

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ДІАГНОСТУВАННЯ СЕРЦЕВИХ ХВОРОБ
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR
DIAGNOSIS OF HEART DISEASES

Виконав: студент(ка) 2 курсу, групи КНТ-120м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Мяло О.Є.

(прізвище та ініціали)

Керівник Льовкін В.М.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

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

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Мяла Олексія Євгеновича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів
діагностування серцевих хвороб
Research and Software Implementation of Methods for Diagnosis of Heart Diseases

керівник проєкту (роботи) Льовкін Валерій Миколайович, к.т.н., доцент
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “5” листопада 2021 року №435

2. Строк подання студентом проєкту (роботи) 10 грудня 2021 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. Проєктування програмного забезпечення. 4. Основні
рішення щодо реалізації компонентів системи. 5. 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
1-5 Основна частина	Льовкін В.М., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “ 3 ” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз та дослідження предметної області	1-2 тиждень	
2	Розробка та затвердження технічного завдання	3 тиждень	Технічне завдання
3	Дослідження методів розв’язання проблеми	4 тиждень	
4	Проектування програмного забезпечення	5-6 тиждень	
5	Розроблення модулів програми	7-8 тиждень	
6	Розроблення програмного коду	9-10 тиждень	Текст програми
7	Тестування та відлагодження програмного забезпечення	11 тиждень	Програма
8	Розробка розділів пояснювальної записки	12-14 тиждень	Пояснювальна записка
9	Розробка графічного матеріалу	15 тиждень	Графічний матеріал
10	Захист магістерської роботи	16 тиждень	

Студент(ка)

(підпис)

Мяло О.Є.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Льовкін В.М.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 95 с., 22 рис., 3 табл., 3 дод., 22 джерела.

МЕДИЧНЕ ДІАГНОСТУВАННЯ, СЕРЦЕВА НЕДОСТАТНІСТЬ, НЕЙРОННА МЕРЕЖА, ЗАДАЧА КЛАСИФІКАЦІЇ, ВЕБЗАСТОСУНОК.

Об'єкт дослідження – процес діагностування серцевих хвороб. Предмет дослідження – методи діагностування серцевої недостатності. Мета роботи – розробити програмне забезпечення діагностування серцевої недостатності для забезпечення підтримки роботи кардіолога.

Матеріали, методи та технічні засоби: мови програмування C#, Python, JavaScript, мова розмітки HTML, мова таблиць стилів CSS, фреймворк ASP.NET Core, система керування базами даних Microsoft SQL Server, нейромережна бібліотека Keras.

Результати. Проведено аналіз проблеми діагностування серцевих хвороб, створено власний метод розв'язання проблеми діагностування серцевої недостатності, виконано проєктування програми з визначенням засобів розробки, вимог до програми, проєктуванням структури бази даних, визначенням структури програми, виконано програмну реалізацію, проведено експериментальне дослідження проблеми.

Наукова новизна роботи полягає в запропонованому методі діагностування серцевої недостатності, який характеризується виділенням двох моделей класифікації на основі структурованих та текстових даних медичних карток пацієнтів, що дозволяє виносити рішення за відсутності одного з типів даних або виносити узгоджене рішення, сприяючи підвищенню обґрунтованості діагностичних рішень.

Висновки. Розроблена програма дозволяє підтримувати діяльність кардіологічного центру з акцентом на роботі спеціаліста-кардіолога.

Галузь використання – кардіологічні центри.

ABSTRACT

Explanatory note to the qualifying work of the master: 95 pages, 22 figures, 3 tables, 3 appendixes, 22 sources.

MEDICAL DIAGNOSIS, HEART FAILURE, NEURAL NETWORK, CLASSIFICATION PROBLEM, WEB APPLICATION.

The object of research is the process of diagnosing heart disease. The subject of research is methods of diagnosing heart failure. The purpose of the work is to develop software to support the work of a cardiologist to provide tools for diagnosing heart failure.

Materials, methods and technical means: C#, Python, JavaScript programming languages, HTML markup language, CSS style sheet language, ASP.NET Core framework, Microsoft SQL Server database management system, Keras neural network library.

Results. The analysis of the problem of diagnosing heart disease was carried out, the own method of solving the problem of diagnosing heart failure was created, the software was designed with the definition of means of development, program requirements, database structure design, software architecture and structure, software implementation was carried out.

The scientific novelty of the work lies in the proposed method of diagnosing heart failure, which is characterized by the allocation of two models of classification based on structured and textual data of medical records of patients, which allows to make an agreed decision, helping to increase the validity of diagnostic decisions.

Conclusions. The developed software allows users to support the activities of the cardiology center, focusing on the work of the cardiologist.

Field of application is cardiac centers.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	11
1.1 Аналіз проблеми діагностування серцевих хвороб	11
1.2 Огляд програмних засобів підтримки роботи кардіолога	14
1.3 Висновки за розділом 1 та постановка завдань дослідження.....	17
2 Матеріали і методи.....	18
2.1 Архітектура нейронної мережі Long Short Term Memory	18
2.2 Архітектура згорткової нейронної мережі	20
2.3 Гібридні архітектури нейронних мереж	21
2.4 Методи відбору ознак	22
2.5 Метод діагностування серцевої недостатності	23
2.6 Висновки за розділом 2	27
3 Проєктування програмного забезпечення	28
3.1 Вибір мови програмування	28
3.2 Визначення функціональних вимог до програми	29
3.3 Визначення структури бази даних	31
3.4 Структура програмного проєкту	40
3.5 Висновки за розділом 3	40
4 Основні рішення щодо реалізації компонентів системи.....	42
4.1 Алгоритм функціонування програми.....	42
4.2 Опис модулів реалізованої програмної системи.....	43
4.3 Висновки за розділом 4	46
5 Експлуатація, тестування та експериментальне дослідження програми	47
5.1 Призначення програми	47
5.2 Умови виконання програми	47
5.3 Робота з програмою	47

5.4 Експериментальне дослідження проблеми діагностування серцевої недостатності.....	50
5.5 Висновки за розділом 5	51
Висновки	52
Перелік джерел посилання	53
Додаток А Технічне завдання	56
Додаток Б Текст програми	60
Додаток В Слайди презентації.....	88

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN – convolutional neural network;

LASSO – least absolute shrinkage selection operator;

LSTM – long short term memory;

ЕКГ – електрокардіограма;

ЕМК – електронна медична картка;

ПЗ – програмне забезпечення.

ВСТУП

За даними Центру контролю та профілактики захворювань, понад 6 мільйонів дорослих у Сполучених Штатах Америки мають серцеву недостатність [1]. У той же час за даними міжнародного дослідження Global Burden of Disease серцеві хвороби є одними з основних причин смертей у всьому світі, до того ж за останні 30 років продемонстрували значну динаміку і подвоїлись, хоча при порівнянні з даними в Україні є певні відмінності [2]-[3]. Так, якщо в Сполучених Штатах Америки серцеві хвороби є причиною 23.4 % від всіх смертей, то в Україні цілих 64.3 % смертей викликані серцевими захворюваннями, хоча динаміка за ті ж самі 30 років складає тільки 8 %.

В даний час серцева недостатність є серйозним захворюванням, яке не піддається лікуванню: однак таке лікування, як зміна здорового способу життя, ліки, деякі пристрої та процедури, можуть допомогти багатьом людям підвищити якість життя [1].

Усе це говорить про важливість прийняття рішень у даній сфері людського життя. На даний період часу активно розвиваються різноманітні програмні системи, інтелектуальні методи, які дозволяють перевести підтримку людських рішень щодо їх обґрунтованості, зменшення суб'єктивності на новий рівень. Проте незважаючи на такий активний розвиток, далеко не всі завдання у цій сфері вирішені на даний момент. Це пов'язано з тим, що зазвичай обсяг факторів, який можна враховувати при прийнятті рішень у даній сфері є великим, часто немає достатньої кількості даних, до того ж активний розвиток інтелектуальних технологій призводить до відкриття задач заново і їх повторного вирішення знову і знову, адже абсолютно ідеальні результати ще не були досягнуті, щоб у сфері охорони здоров'я, де підвищення точності прийняття рішень на десять відсотка вартує життів людей, припинити проведення таких досліджень.

Програмне забезпечення, яке розробляється в даній роботі, повинно, по-перше, підвищити зручність роботи з даними щодо серцевих хвороб для спеціалістів у цій сфері – кардіологів, а по-друге, реалізувати відповідний метод, направлений на роботу з пацієнтами, які мають або потенційно можуть мати серцеві захворювання.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз проблеми діагностування серцевих хвороб

Серцева недостатність – це стан, який розвивається, коли серце не перекачує достатньо крові для потреб організму: це може статися, якщо серце не може наповнитися достатньою кількістю крові, це також може статися, коли серце занадто слабе, щоб правильно качати кров, при цьому термін «серцева недостатність» не означає, що серце зупинилося, однак серцева недостатність є серйозним станом, який потребує медичної допомоги [1], [4].

Серцева недостатність може розвинути раптово (гостра форма) або з часом, коли серце стає слабкішим (хронічний тип), здійснюючи вплив на одну або обидві сторони серця:

- лівостороння та правостороння серцева недостатність можуть мати різні причини, найчастіше серцева недостатність викликана іншим захворюванням, яке пошкоджує серце, включаючи ішемічну хворобу серця, запалення серця, високий кров'яний тиск, кардіоміопатію або нерегулярне серцебиття, серцева недостатність може викликати симптоми не відразу, але в кінцевому підсумку можна відчувати втому і задишку, а також помітити накопичення рідини в нижній частині тіла, навколо живота або шиї;

- серцева недостатність може пошкодити печінку або нирки, а інші ускладнення включають легеневу гіпертензію або інші серцеві захворювання, такі як нерегулярне серцебиття, захворювання серцевих клапанів і раптова зупинка серця;

- лікар діагностує серцеву недостатність на основі медичної та сімейної історії, фізичного огляду та результатів візуалізації та аналізів крові [1].

Для забезпечення роботи ПЗ підтримки діагностування серцевих хвороб важливо визначити детально, як саме відбувається діагностування, щоб зрозуміти, які саме інструменти знадобляться лікарю.

Щоб діагностувати серцеву недостатність, лікар ретельно збирає історію хвороби, переглядає симптоми та проводить фізичний огляд, також перевіряє, чи є у людини фактори ризику серцевої недостатності (високий кров'яний тиск, ішемічна хвороба серця або цукровий діабет), також лікар може прослухати легені на ознаки скупчення рідини (застійні явища в легенях), а серце – на предмет свистячих звуків (шумів), які можуть свідчити про серцеву недостатність, лікар може оглянути вени на шиї та перевірити наявність рідини в животі та ногах, після фізичного огляду лікар може також призначити деякі з цих тестів:

- аналізи крові: аналізи крові проводяться для виявлення ознак захворювань, які можуть вплинути на серце:

- рентген грудної клітки: рентгенівські знімки можуть показати стан легенів і серця;

- електрокардіограма (ЕКГ): цей швидкий і безболісний тест реєструє електричні сигнали в серці, він може показувати час і тривалість серцевих скорочень;

- ехокардіограма: звукові хвилі використовуються для створення зображень серця в русі, цей тест показує розмір і структуру серця і серцевих клапанів, а також потік крові через серце, для вимірювання фракції викиду можна використовувати ехокардіограму, яка показує, наскільки добре серце перекачує кров, і допомагає класифікувати серцеву недостатність і керувати лікуванням;

- стрес-тест: стрес-тести вимірюють здоров'я серця під час фізичної активності, тому пацієнта можуть попросити походити на біговій доріжці, коли він підключений до ЕКГ-апарату, або може отримати внутрішньовенний препарат, який стимулює вплив фізичних вправ на серце, іноді під час носіння маски проводять стрес-тест, який визначає, наскільки добре серце і легені отримують кисень і видихають вуглекислий газ;

- комп'ютерна томографія серця (КТ): під час комп'ютерної томографії серця пацієнт лежить на столі всередині апарата в формі пончика,

рентгенівська трубка всередині апарата обертається навколо його тіла та збирає зображення серця та грудної клітини, іноді надається контраст (оскільки контрастна речовина може вплинути на функцію нирок, потрібно окремо звернутися до лікаря, якщо у пацієнта є проблеми з нирками);

- магнітно-резонансна томографія: під час магнітно-резонансної томографії серця пацієнт лежить на столі всередині довгого, схожого на трубку апарата, радіохвилі створюють образи серця, магнітно-резонансну томографію серця можна зробити з барвником (контрастним), знову ж таки повідомлення про хвороби нирок важливе;

- коронарна ангіограма: у цьому тесті тонка гнучка трубка (катетер) вводиться в кровоносну судину, зазвичай в паху, і направляється до серцевих артерій, через катетер вводиться барвник (контраст), щоб артерії були більш чіткими на рентгенівському знімку, допомагаючи лікарю виявити закупорку;

- біопсія міокарда: під час цього тесту лікар вводить невеликий гнучкий канатик у вену на шиї або паху і видаляє дуже маленькі шматочки серцевого м'яза для дослідження, цей тест можна зробити для діагностики певних типів захворювань серцевого м'яза, які спричиняють серцеву недостатність [4]-[6].

За класифікацією Нью-Йоркської кардіологічної асоціації виділяють 4 класи серцевої недостатності:

- серцева недостатність I класу: симптомів серцевої недостатності немає;

- серцева недостатність II класу: повсякденну діяльність можна виконувати без труднощів, але напруження викликає задишку або втому;

- серцева недостатність III класу: важко виконувати повсякденні справи;

- серцева недостатність IV класу: задишка виникає навіть у стані спокою, до цієї категорії відносять найбільш важку серцеву недостатність [5].

1.2 Огляд програмних засобів підтримки роботи кардіолога

У медичній сфері для роботи з пацієнтами на даний момент зазвичай використовуються системи, побудовані в основі за принципом електронної медичної картки (ЕМК).

Однак у даному випадку ПЗ розглядається з точки зору спеціаліста з серцевих захворювань – кардіолога. Зазвичай кардіологи також використовують загальні системи на основі ЕМК. Проте існує невелика кількість і спеціалізованих засобів, які процес роботи з пацієнтами роблять більш специфічним стосовно термінів і засобів з предметної області, розглянутої в попередньому підрозділі.

Огляд необхідно розпочати з ПЗ, яке застосовується для лікарів різних спеціальностей, у тому числі кардіологів.

Системи ЕМК містять історію хвороби пацієнта та клінічні записи, вони також можуть включати такі функції, як планування, діаграми та портали пацієнтів [7].

Praxis – це настроювана хмарна або серверна система ЕМК, містить дослідницький інструмент DataMiner, який дає змогу миттєво запитувати будь-яку інформацію від вибраної популяції пацієнтів, значну увагу оптимізації досвіду пацієнтів та покращенню якості надання медичної допомоги, при цьому не використовує шаблонів, дозволяючи користувачам створювати потрібні їм форми [7].

Наступні програмні засоби є спеціалізованими для кардіологів.

PracticeStudio – це сертифіковане рішення ЕМК, яке надає універсальне ПЗ для кардіологів, щодо ПЗ Cardiology Electronic Health Records від PracticeStudio можна вказати наступне: має інтегроване керуванням практикою, портал для пацієнтів, інтерфейси лабораторій, персоналізоване резервне копіювання даних і можливості звітності, система достатньо проста у використанні, враховані засоби підтримки відвідування нового пацієнта, ехокардіограми, ЕКГ, стрес-тестів, ультразвукового

дослідження сонних судин, дискомфорту в грудях, гіпертонії, застійної серцевої недостатності, процедура фіксування кардіостимуляторів, захворювань аортального клапана, серцевих катетерів, також надаються можливості кардіологам виконувати свою роботу за допомогою взаємопов'язаного планування прийомів, обміну повідомленнями, виставлення рахунків, звітності, статистики практики, відстеження пацієнтів тощо [8].

Щодо спеціалізованих програмних засобів потрібно виділити загалом наступні принципи:

- картка пацієнта повинна містити такі основні дані, як історія перенесених та хронічних кардіозахворювань (стенокардія, аритмія, інфаркт міокарда, атеросклероз), перенесені операції на серці, наявність кардіостимулятора, а довідники програми повинні включати відповідні списки захворювань і процедур;

- журнал пацієнта повинен підтримувати додавання та перегляд зображень (кардіограм, знімків ультразвукової діагностики), включати графічні трекери для моніторингу ваги, тиску, пульсу, бажано мати у програмі спеціалізовані під кардіологію електронні протоколи оглядів, опитувальники пацієнтів та шаблони для складання епікризу;

- знімки та кардіограми повинні зберігатися в ЕМК [9].

SoftCath Lab є одним зі спеціалізованих засобів, що включає наступні можливості:

- комплексна ЕМК, розроблена з тісно інтегрованими моделями серцево-судинної діагностики для огляду пацієнта, існує багато елементів даних для підтримки клінічних рішень, включаючи демографічні дані пацієнтів, лабораторні звіти, клінічну історію та діагностичні тести, що включають алергію та ліки;

- основні характеристики комплексної ЕМК включають: клінічні панелі приладів, рецепти, адміністративну панель, лабораторну інтеграцію,

портал пацієнта, звітність на основі шаблонів, консультації з обміном повідомленнями;

- розширений робочий процес: доступ до клінічних записів, які містять сигнали, зображення та процедурну інформацію на єдиній платформі, створює централізовану панель інструментів для звітів лабораторії, завдяки сховищу даних пацієнтів і цифровій системі для керування лікарі можуть краще керувати робочими процесами лабораторії;

- електронний запис пацієнта;

- користувачі можуть отримати доступ до масиву зображень і приймати рішення на основі інформації, що надходить у систему;

- доступ і видимість: можливості перегляду вебсторінок ПЗ для звітності роблять роботу робочої станції зручною для користувачів і дозволяють виконувати завдання віддалено, вебдоступ також дозволяє кардіологам готувати та обмінюватися звітами через Інтернет;

- програма побудована з простими у використанні функціями, які враховують регулярність звітності даних: інтерфейс пропонує користувачеві збирати, керувати та аналізувати кардіологічні дані в декількох форматах;

- за допомогою інструменту прийняття рішень користувачі отримують доступ до даних і таким чином отримують можливість приймати обґрунтовані рішення;

- загалом цифровий спосіб роботи з ПЗ усуває ризик втрати даних пацієнтів, історії хвороби, а також ймовірність помилок: під час створення цифрового звіту можна вибрати з кількох попередньо визначених параметрів, тому ймовірність помилок менша [10].

Результати проведеного аналізу зведено до табл. 1.1.

Головною відмінністю системи, що розробляється, є наявність засобів підтримки прийняття діагностуючих рішень, що має бути виконана на основі створеного в роботі метода. Окрім того це спеціалізована система для кардіологів, тому повинна мати засоби внесення специфічних результатів їх обстежень.

Таблиця 1.1 – Порівняння розроблюваної системи та аналогів

Критерій	Praxis	CathLab	Розроблювана система
Внесення результатів обстеження пацієнтів	+	+	+
Внесення історії хвороби	+	+	+
Наявність спеціалізованих засобів протоколів кардіологів	—	+	+
Підтримка прийняття діагностуючих рішень	—	—	+

1.3 Висновки за розділом 1 та постановка завдань дослідження

Виконаний аналіз проблеми діагностування серцевих хвороб дозволив виявити основні поняття даної галузі, особливості процесу виявлення серцевих хвороб щодо аналізів, які можуть при цьому призначатися, особливості діагностування серцевих хвороб, а також класи серцевої недостатності.

Завдання кваліфікаційної роботи магістра були визначені як наступні:

- аналіз предметної області, що було виконано в даному розділі;
- дослідження методів діагностування серцевої недостатності;
- розробка технічного завдання і вибір засобів розробки;
- проєктування програмної системи;
- розробка програмної системи діагностування серцевої недостатності.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Архітектура нейронної мережі Long Short Term Memory

Long Short Term Memory (LSTM) – це один з варіантів організації повторюваної нейронна мережа, адже в рекурентних нейронних мережах, до яких і належить LSTM, вихід з останньої фази розглядається як вхідний параметр для поточної фази, при цьому загалом відносно LSTM можна визначити наступне:

- архітектура була створена для розв'язання питання довгострокових залежностей рекурентних нейронних мереж, в яких рекурентна нейронна мережа не може передбачити слово, що зберігається в довгостроковій пам'яті, однак може дати більш точні прогнози на основі нових даних;

- оскільки довжина розриву збільшується, рекурентна нейронна мережа не може забезпечити ефективну продуктивність, LSTM у свою чергу може зберігати дані протягом тривалого часу, використовується для підготовки, класифікації та прогнозування на основі інформації часових рядів, LSTM має ланцюгову структуру, яка містить чотири специфічні блоки пам'яті і шлюзи;

- блоки відповідають за збереження інформації: блок пам'яті LSTM використовує змінну, яка називається станом, що оновлюється на кожному кроці часу з використанням ряду шлюзів, щоб виділити залежності на великій відстані, уникаючи проблеми зникаючого градієнта традиційних рекурентних нейронних мереж;

- шлюзи відповідають за маніпуляції з пам'яттю;

- кількість разів, коли комірка пам'яті повторюється, дорівнює кількості минулих часових кроків, використаних для прогнозування значення на один крок вперед, тобто розміру вікна;

- на кожному кроці часу блок пам'яті приймає як вхід вектор станів з попереднього кроку часу, вектор активацій з попереднього кроку часу і матрицю вхідних даних;

- довжини векторів станів та активації дорівнюють кількості нейронів у комірці пам'яті;
- форма матриці вхідних даних дорівнює кількості предикторів, помноженій на кількість вікон через часовий ряд;
- шлюз забуття гарантує, що інформація, яка більше не є корисною, буде видалена;
- шлюз оновлення відповідає за додавання корисної інформації до клітинки;
- вихідний шлюз використовується для вилучення корисної інформації з комірок;
- під час обчислень визначається також стан-кандидат, який є кандидатом на заміну значенню, отриманого з попереднього кроку ;
- після визначення стану блоку-кандидата розраховуються шлюз оновлення і шлюз забуття, щоб вирішити, які елементи вектора стану слід оновити від їхнього попереднього значення до відповідного значення кандидата: шлюзи оновлення та забуття є векторами довжини, рівній кількості нейронів блоку, кожен елемент можна вважати рівним 0 або 1 (відповідно визначають оновлення, збереження значення або додавання значення кандидата до його попереднього значення);
- навчання мережі LSTM відноситься до підгонки ваг і значень зміщення мережі і виконується ітераційно: кожна ітерація включає прямий прохід, обчислення втрат (помилка між виходом LSTM та експериментальним значенням) і зворотний прохід, який може розглядатися як зворотний прямому проходу для визначення внеску кожного параметра мережі до помилки, початковий прямий прохід мережі здійснюється шляхом випадкової ініціалізації всіх мережевих ваг;
- можна використовувати декілька функцій втрат, включаючи середню квадратичну помилку, середню абсолютну помилку і середню абсолютну відсоткову помилку [11]-[13].

2.2 Архітектура згорткової нейронної мережі

Згортка нейронна мережа або convolutional neural network (CNN) – це нейронна мережа, яка використовує згортку і використовує наступні поняття і принципи:

- згортка – це математична операція, яка дозволяє об'єднати два набори інформації: до вхідних даних застосовується згортка, щоб відфільтрувати інформацію та створити карту ознак, цей фільтр також називають ядром, або детектором ознак, і його розміри можуть бути, наприклад, 3×3 ;

- для виконання згортки ядро проходить над вхідним масивом, виконуючи множення матриці елемент за елементом, результат для кожного рецептивного поля (області, де відбувається згортка) записується на карті ознак;

- доповнення – це техніка, яка може використовуватися під час роботи нейронної мережі, додає фальшиві пікселі до меж матриці, це робиться тому, що згортка зменшує розмір матриці, наприклад, матриця 5×5 перетворюється на матрицю 3×3 , коли фільтр проходить через неї;

- часто буває, що при роботі зі згортковим шаром потрібно отримати вихід, менший за вхідний, одним із способів досягти цього є використання шару об'єднання, інший спосіб досягти цього – використовувати зсування: ідея кроку полягає в тому, щоб пропускати деякі області, коли ядро ковзає (наприклад, пропускати кожні 2 або 3 пікселі, це зменшує просторову роздільну здатність і робить мережу більш ефективною з точки зору обчислень);

- згортковий шар у згортковій нейронній мережі відповідає за розпізнавання ознак у пікселях: проходження процесу згортки генерує велику кількість даних, що ускладнює навчання нейронної мережі, щоб стиснути дані, нам потрібно пройти через об'єднання;

– шар об'єднання відповідає за те, щоб зробити ознаки більш абстрактними: шар об'єднання отримує результат від згорткового шару і стискає його, фільтр шару об'єднання завжди менший, ніж карта об'єктів, зазвичай він бере квадрат 2×2 і стискає його в одне значення, фільтр 2×2 зменшить кількість пікселів у кожній карті об'єктів до однієї чверті розміру, для об'єднання можна використовувати декілька різних функцій (максимальне об'єднання, що обчислює максимальне значення для кожного фрагмента карти об'єктів, середнє об'єднання, що обчислює середнє значення для кожного фрагмента на карті об'єктів), а після використання шару об'єднання отримуються об'єднані карти об'єктів, які є узагальненою версією об'єктів, виявлених у вхідних даних, рівень об'єднання покращує стабільність нейронної мережі (якщо раніше навіть найменші коливання пікселів призводили до неправильної класифікації моделі, то тепер невеликі зміни розташування об'єкта у вхідних даних, виявлених згортковим шаром, призведуть до об'єднаної карти об'єктів з об'єктом у те саме місце);

– повністю зв'язаний шар відповідає за використання отриманих ознак для передбачення: зведений вихід подається до нейронної мережі з прямим зв'язком, а зворотне поширення застосовується на кожній ітерації навчання, цей рівень надає моделі можливість остаточно зрозуміти зображення або вхідні дані [14]-[16].

2.3 Гібридні архітектури нейронних мереж

З двох розглянутих архітектур нейронних мереж створюються комбінації:

– комбінація архітектури CNN–LSTM включає початковий шар згортки, який відповідає за отримання вхідних даних для вбудовування слів, цей процес дає вихід, який об'єднується в менший розмір і, нарешті, подається в шар LSTM: основна ідея моделі полягає в тому, що локальні ознаки будуть витягнуті за допомогою згортки, так що LSTM

впорядкуватиме ознаки для розуміння впорядкування введеного тексту, під час навчання використовується техніка відключення – це метод ігнорування випадково вибраних нейронів під час навчання (відключення випадковим чином означає, що внесок нейронів в активацію нейронів, що знаходяться нижче, тимчасово припиняється під час прямого проходу, більше того, оновлення ваги більше не застосовуються до нейрона під час зворотного проходу), дана техніка використовується для запобігання перенавчанню моделі;

– модель LSTM–CNN є потужнішою порівняно з моделлю CNN–LSTM: архітектура LSTM–CNN включає початковий рівень LSTM, який відповідає за отримання вбудованих слів для кожного маркера в реченнях як вхідних даних, основна ідея полягає в тому, що вихідний токен міститиме більше інформації як для початкових токенів, так і для попередніх, рівень LSTM в цій моделі відповідає за створення нового кодування для вихідного входу, вихідні дані, згенеровані з шару LSTM, подаються в CNN, який оснащений засобом вилучення локальних ознак, вихідні дані цього шару згортки потім об'єднуються до меншого виміру, а кінцевий результат виходить у вигляді позитивної або негативної мітки [11].

2.4 Методи відбору ознак

Метою відбору ознак у машинному навчанні є пошук найкращого набору ознак, що дозволяє будувати моделі досліджуваних об'єктів, а методи відбору ознак можна розділити на такі категорії:

– методи фільтрації вибирають внутрішні властивості ознак, виміряні за допомогою одновимірної статистики: ці методи є швидшими та менш дорогими, ніж методи обгортки, під час роботи з даними великої розмірності використовувати методи фільтрації дешевше з обчислювальної точки зору;

– методи обгортки: обгортки вимагають певного методу для пошуку простору всіх можливих підмножин ознак, оцінки їх якості шляхом вивчення

та оцінки класифікатора з цією підмножиною ознак, процес вибору функцій заснований на конкретному алгоритмі машинного навчання, який намагаються вмістити в заданий набір даних, він дотримується підходу жадібного пошуку, оцінюючи всі можливі комбінації ознак за критерієм оцінки, методи обгортки зазвичай призводять до кращої точності прогнозування, ніж методи фільтрації;

– вбудовані методи: ці методи охоплюють переваги як методів обгортки, так і методів фільтрації, включаючи взаємодію ознак, а також підтримуючи розумну вартість обчислень, вбудовані методи є ітеративними в тому сенсі, що піклуються про кожну ітерацію процесу навчання моделі та ретельно виділяють ті функції, які найбільше сприяють навчанню для певної ітерації [17].

Одним з методів відбору ознак є регуляризація Least Absolute Shrinkage Selection Operator (LASSO), яка також називається L1 і полягає в додаванні штрафу до різних параметрів моделі машинного навчання, щоб зменшити свободу моделі, тобто уникнути надмірної підгонки: у регуляризації лінійної моделі штраф застосовується до коефіцієнтів, які перемножують кожен з предикторів, регуляризація LASSO має властивість, яка здатна зменшувати деякі коефіцієнти до нуля, у такому випадку ознаку можна видалити з моделі [17].

2.5 Метод діагностування серцевої недостатності

Метод діагностування серцевої недостатності концентрується на етапі раннього діагностування. Він направлений на те, щоб за результатами аналізів, історії хвороби, даних, внесених в ЕМК пацієнта, виявляти захворювання на серцеву недостатність і повідомляти про це пацієнта і відповідно лікаря, щоб він виконав подальше діагностування. На ранньому етапі цінною інформацією для характеристики пацієнта є також інформація, яка записана в його ЕМК. Це неструктурована інформація, яка включає його

скарги щодо стану здоров'я. Дана інформація також може бути розширена за рахунок опитування стану пацієнта про його стан здоров'я за можливості доступу пацієнта самостійно до системи. Проте в такому випадку варто враховувати, що якість такої інформації може бути дещо зниженою, адже пацієнт без участі лікаря не завжди може описати повністю адекватно свій стан здоров'я. Хоча для можливості попереднього діагностування, щоби виявити якомога швидше випадки наростання гострої ситуації, проінформувати пацієнта, щоб таким чином досягти з ним кращої комунікації, підказати можливості доступу до лікаря та провести врешті консультацію для усунення причин, які призводять до погіршення стану пацієнта, така інформація може бути корисною.

Для створення методу діагностування серцевої недостатності було виділено дві окремі моделі:

- перша модель працює над визначенням діагнозу на основі структурованих даних про пацієнта;
- друга модель працює над визначенням діагнозу на основі текстових записів ЕМК.

У такому випадку, якщо обидві моделі приходять до висновку щодо відповідного діагнозу, то виноситься рішення про діагностування серцевої недостатності. Це дозволяє виконувати певну підтримку діагностування, перевіряючи відповідність даних з двох джерел.

Підхід на основі двох моделей замість однієї, де завдання полягає в тому, щоб об'єднувати всередині різні шари, дозволяє досягти спеціалізації моделей для виконання відповідних завдань.

Окрім того такий підхід дозволяє використовувати одну модель без іншої. Тобто у випадку, якщо рішення потрібно винести все-таки використовуючи тільки структуровану інформацію, коли текстова інформація ЕМК пацієнтів, наприклад, недоступна (наприклад, така інформація не накопичується), це можна здійснити на основі однієї моделі і все-таки винести рішення про діагностування. Також у протилежному

випадку, коли недоступною є саме структурована інформація, а це може реалізуватися тоді, коли відбувається підтримка попередніх даних внесених пацієнтом самостійно, рішення також може бути винесене.

Структурованою інформацією про пацієнта зокрема є наявність в нього діабету, фракція викиду лівого шлуночка, систологічний тиск, індекс маси тіла тощо.

Загалом запропонований метод складається з послідовності наступних етапів.

Сформувати вибірку даних A , яка характеризує стан пацієнта за структурованими показниками, і вибірку даних B , яка характеризує стан пацієнта текстовими описами медичної картки.

Виконати передоброблення даних з вибірок A і B шляхом оброблення пропущених значень (використовуючи значення за замовчуванням, усереднені значення для структурованих даних, а для неструктурованих – виділяючи слова з загального тексту).

Виконати відбір ознак з вибірки A для створення відповідної моделі. Відбір ознак запропоновано виконувати на основі методу регуляризації LASSO.

Оскільки зазвичай кожен відповідний стан пацієнта описується великою множиною ознак, то потрібно визначити, які з них дійсно корисні саме для діагностування серцевої недостатності.

Створити модель M_1 на основі нейронної мережі прямого поширення для роботи зі структурованими даними на основі ознак, які були виділено на попередньому етапі.

Виконати навчання моделі M_1 на основі вибірки даних A .

Виконати обробку текстових даних для підтримки роботи відповідної моделі за допомогою методу skip-gram [18].

Створити модель M_2 на основі архітектури LSTM–CNN для роботи з текстовими описами ЕМК. Такими текстовими описами є зокрема скарги пацієнта, а також винесені з цього об’єктивні дані.

Виконати навчання створеної моделі M_2 на основі вибірки даних B .

Визначити стан пацієнта за відповідним звуженим набором ознак та текстовими описами. Допускається відсутність сукупності значень за першою частиною або за другою.

Виконати класифікацію стану пацієнта на основі моделей M_1 і M_2 , формуючи таким чином рішення 1 і 2.

Якщо рішення 1 і 2 співпадають, то винести відповідний діагноз як результат роботи моделі. Якщо рішення не співпадають, то винести рішення про необхідність детального опрацювання даних лікарем. Якщо для пацієнта не було заповнено структуровані (або навпаки неструктуровані дані), то рішення приймається на основі тільки однієї відповідної моделі.

На рис. 2.1 представлена відповідна послідовність етапів, що формують метод діагностування серцевої недостатності.

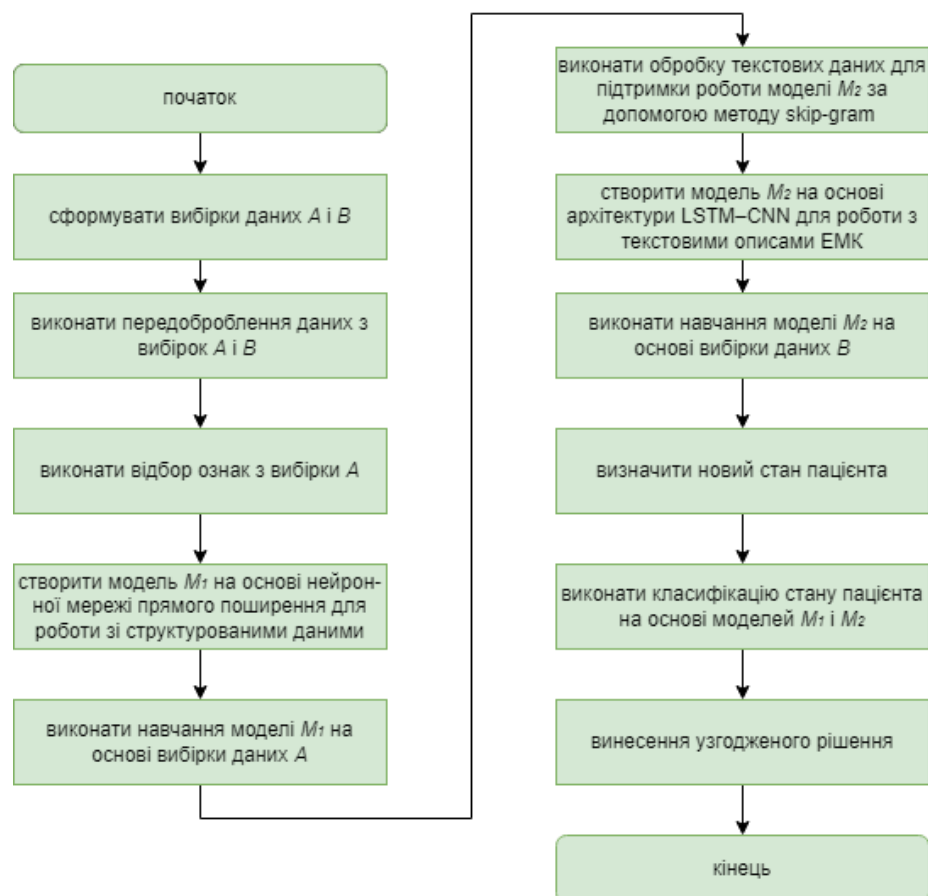


Рисунок 2.1 – Послідовність етапів роботи метода діагностування серцевої недостатності

2.6 Висновки за розділом 2

Виконано аналіз архітектур нейронних мереж, включаючи нейронні мережі архітектури LSTM, згорткові нейронні мережі й гібридні архітектури, та методів відбору ознак. Запропоновано власний метод діагностування серцевої недостатності, який складається з двох моделей, що забезпечують винесення узгодженого рішення на основі текстових описів ЕМК пацієнтів та структурованих даних, які описують стан пацієнта. В якості першої моделі використано багат шарову нейронну мережу прямого поширення сигналу, а в якості другої – нейронна мережа архітектури LSTM–CNN.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір мови програмування

ПЗ діагностування серцевих хвороб повинно забезпечувати доступ до роботи з даними для пацієнтів. Тобто є потенційна можливість роботи як з лікарні, так і віддалено з комп'ютерів користувачів, які є пацієнтами. Тому ПЗ має бути розроблено у вигляді вебзастосунку. Мовами веброзробки, які є придатними для створення великих проєктів, є мови C# і JavaScript [19]-[20].

У табл. 3.1 представлено результати вибору мови програмування для веброзробки.

Таблиця 3.1 – Вибір мови програмування для веброзробки

Критерій	C#	JavaScript
Фреймворк для серверної частини	ASP.NET Core	Node.js
Типізація	Статична	Динамічна
Кросплатформність	Так	Так
Підтримка об'єктно-орієнтованого програмування	Так	Так
Підтримка функціонального програмування	Так	Ні
Автоматичне збирання сміття	Так	Ні
Гнучкість	Висока	Менше
Продуктивність	Більша	Менша

Статична типізація мови C# – вдалий вибір для розробки, оскільки це дозволяє одразу відслідковувати помилки, не витрачаючи на це надто багато часу, окрім того мова C# дозволяє використовувати засоби функціонального програмування, завдяки чому дуже зручно обробляти масиви даних, вибираючи їх потрібним чином (застосовуючи фільтрування, сортування),

також мова має засоби автоматичного збирання сміття, має більшу продуктивність, тому мову C# було обрано для веброзробки.

Для організації роботи з методом діагностування серцевої недостатності потрібен потужний засіб взаємодії з нейромережними моделями. Найактуальніші засоби надаються мовою Python, тому було вирішено виконувати розробку двома мовами, розподіляючи систему на відповідні складові.

3.2 Визначення функціональних вимог до програми

Функціональними вимогами до програми є:

- діагностування серцевої недостатності у пацієнта;
- перегляд проведених оглядів, результатів досліджень, виписаного лікування, проведених операцій, визначених діагнозів, встановлених пристроїв за пацієнтом;
- збереження результатів огляду пацієнта;
- збереження скарг на здоров'я самим пацієнтом;
- збереження результатів первинного огляду пацієнта кардіологом;
- пошук пацієнтів;
- обмеження доступу лікарів до даних пацієнта;
- збереження даних про встановлені пацієнту пристрої для підтримки серцево-судинної системи;
- збереження проведених операцій;
- виписування рецепту на лікування;
- створення довідника діагнозів, медикаментів;
- перегляд результатів досліджень, у тому числі аналізу крові, за заданим проміжком дат;
- збереження структурованих результатів аналізу крові;
- збереження результатів досліджень пацієнта з внесенням окремих файлів;

- встановлення діагнозу пацієнту лікарем;
- оформлення запису на прийом до лікаря;
- автоматичне створення шаблону для запису пацієнтів;
- формування розкладу роботи лікаря.

Відповідну діаграму варіантів використання приведено на рис. 3.1.

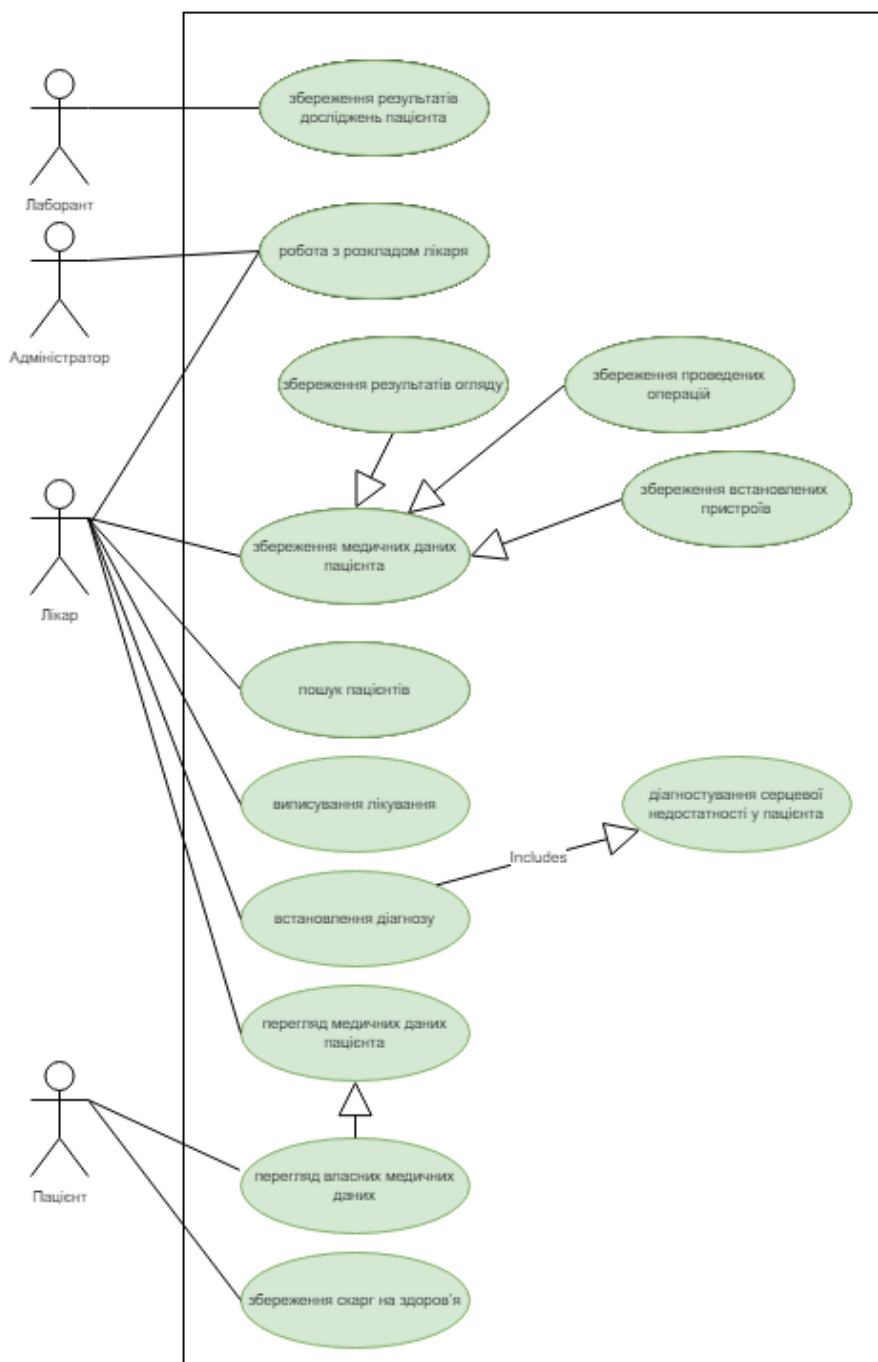


Рисунок 3.1 – Діаграма варіантів використання

3.3 Визначення структури бази даних

Структуру бази даних створено на основі забезпечення сценаріїв роботи користувача, розглянутих у попередньому підрозділі.

Таблиця Doctor визначає лікаря-кардіолога, характеризуючи його полями:

- поле DoctorId – ідентифікатор лікаря;
- поле FullName – повне ім'я лікаря (прізвище та ім'я, можливо по батькові);
- поле Position – конкретна кваліфікація лікаря (наприклад, тут може міститися категорія);
- поле Education – освіта лікаря;
- поле Phone – номер телефону лікаря;
- поле BirthDay – дата народження лікаря;
- поле Experience – досвід роботи лікаря, кількість років;
- поле WorksFrom – дата, з якої почав працювати;
- поле Description – опис;
- поле Photo – фотографія лікаря;
- поле AccountId – обліковий запис, за рахунок якого працює в системі лікар.

Таблиця Schedule визначає розклад роботи кардіолога, характеризуючи його полями:

- поле ScheduleId – номер запису;
- поле DoctorId – лікар, якому відповідає даний запис, є зовнішнім ключем;
- поле DayOfWeek – номер дня тижня, на якому відбувається прийом;
- поле Week – номер тижня (або парність тижня), для дня якого визначено прийом;
- поле TimeBegin – час початку прийому;
- поле TimeEnd – час завершення прийому;

– поле Interval – інтервал між прийомами (може задаватися за необхідності, в тому числі для розширення тривалості прийому).

Таблиця Meeting визначає конкретний запис на прийом до кардіолога (так званий талон), характеризуючи його полями:

- поле MeetingId – номер запису;
- поле ScheduleId – позиція розкладу лікаря, якій відповідає запис;
- поле PatientId – номер пацієнта, який записався на прийом, є зовнішнім ключем;
- поле DateBegin – запланована дата і час початку прийому;
- поле DateEnd – запланована дата і час завершення прийому;
- поле ActualDateBegin – фактична дата і час початку прийому;
- поле ActualDateEnd – запланована дата і час завершення прийому;
- поле State – стан прийому (заплановано, обслуговування, виконано, скасовано).

Таблиця PatientComplaint визначає скарги пацієнта на стан свого здоров'я, характеризуючи його полями:

- поле PatientComplaintId – номер скарги пацієнта;
- поле PatientId – пацієнт, який оформив скаргу, є зовнішнім ключем;
- поле ComplaintDetails – текст скарг;
- поле From – дата і час, коли було оформлено скаргу.

Таблиця Diagnosis визначає діагноз, характеризуючи його полями:

- поле DiagnosisId – номер діагнозу;
- поле Title – назва діагнозу;
- поле Description – опис діагнозу.

Таблиця Operation визначає хірургічне втручання, характеризуючи його полями:

- поле OperationId – номер операції;
- поле DiagnosisId – номер діагнозу, є зовнішнім ключем;
- поле PatientId – номер пацієнта, якому було виконано операцію;
- поле Title – назва;

- поле Description – опис;
- поле From – дата реалізації операції;
- поле Surgeon – хірург, який виконав операцію.

Таблиця Medicament визначає медикамент, характеризуючи його полями:

- поле MedicamentId – номер медикаменту;
- поле Title – назва медикаменту;
- поле Description – опис медикаменту.

Таблиця Procedure визначає процедуру дослідження діагнозу (ЕКГ, комп'ютерна томографія тощо) про серцеву хворобу, характеризуючи її полями:

- поле ProcedureId – номер процедури;
- поле Title – назва процедури;
- поле Description – опис процедури.

Таблиця ProcedureResult визначає результат процедури дослідження для пацієнта, характеризуючи його полями:

- поле ProcedureResultId – номер результату;
- поле ProcedureId – процедура, яку було проведено, є зовнішнім

ключем;

- поле PatientId – пацієнт, для якого проводилась процедура;
- поле From – дата і час внесення результатів процедури;
- поле Description – опис процедури;
- поле Conclusion – висновок за процедурою;
- поле MadeBy – опис того, ким було виконано процедуру.

Таблиця DiagnosisMedicament реалізує зв'язок багато-до-багатьох між таблицями Diagnosis і Medicament, визначаючи медикаменти, які можуть бути призначені за даним діагнозом для лікування пацієнта, формує таким чином довідник. Поля таблиці:

- поле DiagnosisId – діагноз, є зовнішнім ключем;
- поле MedicamentId – медикаменти, є зовнішнім ключем.

Таблиця Treatment визначає виписане лікарем пацієнту лікування, характеризуючи його полями:

- поле TreatmentId – номер виписаного лікування;
- поле PatientId – пацієнт, якому виписано лікування, є зовнішнім ключем;
- поле Description – опис лікування;
- поле From – дата і час, коли було виписано лікування.

Таблиця Access визначає доступ лікаря до даних пацієнта, характеризуючи його полями:

- поле AccessId – номер запису про доступ;
- поле PatientId – пацієнт, до даних якого надано доступ, є зовнішнім ключем;
- поле DoctorId – лікар, якому надано доступ.

Таблиця Examination визначає загальний огляд лікарем пацієнта, характеризуючи його полями:

- поле ExaminationId – номер огляду;
- поле MeetingId – номер запису на прийом до лікаря пацієнтом, є зовнішнім ключем;
- поле Complaints – скарги пацієнта, висловлені на прийомі;
- поле Objective – об'єктивні висновки лікаря про стан здоров'я пацієнта;
- поле Investigation – дослідження, що має бути проведено, щоб в подальшому визначити стан здоров'я пацієнта;
- поле Cure – лікування, призначене пацієнту (у даному випадку це загальна процедура лікування, опис, а лікарські засоби визначаються окремо).

Таблиця CardiologyExamination визначає огляд на бланку кардіолога, що представляє собою розширений варіант, визначаючи особливості, характерні для роботи кардіолога:

- поле CardiologyExaminationId – номер огляду кардіолога;

- поле ExaminationId – номер загального огляду, до якого відноситься даний, є зовнішнім ключем;
- поле From – заклад або лікар, від якого був направлений пацієнт;
- поле Anamnesis – анамнез;
- поле OverallState – загальний стан пацієнта;
- поле Temperature – температура пацієнта;
- поле SkinType – стан шкірних покривів;
- поле SkinLocalization – локалізація виявлених проблем;
- поле MucousMembraneType – стан слизової;
- поле MucousMembraneLocalization – локалізація проблем зі слизовою;
- поле PalatineTonsil – стан мигдалин;
- поле Gland – стан лимфатичних вузлів;
- поле Thyroid – стан щитовидної залози;
- поле Bone – стан кісно-суглобної системи;
- поле MuscularSystem – стан м'язів;
- поле Breath – частота дихання на хвилину;
- поле Noises – дихальні шуми;
- поле Pulse – пульс;
- поле PressureStaying – тиск стоячи;
- поле PressureLaying – тиск лежачи;
- поле HeartTones – тони серця;
- поле HeartLoosening – послаблення в серці;
- поле HeartAccent – акцент в серці;
- поле HeartSeparation – розщеплення в серці;
- поле HeartNoises – шуми в серці;
- поле HeartExpansion – розширені межі (напрямок);
- поле Tongue – стан язика;
- поле Stomach – стан живота;
- поле StomachPain – тип болю в животі;

- поле Liver – стан печінки;
- поле LiverState – відстань, на яку виступає печінка;
- поле Spleen – стан селезінки;
- поле SpleenState – відстань, на яку виступає селезінка;
- поле Kidneys – стан нирок;
- поле Beaten – синдром постукування;
- поле Urination – сечовипускання;
- поле Stool – випорожнення;
- поле Edema – набряк.

Таблиця PatientDiagnosis визначає діагноз, встановлений пацієнту, характеризуючи цей зв'язок полями:

- поле PatientDiagnosisId – номер запису;
- поле PatientId – пацієнт, якому визначено діагноз, є зовнішнім ключем;
- поле DiagnosisId – діагноз, який було встановлено пацієнту, є зовнішнім ключем;
- поле State – стан діагнозу (первинний, підтверджений або прогнозований двома моделями або однією тощо).

Таблиця Patient визначає пацієнта, характеризуючи його полями:

- поле PatientId – номер пацієнта;
- поле FullName – повне ім'я пацієнта;
- поле Phone – номер телефону пацієнта;
- поле Address – адреса пацієнта;
- поле Job – місце роботи пацієнта;
- поле Photo – фотографія пацієнта;
- поле Height – зріст пацієнта;
- поле Weight – вага пацієнта;
- поле Gender – стать пацієнта;
- поле BirthDay – дата народження пацієнта;
- поле Passport – номер і серія паспорту пацієнта;

- поле PatientFrom – дата реєстрації в системі;
- поле AccountId – обліковий запис, за рахунок якого працює в системі пацієнт.

Таблиця Account визначає обліковий запис, характеризуючи його полями:

- поле AccountId – номер облікового запису;
- поле Login – логін;
- поле Password – пароль;
- поле Role – роль користувача (користувач може бути не тільки лікарем або пацієнтом, але і лаборантом, адміністратором).

Таблиця DeviceType визначає тип пристрою, який може бути встановлено пацієнту. Мається на увазі саме кардіопристрій. Таблиця характеризується полями:

- поле DeviceTypeId – номер типу пристрою;
- поле Title – назва типу пристрою.

Таблиця Device визначає конкретний пристрій, встановлений пацієнту, характеризуючи його полями:

- поле DeviceId – номер пристрою;
- поле DeviceTypeId – тип пристрою, є зовнішнім ключем;
- поле Model – модель пристрою;
- поле Description – опис пристрою;
- поле From – дата встановлення пристрою.

Таблиця BloodAnalysis визначає аналіз крові, характеризуючи його полями:

- поле BloodAnalysisId – номер аналізу крові;
- поле PatientId – пацієнт, якому належить цей аналіз, є зовнішнім ключем;
- поле WBC – кількість лейкоцитів у крові;
- поле RBC – кількість еритроцитів у крові;
- поле Hb – рівень гемоглобіну;

- поле Ht – вміст гематокриту;
- поле MCV – середній обсяг еритроцитів;
- поле MCH – середній вміст гемоглобіну в еритроциті;
- поле MCHC – середня концентрація гемоглобіну в еритроциті;
- поле PLT – обсяг тромбоцитів;
- поле MPV – середній обсяг тромбоцитів;
- поле RDWCV – ширина розповсюдження еритроцитів;
- поле Eosinophily – частка еозинофілів;
- поле Monocity – частка моноцитів;
- поле SOE – швидкість з’єднання еритроцитів;
- поле Color – кольоровий показник;
- поле Basophily – частка базофілів;
- поле Lymphocity – частка лімфоцитів;
- поле Neutrophily – частка нейтрофілів;
- поле EosinophilyAbs – абсолютна кількість еозинофілів;
- поле NeutrophilyAbs – абсолютна кількість нейтрофілів;
- поле BasophilyAbs – абсолютна кількість базофілів;
- поле MonocityAbs – абсолютна кількість моноцитів;
- поле LimphoocityAbs – абсолютна кількість лімфоцитів;
- поле RDWSD – ширина розповсюдження еритроцитів, стандартне відхилення;
- поле Laboratory – лабораторія, де було проведено аналіз;
- поле From – дата і час виконання аналізу.

Таблиця EHR містить об’єднану інформацію за дослідженнями, проведеними стосовно пацієнта, які разом з аналізом крові і загальною інформацією про пацієнта використовуються для роботи метода, описаного в розділі 2.

Схема бази даних представлена на рис. 3.2.

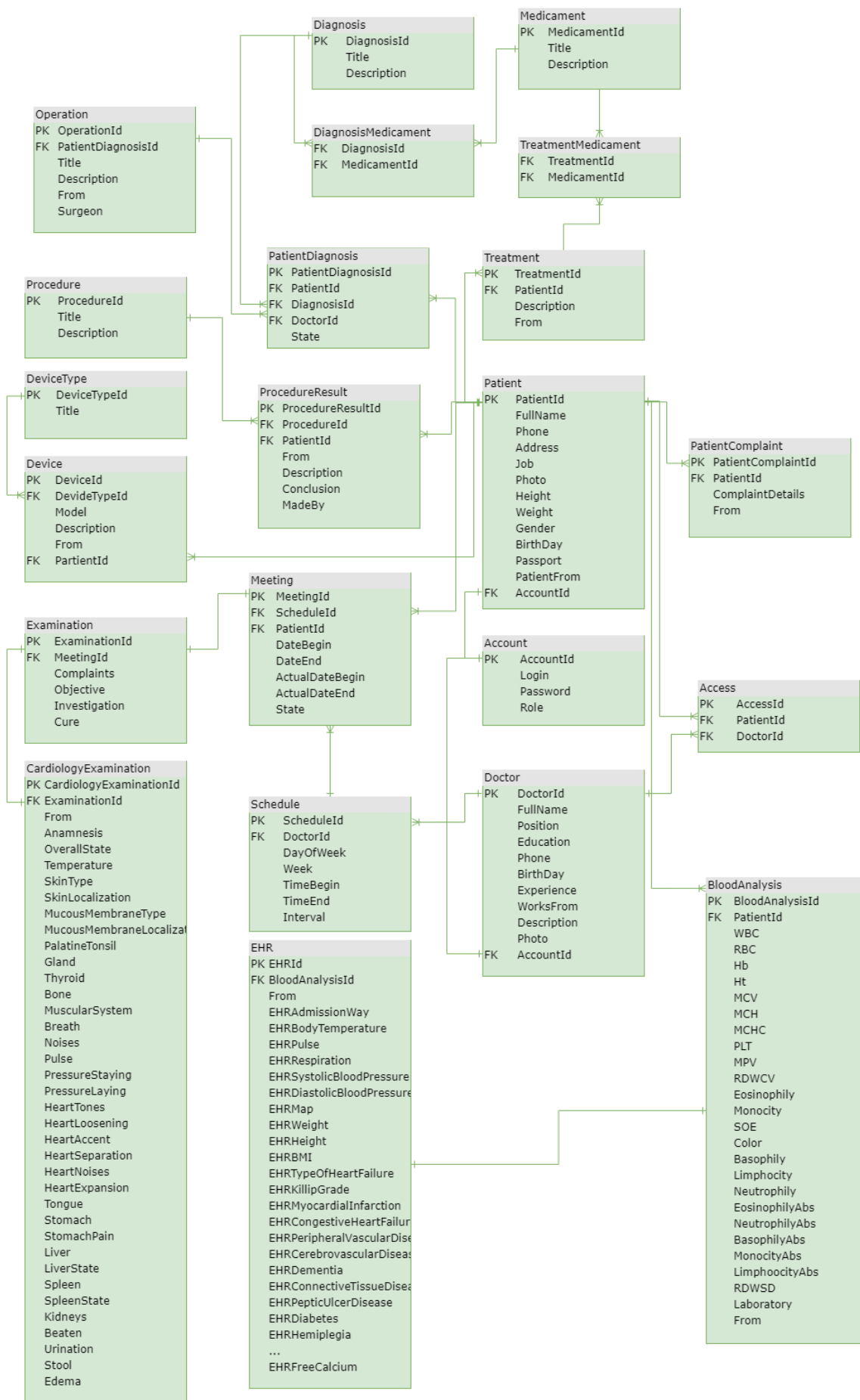


Рисунок 3.2 – Схема бази даних

3.4 Структура програмного проєкту

Структура програмного проєкту (рис. 3.3) створена з виділенням окремої підсистеми діагностування серцевої недостатності і окремо фронтенду та бекенду системи, які представляють вебзастосунок, через який контактує з системою користувач.

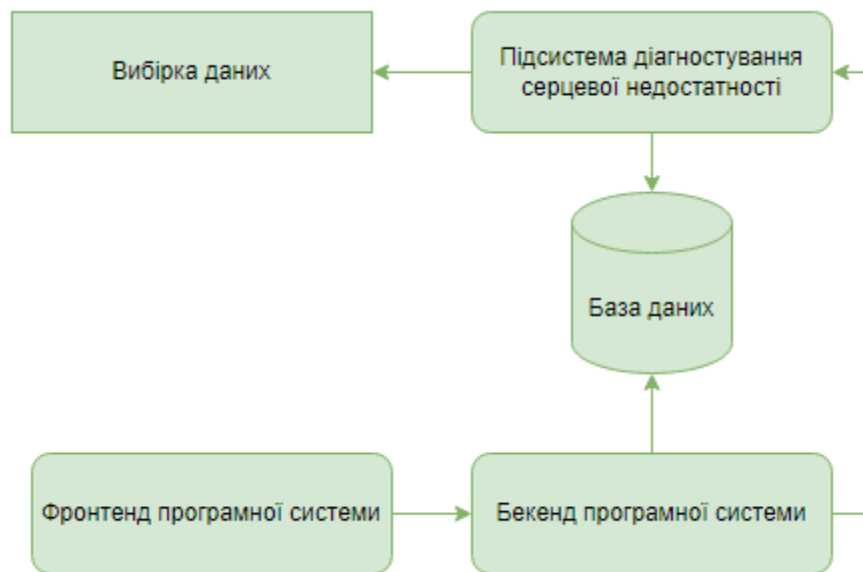


Рисунок 3.3 – Структура програмного проєкту

Підсистема діагностування серцевої недостатності взаємодіє з вибіркою даних для навчання моделей, а отримані результати зберігає в базу даних. У свою чергу бекенд програмної системи, отримуючи запити від фронтенду, або реалізує звернення до бази даних, або звертається до підсистеми діагностування, щоб отримати результат діагностування для вказаних даних пацієнта.

3.5 Висновки за розділом 3

Виконано проєктування програми з визначенням засобів розробки, що включають мови програмування C#, Python, JavaScript, мову розмітки HTML,

мову таблиць стилів CSS, фреймворк ASP.NET Core, систему керування базами даних Microsoft SQL Server.

Визначено вимоги до програми, створено діаграму прецедентів, виконано проєктування структури бази даних, за результатами якого представлено схему бази даних, визначено структуру програми.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Алгоритм функціонування програми

Функціонування програми (рис. 4.1) розпочинається з виведення загальної інформації кардіологічний центр, а далі користувач обов'язково має авторизуватися для виконання подальших дій.

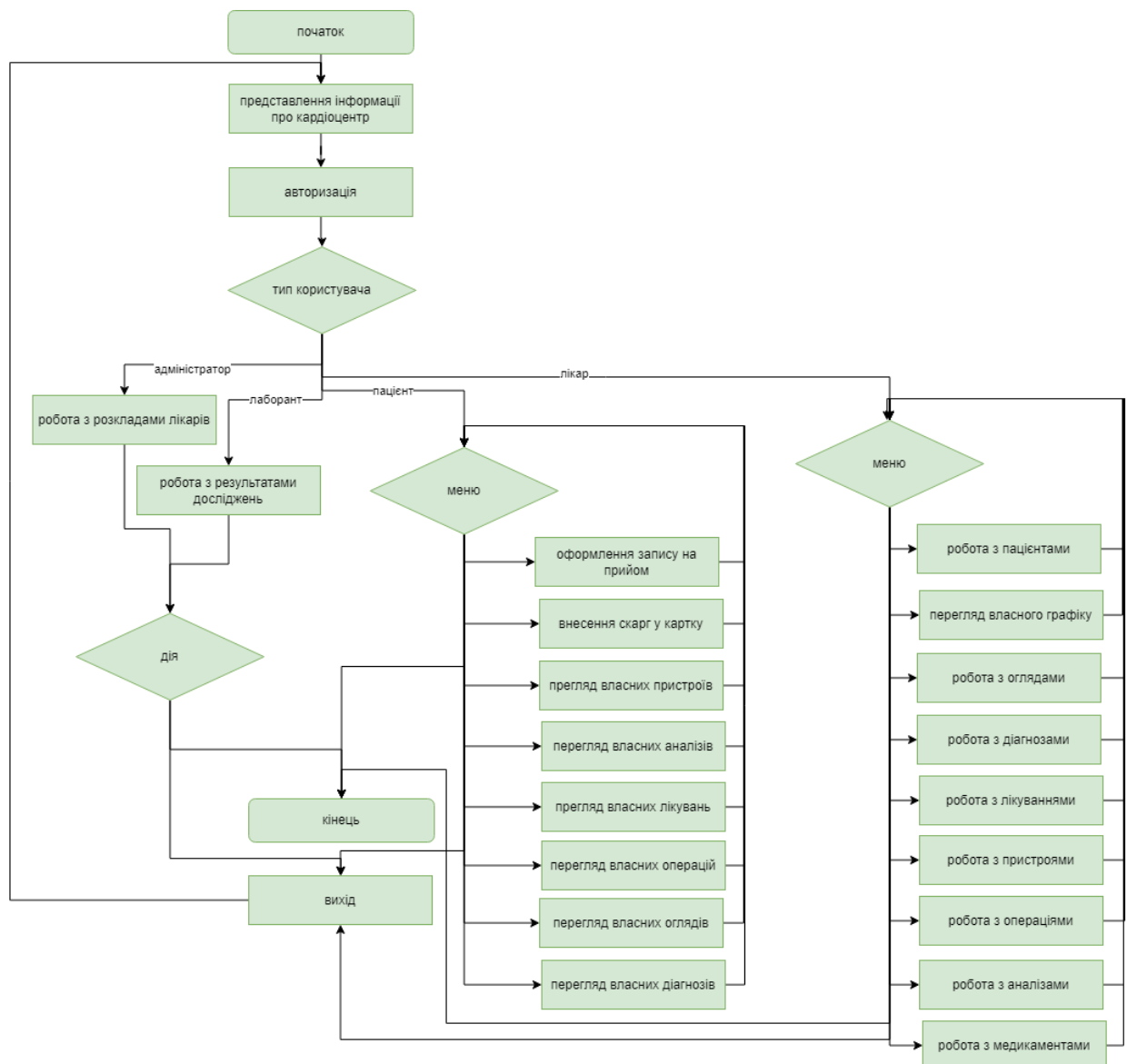


Рисунок 4.1 – Схема функціонування програми

У випадку, якщо авторизований користувач є адміністратором, то йому надається можливість працювати тільки з розкладом лікарів. Адміністратор може внести запис на певний час пацієнта, може переглядати час відвідування за кожним лікарем. Більше засобів роботи йому не надається.

У випадку, якщо користувач є лаборантом, то він реалізує тільки роботу з дослідженнями (аналізами), вносячи відповідні файли і їх описи, а також задаючи результатами аналізу крові. Більше засобів роботи йому не надається.

У випадку, якщо користувач є пацієнтом, то йому надається доступ до меню, яке включає пункти, що стосуються роботи з його власними медичними даними, але розподіленими між окремими пунктами, також пацієнт може записатися на прийом до лікаря. Також йому надається можливість працювати над внесенням власних скарг на здоров'я.

У випадку, якщо користувач є лікарем, то йому вже надається доступ до роботи з даними всіх пацієнтів. Хоча при цьому перевіряється, чи має він доступ до даних відповідного пацієнта. Якщо такого доступу немає, то дані не виводиться. Лікар може працювати з переліком пацієнтів і звідти переходити до даних відповідного пацієнта, наприклад, до його оглядів. Може лікар обрати окремий розділ і звідти отримати всі наявні огляди. Також за цим пунктом можна перейти до створення нового запису про огляд.

4.2 Опис модулів реалізованої програмної системи

ПЗ вебзастосунку реалізовано на основі використання шаблону Model-View-Controller.

Під час реалізації програми було створено набір моделей, що відповідають таблицям бази даних, розглянутим у третьому розділі. При цьому використано модуль Entity Framework, що дозволило не створювати окремо і таблиці бази даних, і класи моделей, а на основі визначених класів

моделей створити структуру бази даних автоматично. Це спростило процес розробки.

Під кожен функцію користувача, представлену в розділі 3 було створено відповідні шаблони. Усі шаблони створено на основі використання шаблону [21].

Окрім моделей і шаблонів було створено контролери, які забезпечують підтримку роботи представлень щодо заповнення їх відповідними даними. Для цього для кожної сутності моделей було передбачено відповідний набір контролерів, що відповідають за роботу основних функцій, що стосуються цієї сутності.

Для кожної сутності на основі відповідного контролера створено метод, що відповідає за додавання об'єктів. Він організований як відповідний метод контролера з іменем Create і в непараметризованому форматі передбачає роботу з GET-формою, а в параметризованому – з POST-формою. У другому варіанті до параметрів передаються шляхом прив'язування значення відповідних полів форми, що забезпечують встановлення значень всіх атрибутів відповідного класу.

Для кожної сутності також передбачено створення методу Update контролеру, який також працює в двох режимах: в непараметризованому форматі передбачає роботу з GET-формою, а в параметризованому – з POST-формою. У другому варіанті до параметрів передаються шляхом прив'язування значення відповідних полів форми, що забезпечують встановлення значень всіх атрибутів відповідного класу, але при цьому також задається конкретний номер відповідного об'єкту, до якого прив'язуються значення.

Метод контролера Details передбачає деталізоване виведення даних відповідного об'єкту за заданим номером цього об'єкта, переданим в якості номера до єдиного параметра цього методу.

Розглянемо на прикладі контролера BloodAnalysisController принципи створення інших методів у складі контролера, які реалізують роботу зі списковими даними.

Метод DataBloodAnalysis(int? std, int? stm, int? sty, int? ed, int? em, int? ey, int? patientnum) реалізує заповнення представлення даними аналізів крові для пацієнта з номером patientnum, що належать періоду часу між заданими датами, сформованими окремо за днем, місяцем і роком початку і відповідно завершення цього інтервалу. При цьому одночасно перевіряється, чи має даний користувач право на доступ до таких даних:

```
Access ac = mddc.Accesses.Include(b => b.Patient).Include(b =>
b.Doctor).FirstOrDefault(b => b.Doctor.AccountId == user.UserId &&
b.Patient.AccountId == patientnum);
```

Якщо об'єкт ac непорожній, то такий доступ є.

Метод PatientBloodAnalysis() виконує заповнення представлення даними результатів аналізу крові для авторизованого у поточну мить користувача-пацієнта.

Метод PatientBloodAnalysisByDoctor(int? patientnum) виконує виведення даних результатів аналізу крові для пацієнта з номером patientnum для лікаря.

Метод DataBloodAnalysis(int? std, int? stm, int? sty, int? ed, int? em, int? ey) працює аналогічно однойменному методу, описаному вище, але при цьому вже не відбувається перевірка на рівень доступу.

Моделі, що працюють в основі методу, представленого в роботі, було реалізовано шляхом використання нейромережної бібліотеки Keras. Створено класи під роботу відповідних моделей StructurizedModel, UnstructurizedModel, клас загальної підтримки роботи методу MedDiagnosticsModel та клас підтримки передоброблення даних DataProcessor.

4.3 Висновки за розділом 4

Описано логіку функціонування програми з розділенням на різні послідовності етапів в залежності від типу користувача. Виконано опис основної логіки структуризації розробленого застосунку. Його створено на основі використання шаблону Model-View-Controller. Описано принципи організації класів.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Призначення програми

Програма призначена для підтримки роботи лікарів, зокрема кардіологів, для чого враховуються особливості їх роботи, адміністраторів, для чого надаються засоби створення розкладу лікарів, пацієнтів, які можуть переглядати власні медичні дані, вносити власні скарги, лаборантів, що представляють лабораторії з дослідження аналізів пацієнтів.

5.2 Умови виконання програми

Для виконання програми потрібно мати наступні мінімальні ресурси на сервері:

- процесор: щонайменше два ядра, мінімальна частота – 2 ГГц;
- оперативний запам'ятовуючий пристрій – не менше 4 Гб;
- жорсткий диск: вільний простір не менше 1 Гб.

5.3 Робота з програмою

Робота з програмою можлива в режимах 4 типів користувачів у відповідності зі схемою функціонування програми, розглянутою в розділі 4. У даному підрозділі розглянемо основні принципи організації інтерфейсу користувача для виконання основних функцій.

На рис. 5.1 представлено приклад роботи з програмою пацієнта. Роль користувача зазначається вгорі сторінки. Меню доступне для вибору потрібного пункту зліва. Пацієнт може або переглядати власні медичні дані, розділені за відповідними пунктами меню, або формувати свої скарги на здоров'я через пункт меню «Скарги», а також може записатися на прийом для лікаря. Для цього йому потрібно обрати пункт меню «Записатися», тоді

відкриється сторінка на якій потрібно обрати з випадуючого переліку одного з доступних лікарів, обрати дату прийому, а тоді натиснути на кнопку «Графік» для підтвердження вибору. Знизу на сторінці при цьому відображається перелік слотів, на які може записатися пацієнт. Якщо слот вже зайнятий іншим пацієнтом, то виводиться повідомлення про недоступність, а іконка в слоті відображається червоною. Зелена іконка відображається для вільних для запису слотів. Для запису потрібно натиснути на посилання «Записатися». Після цього буде виведено повідомлення про успішний запис, якщо все відбулось коректно.

Time Slot	Status	Action
11:00 - 11:15	Не доступно	
11:15 - 11:30	Не доступно	
11:30 - 11:45	Доступно	Записатися
11:45 - 12:00	Доступно	Записатися
12:00 - 12:15	Доступно	Записатися
12:15 - 12:30	Доступно	Записатися

Рисунок 5.1 – Запис на прийом до лікаря

Робота адміністратора реалізується через обмежений інтерфейс, в якому на відміну від запису до лікаря пацієнтом, відображаються також прізвища пацієнтів, які записалися на прийом у зайнятих слотах.

У свою чергу лікар має можливість працювати над практично всіма даними, вносити відповідні записи, редагувати існуючі. Для цього йому доступний перелік пунктів меню, звідки він може обрати потрібний, а також

може почати роботу з пункту меню «Пацієнти», звідки він може обрати потрібного пацієнта, а з узагальненої сторінки переходити на сторінки аналізів, діагнозів вже саме цього пацієнта.

Приклад введення даних нового огляду кардіологом приведено на рис. 5.2. На цій сторінці лікар може обрати пацієнта, підкоригувавши вибір на минулій сторінці за потреби. Після цього лікар має внести всі необхідні дані огляду за пунктами, кожен з яких представляє відповідне поле інтерфейсу.

Рисунок 5.2 – Огляд пацієнта кардіологом

Для роботи над діагнозами різних пацієнтів лікар може відкрити відповідний пункт меню «Діагнози» (рис. 5.3), тоді відобразиться таблиця з визначеними цим лікарем або автоматично визначеними діагнозами. Відповідні значення виводяться в різних стовпчиках таблиці. Якщо модель за структурованими даними рекомендувала одне рішення, а модель за

текстовими описами – інше, то окрім рекомендації першої виводиться також рекомендація другої – перевірити папери.

Лікар може або переглянути всю інформацію про діагноз на окремій сторінці, або вирішити підтвердити діагноз, який був винесений автоматично.

Кардіолог Іван Приймаченко [Вийти](#)

Винесені та рекомендовані діагнози пацієнтів
 Рекомендовані діагнози створюються програмним забезпеченням як порада лікарю.
 Лікар відповідальний за винесене рішення.
☒ додати рекомендовані системою рішення

[Підтвердити](#) [Новий діагноз](#)

Зведені діагнози

10 на сторінку Шукати пацієнта:

Пацієнт	Діагноз винесений	Діагноз рекомендований		
Андрій Іванов	Не встановлено	Серцева недостатність	Переглянути	Підтвердити
Віктор Шиманський	Серцева недостатність першого ступеня	Серцева недостатність відсутня (перевірити папери)	Переглянути	Підтвердити
Ганна Верко	Серцева недостатність третього ступеня	Серцева недостатність	Переглянути	Підтвердити
Олег Зінов'єв	Серцева недостатність другого ступеня	Серцева недостатність	Переглянути	Підтвердити
Євген Мерченко	Серцева недостатність третього ступеня	Серцева недостатність	Переглянути	Підтвердити

з 1 по 5 запис Назад 1 Вперед

Рисунок 5.3 – Робота кардіолога з діагнозами

5.4 Експериментальне дослідження проблеми діагностування серцевої недостатності

Проблему діагностування серцевої недостатності було експериментально досліджено за допомогою вибірки даних, сформованої головним чином на основі даних вибірки [22].

Найкраща точність була отримана за використання методу діагностування серцевої недостатності у 0.91, вона виявилась на 0.03 більшою за точність багатосарової нейронної мережі зі зменшеним простором ознак. Зменшення простору ознак було виконано у відповідності з

концепцією, описаною в запропонованому методі. Підвищення точності було досягнуто за рахунок використання в вибірці усіх екземплярів, у тому числі тих, для яких була пропущена частина даних. Вона була уточнена за рахунок текстових описів з ЕМК пацієнтів.

Отримані результати було зведено до табл. 5.1.

Таблиця 5.1 – Порівняння точності діагностування

Модель або метод	Точність діагностування
Метод k-найближчих сусідів	0.67
Метод опорних векторів	0.81
Дерева рішень	0.82
Багатошарова нейронна мережа	0.8
Багатошарова нейронна мережа зі зменшеним простором ознак	0.88
Метод діагностування серцевої недостатності	0.91

5.5 Висновки за розділом 5

Описано основні режими роботи програми. Проведене експериментальне дослідження було виконано шляхом порівняння методу k-найближчих сусідів, методу опорних векторів, дерев рішень, багатошарової нейронної мережі, її варіанту зі зменшеним простором ознак, а також запропонованого в роботі методу. Його результати підтвердили ефективність запропонованого методу для роботи з відповідною вибіркою даних.

ВИСНОВКИ

У кваліфікаційній роботі магістра розглянуто проблему розроблення ПЗ діагностування серцевої недостатності для забезпечення підтримки роботи кардіолога.

Проведено аналіз стану діагностування серцевих хвороб, що дозволило виявити основні поняття даної галузі, особливості процесу виявлення серцевих хвороб щодо використовуваних аналізів, класи серцевої недостатності.

За результатами аналізу інтелектуальних методів, які можуть використовуватися для розв'язання проблеми, створено власний метод розв'язання проблеми діагностування серцевої недостатності, який характеризується виділенням двох моделей класифікації на основі структурованих та текстових даних медичних карток пацієнтів, що дозволяє виносити рішення за відсутності одного з типів даних або виносити узгоджене рішення, сприяючи підвищенню обґрунтованості діагностичних рішень.

Виконано проєктування програми з визначенням засобів розробки, що включають мови програмування C#, Python, JavaScript, мову розмітки HTML, мову таблиць стилів CSS, фреймворк ASP.NET Core, систему керування базами даних Microsoft SQL Server. Визначено вимоги до програми, виконано проєктування структури бази даних, визначено структуру програми, виконано програмну реалізацію, проведено експериментальне дослідження.

Програма призначена для підтримки роботи лікарів, зокрема кардіологів з врахуванням особливостей їх роботи, забезпечується діагностування стану пацієнтів щодо наявності серцевої недостатності, адміністраторів з наданням засобів створення розкладу лікарів, пацієнтів, які можуть переглядати власні медичні дані, вносити власні скарги, лаборантів, що представляють лабораторії з дослідження аналізів пацієнтів.

Завдання роботи виконані повністю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Heart Failure. NHLBI, NIH [Electronic resource]. – Access mode : <https://www.nhlbi.nih.gov/health-topics/heart-failure>.
2. Global Burden of Disease [Electronic resource]. – Access mode : <https://www.healthdata.org/gbd/2019>.
3. Серцево-судинні захворювання – головна причина смерті українців [Електрон. ресурс]. – Режим доступу : <https://phc.org.ua/news/sercevo-sudinni-zakhvoryuvannya-golovna-prichina-smerti-ukrainciv-visnovki-z-doslidzhennya>.
4. Lilly, L. S. Pathophysiology of Heart Disease [Text] : An Introduction to Cardiovascular Medicine / L S. Lilly. – Philadelphia : LWW, 2020. – 480 p.
5. Heart failure – Diagnosis and Treatment [Electronic resource]. – Access mode : <https://www.mayoclinic.org/diseases-conditions/heart-failure/diagnosis-treatment/drc-20373148>.
6. Bale, B. Beat the Heart Attack Gene [Text] : The Revolutionary Plan to Prevent Heart Disease, Stroke, and Diabetes / B. Bale. – New York : Wiley General Trade, 2014. – 340 p.
7. Top EHR Software Vendors For 2021 Comparison [Electronic resource]. – Access mode : <https://www.praxisemr.com/top-ehr-vendors.html>.
8. EHR System For Cardiology [Electronic resource]. – Access mode : <https://www.practicestudio.net/EHR-Cardiology/Overview>.
9. Компьютерные программы для кардиолога – обзор [Электрон. ресурс]. – Режим доступа : https://www.livemedical.ru/tags/programmy_dlja_kardiologa/.
10. A Powerful EHR Solution for Cath Labs or Cardiology Center – SoftCath [Electronic resource]. – Access mode : <https://www.softcath.com/a-powerful-ehr-solution-for-cath-labs-or-cardiology-center-softcath/>.
11. Priyadarshini, I. A novel LSTM–CNN–grid search-based deep neural network for sentiment analysis [Electronic resource] / Ishaani Priyadarshini, Chase

Cotton // The Journal of Supercomputing. – 2021. – Vol. 77. – P. 13911-13932. – Access mode : <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8097246/>.

12. Singh, G. Understanding Architecture of LSTM [Electronic resource] / G. Singh. – Access mode : <https://www.analyticsvidhya.com/blog/2021/01/understanding-architecture-of-lstm/>.

13. Okedi, T. I. Time series analysis and long short-term memory (LSTM) network prediction of BPV current density [Text] / R. I. Okedi, A. C. Fisher // Energy & Environmental. – 2021. – Vol. 14. – P. 2408-2418.

14. Gavrilova, Y. Convolutional Neural Networks for Beginners [Electronic resource] / Y. Gavrilova. – Access mode : <https://serokell.io/blog/introduction-to-convolutional-neural-networks>.

15. Как работает сверточная нейронная сеть [Электрон. ресурс] : архитектура, примеры, особенности. – Режим доступа : <https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/>.

16. Goodfellow, I. Deep Learning [Text] / Ian Goodfellow, Yoshua Bengio, Aaron Courville. – Cambridge : MIT Press, 2016. – 326 p.

17. Gupta, A. Feature Selection Techniques in Machine Learning [Electronic resource] / A. Gupta. – Access mode : <https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/>.

18. Doshi, S. Skip-Gram [Electronic resource] : NLP context words prediction algorithm / S. Doshi. – Access mode : <https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c>.

19. Lock, A. ASP.NET Core in Action [Text] / A. Lock. – New York : Manning, 2021. – 832 p.

20. Murach, J. Murach's ASP.NET Core MVC [Text] / J. Murach, M. Delamater. – Fresno, CA : Mike Murach & Associates, 2020. – 780 p.

21. Binary Admin Free Website Template [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page204/binary-admin>.

22. Hospitalized patients with heart failure [Electronic resource] : integrating electronic healthcare records and external outcome data (version 1.2) / Z. Zhang, L. Cao, Y. Zhao et al. – Access mode : <https://physionet.org/content/heart-failure-zigong/1.2/#files-panel>.

ДОДАТОК А
Технічне завдання

Вступ

ПЗ, яке розробляється в даній роботі, повинно, по-перше, підвищити зручність роботи з даними щодо серцевих хвороб для спеціалістів у цій сфері – кардіологів, а по-друге, реалізувати відповідний метод, направлений на роботу з пацієнтами, які мають або потенційно можуть мати серцеві захворювання, виконуючи для них діагностування серцевої недостатності.

A.1 Підстави для розробки

Завдання на дипломну кваліфікаційну роботу магістра визначене за темою “Дослідження та програмна реалізація методів діагностування серцевих хвороб” і затверджене наказом № 435 від 5 листопада 2021 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програма призначена для підтримки роботи лікарів, зокрема кардіологів, для чого враховуються особливості їх роботи, адміністраторів, для чого надаються засоби створення розкладу лікарів, пацієнтів, які можуть переглядати власні медичні дані, вносити власні скарги, лаборантів, що представляють лабораторії з дослідження аналізів пацієнтів.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Розробка повинна забезпечувати виконання:

- діагностування серцевої недостатності у пацієнта;
- перегляд проведених оглядів, результатів досліджень, виписаного лікування, проведених операцій, визначених діагнозів, встановлених

пристроїв за пацієнтом;

- збереження результатів огляду пацієнта;
- збереження скарг на здоров'я самим пацієнтом;
- збереження результатів первинного огляду пацієнта кардіологом;
- пошук пацієнтів;
- обмеження доступу лікарів до даних пацієнта;
- збереження даних про встановлені пацієнту пристрої для підтримки

серцево-судинної системи;

- збереження проведених операцій;
- виписування рецепту на лікування;
- створення довідника діагнозів, медикаментів;
- перегляд результатів досліджень, у тому числі аналізу крові, за

заданим проміжком дат;

- збереження структурованих результатів аналізу крові;
- збереження результатів досліджень пацієнта з внесенням окремих

файлів;

- встановлення діагнозу пацієнту лікарем;
- оформлення запису на прийом до лікаря;
- автоматичне створення шаблону для запису пацієнтів;
- формування розкладу роботи лікаря.

А.3.2 Вимоги до надійності

Для забезпечення надійного функціонування програмного забезпечення необхідна постійна підтримка віддалених запитів користувачів як через локальну мережу, так і через мережу Інтернет. Для цього програма має розташовуватися на виділеному для цього сервері, який повинен функціонувати постійно і обробляти запити до вебсервера та сервера бази даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічні засоби, необхідні для підтримки роботи програми, повинні відповідати наступним вимогам:

- процесор: щонайменше два ядра, мінімальна частота – 2 ГГц;
- оперативний запам'ятовуючий пристрій – не менше 4 Гб;
- жорсткий диск: вільний простір не менше 1 Гб.

A.4 Порядок контролю та приймання

Контроль виконання роботи здійснюється за календарним планом роботи. Під час приймання роботи здійснюється перевірка відповідності визначеним функціональним характеристикам програми.

ДОДАТОК Б
Текст програми

Б.1 Текст файла BloodAnalysisController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MedDiagnosis.Data;
using MedDiagnosis.Models;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;

namespace MedDiagnosis.Controllers
{
    public class BloodAnalysisController : Controller
    {
        private readonly MedDiagnosisDataContext mddc;

        public BloodAnalysisController (MedDiagnosisDataContext dc)
        {
            mddc = dc;
        }

        public ActionResult Details(int? blanalysisnum)
        {
            var blanalysis = mddc.BloodAnalyses.Include(a =>
a.BloodAnalysisId).Include(a => a.Patient).Include(a =>
a.WBCStr).Include(a => a.RBCStr).Include(a => a.HbStr).Include(a =>
a.HtStr).Include(a => a.MCVStr).Include(a => a.MCHStr).Include(a =>
a.MCHCStr).Include(a => a.PLTStr).Include(a => a.MPVStr).Include(a =>
a.RDWCVStr).Include(a => a.EosinophilyStr).Include(a =>
a.MonocityStr).Include(a => a.SOEStr).Include(a => a.ColorStr).Include(a
=> a.BasophilyStr).Include(a => a.LymphocityStr).Include(a =>
a.NeutrophilyStr).Include(a => a.EosinophilyAbsStr).Include(a =>
a.BasophilyAbsStr).Include(a => a.MonocityAbsStr).Include(a =>
a.LimphoocityAbsStr).Include(a => a.RDWSDStr).Include(a =>
a.LaboratoryStr).Include(a => a.FromStr).FirstOrDefaultAsync(a =>
a.BloodAnalysisId == blanalysisnum);
            if (blanalysis == null)
            {
                return NotFound();
            }
        }
    }
}

```

```

        return View(blanalysis);
    }
    [Authorize]
    public ActionResult Create()
    {
        ViewData["Patients"] = new SelectList(db.Patients,
"PatientId", "FullName", "Phone");
        return View();
    }

    [HttpPost] [ValidateAntiForgeryToken]
    //форма методом POST
    public ActionResult Create([Bind("BloodAnalysisId, Patient,
WBCS, RBC, Hb, Ht, MCV, MCH, MCHC, PLT, MPV, RDWCV, Eosinophily,
Monocity, SOE, Color, Basophily, Lymphocity, Neutrophily, EosinophilyAbs,
BasophilyAbs, MonocityAbs, LymphoocityAbs, RDWSD, Laboratory, From")]
BloodAnalysis blanalysis)
    {
        if (ModelState.IsValid)
        {
            mddc.Add(blanalysis);
            mddc.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
        return View(blanalysis);
    }

    [Authorize]
    public ActionResult Update(int? blanalysis)
    {
        BloodAnalysis ba =
mddc.BloodAnalysises.Find(blanalysis);
        if (ba == null)
        {
            return NotFound();
        }
        ViewData["Patients"] = new SelectList(db.Patients,
"PatientId", "FullName", "Phone");
        return View(ba);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]

```

```

        //форма методом POST
        public ActionResult Update(int blanalysis,
[Bind("BloodAnalysisId, Patient, WBCS, RBC, Hb, Ht, MCV, MCH, MCHC, PLT,
MPV, RDWCV, Eosinophily, Monocity, SOE, Color, Basophily, Lymphocity,
Neutrophily, EosinophilyAbs, BasophilyAbs, MonocityAbs, LimphoocityAbs,
RDWSD, Laboratory, From")] BloodAnalysis blanalysis)
        {
            if (ModelState.IsValid)
            {
                mddc.BloodAnalysises.Update(blanalysis);
                mddc.SaveChanges();
            }
            ViewData["Patients"] = new SelectList(db.Patients,
"PatientId", "FullName", "Phone");
            return View(blanalysis);
        }

[Authorize]
public ActionResult DataBloodAnalysis(int? std, int? stm,
int? sty, int? ed, int? em, int? ey, int? patientnum)
{
    Access ac = mddc.Accesses.Include(b =>
b.Patient).Include(b => b.Doctor).FirstOrDefault(b => b.Doctor.AccountId
= user.UserId && b.Patient.AccountId = patientnum);
    if (ac)
    {
        DateTime stdate = DateTime(sty, stm, std);
        DateTime edate = DateTime(ey, em, ed);
        var analysis = mddc.BloodAnalysises.Include(a =>
a.BloodAnalysisId).Include(a => a.Patient).Include(a =>
a.WBCStr).Include(a => a.RBCStr).Include(a => a.HbStr).Include(a =>
a.HtStr).Include(a => a.MCVStr).Include(a => a.MCHStr).Include(a =>
a.MCHCStr).Include(a => a.PLTStr).Include(a => a.MPVStr).Include(a =>
a.RDWCVStr).Include(a => a.EosinophilyStr).Include(a =>
a.MonocityStr).Include(a => a.SOEStr).Include(a => a.ColorStr).Include(a
=> a.BasophilyStr).Include(a => a.LymphocityStr).Include(a =>
a.NeutrophilyStr).Include(a => a.EosinophilyAbsStr).Include(a =>
a.BasophilyAbsStr).Include(a => a.MonocityAbsStr).Include(a =>
a.LimphoocityAbsStr).Include(a => a.RDWSDStr).Include(a =>
a.LaboratoryStr).Include(a => a.FromStr).Where(a => a.From >= stdate &&
a.From<= edate && a.Patient.AccountId = patientnum);
        return View(analysis);
    }
}

```

```

    }
    [Authorize]
    public ActionResult PatientBloodAnalysis()
    {
        var analysis = mddc.BloodAnalysises.Include(a =>
a.BloodAnalysisId).Include(a => a.Patient).Include(a =>
a.WBCStr).Include(a => a.RBCStr).Include(a => a.HbStr).Include(a =>
a.HtStr).Include(a => a.MCVStr).Include(a => a.MCHStr).Include(a =>
a.MCHCStr).Include(a => a.PLTStr).Include(a => a.MPVStr).Include(a =>
a.RDWCVStr).Include(a => a.EosinophilyStr).Include(a =>
a.MonocityStr).Include(a => a.SOEStr).Include(a => a.ColorStr).Include(a
=> a.BasophilyStr).Include(a => a.LymphocityStr).Include(a =>
a.NeutrophilyStr).Include(a => a.EosinophilyAbsStr).Include(a =>
a.BasophilyAbsStr).Include(a => a.MonocityAbsStr).Include(a =>
a.LimphoocityAbsStr).Include(a => a.RDWSDStr).Include(a =>
a.LaboratoryStr).Include(a => a.FromStr).Where(a.Patient.AccountId =
user.UserId);

        return View(analysis);
    }
    [Authorize]
    public ActionResult PatientBloodAnalysisByDoctor(int?
patientnum)
    {
        Access ac = mddc.Accesses.Include(b =>
b.Patient).Include(b => b.Doctor).FirstOrDefault(b => b.Doctor.AccountId
= user.UserId && b.Patient.AccountId = patientnum);
        if (ac)
        {
            var analysis = mddc.BloodAnalysises.Include(a =>
a.BloodAnalysisId).Include(a => a.Patient).Include(a =>
a.WBCStr).Include(a => a.RBCStr).Include(a => a.HbStr).Include(a =>
a.HtStr).Include(a => a.MCVStr).Include(a => a.MCHStr).Include(a =>
a.MCHCStr).Include(a => a.PLTStr).Include(a => a.MPVStr).Include(a =>
a.RDWCVStr).Include(a => a.EosinophilyStr).Include(a =>
a.MonocityStr).Include(a => a.SOEStr).Include(a => a.ColorStr).Include(a
=> a.BasophilyStr).Include(a => a.LymphocityStr).Include(a =>
a.NeutrophilyStr).Include(a => a.EosinophilyAbsStr).Include(a =>
a.BasophilyAbsStr).Include(a => a.MonocityAbsStr).Include(a =>
a.LimphoocityAbsStr).Include(a => a.RDWSDStr).Include(a =>
a.LaboratoryStr).Include(a => a.FromStr).Where(a.Patient.AccountId =
patientnum);

            return View(analysis);
        }
    }

```

```

    }
    [Authorize]
    public ActionResult DataBloodAnalysis(int? std, int? stm,
int? sty, int? ed, int? em, int? ey)
    {
        DateTime stdate = DateTime(sty, stm, std);
        DateTime edate = DateTime(ey, em, ed);
        var analysis = mddc.BloodAnalysises.Include(a =>
a.BloodAnalysisId).Include(a => a.Patient).Include(a =>
a.WBCStr).Include(a => a.RBCStr).Include(a => a.HbStr).Include(a =>
a.HtStr).Include(a => a.MCVStr).Include(a => a.MCHStr).Include(a =>
a.MCHCStr).Include(a => a.PLTStr).Include(a => a.MPVStr).Include(a =>
a.RDWCVStr).Include(a => a.EosinophilyStr).Include(a =>
a.MonocityStr).Include(a => a.SOEStr).Include(a => a.ColorStr).Include(a
=> a.BasophilyStr).Include(a => a.LymphocityStr).Include(a =>
a.NeutrophilyStr).Include(a => a.EosinophilyAbsStr).Include(a =>
a.BasophilyAbsStr).Include(a => a.MonocityAbsStr).Include(a =>
a.LimphoocityAbsStr).Include(a => a.RDWSDStr).Include(a =>
a.LaboratoryStr).Include(a => a.FromStr).Where(a => a.From >= stdate &&
a.From<= edate);

        return View(analysis);
    }
}
}

```

Б.2 Текст файла EHR.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace MedDiagnosis.Models
{
    public class EHR
    {
        [Key]
        public int EHRIId {get; set;}
        public int PatientId {get; set;}
        public int BloodAnalysisId {get; set;}
        public DateTime From {get; set;}
        public bool EHRAssessmentWay {get; set;}
    }
}

```

```

public int EHRVisitTimes {get; set;}
public float EHRBodyTemperature {get; set;}
[NotMapped] //не є полем таблиці
public string EHRBodyTemperatureStr
{
    get
    {
        Return          Math.Round(this.EHRBodyTemperature,
1).ToString();
    }
}
public int EHRPulse {get; set;}
public int EHRRespiration {get; set;}
public int EHRsystolicBloodPressure {get; set;}
public int EHRdiastolicBloodPressure {get; set;}
public float EHRMap {get; set;}
[NotMapped] //не є полем таблиці
public string EHRMapStr
{
    get
    {
        return Math.Round(this.EHRMap, 5).ToString();
    }
}
public int EHRWeight {get; set;}
public float EHRHeight {get; set;}
[NotMapped] //не є полем таблиці
public string EHRHeightStr
{
    get
    {
        return Math.Round(this.EHRHeight, 2).ToString();
    }
}
public float EHRBMI {get; set;}
[NotMapped] //не є полем таблиці
public string EHRBMIStr
{
    get
    {
        return Math.Round(this.EHRBMI, 3).ToString();
    }
}
}

```

```

public int EHRTYPEOfHeartFailure {get; set;}
public int EHRNYHACardiacFunctionClassification {get; set;}
public int EHRKillipGrade {get; set;}
public bool EHRMyocardialInfarction {get; set;}
public bool EHRCongestiveHeartFailure {get; set;}
public bool EHRPeripheralVascularDisease {get; set;}
public bool EHCerebrovascularDisease {get; set;}
public bool EHRDementia {get; set;}
public bool EHRChronicObstructivePulmonaryDisease {get;
set;}}

public bool EHRConnectiveTissueDisease {get; set;}
public bool EHRPepticUlcerDisease {get; set;}
public bool EHRDiabetes {get; set;}
public bool EHRModerateToSevereChronicKidneyDisease {get;
set;}}

public bool EHRHemiplegia {get; set;}
public bool EHRLeukemia {get; set;}
public bool EHRMalignantLymphoma {get; set;}
public bool EHRSolidTumor {get; set;}
public bool EHRLiverDisease {get; set;}
public bool EHRAIDS {get; set;}
public int EHRCCIScore {get; set;}
public int EHRTYPEOfRespiratoryFailure {get; set;}
public bool EHRConsciousness {get; set;}
public int EHREyeOpening {get; set;}
public int EHRVerbalResponse {get; set;}
public int EHRMovement {get; set;}
public int EHRRespiratorySupport {get; set;}
public bool EHROxygenInhalation {get; set;}
public int EHRFio2 {get; set;}
public int EHRAcuteRenalFailure {get; set;}
public bool EHRLVEF {get; set;}
public int EHRLeftVentricularEndDiastolicDiameterLV {get;
set;}}

public float EHRMitralValveEMS {get; set;}
[NotMapped] //не є полем таблиці
public string EHRMitralValveEMSStr
{
    get
    {
        return Math.Round(this.EHRMitralValveEMS,
2).ToString();
    }
}

```

```

    }
    public float EHRMitralValveAMS {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRMitralValveAMSStr
    {
        get
        {
            return Math.Round(this.EHRMitralValveAMS,
2).ToString();
        }
    }
    public float EHREA {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHREAStr
    {
        get
        {
            return Math.Round(this.EHREA, 2).ToString();
        }
    }
    public float EHRTricuspidValveReturnVelocity {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRTricuspidValveReturnVelocityStr
    {
        get
        {
            return
Math.Round(this.EHRTricuspidValveReturnVelocity, 2).ToString();
        }
    }
    public float EHRTricuspidValveReturnPressure {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRTricuspidValveReturnPressureStr
    {
        get
        {
            return
Math.Round(this.EHRTricuspidValveReturnPressure, 1).ToString();
        }
    }
    public bool EHROutcomeDuringHospitalization {get; set;}
    public bool EHRDeathWithin28Days {get; set;}
    public bool EHRReAdmissionWithin28Days {get; set;}

```

```

public bool EHRDeathWithin3Months {get; set;}
public bool EHRReAdmissionWithin3Months {get; set;}
public bool EHRDeathWithin6Months {get; set;}
public bool EHRReAdmissionWithin6Months {get; set;}
public bool EHRTIMEOfDeathDaysFromAdmission {get; set;}
public int EHRReAdmissionTimeDaysFromAdmission {get; set;}
public bool EHRReturnToEmergencyDepartmentWithin6Months
{get; set;}

public int EHRTIMEToEmergencyDepartmentWithin6Months {get;
set;}

public float EHRCreatinineEnzymaticMethod {get; set;}
[NotMapped] //не є полем таблиці
public string EHRCreatinineEnzymaticMethodStr
{
    get
    {
        return
Math.Round(this.EHRCreatinineEnzymaticMethod, 1).ToString();
    }
}
public float EHRUrea {get; set;}
[NotMapped] //не є полем таблиці
public string EHRUreaStr
{
    get
    {
        return Math.Round(this.EHRUrea, 2).ToString();
    }
}
public int EHRUricAcid {get; set;}
public float EHRGlomerularFiltrationRate {get; set;}
[NotMapped] //не є полем таблиці
public string EHRGlomerularFiltrationRateStr
{
    get
    {
        return Math.Round(this.EHRGlomerularFiltrationRate,
2).ToString();
    }
}
public float EHCystatin {get; set;}
[NotMapped] //не є полем таблиці
public string EHCystatinStr

```

```

    {
        get
        {
            return Math.Round(this.EHRCystatin, 2).ToString();
        }
    }
    public float EHRWhiteBloodCell {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRWhiteBloodCellStr
    {
        get
        {
            return Math.Round(this.EHRWhiteBloodCell,
2).ToString();
        }
    }
    public float EHRRedBloodCell {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRRedBloodCellStr
    {
        get
        {
            return Math.Round(this.EHRRedBloodCell,
2).ToString();
        }
    }
    public float
EHRCoefficientOfVariationOfRedBloodCellDistributionWidth {get; set;}
    [NotMapped] //не є полем таблиці
    public string
EHRCoefficientOfVariationOfRedBloodCellDistributionWidthStr
    {
        get
        {
            return
Math.Round(this.EHRCoefficientOfVariationOfRedBloodCellDistributionWidth,
1).ToString();
        }
    }
    public float
EHRStandardDeviationOfRedBloodCellDistributionWidth {get; set;}
    [NotMapped] //не є полем таблиці

```

```

public string
EHRStandardDeviationOfRedBloodCellDistributionWidthStr
{
    get
    {
        return
Math.Round(this.EHRStandardDeviationOfRedBloodCellDistributionWidth,
1).ToString();
    }
}
public float EHRMeanCorpuscularVolume {get; set;}
[NotMapped] //не є полем таблиці
public string EHRMeanCorpuscularVolumeStr
{
    get
    {
        return Math.Round(this.EHRMeanCorpuscularVolume,
1).ToString();
    }
}
public float EHRMeanPlateletVolume {get; set;}
[NotMapped] //не є полем таблиці
public string EHRMeanPlateletVolumeStr
{
    get
    {
        return Math.Round(this.EHRMeanPlateletVolume,
1).ToString();
    }
}
public int EHRPlatelet {get; set;}
public float EHRPlateletDistributionWidth {get; set;}
[NotMapped] //не є полем таблиці
public string EHRPlateletDistributionWidthStr
{
    get
    {
        return
Math.Round(this.EHRPlateletDistributionWidth, 1).ToString();
    }
}
public float EHRPlateletHematocrit {get; set;}
[NotMapped] //не є полем таблиці

```

```

public string EHRPlateletHematocritStr
{
    get
    {
        return Math.Round(this.EHRPlateletHematocrit,
1).ToString();
    }
}
public float EHRDDimer {get; set;}
[NotMapped] //не є полем таблиці
public string EHRDDimerStr
{
    get
    {
        return Math.Round(this.EHRDDimer, 1).ToString();
    }
}
public float EHRInternationalNormalizedRatio {get; set;}
[NotMapped] //не є полем таблиці
public string EHRInternationalNormalizedRatioStr
{
    get
    {
        return
Math.Round(this.EHRInternationalNormalizedRatio, 1).ToString();
    }
}
public float EHRActivatedPartialThromboplastinTime {get;
set;}
[NotMapped] //не є полем таблиці
public string EHRActivatedPartialThromboplastinTimeStr
{
    get
    {
        return
Math.Round(this.EHRActivatedPartialThromboplastinTime, 1).ToString();
    }
}
public float EHRThrombinTime {get; set;}
[NotMapped] //не є полем таблиці
public string EHRThrombinTimeStr
{
    get

```

```

        {
            return Math.Round(this.EHRThrombinTime,
1).ToString();
        }
    }
    public float EHRProthrombinActivity {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRProthrombinActivityStr
    {
        get
        {
            return Math.Round(this.EHRProthrombinActivity,
1).ToString();
        }
    }
    public float EHRProthrombinTimeRatio {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRProthrombinTimeRatioStr
    {
        get
        {
            return Math.Round(this.EHRProthrombinTimeRatio,
2).ToString();
        }
    }
    public float EHRFibrinogen {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRFibrinogenStr
    {
        get
        {
            return Math.Round(this.EHRFibrinogen,
2).ToString();
        }
    }
    public float EHRHighSensitivityTroponin {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRHighSensitivityTroponinStr
    {
        get
        {
            return Math.Round(this.EHRHighSensitivityTroponin,
3).ToString();
        }
    }

```

```

    }
}
public float EHRMyoglobin {get; set;}
[NotMapped] //не є полем таблиці
public string EHRMyoglobinStr
{
    get
    {
        return Math.Round(this.EHRMyoglobin, 2).ToString();
    }
}
public float EHRCarbonDioxideBindingCapacity {get; set;}
[NotMapped] //не є полем таблиці
public string EHRCarbonDioxideBindingCapacityStr
{
    get
    {
        return
Math.Round(this.EHRCarbonDioxideBindingCapacity, 1).ToString();
    }
}
public float EHRCalcium {get; set;}
[NotMapped] //не є полем таблиці
public string EHRCalciumStr
{
    get
    {
        return Math.Round(this.EHRCalcium, 2).ToString();
    }
}
public float EHRPotassium {get; set;}
[NotMapped] //не є полем таблиці
public string EHRPotassiumStr
{
    get
    {
        return Math.Round(this.EHRPotassium, 2).ToString();
    }
}
public float EHRChloride {get; set;}
[NotMapped] //не є полем таблиці
public string EHRChlorideStr
{

```

```

        get
        {
            return Math.Round(this.EHRChloride, 1).ToString();
        }
    }
    public float EHR Sodium {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHR SodiumStr
    {
        get
        {
            return Math.Round(this.EHR Sodium, 1).ToString();
        }
    }
    public float EHR InorganicPhosphorus {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHR InorganicPhosphorusStr
    {
        get
        {
            return Math.Round(this.EHR InorganicPhosphorus,
2).ToString();
        }
    }
    public float EHR SerumMagnesium {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHR SerumMagnesiumStr
    {
        get
        {
            return Math.Round(this.EHR SerumMagnesium,
2).ToString();
        }
    }
    public float EHR CreatineKinaseIsoenzymeToCreatineKinase
{get; set;}
    [NotMapped] //не є полем таблиці
    public string EHR CreatineKinaseIsoenzymeToCreatineKinaseStr
    {
        get
        {

```

```

        return
Math.Round(this.EHRCreatineKinaseIsoenzymeToCreatineKinase,
2).ToString();
    }
}
public float
EHRHydroxybutyrateDehydrogenaseToLactateDehydrogenase {get; set;}
    [NotMapped] //не є полем таблиці
public string
EHRHydroxybutyrateDehydrogenaseToLactateDehydrogenaseStr
    {
        get
        {
            return
Math.Round(this.EHRHydroxybutyrateDehydrogenaseToLactateDehydrogenase,
2).ToString();
        }
    }
    public int EHRHydroxybutyrateDehydrogenase {get; set;}
    public int EHRGlutamicOxaloaceticTransaminase {get; set;}
    public int EHRCreatineKinase {get; set;}
    public float EHRCreatineKinaseIsoenzyme {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRCreatineKinaseIsoenzymeStr
    {
        get
        {
            return Math.Round(this.EHRCreatineKinaseIsoenzyme,
1).ToString();
        }
    }
    public int EHRLactateDehydrogenase {get; set;}
    public float EHRBrainNatriureticPeptide {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRBrainNatriureticPeptideStr
    {
        get
        {
            return Math.Round(this.EHRBrainNatriureticPeptide,
2).ToString();
        }
    }
    public float EHRHighSensitivityProtein {get; set;}

```

```

[NotMapped] //не є полем таблиці
public string EHRHighSensitivityProteinStr
{
    get
    {
        return Math.Round(this.EHRHighSensitivityProtein,
1).ToString();
    }
}
public float nt EHRNucleotidase {get; set;}
[NotMapped] //не є полем таблиці
public string EHRNucleotidaseStr
{
    get
    {
        return Math.Round(this.EHRNucleotidase,
1).ToString();
    }
}
public float EHRFucosidase {get; set;}
[NotMapped] //не є полем таблиці
public string EHRFucosidaseStr
{
    get
    {
        return Math.Round(this.EHRFucosidase,
1).ToString();
    }
}
public float EHRAAlbumin {get; set;}
[NotMapped] //не є полем таблиці
public string EHRAAlbuminStr
{
    get
    {
        return Math.Round(this.EHRAAlbumin, 1).ToString();
    }
}
public float EHRWhiteGlobulinRatio {get; set;}
[NotMapped] //не є полем таблиці
public string EHRWhiteGlobulinRatioStr
{
    get

```

```

        {
            return Math.Round(this.EHRWhiteGlobulinRatio,
1).ToString();
        }
    }
    public int EHRCholinesterase {get; set;}
    public int EHRGlutamyltranspeptidase {get; set;}
    public int EHRGlutamicPyruvicTransaminase {get; set;}
    public float EHRGlutamicOxaliplatin {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRGlutamicOxaliplatinStr
    {
        get
        {
            return Math.Round(this.EHRGlutamicOxaliplatin,
2).ToString();
        }
    }
    public float EHRIndirectBilirubin {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRIndirectBilirubinStr
    {
        get
        {
            return Math.Round(this.EHRIndirectBilirubin,
1).ToString();
        }
    }
    public int EHRAlkalinePhosphatase {get; set;}
    public float EHRGlobulin {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRGlobulinStr
    {
        get
        {
            return Math.Round(this.EHRGlobulin, 1).ToString();
        }
    }
    public float EHRDirectBilirubin {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRDirectBilirubinStr
    {
        get

```

```

        {
            return Math.Round(this.EHRDirectBilirubin,
1).ToString();
        }
    }
    public float EHRTotalBilirubin {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRTotalBilirubinStr
    {
        get
        {
            return Math.Round(this.EHRTotalBilirubin,
1).ToString();
        }
    }
    public float EHRTotalBileAcid {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRTotalBileAcidStr
    {
        get
        {
            return Math.Round(this.EHRTotalBileAcid,
1).ToString();
        }
    }
    public float EHRTotalProtein {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRTotalProteinStr
    {
        get
        {
            return Math.Round(this.EHRTotalProtein,
1).ToString();
        }
    }
    public int EHRerythrocyteSedimentationRate {get; set;}
    public float EHRCholesterol {get; set;}
    [NotMapped] //не є полем таблиці
    public string EHRCholesterolStr
    {
        get
        {

```

```

        return Math.Round(this.EHRCholesterol,
2).ToString();
    }
}
public float EHRLowDensityLipoproteinCholesterol {get;
set;}

[NotMapped] //не є полем таблиці
public string EHRLowDensityLipoproteinCholesterolStr
{
    get
    {
        return
Math.Round(this.EHRLowDensityLipoproteinCholesterol, 2).ToString();
    }
}
public float EHRTriglyceride {get; set;}
[NotMapped] //не є полем таблиці
public string EHRTriglycerideStr
{
    get
    {
        return Math.Round(this.EHRTriglyceride,
2).ToString();
    }
}
public float EHRHighDensityLipoproteinCholesterol {get;
set;}

[NotMapped] //не є полем таблиці
public string EHRHighDensityLipoproteinCholesterolStr
{
    get
    {
        return
Math.Round(this.EHRHighDensityLipoproteinCholesterol, 2).ToString();
    }
}
public float EHRHomocysteine {get; set;}
[NotMapped] //не є полем таблиці
public string EHRHomocysteineStr
{
    get
    {

```

```

                return Math.Round(this.EHRHomocysteine,
2).ToString();
            }
        }
        public float EHRApolipoproteinA {get; set;}
        [NotMapped] //не є полем таблиці
        public string EHRApolipoproteinAStr
        {
            get
            {
                return Math.Round(this.EHRApolipoproteinA,
2).ToString();
            }
        }
        public float EHRApolipoproteinB {get; set;}
        [NotMapped] //не є полем таблиці
        public string EHRApolipoproteinBStr
        {
            get
            {
                return Math.Round(this.EHRApolipoproteinB,
2).ToString();
            }
        }
        public int EHRLipoprotein
        public float EHRPH
        [NotMapped] //не є полем таблиці
        public string EHRPHStr
        {
            get
            {
                return Math.Round(this.EHRPH, 2).ToString();
            }
        }
        public float EHRStandardResidualBase
        [NotMapped] //не є полем таблиці
        public string EHRStandardResidualBaseStr
        {
            get
            {
                return Math.Round(this.EHRStandardResidualBase,
1).ToString();
            }
        }

```

```

    }
    public float EHRStandardBicarbonate
    [NotMapped] //не є полем таблиці
    public string EHRStandardBicarbonateStr
    {
        get
        {
            return Math.Round(this.EHRStandardBicarbonate,
1).ToString();
        }
    }
    public int EHRPartialPressureOfCarbonDioxide
    public float EHRTotalCarbonDioxide
    [NotMapped] //не є полем таблиці
    public string EHRTotalCarbonDioxideStr
    {
        get
        {
            return Math.Round(this.EHRTotalCarbonDioxide,
1).ToString();
        }
    }
    public float EHRMethemoglobin
    [NotMapped] //не є полем таблиці
    public string EHRMethemoglobinStr
    {
        get
        {
            return Math.Round(this.EHRMethemoglobin,
1).ToString();
        }
    }
    public int EHRHematocritBloodGas
    public float EHRReducedHemoglobin
    [NotMapped] //не є полем таблиці
    public string EHRReducedHemoglobinStr
    {
        get
        {
            return Math.Round(this.EHRReducedHemoglobin,
1).ToString();
        }
    }
}

```

```

public float EHRPotassiumIon
[NotMapped] //не є полем таблиці
public string EHRPotassiumIonStr
{
    get
    {
        return Math.Round(this.EHRPotassiumIon,
2).ToString();
    }
}
public int EHRChlorideIon
public float EHRsodiumIon
[NotMapped] //не є полем таблиці
public string EHRsodiumIonStr
{
    get
    {
        return Math.Round(this.EHRsodiumIon, 1).ToString();
    }
}
public float EHRGlucoseBloodGas
[NotMapped] //не є полем таблиці
public string EHRGlucoseBloodGasStr
{
    get
    {
        return Math.Round(this.EHRGlucoseBloodGas,
1).ToString();
    }
}
public float EHRLactate
[NotMapped] //не є полем таблиці
public string EHRLactateStr
{
    get
    {
        return Math.Round(this.EHRLactate, 1).ToString();
    }
}
public float EHRMeasuredResidualBase
[NotMapped] //не є полем таблиці
public string EHRMeasuredResidualBaseStr
{

```

```

        get
        {
            return Math.Round(this.EHRMeasuredResidualBase,
1).ToString();
        }
    }
    public float HERMeasuredBicarbonate
    [NotMapped] //не є полем таблиці
    public string HERMeasuredBicarbonateStr
    {
        get
        {
            return Math.Round(this.HERMeasuredBicarbonate,
1).ToString();
        }
    }
    public float EHRCarboxyhemoglobin
    [NotMapped] //не є полем таблиці
    public string EHRCarboxyhemoglobinStr
    {
        get
        {
            return Math.Round(this.EHRCarboxyhemoglobin,
1).ToString();
        }
    }
    public int EHRBodyTemperatureBloodGas
    public int EHROxygenSaturation
    public int EHRPartialOxygenPressure
    public int EHROxyhemoglobin
    public float EHRAnionGap
    [NotMapped] //не є полем таблиці
    public string EHRAnionGapStr
    {
        get
        {
            return Math.Round(this.EHRAnionGap, 1).ToString();
        }
    }
    public float EHRFreeCalcium
    [NotMapped] //не є полем таблиці
    public string EHRFreeCalciumStr
    {

```

```

        get
        {
            return Math.Round(this.EHRFreeCalcium,
2).ToString());
        }
    }
    public int EHRTotalHemoglobin
    public int EHRGCS
    public int EHRDischargeDay
    public int EHRAgeCat
    public virtual Patient Patient
    public virtual BloodAnalysis BloodAnalysis
    [NotMapped] //не є полем таблиці
    public string FromStr
    {
        get
        {
            return
((DateTime)this.From).ToString("dd.MM.yyyy");
        }
    }
}
}

```

Б.3 Текст файла CardiologyExamination.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace MedDiagnosis.Models
{
    public class CardiologyExamination
    {
        [Key]
        public int CardiologyExaminationId {get; set;}
        public int ExaminationId {get; set;}
        public string Anamnesis {get; set;}
        public DateTime From {get; set;}
        public byte OverallState {get; set;}
        public float Temperature {get; set;}
        [NotMapped] //не є полем таблиці
    }
}

```

```

public string TemperatureStr
{
    get
    {
        return Math.Round(this.Temperature, 1).ToString();
    }
}
public byte SkinType {get; set;}
public string SkinLocalization {get; set;}
public byte MucousMembraneType {get; set;}
public string MucousMembraneLocalization {get; set;}
public byte PalatineTonsil {get; set;}
public byte Gland {get; set;}
public byte Thyroid {get; set;}
public byte Bone {get; set;}
public byte MuscularSystem {get; set;}
public byte Breath {get; set;}
public byte Noises {get; set;}
public byte Pulse {get; set;}
public string PressureStaying {get; set;}
public string PressureLaying {get; set;}
public byte HeartTones {get; set;}
public byte HeartLoosening {get; set;}
public byte HeartAccent {get; set;}
public byte HeartSeparation {get; set;}
public byte HeartNoises {get; set;}
public float HeartExpansion {get; set;}
[NotMapped] //не є полем таблиці
public string HeartExpansionStr
{
    get
    {
        return Math.Round(this.HeartExpansion,
2).ToString();
    }
}
public byte Tongue {get; set;}
public byte Stomach {get; set;}
public byte StomachPain {get; set;}
public byte Liver {get; set;}
public byte LiverState {get; set;}
public byte Spleen {get; set;}
public byte SpleenState {get; set;}

```

```

        public byte Kidneys {get; set;}
        public byte Beaten {get; set;}
        public byte Urination {get; set;}
        public byte Stool {get; set;}
        public byte Edema {get; set;}
        public virtual Examination Examination
        [NotMapped] //не є полем таблиці
        public string FromStr
        {
            get
            {
                return
                ((DateTime)this.From).ToString("dd.MM.yyyy");
            }
        }
        [NotMapped] //не є полем таблиці
        public string TemperatureStr
        {
            get
            {
                return Math.Round(this.Temperature, 1).ToString();
            }
        }
        [NotMapped] //не є полем таблиці
        public string HeartExpansionStr
        {
            get
            {
                return Math.Round(this.HeartExpansion,
                1).ToString();
            }
        }
    }
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Кваліфікаційна робота магістра

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ДІАГНОСТУВАННЯ СЕРЦЕВИХ ХВОРОБ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
METHODS FOR DIAGNOSIS OF HEART DISEASES**

Виконав
студент групи КНТ-120м

О.Є. Мяло

Керівник роботи
к.т.н., доцент

В.М. Льовкін

2021

Рисунок В.1 – Слайд 1

2

Об'єкт дослідження – процес діагностування
серцевих хвороб.

Предмет дослідження – методи діагностування
серцевої недостатності.

Мета роботи – розробити програмне
забезпечення діагностування серцевої
недостатності для забезпечення підтримки
роботи кардіолога.

Рисунок В.2 – Слайд 2

Аналіз предметної області

Критерій	Praxis	CathLab	Розроблювана система
Внесення результатів обстеження пацієнтів	+	+	+
Внесення історії хвороби	+	+	+
Наявність спеціалізованих засобів протоколів кардіологів	–	+	+
Підтримка прийняття діагностуючих рішень	–	–	+

Рисунок В.3 – Слайд 3

Метод діагностування серцевої недостатності

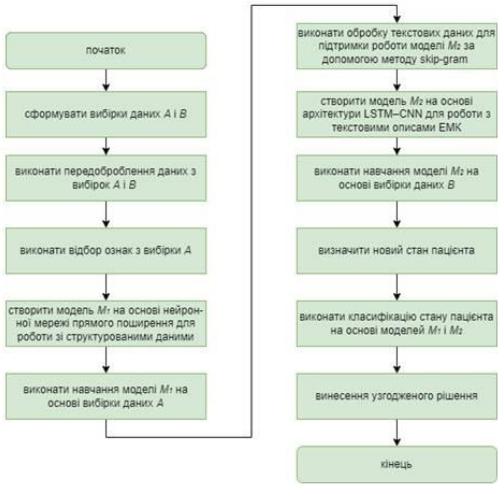


Рисунок В.4 – Слайд 4

Критерій	C#	JavaScript
Фреймворк для серверної частини	ASP.NET Core	Node.js
Типізація	Статична	Динамічна
Кросплатформність	Так	Так
Підтримка об'єктно-орієнтованого програмування	Так	Так
Підтримка функціонального програмування	Так	Ні
Автоматичне збирання сміття	Так	Ні
Гнучкість	Висока	Менше
Продуктивність	Більша	Менша

Діаграма варіантів використання



Рисунок В.6 – Слайд 6

Схема бази даних

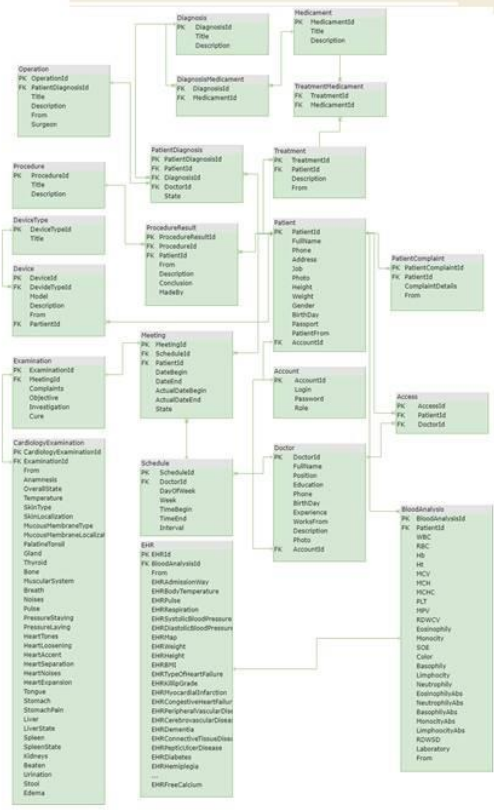


Рисунок В.7 – Слайд 7

Структура програмного проєкту

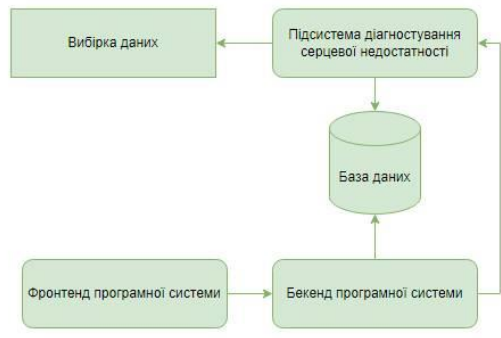


Рисунок В.8 – Слайд 8

Схема функціонування програми

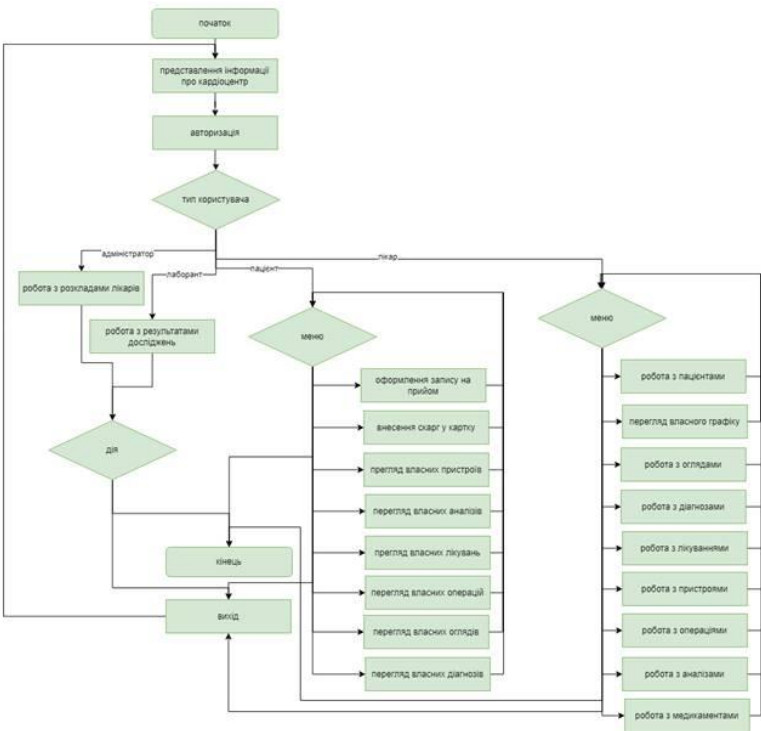


Рисунок В.9 – Слайд 9

Запис до лікаря на прийом

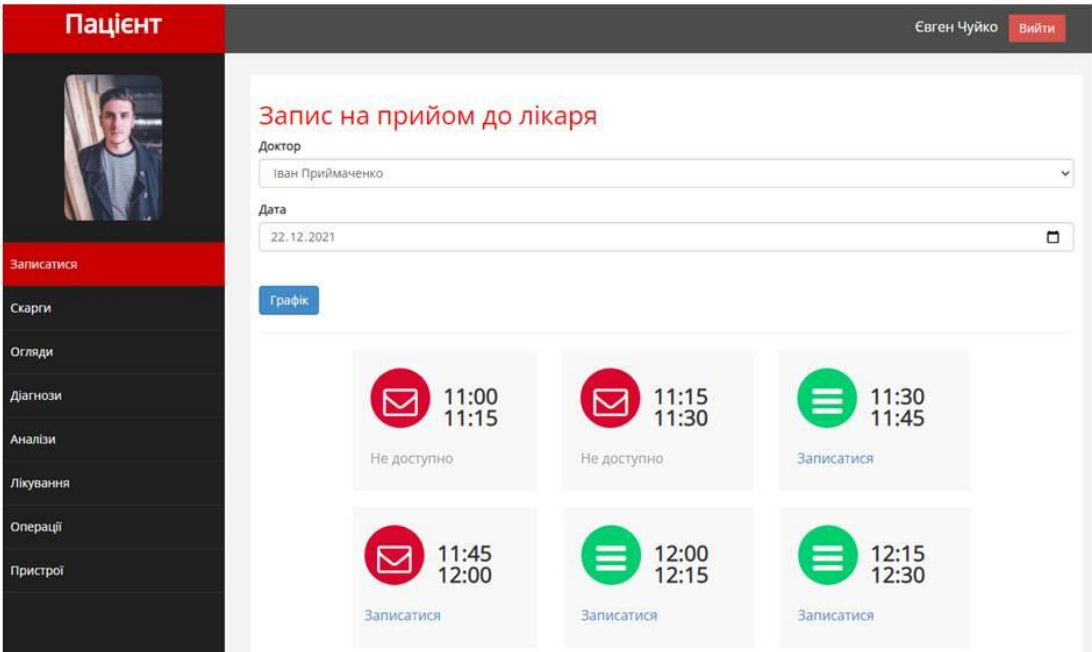


Рисунок В.10 – Слайд 10

Огляд пацієнта кардіологом

Кардіолог



Пацієнти

Графік

Огляди

Діагнози

Аналізи

Медикаменти

Лікування

Операції

Пристрої

Іван Приймаченко

Вийти

Новий огляд

Пацієнт

Андрій Виговський

Форма для нового огляду кардіолога

Загальна частина

Загальний стан

задовільний

Анамнез

Температура тіла

Тиск сидячи

Тиск стоячи

Стан серця

Пульс на хвилину

Тони серця

ясні

Послаблення

Акцент

Розділення

Рисунок В.11 – Слайд 11

Робота кардіолога над діагнозами

Кардіолог



Пацієнти

Графік

Огляди

Діагнози

Аналізи

Медикаменти

Лікування

Операції

Пристрої

Іван Приймаченко

Вийти

Винесені та рекомендовані діагнози пацієнтів

Рекомендовані діагнози створюються програмним забезпеченням як порада лікарю. Лікар відповідальний за винесене рішення.

☒ додати рекомендовані системою рішення

Підтвердити

Новий діагноз

Зведені діагнози

10 на сторінку

Шукати пацієнта:

Пацієнт	Діагноз винесений	Діагноз рекомендований		
Андрій Іванов	Не встановлено	Серцева недостатність	Переглянути	Підтвердити
Віктор Шиманський	Серцева недостатність першого ступеня	Серцева недостатність відсутня (перевірити папери)	Переглянути	Підтвердити
Ганна Верко	Серцева недостатність третього ступеня	Серцева недостатність	Переглянути	Підтвердити
Олег Зінов'єв	Серцева недостатність другого ступеня	Серцева недостатність	Переглянути	Підтвердити
Євген Мерченко	Серцева недостатність третього ступеня	Серцева недостатність	Переглянути	Підтвердити

з 1 по 5 запис

Назад 1 Вперед

Рисунок В.12 – Слайд 12

Експериментальне дослідження

Модель або метод	Точність діагностування
Метод k-найближчих сусідів	0.67
Метод опорних векторів	0.81
Дерева рішень	0.82
Багатошарова нейронна мережа	0.8
Багатошарова нейронна мережа зі зменшеним простором ознак	0.88
Метод діагностування серцевої недостатності	0.91

Рисунок В.13 – Слайд 13

Висновки

- Програма призначена для підтримки роботи лікарів, зокрема кардіологів з врахуванням особливостей їх роботи, забезпечується діагностування стану пацієнтів щодо наявності серцевої недостатності, адміністраторів з анаданням засобів створення розкладу лікарів, пацієнтів, які можуть переглядати власні медичні дані, вносити власні скарги, лаборантів, що представляють лабораторії з дослідження аналізів пацієнтів.
- Наукова новизна роботи полягає в запропонованому методі розв'язання проблеми діагностування серцевої недостатності, який характеризується виділенням двох моделей класифікації на основі структурованих та текстових даних медичних карток пацієнтів, що дозволяє виносити рішення за відсутності одного з типів даних або виносити узгоджене рішення, сприяючи підвищенню обґрунтованості діагностичних рішень.
- Усі поставлені завдання на роботу виконано. 14

Рисунок В.14 – Слайд 14