, чого в межах цього проєкту не очікується. Node.js натомість пропонує легкість, високу швидкість обробки одночасних запитів і спрощений перехід між клієнтською та серверною частиною завдяки єдиній мові програмування. Це робить його чудовим вибором для застосунків реального часу, проте він менш природно інтегрується з інструментами аналізу даних та машинного навчання.

У контексті системи, що поєднує вебфункціональність із машинним навчанням, Python виявляється найбільш збалансованим рішенням. Його синергія з бібліотеками для аналізу даних, моделювання та візуалізації, а також наявність фреймворків, що спрощують побудову серверної частини, забезпечують оптимальне середовище для інтеграції моделей прогнозування

з вебінтерфейсом. Це дає можливість уникнути складних проміжних шарів взаємодії між мовами та значно прискорює процес розробки.

2.2.2 Вибір вебфреймворку

Вибір вебфреймворку так само важливий, як і вибір мови програмування. Правильне рішення допоможе стартувати швидше, зменшити на розробку та плавніше масштабуватися за потреби. З іншого боку, неправильний фреймворк може перетворити розробку вебзастосунку на затяжний, важкий і фруструючий досвід [23].

Django є повноцінним вебфреймворком, орієнтованим на використання принципу «все в одному пакеті». Він пропонує широкий набір інтегрованих інструментів, доступних безпосередньо після встановлення, що дає можливість розробникам зосередитися на реалізації прикладної логіки, а не на виборі та поєднанні окремих технологій. Завдяки такому підходу Django часто розглядають як фреймворк, придатний для корпоративних систем та застосунків, яким необхідно забезпечувати стабільну роботу за умов значного навантаження. До його складу входять система шаблонів, вбудоване об’єктно-реляційне відображення, а також повноцінна адміністративна панель [23].

Адміністративна панель є однією з ключових переваг Django, оскільки вона істотно спрощує реалізацію операцій створення, читання, оновлення та видалення даних (CRUD). Для більшості внутрішніх або службових сценаріїв цього інструменту цілком достатньо для оперативного керування даними та швидкого розгортання функціональності. Водночас використання панелі у роботі з кінцевими користувачами зазвичай потребує додаткового опрацювання інтерфейсу, що передбачає участь фахівців з дизайну [23].

Django також характеризується як повностековий та частково монолітний фреймворк, який дотримується принципу «конвенція понад конфігурацію». Цей підхід зменшує гнучкість у налаштуванні окремих

компонентів, однак водночас забезпечує структурованість, передбачуваність та спрощену підтримку застосунків, оскільки більшість архітектурних рішень стандартизована на рівні фреймворку [23].

У разі, якщо проєкт починається з невеликого застосунку з наміром подальшого масштабування, часто складно заздалегідь передбачити точний напрямок розвитку продукту. Протягом кількох місяців архітектура та функціональні вимоги можуть суттєво змінитися, що є типовою практикою в сучасній розробці [23].

Django виступає ефективним інструментом у таких умовах, оскільки надає повний набір функцій і готових рішень, які покривають більшість потреб на різних етапах росту застосунку. Це дозволяє суттєво спростити адаптацію системи під нові вимоги без необхідності кардинальної перебудови. Загалом, починати розробку з використанням Django є безпечним вибором, що забезпечує гнучкість і масштабованість для майбутнього розвитку проєкту [23].

Великі компанії часто віддають перевагу Django через його комплексність і стандартизований, упорядкований процес розробки. Хоча певна негнучкість фреймворку може сприйматися розробниками як обмеження, з точки зору бізнесу вона забезпечує всебічне охоплення ключових аспектів проєкту та підготовку до складних викликів, з якими інші фреймворки можуть впоратися менш ефективно. Такий підхід суттєво знижує витрати на підтримку й обслуговування системи в довгостроковій перспективі [23].

Flask є мікрофреймворком, який надає базовий набір функціональних можливостей для розробки вебзастосунків. Його перевага полягає в простоті старту та легкості розширення завдяки широкому спектру доступних бібліотек і розширень Python [23].

На відміну від повноцінних фреймворків, Flask має обмежений функціонал «з коробки», тому додаткові можливості реалізуються через сторонні компоненти, що дає розробникам велику свободу у виборі

технологій і архітектурних рішень. Цей підхід контрастує з підходом Django, де пропонується готовий набір інтегрованих інструментів і чітко визначена структура проєкту [23].

Flask відзначається високою гнучкістю, що робить його оптимальним вибором для проєктів, де потрібна адаптивність і можливість змінювати архітектуру та технологічний стек у процесі розробки. Це особливо актуально, коли відсутні чіткі вимоги щодо масштабування або структури застосунку на початкових етапах [23].

Однією з ключових переваг Flask є його простота та якісна документація, що значно полегшує процес ознайомлення з фреймворком для розробників різного рівня підготовки. Така легкість сприйняття має суттєве значення у практичному застосуванні, оскільки дозволяє швидко розробляти та впроваджувати невеликі функціональні модулі, прискорюючи загальний цикл створення програмного забезпечення [23].

Streamlit – це сучасний фреймворк, спеціально розроблений для створення вебзастосунків, орієнтованих на аналіз і візуалізацію даних. Він дозволяє розробникам швидко створювати інтерактивні та настроювані інтерфейси, використовуючи виключно мову Python. Завдяки простому та інтуїтивному API, Streamlit значно спрощує процес розробки візуалізацій, що робить його привабливим інструментом для фахівців із даних [24].

Фреймворк підтримує безшовну інтеграцію з популярними бібліотеками машинного навчання, такими як TensorFlow та PyTorch, що сприяє його широкому застосуванню в проєктах, пов’язаних із штучним інтелектом. Окрім цього, функція живого перегляду змін під час розробки дає можливість оперативно тестувати і вдосконалювати застосунки [24].

Водночас, на відміну від Flask або Django, Streamlit має обмежену гнучкість і не призначений для розробки складних чи масштабованих вебзастосунків. Його архітектура оптимізована для специфічних задач, пов’язаних із візуалізацією та аналітикою даних, а не для широкомасштабних корпоративних рішень із багатофункціональною серверною частиною [24].

Виділимо основні особливості кожного фреймворку та зведемо їх у табл. 2.1.

Таблиця 2.1 – Порівняння вебфреймворків

Особливість	Django	Flask	Streamlit
Архітектурний підхід	Монолітний	Диверсифікований	Односторінковий інтерфейс
Сфера застосування	Корпоративні системи	Мікросервіси, прототипи	Демонстрація даних, моделей, аналітика
Гнучкість	Вбудовані рішення	Підбір власних рішень	Орієнтований лише на свою нішу
Вбудовані інструменти	Об’єктно-реляційне відображення, автентифікація, форми, шаблони, адмін-панель	Мінімальна маршрутизація та шаблони. Все інше через бібліотеки	Віджети, перезавантаження в реальному часі, візуалізація. Щось складніше не передбачено
Масштабованість	Ідеальна	Потребує уваги до вибраних бібліотек	Не масштабується
Обмеження	Дотримування стандарту, закрита архітектура	Відсутність вбудованих готових рішень	Не підходить для складної логіки

Підсумовуючи, можна виділити, що Django є потужним, добре структурованим і формалізованим фреймворком, який ідеально підходить для масштабних проєктів із чіткими бізнес-вимогами. Він забезпечує повний набір інтегрованих інструментів і стандартизовану архітектуру, що сприяє стабільності та прогнозованості розвитку продукту, однак його компоненти складно замінити або суттєво модифікувати, тому його використання краще залишити для більших проєктів. Натомість, Flask пропонує легкий та гнучкий підхід, який дозволяє швидко стартувати та поступово додавати необхідні функціональні блоки, надаючи розробникам свободу у виборі технологій і архітектури. Streamlit, у свою чергу, орієнтований насамперед на побудову

вебзастосунків для роботи з даними і машинним навчанням, забезпечуючи прості й інтерактивні візуалізації, проте він обмежений у можливостях щодо складної архітектури, управління користувачами та масштабованості.

2.2.3 Вибір системи керування базами даних

Системи керування реляційними базами даних (СКРБД, або просто СКБД) є одними з найпоширеніших систем керування базами даних у світі. Засновані на реляційній моделі, ці бази даних зберігають дані у вигляді таблиць і дозволяють пов’язувати дані шляхом встановлення зв’язків між таблицями. Це створює ефективний механізм зберігання даних, де на дані можна посилатися з інших частин бази даних [25].

Розглянемо три найпопулярніші СКБД з відкритим кодом, порівняємо їхні можливості та оберемо найкращий варіант для нашої роботи.

SQLite – це вбудована СКБД на основі файлів, яка не потребує жодного встановлення чи налаштування. Це, у свою чергу, означає, що застосунок не працює під окремим серверним процесом, який потрібно запускати, зупиняти чи конфігурувати. Така безсерверна архітектура забезпечує кросплатформну сумісність бази даних [25].

Повна база даних міститься в одному файлі на диску, і всі операції читання та запису відбуваються безпосередньо з цим файлом. Оскільки дані записуються безпосередньо на дисковий файл, SQLite дотримується властивостей ACID (атомарність, узгодженість, ізоляція, довговічність), щоб захистити транзакції від збоїв розподілу пам’яті та помилок вводу-виводу на диск, які можуть статися через несподівані збої системи чи відключення живлення [25].

Бібліотека SQLite є однією з найкомпактніших, розмір бібліотеки може легко бути меншим за 600 КБ. Завдяки дуже малому обсягу та природі СКБД вона чудово підходить для пристроїв IoT та вбудованих систем [25].

Деякі інші хороші випадки використання включають вебсайти з низьким або середнім трафіком (прибл. 100 тис. запитів на день), тестування та внутрішні цілі розробки, аналіз даних за допомогою Python, а також освітні цілі (її просто налаштувати, і її можна використовувати для навчання студентів концепціям SQL) [25].

Однією з головних переваг SQLite є те, як вона може виступати додатковим рішенням для клієнт-серверних корпоративних СКБД. Наприклад, вона може кешувати дані з клієнт-серверної СКБД локально, тим самим зменшуючи затримку запитів і підтримуючи працездатність кінцевого застосунку в разі збоїв корпоративної СКБД [25].

Одним із недоліків SQLite є обмежені можливості керування доступом та користувачами порівняно з повноцінними серверними СКБД на кшталт MySQL чи PostgreSQL. У ній відсутня вбудована система ролей, розмежування прав та механізми складної авторизації – фактично вся логіка контролю доступу реалізується на рівні застосунку [25].

Оскільки SQLite є файловою СКБД, вона може спричиняти проблеми з продуктивністю при роботі з великими наборами даних через обмеження файлової системи. У таких випадках доцільно обирати клієнт-серверні бази даних, які будуть розглянуті далі [25].

MySQL є однією з найпопулярніших систем СКБД з відкритим кодом і великого масштабу. На відміну від SQLite, вона використовує архітектуру клієнт-сервер, що складається з багатопотокового SQL-сервера. Багатопотокова природа MySQL забезпечує вищу продуктивність, оскільки потоки ядра можуть легко використовувати кілька процесорів. Вона також дотримується системи ACID для узгодженості транзакцій і надає різні конектори та API, як-от C, C++, Java, PHP тощо [25].

MySQL вирізняється поєднанням масштабованості та надійних механізмів безпеки, що робить її поширеним вибором для вебзастосунків середнього та великого масштабу. Система підтримує гнучке керування доступом до даних, автентифікацію користувачів та шифровані з’єднання, що

забезпечує базовий рівень захисту для багатокористувацьких середовищ. Однією з важливих переваг MySQL є підтримка різних моделей реплікації, які дають змогу підвищувати продуктивність читання, організовувати резервування та забезпечувати відмовостійкість під час зростання навантаження. Завдяки цим можливостям MySQL часто розглядають як стабільне та зріле рішення, яке підходить для проєктів, що потребують розширення інфраструктури та надійної роботи з даними [25].

Окрім кількох корпоративних функцій, іншою основною відмінністю MySQL є підтримка можливостей роботи з кількома користувачами. Це, разом із корпоративними функціями та масштабованістю, робить її ідеальним кандидатом для розподілених застосунків. Система вирізняється простотою встановлення та адміністрування, а також широкою підтримкою з боку спільноти, що робить її зручним вибором для розробників різного рівня. Окрім цього, MySQL активно розвивається, зокрема додаючи підтримку гнучкіших моделей роботи з даними як noSQL, що підвищує її універсальність у сучасних вебзастосунках [25].

Однак, оскільки MySQL переміщує частину службових даних у спеціальні внутрішні сегменти відкату, у неї спостерігається зниження продуктивності під час великих обсягів вставок. Крім того, MySQL може працювати менш ефективно за умов довготривалих операцій читання, натомість демонструючи кращі результати для коротших запитів, особливо тих, що виконуються за індексованими полями. До додаткових обмежень належать відсутність вбудованого повнотекстового пошуку у деяких конфігураціях та потенційне зниження швидкодії за інтенсивного паралельного читання і запису [25].

Остання у нашому списку – PostgreSQL – це об’єктно-реляційна СКБД з відкритим кодом, яка робить особливий акцент на розширюваності та відповідності стандартам. Як і MySQL, PostgreSQL використовує модель клієнт-сервер, а серверний процес, що обробляє комунікацію з клієнтами, керує файлами бази даних та операціями, відомий як процес postgres [25].

PostgreSQL обробляє паралельні клієнтські сесії, відгалужуючи новий процес для кожного з’єднання. Цей процес відокремлений від головного процесу postgres і створюється та знищується протягом часу життя клієнтського з’єднання. На відміну від MySQL, PostgreSQL підтримує матеріалізовані представлення (кешовані представлення), що забезпечує швидший доступ до великих і активних таблиць [25].

PostgreSQL пропонує розвинені механізми безпеки та реплікації, що робить її придатною для застосунків, орієнтованих на високу надійність і цілісність даних. Система підтримує синхронну реплікацію, яка дає змогу забезпечувати узгодженість даних між основним сервером і його репліками. Для контролю доступу доступні гнучкі засоби керування користувачами та політики автентифікації, а комунікація між клієнтом і сервером може бути зашифрована через SSL. Крім того, PostgreSQL забезпечує повну відповідність вимогам ACID, що гарантує передбачувану поведінку транзакцій та високу стійкість до збоїв [25].

PostgreSQL використовує технологію, відому як Multiversion Concurrency Control (MVCC), для підтримки узгодженості даних під час паралельного доступу. Ця технологія перевершує просте використання блокувань для паралелізму, оскільки мінімізує конфлікти блокувань у багатокористувацьких середовищах, значно покращуючи продуктивність. Для зворотної сумісності або застосунків, які бажають класичну технологію блокувань, PostgreSQL також дозволяє технології блокування таблиць і рядків [25].

Поточна реалізація механізму зберігання даних у певних сценаріях залишається не цілком оптимальною та потенційно потребує удосконалень. PostgreSQL також характеризується вищим споживанням ресурсів. Оскільки система створює окремий процес для кожного клієнтського з’єднання, одне з’єднання може займати значний обсяг пам’яті (прибл. до 10 МБ). За великої кількості паралельних підключень це може призвести до суттєвого

навантаження, особливо у порівнянні з легшою потоковою моделлю MySQL [25].

Ще одним помітним недоліком є поведінка PostgreSQL під час частих операцій оновлення даних. Через відсутність кластеризованих індексів такі оновлення можуть відчутно впливати на продуктивність, що робить її менш ефективним варіантом у деяких сценаріях з інтенсивними UPDATE-операціями [25].

Виділимо основні особливості кожної СКБД та зведемо їх у табл. 2.2 [25].

Таблиця 2.2 – Порівняння СКБД [25]

Особливість	SQLite	MySQL	PostgreSQL
Архітектура	Файлова	Клієнт-сервер	Клієнт-сервер
Реплікація	Відсутня	Лідер-репліка, лідер-лідер	Лідер-репліка
Хмарні сервіси	Відсутні	Amazon, Microsoft, Google	Amazon, Microsoft, Google
Паралелізм	Обмежений	Підтримує багато з’єднань	Використовує окремі процеси з’єднань
Масштабованість	Невеликі застосунки	Підтримує масштабування на рівні сервера	Підходить для великих навантажень
Використання	Вебсайти з низьким трафіком, тестування, розробка	Вебсайти, вебзастосунки, онлайн-транзакції	Аналітика, добування даних

У підсумку, не зважаючи на те, що SQLite має обмеження щодо масштабованості та одночасної роботи багатьох користувачів, вона є найоптимальнішим вибором для нашого проєкту з огляду на очікувані низькі вимоги до навантаження. Простота використання, відсутність потреби у налаштуванні окремого серверного процесу та компактність бази даних роблять її зручною та швидкою у впровадженні.

2.2.4 Вибір середовища розробки

Останнім етапом у виборі інструментів для нашого проєкту стане визначення середовища розробки. Від правильного вибору залежить ефективність налагодження, тестування та подальшої підтримки застосунку. Середовище розробки має відповідати специфіці проєкту, підтримувати потрібні мови програмування, інтегруватися з використовуваними технологіями, системами контролю версій і забезпечувати комфортну роботу розробника.

Visual Studio Code (VS Code) — це редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. Він здобув значну популярність завдяки своїй безкоштовності, універсальності та зручності у використанні. Програмне забезпечення сумісне з операційними системами Windows, macOS та Linux, що дозволяє встановлювати його на різні комп’ютерні платформи [26].

Крім того, VS Code підтримує роботу через серверне середовище: користувачі можуть підключатися до віддаленого сервера за допомогою SSH та редагувати файли безпосередньо через локальний інтерфейс редактора [26].

Існує також онлайн-версія VS Code, яка дозволяє працювати з локальними файлами. Хоча вона має менший функціонал порівняно з десктопною версією, ця веб-версія залишається достатньо зручною та ефективною для виконання багатьох завдань [26].

Visual Studio Code оснащений вбудованим терміналом, який дозволяє виконувати командні оболонки безпосередньо під час розробки коду. Це особливо корисно для встановлення пакетів через pip або запуску Python-застосунків із командного рядка. Такий термінал значно спрощує робочий процес і підвищує зручність розробки [26].

За наявності відповідних розширень Visual Studio Code можна налаштувати для доступу до файлів на іншому комп’ютері, до якого ви маєте

доступ. Це може бути віддалений сервер або підсистема Windows для Linux (WSL) [26].

Visual Studio Code оснащений функцією IntelliSense, яка значно полегшує написання коду, пропонуючи автодоповнення, підказки та контекстні рекомендації під час програмування [26].

Завдяки відкритій архітектурі VS Code підтримує численні розширення, що дозволяють налаштувати середовище під будь-які потреби, від простого редагування коду до складної розробки із застосуванням відлагодження, тестування і контролю версій [26].

Крім того, Visual Studio Code є повністю безкоштовним, що робить його доступним інструментом як для початківців, так і для професіоналів, які прагнуть гнучкості та широких можливостей налаштування [26].

Однак, варто зауважити, що VS Code за замовчуванням встановлюється лише як текстовий редактор без вбудованого інтерпретатора Python чи інших мов програмування. Тому для повноцінної роботи з Python необхідно окремо встановити інтерпретатор, а також додаткові корисні інструменти, наприклад, менеджер віртуальних середовищ, який допоможе ізолювати залежності проєкту [26].

PyCharm – це інтегроване середовище розробки для програмування на Python, створене компанією JetBrains, яка також відома розробкою популярних IDE, таких як IntelliJ для Java та WebStorm для JavaScript. PyCharm пропонується у двох версіях: Community Edition, яка є безкоштовною та з відкритим кодом, та Professional Edition – платній версії з розширеним функціоналом. У Community Edition користувачі можуть створювати файли Python і HTML, тоді як Professional Edition підтримує додаткову роботу з файлами HTML, JavaScript і SQL, що робить її більш універсальною для складніших веб- та серверних проєктів. Обидві версії підтримують встановлення на операційних системах Windows, MacOS та Linux, що забезпечує широке охоплення користувачів [26].

PyCharm дуже простий у використанні, особливо для початківців. Процес встановлення не викликає складнощів, а створення нового проєкту та налаштування віртуального середовища відбувається інтуїтивно й швидко. Так само легко створювати нові файли та додавати їх до проєкту, що значно полегшує організацію коду [26].

Для додавання пакетів можна скористатися панеллю керування пакетами, де їх легко знайти та встановити без потреби працювати через командний рядок. Запуск програми здійснюється натисненням кнопки Run, що робить процес розробки максимально зручним і не лякає тих, хто тільки починає знайомитися з Python. Завдяки такому інтерфейсу PyCharm дозволяє швидко почати писати код, не заглиблюючись у складні технічні деталі терміналу [26].

При додаванні точок зупину у код, середовище розробки PyCharm забезпечує відображення стану змінних у визначених ділянках програми. Це суттєво спрощує процес верифікації коректності значень змінних у ході виконання коду, що, у підсумку, підвищує ефективність налагодження та загальну продуктивність розробника [26].

PyCharm оснащений інструментами автоматизації, які полегшують проведення рефакторингу коду з мінімальною кількістю дій. Серед доступних функцій – переміщення функцій між файлами, виділення окремих фрагментів коду у функції, а також автоматичне генерування документації у вигляді docstring [26].

Такі можливості сприяють покращенню структурованості та читабельності коду, а також знижують ризик помилок, пов’язаних з неповним оновленням інших частин проєкту. Зокрема, при переміщенні функції в інший модуль PyCharm автоматично оновлює всі імпорти, пов’язані з цією функцією, що гарантує цілісність коду [26].

Встановлення пакетів супроводжується можливістю перегляду відповідної документації безпосередньо в інтегрованому середовищі розробки. Такий підхід забезпечує швидкий доступ до необхідної інформації

без потреби переходу до зовнішніх ресурсів, що сприяє підвищенню зручності роботи [26].

Інтерфейс PyCharm дозволяє отримувати документацію для всіх пакетів, наявних у панелі керування пакетами, включно з публічно доступними бібліотеками, що значно спрощує процес вивчення та використання сторонніх інструментів [26].

З огляду на те, що використання системи контролю версій Git через командний рядок може бути складним і викликати труднощі, PyCharm надає інтегровану панель для керування версіями. Це значно полегшує виконання комітів та керування репозиторіями [26].

Коміт змін у PyCharm реалізовано у вигляді вибору файлів за допомогою чекбоксів та введення повідомлення коміту, після чого користувач може здійснити локальний коміт або відразу відправити зміни до віддаленого репозиторію, що забезпечує оперативність і зручність управління версіями коду [26].

Виділимо основні особливості середовищ розробки та зведемо їх у табл. 2.3 [26].

Таблиця 2.3 – Порівняння середовищ розробки

Особливість	Visual Studio Code	PyCharm
Налаштування	Необхідно встановлювати плагіни, щоб отримати підтримку Python. Самостійне встановлення інтерпретатора	Все працює «з коробки», зручний менеджер інтерпретаторів
Розширення	Величезна екосистема	Менша екосистема, але більший вбудований функціонал
Доступність	Повністю відкрите та безкоштовне	Має платну версію, без якої неможлива робота з деяким функціоналом
Редагування коду	Потужний IntelliSense, але потребує підтримки Python розширенням	Є вбудоване виправлення помилок та форматування
Пакети	Ручне встановлення	Пакетний менеджер
Git	Вбудовано в інтерфейс	Вбудовано в інтерфейс

Підсумовуючи проведене порівняння, слід відзначити, що PyCharm є оптимальним середовищем розробки для нашого вебзастосунку. Його вбудована підтримка Python на всіх етапах розробки, від написання коду й налагодження до керування пакетами та контролю версій, забезпечує високу продуктивність і комфорт роботи. Багатий вбудований функціонал, зокрема, підключення інтерпретатора та робота з пакунками, створюють ефективну синергію, що значно спрощує робочі процеси та мінімізує ймовірність помилок.

2.3 Архітектура системи та модель даних

2.3.1 Загальна структурна схема

Перейдемо до проєктування архітектури системи нашого проєкту. Першим кроком буде створення загальної структурної схеми, яка відображатиме основні компоненти системи та їх взаємодії на логічному рівні. Така схема дозволить наочно побачити, як різні модулі співпрацюють між собою для забезпечення функціональності застосунку, що стане основою для подальшої деталізації архітектури.

Користувач взаємодіє із системою через вебпереглядач, надсилаючи запити до клієнтської частини застосунку. Запити включають в себе створення або авторизацію в акаунт, надсилання вхідних даних для прогнозування або перегляд історії попередніх прогнозів.

Клієнт відправляє запити на сервер, отримує результат запиту і повертає користувачу HTML-сторінку разом зі статичними ресурсами типу оформлення цієї сторінки, що візуалізує цю відповідь. Клієнт відповідає за формування інтерфейсу користувача та відображення результатів обробки.

Сервер – це центральний компонент системи, що діє як координатор між іншими компонентами. Він приймає запити від клієнтської частини і повертає або створює записи у базі даних; звертається до моделей

машинного навчання, передає їм вхідні дані та отримує прогноз, який записує до бази даних та повертає клієнтській частині.

База даних зберігає усі дії користувачів у системі, усі надані ними дані для авторизації та прогнозів, інформації про використовувані моделі машинного навчання та історію всіх прогнозів.

Модуль машинного навчання відповідає за завантаження вже натренованих моделей і повернення результатів обробки параметрів від сервера.

Модуль тренування моделей машинного навчання діє окремо від сервера, в його роботу входить тренування моделей на навчальній вибірці і запис їх у модуль машинного навчання для подальшої роботи із запитами сервера.

Навчальна вибірка зберігає відформатовані дані, які модуль тренування моделей машинного навчання використовує для тренування.

Результат проєктування загальної структурної схеми зображено на рис. 2.1.

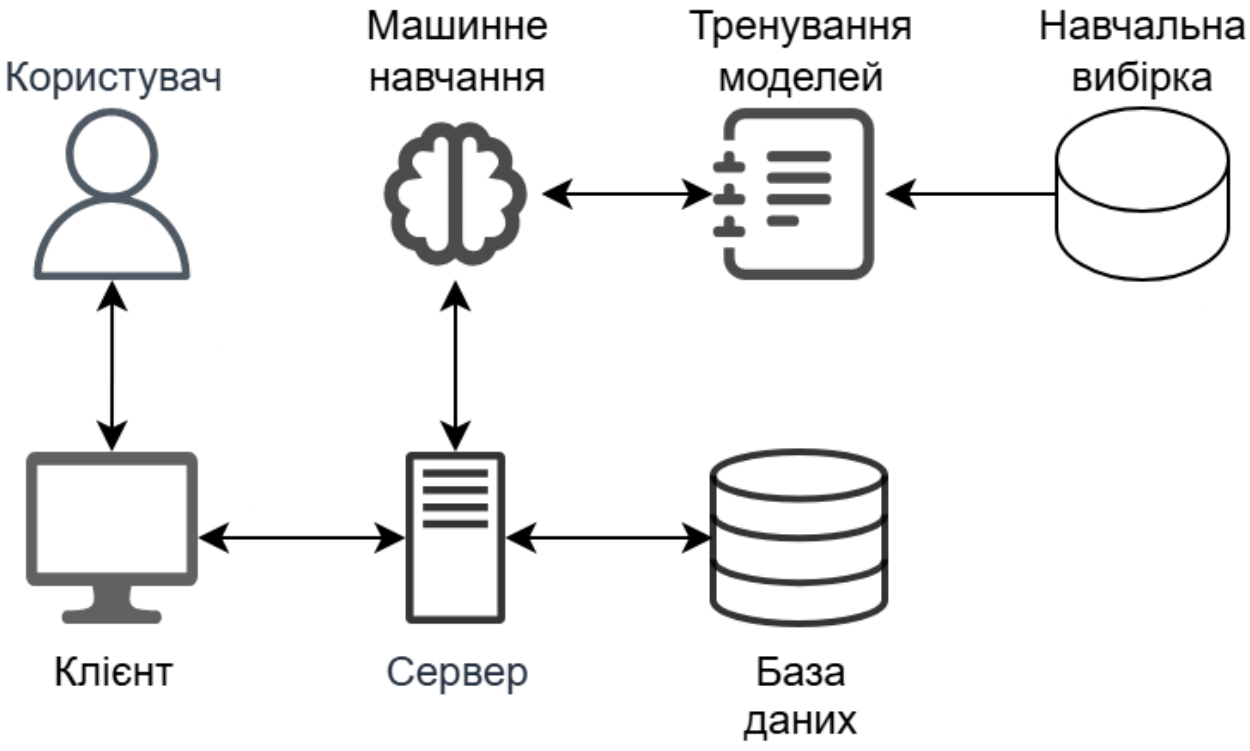


Рисунок 2.1 – Загальна структурна схема

2.3.2 Діаграма компонентів

Спробуємо розбити структуру нашої системи на компоненти та визначити зв’язки та взаємодію між ними.

Flask Application є центральним компонентом, який керує життєвим циклом вебзастосунку, ініціалізацією сервера, підключенням до бази даних та організацією обробки HTTP-запитів. Він поєднується структурними зв’язками з інтерфейсами інших компонентів, а саме форм, інтерфейсу користувача, автентифікації, маршрутизації та допоміжних функцій.

Authentication Component відповідає за реалізацію логіки автентифікації та авторизації користувачів у системі. Він забезпечує перевірку облікових даних, таких як ім’я користувача та пароль, а також керує створенням, підтримкою та завершенням користувацьких сесій. Поєднаний лише з інтерфейсом центрального компонента.

Utility Component включає набір допоміжних функцій і сервісів, які використовуються різними частинами застосунку для виконання загальних завдань. До цього компоненту входять, зокрема, алгоритми гешування паролів для забезпечення безпеки зберігання облікових даних, функції валідації та обробки даних, а також інші утиліти, що спрощують роботу з застосунком. Поєднаний лише з інтерфейсом центрального компонента.

User Interface Component відповідає за візуалізацію інформації та забезпечення зручної взаємодії користувача з вебзастосунком. Він реалізований через HTML-шаблони, які формують структуру та оформлення сторінок, а також обробляють отримане від користувача введення. Цей компонент виступає посередником між серверною логікою та користувачем. Поєднаний з центральним та формовим компонентами через зв’язок асоціацій.

Forms Component відповідає за створення, валідацію та обробку форм, які використовуються користувачами для взаємодії із системою. Цей компонент забезпечує правильність і цілісність введених даних шляхом

застосування різноманітних правил валідації, таких як перевірка обов’язкових полів, форматів введення та логічних умов. Основні форми включають реєстрацію, вхід до системи та введення параметрів для прогнозування. Поєднаний з інтерфейсами центрального компонента та маршрутизації, також через зв’язок асоціації з інтерфейсом користувача.

Routing Component відповідає за визначення та обробку маршрутів вебзастосунку, які приймають HTTP-запити від користувачів. Він виконує роль посередника між клієнтом і серверною логікою, забезпечуючи коректне спрямування запитів до відповідних функцій або сервісів. Цей компонент також координує взаємодію інших модулів системи, запускаючи бізнес-логіку, викликаючи моделі даних або обробляючи введені користувачем дані. Поєднаний з інтерфейсами головного компонента, форм, машинного навчання та рівня доступу даних.

Data Access Layer Component відповідає за безпосередню взаємодію з базою даних, виконуючи операції CRUD даних через визначені моделі. Цей компонент ізолює роботу з даними від бізнес-логіки, що реалізується у маршрутах. Завдяки цьому шарові відбувається централізоване управління доступом до інформації, оптимізація запитів і забезпечення цілісності даних. Поєднаний асоціацією з артефактом бази даних та інтерфейсами компонентів маршрутизації та моделей даних.

Data Models Component включає класи об’єктно-реляційного відображення, які безпосередньо відображають структуру таблиць у базі даних, таких як Users, Predictions та ін. Ці класи визначають поля, типи даних, зв’язки між сутностями та правила валідації, що дозволяє ефективно моделювати інформацію в програмному коді. Поєднаний з інтерфейсом компоненту рівня доступу даних.

Machine Learning Component відповідає за завантаження попередньо навченої моделі машинного навчання, збереженої у вигляді файлу, та її інтеграцію у застосунок. Цей компонент приймає вхідні параметри користувача, обробляє їх відповідно до вимог моделі та виконує

прогнозування, повертаючи результати для подальшої обробки або відображення. Поєднаний з інтерфейсом компоненту маршрутизації та артефактами натренованих збережених для роботи моделей.

ML Training Component відповідає за процес навчання моделей машинного навчання з використанням наявних навчальних даних. Цей компонент включає алгоритми, які аналізують дані, навчають модель, а також оновлюють її при необхідності. Після завершення тренування модель зберігається у вигляді файлу для подальшого використання. Поєднаний лише з інтерфейсами артефактів збережених моделей та навчальної вибірки.

Спроєктовану діаграму компонентів зображено на рис. 2.2.

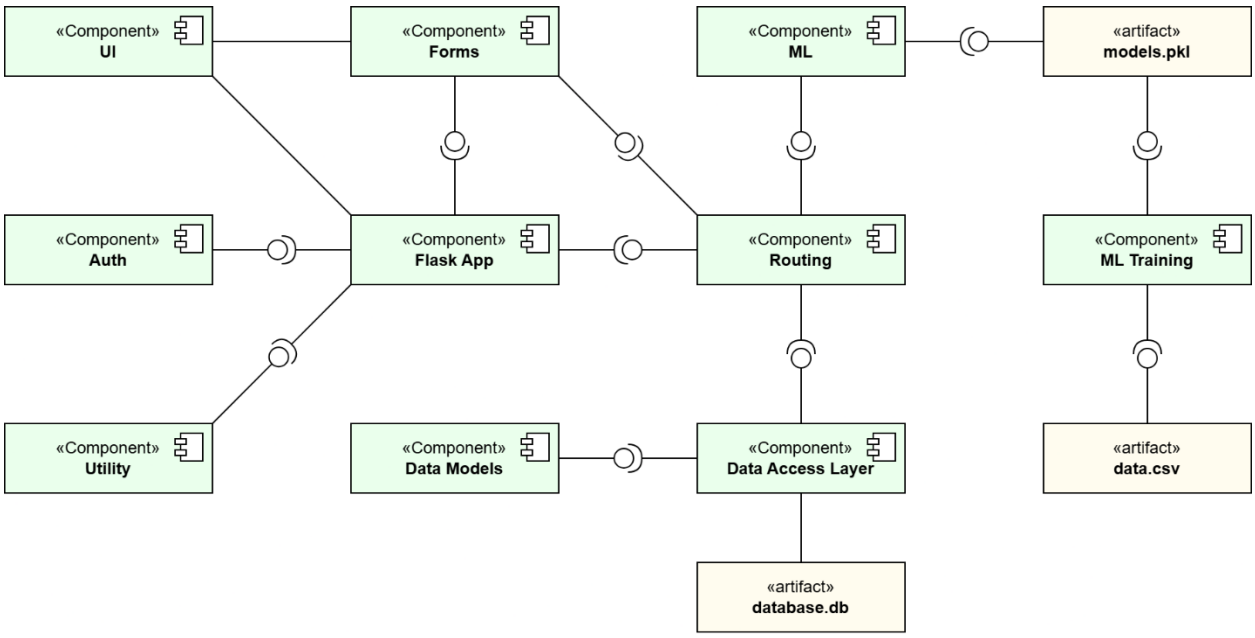


Рисунок 2.2 – Діаграма компонентів

2.3.3 Загальна діаграма сутностей

Для представлення зберігання даних спроєктуємо діаграму сутностей, в якій відобразимо структуру таблиць бази даних.

Таблиця Users містить інформацію про користувачів системи, включно з унікальними ідентифікаторами, обліковими даними та іншими необхідними атрибутами для автентифікації та ідентифікації. Пов’язана з

таблицею прогнозів, яка зберігає результати прогнозів, виконаних для кожного користувача.

Таблиця Predictions відповідає за збереження інформації про конкретні прогнози, зроблені системою на основі машинного навчання. Кожен запис у цій таблиці пов’язаний з конкретним користувачем таблиці користувачів, записом у таблиці вхідних даних, а також з відповідною моделлю машинного навчання з таблиці моделей.

Таблиця Prediction Inputs відповідає за збереження введених користувачем вхідних даних до конкретного прогнозу. Відповідно, записи цієї таблиці пов’язані з таблицею прогнозів.

Таблиця Models зберігає інформацію про навчальні моделі машинного навчання, які використовуються для прогнозування. Кожна модель має свої метадані, а також пов’язана з таблицею прогнозів для визначення, яка саме модель була застосована при створенні конкретного прогнозу.

Таблиця Logs зберігає історію всіх дій користувачів у застосунку з їх часом, це стосується створення акаунту, входів та виходів, надісланих запитів. Кожен запис пов’язаний із користувачем, до якого він відноситься.

Спроєктовану діаграму сутностей зображено на рис. 2.3.

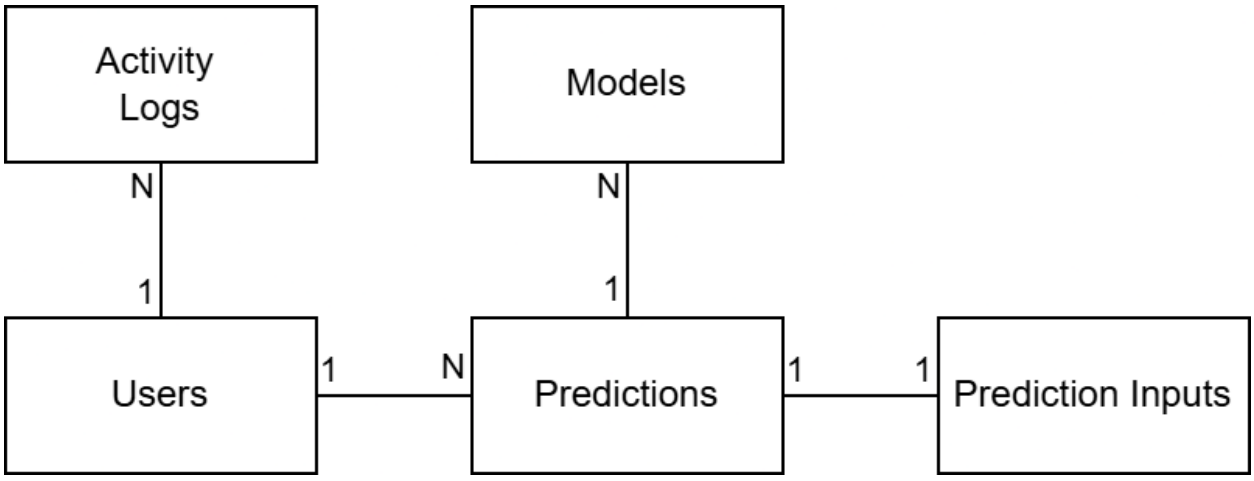


Рисунок 2.3 – Загальна діаграма сутностей

2.4 Висновки до розділу 2

У цьому розділі було визначено ключові вимоги до програмної системи, обґрунтовано вибір інструментів розробки та сформовано основу для подальшого проєктування архітектури.

Аналіз функціональних вимог дав змогу окреслити головні сценарії роботи застосунку. Нефункціональні вимоги забезпечили розуміння того, якими мають бути показники безпеки, продуктивності, стабільності та зручності застосунку для кінцевого користувача. Було визначено оптимальні технічні вимоги для подальшого можливого допрацювання застосунка.

Порівняння мов програмування засвідчило, що Python найкраще відповідає потребам проєкту завдяки природній інтеграції з інструментами аналізу даних та машинного навчання. Оцінка можливостей вебфреймворків показала, що Flask є оптимальним вибором для швидкого створення серверної частини з мінімальним обмеженням в плані вибору інструментів. Аналіз СКБД підтвердив доцільність використання SQLite, оскільки очікуване навантаження на систему не потребує складної клієнт–серверної інфраструктури.

PyCharm було обрано, як основне середовище розробки – він забезпечує зручність, високу інтеграцію з Python та ефективні інструменти налагодження, що сприяє швидкому й упорядкованому процесу створення застосунку.

Сукупність ухвалених рішень формує цілісну технологічну базу для реалізації програмної системи та забезпечує передумови для подальшого проєктування архітектури, її оптимізації та масштабування.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Модуль тренування моделей машинного навчання

У цьому модулі реалізовано тренування моделей машинного навчання як окремої підсистеми проєкту. Модуль включає набір спеціалізованих скриптів, кожен з яких відповідає за підготовку конкретного алгоритму машинного навчання, налаштування його гіперпараметрів та збереження отриманої моделі у файлову структуру системи. Після тренування всі ключові характеристики моделі, такі як тип, точність, час навчання та використані параметри, фіксуються у базі даних. Це створює єдину структуровану основу для подальшого використання навчальних моделей у процесі прогнозування у вебзастосунку.

Серед реалізованих методів машинного навчання у модулі представлено моделі Support Vector Machine, Naïve Bayes, Random Forest, XGBoost, логістичну регресію, а також власну зважену ансамблеву модель, що використовує результати всіх попередніх алгоритмів. Далі детально розглянуто процес їх тренування, налаштування гіперпараметрів та інтеграцію у структуру системи.

Для побудови усіх моделей на першому етапі завантажуються вхідні дані у форматі CSV, після чого вони розділяються на матрицю ознак та цільову змінну. Розмір тестової вибірки при розбитті складає 20% від усіх даних.

У процесі тренування моделі SVM використовується конвеєр, що поєднує масштабування ознак за допомогою StandardScaler та класифікатор SVC. Масштабування включено тому, що SVM коректно працює лише тоді, коли всі ознаки мають порівняний діапазон значень.

Щоб підібрати найкращі налаштування моделі, проводиться автоматичний перебір параметрів:

– C – наскільки сильно модель штрафує себе за помилки;

– gamma – ступінь впливу окремих точок даних на формування роздільної межі між класами;

– радіально-базисне ядро (rbf), що дозволяє моделі враховувати нелінійні залежності.

Під час підбору параметрів дані автоматично діляться на кілька частин, і модель тренується багато разів на різних комбінаціях цих частин. Це робиться для того, щоб знайти налаштування, які добре працюють не лише на одному наборі даних, а загалом.

Наступною моделлю є Naïve Bayes, для якої у скрипті реалізовано тренування класифікатора GaussianNB, що є однією з найпростіших та найшвидших моделей машинного навчання. На відміну від інших методів, вона не потребує жодного підбору гіперпараметрів, тому модель створюється у стандартній конфігурації без додаткових налаштувань. GaussianNB ґрунтується на припущенні про нормальний розподіл ознак та їхню статистичну незалежність, що суттєво спрощує обчислення і забезпечує дуже швидке навчання навіть на повних вибірках.

Після тренування модель одразу перевіряється на тестовому наборі, а її конфігурація фіксується у мінімальному вигляді в полі params, оскільки GaussianNB не містить параметрів, які потребували б оптимізації чи детального документування.

Наступною реалізованою моделлю є Random Forest, яка належить до ансамблевих алгоритмів та формує прогноз на основі колективного рішення багатьох дерев рішень. У скрипті модель ініціалізується з параметром балансування класів. На відміну від NB, цей метод має значно більше налаштувань, тому для отримання оптимальної конфігурації використовується перебір гіперпараметрів. У рамках GridSearchCV здійснюється оптимізація чотирьох параметрів:

– n_estimators – кількість дерев у лісі, що впливає на стабільність та точність моделі;

– max_depth – максимально допустима глибина дерева, яка контролює рівень узагальнення;

– min_samples_split – мінімальна кількість прикладів, необхідних для поділу вузла, що запобігає надмірній складності;

– min_samples_leaf – мінімальна кількість прикладів у листі, яка зменшує ризик перенавчання.

GridSearchCV перебирає комбінації цих параметрів з п’ятикратною перехресною перевіркою та визначає найкращу конфігурацію, оптимальну модель.

Наступною у модулі тренування є модель XGBoost, яка реалізує градієнтний бустинг над деревами рішень і зазвичай демонструє високу якість на структурованих табличних даних. Було вирішено додати цю модель як альтернативний підхід до роботи з деревами рішень, оскільки вона реалізує інший механізм побудови ансамблю порівняно з RF.

На початку виконується перетворення цільової змінної на числовий формат, оскільки XGBoost працює лише з числовими мітками класів. Модель ініціалізується у режимі для двокласових задач, а також має параметр, що прискорює навчання на великих наборах даних, використовуючи побудову гістограм замість повного перебору порогів.

Для оптимізації продуктивності моделі застосовується GridSearchCV з підбором кількох ключових гіперпараметрів. Зокрема:

– n_estimators та max_depth, що є подібними до параметрів RF;

– learning_rate – регулює швидкість навчання ансамблю, зменшуючи або збільшуючи внесок кожного дерева;

– subsample – частка випадково вибраних прикладів для тренування кожного дерева, що допомагає боротися з перенавчанням;

– colsample_bytree – частка вибраних ознак для кожного дерева, що також сприяє кращій здатності узагальнювати.

Наступною моделлю є логістична регресія, що є одним із найпоширеніших лінійних класифікаторів, що оцінює ймовірність

належності об’єкта до певного класу. Перед тренуванням вихідні мітки перетворюються у числовий формат, оскільки алгоритм працює лише з числовими значеннями цільової змінної.

Для цієї моделі також застосовано автоматичний підбір параметрів з використанням GridSearchCV. Під час підбору перевіряються такі налаштування:

– параметр штрафування C, що вже згадувався у SVM;

– тип штрафу l2 для стабільності тренування;

– solver, що визначає один з двох алгоритмів оптимізації, що придатні для логістичної регресії з бінарною цільовою ознакою.

В усіх скриптах після підбору оптимальних параметрів найкраща знайдена модель додатково перевіряється на окремому тестовому наборі, що дає підсумкову точність. Потім модель зберігається у файл .pkl, щоб її можна було використовувати вже під час роботи вебзастосунку.

Наприкінці усі скрипти оновлюють або створюють запис про модель у таблиці моделей – зберігаються шлях до файлу, точність, час тренування та знайдені налаштування. Якщо запис для моделі уже існує, він оновлюється, щоб у системі завжди залишалася найсвіжіша версія моделі.

Останньою реалізованою моделлю є власна реалізація зваженого ансамблю, який об’єднує результати попередньо натренованих класифікаторів. На відміну від інших методів, ансамбль не тренується «з нуля», а завантажує всі вже збережені моделі з бази даних та використовує їхні прогнозні ймовірності для формування спільного результату. Такий підхід дозволяє поєднати різні алгоритми та їхні сильні сторони, що часто дає стабільніші результати порівняно з окремими моделями.

Після отримання списку моделей система завантажує кожну з них з файлу та обчислює їхню вагу. Ваги визначаються пропорційно до точності, піднесеної до вказаної степені, що дозволяє сильнішим моделям суттєвіше впливати на фінальний прогноз, тоді як менш точні моделі мають значно

менший внесок. Такий спосіб агрегування відомий як зважене голосування, коли результати кожної моделі комбінуються залежно від її довіри.

Для кожного класифікатора обчислюються векторні оцінки ймовірностей predict_proba, після чого вони підсумовуються з урахуванням ваг. Остаточний прогноз визначається як клас із максимальною сумарною ймовірністю.

На завершальному етапі ансамбль, як і усі попередні моделі, оцінюється на тестових даних, після чого зберігаються його ваги, список використаних моделей та шлях до створеного файлу. У таблиці моделей оновлюється або створюється запис, який містить інформацію про точність ансамблю, тривалість обчислень та застосовані вагові коефіцієнти.

Такий підхід забезпечує більш універсальне рішення, здатне узагальнювати сильні сторони різних моделей та компенсувати їхні окремі недоліки, що робить ансамбль важливим завершальним елементом модуля тренування.

3.2 Модуль автентифікації

Механізм автентифікації у вебзастосунку реалізовано за допомогою фреймворку Flask у поєднанні з компонентом Flask-Login. Цей модуль відповідає за створення, перевірку та підтримання користувацьких сесій, забезпечуючи доступ до захищених сторінок і контроль за ідентифікацією користувача протягом усього часу роботи із системою. Реалізація побудована за модульним принципом із використанням окремого Blueprint для обробки автентифікаційних маршрутів, що дає змогу легко розширювати функціональність у майбутньому, оскільки кожен модуль містить лише логіку власної підсистеми..

Основою модуля є конфігурація LoginManager, який ініціалізується під час запуску застосунку та інтегрується з моделлю користувачів. Через функцію user_loader система отримує доступ до облікових записів у базі

даних і виконує автоматичне відновлення стану користувача під час кожного запиту. Важливою частиною конфігурації є налаштування адреси, на яку перенаправляються неавторизовані користувачі у разі спроби перейти на захищені сторінки. Таким чином забезпечується єдиний вхід до системи та контрольований доступ до внутрішніх функцій вебзастосунку.

Механізм реєстрації реалізований як окремий маршрут register, що обробляє дані, надіслані користувачем через HTML-форму. Перед створенням облікового запису система виконує серверну валідацію за допомогою методів validate_username та validate_password: перевіряє формат і довжину імені користувача, наявність необхідних символів у паролі та відсутність заборонених конструкцій. Така перевірка дає змогу запобігти створенню некоректних або потенційно небезпечних облікових записів. Після успішної валідації пароль гешується за допомогою алгоритмів з бібліотеки Werkzeug, і лише після цього зберігається у базі даних. Зберігання паролів у вигляді криптографічних гешів є фундаментальним рішенням для забезпечення безпеки користувацьких даних. Такий підхід мінімізує ризики у разі компрометації бази даних, виключає можливість прямого відновлення паролів і відповідає сучасним стандартам інформаційної безпеки. Подія реєстрації фіксується у таблиці activity_logs, що дозволяє сформувати історію взаємодії користувачів із системою.

Авторизація здійснюється на основі тих самих принципів, але на кінцевій точці login: введені дані перевіряються на коректність, пароль зіставляється з гешем у базі, а у разі успішної автентифікації створюється сесія користувача. Одночасно до журналу активності заноситься запис про вхід. Застосунок після логіну автоматично перенаправляє користувача на головну сторінку, доступну лише після автентифікації. Усі маршрути, що потребують підтвердження особи, захищені декоратором @login_required, який гарантує, що неавторизовані користувачі не мають можливості переглядати або виконувати дії, призначені виключно для зареєстрованих осіб.

Процедура виходу з облікового запису відбувається на маршруті logout і також супроводжується записом у журналі активності. Після завершення сесії вебзастосунок видаляє інформацію про авторизованого користувача і повертає його на сторінку входу.

Усі відповідні HTML-шаблони, що надсилаються користувачу при запитах до сервера, створені з використанням базових можливостей Flask щодо передачі змінних у шаблон, системи сповіщень flash та динамічної маршрутизації.

3.3 Модуль прогнозування та історії прогнозів

Модуль прогнозування є центральним елементом функціональності вебзастосунку, оскільки забезпечує взаємодію користувача з натренованими моделями машинного навчання та формує кінцевий результат, а саме оцінку ризику захворювання на основі наданих медичних показників. Архітектура модуля передбачає роботу з кількома компонентами системи: базою даних, файловими моделями у форматі .pkl, механізмом автентифікації та спеціальними сервісами для журналювання активності користувачів.

Після переходу на кінцеву точку створення прогнозу (predict) користувачу відображається форма введення медичних параметрів та список доступних моделей, що завантажуються із таблиці models. Отримані від користувача значення проходять багаторівневу серверну валідацію, а саме перевіряється наявність усіх обов’язкових полів, коректність числових перетворень, відсутність хибних або від’ємних значень, а для параметра статі додатково контролюється відповідність встановленій бінарній логіці (0 – жінка, 1 – чоловік). Така перевірка мінімізує ризик введення некоректних або потенційно шкідливих даних, що могло б призвести до викривлення результатів моделі. Використання перевірок на стороні сервера унеможливлює представлення некоректних або навмисно змінених параметрів, що може статися при обході клієнтської валідації.

Після успішної перевірки даних система завантажує обрану модель машинного навчання, використовуючи метод load_model. Якщо модель є однією з базових, то об’єкт моделі відновлюється з .pkl-файла та використовується для отримання ймовірності позитивного результату. Для зваженого ансамблю застосовується окрема логіка: з файлу завантажується структура, що містить список моделей та їхні індивідуальні ваги. Кожна модель створює власний прогноз, після чого результати агрегуються шляхом вагового підсумовування ймовірностей. Застосований механізм дозволяє підвищувати стійкість прогнозу в ситуаціях, коли окремі моделі можуть демонструвати нижчу стабільність на різних типах даних.

Після отримання прогнозу система зберігає інформацію про введені користувачем дані у таблицю prediction_inputs у форматі JSON. Це забезпечує збереження повного контексту прогнозу та можливість подальшого перегляду або аналізу. Окрім цього, це надає користувачу можливість провести додаткові прогнози з цими ж даними, але з використанням іншої моделі. Далі створюється запис у таблиці predictions, у якому фіксуються ідентифікатор користувача, використана модель, пов’язаний запис із вхідними даними, результат прогнозу та значення ймовірності.

Модуль містить окремий механізм фіксації дій користувачів. Після здійснення прогнозу система створює відповідний запис у таблиці activity_logs, у якому зберігається інформація про виконану операцію. Наявність такого журналу дозволяє контролювати послідовність взаємодій користувача із застосунком, підвищує прозорість його роботи, дає можливість відслідковувати потенційні аномальні дії та забезпечує можливість проведення аудиту при необхідності.

Користувач має можливість переглядати історію всіх попередніх прогнозів через доступ до кінцевої точки history, де записи відсортовані у зворотному хронологічному порядку. Із кожного запису можна перейти до детального перегляду – кінцевої точки prediction/id, у якому відображаються

як вхідні значення, так і отриманий результат. Для забезпечення конфіденційності даних реалізована перевірка належності прогнозу поточному користувачу. Також передбачено функціонал видалення прогнозів разом із пов’язаними вхідними даними. Видалення супроводжується записом у журнал активності.

3.4 Реалізація інтерфейсу користувача

Інтерфейс користувача у вебзастосунку реалізовано на основі шаблонів Jinja2, які поєднують статичне HTML-представлення з динамічними даними, що передаються сервером під час обробки запитів. Такий підхід забезпечує формування інтерфейсу безпосередньо на стороні сервера та гарантує коректність відображення стану системи відповідно до авторизації користувача, контексту прогнозу або доступних моделей машинного навчання.

Усі шаблони побудовані за єдиним стилістичним підходом, а саме використовуються CSS-файли reset для зняття усіх стандартних стилів з елементів та style, що вже формує мінімалістичний дизайн, забезпечує узгодженість зовнішнього вигляду елементів інтерфейсу та читабельність. Окремі стилі кнопок, таблиць, форм та інформаційних блоків створюють чітку візуальну структуру, полегшуючи взаємодію користувача з функціоналом застосунку. Особливе місце займає система флеш-повідомлень, що використовується для повідомлення про результати операцій (реєстрація, логін, помилки валідації, успішне створення прогнозу). Повідомлення автоматично вставляються у шаблони та візуально виділяються залежно від типу події, що сприяє зручності використання.

Інтерфейс включає декілька ключових сторінок. Сторінки login та register реалізують процес автентифікації та використовують обов’язкову серверну валідацію введених полів.

Панель керування dashboard відображає ім’я користувача, надає можливість змінити поточного користувача і забезпечує навігацію до створення нового прогнозу та перегляду історії всіх минулих прогнозів під авторизованим користувачем.

Форма створення прогнозу predict містить структуровані поля для введення клінічних параметрів, радіокнопки для вибору статі та спадаючий список доступних моделей машинного навчання. Окрім зручності введення, форма реалізує відправлення значень у стандартизованому вигляді, що сприяє точності прогнозування.

Сторінка history представлена у вигляді таблиці, яка містить повний список прогнозів користувача, відсортований за датою у спадному порядку. Кожний запис забезпечує миттєвий перехід до сторінки деталізації та можливість видалення прогнозу з підтвердженням дії.

Сторінка prediction_details містить детально структуровану інформацію, а саме результат прогнозу, ймовірність, яку додатково візуалізовано через горизонтальну індикаторну шкалу, доступ за бажанням до використовуваних у цьому прогнозі вхідних даних, що реалізовано за допомогою JS-функції toggleInputData, можливість запустити новий прогноз із цими ж даними, але з використанням іншої моделі, видалення поточного прогнозу з підтвердженням вибору.

3.5 Реалізація зберігання даних

Система даних у розробленому вебзастосунку побудована на основі реляційної бази даних SQLite, інтегрованої у фреймворк Flask за допомогою ORM-бібліотеки SQLAlchemy. Такий підхід забезпечує визначення структури таблиць у вигляді Python-класів, автоматичне створення та оновлення схеми бази, а також прозоре виконання CRUD-операцій без необхідності прямого використання SQL-запитів. Застосування ORM-моделі також сприяє підтриманню структурної цілісності даних та полегшує інтеграцію між

логікою вебзастосунку, модулями машинного навчання та механізмами збереження результатів.

База даних включає низку таблиць, що разом утворюють узгоджену логічну модель роботи системи. Таблиця users містить відомості про зареєстрованих користувачів, зокрема ім’я, геш пароля та час створення облікового запису. На основі цих даних забезпечується автентифікація доступу до застосунку та формування індивідуальної історії дій. З цією метою застосовується таблиця activity_logs, у якій фіксуються всі події, пов’язані з активністю користувача: від авторизації до виконання прогнозів. Такий журнал дозволяє відтворювати послідовність операцій у системі, аналізувати поведінку користувачів і підтримувати загальну прозорість процесів.

Окреме значення має таблиця models, де зберігаються метадані про всі натреновані алгоритми машинного навчання. У ній фіксується тип моделі, шлях до її файлу, точність, структура параметрів та момент тренування. Це забезпечує можливість оперативного вибору відповідної моделі під час прогнозування, а також створення аналітичних оглядів для оцінювання ефективності різних алгоритмів. Тісно пов’язана з процесом прогнозування таблиця prediction_inputs слугує сховищем для введених користувачем медичних даних.

Результати прогнозів характеризуються окремою таблицею predictions, яка містить інформацію про використану модель, відповідні вхідні дані, значення результату та ймовірність класифікації. Кожен запис дає змогу відтворити повний контекст прогнозу, визначити, коли він був виконаний і за якими показниками.

Реалізовану діаграму сутностей бази даних та зв’язків між таблицями зображено на рис. 3.1.

Таким чином, створена база даних формує цілісний і гнучкий фундамент програмного забезпечення. Вона забезпечує безпечне зберігання облікових даних, підтримує механізми журналювання дій користувачів,

інтегрує результати тренування моделей машинного навчання та гарантує відтворюваність прогнозів.

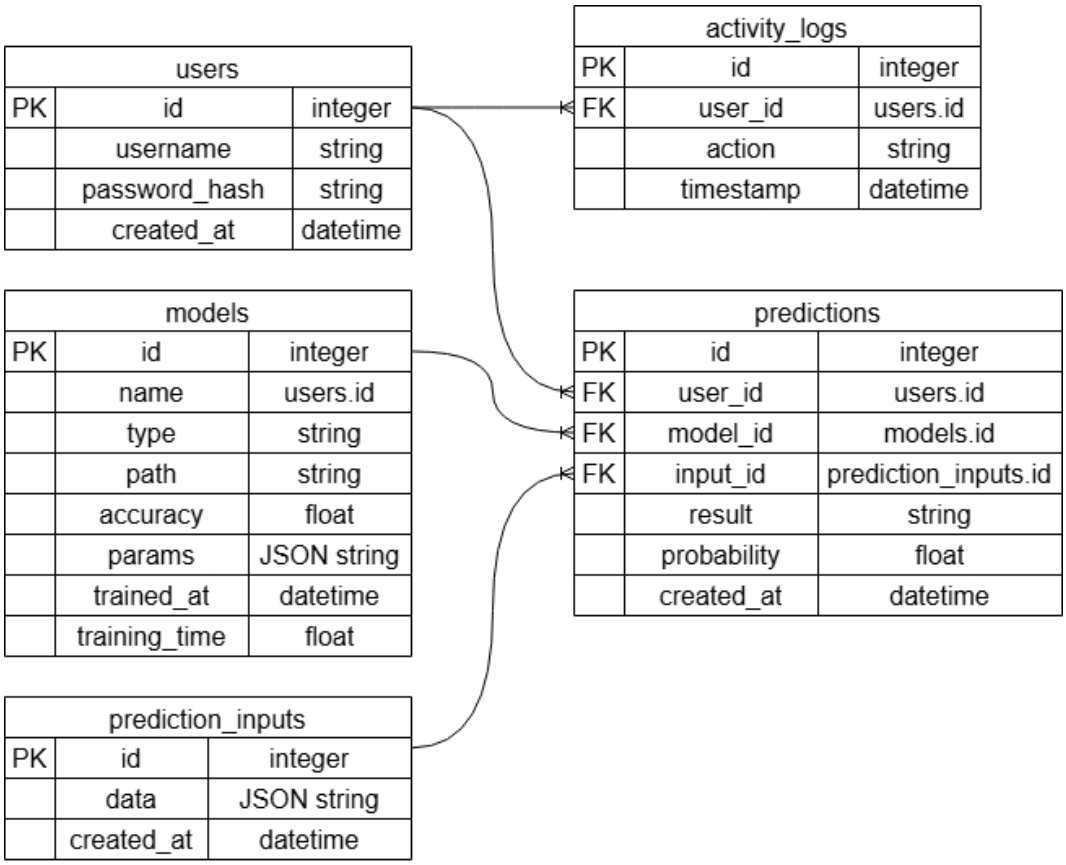


Рисунок 3.1 – Діаграма сутностей та зв’язків бази даних

3.6 Висновки до розділу 3

У цьому розділі було описано процес розробки програмної системи, що поєднує модуль тренування моделей машинного навчання, механізми автентифікації, функціонал прогнозування та комплекс засобів зберігання й відображення даних.

Реалізований модуль навчання моделей забезпечує автоматизоване створення, оптимізацію та кешування алгоритмів, що надалі використовуються під час роботи застосунку. Це дає можливість системі оперувати кількома моделями з різними характеристиками точності та обирати найбільш придатні підходи до прогнозування.

У межах модуля автентифікації було реалізовано безпечну роботу з обліковими записами, включно із перевіркою введених даних, гешуванням паролів, автоматичним перенаправленням неавторизованих користувачів і журналюванням дій.

Модуль прогнозування надає можливість вводити медичні показники, обирати алгоритм, отримувати й аналізувати результати, а також переглядати деталі кожного прогнозу. Реалізовано можливість повторного виконання прогнозу з іншою моделлю, керування власною історією та відображення результатів у зручному форматі.

Описано розробку інтерфейсу користувача. Його структура орієнтована на простоту взаємодії, передбачає використання шаблонів Jinja2, стилізацію через CSS, чітку навігацію та інформативне відображення повідомлень і результатів. Макети форм, таблиць і сторінок було створено таким чином, щоб процес введення й перегляду даних залишався максимально інтуїтивним.

Також реалізовано повноцінну реляційну базу даних SQLite із застосуванням ORM-бібліотеки SQLAlchemy. У ній зберігаються користувачі, моделі, дані для прогнозів, результати обчислень і активність користувачів.

Отже, у цьому розділі було представлено комплекс технічних рішень, що разом формують узгоджену, функціональну та масштабовану систему. Інтеграція кожного модулю створює технологічну основу для подальших експериментальних досліджень, тестування та практичного використання вебзастосунку у сфері прогнозування медичних ризиків.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Порівняння натренованих моделей машинного навчання

Для того щоб оцінити якість навчання побудованих моделей та перевірити обґрунтованість вибору алгоритмів на етапі проєктування, необхідно здійснити детальне порівняння їхньої продуктивності на єдиному тестовому наборі даних. Такий аналіз дає змогу визначити, наскільки ефективно кожен метод розв’язує поставлену задачу, виявити сильні та слабкі сторони окремих підходів, а також оцінити доцільність використання ансамблевого підходу. Порівнюючи моделі між собою за різними метриками, можна зробити висновки щодо їх відповідності характеристикам даних та підтвердити або спростувати початкові припущення щодо вибору цих алгоритмів для системи прогнозування ризику захворювання.

Для проведення порівняльного аналізу було розроблено окремий скрипт, який автоматично завантажує всі попередньо натреновані моделі машинного навчання з бази даних та відповідних файлів. Усім моделям надаються однакові умови тестування – використовується спільний набір даних, розділений на навчальну та тестову вибірки з фіксованим параметром випадковості. У процесі роботи скрипт обчислює основні метрики класифікації і виводить їх у файл формату CSV, також скрипт будує графічні візуалізації (ROC-криві та Precision–Recall криві), що дозволяє наочно оцінити поведінку кожної моделі.

Переформатуємо отриманий CSV-файл з результатами у табл. 4.1 та проведемо аналіз результатів.

Представлені результати показують помітний розрив у якості між простішими моделями та деревоподібними методами. Лінійні підходи, а саме логістична регресія та SVM демонструють стабільну, але помірну ефективність, тоді як Naïve Bayes, попри високу точність на даному наборі,

страждає від підвищеної кількості хибних спрацьовувань (FPR), що свідчить про його обмежену здатність формувати складні роздільні межі.

Таблиця 4.1 – Порівняння метрик моделей

Модель	Accuracy	Precision	Recall	F1-score	FPR	AUC ROC	AUC PRC
SVM	0.786	0.841	0.799	0.819	0.000	0.877	0.925
NB	0.874	0.977	0.811	0.887	0.029	0.968	0.976
RF	0.973	0.981	0.975	0.978	0.000	0.984	0.990
XGB	0.981	0.981	0.987	0.984	0.019	0.980	0.987
LR	0.813	0.877	0.805	0.839	0.010	0.905	0.935
Ensemble	0.969	0.981	0.969	0.975	0.000	0.980	0.984

Найкращі результати дають Random Forest та XGBoost. Обидві моделі забезпечують високу точність класифікації і збалансовані показники Precision, Recall та F1-міри. XGBoost має найвище значення Recall, що робить його найбільш чутливим до позитивного класу, тоді як Random Forest демонструє найнижчий рівень хибних спрацьовувань (нульову на нашому наборі), гарантуючи стабільну роботу без надмірної кількості хибно позитивних результатів. Площі під ROC та PRC кривими для цих моделей також є максимальними, що підтверджує їх узгоджену загальну якість.

Зважений ансамбль очікувано показує результат, близький до найкращих окремих моделей. Його точність та F1-score свідчать про ефективне поєднання прогнозів, хоча він не перевершує XGBoost та Random Forest через незначний, але присутній вплив слабших моделей. Це означає, що вагове голосування дає позитивний ефект, але не здатне значно покращити результат найсильніших моделей, що є типовим у випадках, коли окремі базові алгоритми вже мають дуже високу якість.

Окрім табличних метрик, для повноцінної оцінки роботи моделей скрипт автоматично будує графіки ROC та Precision–Recall кривих (рис. 4.1-4.2). Ці візуалізації дають змогу оцінити стійкість моделей до зміни порога класифікації та порівняти їх поведінку в різних режимах роботи.

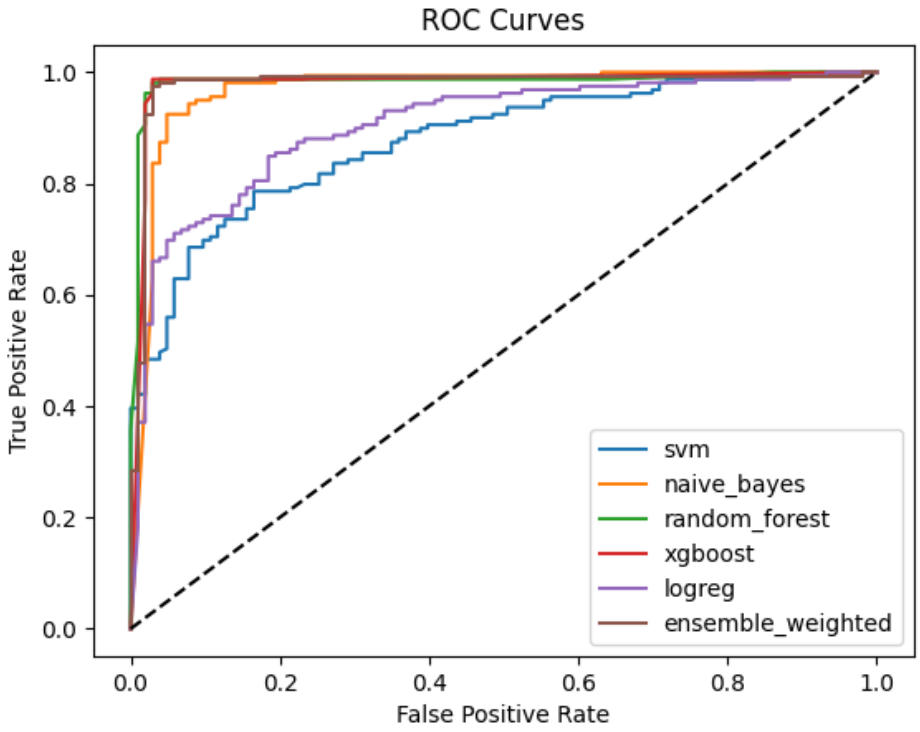


Рисунок 4.1 – ROC-крива

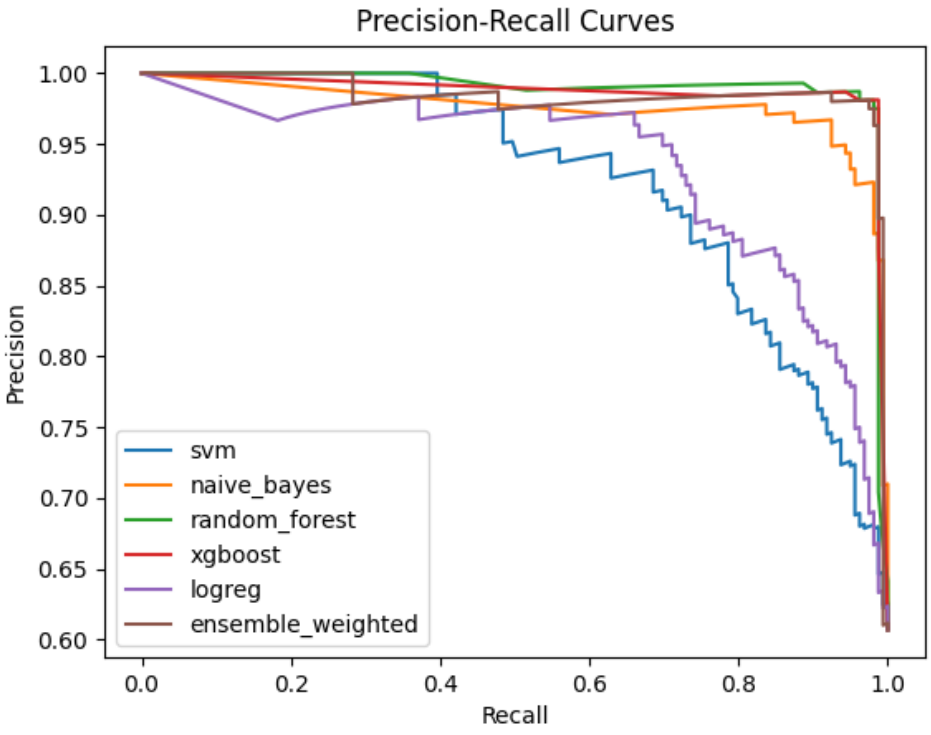


Рисунок 4.2 – Precision-Recall-крива

За графіками видно, що XGBoost і Random Forest демонструють найкращу якість: їхні криві проходять ближче до верхніх меж графіків, що означає стабільно хороші результати за будь-якого порога. Зважений ансамбль також показує сильні позиції – його лінії розташовані практично разом із найкращими моделями, хоча часом трохи їм поступаються. Натомість SVM і логістична регресія знаходяться нижче, що узгоджується з попередніми числовими метриками. Naïve Bayes залишається середнім між двома групами методом, але показує непоганий результат. Precision–Recall графіки підтверджують ті ж тенденції: деревоподібні моделі та ансамбль утримують високу якість навіть при зміні порога, тоді як інші методи швидше втрачають точність або повноту.

Отримані результати виявили тенденції, які не повністю відповідали початковим очікуванням. Деревоподібні методи фактично не були в центрі уваги під час відбору алгоритмів, а XGBoost узагалі не розглядався як один з основних кандидатів. Попри це, саме Random Forest та XGBoost показали найкращі показники за всіма ключовими метриками, продемонструвавши значно вищу ефективність, ніж більш традиційні методи.

SVM, який на етапі проєктування вважався одним із головних претендентів через свою поширеність у задачах класифікації, у підсумку опинився серед найслабших моделей у цій вибірці. Це підкреслює важливість емпіричної перевірки навіть популярних методів, оскільки їхня реальна ефективність сильно залежить від характеру конкретних даних.

Реалізований зважений ансамбль показав результат, близький до найсильніших окремих моделей. Хоча він не перевершив Random Forest чи XGBoost, його стабільна якість свідчить, що в умовах, коли базові моделі працюватимуть нерівномірно або гірше, ансамблевий підхід здатен пом’якшити такі коливання та зберегти високу точність прогнозів.

4.2 Експлуатація програми

4.2.1 Розгортання та керування програмною системою

Для коректного розгортання та подальшої експлуатації вебзастосунку необхідно забезпечити відповідність системного середовища вимогам проєкту. Оскільки серверна частина застосунку реалізована мовою Python, першочерговою умовою є встановлення інтерпретатора Python версії 3.12.10, який гарантує сумісність бібліотек, що використовуються, та стабільність роботи всіх компонентів системи. Пакетний менеджер pip, що постачається разом з Python, використовується для встановлення зовнішніх залежностей, перелік яких визначено у файлі requirements.txt. Після завантаження вихідного коду усі необхідні пакети інсталюються автоматично шляхом передання переліку пакетів пакетному менеджеру, що забезпечує повну відтворюваність програмного середовища.

Особливістю розгортання є наявність модуля машинного навчання, який виконує попереднє тренування моделей. Перед запуском вебсервера користувач повинен перейти до каталогу ml і послідовно виконати всі скрипти, що відповідають за тренування окремих моделей – SVM, Naive Bayes, Random Forest, XGBoost і Logistic Regression. Завершальним етапом є запуск скрипта побудови зваженого ансамблю, який використовує результати всіх попередніх тренувань. Кожен із цих скриптів формує відповідний файл моделі у форматі .pkl та зберігає його до каталогу models, а також створює або оновлює запис у таблиці models у базі даних. Таким чином, перед запуском вебінтерфейсу системи формується повний набір натренованих моделей, необхідний для коректної роботи механізму прогнозування.

Після налаштування середовища та формування моделей система готова до запуску. Основний сервер застосунку розгортається за допомогою запуску файлу main.py, причому робочою директорією має бути коренева папка проєкту, оскільки саме від її розташування залежить коректне

визначення відносних шляхів до модулів, бази даних та каталогу зі збереженими моделями. Під час запуску створюється екземпляр Flask-застосунку, ініціалізуються всі внутрішні модулі, виконується підключення до бази даних SQLite та реєструються маршрути. Сервер відкриває доступ до вебінтерфейсу за портом 5000 локальної адреси, після чого користувач може переходити до авторизації та роботи з функціоналом системи через вебпереглядач.

Таким чином, процедура розгортання програмного забезпечення складається з налаштування Python-середовища, встановлення залежностей, попереднього тренування моделей машинного навчання та запуску серверної частини застосунку. Така послідовність дій забезпечує відтворюваність програмної системи на будь-якій машині, що відповідає технічним вимогам, та дозволяє швидко розгорнути застосунок для подальшого використання або розвитку. За бажанням, система може бути доповнена іншими моделями, чи поточні моделі можуть бути оптимізовані кращими параметрами.

4.2.2 Користування програмною системою

Після завершення процесу розгортання програмної системи та запуску локального сервера користувач може взаємодіяти з вебзастосунком через вебпереглядач. Першою сторінкою, що відкривається при переході за адресою сервера, є форма авторизації. Вона містить поля для введення імені користувача та пароля, а також кнопку для переходу до сторінки реєстрації у випадку, якщо обліковий запис ще не створений. На цій сторінці користувач може увійти до системи або перейти до створення нового профілю. Ця сторінка зображена на рис. 4.3.

Для нових користувачів доступна сторінка реєстрації. Вона містить просту форму, де необхідно ввести ім’я користувача та пароль, що відповідають вимогам валідності. Після успішного заповнення форми та

створення профілю система автоматично пропонує увійти до застосунку з новоствореними обліковими даними. Ця сторінка зображена на рис. 4.4.

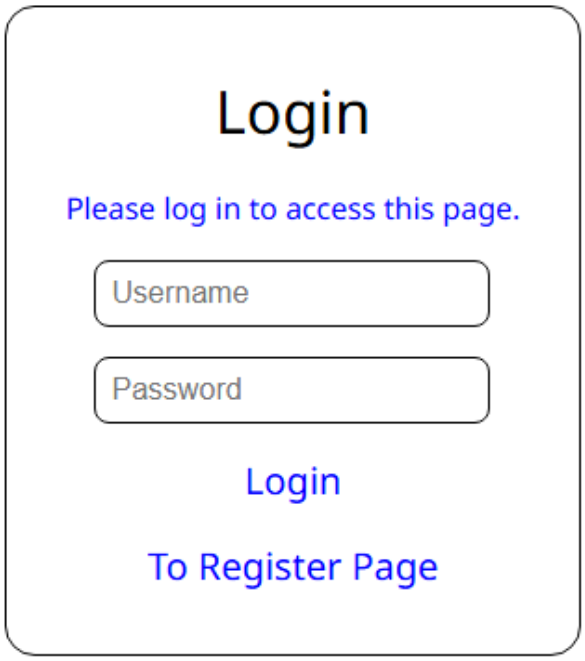


Рисунок 4.3 – Сторінка авторизації

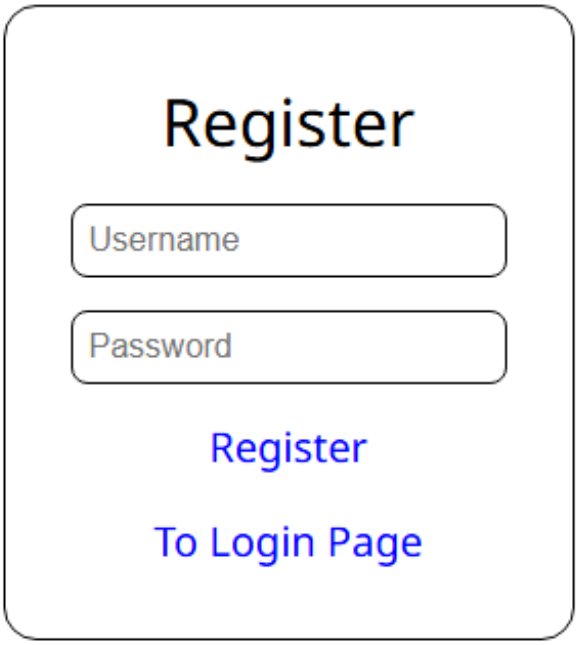


Рисунок 4.4 – Сторінка реєстрації

Після авторизації користувача перенаправляє на головну сторінку застосунку – панель керування. Тут відображається ім’я користувача, кнопка виходу з облікового запису та два основні елементи навігації: перехід до створення нового прогнозу та перегляд історії прогнозів. Інтерфейс є мінімалістичним та призначений для швидкої взаємодії з функціоналом системи. Ця сторінка зображена на рис. 4.5.

Для створення нового прогнозу користувач переходить до відповідної форми, натиснувши кнопку «Predict». На цій сторінці потрібно заповнити поля з медичними показниками: вік, стать, частоту серцевих скорочень, систолічний та діастолічний тиск, рівень глюкози, значення КК-МВ та тропоніну. Крім цього, користувач обирає одну з доступних моделей машинного навчання. Після подання форми система автоматично обробляє введені дані, виконує прогнозування та зберігає результат. Ця сторінка зображена на рис. 4.6.

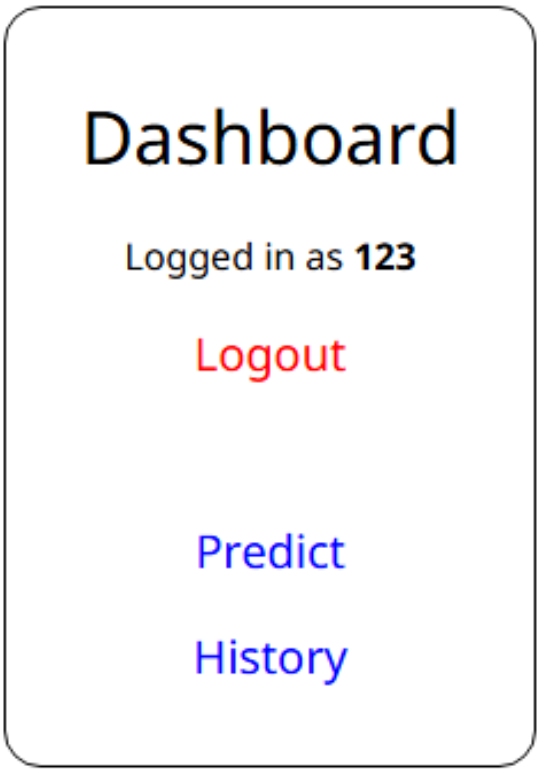


Рисунок 4.5 – Головна сторінка

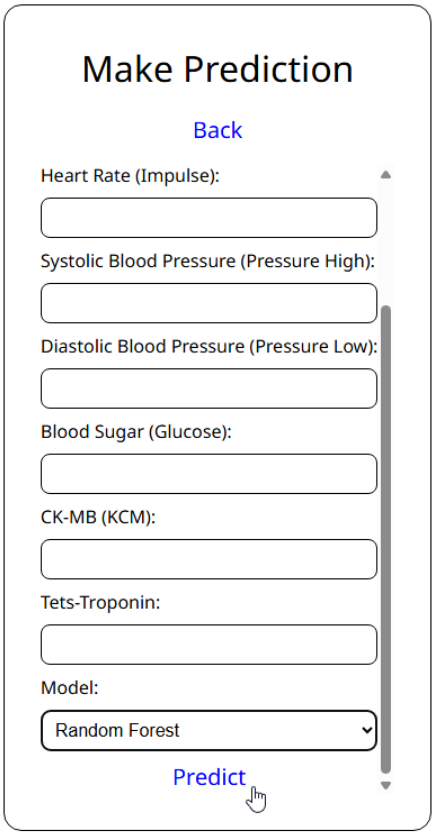


Рисунок 4.6 – Сторінка створення прогнозу

Після успішного створення прогнозу користувач перенаправляється на сторінку з детальною інформацією про результат. Тут відображаються дата виконання прогнозу, використана модель, підсумований результат («positive» або «negative»), а також рівень ймовірності у відсотках. Передбачене графічне відображення ймовірності у вигляді заповненої смуги. Нижче користувач може переглянути всі введені ним показники. На сторінці також доступна можливість повторно виконати прогноз із використанням іншої моделі, не вводячи дані повторно. Ця сторінка зображена на рис. 4.7.

Щоб переглянути всі попередньо створені прогнози, користувач може перейти до розділу «History». На цій сторінці подано таблицю з усіма прогнозами, відсортованими від найновіших до найстаріших. У таблиці відображається дата виконання прогнозу, модель, результат, ймовірність та кнопки для перегляду деталей кожного запису. Звідси ж користувач може

видалити окремий прогноз, після чого відповідні дані в системі також видаляються. Ця сторінка зображена на рис. 4.8.

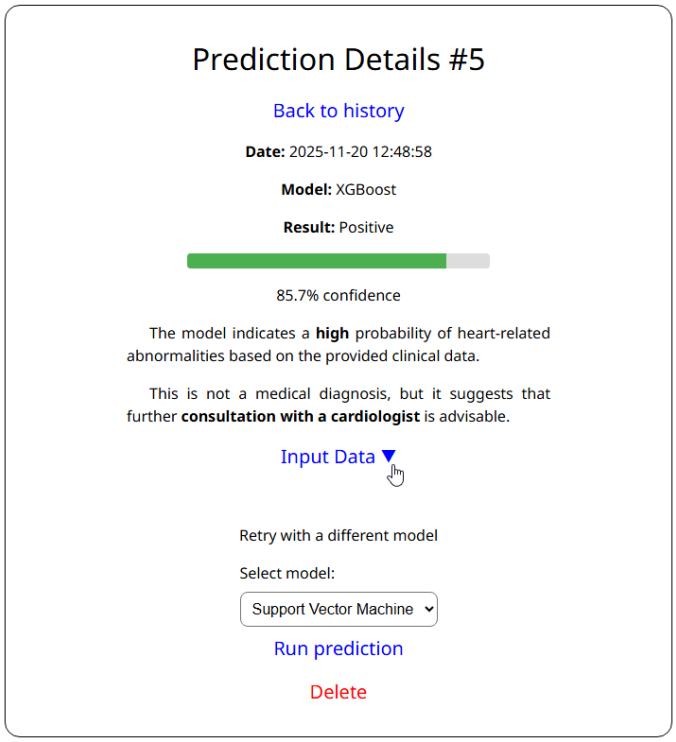
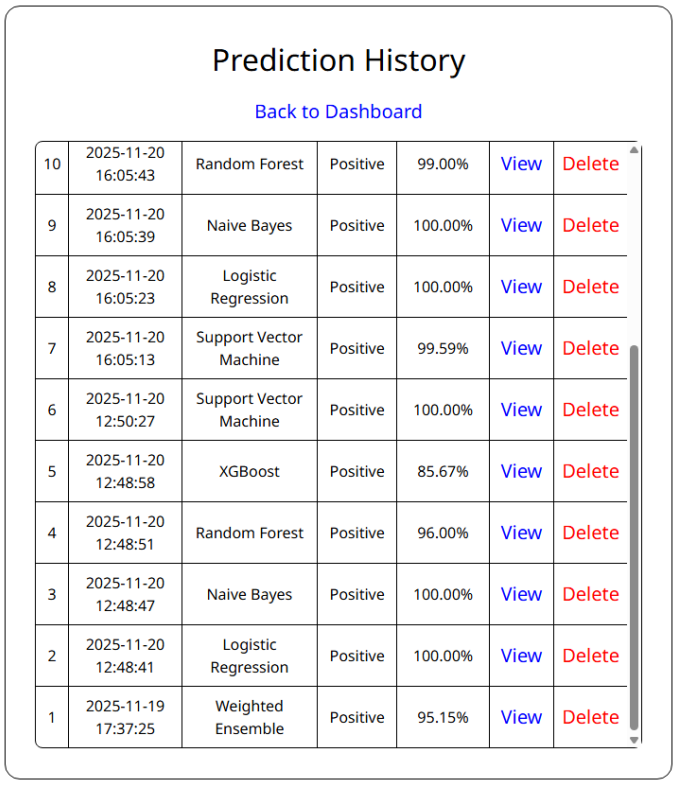


Рисунок 4.7 – Сторінка деталізації прогнозу



10	2025-11-20 16:05:43	Random Forest	Positive	99.00%	View	Delete
9	2025-11-20 16:05:39	Naive Bayes	Positive	100.00%	View	Delete
8	2025-11-20 16:05:23	Logistic Regression	Positive	100.00%	View	Delete
7	2025-11-20 16:05:13	Support Vector Machine	Positive	99.59%	View	Delete
6	2025-11-20 12:50:27	Support Vector Machine	Positive	100.00%	View	Delete
5	2025-11-20 12:48:58	XGBoost	Positive	85.67%	View	Delete
4	2025-11-20 12:48:51	Random Forest	Positive	96.00%	View	Delete
3	2025-11-20 12:48:47	Naive Bayes	Positive	100.00%	View	Delete
2	2025-11-20 12:48:41	Logistic Regression	Positive	100.00%	View	Delete
1	2025-11-19 17:37:25	Weighted Ensemble	Positive	95.15%	View	Delete

Рисунок 4.8 – Сторінка історії прогнозів

Функція видалення прогнозу дозволяє користувачу очищати історію за потреби. Система додатково перевіряє права доступу, щоб запобігти доступу чи модифікації даних іншими користувачами. Перед видаленням виводиться запит підтвердження, що допомагає уникнути випадкового стирання важливих даних. Операцію видалення зображено на рис. 4.9.

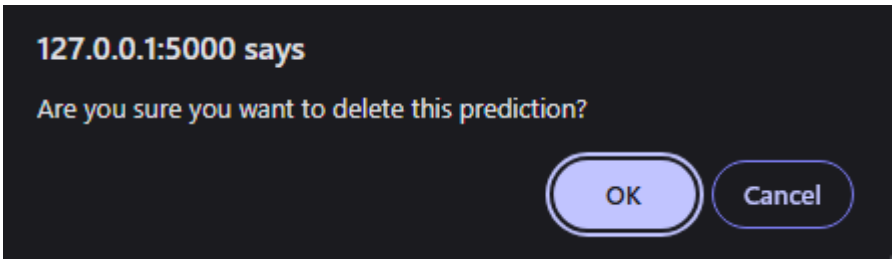


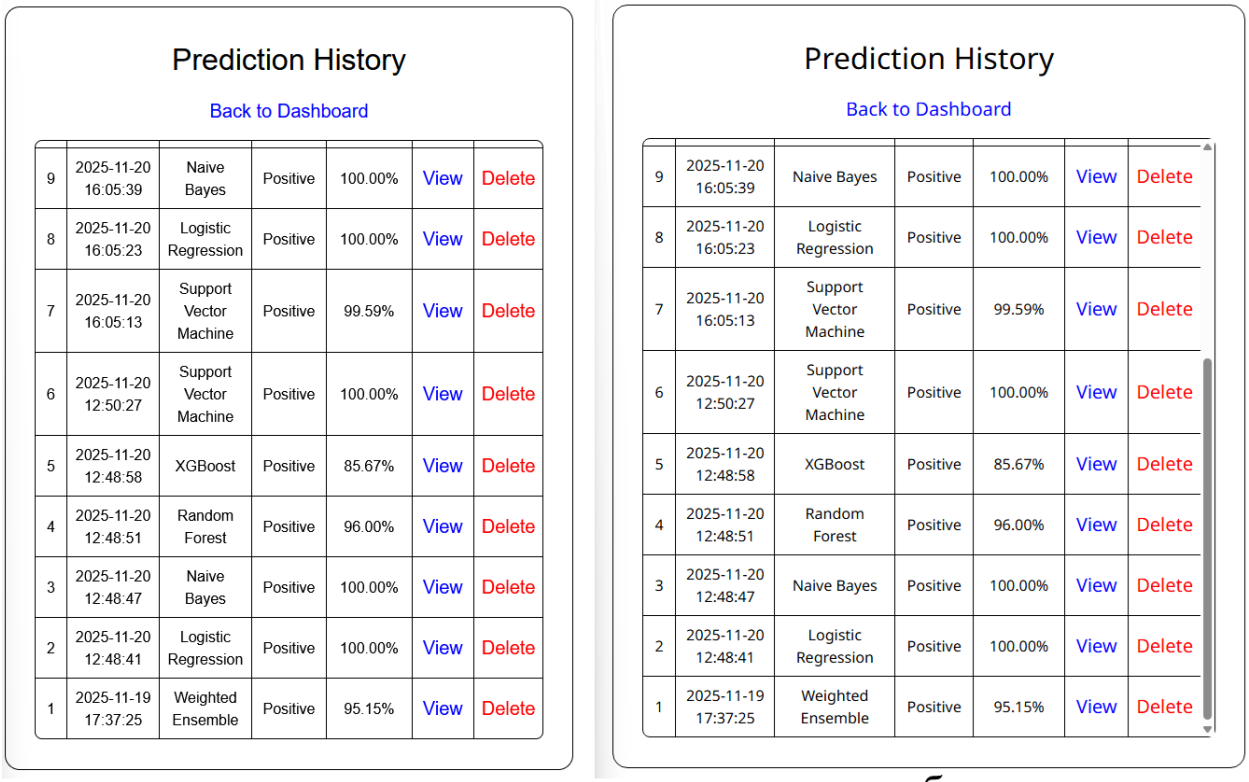
Рисунок 4.9 – Видалення прогнозу з історії

Таким чином, реалізована система забезпечує повний цикл взаємодії з користувачем: від автентифікації та роботи з основними сторінками до створення прогнозів, перегляду деталей, роботи з історією та управління власними даними. Інтерфейс підтримує користувача на всіх етапах роботи та забезпечує зрозумілий, послідовний і логічний спосіб взаємодії з програмою.

4.3 Тестування програмної системи

Для підтвердження коректності роботи програмної системи було проведено комплексне тестування її інтерфейсу та функціоналу у різних вебпереглядачах. Оскільки сучасний вебпростір базується переважно на двох рушіях – Chromium та Gecko, тестування виконувалося саме на них. Рушій Chromium застосовується у більшості популярних браузерів, зокрема у Google Chrome, Microsoft Edge, Opera та Brave, тоді як рушій Gecko використовується Firefox та його похідними. Таким чином, перевірка у двох незалежних рушіях дозволяє переконатися у сумісності вебзастосунку з основними платформами та гарантує правильність його роботи в більшості реальних умов використання.

Першим етапом тестування стала перевірка відображення користувацького інтерфейсу, зокрема коректності верстки, адаптації елементів і узгодженості зовнішнього вигляду у двох різних середовищах рендерингу. Для цього було виконано порівняння вигляду ключових сторінок застосунку у браузерах на базі Chromium та Gecko. Візуальні результати цього етапу наведені на рис. 4.10, де зображено інтерфейс застосунку.



а б

а – у Chromium, б – у Gecko.

Рисунок 4.10 – Порівняння відображення інтерфейсу

Наступним етапом тестування стала перевірка функціональної коректності роботи вебзастосунку. Для цього було підготовлено чекліст, що охоплює ключові сценарії взаємодії користувача з системою від базових операцій до більш складних дій, пов’язаних із отриманням та обробленням прогнозів. Кожен пункт перевірявся окремо у середовищах Chromium та Gecko, що дозволило встановити, наскільки рівномірно реалізований

функціонал працює у різних браузерах. Узагальнений результат тестування подано у табл. 4.2.

Таблиця 4.2 – Чекліст

Параметр	Chromium	Gecko
Проходження реєстрації з валідними даними	Пройдено	Пройдено
Обробка помилок при реєстрації з невалідними даними	Пройдено	Пройдено
Авторизація з правильними обліковими даними	Пройдено	Пройдено
Обробка хибних комбінацій при авторизації	Пройдено	Пройдено
Автоматичне перенаправлення неавторизованого користувача на сторінку логіну	Пройдено	Пройдено
Перенаправлення на головну сторінку після авторизації	Пройдено	Пройдено
Перехід до сторінки створення прогнозу	Пройдено	Пройдено
Робота форми введення медичних показників та її валідації	Пройдено	Пройдено
Зміна моделі прогнозування зі списку доступних	Пройдено	Пройдено
Формування прогнозу після відправлення форми	Пройдено	Пройдено
Перезапуск прогнозу з іншою моделлю з тієї ж сторінки	Пройдено	Пройдено
Перехід до історії прогнозів	Пройдено	Пройдено
Коректне відображення прогнозів у хронологічному порядку	Пройдено	Пройдено
Перегляд деталей існуючого прогнозу зі сторінки історії	Пройдено	Пройдено
Видалення прогнозу і коректне видалення пов’язаних вхідних даних із БД	Пройдено	Пройдено
Запис подій у журнал активності	Пройдено	Пройдено
Робота кнопки виходу з облікового запису	Пройдено	Пройдено
Відсутність помилок рендерингу та скриптів у консолі браузера	Пройдено	Пройдено

Приклад відображення повідомлень про помилки у вебзастосунку зображено на рис. 4.11.

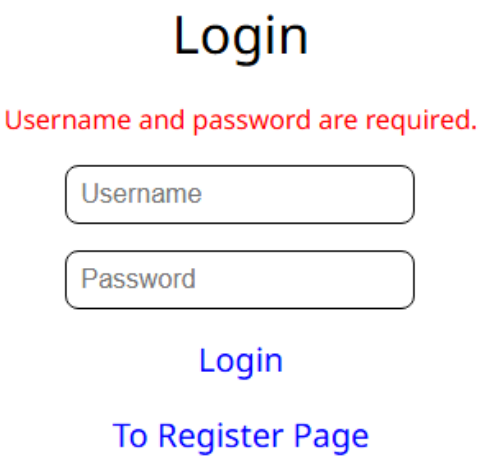


Рисунок 4.11 – Приклад повідомлення про помилку

4.4 Висновки до розділу 4

У цьому розділі було проведено комплексний аналіз роботи створеної програмної системи, що охоплює порівняння моделей машинного навчання, дослідження особливостей експлуатації вебзастосунку та перевірку його функціональної коректності.

На етапі експериментального оцінювання моделей здійснено порівняння їх точності, повноти, F1-міри та інших метрик, що дало змогу визначити найефективніші алгоритми та підтвердити доцільність використання ансамблевого підходу.

У ході дослідження експлуатаційних характеристик системи розглянуто процедуру її розгортання, налаштування та запуску в локальному середовищі розробника, а також детально описано порядок взаємодії користувача з вебзастосунком.

Тестування підтвердило стабільну роботу застосунку в різних середовищах та відповідність його поведінки вимогам, сформульованим на етапі проєктування.

Загалом отримані результати демонструють, що реалізована система є працездатною, функціонально повною та готовою до подальшої інтеграції, вдосконалення або розгортання у реальних умовах використання.

ВИСНОВКИ

У процесі виконання дипломної кваліфікаційної роботи магістра було проведено комплексне дослідження проблеми прогнозування ризиків захворювань на основі медичних показників та проаналізовано можливості використання методів машинного навчання у медичних інформаційних системах. Розглянуто предметну область медичної діагностики, окреслено основні труднощі, пов’язані з обробленням значних обсягів даних і необхідністю підтримки прийняття рішень лікарями.

Огляд існуючих наукових досліджень показав, що для задач прогнозування ризиків захворювань найчастіше застосовують логістичну регресію, дерева рішень, випадкові ліси, SVM, наївний баєсівський класифікатор й інші ансамблеві підходи. У багатьох роботах продемонстровано, що комбіновані або ансамблеві моделі забезпечують помітне підвищення точності порівняно з окремими базовими алгоритмами.

У роботі розроблено архітектуру вебзастосунку, що включає вебінтерфейс користувача, серверну частину, модулі машинного навчання та базу даних. Побудовано структурну схему системи, UML-діаграму компонентів та ER-діаграму таблиць, що забезпечило чітке уявлення про внутрішню організацію застосунку та взаємодію його складових.

У ході роботи створено повноцінний вебзастосунок для прогнозування ризиків захворювань, що поєднує серверну частину на Flask, клієнтський інтерфейс на основі HTML-шаблонів та інтегровані моделі машинного навчання. Реалізовано механізми автентифікації користувачів, обробки введених медичних показників, формування прогнозів та збереження історії результатів у базі даних SQLite. Окремий модуль машинного навчання забезпечує підготовку даних, навчання та оновлення моделей, а також автоматичну реєстрацію отриманих параметрів у системі. Для розв’язання задачі прогнозування впроваджено та налаштовано декілька алгоритмів, а саме Support Vector Machine, Naïve Bayes, Random Forest,

XGBoost, Logistic Regression та зважену ансамблеву модель, що дозволило створити функціональну, інтегровану програмну систему, придатну для подальшого використання й розширення.

Створено окремий модуль тестування, який оцінює отримані моделі, на спільному тестовому наборі та порівнює за низкою метрик: точністю, повнотою, точністю позитивного прогнозу, F1-мірою, AUC ROC та AUC PRC. За результатами експериментів встановлено, що найвищу якість показали моделі XGBoost та Random Forest, які не розглядалися як ключові на етапі відбору методів, але продемонстрували найкращі результати серед усіх. Власна реалізація зваженого ансамблю забезпечила стабільно високі результати, близькі до найсильніших окремих моделей, і може бути корисною у ситуаціях, де окремі алгоритми дають нестійкі прогнози.

У результаті виконаної роботи створено повноцінний вебзастосунок, який забезпечує введення медичних показників користувачем, їх оброблення, аналіз та прогнозування ризику захворювання за допомогою машинного навчання.

Система може застосовуватися у медичних установах як інструмент підтримки прийняття рішень або використовуватися пацієнтами для самостійного моніторингу свого стану.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Laskar, Md Tahmid Rahman. Automated Disease Prediction System (ADPS): A User Input-based Reliable Architecture for Disease Prediction / Md Tahmid Rahman Laskar, Md. Hossain, Abu Kamal, Nafiul Rashid // International Journal of Computer Applications. – 2016. – Vol. 133, № 6. – P. 24–29.

2. Gangadhara Moorthy, Saranya. A comprehensive study on disease risk predictions in machine learning / Saranya Gangadhara Moorthy, A. Pravin // International Journal of Electrical and Computer Engineering (IJECE). – 2020. – Vol. 10, № 4. – P. 4217–4225.

3. Qiu, J. A survey of machine learning for big data processing / [J. Qiu, Q. Wu, G. Ding et al.] // EURASIP Journal on Advances in Signal Processing. – 2016. – № 67.

4. Shickel, B. Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis / B. Shickel, P. J. Tighe, A. Bihorac, P. Rashidi // IEEE Journal of Biomedical and Health Informatics. – 2018. – Vol. 22, № 5. – P. 1589–1604.

5. Badawy, M. Healthcare predictive analytics using machine learning and deep learning techniques: a survey / M. Badawy, N. Ramadan, H. A. Hefny // Journal of Electrical Systems and Information Technology. – 2023. – Vol. 10, № 40.

6. Krishnamoorthi, R. A novel diabetes healthcare disease prediction framework using machine learning techniques / [R. Krishnamoorthi, S. Joshi, H. Z. Almarzouki et al.] // Journal of Healthcare Engineering. – 2022.

7. Deberneh, H. M. Prediction of Type 2 Diabetes Based on Machine Learning Algorithm / H. M. Deberneh, I. Kim // International Journal of Environmental Research and Public Health. – 2021. – Vol. 18, № 6. – Art. 3317.

8. Seera, M. A hybrid intelligent system for medical data classification / M. Seera, C. P. Lim // Expert Systems with Applications. – 2014. – Vol. 41, № 5. – P. 2239–2249.

9. Dangare, Chaitrali S. Improved Study of Heart Disease Prediction System using Data Mining Classification Techniques / Chaitrali S. Dangare, Sulabha S. Apte // International Journal of Computer Applications. – 2012. – Vol. 47, № 10. – P. 44–48.

10. Shouman, Mai. Integrating Naive Bayes and K-Means Clustering with Different Initial Centroid Selection Methods in the Diagnosis of Heart Disease Patients / Mai Shouman, Timothy Turner, Rob Stocker // Computer Science & Information Technology. – 2012. – Vol. 2.

11. Zaitcev, Aleksandr. A Deep Neural Network Application for Improved Prediction of HbA_{1c} in Type 1 Diabetes / [Aleksandr Zaitcev, Mohammad R. Eissa, Zheng Hui et al.] // IEEE Journal of Biomedical and Health Informatics. – 2020. – Vol. 24, № 10. – P. 2932–2941.

12. Janosi, Andras. Heart Disease [Electronic resource] / Andras Janosi, William Steinbrunn, Matthias Pfisterer, Robert Detrano // UCI Machine Learning Repository. – 1989. – Access mode: https://doi.org/10.24432/C52P4X.

13. Mammadov, Rashad. Heart Disease prediction [Electronic resource] / Rashad Mammadov, Sitara Aghayeva // Kaggle. – Access mode: https://www.kaggle.com/datasets/rashadrmammadov/heart-disease-prediction/data.

14. Maghdid, Sozan S. An Extensive Dataset for the Heart Disease Classification System [Electronic resource] / Sozan S. Maghdid, Tarik A. Rashid // Mendeley Data. – 2022. – Version 2. – Access mode: https://data.mendeley.com/datasets/65gxgy2nmg/2.

15. Gupta, M. A comparative study on supervised machine learning algorithm / M. Gupta, S. D. Pandya // International Journal of Research in Applied Science & Engineering Technology (IJRASET). – 2022. – Vol. 10, № 1. – P. 1023–1028.

16. Srivastava, A. Comparison of various machine learning techniques and its uses in different fields / A. Srivastava, S. Saini, D. Gupta // 2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, 12–14 June 2019. – IEEE, 2019. – P. 81–86.

17. Obulesu, O. Machine learning techniques and tools: a survey / O. Obulesu, M. Mahendra, M. Thrilok Reddy // 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, 11–12 July 2018. – IEEE, 2018. – P. 605–611.

18. Batta Mahesh. Machine Learning Algorithms - A Review / Batta Mahesh // International Journal of Science and Research (IJSR). – 2020. – Vol. 9, № 1. – P. 381–386.

19. Alom, Md Zahangir. A State-of-the-Art Survey on Deep Learning Theory and Architectures / [Md Zahangir Alom, Tarek M. Taha, Chris Yakopcic et al.] // Electronics. – 2019. – Vol. 8, № 3. – Art. 292.

20. Classification: Accuracy, recall, precision, and related metrics [Electronic resource] // Google Machine Learning Crash Course. – Access mode: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

21. Classification: ROC and AUC [Electronic resource] // Google Machine Learning Crash Course. – Access mode: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

22. Backend Development Languages Comparison: Python, Java, Node.js [Electronic resource] // Hostragons Global Limited. – 2025. – Access mode: https://www.hostragons.com/en/blog/backend-development-languages-python-java-node-js/.

23. Semik, Wojciech. Flask vs. Django: Which Python Framework Is Better for Your Web Development? [Electronic resource] / Wojciech Semik // STX Next. – 2025. – Access mode: https://www.stxnext.com/blog/flask-vs-django-comparison.

24. Dennis, Yancy. Flask vs Django vs Streamlit [Electronic resource] / Yancy Dennis // Python in Plain English. – 2023. – Access mode: https://python.plainenglish.io/flask-vs-django-vs-streamlit-40da8afccb92.

25. Yigal, Asaf. Sqlite vs. MySQL vs. PostgreSQL: A Comparison of

Relational Databases [Electronic resource] / Asaf Yigal // Logz.io. – 2018. – Access mode: https://logz.io/blog/relational-database-comparison/.

26. Kafesu, Anesu. VS Code vs. Pycharm: The Best IDE for Python [Electronic resource] / Anesu Kafesu // Geekflare. – 2024. – Access mode: https://geekflare.com/dev/vs-code-vs-pycharm/.

ДОДАТОК А

Технічне завдання

А.1 Вступ

Проєкт – розробка вебзастосунку, що здійснює оброблення та аналіз медичних показників користувача із застосуванням моделей машинного навчання, що дає змогу визначати його стан і підтримувати процес моніторингу здоров’я.

А.2 Підстава для розробки

Дипломна робота на здобуття освітнього ступеня бакалавра має тему «Розробка та дослідження системи прогнозування ризиків захворювань на основі методів машинного навчання», затверджену наказом Національного університету «Запорізька політехніка» №447 від 30 вересня 2025 року.

А.3 Призначення розробки

Основною ціллю роботи є створення програмної системи для прогнозування ризиків захворювань на основі методів машинного навчання, який забезпечує аналіз введених користувачем медичних даних і формує прогноз стану його здоров’я.

А.4 Вимоги до розробки проєкту

А.4.1 Вимоги до функціоналу

Застосунок повинен реалізовувати наступні ключові функції:

– реєстрація та авторизація користувачів, що дозволить захистити персональні дані та гарантувати індивідуальний підхід до роботи із системою;

– обробка введених користувачем даних, включаючи валідацію форм, перевірку коректності та збереження інформації у базі даних;

– використання моделей машинного навчання для аналізу та прогнозування на основі введених даних;

– збереження результатів прогнозів із прив’язкою до користувача та часу їх отримання для подальшого перегляду та аналізу;

– логування основних дій користувачів для відстеження роботи системи та виявлення можливих помилок або зловживань;

– забезпечення можливості зміни моделі машинного навчання і повторного аналізу введених даних, а також оновлення моделей машинного навчання.

А.4.2 Вимоги до інтерфейсу

Інтерфейс системи має забезпечувати чітку, послідовну та інтуїтивно зрозумілу взаємодію користувача з функціональними модулями вебзастосунку. Основою інтерфейсу повинні бути HTML-сторінки зі стилізацією за допомогою CSS та підтримкою динамічного формування вмісту.

До інтерфейсу висуваються такі вимоги:

– наявність системи інформування користувача про помилки, результати виконання дій та інші стани системи;

– забезпечення простої та логічної навігації між основними модулями: авторизацією, виконанням прогнозування та історією результатів;

– подання результатів прогнозування у структурованій та зрозумілій формі з відображенням ключових параметрів і допоміжної інформації.

А.4.3 Вимоги до зберігання даних

Дані, що створюються та використовуються у вебзастосунку, повинні зберігатися в реляційній базі даних SQLite, інтегрованій у серверну частину

проєкту. У базі даних має зберігатися інформація про користувачів, журнали їхніх дій, введені параметри прогнозів, результати обчислень та метадані натренованих моделей машинного навчання.

База даних повинна функціонувати локально у середовищі виконання застосунку, що виключає можливість несанкціонованого доступу сторонніх користувачів та забезпечує контрольоване зберігання та оброблення інформації.

А.4.4 Технічні вимоги

Для забезпечення стабільної роботи програмної системи та підтримки процесу розробки необхідно дотримуватися таких технічних вимог:

- операційна система Windows 11 Pro;
- процесор Intel Core i7-13620H;
- оперативна пам’ять обсягом 16 ГБ;
- монітор із роздільною здатністю 1920×1080 пікселів;
- вебпереглядач на основі рушія Chromium версії 142;
- середовище розробки PyCharm Community Edition версії 2025.2.4;
- інтерпретатор мови Python версії 3.12.10.

Зазначені характеристики визначають оптимальні умови для розгортання, тестування та модернізації системи. Коректна робота програмного забезпечення можлива й на апаратних конфігураціях, близьких за параметрами до наведених, а також на новіших чи дещо спрощених системах, за умови підтримки необхідних технологій та залежностей.

ДОДАТОК Б

Текст програми

Б.1 Текст файлу train_ensemble.py

```
import pandas as pd
import pickle
import json
import os
import time
from datetime import datetime

# flask import
import sys
sys.path.append("..")
from app import create_app
from app.database import db
from app.models import Model as ModelEntry

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

def load_model(path):
    with open(path, "rb") as f:
        return pickle.load(f)

def train_ensemble():
    # load data
    df = pd.read_csv("datasets/dataset.csv")

    X = df.drop("class", axis=1)
    y = df["class"]
    y = y.map({"negative": 0, "positive": 1})

    # split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.20, random_state=42
    )

    # init flask app
    app = create_app()

    with app.app_context():
        # load all models except ensemble
        entries = ModelEntry.query.filter(ModelEntry.type != "ensemble_weighted").all()
```

```
if not entries:
    print("No models found in database.")
    return

models = []
accuracies = []

for e in entries:
    try:
        model = load_model(e.path)
        models.append(model)
        accuracies.append(e.accuracy)
    except Exception as err:
        print(f"Failed to load model {e.type}: {err}")

if not models:
    print("No valid models loaded.")
    return

# compute weights
POWER = 3
acc_array = np.array(accuracies, dtype=float)
acc_powered = acc_array ** POWER
weights = acc_powered / acc_powered.sum()

print("Models used:", [e.type for e in entries])
print("Weights:", weights)

# start timer
start = time.time()

# weighted soft voting
prob_sum = np.zeros((len(X_test), 2))

for model, w in zip(models, weights):
    probs = model.predict_proba(X_test)
    prob_sum += probs * w

final_pred = np.argmax(prob_sum, axis=1)

# end timer
end = time.time()
training_time = end - start
```

```
        # accuracy
        accuracy = accuracy_score(y_test, final_pred)

        # save ensemble
        os.makedirs("models", exist_ok=True)
        model_filename = "models/ensemble_weighted.pkl"

        ensemble_data = {
            "models": [(e.type, e.path) for e in entries],
            "weights": weights.tolist()
        }

        with open(model_filename, "wb") as f:
            pickle.dump(ensemble_data, f)

        # save in db
        entry = ModelEntry.query.filter_by(type="ensemble_weighted").first()
        if not entry:
            entry = ModelEntry(name="Weighted Ensemble",
type="ensemble_weighted")
            db.session.add(entry)

        entry.path = model_filename
        entry.accuracy = float(accuracy)
        entry.training_time = float(training_time)
        entry.params = json.dumps({
            "weights": weights.tolist(),
            "models": [e.type for e in entries]
        })
        entry.trained_at = datetime.now()

        db.session.commit()

        print("Ensemble saved at:", model_filename)
        print("Accuracy:", accuracy)
        print("Voting time:", training_time)


if __name__ == "__main__":
    train_ensemble()
```

Б.2 Текст файлу evaluate_models.py

```
import numpy as np
import pandas as pd
import pickle
import os
import sys
import json
import matplotlib.pyplot as plt

# flask
sys.path.append("..")
from app import create_app
from app.database import db
from app.models import Model as ModelEntry

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_curve,
    auc,
    precision_recall_curve,
)

def load_model(path):
    with open(path, "rb") as f:
        return pickle.load(f)

def evaluate_models():
    # load data
    df = pd.read_csv("datasets/dataset.csv")

    X = df.drop("class", axis=1)
    y = df["class"].map({"negative": 0, "positive": 1})

    # split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.20, random_state=42
    )
```

```
# init flask
app = create_app()

# prepare result storage
results = []

# load models
with app.app_context():
    entries = ModelEntry.query.all()

    if not entries:
        print("No models found.")
        return

    models = {}
    for e in entries:
        try:
            model = load_model(e.path)
            models[e.type] = (model, e.accuracy, e.path)
        except Exception as err:
            print(f"Failed to load model {e.type}: {err}")

# create dirs
os.makedirs("evaluation", exist_ok=True)

# for ROC/PRC plotting
roc_curves = {}
prc_curves = {}

# evaluate each ordinary model
for name, (model, stored_acc, path) in models.items():
    if name == "ensemble_weighted":
        continue

    # predict
    y_prob = model.predict_proba(X_test)[:, 1]
    y_pred = (y_prob >= 0.5).astype(int)

    # metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # FPR & ROC
```

```
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        auc_roc = auc(fpr, tpr)

        # PRC
        precision_pts, recall_pts, _ = precision_recall_curve(y_test, y_prob)
        auc_pr = auc(recall_pts, precision_pts)

        # save curves
        roc_curves[name] = (fpr, tpr)
        prc_curves[name] = (precision_pts, recall_pts)

        # append metrics
        results.append({
            "Model": name,
            "Accuracy": acc,
            "Precision": prec,
            "Recall": rec,
            "F1-score": f1,
            "FPR": fpr[1] if len(fpr) > 1 else 0.0,
            "AUC ROC": auc_roc,
            "AUC PRC": auc_pr
        })

        print(f"Evaluated: {name}  acc={acc:.4f}")

    # evaluate ensemble
    if "ensemble_weighted" in models:
        print("\nEvaluating weighted ensemble...")

        # load ensemble structure
        _, _, ensemble_path = models["ensemble_weighted"]
        with open(ensemble_path, "rb") as f:
            ensemble_data = pickle.load(f)

        model_types = [m[0] for m in ensemble_data["models"]]
        weights = np.array(ensemble_data["weights"], dtype=float)

        # load actual model objects
        ens_models = []
        for mtype in model_types:
            mdl, _, _ = models[mtype]
            ens_models.append(mdl)

        # compute weighted probabilities
        prob_sum = np.zeros(len(X_test))
```

```
    for mdl, w in zip(ens_models, weights):
        prob_sum += mdl.predict_proba(X_test)[:, 1] * w

    y_pred = (prob_sum >= 0.5).astype(int)

    # metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # ROC
    fpr, tpr, _ = roc_curve(y_test, prob_sum)
    auc_roc = auc(fpr, tpr)

    # PRC
    precision_pts, recall_pts, _ = precision_recall_curve(y_test, prob_sum)
    auc_pr = auc(recall_pts, precision_pts)

    # append
    results.append({
        "Model": "ensemble_weighted",
        "Accuracy": acc,
        "Precision": prec,
        "Recall": rec,
        "F1-score": f1,
        "FPR": fpr[1] if len(fpr) > 1 else 0.0,
        "AUC ROC": auc_roc,
        "AUC PRC": auc_pr
    })

    roc_curves["ensemble_weighted"] = (fpr, tpr)
    prc_curves["ensemble_weighted"] = (precision_pts, recall_pts)

    print(f"Ensemble evaluated. acc={acc:.4f}")

# convert to DataFrame
df_results = pd.DataFrame(results)
df_results = df_results.round(3)
df_results.to_csv("evaluation/model_comparison.csv", index=False)
plt.figure(figsize=(10, len(df_results) * 0.1))
plt.table(cellText=df_results.values,
          colLabels=df_results.columns,
          loc='center')
```

```
    plt.axis('off')
    plt.savefig('evaluation/model_table.png', dpi=300, bbox_inches='tight')
    plt.close()

    print("\nSaved evaluation table: evaluation/model_comparison.csv")

    # plot ROC
    plt.figure()
    for name, (fpr, tpr) in roc_curves.items():
        plt.plot(fpr, tpr, label=name)
    plt.plot([0, 1], [0, 1], "k--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curves")
    plt.legend()
    plt.savefig("evaluation/roc_curves.png")
    plt.close()

    print("Saved ROC curves: evaluation/roc_curves.png")

    # plot PRC
    plt.figure()
    for name, (prec_pts, rec_pts) in prc_curves.items():
        plt.plot(rec_pts, prec_pts, label=name)
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title("Precision-Recall Curves")
    plt.legend()
    plt.savefig("evaluation/prc_curves.png")
    plt.close()

    print("Saved PRC curves: evaluation/prc_curves.png")

    print("\nDone.")


if __name__ == "__main__":
    evaluate_models()
```

Б.3 Текст файлу auth_routes.py

```
from flask import Blueprint, render_template, request, redirect, url_for, flash
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required, current_user
import re

from .database import db
from .models import User, ActivityLog

auth = Blueprint("auth", __name__)

# validation helpers
def validate_username(u: str) -> bool:
    return bool(re.fullmatch(r"[A-Za-z0-9_]{3,32}", u))

def validate_password(p: str) -> bool:
    if len(p) < 8:
        return False
    if " " in p:
        return False
    if not re.search(r"[A-Za-z]", p):
        return False
    if not re.search(r"[0-9]", p):
        return False
    return True

# register
@auth.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "GET":
        return render_template("register.html")

    username = request.form.get("username", "").strip()
    password = request.form.get("password", "").strip()

    # server validation
    if not validate_username(username):
        flash("Username must be from 3 to 32 characters and contain only letters, digits and '_'.", "error")
        return redirect(url_for("auth.register"))

    if not validate_password(password):
```

```
        flash("Password must be at least 8 characters long, contain letters and digits, and have no spaces.", "error")
        return redirect(url_for("auth.register"))

    existing = User.query.filter_by(username=username).first()
    if existing:
        flash("Username already taken.", "error")
        return redirect(url_for("auth.register"))

    # create user
    password_hash = generate_password_hash(password)
    user = User(username=username, password_hash=password_hash)
    db.session.add(user)
    db.session.commit()

    # log
    log = ActivityLog(user_id=user.id, action="register")
    db.session.add(log)
    db.session.commit()

    flash("Account created successfully. Please log in.", "success")
    return redirect(url_for("auth.login"))

# login
@auth.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "GET":
        return render_template("login.html")

    username = request.form.get("username", "").strip()
    password = request.form.get("password", "").strip()

    if not username or not password:
        flash("Username and password are required.", "error")
        return redirect(url_for("auth.login"))

    user = User.query.filter_by(username=username).first()

    if not user or not check_password_hash(user.password_hash, password):
        flash("Invalid username or password.", "error")
        return redirect(url_for("auth.login"))

    login_user(user)

    log = ActivityLog(user_id=user.id, action="login")
```

```
    db.session.add(log)
    db.session.commit()

    return redirect(url_for("routes.index"))

# logout
@auth.route("/logout")
@login_required
def logout():
    log = ActivityLog(user_id=current_user.id, action="logout")
    db.session.add(log)
    db.session.commit()

    logout_user()
    return redirect(url_for("auth.login"))
```

Б.4 Текст файлу prediction_routes.py

```
import os
from flask import Blueprint, render_template, request, redirect, url_for, flash
from flask_login import login_required, current_user
import pickle
import numpy as np
from datetime import datetime, UTC
import json

from .database import db
from .models import Model, Prediction, PredictionInput, ActivityLog

prediction = Blueprint("prediction", __name__)

# helper to load model
def load_model(path):
    # if path is relative add ml
    if not os.path.isabs(path):
        if not path.startswith("ml/"):
            path = os.path.join("ml", path)

    # full path
    base = os.path.dirname(os.path.dirname(__file__))  # disease_prediction_app/
    full_path = os.path.join(base, path)

    with open(full_path, "rb") as f:
```

```
        return pickle.load(f)

@prediction.route("/predict", methods=["GET", "POST"])
@login_required
def predict():
    if request.method == "GET":
        models = Model.query.all()
        return render_template("predict.html", models=models)

    # get inputs
    # validation
    # expected numeric fields
    fields = [
        "age", "gender", "impulse", "pressure_high",
        "pressure_low", "glucose", "kcm", "troponin"
    ]

    data = {}

    for field in fields:
        raw = request.form.get(field)

        # missing
        if raw is None or raw == "":
            flash(f"{field} is required.", "error")
            return redirect(url_for("prediction.predict"))

        # convert to float
        try:
            value = float(raw)
        except ValueError:
            flash(f"{field} must be a valid number.", "error")
            return redirect(url_for("prediction.predict"))

        # negative check
        if value < 0:
            flash(f"{field} cannot be negative.", "error")
            return redirect(url_for("prediction.predict"))

        data[field] = value

    # gender must be 0 or 1
    if data["gender"] not in (0, 1):
        flash("Gender must be 0 (female) or 1 (male).", "error")
        return redirect(url_for("prediction.predict"))
```

```
# model id
try:
    model_id = int(request.form.get("model_id"))
except:
    flash("Invalid model selection.", "error")
    return redirect(url_for("prediction.predict"))

# get model
model_entry = Model.query.get(model_id)
if not model_entry:
    flash("Model not found.", "error")
    return redirect(url_for("prediction.predict"))

model = load_model(model_entry.path)

# prepare vector
X = np.array([[
    data["age"], data["gender"], data["impulse"],
    data["pressure_high"], data["pressure_low"],
    data["glucose"], data["kcm"], data["troponin"]
]])

# predict
# ensemble
if isinstance(model, dict):
    models_list = model["models"]
    weights = model["weights"]

    probs = []

    for (model_type, model_path), w in zip(models_list, weights):
        model_obj = load_model(model_path)

        # get probability
        p = model_obj.predict_proba(X)[0][1]
        probs.append(p * w)

    y_prob = float(sum(probs))
    y_class = 1 if y_prob >= 0.5 else 0
else:
    # normal model
    y_prob = float(model.predict_proba(X)[0][1])
    y_class = 1 if y_prob >= 0.5 else 0
```

```
    # save prediction input
    inp = PredictionInput(
        data=json.dumps(data),
        created_at=datetime.now(UTC)
    )
    db.session.add(inp)
    db.session.commit()

    # save prediction
    pred = Prediction(
        user_id=current_user.id,
        model_id=model_entry.id,
        input_id=inp.id,
        result=y_class,
        probability=y_prob,
        created_at=datetime.now(UTC)
    )
    db.session.add(pred)
    db.session.commit()

    # activity log
    log = ActivityLog(
        user_id=current_user.id,
        action=f"prediction:{pred.id}",
        timestamp=datetime.now(UTC)
    )
    db.session.add(log)
    db.session.commit()

    # redirect to details page
    return redirect(url_for("prediction.details", prediction_id=pred.id))


@prediction.route("/prediction/<int:prediction_id>")
@login_required
def details(prediction_id):
    pred = Prediction.query.get_or_404(prediction_id)

    # restrict access to user's predictions
    if pred.user_id != current_user.id:
        flash("Access denied.", "error")
        return redirect(url_for("prediction.history"))

    input_data = json.loads(pred.input.data)
```

```
    return render_template(
        "prediction_details.html",
        pred=pred,
        input_data=input_data,
        models=Model.query.all()
    )

@prediction.route("/history")
@login_required
def history():
    preds = Prediction.query.filter_by(
        user_id=current_user.id
    ).order_by(Prediction.created_at.desc())

    return render_template("history.html", preds=preds)

@prediction.route("/delete/<int:prediction_id>", methods=["POST"])
@login_required
def delete_prediction(prediction_id):
    pred = Prediction.query.get_or_404(prediction_id)

    # only in user's possesion
    if pred.user_id != current_user.id:
        flash("Access denied.", "error")
        return redirect(url_for("prediction.history"))

    # delete prediction input
    inp = pred.input
    db.session.delete(pred)

    if inp:
        db.session.delete(inp)

    # activity log
    log = ActivityLog(
        user_id=current_user.id,
        action=f"delete_prediction:{prediction_id}",
    )
    db.session.add(log)

    db.session.commit()

    return redirect(url_for("prediction.history"))
```

ДОДАТОК В

Слайди презентації

Національний університет «Запорізька політехніка»

Дипломна кваліфікаційна робота магістра

Розробка та дослідження системи прогнозування ризиків захворювань на основі методів машинного навчання

Development and Research of a Disease Risk Prediction System Based on Machine Learning Methods

Розробив: ст. гр. КНТ-214м — Андрій ІНДИК

Керівник: доцент, д-р.,філос. — Сергій ЛЕОЩЕНКО

Рисунок В.1 – Слайд 1

Об'єкт дослідження – процес прогнозування захворювань методами машинного навчання.

Предмет дослідження – методи та засоби для прогнозування ризиків захворювань.

Мета роботи – підвищення точності прогнозування захворювань методами машинного навчання.

Етапи роботи:

- аналіз проблеми та постановка завдань дослідження;
- розробка архітектури програми;
- розробка програми;
- експлуатація, тестування та експериментальне дослідження програми.

2

Рисунок В.2 – Слайд 2

Огляд існуючих рішень

Система	Автор	Моделі	Точність	Обмеження
Прогнозування діабету	Крішнамурті та ін.	LR, KNN, SVM, RF	~83%	використано лише структуровані дані; не враховано розширені клінічні фактори
Прогнозування діабету 2 типу	Деберне та Кім	LR, RF, SVM, XGBoost, ансамблі	до 73%	модель перевірена лише на одному наборі електронних медичних записів
Прогнозування раку	Сіра та Лім	гібрид RF, DT та min-max нейронної мережі	98,84%	специфічність набору даних; складність моделі
Виявлення серцевих захворювань	Дангаре та ін.	NB, DT, нейронні мережі	90,74%	використовуються два різні набори даних; можливий вплив різниці у збірці даних
Комбінований метод для діагностики серцевих патологій	Шуман, Тернер та Стокер	кластеризація K-means та NB	84,5%	чутливість до вибору початкових центроїдів; нестабільність у складних просторах ознак

3

Рисунок В.3 – Слайд 3

Методи машинного навчання

Метод	Переваги	Недоліки
Логістична регресія	Простота інтерпретації. Ефективність для бінарної класифікації. Невисокі вимоги до обчислювальних ресурсів	Обмежена здатність моделювати складні нелінійні залежності. Чутливість до викидів. Потреба у відсутності сильної взаємної кореляції між ознаками
XGBoost	Висока точність і швидкість навчання. Підтримує паралельне обчислення	Вимагає більше обчислювальних ресурсів. Може бути складним у налаштуванні
Random Forest	Висока точність навіть з поганими даними. Добре узагальнюється на нових даних. Інтерпретованість. Масштабується до великих наборів	Обчислювальна складність. Чутливість до гіперпараметрів. Довге навчання
SVM	Високовимірний простір для введення. Мало нерелевантних ознак	Збір даних є часозатратним
NB	Простий і зрозумілий	Очікує менший навчальний набір. Припускає незалежність ознак

4

Рисунок В.4 – Слайд 4

Вибір фреймворку

Особливість	Django	Flask	Streamlit
Архітектурний підхід	Монолітний	Диверсифікований	Односторінковий інтерфейс
Сфера застосування	Корпоративні системи	Мікросервіси, прототипи	Демонстрація роботи з даними
Гнучкість	Вбудовані рішення	Підбір власних рішень	Своя вузька ніша
Вбудовані інструменти	Об'єктно-реляційне відображення, автентифікація, форми, шаблони, адмін-панель	Мінімальна маршрутизація та шаблони. Інший функціонал через бібліотеки	Віджети, перезавантаження в реальному часі, візуалізація
Масштабованість	Ідеальна	Потребує уваги до вибраних бібліотек	Не масштабується
Обмеження	Закритий стандарт і архітектура	Відсутність готових рішень	Не підходить для складної логіки

5

Рисунок В.5 – Слайд 5

Вибір системи керування базами даних

Особливість	SQLite	MySQL	PostgreSQL
Архітектура	Файлова	Клієнт-сервер	Клієнт-сервер
Реплікація	Відсутня	Лідер-репліка, лідер-лідер	Лідер-репліка
Хмарні сервіси	Відсутні	Amazon, Microsoft, Google	Amazon, Microsoft, Google
Паралелізм	Обмежений	Підтримує багато з'єднань	Використовує окремі процеси з'єднань
Масштабованість	Невеликі застосунки	Підтримує масштабування на рівні сервера	Підходить для великих навантажень
Використання	Низький трафік, тестування, розробка	Вебсайти, вебзастосунки, онлайн-транзакції	Аналітика, добування даних

6

Рисунок В.6 – Слайд 6

Рисунок В.7 – Слайд 7

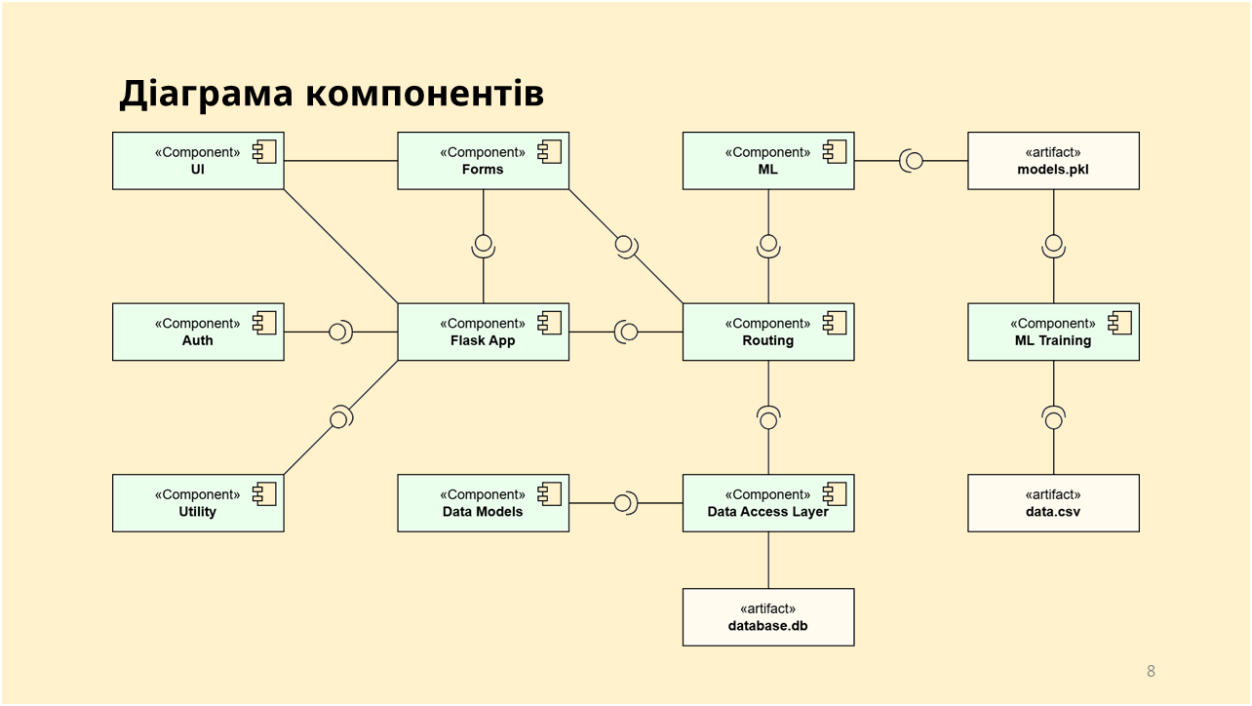
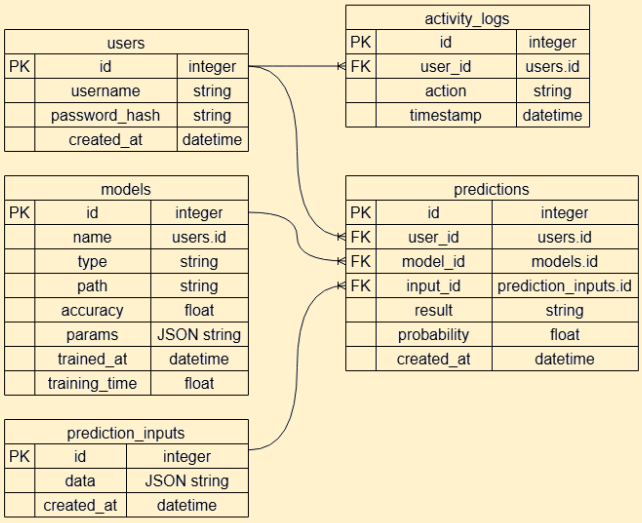


Рисунок В.8 – Слайд 8

Діаграма сутностей та зв’язків бази даних



9

Рисунок В.9 – Слайд 9

Порівняння метрик моделей

Модель	Accuracy	Precision	Recall	F1-score	FPR	AUC ROC	AUC PRC
SVM	0.786	0.841	0.799	0.819	0.000	0.877	0.925
NB	0.874	0.977	0.811	0.887	0.029	0.968	0.976
RF	0.973	0.981	0.975	0.978	0.000	0.984	0.990
XGB	0.981	0.981	0.987	0.984	0.019	0.980	0.987
LR	0.813	0.877	0.805	0.839	0.010	0.905	0.935
Ensemble	0.969	0.981	0.969	0.975	0.000	0.980	0.984

10

Рисунок В.10 – Слайд 10

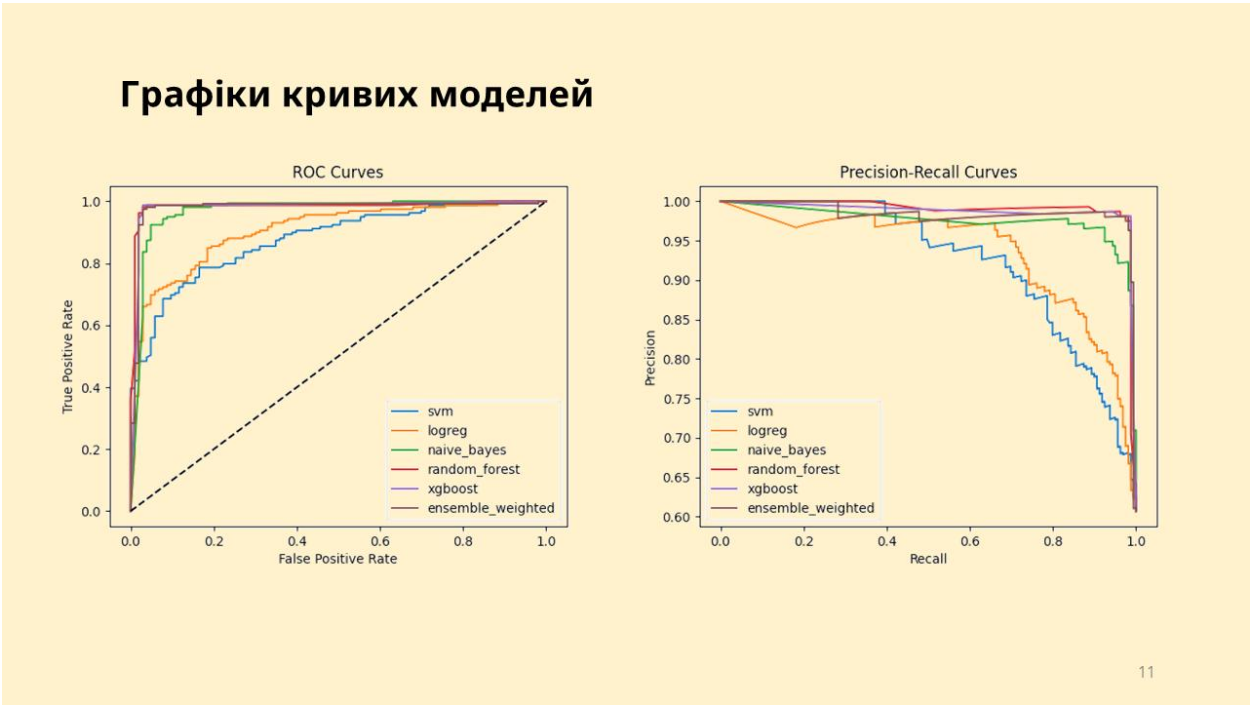


Рисунок В.11 – Слайд 11

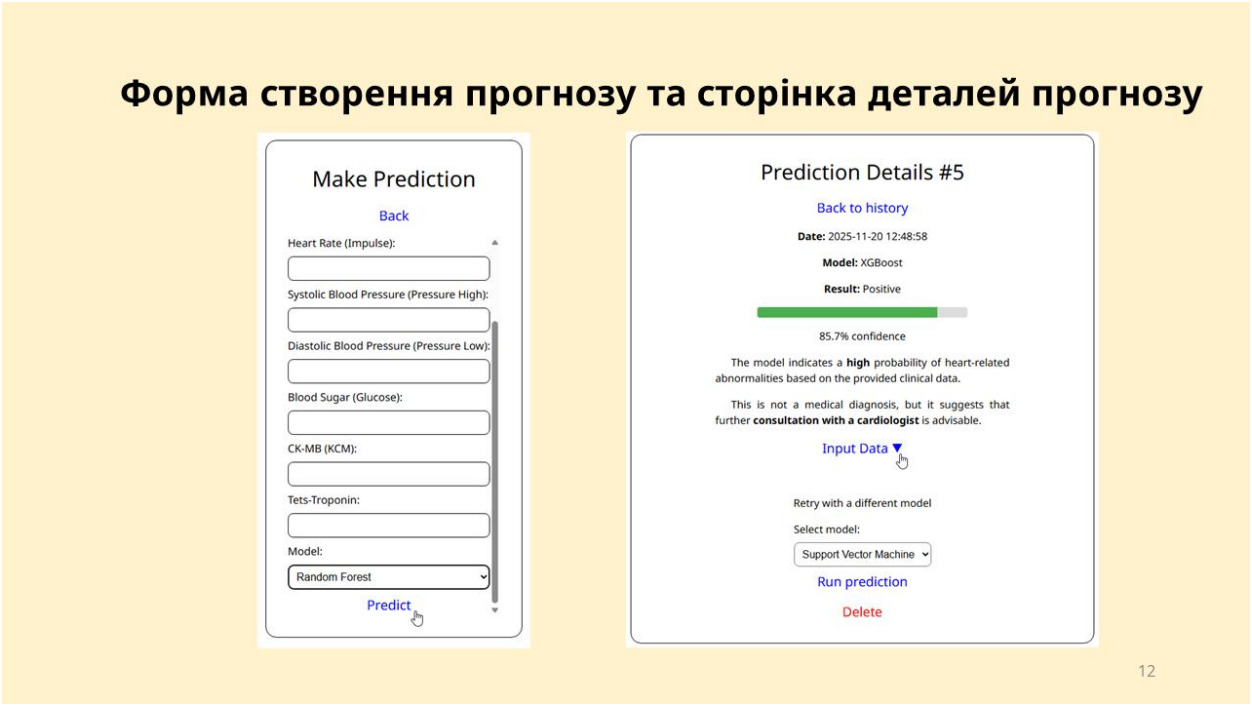
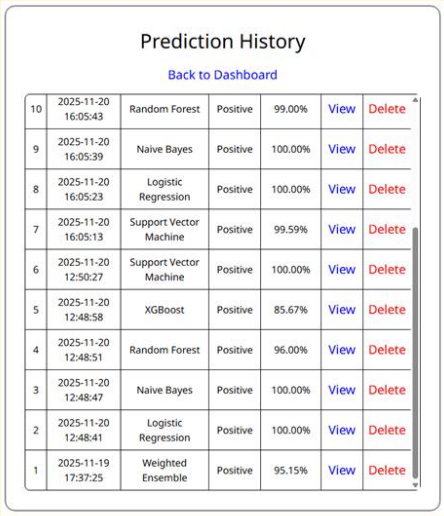


Рисунок В.12 – Слайд 12

Сторінка історії прогнозів користувача



13

Рисунок В.13 – Слайд 13

Чекліст тестування функціоналу та кросбраузерності

Параметр	Chromium	Gecko
Проходження реєстрації з валідними даними	Пройдено	Пройдено
Обробка помилок при реєстрації з невалідними даними	Пройдено	Пройдено
Авторизація з правильними обліковими даними	Пройдено	Пройдено
Обробка хибних комбінацій при авторизації	Пройдено	Пройдено
Автоматичне перенаправлення неавторизованого користувача на сторінку логіну	Пройдено	Пройдено
Перенаправлення на головну сторінку після авторизації	Пройдено	Пройдено
Перехід до сторінки створення прогнозу	Пройдено	Пройдено
Робота форми введення медичних показників та її валідації	Пройдено	Пройдено
Зміна моделі прогнозування зі списку доступних	Пройдено	Пройдено
Формування прогнозу після відправлення форми	Пройдено	Пройдено
Перезапуск прогнозу з іншою моделлю з тієї ж сторінки	Пройдено	Пройдено
Перехід до історії прогнозів	Пройдено	Пройдено
Коректне відображення прогнозів у хронологічному порядку	Пройдено	Пройдено
Перегляд деталей існуючого прогнозу зі сторінки історії	Пройдено	Пройдено
Видалення прогнозу і коректне видалення пов'язаних вхідних даних із БД	Пройдено	Пройдено
Запис подій у журнал активності	Пройдено	Пройдено
Робота кнопки виходу з облікового запису	Пройдено	Пройдено
Відсутність помилок рендерингу та скриптів у консолі браузера	Пройдено	Пройдено

14

Рисунок В.14 – Слайд 14

Висновки

У процесі виконання дипломної кваліфікаційної роботи магістра було проведено дослідження методів машинного навчання для прогнозування ризиків захворювань на основі медичних показників. Проаналізовано наукові підходи, визначено ключові методи класифікації та особливості їх застосування у медичних інформаційних системах.

Функціональну частину розробки реалізовано на основі Python, Flask та SQLite. Створено модулі автентифікації, введення й оброблення медичних даних, формування та зберігання прогнозів, а також модуль тренування моделей машинного навчання. У систему інтегровано кілька алгоритмів, зокрема SVM, Naïve Bayes, Random Forest, Logistic Regression, XGBoost та зважений ансамбль.

У результаті створено вебзастосунок, що забезпечує введення даних, автоматичне прогнозування та перегляд історії результатів. Проведене експериментальне дослідження показало, що ансамблевий підхід забезпечує стабільність роботи системи та може зменшувати коливання якості прогнозів у випадках, коли окремі моделі демонструють нерівномірну ефективність.

15

Рисунок В.15 – Слайд 15